Modifying the Assignment & Configuration File with the setacf Utility

Altera provides the **setacf** utility to help you modify a project's Assignment & Configuration File (**.acf**) from the command line, without opening the file with a text editor. Type `setacf -h` ↵ at a UNIX or DOS prompt to get help on this utility.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Compiling the alt_mf Library

If your VHDL design uses functions from the **alt_mf** library, you must compile this library. To compile the **alt_mf** library, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS® II/Cadence Working Environment. For example, you must ensure that the appropriate directories are specified in the **cds.lib** file located in your working directory.
2. Change to the **alt_mf** directory by typing `cd /usr/maxplus2/simlib/concept/alt_mf` ↵ at the UNIX prompt.
3. Edit the **hdl.var** file located in your working directory to include the following line:

   ```
   DEFINE work alt_mf ↵
   ```

4. Type the following commands at the UNIX prompt from the **/usr/maxplus2/simlib/concept/alt_mf** directory to compile the library:

   ```
   cv -message -file ./src/mf.vhd ↵
   cv -message -file ./src/mf_components.vhd ↵
   ```

<<<<<<< alt_mf.htm

Technical Feedback

=======

Feedback

>>>>>>> 1.7

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Compiling the VITAL Library for Use with Leapfrog Software

If you wish to use MAX+PLUS® II-generated Standard Delay Format (SDF) Output Files (**.sdo**) that contain timing information when performing post-compilation timing simulation with Leapfrog software, you must first compile the VITAL library source files. The VITAL Timing and Primitive package files are located in the **$CDS_INST_DIR/tools/leapfrog/files/IEEE.src** directory.

To compile the **alt_vtl** library, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment. For example, you must ensure that the appropriate directories are specified in the **cds.lib** file that is located in your working directory.
2. Create a VHDL design, as described in Creating VHDL Designs for Use with MAX+PLUS II Software and save it in your working directory.
3. Change to the **alt_vtl** directory by typing `cd /usr/maxplus2/simlib/concept/alt_vtl` ↵ at the UNIX prompt.
4. Edit the **hdl.var** file located in your working directory to include the following line:

   ```
   DEFINE WORK alt_vtl ↵
   ```

5. Create the **/usr/maxplus2/simlib/concept/alt_vtl/lib** directory.
6. Type the following commands at the UNIX prompt from the **/usr/maxplus2/simlib/concept/alt_vtl** directory to compile the library:

   ```
   cv -message -file alt_vtl.vhd ↵
   cv -message -file alt_vtl.cmp ↵
   ```

<<<<<<< alt_vtl.htm

Technical Feedback

=======

Feedback

>>>>>>> 1.7

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

BackAnnotating MAX+PLUS II Pin Assignments to Design Architect Symbols

The MAX+PLUS ® II/Mentor Graphics software interface includes the **annotate_pin** utility. This utility allows you to back-annotate the pin assignments from the MAX+PLUS II-generated Fit File (**.fit**) back to the symbol for the design file. The **annotate_pin** utility has the following syntax:

`annotate_pin [-p` <*property name*>`]` <*symbol name*> <*chip name*> <*Fit File name*> ↵

where <*property name*> is the default name for the pin assignment (default is `PIN_NO`), <*symbol name*> is the pathname of the directory that contains the symbol, <*chip name*> is the chip name specified in the Fit File, and <*Fit File name*> is the name of the Fit File that contains the pin assignment information for back-annotation. If the <*property name*> is not found at a pin number, that pin will not be back-annotated. If the <*chip name*> is not found in the Fit File, the **annotate_pin** utility stops the back-annotation process.

For example:

```
annotate_pin -p PIN_NO /usr/examples/decode decode decode.fit ↵
```

☞ Type `annotate_pin -h` ↵ at the UNIX prompt to display information on how to use this utility.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Using Mentor Graphics Design Architect & MAX+PLUS II Software

The following topics describe how to use the Mentor Graphics Design Architect software with MAX+PLUS® II software. Choose one of the following topics for information:

Open a printable version of all topics listed on this page.

Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment

- Software Requirements
- Altera-Provided Logic & Symbol Libraries
- Local Work Area Directory Structure
- Mentor Graphics Project Directory Structure
- MAX+PLUS II Project Directory Structure
- MAX+PLUS II/Mentor Graphics/Exemplar Logic Interface File Organization

Design Entry

- Design Entry Flow
- Creating Design Architect Schematics for Use with MAX+PLUS II Software
  - Instantiating the clklock Megafunction in Design Architect Schematics
  - Instantiating LPM Functions in Design Architect Schematics
- Entering Resource Assignments
  - Assigning Pins, Logic Cells & Chips
  - Assigning Cliques
  - Assigning Logic Options
  - Modifying the Assignment & Configuration File with the **setacf** Utility
- Creating Hierarchical Projects with Design Architect Software
- Converting Design Architect Schematics into MAX+PLUS II-Compatible EDIF Netlist Files with the ENWrite Utility

Simulation

- Performing a Functional Simulation with DVE & QuickSim II Software
- Performing a Functional Simulation with QuickHDL Pro Software

Related Links

- Compiling Projects with MAX+PLUS II Software
- Programming Altera Devices
- MAX+PLUS II Development Software
- Altera Programming Hardware
- Mentor Graphics web site (http://www.mentor.com)

Using Mentor Graphics Design Architect & MAX+PLUS II Software

The following topics describe how to use the Mentor Graphics Design Architect software with MAX+PLUS® II software. Click on one of the following topics for information:

This file is suitable for printing only. It does not contain hypertext links that allow you to jump from topic to topic.

Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment

- Software Requirements
- Altera-Provided Logic & Symbol Libraries
- Local Work Area Directory Structure
- Mentor Graphics Project Directory Structure
- MAX+PLUS II Project Directory Structure
- MAX+PLUS II/Mentor Graphics/Exemplar Logic Interface File Organization

Design Entry

- Design Entry Flow
- Creating Design Architect Schematics for Use with MAX+PLUS II Software
  - Instantiating the clklock Megafunction in Design Architect Schematics
  - Instantiating LPM Functions in Design Architect Schematics
- Entering Resource Assignments
  - Assigning Pins, Logic Cells & Chips
  - Assigning Cliques
  - Assigning Logic Options
  - Modifying the Assignment & Configuration File with the **setacf** Utility
- Creating Hierarchical Projects with Design Architect Software
- Converting Design Architect Schematics into MAX+PLUS II-Compatible EDIF Netlist Files with the ENWrite Utility

Simulation

- Performing a Functional Simulation with DVE & QuickSim II Software
- Performing a Functional Simulation with QuickHDL Pro Software

Related Topics:

- Go to the following MAX+PLUS II ACCESS℠ Key topics for related information.
  - Compiling Projects with MAX+PLUS II Software
  - Programming Altera Devices

- Go to the following topics, which are available on the web, for additional information:

- MAX+PLUS II Development Software
- Altera Programming Hardware
- Mentor Graphics web site (http://www.mentor.com)

Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment

To use the MAX+PLUS ® II software with Mentor Graphics/Exemplar Logic software, you must install the MAX+PLUS II software, then establish an environment that facilitates entering and processing designs. The MAX+PLUS II/Mentor Graphics/Exemplar Logic interface is installed automatically when you install the MAX+PLUS II software on your computer.

Go to *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual for more information on installation and details on the directories that are created during MAX+PLUS II installation. Go to MAX+PLUS II/Mentor Graphics/Exemplar Logic Interface File Organization for information about the MAX+PLUS II/Mentor Graphics directories that are created during MAX+PLUS II installation.

☞ The information presented here assumes that you are using a C shell and that your MAX+PLUS II system directory is **/usr/maxplus2**. If not, you must use the appropriate syntax and procedures to set environment variables for your shell.

To set up your working environment for the MAX+PLUS II/Mentor Graphics interface, follow these steps:

1. Ensure that you have correctly installed the MAX+PLUS II and Mentor Graphics software versions described in MAX+PLUS II/Mentor Graphics Software Requirements.

2. Add the following environment variables to your **.cshrc** file:

```
setenv ALT_HOME /usr/maxplus2 ↵
setenv MGC_WD <user-specified working directory> ↵
setenv MGC_HOME <Mentor Graphics system directory> ↵
setenv MAX2_MENTOR /usr/maxplus2/mentor/max2 ↵
setenv MGC_LOCATION_MAP <user-specified location_map file> ↵
setenv EXEMPLAR <Galileo or Leonardo system directory> ↵
```

   ☞ Installing the Altera® provided MAX+PLUS II/Mentor Graphics interface on your computer automatically installs a template for these environment variables in the **/usr/maxplus2/mentor/max2/.cshrc** file.

3. Add the **$MGC_HOME/bin**, **$MAX2_MENTOR/bin**, **$ALT_HOME/bin**, **$EXEMPLAR/bin/**<*os*>, and **$ALT_HOME/bin** directories to the `PATH` environment variable in your **.cshrc** file, where <*os*> is the operating system, e.g., `SUN4` for SunOS; `SUN5` for Solaris.

4. If you plan to use the Altera Schematic Express (**sch_exprss**) utility or the Altera VHDL Express (**vhd_exprss**) utility, add the following environment variable to your **.cshrc** file:

```
setenv MAX2_QSIM /usr/maxplus2/simlib/mentor/max2sim ↵
```

5. Type `source ~/.cshrc` at a UNIX prompt to source the **.cshrc** file and validate the settings in steps 1 through 4.

6. Add the following lines to your **MGC_location_map** file:

```
$MAX2_MENTOR ↵
/usr/maxplus2/mentor/max2 ↵
```

```
$MGC_GENLIB ↵
/<user-specified Mentor Graphics GEN_LIB directory> ↵
$MGC_LSLIB ↵
/<user-specified Mentor Graphics LS_LIB directory> ↵
$MAX2_EXAMPLES ↵
/<user-specified example directory> ↵
$MAX2_LMCLIB ↵
/<user-specified Logic Modeling directory> ↵
$MAX2_GENLIB ↵
/usr/maxplus2/simlib/mentor/alt_max2 ↵
$MAX2_QSIM ↵
/usr/maxplus2/simlib/mentor/max2sim ↵
$MAX2_FONT ↵
/usr/maxplus2/mentor/max2/fonts ↵
$MGC_SYS1076_STD ↵
/<user-specified MGC_HOME directory>/pkgs/sys_1076_std/ std ↵
$MGC_SYS1076_ARITHMETIC ↵
/<user-specified MGC_HOME directory>/pkgs/sys_1076_std/arithmetic ↵
$MGC_SYS1076_PORTABLE ↵
/<user-specified MGC_HOME directory>/pkgs/sys_1076_std/mgc_portable ↵
$MGC_SYS1076_IEEE ↵
/<user-specified MGC_HOME directory>/pkgs/sys_1076_std/ieee ↵
$MGC_SYS1076_SRC ↵
/<user-specified MGC_HOME directory>/pkgs/sys_1076_std/ src ↵
$MAX2_MFLIB ↵
/usr/maxplus2/simlib/mentor/alt_mf ↵
```

Installing the Alteraprovided MAX+PLUS II/Mentor Graphics/Exemplar Logic interface on your computer automatically installs a template for these environment variables in the **/usr/maxplus2/mentor/max2/location_map/location_map** file.

7. If you want to use QuickHDL software to simulate VHDL or Verilog HDL designs, add the following line in the `[library]` section of your **quickhdl.ini** file: `altera = $MAX2_MFLIB`.

8. If you plan to use QuickHDL software to simulate VITAL-compliant VHDL files, add the following lines to your **MGC_location_map** file:

```
$MAX2_VTLLIB ↵
/usr/maxplus2/simlib/mentor/alt_vtl ↵
```

9. Copy the **/usr/maxplus2/maxplus2.ini** file to your **$HOME** directory:

```
cp /usr/maxplus2/maxplus2.ini $HOME ↵
chmod u+w $HOME/maxplus2.ini ↵
```

The **maxplus2.ini** file contains both Altera- and user-specified initialization parameters that control the MAX+PLUS II software, such as Alteraprovided logic and symbol library paths and the current project name. The MAX+PLUS II installation procedure creates and copies the **maxplus2.ini** file to

the **/usr/maxplus2** directory.

Normally, you do not have to edit your local copy of **maxplus2.ini**, because the MAX+PLUS II software updates the file automatically whenever you change any parameters or settings. However, if you move the **max2lib** and **max2inc** library subdirectories, you must update the file. Go to "Creating & Using a Local Copy of the **maxplus2.ini** File" in MAX+PLUS II Help for more information.

Related Topics:

- Go to the following topics, which are available on the web, for additional information:
  - ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
  - This manual is also available in 4 parts:
    - Preface & Section 1: MAX+PLUS II Installation
    - Section 2: MAX+PLUS II - A Perspective
    - Section 3: MAX+PLUS II Tutorial
    - Appendices, Glossary & Index

MAX+PLUS II/Mentor Graphics Software Requirements

The following products are used to generate, process, synthesize, and verify a project with the MAX+PLUS ® II software and Mentor Graphics software:

Mentor Graphics		Exemplar	Altera
version C.1:			
System_1076 Compiler QuickSim II Design Architect ENRead ENWrite GEN_LIB library	QuickHDL QuickHDL Pro QuickPath LS_LIB library (optional) DVE	Galileo Extreme version 4.1.1 Leonardo version 4.1.3	MAX+PLUS II version 9.4

The MAX+PLUS II **read.me** file provides up-to-date information on which versions of Mentor Graphics applications are supported by the current version of MAX+PLUS II. It also provides information on installation and operating requirements. You should read the **read.me** file on the CD-ROM before installing the MAX+PLUS II software. After installation, you can open the **read.me** file from the MAX+PLUS II Help menu.

Altera-Provided Logic & Symbol Libraries

The MAX+PLUS ® II/Mentor Graphics environment provides libraries for compiling, synthesizing, and simulating designs.

You can create your own libraries of custom functions for use in Design Architect schematics and VHDL and Verilog HDL design files. You can use custom functions to incorporate an EDIF Input File (**.edf**), Text Design File (**.tdf**), or any other MAX+PLUS II-supported design file into a project. The MAX+PLUS II software uses the Altera® provided **mnt8_bas.lmf** and **exemplar.lmf** Library Mapping Files to map standard Design Architect symbols and VHDL and Verilog HDL functions to equivalent MAX+PLUS II logic functions. To use custom functions, you can create a custom LMF that maps your custom functions to the equivalent EDIF input file, TDF, or other design file. Go to "Library Mapping File" in MAX+PLUS II Help for more information.

Design Architect Libraries

You can enter a Design Architect schematic with logic functions from these Altera-provided symbol libraries: **ALTERA LPMLIB**, **ALTERA GENLIB**, **LSTTL BY TYPE**, and **LSTTL ALL PARTS**. You can access these libraries by choosing **Altera Libraries** (Libraries menu) in the Design Architect software. For information on using library of parameterized modules (LPM) functions, see **ALTERA LPMLIB** Library below.

ALTERA GENLIB Library (Design Architect) & Altera (VHDL) Libraries

The **ALTERA GENLIB** symbol library (called the **Altera** library for VHDL) includes several MAX+PLUS II primitives for controlling design synthesis and fitting. It also includes four macrofunctions (`8count`, `8mcomp`, `8fadd`, and `81mux`) that are optimized for different Altera device families, and the `clklock` phase-locked loop megafunction, which is supported for some FLEX ® 10K devices.

The following table shows the MAX+PLUS II-specific logic functions.

Table 1. MAX+PLUS II-Specific Logic Functions

Macrofunctions *Note (1)*				**Primitives**	
Name	**Description**	**Name**	**Description**	**Name**	**Description**
`8fadd`	8-bit full adder	`LCELL`	Logic cell buffer	`EXP`	MAX ® 5000, MAX 7000 , and MAX 9000 Expander buffer
`8mcomp`	8-bit magnitude comparator	`GLOBAL`	Global input buffer	`SOFT`	Soft buffer
`8count`	8-bit up/down counter	`CASCADE`	FLEX 6000, FLEX 8000, and FLEX 10K cascade buffer	`OPNDRN`	Open-drain buffer
`81mux`	8-to-1 multiplexer	`CARRY`	FLEX 6000, FLEX 8000, and FLEX 10K carry buffer	`DFFE` `DFFE6K` *Note (2)*	D-type flipflop with Clock Enable
`clklock`	Phase-locked loop				

Notes:

1. Logic function names that begin with a number must be preceded by "`a_`" in VHDL designs. For example, `8fadd` must be specified as `a_8fadd` instead.
2. If you want to use QuickHDL software, make sure you have updated your **quickhdl.ini** file, as described in step 7 of Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment.
3. For designs that are targeted for FLEX 6000 devices, you should use the `DFFE` primitive only if the design contains either a Clear or Preset signal, but not both. If your design contains both a Clear and a Preset signal, you must use the `DFFE6K` primitive.

☞ Choose **Old-Style Macrofunctions**, **Primitives**, or **Megafunctions/LPM** from the MAX+PLUS II Help menu for detailed information on these functions.

ALTERA LPMLIB Library

The Alteraprovided **ALTERA LPMLIB** library, which is available for Design Architect schematics and VHDL designs, includes standard functions from the library of parameterized modules (LPM) 2.1.0, except the truth table, finite state machine, and pad functions. The LPM standard defines a set of parameterized modules (i.e., parameterized functions) and their corresponding representations in an EDIF netlist file. These logic functions allow you to create and functionally simulate an LPM-based design without targeting a specific device family. After the design is completed, you can target the design to any device family. The parameters you specify for each LPM function determine which simulation models are generated.

☞

Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu for more information about LPM functions.

Related Topics:

- Go to the following topics, which are available on the web, for additional information:
  - FLEX Devices
  - MAX Devices
  - Classic Device Family

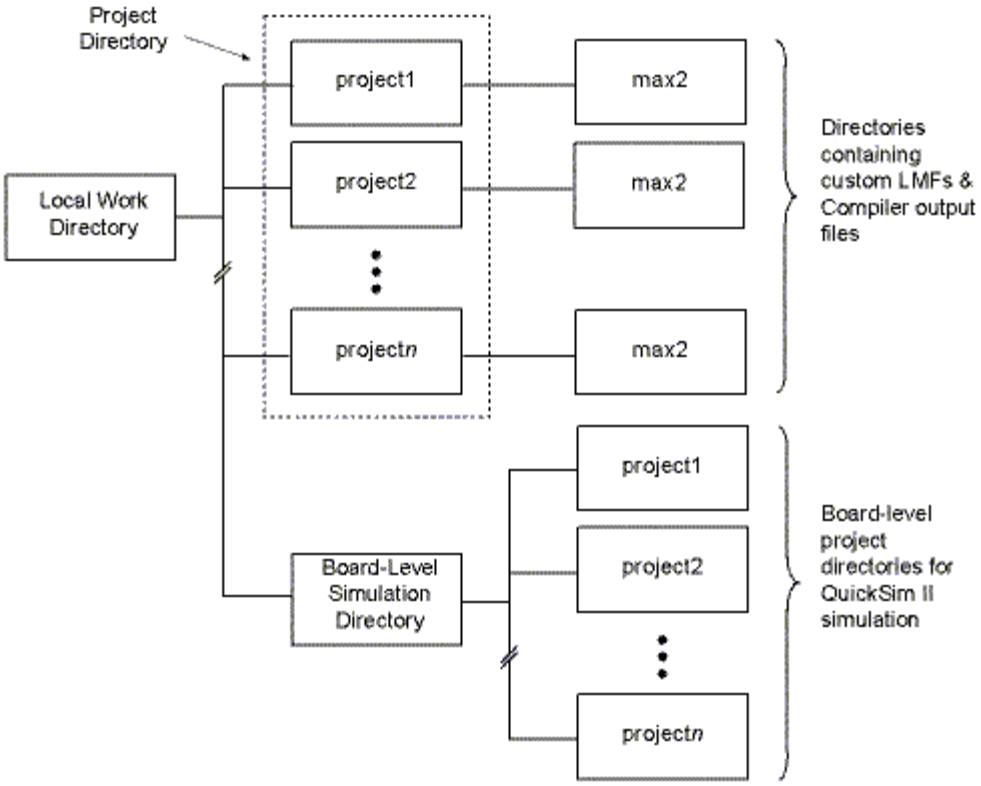
Local Work Area Directory Structure

Design Architect software automatically creates and maintains the project directory structure required for all stages of design entry. Galileo Extreme, Leonardo, and ENWrite software create a **max2** subdirectory, if it does not already exist, under the project directory. These software applications also generate EDIF netlist files, and copy them from the <*project name*> directory to this **max2** subdirectory. All MAX+PLUS ® II Compiler output files are created in the **max2** subdirectory.

Simulation files created with Mentor Graphics applications and Logic Modeling files are located in the board-level simulation subdirectory of the project directory. You can use these files during simulation with QuickSim II software.

The only directory you need to create is the local work directory, which should contain all project directories. Figure 1 shows the recommended file structure.

Figure 1. Recommended File Structure



Related Topics:

- Go to the following MAX+PLUS II ACCESS^{SM} Key topics for related information:

- MAX+PLUS II Project Directory Structure
- Mentor Graphics Project Directory Structure

Mentor Graphics Project Directory Structure

Design Architect software generates the following files for each schematic:

- <*drawing name*>**/mgc_component.attr**
- <*drawing name*>**/part.Eddm_part.attr**
- <*drawing name*>**/part.part_1**
- <*drawing name*>**/schematic.mgc_schematic.attr**
- <*drawing name*>**/schematic/schem_id**
- <*drawing name*>**/schematic/sheet1.mgc_sheet.attr**
- <*drawing name*>**/schematic/sheet1.sgfx_1**
- <*drawing name*>**/schematic/sheet1.ssht_1**

The files generated for each schematic are stored in the schematic's <*drawing name*> directory and should not be edited. Mentor Graphics software automatically manages file storage and retrieval operations through this <*drawing name*> directory structure, which does not reflect hierarchical design relationships. Figure 1 shows a sample file structure with **project1** as the UNIX project directory, and **design1**, **subdesign1**, and **subdesign2** as the directories for the top-level design and subdesigns of the project.

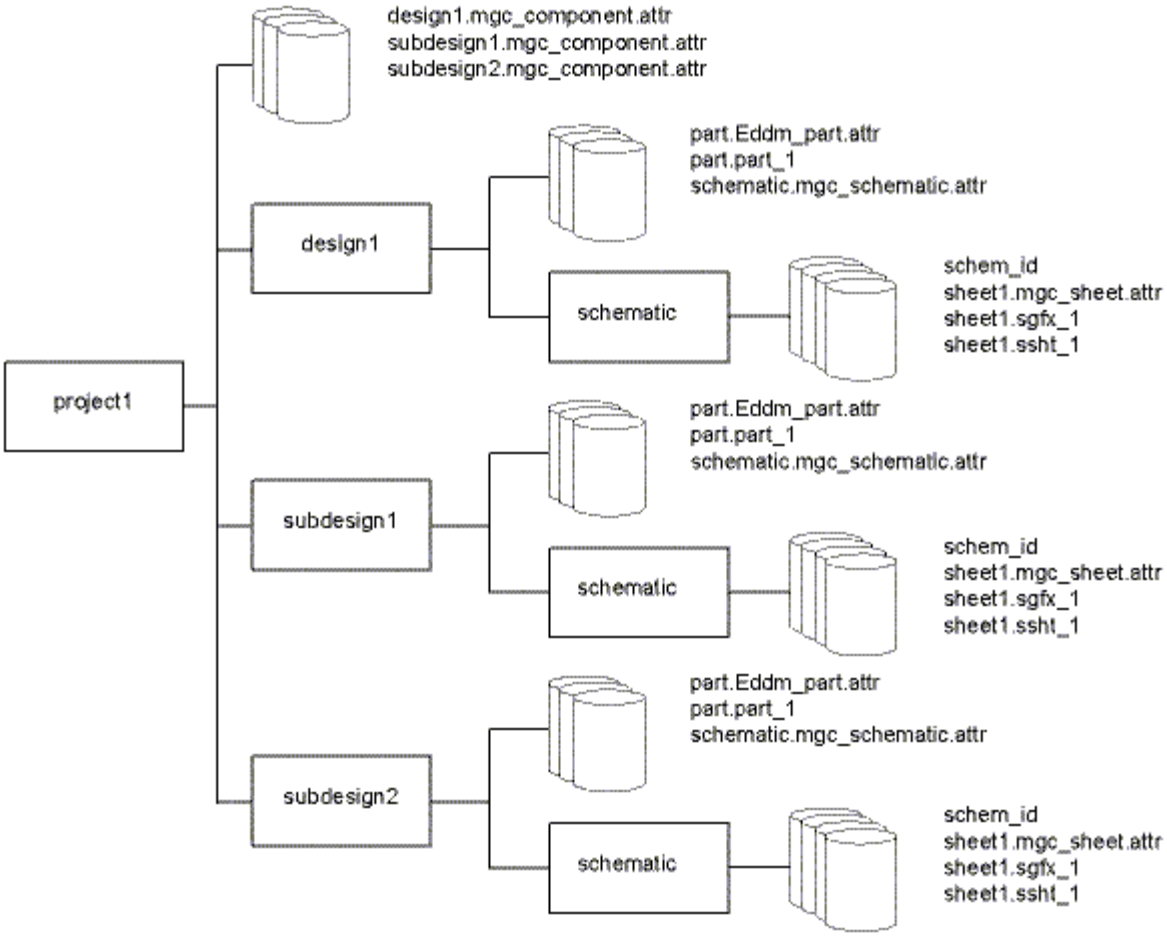


Figure 1. Design Architect Project File Structure

When the ENWrite utility converts the schematic into an EDIF netlist file, it processes the design information and all related file subdirectories, then creates the EDIF netlist file in the directory defined by the user. The EDIF netlist file is named <*project name*>**.edf**, where <*project name*> is the name of the top-level design file. The <*project name*>**.edf** file is automatically moved to the **max2** directory under the project directory.

Related Topics:

- Go to the following MAX+PLUS II ACCESS℠ Key topics for related information:
  - Local Work Area Directory Structure
  - MAX+PLUS II Project Directory Structure

MAX+PLUS II Project Directory Structure

In the MAX+PLUS ® II software, a project name is the name of a top-level design file, without the filename extension. This design file can be an EDIF, VHDL, or Verilog HDL netlist file; an Altera Hardware Description Language (AHDL) Text Design File (TDF); or any other MAX+PLUS II-supported design file. The EDIF netlist file must be created by ENWrite, Galileo Extreme, or Leonardo software and imported into MAX+PLUS II as an EDIF Input File (**.edf**). Figure 1 shows an example of a MAX+PLUS II project directory.

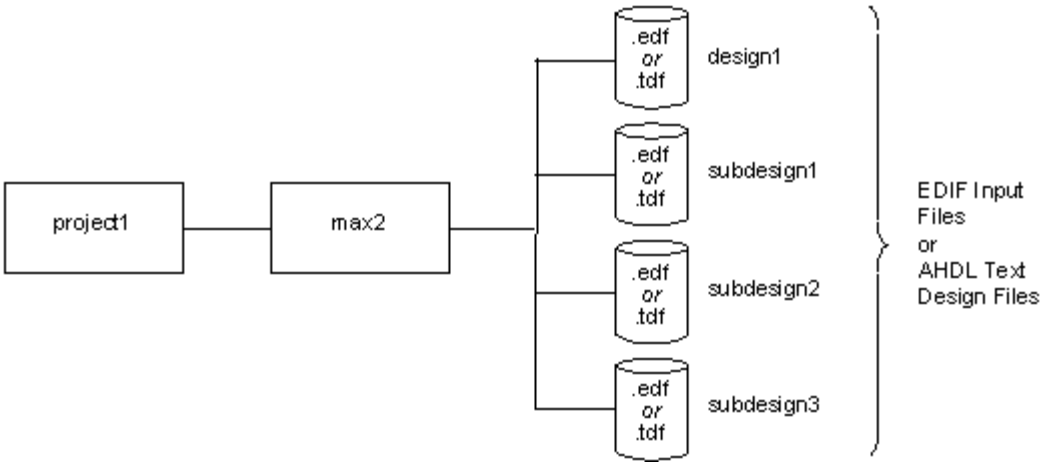


Figure 1. Sample MAX+PLUS II Project Directory

The MAX+PLUS II software stores the connectivity data on the links between design files in a hierarchical project in a Hierarchy Interconnect File (**.hif**), but refers to the entire project only by its project name. The MAX+PLUS II Compiler uses the HIF to build a single, fully flattened project database that integrates all the design files in a project hierarchy.

Related Topics:

- Go to the following MAX+PLUS II ACCESS℠ Key topics for related information:
  - Local Work Area Directory Structure
  - Mentor Graphics Project Directory Structure

MAX+PLUS II/Mentor Graphics/Exemplar Logic Interface File Organization

The following table shows the MAX+PLUS ® II/Mentor Graphics interface subdirectories that are created in the MAX+PLUS II system directory (by default, the **/usr/maxplus2** directory) during MAX+PLUS II installation.

☞ For information on the other directories that are created during MAX+PLUS II installation, see "MAX+PLUS II File Organization" in *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual.

Table 1. MAX+PLUS II Directory Organization

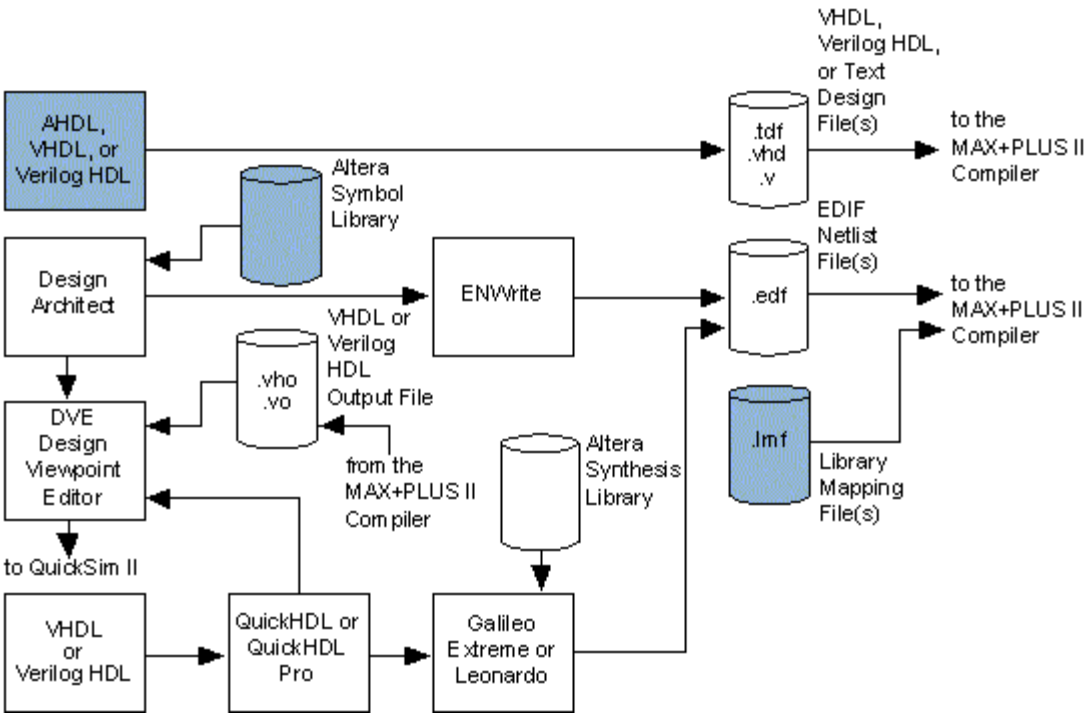
Directory	Description
.lmf	Contains the Altera-provided Library Mapping Files, **mnt8_bas.lmf** and **exemplar.lmf**, that map Mentor Graphics and Exemplar Logic logic functions to equivalent MAX+PLUS II logic functions.
./mentor	Contains the AMPLE userware for the MAX+PLUS II/Mentor Graphics interface.
./simlib/mentor/alt_max2	Contains MAX+PLUS II primitives such as `CARRY`, `CASCADE`, `EXP`, `GLOBAL`, `LCELL`, `SOFT`, `OPNDRN`, `DFFE`, and `DFFE6K` (D flipflop with Clock Enable) for use in Design Architect schematics.
./simlib/mentor/max2sim	Contains the MAX+PLUS II/Mentor Graphics simulation model library, **max2sim**, for use with QuickSim II and QuickPath software.
./simlib/mentor/synlib	Contains the MAX+PLUS II synthesis library for use with AutoLogic II software, which supports synthesis for users running Mentor Graphics version B1.
./simlib/mentor/alt_mf	Contains the MAX+PLUS II macrofunction and megafunction libraries.
./simlib/mentor/alt_vtl	Contains the MAX+PLUS II VITAL library.

Mentor Graphics/Exemplar Logic Design Entry Flow

The following figure shows the design entry flow for the MAX+PLUS® II/Mentor Graphics/Exemplar Logic interface.

Figure 1. MAX+PLUS II/Mentor Graphics/Exemplar Logic Design Entry Flow

Alteraprovided items are shown in blue.



Creating Design Architect Schematics for Use with MAX+PLUS II Software

You can create Design Architect schematics and convert them into EDIF Input Files (**.edf**) that can be processed with the MAX+PLUS ® II Compiler.

To create a Design Architect schematic for use with MAX+PLUS II software, go through the following steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment.

2. Start the MAX+PLUS II/Mentor Graphics interface by typing `max2_dmgr` ↵ at a UNIX prompt.

3. Start the Design Architect software by double-clicking Button 1 on the **max2_da** icon in the Design Manager tools window. You can also start Design Architect software by typing `max2_da` ↵ at the UNIX prompt.

4. Use the graphical user interface to structure and organize your files to create an environment that facilitates entering and processing designs. Go to the following topics for more information:

    - Local Work Area Directory Structure
    - MAX+PLUS II Project Directory Structure
    - Mentor Graphics Project Directory Structure

5. Choose the **OPEN SHEET** button in the Design Architect session_palette, then specify a name for your project in the *Component Name* box. Choose **OK**.

6. Enter logic functions from the following Altera® provided libraries:

    - **ALTERA LPMLIB** includes library of parameterized modules (LPM) functions
    - **ALTERA GENLIB** includes primitives and macrofunctions
    - **LSTTL** includes 74-series macrofunctions

    You can instantiate MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP^{SM}). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.

    The following topics describe special steps needed to instantiate LPM and `clklock` functions:

    - Instantiating LPM Functions in Design Architect Schematics
    - Instantiating the clklock Megafunction in Design Architect Schematics

7. (Optional) To create a hierarchical design that contains symbols representing other design files, such as AHDL or VHDL design files, go to Creating Hierarchical Projects with Design Architect Software.

8. If you wish to make resource assignments in a Design Architect schematic, go to Entering Resource Assignments. You can also enter resource assignments from within the MAX+PLUS II software.

9. Choose **Check Sheet for Altera** (Check menu) to save and check your design. If your design contains LPM functions , the Design Architect software will ask whether you want to compile the LPM model. Choose **YES** if you want to compile the VHDL code for the LPM functions. The software will automatically select the corresponding compiler: System 1076 for B.(*x*) releases and QuickHDL compilers for releases C.1 and later.

10. (Optional) If your schematic design includes models for VHDL or Verilog HDL designs, perform a

functional simulation with the QuickHDL Pro software, as described in Performing a Functional Simulation with QuickHDL Pro Software. If it does not, you can perform a functional simulation with the QuickSim software, as described in Performing a Functional Simulation with DVE & QuickSim II Software.

11. Once you have created a schematic, you can generate an EDIF netlist file that can be imported into the MAX+PLUS II software with either of the following methods:

    - You can create an EDIF netlist file, as described in Converting Design Architect Schematics into MAX+PLUS II-Compatible EDIF Netlist Files with the ENWrite Utility.

    - You can use the Altera Schematic Express utility, **sch_exprss**, to automatically create an EDIF netlist file, compile it with the MAX+PLUS II Compiler, generate an EDIF Output File (**.edo**), and prepare the EDIF Output File for simulation with ENRead and Design Viewpoint Editor (DVE), as described in Using the Altera Schematic Express (**sch_exprss**) Utility.

    Even if your design is a hierarchical design incorporating files created with multiple design entry methods, both the ENWrite and Altera Schematic Express utilities generate EDIF files for all files in the design.

    Installing the Alteraprovided MAX+PLUS II/Mentor Graphics/Exemplar Logic interface on your computer automatically creates the following sample Design Architect schematic files:

- **/usr/maxplus2/examples/mentor/example1/fulladd**
- **/usr/maxplus2/examples/mentor/example3/fulladd2**
- **/usr/maxplus2/examples/mentor/example7/fifo**

Related Topics:

- Go to Compiling Projects with MAX+PLUS II Software in these MAX+PLUS II ACCESS^{SM} Key topics for related information.

Instantiating the clklock Megafunction in Design Architect Schematics

You can instantiate the Altera® provided `clklock` phase-locked loop megafunction, which is supported for some FLEX ® 10K devices, in a Design Architect schematic.

To instantiate the `clklock` megafunction in a Design Architect schematic, follow these steps:

1. Choose **Altera Libraries** (Library menu).
2. Choose **ALTERA GENLIB** (Altera Libraries menu).
3. Choose **clklock** (ALTERA GENLIB menu).
4. Specify appropriate values for the `CLOCKBOOST` and `INPUT_FREQUENCY` variables. Choose **Megafunctions/LPM** from the MAX+PLUS ® II Help menu for detailed information on the `clklock` megafunction.
5. Choose **OK**.
6. Continue with the steps necessary to complete your Design Architect schematic, as described in Creating Design Architect Schematics for Use with MAX+PLUS II Software.

Installing the Alteraprovided MAX+PLUS II/Mentor Graphics/Exemplar Logic interface on your computer automatically creates the sample Design Architect schematic file **/usr/maxplus2/examples/mentor/example7/fifo**, which includes `clklock` megafunction instantiation.

Related Topics:

- Go to FLEX 10K Device Family, which is available on the web, for additional information.

Instantiating LPM Functions in Design Architect Schematics

Design Architect software allows you to instantiate functions included in the library of parameterized modules (LPM) from the **ALTERA LPMLIB** library.

Go through the following steps to instantiate LPM functions in a Design Architect schematic:

1. While you are entering your Design Architect schematic, choose **Altera Libraries** (Library menu).
2. Choose **ALTERA LPMLIB** (Altera Libraries menu).
3. Choose from the available LPM functions on the ALTERA GENLIB menu.
4. In the **LPM_**<*function name*> dialog box, specify appropriate values for the variables displayed for the LPM function you chose in step 3. Make sure that any hexadecimal (Intel-format) file that you use to specify the initial content of a memory function does not have the same name as the design file name. Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu for detailed information on LPM functions.
5. Choose **OK** to generate a symbol for the LPM function you chose in step 3 and a corresponding VHDL simulation model.
6. Continue with the steps necessary to complete your Design Architect schematic, as described in Creating Design Architect Schematics for Use with MAX+PLUS II Software.
7. When you save the schematic, the Design Architect software will ask whether you want to compile the LPM model. Choose **YES** if you want to compile the VHDL code for the LPM functions. The software will automatically select the corresponding compiler: System 1076 for B.(*x*) releases and QuickHDL compilers for releases C.1 and later.

Installing the Altera® provided MAX+PLUS II/Mentor Graphics/Exemplar Logic interface on your computer automatically creates the sample Design Architect schematic file **/usr/maxplus2/examples/mentor/example7/fifo**, which includes LPM instantiation.

Entering Resource Assignments

The MAX+PLUS ® II software allows you to enter a variety of resource and device assignments for your projects. Resource assignments are used to assign logic functions to a particular pin, logic cell, I/O cell, embedded cell, row, column, Logic Array Block (LAB), Embedded Array Block (EAB), chip, clique, local routing, logic option, timing requirement, or connected pin group. In MAX+PLUS II software, you can enter all types of resource and device assignments with Assign menu commands. You can also enter pin, logic cell, I/O cell, embedded cell, LAB, EAB, row, and column assignments in the MAX+PLUS II Floorplan Editor. The Assign menu commands and the Floorplan Editor all save assignment information in the ASCII Assignment & Configuration File (**.acf**) for the project. In addition, you can edit ACFs manually in any standard text editor or with the **setacf** utility.

Design Architect Schematics

In Design Architect schematics, you can assign a limited subset of these resource assignments by assigning properties to symbols. These properties are incorporated into the EDIF netlist file(s). The MAX+PLUS II software

automatically converts assignment information from the EDIF Input File into the ACF format. For information on making MAX+PLUS II-compatible resource assignments, go to the following topics:

- Assigning Pins, Logic Cells & Chips
- Assigning Cliques
- Assigning Logic Options
- Modifying the Assignment & Configuration File with the **setacf** Utility

☞ After you compile a project, you can back-annotate pin assignments, as described in BackAnnotating MAX+PLUS II Pin Assignments to Design Architect Symbols.

Installing the Alteraprovided MAX+PLUS II/Mentor Graphics/Exemplar Logic interface on your computer automatically creates the sample Design Architect schematic file **/usr/maxplus2/examples/mentor/example4/fa2**, which includes resource assignments.

VHDL & Verilog HDL Design Files

For Verilog HDL- and VHDL-based designs, you must use the MAX+PLUS II software or the **setacf** utility to enter resource assignments. Go to Modifying the Assignment & Configuration File with the **setacf** Utility for more information.

Related Topics:

- Go to "Resource Assignments in EDIF Input Files" and "Assigning Resources in a Third-Party Design Editor" in MAX+PLUS II Help for more information on assignments or properties that can be assigned in Design Architect software. For information on entering assignments in MAX+PLUS II software with Assign menu commands or in an ACF, go to "resource assignments" or "ACF, format" in MAX+PLUS II Help using **Search for Help on** (Help menu).

Assigning Pins, Logic Cells & Chips

You can assign a single logic function to a specific pin or logic cell (including I/O cells and embedded cells) within a chip, and assign one or more functions to a specific chip. A chip is a group of logic functions defined as a single, named unit, which can be assigned to a specific device.

You can assign a signal to a particular pin to ensure that the signal is always associated with that pin, regardless of future changes to the project. If you wish to set and maintain the performance of your project, assigning logic to a specific logic cell within a chip can minimize timing delays. In a project that is partitioned among multiple devices, you can assign logic functions that must be kept together in the same device to a chip. Chip assignments allow you to split a project so that only a minimum number of signals travel between devices, and to ensure that no unnecessary device-to-device delays exist on critical timing paths. You can assign a chip to a device in some EDA tools or in the MAX+PLUS® II software.

Use the following syntax for chip, pin, and logic cell assignments:

- To assign a logic function to a chip:

  `CHIP_PIN_LC=`<*chip name*>

  For example: `CHIP_PIN_LC=chip1`

- To assign a pin number within a chip:

  `CHIP_PIN_LC=`<*chip name*>`@`<*pin number*>

For example: `CHIP_PIN_LC=chip1@K2`

- To assign a logic cell, I/O cell, or embedded cell number:

  `CHIP_PIN_LC=`<*chip name*>`@LC`<*logic cell number*>

  `CHIP_PIN_LC=`<*chip name*>`@IOC`<*I/O cell number*>

  `CHIP_PIN_LC=`<*chip name*>`@EC`<*embedded cell number*>

  For example: `CHIP_PIN_LC=chip1@LC44`

Related Topics:

- Refer to the following sources for additional information:
  - Go to "Devices & Adapters" and "Assigning a Device" in MAX+PLUS II Help for information on device pin-outs and assigning devices, respectively, in the MAX+PLUS II software.
  - Go to Back-Annotating MAX+PLUS II Pin Assignments to Design Architect Symbols for information on back-annotating pin assignments in Mentor Graphics Design Architect schematics.

Assigning Cliques

You can define a group of logic functions as a single, named unit, called a clique. The MAX+PLUS® II Compiler attempts to place all logic in the clique in the same logic array block (LAB) to ensure optimum speed. If the project does not use multi-LAB devices, or if it is not possible to fit all clique members into a single LAB, the clique assignment ensures that all members of a clique are placed in the same device. In FLEX® 6000, FLEX 8000, FLEX 10K, and MAX® 9000 devices the Compiler also attempts to place the logic in LABs in the same row. Cliques therefore allow you to partition a project so that only a minimum number of signals travel between LABs, and to ensure that no unnecessary LAB-to-LAB or device-to-device delays exist on critical timing paths.

To assign a clique, use the following syntax:

✔ `CLIQUE=`<*clique name*>

For example: `CLIQUE=fast1`

Related Topics:

- Go to the following topics in MAX+PLUS II Help for related information:
  - Assigning a Clique
  - Guidelines for Achieving Maximum Speed Performance

Assigning Logic Options

Logic option and logic synthesis style assignments allow you to guide logic synthesis with logic optimization features that are specific to Altera® devices. You can assign logic options and styles to individual logic functions in your design. The MAX+PLUS® II Compiler also uses a device-family-specific default logic synthesis style for each project.

Related Topics:

- Go to "Resource Assignments in EDIF Input Files" and "Assigning Resources in a Third-Party Design Editor" in MAX+PLUS II Help for complete and up-to-date information on logic option and logic synthesis style assignments, including definitions and syntax of these assignments.

Performing a Functional Simulation with DVE & QuickSim II Software

You can perform a functional simulation of a Design Architect schematic with the Mentor Graphics Design Viewpoint Editor (DVE) and QuickSim II software before compiling your project with the MAX+PLUS ® II Compiler.

If you wish to functionally simulate a hierarchical design that uses multiple design entry methods, you should use QuickHDL Pro rather than QuickSim. Refer to Performing a Functional Simulation with QuickHDL Pro Software for more information.

To functionally simulate a Design Architect schematic, go through the following steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment.
2. Create a Design Architect schematic that follows the guidelines in Creating Design Architect Schematics for Use with MAX+PLUS II Software.
3. In the Navigator window, select your project's folder, press Button 3, and choose **Open max2_fve** to start DVE. DVE checks the design and creates a viewpoint (called **altera_fsim** by default) for functional simulation with QuickSim II software.
4. Select the **altera_fsim** icon, press Button 3, and choose **Open max2_qsim** from the Navigator window to start the QuickSim II software. You can also start the QuickSim II software by typing `max2_qsim` ↵ at the UNIX prompt.
5. Set the appropriate options and simulate your design.
6. Use the ENWrite utility to generate an EDIF netlist file that can be imported into the MAX+PLUS II software, as described in Converting Design Architect Schematics into MAX+PLUS II-Compatible EDIF Netlist Files with the ENWrite Utility.

Related Topics:

- Go to Compiling Projects with MAX+PLUS II Software in these MAX+PLUS II ACCESS℠ Key topics for related information.

Creating Hierarchical Projects with Design Architect Software

If you wish to create a hierarchical schematic design that contains symbols representing other design files, such as AHDL Text Design Files (**.tdf**), VHDL Design Files (**.vhd**), or Verilog Design Files (**.v**), you can create a hollow-body symbol for the design file and then instantiate it in your top-level design file.

To create a hollow-body symbol for a lower-level design file, follow these steps:

1. (Optional) If you are creating a hollow-body symbol for a VHDL or Verilog HDL design file, you can first functionally simulate the VHDL or Verilog HDL file, as described in Performing a Functional Simulation

with QuickHDL Software.

2. Start the Design Architect software by double-clicking Button 1 on the **max2_da** icon in the Design Manager tools window. You can also start Design Architect software by typing `max2_da` ↵ at the UNIX prompt.

3. Choose the **OPEN SYMBOL** button in the Design Architect session_palette to open the Symbol Editor. Type the lower-level design file name, including the directory path, in the *Component Name* box. Choose **OK**.

4. Create a symbol that represents the inputs and outputs of the lower-level file.

5. Assign `PINTYPE` properties of `IN` or `OUT` to the inputs and outputs of the symbol, and assign appropriate values to any other properties of the symbol so that it can be identified in the top-level schematic.

   ☞ If you are creating a hollow-body symbol for a VHDL design file, be sure to assign the value `qvpro` to the symbol's `model` property so that it can be identified as a VHDL component in the top-level schematic.

6. Check and save the symbol, then close the Symbol Editor.

7. To enter the symbol, choose the **CHOOSE SYMBOL** button from the Design Architect session_palette.

8. Select the symbol file from the Navigator menu and choose **OK**.

9. The MAX+PLUS® II software uses the Altera® provided **mnt8_bas.lmf** Library Mapping File to map Design Architect symbols to equivalent MAX+PLUS II logic functions. To use custom symbols, you must create a custom LMF that maps your custom symbols to the equivalent EDIF Input File, Text Design File (TDF), or other design file. You will also need to specify this LMF in the **EDIF Netlist Reader Settings** dialog box before compiling the design with the MAX+PLUS II software. See Compiling Projects with MAX+PLUS II Software for more information.

10. Continue with the steps necessary to complete your Design Architect schematic, as described in Creating Design Architect Schematics for Use with MAX+PLUS II Software.

Installing the Alteraprovided MAX+PLUS II/Mentor Graphics/Exemplar Logic interface on your computer automatically creates the sample hierarchical Design Architect schematic file **/usr/maxplus2/examples/mentor/example3/fulladd2**.

Converting Design Architect Schematics into MAX+PLUS II- Compatible EDIF Netlist Files with the ENWrite Utility

After you have created a Design Architect schematic or a hierarchical schematic design that uses multiple design entry methods, you can use the Mentor Graphics ENWrite utility to convert it into an EDIF netlist file that can be processed with the MAX+PLUS ® II software.

To generate an EDIF netlist file for use with the MAX+PLUS II Compiler, go through the following steps:

1. Create a Design Architect Schematic that follows the guidelines described in Creating Design Architect Schematics for Use with MAX+PLUS II Software.

2. Select the folder for your project, press Button 3, and choose **Open max2_enw** from the Navigator window to open Design Viewpoint Editor (DVE), then ENWrite. You can also start the ENWrite utility by typing `max2_enw` ↵ at the UNIX prompt.

3. Choose **OK** in the **$invoke_enw** dialog box to accept the default names for the DVE viewpoint **altera_edif**,

which is used internally by ENWrite, and the ENWrite hierarchical EDIF netlist file <*design name*>**.edf**. Specify *OFF* for the port array construct in the EDIF netlist file.

The MAX+PLUS II software supports bus constructs in EDIF 2 0 0 and 3 0 0 netlist files, which allow you to retain any bus structures in your design. To preserve a bus in the EDIF netlist file, turn on the *port array* construct option in the **$invoke_enw** dialog box. However, if your design contains library of parameterized modules (LPM) functions, you should not use this feature because LPM 2.0.1 and 2.1.0 functions do not support EDIF bus constructs.

After DVE checks the Design Architect schematic, ENWrite generates <*design name*>**.edf** and automatically copies it to your project's directory.

4. Compile the resulting EDIF netlist file with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Compiling Projects with MAX+PLUS II Software

The MAX+PLUS® II Compiler can process design files in a variety of formats. This topic describes how to use MAX+PLUS II software to compile projects in which the top-level design file is an EDIF Input File (with the extension **.edf**).

Refer to the following sources for additional information:

- Go to MAX+PLUS II Help for information on compiling VHDL and Verilog HDL, design files directly with the MAX+PLUS II Compiler.
- Go to Running Synopsys Compilers from MAX+PLUS II Software for information on running the Synopsys Design Compiler or FPGA Compiler software on a VHDL or Verilog HDL design from within the MAX+PLUS II Compiler window.

To compile a design (also called a "project") with MAX+PLUS II software, go through the following steps:

1. Create design files that are compatible with the MAX+PLUS II software and convert them into EDIF Input Files with the extension **.edf**. Specific instructions for some tools are described in these MAX+PLUS II ACCESS℠ Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your design entry or synthesis and optimization tool.
2. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, create a mapping for each function in a Library Mapping File (**.lmf**) to map the custom symbol to the corresponding EDIF Input File, AHDL Text Design File (**.tdf**), or other MAX+PLUS II-supported design file. These custom functions are represented in design files as hollow-body symbols or "black box" HDL descriptions.

   Go to "Library Mapping Files (**.lmf**)" in MAX+PLUS II Help for more information.

3. Open MAX+PLUS II and specify the name of your top-level design file as the project name with the **Project Name** command (File menu). If you open an HDL file in the MAX+PLUS II Text Editor, you can choose the **Project Set Project to Current File** command (File menu) instead.

   You can also compile a project from a command line. However, the first time you compile a project, the settings you need to specify are easier to specify from within the MAX+PLUS II software. After you have run the graphical user interface for the MAX+PLUS II software at least once, you can more easily use the command-line **setacf** utility to modify options in the Assignment & Configuration File

(**.acf**) for the project. Type `setacf -h` ↵ and `maxplus2 -h` ↵ for descriptions of **setacf** and MAX+PLUS II command-line syntax.

4. Choose **Device** (Assign menu) and select the target Altera device family in the *Device Family* drop-down list box. If you wish to implement the design logic in a specific device, select it in the *Devices* box. Otherwise, select *AUTO* to allow the MAX+PLUS II Compiler to choose the best device(s) in the current device family. If your design entry or synthesis and optimization tool required you to specify a target family and/or device, specify the same information in this dialog box. For information on partitioning logic among multiple devices, go to MAX+PLUS II Help. Choose **OK**.

5. Open the Compiler window by choosing the **Compiler** command (MAX+PLUS II menu). Go through the following steps to specify the options necessary to compile the design file(s) in your project:

    1. Ensure that all EDIF netlist files have the extension **.edf** and choose **EDIF Netlist Reader Settings** (Interfaces menu).

    2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor. This name should be the name of the vendor whose tool(s) you used to create the EDIF netlist files. If your vendor name does not appear, select *Custom* instead.

        If you are compiling a design created with Synopsys FPGA Express software, select *Synopsys*, choose the **Customize** button, enter <*project name*>**.lmf** in the *LMF #1* box, choose **OK**, and skip to step 6.

    3. If you selected an existing vendor name in the *Vendor* box and your project contains design files that require custom LMF mappings, choose the **Customize** button to expand the dialog box to show all settings. Turn on the *LMF #2* checkbox and type your custom LMF's filename in the corresponding text box, or select a name from the *Files* box. The selection in the *Vendor* box will change to *Custom* and all settings will be retained until you change them again.

    4. If you selected *Custom* in the *Vendor* box, choose the **Customize** button to expand the dialog box to show all settings. Any previously defined custom settings will be displayed. Under *Signal Names*, type one or more names with up to 20 total name characters in the *VCC* or *GND* box if your EDIF Input File(s) use one or more names other than `VCC` or `GND` for the global high or low signals. Multiple signal names must be separated by either a comma (,) or a space. Under *Library Mapping Files*, turn on the *LMF #1* checkbox and type a filename in the text box following it, or select a name from the *Files* box. If necessary, specify another LMF name in the *LMF #2* box. Go to MAX+PLUS II Help for detailed information on the settings available in the **EDIF Netlist Reader Settings** dialog box.

    5. Choose **OK**.

6. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, you may need to ensure that all files are present in your project directory, i.e., the same directory as the top-level design file. Otherwise, you must specify the directories containing these files as user libraries with the **User Libraries** command (Options menu).

7. Follow all guidelines that apply to your design entry or synthesis and optimization tool:

    - Exemplar Logic Galileo Extreme-Specific Compiler Settings
    - Synopsys DesignWare-Specific Compiler Settings
    - Converting Synopsys FPGA Compiler & Design Compiler Timing Constraints into MAX+PLUS II-Compatible Format with the **syn2acf** Utility
    - Synplicity Synplify-Specific Compiler Settings

8. If you wish to generate EDIF, VHDL, or Verilog HDL output files for post-compilation simulation or timing analysis with another EDA tool, go through the following steps:

1. (Optional) Turn on the **Optimize Timing SNF** command (Processing menu) to reduce the size of the output file(s). Turning on this command can reduce the size of output netlists by up to 30%.

   This command does not create optimized timing SNFs on UNIX workstations. However, a non-optimized timing SNF provides the same functional and timing information as an optimized timing SNF.

2. If you wish to generate EDIF Output Files (**.edo**), go through these steps:

   1. Turn on the **EDIF Netlist Writer** command (Interfaces menu). Then choose the **EDIF Netlist Writer Settings** command (Interfaces menu).

   2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor and choose **OK**. If your vendor name does not appear, select *Custom* instead and specify the settings that are appropriate for your simulation or timing analysis tool. Go to MAX+PLUS II Help for detailed information on the options available in the **EDIF Netlist Writer Settings** dialog box.

   3. To generate an optional Standard Delay Format (SDF) Output File (**.sdo**), choose the **Customize** button to expand the dialog box to show all settings. Select one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**.

   The filenames of the EDIF Output File(s) and optional SDF Output File(s) are the same as the user-defined chip name(s) for the project; if no chip names exist, the Compiler assigns filenames that are based on the project name. For a multi-device project, the Compiler also generates a top-level EDIF Output File that is uniquely identified by "**_t**" appended to the project name. In addition, the Compiler automatically generates a VHDL Memory Model Output File, <*project name*>**.vmo**, when it generates an EDIF Output File that contains memory (RAM or ROM).

3. If you wish to generate VHDL Output Files (**.vho**), turn on the **VHDL Netlist Writer** command (Interfaces menu). Then choose **VHDL Netlist Writer Settings** command (Interfaces menu). Select *VHDL Output File (.vho)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF ver. 2.1 files contain timing delay information that allows you to perform back-annotation simulation in VHDL with VITAL-compliant simulation libraries. The VHDL Output Files generated by the Compiler have the extension **.vho**, but are otherwise named in the same way as the EDIF Output Files described above.

4. If you wish to generate Verilog HDL Output Files (**.vo**), turn on the **Verilog Netlist Writer** command (Interfaces menu). Then choose **Verilog Netlist Writer Settings** command (Interfaces menu). Select *Verilog Output File (.vo)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF Output Files contain timing delay information that allows you to perform back-annotation simulation in Verilog HDL. The Verilog Output Files generated by the Compiler have the extension **.vo**, but are otherwise named in the same way as the EDIF Output Files described above.

9. To run the MAX+PLUS II Compiler, choose the **Project Save & Compile** command (File menu) or choose the **Start** button in the Compiler window.

   See step 3 for information on running MAX+PLUS II software from the command line.

10. Once you have compiled the project with the MAX+PLUS II Compiler, you can use the VHDL, Verilog HDL, or EDIF output file(s), and the optional SDF Output File(s) (**.sdo**) to perform timing analysis or timing simulation with another EDA tool. Specific instructions for some tools are described in these MAX+PLUS II ACCESS Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your EDA tool.

The MAX+PLUS II Compiler also generates a Report File (**.rpt**), a Pin-Out File (**.pin**), and one or more of the following files for device programming or configuration:

- JEDEC Files (**.jed**)
- Programmer Object Files (**.pof**)
- SRAM Object Files (**.sof**)
- Hexadecimal (Intel-format) Files (**.hex**)
- Tabular Text Files (**.ttf**)

Related Topics:

- Go to *Compiler Procedures* in MAX+PLUS II Help for information on other available Compiler settings.
- Go to *Programmer Procedures* in MAX+PLUS II Help for instructions on creating other types of programming files and on programming or configuring Altera devices.
- Go to Back-Annotating MAX+PLUS II Pin Assignments to Design Architect Symbols for information on back-annotating pin assignments in Mentor Graphics Design Architect schematics.
- Go to Programming Altera Devices for information on the different programming hardware options for Altera device families.
- Go to the following topics, which are available on the web, for additional information:
    - MAX+PLUS II Development Software
    - Altera Programming Hardware

Using Cadence Tools with MAX+PLUS II Software

The following topics describe how to use a variety of Cadence tools as part of a complete design flow that includes the MAX+PLUS® II software. If you use only one Cadence tool, click List by Tool and select the tool name to view the list of topics only for that tool. Click on one of the following topics for information:

This file is suitable for printing only. It does not contain hypertext links that allow you to jump from topic to topic.

Setting Up the MAX+PLUS II/Cadence Working Environment

- Software Requirements
- Setting Up the MAX+PLUS II/Cadence Concept Work Environment for a Sun SPARCstation Running SunOS Software
- MAX+PLUS II/Cadence Interface File Organization
- MAX+PLUS II Directory Structure
- Concept & RapidSIM Local Work Area Directory Structure
- Concept & HDL Direct Project Directory Structure
- Composer Project File Directory Structure
- Altera-Provided Logic & Symbol Libraries
- Compiling the VITAL Library for Use with Leapfrog Software
- Compiling the **alt_mf** Library

Design Flow for All Cadence Tools

Design Entry

- **Design Entry Flow**

- **Concept**

    - Creating Concept Schematics for Use with MAX+PLUS II Software
        - Instantiating the clklock Megafunction in Concept Schematics
        - Instantiating LPM & Other Parameterized Functions in Concept Schematics
    - Entering Resource Assignments
        - Assigning Pins, Logic Cells & Chips
        - Assigning Cliques
        - Assigning Logic Options
        - Modifying the Assignment & Configuration File with the **setacf** Utility
    - Performing a Functional Simulation of a Concept Schematic with the **hdlconfig** Utility & Verilog-XL Software
    - Performing a Functional Simulation of a Concept Schematic with VerilogLink & Verilog-XL Software
    - Creating Hierarchical Projects in Concept Schematics
    - Converting Concept Schematics into MAX+PLUS II-Compatible EDIF Netlist Files with the **concept2alt** Utility

- **Composer**

- Creating Composer Schematics for Use with MAX+PLUS II Software
- Entering Resource Assignments
  - Assigning Pins, Logic Cells & Chips
  - Assigning Cliques
  - Assigning Logic Options
  - Modifying the Assignment & Configuration File with the **setacf** Utility
- Performing a Functional Simulation of a Composer Schematic with Verilog-XL Software
- Creating Hierarchical Projects in Composer Schematics
- Converting Composer Schematics into MAX+PLUS II-Compatible EDIF Netlist Files with the **altout** Utility

- **VHDL**

  - Creating VHDL Designs for Use with MAX+PLUS II Software
    - Instantiating the clklock Megafunction in VHDL or Verilog HDL
  - Entering Resource Assignments
    - Modifying the Assignment & Configuration File with the **setacf** Utility

- **Verilog HDL**

  - Creating Verilog HDL Designs for Use with MAX+PLUS II Software
    - Instantiating the clklock Megafunction in VHDL or Verilog HDL
  - Entering Resource Assignments
    - Modifying the Assignment & Configuration File with the **setacf** Utility

Synthesis & Optimization

- **VHDL**

  - Synthesizing & Optimizing VHDL Files with Synergy Software
  - Converting VHDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the **vlog2alt** or **altout** Utility

- **Verilog HDL**

  - Synthesizing & Optimizing Verilog HDL Files with Synergy Software
  - Converting Verilog HDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the **vlog2alt** Utility

Compilation

- Project Compilation Flow
- Compiling Projects with MAX+PLUS II Software

Simulation

- Project Simulation Flow
- Initializing Registers in VHDL & Verilog Output Files for Power-Up before Simulation
- Performing a Timing Simulation with RapidSIM Software
- Performing a Timing Simulation with Verilog-XL Software
- Performing a Timing Simulation with Leapfrog Software

Compiling the VITAL Library for Use with Leapfrog Software
- Compiling the **alt_mf** Library

Device Programming

- Programming Altera Devices

Related Topics:

- MAX+PLUS II Development Software
- Altera Programming Hardware
- Cadence web site (http://www.cadence.com)

Setting Up the MAX+PLUS II/Cadence Working Environment

To use MAX+PLUS® II software with Cadence software, you must first install the MAX+PLUS II software, then establish an environment that facilitates entering and processing designs. The MAX+PLUS II/Cadence interface is installed automatically when you install the MAX+PLUS II software on your computer. Go to *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual for more information on installation and details on the directories that are created during MAX+PLUS II installation. Go to MAX+PLUS II/Cadence Interface File Organization for information about the MAX+PLUS II/Cadence directories that are created during MAX+PLUS II installation.

☞ The information presented here assumes that you are using the C shell and that your MAX+PLUS II system directory is **/usr/maxplus2**. If not, you must use the appropriate syntax and procedures to set environment variables for your shell.

To set up your working environment for the MAX+PLUS II/Cadence interface, follow these steps:

1. Ensure that you have correctly installed the MAX+PLUS II and Cadence software versions described in the MAX+PLUS II/Cadence Software Requirements.
2. Add the following environment variables to your **.cshrc** file:

   ```
   setenv ALT_HOME /usr/maxplus2 ↵

   setenv CDS_INST_DIR <Cadence system directory path> ↵
   ```

3. Add the **$ALT_HOME/cadence/bin** and **$CDS_INST_DIR/tools/bin** directories to the `PATH` environment variable in your **.cshrc** file. Make sure these paths are placed before the Cadence hierarchy path.
4. Add **/usr/dt/lib** and **/usr/ucb/lib** to the `LD_LIBRARY_PATH` environment variable in your **.cshrc** file.
5. Create a new **cds.lib** file in your working directory or edit an existing one so that it includes all of the following lines that apply to the Cadence tools you have installed:

   ```
   DEFINE alt_syn ${ALT_HOME}/simlib/concept/alt_syn

   DEFINE lpm_syn ${ALT_HOME}/simlib/concept/lpm_syn

   DEFINE alt_lpm ${ALT_HOME}/simlib/concept/alt_lpm

   DEFINE alt_mf ${ALT_HOME}/simlib/concept/alt_mf

   DEFINE alt max2 ${ALT HOME}/simlib/concept/alt max2
   ```

```
DEFINE alt_max2 ${ALT_HOME}/simlib/composer/alt_max2/alt_max2

DEFINE alt_vtl $ALT_HOME/simlib/concept/alt_vtl/lib

DEFINE altera $ALT_HOME/simlib/concept/alt_mf/lib

SOFTINCLUDE $CDS_INST_DIR/tools/leapfrog/files/cds.lib

DEFINE <design name>.
```

6. Copy the **/usr/maxplus2/maxplus2.ini** file to your **$HOME** directory:

```
cp /usr/maxplus2/maxplus2.ini $HOME↵

chmod u+w $HOME/maxplus2.ini↵
```

The **maxplus2.ini** file contains both Altera- and user-specified initialization parameters that control the MAX+PLUS II software, such as MAX+PLUS II symbol and logic function library paths and the current project name. The MAX+PLUS II installation procedure creates and copies the **maxplus2.ini** file to the **/usr/maxplus2** directory.

Normally, you do not have to edit your local copy of **maxplus2.ini** because the MAX+PLUS II software updates the file automatically whenever you change any parameters or settings. However, if you move the **max2lib** and **max2inc** library subdirectories, you must update the file. Go to "Creating & Using a Local Copy of the **maxplus2.ini** File" in MAX+PLUS II Help for more information.

7. If you are using Concept on a Sun SPARCstation running SunOS, go to Setting Up the MAX+PLUS II/Cadence Concept Work Environment for a Sun SPARCstation Running SunOS Software to install the **redifnet** EDIF netlist reader utility.

8. If you are using Synergy software, edit the **hdl.var** file located in your working directory to include the following line:

```
DEFINE work <design name>↵
```

9. Set up an appropriate directory structure for the tool(s) you are using. See the following topics for information:

    - Composer Project File Directory Structure
    - Concept & RapidSIM Local Work Area Directory Structure

Related Topics:

Go to the following topics, which are available on the web, for additional information:

- ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
- This manual is also available in 4 parts:
    - Preface & Section 1: MAX+PLUS II Installation
    - Section 2: MAX+PLUS II - A Perspective
    - Section 3: MAX+PLUS II Tutorial
    - Appendices, Glossary & Index

MAX+PLUS II/Cadence Software Requirements

The following table shows the software applications that are used to generate, process, synthesize, and verify a project with MAX+PLUS® II and Cadence software:

Cadence		Altera
version 97A: Concept Composer ValidCOMPILER **concept2alt** **vlog2alt** **altout**	VerilogLink Synergy HDL Direct (Concept 2.0 or later) Non-Graphic Simulation Environment (SE) RapidSIM, Verilog-XL, or Leapfrog **redifnet** (SunOS only)	MAX+PLUS II version 9.4

☞ The MAX+PLUS II **read.me** file provides up-to-date information on which versions of Cadence software applications are supported by the current version of MAX+PLUS II. It also provides information on installation and operating requirements. You should read the **read.me** file on the CD-ROM before installing the MAX+PLUS II software. After installation, you can open the **read.me** file from the MAX+PLUS II Help menu.

Setting Up the MAX+PLUS II/Cadence Concept Work Environment for a Sun SPARCstation Running SunOS Software

If you are using Concept software on a Sun SPARCstation running SunOS software, you should also install the **redifnet** EDIF netlist reader utility to convert Concept schematics into MAX+PLUS II-compatible EDIF netlist files. To install the **redifnet** utility, follow these steps:

1. Copy the **redifnet** directory from the **/usr/maxplus2/simlib/concept/edifnet** directory to the Cadence system directory.
2. Copy the **redifnet** and **pinmap_start** files from the **/usr/maxplus2/simlib/concept/edifnet/bin** directory to the /<*Cadence system directory path*>/**tools/bin**.
3. Specify the **-/usr/maxplus2/simlib/concept/edifnet/max2sim** map file as a `PIN_MAP_FILE` in the **redifnet.cmd** file.
4. (Optional) Modify existing templates for directive files such as **compiler.cmd**, **vloglink.cmd**, and **global.cmd**. These templates are located in the **/usr/maxplus2/simlib/concept/edifnet/templates** directory.
5. (Optional) Modify the **expansion.dat** and **max2sim.map** files in the **/usr/maxplus2/simlib/concept/edifnet** directory.

MAX+PLUS II/Cadence Interface File Organization

Table 1 shows the MAX+PLUS® II/Cadence interface subdirectories that are created in the MAX+PLUS II system directory (by default, the **/usr/maxplus2** directory) during MAX+PLUS II installation. For information on the other directories that are created during MAX+PLUS II installation, see "MAX+PLUS II File Organization" in *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual.

Table 1. MAX+PLUS II Directory Organization

Directory	Description

./lmf	Contains the Altera-provided Library Mapping File, **cadence.lmf**, that maps Cadence logic functions to equivalent MAX+PLUS II logic functions.
./examples/cadence	Contains the sample files for Cadence software discussed in these ACCESS^{SM} Key Guidelines.
./cadence	Contains the AMPLE userware for the MAX+PLUS II/Cadence interface.
./simlib/concept/alt_max2	Contains the MAX+PLUS II primitives, including `CARRY`, `CASCADE`, `EXP`, `GLOBAL`, `LCELL`, `SOFT`, `OPNDRN`, `DFFE` (D flipflop with Clock Enable), and `DFFE6K` (D flipflop with Clock Enable and both Clear and Preset for FLEX® 6000 devices only) for use with Concept software.
./simlib/composer/alt_max2	Contains the MAX+PLUS II primitives, including `CARRY`, `CASCADE`, `EXP`, `GLOBAL`, `LCELL`, `SOFT`, `OPNDRN`, `DFFE` (D flipflop with Clock Enable), and `DFFE6K` (D flipflop with Clock Enable and both Clear and Preset for FLEX 6000 devices only) for use with Composer software.
./simlib/concept/alt_lpm	Contains the MAX+PLUS II megafunctions, including library of parameterized modules (LPM) functions, for use with Concept software.
./simlib/concept/max2sim	Contains the MAX+PLUS II/Concept simulation model library, **max2_sim**, for use with RapidSIM software.
./simlib/concept/alt_syn	Contains the MAX+PLUS II synthesis library, **alt_syn**, for use with Synergy and Concept software, and the **vlog2alt** utility.
./simlib/composer/alt_syn	Contains the MAX+PLUS II synthesis library, **alt_syn**, for use with Synergy and Composer software.
./simlib/concept/lpm_syn	Contains the Cadence LPM library, **lpm_syn**, for use with Synergy and Concept software.
./simlib/composer/lpm_syn	Contains the Cadence LPM library, **lpm_syn**, for use with Synergy and Composer software.
./simlib/concept/alt_mf	Contains the MAX+PLUS II VHDL logic function library. (`a_8count` is for the MAX® 7000 and MAX 9000 device families only.)
./simlib/concept/edifnet/templates	Contains template files for Concept directives, i.e., **global.cmd**, **compiler.cmd**, **vloglink.cmd**, **verilog.cmd**, and **master.local**.
./simlib/concept/alt_max2/verilogUdps	Contains Verilog HDL modules that are the equivalent of the primitives contained in **alt_max2** library for use with Concept software.
./simlib/composer/alt_max2/verilogUdps	Contains Verilog HDL modules that are the equivalent of the primitives contained in **alt_max2** library for use with Composer software.
./simlib/concept/alt_vtl **./simlib/composer/alt_vtl**	Contains VITAL library source files for use with Concept or Composer software.
./simlib/composer/alt_max2/verilog	Contains simulation modules for all symbols in the **alt_max2** Composer library.

Related Topics:

Go to the following topics, which are available on the web, for additional information:

- ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
- This manual is also available in 4 parts:
  - Preface & Section 1: MAX+PLUS II Installation
  - Section 2: MAX+PLUS II - A Perspective
  - Section 3: MAX+PLUS II Tutorial
  - Appendices, Glossary & Index
- FLEX Devices
- MAX Devices
- Classic Device Family

MAX+PLUS II Directory Structure

In the MAX+PLUS® II software, a project name is the name of a top-level design file, without the filename extension. This design file can be an EDIF, Verilog HDL, or VHDL netlist file; an AHDL Text Design File (TDF); or any other MAX+PLUS II-supported design file. The EDIF netlist file must be created by the **altout** or **concept2alt** utility and imported into the MAX+PLUS II software as an EDIF Input File (**.edf**).

Project design files and output files are stored in the project directory, with the exception of standard library functions provided by Altera or another EDA tool vendor. The MAX+PLUS II software stores the connectivity data on the links between design files in a hierarchical project in a Hierarchy Interconnect File (**.hif**), but refers to the entire project only by its project name. The MAX+PLUS II Compiler uses the HIF to build a single, fully flattened project database that integrates all design files in a project hierarchy.

Concept & RapidSIM Local Work Area Directory Structure

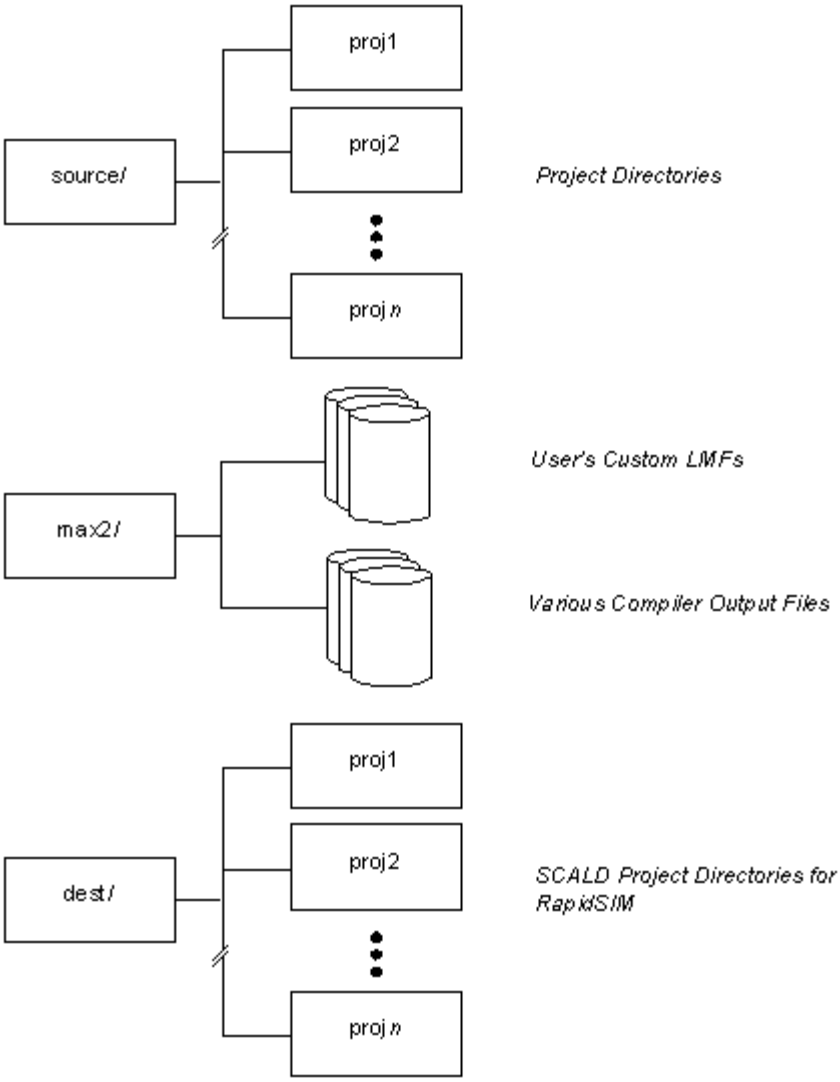
When the **redifnet** utility imports an EDIF netlist file for the RapidSIM software, it creates a SCALD directory for your project. However, creating this directory may overwrite the directory that was created for the original Concept schematic. To prevent overwriting this directory, you should create a file structure that helps you manage your design files.

Altera recommends that you create the following three directories for your design files.

Directory:	Description:
./source	Create Concept schematics and generate EDIF netlist files with the **wedifnet** utility in the **source** directory.
./max2	Copy the EDIF Input File (**.edf**) from the **source** directory to this directory to compile the file with the MAX+PLUS® II software.
./dest	Copy the EDIF Output File (**.edo**) from the **max2** directory to this directory to run the **redifnet** and RapidSIM software.

Copies of the appropriate directives files for Cadence tools must be present in both the **source** and **dest** directories. Figure 1 shows Altera's recommended file structure.

Figure 1. Recommended File Structure



Concept & HDL Direct Project Directory Structure

Concept software generates the following files for each schematic:

- <*drawing name*>**/logic.1.1**
- <*drawing name*>**/logic_bn.1.1**
- <*drawing name*>**/logic_cn.1.1**
- <*drawing name*>**/logic_dp.1.1**

For designs that use HDL Direct software, Concept software also generates the following files:

- <*drawing name*>**/logic_dp.1.1**
- <*drawing name*>**/logic_vd.1.1**
- <*drawing name*>**/logic/verilog.v**
- <*drawing name*>**/logic/vhdl.vhd**
- <*drawing name*>**/logic/hdldirect.dat**
- <*drawing name*>**/entity/vhdl.vhd**

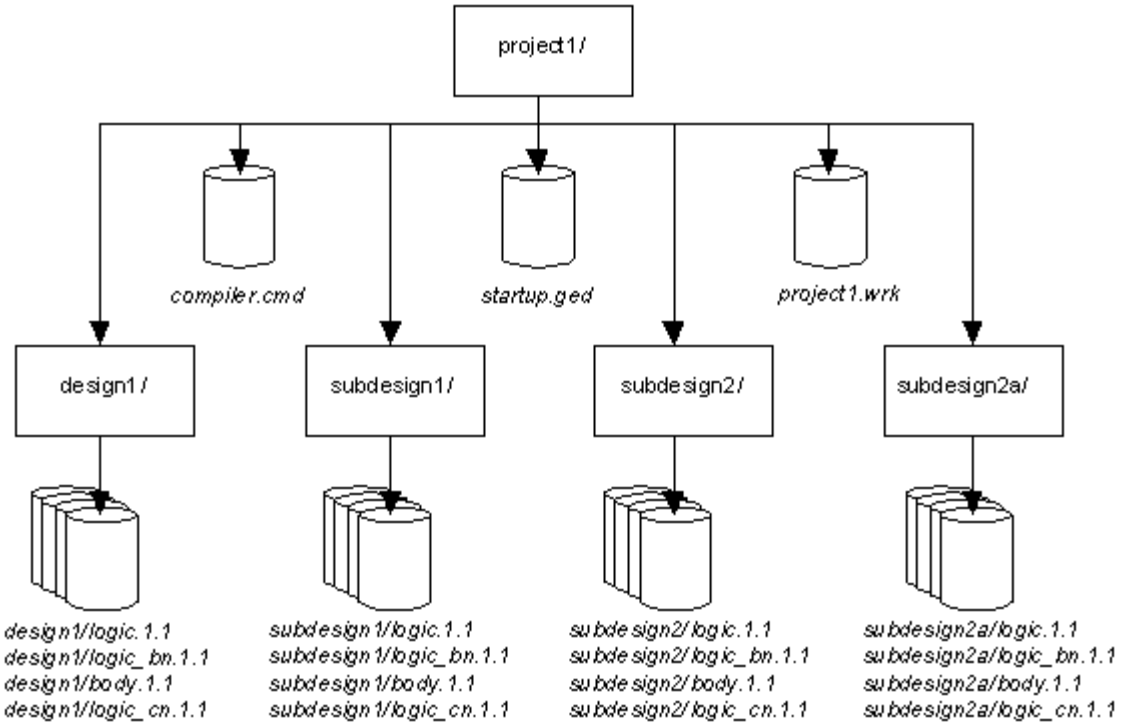
These files are stored in their own <*drawing name*> directories. However, hierarchical relationships between files are not reflected in the file directory structure.

The local SCALD directory has an entry for all <*drawing name*> directories. Cadence software automatically

manages drawing storage and retrieval operations through this special directory. The SCALD directory should have the same name as the UNIX project directory, but with the extension **.wrk**. Figure 1 shows a sample file structure, with **project1** as the UNIX project directory, and **project1.wrk** as the SCALD directory.

When the **concept2alt** utility converts the schematic into an EDIF netlist file, it processes the design information and all related file subdirectories, then creates the EDIF netlist file in the directory defined by the user. The EDIF netlist file is named <*project name*>**.edf**, where <*project name*> is the name of the top-level design file. Figure 1 shows the Cadence project file structure.

Figure 1. Cadence Project File Structure



Composer Project File Directory Structure

The Composer software generates the following files for each schematic (where *x* represents a Composer-generated number):

- ***<drawing name>*_x/schema_59.0_x**
- ***<drawing name>*_x/schema_59.0_x%**

Altera-Provided Logic & Symbol Libraries

The MAX+PLUS® II/Cadence environment provides four logic and symbol libraries that are used for compiling, synthesizing, and simulating designs.

You can create your own libraries of custom symbols and logic functions in Concept and Composer. You can use custom symbols to incorporate an EDIF Input File, Text Design File (TDF), or any other MAX+PLUS II-supported design file into a project. MAX+PLUS II uses the **cadence.lmf** Library Mapping File to map standard Concept or Composer symbols to equivalent MAX+PLUS II logic functions. To use custom symbols, you can create a custom LMF that maps your custom symbols to the equivalent MAX+PLUS II-supported design file.

You must also specify the directory that contains the MAX+PLUS II-supported design file(s) as a user library with the MAX+PLUS II **User Libraries** command (Options menu). Go to "Library Mapping File" and "Cadence Library Mapping File (**cadence.lmf**)" in MAX+PLUS II Help for more information.

The alt_max2 Library

You can enter a Concept or Composer Design Architect schematic with primitives and macrofunctions from the Altera-provided symbol library **alt_max2**. The **alt_max2** library includes 74-series macrofunctions and several MAX+PLUS II primitives with corresponding Verilog HDL simulation models for controlling design synthesis and fitting. It also includes four macrofunctions--`a_8count`, `a_8mcomp`, `a_8fadd`, and `a_81mux`--that are optimized for different device families, and the `clklock` phase-locked loop megafunction, which is supported by some FLEX® 10K devices, with corresponding Verilog HDL and VHDL simulation models. See Table 1. Choose **Old-Style Macrofunctions** and/or **Primitives** from the MAX+PLUS II Help menu for more information on functions in the **alt_max2** library.

The alt_lpm Library

The Altera-provided **alt_lpm** library, which is available for Concept and Verilog HDL designs, includes standard functions from the library of parameterized modules (LPM) 2.1.0, except the truth table, finite state machine, and pad functions. Other parameterized functions, including cycle-shared FIFO (`csfifo`) and cycle-shared dual-port RAM (`csdpram`) are also included. The LPM standard defines a set of parameterized modules (i.e., parameterized megafunctions) and their corresponding representations in an EDIF netlist file. These logic functions allow you to create and functionally simulate an LPM-based design without targeting a specific device family. The parameters you specify for each LPM function determine the simulation models that will be generated. After the design is completed, you can target the design to any device family. In designs created with Concept, the Altera **alt_lpm** library works only with HDL Direct and the **hdlconfig** utility. Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu for more information about LPM functions in the **alt_lpm** library.

The lpm_syn Library

The **lpm_syn** library contains the Altera-provided parameterized functions. The **lpm_syn** library is similar to the **alt_lpm** library, except that it contains VHDL and Verilog HDL logic functions for use with Synergy, Concept, and Composer software.

The alt_mf Library

Altera provides a VHDL logic function library, **alt_mf**, that currently includes four macrofunctions--`a_8count`, `a_8mcomp`, `a_8fadd`, and `a_81mux`--for controlling design synthesis and fitting. These elements can be instantiated directly in your VHDL file. To designate that these logic functions should pass untouched through the EDIF netlist file to the MAX+PLUS II Compiler, you must select the *Maintain* attribute constraint for instances of these functions before running the Synergy software. These models allow you to perform functional VHDL simulation while maintaining an architecture-independent VHDL description.

Table 1 shows the MAX+PLUS II-specific logic functions.

Table 1. MAX+PLUS II-Specific Logic Functions

Macrofunctions *Note (1)*				**Primitives**	
Name	**Description**	**Name**	**Description**	**Name**	**Description**
`8fadd`	8-bit full adder	`LCELL`	Logic cell buffer	`EXP`	MAX® 5000, MAX 7000, and MAX 9000 Expander buffer
`8mcomp`	8-bit magnitude comparator	`GLOBAL`	Global input buffer	`SOFT`	Soft buffer

`8count` *Note (2)*	8-bit up/down counter	`CASCADE`	FLEX 6000, FLEX 8000, and FLEX 10K cascade buffer	`OPNDRN`	Open-drain buffer
`81mux`	8-to-1 multiplexer	`CARRY`	FLEX 6000, FLEX 8000, and FLEX 10K carry buffer	`DFFE` `DFFE6K` *Note (3)*	D-type flipflop with Clock Enable
`clklock`	Phase-locked loop				

Notes:

1. Logic function names that begin with a number must be preceded by "`a_`" in VHDL designs. For example, `8fadd` must be specified as `a_8fadd`.
2. The `a_8count` logic function is for the MAX 7000 and MAX 9000 device families only.
3. For designs that are targeted to FLEX 6000 devices, you should use the `DFFE` primitive only if the design contains either a Clear or Preset signal, but not both. If your design contains both a Clear and a Preset signal, you must use the `DFFE6K` primitive.

Related Topics:

Go to the following topics, which are available on the web, for additional information:

- FLEX Devices
- MAX Devices
- Classic Device Family

Compiling the VITAL Library for Use with Leapfrog Software

If you wish to use MAX+PLUS® II-generated Standard Delay Format (SDF) Output Files (**.sdo**) that contain timing information when performing post-compilation timing simulation with Leapfrog software, you must first compile the VITAL library source files. The VITAL Timing and Primitive package files are located in the **$CDS_INST_DIR/tools/leapfrog/files/IEEE.src** directory.

To compile the **alt_vtl** library, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment. For example, you must ensure that the appropriate directories are specified in the **cds.lib** file that is located in your working directory.
2. Create a VHDL design, as described in Creating VHDL Designs for Use with MAX+PLUS II Software and save it in your working directory.
3. Change to the **alt_vtl** directory by typing `cd /usr/maxplus2/simlib/concept/alt_vtl` ↵ at the UNIX prompt.
4. Edit the **hdl.var** file located in your working directory to include the following line:

   ```
   DEFINE WORK alt_vtl ↵
   ```

5. Create the **/usr/maxplus2/simlib/concept/alt_vtl/lib** directory.
6. Type the following commands at the UNIX prompt from the **/usr/maxplus2/simlib/concept/alt_vtl** directory to compile the library:

   ```
   cv -message -file alt_vtl.vhd ↵
   ```

```
cv -message -file alt_vtl.cmp ↵
```

Compiling the alt_mf Library

If your VHDL design uses functions from the **alt_mf** library, you must compile this library. To compile the **alt_mf** library, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS® II/Cadence Working Environment. For example, you must ensure that the appropriate directories are specified in the **cds.lib** file located in your working directory.
2. Change to the **alt_mf** directory by typing `cd /usr/maxplus2/simlib/concept/alt_mf ↵` at the UNIX prompt.
3. Edit the **hdl.var** file located in your working directory to include the following line:

   ```
   DEFINE work alt_mf ↵
   ```

4. Type the following commands at the UNIX prompt from the **/usr/maxplus2/simlib/concept/alt_mf** directory to compile the library:

   ```
   cv -message -file ./src/mf.vhd ↵
   cv -message -file ./src/mf_components.vhd ↵
   ```

Design Flow for All Cadence Tools

Figure 1 shows the typical design flow for logic circuits created and processed with Cadence and MAX+PLUS® II software. Design Entry Flow, Project Compilation Flow, Project Simulation Flow, and Device Programming Flow show detailed diagrams of each stage of the design flow.

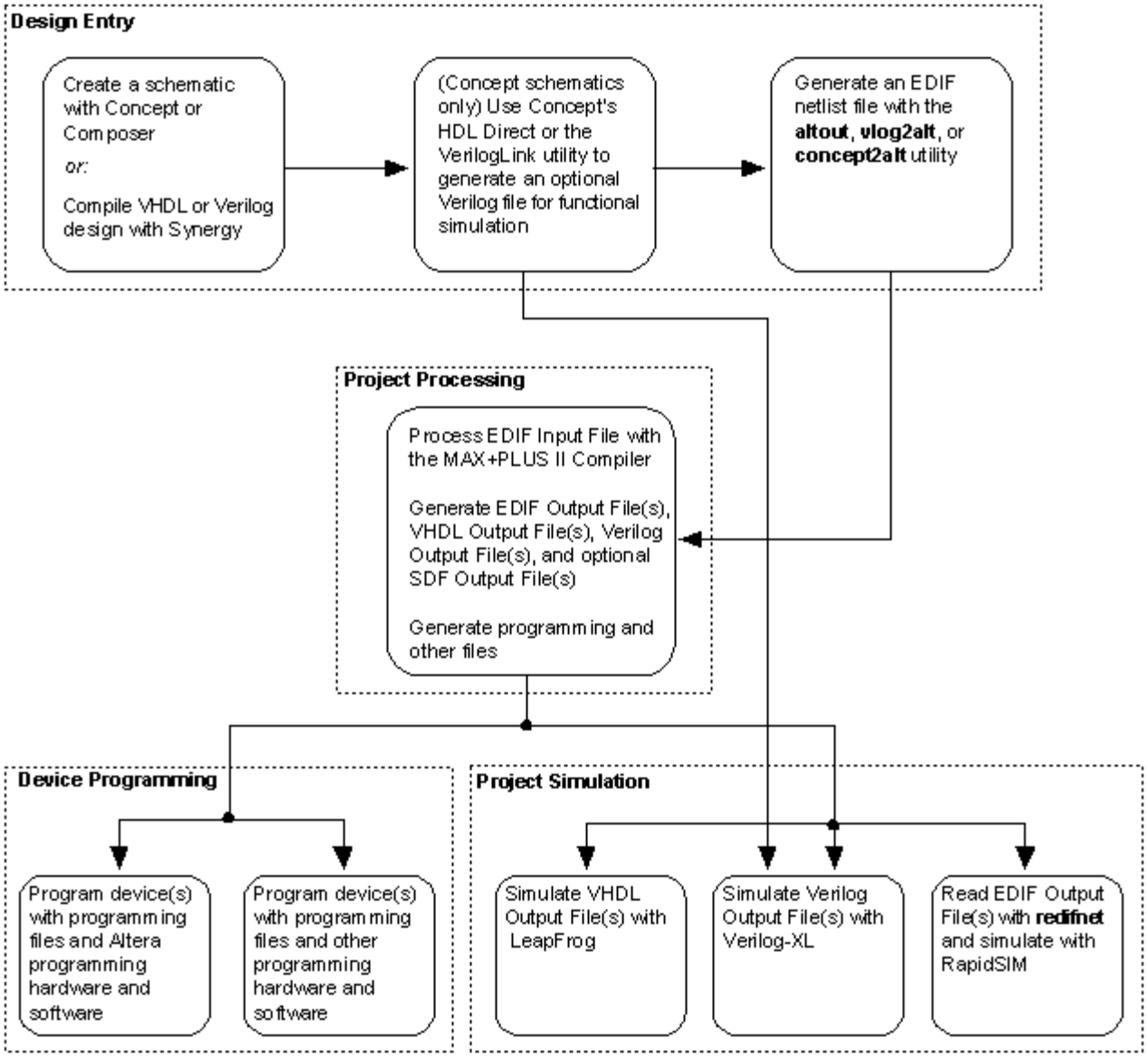


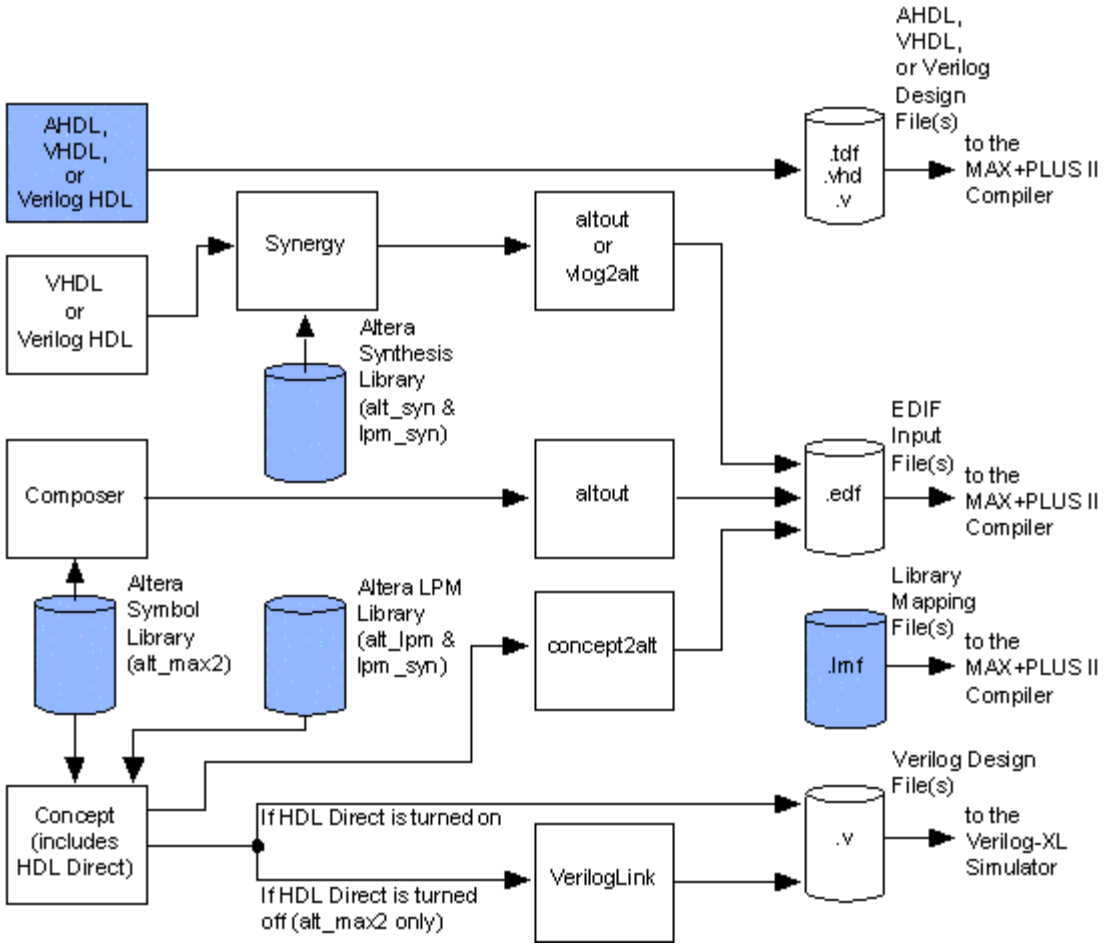
Figure 1. Design Flow between Cadence & MAX+PLUS II Software

Cadence Design Entry Flow

Figure 1 shows the design entry flow for the MAX+PLUS® II/Cadence interface.

Figure 1. MAX+PLUS II/Cadence Design Entry Flow

Altera-provided items are shown in blue.



Creating Concept Schematics for Use with MAX+PLUS II Software

You can create Concept schematics and convert them to EDIF Input Files (**.edf**) that can be processed with the MAX+PLUS® II Compiler. To create a Concept schematic for use with the MAX+PLUS II software, go through the following steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment.

2. Make sure the required directive files are in the /<*working directory*>/<*design name*>/**source** directory. If not, you can use the Altera-provided template files located in the following directories:

   - **/usr/maxplus2/simlib/concept/edifnet/templates**
   - **/usr/maxplus2/simlib/concept/edifnet/redifnet**

3. Start the Concept schematic editor by typing `concept` <*design name*> ↵ at a UNIX prompt from the /<*working directory*>/**source** directory. Use the graphical user interface to structure and organize your files to create an environment that facilitates entering and processing designs. Go to Concept & RapidSIM Local Work Area Directory Structure for more information on directories in Concept.

4. To write a Verilog HDL text file whenever the design is saved, choose the **Block** button in the Concept window.

   ☞ To use the HDL Direct utility to process your design, turn on the *HDL Direct On* option in the Concept window. Go to Concept & HDL Direct Project Directory Structure for information on the files generated by Concept software when using the HDL Direct utility.

5. Enter primitives, megafunctions, and macrofunctions from the following Altera-provided component libraries:

   - **alt_max2** includes macrofunctions, megafunctions, and primitives.
   - **alt_lpm** includes library of parameterized modules (LPM) functions (available only if you use HDL Direct software).

   See the following topics for instructions for specific functions:

   - Instantiating LPM & Other Parameterized Functions in Concept Schematics
   - Instantiating the clklock Megafunction in Concept Schematics

   You can instantiate MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.

6. If you wish to create a hierarchical design that contains symbols representing other design files, such as Altera® Hardware Description Language (AHDL) Text Design Files, go to Creating Hierarchical Projects in Concept Schematics.

7. Enter meaningful instance names for all symbols and functions so that you can easily trace internal node names during simulation and debugging operations. For example, if an `a161` macrofunction is instantiated several times in one design, you should define a unique name for each instance. The instance name for each symbol is controlled by `INST` property. For more information on assigning properties, refer to the Cadence ***Concept Schematic User Guide***.

8. Enter input, output, and bidirectional ports:

   - If you turned on the *HDL Direct On* option in step 4, add `inport` and `outport` symbols from the **hdl_direct_lib** library to the interface symbols.

   - If you are not using HDL Direct, use `flag` symbols from the **standard** library to indicate input, output, and bidirectional ports. Be sure to end pin names with `¥I` to identify them as interface signals.

   If a pin is not used, leave it floating. The **concept2alt** utility removes all unconnected pins when it generates an EDIF netlist file.

9. (Optional) To enter resource assignments in your Concept schematic, go to Entering Resource Assignments. You can also enter resource assignments from within the MAX+PLUS II software.

10. (Optional) Perform a functional simulation, as described in one of the following topics:

    - Performing a Functional Simulation of a Concept Schematic with the **hdlconfig** Utility & Verilog-XL Software
    - Performing a Functional Simulation of a Concept Schematic with VerilogLink & Verilog-XL Software

11. Use the **concept2alt** utility to generate an EDIF netlist file that can be imported into the MAX+PLUS II software, as described in Converting Concept Schematics into MAX+PLUS II-Compatible EDIF Netlist Files with the **concept2alt** utility.

12. Process the <*design name*>**.edf** file with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample Concept schematic files:

- **/usr/maxplus2/examples/cadence/example1/fulladd**
- **/usr/maxplus2/examples/cadence/example4/fulladd2**
- **/usr/maxplus2/examples/cadence/example6/fa2**
- **/usr/maxplus2/examples/cadence/example12/fifo**

Instantiating the clklock Megafunction in Concept Schematics

You can instantiate the `clklock` phase-locked loop megafunction, which is supported in selected FLEX® 10K devices, in a Concept schematic. that employ a phase-locked loop (PLL).

To instantiate the `clklock` megafunction in Cadence Concept schematics, follow these steps:

1. Choose the **Add Part** button from the toolbar or type `add` ↵ in the Concept window to open the Component Browser window.
2. Enter the `clklock` megafunction:
    1. Choose **alt_max2** (Library menu) and select *clklock* from the list box.
    2. Type `attribute`, then select the `clklock` component. Change the *CLOCKBOOST* and *INPUT_FREQUENCY* values as needed. For detailed information on the `clklock` megafunction, choose **Megafunctions/LPM** from the MAX+PLUS® II Help menu.
3. Choose **Done**.
4. Continue with the steps necessary to complete your Concept schematic, as described in Creating Concept Schematics for Use with MAX+PLUS II Software.

    Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample Concept schematic file, which includes `clklock` instantiation:

    - **/usr/maxplus2/examples/cadence/example12/fifo**

Related Topics:

Go to FLEX 10K Device Family, which is available on the web, for additional information. >

Instantiating LPM & Other Parameterized Functions in Concept Schematics

You can use library of parameterized modules (LPM) functions and other Altera® -provided parameterized functions in Concept schematics if you also use the HDL Direct utility.

To instantiate LPM functions, go through the following steps:

1. Choose the **Add Part** button from the toolbar or type `add` from the Concept window to open the Component Browser window.
2. Choose **alt_lpm** (Library menu). All functions in the **alt_lpm** library are MAX+PLUS® II-compatible. Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu to get detailed information on all supported parameterized functions.

3. Type `attribute`, then click on each component to set parameters for each function. See General Guidelines below for additional information.

4. Add `inport` and `outport` symbols from the **hdl_direct_lib** library to the interface signals. Use the `supply_0` and `supply_1` symbols from the **hdl_direct_lib** library to connect a net to `GND` or `VCC`.

5. Continue with the steps necessary to complete your Concept schematic, as described in Creating Concept Schematics for Use with MAX+PLUS II Software.

   Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample Concept schematic file, which includes LPM function instantiation:

   - **/usr/maxplus2/examples/cadence/example12/fifo**

General Guidelines

- If a pin is not used, leave it floating. The **concept2alt** utility removes all unconnected pins when it generates an EDIF netlist file.
- For the `csfifo` function, the value of the `LPM_NUMWORDS` parameter must be between $2^{LPM_WIDTHAD-1}$ and $2^{LPM_WIDTHAD}$.
- Make sure that any hexadecimal (Intel-format) file (**.hex**) that you use to specify the initial content of a memory does not have the same name as the design file name.
- Make sure that all properties and value strings are in uppercase letters, except the filename specified with the `LPM_FILE` property, which should use the actual case of the filename.
- Choose the **Set** button in the Concept window and choose *CAPS_LOCK_OFF* for the *CAPS LOCK* option.
- Only the `LPM_POLARITY` parameter (which can be set to `INVERT` or `NORMAL`) can determine the polarity of the bus or pin. You can display a bubble in the Concept schematic to indicate an inverted pin by typing `BUBBLE` in the Concept command window and selecting the appropriate pin. However, the bubble does not determine the polarity of the pin or bus.
- Avoid using the **Replace** button in the Concept window to replace old symbols with new ones: you may accidentally set unwanted properties. Instead, you should use the **Delete** button to delete old symbols and the **Add** button to add new symbol(s).

Entering Resource Assignments

The MAX+PLUS® II software allows you to enter a variety of resource and device assignments for your projects. Resource assignments are used to assign logic functions to a particular pin, logic cell, I/O cell, embedded cell, row, column, Logic Array Block (LAB), Embedded Array Block (EAB), chip, clique, local routing, logic option, timing requirement, or connected pin group. In MAX+PLUS II software, you can enter all types of resource and device assignments with Assign menu commands. You can also enter pin, logic cell, I/O cell, embedded cell, LAB, EAB, row, and column assignments in the MAX+PLUS II Floorplan Editor. The Assign menu commands and the Floorplan Editor all save assignment information in the ASCII Assignment & Configuration File (**.acf**) for the project. In addition, you can edit ACFs manually in any standard text editor or with the **setacf** utility.

Concept & Composer Schematics

In both Concept and Composer schematics, you can assign a limited subset of these resource assignments by assigning properties to symbols. These properties are incorporated into the EDIF netlist file(s). The MAX+PLUS II

software automatically converts assignment information from the EDIF Input File into the ACF format. For information on making MAX+PLUS II-compatible resource assignments, go to the following topics:

- Assigning Pins, Logic Cells & Chips
- Assigning Cliques
- Assigning Logic Options
- Modifying the Assignment & Configuration File with the **setacf** Utility

Go to the ***Cadence Concept Schematic User Guide*** and ***Composer Reference User Guide*** for details on how to assign properties. Go to "Resource Assignments in EDIF Input Files" and "Assigning Resources in a Third-Party Design Editor" in MAX+PLUS II Help for more information on assignments or properties that can be assigned in Concept and Composer.

Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample Concept and Composer schematic files, which include resource assignments:

- **/usr/maxplus2/examples/cadence/example6/fa2** (Concept)
- **/usr/maxplus2/examples/cadence/example7/fa2** (Composer)

VHDL & Verilog HDL Design Files

For Verilog HDL- and VHDL-based designs, you must use the MAX+PLUS II software or the **setacf** utility to enter resource assignments. For information on using the **setacf** utility, go to Modifying the Assignment & Configuration File with the **setacf** Utility.

Related Topics:

For information on entering assignments in the MAX+PLUS II software with Assign menu commands or in an ACF, go to "resource assignments" or "ACF, format" in MAX+PLUS II Help using **Search for Help on** (Help menu).

Assigning Pins, Logic Cells & Chips

You can assign a single logic function to a specific pin or logic cell (including I/O cells and embedded cells) within a chip, and assign one or more functions to a specific chip. A chip is a group of logic functions defined as a single, named unit, which can be assigned to a specific device.

You can assign a signal to a particular pin to ensure that the signal is always associated with that pin, regardless of future changes to the project. If you wish to set and maintain the performance of your project, assigning logic to a specific logic cell within a chip can minimize timing delays. In a project that is partitioned among multiple devices, you can assign logic functions that must be kept together in the same device to a chip. Chip assignments allow you to split a project so that only a minimum number of signals travel between devices, and to ensure that no unnecessary device-to-device delays exist on critical timing paths. You can assign a chip to a device in some EDA tools or in the MAX+PLUS® II software.

Use the following syntax for chip, pin, and logic cell assignments:

- To assign a logic function to a chip:

  `CHIP_PIN_LC=<`*chip name*`>`

  For example: `CHIP_PIN_LC=chip1`

- To assign a pin number within a chip:

CHIP_PIN_LC=<*chip name*>@<*pin number*>

For example: CHIP_PIN_LC=chip1@K2

- To assign a logic cell, I/O cell, or embedded cell number:

  CHIP_PIN_LC=<*chip name*>@LC<*logic cell number*>

  CHIP_PIN_LC=<*chip name*>@IOC<*I/O cell number*>

  CHIP_PIN_LC=<*chip name*>@EC<*embedded cell number*>

  For example: CHIP_PIN_LC=chip1@LC44

Related Topics:

- Go to "Devices & Adapters" and "Assigning a Device" in MAX+PLUS II Help for information on device pin-outs and assigning devices, respectively, in the MAX+PLUS II software.
- Go to Back-Annotating MAX+PLUS II Pin Assignments to Design Architect Symbols for information on back-annotating pin assignments in Mentor Graphics Design Architect schematics.

Assigning Cliques

You can define a group of logic functions as a single, named unit, called a clique. The MAX+PLUS® II Compiler attempts to place all logic in the clique in the same logic array block (LAB) to ensure optimum speed. If the project does not use multi-LAB devices, or if it is not possible to fit all clique members into a single LAB, the clique assignment ensures that all members of a clique are placed in the same device. In FLEX® 6000, FLEX 8000, FLEX 10K, and MAX® 9000 devices the Compiler also attempts to place the logic in LABs in the same row. Cliques therefore allow you to partition a project so that only a minimum number of signals travel between LABs, and to ensure that no unnecessary LAB-to-LAB or device-to-device delays exist on critical timing paths.

To assign a clique, use the following syntax:

✔ CLIQUE=<*clique name*>

For example: CLIQUE=fast1

Related Topics:

- Assigning a Clique
- Guidelines for Achieving Maximum Speed Performance

Assigning Logic Options

Logic option and logic synthesis style assignments allow you to guide logic synthesis with logic optimization features that are specific to Altera® devices. You can assign logic options and styles to individual logic functions in your design. The MAX+PLUS® II Compiler also uses a device-family-specific default logic synthesis style for each project.

Go to "Resource Assignments in EDIF Input Files" and "Assigning Resources in a Third-Party Design Editor" in

MAX+PLUS II Help for complete and up-to-date information on logic option and logic synthesis style assignments, including definitions and syntax of these assignments.

Modifying the Assignment & Configuration File with the setacf Utility

Altera provides the **setacf** utility to help you modify a project's Assignment & Configuration File (**.acf**) from the command line, without opening the file with a text editor. Type `setacf -h` ↵ at a UNIX or DOS prompt to get help on this utility.

Performing a Functional Simulation of a Concept Schematic with the hdlconfig Utility & Verilog-XL Software

You can perform a functional simulation of a Concept schematic with the **hdlconfig** utility and Verilog-XL software before compiling your project with the MAX+PLUS® II software.

To functionally simulate a Concept schematic, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment.
2. Create a Concept schematic and save it in your working directory, as described in Creating Concept Schematics for Use with MAX+PLUS II Software.
3. Use the **hdlconfig** utility to create a Verilog HDL text file that contains the entire design. Type the following command at the UNIX prompt from the /<*working directory*>/<*design name*>/**source** directory:

   `hdlconfig -a -c -r` <*design name*> `-o` <*design name*>`.v logic verilog_lib` ↵
4. If your design contains RAM or ROM functions (e.g., `lpm_ram_dq`, `lpm_ram_io`, `lpm_rom`, `scfifo`, `dcfifo`, `altdpram`, and `csdpram`), run the **vconfig** utility to link the object **convert_hex2ver.o** to build a new Verilog-XL file that supports these functions by following these steps:
   1. Create a copy of the Verilog executable file by typing the following command at the UNIX prompt:

      `cp -p $CDS_INST_DIR/tools/verilog/bin/verilog $CDS_INST_DIR/tools /verilog/bin/ verilog.bak.` ↵
   2. Type `vconfig` ↵ at the UNIX prompt from the **/usr/maxplus2/cadence/bin** directory to start the script.
   3. Accept `cr_vlog` as the name of the output script.
   4. Accept `1` as the stand-alone target.
   5. Type `new_verilog` as the name for the Verilog-XL target.
   6. Respond `Yes` when you are prompted to compile for the Verilog-XL environment.
   7. Respond `No` when you are prompted to include the Dynamic LAI, STATIC LOGIC AUTOMATION, LMSI HARDWARE MODELER, Verilog Mixed-Signal, and CDC interfaces in this executable.
   8. Respond `Yes` when you are prompted to include the Standard Delay File Annotator (SDF).
   9. Specify **/usr/maxplus2/verilog/veriuser.c** when you are asked the name of the user template file. For

more information about the contents of the **veriuser.c** file, you can refer to the **veriuser.doc** file, which is available in the Cadence *Openbook* product documentation. To locate this document, start *Openbook*, and choose **Alphabetical List of Products** from the main menu. Scroll through the pages until you locate the *PLI 1.0 User Guide & Reference* in the PLI section, and then continue to scroll through the document until you locate the **veriuser.doc** file under "Section A" and "PLI Code Examples."

10. When you are asked the name of files to be linked with the Verilog-XL simulator, specify the hexadecimal (Intel-format) conversion file **/usr/maxplus2/cadence/share/verilog/convert_hex2ver.o**, followed by a single period (.).

11. Run the output script **cr_vlog** to build the new Verilog-XL executable in the **/usr/maxplus2/cadence/bin** directory. Make sure that the `$CDS_INST_DIR/tools/bin` path appears at the beginning of the `PATH` statement in the **.cshrc** file.

12. If your C language library installation is different from the default location **/usr/lang/SC3.0.1**, type the following command at the UNIX prompt:

    `setenv C_DIR` <*C language library installation directory*> ↵

13. If successful, replace the old Verilog executable file with the new one by typing the following command at the UNIX prompt:

    `cp -p new_verilog $CDS_INST_DIR/tools/verilog/bin/verilog` ↵

5. Generate the stimulus file for the design and start the Verilog-XL simulator by typing the following command at the UNIX prompt from the /<*working directory*>/<*design name*>/**source** directory:

   `verilog -y /usr/maxplus2/simlib/concept/alt_max2/verilogUdps +libext+.v+.V` <*stimulus file name*> <*design name*>`.v` ↵

6. When you are ready to compile the project, generate an EDIF netlist file <*design name*>**.edf** with the **concept2alt** utility, as described in Converting Concept Schematics into MAX+PLUS II-Compatible EDIF Netlist Files with the **concept2alt** Utility.

7. Process the <*design name*>**.edf** file with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Performing a Functional Simulation of a Concept Schematic with VerilogLink & Verilog-XL Software

You can perform a functional simulation of a Concept schematic with VerilogLink and Verilog-XL software before compiling your project with the MAX+PLUS® II software.

To functionally simulate a Concept schematic, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment.

2. Create a Concept schematic and save it in your working directory, as described in Creating Concept Schematics for Use with MAX+PLUS II Software.

3. Generate the **global.cmd**, **vloglink.cmd**, **verilog.cmd**, and **expansion.dat** directive files.

4. Type `vloglink` <*design name*> ↵ from the /<*working directory*>/**source** directory to create a **vloglink.v** file from the Concept schematic.

5. Generate the stimulus file for the design and start the Verilog-XL simulator by typing the following command at the UNIX prompt from the /<*working directory*>/<*design name*>/**source** directory:

   `verilog -y /usr/maxplus2/simlib/concept/alt_max2/verilogUdps +libext+.v+.V` <*stimulus file name*> `vloglink.v` ↵

6. When you are ready to compile the project, generate an EDIF netlist file <*design name*>**.edf** with the **concept2alt** utility, as described in Converting Concept Schematics into MAX+PLUS II-Compatible EDIF Netlist Files with the **concept2alt** Utility .

7. Process the <*design name*>**.edf** file with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Creating Hierarchical Projects in Concept Schematics

If you wish to create a hierarchical design that contains symbols representing other MAX+PLUS II-supported design files, such as Altera® Hardware Description Language (AHDL) Text Design Files (**.tdf**), you can create a hollow-body symbol that represents a design file and then instantiate it in your Concept schematic. To create a hierarchical project in your Concept schematic, go through the following steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS® II/Cadence Working Environment.

2. Create a Concept schematic and save it in your working directory, as described in Creating Concept Schematics for Use with MAX+PLUS II Software.

   ☞ You can instantiate MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.

3. Create the hollow-body symbol <*design name*> in Concept by typing the following command from the <*working directory*>/**source** directory that contains the lower-level design file <*design name*>.<*extension*>:

   `concept` <*design name*>`.body` ↵

4. Create a part file to indicate that the body is hollow:

   1. Add the `DEFINE` and `DRAWING` bodies to the part drawing. These bodies should be the only two bodies in the drawing.

   2. Add the `TITLE=`<*design name*> and the `ABBREV=`<*design name*> properties to the `DRAWING` body to identify the drawing.

   3. Save the part drawing with the name <*design name*>**.part.1.1**.

5. Regardless of the hardware description language (HDL) or schematic editor used to create the design, you must create a dummy Verilog HDL module to indicate to the **concept2alt** utility that the design is a "black box" that must pass untouched through the EDIF netlist file.

   1. Type `genview verilog` ↵ in the Concept window.

   2. Type `logic` ↵ when prompted for the Verilog *View* name.

3. If you are using VerilogLink, you must type `genview verilog` again, then type `verilog_lib` ↵ when prompted for the Verilog *View* name.

4. Type `cd` <*design name*>`/logic` ↵ at the UNIX prompt from the **/source** directory to change to the **/source/**<*design name*>**/logic** directory.

5. Edit the **verilog.v** file to add the `cds_action = "ignore"` parameter setting after the Input Declarations and Output Declarations sections. This parameter setting specifies that the <*design name*> should be treated as a "black box."

6. To enter the symbol in the higher-level Concept schematic, choose the **Add Part** button, choose the name of the working **SCALD** directory, then choose the <*design name*> symbol from the Symbol menu.

7. The MAX+PLUS II software uses the **cadence.lmf** Library Mapping File to map Concept symbols to equivalent MAX+PLUS II logic functions. To use custom symbols, you must create a custom LMF that maps your custom symbols to the equivalent EDIF Input File, Text Design File (TDF), or other design file. You will also need to specify this custom LMF in the **EDIF Netlist Reader Settings** dialog box before compiling with the MAX+PLUS II software. See Compiling Projects with MAX+PLUS II Software for more information.

8. Continue with the steps necessary to complete your Concept schematic, as described in Creating Concept Schematics for Use with MAX+PLUS II Software.

   Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample hierarchical AHDL and Concept schematic file:

   - **/usr/maxplus2/examples/cadence/example4/fulladd2**

Converting Concept Schematics into MAX+PLUS II- Compatible EDIF Netlist Files with the concept2alt Utility

You can use the **concept2alt** utility to generate an EDIF netlist file from a Concept schematic. This file can then be imported into the MAX+PLUS® II software as an EDIF Input File (**.edf**).

To convert a Concept schematic into an EDIF netlist file, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment.

2. Create a Concept schematic and save it in your working directory, as described in Creating Concept Schematics for Use with MAX+PLUS II Software.

   You can instantiate MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.

3. Type the following command at the UNIX prompt from the **/source** directory that contains the schematic:

   `concept2alt -rundir ../max2` <*design name*> ↵

   If your design uses library of parameterized modules (LPM) functions, you must also include the `-family` option. For example:

`concept2 alt -family FLEX10K -rundirmax2` *<design name>* ↵

4. Process the *<design name>*.**edf** file with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Creating Composer Schematics for Use with MAX+PLUS II Software

You can create Composer schematics and convert them into EDIF Input Files (**.edf**) that can be processed with the MAX+PLUS® II Compiler. To create a Composer schematic for use with the MAX+PLUS II software, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment.

2. Start the Composer schematic editor from the *<working directory>* by typing `icds` ↵ at a UNIX prompt. Use the graphical user interface to structure and organize your files to create an environment that facilitates entering and processing designs. Go to Composer Project File Directory Structure for more information on directories in Composer.

3. Choose **Library Path Editor** (Tools menu) to create the *<design name>* library. In the **Library** dialog box, type *<project directory name>* as the *Library* name and `./source/`*<design name>* as the *Path* name. Choose **Save** (File menu), then choose **Exit** (File menu) to save the path.

4. Choose **Library Manager** (Tools menu) to start Composer and create a new design.

5. Type *<project directory name>* as the *Library* name, *<design name>* as the *Cell* name, and `schematic` as the *View* name in the **Library Manager** dialog box and press the ↵ key.

6. Enter primitives, megafunctions, and macrofunctions from the following libraries:

    - MAX+PLUS II-compatible primitives, megafunctions, and macrofunctions are available in the Altera-provided **alt_max2** component library.

    - Input, output, and bidirectional pins are available in the Cadence **basic** library located under **/cadence/etc/cdslib**.

    - MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.

    If you wish to create a hierarchical design that contains symbols representing other design files, such as Altera® Hardware Description Language (AHDL) Text Design Files, go to Creating Hierarchical Projects in Composer Schematics.

7. Enter meaningful instance names for all symbols and functions so that you can easily trace internal node names during simulation and debugging operations. For example, if an `a161` macrofunction is instantiated several times in one design, you should define a unique name for each instance. The instance name for each symbol is controlled by `INST` property. For more information on assigning properties, refer to the Cadence ***Composer User Guide***.

8. (Optional) To enter resource assignments in your Composer schematic, go to Entering Resource Assignments. You can also enter resource assignments from within the MAX+PLUS II software.

9. (Optional) Functionally simulate the design with the Verilog-XL simulator. Altera provides Verilog HDL

simulation modules in the **/usr/maxplus2/simlib/composer/alt_max2/verilog** and **/usr/maxplus2/simlib/composer/alt_max2/verilogUdps** directories. Go to Performing a Functional Simulation of a Composer Schematic with Verilog-XL Software for more information.

10. Use the **altout** utility to generate an EDIF netlist file that can be imported into the MAX+PLUS II software, as described in Converting Composer Schematics into MAX+PLUS II-Compatible EDIF Netlist Files with the **altout** Utility.

11. Process the <*design name*>**.edf** file with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample Composer schematic files:

- **/usr/maxplus2/examples/cadence/example2/fulladd**
- **/usr/maxplus2/examples/cadence/example5/fulladd2**
- **/usr/maxplus2/examples/cadence/example7/fa2**

Entering Resource Assignments

The MAX+PLUS® II software allows you to enter a variety of resource and device assignments for your projects. Resource assignments are used to assign logic functions to a particular pin, logic cell, I/O cell, embedded cell, row, column, Logic Array Block (LAB), Embedded Array Block (EAB), chip, clique, local routing, logic option, timing requirement, or connected pin group. In MAX+PLUS II software, you can enter all types of resource and device assignments with Assign menu commands. You can also enter pin, logic cell, I/O cell, embedded cell, LAB, EAB, row, and column assignments in the MAX+PLUS II Floorplan Editor. The Assign menu commands and the Floorplan Editor all save assignment information in the ASCII Assignment & Configuration File (**.acf**) for the project. In addition, you can edit ACFs manually in any standard text editor or with the **setacf** utility.

Concept & Composer Schematics

In both Concept and Composer schematics, you can assign a limited subset of these resource assignments by assigning properties to symbols. These properties are incorporated into the EDIF netlist file(s). The MAX+PLUS II software automatically converts assignment information from the EDIF Input File into the ACF format. For information on making MAX+PLUS II-compatible resource assignments, go to the following topics:

- Assigning Pins, Logic Cells & Chips
- Assigning Cliques
- Assigning Logic Options
- Modifying the Assignment & Configuration File with the **setacf** Utility

Go to the ***Cadence Concept Schematic User Guide*** and ***Composer Reference User Guide*** for details on how to assign properties. Go to "Resource Assignments in EDIF Input Files" and "Assigning Resources in a Third-Party Design Editor" in MAX+PLUS II Help for more information on assignments or properties that can be assigned in Concept and Composer.

Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample Concept and Composer schematic files, which include resource assignments:

- **/usr/maxplus2/examples/cadence/example6/fa2** (Concept)
- **/usr/maxplus2/examples/cadence/example7/fa2** (Composer)

VHDL & Verilog HDL Design Files

For Verilog HDL- and VHDL-based designs, you must use the MAX+PLUS II software or the **setacf** utility to

enter resource assignments. For information on using the **setacf** utility, go to Modifying the Assignment & Configuration File with the **setacf** Utility.

For information on entering assignments in the MAX+PLUS II software with Assign menu commands or in an ACF, go to "resource assignments" or "ACF, format" in MAX+PLUS II Help using **Search for Help on** (Help menu).

Assigning Pins, Logic Cells & Chips

You can assign a single logic function to a specific pin or logic cell (including I/O cells and embedded cells) within a chip, and assign one or more functions to a specific chip. A chip is a group of logic functions defined as a single, named unit, which can be assigned to a specific device.

You can assign a signal to a particular pin to ensure that the signal is always associated with that pin, regardless of future changes to the project. If you wish to set and maintain the performance of your project, assigning logic to a specific logic cell within a chip can minimize timing delays. In a project that is partitioned among multiple devices, you can assign logic functions that must be kept together in the same device to a chip. Chip assignments allow you to split a project so that only a minimum number of signals travel between devices, and to ensure that no unnecessary device-to-device delays exist on critical timing paths. You can assign a chip to a device in some EDA tools or in the MAX+PLUS® II software.

Use the following syntax for chip, pin, and logic cell assignments:

- To assign a logic function to a chip:

  `CHIP_PIN_LC=<`*chip name*`>`

  For example: `CHIP_PIN_LC=chip1`

- To assign a pin number within a chip:

  `CHIP_PIN_LC=<`*chip name*`>@<`*pin number*`>`

  For example: `CHIP_PIN_LC=chip1@K2`

- To assign a logic cell, I/O cell, or embedded cell number:

  `CHIP_PIN_LC=<`*chip name*`>@LC<`*logic cell number*`>`

  `CHIP_PIN_LC=<`*chip name*`>@IOC<`*I/O cell number*`>`

  `CHIP_PIN_LC=<`*chip name*`>@EC<`*embedded cell number*`>`

  For example: `CHIP_PIN_LC=chip1@LC44`

Related Topics:

Refer to the following sources for additional information:

- Go to "Devices & Adapters" and "Assigning a Device" in MAX+PLUS II Help for information on device pin-outs and assigning devices, respectively, in the MAX+PLUS II software.
- Go to Back-Annotating MAX+PLUS II Pin Assignments to Design Architect Symbols for information on back-annotating pin assignments in Mentor Graphics Design Architect schematics.

Assigning Cliques

You can define a group of logic functions as a single, named unit, called a clique. The MAX+PLUS® II Compiler attempts to place all logic in the clique in the same logic array block (LAB) to ensure optimum speed. If the project does not use multi-LAB devices, or if it is not possible to fit all clique members into a single LAB, the clique assignment ensures that all members of a clique are placed in the same device. In FLEX® 6000, FLEX 8000, FLEX 10K, and MAX® 9000 devices the Compiler also attempts to place the logic in LABs in the same row. Cliques therefore allow you to partition a project so that only a minimum number of signals travel between LABs, and to ensure that no unnecessary LAB-to-LAB or device-to-device delays exist on critical timing paths.

To assign a clique, use the following syntax:

✔ `CLIQUE=`<*clique name*>

For example: `CLIQUE=fast1`

Related Topics:

Go to the following topics in MAX+PLUS II Help for related information:

- Assigning a Clique
- Guidelines for Achieving Maximum Speed Performance

Assigning Logic Options

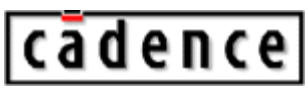
Logic option and logic synthesis style assignments allow you to guide logic synthesis with logic optimization features that are specific to Altera® devices. You can assign logic options and styles to individual logic functions in your design. The MAX+PLUS® II Compiler also uses a device-family-specific default logic synthesis style for each project.

Related Topics:

Go to "Resource Assignments in EDIF Input Files" and "Assigning Resources in a Third-Party Design Editor" in MAX+PLUS II Help for complete and up-to-date information on logic option and logic synthesis style assignments, including definitions and syntax of these assignments.

version 97A:	VerilogLink
Concept	Synergy
Composer	HDL Direct (Concept 2.0 or later)
ValidCOMPILER	Non-Graphic Simulation Environment (SE)
concept2alt	RapidSIM, Verilog-XL, or Leapfrog
vlog2alt	**redifnet** (SunOS only)
altout	

MAX+PLUS II
version 9.4



Description
Name

Description
MAX 7000A, MAX 7000AE, MAX 7000B, MAX 7000S MAX 9000 & MAX 9000A Devices

./lmf Contains the Altera-provided Library Mapping File, **cadence.lmf**, that maps Cadence logic functions to equivalent MAX+PLUS II logic functions.

Modifying the Assignment & Configuration File with the setacf Utility

Altera provides the **setacf** utility to help you modify a project's Assignment & Configuration File (**.acf**) from the command line, without opening the file with a text editor. Type `setacf -h` ↵ at a UNIX or DOS prompt to get help on this utility.

Performing a Functional Simulation of a Composer Schematic with Verilog-XL Software

You can perform a functional simulation of a Cadence Composer schematic with the Verilog-XL simulator before compiling your project with the MAX+PLUS® II software.

To functionally simulate a Composer schematic, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment.
2. Create a Composer schematic and save it in your working directory, as described in Creating Composer Schematics for Use with MAX+PLUS II Software.
3. In Composer, select **Simulation** from the *Tools* drop-down list.
4. Select **Verilog-XL** to start the *Verilog-XL Integration Control* window.
5. When you are ready to compile the project, generate an EDIF netlist file <*design name*>**.edf** with the **altout** utility, as described in Converting Composer Schematics into MAX+PLUS II-Compatible EDIF Netlist Files with the **altout** Utility.
6. Process the <*design name*>**.edf** file with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Creating Hierarchical Projects in Composer Schematics

If you wish to create a hierarchical design that contains symbols representing other design files, such as Altera® Hardware Description Language (AHDL) Text Design Files (**.tdf**), you can create a hollow-body symbol that represents a design file and then instantiate it in your Composer schematic.

To create a hierarchical project in your Composer schematic, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment.
2. Create a Composer schematic and save it in your working directory, as described in Creating Composer Schematics for Use with MAX+PLUS II Software.

You can instantiate MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.

3. Create the hollow-body symbol <*design name*> in Composer by typing `icds` ↵ from the <*working directory*> at the UNIX prompt.
4. Choose **Library Path Editor** (Tools menu) to create the <*design name*> library. Type <*design name*> as the *Library* name and `./source/`<*design name*> as the *Path* name. Choose **Save** (File menu), then choose **Exit** (File menu) to save the path and exit from the Library Path Editor.
5. Choose **Library Manager** (Tools menu) to start Composer and create a symbol for your design. Type <*project directory name*> as the *Library* name, <*lower-level design name*> as the *Cell* name, `symbol` as the *View* name, and then press ↵.
6. Create a hollow-body symbol that represents the inputs and outputs of your design.
7. Enter input and output pins for the symbol.
8. Save the symbol.
9. To enter the symbol in your higher-level schematic design, choose the **Component** button and type <*project directory name*> as the *Library* name, <*lower-level design name*> as the *Cell* name, and `symbol` as the *View* name.
10. The MAX+PLUS II software uses the **cadence.lmf** Library Mapping File to map Concept symbols to equivalent MAX+PLUS II logic functions. To use custom symbols, you must create a custom LMF that maps your custom symbols to the equivalent EDIF Input File, Text Design File (TDF), or other design file. You will also need to specify this custom LMF in the **EDIF Netlist Reader Settings** dialog box before compiling with the MAX+PLUS II software. See Compiling Projects with MAX+PLUS II Software for more information.
11. Continue with the steps necessary to complete your Composer schematic, as described in Creating Composer Schematics for Use with MAX+PLUS II Software.

    Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample hierarchical AHDL and Composer schematic file:

    - **/usr/maxplus2/examples/cadence/example5/fulladd2**

Converting Composer Schematics into MAX+PLUS II- Compatible EDIF Netlist Files with the altout Utility

You can use the **altout** utility to generate an EDIF netlist file from a Composer schematic. This file can then be imported into the MAX+PLUS® II software as an EDIF Input File (**.edf**).

To convert Composer schematics into MAX+PLUS II-compatible EDIF netlist files, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment.
2. Create a Composer schematic and save it in your working directory, as described in Creating Composer Schematics for Use with MAX+PLUS II Software.

You can instantiate MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.

3. Type the following command at the UNIX prompt from the working directory that contains the schematic:

   `altout -lib` <*design name*> `-rundir max2` <*design name*> ↵

4. Process the <*design name*>**.edf** file with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Creating VHDL Designs for Use with MAX+PLUS II Software

You can create VHDL design files with the MAX+PLUS® II Text Editor or another standard text editor and save them in the appropriate directory for your project. The MAX+PLUS II Text Editor offers the following advantages:

- VHDL templates are available with the **VHDL Templates** command (Templates menu). These templates are also available in the ASCII **vhdl.tmp** file, which is located in the **/usr/maxplus2** directory.
- If you use the MAX+PLUS II Text Editor to create your VHDL design, you can use the **Syntax Coloring** command (Options menu). The Syntax Coloring feature displays keywords and other elements in text files in different colors to distinguish them from other forms of syntax.

To create a VHDL design that can be synthesized and optimized with Synergy software, follow these steps:

1. You can instantiate the following MAX+PLUS II-provided logic functions in your VHDL design:
   - The **alt_mf** library contains the Altera® VHDL logic function library, which includes the `a_8count`, `a_8mcomp`, `a_8fadd`, and `a_81mux` macrofunctions. If you wish to instantiate **alt_mf** logic functions in your VHDL design, you must first compile this library, as described in Compiling the **alt_mf** Library.
   - The `clklock` megafunction, which enables the phase-locked loop, or ClockLock , circuitry available on selected Altera FLEX® 10K devices. Go to Instantiating the `clklock` Megafunction in VHDL or Verilog HDL for information.
   - MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.
2. If you wish to use Standard Delay Format (SDF) Output Files (**.sdo**) that contain timing information when performing post-compilation timing simulation with Leapfrog software, you must first compile the VITAL library source files, as described in Compiling the **alt_vtl** Library for for Use with Leapfrog Software.
3. (Optional) To enter resource assignments in your VHDL design, go to Entering Resource Assignments. You

can also enter resource assignments from within the MAX+PLUS II software.

4. After you have completed your VHDL design, synthesize and optimize it with Synergy software, as described in Synthesizing & Optimizing VHDL Files with Synergy Software.

Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample VHDL files, the latter of which includes macrofunction instantiation.

- **/usr/maxplus2/examples/cadence/example9/count4.vhd**
- **/usr/maxplus2/examples/cadence/example10/adder16.vhd**

Related Topics:

Go to FLEX 10K Device Family, which is available on the web, for additional information.

Instantiating the clklock Megafunction in VHDL or Verilog HDL

MAX+PLUS® II interfaces to other EDA tools support the `clklock` phase-locked loop megafunction, which can be used with some FLEX® 10K devices, with the **gencklk** utility, which is available in the MAX+PLUS II system directory. Type `gencklk -h` ↵ at the DOS or UNIX prompt to display information on how to use this utility. The **gencklk** utility generates VHDL or Verilog HDL functional simulation models and a VHDL Component Declaration template file (**.cmp**).

The **gencklk** utility allows parameters for the `clklock` function to be passed from the VHDL or Verilog HDL file to EDIF netlist format. The **gencklk** utility embeds the parameter values in the `clklock` function name; therefore, the values do not need to be declared explicitly.

To instantiate the `clklock` megafunction in VHDL or Verilog HDL, go through the following steps:

1. Type the following command at the DOS or UNIX prompt to generate the `clklock_`*x*`_`*y* function, where *x* is the ClockBoost value and *y* is the input frequency in MHz:

   ✔ Type `gencklk` <*ClockBoost*> <*input frequency*> `-vhdl` ↵ for VHDL designs.

   or:

   ✔ Type `gencklk` <*ClockBoost*> <*input frequency*> `-verilog` ↵ for Verilog HDL designs.

   Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu for more information on the `clklock` megafunction.

2. Create a design file that instantiates the `clklock_`*x*`_`*y***.vhd** or `clklock_`*x*`_`*y***.v** file. The **gencklk** utility automatically generates a VHDL Component Declaration template in the **clklock_*x*_*y*.cmp** file that you can incorporate into a VHDL design file.

☞ In MAX+PLUS II version 8.3 and lower, running `genclklk` on a PC always creates files named as **clklock.vhd**, **clklock.cmp**, and **clklock.v**, regardless of the ClockBoost and input frequency values you specify.

Figures 1 and 2 show a `clklock` function with <*ClockBoost*> = 2 and <*input frequency*> = 40 MHz instantiated in VHDL and Verilog HDL design files, respectively.

Figure 1. VHDL Design File with clklock Instantiation (count8.vhd)

```
LIBRARY ieee;
```

```
USE ieee.std_logic_1164.all;
LIBRARY altera;
USE altera.maxplus2.all;     --  Include Altera Component Declarations



ENTITY count8 IS
   PORT (a    : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
         ldn  : IN STD_LOGIC;
         gn   : IN STD_LOGIC;



dnup : IN STD_LOGIC;
         setn : IN STD_LOGIC;
         clrn : IN STD_LOGIC;
         clk  : IN STD_LOGIC;



co   : OUT STD_LOGIC;
         q    : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END count8;



ARCHITECTURE structure OF count8 IS
   signal clk2x : STD_LOGIC;



COMPONENT clklock_2_40
   PORT (
      INCLK : IN STD_LOGIC;
      OUTCLK : OUT STD_LOGIC
   );
END COMPONENT;



BEGIN
   u1: clklock_2_40
   PORT MAP (inclk=>clk, outclk=>clk2x);



u2: a_8count
   PORT MAP (a=>a(0), b=>a(1), c=>a(2), d=>a(3),
             e=>a(4), f=>a(5), g=>a(6), h=>a(7),
             clk=>clk2x,
             ldn=>ldn,
             gn=>gn,



dnup=>dnup,
             setn=>setn,
             clrn=>clrn,



qa=>q(0), qb=>q(1), qc=>q(2), qd=>q(3),
             qe=>q(4), qf=>q(5), qg=>q(6), qh=>q(7),
             cout=>co);
 END structure;
```

Figure 2. Verilog HDL Design File with clklock Instantiation (count8.v)

```
`timescale 1ns / 10ps
module count8 (a, ldn, gn, dnup, setn, clrn, clk, co, q);
output          co;
output[7:0]     q;
```

```
input[7:0]      a;
input           ldn, gn,dnup, setn, clrn, clk;
wire            clk2x;



clklock_2_40 u1 (.inclk(clk), .outclk(clk2x) );
A_8COUNT u2 (.A(a[0]), .B(a[1]), .C(a[2]), .D(a[3]), .E(a[4]), .F(a[5]),



.G(a[6]), .H(a[7]), .LDN(ldn), .GN(gn), .DNUP(dnup),
             .SETN(setn), .CLRN(clrn), .CLK(clk2x), .QA(q[0]), .QB(q[1]),
             .QC(q[2]), .QD(q[3]), .QE(q[4]), .QF(q[5]), .QG(q[6]),
             .QH(q[7]), .COUT(co) );



endmodule
```

Related Topics:

Go to FLEX 10K Device Family, which is available on the web, for additional information.

Entering Resource Assignments

The MAX+PLUS® II software allows you to enter a variety of resource and device assignments for your projects. Resource assignments are used to assign logic functions to a particular pin, logic cell, I/O cell, embedded cell, row, column, Logic Array Block (LAB), Embedded Array Block (EAB), chip, clique, local routing, logic option, timing requirement, or connected pin group. In MAX+PLUS II software, you can enter all types of resource and device assignments with Assign menu commands. You can also enter pin, logic cell, I/O cell, embedded cell, LAB, EAB, row, and column assignments in the MAX+PLUS II Floorplan Editor. The Assign menu commands and the Floorplan Editor all save assignment information in the ASCII Assignment & Configuration File (**.acf**) for the project. In addition, you can edit ACFs manually in any standard text editor or with the **setacf** utility.

Concept & Composer Schematics

In both Concept and Composer schematics, you can assign a limited subset of these resource assignments by assigning properties to symbols. These properties are incorporated into the EDIF netlist file(s). The MAX+PLUS II software automatically converts assignment information from the EDIF Input File into the ACF format. For information on making MAX+PLUS II-compatible resource assignments, go to the following topics:

- Assigning Pins, Logic Cells & Chips
- Assigning Cliques
- Assigning Logic Options
- Modifying the Assignment & Configuration File with the **setacf** Utility

Go to the ***Cadence Concept Schematic User Guide*** and ***Composer Reference User Guide*** for details on how to assign properties. Go to "Resource Assignments in EDIF Input Files" and "Assigning Resources in a Third-Party Design Editor" in MAX+PLUS II Help for more information on assignments or properties that can be assigned in Concept and Composer.

Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample Concept and Composer schematic files, which include resource assignments:

- **/usr/maxplus2/examples/cadence/example6/fa2** (Concept)

- **/usr/maxplus2/examples/cadence/example7/fa2** (Composer)

VHDL & Verilog HDL Design Files

For Verilog HDL- and VHDL-based designs, you must use the MAX+PLUS II software or the **setacf** utility to enter resource assignments. For information on using the **setacf** utility, go to Modifying the Assignment & Configuration File with the **setacf** Utility.

For information on entering assignments in the MAX+PLUS II software with Assign menu commands or in an ACF, go to "resource assignments" or "ACF, format" in MAX+PLUS II Help using **Search for Help on** (Help menu).

./examples/cadence Contains the sample files for Cadence software discussed in these ACCESS^{SM} Key Guidelines.
./cadence Contains the AMPLE userware for the MAX+PLUS II/Cadence interface.
FLEX® 6000,
FLEX 6000A,
FLEX 8000,
FLEX 10K,
FLEX 10KA,
FLEX 10KB,
&
FLEX 10KE Devices
In-System
Programming/
Configuration
./simlib/concept/alt_max2 Contains the MAX+PLUS II primitives, including `CARRY`, `CASCADE`, `EXP`, `GLOBAL`, `LCELL`, `SOFT`, `OPNDRN`, `DFFE` (D flipflop with Clock Enable), and `DFFE6K` (D flipflop with Clock Enable and both Clear and Preset for FLEX® 6000 devices only) for use with Concept software.

Modifying the Assignment & Configuration File with the setacf Utility

Altera provides the **setacf** utility to help you modify a project's Assignment & Configuration File (**.acf**) from the command line, without opening the file with a text editor. Type `setacf -h` ↵ at a UNIX or DOS prompt to get help on this utility.

Creating Verilog HDL Designs for Use with MAX+PLUS II Software

You can create Verilog HDL design files with the MAX+PLUS® II Text Editor or another standard text editor and save them in the appropriate directory for you project. The MAX+PLUS II Text Editor offers the following advantages:

- Verilog HDL templates are available with the **Verilog Templates** command (Templates menu). These templates are also available in the ASCII **verilog.tmp** file, which is located in the **/usr/maxplus2** directory.
- If you use the MAX+PLUS II Text Editor to create your Verilog HDL design, you can use the **Syntax Coloring** command (Options menu). The Syntax Coloring feature displays keywords and other elements of text in text files in different colors to distinguish them from other forms of syntax.

To create a Verilog HDL design that can be synthesized and optimized with Synergy software, go through the following steps:

1. You can instantiate the following MAX+PLUS II-provided logic functions in your Verilog HDL design:

- The **alt_max2** library, which contains the `a_8count`, `a_8mcomp`, `a_8fadd`, and `a_81mux` macrofunctions that are optimized for different Altera device families.

- The `clklock` megafunction which enables phase-locked loop, or ClockLock , circuitry available on selected Altera FLEX® 10K devices. Go to Instantiating the `clklock` Megafunction in VHDL or Verilog HDL for information.

- The **lpm_syn** library, which contains the Cadence LPM megafunction library for use with Synergy Software and Concept or Composer software.

- MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.

2. You can enter resource assignments in your Verilog HDL design, as described in Entering Resource Assignments.

3. After you have completed your Verilog HDL design, synthesize and optimize it with Synergy software, as described in Synthesizing & Optimizing Verilog HDL Files with Synergy Software.

Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample Verilog HDL files, the latter of which includes LPM function instantiation.

- **/usr/maxplus2/examples/cadence/example11/count8.v**
- **/usr/maxplus2/examples/cadence/example13/rom_test.v**

Related Topics:

Go to FLEX 10K Device Family, which is available on the web, for additional information.

Instantiating the clklock Megafunction in VHDL or Verilog HDL

MAX+PLUS® II interfaces to other EDA tools support the `clklock` phase-locked loop megafunction, which can be used with some FLEX® 10K devices, with the **gencklk** utility, which is available in the MAX+PLUS II system directory. Type `gencklk -h` ↵ at the DOS or UNIX prompt to display information on how to use this utility. The **gencklk** utility generates VHDL or Verilog HDL functional simulation models and a VHDL Component Declaration template file (**.cmp**).

The **gencklk** utility allows parameters for the `clklock` function to be passed from the VHDL or Verilog HDL file to EDIF netlist format. The **gencklk** utility embeds the parameter values in the `clklock` function name; therefore, the values do not need to be declared explicitly.

To instantiate the `clklock` megafunction in VHDL or Verilog HDL, go through the following steps:

1. Type the following command at the DOS or UNIX prompt to generate the `clklock_x_y` function, where *x* is the ClockBoost value and *y* is the input frequency in MHz:

   ✔ Type `gencklk` <*ClockBoost*> <*input frequency*> `-vhdl` ↵ for VHDL designs.

   or:

   ✔ Type `gencklk` <*ClockBoost*> <*input frequency*> `-verilog` ↵ for Verilog HDL designs.

Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu for more information on the `clklock` megafunction.

2. Create a design file that instantiates the `clklock_`*x*`_`*y*`.vhd` or `clklock_`*x*`_`*y*`.v` file. The **genclklk** utility automatically generates a VHDL Component Declaration template in the **clklock_*x*_*y*.cmp** file that you can incorporate into a VHDL design file.

☞ In MAX+PLUS II version 8.3 and lower, running `genclklk` on a PC always creates files named as **clklock.vhd**, **clklock.cmp**, and **clklock.v**, regardless of the ClockBoost and input frequency values you specify.

Figures 1 and 2 show a `clklock` function with <*ClockBoost*> = 2 and <*input frequency*> = 40 MHz instantiated in VHDL and Verilog HDL design files, respectively.

Figure 1. VHDL Design File with clklock Instantiation (count8.vhd)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY altera;
USE altera.maxplus2.all;     --  Include Altera Component Declarations



ENTITY count8 IS
   PORT (a    : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
         ldn  : IN STD_LOGIC;
         gn   : IN STD_LOGIC;



dnup : IN STD_LOGIC;
         setn : IN STD_LOGIC;
         clrn : IN STD_LOGIC;
         clk  : IN STD_LOGIC;



co   : OUT STD_LOGIC;
         q    : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END count8;



ARCHITECTURE structure OF count8 IS
   signal clk2x : STD_LOGIC;



COMPONENT clklock_2_40
   PORT (
      INCLK : IN STD_LOGIC;
      OUTCLK : OUT STD_LOGIC
   );
END COMPONENT;



BEGIN
   u1: clklock_2_40
   PORT MAP (inclk=>clk, outclk=>clk2x);



u2: a_8count
   PORT MAP (a=>a(0), b=>a(1), c=>a(2), d=>a(3),
             e=>a(4), f=>a(5), g=>a(6), h=>a(7),
             clk=>clk2x,
             ldn=>ldn,
             gn=>gn,
```

```
dnup=>dnup,
             setn=>setn,
             clrn=>clrn,



qa=>q(0), qb=>q(1), qc=>q(2), qd=>q(3),
             qe=>q(4), qf=>q(5), qg=>q(6), qh=>q(7),
             cout=>co);
 END structure;
```

Figure 2. Verilog HDL Design File with clklock Instantiation (count8.v)

```
`timescale 1ns / 10ps
module count8 (a, ldn, gn, dnup, setn, clrn, clk, co, q);
output          co;
output[7:0]     q;



input[7:0]      a;
input           ldn, gn,dnup, setn, clrn, clk;
wire            clk2x;



clklock_2_40 u1 (.inclk(clk), .outclk(clk2x) );
A_8COUNT u2 (.A(a[0]), .B(a[1]), .C(a[2]), .D(a[3]), .E(a[4]), .F(a[5]),



.G(a[6]), .H(a[7]), .LDN(ldn), .GN(gn), .DNUP(dnup),
             .SETN(setn), .CLRN(clrn), .CLK(clk2x), .QA(q[0]), .QB(q[1]),
             .QC(q[2]), .QD(q[3]), .QE(q[4]), .QF(q[5]), .QG(q[6]),
             .QH(q[7]), .COUT(co) );



endmodule
```

Related Topics:

Go to FLEX 10K Device Family, which is available on the web, for additional information.

Entering Resource Assignments

The MAX+PLUS® II software allows you to enter a variety of resource and device assignments for your projects. Resource assignments are used to assign logic functions to a particular pin, logic cell, I/O cell, embedded cell, row, column, Logic Array Block (LAB), Embedded Array Block (EAB), chip, clique, local routing, logic option, timing requirement, or connected pin group. In MAX+PLUS II software, you can enter all types of resource and device assignments with Assign menu commands. You can also enter pin, logic cell, I/O cell, embedded cell, LAB, EAB, row, and column assignments in the MAX+PLUS II Floorplan Editor. The Assign menu commands and the Floorplan Editor all save assignment information in the ASCII Assignment & Configuration File (**.acf**) for the project. In addition, you can edit ACFs manually in any standard text editor or with the **setacf** utility.

Concept & Composer Schematics

In both Concept and Composer schematics, you can assign a limited subset of these resource assignments by assigning properties to symbols. These properties are incorporated into the EDIF netlist file(s). The MAX+PLUS II software automatically converts assignment information from the EDIF Input File into the ACF format. For

information on making MAX+PLUS II-compatible resource assignments, go to the following topics:

- Assigning Pins, Logic Cells & Chips
- Assigning Cliques
- Assigning Logic Options
- Modifying the Assignment & Configuration File with the **setacf** Utility

Go to the ***Cadence Concept Schematic User Guide*** and ***Composer Reference User Guide*** for details on how to assign properties. Go to "Resource Assignments in EDIF Input Files" and "Assigning Resources in a Third-Party Design Editor" in MAX+PLUS II Help for more information on assignments or properties that can be assigned in Concept and Composer.

Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample Concept and Composer schematic files, which include resource assignments:

- **/usr/maxplus2/examples/cadence/example6/fa2** (Concept)
- **/usr/maxplus2/examples/cadence/example7/fa2** (Composer)

VHDL & Verilog HDL Design Files

For Verilog HDL- and VHDL-based designs, you must use the MAX+PLUS II software or the **setacf** utility to enter resource assignments. For information on using the **setacf** utility, go to Modifying the Assignment & Configuration File with the **setacf** Utility.

Related Topics:

For information on entering assignments in the MAX+PLUS II software with Assign menu commands or in an ACF, go to "resource assignments" or "ACF, format" in MAX+PLUS II Help using **Search for Help on** (Help menu).

./simlib/composer/alt_max2 Contains the MAX+PLUS II primitives, including `CARRY`, `CASCADE`, `EXP`, `GLOBAL`, `LCELL`, `SOFT`, `OPNDRN`, `DFFE` (D flipflop with Clock Enable), and `DFFE6K` (D flipflop with Clock Enable and both Clear and Preset for FLEX 6000 devices only) for use with Composer software. **./simlib/concept/alt_lpm**Contains the MAX+PLUS II megafunctions, including library of parameterized modules (LPM) functions, for use with Concept software. **./simlib/concept/max2sim**Contains the MAX+PLUS II/Concept simulation model library, **max2_sim**, for use with RapidSIM software.

Modifying the Assignment & Configuration File with the setacf Utility

Altera provides the **setacf** utility to help you modify a project's Assignment & Configuration File (**.acf**) from the command line, without opening the file with a text editor. Type `setacf -h` ↵ at a UNIX or DOS prompt to get help on this utility.

Synthesizing & Optimizing VHDL Files with Synergy Software

You can use Cadence Synergy software to synthesize and optimize your VHDL files and convert them to EDIF input files that can be processed by the MAX+PLUS® II Compiler. The information presented here describes only how to use VHDL files that have been processed by Synergy software. For information on direct MAX+PLUS II support for VHDL Design Files, go to MAX+PLUS II VHDL Help.

To process a VHDL file with Synergy software for use with MAX+PLUS II software, go through the following steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment.

2. Create a VHDL file <*design name*>**.vhd** using the MAX+PLUS II Text Editor or another standard text editor and save it in a working directory. Go to Creating VHDL Designs for Use with MAX+PLUS II Software for more information on VHDL design entry.

3. Start Synergy by typing `synergy -lang vhdl` ↵ at a UNIX prompt from the working directory.

4. Analyze your source file <*design name*>**.vhd**:

    1. Choose **Analyze Files** (File menu) to open the **Select Design** dialog box.

    2. Click on the **Analyze Files** tab.

    3. Select the design name from the *Files* list.

    4. Choose **Analyze** to analyze the source file(s).

5. Choose the **Select Design** tab from the **Select Design** dialog box and specify the following options:

    1. Select the design architecture from the hierarchical list. The design architecture should appear in the *Design* box.

    2. Specify <*design name*>**.run1** as the *Run Directory*.

    3. Type `alt_syn` as the *Target Library* name.

    4. (Optional) If you want to use the Synergy library of parameterized modules (LPM) synthesis capability, choose the **Macro Libraries** ellipse button and select *lpm_syn* in the *Select From* box.

    5. (Optional) If you want to view a synthesized schematic in Concept or Composer, go through the following steps:

    1. Choose **Schematic Generation** (Utilities menu).

    2. Select either *Concept* or *Composer* in the *Generate From* box.

    3. Type `alt_max2` in the *Symbol Libraries* box.

    4. Choose **Apply**, then **Close**.

6. Choose the **Select Design** button from the **Select Design** window.

7. Indicate to the Synergy software that any `clklock` megafunction or any macrofunction instantiated in your VHDL design is a "black box" that must pass untouched through the EDIF netlist file:

    1. Choose **Synthesis** (Constraints menu), then choose **Hierarchy Control**.

    2. Select the module or instance name from the hierarchical **View** list for *Module/Instance*.

    3. Turn on *Maintain Option* in the *Synthesis Constraints* box.

    4. Select *Module/Instance* and *Tree Below* in the *Apply To* box.

    5. Choose **Apply**.

    6. Repeat steps a through e for each instance of the function.

8. Choose **Synthesize** (Synthesis menu) from the Synergy window and specify the following options:
    1. Click on the **Synthesize** tab.
    2. Turn on the *Generate Schematic* option.
    3. Select either *Composer* or *Concept* from the *Type* list box.
    4. Choose **Synthesize** to start synthesizing your design.
9. Generate an EDIF netlist file that can be compiled with MAX+PLUS II software, as described in Converting VHDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the **vlog2alt** or altout **Utility**.
10. Process the <*design name*>**.edf** file with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample VHDL files:

- **/usr/maxplus2/examples/cadence/example9/count4.vhd**
- **/usr/maxplus2/examples/cadence/example10/adder16.vhd**

Converting VHDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the vlog2alt or altout Utility

You can convert a VHDL design into an EDIF netlist file with the extension **.edf**. This file can then be imported into the MAX+PLUS® II software as an EDIF Input File (**.edf**).

To convert a VHDL design into an EDIF netlist file, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment.
2. Synthesize and optimize your VHDL design with Synergy, as described in Synthesizing & Optimizing VHDL Files with Synergy Software.
3. Depending on whether or not you have installed the Concept **alt_syn** library, perform one of the following steps to create <*design name*>**.edf** in the working directory:

    ✔ If you have installed the Concept **alt_syn** library, type the following command at the UNIX prompt from your working directory:

    `vlog2alt` <*design name*> `-rundir max2 -vfiles` <*design name*>`.run1/syn.v` ↵

    or:

    ✔ If you have not installed the Concept **alt_syn** library, follow these steps:

    1. Edit the **cds.lib** file, which is located in your working directory, to include the following line:

        `DEFINE Opt` <*working directory*>/<*design name*>`.run1/Opt` ↵

    2. Type the following command at the UNIX prompt from the working directory:

```
altout -lib Opt -rundir max2 <design name> ↵
```

4. Process the <*design name*>**.edf** file with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample VHDL files:

- **/usr/maxplus2/examples/cadence/example9/count4.vhd**
- **/usr/maxplus2/examples/cadence/example10/adder16.vhd**

Synthesizing & Optimizing Verilog HDL Files with Synergy Software

You can create and process Verilog HDL files and convert them into EDIF input files that can be processed by the MAX+PLUS® II Compiler. To process a Verilog HDL file with Synergy software for use with the MAX+PLUS II software, go through the following steps:

1. Be sure to set up your working environment correctly, as described in Setting up the MAX+PLUS II/Cadence Working Environment.
2. Create a Verilog HDL file <*design name*>**.v** using the MAX+PLUS II Text Editor or another standard text editor and save it in a working directory. Go to Creating Verilog HDL Designs for Use with MAX+PLUS II Software for more information on Verilog HDL design entry.
3. Start Synergy by typing `synergy -lang verilog ↵` at a UNIX prompt from your working directory.
4. Choose **Select Design** (File menu) from the Synergy window and specify the following options:
    1. Select <*design name*>**.v** from the *Verilog Files* list.
    2. Choose the **Verilog Option** tab from the **Select Design** dialog box.
    3. Specify <*design name*>.**run1** as the *Run Directory*.
    4. Type `/usr/maxplus2/simlib/concept/alt_max2/`<*design name*>`/verilog_lib/verilog.v` <*working directory*>/ in the *Library Files (-v)* box.
    5. (Optional) If your design includes library of parameterized modules (LPM) functions, type `+define+SYNTH` in the *Other Compilations* box.
    6. Choose **Select Design**.
5. Choose the **Design** tab from the **Select Design** dialog box and set the target library:
    1. Type `alt_syn` as the *Target Library* name.
    2. (Optional) To use the Synergy LPM synthesis capability, type `lpm_syn` as the *Library* name in the *Macro Cell Library* box.
    3. Choose **OK**.
6. (Optional) To view the synthesized schematic in Concept or Composer, go through the following steps:
    1. Select **Schematic Generation** (Utilities menu).
    2. Select either *Concept* or *Composer* in the *Generate From* box.

3. Type `alt_max2` in the *Symbol Libraries* box.
4. Choose **Apply**, then **Close**.

7. Choose **Select Design** from the **Select Design** window.
8. Choose **Synthesize** (Synthesis menu) from the **Synergy** window and specify the following options:
    1. Click on the **Synthesize** tab.
    2. Turn on the *Generate Schematic* option.
    3. Select either *Composer* or *Concept* from the *Type* list box.
    4. Choose **Synthesize** to start synthesizing your design.
9. Generate an EDIF netlist file that can be compiled by the MAX+PLUS II Compiler, as described in Converting Verilog HDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files.
10. Process the <*design name*>**.edf** file with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample Verilog HDL files:

- **/usr/maxplus2/examples/cadence/example11/count8.v**
- **/usr/maxplus2/examples/cadence/example13/rom_test.v**

Converting Verilog HDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the vlog2alt Utility

You can use the **vlog2alt** utility to convert your Verilog HDL design into an EDIF netlist file. This file can then be imported into the MAX+PLUS® II software as an EDIF Input File with the extension **.edf**.

To convert a Verilog HDL design into an EDIF netlist file, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment.
2. Synthesize and optimize your Verilog HDL design with Synergy, as described in Synthesizing & Optimizing Verilog HDL Files with Synergy Software.
3. To convert your Verilog HDL design into an EDIF netlist file, type the following command at the UNIX prompt from your working directory:

    `vlog2alt` <*design name*> `-rundir max2 -vfiles` <*design name*>`.run1/syn.v` ↵
4. Process the <*design name*>**.edf** file with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample Verilog HDL files:

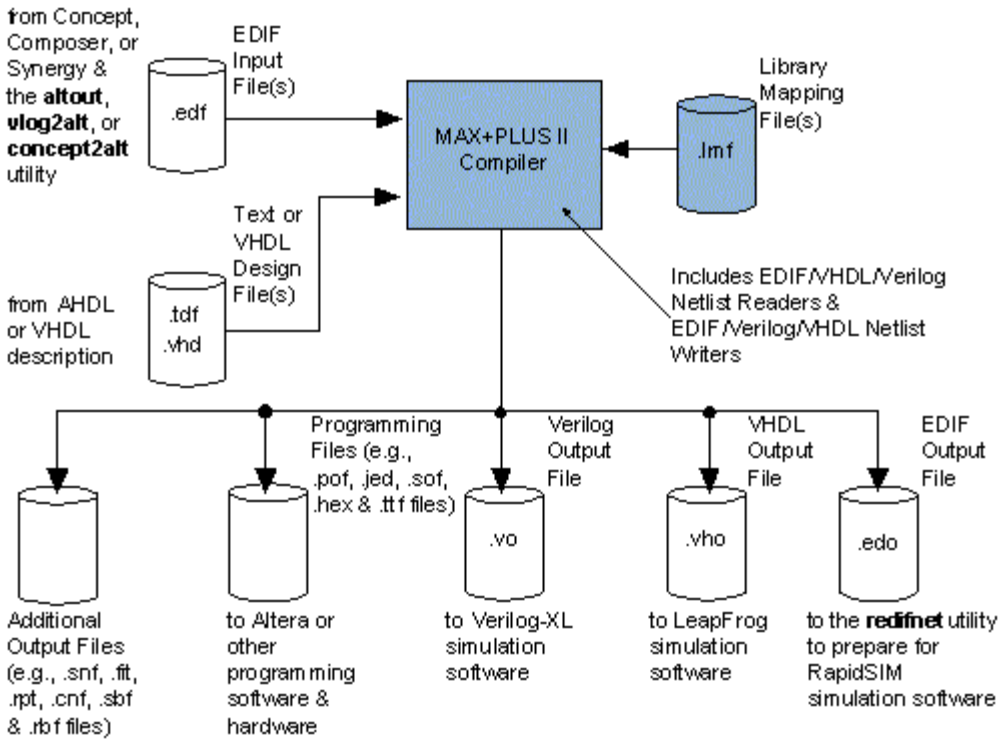
- **/usr/maxplus2/examples/cadence/example11/count8.v**
- **/usr/maxplus2/examples/cadence/example13/rom_test.v**

Project Compilation Flow

Figure 1 shows the MAX+PLUS® II/Cadence project compilation flow.

Figure 1. MAX+PLUS II/Cadence Project Compilation Flow

Altera-provided items are shown in blue.



Compiling Projects with MAX+PLUS II Software

The MAX+PLUS® II Compiler can process design files in a variety of formats. This topic describes how to use MAX+PLUS II software to compile projects in which the top-level design file is an EDIF Input File (with the extension **.edf**).

Related Topics:

Refer to the following sources for additional information:

- Go to MAX+PLUS II Help for information on compiling VHDL and Verilog HDL, design files directly with the MAX+PLUS II Compiler.
- Go to Running Synopsys Compilers from MAX+PLUS II Software for information on running the Synopsys Design Compiler or FPGA Compiler software on a VHDL or Verilog HDL design from within the MAX+PLUS II Compiler window.

To compile a design (also called a "project") with MAX+PLUS II software, go through the following steps:

1. Create design files that are compatible with the MAX+PLUS II software and convert them into EDIF Input Files with the extension **.edf**. Specific instructions for some tools are described in these MAX+PLUS II ACCESS^{SM} Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your design entry or synthesis and optimization tool.

2. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, create a mapping for each function in a Library Mapping File (**.lmf**) to map the custom symbol to the corresponding EDIF Input File, AHDL Text Design File (**.tdf**), or other MAX+PLUS II-supported design file. These custom functions are represented in design files as hollow-body symbols or "black box" HDL descriptions.

   ☞ Go to "Library Mapping Files (**.lmf**)" in MAX+PLUS II Help for more information.

3. Open MAX+PLUS II and specify the name of your top-level design file as the project name with the **Project Name** command (File menu). If you open an HDL file in the MAX+PLUS II Text Editor, you can choose the **Project Set Project to Current File** command (File menu) instead.

   ☞ You can also compile a project from a command line. However, the first time you compile a project, the settings you need to specify are easier to specify from within the MAX+PLUS II software. After you have run the graphical user interface for the MAX+PLUS II software at least once, you can more easily use the command-line **setacf** utility to modify options in the Assignment & Configuration File (**.acf**) for the project. Type `setacf -h` ↵ and `maxplus2 -h` ↵ for descriptions of **setacf** and MAX+PLUS II command-line syntax.

4. Choose **Device** (Assign menu) and select the target Altera device family in the *Device Family* drop-down list box. If you wish to implement the design logic in a specific device, select it in the *Devices* box. Otherwise, select *AUTO* to allow the MAX+PLUS II Compiler to choose the best device(s) in the current device family. If your design entry or synthesis and optimization tool required you to specify a target family and/or device, specify the same information in this dialog box. For information on partitioning logic among multiple devices, go to MAX+PLUS II Help. Choose **OK**.

5. Open the Compiler window by choosing the **Compiler** command (MAX+PLUS II menu). Go through the following steps to specify the options necessary to compile the design file(s) in your project:

   1. Ensure that all EDIF netlist files have the extension **.edf** and choose **EDIF Netlist Reader Settings** (Interfaces menu).

   2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor. This name should be the name of the vendor whose tool(s) you used to create the EDIF netlist files. If your vendor name does not appear, select *Custom* instead.

      ☞ If you are compiling a design created with Synopsys FPGA Express software, select *Synopsys*, choose the **Customize** button, enter <*project name*>**.lmf** in the *LMF #1* box, choose **OK**, and skip to step 6.

   3. If you selected an existing vendor name in the *Vendor* box and your project contains design files that require custom LMF mappings, choose the **Customize** button to expand the dialog box to show all settings. Turn on the *LMF #2* checkbox and type your custom LMF's filename in the corresponding text box, or select a name from the *Files* box. The selection in the *Vendor* box will change to *Custom* and all settings will be retained until you change them again.

   4. If you selected *Custom* in the *Vendor* box, choose the **Customize** button to expand the dialog box to show all settings. Any previously defined custom settings will be displayed. Under *Signal Names*, type one or more names with up to 20 total name characters in the *VCC* or *GND* box if your EDIF Input File(s) use one or more names other than `VCC` or `GND` for the global high or low signals. Multiple signal

names must be separated by either a comma (,) or a space. Under *Library Mapping Files*, turn on the *LMF #1* checkbox and type a filename in the text box following it, or select a name from the *Files* box. If necessary, specify another LMF name in the *LMF #2* box. Go to MAX+PLUS II Help for detailed information on the settings available in the **EDIF Netlist Reader Settings** dialog box.

5. Choose **OK**.

6. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, you may need to ensure that all files are present in your project directory, i.e., the same directory as the top-level design file. Otherwise, you must specify the directories containing these files as user libraries with the **User Libraries** command (Options menu).

7. Follow all guidelines that apply to your design entry or synthesis and optimization tool:

   - Exemplar Logic Galileo Extreme-Specific Compiler Settings
   - Synopsys DesignWare-Specific Compiler Settings
   - Converting Synopsys FPGA Compiler & Design Compiler Timing Constraints into MAX+PLUS II-Compatible Format with the **syn2acf** Utility
   - Synplicity Synplify-Specific Compiler Settings

8. If you wish to generate EDIF, VHDL, or Verilog HDL output files for post-compilation simulation or timing analysis with another EDA tool, go through the following steps:

   1. (Optional) Turn on the **Optimize Timing SNF** command (Processing menu) to reduce the size of the output file(s). Turning on this command can reduce the size of output netlists by up to 30%.

      This command does not create optimized timing SNFs on UNIX workstations. However, a non-optimized timing SNF provides the same functional and timing information as an optimized timing SNF.

   2. If you wish to generate EDIF Output Files (**.edo**), go through these steps:

      1. Turn on the **EDIF Netlist Writer** command (Interfaces menu). Then choose the **EDIF Netlist Writer Settings** command (Interfaces menu).

      2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor and choose **OK**. If your vendor name does not appear, select *Custom* instead and specify the settings that are appropriate for your simulation or timing analysis tool. Go to MAX+PLUS II Help for detailed information on the options available in the **EDIF Netlist Writer Settings** dialog box.

      3. To generate an optional Standard Delay Format (SDF) Output File (**.sdo**), choose the **Customize** button to expand the dialog box to show all settings. Select one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**.

      The filenames of the EDIF Output File(s) and optional SDF Output File(s) are the same as the user-defined chip name(s) for the project; if no chip names exist, the Compiler assigns filenames that are based on the project name. For a multi-device project, the Compiler also generates a top-level EDIF Output File that is uniquely identified by "**_t**" appended to the project name. In addition, the Compiler automatically generates a VHDL Memory Model Output File, *<project name>***.vmo**, when it generates an EDIF Output File that contains memory (RAM or ROM).

   3. If you wish to generate VHDL Output Files (**.vho**), turn on the **VHDL Netlist Writer** command (Interfaces menu). Then choose **VHDL Netlist Writer Settings** command (Interfaces menu). Select *VHDL Output File (.vho)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF ver. 2.1 files contain timing delay information that allows you to perform back-annotation simulation in VHDL with VITAL-compliant simulation libraries. The VHDL Output Files

generated by the Compiler have the extension **.vho**, but are otherwise named in the same way as the EDIF Output Files described above.

4. If you wish to generate Verilog HDL Output Files (**.vo**), turn on the **Verilog Netlist Writer** command (Interfaces menu). Then choose **Verilog Netlist Writer Settings** command (Interfaces menu). Select *Verilog Output File (.vo)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF Output Files contain timing delay information that allows you to perform back-annotation simulation in Verilog HDL. The Verilog Output Files generated by the Compiler have the extension **.vo**, but are otherwise named in the same way as the EDIF Output Files described above.

9. To run the MAX+PLUS II Compiler, choose the **Project Save & Compile** command (File menu) or choose the **Start** button in the Compiler window.

   ☞ See step 3 for information on running MAX+PLUS II software from the command line.

10. Once you have compiled the project with the MAX+PLUS II Compiler, you can use the VHDL, Verilog HDL, or EDIF output file(s), and the optional SDF Output File(s) (**.sdo**) to perform timing analysis or timing simulation with another EDA tool. Specific instructions for some tools are described in these MAX+PLUS II ACCESS Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your EDA tool.

The MAX+PLUS II Compiler also generates a Report File (**.rpt**), a Pin-Out File (**.pin**), and one or more of the following files for device programming or configuration:

- JEDEC Files (**.jed**)
- Programmer Object Files (**.pof**)
- SRAM Object Files (**.sof**)
- Hexadecimal (Intel-format) Files (**.hex**)
- Tabular Text Files (**.ttf**)

Related Topics:

- Go to *Compiler Procedures* in MAX+PLUS II Help for information on other available Compiler settings.
- Go to *Programmer Procedures* in MAX+PLUS II Help for instructions on creating other types of programming files and on programming or configuring Altera devices.
- Go to Back-Annotating MAX+PLUS II Pin Assignments to Design Architect Symbols for information on back-annotating pin assignments in Mentor Graphics Design Architect schematics.
- Go to Programming Altera Devices for information on the different programming hardware options for Altera device families.

Go to the following topics, which are available on the web, for additional information:

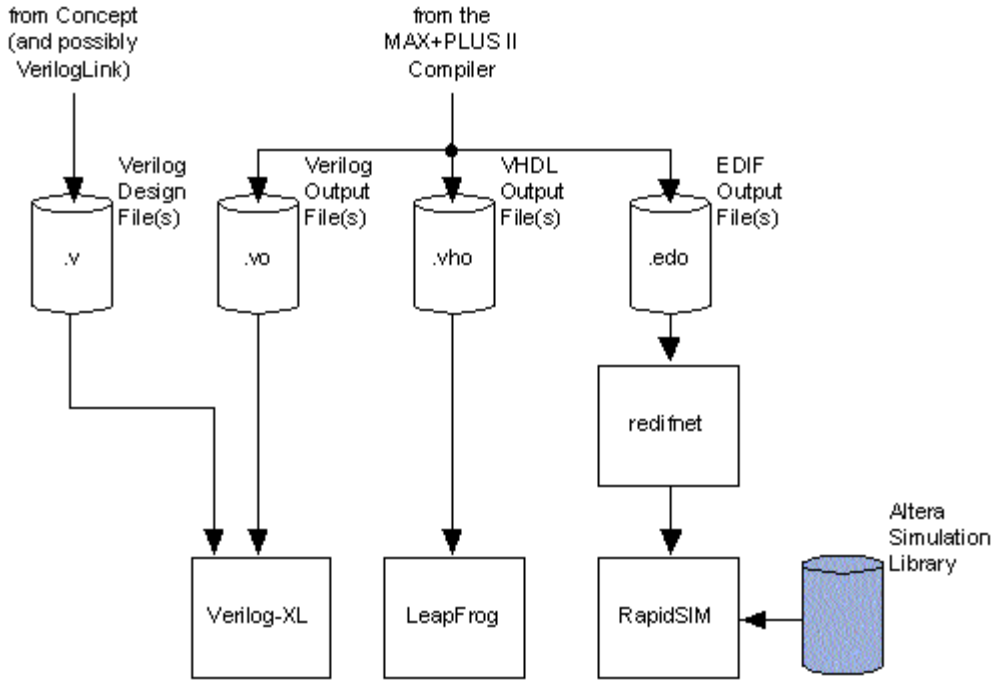
- MAX+PLUS II Development Software
- Altera Programming Hardware

Project Simulation Flow

Figure 1 shows the project simulation flow for the MAX+PLUS® II/Cadence interface.

Figure 1. MAX+PLUS II/Cadence Project Simulation Flow

Altera-provided items are shown in blue.



Initializing Registers in VHDL & Verilog Output Files for Power-Up before Simulation

Altera provides the **add_dc** script, which is availiable in the MAX+PLUS II system directory, to allow you to process MAX+PLUS II-generated Verilog Output Files (**.vo**) and VHDL Output Files (**.vho**) to prepare these files for simulation with another EDA tool. The add_dc script runs the **add_dclr** utility, which inserts a `device_clear` signal that is used for power-up initialization of all registers or flipflops in the design.

The script adds in a top-level signal named `device_clear` and connects it to the `CLRN` pin in all flipflops that should initialize to 0, and to the `PRN` pin of all flipflops that should initialize to 1. If the `CLRN` or `PRN` pin of a flipflop is already being used (i.e., is already connected to a signal), the script modifies the Verilog Output File or VHDL Output File so that the `AND` of the original signal and the `device_clear` pin feed the `CLRN` or `PRN` pin.

To use the **add_dc** script to process Verilog Output Files and VHDL Output Files before simulation with another EDA tool, follow these steps:

1. Make sure that your design file is located in the current directory, or change to the directory in which the design file is located.

2. Type the following command at the command prompt:

   ¥<*path name of* ***add_dc.bat*** *file*>¥add_dc <*design name*> <*path name of* ***add_dclr.exe*** *file*> ↵

For example, if the both the **add_dc.bat** and the **add_dclr.exe** files are located in the **d:¥maxplus2¥exew** directory, and the **d:¥maxplus2¥exew** directory is specified in the search path, you can type the following command at a command prompt to add a `device_clear` signal to a design named `myfifo` in the file **myfifo.vo**:

```
add_dc myfifo d:¥maxplus2¥exew ↵
```

1. The **add_dc** script gives a message if the directory contains both a VHDL Output File and a Verilog Output

File with the same name (<*design name*>**.vo** and <*design*>**.vho**). You should delete or rename whichever of those files should not have the `device_clear` signal added. The **add_dc** script can modify only one design file at a time.

2. When the **add_dc** script processes the Verilog Output File or VHDL Output File, it creates a backup copy of the original file, with the extension **.ori**.

3. The **add_dc** script works only for Verilog Output Files and VHDL Output Files that are generated by MAX+PLUS II.

After you have used the **add_dc** script and are ready to simulate the resulting Verilog Output File or VHDL Output File with another EDA tool, you should assert the active low `device_clear` pin for a period of time that is long enough for the design to initialize. You can then de-assert the pin, and apply simulation vectors to the design.

Performing a Timing Simulation with RapidSIM Software

You can use the Cadence **redifnet** utility to read MAX+PLUS® II-generated EDIF Output Files and prepare them for timing simulation with RapidSIM software. RapidSIM software can simulate both the functionality and the timing of your design. It also checks setup time requirements, hold time requirements, and Clock duty cycle timing requirements on registers.

To simulate projects with RapidSIM software, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment.

2. Generate an EDIF Output File (**.edo**), as described in Compiling Projects with MAX+PLUS II Software.

3. Copy the EDIF Output File <*file name*>**.edo** from the /<*working directory*>/**max2** directory to the /<*working directory*>/**dest** directory.

4. Convert the EDIF Output File into the SCALD project format by typing `redifnet` <*design name*> ↵ at the UNIX prompt from the /<*working directory*>/**dest** directory.

5. Type `lwb_rapidsim` ↵ at the UNIX prompt to generate the **global.cmd** directive file.

6. Choose the **RapidSIM** button from the Logic Workbench window to start RapidSIM and simulate your EDIF Output File.

Performing a Timing Simulation with Verilog-XL Software

Once the MAX+PLUS® II software has compiled a project and generated a Verilog Output File (**.vo**), you can perform a timing simulation using Cadence Verilog-XL software.

To simulate Verilog output files with the Verilog-XL timing simulator, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment.

2. Generate Verilog Output Files (**.vo**), as described in Compiling Projects with MAX+PLUS II Software. The MAX+PLUS II Compiler generates the <*design name*>**.vo** and **alt_max2.vo** files for use with Verilog-XL software.

3. Using any standard text editor, create a stimulus file that includes test vectors for your design.
4. Start the Verilog-XL simulator and simulate your Verilog output files by typing the following command at the UNIX prompt:

   `verilog` <*stimulus filename(s)*> <*design name*> `alt_max2.vo` ↵

Performing a Timing Simulation with Leapfrog Software

Once the MAX+PLUS® II software has compiled a project and generated a VHDL Output File (**.vho**), you can a perform timing simulation using Cadence Leapfrog software.

To simulate a VHDL output file with the Leapfrog timing simulator, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment.
2. If you wish to use MAX+PLUS II-generated Standard Delay Format (SDF) Output Files (**.sdo**) that contain timing information, compile the VITAL library source files, as described in Compiling the VITAL Library for Use with Leapfrog Software.
3. If your design uses functions from the **alt_mf** library, compile the library, as described in Compiling the **alt_mf** Library.
4. Generate a VHDL Output File (**.vho**) and an optional SDF Output File, as described in Compiling Projects with MAX+PLUS II Software.
5. Using any standard text editor, create a stimulus file that includes test vectors for <*design name*>.
6. Start the Leapfrog simulator and simulate the MAX+PLUS II-created VHDL Output File <*design name*>**.vho** by typing `leapfrog` ↵ at the UNIX prompt. Refer to *Chapter 5: SDF Back-Annotation in Leapfrog* in the ***VHDL Simulator User Guide*** or refer to the ***Cadence Openbook*** for more information.

Compiling the VITAL Library for Use with Leapfrog Software

If you wish to use MAX+PLUS® II-generated Standard Delay Format (SDF) Output Files (**.sdo**) that contain timing information when performing post-compilation timing simulation with Leapfrog software, you must first compile the VITAL library source files. The VITAL Timing and Primitive package files are located in the **$CDS_INST_DIR/tools/leapfrog/files/IEEE.src** directory.

To compile the **alt_vtl** library, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment. For example, you must ensure that the appropriate directories are specified in the **cds.lib** file that is located in your working directory.
2. Create a VHDL design, as described in Creating VHDL Designs for Use with MAX+PLUS II Software and save it in your working directory.
3. Change to the **alt_vtl** directory by typing `cd /usr/maxplus2/simlib/concept/alt_vtl` ↵ at the UNIX prompt.
4. Edit the **hdl.var** file located in your working directory to include the following line:

```
DEFINE WORK alt_vtl ↵
```

5. Create the **/usr/maxplus2/simlib/concept/alt_vtl/lib** directory.
6. Type the following commands at the UNIX prompt from the **/usr/maxplus2/simlib/concept/alt_vtl** directory to compile the library:

```
cv -message -file alt_vtl.vhd ↵
cv -message -file alt_vtl.cmp ↵
```

Compiling the alt_mf Library

If your VHDL design uses functions from the **alt_mf** library, you must compile this library. To compile the **alt_mf** library, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS® II/Cadence Working Environment. For example, you must ensure that the appropriate directories are specified in the **cds.lib** file located in your working directory.
2. Change to the **alt_mf** directory by typing `cd /usr/maxplus2/simlib/concept/alt_mf ↵` at the UNIX prompt.
3. Edit the **hdl.var** file located in your working directory to include the following line:

```
DEFINE work alt_mf ↵
```

4. Type the following commands at the UNIX prompt from the **/usr/maxplus2/simlib/concept/alt_mf** directory to compile the library:

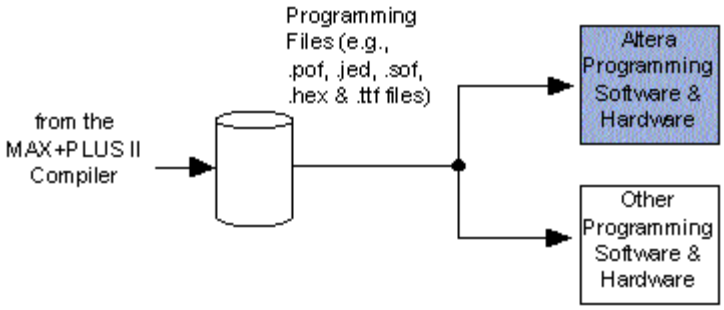
```
cv -message -file ./src/mf.vhd ↵
cv -message -file ./src/mf_components.vhd ↵
```

Programming Altera Devices

Once you have successfully compiled and simulated a project with the MAX+PLUS® II software, you can program an Altera® device and test it in the target circuit. Figure 1 shows the device programming flow for MAX+PLUS II software.

Figure 1. MAX+PLUS II Device Programming Flow

Altera-provided items are shown in blue.



You can program devices with Altera programming hardware and MAX+PLUS II Programmer software installed on a 486- or Pentium-based PC or a UNIX workstation, or with programming hardware and software available from other manufacturers. Table 1 shows the available Altera programming hardware options on PCs and UNIX workstations.

Table 1. Altera Programming Hardware

Programming Hardware Option	PCs	UNIX Work-stations	MAX® 3000A Devices	Classic® & MAX 5000 Devices	MAX 7000 & MAX 7000E Devices	MAX 7000A, MAX 7000AE, MAX 7000B, MAX 7000S MAX 9000 & MAX 9000A Devices	FLEX® 6000, FLEX 6000A, FLEX 8000, FLEX 10K, FLEX 10KA, FLEX 10KB, & FLEX 10KE Devices	In-System Programming/ Configuration
Logic Programmer card, PL-MPU Master Programming Unit, and device-specific adapters	✓		✓	✓	✓	✓		
BitBlaster™ Download Cable	✓	✓	✓			✓	✓	✓
ByteBlasterMV™ Download Cable	✓		✓			✓	✓	✓
MasterBlaster™ Download Cable	✓	✓	✓			✓	✓	✓

If you wish to transfer programming files from a UNIX workstation to a PC over a network with File Transfer Protocol (FTP) or other similar transfer programs, be sure to select binary transfer mode.

Programming hardware from other manufacturers varies, but typically consists of a device connected to one of the serial ports on the workstation. Various vendors, such as Data I/O and BP Microsystems, supply hardware and software for programming Altera devices.

Related Topics:

Go to Compiling Projects with MAX+PLUS II Software for information on creating programming files.

Go to the following topics, which are available on the web, for additional information:

- MAX+PLUS II Development Software
- Altera Programming Hardware
- FLEX Devices
- MAX Devices
- Classic Device Family

Assigning Cliques

You can define a group of logic functions as a single, named unit, called a clique. The MAX+PLUS® II Compiler attempts to place all logic in the clique in the same logic array block (LAB) to ensure optimum speed. If the project does not use multi-LAB devices, or if it is not possible to fit all clique members into a single LAB, the clique assignment ensures that all members of a clique are placed in the same device. In FLEX® 6000, FLEX 8000, FLEX 10K, and MAX® 9000 devices the Compiler also attempts to place the logic in LABs in the same row. Cliques therefore allow you to partition a project so that only a minimum number of signals travel between LABs, and to ensure that no unnecessary LAB-to-LAB or device-to-device delays exist on critical timing paths.

To assign a clique, use the following syntax:

✔ `CLIQUE=`<*clique name*>

For example: `CLIQUE=fast1`

Related Topics:

- Go to the following topics in MAX+PLUS II Help for related information:
  - Assigning a Clique
  - Guidelines for Achieving Maximum Speed Performance

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

You can define a group of logic functions as a single, named unit, called a clique. The MAX+PLUS® II Compiler attempts to place all logic in the clique in the same logic array block (LAB) to ensure optimum speed. If the project does not use multi-LAB devices, or if it is not possible to fit all clique members into a single LAB, the clique assignment ensures that all members of a clique are placed in the same device. In FLEX® 6000, FLEX 8000, MAX® 9000, and FLEX 10K devices, the Compiler also attempts to place the logic in LABs in the same row. Cliques therefore allow you to partition a project so that only a minimum number of signals travel between LABs, and to ensure that no unnecessary LAB-to-LAB or device-to-device delays exist on critical timing paths.

To make pin, logic cell, and chip assignments, use the `set_attribute` command at a **dc_shell** prompt. Before using the `set_attribute` command, add the following line to your **.synopsys_dc.setup** file:

```
edifout_write_properties_list= {LOGIC_OPTION, CLIQUE, CHIP_PIN_LC}
```

↵

✔ To assign a clique, type the following command at a **dc_shell** prompt:

```
set_attribute find(<design object>,(<instance name>))"CLIQUE" -type string "<clique name>" ↵
```

For example:

```
set_attribute find (cell, (U1)) "CLIQUE" -type string "fast1" ↵
```

Related Topics:

- Go to the following topics in MAX+PLUS II Help for related information:
  - Assigning a Clique
  - Guidelines for Achieving Maximum Speed Performance

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Instantiating the clklock Megafunction in Design Architect Schematics

You can instantiate the `clklock` phase-locked loop megafunction, which is supported in selected FLEX® 10K devices, in a Concept schematic. that employ a phase-locked loop (PLL).

To instantiate the `clklock` megafunction in Cadence Concept schematics, follow these steps:

1. Choose the **Add Part** button from the toolbar or type `add` ↵ in the Concept window to open the Component Browser window.
2. Enter the `clklock` megafunction:
   1. Choose **alt_max2** (Library menu) and select *clklock* from the list box.
   2. Type `attribute`, then select the `clklock` component. Change the *CLOCKBOOST* and *INPUT_FREQUENCY* values as needed. For detailed information on the `clklock` megafunction, choose **Megafunctions/LPM** from the MAX+PLUS® II Help menu.
3. Choose **Done**.
4. Continue with the steps necessary to complete your Concept schematic, as described in Creating Concept Schematics for Use with MAX+PLUS II Software.

Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample Concept schematic file, which includes `clklock` instantiation:

- **/usr/maxplus2/examples/cadence/example12/fifo**

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

You can instantiate the Altera Approvided `clklock` phase-locked loop megafunction, which is supported for some FLEX® 10K devices, in a Design Architect schematic.

To instantiate the `clklock` megafunction in a Design Architect schematic, follow these steps:

1. Choose **Altera Libraries** (Library menu).
2. Choose **ALTERA GENLIB** (Altera Libraries menu).
3. Choose **clklock** (ALTERA GENLIB menu).
4. Specify appropriate values for the `CLOCKBOOST` and `INPUT_FREQUENCY` variables. Choose **Megafunctions/LPM** from the MAX+PLUS Â® Â II Help menu for detailed information on the `clklock` megafunction.

5. Choose **OK**.
6. Continue with the steps necessary to complete your Design Architect schematic, as described in Creating Design Architect Schematics for Use with MAX+PLUSÂ II Software.

Installing the AlteraÂprovided MAX+PLUSÂ II/Mentor Graphics/Exemplar Logic interface on your computer automatically creates the sample Design Architect schematic file **/usr/maxplus2/examples/mentor/example7/fifo**, which includes `clklock` megafunction instantiation.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Instantiating the clklock Megafunction in VHDL or Verilog HDL

MAX+PLUS® II interfaces to other EDA tools support the `clklock` phase-locked loop megafunction, which can be used with some FLEX® 10K devices, with the **gencklk** utility. Type `gencklk -h` ↵ at the DOS or UNIX prompt to display information on how to use this utility. The **gencklk** utility generates VHDL or Verilog HDL functional simulation models and a VHDL Component Declaration template file (**.cmp**).

The **gencklk** utility allows parameters for the `clklock` function to be passed from the VHDL or Verilog HDL file to EDIF netlist format. The **gencklk** utility embeds the parameter values in the `clklock` function name; therefore, the values do not need to be declared explicitly.

To instantiate the `clklock` megafunction in VHDL or Verilog HDL, go through the following steps:

1. Type the following command at the DOS or UNIX prompt to generate the `clklock_`*x*`_`*y* function, where *x* is the ClockBoost™ value and *y* is the input frequency in MHz:

   ✔ Type `gencklk` <*ClockBoost*> <*input frequency*> `-vhdl` ↵ for VHDL designs.

   or:

   ✔ Type `gencklk` <*ClockBoost*> <*input frequency*> `-verilog` ↵ for Verilog HDL designs.

   Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu for more information on the `clklock` megafunction.

2. Create a design file that instantiates the `clklock_`*x*`_`*y***.vhd** or `clklock_`*x*`_`*y***.v** file. The **gencklk** utility automatically generates a VHDL Component Declaration template in the **clklock_*x*_*y*.cmp** file that you can incorporate into a VHDL design file.

☞ In MAX+PLUS II version 8.3 and lower, running `genclklk` on a PC always creates files named as **clklock.vhd**, **clklock.cmp**, and **clklock.v**, regardless of the ClockBoost and input frequency values you specify.

Figures 1 and 2 show a `clklock` function with <*ClockBoost*> = 2 and <*input frequency*> = 40 MHz instantiated in VHDL and Verilog HDL design files, respectively.

Figure 1. VHDL Design File with clklock Instantiation (count8.vhd)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY altera;
USE altera.maxplus2.all;     --  Include Altera Component Declarations



ENTITY count8 IS
   PORT (a    : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
         ldn  : IN STD_LOGIC;
         gn   : IN STD_LOGIC;



dnup : IN STD_LOGIC;
```

```
        setn : IN STD_LOGIC;
        clrn : IN STD_LOGIC;
        clk  : IN STD_LOGIC;



co   : OUT STD_LOGIC;
        q    : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END count8;



ARCHITECTURE structure OF count8 IS
   signal clk2x : STD_LOGIC;



COMPONENT clklock_2_40
   PORT (
      INCLK : IN STD_LOGIC;
      OUTCLK : OUT STD_LOGIC
   );
END COMPONENT;



BEGIN
   u1: clklock_2_40
   PORT MAP (inclk=>clk, outclk=>clk2x);



u2: a_8count
   PORT MAP (a=>a(0), b=>a(1), c=>a(2), d=>a(3),
             e=>a(4), f=>a(5), g=>a(6), h=>a(7),
             clk=>clk2x,
             ldn=>ldn,
             gn=>gn,



dnup=>dnup,
             setn=>setn,
             clrn=>clrn,



qa=>q(0), qb=>q(1), qc=>q(2), qd=>q(3),
             qe=>q(4), qf=>q(5), qg=>q(6), qh=>q(7),
             cout=>co);
 END structure;
```

Figure 2. Verilog HDL Design File with clklock Instantiation (count8.v)

```
`timescale 1ns / 10ps
module count8 (a, ldn, gn, dnup, setn, clrn, clk, co, q);
output          co;
output[7:0]     q;



input[7:0]      a;
input           ldn, gn,dnup, setn, clrn, clk;
wire            clk2x;



clklock_2_40 u1 (.inclk(clk), .outclk(clk2x) );
A_8COUNT u2 (.A(a[0]), .B(a[1]), .C(a[2]), .D(a[3]), .E(a[4]), .F(a[5]),



.G(a[6]), .H(a[7]), .LDN(ldn), .GN(gn), .DNUP(dnup),
```

```
            .SETN(setn), .CLRN(clrn), .CLK(clk2x), .QA(q[0]), .QB(q[1]),
            .QC(q[2]), .QD(q[3]), .QE(q[4]), .QF(q[5]), .QG(q[6]),
            .QH(q[7]), .COUT(co) );


endmodule
```

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Powerview Command-Line Syntax

Table 1 shows the command-line syntax for using Powerview functions.

Table 1. Powerview Command-Line Syntax

Action	Command
Start VHDL Analyzer software	`vhdl -v` <*project name*>
Start ViewSynthesis software	`vhdldes`
Load Altera® technology library	`vhdldes> technology altera`
Compile a VHDL design	`vhdldes> vhdl` <*project name*>
Synthesize a design	`vhdldes> synthesize`
Generate wirelist file	`vhdldes> wir`
Create a schematic representation	`vhdldes> viewgen`
Generate a synthesis report file	`vhdldes> report`
Start the graphical user interface for ViewSynthesis	`vhdldes> vdesgui`
Start the VHDL-to-symbol utility	`vhdl2sym` <*project name*>
Start **vsm**	`vsm` <*project name*>
Start ViewSim simulator	`viewsim` <*project name*> -<*project name*>`.cmd`
Start **edifneto**	`edifneto -f` <*project name*>-`l` (`std` *or* `altera`) <*project name*>`.edf`
Start Vantage VHDL Analyzer software	`analyze -src` <*design file*>
Start MOTIVE for Powerview software	`mfp`

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Using Cadence Composer & MAX+PLUS II Software

The following topics describe how to use the Cadence Composer software with MAX+PLUS® II software. Click on one of the following topics for information:

Open a printable version of all topics listed on this page.

Setting Up the MAX+PLUS II/Cadence Working Environment

- Software Requirements
- MAX+PLUS II Directory Structure
- MAX+PLUS II/Cadence Interface File Organization
- Composer Project File Directory Structure
- Altera-Provided Logic & Symbol Libraries

Design Entry

- Design Entry Flow
- Creating Composer Schematics for Use with MAX+PLUS II Software
- Entering Resource Assignments
  - Assigning Pins, Logic Cells & Chips
  - Assigning Cliques
  - Assigning Logic Options
  - Modifying the Assignment & Configuration File with the **setacf** Utility
- Creating Hierarchical Projects in Composer Schematics
- Converting Composer Schematics into MAX+PLUS II-Compatible EDIF Netlist Files with the **altout** Utility

Functional Simulation

- Performing a Functional Simulation of a Composer Schematic with Verilog-XL Software

Related Links:

- Compiling Projects with MAX+PLUS II Software
- Programming Altera Devices
- MAX+PLUS II Development Software
- Altera Programming Hardware
- Cadence web site (http://www.cadence.com)

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Using Cadence Composer & MAX+PLUS II Software

The following topics describe how to use the Cadence Composer software with MAX+PLUS® II software. Click on one of the following topics for information:

This file is suitable for printing only. It does not contain hypertext links that allow you to jump from topic to topic.

Setting Up the MAX+PLUS II/Cadence Working Environment

- Software Requirements
- MAX+PLUS II Directory Structure
- MAX+PLUS II/Cadence Interface File Organization
- Composer Project File Directory Structure
- Altera-Provided Logic & Symbol Libraries

Design Entry

- Design Entry Flow
- Creating Composer Schematics for Use with MAX+PLUS II Software
- Entering Resource Assignments
  - Assigning Pins, Logic Cells & Chips
  - Assigning Cliques
  - Assigning Logic Options
  - Modifying the Assignment & Configuration File with the **setacf** Utility
- Creating Hierarchical Projects in Composer Schematics
- Converting Composer Schematics into MAX+PLUS II-Compatible EDIF Netlist Files with the **altout** Utility

Functional Simulation

- Performing a Functional Simulation of a Composer Schematic with Verilog-XL Software

Related Topics:

- Compiling Projects with MAX+PLUS II Software
- Programming Altera Devices

Go to the following topics, which are available on the web, for additional information:

- MAX+PLUS II Development Software
- Altera Programming Hardware
- Cadence web site (http://www.cadence.com)

Setting Up the MAX+PLUS II/Cadence Working Environment

To use MAX+PLUS® II software with Cadence software, you must first install the MAX+PLUS II software, then establish an environment that facilitates entering and processing designs. The MAX+PLUS II/Cadence interface is installed automatically when you install the MAX+PLUS II software on your computer. Go to *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual for more information on installation and details on the directories that are created during MAX+PLUS II installation. Go to MAX+PLUS II/Cadence Interface File Organization for information about the MAX+PLUS II/Cadence directories that are created during MAX+PLUS II installation.

The information presented here assumes that you are using the C shell and that your MAX+PLUS II system directory is **/usr/maxplus2**. If not, you must use the appropriate syntax and procedures to set environment variables for your shell.

To set up your working environment for the MAX+PLUS II/Cadence interface, follow these steps:

1. Ensure that you have correctly installed the MAX+PLUS II and Cadence software versions described in the MAX+PLUS II/Cadence Software Requirements.

2. Add the following environment variables to your **.cshrc** file:

   ```
   setenv ALT_HOME /usr/maxplus2 ↵
   ```

   ```
   setenv CDS_INST_DIR <Cadence system directory path> ↵
   ```

3. Add the **$ALT_HOME/cadence/bin** and **$CDS_INST_DIR/tools/bin** directories to the `PATH` environment variable in your **.cshrc** file. Make sure these paths are placed before the Cadence hierarchy path.

4. Add **/usr/dt/lib** and **/usr/ucb/lib** to the `LD_LIBRARY_PATH` environment variable in your **.cshrc** file.

5. Create a new **cds.lib** file in your working directory or edit an existing one so that it includes all of the following lines that apply to the Cadence tools you have installed:

   ```
   DEFINE alt_syn ${ALT_HOME}/simlib/concept/alt_syn

   DEFINE lpm_syn ${ALT_HOME}/simlib/concept/lpm_syn

   DEFINE alt_lpm ${ALT_HOME}/simlib/concept/alt_lpm

   DEFINE alt_mf ${ALT_HOME}/simlib/concept/alt_mf

   DEFINE alt_max2 ${ALT_HOME}/simlib/concept/alt_max2

   DEFINE alt_max2 ${ALT_HOME}/simlib/composer/alt_max2/alt_max2

   DEFINE alt_vtl $ALT_HOME/simlib/concept/alt_vtl/lib

   DEFINE altera $ALT_HOME/simlib/concept/alt_mf/lib

   SOFTINCLUDE $CDS_INST_DIR/tools/leapfrog/files/cds.lib

   DEFINE <design name>.
   ```

6. Copy the **/usr/maxplus2/maxplus2.ini** file to your **$HOME** directory:

   ```
   cp /usr/maxplus2/maxplus2.ini $HOME ↵
   ```

```
chmod u+w $HOME/maxplus2.ini ↵
```

The **maxplus2.ini** file contains both Altera- and user-specified initialization parameters that control the MAX+PLUS II software, such as MAX+PLUS II symbol and logic function library paths and the current project name. The MAX+PLUS II installation procedure creates and copies the **maxplus2.ini** file to the **/usr/maxplus2** directory.

Normally, you do not have to edit your local copy of **maxplus2.ini** because the MAX+PLUS II software updates the file automatically whenever you change any parameters or settings. However, if you move the **max2lib** and **max2inc** library subdirectories, you must update the file. Go to "Creating & Using a Local Copy of the **maxplus2.ini** File" in MAX+PLUS II Help for more information.

7. If you are using Concept on a Sun SPARCstation running SunOS, go to Setting Up the MAX+PLUS II/Cadence Concept Work Environment for a Sun SPARCstation Running SunOS Software to install the **redifnet** EDIF netlist reader utility.

8. If you are using Synergy software, edit the **hdl.var** file located in your working directory to include the following line:

   ```
   DEFINE work <design name> ↵
   ```

9. Set up an appropriate directory structure for the tool(s) you are using. See the following topics for information:

   - Composer Project File Directory Structure
   - Concept & RapidSIM Local Work Area Directory Structure

Related Topics:

- ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
- This manual is also available in 4 parts:
  - Preface & Section 1: MAX+PLUS II Installation
  - Section 2: MAX+PLUS II - A Perspective
  - Section 3: MAX+PLUS II Tutorial
  - Appendices, Glossary & Index

MAX+PLUS II Directory Structure

In the MAX+PLUS® II software, a project name is the name of a top-level design file, without the filename extension. This design file can be an EDIF, Verilog HDL, or VHDL netlist file; an AHDL Text Design File (TDF); or any other MAX+PLUS II-supported design file. The EDIF netlist file must be created by the **altout** or **concept2alt** utility and imported into the MAX+PLUS II software as an EDIF Input File (**.edf**).

Project design files and output files are stored in the project directory, with the exception of standard library functions provided by Altera or another EDA tool vendor. The MAX+PLUS II software stores the connectivity data on the links between design files in a hierarchical project in a Hierarchy Interconnect File (**.hif**), but refers to the entire project only by its project name. The MAX+PLUS II Compiler uses the HIF to build a single, fully flattened project database that integrates all design files in a project hierarchy.

MAX+PLUS II/Cadence Interface File Organization

Table 1 shows the MAX+PLUS® II/Cadence interface subdirectories that are created in the MAX+PLUS II system directory (by default, the **/usr/maxplus2** directory) during MAX+PLUS II installation. For information on the other directories that are created during MAX+PLUS II installation, see "MAX+PLUS II File Organization" in *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual.

Table 1. MAX+PLUS II Directory Organization

Directory	Description
./lmf	Contains the Altera-provided Library Mapping File, **cadence.lmf**, that maps Cadence logic functions to equivalent MAX+PLUS II logic functions.
./examples/cadence	Contains the sample files for Cadence software discussed in these ACCESS℠ Key Guidelines.
./cadence	Contains the AMPLE userware for the MAX+PLUS II/Cadence interface.
./simlib/concept/alt_max2	Contains the MAX+PLUS II primitives, including `CARRY`, `CASCADE`, `EXP`, `GLOBAL`, `LCELL`, `SOFT`, `OPNDRN`, `DFFE` (D flipflop with Clock Enable), and `DFFE6K` (D flipflop with Clock Enable and both Clear and Preset for FLEX® 6000 devices only) for use with Concept software.
./simlib/composer/alt_max2	Contains the MAX+PLUS II primitives, including `CARRY`, `CASCADE`, `EXP`, `GLOBAL`, `LCELL`, `SOFT`, `OPNDRN`, `DFFE` (D flipflop with Clock Enable), and `DFFE6K` (D flipflop with Clock Enable and both Clear and Preset for FLEX 6000 devices only) for use with Composer software.
./simlib/concept/alt_lpm	Contains the MAX+PLUS II megafunctions, including library of parameterized modules (LPM) functions, for use with Concept software.
./simlib/concept/max2sim	Contains the MAX+PLUS II/Concept simulation model library, **max2_sim**, for use with RapidSIM software.
./simlib/concept/alt_syn	Contains the MAX+PLUS II synthesis library, **alt_syn**, for use with Synergy and Concept software, and the **vlog2alt** utility.
./simlib/composer/alt_syn	Contains the MAX+PLUS II synthesis library, **alt_syn**, for use with Synergy and Composer software.
./simlib/concept/lpm_syn	Contains the Cadence LPM library, **lpm_syn**, for use with Synergy and Concept software.
./simlib/composer/lpm_syn	Contains the Cadence LPM library, **lpm_syn**, for use with Synergy and Composer software.
./simlib/concept/alt_mf	Contains the MAX+PLUS II VHDL logic function library. (`a_8count` is for the MAX® 7000 and MAX 9000 device families only.)
./simlib/concept/edifnet/templates	Contains template files for Concept directives, i.e., **global.cmd**, **compiler.cmd**, **vloglink.cmd**, **verilog.cmd**, and **master.local**.
./simlib/concept/alt_max2/verilogUdps	Contains Verilog HDL modules that are the equivalent of the primitives contained in **alt_max2** library for use with Concept software.
./simlib/composer/alt_max2/verilogUdps	Contains Verilog HDL modules that are the equivalent of the primitives contained in **alt_max2** library for use with Composer software.
./simlib/concept/alt_vtl **./simlib/composer/alt_vtl**	Contains VITAL library source files for use with Concept or Composer software.
./simlib/composer/alt_max2/verilog	Contains simulation modules for all symbols in the **alt_max2** Composer library.

Related Topics:

- ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)

- This manual is also available in 4 parts:
  - Preface & Section 1: MAX+PLUS II Installation
  - Section 2: MAX+PLUS II - A Perspective
  - Section 3: MAX+PLUS II Tutorial
  - Appendices, Glossary & Index
- FLEX Devices
- MAX Devices
- Classic Device Family

Composer Project File Directory Structure

The Composer software generates the following files for each schematic (where *x* represents a Composer-generated number):

- ***<drawing name>*_x/schema_59.0_x**
- ***<drawing name>*_x/schema_59.0_x%**

Altera-Provided Logic & Symbol Libraries

The MAX+PLUS® II/Cadence environment provides four logic and symbol libraries that are used for compiling, synthesizing, and simulating designs.

You can create your own libraries of custom symbols and logic functions in Concept and Composer. You can use custom symbols to incorporate an EDIF Input File, Text Design File (TDF), or any other MAX+PLUS II-supported design file into a project. MAX+PLUS II uses the **cadence.lmf** Library Mapping File to map standard Concept or Composer symbols to equivalent MAX+PLUS II logic functions. To use custom symbols, you can create a custom LMF that maps your custom symbols to the equivalent MAX+PLUS II-supported design file. You must also specify the directory that contains the MAX+PLUS II-supported design file(s) as a user library with the MAX+PLUS II **User Libraries** command (Options menu). Go to "Library Mapping File" and "Cadence Library Mapping File (**cadence.lmf**)" in MAX+PLUS II Help for more information.

The alt_max2 Library

You can enter a Concept or Composer Design Architect schematic with primitives and macrofunctions from the Altera-provided symbol library **alt_max2**. The **alt_max2** library includes 74-series macrofunctions and several MAX+PLUS II primitives with corresponding Verilog HDL simulation models for controlling design synthesis and fitting. It also includes four macrofunctions--`a_8count`, `a_8mcomp`, `a_8fadd`, and `a_81mux`--that are optimized for different device families, and the `clklock` phase-locked loop megafunction, which is supported by some FLEX® 10K devices, with corresponding Verilog HDL and VHDL simulation models. See Table 1. Choose **Old-Style Macrofunctions** and/or **Primitives** from the MAX+PLUS II Help menu for more information on functions in the **alt_max2** library.

The alt_lpm Library

The Altera-provided **alt_lpm** library, which is available for Concept and Verilog HDL designs, includes standard functions from the library of parameterized modules (LPM) 2.1.0, except the truth table, finite state machine, and pad functions. Other parameterized functions, including cycle-shared FIFO (`csfifo`) and cycle-shared dual-port RAM (`csdpram`) are also included. The LPM standard defines a set of parameterized modules (i.e., parameterized megafunctions) and their corresponding representations in an EDIF netlist file. These logic functions allow you to create and functionally simulate an LPM-based design without targeting a specific device family. The parameters you specify for each LPM function determine the simulation models that will be generated. After the design is completed, you can target the design to any device family. In designs created with Concept, the Altera **alt_lpm** library works only with HDL Direct and the **hdlconfig** utility. Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu for more information about LPM functions in the **alt_lpm** library.

The lpm_syn Library

The **lpm_syn** library contains the Altera-provided parameterized functions. The **lpm_syn** library is similar to the **alt_lpm** library, except that it contains VHDL and Verilog HDL logic functions for use with Synergy, Concept, and Composer software.

The alt_mf Library

Altera provides a VHDL logic function library, **alt_mf**, that currently includes four macrofunctions--`a_8count`, `a_8mcomp`, `a_8fadd`, and `a_81mux`--for controlling design synthesis and fitting. These elements can be instantiated directly in your VHDL file. To designate that these logic functions should pass untouched through the EDIF netlist file to the MAX+PLUS II Compiler, you must select the *Maintain* attribute constraint for instances of these functions before running the Synergy software. These models allow you to perform functional VHDL simulation while maintaining an architecture-independent VHDL description.

Table 1 shows the MAX+PLUS II-specific logic functions.

Macrofunctions *Note (1)*		**Primitives**			
Name	**Description**	**Name**	**Description**	**Name**	**Description**
`8fadd`	8-bit full adder	`LCELL`	Logic cell buffer	`EXP`	MAX® 5000, MAX 7000, and MAX 9000 Expander buffer
`8mcomp`	8-bit magnitude comparator	`GLOBAL`	Global input buffer	`SOFT`	Soft buffer
`8count` *Note (2)*	8-bit up/down counter	`CASCADE`	FLEX 6000, FLEX 8000, and FLEX 10K cascade buffer	`OPNDRN`	Open-drain buffer
`81mux`	8-to-1 multiplexer	`CARRY`	FLEX 6000, FLEX 8000, and FLEX 10K carry buffer	`DFFE` `DFFE6K` *Note (3)*	D-type flipflop with Clock Enable
`clklock`	Phase-locked loop				

Notes:

1. Logic function names that begin with a number must be preceded by "`a_`" in VHDL designs. For example, `8fadd` must be specified as `a_8fadd`.
2. The `a_8count` logic function is for the MAX 7000 and MAX 9000 device families only.
3. For designs that are targeted to FLEX 6000 devices, you should use the `DFFE` primitive only if the design contains either a Clear or Preset signal, but not both. If your design contains both a Clear and a Preset signal, you must use the `DFFE6K` primitive.

Related Topics:

- FLEX Devices
- MAX Devices
- Classic Device Family

- MAX+PLUS II Development Software
- Altera Programming Hardware
- Cadence web site (http://www.cadence.com)

Table 1. MAX+PLUS II-Specific Logic Functions

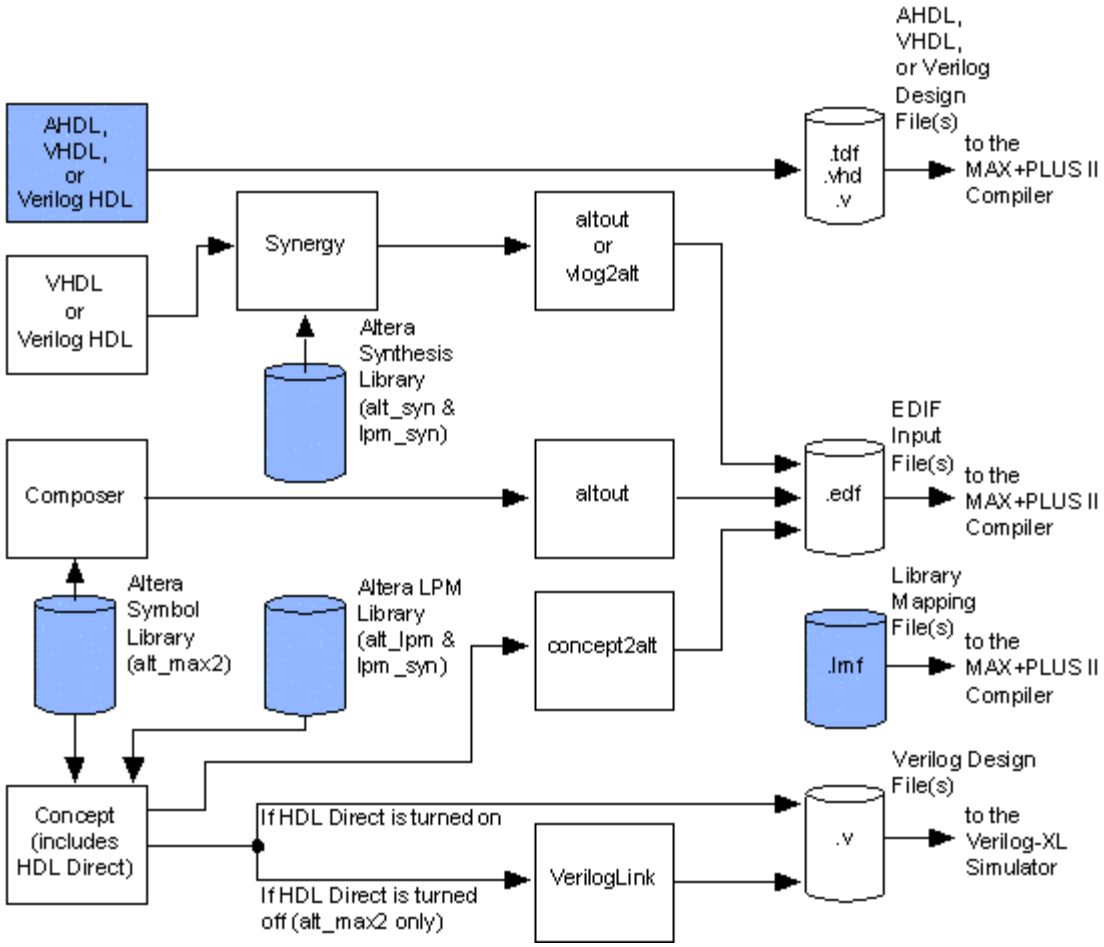
You can create your own libraries of custom symbols and logic functions in Concept and Composer. You can use custom symbols to incorporate an EDIF Input File, Text Design File (TDF), or any other MAX+PLUS II-supported design file into a project. MAX+PLUS II uses the **cadence.lmf** Library Mapping File to map standard Concept or Composer symbols to equivalent MAX+PLUS II logic functions. To use custom symbols, you can create a custom LMF that maps your custom symbols to the equivalent MAX+PLUS II-supported design file. You must also specify the directory that contains the MAX+PLUS II-supported design file(s) as a user library with the MAX+PLUS II **User Libraries** command (Options menu). Go to "Library Mapping File" and "Cadence Library Mapping File (**cadence.lmf**)" in MAX+PLUS II Help for more information.

Cadence Design Entry Flow

Figure 1 shows the design entry flow for the MAX+PLUS® II/Cadence interface.

Figure 1. MAX+PLUS II/Cadence Design Entry Flow

Altera-provided items are shown in blue.



Creating Composer Schematics for Use with MAX+PLUS II Software

You can create Composer schematics and convert them into EDIF Input Files (**.edf**) that can be processed with the MAX+PLUS® II Compiler. To create a Composer schematic for use with the MAX+PLUS II software, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment.

2. Start the Composer schematic editor from the <*working directory*> by typing `icds` ↵ at a UNIX prompt. Use the graphical user interface to structure and organize your files to create an environment that facilitates entering and processing designs. Go to Composer Project File Directory Structure for more information on directories in Composer.

3. Choose **Library Path Editor** (Tools menu) to create the <*design name*> library. In the **Library** dialog box, type <*project directory name*> as the *Library* name and `./source/`<*design name*> as the *Path* name. Choose **Save** (File menu), then choose **Exit** (File menu) to save the path.

4. Choose **Library Manager** (Tools menu) to start Composer and create a new design.

5. Type <*project directory name*> as the *Library* name, <*design name*> as the *Cell* name, and `schematic` as the *View* name in the **Library Manager** dialog box and press the ↵ key.

6. Enter primitives, megafunctions, and macrofunctions from the following libraries:

    - MAX+PLUS II-compatible primitives, megafunctions, and macrofunctions are available in the Altera-

provided **alt_max2** component library.

- Input, output, and bidirectional pins are available in the Cadence **basic** library located under **/cadence/etc/cdslib**.
- MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.

If you wish to create a hierarchical design that contains symbols representing other design files, such as Altera® Hardware Description Language (AHDL) Text Design Files, go to Creating Hierarchical Projects in Composer Schematics.

7. Enter meaningful instance names for all symbols and functions so that you can easily trace internal node names during simulation and debugging operations. For example, if an `a161` macrofunction is instantiated several times in one design, you should define a unique name for each instance. The instance name for each symbol is controlled by `INST` property. For more information on assigning properties, refer to the Cadence ***Composer User Guide***.
8. (Optional) To enter resource assignments in your Composer schematic, go to Entering Resource Assignments. You can also enter resource assignments from within the MAX+PLUS II software.
9. (Optional) Functionally simulate the design with the Verilog-XL simulator. Altera provides Verilog HDL simulation modules in the **/usr/maxplus2/simlib/composer/alt_max2/verilog** and **/usr/maxplus2/simlib/composer/alt_max2/verilogUdps** directories. Go to Performing a Functional Simulation of a Composer Schematic with Verilog-XL Software for more information.
10. Use the **altout** utility to generate an EDIF netlist file that can be imported into the MAX+PLUS II software, as described in Converting Composer Schematics into MAX+PLUS II-Compatible EDIF Netlist Files with the **altout** Utility.
11. Process the <*design name*>**.edf** file with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample Composer schematic files:

- **/usr/maxplus2/examples/cadence/example2/fulladd**
- **/usr/maxplus2/examples/cadence/example5/fulladd2**
- **/usr/maxplus2/examples/cadence/example7/fa2**

Entering Resource Assignments

The MAX+PLUS® II software allows you to enter a variety of resource and device assignments for your projects. Resource assignments are used to assign logic functions to a particular pin, logic cell, I/O cell, embedded cell, row, column, Logic Array Block (LAB), Embedded Array Block (EAB), chip, clique, local routing, logic option, timing requirement, or connected pin group. In MAX+PLUS II software, you can enter all types of resource and device assignments with Assign menu commands. You can also enter pin, logic cell, I/O cell, embedded cell, LAB, EAB, row, and column assignments in the MAX+PLUS II Floorplan Editor. The Assign menu commands and the Floorplan Editor all save assignment information in the ASCII Assignment & Configuration File (**.acf**) for the project. In addition, you can edit ACFs manually in any standard text editor or with the **setacf** utility.

Concept & Composer Schematics

In both Concept and Composer schematics, you can assign a limited subset of these resource assignments by assigning properties to symbols. These properties are incorporated into the EDIF netlist file(s). The MAX+PLUS II software automatically converts assignment information from the EDIF Input File into the ACF format. For information on making MAX+PLUS II-compatible resource assignments, go to the following topics:

- Assigning Pins, Logic Cells & Chips
- Assigning Cliques
- Assigning Logic Options
- Modifying the Assignment & Configuration File with the **setacf** Utility

Go to the ***Cadence Concept Schematic User Guide*** and ***Composer Reference User Guide*** for details on how to assign properties. Go to "Resource Assignments in EDIF Input Files" and "Assigning Resources in a Third-Party Design Editor" in MAX+PLUS II Help for more information on assignments or properties that can be assigned in Concept and Composer.

Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample Concept and Composer schematic files, which include resource assignments:

- **/usr/maxplus2/examples/cadence/example6/fa2** (Concept)
- **/usr/maxplus2/examples/cadence/example7/fa2** (Composer)

VHDL & Verilog HDL Design Files

For Verilog HDL- and VHDL-based designs, you must use the MAX+PLUS II software or the **setacf** utility to enter resource assignments. For information on using the **setacf** utility, go to Modifying the Assignment & Configuration File with the **setacf** Utility.

For information on entering assignments in the MAX+PLUS II software with Assign menu commands or in an ACF, go to "resource assignments" or "ACF, format" in MAX+PLUS II Help using **Search for Help on** (Help menu).

Assigning Pins, Logic Cells & Chips

You can assign a single logic function to a specific pin or logic cell (including I/O cells and embedded cells) within a chip, and assign one or more functions to a specific chip. A chip is a group of logic functions defined as a single, named unit, which can be assigned to a specific device.

You can assign a signal to a particular pin to ensure that the signal is always associated with that pin, regardless of future changes to the project. If you wish to set and maintain the performance of your project, assigning logic to a specific logic cell within a chip can minimize timing delays. In a project that is partitioned among multiple devices, you can assign logic functions that must be kept together in the same device to a chip. Chip assignments allow you to split a project so that only a minimum number of signals travel between devices, and to ensure that no unnecessary device-to-device delays exist on critical timing paths. You can assign a chip to a device in some EDA tools or in the MAX+PLUS® II software.

Use the following syntax for chip, pin, and logic cell assignments:

- To assign a logic function to a chip:

  `CHIP_PIN_LC=<`*chip name*`>`

  For example: `CHIP_PIN_LC=chip1`

- To assign a pin number within a chip:

CHIP_PIN_LC=<*chip name*>@<*pin number*>

For example: CHIP_PIN_LC=chip1@K2

- To assign a logic cell, I/O cell, or embedded cell number:

  CHIP_PIN_LC=<*chip name*>@LC<*logic cell number*>

  CHIP_PIN_LC=<*chip name*>@IOC<*I/O cell number*>

  CHIP_PIN_LC=<*chip name*>@EC<*embedded cell number*>

  For example: CHIP_PIN_LC=chip1@LC44

Related Topics:

- Go to "Devices & Adapters" and "Assigning a Device" in MAX+PLUS II Help for information on device pin-outs and assigning devices, respectively, in the MAX+PLUS II software.
- Go to Back-Annotating MAX+PLUS II Pin Assignments to Design Architect Symbols for information on back-annotating pin assignments in Mentor Graphics Design Architect schematics.

Name

./lmf Contains the Altera-provided Library Mapping File, **cadence.lmf**, that maps Cadence logic functions to equivalent MAX+PLUS II logic functions.

Altera-provided items are shown in blue.

Assigning Cliques

You can define a group of logic functions as a single, named unit, called a clique. The MAX+PLUS® II Compiler attempts to place all logic in the clique in the same logic array block (LAB) to ensure optimum speed. If the project does not use multi-LAB devices, or if it is not possible to fit all clique members into a single LAB, the clique assignment ensures that all members of a clique are placed in the same device. In FLEX® 6000, FLEX 8000, FLEX 10K, and MAX® 9000 devices the Compiler also attempts to place the logic in LABs in the same row. Cliques therefore allow you to partition a project so that only a minimum number of signals travel between LABs, and to ensure that no unnecessary LAB-to-LAB or device-to-device delays exist on critical timing paths.

To assign a clique, use the following syntax:

✔ CLIQUE=<*clique name*>

For example: CLIQUE=fast1

Related Topics:

- Assigning a Clique
- Guidelines for Achieving Maximum Speed Performance

Assigning Logic Options

Logic option and logic synthesis style assignments allow you to guide logic synthesis with logic optimization features that are specific to Altera® devices. You can assign logic options and styles to individual logic functions in your design. The MAX+PLUS® II Compiler also uses a device-family-specific default logic synthesis style for each project.

Related Topics:

- Go to "Resource Assignments in EDIF Input Files" and "Assigning Resources in a Third-Party Design Editor" in MAX+PLUS II Help for complete and up-to-date information on logic option and logic synthesis style assignments, including definitions and syntax of these assignments.

Modifying the Assignment & Configuration File with the setacf Utility

Altera provides the **setacf** utility to help you modify a project's Assignment & Configuration File (**.acf**) from the command line, without opening the file with a text editor. Type `setacf -h` ↵ at a UNIX or DOS prompt to get help on this utility.

Creating Hierarchical Projects in Composer Schematics

If you wish to create a hierarchical design that contains symbols representing other design files, such as Altera® Hardware Description Language (AHDL) Text Design Files (**.tdf**), you can create a hollow-body symbol that represents a design file and then instantiate it in your Composer schematic.

To create a hierarchical project in your Composer schematic, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS® II/Cadence Working Environment.
2. Create a Composer schematic and save it in your working directory, as described in Creating Composer Schematics for Use with MAX+PLUS II Software.

   ☞ You can instantiate MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.

3. Create the hollow-body symbol <*design name*> in Composer by typing `icds` ↵ from the <*working directory*> at the UNIX prompt.
4. Choose **Library Path Editor** (Tools menu) to create the <*design name*> library. Type <*design name*> as the *Library* name and `./source/`<*design name*> as the *Path* name. Choose **Save** (File menu), then choose **Exit** (File menu) to save the path and exit from the Library Path Editor.
5. Choose **Library Manager** (Tools menu) to start Composer and create a symbol for your design. Type <*project directory name*> as the *Library* name, <*lower-level design name*> as the *Cell* name, `symbol` as the *View* name, and then press ↵.
6. Create a hollow-body symbol that represents the inputs and outputs of your design.
7. Enter input and output pins for the symbol.

8. Save the symbol.
9. To enter the symbol in your higher-level schematic design, choose the **Component** button and type <*project directory name*> as the *Library* name, <*lower-level design name*> as the *Cell* name, and `symbol` as the *View* name.
10. The MAX+PLUS II software uses the **cadence.lmf** Library Mapping File to map Concept symbols to equivalent MAX+PLUS II logic functions. To use custom symbols, you must create a custom LMF that maps your custom symbols to the equivalent EDIF Input File, Text Design File (TDF), or other design file. You will also need to specify this custom LMF in the **EDIF Netlist Reader Settings** dialog box before compiling with the MAX+PLUS II software. See Compiling Projects with MAX+PLUS II Software for more information.
11. Continue with the steps necessary to complete your Composer schematic, as described in Creating Composer Schematics for Use with MAX+PLUS II Software.

    Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample hierarchical AHDL and Composer schematic file:

    - **/usr/maxplus2/examples/cadence/example5/fulladd2**

Converting Composer Schematics into MAX+PLUS II- Compatible EDIF Netlist Files with the altout Utility

You can use the **altout** utility to generate an EDIF netlist file from a Composer schematic. This file can then be imported into the MAX+PLUS® II software as an EDIF Input File (**.edf**).

To convert Composer schematics into MAX+PLUS II-compatible EDIF netlist files, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment.
2. Create a Composer schematic and save it in your working directory, as described in Creating Composer Schematics for Use with MAX+PLUS II Software.

   ☞ You can instantiate MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.

3. Type the following command at the UNIX prompt from the working directory that contains the schematic:

   `altout -lib` <*design name*> `-rundir max2` <*design name*> ↵
4. Process the <*design name*>**.edf** file with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Performing a Functional Simulation of a Composer Schematic with Verilog-XL Software

You can perform a functional simulation of a Cadence Composer schematic with the Verilog-XL simulator before compiling your project with the MAX+PLUS® II software.

To functionally simulate a Composer schematic, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment.
2. Create a Composer schematic and save it in your working directory, as described in Creating Composer Schematics for Use with MAX+PLUS II Software.
3. In Composer, select **Simulation** from the *Tools* drop-down list.
4. Select **Verilog-XL** to start the *Verilog-XL Integration Control* window.
5. When you are ready to compile the project, generate an EDIF netlist file <*design name*>**.edf** with the **altout** utility, as described in Converting Composer Schematics into MAX+PLUS II-Compatible EDIF Netlist Files with the **altout** Utility.
6. Process the <*design name*>**.edf** file with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Compiling Projects with MAX+PLUS II Software

The MAX+PLUS® II Compiler can process design files in a variety of formats. This topic describes how to use MAX+PLUS II software to compile projects in which the top-level design file is an EDIF Input File (with the extension **.edf**).

Related Topics:

- Go to MAX+PLUS II Help for information on compiling VHDL and Verilog HDL, design files directly with the MAX+PLUS II Compiler.
- Go to Running Synopsys Compilers from MAX+PLUS II Software for information on running the Synopsys Design Compiler or FPGA Compiler software on a VHDL or Verilog HDL design from within the MAX+PLUS II Compiler window.

To compile a design (also called a "project") with MAX+PLUS II software, go through the following steps:

1. Create design files that are compatible with the MAX+PLUS II software and convert them into EDIF Input Files with the extension **.edf**. Specific instructions for some tools are described in these MAX+PLUS II ACCESS℠ Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your design entry or synthesis and optimization tool.
2. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, create a mapping for each function in a Library Mapping File (**.lmf**) to map the custom symbol to the corresponding EDIF Input File, AHDL Text Design File (**.tdf**), or other MAX+PLUS II-supported design file. These custom functions are represented in design files as hollow-body symbols or "black box" HDL descriptions.

   ☞ Go to "Library Mapping Files (**.lmf**)" in MAX+PLUS II Help for more information.
3. Open MAX+PLUS II and specify the name of your top-level design file as the project name with the **Project Name** command (File menu). If you open an HDL file in the MAX+PLUS II Text Editor, you can choose the **Project Set Project to Current File** command (File menu) instead.

   You can also compile a project from a command line. However, the first time you compile a project,

the settings you need to specify are easier to specify from within the MAX+PLUS II software. After you have run the graphical user interface for the MAX+PLUS II software at least once, you can more easily use the command-line **setacf** utility to modify options in the Assignment & Configuration File (**.acf**) for the project. Type `setacf -h` ↵ and `maxplus2 -h` ↵ for descriptions of **setacf** and MAX+PLUS II command-line syntax.

4. Choose **Device** (Assign menu) and select the target Altera device family in the *Device Family* drop-down list box. If you wish to implement the design logic in a specific device, select it in the *Devices* box. Otherwise, select *AUTO* to allow the MAX+PLUS II Compiler to choose the best device(s) in the current device family. If your design entry or synthesis and optimization tool required you to specify a target family and/or device, specify the same information in this dialog box. For information on partitioning logic among multiple devices, go to MAX+PLUS II Help. Choose **OK**.

5. Open the Compiler window by choosing the **Compiler** command (MAX+PLUS II menu). Go through the following steps to specify the options necessary to compile the design file(s) in your project:

    1. Ensure that all EDIF netlist files have the extension **.edf** and choose **EDIF Netlist Reader Settings** (Interfaces menu).

    2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor. This name should be the name of the vendor whose tool(s) you used to create the EDIF netlist files. If your vendor name does not appear, select *Custom* instead.

        If you are compiling a design created with Synopsys FPGA Express software, select *Synopsys*, choose the **Customize** button, enter <*project name*>**.lmf** in the *LMF #1* box, choose **OK**, and skip to step 6.

    3. If you selected an existing vendor name in the *Vendor* box and your project contains design files that require custom LMF mappings, choose the **Customize** button to expand the dialog box to show all settings. Turn on the *LMF #2* checkbox and type your custom LMF's filename in the corresponding text box, or select a name from the *Files* box. The selection in the *Vendor* box will change to *Custom* and all settings will be retained until you change them again.

    4. If you selected *Custom* in the *Vendor* box, choose the **Customize** button to expand the dialog box to show all settings. Any previously defined custom settings will be displayed. Under *Signal Names*, type one or more names with up to 20 total name characters in the *VCC* or *GND* box if your EDIF Input File(s) use one or more names other than `VCC` or `GND` for the global high or low signals. Multiple signal names must be separated by either a comma (,) or a space. Under *Library Mapping Files*, turn on the *LMF #1* checkbox and type a filename in the text box following it, or select a name from the *Files* box. If necessary, specify another LMF name in the *LMF #2* box. Go to MAX+PLUS II Help for detailed information on the settings available in the **EDIF Netlist Reader Settings** dialog box.

    5. Choose **OK**.

6. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, you may need to ensure that all files are present in your project directory, i.e., the same directory as the top-level design file. Otherwise, you must specify the directories containing these files as user libraries with the **User Libraries** command (Options menu).

7. Follow all guidelines that apply to your design entry or synthesis and optimization tool:

    - Exemplar Logic Galileo Extreme-Specific Compiler Settings
    - Synopsys DesignWare-Specific Compiler Settings
    - Converting Synopsys FPGA Compiler & Design Compiler Timing Constraints into MAX+PLUS II-Compatible Format with the **syn2acf** Utility
    - Synplicity Synplify-Specific Compiler Settings

8. If you wish to generate EDIF, VHDL, or Verilog HDL output files for post-compilation simulation or timing analysis with another EDA tool, go through the following steps:

    1. (Optional) Turn on the **Optimize Timing SNF** command (Processing menu) to reduce the size of the output file(s). Turning on this command can reduce the size of output netlists by up to 30%.

        ☞ This command does not create optimized timing SNFs on UNIX workstations. However, a non-optimized timing SNF provides the same functional and timing information as an optimized timing SNF.

    2. If you wish to generate EDIF Output Files (**.edo**), go through these steps:

        1. Turn on the **EDIF Netlist Writer** command (Interfaces menu). Then choose the **EDIF Netlist Writer Settings** command (Interfaces menu).

        2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor and choose **OK**. If your vendor name does not appear, select *Custom* instead and specify the settings that are appropriate for your simulation or timing analysis tool. Go to MAX+PLUS II Help for detailed information on the options available in the **EDIF Netlist Writer Settings** dialog box.

        3. To generate an optional Standard Delay Format (SDF) Output File (**.sdo**), choose the **Customize** button to expand the dialog box to show all settings. Select one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**.

        The filenames of the EDIF Output File(s) and optional SDF Output File(s) are the same as the user-defined chip name(s) for the project; if no chip names exist, the Compiler assigns filenames that are based on the project name. For a multi-device project, the Compiler also generates a top-level EDIF Output File that is uniquely identified by "**_t**" appended to the project name. In addition, the Compiler automatically generates a VHDL Memory Model Output File, *<project name>***.vmo**, when it generates an EDIF Output File that contains memory (RAM or ROM).

    3. If you wish to generate VHDL Output Files (**.vho**), turn on the **VHDL Netlist Writer** command (Interfaces menu). Then choose **VHDL Netlist Writer Settings** command (Interfaces menu). Select *VHDL Output File (.vho)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF ver. 2.1 files contain timing delay information that allows you to perform back-annotation simulation in VHDL with VITAL-compliant simulation libraries. The VHDL Output Files generated by the Compiler have the extension **.vho**, but are otherwise named in the same way as the EDIF Output Files described above.

    4. If you wish to generate Verilog HDL Output Files (**.vo**), turn on the **Verilog Netlist Writer** command (Interfaces menu). Then choose **Verilog Netlist Writer Settings** command (Interfaces menu). Select *Verilog Output File (.vo)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF Output Files contain timing delay information that allows you to perform back-annotation simulation in Verilog HDL. The Verilog Output Files generated by the Compiler have the extension **.vo**, but are otherwise named in the same way as the EDIF Output Files described above.

9. To run the MAX+PLUS II Compiler, choose the **Project Save & Compile** command (File menu) or choose the **Start** button in the Compiler window.

    ☞ See step 3 for information on running MAX+PLUS II software from the command line.

10. Once you have compiled the project with the MAX+PLUS II Compiler, you can use the VHDL, Verilog HDL, or EDIF output file(s), and the optional SDF Output File(s) (**.sdo**) to perform timing analysis or timing simulation with another EDA tool. Specific instructions for some tools are described in these MAX+PLUS II ACCESS Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your EDA tool.

The MAX+PLUS II Compiler also generates a Report File (**.rpt**), a Pin-Out File (**.pin**), and one or more of the following files for device programming or configuration:

- JEDEC Files (**.jed**)
- Programmer Object Files (**.pof**)
- SRAM Object Files (**.sof**)
- Hexadecimal (Intel-format) Files (**.hex**)
- Tabular Text Files (**.ttf**)

Related Topics:

- Go to *Compiler Procedures* in MAX+PLUS II Help for information on other available Compiler settings.
- Go to *Programmer Procedures* in MAX+PLUS II Help for instructions on creating other types of programming files and on programming or configuring Altera devices.
- Go to Back-Annotating MAX+PLUS II Pin Assignments to Design Architect Symbols for information on back-annotating pin assignments in Mentor Graphics Design Architect schematics.
- Go to Programming Altera Devices for information on the different programming hardware options for Altera device families.

Go to the following topics, which are available on the web, for additional information:

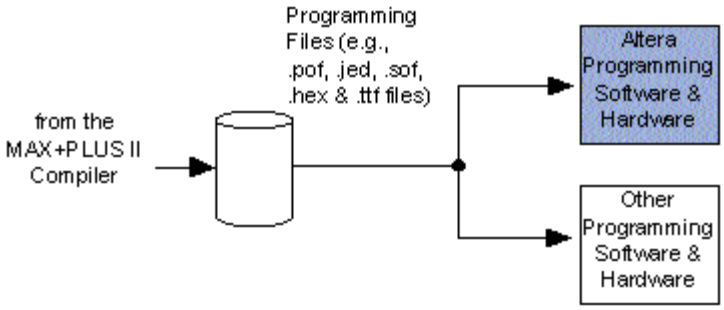
- MAX+PLUS II Development Software
- Altera Programming Hardware

Programming Altera Devices

Once you have successfully compiled and simulated a project with the MAX+PLUS® II software, you can program an Altera® device and test it in the target circuit. Figure 1 shows the device programming flow for MAX+PLUS II software.

Figure 1. MAX+PLUS II Device Programming Flow

Altera-provided items are shown in blue.



You can program devices with Altera programming hardware and MAX+PLUS II Programmer software installed on a 486- or Pentium-based PC or a UNIX workstation, or with programming hardware and software available from other manufacturers. Table 1 shows the available Altera programming hardware options on PCs and UNIX

workstations.

Table 1. Altera Programming Hardware

Programming Hardware Option	PCs	UNIX Work-stations	MAX® 3000A Devices	Classic® & MAX 5000 Devices	MAX 7000 & MAX 7000E Devices	MAX 7000A, MAX 7000AE, MAX 7000B, MAX 7000S MAX 9000 & MAX 9000A Devices	FLEX® 6000, FLEX 6000A, FLEX 8000, FLEX 10K, FLEX 10KA, FLEX 10KB, & FLEX 10KE Devices	In-System Programming/ Configuration
Logic Programmer card, PL-MPU Master Programming Unit, and device-specific adapters	✓		✓	✓	✓	✓		
BitBlaster Download Cable	✓	✓	✓			✓	✓	✓
ByteBlasterMV Download Cable	✓		✓			✓	✓	✓
MasterBlaster Download Cable	✓	✓	✓			✓	✓	✓

If you wish to transfer programming files from a UNIX workstation to a PC over a network with File Transfer Protocol (FTP) or other similar transfer programs, be sure to select binary transfer mode.

Programming hardware from other manufacturers varies, but typically consists of a device connected to one of the serial ports on the workstation. Various vendors, such as Data I/O and BP Microsystems, supply hardware and software for programming Altera devices.

Related Topics:

- Go to Compiling Projects with MAX+PLUS II Software for information on creating programming files.

Go to the following topics, which are available on the web, for additional information:

- MAX+PLUS II Development Software
- Altera Programming Hardware
- FLEX Devices
- MAX Devices
- Classic Device Family

Last updated on December 6, 1999 for the MAX+PLUS II software version 9.4.

Composer Project File Directory Structure

The Composer software generates the following files for each schematic (where *x* represents a Composer-generated number):

- **<*drawing name*>_x/schema_59.0_x**
- **<*drawing name*>_x/schema_59.0_x%**

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

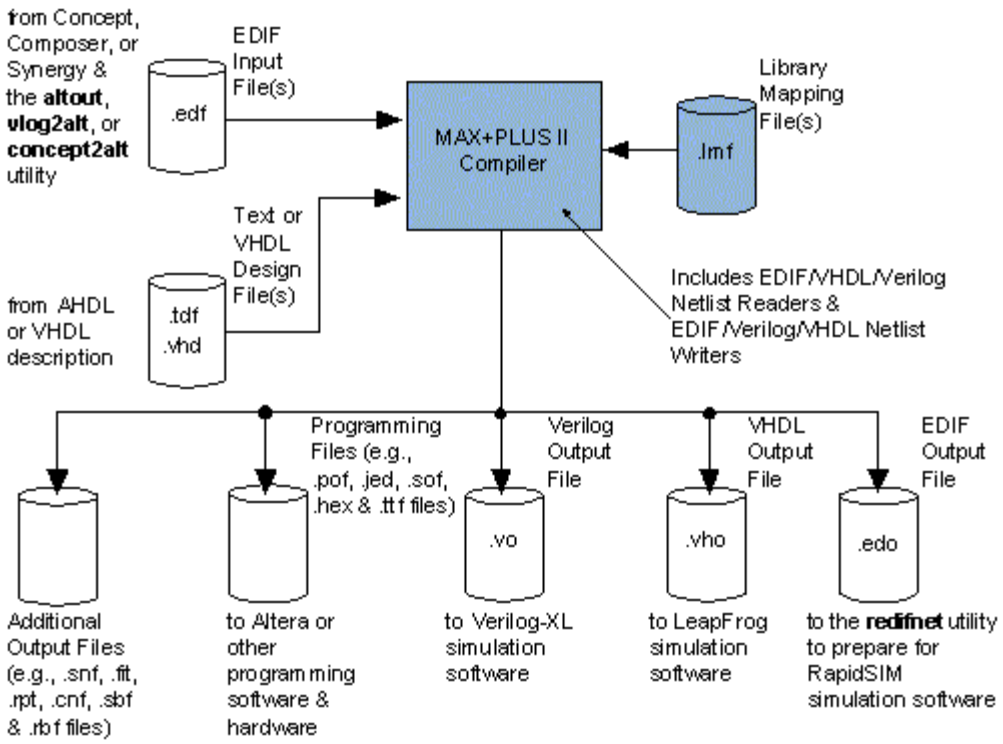
Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Project Compilation Flow

Figure 1 shows the MAX+PLUS® II/Cadence project compilation flow.

Figure 1. MAX+PLUS II/Cadence Project Compilation Flow

Altera-provided items are shown in blue.



Feedback

Did this information help you?

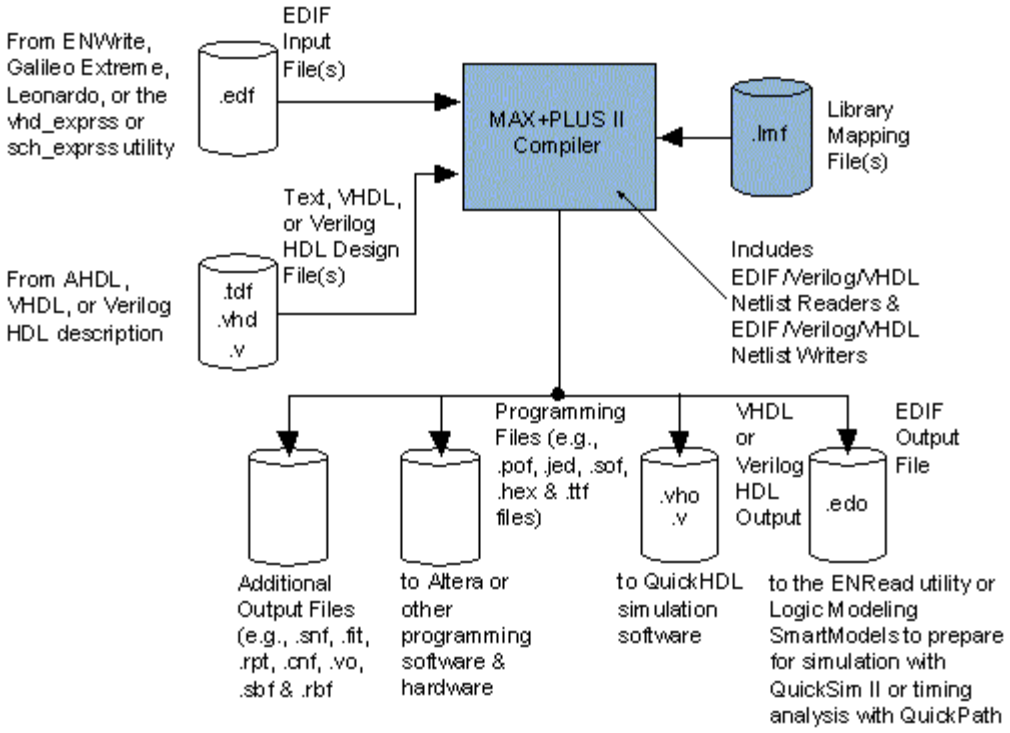
If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

The following figure shows the MAX+PLUS ® II/Mentor Graphics/Exemplar Logic project compilation flow.

Figure 1. MAX+PLUS II/Mentor Graphics/Exemplar Logic Project Compilation Flow

Alteraprovided items are shown in blue.



Feedback

Did this information help you?

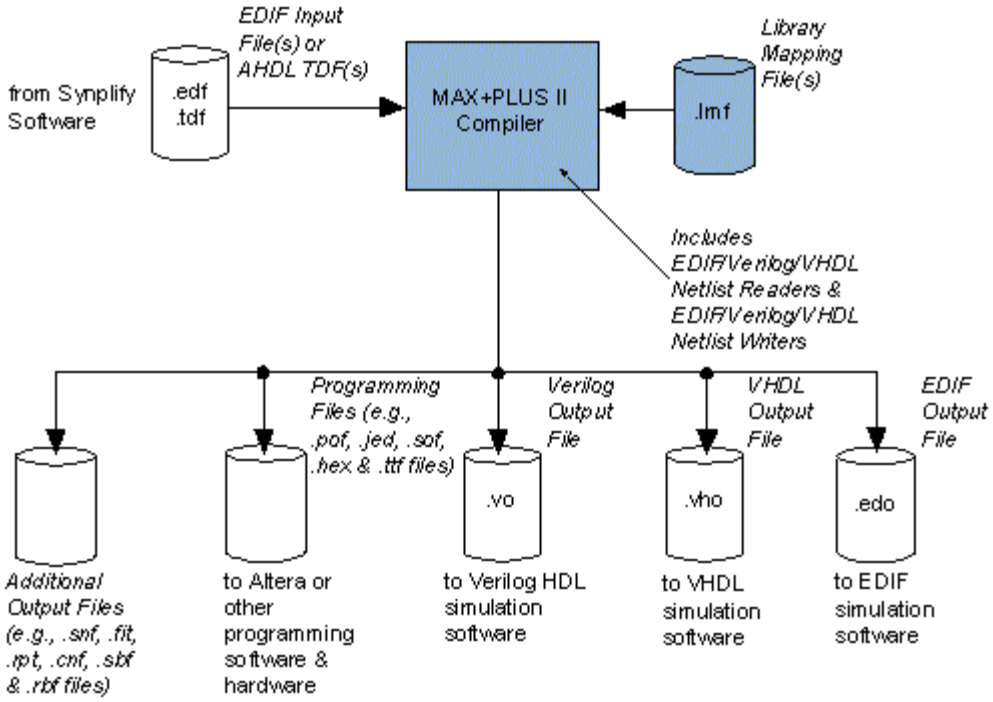
If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Figure 1 shows the MAX+PLUS® II/Synplicity project compilation flow.

Figure 1. MAX+PLUS II/Synplicity Project Compilation Flow

Altera-provided items are shown in blue.



Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Compiling Projects with MAX+PLUS II Software

The MAX+PLUS® II Compiler can process design files in a variety of formats. This topic describes how to use MAX+PLUS II software to compile projects in which the top-level design file is an EDIF Input File (with the extension **.edf**).

☞ Refer to the following sources for additional information:

-
- Go to MAX+PLUS II Help for information on compiling VHDL and Verilog HDL, design files directly with the MAX+PLUS II Compiler.
- Go to Running Synopsys Compilers from MAX+PLUS II Software for information on running the Synopsys Design Compiler or FPGA Compiler software on a VHDL or Verilog HDL design from within the MAX+PLUS II Compiler window.

To compile a design (also called a "project") with MAX+PLUS II software, go through the following steps:

1.
2. Create design files that are compatible with the MAX+PLUS II software and convert them into EDIF Input Files with the extension **.edf**. Specific instructions for some tools are described in these MAX+PLUS II ACCESS℠ Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your design entry or synthesis and optimization tool.
3. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, create a mapping for each function in a Library Mapping File (**.lmf**) to map the custom symbol to the corresponding EDIF Input File, AHDL Text Design File (**.tdf**), or other MAX+PLUS II-supported design file. These custom functions are represented in design files as hollow-body symbols or "black box" HDL descriptions.

   ☞ Go to "Library Mapping Files (**.lmf**)" in MAX+PLUS II Help for more information.
4. Open MAX+PLUS II and specify the name of your top-level design file as the project name with the **Project Name** command (File menu). If you open an HDL file in the MAX+PLUS II Text Editor, you can choose the **Project Set Project to Current File** command (File menu) instead.

   ☞ You can also compile a project from a command line. However, the first time you compile a project, the settings you need to specify are easier to specify from within the MAX+PLUS II software. After you have run the graphical user interface for the MAX+PLUS II software at least once, you can more easily use the command-line **setacf** utility to modify options in the Assignment & Configuration File (**.acf**) for the project. Type `setacf -h` ↵ and `maxplus2 -h` ↵ for descriptions of **setacf** and MAX+PLUS II command-line syntax.
5. Choose **Device** (Assign menu) and select the target Altera device family in the *Device Family* drop-down list box. If you wish to implement the design logic in a specific device, select it in the *Devices* box. Otherwise, select *AUTO* to allow the MAX+PLUS II Compiler to choose the best device(s) in the current device family. If your design entry or synthesis and optimization tool required you to specify a target family and/or device, specify the same information in this dialog box. For information on partitioning logic among multiple devices, go to MAX+PLUS II Help. Choose **OK**.

6. Open the Compiler window by choosing the **Compiler** command (MAX+PLUS II menu). Go through the following steps to specify the options necessary to compile the design file(s) in your project:

    1.

    2. Ensure that all EDIF netlist files have the extension **.edf** and choose **EDIF Netlist Reader Settings** (Interfaces menu).

    3. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor. This name should be the name of the vendor whose tool(s) you used to create the EDIF netlist files. If your vendor name does not appear, select *Custom* instead.

        If you are compiling a design created with Synopsys FPGA Express software, select *Synopsys*, choose the **Customize** button, enter <*project name*>**.lmf** in the *LMF #1* box, choose **OK**, and skip to step 6.

    4. If you selected an existing vendor name in the *Vendor* box and your project contains design files that require custom LMF mappings, choose the **Customize** button to expand the dialog box to show all settings. Turn on the *LMF #2* checkbox and type your custom LMF's filename in the corresponding text box, or select a name from the *Files* box. The selection in the *Vendor* box will change to *Custom* and all settings will be retained until you change them again.

    5. If you selected *Custom* in the *Vendor* box, choose the **Customize** button to expand the dialog box to show all settings. Any previously defined custom settings will be displayed. Under *Signal Names*, type one or more names with up to 20 total name characters in the *VCC* or *GND* box if your EDIF Input File(s) use one or more names other than `VCC` or `GND` for the global high or low signals. Multiple signal names must be separated by either a comma (,) or a space. Under *Library Mapping Files*, turn on the *LMF #1* checkbox and type a filename in the text box following it, or select a name from the *Files* box. If necessary, specify another LMF name in the *LMF #2* box. Go to MAX+PLUS II Help for detailed information on the settings available in the **EDIF Netlist Reader Settings** dialog box.

    6. Choose **OK**.

7. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, you may need to ensure that all files are present in your project directory, i.e., the same directory as the top-level design file. Otherwise, you must specify the directories containing these files as user libraries with the **User Libraries** command (Options menu).

8. Follow all guidelines that apply to your design entry or synthesis and optimization tool:

    - Exemplar Logic Galileo Extreme-Specific Compiler Settings
    - Synopsys DesignWare-Specific Compiler Settings
    - Converting Synopsys FPGA Compiler & Design Compiler Timing Constraints into MAX+PLUS II-Compatible Format with the **syn2acf** Utility
    - Synplicity Synplify-Specific Compiler Settings

9. If you wish to generate EDIF, VHDL, or Verilog HDL output files for post-compilation simulation or timing analysis with another EDA tool, go through the following steps:

    1.

    2. (Optional) Turn on the **Optimize Timing SNF** command (Processing menu) to reduce the size of the output file(s). Turning on this command can reduce the size of output netlists by up to 30%.

        This command does not create optimized timing SNFs on UNIX workstations. However, a non-optimized timing SNF provides the same functional and timing information as an optimized

timing SNF.

3. If you wish to generate EDIF Output Files (**.edo**), go through these steps:

    1.

    2. Turn on the **EDIF Netlist Writer** command (Interfaces menu). Then choose the **EDIF Netlist Writer Settings** command (Interfaces menu).

    3. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor and choose **OK**. If your vendor name does not appear, select *Custom* instead and specify the settings that are appropriate for your simulation or timing analysis tool. Go to MAX+PLUS II Help for detailed information on the options available in the **EDIF Netlist Writer Settings** dialog box.

    4. To generate an optional Standard Delay Format (SDF) Output File (**.sdo**), choose the **Customize** button to expand the dialog box to show all settings. Select one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**.

    The filenames of the EDIF Output File(s) and optional SDF Output File(s) are the same as the user-defined chip name(s) for the project; if no chip names exist, the Compiler assigns filenames that are based on the project name. For a multi-device project, the Compiler also generates a top-level EDIF Output File that is uniquely identified by "**_t**" appended to the project name. In addition, the Compiler automatically generates a VHDL Memory Model Output File, *<project name>***.vmo**, when it generates an EDIF Output File that contains memory (RAM or ROM).

4. If you wish to generate VHDL Output Files (**.vho**), turn on the **VHDL Netlist Writer** command (Interfaces menu). Then choose **VHDL Netlist Writer Settings** command (Interfaces menu). Select *VHDL Output File (.vho)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF ver. 2.1 files contain timing delay information that allows you to perform back-annotation simulation in VHDL with VITAL-compliant simulation libraries. The VHDL Output Files generated by the Compiler have the extension **.vho**, but are otherwise named in the same way as the EDIF Output Files described above.

5. If you wish to generate Verilog HDL Output Files (**.vo**), turn on the **Verilog Netlist Writer** command (Interfaces menu). Then choose **Verilog Netlist Writer Settings** command (Interfaces menu). Select *Verilog Output File (.vo)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF Output Files contain timing delay information that allows you to perform back-annotation simulation in Verilog HDL. The Verilog Output Files generated by the Compiler have the extension **.vo**, but are otherwise named in the same way as the EDIF Output Files described above.

10. To run the MAX+PLUS II Compiler, choose the **Project Save & Compile** command (File menu) or choose the **Start** button in the Compiler window.

    See step 3 for information on running MAX+PLUS II software from the command line.

11. Once you have compiled the project with the MAX+PLUS II Compiler, you can use the VHDL, Verilog HDL, or EDIF output file(s), and the optional SDF Output File(s) (**.sdo**) to perform timing analysis or timing simulation with another EDA tool. Specific instructions for some tools are described in these MAX+PLUS II ACCESS Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your EDA tool.

The MAX+PLUS II Compiler also generates a Report File (**.rpt**), a Pin-Out File (**.pin**), and one or more of the following files for device programming or configuration:

- JEDEC Files (**.jed**)
- Programmer Object Files (**.pof**)

- SRAM Object Files (**.sof**)
- Hexadecimal (Intel-format) Files (**.hex**)
- Tabular Text Files (**.ttf**)

Related Links:

- Go to *Compiler Procedures* in MAX+PLUS II Help for information on other available Compiler settings.
- Go to *Programmer Procedures* in MAX+PLUS II Help for instructions on creating other types of programming files and on programming or configuring Altera devices.
- Go to Back-Annotating MAX+PLUS II Pin Assignments to Design Architect Symbols for information on back-annotating pin assignments in Mentor Graphics Design Architect schematics.
- Go to Programming Altera Devices for information on the different programming hardware options for Altera device families.
- Go to the following topics for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Synplicity Synplify-Specific Compiler Settings

If you are using the MAX+PLUS® II Compiler to compile a design that has been synthesized and optimized with Synplify software, go through the following additional compilation steps:

1. Choose **Global Project Logic Synthesis** (Assign menu) to open the **Global Project Logic Synthesis** dialog box.
2. Select the appropriate logic synthesis style under the *Global Project Synthesis Style*:

   ✔ If you turned on the *Map Logic to LCELLs* option in the Synplify **Set Device Options** dialog box when synthesizing a FLEX® device design with Synplify software, select *WYSIWYG* or *Fast* in the *Global Project Synthesis Style* box.

   or:

   ✔ If you did not turn on the *Map Logic to LCELLs* option in the Synplify **Set Device Options** dialog box when synthesizing your design with Synplify software, or if you are using a MAX® or Classic™ device, select *Normal* in the *Global Project Synthesis Style* box.
3. For FLEX devices, choose **Define Synthesis Style** to display the **Define Synthesis Style** dialog box. Choose **Advanced Options** to display the **Advanced Options** dialog box and turn off the *NOT Gate Push-Back* option. Choose **OK** twice to close the dialog box.
4. Choose **OK** to close the **Global Project Logic Synthesis** dialog box.
5. Continue with the steps necessary to compile your project, as described in Compiling Projects with MAX+PLUS II Software.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Using Cadence Concept & MAX+PLUS II Software

The following topics describe how to use the Cadence Concept software with MAX+PLUS® II software. Click on one of the following topics for information:

This file is suitable for printing only. It does not contain hypertext links that allow you to jump from topic to topic.

Setting Up the MAX+PLUS II/Cadence Working Environment

- Software Requirements
- Setting Up the MAX+PLUS II/Cadence Concept Work Environment for a Sun SPARCstation Running SunOS Software
- MAX+PLUS II Directory Structure
- MAX+PLUS II/Cadence Interface File Organization
- Concept & RapidSIM Local Work Area Directory Structure
- Concept & HDL Direct Project Directory Structure
- Altera-Provided Logic & Symbol Libraries

Design Entry

- Design Entry Flow
- Creating Concept Schematics for Use with MAX+PLUS II Software
  - Instantiating the clklock Megafunction in Concept Schematics
  - Instantiating LPM & Other Parameterized Functions in Concept Schematics
- Entering Resource Assignments
  - Assigning Pins, Logic Cells & Chips
  - Assigning Cliques
  - Assigning Logic Options
  - Modifying the Assignment & Configuration File with the **setacf** Utility
- Creating Hierarchical Projects in Concept Schematics
- Converting Concept Schematics into MAX+PLUS II-Compatible EDIF Netlist Files with the **concept2alt** Utility

Functional Simulation

- Performing a Functional Simulation of a Concept Schematic with the **hdlconfig** Utility & Verilog-XL Software
- Performing a Functional Simulation of a Concept Schematic with VerilogLink & Verilog-XL Software

Related Topics:

Go to the following MAX+PLUS II ACCESS℠ Key topics for related information:

- Compiling Projects with MAX+PLUS II Software
- Programming Altera Devices

Go to the following topics, which are available on the web, for additional information:

- MAX+PLUS II Development Software
- Altera Programming Hardware
- Cadence web site (http://www.cadence.com)

Setting Up the MAX+PLUS II/Cadence Working Environment

To use MAX+PLUS® II software with Cadence software, you must first install the MAX+PLUS II software, then establish an environment that facilitates entering and processing designs. The MAX+PLUS II/Cadence interface is installed automatically when you install the MAX+PLUS II software on your computer. Go to *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual for more information on installation and details on the directories that are created during MAX+PLUS II installation. Go to MAX+PLUS II/Cadence Interface File Organization for information about the MAX+PLUS II/Cadence directories that are created during MAX+PLUS II installation.

The information presented here assumes that you are using the C shell and that your MAX+PLUS II system directory is **/usr/maxplus2**. If not, you must use the appropriate syntax and procedures to set environment variables for your shell.

To set up your working environment for the MAX+PLUS II/Cadence interface, follow these steps:

1. Ensure that you have correctly installed the MAX+PLUS II and Cadence software versions described in the MAX+PLUS II/Cadence Software Requirements.
2. Add the following environment variables to your **.cshrc** file:

   ```
   setenv ALT_HOME /usr/maxplus2 ↵

   setenv CDS_INST_DIR <Cadence system directory path> ↵
   ```

3. Add the **$ALT_HOME/cadence/bin** and **$CDS_INST_DIR/tools/bin** directories to the `PATH` environment variable in your **.cshrc** file. Make sure these paths are placed before the Cadence hierarchy path.
4. Add **/usr/dt/lib** and **/usr/ucb/lib** to the `LD_LIBRARY_PATH` environment variable in your **.cshrc** file.
5. Create a new **cds.lib** file in your working directory or edit an existing one so that it includes all of the following lines that apply to the Cadence tools you have installed:

   ```
   DEFINE alt_syn ${ALT_HOME}/simlib/concept/alt_syn

   DEFINE lpm_syn ${ALT_HOME}/simlib/concept/lpm_syn

   DEFINE alt_lpm ${ALT_HOME}/simlib/concept/alt_lpm

   DEFINE alt_mf ${ALT_HOME}/simlib/concept/alt_mf

   DEFINE alt_max2 ${ALT_HOME}/simlib/concept/alt_max2

   DEFINE alt_max2 ${ALT_HOME}/simlib/composer/alt_max2/alt_max2

   DEFINE alt_vtl $ALT_HOME/simlib/concept/alt_vtl/lib
   ```

```
DEFINE altera $ALT_HOME/simlib/concept/alt_mf/lib

SOFTINCLUDE $CDS_INST_DIR/tools/leapfrog/files/cds.lib

DEFINE <design name>.
```

6. Copy the **/usr/maxplus2/maxplus2.ini** file to your **$HOME** directory:

```
cp /usr/maxplus2/maxplus2.ini $HOME↵

chmod u+w $HOME/maxplus2.ini↵
```

The **maxplus2.ini** file contains both Altera- and user-specified initialization parameters that control the MAX+PLUS II software, such as MAX+PLUS II symbol and logic function library paths and the current project name. The MAX+PLUS II installation procedure creates and copies the **maxplus2.ini** file to the **/usr/maxplus2** directory.

Normally, you do not have to edit your local copy of **maxplus2.ini** because the MAX+PLUS II software updates the file automatically whenever you change any parameters or settings. However, if you move the **max2lib** and **max2inc** library subdirectories, you must update the file. Go to "Creating & Using a Local Copy of the **maxplus2.ini** File" in MAX+PLUS II Help for more information.

7. If you are using Concept on a Sun SPARCstation running SunOS, go to Setting Up the MAX+PLUS II/Cadence Concept Work Environment for a Sun SPARCstation Running SunOS Software to install the **redifnet** EDIF netlist reader utility.

8. If you are using Synergy software, edit the **hdl.var** file located in your working directory to include the following line:

```
DEFINE work <design name>↵
```

9. Set up an appropriate directory structure for the tool(s) you are using. See the following topics for information:

    - Composer Project File Directory Structure
    - Concept & RapidSIM Local Work Area Directory Structure

Related Topics:

Go to the following topics, which are available on the web, for additional information:

- ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
- This manual is also available in 4 parts:
    - Preface & Section 1: MAX+PLUS II Installation
    - Section 2: MAX+PLUS II - A Perspective
    - Section 3: MAX+PLUS II Tutorial
    - Appendices, Glossary & Index

Setting Up the MAX+PLUS II/Cadence Concept Work Environment for a Sun SPARCstation Running SunOS Software

If you are using Concept software on a Sun SPARCstation running SunOS software, you should also install the **redifnet** EDIF netlist reader utility to convert Concept schematics into MAX+PLUS II-compatible EDIF netlist

files. To install the **redifnet** utility, follow these steps:

1. Copy the **redifnet** directory from the **/usr/maxplus2/simlib/concept/edifnet** directory to the Cadence system directory.
2. Copy the **redifnet** and **pinmap_start** files from the **/usr/maxplus2/simlib/concept/edifnet/bin** directory to the /<*Cadence system directory path*>**/tools/bin**.
3. Specify the **-/usr/maxplus2/simlib/concept/edifnet/max2sim** map file as a `PIN_MAP_FILE` in the **redifnet.cmd** file.
4. (Optional) Modify existing templates for directive files such as **compiler.cmd**, **vloglink.cmd**, and **global.cmd**. These templates are located in the **/usr/maxplus2/simlib/concept/edifnet/templates** directory.
5. (Optional) Modify the **expansion.dat** and **max2sim.map** files in the **/usr/maxplus2/simlib/concept/edifnet** directory.

MAX+PLUS II Directory Structure

In the MAX+PLUS® II software, a project name is the name of a top-level design file, without the filename extension. This design file can be an EDIF, Verilog HDL, or VHDL netlist file; an AHDL Text Design File (TDF); or any other MAX+PLUS II-supported design file. The EDIF netlist file must be created by the **altout** or **concept2alt** utility and imported into the MAX+PLUS II software as an EDIF Input File (**.edf**).

Project design files and output files are stored in the project directory, with the exception of standard library functions provided by Altera or another EDA tool vendor. The MAX+PLUS II software stores the connectivity data on the links between design files in a hierarchical project in a Hierarchy Interconnect File (**.hif**), but refers to the entire project only by its project name. The MAX+PLUS II Compiler uses the HIF to build a single, fully flattened project database that integrates all design files in a project hierarchy.

MAX+PLUS II/Cadence Interface File Organization

Table 1 shows the MAX+PLUS® II/Cadence interface subdirectories that are created in the MAX+PLUS II system directory (by default, the **/usr/maxplus2** directory) during MAX+PLUS II installation. For information on the other directories that are created during MAX+PLUS II installation, see "MAX+PLUS II File Organization" in *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual.

Table 1. MAX+PLUS II Directory Organization

Directory	Description
./lmf	Contains the Altera-provided Library Mapping File, **cadence.lmf**, that maps Cadence logic functions to equivalent MAX+PLUS II logic functions.
./examples/cadence	Contains the sample files for Cadence software discussed in these ACCESS℠ Key Guidelines.
./cadence	Contains the AMPLE userware for the MAX+PLUS II/Cadence interface.
./simlib/concept/alt_max2	Contains the MAX+PLUS II primitives, including `CARRY`, `CASCADE`, `EXP`, `GLOBAL`, `LCELL`, `SOFT`, `OPNDRN`, `DFFE` (D flipflop with Clock Enable), and `DFFE6K` (D flipflop with Clock Enable and both Clear and Preset for FLEX® 6000 devices only) for use with Concept software.
	Contains the MAX+PLUS II primitives, including `CARRY`, `CASCADE`, `EXP`,

./simlib/composer/alt_max2	GLOBAL, LCELL, SOFT, OPNDRN, DFFE (D flipflop with Clock Enable), and DFFE6K (D flipflop with Clock Enable and both Clear and Preset for FLEX 6000 devices only) for use with Composer software.
./simlib/concept/alt_lpm	Contains the MAX+PLUS II megafunctions, including library of parameterized modules (LPM) functions, for use with Concept software.
./simlib/concept/max2sim	Contains the MAX+PLUS II/Concept simulation model library, **max2_sim**, for use with RapidSIM software.
./simlib/concept/alt_syn	Contains the MAX+PLUS II synthesis library, **alt_syn**, for use with Synergy and Concept software, and the **vlog2alt** utility.
./simlib/composer/alt_syn	Contains the MAX+PLUS II synthesis library, **alt_syn**, for use with Synergy and Composer software.
./simlib/concept/lpm_syn	Contains the Cadence LPM library, **lpm_syn**, for use with Synergy and Concept software.
./simlib/composer/lpm_syn	Contains the Cadence LPM library, **lpm_syn**, for use with Synergy and Composer software.
./simlib/concept/alt_mf	Contains the MAX+PLUS II VHDL logic function library. (a_8count is for the MAX® 7000 and MAX 9000 device families only.)
./simlib/concept/edifnet/templates	Contains template files for Concept directives, i.e., **global.cmd**, **compiler.cmd**, **vloglink.cmd**, **verilog.cmd**, and **master.local**.
./simlib/concept/alt_max2/verilogUdps	Contains Verilog HDL modules that are the equivalent of the primitives contained in **alt_max2** library for use with Concept software.
./simlib/composer/alt_max2/verilogUdps	Contains Verilog HDL modules that are the equivalent of the primitives contained in **alt_max2** library for use with Composer software.
./simlib/concept/alt_vtl **./simlib/composer/alt_vtl**	Contains VITAL library source files for use with Concept or Composer software.
./simlib/composer/alt_max2/verilog	Contains simulation modules for all symbols in the **alt_max2** Composer library.

Related Topics:

Go to the following topics, which are available on the web, for additional information:

- ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
- This manual is also available in 4 parts:
  - Preface & Section 1: MAX+PLUS II Installation
  - Section 2: MAX+PLUS II - A Perspective
  - Section 3: MAX+PLUS II Tutorial
  - Appendices, Glossary & Index
- FLEX Devices
- MAX Devices
- Classic Device Family

Concept & RapidSIM Local Work Area Directory Structure

When the **redifnet** utility imports an EDIF netlist file for the RapidSIM software, it creates a SCALD directory for your project. However, creating this directory may overwrite the directory that was created for the original Concept schematic. To prevent overwriting this directory, you should create a file structure that helps you manage your design files.

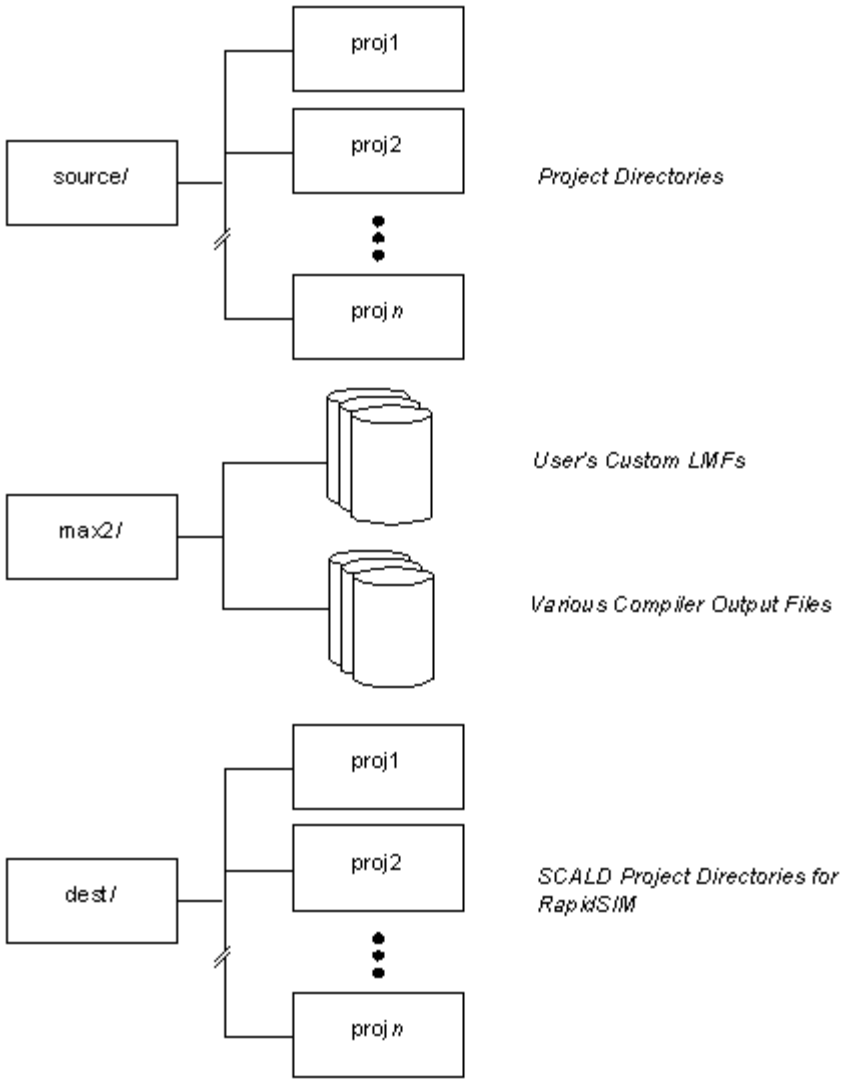
Altera recommends that you create the following three directories for your design files.

Directory: Description:

./source	Create Concept schematics and generate EDIF netlist files with the **wedifnet** utility in the **source** directory.
./max2	Copy the EDIF Input File (**.edf**) from the **source** directory to this directory to compile the file with the MAX+PLUS® II software.
./dest	Copy the EDIF Output File (**.edo**) from the **max2** directory to this directory to run the **redifnet** and RapidSIM software.

Copies of the appropriate directives files for Cadence tools must be present in both the **source** and **dest** directories. Figure 1 shows Altera's recommended file structure.

Figure 1. Recommended File Structure



Concept & HDL Direct Project Directory Structure

Concept software generates the following files for each schematic:

- *<drawing name>***/logic.1.1**
- *<drawing name>***/logic_bn.1.1**
- *<drawing name>***/logic_cn.1.1**
- *<drawing name>***/logic_dp.1.1**

For designs that use HDL Direct software, Concept software also generates the following files:

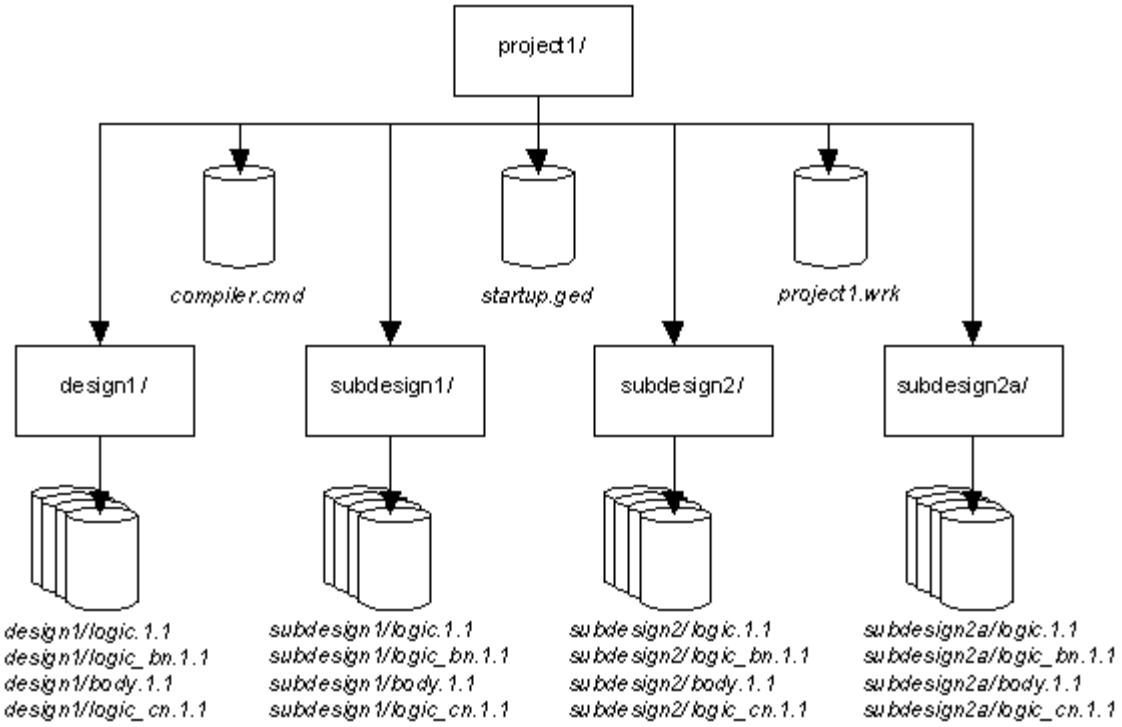
- <*drawing name*>**/logic_dp.1.1**
- <*drawing name*>**/logic_vd.1.1**
- <*drawing name*>**/logic/verilog.v**
- <*drawing name*>**/logic/vhdl.vhd**
- <*drawing name*>**/logic/hdldirect.dat**
- <*drawing name*>**/entity/vhdl.vhd**

These files are stored in their own <*drawing name*> directories. However, hierarchical relationships between files are not reflected in the file directory structure.

The local SCALD directory has an entry for all <*drawing name*> directories. Cadence software automatically manages drawing storage and retrieval operations through this special directory. The SCALD directory should have the same name as the UNIX project directory, but with the extension **.wrk**. Figure 1 shows a sample file structure, with **project1** as the UNIX project directory, and **project1.wrk** as the SCALD directory.

When the **concept2alt** utility converts the schematic into an EDIF netlist file, it processes the design information and all related file subdirectories, then creates the EDIF netlist file in the directory defined by the user. The EDIF netlist file is named <*project name*>**.edf**, where <*project name*> is the name of the top-level design file. Figure 1 shows the Cadence project file structure.

Figure 1. Cadence Project File Structure



Altera-Provided Logic & Symbol Libraries

The MAX+PLUS® II/Cadence environment provides four logic and symbol libraries that are used for compiling, synthesizing, and simulating designs.

You can create your own libraries of custom symbols and logic functions in Concept and Composer. You can use custom symbols to incorporate an EDIF Input File, Text Design File (TDF), or any other MAX+PLUS II-

supported design file into a project. MAX+PLUS II uses the **cadence.lmf** Library Mapping File to map standard Concept or Composer symbols to equivalent MAX+PLUS II logic functions. To use custom symbols, you can create a custom LMF that maps your custom symbols to the equivalent MAX+PLUS II-supported design file. You must also specify the directory that contains the MAX+PLUS II-supported design file(s) as a user library with the MAX+PLUS II **User Libraries** command (Options menu). Go to "Library Mapping File" and "Cadence Library Mapping File (**cadence.lmf**)" in MAX+PLUS II Help for more information.

The alt_max2 Library

You can enter a Concept or Composer Design Architect schematic with primitives and macrofunctions from the Altera-provided symbol library **alt_max2**. The **alt_max2** library includes 74-series macrofunctions and several MAX+PLUS II primitives with corresponding Verilog HDL simulation models for controlling design synthesis and fitting. It also includes four macrofunctions--`a_8count`, `a_8mcomp`, `a_8fadd`, and `a_81mux`--that are optimized for different device families, and the `clklock` phase-locked loop megafunction, which is supported by some FLEX® 10K devices, with corresponding Verilog HDL and VHDL simulation models. See Table 1. Choose **Old-Style Macrofunctions** and/or **Primitives** from the MAX+PLUS II Help menu for more information on functions in the **alt_max2** library.

The alt_lpm Library

The Altera-provided **alt_lpm** library, which is available for Concept and Verilog HDL designs, includes standard functions from the library of parameterized modules (LPM) 2.1.0, except the truth table, finite state machine, and pad functions. Other parameterized functions, including cycle-shared FIFO (`csfifo`) and cycle-shared dual-port RAM (`csdpram`) are also included. The LPM standard defines a set of parameterized modules (i.e., parameterized megafunctions) and their corresponding representations in an EDIF netlist file. These logic functions allow you to create and functionally simulate an LPM-based design without targeting a specific device family. The parameters you specify for each LPM function determine the simulation models that will be generated. After the design is completed, you can target the design to any device family. In designs created with Concept, the Altera **alt_lpm** library works only with HDL Direct and the **hdlconfig** utility. Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu for more information about LPM functions in the **alt_lpm** library.

The lpm_syn Library

The **lpm_syn** library contains the Altera-provided parameterized functions. The **lpm_syn** library is similar to the **alt_lpm** library, except that it contains VHDL and Verilog HDL logic functions for use with Synergy, Concept, and Composer software.

The alt_mf Library

Altera provides a VHDL logic function library, **alt_mf**, that currently includes four macrofunctions--`a_8count`, `a_8mcomp`, `a_8fadd`, and `a_81mux`--for controlling design synthesis and fitting. These elements can be instantiated directly in your VHDL file. To designate that these logic functions should pass untouched through the EDIF netlist file to the MAX+PLUS II Compiler, you must select the *Maintain* attribute constraint for instances of these functions before running the Synergy software. These models allow you to perform functional VHDL simulation while maintaining an architecture-independent VHDL description.

Table 1 shows the MAX+PLUS II-specific logic functions.

Table 1. MAX+PLUS II-Specific Logic Functions

Macrofunctions *Note (1)*				**Primitives**	
Name	**Description**	**Name**	**Description**	**Name**	**Description**
`8fadd`	8-bit full adder	`LCELL`	Logic cell buffer	`EXP`	MAX® 5000, MAX 7000, and MAX 9000 Expander buffer

8mcomp	8-bit magnitude comparator	GLOBAL	Global input buffer	SOFT	Soft buffer
8count *Note (2)*	8-bit up/down counter	CASCADE	FLEX 6000, FLEX 8000, and FLEX 10K cascade buffer	OPNDRN	Open-drain buffer
81mux	8-to-1 multiplexer	CARRY	FLEX 6000, FLEX 8000, and FLEX 10K carry buffer	DFFE DFFE6K *Note (3)*	D-type flipflop with Clock Enable
clklock	Phase-locked loop				

Notes:

1. Logic function names that begin with a number must be preceded by "a_" in VHDL designs. For example, 8fadd must be specified as a_8fadd.
2. The a_8count logic function is for the MAX 7000 and MAX 9000 device families only.
3. For designs that are targeted to FLEX 6000 devices, you should use the DFFE primitive only if the design contains either a Clear or Preset signal, but not both. If your design contains both a Clear and a Preset signal, you must use the DFFE6K primitive.

Related Topics:

Go to the following topics, which are available on the web, for additional information:

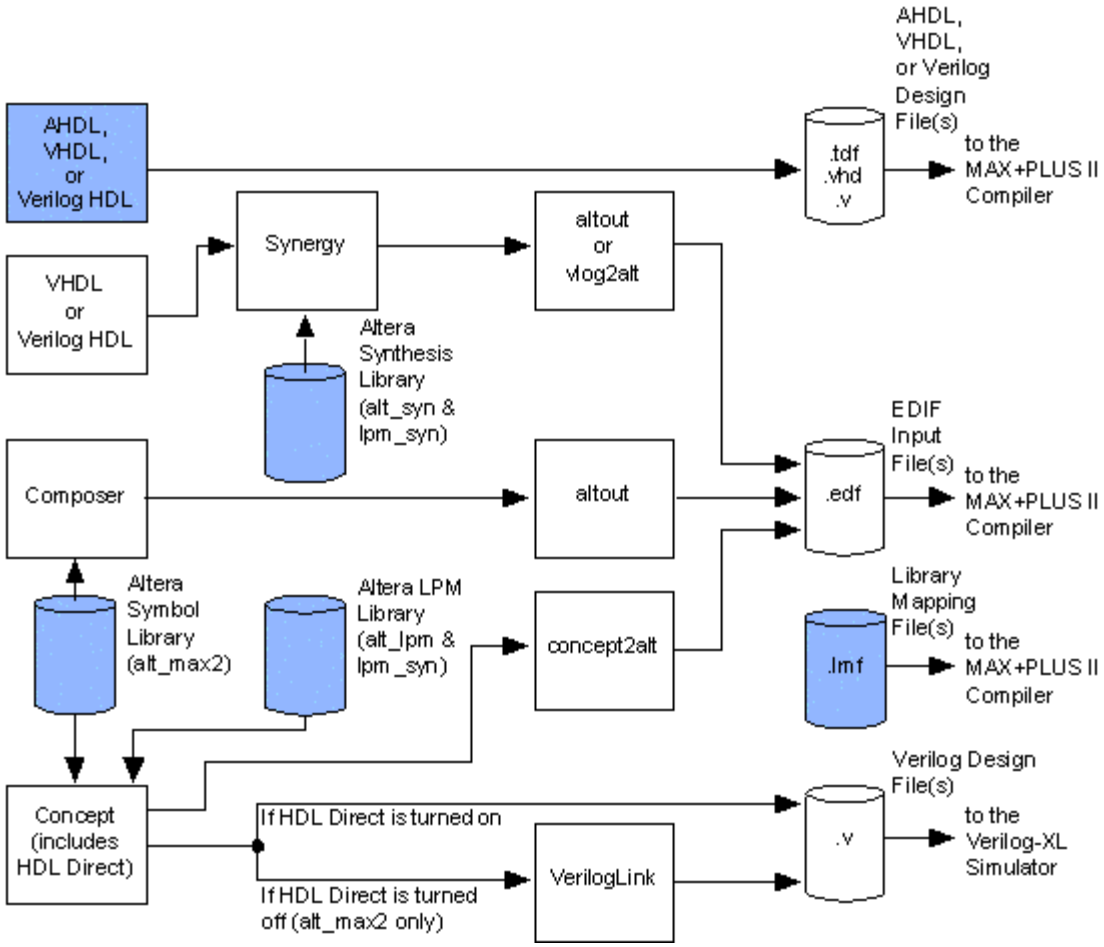
- FLEX Devices
- MAX Devices
- Classic Device Family

Cadence Design Entry Flow

Figure 1 shows the design entry flow for the MAX+PLUS® II/Cadence interface.

Figure 1. MAX+PLUS II/Cadence Design Entry Flow

Altera-provided items are shown in blue.



Creating Concept Schematics for Use with MAX+PLUS II Software

You can create Concept schematics and convert them to EDIF Input Files (**.edf**) that can be processed with the MAX+PLUS® II Compiler. To create a Concept schematic for use with the MAX+PLUS II software, go through the following steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment.

2. Make sure the required directive files are in the /<*working directory*>/<*design name*>/**source** directory. If not, you can use the Altera-provided template files located in the following directories:

   - **/usr/maxplus2/simlib/concept/edifnet/templates**
   - **/usr/maxplus2/simlib/concept/edifnet/redifnet**

3. Start the Concept schematic editor by typing `concept` <*design name*> ↵ at a UNIX prompt from the /<*working directory*>/**source** directory. Use the graphical user interface to structure and organize your files to create an environment that facilitates entering and processing designs. Go to Concept & RapidSIM Local Work Area Directory Structure for more information on directories in Concept.

4. To write a Verilog HDL text file whenever the design is saved, choose the **Block** button in the Concept window.

   ☞ To use the HDL Direct utility to process your design, turn on the *HDL Direct On* option in the Concept window. Go to Concept & HDL Direct Project Directory Structure for information on the files generated by Concept software when using the HDL Direct utility.

5. Enter primitives, megafunctions, and macrofunctions from the following Altera-provided component libraries:

   - **alt_max2** includes macrofunctions, megafunctions, and primitives.
   - **alt_lpm** includes library of parameterized modules (LPM) functions (available only if you use HDL Direct software).

   See the following topics for instructions for specific functions:

   - Instantiating LPM & Other Parameterized Functions in Concept Schematics
   - Instantiating the clklock Megafunction in Concept Schematics

   You can instantiate MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.

6. If you wish to create a hierarchical design that contains symbols representing other design files, such as Altera® Hardware Description Language (AHDL) Text Design Files, go to Creating Hierarchical Projects in Concept Schematics.

7. Enter meaningful instance names for all symbols and functions so that you can easily trace internal node names during simulation and debugging operations. For example, if an `a161` macrofunction is instantiated several times in one design, you should define a unique name for each instance. The instance name for each symbol is controlled by `INST` property. For more information on assigning properties, refer to the Cadence ***Concept Schematic User Guide***.

8. Enter input, output, and bidirectional ports:

   - If you turned on the *HDL Direct On* option in step 4, add `inport` and `outport` symbols from the **hdl_direct_lib** library to the interface symbols.

   - If you are not using HDL Direct, use `flag` symbols from the **standard** library to indicate input, output, and bidirectional ports. Be sure to end pin names with `¥I` to identify them as interface signals.

   If a pin is not used, leave it floating. The **concept2alt** utility removes all unconnected pins when it generates an EDIF netlist file.

9. (Optional) To enter resource assignments in your Concept schematic, go to Entering Resource Assignments. You can also enter resource assignments from within the MAX+PLUS II software.

10. (Optional) Perform a functional simulation, as described in one of the following topics:

    - Performing a Functional Simulation of a Concept Schematic with the **hdlconfig** Utility & Verilog-XL Software
    - Performing a Functional Simulation of a Concept Schematic with VerilogLink & Verilog-XL Software

11. Use the **concept2alt** utility to generate an EDIF netlist file that can be imported into the MAX+PLUS II software, as described in Converting Concept Schematics into MAX+PLUS II-Compatible EDIF Netlist Files with the **concept2alt** utility.

12. Process the <*design name*>**.edf** file with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample Concept schematic files:

- **/usr/maxplus2/examples/cadence/example1/fulladd**
- **/usr/maxplus2/examples/cadence/example4/fulladd2**
- **/usr/maxplus2/examples/cadence/example6/fa2**
- **/usr/maxplus2/examples/cadence/example12/fifo**

Instantiating the clklock Megafunction in Concept Schematics

You can instantiate the `clklock` phase-locked loop megafunction, which is supported in selected FLEX® 10K devices, in a Concept schematic. that employ a phase-locked loop (PLL).

To instantiate the `clklock` megafunction in Cadence Concept schematics, follow these steps:

1. Choose the **Add Part** button from the toolbar or type `add` ↵ in the Concept window to open the Component Browser window.
2. Enter the `clklock` megafunction:
    1. Choose **alt_max2** (Library menu) and select *clklock* from the list box.
    2. Type `attribute`, then select the `clklock` component. Change the *CLOCKBOOST* and *INPUT_FREQUENCY* values as needed. For detailed information on the `clklock` megafunction, choose **Megafunctions/LPM** from the MAX+PLUS® II Help menu.
3. Choose **Done**.
4. Continue with the steps necessary to complete your Concept schematic, as described in Creating Concept Schematics for Use with MAX+PLUS II Software.

    Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample Concept schematic file, which includes `clklock` instantiation:

    - **/usr/maxplus2/examples/cadence/example12/fifo**

Related Topics:

Go to FLEX 10K Device Family, which is available on the web, for additional information.

Instantiating LPM & Other Parameterized Functions in Concept Schematics

You can use library of parameterized modules (LPM) functions and other Altera® -provided parameterized functions in Concept schematics if you also use the HDL Direct utility.

To instantiate LPM functions, go through the following steps:

1. Choose the **Add Part** button from the toolbar or type `add` from the Concept window to open the Component Browser window.
2. Choose **alt_lpm** (Library menu). All functions in the **alt_lpm** library are MAX+PLUS® II-compatible. Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu to get detailed information on all supported parameterized functions.

3. Type `attribute`, then click on each component to set parameters for each function. See General Guidelines below for additional information.

4. Add `inport` and `outport` symbols from the **hdl_direct_lib** library to the interface signals. Use the `supply_0` and `supply_1` symbols from the **hdl_direct_lib** library to connect a net to `GND` or `VCC`.

5. Continue with the steps necessary to complete your Concept schematic, as described in Creating Concept Schematics for Use with MAX+PLUS II Software.

   Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample Concept schematic file, which includes LPM function instantiation:

   - **/usr/maxplus2/examples/cadence/example12/fifo**

General Guidelines

- If a pin is not used, leave it floating. The **concept2alt** utility removes all unconnected pins when it generates an EDIF netlist file.
- For the `csfifo` function, the value of the `LPM_NUMWORDS` parameter must be between $2^{LPM_WIDTHAD-1}$ and $2^{LPM_WIDTHAD}$.
- Make sure that any hexadecimal (Intel-format) file (**.hex**) that you use to specify the initial content of a memory does not have the same name as the design file name.
- Make sure that all properties and value strings are in uppercase letters, except the filename specified with the `LPM_FILE` property, which should use the actual case of the filename.
- Choose the **Set** button in the Concept window and choose *CAPS_LOCK_OFF* for the *CAPS LOCK* option.
- Only the `LPM_POLARITY` parameter (which can be set to `INVERT` or `NORMAL`) can determine the polarity of the bus or pin. You can display a bubble in the Concept schematic to indicate an inverted pin by typing `BUBBLE` in the Concept command window and selecting the appropriate pin. However, the bubble does not determine the polarity of the pin or bus.
- Avoid using the **Replace** button in the Concept window to replace old symbols with new ones: you may accidentally set unwanted properties. Instead, you should use the **Delete** button to delete old symbols and the **Add** button to add new symbol(s).

Entering Resource Assignments

The MAX+PLUS® II software allows you to enter a variety of resource and device assignments for your projects. Resource assignments are used to assign logic functions to a particular pin, logic cell, I/O cell, embedded cell, row, column, Logic Array Block (LAB), Embedded Array Block (EAB), chip, clique, local routing, logic option, timing requirement, or connected pin group. In MAX+PLUS II software, you can enter all types of resource and device assignments with Assign menu commands. You can also enter pin, logic cell, I/O cell, embedded cell, LAB, EAB, row, and column assignments in the MAX+PLUS II Floorplan Editor. The Assign menu commands and the Floorplan Editor all save assignment information in the ASCII Assignment & Configuration File (**.acf**) for the project. In addition, you can edit ACFs manually in any standard text editor or with the **setacf** utility.

Concept & Composer Schematics

In both Concept and Composer schematics, you can assign a limited subset of these resource assignments by assigning properties to symbols. These properties are incorporated into the EDIF netlist file(s). The MAX+PLUS II

software automatically converts assignment information from the EDIF Input File into the ACF format. For information on making MAX+PLUS II-compatible resource assignments, go to the following topics:

- Assigning Pins, Logic Cells & Chips
- Assigning Cliques
- Assigning Logic Options
- Modifying the Assignment & Configuration File with the **setacf** Utility

Go to the ***Cadence Concept Schematic User Guide*** and ***Composer Reference User Guide*** for details on how to assign properties. Go to "Resource Assignments in EDIF Input Files" and "Assigning Resources in a Third-Party Design Editor" in MAX+PLUS II Help for more information on assignments or properties that can be assigned in Concept and Composer.

Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample Concept and Composer schematic files, which include resource assignments:

- **/usr/maxplus2/examples/cadence/example6/fa2** (Concept)
- **/usr/maxplus2/examples/cadence/example7/fa2** (Composer)

VHDL & Verilog HDL Design Files

For Verilog HDL- and VHDL-based designs, you must use the MAX+PLUS II software or the **setacf** utility to enter resource assignments. For information on using the **setacf** utility, go to Modifying the Assignment & Configuration File with the **setacf** Utility.

Related Topics:

For information on entering assignments in the MAX+PLUS II software with Assign menu commands or in an ACF, go to "resource assignments" or "ACF, format" in MAX+PLUS II Help using **Search for Help on** (Help menu).

Assigning Pins, Logic Cells & Chips

You can assign a single logic function to a specific pin or logic cell (including I/O cells and embedded cells) within a chip, and assign one or more functions to a specific chip. A chip is a group of logic functions defined as a single, named unit, which can be assigned to a specific device.

You can assign a signal to a particular pin to ensure that the signal is always associated with that pin, regardless of future changes to the project. If you wish to set and maintain the performance of your project, assigning logic to a specific logic cell within a chip can minimize timing delays. In a project that is partitioned among multiple devices, you can assign logic functions that must be kept together in the same device to a chip. Chip assignments allow you to split a project so that only a minimum number of signals travel between devices, and to ensure that no unnecessary device-to-device delays exist on critical timing paths. You can assign a chip to a device in some EDA tools or in the MAX+PLUS® II software.

Use the following syntax for chip, pin, and logic cell assignments:

- To assign a logic function to a chip:

  `CHIP_PIN_LC=<`*chip name*`>`

  For example: `CHIP_PIN_LC=chip1`

- To assign a pin number within a chip:

CHIP_PIN_LC=<*chip name*>@<*pin number*>

For example: CHIP_PIN_LC=chip1@K2

- To assign a logic cell, I/O cell, or embedded cell number:

  CHIP_PIN_LC=<*chip name*>@LC<*logic cell number*>

  CHIP_PIN_LC=<*chip name*>@IOC<*I/O cell number*>

  CHIP_PIN_LC=<*chip name*>@EC<*embedded cell number*>

  For example: CHIP_PIN_LC=chip1@LC44

Related Topics:

Refer to the following sources for additional information:

- Go to "Devices & Adapters" and "Assigning a Device" in MAX+PLUS II Help for information on device pin-outs and assigning devices, respectively, in the MAX+PLUS II software.
- Go to Back-Annotating MAX+PLUS II Pin Assignments to Design Architect Symbols for information on back-annotating pin assignments in Mentor Graphics Design Architect schematics.

Assigning Cliques

You can define a group of logic functions as a single, named unit, called a clique. The MAX+PLUS® II Compiler attempts to place all logic in the clique in the same logic array block (LAB) to ensure optimum speed. If the project does not use multi-LAB devices, or if it is not possible to fit all clique members into a single LAB, the clique assignment ensures that all members of a clique are placed in the same device. In FLEX® 6000, FLEX 8000, FLEX 10K, and MAX® 9000 devices the Compiler also attempts to place the logic in LABs in the same row. Cliques therefore allow you to partition a project so that only a minimum number of signals travel between LABs, and to ensure that no unnecessary LAB-to-LAB or device-to-device delays exist on critical timing paths.

To assign a clique, use the following syntax:

✔ CLIQUE=<*clique name*>

For example: CLIQUE=fast1

Related Topics:

- Assigning a Clique
- Guidelines for Achieving Maximum Speed Performance

Assigning Logic Options

Logic option and logic synthesis style assignments allow you to guide logic synthesis with logic optimization features that are specific to Altera® devices. You can assign logic options and styles to individual logic functions in your design. The MAX+PLUS® II Compiler also uses a device-family-specific default logic synthesis style for each project.

Related Topics:

Go to "Resource Assignments in EDIF Input Files" and "Assigning Resources in a Third-Party Design Editor" in MAX+PLUS II Help for complete and up-to-date information on logic option and logic synthesis style assignments, including definitions and syntax of these assignments.

Modifying the Assignment & Configuration File with the setacf Utility

Altera provides the **setacf** utility to help you modify a project's Assignment & Configuration File (**.acf**) from the command line, without opening the file with a text editor. Type `setacf -h` ↵ at a UNIX or DOS prompt to get help on this utility.

Creating Hierarchical Projects in Concept Schematics

If you wish to create a hierarchical design that contains symbols representing other MAX+PLUS II-supported design files, such as Altera® Hardware Description Language (AHDL) Text Design Files (**.tdf**), you can create a hollow-body symbol that represents a design file and then instantiate it in your Concept schematic. To create a hierarchical project in your Concept schematic, go through the following steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS® II/Cadence Working Environment.
2. Create a Concept schematic and save it in your working directory, as described in Creating Concept Schematics for Use with MAX+PLUS II Software.

You can instantiate MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.

3. Create the hollow-body symbol <*design name*> in Concept by typing the following command from the <*working directory*>/**source** directory that contains the lower-level design file <*design name*>.<*extension*>:

   `concept` <*design name*>`.body` ↵

4. Create a part file to indicate that the body is hollow:
   1. Add the `DEFINE` and `DRAWING` bodies to the part drawing. These bodies should be the only two bodies in the drawing.
   2. Add the `TITLE=`<*design name*> and the `ABBREV=`<*design name*> properties to the `DRAWING` body to identify the drawing.
   3. Save the part drawing with the name <*design name*>**.part.1.1**.
5. Regardless of the hardware description language (HDL) or schematic editor used to create the design, you must create a dummy Verilog HDL module to indicate to the **concept2alt** utility that the design is a "black box" that must pass untouched through the EDIF netlist file.
   1. Type `genview verilog` ↵ in the Concept window.
   2. Type `logic` ↵ when prompted for the Verilog *View* name.

3. If you are using VerilogLink, you must type `genview verilog` again, then type `verilog_lib` ↵ when prompted for the Verilog *View* name.

4. Type `cd` <*design name*>`/logic` ↵ at the UNIX prompt from the **/source** directory to change to the **/source/**<*design name*>**/logic** directory.

5. Edit the **verilog.v** file to add the `cds_action = "ignore"` parameter setting after the Input Declarations and Output Declarations sections. This parameter setting specifies that the <*design name*> should be treated as a "black box."

6. To enter the symbol in the higher-level Concept schematic, choose the **Add Part** button, choose the name of the working **SCALD** directory, then choose the <*design name*> symbol from the Symbol menu.

7. The MAX+PLUS II software uses the **cadence.lmf** Library Mapping File to map Concept symbols to equivalent MAX+PLUS II logic functions. To use custom symbols, you must create a custom LMF that maps your custom symbols to the equivalent EDIF Input File, Text Design File (TDF), or other design file. You will also need to specify this custom LMF in the **EDIF Netlist Reader Settings** dialog box before compiling with the MAX+PLUS II software. See Compiling Projects with MAX+PLUS II Software for more information.

8. Continue with the steps necessary to complete your Concept schematic, as described in Creating Concept Schematics for Use with MAX+PLUS II Software.

   Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample hierarchical AHDL and Concept schematic file:

   - **/usr/maxplus2/examples/cadence/example4/fulladd2**

Converting Concept Schematics into MAX+PLUS II- Compatible EDIF Netlist Files with the concept2alt Utility

You can use the **concept2alt** utility to generate an EDIF netlist file from a Concept schematic. This file can then be imported into the MAX+PLUS® II software as an EDIF Input File (**.edf**).

To convert a Concept schematic into an EDIF netlist file, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment.

2. Create a Concept schematic and save it in your working directory, as described in Creating Concept Schematics for Use with MAX+PLUS II Software.

   ☞ You can instantiate MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.

3. Type the following command at the UNIX prompt from the **/source** directory that contains the schematic:

   `concept2alt -rundir ../max2` <*design name*> ↵

   If your design uses library of parameterized modules (LPM) functions, you must also include the `-family` option. For example:

```
concept2 alt -family FLEX10K -rundirmax2 <design name> ↵
```

4. Process the <*design name*>**.edf** file with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Performing a Functional Simulation of a Concept Schematic with the hdlconfig Utility & Verilog-XL Software

You can perform a functional simulation of a Concept schematic with the **hdlconfig** utility and Verilog-XL software before compiling your project with the MAX+PLUS® II software.

To functionally simulate a Concept schematic, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment.
2. Create a Concept schematic and save it in your working directory, as described in Creating Concept Schematics for Use with MAX+PLUS II Software.
3. Use the **hdlconfig** utility to create a Verilog HDL text file that contains the entire design. Type the following command at the UNIX prompt from the /<*working directory*>/<*design name*>/**source** directory:

   ```
   hdlconfig -a -c -r <design name> -o <design name>.v logic verilog_lib ↵
   ```

4. If your design contains RAM or ROM functions (e.g., `lpm_ram_dq`, `lpm_ram_io`, `lpm_rom`, `scfifo`, `dcfifo`, `altdpram`, and `csdpram`), run the **vconfig** utility to link the object **convert_hex2ver.o** to build a new Verilog-XL file that supports these functions by following these steps:
   1. Create a copy of the Verilog executable file by typing the following command at the UNIX prompt:

      ```
      cp -p $CDS_INST_DIR/tools/verilog/bin/verilog $CDS_INST_DIR/tools /verilog/bin/
      verilog.bak. ↵
      ```

   2. Type `vconfig` ↵ at the UNIX prompt from the **/usr/maxplus2/cadence/bin** directory to start the script.
   3. Accept `cr_vlog` as the name of the output script.
   4. Accept `1` as the stand-alone target.
   5. Type `new_verilog` as the name for the Verilog-XL target.
   6. Respond `Yes` when you are prompted to compile for the Verilog-XL environment.
   7. Respond `No` when you are prompted to include the Dynamic LAI, STATIC LOGIC AUTOMATION, LMSI HARDWARE MODELER, Verilog Mixed-Signal, and CDC interfaces in this executable.
   8. Respond `Yes` when you are prompted to include the Standard Delay File Annotator (SDF).
   9. Specify **/usr/maxplus2/verilog/veriuser.c** when you are asked the name of the user template file. For more information about the contents of the **veriuser.c** file, you can refer to the **veriuser.doc** file, which is available in the Cadence *Openbook* product documentation. To locate this document, start *Openbook*, and choose **Alphabetical List of Products** from the main menu. Scroll through the pages until you locate the *PLI 1.0 User Guide & Reference* in the PLI section, and then continue to scroll through the document until you locate the **veriuser.doc** file under "Section A" and "PLI Code

Examples."

10. When you are asked the name of files to be linked with the Verilog-XL simulator, specify the hexadecimal (Intel-format) conversion file **/usr/maxplus2/cadence/share/verilog/convert_hex2ver.o**, followed by a single period (.).

11. Run the output script **cr_vlog** to build the new Verilog-XL executable in the **/usr/maxplus2/cadence/bin** directory. Make sure that the `$CDS_INST_DIR/tools/bin` path appears at the beginning of the `PATH` statement in the **.cshrc** file.

12. If your C language library installation is different from the default location **/usr/lang/SC3.0.1**, type the following command at the UNIX prompt:

    `setenv C_DIR` <*C language library installation directory*> ↵

13. If successful, replace the old Verilog executable file with the new one by typing the following command at the UNIX prompt:

    `cp -p new_verilog $CDS_INST_DIR/tools/verilog/bin/verilog` ↵

5. Generate the stimulus file for the design and start the Verilog-XL simulator by typing the following command at the UNIX prompt from the /<*working directory*>/<*design name*>/**source** directory:

   `verilog -y /usr/maxplus2/simlib/concept/alt_max2/verilogUdps +libext+.v+.V` <*stimulus file name*> <*design name*>`.v` ↵

6. When you are ready to compile the project, generate an EDIF netlist file <*design name*>**.edf** with the **concept2alt** utility, as described in Converting Concept Schematics into MAX+PLUS II-Compatible EDIF Netlist Files with the **concept2alt** Utility.

7. Process the <*design name*>**.edf** file with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Performing a Functional Simulation of a Concept Schematic with VerilogLink & Verilog-XL Software

You can perform a functional simulation of a Concept schematic with VerilogLink and Verilog-XL software before compiling your project with the MAX+PLUS® II software.

To functionally simulate a Concept schematic, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment.

2. Create a Concept schematic and save it in your working directory, as described in Creating Concept Schematics for Use with MAX+PLUS II Software.

3. Generate the **global.cmd**, **vloglink.cmd**, **verilog.cmd**, and **expansion.dat** directive files.

4. Type `vloglink` <*design name*> ↵ from the /<*working directory*>/**source** directory to create a **vloglink.v** file from the Concept schematic.

5. Generate the stimulus file for the design and start the Verilog-XL simulator by typing the following command at the UNIX prompt from the /<*working directory*>/<*design name*>/**source** directory:

   `verilog -y /usr/maxplus2/simlib/concept/alt_max2/verilogUdps +libext+.v+.V` <*stimulus file*

name> `vloglink.v` ↵

6. When you are ready to compile the project, generate an EDIF netlist file <*design name*>**.edf** with the **concept2alt** utility, as described in Converting Concept Schematics into MAX+PLUS II-Compatible EDIF Netlist Files with the **concept2alt** Utility .

7. Process the <*design name*>**.edf** file with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Compiling Projects with MAX+PLUS II Software

The MAX+PLUS® II Compiler can process design files in a variety of formats. This topic describes how to use MAX+PLUS II software to compile projects in which the top-level design file is an EDIF Input File (with the extension **.edf**).

Related Topics:

Refer to the following sources for additional information:

- Go to MAX+PLUS II Help for information on compiling VHDL and Verilog HDL, design files directly with the MAX+PLUS II Compiler.

- Go to Running Synopsys Compilers from MAX+PLUS II Software for information on running the Synopsys Design Compiler or FPGA Compiler software on a VHDL or Verilog HDL design from within the MAX+PLUS II Compiler window.

To compile a design (also called a "project") with MAX+PLUS II software, go through the following steps:

1. Create design files that are compatible with the MAX+PLUS II software and convert them into EDIF Input Files with the extension **.edf**. Specific instructions for some tools are described in these MAX+PLUS II ACCESS℠ Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your design entry or synthesis and optimization tool.

2. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, create a mapping for each function in a Library Mapping File (**.lmf**) to map the custom symbol to the corresponding EDIF Input File, AHDL Text Design File (**.tdf**), or other MAX+PLUS II-supported design file. These custom functions are represented in design files as hollow-body symbols or "black box" HDL descriptions.

   ☞ Go to "Library Mapping Files (**.lmf**)" in MAX+PLUS II Help for more information.

3. Open MAX+PLUS II and specify the name of your top-level design file as the project name with the **Project Name** command (File menu). If you open an HDL file in the MAX+PLUS II Text Editor, you can choose the **Project Set Project to Current File** command (File menu) instead.

   ☞ You can also compile a project from a command line. However, the first time you compile a project, the settings you need to specify are easier to specify from within the MAX+PLUS II software. After you have run the graphical user interface for the MAX+PLUS II software at least once, you can more easily use the command-line **setacf** utility to modify options in the Assignment & Configuration File (**.acf**) for the project. Type `setacf -h` ↵ and `maxplus2 -h` ↵ for descriptions of **setacf** and MAX+PLUS II command-line syntax.

4. Choose **Device** (Assign menu) and select the target Altera device family in the *Device Family* drop-down list box. If you wish to implement the design logic in a specific device, select it in the *Devices* box. Otherwise,

select *AUTO* to allow the MAX+PLUS II Compiler to choose the best device(s) in the current device family. If your design entry or synthesis and optimization tool required you to specify a target family and/or device, specify the same information in this dialog box. For information on partitioning logic among multiple devices, go to MAX+PLUS II Help. Choose **OK**.

5. Open the Compiler window by choosing the **Compiler** command (MAX+PLUS II menu). Go through the following steps to specify the options necessary to compile the design file(s) in your project:

    1. Ensure that all EDIF netlist files have the extension **.edf** and choose **EDIF Netlist Reader Settings** (Interfaces menu).

    2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor. This name should be the name of the vendor whose tool(s) you used to create the EDIF netlist files. If your vendor name does not appear, select *Custom* instead.

        ☞ If you are compiling a design created with Synopsys FPGA Express software, select *Synopsys*, choose the **Customize** button, enter <*project name*>**.lmf** in the *LMF #1* box, choose **OK**, and skip to step 6.

    3. If you selected an existing vendor name in the *Vendor* box and your project contains design files that require custom LMF mappings, choose the **Customize** button to expand the dialog box to show all settings. Turn on the *LMF #2* checkbox and type your custom LMF's filename in the corresponding text box, or select a name from the *Files* box. The selection in the *Vendor* box will change to *Custom* and all settings will be retained until you change them again.

    4. If you selected *Custom* in the *Vendor* box, choose the **Customize** button to expand the dialog box to show all settings. Any previously defined custom settings will be displayed. Under *Signal Names*, type one or more names with up to 20 total name characters in the *VCC* or *GND* box if your EDIF Input File(s) use one or more names other than VCC or GND for the global high or low signals. Multiple signal names must be separated by either a comma (,) or a space. Under *Library Mapping Files*, turn on the *LMF #1* checkbox and type a filename in the text box following it, or select a name from the *Files* box. If necessary, specify another LMF name in the *LMF #2* box. Go to MAX+PLUS II Help for detailed information on the settings available in the **EDIF Netlist Reader Settings** dialog box.

    5. Choose **OK**.

6. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, you may need to ensure that all files are present in your project directory, i.e., the same directory as the top-level design file. Otherwise, you must specify the directories containing these files as user libraries with the **User Libraries** command (Options menu).

7. Follow all guidelines that apply to your design entry or synthesis and optimization tool:

    - Exemplar Logic Galileo Extreme-Specific Compiler Settings
    - Synopsys DesignWare-Specific Compiler Settings
    - Converting Synopsys FPGA Compiler & Design Compiler Timing Constraints into MAX+PLUS II-Compatible Format with the **syn2acf** Utility
    - Synplicity Synplify-Specific Compiler Settings

8. If you wish to generate EDIF, VHDL, or Verilog HDL output files for post-compilation simulation or timing analysis with another EDA tool, go through the following steps:

    1. (Optional) Turn on the **Optimize Timing SNF** command (Processing menu) to reduce the size of the output file(s). Turning on this command can reduce the size of output netlists by up to 30%.

        ☞ This command does not create optimized timing SNFs on UNIX workstations. However, a non-optimized timing SNF provides the same functional and timing information as an optimized

timing SNF.

2. If you wish to generate EDIF Output Files (**.edo**), go through these steps:

    1. Turn on the **EDIF Netlist Writer** command (Interfaces menu). Then choose the **EDIF Netlist Writer Settings** command (Interfaces menu).

    2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor and choose **OK**. If your vendor name does not appear, select *Custom* instead and specify the settings that are appropriate for your simulation or timing analysis tool. Go to MAX+PLUS II Help for detailed information on the options available in the **EDIF Netlist Writer Settings** dialog box.

    3. To generate an optional Standard Delay Format (SDF) Output File (**.sdo**), choose the **Customize** button to expand the dialog box to show all settings. Select one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**.

    The filenames of the EDIF Output File(s) and optional SDF Output File(s) are the same as the user-defined chip name(s) for the project; if no chip names exist, the Compiler assigns filenames that are based on the project name. For a multi-device project, the Compiler also generates a top-level EDIF Output File that is uniquely identified by "**_t**" appended to the project name. In addition, the Compiler automatically generates a VHDL Memory Model Output File, *<project name>***.vmo**, when it generates an EDIF Output File that contains memory (RAM or ROM).

3. If you wish to generate VHDL Output Files (**.vho**), turn on the **VHDL Netlist Writer** command (Interfaces menu). Then choose **VHDL Netlist Writer Settings** command (Interfaces menu). Select *VHDL Output File (.vho)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF ver. 2.1 files contain timing delay information that allows you to perform back-annotation simulation in VHDL with VITAL-compliant simulation libraries. The VHDL Output Files generated by the Compiler have the extension **.vho**, but are otherwise named in the same way as the EDIF Output Files described above.

4. If you wish to generate Verilog HDL Output Files (**.vo**), turn on the **Verilog Netlist Writer** command (Interfaces menu). Then choose **Verilog Netlist Writer Settings** command (Interfaces menu). Select *Verilog Output File (.vo)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF Output Files contain timing delay information that allows you to perform back-annotation simulation in Verilog HDL. The Verilog Output Files generated by the Compiler have the extension **.vo**, but are otherwise named in the same way as the EDIF Output Files described above.

9. To run the MAX+PLUS II Compiler, choose the **Project Save & Compile** command (File menu) or choose the **Start** button in the Compiler window.

    ☞ See step 3 for information on running MAX+PLUS II software from the command line.

10. Once you have compiled the project with the MAX+PLUS II Compiler, you can use the VHDL, Verilog HDL, or EDIF output file(s), and the optional SDF Output File(s) (**.sdo**) to perform timing analysis or timing simulation with another EDA tool. Specific instructions for some tools are described in these MAX+PLUS II ACCESS Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your EDA tool.

The MAX+PLUS II Compiler also generates a Report File (**.rpt**), a Pin-Out File (**.pin**), and one or more of the following files for device programming or configuration:

- JEDEC Files (**.jed**)
- Programmer Object Files (**.pof**)
- SRAM Object Files (**.sof**)
- Hexadecimal (Intel-format) Files (**.hex**)

- Tabular Text Files (**.ttf**)

Related Topics:

Refer to the following sources for additional information:

- Go to *Compiler Procedures* in MAX+PLUS II Help for information on other available Compiler settings.
- Go to *Programmer Procedures* in MAX+PLUS II Help for instructions on creating other types of programming files and on programming or configuring Altera devices.
- Go to Back-Annotating MAX+PLUS II Pin Assignments to Design Architect Symbols for information on back-annotating pin assignments in Mentor Graphics Design Architect schematics.
- Go to Programming Altera Devices for information on the different programming hardware options for Altera device families.

Go to the following topics, which are available on the web, for additional information:

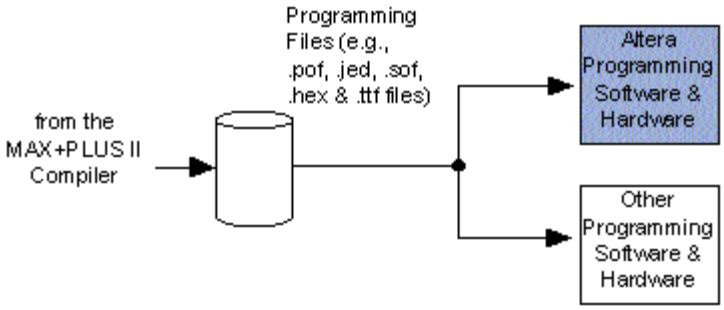
- MAX+PLUS II Development Software
- Altera Programming Hardware

Programming Altera Devices

Once you have successfully compiled and simulated a project with the MAX+PLUS® II software, you can program an Altera® device and test it in the target circuit. Figure 1 shows the device programming flow for MAX+PLUS II software.

Figure 1. MAX+PLUS II Device Programming Flow

Altera-provided items are shown in blue.



You can program devices with Altera programming hardware and MAX+PLUS II Programmer software installed on a 486- or Pentium-based PC or a UNIX workstation, or with programming hardware and software available from other manufacturers. Table 1 shows the available Altera programming hardware options on PCs and UNIX workstations.

Table 1. Altera Programming Hardware

	MAX 7000A, MAX	**FLEX® 6000,**

Programming Hardware Option	PCs	UNIX Work-stations	MAX® 3000A Devices	Classic® & MAX 5000 Devices	MAX 7000 & MAX 7000E Devices	7000AE, MAX 7000B, MAX 7000S MAX 9000 & MAX 9000A Devices	FLEX 6000A, FLEX 8000, FLEX 10K, FLEX 10KA, FLEX 10KB, & FLEX 10KE Devices	In-System Programming/ Configuration
Logic Programmer card, PL-MPU Master Programming Unit, and device-specific adapters	✔		✔	✔	✔	✔		
BitBlaster™ Download Cable	✔	✔	✔			✔	✔	✔
ByteBlasterMV™ Download Cable	✔		✔			✔	✔	✔
MasterBlaster™ Download Cable	✔	✔	✔			✔	✔	✔

If you wish to transfer programming files from a UNIX workstation to a PC over a network with File Transfer Protocol (FTP) or other similar transfer programs, be sure to select binary transfer mode.

Programming hardware from other manufacturers varies, but typically consists of a device connected to one of the serial ports on the workstation. Various vendors, such as Data I/O and BP Microsystems, supply hardware and software for programming Altera devices.

Related Topics:

Go to Compiling Projects with MAX+PLUS II Software for information on creating programming files.

Go to the following topics, which are available on the web, for additional information:

- MAX+PLUS II Development Software
- Altera Programming Hardware
- FLEX Devices
- MAX Devices
- Classic Device Family

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Concept & HDL Direct Project Directory Structure

Concept software generates the following files for each schematic:

- <*drawing name*>**/logic.1.1**
- <*drawing name*>**/logic_bn.1.1**
- <*drawing name*>**/logic_cn.1.1**
- <*drawing name*>**/logic_dp.1.1**

For designs that use HDL Direct software, Concept software also generates the following files:

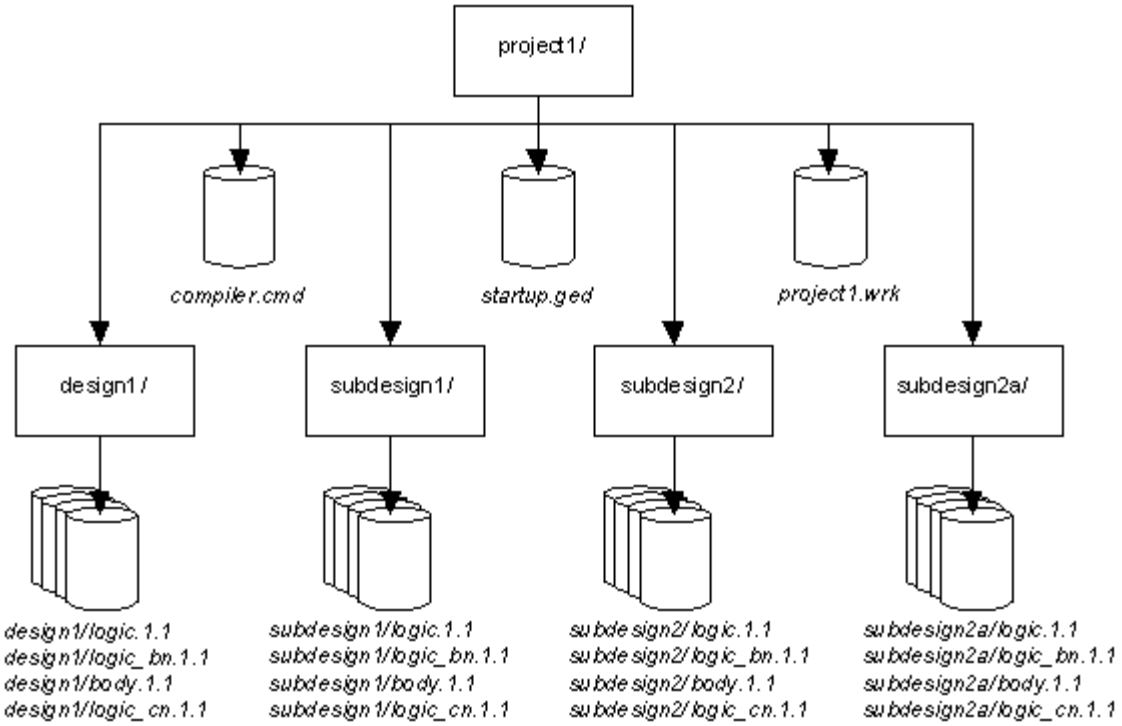
- <*drawing name*>**/logic_dp.1.1**
- <*drawing name*>**/logic_vd.1.1**
- <*drawing name*>**/logic/verilog.v**
- <*drawing name*>**/logic/vhdl.vhd**
- <*drawing name*>**/logic/hdldirect.dat**
- <*drawing name*>**/entity/vhdl.vhd**

These files are stored in their own <*drawing name*> directories. However, hierarchical relationships between files are not reflected in the file directory structure.

The local SCALD directory has an entry for all <*drawing name*> directories. Cadence software automatically manages drawing storage and retrieval operations through this special directory. The SCALD directory should have the same name as the UNIX project directory, but with the extension **.wrk**. Figure 1 shows a sample file structure, with **project1** as the UNIX project directory, and **project1.wrk** as the SCALD directory.

When the **concept2alt** utility converts the schematic into an EDIF netlist file, it processes the design information and all related file subdirectories, then creates the EDIF netlist file in the directory defined by the user. The EDIF netlist file is named <*project name*>**.edf**, where <*project name*> is the name of the top-level design file. Figure 1 shows the Cadence project file structure.

Figure 1. Cadence Project File Structure



Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Setting Up Design Compiler & FPGA Compiler Configuration Files

The **.synopsys_dc.setup** configuration file allows you to set both Design Compiler and FPGA Compiler variables. The compilers read **.synopsys_dc.setup** files from three directories, in the following order:

1. The Synopsys root directory
2. Your home directory
3. The directory where you start the Design Compiler or FPGA Compiler software

The most recently read configuration file has highest priority. For example, a configuration file in the directory where you start the Design Compiler or FPGA Compiler software has priority over the other configuration files, and a configuration file in the home directory has priority over a configuration file in the root directory.

To set up your configuration files, follow these steps:

1. Add the lines shown in Figure 1 to your **.synopsys_dc.setup** configuration file. Altera provides a sample **.synopsys_dc.setup** file in the **./synopsys/config** directory. Figure 1 shows an excerpt from that sample file.

   Figure 1. Excerpt from Sample .synopsys_dc.setup File

   ```
   search_path = {./usr/maxplus2/synopsys/library/alt_syn/<device family>/lib};
   target_library = {<technology library>};
   symbol_library = {altera.sdb};
   link_library = {<technology library>};
   edifout_netlist_only = "true"
   edifout_power_and_ground_representation = "net"
   edifout_power_net_name = "VDD"
   edifout_ground_net_name = "GND"
   edifout_no_array = "false"
   edifin_power_net_name = "VDD"
   edifin_ground_net_name = "GND"
   compile_fix_multiple_port_nets = "true"
   bus_naming_style = "%s<%d>"
   bus_dimension_separator_style = "><"
   bus_inference_style = "%s<%d>"
   ```

2. Specify one of the Design Compiler & FPGA Compiler Technology Libraries for the `target_library` and `link_library` parameters in the **.synopsys_dc.setup** file.

3. If you will instantiate architecture control logic functions from the **alt_mf** library, add the following line to your **.synopsys_dc.setup** file:

   ```
   define_design_lib altera -path /usr/maxplus2/synopsys/library/alt_mf/lib ↵
   ```

   If you wish to use the VHDL System Simulator (VSS) software to simulate a VHDL design containing **alt_mf** library functions, you must compile this library with the **analyze_vss** script. See Setting Up

VSS Configuration Files for more information.

4. If you will use the DesignWare interface for FLEX® 6000, FLEX 8000, or FLEX 10K designs, enter additional lines in your **.synopsys_dc.setup** file, as described in Setting Up the DesignWare Interface.
5. Specify one of the following families for the <*device family*> variable in the `search_path` parameter: `max5000`, `max7000`, `max9000`, `flex6000`, `flex8000`, or `flex10k`.
6. If you wish to resynthesize a design for a different device family, modify the **.synopsys_dc.setup** file by following the steps described in Resynthesizing a Design Using the **alt_vtl** Library & a MAX+PLUS II SDF Output File.

Related Links:

- Go to MAX+PLUS® II /Synopsys Interface File Organization in these MAX+PLUS II ACCESS℠ Key topics for related information.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Using Cadence Concept & MAX+PLUS II Software

The following topics describe how to use the Cadence Concept software with MAX+PLUS® II software. Click on one of the following topics for information:

Open a printable version of all topics listed on this page.

Setting Up the MAX+PLUS II/Cadence Working Environment

- Software Requirements
- Setting Up the MAX+PLUS II/Cadence Concept Work Environment for a Sun SPARCstation Running SunOS Software
- MAX+PLUS II Directory Structure
- MAX+PLUS II/Cadence Interface File Organization
- Concept & RapidSIM Local Work Area Directory Structure
- Concept & HDL Direct Project Directory Structure
- Altera-Provided Logic & Symbol Libraries

Design Entry

- Design Entry Flow
- Creating Concept Schematics for Use with MAX+PLUS II Software
  - Instantiating the clklock Megafunction in Concept Schematics
  - Instantiating LPM & Other Parameterized Functions in Concept Schematics
- Entering Resource Assignments
  - Assigning Pins, Logic Cells & Chips
  - Assigning Cliques
  - Assigning Logic Options
  - Modifying the Assignment & Configuration File with the **setacf** Utility
- Creating Hierarchical Projects in Concept Schematics
- Converting Concept Schematics into MAX+PLUS II-Compatible EDIF Netlist Files with the **concept2alt** Utility

Functional Simulation

- Performing a Functional Simulation of a Concept Schematic with the **hdlconfig** Utility & Verilog-XL Software
- Performing a Functional Simulation of a Concept Schematic with VerilogLink & Verilog-XL Software

Related Links:

- Compiling Projects with MAX+PLUS II Software
- Programming Altera Devices
- MAX+PLUS II Development Software
- Altera Programming Hardware
- Cadence web site (http://www.cadence.com)

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Concept & RapidSIM Local Work Area Directory Structure

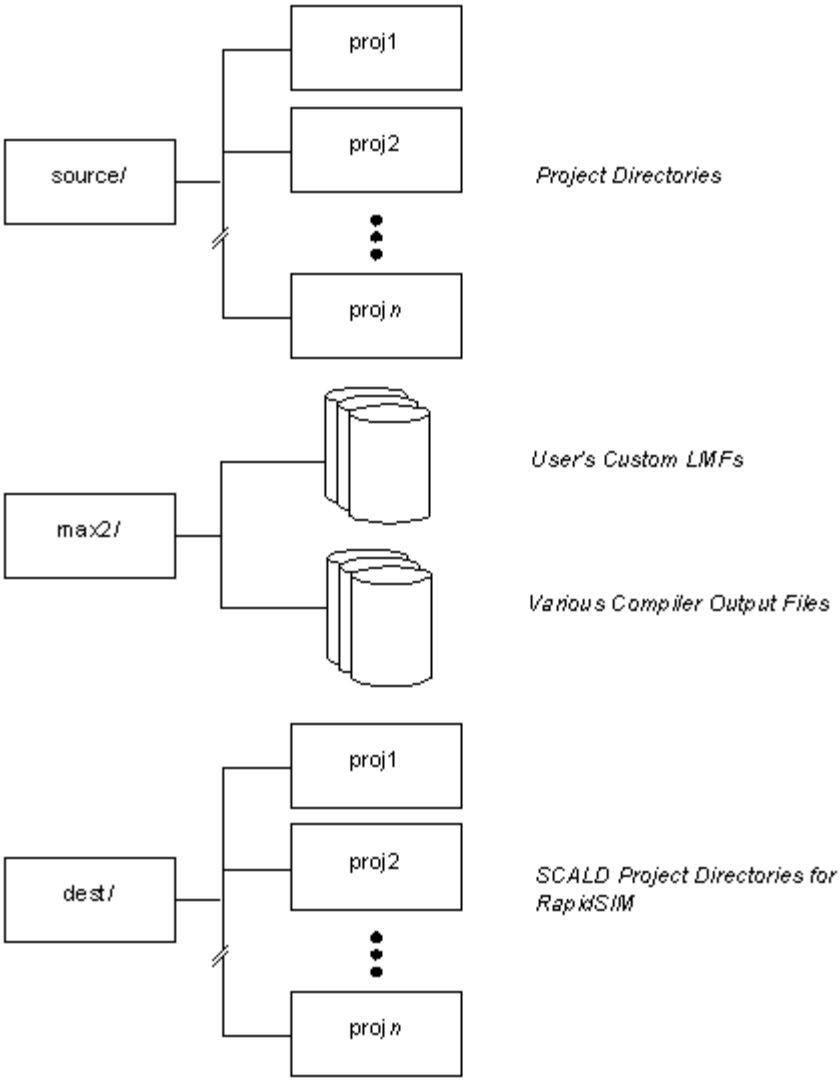
When the **redifnet** utility imports an EDIF netlist file for the RapidSIM software, it creates a SCALD directory for your project. However, creating this directory may overwrite the directory that was created for the original Concept schematic. To prevent overwriting this directory, you should create a file structure that helps you manage your design files.

Altera recommends that you create the following three directories for your design files.

Directory:	Description:
./source	Create Concept schematics and generate EDIF netlist files with the **wedifnet** utility in the **source** directory.
./max2	Copy the EDIF Input File (**.edf**) from the **source** directory to this directory to compile the file with the MAX+PLUS® II software.
./dest	Copy the EDIF Output File (**.edo**) from the **max2** directory to this directory to run the **redifnet** and RapidSIM software.

Copies of the appropriate directives files for Cadence tools must be present in both the **source** and **dest** directories. Figure 1 shows Altera's recommended file structure.

Figure 1. Recommended File Structure



Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Intel® Documentation Conventions

Intel documents use consistent conventions to make it easy for you to find and interpret information:

- Typographic Conventions
- Terminology
- Backus-Naur Form (BNF)
- Key Combinations

Typographic Conventions

Intel documentation uses the following typographic conventions:

Visual Cue	Meaning
Bold Initial Capitals	Command names, dialog box titles, and button names are shown in bold type, with initial capital letters. Examples: **Find Text** command, **Save As** dialog box, **Start** button.
bold type	Directory names, project names, disk drive names, filenames, filename extensions, and software utility names are shown in bold type. Examples: **Â¥maxplus2** directory, **d:** drive, **chiptrip.gdf** file. These items are not case-sensitive in the Windows environment; however, they are case-sensitive in the UNIX workstation environment. Intel documentation shows these items in the case appropriate to the workstation environment.
Initial Capitals	Keyboard keys, user-editable application window fields, and menu names are shown with initial capital letters. Examples: Delete key, the Start Time field, the Options menu.
"Subheading Title"	Subheadings within a document are enclosed in quotation marks. In manuals, titles of help topics are also shown in quotation marks.
Italic Initial Capitals	Help categories, section titles in books, application note and brief names, checkbox options, and options in dialog boxes are shown in italic type with initial capital letters. Examples: *Text Editor Procedures*, the *Check Outputs* option, the *Directories* box in the **Open** dialog box.
italic type	Variables are enclosed in angle brackets (< >) and shown in italic type. Example: *<filename>, <project name>**.acf*** file.
Bold Italic Type	Book and CD-ROM titles are shown in bold italic type with initial capital letters. Example: ***MAX+PLUS II Getting Started***.
`Monospace font`	Anything that must be typed exactly as it appears is shown in monospace font. For example: `c:max work tutorial chiptrip.gdf`. Also, sections of an actual file, such as a Report File, references to parts of files (e.g., the AHDL keyword `SUBDESIGN`), and logic function names (e.g., `DFF` and `16cudslr`) are shown in monospace font.
`Bold Monospace font`	In syntax descriptions, bold monospace font may be used to help distinguish literal text from variables.
1., 2., 3., *and* a., b., c.,...	Numbered steps are used in a list of items when the sequence of the items is important, such as the steps listed in a procedure. Bullets are used in a list of items when the sequence of the items is not important.
✔	The checkmark indicates a procedure that consists of one step only.
☞	The hand points to information that requires special attention.

Terminology

The following terminology is used throughout Intel documentation:

Term	Meaning
Button 1	Left mouse button.
Button 2	For PCs, the right button on a two-button mouse or the middle or right button on a three-button mouse. For UNIX workstations, the right button on a three-button mouse.
"point to"	Indicates that you should move the mouse so that the pointer is over the specified item.
"choose"	Indicates that you need to use a mouse or key combination to start an action. For example, when you use the mouse to choose a button, you point to the button and click Button 1. When you use the keyboard to choose a command, you type Alt and then type letters that are underlined in the menu bar and menu. (In UNIX workstation-based MAX+PLUS® II software, you must use Ctrl instead of Alt.)
"select"	Indicates that you need to highlight text, and/or objects, or an option in a dialog box with a key combination or the mouse. A selection does not start an action. Example: Select the `AND2` primitive, then choose **Delete** from the Edit menu.
"press"	Indicates that you must hold down a mouse button or key.
"turn on"/"turn off"	Indicates that you must click Button 1 on a checkbox or choose a menu command to turn a function on or off.
"click"	Indicates a quick press and release of a mouse button.
"double-click"	Indicates two clicks in rapid succession.
"Related Links"	"Related Links" show you where to go for more information.

Backus-Naur Form (BNF)

Backus-Naur Form (BNF) is used to define the syntax of typed commands, text file formats, and variables. BNF uses the following notation:

Characters	Meaning
::=	"Is defined as"
<...>	Identifiers (i.e., variables)
[...]	Optional items
{ ... }	Repeated items (zero or more times)
...|...	Indicates a choice between items
`:n:n`	Suffix indicates a range (e.g., <*name char*>`:1:8` means "from 1 to 8 name characters")
italic type	Variables in syntax descriptions, i.e., text
`Monospace font`	Literal text in syntax descriptions that you must type
`Bold Monospace font`	sometimes used to help distinguish literal text from *italic variables* in syntax descriptions

Key Combinations

Key combinations and sequences appear in the following format:

Format Cue	Meaning
Key1+Key2	A plus (+) symbol indicates that you must hold down the first key when you press the second key. Example: Ctrl+L means that you must hold down Ctrl while pressing L, then release both keys.
Key1,Key2	A comma (,) indicates that you must press the keys sequentially. Example: Alt,F1 means that you must press Alt and release it, then press F1 and release it.

Feedback

Did this information help you?

If no, Intel® Premier Support customers can file a case by logging into Intel® Premier Support.

Intel does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Creating Concept Schematics for Use with MAX+PLUS II Software

You can create Composer schematics and convert them into EDIF Input Files (**.edf**) that can be processed with the MAX+PLUS® II Compiler. To create a Composer schematic for use with the MAX+PLUS II software, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment.

2. Start the Composer schematic editor from the <*working directory*> by typing `icds` ↵ at a UNIX prompt. Use the graphical user interface to structure and organize your files to create an environment that facilitates entering and processing designs. Go to Composer Project File Directory Structure for more information on directories in Composer.

3. Choose **Library Path Editor** (Tools menu) to create the <*design name*> library. In the **Library** dialog box, type <*project directory name*> as the *Library* name and `./source/`<*design name*> as the *Path* name. Choose **Save** (File menu), then choose **Exit** (File menu) to save the path.

4. Choose **Library Manager** (Tools menu) to start Composer and create a new design.

5. Type <*project directory name*> as the *Library* name, <*design name*> as the *Cell* name, and `schematic` as the *View* name in the **Library Manager** dialog box and press the ↵ key.

6. Enter primitives, megafunctions, and macrofunctions from the following libraries:

   - MAX+PLUS II-compatible primitives, megafunctions, and macrofunctions are available in the Altera-provided **alt_max2** component library.

   - Input, output, and bidirectional pins are available in the Cadence **basic** library located under **/cadence/etc/cdslib**.

   - MegaCore™ functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP™). The OpenCore™ feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.

   ☞ If you wish to create a hierarchical design that contains symbols representing other design files, such as Altera® Hardware Description Language (AHDL) Text Design Files, go to Creating Hierarchical Projects in Composer Schematics.

7. Enter meaningful instance names for all symbols and functions so that you can easily trace internal node names during simulation and debugging operations. For example, if an `a161` macrofunction is instantiated several times in one design, you should define a unique name for each instance. The instance name for each symbol is controlled by `INST` property. For more information on assigning properties, refer to the Cadence ***Composer User Guide***.

8. (Optional) To enter resource assignments in your Composer schematic, go to Entering Resource Assignments. You can also enter resource assignments from within the MAX+PLUS II software.

9. (Optional) Functionally simulate the design with the Verilog-XL simulator. Altera provides Verilog HDL simulation modules in the **/usr/maxplus2/simlib/composer/alt_max2/verilog** and **/usr/maxplus2/simlib/composer/alt_max2/verilogUdps** directories. Go to Performing a Functional Simulation of a Composer Schematic with Verilog-XL Software for more information.

10. Use the **altout** utility to generate an EDIF netlist file that can be imported into the MAX+PLUS II software, as described in Converting Composer Schematics into MAX+PLUS II-Compatible EDIF Netlist Files with the **altout** Utility.

11. Process the <*design name*>**.edf** file with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample Composer schematic files:

- **/usr/maxplus2/examples/cadence/example2/fulladd**
- **/usr/maxplus2/examples/cadence/example5/fulladd2**
- **/usr/maxplus2/examples/cadence/example7/fa2**

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

You can create Concept schematics and convert them to EDIF Input Files (**.edf**) that can be processed with the MAX+PLUS® II Compiler. To create a Concept schematic for use with the MAX+PLUS II software, go through the following steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment.
2. Make sure the required directive files are in the /<*working directory*>/<*design name*>**/source** directory. If not, you can use the Altera-provided template files located in the following directories:
   - **/usr/maxplus2/simlib/concept/edifnet/templates**
   - **/usr/maxplus2/simlib/concept/edifnet/redifnet**
3. Start the Concept schematic editor by typing `concept` <*design name*> ↵ at a UNIX prompt from the /<*working directory*>/**source** directory. Use the graphical user interface to structure and organize your files to create an environment that facilitates entering and processing designs. Go to Concept & RapidSIM Local Work Area Directory Structure for more information on directories in Concept.
4. To write a Verilog HDL text file whenever the design is saved, choose the **Block** button in the Concept window.

   ☞ To use the HDL Direct utility to process your design, turn on the *HDL Direct On* option in the Concept window. Go to Concept & HDL Direct Project Directory Structure for information on the files generated by Concept software when using the HDL Direct utility.
5. Enter primitives, megafunctions, and macrofunctions from the following Altera-provided component libraries:
   - **alt_max2** includes macrofunctions, megafunctions, and primitives.
   - **alt_lpm** includes library of parameterized modules (LPM) functions (available only if you use HDL Direct software).

See the following topics for instructions for specific functions:

- Instantiating LPM & Other Parameterized Functions in Concept Schematics
- Instantiating the clklock Megafunction in Concept Schematics

☞ You can instantiate MegaCore™ functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP™). The OpenCore™ feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.

6. If you wish to create a hierarchical design that contains symbols representing other design files, such as Altera® Hardware Description Language (AHDL) Text Design Files, go to Creating Hierarchical Projects in Concept Schematics.
7. Enter meaningful instance names for all symbols and functions so that you can easily trace internal node names during simulation and debugging operations. For example, if an `a161` macrofunction is instantiated several times in one design, you should define a unique name for each instance. The instance name for each symbol is controlled by `INST` property. For more information on assigning properties, refer to the Cadence ***Concept Schematic User Guide***.
8. Enter input, output, and bidirectional ports:
   - If you turned on the *HDL Direct On* option in step 4, add `inport` and `outport` symbols from the **hdl_direct_lib** library to the interface symbols.
   - If you are not using HDL Direct, use `flag` symbols from the **standard** library to indicate input, output, and bidirectional ports. Be sure to end pin names with `¥I` to identify them as interface signals.

   ☞ If a pin is not used, leave it floating. The **concept2alt** utility removes all unconnected pins when it generates an EDIF netlist file.

9. (Optional) To enter resource assignments in your Concept schematic, go to Entering Resource Assignments. You can also enter resource assignments from within the MAX+PLUS II software.
10. (Optional) Perform a functional simulation, as described in one of the following topics:
    - Performing a Functional Simulation of a Concept Schematic with the **hdlconfig** Utility & Verilog-XL Software
    - Performing a Functional Simulation of a Concept Schematic with VerilogLink & Verilog-XL Software
11. Use the **concept2alt** utility to generate an EDIF netlist file that can be imported into the MAX+PLUS II software, as described in Converting Concept Schematics into MAX+PLUS II-Compatible EDIF Netlist Files with the **concept2alt** utility.
12. Process the <*design name*>**.edf** file with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample Concept schematic files:

- **/usr/maxplus2/examples/cadence/example1/fulladd**
- **/usr/maxplus2/examples/cadence/example4/fulladd2**
- **/usr/maxplus2/examples/cadence/example6/fa2**
- **/usr/maxplus2/examples/cadence/example12/fifo**

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Creating Verilog HDL Designs for Use with MAX+PLUS II Software

You can create Verilog HDL design files with the MAX+PLUS® II Text Editor or another standard text editor and save them in the appropriate directory for you project. The MAX+PLUS II Text Editor offers the following advantages:

- Verilog HDL templates are available with the **Verilog Templates** command (Templates menu). These templates are also available in the ASCII **verilog.tmp** file, which is located in the **/usr/maxplus2** directory.
- If you use the MAX+PLUS II Text Editor to create your Verilog HDL design, you can use the **Syntax Coloring** command (Options menu). The Syntax Coloring feature displays keywords and other elements of text in text files in different colors to distinguish them from other forms of syntax.

To create a Verilog HDL design that can be synthesized and optimized with Synergy software, go through the following steps:

1. You can instantiate the following MAX+PLUS II-provided logic functions in your Verilog HDL design:
   - The **alt_max2** library, which contains the `a_8count`, `a_8mcomp`, `a_8fadd`, and `a_81mux` macrofunctions that are optimized for different Altera device families.
   - The `clklock` megafunction which enables phase-locked loop, or ClockLock™ , circuitry available on selected Altera FLEX® 10K devices. Go to Instantiating the `clklock` Megafunction in VHDL or Verilog HDL for information.
   - The **lpm_syn** library, which contains the Cadence LPM megafunction library for use with Synergy Software and Concept or Composer software.
   - MegaCore™ functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP™). The OpenCore™ feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.
2. You can enter resource assignments in your Verilog HDL design, as described in Entering Resource Assignments.
3. After you have completed your Verilog HDL design, synthesize and optimize it with Synergy software, as described in Synthesizing & Optimizing Verilog HDL Files with Synergy Software.

Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample Verilog HDL files, the latter of which includes LPM function instantiation.

- **/usr/maxplus2/examples/cadence/example11/count8.v**
- **/usr/maxplus2/examples/cadence/example13/rom_test.v**

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

You can create Verilog HDL design files with the MAX+PLUS® II Text Editor, the FPGA Express internal text editor, or another standard text editor and save them in the appropriate directory for your project. The MAX+PLUS II and FPGA Express text editors offer different advantages; use either or both depending on your personal preferences:

-
- The MAX+PLUS II Text Editor offers Verilog HDL templates with the **Verilog HDL Templates** command (Templates menu) and syntax coloring with the **Syntax Coloring** command (Options menu).
- The FPGA Express internal text editor provides automatic error location when you double-click an error in the Output window.

To create a Verilog HDL design that can be synthesized and optimized with the FPGA Express software, follow these steps:

1.
2. Describe your design using FPGA Express-supported Verilog HDL constructs. For information on synthesizable Verilog HDL constructs, refer to the online ***HDL Reference Manual*** provided with the FPGA Express software. The following topics describe how to instantiate additional Altera-specific logic functions in your design:
    - Instantiating the clklock Megafunction in VHDL or Verilog HDL
    - Instantiating RAM & ROM Functions in Verilog HDL
    - Instantiating LPM Functions in Verilog HDL

    You can instantiate MegaCore™ functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP™). The OpenCore™ feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.
3. Once you have created a design, synthesize and optimize it, as described in Synthesizing & Optimizing VHDL or Verilog HDL Files with FPGA Express Software.

Related Links:

- Compiling Projects with MAX+PLUS II Software in these MAX+PLUS II ACCESS℠ Key topics for related information.
- Go to the following topics for additional information:
    - Altera Megafunction Partners Program (AMPP)
    - Altera Megafunctions

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

You can create Verilog HDL design files with the MAX+PLUS® II Text Editor or another standard text editor and

save them in the appropriate directory for your project. The MAX+PLUS II Text Editor offers the following advantages:

- Verilog HDL templates are available with the **Verilog Templates** command (Templates menu). These templates are also available in the ASCII **verilog.tmp** file, which is located in the **/usr/maxplus2** directory.
- If you use the MAX+PLUS II Text Editor to create your Verilog HDL design, you can use the **Syntax Coloring** command (Options menu). The Syntax Coloring feature displays keywords and other elements of text in text files in different colors to distinguish them from other forms of syntax.

To create a Verilog HDL design and convert it to an EDIF netlist file for use with MAX+PLUS II software, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Synplicity Working Environment.
2. Instantiate any MAX+PLUS II-supported logic function in your Verilog HDL design. You can enter the following functions:
    - Parameterized and non-parameterized megafunctions. MAX+PLUS II software also supports all functions in the library of parameterized modules (LPM) 2.1.0, except the truth table, finite state machine, and pad functions.
    - Macrofunctions, including 74-series functions.
    - Buffer primitives, including `lcell`, `soft`, `global`, `carry`, and `cascade`. The Synplicity **altera.v** library provides synthesis support for these functions.
    - MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.

    Choose **Primitives**, **Old-Style Macrofunctions**, and **Megafunctions/LPM** from the MAX+PLUS II Help menu for information on all MAX+PLUS II-supported functions.
3. If your design uses functions from the **altera.v** library, add the library file name to the top of the *Source Files* list in the Synplify window.
4. For each MAX+PLUS II-supported logic function, include a `black_box` synthesis directive. You can omit this step for functions from the **altera.v** library.
5. For any parameterized function, you must declare all parameters used in the function, and their values. Figure 1 shows a Verilog HDL file that instantiates the `lpm_ram_dq` function. A comment in the Module Declaration contains the `synthesis black_box` directive and parameter names and values. This comment must immediately follow the port list and precede the closing semicolon (`;`). When you instantiate an LPM function, the LPM function name must be specified as the value of the `LPM_TYPE` parameter. In addition, each parameter must be listed on a separate line. See Figure 1.

    Figure 1. Verilog HDL Design File with LPM Function Instantiation

```
// Define the black box
module myram_64x16 (data, address, inclock, outclock, we, q)
/* synthesis black_box

                    LPM_WIDTH=16
                    LPM_WIDTHAD=6
                    LPM_TYPE="LPM_RAM_DQ"  */ ;

input [15:0] data;
```

```
input [5:0] address;
input inclock, outclock;
input we;
output [15:0] q;

endmodule

// Instantiate the LPM parameterized module in the
//    higher-level module myram
module myram(clock, we, data, address, q);
input clock, we;
input [15:0] data;
input [5:0] address;
output [15:0] q;


       myram_64x16 inst1 (data, address, clock, clock, we, q);

endmodule
```

6. (Optional) Enter resource assignments for your Verilog HDL design, as described in Entering Resource Assignments.
7. After you have completed your Verilog HDL design, synthesize and optimize it with Synplify software, as described in Synthesizing & Optimizing VHDL or Verilog HDL Files with Synplify Software.

Related Links

- Compiling Projects with MAX+PLUS II Software

Creating VHDL Designs for Use with MAX+PLUS II Software

You can create VHDL design files with the MAX+PLUS® II Text Editor or another standard text editor and save them in the appropriate directory for your project. The MAX+PLUS II Text Editor offers the following advantages:

- VHDL templates are available with the **VHDL Templates** command (Templates menu). These templates are also available in the ASCII **vhdl.tmp** file, which is located in the **/usr/maxplus2** directory.
- If you use the MAX+PLUS II Text Editor to create your VHDL design, you can use the **Syntax Coloring** command (Options menu). The Syntax Coloring feature displays keywords and other elements in text files in different colors to distinguish them from other forms of syntax.

To create a VHDL design that can be synthesized and optimized with Synergy software, follow these steps:

1. You can instantiate the following MAX+PLUS II-provided logic functions in your VHDL design:
   - The **alt_mf** library contains the Altera® VHDL logic function library, which includes the `a_8count`, `a_8mcomp`, `a_8fadd`, and `a_81mux` macrofunctions. If you wish to instantiate **alt_mf** logic functions in your VHDL design, you must first compile this library, as described in Compiling the **alt_mf** Library.
   - The `clklock` megafunction, which enables the phase-locked loop, or ClockLock™ , circuitry available on selected Altera FLEX® 10K devices. Go to Instantiating the `clklock` Megafunction in VHDL or Verilog HDL for information.
   - MegaCore™ functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP™). The OpenCore™ feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.
2. If you wish to use Standard Delay Format (SDF) Output Files (**.sdo**) that contain timing information when performing post-compilation timing simulation with Leapfrog software, you must first compile the VITAL library source files, as described in Compiling the **alt_vtl** Library for for Use with Leapfrog Software.
3. (Optional) To enter resource assignments in your VHDL design, go to Entering Resource Assignments. You can also enter resource assignments from within the MAX+PLUS II software.
4. After you have completed your VHDL design, synthesize and optimize it with Synergy software, as described in Synthesizing & Optimizing VHDL Files with Synergy Software.

Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample VHDL files, the latter of which includes macrofunction instantiation.

- **/usr/maxplus2/examples/cadence/example9/count4.vhd**
- **/usr/maxplus2/examples/cadence/example10/adder16.vhd**

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

You can create VHDL design files with the MAX+PLUS® II Text Editor, the FPGA Express internal text editor, or another standard text editor and save them in the appropriate directory for your project. The MAX+PLUS II and FPGA Express text editors offer different advantages; use either or both depending on your personal preferences:

-
- The MAX+PLUS II Text Editor offers VHDL templates with the **VHDL Templates** command (Templates menu) and syntax coloring with the **Syntax Coloring** command (Options menu).
- The FPGA Express internal text editor provides automatic error location when you double-click an error in the Output window.

To create a VHDL design that can be synthesized and optimized with the FPGA Express software, follow these steps:

1.
2. Describe your design using FPGA Express-supported VHDL constructs. For information on synthesizable VHDL constructs, refer to the online ***VHDL Reference Manual*** provided with the FPGA Express software. The following topics describe how to instantiate additional Altera-specific logic functions in your design:
    - Instantiating the clklock Megafunction in VHDL or Verilog HDL
    - Instantiating RAM & ROM Functions in VHDL
    - Instantiating LPM Functions in VHDL

    ☞ You can instantiate MegaCore™ functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP™). The OpenCore™ feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.
3. Once you have created a design, synthesize and optimize it, as described in Synthesizing & Optimizing VHDL or Verilog HDL Files with FPGA Express Software.

Related Links:

- Go to the following MAX+PLUS II ACCESS℠ Key topics for related information:
    - Using Synopsys VSS & MAX+PLUS II Software
    - Compiling Projects with MAX+PLUS II Software
- Go to the following topics for additional information:
    - Altera Megafunction Partners Program (AMPP)
    - Altera Megafunctions

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

You can create VHDL design files with the MAX+PLUS® II Text Editor or another standard text editor and save them in the appropriate directory for your project. The MAX+PLUS II Text Editor offers the following advantages:

- VHDL templates are available with the **VHDL Templates** command (Templates menu). These templates are also available in the ASCII **vhdl.tmp** file, which is located in the **/usr/maxplus2** directory.

- If you use the MAX+PLUS II Text Editor to create your VHDL design, you can use the **Syntax Coloring** command (Options menu). The Syntax Coloring feature displays keywords and other elements of text in text files in different colors to distinguish them from other forms of syntax.

To create a VHDL design and convert it to an EDIF netlist file for use with MAX+PLUS II software, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Synplicity Working Environment.

2. Instantiate any MAX+PLUS II-supported logic function in your VHDL design. You can enter the following functions:

    - Parameterized and non-parameterized megafunctions. MAX+PLUS II software also supports all functions in the library of parameterized modules (LPM) 2.1.0, except the truth table, finite state machine, and pad functions.

    - Macrofunctions, including 74-series functions.

    - Buffer primitives, including `lcell`, `soft`, `global`, `carry`, and `cascade`. The Synplicity **altera.vhd** library provides synthesis support for these functions.

    - MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.

    Choose **Primitives**, **Old-Style Macrofunctions**, and **Megafunctions/LPM** from the MAX+PLUS II Help menu for information on all MAX+PLUS II-supported functions.

3. If your design uses functions from the **altera.vhd** library, add the following Library and Use clauses to the top of a file that instantiates the macrofunction(s):

```
library altera;
use altera.maxplus2.all;
```

4. For each MAX+PLUS II-supported logic function, include a `black_box` synthesis directive. See Figure 1. You can omit this step for functions from the **altera.vhd** library.

5. For any parameterized function, declare all parameters used in the function, their types, and their values. Attribute Declarations are used to declare the `black_box` attribute and the name and type of each parameter. The `black_box` attribute has the `boolean` type; refer to MAX+PLUS II Help for information on whether a parameter is of `integer` or `string` type. Attribute Specifications then assign values to each parameter. Figure 1 shows a VHDL design file that instantiates the `lpm_ram_dq` function.

Figure 1. VHDL Design File with LPM Function Instantiation

```
entity myram is
port (clock, we: in bit;
      data : in bit_vector (3 downto 0);
      address: in bit_vector (1 downto 0);
      q: out bit_vector (3 downto 0));
end myram;

architecture arch1 of myram is

    -- Declare the component
```

```
    component myram_4x4
              port (data: in bit_vector (3 downto 0);
                       address: in bit_vector (1 downto 0);
                       inclock, outclock, we: in bit;
                       q: out bit_vector (3 downto 0) );
    end component;

--  Declare the black_box and parameters and their types

attribute black_box: boolean;
attribute LPM_WIDTH: integer;
attribute LPM_WIDTHAD: integer;
attribute LPM_TYPE: string;


-- Assign values to each attribute

attribute black_box of myram_4x4: component is true;
attribute LPM_WIDTH of myram_4x4: component is 4;
attribute LPM_WIDTHAD of myram_4x4: component is 2;
-- Specify the name of the LPM function as the value of the
-- LPM_TYPE attribute
attribute LPM_TYPE of myram_4x4: component is "LPM_RAM_DQ"

begin
          -- Instantiate the LPM component
          u1: myram_4x4  port map(data, address, clock,
                                                    clock, we, q);

end arch1;
```

6. (Optional) Enter resource assignments for your VHDL design, as described in Entering Resource Assignments.
7. After you have completed your VHDL design, synthesize and optimize it with Synplify software, as described in Synthesizing & Optimizing VHDL or Verilog HDL Files with Synplify Software.

Related Links:

- Go to Compiling Projects with MAX+PLUS II Software in these MAX+PLUS II ACCESS^{SM} Key topics for related information.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Using Synopsys Design Compiler or FPGA Compiler & MAX+PLUS II Software

The following topics describe how to use the Synopsys Design Compiler and FPGA Compiler software with the MAX+PLUS® II software. Click on one of the following topics for information:

This file is suitable for printing only. It does not contain hypertext links that allow you to jump from topic to topic.

Setting Up the MAX+PLUS II/Synopsys Working Environment

- Software Requirements
- Setting Up Design Compiler & FPGA Compiler Configuration Files
- Setting Up the DesignWare Interface
  - Updating DesignWare Libraries
- Libraries
  - Design Compiler & FPGA Compiler Technology Libraries
  - VHDL & Verilog HDL **alt_mf** Logic Function Library
  - DesignWare FLEX 6000, FLEX 8000 & FLEX 10K Synthetic Libraries
  - Post-Synthesis Libraries
- MAX+PLUS II/Synopsys Interface File Organization
- MAX+PLUS II Project File Structure

Design Entry

- **Design Entry Flow**

- **VHDL**

  - Creating VHDL Designs for Use with MAX+PLUS II Software
    - Instantiating RAM & ROM Functions in VHDL (includes examples)
    - Instantiating the clklock Megafunction in VHDL or Verilog HDL (includes examples)
  - Additional examples:
    - Primitive & Old-Style Macrofunction Instantiation Example for VHDL
    - Architecture Control Logic Function Instantiation Example for VHDL
    - DesignWare Up/Down Counter Function Instantiation Example for VHDL

- **Verilog HDL**

  - Creating Verilog HDL Designs for Use with MAX+PLUS II Software
    - Instantiating RAM or ROM Functions in Verilog HDL (includes examples)
    - Instantiating the clklock Megafunction in VHDL or Verilog HDL (includes examples)
  - Additional examples:
    - Primitive & Old-Style Macrofunction Instantiation Example for Verilog HDL
    - Architecture Control Logic Function Instantiation Example for Verilog HDL

Synthesis & Optimization

- Synthesizing & Optimizing VHDL & Verilog HDL Projects with Synopsys Software
  - Using FPGA Compiler N-Input LUT Optimization for FLEX 6000, FLEX 8000, or FLEX 10K Devices
- Examples:
  - MAX 7000 & MAX 9000 Synthesis Example
  - DesignWare FLEX 8000 Synthesis Example
- Entering Resource Assignments
  - Assigning Pins, Logic Cells & Chips
  - Assigning Cliques
  - Assigning Logic Options
  - Modifying the Assignment & Configuration File with the **setacf** utility
  - Converting Synopsys Timing Constraints into MAX+PLUS II-Compatible Format with the **syn2acf** Utility
  - Converting Synopsys Hierarchical Timing Constraints into MAX+PLUS II-Compatible Format with the **gen_iacf** and **gen_hacf** Utilities
- Performing a Pre-Routing or Functional Simulation with VSS
- Resynthesizing a Design Using the **alt.vtl** Library & a MAX+PLUS II SDF Output File

Related Topics:

- Go to the following MAX+PLUS II ACCESS^{SM} Key topics for related information:
  - Compiling Projects with MAX+PLUS II Software
  - Programming Altera Devices
  - Using Synopsys FPGA Express & MAX+PLUS II Software
  - Using Synopsys PrimeTime & MAX+PLUS II Software
  - Using Synopsys VSS & MAX+PLUS II Software
  - Go to the following topics, which are available on the web, for additional information:
    - MAX+PLUS II Development Software
    - Altera Programming Hardware
    - Synopsys web site (http://www.synopsys.com)

Setting Up the MAX+PLUS II/Synopsys Working Environment

To use the MAX+PLUS® II software with Synopsys software, you must first install the MAX+PLUS II software, then establish an environment that facilitates entering and processing designs by modifying your Synopsys configuration files. The MAX+PLUS II/Synopsys interface is installed automatically when you install the MAX+PLUS II software on your workstation. Go to *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual for more information on installation and details on the directories that are created during MAX+PLUS II installation. Go to MAX+PLUS II/Synopsys Interface File Organization for information about the MAX+PLUS II/Synopsys directories that are created during MAX+PLUS II installation.

The information presented here assumes that you are using C shell and that your MAX+PLUS II system directory is **/usr/maxplus2**. If not, you must use the appropriate syntax and procedures to set environment variables for your shell.

To set up your working environment for the MAX+PLUS II/Synopsys interface, follow these steps:

1. Ensure that you have correctly installed the MAX+PLUS II and Synopsys software versions described in the MAX+PLUS II/Synopsys Software Requirements.

2. Add technology, synthetic, and link library settings to your **.synopsys_dc.setup** configuration file, as described in Setting Up Design Compiler & FPGA Compiler Configuration Files.

   To use the DesignWare interface with FLEX® 6000, FLEX 8000, and FLEX 10K devices, follow the steps in Setting Up the DesignWare Interface.

3. Add simulation library settings to your **.synopsys_vss.setup** file, and analyze the libraries, as described in Setting Up VSS Configuration Files.

4. Add the **/usr/maxplus2/bin** directory to the `PATH` environment variable in your **.cshrc** file in order to run the MAX+PLUS II software.

   (Optional) Change the path in the first line of the perl script files, which are located in the `$ALT_HOME`**/synopsys/bin** directory to specify the correct path of your local perl executable file.

Related Topics:

- Go to the following topics, which are available on the web, for additional information:
  - FLEX Devices
  - ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
  - This manual is also available in 4 parts:
    - Preface & Section 1: MAX+PLUS II Installation
    - Section 2: MAX+PLUS II - A Perspective
    - Section 3: MAX+PLUS II Tutorial
    - Appendices, Glossary & Index

MAX+PLUS II/Synopsys Software Requirements

The following applications are used to generate, process, synthesize, and verify a project with MAX+PLUS® II and Synopsys software:

Synopsys		Altera
version 1998.02: Design Compiler FPGA Compiler Design Analyzer (optional) VHDL Compiler	HDL Compiler for Verilog VHDL System Simulator (VSS) (optional) PrimeTime version 1998.02-PT2.1(optional)	MAX+PLUS II version 9.4

Compilation with the Synopsys Design Compiler and FPGA Compiler is available only on Sun SPARCstations running Solaris 2.4 or higher.

The MAX+PLUS II **read.me** file provides up-to-date information on which versions of Synopsys applications are supported by the current version of MAX+PLUS II. It also provides information on installation and operating requirements. You should read the **read.me** file on the CD-ROM before installing the MAX+PLUS II software. After installation, you can open the **read.me** file from the MAX+PLUS II Help menu.

Setting Up Design Compiler & FPGA Compiler Configuration Files

The **.synopsys_dc.setup** configuration file allows you to set both Design Compiler and FPGA Compiler variables. The compilers read **.synopsys_dc.setup** files from three directories, in the following order:

1. The Synopsys root directory

2. Your home directory
3. The directory where you start the Design Compiler or FPGA Compiler software

The most recently read configuration file has highest priority. For example, a configuration file in the directory where you start the Design Compiler or FPGA Compiler software has priority over the other configuration files, and a configuration file in the home directory has priority over a configuration file in the root directory.

To set up your configuration files, follow these steps:

1. Add the lines shown in Figure 1 to your **.synopsys_dc.setup** configuration file. Altera provides a sample **.synopsys_dc.setup** file in the **./synopsys/config** directory. Figure 1 shows an excerpt from that sample file.

   Figure 1. Excerpt from Sample .synopsys_dc.setup File

   ```
   search_path = {./usr/maxplus2/synopsys/library/alt_syn/<device family>/lib};
   target_library = {<technology library>};
   symbol_library = {altera.sdb};
   link_library = {<technology library>};
   edifout_netlist_only = "true"
   edifout_power_and_ground_representation = "net"
   edifout_power_net_name = "VDD"
   edifout_ground_net_name = "GND"
   edifout_no_array = "false"
   edifin_power_net_name = "VDD"
   edifin_ground_net_name = "GND"
   compile_fix_multiple_port_nets = "true"
   bus_naming_style = "%s<%d>"
   bus_dimension_separator_style = "><"
   bus_inference_style = "%s<%d>"
   ```

2. Specify one of the Design Compiler & FPGA Compiler Technology Libraries for the `target_library` and `link_library` parameters in the **.synopsys_dc.setup** file.

3. If you will instantiate architecture control logic functions from the **alt_mf** library, add the following line to your **.synopsys_dc.setup** file:

   ```
   define_design_lib altera -path /usr/maxplus2/synopsys/library/alt_mf/lib ↵
   ```

   If you wish to use the VHDL System Simulator (VSS) software to simulate a VHDL design containing **alt_mf** library functions, you must compile this library with the **analyze_vss** script. See Setting Up VSS Configuration Files for more information.

4. If you will use the DesignWare interface for FLEX® 6000, FLEX 8000, or FLEX 10K designs, enter additional lines in your **.synopsys_dc.setup** file, as described in Setting Up the DesignWare Interface.

5. Specify one of the following families for the *<device family>* variable in the `search_path` parameter: `max5000`, `max7000`, `max9000`, `flex6000`, `flex8000`, or `flex10k`.

6. If you wish to resynthesize a design for a different device family, modify the **.synopsys_dc.setup** file by following the steps described in Resynthesizing a Design Using the **alt_vtl** Library & a MAX+PLUS II SDF Output File.

Related Topics:

- Go to MAX+PLUS® II /Synopsys Interface File Organization in these MAX+PLUS II ACCESS℠ Key topics for related information.
- Go to the following topics, which are available on the web, for additional information:
  - FLEX Devices
  - MAX Devices
  - Classic Device Family

Setting Up the DesignWare Interface

The DesignWare interface synthesizes FLEX® 6000 , FLEX 8000 and FLEX 10K designs by operator inference. It replaces the HDL operators +, -, >, <, >=, and <= with FLEX-optimized design implementations.

Altera provides DesignWare Synthetic Libraries that are pre-compiled for the current version of Synopsys tools. These library files are located in the **/usr/maxplus2/synopsys/library/alt_syn/<***device family***>/lib** directory.

To use the DesignWare interface with FLEX 6000, FLEX 8000 and FLEX 10K devices, follow these steps:

1. Add `synthetic_library` and `define_design_lib` parameters to your **.synopsys_dc.setup** configuration file and modify the `link_library` parameter as shown in Table 1 or Table 2.

Table 1. DesignWare Parameters to Add to the .synopsys_dc.setup File for the Design Compiler Software

Device Family	*Parameters to Add to the .synopsys_dc.setup File*

FLEX 6000	`synthetic_library = {flex6000<`*speed grade*`>.sldb};` ↵
	`link_library = {flex6000<`*speed grade*`>.sldb flex6000<`*speed grade*`>.db};` ↵
	`define_design_lib DW_FLEX6000<`*speed grade*`> -path`
	`/usr/maxplus2/synopsys/library/alt_syn/flex6000/lib/`
	`dw_flex6000<`*speed grade*`>` ↵

FLEX 8000	`synthetic_library = {flex8000[<`*speed grade*`>].sldb};` ↵
	`link_library = {flex8000[<`*speed grade*`>].sldb flex8000[<`*speed grade*`>].db};` ↵
	`define_design_lib DW_FLEX8000[<`*speed grade*`>] -path`
	`/usr/maxplus2/synopsys/library/alt_syn/flex8000`
	`/lib/dw_flex8000[<`*speed grade*`>]` ↵

FLEX 10K	`synthetic_library = {flex10k[<`*speed grade* `>].sldb};` ↵
	`link_library = {flex10k[<`*speed grade*`>].sldb flex10k[<`*speed grade*`>].db};` ↵
	`define_design_lib DW_FLEX10k[<`*speed grade*`>] -path`
	`/usr/maxplus2/synopsys/library/alt_syn/flex10k/lib`
	`/dw_flex10k[<`*speed grade*`>]` ↵

Table 2. DesignWare Parameters to Add to the .synopsys_dc.setup File for the FPGA Compiler Software

Device Family	*Parameters to Add to the .synopsys_dc.setup File*

FLEX 6000	`synthetic_library = {flex6000`
	`<`*speed grade*`>_fpga.sldb};` ↵
	`link_library = {flex6000<`*speed grade*`>_fpga.sldb flex6000<`*speed grade*`>_fpga.db};` ↵
	`define_design_lib DW_FLEX6000<`*speed grade*`>_FPGA -path`
	`/usr/maxplus2/synopsys/library/alt_syn/flex6000`
	`/lib/dw_flex6000<`*speed grade*`>_fpga` ↵

FLEX 8000	`synthetic_library = {flex8000[<`*speed grade*`>]_fpga.sldb};` ↵ `link_library = {flex8000[<`*speed grade*`>]_fpga.sldb flex8000[<`*speed grade*`>]_fpga.db};` ↵ `define_design_lib DW_FLEX8000[<`*speed grade*`>]_FPGA -path`

```
/usr/maxplus2/synopsys/library/alt_syn/flex8000/lib /dw_flex8000[<speed grade>]_fpga ↵
synthetic_library = {flex10k[<speed grade>]_fpga.sldb}; ↵
link_library = {flex10k[<speed grade>]_fpga.sldb flex10k[<speed grade>]_fpga.db}; ↵
define_design_lib DW_FLEX10k[<speed grade>]_FPGA -path
/usr/maxplus2/synopsys/library/alt_syn/flex10k/lib /dw_flex10k[<speed grade>]_fpga ↵
```

FLEX 10K

2. Specify the libraries listed in Table 3 as your synthetic library and as the first of your link libraries.

   For FLEX 6000 devices, you must specify either `-2` or `-3` for the <*speed grade*> variable. For FLEX 8000 and FLEX 10K devices, you can specify `-2`, `-3`, `-4`, `-5`, or `-6`; or `-2`, `-3`, `-4`, or `-5`; respectively, for the <*speed grade*> variable. If you do not specify a speed grade for FLEX 8000 or FLEX 10K devices, the MAX+PLUS® II software selects the fastest device in the specified family as the target device.

Table 3. FLEX 6000, FLEX 8000 & FLEX 10K DesignWare Synthetic Libraries

Altera® Device Family	Synopsys Design Compiler	Synopsys FPGA Compiler
FLEX 6000 Synthetic Library	**flex6000-2.sldb** **flex6000-3.sldb**	**flex6000-2_fpga.sldb** **flex6000-3_fpga.sldb**
FLEX 8000 Synthetic Library	**flex8000.sldb** **flex8000-2.sldb** **flex8000-3.sldb** **flex8000-4.sldb** **flex8000-5.sldb** **flex8000-6.sldb**	**flex8000_fpga.sldb** **flex8000-2_fpga.sldb** **flex8000-3_fpga.sldb** **flex8000-4_fpga.sldb** **flex8000-5_fpga.sldb** **flex8000-6_fpga.sldb**
FLEX 10K Synthetic Library	**flex10k.sldb** **flex10k-2.sldb** **flex10k-3.sldb** **flex10k-4.sldb** **flex10k-5.sldb**	**flex10k_fpga.sldb** **flex10k-2_fpga.sldb** **flex10k-3_fpga.sldb** **flex10k-4_fpga.sldb** **flex10k-5_fpga.sldb**

3. If necessary, compile the DesignWare libraries, as described in Updating DesignWare Libraries. Altera provides pre-compiled DesignWare libraries, as described above. However, Altera also provides compatible source files and scripts that allow you to automate the compilation process. These source files allow you to use DesignWare with any version of the Design Compiler. They also allow you to install components whose source is written in VHDL, even if you are licensed only for the HDL Compiler for Verilog.

Related Topics:

- Go to the following MAX+PLUS II ACCESS℠ Key topics for related information:
  - Setting Up Design Compiler & FPGA Compiler Configuration Files
  - DesignWare FLEX 8000 Synthesis Example
  - Design Compiler & FPGA Compiler Technology Libraries
- Go to the following topics, which are available on the web, for additional information:
  - FLEX 6000 Device Family
  - FLEX 8000 Device Family
  - FLEX 10K Device Family

Updating DesignWare Libraries

Although Altera provides DesignWare libraries that are pre-compiled for the current version of Synopsys tools, you

may wish to recompile the libraries.

Altera provides compilable source files and scripts that allow you to automate the compilation process. These source files allow you to use DesignWare software with any version of the Design Compiler or FPGA Compiler software.They also allow you to install components whose source is written in VHDL, even if you are licensed only for the Verilog HDL Compiler software.

Source files for the Design Compiler software are automatically installed in the following directories:

- **/usr/maxplus2/synopsys/library/alt_syn/flex10k/src/dw_flex10k[<*speed grade*>]**
- **/usr/maxplus2/synopsys/library/alt_syn/flex8000/src/dw_flex8000[<*speed grade*>]**
- **/usr/maxplus2/synopsys/library/alt_syn/flex6000/src/dw_flex6000<*speed grade*>**

Source files for the FPGA Compiler are automatically installed in the following directories:

- **/usr/maxplus2/synopsys/library/alt_syn/flex10k/src/dw_flex10k[<*speed grade*>]_fpga**
- **/usr/maxplus2/synopsys/library/alt_syn/flex8000/src/dw_flex8000[<*speed grade*>]_fpga**
- **/usr/maxplus2/synopsys/library/alt_syn/flex6000/src/dw_flex6000<*speed grade*>_fpga**

Table 1. Commands for Compiling the Library

Device Family	Synopsys Compiler	Commands for Compiling the Library *Note (1)*
FLEX®6000	Design Compiler	`cd /usr/maxplus2/synopsys/library/alt_syn/flex6000/src/dw_flex6000<`*speed grade*`>` ↵ `dw_flex6000.script` ↵
	FPGA Compiler	`cd/usr/maxplus2/synopsys/library/alt_syn/flex6000/src/dw_flex6000<`*speed grade*`>_fpga` ↵ `dw_flex6000.script` ↵
FLEX 8000	Design Compiler	`cd/usr/maxplus2/synopsys/library/alt_syn/flex8000/src/dw_flex8000[<`*speed grade*`>]` ↵ `dw_flex8000.script` ↵
	FPGA Compiler	`cd/usr/maxplus2/synopsys/library/alt_syn/flex8000/src/dw_flex8000[<`*speed grade*`>]_fpga` ↵ `dw_flex8000.script` ↵
FLEX 10K	Design Compiler	`cd/usr/maxplus2/synopsys/library/alt_syn/flex10k/src/dw_flex10k[<`*speed grade*`>]` ↵ `dw_flex10k.script` ↵
	FPGA Compiler	`cd/usr/maxplus2/synopsys/library/alt_syn/flex10k/src/dw_flex10k[<`*speed grade*`>]_fpga` ↵ `dw_flex10k.script` ↵

1. For FLEX 6000 devices, you must specify either `-2` or `-3` for the <*speed grade*> variable. For FLEX 8000 and FLEX 10K devices, you must specify `-2`, `-3`, `-4`, `-5`, or `-6`; or `-1`, `-2`, `-3`, `-4`, or `-5`; respectively, for the <*speed grade*> variable.

Go to the following topics for additional information:

- Setting Up the DesignWare Interface
- Setting Up the MAX+PLUS II/Synopsys Working Environment
- Setting Up Design Compiler & FPGA Compiler Configuration Files
- Setting Up VSS Configuration Files

Go to the following topics, which are available on the web, for additional information:

- FLEX 6000 Device Family
- FLEX 8000 Device Family
- FLEX 10K Device Family

Design Compiler & FPGA Compiler Technology Libraries

The Altera® -provided Design Compiler and FPGA Compiler technology libraries contain primitives that the Synopsys compilers use to map your designs to the target device architecture. These primitives contain timing and area information that the Synopsys compilers use to meet area and performance requirements. Table 1 shows the functions provided in these libraries. Choose **Primitives** from the MAX+PLUS II Help menu for detailed information on these functions.

Altera recommends instantiating these functions directly in your designs only if the Synopsys compilers do not appear to recognize the functions when synthesizing your design, or if you prefer to hand-optimize certain portions of your design.

Table 1. Altera-Provided Primitives

Name *Note (1), Note (2)*	Description	Name	Description
LCELL	Logic cell buffer primitive	EXP	MAX® 5000, MAX 7000, and MAX 9000 Expander buffer primitive
GLOBAL	Global input buffer primitive	SOFT	Soft buffer primitive
CASCADE	FLEX® 6000, FLEX 8000, and FLEX 10K cascade buffer primitive	OPNDRN	FLEX 6000, FLEX 8000, and FLEX 10K Open-drain buffer primitive
CARRY	FLEX 6000, FLEX 8000, and FLEX 10K cascade buffer primitive	DFF DFFE DFFS *Note (2)*	D-type flipflop with Clock Enable primitive
LATCH	Latch primitive	TFF TFFE TFFS *Note (2)*	T-type flipflop primitive
TRIBUF	Tri-state buffer primitive		

Notes:

(1) All buffer primitive names except OPNDRN must be prefixed with an "A" in FLEX 6000, FLEX 8000, and FLEX 10K designs. The TRIBUF primitive is equivalent to the TRI primitive in the MAX+PLUS II software.

(2) The DFFE and TFFE primitives include a Clock Enable input; the DFFS and TFFS primitives are equivalent to DFF and TFF primitives without Clear or Preset inputs. For designs that are targeted to FLEX 6000 devices, you should use the DFFE or TFFE primitive only if the design contains either a Clear or Preset signal, but not both. If your design contains both a Clear and a Preset signal, you must use the DFFE6K primitive.

The VHDL simulation model **/usr/maxplus2/synopsys/library/alt_pre/<***device family***>/src/<***device*

family>**_components.vhd** file shows the exact cell and pin names for each device family. The Verilog HDL simulation file **/usr/maxplus2/synopsys/library/alt_pre/verilog/src/altera.v** shows the functionality of these cells.

Table 2 lists the technology library names.

Table 2. Altera Technology Libraries

Altera Device Family	Synopsys Design Compiler	Synopsys FPGA Compiler
FLEX® 10K devices	**flex10k.db** **flex10k-2.db** **flex10k-3.db** **flex10k-4.db** **flex10k-5.db**	**flex10k_fpga.db** **flex10k-2_fpga.db** **flex10k-3_fpga.db** **flex10k-4_fpga.db** **flex10k-5_fpga.db**
FLEX 8000 devices	**flex8000.db** **flex8000-2.db** **flex8000-3.db** **flex8000-4.db** **flex8000-5.db** **flex8000-6.db**	**flex8000_fpga.db** **flex8000-2_fpga.db** **flex8000-3_fpga.db** **flex8000-4_fpga.db** **flex8000-5_fpga.db** **flex8000-6_fpga.db**
FLEX 6000 devices	**flex6000-2.db** **flex6000-3.db**	**flex6000-2_fpga.db** **flex6000-3_fpga.db**
MAX® 9000 devices	**max9000.db**	**max9000_fpga.db**
MAX 7000, MAX 7000E, MAX 7000S, & MAX 7000A devices	**max7000.db**	**max7000_fpga.db**
MAX 5000 & Classic® devices	**max5000.db**	**max5000_fpga.db**

Related Topics:

- Go to MAX+PLUS® II /Synopsys Interface File Organization in these MAX+PLUS II ACCESS℠ Key topics for related information.
- Go to the following topics, which are available on the web, for additional information:
  - FLEX Devices
  - MAX Devices
  - Classic Device Family

Altera VHDL & Verilog HDL alt_mf Logic Function Library

The **alt_mf** library contains behavioral VHDL and Verilog HDL models of the Altera® logic functions shown in Table 1. VHDL or Verilog HDL files that instantiate these functions can be simulated with the VHDL System Simulator (VSS) software or the Cadence Verilog-XL simulator, respectively, both before and after being compiled with the Synopsys Design Compiler or FPGA Compiler software.

Table 1. Altera-Provided Architecture Control Logic Functions

Name	Description
`a_8fadd`	8-bit full adder
`a_8mcomp`	8-bit magnitude comparator
`a_8count`	8-bit up/down counter

`a_81mux` 8-to-1 multiplexer

For detailed information on these functions, choose **Search for Help** on from the MAX+PLUS® II Help menu and type the function name, without the "`a_`" prefix.

The behavioral descriptions of these four functions are contained in the **/usr/maxplus2/synopsys/library/alt_mf/src** directory, which contains the following files:

File:	Description:
mf.vhd	Contains behavioral VHDL descriptions of the logic functions.
mf_components.vhd	Contains VHDL Component Declarations for the logic functions.
mf.v	Contains behavioral Verilog HDL descriptions of the logic functions.

If you wish to simulate a VHDL design containing these logic functions, you can use the Altera-provided shell script **analyze_vss** to create a design library called **altera**. This library allows you to reference the functions through the VHDL Library and Use Clauses, which direct the Design Compiler or FPGA Compiler software to incorporate the library files when it compiles your top-level design file. The **analyze_vss** shell script creates the **altera** design library by analyzing the VHDL System Simulator (VSS) simulation models in the **/usr/maxplus2/synopsys/library/alt_mf/lib** directory. See Setting Up VSS Configuration Files for more information on using the **analyze_vss** shell script.

Complete VHDL and Verilog HDL behavioral descriptions of these logic functions are included in the **mf.vhd** and **mf.v** files so that you can optionally retarget your design to other technology libraries.

Altera DesignWare FLEX 6000, FLEX 8000 & FLEX 10K Synthetic Libraries

The Altera® DesignWare interface for the FLEX® 6000, FLEX 8000, and FLEX 10K device families provides accurate area and timing prediction for designs that have been synthesized by the Synopsys design tools and targeted for FLEX devices. Altera's DesignWare interface also ensures that the area and timing information closely matches the final FLEX device implementation generated by the MAX+PLUS® II Compiler. The DesignWare interface synthesizes FLEX 6000 , FLEX 8000 and FLEX 10K designs by operator inference. This interface supports bus widths of up to 32 bits, except adder functions, which support bus widths of up to 64 bits.

The Altera DesignWare interface for FLEX devices offers three major advantages to Synopsys designers:

- Automatic access to FLEX carry and cascade chain functions
- Optimal routing of FLEX designs
- Improved area and performance prediction capability in Synopsys tools

Table 1 lists the Altera DesignWare synthetic libraries for FLEX 6000, FLEX 8000, and FLEX 10K devices.

Table 1. FLEX 6000, FLEX 8000 & FLEX 10K DesignWare Synthetic Libraries

Altera Device Family	Synopsys Design Compiler	Synopsys FPGA Compiler
FLEX 6000 Synthetic Library	**flex6000-2.sldb**	**flex6000-2_fpga.sldb**
	flex6000-3.sldb	**flex6000-3_fpga.sldb**
FLEX 8000	**flex8000.sldb**	**flex8000_fpga.sldb**
	flex8000-2.sldb	**flex8000-2_fpga.sldb**
	flex8000-3.sldb	**flex8000-3_fpga.sldb**

Synthetic Library	**flex8000-4.sldb**	**flex8000-4_fpga.sldb**
	flex8000-5.sldb	**flex8000-5_fpga.sldb**
	flex8000-6.sldb	**flex8000-6_fpga.sldb**
FLEX 10K Synthetic Library	**flex10k.sldb**	**flex10k_fpga.sldb**
	flex10k-2.sldb	**flex10k-2_fpga.sldb**
	flex10k-3.sldb	**flex10k-3_fpga.sldb**
	flex10k-4.sldb	**flex10k-4_fpga.sldb**
	flex10k-5.sldb	**flex10k-5_fpga.sldb**

Table 2 lists functions included in the DesignWare FLEX 6000, FLEX 8000, and FLEX 10K synthetic libraries. Refer to DesignWare FLEX 8000 Synthesis Example for an example showing how DesignWare synthesis affects design processing.

Table 2. FLEX 6000, FLEX 8000, and FLEX 10K Synthetic Library Functions

Name	Function
`flex_add`	Sum of A, B, and Carry-In
`flex_carry`	Carry of A, B, and Carry-In
`flex_sub`	Difference of A, B, and Borrow-In
`flex_borrow`	Borrow of A, B, and Borrow-In
`flex_gt, flex_sgt`	Greater than (`flex_gt` is unsigned; `flex_sgt` is signed)
`flex_carry_gt`	Greater than Carry
`flex_lt, flex_slt`	Less than (`flex_lt` is unsigned; `flex_slt` is signed)
`flex_carry_lt`	Less than Carry
`flex_gteq, flex_sgteq`	Greater than or equal to (`flex_gteq` is unsigned; `flex_sgteq` is signed)
`flex_carry_gteq`	Greater than or equal to Carry
`flex_inc`	Incrementer (Count = Count + 1)
`flex_carry_inc`	Incrementer Carry (Count = Count + 1)
`flex_dec`	Decrementer (Count = Count - 1)
`flex_carry_dec`	Decrementer Carry (Count = Count - 1)
`flex_lteq, flex_slteq`	Less than or equal to (`flex_lteq` is unsigned; `flex_slteq` is signed)
`flex_carry_lteq`	Less than or equal to Carry
`flex_count`	Counter
`aflex_carry_count`	Counter Carry
`flex_add_sub`	Adder/Subtractor
`flex_inc_dec`	Incrementer/Decrementer
`flex_umult, flex_smult`	Multiplier (`flex_umult` is unsigned; `flex_smult` is signed)

Related Topics:

- Go to the following sources for related information:
  - Setting Up the DesignWare Interface in these MAX+PLUS II ACCESS^{SM} Key topics
  - ***Synopsys DesignWare Databook***
  - ***VHDL Compiler Reference Manual***
- Go to the following topics, which are available on the web, for additional information:
  - FLEX 6000 Device Family
  - FLEX 8000 Device Family

- FLEX 10K Device Family

Altera Post-Synthesis Libraries

The **/usr/maxplus2/synopsys/library/alt_post/syn/lib** directory contains the post-synthesis library for technology mapping and timing back-annotation. The Altera® -provided **alt_vtl.db** file in this library contains over three dozen MAX+PLUS® II -generated logic functions.

MAX+PLUS II/Synopsys Interface File Organization

Table 1 shows the MAX+PLUS® II /Synopsys interface subdirectories that are created in the MAX+PLUS II system directory (by default, the **/usr/maxplus2** directory) during the MAX+PLUS II software installation. For information on the other directories that are created during the MAX+PLUS II software installation, see "MAX+PLUS II File Organization" in *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual.

You must add the **/usr/maxplus2/bin** directory to the `PATH` environment variable in your **.cshrc** file in order to run the MAX+PLUS II software.

Table 1. MAX+PLUS II Directory Organization

Directory	Description
./synopsys/bin	Contains script programs to convert Synopsys timing constraints into MAX+PLUS II Assignment & Configuration File (**.acf**) format, and to analyze VHDL System Simulator simulation models.
./synopsys/config	Contains sample **.synopsys_dc.setup** and **.synopsys_vss.setup** files.
./synopsys/examples	Contains sample files, including those discussed in these ACCESS Key Guidelines.
./synopsys/library/alt_pre/<*device family***>/src**	Contains VHDL simulation libraries for functional simulation of VHDL projects.
./synopsys/library/alt_pre/verilog/src	Contains the Verilog HDL functional simulation library for Verilog HDL projects.
./synopsys/library/alt_pre/vital/src	Contains the VITAL 95 simulation library. You use this library when you perform functional simulation of the design before compiling it with the MAX+PLUS II software.
./synopsys/library/alt_syn//<*device family***>/lib**	Contains interface files for the MAX+PLUS II/Synopsys interface. Technology libraries in this directory allow the Design Compiler and FPGA Compiler to map designs to Altera® device architectures.
./synopsys/library/alt_mf/src	Contains behavioral VHDL models of some Altera macrofunctions, along with their component declarations. The `a_81mux`, `a_8count`, `a_8fadd`, and `a_8mcomp` macrofunctions are currently supported. Libraries in this directory allow you to instantiate, synthesize, and simulate these macrofunctions.
./synopsys/library/alt_post/syn/lib	Contains the post-synthesis library for technology mapping.
./synopsys/library/alt_post/sim/src	Contains the VHDL source files for the VITAL 95-compliant library. You use this library when you perform simulation of the design after compiling it with the MAX+PLUS II software.

Related Topics:

- Go to the following topics, which are available on the web, for additional information:
  - ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
  - This manual is also available in 4 parts:
    - Preface & Section 1: MAX+PLUS II Installation
    - Section 2: MAX+PLUS II - A Perspective
    - Section 3: MAX+PLUS II Tutorial
    - Appendices, Glossary & Index

MAX+PLUS II Project File Structure

In MAX+PLUS® II, a project name is the name of a top-level design file, without the filename extension. This design file can be an EDIF, Verilog HDL, or VHDL netlist file; an AHDL TDF; or any other MAX+PLUS II-supported design file. The EDIF netlist file must be created by Synopsys and imported into MAX+PLUS II as an EDIF Input File.

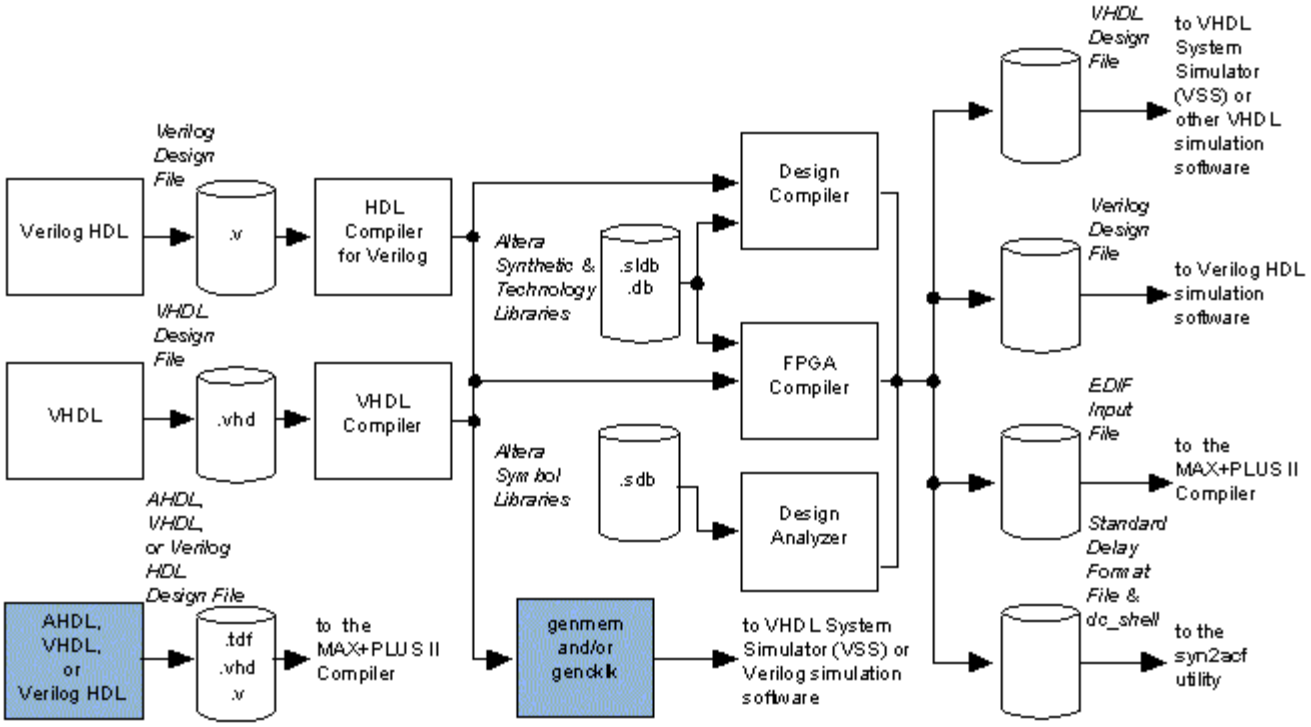
MAX+PLUS II stores the connectivity data on the links between design files in a hierarchical project in a Hierarchy Interconnect File (**.hif**), but refers to the entire project only by its project name. The MAX+PLUS II Compiler uses the HIF to build a single, fully flattened project database that integrates all the design files in a project hierarchy.

Synopsys Design Entry Flow

Figure 1 below shows the design entry flow for the MAX+PLUS® II /Synopsys interface.

Figure 1. MAX+PLUS II/Synopsys Design Entry Flow

Altera-provided items are shown in blue.



Creating VHDL Designs for Use with MAX+PLUS II Software

You can create VHDL design files with the MAX+PLUS® II Text Editor or another standard text editor and save them in the appropriate directory for your project. The MAX+PLUS II Text Editor offers the following advantages:

- VHDL templates are available with the **VHDL Templates** command (Templates menu). These templates are also available in the ASCII **vhdl.tmp** file, which is located in the **/usr/maxplus2** directory.
- If you use the MAX+PLUS II Text Editor to create your VHDL design, you can use the **Syntax Coloring** command (Options menu). The Syntax Coloring feature displays keywords and other elements of text in text files in different colors to distinguish them from other forms of syntax.

Once you have created a VHDL design, you can use the Design Compiler or FPGA Compiler to synthesize and optimize it, and then generate an EDIF netlist file that can be processed with the MAX+PLUS II software.

To create a VHDL design that can be synthesized and optimized with the Design Compiler or FPGA Compiler, follow these steps:

1. Instantiate logic functions with a Component Instantiation, and include a Component Declaration for each function. Altera provides simulation models for the following types of logic functions:
    - Primitives in the Design Compiler & FPGA Compiler Technology Libraries. Go to Primitive & Old-Style Macrofunction Instantiation Example for VHDL for an example.
    - Architecture Control Logic functions in the **alt_mf** library, which includes the `a_8count`, `a_8mcomp`, `a_8fadd`, and `a_81mux` functions. See MAX+PLUS II Architecture Control Logic Function Instantiation Example for VHDL for an example.
    - The DesignWare up/down counter function (`DW03_updn_ctr`). Go to DesignWare Up/Down Counter Function Instantiation Example for VHDL for an example.
    - RAM and ROM functions generated with the **genmem** utility. Go to Instantiating RAM & ROM Functions in VHDL for instructions.
    - The `clklock` megafunction, which is supported for selected FLEX 10K devices. This function is generated with the **genclk** utility. Go to Instantiating the clklock Megafunction in VHDL or Verilog

HDL for instructions.

- MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.

  You can also instantiate any other Altera macrofunction or non-parameterized megafunction, i.e., functions not listed above, for which no simulation models or technology library support is available. These functions are treated as "black boxes" during processing with the Design Compiler or FPGA Compiler. See Primitive & Old-Style Macrofunction Instantiation Example for VHDL for an example.

For information on MAX+PLUS II primitives, megafunctions, and macrofunctions, choose **Primitives**, **Megafunctions/LPM**, or **Old-Style Macrofunctions** from the MAX+PLUS II Help menu. When searching for information on the **alt_mf** library functions, drop the initial "`a_`" from the function name.

2. (Optional) If you instantiate a "black box" logic function for which no simulation/techology library support is available, create a hollow-body design description in order to prevent the Design Compiler or FPGA Compiler from issuing a warning message. See Primitive & Old-Style Macrofunction Instantiation Example for VHDL for an example.

   If you instantiate a "black box" logic function, you must create a Library Mapping File (**.lmf**) to map the function to an equivalent MAX+PLUS II function before you compile the project with the MAX+PLUS II software. See Primitive & Old-Style Macrofunction Instantiation Example for VHDL for an example.

3. Once you have created a VHDL design, you can analyze it, synthesize it, (optional) perform a functional simulation, and generate an EDIF netlist file that can be imported into the MAX+PLUS II software. Go to the following topics for instructions:

   - Synthesizing & Optimizing VHDL & Verilog HDL Projects with Synopsys Software
   - Performing a Pre-Routing or Function Simulation with VSS Software

Installing the Altera-provided MAX+PLUS II/Synopsys Logic interface on your computer automatically creates the following VHDL sample files:

- **/usr/maxplus2/examples/mentor/examples/ministate.vhd**
- **/usr/maxplus2/examples/mentor/examples/count8.vhd**
- **/usr/maxplus2/examples/mentor/examples/tstrom.vhd**

Related Topics:

- Go to Compiling Projects with MAX+PLUS II Software in these MAX+PLUS II ACCESS℠ Key topics for related information.

Primitive & Old-Style Macrofunction Instantiation Example for VHDL

You can instantiate the MAX+PLUS® II primitives listed in Design Compiler & FPGA Compiler Technology Libraries in VHDL designs. These primitives can be used to control synthesis in the MAX+PLUS II software. You can also instantiate MAX+PLUS II megafunctions and old-style macrofunctions.

Go to the following topics for information and examples of how to instantiate functions that are not considered to be hollow bodies, including functions in the **alt_mf** library, RAM and ROM, and the `clklock` megafunction:

- Architecture Control Macrofunction Instantiation Example for VHDL
- Instantiating RAM & ROM Functions in VHDL
- Instantiating the clklock Megafunction in VHDL or Verilog HDL

Unlike other logic functions, MAX+PLUS II primitives do not need to be defined with Component Declarations unless you wish to simulate the design with the VHDL System Simulator (VSS) software. Any references to these primitives are resolved by the Synopsys compilers. All buffer primitives except the `ATRIBUF` and `TRIBUF` primitives also have a "don't touch" attribute already assigned to them, which prevents the Synopsys compilers from optimizing them. The Synopsys compilers also automatically treat mega- and macrofunctions that do not have corresponding synthesis library models as "black boxes."

Figure 1 shows a 4-bit full adder with registered output that also instantiates an `AGLOBAL` or `GLOBAL` primitive. This figure also illustrates the use of global Clock and global Reset pins in the MAX 7000 architecture. The design uses an old-style `7483` macrofunction, which is represented as a hollow body named `fa4`.

Figure 1. 4-Bit Adder Design with Registered Output (adder.vhd)

```
LIBRARY ieee;
   USE ieee.std_logic_1164.ALL;



ENTITY adder IS
   PORT (a, b     : IN  STD_LOGIC_VECTOR(4 DOWNTO 1);
         clk, rst : IN  STD_LOGIC;



cout     : OUT STD_LOGIC;
         regsum   : OUT STD_LOGIC_VECTOR(4 DOWNTO 1));
   END adder;



ARCHITECTURE MAX7000 OF adder IS



SIGNAL sum            : STD_LOGIC_VECTOR(4 DOWNTO 1);
SIGNAL ci, gclk, grst : STD_LOGIC;



-- Component Declaration for GLOBAL primitive
-- For FLEX devices, global, a_in, and a_out should be replaced with
-- aglobal, in1, and Y, respectively
COMPONENT global
   PORT (a_in      : IN  STD_LOGIC;
         a_out     : OUT STD_LOGIC);
END COMPONENT;



-- Component Declaration for fa4 macrofunction
COMPONENT fa4
   PORT (c0,a1,b1,a2,b2,a3,b3,a4,b4 : IN  STD_LOGIC;
         s1,s2,s3,s4,c4             : OUT STD_LOGIC);
END COMPONENT;



BEGIN
   ci <= '0';



-- FA4 Component Instantiation
   u0: fa4
```

```
PORT MAP (ci,a(1),b(1),a(2),b(2),a(3),b(3),a(4),b(4),
            sum(1),sum(2),sum(3),sum(4),cout);



-- GLOBAL Component Instantiation for Clock
-- For FLEX devices, global should be replaced with aglobal
   u1: global
   PORT MAP (clk, gclk);



-- GLOBAL Component Instantiation for Reset
-- For FLEX devices, global should be replaced with aglobal
   u2: global
   PORT MAP (rst, grst);



-- CLOCK process to create registered output
   clocked: PROCESS(gclk,grst)



BEGIN
      IF grst = '0' THEN
      regsum <= "0000"



ELSIF gclk'EVENT AND gclk = '1' THEN
            regsum <= sum;
      END IF;



END PROCESS clocked;

END MAX7000;
```

Before you can analyze the 4-bit adder design, you must first analyze the `fa4` description in Figure 1 with the Synopsys VHDL Compiler software. You can ignore the warning that is issued for any unknown function, including the `fa4` function in this example. If you wish, you can avoid receiving such warning messages by creating a hollow-body description of the function.

A hollow-body VHDL description combines an Entity Declaration with an empty or null Architecture Body. An empty Architecture Body contains the `ARCHITECTURE IS` clause, followed by the `BEGIN` and `END` keywords and a semicolon (;). It does not include any information about the design's function or operation. Figure 2 shows the hollow-body description for the `fa4` function.

Figure 2. Hollow-Body Description of a 4-Bit Full Adder (7483)

```
LIBRARY ieee;
USE     ieee.std_logic_1164.ALL;



-- fa4 maps to 7483. The interface names do not have to match.

ENTITY fa4 IS



PORT (c0,a1,b1,a2,b2,a3,b3,a4,b4 : IN STD_LOGIC;
      s1,s2,s3,s4,c4             : OUT STD_LOGIC);
```

```
END fa4;

ARCHITECTURE map7483 OF fa4 IS


BEGIN

-- This architecture body is left blank, and will map to the
-- 7483 macrofunction in MAX+PLUS II.

END;
```

When you analyze the hollow-body design description with the Synopsys VHDL Compiler software, it produces a hollow-body component that contains a single level of hierarchy with input and output pins, but does not contain any underlying logic.

You can save the synthesized design as an EDIF netlist file (**.edf**) and compile it with the MAX+PLUS II software. After the VHDL Compiler software has successfully processed the design, it generates the schematic shown in Figure 3, which you can view with the Design Analyzer software.

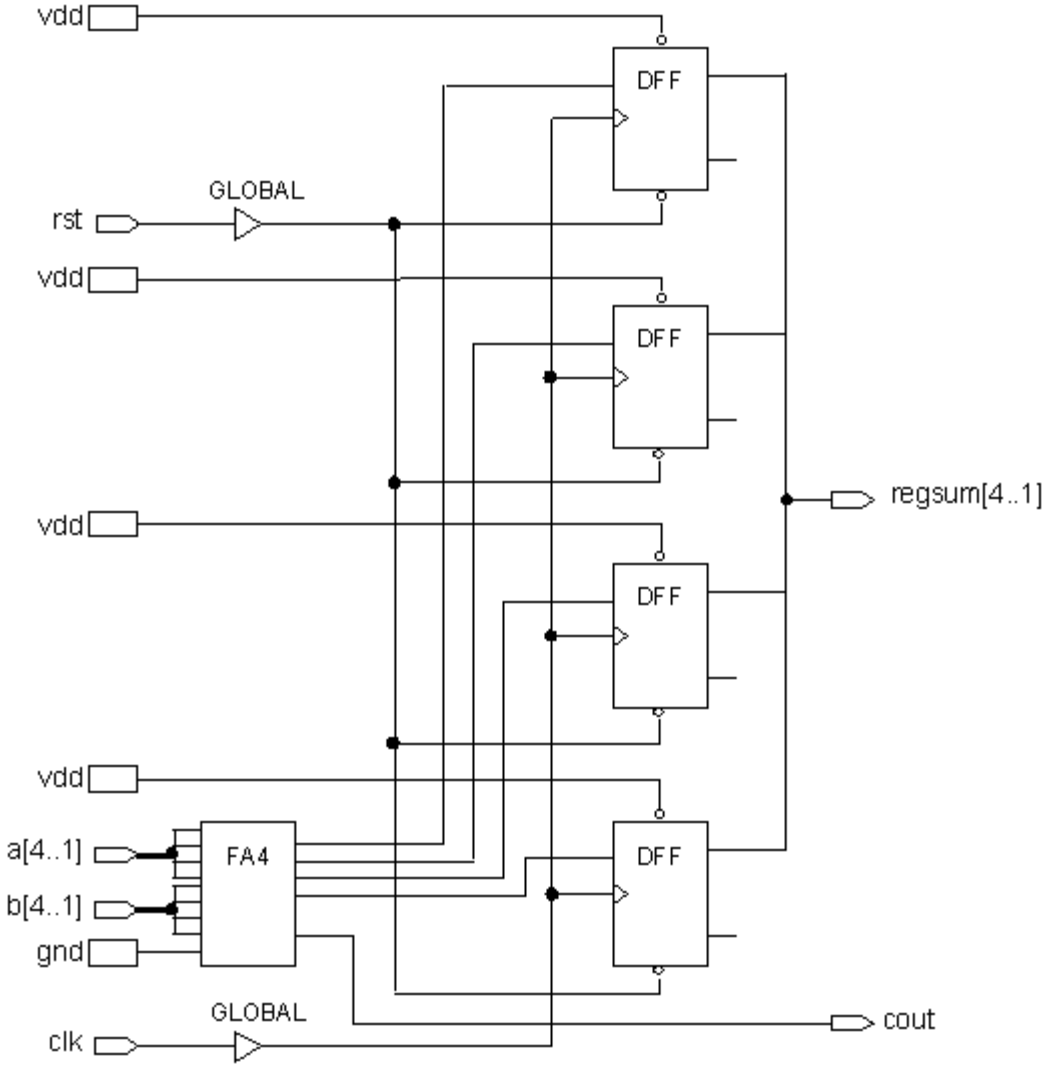


Figure 3. Synthesized Design Generated by the Design Compiler

However, before you compile the EDIF netlist file with the MAX+PLUS II software, you must create the **adder.lmf** file, shown in Figure 3, to map the `fa4` function to the equivalent MAX+PLUS II function (`7483`). You must then specify the LMF as *LMF #2* in the expanded **EDIF Netlist Reader Settings** dialog box (Interfaces

menu) (*LMF #1* is **altsyn.lmf**). For more information about creating LMFs, refer to "Library Mapping Files (**.lmf**)" and "Library Mapping File Format" in MAX+PLUS II Help.

Figure 3. Library Mapping File Excerpt for fa4

```
BEGIN
FUNCTION 7483  (c0, a1, b1, a2, b2, a3, b3, a4, b4,)
RETURNS        (s1, s2, s3, s4, c4)



FUNCTION "fa4" ("c0", "a1", "b1", "a2", "b2", "a3",
               "b3","a4", "b4")
RETURNS        ("s1", "s2", "s3", "s4", "c4")
END
```

When you compile the design with the MAX+PLUS II software, you can disregard the warning "`EDIF cell <name> already has LMF mapping so CONTENTS construct has been ignored`". To verify the global Clock and global Reset usage, as well as the number of logic cells used, see the **adder.rpt** Report File generated by the MAX+PLUS II Compiler.

MAX+PLUS II Architecture Control Logic Function Instantiation Example for VHDL

You can instantiate Altera® -provided logic functions from the **alt_mf** library, which includes the `a_8fadd`, `a_8mcomp`, `a_8count`, and `a_81mux` functions, in VHDL designs. Altera provides behavioral descriptions of these functions that support pre-synthesis/pre-route simulation of your top-level design with the VHDL System Simulator (VSS).

When you instantiate one of these functions, you can either include a Component Declaration for the function, or use the Altera-provided shell script **analyze_vss** to create a design library called **altera** so that you can reference the functions through the VHDL Library and Use Clauses. The Library and Use Clauses direct the Design Compiler or FPGA Compiler to incorporate the library files when it compiles your top-level design file. The **analyze_vss** shell script creates the **altera** design library when it analyzes the VSS simulation models in the **/usr/maxplus2/synopsys/library/alt_mf/lib** directory. See Setting up VSS Configuration Files for more information on using the **analyze_vss** shell script.

Figure 1 shows an example of an 8-bit counter that is instantiated using the `a_8count` function.

Figure 1. Sample VHDL File with Logic Function Instantiation

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;



LIBRARY altera;
USE altera.maxplus2.ALL;



ENTITY counter IS
PORT (clock,ena,load,dnup,set,clear : IN STD_LOGIC;
   i    : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
   q    : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
```

```
cout : OUT STD_LOGIC);
END counter;



ARCHITECTURE structure OF counter IS



BEGIN
   u1   : a_8count



PORT MAP (a=>i(0), b=>i(1), c=>i(2), d=>i(3), e=>i(4),
            f=>i(5), g=>i(6), h=>i(7), ldn=>load, gn=>ena,
            dnup=>dnup, setn=>set, clrn=>clear, clk=>clock,



qa=>q(0), qb=>q(1), qc=>q(2), qd=>q(3), qe=>q(4),
            qf=>q(5), qg=>q(6), qh=>q(7), cout=>cout);



END structure;



CONFIGURATION conf OF counter IS
   FOR structure
   END FOR;
END conf;
```

DesignWare Up/Down Counter Function Instantiation Example for VHDL

The Altera DesignWare Libraries for FLEX devices allow you to instantiate the `DW03_updn_ctr` function, which is the same as the Synopsys `DW03` up/down counter. This function allows you to use the same VHDL code regardless of which FLEX® device is targeted.

Figure 1 shows a VHDL file excerpt with DW03_updn_ctr instantiation.

Figure 1. VHDL File Excerpt with Up/Down Counter Instantiation

```
LIBRARY ieee,DW03;
USE ieee.std_logic_1164.all;
USE DW03.DW03_components.all;



ENTITY updn_4 IS
   PORT (D : IN STD_LOGIC_VECTOR(4-1 DOWNTO 0);
      UP_DN, LD, CE, CLK, RST: IN STD_LOGIC;
      TERCNT : OUT STD_LOGIC;
      Q      : OUT STD_LOGIC_VECTOR(4-1 DOWNTO 0));
END updn_4;



ARCHITECTURE structure OF updn_4 IS



BEGIN
   u0: DW03_updn_ctr
```

```
   GENERIC MAP(width => 4)
   PORT MAP (data => d, clk => clk, reset => rst, up_dn => up_dn,
             load => ld, tercnt => tercnt, cen => ce, count => q);
END structure;
```

Related Topics:

- Go to Setting Up the DesignWare Interface in these MAX+PLUS II ACCESS℠ Key topics for related information.
- Go to the following topics, which are available on the web, for additional information:
  - FLEX 6000 Device Family
  - FLEX 8000 Device Family
  - FLEX 10K Device Family

Instantiating RAM & ROM Functions in VHDL

The MAX+PLUS® II /Synopsys interface offers full support for the memory capabilities of the FLEX® 10K device family, including synchronous and asynchronous RAM and ROM, cycle-shared dual port RAM, dual-port RAM, single-Clock FIFO, and dual-clock FIFO functions. You can use the Altera® -provided **genmem** utility to generate functional simulation models and timing models for these functions. Type `genmem` ↵ at the UNIX prompt to display information on how to use this utility, as well as a list of the functions you can generate.

To instantiate a RAM or ROM function in VHDL, follow these steps:

1. Use the **genmem** utility to generate a memory model by typing the following command at the UNIX prompt:

   `genmem` <*memory type*> <*memory size*> `-vhdl` ↵

   For example: `genmem asynrom 256x15 -vhdl` ↵

2. Create a VHDL design that incorporates the text from the **genmem**-generated Component Declaration, <*memory name*>**.cmp**, and instantiate the <*memory name*> function.

   Figure 1 shows a VHDL design that instantiates **asyn_rom_256x15.vhd**, a 256 x 15 ROM function.

   Figure 1. VHDL Design File with ROM Instantiation (tstrom.vhd)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;



ENTITY tstrom IS
        PORT (
          addr    : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
          memenab : IN STD_LOGIC;
          q       : OUT STD_LOGIC_VECTOR (14 DOWNTO 0));
END tstrom;



ARCHITECTURE behavior OF tstrom IS

COMPONENT asyn_rom_256x15
-- pragma translate_off
     GENERIC (LPM_FILE : string);
```

```
-- pragma translate_on
     PORT (Address : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
           MemEnab : IN STD_LOGIC;
           Q       : OUT STD_LOGIC_VECTOR(14 DOWNTO 0)
     );
END COMPONENT;



BEGIN

   u1: asyn_rom_256x15
-- pragma translate_off
        GENERIC MAP (LPM_FILE => "u1.hex")
-- pragma translate_on
   PORT MAP (Address => addr, MemEnab => memenab, Q =>q);
END behavior;
```

3. (Optional for RAM functions) Specify an initial memory content file:

   - For ROM functions, you must specify the filename of an initial memory content file in the Intel hexadecimal format (**.hex**) or the Altera® Memory Initialization File (**.mif**) format in the Generic Map Clause, with the `LPM_FILE` parameter. See Figure 1. The filename must be the same as the instance name; e.g., the `u1` instance name must be unique throughout the whole project, and must contain only valid VHDL name characters. The initialization file must reside in the directory containing the project's design files.

   - For RAM functions, specifying a memory initialization file is optional. If you want to use it, you must specify it in the Generic Map Clause as described above. If you do not use an initialization file, you should not declare or use the Generic Clause.

   1. The MIF format is supported only for specifying initial memory content when compiling designs within MAX+PLUS II software. You cannot use a MIF to perform simulation with Synopsys tools prior to MAX+PLUS II compilation.

   2. If you use an Intel hexadecimal format file and wish to simulate the design with the VHDL System Simulator (VSS) after MAX+PLUS II compilation, you should use the Synopsys **intelhex** utility to translate the Intel hexadecimal fomat file into a VSS-compatible Synopsys memory file. Refer to the Synopsys ***VHDL System Simulator Software Tool*** manual for details about using the **intelhex** utility.

4. In the VHDL design file, add the compiler directive `-- pragma translate_off` ↵ before the Generic Clause and Generic Map Clause, and add `-- pragma translate_on` ↵ after the Generic Clause and Generic Map Clause. These directives tell the VHDL Compiler software when to stop and start synthesizing. For example, in Figure 1, the `--pragma translate_off` directive instructs the VHDL Compiler software to skip syntax checking until the `--pragma translate_on` directive is read.

5. Because the VHDL Compiler software does not support the data type `string` for the Generic Clause, you must also enter the following command before you read the design:

   ```
   hdlin_translate_off_skip_text=true ↵
   ```

6. The timing model (**.lib**) generated by the **genmem** utility contains pin-to-pin delay information that can be used by the Synopsys Design Compiler and FPGA Compiler software. You must add this timing model to the existing library so that the compiler can access the timing information. Type the following commands at the **dc_shell** prompt:

   ```
   read -f db flex10k[<speed grade>].db ↵
   update_lib flex10k[<speed grade>] <RAM/ROM function name>.lib ↵
   ```

7. (Optional) Enter the following command to update your **flex10k[**<*speed grade*>**].db** file with the RAM/ROM timing information:

`write_lib flex10k[<speed grade>] -o flex10k.db` ↵

8. When you generate the EDIF netlist file from the design, include the bus structure from the RAM or ROM function(s). Go to Setting Up Synopsys Configuration Files for more information.

9. Continue with the steps necessary to complete your VHDL design, as described in Creating VHDL Designs for Use with MAX+PLUS II Software.

Related Topics:

- Go to FLEX 10K Device Family, which is available on the web, for additional information.

Instantiating the clklock Megafunction in VHDL or Verilog HDL

MAX+PLUS® II interfaces with other EDA tools support the `clklock` phase-locked loop megafunction, which can be used with some FLEX® 10K devices, with the **genckllk** utility. Type `gencklk -h` ↵ at the UNIX prompt to display information on how to use this utility. The **genckllk** utility generates VHDL or Verilog HDL functional simulation models and a VHDL Component Declaration template file (**.cmp**).

The **genckllk** utility allows parameters for the `clklock` function to be passed from the VHDL or Verilog HDL file to EDIF netlist format. The **genckllk** utility embeds the parameter values in the `clklock` function name; therefore, the values do not need to be declared explicitly.

To instantiate the `clklock` megafunction in VHDL or Verilog HDL, go through the following steps:

1. Type the following command at the UNIX prompt to generate the `clklock_`*x*`_`*y* file, where *x* is the ClockBoost value and *y* is the input frequency in MHz:

   ✔ Type `gencklk` <*ClockBoost*> <*input frequency*> `-vhdl` ↵ for VHDL designs.

   or:

   ✔ Type `gencklk` <*ClockBoost*> <*input frequency*> `-verilog` ↵ for Verilog HDL designs.

   Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu for more information on the `clklock` megafunction.

2. Create a design file that instantiates the `clklock_`*x*`_`*y* function. The **genckllk** utility automatically generates a VHDL Component Declaration template in the **clklock_*x*_*y*.cmp** file that you can incorporate into a VHDL design file.

Figures 1 and 2 show a `clklock` function with <*ClockBoost*> = 2 and <*input frequency*> = 40 MHz instantiated in VHDL and Verilog HDL design files, respectively.

Figure 1. VHDL Design File with clklock Instantiation (count8.vhd)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY altera;
USE altera.maxplus2.all;     --  Include Altera Component Declarations



ENTITY count8 IS
   PORT (a    : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
         ldn  : IN STD_LOGIC;
```

```
         gn   : IN STD_LOGIC;



dnup : IN STD_LOGIC;
         setn : IN STD_LOGIC;
         clrn : IN STD_LOGIC;
         clk  : IN STD_LOGIC;



co   : OUT STD_LOGIC;
         q    : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END count8;



ARCHITECTURE structure OF count8 IS
   signal clk2x : STD_LOGIC;



COMPONENT clklock_2_40
   PORT (
      INCLK : IN STD_LOGIC;
      OUTCLK : OUT STD_LOGIC
   );
END COMPONENT;



BEGIN
   u1: clklock_2_40
   PORT MAP (inclk=>clk, outclk=>clk2x);



u2: a_8count
   PORT MAP (a=>a(0), b=>a(1), c=>a(2), d=>a(3),
             e=>a(4), f=>a(5), g=>a(6), h=>a(7),
             clk=>clk2x,
             ldn=>ldn,
             gn=>gn,



dnup=>dnup,
             setn=>setn,
             clrn=>clrn,



qa=>q(0), qb=>q(1), qc=>q(2), qd=>q(3),
             qe=>q(4), qf=>q(5), qg=>q(6), qh=>q(7),
             cout=>co);
 END structure;
```

Figure 2. Verilog HDL Design File with clklock Instantiation (count8.v)

```
`timescale 1ns / 10ps
module count8 (a, ldn, gn, dnup, setn, clrn, clk, co, q);
output          co;
output[7:0]     q;



input[7:0]      a;
input           ldn, gn,dnup, setn, clrn, clk;
wire            clk2x;



clklock_2_40 u1 (.inclk(clk), .outclk(clk2x) );
```

```
A_8COUNT u2 (.A(a[0]), .B(a[1]), .C(a[2]), .D(a[3]), .E(a[4]), .F(a[5]),


.G(a[6]), .H(a[7]), .LDN(ldn), .GN(gn), .DNUP(dnup),
              .SETN(setn), .CLRN(clrn), .CLK(clk2x), .QA(q[0]), .QB(q[1]),
              .QC(q[2]), .QD(q[3]), .QE(q[4]), .QF(q[5]), .QG(q[6]),
              .QH(q[7]), .COUT(co) );


endmodule
```

Related Topics:

- Go to FLEX 10K Device Family, which is available on the web, for additional information.

Creating Verilog HDL Designs for Use with MAX+PLUS II Software

You can create Verilog HDL design files with the MAX+PLUS® II Text Editor or another standard text editor and save them in the appropriate directory for your project. The MAX+PLUS II Text Editor offers the following advantages:

- Verilog HDL templates are available with the **Verilog HDL Templates** command (Templates menu). These templates are also available in the ASCII **verilog.tmp** file, which is located in the **/usr/maxplus2** directory.
- If you use the MAX+PLUS II Text Editor to create your Verilog HDL design, you can use the **Syntax Coloring** command (Options menu). The Syntax Coloring feature displays keywords and other elements of text in text files in different colors to distinguish them from other forms of syntax.

Once you have created a Verilog HDL design, you can use the Design Compiler or FPGA Compiler to synthesize and optimize it, and then generate an EDIF netlist file that can be processed with the MAX+PLUS II software.

To create a Verilog HDL design that can be synthesized and optimized with the Design Compiler or FPGA Compiler, follow these steps:

1. Instantiate logic functions with a Module Instantiation, and include a Module Declaration for each function. Altera provides simulation models for the following types of logic functions:
    - Primitives in the Design Compiler & FPGA Compiler Technology Libraries. Go to Primitive & Old-Style Macrofunction Instantiation Example for Verilog HDL for an example.
    - Architecture Control Logic functions in the **alt_mf** library, which includes the `a_8count`, `a_8mcomp`, `a_8fadd`, and `a_81mux` functions. See MAX+PLUS II Architecture Control Logic Function Instantiation Example for Verilog HDL for an example.
    - RAM and ROM functions generated with the **genmem** utility. Go to Instantiating RAM & ROM Functions in VHDL for instructions.
    - The `clklock` megafunction, which is supported for selected FLEX 10K devices. This function is generated with the **genclk** utility. Go to Instantiating the clklock Megafunction in VHDL or Verilog HDL for instructions.
    - MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.

    You can also instantiate any other Altera macrofunction or non-parameterized megafunction, i.e., functions not listed above, for which no simulation models or technology library support is available. These functions

are treated as "black boxes" during processing with the Design Compiler or FPGA Compiler. See Primitive & Old-Style Macrofunction Instantiation Example for Verilog HDL for an example.

For information on MAX+PLUS II primitives, megafunctions, and macrofunctions, choose **Primitives**, **Megafunctions/LPM**, or **Old-Style Macrofunctions** from the MAX+PLUS II Help menu. When searching for information on the **alt_mf** library functions, drop the initial "`a_`" from the function name.

2. (Optional) If you instantiate a "black box" logic function for which no simulation/techology library support is available, create a hollow-body design description in order to prevent the Design Compiler or FPGA Compiler from issuing a warning message. See Primitive & Old-Style Macrofunction Instantiation Example for Verilog HDL for an example.

   If you instantiate a "black box" logic function, you must create a Library Mapping File (**.lmf**) to map the function to an equivalent MAX+PLUS II function before you compile the project with the MAX+PLUS II software. See Primitive & Old-Style Macrofunction Instantiation Example for VHDL for an example.

3. Once you have created a VHDL design, you can analyze it, synthesize it, (optional) perform a functional simulation, and generate an EDIF netlist file that can be imported into the MAX+PLUS II software. Go to the following topics for instructions:

   - Synthesizing & Optimizing VHDL & Verilog HDL Projects with Synopsys Software
   - Performing a Pre-Routing or Function Simulation with VSS Software

   Installing the Altera-provided MAX+PLUS II/Synopsys Logic interface on your computer automatically creates the following VHDL sample files:

   - **/usr/maxplus2/examples/mentor/examples/ministate.v**
   - **/usr/maxplus2/examples/mentor/examples/count8.v**
   - **/usr/maxplus2/examples/mentor/examples/tstrom.v**

Related Topics:

- Go to Compiling Projects with MAX+PLUS II Software in these MAX+PLUS II ACCESS^{SM} Key topics for related information.

Primitive & Old-Style Macrofunction Instantiation Example for Verilog HDL

You can instantiate the MAX+PLUS$^{®}$ II primitives listed in Design Compiler & FPGA Compiler Technology Libraries in Verilog HDL designs. These primitives can be used to control synthesis in the MAX+PLUS II software. You can also instantiate MAX+PLUS II megafunctions and old-style macrofunctions.

Go to the following topics for information and examples of how to instantiate functions that are not considered to be hollow bodies, including functions in the **alt_mf** library, RAM and ROM, and the `clklock` megafunction:

- Architecture Control Macrofunction Instantiation Example for Verilog HDL
- Instantiating RAM & ROM Functions in Verilog HDL
- Instantiating the clklock Megafunction in VHDL or Verilog HDL

Unlike other logic functions, MAX+PLUS II primitives do not need to be defined with hollow-body functions unless you wish to simulate the design with the VHDL System Simulator (VSS) software. Any references to these primitives are resolved by the Synopsys compilers. All buffer primitives except the `ATRIBUF` and `TRIBUF` primitives also have a "don't touch" attribute already assigned to them, which prevents the Synopsys compilers from optimizing them. The Synopsys compilers also automatically treat mega- and

macrofunctions that do not have corresponding synthesis library models as "black boxes."

Figure 1 shows a 4-bit full adder with registered output that also instantiates an `AGLOBAL` or `GLOBAL` primitive. This figure also illustrates the use of global Clock and global Reset pins in the MAX 7000 architecture. The design uses an old-style `7483` macrofunction, which is represented as a hollow body named `fa4`.

Figure 1. 4-Bit Adder Design with Registered Output (adder.v)

```
module adder (a, b, clk, rst, cout, regsum);

output       cout;
output[4:1]  regsum;
input[4:1]   a, b;
input        clk, rst;
wire[4:1]    sum;
reg[4:1]     regsum_int;
wire         grst, gclk;
wire         ci;
assign       ci = 0;



// module instantiation
fa4    u0  ( .c0(ci), .a1(a[1]), .b1(b[1]), .a2(a[2]),
             .b2(b[2]), .a3(a[3]), .b3(b[3]), .a4(a[4]),
             .b4(b[4]), .s1(sum[1]), .s2(sum[2]),
             .s3(sum[3]), .s4(sum[4]), .c4(cout));
// For FLEX devices, GLOBAL, A_IN, and A_OUT should be replaced
// with AGLOBAL, IN1, and Y, respectively
GLOBAL  u1 ( .A_IN(clk), .A_OUT(gclk));
GLOBAL  u2 ( .A_IN(rst), .A_OUT(grst));



always @(posedge gclk or negedge grst)
        if ( !grst )
              regsum_int = 4'b0;
        else regsum_int = sum;
assign regsum = regsum_int;
endmodule



// module declaration for fa4 module
module fa4 ( c0, a1, b1, a2, b2, a3, b3, a4, b4, s1, s2, s3, s4, c4);



output    s1, s2, s3, s4, c4;
   input     c0, a1, b1, a2, b2, a3, b3, a4, b4;
   endmodule



// module declaration for GLOBAL primitive
// For FLEX devices, GLOBAL, A_IN, and A_OUT should be replaced
// with AGLOBAL, IN1, and Y, respectively
module GLOBAL (A_OUT, A_IN);



input       A_IN;
   output      A_OUT;
   endmodule
```

You can analyze the 4-bit adder design with the Synopsys HDL Compiler for Verilog software. The hollow-body description of the `fa4` function is required. It contains port declarations and does not include any information about the design's function or operation. However, the hollow-body description can be in the design file, as shown in Figure 1, or in a separate file, as shown in Figure 2.

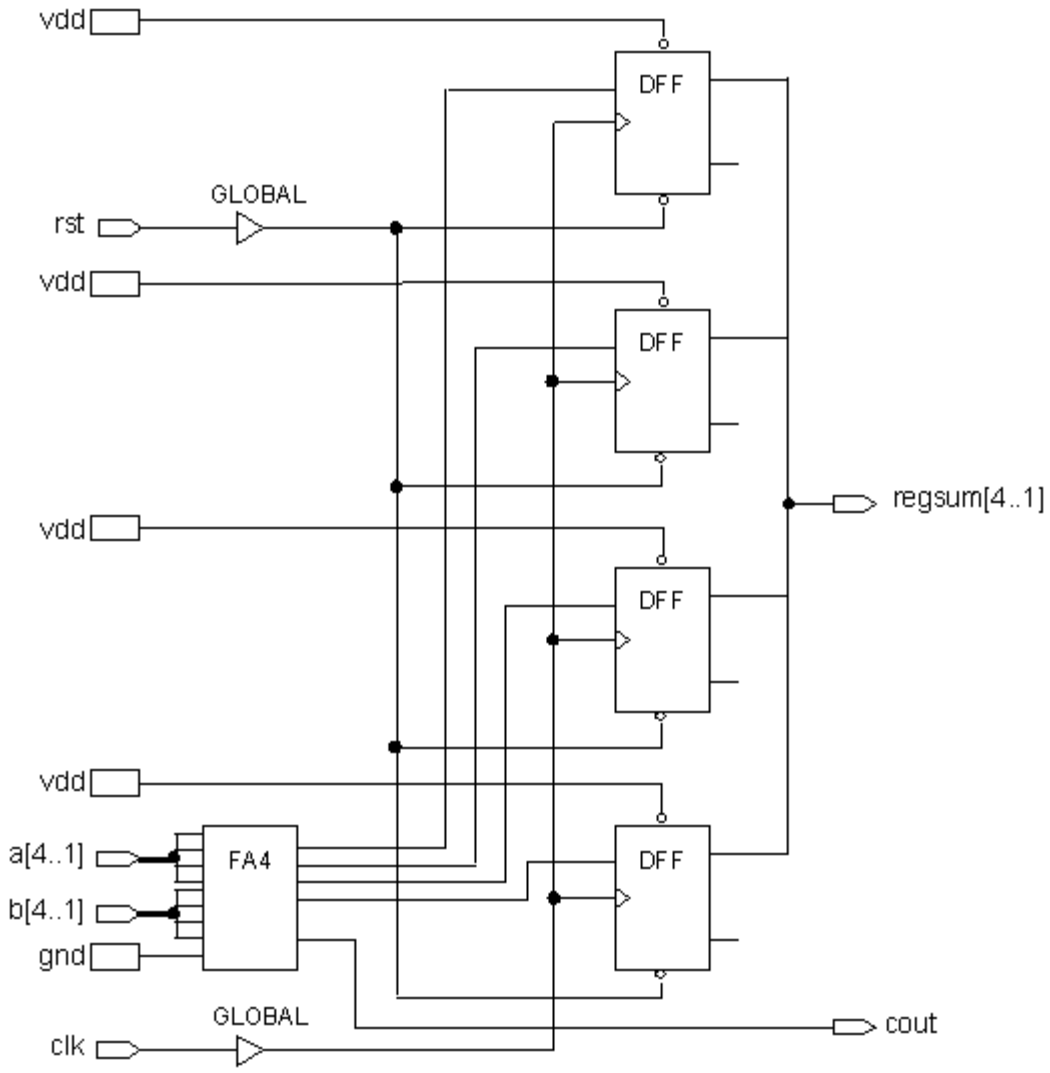
Figure 2. Hollow-Body Description of a 4-Bit Full Adder (7483)

```
module fa4 ( c0, a1, b1, a2, b2, a3, b3, a4, b4, s1, s2, s3, s4, c4);
   output    s1, s2, s3, s4, c4;
   input     c0, a1, b1, a2, b2, a3, b3, a4, b4;
   endmodule
```

If the hollow-body description is in a separate file, you must analyze it before analyzing the higher-level function with the HDL Compiler for Verilog to produce a hollow-body component. This component contains a single level of hierarchy with input and output pins, but does not contain any underlying logic.

You can save the synthesized design as an EDIF netlist file (**.edf**) and compile it with the MAX+PLUS II software. After the HDL Compiler for Verilog software has successfully processed the design, it generates the schematic shown in Figure 3, which you can view with the Design Analyzer software.

Figure 3. Synthesized Design Generated by the Design Compiler



However, before you compile the EDIF netlist file with the MAX+PLUS II software, you must create the **adder.lmf** file, shown in Figure 3, to map the `fa4` function to the equivalent MAX+PLUS II function (`7483`). You must then specify the LMF as *LMF #2* in the expanded **EDIF Netlist Reader Settings** dialog box (Interfaces menu) (*LMF #1* is **altsyn.lmf**). For more information about creating LMFs, refer to "Library Mapping Files (**.lmf**)" and "Library Mapping File Format" in MAX+PLUS II Help.

Figure 3. Library Mapping File Excerpt for fa4

```
BEGIN
FUNCTION 7483  (c0, a1, b1, a2, b2, a3, b3, a4, b4,)
RETURNS        (s1, s2, s3, s4, c4)



FUNCTION "fa4" ("c0", "a1", "b1", "a2", "b2", "a3",
               "b3","a4", "b4")
RETURNS        ("s1", "s2", "s3", "s4", "c4")
END
```

When you compile the design with the MAX+PLUS II software, you can disregard the warning "`EDIF cell <name> already has LMF mapping so CONTENTS construct has been ignored`". To verify the global Clock and global Reset usage, as well as the number of logic cells used, see the **adder.rpt** Report File generated by the MAX+PLUS II Compiler.

MAX+PLUS II Architecture Control Logic Function Instantiation Example for Verilog HDL

You can instantiate Altera® -provided logic functions from the **alt_mf** library, which includes the `a_8fadd`, `a_8mcomp`, `a_8count`, and `a_81mux` functions, in Verilog HDL designs. Altera provides behavioral Verilog HDL descriptions of these functions.

Figure 1 shows an example of an 8-bit counter that is instantiated using the `a_8count` function. Because Verilog HDL is case-sensitive, be sure to use uppercase letters for all of the macrofunction's module names and port names.

Figure 1. Sample Verilog HDL File with Logic Function Instantiation (counter.v)

```
module counter (clock, ena, load, dnup, set, clear, i, q, cout);
output          cout;
output[7:0]     q;
input[7:0]      i;
input           clock, ena, load, dnup, set, clear;
A_8COUNT u1    (.A(i[0]), .B(i[1]), .C(i[2]), .D(i[3]),
                .E(i[4]), .F(i[5]), .G(i[6]), .H(i[7]),
                .LDN(load), .GN(ena), .DNUP(dnup), .SETN(set),
                .CLRN(clear), .CLK(clock), .QA(q[0]), .QB(q[1]),
                .QC(q[2]), .QD(q[3]), .QE(q[4]), .QF(q[5]),
                .QG(q[6]), .QH(q[7]), .COUT(cout) );
endmodule
```

The sample file shown in Figure 1 can be synthesized with the Design Compiler or FPGA Compiler. You can also simulate it with the Cadence Verilog-XL Simulator by typing the following command at the **dc_shell** prompt:

```
verilog counter.v /usr/maxplus2/synopsys/library/alt_mf/src/mf.v↵
```

Instantiating RAM & ROM Functions in Verilog HDL

The MAX+PLUS® II /Synopsys interface offers full support for the memory capabilities of the FLEX® 10K device family, including synchronous and asynchronous RAM and ROM, cycle-shared dual port RAM, dual-port RAM, single-Clock FIFO, and dual-clock FIFO functions. You can use the Altera® -provided **genmem** utility to generate functional simulation models and timing models for these functions. Type `genmem ↵` at the

UNIX prompt to display information on how to use this utility, as well as a list of the functions you can generate.

To instantiate a RAM or ROM function in Verilog HDL, follow these steps:

1. Use the **genmem** utility to generate a memory model by typing the following command at the UNIX prompt:

   `genmem` <*memory type*> <*memory size*> `-verilog` ↵

   For example: `genmem asynrom 256x15 -verilog` ↵

2. Create a Verilog HDL design that instantiates the <*memory name*> function.

   Figure 1 shows a Verilog HDL design that instantiates **asyn_rom_256x15.v**, a 256 x 15 ROM function.

   Figure 1. Verilog HDL File with ROM Instantiation (tstrom.v)

```
module tstrom (addr, enab, q);
parameter LPM_FILE = "u1.hex"
input [7:0] addr;
input enab;
output [14:0] q;



asyn_rom_256x15
// synopsys translate_off
           #(LPM_FILE)



// synopsys translate_on
      u1  (.Address(addr), .Q(q), .MemEnab(enab));

endmodule
```

3. (Optional for RAM functions) Specify an initial memory content file:

   - For ROM functions, you must specify the filename of an initial memory content file in the Intel hexadecimal format (**.hex**) or the Altera® Memory Initialization File (**.mif**) format in the Parameter Statement, with the `LPM_FILE` parameter. See Figure 1. The filename must be the same as the instance name; e.g., the `u1` instance name must be unique throughout the whole project. The initialization file must reside in the directory containing the project's design files.

   - For RAM functions, specifying a memory initialization file is optional. If you want to use it, you must specify it in a Parameter Statement, as described above.

   1. The MIF format is supported only for specifying initial memory content when compiling designs within MAX+PLUS II software. You cannot use a MIF to perform simulation with Synopsys tools prior to MAX+PLUS II compilation.

   2. If you use an Intel hexadecimal format file and wish to simulate the design with the VHDL System Simulator (VSS) after MAX+PLUS II compilation, you should use the Synopsys **intelhex** utility to translate the Intel hexadecimal fomat file into a VSS-compatible Synopsys memory file. Refer to the Synopsys ***VHDL System Simulator Software Tool*** manual for details about using the **intelhex** utility.

4. In the Verilog HDL design, add `// synopsys translate_off` ↵ before the Parameter Statement, and add `// synopsys translate_on` ↵ after the Parameter Statement. These directives tell the HDL

Compiler for Verilog when to stop and start synthesizing. See Figure 1.

5. The timing model (**.lib**) generated by the **genmem** utility contains pin-to-pin delay information that can be used by the Synopsys Design Compiler and FPGA Compiler software. You must add this timing model to the existing library so that the compiler can access the timing information. Type the following commands at the **dc_shell** prompt:

   ```
   read -f db flex10k[<speed grade>].db ↵
   update_lib flex10k[<speed grade>] <RAM/ROM function name>.lib ↵
   ```

6. (Optional) Include the following command to update your **flex10k[***<speed grade>***].db** file with the RAM/ROM timing information:

   ```
   write_lib flex10k[<speed grade>] -o flex10k.db ↵
   ```

7. When you generate the EDIF netlist file from the design, include the bus structure from the RAM or ROM function(s). Go to Setting Up Synopsys Configuration Files for more information.

8. Continue with the steps necessary to complete your Verilog HDL design, as described in Creating Verilog HDL Designs for Use with MAX+PLUS II Software.

Related Topics:

- Go to FLEX 10K Device Family, which is available on the web, for additional information.

Synthesizing & Optimizing VHDL & Verilog HDL Projects with Synopsys Software

The MAX+PLUS® II Compiler can process a VHDL or Verilog HDL file that has been synthesized by the Synopsys Design Compiler or FPGA Compiler software, saved as an EDIF 2 0 0 or 3 0 0 netlist file, and imported into the MAX+PLUS II software. The procedure below explains how to run Synopsys tools separately from MAX+PLUS II Software.

You can also run Synopsys tools from within the MAX+PLUS II software to automatically generate and import an EDIF file. Refer to Running Synopsys Compilers from MAX+PLUS II Software for more information. In addition, if your MAX+PLUS II development system includes VHDL or Verilog HDL synthesis support, the MAX+PLUS II Compiler can directly synthesize VHDL or Verilog HDL logic. For more information, go to MAX+PLUS II VHDL or Verilog HDL Help.

The following steps explain how to synthesize and optimize a VHDL or Verilog HDL design for use with MAX+PLUS II software:

4. Be sure to set up your design environment correctly. This step includes specifying the target device family for the design. See the following topics:
   - Setting Up the Synopsys/MAX+PLUS II Working Environment
   - Setting Up the Design Compiler and FPGA Compiler Configuration Files
   - Setting Up the DesignWare Interface
   - Setting Up the VSS Configuration Files

5. Create a VHDL file, <*design name*>**.vhd**, or a Verilog HDL design, <*design name*>**.v**, using the MAX+PLUS II Text Editor or another standard text editor and save it in a project directory under your login directory. See the following topics for instructions:

   - Creating VHDL Designs for Use with MAX+PLUS II Software.
   - Creating Verilog HDL Designs for Use with MAX+PLUS II Software.

6. Start the Design Compiler or FPGA Compiler software by typing either `dc_shell` ↵ or `fpga_shell` ↵ at the command line, respectively. To work within the graphical user interface, type `design_analyzer` ↵ for either tool.

7. Analyze and then compile the design with the Design Compiler, FPGA Compiler, or Design Analyzer software. The VHDL Compiler or HDL Compiler for Verilog software automatically translates the design into Synopsys database (**.db**) format. Specific steps are necessary for some types of projects before you process the design:

    1. If your FLEX 10K design includes RAM or ROM functions, follow these steps:

        1. (VHDL designs only) Because the VHDL Compiler software does not support the data type `string` for the Generic Clause, you must also enter the following command at the `dc_shell` prompt before you read the design:

            ```
            hdlin_translate_off_skip_text=true ↵
            ```

        2. The timing model (**.lib**) generated by the **genmem** utility contains pin-to-pin delay information that can be used by the Synopsys Design Compiler and FPGA Compiler software. You must add this timing model to the existing library so that the compiler can access the timing information. Type the following commands at the **dc_shell** prompt:

            ```
            read -f db flex10k[<speed grade>].db ↵
            update_lib flex10k[<speed grade>] <RAM/ROM function name>.lib ↵
            ```

        3. (Optional) Enter the following command to update your **flex10k[***<speed grade>***].db** file with the RAM/ROM timing information:

            ```
            write_lib flex10k[<speed grade>] -o flex10k.db ↵
            ```

        See Instantiating RAM & ROM Functions in VHDL or Instantiating RAM & ROM functions in Verilog HDL for additional information.

    2. If you wish to allow the FPGA Compiler to perform *N*-input look-up table (LUT) optimization for a FLEX 6000, FLEX 8000, or FLEX 10K design, enter the following command at the `dc_shell` prompt before compiling the design:

        ```
        edifout_write_properties_list = "lut function" ↵
        ```

        Go to Using FPGA Compiler *N*-Input LUT Optimization for FLEX 6000, FLEX 8000, or FLEX 10K Devices for more information.

    3. If you wish to enter resource assignments, go to Entering Resource Assignments.

    4. If you wish to direct the Design Compiler or FPGA Compiler to use sum-of-products logic in processing a MAX 7000 or MAX 9000 design, type the following commands at the `dc_shell` prompt before compiling the design:

        ```
        set_structure false ↵
        set_flatten -effort low ↵
        ```

        See MAX 7000 & MAX 9000 Synthesis Example for more information.

        For additional information on how the Design Compiler and FPGA Compiler synthesize and optimize a design, see the following topics:

        - ***Synopsys Design Compiler Reference Manual*** or ***Design Analyzer Reference Manual***

8. (Optional) View the optimized project with the Design Analyzer. The Design Analyzer uses the **altera.sdb** library to display optimized projects generated by the Design Compiler or FPGA Compiler.

9. (Optional) To view Synopsys-generated timing information and generate a file detailing primitive usage, type the following commands:

```
report_timing ↵
report_reference > <filename> ↵
```

10. (Optional) To functionally verify the project prior to processing with the MAX+PLUS II software, save the design as a VHDL netlist file, and simulate it as described in Performing a Pre-Routing or Functional Simulation with VSS Software.

11. Save the optimized project as an EDIF netlist file with the extension **.edf**.

12. Process the <*design name*>**.edf** file with the MAX+PLUS II Compiler, as described in Compiling Projects with the MAX+PLUS II Software.

Installing the Altera-provided MAX+PLUS II/Synopsys interface on your computer automatically creates the following sample VHDL and Verilog HDL files:

- **/usr/maxplus2/synopsys/examples/ministate.vhd**
- **/usr/maxplus2/synopsys/examples/ministate.v**

Related Topics:

- Go to the following MAX+PLUS II ACCESS^{SM} Key topics for related information:
  - Resynthesizing a Design Using the **alt_vtl** Library and a MAX+PLUS II SDF Output File
  - Programming Altera Devices

MAX 7000 & MAX 9000 Synthesis Example

The MAX® 7000 (including MAX 7000E, MAX 7000S, and MAX 7000A) and MAX 9000 device families have a sum-of-products architecture. To obtain optimum timing and area results, you can direct the Synopsys Design Compiler or FPGA Compiler software to synthesize your logic into a sum-of-products form. To assist the Synopsys compilers in meeting the timing and area constraints of your designs, the Altera® technology libraries provide models that approximate the timing of the MAX 7000 and MAX 9000 logic cells.

Figure 1 shows two timing models: the standard Altera MAX 7000 timing model and a Synopsys timing model that approximates the MAX 7000 model. The Synopsys model is built on the following three conditions and assumptions:

1. The combinatorial delay in logic cells has been equally divided between product terms and OR gates. Because the product-term delay equals the OR-gate delay, the Synopsys compilers treat them equally, producing a sum-of-products structure. On top of this structure, inverters are used where necessary.

2. A shared expander product term is always used to create combinatorial logic.

3. The Synopsys Design Compiler and FPGA Compiler software do not distinguish between array and global Clocks. Therefore, to estimate setup and hold timing most accurately, you must instantiate GLOBAL buffers to indicate a global clock in either your VHDL or Verilog HDL design.

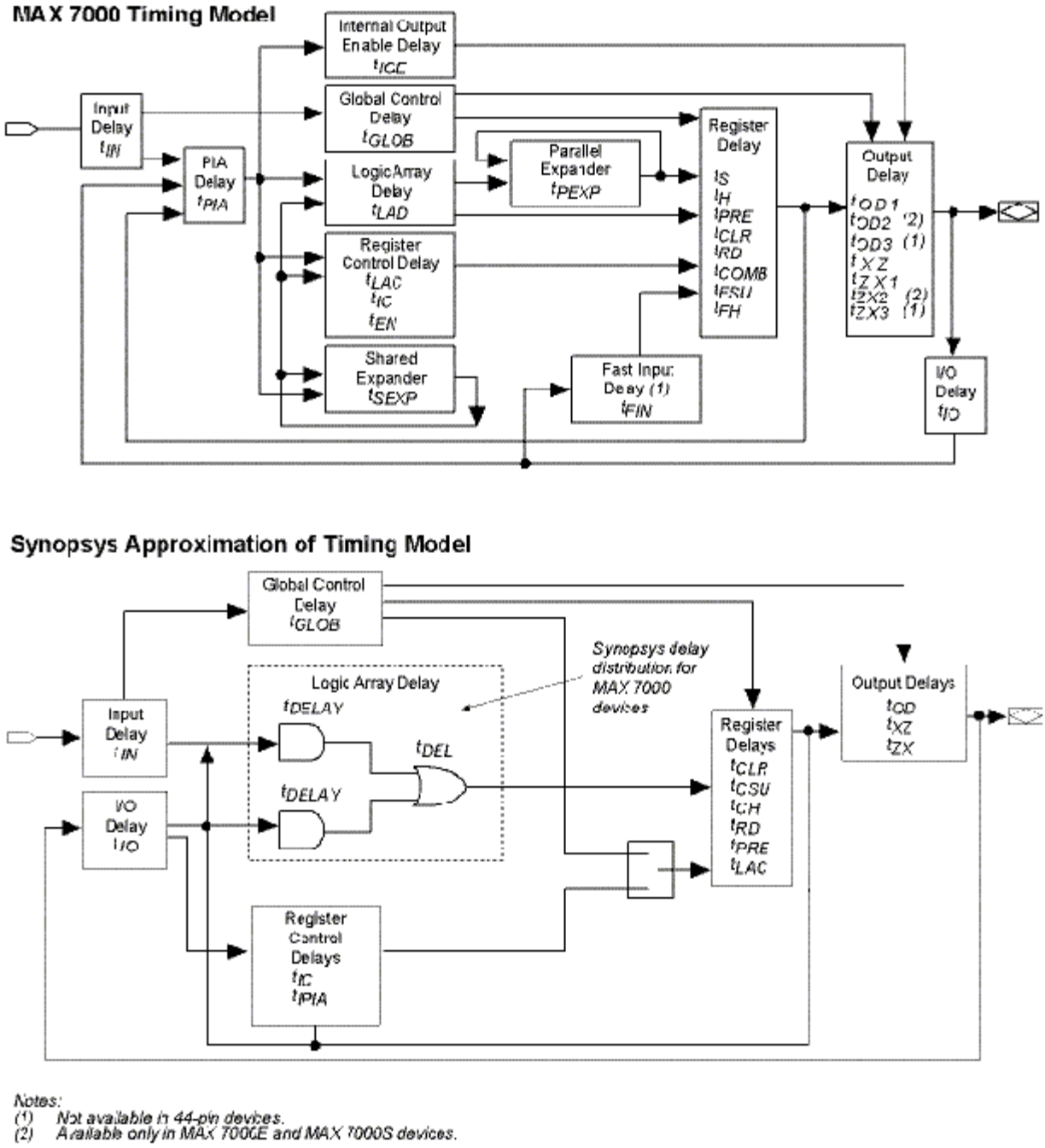


Figure 1. Standard MAX 7000 Timing Model vs. Synopsys Approximation of Timing Model

If you wish to direct the Synopsys Design Compiler or FPGA Compiler software to produce sum-of-products logic that approximates the MAX 7000 or MAX 9000 timing model, you can type the following **dc_shell** prompt commands at the command line before compiling the design:

```
set_structure false ↵

set_flatten -effort low ↵
```

When `set_structure` is set to `false`, structuring is turned off, and the Synopsys Design Compiler and FPGA Compiler software cannot factor and share logic between functions. If you do not enter these commands, the Synopsys compilers may add logic, which can create additional area and timing delays.

Figure 2 shows a combinatorial design that is predictable when structuring is turned off, but is unpredictable when structuring is turned on.

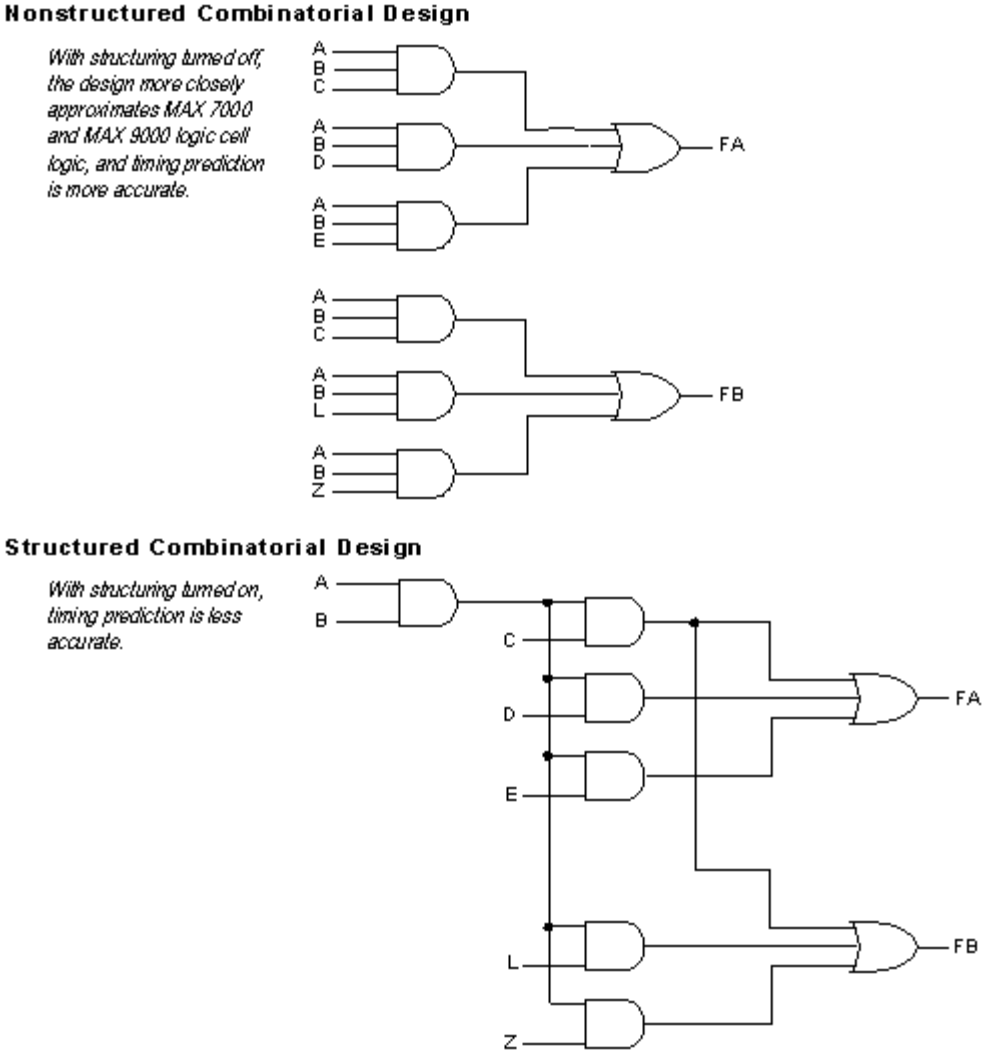


Figure 2. Nonstructured vs. Structured Combinatorial Design

When you use `low` as the argument to the `set_flatten -effort` command, the Synopsys compilers use the shortest compilation time to create the sum-of-products implementation of your design. If you use the `medium` or `high` argument, the Synopsys compilers create optimally flattened designs, but usually require greater compilation time and offer little improvement in timing and area results.

You can type `report_timing` ↵ after compilation to view Synopsys-generated timing information.

If you wish to calculate the area of your design, you can obtain an approximate logic cell count in several ways. Altera recommends that you add the number of registers and combinatorial outputs in a design. Depending on your design, this number may be slightly lower than the final number reported by the MAX+PLUS II software.

To create a file detailing primitive usage in the design, type `report_reference>` <*filename*> ↵ after Synopsys compilation.

To obtain accurate timing information about your design, you must use the MAX+PLUS II Timing Analyzer to analyze your design. For accurate area information, consult the Report File (**.rpt**) generated by the MAX+PLUS II software.

Related Topics:

- Refer to the following sources for related information:

- ***Synopsys Design Compiler Reference Manual*** or ***Synopsys Command Reference Manual***
- ***FPGA Compiler User Guide***
- Synthesizing & Optimizing VHDL & Verilog HDL Pojects with Synopsys Software
- Go to MAX Devices, which is available on the web, for additional information:

DesignWare FLEX 8000 Synthesis Example

Figure 1 shows a sample VHDL design, design_one.vhd, which illustrates component inference with the DesignWare interface for FLEX® 8000 devices.

Figure 1. VHDL Design File (design_one.vhd)

This design illustrates the sum of A + B.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;



ENTITY design_one IS
   PORT (a,b : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
         f   : OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
END design_one;



ARCHITECTURE add_design OF design_one IS



BEGIN
 f <= a + b;
END add_design;
```

When the VHDL Compiler or the HDL Compiler for Verilog software analyzes and elaborates the design, it replaces the "+" operator with its synthetic operator equivalent.

Figure 2 shows the design as it appears in the Design Analyzer software after it has been analyzed and elaborated by the VHDL Compiler software.

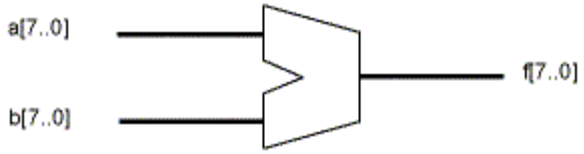


Figure 2. design_one.vhd after Analysis & Elaboration

When you synthesize a design, the Design Compiler or FPGA Compiler software uses the synthetic library to match the synthetic operator to the FLEX-optimized logical implementation in the technology library. The Synopsys Design Compiler or FPGA Compiler software then instantiates and interconnects the correct number of `flex_add` and `flex_carry` functions to produce the 8-bit adder shown in Figure 1. When you save a compiled design as a VHDL, Verilog HDL or EDIF file, the file preserves the number of `flex_add` and `flex_carry`

functions, as well as their interconnections. Consequently, area and performance predictions that you make in the Synopsys design environment closely match the final MAX+PLUS® II result.

Table 2 lists functions included in the DesignWare FLEX 6000, FLEX 8000, and FLEX 10K synthetic libraries.

Table 2. FLEX 6000, FLEX 8000, and FLEX 10K Synthetic Library Functions

Name	Function
`flex_add`	Sum of A, B, and Carry-In
`flex_carry`	Carry of A, B, and Carry-In
`flex_sub`	Difference of A, B, and Borrow-In
`flex_borrow`	Borrow of A, B, and Borrow-In
`flex_gt, flex_sgt`	Greater than (`flex_gt` is unsigned; `flex_sgt` is signed)
`flex_carry_gt`	Greater than Carry
`flex_lt, flex_slt`	Less than (`flex_lt` is unsigned; `flex_slt` is signed)
`flex_carry_lt`	Less than Carry
`flex_gteq, flex_sgteq`	Greater than or equal to (`flex_gteq` is unsigned; `flex_sgteq` is signed)
`flex_carry_gteq`	Greater than or equal to Carry
`flex_inc`	Incrementer (Count = Count + 1)
`flex_carry_inc`	Incrementer Carry (Count = Count + 1)
`flex_dec`	Decrementer (Count = Count - 1)
`flex_carry_dec`	Decrementer Carry (Count = Count - 1)
`flex_lteq, flex_slteq`	Less than or equal to (`flex_lteq` is unsigned; `flex_slteq` is signed)
`flex_carry_lteq`	Less than or equal to Carry
`flex_count`	Counter
`aflex_carry_count`	Counter Carry
`flex_add_sub`	Adder/Subtractor
`flex_inc_dec`	Incrementer/Decrementer
`flex_umult, flex_smult`	Multiplier (`flex_umult` is unsigned; `flex_smult` is signed)

Figure 3 shows **design_one.vhd** after it has been synthesized with the Design Compiler.

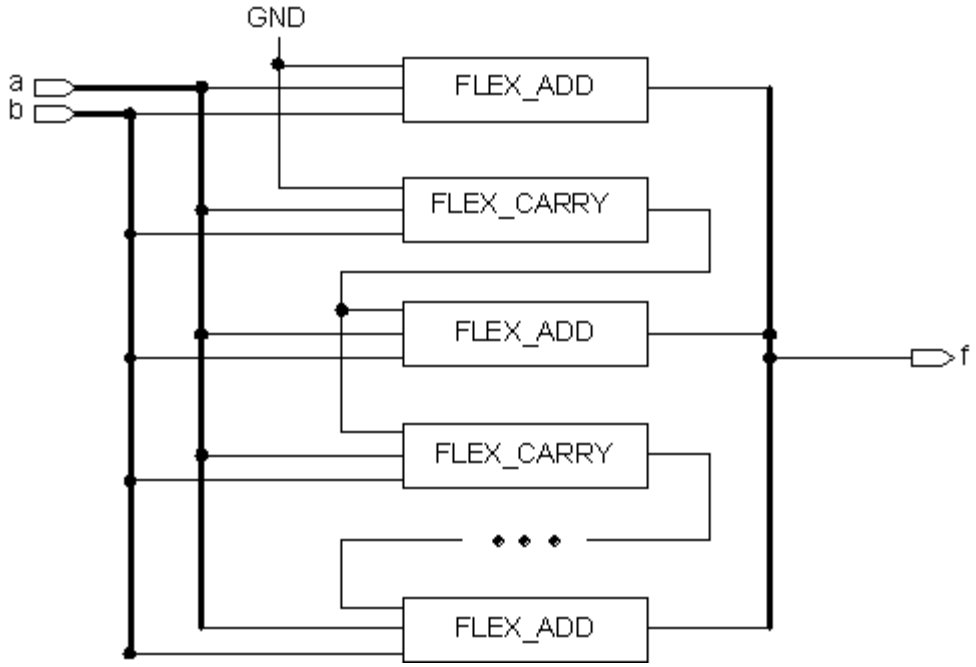


Figure 3. design_one.vhd Synthesized & Resolved for FLEX 6000, FLEX 8000 & FLEX 10K Architecture

After you save the design as an EDIF Input File (**.edf**) and process it with the MAX+PLUS II Compiler, the Compiler replaces instances of `flex_add` and `flex_carry` with FLEX-optimized versions, as shown in Figure 4. The MAX+PLUS II Compiler maps these functions into a single logic element (LE). The result is a high-speed 8-bit adder that fits into 8 LEs.

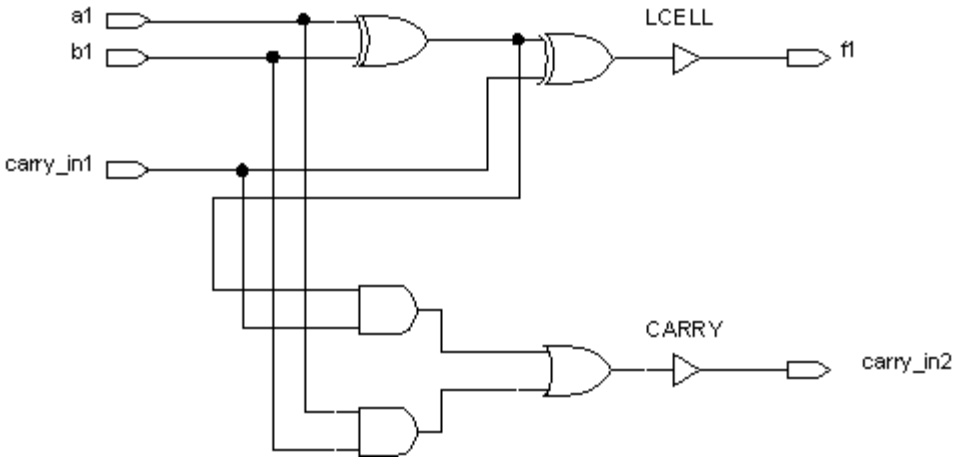


Figure 4. One Slice of the design_one 8-bit Adder Design with Optimized FLEX 8000 Functions

Related Topics:

- Refer to the following sources for related information on DesignWare and the Synopsys VHDL Compiler:
  - ***Synopsys DesignWare Databook***
  - ***VHDL Compiler Reference Manual***
- Go to FLEX Devices, which is available on the web, for additional information

---

Using FPGA Compiler N-Input LUT Optimization for FLEX 6000, FLEX 8000 & FLEX 10K Devices

The Synopsys FPGA Compiler software supports an *N*-input look-up table (LUT) function that improves the quality of the results and the predictability of delay and resource estimates. All Altera® FPGA Compiler libraries for FLEX® 6000, FLEX 8000, and FLEX 10K devices support the *N*-input LUT function.

Figure 1 shows a sample command sequence that FPGA Compiler might require for *N*-input LUT optimization. To use *N*-input LUT optimization, include the `edifout_write_properties_list = "lut_function"` command.

Figure 1. Sample Command Sequence for N-Input LUT Optimization

```
read -f vhdl <design name>.vhd ↵
current_design = <design name> ↵
set_max_area 0 ↵
uniquify ↵
ungroup -all -flatten ↵
compile -ungroup_all ↵
report_area > <design name>.rpa ↵
report_fpga > <design name>.rpf ↵
report_cell > <design name>.rpc ↵
edifout_write_properties_list = "lut_function" ↵
write -f edif -hierarchy -o <design name>.edf ↵
```

Use the area report to determine the circuit area.

If you wish to maintain area report estimates as closely as possible during MAX+PLUS® II processing, Altera recommends that you select the *WYSIWYG* setting for the *Global Project Synthesis Style* in the **Global Project Logic Synthesis** dialog box (Assign menu). However, selecting the *Normal* or *Fast* style may yield a better result.

Related Topics:

- For more information on how to use the FPGA Compiler software optimize your design for FLEX 8000 devices, refer to *Chapter 5: Optimization for the Altera FLEX 8000 Architecture* in the ***Synopsys FPGA Compiler User Guide***.
- Go to FLEX Devices, which is available on the web, for additional information:

Entering Resource Assignments

The MAX+PLUS® II software allows you to enter a variety of resource and device assignments for your projects. Resource assignments are used to assign logic functions to a particular pin, logic cell, I/O cell, embedded cell, row, column, Logic Array Block (LAB), Embedded Array Block (EAB), chip, clique, local routing, logic option, timing requirement, or connected pin group. In the MAX+PLUS II software, you can enter all types of resource and device assignments with Assign menu commands. You can also enter pin, logic cell, I/O cell, embedded cell, LAB, EAB, row, and column assignments in the MAX+PLUS II Floorplan Editor. The Assign menu commands and the Floorplan Editor all save assignment information in the ASCII Assignment & Configuration File (**.acf**) for the project.

In designs targeted for the Synopsys Design Compiler and FPGA Compiler software, you can assign a limited subset of these resource assignments by setting attributes in the VHDL or Verilog HDL design files with the `set_attribute` command. These attributes are incorporated into the EDIF netlist file(s). The

MAX+PLUS II software automatically converts assignment information from the EDIF Input File (**.edf**) into the ACF format. For information on making MAX+PLUS II-compatible resource assignments with the `set_attribute` command, go to the following topics:

- Assigning Pins, Logic Cells, & Chips
- Assigning Cliques
- Assigning Logic Options

You can also modify the ACF for a design to contain timing requirements and other assignments, as described in the following topics:

- Modifying the Assignment & Configuration File with the **setacf** utility
- Converting Synopsys Timing Constraints into MAX+PLUS II-Compatible Format with the **syn2acf** Utility
- Converting Synopsys Hierarchical Timing Constraints into MAX+PLUS II-Compatible Format with the **gen_iacf** and **gen_hacf** Utilities

Related Topics:

- Refer to the following sources for related information:
  - Synopsys documentation for additional information on how to assign properties
  - "Resource Assignments in EDIF Input Files" and "Assigning Resources in a Third-Party Design Editor" in MAX+PLUS II Help for more information on assignments or properties that can be assigned in Synopsys
  - "resource assignments" or "ACF, format" in MAX+PLUS II Help using **Search for Help on** (Help menu), for information on entering assignments in the MAX+PLUS II software with Assign menu commands or in an ACF

Assigning Pins, Logic Cells & Chips

You can assign a single logic function to a specific pin or logic cell (including I/O cells and embedded cells) within a chip, and assign one or more functions to a specific chip. A chip is a group of logic functions defined as a single, named unit, which can be assigned to a specific device.

You can assign a signal to a particular pin to ensure that the signal is always associated with that pin, regardless of future changes to the project. If you wish to set and maintain the performance of your project, assigning logic to a specific logic cell within a chip can minimize timing delays. In a project that is partitioned among multiple devices, you can assign logic functions that must be kept together in the same device to a chip. Chip assignments allow you to split a project so that only a minimum number of signals travel between devices, and to ensure that no unnecessary device-to-device delays exist on critical timing paths. You can assign a chip to a device in MAX+PLUS® II software.

To make pin, logic cell, and chip assignments, use the `set_attribute` command at a **dc_shell** prompt. Before using the `set_attribute` command, add the following line to your **.synopsys_dc.setup** file:

```
edifout_write_properties_list = {LOGIC_OPTION, CLIQUE, CHIP_PIN_LC} ↵
```

Table 1 shows the syntax to use for chip, pin, and logic cell assignments:

Table 1. Commands for Chip, Pin, & Logic Cell Assignments

Assignment Type	Command to Type *Note (1)*
Chip	`set_attribute find (<design object>, (<instance name>)) "CHIP_PIN_LC" -type ↵`

	`string "<chip name>"`
Pin	`set_attribute find (<design object>, (<instance name>)) "CHIP_PIN_LC" -type string "<chip name>@<pin number>"` ↵
Logic cell number	`set_attribute find (<design object>, (<instance name>)) "CHIP_PIN_LC" -type string "<chip name>@LC<logic cell number>"` ↵
I/O cell number	`set_attribute find (<design object>, (<instance name>)) "CHIP_PIN_LC" -type string "<chip name>@IOC<I/O cell number>"` ↵
Embedded cell number	`set_attribute find (<design object>, (<instance name>)) "CHIP_PIN_LC" -type string "<chip name>@EC<embedded cell number>"` ↵

Note:

1. In this table, *<design object>* represents ports, references, cells, nets, or pins.

Examples:

```
set_attribute find (cell, (U1)) "CHIP_PIN_LC" -type string "chip1" ↵

set_attribute find (cell, (U1)) "CHIP_PIN_LC" -type string "chip1@K2" ↵

set_attribute find (cell, (U1)) "CHIP_PIN_LC" -type string "chip1@LC44" ↵
```

Related Topics:

- Go to "Devices & Adapters" and "Assigning a Device" in MAX+PLUS II Help for information on device pin-outs and assigning devices, respectively, in the MAX+PLUS II software.

Assigning Cliques

You can define a group of logic functions as a single, named unit, called a clique. The MAX+PLUS® II Compiler attempts to place all logic in the clique in the same logic array block (LAB) to ensure optimum speed. If the project does not use multi-LAB devices, or if it is not possible to fit all clique members into a single LAB, the clique assignment ensures that all members of a clique are placed in the same device. In FLEX® 6000, FLEX 8000, MAX® 9000, and FLEX 10K devices, the Compiler also attempts to place the logic in LABs in the same row. Cliques therefore allow you to partition a project so that only a minimum number of signals travel between LABs, and to ensure that no unnecessary LAB-to-LAB or device-to-device delays exist on critical timing paths.

To make pin, logic cell, and chip assignments, use the `set_attribute` command at a **dc_shell** prompt. Before using the `set_attribute` command, add the following line to your **.synopsys_dc.setup** file:

```
edifout_write_properties_list = {LOGIC_OPTION, CLIQUE, CHIP_PIN_LC} ↵
```

✔ To assign a clique, type the following command at a **dc_shell** prompt:

`set_attribute find(<design object>,(<instance name>))"CLIQUE" -type string "<clique name>"` ↵

For example:

```
set_attribute find (cell, (U1)) "CLIQUE" -type string "fast1" ↵
```

Related Topics:

- Go to the following topics in MAX+PLUS II Help for related information:
  - Assigning a Clique
  - Guidelines for Achieving Maximum Speed Performance

Assigning Logic Options

Logic options and logic synthesis style assignments allow you to guide logic synthesis with logic optimization features that are specific to Altera® devices. You can assign logic options and styles to individual logic functions in your design. The MAX+PLUS® II Compiler also uses a device family-specific default logic synthesis style for each project.

To make pin, logic cell, and chip assignments, use the `set_attribute` command at a **dc_shell** prompt. Before using the `set_attribute` command, add the following line to your **.synopsys_dc.setup** file:

```
edifout_write_properties_list = {LOGIC_OPTION, CLIQUE, CHIP_PIN_LC} ↵
```

✔ To assign a logic option or a logic synthesis style, type the following command at a **dc_shell** prompt:

```
set_attribute find(<design object>, (<instance name>)) "LOGIC_OPTION"
-type string "<logic option>=<value>" ↵
```

For example:

```
set_attribute find (cell, (U1)) "LOGIC_OPTION" -type string
"STYLE=FAST" ↵
```

To specify multiple logic options, use commas as separators.

For example:

```
set_attribute find (cell, (U1))"LOGIC_OPTION" -type string "STYLE=FAST,
CARRY_CHAIN=MANUAL" ↵
```

Related Topics:

- Go to "Resource Assignments in EDIF Input Files" and "Assigning Resources in a Third-Party Design Editor" in MAX+PLUS II Help for complete and up-to-date information on logic option and logic synthesis style assignments, including definitions and syntax of these assignments.

Modifying the Assignment & Configuration File with the setacf Utility

Altera provides the **setacf** utility to help you modify a project's Assignment & Configuration File (**.acf**) from the command line, without opening the file with a text editor. Type `setacf -h ↵` at a UNIX or DOS prompt to get help on this utility.

Converting Synopsys Timing Constraints into MAX+PLUS II-Compatible Format with the syn2acf Utility

Altera provides the **syn2acf** utility, which is an interface program that converts Synopsys timing constraints from non-hierarchical designs into the MAX+PLUS® II Assignment & Configuration File (**.acf**) format. For

information on converting timing constraints from hierarchical designs, refer to Converting Synopsys Hierarchical Timing Constraints into MAX+PLUS II-Compatible Format with the **gen_iacf** and **gen_hacf** Utilities.

The **syn2acf** utility requires the following input files:

- Flattened EDIF netlist file
- **dc_shell** script file
- Standard Delay Format (SDF) constraints construct
- SDF timing delay construct

To use the **syn2acf** utility, follow these steps:

1. Set the timing constraints by using one of the following methods:

   ✔ Start the Synopsys Design Analyzer and specify timing constraints by choosing appropriate menu commands.

   or:

   ✔ Create the <*design name*>**.cmd** file for use with a **dc_shell** script. See Figure 1.

The **syn2acf** utility does not support `set_arrival` timing constraints for internal nodes.

Figure 1. Sample Command File (.cmd) for Setting Timing Constraints

```
create_clock -period 50 -waveform {0 25} CLK
set_clock_skew -delay 2 CLK
set_input_delay 10 IN2
set_input_delay 5 -clock CLK IN1
set_output_delay 20 OUT2
set_output_delay 5 -clock CLK OUT1
set_max_delay 25 -to OUT1
set_max_delay 35 -to OUT2
set_multicycle_path 2 -to n20_reg
```

1. Compile the design and run the **syn2acf** utility either from the command line or with a Design Compiler **dc** script:

   ✔ Compile the design, then type the following command from the UNIX prompt to start the **syn2acf** utility:

   `/usr/maxplus2/synopsys/bin/syn2acf` <*design name*> ↵

   or:

   ✔ Run a **dc** script inside the **dc_shell** script that reads the VHDL design, compiles it, and runs the **syn2acf** utility. Figure 2 shows a sample **dc** script.

   The **syn2acf** utility uses the `ALT_HOME` environment variable, if it has been specified, to determine the MAX+PLUS II system directory; otherwise, it uses the **/usr/maxplus2** directory. To specify a different MAX+PLUS II system directory with the `ALT_HOME` environment variable, you can either edit the

.cshrc file to specify the correct directory or type the following command at the UNIX prompt:

```
setenv ALT_HOME <MAX+PLUS II system directory> ↵
```

Figure 2. Sample Script for Running the syn2acf Utility

```
/* dc_script example to interface with syn2acf */
dc_shell <<!
read -f vhdl <design name>.vhd
include <design name>.cmd                          /*set timing constraints*/
compile
current_design=<design name>
include /usr/maxplus2/synopsys/bin/syn2acf.cmd     /*generate required files*/
sh /usr/maxplus2/synopsys/bin/syn2acf <design name> /*invoke syn2acf utility*/
quit
!
```

The **syn2acf** utility cannot support maximum Clock frequency ($\mathbf{f_{MAX}}$) correctly if more than one Clock skew is specified in the **dc_shell** command script. This problem occurs because the Synopsys `write_script` command drops the Clock skew information for the registers. The **syn2acf** utility will use the last Clock skew number to calculate $\mathbf{f_{MAX}}$.

The sample **dc** script includes the Altera® -provided **syn2acf.cmd** file, shown in Figure 3, to generate the required input files for the **syn2acf** utility.

Figure 3. Altera-Provided syn2acf.cmd File

```
ungroup -flatten -all
write -f edif
write_script > altsyn.dc
write_constraints -format sdf -cover_design
write_timing -format sdf
```

All timing assignments generated by the **syn2acf** utility are written to the Timing Requirement Assignments Section of the project's ACF, with the assignment source identifier `{synopsys}` at the end of each line. Figure 4 shows a sample ACF excerpt that contains Synopsys timing constraints generated by the **syn2acf** utility.

Figure 4. Sample ACF Excerpt with Synopsys Timing Constraints

```
TIMING_POINT
BEGIN
   "|OUT2"    : TCO = 15.00ns {synopsys};
   "|IN1"     : TPD = 10.00ns {synopsys};
   "|IN2"     : TPD = 5.00ns {synopsys};
   "|OUT1"    : TCO = 20.00ns {synopsys};
   "|IN1"     : TSU = 20.00ns {synopsys};
   "|IN2"     : TSU = 117.00ns {synopsys};
   "|CLK"     : FREQUENCY = 50.00ns {synopsys};
   "|n10_reg" : FREQUENCY = 100.00ns {synopsys};
END;
```

Altera provides sample files for these utilities in the **/usr/maxplus2/synopsys/bin** directory.

Converting Synopsys Hierarchical Timing Constraints into MAX+PLUS II-Compatible Format with the gen_iacf and gen_hacf Utilities

Altera provides the **gen_hacf** and **gen_iacf** utilities, which convert Synopsys timing constraints into the MAX+PLUS® II Assignment & Configuration File (**.acf**) format. For information on converting timing constraints from non-hierarchical designs, refer to Converting Synopsys Timing Constraints into MAX+PLUS II-Compatible Format with the **syn2acf** Utility. The **gen_iacf** utility generates intermediate, individual ACFs with the extension **.iacf** for each subdesign. The **gen_hacf** utility merges the individual **.iacf** files into a single ACF for the whole design.

To use the **gen_iacf** and **gen_hacf** utilities, follow these steps:

You can create a **dc_shell** script that performs most of these steps. Refer to Figure 2 for a sample **dc_shell** script.

1. Make sure that you have specified the correct path of your local Perl executable, as specified in step 5 of Setting Up the MAX+PLUS II/Synopsys Working Environment.

   The **gen_iacf** and **gen_hacf** utilities use the `ALT_HOME` environment variable, if it has been specified, to determine the MAX+PLUS II system directory; otherwise, it uses the **/usr/maxplus2** directory. To specify a different MAX+PLUS II system directory with the `ALT_HOME` environment variable, you can either edit the **.cshrc** file to specify the correct directory or type the following command at the UNIX prompt:

   `setenv ALT_HOME` <*MAX+PLUS II system directory*> ↵

2. Once you have synthesized your design with Design Compiler or FPGA Compiler, generate an hierarchical EDIF netlist file for the top-level design by typing the following command at the **dc_shell** prompt:

   `write -f edif -hierarchy` <*top-level design name*> `-o` <*top-level design name*>`.hier.edf` ↵

3. Generate intermediate ACF files (**.iacf**) for all subdesigns that have constraints, including the top-level design.

   1. Generate the following input files for the **gen_iacf** utility by using a **gen_iacf.cmd** file. Figure 1 shows a sample **gen_iacf.cmd** file.

      - Flattened EDIF netlist file
      - **dc_shell** script file
      - Standard Delay Format (SDF) constraints construct
      - SDF timing delay construct

      The **gen_iacf** and **gen_hacf** utilities do not support `set_arrival` timing constraints for internal nodes.

      Figure 1. Sample gen_iacf.cmd File

      ```
      ungroup -flatten -all
      write -f edif
      write_script > <design_name> + "_setup.dc"
      write constraints -format sdf -cover design
      ```

```
write_timing -format sdf
```

This sample command file assumes that the `design_name` variable has been set before the command file is included.

2. Run the **gen_iacf** utility for each design that has timing constraints (including the top-level design) by typing the following command at the UNIX prompt:

   `gen_iacf` <*design name*> ↵

4. Rename the top-level hierarchichal EDIF netlist file to <*top-level design name*>**.edf**, if you have not already done so.

5. Use the **gen_hacf** utility to merge the **.iacf** files for the top-level design and subdesigns into a single hierarchical ACF file, called ***<top level design name>*.acf**. Type the following command at the **dc_shell** prompt to start the **gen_hacf** utility and merge the files:

   `gen_hacf` <*top-level design name*>`[`<*sub-design file list*>`]` ↵

Figure 2 shows a sample **dc_shell** script, which includes all of the steps for using the **gen_iacf** and **gen_hacf** utilities.

Figure 2. Sample Script for Running the gen_iacf and gen_hacf Utilities

```
/* Sample dc_shell script for converting hierarchical
Synopsys timing constraints to the ACF format
The example design TOP has 3 lower-level
subdesigns - LOWER1, LOWER2, LOWER3.  Only
LOWER1, LOWER2 and TOP designs have constraints. */

link_library            = flex10k-3.db
target_library          = flex10k-3.db
synthetic_library       = flex10k-3.sldb

read -f vhdl LOWER1.vhd
read -f vhdl LOWER2.vhd
read -f vhdl LOWER3.vhd
read -f vhdl TOP.vhd

elaborate LOWER1
current_design=LOWER1

/* Include user-defined timing constraints for LOWER1 */

include timing1.cmd
compile
design_name=LOWER1

/* generate input files for gen_iacf */

include /usr/maxplus2/synopsys/bin/gen_iacf.cmd

/* generate an intermediate ACF (.iacf) file for LOWER1 design */

sh /usr/maxplus2/synopsys/bin/gen_iacf LOWER1

elaborate LOWER2
current_design=LOWER2

/* Include user-defined timing constraints for LOWER2 */

include timing2.cmd
compile
design_name=LOWER2

/* generate input files for gen_iacf */

include /usr/maxplus2/synopsys/bin/gen_iacf.cmd
```

```
/* generate an intermediate ACF (.iacf) file for LOWER2 design */

sh /usr/maxplus2/synopsys/bin/gen_iacf LOWER2

elaborate TOP
current_design=TOP

/* Include user-defined timing constraints for TOP */

include timing3.cmd
compile

/* generate a hierarchical EDIF netlist file for
the top-level design before it is flattened by
the gen_iacf.cmd utility */

write -f edif -hierarchy TOP -o TOP.hier.edf

design_name=TOP

/* generate input files for gen_iacf */

include /usr/maxplus2/synopsys/bin/gen_iacf.cmd

/* generate an intermediate ACF (.iacf) file for design TOP */

sh /usr/maxplus2/synopsys/bin/gen_iacf TOP

/* Rename the hierarchical EDIF netlist file generated
earlier to <top level design>.edf, which is required by
gen_hacf utility and MAX+PLUS II */

sh mv TOP.hier.edf TOP.edf

/* Merge all .iacf files to generate the final top-level ACF
File subdesign.list in the following command lists the names
of subdesigns that have timing constraints, one per line.
In this example it has 2 lines, one each for LOWER1 and LOWER2.
Top-level design name should not be specified in this file. */

sh /usr/maxplus2/synopsys/bin/gen_hacf TOP subdesign.list

quit
```

The **gen_iacf** utility cannot support maximum Clock frequency ($\mathbf{f_{MAX}}$) correctly if more than one Clock skew is specified in the **dc_shell** command script. This problem occurs because the Synopsys `write_script` command drops the Clock skew information for the registers. The **gen_iacf** utility will use the last Clock skew number to calculate $\mathbf{f_{MAX}}$.

All timing assignments generated by the **gen_iacf** utility are written to the Timing Requirement Assignments Section of the project's ACF, with the assignment source identifier `{synopsys}` at the end of each line. Figure 4 shows a sample ACF excerpt that contains Synopsys timing constraints generated by the **gen_iacf** utility.

Figure 4. Sample ACF Excerpt with Synopsys Timing Constraints

```
TIMING_POINT
BEGIN
   "|OUT2"    : TCO = 15.00ns {synopsys};
   "|IN1"     : TPD = 10.00ns {synopsys};
   "|IN2"     : TPD = 5.00ns {synopsys};
   "|OUT1"    : TCO = 20.00ns {synopsys};
   "|IN1"     : TSU = 20.00ns {synopsys};
   "|IN2"     : TSU = 117.00ns {synopsys};
   "|CLK"     : FREQUENCY = 50.00ns {synopsys};
   "|n10_reg" : FREQUENCY = 100.00ns {synopsys};
END;
```

The MAX+PLUS II Compiler flattens the design internally before compiling it, which may convert

some of the ports on the sub-designs into internal or buried nodes. In addition, the **gen_iacf** and **gen_hacf** utilities will correctly pass $\mathbf{t_{CO}}$ and $\mathbf{t_{PD}}$ assignments made at lower levels of hierarchy to the ACF, but the MAX+PLUS II Compiler will ignore them and generate one or more warning messages (e.g., `Warning: Ignored timing assignment for tsu|tpd|tco on buried node |TIME_STATE_MACHING:U1|tb1_3:U115|:30`). In addition, hierarchical timing constraints may result in duplicate assignments in the ACF, and the MAX+PLUS II Compiler could generate an additional warning (e.g., `Warning: Ignored redefinition of resources assignment (logic option assignment) for node 'CLK' Processing . . .`).

Performing a Pre-Routing or Functional Simulation with VSS Software

After you have synthesized and optimized a VHDL or Verilog HDL design with the Design Compiler or FPGA Compiler software, you can perform a pre-routing or functional simulation with the Synopsys VHDL System Simulator (VSS) software.

To perform a pre-routing/functional simulation, follow these steps:

1. Be sure to set up the working environment correctly, as described in the following topics:
   - Setting Up the MAX+PLUS II/Synopsys Working Environment
   - Setting Up Design Compiler & FPGA Compiler Configuration Files
   - Setting Up the DesignWare Interface
   - Setting Up VSS Configuration Files
2. Create a VHDL or Verilog HDL design file that follows the guidelines described in one of the following topics:
   - Creating VHDL Designs for Use with MAX+PLUS II Software
   - Creating Verilog HDL Designs for Use with MAX+PLUS II Software
3. Synthesize and optimize your design with the Design Compiler or FPGA Compiler, as described in Synthesizing & Optimizing VHDL & Verilog HDL Files with Design Compiler or FPGA Compiler Software.
4. Save your design as a VHDL Design File (**.vhd**).

   VSS requires each architecture/entity pair in a VHDL Design File to have a configuration. The Configuration Declaration is necessary for simulation, but not for synthesis.
5. Use VSS and one of the Altera pre-routing functional simulation libraries to simulate the design.
6. When you are ready to compile your project with MAX+PLUS II software, save the design as an EDIF netlist file (**.edf**), then process it as described in Compiling Projects with MAX+PLUS II Software.

Related Topics:

- Refer to the following sources for related information:
- ***VHDL System Simulator Core Programs Manual*** for more information about VSS
- Performing a Timing Simulation with VSS Software

Resynthesizing a Design Using the alt_vtl Library & a MAX+PLUS II SDF

Output File

Altera provides the **alt_vtl.db** post-synthesis library for technology mapping or resynthesis. You can use this library with the MAX+PLUS® II -generated Standard Delay Format (SDF) Output File (**.sdo**) to retarget and resynthesize your design for another device family by performing the following steps:

To retarget and resynthsize a design, follow these steps:

1. Generate an EDIF Output File (**.edo**) and an SDF Output File (**.sdo**), as described in Compiling Projects with MAX+PLUS II Software.

2. Modify your **.synopsys_dc.setup** file to include the following lines:

```
search_path = {./usr/maxplus2/synopsys/library/alt_post/syn/lib
<target library path>}; ↵
target_library = {<target library path>}; ↵
symbol_library = {<target library symbol file>}; ↵
link_library = {alt_vtl.db}; ↵
```

3. In the Design Compiler or FPGA Compiler software, type the following commands to read in the EDIF and SDF output files:

```
read -f edif <design name>.edo ↵
read_timing -load_delay net <design name>.sdo ↵
```

4. Type the following commands to compile your design, report the timing information, and create an EDIF netlist file (**.edf**) that can be processed with the MAX+PLUS II Compiler.

```
compile ↵
report_timing ↵
write -f edif -hierarchy -o <design name>.edf ↵
```

Using Synopsys Design Compiler or FPGA Compiler & MAX+PLUS II Software

The following topics describe how to use the Synopsys Design Compiler and FPGA Compiler software with the MAX+PLUS® II software. Choose one of the following topics for information:

Open a printable version of all topics listed on this page.

Setting Up the MAX+PLUS II/Synopsys Working Environment

- Software Requirements
- Setting Up Design Compiler & FPGA Compiler Configuration Files
- Setting Up the DesignWare Interface
  - Updating DesignWare Libraries
- Libraries
  - Design Compiler & FPGA Compiler Technology Libraries
  - VHDL & Verilog HDL **alt_mf** Logic Function Library
  - DesignWare FLEX 6000, FLEX 8000 & FLEX 10K Synthetic Libraries
  - Post-Synthesis Libraries
- MAX+PLUS II/Synopsys Interface File Organization
- MAX+PLUS II Project File Structure

Design Entry

- **Design Entry Flow**

- **VHDL**

  - Creating VHDL Designs for Use with MAX+PLUS II Software
    - Instantiating RAM & ROM Functions in VHDL (includes examples)
    - Instantiating the clklock Megafunction in VHDL or Verilog HDL (includes examples)
  - Additional examples:
    - Primitive & Old-Style Macrofunction Instantiation Example for VHDL
    - Architecture Control Logic Function Instantiation Example for VHDL
    - DesignWare Up/Down Counter Function Instantiation Example for VHDL

- **Verilog HDL**

  - Creating Verilog HDL Designs for Use with MAX+PLUS II Software
    - Instantiating RAM or ROM Functions in Verilog HDL (includes examples)
    - Instantiating the clklock Megafunction in VHDL or Verilog HDL (includes examples)
  - Additional examples:
    - Primitive & Old-Style Macrofunction Instantiation Example for Verilog HDL
    - Architecture Control Logic Function Instantiation Example for Verilog HDL

Synthesis & Optimization

- Synthesizing & Optimizing VHDL & Verilog HDL Projects with Synopsys Software
  - Using FPGA Compiler N-Input LUT Optimization for FLEX 6000, FLEX 8000, or FLEX 10K Devices
- Examples:
  - MAX 7000 & MAX 9000 Synthesis Example
  - DesignWare FLEX 8000 Synthesis Example
- Entering Resource Assignments
  - Assigning Pins, Logic Cells & Chips
  - Assigning Cliques
  - Assigning Logic Options
  - Modifying the Assignment & Configuration File with the **setacf** utility
  - Converting Synopsys Timing Constraints into MAX+PLUS II-Compatible Format with the **syn2acf** Utility
  - Converting Synopsys Hierarchical Timing Constraints into MAX+PLUS II-Compatible Format with the **gen_iacf** and **gen_hacf** Utilities
- Performing a Pre-Routing or Functional Simulation with VSS
- Resynthesizing a Design Using the **alt.vtl** Library & a MAX+PLUS II SDF Output File

Related Links:

- Go to the following MAX+PLUS II ACCESS^{SM} Key topics for related information:
  - Compiling Projects with MAX+PLUS II Software
  - Programming Altera Devices
  - Using Synopsys FPGA Express & MAX+PLUS II Software
  - Using Synopsys PrimeTime & MAX+PLUS II Software
  - Using Synopsys VSS & MAX+PLUS II Software
- Go to the following topics for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware
  - Synopsys web site (http://www.synopsys.com)

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

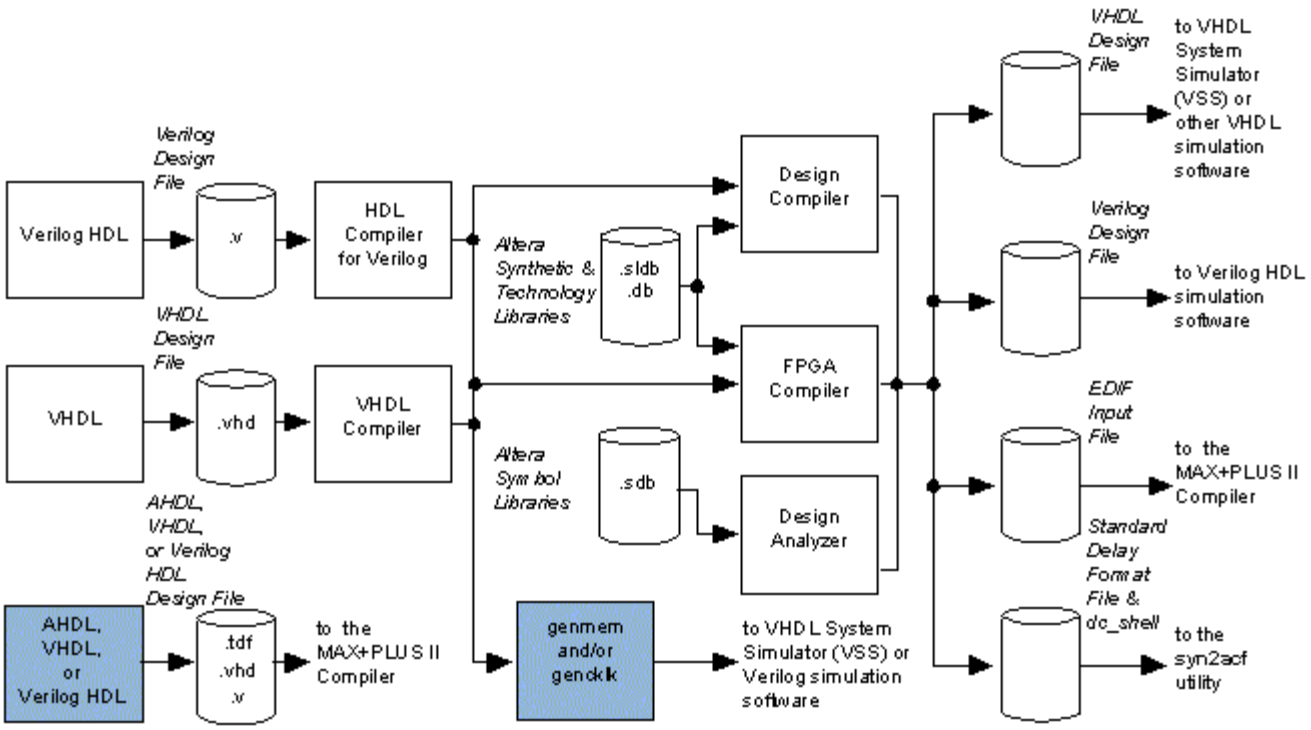
Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Synopsys Design Entry Flow

Figure 1 below shows the design entry flow for the MAX+PLUS® II /Synopsys interface.

Figure 1. MAX+PLUS II/Synopsys Design Entry Flow

Altera-provided items are shown in blue.



Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

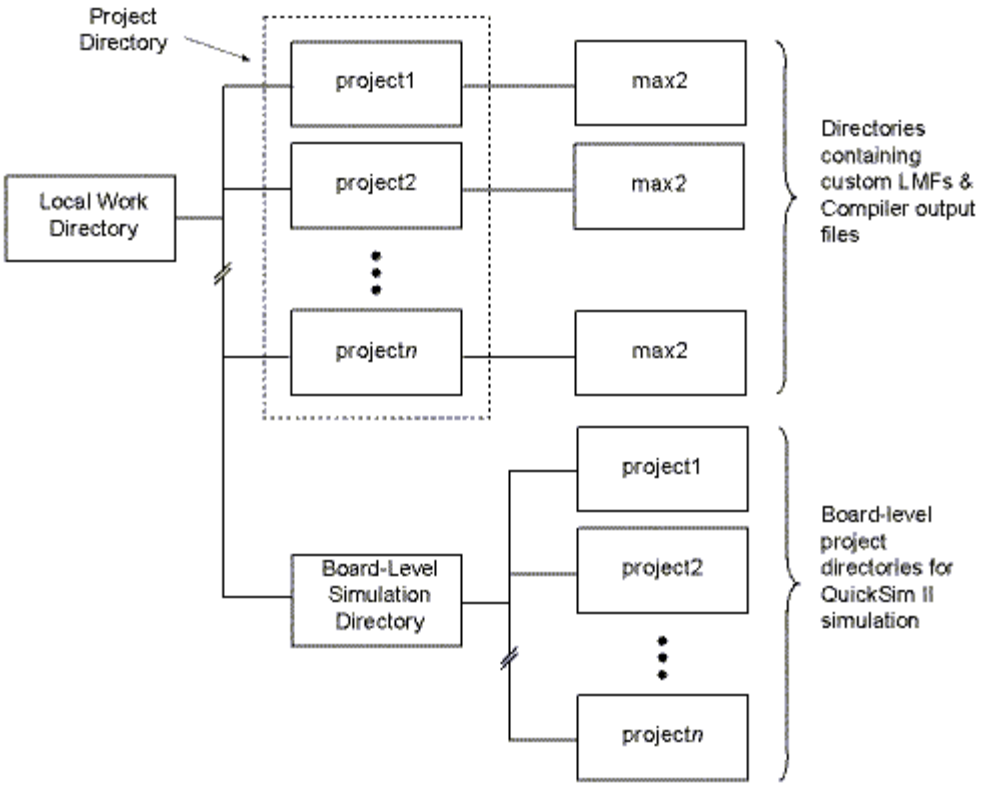
Local Work Area Directory Structure

Design Architect software automatically creates and maintains the project directory structure required for all stages of design entry. Galileo Extreme, Leonardo, and ENWrite software create a **max2** subdirectory, if it does not already exist, under the project directory. These software applications also generate EDIF netlist files, and copy them from the <*project name*> directory to this **max2** subdirectory. All MAX+PLUS ® II Compiler output files are created in the **max2** subdirectory.

Simulation files created with Mentor Graphics applications and Logic Modeling files are located in the board-level simulation subdirectory of the project directory. You can use these files during simulation with QuickSim II software.

The only directory you need to create is the local work directory, which should contain all project directories. Figure 1 shows the recommended file structure.

Figure 1. Recommended File Structure



Related Links:

Go to the following MAX+PLUS II ACCESS[SM] Key topics for related information:

- MAX+PLUS II Project Directory Structure
- Mentor Graphics Project Directory Structure

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Synplicity Design Flow

Figure 1 shows the typical design flow for logic circuits created and processed with Cadence and MAX+PLUS® II software. Design Entry Flow, Project Compilation Flow, Project Simulation Flow, and Device Programming Flow show detailed diagrams of each stage of the design flow.

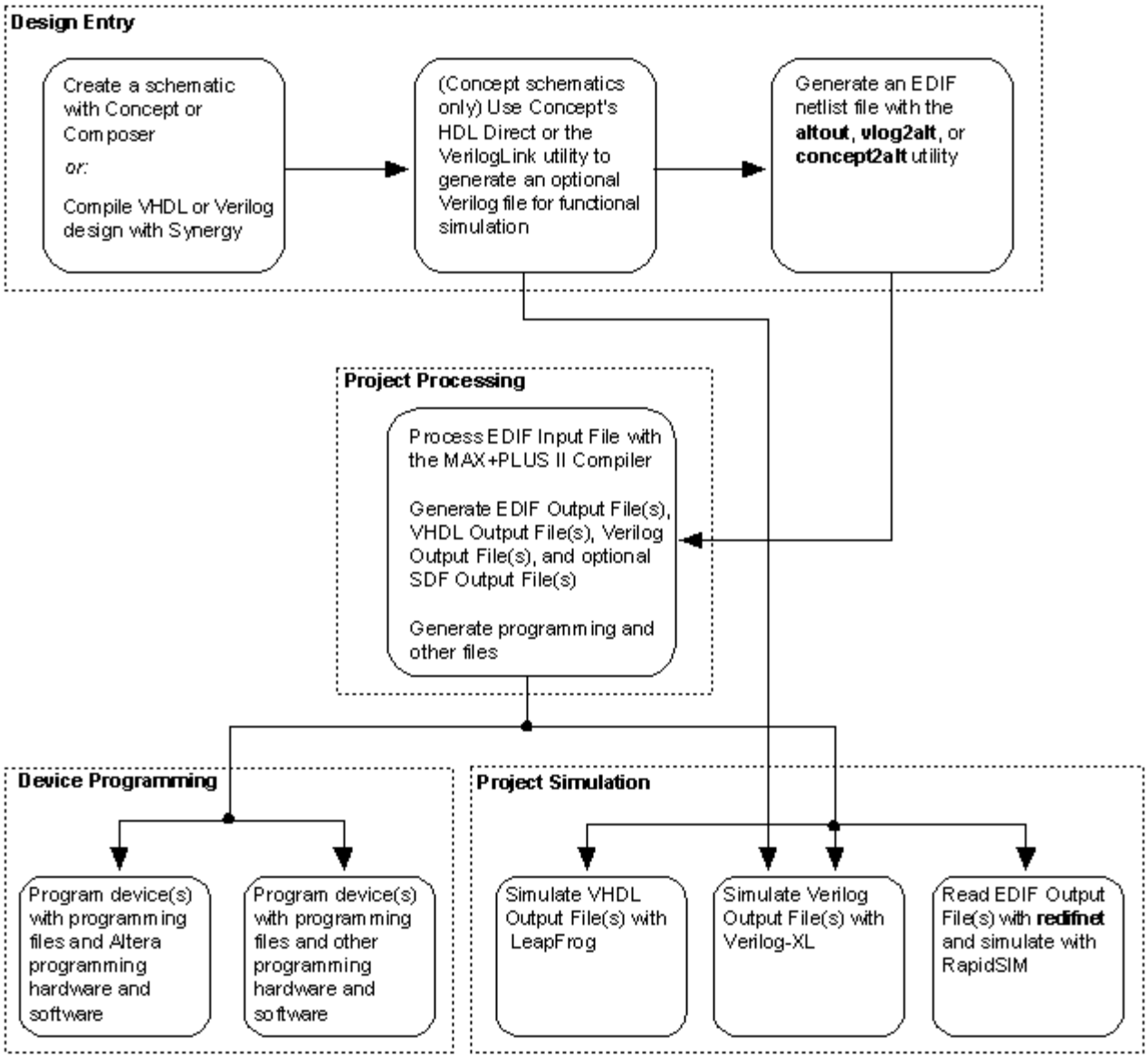


Figure 1. Design Flow between Cadence & MAX+PLUS II Software

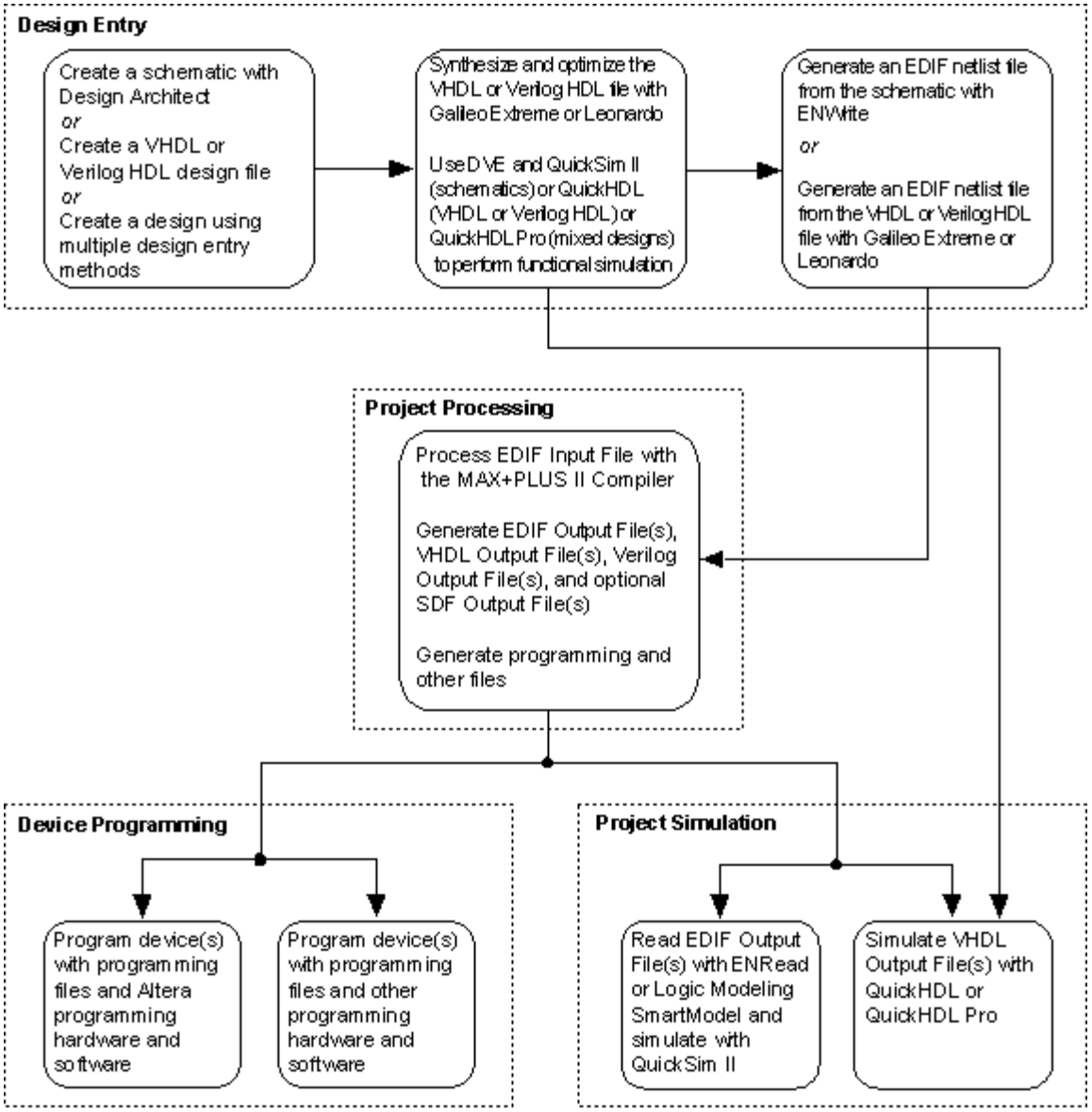
Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

The following figure shows the typical design flow for logic circuits created and processed with the MAX+PLUS ® II and Mentor Graphics/Exemplar Logic software. Detailed diagrams for each stage of the design flow appear in Design Entry Flow, Project Compilation Flow, Project Simulation/Timing Analysis Flow, and Device Programming Flow.



Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Figure 1 shows the typical design flow for logic circuits created and processed with Synplicity and MAX+PLUS® II software. Design Entry Flow, Project Compilation Flow, and Device Programming Flow show detailed diagrams of each stage of the design flow.

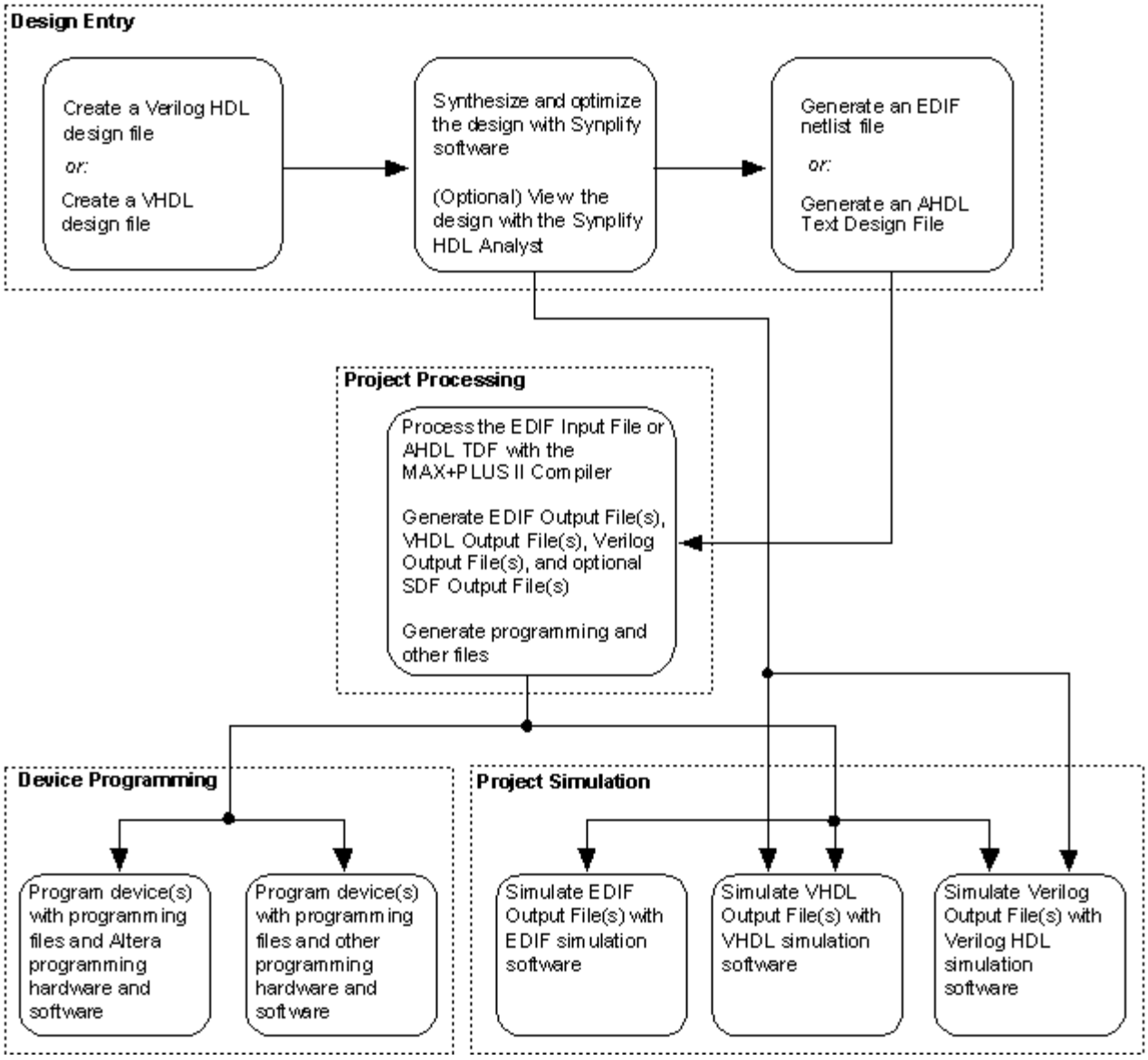


Figure 1. Design Flow between Synplicity & MAX+PLUS II Software

---

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Altera-Provided Logic & Symbol Libraries

The MAX+PLUS® II/Viewlogic Powerview environment provides libraries for compiling, synthesizing, and simulating designs.

You can create your own libraries of custom symbols and logic functions for use in ViewDraw schematics and VHDL design files. You can use custom symbols (and functions) to incorporate an EDIF Input File, TDF, or any other MAX+PLUS II-supported design file into a project. The MAX+PLUS II software uses the **vwlogic.lmf** Library Mapping File to map ViewDraw symbols to equivalent MAX+PLUS II megafunctions, macrofunctions, or primitives. To use custom symbols and functions, you can create a custom LMF that maps your custom functions to equivalent EDIF Input Files, TDFs, or other MAX+PLUS II-supported design files. Go to "Library Mapping File" and "Viewlogic Library Mapping File" in MAX+PLUS II Help for more information.

Logic symbols used in ViewDraw software are available from the MAX+PLUS II **alt_max2** library, the ViewDraw **builtin** and **74ls** libraries, and the ViewDatapath **vdpath** library. VHDL models of MAX+PLUS II logic functions are available from the Altera-provided **alt_mf** library.

The alt_max2 Library

The **alt_max2** library provides MAX+PLUS II-specific logic functions that can be used to take advantage of special architectural features in each Altera® device family. See Table 1. Symbols and functional simulation models are available for all of these elements.

The alt_mf Library

The Altera-provided **alt_mf** library, which supports the Viewlogic Vantage VHDL Analyzer software, contains VHDL simulation models for all logic functions listed in the following table. The library is configured so that these functions pass untouched through the EDIF netlist file to the MAX+PLUS II Compiler, providing you with optimal control over design processing. Altera also provides models for all of the logic functions that you can synthesize and simulate. These models allow you to perform functional VHDL simulation while maintaining an architecture-independent VHDL description.

Table 1. Architecture Control Logic Functions

Name ***Note (1), Note (2)***	**Description**	**Name**	**Description**	**Name**	**Description**
8fadd	8-bit full adder macrofunction	LCELL	Logic cell buffer primitive	EXP	MAX® 5000, MAX 7000, and MAX 9000 Expander buffer primitive
8mcomp	8-bit magnitude comparator macrofunction	GLOBAL	Global input buffer primitive	SOFT	Soft buffer primitive
8count	8-bit up/down counter macrofunction	CASCADE	FLEX® 6000, FLEX 8000, and FLEX 10K cascade buffer primitive	OPNDRN	Open-drain buffer primitive
81mux	8-to-1 multiplexer macrofunction	CARRY	FLEX 6000, FLEX 8000, and FLEX 10K cascade buffer primitive	DFFE *Note (2)*	D-type flipflop with Clock Enable primitive

`clklock`	Phase-locked loop megafunction

Notes:

1. Logic function names that begin with a number must be prefixed with "`a_`" in VHDL designs. For example, `8fadd` must be specified as `a_8fadd`.
2. For designs that are targeted to FLEX 6000 devices, you should use the `DFFE` primitive only if the design contains either a Clear or Preset signal, but not both. If your design contains both a Clear and a Preset signal, you must use the `DFFE6K` primitive.

Related Links:

- Choose **Old-Style Macrofunctions**, **Primitives**, or **Megafunctions/LPM** from the MAX+PLUS II Help menu for detailed information on these functions.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

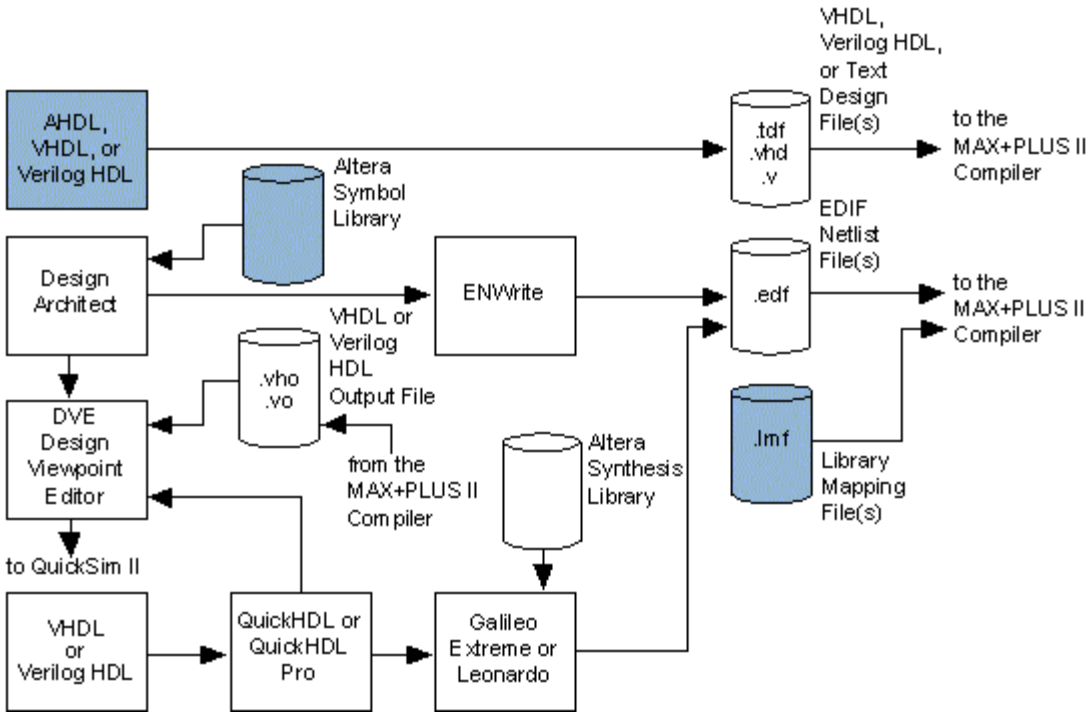
Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Mentor Graphics/Exemplar Logic Design Entry Flow

The following figure shows the design entry flow for the MAX+PLUS® II/Mentor Graphics/Exemplar Logic interface.

Figure 1. MAX+PLUS II/Mentor Graphics/Exemplar Logic Design Entry Flow

Alteraprovided items are shown in blue.



Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Setting Up the DesignWare Interface

The DesignWare interface synthesizes FLEX® 6000 , FLEX 8000 and FLEX 10K designs by operator inference. It replaces the HDL operators +, -, >, <, >=, and <= with FLEX-optimized design implementations.

Altera provides DesignWare Synthetic Libraries that are pre-compiled for the current version of Synopsys tools. These library files are located in the **/usr/maxplus2/synopsys/library/alt_syn/<*device family*>/lib** directory.

To use the DesignWare interface with FLEX 6000, FLEX 8000 and FLEX 10K devices, follow these steps:

1. Add `synthetic_library` and `define_design_lib` parameters to your **.synopsys_dc.setup** configuration file and modify the `link_library` parameter as shown in Table 1 or Table 2.

Table 1. DesignWare Parameters to Add to the .synopsys_dc.setup File for the Design Compiler Software

Device Family	*Parameters to Add to the .synopsys_dc.setup File*

FLEX 6000	`synthetic_library = {flex6000<`*speed grade*`>.sldb};` ↵
	`link_library = {flex6000<`*speed grade*`>.sldb flex6000<`*speed grade*`>.db};` ↵
	`define_design_lib DW_FLEX6000<`*speed grade*`> -path`
	`/usr/maxplus2/synopsys/library/alt_syn/flex6000/lib/`
	`dw_flex6000<`*speed grade*`>` ↵

FLEX 8000	`synthetic_library = {flex8000[<`*speed grade*`>].sldb};` ↵
	`link_library = {flex8000[<`*speed grade*`>].sldb flex8000[<`*speed grade*`>].db};` ↵
	`define_design_lib DW_FLEX8000[<`*speed grade*`>] -path`
	`/usr/maxplus2/synopsys/library/alt_syn/flex8000`
	`/lib/dw_flex8000[<`*speed grade*`>]` ↵

FLEX 10K	`synthetic_library = {flex10k[<`*speed grade* `>].sldb};` ↵
	`link_library = {flex10k[<`*speed grade*`>].sldb flex10k[<`*speed grade*`>].db};` ↵
	`define_design_lib DW_FLEX10k[<`*speed grade*`>] -path`
	`/usr/maxplus2/synopsys/library/alt_syn/flex10k/lib`
	`/dw_flex10k[<`*speed grade*`>]` ↵

Table 2. DesignWare Parameters to Add to the .synopsys_dc.setup File for the FPGA Compiler Software

Device Family	*Parameters to Add to the .synopsys_dc.setup File*

FLEX 6000	`synthetic_library = {flex6000`
	`<`*speed grade*`>_fpga.sldb};` ↵
	`link_library = {flex6000<`*speed grade*`>_fpga.sldb flex6000<`*speed grade*`>_fpga.db};` ↵
	`define_design_lib DW_FLEX6000<`*speed grade*`>_FPGA -path`
	`/usr/maxplus2/synopsys/library/alt_syn/flex6000`
	`/lib/dw_flex6000<`*speed grade*`>_fpga` ↵

FLEX 8000	`synthetic_library = {flex8000[<`*speed grade*`>]_fpga.sldb};` ↵ `link_library = {flex8000[<`*speed grade*`>]_fpga.sldb flex8000[<`*speed grade*`>]_fpga.db};` ↵ `define_design_lib DW_FLEX8000[<`*speed grade*`>]_FPGA -path` `/usr/maxplus2/synopsys/library/alt_syn/flex8000/lib /dw_flex8000[<`*speed*

FLEX 10K	*grade*>]_fpga ↵ synthetic_library = {flex10k[<*speed grade*>]_fpga.sldb}; ↵ link_library = {flex10k[<*speed grade*>]_fpga.sldb flex10k[<*speed grade*>]_fpga.db}; ↵ define_design_lib DW_FLEX10k[<*speed grade*>]_FPGA -path /usr/maxplus2/synopsys/library/alt_syn/flex10k/lib /dw_flex10k[<*speed grade*>]_fpga ↵

2. Specify the libraries listed in Table 3 as your synthetic library and as the first of your link libraries.

   For FLEX 6000 devices, you must specify either -2 or -3 for the <*speed grade*> variable. For FLEX 8000 and FLEX 10K devices, you can specify -2, -3, -4, -5, or -6; or -2, -3, -4, or -5; respectively, for the <*speed grade*> variable. If you do not specify a speed grade for FLEX 8000 or FLEX 10K devices, the MAX+PLUS® II software selects the fastest device in the specified family as the target device.

   Table 3. FLEX 6000, FLEX 8000 & FLEX 10K DesignWare Synthetic Libraries

Altera® Device Family	Synopsys Design Compiler	Synopsys FPGA Compiler
FLEX 6000 Synthetic Library	**flex6000-2.sldb** **flex6000-3.sldb**	**flex6000-2_fpga.sldb** **flex6000-3_fpga.sldb**
FLEX 8000 Synthetic Library	**flex8000.sldb** **flex8000-2.sldb** **flex8000-3.sldb** **flex8000-4.sldb** **flex8000-5.sldb** **flex8000-6.sldb**	**flex8000_fpga.sldb** **flex8000-2_fpga.sldb** **flex8000-3_fpga.sldb** **flex8000-4_fpga.sldb** **flex8000-5_fpga.sldb** **flex8000-6_fpga.sldb**
FLEX 10K Synthetic Library	**flex10k.sldb** **flex10k-2.sldb** **flex10k-3.sldb** **flex10k-4.sldb** **flex10k-5.sldb**	**flex10k_fpga.sldb** **flex10k-2_fpga.sldb** **flex10k-3_fpga.sldb** **flex10k-4_fpga.sldb** **flex10k-5_fpga.sldb**

3. If necessary, compile the DesignWare libraries, as described in Updating DesignWare Libraries. Altera provides pre-compiled DesignWare libraries, as described above. However, Altera also provides compatible source files and scripts that allow you to automate the compilation process. These source files allow you to use DesignWare with any version of the Design Compiler. They also allow you to install components whose source is written in VHDL, even if you are licensed only for the HDL Compiler for Verilog.

Related Links:

- Go to the following MAX+PLUS II ACCESS℠ Key topics for related information:
  - Setting Up Design Compiler & FPGA Compiler Configuration Files
  - DesignWare FLEX 8000 Synthesis Example
  - Design Compiler & FPGA Compiler Technology Libraries

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Synopsys DesignWare-Specific Compiler Settings

If you are compiling a design that was created with the Synopsys DesignWare software, follow these additional steps:

1. Choose **Global Project Logic Synthesis** (Assign menu).
2. Select the desired style in the *Global Project Synthesis Style* box.
3. Choose the **Define Synthesis Style** button to check and/or edit the selected style.
4. In the **Define Synthesis Style** dialog box, select *Manual* in the *Carry Chain* box and also in the *Cascade Chain* box.
5. Choose the **Advanced Options** button in the **Define Synthesis Style** dialog box.
6. Turn on the *SOFT Buffer Insertion* logic option in the **Advanced Options** dialog box if it is not on already. This option should be turned on in all Synopsys designs.
7. Choose **OK** three times to close all dialog boxes.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

DesignWare FLEX 8000 Synthesis Example

Figure 1 shows a sample VHDL design, design_one.vhd, which illustrates component inference with the DesignWare interface for FLEX 8000 devices.

Figure 1. VHDL Design File (design_one.vhd)

This design illustrates the sum of A + B.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;



ENTITY design_one IS
   PORT (a,b : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
         f   : OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
END design_one;



ARCHITECTURE add_design OF design_one IS



BEGIN
 f <= a + b;
END add_design;
```

When the VHDL Compiler or the HDL Compiler for Verilog software analyzes and elaborates the design, it replaces the "+" operator with its synthetic operator equivalent.

Figure 2 shows the design as it appears in the Design Analyzer software after it has been analyzed and elaborated by the VHDL Compiler software.

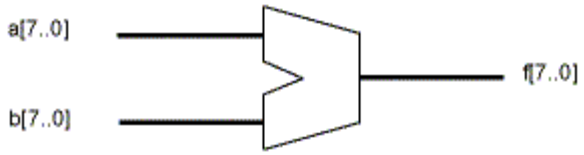


Figure 2. design_one.vhd after Analysis & Elaboration

When you synthesize a design, the Design Compiler or FPGA Compiler software uses the synthetic library to match the synthetic operator to the FLEX-optimized logical implementation in the technology library. The Synopsys Design Compiler or FPGA Compiler software then instantiates and interconnects the correct number of `flex_add` and `flex_carry` functions to produce the 8-bit adder shown in Figure 1. When you save a compiled design as a VHDL, Verilog HDL or EDIF file, the file preserves the number of `flex_add` and `flex_carry` functions, as well as their interconnections. Consequently, area and performance predictions that you make in the

Synopsys design environment closely match the final MAX+PLUS II result.

Table 2 lists functions included in the DesignWare FLEX 6000, FLEX 8000, and FLEX 10K synthetic libraries.

Table 2. FLEX 6000, FLEX 8000, and FLEX 10K Synthetic Library Functions

Name	Function
`flex_add`	Sum of A, B, and Carry-In
`flex_carry`	Carry of A, B, and Carry-In
`flex_sub`	Difference of A, B, and Borrow-In
`flex_borrow`	Borrow of A, B, and Borrow-In
`flex_gt, flex_sgt`	Greater than (`flex_gt` is unsigned; `flex_sgt` is signed)
`flex_carry_gt`	Greater than Carry
`flex_lt, flex_slt`	Less than (`flex_lt` is unsigned; `flex_slt` is signed)
`flex_carry_lt`	Less than Carry
`flex_gteq, flex_sgteq`	Greater than or equal to (`flex_gteq` is unsigned; `flex_sgteq` is signed)
`flex_carry_gteq`	Greater than or equal to Carry
`flex_inc`	Incrementer (Count = Count + 1)
`flex_carry_inc`	Incrementer Carry (Count = Count + 1)
`flex_dec`	Decrementer (Count = Count - 1)
`flex_carry_dec`	Decrementer Carry (Count = Count - 1)
`flex_lteq, flex_slteq`	Less than or equal to (`flex_lteq` is unsigned; `flex_slteq` is signed)
`flex_carry_lteq`	Less than or equal to Carry
`flex_count`	Counter
`aflex_carry_count`	Counter Carry
`flex_add_sub`	Adder/Subtractor
`flex_inc_dec`	Incrementer/Decrementer
`flex_umult, flex_smult`	Multiplier (`flex_umult` is unsigned; `flex_smult` is signed)

Figure 3 shows **design_one.vhd** after it has been synthesized with the Design Compiler.

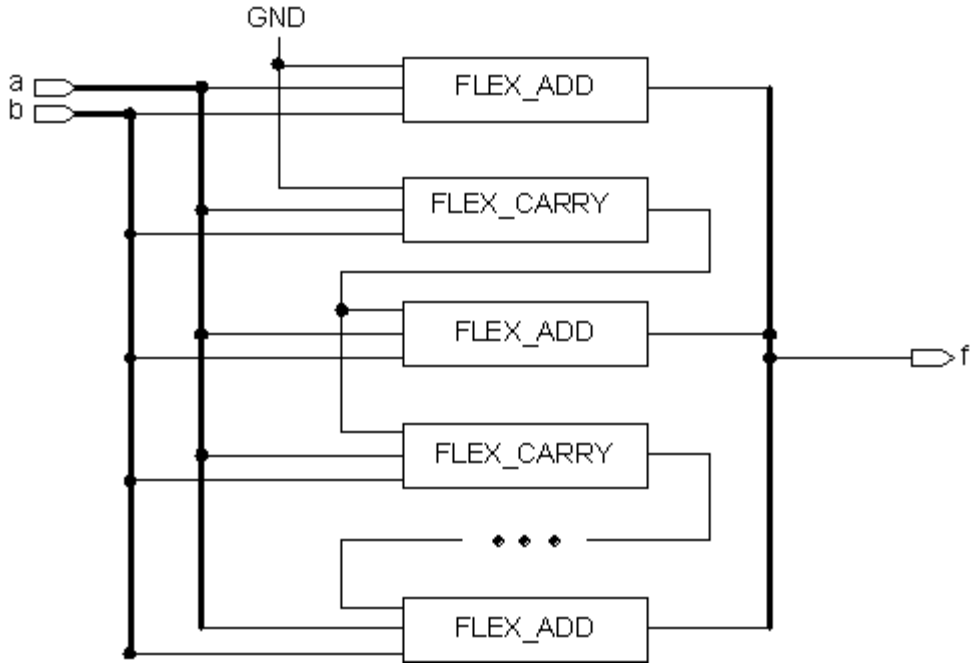


Figure 3. design_one.vhd Synthesized & Resolved for FLEX 6000, FLEX 8000 & FLEX 10K Architecture

After you save the design as an EDIF Input File (**.edf**) and process it with the MAX+PLUS II Compiler, the Compiler replaces instances of `flex_add` and `flex_carry` with FLEX-optimized versions, as shown in Figure 4. The MAX+PLUS II Compiler maps these functions into a single logic element (LE). The result is a high-speed 8-bit adder that fits into 8 LEs.

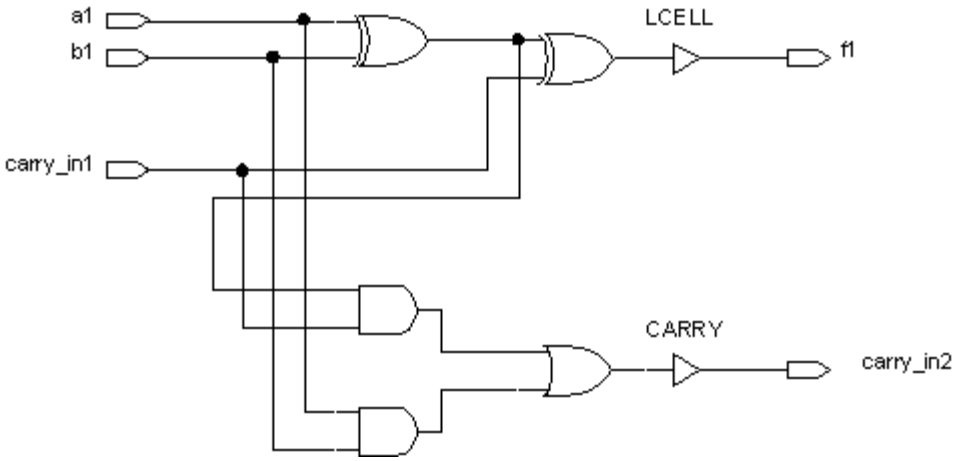


Figure 4. One Slice of the design_one 8-bit Adder Design with Optimized FLEX 8000 Functions

Related Links:

- Refer to the following sources for related information on DesignWare and the Synopsys VHDL Compiler:
  - ***Synopsys DesignWare Databook***
  - ***VHDL Compiler Reference Manual***

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Performing a Functional Simulation with DVE & QuickSim II Software

You can perform a functional simulation of a Design Architect schematic with the Mentor Graphics Design Viewpoint Editor (DVE) and QuickSim II software before compiling your project with the MAX+PLUS ® II Compiler.

☞ If you wish to functionally simulate a hierarchical design that uses multiple design entry methods, you should use QuickHDL Pro rather than QuickSim. Refer to Performing a Functional Simulation with QuickHDL Pro Software for more information.

To functionally simulate a Design Architect schematic, go through the following steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment.
2. Create a Design Architect schematic that follows the guidelines in Creating Design Architect Schematics for Use with MAX+PLUS II Software.
3. In the Navigator window, select your project's folder, press Button 3, and choose **Open max2_fve** to start DVE. DVE checks the design and creates a viewpoint (called **altera_fsim** by default) for functional simulation with QuickSim II software.
4. Select the **altera_fsim** icon, press Button 3, and choose **Open max2_qsim** from the Navigator window to start the QuickSim II software. You can also start the QuickSim II software by typing `max2_qsim` ↵ at the UNIX prompt.
5. Set the appropriate options and simulate your design.
6. Use the ENWrite utility to generate an EDIF netlist file that can be imported into the MAX+PLUS II software, as described in Converting Design Architect Schematics into MAX+PLUS II-Compatible EDIF Netlist Files with the ENWrite Utility.

Related Links:

- Go to Compiling Projects with MAX+PLUS II Software in these MAX+PLUS II ACCESS℠ Key topics for related information.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Converting Concept Schematics into MAX+PLUS II-Compatible EDIF Netlist Files with the concept2alt Utility

You can use the **altout** utility to generate an EDIF netlist file from a Composer schematic. This file can then be imported into the MAX+PLUS® II software as an EDIF Input File (**.edf**).

To convert Composer schematics into MAX+PLUS II-compatible EDIF netlist files, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment.
2. Create a Composer schematic and save it in your working directory, as described in Creating Composer Schematics for Use with MAX+PLUS II Software.

   You can instantiate MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.

3. Type the following command at the UNIX prompt from the working directory that contains the schematic:

   `altout -lib` <*design name*> `-rundir max2` <*design name*> ↵
4. Process the <*design name*>**.edf** file with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

You can use the **concept2alt** utility to generate an EDIF netlist file from a Concept schematic. This file can then be imported into the MAX+PLUS® II software as an EDIF Input File (**.edf**).

To convert a Concept schematic into an EDIF netlist file, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment.
2. Create a Concept schematic and save it in your working directory, as described in Creating Concept

Schematics for Use with MAX+PLUS II Software.

You can instantiate MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.

3. Type the following command at the UNIX prompt from the **/source** directory that contains the schematic:

   `concept2alt -rundir ../max2` *<design name>* ↵

   If your design uses library of parameterized modules (LPM) functions, you must also include the `-family` option. For example:

   `concept2 alt -family FLEX10K -rundirmax2` *<design name>* ↵

4. Process the *<design name>***.edf** file with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Converting Design Architect Schematics into MAX+PLUS II-Compatible EDIF Netlist Files with the ENWrite Utility

After you have created a Design Architect schematic or a hierarchical schematic design that uses multiple design entry methods, you can use the Mentor Graphics ENWrite utility to convert it into an EDIF netlist file that can be processed with the MAX+PLUS ® II software.

To generate an EDIF netlist file for use with the MAX+PLUS II Compiler, go through the following steps:

1. Create a Design Architect Schematic that follows the guidelines described in Creating Design Architect Schematics for Use with MAX+PLUS II Software.

2. Select the folder for your project, press Button 3, and choose **Open max2_enw** from the Navigator window to open Design Viewpoint Editor (DVE), then ENWrite. You can also start the ENWrite utility by typing `max2_enw` ↵ at the UNIX prompt.

3. Choose **OK** in the **$invoke_enw** dialog box to accept the default names for the DVE viewpoint **altera_edif**, which is used internally by ENWrite, and the ENWrite hierarchical EDIF netlist file <*design name*>**.edf**. Specify *OFF* for the port array construct in the EDIF netlist file.

   ☞ The MAX+PLUS II software supports bus constructs in EDIF 2 0 0 and 3 0 0 netlist files, which allow you to retain any bus structures in your design. To preserve a bus in the EDIF netlist file, turn on the *port array* construct option in the **$invoke_enw** dialog box. However, if your design contains library of parameterized modules (LPM) functions, you should not use this feature because LPM 2.0.1 and 2.1.0 functions do not support EDIF bus constructs.

   After DVE checks the Design Architect schematic, ENWrite generates <*design name*>**.edf** and automatically copies it to your project's directory.

4. Compile the resulting EDIF netlist file with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

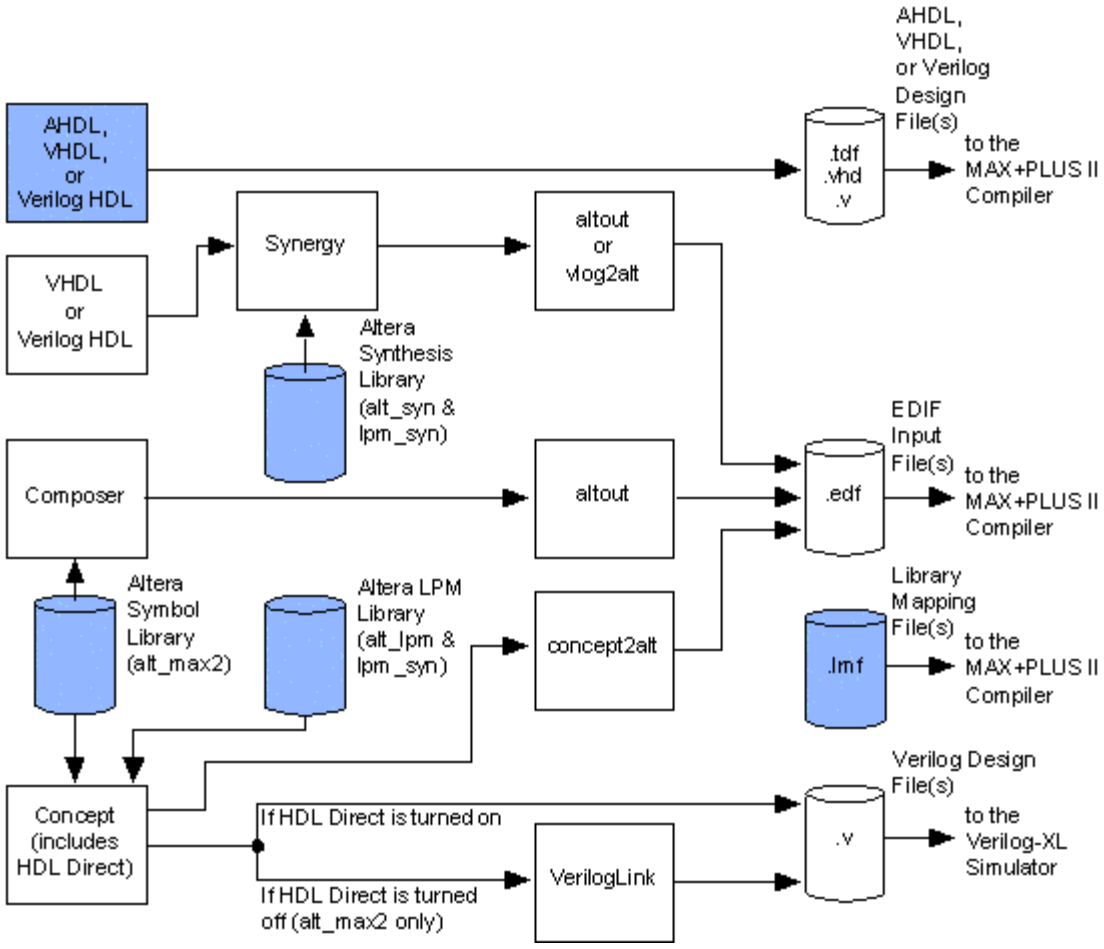
Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Cadence Design Entry Flow

Figure 1 shows the design entry flow for the MAX+PLUS® II/Cadence interface.

Figure 1. MAX+PLUS II/Cadence Design Entry Flow

Altera-provided items are shown in blue.



Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

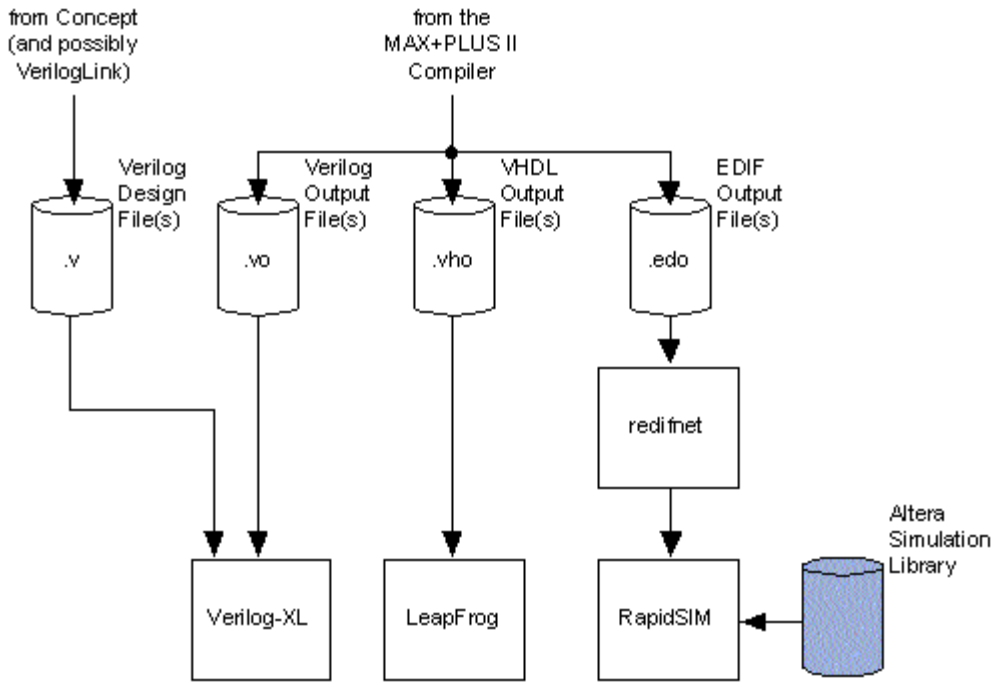
Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Project Simulation Flow

Figure 1 shows the project simulation flow for the MAX+PLUS® II/Cadence interface.

Figure 1. MAX+PLUS II/Cadence Project Simulation Flow

Altera-provided items are shown in blue.



Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

MAX+PLUS II/Mentor Graphics/Exemplar Logic Interface File Organization

The following table shows the MAX+PLUS ® II/Mentor Graphics interface subdirectories that are created in the MAX+PLUS II system directory (by default, the **/usr/maxplus2** directory) during MAX+PLUS II installation.

☞ For information on the other directories that are created during MAX+PLUS II installation, see "MAX+PLUS II File Organization" in *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual.

Table 1. MAX+PLUS II Directory Organization

Directory	Description
.lmf	Contains the Altera-provided Library Mapping Files, **mnt8_bas.lmf** and **exemplar.lmf**, that map Mentor Graphics and Exemplar Logic logic functions to equivalent MAX+PLUS II logic functions.
./mentor	Contains the AMPLE userware for the MAX+PLUS II/Mentor Graphics interface.
./simlib/mentor/alt_max2	Contains MAX+PLUS II primitives such as `CARRY`, `CASCADE`, `EXP`, `GLOBAL`, `LCELL`, `SOFT`, `OPNDRN`, `DFFE`, and `DFFE6K` (D flipflop with Clock Enable) for use in Design Architect schematics.
./simlib/mentor/max2sim	Contains the MAX+PLUS II/Mentor Graphics simulation model library, **max2sim**, for use with QuickSim II and QuickPath software.
./simlib/mentor/synlib	Contains the MAX+PLUS II synthesis library for use with AutoLogic II software, which supports synthesis for users running Mentor Graphics version B1.
./simlib/mentor/alt_mf	Contains the MAX+PLUS II macrofunction and megafunction libraries.
./simlib/mentor/alt_vtl	Contains the MAX+PLUS II VITAL library.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

MAX+PLUS II/Synopsys Interface File Organization

Table 1 shows the MAX+PLUS® II /Synopsys interface subdirectories that are created in the MAX+PLUS II system directory (by default, the /usr/maxplus2 directory) during the MAX+PLUS II software installation. For information on the other directories that are created during the MAX+PLUS II software installation, see "MAX+PLUS II File Organization" in MAX+PLUS II Installation in the MAX+PLUS II Getting Started manual.

☞ You must add the **/usr/maxplus2/bin** directory to the `PATH` environment variable in your **.cshrc** file in order to run the MAX+PLUS II software.

Table 1. MAX+PLUS II Directory Organization

Directory	Description
./synopsys/bin	Contains script programs to convert Synopsys timing constraints into MAX+PLUS II Assignment & Configuration File (**.acf**) format, and to analyze VHDL System Simulator simulation models.
./synopsys/config	Contains sample **.synopsys_dc.setup** and **.synopsys_vss.setup** files.
./synopsys/examples	Contains sample files, including those discussed in these ACCESS Key Guidelines.
./synopsys/library/alt_pre/<*device family*>**/src**	Contains VHDL simulation libraries for functional simulation of VHDL projects.
./synopsys/library/alt_pre/verilog/src	Contains the Verilog HDL functional simulation library for Verilog HDL projects.
./synopsys/library/alt_pre/vital/src	Contains the VITAL 95 simulation library. You use this library when you perform functional simulation of the design before compiling it with the MAX+PLUS II software.
./synopsys/library/alt_syn//<*device family*>**/lib**	Contains interface files for the MAX+PLUS II/Synopsys interface. Technology libraries in this directory allow the Design Compiler and FPGA Compiler to map designs to Altera® device architectures.
./synopsys/library/alt_mf/src	Contains behavioral VHDL models of some Altera macrofunctions, along with their component declarations. The `a_81mux`, `a_8count`, `a_8fadd`, and `a_8mcomp` macrofunctions are currently supported. Libraries in this directory allow you to instantiate, synthesize, and simulate these macrofunctions.
./synopsys/library/alt_post/syn/lib	Contains the post-synthesis library for technology mapping.
./synopsys/library/alt_post/sim/src	Contains the VHDL source files for the VITAL 95-compliant library. You use this library when you perform simulation of the design after compiling it with the MAX+PLUS II software.

Related Links:

- Go to the following topics for additional information:
  - ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
  - This manual is also available in 4 parts:
    - Preface & Section 1: MAX+PLUS II Installation
    - Section 2: MAX+PLUS II - A Perspective
    - Section 3: MAX+PLUS II Tutorial
    - Appendices, Glossary & Index

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

MAX+PLUS II/Synplicity Interface File Organization

Table 1 shows the MAX+PLUS® II/Synplicity interface subdirectories that are created in the MAX+PLUS II system directory (by default, the **/usr/maxplus2** directory) during MAX+PLUS II installation. For information on the other directories that are created during MAX+PLUS II installation, see "MAX+PLUS II File Organization" in *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual.

Table 1. MAX+PLUS II Directory Organization

Directory	Description
./lmf	Contains the Altera-provided Library Mapping File, **synplcty.lmf**, which maps Synplicity logic functions to equivalent MAX+PLUS II logic functions.

Related Links:

- Go to the following topics for additional information:
  - ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
  - This manual is also available in 4 parts:
    - Preface & Section 1: MAX+PLUS II Installation
    - Section 2: MAX+PLUS II - A Perspective
    - Section 3: MAX+PLUS II Tutorial
    - Appendices, Glossary & Index

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Altera DesignWare FLEX 6000, FLEX 8000 & FLEX 10K Synthetic Libraries

The Altera® DesignWare interface for the FLEX® 6000, FLEX 8000, and FLEX 10K device families provides accurate area and timing prediction for designs that have been synthesized by the Synopsys design tools and targeted for FLEX devices. Altera's DesignWare interface also ensures that the area and timing information closely matches the final FLEX device implementation generated by the MAX+PLUS® II Compiler. The DesignWare interface synthesizes FLEX 6000 , FLEX 8000 and FLEX 10K designs by operator inference. This interface supports bus widths of up to 32 bits, except adder functions, which support bus widths of up to 64 bits.

The Altera DesignWare interface for FLEX devices offers three major advantages to Synopsys designers:

- Automatic access to FLEX carry and cascade chain functions
- Optimal routing of FLEX designs
- Improved area and performance prediction capability in Synopsys tools

Table 1 lists the Altera DesignWare synthetic libraries for FLEX 6000, FLEX 8000, and FLEX 10K devices.

Table 1. FLEX 6000, FLEX 8000 & FLEX 10K DesignWare Synthetic Libraries

Altera Device Family	Synopsys Design Compiler	Synopsys FPGA Compiler
FLEX 6000 Synthetic Library	**flex6000-2.sldb**	**flex6000-2_fpga.sldb**
	flex6000-3.sldb	**flex6000-3_fpga.sldb**
FLEX 8000 Synthetic Library	**flex8000.sldb**	**flex8000_fpga.sldb**
	flex8000-2.sldb	**flex8000-2_fpga.sldb**
	flex8000-3.sldb	**flex8000-3_fpga.sldb**
	flex8000-4.sldb	**flex8000-4_fpga.sldb**
	flex8000-5.sldb	**flex8000-5_fpga.sldb**
	flex8000-6.sldb	**flex8000-6_fpga.sldb**
FLEX 10K Synthetic Library	**flex10k.sldb**	**flex10k_fpga.sldb**
	flex10k-2.sldb	**flex10k-2_fpga.sldb**
	flex10k-3.sldb	**flex10k-3_fpga.sldb**
	flex10k-4.sldb	**flex10k-4_fpga.sldb**
	flex10k-5.sldb	**flex10k-5_fpga.sldb**

Table 2 lists functions included in the DesignWare FLEX 6000, FLEX 8000, and FLEX 10K synthetic libraries. Refer to DesignWare FLEX 8000 Synthesis Example for an example showing how DesignWare synthesis affects design processing.

Table 2. FLEX 6000, FLEX 8000, and FLEX 10K Synthetic Library Functions

Name	Function
`flex_add`	Sum of A, B, and Carry-In
`flex_carry`	Carry of A, B, and Carry-In

flex_sub	Difference of A, B, and Borrow-In
flex_borrow	Borrow of A, B, and Borrow-In
flex_gt, flex_sgt	Greater than (flex_gt is unsigned; flex_sgt is signed)
flex_carry_gt	Greater than Carry
flex_lt, flex_slt	Less than (flex_lt is unsigned; flex_slt is signed)
flex_carry_lt	Less than Carry
flex_gteq, flex_sgteq	Greater than or equal to (flex_gteq is unsigned; flex_sgteq is signed)
flex_carry_gteq	Greater than or equal to Carry
flex_inc	Incrementer (Count = Count + 1)
flex_carry_inc	Incrementer Carry (Count = Count + 1)
flex_dec	Decrementer (Count = Count - 1)
flex_carry_dec	Decrementer Carry (Count = Count - 1)
flex_lteq, flex_slteq	Less than or equal to (flex_lteq is unsigned; flex_slteq is signed)
flex_carry_lteq	Less than or equal to Carry
flex_count	Counter
aflex_carry_count	Counter Carry
flex_add_sub	Adder/Subtractor
flex_inc_dec	Incrementer/Decrementer
flex_umult, flex_smult	Multiplier (flex_umult is unsigned; flex_smult is signed)

Related Links

- Setting Up the DesignWare Interface in these MAX+PLUS II ACCESS^{SM} Key topics
- *Synopsys DesignWare Databook*
- *VHDL Compiler Reference Manual*

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Using Synopsys* FPGA Express and MAX+PLUS® II Software

The following topics describe how to use the FPGA Express and MAX+PLUS® II software. Choose one of the following topics for more information:

Open a printable version of all topics listed on this page.

Software Requirements

Design Flow for FPGA Express Software

Design Entry

- **Design Entry Flow**
- **VHDL**
  - Creating VHDL Designs for Use with MAX+PLUS II Software
    - Instantiating the clklock Megafunction in VHDL or Verilog HDL
    - Instantiating LPM Functions in VHDL
    - Instantiating RAM & ROM Functions in VHDL
- **Verilog HDL**
  - Creating Verilog HDL Designs for Use with MAX+PLUS II Software
    - Instantiating the clklock Megafunction in VHDL or Verilog HDL
    - Instantiating LPM Functions in Verilog HDL
    - Instantiating RAM & ROM Functions in Verilog HDL

Synthesis & Optimization

- Synthesizing & Optimizing VHDL or Verilog HDL Files with FPGA Express Software
- Entering Resource, Device & Global Logic Synthesis Assignments
  - Assigning a Device & Clock Frequency ($\mathbf{f_{MAX}}$)
  - Assigning Pins, Logic Options, and $\mathbf{t_{SU}}$, $\mathbf{t_{CO}}$ and $\mathbf{t_{PD}}$ Timing Constraints
  - Specifying the MAX+PLUS II Logic Synthesis Style with FPGA Express Software
  - Using ACFs Generated by FPGA Express Software
  - Modifying the Assignment & Configuration File with the **setacf** Utility
- Analyzing Estimated Timing with the FPGA Express Time Tracker
- Specifying Speed/Area & CPU Effort Settings with the FPGA Express Software

Compilation

- Project Compilation Flow
- Compiling Projects with MAX+PLUS II Software

Related Links

- Programming Altera Devices
- Using Synopsys Design Compiler or FPGA Compiler & MAX+PLUS II Software
- Using Synopsys PrimeTime & MAX+PLUS II Software
- Using Synopsys VSS & MAX+PLUS II Software
- About the MAX+PLUS II Software
- Altera Programming Hardware
- Synopsys web site (http://www.synopsys.com)

Using ACFs Generated by FPGA Express Software

With FPGA Express software, you can either generate a new Assignment & Configuration File (**.acf**), along with an EDIF netlist file (**.edf**) and Library Mapping File (**.lmf**), to be imported into the MAX+PLUS® II software, or you can place a copy of an existing ACF in the FPGA Express output directory. If you use an existing ACF, FPGA Express updates the ACF with additional information, such as the global project synthesis style, as it processes the design. Each line in the ACF that is modified by the FPGA Express software is marked with a `{synopsys}` comment at the end. You should then place this ACF in the MAX+PLUS II project directory.

☞ If an existing ACF has been modified by FPGA Express, then processed by the MAX+PLUS II Compiler, the resulting ACF may specify an incorrect LMF. If so, the MAX+PLUS II software displays the error message `Can't find design file` <*cell name*>. You can correct this error in MAX+PLUS II software by specifying the FPGA Express-generated <*project name*>**.lmf** file in the *LMF #1* box in the **EDIF Netlist Reader Settings** dialog box. See Compiling Projects with MAX+PLUS II Software for more information.

The ACF incorporates the following assignments from FPGA Express software and passes them to the MAX+PLUS II software:

- The Altera device family and optional specific device and speed grade specified in the **Create Implementation** dialog box
- The global project logic synthesis style specified in the **Create Implementation** dialog box
- The Clock frequency ($\mathbf{f_{MAX}}$) specified in the **Create Implementation** dialog box or the *Clocks* constraint table
- Clock speeds ($\mathbf{t_{PD}}$) specified in the *Clocks* constraint table
- Path group constraints specified in the *Paths* constraint table
- Pin assignments; settings for the Slow Slew Rate and Fast I/O logic options; and input-to-setup ($\mathbf{t_{SU}}$) and clock-to-output ($\mathbf{t_{CO}}$) delays specified in the *Ports* constraint table

Figure 1 shows an example of a typical ACF generated by the FPGA Express software.

Figure 1. FPGA Express-Generated Assignment & Configuration File

```
CHIP my_chip
        DEVICE = EPF10K100GC503-3 {synopsys};
        "|_CONFIG" : PIN = P40 {synopsys};
        "|_STATUS" : PIN = P41 {synopsys};

GLOBAL_PROJECT_SYNTHESIS_ASSIGNMENT_OPTIONS
        DEVICE_FAMILY = FLEX10K {synopsys};
        STYLE = WYSIWYG {synopsys};
        OPTIMIZE_FOR_SPEED = 5 {synopsys};
        AUTO_GLOBAL_CLOCK = ON {synopsys};

LOGIC_OPTIONS
        "|TX_FIFOA_D6" : IO_CELL_REGISTER = ON {synopsys};
        "|DEST_RAM_D6" : SLOW_SLEW_RATE = ON {synopsys};
        "|DEST_RAM_D5" : SLOW_SLEW_RATE = OFF {synopsys};

COMPILER_INTERFACES_CONFIGURATION
        EDIF_INPUT_VCC = VDD {synopsys};
        EDIF_INPUT_GND = GND {synopsys};
        EDIF_INPUT_USE_LMF1 = ON {synopsys};
        EDIF_INPUT_LMF1 = "my_chip.lmf" {synopsys};

TIMING_POINT
        FREQUENCY = 50MHz {synopsys};
        "CLK80" : FREQUENCY = 80MHz {synopsys};
        "G" : FREQUENCY = 25MHz {synopsys};
```

```
TPD = 10ns {synopsys};
"inp1" : TSU = 20ns {synopsys};
"out1" : TCO = 15ns {synopsys};
```

☞ FPGA Express-generated ACFs show the Fast I/O logic option as `IO_CELL_REGISTER`. The MAX+PLUS II software automatically interprets this assignment as a `FAST_IO` assignment.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Using Synopsys FPGA Express & MAX+PLUS II Software

The following topics describe how to use the FPGA Express and MAX+PLUS® II software. Choose one of the following topics for more information:

This file is suitable for printing only. It does not contain hypertext links that allow you to jump from topic to topic.

Software Requirements

Design Flow for FPGA Express Software

Design Entry

- **Design Entry Flow**
- **VHDL**
    - Creating VHDL Designs for Use with MAX+PLUS II Software
        - Instantiating the clklock Megafunction in VHDL or Verilog HDL
        - Instantiating LPM Functions in VHDL
        - Instantiating RAM & ROM Functions in VHDL
- **Verilog HDL**
    - Creating Verilog HDL Designs for Use with MAX+PLUS II Software
        - Instantiating the clklock Megafunction in VHDL or Verilog HDL
        - Instantiating LPM Functions in Verilog HDL
        - Instantiating RAM & ROM Functions in Verilog HDL

Synthesis & Optimization

- Synthesizing & Optimizing VHDL or Verilog HDL Files with FPGA Express Software
- Entering Resource, Device & Global Logic Synthesis Assignments
    - Assigning a Device & Clock Frequency ($\mathbf{f_{MAX}}$)
    - Assigning Pins, Logic Options, and $\mathbf{t_{SU}}$, $\mathbf{t_{CO}}$ and $\mathbf{t_{PD}}$ Timing Constraints
    - Specifying the MAX+PLUS II Logic Synthesis Style with FPGA Express Software
    - Using ACFs Generated by FPGA Express Software
    - Modifying the Assignment & Configuration File with the **setacf** Utility
- Analyzing Estimated Timing with the FPGA Express Time Tracker
- Specifying Speed/Area & CPU Effort Settings with the FPGA Express Software

Compilation

- Project Compilation Flow
- Compiling Projects with MAX+PLUS II Software

Related Topics:

- Go to the following MAX+PLUS II ACCESS℠ Key topics for related information:
  - Programming Altera Devices
  - Using Synopsys Design Compiler or FPGA Compiler & MAX+PLUS II Software
  - Using Synopsys PrimeTime & MAX+PLUS II Software
  - Using Synopsys VSS & MAX+PLUS II Software
- Go to the following topics for additional information:
  - About the MAX+PLUS II Software
  - Altera Programming Hardware
  - Synopsys web site (http://www.synopsys.com)

MAX+PLUS II/FPGA Express Software Requirements

Table 1 shows the software applications that are used to generate, process, synthesize, and verify a project with MAX+PLUS® II and FPGA Express software:

Table 1. Software Requirements

Synopsys	Altera
FPGA Express version 2.1	MAX+PLUS II version 9.3 and higher

The FPGA Express software supports devices from all FLEX device families and the MAX 7000 and MAX 9000 device families.

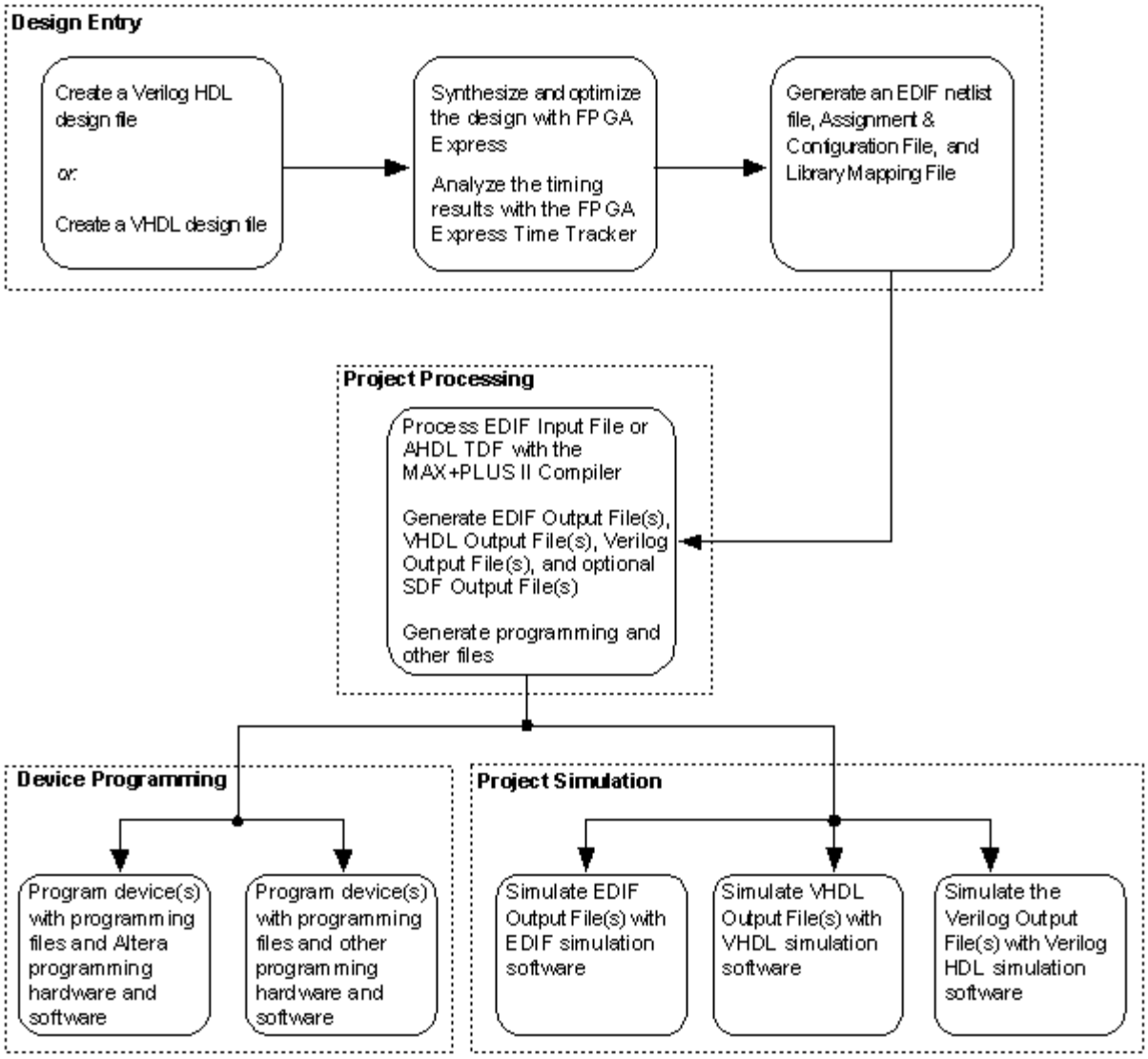
Related Topics:

- Go to the FPGA Express release notes for information on installing the FPGA Express software and a description of its latest changes. These notes are available in the **readme.htm** file on the FPGA Express installation CD-ROM.
- Go to the following topics for additional information:
  - FLEX Devices
  - MAX 7000 Devices
  - MAX 9000 Devices

Design Flow for FPGA Express Software

Figure 1 shows the design flow for the MAX+PLUS® II /FPGA Express interface.

Figure 1. MAX+PLUS II/FPGA Express Design Flow



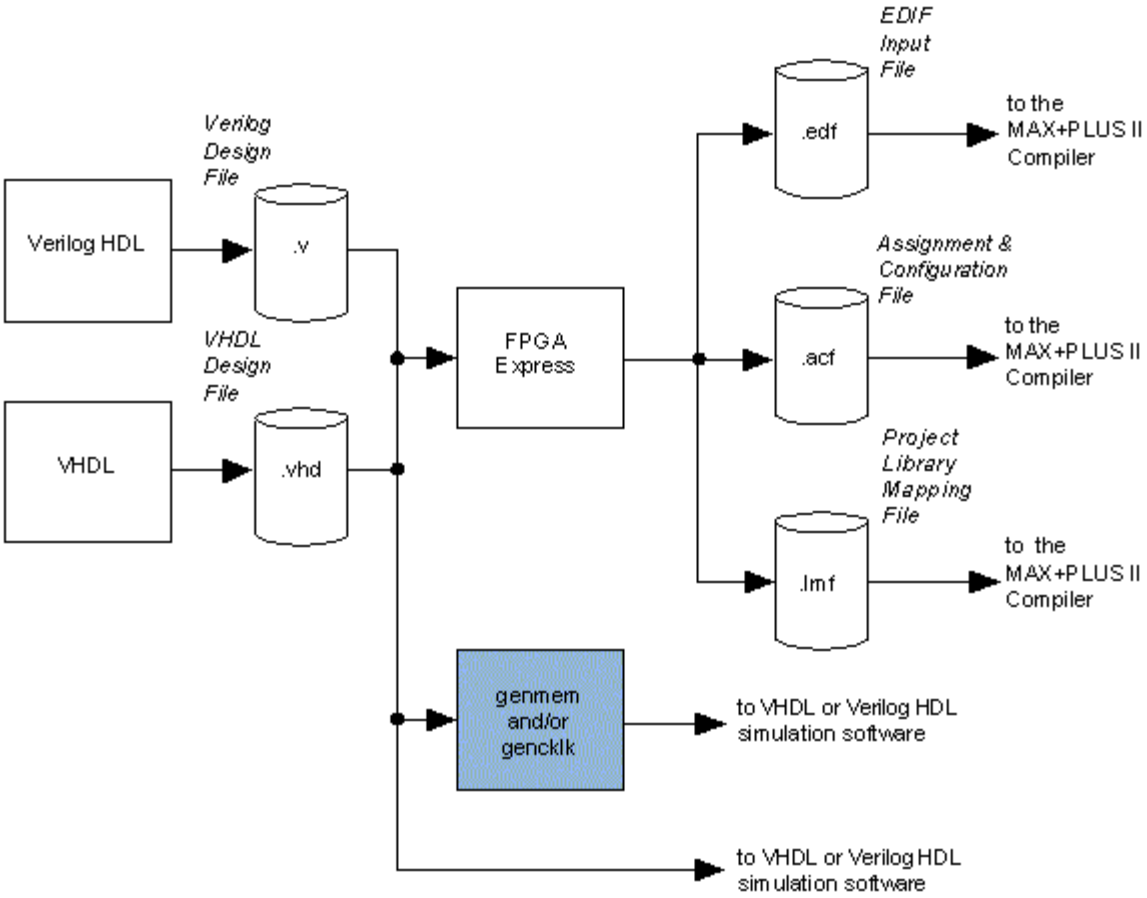
FPGA Express Design Entry Flow

Figure 1 shows the design entry flow for the MAX+PLUS® II /FPGA Express interface.

Figure 1. MAX+PLUS II/FPGA Express Design Entry Flow

☞ The MAX+PLUS II **read.me** file provides up-to-date information on which versions of Synopsys applications are supported by the current version of the MAX+PLUS II software. It also provides information on installation and operating requirements. You should read the **read.me** file on the ***MAX+PLUS II CD-ROM*** before installing the MAX+PLUS II software. After installation, you can open the **read.me** file from the MAX+PLUS II Help menu.

Altera-provided items are shown in blue.



Creating VHDL Designs for Use with MAX+PLUS II Software

You can create VHDL design files with the MAX+PLUS® II Text Editor, the FPGA Express internal text editor, or another standard text editor and save them in the appropriate directory for your project. The MAX+PLUS II and FPGA Express text editors offer different advantages; use either or both depending on your personal preferences:

-
- The MAX+PLUS II Text Editor offers VHDL templates with the **VHDL Templates** command (Templates menu) and syntax coloring with the **Syntax Coloring** command (Options menu).
- The FPGA Express internal text editor provides automatic error location when you double-click an error in the Output window.

To create a VHDL design that can be synthesized and optimized with the FPGA Express software, follow these steps:

1.
2. Describe your design using FPGA Express-supported VHDL constructs. For information on synthesizable VHDL constructs, refer to the online ***VHDL Reference Manual*** provided with the FPGA Express software. The following topics describe how to instantiate additional Altera-specific logic functions in your design:
    - Instantiating the clklock Megafunction in VHDL or Verilog HDL
    - Instantiating RAM & ROM Functions in VHDL

Instantiating LPM Functions in VHDL

You can instantiate MegaCore™ functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP™). The OpenCore™ feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.

3. Once you have created a design, synthesize and optimize it, as described in Synthesizing & Optimizing VHDL or Verilog HDL Files with FPGA Express Software.

Related Topics:

- Go to the following MAX+PLUS II ACCESS℠ Key topics for related information:
  - Using Synopsys VSS & MAX+PLUS II Software
  - Compiling Projects with MAX+PLUS II Software
- Go to the following topics for additional information:
  - Altera Megafunction Partners Program (AMPP)
  - Altera Megafunctions

Instantiating the clklock Megafunction in VHDL or Verilog HDL

MAX+PLUS® II interfaces to other EDA tools support the `clklock` phase-locked loop megafunction, which can be used with some FLEX® 10K devices, with the **gencklk** utility. Type `gencklk -h` ↵ at the DOS or UNIX prompt to display information on how to use this utility. The **gencklk** utility generates VHDL or Verilog HDL functional simulation models and a VHDL Component Declaration template file (**.cmp**).

The **gencklk** utility allows parameters for the `clklock` function to be passed from the VHDL or Verilog HDL file to EDIF netlist format. The **gencklk** utility embeds the parameter values in the `clklock` function name; therefore, the values do not need to be declared explicitly.

To instantiate the `clklock` megafunction in VHDL or Verilog HDL, go through the following steps:

1.

2. Type the following command at the DOS or UNIX prompt to generate the `clklock_x_y` function, where *x* is the ClockBoost™ value and *y* is the input frequency in MHz:

   ✔ Type `gencklk` <*ClockBoost*> <*input frequency*> `-vhdl` □ for VHDL designs.

   or:

   ✔ Type `gencklk` <*ClockBoost*> <*input frequency*> `-verilog` □ for Verilog HDL designs.

   Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu for more information on the `clklock` megafunction.

3. Create a design file that instantiates the `clklock_x_y`**.vhd** or `clklock_x_y`**.v** file. The **gencklk** utility automatically generates a VHDL Component Declaration template in the **clklock_*x_y*.cmp** file that you can incorporate into a VHDL design file.

□ In MAX+PLUS II version 8.3 and lower, running `genclklk` on a PC always creates files named as **clklock.vhd**,

clklock.cmp, and **clklock.v**, regardless of the ClockBoost and input frequency values you specify.

Figures 1 and 2 show a `clklock` function with <*ClockBoost*> = 2 and <*input frequency*> = 40 MHz instantiated in VHDL and Verilog HDL design files, respectively.

Figure 1. VHDL Design File with clklock Instantiation (count8.vhd)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY altera;
USE altera.maxplus2.all;     --  Include Altera Component Declarations



ENTITY count8 IS
   PORT (a    : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
         ldn  : IN STD_LOGIC;
         gn   : IN STD_LOGIC;



dnup : IN STD_LOGIC;
         setn : IN STD_LOGIC;
         clrn : IN STD_LOGIC;
         clk  : IN STD_LOGIC;



co   : OUT STD_LOGIC;
         q    : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END count8;



ARCHITECTURE structure OF count8 IS
   signal clk2x : STD_LOGIC;



COMPONENT clklock_2_40
   PORT (
      INCLK : IN STD_LOGIC;
      OUTCLK : OUT STD_LOGIC
   );
END COMPONENT;



BEGIN
   u1: clklock_2_40
   PORT MAP (inclk=>clk, outclk=>clk2x);



u2: a_8count
   PORT MAP (a=>a(0), b=>a(1), c=>a(2), d=>a(3),
             e=>a(4), f=>a(5), g=>a(6), h=>a(7),
             clk=>clk2x,
             ldn=>ldn,
             gn=>gn,



dnup=>dnup,
             setn=>setn,
             clrn=>clrn,



qa=>q(0), qb=>q(1), qc=>q(2), qd=>q(3),
             qe=>q(4), qf=>q(5), qg=>q(6), qh=>q(7),
             cout=>co);
 END structure;
```

Figure 2. Verilog HDL Design File with clklock Instantiation (count8.v)

```
`timescale 1ns / 10ps
module count8 (a, ldn, gn, dnup, setn, clrn, clk, co, q);
output          co;
output[7:0]     q;



input[7:0]      a;
input           ldn, gn,dnup, setn, clrn, clk;
wire            clk2x;



clklock_2_40 u1 (.inclk(clk), .outclk(clk2x) );
A_8COUNT u2 (.A(a[0]), .B(a[1]), .C(a[2]), .D(a[3]), .E(a[4]), .F(a[5]),



.G(a[6]), .H(a[7]), .LDN(ldn), .GN(gn), .DNUP(dnup),
             .SETN(setn), .CLRN(clrn), .CLK(clk2x), .QA(q[0]), .QB(q[1]),
             .QC(q[2]), .QD(q[3]), .QE(q[4]), .QF(q[5]), .QG(q[6]),
             .QH(q[7]), .COUT(co) );



endmodule
```

Related Topics:

- Go to FLEX 10K Device Family, which is available on the web, for additional information.

Instantiating LPM Functions in VHDL

You can enter library of parameterized modules (LPM) functions in your VHDL design. The MAX+PLUS® II software supports all LPM functions except the truth table, finite state machine, and pad functions. The FPGA Express software supports all LPM functions that are supported in the MAX+PLUS II software except the `lpm_and`, `lpm_or`, `lpm_xor`, and `lpm_mux` functions. Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu for detailed information on all LPM functions.

To instantiate an LPM function in a VHDL design, follow these steps:

1.

2. Create an instance of an LPM function in VHDL with a Component Instantiation Statement. VHDL Component Declarations for LPM functions are available in MAX+PLUS II Help and also installed automatically in the following FPGA Express directory:

   <*drive*>:Â¥synopsysÂ¥fpga_expressÂ¥libÂ¥packagesÂ¥lpmÂ¥lpm_components.vhd

   Use named association to specify parameter values in the Generic Map Clauses of LPM function instantiations. Figure 1 shows the Component Declaration for the `lpm_ram_dq` function.

   Figure 1. Component Declaration for lpm_ram_dq (from lpm_components.vhd)

   ```
   COMPONENT lpm_ram_dq
      GENERIC (
                LPM WIDTH: POSITIVE;
   ```

```
            LPM_TYPE : STRING := L_RAM_DQ;
            LPM_WIDTHAD: POSITIVE;
            LPM_NUMWORDS: STRING := UNUSED;
            LPM_FILE: STRING := UNUSED;
            LPM_INDATA: STRING := REGISTERED;
            LPM_ADDRESS_CONTROL : STRING := REGISTERED;
            LPM_OUTDATA: STRING := REGISTERED
            );
   PORT     (
            data     : IN STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0);
            we       : IN STD_LOGIC := '1';
            inclock  : IN STD_LOGIC := '1';
            outclock : IN STD_LOGIC := '1';
            address  : IN STD_LOGIC_VECTOR(LPM_WIDTHAD-1 DOWNTO 0);
            q        : OUT STD_LOGIC_VECTOR (LPM_WIDTH-1 DOWNTO 0)
           );
END COMPONENT;
```

3. Manually tie any pins that require an initial value of logic 1 to `vcc`. The FPGA Express software does not support initial values of logic 1 in Component Declarations. However, it does support initial values of logic 0.

   Figure 2 shows an example of instantiating an `lpm_ram_dq` function in VHDL.

   Figure 2. VHDL Design File with lpm_ram_dq Instantiation

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY lpm;
USE lpm.lpm_components.all;

ENTITY design IS
   PORT(
        din  : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
        we   : IN STD_LOGIC;
        clk  : IN STD_LOGIC;
        addr : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        dat  : OUT STD_LOGIC_VECTOR(15 DOWNTO 0)
        );
END design;

ARCHITECTURE struct OF design IS
   SIGNAL vcc : STD_LOGIC;

BEGIN  --  struct
   vcc <= '1';

u1:  lpm_ram_dq
   GENERIC MAP(
               LPM_WIDTH   => 16,
               LPM_WIDTHAD => 4,
               LPM_INDATA  => "UNREGISTERED",
               LPM_OUTDATA => "UNREGISTERED"
              )

   PORT MAP(
            data     => din,
            address  => addr,
            we       => we,
            q        => dat,
            inclock  => vcc,
            outclock => clk
            );
END struct;
```

4. Continue with the steps necessary to complete your VHDL design, as described in Creating VHDL Designs for Use with MAX+PLUS II Software.

Instantiating RAM & ROM Functions in VHDL

The MAX+PLUS® II /Synopsys interface offers full support for the memory capabilities of the FLEX®10K device family, including synchronous and asynchronous RAM and ROM, cycle-shared dual port RAM, dual-port RAM, single-Clock FIFO, and dual-clock FIFO functions. Altera recommends using the LPM functions `lpm_ram_dq`, `lpm_ram_io`, and `lpm_rom` to instantiate synchronous and asynchronous RAM and ROM. However, if you wish to enter cycle-shared dual port RAM (`csdpram`), dual-port RAM (`altdpram`), single-Clock FIFO (`scfifo`), and dual-clock FIFO (`dcfifo`) functions, or if you wish to create simulation models for any supported RAM or ROM function, you can use the Altera-provided **genmem** utility. Instantiations created with **genmem** for use with other Synopsys products, such as FPGA Compiler or Design Compiler, are supported for backward compatibility. Type `genmem` ↵ at the DOS or UNIX prompt to display information on how to use this utility, as well as a list of the functions you can generate.

To instantiate an `lpm_ram_dq`, `lpm_ram_io`, or `lpm_rom` function:

✔ Follow the guidelines in Instantiating LPM Functions in VHDL.

To instantiate other RAM and ROM functions in VHDL, follow these steps:

1.

2. Use the **genmem** utility to generate a memory model by typing the following command at the DOS or UNIX prompt:

   `genmem` <*memory type*> <*memory size*> `-vhdl` ↵

   For example: `genmem scfifo 16x8 -vhdl` ↵

3. Create a VHDL design that incorporates the text from the **genmem**-generated Component Declaration, ***<memory name>*.cmp**, and instantiate the ***<memory name>*.vhd** function. The **genmem** utility produces files with descriptive names that typically include both the memory type and the memory size (e.g., **scfifo_16x8_d.vhd**).

   ☞ In MAX+PLUS II version 8.3 and lower, running `genmem` on a PC always creates files named as **genmem.vhd**, **genmem.cmp**, and **genmem.v**, regardless of the memory type and memory size you specify.

1. (Optional for RAM functions) Specify an initial memory content file:

   -

   - For ROM functions, you must specify the filename of an initial memory content file in the Intel hexadecimal format (**.hex**) or the Altera® Memory Initialization File (**.mif**) format in the Generic Map Clause, with the `LPM_FILE` parameter. The filename must be the same as the instance name; e.g., the `u1` instance name shown in Figure 1 must be unique throughout the whole project, and must contain only valid VHDL name characters. The initialization file must reside in the directory containing the project's design files.

   - For RAM functions, specifying memory initialization file is optional. If you want to use it, you must specify it in the Generic Map Clause as described above. If you do not use an initialization file, you should not declare or use the Generic Clause.

     1.

     2. The MIF format is supported only for specifying initial memory content when compiling designs within MAX+PLUS II software. You cannot use an MIF to perform simulation with Synopsys tools prior to MAX+PLUS II compilation.

   ☞

3. If you use an Intel hexadecimal format file and wish to simulate the file with the VHDL System Simulator (VSS) after MAX+PLUS II compilation, you should use the Synopsys **intelhex** utility to translate the Intel hexadecimal fomat file into a VSS-compatible Synopsys memory file. Refer to the Synopsys ***VHDL System Simulator Software Tool*** manual for details about using the **intelhex** utility.

2. Continue with the steps necessary to complete your VHDL design, as described in Creating VHDL Designs for Use with MAX+PLUS II Software.

Related Topics:

- Go to Using Synopsys VSS & MAX+PLUS II Software in these MAX+PLUS II ACCESS℠ Key topics for related information.
- Go to FLEX 10K Device Family, which is available on the web, for additional information.

Creating Verilog HDL Designs for Use with MAX+PLUS II Software

You can create Verilog HDL design files with the MAX+PLUS® II Text Editor, the FPGA Express internal text editor, or another standard text editor and save them in the appropriate directory for your project. The MAX+PLUS II and FPGA Express text editors offer different advantages; use either or both depending on your personal preferences:

-
- The MAX+PLUS II Text Editor offers Verilog HDL templates with the **Verilog HDL Templates** command (Templates menu) and syntax coloring with the **Syntax Coloring** command (Options menu).
- The FPGA Express internal text editor provides automatic error location when you double-click an error in the Output window.

To create a Verilog HDL design that can be synthesized and optimized with the FPGA Express software, follow these steps:

1.
2. Describe your design using FPGA Express-supported Verilog HDL constructs. For information on synthesizable Verilog HDL constructs, refer to the online ***HDL Reference Manual*** provided with the FPGA Express software. The following topics describe how to instantiate additional Altera-specific logic functions in your design:

    - Instantiating the clklock Megafunction in VHDL or Verilog HDL
    - Instantiating RAM & ROM Functions in Verilog HDL
    - Instantiating LPM Functions in Verilog HDL

    You can instantiate MegaCore™ functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP™). The OpenCore™ feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.

3. Once you have created a design, synthesize and optimize it, as described in Synthesizing & Optimizing VHDL or Verilog HDL Files with FPGA Express Software.

Related Topics:

- Go to Compiling Projects with MAX+PLUS II Software in these MAX+PLUS II ACCESS℠ Key topics for related information.
- Go to the following topics for additional information:
  - Altera Megafunction Partners Program (AMPP)
  - Altera Megafunctions

Instantiating LPM Functions in Verilog HDL

You can enter library of parameterized modules (LPM) functions in your Verilog HDL design. The MAX+PLUS® II software supports all LPM functions except the truth table, finite state machine, and pad functions. The FPGA Express software supports all LPM functions that are supported in the MAX+PLUS II software except the `lpm_and`, `lpm_or`, `lpm_xor`, and `lpm_mux` functions. Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu for detailed information on all LPM functions.

To instantiate an LPM function in a Verilog HDL design, follow these steps:

1.

2. Use a Module Instantiation to instantiate an LPM function. You must associate all parameters, and only positional association is allowed.

   Figure 1 shows an example of instantiating an `lpm_ram_dq` function in Verilog HDL.

   Figure 1. Verilog HDL Design File with lpm_ram_dq Instantiation

```
// RAM design

module design(din, we, clk, addr, dat);

input [15:0] din;
input we, clk;
input [3:0] addr;
output [15:0] dat;

supply1 vcc;

lpm_ram_dq #(
16,          // LPM_WIDTH
"LPM_RAM_DQ",        // LPM_TYPE
4,           // LPM_WIDTHAD
16,          // LPM_NUMWORDS
"UNUSED",        // LPM_FILE
"UNREGISTERED",      // LPM_INDATA
"UNREGISTERED",      // LPM_ADDRESS_CONTROL
"UNREGISTERED"       // LPM_OUTDATA
)

u1(
.data(din),
.address(addr),
.we(we),
.q(dat),
.inclock(vcc),
.outclock(clk)
);

endmodule
```

3. Continue with the steps necessary to complete your Verilog HDL design file, as described in Creating Verilog HDL Designs for Use with MAX+PLUS II Software.

Instantiating RAM & ROM Functions in Verilog HDL

The MAX+PLUS® II /Synopsys interface offers full support for the memory capabilities of the FLEX® 10K device family, including synchronous and asynchronous RAM and ROM, cycle-shared dual port RAM, dual-port RAM, single-Clock FIFO, and dual-clock FIFO functions. Altera recommends using the LPM functions `lpm_ram_dq`, `lpm_ram_io`, and `lpm_rom` to instantiate synchronous and asynchronous RAM and ROM. However, if you wish to enter cycle-shared dual port ram (`csdpram`), dual-port RAM (`altdpram`), single-Clock FIFO (`scfifo`), and dual-clock FIFO (`dcfifo`) functions, or if you wish create simulation models for any supported RAM or ROM function, you can use the Altera-provided **genmem** utility. Designs that instantiate genmem-generated synchronous and asynchronous RAM and ROM -- such as those used with FPGA Compiler or Design Compiler -- are supported for backward compatibility. Type `genmem` ↵ at the DOS or UNIX prompt to display information on how to use this utility, as well as a list of the functions you can generate.

To instantiate an `lpm_ram_dq`, `lpm_ram_io`, or `lpm_rom` function:

✔ Follow the guidelines in Instantiating LPM Functions in Verilog HDL.

To instantiate other RAM and ROM functions in Verilog HDL, follow these steps:

1.

2. Use the **genmem** utility to generate a memory model by typing the following command at a DOS or UNIX prompt:

   `genmem` <*memory type*> <*memory size*> `-verilog` ↵

   For example: `genmem scfifo 16x8 -verilog` ↵

3. Create a Verilog HDL design that instantiates the ***<memory name>*.v** function. The **genmem** utility produces files with descriptive names that typically include both the memory type and the memory size (e.g., **scfifo_16x8_d.v**).

   ☞ In MAX+PLUS II version 8.3 and lower, running `genmem` on a PC always creates files named as **genmem.vhd**, **genmem.cmp**, and **genmem.v**, regardless of the memory type and memory size values you specify.

1.

2. (Optional for RAM functions) Specify an initial memory content file:

   -

   - For ROM functions, you must specify the filename of an initial memory content file in the Intel hexadecimal format (**.hex**) or the Altera® Memory Initialization File (**.mif**) format in the Parameter Statement, using the `LPM_FILE` parameter. The filename must be the same as the instance name; e.g., the `u1` instance name must be unique throughout the whole project, and must contain only valid Verilog HDL name characters. The initialization file must reside in the directory containing the project's design files.

   - For RAM functions, specifying a memory initialization file is optional. If you want to use it, you must specify it in the Parameter Statement as described above.

1. The MIF format is supported only for specifying initial memory content when compiling designs within MAX+PLUS II software. You cannot use an MIF to perform simulation with Synopsys tools prior to MAX+PLUS II compilation.

☞

2. If you use an Intel hexadecimal format file and wish to simulate the file with the VHDL System Simulator Software (VSS) after MAX+PLUS II compilation, you should use the Synopsys **intelhex** utility to translate the Intel hexadecimal fomat file into a VSS-compatible Synopsys memory file. Refer to the Synopsys ***VHDL System Simulator Software Tool*** manual for details about using the **intelhex** utility.

1.

2. Continue with the steps necessary to complete your Verilog HDL design, as described in Creating Verilog HDL Designs for Use with MAX+PLUS II Software.

Related Topics:

- Go to FLEX 10K Devices, which is available on the web, for additional information.

Synthesizing & Optimizing VHDL or Verilog HDL Files with FPGA Express Software

You can analyze, synthesize, and optimize design files using the FPGA Express software, then convert them to EDIF netlist files that can be processed by the MAX+PLUS® II software.

To process a VHDL or Verilog HDL design for use with MAX+PLUS II software, follow these steps:

1.

2. Create a VHDL file, <*design name*>**.vhd**, or Verilog HDL file, <*design name*>**.v**, using the MAX+PLUS II Text Editor or another standard text editor and save it in your working directory. Go to Creating VHDL Designs for Use with MAX+PLUS II Software or Creating Verilog HDL Designs for Use with MAX+PLUS II Software for more information on VHDL or Verilog HDL design entry.

3. Start the FPGA Express software. Select *Create a new project* in the **Startup** dialog box and choose **OK**. The **Create Project Folder** dialog box is displayed. You can also view the **Create Project Folder** dialog box by choosing **New** (File menu).

4. Specify the full file and path name of the project in the **Create Project Folder** dialog box and choose *Create*. The FPGA Express software creates the project and opens the **Identify Source File** dialog box.

5. Identify and analyze the source files for the project by selecting them in the **Identify Source File** dialog box and choosing **Add**. The FPGA Express internal text editor automatically analyzes each source file as it appears on the left-hand side of the Project window. A green checkmark appears to the left of each filename for the files that have no errors or warnings; a red cross appears for files with errors; and an exclamation point appears for files with warnings.

6. Select the source file icon to display any errors or warnings in the Output window. To fix an error, double-click on the error. The FPGA Express internal text editor automatically displays the source file and highlights the line containing the error or warning in red. To view help on the error or warning, double-click on the error or warning code number (shown in parentheses) in the Output window.

   ☞ FPGA Express software does not copy source files; it identifies and analyzes them in their current location. Refer to FPGA Express Help for more information.

7. Specify the MAX+PLUS II logic synthesis style. Refer to Specifying the MAX+PLUS II Logic Synthesis Style with FPGA Express software for more information.

8. From the Project window, identify the top-level design for your project. Select the top-level design from the

Top-Level Design drop-down list on the toolbar. The **Create Implementation** dialog box is displayed.

9. In the **Create Implementation** dialog box, specify the following options:
    1.
    2. Assign a device and the Clock frequency. Refer to Assigning a Device & Clock Frequency ($\mathbf{f_{MAX}}$) for more information.
    3. Select a global optimization goal (*speed* or *area*) and a CPU effort designation (*high* or *low*). Refer to Specifying the Speed/Area & CPU Effort Settings with the FPGA Express Software for information.
    4. Close the **Create Implementation** dialog box by choosing **OK**.

    The FPGA Express software processes each source file and determines the complete hierarchical structure and topology of the design, including multi-level links and references between subdesigns. With this information, the FPGA Express software produces an intermediate, unoptimized design implementation. The right-hand side of the Project window displays the implementation name and target device. The implementation icon also indicates any errors, warnings, or other information. To correct error or warning conditions, refer to step 5.

10. (Optional) Select the design implementation icon in the Chips window, press Button 2, and choose the **Edit Constraints** command from the pop-up menu to display the Altera-specific constraint tables. These constraint tables allow you to specify pin, logic option, and timing assignments for your design. All design-specific information, such as Clock names, port names, and design hierarchy assignments is extracted automatically from the design. Altera recommends entering specific requirements directly into these tables to obtain the desired optimization. Refer to Entering Resource, Device & Global Logic Synthesis Assignments for information.

11. Optimize the design by selecting the design implementation in the Project window and choosing the **Optimize** button on the toolbar. A new optimized implementation icon appears beneath the original implementation icon. When you open the optimized implementation, the constraint tables are back-annotated with the optimization results. The FPGA Express software optimizes a design for either speed or area, based on the settings you specified in step 8.

12. Identify and optimize critical paths in your design with the Time Tracker static timing analyzer, as described in Analyzing Estimated Timing with the FPGA Express Time Tracker.

13. Generate a project report by selecting the optimized design implementation and clicking the **Report** icon on the toolbar. An FPGA Express project report documents the design through the synthesis and optimization design flow. The report includes information about design source data, constraints, and optimization options.

14. Generate MAX+PLUS II-compatible EDIF netlist files by selecting the optimized design implementation and choosing the **Export Netlist** button on the toolbar. In the **Export** dialog box, specify the following options:
    1.
    2. Specify the name and location of the directory for the EDIF netlist files in the *Export Directory* box.
    3. Select the EDIF netlist file's output bus from the *Bus Style* drop-down list. The MAX+PLUS II software accepts either flattened or unflattened buses. In the FPGA Express software, the default setting, *EXPAND*, flattens each bus by writing each bus bit as an individual I/O port. To export an EDIF netlist file without flattening the bus names, select any of the other settings, which include delimiters for different bus notations: `[]`, <>, `()`, and `{}`.
    4. If you wish to generate a VHDL or Verilog HDL netlist file for functional simulation prior to MAX+PLUS II compilation, select a language option (*VHDL* or *Verilog*) from the *Output Format* drop-down list. Otherwise, select *NONE* for this option instead.

5. Turn on the *Export Primitives* option to export VHDL or Verilog HDL primitives into the simulation netlist file. However, if the simulation is to be performed with an external library, turn the option off.

6. Close the **Export** dialog box by choosing **OK**. The FPGA Express software creates the following MAX+PLUS II-compatible files:

    - <*design name*>**.edf** (EDIF format)
    - <*design name*>**.acf**, an Assignment & Configuration File that contains design constraints
    - <*design name*>**.lmf**, a Library Mapping File that maps FPGA Express functions to MAX+PLUS II functions

15. Copy all three types of output files (EDIF netlist file(s), ACF, and LMF) to a MAX+PLUS II project directory. Process the <*design name*>**.edf** file with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Entering Resource, Device & Global Logic Synthesis Assignments

The MAX+PLUS® II software allows you to enter a variety of resource, device, and global logic synthesis assignments for your projects. Resource assignments are used to assign logic functions to a particular pin, logic cell, I/O cell, embedded cell, row, column, Logic Array Block (LAB), Embedded Array Block (EAB), chip, clique, local routing, logic option, timing requirement, or connected pin group. With MAX+PLUS II software, you can enter all types of resource, device, and global logic synthesis assignments with Assign menu commands. You can also enter pin, logic cell, I/O cell, embedded cell, LAB, EAB, row, and column assignments in the MAX+PLUS II Floorplan Editor. The Assign menu commands and the Floorplan Editor all save assignment information in the ASCII Assignment & Configuration File (**.acf**) for the project.

In the Synopsys FPGA Express software, you can assign a limited subset of these assignments in the **Create Implementation** dialog box, in the **Options** dialog box, and by specifying options in constraint tables. These attributes are incorporated into the ACF generated by the FPGA Express software. Refer to the following topics for more information:

- Assigning a Device & Clock Frequency ($\mathbf{f_{MAX}}$)
- Assigning Pins, Logic Options, and $\mathbf{t_{SU}}$, $\mathbf{t_{CO}}$ & $\mathbf{t_{PD}}$ Timing Constraints
- Specifying the MAX+PLUS II Logic Synthesis Style with FPGA Express Software
- Using ACFs Generated by FPGA Express Software
- Modifying the Assignment & Configuration File with the **setacf** utility

Related Topics:

- Go to the following sources for related information:
    - FPGA Express Help
    - "resource assignments" or "ACF, format" in MAX+PLUS II Help
    - Specifying the Speed/Area & CPU Effort Settings with the FPGA Express Software in these MAX+PLUS II ACCESS℠ Key topics

---

Assigning a Device & Clock Frequency ($\mathbf{f_{MAX}}$)

You can specify the desired Clock frequency (called $\mathbf{f_{MAX}}$ in the MAX+PLUS® II software) and the target device family before synthesizing and optimizing the design with the FPGA Express software. You can optionally select a specific device and speed grade within the target device family. These assignments are stored in the design's

Assignment & Configuration File, <*design name*>**.acf**, which is generated automatically by the FPGA Express software.

To assign a device or device family and the Clock frequency, follow these steps:

1.
2. If you have not already done so, identify the top-level design for your project from the Design Sources window. Select the top-level design from the *Top-Level Design* drop-down list on the toolbar. The **Create Implementation** dialog box is displayed.
3. Enter an implementation name in the *Implementation Name* box. If you do not enter a name, FPGA Express software automatically creates a unique implementation name.
4. Select *Altera* from the *Vendor* list.
5. Select the appropriate Altera device family from the *Family* list.
6. (Optional) Select a specific device from the *Device* list, and select a specific speed grade from the *Speed Grade* list.
7. Type the desired Clock frequency in the *Clock Frequency* text box. This Clock frequency is used as the default value for all Clock signals in the design.
8. (Optional) Select speed/area and CPU effort settings, as described in Specifying the Speed/Area & CPU Effort Settings with the FPGA Express Software.
9. Continue with the steps necessary to process your design, as described in Synthesizing & Optimizing VHDL or Verilog HDL Files with FPGA Express Software.

You can also edit the Clock frequency by double-clicking on the design implementation name to open the constraint tables and entering information on the **Clock** tab. For more information, refer to Assigning Pins, Logic Options, and $\mathbf{t_{SU}}$, $\mathbf{t_{CO}}$ & $\mathbf{t_{PD}}$ Timing Constraints.

Related Topics:

- Go to the following sources for related information:
  - FPGA Express Help
  - "Guidelines for Achieving Maximum Speed Performance" in MAX+PLUS II Help

---

Assigning Pins, Logic Options, and t_{SU}, t_{CO} & t_{PD} Timing Constraints

You can assign pins, logic options, or timing constraints to your design in FPGA Express constraint tables. Some design-specific information is extracted automatically from your design and displayed in the constraint tables; you can also manually enter specific assignments in these tables. The FPGA Express software saves the assignments to an Assignment & Configuration File (**.acf**) when it synthesizes and optimizes the design. The MAX+PLUS® II software uses the assignment information from the ACF when it processes the design. Refer to Using ACFs Generated by FPGA Express Software for more information.

To enter resource assignments in FPGA Express software, follow these steps:

1.
2. Select the design implementation icon in the Chips window, press Button 2, and choose the **Edit Constraints**

commmand from the pop-up menu to display the Altera-specific constraint tables. These tables allow you to specify resource assignments for your design. All design-specific information such as Clock names, port names, and design hierarchy is extracted automatically from the design. Altera recommends entering specific requirements directly into these tables to obtain the desired optimization. For information on creating a design implementation, refer to steps 1 through 8 in Synthesizing & Optimizing VHDL or Verilog HDL Files with FPGA Express Software.

3. Enter assignments in the appropriate constraint tables. You can click on a tab to toggle between tables. Refer to Table 1, which shows the available MAX+PLUS II resource assignment options in the FPGA Express constraint tables. The **Clock** and **Path** tables already contain information that you previously entered in the **Create Implementation** dialog box. Refer to Assigning a Device & Clock Frequency ($\mathbf{f_{MAX}}$) for more information.

Table 1. MAX+PLUS II Resource Assignments in FPGA Express Constraint Tables

MAX+PLUS II Resource Assignment	Equivalent FPGA Express Constraint Table Setting	
	Tab Name	**Action**
Pin assignment	**Ports**	Specify the pin number in the *Pad Location* column.
$\mathbf{t_{SU}}$ timing assignment	**Ports**	Specify the time in the *Input Delay* column.
$\mathbf{t_{CO}}$ timing assignment	**Ports**	Specify the time in the *Output Delay* column.
Slow Slew Rate logic option assignment	**Ports**	Click on the appropriate cell in the *Slew Rate* column and select <*default*>, *FAST*, or *SLOW* from the list.
Fast I/O logic option assignment	**Ports**	Click on the appropriate cell in the *Use I/O Reg* column and select <*default*>, *ON*, or *OFF* from the list.
$\mathbf{t_{PD}}$ timing assignment	**Paths**	Specify the time in the *Delay* column.

4. Choose **Save** and then **Close** to exit from the FPGA Express constraint tables.

Related Topics:

- Go to the following sources for related information:
  - FPGA Express Help
  - "Guidelines for Achieving Maximum Speed Performance" in MAX+PLUS II Help
  - Specifying the MAX+PLUS II Logic Synthesis Style with the FPGA Express Software in these MAX+PLUS II ACCESS℠ Key topics

---

Specifying the MAX+PLUS II Logic Synthesis Style with FPGA Express Software

You can specify the MAX+PLUS® II global project logic synthesis style for FLEX® devices from within the FPGA Express software. To specify the global project logic synthesis style, follow these steps:

1. Choose **Options** (Synthesis menu) to display the **Options** dialog box.
2. Choose the **Behavior** tab.
3. Turn the *Insert LCELL Buffers, Style WYSIWYG (Altera FLEX Only)* option on or off:
   -
   - Turning this option on is the equivalent of specifying the WYSIWYG (What You See Is What You Get) logic synthesis style in the MAX+PLUS II software. The WYSIWYG style directs the Compiler's Logic Synthesizer module to change the logic in your project as little as possible during compilation. The WYSIWYG style avoids removing or inserting additional logic, and turns off many logic options that might help the project to fit. In addition, if this option is turned on, the FPGA Express software

inserts LCELL buffers for look-up table (LUT) outputs so that the MAX+PLUS II software will not alter the logic cell implementations. This option is recommended when a design's area optimization has priority over its speed.

- Turning this option off is the equivalent of specifying the Fast logic synthesis style in the MAX+PLUS II software. The Fast style directs the Compiler's Logic Synthesizer module to optimize your project for maximum speed, rather than for minimum silicon usage. In addition, if this option is turned off, the FPGA Express software does not insert LCELL buffers, thereby allowing the MAX+PLUS II software to optimize the LUT logic to improve performance.

4. Choose **OK**.

Related Topics:

- Go to the following topics for additional information:
  - FLEX Devices
  - MAX® Devices
  - Classic Device Family

Using ACFs Generated by FPGA Express Software

With FPGA Express software, you can either generate a new Assignment & Configuration File (**.acf**), along with an EDIF netlist file (**.edf**) and Library Mapping File (**.lmf**), to be imported into the MAX+PLUS® II software, or you can place a copy of an existing ACF in the FPGA Express output directory. If you use an existing ACF, FPGA Express updates the ACF with additional information, such as the global project synthesis style, as it processes the design. Each line in the ACF that is modified by the FPGA Express software is marked with a `{synopsys}` comment at the end. You should then place this ACF in the MAX+PLUS II project directory.

If an existing ACF has been modified by FPGA Express, then processed by the MAX+PLUS II Compiler, the resulting ACF may specify an incorrect LMF. If so, the MAX+PLUS II software displays the error message `Can't find design file` <*cell name*>. You can correct this error in MAX+PLUS II software by specifying the FPGA Express-generated <*project name*>**.lmf** file in the *LMF #1* box in the **EDIF Netlist Reader Settings** dialog box. See Compiling Projects with MAX+PLUS II Software for more information.

The ACF incorporates the following assignments from FPGA Express software and passes them to the MAX+PLUS II software:

- The Altera device family and optional specific device and speed grade specified in the **Create Implementation** dialog box
- The global project logic synthesis style specified in the **Create Implementation** dialog box
- The Clock frequency (**f_{MAX}**) specified in the **Create Implementation** dialog box or the *Clocks* constraint table
- Clock speeds (**t_{PD}**) specified in the *Clocks* constraint table
- Path group constraints specified in the *Paths* constraint table
- Pin assignments; settings for the Slow Slew Rate and Fast I/O logic options; and input-to-setup (**t_{SU}**) and clock-to-output (**t_{CO}**) delays specified in the *Ports* constraint table

Figure 1 shows an example of a typical ACF generated by the FPGA Express software.

Figure 1. FPGA Express-Generated Assignment & Configuration File

```
CHIP my_chip
    DEVICE = EPF10K100GC503-3 {synopsys};
    "|_CONFIG" : PIN = P40 {synopsys};
    "|_STATUS" : PIN = P41 {synopsys};
```

```
GLOBAL_PROJECT_SYNTHESIS_ASSIGNMENT_OPTIONS
    DEVICE_FAMILY = FLEX10K {synopsys};
    STYLE = WYSIWYG {synopsys};
    OPTIMIZE_FOR_SPEED = 5 {synopsys};
    AUTO_GLOBAL_CLOCK = ON {synopsys};

LOGIC_OPTIONS
    "|TX_FIFOA_D6" : IO_CELL_REGISTER = ON {synopsys};
    "|DEST_RAM_D6" : SLOW_SLEW_RATE = ON {synopsys};
    "|DEST_RAM_D5" : SLOW_SLEW_RATE = OFF {synopsys};

COMPILER_INTERFACES_CONFIGURATION
    EDIF_INPUT_VCC = VDD {synopsys};
    EDIF_INPUT_GND = GND {synopsys};
    EDIF_INPUT_USE_LMF1 = ON {synopsys};
    EDIF_INPUT_LMF1 = "my_chip.lmf" {synopsys};

TIMING_POINT
    FREQUENCY = 50MHz {synopsys};
    "CLK80" : FREQUENCY = 80MHz {synopsys};
    "G" : FREQUENCY = 25MHz {synopsys};
    TPD = 10ns {synopsys};
    "inp1" : TSU = 20ns {synopsys};
    "out1" : TCO = 15ns {synopsys};
```

☞ FPGA Express-generated ACFs show the Fast I/O logic option as `IO_CELL_REGISTER`. The MAX+PLUS II software automatically interprets this assignment as a `FAST_IO` assignment.

Modifying the Assignment & Configuration File with the setacf Utility

Altera provides the **setacf** utility to help you modify a project's Assignment & Configuration File (**.acf**) from the command line, without opening the file with a text editor. Type `setacf -h` ↵ at a UNIX or DOS prompt to get help on this utility.

Analyzing Estimated Timing with the FPGA Express Time Tracker

You can use the FPGA Express Time Tracker static timing analyzer to display estimated delays of critical paths in your project. This timing analyzer provides timing information and a detailed listing of critical paths.

To use the Time Tracker timing analyzer, follow these steps:

1.
2. Select the design implementation icon in the Chips window, press Button 2, and choose the **View Results** commmand from the pop-up menu to display the Time Tracker tabs.
3. Analyze the timing of your design by viewing the different tables within the **Clocks**, **Paths**, and **Ports** Time Tracker tabs:
    -
    - To analyze the Clock frequency ($\mathbf{f_{MAX}}$), select the **Clocks** tab. The table on the **Clocks** tab contains a column showing the actual Clock frequency for each Clock in your design next to the desired frequency derived from your timing constraints. Clocks that fail to meet their constraints are highlighted in red.
    - To check critical timing paths, select the **Paths** tab. The table on the **Paths** tab contains an **Est. Delay** column displaying path delays. Paths that fail to meet constraints are highlighted in red. You can select a path or path group to display additional tables with increasing detail, in order to identify exactly

which paths failed to meet their timing constraints.

- To view I/O port delays, select the **Ports** tab. The **Ports** tab displays the slack for each I/O port, i.e., the Clock period minus the propagation delay through the port in the *Input Slack* column for input ports and the *Output Slack* column for output ports. Negative values are highlighted in red, indicating that the propagation delay exceeds the Clock period, causing a timing violation.

4. If necessary, change the design logic or adjust your timing constraints as described in Assigning Pins, Logic Options, and $\mathbf{t_{SU}}$, $\mathbf{t_{CO}}$ & $\mathbf{t_{PD}}$ Timing Constraints, then re-optimize the design.

5. Continue with the steps necessary to process your design, as described in Synthesizing & Optimizing VHDL or Verilog HDL Files with FPGA Express Software.

Specifying Speed/Area & CPU Effort Settings with the FPGA Express Software

FPGA Express software allows you to choose either speed or area options and to specify either high or low CPU effort in logic optimization. Optimization goals are set on a global basis or on particular levels of hierarchy.

To set global optimization controls in the FPGA Express software, follow these steps:

1.

2. If you have not already done so, identify the top-level design for your project in the Design Sources window. Select the top-level design from the *Top-Level Design* drop-down list on the toolbar. The **Create Implementation** dialog box is displayed.

3. Select either *speed* or *area* under *Optimize for* to specify the optimization goal for the entire design:

   -

   - Selecting the *speed* option minimizes delay by synthesizing circuits to contain the least number of levels of combinatorial logic, sometimes yielding increased design area. This setting maximizes operating frequency and minimizes combinatorial path delays.

   - Selecting the *area* option minimizes the combinatorial logic resources used, sometimes yielding reduced speed. This setting minimizes combinatorial logic usage. Altera also recommends selecting the WYSIWYG synthesis style when optimizing for area, as described in Specifying the MAX+PLUS II Logic Synthesis Style with FPGA Express Software.

4. Select either *high* or *low* under *Effort* to specify the CPU effort level:
   -

   - Selecting the *low* option increases compilation speed at the expense of larger combinatorial area. This option is most useful for minimizing compilation time for very large designs when neither speed nor area are critical.

   - Selecting the *high* option decreases the combinatorial area at the expense of compilation speed. This option is recommended in speed- or area-critical designs.

You can set the same optimization controls on individual levels of hierarchy for greater control. This strategy is useful when your design contains hierarchical blocks with different requirements. For example, some blocks may be time-critical while others are not. To obtain the best resuls, you should optimize time-critical blocks for speed and other blocks for area.

To set optimization goals on a particular level of hierarchy, follow these steps:

1. Select the pre-optimized chip icon in the Chips window, press Button 2 and choose **Edit Constraints** to display the constraints tables.
2. Select the **Modules** tab.
3. Find the row that displays the level of hierarchy for which you want to set an optimization goal.
4. In the *Optimize for* column of that row, click inside the cell and select either *speed* or *area* from the options that appear.
5. In the *Effort* column of that row, click inside the cell and select either *high* or *low* from the available options.

1.

2. If you have not already done so, assign a device and Clock frequency, as described in Assigning a Device & Clock Frequency (f_{MAX}).

3. Continue with the steps necessary to process your design, as described in Synthesizing & Optimizing VHDL or Verilog HDL Files with FPGA Express Software.

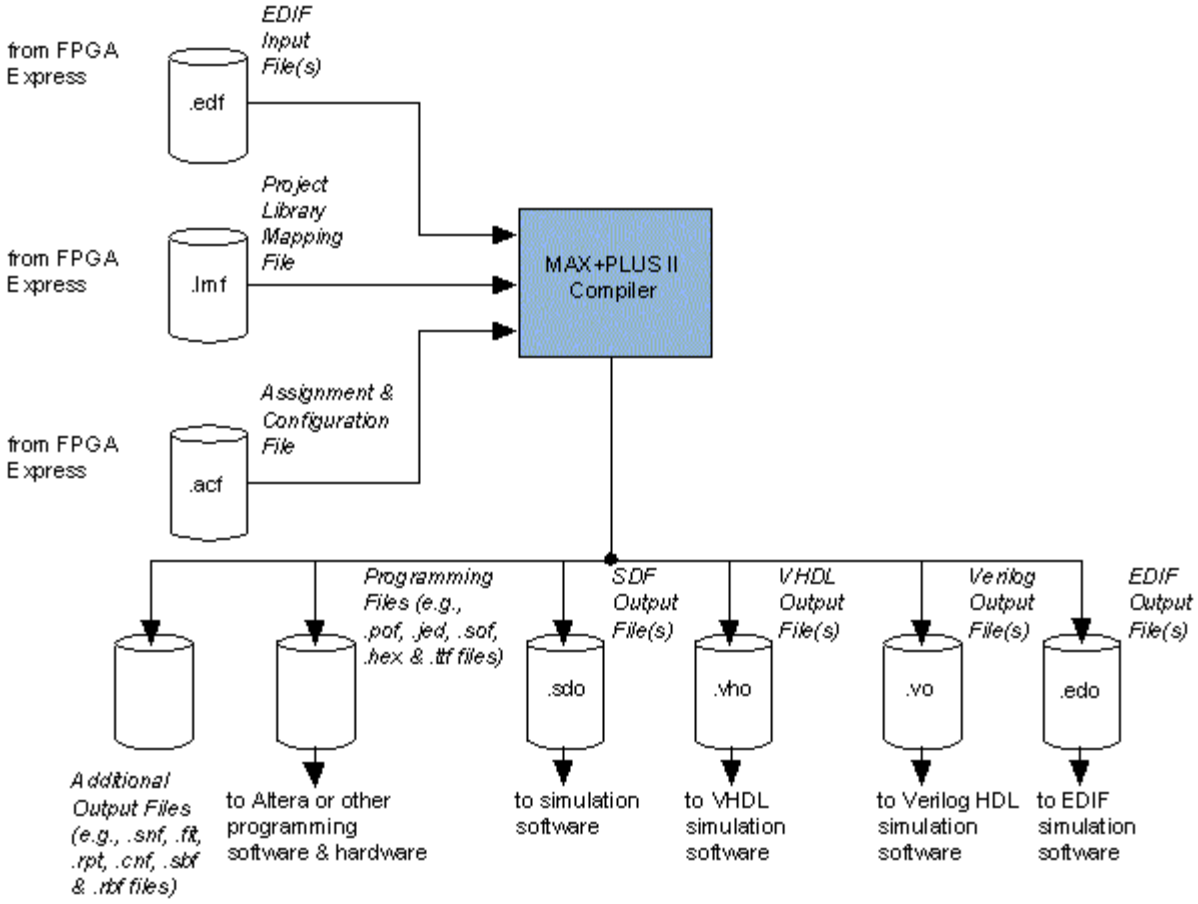
☞ Optimization settings are the same for an entire design file, regardless of its level of hierarchy.

MAX+PLUS II/FPGA Express Project Compilation Flow

Figure 1 below shows the project compilation flow for the MAX+PLUS® II /FPGA Express interface.

Figure 1. MAX+PLUS II/FPGA Express Project Compilation Flow

Altera-provided items are shown in blue.



Compiling Projects with MAX+PLUS II Software

The MAX+PLUS® II Compiler can process design files in a variety of formats. This topic describes how to use MAX+PLUS II software to compile projects in which the top-level design file is an EDIF Input File (with the extension **.edf**).

Related Topics:

- Refer to the following sources for additional information:
  -
  - Go to MAX+PLUS II Help for information on compiling VHDL and Verilog HDL, design files directly with the MAX+PLUS II Compiler.
  - Go to Running Synopsys Compilers from MAX+PLUS II Software for information on running the Synopsys Design Compiler or FPGA Compiler software on a VHDL or Verilog HDL design from within the MAX+PLUS II Compiler window.

To compile a design (also called a "project") with MAX+PLUS II software, go through the following steps:

1.
2. Create design files that are compatible with the MAX+PLUS II software and convert them into EDIF Input Files with the extension **.edf**. Specific instructions for some tools are described in these MAX+PLUS II SM

ACCESS Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your design entry or synthesis and optimization tool.

3. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, create a mapping for each function in a Library Mapping File (**.lmf**) to map the custom symbol to the corresponding EDIF Input File, AHDL Text Design File (**.tdf**), or other MAX+PLUS II-supported design file. These custom functions are represented in design files as hollow-body symbols or "black box" HDL descriptions.

   ☞ Go to "Library Mapping Files (**.lmf**)" in MAX+PLUS II Help for more information.

4. Open MAX+PLUS II and specify the name of your top-level design file as the project name with the **Project Name** command (File menu). If you open an HDL file in the MAX+PLUS II Text Editor, you can choose the **Project Set Project to Current File** command (File menu) instead.

   ☞ You can also compile a project from a command line. However, the first time you compile a project, the settings you need to specify are easier to specify from within the MAX+PLUS II software. After you have run the graphical user interface for the MAX+PLUS II software at least once, you can more easily use the command-line **setacf** utility to modify options in the Assignment & Configuration File (**.acf**) for the project. Type `setacf -h` ↵ and `maxplus2 -h` ↵ for descriptions of **setacf** and MAX+PLUS II command-line syntax.

5. Choose **Device** (Assign menu) and select the target Altera device family in the *Device Family* drop-down list box. If you wish to implement the design logic in a specific device, select it in the *Devices* box. Otherwise, select *AUTO* to allow the MAX+PLUS II Compiler to choose the best device(s) in the current device family. If your design entry or synthesis and optimization tool required you to specify a target family and/or device, specify the same information in this dialog box. For information on partitioning logic among multiple devices, go to MAX+PLUS II Help. Choose **OK**.

6. Open the Compiler window by choosing the **Compiler** command (MAX+PLUS II menu). Go through the following steps to specify the options necessary to compile the design file(s) in your project:

   1.

   2. Ensure that all EDIF netlist files have the extension **.edf** and choose **EDIF Netlist Reader Settings** (Interfaces menu).

   3. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor. This name should be the name of the vendor whose tool(s) you used to create the EDIF netlist files. If your vendor name does not appear, select *Custom* instead.

      ☞ If you are compiling a design created with Synopsys FPGA Express software, select *Synopsys*, choose the **Customize** button, enter <*project name*>**.lmf** in the *LMF #1* box, choose **OK**, and skip to step 6.

   4. If you selected an existing vendor name in the *Vendor* box and your project contains design files that require custom LMF mappings, choose the **Customize** button to expand the dialog box to show all settings. Turn on the *LMF #2* checkbox and type your custom LMF's filename in the corresponding text box, or select a name from the *Files* box. The selection in the *Vendor* box will change to *Custom* and all settings will be retained until you change them again.

   5. If you selected *Custom* in the *Vendor* box, choose the **Customize** button to expand the dialog box to show all settings. Any previously defined custom settings will be displayed. Under *Signal Names*, type one or more names with up to 20 total name characters in the *VCC* or *GND* box if your EDIF Input File(s) use one or more names other than `VCC` or `GND` for the global high or low signals. Multiple signal names must be separated by either a comma (,) or a space. Under *Library Mapping Files*, turn on the

LMF #1 checkbox and type a filename in the text box following it, or select a name from the *Files* box. If necessary, specify another LMF name in the *LMF #2* box. Go to MAX+PLUS II Help for detailed information on the settings available in the **EDIF Netlist Reader Settings** dialog box.

6. Choose **OK**.

7. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, you may need to ensure that all files are present in your project directory, i.e., the same directory as the top-level design file. Otherwise, you must specify the directories containing these files as user libraries with the **User Libraries** command (Options menu).

8. Follow all guidelines that apply to your design entry or synthesis and optimization tool:

   - Exemplar Logic Galileo Extreme-Specific Compiler Settings
   - Synopsys DesignWare-Specific Compiler Settings
   - Converting Synopsys FPGA Compiler & Design Compiler Timing Constraints into MAX+PLUS II-Compatible Format with the **syn2acf** Utility
   - Synplicity Synplify-Specific Compiler Settings

9. If you wish to generate EDIF, VHDL, or Verilog HDL output files for post-compilation simulation or timing analysis with another EDA tool, go through the following steps:

   1.

   2. (Optional) Turn on the **Optimize Timing SNF** command (Processing menu) to reduce the size of the output file(s). Turning on this command can reduce the size of output netlists by up to 30%.

      This command does not create optimized timing SNFs on UNIX workstations. However, a non-optimized timing SNF provides the same functional and timing information as an optimized timing SNF.

   3. If you wish to generate EDIF Output Files (**.edo**), go through these steps:

      1.

      2. Turn on the **EDIF Netlist Writer** command (Interfaces menu). Then choose the **EDIF Netlist Writer Settings** command (Interfaces menu).

      3. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor and choose **OK**. If your vendor name does not appear, select *Custom* instead and specify the settings that are appropriate for your simulation or timing analysis tool. Go to MAX+PLUS II Help for detailed information on the options available in the **EDIF Netlist Writer Settings** dialog box.

      4. To generate an optional Standard Delay Format (SDF) Output File (**.sdo**), choose the **Customize** button to expand the dialog box to show all settings. Select one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**.

      The filenames of the EDIF Output File(s) and optional SDF Output File(s) are the same as the user-defined chip name(s) for the project; if no chip names exist, the Compiler assigns filenames that are based on the project name. For a multi-device project, the Compiler also generates a top-level EDIF Output File that is uniquely identified by "**_t**" appended to the project name. In addition, the Compiler automatically generates a VHDL Memory Model Output File, *<project name>***.vmo**, when it generates an EDIF Output File that contains memory (RAM or ROM).

   4. If you wish to generate VHDL Output Files (**.vho**), turn on the **VHDL Netlist Writer** command (Interfaces menu). Then choose **VHDL Netlist Writer Settings** command (Interfaces menu). Select

VHDL Output File (.vho) or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF ver. 2.1 files contain timing delay information that allows you to perform back-annotation simulation in VHDL with VITAL-compliant simulation libraries. The VHDL Output Files generated by the Compiler have the extension **.vho**, but are otherwise named in the same way as the EDIF Output Files described above.

5. If you wish to generate Verilog HDL Output Files (**.vo**), turn on the **Verilog Netlist Writer** command (Interfaces menu). Then choose **Verilog Netlist Writer Settings** command (Interfaces menu). Select *Verilog Output File (.vo)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF Output Files contain timing delay information that allows you to perform back-annotation simulation in Verilog HDL. The Verilog Output Files generated by the Compiler have the extension **.vo**, but are otherwise named in the same way as the EDIF Output Files described above.

10. To run the MAX+PLUS II Compiler, choose the **Project Save & Compile** command (File menu) or choose the **Start** button in the Compiler window.

    ☞ See step 3 for information on running MAX+PLUS II software from the command line.

11. Once you have compiled the project with the MAX+PLUS II Compiler, you can use the VHDL, Verilog HDL, or EDIF output file(s), and the optional SDF Output File(s) (**.sdo**) to perform timing analysis or timing simulation with another EDA tool. Specific instructions for some tools are described in these MAX+PLUS II ACCESS Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your EDA tool.

The MAX+PLUS II Compiler also generates a Report File (**.rpt**), a Pin-Out File (**.pin**), and one or more of the following files for device programming or configuration:

- JEDEC Files (**.jed**)
- Programmer Object Files (**.pof**)
- SRAM Object Files (**.sof**)
- Hexadecimal (Intel-format) Files (**.hex**)
- Tabular Text Files (**.ttf**)

Related Topics:

- Refer to the following sources for additional information:
  - Go to *Compiler Procedures* in MAX+PLUS II Help for information on other available Compiler settings.
  - Go to *Programmer Procedures* in MAX+PLUS II Help for instructions on creating other types of programming files and on programming or configuring Altera devices.
  - Go to Back-Annotating MAX+PLUS II Pin Assignments to Design Architect Symbols for information on back-annotating pin assignments in Mentor Graphics Design Architect schematics.
  - Go to Programming Altera Devices for information on the different programming hardware options for Altera device families.

- Go to the following topics for additional information:

  - MAX+PLUS II Development Software
  - Altera Programming Hardware

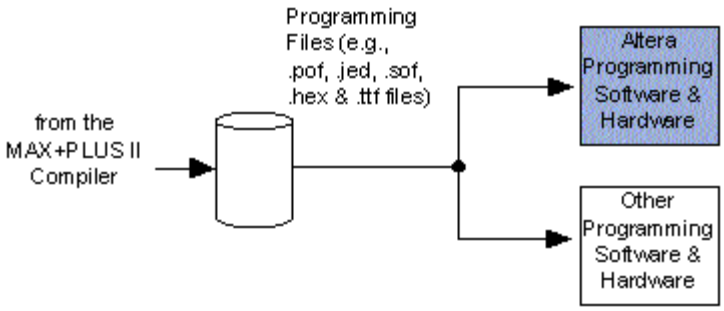
Programming Altera Devices

Once you have successfully compiled and simulated a project with the MAX+PLUS® II software, you can program an Altera® device and test it in the target circuit. Figure 1 shows the device programming flow for MAX+PLUS II

software.

Figure 1. MAX+PLUS II Device Programming Flow

Altera-provided items are shown in blue.



You can program devices with Altera programming hardware and MAX+PLUS II Programmer software installed on a 486- or Pentium-based PC or a UNIX workstation, or with programming hardware and software available from other manufacturers. Table 1 shows the available Altera programming hardware options on PCs and UNIX workstations.

Table 1. Altera Programming Hardware

Programming Hardware Option	PCs	UNIX Work-stations	MAX® 3000A Devices	Classic® & MAX 5000 Devices	MAX 7000 & MAX 7000E Devices	MAX 7000A, MAX 7000AE, MAX 7000S MAX 9000 & MAX 9000A Devices	FLEX® 6000, FLEX 6000A, FLEX 8000, FLEX 10K, FLEX 10KA, FLEX 10KB, & FLEX 10KE Devices	In-System Programming/ Configuration
Logic Programmer card, PL-MPU Master Programming Unit, and device-specific adapters	✓		✓	✓	✓	✓		
BitBlaster™ Download Cable	✓	✓	✓			✓	✓	✓
ByteBlasterMV™ Download Cable	✓		✓			✓	✓	✓

If you wish to transfer programming files from a UNIX workstation to a PC over a network with File Transfer Protocol (FTP) or other similar transfer programs, be sure to select binary transfer mode.

Programming hardware from other manufacturers varies, but typically consists of a device connected to one of the serial ports on the workstation. Various vendors, such as Data I/O and BP Microsystems, supply hardware and software for programming Altera devices.

Related Links

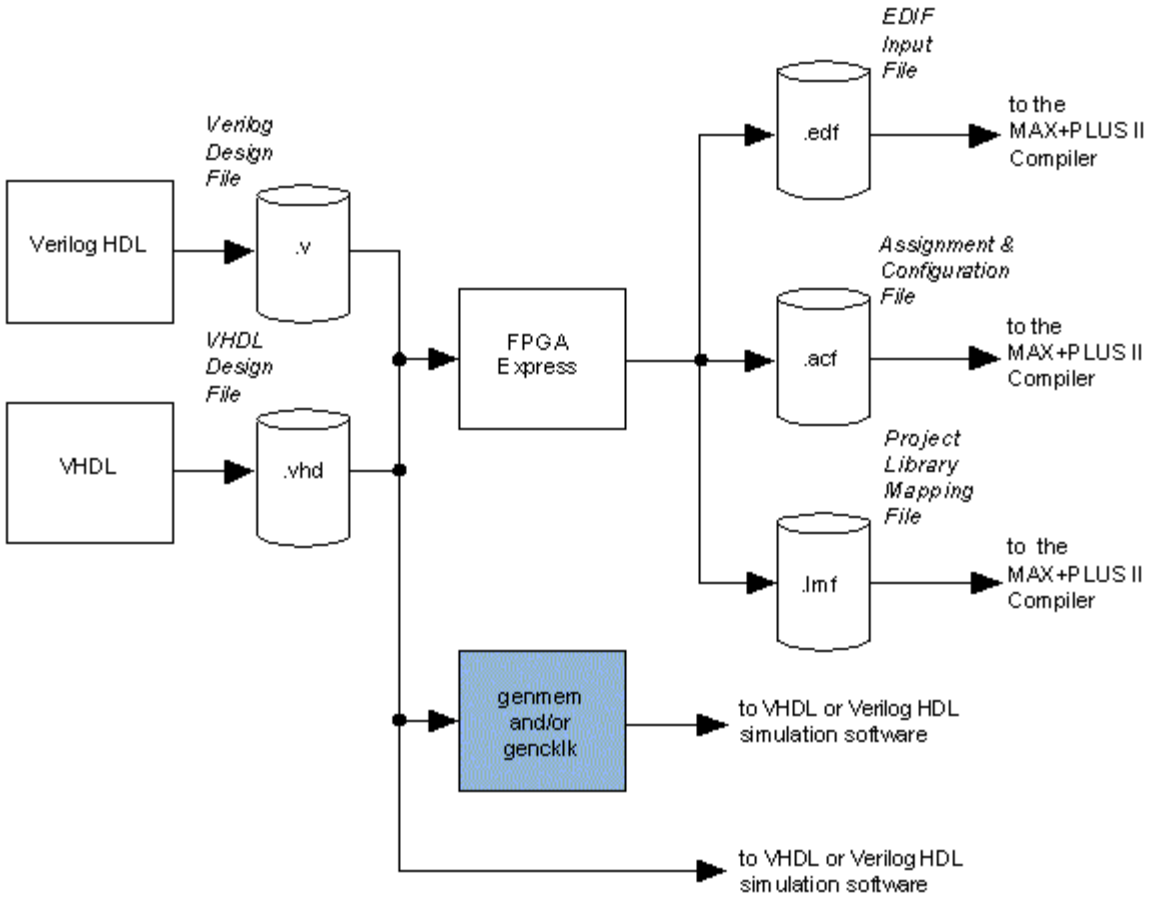
- MAX+PLUS II Development Software
- Altera Programming Hardware

FPGA Express Design Entry Flow

Figure 1 shows the design entry flow for the MAX+PLUS® II /FPGA Express interface.

Figure 1. MAX+PLUS II/FPGA Express Design Entry Flow

Altera-provided items are shown in blue.



Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Assigning a Device & Clock Frequency (fMAX)

You can specify the desired Clock frequency (called **f_{MAX}** in the MAX+PLUS® II software) and the target device family before synthesizing and optimizing the design with the FPGA Express software. You can optionally select a specific device and speed grade within the target device family. These assignments are stored in the design's Assignment & Configuration File, <*design name*>**.acf**, which is generated automatically by the FPGA Express software.

To assign a device or device family and the Clock frequency, follow these steps:

1. If you have not already done so, identify the top-level design for your project from the Design Sources window. Select the top-level design from the *Top-Level Design* drop-down list on the toolbar. The **Create Implementation** dialog box is displayed.
2. Enter an implementation name in the *Implementation Name* box. If you do not enter a name, FPGA Express software automatically creates a unique implementation name.
3. Select *Altera* from the *Vendor* list.
4. Select the appropriate Altera device family from the *Family* list.
5. (Optional) Select a specific device from the *Device* list, and select a specific speed grade from the *Speed Grade* list.
6. Type the desired Clock frequency in the *Clock Frequency* text box. This Clock frequency is used as the default value for all Clock signals in the design.
7. (Optional) Select speed/area and CPU effort settings, as described in Specifying the Speed/Area & CPU Effort Settings with the FPGA Express Software.
8. Continue with the steps necessary to process your design, as described in Synthesizing & Optimizing VHDL or Verilog HDL Files with FPGA Express Software.

☞ You can also edit the Clock frequency by double-clicking on the design implementation name to open the constraint tables and entering information on the **Clock** tab. For more information, refer to Assigning Pins, Logic Options, and t_{SU}, t_{CO} & t_{PD} Timing Constraints.

Related Topics:

- Go to the following sources for related information:
  - FPGA Express Help
  - "Guidelines for Achieving Maximum Speed Performance" in MAX+PLUS II Help

---

Feedback

Did this information help you?

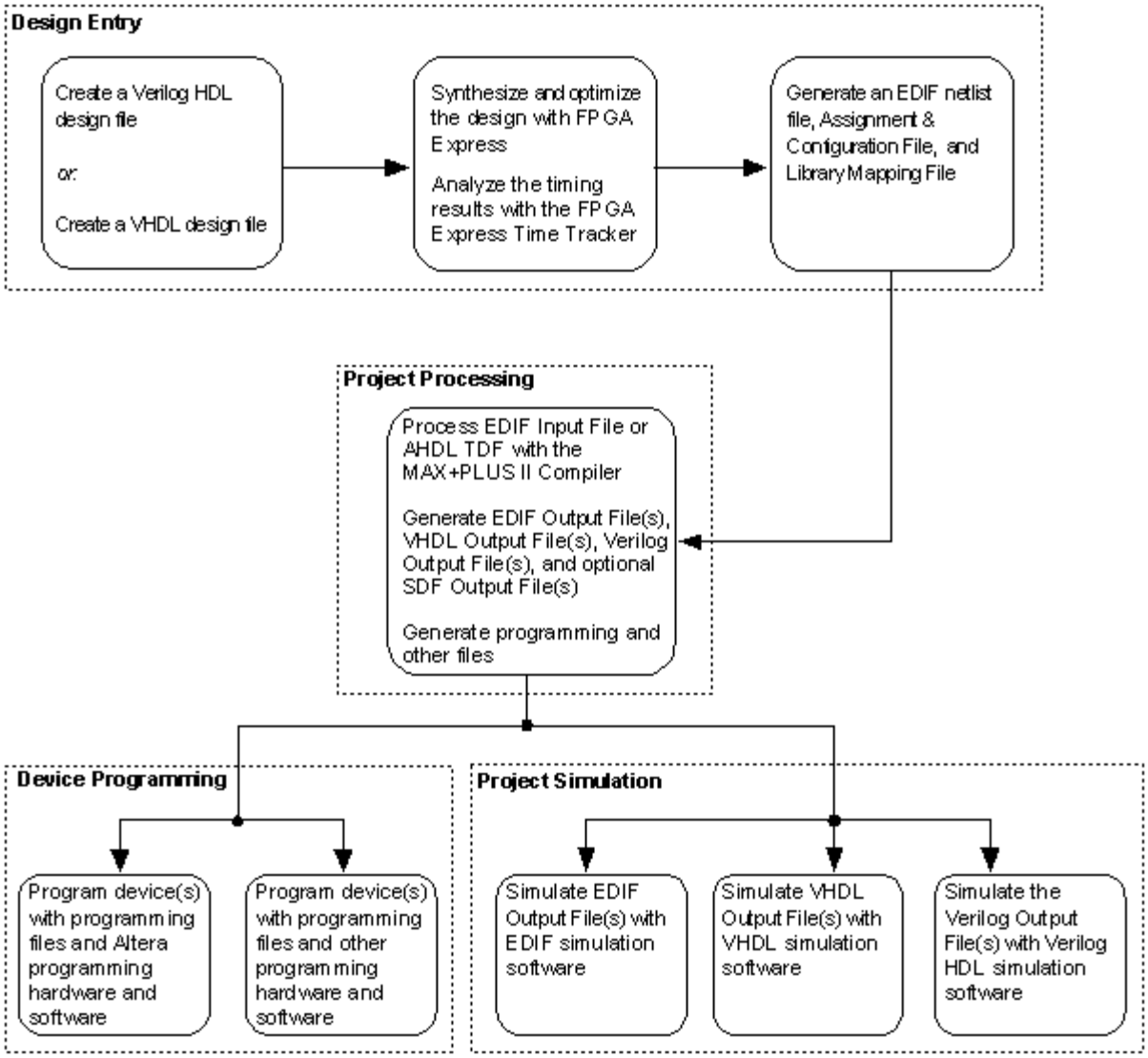
If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Design Flow for FPGA Express Software

Figure 1 shows the design flow for the MAX+PLUS® II /FPGA Express interface.

Figure 1. MAX+PLUS II/FPGA Express Design Flow



Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Instantiating LPM Functions in VHDL

You can enter library of parameterized modules (LPM) functions in your VHDL design. The MAX+PLUS® II software supports all LPM functions except the truth table, finite state machine, and pad functions. The FPGA Express software supports all LPM functions that are supported in the MAX+PLUS II software except the `lpm_and`, `lpm_or`, `lpm_xor`, and `lpm_mux` functions. Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu for detailed information on all LPM functions.

To instantiate an LPM function in a VHDL design, follow these steps:

1. Create an instance of an LPM function in VHDL with a Component Instantiation Statement. VHDL Component Declarations for LPM functions are available in MAX+PLUS II Help and also installed automatically in the following FPGA Express directory:

   <*drive*>:Ã‚Â¥synopsysÃ‚Â¥fpga_expressÃ‚Â¥libÃ‚Â¥packagesÃ‚Â¥lpmÃ‚Â¥lpm_components.vhd

   Use named association to specify parameter values in the Generic Map Clauses of LPM function instantiations. Figure 1 shows the Component Declaration for the `lpm_ram_dq` function.

   Figure 1. Component Declaration for lpm_ram_dq (from lpm_components.vhd)

```
COMPONENT lpm_ram_dq
   GENERIC (
            LPM_WIDTH: POSITIVE;
            LPM_TYPE : STRING := L_RAM_DQ;
            LPM_WIDTHAD: POSITIVE;
            LPM_NUMWORDS: STRING := UNUSED;
            LPM_FILE: STRING := UNUSED;
            LPM_INDATA: STRING := REGISTERED;
            LPM_ADDRESS_CONTROL : STRING := REGISTERED;
            LPM_OUTDATA: STRING := REGISTERED
            );
   PORT    (
            data     : IN STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0);
            we       : IN STD_LOGIC := '1';
            inclock  : IN STD_LOGIC := '1';
            outclock : IN STD_LOGIC := '1';
            address  : IN STD_LOGIC_VECTOR(LPM_WIDTHAD-1 DOWNTO 0);
            q        : OUT STD_LOGIC_VECTOR (LPM_WIDTH-1 DOWNTO 0)
           );
END COMPONENT;
```

2. Manually tie any pins that require an initial value of logic 1 to `vcc`. The FPGA Express software does not support initial values of logic 1 in Component Declarations. However, it does support initial values of logic 0.

   Figure 2 shows an example of instantiating an `lpm_ram_dq` function in VHDL.

   Figure 2. VHDL Design File with lpm_ram_dq Instantiation

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY lpm;
USE lpm.lpm_components.all;

ENTITY design IS
   PORT(
        din  : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
        we   : IN STD_LOGIC;
        clk  : IN STD_LOGIC;
        addr : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        dat  : OUT STD_LOGIC_VECTOR(15 DOWNTO 0)
```

```
        );
END design;

ARCHITECTURE struct OF design IS
   SIGNAL vcc : STD_LOGIC;

BEGIN  --  struct
   vcc <= '1';

u1:  lpm_ram_dq
   GENERIC MAP(
               LPM_WIDTH   => 16,
               LPM_WIDTHAD => 4,
               LPM_INDATA  => "UNREGISTERED",
               LPM_OUTDATA => "UNREGISTERED"
              )

   PORT MAP(
            data     => din,
            address  => addr,
            we       => we,
            q        => dat,
            inclock  => vcc,
            outclock => clk
            );
END struct;
```

3. Continue with the steps necessary to complete your VHDL design, as described in Creating VHDL Designs for Use with MAX+PLUS II Software.

Instantiating LPM Functions in Verilog HDL

You can enter library of parameterized modules (LPM) functions in your Verilog HDL design. The MAX+PLUS® II software supports all LPM functions except the truth table, finite state machine, and pad functions. The FPGA Express software supports all LPM functions that are supported in the MAX+PLUS II software except the `lpm_and`, `lpm_or`, `lpm_xor`, and `lpm_mux` functions. Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu for detailed information on all LPM functions.

To instantiate an LPM function in a Verilog HDL design, follow these steps:

1. Use a Module Instantiation to instantiate an LPM function. You must associate all parameters, and only positional association is allowed.

   Figure 1 shows an example of instantiating an `lpm_ram_dq` function in Verilog HDL.

   Figure 1. Verilog HDL Design File with lpm_ram_dq Instantiation

```
// RAM design

module design(din, we, clk, addr, dat);

input [15:0] din;
input we, clk;
input [3:0] addr;
output [15:0] dat;

supply1 vcc;

lpm_ram_dq #(
16,                     // LPM_WIDTH
"LPM_RAM_DQ",           // LPM_TYPE
4,                      // LPM_WIDTHAD
16,                     // LPM_NUMWORDS
"UNUSED",               // LPM_FILE
"UNREGISTERED",         // LPM_INDATA
"UNREGISTERED",         // LPM_ADDRESS_CONTROL
"UNREGISTERED"          // LPM_OUTDATA
)

u1(
.data(din),
.address(addr),
.we(we),
.q(dat),
.inclock(vcc),
.outclock(clk)
);

endmodule
```

2. Continue with the steps necessary to complete your Verilog HDL design file, as described in Creating Verilog HDL Designs for Use with MAX+PLUS II Software.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Specifying Speed/Area & CPU Effort Settings with the FPGA Express Software

FPGA Express software allows you to choose either speed or area options and to specify either high or low CPU effort in logic optimization. Optimization goals are set on a global basis or on particular levels of hierarchy.

To set global optimization controls in the FPGA Express software, follow these steps:

1. If you have not already done so, identify the top-level design for your project in the Design Sources window. Select the top-level design from the *Top-Level Design* drop-down list on the toolbar. The **Create Implementation** dialog box is displayed.
2. Select either *speed* or *area* under *Optimize for* to specify the optimization goal for the entire design:
   - Selecting the *speed* option minimizes delay by synthesizing circuits to contain the least number of levels of combinatorial logic, sometimes yielding increased design area. This setting maximizes operating frequency and minimizes combinatorial path delays.
   - Selecting the *area* option minimizes the combinatorial logic resources used, sometimes yielding reduced speed. This setting minimizes combinatorial logic usage. Altera also recommends selecting the WYSIWYG synthesis style when optimizing for area, as described in Specifying the MAX+PLUSÂ II Logic Synthesis Style with FPGA Express Software.
3. Select either *high* or *low* under *Effort* to specify the CPU effort level:
   - Selecting the *low* option increases compilation speed at the expense of larger combinatorial area. This option is most useful for minimizing compilation time for very large designs when neither speed nor area are critical.
   - Selecting the *high* option decreases the combinatorial area at the expense of compilation speed. This option is recommended in speed- or area-critical designs.

You can set the same optimization controls on individual levels of hierarchy for greater control. This strategy is useful when your design contains hierarchical blocks with different requirements. For example, some blocks may be time-critical while others are not. To obtain the best resuls, you should optimize time-critical blocks for speed and other blocks for area.

To set optimization goals on a particular level of hierarchy, follow these steps:

1. Select the pre-optimized chip icon in the Chips window, press Button 2 and choose **Edit Constraints** to display the constraints tables.
2. Select the **Modules** tab.
3. Find the row that displays the level of hierarchy for which you want to set an optimization goal.
4. In the *Optimize for* column of that row, click inside the cell and select either *speed* or *area* from the options that appear.
5. In the *Effort* column of that row, click inside the cell and select either *high* or *low* from the available options.

1. If you have not already done so, assign a device and Clock frequency, as described in Assigning a Device & Clock Frequency (f_{MAX}).
2. Continue with the steps necessary to process your design, as described in Synthesizing & Optimizing VHDL or Verilog HDL Files with FPGA Express Software.

☞ Optimization settings are the same for an entire design file, regardless of its level of hierarchy.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Assigning Pins, Logic Options, and tSU, tCO & tPD Timing Constraints

You can assign pins, logic options, or timing constraints to your design in FPGA Express constraint tables. Some design-specific information is extracted automatically from your design and displayed in the constraint tables; you can also manually enter specific assignments in these tables. The FPGA Express software saves the assignments to an Assignment & Configuration File (**.acf**) when it synthesizes and optimizes the design. The MAX+PLUS® II software uses the assignment information from the ACF when it processes the design. Refer to Using ACFs Generated by FPGA Express Software for more information.

To enter resource assignments in FPGA Express software, follow these steps:

1. Select the design implementation icon in the Chips window, press Button 2, and choose the **Edit Constraints** commmand from the pop-up menu to display the Altera-specific constraint tables. These tables allow you to specify resource assignments for your design. All design-specific information such as Clock names, port names, and design hierarchy is extracted automatically from the design. Altera recommends entering specific requirements directly into these tables to obtain the desired optimization. For information on creating a design implementation, refer to steps 1 through 8 in Synthesizing & Optimizing VHDL or Verilog HDL Files with FPGA Express Software.

2. Enter assignments in the appropriate constraint tables. You can click on a tab to toggle between tables. Refer to Table 1, which shows the available MAX+PLUS II resource assignment options in the FPGA Express constraint tables. The **Clock** and **Path** tables already contain information that you previously entered in the **Create Implementation** dialog box. Refer to Assigning a Device & Clock Frequency (f_{MAX}) for more information.

   Table 1. MAX+PLUS II Resource Assignments in FPGA Express Constraint Tables

MAX+PLUS II Resource Assignment	Equivalent FPGA Express Constraint Table Setting	
	Tab Name	**Action**
Pin assignment	**Ports**	Specify the pin number in the *Pad Location* column.
t_{SU} timing assignment	**Ports**	Specify the time in the *Input Delay* column.
t_{CO} timing assignment	**Ports**	Specify the time in the *Output Delay* column.
Slow Slew Rate logic option assignment	**Ports**	Click on the appropriate cell in the *Slew Rate* column and select *<default>*, *FAST*, or *SLOW* from the list.
Fast I/O logic option assignment	**Ports**	Click on the appropriate cell in the *Use I/O Reg* column and select *<default>*, *ON*, or *OFF* from the list.
t_{PD} timing assignment	**Paths**	Specify the time in the *Delay* column.

3. Choose **Save** and then **Close** to exit from the FPGA Express constraint tables.

Related Links:

- Go to the following sources for related information:
  - FPGA Express Help
  - "Guidelines for Achieving Maximum Speed Performance" in MAX+PLUS II Help
  - Specifying the MAX+PLUS II Logic Synthesis Style with the FPGA Express Software in these

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

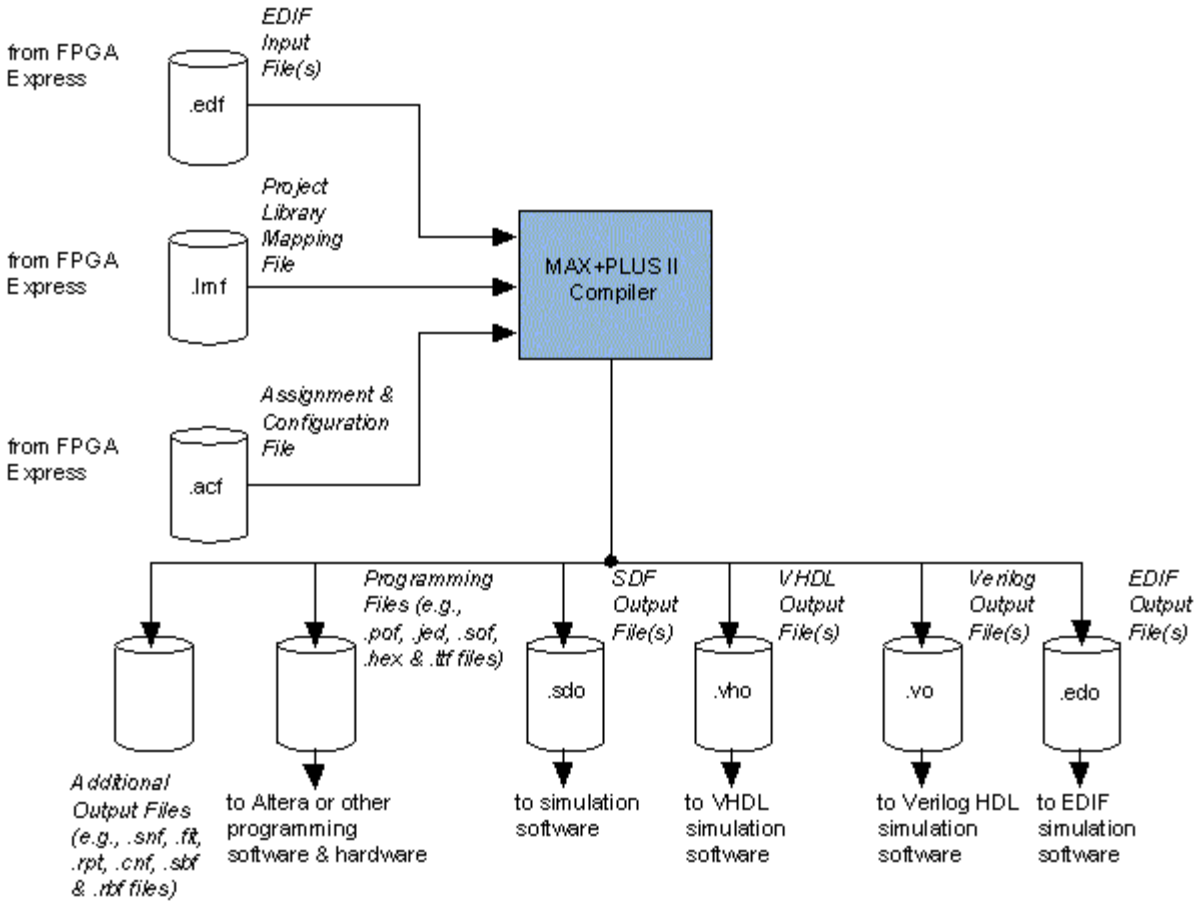
Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

MAX+PLUS II/FPGA Express Project Compilation Flow

Figure 1 below shows the project compilation flow for the MAX+PLUS® II /FPGA Express interface.

Figure 1. MAX+PLUS II/FPGA Express Project Compilation Flow

Altera-provided items are shown in blue.



Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Instantiating RAM & ROM Functions in Verilog HDL

The MAX+PLUS® II /Synopsys interface offers full support for the memory capabilities of the FLEX® 10K device family, including synchronous and asynchronous RAM and ROM, cycle-shared dual port RAM, dual-port RAM, single-Clock FIFO, and dual-clock FIFO functions. Altera recommends using the LPM functions `lpm_ram_dq`, `lpm_ram_io`, and `lpm_rom` to instantiate synchronous and asynchronous RAM and ROM. However, if you wish to enter cycle-shared dual port ram (`csdpram`), dual-port RAM (`altdpram`), single-Clock FIFO (`scfifo`), and dual-clock FIFO (`dcfifo`) functions, or if you wish create simulation models for any supported RAM or ROM function, you can use the Altera-provided **genmem** utility. Designs that instantiate genmem-generated synchronous and asynchronous RAM and ROM -- such as those used with FPGA Compiler or Design Compiler -- are supported for backward compatibility. Type `genmem` ↵ at the DOS or UNIX prompt to display information on how to use this utility, as well as a list of the functions you can generate.

To instantiate an `lpm_ram_dq`, `lpm_ram_io`, or `lpm_rom` function:

✔ Follow the guidelines in Instantiating LPM Functions in Verilog HDL.

To instantiate other RAM and ROM functions in Verilog HDL, follow these steps:

1. Use the **genmem** utility to generate a memory model by typing the following command at a DOS or UNIX prompt:

   `genmem` <*memory type*> <*memory size*> `-verilog` ↵

   For example: `genmem scfifo 16x8 -verilog` ↵

2. Create a Verilog HDL design that instantiates the **<*memory name*>.v** function. The **genmem** utility produces files with descriptive names that typically include both the memory type and the memory size (e.g., **scfifo_16x8_d.v**).

   ☞ In MAX+PLUS II version 8.3 and lower, running `genmem` on a PC always creates files named as **genmem.vhd**, **genmem.cmp**, and **genmem.v**, regardless of the memory type and memory size values you specify.

1. (Optional for RAM functions) Specify an initial memory content file:

   - For ROM functions, you must specify the filename of an initial memory content file in the Intel hexadecimal format (**.hex**) or the Altera® Memory Initialization File (**.mif**) format in the Parameter Statement, using the `LPM_FILE` parameter. The filename must be the same as the instance name; e.g., the `u1` instance name must be unique throughout the whole project, and must contain only valid Verilog HDL name characters. The initialization file must reside in the directory containing the project's design files.
   - For RAM functions, specifying a memory initialization file is optional. If you want to use it, you must specify it in the Parameter Statement as described above.

1. The MIF format is supported only for specifying initial memory content when compiling designs within MAX+PLUS II software. You cannot use an MIF to perform simulation with Synopsys tools prior to MAX+PLUS II compilation.

☞

2. If you use an Intel hexadecimal format file and wish to simulate the file with the VHDL System Simulator Software (VSS) after MAX+PLUS II compilation, you should use the Synopsys **intelhex** utility to translate the Intel hexadecimal fomat file into a VSS-compatible Synopsys memory file. Refer to the Synopsys ***VHDL System Simulator Software Tool*** manual for details about using the **intelhex** utility.

1. Continue with the steps necessary to complete your Verilog HDL design, as described in Creating Verilog HDL Designs for Use with MAX+PLUS II Software.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Instantiating RAM & ROM Functions in VHDL

The MAX+PLUS® II /Synopsys interface offers full support for the memory capabilities of the FLEX®10K device family, including synchronous and asynchronous RAM and ROM, cycle-shared dual port RAM, dual-port RAM, single-Clock FIFO, and dual-clock FIFO functions. Altera recommends using the LPM functions `lpm_ram_dq`, `lpm_ram_io`, and `lpm_rom` to instantiate synchronous and asynchronous RAM and ROM. However, if you wish to enter cycle-shared dual port RAM (`csdpram`), dual-port RAM (`altdpram`), single-Clock FIFO (`scfifo`), and dual-clock FIFO (`dcfifo`) functions, or if you wish to create simulation models for any supported RAM or ROM function, you can use the Altera-provided **genmem** utility. Instantiations created with **genmem** for use with other Synopsys products, such as FPGA Compiler or Design Compiler, are supported for backward compatibility. Type `genmem` ↵ at the DOS or UNIX prompt to display information on how to use this utility, as well as a list of the functions you can generate.

To instantiate an `lpm_ram_dq`, `lpm_ram_io`, or `lpm_rom` function:

✔ Follow the guidelines in Instantiating LPM Functions in VHDL.

To instantiate other RAM and ROM functions in VHDL, follow these steps:

1. Use the **genmem** utility to generate a memory model by typing the following command at the DOS or UNIX prompt:

   `genmem` <*memory type*> <*memory size*> `-vhdl` ↵

   For example: `genmem scfifo 16x8 -vhdl` ↵

2. Create a VHDL design that incorporates the text from the **genmem**-generated Component Declaration, ***<memory name>*.cmp**, and instantiate the ***<memory name>*.vhd** function. The **genmem** utility produces files with descriptive names that typically include both the memory type and the memory size (e.g., **scfifo_16x8_d.vhd**).

   ☞ In MAX+PLUS II version 8.3 and lower, running `genmem` on a PC always creates files named as **genmem.vhd**, **genmem.cmp**, and **genmem.v**, regardless of the memory type and memory size you specify.

1. (Optional for RAM functions) Specify an initial memory content file:

   - For ROM functions, you must specify the filename of an initial memory content file in the Intel hexadecimal format (**.hex**) or the Altera® Memory Initialization File (**.mif**) format in the Generic Map Clause, with the `LPM_FILE` parameter. The filename must be the same as the instance name; e.g., the `u1` instance name shown in Figure 1 must be unique throughout the whole project, and must contain only valid VHDL name characters. The initialization file must reside in the directory containing the project's design files.
   - For RAM functions, specifying memory initialization file is optional. If you want to use it, you must specify it in the Generic Map Clause as described above. If you do not use an initialization file, you should not declare or use the Generic Clause.

     1. The MIF format is supported only for specifying initial memory content when compiling designs within MAX+PLUS II software. You cannot use an MIF to perform simulation with Synopsys tools prior to MAX+PLUS II compilation.

     ☞ 2. If you use an Intel hexadecimal format file and wish to simulate the file with the VHDL System Simulator (VSS) after MAX+PLUS II compilation, you should use the Synopsys

intelhex utility to translate the Intel hexadecimal fomat file into a VSS-compatible Synopsys memory file. Refer to the Synopsys ***VHDL System Simulator Software Tool*** manual for details about using the **intelhex** utility.

2. Continue with the steps necessary to complete your VHDL design, as described in Creating VHDL Designs for Use with MAX+PLUS II Software.

Related Links:

- Go to Using Synopsys VSS & MAX+PLUS II Software in these MAX+PLUS II ACCESS^{SM} Key topics for related information.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Entering Resource, Device & Global Logic Synthesis Assignments

The MAX+PLUS® II software allows you to enter a variety of resource, device, and global logic synthesis assignments for your projects. Resource assignments are used to assign logic functions to a particular pin, logic cell, I/O cell, embedded cell, row, column, Logic Array Block (LAB), Embedded Array Block (EAB), chip, clique, local routing, logic option, timing requirement, or connected pin group. With MAX+PLUS II software, you can enter all types of resource, device, and global logic synthesis assignments with Assign menu commands. You can also enter pin, logic cell, I/O cell, embedded cell, LAB, EAB, row, and column assignments in the MAX+PLUS II Floorplan Editor. The Assign menu commands and the Floorplan Editor all save assignment information in the ASCII Assignment & Configuration File (**.acf**) for the project.

In the Synopsys FPGA Express software, you can assign a limited subset of these assignments in the **Create Implementation** dialog box, in the **Options** dialog box, and by specifying options in constraint tables. These attributes are incorporated into the ACF generated by the FPGA Express software. Refer to the following topics for more information:

- Assigning a Device & Clock Frequency ($\mathbf{f_{MAX}}$)
- Assigning Pins, Logic Options, and $\mathbf{t_{SU}}$, $\mathbf{t_{CO}}$ & $\mathbf{t_{PD}}$ Timing Constraints
- Specifying the MAX+PLUS II Logic Synthesis Style with FPGA Express Software
- Using ACFs Generated by FPGA Express Software
- Modifying the Assignment & Configuration File with the **setacf** utility

Related Links:

- Go to the following sources for related information:
  - FPGA Express Help
  - "resource assignments" or "ACF, format" in MAX+PLUS II Help
  - Specifying the Speed/Area & CPU Effort Settings with the FPGA Express Software in these MAX+PLUS II ACCESS℠ Key topics

---

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

MAX+PLUS II/FPGA Express Software Requirements

Table 1 shows the software applications that are used to generate, process, synthesize, and verify a project with MAX+PLUS® II and FPGA Express software:

Table 1. Software Requirements

Synopsys	Altera
FPGA Express version 3.0	MAX+PLUS II version 9.3 and higher

The FPGA Express software supports devices from all FLEX device families and the MAX 7000 and MAX 9000 device families.

The MAX+PLUS II **read.me** file provides up-to-date information on which versions of Synopsys applications are supported by the current version of the MAX+PLUS II software. It also provides information on installation and operating requirements. You should read the **read.me** file on the ***MAX+PLUS II CD-ROM*** before installing the MAX+PLUS II software. After installation, you can open the **read.me** file from the MAX+PLUS II Help menu.

Related Links:

- Go to the FPGA Express release notes for information on installing the FPGA Express software and a description of its latest changes. These notes are available in the **readme.htm** file on the FPGA Express installation CD-ROM.
- Go to the following topics for additional information:

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Specifying the MAX+PLUS II Logic Synthesis Style with FPGA Express Software

You can specify the MAX+PLUS® II global project logic synthesis style for FLEX® devices from within the FPGA Express software. To specify the global project logic synthesis style, follow these steps:

1. Choose **Options** (Synthesis menu) to display the **Options** dialog box.
2. Choose the **Behavior** tab.
3. Turn the *Insert LCELL Buffers, Style WYSIWYG (Altera FLEX Only)* option on or off:
    - Turning this option on is the equivalent of specifying the WYSIWYG (What You See Is What You Get) logic synthesis style in the MAX+PLUS II software. The WYSIWYG style directs the Compiler's Logic Synthesizer module to change the logic in your project as little as possible during compilation. The WYSIWYG style avoids removing or inserting additional logic, and turns off many logic options that might help the project to fit. In addition, if this option is turned on, the FPGA Express software inserts `LCELL` buffers for look-up table (LUT) outputs so that the MAX+PLUS II software will not alter the logic cell implementations. This option is recommended when a design's area optimization has priority over its speed.
    - Turning this option off is the equivalent of specifying the Fast logic synthesis style in the MAX+PLUS II software. The Fast style directs the Compiler's Logic Synthesizer module to optimize your project for maximum speed, rather than for minimum silicon usage. In addition, if this option is turned off, the FPGA Express software does not insert `LCELL` buffers, thereby allowing the MAX+PLUS II software to optimize the LUT logic to improve performance.
4. Choose **OK**.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Synthesizing & Optimizing VHDL or Verilog HDL Files with FPGA Express Software

You can analyze, synthesize, and optimize design files using the FPGA Express software, then convert them to EDIF netlist files that can be processed by the MAX+PLUSÂ® Â II software.

To process a VHDL or Verilog HDL design for use with MAX+PLUSÂ II software, follow these steps:

1. Create a VHDL file, <*design name*>**.vhd**, or Verilog HDL file, <*design name*>**.v**, using the MAX+PLUSÂ II Text Editor or another standard text editor and save it in your working directory. Go to Creating VHDL Designs for Use with MAX+PLUSÂ II Software or Creating Verilog HDL Designs for Use with MAX+PLUSÂ II Software for more information on VHDL or Verilog HDL design entry.
2. Start the FPGA Express software. Select *Create a new project* in the **Startup** dialog box and choose **OK**. The **Create Project Folder** dialog box is displayed. You can also view the **Create Project Folder** dialog box by choosing **New** (File menu).
3. Specify the full file and path name of the project in the **Create Project Folder** dialog box and choose *Create*. The FPGA Express software creates the project and opens the **Identify Source File** dialog box.
4. Identify and analyze the source files for the project by selecting them in the **Identify Source File** dialog box and choosing **Add**. The FPGA Express internal text editor automatically analyzes each source file as it appears on the left-hand side of the Project window. A green checkmark appears to the left of each filename for the files that have no errors or warnings; a red cross appears for files with errors; and an exclamation point appears for files with warnings.
5. Select the source file icon to display any errors or warnings in the Output window. To fix an error, double-click on the error. The FPGA Express internal text editor automatically displays the source file and highlights the line containing the error or warning in red. To view help on the error or warning, double-click on the error or warning code number (shown in parentheses) in the Output window.

   ☞ FPGA Express software does not copy source files; it identifies and analyzes them in their current location. Refer to FPGA Express Help for more information.
6. Specify the MAX+PLUSÂ II logic synthesis style. Refer to Specifying the MAX+PLUSÂ II Logic Synthesis Style with FPGA Express software for more information.
7. From the Project window, identify the top-level design for your project. Select the top-level design from the *Top-Level Design* drop-down list on the toolbar. The **Create Implementation** dialog box is displayed.
8. In the **Create Implementation** dialog box, specify the following options:
   1. Assign a device and the Clock frequency. Refer to Assigning a Device & Clock Frequency (f_{MAX}) for more information.
   2. Select a global optimization goal (*speed* or *area*) and a CPU effort designation (*high* or *low*). Refer to Specifying the Speed/Area & CPU Effort Settings with the FPGA Express Software for information.
   3. Close the **Create Implementation** dialog box by choosing **OK**.

   The FPGA Express software processes each source file and determines the complete hierarchical structure and topology of the design, including multi-level links and references between subdesigns. With this information, the FPGA Express software produces an intermediate, unoptimized design implementation. The right-hand side of the Project window displays the implementation name and target device. The implementation icon also indicates any errors, warnings, or other information. To correct error or warning conditions, refer to step 5.

9. (Optional) Select the design implementation icon in the Chips window, press Button 2, and choose the **Edit Constraints** command from the pop-up menu to display the Altera-specific constraint tables. These constraint tables allow you to specify pin, logic option, and timing assignments for your design. All design-specific information, such as Clock names, port names, and design hierarchy assignments is extracted

automatically from the design. Altera recommends entering specific requirements directly into these tables to obtain the desired optimization. Refer to Entering Resource, Device & Global Logic Synthesis Assignments for information.

10. Optimize the design by selecting the design implementation in the Project window and choosing the **Optimize** button on the toolbar. A new optimized implementation icon appears beneath the original implementation icon. When you open the optimized implementation, the constraint tables are back-annotated with the optimization results. The FPGA Express software optimizes a design for either speed or area, based on the settings you specified in step 8.
11. Identify and optimize critical paths in your design with the Time Tracker static timing analyzer, as described in Analyzing Estimated Timing with the FPGA Express Time Tracker.
12. Generate a project report by selecting the optimized design implementation and clicking the **Report** icon on the toolbar. An FPGA Express project report documents the design through the synthesis and optimization design flow. The report includes information about design source data, constraints, and optimization options.
13. Generate MAX+PLUSÂ II-compatible EDIF netlist files by selecting the optimized design implementation and choosing the **Export Netlist** button on the toolbar. In the **Export** dialog box, specify the following options:
    1. Specify the name and location of the directory for the EDIF netlist files in the *Export Directory* box.
    2. Select the EDIF netlist file's output bus from the *Bus Style* drop-down list. The MAX+PLUSÂ II software accepts either flattened or unflattened buses. In the FPGA Express software, the default setting, *EXPAND*, flattens each bus by writing each bus bit as an individual I/O port. To export an EDIF netlist file without flattening the bus names, select any of the other settings, which include delimiters for different bus notations: `[]`, `<>`, `()`, and `{}`.
    3. If you wish to generate a VHDL or Verilog HDL netlist file for functional simulation prior to MAX+PLUSÂ II compilation, select a language option (*VHDL* or *Verilog*) from the *Output Format* drop-down list. Otherwise, select *NONE* for this option instead.
    4. Turn on the *Export Primitives* option to export VHDL or Verilog HDL primitives into the simulation netlist file. However, if the simulation is to be performed with an external library, turn the option off.
    5. Close the **Export** dialog box by choosing **OK**. The FPGA Express software creates the following MAX+PLUSÂ II-compatible files:
        - <*design name*>**.edf** (EDIF format)
        - <*design name*>**.acf**, an Assignment & Configuration File that contains design constraints
        - <*design name*>**.lmf**, a Library Mapping File that maps FPGA Express functions to MAX+PLUSÂ II functions
14. Copy all three types of output files (EDIF netlist file(s), ACF, and LMF) to a MAX+PLUSÂ II project directory. Process the <*design name*>**.edf** file with the MAX+PLUSÂ II Compiler, as described in Compiling Projects with MAX+PLUSÂ II Software.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Analyzing Estimated Timing with the FPGA Express Time Tracker

You can use the FPGA Express Time Tracker static timing analyzer to display estimated delays of critical paths in your project. This timing analyzer provides timing information and a detailed listing of critical paths.

To use the Time Tracker timing analyzer, follow these steps:

1. Select the design implementation icon in the Chips window, press Button 2, and choose the **View Results** commmand from the pop-up menu to display the Time Tracker tabs.
2. Analyze the timing of your design by viewing the different tables within the **Clocks**, **Paths**, and **Ports** Time Tracker tabs:
   - To analyze the Clock frequency ($\mathbf{f_{MAX}}$), select the **Clocks** tab. The table on the **Clocks** tab contains a column showing the actual Clock frequency for each Clock in your design next to the desired frequency derived from your timing constraints. Clocks that fail to meet their constraints are highlighted in red.
   - To check critical timing paths, select the **Paths** tab. The table on the **Paths** tab contains an **Est. Delay** column displaying path delays. Paths that fail to meet constraints are highlighted in red. You can select a path or path group to display additional tables with increasing detail, in order to identify exactly which paths failed to meet their timing constraints.
   - To view I/O port delays, select the **Ports** tab. The **Ports** tab displays the slack for each I/O port, i.e., the Clock period minus the propagation delay through the port in the *Input Slack* column for input ports and the *Output Slack* column for output ports. Negative values are highlighted in red, indicating that the propagation delay exceeds the Clock period, causing a timing violation.
3. If necessary, change the design logic or adjust your timing constraints as described in Assigning Pins, Logic Options, and $\mathbf{t_{SU}}$, $\mathbf{t_{CO}}$ & $\mathbf{t_{PD}}$ Timing Constraints, then re-optimize the design.
4. Continue with the steps necessary to process your design, as described in Synthesizing & Optimizing VHDL or Verilog HDL Files with FPGA Express Software.

Analyzing Estimated Timing with the FPGA Express Time Tracker

You can use the FPGA Express Time Tracker static timing analyzer to display estimated delays of critical paths in your project. This timing analyzer provides timing information and a detailed listing of critical paths.

To use the Time Tracker timing analyzer, follow these steps:

1. Select the design implementation icon in the Chips window, press Button 2, and choose the **View Results** commmand from the pop-up menu to display the Time Tracker tabs.
2. Analyze the timing of your design by viewing the different tables within the **Clocks**, **Paths**, and **Ports** Time Tracker tabs:
   - To analyze the Clock frequency ($\mathbf{f_{MAX}}$), select the **Clocks** tab. The table on the **Clocks** tab contains a column showing the actual Clock frequency for each Clock in your design next to the desired frequency derived from your timing constraints. Clocks that fail to meet their constraints are highlighted in red.
   - To check critical timing paths, select the **Paths** tab. The table on the **Paths** tab contains an **Est. Delay** column displaying path delays. Paths that fail to meet constraints are highlighted in red. You can select a path or path group to display additional tables with increasing detail, in order to identify exactly which paths failed to meet their timing constraints.

To view I/O port delays, select the **Ports** tab. The **Ports** tab displays the slack for each I/O port, i.e., the Clock period minus the propagation delay through the port in the *Input Slack* column for input ports and the *Output Slack* column for output ports. Negative values are highlighted in red, indicating that the propagation delay exceeds the Clock period, causing a timing violation.

3. If necessary, change the design logic or adjust your timing constraints as described in Assigning Pins, Logic Options, and t_{SU}, t_{CO} & t_{PD} Timing Constraints, then re-optimize the design.
4. Continue with the steps necessary to process your design, as described in Synthesizing & Optimizing VHDL or Verilog HDL Files with FPGA Express Software.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Using Cadence Tools with MAX+PLUS II Software

The following topics describe how to use a variety of Cadence tools as part of a complete design flow that includes the MAX+PLUS® II software. If you use only one Cadence tool, click List by Tool and select the tool name to view the list of topics only for that tool. Click on one of the following topics for information:

Open a printable version of all topics listed on this page.

Setting Up the MAX+PLUS II/Cadence Working Environment

- Software Requirements
- Setting Up the MAX+PLUS II/Cadence Concept Work Environment for a Sun SPARCstation Running SunOS Software
- MAX+PLUS II/Cadence Interface File Organization
- MAX+PLUS II Directory Structure
- Concept & RapidSIM Local Work Area Directory Structure
- Concept & HDL Direct Project Directory Structure
- Composer Project File Directory Structure
- Altera-Provided Logic & Symbol Libraries
- Compiling the VITAL Library for Use with Leapfrog Software
- Compiling the **alt_mf** Library

Design Flow for All Cadence Tools

Design Entry

- **Design Entry Flow**
- **Concept**
  - Creating Concept Schematics for Use with MAX+PLUS II Software
    - Instantiating the clklock Megafunction in Concept Schematics
    - Instantiating LPM & Other Parameterized Functions in Concept Schematics
  - Entering Resource Assignments
    - Assigning Pins, Logic Cells & Chips
    - Assigning Cliques
    - Assigning Logic Options
    - Modifying the Assignment & Configuration File with the **setacf** Utility
  - Performing a Functional Simulation of a Concept Schematic with the **hdlconfig** Utility & Verilog-XL Software
  - Performing a Functional Simulation of a Concept Schematic with VerilogLink & Verilog-XL Software
  - Creating Hierarchical Projects in Concept Schematics
  - Converting Concept Schematics into MAX+PLUS II-Compatible EDIF Netlist Files with the **concept2alt** Utility
- **Composer**

- Creating Composer Schematics for Use with MAX+PLUS II Software
- Entering Resource Assignments
  - Assigning Pins, Logic Cells & Chips
  - Assigning Cliques
  - Assigning Logic Options
  - Modifying the Assignment & Configuration File with the **setacf** Utility
- Performing a Functional Simulation of a Composer Schematic with Verilog-XL Software
- Creating Hierarchical Projects in Composer Schematics
- Converting Composer Schematics into MAX+PLUS II-Compatible EDIF Netlist Files with the **altout** Utility

- **VHDL**

  - Creating VHDL Designs for Use with MAX+PLUS II Software
    - Instantiating the clklock Megafunction in VHDL or Verilog HDL
  - Entering Resource Assignments
    - Modifying the Assignment & Configuration File with the **setacf** Utility

- **Verilog HDL**

  - Creating Verilog HDL Designs for Use with MAX+PLUS II Software
    - Instantiating the clklock Megafunction in VHDL or Verilog HDL
  - Entering Resource Assignments
    - Modifying the Assignment & Configuration File with the **setacf** Utility

Synthesis & Optimization

- **VHDL**

  - Synthesizing & Optimizing VHDL Files with Synergy Software
  - Converting VHDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the **vlog2alt** or **altout** Utility

- **Verilog HDL**

  - Synthesizing & Optimizing Verilog HDL Files with Synergy Software
  - Converting Verilog HDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the **vlog2alt** Utility

Compilation

- Project Compilation Flow
- Compiling Projects with MAX+PLUS II Software

Simulation

- Project Simulation Flow
- Initializing Registers in VHDL & Verilog Output Files for Power-Up before Simulation
- Performing a Timing Simulation with RapidSIM Software
- Performing a Timing Simulation with Verilog-XL Software
- Performing a Timing Simulation with Leapfrog Software
  - Compiling the VITAL Library for Use with Leapfrog Software

- Compiling the **alt_mf** Library

Device Programming

- Programming Altera Devices

Related Links:

- MAX+PLUS II Development Software
- Altera Programming Hardware
- Cadence web site (http://www.cadence.com)

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Using Cadence Leapfrog & MAX+PLUS II Software

The following topics describe how to use the Cadence Leapfrog software with the MAX+PLUS® II software. Click on one of the following topics for information:

This file is suitable for printing only. It does not contain hypertext links that allow you to jump from topic to topic.

Setting Up the MAX+PLUS II/Cadence Working Environment

- Software Requirements
- MAX+PLUS II Directory Structure
- MAX+PLUS II/Cadence Interface File Organization
- Compiling the VITAL Library for Use with Leapfrog Software
- Compiling the **alt_mf** Library

Simulation

- Project Simulation Flow
- Initializing Registers in VHDL & Verilog Output Files for Power-Up before Simulation
- Performing a Timing Simulation with Leapfrog Software

Related Topics:

Go to the following MAX+PLUS II ACCESS℠ Key topics for related information:

- Compiling Projects with MAX+PLUS II Software
- Programming Altera Devices

Go to the following topics, which are available on the web, for additional information:

- MAX+PLUS II Development Software
- Altera Programming Hardware
- Cadence web site (http://www.cadence.com)

Setting Up the MAX+PLUS II/Cadence Working Environment

To use MAX+PLUS® II software with Cadence software, you must first install the MAX+PLUS II software, then establish an environment that facilitates entering and processing designs. The MAX+PLUS II/Cadence interface is installed automatically when you install the MAX+PLUS II software on your computer. Go to *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual for more information on installation and details on the directories that are created during MAX+PLUS II installation. Go to MAX+PLUS II/Cadence Interface File

Organization for information about the MAX+PLUS II/Cadence directories that are created during MAX+PLUS II installation.

The information presented here assumes that you are using the C shell and that your MAX+PLUS II system directory is **/usr/maxplus2**. If not, you must use the appropriate syntax and procedures to set environment variables for your shell.

To set up your working environment for the MAX+PLUS II/Cadence interface, follow these steps:

1. Ensure that you have correctly installed the MAX+PLUS II and Cadence software versions described in the MAX+PLUS II/Cadence Software Requirements.

2. Add the following environment variables to your **.cshrc** file:

   ```
   setenv ALT_HOME /usr/maxplus2 ↵

   setenv CDS_INST_DIR <Cadence system directory path> ↵
   ```

3. Add the **$ALT_HOME/cadence/bin** and **$CDS_INST_DIR/tools/bin** directories to the `PATH` environment variable in your **.cshrc** file. Make sure these paths are placed before the Cadence hierarchy path.

4. Add **/usr/dt/lib** and **/usr/ucb/lib** to the `LD_LIBRARY_PATH` environment variable in your **.cshrc** file.

5. Create a new **cds.lib** file in your working directory or edit an existing one so that it includes all of the following lines that apply to the Cadence tools you have installed:

   ```
   DEFINE alt_syn ${ALT_HOME}/simlib/concept/alt_syn

   DEFINE lpm_syn ${ALT_HOME}/simlib/concept/lpm_syn

   DEFINE alt_lpm ${ALT_HOME}/simlib/concept/alt_lpm

   DEFINE alt_mf ${ALT_HOME}/simlib/concept/alt_mf

   DEFINE alt_max2 ${ALT_HOME}/simlib/concept/alt_max2

   DEFINE alt_max2 ${ALT_HOME}/simlib/composer/alt_max2/alt_max2

   DEFINE alt_vtl $ALT_HOME/simlib/concept/alt_vtl/lib

   DEFINE altera $ALT_HOME/simlib/concept/alt_mf/lib

   SOFTINCLUDE $CDS_INST_DIR/tools/leapfrog/files/cds.lib

   DEFINE <design name>.
   ```

6. Copy the **/usr/maxplus2/maxplus2.ini** file to your **$HOME** directory:

   ```
   cp /usr/maxplus2/maxplus2.ini $HOME ↵

   chmod u+w $HOME/maxplus2.ini ↵
   ```

   The **maxplus2.ini** file contains both Altera- and user-specified initialization parameters that control the MAX+PLUS II software, such as MAX+PLUS II symbol and logic function library paths and the current project name. The MAX+PLUS II installation procedure creates and copies the **maxplus2.ini** file to the **/usr/maxplus2** directory.

Normally, you do not have to edit your local copy of **maxplus2.ini** because the MAX+PLUS II software updates the file automatically whenever you change any parameters or settings. However, if you move the **max2lib** and **max2inc** library subdirectories, you must update the file. Go to "Creating & Using a Local Copy of the **maxplus2.ini** File" in MAX+PLUS II Help for more information.

7. If you are using Concept on a Sun SPARCstation running SunOS, go to Setting Up the MAX+PLUS II/Cadence Concept Work Environment for a Sun SPARCstation Running SunOS Software to install the **redifnet** EDIF netlist reader utility.

8. If you are using Synergy software, edit the **hdl.var** file located in your working directory to include the following line:

   `DEFINE work` *<design name>* ↵

9. Set up an appropriate directory structure for the tool(s) you are using. See the following topics for information:

   - Composer Project File Directory Structure
   - Concept & RapidSIM Local Work Area Directory Structure

Related Topics:

Go to the following topics, which are available on the web, for additional information:

- ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
- This manual is also available in 4 parts:
  - Preface & Section 1: MAX+PLUS II Installation
  - Section 2: MAX+PLUS II - A Perspective
  - Section 3: MAX+PLUS II Tutorial
  - Appendices, Glossary & Index

MAX+PLUS II/Cadence Software Requirements

The following table shows the software applications that are used to generate, process, synthesize, and verify a project with MAX+PLUS® II and Cadence software:

	Cadence	Altera
version 97A: Concept Composer ValidCOMPILER **concept2alt** **vlog2alt** **altout**	VerilogLink Synergy HDL Direct (Concept 2.0 or later) Non-Graphic Simulation Environment (SE) RapidSIM, Verilog-XL, or Leapfrog **redifnet** (SunOS only)	MAX+PLUS II version 9.4

☞ The MAX+PLUS II **read.me** file provides up-to-date information on which versions of Cadence software applications are supported by the current version of MAX+PLUS II. It also provides information on installation and operating requirements. You should read the **read.me** file on the CD-ROM before installing the MAX+PLUS II software. After installation, you can open the **read.me** file from the MAX+PLUS II Help menu.

MAX+PLUS II Directory Structure

In the MAX+PLUS® II software, a project name is the name of a top-level design file, without the filename extension. This design file can be an EDIF, Verilog HDL, or VHDL netlist file; an AHDL Text Design File (TDF); or any other MAX+PLUS II-supported design file. The EDIF netlist file must be created by the **altout** or **concept2alt** utility and imported into the MAX+PLUS II software as an EDIF Input File (**.edf**).

Project design files and output files are stored in the project directory, with the exception of standard library functions provided by Altera or another EDA tool vendor. The MAX+PLUS II software stores the connectivity data on the links between design files in a hierarchical project in a Hierarchy Interconnect File (**.hif**), but refers to the entire project only by its project name. The MAX+PLUS II Compiler uses the HIF to build a single, fully flattened project database that integrates all design files in a project hierarchy.

MAX+PLUS II/Cadence Interface File Organization

Table 1 shows the MAX+PLUS® II/Cadence interface subdirectories that are created in the MAX+PLUS II system directory (by default, the **/usr/maxplus2** directory) during MAX+PLUS II installation. For information on the other directories that are created during MAX+PLUS II installation, see "MAX+PLUS II File Organization" in *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual.

Table 1. MAX+PLUS II Directory Organization

Directory	Description
./lmf	Contains the Altera-provided Library Mapping File, **cadence.lmf**, that maps Cadence logic functions to equivalent MAX+PLUS II logic functions.
./examples/cadence	Contains the sample files for Cadence software discussed in these ACCESS℠ Key Guidelines.
./cadence	Contains the AMPLE userware for the MAX+PLUS II/Cadence interface.
./simlib/concept/alt_max2	Contains the MAX+PLUS II primitives, including `CARRY`, `CASCADE`, `EXP`, `GLOBAL`, `LCELL`, `SOFT`, `OPNDRN`, `DFFE` (D flipflop with Clock Enable), and `DFFE6K` (D flipflop with Clock Enable and both Clear and Preset for FLEX® 6000 devices only) for use with Concept software.
./simlib/composer/alt_max2	Contains the MAX+PLUS II primitives, including `CARRY`, `CASCADE`, `EXP`, `GLOBAL`, `LCELL`, `SOFT`, `OPNDRN`, `DFFE` (D flipflop with Clock Enable), and `DFFE6K` (D flipflop with Clock Enable and both Clear and Preset for FLEX 6000 devices only) for use with Composer software.
./simlib/concept/alt_lpm	Contains the MAX+PLUS II megafunctions, including library of parameterized modules (LPM) functions, for use with Concept software.
./simlib/concept/max2sim	Contains the MAX+PLUS II/Concept simulation model library, **max2_sim**, for use with RapidSIM software.
./simlib/concept/alt_syn	Contains the MAX+PLUS II synthesis library, **alt_syn**, for use with Synergy and Concept software, and the **vlog2alt** utility.
./simlib/composer/alt_syn	Contains the MAX+PLUS II synthesis library, **alt_syn**, for use with Synergy and Composer software.
./simlib/concept/lpm_syn	Contains the Cadence LPM library, **lpm_syn**, for use with Synergy and Concept software.
./simlib/composer/lpm_syn	Contains the Cadence LPM library, **lpm_syn**, for use with Synergy and Composer software.
	Contains the MAX+PLUS II VHDL logic function library. (`a_8count` is for

./simlib/concept/alt_mf	the MAX® 7000 and MAX 9000 device families only.)
./simlib/concept/edifnet/templates	Contains template files for Concept directives, i.e., **global.cmd**, **compiler.cmd**, **vloglink.cmd**, **verilog.cmd**, and **master.local**.
./simlib/concept/alt_max2/verilogUdps	Contains Verilog HDL modules that are the equivalent of the primitives contained in **alt_max2** library for use with Concept software.
./simlib/composer/alt_max2/verilogUdps	Contains Verilog HDL modules that are the equivalent of the primitives contained in **alt_max2** library for use with Composer software.
./simlib/concept/alt_vtl **./simlib/composer/alt_vtl**	Contains VITAL library source files for use with Concept or Composer software.
./simlib/composer/alt_max2/verilog	Contains simulation modules for all symbols in the **alt_max2** Composer library.

Related Topics:

Go to the following topics, which are available on the web, for additional information:

- ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
- This manual is also available in 4 parts:
  - Preface & Section 1: MAX+PLUS II Installation
  - Section 2: MAX+PLUS II - A Perspective
  - Section 3: MAX+PLUS II Tutorial
  - Appendices, Glossary & Index
- FLEX Devices
- MAX Devices
- Classic Device Family

Compiling the VITAL Library for Use with Leapfrog Software

If you wish to use MAX+PLUS® II-generated Standard Delay Format (SDF) Output Files (**.sdo**) that contain timing information when performing post-compilation timing simulation with Leapfrog software, you must first compile the VITAL library source files. The VITAL Timing and Primitive package files are located in the **$CDS_INST_DIR/tools/leapfrog/files/IEEE.src** directory.

To compile the **alt_vtl** library, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment. For example, you must ensure that the appropriate directories are specified in the **cds.lib** file that is located in your working directory.

2. Create a VHDL design, as described in Creating VHDL Designs for Use with MAX+PLUS II Software and save it in your working directory.

3. Change to the **alt_vtl** directory by typing `cd /usr/maxplus2/simlib/concept/alt_vtl` ↵ at the UNIX prompt.

4. Edit the **hdl.var** file located in your working directory to include the following line:

   ```
   DEFINE WORK alt_vtl ↵
   ```

5. Create the **/usr/maxplus2/simlib/concept/alt_vtl/lib** directory.

6. Type the following commands at the UNIX prompt from the **/usr/maxplus2/simlib/concept/alt_vtl** directory to compile the library:

```
cv -message -file alt_vtl.vhd↵
cv -message -file alt_vtl.cmp↵
```

Compiling the alt_mf Library

If your VHDL design uses functions from the **alt_mf** library, you must compile this library. To compile the **alt_mf** library, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS® II/Cadence Working Environment. For example, you must ensure that the appropriate directories are specified in the **cds.lib** file located in your working directory.
2. Change to the **alt_mf** directory by typing `cd /usr/maxplus2/simlib/concept/alt_mf↵` at the UNIX prompt.
3. Edit the **hdl.var** file located in your working directory to include the following line:

   ```
   DEFINE work alt_mf↵
   ```

4. Type the following commands at the UNIX prompt from the **/usr/maxplus2/simlib/concept/alt_mf** directory to compile the library:

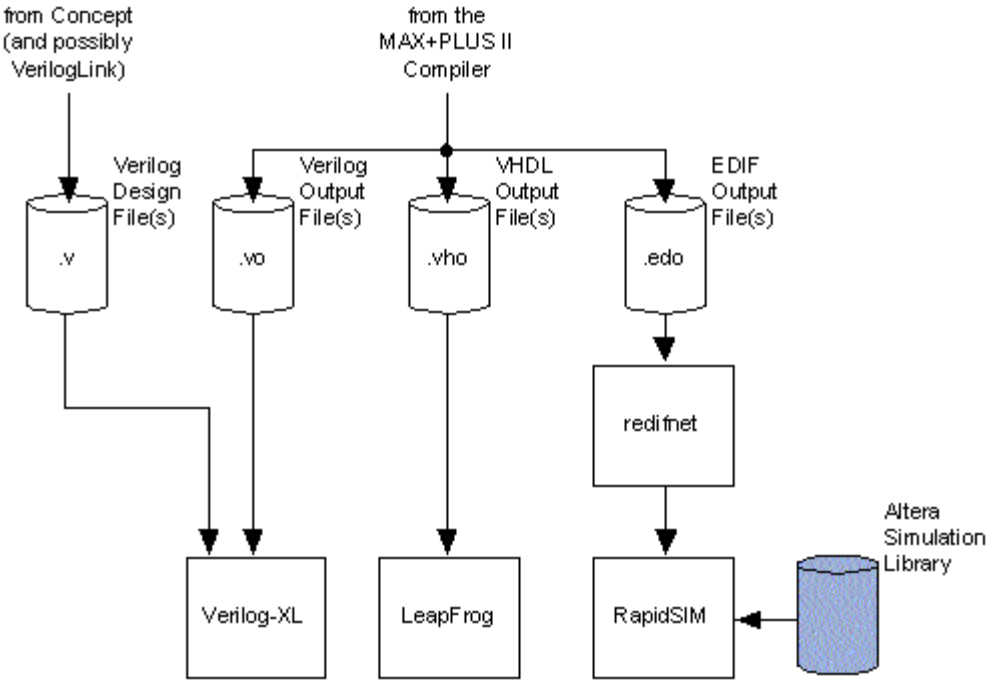
   ```
   cv -message -file ./src/mf.vhd↵
   cv -message -file ./src/mf_components.vhd↵
   ```

Project Simulation Flow

Figure 1 shows the project simulation flow for the MAX+PLUS® II/Cadence interface.

Figure 1. MAX+PLUS II/Cadence Project Simulation Flow

Altera-provided items are shown in blue.



Initializing Registers in VHDL & Verilog Output Files for Power-Up before Simulation

Altera provides the **add_dc** script, which is availiable in the MAX+PLUS II system directory, to allow you to process MAX+PLUS II-generated Verilog Output Files (**.vo**) and VHDL Output Files (**.vho**) to prepare these files for simulation with another EDA tool. The add_dc script runs the **add_dclr** utility, which inserts a `device_clear` signal that is used for power-up initialization of all registers or flipflops in the design.

The script adds in a top-level signal named `device_clear` and connects it to the `CLRN` pin in all flipflops that should initialize to 0, and to the `PRN` pin of all flipflops that should initialize to 1. If the `CLRN` or `PRN` pin of a flipflop is already being used (i.e., is already connected to a signal), the script modifies the Verilog Output File or VHDL Output File so that the `AND` of the original signal and the `device_clear` pin feed the `CLRN` or `PRN` pin.

To use the **add_dc** script to process Verilog Output Files and VHDL Output Files before simulation with another EDA tool, follow these steps:

1. Make sure that your design file is located in the current directory, or change to the directory in which the design file is located.

2. Type the following command at the command prompt:

   ¥<*path name of* ***add_dc.bat*** *file*>¥add_dc <*design name*> <*path name of* ***add_dclr.exe*** *file*> ↵

For example, if the both the **add_dc.bat** and the **add_dclr.exe** files are located in the **d:¥maxplus2¥exew** directory, and the **d:¥maxplus2¥exew** directory is specified in the search path, you can type the following command at a command prompt to add a `device_clear` signal to a design named `myfifo` in the file **myfifo.vo**:

```
add_dc myfifo d:¥maxplus2¥exew ↵
```

1. The **add_dc** script gives a message if the directory contains both a VHDL Output File and a Verilog Output File with the same name (<*design name*>**.vo** and <*design*>**.vho**). You should delete or rename whichever of those files should not have the `device_clear` signal added. The **add_dc** script can modify only one design file at a time.

2. When the **add_dc** script processes the Verilog Output File or VHDL Output File, it creates a backup copy of the original file, with the extension **.ori**.

3. The **add_dc** script works only for Verilog Output Files and VHDL Output Files that are generated by MAX+PLUS II.

After you have used the **add_dc** script and are ready to simulate the resulting Verilog Output File or VHDL Output File with another EDA tool, you should assert the active low `device_clear` pin for a period of time that is long enough for the design to initialize. You can then de-assert the pin, and apply simulation vectors to the design.

Performing a Timing Simulation with Leapfrog Software

Once the MAX+PLUS® II software has compiled a project and generated a VHDL Output File (**.vho**), you can a perform timing simulation using Cadence Leapfrog software.

To simulate a VHDL output file with the Leapfrog timing simulator, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment.

2. If you wish to use MAX+PLUS II-generated Standard Delay Format (SDF) Output Files (**.sdo**) that contain timing information, compile the VITAL library source files, as described in Compiling the VITAL Library for Use with Leapfrog Software.

3. If your design uses functions from the **alt_mf** library, compile the library, as described in Compiling the **alt_mf** Library.

4. Generate a VHDL Output File (**.vho**) and an optional SDF Output File, as described in Compiling Projects with MAX+PLUS II Software.

5. Using any standard text editor, create a stimulus file that includes test vectors for <*design name*>.

6. Start the Leapfrog simulator and simulate the MAX+PLUS II-created VHDL Output File <*design name*>**.vho** by typing `leapfrog` ↵ at the UNIX prompt. Refer to *Chapter 5: SDF Back-Annotation in Leapfrog* in the ***VHDL Simulator User Guide*** or refer to the ***Cadence Openbook*** for more information.

Compiling Projects with MAX+PLUS II Software

The MAX+PLUS® II Compiler can process design files in a variety of formats. This topic describes how to use MAX+PLUS II software to compile projects in which the top-level design file is an EDIF Input File (with the extension **.edf**).

Related Topics:

- Go to MAX+PLUS II Help for information on compiling VHDL and Verilog HDL, design files directly with the MAX+PLUS II Compiler.

- Go to Running Synopsys Compilers from MAX+PLUS II Software for information on running the Synopsys Design Compiler or FPGA Compiler software on a VHDL or Verilog HDL design from within the MAX+PLUS II Compiler window.

To compile a design (also called a "project") with MAX+PLUS II software, go through the following steps:

1. Create design files that are compatible with the MAX+PLUS II software and convert them into EDIF Input Files with the extension **.edf**. Specific instructions for some tools are described in these MAX+PLUS II ACCESS^{SM} Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your design entry or synthesis and optimization tool.

2. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, create a mapping for each function in a Library Mapping File (**.lmf**) to map the custom symbol to the corresponding EDIF Input File, AHDL Text Design File (**.tdf**), or other MAX+PLUS II-supported design file. These custom functions are represented in design files as hollow-body symbols or "black box" HDL descriptions.

   ☞ Go to "Library Mapping Files (**.lmf**)" in MAX+PLUS II Help for more information.

3. Open MAX+PLUS II and specify the name of your top-level design file as the project name with the **Project Name** command (File menu). If you open an HDL file in the MAX+PLUS II Text Editor, you can choose the **Project Set Project to Current File** command (File menu) instead.

   ☞ You can also compile a project from a command line. However, the first time you compile a project, the settings you need to specify are easier to specify from within the MAX+PLUS II software. After you have run the graphical user interface for the MAX+PLUS II software at least once, you can more easily use the command-line **setacf** utility to modify options in the Assignment & Configuration File (**.acf**) for the project. Type `setacf -h` ↵ and `maxplus2 -h` ↵ for descriptions of **setacf** and MAX+PLUS II command-line syntax.

4. Choose **Device** (Assign menu) and select the target Altera device family in the *Device Family* drop-down list box. If you wish to implement the design logic in a specific device, select it in the *Devices* box. Otherwise, select *AUTO* to allow the MAX+PLUS II Compiler to choose the best device(s) in the current device family. If your design entry or synthesis and optimization tool required you to specify a target family and/or device, specify the same information in this dialog box. For information on partitioning logic among multiple devices, go to MAX+PLUS II Help. Choose **OK**.

5. Open the Compiler window by choosing the **Compiler** command (MAX+PLUS II menu). Go through the following steps to specify the options necessary to compile the design file(s) in your project:

   1. Ensure that all EDIF netlist files have the extension **.edf** and choose **EDIF Netlist Reader Settings** (Interfaces menu).

   2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor. This name should be the name of the vendor whose tool(s) you used to create the EDIF netlist files. If your vendor name does not appear, select *Custom* instead.

      ☞ If you are compiling a design created with Synopsys FPGA Express software, select *Synopsys*, choose the **Customize** button, enter <*project name*>**.lmf** in the *LMF #1* box, choose **OK**, and skip to step 6.

   3. If you selected an existing vendor name in the *Vendor* box and your project contains design files that require custom LMF mappings, choose the **Customize** button to expand the dialog box to show all settings. Turn on the *LMF #2* checkbox and type your custom LMF's filename in the corresponding text box, or select a name from the *Files* box. The selection in the *Vendor* box will change to *Custom* and all settings will be retained until you change them again.

   4. If you selected *Custom* in the *Vendor* box, choose the **Customize** button to expand the dialog box to show all settings. Any previously defined custom settings will be displayed. Under *Signal Names*, type one or more names with up to 20 total name characters in the *VCC* or *GND* box if your EDIF Input File(s) use one or more names other than `VCC` or `GND` for the global high or low signals. Multiple signal names must be separated by either a comma (,) or a space. Under *Library Mapping Files*, turn on the

LMF #1 checkbox and type a filename in the text box following it, or select a name from the *Files* box. If necessary, specify another LMF name in the *LMF #2* box. Go to MAX+PLUS II Help for detailed information on the settings available in the **EDIF Netlist Reader Settings** dialog box.

5. Choose **OK**.

6. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, you may need to ensure that all files are present in your project directory, i.e., the same directory as the top-level design file. Otherwise, you must specify the directories containing these files as user libraries with the **User Libraries** command (Options menu).

7. Follow all guidelines that apply to your design entry or synthesis and optimization tool:

   - Exemplar Logic Galileo Extreme-Specific Compiler Settings
   - Synopsys DesignWare-Specific Compiler Settings
   - Converting Synopsys FPGA Compiler & Design Compiler Timing Constraints into MAX+PLUS II-Compatible Format with the **syn2acf** Utility
   - Synplicity Synplify-Specific Compiler Settings

8. If you wish to generate EDIF, VHDL, or Verilog HDL output files for post-compilation simulation or timing analysis with another EDA tool, go through the following steps:

   1. (Optional) Turn on the **Optimize Timing SNF** command (Processing menu) to reduce the size of the output file(s). Turning on this command can reduce the size of output netlists by up to 30%.

      This command does not create optimized timing SNFs on UNIX workstations. However, a non-optimized timing SNF provides the same functional and timing information as an optimized timing SNF.

   2. If you wish to generate EDIF Output Files (**.edo**), go through these steps:

      1. Turn on the **EDIF Netlist Writer** command (Interfaces menu). Then choose the **EDIF Netlist Writer Settings** command (Interfaces menu).

      2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor and choose **OK**. If your vendor name does not appear, select *Custom* instead and specify the settings that are appropriate for your simulation or timing analysis tool. Go to MAX+PLUS II Help for detailed information on the options available in the **EDIF Netlist Writer Settings** dialog box.

      3. To generate an optional Standard Delay Format (SDF) Output File (**.sdo**), choose the **Customize** button to expand the dialog box to show all settings. Select one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**.

      The filenames of the EDIF Output File(s) and optional SDF Output File(s) are the same as the user-defined chip name(s) for the project; if no chip names exist, the Compiler assigns filenames that are based on the project name. For a multi-device project, the Compiler also generates a top-level EDIF Output File that is uniquely identified by "**_t**" appended to the project name. In addition, the Compiler automatically generates a VHDL Memory Model Output File, <*project name*>**.vmo**, when it generates an EDIF Output File that contains memory (RAM or ROM).

   3. If you wish to generate VHDL Output Files (**.vho**), turn on the **VHDL Netlist Writer** command (Interfaces menu). Then choose **VHDL Netlist Writer Settings** command (Interfaces menu). Select *VHDL Output File (.vho)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF ver. 2.1 files contain timing delay information that allows you to perform back-annotation simulation in VHDL with VITAL-compliant simulation libraries. The VHDL Output Files generated by the Compiler have the extension **.vho**, but are otherwise named in the same way as the

EDIF Output Files described above.

4. If you wish to generate Verilog HDL Output Files (**.vo**), turn on the **Verilog Netlist Writer** command (Interfaces menu). Then choose **Verilog Netlist Writer Settings** command (Interfaces menu). Select *Verilog Output File (.vo)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF Output Files contain timing delay information that allows you to perform back-annotation simulation in Verilog HDL. The Verilog Output Files generated by the Compiler have the extension **.vo**, but are otherwise named in the same way as the EDIF Output Files described above.

9. To run the MAX+PLUS II Compiler, choose the **Project Save & Compile** command (File menu) or choose the **Start** button in the Compiler window.

   ☞ See step 3 for information on running MAX+PLUS II software from the command line.

10. Once you have compiled the project with the MAX+PLUS II Compiler, you can use the VHDL, Verilog HDL, or EDIF output file(s), and the optional SDF Output File(s) (**.sdo**) to perform timing analysis or timing simulation with another EDA tool. Specific instructions for some tools are described in these MAX+PLUS II ACCESS Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your EDA tool.

The MAX+PLUS II Compiler also generates a Report File (**.rpt**), a Pin-Out File (**.pin**), and one or more of the following files for device programming or configuration:

- JEDEC Files (**.jed**)
- Programmer Object Files (**.pof**)
- SRAM Object Files (**.sof**)
- Hexadecimal (Intel-format) Files (**.hex**)
- Tabular Text Files (**.ttf**)

Related Topics:

- Go to *Compiler Procedures* in MAX+PLUS II Help for information on other available Compiler settings.
- Go to *Programmer Procedures* in MAX+PLUS II Help for instructions on creating other types of programming files and on programming or configuring Altera devices.
- Go to Back-Annotating MAX+PLUS II Pin Assignments to Design Architect Symbols for information on back-annotating pin assignments in Mentor Graphics Design Architect schematics.
- Go to Programming Altera Devices for information on the different programming hardware options for Altera device families.

Go to the following topics, which are available on the web, for additional information:

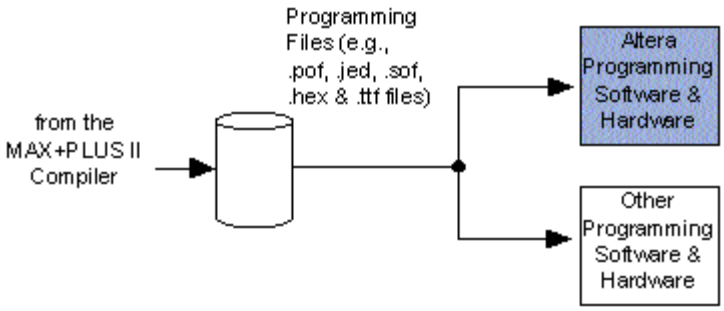
- MAX+PLUS II Development Software
- Altera Programming Hardware

Programming Altera Devices

Once you have successfully compiled and simulated a project with the MAX+PLUS® II software, you can program an Altera® device and test it in the target circuit. Figure 1 shows the device programming flow for MAX+PLUS II software.

Figure 1. MAX+PLUS II Device Programming Flow

Altera-provided items are shown in blue.



You can program devices with Altera programming hardware and MAX+PLUS II Programmer software installed on a 486- or Pentium-based PC or a UNIX workstation, or with programming hardware and software available from other manufacturers. Table 1 shows the available Altera programming hardware options on PCs and UNIX workstations.

Table 1. Altera Programming Hardware

Programming Hardware Option	PCs	UNIX Work-stations	MAX® 3000A Devices	Classic® & MAX 5000 Devices	MAX 7000 & MAX 7000E Devices	MAX 7000A, MAX 7000AE, MAX 7000B, MAX 7000S MAX 9000 & MAX 9000A Devices	FLEX® 6000, FLEX 6000A, FLEX 8000, FLEX 10K, FLEX 10KA, FLEX 10KB, & FLEX 10KE Devices	In-System Programming/ Configuration
Logic Programmer card, PL-MPU Master Programming Unit, and device-specific adapters	✓		✓	✓	✓	✓		
BitBlaster™ Download Cable	✓	✓	✓			✓	✓	✓
ByteBlasterMV™ Download Cable	✓		✓			✓	✓	✓
MasterBlaster™ Download Cable	✓	✓	✓			✓	✓	✓

If you wish to transfer programming files from a UNIX workstation to a PC over a network with File Transfer Protocol (FTP) or other similar transfer programs, be sure to select binary transfer mode.

Programming hardware from other manufacturers varies, but typically consists of a device connected to one of the serial ports on the workstation. Various vendors, such as Data I/O and BP Microsystems, supply hardware and

software for programming Altera devices.

Related Topics:

- Go to Compiling Projects with MAX+PLUS II Software for information on creating programming files.

Go to the following topics, which are available on the web, for additional information:

- MAX+PLUS II Development Software
- Altera Programming Hardware
- FLEX Devices
- MAX Devices
- Classic Device Family

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

ACCESS Partner EDA Tools, Listed by Function

Click on one of the following tool names for information on using it with the MAX+PLUS® II software:

Design Entry

- Composer (Cadence)
- Concept (Cadence)
- Design Architect (Mentor Graphics)

Synthesis & Optimization

- Certify (Synplicity)
- Design Compiler (Synopsys)
- FPGA Compiler (Synopsys)
- FPGA Express (Synopsys)
- Galileo Extreme (Exemplar Logic)
- Leonardo (Exemplar Logic)
- Synergy (Cadence)
- Synplify (Synplicity)

Simulation

- Design Viewpoint Editor (see QuickSim II)
- Leapfrog (Cadence)
- QuickHDL and QuickHDL Pro (Mentor Graphics)
- QuickSim II (Mentor Graphics)
- RapidSIM (Cadence)
- Verilog-XL (Cadence)
- VHDL System Simulator [VSS] (Synopsys)

Timing Analysis/Verification

- MOTIVE and MOTIVE for Powerview (Viewlogic)
- PrimeTime (Synopsys)
- QuickPath (Mentor Graphics)

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Performing a Functional Simulation with QuickHDL Software

You can use Mentor Graphics QuickHDL software to functionally simulate VHDL or Verilog HDL design files before compiling them with the MAX+PLUS ® II Compiler.

☞ If you wish to functionally simulate a hierarchical design that uses multiple design entry methods, you should use QuickHDL Pro rather than QuickHDL. Refer to Performing a Functional Simulation with QuickHDL Pro Software for more information.

To functionally simulate a VHDL or Verilog HDL design, follow these steps:

1. Be sure to set up the working environment correctly, as described in Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment.
2. Create a VHDL or Verilog HDL design file that follows the guidelines described in Creating VHDL & Verilog HDL Designs for Use with MAX+PLUS II Software.
3. Start Design Architect by double-clicking Button 1 on the **max_da** icon in the Design Manager tools window. You can also start Design Architect software by typing `max2_da` ↵ at the UNIX prompt.
4. Choose **Lib** (QuickHDL menu) and specify your work library name as the *Work Library* name. Choose **OK**.
5. Choose **Map** (QuickHDL menu) to map the instantiated function to the equivalent function in the **Altera** logic function library. Choose **Set** to specify *altera* as the *Logical Name* and *$MAX2_MFLIB* as the *Physical Name*. Choose **OK**.
6. Choose **Compile** (QuickHDL menu) and use the Navigator window to select the icon for your project. Specify your work library name as the *Work Library* name and select the *Simulation* setting in the Set VHDL Compilation Options or Set Verilog HDL Compilation Options window. Choose **OK** to compile.
7. Choose **Simulate** (QuickHDL menu) and specify your work library name as the *Work Library* name. Choose **OK** to start the QuickHDL Startup window.
8. Select the icon for your project in the Entity Configuration window and choose **OK** to simulate the design.
9. Synthesize and optimize the design, as described in Synthesizing & Optimizing VHDL & Verilog HDL Projects with Galileo Extreme Software or Synthesizing & Optimizing VHDL & Verilog HDL Projects with Leonardo Software.

If your Verilog HDL design uses memory functions (RAM or ROM) that can be initialized with a hexadecimal file (Intel-format) initialization, you must convert these files into Verilog HDL format using the Programming Language Interface (PLI). To use the Altera-provided source code for PLI, perform the following steps:

1. Download the file **http://www.edif.org/lpmweb/convert_hex2ver.c** to your project directory.
2. Copy the following two files from the **$MGC_HOME/shared/pkgs/quickhdl/include** directory into the **/usr/maxplus2** directory:

- **$MGC_HOME/shared/pkgs/quickhdl/include/veriuser**
- **$MGC_HOME/shared/pkgs/quickhdl/include/acc_user**

Refer to the ***Mentor Graphics QuickHDL User's Reference Manual***, version 8.5-4.6i, for information on compiling the PLI application on different platforms and using the Verilog HDL PLI.

Related Links:

- Go to the following MAX+PLUS II ACCESS^{SM} Key topics for related information:
  - Compiling Projects with MAX+PLUS II Software
  - Performing a Timing Simulation with QuickHDL Software
  - Performing a Functional Simulation with QuickHDL Pro Software

---

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Using Exemplar Logic Galileo Extreme & MAX+PLUS II Software

The following topic describes how to use the Exemplar Logic Galileo Extreme software with MAX+PLUS® II software. Click on one of the following topics for information:

This file is suitable for printing only. It does not contain hypertext links that allow you to jump from topic to topic.

Setting Up the MAX+PLUS ® II/Mentor Graphics/Exemplar Logic Working Environment

- Software Requirements
- Altera-Provided Logic & Symbol Libraries
- Local Work Area Directory Structure
- Mentor Graphics Project Directory Structure
- MAX+PLUS II Project Directory Structure
- MAX+PLUS II/Mentor Graphics/Exemplar Logic Interface File Organization

Design Entry

- Design Entry Flow
- Creating VHDL & Verilog HDL Designs for Use with MAX+PLUS II Software
- Performing a Functional Simulation with QuickHDL Pro Software

Synthesis & Optimization

- Synthesizing & Optimizing VHDL & Verilog HDL Projects with Galileo Extreme Software

Related Topics:

- Go to the following MAX+PLUS II ACCESS℠ Key topics for related information:
  - Compiling Projects with MAX+PLUS II Software
  - Programming Altera Devices
- Go to the following topics, which are available on the web, for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware
  - Exemplar Logic web site (http://www.exemplar.com)
  - Mentor Graphics web site (http://www.mentor.com)

Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment

To use the MAX+PLUS ® II software with Mentor Graphics/Exemplar Logic software, you must install the MAX+PLUS II software, then establish an environment that facilitates entering and processing designs. The MAX+PLUS II/Mentor Graphics/Exemplar Logic interface is installed automatically when you install the MAX+PLUS II software on your computer.

Go to *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual for more information on installation and details on the directories that are created during MAX+PLUS II installation. Go to MAX+PLUS II/Mentor Graphics/Exemplar Logic Interface File Organization for information about the MAX+PLUS II/Mentor Graphics directories that are created during MAX+PLUS II installation.

☞ The information presented here assumes that you are using a C shell and that your MAX+PLUS II system directory is **/usr/maxplus2**. If not, you must use the appropriate syntax and procedures to set environment variables for your shell.

To set up your working environment for the MAX+PLUS II/Mentor Graphics interface, follow these steps:

1. Ensure that you have correctly installed the MAX+PLUS II and Mentor Graphics software versions described in MAX+PLUS II/Mentor Graphics Software Requirements.
2. Add the following environment variables to your **.cshrc** file:

   `setenv ALT_HOME /usr/maxplus2` ↵

   `setenv MGC_WD` <*user-specified working directory*> ↵

   `setenv MGC_HOME` <*Mentor Graphics system directory*> ↵

   `setenv MAX2_MENTOR /usr/maxplus2/mentor/max2` ↵

   `setenv MGC_LOCATION_MAP` <*user-specified location_map file*> ↵

   `setenv EXEMPLAR` <*Galileo or Leonardo system directory*> ↵

   ☞ Installing the Altera® provided MAX+PLUS II/Mentor Graphics interface on your computer automatically installs a template for these environment variables in the **/usr/maxplus2/mentor/max2/.cshrc** file.
3. Add the **$MGC_HOME/bin**, **$MAX2_MENTOR/bin**, **$ALT_HOME/bin**, **$EXEMPLAR/bin/**<*os*>, and **$ALT_HOME/bin** directories to the `PATH` environment variable in your **.cshrc** file, where <*os*> is the operating system, e.g., `SUN4` for SunOS; `SUN5` for Solaris.
4. If you plan to use the Altera Schematic Express (**sch_exprss**) utility or the Altera VHDL Express (**vhd_exprss**) utility, add the following environment variable to your **.cshrc** file:

   `setenv MAX2_QSIM /usr/maxplus2/simlib/mentor/max2sim` ↵
5. Type `source ?/.cshrc` at a UNIX prompt to source the **.cshrc** file and validate the settings in steps 1 through 4.
6. Add the following lines to your **MGC_location_map** file:

   `$MAX2_MENTOR` ↵

   `/usr/maxplus2/mentor/max2` ↵

   `$MGC_GENLIB` ↵

   /<*user-specified Mentor Graphics GEN_LIB directory*> ↵

   `$MGC_LSLIB` ↵

   /<*user-specified Mentor Graphics LS_LIB directory*> ↵

   `$MAX2_EXAMPLES` ↵

   /<*user-specified example directory*> ↵

   `$MAX2_LMCLIB` ↵

   /<*user-specified Logic Modeling directory*> ↵

   `$MAX2_GENLIB` ↵

   `/usr/maxplus2/simlib/mentor/alt_max2` ↵

   `$MAX2_QSIM` ↵

   `/usr/maxplus2/simlib/mentor/max2sim` ↵

   `$MAX2_FONT` ↵

   `/usr/maxplus2/mentor/max2/fonts` ↵

   `$MGC_SYS1076_STD` ↵

   /<*user-specified MGC_HOME directory*>`/pkgs/sys_1076_std/ std` ↵

```
$MGC_SYS1076_ARITHMETIC ↵
/<user-specified MGC_HOME directory>/pkgs/sys_1076_std/arithmetic ↵
$MGC_SYS1076_PORTABLE ↵
/<user-specified MGC_HOME directory>/pkgs/sys_1076_std/mgc_portable ↵
$MGC_SYS1076_IEEE ↵
/<user-specified MGC_HOME directory>/pkgs/sys_1076_std/ieee ↵
$MGC_SYS1076_SRC ↵
/<user-specified MGC_HOME directory>/pkgs/sys_1076_std/ src ↵
$MAX2_MFLIB ↵
/usr/maxplus2/simlib/mentor/alt_mf ↵
```

☞ Installing the Alteraprovided MAX+PLUS II/Mentor Graphics/Exemplar Logic interface on your computer automatically installs a template for these environment variables in the **/usr/maxplus2/mentor/max2/location_map/location_map** file.

7. If you want to use QuickHDL software to simulate VHDL or Verilog HDL designs, add the following line in the `[library]` section of your **quickhdl.ini** file: `altera = $MAX2_MFLIB`.
8. If you plan to use QuickHDL software to simulate VITAL-compliant VHDL files, add the following lines to your **MGC_location_map** file:

```
$MAX2_VTLLIB ↵
/usr/maxplus2/simlib/mentor/alt_vtl ↵
```

9. Copy the **/usr/maxplus2/maxplus2.ini** file to your **$HOME** directory:

```
cp /usr/maxplus2/maxplus2.ini $HOME ↵
chmod u+w $HOME/maxplus2.ini ↵
```

The **maxplus2.ini** file contains both Altera- and user-specified initialization parameters that control the MAX+PLUS II software, such as Alteraprovided logic and symbol library paths and the current project name. The MAX+PLUS II installation procedure creates and copies the **maxplus2.ini** file to the **/usr/maxplus2** directory.

☞ Normally, you do not have to edit your local copy of **maxplus2.ini**, because the MAX+PLUS II software updates the file automatically whenever you change any parameters or settings. However, if you move the **max2lib** and **max2inc** library subdirectories, you must update the file. Go to "Creating & Using a Local Copy of the **maxplus2.ini** File" in MAX+PLUS II Help for more information.

Related Topics:

- Go to the following topics, which are available on the web, for additional information:
  - ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
  - This manual is also available in 4 parts:
    - Preface & Section 1: MAX+PLUS II Installation
    - Section 2: MAX+PLUS II - A Perspective
    - Section 3: MAX+PLUS II Tutorial
    - Appendices, Glossary & Index

MAX+PLUS II/Mentor Graphics Software Requirements

The following products are used to generate, process, synthesize, and verify a project with the MAX+PLUS ® II software and Mentor Graphics software:

Mentor Graphics		Exemplar	Altera
version C.1:			
System_1076 Compiler	QuickHDL	Galileo Extreme	

QuickSim II Design Architect ENRead ENWrite GEN_LIB library	QuickHDL Pro QuickPath LS_LIB library (optional) DVE	version 4.1.1 Leonardo version 4.1.3	MAX+PLUS II version 9.4

The MAX+PLUS II **read.me** file provides up-to-date information on which versions of Mentor Graphics applications are supported by the current version of MAX+PLUS II. It also provides information on installation and operating requirements. You should read the **read.me** file on the CD-ROM before installing the MAX+PLUS II software. After installation, you can open the **read.me** file from the MAX+PLUS II Help menu.

Altera-Provided Logic & Symbol Libraries

The MAX+PLUS ® II/Mentor Graphics environment provides libraries for compiling, synthesizing, and simulating designs.

You can create your own libraries of custom functions for use in Design Architect schematics and VHDL and Verilog HDL design files. You can use custom functions to incorporate an EDIF Input File (**.edf**), Text Design File (**.tdf**), or any other MAX+PLUS II-supported design file into a project. The MAX+PLUS II software uses the Altera® provided **mnt8_bas.lmf** and **exemplar.lmf** Library Mapping Files to map standard Design Architect symbols and VHDL and Verilog HDL functions to equivalent MAX+PLUS II logic functions. To use custom functions, you can create a custom LMF that maps your custom functions to the equivalent EDIF input file, TDF, or other design file. Go to "Library Mapping File" in MAX+PLUS II Help for more information.

Design Architect Libraries

You can enter a Design Architect schematic with logic functions from these Altera-provided symbol libraries: **ALTERA LPMLIB**, **ALTERA GENLIB**, **LSTTL BY TYPE**, and **LSTTL ALL PARTS**. You can access these libraries by choosing **Altera Libraries** (Libraries menu) in the Design Architect software. For information on using library of parameterized modules (LPM) functions, see **ALTERA LPMLIB** Library below.

ALTERA GENLIB Library (Design Architect) & Altera (VHDL) Libraries

The **ALTERA GENLIB** symbol library (called the **Altera** library for VHDL) includes several MAX+PLUS II primitives for controlling design synthesis and fitting. It also includes four macrofunctions (`8count`, `8mcomp`, `8fadd`, and `81mux`) that are optimized for different Altera device families, and the `clklock` phase-locked loop megafunction, which is supported for some FLEX ® 10K devices.

The following table shows the MAX+PLUS II-specific logic functions.

Table 1. MAX+PLUS II-Specific Logic Functions

Macrofunctions *Note (1)*				**Primitives**	
Name	**Description**	**Name**	**Description**	**Name**	**Description**
`8fadd`	8-bit full adder	`LCELL`	Logic cell buffer	`EXP`	MAX ® 5000, MAX 7000 , and MAX 9000 Expander buffer
`8mcomp`	8-bit magnitude comparator	`GLOBAL`	Global input buffer	`SOFT`	Soft buffer
`8count`	8-bit up/down counter	`CASCADE`	FLEX 6000, FLEX 8000, and FLEX 10K cascade buffer	`OPNDRN`	Open-drain buffer
`81mux`	8-to-1			`DFFE`	

	multiplexer	CARRY	FLEX 6000, FLEX 8000, and FLEX 10K carry buffer	DFFE6K	D-type flipflop with Clock Enable *Note (2)*
clklock	Phase-locked loop				

Notes:

1. Logic function names that begin with a number must be preceded by "a_" in VHDL designs. For example, 8fadd must be specified as a_8fadd instead.
2. If you want to use QuickHDL software, make sure you have updated your **quickhdl.ini** file, as described in step 7 of Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment.
3. For designs that are targeted for FLEX 6000 devices, you should use the DFFE primitive only if the design contains either a Clear or Preset signal, but not both. If your design contains both a Clear and a Preset signal, you must use the DFFE6K primitive.

☞ Choose **Old-Style Macrofunctions**, **Primitives**, or **Megafunctions/LPM** from the MAX+PLUS II Help menu for detailed information on these functions.

ALTERA LPMLIB Library

The Alteraprovided **ALTERA LPMLIB** library, which is available for Design Architect schematics and VHDL designs, includes standard functions from the library of parameterized modules (LPM) 2.1.0, except the truth table, finite state machine, and pad functions. The LPM standard defines a set of parameterized modules (i.e., parameterized functions) and their corresponding representations in an EDIF netlist file. These logic functions allow you to create and functionally simulate an LPM-based design without targeting a specific device family. After the design is completed, you can target the design to any device family. The parameters you specify for each LPM function determine which simulation models are generated.

☞ Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu for more information about LPM functions.

Related Topics:

- FLEX Devices
- MAX Devices
- Classic Device Family

Local Work Area Directory Structure

Design Architect software automatically creates and maintains the project directory structure required for all stages of design entry. Galileo Extreme, Leonardo, and ENWrite software create a **max2** subdirectory, if it does not already exist, under the project directory. These software applications also generate EDIF netlist files, and copy them from the *<project name>* directory to this **max2** subdirectory. All MAX+PLUS ® II Compiler output files are created in the **max2** subdirectory.

Simulation files created with Mentor Graphics applications and Logic Modeling files are located in the board-level simulation subdirectory of the project directory. You can use these files during simulation with QuickSim II software.

The only directory you need to create is the local work directory, which should contain all project directories. Figure 1 shows the recommended file structure.

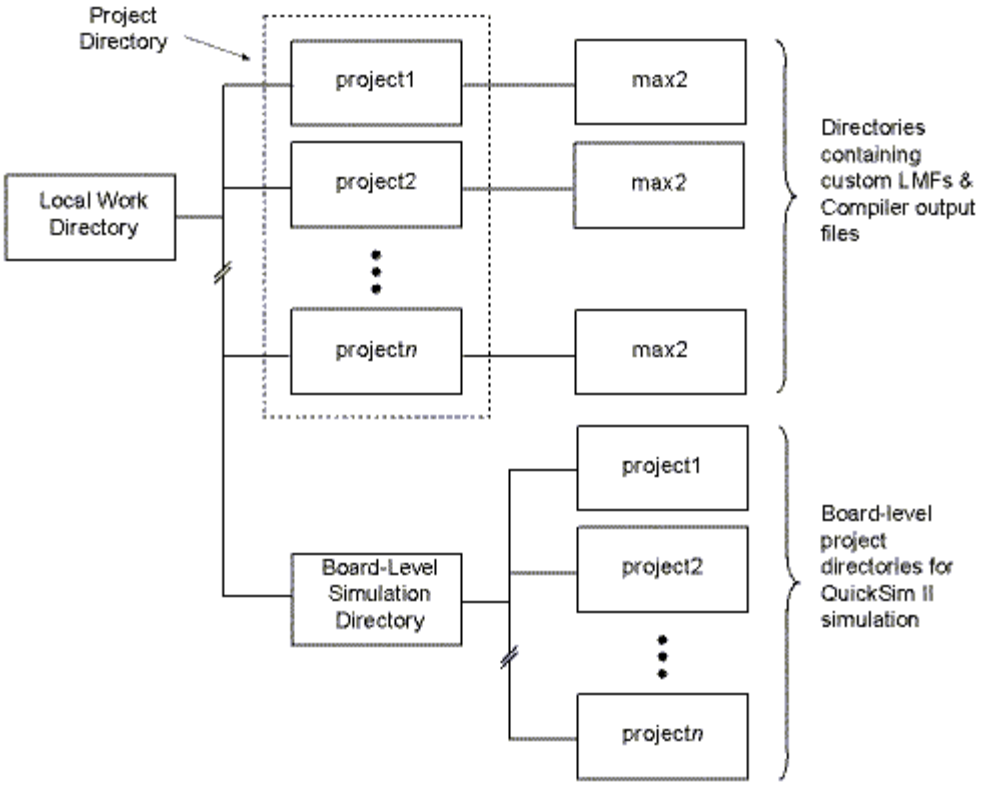


Figure 1. Recommended File Structure

Related Topics:

- Go to the following MAX+PLUS II ACCESS[SM] Key topics for related information:
  - MAX+PLUS II Project Directory Structure
  - Mentor Graphics Project Directory Structure

Mentor Graphics Project Directory Structure

Design Architect software generates the following files for each schematic:

- <*drawing name*>**/mgc_component.attr**
- <*drawing name*>**/part.Eddm_part.attr**
- <*drawing name*>**/part.part_1**
- <*drawing name*>**/schematic.mgc_schematic.attr**
- <*drawing name*>**/schematic/schem_id**
- <*drawing name*>**/schematic/sheet1.mgc_sheet.attr**
- <*drawing name*>**/schematic/sheet1.sgfx_1**
- <*drawing name*>**/schematic/sheet1.ssht_1**

The files generated for each schematic are stored in the schematic's <*drawing name*> directory and should not be edited. Mentor Graphics software automatically manages file storage and retrieval operations through this <*drawing name*> directory structure, which does not reflect hierarchical design relationships. Figure 1 shows a sample file structure with **project1** as the UNIX project directory, and **design1**, **subdesign1**, and **subdesign2** as the directories for the top-level design and subdesigns of the project.

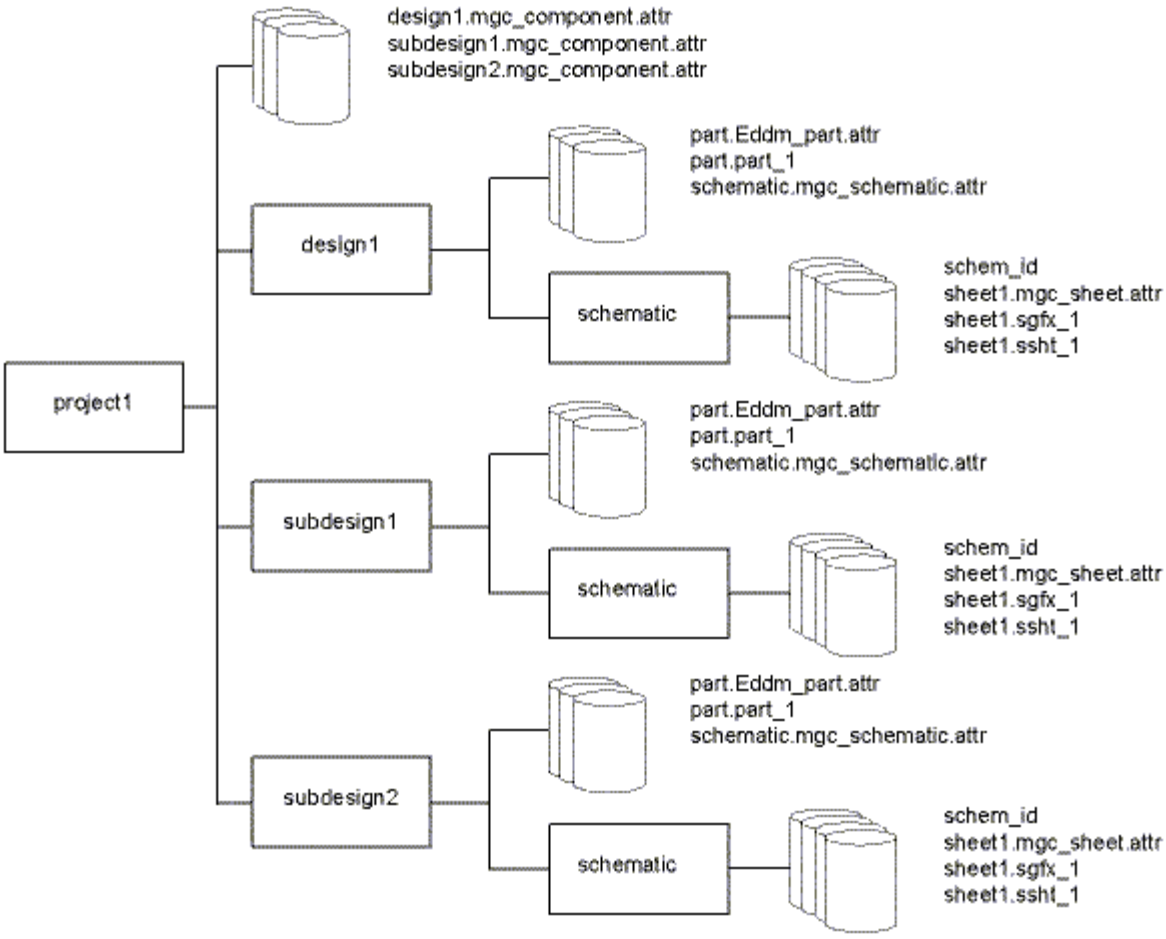


Figure 1. Design Architect Project File Structure

When the ENWrite utility converts the schematic into an EDIF netlist file, it processes the design information and all related file subdirectories, then creates the EDIF netlist file in the directory defined by the user. The EDIF netlist file is named <*project name*>**.edf**, where <*project name*> is the name of the top-level design file. The <*project name*>**.edf** file is automatically moved to the **max2** directory under the project directory.

Related Topics:

- Go to the following MAX+PLUS II ACCESS^{SM} Key topics for related information:
  - Local Work Area Directory Structure
  - MAX+PLUS II Project Directory Structure

MAX+PLUS II Project Directory Structure

In the MAX+PLUS $^{®}$ II software, a project name is the name of a top-level design file, without the filename extension. This design file can be an EDIF, VHDL, or Verilog HDL netlist file; an Altera Hardware Description Language (AHDL) Text Design File (TDF); or any other MAX+PLUS II-supported design file. The EDIF netlist file must be created by ENWrite, Galileo Extreme, or Leonardo software and imported into MAX+PLUS II as an EDIF Input File (**.edf**). Figure 1 shows an example of a MAX+PLUS II project directory.

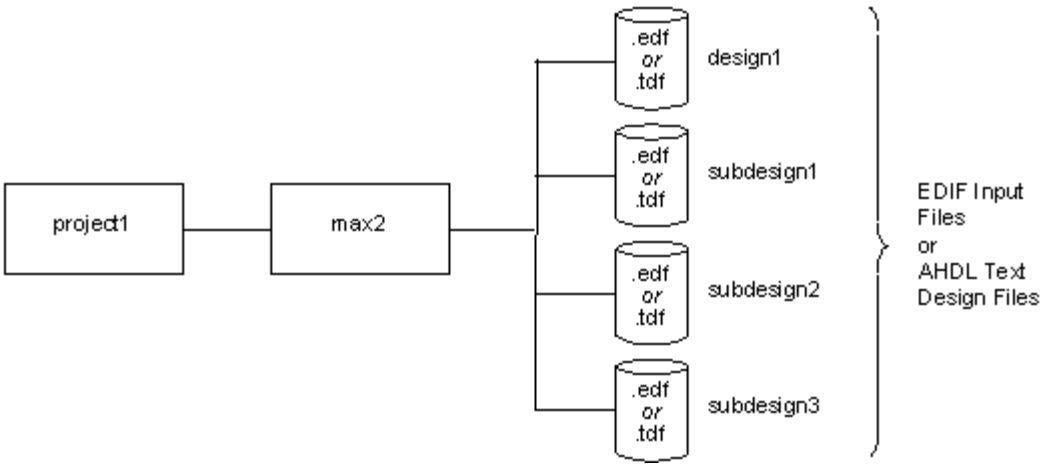


Figure 1. Sample MAX+PLUS II Project Directory

The MAX+PLUS II software stores the connectivity data on the links between design files in a hierarchical project in a Hierarchy Interconnect File (**.hif**), but refers to the entire project only by its project name. The MAX+PLUS II Compiler uses the HIF to build a single, fully flattened project database that integrates all the design files in a project hierarchy.

Related Topics:

- Go to the following MAX+PLUS II ACCESS^{SM} Key topics for related information:
  - Local Work Area Directory Structure
  - Mentor Graphics Project Directory Structure

MAX+PLUS II/Mentor Graphics/Exemplar Logic Interface File Organization

The following table shows the MAX+PLUS $^{®}$ II/Mentor Graphics interface subdirectories that are created in the MAX+PLUS II system directory (by default, the **/usr/maxplus2** directory) during MAX+PLUS II installation.

☞ For information on the other directories that are created during MAX+PLUS II installation, see "MAX+PLUS II File Organization" in *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual.

Table 1. MAX+PLUS II Directory Organization

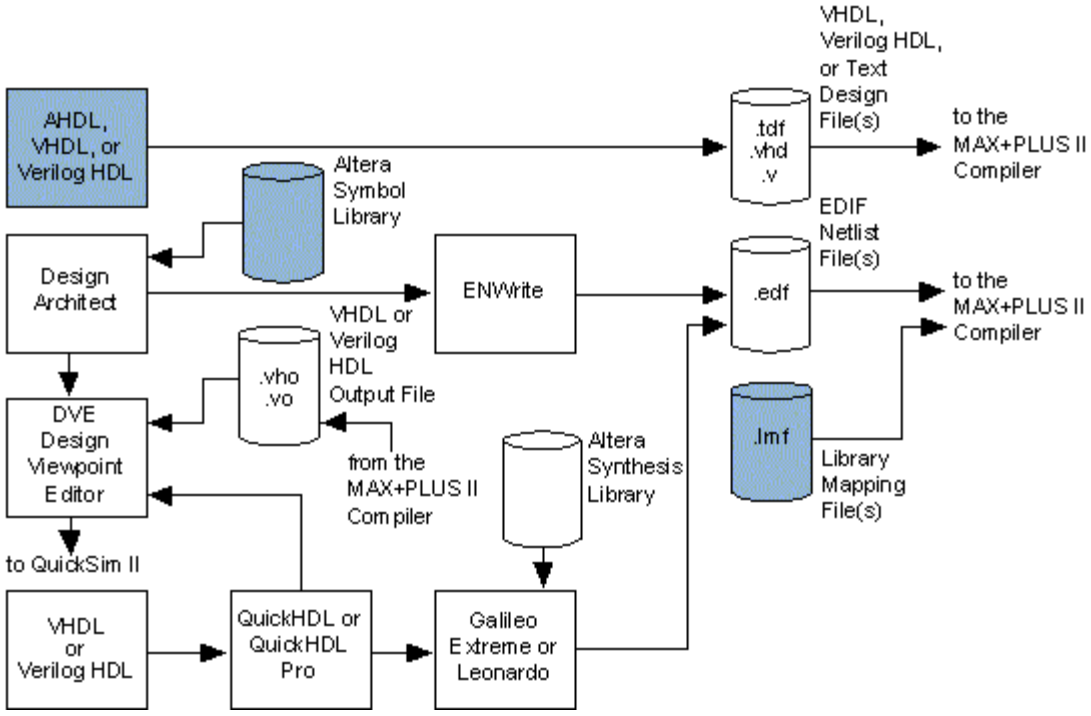
Directory	Description
.lmf	Contains the Altera-provided Library Mapping Files, **mnt8_bas.lmf** and **exemplar.lmf**, that map Mentor Graphics and Exemplar Logic logic functions to equivalent MAX+PLUS II logic functions.
./mentor	Contains the AMPLE userware for the MAX+PLUS II/Mentor Graphics interface.
./simlib/mentor/alt_max2	Contains MAX+PLUS II primitives such as `CARRY`, `CASCADE`, `EXP`, `GLOBAL`, `LCELL`, `SOFT`, `OPNDRN`, `DFFE`, and `DFFE6K` (D flipflop with Clock Enable) for use in Design Architect schematics.
./simlib/mentor/max2sim	Contains the MAX+PLUS II/Mentor Graphics simulation model library, **max2sim**, for use with QuickSim II and QuickPath software.
./simlib/mentor/synlib	Contains the MAX+PLUS II synthesis library for use with AutoLogic II software, which supports synthesis for users running Mentor Graphics version B1.
./simlib/mentor/alt_mf	Contains the MAX+PLUS II macrofunction and megafunction libraries.
./simlib/mentor/alt_vtl	Contains the MAX+PLUS II VITAL library.

Mentor Graphics/Exemplar Logic Design Entry Flow

The following figure shows the design entry flow for the MAX+PLUS® II/Mentor Graphics/Exemplar Logic interface.

Figure 1. MAX+PLUS II/Mentor Graphics/Exemplar Logic Design Entry Flow

Alteraprovided items are shown in blue.



Creating VHDL & Verilog HDL Designs for Use with MAX+PLUS II Software

You can create VHDL and Verilog HDL design files with the MAX+PLUS ® II Text Editor or another standard text editor and save them in the appropriate directory for your project.

The MAX+PLUS II Text Editor offers the following advantages:

- Templates are available with the **VHDL Templates** and **Verilog Templates** commands (Template menu). These templates are also available in the ASCII **vhdl.tmp** and **verilog.tmp** files, respectively, which are located in the **/usr/maxplus2** directory.
- If you use the MAX+PLUS II Text Editor to create your VHDL design, you can turn on the **Syntax Coloring** command (Options menu). The Syntax Coloring feature displays keywords and other elements of text in text files in different colors to distinguish them from other forms of syntax.

To create a VHDL or Verilog HDL design file for use with the MAX+PLUS II software, go through the following steps:

1. Enter a VHDL or Verilog HDL design in the MAX+PLUS II Text Editor or another standard text editor and

save it in your working directory.

2. Enter primitives, macrofunctions, and megafunctions in your VHDL or Verilog HDL design from the **Altera** library.

   The following topics describe special steps needed to instantiate LPM and `clklock` functions:

   - Instantiating LPM Functions in VHDL
   - Instantiating the clklock Megafunction in VHDL or Verilog HDL

   You can instantiate MegaCore™ functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP™). The OpenCore™ feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.

3. (Optional) Use the QuickHDL software to functionally simulate the design file, as described in Performing a Functional Simulation with QuickHDL Software and Performing a Functional Simulation with QuickHDL Pro Software.

4. Once you have created a VHDL or Verilog HDL design, you can generate an EDIF netlist file that can be imported into the MAX+PLUS II software with either of the following methods:

   - You can synthesize and optimize your design and create an EDIF netlist file, as described in Synthesizing & Optimizing VHDL & Verilog HDL Projects with Galileo Extreme Software or Synthesizing & Optimizing VHDL & Verilog HDL Projects with Leonardo Software.

   - You can use the Altera VHDL Express utility, **vhd_exprss**, to automatically create an EDIF netlist file, compile it with the MAX+PLUS II Compiler, generate an EDIF Output File (**.edo**), and prepare the EDIF Output File for simulation with QuickHDL software, as described in Using the Altera Schematic Express (**vhd_exprss**) Utility.

Installing the Alteraprovided MAX+PLUS II/Mentor Graphics/Exemplar Logic interface on your computer automatically creates the following sample VHDL design files:

- **/usr/maxplus2/examples/mentor/example5/count4.vhd**
- **/usr/maxplus2/examples/mentor/example6/count8.vhd**
- **/usr/maxplus2/examples/mentor/example8/adder16.vhd**

Related Topics:

- Go to Compiling Projects with MAX+PLUS II Software in these MAX+PLUS II ACCESS℠ Key topics for related information.

Synthesizing & Optimizing VHDL & Verilog HDL Projects with Galileo Extreme Software

After you have created a VHDL or Verilog HDL design, you can use Exemplar Logic's Galileo Extreme software to synthesize and optimize your VHDL Design File (**.vhd**) or Verilog Design File (**.v**) and prepare it for compilation with the MAX+PLUS ® II Compiler.

To synthesize and optimize your project and generate an EDIF netlist file with Galileo Extreme software, go through the following steps:

1. Be sure to set up the working environment correctly, as described in Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment.
2. Create a VHDL or Verilog HDL design that follows the guidelines described in Creating VHDL & Verilog

HDL Designs for Use with MAX+PLUS II Software.

3. (Optional) Use the QuickHDL software to functionally simulate the design file, as described in Performing a Functional Simulation with QuickHDL Software.
4. Select the icon for your design file in the appropriate directory, press Button 3, and choose **max2_galileo** in the Navigator window to start the Galileo Extreme software. You can also start Galileo Extreme software by typing `max2_galileo` ↵ at the UNIX prompt.
5. Specify settings for the *Filename* and *Format* options under *INPUT DESIGN*.
6. Specify settings for the *Filename*, *Format*, and *Technology* options under *OUTPUT DESIGN*. Verify that EDIF is specified in the *Format* box.
7. Choose the **Altera Output Options** button if you want to specify settings for various parameters, including *Maximum Fanin* for MAX devices and *Part Number* for FLEX devices. You can also turn on the *Run MAX+PLUS II* option for design compilation, which specifies that the MAX+PLUS II Compiler should start processing your design immediately after you run Galileo Extreme. Choose **OK** to save any setting changes.
8. Choose **Start Run**. The Galileo Extreme software generates <*design name*>**.edf** in the <*project directory*>/**max2** subdirectory and then closes, returning you to the Navigator window.
9. Process your design with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software. If you turned on the *Run MAX+PLUS II* option in step 7, the MAX+PLUS II Compiler automatically starts processing your design after you run Galileo Extreme.

Installing the Altera® provided MAX+PLUS II/Mentor Graphics/Exemplar Logic interface on your computer automatically creates the following sample VHDL Design Files:

- **/usr/maxplus2/examples/mentor/example5/count4.vhd**
- **/usr/maxplus2/examples/mentor/example6/count8.vhd**
- **/usr/maxplus2/examples/mentor/example8/adder16.vhd**

Related Topics:

- Go to the following MAX+PLUS II ACCESS℠ Key topics for related information:
  - Synthesizing & Optimizing VHDL & Verilog HDL Projects with Leonardo Software
  - Performing a Timing Simulation with QuickHDL Software
  - Performing a Timing Analysis with QuickPath Software

Compiling Projects with MAX+PLUS II Software

The MAX+PLUS® II Compiler can process design files in a variety of formats. This topic describes how to use MAX+PLUS II software to compile projects in which the top-level design file is an EDIF Input File (with the extension **.edf**).

☞ Refer to the following sources for additional information:

- Go to MAX+PLUS II Help for information on compiling VHDL and Verilog HDL, design files directly with the MAX+PLUS II Compiler.
- Go to Running Synopsys Compilers from MAX+PLUS II Software for information on running the Synopsys Design Compiler or FPGA Compiler software on a VHDL or Verilog HDL design from within the MAX+PLUS II Compiler window.

To compile a design (also called a "project") with MAX+PLUS II software, go through the following steps:

1. Create design files that are compatible with the MAX+PLUS II software and convert them into EDIF Input Files with the extension **.edf**. Specific instructions for some tools are described in these MAX+PLUS II ACCESS℠ Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your design entry or synthesis and optimization tool.

2. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, create a mapping for each function in a Library Mapping File (**.lmf**) to map the custom symbol to the corresponding EDIF Input File, AHDL Text Design File (**.tdf**), or other MAX+PLUS II-supported design file. These custom functions are represented in design files as hollow-body symbols or "black box" HDL descriptions.

   ☞ Go to "Library Mapping Files (**.lmf**)" in MAX+PLUS II Help for more information.

3. Open MAX+PLUS II and specify the name of your top-level design file as the project name with the **Project Name** command (File menu). If you open an HDL file in the MAX+PLUS II Text Editor, you can choose the **Project Set Project to Current File** command (File menu) instead.

   ☞ You can also compile a project from a command line. However, the first time you compile a project, the settings you need to specify are easier to specify from within the MAX+PLUS II software. After you have run the graphical user interface for the MAX+PLUS II software at least once, you can more easily use the command-line **setacf** utility to modify options in the Assignment & Configuration File (**.acf**) for the project. Type `setacf -h` ↵ and `maxplus2 -h` ↵ for descriptions of **setacf** and MAX+PLUS II command-line syntax.

4. Choose **Device** (Assign menu) and select the target Altera device family in the *Device Family* drop-down list box. If you wish to implement the design logic in a specific device, select it in the *Devices* box. Otherwise, select *AUTO* to allow the MAX+PLUS II Compiler to choose the best device(s) in the current device family. If your design entry or synthesis and optimization tool required you to specify a target family and/or device, specify the same information in this dialog box. For information on partitioning logic among multiple devices, go to MAX+PLUS II Help. Choose **OK**.
5. Open the Compiler window by choosing the **Compiler** command (MAX+PLUS II menu). Go through the following steps to specify the options necessary to compile the design file(s) in your project:
   1. Ensure that all EDIF netlist files have the extension **.edf** and choose **EDIF Netlist Reader Settings** (Interfaces menu).
   2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor. This name should be the name of the vendor whose tool(s) you used to create the EDIF netlist files. If your vendor name does not appear, select *Custom* instead.

      ☞ If you are compiling a design created with Synopsys FPGA Express software, select *Synopsys*, choose the **Customize** button, enter <*project name*>**.lmf** in the *LMF #1* box, choose **OK**, and skip to step 6.

   3. If you selected an existing vendor name in the *Vendor* box and your project contains design files that require custom LMF mappings, choose the **Customize** button to expand the dialog box to show all settings. Turn on the *LMF #2* checkbox and type your custom LMF's filename in the corresponding text box, or select a name from the *Files* box. The selection in the *Vendor* box will change to *Custom* and all settings will be retained until you change them again.
   4. If you selected *Custom* in the *Vendor* box, choose the **Customize** button to expand the dialog box to show all settings. Any previously defined custom settings will be displayed. Under *Signal Names*, type one or more names with up to 20 total name characters in the *VCC* or *GND* box if your EDIF Input File(s) use one or more names other than `VCC` or `GND` for the global high or low signals. Multiple signal names must be separated by either a comma (,) or a space. Under *Library Mapping Files*, turn on the *LMF #1* checkbox and type a filename in the text box following it, or select a name from the *Files* box. If necessary, specify another LMF name in the *LMF #2* box. Go to MAX+PLUS II Help for detailed information on the settings available in the **EDIF Netlist Reader Settings** dialog box.
   5. Choose **OK**.
6. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, you may need to ensure that all files are present in your project directory, i.e., the same directory as the top-level design file. Otherwise, you must specify the directories containing these files as user libraries with the **User Libraries** command (Options menu).
7. Follow all guidelines that apply to your design entry or synthesis and optimization tool:
   - Exemplar Logic Galileo Extreme-Specific Compiler Settings
   - Synopsys DesignWare-Specific Compiler Settings
   - Converting Synopsys FPGA Compiler & Design Compiler Timing Constraints into MAX+PLUS II-Compatible Format with the **syn2acf** Utility
   - Synplicity Synplify-Specific Compiler Settings
8. If you wish to generate EDIF, VHDL, or Verilog HDL output files for post-compilation simulation or timing

analysis with another EDA tool, go through the following steps:

1. (Optional) Turn on the **Optimize Timing SNF** command (Processing menu) to reduce the size of the output file(s). Turning on this command can reduce the size of output netlists by up to 30%.

   This command does not create optimized timing SNFs on UNIX workstations. However, a non-optimized timing SNF provides the same functional and timing information as an optimized timing SNF.

2. If you wish to generate EDIF Output Files (**.edo**), go through these steps:
   1. Turn on the **EDIF Netlist Writer** command (Interfaces menu). Then choose the **EDIF Netlist Writer Settings** command (Interfaces menu).
   2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor and choose **OK**. If your vendor name does not appear, select *Custom* instead and specify the settings that are appropriate for your simulation or timing analysis tool. Go to MAX+PLUS II Help for detailed information on the options available in the **EDIF Netlist Writer Settings** dialog box.
   3. To generate an optional Standard Delay Format (SDF) Output File (**.sdo**), choose the **Customize** button to expand the dialog box to show all settings. Select one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**.

   The filenames of the EDIF Output File(s) and optional SDF Output File(s) are the same as the user-defined chip name(s) for the project; if no chip names exist, the Compiler assigns filenames that are based on the project name. For a multi-device project, the Compiler also generates a top-level EDIF Output File that is uniquely identified by "**_t**" appended to the project name. In addition, the Compiler automatically generates a VHDL Memory Model Output File, <*project name*>**.vmo**, when it generates an EDIF Output File that contains memory (RAM or ROM).

3. If you wish to generate VHDL Output Files (**.vho**), turn on the **VHDL Netlist Writer** command (Interfaces menu). Then choose **VHDL Netlist Writer Settings** command (Interfaces menu). Select *VHDL Output File (.vho)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF ver. 2.1 files contain timing delay information that allows you to perform back-annotation simulation in VHDL with VITAL-compliant simulation libraries. The VHDL Output Files generated by the Compiler have the extension **.vho**, but are otherwise named in the same way as the EDIF Output Files described above.
4. If you wish to generate Verilog HDL Output Files (**.vo**), turn on the **Verilog Netlist Writer** command (Interfaces menu). Then choose **Verilog Netlist Writer Settings** command (Interfaces menu). Select *Verilog Output File (.vo)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF Output Files contain timing delay information that allows you to perform back-annotation simulation in Verilog HDL. The Verilog Output Files generated by the Compiler have the extension **.vo**, but are otherwise named in the same way as the EDIF Output Files described above.

9. To run the MAX+PLUS II Compiler, choose the **Project Save & Compile** command (File menu) or choose the **Start** button in the Compiler window.

   See step 3 for information on running MAX+PLUS II software from the command line.

10. Once you have compiled the project with the MAX+PLUS II Compiler, you can use the VHDL, Verilog HDL, or EDIF output file(s), and the optional SDF Output File(s) (**.sdo**) to perform timing analysis or timing simulation with another EDA tool. Specific instructions for some tools are described in these MAX+PLUS II ACCESS Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your EDA tool.

The MAX+PLUS II Compiler also generates a Report File (**.rpt**), a Pin-Out File (**.pin**), and one or more of the following files for device programming or configuration:

- JEDEC Files (**.jed**)
- Programmer Object Files (**.pof**)
- SRAM Object Files (**.sof**)
- Hexadecimal (Intel-format) Files (**.hex**)
- Tabular Text Files (**.ttf**)

Related Topics:

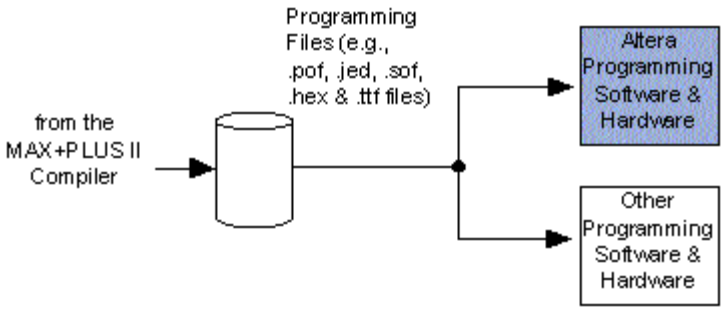
- Refer to the following sources for additional information:
  - Go to *Compiler Procedures* in MAX+PLUS II Help for information on other available Compiler settings.
  - Go to *Programmer Procedures* in MAX+PLUS II Help for instructions on creating other types of programming files and on programming or configuring Altera devices.
  - Go to Back-Annotating MAX+PLUS II Pin Assignments to Design Architect Symbols for information on back-annotating pin assignments in Mentor Graphics Design Architect schematics.
  - Go to Programming Altera Devices for information on the different programming hardware options for Altera device families.
- Go to the following topics, which are available on the web, for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware

Programming Altera Devices

Once you have successfully compiled and simulated a project with the MAX+PLUS® II software, you can program an Altera® device and test it in the target circuit. Figure 1 shows the device programming flow for MAX+PLUS II software.

Figure 1. MAX+PLUS II Device Programming Flow

Altera-provided items are shown in blue.



You can program devices with Altera programming hardware and MAX+PLUS II Programmer software installed on a 486- or Pentium-based PC or a UNIX workstation, or with programming hardware and software available from other manufacturers. Table 1 shows the available Altera programming hardware options on PCs and UNIX workstations.

Table 1. Altera Programming Hardware

Programming Hardware	UNIX PCs	Work-	MAX® 3000A	Classic® & MAX	MAX 7000 & MAX	MAX 7000A, MAX 7000AE, MAX 7000B, MAX	FLEX® 6000, FLEX 6000A, FLEX 8000, FLEX 10K, FLEX 10KA,	In-System Programming/

Option	stations	Devices	5000 Devices	7000E Devices	7000S MAX 9000 & MAX 9000A Devices	FLEX 10KB, & FLEX 10KE Devices	Configuration
Logic Programmer card, PL-MPU Master Programming Unit, and device-specific adapters	✓	✓	✓	✓	✓		
BitBlaster™ Download Cable	✓	✓	✓		✓	✓	✓
ByteBlasterMV™ Download Cable	✓		✓		✓	✓	✓
MasterBlaster™ Download Cable	✓	✓	✓		✓	✓	✓

If you wish to transfer programming files from a UNIX workstation to a PC over a network with File Transfer Protocol (FTP) or other similar transfer programs, be sure to select binary transfer mode.

Programming hardware from other manufacturers varies, but typically consists of a device connected to one of the serial ports on the workstation. Various vendors, such as Data I/O and BP Microsystems, supply hardware and software for programming Altera devices.

Related Topics:

- Go to Compiling Projects with MAX+PLUS II Software for information on creating programming files.
- Go to the following topics, which are available on the web, for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware
  - FLEX Devices
  - MAX Devices
  - Classic Device Family

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Synthesizing & Optimizing VHDL & Verilog HDL Projects with Galileo Extreme Software

After you have created a VHDL or Verilog HDL design, you can use Exemplar Logic's Galileo Extreme software to synthesize and optimize your VHDL Design File (**.vhd**) or Verilog Design File (**.v**) and prepare it for compilation with the MAX+PLUS ® II Compiler.

To synthesize and optimize your project and generate an EDIF netlist file with Galileo Extreme software, go through the following steps:

1. Be sure to set up the working environment correctly, as described in Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment.
2. Create a VHDL or Verilog HDL design that follows the guidelines described in Creating VHDL & Verilog HDL Designs for Use with MAX+PLUS II Software.
3. (Optional) Use the QuickHDL software to functionally simulate the design file, as described in Performing a Functional Simulation with QuickHDL Software.
4. Select the icon for your design file in the appropriate directory, press Button 3, and choose **max2_galileo** in the Navigator window to start the Galileo Extreme software. You can also start Galileo Extreme software by typing `max2_galileo` ↵ at the UNIX prompt.
5. Specify settings for the *Filename* and *Format* options under *INPUT DESIGN*.
6. Specify settings for the *Filename*, *Format*, and *Technology* options under *OUTPUT DESIGN*. Verify that EDIF is specified in the *Format* box.
7. Choose the **Altera Output Options** button if you want to specify settings for various parameters, including *Maximum Fanin* for MAX devices and *Part Number* for FLEX devices. You can also turn on the *Run MAX+PLUS II* option for design compilation, which specifies that the MAX+PLUS II Compiler should start processing your design immediately after you run Galileo Extreme. Choose **OK** to save any setting changes.
8. Choose **Start Run**. The Galileo Extreme software generates <*design name*>**.edf** in the <*project directory*>/**max2** subdirectory and then closes, returning you to the Navigator window.
9. Process your design with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software. If you turned on the *Run MAX+PLUS II* option in step 7, the MAX+PLUS II Compiler automatically starts processing your design after you run Galileo Extreme.

Installing the Altera® provided MAX+PLUS II/Mentor Graphics/Exemplar Logic interface on your computer automatically creates the following sample VHDL Design Files:

- **/usr/maxplus2/examples/mentor/example5/count4.vhd**
- **/usr/maxplus2/examples/mentor/example6/count8.vhd**
- **/usr/maxplus2/examples/mentor/example8/adder16.vhd**

Related Links:

- Go to the following MAX+PLUS II ACCESS℠ Key topics for related information:

- Synthesizing & Optimizing VHDL & Verilog HDL Projects with Leonardo Software
- Performing a Timing Simulation with QuickHDL Software
- Performing a Timing Analysis with QuickPath Software

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Converting Synopsys Hierarchical Timing Constraints into MAX+PLUS II-Compatible Format with the gen_iacf and gen_hacf Utilities

Altera provides the **gen_hacf** and **gen_iacf** utilities, which convert Synopsys timing constraints into the MAX+PLUS® II Assignment & Configuration File (**.acf**) format. For information on converting timing constraints from non-hierarchical designs, refer to Converting Synopsys Timing Constraints into MAX+PLUS II-Compatible Format with the **syn2acf** Utility. The **gen_iacf** utility generates intermediate, individual ACFs with the extension **.iacf** for each subdesign. The **gen_hacf** utility merges the individual **.iacf** files into a single ACF for the whole design.

To use the **gen_iacf** and **gen_hacf** utilities, follow these steps:

☞ You can create a **dc_shell** script that performs most of these steps. Refer to Figure 2 for a sample **dc_shell** script.

1. Make sure that you have specified the correct path of your local Perl executable, as specified in step 5 of Setting Up the MAX+PLUS II/Synopsys Working Environment.

   ☞ The **gen_iacf** and **gen_hacf** utilities use the `ALT_HOME` environment variable, if it has been specified, to determine the MAX+PLUS II system directory; otherwise, it uses the **/usr/maxplus2** directory. To specify a different MAX+PLUS II system directory with the `ALT_HOME` environment variable, you can either edit the **.cshrc** file to specify the correct directory or type the following command at the UNIX prompt:

   `setenv ALT_HOME` <*MAX+PLUS II system directory*> ↵

2. Once you have synthesized your design with Design Compiler or FPGA Compiler, generate an hierarchical EDIF netlist file for the top-level design by typing the following command at the **dc_shell** prompt:

   `write -f edif -hierarchy` <*top-level design name*> `-o` <*top-level design name*>`.hier.edf` ↵

3. Generate intermediate ACF files (**.iacf**) for all subdesigns that have constraints, including the top-level design.

   1. Generate the following input files for the **gen_iacf** utility by using a **gen_iacf.cmd** file. Figure 1 shows a sample **gen_iacf.cmd** file.

      - Flattened EDIF netlist file
      - **dc_shell** script file
      - Standard Delay Format (SDF) constraints construct
      - SDF timing delay construct

      ☞ The **gen_iacf** and **gen_hacf** utilities do not support `set_arrival` timing constraints for internal nodes.

      Figure 1. Sample gen_iacf.cmd File

```
ungroup -flatten -all
write -f edif
write_script > <design_name> + "_setup.dc"
write_constraints -format sdf -cover_design
write_timing -format sdf
```

☞ This sample command file assumes that the `design_name` variable has been set before the command file is included.

2. Run the **gen_iacf** utility for each design that has timing constraints (including the top-level design) by typing the following command at the UNIX prompt:

   `gen_iacf` <*design name*> ↵

4. Rename the top-level hierarchichal EDIF netlist file to <*top-level design name*>**.edf**, if you have not already done so.

5. Use the **gen_hacf** utility to merge the **.iacf** files for the top-level design and subdesigns into a single hierarchical ACF file, called ***<top level design name>*.acf**. Type the following command at the **dc_shell** prompt to start the **gen_hacf** utility and merge the files:

   `gen_hacf` <*top-level design name*>`[`<*sub-design file list*>`]` ↵

Figure 2 shows a sample **dc_shell** script, which includes all of the steps for using the **gen_iacf** and **gen_hacf** utilities.

Figure 2. Sample Script for Running the gen_iacf and gen_hacf Utilities

```
/* Sample dc_shell script for converting hierarchical
Synopsys timing constraints to the ACF format
The example design TOP has 3 lower-level
subdesigns - LOWER1, LOWER2, LOWER3.  Only
LOWER1, LOWER2 and TOP designs have constraints. */

link_library            = flex10k-3.db
target_library          = flex10k-3.db
synthetic_library       = flex10k-3.sldb

read -f vhdl LOWER1.vhd
read -f vhdl LOWER2.vhd
read -f vhdl LOWER3.vhd
read -f vhdl TOP.vhd

elaborate LOWER1
current_design=LOWER1

/* Include user-defined timing constraints for LOWER1 */

include timing1.cmd
compile
design_name=LOWER1

/* generate input files for gen_iacf */

include /usr/maxplus2/synopsys/bin/gen_iacf.cmd

/* generate an intermediate ACF (.iacf) file for LOWER1 design */

sh /usr/maxplus2/synopsys/bin/gen_iacf LOWER1

elaborate LOWER2
current_design=LOWER2

/* Include user-defined timing constraints for LOWER2 */
```

```
include timing2.cmd
compile
design_name=LOWER2

/* generate input files for gen_iacf */

include /usr/maxplus2/synopsys/bin/gen_iacf.cmd

/* generate an intermediate ACF (.iacf) file for LOWER2 design */

sh /usr/maxplus2/synopsys/bin/gen_iacf LOWER2

elaborate TOP
current_design=TOP

/* Include user-defined timing constraints for TOP */

include timing3.cmd
compile

/* generate a hierarchical EDIF netlist file for
the top-level design before it is flattened by
the gen_iacf.cmd utility */

write -f edif -hierarchy TOP -o TOP.hier.edf

design_name=TOP

/* generate input files for gen_iacf */

include /usr/maxplus2/synopsys/bin/gen_iacf.cmd

/* generate an intermediate ACF (.iacf) file for design TOP */

sh /usr/maxplus2/synopsys/bin/gen_iacf TOP

/* Rename the hierarchical EDIF netlist file generated
earlier to <top level design>.edf, which is required by
gen_hacf utility and MAX+PLUS II */

sh mv TOP.hier.edf TOP.edf

/* Merge all .iacf files to generate the final top-level ACF
File subdesign.list in the following command lists the names
of subdesigns that have timing constraints, one per line.
In this example it has 2 lines, one each for LOWER1 and LOWER2.
Top-level design name should not be specified in this file. */

sh /usr/maxplus2/synopsys/bin/gen_hacf TOP subdesign.list

quit
```

The **gen_iacf** utility cannot support maximum Clock frequency ($\mathbf{f_{MAX}}$) correctly if more than one Clock skew is specified in the **dc_shell** command script. This problem occurs because the Synopsys `write_script` command drops the Clock skew information for the registers. The **gen_iacf** utility will use the last Clock skew number to calculate $\mathbf{f_{MAX}}$.

All timing assignments generated by the **gen_iacf** utility are written to the Timing Requirement Assignments Section of the project's ACF, with the assignment source identifier `{synopsys}` at the end of each line. Figure 4 shows a sample ACF excerpt that contains Synopsys timing constraints generated by the **gen_iacf** utility.

Figure 4. Sample ACF Excerpt with Synopsys Timing Constraints

```
TIMING_POINT
BEGIN
   "|OUT2"     : TCO = 15.00ns {synopsys};
   "|IN1"      : TPD = 10.00ns {synopsys};
   "|IN2"      : TPD = 5.00ns {synopsys};
   "|OUT1"     : TCO = 20.00ns {synopsys};
```

```
   "|IN1"      : TSU = 20.00ns {synopsys};
   "|IN2"      : TSU = 117.00ns {synopsys};
   "|CLK"      : FREQUENCY = 50.00ns {synopsys};
   "|n10_reg" : FREQUENCY = 100.00ns {synopsys};
END;
```

The MAX+PLUS II Compiler flattens the design internally before compiling it, which may convert some of the ports on the sub-designs into internal or buried nodes. In addition, the **gen_iacf** and **gen_hacf** utilities will correctly pass $\mathbf{t_{CO}}$ and $\mathbf{t_{PD}}$ assignments made at lower levels of hierarchy to the ACF, but the MAX+PLUS II Compiler will ignore them and generate one or more warning messages (e.g., `Warning: Ignored timing assignment for tsu|tpd|tco on buried node |TIME_STATE_MACHING:U1|tb1_3:U115|:30`). In addition, hierarchical timing constraints may result in duplicate assignments in the ACF, and the MAX+PLUS II Compiler could generate an additional warning (e.g., `Warning: Ignored redefinition of resources assignment (logic option assignment) for node 'CLK' Processing . . .`).

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Using Exemplar Logic Galileo Extreme & MAX+PLUS II Software

The following topic describes how to use the Exemplar Logic Galileo Extreme software with MAX+PLUS® II software. Choose one of the following topics for information:

Open a printable version of all topics listed on this page.

Setting Up the MAX+PLUS ® II/Mentor Graphics/Exemplar Logic Working Environment

- Software Requirements
- Altera-Provided Logic & Symbol Libraries
- Local Work Area Directory Structure
- Mentor Graphics Project Directory Structure
- MAX+PLUS II Project Directory Structure
- MAX+PLUS II/Mentor Graphics/Exemplar Logic Interface File Organization

Design Entry

- Design Entry Flow
- Creating VHDL & Verilog HDL Designs for Use with MAX+PLUS II Software
- Performing a Functional Simulation with QuickHDL Pro Software

Synthesis & Optimization

- Synthesizing & Optimizing VHDL & Verilog HDL Projects with Galileo Extreme Software

Related Links:

- Go to the following MAX+PLUS II ACCESS℠ Key topics for related information:
  - Compiling Projects with MAX+PLUS II Software
  - Programming Altera Devices
- Go to the following topics for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware
  - Exemplar Logic web site (http://www.exemplar.com)
  - Mentor Graphics web site (http://www.mentor.com)

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Exemplar Logic Galileo ExtremeSpecific Compiler Settings

If you are using MAX+PLUS ® II software to compile a FLEX ® design that was created with Galileo Extreme software, go through the following additional compilation steps:

1. Choose **Global Project Logic Synthesis** (Assign menu) to open the **Global Project Logic Synthesis** dialog box.

2. Select the appropriate logic synthesis style under *Global Project Logic Synthesis Style*:

   ✔ If you turned on the *Lock Lcells* option under *SYNTHESIS SWITCHES* in the Galileo Extreme **Altera FLEX Output Options** dialog box when synthesizing your design with Galileo Extreme software, select *WYSIWIG* in the *Global Project Synthesis Style* box.

   or:

   ✔ If you did not turn on the *Lock Lcells* option, select *FAST* in the *Global Project Synthesis Style* box.

3. - *Automatic Fast I/O*
   - *Automatic Register Packing*
   - (FLEX 10K devices only) *Automatic Implement in EAB*

   (Optional) Turning on one or more of the following options may help to improve area usage and timing delays:

4. Choose **OK** to close the **Global Project Logic Synthesis** dialog box.

5. Continue with the steps necessary to compile your project, as described in Compiling Projects with MAX+PLUS II Software.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Viewlogic Powerview Graphical User Interface & the Altera Toolbox

You use the Powerview graphical interface manager, the Cockpit, and the Altera® Toolbox to start all Powerview and Altera tools. Within the Altera Toolbox, you can specify the Max2 Express Drawer or the Design Tools Drawer to work with the Altera/Viewlogic Powerview interface.

The Max2 Express Drawer provides a quick and seamless way to transfer designs created in Powerview to the MAX+PLUS® II software for compilation, then return the compiled designs to Powerview for simulation and timing verification. Table 1 describes the Max2 Express Drawer tools.

Table 1. Max2 Express Drawer Tools

Tool	Description
max2_VDraw	Launches the Powerview ViewDraw schematic entry tool.
VHDL<->max2	Launches all tools necessary to synthesize a VHDL design, compile for an Altera device, and generate a **.vsm** file for simulation with the Powerview ViewSim simulator.
SCH<->max2	Launches all tools necessary to compile a schematic design entered with Powerview ViewDraw software for an Altera device and to generate a **.vsm** file for simulation with Powerview ViewSim and **.edo**, **.sdo**, and **.vmo** files for timing analysis with MOTIVE for Powerview.
max2_VSim	Launches the Powerview ViewSim simulator.
max2_VTrace	Launches the Powerview ViewTrace simulation waveform editor.
max2_MOTIVE	Launches the MOTIVE for Powerview ViewDraw static timing verification tool.

The Design Tools Drawer provides tools that enable you to create a design with the Powerview tools, compile the design in the MAX+PLUS II software, and simulate and verify the design with Powerview software. Table 2 describes the Design Tools Drawer tools.

Table 2. Design Tools Drawer Tools

Tool	Description
max2_VDraw	Launches the Powerview ViewDraw schematic entry tool.
max2_analyzer	Launches the Powerview VHDL Analyzer software.
max2_syn	Launches the Powerview VHDL synthesis tool.
max2_chk	Launches the Powerview schematic verification tool.
max2_vsmnet	Launches the Powerview **vsm** utility that converts a wirelist file into a **.vsm** file.
max2_VSim	Launches the Powerview ViewSim simulator.
max2_VTrace	Launches the Powerview ViewTrace simulator.
max2_edifo	Launches the Powerview EDIF netlist writer, **edifneto**.
max2_VGen	Launches the Powerview ViewGen utility that generates a schematic from a wirelist file.
max2	Launches the MAX+PLUS II Compiler.
max2_edifi	Launches the Powerview EDIF Netlist Reader, **edifneti**.
max2_vhdl2sym	Launches the Powerview **vhdl2sym** utility that generates a symbol from a VHDL file.
	Launches the Powerview Vantage VHDL Library Manager tool.

max2_VantgMgr	
max2_VantgAnlz	Launches the Vantage VHDL Analyzer software.
max2_VCS	Launches the Fusion/VCS Simulator.
max2_MOTIVE	Launches the MOTIVE for Powerview static timing verification tool.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Analyzing VHDL or Verilog HDL Designs with the Synplify HDL Analyst

You can use the optional Synplify HDL Analyst to analyze and evaluate the performance of your design graphically. The Synplify HDL Analyst instantly generates Register Transfer Level (RTL) schematics, as well as technology-mapped, gate-level schematics. You can instantly identify and fix potential problems earlier in the design cycle by cross-probing between the RTL schematics, gate-level schematics, and HDL source code. The Synplify HDL Analyst also highlights critical paths within the design to show which signals require optimization for performance. After you determine the critical speed paths, you can add timing constraints either to the VHDL or Verilog HDL source file or to a separate Synplify Design Constraints File (**.sdc**) to improve design performance.

To use the Synplify HDL Analyst after synthesizing your design with Synplify software, go through the following steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS® II/Synplicity Working Environment.
2. Create a VHDL or Verilog HDL design and save it in your working directory, as described in Creating VHDL Designs for Use with MAX+PLUS II Software or Creating Verilog HDL Designs for Use with MAX+PLUS II Software.
3. Synthesize and optimize your VHDL or Verilog HDL design with Synplify software, as described in Synthesizing & Optimizing VHDL or Verilog HDL Files with Synplify Software.
4. Choose an HDL Analyst view:

   ✔ Choose **RTL View** (HDL_Analyst menu) to view the RTL schematic. When you select this view, the HDL Analyst displays a graphical representation of the design and the mouse pointer becomes a plus (+) symbol.

   or:

   ✔ Choose **Technology View** (HDL_Analyst menu) to view the gate-level schematic. When you select this view, the HDL Analyst displays a graphical representation of the design and the mouse pointer becomes a plus (+) symbol.

5. In either the RTL or Technology View, perform one or more of the following actions:
   - Double-click the plus (+) symbol pointer on a port name or symbol to cross-probe your VHDL or Verilog HDL source design files.

     ☞ Because Synplify combines the `a + b` and `a - b` operations, cross-probing will highlight the Case Statement that defines both functions.

   - Choose **Find** (HDL_Analyst menu) to select specific signals quickly in your design.
   - Choose **Show Critical Path** (HDL_Analyst menu) to highlight the critical paths in the design.
   - Select **Filter Schematic** (HDL_Analyst menu) to show only the nodes you have selected.

6. If necessary, correct the design and repeat the steps described in Synthesizing & Optimizing VHDL or Verilog HDL Files with Synplify Software.
7. Process your design with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Performing a Functional Simulation of a Concept Schematic with the hdlconfig Utility & Verilog-XL Software

You can perform a functional simulation of a Concept schematic with the **hdlconfig** utility and Verilog-XL software before compiling your project with the MAX+PLUS® II software.

To functionally simulate a Concept schematic, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment.

2. Create a Concept schematic and save it in your working directory, as described in Creating Concept Schematics for Use with MAX+PLUS II Software.

3. Use the **hdlconfig** utility to create a Verilog HDL text file that contains the entire design. Type the following command at the UNIX prompt from the /<*working directory*>/<*design name*>/**source** directory:

   `hdlconfig -a -c -r` <*design name*> `-o` <*design name*>`.v logic verilog_lib` ↵

4. If your design contains RAM or ROM functions (e.g., `lpm_ram_dq`, `lpm_ram_io`, `lpm_rom`, `scfifo`, `dcfifo`, `altdpram`, and `csdpram`), run the **vconfig** utility to link the object **convert_hex2ver.o** to build a new Verilog-XL file that supports these functions by following these steps:

   1. Create a copy of the Verilog executable file by typing the following command at the UNIX prompt:

      `cp -p $CDS_INST_DIR/tools/verilog/bin/verilog $CDS_INST_DIR/tools /verilog/bin/verilog.bak.` ↵

   2. Type `vconfig` ↵ at the UNIX prompt from the **/usr/maxplus2/cadence/bin** directory to start the script.

   3. Accept `cr_vlog` as the name of the output script.

   4. Accept `1` as the stand-alone target.

   5. Type `new_verilog` as the name for the Verilog-XL target.

   6. Respond `Yes` when you are prompted to compile for the Verilog-XL environment.

   7. Respond `No` when you are prompted to include the Dynamic LAI, STATIC LOGIC AUTOMATION, LMSI HARDWARE MODELER, Verilog Mixed-Signal, and CDC interfaces in this executable.

   8. Respond `Yes` when you are prompted to include the Standard Delay File Annotator (SDF).

   9. Specify **/usr/maxplus2/verilog/veriuser.c** when you are asked the name of the user template file. For more information about the contents of the **veriuser.c** file, you can refer to the **veriuser.doc** file, which is available in the Cadence *Openbook* product documentation. To locate this document, start *Openbook*, and choose **Alphabetical List of Products** from the main menu. Scroll through the pages until you locate the *PLI 1.0 User Guide & Reference* in the PLI section, and then continue to scroll

through the document until you locate the **veriuser.doc** file under "Section A" and "PLI Code Examples."

10. When you are asked the name of files to be linked with the Verilog-XL simulator, specify the hexadecimal (Intel-format) conversion file **/usr/maxplus2/cadence/share/verilog/convert_hex2ver.o**, followed by a single period (.).

11. Run the output script **cr_vlog** to build the new Verilog-XL executable in the **/usr/maxplus2/cadence/bin** directory. Make sure that the `$CDS_INST_DIR/tools/bin` path appears at the beginning of the `PATH` statement in the **.cshrc** file.

12. If your C language library installation is different from the default location **/usr/lang/SC3.0.1**, type the following command at the UNIX prompt:

    `setenv C_DIR` <*C language library installation directory*> ↵

13. If successful, replace the old Verilog executable file with the new one by typing the following command at the UNIX prompt:

    `cp -p new_verilog $CDS_INST_DIR/tools/verilog/bin/verilog` ↵

1. Generate the stimulus file for the design and start the Verilog-XL simulator by typing the following command at the UNIX prompt from the /<*working directory*>/<*design name*>/**source** directory:

    `verilog -y /usr/maxplus2/simlib/concept/alt_max2/verilogUdps +libext+.v+.V` <*stimulus file name*> <*design name*>`.v` ↵

2. When you are ready to compile the project, generate an EDIF netlist file <*design name*>**.edf** with the **concept2alt** utility, as described in Converting Concept Schematics into MAX+PLUS II-Compatible EDIF Netlist Files with the **concept2alt** Utility.

3. Process the <*design name*>**.edf** file with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Creating VHDL & Verilog HDL Designs for Use with MAX+PLUSÂ II Software

You can create VHDL and Verilog HDL design files with the MAX+PLUS Â® Â II Text Editor or another standard text editor and save them in the appropriate directory for your project.

The MAX+PLUSÂ II Text Editor offers the following advantages:

- Templates are available with the **VHDL Templates** and **Verilog Templates** commands (Template menu). These templates are also available in the ASCII **vhdl.tmp** and **verilog.tmp** files, respectively, which are located in the **/usr/maxplus2** directory.
- If you use the MAX+PLUSÂ II Text Editor to create your VHDL design, you can turn on the **Syntax Coloring** command (Options menu). The Syntax Coloring feature displays keywords and other elements of text in text files in different colors to distinguish them from other forms of syntax.

To create a VHDL or Verilog HDL design file for use with the MAX+PLUSÂ II software, go through the following steps:

1. Enter a VHDL or Verilog HDL design in the MAX+PLUSÂ II Text Editor or another standard text editor and save it in your working directory.
2. Enter primitives, macrofunctions, and megafunctions in your VHDL or Verilog HDL design from the **Altera** library.

   The following topics describe special steps needed to instantiate LPM and `clklock` functions:

   - Instantiating LPM Functions in VHDL
   - Instantiating the clklock Megafunction in VHDL or VerilogÂ HDL

   ☞ You can instantiate MegaCore™ functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP™). The OpenCore™ feature in the MAX+PLUSÂ II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.
3. (Optional) Use the QuickHDL software to functionally simulate the design file, as described in Performing a Functional Simulation with QuickHDL Software and Performing a Functional Simulation with QuickHDL Pro Software.
4. Once you have created a VHDL or Verilog HDL design, you can generate an EDIF netlist file that can be imported into the MAX+PLUSÂ II software with either of the following methods:

   - You can synthesize and optimize your design and create an EDIF netlist file, as described in Synthesizing & Optimizing VHDL & Verilog HDL Projects with Galileo Extreme Software or Synthesizing & Optimizing VHDL & Verilog HDL Projects with Leonardo Software.
   - You can use the Altera VHDL Express utility, **vhd_exprss**, to automatically create an EDIF netlist file, compile it with the MAX+PLUSÂ II Compiler, generate an EDIF Output File (**.edo**), and prepare the EDIF Output File for simulation with QuickHDL software, as described in Using the Altera Schematic Express (**vhd_exprss**) Utility.

Installing the AlteraÂprovided MAX+PLUSÂ II/Mentor Graphics/Exemplar Logic interface on your computer automatically creates the following sample VHDL design files:

- **/usr/maxplus2/examples/mentor/example5/count4.vhd**
- **/usr/maxplus2/examples/mentor/example6/count8.vhd**
- **/usr/maxplus2/examples/mentor/example8/adder16.vhd**

Related Links:

- Go to Compiling Projects with MAX+PLUSÂ II Software in these MAX+PLUSÂ II ACCESS^{SM} Key topics for related information.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Creating Hierarchical Projects in Concept Schematics

If you wish to create a hierarchical design that contains symbols representing other design files, such as Altera® Hardware Description Language (AHDL) Text Design Files (**.tdf**), you can create a hollow-body symbol that represents a design file and then instantiate it in your Composer schematic.

To create a hierarchical project in your Composer schematic, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS® II/Cadence Working Environment.
2. Create a Composer schematic and save it in your working directory, as described in Creating Composer Schematics for Use with MAX+PLUS II Software.

   ☞ You can instantiate MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.

3. Create the hollow-body symbol <*design name*> in Composer by typing `icds` ↵ from the <*working directory*> at the UNIX prompt.
4. Choose **Library Path Editor** (Tools menu) to create the <*design name*> library. Type <*design name*> as the *Library* name and `./source/`<*design name*> as the *Path* name. Choose **Save** (File menu), then choose **Exit** (File menu) to save the path and exit from the Library Path Editor.
5. Choose **Library Manager** (Tools menu) to start Composer and create a symbol for your design. Type <*project directory name*> as the *Library* name, <*lower-level design name*> as the *Cell* name, `symbol` as the *View* name, and then press ↵.
6. Create a hollow-body symbol that represents the inputs and outputs of your design.
7. Enter input and output pins for the symbol.
8. Save the symbol.
9. To enter the symbol in your higher-level schematic design, choose the **Component** button and type <*project directory name*> as the *Library* name, <*lower-level design name*> as the *Cell* name, and `symbol` as the *View* name.
10. The MAX+PLUS II software uses the **cadence.lmf** Library Mapping File to map Concept symbols to equivalent MAX+PLUS II logic functions. To use custom symbols, you must create a custom LMF that maps your custom symbols to the equivalent EDIF Input File, Text Design File (TDF), or other design file. You will also need to specify this custom LMF in the **EDIF Netlist Reader Settings** dialog box before compiling with the MAX+PLUS II software. See Compiling Projects with MAX+PLUS II Software for more information.
11. Continue with the steps necessary to complete your Composer schematic, as described in Creating Composer Schematics for Use with MAX+PLUS II Software.

    Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the

following sample hierarchical AHDL and Composer schematic file:

- **/usr/maxplus2/examples/cadence/example5/fulladd2**

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

If you wish to create a hierarchical design that contains symbols representing other MAX+PLUS II-supported design files, such as Altera® Hardware Description Language (AHDL) Text Design Files (**.tdf**), you can create a hollow-body symbol that represents a design file and then instantiate it in your Concept schematic. To create a hierarchical project in your Concept schematic, go through the following steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS® II/Cadence Working Environment.
2. Create a Concept schematic and save it in your working directory, as described in Creating Concept Schematics for Use with MAX+PLUS II Software.

   ☞ You can instantiate MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.

3. Create the hollow-body symbol <*design name*> in Concept by typing the following command from the <*working directory*>/**source** directory that contains the lower-level design file <*design name*>.<*extension*>:

   `concept` <*design name*>`.body` ↵

4. Create a part file to indicate that the body is hollow:

   1. Add the `DEFINE` and `DRAWING` bodies to the part drawing. These bodies should be the only two bodies in the drawing.
   2. Add the `TITLE=`<*design name*> and the `ABBREV=`<*design name*> properties to the `DRAWING` body to identify the drawing.
   3. Save the part drawing with the name <*design name*>**.part.1.1**.

5. Regardless of the hardware description language (HDL) or schematic editor used to create the design, you must create a dummy Verilog HDL module to indicate to the **concept2alt** utility that the design is a "black box" that must pass untouched through the EDIF netlist file.

   1. Type `genview verilog` ↵ in the Concept window.
   2. Type `logic` ↵ when prompted for the Verilog *View* name.
   3. If you are using VerilogLink, you must type `genview verilog` again, then type `verilog_lib` ↵ when

prompted for the Verilog *View* name.

4. Type `cd` <*design name*>`/logic` ↵ at the UNIX prompt from the **/source** directory to change to the **/source/**<*design name*>**/logic** directory.

5. Edit the **verilog.v** file to add the `cds_action = "ignore"` parameter setting after the Input Declarations and Output Declarations sections. This parameter setting specifies that the <*design name*> should be treated as a "black box."

6. To enter the symbol in the higher-level Concept schematic, choose the **Add Part** button, choose the name of the working **SCALD** directory, then choose the <*design name*> symbol from the Symbol menu.

7. The MAX+PLUS II software uses the **cadence.lmf** Library Mapping File to map Concept symbols to equivalent MAX+PLUS II logic functions. To use custom symbols, you must create a custom LMF that maps your custom symbols to the equivalent EDIF Input File, Text Design File (TDF), or other design file. You will also need to specify this custom LMF in the **EDIF Netlist Reader Settings** dialog box before compiling with the MAX+PLUS II software. See Compiling Projects with MAX+PLUS II Software for more information.

8. Continue with the steps necessary to complete your Concept schematic, as described in Creating Concept Schematics for Use with MAX+PLUS II Software.

   Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample hierarchical AHDL and Concept schematic file:

   - **/usr/maxplus2/examples/cadence/example4/fulladd2**

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Creating Hierarchical Projects with Design Architect Software

If you wish to create a hierarchical schematic design that contains symbols representing other design files, such as AHDL Text Design Files (**.tdf**), VHDL Design Files (**.vhd**), or Verilog Design Files (**.v**), you can create a hollow-body symbol for the design file and then instantiate it in your top-level design file.

To create a hollow-body symbol for a lower-level design file, follow these steps:

1. (Optional) If you are creating a hollow-body symbol for a VHDL or Verilog HDL design file, you can first functionally simulate the VHDL or Verilog HDL file, as described in Performing a Functional Simulation with QuickHDL Software.
2. Start the Design Architect software by double-clicking Button 1 on the **max2_da** icon in the Design Manager tools window. You can also start Design Architect software by typing `max2_da` ↵ at the UNIX prompt.
3. Choose the **OPEN SYMBOL** button in the Design Architect session_palette to open the Symbol Editor. Type the lower-level design file name, including the directory path, in the *Component Name* box. Choose **OK**.
4. Create a symbol that represents the inputs and outputs of the lower-level file.
5. Assign `PINTYPE` properties of `IN` or `OUT` to the inputs and outputs of the symbol, and assign appropriate values to any other properties of the symbol so that it can be identified in the top-level schematic.

   ☞ If you are creating a hollow-body symbol for a VHDL design file, be sure to assign the value `qvpro` to the symbol's `model` property so that it can be identified as a VHDL component in the top-level schematic.
6. Check and save the symbol, then close the Symbol Editor.
7. To enter the symbol, choose the **CHOOSE SYMBOL** button from the Design Architect session_palette.
8. Select the symbol file from the Navigator menu and choose **OK**.
9. The MAX+PLUS® II software uses the Altera® provided **mnt8_bas.lmf** Library Mapping File to map Design Architect symbols to equivalent MAX+PLUS II logic functions. To use custom symbols, you must create a custom LMF that maps your custom symbols to the equivalent EDIF Input File, Text Design File (TDF), or other design file. You will also need to specify this LMF in the **EDIF Netlist Reader Settings** dialog box before compiling the design with the MAX+PLUS II software. See Compiling Projects with MAX+PLUS II Software for more information.
10. Continue with the steps necessary to complete your Design Architect schematic, as described in Creating Design Architect Schematics for Use with MAX+PLUS II Software.

Installing the Alteraprovided MAX+PLUS II/Mentor Graphics/Exemplar Logic interface on your computer automatically creates the sample hierarchical Design Architect schematic file **/usr/maxplus2/examples/mentor/example3/fulladd2**.

Feedback

Did this information help you?

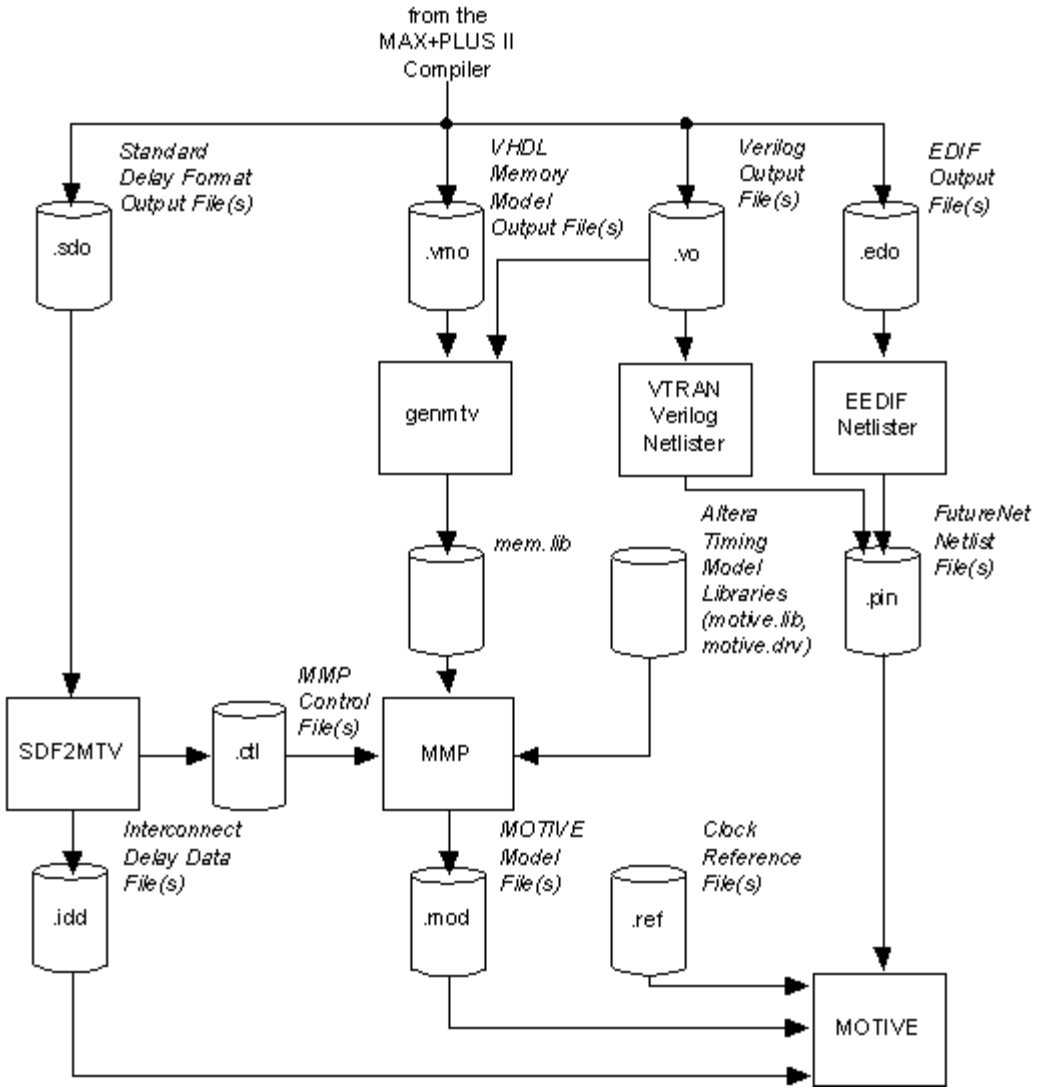
If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

MAX+PLUS II/Viewlogic Powerview Timing Verification Flow

Figure 1 shows the timing verification flow for the MAX+PLUS® II/Viewlogic Powerview interface.

Figure 1. MAX+PLUS II/Viewlogic Powerview Project Timing Verification Flow



Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability

for use of or reliance on the solution.

Initializing Registers in VHDL & Verilog Output Files for Power-Up before Simulation

Altera provides the **add_dc** script, which is availiable in the MAX+PLUS II system directory, to allow you to process MAX+PLUS II-generated Verilog Output Files (**.vo**) and VHDL Output Files (**.vho**) to prepare these files for simulation with another EDA tool. The add_dc script runs the **add_dclr** utility, which inserts a `device_clear` signal that is used for power-up initialization of all registers or flipflops in the design.

The script adds in a top-level signal named `device_clear` and connects it to the `CLRN` pin in all flipflops that should initialize to 0, and to the `PRN` pin of all flipflops that should initialize to 1. If the `CLRN` or `PRN` pin of a flipflop is already being used (i.e., is already connected to a signal), the script modifies the Verilog Output File or VHDL Output File so that the `AND` of the original signal and the `device_clear` pin feed the `CLRN` or `PRN` pin.

To use the **add_dc** script to process Verilog Output Files and VHDL Output Files before simulation with another EDA tool, follow these steps:

1. Make sure that your design file is located in the current directory, or change to the directory in which the design file is located.
2. Type the following command at the command prompt:

   `Â¥`<*path name of* ***add_dc.bat*** *file*>`Â¥add_dc` <*design name*> <*path name of* ***add_dclr.exe*** *file*> ↵

For example, if the both the **add_dc.bat** and the **add_dclr.exe** files are located in the **d:Â¥maxplus2Â¥exew** directory, and the **d:Â¥maxplus2Â¥exew** directory is specified in the search path, you can type the following command at a command prompt to add a `device_clear` signal to a design named `myfifo` in the file **myfifo.vo**:

```
add_dc myfifo d:Â¥maxplus2Â¥exew ↵
```

1. The **add_dc** script gives a message if the directory contains both a VHDL Output File and a Verilog Output File with the same name (<*design name*>**.vo** and <*design*>**.vho**). You should delete or rename whichever of those files should not have the `device_clear` signal added. The **add_dc** script can modify only one design file at a time.

2. When the **add_dc** script processes the Verilog Output File or VHDL Output File, it creates a backup copy of the original file, with the extension **.ori**.
3. The **add_dc** script works only for Verilog Output Files and VHDL Output Files that are generated by MAX+PLUS II.

After you have used the **add_dc** script and are ready to simulate the resulting Verilog Output File or VHDL Output File with another EDA tool, you should assert the active low `device_clear` pin for a period of time that is long enough for the design to initialize. You can then de-assert the pin, and apply simulation vectors to the design.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Altera VHDL & Verilog HDL alt_mf Logic Function Library

The **alt_mf** library contains behavioral VHDL and Verilog HDL models of the Altera® logic functions shown in Table 1. VHDL or Verilog HDL files that instantiate these functions can be simulated with the VHDL System Simulator (VSS) software or the Cadence Verilog-XL simulator, respectively, both before and after being compiled with the Synopsys Design Compiler or FPGA Compiler software.

Table 1. Altera-Provided Architecture Control Logic Functions

Name	Description
`a_8fadd`	8-bit full adder
`a_8mcomp`	8-bit magnitude comparator
`a_8count`	8-bit up/down counter
`a_81mux`	8-to-1 multiplexer

For detailed information on these functions, choose **Search for Help** on from the MAX+PLUS® II Help menu and type the function name, without the "`a_`" prefix.

The behavioral descriptions of these four functions are contained in the **/usr/maxplus2/synopsys/library/alt_mf/src** directory, which contains the following files:

File:	Description:
mf.vhd	Contains behavioral VHDL descriptions of the logic functions.
mf_components.vhd	Contains VHDL Component Declarations for the logic functions.
mf.v	Contains behavioral Verilog HDL descriptions of the logic functions.

If you wish to simulate a VHDL design containing these logic functions, you can use the Altera-provided shell script **analyze_vss** to create a design library called **altera**. This library allows you to reference the functions through the VHDL Library and Use Clauses, which direct the Design Compiler or FPGA Compiler software to incorporate the library files when it compiles your top-level design file. The **analyze_vss** shell script creates the **altera** design library by analyzing the VHDL System Simulator (VSS) simulation models in the **/usr/maxplus2/synopsys/library/alt_mf/lib** directory. See Setting Up VSS Configuration Files for more information on using the **analyze_vss** shell script.

Complete VHDL and Verilog HDL behavioral descriptions of these logic functions are included in the **mf.vhd** and **mf.v** files so that you can optionally retarget your design to other technology libraries.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability

for use of or reliance on the solution.

Instantiating RAM & ROM Functions in VHDL

The MAX+PLUS® II /Synopsys interface offers full support for the memory capabilities of the FLEX® 10K device family, including synchronous and asynchronous RAM and ROM, cycle-shared dual port RAM, dual-port RAM, single-Clock FIFO, and dual-clock FIFO functions. You can use the Altera® -provided **genmem** utility to generate functional simulation models and timing models for these functions. Type `genmem` ↵ at the UNIX prompt to display information on how to use this utility, as well as a list of the functions you can generate.

To instantiate a RAM or ROM function in VHDL, follow these steps:

1. Use the **genmem** utility to generate a memory model by typing the following command at the UNIX prompt:

   `genmem` <*memory type*> <*memory size*> `-vhdl` ↵

   For example: `genmem asynrom 256x15 -vhdl` ↵

2. Create a VHDL design that incorporates the text from the **genmem**-generated Component Declaration, <*memory name*>**.cmp**, and instantiate the <*memory name*> function.

   Figure 1 shows a VHDL design that instantiates **asyn_rom_256x15.vhd**, a 256 x 15 ROM function.

   Figure 1. VHDL Design File with ROM Instantiation (tstrom.vhd)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;



ENTITY tstrom IS
        PORT (
          addr    : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
          memenab : IN STD_LOGIC;
          q       : OUT STD_LOGIC_VECTOR (14 DOWNTO 0));
END tstrom;



ARCHITECTURE behavior OF tstrom IS

COMPONENT asyn_rom_256x15
-- pragma translate_off
     GENERIC (LPM_FILE : string);



-- pragma translate_on
     PORT (Address : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
           MemEnab : IN STD_LOGIC;
           Q       : OUT STD_LOGIC_VECTOR(14 DOWNTO 0)
     );
END COMPONENT;



BEGIN

   u1: asyn_rom_256x15
-- pragma translate_off
        GENERIC MAP (LPM_FILE => "u1.hex")
-- pragma translate_on
   PORT MAP (Address => addr, MemEnab => memenab, Q =>q);
```

```
END behavior;
```

3. (Optional for RAM functions) Specify an initial memory content file:
   - For ROM functions, you must specify the filename of an initial memory content file in the Intel hexadecimal format (**.hex**) or the Altera® Memory Initialization File (**.mif**) format in the Generic Map Clause, with the `LPM_FILE` parameter. See Figure 1. The filename must be the same as the instance name; e.g., the `u1` instance name must be unique throughout the whole project, and must contain only valid VHDL name characters. The initialization file must reside in the directory containing the project's design files.
   - For RAM functions, specifying a memory initialization file is optional. If you want to use it, you must specify it in the Generic Map Clause as described above. If you do not use an initialization file, you should not declare or use the Generic Clause.
     1. The MIF format is supported only for specifying initial memory content when compiling designs within MAX+PLUS II software. You cannot use a MIF to perform simulation with Synopsys tools prior to MAX+PLUS II compilation.

     2. If you use an Intel hexadecimal format file and wish to simulate the design with the VHDL System Simulator (VSS) after MAX+PLUS II compilation, you should use the Synopsys **intelhex** utility to translate the Intel hexadecimal fomat file into a VSS-compatible Synopsys memory file. Refer to the Synopsys ***VHDL System Simulator Software Tool*** manual for details about using the **intelhex** utility.
4. In the VHDL design file, add the compiler directive `-- pragma translate_off` ↵ before the Generic Clause and Generic Map Clause, and add `-- pragma translate_on`↵ after the Generic Clause and Generic Map Clause. These directives tell the VHDL Compiler software when to stop and start synthesizing. For example, in Figure 1, the `--pragma translate_off` directive instructs the VHDL Compiler software to skip syntax checking until the `--pragma translate_on` directive is read.
5. Because the VHDL Compiler software does not support the data type `string` for the Generic Clause, you must also enter the following command before you read the design:

   ```
   hdlin_translate_off_skip_text=true↵
   ```

6. The timing model (**.lib**) generated by the **genmem** utility contains pin-to-pin delay information that can be used by the Synopsys Design Compiler and FPGA Compiler software. You must add this timing model to the existing library so that the compiler can access the timing information. Type the following commands at the **dc_shell** prompt:

   ```
   read -f db flex10k[<speed grade>].db ↵
   update_lib flex10k[<speed grade>] <RAM/ROM function name>.lib↵
   ```

7. (Optional) Enter the following command to update your **flex10k[***<speed grade>***].db** file with the RAM/ROM timing information:

   ```
   write_lib flex10k[<speed grade>] -o flex10k.db↵
   ```

8. When you generate the EDIF netlist file from the design, include the bus structure from the RAM or ROM function(s). Go to Setting Up Synopsys Configuration Files for more information.

9. Continue with the steps necessary to complete your VHDL design, as described in Creating VHDL Designs for Use with MAX+PLUS II Software.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Instantiating RAM & ROM Functions in Verilog HDL

The MAX+PLUS® II /Synopsys interface offers full support for the memory capabilities of the FLEX® 10K device family, including synchronous and asynchronous RAM and ROM, cycle-shared dual port RAM, dual-port RAM, single-Clock FIFO, and dual-clock FIFO functions. You can use the Altera® -provided **genmem** utility to generate functional simulation models and timing models for these functions. Type `genmem` ↵ at the UNIX prompt to display information on how to use this utility, as well as a list of the functions you can generate.

To instantiate a RAM or ROM function in Verilog HDL, follow these steps:

1. Use the **genmem** utility to generate a memory model by typing the following command at the UNIX prompt:

   `genmem` <*memory type*> <*memory size*> `-verilog` ↵

   For example: `genmem asynrom 256x15 -verilog` ↵

2. Create a Verilog HDL design that instantiates the <*memory name*> function.

   Figure 1 shows a Verilog HDL design that instantiates **asyn_rom_256x15.v**, a 256 x 15 ROM function.

   Figure 1. Verilog HDL File with ROM Instantiation (tstrom.v)

```
module tstrom (addr, enab, q);
parameter LPM_FILE = "u1.hex"
input [7:0] addr;
input enab;
output [14:0] q;



asyn_rom_256x15
// synopsys translate_off
           #(LPM_FILE)



// synopsys translate_on
      u1  (.Address(addr), .Q(q), .MemEnab(enab));

endmodule
```

3. (Optional for RAM functions) Specify an initial memory content file:
   - For ROM functions, you must specify the filename of an initial memory content file in the Intel hexadecimal format (**.hex**) or the Altera® Memory Initialization File (**.mif**) format in the Parameter Statement, with the `LPM_FILE` parameter. See Figure 1. The filename must be the same as the instance name; e.g., the `u1` instance name must be unique throughout the whole project. The initialization file must reside in the directory containing the project's design files.
   - For RAM functions, specifying a memory initialization file is optional. If you want to use it, you must specify it in a Parameter Statement, as described above.
     1. The MIF format is supported only for specifying initial memory content when compiling designs within MAX+PLUS II software. You cannot use a MIF to perform simulation with Synopsys tools prior to MAX+PLUS II compilation.

     ☞ 2. If you use an Intel hexadecimal format file and wish to simulate the design with the VHDL System Simulator (VSS) after MAX+PLUS II compilation, you should use the Synopsys

intelhex utility to translate the Intel hexadecimal fomat file into a VSS-compatible Synopsys memory file. Refer to the Synopsys ***VHDL System Simulator Software Tool*** manual for details about using the **intelhex** utility.

4. In the Verilog HDL design, add `// synopsys translate_off` ↵ before the Parameter Statement, and add `// synopsys translate_on` ↵ after the Parameter Statement. These directives tell the HDL Compiler for Verilog when to stop and start synthesizing. See Figure 1.
5. The timing model (**.lib**) generated by the **genmem** utility contains pin-to-pin delay information that can be used by the Synopsys Design Compiler and FPGA Compiler software. You must add this timing model to the existing library so that the compiler can access the timing information. Type the following commands at the **dc_shell** prompt:

   `read -f db flex10k[`<*speed grade*>`].db` ↵
   `update_lib flex10k[`<*speed grade*>`]` <*RAM/ROM function name*>`.lib`↵
6. (Optional) Include the following command to update your **flex10k[**<*speed grade*>**].db** file with the RAM/ROM timing information:

   `write_lib flex10k[`<*speed grade*>`] -o flex10k.db`↵
7. When you generate the EDIF netlist file from the design, include the bus structure from the RAM or ROM function(s). Go to Setting Up Synopsys Configuration Files for more information.
8. Continue with the steps necessary to complete your Verilog HDL design, as described in Creating Verilog HDL Designs for Use with MAX+PLUS II Software.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

MAX+PLUS II Architecture Control Logic Function Instantiation Example for VHDL

You can instantiate Altera® -provided logic functions from the **alt_mf** library, which includes the `a_8fadd`, `a_8mcomp`, `a_8count`, and `a_81mux` functions, in VHDL designs. Altera provides behavioral descriptions of these functions that support pre-synthesis/pre-route simulation of your top-level design with the VHDL System Simulator (VSS).

When you instantiate one of these functions, you can either include a Component Declaration for the function, or use the Altera-provided shell script **analyze_vss** to create a design library called **altera** so that you can reference the functions through the VHDL Library and Use Clauses. The Library and Use Clauses direct the Design Compiler or FPGA Compiler to incorporate the library files when it compiles your top-level design file. The **analyze_vss** shell script creates the **altera** design library when it analyzes the VSS simulation models in the **/usr/maxplus2/synopsys/library/alt_mf/lib** directory. See Setting up VSS Configuration Files for more information on using the **analyze_vss** shell script.

Figure 1 shows an example of an 8-bit counter that is instantiated using the `a_8count` function.

Figure 1. Sample VHDL File with Logic Function Instantiation

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;



LIBRARY altera;
USE altera.maxplus2.ALL;



ENTITY counter IS
PORT (clock,ena,load,dnup,set,clear : IN STD_LOGIC;
   i     : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
   q     : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);



cout : OUT STD_LOGIC);
END counter;



ARCHITECTURE structure OF counter IS



BEGIN
   u1    : a_8count



PORT MAP (a=>i(0), b=>i(1), c=>i(2), d=>i(3), e=>i(4),
            f=>i(5), g=>i(6), h=>i(7), ldn=>load, gn=>ena,
            dnup=>dnup, setn=>set, clrn=>clear, clk=>clock,
```

```
qa=>q(0), qb=>q(1), qc=>q(2), qd=>q(3), qe=>q(4),
            qf=>q(5), qg=>q(6), qh=>q(7), cout=>cout);



END structure;



CONFIGURATION conf OF counter IS
   FOR structure
   END FOR;
END conf;
```

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

MAX+PLUS II Architecture Control Logic Function Instantiation Example for Verilog HDL

You can instantiate Altera® -provided logic functions from the **alt_mf** library, which includes the `a_8fadd`, `a_8mcomp`, `a_8count`, and `a_81mux` functions, in Verilog HDL designs. Altera provides behavioral Verilog HDL descriptions of these functions.

Figure 1 shows an example of an 8-bit counter that is instantiated using the `a_8count` function. Because Verilog HDL is case-sensitive, be sure to use uppercase letters for all of the macrofunction's module names and port names.

Figure 1. Sample Verilog HDL File with Logic Function Instantiation (counter.v)

```
module counter (clock, ena, load, dnup, set, clear, i, q, cout);
output          cout;
output[7:0]     q;
input[7:0]      i;
input           clock, ena, load, dnup, set, clear;
A_8COUNT u1    (.A(i[0]), .B(i[1]), .C(i[2]), .D(i[3]),
                .E(i[4]), .F(i[5]), .G(i[6]), .H(i[7]),
                .LDN(load), .GN(ena), .DNUP(dnup), .SETN(set),
                .CLRN(clear), .CLK(clock), .QA(q[0]), .QB(q[1]),
                .QC(q[2]), .QD(q[3]), .QE(q[4]), .QF(q[5]),
                .QG(q[6]), .QH(q[7]), .COUT(cout) );
endmodule
```

The sample file shown in Figure 1 can be synthesized with the Design Compiler or FPGA Compiler. You can also simulate it with the Cadence Verilog-XL Simulator by typing the following command at the **dc_shell** prompt:

```
verilog counter.v /usr/maxplus2/synopsys/library/alt_mf/src/mf.v↵
```

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Primitive & Old-Style Macrofunction Instantiation Example for Verilog HDL

You can instantiate the MAX+PLUS® II primitives listed in Design Compiler & FPGA Compiler Technology Libraries in Verilog HDL designs. These primitives can be used to control synthesis in the MAX+PLUS II software. You can also instantiate MAX+PLUS II megafunctions and old-style macrofunctions.

Go to the following topics for information and examples of how to instantiate functions that are not considered to be hollow bodies, including functions in the **alt_mf** library, RAM and ROM, and the `clklock` megafunction:

- Architecture Control Macrofunction Instantiation Example for Verilog HDL
- Instantiating RAM & ROM Functions in Verilog HDL
- Instantiating the clklock Megafunction in VHDL or Verilog HDL

Unlike other logic functions, MAX+PLUS II primitives do not need to be defined with hollow-body functions unless you wish to simulate the design with the VHDL System Simulator (VSS) software. Any references to these primitives are resolved by the Synopsys compilers. All buffer primitives except the `ATRIBUF` and `TRIBUF` primitives also have a "don't touch" attribute already assigned to them, which prevents the Synopsys compilers from optimizing them. The Synopsys compilers also automatically treat mega- and macrofunctions that do not have corresponding synthesis library models as "black boxes."

Figure 1 shows a 4-bit full adder with registered output that also instantiates an AGLOBAL or GLOBAL primitive. This figure also illustrates the use of global Clock and global Reset pins in the MAX 7000 architecture. The design uses an old-style 7483 macrofunction, which is represented as a hollow body named fa4.

Figure 1. 4-Bit Adder Design with Registered Output (adder.v)

```
module adder (a, b, clk, rst, cout, regsum);

output       cout;
output[4:1]  regsum;
input[4:1]   a, b;
input        clk, rst;
wire[4:1]    sum;
reg[4:1]     regsum_int;
wire         grst, gclk;
wire         ci;
assign       ci = 0;



// module instantiation
fa4    u0  ( .c0(ci), .a1(a[1]), .b1(b[1]), .a2(a[2]),
             .b2(b[2]), .a3(a[3]), .b3(b[3]), .a4(a[4]),
             .b4(b[4]), .s1(sum[1]), .s2(sum[2]),
             .s3(sum[3]), .s4(sum[4]), .c4(cout));
// For FLEX devices, GLOBAL, A_IN, and A_OUT should be replaced
// with AGLOBAL, IN1, and Y, respectively
GLOBAL  u1 ( .A_IN(clk), .A_OUT(gclk));
GLOBAL  u2 ( .A_IN(rst), .A_OUT(grst));
```

```
always @(posedge gclk or negedge grst)
        if ( !grst )
                regsum_int = 4'b0;
        else regsum_int = sum;
assign regsum = regsum_int;
endmodule



// module declaration for fa4 module
module fa4 ( c0, a1, b1, a2, b2, a3, b3, a4, b4, s1, s2, s3, s4, c4);



output    s1, s2, s3, s4, c4;
   input     c0, a1, b1, a2, b2, a3, b3, a4, b4;
   endmodule



// module declaration for GLOBAL primitive
// For FLEX devices, GLOBAL, A_IN, and A_OUT should be replaced
// with AGLOBAL, IN1, and Y, respectively
module GLOBAL (A_OUT, A_IN);



input       A_IN;
   output      A_OUT;
   endmodule
```

You can analyze the 4-bit adder design with the Synopsys HDL Compiler for Verilog software. The hollow-body description of the `fa4` function is required. It contains port declarations and does not include any information about the design's function or operation. However, the hollow-body description can be in the design file, as shown in Figure 1, or in a separate file, as shown in Figure 2.

Figure 2. Hollow-Body Description of a 4-Bit Full Adder (7483)

```
module fa4 ( c0, a1, b1, a2, b2, a3, b3, a4, b4, s1, s2, s3, s4, c4);
   output    s1, s2, s3, s4, c4;
   input     c0, a1, b1, a2, b2, a3, b3, a4, b4;
   endmodule
```

If the hollow-body description is in a separate file, you must analyze it before analyzing the higher-level function with the HDL Compiler for Verilog to produce a hollow-body component. This component contains a single level of hierarchy with input and output pins, but does not contain any underlying logic.

You can save the synthesized design as an EDIF netlist file (**.edf**) and compile it with the MAX+PLUS II software. After the HDL Compiler for Verilog software has successfully processed the design, it generates the schematic shown in Figure 3, which you can view with the Design Analyzer software.

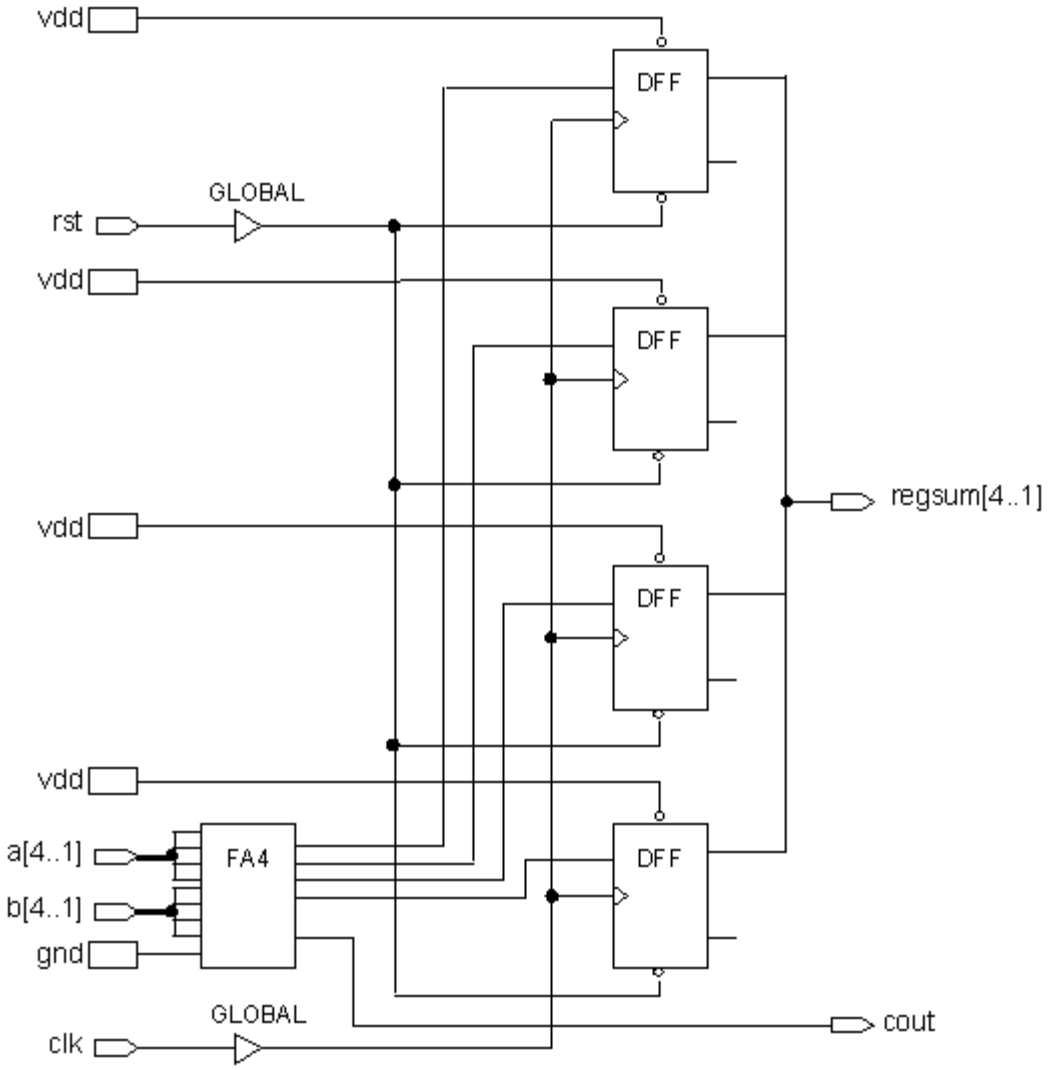


Figure 3. Synthesized Design Generated by the Design Compiler

However, before you compile the EDIF netlist file with the MAX+PLUS II software, you must create the **adder.lmf** file, shown in Figure 3, to map the `fa4` function to the equivalent MAX+PLUS II function (`7483`). You must then specify the LMF as *LMF #2* in the expanded **EDIF Netlist Reader Settings** dialog box (Interfaces menu) (*LMF #1* is **altsyn.lmf**). For more information about creating LMFs, refer to "Library Mapping Files (**.lmf**)" and "Library Mapping File Format" in MAX+PLUS II Help.

Figure 3. Library Mapping File Excerpt for fa4

```
BEGIN
FUNCTION 7483  (c0, a1, b1, a2, b2, a3, b3, a4, b4,)
RETURNS        (s1, s2, s3, s4, c4)



FUNCTION "fa4" ("c0", "a1", "b1", "a2", "b2", "a3",
               "b3","a4", "b4")
RETURNS        ("s1", "s2", "s3", "s4", "c4")
END
```

When you compile the design with the MAX+PLUS II software, you can disregard the warning "`EDIF cell <`*name*`> already has LMF mapping so CONTENTS construct has been ignored`". To verify the global Clock and global Reset usage, as well as the number of logic cells used, see the **adder.rpt** Report File generated by the MAX+PLUS II Compiler.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Setting Up the MAX+PLUS II/Cadence Working Environment

To use MAX+PLUS® II software with Cadence software, you must first install the MAX+PLUS II software, then establish an environment that facilitates entering and processing designs. The MAX+PLUS II/Cadence interface is installed automatically when you install the MAX+PLUS II software on your computer. Go to *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual for more information on installation and details on the directories that are created during MAX+PLUS II installation. Go to MAX+PLUS II/Cadence Interface File Organization for information about the MAX+PLUS II/Cadence directories that are created during MAX+PLUS II installation.

The information presented here assumes that you are using the C shell and that your MAX+PLUS II system directory is **/usr/maxplus2**. If not, you must use the appropriate syntax and procedures to set environment variables for your shell.

To set up your working environment for the MAX+PLUS II/Cadence interface, follow these steps:

1. Ensure that you have correctly installed the MAX+PLUS II and Cadence software versions described in the MAX+PLUS II/Cadence Software Requirements.
2. Add the following environment variables to your **.cshrc** file:

   ```
   setenv ALT_HOME /usr/maxplus2 ↵

   setenv CDS_INST_DIR <Cadence system directory path> ↵
   ```

3. Add the **$ALT_HOME/cadence/bin** and **$CDS_INST_DIR/tools/bin** directories to the `PATH` environment variable in your **.cshrc** file. Make sure these paths are placed before the Cadence hierarchy path.
4. Add **/usr/dt/lib** and **/usr/ucb/lib** to the `LD_LIBRARY_PATH` environment variable in your **.cshrc** file.
5. Create a new **cds.lib** file in your working directory or edit an existing one so that it includes all of the following lines that apply to the Cadence tools you have installed:

   ```
   DEFINE alt_syn ${ALT_HOME}/simlib/concept/alt_syn

   DEFINE lpm_syn ${ALT_HOME}/simlib/concept/lpm_syn

   DEFINE alt_lpm ${ALT_HOME}/simlib/concept/alt_lpm

   DEFINE alt_mf ${ALT_HOME}/simlib/concept/alt_mf

   DEFINE alt_max2 ${ALT_HOME}/simlib/concept/alt_max2

   DEFINE alt_max2 ${ALT_HOME}/simlib/composer/alt_max2/alt_max2

   DEFINE alt_vtl $ALT_HOME/simlib/concept/alt_vtl/lib

   DEFINE altera $ALT_HOME/simlib/concept/alt_mf/lib

   SOFTINCLUDE $CDS_INST_DIR/tools/leapfrog/files/cds.lib
   ```

```
DEFINE <design name>.
```

6. Copy the **/usr/maxplus2/maxplus2.ini** file to your **$HOME** directory:

```
cp /usr/maxplus2/maxplus2.ini $HOME↵

chmod u+w $HOME/maxplus2.ini↵
```

The **maxplus2.ini** file contains both Altera- and user-specified initialization parameters that control the MAX+PLUS II software, such as MAX+PLUS II symbol and logic function library paths and the current project name. The MAX+PLUS II installation procedure creates and copies the **maxplus2.ini** file to the **/usr/maxplus2** directory.

Normally, you do not have to edit your local copy of **maxplus2.ini** because the MAX+PLUS II software updates the file automatically whenever you change any parameters or settings. However, if you move the **max2lib** and **max2inc** library subdirectories, you must update the file. Go to "Creating & Using a Local Copy of the **maxplus2.ini** File" in MAX+PLUS II Help for more information.

7. If you are using Concept on a Sun SPARCstation running SunOS, go to Setting Up the MAX+PLUS II/Cadence Concept Work Environment for a Sun SPARCstation Running SunOS Software to install the **redifnet** EDIF netlist reader utility.

8. If you are using Synergy software, edit the **hdl.var** file located in your working directory to include the following line:

```
DEFINE work <design name>↵
```

9. Set up an appropriate directory structure for the tool(s) you are using. See the following topics for information:

   - Composer Project File Directory Structure
   - Concept & RapidSIM Local Work Area Directory Structure

Related Links:

Go to the following topics, which are available on the web, for additional information:

- ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
- This manual is also available in 4 parts:
  - Preface & Section 1: MAX+PLUS II Installation
  - Section 2: MAX+PLUS II - A Perspective
  - Section 3: MAX+PLUS II Tutorial
  - Appendices, Glossary & Index

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Performing a Timing Simulation with Leapfrog Software

Once the MAX+PLUS® II software has compiled a project and generated a VHDL Output File (**.vho**), you can a perform timing simulation using Cadence Leapfrog software.

To simulate a VHDL output file with the Leapfrog timing simulator, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment.
2. If you wish to use MAX+PLUS II-generated Standard Delay Format (SDF) Output Files (**.sdo**) that contain timing information, compile the VITAL library source files, as described in Compiling the VITAL Library for Use with Leapfrog Software.
3. If your design uses functions from the **alt_mf** library, compile the library, as described in Compiling the **alt_mf** Library.
4. Generate a VHDL Output File (**.vho**) and an optional SDF Output File, as described in Compiling Projects with MAX+PLUS II Software.
5. Using any standard text editor, create a stimulus file that includes test vectors for <*design name*>.
6. Start the Leapfrog simulator and simulate the MAX+PLUS II-created VHDL Output File <*design name*>**.vho** by typing `leapfrog` ↵ at the UNIX prompt. Refer to *Chapter 5: SDF Back-Annotation in Leapfrog* in the ***VHDL Simulator User Guide*** or refer to the ***Cadence Openbook*** for more information.

<<<<<<< leapfrog.htm

Technical Feedback

=======

Feedback

>>>>>>> 1.7

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Using Cadence Leapfrog & MAX+PLUS II Software

The following topics describe how to use the Cadence Leapfrog software with the MAX+PLUS® II software. Choose one of the following topics for information:

Open a printable version of all topics listed on this page.

Setting Up the MAX+PLUS II/Cadence Working Environment

- Software Requirements
- MAX+PLUS II Directory Structure
- MAX+PLUS II/Cadence Interface File Organization
- Compiling the VITAL Library for Use with Leapfrog Software
- Compiling the **alt_mf** Library

Simulation

- Project Simulation Flow
- Initializing Registers in VHDL & Verilog Output Files for Power-Up before Simulation
- Performing a Timing Simulation with Leapfrog Software

Related Links:

- Compiling Projects with MAX+PLUS II Software
- Programming Altera Devices
- MAX+PLUS II Development Software
- Altera Programming Hardware
- Cadence web site (http://www.cadence.com)

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Using Exemplar Logic Leonardo & MAX+PLUS II Software

The following topic describes how to use the Exemplar Logic Leonardo software with MAX+PLUS® II software. Choose one of the following topics for information:

Open a printable version of all topics listed on this page.

Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment

- Software Requirements
- Altera-Provided Logic & Symbol Libraries
- Local Work Area Directory Structure
- Mentor Graphics Project Directory Structure
- MAX+PLUS II Project Directory Structure
- MAX+PLUS II/Mentor Graphics/Exemplar Logic Interface File Organization

Design Entry

- Design Entry Flow
- Creating VHDL & Verilog HDL Designs for Use with MAX+PLUS II Software
- Performing a Functional Simulation with QuickHDL Pro Software

Synthesis & Optimization

- Synthesizing & Optimizing VHDL & Verilog HDL Projects with Leonardo Software

Related Links

- Compiling Projects with MAX+PLUS II Software
- Programming Altera Devices
- MAX+PLUS II Development Software
- Altera Programming Hardware
- Exemplar Logic web site (http://www.exemplar.com)
- Mentor Graphics web site (http://www.mentor.com)

Using Exemplar Logic Leonardo & MAX+PLUS II Software

The following topic describes how to use the Exemplar Logic Leonardo software with MAX+PLUS® II software. Click on one of the following topics for information:

This file is suitable for printing only. It does not contain hypertext links that allow you to jump from topic to topic.

Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment

- Software Requirements
- Altera-Provided Logic & Symbol Libraries
- Local Work Area Directory Structure
- Mentor Graphics Project Directory Structure
- MAX+PLUS II Project Directory Structure
- MAX+PLUS II/Mentor Graphics/Exemplar Logic Interface File Organization

Design Entry

- Design Entry Flow
- Creating VHDL & Verilog HDL Designs for Use with MAX+PLUS II Software
- Performing a Functional Simulation with QuickHDL Pro Software

Synthesis & Optimization

- Synthesizing & Optimizing VHDL & Verilog HDL Projects with Leonardo Software

Related Links:

- Go to the following MAX+PLUS II ACCESS℠ Key topics for related information:
  - Compiling Projects with MAX+PLUS II Software
  - Programming Altera Devices
- Go to the following topics, which are available on the web, for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware
  - Exemplar Logic web site (http://www.exemplar.com)
  - Mentor Graphics web site (http://www.mentor.com)

Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment

To use the MAX+PLUS ® II software with Mentor Graphics/Exemplar Logic software, you must install the

MAX+PLUS II software, then establish an environment that facilitates entering and processing designs. The MAX+PLUS II/Mentor Graphics/Exemplar Logic interface is installed automatically when you install the MAX+PLUS II software on your computer.

Go to *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual for more information on installation and details on the directories that are created during MAX+PLUS II installation. Go to MAX+PLUS II/Mentor Graphics/Exemplar Logic Interface File Organization for information about the MAX+PLUS II/Mentor Graphics directories that are created during MAX+PLUS II installation.

The information presented here assumes that you are using a C shell and that your MAX+PLUS II system directory is **/usr/maxplus2**. If not, you must use the appropriate syntax and procedures to set environment variables for your shell.

To set up your working environment for the MAX+PLUS II/Mentor Graphics interface, follow these steps:

1. Ensure that you have correctly installed the MAX+PLUS II and Mentor Graphics software versions described in MAX+PLUS II/Mentor Graphics Software Requirements.

2. Add the following environment variables to your **.cshrc** file:

```
setenv ALT_HOME /usr/maxplus2 ↵
setenv MGC_WD <user-specified working directory> ↵
setenv MGC_HOME <Mentor Graphics system directory> ↵
setenv MAX2_MENTOR /usr/maxplus2/mentor/max2 ↵
setenv MGC_LOCATION_MAP <user-specified location_map file> ↵
setenv EXEMPLAR <Galileo or Leonardo system directory> ↵
```

   Installing the Altera® provided MAX+PLUS II/Mentor Graphics interface on your computer automatically installs a template for these environment variables in the **/usr/maxplus2/mentor/max2/.cshrc** file.

3. Add the **$MGC_HOME/bin**, **$MAX2_MENTOR/bin**, **$ALT_HOME/bin**, **$EXEMPLAR/bin/**<*os*>, and **$ALT_HOME/bin** directories to the `PATH` environment variable in your **.cshrc** file, where <*os*> is the operating system, e.g., `SUN4` for SunOS; `SUN5` for Solaris.

4. If you plan to use the Altera Schematic Express (**sch_exprss**) utility or the Altera VHDL Express (**vhd_exprss**) utility, add the following environment variable to your **.cshrc** file:

```
setenv MAX2_QSIM /usr/maxplus2/simlib/mentor/max2sim ↵
```

5. Type source ~/.cshrc at a UNIX prompt to source the **.cshrc** file and validate the settings in steps 1 through 4.

6. Add the following lines to your **MGC_location_map** file:

```
$MAX2_MENTOR ↵
/usr/maxplus2/mentor/max2 ↵
$MGC_GENLIB ↵
/<user-specified Mentor Graphics GEN_LIB directory> ↵
$MGC_LSLIB ↵
/<user-specified Mentor Graphics LS_LIB directory> ↵
$MAX2_EXAMPLES ↵
/<user-specified example directory> ↵
$MAX2_LMCLIB ↵
```

```
/<user-specified Logic Modeling directory> ↵
$MAX2_GENLIB ↵
/usr/maxplus2/simlib/mentor/alt_max2 ↵
$MAX2_QSIM ↵
/usr/maxplus2/simlib/mentor/max2sim ↵
$MAX2_FONT ↵
/usr/maxplus2/mentor/max2/fonts ↵
$MGC_SYS1076_STD ↵
/<user-specified MGC_HOME directory>/pkgs/sys_1076_std/ std ↵
$MGC_SYS1076_ARITHMETIC ↵
/<user-specified MGC_HOME directory>/pkgs/sys_1076_std/arithmetic ↵
$MGC_SYS1076_PORTABLE ↵
/<user-specified MGC_HOME directory>/pkgs/sys_1076_std/mgc_portable ↵
$MGC_SYS1076_IEEE ↵
/<user-specified MGC_HOME directory>/pkgs/sys_1076_std/ieee ↵
$MGC_SYS1076_SRC ↵
/<user-specified MGC_HOME directory>/pkgs/sys_1076_std/ src ↵
$MAX2_MFLIB ↵
/usr/maxplus2/simlib/mentor/alt_mf ↵
```

Installing the Alteraprovided MAX+PLUS II/Mentor Graphics/Exemplar Logic interface on your computer automatically installs a template for these environment variables in the **/usr/maxplus2/mentor/max2/location_map/location_map** file.

7. If you want to use QuickHDL software to simulate VHDL or Verilog HDL designs, add the following line in the `[library]` section of your **quickhdl.ini** file: `altera = $MAX2_MFLIB`.

8. If you plan to use QuickHDL software to simulate VITAL-compliant VHDL files, add the following lines to your **MGC_location_map** file:

```
$MAX2_VTLLIB ↵
/usr/maxplus2/simlib/mentor/alt_vtl ↵
```

9. Copy the **/usr/maxplus2/maxplus2.ini** file to your **$HOME** directory:

```
cp /usr/maxplus2/maxplus2.ini $HOME ↵
chmod u+w $HOME/maxplus2.ini ↵
```

The **maxplus2.ini** file contains both Altera- and user-specified initialization parameters that control the MAX+PLUS II software, such as Alteraprovided logic and symbol library paths and the current project name. The MAX+PLUS II installation procedure creates and copies the **maxplus2.ini** file to the **/usr/maxplus2** directory.

Normally, you do not have to edit your local copy of **maxplus2.ini**, because the MAX+PLUS II software updates the file automatically whenever you change any parameters or settings. However, if you move the **max2lib** and **max2inc** library subdirectories, you must update the file. Go to "Creating & Using a Local Copy of the **maxplus2.ini** File" in MAX+PLUS II Help for more information.

Related Links:

Go to the following topics, which are available on the web, for additional information:

- ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
- This manual is also available in 4 parts:
  - Preface & Section 1: MAX+PLUS II Installation
  - Section 2: MAX+PLUS II - A Perspective
  - Section 3: MAX+PLUS II Tutorial
  - Appendices, Glossary & Index

MAX+PLUS II/Mentor Graphics Software Requirements

The following products are used to generate, process, synthesize, and verify a project with the MAX+PLUS ® II software and Mentor Graphics software:

Mentor Graphics		Exemplar	Altera
version C.1:			
System_1076 Compiler QuickSim II Design Architect ENRead ENWrite GEN_LIB library	QuickHDL QuickHDL Pro QuickPath LS_LIB library (optional) DVE	Galileo Extreme version 4.1.1 Leonardo version 4.1.3	MAX+PLUS II version 9.4

☞ The MAX+PLUS II **read.me** file provides up-to-date information on which versions of Mentor Graphics applications are supported by the current version of MAX+PLUS II. It also provides information on installation and operating requirements. You should read the **read.me** file on the CD-ROM before installing the MAX+PLUS II software. After installation, you can open the **read.me** file from the MAX+PLUS II Help menu.

Altera-Provided Logic & Symbol Libraries

The MAX+PLUS ® II/Mentor Graphics environment provides libraries for compiling, synthesizing, and simulating designs.

☞ You can create your own libraries of custom functions for use in Design Architect schematics and VHDL and Verilog HDL design files. You can use custom functions to incorporate an EDIF Input File (**.edf**), Text Design File (**.tdf**), or any other MAX+PLUS II-supported design file into a project. The MAX+PLUS II software uses the Altera® provided **mnt8_bas.lmf** and **exemplar.lmf** Library Mapping Files to map standard Design Architect symbols and VHDL and Verilog HDL functions to equivalent MAX+PLUS II logic functions. To use custom functions, you can create a custom LMF that maps your custom functions to the equivalent EDIF input file, TDF, or other design file. Go to "Library Mapping File" in MAX+PLUS II Help for more information.

Design Architect Libraries

You can enter a Design Architect schematic with logic functions from these Altera-provided symbol libraries: **ALTERA LPMLIB**, **ALTERA GENLIB**, **LSTTL BY TYPE**, and **LSTTL ALL PARTS**. You can access these libraries by choosing **Altera Libraries** (Libraries menu) in the Design Architect software. For information on using library of parameterized modules (LPM) functions, see **ALTERA LPMLIB** Library below.

ALTERA GENLIB Library (Design Architect) & Altera (VHDL) Libraries

The **ALTERA GENLIB** symbol library (called the **Altera** library for VHDL) includes several MAX+PLUS II primitives for controlling design synthesis and fitting. It also includes four macrofunctions (`8count`, `8mcomp`, `8fadd`,

and 81mux) that are optimized for different Altera device families, and the clklock phase-locked loop megafunction, which is supported for some FLEX ® 10K devices.

The following table shows the MAX+PLUS II-specific logic functions.

Table 1. MAX+PLUS II-Specific Logic Functions

Macrofunctions *Note (1)*				**Primitives**	
Name	**Description**	**Name**	**Description**	**Name**	**Description**
8fadd	8-bit full adder	LCELL	Logic cell buffer	EXP	MAX ® 5000, MAX 7000 , and MAX 9000 Expander buffer
8mcomp	8-bit magnitude comparator	GLOBAL	Global input buffer	SOFT	Soft buffer
8count	8-bit up/down counter	CASCADE	FLEX 6000, FLEX 8000, and FLEX 10K cascade buffer	OPNDRN	Open-drain buffer
81mux	8-to-1 multiplexer	CARRY	FLEX 6000, FLEX 8000, and FLEX 10K carry buffer	DFFE DFFE6K *Note (2)*	D-type flipflop with Clock Enable
clklock	Phase-locked loop				

Notes:

1. Logic function names that begin with a number must be preceded by "a_" in VHDL designs. For example, 8fadd must be specified as a_8fadd instead.
2. If you want to use QuickHDL software, make sure you have updated your **quickhdl.ini** file, as described in step 7 of Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment.
3. For designs that are targeted for FLEX 6000 devices, you should use the DFFE primitive only if the design contains either a Clear or Preset signal, but not both. If your design contains both a Clear and a Preset signal, you must use the DFFE6K primitive.

☞ Choose **Old-Style Macrofunctions**, **Primitives**, or **Megafunctions/LPM** from the MAX+PLUS II Help menu for detailed information on these functions.

ALTERA LPMLIB Library

The Alteraprovided **ALTERA LPMLIB** library, which is available for Design Architect schematics and VHDL designs, includes standard functions from the library of parameterized modules (LPM) 2.1.0, except the truth table, finite state machine, and pad functions. The LPM standard defines a set of parameterized modules (i.e., parameterized functions) and their corresponding representations in an EDIF netlist file. These logic functions allow you to create and functionally simulate an LPM-based design without targeting a specific device family. After the design is completed, you can target the design to any device family. The parameters you specify for each LPM function determine which simulation models are generated.

☞ Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu for more information about LPM functions.

Related Links:

- Go to the following topics, which are available on the web, for additional information:
  - FLEX Devices
  - MAX Devices
  - Classic Device Family

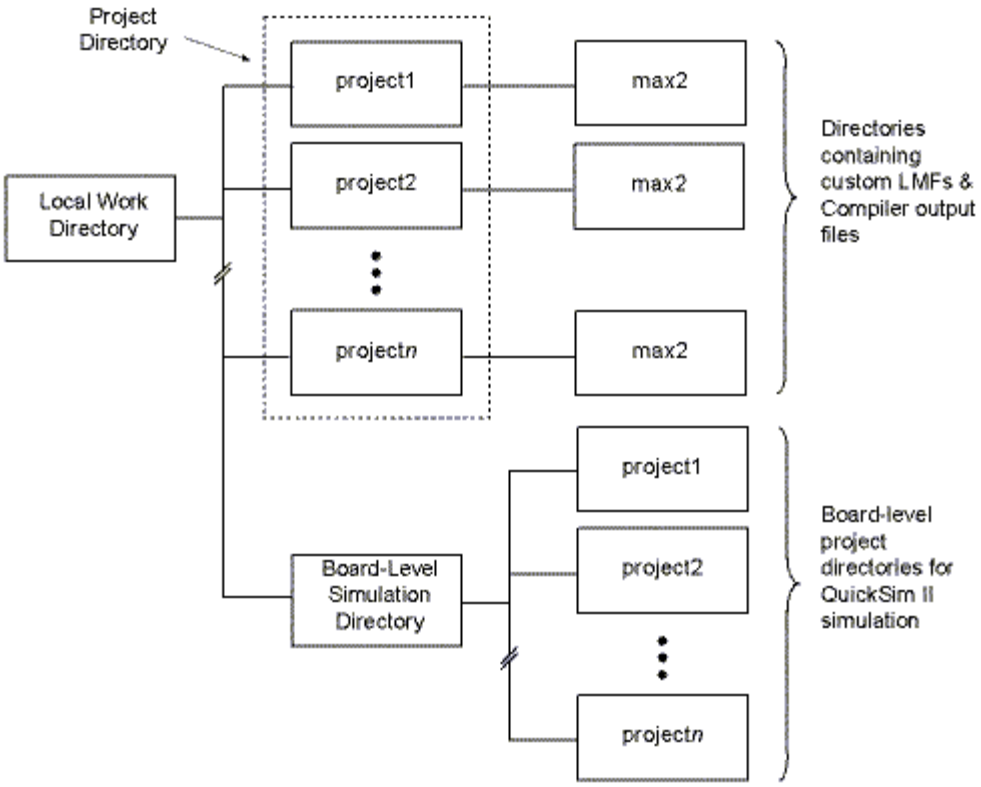
Local Work Area Directory Structure

Design Architect software automatically creates and maintains the project directory structure required for all stages of design entry. Galileo Extreme, Leonardo, and ENWrite software create a **max2** subdirectory, if it does not already exist, under the project directory. These software applications also generate EDIF netlist files, and copy them from the <*project name*> directory to this **max2** subdirectory. All MAX+PLUS ® II Compiler output files are created in the **max2** subdirectory.

Simulation files created with Mentor Graphics applications and Logic Modeling files are located in the board-level simulation subdirectory of the project directory. You can use these files during simulation with QuickSim II software.

The only directory you need to create is the local work directory, which should contain all project directories. Figure 1 shows the recommended file structure.

Figure 1. Recommended File Structure



Related Topics:

- Go to the following MAX+PLUS II ACCESS^{SM} Key topics for related information:
  - MAX+PLUS II Project Directory Structure
  - Mentor Graphics Project Directory Structure

Mentor Graphics Project Directory Structure

Design Architect software generates the following files for each schematic:

- <*drawing name*>/**mgc_component.attr**
- <*drawing name*>/**part.Eddm_part.attr**

- <*drawing name*>**/part.part_1**
- <*drawing name*>**/schematic.mgc_schematic.attr**
- <*drawing name*>**/schematic/schem_id**
- <*drawing name*>**/schematic/sheet1.mgc_sheet.attr**
- <*drawing name*>**/schematic/sheet1.sgfx_1**
- <*drawing name*>**/schematic/sheet1.ssht_1**

The files generated for each schematic are stored in the schematic's <*drawing name*> directory and should not be edited. Mentor Graphics software automatically manages file storage and retrieval operations through this <*drawing name*> directory structure, which does not reflect hierarchical design relationships. Figure 1 shows a sample file structure with **project1** as the UNIX project directory, and **design1**, **subdesign1**, and **subdesign2** as the directories for the top-level design and subdesigns of the project.

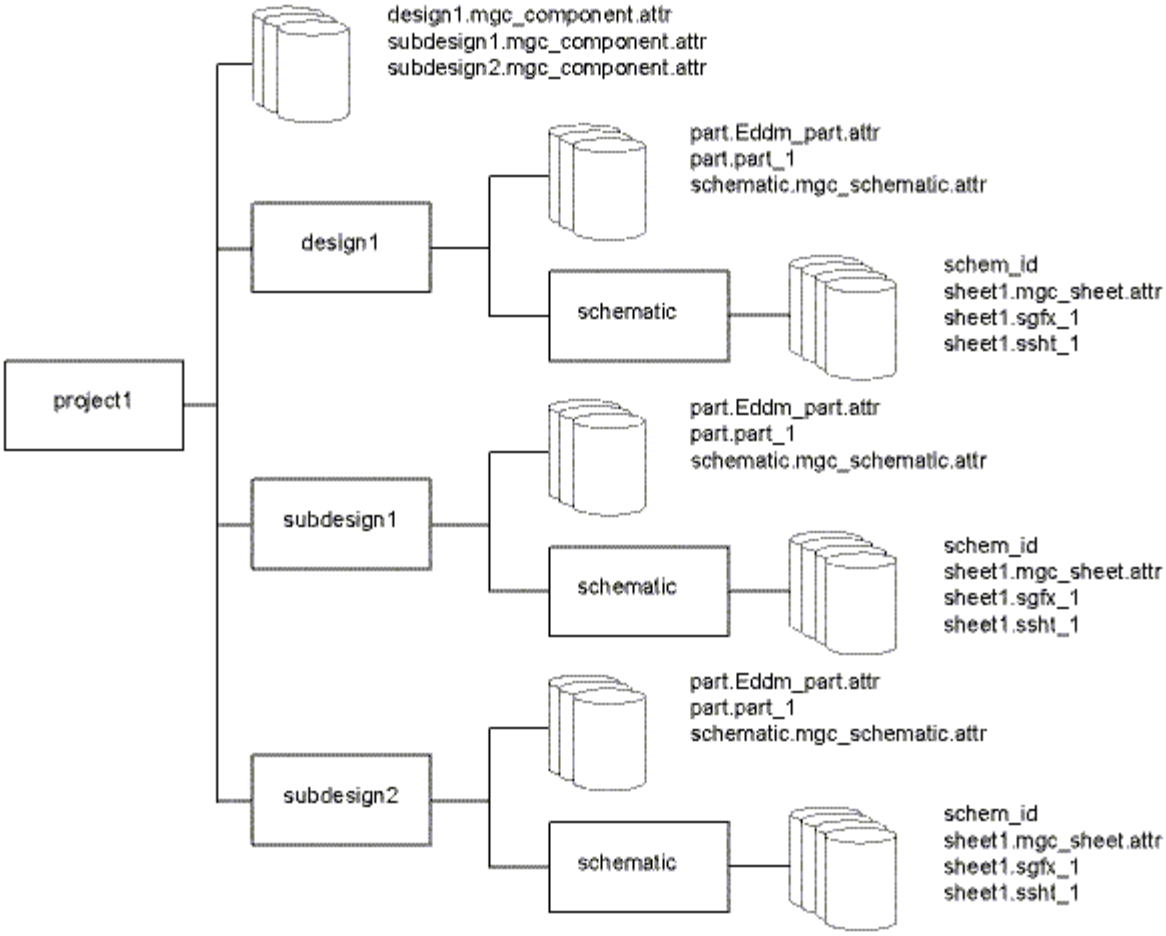


Figure 1. Design Architect Project File Structure

When the ENWrite utility converts the schematic into an EDIF netlist file, it processes the design information and all related file subdirectories, then creates the EDIF netlist file in the directory defined by the user. The EDIF netlist file is named <*project name*>**.edf**, where <*project name*> is the name of the top-level design file. The <*project name*>**.edf** file is automatically moved to the **max2** directory under the project directory.

Related Links:

- Go to the following MAX+PLUS II ACCESS^{SM} Key topics for related information:
  - Local Work Area Directory Structure
  - MAX+PLUS II Project Directory Structure

MAX+PLUS II Project Directory Structure

In the MAX+PLUS ® II software, a project name is the name of a top-level design file, without the filename extension. This design file can be an EDIF, VHDL, or Verilog HDL netlist file; an Altera Hardware Description Language (AHDL) Text Design File (TDF); or any other MAX+PLUS II-supported design file. The EDIF netlist file must be created by ENWrite, Galileo Extreme, or Leonardo software and imported into MAX+PLUS II as an EDIF Input File (**.edf**). Figure 1 shows an example of a MAX+PLUS II project directory.

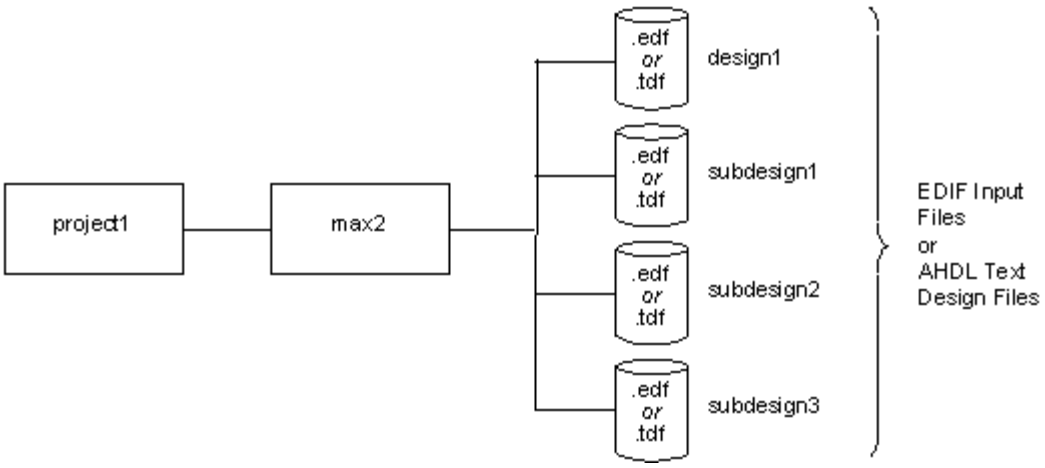


Figure 1. Sample MAX+PLUS II Project Directory

The MAX+PLUS II software stores the connectivity data on the links between design files in a hierarchical project in a Hierarchy Interconnect File (**.hif**), but refers to the entire project only by its project name. The MAX+PLUS II Compiler uses the HIF to build a single, fully flattened project database that integrates all the design files in a project hierarchy.

Related Links:

- Go to the following MAX+PLUS II ACCESS^{SM} Key topics for related information:
  - Local Work Area Directory Structure
  - Mentor Graphics Project Directory Structure

MAX+PLUS II/Mentor Graphics/Exemplar Logic Interface File Organization

The following table shows the MAX+PLUS ® II/Mentor Graphics interface subdirectories that are created in the MAX+PLUS II system directory (by default, the **/usr/maxplus2** directory) during MAX+PLUS II installation.

Related Links:

- For information on the other directories that are created during MAX+PLUS II installation, see "MAX+PLUS II File Organization" in *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual.

Table 1. MAX+PLUS II Directory Organization

Directory	Description
.lmf	Contains the Altera-provided Library Mapping Files, **mnt8_bas.lmf** and **exemplar.lmf**, that map Mentor Graphics and Exemplar Logic logic functions to equivalent MAX+PLUS II logic functions.
./mentor	Contains the AMPLE userware for the MAX+PLUS II/Mentor Graphics interface.
	Contains MAX+PLUS II primitives such as `CARRY`, `CASCADE`, `EXP`, `GLOBAL`, `LCELL`, `SOFT`,

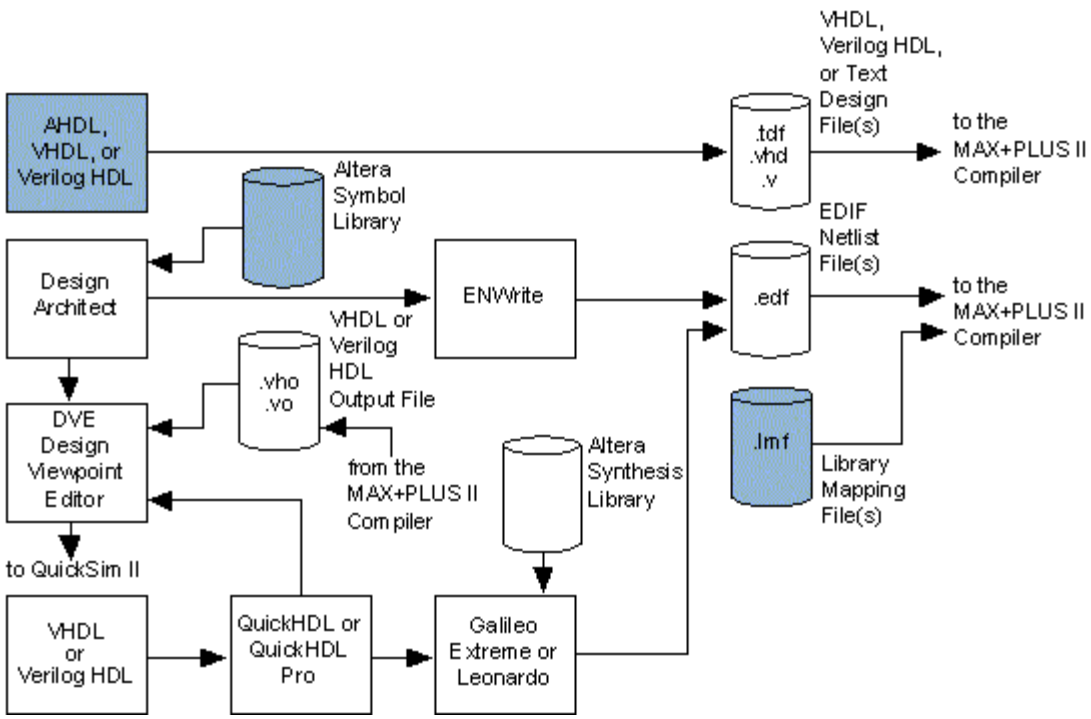
./simlib/mentor/alt_max2	OPNDRN, DFFE, and DFFE6K (D flipflop with Clock Enable) for use in Design Architect schematics.
./simlib/mentor/max2sim	Contains the MAX+PLUS II/Mentor Graphics simulation model library, **max2sim**, for use with QuickSim II and QuickPath software.
./simlib/mentor/synlib	Contains the MAX+PLUS II synthesis library for use with AutoLogic II software, which supports synthesis for users running Mentor Graphics version B1.
./simlib/mentor/alt_mf	Contains the MAX+PLUS II macrofunction and megafunction libraries.
./simlib/mentor/alt_vtl	Contains the MAX+PLUS II VITAL library.

Mentor Graphics/Exemplar Logic Design Entry Flow

The following figure shows the design entry flow for the MAX+PLUS® II/Mentor Graphics/Exemplar Logic interface.

Figure 1. MAX+PLUS II/Mentor Graphics/Exemplar Logic Design Entry Flow

Alteraprovided items are shown in blue.



Creating VHDL & Verilog HDL Designs for Use with MAX+PLUS II Software

You can create VHDL and Verilog HDL design files with the MAX+PLUS ® II Text Editor or another standard text editor and save them in the appropriate directory for your project.

The MAX+PLUS II Text Editor offers the following advantages:

- Templates are available with the **VHDL Templates** and **Verilog Templates** commands (Template menu).

These templates are also available in the ASCII **vhdl.tmp** and **verilog.tmp** files, respectively, which are located in the **/usr/maxplus2** directory.

- If you use the MAX+PLUS II Text Editor to create your VHDL design, you can turn on the **Syntax Coloring** command (Options menu). The Syntax Coloring feature displays keywords and other elements of text in text files in different colors to distinguish them from other forms of syntax.

To create a VHDL or Verilog HDL design file for use with the MAX+PLUS II software, go through the following steps:

1. Enter a VHDL or Verilog HDL design in the MAX+PLUS II Text Editor or another standard text editor and save it in your working directory.

2. Enter primitives, macrofunctions, and megafunctions in your VHDL or Verilog HDL design from the **Altera** library.

   The following topics describe special steps needed to instantiate LPM and `clklock` functions:

   - Instantiating LPM Functions in VHDL
   - Instantiating the clklock Megafunction in VHDL or Verilog HDL

   You can instantiate MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.

3.

   (Optional) Use the QuickHDL software to functionally simulate the design file, as described in Performing a Functional Simulation with QuickHDL Software and Performing a Functional Simulation with QuickHDL Pro Software.

4. Once you have created a VHDL or Verilog HDL design, you can generate an EDIF netlist file that can be imported into the MAX+PLUS II software with either of the following methods:

   - You can synthesize and optimize your design and create an EDIF netlist file, as described in Synthesizing & Optimizing VHDL & Verilog HDL Projects with Galileo Extreme Software or Synthesizing & Optimizing VHDL & Verilog HDL Projects with Leonardo Software.

   - You can use the Altera VHDL Express utility, **vhd_exprss**, to automatically create an EDIF netlist file, compile it with the MAX+PLUS II Compiler, generate an EDIF Output File (**.edo**), and prepare the EDIF Output File for simulation with QuickHDL software, as described in Using the Altera Schematic Express (**vhd_exprss**) Utility.

   Installing the Alteraprovided MAX+PLUS II/Mentor Graphics/Exemplar Logic interface on your computer automatically creates the following sample VHDL design files:

- **/usr/maxplus2/examples/mentor/example5/count4.vhd**
- **/usr/maxplus2/examples/mentor/example6/count8.vhd**
- **/usr/maxplus2/examples/mentor/example8/adder16.vhd**

Related Links:

- Go to Compiling Projects with MAX+PLUS II Software in these MAX+PLUS II ACCESS^{SM} Key topics for related information.

Synthesizing & Optimizing VHDL & Verilog HDL Projects with Leonardo Software

After you have created a VHDL or Verilog HDL design, you can use Exemplar Logic's Leonardo software to synthesize and optimize your VHDL Design File (**.vhd**) or Verilog Design File (**.v**) and prepare it for compilation with the MAX+PLUS ® II Compiler.

To synthesize and optimize your project and generate an EDIF netlist file with Leonardo software, go through the following steps:

1. Be sure to set up the working environment correctly, as described in Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment.
2. Create a VHDL or Verilog HDL design that follows the guidelines described in Creating VHDL & Verilog HDL Designs for Use with MAX+PLUS II Software.
3. (Optional) Use the QuickHDL software to functionally simulate the design file, as described in Performing a Functional Simulation with QuickHDL Software.
4. Select the icon for your project's design file from the Navigator window, press Button 3, and choose **max2_leonardo** to start the Leonardo software and open the Exemplar Logic Leonardo window. You can also start Leonardo by typing `max2_leonardo` ↵ at the UNIX prompt.
5. Click Button 1 on the **Flow Guide** toolbar button to open the **Customize Flow Guide** dialog box.
6. Turn on the *Altera EDIF Output File* checkbox under *Output Flow*.
7. Choose **Run Flow Guide** to open the Flow Guide window and specify the appropriate options in the following modules to synthesize your project:
    1. Choose **Load Library** to open the **Load Library** dialog box. If necessary, select *FPGA Enhanced* from the *Tech Type* drop-down list box. Select the target Altera® device family from the list of supported device families and choose **Load** to close the dialog box.
    2. Choose **Read** to open the **Read** dialog box. Turn on *VHDL* or *Verilog HDL* under *Format*, ensure that the appropriate library name appears under *Work Library*, and type the name of your design file in the *Filename* box or select it from the **Select a File** dialog box. Choose **Read** to close the dialog box.
    3. Choose **Pre-Optimize** to open the **Pre-Optimize** dialog box. Choose **Pre-Optimize** to accept the default pre-optimization settings and close the dialog box.
    4. Choose **Optimize** to open the **Optimize** dialog box. Choose **Optimize** to accept the default optimization settings and close the dialog box.
    5. Choose **Write Altera** to open the **Convenience Procedures** dialog box. Type `write_altera` in the *Procedure* box or select *write_altera* from the list box and choose **Run** to automatically generate *<design name>***.edf**.
    6. Choose **Exit Flow Guide** to return to the Leonardo window.
8. Process your design with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Installing the Alteraprovided MAX+PLUS II/Mentor Graphics/Exemplar Logic interface on your computer automatically creates the following sample VHDL Design Files:

- **/usr/maxplus2/examples/mentor/example5/count4.vhd**
- **/usr/maxplus2/examples/mentor/example6/count8.vhd**

- **/usr/maxplus2/examples/mentor/example8/adder16.vhd**

Related Links:

- Go to Synthesizing & Optimizing VHDL & Verilog HDL Projects with Galileo Extreme Software in these MAX+PLUS II ACCESS^{SM} Key topics for related information.

Compiling Projects with MAX+PLUS II Software

The MAX+PLUS® II Compiler can process design files in a variety of formats. This topic describes how to use MAX+PLUS II software to compile projects in which the top-level design file is an EDIF Input File (with the extension **.edf**).

Related Links:

- Refer to the following sources for additional information:
    - Go to MAX+PLUS II Help for information on compiling VHDL and Verilog HDL, design files directly with the MAX+PLUS II Compiler.
    - Go to Running Synopsys Compilers from MAX+PLUS II Software for information on running the Synopsys Design Compiler or FPGA Compiler software on a VHDL or Verilog HDL design from within the MAX+PLUS II Compiler window.

To compile a design (also called a "project") with MAX+PLUS II software, go through the following steps:

1. Create design files that are compatible with the MAX+PLUS II software and convert them into EDIF Input Files with the extension **.edf**. Specific instructions for some tools are described in these MAX+PLUS II ACCESS^{SM} Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your design entry or synthesis and optimization tool.

2. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, create a mapping for each function in a Library Mapping File (**.lmf**) to map the custom symbol to the corresponding EDIF Input File, AHDL Text Design File (**.tdf**), or other MAX+PLUS II-supported design file. These custom functions are represented in design files as hollow-body symbols or "black box" HDL descriptions.

   ☞ Go to "Library Mapping Files (**.lmf**)" in MAX+PLUS II Help for more information.

3. Open MAX+PLUS II and specify the name of your top-level design file as the project name with the **Project Name** command (File menu). If you open an HDL file in the MAX+PLUS II Text Editor, you can choose the **Project Set Project to Current File** command (File menu) instead.

   ☞ You can also compile a project from a command line. However, the first time you compile a project, the settings you need to specify are easier to specify from within the MAX+PLUS II software. After you have run the graphical user interface for the MAX+PLUS II software at least once, you can more easily use the command-line **setacf** utility to modify options in the Assignment & Configuration File (**.acf**) for the project. Type `setacf -h` ↵ and `maxplus2 -h` ↵ for descriptions of **setacf** and MAX+PLUS II command-line syntax.

4. Choose **Device** (Assign menu) and select the target Altera device family in the *Device Family* drop-down list box. If you wish to implement the design logic in a specific device, select it in the *Devices* box. Otherwise, select *AUTO* to allow the MAX+PLUS II Compiler to choose the best device(s) in the current device family.

If your design entry or synthesis and optimization tool required you to specify a target family and/or device, specify the same information in this dialog box. For information on partitioning logic among multiple devices, go to MAX+PLUS II Help. Choose **OK**.

5. Open the Compiler window by choosing the **Compiler** command (MAX+PLUS II menu). Go through the following steps to specify the options necessary to compile the design file(s) in your project:

    1. Ensure that all EDIF netlist files have the extension **.edf** and choose **EDIF Netlist Reader Settings** (Interfaces menu).

    2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor. This name should be the name of the vendor whose tool(s) you used to create the EDIF netlist files. If your vendor name does not appear, select *Custom* instead.

        ☞ If you are compiling a design created with Synopsys FPGA Express software, select *Synopsys*, choose the **Customize** button, enter <*project name*>**.lmf** in the *LMF #1* box, choose **OK**, and skip to step 6.

    3. If you selected an existing vendor name in the *Vendor* box and your project contains design files that require custom LMF mappings, choose the **Customize** button to expand the dialog box to show all settings. Turn on the *LMF #2* checkbox and type your custom LMF's filename in the corresponding text box, or select a name from the *Files* box. The selection in the *Vendor* box will change to *Custom* and all settings will be retained until you change them again.

    4. If you selected *Custom* in the *Vendor* box, choose the **Customize** button to expand the dialog box to show all settings. Any previously defined custom settings will be displayed. Under *Signal Names*, type one or more names with up to 20 total name characters in the *VCC* or *GND* box if your EDIF Input File(s) use one or more names other than `VCC` or `GND` for the global high or low signals. Multiple signal names must be separated by either a comma (,) or a space. Under *Library Mapping Files*, turn on the *LMF #1* checkbox and type a filename in the text box following it, or select a name from the *Files* box. If necessary, specify another LMF name in the *LMF #2* box. Go to MAX+PLUS II Help for detailed information on the settings available in the **EDIF Netlist Reader Settings** dialog box.

    5. Choose **OK**.

6. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, you may need to ensure that all files are present in your project directory, i.e., the same directory as the top-level design file. Otherwise, you must specify the directories containing these files as user libraries with the **User Libraries** command (Options menu).

7. Follow all guidelines that apply to your design entry or synthesis and optimization tool:

    - Exemplar Logic Galileo Extreme-Specific Compiler Settings
    - Synopsys DesignWare-Specific Compiler Settings
    - Converting Synopsys FPGA Compiler & Design Compiler Timing Constraints into MAX+PLUS II-Compatible Format with the **syn2acf** Utility
    - Synplicity Synplify-Specific Compiler Settings

8. If you wish to generate EDIF, VHDL, or Verilog HDL output files for post-compilation simulation or timing analysis with another EDA tool, go through the following steps:

    1. (Optional) Turn on the **Optimize Timing SNF** command (Processing menu) to reduce the size of the output file(s). Turning on this command can reduce the size of output netlists by up to 30%.

        ☞ This command does not create optimized timing SNFs on UNIX workstations. However, a non-optimized timing SNF provides the same functional and timing information as an optimized timing SNF.

2. If you wish to generate EDIF Output Files (**.edo**), go through these steps:

    1. Turn on the **EDIF Netlist Writer** command (Interfaces menu). Then choose the **EDIF Netlist Writer Settings** command (Interfaces menu).

    2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor and choose **OK**. If your vendor name does not appear, select *Custom* instead and specify the settings that are appropriate for your simulation or timing analysis tool. Go to MAX+PLUS II Help for detailed information on the options available in the **EDIF Netlist Writer Settings** dialog box.

    3. To generate an optional Standard Delay Format (SDF) Output File (**.sdo**), choose the **Customize** button to expand the dialog box to show all settings. Select one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**.

    The filenames of the EDIF Output File(s) and optional SDF Output File(s) are the same as the user-defined chip name(s) for the project; if no chip names exist, the Compiler assigns filenames that are based on the project name. For a multi-device project, the Compiler also generates a top-level EDIF Output File that is uniquely identified by "**_t**" appended to the project name. In addition, the Compiler automatically generates a VHDL Memory Model Output File, *<project name>***.vmo**, when it generates an EDIF Output File that contains memory (RAM or ROM).

3. If you wish to generate VHDL Output Files (**.vho**), turn on the **VHDL Netlist Writer** command (Interfaces menu). Then choose **VHDL Netlist Writer Settings** command (Interfaces menu). Select *VHDL Output File (.vho)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF ver. 2.1 files contain timing delay information that allows you to perform back-annotation simulation in VHDL with VITAL-compliant simulation libraries. The VHDL Output Files generated by the Compiler have the extension **.vho**, but are otherwise named in the same way as the EDIF Output Files described above.

4. If you wish to generate Verilog HDL Output Files (**.vo**), turn on the **Verilog Netlist Writer** command (Interfaces menu). Then choose **Verilog Netlist Writer Settings** command (Interfaces menu). Select *Verilog Output File (.vo)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF Output Files contain timing delay information that allows you to perform back-annotation simulation in Verilog HDL. The Verilog Output Files generated by the Compiler have the extension **.vo**, but are otherwise named in the same way as the EDIF Output Files described above.

9. To run the MAX+PLUS II Compiler, choose the **Project Save & Compile** command (File menu) or choose the **Start** button in the Compiler window.

    See step 3 for information on running MAX+PLUS II software from the command line.

10. Once you have compiled the project with the MAX+PLUS II Compiler, you can use the VHDL, Verilog HDL, or EDIF output file(s), and the optional SDF Output File(s) (**.sdo**) to perform timing analysis or timing simulation with another EDA tool. Specific instructions for some tools are described in these MAX+PLUS II ACCESS Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your EDA tool.

The MAX+PLUS II Compiler also generates a Report File (**.rpt**), a Pin-Out File (**.pin**), and one or more of the following files for device programming or configuration:

- JEDEC Files (**.jed**)
- Programmer Object Files (**.pof**)
- SRAM Object Files (**.sof**)
- Hexadecimal (Intel-format) Files (**.hex**)
- Tabular Text Files (**.ttf**)

Related Links:

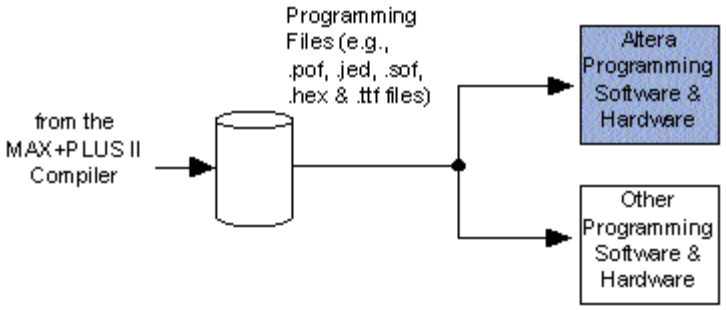
- Refer to the following sources for additional information:
  - Go to *Compiler Procedures* in MAX+PLUS II Help for information on other available Compiler settings.
  - Go to *Programmer Procedures* in MAX+PLUS II Help for instructions on creating other types of programming files and on programming or configuring Altera devices.
  - Go to Back-Annotating MAX+PLUS II Pin Assignments to Design Architect Symbols for information on back-annotating pin assignments in Mentor Graphics Design Architect schematics.
  - Go to Programming Altera Devices for information on the different programming hardware options for Altera device families.
- Go to the following topics, which are available on the web, for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware

Programming Altera Devices

Once you have successfully compiled and simulated a project with the MAX+PLUS® II software, you can program an Altera® device and test it in the target circuit. Figure 1 shows the device programming flow for MAX+PLUS II software.

Figure 1. MAX+PLUS II Device Programming Flow

Altera-provided items are shown in blue.



You can program devices with Altera programming hardware and MAX+PLUS II Programmer software installed on a 486- or Pentium-based PC or a UNIX workstation, or with programming hardware and software available from other manufacturers. Table 1 shows the available Altera programming hardware options on PCs and UNIX workstations.

Table 1. Altera Programming Hardware

Programming Hardware	UNIX PCs Work-	MAX® 3000A	Classic® & MAX	MAX 7000 &	MAX 7000A, MAX 7000AE, MAX 7000B, MAX	FLEX® 6000, FLEX 6000A, FLEX 8000, FLEX 10K, FLEX 10KA,	In-System Programming/

Option	stations	Devices	5000 Devices	MAX 7000E Devices	7000S MAX 9000 & MAX 9000A Devices	FLEX 10KB, & FLEX 10KE Devices	Configuration
Logic Programmer card, PL-MPU Master Programming Unit, and device-specific adapters	✓	✓	✓	✓	✓		
BitBlaster™ Download Cable	✓	✓	✓		✓	✓	✓
ByteBlasterMV™ Download Cable	✓		✓		✓	✓	✓
MasterBlaster™ Download Cable	✓	✓	✓		✓	✓	✓

If you wish to transfer programming files from a UNIX workstation to a PC over a network with File Transfer Protocol (FTP) or other similar transfer programs, be sure to select binary transfer mode.

Programming hardware from other manufacturers varies, but typically consists of a device connected to one of the serial ports on the workstation. Various vendors, such as Data I/O and BP Microsystems, supply hardware and software for programming Altera devices.

Related Links:

- Go to Compiling Projects with MAX+PLUS II Software for information on creating programming files.
- Go to the following topics, which are available on the web, for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware
  - FLEX Devices
  - MAX Devices
  - Classic Device Family

Synthesizing & Optimizing VHDL & Verilog HDL Projects with Leonardo Software

After you have created a VHDL or Verilog HDL design, you can use Exemplar Logic's Leonardo software to synthesize and optimize your VHDL Design File (**.vhd**) or Verilog Design File (**.v**) and prepare it for compilation with the MAX+PLUS ® II Compiler.

To synthesize and optimize your project and generate an EDIF netlist file with Leonardo software, go through the following steps:

1. Be sure to set up the working environment correctly, as described in Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment.
2. Create a VHDL or Verilog HDL design that follows the guidelines described in Creating VHDL & Verilog HDL Designs for Use with MAX+PLUS II Software.
3. (Optional) Use the QuickHDL software to functionally simulate the design file, as described in Performing a Functional Simulation with QuickHDL Software.
4. Select the icon for your project's design file from the Navigator window, press Button 3, and choose **max2_leonardo** to start the Leonardo software and open the Exemplar Logic Leonardo window. You can also start Leonardo by typing `max2_leonardo` ↵ at the UNIX prompt.
5. Click Button 1 on the **Flow Guide** toolbar button to open the **Customize Flow Guide** dialog box.
6. Turn on the *Altera EDIF Output File* checkbox under *Output Flow*.
7. Choose **Run Flow Guide** to open the Flow Guide window and specify the appropriate options in the following modules to synthesize your project:
    1. Choose **Load Library** to open the **Load Library** dialog box. If necessary, select *FPGA Enhanced* from the *Tech Type* drop-down list box. Select the target Altera® device family from the list of supported device families and choose **Load** to close the dialog box.
    2. Choose **Read** to open the **Read** dialog box. Turn on *VHDL* or *Verilog HDL* under *Format*, ensure that the appropriate library name appears under *Work Library*, and type the name of your design file in the *Filename* box or select it from the **Select a File** dialog box. Choose **Read** to close the dialog box.
    3. Choose **Pre-Optimize** to open the **Pre-Optimize** dialog box. Choose **Pre-Optimize** to accept the default pre-optimization settings and close the dialog box.
    4. Choose **Optimize** to open the **Optimize** dialog box. Choose **Optimize** to accept the default optimization settings and close the dialog box.
    5. Choose **Write Altera** to open the **Convenience Procedures** dialog box. Type `write_altera` in the *Procedure* box or select *write_altera* from the list box and choose **Run** to automatically generate *<design name>***.edf**.
    6. Choose **Exit Flow Guide** to return to the Leonardo window.
8. Process your design with the MAX+PLUS II Compiler, as described in Compiling Projects with

MAX+PLUS II Software.

Installing the Alteraprovided MAX+PLUS II/Mentor Graphics/Exemplar Logic interface on your computer automatically creates the following sample VHDL Design Files:

- **/usr/maxplus2/examples/mentor/example5/count4.vhd**
- **/usr/maxplus2/examples/mentor/example6/count8.vhd**
- **/usr/maxplus2/examples/mentor/example8/adder16.vhd**

Related Links:

- Go to Synthesizing & Optimizing VHDL & Verilog HDL Projects with Galileo Extreme Software in these MAX+PLUS II ACCESS^{SM} Key topics for related information.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Altera-Provided Logic & Symbol Libraries

The MAX+PLUS ® II/Mentor Graphics environment provides libraries for compiling, synthesizing, and simulating designs.

☞ You can create your own libraries of custom functions for use in Design Architect schematics and VHDL and Verilog HDL design files. You can use custom functions to incorporate an EDIF Input File (**.edf**), Text Design File (**.tdf**), or any other MAX+PLUS II-supported design file into a project. The MAX+PLUS II software uses the Altera® provided **mnt8_bas.lmf** and **exemplar.lmf** Library Mapping Files to map standard Design Architect symbols and VHDL and Verilog HDL functions to equivalent MAX+PLUS II logic functions. To use custom functions, you can create a custom LMF that maps your custom functions to the equivalent EDIF input file, TDF, or other design file. Go to "Library Mapping File" in MAX+PLUS II Help for more information.

Design Architect Libraries

You can enter a Design Architect schematic with logic functions from these Altera-provided symbol libraries: **ALTERA LPMLIB**, **ALTERA GENLIB**, **LSTTL BY TYPE**, and **LSTTL ALL PARTS**. You can access these libraries by choosing **Altera Libraries** (Libraries menu) in the Design Architect software. For information on using library of parameterized modules (LPM) functions, see **ALTERA LPMLIB** Library below.

ALTERA GENLIB Library (Design Architect) & Altera (VHDL) Libraries

The **ALTERA GENLIB** symbol library (called the **Altera** library for VHDL) includes several MAX+PLUS II primitives for controlling design synthesis and fitting. It also includes four macrofunctions (`8count`, `8mcomp`, `8fadd`, and `81mux`) that are optimized for different Altera device families, and the `clklock` phase-locked loop megafunction, which is supported for some FLEX ® 10K devices.

The following table shows the MAX+PLUS II-specific logic functions.

Table 1. MAX+PLUS II-Specific Logic Functions

Macrofunctions *Note (1)*				**Primitives**	
Name	**Description**	**Name**	**Description**	**Name**	**Description**
`8fadd`	8-bit full adder	`LCELL`	Logic cell buffer	`EXP`	MAX ® 5000, MAX 7000 , and MAX 9000 Expander buffer
`8mcomp`	8-bit magnitude comparator	`GLOBAL`	Global input buffer	`SOFT`	Soft buffer
`8count`	8-bit up/down counter	`CASCADE`	FLEX 6000, FLEX 8000, and FLEX 10K cascade buffer	`OPNDRN`	Open-drain buffer
`81mux`	8-to-1 multiplexer	`CARRY`	FLEX 6000, FLEX 8000, and FLEX 10K carry buffer	`DFFE` `DFFE6K` *Note (2)*	D-type flipflop with Clock Enable
`clklock`	Phase-locked loop				

Notes:

1. Logic function names that begin with a number must be preceded by "`a_`" in VHDL designs. For example,

`8fadd` must be specified as `a_8fadd` instead.

2. If you want to use QuickHDL software, make sure you have updated your **quickhdl.ini** file, as described in step 7 of Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment.
3. For designs that are targeted for FLEX 6000 devices, you should use the `DFFE` primitive only if the design contains either a Clear or Preset signal, but not both. If your design contains both a Clear and a Preset signal, you must use the `DFFE6K` primitive.

☞ Choose **Old-Style Macrofunctions**, **Primitives**, or **Megafunctions/LPM** from the MAX+PLUS II Help menu for detailed information on these functions.

ALTERA LPMLIB Library

ALTERA LPMLIB Library

The Alteraprovided **ALTERA LPMLIB** library, which is available for Design Architect schematics and VHDL designs, includes standard functions from the library of parameterized modules (LPM) 2.1.0, except the truth table, finite state machine, and pad functions. The LPM standard defines a set of parameterized modules (i.e., parameterized functions) and their corresponding representations in an EDIF netlist file. These logic functions allow you to create and functionally simulate an LPM-based design without targeting a specific device family. After the design is completed, you can target the design to any device family. The parameters you specify for each LPM function determine which simulation models are generated.

☞ Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu for more information about LPM functions.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Assigning the Implement in EAB Logic Option

Logic option and logic synthesis style assignments allow you to guide logic synthesis with logic optimization features that are specific to Altera® devices. You can assign logic options and styles to individual logic functions in your design. The MAX+PLUS® II Compiler also uses a device-family-specific default logic synthesis style for each project.

Related Topics:

- Go to "Resource Assignments in EDIF Input Files" and "Assigning Resources in a Third-Party Design Editor" in MAX+PLUS II Help for complete and up-to-date information on logic option and logic synthesis style assignments, including definitions and syntax of these assignments.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

You can assign the Implement in EAB logic option to individual logic functions in a FLEX® 10K design. This option directs the MAX+PLUS® II Compiler's Logic Synthesizer module to implement the function in an embedded array block (EAB) rather than in logic cell(s). You can specify the Implement in EAB Logic Option in VHDL or Verilog HDL designs, or in a Synplify Design Constraints File (**.sdc**). If you add timing constraints or resource assignments in a separate Synplify Design Constraints File (**.sdc**), you must add the Synplify Design Constraints File (**.sdc**) to the project by adding it to the *Source Files* list in the Synplify window.

☞ If your design uses resource assignment attributes that you wish to pass to the MAX+PLUS II software, you should save your file in EDIF netlist file format. See Entering Resource Assignments for more information.

VHDL Syntax

Use the following syntax to assign the Implement in EAB logic option in VHDL:

```
attribute altera_implement_in_eab : boolean;
attribute altera_implement_in_eab of <port name>: label is true;
```

Example:

```
attribute altera_implement_in_eab of U1: label is true;
    begin
        U1: mymux port map (in1 => a, sel => s, dout => o);
```

Verilog HDL Syntax

Use the following syntax to assign the Implement in EAB logic option in Verilog HDL:

<*module or architecture name*> `/* synthesis altera_implement_in_eab=1 */;`

Example:

```
sqrtb sq (.z(sqa), .a(a))  /* synthesis altera_implement_in_eab=1 */;
defparam sq.asize = 8;
```

Synplify Design Constraints File Syntax

Use the following syntax to assign the Implement in EAB logic option in a Synplify Design Constraints File (**.sdc**):

`define_attribute {`<*module or architecture name*>`} altera_implement_in_eab 1`

Example:

```
define_attribute {inst1.sqrt8} altera_implement_in_eab 1
```

Related Links:

- Refer to the following sources for more information:
  - Go to Entering Resource Assignments in these MAX+PLUS II ACCESS^{SM} Key topics for information on entering other types of assignments.
  - Go to "Resource Assignments in EDIF Input Files" and "Assigning Resources in a Third-Party Design Editor" in MAX+PLUS II Help for complete and up-to-date information on other logic options and logic synthesis style assignments, including definitions and syntax of these assignments.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Assigning Logic Options

Logic options and logic synthesis style assignments allow you to guide logic synthesis with logic optimization features that are specific to Altera® devices. You can assign logic options and styles to individual logic functions in your design. The MAX+PLUS® II Compiler also uses a device family-specific default logic synthesis style for each project.

To make pin, logic cell, and chip assignments, use the `set_attribute` command at a **dc_shell** prompt. Before using the `set_attribute` command, add the following line to your **.synopsys_dc.setup** file:

```
edifout_write_properties_list = {LOGIC_OPTION, CLIQUE, CHIP_PIN_LC} ↵
```

✔ To assign a logic option or a logic synthesis style, type the following command at a **dc_shell** prompt:

```
set_attribute find(<design object>, (<instance name>)) "LOGIC_OPTION"
-type string "<logic option>=<value>" ↵
```

For example:

```
set_attribute find (cell, (U1)) "LOGIC_OPTION" -type string
"STYLE=FAST" ↵
```

To specify multiple logic options, use commas as separators.

For example:

```
set_attribute find (cell, (U1))"LOGIC_OPTION" -type string "STYLE=FAST,
CARRY_CHAIN=MANUAL" ↵
```

Related Topics:

- Go to "Resource Assignments in EDIF Input Files" and "Assigning Resources in a Third-Party Design Editor" in MAX+PLUS II Help for complete and up-to-date information on logic option and logic synthesis style assignments, including definitions and syntax of these assignments.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Instantiating LPM & Other Parameterized Functions in Concept Schematics

You can use library of parameterized modules (LPM) functions and other Altera® -provided parameterized functions in Concept schematics if you also use the HDL Direct utility.

To instantiate LPM functions, go through the following steps:

1. Choose the **Add Part** button from the toolbar or type `add` from the Concept window to open the Component Browser window.
2. Choose **alt_lpm** (Library menu). All functions in the **alt_lpm** library are MAX+PLUS® II-compatible. Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu to get detailed information on all supported parameterized functions.
3. Type `attribute`, then click on each component to set parameters for each function. See General Guidelines below for additional information.
4. Add `inport` and `outport` symbols from the **hdl_direct_lib** library to the interface signals. Use the `supply_0` and `supply_1` symbols from the **hdl_direct_lib** library to connect a net to `GND` or `VCC`.
5. Continue with the steps necessary to complete your Concept schematic, as described in Creating Concept Schematics for Use with MAX+PLUS II Software.

Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample Concept schematic file, which includes LPM function instantiation:

- **/usr/maxplus2/examples/cadence/example12/fifo**

General Guidelines

- If a pin is not used, leave it floating. The **concept2alt** utility removes all unconnected pins when it generates an EDIF netlist file.
- For the `csfifo` function, the value of the `LPM_NUMWORDS` parameter must be between $2^{LPM_WIDTHAD-1}$ and $2^{LPM_WIDTHAD}$.
- Make sure that any hexadecimal (Intel-format) file (**.hex**) that you use to specify the initial content of a memory does not have the same name as the design file name.
- Make sure that all properties and value strings are in uppercase letters, except the filename specified with the `LPM_FILE` property, which should use the actual case of the filename.
- Choose the **Set** button in the Concept window and choose *CAPS_LOCK_OFF* for the *CAPS LOCK* option.
- Only the `LPM_POLARITY` parameter (which can be set to `INVERT` or `NORMAL`) can determine the polarity of the bus or pin. You can display a bubble in the Concept schematic to indicate an inverted pin by typing `BUBBLE` in the Concept command window and selecting the appropriate pin. However, the bubble does not determine the polarity of the pin or bus.
- Avoid using the **Replace** button in the Concept window to replace old symbols with new ones: you may accidentally set unwanted properties. Instead, you should use the **Delete** button to delete old symbols and the **Add** button to add new symbol(s).

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Instantiating LPM Functions in Design Architect Schematics

Design Architect software allows you to instantiate functions included in the library of parameterized modules (LPM) from the **ALTERA LPMLIB** library.

Go through the following steps to instantiate LPM functions in a Design Architect schematic:

1. While you are entering your Design Architect schematic, choose **Altera Libraries** (Library menu).
2. Choose **ALTERA LPMLIB** (Altera Libraries menu).
3. Choose from the available LPM functions on the ALTERA GENLIB menu.
4. In the **LPM_**<*function name*> dialog box, specify appropriate values for the variables displayed for the LPM function you chose in step 3. Make sure that any hexadecimal (Intel-format) file that you use to specify the initial content of a memory function does not have the same name as the design file name. Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu for detailed information on LPM functions.
5. Choose **OK** to generate a symbol for the LPM function you chose in step 3 and a corresponding VHDL simulation model.
6. Continue with the steps necessary to complete your Design Architect schematic, as described in Creating Design Architect Schematics for Use with MAX+PLUS II Software.
7. When you save the schematic, the Design Architect software will ask whether you want to compile the LPM model. Choose **YES** if you want to compile the VHDL code for the LPM functions. The software will automatically select the corresponding compiler: System 1076 for B.(*x*) releases and QuickHDL compilers for releases C.1 and later.

Installing the Altera® provided MAX+PLUS II/Mentor Graphics/Exemplar Logic interface on your computer automatically creates the sample Design Architect schematic file **/usr/maxplus2/examples/mentor/example7/fifo**, which includes LPM instantiation.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Using FPGA Compiler N-Input LUT Optimization for FLEX 6000, FLEX 8000 & FLEX 10K Devices

The Synopsys FPGA Compiler software supports an *N*-input look-up table (LUT) function that improves the quality of the results and the predictability of delay and resource estimates. All Altera® FPGA Compiler libraries for FLEX® 6000, FLEX 8000, and FLEX 10K devices support the *N*-input LUT function.

Figure 1 shows a sample command sequence that FPGA Compiler might require for N-input LUT optimization. To use N-input LUT optimization, include the edifout_write_properties_list = "lut_function" command.

Figure 1. Sample Command Sequence for N-Input LUT Optimization

```
read -f vhdl <design name>.vhd ↵
current_design = <design name> ↵
set_max_area 0 ↵
uniquify ↵
ungroup -all -flatten ↵
compile -ungroup_all ↵
report_area > <design name>.rpa ↵
report_fpga > <design name>.rpf ↵
report_cell > <design name>.rpc ↵
edifout_write_properties_list = "lut_function" ↵
write -f edif -hierarchy -o <design name>.edf ↵
```

Use the area report to determine the circuit area.

If you wish to maintain area report estimates as closely as possible during MAX+PLUS® II processing, Altera recommends that you select the *WYSIWYG* setting for the *Global Project Synthesis Style* in the **Global Project Logic Synthesis** dialog box (Assign menu). However, selecting the *Normal* or *Fast* style may yield a better result.

Related Links:

- For more information on how to use the FPGA Compiler software optimize your design for FLEX 8000 devices, refer to *Chapter 5: Optimization for the Altera FLEX 8000 Architecture* in the ***Synopsys FPGA Compiler User Guide***.
- Go to FLEX Devices, which is available on the web, for additional information.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

MAX+PLUS IIProject File Structure

In MAX+PLUS® II, a project name is the name of a top-level design file, without the filename extension. This design file can be an EDIF, Verilog HDL, or VHDL netlist file; an AHDL TDF; or any other MAX+PLUS II-supported design file. The EDIF netlist file must be created by Synopsys and imported into MAX+PLUS II as an EDIF Input File.

MAX+PLUS II stores the connectivity data on the links between design files in a hierarchical project in a Hierarchy Interconnect File (**.hif**), but refers to the entire project only by its project name. The MAX+PLUS II Compiler uses the HIF to build a single, fully flattened project database that integrates all the design files in a project hierarchy.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

MAX 7000 & MAX 9000 Synthesis Example

The MAX® 7000 (including MAX 7000E, MAX 7000S, and MAX 7000A) and MAX 9000 device families have a sum-of-products architecture. To obtain optimum timing and area results, you can direct the Synopsys Design Compiler or FPGA Compiler software to synthesize your logic into a sum-of-products form. To assist the Synopsys compilers in meeting the timing and area constraints of your designs, the Altera® technology libraries provide models that approximate the timing of the MAX 7000 and MAX 9000 logic cells.

Figure 1 shows two timing models: the standard Altera MAX 7000 timing model and a Synopsys timing model that approximates the MAX 7000 model. The Synopsys model is built on the following three conditions and assumptions:

1. The combinatorial delay in logic cells has been equally divided between product terms and `OR` gates. Because the product-term delay equals the OR-gate delay, the Synopsys compilers treat them equally, producing a sum-of-products structure. On top of this structure, inverters are used where necessary.

2. A shared expander product term is always used to create combinatorial logic.

3. The Synopsys Design Compiler and FPGA Compiler software do not distinguish between array and global Clocks. Therefore, to estimate setup and hold timing most accurately, you must instantiate GLOBAL buffers to indicate a global clock in either your VHDL or Verilog HDL design.

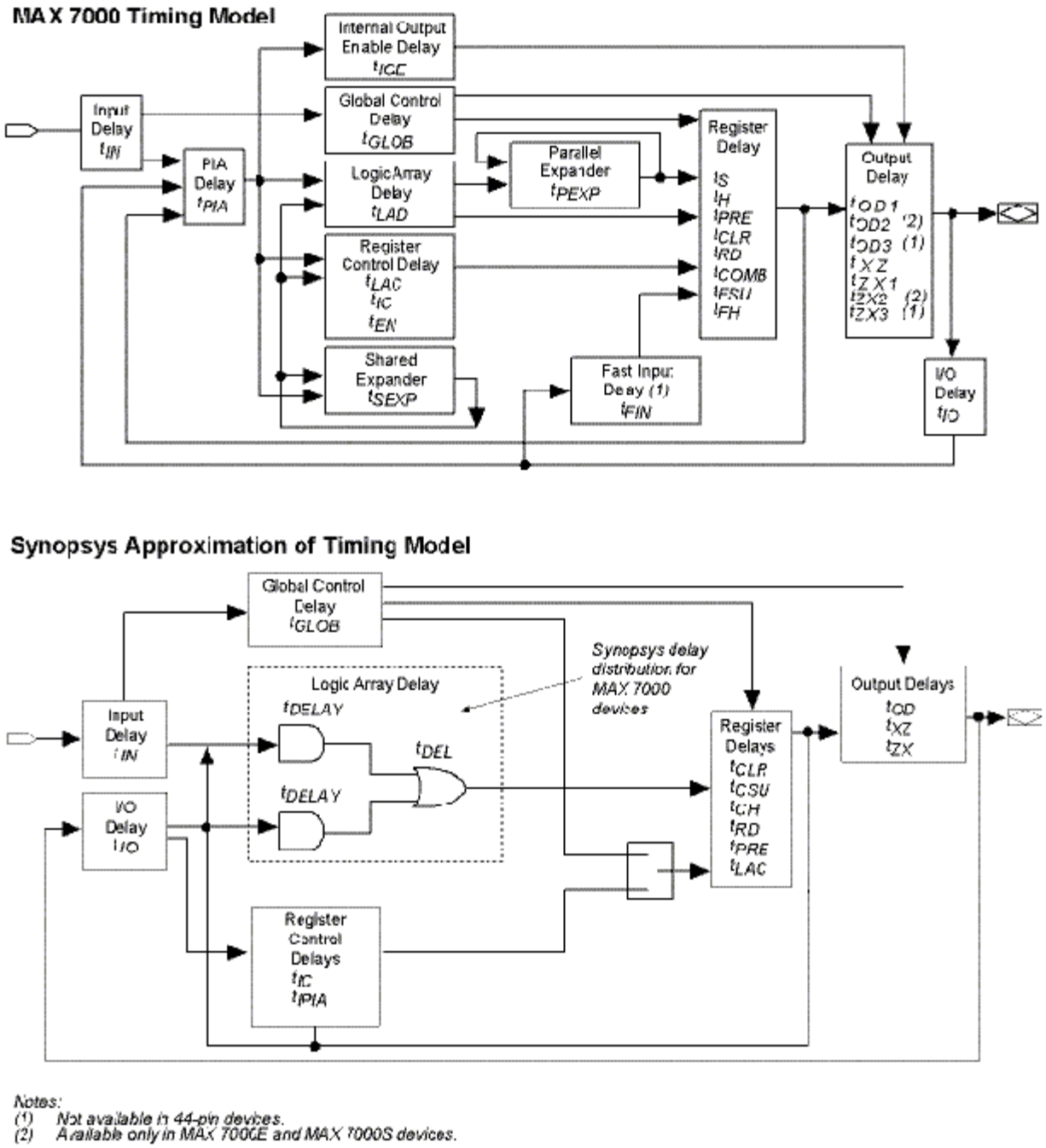


Figure 1. Standard MAX 7000 Timing Model vs. Synopsys Approximation of Timing Model

If you wish to direct the Synopsys Design Compiler or FPGA Compiler software to produce sum-of-products logic that approximates the MAX 7000 or MAX 9000 timing model, you can type the following **dc_shell** prompt commands at the command line before compiling the design:

```
set_structure false ↵

set_flatten -effort low ↵
```

When `set_structure` is set to `false`, structuring is turned off, and the Synopsys Design Compiler and FPGA Compiler software cannot factor and share logic between functions. If you do not enter these commands, the Synopsys compilers may add logic, which can create additional area and timing delays.

Figure 2 shows a combinatorial design that is predictable when structuring is turned off, but is unpredictable when structuring is turned on.

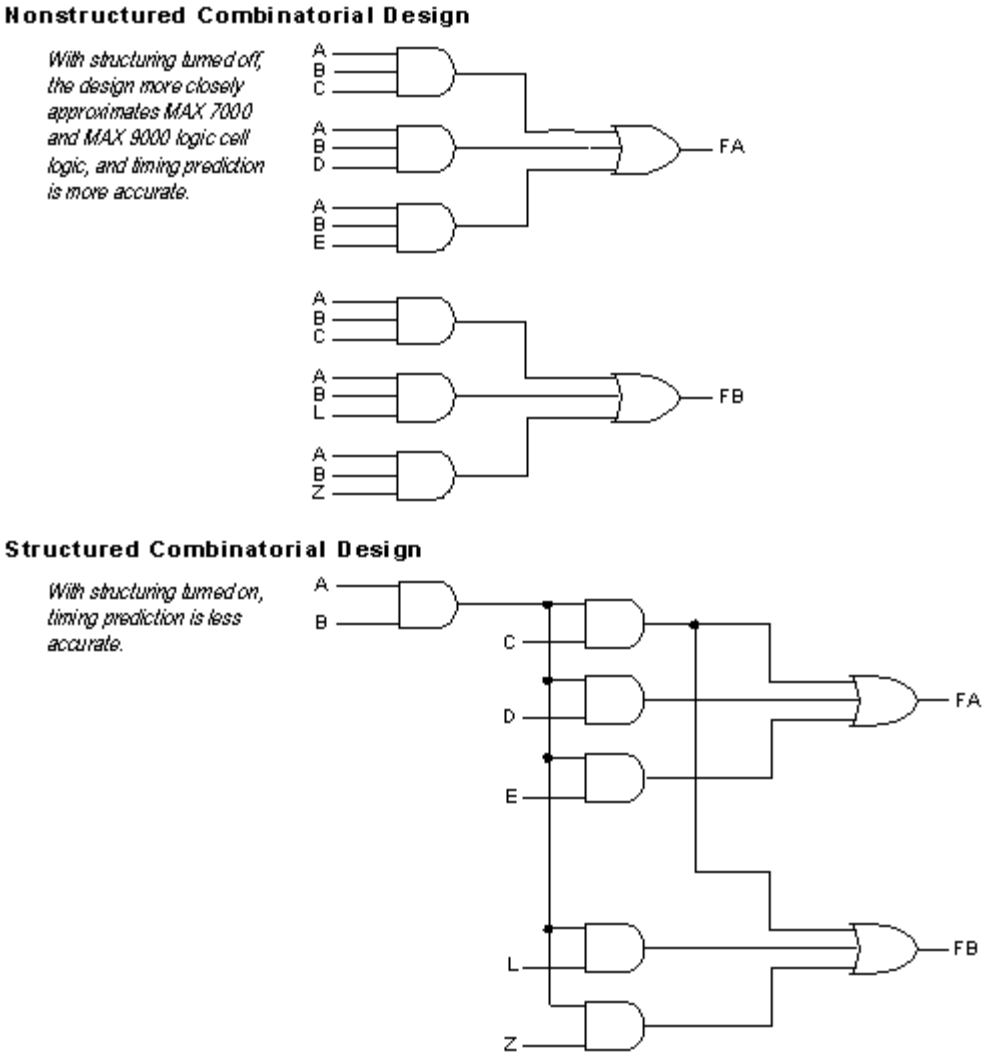


Figure 2. Nonstructured vs. Structured Combinatorial Design

When you use `low` as the argument to the `set_flatten -effort` command, the Synopsys compilers use the shortest compilation time to create the sum-of-products implementation of your design. If you use the `medium` or `high` argument, the Synopsys compilers create optimally flattened designs, but usually require greater compilation time and offer little improvement in timing and area results.

You can type `report_timing` ↵ after compilation to view Synopsys-generated timing information.

If you wish to calculate the area of your design, you can obtain an approximate logic cell count in several ways. Altera recommends that you add the number of registers and combinatorial outputs in a design. Depending on your design, this number may be slightly lower than the final number reported by the MAX+PLUS II software.

To create a file detailing primitive usage in the design, type `report_reference>` <*filename*> ↵ after Synopsys compilation.

☞ To obtain accurate timing information about your design, you must use the MAX+PLUS II Timing Analyzer to analyze your design. For accurate area information, consult the Report File (**.rpt**) generated by the MAX+PLUS II software.

Related Links:

- Refer to the following sources for related information:

- ***Synopsys Design Compiler Reference Manual*** or ***Synopsys Command Reference Manual***
- ***FPGA Compiler User Guide***
- Synthesizing & Optimizing VHDL & Verilog HDL Pojects with Synopsys Software
- Go to MAX Devices, which is available on the web, for additional information:

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Using Mentor Graphics & Exemplar Logic Tools with MAX+PLUS II Software

---

The following topics describe how to use a variety of Mentor Graphics and Exemplar Logic tools as part of a complete design flow that includes the MAX+PLUS® II software. If you use only one Mentor Graphics or Exemplar Logic tool, click List by Tool and select the tool name to view the list of topics only for that tool. Click on one of the following topics for information:

This file is suitable for printing only. It does not contain hypertext links that allow you to jump from topic to topic.

Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment

- Software Requirements
- Altera-Provided Logic & Symbol Libraries
- Local Work Area Directory Structure
- Mentor Graphics Project Directory Structure
- MAX+PLUS II Project Directory Structure
- MAX+PLUS II/Mentor Graphics/Exemplar Logic Interface File Organization

Design Flow For All Mentor Graphics/Exemplar Logic Tools

Design Entry

- **Design Entry Flow**

- **Design Architect**

    - Creating Design Architect Schematics for Use with MAX+PLUS II Software
        - Instantiating the clklock Megafunction in Design Architect Schematics
        - Instantiating LPM Functions in Design Architect Schematics
    - Entering Resource Assignments
        - Assigning Pins, Logic Cells & Chips
        - Assigning Cliques
        - Assigning Logic Options
        - Modifying the Assignment & Configuration File with the **setacf** Utility
        - BackAnnotating MAX+PLUS II Pin Assignments to Design Architect Symbols
    - Creating Hierarchical Projects with Design Architect Software
    - Performing a Functional Simulation with DVE & QuickSim II Software
    - Performing a Functional Simulation with QuickHDL Pro Software
    - Converting Design Architect Schematics into MAX+PLUS II-Compatible EDIF Netlist Files with the ENWrite Utility

-

VHDL & Verilog HDL

- Creating VHDL & Verilog HDL Designs for Use with MAX+PLUS II Software
  - Instantiating the clklock Megafunction in VHDL or Verilog HDL
  - Instantiating LPM Functions in VHDL
- Entering Resource Assignments
  - Modifying the Assignment & Configuration File with the **setacf** Utility
- Performing a Functional Simulation with QuickHDL Software
- Performing a Functional Simulation with QuickHDL Pro Software
- Creating Hierarchical Projects with Design Architect Software

Synthesis & Optimization

- Synthesizing & Optimizing VHDL & Verilog HDL Projects with Galileo Extreme Software
- Synthesizing & Optimizing VHDL & Verilog HDL Projects with Leonardo Software

Compilation

- Project Compilation Flow
- Compiling Projects with MAX+PLUS II Software
  - Exemplar Logic Galileo ExtremeSpecific Compiler Settings
- Using the Altera Schematic Express (**sch_exprss**) Utility
- Using the Altera VHDL Express (**vhd_exprss**) Utility

BackAnnotation

- BackAnnotating MAX+PLUS II Pin Assignments to Design Architect Symbols

Simulation/Timing Analysis

- Project Simulation/Timing Analysis Flow
- Initializing Registers in VHDL & Verilog Output Files for Power-Up before Simulation
- Performing a Timing Simulation with DVE & QuickSim II Software
- Performing a Timing Simulation with QuickHDL Software
- Performing a Timing Analysis with QuickPath Software

Device Programming

- Programming Altera Devices

Related Topics:

- MAX+PLUS II Development Software
- Altera Programming Hardware
- Mentor Graphics web site (http://www.mentor.com)

Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment

To use the MAX+PLUS ® II software with Mentor Graphics/Exemplar Logic software, you must install the MAX+PLUS II software, then establish an environment that facilitates entering and processing designs. The MAX+PLUS II/Mentor Graphics/Exemplar Logic interface is installed automatically when you install the MAX+PLUS II software on your computer.

Go to *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual for more information on installation and details on the directories that are created during MAX+PLUS II installation. Go to MAX+PLUS II/Mentor Graphics/Exemplar Logic Interface File Organization for information about the MAX+PLUS II/Mentor Graphics directories that are created during MAX+PLUS II installation.

The information presented here assumes that you are using a C shell and that your MAX+PLUS II system directory is **/usr/maxplus2**. If not, you must use the appropriate syntax and procedures to set environment variables for your shell.

To set up your working environment for the MAX+PLUS II/Mentor Graphics interface, follow these steps:

1. Ensure that you have correctly installed the MAX+PLUS II and Mentor Graphics software versions described in MAX+PLUS II/Mentor Graphics Software Requirements.

2. Add the following environment variables to your **.cshrc** file:

   ```
   setenv ALT_HOME /usr/maxplus2 ↵
   setenv MGC_WD <user-specified working directory> ↵
   setenv MGC_HOME <Mentor Graphics system directory> ↵
   setenv MAX2_MENTOR /usr/maxplus2/mentor/max2 ↵
   setenv MGC_LOCATION_MAP <user-specified location_map file> ↵
   setenv EXEMPLAR <Galileo or Leonardo system directory> ↵
   ```

   Installing the Altera® provided MAX+PLUS II/Mentor Graphics interface on your computer automatically installs a template for these environment variables in the **/usr/maxplus2/mentor/max2/.cshrc** file.

3. Add the **$MGC_HOME/bin**, **$MAX2_MENTOR/bin**, **$ALT_HOME/bin**, **$EXEMPLAR/bin/**<*os*>, and **$ALT_HOME/bin** directories to the `PATH` environment variable in your **.cshrc** file, where <*os*> is the operating system, e.g., `SUN4` for SunOS; `SUN5` for Solaris.

4. If you plan to use the Altera Schematic Express (**sch_exprss**) utility or the Altera VHDL Express (**vhd_exprss**) utility, add the following environment variable to your **.cshrc** file:

   ```
   setenv MAX2_QSIM /usr/maxplus2/simlib/mentor/max2sim ↵
   ```

5. Type `source â€¾/.cshrc` at a UNIX prompt to source the **.cshrc** file and validate the settings in steps 1 through 4.

6. Add the following lines to your **MGC_location_map** file:

   ```
   $MAX2_MENTOR ↵
   /usr/maxplus2/mentor/max2 ↵
   $MGC_GENLIB ↵
   /<user-specified Mentor Graphics GEN_LIB directory> ↵
   $MGC_LSLIB ↵
   /<user-specified Mentor Graphics LS_LIB directory> ↵
   $MAX2_EXAMPLES ↵
   /<user-specified example directory> ↵
   $MAX2_LMCLIB ↵
   /<user-specified Logic Modeling directory> ↵
   ```

```
$MAX2_GENLIB ↵
/usr/maxplus2/simlib/mentor/alt_max2 ↵
$MAX2_QSIM ↵
/usr/maxplus2/simlib/mentor/max2sim ↵
$MAX2_FONT ↵
/usr/maxplus2/mentor/max2/fonts ↵
$MGC_SYS1076_STD ↵
/<user-specified MGC_HOME directory>/pkgs/sys_1076_std/ std ↵
$MGC_SYS1076_ARITHMETIC ↵
/<user-specified MGC_HOME directory>/pkgs/sys_1076_std/arithmetic ↵
$MGC_SYS1076_PORTABLE ↵
/<user-specified MGC_HOME directory>/pkgs/sys_1076_std/mgc_portable ↵
$MGC_SYS1076_IEEE ↵
/<user-specified MGC_HOME directory>/pkgs/sys_1076_std/ieee ↵
$MGC_SYS1076_SRC ↵
/<user-specified MGC_HOME directory>/pkgs/sys_1076_std/ src ↵
$MAX2_MFLIB ↵
/usr/maxplus2/simlib/mentor/alt_mf ↵
```

Installing the Alteraprovided MAX+PLUS II/Mentor Graphics/Exemplar Logic interface on your computer automatically installs a template for these environment variables in the **/usr/maxplus2/mentor/max2/location_map/location_map** file.

7. If you want to use QuickHDL software to simulate VHDL or Verilog HDL designs, add the following line in the `[library]` section of your **quickhdl.ini** file: `altera = $MAX2_MFLIB`.

8. If you plan to use QuickHDL software to simulate VITAL-compliant VHDL files, add the following lines to your **MGC_location_map** file:

```
$MAX2_VTLLIB ↵
/usr/maxplus2/simlib/mentor/alt_vtl ↵
```

9. Copy the **/usr/maxplus2/maxplus2.ini** file to your **$HOME** directory:

```
cp /usr/maxplus2/maxplus2.ini $HOME ↵
chmod u+w $HOME/maxplus2.ini ↵
```

The **maxplus2.ini** file contains both Altera- and user-specified initialization parameters that control the MAX+PLUS II software, such as Alteraprovided logic and symbol library paths and the current project name. The MAX+PLUS II installation procedure creates and copies the **maxplus2.ini** file to the **/usr/maxplus2** directory.

Normally, you do not have to edit your local copy of **maxplus2.ini**, because the MAX+PLUS II software updates the file automatically whenever you change any parameters or settings. However, if you move the **max2lib** and **max2inc** library subdirectories, you must update the file. Go to "Creating & Using a Local Copy of the **maxplus2.ini** File" in MAX+PLUS II Help for more information.

Related Topics:

- Go to the following topics, which are available on the web, for additional information:
  - ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)

- This manual is also available in 4 parts:
  - Preface & Section 1: MAX+PLUS II Installation
  - Section 2: MAX+PLUS II - A Perspective
  - Section 3: MAX+PLUS II Tutorial
  - Appendices, Glossary & Index

MAX+PLUS II/Mentor Graphics Software Requirements

The following products are used to generate, process, synthesize, and verify a project with the MAX+PLUS ® II software and Mentor Graphics software:

Mentor Graphics		Exemplar	Altera
version C.1:			
System_1076 Compiler QuickSim II Design Architect ENRead ENWrite GEN_LIB library	QuickHDL QuickHDL Pro QuickPath LS_LIB library (optional) DVE	Galileo Extreme version 4.1.1 Leonardo version 4.1.3	MAX+PLUS II version 9.4

The MAX+PLUS II **read.me** file provides up-to-date information on which versions of Mentor Graphics applications are supported by the current version of MAX+PLUS II. It also provides information on installation and operating requirements. You should read the **read.me** file on the CD-ROM before installing the MAX+PLUS II software. After installation, you can open the **read.me** file from the MAX+PLUS II Help menu.

Altera-Provided Logic & Symbol Libraries

The MAX+PLUS ® II/Mentor Graphics environment provides libraries for compiling, synthesizing, and simulating designs.

You can create your own libraries of custom functions for use in Design Architect schematics and VHDL and Verilog HDL design files. You can use custom functions to incorporate an EDIF Input File (**.edf**), Text Design File (**.tdf**), or any other MAX+PLUS II-supported design file into a project. The MAX+PLUS II software uses the Altera® provided **mnt8_bas.lmf** and **exemplar.lmf** Library Mapping Files to map standard Design Architect symbols and VHDL and Verilog HDL functions to equivalent MAX+PLUS II logic functions. To use custom functions, you can create a custom LMF that maps your custom functions to the equivalent EDIF input file, TDF, or other design file. Go to "Library Mapping File" in MAX+PLUS II Help for more information.

Design Architect Libraries

You can enter a Design Architect schematic with logic functions from these Altera-provided symbol libraries: **ALTERA LPMLIB**, **ALTERA GENLIB**, **LSTTL BY TYPE**, and **LSTTL ALL PARTS**. You can access these libraries by choosing **Altera Libraries** (Libraries menu) in the Design Architect software. For information on using library of parameterized modules (LPM) functions, see **ALTERA LPMLIB** Library below.

ALTERA GENLIB Library (Design Architect) & Altera (VHDL) Libraries

The **ALTERA GENLIB** symbol library (called the **Altera** library for VHDL) includes several MAX+PLUS II primitives for controlling design synthesis and fitting. It also includes four macrofunctions (`8count`, `8mcomp`, `8fadd`, and `81mux`) that are optimized for different Altera device families, and the `clklock` phase-locked loop megafunction, which is supported for some FLEX ® 10K devices.

The following table shows the MAX+PLUS II-specific logic functions.

Table 1. MAX+PLUS II-Specific Logic Functions

Macrofunctions *Note (1)*				**Primitives**	
Name	**Description**	**Name**	**Description**	**Name**	**Description**
`8fadd`	8-bit full adder	`LCELL`	Logic cell buffer	`EXP`	MAX ® 5000, MAX 7000 , and MAX 9000 Expander buffer
`8mcomp`	8-bit magnitude comparator	`GLOBAL`	Global input buffer	`SOFT`	Soft buffer
`8count`	8-bit up/down counter	`CASCADE`	FLEX 6000, FLEX 8000, and FLEX 10K cascade buffer	`OPNDRN`	Open-drain buffer
`81mux`	8-to-1 multiplexer	`CARRY`	FLEX 6000, FLEX 8000, and FLEX 10K carry buffer	`DFFE` `DFFE6K` *Note (2)*	D-type flipflop with Clock Enable
`clklock`	Phase-locked loop				

Notes:

1. Logic function names that begin with a number must be preceded by "`a_`" in VHDL designs. For example, `8fadd` must be specified as `a_8fadd` instead.
2. If you want to use QuickHDL software, make sure you have updated your **quickhdl.ini** file, as described in step 7 of Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment.
3. For designs that are targeted for FLEX 6000 devices, you should use the `DFFE` primitive only if the design contains either a Clear or Preset signal, but not both. If your design contains both a Clear and a Preset signal, you must use the `DFFE6K` primitive.

☞ Choose **Old-Style Macrofunctions**, **Primitives**, or **Megafunctions/LPM** from the MAX+PLUS II Help menu for detailed information on these functions.

ALTERA LPMLIB Library

The Alteraprovided **ALTERA LPMLIB** library, which is available for Design Architect schematics and VHDL designs, includes standard functions from the library of parameterized modules (LPM) 2.1.0, except the truth table, finite state machine, and pad functions. The LPM standard defines a set of parameterized modules (i.e., parameterized functions) and their corresponding representations in an EDIF netlist file. These logic functions allow you to create and functionally simulate an LPM-based design without targeting a specific device family. After the design is completed, you can target the design to any device family. The parameters you specify for each LPM function determine which simulation models are generated.

☞ Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu for more information about LPM functions.

Related Topics:

- Go to the following topics, which are available on the web, for additional information:=
  - FLEX Devices
  - MAX Devices
  - Classic Device Family

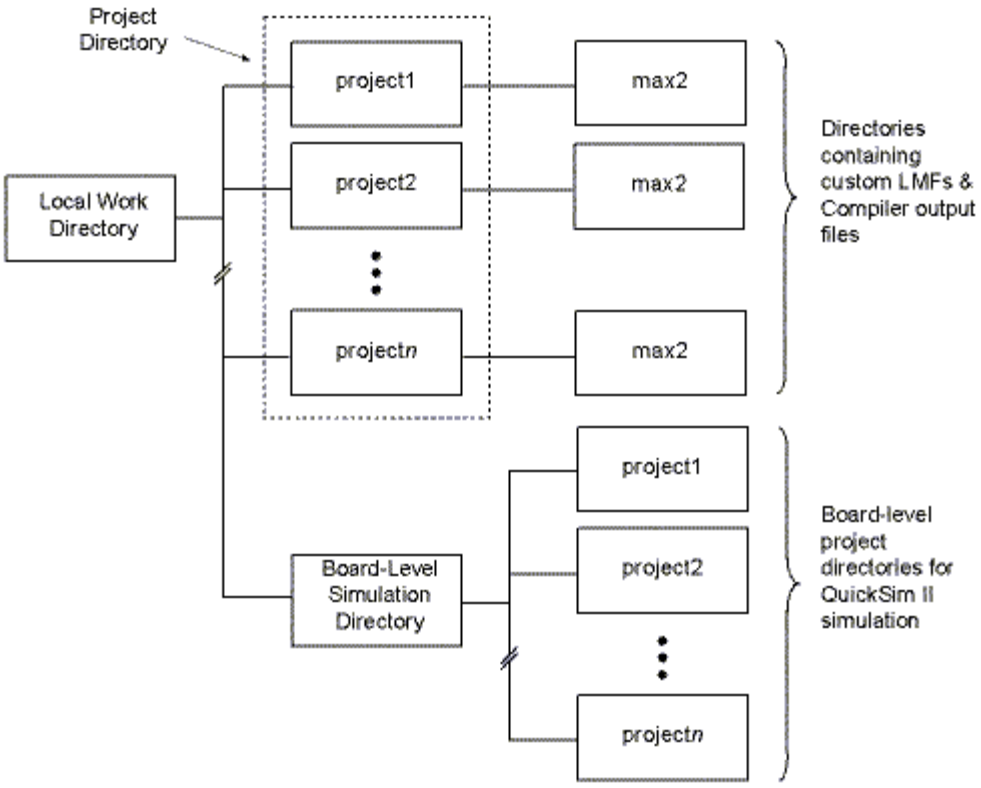
Local Work Area Directory Structure

Design Architect software automatically creates and maintains the project directory structure required for all stages of design entry. Galileo Extreme, Leonardo, and ENWrite software create a **max2** subdirectory, if it does not already exist, under the project directory. These software applications also generate EDIF netlist files, and copy them from the <*project name*> directory to this **max2** subdirectory. All MAX+PLUS ® II Compiler output files are created in the **max2** subdirectory.

Simulation files created with Mentor Graphics applications and Logic Modeling files are located in the board-level simulation subdirectory of the project directory. You can use these files during simulation with QuickSim II software.

The only directory you need to create is the local work directory, which should contain all project directories. Figure 1 shows the recommended file structure.

Figure 1. Recommended File Structure



Related Topics:

- Go to the following MAX+PLUS II ACCESS℠ Key topics for related information:
  - MAX+PLUS II Project Directory Structure
  - Mentor Graphics Project Directory Structure

Mentor Graphics Project Directory Structure

Design Architect software generates the following files for each schematic:

- <*drawing name*>**/mgc_component.attr**
- <*drawing name*>**/part.Eddm_part.attr**
- <*drawing name*>**/part.part_1**
- <*drawing name*>**/schematic.mgc_schematic.attr**
- <*drawing name*>**/schematic/schem_id**

- <*drawing name*>**/schematic/sheet1.mgc_sheet.attr**
- <*drawing name*>**/schematic/sheet1.sgfx_1**
- <*drawing name*>**/schematic/sheet1.ssht_1**

The files generated for each schematic are stored in the schematic's <*drawing name*> directory and should not be edited. Mentor Graphics software automatically manages file storage and retrieval operations through this <*drawing name*> directory structure, which does not reflect hierarchical design relationships. Figure 1 shows a sample file structure with **project1** as the UNIX project directory, and **design1**, **subdesign1**, and **subdesign2** as the directories for the top-level design and subdesigns of the project.

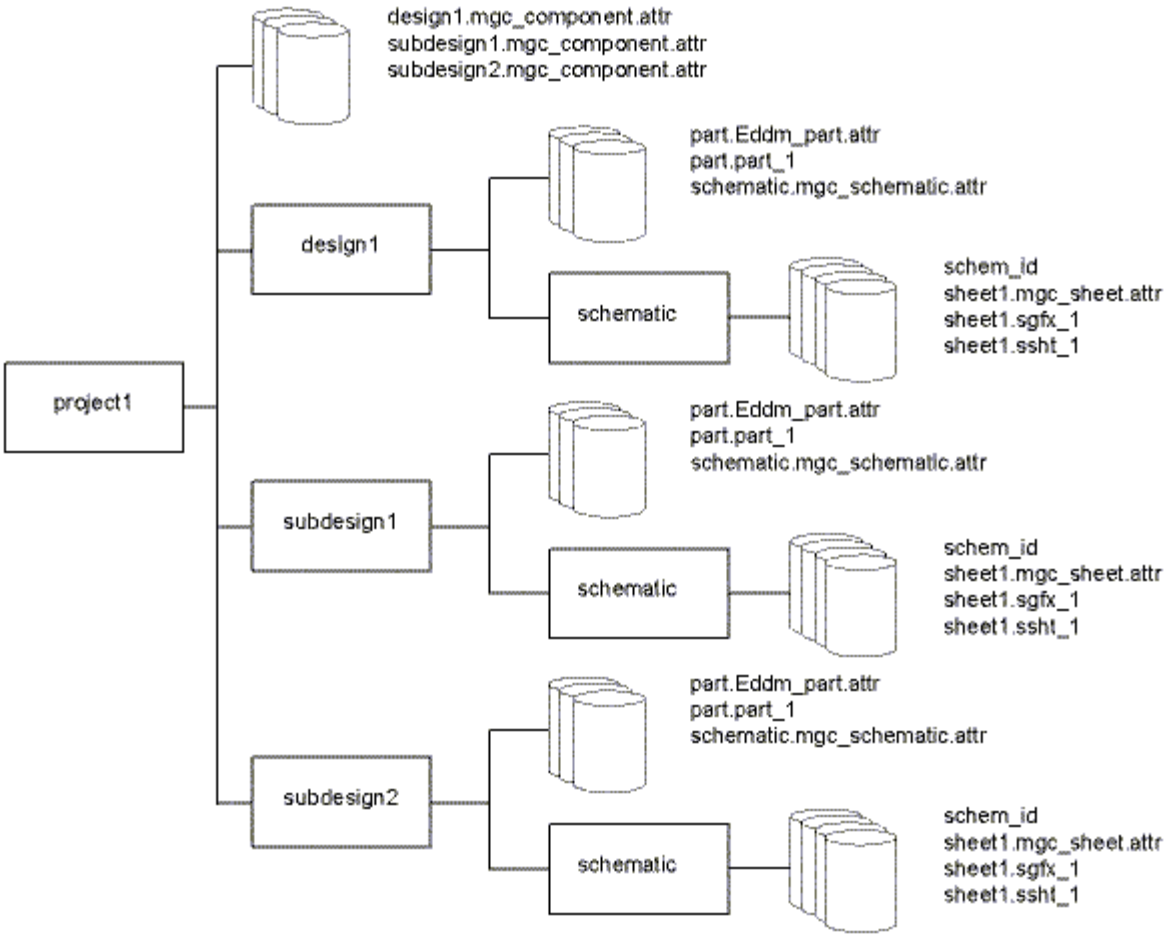


Figure 1. Design Architect Project File Structure

When the ENWrite utility converts the schematic into an EDIF netlist file, it processes the design information and all related file subdirectories, then creates the EDIF netlist file in the directory defined by the user. The EDIF netlist file is named <*project name*>**.edf**, where <*project name*> is the name of the top-level design file. The <*project name*>**.edf** file is automatically moved to the **max2** directory under the project directory.

Related Topics:

- Go to the following MAX+PLUS II ACCESS^{SM} Key topics for related information:
  - Local Work Area Directory Structure
  - MAX+PLUS II Project Directory Structure

MAX+PLUS II Project Directory Structure

In the MAX+PLUS $^{®}$ II software, a project name is the name of a top-level design file, without the filename extension. This design file can be an EDIF, VHDL, or Verilog HDL netlist file; an Altera Hardware Description

Language (AHDL) Text Design File (TDF); or any other MAX+PLUS II-supported design file. The EDIF netlist file must be created by ENWrite, Galileo Extreme, or Leonardo software and imported into MAX+PLUS II as an EDIF Input File (**.edf**). Figure 1 shows an example of a MAX+PLUS II project directory.

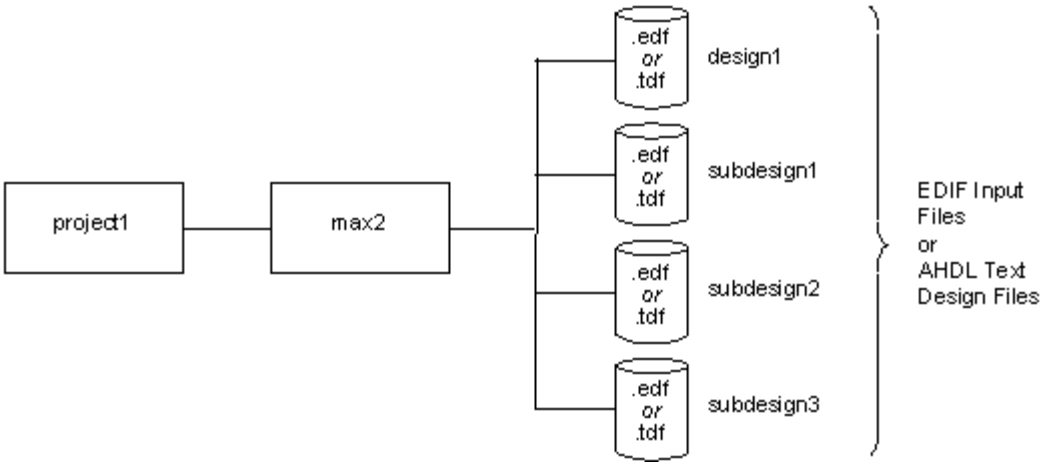


Figure 1. Sample MAX+PLUS II Project Directory

The MAX+PLUS II software stores the connectivity data on the links between design files in a hierarchical project in a Hierarchy Interconnect File (**.hif**), but refers to the entire project only by its project name. The MAX+PLUS II Compiler uses the HIF to build a single, fully flattened project database that integrates all the design files in a project hierarchy.

Related Topics:

- Go to the following MAX+PLUS II ACCESS^{SM} Key topics for related information:
  - Local Work Area Directory Structure
  - Mentor Graphics Project Directory Structure

MAX+PLUS II/Mentor Graphics/Exemplar Logic Interface File Organization

The following table shows the MAX+PLUS $^{®}$ II/Mentor Graphics interface subdirectories that are created in the MAX+PLUS II system directory (by default, the **/usr/maxplus2** directory) during MAX+PLUS II installation.

Related Topics:

- For information on the other directories that are created during MAX+PLUS II installation, see "MAX+PLUS II File Organization" in *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual.

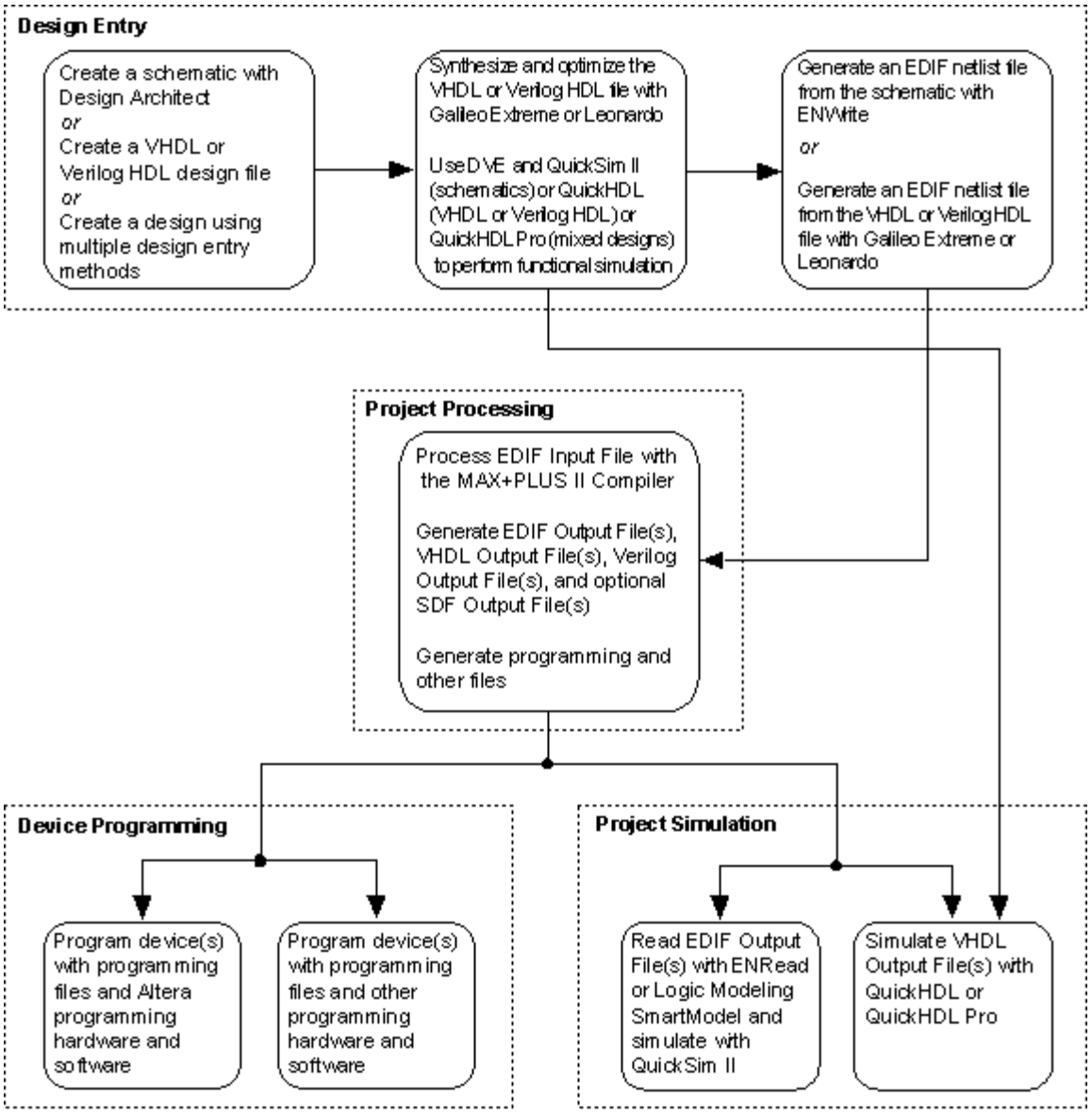
Table 1. MAX+PLUS II Directory Organization

Directory	Description
.lmf	Contains the Altera-provided Library Mapping Files, **mnt8_bas.lmf** and **exemplar.lmf**, that map Mentor Graphics and Exemplar Logic logic functions to equivalent MAX+PLUS II logic functions.
./mentor	Contains the AMPLE userware for the MAX+PLUS II/Mentor Graphics interface.
./simlib/mentor/alt_max2	Contains MAX+PLUS II primitives such as `CARRY`, `CASCADE`, `EXP`, `GLOBAL`, `LCELL`, `SOFT`, `OPNDRN`, `DFFE`, and `DFFE6K` (D flipflop with Clock Enable) for use in Design Architect schematics.

./simlib/mentor/max2sim	Contains the MAX+PLUS II/Mentor Graphics simulation model library, **max2sim**, for use with QuickSim II and QuickPath software.
./simlib/mentor/synlib	Contains the MAX+PLUS II synthesis library for use with AutoLogic II software, which supports synthesis for users running Mentor Graphics version B1.
./simlib/mentor/alt_mf	Contains the MAX+PLUS II macrofunction and megafunction libraries.
./simlib/mentor/alt_vtl	Contains the MAX+PLUS II VITAL library.

Altera/Mentor Graphics/Exemplar Logic Design Flow

The following figure shows the typical design flow for logic circuits created and processed with the MAX+PLUS ® II and Mentor Graphics/Exemplar Logic software. Detailed diagrams for each stage of the design flow appear in Design Entry Flow, Project Compilation Flow, Project Simulation/Timing Analysis Flow, and Device Programming Flow.



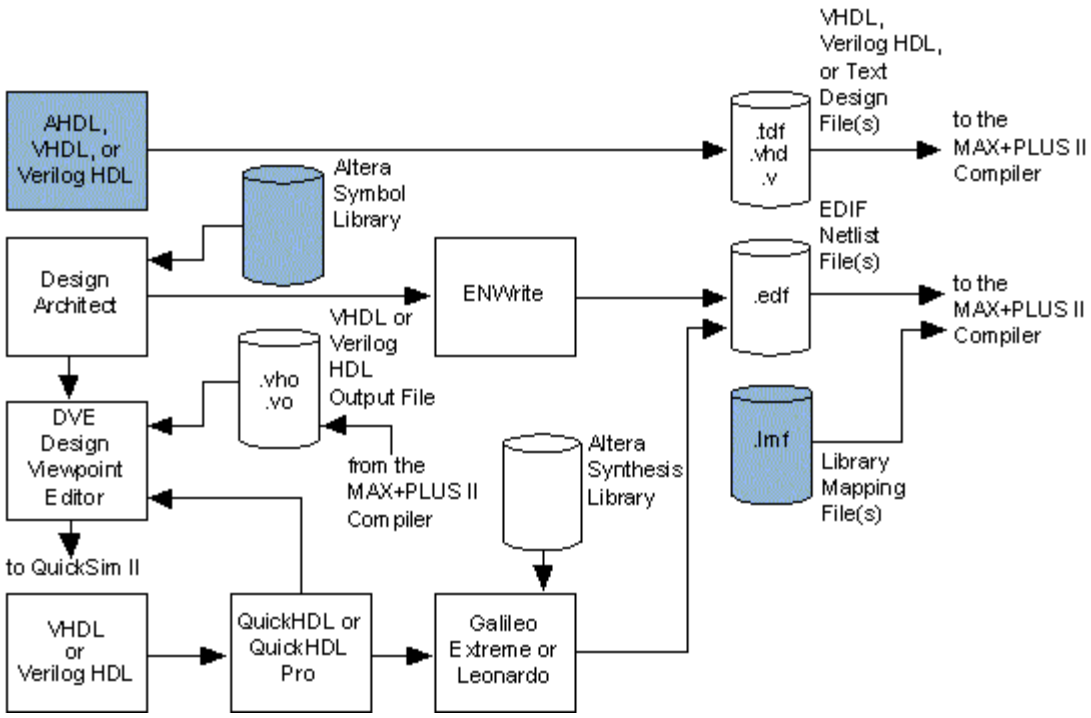
Mentor Graphics/Exemplar Logic Design Entry Flow

The following figure shows the design entry flow for the MAX+PLUS® II/Mentor Graphics/Exemplar Logic

interface.

Figure 1. MAX+PLUS II/Mentor Graphics/Exemplar Logic Design Entry Flow

Alteraprovided items are shown in blue.



Creating Design Architect Schematics for Use with MAX+PLUS II Software

You can create Design Architect schematics and convert them into EDIF Input Files (**.edf**) that can be processed with the MAX+PLUS ® II Compiler.

To create a Design Architect schematic for use with MAX+PLUS II software, go through the following steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment.
2. Start the MAX+PLUS II/Mentor Graphics interface by typing `max2_dmgr` ↵ at a UNIX prompt.
3. Start the Design Architect software by double-clicking Button 1 on the **max2_da** icon in the Design Manager tools window. You can also start Design Architect software by typing `max2_da` ↵ at the UNIX prompt.
4. Use the graphical user interface to structure and organize your files to create an environment that facilitates entering and processing designs. Go to the following topics for more information:
    - Local Work Area Directory Structure
    - MAX+PLUS II Project Directory Structure
    - Mentor Graphics Project Directory Structure
5. Choose the **OPEN SHEET** button in the Design Architect session_palette, then specify a name for your project in the *Component Name* box. Choose **OK**.

6. Enter logic functions from the following Altera® provided libraries:

    - **ALTERA LPMLIB** includes library of parameterized modules (LPM) functions
    - **ALTERA GENLIB** includes primitives and macrofunctions
    - **LSTTL** includes 74-series macrofunctions

    You can instantiate MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP^{SM}). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.

    The following topics describe special steps needed to instantiate LPM and `clklock` functions:

    - Instantiating LPM Functions in Design Architect Schematics
    - Instantiating the clklock Megafunction in Design Architect Schematics

7. (Optional) To create a hierarchical design that contains symbols representing other design files, such as AHDL or VHDL design files, go to Creating Hierarchical Projects with Design Architect Software.

8. If you wish to make resource assignments in a Design Architect schematic, go to Entering Resource Assignments. You can also enter resource assignments from within the MAX+PLUS II software.

9. Choose **Check Sheet for Altera** (Check menu) to save and check your design. If your design contains LPM functions , the Design Architect software will ask whether you want to compile the LPM model. Choose **YES** if you want to compile the VHDL code for the LPM functions. The software will automatically select the corresponding compiler: System 1076 for B.(*x*) releases and QuickHDL compilers for releases C.1 and later.

10. (Optional) If your schematic design includes models for VHDL or Verilog HDL designs, perform a functional simulation with the QuickHDL Pro software, as described in Performing a Functional Simulation with QuickHDL Pro Software. If it does not, you can perform a functional simulation with the QuickSim software, as described in Performing a Functional Simulation with DVE & QuickSim II Software.

11. Once you have created a schematic, you can generate an EDIF netlist file that can be imported into the MAX+PLUS II software with either of the following methods:

    - You can create an EDIF netlist file, as described in Converting Design Architect Schematics into MAX+PLUS II-Compatible EDIF Netlist Files with the ENWrite Utility.

    - You can use the Altera Schematic Express utility, **sch_exprss**, to automatically create an EDIF netlist file, compile it with the MAX+PLUS II Compiler, generate an EDIF Output File (**.edo**), and prepare the EDIF Output File for simulation with ENRead and Design Viewpoint Editor (DVE), as described in Using the Altera Schematic Express (**sch_exprss**) Utility.

    Even if your design is a hierarchical design incorporating files created with multiple design entry methods, both the ENWrite and Altera Schematic Express utilities generate EDIF files for all files in the design.

    Installing the Alteraprovided MAX+PLUS II/Mentor Graphics/Exemplar Logic interface on your computer automatically creates the following sample Design Architect schematic files:

- **/usr/maxplus2/examples/mentor/example1/fulladd**
- **/usr/maxplus2/examples/mentor/example3/fulladd2**
- **/usr/maxplus2/examples/mentor/example7/fifo**

Related Topics:

- Go to Compiling Projects with MAX+PLUS II Software in these MAX+PLUS II ACCESS℠ Key topics for related information.

Instantiating the clklock Megafunction in Design Architect Schematics

You can instantiate the Altera® provided `clklock` phase-locked loop megafunction, which is supported for some FLEX ® 10K devices, in a Design Architect schematic.

To instantiate the `clklock` megafunction in a Design Architect schematic, follow these steps:

1. Choose **Altera Libraries** (Library menu).
2. Choose **ALTERA GENLIB** (Altera Libraries menu).
3. Choose **clklock** (ALTERA GENLIB menu).
4. Specify appropriate values for the `CLOCKBOOST` and `INPUT_FREQUENCY` variables. Choose **Megafunctions/LPM** from the MAX+PLUS ® II Help menu for detailed information on the `clklock` megafunction.
5. Choose **OK**.
6. Continue with the steps necessary to complete your Design Architect schematic, as described in Creating Design Architect Schematics for Use with MAX+PLUS II Software.

Installing the Alteraprovided MAX+PLUS II/Mentor Graphics/Exemplar Logic interface on your computer automatically creates the sample Design Architect schematic file **/usr/maxplus2/examples/mentor/example7/fifo**, which includes `clklock` megafunction instantiation.

Related Topics:

- Go to FLEX 10K Device Family, which is available on the web, for additional information.

Instantiating LPM Functions in Design Architect Schematics

Design Architect software allows you to instantiate functions included in the library of parameterized modules (LPM) from the **ALTERA LPMLIB** library.

Go through the following steps to instantiate LPM functions in a Design Architect schematic:

1. While you are entering your Design Architect schematic, choose **Altera Libraries** (Library menu).
2. Choose **ALTERA LPMLIB** (Altera Libraries menu).
3. Choose from the available LPM functions on the ALTERA GENLIB menu.
4. In the **LPM_**<*function name*> dialog box, specify appropriate values for the variables displayed for the LPM function you chose in step 3. Make sure that any hexadecimal (Intel-format) file that you use to specify the initial content of a memory function does not have the same name as the design file name. Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu for detailed information on LPM functions.
5. Choose **OK** to generate a symbol for the LPM function you chose in step 3 and a corresponding VHDL

simulation model.

6. Continue with the steps necessary to complete your Design Architect schematic, as described in Creating Design Architect Schematics for Use with MAX+PLUS II Software.

7. When you save the schematic, the Design Architect software will ask whether you want to compile the LPM model. Choose **YES** if you want to compile the VHDL code for the LPM functions. The software will automatically select the corresponding compiler: System 1076 for B.(*x*) releases and QuickHDL compilers for releases C.1 and later.

Installing the Altera® provided MAX+PLUS II/Mentor Graphics/Exemplar Logic interface on your computer automatically creates the sample Design Architect schematic file **/usr/maxplus2/examples/mentor/example7/fifo**, which includes LPM instantiation.

Entering Resource Assignments

The MAX+PLUS ® II software allows you to enter a variety of resource and device assignments for your projects. Resource assignments are used to assign logic functions to a particular pin, logic cell, I/O cell, embedded cell, row, column, Logic Array Block (LAB), Embedded Array Block (EAB), chip, clique, local routing, logic option, timing requirement, or connected pin group. In MAX+PLUS II software, you can enter all types of resource and device assignments with Assign menu commands. You can also enter pin, logic cell, I/O cell, embedded cell, LAB, EAB, row, and column assignments in the MAX+PLUS II Floorplan Editor. The Assign menu commands and the Floorplan Editor all save assignment information in the ASCII Assignment & Configuration File (**.acf**) for the project. In addition, you can edit ACFs manually in any standard text editor or with the **setacf** utility.

Design Architect Schematics

In Design Architect schematics, you can assign a limited subset of these resource assignments by assigning properties to symbols. These properties are incorporated into the EDIF netlist file(s). The MAX+PLUS II software automatically converts assignment information from the EDIF Input File into the ACF format. For information on making MAX+PLUS II-compatible resource assignments, go to the following topics:

- Assigning Pins, Logic Cells & Chips
- Assigning Cliques
- Assigning Logic Options
- Modifying the Assignment & Configuration File with the **setacf** Utility

☞ After you compile a project, you can back-annotate pin assignments, as described in BackAnnotating MAX+PLUS II Pin Assignments to Design Architect Symbols.

Installing the Alteraprovided MAX+PLUS II/Mentor Graphics/Exemplar Logic interface on your computer automatically creates the sample Design Architect schematic file **/usr/maxplus2/examples/mentor/example4/fa2**, which includes resource assignments.

VHDL & Verilog HDL Design Files

For Verilog HDL- and VHDL-based designs, you must use the MAX+PLUS II software or the **setacf** utility to enter resource assignments. Go to Modifying the Assignment & Configuration File with the **setacf** Utility for more information.

Related Topics:

- Go to "Resource Assignments in EDIF Input Files" and "Assigning Resources in a Third-Party Design

Editor" in MAX+PLUS II Help for more information on assignments or properties that can be assigned in Design Architect software. For information on entering assignments in MAX+PLUS II software with Assign menu commands or in an ACF, go to "resource assignments" or "ACF, format" in MAX+PLUS II Help using **Search for Help on** (Help menu).

Assigning Pins, Logic Cells & Chips

You can assign a single logic function to a specific pin or logic cell (including I/O cells and embedded cells) within a chip, and assign one or more functions to a specific chip. A chip is a group of logic functions defined as a single, named unit, which can be assigned to a specific device.

You can assign a signal to a particular pin to ensure that the signal is always associated with that pin, regardless of future changes to the project. If you wish to set and maintain the performance of your project, assigning logic to a specific logic cell within a chip can minimize timing delays. In a project that is partitioned among multiple devices, you can assign logic functions that must be kept together in the same device to a chip. Chip assignments allow you to split a project so that only a minimum number of signals travel between devices, and to ensure that no unnecessary device-to-device delays exist on critical timing paths. You can assign a chip to a device in some EDA tools or in the MAX+PLUS® II software.

Use the following syntax for chip, pin, and logic cell assignments:

- To assign a logic function to a chip:

  `CHIP_PIN_LC=`*<chip name>*

  For example: `CHIP_PIN_LC=chip1`

- To assign a pin number within a chip:

  `CHIP_PIN_LC=`*<chip name>*`@`*<pin number>*

  For example: `CHIP_PIN_LC=chip1@K2`

- To assign a logic cell, I/O cell, or embedded cell number:

  `CHIP_PIN_LC=`*<chip name>*`@LC`*<logic cell number>*

  `CHIP_PIN_LC=`*<chip name>*`@IOC`*<I/O cell number>*

  `CHIP_PIN_LC=`*<chip name>*`@EC`*<embedded cell number>*

  For example: `CHIP_PIN_LC=chip1@LC44`

Related Topics:

- Go to "Devices & Adapters" and "Assigning a Device" in MAX+PLUS II Help for information on device pin-outs and assigning devices, respectively, in the MAX+PLUS II software.
- Go to Back-Annotating MAX+PLUS II Pin Assignments to Design Architect Symbols for information on back-annotating pin assignments in Mentor Graphics Design Architect schematics.

Assigning Cliques

You can define a group of logic functions as a single, named unit, called a clique. The MAX+PLUS® II Compiler attempts to place all logic in the clique in the same logic array block (LAB) to ensure optimum speed. If the project

does not use multi-LAB devices, or if it is not possible to fit all clique members into a single LAB, the clique assignment ensures that all members of a clique are placed in the same device. In FLEX® 6000, FLEX 8000, FLEX 10K, and MAX® 9000 devices the Compiler also attempts to place the logic in LABs in the same row. Cliques therefore allow you to partition a project so that only a minimum number of signals travel between LABs, and to ensure that no unnecessary LAB-to-LAB or device-to-device delays exist on critical timing paths.

To assign a clique, use the following syntax:

✔ `CLIQUE=`<*clique name*>

For example: `CLIQUE=fast1`

Related Topics:

- Go to the following topics in MAX+PLUS II Help for related information:
  - Assigning a Clique
  - Guidelines for Achieving Maximum Speed Performance

Assigning Logic Options

Logic option and logic synthesis style assignments allow you to guide logic synthesis with logic optimization features that are specific to Altera® devices. You can assign logic options and styles to individual logic functions in your design. The MAX+PLUS® II Compiler also uses a device-family-specific default logic synthesis style for each project.

Related Topics:

- Go to "Resource Assignments in EDIF Input Files" and "Assigning Resources in a Third-Party Design Editor" in MAX+PLUS II Help for complete and up-to-date information on logic option and logic synthesis style assignments, including definitions and syntax of these assignments.

Modifying the Assignment & Configuration File with the setacf Utility

Altera provides the **setacf** utility to help you modify a project's Assignment & Configuration File (**.acf**) from the command line, without opening the file with a text editor. Type `setacf -h` ↵ at a UNIX or DOS prompt to get help on this utility.

BackAnnotating MAX+PLUS II Pin Assignments to Design Architect Symbols

The MAX+PLUS ® II/Mentor Graphics software interface includes the **annotate_pin** utility. This utility allows you to back-annotate the pin assignments from the MAX+PLUS II-generated Fit File (**.fit**) back to the symbol for the design file. The **annotate_pin** utility has the following syntax:

`annotate_pin [-p` <*property name*>`]` <*symbol name*> <*chip name*> <*Fit File name*> ↵

where <*property name*> is the default name for the pin assignment (default is `PIN_NO`), <*symbol name*> is the pathname of the directory that contains the symbol, <*chip name*> is the chip name specified in the Fit File, and <*Fit File name*> is the name of the Fit File that contains the pin assignment information for back-annotation. If the

<*property name*> is not found at a pin number, that pin will not be back-annotated. If the <*chip name*> is not found in the Fit File, the **annotate_pin** utility stops the back-annotation process.

For example:

```
annotate_pin -p PIN_NO /usr/examples/decode decode decode.fit ↵
```

☞ Type `annotate_pin -h ↵` at the UNIX prompt to display information on how to use this utility.

Creating Hierarchical Projects with Design Architect Software

If you wish to create a hierarchical schematic design that contains symbols representing other design files, such as AHDL Text Design Files (**.tdf**), VHDL Design Files (**.vhd**), or Verilog Design Files (**.v**), you can create a hollow-body symbol for the design file and then instantiate it in your top-level design file.

To create a hollow-body symbol for a lower-level design file, follow these steps:

1. (Optional) If you are creating a hollow-body symbol for a VHDL or Verilog HDL design file, you can first functionally simulate the VHDL or Verilog HDL file, as described in Performing a Functional Simulation with QuickHDL Software.
2. Start the Design Architect software by double-clicking Button 1 on the **max2_da** icon in the Design Manager tools window. You can also start Design Architect software by typing `max2_da ↵` at the UNIX prompt.
3. Choose the **OPEN SYMBOL** button in the Design Architect session_palette to open the Symbol Editor. Type the lower-level design file name, including the directory path, in the *Component Name* box. Choose **OK**.
4. Create a symbol that represents the inputs and outputs of the lower-level file.
5. Assign `PINTYPE` properties of `IN` or `OUT` to the inputs and outputs of the symbol, and assign appropriate values to any other properties of the symbol so that it can be identified in the top-level schematic.

   ☞ If you are creating a hollow-body symbol for a VHDL design file, be sure to assign the value `qvpro` to the symbol's `model` property so that it can be identified as a VHDL component in the top-level schematic.
6. Check and save the symbol, then close the Symbol Editor.
7. To enter the symbol, choose the **CHOOSE SYMBOL** button from the Design Architect session_palette.
8. Select the symbol file from the Navigator menu and choose **OK**.
9. The MAX+PLUS® II software uses the Altera® provided **mnt8_bas.lmf** Library Mapping File to map Design Architect symbols to equivalent MAX+PLUS II logic functions. To use custom symbols, you must create a custom LMF that maps your custom symbols to the equivalent EDIF Input File, Text Design File (TDF), or other design file. You will also need to specify this LMF in the **EDIF Netlist Reader Settings** dialog box before compiling the design with the MAX+PLUS II software. See Compiling Projects with MAX+PLUS II Software for more information.
10. Continue with the steps necessary to complete your Design Architect schematic, as described in Creating Design Architect Schematics for Use with MAX+PLUS II Software.

Installing the Alteraprovided MAX+PLUS II/Mentor Graphics/Exemplar Logic interface on your computer automatically creates the sample hierarchical Design Architect schematic file

/usr/maxplus2/examples/mentor/example3/fulladd2.

Performing a Functional Simulation with DVE & QuickSim II Software

You can perform a functional simulation of a Design Architect schematic with the Mentor Graphics Design Viewpoint Editor (DVE) and QuickSim II software before compiling your project with the MAX+PLUS ® II Compiler.

☞ If you wish to functionally simulate a hierarchical design that uses multiple design entry methods, you should use QuickHDL Pro rather than QuickSim. Refer to Performing a Functional Simulation with QuickHDL Pro Software for more information.

To functionally simulate a Design Architect schematic, go through the following steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment.
2. Create a Design Architect schematic that follows the guidelines in Creating Design Architect Schematics for Use with MAX+PLUS II Software.
3. In the Navigator window, select your project's folder, press Button 3, and choose **Open max2_fve** to start DVE. DVE checks the design and creates a viewpoint (called **altera_fsim** by default) for functional simulation with QuickSim II software.
4. Select the **altera_fsim** icon, press Button 3, and choose **Open max2_qsim** from the Navigator window to start the QuickSim II software. You can also start the QuickSim II software by typing `max2_qsim` ↵ at the UNIX prompt.
5. Set the appropriate options and simulate your design.
6. Use the ENWrite utility to generate an EDIF netlist file that can be imported into the MAX+PLUS II software, as described in Converting Design Architect Schematics into MAX+PLUS II-Compatible EDIF Netlist Files with the ENWrite Utility.

Related Topics:

- Go to Compiling Projects with MAX+PLUS II Software in these MAX+PLUS II ACCESS℠ Key topics for related information.

Converting Design Architect Schematics into MAX+PLUS II- Compatible EDIF Netlist Files with the ENWrite Utility

After you have created a Design Architect schematic or a hierarchical schematic design that uses multiple design entry methods, you can use the Mentor Graphics ENWrite utility to convert it into an EDIF netlist file that can be processed with the MAX+PLUS ® II software.

To generate an EDIF netlist file for use with the MAX+PLUS II Compiler, go through the following steps:

1. Create a Design Architect Schematic that follows the guidelines described in Creating Design Architect Schematics for Use with MAX+PLUS II Software.
2. Select the folder for your project, press Button 3, and choose **Open max2_enw** from the Navigator window to open Design Viewpoint Editor (DVE), then ENWrite. You can also start the ENWrite utility by typing

`max2_enw` ↵ at the UNIX prompt.

3. Choose **OK** in the **$invoke_enw** dialog box to accept the default names for the DVE viewpoint **altera_edif**, which is used internally by ENWrite, and the ENWrite hierarchical EDIF netlist file <*design name*>**.edf**. Specify *OFF* for the port array construct in the EDIF netlist file.

   ☞ The MAX+PLUS II software supports bus constructs in EDIF 2 0 0 and 3 0 0 netlist files, which allow you to retain any bus structures in your design. To preserve a bus in the EDIF netlist file, turn on the *port array* construct option in the **$invoke_enw** dialog box. However, if your design contains library of parameterized modules (LPM) functions, you should not use this feature because LPM 2.0.1 and 2.1.0 functions do not support EDIF bus constructs.

   After DVE checks the Design Architect schematic, ENWrite generates <*design name*>**.edf** and automatically copies it to your project's directory.

4. Compile the resulting EDIF netlist file with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Creating VHDL & Verilog HDL Designs for Use with MAX+PLUS II Software

You can create VHDL and Verilog HDL design files with the MAX+PLUS ® II Text Editor or another standard text editor and save them in the appropriate directory for your project.

The MAX+PLUS II Text Editor offers the following advantages:

- Templates are available with the **VHDL Templates** and **Verilog Templates** commands (Template menu). These templates are also available in the ASCII **vhdl.tmp** and **verilog.tmp** files, respectively, which are located in the **/usr/maxplus2** directory.
- If you use the MAX+PLUS II Text Editor to create your VHDL design, you can turn on the **Syntax Coloring** command (Options menu). The Syntax Coloring feature displays keywords and other elements of text in text files in different colors to distinguish them from other forms of syntax.

To create a VHDL or Verilog HDL design file for use with the MAX+PLUS II software, go through the following steps:

1. Enter a VHDL or Verilog HDL design in the MAX+PLUS II Text Editor or another standard text editor and save it in your working directory.
2. Enter primitives, macrofunctions, and megafunctions in your VHDL or Verilog HDL design from the **Altera** library.

   The following topics describe special steps needed to instantiate LPM and `clklock` functions:

   - Instantiating LPM Functions in VHDL
   - Instantiating the clklock Megafunction in VHDL or Verilog HDL

   ☞ You can instantiate MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.

3. (Optional) Use the QuickHDL software to functionally simulate the design file, as described in Performing a

Functional Simulation with QuickHDL Software and Performing a Functional Simulation with QuickHDL Pro Software.

4. Once you have created a VHDL or Verilog HDL design, you can generate an EDIF netlist file that can be imported into the MAX+PLUS II software with either of the following methods:

    - You can synthesize and optimize your design and create an EDIF netlist file, as described in Synthesizing & Optimizing VHDL & Verilog HDL Projects with Galileo Extreme Software or Synthesizing & Optimizing VHDL & Verilog HDL Projects with Leonardo Software.

    - You can use the Altera VHDL Express utility, **vhd_exprss**, to automatically create an EDIF netlist file, compile it with the MAX+PLUS II Compiler, generate an EDIF Output File (**.edo**), and prepare the EDIF Output File for simulation with QuickHDL software, as described in Using the Altera Schematic Express (**vhd_exprss**) Utility.

    Installing the Alteraprovided MAX+PLUS II/Mentor Graphics/Exemplar Logic interface on your computer automatically creates the following sample VHDL design files:

- **/usr/maxplus2/examples/mentor/example5/count4.vhd**
- **/usr/maxplus2/examples/mentor/example6/count8.vhd**
- **/usr/maxplus2/examples/mentor/example8/adder16.vhd**

Related Topics:

- Go to Compiling Projects with MAX+PLUS II Software in these MAX+PLUS II ACCESS^{SM} Key topics for related information.

Instantiating the clklock Megafunction in VHDL & Verilog HDL Designs

Altera provides the **genckklk** utility to allow you to instantiate clklock (phaselocked loop) functions in Mentor Graphics/Exemplar Logic software. The **genckklk** utility appends the parameter values to the clklock function name, so you don't need to declare attributes explicitly. The naming rule for the clklock function is clklock_ <*ClockBoost*>_<*inputfrequency*>. The **genckklk** utility has the following syntax:

genckklk <*ClockBoost*> <*inputfrequency*> [vhdl] [verilog] ↵

For the <*ClockBoost*> variable, you should specify a ClockBoost value of 1 or 2 (default value is 1). For the <*inputfrequency*> variable, you should specify a decimal value in MHz (default value is 50). To generate a VHDL file (which is the default if no option is present), specify vhdl; to generate a Verilog HDL file, specify verilog.

For example, to create the VHDL file **clklock_2_50.vhd** and the corresponding Component Declaration file **clklock_2_50.cmp**, type the following command at the UNIX prompt:

genckklk 2 50 -vhdl ↵

Installing the Alteraprovided MAX+PLUS II/Mentor Graphics interface on your computer automatically creates the sample VHDL design file **/usr/maxplus2/examples/mentor/example6/count8.vhd**, which includes clklock megafunction instantiation.

Instantiating LPM Functions in VHDL

You can use Mentor Graphics Design Architect software to help you instantiate library of parameterized modules (LPM) functions in your VHDL design files.

To incorporate an LPM function into a VHDL design file, perform the following steps:

1. Be sure to set up the Design Architect working environment correctly, as described in Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment.
2. Open a dummy schematic in the Design Architect software:
    1. Start the Altera® /Mentor Graphics interface by typing `max2_dmgr` ↵ at a UNIX prompt.
    2. Start the Design Architect software by double-clicking Button 1 on the **max2_da** icon in the Design Manager tools window.
    3. Choose the **OPEN_SHEET** button in the Design Architect session_palette, then specify your project name in the *Component Name* box. Choose **OK**.
3. Instantiate the desired LPM function in the dummy schematic:
    1. Choose **Altera Libraries** (Library menu).
    2. Choose **ALTERA LPMLIB** (Altera Libraries menu).
    3. Choose from the available LPM functions on the ALTERA LPMLIB menu.
    4. In the **LPM_**<*lpm function*> dialog box, specify a name for the LPM function in the *Cell Name* box and appropriate values for the function's parameters. Make sure that any hexadecimal (Intel-format) file that you use to specify the initial content of a memory function does not have the same name as the design file name. Choose **Megafunctions/LPM** from the MAX+PLUS® II Help menu for detailed information about LPM functions.
    5. Choose **OK** to generate the LPM function, the corresponding VHDL simulation model, and a VHDL Component Declaration/Attribute Declaration/Attribute Specification (**.cmp**) template.
4. Close the Design Architect software without saving the dummy schematic.
5. Instantiate the function created in step 2 in your design file. Use the template file to help prevent syntax and other errors.
6. Continue with the steps necessary to complete your design file, as described in Creating VHDL & Verilog HDL Designs for Use with MAX+PLUS ® II Software.

Installing the Alteraprovided MAX+PLUS II/Mentor Graphics/Exemplar Logic interface on your computer automatically creates the sample hierarchical VHDL design file **/usr/maxplus2/examples/mentor/example8/adder16.vhd**, which includes LPM function instantiation.

Entering Resource Assignments

The MAX+PLUS ® II software allows you to enter a variety of resource and device assignments for your projects. Resource assignments are used to assign logic functions to a particular pin, logic cell, I/O cell, embedded cell, row, column, Logic Array Block (LAB), Embedded Array Block (EAB), chip, clique, local routing, logic option, timing requirement, or connected pin group. In MAX+PLUS II software, you can enter all types of resource and device assignments with Assign menu commands. You can also enter pin, logic cell, I/O cell, embedded cell, LAB, EAB,

row, and column assignments in the MAX+PLUS II Floorplan Editor. The Assign menu commands and the Floorplan Editor all save assignment information in the ASCII Assignment & Configuration File (**.acf**) for the project. In addition, you can edit ACFs manually in any standard text editor or with the **setacf** utility.

Design Architect Schematics

In Design Architect schematics, you can assign a limited subset of these resource assignments by assigning properties to symbols. These properties are incorporated into the EDIF netlist file(s). The MAX+PLUS II software automatically converts assignment information from the EDIF Input File into the ACF format. For information on making MAX+PLUS II-compatible resource assignments, go to the following topics:

- Assigning Pins, Logic Cells & Chips
- Assigning Cliques
- Assigning Logic Options
- Modifying the Assignment & Configuration File with the **setacf** Utility

After you compile a project, you can back-annotate pin assignments, as described in BackAnnotating MAX+PLUS II Pin Assignments to Design Architect Symbols.

Installing the Alteraprovided MAX+PLUS II/Mentor Graphics/Exemplar Logic interface on your computer automatically creates the sample Design Architect schematic file **/usr/maxplus2/examples/mentor/example4/fa2**, which includes resource assignments.

VHDL & Verilog HDL Design Files

For Verilog HDL- and VHDL-based designs, you must use the MAX+PLUS II software or the **setacf** utility to enter resource assignments. Go to Modifying the Assignment & Configuration File with the **setacf** Utility for more information.

Related Topics:

- Go to "Resource Assignments in EDIF Input Files" and "Assigning Resources in a Third-Party Design Editor" in MAX+PLUS II Help for more information on assignments or properties that can be assigned in Design Architect software. For information on entering assignments in MAX+PLUS II software with Assign menu commands or in an ACF, go to "resource assignments" or "ACF, format" in MAX+PLUS II Help using **Search for Help on** (Help menu).

Modifying the Assignment & Configuration File with the setacf Utility

Altera provides the **setacf** utility to help you modify a project's Assignment & Configuration File (**.acf**) from the command line, without opening the file with a text editor. Type `setacf -h` ↵ at a UNIX or DOS prompt to get help on this utility.

Performing a Functional Simulation with QuickHDL Software

You can use Mentor Graphics QuickHDL software to functionally simulate VHDL or Verilog HDL design files before compiling them with the MAX+PLUS ® II Compiler.

If you wish to functionally simulate a hierarchical design that uses multiple design entry methods, you should use QuickHDL Pro rather than QuickHDL. Refer to Performing a Functional Simulation with QuickHDL Pro Software for more information.

To functionally simulate a VHDL or Verilog HDL design, follow these steps:

1. Be sure to set up the working environment correctly, as described in Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment.
2. Create a VHDL or Verilog HDL design file that follows the guidelines described in Creating VHDL & Verilog HDL Designs for Use with MAX+PLUS II Software.
3. Start Design Architect by double-clicking Button 1 on the **max_da** icon in the Design Manager tools window. You can also start Design Architect software by typing `max2_da` ↵ at the UNIX prompt.
4. Choose **Lib** (QuickHDL menu) and specify your work library name as the *Work Library* name. Choose **OK**.
5. Choose **Map** (QuickHDL menu) to map the instantiated function to the equivalent function in the **Altera** logic function library. Choose **Set** to specify *altera* as the *Logical Name* and *$MAX2_MFLIB* as the *Physical Name*. Choose **OK**.
6. Choose **Compile** (QuickHDL menu) and use the Navigator window to select the icon for your project. Specify your work library name as the *Work Library* name and select the *Simulation* setting in the Set VHDL Compilation Options or Set Verilog HDL Compilation Options window. Choose **OK** to compile.
7. Choose **Simulate** (QuickHDL menu) and specify your work library name as the *Work Library* name. Choose **OK** to start the QuickHDL Startup window.
8. Select the icon for your project in the Entity Configuration window and choose **OK** to simulate the design.
9. Synthesize and optimize the design, as described in Synthesizing & Optimizing VHDL & Verilog HDL Projects with Galileo Extreme Software or Synthesizing & Optimizing VHDL & Verilog HDL Projects with Leonardo Software.

If your Verilog HDL design uses memory functions (RAM or ROM) that can be initialized with a hexadecimal file (Intel-format) initialization, you must convert these files into Verilog HDL format using the Programming Language Interface (PLI). To use the Altera-provided source code for PLI, perform the following steps:

1. Download the file **http://www.edif.org/lpmweb/convert_hex2ver.c** to your project directory.
2. Copy the following two files from the **$MGC_HOME/shared/pkgs/quickhdl/include** directory into the **/usr/maxplus2** directory:

- **$MGC_HOME/shared/pkgs/quickhdl/include/veriuser**
- **$MGC_HOME/shared/pkgs/quickhdl/include/acc_user**

Refer to the ***Mentor Graphics QuickHDL User's Reference Manual***, version 8.5-4.6i, for information on compiling the PLI application on different platforms and using the Verilog HDL PLI.

Related Topics:

- Go to the following MAX+PLUS II ACCESS^{SM} Key topics for related information:
  - Compiling Projects with MAX+PLUS II Software
  - Performing a Timing Simulation with QuickHDL Software
  - Performing a Functional Simulation with QuickHDL Pro Software

---

Performing a Functional Simulation with QuickHDL Pro Software

You can use Mentor Graphics QuickHDL Pro software to functionally simulate mixed-level schematic and VHDL

designs before compiling them with the MAX+PLUS ® II Compiler.

Refer to Mentor Graphics *Getting Started with QuickHDL Pro* page 2-1 and 3-1 for compatible design configurations.

To functionally simulate a QuickHDL at Top Level design, follow the steps in *Getting Started with QuickHDL Pro*, Chapter 2.

To functionally simulate a QuickSim II at Top Level design, go through the following steps:

1. Be sure to set up the working environment correctly, as described in Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment.
2. Create a schematic design using QuickHDL models. Refer to Creating Design Architect Schematics for Use with MAX+PLUS II Software.
3. Compile the QuickHDL model using the QuickHDL Compiler with the **-qhpro_syminfo** option. (This is done automatically for LPM functions if you choose to compile the LPM models when saving the schematic.)
4. Start Design Architect by double-clicking Button 1 on the **max_da** icon in the Design Manager tools window.
5. Choose **Open** from the File menu, then choose **Sheet** from the Open menu to open the top level schematic.
6. Select the symbol for the VHDL model and choose **Begin Edit Symbol** from the Edit menu.
7. Press Button 3 to display the the Design Architect pop-up menu. Choose Add Menu from the Other Menus menu, then choose **Set VHDL Info**. Choose Import from Entity to display the "Import Entity Info" dialog box.
8. Specify the following options in the "Import Entity Info" dialog box:

    1. *QHDL InitFile*: Specify your **quickhdl.ini** file.

    2. *Library Logical Name*: Click on **Choose Library** button and fill the "Choose VHDL Library" form with your work library.

    3. *Entity Name*: Click on **Choose Entity** button and select the name of your entity.

    4. *Default Architecture*: Click on **Choose Arch** button and select corresponding architecture for the entity.

After filling in the above information, click on **OK** to close the form.

- Check the symbol with defaults. If there are no errors, save the symbol with default registration by choosing **Save Symbol** from the File menu, then choose **Default Registration**.
- Choose **End Edit Symbol** from the Edit menu to close the Symbol Editor session. In the schematic window, select the symbol you have just edited and choose **Object** from the Report menu, then choose **All** from the Selected menu. In the report transcript, make sure the *MODEL* property is set to *qhpro* to ensure that the model will work with QuickHDL Pro.
- Select the folder for your project, press button 3, and choose **Open max2_qvpro** to start QuickHDL Pro. You can also start QuickHDL Pro by typing max2_qvpro ↵ at the UNIX prompt. In the **QVHDL Pro System** dialog box, make sure *EDDM Design* is selected for *Invoke on* and the correct path name is specified for the design. Choose **OK** to start the QuickHDL Pro. A QHPro (QuickSim II) window and a QHPro (QuickHDL) window appear on the screen.
- Use the QuickSim II window to simulate the top level schematic and the QuickHDL window to simulate the VHDL portion of the design.

Related Topics:

- Go to the following MAX+PLUS II ACCESS℠ Key topics for related information:
    - Compiling Projects with MAX+PLUS II Software

Instantiating LPM Functions in Design Architect Schematics
- Performing a Functional Simulation with QuickHDL Software

Creating Hierarchical Projects with Design Architect Software

If you wish to create a hierarchical schematic design that contains symbols representing other design files, such as AHDL Text Design Files (**.tdf**), VHDL Design Files (**.vhd**), or Verilog Design Files (**.v**), you can create a hollow-body symbol for the design file and then instantiate it in your top-level design file.

To create a hollow-body symbol for a lower-level design file, follow these steps:

1. (Optional) If you are creating a hollow-body symbol for a VHDL or Verilog HDL design file, you can first functionally simulate the VHDL or Verilog HDL file, as described in Performing a Functional Simulation with QuickHDL Software.
2. Start the Design Architect software by double-clicking Button 1 on the **max2_da** icon in the Design Manager tools window. You can also start Design Architect software by typing `max2_da` ↵ at the UNIX prompt.
3. Choose the **OPEN SYMBOL** button in the Design Architect session_palette to open the Symbol Editor. Type the lower-level design file name, including the directory path, in the *Component Name* box. Choose **OK**.
4. Create a symbol that represents the inputs and outputs of the lower-level file.
5. Assign `PINTYPE` properties of `IN` or `OUT` to the inputs and outputs of the symbol, and assign appropriate values to any other properties of the symbol so that it can be identified in the top-level schematic.

   ☞ If you are creating a hollow-body symbol for a VHDL design file, be sure to assign the value `qvpro` to the symbol's `model` property so that it can be identified as a VHDL component in the top-level schematic.

6. Check and save the symbol, then close the Symbol Editor.
7. To enter the symbol, choose the **CHOOSE SYMBOL** button from the Design Architect session_palette.
8. Select the symbol file from the Navigator menu and choose **OK**.
9. The MAX+PLUS® II software uses the Altera® provided **mnt8_bas.lmf** Library Mapping File to map Design Architect symbols to equivalent MAX+PLUS II logic functions. To use custom symbols, you must create a custom LMF that maps your custom symbols to the equivalent EDIF Input File, Text Design File (TDF), or other design file. You will also need to specify this LMF in the **EDIF Netlist Reader Settings** dialog box before compiling the design with the MAX+PLUS II software. See Compiling Projects with MAX+PLUS II Software for more information.
10. Continue with the steps necessary to complete your Design Architect schematic, as described in Creating Design Architect Schematics for Use with MAX+PLUS II Software.

Installing the Alteraprovided MAX+PLUS II/Mentor Graphics/Exemplar Logic interface on your computer automatically creates the sample hierarchical Design Architect schematic file **/usr/maxplus2/examples/mentor/example3/fulladd2**.

Synthesizing & Optimizing VHDL & Verilog HDL Projects with Galileo Extreme Software

After you have created a VHDL or Verilog HDL design, you can use Exemplar Logic's Galileo Extreme software

to synthesize and optimize your VHDL Design File (**.vhd**) or Verilog Design File (**.v**) and prepare it for compilation with the MAX+PLUS ® II Compiler.

To synthesize and optimize your project and generate an EDIF netlist file with Galileo Extreme software, go through the following steps:

1. Be sure to set up the working environment correctly, as described in Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment.
2. Create a VHDL or Verilog HDL design that follows the guidelines described in Creating VHDL & Verilog HDL Designs for Use with MAX+PLUS II Software.
3. (Optional) Use the QuickHDL software to functionally simulate the design file, as described in Performing a Functional Simulation with QuickHDL Software.
4. Select the icon for your design file in the appropriate directory, press Button 3, and choose **max2_galileo** in the Navigator window to start the Galileo Extreme software. You can also start Galileo Extreme software by typing `max2_galileo` ↵ at the UNIX prompt.
5. Specify settings for the *Filename* and *Format* options under *INPUT DESIGN*.
6. Specify settings for the *Filename*, *Format*, and *Technology* options under *OUTPUT DESIGN*. Verify that EDIF is specified in the *Format* box.
7. Choose the **Altera Output Options** button if you want to specify settings for various parameters, including *Maximum Fanin* for MAX devices and *Part Number* for FLEX devices. You can also turn on the *Run MAX+PLUS II* option for design compilation, which specifies that the MAX+PLUS II Compiler should start processing your design immediately after you run Galileo Extreme. Choose **OK** to save any setting changes.
8. Choose **Start Run**. The Galileo Extreme software generates <*design name*>**.edf** in the <*project directory*>/**max2** subdirectory and then closes, returning you to the Navigator window.
9. Process your design with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software. If you turned on the *Run MAX+PLUS II* option in step 7, the MAX+PLUS II Compiler automatically starts processing your design after you run Galileo Extreme.

Installing the Altera® provided MAX+PLUS II/Mentor Graphics/Exemplar Logic interface on your computer automatically creates the following sample VHDL Design Files:

- **/usr/maxplus2/examples/mentor/example5/count4.vhd**
- **/usr/maxplus2/examples/mentor/example6/count8.vhd**
- **/usr/maxplus2/examples/mentor/example8/adder16.vhd**

Related Topics:

- Go to the following MAX+PLUS II ACCESS℠ Key topics for related information:
  - Synthesizing & Optimizing VHDL & Verilog HDL Projects with Leonardo Software
  - Performing a Timing Simulation with QuickHDL Software
  - Performing a Timing Analysis with QuickPath Software

Synthesizing & Optimizing VHDL & Verilog HDL Projects with Leonardo Software

After you have created a VHDL or Verilog HDL design, you can use Exemplar Logic's Leonardo software to synthesize and optimize your VHDL Design File (**.vhd**) or Verilog Design File (**.v**) and prepare it for compilation with the MAX+PLUS ® II Compiler.

To synthesize and optimize your project and generate an EDIF netlist file with Leonardo software, go through the following steps:

1. Be sure to set up the working environment correctly, as described in Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment.

2. Create a VHDL or Verilog HDL design that follows the guidelines described in Creating VHDL & Verilog HDL Designs for Use with MAX+PLUS II Software.

3. (Optional) Use the QuickHDL software to functionally simulate the design file, as described in Performing a Functional Simulation with QuickHDL Software.

4. Select the icon for your project's design file from the Navigator window, press Button 3, and choose **max2_leonardo** to start the Leonardo software and open the Exemplar Logic Leonardo window. You can also start Leonardo by typing `max2_leonardo` ↵ at the UNIX prompt.

5. Click Button 1 on the **Flow Guide** toolbar button to open the **Customize Flow Guide** dialog box.

6. Turn on the *Altera EDIF Output File* checkbox under *Output Flow*.

7. Choose **Run Flow Guide** to open the Flow Guide window and specify the appropriate options in the following modules to synthesize your project:

    1. Choose **Load Library** to open the **Load Library** dialog box. If necessary, select *FPGA Enhanced* from the *Tech Type* drop-down list box. Select the target Altera® device family from the list of supported device families and choose **Load** to close the dialog box.

    2. Choose **Read** to open the **Read** dialog box. Turn on *VHDL* or *Verilog HDL* under *Format*, ensure that the appropriate library name appears under *Work Library*, and type the name of your design file in the *Filename* box or select it from the **Select a File** dialog box. Choose **Read** to close the dialog box.

    3. Choose **Pre-Optimize** to open the **Pre-Optimize** dialog box. Choose **Pre-Optimize** to accept the default pre-optimization settings and close the dialog box.

    4. Choose **Optimize** to open the **Optimize** dialog box. Choose **Optimize** to accept the default optimization settings and close the dialog box.

    5. Choose **Write Altera** to open the **Convenience Procedures** dialog box. Type `write_altera` in the *Procedure* box or select *write_altera* from the list box and choose **Run** to automatically generate *<design name>***.edf**.

    6. Choose **Exit Flow Guide** to return to the Leonardo window.

8. Process your design with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Installing the Alteraprovided MAX+PLUS II/Mentor Graphics/Exemplar Logic interface on your computer automatically creates the following sample VHDL Design Files:

- **/usr/maxplus2/examples/mentor/example5/count4.vhd**
- **/usr/maxplus2/examples/mentor/example6/count8.vhd**
- **/usr/maxplus2/examples/mentor/example8/adder16.vhd**

Related Topics:

- Go to Synthesizing & Optimizing VHDL & Verilog HDL Projects with Galileo Extreme Software in these

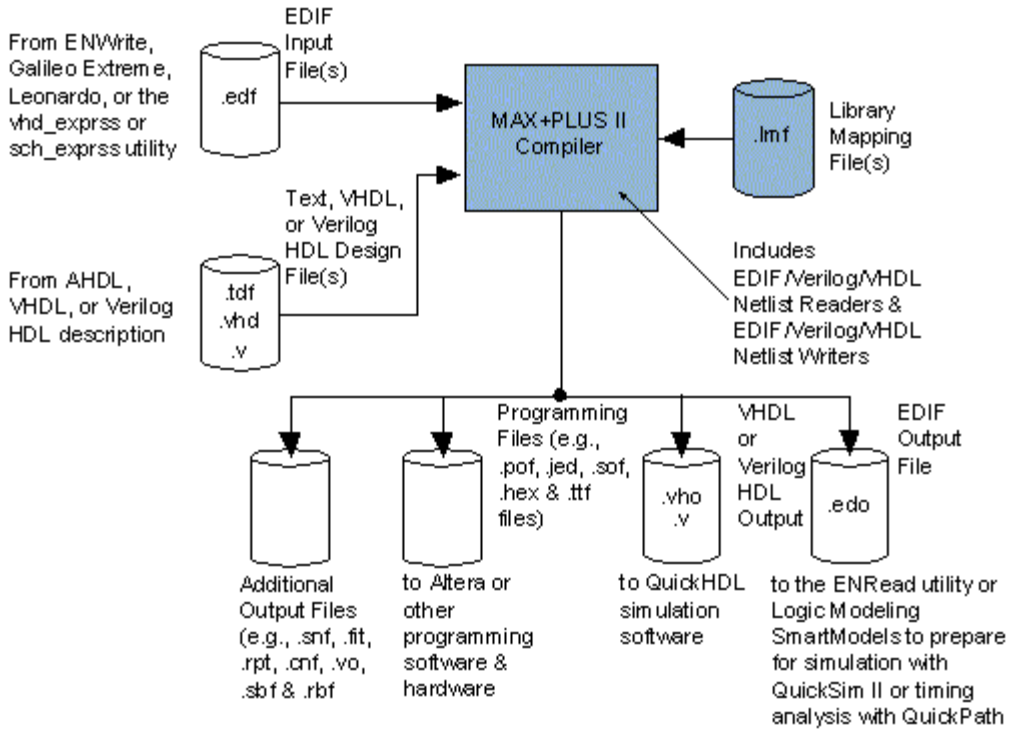
MAX+PLUS II ACCESS[SM] Key topics for related information.

Project Compilation Flow

The following figure shows the MAX+PLUS [®] II/Mentor Graphics/Exemplar Logic project compilation flow.

Figure 1. MAX+PLUS II/Mentor Graphics/Exemplar Logic Project Compilation Flow

Alteraprovided items are shown in blue.



Compiling Projects with MAX+PLUS II Software

The MAX+PLUS[®] II Compiler can process design files in a variety of formats. This topic describes how to use MAX+PLUS II software to compile projects in which the top-level design file is an EDIF Input File (with the extension **.edf**).

Related Topics:

- Go to MAX+PLUS II Help for information on compiling VHDL and Verilog HDL, design files directly with the MAX+PLUS II Compiler.
- Go to Running Synopsys Compilers from MAX+PLUS II Software for information on running the Synopsys Design Compiler or FPGA Compiler software on a VHDL or Verilog HDL design from within the MAX+PLUS II Compiler window.

To compile a design (also called a "project") with MAX+PLUS II software, go through the following steps:

1. Create design files that are compatible with the MAX+PLUS II software and convert them into EDIF Input Files with the extension **.edf**. Specific instructions for some tools are described in these MAX+PLUS II ACCESS^{SM} Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your design entry or synthesis and optimization tool.

2. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, create a mapping for each function in a Library Mapping File (**.lmf**) to map the custom symbol to the corresponding EDIF Input File, AHDL Text Design File (**.tdf**), or other MAX+PLUS II-supported design file. These custom functions are represented in design files as hollow-body symbols or "black box" HDL descriptions.

   ☞ Go to "Library Mapping Files (**.lmf**)" in MAX+PLUS II Help for more information.

3. Open MAX+PLUS II and specify the name of your top-level design file as the project name with the **Project Name** command (File menu). If you open an HDL file in the MAX+PLUS II Text Editor, you can choose the **Project Set Project to Current File** command (File menu) instead.

   ☞ You can also compile a project from a command line. However, the first time you compile a project, the settings you need to specify are easier to specify from within the MAX+PLUS II software. After you have run the graphical user interface for the MAX+PLUS II software at least once, you can more easily use the command-line **setacf** utility to modify options in the Assignment & Configuration File (**.acf**) for the project. Type `setacf -h` ↵ and `maxplus2 -h` ↵ for descriptions of **setacf** and MAX+PLUS II command-line syntax.

4. Choose **Device** (Assign menu) and select the target Altera device family in the *Device Family* drop-down list box. If you wish to implement the design logic in a specific device, select it in the *Devices* box. Otherwise, select *AUTO* to allow the MAX+PLUS II Compiler to choose the best device(s) in the current device family. If your design entry or synthesis and optimization tool required you to specify a target family and/or device, specify the same information in this dialog box. For information on partitioning logic among multiple devices, go to MAX+PLUS II Help. Choose **OK**.

5. Open the Compiler window by choosing the **Compiler** command (MAX+PLUS II menu). Go through the following steps to specify the options necessary to compile the design file(s) in your project:

   1. Ensure that all EDIF netlist files have the extension **.edf** and choose **EDIF Netlist Reader Settings** (Interfaces menu).

   2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor. This name should be the name of the vendor whose tool(s) you used to create the EDIF netlist files. If your vendor name does not appear, select *Custom* instead.

      ☞ If you are compiling a design created with Synopsys FPGA Express software, select *Synopsys*, choose the **Customize** button, enter <*project name*>**.lmf** in the *LMF #1* box, choose **OK**, and skip to step 6.

   3. If you selected an existing vendor name in the *Vendor* box and your project contains design files that require custom LMF mappings, choose the **Customize** button to expand the dialog box to show all settings. Turn on the *LMF #2* checkbox and type your custom LMF's filename in the corresponding text box, or select a name from the *Files* box. The selection in the *Vendor* box will change to *Custom* and all settings will be retained until you change them again.

   4. If you selected *Custom* in the *Vendor* box, choose the **Customize** button to expand the dialog box to show all settings. Any previously defined custom settings will be displayed. Under *Signal Names*, type one or more names with up to 20 total name characters in the *VCC* or *GND* box if your EDIF Input File(s) use one or more names other than `VCC` or `GND` for the global high or low signals. Multiple signal

names must be separated by either a comma (,) or a space. Under *Library Mapping Files*, turn on the *LMF #1* checkbox and type a filename in the text box following it, or select a name from the *Files* box. If necessary, specify another LMF name in the *LMF #2* box. Go to MAX+PLUS II Help for detailed information on the settings available in the **EDIF Netlist Reader Settings** dialog box.

5. Choose **OK**.

6. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, you may need to ensure that all files are present in your project directory, i.e., the same directory as the top-level design file. Otherwise, you must specify the directories containing these files as user libraries with the **User Libraries** command (Options menu).

7. Follow all guidelines that apply to your design entry or synthesis and optimization tool:

   - Exemplar Logic Galileo Extreme-Specific Compiler Settings
   - Synopsys DesignWare-Specific Compiler Settings
   - Converting Synopsys FPGA Compiler & Design Compiler Timing Constraints into MAX+PLUS II-Compatible Format with the **syn2acf** Utility
   - Synplicity Synplify-Specific Compiler Settings

8. If you wish to generate EDIF, VHDL, or Verilog HDL output files for post-compilation simulation or timing analysis with another EDA tool, go through the following steps:

   1. (Optional) Turn on the **Optimize Timing SNF** command (Processing menu) to reduce the size of the output file(s). Turning on this command can reduce the size of output netlists by up to 30%.

      This command does not create optimized timing SNFs on UNIX workstations. However, a non-optimized timing SNF provides the same functional and timing information as an optimized timing SNF.

   2. If you wish to generate EDIF Output Files (**.edo**), go through these steps:

      1. Turn on the **EDIF Netlist Writer** command (Interfaces menu). Then choose the **EDIF Netlist Writer Settings** command (Interfaces menu).

      2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor and choose **OK**. If your vendor name does not appear, select *Custom* instead and specify the settings that are appropriate for your simulation or timing analysis tool. Go to MAX+PLUS II Help for detailed information on the options available in the **EDIF Netlist Writer Settings** dialog box.

      3. To generate an optional Standard Delay Format (SDF) Output File (**.sdo**), choose the **Customize** button to expand the dialog box to show all settings. Select one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**.

      The filenames of the EDIF Output File(s) and optional SDF Output File(s) are the same as the user-defined chip name(s) for the project; if no chip names exist, the Compiler assigns filenames that are based on the project name. For a multi-device project, the Compiler also generates a top-level EDIF Output File that is uniquely identified by "**_t**" appended to the project name. In addition, the Compiler automatically generates a VHDL Memory Model Output File, *<project name>***.vmo**, when it generates an EDIF Output File that contains memory (RAM or ROM).

   3. If you wish to generate VHDL Output Files (**.vho**), turn on the **VHDL Netlist Writer** command (Interfaces menu). Then choose **VHDL Netlist Writer Settings** command (Interfaces menu). Select *VHDL Output File (.vho)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF ver. 2.1 files contain timing delay information that allows you to perform back-annotation simulation in VHDL with VITAL-compliant simulation libraries. The VHDL Output Files

generated by the Compiler have the extension **.vho**, but are otherwise named in the same way as the EDIF Output Files described above.

4. If you wish to generate Verilog HDL Output Files (**.vo**), turn on the **Verilog Netlist Writer** command (Interfaces menu). Then choose **Verilog Netlist Writer Settings** command (Interfaces menu). Select *Verilog Output File (.vo)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF Output Files contain timing delay information that allows you to perform back-annotation simulation in Verilog HDL. The Verilog Output Files generated by the Compiler have the extension **.vo**, but are otherwise named in the same way as the EDIF Output Files described above.

9. To run the MAX+PLUS II Compiler, choose the **Project Save & Compile** command (File menu) or choose the **Start** button in the Compiler window.

   ☞ See step 3 for information on running MAX+PLUS II software from the command line.

10. Once you have compiled the project with the MAX+PLUS II Compiler, you can use the VHDL, Verilog HDL, or EDIF output file(s), and the optional SDF Output File(s) (**.sdo**) to perform timing analysis or timing simulation with another EDA tool. Specific instructions for some tools are described in these MAX+PLUS II ACCESS Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your EDA tool.

The MAX+PLUS II Compiler also generates a Report File (**.rpt**), a Pin-Out File (**.pin**), and one or more of the following files for device programming or configuration:

- JEDEC Files (**.jed**)
- Programmer Object Files (**.pof**)
- SRAM Object Files (**.sof**)
- Hexadecimal (Intel-format) Files (**.hex**)
- Tabular Text Files (**.ttf**)

Related Topics:

- Go to *Compiler Procedures* in MAX+PLUS II Help for information on other available Compiler settings.
- Go to *Programmer Procedures* in MAX+PLUS II Help for instructions on creating other types of programming files and on programming or configuring Altera devices.
- Go to Back-Annotating MAX+PLUS II Pin Assignments to Design Architect Symbols for information on back-annotating pin assignments in Mentor Graphics Design Architect schematics.
- Go to Programming Altera Devices for information on the different programming hardware options for Altera device families.

- Go to the following topics, which are available on the web, for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware

Exemplar Logic Galileo ExtremeSpecific Compiler Settings

If you are using MAX+PLUS ® II software to compile a FLEX ® design that was created with Galileo Extreme software, go through the following additional compilation steps:

1. Choose **Global Project Logic Synthesis** (Assign menu) to open the **Global Project Logic Synthesis** dialog box.

2. Select the appropriate logic synthesis style under *Global Project Logic Synthesis Style*:

✔ If you turned on the *Lock Lcells* option under *SYNTHESIS SWITCHES* in the Galileo Extreme **Altera FLEX Output Options** dialog box when synthesizing your design with Galileo Extreme software, select *WYSIWIG* in the *Global Project Synthesis Style* box.

or:

✔ If you did not turn on the *Lock Lcells* option, select *FAST* in the *Global Project Synthesis Style* box.

3.

(Optional) Turning on one or more of the following options may help to improve area usage and timing delays:

- *Automatic Fast I/O*
- *Automatic Register Packing*
- (FLEX 10K devices only) *Automatic Implement in EAB*

•

Choose **OK** to close the **Global Project Logic Synthesis** dialog box.

• Continue with the steps necessary to compile your project, as described in Compiling Projects with MAX+PLUS II Software.

Related Topics:

- Go to the following topics, which are available on the web, for additional information:
  - FLEX Devices
  - MAX® Devices
  - Classic Device Family

Using the Altera Schematic Express (sch_exprss) Utility

Once you have created a Design Architect schematic, you can use the Altera Schematic Express utility (**sch_exprss**) to generate a Design Viewpoint Editor (DVE) viewpoint and an EDIF netlist file from the schematic; process the EDIF Input File (**.edf**) with the MAX+PLUS ® II software to generate an EDIF Output File (**.edo**); process the EDIF Output File with ENRead and DVE software; and generate an **altera_asim** viewpoint for simulation. The **sch_exprss** utility creates all necessary subdirectories and copies all of the files to the correct locations.

To use the **sch_exprss** utility, follow these steps:

1. Be sure to set up the working environment correctly, as described in Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment.

2. Create a Design Architect schematic that follows the guidelines described in Creating Design Architect Schematics for Use with MAX+PLUS II Software.

3. Select your project's folder, press Button 3, and choose **Open sch_exprss** from the Mentor Graphics Navigator window to start the Altera Schematic Express tool.

4. Specify settings for the *Input Schematic*, *Altera Device Family*, *MAX+PLUS II Synthesis Style*, *Process Direction*, and *Verbose* options in the **sch_exprss** dialog box and choose **OK** to generate the **altera_asim** file for simulation with QuickSim II software.

5. If necessary, correct any errors in the Design Architect schematic design file and recompile the project. The **sch_exprss** utility generates the **altera_asim** viewpoint in the appropriate directory.

6. Simulate your project, as described in Performing a Timing Simulation with DVE & QuickSim II Software.

Related Topics:

- Go to Performing a Timing Analysis with QuickPath Software in these MAX+PLUS II ACCESS℠ Key topics for related information.

Using the Altera VHDL Express (vhd_exprss) Utility

Once you have created a VHDL Design File (**.vhd**) for your project, you can use the Altera ® VHDL Express (**vhd_exprss**) utility to synthesize and optimize the design and generate an EDIF netlist file with Galileo Extreme software; process the EDIF netlist file with the MAX+PLUS II software to generate a VHDL Output File (**.vho**); and prepare the VHDL Output File for simulation with QuickHDL software. The **vhd_exprss** utility creates all necessary subdirectories and copies all files to the correct locations.

To use the **vhd_exprss** utility, follow these steps:

1. Be sure to set up the working environment correctly, as described in Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment.

2. Create a VHDL Design File that follows the guidelines described in Creating VHDL & Verilog HDL Designs for Use with MAX+PLUS II Software.

3. Select the VHDL Design File for your project, press Button 3, and choose **Open vhd_exprss** from the Navigator window to start the Altera VHDL Express tool.

4. Specify settings for the *Input HDL File*, *Altera Device Family*, *Max2 Synthesis Style*, *Process Direction*, and *Verbose* options, and the *Optimize* and *Effort* runtime options, in the **vhd_exprss** dialog box, and choose **OK**.

5. If necessary, correct any errors in the VHDL Design File and recompile the project. The **vhd_exprss** utility generates a VHDL output file in the appropriate directory.

6. Simulate your project, as described in Performing a Timing Simulation with QuickHDL Software.

Related Topics:

- Go to Performing a Timing Analysis with QuickPath Software in these MAX+PLUS II ACCESS℠ Key topics for related information.

BackAnnotating MAX+PLUS II Pin Assignments to Design Architect Symbols

The MAX+PLUS ® II/Mentor Graphics software interface includes the **annotate_pin** utility. This utility allows you to back-annotate the pin assignments from the MAX+PLUS II-generated Fit File (**.fit**) back to the symbol for the design file. The **annotate_pin** utility has the following syntax:

`annotate_pin [-p` <*property name*>`]` <*symbol name*> <*chip name*> <*Fit File name*> ↵

where <*property name*> is the default name for the pin assignment (default is `PIN_NO`), <*symbol name*> is the

pathname of the directory that contains the symbol, <*chip name*> is the chip name specified in the Fit File, and <*Fit File name*> is the name of the Fit File that contains the pin assignment information for back-annotation. If the <*property name*> is not found at a pin number, that pin will not be back-annotated. If the <*chip name*> is not found in the Fit File, the **annotate_pin** utility stops the back-annotation process.

For example:

```
annotate_pin -p PIN_NO /usr/examples/decode decode decode.fit ↵
```

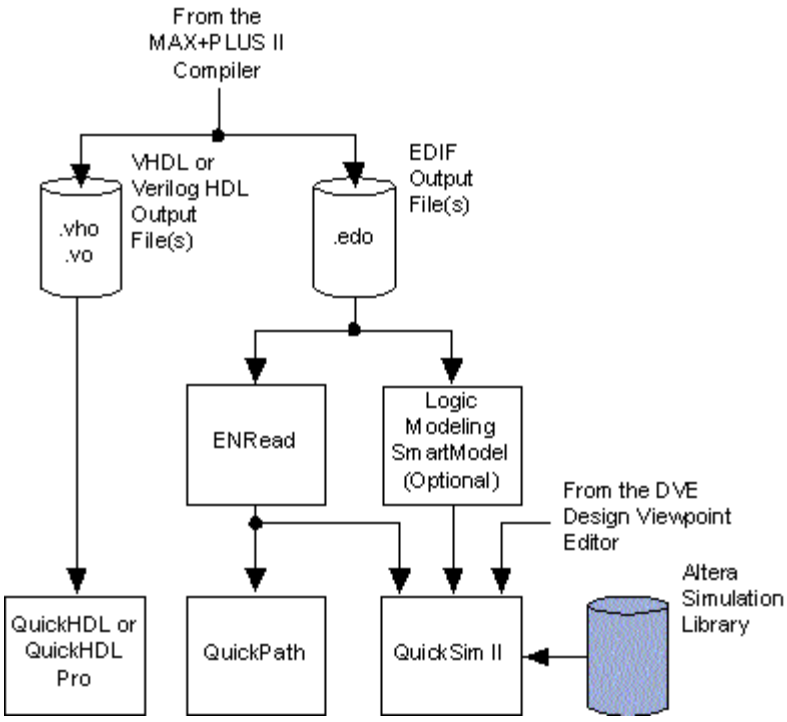
☞ Type `annotate_pin -h ↵` at the UNIX prompt to display information on how to use this utility.

Project Simulation/Timing Analysis Flow

The following figure shows the project simulation and timing analysis flow for the MAX+PLUS® II /Mentor Graphics interface.

Figure 1. MAX+PLUS II/Mentor Graphics Project Simulation/Timing Analysis Flow

Alteraprovided items are shown in blue.



Initializing Registers in VHDL & Verilog Output Files for Power-Up before Simulation

Altera provides the **add_dc** script, which is availiable in the MAX+PLUS II system directory, to allow you to process MAX+PLUS II-generated Verilog Output Files (**.vo**) and VHDL Output Files (**.vho**) to prepare these files for simulation with another EDA tool. The add_dc script runs the **add_dclr** utility, which inserts a device_clear signal that is used for power-up initialization of all registers or flipflops in the design.

The script adds in a top-level signal named device_clear and connects it to the CLRNpin in all flipflops that should initialize to 0, and to the PRN pin of all flipflops that should initialize to 1. If the CLRNor PRN pin of a flipflop is already being used (i.e., is already connected to a signal), the script modifies the Verilog Output File or VHDL Output File so that the ANDof the original signal and the device_clear pin feed the CLRNor PRN pin.

To use the **add_dc** script to process Verilog Output Files and VHDL Output Files before simulation with another EDA tool, follow these steps:

1. Make sure that your design file is located in the current directory, or change to the directory in which the design file is located.
2. Type the following command at the command prompt:

   Â¥<*path name of* ***add_dc.bat*** *file*>Â¥add_dc <*design name*> <*path name of* ***add_dclr.exe*** *file*> ↵

For example, if the both the **add_dc.bat** and the **add_dclr.exe** files are located in the **d:Â¥maxplus2Â¥exew** directory, and the **d:Â¥maxplus2Â¥exew** directory is specified in the search path, you can type the following command at a command prompt to add a device_clear signal to a design named myfifo in the file **myfifo.vo**:

add_dc myfifo d:Â¥maxplus2Â¥exew ↵

1. The **add_dc** script gives a message if the directory contains both a VHDL Output File and a Verilog Output File with the same name (<*design name*>**.vo** and <*design*>**.vho**). You should delete or rename whichever of those files should not have the `device_clear` signal added. The **add_dc** script can modify only one design file at a time.

2. When the **add_dc** script processes the Verilog Output File or VHDL Output File, it creates a backup copy of the original file, with the extension **.ori**.
3. The **add_dc** script works only for Verilog Output Files and VHDL Output Files that are generated by MAX+PLUS II.

After you have used the **add_dc** script and are ready to simulate the resulting Verilog Output File or VHDL Output File with another EDA tool, you should assert the active low device_clear pin for a period of time that is long enough for the design to initialize. You can then de-assert the pin, and apply simulation vectors to the design.

Performing a Timing Simulation with DVE & QuickSim II Software

After you have compiled a design with the MAX+PLUS® II Compiler, you can prepare the MAX+PLUS IIgenerated EDIF Output File (**.edo**) with Mentor Graphics Design Viewpoint Editor (DVE) and simulate it with the Mentor Graphics QuickSim II software.

To simulate an EDIF Output File with the QuickSim II software, follow these steps:

1. Be sure to set up the working environment correctly, as described in Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment.
2. Generate an EDIF Output File for your project, as described in Compiling Projects with MAX+PLUS II Software or Using the Altera Schematic Express (**sch_exprss**) Utility.
3. If you used the Altera Schematic Express (**sch_exprss**) utility to process your design, skip to step 5. Otherwise, go to step 4.
4. In the Navigator window, select your project's icon, press Button 3, and choose **Open max2_enr** to read your

project's EDIF Output File with the ENRead utility. You can also start ENRead software by typing `max2_enr` ↵ at the UNIX prompt.

5. Select your project's folder, press Button 3, and choose **Open max2_ave** to open DVE, which will prepare your project's simulation component for QuickSim II timing simulation. DVE automatically generates an appropriately named viewpoint for your project. You can also start DVE by typing `max2_ave` ↵ at the UNIX prompt.

6. Select your project's folder, press Button 3, and choose **Open max2_qsim** to simulate your project and its DVE viewpoint with QuickSim II software. You can also start QuickSim II by typing `max2_qsim` ↵ at the UNIX prompt.

7. In the **Altera QuickSim** dialog box, type the name of your project's viewpoint in the *Viewpoint Name* box. Select *Timing* as the *Timing Mode*. Select the *Max timing* option. Choose *Scale Factor* for *Delay Scale*, and be sure that `0.1` is specified for the *Value*. Choose **OK**.

   ☞ If the delay scale value is not set to 0.1 (i.e., divided by ten), the QuickSim II software will not reflect the correct timing simulation values.

Related Topics:

- Go to Performing a Timing Analysis with QuickPath Software in these MAX+PLUS II ACCESS℠ Key topics for related information.

Preparing EDIF Output Files for Timing Simulation with Logic Modeling SmartModel Software

Once you have generated an EDIF Output File (**.edo**) for a design with the MAX+PLUS® II Compiler, you can use Logic Modeling SmartModel software to prepare it for simulation.

To use Logic Modeling SmartModel software, follow these steps:

1. Be sure to set up the working environment correctly, as described in Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment.

2. Generate an EDIF Output File for your project, as described in Compiling Projects with MAX+PLUS II Software.

3. In the Navigator window, select your project's icon, press Button 3, and choose **Open max2_lmi** to start the Logic Modeling EDIF2SCF compiler and create a SmartModel Configuration Format File, as described in Creating a Logic Modeling SmartModel Configuration Format File.

4. Create a schematic, as described in Creating Design Architect Schematics for Use with MAX+PLUS II Software, that contains an instance of a Logic Modeling SmartModel for the Altera® device specified in your project:

   1. Start Design Architect by double-clicking on your project's icon in the Design Manager tools window.

   2. Choose **Altera Libraries** (Libraries menu) and choose **LOGIC MODELING** (Altera Libraries menu). Select the appropriate device model for the Altera device specified in your project.

   3. Change the `PLDFILE` property for the Altera device model to the full pathname of the SmartModel Configuration Format File generated in step 2.

4. Add pins to the device model.
5. Check and save the schematic in your project directory.

5.

Select the icon for the folder representing the schematic generated in step 3, press Button 3, and choose **Open max2_lve** to open DVE and prepare your EDIF Output File for QuickSim II timing simulation. DVE generates a viewpoint named **altera_lsim**.

6. Select the icon for the folder representing the schematic generated in steps 2a through 2e, press Button 3, and perform a timing simulation on your project, as described in steps 5 and 6 of Performing a Timing Simulation with DVE & QuickSim II Software.

Creating a Logic Modeling SmartModel Configuration Format File

The Logic Modeling SmartModel EDIF2SCF for Windows Compiler reads the <*project name*>**.edo** file generated by the MAX+PLUS® II Compiler and creates the <*project name*>**.scf** file. The EDIF2SCF compiler also extracts all state and internal net information for simulation of internal nodes (with the x and w options).

When the EDIF2SCF compiler opens, an Options prompt appears, providing options for the *help_type*, *netlist_type*, *design_name*, *window_file*, *output_file*, and *interface_file*. If some fields are not visible, press the TAB key to cycle through all six fields. The information you enter in these fields configures the EDIF2SCF compiler, and is equivalent to setting the various commandline options when compiling with EDIF2SCF.

Refer to the following table when setting options at the EDIF2SCF compiler's Options prompt. For information on these options, refer to *EDIF2SCF Compiler* in the Logic Modeling ***SmartModel Library Reference Manual***.

Table 1. EDIF2SCF Options for Logic Modeling SmartModel

Setting(s)	Effect on EDIF2SCF
help_type option is turned on	Compiles with the h option
netlist_type option is set to *extract*	Compiles with the x option
netlist_type option is set to *windows*	Compiles with no options
netlist_type option is set to *windows* and the *window_file* option is set to the pathname of the windows definition file	Compiles with the w option
netlist_type option is set to *windows* and the *output_file* option is set to the pathname of a <*filename*>**.scf** file	Compiles with the o option
netlist_type option is set to *windows* and the *interface_file* option is set to the pathname of a custom <*filename*>**.inf** file	Compiles with the i option

---

Performing a Timing Simulation with QuickHDL Software

After you have entered a VHDL or Verilog HDL design file and compiled it with the MAX+PLUS ® II Compiler, you can use Mentor Graphics QuickHDL software to simulate the MAX+PLUS IIgenerated VHDL Output File (**.vhd**) or Verilog Output File (**.vo**) and the Standard Delay Format (SDF) Output File (**.sdo**).

To simulate your VHDL or Verilog HDL design, go through the following steps:

1. Be sure to set up the working environment correctly, as described in Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment.

2. Generate a VHDL or Verilog HDL output file and an SDF output file for your project, as described in Compiling Projects with MAX+PLUS II Software.

3. Change to your project's directory.

4. Copy your **quickhdl.ini** file to the same directory as your VHDL or Verilog HDL file.

5. Type the following sets of commands at the UNIX prompt to create the work library and compile your project's VHDL or Verilog HDL output file:

VHDL:	**Verilog HDL:**
`setenv MGC_WD 'pwd'` ↵	`setenv MGC_WD 'pwd'` ↵
`qhlib work` ↵	`qhlib work` ↵
`qvhcom` <*project name*>`.vho` ↵	`qvlcom` <*project name*>`.vo` ↵

6. Type `qhsim -sdftyp` <*project name*>`.sdo` ↵ at the UNIX prompt to perform timing back-annotation and simulation and to display the QuickHDL simulation window.

If your Verilog HDL design uses memory functions (RAM or ROM) that can be initialized with a hexadecimal file (Intel-format) initialization, you must convert these files into Verilog HDL format using the Programming Language Interface (PLI). To use the Altera-provided source code for PLI, perform the following steps:

1. Download the file **http://www.edif.org/lpmweb/convert_hex2ver.c** to your project directory.

2. Copy the following two files from the **$MGC_HOME/shared/pkgs/quickhdl/include** directory into the **/usr/maxplus2** directory:

   - **$MGC_HOME/shared/pkgs/quickhdl/include/veriuser**
   - **$MGC_HOME/shared/pkgs/quickhdl/include/acc_user**

Refer to the ***Mentor Graphics QuickHDL User's Reference Manual***, version 8.5-4.6i, for information on compiling the PLI application on different platforms and using the Verilog HDL PLI.

Related Topics:

- Go to Performing a Functional Simulation with QuickHDL Software in these MAX+PLUS II ACCESS℠ Key topics for related information.

Performing a Timing Analysis with QuickPath Software

After you have compiled your project with the MAX+PLUS ® II Compiler and generated an EDIF Output File (**.edo**), you can use Mentor Graphics QuickPath software to perform a timing analysis of your project.

To perform a timing analysis with QuickPath software, follow these steps:

1. Be sure to set up the working environment correctly, as described in Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment.

2. Generate an EDIF Output File for your project using one of the following methods:

- Compiling Projects with MAX+PLUS II Software
- Using the Altera Schematic Express (**sch_exprss**) Utility
- Using the Altera VHDL Express (**vhd_exprss**) Utility

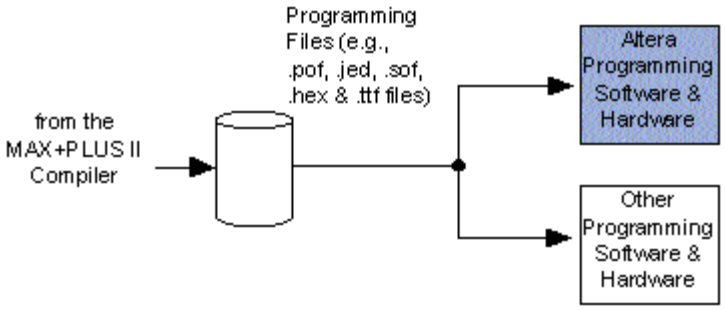
3. Select your project's folder from the **ALTERA** directory, press Button 3, and choose **Open max2_qpath** to start the QuickPath software. You can also start the QuickPath software by typing `max2_qpath` ↵ at the UNIX prompt.

Programming Altera Devices

Once you have successfully compiled and simulated a project with the MAX+PLUS® II software, you can program an Altera® device and test it in the target circuit. Figure 1 shows the device programming flow for MAX+PLUS II software.

Figure 1. MAX+PLUS II Device Programming Flow

Altera-provided items are shown in blue.



You can program devices with Altera programming hardware and MAX+PLUS II Programmer software installed on a 486- or Pentium-based PC or a UNIX workstation, or with programming hardware and software available from other manufacturers. Table 1 shows the available Altera programming hardware options on PCs and UNIX workstations.

Table 1. Altera Programming Hardware

Programming Hardware Option	UNIX PCs	Work-stations	MAX® 3000A Devices	Classic® & MAX 5000 Devices	MAX 7000 & MAX 7000E Devices	MAX 7000A, MAX 7000AE, MAX 7000B, MAX 7000S MAX 9000 & MAX 9000A Devices	FLEX® 6000, FLEX 6000A, FLEX 8000, FLEX 10K, FLEX 10KA, FLEX 10KB, & FLEX 10KE Devices	In-System Programming/ Configuration
Logic Programmer								

card, PL-MPU Master Programming Unit, and device-specific adapters	✓		✓	✓	✓	✓		
BitBlaster™ Download Cable	✓	✓	✓			✓	✓	✓
ByteBlasterMV™ Download Cable	✓		✓			✓	✓	✓
MasterBlaster™ Download Cable	✓	✓	✓			✓	✓	✓

If you wish to transfer programming files from a UNIX workstation to a PC over a network with File Transfer Protocol (FTP) or other similar transfer programs, be sure to select binary transfer mode.

Programming hardware from other manufacturers varies, but typically consists of a device connected to one of the serial ports on the workstation. Various vendors, such as Data I/O and BP Microsystems, supply hardware and software for programming Altera devices.

Related Topics:

- Go to Compiling Projects with MAX+PLUS II Software for information on creating programming files.
- Go to the following topics, which are available on the web, for additional information:
    - MAX+PLUS II Development Software
    - Altera Programming Hardware
    - FLEX Devices
    - MAX Devices
    - Classic Device Family

Mentor Graphics Project Directory Structure

Design Architect software generates the following files for each schematic:

- <*drawing name*>/**mgc_component.attr**
- <*drawing name*>/**part.Eddm_part.attr**
- <*drawing name*>/**part.part_1**
- <*drawing name*>/**schematic.mgc_schematic.attr**
- <*drawing name*>/**schematic/schem_id**
- <*drawing name*>/**schematic/sheet1.mgc_sheet.attr**
- <*drawing name*>/**schematic/sheet1.sgfx_1**
- <*drawing name*>/**schematic/sheet1.ssht_1**

The files generated for each schematic are stored in the schematic's <*drawing name*> directory and should not be edited. Mentor Graphics software automatically manages file storage and retrieval operations through this <*drawing name*> directory structure, which does not reflect hierarchical design relationships. Figure 1 shows a sample file structure with **project1** as the UNIX project directory, and **design1**, **subdesign1**, and **subdesign2** as the directories for the top-level design and subdesigns of the project.

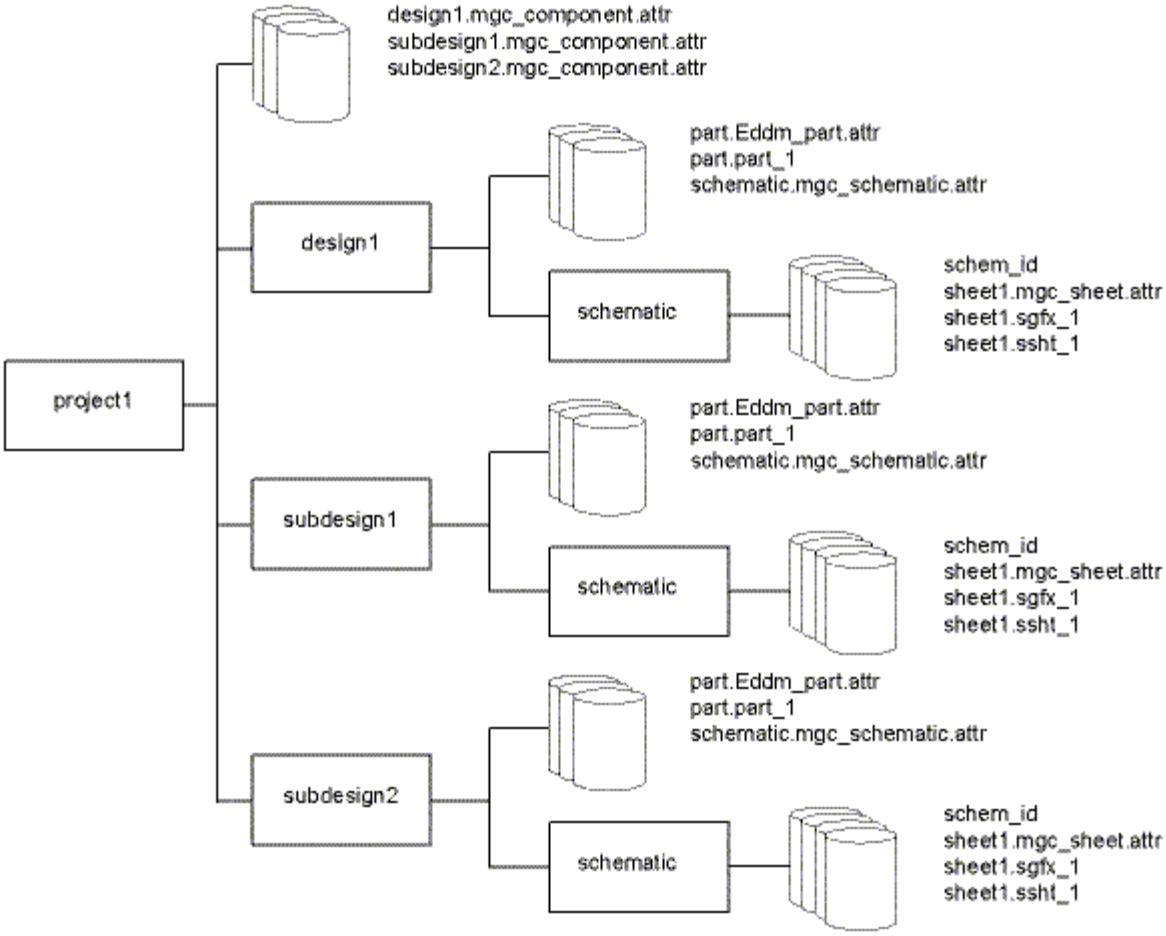


Figure 1. Design Architect Project File Structure

When the ENWrite utility converts the schematic into an EDIF netlist file, it processes the design information and all related file subdirectories, then creates the EDIF netlist file in the directory defined by the user. The EDIF netlist file is named <*project name*>**.edf**, where <*project name*> is the name of the top-level design file. The <*project*

name>**.edf** file is automatically moved to the **max2** directory under the project directory.

Related Links:

- Go to the following MAX+PLUS II ACCESS℠ Key topics for related information:
- Local Work Area Directory Structure
- MAX+PLUS II Project Directory Structure

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Using Mentor Graphics & Exemplar Logic Tools with MAX+PLUS II Software

The following topics describe how to use a variety of Mentor Graphics and Exemplar Logic tools as part of a complete design flow that includes the MAX+PLUS® II software. If you use only one Mentor Graphics or Exemplar Logic tool, click List by Tool and select the tool name to view the list of topics only for that tool. Click on one of the following topics for information:

Open a printable version of all topics listed on this page.

Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment

- Software Requirements
- Altera-Provided Logic & Symbol Libraries
- Local Work Area Directory Structure
- Mentor Graphics Project Directory Structure
- MAX+PLUS II Project Directory Structure
- MAX+PLUS II/Mentor Graphics/Exemplar Logic Interface File Organization

Design Flow For All Mentor Graphics/Exemplar Logic Tools

Design Entry

- **Design Entry Flow**

- **Design Architect**

  - Creating Design Architect Schematics for Use with MAX+PLUS II Software
    - Instantiating the clklock Megafunction in Design Architect Schematics
    - Instantiating LPM Functions in Design Architect Schematics
  - Entering Resource Assignments
    - Assigning Pins, Logic Cells & Chips
    - Assigning Cliques
    - Assigning Logic Options
    - Modifying the Assignment & Configuration File with the **setacf** Utility
    - BackAnnotating MAX+PLUS II Pin Assignments to Design Architect Symbols
  - Creating Hierarchical Projects with Design Architect Software
  - Performing a Functional Simulation with DVE & QuickSim II Software
  - Performing a Functional Simulation with QuickHDL Pro Software
  - Converting Design Architect Schematics into MAX+PLUS II-Compatible EDIF Netlist Files with the ENWrite Utility

- **VHDL & Verilog HDL**
    - Creating VHDL & Verilog HDL Designs for Use with MAX+PLUS II Software
        - Instantiating the clklock Megafunction in VHDL or Verilog HDL
        - Instantiating LPM Functions in VHDL
    - Entering Resource Assignments
        - Modifying the Assignment & Configuration File with the **setacf** Utility
    - Performing a Functional Simulation with QuickHDL Software
    - Performing a Functional Simulation with QuickHDL Pro Software
    - Creating Hierarchical Projects with Design Architect Software

Synthesis & Optimization

- Synthesizing & Optimizing VHDL & Verilog HDL Projects with Galileo Extreme Software
- Synthesizing & Optimizing VHDL & Verilog HDL Projects with Leonardo Software

Compilation

- Project Compilation Flow
- Compiling Projects with MAX+PLUS II Software
    - Exemplar Logic Galileo ExtremeSpecific Compiler Settings
- Using the Altera Schematic Express (**sch_exprss**) Utility
- Using the Altera VHDL Express (**vhd_exprss**) Utility

BackAnnotation

- BackAnnotating MAX+PLUS II Pin Assignments to Design Architect Symbols

Simulation/Timing Analysis

- Project Simulation/Timing Analysis Flow
- Initializing Registers in VHDL & Verilog Output Files for Power-Up before Simulation
- Performing a Timing Simulation with DVE & QuickSim II Software
- Performing a Timing Simulation with QuickHDL Software
- Performing a Timing Analysis with QuickPath Software

Device Programming

- Programming Altera Devices

Related Links:

- MAX+PLUS II Development Software
- Altera Programming Hardware
- Mentor Graphics web site (http://www.mentor.com)

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Using Viewlogic MOTIVE & MOTIVE for Powerview Software with MAX+PLUS II Software

The following topics describe how to use the Viewlogic MOTIVE and MOTIVE for Powerview software with MAX+PLUS® II software. Choose one of the following topics for information:

Open a printable version of all topics listed on this page.

Setting Up the MAX+PLUS II/Viewlogic Powerview Working Environment

- Software Requirements
- MAX+PLUS II/Viewlogic Powerview Interface File Organization
- MAX+PLUS II/Viewlogic Powerview Project File Structure

Timing Verification

- Timing Verification Flow
- Performing Timing Verification of EDIF Output Files (.edo) with MOTIVE & MOTIVE for Powerview Software
- Performing Timing Verification of Verilog Output Files (.vo) with MOTIVE Software

Related Links

- Viewlogic Powerview Graphical User Interface & the Altera Toolbox
- Powerview Command-Line Syntax
- Compiling Projects with MAX+PLUS II Software
- Programming AlteraÂ® Devices
- MAX+PLUS II Development Software
- Altera Programming Hardware
- Viewlogic web site (http://www.viewlogic.com)

Performing Timing Verification of EDIF Output Files (.edo) with MOTIVE & MOTIVE for Powerview Software

After you have compiled a project and generated an EDIF Output File (**.edo**) with the MAX+PLUS® II software, you can use Viewlogic MOTIVE or MOTIVE for Powerview software to perform timing verification. The **max2_MOTIVE** tool is located in both the Altera® Toolbox Design Tools Drawer and the Altera Toolbox Max2 Express Drawer. The MOTIVE timing model library, **motive.lib**, provides models of basic primitives and the clklock megafunction for timing verification.

To perform timing verification for EDIF Output Files with MOTIVE or MOTIVE for Powerview software, follow these steps:

1. Set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Viewlogic Powerview Working Environment.
2. Generate an EDIF Output File (**.edo**) by compiling your design with the MAX+PLUS II software, as described in Compiling Projects with MAX+PLUS II Software.
3. Start the MOTIVE for Powerview software by double-clicking Button 1 on the **max2_MOTIVE** icon in the Altera Toolbox Design Tools Drawer. The MOTIVE for Powerview Control Panel opens.
4. Choose **Setup Environment** (File menu) to open the **Environment Parameters** dialog box, and specify the following options:
    1. Specify the directory for the *Project Directory* option.
    2. Specify */usr/maxplus2/vwlogic/library/alt_time/motive.lib* for the *Model Library Search Path* option.
    3. Select *EDIF* for the *Netlist Input Format* option.
    4. Choose **Accept**. The MOTIVE for Powerview software automatically creates a **tim** subdirectory, which contains MOTIVE design cases and related files, in the current working directory.
5. Choose **Save Parameters** (File menu) to save your customized project setup.
6. To specify the project name, choose the **New Design** button to open the **Adding a New Design** dialog box. Type the design name in the *New Design* box. Choose **Accept**, then **Dismiss**.
7. To specify the case name, choose the **New Case** button to open the **Adding a New Case** dialog box. Type the case name in the *New Case* box. Select *Default* as the *New Case Type.* Choose **Accept**, then **Dismiss**.
8. Choose **Browse Cases** (File menu) to open the **Case Display** dialog box. In the **Case Display** dialog box, double-click Button 1 on the field that contains the case for the project. Double-clicking on the field opens a file manager listing all the project files located under that case. Choose **Dismiss** in the **Case Display** dialog box.
    1. Choose the **Get File** button from the file manager to display the *Get File* box at the bottom of the window. This box allows you to specify which file(s) you would like to add to the list of files for the current case.

2. Type /<*working directory*>/<*project name*>`.edo` in the *Get File* box and choose **Copy**. The new file appears in the list of design files.

3. Type /<*working directory*>/<*project name*>`.sdo` in the *Get File* box and choose **Copy**.

4. Type /<*working directory*>/<*project name*>`.ref` in the *Get File* box and choose **Copy**.

5. If your project contains memory functions, such as `ram`, `rom`, `dpram`, `scfifo`, `dcfifo`, `altdpram`, or `clklock`, type <*project name*>**.vmo** in the *Get File* box and choose **Copy** to add the MAX+PLUS II-generated VHDL Memory Model Output File (**.vmo**) to the list of files for the case. The MAX+PLUS II Compiler automatically generates this file for a project that contains memory functions.

   Every MOTIVE analysis requires a MOTIVE Clock Reference File (**.ref**). If the project is simple, you can create the file in the Setup Advisor. Otherwise, you must create the file in a text editor using MOTIVE syntax. For more information on the purpose, function, and syntax of MOTIVE Clock Reference Files, see the ***MOTIVE System Reference***.

6. Choose **Dismiss**.

9. Choose the **Netlister** button in the MOTIVE for Powerview Control Panel to open the **EDIF Netlist Parameters** dialog box. To create a FutureNet Format Netlist File (**.pin**) with the EEDIF Netlister for your design, follow these steps:

   1. Choose the **Select Design** button to open the **Select Design** dialog box.

   2. Double-click Button 1 on the project name to open the **Select Case** dialog box.

   3. Double-click Button 1 on the case name in the **Select Case** dialog box to open the **Select File** dialog box.

   4. Double-click Button 1 on the EDIF Output File, <*project name*>**.edo**, in the **Select File** dialog box.

   5. Select *Keep* for all *Case Sensitivity* options in the *EDIF Netlist Parameters* dialog box.

   6. Choose **Accept**, then **Dismiss** to close the **EDIF Netlist Parameters** dialog box.

10. Choose the **SDF2MTV** button in the Control Panel to open the **SDF2MTV (MOTIVE SDF Reader) Parameters** dialog box and specify the following options:

    1. Choose the **Select** button next to the *SDF Filename* box to open the **Select File** dialog box.

    2. Double-click Button 1 on the project's Standard Delay Format (SDF) Output File, <*project name*>**.sdo**, in the **Select File** dialog box. The **SDF2MTV** utility creates a MOTIVE Model Pre-Processor (MMP) Control File (**.ctl**) that allows you to annotate the parameterized library, and an Interconnect Delay Data File (**.idd**).

    3. Choose **Accept**, then **Dismiss** to close the **Select File** dialog box.

11. If your project contains `ram`, `rom`, `dpram`, `scfifo`, `dcfifo`, `altdpram`, or `clklock` megafunctions, use the **genmtv** utility to back-annotate the MMP Control File and to allow the MMP Control File to recognize the function. The input to the **genmtv** utility is the VHDL Memory Model Output File (**.vmo**) described above. From the ***/<working directory>/<project name>/<case name>*** directory, type the following command at the UNIX prompt:

    `genmtv` <*project name*> ↵

12. If your project contains RAM or ROM functions and you turned on the *Flatten Bus* option in the MAX+PLUS II Compiler's **EDIF Netlist Writer Settings** dialog box when you compiled your project, you must edit the **mem.lib** file, i.e., the MOTIVE Model Pre-Processor timing library file created with the **genmtv** utility. You must remove bracket `[ ]` characters from all occurrences of the address bus, e.g., change `A[0]` to `A0`, in both the `INPUTS` and `MIXED` sections of every RAM and ROM cell definition in **mem.lib**.

13. Choose the **MMP** button from the Control Panel to open the **MOTIVE Model Pre-processor (MMP) Parameters** dialog box and specify the following options:

    1. Choose the **Select** button next to the *MMP Ctl File* box to open the **Select File** dialog box.

    2. Double-click Button 1 on the project's MMP Control File, <*project name*>**.ctl**, in the **Select File** dialog box.

    3. In the **MOTIVE Model Pre-processor (MMP) Parameters** dialog box, choose the **Setup Model Libraries** button to display boxes on the right side of the dialog box that allow you to list additional source model libraries. In one of these boxes, type the following path and filename:

       `/usr/maxplus2/vwlogic/library/alt_time/motive.drv` ↵

    4. If your project contains RAM or ROM functions, repeat step 13c but specify the pathname of the **mem.lib** file created in step 12. For example:

       `/usr/maxplus2/`<***working directory***>**`/..../`**<***case name***>**`/mem.lib`** ↵

    5. In the **MOTIVE Model Pre-processor (MMP) Parameters** dialog box, choose **Accept**, then **Dismiss**. The **MMP** utility creates a design-specific Timing Model Library File (**.mod**).

14. Choose the **Analyze** button from the Control Panel to expand the Control Panel.

15. Double-click Button 1 on the project name in the *Select Design* box in the Control Panel to open the *Select Case* box.

16. Select the specific case of the project in the *Select Case* box and double-click Button 1 on the case name to open MOTIVE software and its Setup Advisor. The Setup Advisor helps guide you through the following steps to set up and configure a case analysis:

    1. In the Setup Advisor window, choose the **Continue** button to open the **Project Name Selection** dialog box, which displays the project name.

    2. Choose the **Begin analysis** button to open the **Checking for existing project** dialog box.

    3. Choose **Continue** to open the **Design Specific Flow(s)** dialog box and set up the project through the Setup Advisor. The *Design Name* option lists the project filename.

    4. Choose **Continue** to open the **Flow and Translation Selection** dialog box.

    5. Select the *Manual Translation Flow* option to specify input files and the steps to perform in the timing verification flow for MOTIVE software. Choose **Continue** to open the **Manual Flow Selection** dialog box and specify the following options:

Option:	**Setting:**
Netlist/Pinlist	*FutureNet (.pin)*
Parametric	*OVI Verilog (.sdf)*

       In the *Other* box, select *Use available MOTIVE files* to use the input files you created in previous steps. Choose **Continue** to open the **FutureNet Pinlist Preparation** dialog box.

6. Type the project name in the *Root Block* box. Choose **Continue** to open the **OVI Standard Parametric Back-annotation** dialog box.

7. Type <*project name*>**.sdo** in the *OVI (SDF) back-annotation file* box. Choose **Continue** to open the **MOTIVE Model Compilation** dialog box.

8. Replace the entry in the *Control file(s)* box with <*project name*>`.ctl`. Type the following two filenames, which must be separated by a space, in the *Libraries(s)* box:

   ```
   /usr/maxplus2/vwlogic/library/alt_time/motive.lib
   /usr/maxplus2/vwlogic/library/alt_time/motive.drv
   ```

9. If your project contains RAM or ROM functions, add the **mem.lib** file to the directories specified in step 16h.

10. Choose **Continue** to open the **Quick Definition of Existing MOTIVE Files** dialog box. The <*project name*>**.ref** filename appears in the *Clock Reference File (.ref)* box.

11. Replace the entry in the *Design's (pre-compiled) Model File (.mod)* box with <*project name*>*.mod.* Choose **Continue** to open the **Congratulations** dialog box.

12. Choose **Continue** to open the **Cleaning up** dialog box after completing the Setup Advisor interview. Select *Save under Project name* to save your setup, and choose **Continue** to close the Setup Advisor window.

17. In the MOTIVE window, choose **Verify** (Analyze menu) and then choose **Execute** to start verification. To view the output files, choose **Output Files** (View menu).

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Using Viewlogic MOTIVE & MOTIVE for Powerview Software with MAX+PLUS II Software

---

The following topics describe how to use the Viewlogic MOTIVE and MOTIVE for Powerview software with MAX+PLUS® II software. Click on one of the following topics for information:

This file is suitable for printing only. It does not contain hypertext links that allow you to jump from topic to topic.

Setting Up the MAX+PLUS II/Viewlogic Powerview Working Environment

- Software Requirements
- MAX+PLUS II/Viewlogic Powerview Interface File Organization
- MAX+PLUS II/Viewlogic Powerview Project File Structure

Timing Verification

- Timing Verification Flow
- Performing Timing Verification of EDIF Output Files (.edo) with MOTIVE & MOTIVE for Powerview Software
- Performing Timing Verification of Verilog Output Files (.vo) with MOTIVE Software

Related Topics:

- Go to the following MAX+PLUS II ACCESS℠ Key topics for related information:
  - Viewlogic Powerview Graphical User Interface & the Altera Toolbox
  - Powerview Command-Line Syntax
  - Compiling Projects with MAX+PLUS II Software
  - Programming Altera® Devices
- Go to the following topics, which are available on the web, for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware
  - Viewlogic web site (http://www.viewlogic.com)

Setting Up the MAX+PLUS II/Viewlogic Powerview Working Environment

To use the MAX+PLUS® II software with Viewlogic's Powerview software, you must install the MAX+PLUS II software, familiarize yourself with the Altera® Toolbox in the Powerview Cockpit, and then establish an environment that facilitates entering and processing designs. The MAX+PLUS II /Viewlogic Powerview interface is installed automatically when you install the MAX+PLUS II software on your workstation.

To set up your working environment for the MAX+PLUS II/Viewlogic Powerview interface, follow these steps:

1. Ensure that you have correctly installed the MAX+PLUS II and Viewlogic software versions described in MAX+PLUS II/Viewlogic Powerview Software Requirements.

2. Add the following environment variable to your **.cshrc** file to specify **/usr/maxplus2** as the MAX+PLUS II system directory:

   ```
   setenv ALT_HOME /usr/maxplus2 ↵
   ```

3. Add the **$ALT_HOME/viewlogic/standard**, **$ALT_HOME/bin**, and **$ALT_HOME/viewlogic/bin** directories to the `PATH` environment variable in your **.cshrc** file.

4. Add the **$ALT_HOME/viewlogic/standard** directory to the `WDIR` environment variable in your **.cshrc** file using the following syntax:

   ```
   setenv WDIR $ALT_HOME/viewlogic/standard:/<Powerview system directory>/standard ↵
   ```

   ☞ Make sure the **$ALT_HOME/viewlogic/standard** directory is the first directory in your `WDIR` path.

5. Source your **.cshrc** file by typing `source .cshrc ↵` at the UNIX prompt.

6. Create the Viewlogic Powerview **viewdraw.ini** configuration file.

7. Copy the **/usr/maxplus2/maxplus2.ini** file to your **$HOME** directory:

   ```
   cp /usr/maxplus2/maxplus2.ini $HOME ↵

   chmod u+w $HOME/maxplus2.ini ↵
   ```

   The **maxplus2.ini** file contains both Altera- and user-specified initialization parameters that control the MAX+PLUS II software, such as MAX+PLUS II symbol and logic function library paths and the current project name. The MAX+PLUS II installation procedure creates and copies the **maxplus2.ini** file to the **/usr/maxplus2** directory.

   ☞ Normally, you do not have to edit your local copy of **maxplus2.ini**, because the MAX+PLUS II software updates the file automatically whenever you change any parameters or settings. However, if you move the **max2lib** and **max2inc** library subdirectories, you must update the file. Go to "Creating & Using a Local Copy of the **maxplus2.ini** File" in MAX+PLUS II Help for more information.

8. If you plan to instantiate Library of Parameterized Modules (LPM) functions in ViewDraw schematics, you must create a new file with the name **vdraw.vs**. The **vdraw.vs** file must include the following line:

   ```
   load ("vdpath")
   ```

   You must also make sure that you specify the **vdraw.vs** file in your `WDIR` path.

9. Set up a directory structure that facilitates working with the MAX+PLUS II/Viewlogic Powerview interface. Refer to MAX+PLUS II/Viewlogic Powerview Project File Structure.

Related Topics:

- Go to *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual for more information on installation and details on the directories that are created during MAX+PLUS II installation. Go to MAX+PLUS II/Viewlogic Powerview Interface File Organization for information about the MAX+PLUS II/Viewlogic Powerview directories that are created during MAX+PLUS II installation.
- Go to the following topics, which are available on the web, for additional information:

- ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
- This manual is also available in 4 parts:
  - Preface & Section 1: MAX+PLUS II Installation
  - Section 2: MAX+PLUS II - A Perspective
  - Section 3: MAX+PLUS II Tutorial
  - Appendices, Glossary & Index

MAX+PLUS II/Viewlogic Powerview Software Requirements

The following applications and utilities are used to generate, process, synthesize, and verify a project with MAX+PLUS® II and Viewlogic Powerview software.

	Viewlogic	**Altera**
ViewDraw	ViewGen	MAX+PLUS II version 9.4
VHDL Analyzer	ViewPath (optional)	
Vantage VHDL Analyzer	ViewTrace	
VHDL -> sym	ViewData Path	
edifneto	MOTIVE version 5.1.6 *Note (1)*	
edifneti	MOTIVE for Powerview version 3.2.1 (optional) *Note (1)*	
EEDIF (optional)	SDF2MTV (optional)	
MMP (optional)	Fusion/VCS	
vsm		

Note:

(1) MOTIVE for Powerview, a wrapper application for MOTIVE, provides a graphical user interface for the utilities (i.e., EEDIF, SDF2MTV, and MMP) used during a static timing verification with MOTIVE. MOTIVE alone does not accept EDIF files through the Setup Advisor.

☞ The MAX+PLUS II **read.me** file provides up-to-date information on which versions of Viewlogic Powerview applications the current version of the MAX+PLUS II software supports. It also provides information on installation and operating requirements. You should read the **read.me** file on the CD-ROM before installing the MAX+PLUS II software. After installation, you can open the **read.me** file from the MAX+PLUS II Help menu.

MAX+PLUS II/Viewlogic Powerview Interface File Organization

Table 1 shows the MAX+PLUS® II/Viewlogic Powerview interface subdirectories that are created in the MAX+PLUS II system directory (by default, the **/usr/maxplus2** directory) during MAX+PLUS II installation.

☞ For information on the other directories that are created during MAX+PLUS II installation, see "MAX+PLUS II File Organization" in *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual.

Table 1. MAX+PLUS II Directory Organization

Directory	**Description**
./lmf	Contains the Altera-provided Library Mapping File, **vwlogic.lmf**, that maps Viewlogic logic functions to equivalent MAX+PLUS II logic functions.
./viewlogic	Contains the **alt_edif.cfg** EDIF configuration file that is used with the **edifneti** utility. Also contains the library and sample subdirectories.

./viewlogic/examples	Contains the sample Viewlogic designs.
./viewlogic/library/max2sim	Contains the MAX+PLUS II simulation model library (**max2_sim**) for use in ViewSim software.
./viewlogic/library/alt_max2	Contains MAX+PLUS II primitives (`EXP`, `GLOBAL`, `LCELL`, `SOFT`, `CARRY`, `CASCADE`, `DFFE`, `DFFE6K`, and `OPNDRN`), macrofunctions (`a_8fadd`, `a_8mcomp`, `a_8count`, `a_81mux`), and megafunctions (`clklock`) for use in ViewDraw schematics. These logic functions support specific architectural features of Altera® devices. The **alt_max2** library also contains modified versions of the ViewDraw primitives that use tri-state buffers, because these primitives require special handling in the MAX+PLUS II /Viewlogic Powerview interface.
./viewlogic/library/synlib	Contains the Altera-provided synthesis library **altera**, which includes MAX+PLUS II primitives, the **altera.sml** file, a **sym** directory, and a **wir** directory for use with ViewSynthesis software.
./viewlogic/library/alt_mf	Contains the VHDL models for the MAX+PLUS II primitives (`EXP`, `GLOBAL`, `LCELL`, `SOFT`, `CARRY`, `CASCADE`, `DFFE`, and `OPNDRN`), macrofunctions (`clklock`) for use with ViewSynthesis software, the Vantage VHDL Analyzer software, and the VHDL source files. These logic functions are used to maintain portability to other architectures.
./viewlogic/library/alt_time	Contains MOTIVE timing models for MAX+PLUS II logic functions (**motive.lib**), including the `clklock` megafunction, and MAX+PLUS II driver models (**motive.drv**).
./viewlogic/library/alt_vtl	Contains the VHDL source files for the VITAL 3.0-compliant library. This library is available for ViewSim software.
./viewlogic/bin	Contains all MAX+PLUS II, Viewlogic, and interface-related scripts.
./viewlogic/standard	Contains all standard **.ini** files and standard tools.

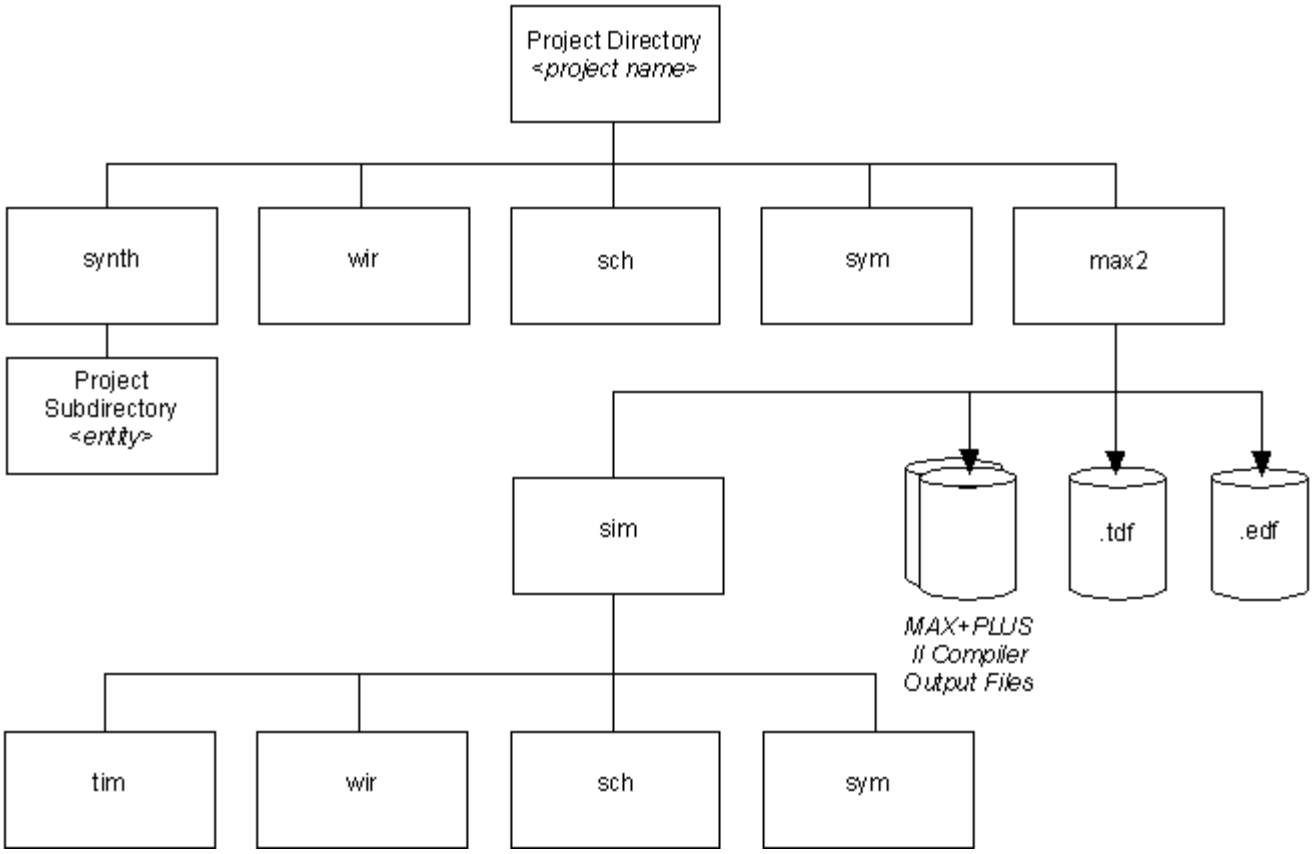
Related Topics:

- Go to the following topics, which are available on the web, for additional information:
  - ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
  - This manual is also available in 4 parts:
    - Preface & Section 1: MAX+PLUS II Installation
    - Section 2: MAX+PLUS II - A Perspective
    - Section 3: MAX+PLUS II Tutorial
    - Appendices, Glossary & Index

MAX+PLUS II/Viewlogic Powerview Project File Structure

In the MAX+PLUS® II software, a project name is the name of a top-level design file, without the filename extension. This design file can be an EDIF, Verilog HDL, or VHDL netlist file; an Altera® Hardware Description Language (AHDL) TDF; or any other MAX+PLUS II- supported design file. The EDIF netlist file must be created by Powerview and imported into the MAX+PLUS II software as an EDIF Input File (**.edf**). Figure 1 shows an example of MAX+PLUS II project directory structure that includes Powerview-generated files.

Figure 1. Sample MAX+PLUS II Project Organization



The MAX+PLUS II software stores the connectivity data on the links between design files in a hierarchical project in a Hierarchy Interconnect File (**.hif**), but refers to the entire project only by its project name. The MAX+PLUS II Compiler uses the HIF to build a single, fully flattened project database that integrates all the design files in a project hierarchy.

Unlike Powerview, the MAX+PLUS II software does not automatically create a project directory when you create a project. A single directory can contain several MAX+PLUS II design files, and you can specify any one of the designs in the directory as a project in the MAX+PLUS II software.

Viewlogic Powerview Local Work Area Structure

When you create a project with the Powerview Cockpit's **Create** command (Project menu), the project directory is created. You should generate design files and functional simulation files under this directory. A **max2** subdirectory is automatically created under your current project directory when you generate an EDIF file from your schematic or VHDL file. The ***<project name>*.edf** file is stored in the **max2** subdirectory. All MAX+PLUS® II Compiler output files are created in the **/*<project name>*/max2** subdirectory.

☞ ViewDraw files are identified by their directories and not by their extensions, so it is easy to overwrite files unintentionally. To avoid overwriting files, Altera recommends that you create a new project directory, ***<project name>*/max2/sim**, where you can generate all the files needed for simulation.

ViewDraw Project File Structure

Each ViewDraw project directory contains three subdirectories: **wir**, **sch**, and **sym**. See Table 1.

Table 1. ViewDraw Subdirectories

Directory	Topics

./wir	Wirelist files that contain connectivity information for a particular logic block
./sch	Schematics that contain logic
./sym	Symbol files that are the ViewDraw graphical representation of the logic blocks

Each file type uses the filename extension **.1**. Different file types are distinguished only by their directory: **/lib/wir/**<*project name*>**.1** is a wirelist file; **/lib/sch/**<*project name*>**.1** is the corresponding schematic file; and **/lib/sym/**<*project name*>**.1** is the corresponding symbol.

VHDL Project File Structure

Each VHDL project directory contains three subdirectories. See Table 2.

Table 2. VHDL Subdirectories

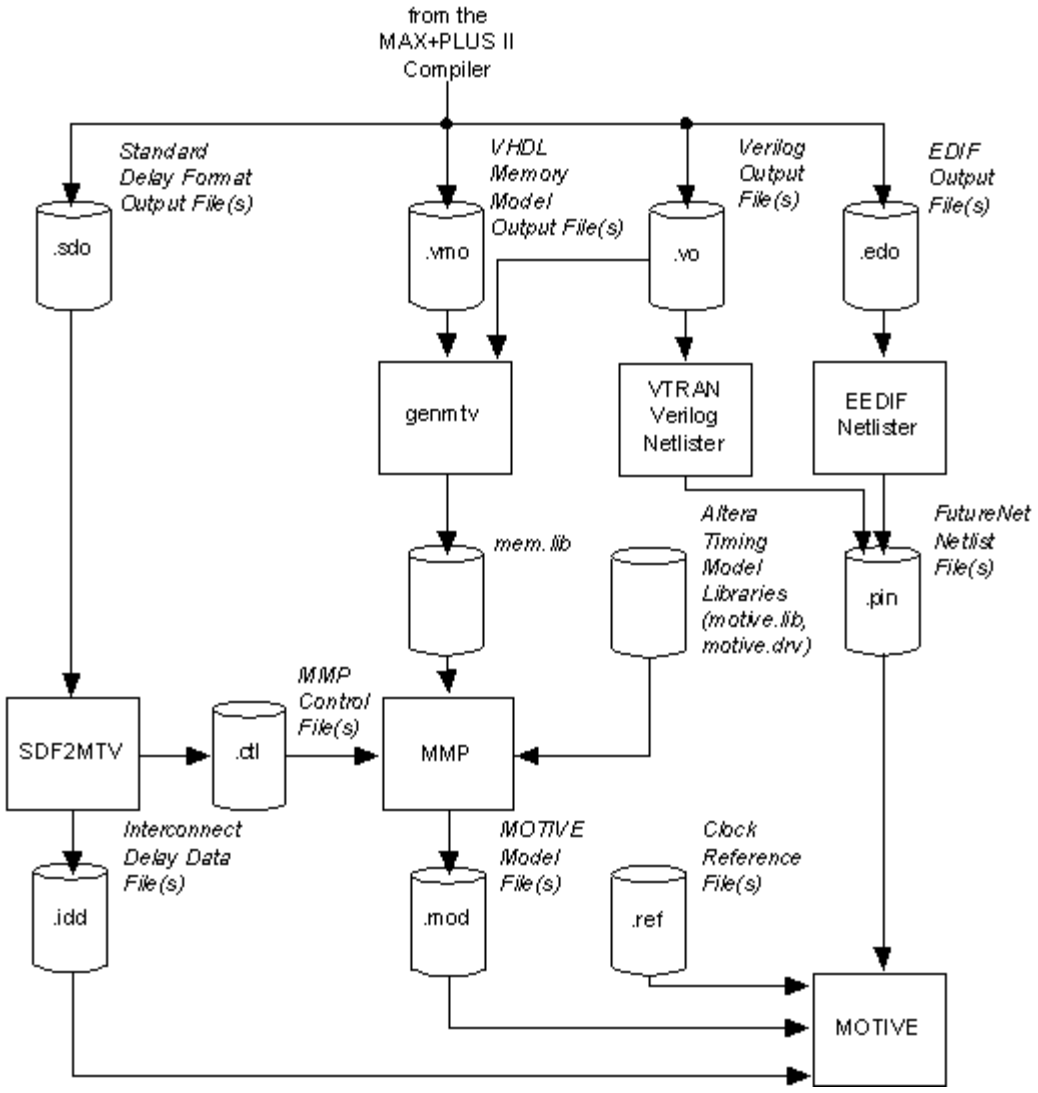
Directory	Topics
./synth	All synthesis-related files and directories
./synth/<*entity*>	Four types of files: <*entity*>**.pdf**, <*entity*>**.opt**, <*entity*>**.sta**, and <*entity*>**.gnl**
./wir	Wirelist for synthesized VHDL modules

☞ For each VHDL entity in the design, there is a corresponding **./synth/<*entity*>** directory.

MAX+PLUS II/Viewlogic Powerview Timing Verification Flow

Figure 1 shows the timing verification flow for the MAX+PLUS® II/Viewlogic Powerview interface.

Figure 1. MAX+PLUS II/Viewlogic Powerview Project Timing Verification Flow



Performing Timing Verification of EDIF Output Files (.edo) with MOTIVE & MOTIVE for Powerview Software

After you have compiled a project and generated an EDIF Output File (**.edo**) with the MAX+PLUS® II software, you can use Viewlogic MOTIVE or MOTIVE for Powerview software to perform timing verification. The **max2_MOTIVE** tool is located in both the Altera® Toolbox Design Tools Drawer and the Altera Toolbox Max2 Express Drawer. The MOTIVE timing model library, **motive.lib**, provides models of basic primitives and the clklock megafunction for timing verification.

To perform timing verification for EDIF Output Files with MOTIVE or MOTIVE for Powerview software, follow these steps:

1. Set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Viewlogic Powerview Working Environment.
2. Generate an EDIF Output File (**.edo**) by compiling your design with the MAX+PLUS II software, as described in Compiling Projects with MAX+PLUS II Software.
3. Start the MOTIVE for Powerview software by double-clicking Button 1 on the **max2_MOTIVE** icon in the Altera Toolbox Design Tools Drawer. The MOTIVE for Powerview Control Panel opens.

4. Choose **Setup Environment** (File menu) to open the **Environment Parameters** dialog box, and specify the following options:

    1. Specify the directory for the *Project Directory* option.

    2. Specify */usr/maxplus2/vwlogic/library/alt_time/motive.lib* for the *Model Library Search Path* option.

    3. Select *EDIF* for the *Netlist Input Format* option.

    4. Choose **Accept**. The MOTIVE for Powerview software automatically creates a **tim** subdirectory, which contains MOTIVE design cases and related files, in the current working directory.

5. Choose **Save Parameters** (File menu) to save your customized project setup.

6. To specify the project name, choose the **New Design** button to open the **Adding a New Design** dialog box. Type the design name in the *New Design* box. Choose **Accept**, then **Dismiss**.

7. To specify the case name, choose the **New Case** button to open the **Adding a New Case** dialog box. Type the case name in the *New Case* box. Select *Default* as the *New Case Type.* Choose **Accept**, then **Dismiss**.

8. Choose **Browse Cases** (File menu) to open the **Case Display** dialog box. In the **Case Display** dialog box, double-click Button 1 on the field that contains the case for the project. Double-clicking on the field opens a file manager listing all the project files located under that case. Choose **Dismiss** in the **Case Display** dialog box.

    1. Choose the **Get File** button from the file manager to display the *Get File* box at the bottom of the window. This box allows you to specify which file(s) you would like to add to the list of files for the current case.

    2. Type /<*working directory*>/<*project name*>`.edo` in the *Get File* box and choose **Copy**. The new file appears in the list of design files.

    3. Type /<*working directory*>/<*project name*>`.sdo` in the *Get File* box and choose **Copy**.

    4. Type /<*working directory*>/<*project name*>`.ref` in the *Get File* box and choose **Copy**.

    5. If your project contains memory functions, such as `ram`, `rom`, `dpram`, `scfifo`, `dcfifo`, `altdpram`, or `clklock`, type <*project name*>**.vmo** in the *Get File* box and choose **Copy** to add the MAX+PLUS II-generated VHDL Memory Model Output File (**.vmo**) to the list of files for the case. The MAX+PLUS II Compiler automatically generates this file for a project that contains memory functions.

        Every MOTIVE analysis requires a MOTIVE Clock Reference File (**.ref**). If the project is simple, you can create the file in the Setup Advisor. Otherwise, you must create the file in a text editor using MOTIVE syntax. For more information on the purpose, function, and syntax of MOTIVE Clock Reference Files, see the ***MOTIVE System Reference***.

    6. Choose **Dismiss**.

9. Choose the **Netlister** button in the MOTIVE for Powerview Control Panel to open the **EDIF Netlist Parameters** dialog box. To create a FutureNet Format Netlist File (**.pin**) with the EEDIF Netlister for your design, follow these steps:

    1. Choose the **Select Design** button to open the **Select Design** dialog box.

    2. Double-click Button 1 on the project name to open the **Select Case** dialog box.

    3. Double-click Button 1 on the case name in the **Select Case** dialog box to open the **Select File** dialog

box.

4. Double-click Button 1 on the EDIF Output File, <*project name*>**.edo**, in the **Select File** dialog box.

5. Select *Keep* for all *Case Sensitivity* options in the *EDIF Netlist Parameters* dialog box.

6. Choose **Accept**, then **Dismiss** to close the **EDIF Netlist Parameters** dialog box.

10. Choose the **SDF2MTV** button in the Control Panel to open the **SDF2MTV (MOTIVE SDF Reader) Parameters** dialog box and specify the following options:

    1. Choose the **Select** button next to the *SDF Filename* box to open the **Select File** dialog box.

    2. Double-click Button 1 on the project's Standard Delay Format (SDF) Output File, <*project name*>**.sdo**, in the **Select File** dialog box. The **SDF2MTV** utility creates a MOTIVE Model Pre-Processor (MMP) Control File (**.ctl**) that allows you to annotate the parameterized library, and an Interconnect Delay Data File (**.idd**).

    3. Choose **Accept**, then **Dismiss** to close the **Select File** dialog box.

11. If your project contains `ram`, `rom`, `dpram`, `scfifo`, `dcfifo`, `altdpram`, or `clklock` megafunctions, use the **genmtv** utility to back-annotate the MMP Control File and to allow the MMP Control File to recognize the function. The input to the **genmtv** utility is the VHDL Memory Model Output File (**.vmo**) described above. From the ***/<working directory>/<project name>/<case name>*** directory, type the following command at the UNIX prompt:

    `genmtv` <*project name*> ↵

12. If your project contains RAM or ROM functions and you turned on the *Flatten Bus* option in the MAX+PLUS II Compiler's **EDIF Netlist Writer Settings** dialog box when you compiled your project, you must edit the **mem.lib** file, i.e., the MOTIVE Model Pre-Processor timing library file created with the **genmtv** utility. You must remove bracket `[ ]` characters from all occurrences of the address bus, e.g., change `A[0]` to `A0`, in both the `INPUTS` and `MIXED` sections of every RAM and ROM cell definition in **mem.lib**.

13. Choose the **MMP** button from the Control Panel to open the **MOTIVE Model Pre-processor (MMP) Parameters** dialog box and specify the following options:

    1. Choose the **Select** button next to the *MMP Ctl File* box to open the **Select File** dialog box.

    2. Double-click Button 1 on the project's MMP Control File, <*project name*>**.ctl**, in the **Select File** dialog box.

    3. In the **MOTIVE Model Pre-processor (MMP) Parameters** dialog box, choose the **Setup Model Libraries** button to display boxes on the right side of the dialog box that allow you to list additional source model libraries. In one of these boxes, type the following path and filename:

       `/usr/maxplus2/vwlogic/library/alt_time/motive.drv` ↵

    4. If your project contains RAM or ROM functions, repeat step 13c but specify the pathname of the **mem.lib** file created in step 12. For example:

       `/usr/maxplus2/`*<working directory>*`/..../`*<case name>*`/mem.lib` ↵

    5. In the **MOTIVE Model Pre-processor (MMP) Parameters** dialog box, choose **Accept**, then **Dismiss**. The **MMP** utility creates a design-specific Timing Model Library File (**.mod**).

14. Choose the **Analyze** button from the Control Panel to expand the Control Panel.

15. Double-click Button 1 on the project name in the *Select Design* box in the Control Panel to open the *Select Case* box.

16. Select the specific case of the project in the *Select Case* box and double-click Button 1 on the case name to open MOTIVE software and its Setup Advisor. The Setup Advisor helps guide you through the following steps to set up and configure a case analysis:

    1. In the Setup Advisor window, choose the **Continue** button to open the **Project Name Selection** dialog box, which displays the project name.

    2. Choose the **Begin analysis** button to open the **Checking for existing project** dialog box.

    3. Choose **Continue** to open the **Design Specific Flow(s)** dialog box and set up the project through the Setup Advisor. The *Design Name* option lists the project filename.

    4. Choose **Continue** to open the **Flow and Translation Selection** dialog box.

    5. Select the *Manual Translation Flow* option to specify input files and the steps to perform in the timing verification flow for MOTIVE software. Choose **Continue** to open the **Manual Flow Selection** dialog box and specify the following options:

Option:	**Setting:**
Netlist/Pinlist	*FutureNet (.pin)*
Parametric	*OVI Verilog (.sdf)*

       In the *Other* box, select *Use available MOTIVE files* to use the input files you created in previous steps. Choose **Continue** to open the **FutureNet Pinlist Preparation** dialog box.

    6. Type the project name in the *Root Block* box. Choose **Continue** to open the **OVI Standard Parametric Back-annotation** dialog box.

    7. Type <*project name*>**.sdo** in the *OVI (SDF) back-annotation file* box. Choose **Continue** to open the **MOTIVE Model Compilation** dialog box.

    8. Replace the entry in the *Control file(s)* box with <*project name*>`.ctl`. Type the following two filenames, which must be separated by a space, in the *Libraries(s)* box:

       ```
       /usr/maxplus2/vwlogic/library/alt_time/motive.lib
       /usr/maxplus2/vwlogic/library/alt_time/motive.drv
       ```

    9. If your project contains RAM or ROM functions, add the **mem.lib** file to the directories specified in step 16h.

    10. Choose **Continue** to open the **Quick Definition of Existing MOTIVE Files** dialog box. The <*project name*>**.ref** filename appears in the *Clock Reference File (.ref)* box.

    11. Replace the entry in the *Design's (pre-compiled) Model File (.mod)* box with <*project name*>*.mod.* Choose **Continue** to open the **Congratulations** dialog box.

    12. Choose **Continue** to open the **Cleaning up** dialog box after completing the Setup Advisor interview. Select *Save under Project name* to save your setup, and choose **Continue** to close the Setup Advisor window.

17. In the MOTIVE window, choose **Verify** (Analyze menu) and then choose **Execute** to start verification. To view the output files, choose **Output Files** (View menu).

Performing Timing Verification of Verilog Output Files (.vo) with MOTIVE Software

After you have compiled a project and generated a Verilog Output File (**.vo**) with the MAX+PLUS II Software, you can use Viewlogic MOTIVE to perform timing verification. The MOTIVE timing model library, **motive.lib**, provides basic primitives and the `clklock` megafunction for timing verification.

To perform timing verification for Verilog Output Files with MOTIVE software, follow these steps:

1. Set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Viewlogic Powerview Working Environment.

2. Generate a Verilog Output File by compiling your design with the MAX+PLUS II software, as described in Compiling Projects with MAX+PLUS II Software.

3. Start the MOTIVE software by typing `motive` ↵ at the UNIX prompt. The MOTIVE Session Log and Setup Advisor windows are displayed. Choose **OK**.

4. Choose **Project** on the vertical menubar in the Setup Advisor, then choose the **Name** (Select project name) tab and specify the name of the project for *Project name*. The directory in which you started MOTIVE will be selected automatically for *Current directory*. Choose **Accept**. MOTIVE then searches for the <*project name*>**.stm** file. If this is a new file, a message will appear in the Session Log window that mentions that MOTIVE found a license and the message `could not open the <project name>.stm file -- assuming a new design.`

5. Choose **Flow** from the vertical menubar, then choose the **Type** (Select flow type) tab. Select the *Using Verilog and SDF* option and choose **Accept**.

6. Choose **Options** from the vertical menubar, then choose the **Options** (Miscellaneous usage options) tab. If desired, specify a different value for the *MOTIVE analysis cycle time* option. Choose **Accept**.

7. Choose **Verilog** on the vertical menubar and specify the following Verilog HDL input options:

    1. Choose the **Translate** (Translate Verilog netlist file) tab. Specify the name of the MAX+PLUS II-generated Verilog Output File (**.vo**) for the *Verilog netlist* option. Choose the **Common Options** button to display the **Common Options** dialog box. Select the *Special Options* option and turn on the *Skip Behavioral Constructs* option. Type either `pinlist` or a period (`.`) for the *Generated pin files* option. Choose **OK** to close the **Common Options** dialog box and return to the **Translate** tab.

    2. Specify the location of the MAX+PLUS II-generated **alt_max2.vo** file for the *Vendor module definition* option. Choose the **Translate** button. The **Process Execution Log & Tips** dialog box displays the current status of the translation to **.pin** files. Choose **OK** after successful translation.

    3. Choose the **Import** (Confirm Adding hierarchical blocks) tab. Choose the **Import Blocks** button. The **MOTIVE Interaction Log & Tips** dialog displays the current import status. Choose **OK** after a successful completion.

    4. Select the **Hierarchy** (Configure hierarchy options) tab. Type the name of the rootblock for the *Rootblock of design* option, or choose the **Find Rootblock** button to display the rootblock name. Choose **Accept**.

8. Choose the **Check** (Review and/or build the netlist database) tab. Choose the **Incremental Build** button. The **MOTIVE Interaction Log & Tips** dialog displays the current build status. Choose **OK** after a successful completion.

9. Select **SDF** on the vertical menubar, then select the **Translate** (SDF model preparation) tab. Type <*project name*>`.sdo` for the *SDF file* option, making sure that you specify the **.sdo** extension. Type <*project name*>`.ctl` for the *MPP control file name*, and <*project name*>`.idd` for the *IDD file name*.

10. Choose the **Process SDF File** button.

11. If your project contains the `clklock` megafunction, use the **genmtv** utility to back-annotate the MPP Control File and to allow the MPP Control File to recognize the `clklock` function. The input to the **genmtv** utility is the Verilog netlist file (**.vo**). From the /<*working directory*>/<*project name*>/<*case name*> directory, type the following command at the UNIX prompt:

    `genmtv -v` <*project name*> ↵

12. If your project contains RAM or ROM functions and you turned on the *Flatten Bus* option the MAX+PLUS II Compiler's **Verilog Netlist Writer Settings** dialog box when you compiled your project, you must edit the **mem.lib** file, i.e., the MOTIVE Model Pre-Processor timing library file generated with the **genmtv** utility. You must remove the bracket `[ ]` characters from all occurrences of the address bus, e.g., change `A[0]` to `A0`, in both the `INPUTS` and `MIXED` sections of every RAM and ROM cell definition in **mem.lib**.

13. Select the **MPP** (MOTIVE model compilation) tab. Type <*project name*>`.ctl` for the *Control file* option. Type `/usr/maxplus2/viewlogic/library/alt_time/motive.lib /usr/maxplus2/viewlogic/library/alt_time/motive.drv` for the *Libraries* option. If the project contains memory functions, you should also specify the location of the **mem.lib** file for the *Libraries* option. Type <*project name*>`.mod` for the *Generated model file* option and <*project name*>`.rcf` for the *Revised control file* option. Choose the **RUN MMP** button. The **MOTIVE Execution Log & Tips** dialog displays and shows the current status. Choose **OK** after a successful completion.

14. Select **Save** from the File menu in the Setup Advisor to write all the selections made so far to the <*project name*>**.stm** file.

15. Select **Clock** on the vertical menubar, then choose the **File** (Check reference file and timebase options) tab. The correct name of the Clock Reference File (**.ref**) should be displayed for the *Clock reference file* option. Choose **Accept**.

    ☞ Every MOTIVE analysis requires a MOTIVE Clock Reference File. If the project is simple, you can create the file in the Setup Advisor. Otherwise, you must create the file with a text editor using MOTIVE syntax. For more information on the purpose, function, and syntax of MOTIVE Clock Reference Files, see the ***MOTIVE System*** **Reference**.

16. Choose the **Edit** (Simple clock reference generation) tab. Specify the names for the *Clock reference* and *Clock net name* options. Choose **Generate**.

17. Choose the **Check** (Choose incremental definitions) tab, then choose the **Load Clock** button.

18. Choose **Finish** from the vertical menubar, then choose the **Build** button. The **MOTIVE Interaction Log & Tips** dialog displays the current status. Choose **OK** after a successful completion.

19. Select **Save** from the File menu in the Setup Advisor.

20. In the MOTIVE Session Log window, choose **Verify** (Analyze menu) and then choose the **Execute** button to start verification. To view the output files, choose **Output Files** (View menu).

Alternatively, you can run MOTIVE analysis on the command line by following these steps:

1. Type the following commands at the UNIX prompt:

    `vtran` <*project name*>`.vo -b -h -u alt_max2.vo` ↵ (generates **.pin** files)

    `sdf2mtv` <*project name*>`.sdfo` ↵ (generates **.ctl** files)

2. If your project contains `ram`, `rom`, `dpram`, or `clklock` functions, you should also type the following commands

at the UNIX prompt:

```
genmtv -v <project name> ↵
```

```
mmp <project name>.ctl -l /usr/maxplus2/viewlogic/library/alt_time/motive.lib -l
/usr/maxplus2/viewlogic/library/alt_time/motive/drv -l mem.lib ↵
```

3. Type the following command at the UNIX prompt:

```
amtv <project name>
```

Viewlogic Powerview Graphical User Interface & the Altera Toolbox

You use the Powerview graphical interface manager, the Cockpit, and the Altera® Toolbox to start all Powerview and Altera tools. Within the Altera Toolbox, you can specify the Max2 Express Drawer or the Design Tools Drawer to work with the Altera/Viewlogic Powerview interface.

The Max2 Express Drawer provides a quick and seamless way to transfer designs created in Powerview to the MAX+PLUS® II software for compilation, then return the compiled designs to Powerview for simulation and timing verification. Table 1 describes the Max2 Express Drawer tools.

Table 1. Max2 Express Drawer Tools

Tool	Description
max2_VDraw	Launches the Powerview ViewDraw schematic entry tool.
VHDL<->max2	Launches all tools necessary to synthesize a VHDL design, compile for an Altera device, and generate a **.vsm** file for simulation with the Powerview ViewSim simulator.
SCH<->max2	Launches all tools necessary to compile a schematic design entered with Powerview ViewDraw software for an Altera device and to generate a **.vsm** file for simulation with Powerview ViewSim and **.edo**, **.sdo**, and **.vmo** files for timing analysis with MOTIVE for Powerview.
max2_VSim	Launches the Powerview ViewSim simulator.
max2_VTrace	Launches the Powerview ViewTrace simulation waveform editor.
max2_MOTIVE	Launches the MOTIVE for Powerview ViewDraw static timing verification tool.

The Design Tools Drawer provides tools that enable you to create a design with the Powerview tools, compile the design in the MAX+PLUS II software, and simulate and verify the design with Powerview software. Table 2 describes the Design Tools Drawer tools.

Table 2. Design Tools Drawer Tools

Tool	Description
max2_VDraw	Launches the Powerview ViewDraw schematic entry tool.
max2_analyzer	Launches the Powerview VHDL Analyzer software.
max2_syn	Launches the Powerview VHDL synthesis tool.
max2_chk	Launches the Powerview schematic verification tool.
max2_vsmnet	Launches the Powerview **vsm** utility that converts a wirelist file into a **.vsm** file.
max2_VSim	Launches the Powerview ViewSim simulator.
max2_VTrace	Launches the Powerview ViewTrace simulator.
max2_edifo	Launches the Powerview EDIF netlist writer, **edifneto**.
max2_VGen	Launches the Powerview ViewGen utility that generates a schematic from a wirelist file.

max2	Launches the MAX+PLUS II Compiler.
max2_edifi	Launches the Powerview EDIF Netlist Reader, **edifneti**.
max2_vhdl2sym	Launches the Powerview **vhdl2sym** utility that generates a symbol from a VHDL file.
max2_VantgMgr	Launches the Powerview Vantage VHDL Library Manager tool.
max2_VantgAnlz	Launches the Vantage VHDL Analyzer software.
max2_VCS	Launches the Fusion/VCS Simulator.
max2_MOTIVE	Launches the MOTIVE for Powerview static timing verification tool.

Powerview Command-Line Syntax

Table 1 shows the command-line syntax for using Powerview functions.

Table 1. Powerview Command-Line Syntax

Action	Command
Start VHDL Analyzer software	`vhdl -v` <*project name*>
Start ViewSynthesis software	`vhdldes`
Load Altera® technology library	`vhdldes> technology altera`
Compile a VHDL design	`vhdldes> vhdl` <*project name*>
Synthesize a design	`vhdldes> synthesize`
Generate wirelist file	`vhdldes> wir`
Create a schematic representation	`vhdldes> viewgen`
Generate a synthesis report file	`vhdldes> report`
Start the graphical user interface for ViewSynthesis	`vhdldes> vdesgui`
Start the VHDL-to-symbol utility	`vhdl2sym` <*project name*>
Start **vsm**	`vsm` <*project name*>
Start ViewSim simulator	`viewsim` <*project name*> -<*project name*>`.cmd`
Start **edifneto**	`edifneto -f` <*project name*>-`l` (`std` *or* `altera`) <*project name*>`.edf`
Start Vantage VHDL Analyzer software	`analyze -src` <*design file*>
Start MOTIVE for Powerview software	`mfp`

Compiling Projects with MAX+PLUS II Software

The MAX+PLUS® II Compiler can process design files in a variety of formats. This topic describes how to use MAX+PLUS II software to compile projects in which the top-level design file is an EDIF Input File (with the extension **.edf**).

☞ Refer to the following sources for additional information:

- Go to MAX+PLUS II Help for information on compiling VHDL and Verilog HDL, design files directly with the MAX+PLUS II Compiler.
- Go to Running Synopsys Compilers from MAX+PLUS II Software for information on running the Synopsys Design Compiler or FPGA Compiler software on a VHDL or Verilog HDL design from within the MAX+PLUS II Compiler window.

To compile a design (also called a "project") with MAX+PLUS II software, go through the following steps:

1. Create design files that are compatible with the MAX+PLUS II software and convert them into EDIF Input Files with the extension **.edf**. Specific instructions for some tools are described in these MAX+PLUS II ACCESS^{SM} Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your design entry or synthesis and optimization tool.

2. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, create a mapping for each function in a Library Mapping File (**.lmf**) to map the custom symbol to the corresponding EDIF Input File, AHDL Text Design File (**.tdf**), or other MAX+PLUS II-supported design file. These custom functions are represented in design files as hollow-body symbols or "black box" HDL descriptions.

   ☞ Go to "Library Mapping Files (**.lmf**)" in MAX+PLUS II Help for more information.

3. Open MAX+PLUS II and specify the name of your top-level design file as the project name with the **Project Name** command (File menu). If you open an HDL file in the MAX+PLUS II Text Editor, you can choose the **Project Set Project to Current File** command (File menu) instead.

   ☞ You can also compile a project from a command line. However, the first time you compile a project, the settings you need to specify are easier to specify from within the MAX+PLUS II software. After you have run the graphical user interface for the MAX+PLUS II software at least once, you can more easily use the command-line **setacf** utility to modify options in the Assignment & Configuration File (**.acf**) for the project. Type `setacf -h` ↵ and `maxplus2 -h` ↵ for descriptions of **setacf** and MAX+PLUS II command-line syntax.

4. Choose **Device** (Assign menu) and select the target Altera device family in the *Device Family* drop-down list box. If you wish to implement the design logic in a specific device, select it in the *Devices* box. Otherwise, select *AUTO* to allow the MAX+PLUS II Compiler to choose the best device(s) in the current device family. If your design entry or synthesis and optimization tool required you to specify a target family and/or device, specify the same information in this dialog box. For information on partitioning logic among multiple devices, go to MAX+PLUS II Help. Choose **OK**.

5. Open the Compiler window by choosing the **Compiler** command (MAX+PLUS II menu). Go through the following steps to specify the options necessary to compile the design file(s) in your project:

   1. Ensure that all EDIF netlist files have the extension **.edf** and choose **EDIF Netlist Reader Settings** (Interfaces menu).

   2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor. This name should be the name of the vendor whose tool(s) you used to create the EDIF netlist files. If your vendor name does not appear, select *Custom* instead.

      ☞ If you are compiling a design created with Synopsys FPGA Express software, select *Synopsys*, choose the **Customize** button, enter <*project name*>**.lmf** in the *LMF #1* box, choose **OK**, and skip to step 6.

   3. If you selected an existing vendor name in the *Vendor* box and your project contains design files that require custom LMF mappings, choose the **Customize** button to expand the dialog box to show all settings. Turn on the *LMF #2* checkbox and type your custom LMF's filename in the corresponding text box, or select a name from the *Files* box. The selection in the *Vendor* box will change to *Custom* and all settings will be retained until you change them again.

   4. If you selected *Custom* in the *Vendor* box, choose the **Customize** button to expand the dialog box to

show all settings. Any previously defined custom settings will be displayed. Under *Signal Names*, type one or more names with up to 20 total name characters in the *VCC* or *GND* box if your EDIF Input File(s) use one or more names other than `VCC` or `GND` for the global high or low signals. Multiple signal names must be separated by either a comma (,) or a space. Under *Library Mapping Files*, turn on the *LMF #1* checkbox and type a filename in the text box following it, or select a name from the *Files* box. If necessary, specify another LMF name in the *LMF #2* box. Go to MAX+PLUS II Help for detailed information on the settings available in the **EDIF Netlist Reader Settings** dialog box.

5. Choose **OK**.

6. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, you may need to ensure that all files are present in your project directory, i.e., the same directory as the top-level design file. Otherwise, you must specify the directories containing these files as user libraries with the **User Libraries** command (Options menu).

7. Follow all guidelines that apply to your design entry or synthesis and optimization tool:

   - Exemplar Logic Galileo Extreme-Specific Compiler Settings
   - Synopsys DesignWare-Specific Compiler Settings
   - Converting Synopsys FPGA Compiler & Design Compiler Timing Constraints into MAX+PLUS II-Compatible Format with the **syn2acf** Utility
   - Synplicity Synplify-Specific Compiler Settings

8. If you wish to generate EDIF, VHDL, or Verilog HDL output files for post-compilation simulation or timing analysis with another EDA tool, go through the following steps:

   1. (Optional) Turn on the **Optimize Timing SNF** command (Processing menu) to reduce the size of the output file(s). Turning on this command can reduce the size of output netlists by up to 30%.

      This command does not create optimized timing SNFs on UNIX workstations. However, a non-optimized timing SNF provides the same functional and timing information as an optimized timing SNF.

   2. If you wish to generate EDIF Output Files (**.edo**), go through these steps:

      1. Turn on the **EDIF Netlist Writer** command (Interfaces menu). Then choose the **EDIF Netlist Writer Settings** command (Interfaces menu).

      2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor and choose **OK**. If your vendor name does not appear, select *Custom* instead and specify the settings that are appropriate for your simulation or timing analysis tool. Go to MAX+PLUS II Help for detailed information on the options available in the **EDIF Netlist Writer Settings** dialog box.

      3. To generate an optional Standard Delay Format (SDF) Output File (**.sdo**), choose the **Customize** button to expand the dialog box to show all settings. Select one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**.

      The filenames of the EDIF Output File(s) and optional SDF Output File(s) are the same as the user-defined chip name(s) for the project; if no chip names exist, the Compiler assigns filenames that are based on the project name. For a multi-device project, the Compiler also generates a top-level EDIF Output File that is uniquely identified by "**_t**" appended to the project name. In addition, the Compiler automatically generates a VHDL Memory Model Output File, *<project name>***.vmo**, when it generates an EDIF Output File that contains memory (RAM or ROM).

   3. If you wish to generate VHDL Output Files (**.vho**), turn on the **VHDL Netlist Writer** command (Interfaces menu). Then choose **VHDL Netlist Writer Settings** command (Interfaces menu). Select

VHDL Output File (.vho) or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF ver. 2.1 files contain timing delay information that allows you to perform back-annotation simulation in VHDL with VITAL-compliant simulation libraries. The VHDL Output Files generated by the Compiler have the extension **.vho**, but are otherwise named in the same way as the EDIF Output Files described above.

4. If you wish to generate Verilog HDL Output Files (**.vo**), turn on the **Verilog Netlist Writer** command (Interfaces menu). Then choose **Verilog Netlist Writer Settings** command (Interfaces menu). Select *Verilog Output File (.vo)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF Output Files contain timing delay information that allows you to perform back-annotation simulation in Verilog HDL. The Verilog Output Files generated by the Compiler have the extension **.vo**, but are otherwise named in the same way as the EDIF Output Files described above.

9. To run the MAX+PLUS II Compiler, choose the **Project Save & Compile** command (File menu) or choose the **Start** button in the Compiler window.

   ☞ See step 3 for information on running MAX+PLUS II software from the command line.

10. Once you have compiled the project with the MAX+PLUS II Compiler, you can use the VHDL, Verilog HDL, or EDIF output file(s), and the optional SDF Output File(s) (**.sdo**) to perform timing analysis or timing simulation with another EDA tool. Specific instructions for some tools are described in these MAX+PLUS II ACCESS Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your EDA tool.

The MAX+PLUS II Compiler also generates a Report File (**.rpt**), a Pin-Out File (**.pin**), and one or more of the following files for device programming or configuration:

- JEDEC Files (**.jed**)
- Programmer Object Files (**.pof**)
- SRAM Object Files (**.sof**)
- Hexadecimal (Intel-format) Files (**.hex**)
- Tabular Text Files (**.ttf**)

Related Topics:

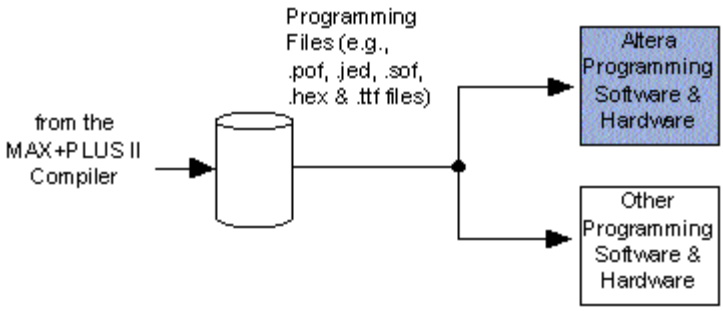
- Refer to the following sources for additional information:
  - Go to *Compiler Procedures* in MAX+PLUS II Help for information on other available Compiler settings.
  - Go to *Programmer Procedures* in MAX+PLUS II Help for instructions on creating other types of programming files and on programming or configuring Altera devices.
  - Go to Back-Annotating MAX+PLUS II Pin Assignments to Design Architect Symbols for information on back-annotating pin assignments in Mentor Graphics Design Architect schematics.
  - Go to Programming Altera Devices for information on the different programming hardware options for Altera device families.
- Go to the following topics, which are available on the web, for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware

Programming Altera Devices

Once you have successfully compiled and simulated a project with the MAX+PLUS® II software, you can program an Altera® device and test it in the target circuit. Figure 1 shows the device programming flow for MAX+PLUS II software.

Figure 1. MAX+PLUS II Device Programming Flow

Altera-provided items are shown in blue.



You can program devices with Altera programming hardware and MAX+PLUS II Programmer software installed on a 486- or Pentium-based PC or a UNIX workstation, or with programming hardware and software available from other manufacturers. Table 1 shows the available Altera programming hardware options on PCs and UNIX workstations.

Table 1. Altera Programming Hardware

Programming Hardware Option	PCs	UNIX Work-stations	MAX® 3000A Devices	Classic® & MAX 5000 Devices	MAX 7000 & MAX 7000E Devices	MAX 7000A, MAX 7000AE, MAX 7000B, MAX 7000S MAX 9000 & MAX 9000A Devices	FLEX® 6000, FLEX 6000A, FLEX 8000, FLEX 10K, FLEX 10KA, FLEX 10KB, & FLEX 10KE Devices	In-System Programming/ Configuration
Logic Programmer card, PL-MPU Master Programming Unit, and device-specific adapters	✓		✓	✓	✓	✓		
BitBlaster™ Download Cable	✓	✓	✓			✓	✓	✓
ByteBlasterMV™ Download Cable	✓		✓			✓	✓	✓
MasterBlaster™ Download Cable	✓	✓	✓			✓	✓	✓

If you wish to transfer programming files from a UNIX workstation to a PC over a network with File Transfer Protocol (FTP) or other similar transfer programs, be sure to select binary transfer mode.

Programming hardware from other manufacturers varies, but typically consists of a device connected to one of the serial ports on the workstation. Various vendors, such as Data I/O and BP Microsystems, supply hardware and software for programming Altera devices.

Related Topics:

- Go to Compiling Projects with MAX+PLUS II Software for information on creating programming files.
- Go to the following topics, which are available on the web, for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware
  - FLEX Devices
  - MAX Devices
  - Classic Device Family

Performing Timing Verification of Verilog Output Files (.vo) with MOTIVE Software

After you have compiled a project and generated a Verilog Output File (**.vo**) with the MAX+PLUS II Software, you can use Viewlogic MOTIVE to perform timing verification. The MOTIVE timing model library, **motive.lib**, provides basic primitives and the `clklock` megafunction for timing verification.

To perform timing verification for Verilog Output Files with MOTIVE software, follow these steps:

1. Set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Viewlogic Powerview Working Environment.
2. Generate a Verilog Output File by compiling your design with the MAX+PLUS II software, as described in Compiling Projects with MAX+PLUS II Software.
3. Start the MOTIVE software by typing `motive` ↵ at the UNIX prompt. The MOTIVE Session Log and Setup Advisor windows are displayed. Choose **OK**.
4. Choose **Project** on the vertical menubar in the Setup Advisor, then choose the **Name** (Select project name) tab and specify the name of the project for *Project name*. The directory in which you started MOTIVE will be selected automatically for *Current directory*. Choose **Accept**. MOTIVE then searches for the <*project name*>**.stm** file. If this is a new file, a message will appear in the Session Log window that mentions that MOTIVE found a license and the message `could not open the <project name>.stm file -- assuming a new design.`
5. Choose **Flow** from the vertical menubar, then choose the **Type** (Select flow type) tab. Select the *Using Verilog and SDF* option and choose **Accept**.
6. Choose **Options** from the vertical menubar, then choose the **Options** (Miscellaneous usage options) tab. If desired, specify a different value for the *MOTIVE analysis cycle time* option. Choose **Accept**.
7. Choose **Verilog** on the vertical menubar and specify the following Verilog HDL input options:
    1. Choose the **Translate** (Translate Verilog netlist file) tab. Specify the name of the MAX+PLUS II-generated Verilog Output File (**.vo**) for the *Verilog netlist* option. Choose the **Common Options** button to display the **Common Options** dialog box. Select the *Special Options* option and turn on the *Skip Behavioral Constructs* option. Type either `pinlist` or a period (`.`) for the *Generated pin files* option. Choose **OK** to close the **Common Options** dialog box and return to the **Translate** tab.
    2. Specify the location of the MAX+PLUS II-generated **alt_max2.vo** file for the *Vendor module definition* option. Choose the **Translate** button. The **Process Execution Log & Tips** dialog box displays the current status of the translation to **.pin** files. Choose **OK** after successful translation.
    3. Choose the **Import** (Confirm Adding hierarchical blocks) tab. Choose the **Import Blocks** button. The **MOTIVE Interaction Log & Tips** dialog displays the current import status. Choose **OK** after a successful completion.
    4. Select the **Hierarchy** (Configure hierarchy options) tab. Type the name of the rootblock for the *Rootblock of design* option, or choose the **Find Rootblock** button to display the rootblock name. Choose **Accept**.

8. Choose the **Check** (Review and/or build the netlist database) tab. Choose the **Incremental Build** button. The **MOTIVE Interaction Log & Tips** dialog displays the current build status. Choose **OK** after a successful completion.

9. Select **SDF** on the vertical menubar, then select the **Translate** (SDF model preparation) tab. Type <*project name*>`.sdo` for the *SDF file* option, making sure that you specify the **.sdo** extension. Type <*project name*>`.ctl` for the *MPP control file name*, and <*project name*>`.idd` for the *IDD file name*.

10. Choose the **Process SDF File** button.

11. If your project contains the `clklock` megafunction, use the **genmtv** utility to back-annotate the MPP Control File and to allow the MPP Control File to recognize the `clklock` function. The input to the **genmtv** utility is the Verilog netlist file (**.vo**). From the /<*working directory*>/<*project name*>/<*case name*> directory, type the following command at the UNIX prompt:

    `genmtv -v` <*project name*> ↵

12. If your project contains RAM or ROM functions and you turned on the *Flatten Bus* option the MAX+PLUS II Compiler's **Verilog Netlist Writer Settings** dialog box when you compiled your project, you must edit the **mem.lib** file, i.e., the MOTIVE Model Pre-Processor timing library file generated with the **genmtv** utility. You must remove the bracket `[ ]` characters from all occurrences of the address bus, e.g., change `A[0]` to `A0`, in both the `INPUTS` and `MIXED` sections of every RAM and ROM cell definition in **mem.lib**.

13. Select the **MPP** (MOTIVE model compilation) tab. Type <*project name*>`.ctl` for the *Control file* option. Type `/usr/maxplus2/viewlogic/library/alt_time/motive.lib /usr/maxplus2/viewlogic/library/alt_time/motive.drv` for the *Libraries* option. If the project contains memory functions, you should also specify the location of the **mem.lib** file for the *Libraries* option. Type <*project name*>`.mod` for the *Generated model file* option and <*project name*>`.rcf` for the *Revised control file* option. Choose the **RUN MMP** button. The **MOTIVE Execution Log & Tips** dialog displays and shows the current status. Choose **OK** after a successful completion.

14. Select **Save** from the File menu in the Setup Advisor to write all the selections made so far to the <*project name*>**.stm** file.

15. Select **Clock** on the vertical menubar, then choose the **File** (Check reference file and timebase options) tab. The correct name of the Clock Reference File (**.ref**) should be displayed for the *Clock reference file* option. Choose **Accept**.

    Every MOTIVE analysis requires a MOTIVE Clock Reference File. If the project is simple, you can create the file in the Setup Advisor. Otherwise, you must create the file with a text editor using MOTIVE syntax. For more information on the purpose, function, and syntax of MOTIVE Clock Reference Files, see the ***MOTIVE System*** **Reference**.

16. Choose the **Edit** (Simple clock reference generation) tab. Specify the names for the *Clock reference* and *Clock net name* options. Choose **Generate**.

17. Choose the **Check** (Choose incremental definitions) tab, then choose the **Load Clock** button.

18. Choose **Finish** from the vertical menubar, then choose the **Build** button. The **MOTIVE Interaction Log & Tips** dialog displays the current status. Choose **OK** after a successful completion.

19. Select **Save** from the File menu in the Setup Advisor.

20. In the MOTIVE Session Log window, choose **Verify** (Analyze menu) and then choose the **Execute** button to start verification. To view the output files, choose **Output Files** (View menu).

Alternatively, you can run MOTIVE analysis on the command line by following these steps:

1. Type the following commands at the UNIX prompt:

   `vtran` <*project name*>`.vo -b -h -u alt_max2.vo` ↵ (generates **.pin** files)

   `sdf2mtv` <*project name*>`.sdfo` ↵ (generates **.ctl** files)

2. If your project contains `ram`, `rom`, `dpram`, or `clklock` functions, you should also type the following commands at the UNIX prompt:

   `genmtv -v` <*project name*> ↵

   `mmp` <project name>`.ctl -l /usr/maxplus2/viewlogic/library/alt_time/motive.lib -l /usr/maxplus2/viewlogic/library/alt_time/motive/drv -l mem.lib` ↵

3. Type the following command at the UNIX prompt:

   `amtv` <*project name*>

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

MAX+PLUS II Directory Structure (Synplicity Environment)

In the MAX+PLUS® II software, a project name is the name of a top-level design file, without the filename extension. This design file can be an EDIF, Verilog HDL, or VHDL netlist file; an AHDL Text Design File (TDF); or any other MAX+PLUS II-supported design file. You can use a standard EDA tool to create an EDIF netlist file and import it into MAX+PLUS II software as an EDIF Input File (**.edf**).

Project design files and output files are stored in the project directory, with the exception of standard library functions provided by Altera or another EDA tool vendor. The MAX+PLUS II software stores the connectivity data on the links between design files in a hierarchical project in a Hierarchy Interconnect File (**.hif**) in the project directory, but refers to the entire project only by its project name. The MAX+PLUS II Compiler uses the HIF to build a single, fully flattened project database that integrates all the design files in a project hierarchy.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

MAX+PLUS II Project Directory Structure

In the MAX+PLUS ® II software, a project name is the name of a top-level design file, without the filename extension. This design file can be an EDIF, VHDL, or Verilog HDL netlist file; an Altera Hardware Description Language (AHDL) Text Design File (TDF); or any other MAX+PLUS II-supported design file. The EDIF netlist file must be created by ENWrite, Galileo Extreme, or Leonardo software and imported into MAX+PLUS II as an EDIF Input File (**.edf**). Figure 1 shows an example of a MAX+PLUS II project directory.

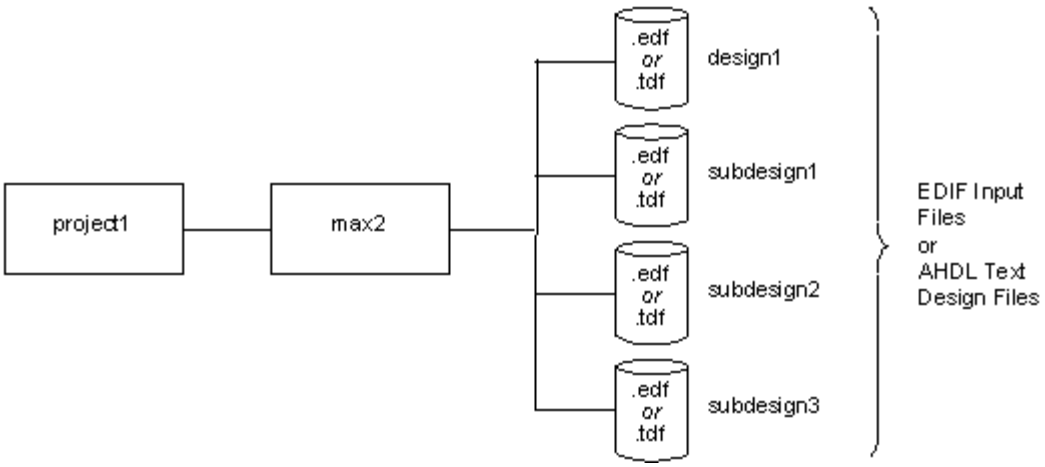


Figure 1. Sample MAX+PLUS II Project Directory

The MAX+PLUS II software stores the connectivity data on the links between design files in a hierarchical project in a Hierarchy Interconnect File (**.hif**), but refers to the entire project only by its project name. The MAX+PLUS II Compiler uses the HIF to build a single, fully flattened project database that integrates all the design files in a project hierarchy.

Related Links:

- Go to the following MAX+PLUS II ACCESS℠ Key topics for related information:
  - Local Work Area Directory Structure
  - Mentor Graphics Project Directory Structure

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

MAX+PLUS II Directory Structure

In the MAX+PLUS® II software, a project name is the name of a top-level design file, without the filename extension. This design file can be an EDIF, Verilog HDL, or VHDL netlist file; an AHDL Text Design File (TDF); or any other MAX+PLUS II-supported design file. The EDIF netlist file must be created by the **altout** or **concept2alt** utility and imported into the MAX+PLUS II software as an EDIF Input File (**.edf**).

Project design files and output files are stored in the project directory, with the exception of standard library functions provided by Altera or another EDA tool vendor. The MAX+PLUS II software stores the connectivity data on the links between design files in a hierarchical project in a Hierarchy Interconnect File (**.hif**), but refers to the entire project only by its project name. The MAX+PLUS II Compiler uses the HIF to build a single, fully flattened project database that integrates all design files in a project hierarchy.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

MAX+PLUSÂ II/Viewlogic Powerview Interface File Organization

Table 1 shows the MAX+PLUS® II/Cadence interface subdirectories that are created in the MAX+PLUS II system directory (by default, the **/usr/maxplus2** directory) during MAX+PLUS II installation. For information on the other directories that are created during MAX+PLUS II installation, see "MAX+PLUS II File Organization" in *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual.

Table 1. MAX+PLUS II Directory Organization

Directory	Description
./lmf	Contains the Altera-provided Library Mapping File, **cadence.lmf**, that maps Cadence logic functions to equivalent MAX+PLUS II logic functions.
./examples/cadence	Contains the sample files for Cadence software discussed in these ACCESS℠ Key Guidelines.
./cadence	Contains the AMPLE userware for the MAX+PLUS II/Cadence interface.
./simlib/concept/alt_max2	Contains the MAX+PLUS II primitives, including `CARRY`, `CASCADE`, `EXP`, `GLOBAL`, `LCELL`, `SOFT`, `OPNDRN`, `DFFE` (D flipflop with Clock Enable), and `DFFE6K` (D flipflop with Clock Enable and both Clear and Preset for FLEX® 6000 devices only) for use with Concept software.
./simlib/composer/alt_max2	Contains the MAX+PLUS II primitives, including `CARRY`, `CASCADE`, `EXP`, `GLOBAL`, `LCELL`, `SOFT`, `OPNDRN`, `DFFE` (D flipflop with Clock Enable), and `DFFE6K` (D flipflop with Clock Enable and both Clear and Preset for FLEX 6000 devices only) for use with Composer software.
./simlib/concept/alt_lpm	Contains the MAX+PLUS II megafunctions, including library of parameterized modules (LPM) functions, for use with Concept software.
./simlib/concept/max2sim	Contains the MAX+PLUS II/Concept simulation model library, **max2_sim**, for use with RapidSIM software.
./simlib/concept/alt_syn	Contains the MAX+PLUS II synthesis library, **alt_syn**, for use with Synergy and Concept software, and the **vlog2alt** utility.
./simlib/composer/alt_syn	Contains the MAX+PLUS II synthesis library, **alt_syn**, for use with Synergy and Composer software.
./simlib/concept/lpm_syn	Contains the Cadence LPM library, **lpm_syn**, for use with Synergy and Concept software.
./simlib/composer/lpm_syn	Contains the Cadence LPM library, **lpm_syn**, for use with Synergy and Composer software.
./simlib/concept/alt_mf	Contains the MAX+PLUS II VHDL logic function library. (`a_8count` is for the MAX® 7000 and MAX 9000 device families only.)
./simlib/concept/edifnet/templates	Contains template files for Concept directives, i.e., **global.cmd**, **compiler.cmd**, **vloglink.cmd**, **verilog.cmd**, and **master.local**.
./simlib/concept/alt_max2/verilogUdps	Contains Verilog HDL modules that are the equivalent of the primitives contained in **alt_max2** library for use with Concept software.
./simlib/composer/alt_max2/verilogUdps	Contains Verilog HDL modules that are the equivalent of the primitives contained in **alt_max2** library for use with Composer software.

./simlib/concept/alt_vtl **./simlib/composer/alt_vtl**	Contains VITAL library source files for use with Concept or Composer software.
./simlib/composer/alt_max2/verilog	Contains simulation modules for all symbols in the **alt_max2** Composer library.

Related Links:

- ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
- This manual is also available in 4 parts:
  - Preface & Section 1: MAX+PLUS II Installation
  - Section 2: MAX+PLUS II - A Perspective
  - Section 3: MAX+PLUS II Tutorial
  - Appendices, Glossary & Index

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Table 1 shows the MAX+PLUS® II/Viewlogic Powerview interface subdirectories that are created in the MAX+PLUS II system directory (by default, the **/usr/maxplus2** directory) during MAX+PLUSÂ II installation.

☞ For information on the other directories that are created during MAX+PLUSÂ II installation, see "MAX+PLUSÂ II File Organization" in *MAX+PLUSÂ II Installation* in the ***MAX+PLUSÂ II Getting Started*** manual.

Table 1. MAX+PLUSÂ II Directory Organization

Directory	Description
./lmf	Contains the Altera-provided Library Mapping File, **vwlogic.lmf**, that maps Viewlogic logic functions to equivalent MAX+PLUSÂ II logic functions.
./viewlogic	Contains the **alt_edif.cfg** EDIF configuration file that is used with the **edifneti** utility. Also contains the library and sample subdirectories.
./viewlogic/examples	Contains the sample Viewlogic designs.
./viewlogic/library/max2sim	Contains the MAX+PLUSÂ II simulation model library (**max2_sim**) for use in ViewSim software.
./viewlogic/library/alt_max2	Contains MAX+PLUSÂ II primitives (`EXP`, `GLOBAL`, `LCELL`, `SOFT`, `CARRY`, `CASCADE`, `DFFE`, `DFFE6K`, and `OPNDRN`), macrofunctions (`a_8fadd`, `a_8mcomp`, `a_8count`, `a_81mux`), and megafunctions (`clklock`) for use in ViewDraw schematics. These logic functions support specific architectural features of AlteraÂ® devices. The **alt_max2** library also contains modified versions of the ViewDraw primitives that use tri-state buffers, because these primitives require special handling in the MAX+PLUSÂ II /Viewlogic Powerview interface.
./viewlogic/library/synlib	Contains the Altera-provided synthesis library **altera**, which includes MAX+PLUSÂ II primitives, the **altera.sml** file, a **sym** directory, and a **wir** directory for use with

	ViewSynthesis software.
./viewlogic/library/alt_mf	Contains the VHDL models for the MAX+PLUSÂ II primitives (`EXP`, `GLOBAL`, `LCELL`, `SOFT`, `CARRY`, `CASCADE`, `DFFE`, and `OPNDRN`), macrofunctions (`clklock`) for use with ViewSynthesis software, the Vantage VHDL Analyzer software, and the VHDL source files. These logic functions are used to maintain portability to other architectures.
./viewlogic/library/alt_time	Contains MOTIVE timing models for MAX+PLUSÂ II logic functions (**motive.lib**), including the `clklock` megafunction, and MAX+PLUSÂ II driver models (**motive.drv**).
./viewlogic/library/alt_vtl	Contains the VHDL source files for the VITAL 3.0-compliant library. This library is available for ViewSim software.
./viewlogic/bin	Contains all MAX+PLUSÂ II, Viewlogic, and interface-related scripts.
./viewlogic/standard	Contains all standard **.ini** files and standard tools.

Related Links:

- Go to the following topics for additional information:
  - ***MAX+PLUSÂ II Getting Started*** version 8.1 (5.4 MB)
  - This manual is also available in 4 parts:
    - Preface & Section 1: MAX+PLUSÂ II Installation
    - Section 2: MAX+PLUSÂ II - A Perspective
    - Section 3: MAX+PLUSÂ II Tutorial
    - Appendices, Glossary & Index

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

The vdpath & mega_lpm Libraries

The library of parameterized modules (LPM) 2.1.0 standard defines a set of parameterized functions and their corresponding representations in an EDIF netlist file. These logic functions allow you to create and functionally simulate an LPM-based design without targeting a specific device family. After the design is completed, you can target the design to any device family.

When the MAX+PLUS® II software processes projects that include Viewlogic-provided **vdpath** LPM functions, it uses functions from the Altera-provided **mega_lpm** library. This library includes all standard LPM functions except the truth table, finite state machine, and pad functions. Altera does not directly support the `lpm_ram_dq`, `lpm_ram_io`, and `lpm_rom` functions. Refer to Instantiating RAM & ROM Functions in Viewlogic Powerview Designs for instructions on instantiating RAM and ROM functions.

☞ Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu for more information about LPM functions.

Support

Intel Community Forums provides a place to ask and answer questions about Intel products.

Intel does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Instantiating the clklock Megafunction in VHDL or Verilog HDL

MAX+PLUS® II interfaces with other EDA tools support the `clklock` phase-locked loop megafunction, which can be used with some FLEX® 10K devices, with the **gencklk** utility. Type `gencklk -h` ↵ at the UNIX prompt to display information on how to use this utility. The **gencklk** utility generates VHDL or Verilog HDL functional simulation models and a VHDL Component Declaration template file (**.cmp**).

The **gencklk** utility allows parameters for the `clklock` function to be passed from the VHDL or Verilog HDL file to EDIF netlist format. The **gencklk** utility embeds the parameter values in the `clklock` function name; therefore, the values do not need to be declared explicitly.

To instantiate the `clklock` megafunction in VHDL or Verilog HDL, go through the following steps:

1. Type the following command at the UNIX prompt to generate the `clklock_`*x*`_`*y* file, where *x* is the ClockBoost value and *y* is the input frequency in MHz:

   ✔ Type `gencklk` <*ClockBoost*> <*input frequency*> `-vhdl` ↵ for VHDL designs.

   or:

   ✔ Type `gencklk` <*ClockBoost*> <*input frequency*> `-verilog` ↵ for Verilog HDL designs.

   Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu for more information on the `clklock` megafunction.

2. Create a design file that instantiates the `clklock_`*x*`_`*y* function. The **gencklk** utility automatically generates a VHDL Component Declaration template in the **clklock_*x*_*y*.cmp** file that you can incorporate into a VHDL design file.

Figures 1 and 2 show a `clklock` function with <*ClockBoost*> = 2 and <*input frequency*> = 40 MHz instantiated in VHDL and Verilog HDL design files, respectively.

Figure 1. VHDL Design File with clklock Instantiation (count8.vhd)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY altera;
USE altera.maxplus2.all;     --  Include Altera Component Declarations



ENTITY count8 IS
   PORT (a    : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
         ldn  : IN STD_LOGIC;
         gn   : IN STD_LOGIC;



dnup : IN STD_LOGIC;
         setn : IN STD_LOGIC;
         clrn : IN STD_LOGIC;
         clk  : IN STD_LOGIC;
```

```
co   : OUT STD_LOGIC;
        q    : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END count8;



ARCHITECTURE structure OF count8 IS
   signal clk2x : STD_LOGIC;



COMPONENT clklock_2_40
   PORT (
      INCLK : IN STD_LOGIC;
      OUTCLK : OUT STD_LOGIC
   );
END COMPONENT;



BEGIN
   u1: clklock_2_40
   PORT MAP (inclk=>clk, outclk=>clk2x);



u2: a_8count
   PORT MAP (a=>a(0), b=>a(1), c=>a(2), d=>a(3),
             e=>a(4), f=>a(5), g=>a(6), h=>a(7),
             clk=>clk2x,
             ldn=>ldn,
             gn=>gn,



dnup=>dnup,
             setn=>setn,
             clrn=>clrn,



qa=>q(0), qb=>q(1), qc=>q(2), qd=>q(3),
             qe=>q(4), qf=>q(5), qg=>q(6), qh=>q(7),
             cout=>co);
 END structure;
```

Figure 2. Verilog HDL Design File with clklock Instantiation (count8.v)

```
`timescale 1ns / 10ps
module count8 (a, ldn, gn, dnup, setn, clrn, clk, co, q);
output          co;
output[7:0]     q;



input[7:0]      a;
input           ldn, gn,dnup, setn, clrn, clk;
wire            clk2x;



clklock_2_40 u1 (.inclk(clk), .outclk(clk2x) );
A_8COUNT u2 (.A(a[0]), .B(a[1]), .C(a[2]), .D(a[3]), .E(a[4]), .F(a[5]),



.G(a[6]), .H(a[7]), .LDN(ldn), .GN(gn), .DNUP(dnup),
             .SETN(setn), .CLRN(clrn), .CLK(clk2x), .QA(q[0]), .QB(q[1]),
             .QC(q[2]), .QD(q[3]), .QE(q[4]), .QF(q[5]), .QG(q[6]),
             .QH(q[7]), .COUT(co) );
```

```
endmodule
```

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Assigning Pins

You can assign a single logic function to a specific pin or logic cell (including I/O cells and embedded cells) within a chip, and assign one or more functions to a specific chip. A chip is a group of logic functions defined as a single, named unit, which can be assigned to a specific device.

You can assign a signal to a particular pin to ensure that the signal is always associated with that pin, regardless of future changes to the project. If you wish to set and maintain the performance of your project, assigning logic to a specific logic cell within a chip can minimize timing delays. In a project that is partitioned among multiple devices, you can assign logic functions that must be kept together in the same device to a chip. Chip assignments allow you to split a project so that only a minimum number of signals travel between devices, and to ensure that no unnecessary device-to-device delays exist on critical timing paths. You can assign a chip to a device in some EDA tools or in the MAX+PLUS® II software.

Use the following syntax for chip, pin, and logic cell assignments:

- To assign a logic function to a chip:

  `CHIP_PIN_LC=`*<chip name>*

  For example: `CHIP_PIN_LC=chip1`

- To assign a pin number within a chip:

  `CHIP_PIN_LC=`*<chip name>*`@`*<pin number>*

  For example: `CHIP_PIN_LC=chip1@K2`

- To assign a logic cell, I/O cell, or embedded cell number:

  `CHIP_PIN_LC=`*<chip name>*`@LC`*<logic cell number>*

  `CHIP_PIN_LC=`*<chip name>*`@IOC`*<I/O cell number>*

  `CHIP_PIN_LC=`*<chip name>*`@EC`*<embedded cell number>*

  For example: `CHIP_PIN_LC=chip1@LC44`

Related Links:

- Go to "Devices & Adapters" and "Assigning a Device" in MAX+PLUS II Help for information on device pin-outs and assigning devices, respectively, in the MAX+PLUS II software.
- Go to Back-Annotating MAX+PLUS II Pin Assignments to Design Architect Symbols for information on back-annotating pin assignments in Mentor Graphics Design Architect schematics.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

You can assign a single port to a specific pin to ensure that the signal is always associated with that pin, regardless of future changes to the project. You can specify pins in VHDL or Verilog HDL designs, or in a Synplify Design Constraints File (**.sdc**). If you add timing constraints or resource assignments in a separate Synplify Design Constraints File (**.sdc**), you must add the Synplify Design Constraints File (**.sdc**) to the project by adding it to the *Source Files* list in the Synplify window.

☞ If your design uses resource assignment attributes that you wish to pass to the MAX+PLUS® II software, you should save your file in EDIF netlist file format. See Entering Resource Assignments for more information.

VHDL Syntax

Use the following syntax to assign a pin in VHDL:

```
attribute altera_chip_pin_lc : string;
attribute altera_chip_pin_lc of <port name> : signal is "@<pin number(s)>"
```

Example:

```
attribute altera_chip_pin_lc : string;
attribute altera_chip_pin_lc of result : signal is
        "@17, @166, @191, @152, @15, @148, @147, @149"
```

Verilog HDL Syntax

Use the following syntax to assign a pin in Verilog HDL:

```
<port name> /* synthesis altera_chip_pin_lc="@<pin number(s)>" */;
```

Example:

```
output [7:0] sum /* synthesis altera_chip_pin_lc="@17, @166, @191, @152, Â¥
        @15, @148, @147, @149" */;
```

Synplify Design Constraints File Syntax

Use the following syntax to assign a pin in a Synplify Design Constraints file:

```
define_attribute <port name> altera_chip_pin_lc "@<pin number>"
```

Example:

```
define_attribute {DATA0[7:0]} altera_chip_pin_lc "@115,@116,@117,
@118,@119,@120,@121,@122"
```

Related Links:

- Refer to the following sources for related information:
  - Go to Entering Resource Assignments in these MAX+PLUS II ACCESS℠ Key topics for information on entering other types of assignments.
  - Go to "Devices & Adapters" and "Assigning a Device" in MAX+PLUS II Help for information on device pin-outs and assigning devices, respectively, in the MAX+PLUS II software.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Assigning Pins, Logic Cells & Chips

You can assign a single logic function to a specific pin or logic cell (including I/O cells and embedded cells) within a chip, and assign one or more functions to a specific chip. A chip is a group of logic functions defined as a single, named unit, which can be assigned to a specific device.

You can assign a signal to a particular pin to ensure that the signal is always associated with that pin, regardless of future changes to the project. If you wish to set and maintain the performance of your project, assigning logic to a specific logic cell within a chip can minimize timing delays. In a project that is partitioned among multiple devices, you can assign logic functions that must be kept together in the same device to a chip. Chip assignments allow you to split a project so that only a minimum number of signals travel between devices, and to ensure that no unnecessary device-to-device delays exist on critical timing paths. You can assign a chip to a device in MAX+PLUS® II software.

To make pin, logic cell, and chip assignments, use the `set_attribute` command at a **dc_shell** prompt. Before using the `set_attribute` command, add the following line to your **.synopsys_dc.setup** file:

```
edifout_write_properties_list= {LOGIC_OPTION, CLIQUE, CHIP_PIN_LC}
↵
```

Table 1 shows the syntax to use for chip, pin, and logic cell assignments:

Table 1. Commands for Chip, Pin, & Logic Cell Assignments

Assignment Type	Command to Type *Note (1)*
Chip	`set_attribute find (`*<design object>*`, (`*<instance name>*`)) "CHIP_PIN_LC" -type string "`*<chip name>*`" ↵`
Pin	`set_attribute find (`*<design object>*`, (`*<instance name>*`)) "CHIP_PIN_LC" -type string "`*<chip name>*`@`*<pin number>*`" ↵`
Logic cell number	`set_attribute find (`*<design object>*`, (`*<instance name>*`)) "CHIP_PIN_LC" -type string "`*<chip name>*`@LC`*<logic cell number>*`" ↵`
I/O cell number	`set_attribute find (`*<design object>*`, (`*<instance name>*`)) "CHIP_PIN_LC" -type string "`*<chip name>*`@IOC`*<I/O cell number>*`" ↵`
Embedded cell number	`set_attribute find (`*<design object>*`, (`*<instance name>*`)) "CHIP_PIN_LC" -type string "`*<chip name>*`@EC`*<embedded cell number>*`" ↵`

Note:

1. In this table, *<design object>* represents ports, references, cells, nets, or pins.

Examples:

```
set_attribute find (cell, (U1)) "CHIP_PIN_LC" -type string "chip1" ↵

set_attribute find (cell, (U1)) "CHIP_PIN_LC" -type string "chip1@K2" ↵

set_attribute find (cell, (U1)) "CHIP_PIN_LC" -type string "chip1@LC44" ↵
```

Related Topics:

- Go to "Devices & Adapters" and "Assigning a Device" in MAX+PLUS II Help for information on device pin-outs and assigning devices, respectively, in the MAX+PLUS II software.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Altera Post-Synthesis Libraries

The **/usr/maxplus2/synopsys/library/alt_post/syn/lib** directory contains the post-synthesis library for technology mapping and timing back-annotation. The Altera® -provided **alt_vtl.db** file in this library contains over three dozen MAX+PLUS® II -generated logic functions.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

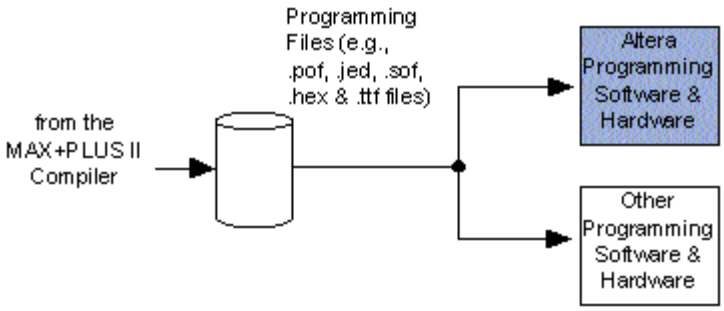
Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Programming Altera Devices

Once you have successfully compiled and simulated a project with the MAX+PLUS II software, you can program an Altera device and test it in the target circuit. Figure 1 shows the device programming flow for MAX+PLUS II software.

Figure 1. MAX+PLUS II Device Programming Flow

Altera-provided items are shown in blue.



You can program devices with Altera programming hardware and MAX+PLUS II Programmer software installed on a 486- or Pentium-based PC or a UNIX workstation, or with programming hardware and software available from other manufacturers. Table 1 shows the available Altera programming hardware options on PCs and UNIX workstations.

Table 1. Altera Programming Hardware

Programming Hardware Option	PCs	UNIX Work-stations	MAX® 3000A Devices	Classic® & MAX 5000 Devices	MAX 7000 & MAX 7000E Devices	MAX 7000A, MAX 7000AE, MAX 7000B, MAX 7000S MAX 9000 & MAX 9000A Devices	FLEX® 6000, FLEX 6000A, FLEX 8000, FLEX 10K, FLEX 10KA, FLEX 10KB, & FLEX 10KE Devices	In-System Programming/ Configuration
Logic Programmer card, PL-MPU Master Programming Unit, and device-specific adapters	✓		✓	✓	✓	✓		
BitBlaster	✓	✓	✓			✓	✓	✓

Download Cable							
ByteBlasterMV Download Cable	✓		✓		✓	✓	✓
MasterBlaster™ Download Cable	✓	✓	✓		✓	✓	✓

If you wish to transfer programming files from a UNIX workstation to a PC over a network with File Transfer Protocol (FTP) or other similar transfer programs, be sure to select binary transfer mode.

Programming hardware from other manufacturers varies, but typically consists of a device connected to one of the serial ports on the workstation. Various vendors, such as Data I/O and BP Microsystems, supply hardware and software for programming Altera devices.

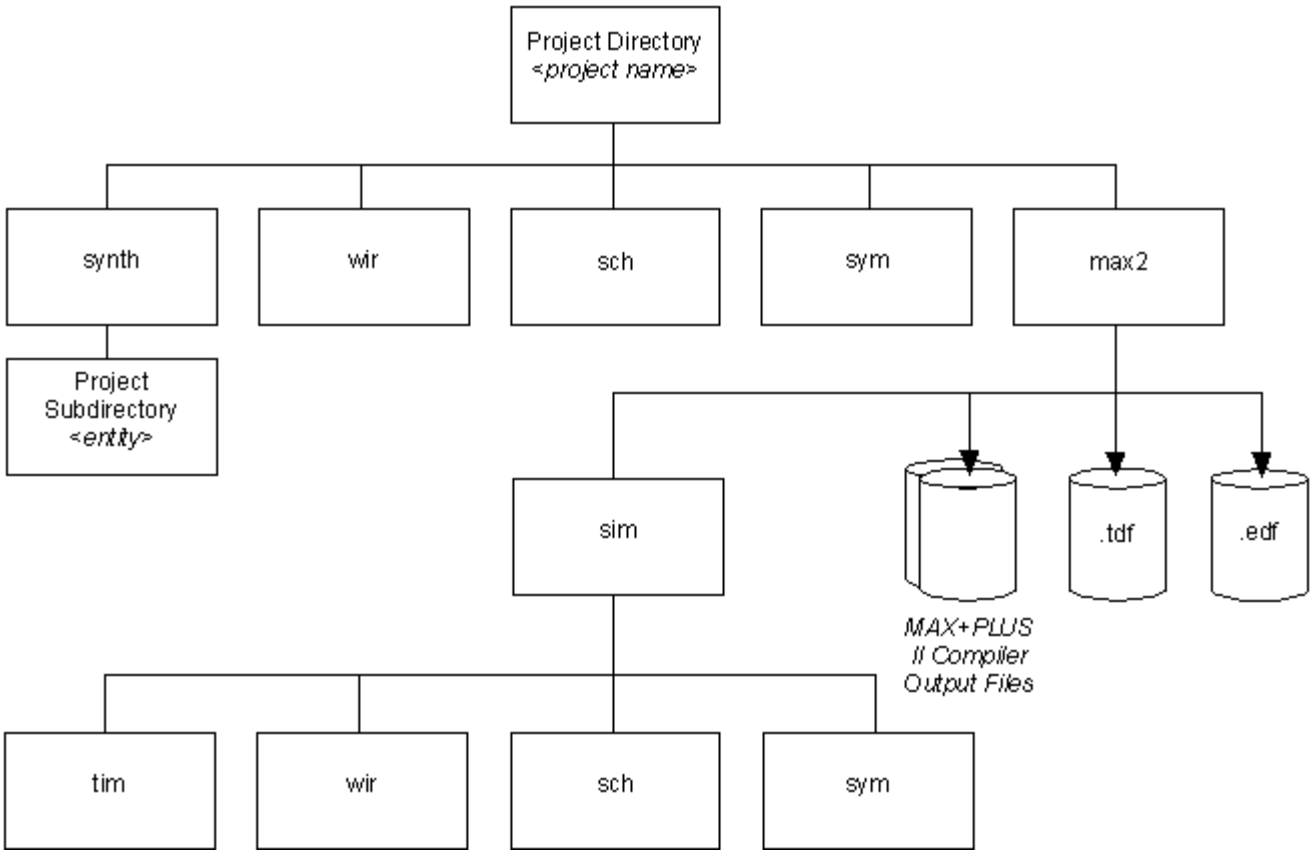
Related Links

- Compiling Projects with MAX+PLUS II Software
- MAX+PLUS II Development Software
- Altera Programming Hardware

MAX+PLUS II/Viewlogic Powerview Project File Structure

In the MAX+PLUS® II software, a project name is the name of a top-level design file, without the filename extension. This design file can be an EDIF, Verilog HDL, or VHDL netlist file; an Altera® Hardware Description Language (AHDL) TDF; or any other MAX+PLUS II- supported design file. The EDIF netlist file must be created by Powerview and imported into the MAX+PLUS II software as an EDIF Input File (**.edf**). Figure 1 shows an example of MAX+PLUS II project directory structure that includes Powerview-generated files.

Figure 1. Sample MAX+PLUS II Project Organization



The MAX+PLUS II software stores the connectivity data on the links between design files in a hierarchical project in a Hierarchy Interconnect File (**.hif**), but refers to the entire project only by its project name. The MAX+PLUS II Compiler uses the HIF to build a single, fully flattened project database that integrates all the design files in a project hierarchy.

Unlike Powerview, the MAX+PLUS II software does not automatically create a project directory when you create a project. A single directory can contain several MAX+PLUS II design files, and you can specify any one of the designs in the directory as a project in the MAX+PLUS II software.

Viewlogic Powerview Local Work Area Structure

When you create a project with the Powerview Cockpit's **Create** command (Project menu), the project directory is

created. You should generate design files and functional simulation files under this directory. A **max2** subdirectory is automatically created under your current project directory when you generate an EDIF file from your schematic or VHDL file. The ***<project name>*.edf** file is stored in the **max2** subdirectory. All MAX+PLUS® II Compiler output files are created in the **/*<project name>*/max2** subdirectory.

☞ ViewDraw files are identified by their directories and not by their extensions, so it is easy to overwrite files unintentionally. To avoid overwriting files, Altera recommends that you create a new project directory, ***<project name>*/max2/sim**, where you can generate all the files needed for simulation.

ViewDraw Project File Structure

Each ViewDraw project directory contains three subdirectories: **wir**, **sch**, and **sym**. See Table 1.

Table 1. ViewDraw Subdirectories

Directory	Topics
./wir	Wirelist files that contain connectivity information for a particular logic block
./sch	Schematics that contain logic
./sym	Symbol files that are the ViewDraw graphical representation of the logic blocks

Each file type uses the filename extension **.1**. Different file types are distinguished only by their directory: **/lib/wir/**<*project name*>**.1** is a wirelist file; **/lib/sch/**<*project name*>**.1** is the corresponding schematic file; and **/lib/sym/**<*project name*>**.1** is the corresponding symbol.

VHDL Project File Structure

Each VHDL project directory contains three subdirectories. See Table 2.

Table 2. VHDL Subdirectories

Directory	Topics
./synth	All synthesis-related files and directories
./synth/<*entity*>	Four types of files: <*entity*>**.pdf**, <*entity*>**.opt**, <*entity*>**.sta**, and <*entity*>**.gnl**
./wir	Wirelist for synthesized VHDL modules

☞ For each VHDL entity in the design, there is a corresponding **./synth/<*entity*>** directory.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Performing a Pre-Routing or Functional Simulation with VSS Software

After you have synthesized and optimized a VHDL or Verilog HDL design with the Design Compiler or FPGA Compiler software, you can perform a pre-routing or functional simulation with the Synopsys VHDL System Simulator (VSS) software.

To perform a pre-routing/functional simulation, follow these steps:

- Be sure to set up the working environment correctly, as described in the following topics:
  - Setting Up the MAX+PLUS II/Synopsys Working Environment
  - Setting Up Design Compiler & FPGA Compiler Configuration Files
  - Setting Up the DesignWare Interface
  - Setting Up VSS Configuration Files
  - Create a VHDL or Verilog HDL design file that follows the guidelines described in one of the following topics:
    - Creating VHDL Designs for Use with MAX+PLUS II Software
    - Creating Verilog HDL Designs for Use with MAX+PLUS II Software
    - Synthesize and optimize your design with the Design Compiler or FPGA Compiler, as described in Synthesizing & Optimizing VHDL & Verilog HDL Files with Design Compiler or FPGA Compiler Software.
    - Save your design as a VHDL Design File (**.vhd**)

      VSS requires each architecture/entity pair in a VHDL Design File to have a configuration. The Configuration Declaration is necessary for simulation, but not for synthesis.

    - Use VSS and one of the Altera pre-routing functional simulation libraries to simulate the design.
    - When you are ready to compile your project with MAX+PLUS II software, save the design as an EDIF netlist file (**.edf**), then process it as described in Compiling Projects with MAX+PLUS II Software.

Related Links

- ***VHDL System Simulator Core Programs Manual*** for more information about VSS
- Performing a Timing Simulation with VSS Software

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Performing a Timing Simulation with VSS Software

Once the MAX+PLUS® II software has compiled a project and generated a VHDL Output File (**.vho**) and an optional Standard Delay Format (SDF) Output File (**.sdo**), you can perform timing simulation with the Synopsys VHDL Simulator Software (VSS).

To simulate a VHDL Output File with VSS, follow these steps:

Be sure to set up the working environment correctly, as described in the following topics:

- Setting Up the MAX+PLUS II/Synopsys Working Environment
- Setting Up Design Compiler & FPGA Compiler Configuration Files
- Setting Up the DesignWare Interface
- Setting Up VSS Configuration Files

1. Generate a VHDL Output File (**.vho**) and an optional SDF Output File (**.sdo**), as described in Compiling Projects with MAX+PLUS II Software.
2. (Optional) Analyze the VITAL 95-compliant **alt_vtl** library , then back-annotate timing information through the SDF Output File:
    1. Use the **analyze_vss** script to analyze the **alt_vtl** Post-Routing Timing Simulation library, as described in Setting Up VSS Configuration Files.
    2. Enter the following command to back-annotate timing information through the SDF Output File:

        ```
        vhdlsim -sdf_top /<design name>/<design name> -sdf
        <design name>.sdo ↵
        ```
3. Simulate the VHDL Output File with the VSS software.

Related Topics:

- Go to the ***VSS User's Guide*** for more details on post-routing simulation.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Using Synopsys PrimeTime & MAX+PLUS II Software

The following topics describe how to use the Synopsys PrimeTime and MAX+PLUS® II software. Click on one of the following topics for information:

This file is suitable for printing only. It does not contain hypertext links that allow you to jump from topic to topic.

Setting Up the MAX+PLUS II/Synopsys Working Environment

- Software Requirements
- MAX+PLUS II/Synopsys Interface File Organization
- MAX+PLUS II Project File Structure

Timing Verification

- Timing Verification Flow
- Preparing Files for Timing Verification with PrimeTime Software using the **genpt** Utility

Related Topics:

- Go to the following topics in these MAX+PLUS II ACCESS℠ Key topics for related information:
  - Compiling Projects with MAX+PLUS II Software
  - Programming Altera Devices
  - Using Synopsys Design Compiler or FPGA Compiler & MAX+PLUS II Software
  - Using Synopsys FPGA Express and MAX+PLUS II Software
  - Using Synopsys VSS & MAX+PLUS II Software
- Go to the following topics, which are available on the web, for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware
  - Synopsys web site (http://www.synopsys.com)

Setting Up the MAX+PLUS II/Synopsys Working Environment

To use the MAX+PLUS® II software with Synopsys software, you must first install the MAX+PLUS II software, then establish an environment that facilitates entering and processing designs by modifying your Synopsys configuration files. The MAX+PLUS II/Synopsys interface is installed automatically when you install the MAX+PLUS II software on your workstation. Go to *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual for more information on installation and details on the directories that are created during MAX+PLUS II installation. Go to MAX+PLUS II/Synopsys Interface File Organization for information about the MAX+PLUS II/Synopsys directories that are created during MAX+PLUS II installation.

☞ The information presented here assumes that you are using C shell and that your MAX+PLUS II system directory is **/usr/maxplus2**. If not, you must use the appropriate syntax and procedures to set environment variables for your shell.

To set up your working environment for the MAX+PLUS II/Synopsys interface, follow these steps:

1. Ensure that you have correctly installed the MAX+PLUS II and Synopsys software versions described in the MAX+PLUS II/Synopsys Software Requirements.
2. Add technology, synthetic, and link library settings to your **.synopsys_dc.setup** configuration file, as described in Setting Up Design Compiler & FPGA Compiler Configuration Files.

   ☞ To use the DesignWare interface with FLEX® 6000, FLEX 8000, and FLEX 10K devices, follow the steps in Setting Up the DesignWare Interface.

3. Add simulation library settings to your **.synopsys_vss.setup** file, and analyze the libraries, as described in Setting Up VSS Configuration Files.
4. Add the **/usr/maxplus2/bin** directory to the `PATH` environment variable in your **.cshrc** file in order to run the MAX+PLUS II software.
   `$ALT_HOME`**/synopsys/bin**

Related Topics:

- Go to the following topics, which are available on the web, for additional information:
  - FLEX Devices
  - ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
  - This manual is also available in 4 parts:
    - Preface & Section 1: MAX+PLUS II Installation
    - Section 2: MAX+PLUS II - A Perspective
    - Section 3: MAX+PLUS II Tutorial
    - Appendices, Glossary & Index

MAX+PLUS II/Synopsys Interface File Organization

Table 1 shows the MAX+PLUS® II /Synopsys interface subdirectories that are created in the MAX+PLUS II system directory (by default, the /usr/maxplus2 directory) during the MAX+PLUS II software installation. For information on the other directories that are created during the MAX+PLUS II software installation, see "MAX+PLUS II File Organization" in MAX+PLUS II Installation in the MAX+PLUS II Getting Started manual.

☞ You must add the **/usr/maxplus2/bin** directory to the `PATH` environment variable in your **.cshrc** file in order to run the MAX+PLUS II software.

Table 1. MAX+PLUS II Directory Organization

Directory	Description
./synopsys/bin	Contains script programs to convert Synopsys timing constraints into MAX+PLUS II Assignment & Configuration File (**.acf**) format, and to analyze VHDL System Simulator simulation models.
./synopsys/config	Contains sample **.synopsys_dc.setup** and **.synopsys_vss.setup** files.
	Contains sample files, including those discussed in these ACCESS Key

./synopsys/examples	Guidelines.
./synopsys/library/alt_pre/<*device family***>/src**	Contains VHDL simulation libraries for functional simulation of VHDL projects.
./synopsys/library/alt_pre/verilog/src	Contains the Verilog HDL functional simulation library for Verilog HDL projects.
./synopsys/library/alt_pre/vital/src	Contains the VITAL 95 simulation library. You use this library when you perform functional simulation of the design before compiling it with the MAX+PLUS II software.
./synopsys/library/alt_syn//<*device family***>/lib**	Contains interface files for the MAX+PLUS II/Synopsys interface. Technology libraries in this directory allow the Design Compiler and FPGA Compiler to map designs to Altera® device architectures.
./synopsys/library/alt_mf/src	Contains behavioral VHDL models of some Altera macrofunctions, along with their component declarations. The `a_81mux`, `a_8count`, `a_8fadd`, and `a_8mcomp` macrofunctions are currently supported. Libraries in this directory allow you to instantiate, synthesize, and simulate these macrofunctions.
./synopsys/library/alt_post/syn/lib	Contains the post-synthesis library for technology mapping.
./synopsys/library/alt_post/sim/src	Contains the VHDL source files for the VITAL 95-compliant library. You use this library when you perform simulation of the design after compiling it with the MAX+PLUS II software.

Related Topics:

- Go to the following topics, which are available on the web, for additional information:
  - ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
  - This manual is also available in 4 parts:
    - Preface & Section 1: MAX+PLUS II Installation
    - Section 2: MAX+PLUS II - A Perspective
    - Section 3: MAX+PLUS II Tutorial
    - Appendices, Glossary & Index

MAX+PLUS II Project File Structure

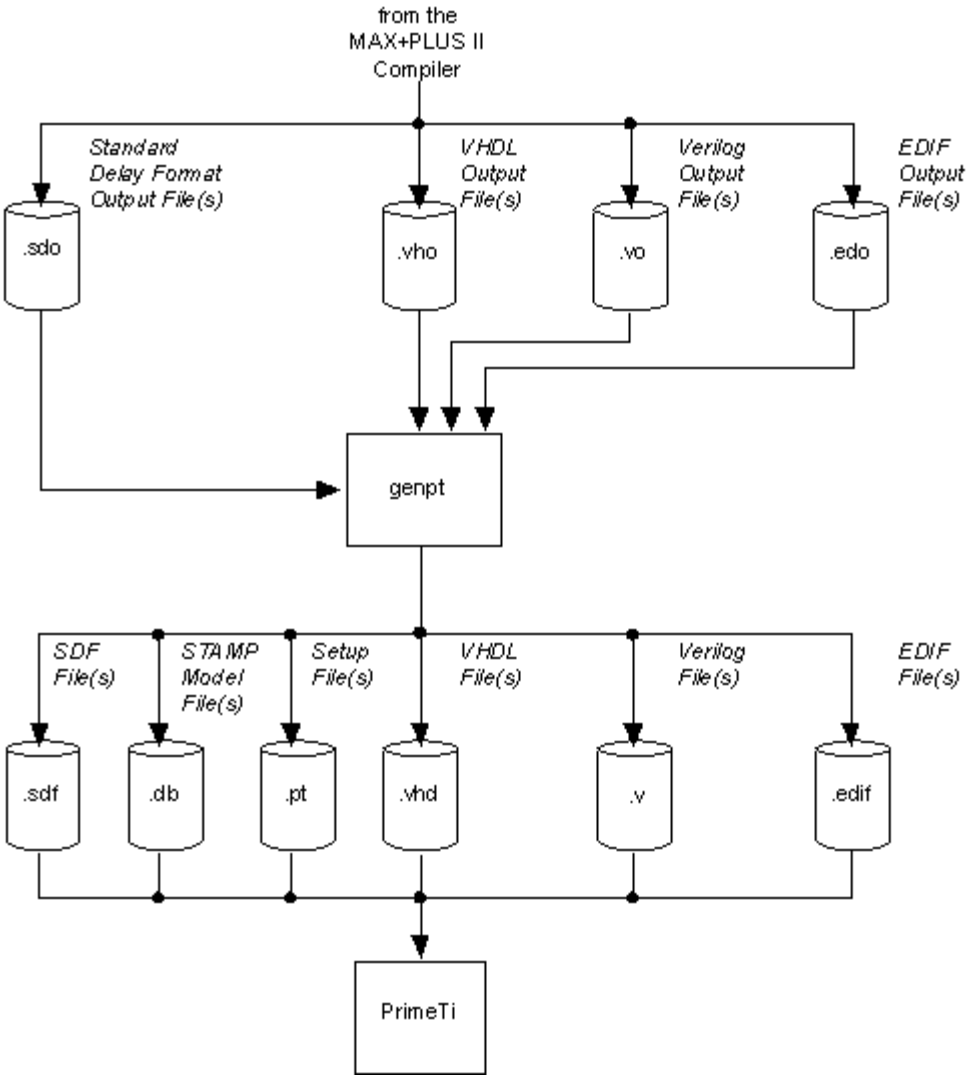
In MAX+PLUS® II, a project name is the name of a top-level design file, without the filename extension. This design file can be an EDIF, Verilog HDL, or VHDL netlist file; an AHDL™ TDF; or any other MAX+PLUS II-supported design file. The EDIF netlist file must be created by Synopsys and imported into MAX+PLUS II as an EDIF Input File.

MAX+PLUS II stores the connectivity data on the links between design files in a hierarchical project in a Hierarchy Interconnect File (**.hif**), but refers to the entire project only by its project name. The MAX+PLUS II Compiler uses the HIF to build a single, fully flattened project database that integrates all the design files in a project hierarchy.

MAX+PLUS II/Synopsys PrimeTime Timing Verification Flow

Figure 1 shows the project timing verification flow for the MAX+PLUS® II/Synopsys PrimeTime interface.

Figure 1. MAX+PLUS II/Synopsys PrimeTime Project Timing Verification Flow



Preparing Files for Timing Verification with PrimeTime Software Using the genpt Utility

After you have compiled a project and generated an EDIF Output File (**.edo**), Verilog Output File (**.vo**), or VHDL Output File (**.vho**) with the MAX+PLUS® II software, you can use Synopsys PrimeTime software to perform timing verification. The Altera-provided **genpt** utility converts EDIF, Verilog HDL, and VHDL output files for use with Synopsys PrimeTime software.

To prepare MAX+PLUS II-generated EDIF, Verilog HDL, or VHDL output files for timing verification with the Synopsys PrimeTime software, follow these steps:

1. Set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Synopsys Working Environment . Make sure that you have specified the correct path of your local Perl executable, as described in step 4 of that procedure, and that the path in the **genpt** utility points to that executable.
2. Generate an EDIF Output File (**.edo**), Verilog Output File (**.vo**), or VHDL Output File (**.vho**) and a Standard Delay Format (SDF) Output File (**.sdo**) by compiling your design with the MAX+PLUS II software, as described in Compiling Projects with MAX+PLUS II Software .
3. Use the **genpt** utility to convert the EDIF, Verilog HDL, or VHDL output file(s) to PrimeTime-compatible files by typing the following command at the UNIX prompt:
    - A PrimeTime-compatible Verilog HDL file <*design name*>**_pt.v**
    - A VHDL file <*design name*>**_pt.vhd** or an EDIF netlist file <*design name*>**_pt.edif**
    - An SDF file <*design name*>**_pt.sdf**.

`genpt (-verilog | -vhdl | -edif)` <*design name*> `[`<*output netlist filename*>`]` ↵

where <*design name*> is the name of the MAX+PLUS II-generated output file, without the extension. For example, you can type `genpt -vhdl fifo` ↵ at the UNIX prompt to convert MAX+PLUS II-generated **fifo.vhd** and **fifo.sdo** files into PrimeTime-compatible VHDL and SDF files.

Based on your settings, the **genpt** utility generates the following files:

If the project contains RAM, ROM, dual-port RAM, or `clklock` functions, the **genpt** utility generates a <*design name*>_<*type*>**.db** file, where <*type*> is `ram`, `rom`, `dpram`, or `cklk`, which contains compiled STAMP library cell models for the PrimeTime software. The **genpt** utility also generates a <*design name*>**_setup.pt** PrimeTime setup file, which contains PrimeTime setup commands for compiling generated STAMP models and for reading in the EDIF, Verilog, or VHDL file and the SDF file.

4. Start the PrimeTime software by typing `primetime` ↵ at the UNIX prompt. You can also type `pt_shell` ↵ at the UNIX prompt to run the PrimeTime software in command-line mode.
5. Source the <*design name*>**_setup.pt** PrimeTime setup file. Refer to Synopsys PrimeTime documentation for information on how to perform timing verification with the PrimeTime software.

Preparing Files for Timing Verification with PrimeTime Software Using the genpt Utility

After you have compiled a project and generated an EDIF Output File (**.edo**), Verilog Output File (**.vo**), or VHDL Output File (**.vho**) with the MAX+PLUS® II software, you can use Synopsys PrimeTime software to perform timing verification. The Altera-provided **genpt** utility converts EDIF, Verilog HDL, and VHDL output files for use with Synopsys PrimeTime software.

To prepare MAX+PLUS II-generated EDIF, Verilog HDL, or VHDL output files for timing verification with the Synopsys PrimeTime software, follow these steps:

1. Set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Synopsys Working Environment. Make sure that you have specified the correct path of your local Perl executable, as described in step 4 of that procedure, and that the path in the **genpt** utility points to that executable.

2. Generate an EDIF Output File (**.edo**), Verilog Output File (**.vo**), or VHDL Output File (**.vho**) and a Standard Delay Format (SDF) Output File (**.sdo**) by compiling your design with the MAX+PLUS II software, as described in Compiling Projects with MAX+PLUS II Software.

3. Use the **genpt** utility to convert the EDIF, Verilog HDL, or VHDL output file(s) to PrimeTime-compatible files by typing the following command at the UNIX prompt:

   `genpt (-verilog | -vhdl | -edif)` <*design name*> `[`<*output netlist filename*>`]` ↵

   where <*design name*> is the name of the MAX+PLUS II-generated output file, without the extension. For example, you can type `genpt -vhdl fifo` ↵ at the UNIX prompt to convert MAX+PLUS II-generated **fifo.vhd** and **fifo.sdo** files into PrimeTime-compatible VHDL and SDF files.

   Based on your settings, the **genpt** utility generates the following files:

   - A PrimeTime-compatible Verilog HDL file <*design name*>**_pt.v**
   - A VHDL file <*design name*>**_pt.vhd** or an EDIF netlist file <*design name*>**_pt.edif**
   - An SDF file <*design name*>**_pt.sdf**.

   If the project contains RAM, ROM, dual-port RAM, or `clklock` functions, the **genpt** utility generates a <*design name*>_<*type*>**.db** file, where <*type*> is `ram`, `rom`, `dpram`, or `cklk`, which contains compiled STAMP library cell models for the PrimeTime software. The **genpt** utility also generates a <*design name*>**_setup.pt** PrimeTime setup file, which contains PrimeTime setup commands for compiling generated STAMP models and for reading in the EDIF, Verilog, or VHDL file and the SDF file.

4. Start the PrimeTime software by typing `primetime` ↵ at the UNIX prompt. You can also type `pt_shell` ↵ at the UNIX prompt to run the PrimeTime software in command-line mode.

5. Source the <*design name*>**_setup.pt** PrimeTime setup file. Refer to Synopsys PrimeTime documentation for information on how to perform timing verification with the PrimeTime software.

Using Synopsys PrimeTime & MAX+PLUS II Software

The following topics describe how to use the Synopsys PrimeTime and MAX+PLUS® II software. Choose one of the following topics for information:

Open a printable version of all topics listed on this page.

Setting Up the MAX+PLUS II/Synopsys Working Environment

- Software Requirements
- MAX+PLUS II/Synopsys Interface File Organization
- MAX+PLUS II Project File Structure

Timing Verification

- Timing Verification Flow
- Preparing Files for Timing Verification with PrimeTime Software using the **genpt** Utility

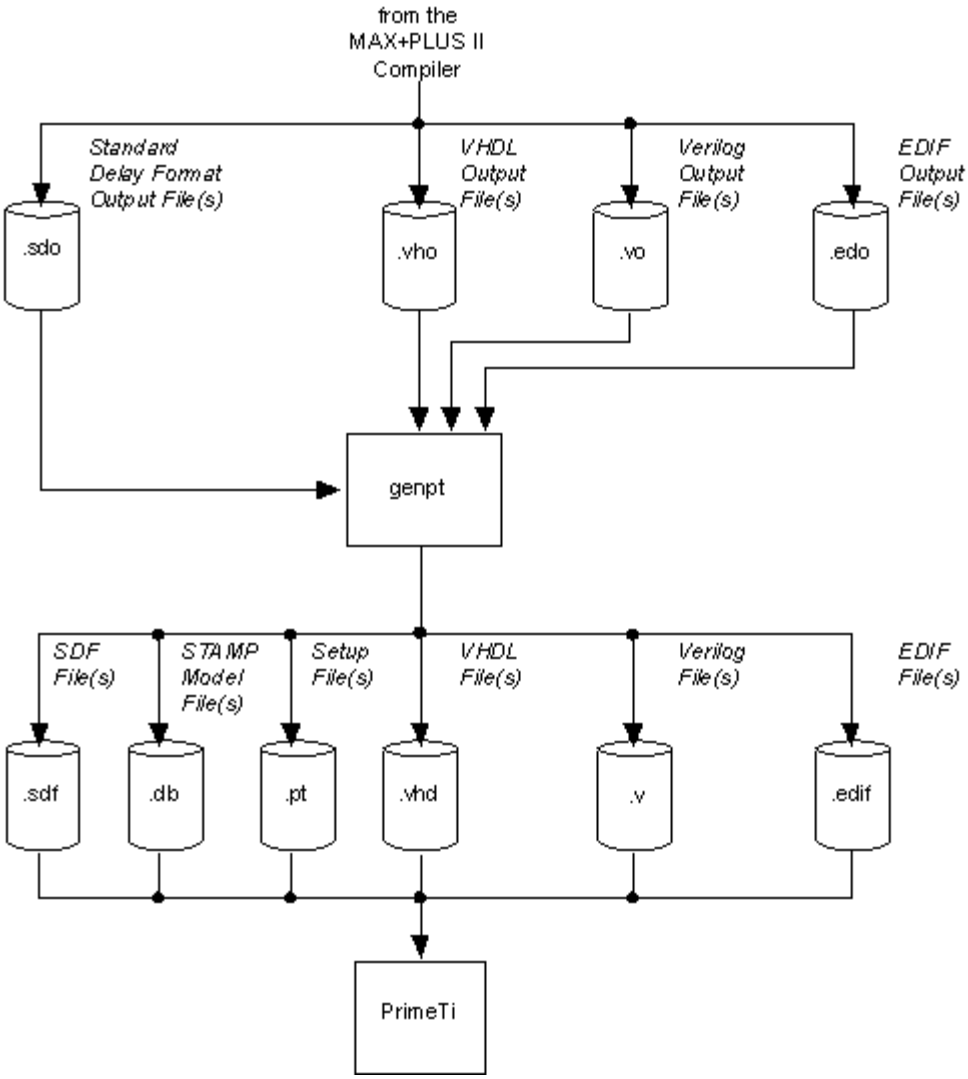
Related Links

- Compiling Projects with MAX+PLUS II Software
- Programming Altera Devices
- Using Synopsys Design Compiler or FPGA Compiler & MAX+PLUS II Software
- Using Synopsys FPGA Express and MAX+PLUS II Software
- Using Synopsys VSS & MAX+PLUS II Software
- MAX+PLUS II Development Software
- Altera Programming Hardware
- Synopsys web site (http://www.synopsys.com)

MAX+PLUS II/Synopsys PrimeTime Timing Verification Flow

Figure 1 shows the project timing verification flow for the MAX+PLUS® II/Synopsys PrimeTime interface.

Figure 1. MAX+PLUS II/Synopsys PrimeTime Project Timing Verification Flow



Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Using Mentor Graphics QuickHDL and QuickHDL Pro & MAX+PLUS II Software

The following topics describe how to use the Mentor Graphics QuickHDL and QuickHDL Pro software with MAX+PLUS® II software. Choose one of the following topics for information:

Open a printable version of all topics listed on this page.

Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment

- Software Requirements
- Altera-Provided Logic & Symbol Libraries
- Local Work Area Directory Structure
- Mentor Graphics Project Directory Structure
- MAX+PLUS II Project Directory Structure
- MAX+PLUS II/Mentor Graphics/Exemplar Logic Interface File Organization

Functional Simulation

- Design Entry Flow
- Performing a Functional Simulation with QuickHDL Software
- Performing a Functional Simulation with QuickHDL Pro Software

Timing Simulation

- Project Simulation/Timing Analysis Flow
- Initializing Registers in VHDL & Verilog Output Files for Power-Up before Simulation
- Performing a Timing Simulation with QuickHDL Software

Related Links:

- Go to the following MAX+PLUS II ACCESS℠ Key topics for related information:
  - Creating VHDL & Verilog HDL Designs for Use with MAX+PLUS II Software
  - Compiling Projects with MAX+PLUS II Software
  - Programming Altera Devices
- Go to the following topics, which are available on the web, for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware
  - Mentor Graphics web site (http://www.mentor.com)

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Using Mentor Graphics QuickHDL and QuickHDL Pro & MAX+PLUS II Software

---

The following topics describe how to use the Mentor Graphics QuickHDL and QuickHDL Pro software with MAX+PLUS® II software. Click on one of the following topics for information:

This file is suitable for printing only. It does not contain hypertext links that allow you to jump from topic to topic.

Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment

- Software Requirements
- Altera-Provided Logic & Symbol Libraries
- Local Work Area Directory Structure
- Mentor Graphics Project Directory Structure
- MAX+PLUS II Project Directory Structure
- MAX+PLUS II/Mentor Graphics/Exemplar Logic Interface File Organization

Functional Simulation

- Design Entry Flow
- Performing a Functional Simulation with QuickHDL Software
- Performing a Functional Simulation with QuickHDL Pro Software

Timing Simulation

- Project Simulation/Timing Analysis Flow
- Initializing Registers in VHDL & Verilog Output Files for Power-Up before Simulation
- Performing a Timing Simulation with QuickHDL Software

Related Topics:

- Go to the following MAX+PLUS II ACCESS℠ Key topics for related information:
  - Creating VHDL & Verilog HDL Designs for Use with MAX+PLUS II Software
  - Compiling Projects with MAX+PLUS II Software
  - Programming Altera Devices
- Go to the following topics, which are available on the web, for additional information:
- MAX+PLUS II Development Software
- Altera Programming Hardware
- Mentor Graphics web site (http://www.mentor.com)

Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment

To use the MAX+PLUS ® II software with Mentor Graphics/Exemplar Logic software, you must install the MAX+PLUS II software, then establish an environment that facilitates entering and processing designs. The MAX+PLUS II/Mentor Graphics/Exemplar Logic interface is installed automatically when you install the

MAX+PLUS II software on your computer.

Go to *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual for more information on installation and details on the directories that are created during MAX+PLUS II installation. Go to MAX+PLUS II/Mentor Graphics/Exemplar Logic Interface File Organization for information about the MAX+PLUS II/Mentor Graphics directories that are created during MAX+PLUS II installation.

The information presented here assumes that you are using a C shell and that your MAX+PLUS II system directory is **/usr/maxplus2**. If not, you must use the appropriate syntax and procedures to set environment variables for your shell.

To set up your working environment for the MAX+PLUS II/Mentor Graphics interface, follow these steps:

1. Ensure that you have correctly installed the MAX+PLUS II and Mentor Graphics software versions described in MAX+PLUS II/Mentor Graphics Software Requirements.
2. Add the following environment variables to your **.cshrc** file:

   ```
   setenv ALT_HOME /usr/maxplus2 ↵
   setenv MGC_WD <user-specified working directory> ↵
   setenv MGC_HOME <Mentor Graphics system directory> ↵
   setenv MAX2_MENTOR /usr/maxplus2/mentor/max2 ↵
   setenv MGC_LOCATION_MAP <user-specified location_map file> ↵
   setenv EXEMPLAR <Galileo or Leonardo system directory> ↵
   ```

   Installing the Altera® provided MAX+PLUS II/Mentor Graphics interface on your computer automatically installs a template for these environment variables in the **/usr/maxplus2/mentor/max2/.cshrc** file.
3. Add the **$MGC_HOME/bin**, **$MAX2_MENTOR/bin**, **$ALT_HOME/bin**, **$EXEMPLAR/bin/**<*os*>, and **$ALT_HOME/bin** directories to the PATH environment variable in your **.cshrc** file, where <*os*> is the operating system, e.g., SUN4 for SunOS; SUN5 for Solaris.
4. If you plan to use the Altera Schematic Express (**sch_exprss**) utility or the Altera VHDL Express (**vhd_exprss**) utility, add the following environment variable to your **.cshrc** file:

   ```
   setenv MAX2_QSIM /usr/maxplus2/simlib/mentor/max2sim ↵
   ```
5. Type source ~/.cshrc at a UNIX prompt to source the **.cshrc** file and validate the settings in steps 1 through 4.
6. Add the following lines to your **MGC_location_map** file:

   ```
   $MAX2_MENTOR ↵
   /usr/maxplus2/mentor/max2 ↵
   $MGC_GENLIB ↵
   /<user-specified Mentor Graphics GEN_LIB directory> ↵
   $MGC_LSLIB ↵
   /<user-specified Mentor Graphics LS_LIB directory> ↵
   $MAX2_EXAMPLES ↵
   /<user-specified example directory> ↵
   $MAX2_LMCLIB ↵
   /<user-specified Logic Modeling directory> ↵
   $MAX2_GENLIB ↵
   /usr/maxplus2/simlib/mentor/alt_max2 ↵
   $MAX2_QSIM ↵
   /usr/maxplus2/simlib/mentor/max2sim ↵
   $MAX2_FONT ↵
   /usr/maxplus2/mentor/max2/fonts ↵
   $MGC_SYS1076_STD ↵
   ```

/<*user-specified MGC_HOME directory*>/pkgs/sys_1076_std/ std ↵

$MGC_SYS1076_ARITHMETIC ↵

/<*user-specified MGC_HOME directory*>/pkgs/sys_1076_std/arithmetic ↵

$MGC_SYS1076_PORTABLE ↵

/<*user-specified MGC_HOME directory*>/pkgs/sys_1076_std/mgc_portable ↵

$MGC_SYS1076_IEEE ↵

/<*user-specified MGC_HOME directory*>/pkgs/sys_1076_std/ieee ↵

$MGC_SYS1076_SRC ↵

/<*user-specified MGC_HOME directory*>/pkgs/sys_1076_std/ src ↵

$MAX2_MFLIB ↵

/usr/maxplus2/simlib/mentor/alt_mf ↵

☞ Installing the Alteraprovided MAX+PLUS II/Mentor Graphics/Exemplar Logic interface on your computer automatically installs a template for these environment variables in the **/usr/maxplus2/mentor/max2/location_map/location_map** file.

7. If you want to use QuickHDL software to simulate VHDL or Verilog HDL designs, add the following line in the [library] section of your **quickhdl.ini** file: altera = $MAX2_MFLIB .
8. If you plan to use QuickHDL software to simulate VITAL-compliant VHDL files, add the following lines to your **MGC_location_map** file:

   $MAX2_VTLLIB ↵

   /usr/maxplus2/simlib/mentor/alt_vtl ↵
9. Copy the **/usr/maxplus2/maxplus2.ini** file to your **$HOME** directory:

   cp /usr/maxplus2/maxplus2.ini $HOME ↵

   chmod u+w $HOME/maxplus2.ini ↵

   The **maxplus2.ini** file contains both Altera- and user-specified initialization parameters that control the MAX+PLUS II software, such as Alteraprovided logic and symbol library paths and the current project name. The MAX+PLUS II installation procedure creates and copies the **maxplus2.ini** file to the **/usr/maxplus2** directory.

   ☞ Normally, you do not have to edit your local copy of **maxplus2.ini**, because the MAX+PLUS II software updates the file automatically whenever you change any parameters or settings. However, if you move the **max2lib** and **max2inc** library subdirectories, you must update the file. Go to "Creating & Using a Local Copy of the **maxplus2.ini** File" in MAX+PLUS II Help for more information.

Related Topics:

- Go to the following topics, which are available on the web, for additional information:
  - ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
  - This manual is also available in 4 parts:
    - Preface & Section 1: MAX+PLUS II Installation
    - Section 2: MAX+PLUS II - A Perspective
    - Section 3: MAX+PLUS II Tutorial
    - Appendices, Glossary & Index

MAX+PLUS II/Mentor Graphics Software Requirements

The following products are used to generate, process, synthesize, and verify a project with the MAX+PLUS ® II software and Mentor Graphics software:

Mentor Graphics	Exemplar	Altera

version C.1:

System_1076 Compiler QuickSim II Design Architect ENRead ENWrite GEN_LIB library	QuickHDL QuickHDL Pro QuickPath LS_LIB library (optional) DVE	Galileo Extreme version 4.1.1 Leonardo version 4.1.3	MAX+PLUS II version 9.4

☞ The MAX+PLUS II **read.me** file provides up-to-date information on which versions of Mentor Graphics applications are supported by the current version of MAX+PLUS II. It also provides information on installation and operating requirements. You should read the **read.me** file on the CD-ROM before installing the MAX+PLUS II software. After installation, you can open the **read.me** file from the MAX+PLUS II Help menu.

Altera-Provided Logic & Symbol Libraries

The MAX+PLUS ® II/Mentor Graphics environment provides libraries for compiling, synthesizing, and simulating designs.

☞ You can create your own libraries of custom functions for use in Design Architect schematics and VHDL and Verilog HDL design files. You can use custom functions to incorporate an EDIF Input File (**.edf**), Text Design File (**.tdf**), or any other MAX+PLUS II-supported design file into a project. The MAX+PLUS II software uses the Altera® provided **mnt8_bas.lmf** and **exemplar.lmf** Library Mapping Files to map standard Design Architect symbols and VHDL and Verilog HDL functions to equivalent MAX+PLUS II logic functions. To use custom functions, you can create a custom LMF that maps your custom functions to the equivalent EDIF input file, TDF, or other design file. Go to "Library Mapping File" in MAX+PLUS II Help for more information.

Design Architect Libraries

You can enter a Design Architect schematic with logic functions from these Altera-provided symbol libraries: **ALTERA LPMLIB**, **ALTERA GENLIB**, **LSTTL BY TYPE**, and **LSTTL ALL PARTS**. You can access these libraries by choosing **Altera Libraries** (Libraries menu) in the Design Architect software. For information on using library of parameterized modules (LPM) functions, see **ALTERA LPMLIB** Library below.

ALTERA GENLIB Library (Design Architect) & Altera (VHDL) Libraries

The **ALTERA GENLIB** symbol library (called the **Altera** library for VHDL) includes several MAX+PLUS II primitives for controlling design synthesis and fitting. It also includes four macrofunctions (`8count`, `8mcomp`, `8fadd`, and `81mux`) that are optimized for different Altera device families, and the `clklock` phase-locked loop megafunction, which is supported for some FLEX ® 10K devices.

The following table shows the MAX+PLUS II-specific logic functions.

Table 1. MAX+PLUS II-Specific Logic Functions

Macrofunctions *Note (1)*				**Primitives**	
Name	**Description**	**Name**	**Description**	**Name**	**Description**
`8fadd`	8-bit full adder	`LCELL`	Logic cell buffer	`EXP`	MAX ® 5000, MAX 7000 , and MAX 9000 Expander buffer
`8mcomp`	8-bit magnitude comparator	`GLOBAL`	Global input buffer	`SOFT`	Soft buffer
`8count`	8-bit up/down counter	`CASCADE`	FLEX 6000, FLEX 8000, and FLEX 10K cascade buffer	`OPNDRN`	Open-drain buffer

`81mux`	8-to-1 multiplexer	`CARRY`	FLEX 6000, FLEX 8000, and FLEX 10K carry buffer	`DFFE` `DFFE6K`	D-type flipflop with Clock Enable *Note (2)*
`clklock`	Phase-locked loop				

Notes:

1. Logic function names that begin with a number must be preceded by "`a_`" in VHDL designs. For example, `8fadd` must be specified as `a_8fadd` instead.
2. If you want to use QuickHDL software, make sure you have updated your **quickhdl.ini** file, as described in step 7 of Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment.
3. For designs that are targeted for FLEX 6000 devices, you should use the `DFFE` primitive only if the design contains either a Clear or Preset signal, but not both. If your design contains both a Clear and a Preset signal, you must use the `DFFE6K` primitive.

☞ Choose **Old-Style Macrofunctions**, **Primitives**, or **Megafunctions/LPM** from the MAX+PLUS II Help menu for detailed information on these functions.

ALTERA LPMLIB Library

The Alteraprovided **ALTERA LPMLIB** library, which is available for Design Architect schematics and VHDL designs, includes standard functions from the library of parameterized modules (LPM) 2.1.0, except the truth table, finite state machine, and pad functions. The LPM standard defines a set of parameterized modules (i.e., parameterized functions) and their corresponding representations in an EDIF netlist file. These logic functions allow you to create and functionally simulate an LPM-based design without targeting a specific device family. After the design is completed, you can target the design to any device family. The parameters you specify for each LPM function determine which simulation models are generated.

☞ Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu for more information about LPM functions.

Related Topics:

- Go to the following topics, which are available on the web, for additional information:
  - FLEX Devices
  - MAX Devices
  - Classic Device Family

Local Work Area Directory Structure

Design Architect software automatically creates and maintains the project directory structure required for all stages of design entry. Galileo Extreme, Leonardo, and ENWrite software create a **max2** subdirectory, if it does not already exist, under the project directory. These software applications also generate EDIF netlist files, and copy them from the <*project name*> directory to this **max2** subdirectory. All MAX+PLUS ® II Compiler output files are created in the **max2** subdirectory.

Simulation files created with Mentor Graphics applications and Logic Modeling files are located in the board-level simulation subdirectory of the project directory. You can use these files during simulation with QuickSim II software.

The only directory you need to create is the local work directory, which should contain all project directories. Figure 1 shows the recommended file structure.

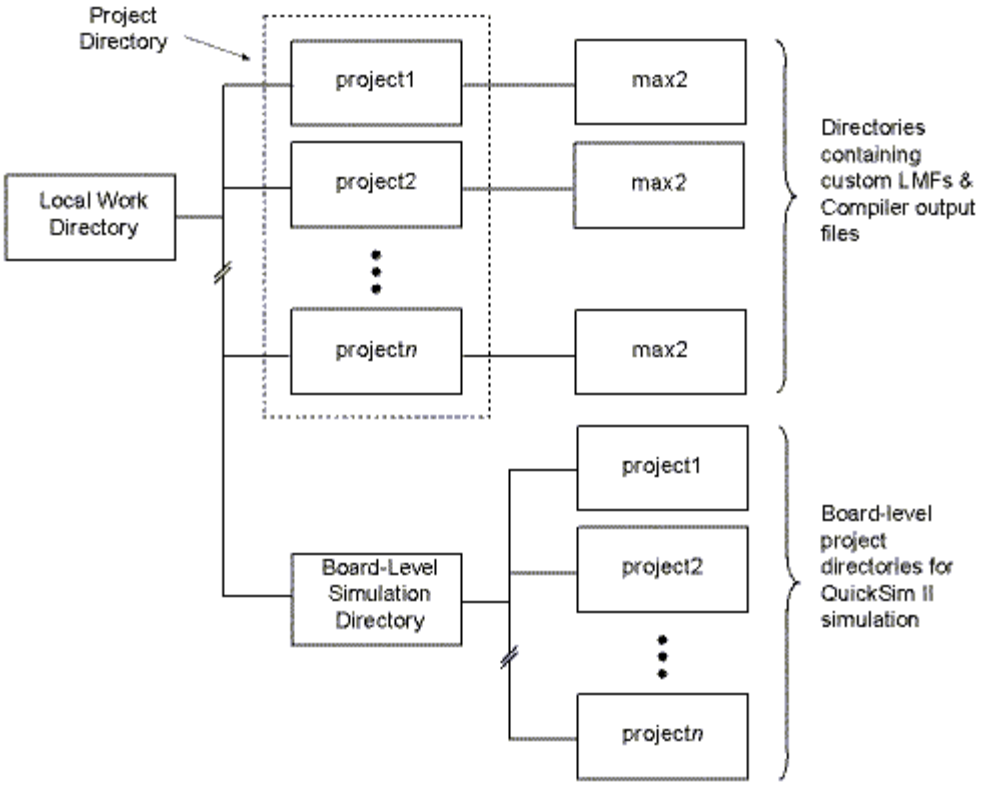


Figure 1. Recommended File Structure

Related Topics:

- Go to the following MAX+PLUS II ACCESS[SM] Key topics for related information:
  - MAX+PLUS II Project Directory Structure
  - Mentor Graphics Project Directory Structure

Mentor Graphics Project Directory Structure

Design Architect software generates the following files for each schematic:

- <*drawing name*>**/mgc_component.attr**
- <*drawing name*>**/part.Eddm_part.attr**
- <*drawing name*>**/part.part_1**
- <*drawing name*>**/schematic.mgc_schematic.attr**
- <*drawing name*>**/schematic/schem_id**
- <*drawing name*>**/schematic/sheet1.mgc_sheet.attr**
- <*drawing name*>**/schematic/sheet1.sgfx_1**
- <*drawing name*>**/schematic/sheet1.ssht_1**

The files generated for each schematic are stored in the schematic's <*drawing name*> directory and should not be edited. Mentor Graphics software automatically manages file storage and retrieval operations through this <*drawing name*> directory structure, which does not reflect hierarchical design relationships. Figure 1 shows a sample file structure with **project1** as the UNIX project directory, and **design1**, **subdesign1**, and **subdesign2** as the directories for the top-level design and subdesigns of the project.

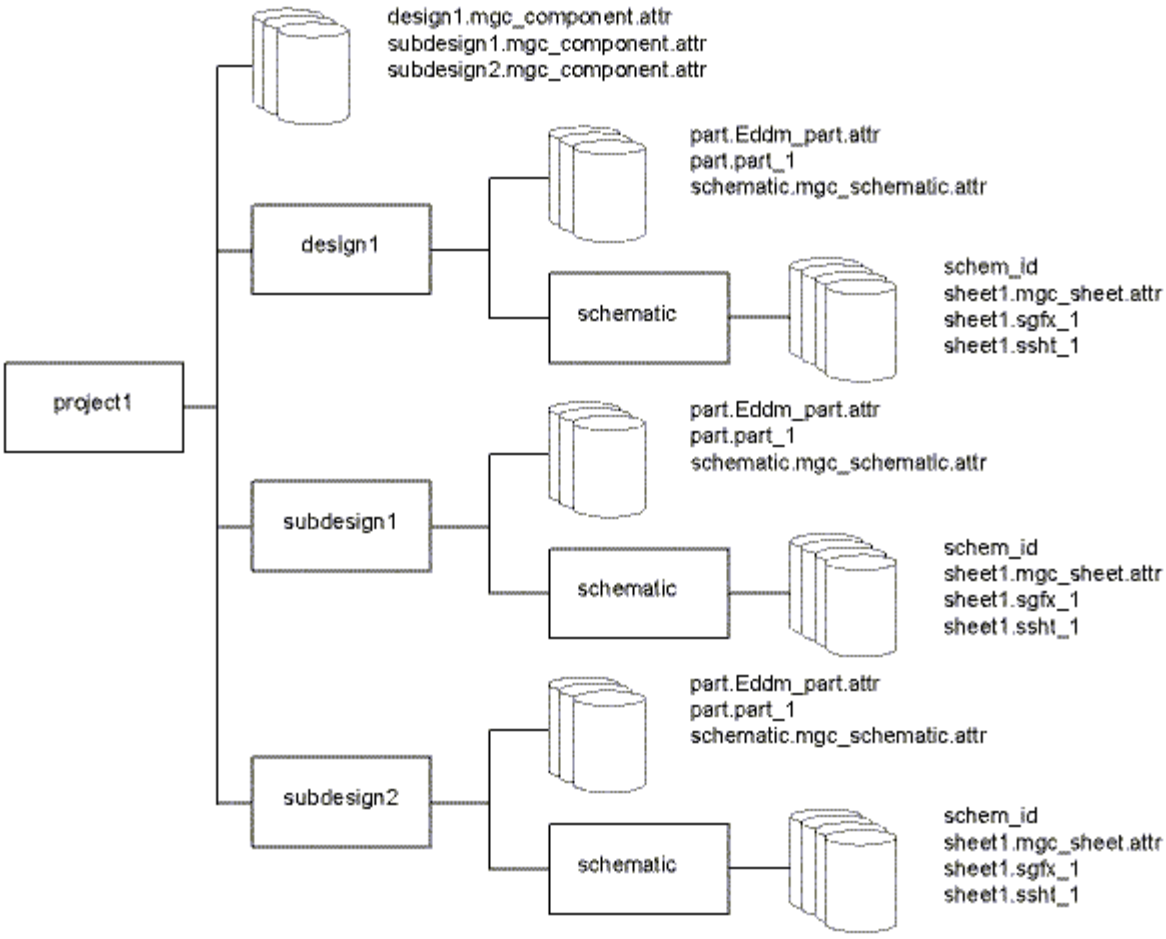


Figure 1. Design Architect Project File Structure

When the ENWrite utility converts the schematic into an EDIF netlist file, it processes the design information and all related file subdirectories, then creates the EDIF netlist file in the directory defined by the user. The EDIF netlist file is named <*project name*>**.edf**, where <*project name*> is the name of the top-level design file. The <*project name*>**.edf** file is automatically moved to the **max2** directory under the project directory.

Related Topics:

- Go to the following MAX+PLUS II ACCESS^{SM} Key topics for related information:
  - Local Work Area Directory Structure
  - MAX+PLUS II Project Directory Structure

MAX+PLUS II Project Directory Structure

In the MAX+PLUS $^{®}$ II software, a project name is the name of a top-level design file, without the filename extension. This design file can be an EDIF, VHDL, or Verilog HDL netlist file; an Altera Hardware Description Language (AHDL) Text Design File (TDF); or any other MAX+PLUS II-supported design file. The EDIF netlist file must be created by ENWrite, Galileo Extreme, or Leonardo software and imported into MAX+PLUS II as an EDIF Input File (**.edf**). Figure 1 shows an example of a MAX+PLUS II project directory.

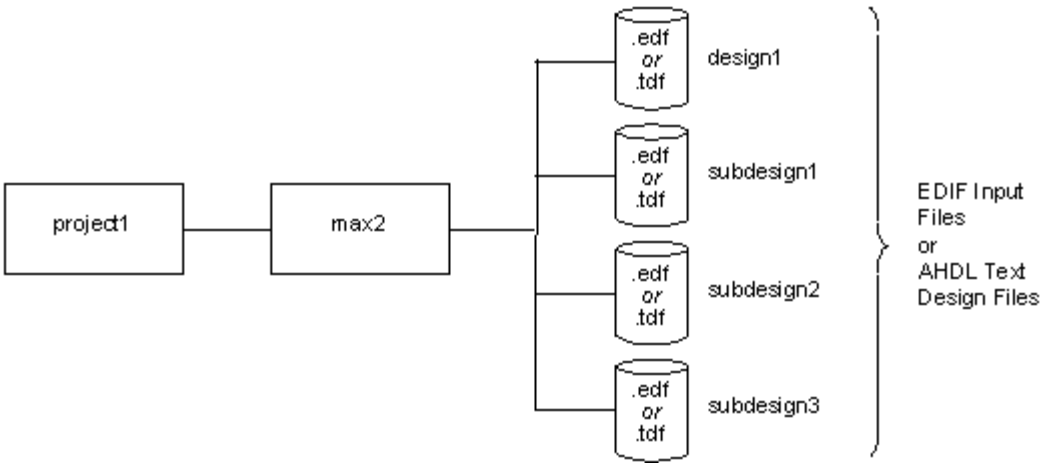


Figure 1. Sample MAX+PLUS II Project Directory

The MAX+PLUS II software stores the connectivity data on the links between design files in a hierarchical project in a Hierarchy Interconnect File (**.hif**), but refers to the entire project only by its project name. The MAX+PLUS II Compiler uses the HIF to build a single, fully flattened project database that integrates all the design files in a project hierarchy.

Related Topics:

- Go to the following MAX+PLUS II ACCESS^{SM} Key topics for related information:
  - Local Work Area Directory Structure
  - Mentor Graphics Project Directory Structure

MAX+PLUS II/Mentor Graphics/Exemplar Logic Interface File Organization

The following table shows the MAX+PLUS $^{®}$ II/Mentor Graphics interface subdirectories that are created in the MAX+PLUS II system directory (by default, the **/usr/maxplus2** directory) during MAX+PLUS II installation.

Related Topics:

- For information on the other directories that are created during MAX+PLUS II installation, see "MAX+PLUS II File Organization" in *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual.

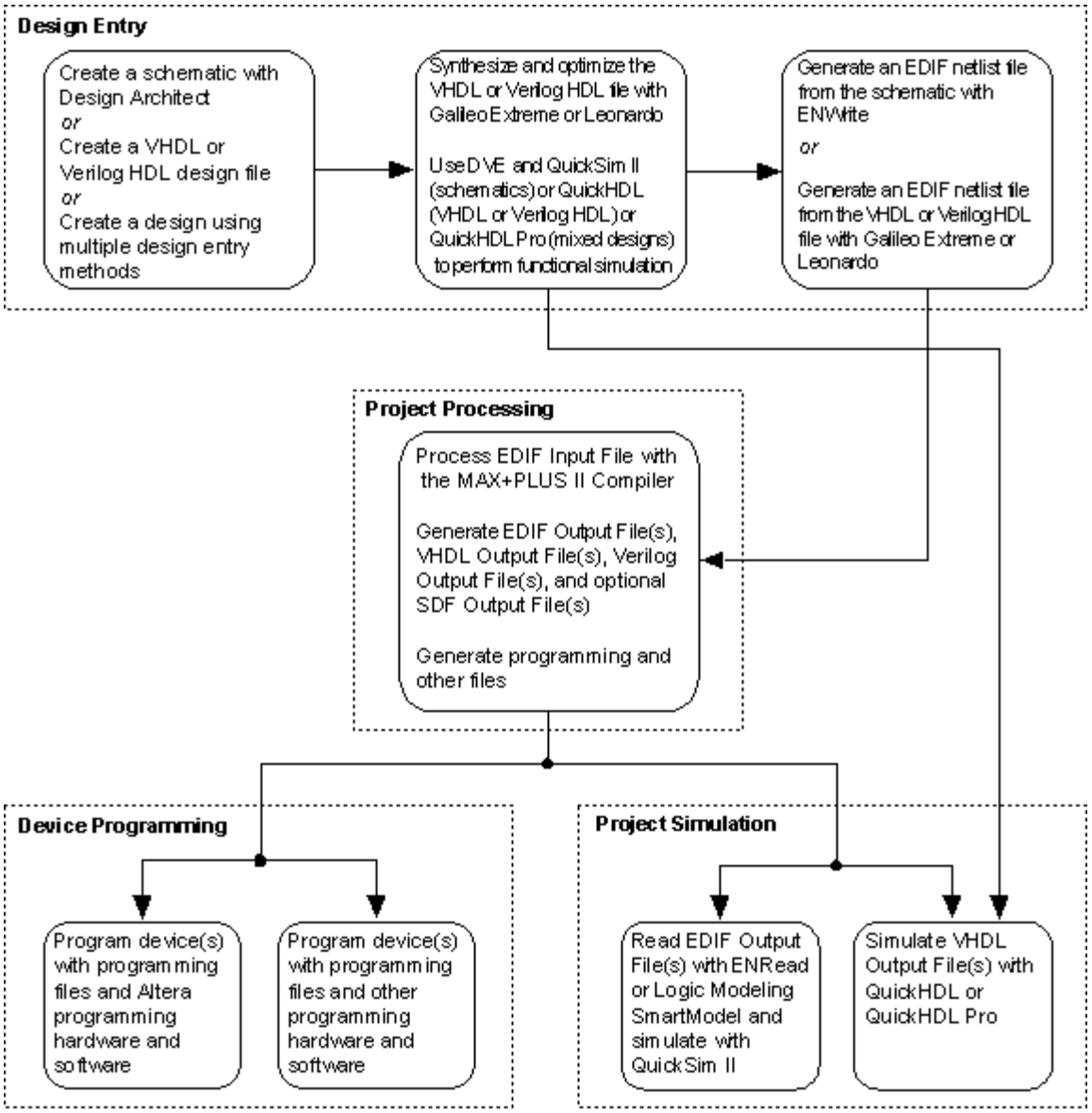
Table 1. MAX+PLUS II Directory Organization

Directory	Description
.lmf	Contains the Altera-provided Library Mapping Files, **mnt8_bas.lmf** and **exemplar.lmf**, that map Mentor Graphics and Exemplar Logic logic functions to equivalent MAX+PLUS II logic functions.
./mentor	Contains the AMPLE userware for the MAX+PLUS II/Mentor Graphics interface.
./simlib/mentor/alt_max2	Contains MAX+PLUS II primitives such as `CARRY`, `CASCADE`, `EXP`, `GLOBAL`, `LCELL`, `SOFT`, `OPNDRN`, `DFFE`, and `DFFE6K` (D flipflop with Clock Enable) for use in Design Architect schematics.
./simlib/mentor/max2sim	Contains the MAX+PLUS II/Mentor Graphics simulation model library, **max2sim**, for use with QuickSim II and QuickPath software.
./simlib/mentor/synlib	Contains the MAX+PLUS II synthesis library for use with AutoLogic II software, which supports synthesis for users running Mentor Graphics version B1.

./simlib/mentor/alt_mf Contains the MAX+PLUS II macrofunction and megafunction libraries.
./simlib/mentor/alt_vtl Contains the MAX+PLUS II VITAL library.

Altera/Mentor Graphics/Exemplar Logic Design Flow

The following figure shows the typical design flow for logic circuits created and processed with the MAX+PLUS ® II and Mentor Graphics/Exemplar Logic software. Detailed diagrams for each stage of the design flow appear in Design Entry Flow, Project Compilation Flow, Project Simulation/Timing Analysis Flow, and Device Programming Flow.



Performing a Functional Simulation with QuickHDL Software

You can use Mentor Graphics QuickHDL software to functionally simulate VHDL or Verilog HDL design files before compiling them with the MAX+PLUS ® II Compiler.

☞ If you wish to functionally simulate a hierarchical design that uses multiple design entry methods, you should use QuickHDL Pro rather than QuickHDL. Refer to Performing a Functional Simulation with QuickHDL Pro

Software for more information.

To functionally simulate a VHDL or Verilog HDL design, follow these steps:

1. Be sure to set up the working environment correctly, as described in Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment.
2. Create a VHDL or Verilog HDL design file that follows the guidelines described in Creating VHDL & Verilog HDL Designs for Use with MAX+PLUS II Software.
3. Start Design Architect by double-clicking Button 1 on the **max_da** icon in the Design Manager tools window. You can also start Design Architect software by typing `max2_da` ↵ at the UNIX prompt.
4. Choose **Lib** (QuickHDL menu) and specify your work library name as the *Work Library* name. Choose **OK**.
5. Choose **Map** (QuickHDL menu) to map the instantiated function to the equivalent function in the **Altera** logic function library. Choose **Set** to specify *altera* as the *Logical Name* and *$MAX2_MFLIB* as the *Physical Name*. Choose **OK**.
6. Choose **Compile** (QuickHDL menu) and use the Navigator window to select the icon for your project. Specify your work library name as the *Work Library* name and select the *Simulation* setting in the Set VHDL Compilation Options or Set Verilog HDL Compilation Options window. Choose **OK** to compile.
7. Choose **Simulate** (QuickHDL menu) and specify your work library name as the *Work Library* name. Choose **OK** to start the QuickHDL Startup window.
8. Select the icon for your project in the Entity Configuration window and choose **OK** to simulate the design.
9. Synthesize and optimize the design, as described in Synthesizing & Optimizing VHDL & Verilog HDL Projects with Galileo Extreme Software or Synthesizing & Optimizing VHDL & Verilog HDL Projects with Leonardo Software.

If your Verilog HDL design uses memory functions (RAM or ROM) that can be initialized with a hexadecimal file (Intel-format) initialization, you must convert these files into Verilog HDL format using the Programming Language Interface (PLI). To use the Altera-provided source code for PLI, perform the following steps:

1. Download the file **http://www.edif.org/lpmweb/convert_hex2ver.c** to your project directory.
2. Copy the following two files from the **$MGC_HOME/shared/pkgs/quickhdl/include** directory into the **/usr/maxplus2** directory:

- **$MGC_HOME/shared/pkgs/quickhdl/include/veriuser**
- **$MGC_HOME/shared/pkgs/quickhdl/include/acc_user**

Refer to the ***Mentor Graphics QuickHDL User's Reference Manual***, version 8.5-4.6i, for information on compiling the PLI application on different platforms and using the Verilog HDL PLI.

Related Topics:

- Go to the following MAX+PLUS II ACCESS^{SM} Key topics for related information:
  - Compiling Projects with MAX+PLUS II Software
  - Performing a Timing Simulation with QuickHDL Software
  - Performing a Functional Simulation with QuickHDL Pro Software

---

Performing a Functional Simulation with QuickHDL Pro Software

You can use Mentor Graphics QuickHDL Pro software to functionally simulate mixed-level schematic and VHDL designs before compiling them with the MAX+PLUS ® II Compiler.

Refer to Mentor Graphics *Getting Started with QuickHDL Pro* page 2-1 and 3-1 for compatible design configurations.

To functionally simulate a QuickHDL at Top Level design, follow the steps in *Getting Started with QuickHDL Pro*, Chapter 2.

To functionally simulate a QuickSim II at Top Level design, go through the following steps:

1. Be sure to set up the working environment correctly, as described in Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment.
2. Create a schematic design using QuickHDL models. Refer to Creating Design Architect Schematics for Use with MAX+PLUS II Software.
3. Compile the QuickHDL model using the QuickHDL Compiler with the **-qhpro_syminfo** option. (This is done automatically for LPM functions if you choose to compile the LPM models when saving the schematic.)
4. Start Design Architect by double-clicking Button 1 on the **max_da** icon in the Design Manager tools window.
5. Choose **Open** from the File menu, then choose **Sheet** from the Open menu to open the top level schematic.
6. Select the symbol for the VHDL model and choose **Begin Edit Symbol** from the Edit menu.
7. Press Button 3 to display the the Design Architect pop-up menu. Choose Add Menu from the Other Menus menu, then choose **Set VHDL Info**. Choose Import from Entity to display the "Import Entity Info" dialog box.
8. Specify the following options in the "Import Entity Info" dialog box:
    1. *QHDL InitFile*: Specify your **quickhdl.ini** file.
    2. *Library Logical Name*: Click on **Choose Library** button and fill the "Choose VHDL Library" form with your work library.
    3. *Entity Name*: Click on **Choose Entity** button and select the name of your entity.
    4. *Default Architecture*: Click on **Choose Arch** button and select corresponding architecture for the entity.

    After filling in the above information, click on **OK** to close the form.

9. Check the symbol with defaults. If there are no errors, save the symbol with default registration by choosing **Save Symbol** from the File menu, then choose **Default Registration**.
10. Choose **End Edit Symbol** from the Edit menu to close the Symbol Editor session. In the schematic window, select the symbol you have just edited and choose **Object** from the Report menu, then choose **All** from the Selected menu. In the report transcript, make sure the *MODEL* property is set to *qhpro* to ensure that the model will work with QuickHDL Pro.
11. Select the folder for your project, press button 3, and choose **Open max2_qvpro** to start QuickHDL Pro. You can also start QuickHDL Pro by typing max2_qvpro ↵ at the UNIX prompt. In the **QVHDL Pro System** dialog box, make sure *EDDM Design* is selected for *Invoke on* and the correct path name is specified for the design. Choose **OK** to start the QuickHDL Pro. A QHPro (QuickSim II) window and a QHPro (QuickHDL) window appear on the screen.
12. Use the QuickSim II window to simulate the top level schematic and the QuickHDL window to simulate the VHDL portion of the design.

Related Topics:

- Go to the following MAX+PLUS II ACCESS^{SM} Key topics for related information:<
    - Compiling Projects with MAX+PLUS II Software
    - Instantiating LPM Functions in Design Architect Schematics
    - Performing a Functional Simulation with QuickHDL Software

---

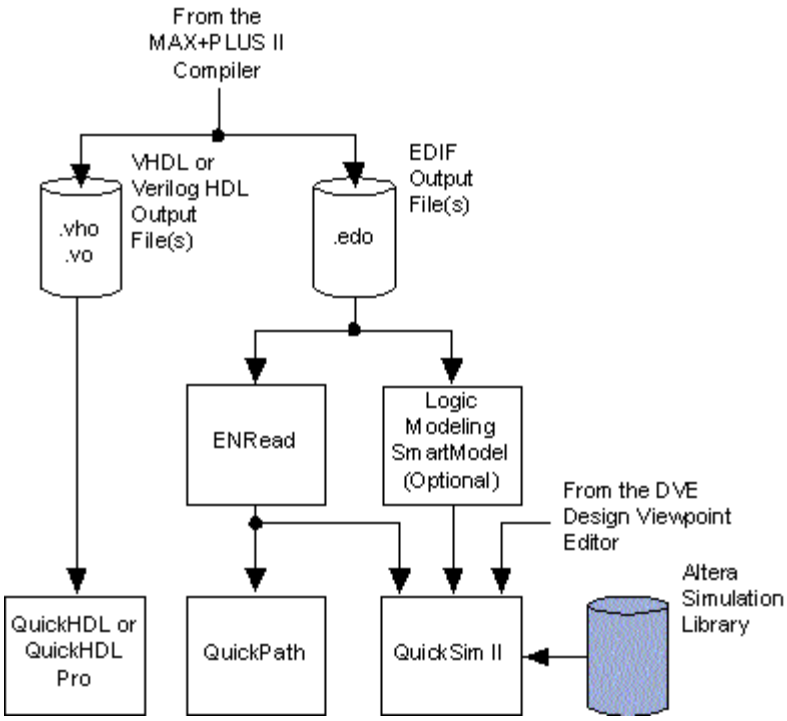
Project Simulation/Timing Analysis Flow

The following figure shows the project simulation and timing analysis flow for the MAX+PLUS® II /Mentor Graphics interface.

Figure 1. MAX+PLUS II/Mentor Graphics Project Simulation/Timing

Analysis Flow

Alteraprovided items are shown in blue.



Performing a Timing Simulation with QuickHDL Software

After you have entered a VHDL or Verilog HDL design file and compiled it with the MAX+PLUS ® II Compiler, you can use Mentor Graphics QuickHDL software to simulate the MAX+PLUS IIgenerated VHDL Output File (**.vhd**) or Verilog Output File (**.vo**) and the Standard Delay Format (SDF) Output File (**.sdo**).

To simulate your VHDL or Verilog HDL design, go through the following steps:

1. Be sure to set up the working environment correctly, as described in Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment.
2. Generate a VHDL or Verilog HDL output file and an SDF output file for your project, as described in Compiling Projects with MAX+PLUS II Software.
3. Change to your project's directory.
4. Copy your **quickhdl.ini** file to the same directory as your VHDL or Verilog HDL file.
5. Type the following sets of commands at the UNIX prompt to create the work library and compile your project's VHDL or Verilog HDL output file:

VHDL:	**Verilog HDL:**
`setenv MGC_WD 'pwd'` ↵	`setenv MGC_WD 'pwd'` ↵
`qhlib work` ↵	`qhlib work` ↵
`qvhcom` <*project name*>`.vho` ↵	`qvlcom` <*project name*>`.vo` ↵

6. Type `qhsim -sdftyp` <*project name*>`.sdo` ↵ at the UNIX prompt to perform timing back-annotation and

simulation and to display the QuickHDL simulation window.

If your Verilog HDL design uses memory functions (RAM or ROM) that can be initialized with a hexadecimal file (Intel-format) initialization, you must convert these files into Verilog HDL format using the Programming Language Interface (PLI). To use the Altera-provided source code for PLI, perform the following steps:

1. Download the file **http://www.edif.org/lpmweb/convert_hex2ver.c** to your project directory.
2. Copy the following two files from the **$MGC_HOME/shared/pkgs/quickhdl/include** directory into the **/usr/maxplus2** directory:
    - **$MGC_HOME/shared/pkgs/quickhdl/include/veriuser**
    - **$MGC_HOME/shared/pkgs/quickhdl/include/acc_user**

Refer to the ***Mentor Graphics QuickHDL User's Reference Manual***, version 8.5-4.6i, for information on compiling the PLI application on different platforms and using the Verilog HDL PLI.

Related Topics:

- Go to Performing a Functional Simulation with QuickHDL Software in these MAX+PLUS II ACCESS℠ Key topics for related information.

Creating VHDL & Verilog HDL Designs for Use with MAX+PLUS II Software

You can create VHDL and Verilog HDL design files with the MAX+PLUS ® II Text Editor or another standard text editor and save them in the appropriate directory for your project.

The MAX+PLUS II Text Editor offers the following advantages:

- Templates are available with the **VHDL Templates** and **Verilog Templates** commands (Template menu). These templates are also available in the ASCII **vhdl.tmp** and **verilog.tmp** files, respectively, which are located in the **/usr/maxplus2** directory.
- If you use the MAX+PLUS II Text Editor to create your VHDL design, you can turn on the **Syntax Coloring** command (Options menu). The Syntax Coloring feature displays keywords and other elements of text in text files in different colors to distinguish them from other forms of syntax.

To create a VHDL or Verilog HDL design file for use with the MAX+PLUS II software, go through the following steps:

1. Enter a VHDL or Verilog HDL design in the MAX+PLUS II Text Editor or another standard text editor and save it in your working directory.
2. Enter primitives, macrofunctions, and megafunctions in your VHDL or Verilog HDL design from the **Altera** library.

    The following topics describe special steps needed to instantiate LPM and `clklock` functions:

    - Instantiating LPM Functions in VHDL
    - Instantiating the clklock Megafunction in VHDL or Verilog HDL

    ☞ You can instantiate MegaCore™ functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP™). The OpenCore™ feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.

3. (Optional) Use the QuickHDL software to functionally simulate the design file, as described in Performing a Functional Simulation with QuickHDL Software and Performing a Functional Simulation with QuickHDL Pro Software.

4. Once you have created a VHDL or Verilog HDL design, you can generate an EDIF netlist file that can be imported into the MAX+PLUS II software with either of the following methods:

    - You can synthesize and optimize your design and create an EDIF netlist file, as described in Synthesizing & Optimizing VHDL & Verilog HDL Projects with Galileo Extreme Software or Synthesizing & Optimizing VHDL & Verilog HDL Projects with Leonardo Software.

    - You can use the Altera VHDL Express utility, **vhd_exprss**, to automatically create an EDIF netlist file, compile it with the MAX+PLUS II Compiler, generate an EDIF Output File (**.edo**), and prepare the EDIF Output File for simulation with QuickHDL software, as described in Using the Altera Schematic Express (**vhd_exprss**) Utility.

Installing the Alteraprovided MAX+PLUS II/Mentor Graphics/Exemplar Logic interface on your computer automatically creates the following sample VHDL design files:

- **/usr/maxplus2/examples/mentor/example5/count4.vhd**
- **/usr/maxplus2/examples/mentor/example6/count8.vhd**
- **/usr/maxplus2/examples/mentor/example8/adder16.vhd**

Related Topics:

- Go to Compiling Projects with MAX+PLUS II Software in these MAX+PLUS II ACCESS^{SM} Key topics for related information.

Compiling Projects with MAX+PLUS II Software

The MAX+PLUS$^{®}$ II Compiler can process design files in a variety of formats. This topic describes how to use MAX+PLUS II software to compile projects in which the top-level design file is an EDIF Input File (with the extension **.edf**).

Related Topics:

- Refer to the following sources for additional information:
    - Go to MAX+PLUS II Help for information on compiling VHDL and Verilog HDL, design files directly with the MAX+PLUS II Compiler.
    - Go to Running Synopsys Compilers from MAX+PLUS II Software for information on running the Synopsys Design Compiler or FPGA Compiler software on a VHDL or Verilog HDL design from within the MAX+PLUS II Compiler window.

To compile a design (also called a "project") with MAX+PLUS II software, go through the following steps:

1. Create design files that are compatible with the MAX+PLUS II software and convert them into EDIF Input Files with the extension **.edf**. Specific instructions for some tools are described in these MAX+PLUS II ACCESS^{SM} Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your design entry or synthesis and optimization tool.
2. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, create a mapping for each function in a Library Mapping File (**.lmf**) to map the custom symbol to the corresponding EDIF Input File, AHDL Text Design File (**.tdf**), or other MAX+PLUS II-supported design file. These custom functions are represented in design files as hollow-body symbols or "black box" HDL

descriptions.

☞ Go to "Library Mapping Files (**.lmf**)" in MAX+PLUS II Help for more information.

3. Open MAX+PLUS II and specify the name of your top-level design file as the project name with the **Project Name** command (File menu). If you open an HDL file in the MAX+PLUS II Text Editor, you can choose the **Project Set Project to Current File** command (File menu) instead.

   ☞ You can also compile a project from a command line. However, the first time you compile a project, the settings you need to specify are easier to specify from within the MAX+PLUS II software. After you have run the graphical user interface for the MAX+PLUS II software at least once, you can more easily use the command-line **setacf** utility to modify options in the Assignment & Configuration File (**.acf**) for the project. Type `setacf -h` ↵ and `maxplus2 -h` ↵ for descriptions of **setacf** and MAX+PLUS II command-line syntax.

4. Choose **Device** (Assign menu) and select the target Altera device family in the *Device Family* drop-down list box. If you wish to implement the design logic in a specific device, select it in the *Devices* box. Otherwise, select *AUTO* to allow the MAX+PLUS II Compiler to choose the best device(s) in the current device family. If your design entry or synthesis and optimization tool required you to specify a target family and/or device, specify the same information in this dialog box. For information on partitioning logic among multiple devices, go to MAX+PLUS II Help. Choose **OK**.
5. Open the Compiler window by choosing the **Compiler** command (MAX+PLUS II menu). Go through the following steps to specify the options necessary to compile the design file(s) in your project:
   1. Ensure that all EDIF netlist files have the extension **.edf** and choose **EDIF Netlist Reader Settings** (Interfaces menu).
   2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor. This name should be the name of the vendor whose tool(s) you used to create the EDIF netlist files. If your vendor name does not appear, select *Custom* instead.

      ☞ If you are compiling a design created with Synopsys FPGA Express software, select *Synopsys*, choose the **Customize** button, enter <*project name*>**.lmf** in the *LMF #1* box, choose **OK**, and skip to step 6.

   3. If you selected an existing vendor name in the *Vendor* box and your project contains design files that require custom LMF mappings, choose the **Customize** button to expand the dialog box to show all settings. Turn on the *LMF #2* checkbox and type your custom LMF's filename in the corresponding text box, or select a name from the *Files* box. The selection in the *Vendor* box will change to *Custom* and all settings will be retained until you change them again.
   4. If you selected *Custom* in the *Vendor* box, choose the **Customize** button to expand the dialog box to show all settings. Any previously defined custom settings will be displayed. Under *Signal Names*, type one or more names with up to 20 total name characters in the *VCC* or *GND* box if your EDIF Input File(s) use one or more names other than `VCC` or `GND` for the global high or low signals. Multiple signal names must be separated by either a comma (,) or a space. Under *Library Mapping Files*, turn on the *LMF #1* checkbox and type a filename in the text box following it, or select a name from the *Files* box. If necessary, specify another LMF name in the *LMF #2* box. Go to MAX+PLUS II Help for detailed information on the settings available in the **EDIF Netlist Reader Settings** dialog box.
   5. Choose **OK**.
6. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, you may need to ensure that all files are present in your project directory, i.e., the same directory as the top-level design file. Otherwise, you must specify the directories containing these files as user libraries with the **User Libraries** command (Options menu).
7. Follow all guidelines that apply to your design entry or synthesis and optimization tool:
   - Exemplar Logic Galileo Extreme-Specific Compiler Settings
   - Synopsys DesignWare-Specific Compiler Settings
   - Converting Synopsys FPGA Compiler & Design Compiler Timing Constraints into MAX+PLUS II-Compatible Format with the **syn2acf** Utility
   - Synplicity Synplify-Specific Compiler Settings
8. If you wish to generate EDIF, VHDL, or Verilog HDL output files for post-compilation simulation or timing analysis with another EDA tool, go through the following steps:
   1. (Optional) Turn on the **Optimize Timing SNF** command (Processing menu) to reduce the size of the output file(s). Turning on this command can reduce the size of output netlists by up to 30%.

This command does not create optimized timing SNFs on UNIX workstations. However, a non-optimized timing SNF provides the same functional and timing information as an optimized timing SNF.

2. If you wish to generate EDIF Output Files (**.edo**), go through these steps:
    1. Turn on the **EDIF Netlist Writer** command (Interfaces menu). Then choose the **EDIF Netlist Writer Settings** command (Interfaces menu).
    2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor and choose **OK**. If your vendor name does not appear, select *Custom* instead and specify the settings that are appropriate for your simulation or timing analysis tool. Go to MAX+PLUS II Help for detailed information on the options available in the **EDIF Netlist Writer Settings** dialog box.
    3. To generate an optional Standard Delay Format (SDF) Output File (**.sdo**), choose the **Customize** button to expand the dialog box to show all settings. Select one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**.

    The filenames of the EDIF Output File(s) and optional SDF Output File(s) are the same as the user-defined chip name(s) for the project; if no chip names exist, the Compiler assigns filenames that are based on the project name. For a multi-device project, the Compiler also generates a top-level EDIF Output File that is uniquely identified by "**_t**" appended to the project name. In addition, the Compiler automatically generates a VHDL Memory Model Output File, *<project name>***.vmo**, when it generates an EDIF Output File that contains memory (RAM or ROM).

3. If you wish to generate VHDL Output Files (**.vho**), turn on the **VHDL Netlist Writer** command (Interfaces menu). Then choose **VHDL Netlist Writer Settings** command (Interfaces menu). Select *VHDL Output File (.vho)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF ver. 2.1 files contain timing delay information that allows you to perform back-annotation simulation in VHDL with VITAL-compliant simulation libraries. The VHDL Output Files generated by the Compiler have the extension **.vho**, but are otherwise named in the same way as the EDIF Output Files described above.
4. If you wish to generate Verilog HDL Output Files (**.vo**), turn on the **Verilog Netlist Writer** command (Interfaces menu). Then choose **Verilog Netlist Writer Settings** command (Interfaces menu). Select *Verilog Output File (.vo)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF Output Files contain timing delay information that allows you to perform back-annotation simulation in Verilog HDL. The Verilog Output Files generated by the Compiler have the extension **.vo**, but are otherwise named in the same way as the EDIF Output Files described above.

9. To run the MAX+PLUS II Compiler, choose the **Project Save & Compile** command (File menu) or choose the **Start** button in the Compiler window.

    See step 3 for information on running MAX+PLUS II software from the command line.

10. Once you have compiled the project with the MAX+PLUS II Compiler, you can use the VHDL, Verilog HDL, or EDIF output file(s), and the optional SDF Output File(s) (**.sdo**) to perform timing analysis or timing simulation with another EDA tool. Specific instructions for some tools are described in these MAX+PLUS II ACCESS Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your EDA tool.

The MAX+PLUS II Compiler also generates a Report File (**.rpt**), a Pin-Out File (**.pin**), and one or more of the following files for device programming or configuration:

- JEDEC Files (**.jed**)
- Programmer Object Files (**.pof**)
- SRAM Object Files (**.sof**)
- Hexadecimal (Intel-format) Files (**.hex**)
- Tabular Text Files (**.ttf**)

Related Topics:

- Refer to the following sources for additional information:

- Go to *Compiler Procedures* in MAX+PLUS II Help for information on other available Compiler settings.
- Go to *Programmer Procedures* in MAX+PLUS II Help for instructions on creating other types of programming files and on programming or configuring Altera devices.
- Go to Back-Annotating MAX+PLUS II Pin Assignments to Design Architect Symbols for information on back-annotating pin assignments in Mentor Graphics Design Architect schematics.
- Go to Programming Altera Devices for information on the different programming hardware options for Altera device families.

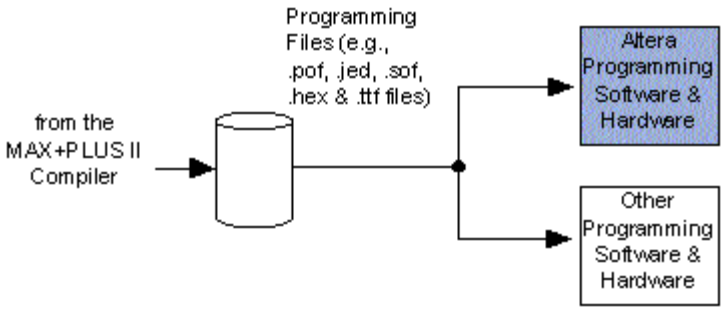
- Go to the following topics, which are available on the web, for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware

Programming Altera Devices

Once you have successfully compiled and simulated a project with the MAX+PLUS® II software, you can program an Altera® device and test it in the target circuit. Figure 1 shows the device programming flow for MAX+PLUS II software.

Figure 1. MAX+PLUS II Device Programming Flow

Altera-provided items are shown in blue.



You can program devices with Altera programming hardware and MAX+PLUS II Programmer software installed on a 486- or Pentium-based PC or a UNIX workstation, or with programming hardware and software available from other manufacturers. Table 1 shows the available Altera programming hardware options on PCs and UNIX workstations.

Table 1. Altera Programming Hardware

Programming Hardware Option	UNIX PCs Work-stations	MAX® 3000A Devices	Classic® & MAX 5000 Devices	MAX 7000 & MAX 7000E Devices	MAX 7000A, MAX 7000AE, MAX 7000B, MAX 7000S MAX 9000 & MAX	FLEX® 6000, FLEX 6000A, FLEX 8000, FLEX 10K, FLEX 10KA, FLEX 10KB, & FLEX 10KE Devices	In-System Programming/ Configuration

						9000A Devices		
Logic Programmer card, PL-MPU Master Programming Unit, and device-specific adapters	✓		✓	✓	✓	✓		
BitBlaster™ Download Cable	✓	✓	✓			✓	✓	✓
ByteBlasterMV™ Download Cable	✓		✓			✓	✓	✓
MasterBlaster™ Download Cable	✓	✓	✓			✓	✓	✓

If you wish to transfer programming files from a UNIX workstation to a PC over a network with File Transfer Protocol (FTP) or other similar transfer programs, be sure to select binary transfer mode.

Programming hardware from other manufacturers varies, but typically consists of a device connected to one of the serial ports on the workstation. Various vendors, such as Data I/O and BP Microsystems, supply hardware and software for programming Altera devices.

Related Topics:

- Go to Compiling Projects with MAX+PLUS II Software for information on creating programming files.
- Go to the following topics, which are available on the web, for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware
  - FLEX Devices
  - MAX Devices
  - Classic Device Family

Performing a Functional Simulation with QuickHDL Pro Software

You can use Mentor Graphics QuickHDL Pro software to functionally simulate mixed-level schematic and VHDL designs before compiling them with the MAX+PLUS ® II Compiler.

Refer to Mentor Graphics *Getting Started with QuickHDL Pro* page 2-1 and 3-1 for compatible design configurations.

To functionally simulate a QuickHDL at Top Level design, follow the steps in *Getting Started with QuickHDL Pro*, Chapter 2.

To functionally simulate a QuickSim II at Top Level design, go through the following steps:

1. Be sure to set up the working environment correctly, as described in Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment.
2. Create a schematic design using QuickHDL models. Refer to Creating Design Architect Schematics for Use with MAX+PLUS II Software.
3. Compile the QuickHDL model using the QuickHDL Compiler with the **-qhpro_syminfo** option. (This is done automatically for LPM functions if you choose to compile the LPM models when saving the schematic.)
4. Start Design Architect by double-clicking Button 1 on the **max_da** icon in the Design Manager tools window.
5. Choose **Open** from the File menu, then choose **Sheet** from the Open menu to open the top level schematic.
6. Select the symbol for the VHDL model and choose **Begin Edit Symbol** from the Edit menu.
7. Press Button 3 to display the the Design Architect pop-up menu. Choose Add Menu from the Other Menus menu, then choose **Set VHDL Info**. Choose Import from Entity to display the "Import Entity Info" dialog box.
8. Specify the following options in the "Import Entity Info" dialog box:
    1. *QHDL InitFile*: Specify your **quickhdl.ini** file.
    2. *Library Logical Name*: Click on **Choose Library** button and fill the "Choose VHDL Library" form with your work library.
    3. *Entity Name*: Click on **Choose Entity** button and select the name of your entity.
    4. *Default Architecture*: Click on **Choose Arch** button and select corresponding architecture for the entity.

After filling in the above information, click on **OK** to close the form.

- Check the symbol with defaults. If there are no errors, save the symbol with default registration by choosing **Save Symbol** from the File menu, then choose **Default Registration**.
- Choose **End Edit Symbol** from the Edit menu to close the Symbol Editor session. In the schematic window, select the symbol you have just edited and choose **Object** from the Report menu, then choose **All** from the Selected menu. In the report transcript, make sure the *MODEL* property is set to *qhpro* to ensure that the model will work with QuickHDL Pro.
- Select the folder for your project, press button 3, and choose **Open max2_qvpro** to start QuickHDL Pro. You can also start QuickHDL Pro by typing max2_qvpro ↵ at the UNIX prompt. In the **QVHDL Pro System** dialog box, make sure *EDDM Design* is selected for *Invoke on* and the correct path name is specified for the design. Choose **OK** to start the QuickHDL Pro. A QHPro (QuickSim II) window and a QHPro (QuickHDL) window appear on the screen.
- Use the QuickSim II window to simulate the top level schematic and the QuickHDL window to simulate the VHDL portion of the design.

Related Links:

- Compiling Projects with MAX+PLUS II Software
- Instantiating LPM Functions in Design Architect Schematics
- Performing a Functional Simulation with QuickHDL Software

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Using Mentor Graphics QuickPath & MAX+PLUS II Software

The following topics describe how to use the Mentor Graphics QuickPath software with MAX+PLUS® II software. Click on one of the following topics for information:

This file is suitable for printing only. It does not contain hypertext links that allow you to jump from topic to topic.

Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment

- Software Requirements
- Altera-Provided Logic & Symbol Libraries
- Local Work Area Directory Structure
- Mentor Graphics Project Directory Structure
- MAX+PLUS II Project Directory Structure
- MAX+PLUS II/Mentor Graphics/Exemplar Logic Interface File Organization

Timing Analysis

- Project Simulation/Timing Analysis Flow
- Performing a Timing Analysis with QuickPath Software

Related Topics:

- Go to the following MAX+PLUS II ACCESS℠ Key topics for related information:
  - Compiling Projects with MAX+PLUS II Software
  - Performing a Timing Simulation with DVE & QuickSim II Software
  - Programming Altera Devices
- Go to the following topics for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware
  - Mentor Graphics web site (http://www.mentor.com)

Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment

To use the MAX+PLUS ® II software with Mentor Graphics/Exemplar Logic software, you must install the MAX+PLUS II software, then establish an environment that facilitates entering and processing designs. The MAX+PLUS II/Mentor Graphics/Exemplar Logic interface is installed automatically when you install the MAX+PLUS II software on your computer.

Go to *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual for more information on installation and details on the directories that are created during MAX+PLUS II installation. Go to MAX+PLUS II/Mentor Graphics/Exemplar Logic Interface File Organization for information about the MAX+PLUS II/Mentor

Graphics directories that are created during MAX+PLUS II installation.

The information presented here assumes that you are using a C shell and that your MAX+PLUS II system directory is **/usr/maxplus2**. If not, you must use the appropriate syntax and procedures to set environment variables for your shell.

To set up your working environment for the MAX+PLUS II/Mentor Graphics interface, follow these steps:

1. Ensure that you have correctly installed the MAX+PLUS II and Mentor Graphics software versions described in MAX+PLUS II/Mentor Graphics Software Requirements.

2. Add the following environment variables to your **.cshrc** file:

```
setenv ALT_HOME /usr/maxplus2 ↵
setenv MGC_WD <user-specified working directory> ↵
setenv MGC_HOME <Mentor Graphics system directory> ↵
setenv MAX2_MENTOR /usr/maxplus2/mentor/max2 ↵
setenv MGC_LOCATION_MAP <user-specified location_map file> ↵
setenv EXEMPLAR <Galileo or Leonardo system directory> ↵
```

   Installing the Altera® provided MAX+PLUS II/Mentor Graphics interface on your computer automatically installs a template for these environment variables in the **/usr/maxplus2/mentor/max2/.cshrc** file.

3. Add the **$MGC_HOME/bin**, **$MAX2_MENTOR/bin**, **$ALT_HOME/bin**, **$EXEMPLAR/bin/**<*os*>, and **$ALT_HOME/bin** directories to the `PATH` environment variable in your **.cshrc** file, where <*os*> is the operating system, e.g., `SUN4` for SunOS; `SUN5` for Solaris.

4. If you plan to use the Altera Schematic Express (**sch_exprss**) utility or the Altera VHDL Express (**vhd_exprss**) utility, add the following environment variable to your **.cshrc** file:

```
setenv MAX2_QSIM /usr/maxplus2/simlib/mentor/max2sim ↵
```

5. Type `source ?/.cshrc` at a UNIX prompt to source the **.cshrc** file and validate the settings in steps 1 through 4.

6. Add the following lines to your **MGC_location_map** file:

```
$MAX2_MENTOR ↵
/usr/maxplus2/mentor/max2 ↵
$MGC_GENLIB ↵
/<user-specified Mentor Graphics GEN_LIB directory> ↵
$MGC_LSLIB ↵
/<user-specified Mentor Graphics LS_LIB directory> ↵
$MAX2_EXAMPLES ↵
/<user-specified example directory> ↵
$MAX2_LMCLIB ↵
/<user-specified Logic Modeling directory> ↵
$MAX2_GENLIB ↵
/usr/maxplus2/simlib/mentor/alt_max2 ↵
$MAX2_QSIM ↵
                                    ↵
```

```
/usr/maxplus2/simlib/mentor/max2sim
$MAX2_FONT ↵
/usr/maxplus2/mentor/max2/fonts ↵
$MGC_SYS1076_STD ↵
/<user-specified MGC_HOME directory>/pkgs/sys_1076_std/ std ↵
$MGC_SYS1076_ARITHMETIC ↵
/<user-specified MGC_HOME directory>/pkgs/sys_1076_std/arithmetic ↵
$MGC_SYS1076_PORTABLE ↵
/<user-specified MGC_HOME directory>/pkgs/sys_1076_std/mgc_portable ↵
$MGC_SYS1076_IEEE ↵
/<user-specified MGC_HOME directory>/pkgs/sys_1076_std/ieee ↵
$MGC_SYS1076_SRC ↵
/<user-specified MGC_HOME directory>/pkgs/sys_1076_std/ src ↵
$MAX2_MFLIB ↵
/usr/maxplus2/simlib/mentor/alt_mf ↵
```

Installing the Alteraprovided MAX+PLUS II/Mentor Graphics/Exemplar Logic interface on your computer automatically installs a template for these environment variables in the **/usr/maxplus2/mentor/max2/location_map/location_map** file.

7. If you want to use QuickHDL software to simulate VHDL or Verilog HDL designs, add the following line in the `[library]` section of your **quickhdl.ini** file: `altera = $MAX2_MFLIB`.

8. If you plan to use QuickHDL software to simulate VITAL-compliant VHDL files, add the following lines to your **MGC_location_map** file:

```
$MAX2_VTLLIB ↵
/usr/maxplus2/simlib/mentor/alt_vtl ↵
```

9. Copy the **/usr/maxplus2/maxplus2.ini** file to your **$HOME** directory:

```
cp /usr/maxplus2/maxplus2.ini $HOME ↵
chmod u+w $HOME/maxplus2.ini ↵
```

The **maxplus2.ini** file contains both Altera- and user-specified initialization parameters that control the MAX+PLUS II software, such as Alteraprovided logic and symbol library paths and the current project name. The MAX+PLUS II installation procedure creates and copies the **maxplus2.ini** file to the **/usr/maxplus2** directory.

Normally, you do not have to edit your local copy of **maxplus2.ini**, because the MAX+PLUS II software updates the file automatically whenever you change any parameters or settings. However, if you move the **max2lib** and **max2inc** library subdirectories, you must update the file. Go to "Creating & Using a Local Copy of the **maxplus2.ini** File" in MAX+PLUS II Help for more information.

Related Topics:

- Go to the following topics, which are available on the web, for additional information:
  - ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
  - This manual is also available in 4 parts:
    - Preface & Section 1: MAX+PLUS II Installation
    - Section 2: MAX+PLUS II - A Perspective

Section 3: MAX+PLUS II Tutorial
- Appendices, Glossary & Index

MAX+PLUS II/Mentor Graphics Software Requirements

The following products are used to generate, process, synthesize, and verify a project with the MAX+PLUS ® II software and Mentor Graphics software:

Mentor Graphics		Exemplar	Altera
version C.1:			
System_1076 Compiler QuickSim II Design Architect ENRead ENWrite GEN_LIB library	QuickHDL QuickHDL Pro QuickPath LS_LIB library (optional) DVE	Galileo Extreme version 4.1.1 Leonardo version 4.1.3	MAX+PLUS II version 9.4

☞ The MAX+PLUS II **read.me** file provides up-to-date information on which versions of Mentor Graphics applications are supported by the current version of MAX+PLUS II. It also provides information on installation and operating requirements. You should read the **read.me** file on the CD-ROM before installing the MAX+PLUS II software. After installation, you can open the **read.me** file from the MAX+PLUS II Help menu.

Altera-Provided Logic & Symbol Libraries

The MAX+PLUS ® II/Mentor Graphics environment provides libraries for compiling, synthesizing, and simulating designs.

☞ You can create your own libraries of custom functions for use in Design Architect schematics and VHDL and Verilog HDL design files. You can use custom functions to incorporate an EDIF Input File (**.edf**), Text Design File (**.tdf**), or any other MAX+PLUS II-supported design file into a project. The MAX+PLUS II software uses the Altera® provided **mnt8_bas.lmf** and **exemplar.lmf** Library Mapping Files to map standard Design Architect symbols and VHDL and Verilog HDL functions to equivalent MAX+PLUS II logic functions. To use custom functions, you can create a custom LMF that maps your custom functions to the equivalent EDIF input file, TDF, or other design file. Go to "Library Mapping File" in MAX+PLUS II Help for more information.

Design Architect Libraries

You can enter a Design Architect schematic with logic functions from these Altera-provided symbol libraries: **ALTERA LPMLIB**, **ALTERA GENLIB**, **LSTTL BY TYPE**, and **LSTTL ALL PARTS**. You can access these libraries by choosing **Altera Libraries** (Libraries menu) in the Design Architect software. For information on using library of parameterized modules (LPM) functions, see **ALTERA LPMLIB** Library below.

ALTERA GENLIB Library (Design Architect) & Altera (VHDL) Libraries

The **ALTERA GENLIB** symbol library (called the **Altera** library for VHDL) includes several MAX+PLUS II primitives for controlling design synthesis and fitting. It also includes four macrofunctions (`8count`, `8mcomp`, `8fadd`, and `81mux`) that are optimized for different Altera device families, and the `clklock` phase-locked loop megafunction, which is supported for some FLEX ® 10K devices.

The following table shows the MAX+PLUS II-specific logic functions.

Table 1. MAX+PLUS II-Specific Logic Functions

Macrofunctions *Note (1)*				**Primitives**	
Name	**Description**	**Name**	**Description**	**Name**	**Description**
`8fadd`	8-bit full adder	`LCELL`	Logic cell buffer	`EXP`	MAX ® 5000, MAX 7000 , and MAX 9000 Expander buffer
`8mcomp`	8-bit magnitude comparator	`GLOBAL`	Global input buffer	`SOFT`	Soft buffer
`8count`	8-bit up/down counter	`CASCADE`	FLEX 6000, FLEX 8000, and FLEX 10K cascade buffer	`OPNDRN`	Open-drain buffer
`81mux`	8-to-1 multiplexer	`CARRY`	FLEX 6000, FLEX 8000, and FLEX 10K carry buffer	`DFFE` `DFFE6K` *Note (2)*	D-type flipflop with Clock Enable
`clklock`	Phase-locked loop				

Notes:

1. Logic function names that begin with a number must be preceded by "`a_`" in VHDL designs. For example, `8fadd` must be specified as `a_8fadd` instead.
2. If you want to use QuickHDL software, make sure you have updated your **quickhdl.ini** file, as described in step 7 of Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment.
3. For designs that are targeted for FLEX 6000 devices, you should use the `DFFE` primitive only if the design contains either a Clear or Preset signal, but not both. If your design contains both a Clear and a Preset signal, you must use the `DFFE6K` primitive.

☞ Choose **Old-Style Macrofunctions**, **Primitives**, or **Megafunctions/LPM** from the MAX+PLUS II Help menu for detailed information on these functions.

ALTERA LPMLIB Library

The Alteraprovided **ALTERA LPMLIB** library, which is available for Design Architect schematics and VHDL designs, includes standard functions from the library of parameterized modules (LPM) 2.1.0, except the truth table, finite state machine, and pad functions. The LPM standard defines a set of parameterized modules (i.e., parameterized functions) and their corresponding representations in an EDIF netlist file. These logic functions allow you to create and functionally simulate an LPM-based design without targeting a specific device family. After the design is completed, you can target the design to any device family. The parameters you specify for each LPM function determine which simulation models are generated.

☞ Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu for more information about LPM functions.

Related Topics:

- Go to the following topics, which are available on the web, for additional information:
  - FLEX Devices
  - MAX Devices
  - Classic Device Family

Local Work Area Directory Structure

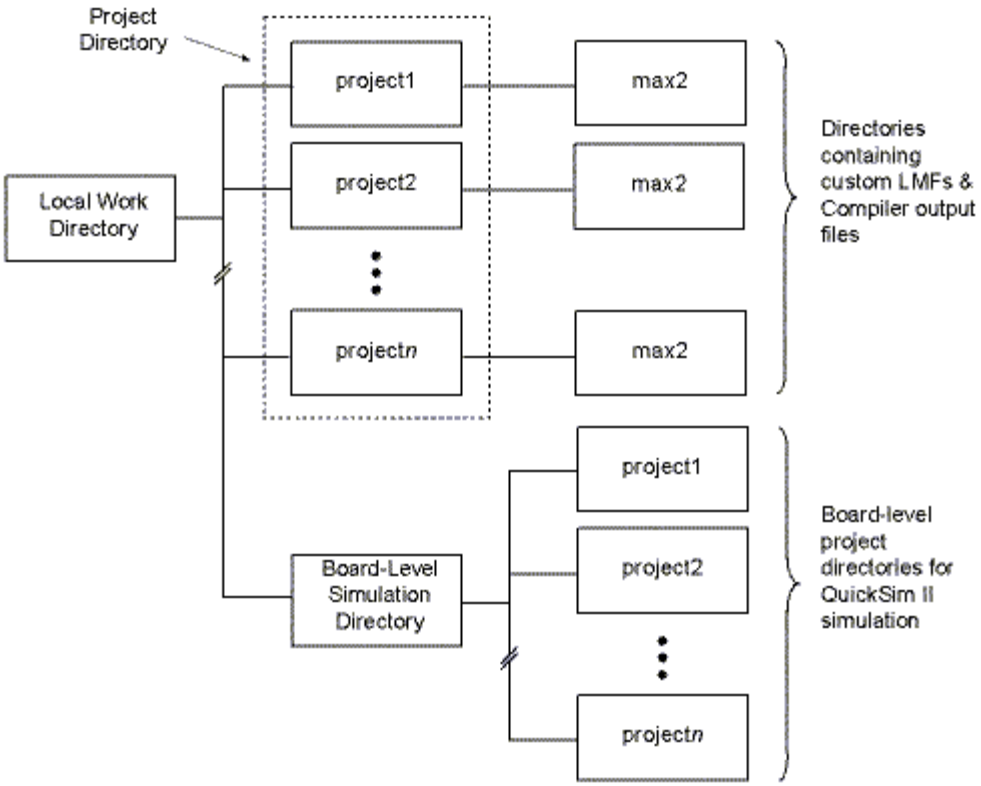
Design Architect software automatically creates and maintains the project directory structure required for all stages of design entry. Galileo Extreme, Leonardo, and ENWrite software create a **max2** subdirectory, if it does not

already exist, under the project directory. These software applications also generate EDIF netlist files, and copy them from the <*project name*> directory to this **max2** subdirectory. All MAX+PLUS ® II Compiler output files are created in the **max2** subdirectory.

Simulation files created with Mentor Graphics applications and Logic Modeling files are located in the board-level simulation subdirectory of the project directory. You can use these files during simulation with QuickSim II software.

The only directory you need to create is the local work directory, which should contain all project directories. Figure 1 shows the recommended file structure.

Figure 1. Recommended File Structure



Related Topics:

- Go to the following MAX+PLUS II ACCESS[SM] Key topics for related information:
  - MAX+PLUS II Project Directory Structure
  - Mentor Graphics Project Directory Structure

Mentor Graphics Project Directory Structure

Design Architect software generates the following files for each schematic:

- <*drawing name*>**/mgc_component.attr**
- <*drawing name*>**/part.Eddm_part.attr**
- <*drawing name*>**/part.part_1**
- <*drawing name*>**/schematic.mgc_schematic.attr**
- <*drawing name*>**/schematic/schem_id**
- <*drawing name*>**/schematic/sheet1.mgc_sheet.attr**
- <*drawing name*>**/schematic/sheet1.sgfx_1**

- <*drawing name*>**/schematic/sheet1.ssht_1**

The files generated for each schematic are stored in the schematic's <*drawing name*> directory and should not be edited. Mentor Graphics software automatically manages file storage and retrieval operations through this <*drawing name*> directory structure, which does not reflect hierarchical design relationships. Figure 1 shows a sample file structure with **project1** as the UNIX project directory, and **design1**, **subdesign1**, and **subdesign2** as the directories for the top-level design and subdesigns of the project.

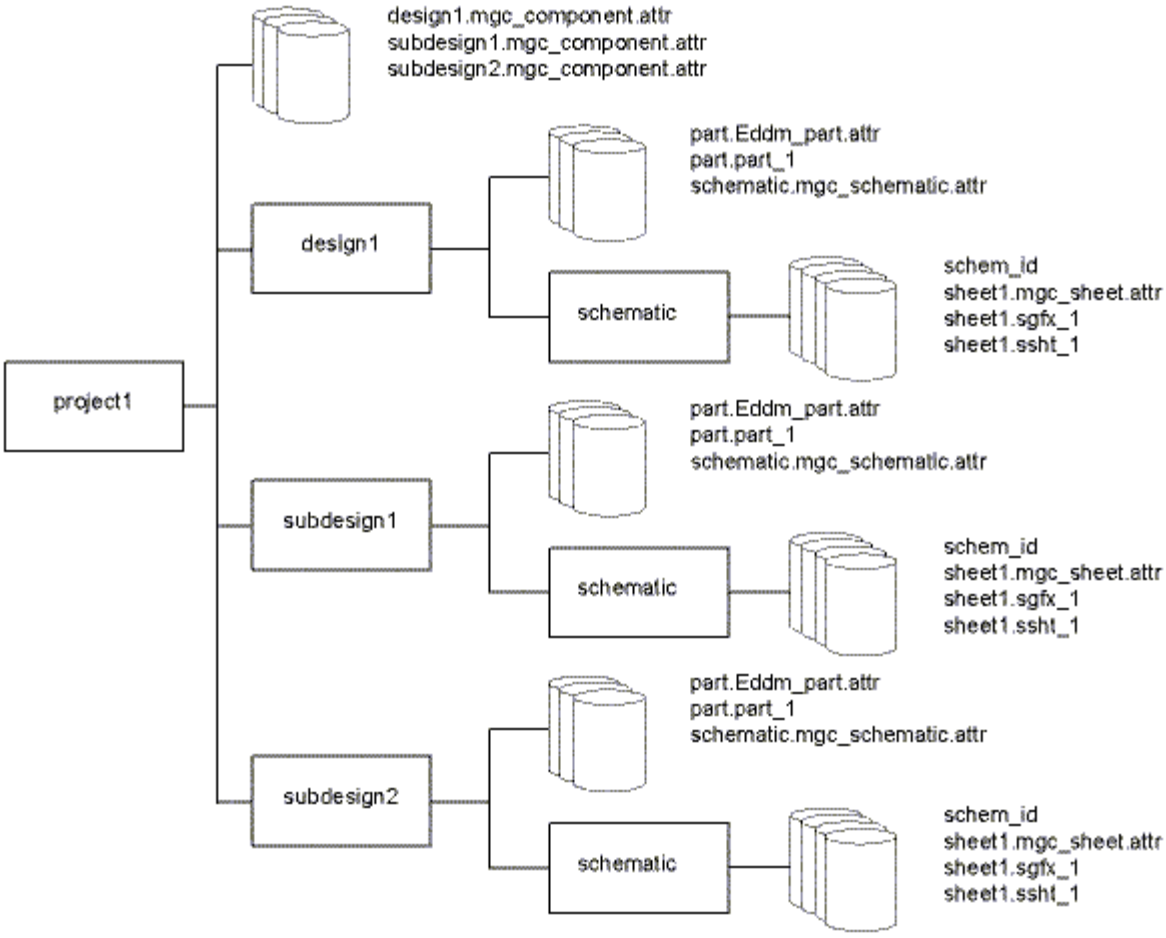


Figure 1. Design Architect Project File Structure

When the ENWrite utility converts the schematic into an EDIF netlist file, it processes the design information and all related file subdirectories, then creates the EDIF netlist file in the directory defined by the user. The EDIF netlist file is named <*project name*>**.edf**, where <*project name*> is the name of the top-level design file. The <*project name*>**.edf** file is automatically moved to the **max2** directory under the project directory.

Related Topics:

- Go to the following MAX+PLUS II ACCESS^{SM} Key topics for related information:
  - Local Work Area Directory Structure
  - MAX+PLUS II Project Directory Structure

MAX+PLUS II Project Directory Structure

In the MAX+PLUS $^{®}$ II software, a project name is the name of a top-level design file, without the filename extension. This design file can be an EDIF, VHDL, or Verilog HDL netlist file; an Altera Hardware Description Language (AHDL) Text Design File (TDF); or any other MAX+PLUS II-supported design file. The EDIF netlist file must be created by ENWrite, Galileo Extreme, or Leonardo software and imported into MAX+PLUS II as an

EDIF Input File (**.edf**). Figure 1 shows an example of a MAX+PLUS II project directory.

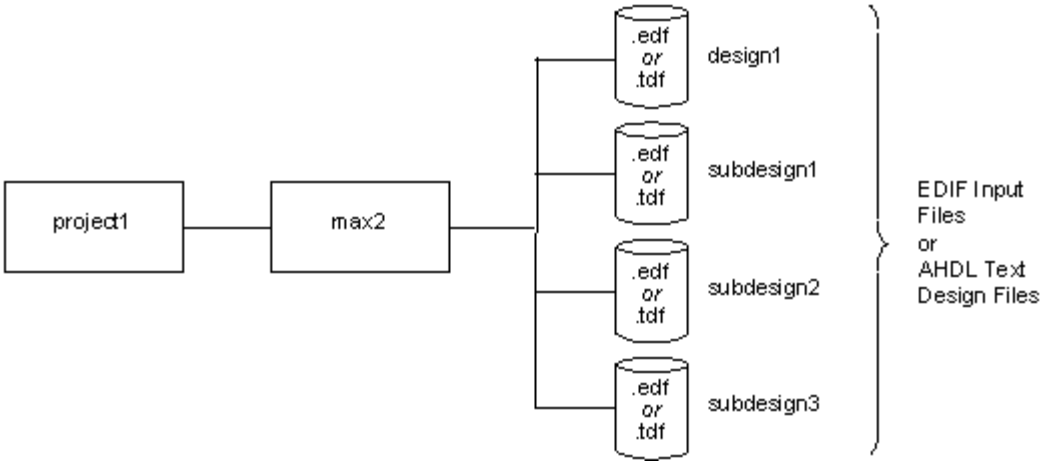


Figure 1. Sample MAX+PLUS II Project Directory

The MAX+PLUS II software stores the connectivity data on the links between design files in a hierarchical project in a Hierarchy Interconnect File (**.hif**), but refers to the entire project only by its project name. The MAX+PLUS II Compiler uses the HIF to build a single, fully flattened project database that integrates all the design files in a project hierarchy.

Related Topics:

- Go to the following MAX+PLUS II ACCESS^{SM} Key topics for related information:
  - Local Work Area Directory Structure
  - Mentor Graphics Project Directory Structure

MAX+PLUS II/Mentor Graphics/Exemplar Logic Interface File Organization

The following table shows the MAX+PLUS $^{®}$ II/Mentor Graphics interface subdirectories that are created in the MAX+PLUS II system directory (by default, the **/usr/maxplus2** directory) during MAX+PLUS II installation.

Related Topics:

- For information on the other directories that are created during MAX+PLUS II installation, see "MAX+PLUS II File Organization" in *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual.

Table 1. MAX+PLUS II Directory Organization

Directory	Description
.lmf	Contains the Altera-provided Library Mapping Files, **mnt8_bas.lmf** and **exemplar.lmf**, that map Mentor Graphics and Exemplar Logic logic functions to equivalent MAX+PLUS II logic functions.
./mentor	Contains the AMPLE userware for the MAX+PLUS II/Mentor Graphics interface.
./simlib/mentor/alt_max2	Contains MAX+PLUS II primitives such as CARRY, CASCADE, EXP, GLOBAL, LCELL, SOFT, OPNDRN, DFFE, and DFFE6K (D flipflop with Clock Enable) for use in Design Architect

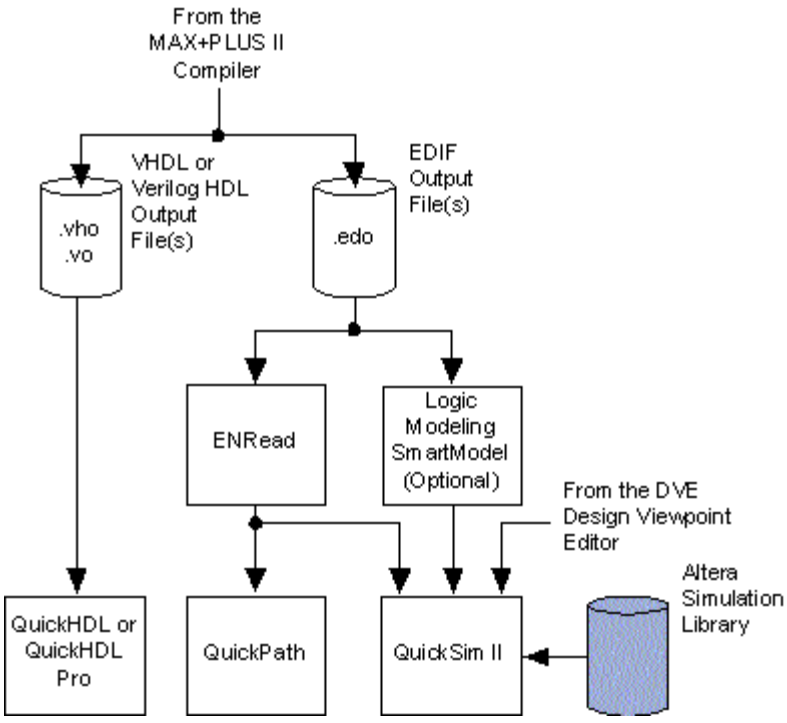
	schematics.
./simlib/mentor/max2sim	Contains the MAX+PLUS II/Mentor Graphics simulation model library, **max2sim**, for use with QuickSim II and QuickPath software.
./simlib/mentor/synlib	Contains the MAX+PLUS II synthesis library for use with AutoLogic II software, which supports synthesis for users running Mentor Graphics version B1.
./simlib/mentor/alt_mf	Contains the MAX+PLUS II macrofunction and megafunction libraries.
./simlib/mentor/alt_vtl	Contains the MAX+PLUS II VITAL library.

Project Simulation/Timing Analysis Flow

The following figure shows the project simulation and timing analysis flow for the MAX+PLUS® II /Mentor Graphics interface.

Figure 1. MAX+PLUS II/Mentor Graphics Project Simulation/Timing Analysis Flow

Alteraprovided items are shown in blue.



Performing a Timing Analysis with QuickPath Software

After you have compiled your project with the MAX+PLUS ® II Compiler and generated an EDIF Output File (**.edo**), you can use Mentor Graphics QuickPath software to perform a timing analysis of your project.

To perform a timing analysis with QuickPath software, follow these steps:

1. Be sure to set up the working environment correctly, as described in Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment.

2. Generate an EDIF Output File for your project using one of the following methods:
    - Compiling Projects with MAX+PLUS II Software
    - Using the Altera Schematic Express (**sch_exprss**) Utility
    - Using the Altera VHDL Express (**vhd_exprss**) Utility
3. Select your project's folder from the **ALTERA** directory, press Button 3, and choose **Open max2_qpath** to start the QuickPath software. You can also start the QuickPath software by typing `max2_qpath` ↵ at the UNIX prompt.

Compiling Projects with MAX+PLUS II Software

The MAX+PLUS® II Compiler can process design files in a variety of formats. This topic describes how to use MAX+PLUS II software to compile projects in which the top-level design file is an EDIF Input File (with the extension **.edf**).

Related Topics:

- Refer to the following sources for additional information:
    - Go to MAX+PLUS II Help for information on compiling VHDL and Verilog HDL, design files directly with the MAX+PLUS II Compiler.
    - Go to Running Synopsys Compilers from MAX+PLUS II Software for information on running the Synopsys Design Compiler or FPGA Compiler software on a VHDL or Verilog HDL design from within the MAX+PLUS II Compiler window.

To compile a design (also called a "project") with MAX+PLUS II software, go through the following steps:

1. Create design files that are compatible with the MAX+PLUS II software and convert them into EDIF Input Files with the extension **.edf**. Specific instructions for some tools are described in these MAX+PLUS II ACCESS℠ Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your design entry or synthesis and optimization tool.
2. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, create a mapping for each function in a Library Mapping File (**.lmf**) to map the custom symbol to the corresponding EDIF Input File, AHDL Text Design File (**.tdf**), or other MAX+PLUS II-supported design file. These custom functions are represented in design files as hollow-body symbols or "black box" HDL descriptions.

    ☞ Go to "Library Mapping Files (**.lmf**)" in MAX+PLUS II Help for more information.
3. Open MAX+PLUS II and specify the name of your top-level design file as the project name with the **Project Name** command (File menu). If you open an HDL file in the MAX+PLUS II Text Editor, you can choose the **Project Set Project to Current File** command (File menu) instead.

    ☞ You can also compile a project from a command line. However, the first time you compile a project, the settings you need to specify are easier to specify from within the MAX+PLUS II software. After you have run the graphical user interface for the MAX+PLUS II software at least once, you can more easily use the command-line **setacf** utility to modify options in the Assignment & Configuration File (**.acf**) for the project. Type `setacf -h` ↵ and `maxplus2 -h` ↵ for descriptions of **setacf** and MAX+PLUS II command-line syntax.
4. Choose **Device** (Assign menu) and select the target Altera device family in the *Device Family* drop-down list

box. If you wish to implement the design logic in a specific device, select it in the *Devices* box. Otherwise, select *AUTO* to allow the MAX+PLUS II Compiler to choose the best device(s) in the current device family. If your design entry or synthesis and optimization tool required you to specify a target family and/or device, specify the same information in this dialog box. For information on partitioning logic among multiple devices, go to MAX+PLUS II Help. Choose **OK**.

5. Open the Compiler window by choosing the **Compiler** command (MAX+PLUS II menu). Go through the following steps to specify the options necessary to compile the design file(s) in your project:

    1. Ensure that all EDIF netlist files have the extension **.edf** and choose **EDIF Netlist Reader Settings** (Interfaces menu).

    2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor. This name should be the name of the vendor whose tool(s) you used to create the EDIF netlist files. If your vendor name does not appear, select *Custom* instead.

        If you are compiling a design created with Synopsys FPGA Express software, select *Synopsys*, choose the **Customize** button, enter <*project name*>**.lmf** in the *LMF #1* box, choose **OK**, and skip to step 6.

    3. If you selected an existing vendor name in the *Vendor* box and your project contains design files that require custom LMF mappings, choose the **Customize** button to expand the dialog box to show all settings. Turn on the *LMF #2* checkbox and type your custom LMF's filename in the corresponding text box, or select a name from the *Files* box. The selection in the *Vendor* box will change to *Custom* and all settings will be retained until you change them again.

    4. If you selected *Custom* in the *Vendor* box, choose the **Customize** button to expand the dialog box to show all settings. Any previously defined custom settings will be displayed. Under *Signal Names*, type one or more names with up to 20 total name characters in the *VCC* or *GND* box if your EDIF Input File(s) use one or more names other than `VCC` or `GND` for the global high or low signals. Multiple signal names must be separated by either a comma (,) or a space. Under *Library Mapping Files*, turn on the *LMF #1* checkbox and type a filename in the text box following it, or select a name from the *Files* box. If necessary, specify another LMF name in the *LMF #2* box. Go to MAX+PLUS II Help for detailed information on the settings available in the **EDIF Netlist Reader Settings** dialog box.

    5. Choose **OK**.

6. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, you may need to ensure that all files are present in your project directory, i.e., the same directory as the top-level design file. Otherwise, you must specify the directories containing these files as user libraries with the **User Libraries** command (Options menu).

7. Follow all guidelines that apply to your design entry or synthesis and optimization tool:

    - Exemplar Logic Galileo Extreme-Specific Compiler Settings
    - Synopsys DesignWare-Specific Compiler Settings
    - Converting Synopsys FPGA Compiler & Design Compiler Timing Constraints into MAX+PLUS II-Compatible Format with the **syn2acf** Utility
    - Synplicity Synplify-Specific Compiler Settings

8. If you wish to generate EDIF, VHDL, or Verilog HDL output files for post-compilation simulation or timing analysis with another EDA tool, go through the following steps:

    1. (Optional) Turn on the **Optimize Timing SNF** command (Processing menu) to reduce the size of the output file(s). Turning on this command can reduce the size of output netlists by up to 30%.

        This command does not create optimized timing SNFs on UNIX workstations. However, a non-

☞ optimized timing SNF provides the same functional and timing information as an optimized timing SNF.

2. If you wish to generate EDIF Output Files (**.edo**), go through these steps:

    1. Turn on the **EDIF Netlist Writer** command (Interfaces menu). Then choose the **EDIF Netlist Writer Settings** command (Interfaces menu).

    2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor and choose **OK**. If your vendor name does not appear, select *Custom* instead and specify the settings that are appropriate for your simulation or timing analysis tool. Go to MAX+PLUS II Help for detailed information on the options available in the **EDIF Netlist Writer Settings** dialog box.

    3. To generate an optional Standard Delay Format (SDF) Output File (**.sdo**), choose the **Customize** button to expand the dialog box to show all settings. Select one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**.

    The filenames of the EDIF Output File(s) and optional SDF Output File(s) are the same as the user-defined chip name(s) for the project; if no chip names exist, the Compiler assigns filenames that are based on the project name. For a multi-device project, the Compiler also generates a top-level EDIF Output File that is uniquely identified by "**_t**" appended to the project name. In addition, the Compiler automatically generates a VHDL Memory Model Output File, *<project name>***.vmo**, when it generates an EDIF Output File that contains memory (RAM or ROM).

3. If you wish to generate VHDL Output Files (**.vho**), turn on the **VHDL Netlist Writer** command (Interfaces menu). Then choose **VHDL Netlist Writer Settings** command (Interfaces menu). Select *VHDL Output File (.vho)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF ver. 2.1 files contain timing delay information that allows you to perform back-annotation simulation in VHDL with VITAL-compliant simulation libraries. The VHDL Output Files generated by the Compiler have the extension **.vho**, but are otherwise named in the same way as the EDIF Output Files described above.

4. If you wish to generate Verilog HDL Output Files (**.vo**), turn on the **Verilog Netlist Writer** command (Interfaces menu). Then choose **Verilog Netlist Writer Settings** command (Interfaces menu). Select *Verilog Output File (.vo)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF Output Files contain timing delay information that allows you to perform back-annotation simulation in Verilog HDL. The Verilog Output Files generated by the Compiler have the extension **.vo**, but are otherwise named in the same way as the EDIF Output Files described above.

9. To run the MAX+PLUS II Compiler, choose the **Project Save & Compile** command (File menu) or choose the **Start** button in the Compiler window.

    ☞ See step 3 for information on running MAX+PLUS II software from the command line.

10. Once you have compiled the project with the MAX+PLUS II Compiler, you can use the VHDL, Verilog HDL, or EDIF output file(s), and the optional SDF Output File(s) (**.sdo**) to perform timing analysis or timing simulation with another EDA tool. Specific instructions for some tools are described in these MAX+PLUS II ACCESS Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your EDA tool.

The MAX+PLUS II Compiler also generates a Report File (**.rpt**), a Pin-Out File (**.pin**), and one or more of the following files for device programming or configuration:

- JEDEC Files (**.jed**)
- Programmer Object Files (**.pof**)
- SRAM Object Files (**.sof**)

- Hexadecimal (Intel-format) Files (**.hex**)
- Tabular Text Files (**.ttf**)

Related Topics:

- Refer to the following sources for additional information:
  - Go to *Compiler Procedures* in MAX+PLUS II Help for information on other available Compiler settings.
  - Go to *Programmer Procedures* in MAX+PLUS II Help for instructions on creating other types of programming files and on programming or configuring Altera devices.
  - Go to Back-Annotating MAX+PLUS II Pin Assignments to Design Architect Symbols for information on back-annotating pin assignments in Mentor Graphics Design Architect schematics.
  - Go to Programming Altera Devices for information on the different programming hardware options for Altera device families.
- Go to the following topics, which are available on the web, for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware

Performing a Timing Simulation with DVE & QuickSim II Software

After you have compiled a design with the MAX+PLUS® II Compiler, you can prepare the MAX+PLUS IIgenerated EDIF Output File (**.edo**) with Mentor Graphics Design Viewpoint Editor (DVE) and simulate it with the Mentor Graphics QuickSim II software.

To simulate an EDIF Output File with the QuickSim II software, follow these steps:

1. Be sure to set up the working environment correctly, as described in Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment.

2. Generate an EDIF Output File for your project, as described in Compiling Projects with MAX+PLUS II Software or Using the Altera Schematic Express (**sch_exprss**) Utility.

3. If you used the Altera Schematic Express (**sch_exprss**) utility to process your design, skip to step 5. Otherwise, go to step 4.

4. In the Navigator window, select your project's icon, press Button 3, and choose **Open max2_enr** to read your project's EDIF Output File with the ENRead utility. You can also start ENRead software by typing `max2_enr` ↵ at the UNIX prompt.

5. Select your project's folder, press Button 3, and choose **Open max2_ave** to open DVE, which will prepare your project's simulation component for QuickSim II timing simulation. DVE automatically generates an appropriately named viewpoint for your project. You can also start DVE by typing `max2_ave` ↵ at the UNIX prompt.

6. Select your project's folder, press Button 3, and choose **Open max2_qsim** to simulate your project and its DVE viewpoint with QuickSim II software. You can also start QuickSim II by typing `max2_qsim` ↵ at the UNIX prompt.

7. In the **Altera QuickSim** dialog box, type the name of your project's viewpoint in the *Viewpoint Name* box. Select *Timing* as the *Timing Mode*. Select the *Max timing* option. Choose *Scale Factor* for *Delay Scale*, and be sure that `0.1` is specified for the *Value*. Choose **OK**.

   ☞ If the delay scale value is not set to 0.1 (i.e., divided by ten), the QuickSim II software will not reflect the correct timing simulation values.

Related Topics:

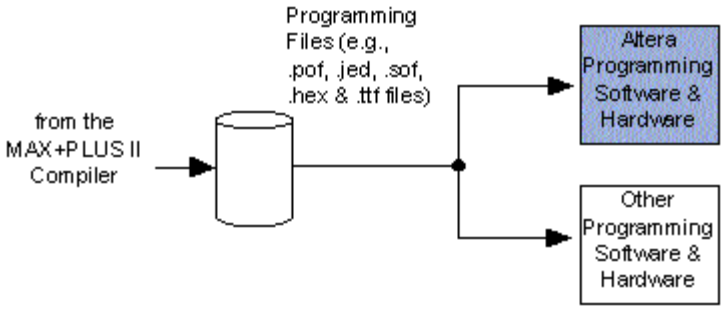
- Go to Performing a Timing Analysis with QuickPath Software in these MAX+PLUS II ACCESS℠ Key topics for related information.

Programming Altera Devices

Once you have successfully compiled and simulated a project with the MAX+PLUS® II software, you can program an Altera® device and test it in the target circuit. Figure 1 shows the device programming flow for MAX+PLUS II software.

Figure 1. MAX+PLUS II Device Programming Flow

Altera-provided items are shown in blue.



You can program devices with Altera programming hardware and MAX+PLUS II Programmer software installed on a 486- or Pentium-based PC or a UNIX workstation, or with programming hardware and software available from other manufacturers. Table 1 shows the available Altera programming hardware options on PCs and UNIX workstations.

Table 1. Altera Programming Hardware

Programming Hardware Option	PCs	UNIX Work-stations	MAX® 3000A Devices	Classic® & MAX 5000 Devices	MAX 7000 & MAX 7000E Devices	MAX 7000A, MAX 7000AE, MAX 7000B, MAX 7000S MAX 9000 & MAX 9000A Devices	FLEX® 6000, FLEX 6000A, FLEX 8000, FLEX 10K, FLEX 10KA, FLEX 10KB, & FLEX 10KE Devices	In-System Programming/ Configuration
Logic Programmer card, PL-MPU Master								

Programming Unit, and device-specific adapters	✓		✓	✓	✓	✓		
BitBlaster™ Download Cable	✓	✓	✓			✓	✓	✓
ByteBlasterMV™ Download Cable	✓		✓			✓	✓	✓
MasterBlaster™ Download Cable	✓	✓	✓			✓	✓	✓

If you wish to transfer programming files from a UNIX workstation to a PC over a network with File Transfer Protocol (FTP) or other similar transfer programs, be sure to select binary transfer mode.

Programming hardware from other manufacturers varies, but typically consists of a device connected to one of the serial ports on the workstation. Various vendors, such as Data I/O and BP Microsystems, supply hardware and software for programming Altera devices.

Related Topics:

- Go to Compiling Projects with MAX+PLUS II Software for information on creating programming files.
- Go to the following topics, which are available on the web, for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware
  - FLEX Devices
  - MAX Devices
  - Classic Device Family

Using Mentor Graphics QuickPath & MAX+PLUS II Software

The following topics describe how to use the Mentor Graphics QuickPath software with MAX+PLUS® II software. Choose one of the following topics for information:

Open a printable version of all topics listed on this page.

Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment

- Software Requirements
- Altera-Provided Logic & Symbol Libraries
- Local Work Area Directory Structure
- Mentor Graphics Project Directory Structure
- MAX+PLUS II Project Directory Structure
- MAX+PLUS II/Mentor Graphics/Exemplar Logic Interface File Organization

Timing Analysis

- Project Simulation/Timing Analysis Flow
- Performing a Timing Analysis with QuickPath Software

Related Links

- Compiling Projects with MAX+PLUS II Software
- Performing a Timing Simulation with DVE & QuickSim II Software
- Programming Altera Devices
- MAX+PLUS II Development Software
- Altera Programming Hardware
- Mentor Graphics web site (http://www.mentor.com)

Using Mentor Graphics QuickSim II & MAX+PLUS II Software

The following topics describe how to use the Mentor Graphics QuickSim II (and DVE) software with MAX+PLUS® II software. Choose one of the following topics for information:

Open a printable version of all topics listed on this page.

Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment

- Software Requirements
- Altera-Provided Logic & Symbol Libraries
- Local Work Area Directory Structure
- Mentor Graphics Project Directory Structure
- MAX+PLUS II Project Directory Structure
- MAX+PLUS II/Mentor Graphics/Exemplar Logic Interface File Organization

Functional Simulation

- Design Entry Flow
- Performing a Functional Simulation with DVE & QuickSim II Software

Timing Simulation

- Project Simulation/Timing Analysis Flow
- Performing a Timing Simulation with DVE & QuickSim II Software

Related Links

- Compiling Projects with MAX+PLUS II Software
- Programming Altera Devices
- MAX+PLUS II Development Software
- Altera Programming Hardware
- Mentor Graphics web site (http://www.mentor.com)

Using Mentor Graphics QuickHDL and QuickHDL Pro & MAX+PLUS II Software

The following topics describe how to use the Mentor Graphics QuickHDL and QuickHDL Pro software with MAX+PLUS® II software. Click on one of the following topics for information:

This file is suitable for printing only. It does not contain hypertext links that allow you to jump from topic to topic.

Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment

- Software Requirements
- Altera-Provided Logic & Symbol Libraries
- Local Work Area Directory Structure
- Mentor Graphics Project Directory Structure
- MAX+PLUS II Project Directory Structure
- MAX+PLUS II/Mentor Graphics/Exemplar Logic Interface File Organization

Functional Simulation

- Design Entry Flow
- Performing a Functional Simulation with QuickHDL Software
- Performing a Functional Simulation with QuickHDL Pro Software

Timing Simulation

- Project Simulation/Timing Analysis Flow
- Initializing Registers in VHDL & Verilog Output Files for Power-Up before Simulation
- Performing a Timing Simulation with QuickHDL Software

Related Topics:

- Go to the following MAX+PLUS II ACCESS℠ Key topics for related information:
  - Creating VHDL & Verilog HDL Designs for Use with MAX+PLUS II Software
  - Compiling Projects with MAX+PLUS II Software
  - Programming Altera Devices
- Go to the following topics, which are available on the web, for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware
  - Mentor Graphics web site (http://www.mentor.com)

Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment

To use the MAX+PLUS ® II software with Mentor Graphics/Exemplar Logic software, you must install the

MAX+PLUS II software, then establish an environment that facilitates entering and processing designs. The MAX+PLUS II/Mentor Graphics/Exemplar Logic interface is installed automatically when you install the MAX+PLUS II software on your computer.

Go to *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual for more information on installation and details on the directories that are created during MAX+PLUS II installation. Go to MAX+PLUS II/Mentor Graphics/Exemplar Logic Interface File Organization for information about the MAX+PLUS II/Mentor Graphics directories that are created during MAX+PLUS II installation.

The information presented here assumes that you are using a C shell and that your MAX+PLUS II system directory is **/usr/maxplus2**. If not, you must use the appropriate syntax and procedures to set environment variables for your shell.

To set up your working environment for the MAX+PLUS II/Mentor Graphics interface, follow these steps:

1. Ensure that you have correctly installed the MAX+PLUS II and Mentor Graphics software versions described in MAX+PLUS II/Mentor Graphics Software Requirements.

2. Add the following environment variables to your **.cshrc** file:

   ```
   setenv ALT_HOME /usr/maxplus2 ↵
   setenv MGC_WD <user-specified working directory> ↵
   setenv MGC_HOME <Mentor Graphics system directory> ↵
   setenv MAX2_MENTOR /usr/maxplus2/mentor/max2 ↵
   setenv MGC_LOCATION_MAP <user-specified location_map file> ↵
   setenv EXEMPLAR <Galileo or Leonardo system directory> ↵
   ```

   Installing the Altera® provided MAX+PLUS II/Mentor Graphics interface on your computer automatically installs a template for these environment variables in the **/usr/maxplus2/mentor/max2/.cshrc** file.

3. Add the **$MGC_HOME/bin**, **$MAX2_MENTOR/bin**, **$ALT_HOME/bin**, **$EXEMPLAR/bin/**<*os*>, and **$ALT_HOME/bin** directories to the `PATH` environment variable in your **.cshrc** file, where <*os*> is the operating system, e.g., `SUN4` for SunOS; `SUN5` for Solaris.

4. If you plan to use the Altera Schematic Express (**sch_exprss**) utility or the Altera VHDL Express (**vhd_exprss**) utility, add the following environment variable to your **.cshrc** file:

   ```
   setenv MAX2_QSIM /usr/maxplus2/simlib/mentor/max2sim ↵
   ```

5. Type `source ~/.cshrc` at a UNIX prompt to source the **.cshrc** file and validate the settings in steps 1 through 4.

6. Add the following lines to your **MGC_location_map** file:

   ```
   $MAX2_MENTOR ↵
   /usr/maxplus2/mentor/max2 ↵
   $MGC_GENLIB ↵
   /<user-specified Mentor Graphics GEN_LIB directory> ↵
   $MGC_LSLIB ↵
   /<user-specified Mentor Graphics LS_LIB directory> ↵
   $MAX2_EXAMPLES ↵
   /<user-specified example directory> ↵
   ```

```
$MAX2_LMCLIB ↵
/<user-specified Logic Modeling directory> ↵
$MAX2_GENLIB ↵
/usr/maxplus2/simlib/mentor/alt_max2 ↵
$MAX2_QSIM ↵
/usr/maxplus2/simlib/mentor/max2sim ↵
$MAX2_FONT ↵
/usr/maxplus2/mentor/max2/fonts ↵
$MGC_SYS1076_STD ↵
/<user-specified MGC_HOME directory>/pkgs/sys_1076_std/ std ↵
$MGC_SYS1076_ARITHMETIC ↵
/<user-specified MGC_HOME directory>/pkgs/sys_1076_std/arithmetic ↵
$MGC_SYS1076_PORTABLE ↵
/<user-specified MGC_HOME directory>/pkgs/sys_1076_std/mgc_portable ↵
$MGC_SYS1076_IEEE ↵
/<user-specified MGC_HOME directory>/pkgs/sys_1076_std/ieee ↵
$MGC_SYS1076_SRC ↵
/<user-specified MGC_HOME directory>/pkgs/sys_1076_std/ src ↵
$MAX2_MFLIB ↵
/usr/maxplus2/simlib/mentor/alt_mf ↵
```

Installing the Alteraprovided MAX+PLUS II/Mentor Graphics/Exemplar Logic interface on your computer automatically installs a template for these environment variables in the **/usr/maxplus2/mentor/max2/location_map/location_map** file.

7. If you want to use QuickHDL software to simulate VHDL or Verilog HDL designs, add the following line in the `[library]` section of your **quickhdl.ini** file: `altera = $MAX2_MFLIB`.

8. If you plan to use QuickHDL software to simulate VITAL-compliant VHDL files, add the following lines to your **MGC_location_map** file:

```
$MAX2_VTLLIB ↵
/usr/maxplus2/simlib/mentor/alt_vtl ↵
```

9. Copy the **/usr/maxplus2/maxplus2.ini** file to your **$HOME** directory:

```
cp /usr/maxplus2/maxplus2.ini $HOME ↵
chmod u+w $HOME/maxplus2.ini ↵
```

The **maxplus2.ini** file contains both Altera- and user-specified initialization parameters that control the MAX+PLUS II software, such as Alteraprovided logic and symbol library paths and the current project name. The MAX+PLUS II installation procedure creates and copies the **maxplus2.ini** file to the **/usr/maxplus2** directory.

Normally, you do not have to edit your local copy of **maxplus2.ini**, because the MAX+PLUS II software updates the file automatically whenever you change any parameters or settings. However, if you move the **max2lib** and **max2inc** library subdirectories, you must update the file. Go to "Creating & Using a Local Copy of the **maxplus2.ini** File" in MAX+PLUS II Help for more information.

Related Topics:

- Go to the following topics, which are available on the web, for additional information:
  - ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
  - This manual is also available in 4 parts:
    - Preface & Section 1: MAX+PLUS II Installation
    - Section 2: MAX+PLUS II - A Perspective
    - Section 3: MAX+PLUS II Tutorial
    - Appendices, Glossary & Index

MAX+PLUS II/Mentor Graphics Software Requirements

The following products are used to generate, process, synthesize, and verify a project with the MAX+PLUS ® II software and Mentor Graphics software:

Mentor Graphics		Exemplar	Altera
version C.1:			
System_1076 Compiler QuickSim II Design Architect ENRead ENWrite GEN_LIB library	QuickHDL QuickHDL Pro QuickPath LS_LIB library (optional) DVE	Galileo Extreme version 4.1.1 Leonardo version 4.1.3	MAX+PLUS II version 9.4

☞ The MAX+PLUS II **read.me** file provides up-to-date information on which versions of Mentor Graphics applications are supported by the current version of MAX+PLUS II. It also provides information on installation and operating requirements. You should read the **read.me** file on the CD-ROM before installing the MAX+PLUS II software. After installation, you can open the **read.me** file from the MAX+PLUS II Help menu.

Altera-Provided Logic & Symbol Libraries

The MAX+PLUS ® II/Mentor Graphics environment provides libraries for compiling, synthesizing, and simulating designs.

☞ You can create your own libraries of custom functions for use in Design Architect schematics and VHDL and Verilog HDL design files. You can use custom functions to incorporate an EDIF Input File (**.edf**), Text Design File (**.tdf**), or any other MAX+PLUS II-supported design file into a project. The MAX+PLUS II software uses the Altera® provided **mnt8_bas.lmf** and **exemplar.lmf** Library Mapping Files to map standard Design Architect symbols and VHDL and Verilog HDL functions to equivalent MAX+PLUS II logic functions. To use custom functions, you can create a custom LMF that maps your custom functions to the equivalent EDIF input file, TDF, or other design file. Go to "Library Mapping File" in MAX+PLUS II Help for more information.

Design Architect Libraries

You can enter a Design Architect schematic with logic functions from these Altera-provided symbol libraries: **ALTERA LPMLIB**, **ALTERA GENLIB**, **LSTTL BY TYPE**, and **LSTTL ALL PARTS**. You can access these libraries by choosing **Altera Libraries** (Libraries menu) in the Design Architect software. For information on using library of parameterized modules (LPM) functions, see **ALTERA LPMLIB** Library below.

ALTERA GENLIB Library (Design Architect) & Altera (VHDL) Libraries

The **ALTERA GENLIB** symbol library (called the **Altera** library for VHDL) includes several MAX+PLUS II primitives for controlling design synthesis and fitting. It also includes four macrofunctions (`8count`, `8mcomp`, `8fadd`,

and 81mux) that are optimized for different Altera device families, and the clklock phase-locked loop megafunction, which is supported for some FLEX ® 10K devices.

The following table shows the MAX+PLUS II-specific logic functions.

Table 1. MAX+PLUS II-Specific Logic Functions

Macrofunctions *Note (1)*				**Primitives**	
Name	**Description**	**Name**	**Description**	**Name**	**Description**
8fadd	8-bit full adder	LCELL	Logic cell buffer	EXP	MAX ® 5000, MAX 7000 , and MAX 9000 Expander buffer
8mcomp	8-bit magnitude comparator	GLOBAL	Global input buffer	SOFT	Soft buffer
8count	8-bit up/down counter	CASCADE	FLEX 6000, FLEX 8000, and FLEX 10K cascade buffer	OPNDRN	Open-drain buffer
81mux	8-to-1 multiplexer	CARRY	FLEX 6000, FLEX 8000, and FLEX 10K carry buffer	DFFE DFFE6K *Note (2)*	D-type flipflop with Clock Enable
clklock	Phase-locked loop				

Notes:

1. Logic function names that begin with a number must be preceded by "a_" in VHDL designs. For example, 8fadd must be specified as a_8fadd instead.
2. If you want to use QuickHDL software, make sure you have updated your **quickhdl.ini** file, as described in step 7 of Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment.
3. For designs that are targeted for FLEX 6000 devices, you should use the DFFE primitive only if the design contains either a Clear or Preset signal, but not both. If your design contains both a Clear and a Preset signal, you must use the DFFE6K primitive.

☞ Choose **Old-Style Macrofunctions**, **Primitives**, or **Megafunctions/LPM** from the MAX+PLUS II Help menu for detailed information on these functions.

ALTERA LPMLIB Library

The Alteraprovided **ALTERA LPMLIB** library, which is available for Design Architect schematics and VHDL designs, includes standard functions from the library of parameterized modules (LPM) 2.1.0, except the truth table, finite state machine, and pad functions. The LPM standard defines a set of parameterized modules (i.e., parameterized functions) and their corresponding representations in an EDIF netlist file. These logic functions allow you to create and functionally simulate an LPM-based design without targeting a specific device family. After the design is completed, you can target the design to any device family. The parameters you specify for each LPM function determine which simulation models are generated.

☞ Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu for more information about LPM functions.

Related Topics:

- Go to the following topics, which are available on the web, for additional information:
  - FLEX Devices
  - MAX Devices
  - Classic Device Family

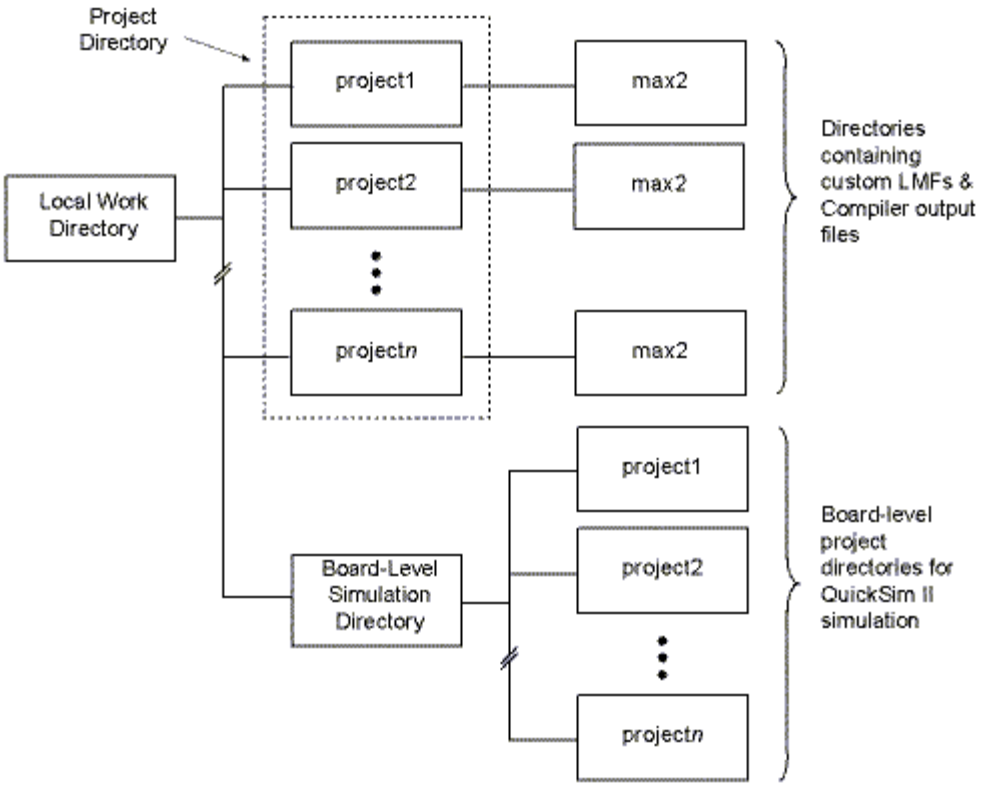
Local Work Area Directory Structure

Design Architect software automatically creates and maintains the project directory structure required for all stages of design entry. Galileo Extreme, Leonardo, and ENWrite software create a **max2** subdirectory, if it does not already exist, under the project directory. These software applications also generate EDIF netlist files, and copy them from the <*project name*> directory to this **max2** subdirectory. All MAX+PLUS ® II Compiler output files are created in the **max2** subdirectory.

Simulation files created with Mentor Graphics applications and Logic Modeling files are located in the board-level simulation subdirectory of the project directory. You can use these files during simulation with QuickSim II software.

The only directory you need to create is the local work directory, which should contain all project directories. Figure 1 shows the recommended file structure.

Figure 1. Recommended File Structure



Related Topics:

- Go to the following MAX+PLUS II ACCESS^{SM} Key topics for related information:
  - MAX+PLUS II Project Directory Structure
  - Mentor Graphics Project Directory Structure

Mentor Graphics Project Directory Structure

Design Architect software generates the following files for each schematic:

- <*drawing name*>/**mgc_component.attr**
- <*drawing name*>/**part.Eddm_part.attr**

- <*drawing name*>**/part.part_1**
- <*drawing name*>**/schematic.mgc_schematic.attr**
- <*drawing name*>**/schematic/schem_id**
- <*drawing name*>**/schematic/sheet1.mgc_sheet.attr**
- <*drawing name*>**/schematic/sheet1.sgfx_1**
- <*drawing name*>**/schematic/sheet1.ssht_1**

The files generated for each schematic are stored in the schematic's <*drawing name*> directory and should not be edited. Mentor Graphics software automatically manages file storage and retrieval operations through this <*drawing name*> directory structure, which does not reflect hierarchical design relationships. Figure 1 shows a sample file structure with **project1** as the UNIX project directory, and **design1**, **subdesign1**, and **subdesign2** as the directories for the top-level design and subdesigns of the project.

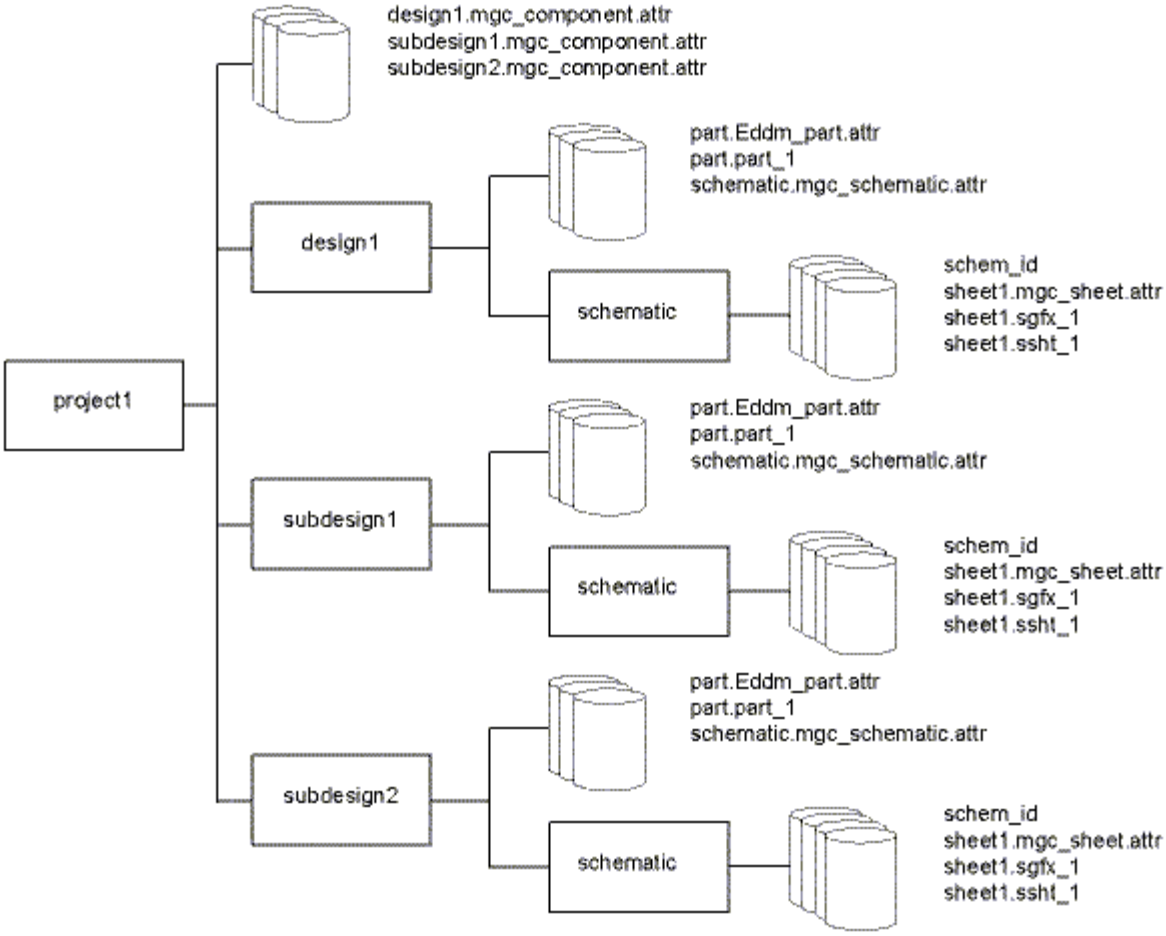


Figure 1. Design Architect Project File Structure

When the ENWrite utility converts the schematic into an EDIF netlist file, it processes the design information and all related file subdirectories, then creates the EDIF netlist file in the directory defined by the user. The EDIF netlist file is named <*project name*>**.edf**, where <*project name*> is the name of the top-level design file. The <*project name*>**.edf** file is automatically moved to the **max2** directory under the project directory.

Related Topics:

- Go to the following MAX+PLUS II ACCESS^{SM} Key topics for related information:
  - Local Work Area Directory Structure
  - MAX+PLUS II Project Directory Structure

MAX+PLUS II Project Directory Structure

In the MAX+PLUS ® II software, a project name is the name of a top-level design file, without the filename extension. This design file can be an EDIF, VHDL, or Verilog HDL netlist file; an Altera Hardware Description Language (AHDL) Text Design File (TDF); or any other MAX+PLUS II-supported design file. The EDIF netlist file must be created by ENWrite, Galileo Extreme, or Leonardo software and imported into MAX+PLUS II as an EDIF Input File (**.edf**). Figure 1 shows an example of a MAX+PLUS II project directory.

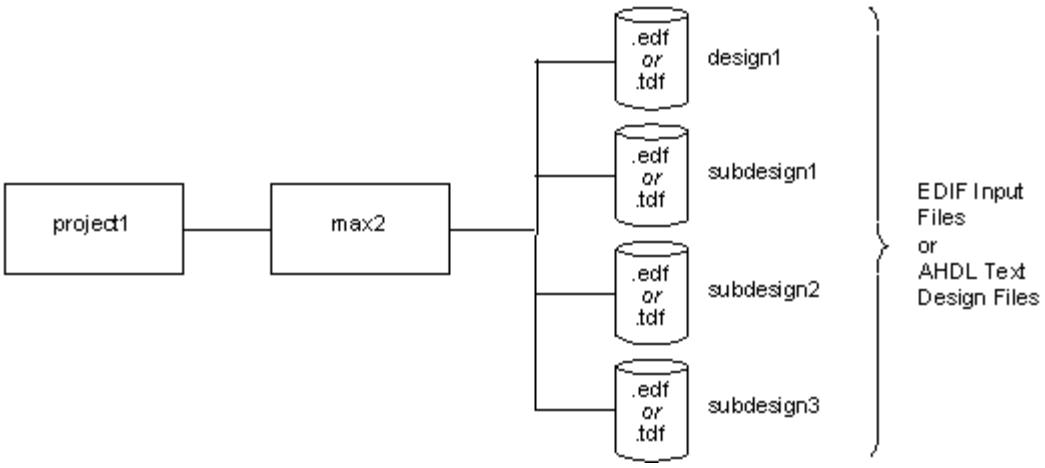


Figure 1. Sample MAX+PLUS II Project Directory

The MAX+PLUS II software stores the connectivity data on the links between design files in a hierarchical project in a Hierarchy Interconnect File (**.hif**), but refers to the entire project only by its project name. The MAX+PLUS II Compiler uses the HIF to build a single, fully flattened project database that integrates all the design files in a project hierarchy.

Related Topics:

- Go to the following MAX+PLUS II ACCESS^{SM} Key topics for related information:
  - Local Work Area Directory Structure
  - Mentor Graphics Project Directory Structure

MAX+PLUS II/Mentor Graphics/Exemplar Logic Interface File Organization

The following table shows the MAX+PLUS ® II/Mentor Graphics interface subdirectories that are created in the MAX+PLUS II system directory (by default, the **/usr/maxplus2** directory) during MAX+PLUS II installation.

Related Topics:

- For information on the other directories that are created during MAX+PLUS II installation, see "MAX+PLUS II File Organization" in *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual.

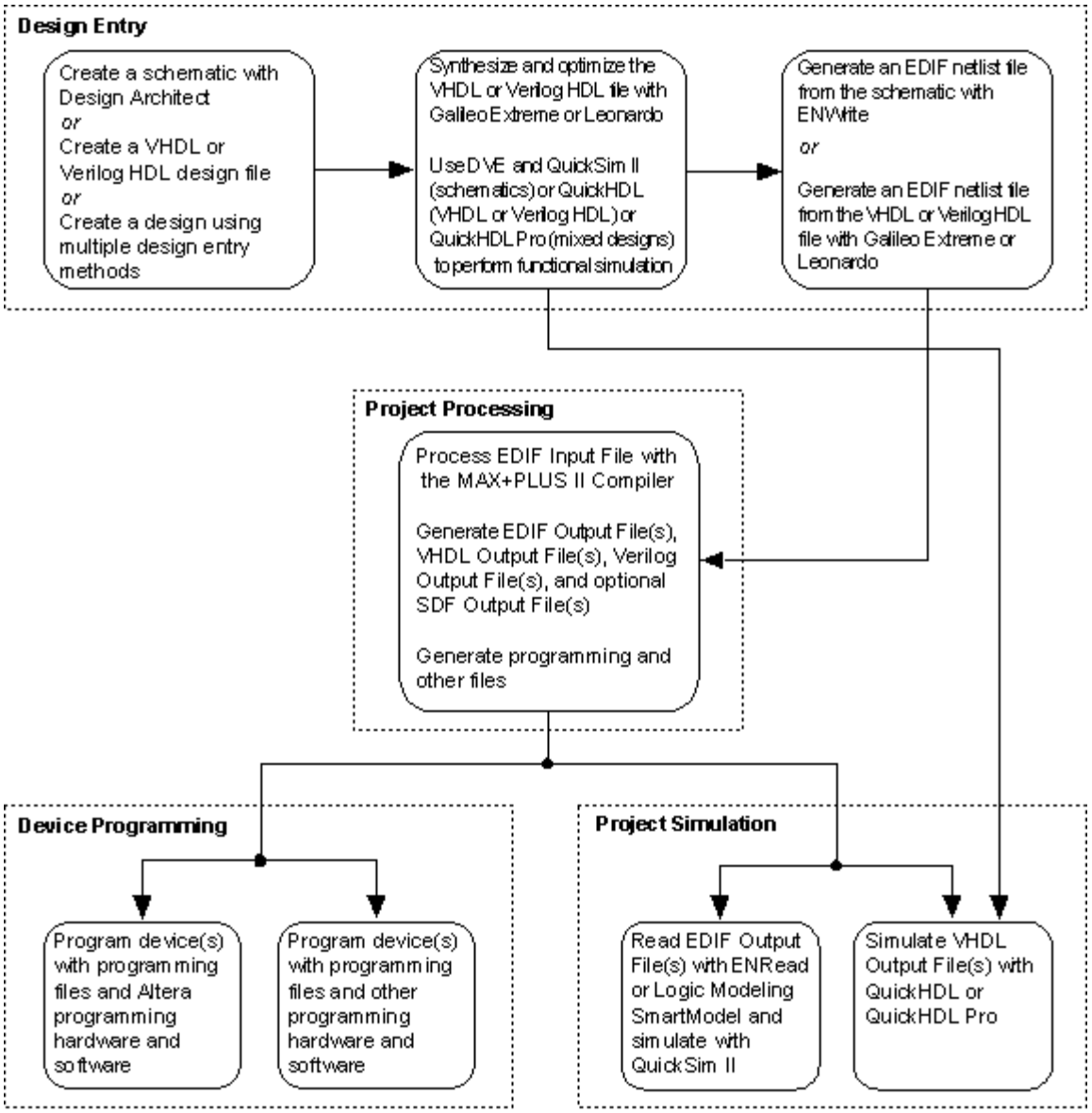
Table 1. MAX+PLUS II Directory Organization

Directory	Description
.lmf	Contains the Altera-provided Library Mapping Files, **mnt8_bas.lmf** and **exemplar.lmf**, that map Mentor Graphics and Exemplar Logic logic functions to equivalent MAX+PLUS II logic functions.
./mentor	Contains the AMPLE userware for the MAX+PLUS II/Mentor Graphics interface.
	Contains MAX+PLUS II primitives such as `CARRY`, `CASCADE`, `EXP`, `GLOBAL`, `LCELL`, `SOFT`,

./simlib/mentor/alt_max2	OPNDRN, DFFE, and DFFE6K (D flipflop with Clock Enable) for use in Design Architect schematics.
./simlib/mentor/max2sim	Contains the MAX+PLUS II/Mentor Graphics simulation model library, **max2sim**, for use with QuickSim II and QuickPath software.
./simlib/mentor/synlib	Contains the MAX+PLUS II synthesis library for use with AutoLogic II software, which supports synthesis for users running Mentor Graphics version B1.
./simlib/mentor/alt_mf	Contains the MAX+PLUS II macrofunction and megafunction libraries.
./simlib/mentor/alt_vtl	Contains the MAX+PLUS II VITAL library.

Altera/Mentor Graphics/Exemplar Logic Design Flow

The following figure shows the typical design flow for logic circuits created and processed with the MAX+PLUS ® II and Mentor Graphics/Exemplar Logic software. Detailed diagrams for each stage of the design flow appear in Design Entry Flow, Project Compilation Flow, Project Simulation/Timing Analysis Flow, and Device Programming Flow.



Performing a Functional Simulation with QuickHDL Software

You can use Mentor Graphics QuickHDL software to functionally simulate VHDL or Verilog HDL design files before compiling them with the MAX+PLUS ® II Compiler.

If you wish to functionally simulate a hierarchical design that uses multiple design entry methods, you should use QuickHDL Pro rather than QuickHDL. Refer to Performing a Functional Simulation with QuickHDL Pro Software for more information.

To functionally simulate a VHDL or Verilog HDL design, follow these steps:

1. Be sure to set up the working environment correctly, as described in Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment.
2. Create a VHDL or Verilog HDL design file that follows the guidelines described in Creating VHDL & Verilog HDL Designs for Use with MAX+PLUS II Software.
3. Start Design Architect by double-clicking Button 1 on the **max_da** icon in the Design Manager tools window. You can also start Design Architect software by typing `max2_da` ↵ at the UNIX prompt.
4. Choose **Lib** (QuickHDL menu) and specify your work library name as the *Work Library* name. Choose **OK**.
5. Choose **Map** (QuickHDL menu) to map the instantiated function to the equivalent function in the **Altera** logic function library. Choose **Set** to specify *altera* as the *Logical Name* and *$MAX2_MFLIB* as the *Physical Name*. Choose **OK**.
6. Choose **Compile** (QuickHDL menu) and use the Navigator window to select the icon for your project. Specify your work library name as the *Work Library* name and select the *Simulation* setting in the Set VHDL Compilation Options or Set Verilog HDL Compilation Options window. Choose **OK** to compile.
7. Choose **Simulate** (QuickHDL menu) and specify your work library name as the *Work Library* name. Choose **OK** to start the QuickHDL Startup window.
8. Select the icon for your project in the Entity Configuration window and choose **OK** to simulate the design.
9. Synthesize and optimize the design, as described in Synthesizing & Optimizing VHDL & Verilog HDL Projects with Galileo Extreme Software or Synthesizing & Optimizing VHDL & Verilog HDL Projects with Leonardo Software.

If your Verilog HDL design uses memory functions (RAM or ROM) that can be initialized with a hexadecimal file (Intel-format) initialization, you must convert these files into Verilog HDL format using the Programming Language Interface (PLI). To use the Altera-provided source code for PLI, perform the following steps:

1. Download the file **http://www.edif.org/lpmweb/convert_hex2ver.c** to your project directory.
2. Copy the following two files from the **$MGC_HOME/shared/pkgs/quickhdl/include** directory into the **/usr/maxplus2** directory:

- **$MGC_HOME/shared/pkgs/quickhdl/include/veriuser**
- **$MGC_HOME/shared/pkgs/quickhdl/include/acc_user**

Refer to the ***Mentor Graphics QuickHDL User's Reference Manual***, version 8.5-4.6i, for information on compiling the PLI application on different platforms and using the Verilog HDL PLI.

Related Topics:

- Go to the following MAX+PLUS II ACCESS℠ Key topics for related information:
  - Compiling Projects with MAX+PLUS II Software

- Performing a Timing Simulation with QuickHDL Software
- Performing a Functional Simulation with QuickHDL Pro Software

---

Performing a Functional Simulation with QuickHDL Pro Software

You can use Mentor Graphics QuickHDL Pro software to functionally simulate mixed-level schematic and VHDL designs before compiling them with the MAX+PLUS ® II Compiler.

Refer to Mentor Graphics *Getting Started with QuickHDL Pro* page 2-1 and 3-1 for compatible design configurations.

To functionally simulate a QuickHDL at Top Level design, follow the steps in *Getting Started with QuickHDL Pro*, Chapter 2.

To functionally simulate a QuickSim II at Top Level design, go through the following steps:

1. Be sure to set up the working environment correctly, as described in Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment.
2. Create a schematic design using QuickHDL models. Refer to Creating Design Architect Schematics for Use with MAX+PLUS II Software.
3. Compile the QuickHDL model using the QuickHDL Compiler with the **-qhpro_syminfo** option. (This is done automatically for LPM functions if you choose to compile the LPM models when saving the schematic.)
4. Start Design Architect by double-clicking Button 1 on the **max_da** icon in the Design Manager tools window.
5. Choose **Open** from the File menu, then choose **Sheet** from the Open menu to open the top level schematic.
6. Select the symbol for the VHDL model and choose **Begin Edit Symbol** from the Edit menu.
7. Press Button 3 to display the the Design Architect pop-up menu. Choose Add Menu from the Other Menus menu, then choose **Set VHDL Info**. Choose Import from Entity to display the "Import Entity Info" dialog box.
8. Specify the following options in the "Import Entity Info" dialog box:

    1. *QHDL InitFile*: Specify your **quickhdl.ini** file.

    2. *Library Logical Name*: Click on **Choose Library** button and fill the "Choose VHDL Library" form with your work library.

    3. *Entity Name*: Click on **Choose Entity** button and select the name of your entity.

    4. *Default Architecture*: Click on **Choose Arch** button and select corresponding architecture for the entity.

After filling in the above information, click on **OK** to close the form.

- Check the symbol with defaults. If there are no errors, save the symbol with default registration by choosing **Save Symbol** from the File menu, then choose **Default Registration**.
- Choose **End Edit Symbol** from the Edit menu to close the Symbol Editor session. In the schematic window, select the symbol you have just edited and choose **Object** from the Report menu, then choose **All** from the Selected menu. In the report transcript, make sure the *MODEL* property is set to *qhpro* to ensure that the model will work with QuickHDL Pro.
- Select the folder for your project, press button 3, and choose **Open max2_qvpro** to start QuickHDL Pro. You can also start QuickHDL Pro by typing max2_qvpro ↵ at the UNIX prompt. In the **QVHDL Pro System** dialog box, make sure *EDDM Design* is selected for *Invoke on* and the correct path name is specified for the design. Choose **OK** to start the QuickHDL Pro. A QHPro (QuickSim II) window and a QHPro (QuickHDL) window appear on the screen.
- Use the QuickSim II window to simulate the top level schematic and the QuickHDL window to simulate the

VHDL portion of the design.

Related Topics:

- Go to the following MAX+PLUS II ACCESS[SM] Key topics for related information:
  - Compiling Projects with MAX+PLUS II Software
  - Instantiating LPM Functions in Design Architect Schematics
  - Performing a Functional Simulation with QuickHDL Software

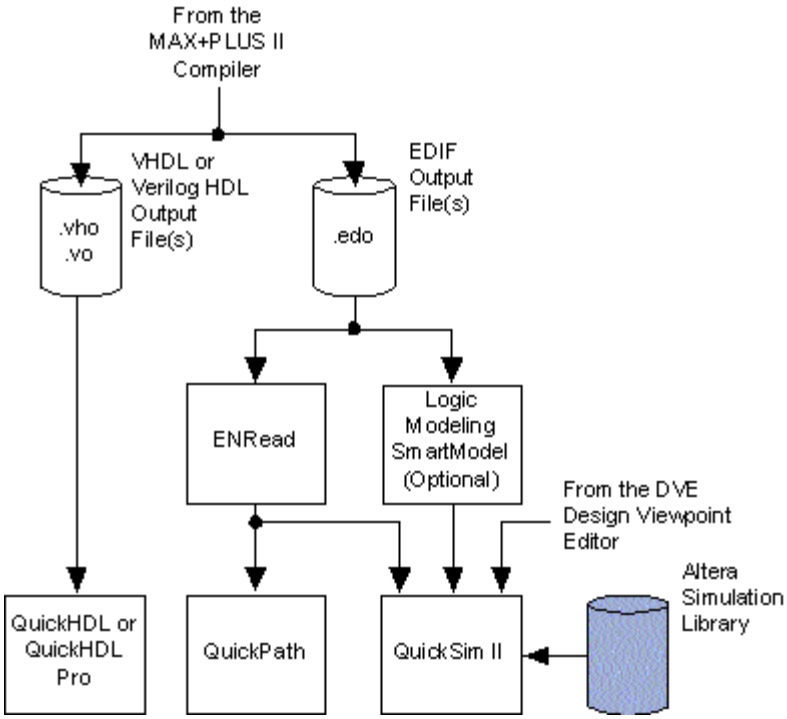
---

Project Simulation/Timing Analysis Flow

The following figure shows the project simulation and timing analysis flow for the MAX+PLUS® II /Mentor Graphics interface.

Figure 1. MAX+PLUS II/Mentor Graphics Project Simulation/Timing Analysis Flow

Alteraprovided items are shown in blue.



Performing a Timing Simulation with QuickHDL Software

After you have entered a VHDL or Verilog HDL design file and compiled it with the MAX+PLUS ® II Compiler, you can use Mentor Graphics QuickHDL software to simulate the MAX+PLUS IIgenerated VHDL Output File (**.vhd**) or Verilog Output File (**.vo**) and the Standard Delay Format (SDF) Output File (**.sdo**).

To simulate your VHDL or Verilog HDL design, go through the following steps:

1. Be sure to set up the working environment correctly, as described in Setting Up the MAX+PLUS II/Mentor

Graphics/Exemplar Logic Working Environment.

2. Generate a VHDL or Verilog HDL output file and an SDF output file for your project, as described in Compiling Projects with MAX+PLUS II Software.

3. Change to your project's directory.

4. Copy your **quickhdl.ini** file to the same directory as your VHDL or Verilog HDL file.

5. Type the following sets of commands at the UNIX prompt to create the work library and compile your project's VHDL or Verilog HDL output file:

VHDL:	**Verilog HDL:**
`setenv MGC_WD 'pwd'` ↵	`setenv MGC_WD 'pwd'` ↵
`qhlib work` ↵	`qhlib work` ↵
`qvhcom <`*project name*`>.vho` ↵	`qvlcom <`*project name*`>.vo` ↵

6. Type `qhsim -sdftyp <`*project name*`>.sdo` ↵ at the UNIX prompt to perform timing back-annotation and simulation and to display the QuickHDL simulation window.

If your Verilog HDL design uses memory functions (RAM or ROM) that can be initialized with a hexadecimal file (Intel-format) initialization, you must convert these files into Verilog HDL format using the Programming Language Interface (PLI). To use the Altera-provided source code for PLI, perform the following steps:

1. Download the file **http://www.edif.org/lpmweb/convert_hex2ver.c** to your project directory.

2. Copy the following two files from the **$MGC_HOME/shared/pkgs/quickhdl/include** directory into the **/usr/maxplus2** directory:

    - **$MGC_HOME/shared/pkgs/quickhdl/include/veriuser**
    - **$MGC_HOME/shared/pkgs/quickhdl/include/acc_user**

Refer to the ***Mentor Graphics QuickHDL User's Reference Manual***, version 8.5-4.6i, for information on compiling the PLI application on different platforms and using the Verilog HDL PLI.

Related Topics:

- Go to Performing a Functional Simulation with QuickHDL Software in these MAX+PLUS II ACCESS^{SM} Key topics for related information.

Creating VHDL & Verilog HDL Designs for Use with MAX+PLUS II Software

You can create VHDL and Verilog HDL design files with the MAX+PLUS ® II Text Editor or another standard text editor and save them in the appropriate directory for your project.

The MAX+PLUS II Text Editor offers the following advantages:

- Templates are available with the **VHDL Templates** and **Verilog Templates** commands (Template menu). These templates are also available in the ASCII **vhdl.tmp** and **verilog.tmp** files, respectively, which are located in the **/usr/maxplus2** directory.

- If you use the MAX+PLUS II Text Editor to create your VHDL design, you can turn on the **Syntax Coloring** command (Options menu). The Syntax Coloring feature displays keywords and other elements of text in text

files in different colors to distinguish them from other forms of syntax.

To create a VHDL or Verilog HDL design file for use with the MAX+PLUS II software, go through the following steps:

1. Enter a VHDL or Verilog HDL design in the MAX+PLUS II Text Editor or another standard text editor and save it in your working directory.

2. Enter primitives, macrofunctions, and megafunctions in your VHDL or Verilog HDL design from the **Altera** library.

   The following topics describe special steps needed to instantiate LPM and `clklock` functions:

   - Instantiating LPM Functions in VHDL
   - Instantiating the clklock Megafunction in VHDL or Verilog HDL

   You can instantiate MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.

3. (Optional) Use the QuickHDL software to functionally simulate the design file, as described in Performing a Functional Simulation with QuickHDL Software and Performing a Functional Simulation with QuickHDL Pro Software.

4. Once you have created a VHDL or Verilog HDL design, you can generate an EDIF netlist file that can be imported into the MAX+PLUS II software with either of the following methods:

   - You can synthesize and optimize your design and create an EDIF netlist file, as described in Synthesizing & Optimizing VHDL & Verilog HDL Projects with Galileo Extreme Software or Synthesizing & Optimizing VHDL & Verilog HDL Projects with Leonardo Software.

   - You can use the Altera VHDL Express utility, **vhd_exprss**, to automatically create an EDIF netlist file, compile it with the MAX+PLUS II Compiler, generate an EDIF Output File (**.edo**), and prepare the EDIF Output File for simulation with QuickHDL software, as described in Using the Altera Schematic Express (**vhd_exprss**) Utility.

   Installing the Alteraprovided MAX+PLUS II/Mentor Graphics/Exemplar Logic interface on your computer automatically creates the following sample VHDL design files:

- **/usr/maxplus2/examples/mentor/example5/count4.vhd**
- **/usr/maxplus2/examples/mentor/example6/count8.vhd**
- **/usr/maxplus2/examples/mentor/example8/adder16.vhd**

Related Topics:

- Go to Compiling Projects with MAX+PLUS II Software in these MAX+PLUS II ACCESS℠ Key topics for related information.

Compiling Projects with MAX+PLUS II Software

The MAX+PLUS® II Compiler can process design files in a variety of formats. This topic describes how to use MAX+PLUS II software to compile projects in which the top-level design file is an EDIF Input File (with the

extension **.edf**).

Related Topics:

- Refer to the following sources for additional information:
    - Go to MAX+PLUS II Help for information on compiling VHDL and Verilog HDL, design files directly with the MAX+PLUS II Compiler.
    - Go to Running Synopsys Compilers from MAX+PLUS II Software for information on running the Synopsys Design Compiler or FPGA Compiler software on a VHDL or Verilog HDL design from within the MAX+PLUS II Compiler window.

To compile a design (also called a "project") with MAX+PLUS II software, go through the following steps:

1. Create design files that are compatible with the MAX+PLUS II software and convert them into EDIF Input Files with the extension **.edf**. Specific instructions for some tools are described in these MAX+PLUS II ACCESS^{SM} Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your design entry or synthesis and optimization tool.

2. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, create a mapping for each function in a Library Mapping File (**.lmf**) to map the custom symbol to the corresponding EDIF Input File, AHDL Text Design File (**.tdf**), or other MAX+PLUS II-supported design file. These custom functions are represented in design files as hollow-body symbols or "black box" HDL descriptions.

   ☞ Go to "Library Mapping Files (**.lmf**)" in MAX+PLUS II Help for more information.

3. Open MAX+PLUS II and specify the name of your top-level design file as the project name with the **Project Name** command (File menu). If you open an HDL file in the MAX+PLUS II Text Editor, you can choose the **Project Set Project to Current File** command (File menu) instead.

   ☞ You can also compile a project from a command line. However, the first time you compile a project, the settings you need to specify are easier to specify from within the MAX+PLUS II software. After you have run the graphical user interface for the MAX+PLUS II software at least once, you can more easily use the command-line **setacf** utility to modify options in the Assignment & Configuration File (**.acf**) for the project. Type `setacf -h` ↵ and `maxplus2 -h` ↵ for descriptions of **setacf** and MAX+PLUS II command-line syntax.

4. Choose **Device** (Assign menu) and select the target Altera device family in the *Device Family* drop-down list box. If you wish to implement the design logic in a specific device, select it in the *Devices* box. Otherwise, select *AUTO* to allow the MAX+PLUS II Compiler to choose the best device(s) in the current device family. If your design entry or synthesis and optimization tool required you to specify a target family and/or device, specify the same information in this dialog box. For information on partitioning logic among multiple devices, go to MAX+PLUS II Help. Choose **OK**.

5. Open the Compiler window by choosing the **Compiler** command (MAX+PLUS II menu). Go through the following steps to specify the options necessary to compile the design file(s) in your project:

    1. Ensure that all EDIF netlist files have the extension **.edf** and choose **EDIF Netlist Reader Settings** (Interfaces menu).

    2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor. This name should be the name of the vendor whose tool(s) you used to create the EDIF netlist files. If your vendor name does not appear, select *Custom* instead.

If you are compiling a design created with Synopsys FPGA Express software, select *Synopsys*, choose the **Customize** button, enter <*project name*>**.lmf** in the *LMF #1* box, choose **OK**, and skip to step 6.

3. If you selected an existing vendor name in the *Vendor* box and your project contains design files that require custom LMF mappings, choose the **Customize** button to expand the dialog box to show all settings. Turn on the *LMF #2* checkbox and type your custom LMF's filename in the corresponding text box, or select a name from the *Files* box. The selection in the *Vendor* box will change to *Custom* and all settings will be retained until you change them again.

4. If you selected *Custom* in the *Vendor* box, choose the **Customize** button to expand the dialog box to show all settings. Any previously defined custom settings will be displayed. Under *Signal Names*, type one or more names with up to 20 total name characters in the *VCC* or *GND* box if your EDIF Input File(s) use one or more names other than `VCC` or `GND` for the global high or low signals. Multiple signal names must be separated by either a comma (,) or a space. Under *Library Mapping Files*, turn on the *LMF #1* checkbox and type a filename in the text box following it, or select a name from the *Files* box. If necessary, specify another LMF name in the *LMF #2* box. Go to MAX+PLUS II Help for detailed information on the settings available in the **EDIF Netlist Reader Settings** dialog box.

5. Choose **OK**.

6. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, you may need to ensure that all files are present in your project directory, i.e., the same directory as the top-level design file. Otherwise, you must specify the directories containing these files as user libraries with the **User Libraries** command (Options menu).

7. Follow all guidelines that apply to your design entry or synthesis and optimization tool:

   - Exemplar Logic Galileo Extreme-Specific Compiler Settings
   - Synopsys DesignWare-Specific Compiler Settings
   - Converting Synopsys FPGA Compiler & Design Compiler Timing Constraints into MAX+PLUS II-Compatible Format with the **syn2acf** Utility
   - Synplicity Synplify-Specific Compiler Settings

8. If you wish to generate EDIF, VHDL, or Verilog HDL output files for post-compilation simulation or timing analysis with another EDA tool, go through the following steps:

   1. (Optional) Turn on the **Optimize Timing SNF** command (Processing menu) to reduce the size of the output file(s). Turning on this command can reduce the size of output netlists by up to 30%.

      This command does not create optimized timing SNFs on UNIX workstations. However, a non-optimized timing SNF provides the same functional and timing information as an optimized timing SNF.

   2. If you wish to generate EDIF Output Files (**.edo**), go through these steps:

      1. Turn on the **EDIF Netlist Writer** command (Interfaces menu). Then choose the **EDIF Netlist Writer Settings** command (Interfaces menu).

      2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor and choose **OK**. If your vendor name does not appear, select *Custom* instead and specify the settings that are appropriate for your simulation or timing analysis tool. Go to MAX+PLUS II Help for detailed information on the options available in the **EDIF Netlist Writer Settings** dialog box.

      3. To generate an optional Standard Delay Format (SDF) Output File (**.sdo**), choose the **Customize** button to expand the dialog box to show all settings. Select one of the SDF Output File options

under *Write Delay Constructs To*, and choose **OK**.

The filenames of the EDIF Output File(s) and optional SDF Output File(s) are the same as the user-defined chip name(s) for the project; if no chip names exist, the Compiler assigns filenames that are based on the project name. For a multi-device project, the Compiler also generates a top-level EDIF Output File that is uniquely identified by "**_t**" appended to the project name. In addition, the Compiler automatically generates a VHDL Memory Model Output File, *<project name>***.vmo**, when it generates an EDIF Output File that contains memory (RAM or ROM).

3. If you wish to generate VHDL Output Files (**.vho**), turn on the **VHDL Netlist Writer** command (Interfaces menu). Then choose **VHDL Netlist Writer Settings** command (Interfaces menu). Select *VHDL Output File (.vho)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF ver. 2.1 files contain timing delay information that allows you to perform back-annotation simulation in VHDL with VITAL-compliant simulation libraries. The VHDL Output Files generated by the Compiler have the extension **.vho**, but are otherwise named in the same way as the EDIF Output Files described above.

4. If you wish to generate Verilog HDL Output Files (**.vo**), turn on the **Verilog Netlist Writer** command (Interfaces menu). Then choose **Verilog Netlist Writer Settings** command (Interfaces menu). Select *Verilog Output File (.vo)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF Output Files contain timing delay information that allows you to perform back-annotation simulation in Verilog HDL. The Verilog Output Files generated by the Compiler have the extension **.vo**, but are otherwise named in the same way as the EDIF Output Files described above.

9. To run the MAX+PLUS II Compiler, choose the **Project Save & Compile** command (File menu) or choose the **Start** button in the Compiler window.

See step 3 for information on running MAX+PLUS II software from the command line.

10. Once you have compiled the project with the MAX+PLUS II Compiler, you can use the VHDL, Verilog HDL, or EDIF output file(s), and the optional SDF Output File(s) (**.sdo**) to perform timing analysis or timing simulation with another EDA tool. Specific instructions for some tools are described in these MAX+PLUS II ACCESS Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your EDA tool.

The MAX+PLUS II Compiler also generates a Report File (**.rpt**), a Pin-Out File (**.pin**), and one or more of the following files for device programming or configuration:

- JEDEC Files (**.jed**)
- Programmer Object Files (**.pof**)
- SRAM Object Files (**.sof**)
- Hexadecimal (Intel-format) Files (**.hex**)
- Tabular Text Files (**.ttf**)

Related Topics:

- Refer to the following sources for additional information:
  - Go to *Compiler Procedures* in MAX+PLUS II Help for information on other available Compiler settings.
  - Go to *Programmer Procedures* in MAX+PLUS II Help for instructions on creating other types of programming files and on programming or configuring Altera devices.
  - Go to Back-Annotating MAX+PLUS II Pin Assignments to Design Architect Symbols for information on back-annotating pin assignments in Mentor Graphics Design Architect schematics.
  - Go to Programming Altera Devices for information on the different programming hardware options for Altera device families.
- Go to the following topics, which are available on the web, for additional information:

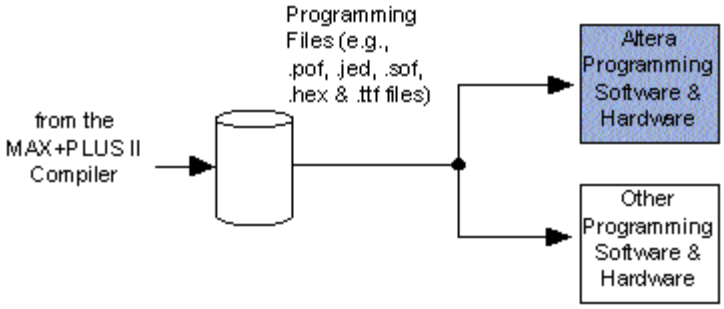
MAX+PLUS II Development Software
- Altera Programming Hardware

Programming Altera Devices

Once you have successfully compiled and simulated a project with the MAX+PLUS® II software, you can program an Altera® device and test it in the target circuit. Figure 1 shows the device programming flow for MAX+PLUS II software.

Figure 1. MAX+PLUS II Device Programming Flow

Altera-provided items are shown in blue.



You can program devices with Altera programming hardware and MAX+PLUS II Programmer software installed on a 486- or Pentium-based PC or a UNIX workstation, or with programming hardware and software available from other manufacturers. Table 1 shows the available Altera programming hardware options on PCs and UNIX workstations.

Table 1. Altera Programming Hardware

Programming Hardware Option	UNIX PCs	Work-stations	MAX® 3000A Devices	Classic® & MAX 5000 Devices	MAX 7000 & MAX 7000E Devices	MAX 7000A, MAX 7000AE, MAX 7000B, MAX 7000S MAX 9000 & MAX 9000A Devices	FLEX® 6000, FLEX 6000A, FLEX 8000, FLEX 10K, FLEX 10KA, FLEX 10KB, & FLEX 10KE Devices	In-System Programming/ Configuration
Logic Programmer card, PL-MPU Master Programming Unit, and device-specific adapters	✓		✓	✓	✓	✓		

BitBlaster™ Download Cable	✓	✓	✓		✓	✓	✓
ByteBlasterMV™ Download Cable	✓		✓		✓	✓	✓
MasterBlaster™ Download Cable	✓	✓	✓		✓	✓	✓

If you wish to transfer programming files from a UNIX workstation to a PC over a network with File Transfer Protocol (FTP) or other similar transfer programs, be sure to select binary transfer mode.

Programming hardware from other manufacturers varies, but typically consists of a device connected to one of the serial ports on the workstation. Various vendors, such as Data I/O and BP Microsystems, supply hardware and software for programming Altera devices.

Related Topics:

- Go to Compiling Projects with MAX+PLUS II Software for information on creating programming files.
- Go to the following topics, which are available on the web, for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware
  - FLEX Devices
  - MAX Devices
  - Classic Device Family

Performing a Timing Simulation with QuickHDL Software

After you have entered a VHDL or Verilog HDL design file and compiled it with the MAX+PLUS ® II Compiler, you can use Mentor Graphics QuickHDL software to simulate the MAX+PLUS IIgenerated VHDL Output File (**.vhd**) or Verilog Output File (**.vo**) and the Standard Delay Format (SDF) Output File (**.sdo**).

To simulate your VHDL or Verilog HDL design, go through the following steps:

1. Be sure to set up the working environment correctly, as described in Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment.
2. Generate a VHDL or Verilog HDL output file and an SDF output file for your project, as described in Compiling Projects with MAX+PLUS II Software.
3. Change to your project's directory.
4. Copy your **quickhdl.ini** file to the same directory as your VHDL or Verilog HDL file.
5. Type the following sets of commands at the UNIX prompt to create the work library and compile your project's VHDL or Verilog HDL output file:

VHDL:	**Verilog HDL:**
`setenv MGC_WD 'pwd'` ↵	`setenv MGC_WD 'pwd'` ↵
`qhlib work` ↵	`qhlib work` ↵
`qvhcom` <*project name*>`.vho` ↵	`qvlcom` <*project name*>`.vo` ↵

6. Type `qhsim -sdftyp` <*project name*>`.sdo` ↵ at the UNIX prompt to perform timing back-annotation and simulation and to display the QuickHDL simulation window.

If your Verilog HDL design uses memory functions (RAM or ROM) that can be initialized with a hexadecimal file (Intel-format) initialization, you must convert these files into Verilog HDL format using the Programming Language Interface (PLI). To use the Altera-provided source code for PLI, perform the following steps:

1. Download the file **http://www.edif.org/lpmweb/convert_hex2ver.c** to your project directory.
2. Copy the following two files from the **$MGC_HOME/shared/pkgs/quickhdl/include** directory into the **/usr/maxplus2** directory:
   - **$MGC_HOME/shared/pkgs/quickhdl/include/veriuser**
   - **$MGC_HOME/shared/pkgs/quickhdl/include/acc_user**

Refer to the ***Mentor Graphics QuickHDL User's Reference Manual***, version 8.5-4.6i, for information on compiling the PLI application on different platforms and using the Verilog HDL PLI.

Related Links:

- Go to Performing a Functional Simulation with QuickHDL Software in these MAX+PLUS II ACCESS^{SM} Key topics for related information.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Performing a Timing Simulation with DVE & QuickSim II Software

After you have compiled a design with the MAX+PLUS® II Compiler, you can prepare the MAX+PLUS IIgenerated EDIF Output File (**.edo**) with Mentor Graphics Design Viewpoint Editor (DVE) and simulate it with the Mentor Graphics QuickSim II software.

To simulate an EDIF Output File with the QuickSim II software, follow these steps:

1. Be sure to set up the working environment correctly, as described in Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment.
2. Generate an EDIF Output File for your project, as described in Compiling Projects with MAX+PLUS II Software or Using the Altera Schematic Express (**sch_exprss**) Utility.
3. If you used the Altera Schematic Express (**sch_exprss**) utility to process your design, skip to step 5. Otherwise, go to step 4.
4. In the Navigator window, select your project's icon, press Button 3, and choose **Open max2_enr** to read your project's EDIF Output File with the ENRead utility. You can also start ENRead software by typing `max2_enr` ↵ at the UNIX prompt.
5. Select your project's folder, press Button 3, and choose **Open max2_ave** to open DVE, which will prepare your project's simulation component for QuickSim II timing simulation. DVE automatically generates an appropriately named viewpoint for your project. You can also start DVE by typing `max2_ave` ↵ at the UNIX prompt.
6. Select your project's folder, press Button 3, and choose **Open max2_qsim** to simulate your project and its DVE viewpoint with QuickSim II software. You can also start QuickSim II by typing `max2_qsim` ↵ at the UNIX prompt.
7. In the **Altera QuickSim** dialog box, type the name of your project's viewpoint in the *Viewpoint Name* box. Select *Timing* as the *Timing Mode*. Select the *Max timing* option. Choose *Scale Factor* for *Delay Scale*, and be sure that `0.1` is specified for the *Value*. Choose **OK**.

   ☞ If the delay scale value is not set to 0.1 (i.e., divided by ten), the QuickSim II software will not reflect the correct timing simulation values.

Related Topics:

- Performing a Timing Analysis with QuickPath Software in these MAX+PLUS II ACCESS℠ Key topics for related information.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Performing a Timing Analysis with QuickPath Software

After you have compiled your project with the MAX+PLUS ® II Compiler and generated an EDIF Output File (**.edo**), you can use Mentor Graphics QuickPath software to perform a timing analysis of your project.

To perform a timing analysis with QuickPath software, follow these steps:

1. Be sure to set up the working environment correctly, as described in Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment.
2. Generate an EDIF Output File for your project using one of the following methods:
    - Compiling Projects with MAX+PLUS II Software
    - Using the Altera Schematic Express (**sch_exprss**) Utility
    - Using the Altera VHDL Express (**vhd_exprss**) Utility
3. Select your project's folder from the **ALTERA** directory, press Button 3, and choose **Open max2_qpath** to start the QuickPath software. You can also start the QuickPath software by typing `max2_qpath` ↵ at the UNIX prompt.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Instantiating RAM & ROM Functions in Viewlogic Powerview Designs

The MAX+PLUS®II /Viewlogic Powerview interface offers full support for the memory capabilities of the FLEX® 10K device family, including synchronous and asynchronous RAM and ROM, cycle-shared dual-port RAM, dual-port RAM, single-Clock FIFO, and dual-Clock FIFO functions. You can use the Altera-provided **genmem** utility to generate functional simulation models and timing models for these functions. Type `genmem` ↵ at the UNIX prompt to display information on how to use this utility, as well as a list of the functions you can generate. RAM and ROM can be instantiated in both ViewDraw schematics and VHDL designs.

☞ Refer to Viewlogic documentation for information on simulating projects that contain RAM functions. The procedure for reading an EDIF Output File and preparing it for simulation with ViewSim requires additional steps when the project contains RAM functions.

When you instantiate a RAM or ROM function, follow these general guidelines:

- For ROM functions, you must specify an initial memory content file in the Intel hexadecimal format (**.hex**) or the Altera® Memory Initialization File (**.mif**) format. The filename must be the same as the instance name; e.g., the instance name must be unique throughout the whole project, and must contain only valid name characters. The initialization file must reside in the directory containing the project's design files.
- For RAM functions, specifying a memory initialization file is optional.
- For VHDL designs, specify the name of the initial memory content file in the Generic Map Clause of the instance, with the specified type `LPM_FILE`. If you do not use an initial memory content file (e.g., for a RAM function), you should not declare or use the Generic Clause.
- Do not synthesize the **genmem**-generated VHDL file: it is intended for simulation only.

☞ The MIF format is supported only for specifying initial memory content when compiling designs within the MAX+PLUS II software. You cannot use a MIF to perform simulation with Viewlogic tools prior to MAX+PLUS II compilation.

To instantiate RAM or ROM in a ViewDraw schematic, follow these steps:

1. Use the **genmem** utility to generate a memory model by typing the following command at the UNIX prompt:

   `genmem` <*memory type*> <*memory size*> `-vwlogic` ↵

   For example: `genmem asynrom 256x15 -vwlogic` ↵

2. Start the VHDL-to-symbol utility, **vhdl2sym**, by double-clicking Button 1 on the **max2_vhdl2sym** icon in the Altera® Toolbox Design Tools Drawer.

3. Specify the following options in the **vhdl2sym** dialog box and choose **OK** to create a symbol. For example, to create the symbol for a 256x15 asynchronous ROM, enter the following settings:

Option:	**Setting:**
VHDL Source Filename	*asyn_rom_256x15.vhd*

Add LEVEL attribute *On*

4. Choose **Comp** (Add menu), type <*design name*> in the *Enter Name* box, and choose **OK**.

To instantiate a RAM or ROM function in VHDL, follow these steps:

1. Repeat step 1 above.
2. Create a VHDL design that incorporates the text from the **genmem**-generated Component Declaration, <*memory name*>**.cmp**, and instantiate the <*memory name*> function.

Figure 1 shows a VHDL design that instantiates asyn_rom_256x15.vhd, a 256 x 15 ROM function.

Figure 1. VHDL Design File with ROM Instantiation (tstrom.vhd)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;



ENTITY tstrom IS
        PORT (
          addr    : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
          memenab : IN STD_LOGIC;
          q       : OUT STD_LOGIC_VECTOR (14 DOWNTO 0));
END tstrom;



ARCHITECTURE behavior OF tstrom IS

COMPONENT asyn_rom_256x15
     GENERIC (LPM_FILE : string);



PORT (Address : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
           MemEnab : IN STD_LOGIC;
           Q       : OUT STD_LOGIC_VECTOR(14 DOWNTO 0)
     );
END COMPONENT;



BEGIN

   u1: asyn_rom_256x15
        GENERIC MAP (LPM_FILE => "u1.hex")
   PORT MAP (Address => addr, MemEnab => memenab, Q =>q);
END behavior;
```

Support

Intel Community Forums provides a place to ask and answer questions about Intel products.

Intel does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Using Cadence RapidSIM & MAX+PLUS II Software

The following topics describe how to use the Cadence RapidSIM software with MAX+PLUS® II software. Click on one of the following topics for information:

This file is suitable for printing only. It does not contain hypertext links that allow you to jump from topic to topic.

Setting Up the MAX+PLUS II/Cadence Working Environment

- Software Requirements
- MAX+PLUS II Directory Structure
- MAX+PLUS II/Cadence Interface File Organization
- Concept & RapidSIM Local Work Area Directory Structure

Simulation

- Project Simulation Flow
- Performing a Timing Simulation with RapidSIM Software

Related Topics:

Go to the following MAX+PLUS II ACCESS℠ Key topics for related information:

- Compiling Projects with MAX+PLUS II Software
- Programming Altera Devices

Go to the following topics, which are available on the web, for additional information:

- MAX+PLUS II Development Software
- Altera Programming Hardware
- Cadence web site (http://www.cadence.com)

Setting Up the MAX+PLUS II/Cadence Working Environment

To use MAX+PLUS® II software with Cadence software, you must first install the MAX+PLUS II software, then establish an environment that facilitates entering and processing designs. The MAX+PLUS II/Cadence interface is installed automatically when you install the MAX+PLUS II software on your computer. Go to *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual for more information on installation and details on the directories that are created during MAX+PLUS II installation. Go to MAX+PLUS II/Cadence Interface File Organization for information about the MAX+PLUS II/Cadence directories that are created during MAX+PLUS II installation.

☞ The information presented here assumes that you are using the C shell and that your MAX+PLUS II system directory is **/usr/maxplus2**. If not, you must use the appropriate syntax and procedures to set environment

variables for your shell.

To set up your working environment for the MAX+PLUS II/Cadence interface, follow these steps:

1. Ensure that you have correctly installed the MAX+PLUS II and Cadence software versions described in the MAX+PLUS II/Cadence Software Requirements.

2. Add the following environment variables to your **.cshrc** file:

   ```
   setenv ALT_HOME /usr/maxplus2 ↵

   setenv CDS_INST_DIR <Cadence system directory path> ↵
   ```

3. Add the **$ALT_HOME/cadence/bin** and **$CDS_INST_DIR/tools/bin** directories to the `PATH` environment variable in your **.cshrc** file. Make sure these paths are placed before the Cadence hierarchy path.

4. Add **/usr/dt/lib** and **/usr/ucb/lib** to the `LD_LIBRARY_PATH` environment variable in your **.cshrc** file.

5. Create a new **cds.lib** file in your working directory or edit an existing one so that it includes all of the following lines that apply to the Cadence tools you have installed:

   ```
   DEFINE alt_syn ${ALT_HOME}/simlib/concept/alt_syn

   DEFINE lpm_syn ${ALT_HOME}/simlib/concept/lpm_syn

   DEFINE alt_lpm ${ALT_HOME}/simlib/concept/alt_lpm

   DEFINE alt_mf ${ALT_HOME}/simlib/concept/alt_mf

   DEFINE alt_max2 ${ALT_HOME}/simlib/concept/alt_max2

   DEFINE alt_max2 ${ALT_HOME}/simlib/composer/alt_max2/alt_max2

   DEFINE alt_vtl $ALT_HOME/simlib/concept/alt_vtl/lib

   DEFINE altera $ALT_HOME/simlib/concept/alt_mf/lib

   SOFTINCLUDE $CDS_INST_DIR/tools/leapfrog/files/cds.lib

   DEFINE <design name>.
   ```

6. Copy the **/usr/maxplus2/maxplus2.ini** file to your **$HOME** directory:

   ```
   cp /usr/maxplus2/maxplus2.ini $HOME ↵

   chmod u+w $HOME/maxplus2.ini ↵
   ```

The **maxplus2.ini** file contains both Altera- and user-specified initialization parameters that control the MAX+PLUS II software, such as MAX+PLUS II symbol and logic function library paths and the current project name. The MAX+PLUS II installation procedure creates and copies the **maxplus2.ini** file to the **/usr/maxplus2** directory.

Normally, you do not have to edit your local copy of **maxplus2.ini** because the MAX+PLUS II software updates the file automatically whenever you change any parameters or settings. However, if you move the **max2lib** and **max2inc** library subdirectories, you must update the file. Go to "Creating & Using a Local Copy of the **maxplus2.ini** File" in MAX+PLUS II Help for more information.

7. If you are using Concept on a Sun SPARCstation running SunOS, go to Setting Up the MAX+PLUS II/Cadence Concept Work Environment for a Sun SPARCstation Running SunOS Software to install the **redifnet** EDIF netlist reader utility.

8. If you are using Synergy software, edit the **hdl.var** file located in your working directory to include the following line:

   `DEFINE work` <*design name*>↵

9. Set up an appropriate directory structure for the tool(s) you are using. See the following topics for information:

   - Composer Project File Directory Structure
   - Concept & RapidSIM Local Work Area Directory Structure

Related Topics:

Go to the following topics, which are available on the web, for additional information:

- ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
- This manual is also available in 4 parts:
  - Preface & Section 1: MAX+PLUS II Installation
  - Section 2: MAX+PLUS II - A Perspective
  - Section 3: MAX+PLUS II Tutorial
  - Appendices, Glossary & Index

MAX+PLUS II Directory Structure

In the MAX+PLUS® II software, a project name is the name of a top-level design file, without the filename extension. This design file can be an EDIF, Verilog HDL, or VHDL netlist file; an AHDL Text Design File (TDF); or any other MAX+PLUS II-supported design file. The EDIF netlist file must be created by the **altout** or **concept2alt** utility and imported into the MAX+PLUS II software as an EDIF Input File (**.edf**).

Project design files and output files are stored in the project directory, with the exception of standard library functions provided by Altera or another EDA tool vendor. The MAX+PLUS II software stores the connectivity data on the links between design files in a hierarchical project in a Hierarchy Interconnect File (**.hif**), but refers to the entire project only by its project name. The MAX+PLUS II Compiler uses the HIF to build a single, fully flattened project database that integrates all design files in a project hierarchy.

MAX+PLUS II/Cadence Interface File Organization

Table 1 shows the MAX+PLUS® II/Cadence interface subdirectories that are created in the MAX+PLUS II system directory (by default, the **/usr/maxplus2** directory) during MAX+PLUS II installation. For information on the other directories that are created during MAX+PLUS II installation, see "MAX+PLUS II File Organization" in *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual.

Table 1. MAX+PLUS II Directory Organization

Directory	Description
./lmf	Contains the Altera-provided Library Mapping File, **cadence.lmf**, that maps Cadence logic functions to equivalent MAX+PLUS II logic

	functions.
./examples/cadence	Contains the sample files for Cadence software discussed in these ACCESS^{SM} Key Guidelines.
./cadence	Contains the AMPLE userware for the MAX+PLUS II/Cadence interface.
./simlib/concept/alt_max2	Contains the MAX+PLUS II primitives, including `CARRY`, `CASCADE`, `EXP`, `GLOBAL`, `LCELL`, `SOFT`, `OPNDRN`, `DFFE` (D flipflop with Clock Enable), and `DFFE6K` (D flipflop with Clock Enable and both Clear and Preset for FLEX® 6000 devices only) for use with Concept software.
./simlib/composer/alt_max2	Contains the MAX+PLUS II primitives, including `CARRY`, `CASCADE`, `EXP`, `GLOBAL`, `LCELL`, `SOFT`, `OPNDRN`, `DFFE` (D flipflop with Clock Enable), and `DFFE6K` (D flipflop with Clock Enable and both Clear and Preset for FLEX 6000 devices only) for use with Composer software.
./simlib/concept/alt_lpm	Contains the MAX+PLUS II megafunctions, including library of parameterized modules (LPM) functions, for use with Concept software.
./simlib/concept/max2sim	Contains the MAX+PLUS II/Concept simulation model library, **max2_sim**, for use with RapidSIM software.
./simlib/concept/alt_syn	Contains the MAX+PLUS II synthesis library, **alt_syn**, for use with Synergy and Concept software, and the **vlog2alt** utility.
./simlib/composer/alt_syn	Contains the MAX+PLUS II synthesis library, **alt_syn**, for use with Synergy and Composer software.
./simlib/concept/lpm_syn	Contains the Cadence LPM library, **lpm_syn**, for use with Synergy and Concept software.
./simlib/composer/lpm_syn	Contains the Cadence LPM library, **lpm_syn**, for use with Synergy and Composer software.
./simlib/concept/alt_mf	Contains the MAX+PLUS II VHDL logic function library. (`a_8count` is for the MAX® 7000 and MAX 9000 device families only.)
./simlib/concept/edifnet/templates	Contains template files for Concept directives, i.e., **global.cmd**, **compiler.cmd**, **vloglink.cmd**, **verilog.cmd**, and **master.local**.
./simlib/concept/alt_max2/verilogUdps	Contains Verilog HDL modules that are the equivalent of the primitives contained in **alt_max2** library for use with Concept software.
./simlib/composer/alt_max2/verilogUdps	Contains Verilog HDL modules that are the equivalent of the primitives contained in **alt_max2** library for use with Composer software.
./simlib/concept/alt_vtl **./simlib/composer/alt_vtl**	Contains VITAL library source files for use with Concept or Composer software.
./simlib/composer/alt_max2/verilog	Contains simulation modules for all symbols in the **alt_max2** Composer library.

Related Topics:

Go to the following topics, which are available on the web, for additional information:

- ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
- This manual is also available in 4 parts:
  - Preface & Section 1: MAX+PLUS II Installation
  - Section 2: MAX+PLUS II - A Perspective
  - Section 3: MAX+PLUS II Tutorial
  - Appendices, Glossary & Index
- FLEX Devices
- MAX Devices
- Classic Device Family

Concept & RapidSIM Local Work Area Directory Structure

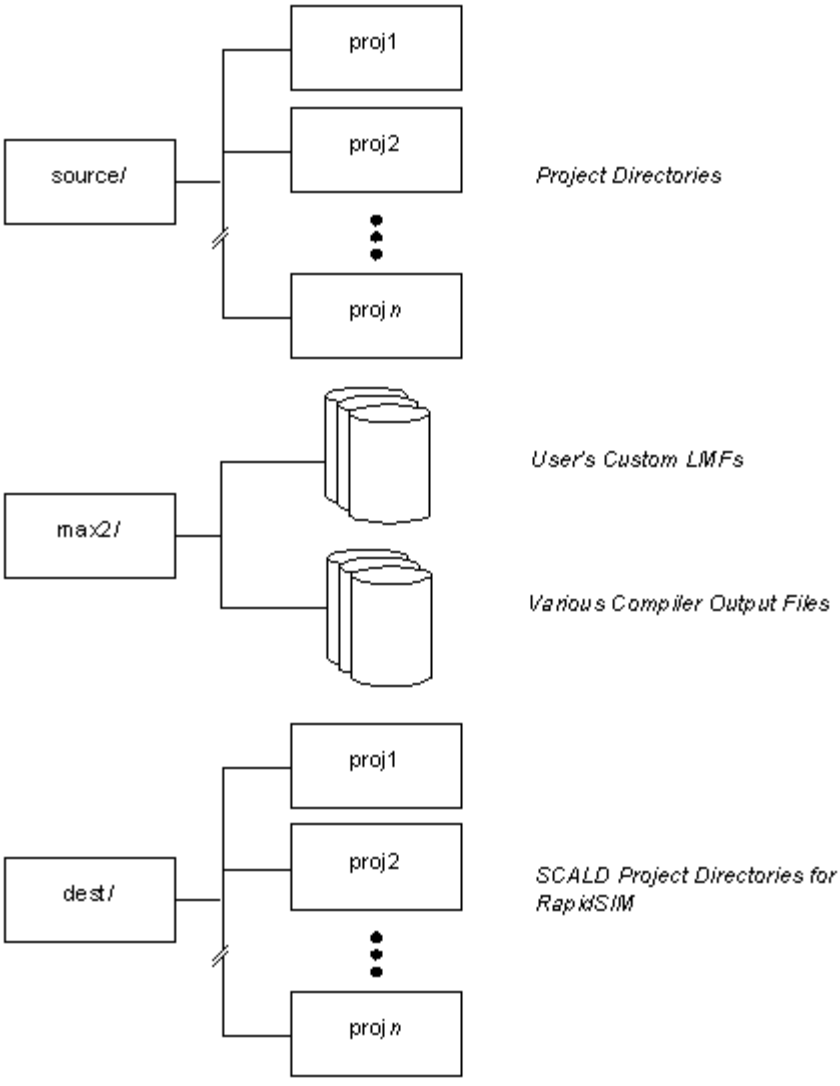
When the **redifnet** utility imports an EDIF netlist file for the RapidSIM software, it creates a SCALD directory for your project. However, creating this directory may overwrite the directory that was created for the original Concept schematic. To prevent overwriting this directory, you should create a file structure that helps you manage your design files.

Altera recommends that you create the following three directories for your design files.

Directory:	Description:
./source	Create Concept schematics and generate EDIF netlist files with the **wedifnet** utility in the **source** directory.
./max2	Copy the EDIF Input File (**.edf**) from the **source** directory to this directory to compile the file with the MAX+PLUS® II software.
./dest	Copy the EDIF Output File (**.edo**) from the **max2** directory to this directory to run the **redifnet** and RapidSIM software.

Copies of the appropriate directives files for Cadence tools must be present in both the **source** and **dest** directories. Figure 1 shows Altera's recommended file structure.

Figure 1. Recommended File Structure

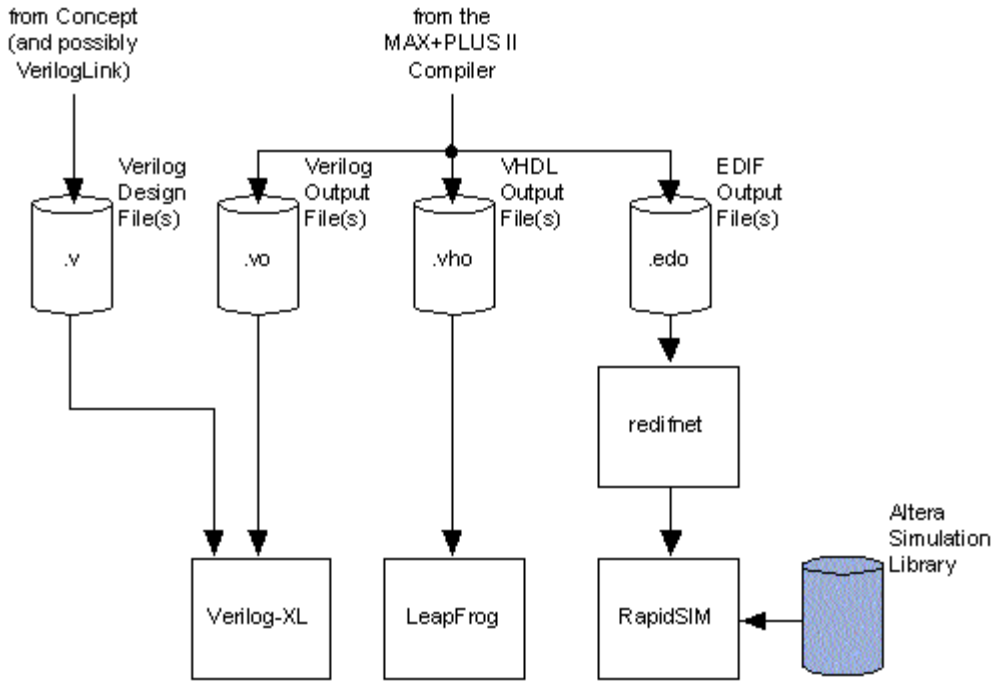


Project Simulation Flow

Figure 1 shows the project simulation flow for the MAX+PLUS® II/Cadence interface.

Figure 1. MAX+PLUS II/Cadence Project Simulation Flow

Altera-provided items are shown in blue.



Performing a Timing Simulation with RapidSIM Software

You can use the Cadence **redifnet** utility to read MAX+PLUS® II-generated EDIF Output Files and prepare them for timing simulation with RapidSIM software. RapidSIM software can simulate both the functionality and the timing of your design. It also checks setup time requirements, hold time requirements, and Clock duty cycle timing requirements on registers.

To simulate projects with RapidSIM software, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment.
2. Generate an EDIF Output File (**.edo**), as described in Compiling Projects with MAX+PLUS II Software.
3. Copy the EDIF Output File <*file name*>**.edo** from the /<*working directory*>/**max2** directory to the /<*working directory*>/**dest** directory.
4. Convert the EDIF Output File into the SCALD project format by typing `redifnet` <*design name*> ↵ at the UNIX prompt from the /<*working directory*>/**dest** directory.
5. Type `lwb_rapidsim` ↵ at the UNIX prompt to generate the **global.cmd** directive file.

6. Choose the **RapidSIM** button from the Logic Workbench window to start RapidSIM and simulate your EDIF Output File.

Compiling Projects with MAX+PLUS II Software

The MAX+PLUS® II Compiler can process design files in a variety of formats. This topic describes how to use MAX+PLUS II software to compile projects in which the top-level design file is an EDIF Input File (with the extension **.edf**).

Related Topics:

- Go to MAX+PLUS II Help for information on compiling VHDL and Verilog HDL, design files directly with the MAX+PLUS II Compiler.
- Go to Running Synopsys Compilers from MAX+PLUS II Software for information on running the Synopsys Design Compiler or FPGA Compiler software on a VHDL or Verilog HDL design from within the MAX+PLUS II Compiler window.

To compile a design (also called a "project") with MAX+PLUS II software, go through the following steps:

1. Create design files that are compatible with the MAX+PLUS II software and convert them into EDIF Input Files with the extension **.edf**. Specific instructions for some tools are described in these MAX+PLUS II ACCESS℠ Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your design entry or synthesis and optimization tool.

2. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, create a mapping for each function in a Library Mapping File (**.lmf**) to map the custom symbol to the corresponding EDIF Input File, AHDL Text Design File (**.tdf**), or other MAX+PLUS II-supported design file. These custom functions are represented in design files as hollow-body symbols or "black box" HDL descriptions.

   ☞ Go to "Library Mapping Files (**.lmf**)" in MAX+PLUS II Help for more information.

3. Open MAX+PLUS II and specify the name of your top-level design file as the project name with the **Project Name** command (File menu). If you open an HDL file in the MAX+PLUS II Text Editor, you can choose the **Project Set Project to Current File** command (File menu) instead.

   ☞ You can also compile a project from a command line. However, the first time you compile a project, the settings you need to specify are easier to specify from within the MAX+PLUS II software. After you have run the graphical user interface for the MAX+PLUS II software at least once, you can more easily use the command-line **setacf** utility to modify options in the Assignment & Configuration File (**.acf**) for the project. Type `setacf -h` ↵ and `maxplus2 -h` ↵ for descriptions of **setacf** and MAX+PLUS II command-line syntax.

4. Choose **Device** (Assign menu) and select the target Altera device family in the *Device Family* drop-down list box. If you wish to implement the design logic in a specific device, select it in the *Devices* box. Otherwise, select *AUTO* to allow the MAX+PLUS II Compiler to choose the best device(s) in the current device family. If your design entry or synthesis and optimization tool required you to specify a target family and/or device, specify the same information in this dialog box. For information on partitioning logic among multiple devices, go to MAX+PLUS II Help. Choose **OK**.

5. Open the Compiler window by choosing the **Compiler** command (MAX+PLUS II menu). Go through the following steps to specify the options necessary to compile the design file(s) in your project:

1. Ensure that all EDIF netlist files have the extension **.edf** and choose **EDIF Netlist Reader Settings** (Interfaces menu).

2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor. This name should be the name of the vendor whose tool(s) you used to create the EDIF netlist files. If your vendor name does not appear, select *Custom* instead.

   ☞ If you are compiling a design created with Synopsys FPGA Express software, select *Synopsys*, choose the **Customize** button, enter <*project name*>**.lmf** in the *LMF #1* box, choose **OK**, and skip to step 6.

3. If you selected an existing vendor name in the *Vendor* box and your project contains design files that require custom LMF mappings, choose the **Customize** button to expand the dialog box to show all settings. Turn on the *LMF #2* checkbox and type your custom LMF's filename in the corresponding text box, or select a name from the *Files* box. The selection in the *Vendor* box will change to *Custom* and all settings will be retained until you change them again.

4. If you selected *Custom* in the *Vendor* box, choose the **Customize** button to expand the dialog box to show all settings. Any previously defined custom settings will be displayed. Under *Signal Names*, type one or more names with up to 20 total name characters in the *VCC* or *GND* box if your EDIF Input File(s) use one or more names other than `VCC` or `GND` for the global high or low signals. Multiple signal names must be separated by either a comma (,) or a space. Under *Library Mapping Files*, turn on the *LMF #1* checkbox and type a filename in the text box following it, or select a name from the *Files* box. If necessary, specify another LMF name in the *LMF #2* box. Go to MAX+PLUS II Help for detailed information on the settings available in the **EDIF Netlist Reader Settings** dialog box.

5. Choose **OK**.

6. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, you may need to ensure that all files are present in your project directory, i.e., the same directory as the top-level design file. Otherwise, you must specify the directories containing these files as user libraries with the **User Libraries** command (Options menu).

7. Follow all guidelines that apply to your design entry or synthesis and optimization tool:

   - Exemplar Logic Galileo Extreme-Specific Compiler Settings
   - Synopsys DesignWare-Specific Compiler Settings
   - Converting Synopsys FPGA Compiler & Design Compiler Timing Constraints into MAX+PLUS II-Compatible Format with the **syn2acf** Utility
   - Synplicity Synplify-Specific Compiler Settings

8. If you wish to generate EDIF, VHDL, or Verilog HDL output files for post-compilation simulation or timing analysis with another EDA tool, go through the following steps:

   1. (Optional) Turn on the **Optimize Timing SNF** command (Processing menu) to reduce the size of the output file(s). Turning on this command can reduce the size of output netlists by up to 30%.

      ☞ This command does not create optimized timing SNFs on UNIX workstations. However, a non-optimized timing SNF provides the same functional and timing information as an optimized timing SNF.

   2. If you wish to generate EDIF Output Files (**.edo**), go through these steps:

      1. Turn on the **EDIF Netlist Writer** command (Interfaces menu). Then choose the **EDIF Netlist Writer Settings** command (Interfaces menu).

      2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that

vendor and choose **OK**. If your vendor name does not appear, select *Custom* instead and specify the settings that are appropriate for your simulation or timing analysis tool. Go to MAX+PLUS II Help for detailed information on the options available in the **EDIF Netlist Writer Settings** dialog box.

3. To generate an optional Standard Delay Format (SDF) Output File (**.sdo**), choose the **Customize** button to expand the dialog box to show all settings. Select one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**.

The filenames of the EDIF Output File(s) and optional SDF Output File(s) are the same as the user-defined chip name(s) for the project; if no chip names exist, the Compiler assigns filenames that are based on the project name. For a multi-device project, the Compiler also generates a top-level EDIF Output File that is uniquely identified by "**_t**" appended to the project name. In addition, the Compiler automatically generates a VHDL Memory Model Output File, *<project name>***.vmo**, when it generates an EDIF Output File that contains memory (RAM or ROM).

3. If you wish to generate VHDL Output Files (**.vho**), turn on the **VHDL Netlist Writer** command (Interfaces menu). Then choose **VHDL Netlist Writer Settings** command (Interfaces menu). Select *VHDL Output File (.vho)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF ver. 2.1 files contain timing delay information that allows you to perform back-annotation simulation in VHDL with VITAL-compliant simulation libraries. The VHDL Output Files generated by the Compiler have the extension **.vho**, but are otherwise named in the same way as the EDIF Output Files described above.

4. If you wish to generate Verilog HDL Output Files (**.vo**), turn on the **Verilog Netlist Writer** command (Interfaces menu). Then choose **Verilog Netlist Writer Settings** command (Interfaces menu). Select *Verilog Output File (.vo)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF Output Files contain timing delay information that allows you to perform back-annotation simulation in Verilog HDL. The Verilog Output Files generated by the Compiler have the extension **.vo**, but are otherwise named in the same way as the EDIF Output Files described above.

9. To run the MAX+PLUS II Compiler, choose the **Project Save & Compile** command (File menu) or choose the **Start** button in the Compiler window.

See step 3 for information on running MAX+PLUS II software from the command line.

10. Once you have compiled the project with the MAX+PLUS II Compiler, you can use the VHDL, Verilog HDL, or EDIF output file(s), and the optional SDF Output File(s) (**.sdo**) to perform timing analysis or timing simulation with another EDA tool. Specific instructions for some tools are described in these MAX+PLUS II ACCESS Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your EDA tool.

The MAX+PLUS II Compiler also generates a Report File (**.rpt**), a Pin-Out File (**.pin**), and one or more of the following files for device programming or configuration:

- JEDEC Files (**.jed**)
- Programmer Object Files (**.pof**)
- SRAM Object Files (**.sof**)
- Hexadecimal (Intel-format) Files (**.hex**)
- Tabular Text Files (**.ttf**)

Related Topics:

- Go to *Compiler Procedures* in MAX+PLUS II Help for information on other available Compiler settings.
- Go to *Programmer Procedures* in MAX+PLUS II Help for instructions on creating other types of programming files and on programming or configuring Altera devices.

Go to Back-Annotating MAX+PLUS II Pin Assignments to Design Architect Symbols for information on back-annotating pin assignments in Mentor Graphics Design Architect schematics.

- Go to Programming Altera Devices for information on the different programming hardware options for Altera device families.

Go to the following topics, which are available on the web, for additional information:

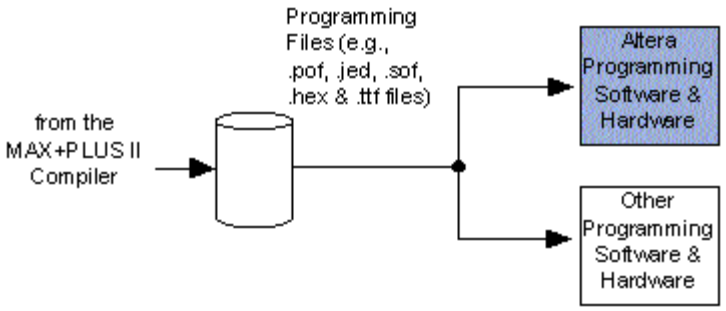
- MAX+PLUS II Development Software
- Altera Programming Hardware

Programming Altera Devices

Once you have successfully compiled and simulated a project with the MAX+PLUS® II software, you can program an Altera® device and test it in the target circuit. Figure 1 shows the device programming flow for MAX+PLUS II software.

Figure 1. MAX+PLUS II Device Programming Flow

Altera-provided items are shown in blue.



You can program devices with Altera programming hardware and MAX+PLUS II Programmer software installed on a 486- or Pentium-based PC or a UNIX workstation, or with programming hardware and software available from other manufacturers. Table 1 shows the available Altera programming hardware options on PCs and UNIX workstations.

Table 1. Altera Programming Hardware

Programming Hardware Option	UNIX PCs	Work-stations	MAX® 3000A Devices	Classic® & MAX 5000 Devices	MAX 7000 & MAX 7000E Devices	MAX 7000A, MAX 7000AE, MAX 7000B, MAX 7000S MAX 9000 & MAX 9000A Devices	FLEX® 6000, FLEX 6000A, FLEX 8000, FLEX 10K, FLEX 10KA, FLEX 10KB, & FLEX 10KE Devices	In-System Programming/ Configuration

Logic Programmer card, PL-MPU Master Programming Unit, and device-specific adapters	✓		✓	✓	✓	✓		
BitBlaster™ Download Cable	✓	✓	✓			✓	✓	✓
ByteBlasterMV™ Download Cable	✓		✓			✓	✓	✓
MasterBlaster™ Download Cable	✓	✓	✓			✓	✓	✓

If you wish to transfer programming files from a UNIX workstation to a PC over a network with File Transfer Protocol (FTP) or other similar transfer programs, be sure to select binary transfer mode.

Programming hardware from other manufacturers varies, but typically consists of a device connected to one of the serial ports on the workstation. Various vendors, such as Data I/O and BP Microsystems, supply hardware and software for programming Altera devices.

Related Topics:

- Go to Compiling Projects with MAX+PLUS II Software for information on creating programming files.

Go to the following topics, which are available on the web, for additional information:

- MAX+PLUS II Development Software
- Altera Programming Hardware
- FLEX Devices
- MAX Devices
- Classic Device Family

Last updated on December 6, 1999 for the MAX+PLUS II software version 9.4.

Performing a Timing Simulation with RapidSIM Software

You can use the Cadence **redifnet** utility to read MAX+PLUS® II-generated EDIF Output Files and prepare them for timing simulation with RapidSIM software. RapidSIM software can simulate both the functionality and the timing of your design. It also checks setup time requirements, hold time requirements, and Clock duty cycle timing requirements on registers.

To simulate projects with RapidSIM software, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment.
2. Generate an EDIF Output File (**.edo**), as described in Compiling Projects with MAX+PLUS II Software.
3. Copy the EDIF Output File <*file name*>**.edo** from the /<*working directory*>/**max2** directory to the /<*working directory*>/**dest** directory.
4. Convert the EDIF Output File into the SCALD project format by typing `redifnet` <*design name*> ↵ at the UNIX prompt from the /<*working directory*>/**dest** directory.
5. Type `lwb_rapidsim` ↵ at the UNIX prompt to generate the **global.cmd** directive file.
6. Choose the **RapidSIM** button from the Logic Workbench window to start RapidSIM and simulate your EDIF Output File.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Using Cadence RapidSIM & MAX+PLUS II Software

The following topics describe how to use the Cadence RapidSIM software with MAX+PLUS® II software. Choose one of the following topics for information:

Open a printable version of all topics listed on this page.

Setting Up the MAX+PLUS II/Cadence Working Environment

- Software Requirements
- MAX+PLUS II Directory Structure
- MAX+PLUS II/Cadence Interface File Organization
- Concept & RapidSIM Local Work Area Directory Structure

Simulation

- Project Simulation Flow
- Performing a Timing Simulation with RapidSIM Software

Related Links:

Go to the following MAX+PLUS II ACCESS℠ Key topics for related information:

- Compiling Projects with MAX+PLUS II Software
- Programming Altera Devices

Go to the following topics for additional information:

- MAX+PLUS II Development Software
- Altera Programming Hardware
- Cadence web site (http://www.cadence.com)

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Entering Resource Assignments

The MAX+PLUS® II software allows you to enter a variety of resource and device assignments for your projects. Resource assignments are used to assign logic functions to a particular pin, logic cell, I/O cell, embedded cell, row, column, Logic Array Block (LAB), Embedded Array Block (EAB), chip, clique, local routing, logic option, timing requirement, or connected pin group. In the MAX+PLUS II software, you can enter all types of resource and device assignments with Assign menu commands. You can also enter pin, logic cell, I/O cell, embedded cell, LAB, EAB, row, and column assignments in the MAX+PLUS II Floorplan Editor. The Assign menu commands and the Floorplan Editor all save assignment information in the ASCII Assignment & Configuration File (**.acf**) for the project.

In designs targeted for the Synopsys Design Compiler and FPGA Compiler software, you can assign a limited subset of these resource assignments by setting attributes in the VHDL or Verilog HDL design files with the `set_attribute` command. These attributes are incorporated into the EDIF netlist file(s). The MAX+PLUS II software automatically converts assignment information from the EDIF Input File (**.edf**) into the ACF format. For information on making MAX+PLUS II-compatible resource assignments with the `set_attribute` command, go to the following topics:

- Assigning Pins, Logic Cells, & Chips
- Assigning Cliques
- Assigning Logic Options

You can also modify the ACF for a design to contain timing requirements and other assignments, as described in the following topics:

- Modifying the Assignment & Configuration File with the **setacf** utility
- Converting Synopsys Timing Constraints into MAX+PLUS II-Compatible Format with the **syn2acf** Utility
- Converting Synopsys Hierarchical Timing Constraints into MAX+PLUS II-Compatible Format with the **gen_iacf** and **gen_hacf** Utilities

Related Topics:

- Refer to the following sources for related information:
  - Synopsys documentation for additional information on how to assign properties
  - "Resource Assignments in EDIF Input Files" and "Assigning Resources in a Third-Party Design Editor" in MAX+PLUS II Help for more information on assignments or properties that can be assigned in Synopsys
  - "resource assignments" or "ACF, format" in MAX+PLUS II Help using **Search for Help on** (Help menu), for information on entering assignments in the MAX+PLUS II software with Assign menu commands or in an ACF

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability

for use of or reliance on the solution.

MAX+PLUS II/Mentor Graphics Software Requirements

The following products are used to generate, process, synthesize, and verify a project with the MAX+PLUS ® II software and Mentor Graphics software:

Mentor Graphics		Exemplar	Altera
version C.1:			
System_1076 Compiler	QuickHDL		
QuickSim II	QuickHDL Pro		
Design Architect	QuickPath		
ENRead	LS_LIB library (optional)	LeonardoSpectrum version 2000.1b	MAX+PLUS II version 10.0
ENWrite	DVE		
GEN_LIB library			

The MAX+PLUS II **read.me** file provides up-to-date information on which versions of Mentor Graphics applications are supported by the current version of MAX+PLUS II. It also provides information on installation and operating requirements. You should read the **read.me** file on the CD-ROM before installing the MAX+PLUS II software. After installation, you can open the **read.me** file from the MAX+PLUS II Help menu.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Entering Resource Assignments

The MAX+PLUS® II software allows you to enter a variety of resource and device assignments for your projects. Resource assignments are used to assign logic functions to a particular pin, logic cell, I/O cell, embedded cell, row, column, Logic Array Block (LAB), Embedded Array Block (EAB), chip, clique, local routing, logic option, timing requirement, or connected pin group. In MAX+PLUS II software, you can enter all types of resource and device assignments with Assign menu commands. You can also enter pin, logic cell, I/O cell, embedded cell, LAB, EAB, row, and column assignments in the MAX+PLUS II Floorplan Editor. The Assign menu commands and the Floorplan Editor all save assignment information in the ASCII Assignment & Configuration File (**.acf**) for the project. In addition, you can edit ACFs manually in any standard text editor or with the **setacf** utility.

Concept & Composer Schematics

In both Concept and Composer schematics, you can assign a limited subset of these resource assignments by assigning properties to symbols. These properties are incorporated into the EDIF netlist file(s). The MAX+PLUS II software automatically converts assignment information from the EDIF Input File into the ACF format. For information on making MAX+PLUS II-compatible resource assignments, go to the following topics:

- Assigning Pins, Logic Cells & Chips
- Assigning Cliques
- Assigning Logic Options
- Modifying the Assignment & Configuration File with the **setacf** Utility

Go to the ***Cadence Concept Schematic User Guide*** and ***Composer Reference User Guide*** for details on how to assign properties. Go to "Resource Assignments in EDIF Input Files" and "Assigning Resources in a Third-Party Design Editor" in MAX+PLUS II Help for more information on assignments or properties that can be assigned in Concept and Composer.

Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample Concept and Composer schematic files, which include resource assignments:

- **/usr/maxplus2/examples/cadence/example6/fa2** (Concept)
- **/usr/maxplus2/examples/cadence/example7/fa2** (Composer)

VHDL & Verilog HDL Design Files

For Verilog HDL- and VHDL-based designs, you must use the MAX+PLUS II software or the **setacf** utility to enter resource assignments. For information on using the **setacf** utility, go to Modifying the Assignment & Configuration File with the **setacf** Utility.

Related Topics:

- For information on entering assignments in the MAX+PLUS II software with Assign menu commands or in an ACF, go to "resource assignments" or "ACF, format" in MAX+PLUS II Help using **Search for Help on** (Help menu).

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

The MAX+PLUS ® II software allows you to enter a variety of resource and device assignments for your projects. Resource assignments are used to assign logic functions to a particular pin, logic cell, I/O cell, embedded cell, row, column, Logic Array Block (LAB), Embedded Array Block (EAB), chip, clique, local routing, logic option, timing requirement, or connected pin group. In MAX+PLUS II software, you can enter all types of resource and device assignments with Assign menu commands. You can also enter pin, logic cell, I/O cell, embedded cell, LAB, EAB, row, and column assignments in the MAX+PLUS II Floorplan Editor. The Assign menu commands and the Floorplan Editor all save assignment information in the ASCII Assignment & Configuration File (**.acf**) for the project. In addition, you can edit ACFs manually in any standard text editor or with the **setacf** utility.

Design Architect Schematics

In Design Architect schematics, you can assign a limited subset of these resource assignments by assigning properties to symbols. These properties are incorporated into the EDIF netlist file(s). The MAX+PLUS II software automatically converts assignment information from the EDIF Input File into the ACF format. For information on making MAX+PLUS II-compatible resource assignments, go to the following topics:

- Assigning Pins, Logic Cells & Chips
- Assigning Cliques
- Assigning Logic Options
- Modifying the Assignment & Configuration File with the **setacf** Utility

☞ After you compile a project, you can back-annotate pin assignments, as described in BackAnnotating MAX+PLUS II Pin Assignments to Design Architect Symbols.

Installing the Alteraprovided MAX+PLUS II/Mentor Graphics/Exemplar Logic interface on your computer automatically creates the sample Design Architect schematic file **/usr/maxplus2/examples/mentor/example4/fa2**, which includes resource assignments.

VHDL & Verilog HDL Design Files

For Verilog HDL- and VHDL-based designs, you must use the MAX+PLUS II software or the **setacf** utility to enter resource assignments. Go to Modifying the Assignment & Configuration File with the **setacf** Utility for more information.

Related Topics:

- Go to "Resource Assignments in EDIF Input Files" and "Assigning Resources in a Third-Party Design Editor" in MAX+PLUS II Help for more information on assignments or properties that can be assigned in Design Architect software. For information on entering assignments in MAX+PLUS II software with Assign menu commands or in an ACF, go to "resource assignments" or "ACF, format" in MAX+PLUS II Help using **Search for Help on** (Help menu).

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

The MAX+PLUS® II software allows you to enter a variety of resource and device assignments for your projects. Resource assignments are used to assign logic functions to a particular pin, logic cell, I/O cell, embedded cell, row, column, Logic Array Block (LAB), Embedded Array Block (EAB), chip, clique, local routing, logic option, timing requirement, or connected pin group. In MAX+PLUS II software, you can enter all types of resource and device assignments with Assign menu commands. You can also enter pin, logic cell, I/O cell, embedded cell, LAB, EAB, row, and column assignments in the MAX+PLUS II Floorplan Editor. The Assign menu commands and the Floorplan Editor all save assignment information in the ASCII Assignment & Configuration File (**.acf**) for the project. In addition, you can edit ACFs manually in any standard text editor or with the **setacf** utility.

VHDL & Verilog HDL Design Files

When you use Synplicity Synplify software, you can assign a limited subset of these resource assignments by specifying attributes in the Synplify Design Constraints File (**.sdc**) or in the VHDL or Verilog HDL design files. The Synplify software automatically incorporates these attributes into the EDIF netlist file(s) generated from the HDL design files. MAX+PLUS II then automatically converts assignment information from the EDIF Input File into the ACF format. The following topics describe how to make MAX+PLUS II-compatible resource assignments before design processing with the Synplify software:

- Assigning Pins
- Assigning the Implement in EAB Logic Option
- Modifying the Assignment & Configuration File with the **setacf** Utility

Related Topics:

- Refer to the following sources for more information:
    - Go to the ***Synplify User's Guide*** for details on how to assign properties.
    - Go to "Resource Assignments in EDIF Input Files" and "Assigning Resources in a Third-Party Design Editor" in MAX+PLUS II Help for more information on assignments or properties that can be assigned when you use the Synplify software.
    - Go to "resource assignments" or "ACF, format" in MAX+PLUS II Help using **Search for Help on** (Help menu) for information on entering assignments in the MAX+PLUS II software with Assign menu commands or in an ACF.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Using the Altera Schematic Express (sch_exprss) Utility

Once you have created a Design Architect schematic, you can use the Altera Schematic Express utility (**sch_exprss**) to generate a Design Viewpoint Editor (DVE) viewpoint and an EDIF netlist file from the schematic; process the EDIF Input File (**.edf**) with the MAX+PLUS ® II software to generate an EDIF Output File (**.edo**); process the EDIF Output File with ENRead and DVE software; and generate an **altera_asim** viewpoint for simulation. The **sch_exprss** utility creates all necessary subdirectories and copies all of the files to the correct locations.

To use the **sch_exprss** utility, follow these steps:

1. Be sure to set up the working environment correctly, as described in Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment.
2. Create a Design Architect schematic that follows the guidelines described in Creating Design Architect Schematics for Use with MAX+PLUS II Software.
3. Select your project's folder, press Button 3, and choose **Open sch_exprss** from the Mentor Graphics Navigator window to start the Altera Schematic Express tool.
4. Specify settings for the *Input Schematic*, *Altera Device Family*, *MAX+PLUS II Synthesis Style*, *Process Direction*, and *Verbose* options in the **sch_exprss** dialog box and choose **OK** to generate the **altera_asim** file for simulation with QuickSim II software.
5. If necessary, correct any errors in the Design Architect schematic design file and recompile the project. The **sch_exprss** utility generates the **altera_asim** viewpoint in the appropriate directory.
6. Simulate your project, as described in Performing a Timing Simulation with DVE & QuickSim II Software.

Related Links:

- Go to Performing a Timing Analysis with QuickPath Software in these MAX+PLUS II ACCESS℠ Key topics for related information.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Creating Design Architect Schematics for Use with MAX+PLUSÂ II Software

You can create Design Architect schematics and convert them into EDIF Input Files (**.edf**) that can be processed with the MAX+PLUS Â® Â II Compiler.

To create a Design Architect schematic for use with MAX+PLUSÂ II software, go through the following steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUSÂ II/Mentor Graphics/Exemplar Logic Working Environment.
2. Start the MAX+PLUSÂ II/Mentor Graphics interface by typing `max2_dmgr` ↵ at a UNIX prompt.
3. Start the Design Architect software by double-clicking Button 1 on the **max2_da** icon in the Design Manager tools window. You can also start Design Architect software by typing `max2_da`Â ↵ at the UNIX prompt.
4. Use the graphical user interface to structure and organize your files to create an environment that facilitates entering and processing designs. Go to the following topics for more information:
   - Local Work Area Directory Structure
   - MAX+PLUSÂ II Project Directory Structure
   - Mentor Graphics Project Directory Structure
5. Choose the **OPEN SHEET** button in the Design Architect session_palette, then specify a name for your project in the *Component Name* box. Choose **OK**.
6. Enter logic functions from the following Altera Â® Âprovided libraries:
   - **ALTERA LPMLIB** includes library of parameterized modules (LPM) functions
   - **ALTERA GENLIB** includes primitives and macrofunctions
   - **LSTTL** includes 74-series macrofunctions

   ☞ You can instantiate MegaCore™ functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP SM). The OpenCore™ feature in the MAX+PLUSÂ II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.

   The following topics describe special steps needed to instantiate LPM and `clklock` functions:

   - Instantiating LPM Functions in Design Architect Schematics
   - Instantiating the clklock Megafunction in Design Architect Schematics
7. (Optional) To create a hierarchical design that contains symbols representing other design files, such as AHDL or VHDL design files, go to Creating Hierarchical Projects with Design Architect Software.
8. If you wish to make resource assignments in a Design Architect schematic, go to Entering Resource Assignments. You can also enter resource assignments from within the MAX+PLUSÂ II software.
9. Choose **Check Sheet for Altera** (Check menu) to save and check your design. If your design contains LPM functions , the Design Architect software will ask whether you want to compile the LPM model. Choose **YES** if you want to compile the VHDL code for the LPM functions. The software will automatically select the corresponding compiler: System 1076 for B.(*x*) releases and QuickHDL compilers for releases C.1 and later.
10. (Optional) If your schematic design includes models for VHDL or VerilogÂ HDL designs, perform a functional simulation with the QuickHDLÂ Pro software, as described in Performing a Functional Simulation with QuickHDL Pro Software. If it does not, you can perform a functional simulation with the QuickSim software, as described in Performing a Functional Simulation with DVE & QuickSimÂ II Software.
11. Once you have created a schematic, you can generate an EDIF netlist file that can be imported into the MAX+PLUSÂ II software with either of the following methods:
    - You can create an EDIF netlist file, as described in Converting Design Architect Schematics into

MAX+PLUSÂ II-Compatible EDIF Netlist Files with the ENWrite Utility.

- You can use the Altera Schematic Express utility, **sch_exprss**, to automatically create an EDIF netlist file, compile it with the MAX+PLUSÂ II Compiler, generate an EDIF Output File (**.edo**), and prepare the EDIF Output File for simulation with ENRead and Design Viewpoint Editor (DVE), as described in Using the Altera Schematic Express (**sch_exprss**) Utility.

Even if your design is a hierarchical design incorporating files created with multiple design entry methods, both the ENWrite and Altera Schematic Express utilities generate EDIF files for all files in the design.

Installing the AlteraÂprovided MAX+PLUSÂ II/Mentor Graphics/Exemplar Logic interface on your computer automatically creates the following sample Design Architect schematic files:

- **/usr/maxplus2/examples/mentor/example1/fulladd**
- **/usr/maxplus2/examples/mentor/example3/fulladd2**
- **/usr/maxplus2/examples/mentor/example7/fifo**

Related Links:

- Go to Compiling Projects with MAX+PLUSÂ II Software in these MAX+PLUSÂ II ACCESS^{SM} Key topics for related information.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Resynthesizing a Design Using the alt_vtl Library & a MAX+PLUS II SDF Output File

Altera provides the **alt_vtl.db** post-synthesis library for technology mapping or resynthesis. You can use this library with the MAX+PLUS® II -generated Standard Delay Format (SDF) Output File (**.sdo**) to retarget and resynthesize your design for another device family by performing the following steps:

To retarget and resynthsize a design, follow these steps:

1. Generate an EDIF Output File (**.edo**) and an SDF Output File (**.sdo**), as described in Compiling Projects with MAX+PLUS II Software.

2. Modify your **.synopsys_dc.setup** file to include the following lines:

```
search_path = {./usr/maxplus2/synopsys/library/alt_post/syn/lib
<target library path>}; ↵
target_library = {<target library path>}; ↵
symbol_library = {<target library symbol file>}; ↵
link_library = {alt_vtl.db}; ↵
```

3. In the Design Compiler or FPGA Compiler software, type the following commands to read in the EDIF and SDF output files:

```
read -f edif <design name>.edo ↵
read_timing -load_delay net <design name>.sdo ↵
```

4. Type the following commands to compile your design, report the timing information, and create an EDIF netlist file (**.edf**) that can be processed with the MAX+PLUS II Compiler.

```
compile ↵
report_timing ↵
write -f edif -hierarchy -o <design name>.edf ↵
```

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Setting Up the MAX+PLUS II/Synplicity Working Environment

To use MAX+PLUS® II software with Synplicity software, you must first install the MAX+PLUS II software, then establish an environment that facilitates entering and processing designs. The MAX+PLUS II/Synplicity interface is installed automatically when you install the MAX+PLUS II software on your computer. Ensure that you have correctly installed the MAX+PLUS II and Synplicity software versions described in the MAX+PLUS II/Synplicity Software Requirements.

You do not need to set any initialization or project variables before using Synplicity software with MAX+PLUS II software. Synplicity software features Direct Synthesis Technology that performs technology mapping directly to Altera® device logic cells by inserting architecture-specific primitives to implement features such as logic cells, parallel expanders, carry chains, and cascade chains.

Related Links:

- Go to *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual for more information on installation and details on the directories created during MAX+PLUS II installation.
- Go to MAX+PLUS II/Synplicity Interface File Organization in these MAX+PLUS II ACCESS℠ Key topics for information about the MAX+PLUS II/Synplicity directories that are created during MAX+PLUS II installation.
- Go to the following topics for additional information:
  - ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
  - This manual is also available in 4 parts:
    - Preface & Section 1: MAX+PLUS II Installation
    - Section 2: MAX+PLUS II - A Perspective
    - Section 3: MAX+PLUS II Tutorial
    - Appendices, Glossary & Index

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Setting Up the MAX+PLUS II/Viewlogic Powerview Working Environment

To use the MAX+PLUS ® II software with Mentor Graphics/Exemplar Logic software, you must install the MAX+PLUS II software, then establish an environment that facilitates entering and processing designs. The MAX+PLUS II/Mentor Graphics/Exemplar Logic interface is installed automatically when you install the MAX+PLUS II software on your computer.

Go to *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual for more information on installation and details on the directories that are created during MAX+PLUS II installation. Go to MAX+PLUS II/Mentor Graphics/Exemplar Logic Interface File Organization for information about the MAX+PLUS II/Mentor Graphics directories that are created during MAX+PLUS II installation.

The information presented here assumes that you are using a C shell and that your MAX+PLUS II system directory is **/usr/maxplus2**. If not, you must use the appropriate syntax and procedures to set environment variables for your shell.

To set up your working environment for the MAX+PLUS II/Mentor Graphics interface, follow these steps:

1. Ensure that you have correctly installed the MAX+PLUS II and Mentor Graphics software versions described in MAX+PLUS II/Mentor Graphics Software Requirements.

2. Add the following environment variables to your **.cshrc** file:

```
setenv ALT_HOME /usr/maxplus2 ↵
setenv MGC_WD <user-specified working directory> ↵
setenv MGC_HOME <Mentor Graphics system directory> ↵
setenv MAX2_MENTOR /usr/maxplus2/mentor/max2 ↵
setenv MGC_LOCATION_MAP <user-specified location_map file> ↵
setenv EXEMPLAR <Galileo or Leonardo system directory> ↵
```

   Installing the Altera® provided MAX+PLUS II/Mentor Graphics interface on your computer automatically installs a template for these environment variables in the **/usr/maxplus2/mentor/max2/.cshrc** file.

3. Add the **$MGC_HOME/bin**, **$MAX2_MENTOR/bin**, **$ALT_HOME/bin**, **$EXEMPLAR/bin/**<*os*>, and **$ALT_HOME/bin** directories to the `PATH` environment variable in your **.cshrc** file, where <*os*> is the operating system, e.g., `SUN4` for SunOS; `SUN5` for Solaris.

4. If you plan to use the Altera Schematic Express (**sch_exprss**) utility or the Altera VHDL Express (**vhd_exprss**) utility, add the following environment variable to your **.cshrc** file:

```
setenv MAX2_QSIM /usr/maxplus2/simlib/mentor/max2sim ↵
```

5. Type `source ~/.cshrc` at a UNIX prompt to source the **.cshrc** file and validate the settings in steps 1 through 4.

6. Add the following lines to your **MGC_location_map** file:

```
$MAX2_MENTOR ↵
/usr/maxplus2/mentor/max2 ↵
$MGC_GENLIB ↵
/<user-specified Mentor Graphics GEN_LIB directory> ↵
$MGC_LSLIB ↵
/<user-specified Mentor Graphics LS_LIB directory> ↵
$MAX2_EXAMPLES ↵
/<user-specified example directory> ↵
$MAX2_LMCLIB ↵
/<user-specified Logic Modeling directory> ↵
$MAX2_GENLIB ↵
/usr/maxplus2/simlib/mentor/alt_max2 ↵
$MAX2_QSIM ↵
/usr/maxplus2/simlib/mentor/max2sim ↵
$MAX2_FONT ↵
/usr/maxplus2/mentor/max2/fonts ↵
$MGC_SYS1076_STD ↵
/<user-specified MGC_HOME directory>/pkgs/sys_1076_std/ std ↵
$MGC_SYS1076_ARITHMETIC ↵
/<user-specified MGC_HOME directory>/pkgs/sys_1076_std/arithmetic ↵
$MGC_SYS1076_PORTABLE ↵
/<user-specified MGC_HOME directory>/pkgs/sys_1076_std/mgc_portable ↵
$MGC_SYS1076_IEEE ↵
/<user-specified MGC_HOME directory>/pkgs/sys_1076_std/ieee ↵
$MGC_SYS1076_SRC ↵
/<user-specified MGC_HOME directory>/pkgs/sys_1076_std/ src ↵
$MAX2_MFLIB ↵
/usr/maxplus2/simlib/mentor/alt_mf ↵
```

Installing the Alteraprovided MAX+PLUS II/Mentor Graphics/Exemplar Logic interface on your computer automatically installs a template for these environment variables in the **/usr/maxplus2/mentor/max2/location_map/location_map** file.

7. If you want to use QuickHDL software to simulate VHDL or Verilog HDL designs, add the following line in the `[library]` section of your **quickhdl.ini** file: `altera = $MAX2_MFLIB`.

8. If you plan to use QuickHDL software to simulate VITAL-compliant VHDL files, add the following lines to your **MGC_location_map** file:

```
$MAX2_VTLLIB ↵
/usr/maxplus2/simlib/mentor/alt_vtl ↵
```

9. Copy the **/usr/maxplus2/maxplus2.ini** file to your **$HOME** directory:

```
cp /usr/maxplus2/maxplus2.ini $HOME ↵
chmod u+w $HOME/maxplus2.ini ↵
```

The **maxplus2.ini** file contains both Altera- and user-specified initialization parameters that control

the MAX+PLUS II software, such as Alteraprovided logic and symbol library paths and the current project name. The MAX+PLUS II installation procedure creates and copies the **maxplus2.ini** file to the **/usr/maxplus2** directory.

Normally, you do not have to edit your local copy of **maxplus2.ini**, because the MAX+PLUS II software updates the file automatically whenever you change any parameters or settings. However, if you move the **max2lib** and **max2inc** library subdirectories, you must update the file. Go to "Creating & Using a Local Copy of the **maxplus2.ini** File" in MAX+PLUS II Help for more information.

Related Links:

- ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
- This manual is also available in 4 parts:
  - Preface & Section 1: MAX+PLUS II Installation
  - Section 2: MAX+PLUS II - A Perspective
  - Section 3: MAX+PLUS II Tutorial
  - Appendices, Glossary & Index

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

To use the MAX+PLUS® II software with Synopsys software, you must first install the MAX+PLUS II software, then establish an environment that facilitates entering and processing designs by modifying your Synopsys configuration files. The MAX+PLUS II/Synopsys interface is installed automatically when you install the MAX+PLUS II software on your workstation. Go to *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual for more information on installation and details on the directories that are created during MAX+PLUS II installation. Go to MAX+PLUS II/Synopsys Interface File Organization for information about the MAX+PLUS II/Synopsys directories that are created during MAX+PLUS II installation.

The information presented here assumes that you are using C shell and that your MAX+PLUS II system directory is **/usr/maxplus2**. If not, you must use the appropriate syntax and procedures to set environment variables for your shell.

To set up your working environment for the MAX+PLUS II/Synopsys interface, follow these steps:

1. Ensure that you have correctly installed the MAX+PLUS II and Synopsys software versions described in the MAX+PLUS II/Synopsys Software Requirements.

2. Add technology, synthetic, and link library settings to your **.synopsys_dc.setup** configuration file, as

described in Setting Up Design Compiler & FPGA Compiler Configuration Files.

☞ To use the DesignWare interface with FLEX® 6000, FLEX 8000, and FLEX 10K devices, follow the steps in Setting Up the DesignWare Interface.

3. Add simulation library settings to your **.synopsys_vss.setup** file, and analyze the libraries, as described in Setting Up VSS Configuration Files.

4. Add the **/usr/maxplus2/bin** directory to the `PATH` environment variable in your **.cshrc** file in order to run the MAX+PLUS II software.

`$ALT_HOME`**/synopsys/bin**

Related Links:

- Go to the following topics for additional information:
  - FLEX Devices
  - ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
  - This manual is also available in 4 parts:
    - Preface & Section 1: MAX+PLUS II Installation
    - Section 2: MAX+PLUS II - A Perspective
    - Section 3: MAX+PLUS II Tutorial
    - Appendices, Glossary & Index

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

To use the MAX+PLUS® II software with Viewlogic's Powerview software, you must install the MAX+PLUS II software, familiarize yourself with the Altera® Toolbox in the Powerview Cockpit, and then establish an environment that facilitates entering and processing designs. The MAX+PLUS II /Viewlogic Powerview interface is installed automatically when you install the MAX+PLUS II software on your workstation.

To set up your working environment for the MAX+PLUS II/Viewlogic Powerview interface, follow these steps:

1. Ensure that you have correctly installed the MAX+PLUS II and Viewlogic software versions described in MAX+PLUS II/Viewlogic Powerview Software Requirements.

2. Add the following environment variable to your **.cshrc** file to specify **/usr/maxplus2** as the MAX+PLUS II system directory:

   ```
   setenv ALT_HOME /usr/maxplus2 ↵
   ```

3. Add the **$ALT_HOME/viewlogic/standard**, **$ALT_HOME/bin**, and **$ALT_HOME/viewlogic/bin** directories to the PATH environment variable in your **.cshrc** file.

4. Add the **$ALT_HOME/viewlogic/standard** directory to the WDIR environment variable in your **.cshrc** file using the following syntax:

   ```
   setenv WDIR $ALT_HOME/viewlogic/standard:/<Powerview system directory>/standard ↵
   ```

   ☞ Make sure the **$ALT_HOME/viewlogic/standard** directory is the first directory in your WDIR path.

5. Source your **.cshrc** file by typing `source .cshrc ↵` at the UNIX prompt.

6. Create the Viewlogic Powerview **viewdraw.ini** configuration file.

7. Copy the **/usr/maxplus2/maxplus2.ini** file to your **$HOME** directory:

   ```
   cp /usr/maxplus2/maxplus2.ini $HOME ↵

   chmod u+w $HOME/maxplus2.ini ↵
   ```

   The **maxplus2.ini** file contains both Altera- and user-specified initialization parameters that control the MAX+PLUS II software, such as MAX+PLUS II symbol and logic function library paths and the current project name. The MAX+PLUS II installation procedure creates and copies the **maxplus2.ini** file to the **/usr/maxplus2** directory.

   ☞ Normally, you do not have to edit your local copy of **maxplus2.ini**, because the MAX+PLUS II software updates the file automatically whenever you change any parameters or settings. However, if you move the **max2lib** and **max2inc** library subdirectories, you must update the file. Go to "Creating & Using a Local Copy of the **maxplus2.ini** File" in MAX+PLUS II Help for more information.

8. If you plan to instantiate Library of Parameterized Modules (LPM) functions in ViewDraw schematics, you must create a new file with the name **vdraw.vs**. The **vdraw.vs** file must include the following line:

   ```
   load ("vdpath")
   ```

   You must also make sure that you specify the **vdraw.vs** file in your WDIR path.

9. Set up a directory structure that facilitates working with the MAX+PLUS II/Viewlogic Powerview interface. Refer to MAX+PLUS II/Viewlogic Powerview Project File Structure.

Related Links:

- Go to *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual for more information on installation and details on the directories that are created during MAX+PLUS II installation. Go to MAX+PLUS II/Viewlogic Powerview Interface File Organization for information about the MAX+PLUS II/Viewlogic Powerview directories that are created during MAX+PLUS II installation.
- Go to the following topics for additional information:

MAX+PLUS II Getting Started version 8.1 (5.4 MB)

- This manual is also available in 4 parts:
  - Preface & Section 1: MAX+PLUS II Installation
  - Section 2: MAX+PLUS II - A Perspective
  - Section 3: MAX+PLUS II Tutorial
  - Appendices, Glossary & Index

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

MAX+PLUS II/Synopsys Software Requirements

The following applications are used to generate, process, synthesize, and verify a project with MAX+PLUS® II and Synopsys software:

Synopsys		Altera
Version 1998.02: Design Compiler FPGA Compiler Design Analyzer (optional) VHDL Compiler	HDL Compiler for Verilog VHDL System Simulator (VSS) (optional) PrimeTime version 1998.02-PT2.1(optional)	MAX+PLUS II Version 10.0

Compilation with the Synopsys Design Compiler and FPGA Compiler is available only on Sun SPARCstations running Solaris 2.4 or higher.

☞ The MAX+PLUS II **read.me** file provides up-to-date information on which versions of Synopsys applications are supported by the current version of MAX+PLUS II. It also provides information on installation and operating requirements. You should read the **read.me** file on the CD-ROM before installing the MAX+PLUS II software. After installation, you can open the **read.me** file from the MAX+PLUS II Help menu.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

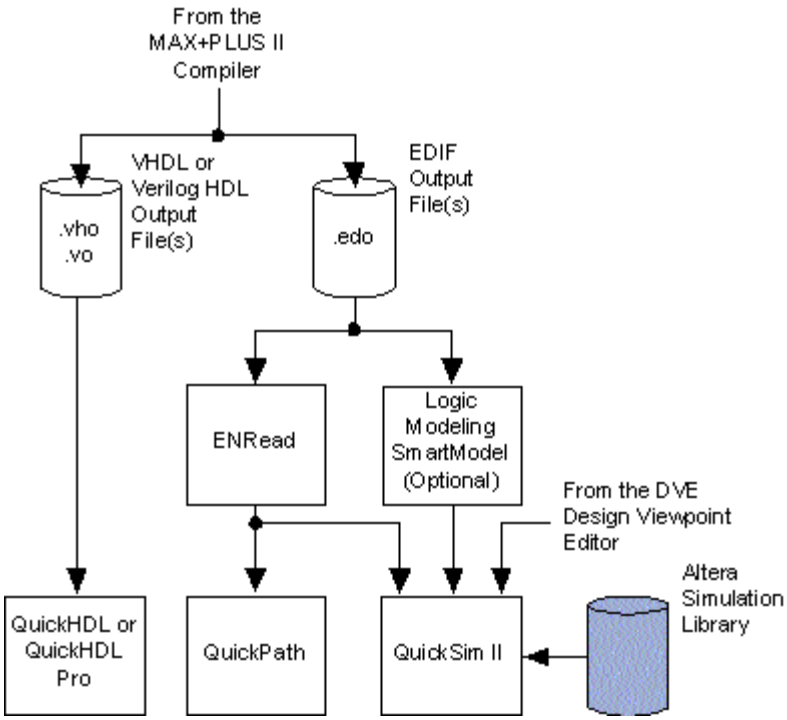
Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Project Simulation/Timing Analysis Flow

The following figure shows the project simulation and timing analysis flow for the MAX+PLUS® II /Mentor Graphics interface.

Figure 1. MAX+PLUS II/Mentor Graphics Project Simulation/Timing Analysis Flow

Alteraprovided items are shown in blue.



Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

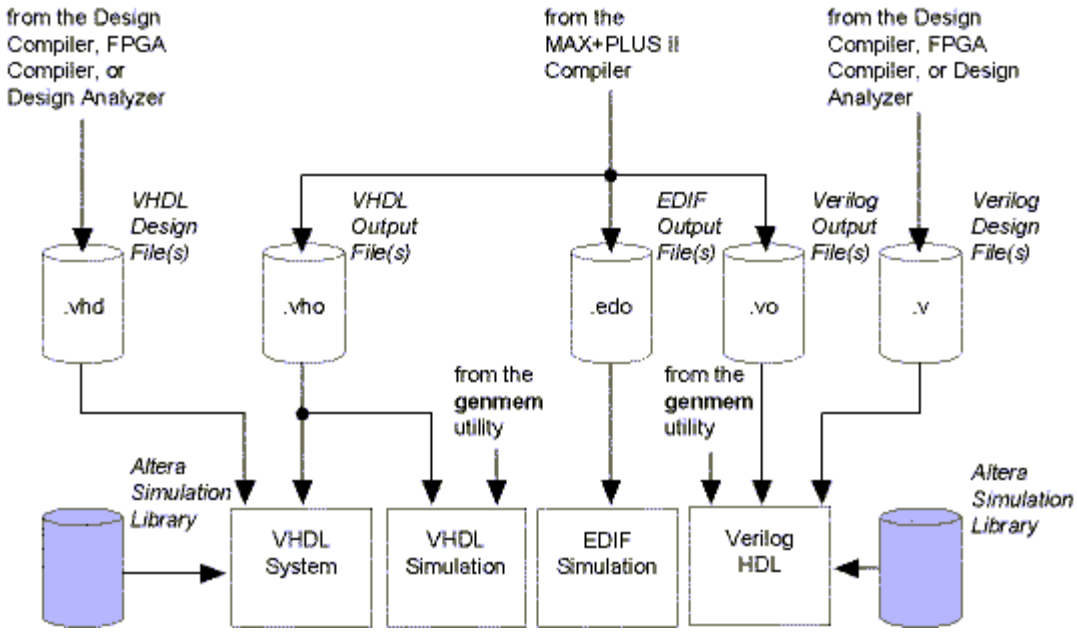
Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Project Simulation Flow

Figure 1 shows the project simulation flow for the MAX+PLUS® II /Synopsys interface.

Figure 1. MAX+PLUS II/Synopsys Project Simulation Flow

Altera-provided items are shown in blue.



The MAX+PLUS II/Synopsys design environment fully supports design verification with the Synopsys VHDL System Simulator (VSS). For pre-route simulation, you can simulate a design that has been compiled with one of the Synopsys compilers. For post-route simulation, you can simulate the VHDL Output File (**.vho**) that MAX+PLUS II® software generates during project compilation.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

MAX+PLUS II Software Support

MAX+PLUS® II software is intended only to support legacy designs. MAX+PLUS II software does not support MAX® II CPLDs, Cyclone®, Arria®, or Stratix® series FPGAs, or any newer devices.

Quartus® II software is Altera's primary development software and supports Altera's newest device families and most older device families. Download the free Quartus II Web Edition software today.

For MAX+PLUS II legacy software support, refer to the resources below.

Download

- MAX+PLUS II Software Updates
- MAX+PLUS II BASELINE Software
- MAX+PLUS II Advanced Synthesis Software

Installation and Licensing

- MAX+PLUS II BASELINE Installation Instructions
- Altera® Software Licensing (Request a license)
- MAX+PLUS II Software Licensing

MAX+PLUS II General Information

- Make the Move from MAX+PLUS II to Quartus II Software
- MAX+PLUS II Development Software Literature (Complete listing)
- *MAX+PLUS II Getting Started version 8.1* (PDF)
- *MAX+PLUS II Advanced Synthesis* (PDF)
- MAX+PLUS II Support Solutions
- ACCESS Key Guidelines for EDA tool support

Related Links

- MAX+PLUS II Users - Make the Move
- Quartus II Development Software Literature

MAX+PLUS II/Viewlogic Powerview Software Requirements

The following table shows the software applications that are used to generate, process, synthesize, and verify a project with MAX+PLUS® II and Cadence software:

Cadence		Altera
Version 97A: Concept Composer ValidCOMPILER **concept2alt** **vlog2alt** **altout**	VerilogLink Synergy HDL Direct (Concept 2.0 or later) Non-Graphic Simulation Environment (SE) RapidSIM, Verilog-XL, or Leapfrog **redifnet** (SunOS only)	MAX+PLUS II Version 10.0

The MAX+PLUS II **read.me** file provides up-to-date information on which versions of Cadence software applications are supported by the current version of MAX+PLUS II. It also provides information on installation and operating requirements. You should read the **read.me** file on the CD-ROM before installing the MAX+PLUS II software. After installation, you can open the **read.me** file from the MAX+PLUS II Help menu.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

The following applications and utilities are used to generate, process, synthesize, and verify a project with MAX+PLUS® II and Viewlogic Powerview software.

Viewlogic		Altera
ViewDraw	ViewGen	MAX+PLUS II version 9.3
VHDL Analyzer	ViewPath (optional)	
Vantage VHDL Analyzer	ViewTrace	
VHDL -> sym	ViewData Path	
edifneto	MOTIVE version 5.1.6 *Note (1)*	
edifneti	MOTIVE for Powerview version 3.2.1 (optional) *Note (1)*	
EEDIF (optional)	SDF2MTV (optional)	
MMP (optional)	Fusion/VCS	
vsm		

Note:

(1) MOTIVE for Powerview, a wrapper application for MOTIVE, provides a graphical user interface for the utilities (i.e., EEDIF, SDF2MTV, and MMP) used during a static timing verification with MOTIVE. MOTIVE alone does not accept EDIF files through the Setup Advisor.

The MAX+PLUS II **read.me** file provides up-to-date information on which versions of Viewlogic Powerview applications the current version of the MAX+PLUS II software supports. It also provides information on installation and operating requirements. You should read the **read.me** file on the CD-ROM before installing the MAX+PLUS II software. After installation, you can open the **read.me** file from the MAX+PLUS II Help menu.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Setting Up the MAX+PLUS II/Cadence Concept Work Environment for a Sun SPARCstation Running SunOS Software

If you are using Concept software on a Sun SPARCstation running SunOS software, you should also install the **redifnet** EDIF netlist reader utility to convert Concept schematics into MAX+PLUS II-compatible EDIF netlist files. To install the **redifnet** utility, follow these steps:

1. Copy the **redifnet** directory from the **/usr/maxplus2/simlib/concept/edifnet** directory to the Cadence system directory.
2. Copy the **redifnet** and **pinmap_start** files from the **/usr/maxplus2/simlib/concept/edifnet/bin** directory to the /<*Cadence system directory path*>/**tools/bin**.
3. Specify the **-/usr/maxplus2/simlib/concept/edifnet/max2sim** map file as a `PIN_MAP_FILE` in the **redifnet.cmd** file.
4. (Optional) Modify existing templates for directive files such as **compiler.cmd**, **vloglink.cmd**, and **global.cmd**. These templates are located in the **/usr/maxplus2/simlib/concept/edifnet/templates** directory.
5. (Optional) Modify the **expansion.dat** and **max2sim.map** files in the **/usr/maxplus2/simlib/concept/edifnet** directory.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Using Synplicity Synplify & MAX+PLUS II Software

The following topics describe how to use the Synplicity Synplify software with MAX+PLUS® II software. Click on one of the following topics for information:

This file is suitable for printing only. It does not contain hypertext links that allow you to jump from topic to topic.

Setting Up the MAX+PLUS II/Synplicity Working Environment

- Software Requirements
- MAX+PLUS II Directory Structure
- MAX+PLUS II/Synplicity Interface File Organization
- Synplicity-Provided Logic Libraries

Design Flow

Design Entry

- **Design Entry Flow**
- **VHDL**
  - Creating VHDL Designs for Use with MAX+PLUS II Software
  - Entering Resource Assignments
    - Assigning Pins
    - Assigning the Implement in EAB Logic Option
    - Modifying the Assignment & Configuration File with the **setacf** Utility
- **Verilog HDL**
  - Creating Verilog HDL Designs for Use with MAX+PLUS II Software
  - Entering Resource Assignments
    - Assigning Pins
    - Assigning the Implement in EAB Logic Option
    - Modifying the Assignment & Configuration File with the **setacf** Utility

Synthesis & Optimization

- Synthesizing & Optimizing VHDL or Verilog HDL Files with Synplify Software
- Analyzing VHDL or Verilog HDL Designs with the Synplify HDL Analyst

Compilation

- Project Compilation Flow

- Compiling Projects with MAX+PLUS II Software
  - Synplicity Synplify-Specific Compiler Settings

Device Programming

- Programming Altera Devices

Related Topics:

- Go to the following topics for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware
  - Synplicity web site (http://www.synplicity.com)

Setting Up the MAX+PLUS II/Synplicity Working Environment

To use MAX+PLUS® II software with Synplicity software, you must first install the MAX+PLUS II software, then establish an environment that facilitates entering and processing designs. The MAX+PLUS II/Synplicity interface is installed automatically when you install the MAX+PLUS II software on your computer. Ensure that you have correctly installed the MAX+PLUS II and Synplicity software versions described in the MAX+PLUS II/Synplicity Software Requirements.

You do not need to set any initialization or project variables before using Synplicity software with MAX+PLUS II software. Synplicity software features Direct Synthesis Technology that performs technology mapping directly to Altera® device logic cells by inserting architecture-specific primitives to implement features such as logic cells, parallel expanders, carry chains, and cascade chains.

Related Topics:

- Refer to the following sources for more information:
  - Go to *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual for more information on installation and details on the directories created during MAX+PLUS II installation.
  - Go to MAX+PLUS II/Synplicity Interface File Organization in these MAX+PLUS II ACCESS℠ Key topics for information about the MAX+PLUS II/Synplicity directories that are created during MAX+PLUS II installation.
- Go to the following topics for additional information:
  - ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
  - This manual is also available in 4 parts:
    - Preface & Section 1: MAX+PLUS II Installation
    - Section 2: MAX+PLUS II - A Perspective
    - Section 3: MAX+PLUS II Tutorial
    - Appendices, Glossary & Index

MAX+PLUS II/Synplicity Software Requirements

Table 1 shows the software applications that are used to generate, process, synthesize, and verify a project with MAX+PLUS® II and Synplicity software:

Table 1. Software Requirements

Synplicity	Altera
version 3.0C1: Synplify HDL Analyst	MAX+PLUS II version 9.4

☞ The MAX+PLUS II **read.me** file provides up-to-date information on which versions of Synplicity software applications are supported by the current version of MAX+PLUS II. It also provides information on installation and operating requirements. You should read the **read.me** file on the CD-ROM before installing the MAX+PLUS II software. After installation, you can open the **read.me** file from the MAX+PLUS II Help menu.

MAX+PLUS II Directory Structure (Synplicity Environment)

In the MAX+PLUS® II software, a project name is the name of a top-level design file, without the filename extension. This design file can be an EDIF, Verilog HDL, or VHDL netlist file; an AHDL Text Design File (TDF); or any other MAX+PLUS II-supported design file. You can use a standard EDA tool to create an EDIF netlist file and import it into MAX+PLUS II software as an EDIF Input File (**.edf**).

Project design files and output files are stored in the project directory, with the exception of standard library functions provided by Altera or another EDA tool vendor. The MAX+PLUS II software stores the connectivity data on the links between design files in a hierarchical project in a Hierarchy Interconnect File (**.hif**) in the project directory, but refers to the entire project only by its project name. The MAX+PLUS II Compiler uses the HIF to build a single, fully flattened project database that integrates all the design files in a project hierarchy.

MAX+PLUS II/Synplicity Interface File Organization

Table 1 shows the MAX+PLUS® II/Synplicity interface subdirectories that are created in the MAX+PLUS II system directory (by default, the **/usr/maxplus2** directory) during MAX+PLUS II installation. For information on the other directories that are created during MAX+PLUS II installation, see "MAX+PLUS II File Organization" in *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual.

Table 1. MAX+PLUS II Directory Organization

Directory	Description
./lmf	Contains the Altera-provided Library Mapping File, **synplcty.lmf**, which maps Synplicity logic functions to equivalent MAX+PLUS II logic functions.

Related Topics:

- Go to the following topics, which are available on the web, for additional information:
  - ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
  - This manual is also available in 4 parts:
    - Preface & Section 1: MAX+PLUS II Installation
    - Section 2: MAX+PLUS II - A Perspective
    - Section 3: MAX+PLUS II Tutorial
    - Appendices, Glossary & Index

Synplicity-Provided Logic Libraries

Synplicity software provides the **altera** logic library that is used for synthesizing and compiling VHDL and Verilog HDL designs. The **altera** library includes the following library files:

Library:	Description:
altera.vhd	A VHDL logic function library that includes the LCELL, SOFT, GLOBAL, CASCADE, and CARRY primitives for controlling design synthesis and fitting. These primitives can be instantiated directly in your VHDL file. These models allow you to perform functional VHDL simulation while maintaining an architecture-independent VHDL description.
altera.v	A Verilog HDL logic function library equivalent to the **altera.vhd** library file.

☞ You can create your own libraries of custom logic functions for use with Synplicity software. You can use custom logic functions to incorporate an EDIF Input File, Text Design File (**.tdf**), or any other MAX+PLUS® II-supported design file into a project. The MAX+PLUS II software uses the **synplcty.lmf** Library Mapping File to map standard Synplicity logic functions to equivalent MAX+PLUS II logic functions. To use custom logic functions, you can create a custom LMF that maps your custom logic functions to the equivalent EDIF Input File, Text Design File (**.tdf**), or other design file. Go to "Library Mapping File" in MAX+PLUS II Help for more information.

Synplicity Design Flow

Figure 1 shows the typical design flow for logic circuits created and processed with Synplicity and MAX+PLUS® II software. Design Entry Flow, Project Compilation Flow, and Device Programming Flow show detailed diagrams of each stage of the design flow.

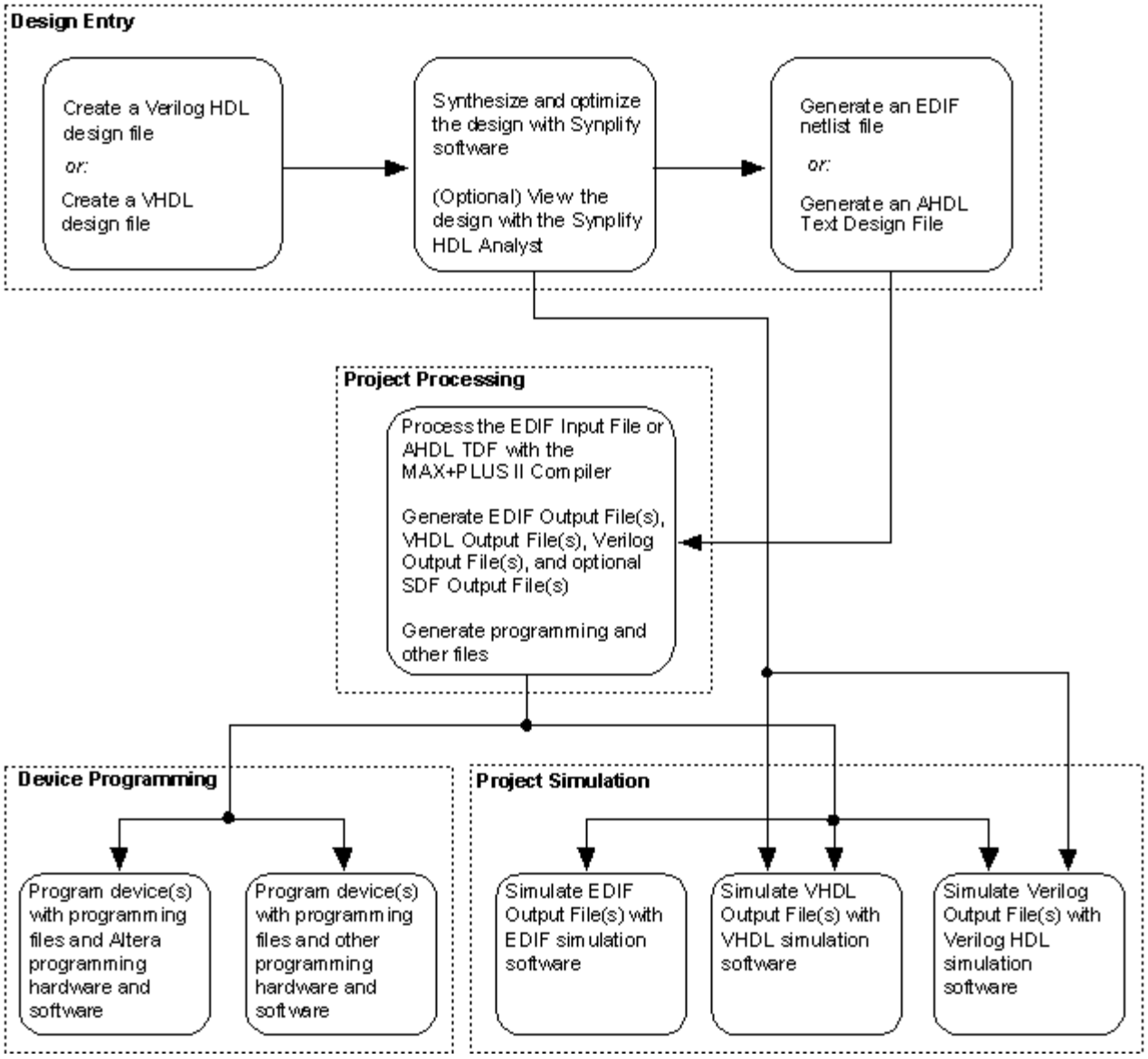


Figure 1. Design Flow between Synplicity & MAX+PLUS II Software

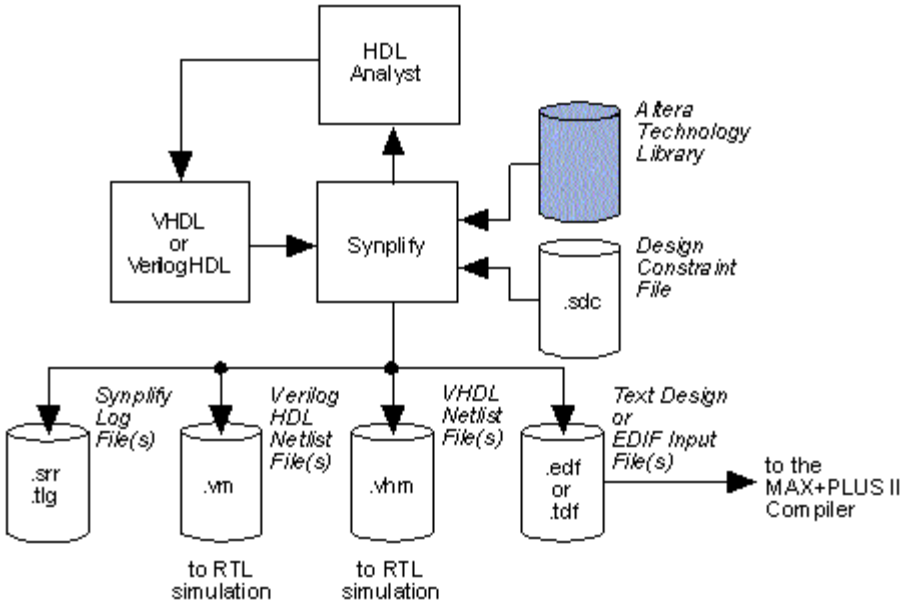
---

Synplicity Design Entry Flow

Figure 1 shows the design entry flow for the MAX+PLUS® II/Synplicity interface.

Figure 1. MAX+PLUS II/Synplicity Design Entry Flow

Altera-provided items are shown in blue.



Creating VHDL Designs for Use with MAX+PLUS II Software

You can create VHDL design files with the MAX+PLUS® II Text Editor or another standard text editor and save them in the appropriate directory for your project. The MAX+PLUS II Text Editor offers the following advantages:

- VHDL templates are available with the **VHDL Templates** command (Templates menu). These templates are also available in the ASCII **vhdl.tmp** file, which is located in the **/usr/maxplus2** directory.
- If you use the MAX+PLUS II Text Editor to create your VHDL design, you can use the **Syntax Coloring** command (Options menu). The Syntax Coloring feature displays keywords and other elements of text in text files in different colors to distinguish them from other forms of syntax.

To create a VHDL design and convert it to an EDIF netlist file for use with MAX+PLUS II software, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Synplicity Working Environment.
2. Instantiate any MAX+PLUS II-supported logic function in your VHDL design. You can enter the following functions:
    - Parameterized and non-parameterized megafunctions. MAX+PLUS II software also supports all functions in the library of parameterized modules (LPM) 2.1.0, except the truth table, finite state machine, and pad functions.
    - Macrofunctions, including 74-series functions.
    - Buffer primitives, including `lcell`, `soft`, `global`, `carry`, and `cascade`. The Synplicity **altera.vhd** library provides synthesis support for these functions.
    - MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.

    Choose **Primitives**, **Old-Style Macrofunctions**, and **Megafunctions/LPM** from the MAX+PLUS II Help menu for information on all MAX+PLUS II-supported functions.

3. If your design uses functions from the **altera.vhd** library, add the following Library and Use clauses to the top of a file that instantiates the macrofunction(s):

```
library altera;
use altera.maxplus2.all;
```

4. For each MAX+PLUS II-supported logic function, include a `black_box` synthesis directive. See Figure 1. You can omit this step for functions from the **altera.vhd** library.

5. For any parameterized function, declare all parameters used in the function, their types, and their values. Attribute Declarations are used to declare the `black_box` attribute and the name and type of each parameter. The `black_box` attribute has the `boolean` type; refer to MAX+PLUS II Help for information on whether a parameter is of `integer` or `string` type. Attribute Specifications then assign values to each parameter. Figure 1 shows a VHDL design file that instantiates the `lpm_ram_dq` function.

Figure 1. VHDL Design File with LPM Function Instantiation

```
entity myram is
port (clock, we: in bit;
      data : in bit_vector (3 downto 0);
      address: in bit_vector (1 downto 0);
      q: out bit_vector (3 downto 0));
end myram;

architecture arch1 of myram is

    -- Declare the component

    component myram_4x4
              port (data: in bit_vector (3 downto 0);
                        address: in bit_vector (1 downto 0);
                        inclock, outclock, we: in bit;
                        q: out bit_vector (3 downto 0) );
    end component;

--  Declare the black_box and parameters and their types

attribute black_box: boolean;
attribute LPM_WIDTH: integer;
attribute LPM_WIDTHAD: integer;
attribute LPM_TYPE: string;


-- Assign values to each attribute

attribute black_box of myram_4x4: component is true;
attribute LPM_WIDTH of myram_4x4: component is 4;
attribute LPM_WIDTHAD of myram_4x4: component is 2;
-- Specify the name of the LPM function as the value of the
-- LPM_TYPE attribute
attribute LPM_TYPE of myram_4x4: component is "LPM_RAM_DQ"

begin
          -- Instantiate the LPM component
          u1: myram_4x4  port map(data, address, clock,
                                                    clock, we, q);

end arch1;
```

6. (Optional) Enter resource assignments for your VHDL design, as described in Entering Resource Assignments.

7. After you have completed your VHDL design, synthesize and optimize it with Synplify software, as described in Synthesizing & Optimizing VHDL or Verilog HDL Files with Synplify Software.

Related Topics:

- Go to Compiling Projects with MAX+PLUS II Software in these MAX+PLUS II ACCESS℠ Key topics for related information.

Entering Resource Assignments

The MAX+PLUS® II software allows you to enter a variety of resource and device assignments for your projects. Resource assignments are used to assign logic functions to a particular pin, logic cell, I/O cell, embedded cell, row, column, Logic Array Block (LAB), Embedded Array Block (EAB), chip, clique, local routing, logic option, timing requirement, or connected pin group. In MAX+PLUS II software, you can enter all types of resource and device assignments with Assign menu commands. You can also enter pin, logic cell, I/O cell, embedded cell, LAB, EAB, row, and column assignments in the MAX+PLUS II Floorplan Editor. The Assign menu commands and the Floorplan Editor all save assignment information in the ASCII Assignment & Configuration File (**.acf**) for the project. In addition, you can edit ACFs manually in any standard text editor or with the **setacf** utility.

VHDL & Verilog HDL Design Files

When you use Synplicity Synplify software, you can assign a limited subset of these resource assignments by specifying attributes in the Synplify Design Constraints File (**.sdc**) or in the VHDL or Verilog HDL design files. The Synplify software automatically incorporates these attributes into the EDIF netlist file(s) generated from the HDL design files. MAX+PLUS II then automatically converts assignment information from the EDIF Input File into the ACF format. The following topics describe how to make MAX+PLUS II-compatible resource assignments before design processing with the Synplify software:

- Assigning Pins
- Assigning the Implement in EAB Logic Option
- Modifying the Assignment & Configuration File with the **setacf** Utility

Related Topics:

- Refer to the following sources for more information:
    - Go to the ***Synplify User's Guide*** for details on how to assign properties.
    - Go to "Resource Assignments in EDIF Input Files" and "Assigning Resources in a Third-Party Design Editor" in MAX+PLUS II Help for more information on assignments or properties that can be assigned when you use the Synplify software.
    - Go to "resource assignments" or "ACF, format" in MAX+PLUS II Help using **Search for Help on** (Help menu) for information on entering assignments in the MAX+PLUS II software with Assign menu commands or in an ACF.

Assigning Pins

You can assign a single port to a specific pin to ensure that the signal is always associated with that pin, regardless of future changes to the project. You can specify pins in VHDL or Verilog HDL designs, or in a Synplify Design Constraints File (**.sdc**). If you add timing constraints or resource assignments in a separate Synplify Design Constraints File (**.sdc**), you must add the Synplify Design Constraints File (**.sdc**) to the project by adding it to the *Source Files* list in the Synplify window.

If your design uses resource assignment attributes that you wish to pass to the MAX+PLUS® II software, you should save your file in EDIF netlist file format. See Entering Resource Assignments for more information.

VHDL Syntax

Use the following syntax to assign a pin in VHDL:

```
attribute altera_chip_pin_lc : string;
attribute altera_chip_pin_lc of <port name> : signal is "@<pin number(s)>"
```

Example:

```
attribute altera_chip_pin_lc : string;
attribute altera_chip_pin_lc of result : signal is
        "@17, @166, @191, @152, @15, @148, @147, @149"
```

Verilog HDL Syntax

Use the following syntax to assign a pin in Verilog HDL:

```
<port name> /* synthesis altera_chip_pin_lc="@<pin number(s)>" */;
```

Example:

```
output [7:0] sum /* synthesis altera_chip_pin_lc="@17, @166, @191, @152, \
        @15, @148, @147, @149" */;
```

Synplify Design Constraints File Syntax

Use the following syntax to assign a pin in a Synplify Design Constraints file:

```
define_attribute <port name> altera_chip_pin_lc "@<pin number>"
```

Example:

```
define_attribute {DATA0[7:0]} altera_chip_pin_lc "@115,@116,@117,
@118,@119,@120,@121,@122"
```

Related Topics:

- Refer to the following sources for related information:
  - Go to Entering Resource Assignments in these MAX+PLUS II ACCESS℠ Key topics for information on entering other types of assignments.
  - Go to "Devices & Adapters" and "Assigning a Device" in MAX+PLUS II Help for information on device pin-outs and assigning devices, respectively, in the MAX+PLUS II software.

Assigning the Implement in EAB Logic Option

You can assign the Implement in EAB logic option to individual logic functions in a FLEX® 10K design. This option directs the MAX+PLUS® II Compiler's Logic Synthesizer module to implement the function in an embedded array block (EAB) rather than in logic cell(s). You can specify the Implement in EAB Logic Option in VHDL or Verilog HDL designs, or in a Synplify Design Constraints File (**.sdc**). If you add timing constraints or resource assignments in a separate Synplify Design Constraints File (**.sdc**), you must add the Synplify Design Constraints File (**.sdc**) to the project by adding it to the *Source Files* list in the Synplify window.

If your design uses resource assignment attributes that you wish to pass to the MAX+PLUS II software, you should save your file in EDIF netlist file format. See Entering Resource Assignments for more information.

VHDL Syntax

Use the following syntax to assign the Implement in EAB logic option in VHDL:

```
attribute altera_implement_in_eab : boolean;
attribute altera_implement_in_eab of <port name>: label is true;
```

Example:

```
attribute altera_implement_in_eab of U1: label is true;
    begin
        U1: mymux port map (in1 => a, sel => s, dout => o);
```

Verilog HDL Syntax

Use the following syntax to assign the Implement in EAB logic option in Verilog HDL:

<*module or architecture name*> `/* synthesis altera_implement_in_eab=1 */;`

Example:

```
sqrtb sq (.z(sqa), .a(a))  /* synthesis altera_implement_in_eab=1 */;
defparam sq.asize = 8;
```

Synplify Design Constraints File Syntax

Use the following syntax to assign the Implement in EAB logic option in a Synplify Design Constraints File (**.sdc**):

`define_attribute {`<*module or architecture name*>`} altera_implement_in_eab 1`

Example:

```
define_attribute {inst1.sqrt8} altera_implement_in_eab 1
```

Related Topics:

- Refer to the following sources for more information:
  - Go to Entering Resource Assignments in these MAX+PLUS II ACCESS^{SM} Key topics for information on entering other types of assignments.
  - Go to "Resource Assignments in EDIF Input Files" and "Assigning Resources in a Third-Party Design Editor" in MAX+PLUS II Help for complete and up-to-date information on other logic options and logic synthesis style assignments, including definitions and syntax of these assignments.
- Go to FLEX 10K Device Family, which is available on the web, for additional information.

Modifying the Assignment & Configuration File with the setacf Utility

Altera provides the **setacf** utility to help you modify a project's Assignment & Configuration File (**.acf**) from the command line, without opening the file with a text editor. Type `setacf -h` ↵ at a UNIX or DOS prompt to get help on this utility.

Creating Verilog HDL Designs for Use with MAX+PLUS II Software

You can create Verilog HDL design files with the MAX+PLUS II Text Editor or another standard text editor and save them in the appropriate directory for your project. The MAX+PLUS II Text Editor offers the following advantages:

- Verilog HDL templates are available with the **Verilog Templates** command (Templates menu). These templates are also available in the ASCII **verilog.tmp** file, which is located in the **/usr/maxplus2** directory.
- If you use the MAX+PLUS II Text Editor to create your Verilog HDL design, you can use the **Syntax Coloring** command (Options menu). The Syntax Coloring feature displays keywords and other elements of text in text files in different colors to distinguish them from other forms of syntax.

To create a Verilog HDL design and convert it to an EDIF netlist file for use with MAX+PLUS II software, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Synplicity Working Environment.
2. Instantiate any MAX+PLUS II-supported logic function in your Verilog HDL design. You can enter the following functions:
    - Parameterized and non-parameterized megafunctions. MAX+PLUS II software also supports all functions in the library of parameterized modules (LPM) 2.1.0, except the truth table, finite state machine, and pad functions.
    - Macrofunctions, including 74-series functions.
    - Buffer primitives, including `lcell`, `soft`, `global`, `carry`, and `cascade`. The Synplicity **altera.v** library provides synthesis support for these functions.
    - MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.

    Choose **Primitives**, **Old-Style Macrofunctions**, and **Megafunctions/LPM** from the MAX+PLUS II Help menu for information on all MAX+PLUS II-supported functions.
3. If your design uses functions from the **altera.v** library, add the library file name to the top of the *Source Files* list in the Synplify window.
4. For each MAX+PLUS II-supported logic function, include a `black_box` synthesis directive. You can omit this step for functions from the **altera.v** library.
5. For any parameterized function, you must declare all parameters used in the function, and their values. Figure 1 shows a Verilog HDL file that instantiates the `lpm_ram_dq` function. A comment in the Module Declaration contains the `synthesis black_box` directive and parameter names and values. This comment must immediately follow the port list and precede the closing semicolon (`;`). When you instantiate an LPM function, the LPM function name must be specified as the value of the `LPM_TYPE` parameter. In addition, each parameter must be listed on a separate line. See Figure 1.

Figure 1. Verilog HDL Design File with LPM Function Instantiation

```
// Define the black box
module myram_64x16 (data, address, inclock, outclock, we, q)
/* synthesis black_box

                  LPM_WIDTH=16
                  LPM_WIDTHAD=6
                  LPM_TYPE="LPM_RAM_DQ"  */ ;
```

```
input [15:0] data;
input [5:0] address;
input inclock, outclock;
input we;
output [15:0] q;

endmodule

// Instantiate the LPM parameterized module in the
//    higher-level module myram
module myram(clock, we, data, address, q);
input clock, we;
input [15:0] data;
input [5:0] address;
output [15:0] q;


       myram_64x16 inst1 (data, address, clock, clock, we, q);

endmodule
```

6. (Optional) Enter resource assignments for your Verilog HDL design, as described in Entering Resource Assignments.

7. After you have completed your Verilog HDL design, synthesize and optimize it with Synplify software, as described in Synthesizing & Optimizing VHDL or Verilog HDL Files with Synplify Software.

Related Topics:

- Go to Compiling Projects with MAX+PLUS II Software in these MAX+PLUS II ACCESS℠ Key topics for related information.

Entering Resource Assignments

The MAX+PLUS® II software allows you to enter a variety of resource and device assignments for your projects. Resource assignments are used to assign logic functions to a particular pin, logic cell, I/O cell, embedded cell, row, column, Logic Array Block (LAB), Embedded Array Block (EAB), chip, clique, local routing, logic option, timing requirement, or connected pin group. In MAX+PLUS II software, you can enter all types of resource and device assignments with Assign menu commands. You can also enter pin, logic cell, I/O cell, embedded cell, LAB, EAB, row, and column assignments in the MAX+PLUS II Floorplan Editor. The Assign menu commands and the Floorplan Editor all save assignment information in the ASCII Assignment & Configuration File (**.acf**) for the project. In addition, you can edit ACFs manually in any standard text editor or with the **setacf** utility.

VHDL & Verilog HDL Design Files

When you use Synplicity Synplify software, you can assign a limited subset of these resource assignments by specifying attributes in the Synplify Design Constraints File (**.sdc**) or in the VHDL or Verilog HDL design files. The Synplify software automatically incorporates these attributes into the EDIF netlist file(s) generated from the HDL design files. MAX+PLUS II then automatically converts assignment information from the EDIF Input File into the ACF format. The following topics describe how to make MAX+PLUS II-compatible resource assignments before design processing with the Synplify software:

- Assigning Pins
- Assigning the Implement in EAB Logic Option
- Modifying the Assignment & Configuration File with the **setacf** Utility

Related Topics:

- Refer to the following sources for more information:

- Go to the ***Synplify User's Guide*** for details on how to assign properties.
- Go to "Resource Assignments in EDIF Input Files" and "Assigning Resources in a Third-Party Design Editor" in MAX+PLUS II Help for more information on assignments or properties that can be assigned when you use the Synplify software.
- Go to "resource assignments" or "ACF, format" in MAX+PLUS II Help using **Search for Help on** (Help menu) for information on entering assignments in the MAX+PLUS II software with Assign menu commands or in an ACF.

Assigning Pins

You can assign a single port to a specific pin to ensure that the signal is always associated with that pin, regardless of future changes to the project. You can specify pins in VHDL or Verilog HDL designs, or in a Synplify Design Constraints File (**.sdc**). If you add timing constraints or resource assignments in a separate Synplify Design Constraints File (**.sdc**), you must add the Synplify Design Constraints File (**.sdc**) to the project by adding it to the *Source Files* list in the Synplify window.

If your design uses resource assignment attributes that you wish to pass to the MAX+PLUS® II software, you should save your file in EDIF netlist file format. See Entering Resource Assignments for more information.

VHDL Syntax

Use the following syntax to assign a pin in VHDL:

```
attribute altera_chip_pin_lc : string;
attribute altera_chip_pin_lc of <port name> : signal is "@<pin number(s)>"
```

Example:

```
attribute altera_chip_pin_lc : string;
attribute altera_chip_pin_lc of result : signal is
        "@17, @166, @191, @152, @15, @148, @147, @149"
```

Verilog HDL Syntax

Use the following syntax to assign a pin in Verilog HDL:

```
<port name> /* synthesis altera_chip_pin_lc="@<pin number(s)>" */;
```

Example:

```
output [7:0] sum /* synthesis altera_chip_pin_lc="@17, @166, @191, @152, \
        @15, @148, @147, @149" */;
```

Synplify Design Constraints File Syntax

Use the following syntax to assign a pin in a Synplify Design Constraints file:

```
define_attribute <port name> altera_chip_pin_lc "@<pin number>"
```

Example:

```
define_attribute {DATA0[7:0]} altera_chip_pin_lc "@115,@116,@117,
@118,@119,@120,@121,@122"
```

Related Topics:

- Refer to the following sources for related information:
  - Go to Entering Resource Assignments in these MAX+PLUS II ACCESS℠ Key topics for information on entering other types of assignments.
  - Go to "Devices & Adapters" and "Assigning a Device" in MAX+PLUS II Help for information on device pin-outs and assigning devices, respectively, in the MAX+PLUS II software.

Assigning the Implement in EAB Logic Option

You can assign the Implement in EAB logic option to individual logic functions in a FLEX® 10K design. This option directs the MAX+PLUS® II Compiler's Logic Synthesizer module to implement the function in an embedded array block (EAB) rather than in logic cell(s). You can specify the Implement in EAB Logic Option in VHDL or Verilog HDL designs, or in a Synplify Design Constraints File (**.sdc**). If you add timing constraints or resource assignments in a separate Synplify Design Constraints File (**.sdc**), you must add the Synplify Design Constraints File (**.sdc**) to the project by adding it to the *Source Files* list in the Synplify window.

☞ If your design uses resource assignment attributes that you wish to pass to the MAX+PLUS II software, you should save your file in EDIF netlist file format. See Entering Resource Assignments for more information.

VHDL Syntax

Use the following syntax to assign the Implement in EAB logic option in VHDL:

```
attribute altera_implement_in_eab : boolean;
attribute altera_implement_in_eab of <port name>: label is true;
```

Example:

```
attribute altera_implement_in_eab of U1: label is true;
    begin
        U1: mymux port map (in1 => a, sel => s, dout => o);
```

Verilog HDL Syntax

Use the following syntax to assign the Implement in EAB logic option in Verilog HDL:

<*module or architecture name*> `/* synthesis altera_implement_in_eab=1 */;`

Example:

```
sqrtb sq (.z(sqa), .a(a))  /* synthesis altera_implement_in_eab=1 */;
defparam sq.asize = 8;
```

Synplify Design Constraints File Syntax

Use the following syntax to assign the Implement in EAB logic option in a Synplify Design Constraints File (**.sdc**):

`define_attribute {`<*module or architecture name*>`} altera_implement_in_eab 1`

Example:

```
define_attribute {inst1.sqrt8} altera_implement_in_eab 1
```

Related Topics:

- Refer to the following sources for more information:
  - Go to Entering Resource Assignments in these MAX+PLUS II ACCESS℠ Key topics for information on entering other types of assignments.
  - Go to "Resource Assignments in EDIF Input Files" and "Assigning Resources in a Third-Party Design Editor" in MAX+PLUS II Help for complete and up-to-date information on other logic options and logic synthesis style assignments, including definitions and syntax of these assignments.
- Go to FLEX 10K Device Family, which is available on the web, for additional information.

Modifying the Assignment & Configuration File with the setacf Utility

Altera provides the **setacf** utility to help you modify a project's Assignment & Configuration File (**.acf**) from the command line, without opening the file with a text editor. Type `setacf -h` ↵ at a UNIX or DOS prompt to get help on this utility.

Synthesizing & Optimizing VHDL or Verilog HDL Files with Synplify Software

You can create and process VHDL or Verilog HDL files and convert them to Altera® Hardware Description Language (AHDL) Text Design Files (**.tdf**) or EDIF Input Files (**.edf**) that can be processed by the MAX+PLUS® II Compiler. The MAX+PLUS II Compiler can process a VHDL or Verilog HDL file that has been synthesized by Synplify software, saved as an AHDL TDF or an EDIF netlist file, and imported into the MAX+PLUS II software. The information presented here describes only how to use VHDL or Verilog HDL files that have been processed by Synplify software. For information on direct MAX+PLUS II support for VHDL or Verilog HDL Design Files, go to MAX+PLUS II VHDL or Verilog HDL Help.

To process a VHDL or Verilog HDL file with Synplify software for use with MAX+PLUS II software, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Synplicity Working Environment.

2. Create a VHDL file, <*design name*>**.vhd**, or a Verilog HDL file, <*design name*>**.v**, using the MAX+PLUS II Text Editor or another standard text editor and save it in a working directory. Go to Creating VHDL Designs for Use with MAX+PLUS II Software or Creating Verilog HDL Designs for Use with MAX+PLUS II Software for more information on HDL design entry.

3. Start the Synplify software:

   ✔ On a UNIX workstation, type `synplify` ↵ at a UNIX prompt from your working directory.

   or:

   ✔ On a PC, double-click the **synplify.exe** icon in your **\synplicity\bin** directory.

4. Create a new project:

   1. Choose **New** (File menu) to display the **New** dialog box, then select *Project* from the list. Choose **OK.**

   2. Choose the **Add** button from the Project window. The **Add Source Files** dialog box is displayed.

3. Select your design file(s) and choose the **Open** button to add the file(s) to your *Source Files* list in the Synplify window.

☞ If you wish to create a hierarchical project, make sure the top-level design file is at the bottom of the *Source Files* list by selecting the file and dragging it to the bottom of the list.

5. Select the target Altera device:

   1. Choose the **Change** button in the *Target* section. The **Set Device Option** dialog box is displayed.
   2. Select an Altera MAX® (which includes Classic) or FLEX® device family from the *Technology* list.
   3. Select a device from the *Part* list.
   4. (Optional) Turn on the *Map logic cells to LCELLs* option to increase performance. However, turning on this option may decrease area optimization.

      For MAX or Classic designs, specify the following options:

      1. Enter an appropriate value for the *Percent of design to optimize for timing* box.
      2. Enter an appropriate value for the *Maximum cell fan-in* box.
      3. (Optional) Turn on the *Make Non-critical Cells Soft* option to allow the MAX+PLUS II software to reduce the number of logic cells used in implementing non-timing critical portions of the design.

      or:

      ✔ For FLEX designs, select an appropriate value from the *Speed Grade* list.

   5. Select *EDIF* or *AHDL* in the *Result Format* box to specify the output file format from the Synplify software. Choose **OK**.

☞ Saving your project in AHDL TDF format may improve compilation time. However, if your design uses resource assignment attributes, you should save your file in EDIF netlist file format. See Entering Resource Assignments for more information.

6. Enter the frequency value for the project in the *Frequency (MHz)* box in the Synplify window.
7. (Optional) Turn on the *Symbolic FSM Compiler* option in the Synplify window to direct the Synplify software to automatically find and re-encode state machines in your design. Turning this option on may reduce unnecessary states and transitional logic.
8. Run the Synplify software by choosing the **Run** button in the Synplify window. Synplify software synthesizes and optimizes the design, and creates the EDIF netlist file <*design name*>**.edf** or the AHDL TDF <*design name*>**.tdf**.
9. (Optional) Run the HDL Analyst to analyze and evaluate the performance of your design, as described in Analyzing VHDL or Verilog HDL Designs with the Synplify HDL Analyst.
10. (Optional) Add appropriate timing constraints in a separate Synplify Design Constraints File (**.sdc**) or in the VHDL or Verilog HDL source file. If you add timing constraints or resource assignments in a separate **.sdc** file, you must add the **.sdc** file to the *Source Files* list in the Synplify window.
11. Correct any errors or warnings.

12. If you have corrected errors or warnings, or added timing constraints to your project, repeat step 8 to implement the changes in your synthesized design.

13. Create the /<*project directory*>/**max2** subdirectory.

14. Copy the <*design name*>**.edf** or <*design name*>**.tdf** generated in step 8 to the /<*project directory*>/**max2** directory.

15. Process your design with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Related Topics:

- Go to the following topics, which are available on the web, for additional information:
  - FLEX Devices
  - MAX Devices
  - Classic Device Family

Analyzing VHDL or Verilog HDL Designs with the Synplify HDL Analyst

You can use the optional Synplify HDL Analyst to analyze and evaluate the performance of your design graphically. The Synplify HDL Analyst instantly generates Register Transfer Level (RTL) schematics, as well as technology-mapped, gate-level schematics. You can instantly identify and fix potential problems earlier in the design cycle by cross-probing between the RTL schematics, gate-level schematics, and HDL source code. The Synplify HDL Analyst also highlights critical paths within the design to show which signals require optimization for performance. After you determine the critical speed paths, you can add timing constraints either to the VHDL or Verilog HDL source file or to a separate Synplify Design Constraints File (**.sdc**) to improve design performance.

To use the Synplify HDL Analyst after synthesizing your design with Synplify software, go through the following steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS® II/Synplicity Working Environment.

2. Create a VHDL or Verilog HDL design and save it in your working directory, as described in Creating VHDL Designs for Use with MAX+PLUS II Software or Creating Verilog HDL Designs for Use with MAX+PLUS II Software.

3. Synthesize and optimize your VHDL or Verilog HDL design with Synplify software, as described in Synthesizing & Optimizing VHDL or Verilog HDL Files with Synplify Software.

4. Choose an HDL Analyst view:

   ✔ Choose **RTL View** (HDL_Analyst menu) to view the RTL schematic. When you select this view, the HDL Analyst displays a graphical representation of the design and the mouse pointer becomes a plus (+) symbol.

   or:

   ✔ Choose **Technology View** (HDL_Analyst menu) to view the gate-level schematic. When you select this view, the HDL Analyst displays a graphical representation of the design and the mouse pointer becomes a plus (+) symbol.

5. In either the RTL or Technology View, perform one or more of the following actions:

- Double-click the plus (+) symbol pointer on a port name or symbol to cross-probe your VHDL or Verilog HDL source design files.

  ☞ Because Synplify combines the `a + b` and `a - b` operations, cross-probing will highlight the Case Statement that defines both functions.

- Choose **Find** (HDL_Analyst menu) to select specific signals quickly in your design.
- Choose **Show Critical Path** (HDL_Analyst menu) to highlight the critical paths in the design.
- Select **Filter Schematic** (HDL_Analyst menu) to show only the nodes you have selected.

6. If necessary, correct the design and repeat the steps described in Synthesizing & Optimizing VHDL or Verilog HDL Files with Synplify Software.
7. Process your design with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Compiling Projects with MAX+PLUS II Software

The MAX+PLUS® II Compiler can process design files in a variety of formats. This topic describes how to use MAX+PLUS II software to compile projects in which the top-level design file is an EDIF Input File (with the extension **.edf**).

☞ Refer to the following sources for additional information:

- Go to MAX+PLUS II Help for information on compiling VHDL and Verilog HDL, design files directly with the MAX+PLUS II Compiler.
- Go to Running Synopsys Compilers from MAX+PLUS II Software for information on running the Synopsys Design Compiler or FPGA Compiler software on a VHDL or Verilog HDL design from within the MAX+PLUS II Compiler window.

To compile a design (also called a "project") with MAX+PLUS II software, go through the following steps:

1. Create design files that are compatible with the MAX+PLUS II software and convert them into EDIF Input Files with the extension **.edf**. Specific instructions for some tools are described in these MAX+PLUS II ACCESS℠ Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your design entry or synthesis and optimization tool.
2. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, create a mapping for each function in a Library Mapping File (**.lmf**) to map the custom symbol to the corresponding EDIF Input File, AHDL Text Design File (**.tdf**), or other MAX+PLUS II-supported design file. These custom functions are represented in design files as hollow-body symbols or "black box" HDL descriptions.

   ☞ Go to "Library Mapping Files (**.lmf**)" in MAX+PLUS II Help for more information.

3. Open MAX+PLUS II and specify the name of your top-level design file as the project name with the **Project Name** command (File menu). If you open an HDL file in the MAX+PLUS II Text Editor, you can choose the **Project Set Project to Current File** command (File menu) instead.

   You can also compile a project from a command line. However, the first time you compile a project,

the settings you need to specify are easier to specify from within the MAX+PLUS II software. After you have run the graphical user interface for the MAX+PLUS II software at least once, you can more easily use the command-line **setacf** utility to modify options in the Assignment & Configuration File (**.acf**) for the project. Type `setacf -h` ↵ and `maxplus2 -h` ↵ for descriptions of **setacf** and MAX+PLUS II command-line syntax.

4. Choose **Device** (Assign menu) and select the target Altera device family in the *Device Family* drop-down list box. If you wish to implement the design logic in a specific device, select it in the *Devices* box. Otherwise, select *AUTO* to allow the MAX+PLUS II Compiler to choose the best device(s) in the current device family. If your design entry or synthesis and optimization tool required you to specify a target family and/or device, specify the same information in this dialog box. For information on partitioning logic among multiple devices, go to MAX+PLUS II Help. Choose **OK**.

5. Open the Compiler window by choosing the **Compiler** command (MAX+PLUS II menu). Go through the following steps to specify the options necessary to compile the design file(s) in your project:

    1. Ensure that all EDIF netlist files have the extension **.edf** and choose **EDIF Netlist Reader Settings** (Interfaces menu).

    2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor. This name should be the name of the vendor whose tool(s) you used to create the EDIF netlist files. If your vendor name does not appear, select *Custom* instead.

        If you are compiling a design created with Synopsys FPGA Express software, select *Synopsys*, choose the **Customize** button, enter <*project name*>**.lmf** in the *LMF #1* box, choose **OK**, and skip to step 6.

    3. If you selected an existing vendor name in the *Vendor* box and your project contains design files that require custom LMF mappings, choose the **Customize** button to expand the dialog box to show all settings. Turn on the *LMF #2* checkbox and type your custom LMF's filename in the corresponding text box, or select a name from the *Files* box. The selection in the *Vendor* box will change to *Custom* and all settings will be retained until you change them again.

    4. If you selected *Custom* in the *Vendor* box, choose the **Customize** button to expand the dialog box to show all settings. Any previously defined custom settings will be displayed. Under *Signal Names*, type one or more names with up to 20 total name characters in the *VCC* or *GND* box if your EDIF Input File(s) use one or more names other than `VCC` or `GND` for the global high or low signals. Multiple signal names must be separated by either a comma (,) or a space. Under *Library Mapping Files*, turn on the *LMF #1* checkbox and type a filename in the text box following it, or select a name from the *Files* box. If necessary, specify another LMF name in the *LMF #2* box. Go to MAX+PLUS II Help for detailed information on the settings available in the **EDIF Netlist Reader Settings** dialog box.

    5. Choose **OK**.

6. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, you may need to ensure that all files are present in your project directory, i.e., the same directory as the top-level design file. Otherwise, you must specify the directories containing these files as user libraries with the **User Libraries** command (Options menu).

7. Follow all guidelines that apply to your design entry or synthesis and optimization tool:

    - Exemplar Logic Galileo Extreme-Specific Compiler Settings
    - Synopsys DesignWare-Specific Compiler Settings
    - Converting Synopsys FPGA Compiler & Design Compiler Timing Constraints into MAX+PLUS II-Compatible Format with the **syn2acf** Utility
    - Synplicity Synplify-Specific Compiler Settings

8. If you wish to generate EDIF, VHDL, or Verilog HDL output files for post-compilation simulation or timing analysis with another EDA tool, go through the following steps:

    1. (Optional) Turn on the **Optimize Timing SNF** command (Processing menu) to reduce the size of the output file(s). Turning on this command can reduce the size of output netlists by up to 30%.

        ☞ This command does not create optimized timing SNFs on UNIX workstations. However, a non-optimized timing SNF provides the same functional and timing information as an optimized timing SNF.

    2. If you wish to generate EDIF Output Files (**.edo**), go through these steps:

        1. Turn on the **EDIF Netlist Writer** command (Interfaces menu). Then choose the **EDIF Netlist Writer Settings** command (Interfaces menu).

        2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor and choose **OK**. If your vendor name does not appear, select *Custom* instead and specify the settings that are appropriate for your simulation or timing analysis tool. Go to MAX+PLUS II Help for detailed information on the options available in the **EDIF Netlist Writer Settings** dialog box.

        3. To generate an optional Standard Delay Format (SDF) Output File (**.sdo**), choose the **Customize** button to expand the dialog box to show all settings. Select one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**.

        The filenames of the EDIF Output File(s) and optional SDF Output File(s) are the same as the user-defined chip name(s) for the project; if no chip names exist, the Compiler assigns filenames that are based on the project name. For a multi-device project, the Compiler also generates a top-level EDIF Output File that is uniquely identified by "**_t**" appended to the project name. In addition, the Compiler automatically generates a VHDL Memory Model Output File, *<project name>***.vmo**, when it generates an EDIF Output File that contains memory (RAM or ROM).

    3. If you wish to generate VHDL Output Files (**.vho**), turn on the **VHDL Netlist Writer** command (Interfaces menu). Then choose **VHDL Netlist Writer Settings** command (Interfaces menu). Select *VHDL Output File (.vho)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF ver. 2.1 files contain timing delay information that allows you to perform back-annotation simulation in VHDL with VITAL-compliant simulation libraries. The VHDL Output Files generated by the Compiler have the extension **.vho**, but are otherwise named in the same way as the EDIF Output Files described above.

    4. If you wish to generate Verilog HDL Output Files (**.vo**), turn on the **Verilog Netlist Writer** command (Interfaces menu). Then choose **Verilog Netlist Writer Settings** command (Interfaces menu). Select *Verilog Output File (.vo)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF Output Files contain timing delay information that allows you to perform back-annotation simulation in Verilog HDL. The Verilog Output Files generated by the Compiler have the extension **.vo**, but are otherwise named in the same way as the EDIF Output Files described above.

9. To run the MAX+PLUS II Compiler, choose the **Project Save & Compile** command (File menu) or choose the **Start** button in the Compiler window.

    ☞ See step 3 for information on running MAX+PLUS II software from the command line.

10. Once you have compiled the project with the MAX+PLUS II Compiler, you can use the VHDL, Verilog HDL, or EDIF output file(s), and the optional SDF Output File(s) (**.sdo**) to perform timing analysis or timing simulation with another EDA tool. Specific instructions for some tools are described in these MAX+PLUS II ACCESS Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your EDA tool.

The MAX+PLUS II Compiler also generates a Report File (**.rpt**), a Pin-Out File (**.pin**), and one or more of the following files for device programming or configuration:

- JEDEC Files (**.jed**)
- Programmer Object Files (**.pof**)
- SRAM Object Files (**.sof**)
- Hexadecimal (Intel-format) Files (**.hex**)
- Tabular Text Files (**.ttf**)

Related Topics:

- Refer to the following sources for additional information:
  - Go to *Compiler Procedures* in MAX+PLUS II Help for information on other available Compiler settings.
  - Go to *Programmer Procedures* in MAX+PLUS II Help for instructions on creating other types of programming files and on programming or configuring Altera devices.
  - Go to Back-Annotating MAX+PLUS II Pin Assignments to Design Architect Symbols for information on back-annotating pin assignments in Mentor Graphics Design Architect schematics.
  - Go to Programming Altera Devices for information on the different programming hardware options for Altera device families.

- Go to the following topics, which are available on the web, for additional information:

  - MAX+PLUS II Development Software
  - Altera Programming Hardware

Synplicity Synplify-Specific Compiler Settings

If you are using the MAX+PLUS® II Compiler to compile a design that has been synthesized and optimized with Synplify software, go through the following additional compilation steps:

1. Choose **Global Project Logic Synthesis** (Assign menu) to open the **Global Project Logic Synthesis** dialog box.

2. Select the appropriate logic synthesis style under the *Global Project Synthesis Style*:

   ✔ If you turned on the *Map Logic to LCELLs* option in the Synplify **Set Device Options** dialog box when synthesizing a FLEX® device design with Synplify software, select *WYSIWYG* or *Fast* in the *Global Project Synthesis Style* box.

   or:

   ✔ If you did not turn on the *Map Logic to LCELLs* option in the Synplify **Set Device Options** dialog box when synthesizing your design with Synplify software, or if you are using a MAX® or Classic device, select *Normal* in the *Global Project Synthesis Style* box.

3. For FLEX devices, choose **Define Synthesis Style** to display the **Define Synthesis Style** dialog box. Choose **Advanced Options** to display the **Advanced Options** dialog box and turn off the *NOT Gate Push-Back* option. Choose **OK** twice to close the dialog box.

4. Choose **OK** to close the **Global Project Logic Synthesis** dialog box.

5. Continue with the steps necessary to compile your project, as described in Compiling Projects with

MAX+PLUS II Software.

Related Topics:

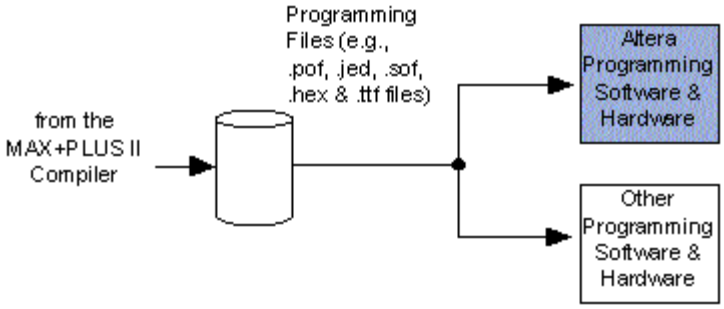
- Go to the following topics for additional information:
  - FLEX Devices
  - MAX Devices
  - Classic Device Family

Programming Altera Devices

Once you have successfully compiled and simulated a project with the MAX+PLUS® II software, you can program an Altera® device and test it in the target circuit. Figure 1 shows the device programming flow for MAX+PLUS II software.

Figure 1. MAX+PLUS II Device Programming Flow

Altera-provided items are shown in blue.



You can program devices with Altera programming hardware and MAX+PLUS II Programmer software installed on a 486- or Pentium-based PC or a UNIX workstation, or with programming hardware and software available from other manufacturers. Table 1 shows the available Altera programming hardware options on PCs and UNIX workstations.

Table 1. Altera Programming Hardware

Programming Hardware Option	UNIX PCs	Work-stations	MAX® 3000A Devices	Classic® & MAX 5000 Devices	MAX 7000 & MAX 7000E Devices	MAX 7000A, MAX 7000AE, MAX 7000B, MAX 7000S MAX 9000 & MAX 9000A Devices	FLEX® 6000, FLEX 6000A, FLEX 8000, FLEX 10K, FLEX 10KA, FLEX 10KB, & FLEX 10KE Devices	In-System Programming/ Configuration

Logic Programmer card, PL-MPU Master Programming Unit, and device-specific adapters	✓		✓	✓	✓	✓		
BitBlaster™ Download Cable	✓	✓	✓			✓	✓	✓
ByteBlasterMV™ Download Cable	✓		✓			✓	✓	✓
MasterBlaster™ Download Cable	✓	✓	✓			✓	✓	✓

If you wish to transfer programming files from a UNIX workstation to a PC over a network with File Transfer Protocol (FTP) or other similar transfer programs, be sure to select binary transfer mode.

Programming hardware from other manufacturers varies, but typically consists of a device connected to one of the serial ports on the workstation. Various vendors, such as Data I/O and BP Microsystems, supply hardware and software for programming Altera devices.

Related Topics:

- Go to Compiling Projects with MAX+PLUS II Software for information on creating programming files.
- Go to the following topics, which are available on the web, for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware
  - FLEX Devices
  - MAX Devices
  - Classic Device Family

Introduction to the ACCESS Key Guidelines

Altera established the Altera® Commitment to Cooperative Engineering Solutions (ACCESS℠) program—an alliance between Altera and EDA vendors—to provide either direct EDA support for Altera PLDs or seamless integration with Altera's MAX+PLUS® II development software.

The MAX+PLUS II ACCESS Key Guidelines provide complete instructions on how to create, compile, and simulate your design with a combination of tools from leading EDA vendors and the MAX+PLUS II software.

Click one of the following icons to go to the guidelines:

List by Vendor

List by Tool

List by Function

Related Links

- ACCESS Program
- MAX+PLUS II Development Software
- Synopsys

Altera-Provided Logic & Symbol Libraries

The MAX+PLUS® II/Cadence environment provides four logic and symbol libraries that are used for compiling, synthesizing, and simulating designs.

You can create your own libraries of custom symbols and logic functions in Concept and Composer. You can use custom symbols to incorporate an EDIF Input File, Text Design File (TDF), or any other MAX+PLUS II-supported design file into a project. MAX+PLUS II uses the **cadence.lmf** Library Mapping File to map standard Concept or Composer symbols to equivalent MAX+PLUS II logic functions. To use custom symbols, you can create a custom LMF that maps your custom symbols to the equivalent MAX+PLUS II-supported design file. You must also specify the directory that contains the MAX+PLUS II-supported design file(s) as a user library with the MAX+PLUS II **User Libraries** command (Options menu). Go to "Library Mapping File" and "Cadence Library Mapping File (**cadence.lmf**)" in MAX+PLUS II Help for more information.

The alt_max2 Library

You can enter a Concept or Composer Design Architect schematic with primitives and macrofunctions from the Altera-provided symbol library **alt_max2**. The **alt_max2** library includes 74-series macrofunctions and several MAX+PLUS II primitives with corresponding Verilog HDL simulation models for controlling design synthesis and fitting. It also includes four macrofunctions--`a_8count`, `a_8mcomp`, `a_8fadd`, and `a_81mux`--that are optimized for different device families, and the `clklock` phase-locked loop megafunction, which is supported by some FLEX® 10K devices, with corresponding Verilog HDL and VHDL simulation models. See Table 1. Choose **Old-Style Macrofunctions** and/or **Primitives** from the MAX+PLUS II Help menu for more information on functions in the **alt_max2** library.

The alt_lpm Library

The Altera-provided **alt_lpm** library, which is available for Concept and Verilog HDL designs, includes standard functions from the library of parameterized modules (LPM) 2.1.0, except the truth table, finite state machine, and pad functions. Other parameterized functions, including cycle-shared FIFO (`csfifo`) and cycle-shared dual-port RAM (`csdpram`) are also included. The LPM standard defines a set of parameterized modules (i.e., parameterized megafunctions) and their corresponding representations in an EDIF netlist file. These logic functions allow you to create and functionally simulate an LPM-based design without targeting a specific device family. The parameters you specify for each LPM function determine the simulation models that will be generated. After the design is completed, you can target the design to any device family. In designs created with Concept, the Altera **alt_lpm** library works only with HDL Direct and the **hdlconfig** utility. Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu for more information about LPM functions in the **alt_lpm** library.

The lpm_syn Library

The **lpm_syn** library contains the Altera-provided parameterized functions. The **lpm_syn** library is similar to the **alt_lpm** library, except that it contains VHDL and Verilog HDL logic functions for use with Synergy, Concept, and Composer software.

The alt_mf Library

Altera provides a VHDL logic function library, **alt_mf**, that currently includes four macrofunctions--`a_8count`,

a_8mcomp, a_8fadd, and a_81mux--for controlling design synthesis and fitting. These elements can be instantiated directly in your VHDL file. To designate that these logic functions should pass untouched through the EDIF netlist file to the MAX+PLUS II Compiler, you must select the *Maintain* attribute constraint for instances of these functions before running the Synergy software. These models allow you to perform functional VHDL simulation while maintaining an architecture-independent VHDL description.

Table 1 shows the MAX+PLUS II-specific logic functions.

Table 1. MAX+PLUS II-Specific Logic Functions

Macrofunctions *Note (1)*				**Primitives**	
Name	**Description**	**Name**	**Description**	**Name**	**Description**
8fadd	8-bit full adder	LCELL	Logic cell buffer	EXP	MAX® 5000, MAX 7000, and MAX 9000 Expander buffer
8mcomp	8-bit magnitude comparator	GLOBAL	Global input buffer	SOFT	Soft buffer
8count *Note (2)*	8-bit up/down counter	CASCADE	FLEX 6000, FLEX 8000, and FLEX 10K cascade buffer	OPNDRN	Open-drain buffer
81mux	8-to-1 multiplexer	CARRY	FLEX 6000, FLEX 8000, and FLEX 10K carry buffer	DFFE DFFE6K *Note (3)*	D-type flipflop with Clock Enable
clklock	Phase-locked loop				

Notes:

1. Logic function names that begin with a number must be preceded by "a_" in VHDL designs. For example, 8fadd must be specified as a_8fadd.
2. The a_8count logic function is for the MAX 7000 and MAX 9000 device families only.
3. For designs that are targeted to FLEX 6000 devices, you should use the DFFE primitive only if the design contains either a Clear or Preset signal, but not both. If your design contains both a Clear and a Preset signal, you must use the DFFE6K primitive.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Using Cadence Synergy & MAX+PLUS II Software

The following topics describe how to use the Cadence Synergy software with MAX+PLUS® II software. Click on one of the following topics for information:

This file is suitable for printing only. It does not contain hypertext links that allow you to jump from topic to topic.

Setting Up the MAX+PLUS II/Cadence Working Environment

- Software Requirements
- MAX+PLUS II/Cadence Interface File Organization
- Altera-Provided Logic & Symbol Libraries

Design Entry

- **Design Entry Flow**
- **Creating VHDL Projects**
  - Creating VHDL Designs for Use with MAX+PLUS II Software
    - Instantiating the clklock Megafunction in VHDL or Verilog HDL
  - Entering Resource Assignments
    - Modifying the Assignment & Configuration File with the **setacf** Utility
- **Creating Verilog HDL Projects**
  - Creating Verilog HDL Designs for Use with MAX+PLUS II Software
    - Instantiating the clklock Megafunction in VHDL or Verilog HDL
  - Entering Resource Assignments
    - Modifying the Assignment & Configuration File with the **setacf** Utility

Synthesis & Optimization

- **VHDL**
  - Synthesizing & Optimizing VHDL Files with Synergy Software
  - Converting VHDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the **vlog2alt** or **altout** Utility
- **Verilog HDL**
  - Synthesizing & Optimizing Verilog HDL Files with Synergy Software
  - Converting Verilog HDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the

vlog2alt Utility

Related Topics:

Go to the following MAX+PLUS II ACCESS℠ Key topics for related information:

- Compiling Projects with MAX+PLUS II Software
- Programming Altera Devices

Go to the following topics, which are available on the web, for additional information:

- MAX+PLUS II Development Software
- Altera Programming Hardware
- Cadence web site (http://www.cadence.com)

Setting Up the MAX+PLUS II/Cadence Working Environment

To use MAX+PLUS® II software with Cadence software, you must first install the MAX+PLUS II software, then establish an environment that facilitates entering and processing designs. The MAX+PLUS II/Cadence interface is installed automatically when you install the MAX+PLUS II software on your computer. Go to *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual for more information on installation and details on the directories that are created during MAX+PLUS II installation. Go to MAX+PLUS II/Cadence Interface File Organization for information about the MAX+PLUS II/Cadence directories that are created during MAX+PLUS II installation.

The information presented here assumes that you are using the C shell and that your MAX+PLUS II system directory is **/usr/maxplus2**. If not, you must use the appropriate syntax and procedures to set environment variables for your shell.

To set up your working environment for the MAX+PLUS II/Cadence interface, follow these steps:

1. Ensure that you have correctly installed the MAX+PLUS II and Cadence software versions described in the MAX+PLUS II/Cadence Software Requirements.
2. Add the following environment variables to your **.cshrc** file:

   ```
   setenv ALT_HOME /usr/maxplus2 ↵

   setenv CDS_INST_DIR <Cadence system directory path> ↵
   ```

3. Add the **$ALT_HOME/cadence/bin** and **$CDS_INST_DIR/tools/bin** directories to the `PATH` environment variable in your **.cshrc** file. Make sure these paths are placed before the Cadence hierarchy path.
4. Add **/usr/dt/lib** and **/usr/ucb/lib** to the `LD_LIBRARY_PATH` environment variable in your **.cshrc** file.
5. Create a new **cds.lib** file in your working directory or edit an existing one so that it includes all of the following lines that apply to the Cadence tools you have installed:

   ```
   DEFINE alt_syn ${ALT_HOME}/simlib/concept/alt_syn

   DEFINE lpm_syn ${ALT_HOME}/simlib/concept/lpm_syn

   DEFINE alt_lpm ${ALT_HOME}/simlib/concept/alt_lpm

   DEFINE alt_mf ${ALT_HOME}/simlib/concept/alt_mf
   ```

```
DEFINE alt_max2 ${ALT_HOME}/simlib/concept/alt_max2

DEFINE alt_max2 ${ALT_HOME}/simlib/composer/alt_max2/alt_max2

DEFINE alt_vtl $ALT_HOME/simlib/concept/alt_vtl/lib

DEFINE altera $ALT_HOME/simlib/concept/alt_mf/lib

SOFTINCLUDE $CDS_INST_DIR/tools/leapfrog/files/cds.lib

DEFINE <design name>.
```

6. Copy the **/usr/maxplus2/maxplus2.ini** file to your **$HOME** directory:

```
cp /usr/maxplus2/maxplus2.ini $HOME ↵

chmod u+w $HOME/maxplus2.ini ↵
```

The **maxplus2.ini** file contains both Altera- and user-specified initialization parameters that control the MAX+PLUS II software, such as MAX+PLUS II symbol and logic function library paths and the current project name. The MAX+PLUS II installation procedure creates and copies the **maxplus2.ini** file to the **/usr/maxplus2** directory.

Normally, you do not have to edit your local copy of **maxplus2.ini** because the MAX+PLUS II software updates the file automatically whenever you change any parameters or settings. However, if you move the **max2lib** and **max2inc** library subdirectories, you must update the file. Go to "Creating & Using a Local Copy of the **maxplus2.ini** File" in MAX+PLUS II Help for more information.

7. If you are using Concept on a Sun SPARCstation running SunOS, go to Setting Up the MAX+PLUS II/Cadence Concept Work Environment for a Sun SPARCstation Running SunOS Software to install the **redifnet** EDIF netlist reader utility.

8. If you are using Synergy software, edit the **hdl.var** file located in your working directory to include the following line:

```
DEFINE work <design name> ↵
```

9. Set up an appropriate directory structure for the tool(s) you are using. See the following topics for information:

    - Composer Project File Directory Structure
    - Concept & RapidSIM Local Work Area Directory Structure

Related Topics:

Go to the following topics, which are available on the web, for additional information:

- ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
- This manual is also available in 4 parts:
    - Preface & Section 1: MAX+PLUS II Installation
    - Section 2: MAX+PLUS II - A Perspective
    - Section 3: MAX+PLUS II Tutorial
    - Appendices, Glossary & Index

MAX+PLUS II/Cadence Software Requirements

The following table shows the software applications that are used to generate, process, synthesize, and verify a project with MAX+PLUS® II and Cadence software:

Cadence		Altera
version 97A: Concept Composer ValidCOMPILER **concept2alt** **vlog2alt** **altout**	VerilogLink Synergy HDL Direct (Concept 2.0 or later) Non-Graphic Simulation Environment (SE) RapidSIM, Verilog-XL, or Leapfrog **redifnet** (SunOS only)	MAX+PLUS II version 9.4

☞ The MAX+PLUS II **read.me** file provides up-to-date information on which versions of Cadence software applications are supported by the current version of MAX+PLUS II. It also provides information on installation and operating requirements. You should read the **read.me** file on the CD-ROM before installing the MAX+PLUS II software. After installation, you can open the **read.me** file from the MAX+PLUS II Help menu.

MAX+PLUS II/Cadence Interface File Organization

Table 1 shows the MAX+PLUS® II/Cadence interface subdirectories that are created in the MAX+PLUS II system directory (by default, the **/usr/maxplus2** directory) during MAX+PLUS II installation. For information on the other directories that are created during MAX+PLUS II installation, see "MAX+PLUS II File Organization" in *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual.

Table 1. MAX+PLUS II Directory Organization

Directory	Description
./lmf	Contains the Altera-provided Library Mapping File, **cadence.lmf**, that maps Cadence logic functions to equivalent MAX+PLUS II logic functions.
./examples/cadence	Contains the sample files for Cadence software discussed in these ACCESS℠ Key Guidelines.
./cadence	Contains the AMPLE userware for the MAX+PLUS II/Cadence interface.
./simlib/concept/alt_max2	Contains the MAX+PLUS II primitives, including `CARRY`, `CASCADE`, `EXP`, `GLOBAL`, `LCELL`, `SOFT`, `OPNDRN`, `DFFE` (D flipflop with Clock Enable), and `DFFE6K` (D flipflop with Clock Enable and both Clear and Preset for FLEX® 6000 devices only) for use with Concept software.
./simlib/composer/alt_max2	Contains the MAX+PLUS II primitives, including `CARRY`, `CASCADE`, `EXP`, `GLOBAL`, `LCELL`, `SOFT`, `OPNDRN`, `DFFE` (D flipflop with Clock Enable), and `DFFE6K` (D flipflop with Clock Enable and both Clear and Preset for FLEX 6000 devices only) for use with Composer software.
./simlib/concept/alt_lpm	Contains the MAX+PLUS II megafunctions, including library of parameterized modules (LPM) functions, for use with Concept software.
./simlib/concept/max2sim	Contains the MAX+PLUS II/Concept simulation model library, **max2_sim**, for use with RapidSIM software.
./simlib/concept/alt_syn	Contains the MAX+PLUS II synthesis library, **alt_syn**, for use with Synergy and Concept software, and the **vlog2alt** utility.

./simlib/composer/alt_syn	Contains the MAX+PLUS II synthesis library, **alt_syn**, for use with Synergy and Composer software.
./simlib/concept/lpm_syn	Contains the Cadence LPM library, **lpm_syn**, for use with Synergy and Concept software.
./simlib/composer/lpm_syn	Contains the Cadence LPM library, **lpm_syn**, for use with Synergy and Composer software.
./simlib/concept/alt_mf	Contains the MAX+PLUS II VHDL logic function library. (`a_8count` is for the MAX® 7000 and MAX 9000 device families only.)
./simlib/concept/edifnet/templates	Contains template files for Concept directives, i.e., **global.cmd**, **compiler.cmd**, **vloglink.cmd**, **verilog.cmd**, and **master.local**.
./simlib/concept/alt_max2/verilogUdps	Contains Verilog HDL modules that are the equivalent of the primitives contained in **alt_max2** library for use with Concept software.
./simlib/composer/alt_max2/verilogUdps	Contains Verilog HDL modules that are the equivalent of the primitives contained in **alt_max2** library for use with Composer software.
./simlib/concept/alt_vtl **./simlib/composer/alt_vtl**	Contains VITAL library source files for use with Concept or Composer software.
./simlib/composer/alt_max2/verilog	Contains simulation modules for all symbols in the **alt_max2** Composer library.

Related Topics:

Go to the following topics, which are available on the web, for additional information:

- ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
- This manual is also available in 4 parts:
  - Preface & Section 1: MAX+PLUS II Installation
  - Section 2: MAX+PLUS II - A Perspective
  - Section 3: MAX+PLUS II Tutorial
  - Appendices, Glossary & Index
- FLEX Devices
- MAX Devices
- Classic Device Family

Altera-Provided Logic & Symbol Libraries

The MAX+PLUS® II/Cadence environment provides four logic and symbol libraries that are used for compiling, synthesizing, and simulating designs.

You can create your own libraries of custom symbols and logic functions in Concept and Composer. You can use custom symbols to incorporate an EDIF Input File, Text Design File (TDF), or any other MAX+PLUS II-supported design file into a project. MAX+PLUS II uses the **cadence.lmf** Library Mapping File to map standard Concept or Composer symbols to equivalent MAX+PLUS II logic functions. To use custom symbols, you can create a custom LMF that maps your custom symbols to the equivalent MAX+PLUS II-supported design file. You must also specify the directory that contains the MAX+PLUS II-supported design file(s) as a user library with the MAX+PLUS II **User Libraries** command (Options menu). Go to "Library Mapping File" and "Cadence Library Mapping File (**cadence.lmf**)" in MAX+PLUS II Help for more information.

The alt_max2 Library

You can enter a Concept or Composer Design Architect schematic with primitives and macrofunctions from the

Altera-provided symbol library **alt_max2**. The **alt_max2** library includes 74-series macrofunctions and several MAX+PLUS II primitives with corresponding Verilog HDL simulation models for controlling design synthesis and fitting. It also includes four macrofunctions--a_8count, a_8mcomp, a_8fadd, and a_81mux--that are optimized for different device families, and the clklock phase-locked loop megafunction, which is supported by some FLEX® 10K devices, with corresponding Verilog HDL and VHDL simulation models. See Table 1. Choose **Old-Style Macrofunctions** and/or **Primitives** from the MAX+PLUS II Help menu for more information on functions in the **alt_max2** library.

The alt_lpm Library

The Altera-provided **alt_lpm** library, which is available for Concept and Verilog HDL designs, includes standard functions from the library of parameterized modules (LPM) 2.1.0, except the truth table, finite state machine, and pad functions. Other parameterized functions, including cycle-shared FIFO (csfifo) and cycle-shared dual-port RAM (csdpram) are also included. The LPM standard defines a set of parameterized modules (i.e., parameterized megafunctions) and their corresponding representations in an EDIF netlist file. These logic functions allow you to create and functionally simulate an LPM-based design without targeting a specific device family. The parameters you specify for each LPM function determine the simulation models that will be generated. After the design is completed, you can target the design to any device family. In designs created with Concept, the Altera **alt_lpm** library works only with HDL Direct and the **hdlconfig** utility. Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu for more information about LPM functions in the **alt_lpm** library.

The lpm_syn Library

The **lpm_syn** library contains the Altera-provided parameterized functions. The **lpm_syn** library is similar to the **alt_lpm** library, except that it contains VHDL and Verilog HDL logic functions for use with Synergy, Concept, and Composer software.

The alt_mf Library

Altera provides a VHDL logic function library, **alt_mf**, that currently includes four macrofunctions--a_8count, a_8mcomp, a_8fadd, and a_81mux--for controlling design synthesis and fitting. These elements can be instantiated directly in your VHDL file. To designate that these logic functions should pass untouched through the EDIF netlist file to the MAX+PLUS II Compiler, you must select the *Maintain* attribute constraint for instances of these functions before running the Synergy software. These models allow you to perform functional VHDL simulation while maintaining an architecture-independent VHDL description.

Table 1 shows the MAX+PLUS II-specific logic functions.

Table 1. MAX+PLUS II-Specific Logic Functions

Macrofunctions *Note (1)*		**Primitives**			
Name	**Description**	**Name**	**Description**	**Name**	**Description**
8fadd	8-bit full adder	LCELL	Logic cell buffer	EXP	MAX® 5000, MAX 7000, and MAX 9000 Expander buffer
8mcomp	8-bit magnitude comparator	GLOBAL	Global input buffer	SOFT	Soft buffer
8count *Note (2)*	8-bit up/down counter	CASCADE	FLEX 6000, FLEX 8000, and FLEX 10K cascade buffer	OPNDRN	Open-drain buffer
81mux	8-to-1 multiplexer	CARRY	FLEX 6000, FLEX 8000, and FLEX 10K carry buffer	DFFE DFFE6K *Note (3)*	D-type flipflop with Clock Enable
clklock	Phase-locked loop				

Notes:

1. Logic function names that begin with a number must be preceded by "a_" in VHDL designs. For example, 8fadd must be specified as a_8fadd.
2. The a_8count logic function is for the MAX 7000 and MAX 9000 device families only.
3. For designs that are targeted to FLEX 6000 devices, you should use the DFFE primitive only if the design contains either a Clear or Preset signal, but not both. If your design contains both a Clear and a Preset signal, you must use the DFFE6K primitive.

Related Topics:

Go to the following topics, which are available on the web, for additional information:

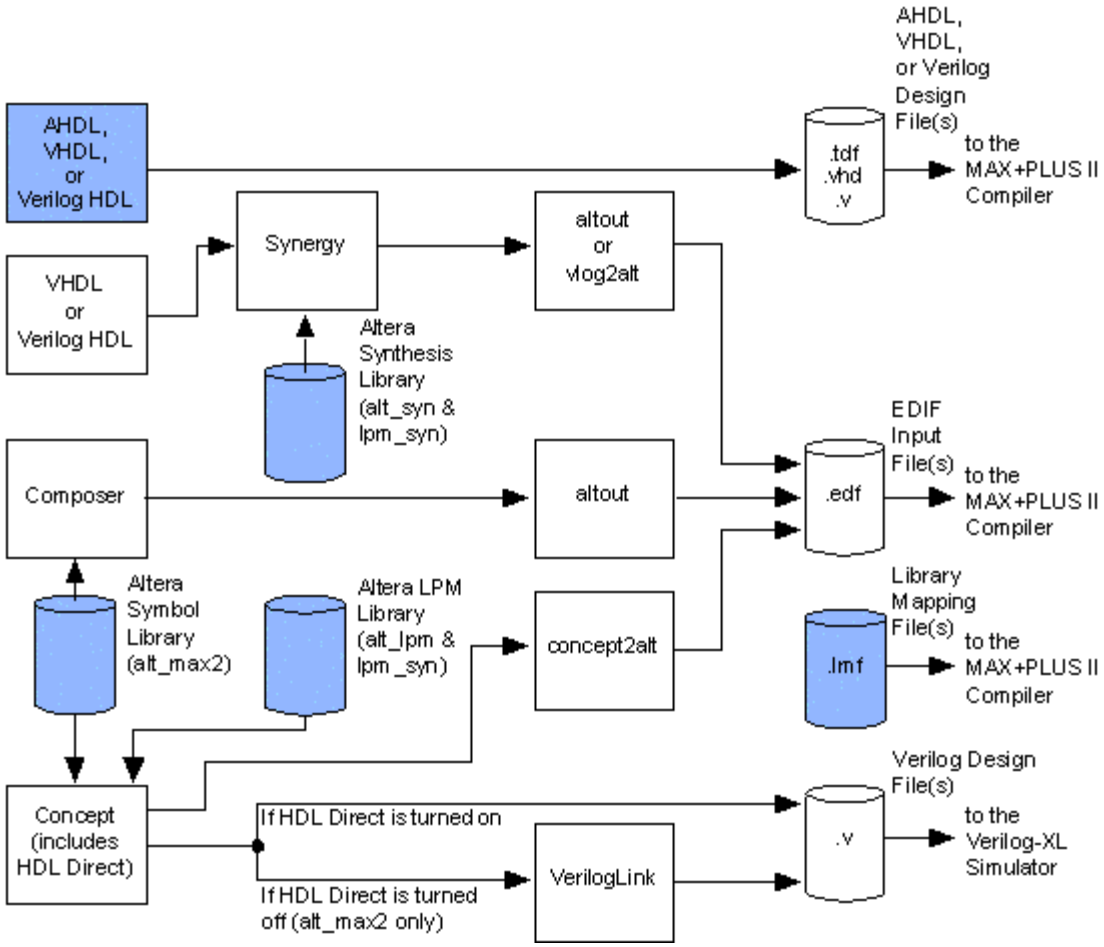
- FLEX Devices
- MAX Devices
- Classic Device Family

Cadence Design Entry Flow

Figure 1 shows the design entry flow for the MAX+PLUS® II/Cadence interface.

Figure 1. MAX+PLUS II/Cadence Design Entry Flow

Altera-provided items are shown in blue.



Creating VHDL Designs for Use with MAX+PLUS II Software

You can create VHDL design files with the MAX+PLUS® II Text Editor or another standard text editor and save them in the appropriate directory for your project. The MAX+PLUS II Text Editor offers the following advantages:

- VHDL templates are available with the **VHDL Templates** command (Templates menu). These templates are also available in the ASCII **vhdl.tmp** file, which is located in the **/usr/maxplus2** directory.
- If you use the MAX+PLUS II Text Editor to create your VHDL design, you can use the **Syntax Coloring** command (Options menu). The Syntax Coloring feature displays keywords and other elements in text files in different colors to distinguish them from other forms of syntax.

To create a VHDL design that can be synthesized and optimized with Synergy software, follow these steps:

1. You can instantiate the following MAX+PLUS II-provided logic functions in your VHDL design:
    - The **alt_mf** library contains the Altera® VHDL logic function library, which includes the `a_8count`, `a_8mcomp`, `a_8fadd`, and `a_81mux` macrofunctions. If you wish to instantiate **alt_mf** logic functions in your VHDL design, you must first compile this library, as described in Compiling the **alt_mf** Library.
    - The `clklock` megafunction, which enables the phase-locked loop, or ClockLock , circuitry available on selected Altera FLEX® 10K devices. Go to Instantiating the `clklock` Megafunction in VHDL or Verilog HDL for information.
    - MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile,

and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.

2. If you wish to use Standard Delay Format (SDF) Output Files (**.sdo**) that contain timing information when performing post-compilation timing simulation with Leapfrog software, you must first compile the VITAL library source files, as described in Compiling the **alt_vtl** Library for for Use with Leapfrog Software.

3. (Optional) To enter resource assignments in your VHDL design, go to Entering Resource Assignments. You can also enter resource assignments from within the MAX+PLUS II software.

4. After you have completed your VHDL design, synthesize and optimize it with Synergy software, as described in Synthesizing & Optimizing VHDL Files with Synergy Software.

Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample VHDL files, the latter of which includes macrofunction instantiation.

- **/usr/maxplus2/examples/cadence/example9/count4.vhd**
- **/usr/maxplus2/examples/cadence/example10/adder16.vhd**

Related Topics:

- Go to FLEX 10K Device Family, which is available on the web, for additional information.

Instantiating the clklock Megafunction in VHDL or Verilog HDL

MAX+PLUS® II interfaces to other EDA tools support the `clklock` phase-locked loop megafunction, which can be used with some FLEX® 10K devices, with the **gencklk** utility, which is available in the MAX+PLUS II system directory. Type `gencklk -h` ↵ at the DOS or UNIX prompt to display information on how to use this utility. The **gencklk** utility generates VHDL or Verilog HDL functional simulation models and a VHDL Component Declaration template file (**.cmp**).

The **gencklk** utility allows parameters for the `clklock` function to be passed from the VHDL or Verilog HDL file to EDIF netlist format. The **gencklk** utility embeds the parameter values in the `clklock` function name; therefore, the values do not need to be declared explicitly.

To instantiate the `clklock` megafunction in VHDL or Verilog HDL, go through the following steps:

1. Type the following command at the DOS or UNIX prompt to generate the `clklock_`*x*`_`*y* function, where *x* is the ClockBoost value and *y* is the input frequency in MHz:

   ✔ Type `gencklk` <*ClockBoost*> <*input frequency*> `-vhdl` ↵ for VHDL designs.

   or:

   ✔ Type `gencklk` <*ClockBoost*> <*input frequency*> `-verilog` ↵ for Verilog HDL designs.

   Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu for more information on the `clklock` megafunction.

2. Create a design file that instantiates the `clklock_`*x*`_`*y***.vhd** or `clklock_`*x*`_`*y***.v** file. The **gencklk** utility automatically generates a VHDL Component Declaration template in the **clklock_*x*_*y*.cmp** file that you can incorporate into a VHDL design file.

☞ In MAX+PLUS II version 8.3 and lower, running `genclklk` on a PC always creates files named as **clklock.vhd**,

clklock.cmp, and **clklock.v**, regardless of the ClockBoost and input frequency values you specify.

Figures 1 and 2 show a `clklock` function with <*ClockBoost*> = 2 and <*input frequency*> = 40 MHz instantiated in VHDL and Verilog HDL design files, respectively.

Figure 1. VHDL Design File with clklock Instantiation (count8.vhd)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY altera;
USE altera.maxplus2.all;     --  Include Altera Component Declarations



ENTITY count8 IS
   PORT (a    : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
         ldn  : IN STD_LOGIC;
         gn   : IN STD_LOGIC;



dnup : IN STD_LOGIC;
         setn : IN STD_LOGIC;
         clrn : IN STD_LOGIC;
         clk  : IN STD_LOGIC;



co   : OUT STD_LOGIC;
         q    : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END count8;



ARCHITECTURE structure OF count8 IS
   signal clk2x : STD_LOGIC;



COMPONENT clklock_2_40
   PORT (
      INCLK : IN STD_LOGIC;
      OUTCLK : OUT STD_LOGIC
   );
END COMPONENT;



BEGIN
   u1: clklock_2_40
   PORT MAP (inclk=>clk, outclk=>clk2x);



u2: a_8count
   PORT MAP (a=>a(0), b=>a(1), c=>a(2), d=>a(3),
             e=>a(4), f=>a(5), g=>a(6), h=>a(7),
             clk=>clk2x,
             ldn=>ldn,
             gn=>gn,



dnup=>dnup,
             setn=>setn,
             clrn=>clrn,



qa=>q(0), qb=>q(1), qc=>q(2), qd=>q(3),
             qe=>q(4), qf=>q(5), qg=>q(6), qh=>q(7),
             cout=>co);
 END structure;
```

Figure 2. Verilog HDL Design File with clklock Instantiation (count8.v)

```
`timescale 1ns / 10ps
module count8 (a, ldn, gn, dnup, setn, clrn, clk, co, q);
output          co;
output[7:0]     q;



input[7:0]      a;
input           ldn, gn,dnup, setn, clrn, clk;
wire            clk2x;



clklock_2_40 u1 (.inclk(clk), .outclk(clk2x) );
A_8COUNT u2 (.A(a[0]), .B(a[1]), .C(a[2]), .D(a[3]), .E(a[4]), .F(a[5]),



.G(a[6]), .H(a[7]), .LDN(ldn), .GN(gn), .DNUP(dnup),
             .SETN(setn), .CLRN(clrn), .CLK(clk2x), .QA(q[0]), .QB(q[1]),
             .QC(q[2]), .QD(q[3]), .QE(q[4]), .QF(q[5]), .QG(q[6]),
             .QH(q[7]), .COUT(co) );



endmodule
```

Related Topics:

- Go to FLEX 10K Device Family, which is available on the web, for additional information.

Entering Resource Assignments

The MAX+PLUS® II software allows you to enter a variety of resource and device assignments for your projects. Resource assignments are used to assign logic functions to a particular pin, logic cell, I/O cell, embedded cell, row, column, Logic Array Block (LAB), Embedded Array Block (EAB), chip, clique, local routing, logic option, timing requirement, or connected pin group. In MAX+PLUS II software, you can enter all types of resource and device assignments with Assign menu commands. You can also enter pin, logic cell, I/O cell, embedded cell, LAB, EAB, row, and column assignments in the MAX+PLUS II Floorplan Editor. The Assign menu commands and the Floorplan Editor all save assignment information in the ASCII Assignment & Configuration File (**.acf**) for the project. In addition, you can edit ACFs manually in any standard text editor or with the **setacf** utility.

Concept & Composer Schematics

In both Concept and Composer schematics, you can assign a limited subset of these resource assignments by assigning properties to symbols. These properties are incorporated into the EDIF netlist file(s). The MAX+PLUS II software automatically converts assignment information from the EDIF Input File into the ACF format. For information on making MAX+PLUS II-compatible resource assignments, go to the following topics:

- Assigning Pins, Logic Cells & Chips
- Assigning Cliques
- Assigning Logic Options
- Modifying the Assignment & Configuration File with the **setacf** Utility

Go to the ***Cadence Concept Schematic User Guide*** and ***Composer Reference User Guide*** for details on how to

assign properties. Go to "Resource Assignments in EDIF Input Files" and "Assigning Resources in a Third-Party Design Editor" in MAX+PLUS II Help for more information on assignments or properties that can be assigned in Concept and Composer.

Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample Concept and Composer schematic files, which include resource assignments:

- **/usr/maxplus2/examples/cadence/example6/fa2** (Concept)
- **/usr/maxplus2/examples/cadence/example7/fa2** (Composer)

VHDL & Verilog HDL Design Files

For Verilog HDL- and VHDL-based designs, you must use the MAX+PLUS II software or the **setacf** utility to enter resource assignments. For information on using the **setacf** utility, go to Modifying the Assignment & Configuration File with the **setacf** Utility.

Related Topics:

- For information on entering assignments in the MAX+PLUS II software with Assign menu commands or in an ACF, go to "resource assignments" or "ACF, format" in MAX+PLUS II Help using **Search for Help on** (Help menu).

Modifying the Assignment & Configuration File with the setacf Utility

Altera provides the **setacf** utility to help you modify a project's Assignment & Configuration File (**.acf**) from the command line, without opening the file with a text editor. Type `setacf -h` ↵ at a UNIX or DOS prompt to get help on this utility.

Creating Verilog HDL Designs for Use with MAX+PLUS II Software

You can create Verilog HDL design files with the MAX+PLUS® II Text Editor or another standard text editor and save them in the appropriate directory for you project. The MAX+PLUS II Text Editor offers the following advantages:

- Verilog HDL templates are available with the **Verilog Templates** command (Templates menu). These templates are also available in the ASCII **verilog.tmp** file, which is located in the **/usr/maxplus2** directory.
- If you use the MAX+PLUS II Text Editor to create your Verilog HDL design, you can use the **Syntax Coloring** command (Options menu). The Syntax Coloring feature displays keywords and other elements of text in text files in different colors to distinguish them from other forms of syntax.

To create a Verilog HDL design that can be synthesized and optimized with Synergy software, go through the following steps:

1. You can instantiate the following MAX+PLUS II-provided logic functions in your Verilog HDL design:
    - The **alt_max2** library, which contains the `a_8count`, `a_8mcomp`, `a_8fadd`, and `a_81mux` macrofunctions that are optimized for different Altera device families.
    - The `clklock` megafunction which enables phase-locked loop, or ClockLock , circuitry available on selected Altera FLEX® 10K devices. Go to Instantiating the `clklock` Megafunction in VHDL or Verilog HDL for information.

- The **lpm_syn** library, which contains the Cadence LPM megafunction library for use with Synergy Software and Concept or Composer software.

- MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.

2. You can enter resource assignments in your Verilog HDL design, as described in Entering Resource Assignments.

3. After you have completed your Verilog HDL design, synthesize and optimize it with Synergy software, as described in Synthesizing & Optimizing Verilog HDL Files with Synergy Software.

Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample Verilog HDL files, the latter of which includes LPM function instantiation.

- **/usr/maxplus2/examples/cadence/example11/count8.v**
- **/usr/maxplus2/examples/cadence/example13/rom_test.v**

Related Topics:

- Go to FLEX 10K Device Family, which is available on the web, for additional information.

Instantiating the clklock Megafunction in VHDL or Verilog HDL

MAX+PLUS® II interfaces to other EDA tools support the `clklock` phase-locked loop megafunction, which can be used with some FLEX® 10K devices, with the **gencklk** utility, which is available in the MAX+PLUS II system directory. Type `gencklk -h` ↵ at the DOS or UNIX prompt to display information on how to use this utility. The **gencklk** utility generates VHDL or Verilog HDL functional simulation models and a VHDL Component Declaration template file (**.cmp**).

The **gencklk** utility allows parameters for the `clklock` function to be passed from the VHDL or Verilog HDL file to EDIF netlist format. The **gencklk** utility embeds the parameter values in the `clklock` function name; therefore, the values do not need to be declared explicitly.

To instantiate the `clklock` megafunction in VHDL or Verilog HDL, go through the following steps:

1. Type the following command at the DOS or UNIX prompt to generate the `clklock_x_y` function, where *x* is the ClockBoost value and *y* is the input frequency in MHz:

   ✔ Type `gencklk` <*ClockBoost*> <*input frequency*> `-vhdl` ↵ for VHDL designs.

   or:

   ✔ Type `gencklk` <*ClockBoost*> <*input frequency*> `-verilog` ↵ for Verilog HDL designs.

   Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu for more information on the `clklock` megafunction.

2. Create a design file that instantiates the `clklock_x_y`**.vhd** or `clklock_x_y`**.v** file. The **gencklk** utility automatically generates a VHDL Component Declaration template in the **clklock_*x_y*.cmp** file that you can incorporate into a VHDL design file.

☞ In MAX+PLUS II version 8.3 and lower, running `genclklk` on a PC always creates files named as **clklock.vhd**, **clklock.cmp**, and **clklock.v**, regardless of the ClockBoost and input frequency values you specify.

Figures 1 and 2 show a `clklock` function with <*ClockBoost*> = 2 and <*input frequency*> = 40 MHz instantiated in VHDL and Verilog HDL design files, respectively.

Figure 1. VHDL Design File with clklock Instantiation (count8.vhd)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY altera;
USE altera.maxplus2.all;     --  Include Altera Component Declarations



ENTITY count8 IS
   PORT (a    : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
         ldn  : IN STD_LOGIC;
         gn   : IN STD_LOGIC;



dnup : IN STD_LOGIC;
         setn : IN STD_LOGIC;
         clrn : IN STD_LOGIC;
         clk  : IN STD_LOGIC;



co   : OUT STD_LOGIC;
         q    : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END count8;



ARCHITECTURE structure OF count8 IS
   signal clk2x : STD_LOGIC;



COMPONENT clklock_2_40
   PORT (
      INCLK : IN STD_LOGIC;
      OUTCLK : OUT STD_LOGIC
   );
END COMPONENT;



BEGIN
   u1: clklock_2_40
   PORT MAP (inclk=>clk, outclk=>clk2x);



u2: a_8count
   PORT MAP (a=>a(0), b=>a(1), c=>a(2), d=>a(3),
             e=>a(4), f=>a(5), g=>a(6), h=>a(7),
             clk=>clk2x,
             ldn=>ldn,
             gn=>gn,



dnup=>dnup,
             setn=>setn,
             clrn=>clrn,



qa=>q(0), qb=>q(1), qc=>q(2), qd=>q(3),
             qe=>q(4), qf=>q(5), qg=>q(6), qh=>q(7),
```

```
            cout=>co);
 END structure;
```

Figure 2. Verilog HDL Design File with clklock Instantiation (count8.v)

```
`timescale 1ns / 10ps
module count8 (a, ldn, gn, dnup, setn, clrn, clk, co, q);
output          co;
output[7:0]     q;



input[7:0]      a;
input           ldn, gn,dnup, setn, clrn, clk;
wire            clk2x;



clklock_2_40 u1 (.inclk(clk), .outclk(clk2x) );
A_8COUNT u2 (.A(a[0]), .B(a[1]), .C(a[2]), .D(a[3]), .E(a[4]), .F(a[5]),



.G(a[6]), .H(a[7]), .LDN(ldn), .GN(gn), .DNUP(dnup),
             .SETN(setn), .CLRN(clrn), .CLK(clk2x), .QA(q[0]), .QB(q[1]),
             .QC(q[2]), .QD(q[3]), .QE(q[4]), .QF(q[5]), .QG(q[6]),
             .QH(q[7]), .COUT(co) );



endmodule
```

- Go to FLEX 10K Device Family, which is available on the web, for additional information.

Entering Resource Assignments

The MAX+PLUS® II software allows you to enter a variety of resource and device assignments for your projects. Resource assignments are used to assign logic functions to a particular pin, logic cell, I/O cell, embedded cell, row, column, Logic Array Block (LAB), Embedded Array Block (EAB), chip, clique, local routing, logic option, timing requirement, or connected pin group. In MAX+PLUS II software, you can enter all types of resource and device assignments with Assign menu commands. You can also enter pin, logic cell, I/O cell, embedded cell, LAB, EAB, row, and column assignments in the MAX+PLUS II Floorplan Editor. The Assign menu commands and the Floorplan Editor all save assignment information in the ASCII Assignment & Configuration File (**.acf**) for the project. In addition, you can edit ACFs manually in any standard text editor or with the **setacf** utility.

Concept & Composer Schematics

In both Concept and Composer schematics, you can assign a limited subset of these resource assignments by assigning properties to symbols. These properties are incorporated into the EDIF netlist file(s). The MAX+PLUS II software automatically converts assignment information from the EDIF Input File into the ACF format. For information on making MAX+PLUS II-compatible resource assignments, go to the following topics:

- Assigning Pins, Logic Cells & Chips
- Assigning Cliques
- Assigning Logic Options
- Modifying the Assignment & Configuration File with the **setacf** Utility

Go to the ***Cadence Concept Schematic User Guide*** and ***Composer Reference User Guide*** for details on how to assign properties. Go to "Resource Assignments in EDIF Input Files" and "Assigning Resources in a Third-Party

Design Editor" in MAX+PLUS II Help for more information on assignments or properties that can be assigned in Concept and Composer.

Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample Concept and Composer schematic files, which include resource assignments:

- **/usr/maxplus2/examples/cadence/example6/fa2** (Concept)
- **/usr/maxplus2/examples/cadence/example7/fa2** (Composer)

VHDL & Verilog HDL Design Files

For Verilog HDL- and VHDL-based designs, you must use the MAX+PLUS II software or the **setacf** utility to enter resource assignments. For information on using the **setacf** utility, go to Modifying the Assignment & Configuration File with the **setacf** Utility.

Related Topics:

- For information on entering assignments in the MAX+PLUS II software with Assign menu commands or in an ACF, go to "resource assignments" or "ACF, format" in MAX+PLUS II Help using **Search for Help on** (Help menu).

Modifying the Assignment & Configuration File with the setacf Utility

Altera provides the **setacf** utility to help you modify a project's Assignment & Configuration File (**.acf**) from the command line, without opening the file with a text editor. Type `setacf -h` ↵ at a UNIX or DOS prompt to get help on this utility.

Synthesizing & Optimizing VHDL Files with Synergy Software

You can use Cadence Synergy software to synthesize and optimize your VHDL files and convert them to EDIF input files that can be processed by the MAX+PLUS® II Compiler. The information presented here describes only how to use VHDL files that have been processed by Synergy software. For information on direct MAX+PLUS II support for VHDL Design Files, go to MAX+PLUS II VHDL Help.

To process a VHDL file with Synergy software for use with MAX+PLUS II software, go through the following steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment.
2. Create a VHDL file <*design name*>**.vhd** using the MAX+PLUS II Text Editor or another standard text editor and save it in a working directory. Go to Creating VHDL Designs for Use with MAX+PLUS II Software for more information on VHDL design entry.
3. Start Synergy by typing `synergy -lang vhdl` ↵ at a UNIX prompt from the working directory.
4. Analyze your source file <*design name*>**.vhd**:
    1. Choose **Analyze Files** (File menu) to open the **Select Design** dialog box.
    2. Click on the **Analyze Files** tab.
    3. Select the design name from the *Files* list.

4. Choose **Analyze** to analyze the source file(s).

5. Choose the **Select Design** tab from the **Select Design** dialog box and specify the following options:

    1. Select the design architecture from the hierarchical list. The design architecture should appear in the *Design* box.

    2. Specify <*design name*>**.run1** as the *Run Directory*.

    3. Type `alt_syn` as the *Target Library* name.

    4. (Optional) If you want to use the Synergy library of parameterized modules (LPM) synthesis capability, choose the **Macro Libraries** ellipse button and select *lpm_syn* in the *Select From* box.

    5. (Optional) If you want to view a synthesized schematic in Concept or Composer, go through the following steps:

    1. Choose **Schematic Generation** (Utilities menu).

    2. Select either *Concept* or *Composer* in the *Generate From* box.

    3. Type `alt_max2` in the *Symbol Libraries* box.

    4. Choose **Apply**, then **Close**.

6. Choose the **Select Design** button from the **Select Design** window.

7. Indicate to the Synergy software that any `clklock` megafunction or any macrofunction instantiated in your VHDL design is a "black box" that must pass untouched through the EDIF netlist file:

    1. Choose **Synthesis** (Constraints menu), then choose **Hierarchy Control**.

    2. Select the module or instance name from the hierarchical **View** list for *Module/Instance*.

    3. Turn on *Maintain Option* in the *Synthesis Constraints* box.

    4. Select *Module/Instance* and *Tree Below* in the *Apply To* box.

    5. Choose **Apply**.

    6. Repeat steps a through e for each instance of the function.

8. Choose **Synthesize** (Synthesis menu) from the Synergy window and specify the following options:

    1. Click on the **Synthesize** tab.

    2. Turn on the *Generate Schematic* option.

    3. Select either *Composer* or *Concept* from the *Type* list box.

    4. Choose **Synthesize** to start synthesizing your design.

9. Generate an EDIF netlist file that can be compiled with MAX+PLUS II software, as described in Converting VHDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the **vlog2alt** or altout **Utility**.

10. Process the <*design name*>**.edf** file with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample VHDL files:

- **/usr/maxplus2/examples/cadence/example9/count4.vhd**
- **/usr/maxplus2/examples/cadence/example10/adder16.vhd**

Converting VHDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the vlog2alt or altout Utility

You can convert a VHDL design into an EDIF netlist file with the extension **.edf**. This file can then be imported into the MAX+PLUS® II software as an EDIF Input File (**.edf**).

To convert a VHDL design into an EDIF netlist file, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment.
2. Synthesize and optimize your VHDL design with Synergy, as described in Synthesizing & Optimizing VHDL Files with Synergy Software.
3. Depending on whether or not you have installed the Concept **alt_syn** library, perform one of the following steps to create <*design name*>**.edf** in the working directory:

   ✔ If you have installed the Concept **alt_syn** library, type the following command at the UNIX prompt from your working directory:

   `vlog2alt` <*design name*> `-rundir max2 -vfiles` <*design name*>`.run1/syn.v` ↵

   or:

   ✔ If you have not installed the Concept **alt_syn** library, follow these steps:

   1. Edit the **cds.lib** file, which is located in your working directory, to include the following line:

      `DEFINE Opt` <*working directory*>/<*design name*>`.run1/Opt` ↵

   2. Type the following command at the UNIX prompt from the working directory:

      `altout -lib Opt -rundir max2` <*design name*> ↵

4. Process the <*design name*>**.edf** file with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample VHDL files:

- **/usr/maxplus2/examples/cadence/example9/count4.vhd**
- **/usr/maxplus2/examples/cadence/example10/adder16.vhd**

Synthesizing & Optimizing Verilog HDL Files with Synergy Software

You can create and process Verilog HDL files and convert them into EDIF input files that can be processed by the MAX+PLUS® II Compiler. To process a Verilog HDL file with Synergy software for use with the MAX+PLUS II software, go through the following steps:

1. Be sure to set up your working environment correctly, as described in Setting up the MAX+PLUS II/Cadence Working Environment.
2. Create a Verilog HDL file <*design name*>**.v** using the MAX+PLUS II Text Editor or another standard text editor and save it in a working directory. Go to Creating Verilog HDL Designs for Use with MAX+PLUS II Software for more information on Verilog HDL design entry.
3. Start Synergy by typing `synergy -lang verilog` ↵ at a UNIX prompt from your working directory.
4. Choose **Select Design** (File menu) from the Synergy window and specify the following options:
    1. Select <*design name*>**.v** from the *Verilog Files* list.
    2. Choose the **Verilog Option** tab from the **Select Design** dialog box.
    3. Specify <*design name*>.**run1** as the *Run Directory*.
    4. Type `/usr/maxplus2/simlib/concept/alt_max2/`<*design name*>`/verilog_lib/verilog.v` <*working directory*>/ in the *Library Files (-v)* box.
    5. (Optional) If your design includes library of parameterized modules (LPM) functions, type `+define+SYNTH` in the *Other Compilations* box.
    6. Choose **Select Design**.
5. Choose the **Design** tab from the **Select Design** dialog box and set the target library:
    1. Type `alt_syn` as the *Target Library* name.
    2. (Optional) To use the Synergy LPM synthesis capability, type `lpm_syn` as the *Library* name in the *Macro Cell Library* box.
    3. Choose **OK**.
6. (Optional) To view the synthesized schematic in Concept or Composer, go through the following steps:
    1. Select **Schematic Generation** (Utilities menu).
    2. Select either *Concept* or *Composer* in the *Generate From* box.
    3. Type `alt_max2` in the *Symbol Libraries* box.
    4. Choose **Apply**, then **Close**.
7. Choose **Select Design** from the **Select Design** window.
8. Choose **Synthesize** (Synthesis menu) from the **Synergy** window and specify the following options:
    1. Click on the **Synthesize** tab.
    2. Turn on the *Generate Schematic* option.
    3. Select either *Composer* or *Concept* from the *Type* list box.

4. Choose **Synthesize** to start synthesizing your design.

9. Generate an EDIF netlist file that can be compiled by the MAX+PLUS II Compiler, as described in Converting Verilog HDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files.

10. Process the <*design name*>**.edf** file with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample Verilog HDL files:

- **/usr/maxplus2/examples/cadence/example11/count8.v**
- **/usr/maxplus2/examples/cadence/example13/rom_test.v**

Converting Verilog HDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the vlog2alt Utility

You can use the **vlog2alt** utility to convert your Verilog HDL design into an EDIF netlist file. This file can then be imported into the MAX+PLUS® II software as an EDIF Input File with the extension **.edf**.

To convert a Verilog HDL design into an EDIF netlist file, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment.

2. Synthesize and optimize your Verilog HDL design with Synergy, as described in Synthesizing & Optimizing Verilog HDL Files with Synergy Software.

3. To convert your Verilog HDL design into an EDIF netlist file, type the following command at the UNIX prompt from your working directory:

   `vlog2alt` <*design name*> `-rundir max2 -vfiles` <*design name*>`.run1/syn.v` ↵

4. Process the <*design name*>**.edf** file with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample Verilog HDL files:

- **/usr/maxplus2/examples/cadence/example11/count8.v**
- **/usr/maxplus2/examples/cadence/example13/rom_test.v**

Compiling Projects with MAX+PLUS II Software

The MAX+PLUS® II Compiler can process design files in a variety of formats. This topic describes how to use MAX+PLUS II software to compile projects in which the top-level design file is an EDIF Input File (with the extension **.edf**).

Related Topics:

- Go to MAX+PLUS II Help for information on compiling VHDL and Verilog HDL, design files directly with the MAX+PLUS II Compiler.

- Go to Running Synopsys Compilers from MAX+PLUS II Software for information on running the Synopsys Design Compiler or FPGA Compiler software on a VHDL or Verilog HDL design from within the MAX+PLUS II Compiler window.

To compile a design (also called a "project") with MAX+PLUS II software, go through the following steps:

1. Create design files that are compatible with the MAX+PLUS II software and convert them into EDIF Input Files with the extension **.edf**. Specific instructions for some tools are described in these MAX+PLUS II ACCESS^{SM} Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your design entry or synthesis and optimization tool.

2. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, create a mapping for each function in a Library Mapping File (**.lmf**) to map the custom symbol to the corresponding EDIF Input File, AHDL Text Design File (**.tdf**), or other MAX+PLUS II-supported design file. These custom functions are represented in design files as hollow-body symbols or "black box" HDL descriptions.

   Go to "Library Mapping Files (**.lmf**)" in MAX+PLUS II Help for more information.

3. Open MAX+PLUS II and specify the name of your top-level design file as the project name with the **Project Name** command (File menu). If you open an HDL file in the MAX+PLUS II Text Editor, you can choose the **Project Set Project to Current File** command (File menu) instead.

   You can also compile a project from a command line. However, the first time you compile a project, the settings you need to specify are easier to specify from within the MAX+PLUS II software. After you have run the graphical user interface for the MAX+PLUS II software at least once, you can more easily use the command-line **setacf** utility to modify options in the Assignment & Configuration File (**.acf**) for the project. Type `setacf -h` ↵ and `maxplus2 -h` ↵ for descriptions of **setacf** and MAX+PLUS II command-line syntax.

4. Choose **Device** (Assign menu) and select the target Altera device family in the *Device Family* drop-down list box. If you wish to implement the design logic in a specific device, select it in the *Devices* box. Otherwise, select *AUTO* to allow the MAX+PLUS II Compiler to choose the best device(s) in the current device family. If your design entry or synthesis and optimization tool required you to specify a target family and/or device, specify the same information in this dialog box. For information on partitioning logic among multiple devices, go to MAX+PLUS II Help. Choose **OK**.

5. Open the Compiler window by choosing the **Compiler** command (MAX+PLUS II menu). Go through the following steps to specify the options necessary to compile the design file(s) in your project:

   1. Ensure that all EDIF netlist files have the extension **.edf** and choose **EDIF Netlist Reader Settings** (Interfaces menu).

   2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor. This name should be the name of the vendor whose tool(s) you used to create the EDIF netlist files. If your vendor name does not appear, select *Custom* instead.

       If you are compiling a design created with Synopsys FPGA Express software, select *Synopsys*, choose the **Customize** button, enter <*project name*>**.lmf** in the *LMF #1* box, choose **OK**, and skip to step 6.

   3. If you selected an existing vendor name in the *Vendor* box and your project contains design files that require custom LMF mappings, choose the **Customize** button to expand the dialog box to show all settings. Turn on the *LMF #2* checkbox and type your custom LMF's filename in the corresponding text box, or select a name from the *Files* box. The selection in the *Vendor* box will change to *Custom* and all settings will be retained until you change them again.

4. If you selected *Custom* in the *Vendor* box, choose the **Customize** button to expand the dialog box to show all settings. Any previously defined custom settings will be displayed. Under *Signal Names*, type one or more names with up to 20 total name characters in the *VCC* or *GND* box if your EDIF Input File(s) use one or more names other than `VCC` or `GND` for the global high or low signals. Multiple signal names must be separated by either a comma (,) or a space. Under *Library Mapping Files*, turn on the *LMF #1* checkbox and type a filename in the text box following it, or select a name from the *Files* box. If necessary, specify another LMF name in the *LMF #2* box. Go to MAX+PLUS II Help for detailed information on the settings available in the **EDIF Netlist Reader Settings** dialog box.

5. Choose **OK**.

6. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, you may need to ensure that all files are present in your project directory, i.e., the same directory as the top-level design file. Otherwise, you must specify the directories containing these files as user libraries with the **User Libraries** command (Options menu).

7. Follow all guidelines that apply to your design entry or synthesis and optimization tool:

   - Exemplar Logic Galileo Extreme-Specific Compiler Settings
   - Synopsys DesignWare-Specific Compiler Settings
   - Converting Synopsys FPGA Compiler & Design Compiler Timing Constraints into MAX+PLUS II-Compatible Format with the **syn2acf** Utility
   - Synplicity Synplify-Specific Compiler Settings

8. If you wish to generate EDIF, VHDL, or Verilog HDL output files for post-compilation simulation or timing analysis with another EDA tool, go through the following steps:

   1. (Optional) Turn on the **Optimize Timing SNF** command (Processing menu) to reduce the size of the output file(s). Turning on this command can reduce the size of output netlists by up to 30%.

      This command does not create optimized timing SNFs on UNIX workstations. However, a non-optimized timing SNF provides the same functional and timing information as an optimized timing SNF.

   2. If you wish to generate EDIF Output Files (**.edo**), go through these steps:

      1. Turn on the **EDIF Netlist Writer** command (Interfaces menu). Then choose the **EDIF Netlist Writer Settings** command (Interfaces menu).

      2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor and choose **OK**. If your vendor name does not appear, select *Custom* instead and specify the settings that are appropriate for your simulation or timing analysis tool. Go to MAX+PLUS II Help for detailed information on the options available in the **EDIF Netlist Writer Settings** dialog box.

      3. To generate an optional Standard Delay Format (SDF) Output File (**.sdo**), choose the **Customize** button to expand the dialog box to show all settings. Select one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**.

      The filenames of the EDIF Output File(s) and optional SDF Output File(s) are the same as the user-defined chip name(s) for the project; if no chip names exist, the Compiler assigns filenames that are based on the project name. For a multi-device project, the Compiler also generates a top-level EDIF Output File that is uniquely identified by "**_t**" appended to the project name. In addition, the Compiler automatically generates a VHDL Memory Model Output File, *<project name>***.vmo**, when it generates an EDIF Output File that contains memory (RAM or ROM).

   3. If you wish to generate VHDL Output Files (**.vho**), turn on the **VHDL Netlist Writer** command

(Interfaces menu). Then choose **VHDL Netlist Writer Settings** command (Interfaces menu). Select *VHDL Output File (.vho)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF ver. 2.1 files contain timing delay information that allows you to perform back-annotation simulation in VHDL with VITAL-compliant simulation libraries. The VHDL Output Files generated by the Compiler have the extension **.vho**, but are otherwise named in the same way as the EDIF Output Files described above.

4. If you wish to generate Verilog HDL Output Files (**.vo**), turn on the **Verilog Netlist Writer** command (Interfaces menu). Then choose **Verilog Netlist Writer Settings** command (Interfaces menu). Select *Verilog Output File (.vo)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF Output Files contain timing delay information that allows you to perform back-annotation simulation in Verilog HDL. The Verilog Output Files generated by the Compiler have the extension **.vo**, but are otherwise named in the same way as the EDIF Output Files described above.

9. To run the MAX+PLUS II Compiler, choose the **Project Save & Compile** command (File menu) or choose the **Start** button in the Compiler window.

   ☞ See step 3 for information on running MAX+PLUS II software from the command line.

10. Once you have compiled the project with the MAX+PLUS II Compiler, you can use the VHDL, Verilog HDL, or EDIF output file(s), and the optional SDF Output File(s) (**.sdo**) to perform timing analysis or timing simulation with another EDA tool. Specific instructions for some tools are described in these MAX+PLUS II ACCESS Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your EDA tool.

The MAX+PLUS II Compiler also generates a Report File (**.rpt**), a Pin-Out File (**.pin**), and one or more of the following files for device programming or configuration:

- JEDEC Files (**.jed**)
- Programmer Object Files (**.pof**)
- SRAM Object Files (**.sof**)
- Hexadecimal (Intel-format) Files (**.hex**)
- Tabular Text Files (**.ttf**)

Related Topics:

- Go to *Compiler Procedures* in MAX+PLUS II Help for information on other available Compiler settings.
- Go to *Programmer Procedures* in MAX+PLUS II Help for instructions on creating other types of programming files and on programming or configuring Altera devices.
- Go to Back-Annotating MAX+PLUS II Pin Assignments to Design Architect Symbols for information on back-annotating pin assignments in Mentor Graphics Design Architect schematics.
- Go to Programming Altera Devices for information on the different programming hardware options for Altera device families.

Go to the following topics, which are available on the web, for additional information:

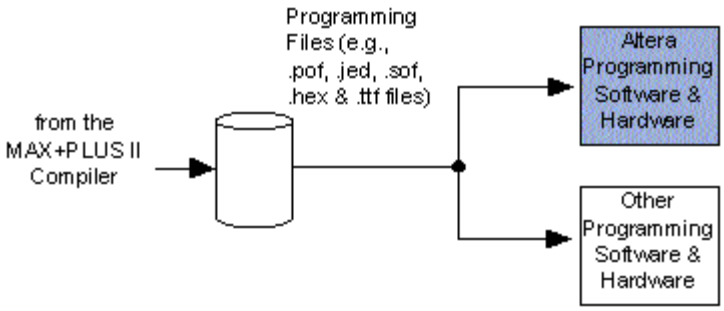
- MAX+PLUS II Development Software
- Altera Programming Hardware

Programming Altera Devices

Once you have successfully compiled and simulated a project with the MAX+PLUS® II software, you can program an Altera® device and test it in the target circuit. Figure 1 shows the device programming flow for MAX+PLUS II software.

Figure 1. MAX+PLUS II Device Programming Flow

Altera-provided items are shown in blue.



You can program devices with Altera programming hardware and MAX+PLUS II Programmer software installed on a 486- or Pentium-based PC or a UNIX workstation, or with programming hardware and software available from other manufacturers. Table 1 shows the available Altera programming hardware options on PCs and UNIX workstations.

Table 1. Altera Programming Hardware

Programming Hardware Option	PCs	UNIX Work-stations	MAX® 3000A Devices	Classic® & MAX 5000 Devices	MAX 7000 & MAX 7000E Devices	MAX 7000A, MAX 7000AE, MAX 7000B, MAX 7000S MAX 9000 & MAX 9000A Devices	FLEX® 6000, FLEX 6000A, FLEX 8000, FLEX 10K, FLEX 10KA, FLEX 10KB, & FLEX 10KE Devices	In-System Programming/ Configuration
Logic Programmer card, PL-MPU Master Programming Unit, and device-specific adapters	✓		✓	✓	✓	✓		
BitBlaster™ Download Cable	✓	✓	✓			✓	✓	✓
ByteBlasterMV™ Download Cable	✓		✓			✓	✓	✓
MasterBlaster™ Download Cable	✓	✓	✓			✓	✓	✓

If you wish to transfer programming files from a UNIX workstation to a PC over a network with File Transfer

Protocol (FTP) or other similar transfer programs, be sure to select binary transfer mode.

Programming hardware from other manufacturers varies, but typically consists of a device connected to one of the serial ports on the workstation. Various vendors, such as Data I/O and BP Microsystems, supply hardware and software for programming Altera devices.

Related Topics:

- Go to Compiling Projects with MAX+PLUS II Software for information on creating programming files.
- Go to the following topics, which are available on the web, for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware
  - FLEX Devices
  - MAX Devices
  - Classic Device Family

Running Synopsys Compilers from the MAX+PLUS II Software

With MAX+PLUS® II software, you can automatically process Verilog HDL and VHDL designs with the Synopsys Design Compiler or FPGA Compiler by following these steps:

1. Create a project directory under your login directory.
2. Add the following environment variables to your **.cshrc** file:

   ```
   setenv ALT_HOME /<MAX+PLUS II system directory> ↵
   setenv SYNOPSYS /<Synopsys system directory> ↵
   ```

3. Add the **$ALT_HOME/synopsys/bin** and **$SYNOPSYS/$ARCH/syn/bin** directories to the PATH environment variable in your **.cshrc** file. The $ARCH environment variable specifies the platform on which the Synopsys Design Compiler is running. Valid platform names are `sparc`, `sparcOS5`, `rs6000`, and `hp700`.
4. Source your **.cshrc** file to update the environment variables.

   ☞ If you use additional custom libraries, you must specify them in a **.synopsys_dc.setup** file, and verify that it contains the correct library settings for mapping to the target family. See Setting Up the Synopsys/MAX+PLUS II Working Environment for more information about the **.synopsys_dc.setup** file.

5. Create your project in Verilog HDL or VHDL using the MAX+PLUS II Text Editor or another standard text editor. You must save Verilog HDL files with the extension **.v** and VHDL files with the extension **.vhd**.

   ☞ If you use the MAX+PLUS II Text Editor to create your design, you can insert templates for Verilog HDL or VHDL constructs with the **Verilog Template** and **VHDL Template** commands (Templates menu). The MAX+PLUS II Text Editor also provides syntax coloring for Verilog HDL and VHDL files to improve file readability.

6. In the MAX+PLUS II software, specify the project to be compiled with **Project Name** (File menu). Make sure the project name specified in MAX+PLUS II software matches both the name of the top-level design file and the Entity Declaration name specified in the top-level design file.
7. Click Button 1 on the **Compiler** toolbar button or choose the **Compiler** command (MAX+PLUS II menu) to open the Compiler.
8. In the MAX+PLUS II Compiler, turn on the **Synopsys Compiler** command (Interfaces menu).
9. Open the **Synopsys Compiler Settings** dialog box by choosing **Synopsys Compiler Settings** (Interfaces menu). Specify the appropriate options:

   1. Select either *Design Compiler or FPGA Compiler* in the *Compiler* box to specify which Synopsys compiler you want to process the design.
   2. If you wish to use the DesignWare interface and libraries, turn on the *DesignWare (FLEX® devices only)* option (FLEX 6000, FLEX 8000, and FLEX 10K devices only).

3. To preserve the design hierarchy during Synopsys compilation, turn on the *Hierarchical Compilation* option. Turning off this option allows the Synopsys compiler to flatten the design.

4. To allow the Synopsys compiler to optimize across all hierarchical boundaries, turn on the *Boundary Optimization* option.

5. Select the *Low, Medium,* or *High* option for *Mapping Effort*.

6. Choose **OK** to save all changes.

10. If you have turned on the *DesignWare (FLEX devices only)* option in the **Synopsys Compiler Settings** dialog box, ensure that the global project synthesis style uses the correct settings. Refer Compiling Projects with MAX+PLUS II Software for more information.

11. Specify the device(s) and output file(s) for the project. If you do not specify a device, the MAX+PLUS II Compiler automatically selects one or more devices from the current device family. Refer to Creating VHDL Design Files for Use with MAX+PLUS II Software for more information.

12. Choose the **Start** button to compile the project. The MAX+PLUS II software converts the EDIF Input File, flattens the project, fits it into one or more Altera® devices, and generates the selected output files, including programming files. The MAX+PLUS II Message Processor notifies you when one of the Synopsys compilers is processing your design. When it has finished processing the design file(s), the Synopsys compiler generates an EDIF netlist file for each design in the hierarchy, and the MAX+PLUS II software immediately compiles the EDIF Input File(s).

Altera provides the **mp2dc_ana** and **mp2dc_cmp** shell scripts, which specify Synopsys Design Compiler or FPGA Compiler settings automatically. These scripts read the settings you have specified in the **Device** (Assign menu) and **Synopsys Compiler Settings** (Interfaces menu) dialog boxes for the project device(s), search path, link library, target library, synthetic library options (if you have turned on the *DesignWare* option in the **Synopsys Compiler Settings** dialog box), and other optimization options. You do not need to provide your own **.synopsys_dc.setup** file unless you use libraries other than Altera libraries. See Setting Up Synopsys Configuration Files for more information.

The MAX+PLUS II software runs both the **mp2dc_ana** and **mp2dc_cmp** shell scripts automatically when you compile a VHDL or Verilog HDL design file with the **Synopsys Compiler** command (Interfaces menu) turned on. The **mp2dc_ana** shell script analyzes your designs and generates a single hierarchical **.db** database file. The analysis output information is recorded in the <*project name*>**.log** file. If the Design Compiler or FPGA Compiler generates errors or warning messages during processing, the messages appear in the MAX+PLUS II Message Processor window. You can select a message that includes a line number and click Button 1 on the **Locate** button to locate the source of a message in the MAX+PLUS II Text Editor. If no errors occur during analysis, the MAX+PLUS II software then starts the **mp2dc_cmp** shell script to read the **.db** file, compile the design, and generate an EDIF netlist file for each design file in the hierarchy, which the MAX+PLUS II software then processes as an EDIF Input File (**.edf**).

The the **mp2dc_ana** and **mp2dc_cmp** shell scripts are located in the **/usr/maxplus2/synopsys/bin** directory. You can copy the **mp2dc_cmp** shell script to your project directory and specify custom settings for your design, such as Clock frequency or timing constraints settings. Alternatively, you can create your own custom **dc_shell** script and name the file **my_mp2dc.scr**. The **mp2dc_cmp** shell script will then use the commands in the **my_mp2dc.scr** file and ignore the current settings or default settings for Synopsys compilation options. Figure 1 shows an excerpt of the Altera-provided **mp2dc_cmp** shell script.

```
read -f db $proj.db >> $proj.log
   if (dc_shell_status == {}) {
   quit
   }
```

```
current_design=$design
uniquify
set_max_area 0



designs= find(design, "*")
foreach (dsgn, designs) {
        current_design= dsgn
        edfout_file = ""
        edfout_file = dsgn
        edfout_file = edfout_file + ".edf"
        set_max_area 0



/* If you do not use my_mp2dc.scr to customize your compilation, the */
/* customizable settings in the following section are used. You can  */
/* customize these settings only if the mp2dc_cmp file is located in */
/* your project directory.                                           */

   if ( "$use_my_cmd" == "true") {
      include my_mp2dc.scr
   } else {
   /* if no hierarchical compilation, then flatten the design  */
      if ( "$hierarchical_compile" == "OFF") {
            set_structure false
            set_flatten -effort low
            ungroup -all
      }



/* test compile options */
      if ( "$boundary_opt" == "ON") {
            compile -boundary_optimization -map_effort $map_effort
      } else {
          compile  -map_effort $map_effort
      }



/* If you use FPGA Compiler for FLEX devices, the LUT equation is output.*/
      /* If you use Design Compiler for FLEX devices, a TBL cell is output.    */
      if ( ("$family" == "flex8000") || ("$family" == "flex10k") ) {



if ( "$synopsys_compiler" == "FPGA" ){
            edifout_write_properties_list = {"lut_function"}
         }  else {
            replace_fpga
         }
      }
 }    /* End of customizable compilation settings section */



write  -f edif current_design  -o edfout_file



if (dc_shell_status == {}) {
            quit
      }
}
```

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Synplicity-Provided Logic Libraries

Synplicity software provides the **altera** logic library that is used for synthesizing and compiling VHDL and Verilog HDL designs. The **altera** library includes the following library files:

Library:	Description:
altera.vhd	A VHDL logic function library that includes the LCELL, SOFT, GLOBAL, CASCADE, and CARRY primitives for controlling design synthesis and fitting. These primitives can be instantiated directly in your VHDL file. These models allow you to perform functional VHDL simulation while maintaining an architecture-independent VHDL description.
altera.v	A Verilog HDL logic function library equivalent to the **altera.vhd** library file.

You can create your own libraries of custom logic functions for use with Synplicity software. You can use custom logic functions to incorporate an EDIF Input File, Text Design File (**.tdf**), or any other MAX+PLUS® II-supported design file into a project. The MAX+PLUS II software uses the **synplcty.lmf** Library Mapping File to map standard Synplicity logic functions to equivalent MAX+PLUS II logic functions. To use custom logic functions, you can create a custom LMF that maps your custom logic functions to the equivalent EDIF Input File, Text Design File (**.tdf**), or other design file. Go to "Library Mapping File" in MAX+PLUS II Help for more information.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Using Cadence Synergy & MAX+PLUS II Software

The following topics describe how to use the Cadence Synergy software with MAX+PLUS® II software. Choose one of the following topics for information:

Open a printable version of all topics listed on this page.

Setting Up the MAX+PLUS II/Cadence Working Environment

- Software Requirements
- MAX+PLUS II/Cadence Interface File Organization
- Altera-Provided Logic & Symbol Libraries

Design Entry

- **Design Entry Flow**
- **Creating VHDL Projects**
  - Creating VHDL Designs for Use with MAX+PLUS II Software
    - Instantiating the clklock Megafunction in VHDL or Verilog HDL
  - Entering Resource Assignments
    - Modifying the Assignment & Configuration File with the **setacf** Utility
- **Creating Verilog HDL Projects**
  - Creating Verilog HDL Designs for Use with MAX+PLUS II Software
    - Instantiating the clklock Megafunction in VHDL or Verilog HDL
  - Entering Resource Assignments
    - Modifying the Assignment & Configuration File with the **setacf** Utility

Synthesis & Optimization

- **VHDL**
  - Synthesizing & Optimizing VHDL Files with Synergy Software
  - Converting VHDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the **vlog2alt** or **altout** Utility
- **Verilog HDL**
  - Synthesizing & Optimizing Verilog HDL Files with Synergy Software
  - Converting Verilog HDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the **vlog2alt** Utility

Related Links:

Go to the following MAX+PLUS II ACCESS[SM] Key topics for related information:

- Compiling Projects with MAX+PLUS II Software
- Programming Altera Devices

Go to the following topics for additional information:

- MAX+PLUS II Development Software
- Altera Programming Hardware
- Cadence web site (http://www.cadence.com)

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Using Synplicity Synplify & MAX+PLUS II Software

The following topics describe how to use the Synplicity Synplify software with MAX+PLUS® II software. Choose one of the following topics for information:

Open a printable version of all topics listed on this page.

Setting Up the MAX+PLUS II/Synplicity Working Environment

- Software Requirements
- MAX+PLUS II Directory Structure
- MAX+PLUS II/Synplicity Interface File Organization
- Synplicity-Provided Logic Libraries

Design Flow

Design Entry

- **Design Entry Flow**

- **VHDL**

    - Creating VHDL Designs for Use with MAX+PLUS II Software
    - Entering Resource Assignments
        - Assigning Pins
        - Assigning the Implement in EAB Logic Option
        - Modifying the Assignment & Configuration File with the **setacf** Utility

- **Verilog HDL**

    - Creating Verilog HDL Designs for Use with MAX+PLUS II Software
    - Entering Resource Assignments
        - Assigning Pins
        - Assigning the Implement in EAB Logic Option
        - Modifying the Assignment & Configuration File with the **setacf** Utility

Synthesis & Optimization

- Synthesizing & Optimizing VHDL or Verilog HDL Files with Synplify Software
- Analyzing VHDL or Verilog HDL Designs with the Synplify HDL Analyst

Compilation

- Project Compilation Flow
- Compiling Projects with MAX+PLUS II Software
  - Synplicity Synplify-Specific Compiler Settings

Device Programming

- Programming Altera Devices

Related Links

- MAX+PLUS II Development Software
- Altera Programming Hardware
- Synplicity web site (http://www.synplicity.com)

MAX+PLUS II/Synplicity Software Requirements

Table 1 shows the software applications that are used to generate, process, synthesize, and verify a project with MAX+PLUS® II and Synplicity software:

Table 1. Software Requirements

Synplicity	Altera
version 6.1 Synplify HDL Analyst	MAX+PLUS II version 10.0

The MAX+PLUS II **read.me** file provides up-to-date information on which versions of Synplicity software applications are supported by the current version of MAX+PLUS II. It also provides information on installation and operating requirements. You should read the **read.me** file on the CD-ROM before installing the MAX+PLUS II software. After installation, you can open the **read.me** file from the MAX+PLUS II Help menu.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Synthesizing & Optimizing VHDL or Verilog HDL Files with Synplify Software

You can create and process VHDL or Verilog HDL files and convert them to Altera® Hardware Description Language (AHDL) Text Design Files (**.tdf**) or EDIF Input Files (**.edf**) that can be processed by the MAX+PLUS® II Compiler. The MAX+PLUS II Compiler can process a VHDL or Verilog HDL file that has been synthesized by Synplify software, saved as an AHDL TDF or an EDIF netlist file, and imported into the MAX+PLUS II software. The information presented here describes only how to use VHDL or Verilog HDL files that have been processed by Synplify software. For information on direct MAX+PLUS II support for VHDL or Verilog HDL Design Files, go to MAX+PLUS II VHDL or Verilog HDL Help.

To process a VHDL or Verilog HDL file with Synplify software for use with MAX+PLUS II software, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Synplicity Working Environment.
2. Create a VHDL file, <*design name*>**.vhd**, or a Verilog HDL file, <*design name*>**.v**, using the MAX+PLUS II Text Editor or another standard text editor and save it in a working directory. Go to Creating VHDL Designs for Use with MAX+PLUS II Software or Creating Verilog HDL Designs for Use with MAX+PLUS II Software for more information on HDL design entry.
3. Start the Synplify software:
   - ✔ On a UNIX workstation, type `synplify` ↵ at a UNIX prompt from your working directory.

   or:

   - ✔ On a PC, double-click the **synplify.exe** icon in your **Â¥synplicityÂ¥bin** directory.
4. Create a new project:
   1. Choose **New** (File menu) to display the **New** dialog box, then select *Project* from the list. Choose **OK.**
   2. Choose the **Add** button from the Project window. The **Add Source Files** dialog box is displayed.
   3. Select your design file(s) and choose the **Open** button to add the file(s) to your *Source Files* list in the Synplify window.

   ☞ If you wish to create a hierarchical project, make sure the top-level design file is at the bottom of the *Source Files* list by selecting the file and dragging it to the bottom of the list.
5. Select the target Altera device:
   1. Choose the **Change** button in the *Target* section. The **Set Device Option** dialog box is displayed.
   2. Select an Altera MAX® (which includes Classic™) or FLEX® device family from the *Technology* list.
   3. Select a device from the *Part* list.
   4. (Optional) Turn on the *Map logic cells to LCELLs* option to increase performance. However, turning on this option may decrease area optimization.

      For MAX or Classic designs, specify the following options:

      ✔
      1. Enter an appropriate value for the *Percent of design to optimize for timing* box.
      2. Enter an appropriate value for the *Maximum cell fan-in* box.
      3. (Optional) Turn on the *Make Non-critical Cells Soft* option to allow the MAX+PLUS II software to reduce the number of logic cells used in implementing non-timing critical portions of the design.

      or:

      ✔ For FLEX designs, select an appropriate value from the *Speed Grade* list.
   5. Select *EDIF* or *AHDL* in the *Result Format* box to specify the output file format from the Synplify

software. Choose **OK**.

☞ Saving your project in AHDL TDF format may improve compilation time. However, if your design uses resource assignment attributes, you should save your file in EDIF netlist file format. See Entering Resource Assignments for more information.

6. Enter the frequency value for the project in the *Frequency (MHz)* box in the Synplify window.
7. (Optional) Turn on the *Symbolic FSM Compiler* option in the Synplify window to direct the Synplify software to automatically find and re-encode state machines in your design. Turning this option on may reduce unnecessary states and transitional logic.
8. Run the Synplify software by choosing the **Run** button in the Synplify window. Synplify software synthesizes and optimizes the design, and creates the EDIF netlist file <*design name*>**.edf** or the AHDL TDF <*design name*>**.tdf**.
9. (Optional) Run the HDL Analyst to analyze and evaluate the performance of your design, as described in Analyzing VHDL or Verilog HDL Designs with the Synplify HDL Analyst.
10. (Optional) Add appropriate timing constraints in a separate Synplify Design Constraints File (**.sdc**) or in the VHDL or Verilog HDL source file. If you add timing constraints or resource assignments in a separate **.sdc** file, you must add the **.sdc** file to the *Source Files* list in the Synplify window.
11. Correct any errors or warnings.
12. If you have corrected errors or warnings, or added timing constraints to your project, repeat step 8 to implement the changes in your synthesized design.
13. Create the /<*project directory*>/**max2** subdirectory.
14. Copy the <*design name*>**.edf** or <*design name*>**.tdf** generated in step 8 to the /<*project directory*>/**max2** directory.
15. Process your design with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

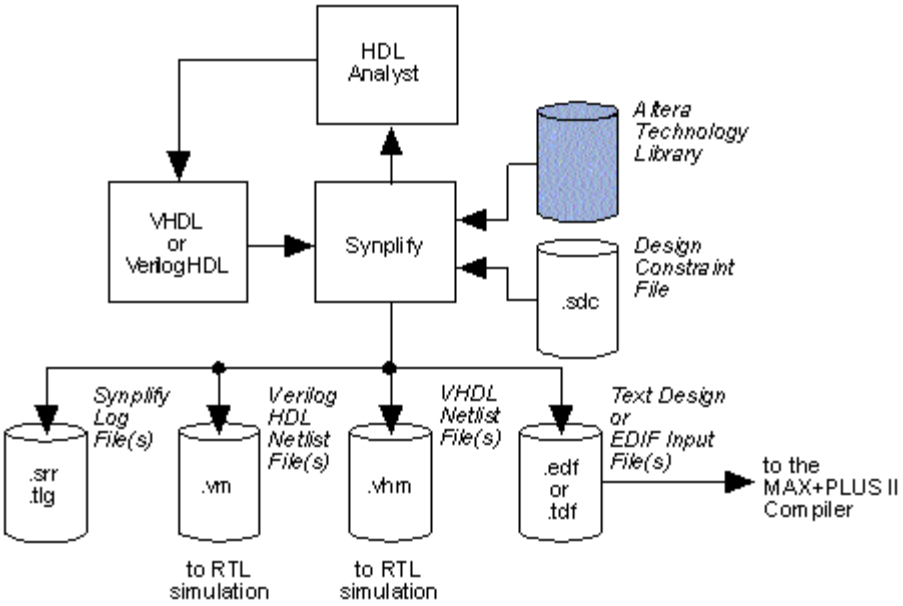
Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Synplicity Design Entry Flow

Figure 1 shows the design entry flow for the MAX+PLUS® II/Synplicity interface.

Figure 1. MAX+PLUS II/Synplicity Design Entry Flow

Altera-provided items are shown in blue.



Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

The MAX+PLUS II System Directory

The information presented here assumes that you are using C shell and that your MAX+PLUS II system directory is **/usr/maxplus2**. If not, you must use the appropriate syntax and procedures to set environment variables for your shell.

Support

Intel Community Forums provides a place to ask and answer questions about Intel products.

Intel does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Design Compiler & FPGA Compiler Technology Libraries

The Altera® -provided Design Compiler and FPGA Compiler technology libraries contain primitives that the Synopsys compilers use to map your designs to the target device architecture. These primitives contain timing and area information that the Synopsys compilers use to meet area and performance requirements. Table 1 shows the functions provided in these libraries. Choose **Primitives** from the MAX+PLUS II Help menu for detailed information on these functions.

Altera recommends instantiating these functions directly in your designs only if the Synopsys compilers do not appear to recognize the functions when synthesizing your design, or if you prefer to hand-optimize certain portions of your design.

Table 1. Altera-Provided Primitives

Name *Note (1), Note (2)*	Description	Name	Description
LCELL	Logic cell buffer primitive	EXP	MAX® 5000, MAX 7000, and MAX 9000 Expander buffer primitive
GLOBAL	Global input buffer primitive	SOFT	Soft buffer primitive
CASCADE	FLEX® 6000, FLEX 8000, and FLEX 10K cascade buffer primitive	OPNDRN	FLEX 6000, FLEX 8000, and FLEX 10K Open-drain buffer primitive
CARRY	FLEX 6000, FLEX 8000, and FLEX 10K cascade buffer primitive	DFF DFFE DFFS *Note (2)*	D-type flipflop with Clock Enable primitive
LATCH	Latch primitive	TFF TFFE TFFS *Note (2)*	T-type flipflop primitive
TRIBUF	Tri-state buffer primitive		

Notes:

(1) All buffer primitive names except OPNDRN must be prefixed with an "A" in FLEX 6000, FLEX 8000, and FLEX 10K designs. The TRIBUF primitive is equivalent to the TRI primitive in the MAX+PLUS II software.

(2) The DFFE and TFFE primitives include a Clock Enable input; the DFFS and TFFS primitives are equivalent to DFF and TFF primitives without Clear or Preset inputs. For designs that are targeted to FLEX 6000 devices, you should use the DFFE or TFFE primitive only if the design contains either a Clear or Preset signal, but not both. If your design contains both a Clear and a Preset signal, you must use the DFFE6K primitive.

The VHDL simulation model **/usr/maxplus2/synopsys/library/alt_pre/<***device family***>/src/<***device family***>_components.vhd** file shows the exact cell and pin names for each device family. The Verilog HDL simulation file **/usr/maxplus2/synopsys/library/alt_pre/verilog/src/altera.v** shows the functionality of these cells.

Table 2 lists the technology library names.

Table 2. Altera Technology Libraries

Altera Device Family	Synopsys Design Compiler	Synopsys FPGA Compiler
FLEX® 10K devices	**flex10k.db** **flex10k-2.db** **flex10k-3.db** **flex10k-4.db** **flex10k-5.db**	**flex10k_fpga.db** **flex10k-2_fpga.db** **flex10k-3_fpga.db** **flex10k-4_fpga.db** **flex10k-5_fpga.db**
FLEX 8000 devices	**flex8000.db** **flex8000-2.db** **flex8000-3.db** **flex8000-4.db** **flex8000-5.db** **flex8000-6.db**	**flex8000_fpga.db** **flex8000-2_fpga.db** **flex8000-3_fpga.db** **flex8000-4_fpga.db** **flex8000-5_fpga.db** **flex8000-6_fpga.db**
FLEX 6000 devices	**flex6000-2.db** **flex6000-3.db**	**flex6000-2_fpga.db** **flex6000-3_fpga.db**
MAX® 9000 devices	**max9000.db**	**max9000_fpga.db**
MAX 7000, MAX 7000E, MAX 7000S, & MAX 7000A devices	**max7000.db**	**max7000_fpga.db**
MAX 5000 & Classic® devices	**max5000.db**	**max5000_fpga.db**

Related Links:

- Go to MAX+PLUS® II /Synopsys Interface File Organization in these MAX+PLUS II ACCESS℠ Key topics for related information.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Converting Synopsys Timing Constraints into MAX+PLUS II-Compatible Format with the syn2acf Utility

Altera provides the **syn2acf** utility, which is an interface program that converts Synopsys timing constraints from non-hierarchical designs into the MAX+PLUS® II Assignment & Configuration File (**.acf**) format. For information on converting timing constraints from hierarchical designs, refer to Converting Synopsys Hierarchical Timing Constraints into MAX+PLUS II-Compatible Format with the **gen_iacf** and **gen_hacf** Utilities.

The **syn2acf** utility requires the following input files:

- Flattened EDIF netlist file
- **dc_shell** script file
- Standard Delay Format (SDF) constraints construct
- SDF timing delay construct

To use the **syn2acf** utility, follow these steps:

1. Set the timing constraints by using one of the following methods:

   ✔ Start the Synopsys Design Analyzer and specify timing constraints by choosing appropriate menu commands.

   or:

   ✔ Create the <*design name*>**.cmd** file for use with a **dc_shell** script. See Figure 1.

☞ The **syn2acf** utility does not support `set_arrival` timing constraints for internal nodes.

Figure 1. Sample Command File (.cmd) for Setting Timing Constraints

```
create_clock -period 50 -waveform {0 25} CLK
set_clock_skew -delay 2 CLK
set_input_delay 10 IN2
set_input_delay 5 -clock CLK IN1
set_output_delay 20 OUT2
set_output_delay 5 -clock CLK OUT1
set_max_delay 25 -to OUT1
set_max_delay 35 -to OUT2
set_multicycle_path 2 -to n20_reg
```

1. Compile the design and run the **syn2acf** utility either from the command line or with a Design Compiler **dc**

script:

- Compile the design, then type the following command from the UNIX prompt to start the **syn2acf** utility:

```
/usr/maxplus2/synopsys/bin/syn2acf <design name>↵
```

or:

- Run a **dc** script inside the **dc_shell** script that reads the VHDL design, compiles it, and runs the **syn2acf** utility. Figure 2 shows a sample **dc** script.

The **syn2acf** utility uses the `ALT_HOME` environment variable, if it has been specified, to determine the MAX+PLUS II system directory; otherwise, it uses the **/usr/maxplus2** directory. To specify a different MAX+PLUS II system directory with the `ALT_HOME` environment variable, you can either edit the **.cshrc** file to specify the correct directory or type the following command at the UNIX prompt:

```
setenv ALT_HOME <MAX+PLUS II system directory> ↵
```

Figure 2. Sample Script for Running the syn2acf Utility

```
/* dc_script example to interface with syn2acf */
dc_shell <<!
read -f vhdl <design name>.vhd
include <design name>.cmd                          /*set timing constraints*/
compile
current_design=<design name>
include /usr/maxplus2/synopsys/bin/syn2acf.cmd     /*generate required files*/
sh /usr/maxplus2/synopsys/bin/syn2acf <design name> /*invoke syn2acf utility*/
quit
!
```

The **syn2acf** utility cannot support maximum Clock frequency ($\mathbf{f_{MAX}}$) correctly if more than one Clock skew is specified in the **dc_shell** command script. This problem occurs because the Synopsys `write_script` command drops the Clock skew information for the registers. The **syn2acf** utility will use the last Clock skew number to calculate $\mathbf{f_{MAX}}$.

The sample **dc** script includes the Altera® -provided **syn2acf.cmd** file, shown in Figure 3, to generate the required input files for the **syn2acf** utility.

Figure 3. Altera-Provided syn2acf.cmd File

```
ungroup -flatten -all write -f edif write_script > altsyn.dc write_constraints -format
sdf -cover_design write_timing -format sdf
```

All timing assignments generated by the **syn2acf** utility are written to the Timing Requirement Assignments Section of the project's ACF, with the assignment source identifier `{synopsys}` at the end of each line. Figure 4 shows a sample ACF excerpt that contains Synopsys timing constraints generated by the **syn2acf** utility.

Figure 4. Sample ACF Excerpt with Synopsys Timing Constraints

```
TIMING_POINT
BEGIN
   "|OUT2"    : TCO = 15.00ns {synopsys};
   "|IN1"     : TPD = 10.00ns {synopsys};
   "|IN2"     : TPD = 5.00ns {synopsys};
   "|OUT1"    : TCO = 20.00ns {synopsys};
   "|IN1"     : TSU = 20.00ns {synopsys};
   "|IN2"     : TSU = 117.00ns {synopsys};
   "|CLK"     : FREQUENCY = 50.00ns {synopsys};
   "|n10_reg" : FREQUENCY = 100.00ns {synopsys};
END;
```

Altera provides sample files for these utilities in the **/usr/maxplus2/synopsys/bin** directory.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

ACCESS Partner EDA Tools, Listed by Tool

Click on one of the following tool names for information on how to use it with the MAX+PLUS® II software:

- Certify (Synplicity)
- Composer (Cadence)
- Concept (Cadence)
- Design Architect (Mentor Graphics)
- Design Compiler (Synopsys)
- Design Viewpoint Editor (see QuickSim II)
- FPGA Compiler (Synopsys)
- FPGA Express (Synopsys)
- Galileo Extreme (Exemplar Logic)
- Leapfrog (Cadence)
- Leonardo (Exemplar Logic)
- PrimeTime (Synopsys)
- QuickHDL and QuickHDL Pro (Mentor Graphics)
- QuickPath (Mentor Graphics)
- QuickSim II (Mentor Graphics)
- RapidSIM (Cadence)
- Synergy (Cadence)
- Synplify (Synplicity)
- Synplify Pro (Synplicity)
- Verilog-XL (Cadence)
- VHDL System Simulator [VSS] (Synopsys)

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

DesignWare Up/Down Counter Function Instantiation Example for VHDL

The Altera DesignWare Libraries for FLEX devices allow you to instantiate the `DW03_updn_ctr` function, which is the same as the Synopsys `DW03` up/down counter. This function allows you to use the same VHDL code regardless of which FLEX® device is targeted.

Figure 1 shows a VHDL file excerpt with DW03_updn_ctr instantiation.

Figure 1. VHDL File Excerpt with Up/Down Counter Instantiation

```
LIBRARY ieee,DW03;
USE ieee.std_logic_1164.all;
USE DW03.DW03_components.all;



ENTITY updn_4 IS
   PORT (D : IN STD_LOGIC_VECTOR(4-1 DOWNTO 0);
      UP_DN, LD, CE, CLK, RST: IN STD_LOGIC;
      TERCNT : OUT STD_LOGIC;
      Q      : OUT STD_LOGIC_VECTOR(4-1 DOWNTO 0));
END updn_4;



ARCHITECTURE structure OF updn_4 IS



BEGIN
   u0: DW03_updn_ctr
   GENERIC MAP(width => 4)
   PORT MAP (data => d, clk => clk, reset => rst, up_dn => up_dn,
             load => ld, tercnt => tercnt, cen => ce, count => q);
END structure;
```

Related Links:

- Go to Setting Up the DesignWare Interface in these MAX+PLUS II ACCESS℠ Key topics for related information.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Updating DesignWare Libraries

Although Altera provides DesignWare libraries that are pre-compiled for the current version of Synopsys tools, you may wish to recompile the libraries.

Altera provides compilable source files and scripts that allow you to automate the compilation process. These source files allow you to use DesignWare software with any version of the Design Compiler or FPGA Compiler software.They also allow you to install components whose source is written in VHDL, even if you are licensed only for the Verilog HDL Compiler software.

Source files for the Design Compiler software are automatically installed in the following directories:

- **/usr/maxplus2/synopsys/library/alt_syn/flex10k/src/dw_flex10k[<*speed grade*>]**
- **/usr/maxplus2/synopsys/library/alt_syn/flex8000/src/dw_flex8000[<*speed grade*>]**
- **/usr/maxplus2/synopsys/library/alt_syn/flex6000/src/dw_flex6000<*speed grade*>**

Source files for the FPGA Compiler are automatically installed in the following directories:

- **/usr/maxplus2/synopsys/library/alt_syn/flex10k/src/dw_flex10k[<*speed grade*>]_fpga**
- **/usr/maxplus2/synopsys/library/alt_syn/flex8000/src/dw_flex8000[<*speed grade*>]_fpga**
- **/usr/maxplus2/synopsys/library/alt_syn/flex6000/src/dw_flex6000<*speed grade*>_fpga**

Table 1. Commands for Compiling the Library

Device Family	Synopsys Compiler	Commands for Compiling the Library *Note (1)*
FLEX®6000	Design Compiler	`cd /usr/maxplus2/synopsys/library/alt_syn/flex6000/src/dw_flex6000<speed grade>` ↵ `dw_flex6000.script` ↵
	FPGA Compiler	`cd /usr/maxplus2/synopsys/library/alt_syn/flex6000/src/dw_flex6000<speed grade>_fpga` ↵ `dw_flex6000.script` ↵
FLEX 8000	Design Compiler	`cd/usr/maxplus2/synopsys/library/alt_syn/flex8000/src/dw_flex8000[<speed grade>]` ↵ `dw_flex8000.script` ↵
	FPGA Compiler	`cd/usr/maxplus2/synopsys/library/alt_syn/flex8000/src/dw_flex8000[<speed grade>]_fpga` ↵ `dw_flex8000.script` ↵
FLEX 10K	Design Compiler	`cd/usr/maxplus2/synopsys/library/alt_syn/flex10k/src/dw_flex10k[<speed grade>]` ↵ `dw_flex10k.script` ↵
	FPGA Compiler	`cd/usr/maxplus2/synopsys/library/alt_syn/flex10k/src/dw_flex10k[<speed grade>]_fpga` ↵ `dw_flex10k.script` ↵

1. For FLEX 6000 devices, you must specify either `-2` or `-3` for the <*speed grade*> variable. For FLEX 8000 and FLEX 10K devices, you must specify `-2`, `-3`, `-4`, `-5`, or `-6`; or `-2`, `-3`, `-4`, or `-5`; respectively, for the <*speed grade*> variable.

Related Links:

- Go to the following topics for additional information:
  - Setting Up the DesignWare Interface
  - Setting Up the MAX+PLUS II/Synopsys Working Environment
  - Setting Up Design Compiler & FPGA Compiler Configuration Files
  - Setting Up VSS Configuration Files

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Altera Simulation Libraries

Altera provides simulation libraries for both pre-routing functional simulation and post-routing timing simulation.

Pre-Routing Functional Simulation Libraries (VITAL-Compliant)

The **/usr/maxplus2/synopsys/library/alt_pre/vital/src** directory contains Altera® -provided VHDL simulation models in VITAL 95 format. This library contains functional descriptions of all primitives that appear in Altera-specific technology libraries. These libraries allow you to perform a functional or pre-routing simulation that verifies the netlist structure generated by the Synopsys Design Compiler or FPGA Compiler software. Altera provides the **flex.cmp** and **flex.vhd** files in the **/usr/maxplus2/synopsys/library/alt_pre/vital/src** directory.

Similarly, the **/usr/maxplus2/synopsys/library/alt_pre/verilog/src** directory contains Altera-provided Verilog HDL simulation models for all device families. The **altera.v** file can be used for simulation with the Cadence Verilog-XL simulator.

Pre-Routing Functional Simulation Libraries with Estimated Timing Information

The **/usr/maxplus2/synopsys/library/alt_pre/**<*device family*>**/src** directory contains Altera® -provided VHDL simulation libraries, which give both functional and area descriptions of all primitives that appear in all Altera technology libraries. These simulation libraries allow you to verify the function of VHDL projects, with estimated timing, after synthesizing them with the Synopsys Design Compiler or FPGA Compiler, but before submitting them to MAX+PLUS® II software for compilation.

Altera provides an encrypted Full Timing Structural Model (FTSM) and a Full Timing Gate-Level Simulation model (FTGS) for the VHDL simulation libraries listed in Table 1.

Table 1. VHDL Functional Simulation Libraries

Device Family	Functional Simulation Libraries	Device Family	Functional Simulation Libraries
FLEX® 10K	**flex10k_FTSM.vhd.E** **flex10k_fpga_FTSM.vhd.E** **flex10k_FTGS.vhd.E** **flex10k_fpga_FTGS.vhd.E** **flex10k_components.vhd** **flex10k_fpga_components.vhd**	MAX® 9000	**max9000_FTSM.vhd.E** **max9000_fpga_FTSM.vhd.E** **max9000_FTGS.vhd.E** **max9000_fpga_FTGS.vhd.E** **max9000_components.vhd** **max9000_fpga_components.vhd**
FLEX 8000	**flex8000_FTSM.vhd.E** **flex8000_fpga_FTSM.vhd.E** **flex8000_FTGS.vhd.E** **flex8000_fpga_FTGS.vhd.E** **flex8000_components.vhd** **flex8000_fpga_components.vhd**	MAX 7000	**max7000_FTSM.vhd.E** **max7000_fpga_FTSM.vhd.E** **max7000_FTGS.vhd.E** **max7000_fpga_FTGS.vhd.E** **max7000_components.vhd** **max7000_fpga_components.vhd**
FLEX 6000	**flex6000_FTSM.vhd.E** **flex6000_fpga_FTSM.vhd.E** **flex6000_FTGS.vhd.E** **flex6000_fpga_FTGS.vhd.E** **flex6000_components.vhd** **flex6000_fpga_components.vhd**	MAX 5000 & Classic®	**max5000_FTSM.vhd.E** **max5000_fpga_FTSM.vhd.E** **max5000_FTGS.vhd.E** **max5000_fpga_FTGS.vhd.E** **max5000_components.vhd** **max5000_fpga_components.vhd**

Post-Routing Timing Simulation Libraries

The **/usr/maxplus2/synopsys/library/alt_post/sim/src** directory contains the Altera® -provided library files for performing timing simulation of designs that have been compiled with the MAX+PLUS II software. The VITAL 95-compliant post-simulation source files included in this directory are **alt_vtl.vhd** and **alt_vtl.cmp**. See Performing a Timing Simulation with VSS Software for more information.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Performing a Timing Simulation with Verilog-XL Software

Once the MAX+PLUS® II software has compiled a project and generated a Verilog Output File (**.vo**), you can perform a timing simulation using Cadence Verilog-XL software.

To simulate Verilog output files with the Verilog-XL timing simulator, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment.
2. Generate Verilog Output Files (**.vo**), as described in Compiling Projects with MAX+PLUS II Software. The MAX+PLUS II Compiler generates the *<design name>***.vo** and **alt_max2.vo** files for use with Verilog-XL software.
3. Using any standard text editor, create a stimulus file that includes test vectors for your design.
4. Start the Verilog-XL simulator and simulate your Verilog output files by typing the following command at the UNIX prompt:

   `verilog` *<stimulus filename(s)>* *<design name>* `alt_max2.vo` ↵

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Using Cadence Verilog-XL & MAX+PLUS II Software

The following topics describe how to use the Cadence Verilog-XL software with MAX+PLUS® II software. Choose one of the following topics for information:

Open a printable version of all topics listed on this page.

Setting Up the MAX+PLUS II/Cadence Working Environment

- Software Requirements
- MAX+PLUS II Directory Structure
- MAX+PLUS II/Cadence Interface File Organization

Functional Simulation

- Performing a Functional Simulation of a Concept Schematic with the **hdlconfig** Utility & Verilog-XL Software
- Performing a Functional Simulation of a Concept Schematic with VerilogLink & Verilog-XL Software

Timing Simulation

- Project Simulation Flow
- Initializing Registers in VHDL & Verilog Output Files for Power-Up before Simulation
- Performing a Timing Simulation with Verilog-XL Software

Related Links:

Go to the following MAX+PLUS II ACCESS℠ Key topics for related information:

- Using Cadence Concept & MAX+PLUS II Software
- Compiling Projects with MAX+PLUS II Software
- Programming Altera Devices

Go to the following topics, which are available on the web, for additional information:

- MAX+PLUS II Development Software
- Altera Programming Hardware
- Cadence web site (http://www.cadence.com)

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Using the Altera VHDL Express (vhd_exprss) Utility

Once you have created a VHDL Design File (**.vhd**) for your project, you can use the Altera ® VHDL Express (**vhd_exprss**) utility to synthesize and optimize the design and generate an EDIF netlist file with Galileo Extreme software; process the EDIF netlist file with the MAX+PLUS II software to generate a VHDL Output File (**.vho**); and prepare the VHDL Output File for simulation with QuickHDL software. The **vhd_exprss** utility creates all necessary subdirectories and copies all files to the correct locations.

To use the **vhd_exprss** utility, follow these steps:

1. Be sure to set up the working environment correctly, as described in Setting Up the MAX+PLUS II/Mentor Graphics/Exemplar Logic Working Environment.
2. Create a VHDL Design File that follows the guidelines described in Creating VHDL & Verilog HDL Designs for Use with MAX+PLUS II Software.
3. Select the VHDL Design File for your project, press Button 3, and choose **Open vhd_exprss** from the Navigator window to start the Altera VHDL Express tool.
4. Specify settings for the *Input HDL File*, *Altera Device Family*, *Max2 Synthesis Style*, *Process Direction*, and *Verbose* options, and the *Optimize* and *Effort* runtime options, in the **vhd_exprss** dialog box, and choose **OK**.
5. If necessary, correct any errors in the VHDL Design File and recompile the project. The **vhd_exprss** utility generates a VHDL output file in the appropriate directory.
6. Simulate your project, as described in Performing a Timing Simulation with QuickHDL Software.

Related Topics:

- Performing a Timing Analysis with QuickPath Software in these MAX+PLUS II ACCESS℠ Key topics for related information.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Instantiating LPM Functions in VHDL

You can use Mentor Graphics Design Architect software to help you instantiate library of parameterized modules (LPM) functions in your VHDL design files.

To incorporate an LPM function into a VHDL design file, perform the following steps:

1. Be sure to set up the Design Architect working environment correctly, as described in Setting Up the MAX+PLUSÂ II/Mentor Graphics/Exemplar Logic Working Environment.
2. Open a dummy schematic in the Design Architect software:
    1. Start the AlteraÂ® /Mentor Graphics interface by typing `max2_dmgr` ↵ at a UNIX prompt.
    2. Start the Design Architect software by double-clicking Button 1 on the **max2_da** icon in the Design Manager tools window.
    3. Choose the **OPEN_SHEET** button in the Design Architect session_palette, then specify your project name in the *Component Name* box. Choose **OK**.
3. Instantiate the desired LPM function in the dummy schematic:
    1. Choose **Altera Libraries** (Library menu).
    2. Choose **ALTERA LPMLIB** (Altera Libraries menu).
    3. Choose from the available LPM functions on the ALTERA LPMLIB menu.
    4. In the **LPM_**<*lpm function*> dialog box, specify a name for the LPM function in the *Cell Name* box and appropriate values for the function's parameters. Make sure that any hexadecimal (Intel-format) file that you use to specify the initial content of a memory function does not have the same name as the design file name. Choose **Megafunctions/LPM** from the MAX+PLUSÂ® Â II Help menu for detailed information about LPM functions.
    5. Choose **OK** to generate the LPM function, the corresponding VHDL simulation model, and a VHDL Component Declaration/Attribute Declaration/Attribute Specification (**.cmp**) template.
4. Close the Design Architect software without saving the dummy schematic.
5. Instantiate the function created in step 2 in your design file. Use the template file to help prevent syntax and other errors.
6. Continue with the steps necessary to complete your design file, as described in Creating VHDL & Verilog HDL Designs for Use with MAX+PLUSÂ® Â II Software.

Installing the AlteraÂprovided MAX+PLUSÂ II/Mentor Graphics/Exemplar Logic interface on your computer automatically creates the sample hierarchical VHDL design file **/usr/maxplus2/examples/mentor/example8/adder16.vhd**, which includes LPM function instantiation.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Converting VHDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the vlog2alt or altout Utility

You can convert a VHDL design into an EDIF netlist file with the extension **.edf**. This file can then be imported into the MAX+PLUS® II software as an EDIF Input File (**.edf**).

To convert a VHDL design into an EDIF netlist file, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment.
2. Synthesize and optimize your VHDL design with Synergy, as described in Synthesizing & Optimizing VHDL Files with Synergy Software.
3. Depending on whether or not you have installed the Concept **alt_syn** library, perform one of the following steps to create <*design name*>**.edf** in the working directory:

   ✔ If you have installed the Concept **alt_syn** library, type the following command at the UNIX prompt from your working directory:

   `vlog2alt` <*design name*> `-rundir max2 -vfiles` <*design name*>`.run1/syn.v` ↵

   or:

   ✔ If you have not installed the Concept **alt_syn** library, follow these steps:

   1. Edit the **cds.lib** file, which is located in your working directory, to include the following line:

      `DEFINE Opt` <*working directory*>/<*design name*>`.run1/Opt` ↵

   2. Type the following command at the UNIX prompt from the working directory:

      `altout -lib Opt -rundir max2` <*design name*> ↵

4. Process the <*design name*>**.edf** file with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample VHDL files:

- **/usr/maxplus2/examples/cadence/example9/count4.vhd**
- **/usr/maxplus2/examples/cadence/example10/adder16.vhd**

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Primitive & Old-Style Macrofunction Instantiation Example for VHDL

You can instantiate the MAX+PLUS® II primitives listed in Design Compiler & FPGA Compiler Technology Libraries in VHDL designs. These primitives can be used to control synthesis in the MAX+PLUS II software. You can also instantiate MAX+PLUS II megafunctions and old-style macrofunctions.

Go to the following topics for information and examples of how to instantiate functions that are not considered to be hollow bodies, including functions in the **alt_mf** library, RAM and ROM, and the `clklock` megafunction:

- Architecture Control Macrofunction Instantiation Example for VHDL
- Instantiating RAM & ROM Functions in VHDL
- Instantiating the clklock Megafunction in VHDL or Verilog HDL

Unlike other logic functions, MAX+PLUS II primitives do not need to be defined with Component Declarations unless you wish to simulate the design with the VHDL System Simulator (VSS) software. Any references to these primitives are resolved by the Synopsys compilers. All buffer primitives except the `ATRIBUF` and `TRIBUF` primitives also have a "don't touch" attribute already assigned to them, which prevents the Synopsys compilers from optimizing them. The Synopsys compilers also automatically treat mega- and macrofunctions that do not have corresponding synthesis library models as "black boxes."

Figure 1 shows a 4-bit full adder with registered output that also instantiates an `AGLOBAL` or `GLOBAL` primitive. This figure also illustrates the use of global Clock and global Reset pins in the MAX 7000 architecture. The design uses an old-style `7483` macrofunction, which is represented as a hollow body named `fa4`.

Figure 1. 4-Bit Adder Design with Registered Output (adder.vhd)

```
LIBRARY ieee;
   USE ieee.std_logic_1164.ALL;



ENTITY adder IS
   PORT (a, b     : IN  STD_LOGIC_VECTOR(4 DOWNTO 1);
         clk, rst : IN  STD_LOGIC;



cout     : OUT STD_LOGIC;
         regsum   : OUT STD_LOGIC_VECTOR(4 DOWNTO 1));
   END adder;



ARCHITECTURE MAX7000 OF adder IS



SIGNAL sum            : STD_LOGIC_VECTOR(4 DOWNTO 1);
SIGNAL ci, gclk, grst : STD_LOGIC;
```

```
-- Component Declaration for GLOBAL primitive
-- For FLEX devices, global, a_in, and a_out should be replaced with
-- aglobal, in1, and Y, respectively
COMPONENT global
   PORT (a_in      : IN  STD_LOGIC;
         a_out     : OUT STD_LOGIC);
END COMPONENT;



-- Component Declaration for fa4 macrofunction
COMPONENT fa4
   PORT (c0,a1,b1,a2,b2,a3,b3,a4,b4 : IN  STD_LOGIC;
         s1,s2,s3,s4,c4             : OUT STD_LOGIC);
END COMPONENT;



BEGIN
   ci <= '0';



-- FA4 Component Instantiation
   u0: fa4



PORT MAP (ci,a(1),b(1),a(2),b(2),a(3),b(3),a(4),b(4),
             sum(1),sum(2),sum(3),sum(4),cout);



-- GLOBAL Component Instantiation for Clock
-- For FLEX devices, global should be replaced with aglobal
   u1: global
   PORT MAP (clk, gclk);



-- GLOBAL Component Instantiation for Reset
-- For FLEX devices, global should be replaced with aglobal
   u2: global
   PORT MAP (rst, grst);



-- CLOCK process to create registered output
   clocked: PROCESS(gclk,grst)



BEGIN
      IF grst = '0' THEN
      regsum <= "0000"



ELSIF gclk'EVENT AND gclk = '1' THEN
            regsum <= sum;
      END IF;



END PROCESS clocked;

END MAX7000;
```

Before you can analyze the 4-bit adder design, you must first analyze the `fa4` description in Figure 1 with the Synopsys VHDL Compiler software. You can ignore the warning that is issued for any unknown function, including the `fa4` function in this example. If you wish, you can avoid receiving such warning messages by creating

a hollow-body description of the function.

A hollow-body VHDL description combines an Entity Declaration with an empty or null Architecture Body. An empty Architecture Body contains the `ARCHITECTURE IS` clause, followed by the `BEGIN` and `END` keywords and a semicolon (;). It does not include any information about the design's function or operation. Figure 2 shows the hollow-body description for the `fa4` function.

Figure 2. Hollow-Body Description of a 4-Bit Full Adder (7483)

```
LIBRARY ieee;
USE     ieee.std_logic_1164.ALL;



-- fa4 maps to 7483. The interface names do not have to match.

ENTITY fa4 IS



PORT (c0,a1,b1,a2,b2,a3,b3,a4,b4 : IN STD_LOGIC;
      s1,s2,s3,s4,c4             : OUT STD_LOGIC);



END fa4;

ARCHITECTURE map7483 OF fa4 IS



BEGIN

-- This architecture body is left blank, and will map to the
-- 7483 macrofunction in MAX+PLUS II.

END;
```

When you analyze the hollow-body design description with the Synopsys VHDL Compiler software, it produces a hollow-body component that contains a single level of hierarchy with input and output pins, but does not contain any underlying logic.

You can save the synthesized design as an EDIF netlist file (**.edf**) and compile it with the MAX+PLUS II software. After the VHDL Compiler software has successfully processed the design, it generates the schematic shown in Figure 3, which you can view with the Design Analyzer software.

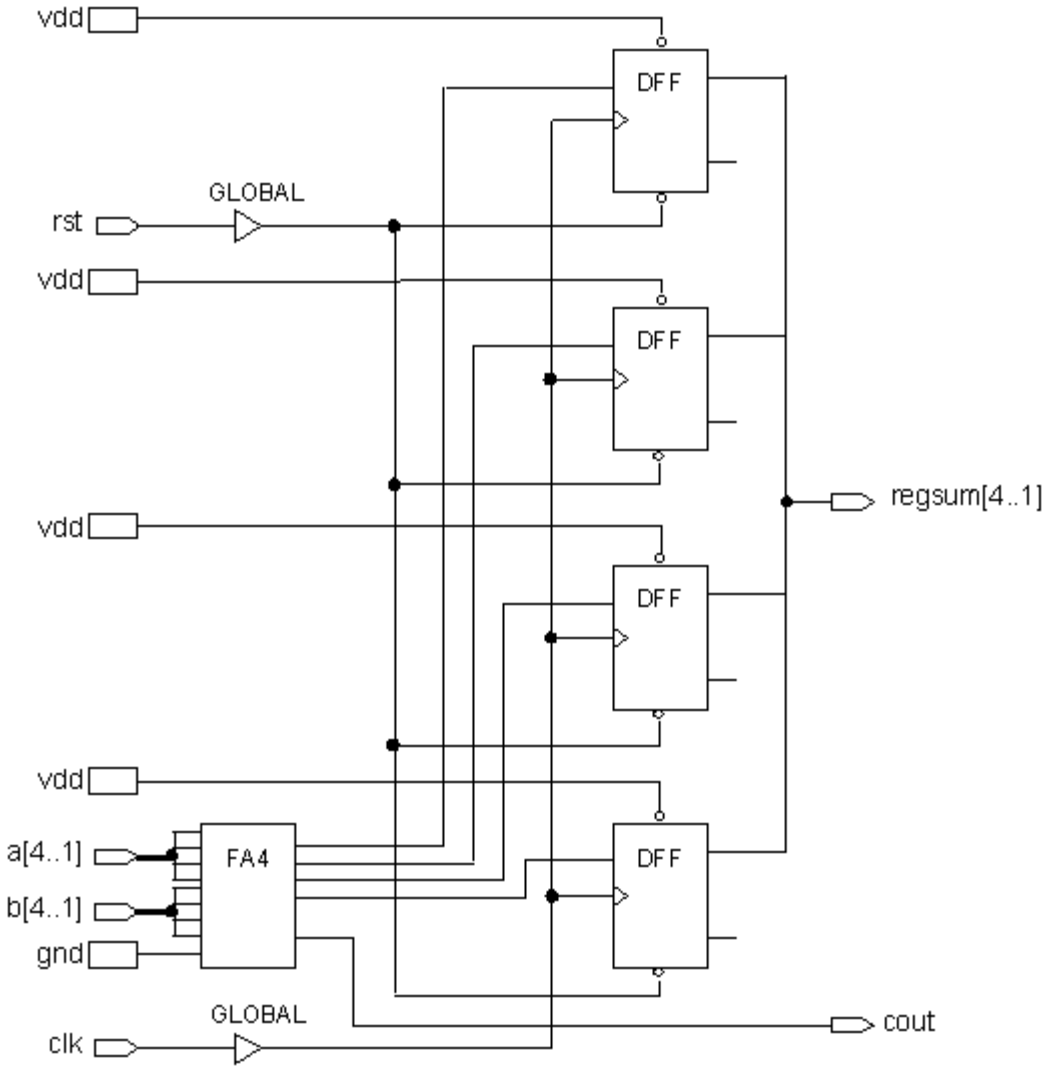


Figure 3. Synthesized Design Generated by the Design Compiler

However, before you compile the EDIF netlist file with the MAX+PLUS II software, you must create the **adder.lmf** file, shown in Figure 3, to map the `fa4` function to the equivalent MAX+PLUS II function (`7483`). You must then specify the LMF as *LMF #2* in the expanded **EDIF Netlist Reader Settings** dialog box (Interfaces menu) (*LMF #1* is **altsyn.lmf**). For more information about creating LMFs, refer to "Library Mapping Files (**.lmf**)" and "Library Mapping File Format" in MAX+PLUS II Help.

Figure 3. Library Mapping File Excerpt for fa4

```
BEGIN
FUNCTION 7483  (c0, a1, b1, a2, b2, a3, b3, a4, b4,)
RETURNS        (s1, s2, s3, s4, c4)



FUNCTION "fa4" ("c0", "a1", "b1", "a2", "b2", "a3",
               "b3","a4", "b4")
RETURNS        ("s1", "s2", "s3", "s4", "c4")
END
```

When you compile the design with the MAX+PLUS II software, you can disregard the warning "`EDIF cell <`*name*`> already has LMF mapping so CONTENTS construct has been ignored`". To verify the global Clock and global Reset usage, as well as the number of logic cells used, see the **adder.rpt** Report File generated by the MAX+PLUS II Compiler.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Creating VHDL Designs for Use with MAX+PLUS II Software

You can create VHDL design files with the MAX+PLUS® II Text Editor or another standard text editor and save them in the appropriate directory for your project. The MAX+PLUS II Text Editor offers the following advantages:

- VHDL templates are available with the **VHDL Templates** command (Templates menu). These templates are also available in the ASCII **vhdl.tmp** file, which is located in the **/usr/maxplus2** directory.
- If you use the MAX+PLUS II Text Editor to create your VHDL design, you can use the **Syntax Coloring** command (Options menu). The Syntax Coloring feature displays keywords and other elements of text in text files in different colors to distinguish them from other forms of syntax.

Once you have created a VHDL design, you can use the Design Compiler or FPGA Compiler to synthesize and optimize it, and then generate an EDIF netlist file that can be processed with the MAX+PLUS II software.

To create a VHDL design that can be synthesized and optimized with the Design Compiler or FPGA Compiler, follow these steps:

1. Instantiate logic functions with a Component Instantiation, and include a Component Declaration for each function. Altera provides simulation models for the following types of logic functions:
   - Primitives in the Design Compiler & FPGA Compiler Technology Libraries. Go to Primitive & Old-Style Macrofunction Instantiation Example for VHDL for an example.
   - Architecture Control Logic functions in the **alt_mf** library, which includes the `a_8count`, `a_8mcomp`, `a_8fadd`, and `a_81mux` functions. See MAX+PLUS II Architecture Control Logic Function Instantiation Example for VHDL for an example.
   - The DesignWare up/down counter function (`DW03_updn_ctr`). Go to DesignWare Up/Down Counter Function Instantiation Example for VHDL for an example.
   - RAM and ROM functions generated with the **genmem** utility. Go to Instantiating RAM & ROM Functions in VHDL for instructions.
   - The `clklock` megafunction, which is supported for selected FLEX 10K devices. This function is generated with the **genclklk** utility. Go to Instantiating the clklock Megafunction in VHDL or Verilog HDL for instructions.
   - MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.

     You can also instantiate any other Altera macrofunction or non-parameterized megafunction, i.e., functions not listed above, for which no simulation models or technology library support is available. These functions are treated as "black boxes" during processing with the Design Compiler or FPGA Compiler. See Primitive & Old-Style Macrofunction Instantiation Example for VHDL for an example.

   For information on MAX+PLUS II primitives, megafunctions, and macrofunctions, choose **Primitives**, **Megafunctions/LPM**, or **Old-Style Macrofunctions** from the MAX+PLUS II Help menu. When searching for information on the **alt_mf** library functions, drop the initial "`a_`" from the function name.

2. (Optional) If you instantiate a "black box" logic function for which no simulation/techology library support is

available, create a hollow-body design description in order to prevent the Design Compiler or FPGA Compiler from issuing a warning message. See Primitive & Old-Style Macrofunction Instantiation Example for VHDL for an example.

If you instantiate a "black box" logic function, you must create a Library Mapping File (**.lmf**) to map the function to an equivalent MAX+PLUS II function before you compile the project with the MAX+PLUS II software. See Primitive & Old-Style Macrofunction Instantiation Example for VHDL for an example.

3. Once you have created a VHDL design, you can analyze it, synthesize it, (optional) perform a functional simulation, and generate an EDIF netlist file that can be imported into the MAX+PLUS II software. Go to the following topics for instructions:

    - Synthesizing & Optimizing VHDL & Verilog HDL Projects with Synopsys Software
    - Performing a Pre-Routing or Function Simulation with VSS Software

Installing the Altera-provided MAX+PLUS II/Synopsys Logic interface on your computer automatically creates the following VHDL sample files:

- **/usr/maxplus2/examples/mentor/examples/ministate.vhd**
- **/usr/maxplus2/examples/mentor/examples/count8.vhd**
- **/usr/maxplus2/examples/mentor/examples/tstrom.vhd**

Related Links:

- Go to Compiling Projects with MAX+PLUS II Software in these MAX+PLUS II ACCESS^{SM} Key topics for related information.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Synthesizing & Optimizing VHDL Files with Synergy Software

You can use Cadence Synergy software to synthesize and optimize your VHDL files and convert them to EDIF input files that can be processed by the MAX+PLUS® II Compiler. The information presented here describes only how to use VHDL files that have been processed by Synergy software. For information on direct MAX+PLUS II support for VHDL Design Files, go to MAX+PLUS II VHDL Help.

To process a VHDL file with Synergy software for use with MAX+PLUS II software, go through the following steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment.
2. Create a VHDL file <*design name*>**.vhd** using the MAX+PLUS II Text Editor or another standard text editor and save it in a working directory. Go to Creating VHDL Designs for Use with MAX+PLUS II Software for more information on VHDL design entry.
3. Start Synergy by typing `synergy -lang vhdl` ↵ at a UNIX prompt from the working directory.
4. Analyze your source file <*design name*>**.vhd**:
    1. Choose **Analyze Files** (File menu) to open the **Select Design** dialog box.
    2. Click on the **Analyze Files** tab.
    3. Select the design name from the *Files* list.
    4. Choose **Analyze** to analyze the source file(s).
5. Choose the **Select Design** tab from the **Select Design** dialog box and specify the following options:
    1. Select the design architecture from the hierarchical list. The design architecture should appear in the *Design* box.
    2. Specify <*design name*>**.run1** as the *Run Directory*.
    3. Type `alt_syn` as the *Target Library* name.
    4. (Optional) If you want to use the Synergy library of parameterized modules (LPM) synthesis capability, choose the **Macro Libraries** ellipse button and select *lpm_syn* in the *Select From* box.
    5. (Optional) If you want to view a synthesized schematic in Concept or Composer, go through the following steps:
        1. Choose **Schematic Generation** (Utilities menu).
        2. Select either *Concept* or *Composer* in the *Generate From* box.
        3. Type `alt_max2` in the *Symbol Libraries* box.

4. Choose **Apply**, then **Close**.

6. Choose the **Select Design** button from the **Select Design** window.

7. Indicate to the Synergy software that any `clklock` megafunction or any macrofunction instantiated in your VHDL design is a "black box" that must pass untouched through the EDIF netlist file:

    1. Choose **Synthesis** (Constraints menu), then choose **Hierarchy Control**.

    2. Select the module or instance name from the hierarchical **View** list for *Module/Instance*.

    3. Turn on *Maintain Option* in the *Synthesis Constraints* box.

    4. Select *Module/Instance* and *Tree Below* in the *Apply To* box.

    5. Choose **Apply**.

    6. Repeat steps a through e for each instance of the function.

8. Choose **Synthesize** (Synthesis menu) from the Synergy window and specify the following options:

    1. Click on the **Synthesize** tab.

    2. Turn on the *Generate Schematic* option.

    3. Select either *Composer* or *Concept* from the *Type* list box.

    4. Choose **Synthesize** to start synthesizing your design.

9. Generate an EDIF netlist file that can be compiled with MAX+PLUS II software, as described in Converting VHDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the **vlog2alt** or altout **Utility**.

10. Process the <*design name*>**.edf** file with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample VHDL files:

- **/usr/maxplus2/examples/cadence/example9/count4.vhd**
- **/usr/maxplus2/examples/cadence/example10/adder16.vhd**

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Using Cadence Verilog-XL & MAX+PLUS II Software

The following topics describe how to use the Cadence Verilog-XL software with MAX+PLUS® II software. Click on one of the following topics for information:

This file is suitable for printing only. It does not contain hypertext links that allow you to jump from topic to topic.

Setting Up the MAX+PLUS II/Cadence Working Environment

- Software Requirements
- MAX+PLUS II Directory Structure
- MAX+PLUS II/Cadence Interface File Organization

Functional Simulation

- Performing a Functional Simulation of a Concept Schematic with the **hdlconfig** Utility & Verilog-XL Software
- Performing a Functional Simulation of a Concept Schematic with VerilogLink & Verilog-XL Software

Timing Simulation

- Project Simulation Flow
- Initializing Registers in VHDL & Verilog Output Files for Power-Up before Simulation
- Performing a Timing Simulation with Verilog-XL Software

Related Topics:

Go to the following MAX+PLUS II ACCESS℠ Key topics for related information:

- Using Cadence Concept & MAX+PLUS II Software
- Compiling Projects with MAX+PLUS II Software
- Programming Altera Devices

Go to the following topics, which are available on the web, for additional information:

- MAX+PLUS II Development Software
- Altera Programming Hardware
- Cadence web site (http://www.cadence.com)

Setting Up the MAX+PLUS II/Cadence Working Environment

To use MAX+PLUS® II software with Cadence software, you must first install the MAX+PLUS II software, then establish an environment that facilitates entering and processing designs. The MAX+PLUS II/Cadence interface is installed automatically when you install the MAX+PLUS II software on your computer. Go to *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual for more information on installation and details on the directories that are created during MAX+PLUS II installation. Go to MAX+PLUS II/Cadence Interface File Organization for information about the MAX+PLUS II/Cadence directories that are created during MAX+PLUS II

installation.

The information presented here assumes that you are using the C shell and that your MAX+PLUS II system directory is **/usr/maxplus2**. If not, you must use the appropriate syntax and procedures to set environment variables for your shell.

To set up your working environment for the MAX+PLUS II/Cadence interface, follow these steps:

1. Ensure that you have correctly installed the MAX+PLUS II and Cadence software versions described in the MAX+PLUS II/Cadence Software Requirements.

2. Add the following environment variables to your **.cshrc** file:

   ```
   setenv ALT_HOME /usr/maxplus2 ↵

   setenv CDS_INST_DIR <Cadence system directory path> ↵
   ```

3. Add the **$ALT_HOME/cadence/bin** and **$CDS_INST_DIR/tools/bin** directories to the `PATH` environment variable in your **.cshrc** file. Make sure these paths are placed before the Cadence hierarchy path.
4. Add **/usr/dt/lib** and **/usr/ucb/lib** to the `LD_LIBRARY_PATH` environment variable in your **.cshrc** file.
5. Create a new **cds.lib** file in your working directory or edit an existing one so that it includes all of the following lines that apply to the Cadence tools you have installed:

   ```
   DEFINE alt_syn ${ALT_HOME}/simlib/concept/alt_syn

   DEFINE lpm_syn ${ALT_HOME}/simlib/concept/lpm_syn

   DEFINE alt_lpm ${ALT_HOME}/simlib/concept/alt_lpm

   DEFINE alt_mf ${ALT_HOME}/simlib/concept/alt_mf

   DEFINE alt_max2 ${ALT_HOME}/simlib/concept/alt_max2

   DEFINE alt_max2 ${ALT_HOME}/simlib/composer/alt_max2/alt_max2

   DEFINE alt_vtl $ALT_HOME/simlib/concept/alt_vtl/lib

   DEFINE altera $ALT_HOME/simlib/concept/alt_mf/lib

   SOFTINCLUDE $CDS_INST_DIR/tools/leapfrog/files/cds.lib

   DEFINE <design name>.
   ```

6. Copy the **/usr/maxplus2/maxplus2.ini** file to your **$HOME** directory:

   ```
   cp /usr/maxplus2/maxplus2.ini $HOME ↵

   chmod u+w $HOME/maxplus2.ini ↵
   ```

   The **maxplus2.ini** file contains both Altera- and user-specified initialization parameters that control the MAX+PLUS II software, such as MAX+PLUS II symbol and logic function library paths and the current project name. The MAX+PLUS II installation procedure creates and copies the **maxplus2.ini** file to the **/usr/maxplus2** directory.

   Normally, you do not have to edit your local copy of **maxplus2.ini** because the MAX+PLUS II software updates the file automatically whenever you change any parameters or settings. However, if you move the **max2lib** and **max2inc** library subdirectories, you must update the file. Go to "Creating

& Using a Local Copy of the **maxplus2.ini** File" in MAX+PLUS II Help for more information.

7. If you are using Concept on a Sun SPARCstation running SunOS, go to Setting Up the MAX+PLUS II/Cadence Concept Work Environment for a Sun SPARCstation Running SunOS Software to install the **redifnet** EDIF netlist reader utility.
8. If you are using Synergy software, edit the **hdl.var** file located in your working directory to include the following line:

   `DEFINE work` <*design name*> ↵

9. Set up an appropriate directory structure for the tool(s) you are using. See the following topics for information:
   - Composer Project File Directory Structure
   - Concept & RapidSIM Local Work Area Directory Structure

Related Topics:

Go to the following topics, which are available on the web, for additional information:

- ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
- This manual is also available in 4 parts:
  - Preface & Section 1: MAX+PLUS II Installation
  - Section 2: MAX+PLUS II - A Perspective
  - Section 3: MAX+PLUS II Tutorial
  - Appendices, Glossary & Index

MAX+PLUS II/Cadence Software Requirements

The following table shows the software applications that are used to generate, process, synthesize, and verify a project with MAX+PLUS® II and Cadence software:

	Cadence	Altera
version 97A: Concept Composer ValidCOMPILER **concept2alt** **vlog2alt** **altout**	VerilogLink Synergy HDL Direct (Concept 2.0 or later) Non-Graphic Simulation Environment (SE) RapidSIM, Verilog-XL, or Leapfrog **redifnet** (SunOS only)	MAX+PLUS II version 9.4

☞ The MAX+PLUS II **read.me** file provides up-to-date information on which versions of Cadence software applications are supported by the current version of MAX+PLUS II. It also provides information on installation and operating requirements. You should read the **read.me** file on the CD-ROM before installing the MAX+PLUS II software. After installation, you can open the **read.me** file from the MAX+PLUS II Help menu.

MAX+PLUS II Directory Structure

In the MAX+PLUS® II software, a project name is the name of a top-level design file, without the filename extension. This design file can be an EDIF, Verilog HDL, or VHDL netlist file; an AHDL Text Design File (TDF); or any other MAX+PLUS II-supported design file. The EDIF netlist file must be created by the **altout** or **concept2alt** utility and imported into the MAX+PLUS II software as an EDIF Input File (**.edf**).

Project design files and output files are stored in the project directory, with the exception of standard library functions provided by Altera or another EDA tool vendor. The MAX+PLUS II software stores the connectivity data on the links between design files in a hierarchical project in a Hierarchy Interconnect File (**.hif**), but refers to the entire project only by its project name. The MAX+PLUS II Compiler uses the HIF to build a single, fully flattened project database that integrates all design files in a project hierarchy.

MAX+PLUS II/Cadence Interface File Organization

Table 1 shows the MAX+PLUS® II/Cadence interface subdirectories that are created in the MAX+PLUS II system directory (by default, the **/usr/maxplus2** directory) during MAX+PLUS II installation. For information on the other directories that are created during MAX+PLUS II installation, see "MAX+PLUS II File Organization" in *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual.

Table 1. MAX+PLUS II Directory Organization

Directory	Description
./lmf	Contains the Altera-provided Library Mapping File, **cadence.lmf**, that maps Cadence logic functions to equivalent MAX+PLUS II logic functions.
./examples/cadence	Contains the sample files for Cadence software discussed in these ACCESS℠ Key Guidelines.
./cadence	Contains the AMPLE userware for the MAX+PLUS II/Cadence interface.
./simlib/concept/alt_max2	Contains the MAX+PLUS II primitives, including `CARRY`, `CASCADE`, `EXP`, `GLOBAL`, `LCELL`, `SOFT`, `OPNDRN`, `DFFE` (D flipflop with Clock Enable), and `DFFE6K` (D flipflop with Clock Enable and both Clear and Preset for FLEX® 6000 devices only) for use with Concept software.
./simlib/composer/alt_max2	Contains the MAX+PLUS II primitives, including `CARRY`, `CASCADE`, `EXP`, `GLOBAL`, `LCELL`, `SOFT`, `OPNDRN`, `DFFE` (D flipflop with Clock Enable), and `DFFE6K` (D flipflop with Clock Enable and both Clear and Preset for FLEX 6000 devices only) for use with Composer software.
./simlib/concept/alt_lpm	Contains the MAX+PLUS II megafunctions, including library of parameterized modules (LPM) functions, for use with Concept software.
./simlib/concept/max2sim	Contains the MAX+PLUS II/Concept simulation model library, **max2_sim**, for use with RapidSIM software.
./simlib/concept/alt_syn	Contains the MAX+PLUS II synthesis library, **alt_syn**, for use with Synergy and Concept software, and the **vlog2alt** utility.
./simlib/composer/alt_syn	Contains the MAX+PLUS II synthesis library, **alt_syn**, for use with Synergy and Composer software.
./simlib/concept/lpm_syn	Contains the Cadence LPM library, **lpm_syn**, for use with Synergy and Concept software.
./simlib/composer/lpm_syn	Contains the Cadence LPM library, **lpm_syn**, for use with Synergy and Composer software.
./simlib/concept/alt_mf	Contains the MAX+PLUS II VHDL logic function library. (`a_8count` is for the MAX® 7000 and MAX 9000 device families only.)
./simlib/concept/edifnet/templates	Contains template files for Concept directives, i.e., **global.cmd**, **compiler.cmd**, **vloglink.cmd**, **verilog.cmd**, and **master.local**.
./simlib/concept/alt_max2/verilogUdps	Contains Verilog HDL modules that are the equivalent of the primitives contained in **alt_max2** library for use with Concept software.
./simlib/composer/alt_max2/verilogUdps	Contains Verilog HDL modules that are the equivalent of the primitives

	contained in **alt_max2** library for use with Composer software.
./simlib/concept/alt_vtl **./simlib/composer/alt_vtl**	Contains VITAL library source files for use with Concept or Composer software.
./simlib/composer/alt_max2/verilog	Contains simulation modules for all symbols in the **alt_max2** Composer library.

Related Topics:

Go to the following topics, which are available on the web, for additional information:

- ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
- This manual is also available in 4 parts:
  - Preface & Section 1: MAX+PLUS II Installation
  - Section 2: MAX+PLUS II - A Perspective
  - Section 3: MAX+PLUS II Tutorial
  - Appendices, Glossary & Index
- FLEX Devices
- MAX Devices
- Classic Device Family

Performing a Functional Simulation of a Concept Schematic with the hdlconfig Utility & Verilog-XL Software

You can perform a functional simulation of a Concept schematic with the **hdlconfig** utility and Verilog-XL software before compiling your project with the MAX+PLUS® II software.

To functionally simulate a Concept schematic, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment.
2. Create a Concept schematic and save it in your working directory, as described in Creating Concept Schematics for Use with MAX+PLUS II Software.
3. Use the **hdlconfig** utility to create a Verilog HDL text file that contains the entire design. Type the following command at the UNIX prompt from the */<working directory>/<design name>/***source** directory:

   `hdlconfig -a -c -r` *<design name>* `-o` *<design name>*`.v logic verilog_lib` ↵

4. If your design contains RAM or ROM functions (e.g., `lpm_ram_dq`, `lpm_ram_io`, `lpm_rom`, `scfifo`, `dcfifo`, `altdpram`, and `csdpram`), run the **vconfig** utility to link the object **convert_hex2ver.o** to build a new Verilog-XL file that supports these functions by following these steps:
   1. Create a copy of the Verilog executable file by typing the following command at the UNIX prompt:

      ```
      cp -p $CDS_INST_DIR/tools/verilog/bin/verilog $CDS_INST_DIR/tools /verilog/bin/
      verilog.bak. ↵
      ```

   2. Type `vconfig` ↵ at the UNIX prompt from the **/usr/maxplus2/cadence/bin** directory to start the script.
   3. Accept `cr_vlog` as the name of the output script.
   4. Accept `1` as the stand-alone target.
   5. Type `new_verilog` as the name for the Verilog-XL target.
   6. Respond `Yes` when you are prompted to compile for the Verilog-XL environment.
   7. Respond `No` when you are prompted to include the Dynamic LAI, STATIC LOGIC AUTOMATION, LMSI HARDWARE MODELER, Verilog Mixed-Signal, and CDC interfaces in this executable.
   8. Respond `Yes` when you are prompted to include the Standard Delay File Annotator (SDF).

9. Specify **/usr/maxplus2/verilog/veriuser.c** when you are asked the name of the user template file. For more information about the contents of the **veriuser.c** file, you can refer to the **veriuser.doc** file, which is available in the Cadence *Openbook* product documentation. To locate this document, start *Openbook*, and choose **Alphabetical List of Products** from the main menu. Scroll through the pages until you locate the *PLI 1.0 User Guide & Reference* in the PLI section, and then continue to scroll through the document until you locate the **veriuser.doc** file under "Section A" and "PLI Code Examples."
10. When you are asked the name of files to be linked with the Verilog-XL simulator, specify the hexadecimal (Intel-format) conversion file **/usr/maxplus2/cadence/share/verilog/convert_hex2ver.o**, followed by a single period (.).
11. Run the output script **cr_vlog** to build the new Verilog-XL executable in the **/usr/maxplus2/cadence/bin** directory. Make sure that the `$CDS_INST_DIR/tools/bin` path appears at the beginning of the `PATH` statement in the **.cshrc** file.
12. If your C language library installation is different from the default location **/usr/lang/SC3.0.1**, type the following command at the UNIX prompt:

    `setenv C_DIR` <*C language library installation directory*> ↵

13. If successful, replace the old Verilog executable file with the new one by typing the following command at the UNIX prompt:

    `cp -p new_verilog $CDS_INST_DIR/tools/verilog/bin/verilog` ↵

5. Generate the stimulus file for the design and start the Verilog-XL simulator by typing the following command at the UNIX prompt from the /<*working directory*>/<*design name*>/**source** directory:

   `verilog -y /usr/maxplus2/simlib/concept/alt_max2/verilogUdps +libext+.v+.V` <*stimulus file name*> <*design name*>`.v` ↵

6. When you are ready to compile the project, generate an EDIF netlist file <*design name*>**.edf** with the **concept2alt** utility, as described in Converting Concept Schematics into MAX+PLUS II-Compatible EDIF Netlist Files with the **concept2alt** Utility.
7. Process the <*design name*>**.edf** file with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Performing a Functional Simulation of a Concept Schematic with VerilogLink & Verilog-XL Software

You can perform a functional simulation of a Concept schematic with VerilogLink and Verilog-XL software before compiling your project with the MAX+PLUS® II software.

To functionally simulate a Concept schematic, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment.
2. Create a Concept schematic and save it in your working directory, as described in Creating Concept Schematics for Use with MAX+PLUS II Software.
3. Generate the **global.cmd**, **vloglink.cmd**, **verilog.cmd**, and **expansion.dat** directive files.
4. Type `vloglink` <*design name*> ↵ from the /<*working directory*>/**source** directory to create a **vloglink.v** file from the Concept schematic.
5. Generate the stimulus file for the design and start the Verilog-XL simulator by typing the following command at the UNIX prompt from the /<*working directory*>/<*design name*>/**source** directory:

   `verilog -y /usr/maxplus2/simlib/concept/alt_max2/verilogUdps +libext+.v+.V` <*stimulus file name*> `vloglink.v` ↵

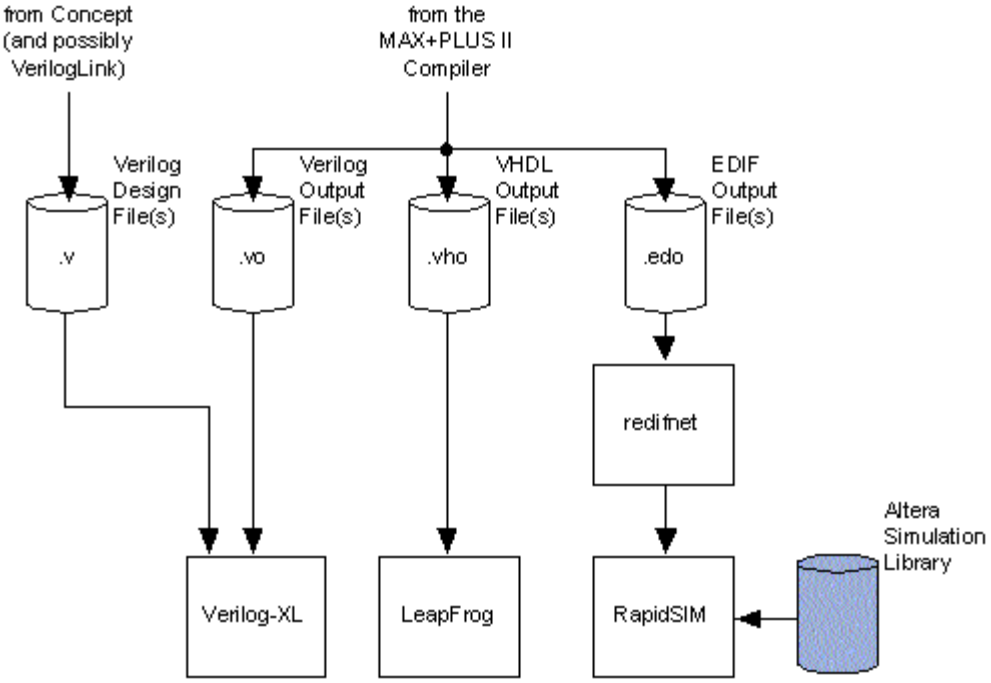
6. When you are ready to compile the project, generate an EDIF netlist file <*design name*>**.edf** with the **concept2alt** utility, as described in Converting Concept Schematics into MAX+PLUS II-Compatible EDIF Netlist Files with the **concept2alt** Utility .
7. Process the <*design name*>**.edf** file with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Project Simulation Flow

Figure 1 shows the project simulation flow for the MAX+PLUS® II/Cadence interface.

Figure 1. MAX+PLUS II/Cadence Project Simulation Flow

Altera-provided items are shown in blue.



Initializing Registers in VHDL & Verilog Output Files for Power-Up before Simulation

Altera provides the **add_dc** script, which is availiable in the MAX+PLUS II system directory, to allow you to process MAX+PLUS II-generated Verilog Output Files (**.vo**) and VHDL Output Files (**.vho**) to prepare these files for simulation with another EDA tool. The add_dc script runs the **add_dclr** utility, which inserts a `device_clear` signal that is used for power-up initialization of all registers or flipflops in the design.

The script adds in a top-level signal named `device_clear` and connects it to the `CLRN` pin in all flipflops that should initialize to 0, and to the `PRN` pin of all flipflops that should initialize to 1. If the `CLRN` or `PRN` pin of a flipflop is already being used (i.e., is already connected to a signal), the script modifies the Verilog Output File or VHDL Output File so that the `AND` of the original signal and the `device_clear` pin feed the `CLRN` or `PRN` pin.

To use the **add_dc** script to process Verilog Output Files and VHDL Output Files before simulation with another EDA tool, follow these steps:

1. Make sure that your design file is located in the current directory, or change to the directory in which the design file is located.

2. Type the following command at the command prompt:

   ¥<*path name of* ***add_dc.bat*** *file*>¥add_dc <*design name*> <*path name of* ***add_dclr.exe*** *file*> ↵

For example, if the both the **add_dc.bat** and the **add_dclr.exe** files are located in the **d:¥maxplus2¥exew** directory, and the **d:¥maxplus2¥exew** directory is specified in the search path, you can type the following command at a command prompt to add a `device_clear` signal to a design named `myfifo` in the file **myfifo.vo**:

```
add_dc myfifo d:¥maxplus2¥exew ↵
```

1. The **add_dc** script gives a message if the directory contains both a VHDL Output File and a Verilog Output File with the same name (<*design name*>**.vo** and <*design*>**.vho**). You should delete or rename whichever of those files should not have the `device_clear` signal added. The **add_dc** script can modify only one design file at a time.

2. When the **add_dc** script processes the Verilog Output File or VHDL Output File, it creates a backup copy of the original file, with the extension **.ori**.

3. The **add_dc** script works only for Verilog Output Files and VHDL Output Files that are generated by MAX+PLUS II.

After you have used the **add_dc** script and are ready to simulate the resulting Verilog Output File or VHDL Output File with another EDA tool, you should assert the active low `device_clear` pin for a period of time that is long enough for the design to initialize. You can then de-assert the pin, and apply simulation vectors to the design.

Performing a Timing Simulation with Verilog-XL Software

Once the MAX+PLUS® II software has compiled a project and generated a Verilog Output File (**.vo**), you can perform a timing simulation using Cadence Verilog-XL software.

To simulate Verilog output files with the Verilog-XL timing simulator, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment.
2. Generate Verilog Output Files (**.vo**), as described in Compiling Projects with MAX+PLUS II Software. The MAX+PLUS II Compiler generates the <*design name*>**.vo** and **alt_max2.vo** files for use with Verilog-XL software.
3. Using any standard text editor, create a stimulus file that includes test vectors for your design.
4. Start the Verilog-XL simulator and simulate your Verilog output files by typing the following command at the UNIX prompt:

   `verilog` <*stimulus filename(s)*> <*design name*> `alt_max2.vo` ↵

Compiling Projects with MAX+PLUS II Software

The MAX+PLUS® II Compiler can process design files in a variety of formats. This topic describes how to use MAX+PLUS II software to compile projects in which the top-level design file is an EDIF Input File (with the extension **.edf**).

Related Topics:

- Go to MAX+PLUS II Help for information on compiling VHDL and Verilog HDL, design files directly with the MAX+PLUS II Compiler.
- Go to Running Synopsys Compilers from MAX+PLUS II Software for information on running the Synopsys Design Compiler or FPGA Compiler software on a VHDL or Verilog HDL design from within the MAX+PLUS II Compiler window.

To compile a design (also called a "project") with MAX+PLUS II software, go through the following steps:

1. Create design files that are compatible with the MAX+PLUS II software and convert them into EDIF Input Files with the extension **.edf**. Specific instructions for some tools are described in these MAX+PLUS II ACCESS^{SM} Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your design entry or synthesis and optimization tool.
2. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, create a mapping for each function in a Library Mapping File (**.lmf**) to map the custom symbol to the corresponding EDIF Input File, AHDL Text Design File (**.tdf**), or other MAX+PLUS II-supported design file. These custom functions are represented in design files as hollow-body symbols or "black box" HDL descriptions.

   ☞ Go to "Library Mapping Files (**.lmf**)" in MAX+PLUS II Help for more information.
3. Open MAX+PLUS II and specify the name of your top-level design file as the project name with the **Project Name** command (File menu). If you open an HDL file in the MAX+PLUS II Text Editor, you can choose the **Project Set Project to Current File** command (File menu) instead.

   ☞ You can also compile a project from a command line. However, the first time you compile a project, the settings you need to specify are easier to specify from within the MAX+PLUS II software. After you have run the graphical user interface for the MAX+PLUS II software at least once, you can more easily use the command-line **setacf** utility to modify options in the Assignment & Configuration File (**.acf**) for the project. Type `setacf -h` ↵ and `maxplus2 -h` ↵ for descriptions of **setacf** and MAX+PLUS II command-line syntax.
4. Choose **Device** (Assign menu) and select the target Altera device family in the *Device Family* drop-down list box. If you wish to implement the design logic in a specific device, select it in the *Devices* box. Otherwise, select *AUTO* to allow the MAX+PLUS II Compiler to choose the best device(s) in the current device family. If your design entry or synthesis and optimization tool required you to specify a target family and/or device, specify the same information in this dialog box. For information on partitioning logic among multiple devices, go to MAX+PLUS II Help. Choose **OK**.
5. Open the Compiler window by choosing the **Compiler** command (MAX+PLUS II menu). Go through the following steps to specify the options necessary to compile the design file(s) in your project:
   1. Ensure that all EDIF netlist files have the extension **.edf** and choose **EDIF Netlist Reader Settings** (Interfaces menu).
   2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor. This name should be the name of the vendor whose tool(s) you used to create the EDIF netlist files. If your vendor name does not appear, select *Custom* instead.

      ☞ If you are compiling a design created with Synopsys FPGA Express software, select *Synopsys*, choose the **Customize** button, enter <*project name*>**.lmf** in the *LMF #1* box, choose **OK**, and skip to step 6.
   3. If you selected an existing vendor name in the *Vendor* box and your project contains design files that require custom LMF mappings, choose the **Customize** button to expand the dialog box to show all settings. Turn on the *LMF #2* checkbox and type your custom LMF's filename in the corresponding text box, or select a name from the *Files* box. The selection in the *Vendor* box will change to *Custom* and all settings will be retained until you change them again.
   4. If you selected *Custom* in the *Vendor* box, choose the **Customize** button to expand the dialog box to show all settings. Any previously defined custom settings will be displayed. Under *Signal Names*, type one or more names with up to 20 total name characters in the *VCC* or *GND* box if your EDIF Input File(s) use one or more names other than `VCC` or `GND` for the global high or low signals. Multiple signal names must be separated by either a comma (,) or a space. Under *Library Mapping Files*, turn on the

LMF #1 checkbox and type a filename in the text box following it, or select a name from the *Files* box. If necessary, specify another LMF name in the *LMF #2* box. Go to MAX+PLUS II Help for detailed information on the settings available in the **EDIF Netlist Reader Settings** dialog box.

5. Choose **OK**.

6. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, you may need to ensure that all files are present in your project directory, i.e., the same directory as the top-level design file. Otherwise, you must specify the directories containing these files as user libraries with the **User Libraries** command (Options menu).
7. Follow all guidelines that apply to your design entry or synthesis and optimization tool:
   - Exemplar Logic Galileo Extreme-Specific Compiler Settings
   - Synopsys DesignWare-Specific Compiler Settings
   - Converting Synopsys FPGA Compiler & Design Compiler Timing Constraints into MAX+PLUS II-Compatible Format with the **syn2acf** Utility
   - Synplicity Synplify-Specific Compiler Settings
8. If you wish to generate EDIF, VHDL, or Verilog HDL output files for post-compilation simulation or timing analysis with another EDA tool, go through the following steps:
   1. (Optional) Turn on the **Optimize Timing SNF** command (Processing menu) to reduce the size of the output file(s). Turning on this command can reduce the size of output netlists by up to 30%.

      ☞ This command does not create optimized timing SNFs on UNIX workstations. However, a non-optimized timing SNF provides the same functional and timing information as an optimized timing SNF.

   2. If you wish to generate EDIF Output Files (**.edo**), go through these steps:
      1. Turn on the **EDIF Netlist Writer** command (Interfaces menu). Then choose the **EDIF Netlist Writer Settings** command (Interfaces menu).
      2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor and choose **OK**. If your vendor name does not appear, select *Custom* instead and specify the settings that are appropriate for your simulation or timing analysis tool. Go to MAX+PLUS II Help for detailed information on the options available in the **EDIF Netlist Writer Settings** dialog box.
      3. To generate an optional Standard Delay Format (SDF) Output File (**.sdo**), choose the **Customize** button to expand the dialog box to show all settings. Select one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**.

      The filenames of the EDIF Output File(s) and optional SDF Output File(s) are the same as the user-defined chip name(s) for the project; if no chip names exist, the Compiler assigns filenames that are based on the project name. For a multi-device project, the Compiler also generates a top-level EDIF Output File that is uniquely identified by "**_t**" appended to the project name. In addition, the Compiler automatically generates a VHDL Memory Model Output File, *<project name>***.vmo**, when it generates an EDIF Output File that contains memory (RAM or ROM).

   3. If you wish to generate VHDL Output Files (**.vho**), turn on the **VHDL Netlist Writer** command (Interfaces menu). Then choose **VHDL Netlist Writer Settings** command (Interfaces menu). Select *VHDL Output File (.vho)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF ver. 2.1 files contain timing delay information that allows you to perform back-annotation simulation in VHDL with VITAL-compliant simulation libraries. The VHDL Output Files generated by the Compiler have the extension **.vho**, but are otherwise named in the same way as the EDIF Output Files described above.
   4. If you wish to generate Verilog HDL Output Files (**.vo**), turn on the **Verilog Netlist Writer** command (Interfaces menu). Then choose **Verilog Netlist Writer Settings** command (Interfaces menu). Select *Verilog Output File (.vo)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF Output Files contain timing delay information that allows you to perform back-annotation simulation in Verilog HDL. The Verilog Output Files generated by the Compiler have the extension **.vo**, but are otherwise named in the same way as the EDIF Output Files described above.
9. To run the MAX+PLUS II Compiler, choose the **Project Save & Compile** command (File menu) or choose the **Start** button in the Compiler window.

   ☞ See step 3 for information on running MAX+PLUS II software from the command line.

10. Once you have compiled the project with the MAX+PLUS II Compiler, you can use the VHDL, Verilog HDL, or EDIF output file(s), and the optional SDF Output File(s) (**.sdo**) to perform timing analysis or timing simulation with another EDA tool. Specific instructions for some tools are described in these MAX+PLUS II ACCESS Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your EDA tool.

The MAX+PLUS II Compiler also generates a Report File (**.rpt**), a Pin-Out File (**.pin**), and one or more of the following files for device programming or configuration:

- JEDEC Files (**.jed**)
- Programmer Object Files (**.pof**)
- SRAM Object Files (**.sof**)
- Hexadecimal (Intel-format) Files (**.hex**)
- Tabular Text Files (**.ttf**)

Related Topics:

- Go to *Compiler Procedures* in MAX+PLUS II Help for information on other available Compiler settings.
- Go to *Programmer Procedures* in MAX+PLUS II Help for instructions on creating other types of programming files and on programming or configuring Altera devices.
- Go to Back-Annotating MAX+PLUS II Pin Assignments to Design Architect Symbols for information on back-annotating pin assignments in Mentor Graphics Design Architect schematics.
- Go to Programming Altera Devices for information on the different programming hardware options for Altera device families.

Go to the following topics, which are available on the web, for additional information:

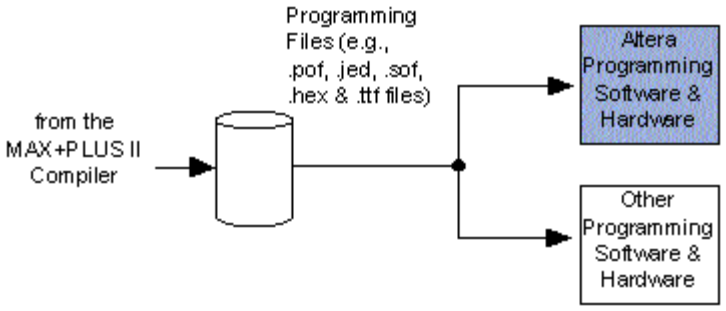
- MAX+PLUS II Development Software
- Altera Programming Hardware

Programming Altera Devices

Once you have successfully compiled and simulated a project with the MAX+PLUS® II software, you can program an Altera® device and test it in the target circuit. Figure 1 shows the device programming flow for MAX+PLUS II software.

Figure 1. MAX+PLUS II Device Programming Flow

Altera-provided items are shown in blue.



You can program devices with Altera programming hardware and MAX+PLUS II Programmer software installed

on a 486- or Pentium-based PC or a UNIX workstation, or with programming hardware and software available from other manufacturers. Table 1 shows the available Altera programming hardware options on PCs and UNIX workstations.

Table 1. Altera Programming Hardware

Programming Hardware Option	PCs	UNIX Work-stations	MAX® 3000A Devices	Classic® & MAX 5000 Devices	MAX 7000 & MAX 7000E Devices	MAX 7000A, MAX 7000AE, MAX 7000B, MAX 7000S MAX 9000 & MAX 9000A Devices	FLEX® 6000, FLEX 6000A, FLEX 8000, FLEX 10K, FLEX 10KA, FLEX 10KB, & FLEX 10KE Devices	In-System Programming/ Configuration
Logic Programmer card, PL-MPU Master Programming Unit, and device-specific adapters	✓		✓	✓	✓	✓		
BitBlaster™ Download Cable	✓	✓	✓			✓	✓	✓
ByteBlasterMV™ Download Cable	✓		✓			✓	✓	✓
MasterBlaster™ Download Cable	✓	✓	✓			✓	✓	✓

If you wish to transfer programming files from a UNIX workstation to a PC over a network with File Transfer Protocol (FTP) or other similar transfer programs, be sure to select binary transfer mode.

Programming hardware from other manufacturers varies, but typically consists of a device connected to one of the serial ports on the workstation. Various vendors, such as Data I/O and BP Microsystems, supply hardware and software for programming Altera devices.

Related Topics:

- Go to Compiling Projects with MAX+PLUS II Software for information on creating programming files.
- Go to the following topics, which are available on the web, for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware
  - FLEX Devices
  - MAX Devices
  - Classic Device Family

Last updated on December 6, 1999 for the MAX+PLUS II software version 9.4.

Performing a Functional Simulation of a Composer Schematic with Verilog-XL Software

You can perform a functional simulation of a Cadence Composer schematic with the Verilog-XL simulator before compiling your project with the MAX+PLUS® II software.

To functionally simulate a Composer schematic, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment.
2. Create a Composer schematic and save it in your working directory, as described in Creating Composer Schematics for Use with MAX+PLUS II Software.
3. In Composer, select **Simulation** from the *Tools* drop-down list.
4. Select **Verilog-XL** to start the *Verilog-XL Integration Control* window.
5. When you are ready to compile the project, generate an EDIF netlist file <*design name*>**.edf** with the **altout** utility, as described in Converting Composer Schematics into MAX+PLUS II-Compatible EDIF Netlist Files with the **altout** Utility.
6. Process the <*design name*>**.edf** file with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Converting Verilog HDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the vlog2alt Utility

You can use the **vlog2alt** utility to convert your Verilog HDL design into an EDIF netlist file. This file can then be imported into the MAX+PLUS® II software as an EDIF Input File with the extension **.edf**.

To convert a Verilog HDL design into an EDIF netlist file, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment.
2. Synthesize and optimize your Verilog HDL design with Synergy, as described in Synthesizing & Optimizing Verilog HDL Files with Synergy Software.
3. To convert your Verilog HDL design into an EDIF netlist file, type the following command at the UNIX prompt from your working directory:

   `vlog2alt <design name> -rundir max2 -vfiles <design name>.run1/syn.v` ↵
4. Process the <*design name*>**.edf** file with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample Verilog HDL files:

- **/usr/maxplus2/examples/cadence/example11/count8.v**
- **/usr/maxplus2/examples/cadence/example13/rom_test.v**

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Performing a Functional Simulation of a Concept Schematic with VerilogLink & Verilog-XL Software

You can perform a functional simulation of a Concept schematic with VerilogLink and Verilog-XL software before compiling your project with the MAX+PLUS® II software.

To functionally simulate a Concept schematic, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Cadence Working Environment.
2. Create a Concept schematic and save it in your working directory, as described in Creating Concept Schematics for Use with MAX+PLUS II Software.
3. Generate the **global.cmd**, **vloglink.cmd**, **verilog.cmd**, and **expansion.dat** directive files.
4. Type `vloglink` <*design name*> ↵ from the /<*working directory*>/**source** directory to create a **vloglink.v** file from the Concept schematic.
5. Generate the stimulus file for the design and start the Verilog-XL simulator by typing the following command at the UNIX prompt from the /<*working directory*>/<*design name*>/**source** directory:

   `verilog -y /usr/maxplus2/simlib/concept/alt_max2/verilogUdps +libext+.v+.V` <*stimulus file name*> `vloglink.v` ↵
6. When you are ready to compile the project, generate an EDIF netlist file <*design name*>**.edf** with the **concept2alt** utility, as described in Converting Concept Schematics into MAX+PLUS II-Compatible EDIF Netlist Files with the **concept2alt** Utility .
7. Process the <*design name*>**.edf** file with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Synthesizing & Optimizing Verilog HDL Files with Synergy Software

You can create and process Verilog HDL files and convert them into EDIF input files that can be processed by the MAX+PLUS® II Compiler. To process a Verilog HDL file with Synergy software for use with the MAX+PLUS II software, go through the following steps:

1. Be sure to set up your working environment correctly, as described in Setting up the MAX+PLUS II/Cadence Working Environment.
2. Create a Verilog HDL file <*design name*>**.v** using the MAX+PLUS II Text Editor or another standard text editor and save it in a working directory. Go to Creating Verilog HDL Designs for Use with MAX+PLUS II Software for more information on Verilog HDL design entry.
3. Start Synergy by typing `synergy -lang verilog` ↵ at a UNIX prompt from your working directory.
4. Choose **Select Design** (File menu) from the Synergy window and specify the following options:
    1. Select <*design name*>**.v** from the *Verilog Files* list.
    2. Choose the **Verilog Option** tab from the **Select Design** dialog box.
    3. Specify <*design name*>.**run1** as the *Run Directory*.
    4. Type `/usr/maxplus2/simlib/concept/alt_max2/`<*design name*>`/verilog_lib/verilog.v` <*working directory*>/ in the *Library Files (-v)* box.
    5. (Optional) If your design includes library of parameterized modules (LPM) functions, type `+define+SYNTH` in the *Other Compilations* box.
    6. Choose **Select Design**.
5. Choose the **Design** tab from the **Select Design** dialog box and set the target library:
    1. Type `alt_syn` as the *Target Library* name.
    2. (Optional) To use the Synergy LPM synthesis capability, type `lpm_syn` as the *Library* name in the *Macro Cell Library* box.
    3. Choose **OK**.
6. (Optional) To view the synthesized schematic in Concept or Composer, go through the following steps:
    1. Select **Schematic Generation** (Utilities menu).
    2. Select either *Concept* or *Composer* in the *Generate From* box.
    3. Type `alt_max2` in the *Symbol Libraries* box.
    4. Choose **Apply**, then **Close**.

7. Choose **Select Design** from the **Select Design** window.
8. Choose **Synthesize** (Synthesis menu) from the **Synergy** window and specify the following options:
    1. Click on the **Synthesize** tab.
    2. Turn on the *Generate Schematic* option.
    3. Select either *Composer* or *Concept* from the *Type* list box.
    4. Choose **Synthesize** to start synthesizing your design.
9. Generate an EDIF netlist file that can be compiled by the MAX+PLUS II Compiler, as described in Converting Verilog HDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files.
10. Process the <*design name*>**.edf** file with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Installing the Altera-provided MAX+PLUS II/Cadence interface on your computer automatically creates the following sample Verilog HDL files:

- **/usr/maxplus2/examples/cadence/example11/count8.v**
- **/usr/maxplus2/examples/cadence/example13/rom_test.v**

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Creating Verilog HDL Designs for Use with MAX+PLUS II Software

You can create Verilog HDL design files with the MAX+PLUS® II Text Editor or another standard text editor and save them in the appropriate directory for your project. The MAX+PLUS II Text Editor offers the following advantages:

- Verilog HDL templates are available with the **Verilog HDL Templates** command (Templates menu). These templates are also available in the ASCII **verilog.tmp** file, which is located in the **/usr/maxplus2** directory.
- If you use the MAX+PLUS II Text Editor to create your Verilog HDL design, you can use the **Syntax Coloring** command (Options menu). The Syntax Coloring feature displays keywords and other elements of text in text files in different colors to distinguish them from other forms of syntax.

Once you have created a Verilog HDL design, you can use the Design Compiler or FPGA Compiler to synthesize and optimize it, and then generate an EDIF netlist file that can be processed with the MAX+PLUS II software.

To create a Verilog HDL design that can be synthesized and optimized with the Design Compiler or FPGA Compiler, follow these steps:

1. Instantiate logic functions with a Module Instantiation, and include a Module Declaration for each function. Altera provides simulation models for the following types of logic functions:
   - Primitives in the Design Compiler & FPGA Compiler Technology Libraries. Go to Primitive & Old-Style Macrofunction Instantiation Example for Verilog HDL for an example.
   - Architecture Control Logic functions in the **alt_mf** library, which includes the `a_8count`, `a_8mcomp`, `a_8fadd`, and `a_81mux` functions. See MAX+PLUS II Architecture Control Logic Function Instantiation Example for Verilog HDL for an example.
   - RAM and ROM functions generated with the **genmem** utility. Go to Instantiating RAM & ROM Functions in VHDL for instructions.
   - The `clklock` megafunction, which is supported for selected FLEX 10K devices. This function is generated with the **genclklk** utility. Go to Instantiating the clklock Megafunction in VHDL or Verilog HDL for instructions.
   - MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.

     You can also instantiate any other Altera macrofunction or non-parameterized megafunction, i.e., functions not listed above, for which no simulation models or technology library support is available. These functions are treated as "black boxes" during processing with the Design Compiler or FPGA Compiler. See Primitive & Old-Style Macrofunction Instantiation Example for Verilog HDL for an example.

     For information on MAX+PLUS II primitives, megafunctions, and macrofunctions, choose **Primitives**, **Megafunctions/LPM**, or **Old-Style Macrofunctions** from the MAX+PLUS II Help menu. When searching for information on the **alt_mf** library functions, drop the initial "`a_`" from the function name.

2. (Optional) If you instantiate a "black box" logic function for which no simulation/techology library support is

available, create a hollow-body design description in order to prevent the Design Compiler or FPGA Compiler from issuing a warning message. See Primitive & Old-Style Macrofunction Instantiation Example for Verilog HDL for an example.

If you instantiate a "black box" logic function, you must create a Library Mapping File (**.lmf**) to map the function to an equivalent MAX+PLUS II function before you compile the project with the MAX+PLUS II software. See Primitive & Old-Style Macrofunction Instantiation Example for VHDL for an example.

3. Once you have created a VHDL design, you can analyze it, synthesize it, (optional) perform a functional simulation, and generate an EDIF netlist file that can be imported into the MAX+PLUS II software. Go to the following topics for instructions:

    - Synthesizing & Optimizing VHDL & Verilog HDL Projects with Synopsys Software
    - Performing a Pre-Routing or Function Simulation with VSS Software

Installing the Altera-provided MAX+PLUS II/Synopsys Logic interface on your computer automatically creates the following VHDL sample files:

- **/usr/maxplus2/examples/mentor/examples/ministate.v**
- **/usr/maxplus2/examples/mentor/examples/count8.v**
- **/usr/maxplus2/examples/mentor/examples/tstrom.v**

Related Links:

- Go to Compiling Projects with MAX+PLUS II Software in these MAX+PLUS II ACCESS^{SM} Key topics for related information.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Using Synopsys VSS & MAX+PLUS II Software

The following topics describe how to use the Synopsys VHDL System Simulator (VSS) and MAX+PLUS® II software. Choose one of the following topics for information:

This file is suitable for printing only. It does not contain hypertext links that allow you to jump from topic to topic.

Setting Up the MAX+PLUS II/Synopsys Working Environment

- Software Requirements
- Setting Up VSS Configuration Files
- Simulation Libraries
- MAX+PLUS II/Synopsys Interface File Organization
- MAX+PLUS II Project File Structure

Functional Simulation

- Design Entry Flow
- Performing a Pre-Routing or Functional Simulation with VSS Software

Timing Simulation

- Project Simulation Flow
- Performing a Timing Simulation with VSS Software

Related Topics:

- Go to the following topics in these MAX+PLUS II ACCESS℠ Key topics for related information:
  - Compiling Projects with MAX+PLUS II Software
  - Programming Altera Devices
  - Resynthesizing a Design Using the **alt.vtl** Library & a MAX+PLUS II SDF Output File
  - Using Synopsys Design Compiler or FPGA Compiler & MAX+PLUS II Software
  - Using Synopsys FPGA Express & MAX+PLUS II Software
  - Using Synopsys PrimeTime & MAX+PLUS II Software

- Go to the following topics for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware
  - Synopsys web site (http://www.synopsys.com)

Setting Up the MAX+PLUS II/Synopsys Working Environment

To use the MAX+PLUS® II software with Synopsys software, you must first install the MAX+PLUS II software, then establish an environment that facilitates entering and processing designs by modifying your Synopsys configuration files. The MAX+PLUS II/Synopsys interface is installed automatically when you install the MAX+PLUS II software on your workstation. Go to *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual for more information on installation and details on the directories that are created during MAX+PLUS II installation. Go to MAX+PLUS II/Synopsys Interface File Organization for information about the MAX+PLUS II/Synopsys directories that are created during MAX+PLUS II installation.

The information presented here assumes that you are using C shell and that your MAX+PLUS II system directory is **/usr/maxplus2**. If not, you must use the appropriate syntax and procedures to set environment variables for your shell.

To set up your working environment for the MAX+PLUS II/Synopsys interface, follow these steps:

1.
2. Ensure that you have correctly installed the MAX+PLUS II and Synopsys software versions described in the MAX+PLUS II/Synopsys Software Requirements.
3. Add technology, synthetic, and link library settings to your **.synopsys_dc.setup** configuration file, as described in Setting Up Design Compiler & FPGA Compiler Configuration Files.

   To use the DesignWare interface with FLEX® 6000, FLEX 8000, and FLEX 10K devices, follow the steps in Setting Up the DesignWare Interface.
4. Add simulation library settings to your **.synopsys_vss.setup** file, and analyze the libraries, as described in Setting Up VSS Configuration Files.
5. Add the **/usr/maxplus2/bin** directory to the `PATH` environment variable in your **.cshrc** file in order to run the MAX+PLUS II software.

   (Optional) Change the path in the first line of the perl script files, which are located in the `$ALT_HOME`**/synopsys/bin** directory to specify the correct path of your local perl executable file.

Related Topics:

- Go to the following topics for additional information:
  - FLEX Devices
  - ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
  - This manual is also available in 4 parts:
    - Preface & Section 1: MAX+PLUS II Installation
    - Section 2: MAX+PLUS II - A Perspective
    - Section 3: MAX+PLUS II Tutorial
    - Appendices, Glossary & Index

MAX+PLUS II/Synopsys Software Requirements

The following applications are used to generate, process, synthesize, and verify a project with MAX+PLUS® II and Synopsys software:

Synopsys	Altera

version 1998.02:
Design Compiler
FPGA Compiler
Design Analyzer (optional)
VHDL Compiler

HDL Compiler for Verilog
VHDL System Simulator (VSS) (optional)
PrimeTime version 1998.02-PT2.1(optional)

MAX+PLUS II
version 9.3

Compilation with the Synopsys Design Compiler and FPGA Compiler is available only on Sun SPARCstations running Solaris 2.4 or higher.

The MAX+PLUS II **read.me** file provides up-to-date information on which versions of Synopsys applications are supported by the current version of MAX+PLUS II. It also provides information on installation and operating requirements. You should read the **read.me** file on the CD-ROM before installing the MAX+PLUS II software. After installation, you can open the **read.me** file from the MAX+PLUS II Help menu.

Setting Up VSS Configuration Files

The **.synopsys_vss.setup** file contains the mapping information that directs the VHDL System Simulator (VSS) Software to use Altera® -supplied Altera Simulation Libraries during simulation. To configure your environment for the MAX+PLUS® II /Synopsys interface, follow these steps:

1.

2. Add the lines shown in Figure 1 to your **.synopsys_vss.setup** file. Altera provides a sample setup file, **.synopsys_vss.setup**, in the **/usr/maxplus2/synopsys/config** directory. See Figure 1.

Figure 1. Sample .synopsys_vss.setup File

```
WORK > DEFAULT
DEFAULT             : .



altera              : /usr/maxplus2/synopsys/library/alt_mf/lib



flex_vtl            : /usr/maxplus2/synopsys/library/alt_pre/vital/
                        lib/flex_vtl
alt_vtl             : /usr/maxplus2/synopsys/library/alt_post/sim/
                        lib/alt_vtl
flex10k_ftsm        : /usr/maxplus2/synopsys/library/alt_pre/flex10k/
                        lib/flex10k_ftsm
flex10k_ftgs        : /usr/maxplus2/synopsys/library/alt_pre/flex10k/
                        lib/flex10k_ftgs
max9000_ftsm        : /usr/maxplus2/synopsys/library/alt_pre/max9000/
                        lib/max9000_ftsm
max9000_ftgs        : /usr/maxplus2/synopsys/library/alt_pre/max9000/
                        lib/max9000_ftgs
flex8000_ftsm       : /usr/maxplus2/synopsys/library/alt_pre/flex8000/
                        lib/flex8000_ftsm
flex8000_ftgs       : /usr/maxplus2/synopsys/library/alt_pre/flex8000/
                        lib/flex8000_ftgs
max7000_ftsm        : /usr/maxplus2/synopsys/library/alt_pre/max7000/
                        lib/max7000_ftsm
max7000_ftgs        : /usr/maxplus2/synopsys/library/alt_pre/max7000/
                        lib/max7000_ftgs
flex6000_ftsm       : /usr/maxplus2/synopsys/library/alt_pre/flex6000/
                        lib/flex6000_ftsm
flex6000_ftgs       : /usr/maxplus2/synopsys/library/alt_pre/flex6000/
                        lib/flex6000_ftgs
max5000_ftsm        : /usr/maxplus2/synopsys/library/alt_pre/max5000/
                        lib/max5000_ftsm
max5000_ftgs        : /usr/maxplus2/synopsys/library/alt_pre/max5000/
```

```
                          lib/max5000_ftgs
flex10k_fpga_ftsm  : /usr/maxplus2/synopsys/library/alt_pre/flex10k/
                          lib/flex10k_fpga_ftsm
flex10k_fpga_ftsm  : /usr/maxplus2/synopsys/library/alt_pre/flex10k/
                          lib/flex10k_fpga_ftgs
max9000_fpga_ftsm  : /usr/maxplus2/synopsys/library/alt_pre/max9000/
                          lib/max9000_fpga_ftsm
max9000_fpga_ftgs  : /usr/maxplus2/synopsys/library/alt_pre/max9000/
                          lib/max9000_fpga_ftgs
flex8000_fpga_ftsm : /usr/maxplus2/synopsys/library/alt_pre/flex8000/
                          lib/flex8000_fpga_ftsm
flex8000_fpga_ftgs : /usr/maxplus2/synopsys/library/alt_pre/flex8000/
                          lib/flex8000_fpga_ftgs
max7000_fpga_ftsm  : /usr/maxplus2/synopsys/library/alt_pre/max7000/
                          lib/max7000_fpga_ftsm
max7000_fpga_ftgs  : /usr/maxplus2/synopsys/library/alt_pre/max7000/
                          lib/max7000_fpga_ftgs
flex6000_fpga_ftsm : /usr/maxplus2/synopsys/library/alt_pre/flex6000/
                          lib/flex6000_fpga_ftsm
flex6000_fpga_ftgs : /usr/maxplus2/synopsys/library/alt_pre/flex6000/
                          lib/flex6000_fpga_ftgs
max5000_fpga_ftsm  : /usr/maxplus2/synopsys/library/alt_pre/max5000/
                          lib/max5000_fpga_ftsm
max5000_fpga_ftgs  : /usr/maxplus2/synopsys/library/alt_pre/max5000/
                          lib/max5000_fpga_ftgs
```

The variables in the **.synopsys_vss** setup file perform the following functions:

-
- The WORK variable specifies your working directory, i.e., the directory where you start the Synopsys tools. If not explicitly specified elsewhere, the results of any analysis or compilation are written to this directory. The first line of the file shown in Figure 1 maps WORK to the design library variable called DEFAULT.
- The DEFAULT variable is used to create library aliases, which allows you to map the WORK variable to various paths. In Figure 1, the DEFAULT variable specifies the current directory.
- The **altera** library is listed to allow you to simulate the architecture control logic functions in the **alt_mf** library.
- The remaining lines in the file specify the path and name of the directories that contain the device simulation libraries for Altera device families.

3. Analyze the target device simulation library to ensure that the correct timing and functional information is provided to VSS. Analyzing the simulation library produces VSS simulation models of the primitives that appear in all Altera-provided technology libraries.

   You can analyze device simulation libraries by using the Altera-provided shell script **analyze_vss**:

   1.
   2. Add the **/usr/maxplus2/synopsys/bin** directory, which contains the **analyze_vss** scripts, to the PATH environment variable in your **.cshrc** file.
   3. Make sure that you have write privileges for the **/usr/maxplus2/synopsys/library/alt_pre/**<*device family*> directory because the analyzed model is placed in the **/usr/maxplus2/synopsys/library/alt_pre/**<*device family*>**/lib** directory and the analysis log file is placed in the **./synopsys/library/alt_pre/**<*device family*>**/src** directory.
   4. Run the **analyze_vss** shell script by typing `analyze_vss` ↵ at the **dc_shell** prompt. When you run the **analyze_vss** shell script, you are prompted to select the appropriate device family simulation model(s) for analysis. Figure 2 shows the **analyze_vss** shell script.

Figure 2. The analyze_vss Shell Script

Type the full pathname of the directory where the MAX+PLUS® II software is installed (default: **/usr/maxplus2**):

<MAX+PLUS II system directory> ↵

```
Analyze VSS Simulation Models:
1. flex10k_FTGS
2. flex10k_FTSM
3. flex10k_fpga_FTGS
4. flex10k_fpga_FTSM
5. max9000_FTGS
6. max9000_FTSM
7. max9000_fpga_FTGS
8. max9000_fpga_FTSM
9. flex8000_FTGS
10. flex8000_FTSM
11. flex8000_fpga_FTGS
12. flex8000_fpga_FTSM
13. max7000_FTGS
14. max7000_FTSM
15. max7000_fpga_FTGS
16. max7000_fpga_FTSM
17. flex6000_FTGS
18. flex6000_FTSM
19. flex6000_fpga_FTGS
20. flex6000_fpga_FTSM
21. max5000_FTGS
22. max5000_FTSM
23. max5000_fpga_FTGS
24. max5000_fpga_FTSM
25. alt_vtl
26. flex_vtl
27. Quit
```

Enter one or more numbers: *<device library numbers>* ↵

5. Check the log file to make sure that no errors occurred during the analysis of the simulation models.

4. Use VSS to simulate your pre-routed VHDL design.

Related Topics:

- Refer to the ***VHDL System Simulator Core Programs Manual*** for more information about VSS.

Altera Simulation Libraries

Altera provides simulation libraries for both pre-routing functional simulation and post-routing timing simulation.

Pre-Routing Functional Simulation Libraries (VITAL-Compliant)

The **/usr/maxplus2/synopsys/library/alt_pre/vital/src** directory contains Altera® -provided VHDL simulation models in VITAL 95 format. This library contains functional descriptions of all primitives that appear in Altera-specific technology libraries. These libraries allow you to perform a functional or pre-routing simulation that verifies the netlist structure generated by the Synopsys Design Compiler or FPGA Compiler software. Altera provides the **flex.cmp** and **flex.vhd** files in the **/usr/maxplus2/synopsys/library/alt_pre/vital/src** directory.

Similarly, the **/usr/maxplus2/synopsys/library/alt_pre/verilog/src** directory contains Altera-provided Verilog HDL simulation models for all device families. The **altera.v** file can be used for simulation with the Cadence Verilog-XL simulator.

Pre-Routing Functional Simulation Libraries with Estimated Timing Information

The **/usr/maxplus2/synopsys/library/alt_pre/**<*device family*>**/src** directory contains Altera® -provided VHDL simulation libraries, which give both functional and area descriptions of all primitives that appear in all Altera technology libraries. These simulation libraries allow you to verify the function of VHDL projects, with estimated timing, after synthesizing them with the Synopsys Design Compiler or FPGA Compiler, but before submitting them to MAX+PLUS® II software for compilation.

Altera provides an encrypted Full Timing Structural Model (FTSM) and a Full Timing Gate-Level Simulation model (FTGS) for the VHDL simulation libraries listed in Table 1.

Table 1. VHDL Functional Simulation Libraries

Device Family	Functional Simulation Libraries	Device Family	Functional Simulation Libraries
FLEX® 10K	**flex10k_FTSM.vhd.E** **flex10k_fpga_FTSM.vhd.E** **flex10k_FTGS.vhd.E** **flex10k_fpga_FTGS.vhd.E** **flex10k_components.vhd** **flex10k_fpga_components.vhd**	MAX® 9000	**max9000_FTSM.vhd.E** **max9000_fpga_FTSM.vhd.E** **max9000_FTGS.vhd.E** **max9000_fpga_FTGS.vhd.E** **max9000_components.vhd** **max9000_fpga_components.vhd**
FLEX 8000	**flex8000_FTSM.vhd.E** **flex8000_fpga_FTSM.vhd.E** **flex8000_FTGS.vhd.E** **flex8000_fpga_FTGS.vhd.E** **flex8000_components.vhd** **flex8000_fpga_components.vhd**	MAX 7000	**max7000_FTSM.vhd.E** **max7000_fpga_FTSM.vhd.E** **max7000_FTGS.vhd.E** **max7000_fpga_FTGS.vhd.E** **max7000_components.vhd** **max7000_fpga_components.vhd**
FLEX 6000	**flex6000_FTSM.vhd.E** **flex6000_fpga_FTSM.vhd.E** **flex6000_FTGS.vhd.E** **flex6000_fpga_FTGS.vhd.E** **flex6000_components.vhd** **flex6000_fpga_components.vhd**	MAX 5000 & Classic®	**max5000_FTSM.vhd.E** **max5000_fpga_FTSM.vhd.E** **max5000_FTGS.vhd.E** **max5000_fpga_FTGS.vhd.E** **max5000_components.vhd** **max5000_fpga_components.vhd**

Post-Routing Timing Simulation Libraries

The **/usr/maxplus2/synopsys/library/alt_post/sim/src** directory contains the Altera® -provided library files for performing timing simulation of designs that have been compiled with the MAX+PLUS II software. The VITAL 95-compliant post-simulation source files included in this directory are **alt_vtl.vhd** and **alt_vtl.cmp**. See Performing a Timing Simulation with VSS Software for more information.

Related Topics:

- Go to the following topics for additional information:
  - FLEX Devices
  - MAX Devices
  - Classic Device Family

MAX+PLUS II/Synopsys Interface File Organization

Table 1 shows the MAX+PLUS® II /Synopsys interface subdirectories that are created in the MAX+PLUS II system directory (by default, the /usr/maxplus2 directory) during the MAX+PLUS II software installation. For information on the other directories that are created during the MAX+PLUS II software installation, see "MAX+PLUS II File Organization" in MAX+PLUS II Installation in the MAX+PLUS II Getting Started manual.

☞ You must add the **/usr/maxplus2/bin** directory to the `PATH` environment variable in your **.cshrc** file in order to run the MAX+PLUS II software.

Table 1. MAX+PLUS II Directory Organization

Directory	Description
./synopsys/bin	Contains script programs to convert Synopsys timing constraints into MAX+PLUS II Assignment & Configuration File (**.acf**) format, and to analyze VHDL System Simulator simulation models.
./synopsys/config	Contains sample **.synopsys_dc.setup** and **.synopsys_vss.setup** files.
./synopsys/examples	Contains sample files, including those discussed in these ACCESS Key Guidelines.
./synopsys/library/alt_pre/<*device family*>**/src**	Contains VHDL simulation libraries for functional simulation of VHDL projects.
./synopsys/library/alt_pre/verilog/src	Contains the Verilog HDL functional simulation library for Verilog HDL projects.
./synopsys/library/alt_pre/vital/src	Contains the VITAL 95 simulation library. You use this library when you perform functional simulation of the design before compiling it with the MAX+PLUS II software.
./synopsys/library/alt_syn//<*device family*>**/lib**	Contains interface files for the MAX+PLUS II/Synopsys interface. Technology libraries in this directory allow the Design Compiler and FPGA Compiler to map designs to Altera® device architectures.
./synopsys/library/alt_mf/src	Contains behavioral VHDL models of some Altera macrofunctions, along with their component declarations. The `a_81mux`, `a_8count`, `a_8fadd`, and `a_8mcomp` macrofunctions are currently supported. Libraries in this directory allow you to instantiate, synthesize, and simulate these macrofunctions.
./synopsys/library/alt_post/syn/lib	Contains the post-synthesis library for technology mapping.
./synopsys/library/alt_post/sim/src	Contains the VHDL source files for the VITAL 95-compliant library. You use this library when you perform simulation of the design after compiling it with the MAX+PLUS II software.

Related Topics:

- Go to the following topics for additional information:
  - ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
  - This manual is also available in 4 parts:
    - Preface & Section 1: MAX+PLUS II Installation
    - Section 2: MAX+PLUS II - A Perspective
    - Section 3: MAX+PLUS II Tutorial
    - Appendices, Glossary & Index

MAX+PLUS II Project File Structure

In MAX+PLUS® II, a project name is the name of a top-level design file, without the filename extension. This design file can be an EDIF, Verilog HDL, or VHDL netlist file; an AHDL™ TDF; or any other MAX+PLUS II-supported design file. The EDIF netlist file must be created by Synopsys and imported into MAX+PLUS II as an EDIF Input File.

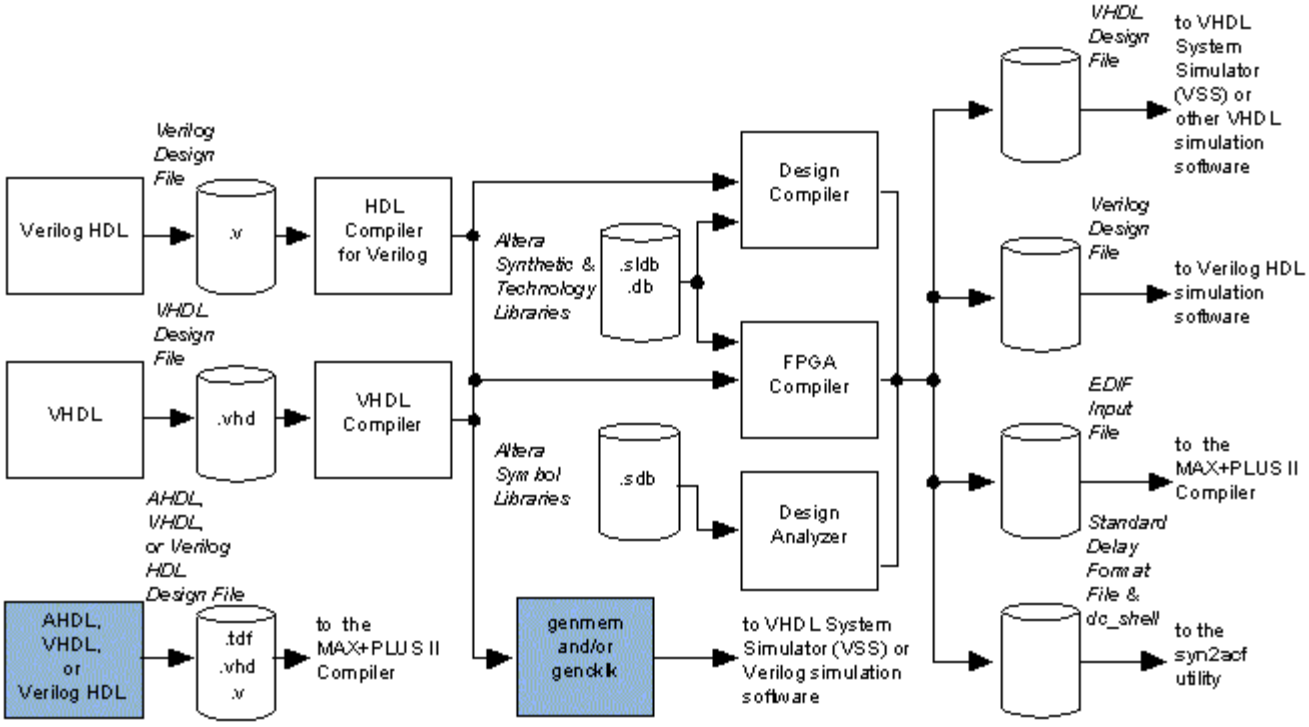
MAX+PLUS II stores the connectivity data on the links between design files in a hierarchical project in a Hierarchy Interconnect File (**.hif**), but refers to the entire project only by its project name. The MAX+PLUS II Compiler uses the HIF to build a single, fully flattened project database that integrates all the design files in a project hierarchy.

Synopsys Design Entry Flow

Figure 1 below shows the design entry flow for the MAX+PLUS® II /Synopsys interface.

Figure 1. MAX+PLUS II/Synopsys Design Entry Flow

Altera-provided items are shown in blue.



Performing a Pre-Routing or Functional Simulation with VSS Software

After you have synthesized and optimized a VHDL or Verilog HDL design with the Design Compiler or FPGA Compiler software, you can perform a pre-routing or functional simulation with the Synopsys VHDL System Simulator (VSS) software.

To perform a pre-routing/functional simulation, follow these steps:

1.
2. Be sure to set up the working environment correctly, as described in the following topics:
    - Setting Up the MAX+PLUS II/Synopsys Working Environment
    - Setting Up Design Compiler & FPGA Compiler Configuration Files
    - Setting Up the DesignWare Interface
    - Setting Up VSS Configuration Files
3. Create a VHDL or Verilog HDL design file that follows the guidelines described in one of the following topics:
    - Creating VHDL Designs for Use with MAX+PLUS II Software
    - Creating Verilog HDL Designs for Use with MAX+PLUS II Software
4. Synthesize and optimize your design with the Design Compiler or FPGA Compiler, as described in Synthesizing & Optimizing VHDL & Verilog HDL Files with Design Compiler or FPGA Compiler Software.
5. Save your design as a VHDL Design File (**.vhd**).

    ☞ VSS requires each architecture/entity pair in a VHDL Design File to have a configuration. The Configuration Declaration is necessary for simulation, but not for synthesis.
6. Use VSS and one of the Altera pre-routing functional simulation libraries to simulate the design.
7. When you are ready to compile your project with MAX+PLUS II software, save the design as an EDIF netlist file (**.edf**), then process it as described in Compiling Projects with MAX+PLUS II Software.

Related Topics:

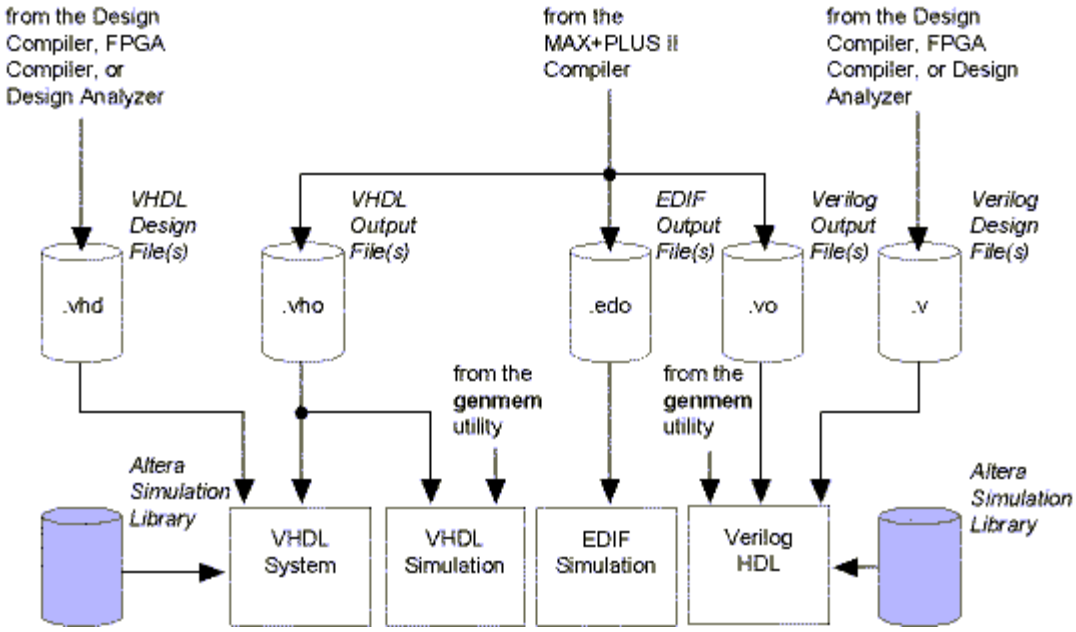
- Refer to the following sources for related information:
    - ***VHDL System Simulator Core Programs Manual*** for more information about VSS
    - Performing a Timing Simulation with VSS Software

Project Simulation Flow

Figure 1 shows the project simulation flow for the MAX+PLUS® II /Synopsys interface.

Figure 1. MAX+PLUS II/Synopsys Project Simulation Flow

Altera-provided items are shown in blue.



The MAX+PLUS II/Synopsys design environment fully supports design verification with the Synopsys VHDL System Simulator (VSS). For pre-route simulation, you can simulate a design that has been compiled with one of the Synopsys compilers. For post-route simulation, you can simulate the VHDL Output File (**.vho**) that MAX+PLUS II® software generates during project compilation.

Performing a Timing Simulation with VSS Software

Once the MAX+PLUS® II software has compiled a project and generated a VHDL Output File (**.vho**) and an optional Standard Delay Format (SDF) Output File (**.sdo**), you can perform timing simulation with the Synopsys VHDL Simulator Software (VSS).

To simulate a VHDL Output File with VSS, follow these steps:

Be sure to set up the working environment correctly, as described in the following topics:

- Setting Up the MAX+PLUS II/Synopsys Working Environment
- Setting Up Design Compiler & FPGA Compiler Configuration Files
- Setting Up the DesignWare Interface
- Setting Up VSS Configuration Files

1. Generate a VHDL Output File (**.vho**) and an optional SDF Output File (**.sdo**), as described in Compiling Projects with MAX+PLUS II Software.

2. (Optional) Analyze the VITAL 95-compliant **alt_vtl** library , then back-annotate timing information through the SDF Output File:

    a.

    b. Use the **analyze_vss** script to analyze the **alt_vtl** Post-Routing Timing Simulation library, as described in Setting Up VSS Configuration Files.

    c. Enter the following command to back-annotate timing information through the SDF Output File:

    `vhdlsim -sdf_top` /<*design name*>/<*design name*> `-sdf`

<*design name*>.sdo ↵

3. Simulate the VHDL Output File with the VSS software.

Related Topics:

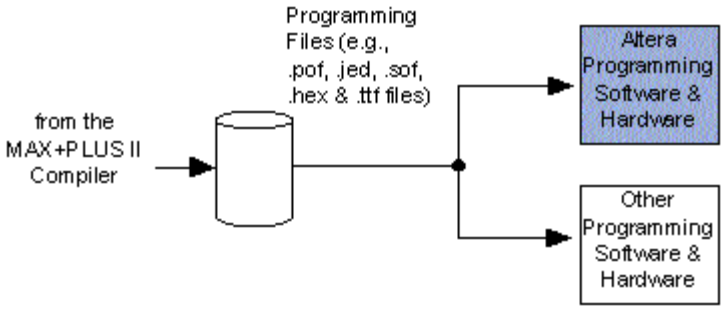
- Go to the ***VSS User's Guide*** for more details on post-routing simulation.

Programming Altera Devices

Once you have successfully compiled and simulated a project with the MAX+PLUS® II software, you can program an Altera® device and test it in the target circuit. Figure 1 shows the device programming flow for MAX+PLUS II software.

Figure 1. MAX+PLUS II Device Programming Flow

Altera-provided items are shown in blue.



You can program devices with Altera programming hardware and MAX+PLUS II Programmer software installed on a 486- or Pentium-based PC or a UNIX workstation, or with programming hardware and software available from other manufacturers. Table 1 shows the available Altera programming hardware options on PCs and UNIX workstations.

Table 1. Altera Programming Hardware

Programming Hardware Option	UNIX PCs	Work-stations	MAX® 3000A Devices	Classic® & MAX 5000 Devices	MAX 7000 & MAX 7000E Devices	MAX 7000A, MAX 7000AE, MAX 7000S MAX 9000 & MAX 9000A Devices	FLEX® 6000, FLEX 6000A, FLEX 8000, FLEX 10K, FLEX 10KA, FLEX 10KB, & FLEX 10KE Devices	In-System Programming/ Configuration
Logic Programmer card, PL-MPU Master Programming Unit, and device-specific	✓		✓	✓	✓	✓		

adapters						
BitBlaster™ Download Cable	✓	✓	✓		✓	✓	✓
ByteBlasterMV™ Download Cable	✓		✓		✓	✓	✓

If you wish to transfer programming files from a UNIX workstation to a PC over a network with File Transfer Protocol (FTP) or other similar transfer programs, be sure to select binary transfer mode.

Programming hardware from other manufacturers varies, but typically consists of a device connected to one of the serial ports on the workstation. Various vendors, such as Data I/O and BP Microsystems, supply hardware and software for programming Altera devices.

Last updated on December 6, 1999 for the MAX+PLUS II software version 9.4.

Related Links

- MAX+PLUS II Development Software
- Altera Programming Hardware

Setting Up VSS Configuration Files

The **.synopsys_vss.setup** file contains the mapping information that directs the VHDL System Simulator (VSS) Software to use Altera® -supplied Altera Simulation Libraries during simulation. To configure your environment for the MAX+PLUS® II /Synopsys interface, follow these steps:

1. Add the lines shown in Figure 1 to your **.synopsys_vss.setup** file. Altera provides a sample setup file, **.synopsys_vss.setup**, in the **/usr/maxplus2/synopsys/config** directory. See Figure 1.

Figure 1. Sample .synopsys_vss.setup File

```
WORK > DEFAULT
DEFAULT             : .



altera              : /usr/maxplus2/synopsys/library/alt_mf/lib



flex_vtl            : /usr/maxplus2/synopsys/library/alt_pre/vital/
                         lib/flex_vtl
alt_vtl             : /usr/maxplus2/synopsys/library/alt_post/sim/
                         lib/alt_vtl
flex10k_ftsm        : /usr/maxplus2/synopsys/library/alt_pre/flex10k/
                         lib/flex10k_ftsm
flex10k_ftgs        : /usr/maxplus2/synopsys/library/alt_pre/flex10k/
                         lib/flex10k_ftgs
max9000_ftsm        : /usr/maxplus2/synopsys/library/alt_pre/max9000/
                         lib/max9000_ftsm
max9000_ftgs        : /usr/maxplus2/synopsys/library/alt_pre/max9000/
                         lib/max9000_ftgs
flex8000_ftsm       : /usr/maxplus2/synopsys/library/alt_pre/flex8000/
                           lib/flex8000_ftsm
flex8000_ftgs       : /usr/maxplus2/synopsys/library/alt_pre/flex8000/
                           lib/flex8000_ftgs
max7000_ftsm        : /usr/maxplus2/synopsys/library/alt_pre/max7000/
                         lib/max7000_ftsm
max7000_ftgs        : /usr/maxplus2/synopsys/library/alt_pre/max7000/
                         lib/max7000_ftgs
flex6000_ftsm       : /usr/maxplus2/synopsys/library/alt_pre/flex6000/
                           lib/flex6000_ftsm
flex6000_ftgs       : /usr/maxplus2/synopsys/library/alt_pre/flex6000/
                           lib/flex6000_ftgs
max5000_ftsm        : /usr/maxplus2/synopsys/library/alt_pre/max5000/
                         lib/max5000_ftsm
max5000_ftgs        : /usr/maxplus2/synopsys/library/alt_pre/max5000/
                         lib/max5000_ftgs
flex10k_fpga_ftsm   : /usr/maxplus2/synopsys/library/alt_pre/flex10k/
                         lib/flex10k_fpga_ftsm
flex10k_fpga_ftsm   : /usr/maxplus2/synopsys/library/alt_pre/flex10k/
                         lib/flex10k_fpga_ftgs
max9000_fpga_ftsm   : /usr/maxplus2/synopsys/library/alt_pre/max9000/
                         lib/max9000_fpga_ftsm
max9000_fpga_ftgs   : /usr/maxplus2/synopsys/library/alt_pre/max9000/
                         lib/max9000_fpga_ftgs
flex8000_fpga_ftsm  : /usr/maxplus2/synopsys/library/alt_pre/flex8000/
                         lib/flex8000_fpga_ftsm
flex8000_fpga_ftgs  : /usr/maxplus2/synopsys/library/alt_pre/flex8000/
                         lib/flex8000_fpga_ftgs
max7000_fpga_ftsm   : /usr/maxplus2/synopsys/library/alt_pre/max7000/
                         lib/max7000_fpga_ftsm
max7000_fpga_ftgs   : /usr/maxplus2/synopsys/library/alt_pre/max7000/
                         lib/max7000_fpga_ftgs
flex6000_fpga_ftsm  : /usr/maxplus2/synopsys/library/alt_pre/flex6000/
                         lib/flex6000_fpga_ftsm
flex6000_fpga_ftgs  : /usr/maxplus2/synopsys/library/alt_pre/flex6000/
                         lib/flex6000_fpga_ftgs
```

```
max5000_fpga_ftsm  : /usr/maxplus2/synopsys/library/alt_pre/max5000/
                        lib/max5000_fpga_ftsm
max5000_fpga_ftgs  : /usr/maxplus2/synopsys/library/alt_pre/max5000/
                        lib/max5000_fpga_ftgs
```

The variables in the **.synopsys_vss** setup file perform the following functions:

- The WORK variable specifies your working directory, i.e., the directory where you start the Synopsys tools. If not explicitly specified elsewhere, the results of any analysis or compilation are written to this directory. The first line of the file shown in Figure 1 maps WORK to the design library variable called DEFAULT.
- The DEFAULT variable is used to create library aliases, which allows you to map the WORK variable to various paths. In Figure 1, the DEFAULT variable specifies the current directory.
- The **altera** library is listed to allow you to simulate the architecture control logic functions in the **alt_mf** library.
- The remaining lines in the file specify the path and name of the directories that contain the device simulation libraries for Altera device families.

2. Analyze the target device simulation library to ensure that the correct timing and functional information is provided to VSS. Analyzing the simulation library produces VSS simulation models of the primitives that appear in all Altera-provided technology libraries.

   You can analyze device simulation libraries by using the Altera-provided shell script **analyze_vss**:

   1. Add the **/usr/maxplus2/synopsys/bin** directory, which contains the **analyze_vss** scripts, to the PATH environment variable in your **.cshrc** file.
   2. Make sure that you have write privileges for the **/usr/maxplus2/synopsys/library/alt_pre/**<*device family*> directory because the analyzed model is placed in the **/usr/maxplus2/synopsys/library/alt_pre/**<*device family*>**/lib** directory and the analysis log file is placed in the **./synopsys/library/alt_pre/**<*device family*>**/src** directory.
   3. Run the **analyze_vss** shell script by typing `analyze_vss` ↵ at the **dc_shell** prompt. When you run the **analyze_vss** shell script, you are prompted to select the appropriate device family simulation model(s) for analysis. Figure 2 shows the **analyze_vss** shell script.

      Figure 2. The analyze_vss Shell Script

      Type the full pathname of the directory where the MAX+PLUS® II software is installed (default: **/usr/maxplus2**):

      <*MAX+PLUS II system directory*> ↵

```
Analyze VSS Simulation Models:
1. flex10k_FTGS
2. flex10k_FTSM
3. flex10k_fpga_FTGS
4. flex10k_fpga_FTSM
5. max9000_FTGS
6. max9000_FTSM
7. max9000_fpga_FTGS
8. max9000_fpga_FTSM
9. flex8000_FTGS
10. flex8000_FTSM
11. flex8000_fpga_FTGS
12. flex8000_fpga_FTSM
13. max7000_FTGS
14. max7000_FTSM
```

```
15. max7000_fpga_FTGS
16. max7000_fpga_FTSM
17. flex6000_FTGS
18. flex6000_FTSM
19. flex6000_fpga_FTGS
20. flex6000_fpga_FTSM
21. max5000_FTGS
22. max5000_FTSM
23. max5000_fpga_FTGS
24. max5000_fpga_FTSM
25. alt_vtl
26. flex_vtl
27. Quit
```

Enter one or more numbers: *<device library numbers>* ↵

4. Check the log file to make sure that no errors occurred during the analysis of the simulation models.

3. Use VSS to simulate your pre-routed VHDL design.

Related Topics:

- Refer to the ***VHDL System Simulator Core Programs Manual*** for more information about VSS.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Using Synopsys VSS & MAX+PLUS II Software

The following topics describe how to use the Synopsys VHDL System Simulator (VSS) and MAX+PLUS® II software. Choose one of the following topics for information:

Open a printable version of all topics listed on this page.

Setting Up the MAX+PLUS II/Synopsys Working Environment

- Software Requirements
- Setting Up VSS Configuration Files
- Simulation Libraries
- MAX+PLUS II/Synopsys Interface File Organization
- MAX+PLUS II Project File Structure

Functional Simulation

- Design Entry Flow
- Performing a Pre-Routing or Functional Simulation with VSS Software

Timing Simulation

- Project Simulation Flow
- Initializing Registers in VHDL & Verilog Output Files for Power-Up before Simulation
- Performing a Timing Simulation with VSS Software

Related Links:

- Go to the following topics in these MAX+PLUS II ACCESS℠ Key topics for related information:
  - Compiling Projects with MAX+PLUS II Software
  - Programming Altera Devices
  - Resynthesizing a Design Using the **alt.vtl** Library & a MAX+PLUS II SDF Output File
  - Using Synopsys Design Compiler or FPGA Compiler & MAX+PLUS II Software
  - Using Synopsys FPGA Express & MAX+PLUS II Software
  - Using Synopsys PrimeTime & MAX+PLUS II Software
- Go to the following topics, which are available on the web, for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware
  - Synopsys web site (http://www.synopsys.com)

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Synthesizing & Optimizing VHDL & Verilog HDL Projects with Synopsys Software

The MAX+PLUS® II Compiler can process a VHDL or Verilog HDL file that has been synthesized by the Synopsys Design Compiler or FPGA Compiler software, saved as an EDIF 2 0 0 or 3 0 0 netlist file, and imported into the MAX+PLUS II software. The procedure below explains how to run Synopsys tools separately from MAX+PLUS II Software.

☞ You can also run Synopsys tools from within the MAX+PLUS II software to automatically generate and import an EDIF file. Refer to Running Synopsys Compilers from MAX+PLUS II Software for more information. In addition, if your MAX+PLUS II development system includes VHDL or Verilog HDL synthesis support, the MAX+PLUS II Compiler can directly synthesize VHDL or Verilog HDL logic. For more information, go to MAX+PLUS II VHDL or Verilog HDL Help.

The following steps explain how to synthesize and optimize a VHDL or Verilog HDL design for use with MAX+PLUS II software:

1. Be sure to set up your design environment correctly. This step includes specifying the target device family for the design. See the following topics:
   - Setting Up the Synopsys/MAX+PLUS II Working Environment
   - Setting Up the Design Compiler and FPGA Compiler Configuration Files
   - Setting Up the DesignWare Interface
   - Setting Up the VSS Configuration Files
2. Create a VHDL file, <*design name*>**.vhd**, or a Verilog HDL design, <*design name*>**.v**, using the MAX+PLUS II Text Editor or another standard text editor and save it in a project directory under your login directory. See the following topics for instructions:
   - Creating VHDL Designs for Use with MAX+PLUS II Software.
   - Creating Verilog HDL Designs for Use with MAX+PLUS II Software.
3. Start the Design Compiler or FPGA Compiler software by typing either `dc_shell` ↵ or `fpga_shell` ↵ at the command line, respectively. To work within the graphical user interface, type `design_analyzer` ↵ for either tool.
4. Analyze and then compile the design with the Design Compiler, FPGA Compiler, or Design Analyzer software. The VHDL Compiler or HDL Compiler for Verilog software automatically translates the design into Synopsys database (**.db**) format. Specific steps are necessary for some types of projects before you process the design:
   1. If your FLEX 10K design includes RAM or ROM functions, follow these steps:
      1. (VHDL designs only) Because the VHDL Compiler software does not support the data type `string` for the Generic Clause, you must also enter the following command at the `dc_shell` prompt before you read the design:

         ```
         hdlin_translate_off_skip_text=true ↵
         ```

      2. The timing model (**.lib**) generated by the **genmem** utility contains pin-to-pin delay information that can be used by the Synopsys Design Compiler and FPGA Compiler software. You must add

this timing model to the existing library so that the compiler can access the timing information. Type the following commands at the **dc_shell** prompt:

```
read -f db flex10k[<speed grade>].db ↵
update_lib flex10k[<speed grade>] <RAM/ROM function name>.lib ↵
```

3. (Optional) Enter the following command to update your **flex10k[***<speed grade>***].db** file with the RAM/ROM timing information:

```
write_lib flex10k[<speed grade>] -o flex10k.db ↵
```

See Instantiating RAM & ROM Functions in VHDL or Instantiating RAM & ROM functions in Verilog HDL for additional information.

2. If you wish to allow the FPGA Compiler to perform *N*-input look-up table (LUT) optimization for a FLEX 6000, FLEX 8000, or FLEX 10K design, enter the following command at the `dc_shell` prompt before compiling the design:

```
edifout_write_properties_list = "lut function" ↵
```

Go to Using FPGA Compiler *N*-Input LUT Optimization for FLEX 6000, FLEX 8000, or FLEX 10K Devices for more information.

3. If you wish to enter resource assignments, go to Entering Resource Assignments.

4. If you wish to direct the Design Compiler or FPGA Compiler to use sum-of-products logic in processing a MAX 7000 or MAX 9000 design, type the following commands at the `dc_shell` prompt before compiling the design:

```
set_structure false ↵
set_flatten -effort low ↵
```

See MAX 7000 & MAX 9000 Synthesis Example for more information.

For additional information on how the Design Compiler and FPGA Compiler synthesize and optimize a design, see the following topics:

- ***Synopsys Design Compiler Reference Manual*** or ***Design Analyzer Reference Manual***
- DesignWare FLEX 8000 Synthesis Example

5. (Optional) View the optimized project with the Design Analyzer. The Design Analyzer uses the **altera.sdb** library to display optimized projects generated by the Design Compiler or FPGA Compiler.

6. (Optional) To view Synopsys-generated timing information and generate a file detailing primitive usage, type the following commands:

```
report_timing ↵
report_reference > <filename> ↵
```

7. (Optional) To functionally verify the project prior to processing with the MAX+PLUS II software, save the design as a VHDL netlist file, and simulate it as described in Performing a Pre-Routing or Functional Simulation with VSS Software.

8. Save the optimized project as an EDIF netlist file with the extension **.edf**.

9. Process the *<design name>***.edf** file with the MAX+PLUS II Compiler, as described in Compiling Projects with the MAX+PLUS II Software.

Installing the Altera-provided MAX+PLUS II/Synopsys interface on your computer automatically creates the following sample VHDL and Verilog HDL files:

- **/usr/maxplus2/synopsys/examples/ministate.vhd**
- **/usr/maxplus2/synopsys/examples/ministate.v**

Related Links:

- Go to the following MAX+PLUS II ACCESS[SM] Key topics for related information:
  - Resynthesizing a Design Using the **alt_vtl** Library and a MAX+PLUS II SDF Output File
  - Programming Altera Devices

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Viewlogic Powerview viewdraw.ini Configuration File

Each Powerview project is configured with the **viewdraw.ini** file that resides in the project directory. The DIR statements at the end of **viewdraw.ini** are paths to library directories that are used by the various Powerview applications. Figure 1 shows a sample of the DIR statements that are required to use the libraries.

Figure 1. Excerpt from viewdraw.ini

```
DIR [pw] .
DIR [r]  /usr/maxplus2/vwlogic/library/alt_max2 (alt_max2)
DIR [r]  /usr/maxplus2/vwlogic/library/max2sim (max2_sim)
DIR [r]  /usr/maxplus2/vwlogic/library/synlib (altera)
DIR [r]  /usr/maxplus2/vwlogic/library/alt_mf (alt_mf)
DIR [r]  /usr/maxplus2/vwlogic/library/alt_vtl (alt_vtl)
DIR [rm] /<Powerview system directory>/lib/builtin (builtin)
DIR [rm] /<Powerview system directory>/simmods/vl/dip/74ls (vl74ls)
DIR [rm] /<Powerview system directory>/symsets/vl/dip/74ls (vl74ls)
DIR [r]  /<Powerview system directory>/lib/vdpath (vdpath)
```

☞ When you add the libraries to the **/usr/maxplus2/vwlogic/standard/viewdraw.ini** file, they are automatically set when you create a new project. Powerview tools search these libraries sequentially, so it is important to add them in the order in which they are listed in Figure 1.

Table 1 shows the libraries that must be specified in the DIR statements in the **viewdraw.ini** file.

Table 1. Powerview Application Libraries

Library	Library Alias	Source	Topics
alt_max2	alt_max2	Altera	Graphical elements for ViewDraw
max2sim	max2_sim	Altera	Models for project simulation
synlib	altera	Altera	VHDL synthesis library for the MAX+PLUS Â® Â II software
alt_mf	alt_mf	Altera	VHDL models of MAX+PLUSÂ II logic functions
alt_vtl	alt_vtl	Altera	VITAL-compliant primitives
builtin	builtin	Altera	Basic primitives such as INPUT pins, OUTPUT pins, AND gates, OR gates, etc.
74ls	vl74ls	Viewlogic	74-series macrofunctions
vdpath	vdpath	Viewlogic	Standard library of parameterized modules (LPM) functions

☞ The Altera-provided libraries must be listed before the Viewlogic-provided libraries in the **viewdraw.ini** file to ensure that the correct versions of the megafunctions, macrofunctions, and primitives are used.

Related Topics:

- Go to Altera-Provided Logic & Symbol Libraries for more information on Altera-supplied libraries. Refer to the Powerview documentation for more information on setting up the **viewdraw.ini** file.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

ACCESS Partner EDA Tools, Listed by Vendor

Click on one of the following topics for information:

ACCESS[SM] Partner	Guidelines for Using a Complete Multi-Tool Design Flow	Guidelines for Using an Individual Tool
	Using Cadence Tools with MAX+PLUS® II Software	• Concept • Composer • Leapfrog • RapidSIM • Synergy • Verilog-XL
	Using Mentor Graphics & Exemplar Logic Tools with MAX+PLUS II Software	• Galileo Extreme • Leonardo
	Using Mentor Graphics & Exemplar Logic Tools with MAX+PLUS II Software	• Design Architect • QuickHDL & QuickHDL Pro • QuickPath • QuickSim II (includes Design Viewpoint Editor)
	(not applicable)	• Design Compiler • FPGA Compiler • FPGA Express • PrimeTime • VHDL System Simulator (VSS)
	(not applicable)	• Certify • Synplify • Synplify Pro

Legal Notice: Copyright, Trademark, Patent & Warranty Information

Please read this page before accessing the documentation on this CD-ROM.

This CD-ROM contains documentation and other information related to products and services of Altera Corporation ("Altera"). This documentation is provided as a courtesy to Altera's customers and potential customers. By accessing, copying, or using any information contained on this CD-ROM, you agree to be bound by the terms and conditions described in this Legal Notice.

Copyright Notice

The documentation, software, and other materials contained on this CD-ROM are owned and copyrighted by Altera. Copyright © 1998 Altera Corporation, 101 Innovation Dr., San Jose, California 95134, USA, all rights reserved.

License to Copy Information

You are licensed to copy documentation and other materials from this CD-ROM provided you agree to the following terms and conditions.

- You may use the materials for informational, non-commercial purposes only.
- You may not alter or modify the materials in any way.
- You may not use any graphics separate from any accompanying text.
- You may distribute copies of the documentation on this CD-ROM only to customers and potential customers of Altera products. However, you may not charge them for such use. Any other distribution to third parties is prohibited unless you obtain the prior written consent of Altera.
- You may not use the materials in any way that may be adverse to Altera's interests.

All copies of materials that you copy from this CD-ROM must include a copy of this Legal Notice.

Failure to comply with these terms and conditions will terminate the license.

Other Intellectual Property Rights

Altera, MAX, MAX+PLUS, FLEX, and other names of Altera products, product features, and services are trademarks and/or service marks of Altera Corporation in the United States and other countries. Other product and company names mentioned on this CD-ROM may be the trademarks of their respective owners.

Nothing contained in this Legal Notice shall be construed as conferring by implication, estoppel, or any other legal theory, a license or right to any patent, trademark, copyright, or other intellectual property right, except those expressly provided herein. The products, processes, software, and other technology described on this CD-ROM may be the subject of other intellectual property rights owned by Altera or by third parties, and no licenses are granted herein.

Disclaimers

NO WARRANTIES: THE DOCUMENTATION PROVIDED ON THIS CD-ROM IS "AS IS" WITHOUT ANY

EXPRESS OR IMPLIED WARRANTY OF ANY KIND INCLUDING WARRANTIES OF MERCHANTABILITY, NONINFRINGEMENT OF INTELLECTUAL PROPERTY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT SHALL ALTERA OR ITS SUPPLIERS BE LIABLE FOR ANY DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OF OR INABILITY TO USE THE DOCUMENTATION PROVIDED ON THIS CD-ROM, EVEN IF ALTERA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. BECAUSE SOME JURISDICTIONS PROHIBIT THE EXCLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES, SOME OF THE ABOVE LIMITATIONS MAY NOT APPLY TO YOU. Altera further does not warrant the accuracy or completeness of the information, text, graphics, or other items contained within this CD-ROM. Altera may make changes to these materials, or to the products described therein, at any time without notice.

U.S. GOVERNMENT RESTRICTED RIGHTS: The materials and documentation are provided with "RESTRICTED RIGHTS". Use, duplication, or disclosure by the Government is subject to restrictions as set forth in FAR52.227-14 and DFAR252.227-7013 et seq. or its successor. Use of the documentation and materials by the Government constitutes acknowledgment of Altera's proprietary rights in them.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

ACCESS Key Guidelines Feedback

Thank you. Your feedback has been forwarded

Feedback

Instantiating LPM Functions in Design Architect Schematics

Design Architect software allows you to instantiate functions included in the library of parameterized modules (LPM) from the **ALTERA LPMLIB** library.

Go through the following steps to instantiate LPM functions in a Design Architect schematic:

1. While you are entering your Design Architect schematic, choose **Altera Libraries** (Library menu).
2. Choose **ALTERA LPMLIB** (Altera Libraries menu).
3. Choose from the available LPM functions on the ALTERA GENLIB menu.
4. In the **LPM_**<*function name*> dialog box, specify appropriate values for the variables displayed for the LPM function you chose in step 3. Make sure that any hexadecimal (Intel-format) file that you use to specify the initial content of a memory function does not have the same name as the design file name. Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu for detailed information on LPM functions.
5. Choose **OK** to generate a symbol for the LPM function you chose in step 3 and a corresponding VHDL simulation model.
6. Continue with the steps necessary to complete your Design Architect schematic, as described in Creating Design Architect Schematics for Use with MAX+PLUS II Software.
7. When you save the schematic, the Design Architect software will ask whether you want to compile the LPM model. Choose **YES** if you want to compile the VHDL code for the LPM functions. The software will automatically select the corresponding compiler: System 1076 for B.(*x*) releases and QuickHDL compilers for releases C.1 and later.

Installing the Altera® provided MAX+PLUS II/Mentor Graphics/Exemplar Logic interface on your computer automatically creates the sample Design Architect schematic file **/usr/maxplus2/examples/mentor/example7/fifo**, which includes LPM instantiation.

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Instantiating the clklock Megafunction in VHDL & Verilog HDL Designs

Altera provides the **genckllk** utility to allow you to instantiate clklock (phaselocked loop) functions in Mentor Graphics/Exemplar Logic software. The **genckllk** utility appends the parameter values to the clklock function name, so you don't need to declare attributes explicitly. The naming rule for the clklock function is clklock_ <*ClockBoost*>_<*inputfrequency*>. The **genckllk** utility has the following syntax:

genckllk <*ClockBoost*> <*inputfrequency*> [vhdl] [verilog] ↵

For the <*ClockBoost*> variable, you should specify a ClockBoost value of 1 or 2 (default value is 1). For the <*inputfrequency*> variable, you should specify a decimal value in MHz (default value is 50). To generate a VHDL file (which is the default if no option is present), specify vhdl; to generate a Verilog HDL file, specify verilog.

For example, to create the VHDL file **clklock_2_50.vhd** and the corresponding Component Declaration file **clklock_2_50.cmp**, type the following command at the UNIX prompt:

genckllk 2 50 -vhdl ↵

Installing the Alteraprovided MAX+PLUS II/Mentor Graphics interface on your computer automatically creates the sample VHDL design file **/usr/maxplus2/examples/mentor/example6/count8.vhd**, which includes clklock megafunction instantiation.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

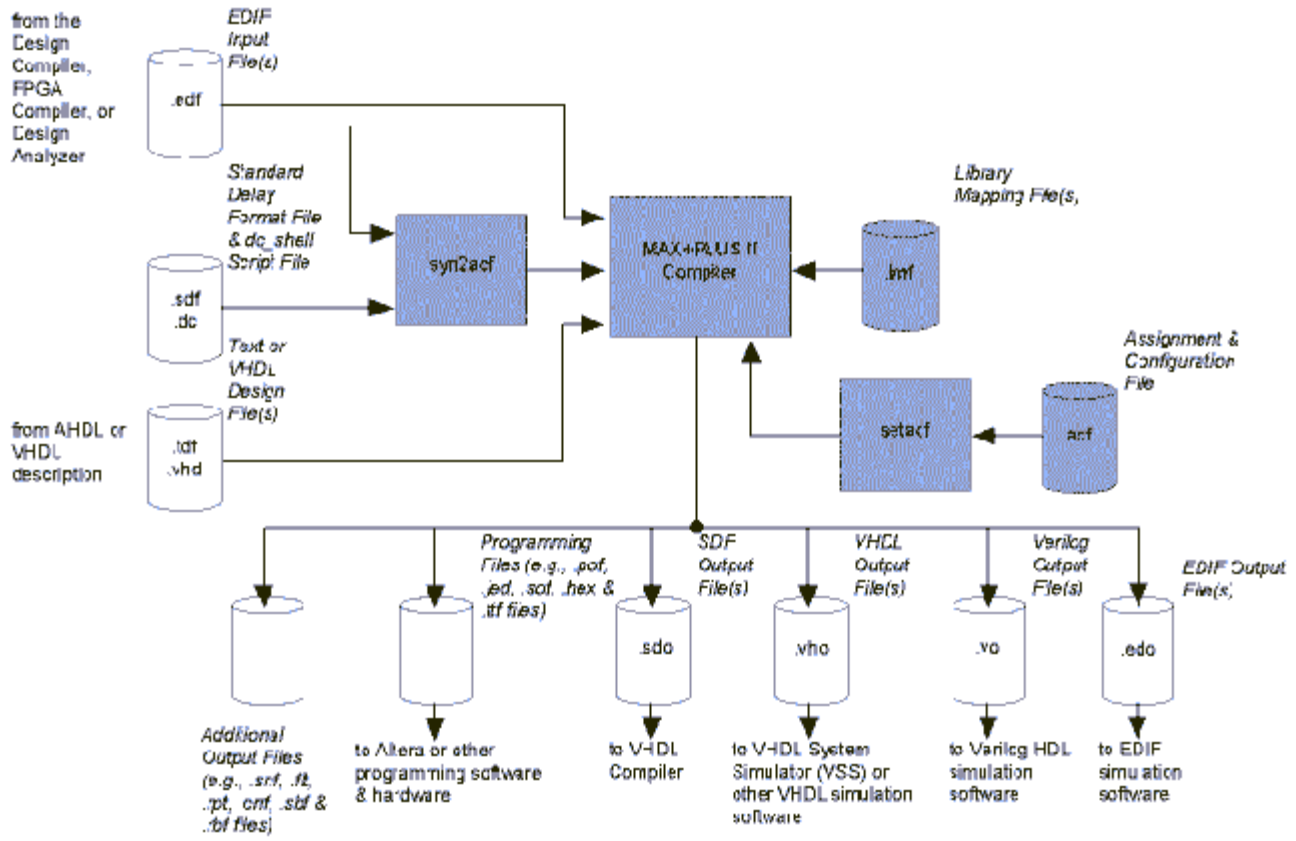
Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Altera/Synopsys Project Compilation Flow

The following figure shows the MAX+PLUS® II /Synopsys project compilation flow.

Figure 1. MAX+PLUS II/Synopsys Project Compilation Flow

Altera-provided items are shown in blue.



Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Design Flow for All Synopsys Tools

Figure 1 shows the design flow between Synopsys and MAX+PLUS ® II software. Design Entry Flow, Project Compilation Flow, Project Simulation Flow, and Device Programming Flow show detailed diagrams of each stage of the design flow. For information on how to use the Synopsys Design Compiler or FPGA Compiler from within the MAX+PLUS II software, see Running Synopsys Compilers from the MAX+PLUS II Software.

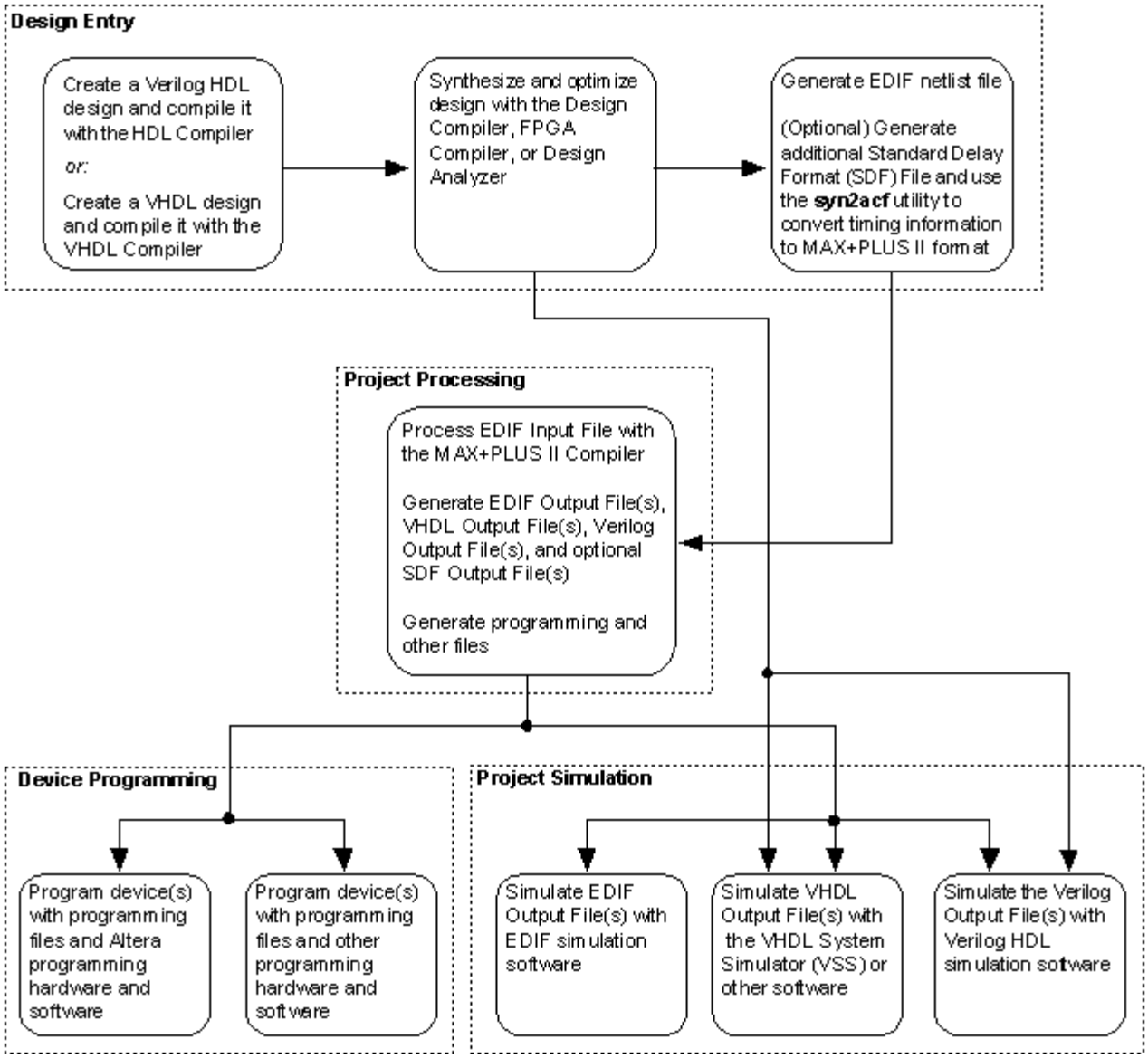


Figure 1. Design Flow between Synopsys & MAX+PLUS II Software

---

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Setting Up the MAX+PLUS II/Synopsys Working Environment

To use the MAX+PLUS® II software with Synopsys software, you must first install the MAX+PLUS II software, then establish an environment that facilitates entering and processing designs by modifying your Synopsys configuration files. The MAX+PLUS II/Synopsys interface is installed automatically when you install the MAX+PLUS II software on your workstation. Go to *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual for more information on installation and details on the directories that are created during MAX+PLUS II installation. Go to MAX+PLUS II/Synopsys Interface File Organization for information about the MAX+PLUS II/Synopsys directories that are created during MAX+PLUS II installation.

☞ The information presented here assumes that you are using C shell and that your MAX+PLUS II system directory is **/usr/maxplus2**. If not, you must use the appropriate syntax and procedures to set environment variables for your shell.

To set up your working environment for the MAX+PLUS II/Synopsys interface, follow these steps:

1. Ensure that you have correctly installed the MAX+PLUS II and Synopsys software versions described in the MAX+PLUS II/Synopsys Software Requirements.
2. Add technology, synthetic, and link library settings to your **.synopsys_dc.setup** configuration file, as described in Setting Up Design Compiler & FPGA Compiler Configuration Files.

   ☞ To use the DesignWare interface with FLEX® 6000, FLEX 8000, and FLEX 10K devices, follow the steps in Setting Up the DesignWare Interface.
3. Add simulation library settings to your **.synopsys_vss.setup** file, and analyze the libraries, as described in Setting Up VSS Configuration Files.
4. Add the **/usr/maxplus2/bin** directory to the `PATH` environment variable in your **.cshrc** file in order to run the MAX+PLUS II software.

   (Optional) Change the path in the first line of the perl script files, which are located in the `$ALT_HOME`**/synopsys/bin** directory to specify the correct path of your local perl executable file.

Related Topics:

- Go to the following topics, which are available on the web, for additional information:
  - FLEX Devices
  - ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
  - This manual is also available in 4 parts:
    - Preface & Section 1: MAX+PLUS II Installation
    - Section 2: MAX+PLUS II - A Perspective
    - Section 3: MAX+PLUS II Tutorial
    - Appendices, Glossary & Index

MAX+PLUS II/Synopsys Software Requirements

The following applications are used to generate, process, synthesize, and verify a project with MAX+PLUS® II and Synopsys software:

Synopsys		Altera
version 1998.02: Design Compiler FPGA Compiler Design Analyzer (optional) VHDL Compiler	HDL Compiler for Verilog VHDL System Simulator (VSS) (optional) PrimeTime version 1998.02-PT2.1(optional)	MAX+PLUS II version 9.4

Compilation with the Synopsys Design Compiler and FPGA Compiler is available only on Sun SPARCstations running Solaris 2.4 or higher.

☞ The MAX+PLUS II **read.me** file provides up-to-date information on which versions of Synopsys applications are supported by the current version of MAX+PLUS II. It also provides information on installation and operating requirements. You should read the **read.me** file on the CD-ROM before installing the MAX+PLUS II software. After installation, you can open the **read.me** file from the MAX+PLUS II Help menu.

Setting Up Design Compiler & FPGA Compiler Configuration Files

The **.synopsys_dc.setup** configuration file allows you to set both Design Compiler and FPGA Compiler variables. The compilers read **.synopsys_dc.setup** files from three directories, in the following order:

1. The Synopsys root directory
2. Your home directory
3. The directory where you start the Design Compiler or FPGA Compiler software

The most recently read configuration file has highest priority. For example, a configuration file in the directory where you start the Design Compiler or FPGA Compiler software has priority over the other configuration files, and a configuration file in the home directory has priority over a configuration file in the root directory.

To set up your configuration files, follow these steps:

1. Add the lines shown in Figure 1 to your **.synopsys_dc.setup** configuration file. Altera provides a sample **.synopsys_dc.setup** file in the **./synopsys/config** directory. Figure 1 shows an excerpt from that sample file.

   Figure 1. Excerpt from Sample .synopsys_dc.setup File

   ```
   search_path = {./usr/maxplus2/synopsys/library/alt_syn/<device family>/lib};
   target_library = {<technology library>};
   symbol_library = {altera.sdb};
   link_library = {<technology library>};
   edifout_netlist_only = "true"
   edifout_power_and_ground_representation = "net"
   edifout_power_net_name = "VDD"
   edifout_ground_net_name = "GND"
   edifout_no_array = "false"
   edifin_power_net_name = "VDD"
   edifin_ground_net_name = "GND"
   compile_fix_multiple_port_nets = "true"
   ```

```
bus_naming_style = "%s<%d>"
bus_dimension_separator_style = "><"
bus_inference_style = "%s<%d>"
```

2. Specify one of the Design Compiler & FPGA Compiler Technology Libraries for the `target_library` and `link_library` parameters in the **.synopsys_dc.setup** file.
3. If you will instantiate architecture control logic functions from the **alt_mf** library, add the following line to your **.synopsys_dc.setup** file:

   ```
   define_design_lib altera -path /usr/maxplus2/synopsys/library/alt_mf/lib ↵
   ```

   ☞ If you wish to use the VHDL System Simulator (VSS) software to simulate a VHDL design containing **alt_mf** library functions, you must compile this library with the **analyze_vss** script. See Setting Up VSS Configuration Files for more information.

4. If you will use the DesignWare interface for FLEX® 6000, FLEX 8000, or FLEX 10K designs, enter additional lines in your **.synopsys_dc.setup** file, as described in Setting Up the DesignWare Interface.
5. Specify one of the following families for the <*device family*> variable in the `search_path` parameter: `max5000`, `max7000`, `max9000`, `flex6000`, `flex8000`, or `flex10k`.
6. If you wish to resynthesize a design for a different device family, modify the **.synopsys_dc.setup** file by following the steps described in Resynthesizing a Design Using the **alt_vtl** Library & a MAX+PLUS II SDF Output File.

Related Topics:

- Go to MAX+PLUS® II /Synopsys Interface File Organization in these MAX+PLUS II ACCESS℠ Key topics for related information.
- Go to the following topics, which are available on the web, for additional information:
  - FLEX Devices
  - MAX Devices
  - Classic Device Family

Setting Up the DesignWare Interface

The DesignWare interface synthesizes FLEX® 6000 , FLEX 8000 and FLEX 10K designs by operator inference. It replaces the HDL operators +, -, >, <, >=, and <= with FLEX-optimized design implementations.

Altera provides DesignWare Synthetic Libraries that are pre-compiled for the current version of Synopsys tools. These library files are located in the **/usr/maxplus2/synopsys/library/alt_syn/<***device family***>/lib** directory.

To use the DesignWare interface with FLEX 6000, FLEX 8000 and FLEX 10K devices, follow these steps:

1. Add `synthetic_library` and `define_design_lib` parameters to your **.synopsys_dc.setup** configuration file and modify the `link_library` parameter as shown in Table 1 or Table 2.

Table 1. DesignWare Parameters to Add to the .synopsys_dc.setup File for the Design Compiler Software

Device Family	*Parameters to Add to the .synopsys_dc.setup File*
	`synthetic_library = {flex6000<`*speed grade*`>.sldb};` ↵

FLEX 6000	`link_library = {flex6000<`*speed grade*`>.sldb flex6000<`*speed grade*`>.db};` ↵
	`define_design_lib DW_FLEX6000<`*speed grade*`> -path`
	`/usr/maxplus2/synopsys/library/alt_syn/flex6000/lib/`
	`dw_flex6000<`*speed grade*`>` ↵

FLEX 8000	`synthetic_library = {flex8000[<`*speed grade*`>].sldb};` ↵
	`link_library = {flex8000[<`*speed grade*`>].sldb flex8000[<`*speed grade*`>].db};` ↵
	`define_design_lib DW_FLEX8000[<`*speed grade*`>] -path`
	`/usr/maxplus2/synopsys/library/alt_syn/flex8000`
	`/lib/dw_flex8000[<`*speed grade*`>]` ↵

FLEX 10K	`synthetic_library = {flex10k[<`*speed grade*`>].sldb};` ↵
	`link_library = {flex10k[<`*speed grade*`>].sldb flex10k[<`*speed grade*`>].db};` ↵
	`define_design_lib DW_FLEX10k[<`*speed grade*`>] -path`
	`/usr/maxplus2/synopsys/library/alt_syn/flex10k/lib`
	`/dw_flex10k[<`*speed grade*`>]` ↵

Table 2. DesignWare Parameters to Add to the .synopsys_dc.setup File for the FPGA Compiler Software

Device Family	*Parameters to Add to the .synopsys_dc.setup File*

FLEX 6000	`synthetic_library = {flex6000`
	`<`*speed grade*`>_fpga.sldb};` ↵
	`link_library = {flex6000<`*speed grade*`>_fpga.sldb flex6000<`*speed grade*`>_fpga.db};` ↵
	`define_design_lib DW_FLEX6000<`*speed grade*`>_FPGA -path`
	`/usr/maxplus2/synopsys/library/alt_syn/flex6000`
	`/lib/dw_flex6000<`*speed grade*`>_fpga` ↵

FLEX 8000	`synthetic_library = {flex8000[<`*speed grade*`>]_fpga.sldb};` ↵ `link_library = {flex8000[<`*speed grade*`>]_fpga.sldb flex8000[<`*speed grade*`>]_fpga.db};` ↵ `define_design_lib DW_FLEX8000[<`*speed grade*`>]_FPGA -path` `/usr/maxplus2/synopsys/library/alt_syn/flex8000/lib /dw_flex8000[<`*speed grade*`>]_fpga` ↵
FLEX 10K	`synthetic_library = {flex10k[<`*speed grade*`>]_fpga.sldb};` ↵ `link_library = {flex10k[<`*speed grade*`>]_fpga.sldb flex10k[<`*speed grade*`>]_fpga.db};` ↵ `define_design_lib DW_FLEX10k[<`*speed grade*`>]_FPGA -path` `/usr/maxplus2/synopsys/library/alt_syn/flex10k/lib /dw_flex10k[<`*speed grade*`>]_fpga` ↵

2. Specify the libraries listed in Table 3 as your synthetic library and as the first of your link libraries.

   For FLEX 6000 devices, you must specify either `-2` or `-3` for the <*speed grade*> variable. For FLEX 8000 and FLEX 10K devices, you can specify `-2`, `-3`, `-4`, `-5`, or `-6`; or `-2`, `-3`, `-4`, or `-5`; respectively, for the <*speed grade*> variable. If you do not specify a speed grade for FLEX 8000 or FLEX 10K devices, the MAX+PLUS® II software selects the fastest device in the specified family as the target device.

Table 3. FLEX 6000, FLEX 8000 & FLEX 10K DesignWare Synthetic Libraries

Altera® Device Family	Synopsys Design Compiler	Synopsys FPGA Compiler
FLEX 6000 Synthetic Library	**flex6000-2.sldb**	**flex6000-2_fpga.sldb**
	flex6000-3.sldb	**flex6000-3_fpga.sldb**
FLEX 8000 Synthetic Library	**flex8000.sldb**	**flex8000_fpga.sldb**
	flex8000-2.sldb	**flex8000-2_fpga.sldb**
	flex8000-3.sldb	**flex8000-3_fpga.sldb**
	flex8000-4.sldb	**flex8000-4_fpga.sldb**

FLEX 10K Synthetic Library	**flex8000-5.sldb** **flex8000-6.sldb**	**flex8000-5_fpga.sldb** **flex8000-6_fpga.sldb**
	flex10k.sldb **flex10k-2.sldb** **flex10k-3.sldb** **flex10k-4.sldb** **flex10k-5.sldb**	**flex10k_fpga.sldb** **flex10k-2_fpga.sldb** **flex10k-3_fpga.sldb** **flex10k-4_fpga.sldb** **flex10k-5_fpga.sldb**

3. If necessary, compile the DesignWare libraries, as described in Updating DesignWare Libraries. Altera provides pre-compiled DesignWare libraries, as described above. However, Altera also provides compatible source files and scripts that allow you to automate the compilation process. These source files allow you to use DesignWare with any version of the Design Compiler. They also allow you to install components whose source is written in VHDL, even if you are licensed only for the HDL Compiler for Verilog.

Related Topics:

- Go to the following MAX+PLUS II ACCESS^{SM} Key topics for related information:
- Setting Up Design Compiler & FPGA Compiler Configuration Files
- DesignWare FLEX 8000 Synthesis Example
- Design Compiler & FPGA Compiler Technology Libraries

- FLEX 6000 Device Family
- FLEX 8000 Device Family
- FLEX 10K Device Family

Updating DesignWare Libraries

Although Altera provides DesignWare libraries that are pre-compiled for the current version of Synopsys tools, you may wish to recompile the libraries.

Altera provides compilable source files and scripts that allow you to automate the compilation process. These source files allow you to use DesignWare software with any version of the Design Compiler or FPGA Compiler software.They also allow you to install components whose source is written in VHDL, even if you are licensed only for the Verilog HDL Compiler software.

Source files for the Design Compiler software are automatically installed in the following directories:

- **/usr/maxplus2/synopsys/library/alt_syn/flex10k/src/dw_flex10k[<*speed grade*>]**
- **/usr/maxplus2/synopsys/library/alt_syn/flex8000/src/dw_flex8000[<*speed grade*>]**
- **/usr/maxplus2/synopsys/library/alt_syn/flex6000/src/dw_flex6000<*speed grade*>**

Source files for the FPGA Compiler are automatically installed in the following directories:

- **/usr/maxplus2/synopsys/library/alt_syn/flex10k/src/dw_flex10k[<*speed grade*>]_fpga**
- **/usr/maxplus2/synopsys/library/alt_syn/flex8000/src/dw_flex8000[<*speed grade*>]_fpga**
- **/usr/maxplus2/synopsys/library/alt_syn/flex6000/src/dw_flex6000<*speed grade*>_fpga**

Table 1. Commands for Compiling the Library

Device Family	Synopsys Compiler	Commands for Compiling the Library *Note (1)*
FLEX®6000	Design Compiler	`cd /usr/maxplus2/synopsys/library/alt_syn/flex6000/src/dw_flex6000<speed grade>` ↵ `dw_flex6000.script` ↵

FLEX 8000	FPGA Compiler	`cd /usr/maxplus2/synopsys/library/alt_syn/flex6000/src/dw_flex6000<`*speed grade*`>_fpga ↵` `dw_flex6000.script ↵`
	Design Compiler	`cd /usr/maxplus2/synopsys/library/alt_syn/flex8000/src/dw_flex8000[<`*speed grade*`>] ↵` `dw_flex8000.script ↵`
	FPGA Compiler	`cd /usr/maxplus2/synopsys/library/alt_syn/flex8000/src/dw_flex8000[<`*speed grade*`>]_fpga ↵` `dw_flex8000.script ↵`
FLEX 10K	Design Compiler	`cd /usr/maxplus2/synopsys/library/alt_syn/flex10k/src/dw_flex10k[<`*speed grade*`>] ↵` `dw_flex10k.script ↵`
	FPGA Compiler	`cd /usr/maxplus2/synopsys/library/alt_syn/flex10k/src/dw_flex10k[<`*speed grade*`>]_fpga ↵` `dw_flex10k.script ↵`

1. For FLEX 6000 devices, you must specify either `-2` or `-3` for the <*speed grade*> variable. For FLEX 8000 and FLEX 10K devices, you must specify `-2`, `-3`, `-4`, `-5`, or `-6`; or `-1`, `-2`, `-3`, `-4`, or `-5`; respectively, for the <*speed grade*> variable.

Related Topics:

- Go to the following topics for additional information:
  - Setting Up the DesignWare Interface
  - Setting Up the MAX+PLUS II/Synopsys Working Environment
  - Setting Up Design Compiler & FPGA Compiler Configuration Files
  - Setting Up VSS Configuration Files
- Go to the following topics, which are available on the web, for additional information:
- FLEX 6000 Device Family
- FLEX 8000 Device Family
- FLEX 10K Device Family

Setting Up VSS Configuration Files

The **.synopsys_vss.setup** file contains the mapping information that directs the VHDL System Simulator (VSS) Software to use Altera® -supplied Altera Simulation Libraries during simulation. To configure your environment for the MAX+PLUS® II /Synopsys interface, follow these steps:

1. Add the lines shown in Figure 1 to your **.synopsys_vss.setup** file. Altera provides a sample setup file, **.synopsys_vss.setup**, in the **/usr/maxplus2/synopsys/config** directory. See Figure 1.

   Figure 1. Sample .synopsys_vss.setup File

```
WORK > DEFAULT
DEFAULT             : .


altera              : /usr/maxplus2/synopsys/library/alt_mf/lib


flex_vtl            : /usr/maxplus2/synopsys/library/alt_pre/vital/
                          lib/flex_vtl
```

```
alt_vtl            : /usr/maxplus2/synopsys/library/alt_post/sim/
                        lib/alt_vtl
flex10k_ftsm       : /usr/maxplus2/synopsys/library/alt_pre/flex10k/
                        lib/flex10k_ftsm
flex10k_ftgs       : /usr/maxplus2/synopsys/library/alt_pre/flex10k/
                        lib/flex10k_ftgs
max9000_ftsm       : /usr/maxplus2/synopsys/library/alt_pre/max9000/
                        lib/max9000_ftsm
max9000_ftgs       : /usr/maxplus2/synopsys/library/alt_pre/max9000/
                        lib/max9000_ftgs
flex8000_ftsm      : /usr/maxplus2/synopsys/library/alt_pre/flex8000/
                        lib/flex8000_ftsm
flex8000_ftgs      : /usr/maxplus2/synopsys/library/alt_pre/flex8000/
                        lib/flex8000_ftgs
max7000_ftsm       : /usr/maxplus2/synopsys/library/alt_pre/max7000/
                        lib/max7000_ftsm
max7000_ftgs       : /usr/maxplus2/synopsys/library/alt_pre/max7000/
                        lib/max7000_ftgs
flex6000_ftsm      : /usr/maxplus2/synopsys/library/alt_pre/flex6000/
                        lib/flex6000_ftsm
flex6000_ftgs      : /usr/maxplus2/synopsys/library/alt_pre/flex6000/
                        lib/flex6000_ftgs
max5000_ftsm       : /usr/maxplus2/synopsys/library/alt_pre/max5000/
                        lib/max5000_ftsm
max5000_ftgs       : /usr/maxplus2/synopsys/library/alt_pre/max5000/
                        lib/max5000_ftgs
flex10k_fpga_ftsm  : /usr/maxplus2/synopsys/library/alt_pre/flex10k/
                        lib/flex10k_fpga_ftsm
flex10k_fpga_ftsm  : /usr/maxplus2/synopsys/library/alt_pre/flex10k/
                        lib/flex10k_fpga_ftgs
max9000_fpga_ftsm  : /usr/maxplus2/synopsys/library/alt_pre/max9000/
                        lib/max9000_fpga_ftsm
max9000_fpga_ftgs  : /usr/maxplus2/synopsys/library/alt_pre/max9000/
                        lib/max9000_fpga_ftgs
flex8000_fpga_ftsm : /usr/maxplus2/synopsys/library/alt_pre/flex8000/
                        lib/flex8000_fpga_ftsm
flex8000_fpga_ftgs : /usr/maxplus2/synopsys/library/alt_pre/flex8000/
                        lib/flex8000_fpga_ftgs
max7000_fpga_ftsm  : /usr/maxplus2/synopsys/library/alt_pre/max7000/
                        lib/max7000_fpga_ftsm
max7000_fpga_ftgs  : /usr/maxplus2/synopsys/library/alt_pre/max7000/
                        lib/max7000_fpga_ftgs
flex6000_fpga_ftsm : /usr/maxplus2/synopsys/library/alt_pre/flex6000/
                        lib/flex6000_fpga_ftsm
flex6000_fpga_ftgs : /usr/maxplus2/synopsys/library/alt_pre/flex6000/
                        lib/flex6000_fpga_ftgs
max5000_fpga_ftsm  : /usr/maxplus2/synopsys/library/alt_pre/max5000/
                        lib/max5000_fpga_ftsm
max5000_fpga_ftgs  : /usr/maxplus2/synopsys/library/alt_pre/max5000/
                        lib/max5000_fpga_ftgs
```

The variables in the **.synopsys_vss** setup file perform the following functions:

- The WORK variable specifies your working directory, i.e., the directory where you start the Synopsys tools. If not explicitly specified elsewhere, the results of any analysis or compilation are written to this directory. The first line of the file shown in Figure 1 maps WORK to the design library variable called DEFAULT.
- The DEFAULT variable is used to create library aliases, which allows you to map the WORK variable to various paths. In Figure 1, the DEFAULT variable specifies the current directory.
- The **altera** library is listed to allow you to simulate the architecture control logic functions in the **alt_mf** library.
- The remaining lines in the file specify the path and name of the directories that contain the device simulation libraries for Altera device families.

2. Analyze the target device simulation library to ensure that the correct timing and functional information is provided to VSS. Analyzing the simulation library produces VSS simulation models of the primitives that appear in all Altera-provided technology libraries.

You can analyze device simulation libraries by using the Altera-provided shell script **analyze_vss**:

1. Add the **/usr/maxplus2/synopsys/bin** directory, which contains the **analyze_vss** scripts, to the `PATH` environment variable in your **.cshrc** file.
2. Make sure that you have write privileges for the **/usr/maxplus2/synopsys/library/alt_pre/**<*device family*> directory because the analyzed model is placed in the **/usr/maxplus2/synopsys/library/alt_pre/**<*device family*>**/lib** directory and the analysis log file is placed in the **./synopsys/library/alt_pre/**<*device family*>**/src** directory.
3. Run the **analyze_vss** shell script by typing `analyze_vss` ↵ at the **dc_shell** prompt. When you run the **analyze_vss** shell script, you are prompted to select the appropriate device family simulation model(s) for analysis. Figure 2 shows the **analyze_vss** shell script.

   Figure 2. The analyze_vss Shell Script

   Type the full pathname of the directory where the MAX+PLUS® II software is installed (default: **/usr/maxplus2**):

   <*MAX+PLUS II system directory*> ↵

   ```
   Analyze VSS Simulation Models:
   1. flex10k_FTGS
   2. flex10k_FTSM
   3. flex10k_fpga_FTGS
   4. flex10k_fpga_FTSM
   5. max9000_FTGS
   6. max9000_FTSM
   7. max9000_fpga_FTGS
   8. max9000_fpga_FTSM
   9. flex8000_FTGS
   10. flex8000_FTSM
   11. flex8000_fpga_FTGS
   12. flex8000_fpga_FTSM
   13. max7000_FTGS
   14. max7000_FTSM
   15. max7000_fpga_FTGS
   16. max7000_fpga_FTSM
   17. flex6000_FTGS
   18. flex6000_FTSM
   19. flex6000_fpga_FTGS
   20. flex6000_fpga_FTSM
   21. max5000_FTGS
   22. max5000_FTSM
   23. max5000_fpga_FTGS
   24. max5000_fpga_FTSM
   25. alt_vtl
   26. flex_vtl
   27. Quit
   ```

   Enter one or more numbers: <*device library numbers*> ↵

4. Check the log file to make sure that no errors occurred during the analysis of the simulation models.

3. Use VSS to simulate your pre-routed VHDL design.

Related Topics:

- Refer to the ***VHDL System Simulator Core Programs Manual*** for more information about VSS.

Design Compiler & FPGA Compiler Technology Libraries

The Altera® -provided Design Compiler and FPGA Compiler technology libraries contain primitives that the Synopsys compilers use to map your designs to the target device architecture. These primitives contain timing and area information that the Synopsys compilers use to meet area and performance requirements. Table 1 shows the functions provided in these libraries. Choose **Primitives** from the MAX+PLUS II Help menu for detailed information on these functions.

Altera recommends instantiating these functions directly in your designs only if the Synopsys compilers do not appear to recognize the functions when synthesizing your design, or if you prefer to hand-optimize certain portions of your design.

Table 1. Altera-Provided Primitives

Name *Note (1), Note (2)*	Description	Name	Description
LCELL	Logic cell buffer primitive	EXP	MAX® 5000, MAX 7000, and MAX 9000 Expander buffer primitive
GLOBAL	Global input buffer primitive	SOFT	Soft buffer primitive
CASCADE	FLEX® 6000, FLEX 8000, and FLEX 10K cascade buffer primitive	OPNDRN	FLEX 6000, FLEX 8000, and FLEX 10K Open-drain buffer primitive
CARRY	FLEX 6000, FLEX 8000, and FLEX 10K cascade buffer primitive	DFF DFFE DFFS *Note (2)*	D-type flipflop with Clock Enable primitive
LATCH	Latch primitive	TFF TFFE TFFS *Note (2)*	T-type flipflop primitive
TRIBUF	Tri-state buffer primitive		

Notes:

(1) All buffer primitive names except OPNDRN must be prefixed with an "A" in FLEX 6000, FLEX 8000, and FLEX 10K designs. The TRIBUF primitive is equivalent to the TRI primitive in the MAX+PLUS II software.

(2) The DFFE and TFFE primitives include a Clock Enable input; the DFFS and TFFS primitives are equivalent to DFF and TFF primitives without Clear or Preset inputs. For designs that are targeted to FLEX 6000 devices, you should use the DFFE or TFFE primitive only if the design contains either a Clear or Preset signal, but not both. If your design contains both a Clear and a Preset signal, you must use the DFFE6K primitive.

The VHDL simulation model **/usr/maxplus2/synopsys/library/alt_pre/<***device family***>/src/<***device family***>_components.vhd** file shows the exact cell and pin names for each device family. The Verilog HDL simulation file **/usr/maxplus2/synopsys/library/alt_pre/verilog/src/altera.v** shows the functionality of these cells.

Table 2 lists the technology library names.

Table 2. Altera Technology Libraries

Altera Device Family	Synopsys Design Compiler	Synopsys FPGA Compiler
	flex10k.db **flex10k-2.db**	**flex10k_fpga.db** **flex10k-2_fpga.db**

FLEX® 10K devices	**flex10k-3.db** **flex10k-4.db** **flex10k-5.db**	**flex10k-3_fpga.db** **flex10k-4_fpga.db** **flex10k-5_fpga.db**
FLEX 8000 devices	**flex8000.db** **flex8000-2.db** **flex8000-3.db** **flex8000-4.db** **flex8000-5.db** **flex8000-6.db**	**flex8000_fpga.db** **flex8000-2_fpga.db** **flex8000-3_fpga.db** **flex8000-4_fpga.db** **flex8000-5_fpga.db** **flex8000-6_fpga.db**
FLEX 6000 devices	**flex6000-2.db** **flex6000-3.db**	**flex6000-2_fpga.db** **flex6000-3_fpga.db**
MAX® 9000 devices	**max9000.db**	**max9000_fpga.db**
MAX 7000, MAX 7000E, MAX 7000S, & MAX 7000A devices	**max7000.db**	**max7000_fpga.db**
MAX 5000 & Classic® devices	**max5000.db**	**max5000_fpga.db**

Related Topics:

- Go to MAX+PLUS® II /Synopsys Interface File Organization in these MAX+PLUS II ACCESS℠ Key topics for related information.
- Go to the following topics, which are available on the web, for additional information:
  - FLEX Devices
  - MAX Devices
  - Classic Device Family

Altera VHDL & Verilog HDL alt_mf Logic Function Library

The **alt_mf** library contains behavioral VHDL and Verilog HDL models of the Altera® logic functions shown in Table 1. VHDL or Verilog HDL files that instantiate these functions can be simulated with the VHDL System Simulator (VSS) software or the Cadence Verilog-XL simulator, respectively, both before and after being compiled with the Synopsys Design Compiler or FPGA Compiler software.

Table 1. Altera-Provided Architecture Control Logic Functions

Name	Description
`a_8fadd`	8-bit full adder
`a_8mcomp`	8-bit magnitude comparator
`a_8count`	8-bit up/down counter
`a_81mux`	8-to-1 multiplexer

☞ For detailed information on these functions, choose **Search for Help** on from the MAX+PLUS® II Help menu and type the function name, without the "`a_`" prefix.

The behavioral descriptions of these four functions are contained in the **/usr/maxplus2/synopsys/library/alt_mf/src** directory, which contains the following files:

File:	Description:
mf.vhd	Contains behavioral VHDL descriptions of the logic functions.

mf_components.vhd Contains VHDL Component Declarations for the logic functions.
mf.v Contains behavioral Verilog HDL descriptions of the logic functions.

If you wish to simulate a VHDL design containing these logic functions, you can use the Altera-provided shell script **analyze_vss** to create a design library called **altera**. This library allows you to reference the functions through the VHDL Library and Use Clauses, which direct the Design Compiler or FPGA Compiler software to incorporate the library files when it compiles your top-level design file. The **analyze_vss** shell script creates the **altera** design library by analyzing the VHDL System Simulator (VSS) simulation models in the **/usr/maxplus2/synopsys/library/alt_mf/lib** directory. See Setting Up VSS Configuration Files for more information on using the **analyze_vss** shell script.

Complete VHDL and Verilog HDL behavioral descriptions of these logic functions are included in the **mf.vhd** and **mf.v** files so that you can optionally retarget your design to other technology libraries.

Altera DesignWare FLEX 6000, FLEX 8000 & FLEX 10K Synthetic Libraries

The Altera® DesignWare interface for the FLEX® 6000, FLEX 8000, and FLEX 10K device families provides accurate area and timing prediction for designs that have been synthesized by the Synopsys design tools and targeted for FLEX devices. Altera's DesignWare interface also ensures that the area and timing information closely matches the final FLEX device implementation generated by the MAX+PLUS® II Compiler. The DesignWare interface synthesizes FLEX 6000 , FLEX 8000 and FLEX 10K designs by operator inference. This interface supports bus widths of up to 32 bits, except adder functions, which support bus widths of up to 64 bits.

The Altera DesignWare interface for FLEX devices offers three major advantages to Synopsys designers:

- Automatic access to FLEX carry and cascade chain functions
- Optimal routing of FLEX designs
- Improved area and performance prediction capability in Synopsys tools

Table 1 lists the Altera DesignWare synthetic libraries for FLEX 6000, FLEX 8000, and FLEX 10K devices.

Table 1. FLEX 6000, FLEX 8000 & FLEX 10K DesignWare Synthetic Libraries

Altera Device Family	Synopsys Design Compiler	Synopsys FPGA Compiler
FLEX 6000 Synthetic Library	**flex6000-2.sldb** **flex6000-3.sldb**	**flex6000-2_fpga.sldb** **flex6000-3_fpga.sldb**
FLEX 8000 Synthetic Library	**flex8000.sldb** **flex8000-2.sldb** **flex8000-3.sldb** **flex8000-4.sldb** **flex8000-5.sldb** **flex8000-6.sldb**	**flex8000_fpga.sldb** **flex8000-2_fpga.sldb** **flex8000-3_fpga.sldb** **flex8000-4_fpga.sldb** **flex8000-5_fpga.sldb** **flex8000-6_fpga.sldb**
FLEX 10K Synthetic Library	**flex10k.sldb** **flex10k-2.sldb** **flex10k-3.sldb** **flex10k-4.sldb** **flex10k-5.sldb**	**flex10k_fpga.sldb** **flex10k-2_fpga.sldb** **flex10k-3_fpga.sldb** **flex10k-4_fpga.sldb** **flex10k-5_fpga.sldb**

Table 2 lists functions included in the DesignWare FLEX 6000, FLEX 8000, and FLEX 10K synthetic libraries. Refer to DesignWare FLEX 8000 Synthesis Example for an example showing how DesignWare synthesis affects design processing.

Table 2. FLEX 6000, FLEX 8000, and FLEX 10K Synthetic Library Functions

Name	Function
`flex_add`	Sum of A, B, and Carry-In
`flex_carry`	Carry of A, B, and Carry-In
`flex_sub`	Difference of A, B, and Borrow-In
`flex_borrow`	Borrow of A, B, and Borrow-In
`flex_gt, flex_sgt`	Greater than (`flex_gt` is unsigned; `flex_sgt` is signed)
`flex_carry_gt`	Greater than Carry
`flex_lt, flex_slt`	Less than (`flex_lt` is unsigned; `flex_slt` is signed)
`flex_carry_lt`	Less than Carry
`flex_gteq, flex_sgteq`	Greater than or equal to (`flex_gteq` is unsigned; `flex_sgteq` is signed)
`flex_carry_gteq`	Greater than or equal to Carry
`flex_inc`	Incrementer (Count = Count + 1)
`flex_carry_inc`	Incrementer Carry (Count = Count + 1)
`flex_dec`	Decrementer (Count = Count - 1)
`flex_carry_dec`	Decrementer Carry (Count = Count - 1)
`flex_lteq, flex_slteq`	Less than or equal to (`flex_lteq` is unsigned; `flex_slteq` is signed)
`flex_carry_lteq`	Less than or equal to Carry
`flex_count`	Counter
`aflex_carry_count`	Counter Carry
`flex_add_sub`	Adder/Subtractor
`flex_inc_dec`	Incrementer/Decrementer
`flex_umult, flex_smult`	Multiplier (`flex_umult` is unsigned; `flex_smult` is signed)

Related Topics:

- Go to the following sources for related information:
  - Setting Up the DesignWare Interface in these MAX+PLUS II ACCESS^{SM} Key topics
  - ***Synopsys DesignWare Databook***
  - ***VHDL Compiler Reference Manual***
- Go to the following topics, which are available on the web, for additional information:
  - FLEX 6000 Device Family
  - FLEX 8000 Device Family
  - FLEX 10K Device Family

Altera Simulation Libraries

Altera provides simulation libraries for both pre-routing functional simulation and post-routing timing simulation.

Pre-Routing Functional Simulation Libraries (VITAL-Compliant)

The **/usr/maxplus2/synopsys/library/alt_pre/vital/src** directory contains Altera® -provided VHDL simulation models in VITAL 95 format. This library contains functional descriptions of all primitives that appear in Altera-specific technology libraries. These libraries allow you to perform a functional or pre-routing simulation that

verifies the netlist structure generated by the Synopsys Design Compiler or FPGA Compiler software. Altera provides the **flex.cmp** and **flex.vhd** files in the **/usr/maxplus2/synopsys/library/alt_pre/vital/src** directory.

Similarly, the **/usr/maxplus2/synopsys/library/alt_pre/verilog/src** directory contains Altera-provided Verilog HDL simulation models for all device families. The **altera.v** file can be used for simulation with the Cadence Verilog-XL simulator.

Pre-Routing Functional Simulation Libraries with Estimated Timing Information

The **/usr/maxplus2/synopsys/library/alt_pre/**<*device family*>**/src** directory contains Altera® -provided VHDL simulation libraries, which give both functional and area descriptions of all primitives that appear in all Altera technology libraries. These simulation libraries allow you to verify the function of VHDL projects, with estimated timing, after synthesizing them with the Synopsys Design Compiler or FPGA Compiler, but before submitting them to MAX+PLUS® II software for compilation.

Altera provides an encrypted Full Timing Structural Model (FTSM) and a Full Timing Gate-Level Simulation model (FTGS) for the VHDL simulation libraries listed in Table 1.

Table 1. VHDL Functional Simulation Libraries

Device Family	Functional Simulation Libraries	Device Family	Functional Simulation Libraries
FLEX® 10K	**flex10k_FTSM.vhd.E** **flex10k_fpga_FTSM.vhd.E** **flex10k_FTGS.vhd.E** **flex10k_fpga_FTGS.vhd.E** **flex10k_components.vhd** **flex10k_fpga_components.vhd**	MAX® 9000	**max9000_FTSM.vhd.E** **max9000_fpga_FTSM.vhd.E** **max9000_FTGS.vhd.E** **max9000_fpga_FTGS.vhd.E** **max9000_components.vhd** **max9000_fpga_components.vhd**
FLEX 8000	**flex8000_FTSM.vhd.E** **flex8000_fpga_FTSM.vhd.E** **flex8000_FTGS.vhd.E** **flex8000_fpga_FTGS.vhd.E** **flex8000_components.vhd** **flex8000_fpga_components.vhd**	MAX 7000	**max7000_FTSM.vhd.E** **max7000_fpga_FTSM.vhd.E** **max7000_FTGS.vhd.E** **max7000_fpga_FTGS.vhd.E** **max7000_components.vhd** **max7000_fpga_components.vhd**
FLEX 6000	**flex6000_FTSM.vhd.E** **flex6000_fpga_FTSM.vhd.E** **flex6000_FTGS.vhd.E** **flex6000_fpga_FTGS.vhd.E** **flex6000_components.vhd** **flex6000_fpga_components.vhd**	MAX 5000 & Classic®	**max5000_FTSM.vhd.E** **max5000_fpga_FTSM.vhd.E** **max5000_FTGS.vhd.E** **max5000_fpga_FTGS.vhd.E** **max5000_components.vhd** **max5000_fpga_components.vhd**

Post-Routing Timing Simulation Libraries

The **/usr/maxplus2/synopsys/library/alt_post/sim/src** directory contains the Altera® -provided library files for performing timing simulation of designs that have been compiled with the MAX+PLUS II software. The VITAL 95-compliant post-simulation source files included in this directory are **alt_vtl.vhd** and **alt_vtl.cmp**. See Performing a Timing Simulation with VSS Software for more information.

Related Topics:

- Go to the following topics, which are available on the web, for additional information:
  - FLEX Devices

- MAX Devices
- Classic Device Family

Altera Post-Synthesis Libraries

The **/usr/maxplus2/synopsys/library/alt_post/syn/lib** directory contains the post-synthesis library for technology mapping and timing back-annotation. The Altera® -provided **alt_vtl.db** file in this library contains over three dozen MAX+PLUS® II -generated logic functions.

MAX+PLUS II/Synopsys Interface File Organization

Table 1 shows the MAX+PLUS® II /Synopsys interface subdirectories that are created in the MAX+PLUS II system directory (by default, the /usr/maxplus2 directory) during the MAX+PLUS II software installation. For information on the other directories that are created during the MAX+PLUS II software installation, see "MAX+PLUS II File Organization" in MAX+PLUS II Installation in the MAX+PLUS II Getting Started manual.

☞ You must add the **/usr/maxplus2/bin** directory to the `PATH` environment variable in your **.cshrc** file in order to run the MAX+PLUS II software.

Table 1. MAX+PLUS II Directory Organization

Directory	Description
./synopsys/bin	Contains script programs to convert Synopsys timing constraints into MAX+PLUS II Assignment & Configuration File (**.acf**) format, and to analyze VHDL System Simulator simulation models.
./synopsys/config	Contains sample **.synopsys_dc.setup** and **.synopsys_vss.setup** files.
./synopsys/examples	Contains sample files, including those discussed in these ACCESS Key Guidelines.
./synopsys/library/alt_pre/<*device family***>/src**	Contains VHDL simulation libraries for functional simulation of VHDL projects.
./synopsys/library/alt_pre/verilog/src	Contains the Verilog HDL functional simulation library for Verilog HDL projects.
./synopsys/library/alt_pre/vital/src	Contains the VITAL 95 simulation library. You use this library when you perform functional simulation of the design before compiling it with the MAX+PLUS II software.
./synopsys/library/alt_syn//<*device family***>/lib**	Contains interface files for the MAX+PLUS II/Synopsys interface. Technology libraries in this directory allow the Design Compiler and FPGA Compiler to map designs to Altera® device architectures.
./synopsys/library/alt_mf/src	Contains behavioral VHDL models of some Altera macrofunctions, along with their component declarations. The `a_81mux`, `a_8count`, `a_8fadd`, and `a_8mcomp` macrofunctions are currently supported. Libraries in this directory allow you to

	instantiate, synthesize, and simulate these macrofunctions.
./synopsys/library/alt_post/syn/lib	Contains the post-synthesis library for technology mapping.
./synopsys/library/alt_post/sim/src	Contains the VHDL source files for the VITAL 95-compliant library. You use this library when you perform simulation of the design after compiling it with the MAX+PLUS II software.

Related Topics:

- Go to the following topics, which are available on the web, for additional information:
  - ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
  - This manual is also available in 4 parts:
    - Preface & Section 1: MAX+PLUS II Installation
    - Section 2: MAX+PLUS II - A Perspective
    - Section 3: MAX+PLUS II Tutorial
    - Appendices, Glossary & Index

MAX+PLUS II Project File Structure

In MAX+PLUS® II, a project name is the name of a top-level design file, without the filename extension. This design file can be an EDIF, Verilog HDL, or VHDL netlist file; an AHDL TDF; or any other MAX+PLUS II-supported design file. The EDIF netlist file must be created by Synopsys and imported into MAX+PLUS II as an EDIF Input File.

MAX+PLUS II stores the connectivity data on the links between design files in a hierarchical project in a Hierarchy Interconnect File (**.hif**), but refers to the entire project only by its project name. The MAX+PLUS II Compiler uses the HIF to build a single, fully flattened project database that integrates all the design files in a project hierarchy.

Design Flow for All Synopsys Tools

Figure 1 shows the design flow between Synopsys and MAX+PLUS ® II software. Design Entry Flow, Project Compilation Flow, Project Simulation Flow, and Device Programming Flow show detailed diagrams of each stage of the design flow. For information on how to use the Synopsys Design Compiler or FPGA Compiler from within the MAX+PLUS II software, see Running Synopsys Compilers from the MAX+PLUS II Software.

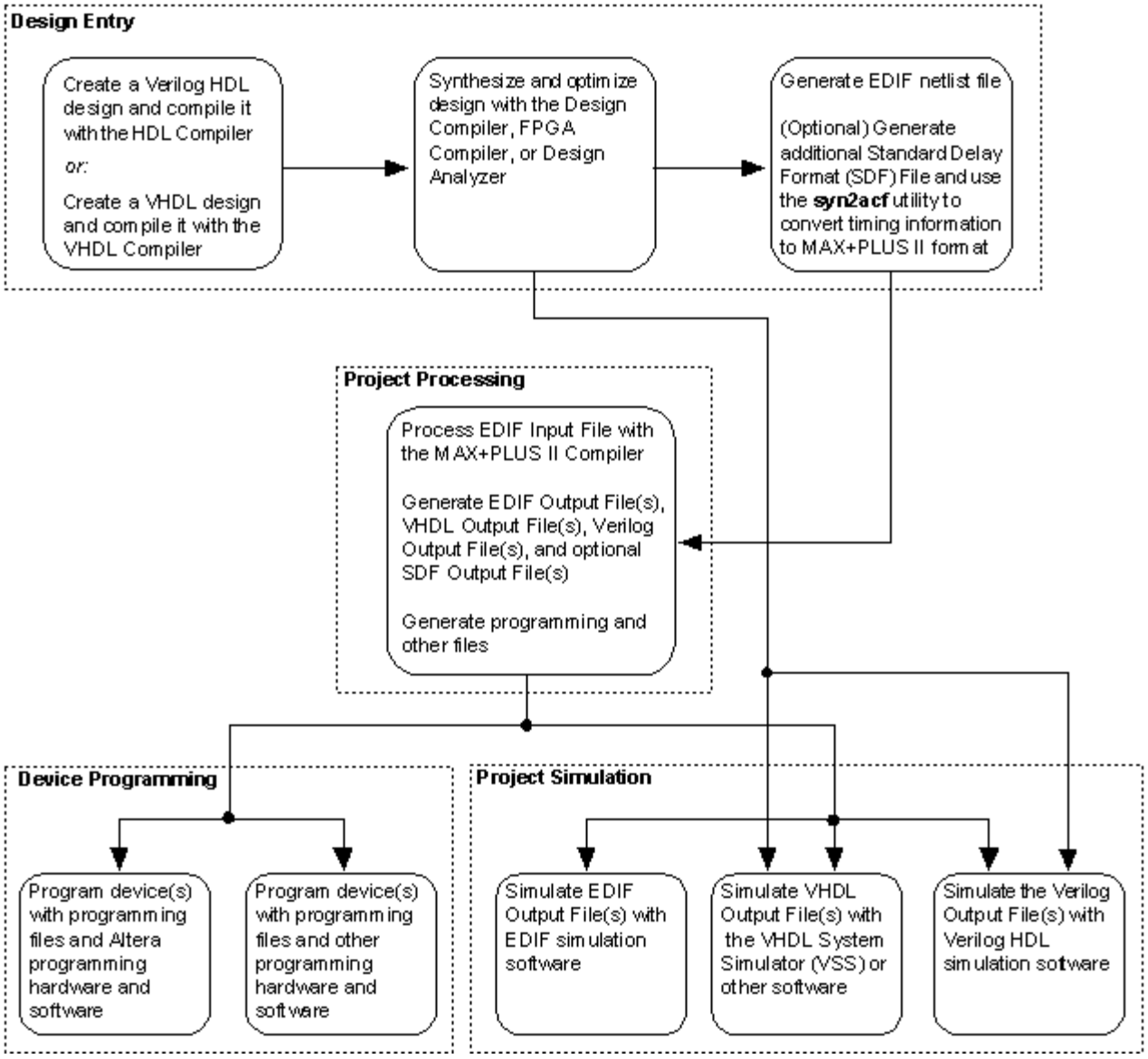


Figure 1. Design Flow between Synopsys & MAX+PLUS II Software

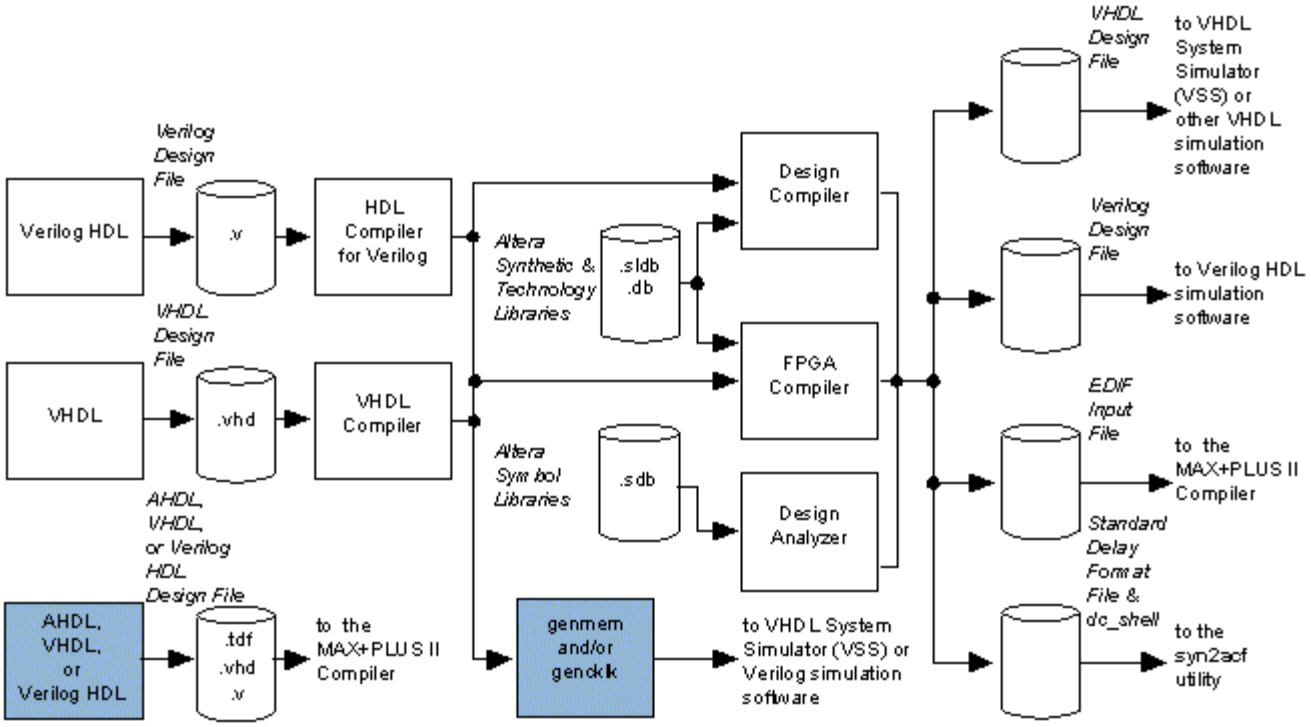
---

Synopsys Design Entry Flow

Figure 1 below shows the design entry flow for the MAX+PLUS® II /Synopsys interface.

Figure 1. MAX+PLUS II/Synopsys Design Entry Flow

Altera-provided items are shown in blue.



Creating VHDL Designs for Use with MAX+PLUS II Software

You can create VHDL design files with the MAX+PLUS® II Text Editor or another standard text editor and save them in the appropriate directory for your project. The MAX+PLUS II Text Editor offers the following advantages:

- VHDL templates are available with the **VHDL Templates** command (Templates menu). These templates are also available in the ASCII **vhdl.tmp** file, which is located in the **/usr/maxplus2** directory.
- If you use the MAX+PLUS II Text Editor to create your VHDL design, you can use the **Syntax Coloring** command (Options menu). The Syntax Coloring feature displays keywords and other elements of text in text files in different colors to distinguish them from other forms of syntax.

Once you have created a VHDL design, you can use the Design Compiler or FPGA Compiler to synthesize and optimize it, and then generate an EDIF netlist file that can be processed with the MAX+PLUS II software.

To create a VHDL design that can be synthesized and optimized with the Design Compiler or FPGA Compiler, follow these steps:

1. Instantiate logic functions with a Component Instantiation, and include a Component Declaration for each function. Altera provides simulation models for the following types of logic functions:
    - Primitives in the Design Compiler & FPGA Compiler Technology Libraries. Go to Primitive & Old-Style Macrofunction Instantiation Example for VHDL for an example.
    - Architecture Control Logic functions in the **alt_mf** library, which includes the `a_8count`, `a_8mcomp`, `a_8fadd`, and `a_81mux` functions. See MAX+PLUS II Architecture Control Logic Function Instantiation Example for VHDL for an example.
    - The DesignWare up/down counter function (`DW03_updn_ctr`). Go to DesignWare Up/Down Counter Function Instantiation Example for VHDL for an example.
    - RAM and ROM functions generated with the **genmem** utility. Go to Instantiating RAM & ROM Functions in VHDL for instructions.
    - The `clklock` megafunction, which is supported for selected FLEX 10K devices. This function is generated with the **genclk** utility. Go to Instantiating the clklock Megafunction in VHDL or Verilog

HDL for instructions.

- MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.

  You can also instantiate any other Altera macrofunction or non-parameterized megafunction, i.e., functions not listed above, for which no simulation models or technology library support is available. These functions are treated as "black boxes" during processing with the Design Compiler or FPGA Compiler. See Primitive & Old-Style Macrofunction Instantiation Example for VHDL for an example.

☞ For information on MAX+PLUS II primitives, megafunctions, and macrofunctions, choose **Primitives**, **Megafunctions/LPM**, or **Old-Style Macrofunctions** from the MAX+PLUS II Help menu. When searching for information on the **alt_mf** library functions, drop the initial "`a_`" from the function name.

2. (Optional) If you instantiate a "black box" logic function for which no simulation/techology library support is available, create a hollow-body design description in order to prevent the Design Compiler or FPGA Compiler from issuing a warning message. See Primitive & Old-Style Macrofunction Instantiation Example for VHDL for an example.

   ☞ If you instantiate a "black box" logic function, you must create a Library Mapping File (**.lmf**) to map the function to an equivalent MAX+PLUS II function before you compile the project with the MAX+PLUS II software. See Primitive & Old-Style Macrofunction Instantiation Example for VHDL for an example.

3. Once you have created a VHDL design, you can analyze it, synthesize it, (optional) perform a functional simulation, and generate an EDIF netlist file that can be imported into the MAX+PLUS II software. Go to the following topics for instructions:

   - Synthesizing & Optimizing VHDL & Verilog HDL Projects with Synopsys Software
   - Performing a Pre-Routing or Function Simulation with VSS Software

Installing the Altera-provided MAX+PLUS II/Synopsys Logic interface on your computer automatically creates the following VHDL sample files:

- **/usr/maxplus2/examples/mentor/examples/ministate.vhd**
- **/usr/maxplus2/examples/mentor/examples/count8.vhd**
- **/usr/maxplus2/examples/mentor/examples/tstrom.vhd**

Related Topics:

- Go to Compiling Projects with MAX+PLUS II Software in these MAX+PLUS II ACCESS^{SM} Key topics for related information.

Primitive & Old-Style Macrofunction Instantiation Example for VHDL

You can instantiate the MAX+PLUS® II primitives listed in Design Compiler & FPGA Compiler Technology Libraries in VHDL designs. These primitives can be used to control synthesis in the MAX+PLUS II software. You can also instantiate MAX+PLUS II megafunctions and old-style macrofunctions.

Go to the following topics for information and examples of how to instantiate functions that are not considered to be hollow bodies, including functions in the **alt_mf** library, RAM and ROM, and the `clklock` megafunction:

- Architecture Control Macrofunction Instantiation Example for VHDL
- Instantiating RAM & ROM Functions in VHDL
- Instantiating the clklock Megafunction in VHDL or Verilog HDL

Unlike other logic functions, MAX+PLUS II primitives do not need to be defined with Component Declarations unless you wish to simulate the design with the VHDL System Simulator (VSS) software. Any references to these primitives are resolved by the Synopsys compilers. All buffer primitives except the `ATRIBUF` and `TRIBUF` primitives also have a "don't touch" attribute already assigned to them, which prevents the Synopsys compilers from optimizing them. The Synopsys compilers also automatically treat mega- and macrofunctions that do not have corresponding synthesis library models as "black boxes."

Figure 1 shows a 4-bit full adder with registered output that also instantiates an `AGLOBAL` or `GLOBAL` primitive. This figure also illustrates the use of global Clock and global Reset pins in the MAX 7000 architecture. The design uses an old-style `7483` macrofunction, which is represented as a hollow body named `fa4`.

Figure 1. 4-Bit Adder Design with Registered Output (adder.vhd)

```
LIBRARY ieee;
   USE ieee.std_logic_1164.ALL;



ENTITY adder IS
   PORT (a, b     : IN  STD_LOGIC_VECTOR(4 DOWNTO 1);
         clk, rst : IN  STD_LOGIC;



cout     : OUT STD_LOGIC;
         regsum   : OUT STD_LOGIC_VECTOR(4 DOWNTO 1));
   END adder;



ARCHITECTURE MAX7000 OF adder IS



SIGNAL sum            : STD_LOGIC_VECTOR(4 DOWNTO 1);
SIGNAL ci, gclk, grst : STD_LOGIC;



-- Component Declaration for GLOBAL primitive
-- For FLEX devices, global, a_in, and a_out should be replaced with
-- aglobal, in1, and Y, respectively
COMPONENT global
   PORT (a_in      : IN  STD_LOGIC;
         a_out     : OUT STD_LOGIC);
END COMPONENT;



-- Component Declaration for fa4 macrofunction
COMPONENT fa4
   PORT (c0,a1,b1,a2,b2,a3,b3,a4,b4 : IN  STD_LOGIC;
         s1,s2,s3,s4,c4             : OUT STD_LOGIC);
END COMPONENT;



BEGIN
   ci <= '0';
```

```
-- FA4 Component Instantiation
   u0: fa4



PORT MAP (ci,a(1),b(1),a(2),b(2),a(3),b(3),a(4),b(4),
             sum(1),sum(2),sum(3),sum(4),cout);



-- GLOBAL Component Instantiation for Clock
-- For FLEX devices, global should be replaced with aglobal
   u1: global
   PORT MAP (clk, gclk);



-- GLOBAL Component Instantiation for Reset
-- For FLEX devices, global should be replaced with aglobal
   u2: global
   PORT MAP (rst, grst);



-- CLOCK process to create registered output
   clocked: PROCESS(gclk,grst)



BEGIN
      IF grst = '0' THEN
      regsum <= "0000"



ELSIF gclk'EVENT AND gclk = '1' THEN
           regsum <= sum;
      END IF;



END PROCESS clocked;

END MAX7000;
```

Before you can analyze the 4-bit adder design, you must first analyze the `fa4` description in Figure 1 with the Synopsys VHDL Compiler software. You can ignore the warning that is issued for any unknown function, including the `fa4` function in this example. If you wish, you can avoid receiving such warning messages by creating a hollow-body description of the function.

A hollow-body VHDL description combines an Entity Declaration with an empty or null Architecture Body. An empty Architecture Body contains the `ARCHITECTURE IS` clause, followed by the `BEGIN` and `END` keywords and a semicolon (;). It does not include any information about the design's function or operation. Figure 2 shows the hollow-body description for the `fa4` function.

Figure 2. Hollow-Body Description of a 4-Bit Full Adder (7483)

```
LIBRARY ieee;
USE     ieee.std_logic_1164.ALL;



-- fa4 maps to 7483. The interface names do not have to match.

ENTITY fa4 IS
```

```
PORT (c0,a1,b1,a2,b2,a3,b3,a4,b4 : IN STD_LOGIC;
      s1,s2,s3,s4,c4             : OUT STD_LOGIC);



END fa4;

ARCHITECTURE map7483 OF fa4 IS



BEGIN

-- This architecture body is left blank, and will map to the
-- 7483 macrofunction in MAX+PLUS II.

END;
```

When you analyze the hollow-body design description with the Synopsys VHDL Compiler software, it produces a hollow-body component that contains a single level of hierarchy with input and output pins, but does not contain any underlying logic.

You can save the synthesized design as an EDIF netlist file (**.edf**) and compile it with the MAX+PLUS II software. After the VHDL Compiler software has successfully processed the design, it generates the schematic shown in Figure 3, which you can view with the Design Analyzer software.

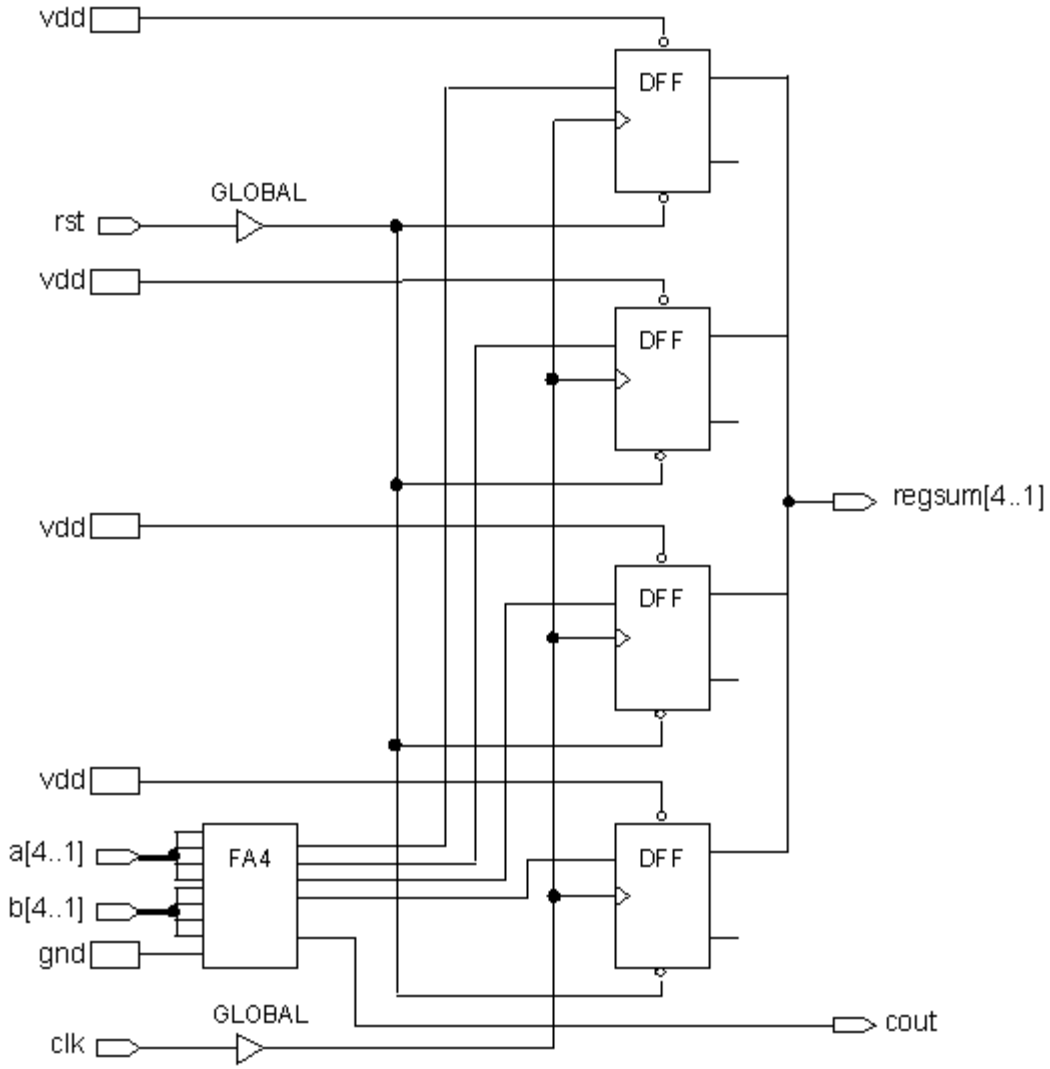


Figure 3. Synthesized Design Generated by the Design Compiler

However, before you compile the EDIF netlist file with the MAX+PLUS II software, you must create the **adder.lmf** file, shown in Figure 3, to map the `fa4` function to the equivalent MAX+PLUS II function (`7483`). You must then specify the LMF as *LMF #2* in the expanded **EDIF Netlist Reader Settings** dialog box (Interfaces menu) (*LMF #1* is **altsyn.lmf**). For more information about creating LMFs, refer to "Library Mapping Files (**.lmf**)" and "Library Mapping File Format" in MAX+PLUS II Help.

Figure 3. Library Mapping File Excerpt for fa4

```
BEGIN
FUNCTION 7483  (c0, a1, b1, a2, b2, a3, b3, a4, b4,)
RETURNS        (s1, s2, s3, s4, c4)



FUNCTION "fa4" ("c0", "a1", "b1", "a2", "b2", "a3",
               "b3","a4", "b4")
RETURNS        ("s1", "s2", "s3", "s4", "c4")
END
```

When you compile the design with the MAX+PLUS II software, you can disregard the warning "`EDIF cell <name> already has LMF mapping so CONTENTS construct has been ignored`". To verify the global Clock and global Reset usage, as well as the number of logic cells used, see the **adder.rpt** Report File generated by the MAX+PLUS II Compiler.

MAX+PLUS II Architecture Control Logic Function Instantiation Example for VHDL

You can instantiate Altera® -provided logic functions from the **alt_mf** library, which includes the `a_8fadd`, `a_8mcomp`, `a_8count`, and `a_81mux` functions, in VHDL designs. Altera provides behavioral descriptions of these functions that support pre-synthesis/pre-route simulation of your top-level design with the VHDL System Simulator (VSS).

When you instantiate one of these functions, you can either include a Component Declaration for the function, or use the Altera-provided shell script **analyze_vss** to create a design library called **altera** so that you can reference the functions through the VHDL Library and Use Clauses. The Library and Use Clauses direct the Design Compiler or FPGA Compiler to incorporate the library files when it compiles your top-level design file. The **analyze_vss** shell script creates the **altera** design library when it analyzes the VSS simulation models in the **/usr/maxplus2/synopsys/library/alt_mf/lib** directory. See Setting up VSS Configuration Files for more information on using the **analyze_vss** shell script.

Figure 1 shows an example of an 8-bit counter that is instantiated using the `a_8count` function.

Figure 1. Sample VHDL File with Logic Function Instantiation

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;



LIBRARY altera;
USE altera.maxplus2.ALL;



ENTITY counter IS
PORT (clock,ena,load,dnup,set,clear : IN STD_LOGIC;
   i    : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
```

```
   q    : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);


cout : OUT STD_LOGIC);
END counter;



ARCHITECTURE structure OF counter IS



BEGIN
   u1   : a_8count



PORT MAP (a=>i(0), b=>i(1), c=>i(2), d=>i(3), e=>i(4),
            f=>i(5), g=>i(6), h=>i(7), ldn=>load, gn=>ena,
            dnup=>dnup, setn=>set, clrn=>clear, clk=>clock,



qa=>q(0), qb=>q(1), qc=>q(2), qd=>q(3), qe=>q(4),
            qf=>q(5), qg=>q(6), qh=>q(7), cout=>cout);



END structure;



CONFIGURATION conf OF counter IS
   FOR structure
   END FOR;
END conf;
```

DesignWare Up/Down Counter Function Instantiation Example for VHDL

The Altera DesignWare Libraries for FLEX devices allow you to instantiate the `DW03_updn_ctr` function, which is the same as the Synopsys `DW03` up/down counter. This function allows you to use the same VHDL code regardless of which FLEX® device is targeted.

Figure 1 shows a VHDL file excerpt with DW03_updn_ctr instantiation.

Figure 1. VHDL File Excerpt with Up/Down Counter Instantiation

```
LIBRARY ieee,DW03;
USE ieee.std_logic_1164.all;
USE DW03.DW03_components.all;



ENTITY updn_4 IS
   PORT (D : IN STD_LOGIC_VECTOR(4-1 DOWNTO 0);
      UP_DN, LD, CE, CLK, RST: IN STD_LOGIC;
      TERCNT : OUT STD_LOGIC;
      Q      : OUT STD_LOGIC_VECTOR(4-1 DOWNTO 0));
END updn_4;



ARCHITECTURE structure OF updn_4 IS
```

```
BEGIN
   u0: DW03_updn_ctr
   GENERIC MAP(width => 4)
   PORT MAP (data => d, clk => clk, reset => rst, up_dn => up_dn,
             load => ld, tercnt => tercnt, cen => ce, count => q);
END structure;
```

Related Topics:

- Go to Setting Up the DesignWare Interface in these MAX+PLUS II ACCESS^{SM} Key topics for related information.
- Go to the following topics, which are available on the web, for additional information:
  - FLEX 6000 Device Family
  - FLEX 8000 Device Family
  - FLEX 10K Device Family

Instantiating RAM & ROM Functions in VHDL

The MAX+PLUS® II /Synopsys interface offers full support for the memory capabilities of the FLEX® 10K device family, including synchronous and asynchronous RAM and ROM, cycle-shared dual port RAM, dual-port RAM, single-Clock FIFO, and dual-clock FIFO functions. You can use the Altera® -provided **genmem** utility to generate functional simulation models and timing models for these functions. Type `genmem` ↵ at the UNIX prompt to display information on how to use this utility, as well as a list of the functions you can generate.

To instantiate a RAM or ROM function in VHDL, follow these steps:

1. Use the **genmem** utility to generate a memory model by typing the following command at the UNIX prompt:

   `genmem` <*memory type*> <*memory size*> `-vhdl` ↵

   For example: `genmem asynrom 256x15 -vhdl` ↵

2. Create a VHDL design that incorporates the text from the **genmem**-generated Component Declaration, <*memory name*>**.cmp**, and instantiate the <*memory name*> function.

   Figure 1 shows a VHDL design that instantiates **asyn_rom_256x15.vhd**, a 256 x 15 ROM function.

   Figure 1. VHDL Design File with ROM Instantiation (tstrom.vhd)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;



ENTITY tstrom IS
        PORT (
          addr    : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
          memenab : IN STD_LOGIC;
          q       : OUT STD_LOGIC_VECTOR (14 DOWNTO 0));
END tstrom;



ARCHITECTURE behavior OF tstrom IS

COMPONENT asyn_rom_256x15
-- pragma translate_off
     GENERIC (LPM_FILE : string);
```

```
-- pragma translate_on
     PORT (Address : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
           MemEnab : IN STD_LOGIC;
           Q       : OUT STD_LOGIC_VECTOR(14 DOWNTO 0)
     );
END COMPONENT;



BEGIN

   u1: asyn_rom_256x15
-- pragma translate_off
        GENERIC MAP (LPM_FILE => "u1.hex")
-- pragma translate_on
   PORT MAP (Address => addr, MemEnab => memenab, Q =>q);
END behavior;
```

3. (Optional for RAM functions) Specify an initial memory content file:

   - For ROM functions, you must specify the filename of an initial memory content file in the Intel hexadecimal format (**.hex**) or the Altera® Memory Initialization File (**.mif**) format in the Generic Map Clause, with the `LPM_FILE` parameter. See Figure 1. The filename must be the same as the instance name; e.g., the `u1` instance name must be unique throughout the whole project, and must contain only valid VHDL name characters. The initialization file must reside in the directory containing the project's design files.

   - For RAM functions, specifying a memory initialization file is optional. If you want to use it, you must specify it in the Generic Map Clause as described above. If you do not use an initialization file, you should not declare or use the Generic Clause.

     1. The MIF format is supported only for specifying initial memory content when compiling designs within MAX+PLUS II software. You cannot use a MIF to perform simulation with Synopsys tools prior to MAX+PLUS II compilation.

     2. If you use an Intel hexadecimal format file and wish to simulate the design with the VHDL System Simulator (VSS) after MAX+PLUS II compilation, you should use the Synopsys **intelhex** utility to translate the Intel hexadecimal fomat file into a VSS-compatible Synopsys memory file. Refer to the Synopsys ***VHDL System Simulator Software Tool*** manual for details about using the **intelhex** utility.

4. In the VHDL design file, add the compiler directive `-- pragma translate_off` ↵ before the Generic Clause and Generic Map Clause, and add `-- pragma translate_on` ↵ after the Generic Clause and Generic Map Clause. These directives tell the VHDL Compiler software when to stop and start synthesizing. For example, in Figure 1, the `--pragma translate_off` directive instructs the VHDL Compiler software to skip syntax checking until the `--pragma translate_on` directive is read.

5. Because the VHDL Compiler software does not support the data type `string` for the Generic Clause, you must also enter the following command before you read the design:

   ```
   hdlin_translate_off_skip_text=true ↵
   ```

6. The timing model (**.lib**) generated by the **genmem** utility contains pin-to-pin delay information that can be used by the Synopsys Design Compiler and FPGA Compiler software. You must add this timing model to the existing library so that the compiler can access the timing information. Type the following commands at the **dc_shell** prompt:

   `read -f db flex10k[<`*speed grade*`>].db` ↵

```
update_lib flex10k[<speed grade>] <RAM/ROM function name>.lib ↵
```

7. (Optional) Enter the following command to update your **flex10k[**<*speed grade*>**].db** file with the RAM/ROM timing information:

   ```
   write_lib flex10k[<speed grade>] -o flex10k.db ↵
   ```

8. When you generate the EDIF netlist file from the design, include the bus structure from the RAM or ROM function(s). Go to Setting Up Synopsys Configuration Files for more information.

9. Continue with the steps necessary to complete your VHDL design, as described in Creating VHDL Designs for Use with MAX+PLUS II Software.

Related Topics:

- Go to FLEX 10K Device Family, which is available on the web, for additional information.

Instantiating the clklock Megafunction in VHDL or Verilog HDL

MAX+PLUS® II interfaces with other EDA tools support the `clklock` phase-locked loop megafunction, which can be used with some FLEX® 10K devices, with the **gencklk** utility. Type `gencklk -h ↵` at the UNIX prompt to display information on how to use this utility. The **gencklk** utility generates VHDL or Verilog HDL functional simulation models and a VHDL Component Declaration template file (**.cmp**).

The **gencklk** utility allows parameters for the `clklock` function to be passed from the VHDL or Verilog HDL file to EDIF netlist format. The **gencklk** utility embeds the parameter values in the `clklock` function name; therefore, the values do not need to be declared explicitly.

To instantiate the `clklock` megafunction in VHDL or Verilog HDL, go through the following steps:

1. Type the following command at the UNIX prompt to generate the `clklock_`*x*`_`*y* file, where *x* is the ClockBoost value and *y* is the input frequency in MHz:

   ✔ Type `gencklk` <*ClockBoost*> <*input frequency*> `-vhdl ↵` for VHDL designs.

   or:

   ✔ Type `gencklk` <*ClockBoost*> <*input frequency*> `-verilog ↵` for Verilog HDL designs.

   Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu for more information on the `clklock` megafunction.

2. Create a design file that instantiates the `clklock_`*x*`_`*y* function. The **gencklk** utility automatically generates a VHDL Component Declaration template in the **clklock_*x*_*y*.cmp** file that you can incorporate into a VHDL design file.

Figures 1 and 2 show a `clklock` function with <*ClockBoost*> = 2 and <*input frequency*> = 40 MHz instantiated in VHDL and Verilog HDL design files, respectively.

Figure 1. VHDL Design File with clklock Instantiation (count8.vhd)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY altera;
USE altera.maxplus2.all;     --  Include Altera Component Declarations
```

```
ENTITY count8 IS
   PORT (a    : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
         ldn  : IN STD_LOGIC;
         gn   : IN STD_LOGIC;



dnup : IN STD_LOGIC;
         setn : IN STD_LOGIC;
         clrn : IN STD_LOGIC;
         clk  : IN STD_LOGIC;



co   : OUT STD_LOGIC;
         q    : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END count8;



ARCHITECTURE structure OF count8 IS
   signal clk2x : STD_LOGIC;



COMPONENT clklock_2_40
   PORT (
      INCLK : IN STD_LOGIC;
      OUTCLK : OUT STD_LOGIC
   );
END COMPONENT;



BEGIN
   u1: clklock_2_40
   PORT MAP (inclk=>clk, outclk=>clk2x);



u2: a_8count
   PORT MAP (a=>a(0), b=>a(1), c=>a(2), d=>a(3),
             e=>a(4), f=>a(5), g=>a(6), h=>a(7),
             clk=>clk2x,
             ldn=>ldn,
             gn=>gn,



dnup=>dnup,
             setn=>setn,
             clrn=>clrn,



qa=>q(0), qb=>q(1), qc=>q(2), qd=>q(3),
             qe=>q(4), qf=>q(5), qg=>q(6), qh=>q(7),
             cout=>co);
 END structure;
```

Figure 2. Verilog HDL Design File with clklock Instantiation (count8.v)

```
`timescale 1ns / 10ps
module count8 (a, ldn, gn, dnup, setn, clrn, clk, co, q);
output          co;
output[7:0]     q;



input[7:0]      a;
```

```
input           ldn, gn,dnup, setn, clrn, clk;
wire            clk2x;



clklock_2_40 u1 (.inclk(clk), .outclk(clk2x) );
A_8COUNT u2 (.A(a[0]), .B(a[1]), .C(a[2]), .D(a[3]), .E(a[4]), .F(a[5]),



.G(a[6]), .H(a[7]), .LDN(ldn), .GN(gn), .DNUP(dnup),
             .SETN(setn), .CLRN(clrn), .CLK(clk2x), .QA(q[0]), .QB(q[1]),
             .QC(q[2]), .QD(q[3]), .QE(q[4]), .QF(q[5]), .QG(q[6]),
             .QH(q[7]), .COUT(co) );



endmodule
```

Related Topics:

- Go to FLEX 10K Device Family, which is available on the web, for additional information.

Creating Verilog HDL Designs for Use with MAX+PLUS II Software

You can create Verilog HDL design files with the MAX+PLUS® II Text Editor or another standard text editor and save them in the appropriate directory for your project. The MAX+PLUS II Text Editor offers the following advantages:

- Verilog HDL templates are available with the **Verilog HDL Templates** command (Templates menu). These templates are also available in the ASCII **verilog.tmp** file, which is located in the **/usr/maxplus2** directory.
- If you use the MAX+PLUS II Text Editor to create your Verilog HDL design, you can use the **Syntax Coloring** command (Options menu). The Syntax Coloring feature displays keywords and other elements of text in text files in different colors to distinguish them from other forms of syntax.

Once you have created a Verilog HDL design, you can use the Design Compiler or FPGA Compiler to synthesize and optimize it, and then generate an EDIF netlist file that can be processed with the MAX+PLUS II software.

To create a Verilog HDL design that can be synthesized and optimized with the Design Compiler or FPGA Compiler, follow these steps:

1. Instantiate logic functions with a Module Instantiation, and include a Module Declaration for each function. Altera provides simulation models for the following types of logic functions:
    - Primitives in the Design Compiler & FPGA Compiler Technology Libraries. Go to Primitive & Old-Style Macrofunction Instantiation Example for Verilog HDL for an example.
    - Architecture Control Logic functions in the **alt_mf** library, which includes the `a_8count`, `a_8mcomp`, `a_8fadd`, and `a_81mux` functions. See MAX+PLUS II Architecture Control Logic Function Instantiation Example for Verilog HDL for an example.
    - RAM and ROM functions generated with the **genmem** utility. Go to Instantiating RAM & ROM Functions in VHDL for instructions.
    - The `clklock` megafunction, which is supported for selected FLEX 10K devices. This function is generated with the **genclklk** utility. Go to Instantiating the clklock Megafunction in VHDL or Verilog HDL for instructions.
    - MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile,

and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.

You can also instantiate any other Altera macrofunction or non-parameterized megafunction, i.e., functions not listed above, for which no simulation models or technology library support is available. These functions are treated as "black boxes" during processing with the Design Compiler or FPGA Compiler. See Primitive & Old-Style Macrofunction Instantiation Example for Verilog HDL for an example.

For information on MAX+PLUS II primitives, megafunctions, and macrofunctions, choose **Primitives**, **Megafunctions/LPM**, or **Old-Style Macrofunctions** from the MAX+PLUS II Help menu. When searching for information on the **alt_mf** library functions, drop the initial "`a_`" from the function name.

2. (Optional) If you instantiate a "black box" logic function for which no simulation/techology library support is available, create a hollow-body design description in order to prevent the Design Compiler or FPGA Compiler from issuing a warning message. See Primitive & Old-Style Macrofunction Instantiation Example for Verilog HDL for an example.

   If you instantiate a "black box" logic function, you must create a Library Mapping File (**.lmf**) to map the function to an equivalent MAX+PLUS II function before you compile the project with the MAX+PLUS II software. See Primitive & Old-Style Macrofunction Instantiation Example for VHDL for an example.

3. Once you have created a VHDL design, you can analyze it, synthesize it, (optional) perform a functional simulation, and generate an EDIF netlist file that can be imported into the MAX+PLUS II software. Go to the following topics for instructions:

   - Synthesizing & Optimizing VHDL & Verilog HDL Projects with Synopsys Software
   - Performing a Pre-Routing or Function Simulation with VSS Software

Installing the Altera-provided MAX+PLUS II/Synopsys Logic interface on your computer automatically creates the following VHDL sample files:

- **/usr/maxplus2/examples/mentor/examples/ministate.v**
- **/usr/maxplus2/examples/mentor/examples/count8.v**
- **/usr/maxplus2/examples/mentor/examples/tstrom.v**

Related Topics:

- Go to Compiling Projects with MAX+PLUS II Software in these MAX+PLUS II ACCESS℠ Key topics for related information.

Primitive & Old-Style Macrofunction Instantiation Example for Verilog HDL

You can instantiate the MAX+PLUS® II primitives listed in Design Compiler & FPGA Compiler Technology Libraries in Verilog HDL designs. These primitives can be used to control synthesis in the MAX+PLUS II software. You can also instantiate MAX+PLUS II megafunctions and old-style macrofunctions.

Go to the following topics for information and examples of how to instantiate functions that are not considered to be hollow bodies, including functions in the **alt_mf** library, RAM and ROM, and the `clklock` megafunction:

- Architecture Control Macrofunction Instantiation Example for Verilog HDL

- Instantiating RAM & ROM Functions in Verilog HDL
- Instantiating the clklock Megafunction in VHDL or Verilog HDL

Unlike other logic functions, MAX+PLUS II primitives do not need to be defined with hollow-body functions unless you wish to simulate the design with the VHDL System Simulator (VSS) software. Any references to these primitives are resolved by the Synopsys compilers. All buffer primitives except the `ATRIBUF` and `TRIBUF` primitives also have a "don't touch" attribute already assigned to them, which prevents the Synopsys compilers from optimizing them. The Synopsys compilers also automatically treat mega- and macrofunctions that do not have corresponding synthesis library models as "black boxes."

Figure 1 shows a 4-bit full adder with registered output that also instantiates an AGLOBAL or GLOBAL primitive. This figure also illustrates the use of global Clock and global Reset pins in the MAX 7000 architecture. The design uses an old-style 7483 macrofunction, which is represented as a hollow body named fa4.

Figure 1. 4-Bit Adder Design with Registered Output (adder.v)

```
module adder (a, b, clk, rst, cout, regsum);

output       cout;
output[4:1]  regsum;
input[4:1]   a, b;
input        clk, rst;
wire[4:1]    sum;
reg[4:1]     regsum_int;
wire         grst, gclk;
wire         ci;
assign       ci = 0;



// module instantiation
fa4    u0  ( .c0(ci), .a1(a[1]), .b1(b[1]), .a2(a[2]),
             .b2(b[2]), .a3(a[3]), .b3(b[3]), .a4(a[4]),
             .b4(b[4]), .s1(sum[1]), .s2(sum[2]),
             .s3(sum[3]), .s4(sum[4]), .c4(cout));
// For FLEX devices, GLOBAL, A_IN, and A_OUT should be replaced
// with AGLOBAL, IN1, and Y, respectively
GLOBAL  u1 ( .A_IN(clk), .A_OUT(gclk));
GLOBAL  u2 ( .A_IN(rst), .A_OUT(grst));



always @(posedge gclk or negedge grst)
        if ( !grst )
                regsum_int = 4'b0;
        else regsum_int = sum;
assign regsum = regsum_int;
endmodule



// module declaration for fa4 module
module fa4 ( c0, a1, b1, a2, b2, a3, b3, a4, b4, s1, s2, s3, s4, c4);



output    s1, s2, s3, s4, c4;
   input     c0, a1, b1, a2, b2, a3, b3, a4, b4;
   endmodule
```

```
// module declaration for GLOBAL primitive
// For FLEX devices, GLOBAL, A_IN, and A_OUT should be replaced
// with AGLOBAL, IN1, and Y, respectively
module GLOBAL (A_OUT, A_IN);



input       A_IN;
   output      A_OUT;
   endmodule
```

You can analyze the 4-bit adder design with the Synopsys HDL Compiler for Verilog software. The hollow-body description of the `fa4` function is required. It contains port declarations and does not include any information about the design's function or operation. However, the hollow-body description can be in the design file, as shown in Figure 1, or in a separate file, as shown in Figure 2.

Figure 2. Hollow-Body Description of a 4-Bit Full Adder (7483)

```
module fa4 ( c0, a1, b1, a2, b2, a3, b3, a4, b4, s1, s2, s3, s4, c4);
   output    s1, s2, s3, s4, c4;
   input     c0, a1, b1, a2, b2, a3, b3, a4, b4;
   endmodule
```

If the hollow-body description is in a separate file, you must analyze it before analyzing the higher-level function with the HDL Compiler for Verilog to produce a hollow-body component. This component contains a single level of hierarchy with input and output pins, but does not contain any underlying logic.

You can save the synthesized design as an EDIF netlist file (**.edf**) and compile it with the MAX+PLUS II software. After the HDL Compiler for Verilog software has successfully processed the design, it generates the schematic shown in Figure 3, which you can view with the Design Analyzer software.

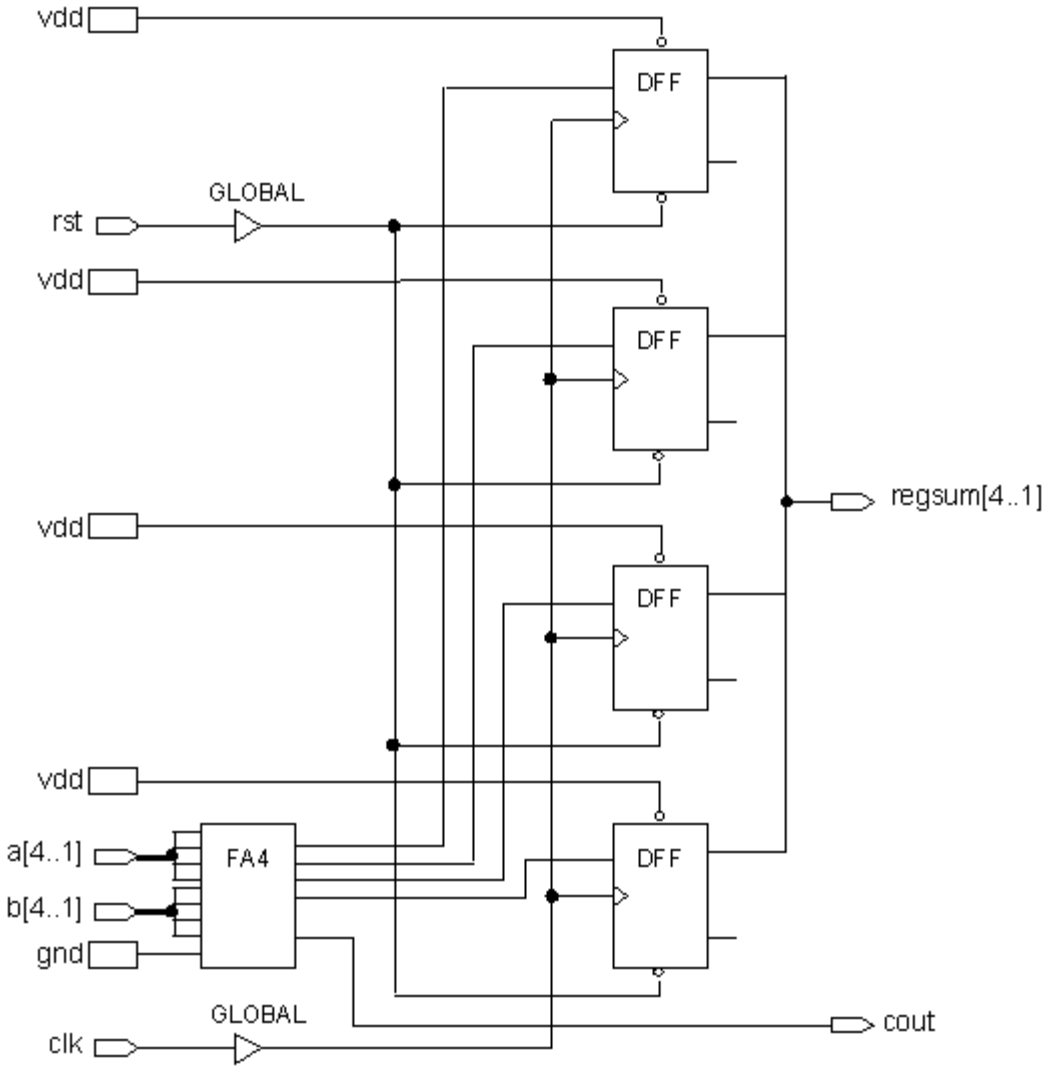


Figure 3. Synthesized Design Generated by the Design Compiler

However, before you compile the EDIF netlist file with the MAX+PLUS II software, you must create the **adder.lmf** file, shown in Figure 3, to map the `fa4` function to the equivalent MAX+PLUS II function (`7483`). You must then specify the LMF as *LMF #2* in the expanded **EDIF Netlist Reader Settings** dialog box (Interfaces menu) (*LMF #1* is **altsyn.lmf**). For more information about creating LMFs, refer to "Library Mapping Files (**.lmf**)" and "Library Mapping File Format" in MAX+PLUS II Help.

Figure 3. Library Mapping File Excerpt for fa4

```
BEGIN
FUNCTION 7483  (c0, a1, b1, a2, b2, a3, b3, a4, b4,)
RETURNS        (s1, s2, s3, s4, c4)



FUNCTION "fa4" ("c0", "a1", "b1", "a2", "b2", "a3",
               "b3","a4", "b4")
RETURNS        ("s1", "s2", "s3", "s4", "c4")
END
```

When you compile the design with the MAX+PLUS II software, you can disregard the warning "`EDIF cell <`*name*`> already has LMF mapping so CONTENTS construct has been ignored`". To verify the global Clock and global Reset usage, as well as the number of logic cells used, see the **adder.rpt** Report File generated by the MAX+PLUS II Compiler.

MAX+PLUS II Architecture Control Logic Function Instantiation Example for Verilog HDL

You can instantiate Altera® -provided logic functions from the **alt_mf** library, which includes the `a_8fadd`, `a_8mcomp`, `a_8count`, and `a_81mux` functions, in Verilog HDL designs. Altera provides behavioral Verilog HDL descriptions of these functions.

Figure 1 shows an example of an 8-bit counter that is instantiated using the `a_8count` function. Because Verilog HDL is case-sensitive, be sure to use uppercase letters for all of the macrofunction's module names and port names.

Figure 1. Sample Verilog HDL File with Logic Function Instantiation (counter.v)

```
module counter (clock, ena, load, dnup, set, clear, i, q, cout);
output          cout;
output[7:0]     q;
input[7:0]      i;
input           clock, ena, load, dnup, set, clear;
A_8COUNT u1    (.A(i[0]), .B(i[1]), .C(i[2]), .D(i[3]),
                .E(i[4]), .F(i[5]), .G(i[6]), .H(i[7]),
                .LDN(load), .GN(ena), .DNUP(dnup), .SETN(set),
                .CLRN(clear), .CLK(clock), .QA(q[0]), .QB(q[1]),
                .QC(q[2]), .QD(q[3]), .QE(q[4]), .QF(q[5]),
                .QG(q[6]), .QH(q[7]), .COUT(cout) );
endmodule
```

The sample file shown in Figure 1 can be synthesized with the Design Compiler or FPGA Compiler. You can also simulate it with the Cadence Verilog-XL Simulator by typing the following command at the **dc_shell** prompt:

```
verilog counter.v /usr/maxplus2/synopsys/library/alt_mf/src/mf.v ↵
```

Instantiating RAM & ROM Functions in Verilog HDL

The MAX+PLUS® II /Synopsys interface offers full support for the memory capabilities of the FLEX® 10K device family, including synchronous and asynchronous RAM and ROM, cycle-shared dual port RAM, dual-port RAM, single-Clock FIFO, and dual-clock FIFO functions. You can use the Altera® -provided **genmem** utility to generate functional simulation models and timing models for these functions. Type `genmem` ↵ at the UNIX prompt to display information on how to use this utility, as well as a list of the functions you can generate.

To instantiate a RAM or ROM function in Verilog HDL, follow these steps:

1. Use the **genmem** utility to generate a memory model by typing the following command at the UNIX prompt:

   `genmem` <*memory type*> <*memory size*> `-verilog` ↵

   For example: `genmem asynrom 256x15 -verilog` ↵

2. Create a Verilog HDL design that instantiates the <*memory name*> function.

   Figure 1 shows a Verilog HDL design that instantiates **asyn_rom_256x15.v**, a 256 x 15 ROM function.

   Figure 1. Verilog HDL File with ROM Instantiation (tstrom.v)

   ```
   module tstrom (addr, enab, q);
   parameter LPM_FILE = "u1.hex"
   input [7:0] addr;
   input enab;
   output [14:0] q;
   ```

```
asyn_rom_256x15
// synopsys translate_off
           #(LPM_FILE)



// synopsys translate_on
      u1   (.Address(addr), .Q(q), .MemEnab(enab));

endmodule
```

3. (Optional for RAM functions) Specify an initial memory content file:
   - For ROM functions, you must specify the filename of an initial memory content file in the Intel hexadecimal format (**.hex**) or the Altera® Memory Initialization File (**.mif**) format in the Parameter Statement, with the `LPM_FILE` parameter. See Figure 1. The filename must be the same as the instance name; e.g., the `u1` instance name must be unique throughout the whole project. The initialization file must reside in the directory containing the project's design files.
   - For RAM functions, specifying a memory initialization file is optional. If you want to use it, you must specify it in a Parameter Statement, as described above.

     1. The MIF format is supported only for specifying initial memory content when compiling designs within MAX+PLUS II software. You cannot use a MIF to perform simulation with Synopsys tools prior to MAX+PLUS II compilation.

     2. If you use an Intel hexadecimal format file and wish to simulate the design with the VHDL System Simulator (VSS) after MAX+PLUS II compilation, you should use the Synopsys **intelhex** utility to translate the Intel hexadecimal fomat file into a VSS-compatible Synopsys memory file. Refer to the Synopsys ***VHDL System Simulator Software Tool*** manual for details about using the **intelhex** utility.

4. In the Verilog HDL design, add `// synopsys translate_off` ↵ before the Parameter Statement, and add `// synopsys translate_on` ↵ after the Parameter Statement. These directives tell the HDL Compiler for Verilog when to stop and start synthesizing. See Figure 1.

5. The timing model (**.lib**) generated by the **genmem** utility contains pin-to-pin delay information that can be used by the Synopsys Design Compiler and FPGA Compiler software. You must add this timing model to the existing library so that the compiler can access the timing information. Type the following commands at the **dc_shell** prompt:

   `read -f db flex10k[`<*speed grade*>`].db` ↵
   `update_lib flex10k[`<*speed grade*>`]` <*RAM/ROM function name*>`.lib` ↵

6. (Optional) Include the following command to update your **flex10k[**<*speed grade*>**].db** file with the RAM/ROM timing information:

   `write_lib flex10k[`<*speed grade*>`] -o flex10k.db` ↵

7. When you generate the EDIF netlist file from the design, include the bus structure from the RAM or ROM function(s). Go to Setting Up Synopsys Configuration Files for more information.

8. Continue with the steps necessary to complete your Verilog HDL design, as described in Creating Verilog HDL Designs for Use with MAX+PLUS II Software.

Related Topics:

- Go to FLEX 10K Device Family, which is available on the web, for additional information.

Synthesizing & Optimizing VHDL & Verilog HDL Projects with Synopsys Software

The MAX+PLUS® II Compiler can process a VHDL or Verilog HDL file that has been synthesized by the Synopsys Design Compiler or FPGA Compiler software, saved as an EDIF 2 0 0 or 3 0 0 netlist file, and imported into the MAX+PLUS II software. The procedure below explains how to run Synopsys tools separately from MAX+PLUS II Software.

You can also run Synopsys tools from within the MAX+PLUS II software to automatically generate and import an EDIF file. Refer to Running Synopsys Compilers from MAX+PLUS II Software for more information. In addition, if your MAX+PLUS II development system includes VHDL or Verilog HDL synthesis support, the MAX+PLUS II Compiler can directly synthesize VHDL or Verilog HDL logic. For more information, go to MAX+PLUS II VHDL or Verilog HDL Help.

The following steps explain how to synthesize and optimize a VHDL or Verilog HDL design for use with MAX+PLUS II software:

1. Be sure to set up your design environment correctly. This step includes specifying the target device family for the design. See the following topics:
    - Setting Up the Synopsys/MAX+PLUS II Working Environment
    - Setting Up the Design Compiler and FPGA Compiler Configuration Files
    - Setting Up the DesignWare Interface
    - Setting Up the VSS Configuration Files
2. Create a VHDL file, <*design name*>**.vhd**, or a Verilog HDL design, <*design name*>**.v**, using the MAX+PLUS II Text Editor or another standard text editor and save it in a project directory under your login directory. See the following topics for instructions:
    - Creating VHDL Designs for Use with MAX+PLUS II Software.
    - Creating Verilog HDL Designs for Use with MAX+PLUS II Software.
3. Start the Design Compiler or FPGA Compiler software by typing either `dc_shell` ↵ or `fpga_shell` ↵ at the command line, respectively. To work within the graphical user interface, type `design_analyzer` ↵ for either tool.
4. Analyze and then compile the design with the Design Compiler, FPGA Compiler, or Design Analyzer software. The VHDL Compiler or HDL Compiler for Verilog software automatically translates the design into Synopsys database (**.db**) format. Specific steps are necessary for some types of projects before you process the design:
    1. If your FLEX 10K design includes RAM or ROM functions, follow these steps:
        1. (VHDL designs only) Because the VHDL Compiler software does not support the data type `string` for the Generic Clause, you must also enter the following command at the `dc_shell` prompt before you read the design:

            ```
            hdlin_translate_off_skip_text=true ↵
            ```

        2. The timing model (**.lib**) generated by the **genmem** utility contains pin-to-pin delay information that can be used by the Synopsys Design Compiler and FPGA Compiler software. You must add this timing model to the existing library so that the compiler can access the timing information. Type the following commands at the **dc_shell** prompt:

```
read -f db flex10k[<speed grade>].db ↵
update_lib flex10k[<speed grade>] <RAM/ROM function name>.lib ↵
```

3. (Optional) Enter the following command to update your **flex10k[***<speed grade>***].db** file with the RAM/ROM timing information:

   ```
   write_lib flex10k[<speed grade>] -o flex10k.db ↵
   ```

   See Instantiating RAM & ROM Functions in VHDL or Instantiating RAM & ROM functions in Verilog HDL for additional information.

2. If you wish to allow the FPGA Compiler to perform *N*-input look-up table (LUT) optimization for a FLEX 6000, FLEX 8000, or FLEX 10K design, enter the following command at the `dc_shell` prompt before compiling the design:

   ```
   edifout_write_properties_list = "lut function" ↵
   ```

   Go to Using FPGA Compiler *N*-Input LUT Optimization for FLEX 6000, FLEX 8000, or FLEX 10K Devices for more information.

3. If you wish to enter resource assignments, go to Entering Resource Assignments.

4. If you wish to direct the Design Compiler or FPGA Compiler to use sum-of-products logic in processing a MAX 7000 or MAX 9000 design, type the following commands at the `dc_shell` prompt before compiling the design:

   ```
   set_structure false ↵
   set_flatten -effort low ↵
   ```

   See MAX 7000 & MAX 9000 Synthesis Example for more information.

   For additional information on how the Design Compiler and FPGA Compiler synthesize and optimize a design, see the following topics:

   - ***Synopsys Design Compiler Reference Manual*** or ***Design Analyzer Reference Manual***
   - DesignWare FLEX 8000 Synthesis Example

5. (Optional) View the optimized project with the Design Analyzer. The Design Analyzer uses the **altera.sdb** library to display optimized projects generated by the Design Compiler or FPGA Compiler.

6. (Optional) To view Synopsys-generated timing information and generate a file detailing primitive usage, type the following commands:

   ```
   report_timing ↵
   report_reference > <filename> ↵
   ```

7. (Optional) To functionally verify the project prior to processing with the MAX+PLUS II software, save the design as a VHDL netlist file, and simulate it as described in Performing a Pre-Routing or Functional Simulation with VSS Software.

8. Save the optimized project as an EDIF netlist file with the extension **.edf**.

9. Process the *<design name>***.edf** file with the MAX+PLUS II Compiler, as described in Compiling Projects with the MAX+PLUS II Software.

Installing the Altera-provided MAX+PLUS II/Synopsys interface on your computer automatically creates the following sample VHDL and Verilog HDL files:

- **/usr/maxplus2/synopsys/examples/ministate.vhd**
- **/usr/maxplus2/synopsys/examples/ministate.v**

Related Topics:

- Go to the following MAX+PLUS II ACCESS^{SM} Key topics for related information:
  - Resynthesizing a Design Using the **alt_vtl** Library and a MAX+PLUS II SDF Output File
  - Programming Altera Devices

MAX 7000 & MAX 9000 Synthesis Example

The MAX® 7000 (including MAX 7000E, MAX 7000S, and MAX 7000A) and MAX 9000 device families have a sum-of-products architecture. To obtain optimum timing and area results, you can direct the Synopsys Design Compiler or FPGA Compiler software to synthesize your logic into a sum-of-products form. To assist the Synopsys compilers in meeting the timing and area constraints of your designs, the Altera® technology libraries provide models that approximate the timing of the MAX 7000 and MAX 9000 logic cells.

Figure 1 shows two timing models: the standard Altera MAX 7000 timing model and a Synopsys timing model that approximates the MAX 7000 model. The Synopsys model is built on the following three conditions and assumptions:

1. The combinatorial delay in logic cells has been equally divided between product terms and OR gates. Because the product-term delay equals the OR-gate delay, the Synopsys compilers treat them equally, producing a sum-of-products structure. On top of this structure, inverters are used where necessary.

2. A shared expander product term is always used to create combinatorial logic.

3. The Synopsys Design Compiler and FPGA Compiler software do not distinguish between array and global Clocks. Therefore, to estimate setup and hold timing most accurately, you must instantiate GLOBAL buffers to indicate a global clock in either your VHDL or Verilog HDL design.

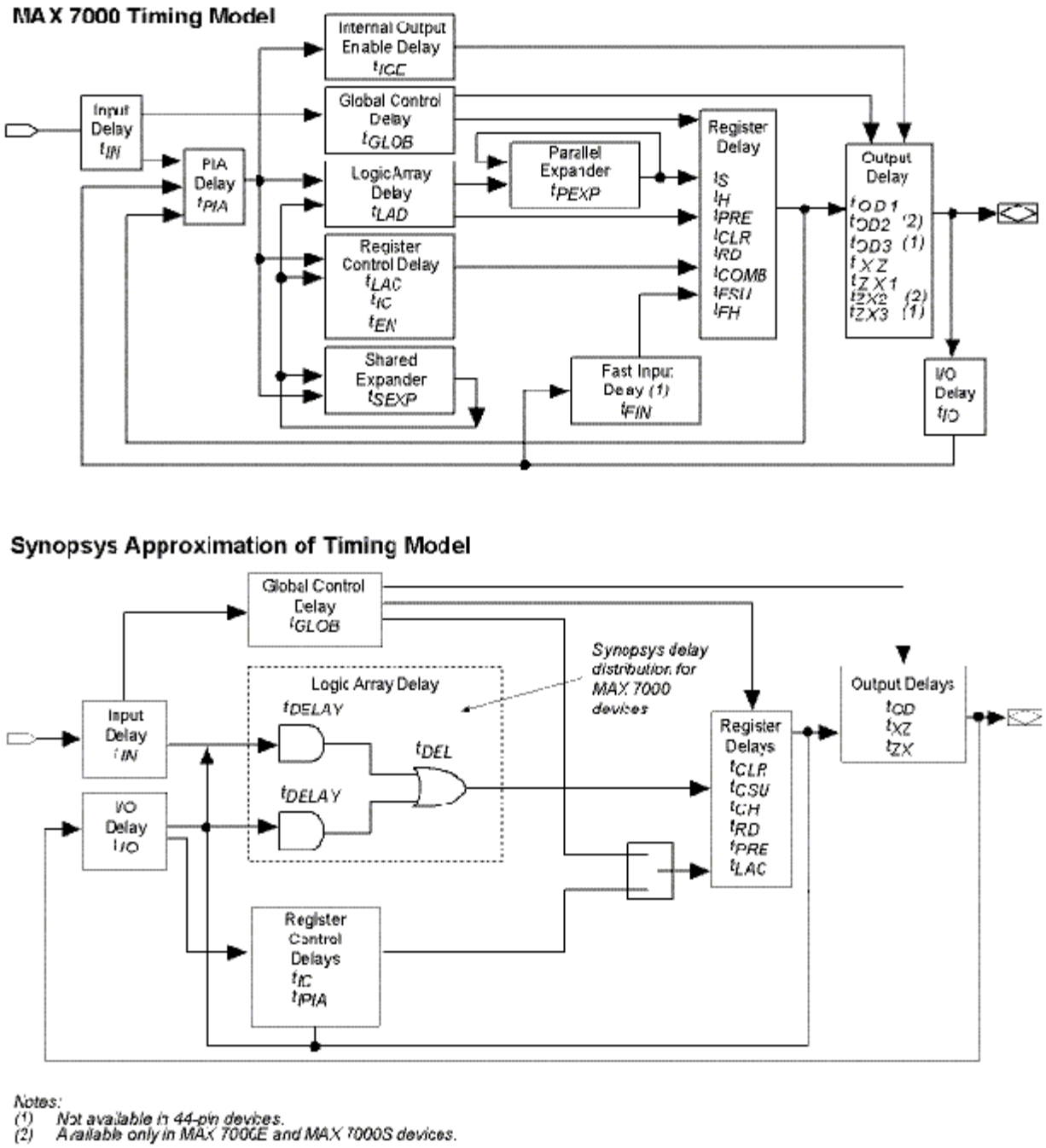


Figure 1. Standard MAX 7000 Timing Model vs. Synopsys Approximation of Timing Model

If you wish to direct the Synopsys Design Compiler or FPGA Compiler software to produce sum-of-products logic that approximates the MAX 7000 or MAX 9000 timing model, you can type the following **dc_shell** prompt commands at the command line before compiling the design:

```
set_structure false ↵

set_flatten -effort low ↵
```

When `set_structure` is set to `false`, structuring is turned off, and the Synopsys Design Compiler and FPGA Compiler software cannot factor and share logic between functions. If you do not enter these commands, the Synopsys compilers may add logic, which can create additional area and timing delays.

Figure 2 shows a combinatorial design that is predictable when structuring is turned off, but is unpredictable when structuring is turned on.

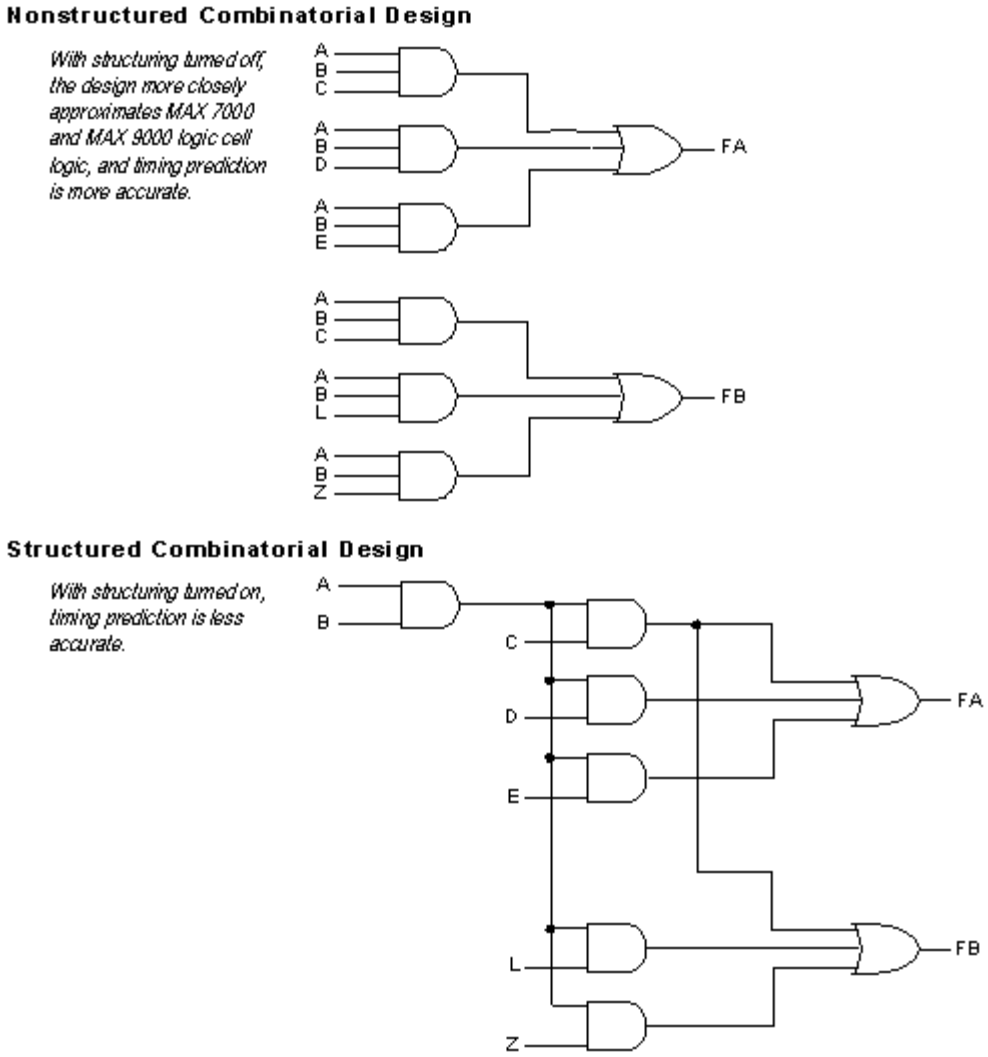


Figure 2. Nonstructured vs. Structured Combinatorial Design

When you use `low` as the argument to the `set_flatten -effort` command, the Synopsys compilers use the shortest compilation time to create the sum-of-products implementation of your design. If you use the `medium` or `high` argument, the Synopsys compilers create optimally flattened designs, but usually require greater compilation time and offer little improvement in timing and area results.

You can type `report_timing` ↵ after compilation to view Synopsys-generated timing information.

If you wish to calculate the area of your design, you can obtain an approximate logic cell count in several ways. Altera recommends that you add the number of registers and combinatorial outputs in a design. Depending on your design, this number may be slightly lower than the final number reported by the MAX+PLUS II software.

To create a file detailing primitive usage in the design, type `report_reference>` <*filename*> ↵ after Synopsys compilation.

☞ To obtain accurate timing information about your design, you must use the MAX+PLUS II Timing Analyzer to analyze your design. For accurate area information, consult the Report File (**.rpt**) generated by the MAX+PLUS II software.

Related Topics:

- Refer to the following sources for related information:
  - ***Synopsys Design Compiler Reference Manual*** or ***Synopsys Command Reference Manual***

- ***FPGA Compiler User Guide***
- Synthesizing & Optimizing VHDL & Verilog HDL Pojects with Synopsys Software

- Go to MAX Devices, which is available on the web, for additional information:

DesignWare FLEX 8000 Synthesis Example

Figure 1 shows a sample VHDL design, design_one.vhd, which illustrates component inference with the DesignWare interface for FLEX® 8000 devices.

Figure 1. VHDL Design File (design_one.vhd)

This design illustrates the sum of A + B.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;



ENTITY design_one IS
   PORT (a,b : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
         f   : OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
END design_one;



ARCHITECTURE add_design OF design_one IS



BEGIN
 f <= a + b;
END add_design;
```

When the VHDL Compiler or the HDL Compiler for Verilog software analyzes and elaborates the design, it replaces the "+" operator with its synthetic operator equivalent.

Figure 2 shows the design as it appears in the Design Analyzer software after it has been analyzed and elaborated by the VHDL Compiler software.

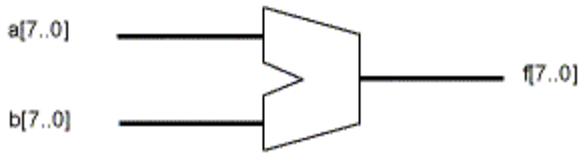


Figure 2. design_one.vhd after Analysis & Elaboration

When you synthesize a design, the Design Compiler or FPGA Compiler software uses the synthetic library to match the synthetic operator to the FLEX-optimized logical implementation in the technology library. The Synopsys Design Compiler or FPGA Compiler software then instantiates and interconnects the correct number of `flex_add` and `flex_carry` functions to produce the 8-bit adder shown in Figure 1. When you save a compiled design as a VHDL, Verilog HDL or EDIF file, the file preserves the number of `flex_add` and `flex_carry` functions, as well as their interconnections. Consequently, area and performance predictions that you make in the

Synopsys design environment closely match the final MAX+PLUS® II result.

Table 2 lists functions included in the DesignWare FLEX 6000, FLEX 8000, and FLEX 10K synthetic libraries.

Table 2. FLEX 6000, FLEX 8000, and FLEX 10K Synthetic Library Functions

Name	Function
`flex_add`	Sum of A, B, and Carry-In
`flex_carry`	Carry of A, B, and Carry-In
`flex_sub`	Difference of A, B, and Borrow-In
`flex_borrow`	Borrow of A, B, and Borrow-In
`flex_gt, flex_sgt`	Greater than (`flex_gt` is unsigned; `flex_sgt` is signed)
`flex_carry_gt`	Greater than Carry
`flex_lt, flex_slt`	Less than (`flex_lt` is unsigned; `flex_slt` is signed)
`flex_carry_lt`	Less than Carry
`flex_gteq, flex_sgteq`	Greater than or equal to (`flex_gteq` is unsigned; `flex_sgteq` is signed)
`flex_carry_gteq`	Greater than or equal to Carry
`flex_inc`	Incrementer (Count = Count + 1)
`flex_carry_inc`	Incrementer Carry (Count = Count + 1)
`flex_dec`	Decrementer (Count = Count - 1)
`flex_carry_dec`	Decrementer Carry (Count = Count - 1)
`flex_lteq, flex_slteq`	Less than or equal to (`flex_lteq` is unsigned; `flex_slteq` is signed)
`flex_carry_lteq`	Less than or equal to Carry
`flex_count`	Counter
`aflex_carry_count`	Counter Carry
`flex_add_sub`	Adder/Subtractor
`flex_inc_dec`	Incrementer/Decrementer
`flex_umult, flex_smult`	Multiplier (`flex_umult` is unsigned; `flex_smult` is signed)

Figure 3 shows **design_one.vhd** after it has been synthesized with the Design Compiler.

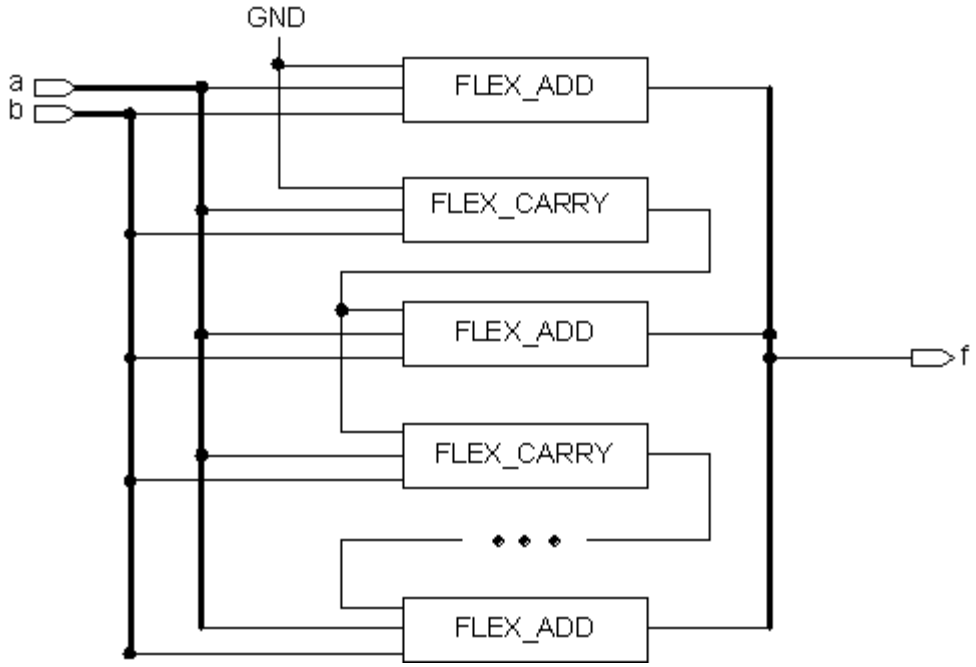


Figure 3. design_one.vhd Synthesized & Resolved for FLEX 6000, FLEX 8000 & FLEX 10K Architecture

After you save the design as an EDIF Input File (**.edf**) and process it with the MAX+PLUS II Compiler, the Compiler replaces instances of `flex_add` and `flex_carry` with FLEX-optimized versions, as shown in Figure 4. The MAX+PLUS II Compiler maps these functions into a single logic element (LE). The result is a high-speed 8-bit adder that fits into 8 LEs.

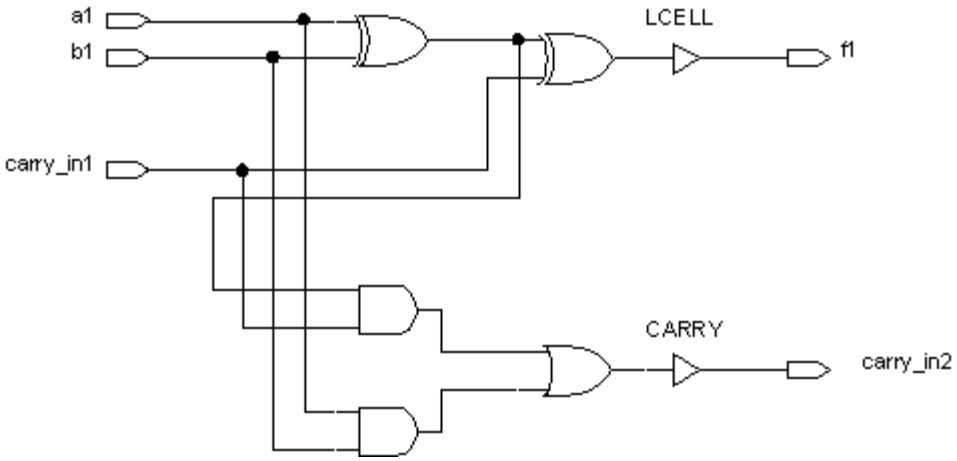


Figure 4. One Slice of the design_one 8-bit Adder Design with Optimized FLEX 8000 Functions

Related Topics:

- Refer to the following sources for related information on DesignWare and the Synopsys VHDL Compiler:
  - ***Synopsys DesignWare Databook***
  - ***VHDL Compiler Reference Manual***
- Go to FLEX Devices, which is available on the web, for additional information:

Using FPGA Compiler N-Input LUT Optimization for FLEX 6000, FLEX 8000 & FLEX 10K Devices

The Synopsys FPGA Compiler software supports an *N*-input look-up table (LUT) function that improves the quality of the results and the predictability of delay and resource estimates. All Altera® FPGA Compiler libraries for FLEX® 6000, FLEX 8000, and FLEX 10K devices support the *N*-input LUT function.

Figure 1 shows a sample command sequence that FPGA Compiler might require for N-input LUT optimization. To use N-input LUT optimization, include the edifout_write_properties_list = "lut_function" command.

Figure 1. Sample Command Sequence for N-Input LUT Optimization

```
read -f vhdl <design name>.vhd ↵
current_design = <design name> ↵
set_max_area 0 ↵
uniquify ↵
ungroup -all -flatten ↵
compile -ungroup_all ↵
report_area > <design name>.rpa ↵
report_fpga > <design name>.rpf ↵
report_cell > <design name>.rpc ↵
edifout_write_properties_list = "lut_function" ↵
write -f edif -hierarchy -o <design name>.edf ↵
```

Use the area report to determine the circuit area.

If you wish to maintain area report estimates as closely as possible during MAX+PLUS® II processing, Altera recommends that you select the *WYSIWYG* setting for the *Global Project Synthesis Style* in the **Global Project Logic Synthesis** dialog box (Assign menu). However, selecting the *Normal* or *Fast* style may yield a better result.

Related Topics:

- For more information on how to use the FPGA Compiler software optimize your design for FLEX 8000 devices, refer to *Chapter 5: Optimization for the Altera FLEX 8000 Architecture* in the ***Synopsys FPGA Compiler User Guide***.
- Go to FLEX Devices, which is available on the web, for additional information.

Entering Resource Assignments

The MAX+PLUS® II software allows you to enter a variety of resource and device assignments for your projects. Resource assignments are used to assign logic functions to a particular pin, logic cell, I/O cell, embedded cell, row, column, Logic Array Block (LAB), Embedded Array Block (EAB), chip, clique, local routing, logic option, timing requirement, or connected pin group. In the MAX+PLUS II software, you can enter all types of resource and device assignments with Assign menu commands. You can also enter pin, logic cell, I/O cell, embedded cell, LAB, EAB, row, and column assignments in the MAX+PLUS II Floorplan Editor. The Assign menu commands and the Floorplan Editor all save assignment information in the ASCII Assignment & Configuration File (**.acf**) for the project.

In designs targeted for the Synopsys Design Compiler and FPGA Compiler software, you can assign a limited subset of these resource assignments by setting attributes in the VHDL or Verilog HDL design files with the `set_attribute` command. These attributes are incorporated into the EDIF netlist file(s). The MAX+PLUS II

software automatically converts assignment information from the EDIF Input File (**.edf**) into the ACF format. For information on making MAX+PLUS II-compatible resource assignments with the `set_attribute` command, go to the following topics:

- Assigning Pins, Logic Cells, & Chips
- Assigning Cliques
- Assigning Logic Options

You can also modify the ACF for a design to contain timing requirements and other assignments, as described in the following topics:

- Modifying the Assignment & Configuration File with the **setacf** utility
- Converting Synopsys Timing Constraints into MAX+PLUS II-Compatible Format with the **syn2acf** Utility
- Converting Synopsys Hierarchical Timing Constraints into MAX+PLUS II-Compatible Format with the **gen_iacf** and **gen_hacf** Utilities

Related Topics:

- Refer to the following sources for related information:
  - Synopsys documentation for additional information on how to assign properties
  - "Resource Assignments in EDIF Input Files" and "Assigning Resources in a Third-Party Design Editor" in MAX+PLUS II Help for more information on assignments or properties that can be assigned in Synopsys
  - "resource assignments" or "ACF, format" in MAX+PLUS II Help using **Search for Help on** (Help menu), for information on entering assignments in the MAX+PLUS II software with Assign menu commands or in an ACF

Assigning Pins, Logic Cells & Chips

You can assign a single logic function to a specific pin or logic cell (including I/O cells and embedded cells) within a chip, and assign one or more functions to a specific chip. A chip is a group of logic functions defined as a single, named unit, which can be assigned to a specific device.

You can assign a signal to a particular pin to ensure that the signal is always associated with that pin, regardless of future changes to the project. If you wish to set and maintain the performance of your project, assigning logic to a specific logic cell within a chip can minimize timing delays. In a project that is partitioned among multiple devices, you can assign logic functions that must be kept together in the same device to a chip. Chip assignments allow you to split a project so that only a minimum number of signals travel between devices, and to ensure that no unnecessary device-to-device delays exist on critical timing paths. You can assign a chip to a device in MAX+PLUS® II software.

To make pin, logic cell, and chip assignments, use the `set_attribute` command at a **dc_shell** prompt. Before using the `set_attribute` command, add the following line to your **.synopsys_dc.setup** file:

```
edifout_write_properties_list= {LOGIC_OPTION, CLIQUE, CHIP_PIN_LC}
```

↵

Table 1 shows the syntax to use for chip, pin, and logic cell assignments:

Table 1. Commands for Chip, Pin, & Logic Cell Assignments

Assignment Type	Command to Type *Note (1)*
Chip	`set_attribute find (`*<design object>*`, (`*<instance name>*`)) "CHIP_PIN_LC" -type string "`*<chip name>*`"` ↵
Pin	`set_attribute find (`*<design object>*`, (`*<instance name>*`)) "CHIP_PIN_LC" -type string "`*<chip name>*`@`*<pin number>*`"` ↵
Logic cell number	`set_attribute find (`*<design object>*`, (`*<instance name>*`)) "CHIP_PIN_LC" -type string "`*<chip name>*`@LC`*<logic cell number>*`"` ↵
I/O cell number	`set_attribute find (`*<design object>*`, (`*<instance name>*`)) "CHIP_PIN_LC" -type string "`*<chip name>*`@IOC`*<I/O cell number>*`"` ↵
Embedded cell number	`set_attribute find (`*<design object>*`, (`*<instance name>*`)) "CHIP_PIN_LC" -type string "`*<chip name>*`@EC`*<embedded cell number>*`"` ↵

Note:

1. In this table, *<design object>* represents ports, references, cells, nets, or pins.

Examples:

```
set_attribute find (cell, (U1)) "CHIP_PIN_LC" -type string "chip1" ↵

set_attribute find (cell, (U1)) "CHIP_PIN_LC" -type string "chip1@K2" ↵

set_attribute find (cell, (U1)) "CHIP_PIN_LC" -type string "chip1@LC44" ↵
```

Related Topics:

- Go to "Devices & Adapters" and "Assigning a Device" in MAX+PLUS II Help for information on device pin-outs and assigning devices, respectively, in the MAX+PLUS II software.

Assigning Cliques

You can define a group of logic functions as a single, named unit, called a clique. The MAX+PLUS® II Compiler attempts to place all logic in the clique in the same logic array block (LAB) to ensure optimum speed. If the project does not use multi-LAB devices, or if it is not possible to fit all clique members into a single LAB, the clique assignment ensures that all members of a clique are placed in the same device. In FLEX® 6000, FLEX 8000, MAX® 9000, and FLEX 10K devices, the Compiler also attempts to place the logic in LABs in the same row. Cliques therefore allow you to partition a project so that only a minimum number of signals travel between LABs, and to ensure that no unnecessary LAB-to-LAB or device-to-device delays exist on critical timing paths.

To make pin, logic cell, and chip assignments, use the `set_attribute` command at a **dc_shell** prompt. Before using the `set_attribute` command, add the following line to your **.synopsys_dc.setup** file:

```
edifout_write_properties_list= {LOGIC_OPTION, CLIQUE, CHIP_PIN_LC}

    ↵
```

✔ To assign a clique, type the following command at a **dc_shell** prompt:

```
set_attribute find(<design object>,(<instance name>))"CLIQUE" -type string "<clique name>" ↵
```

For example:

```
set_attribute find (cell, (U1)) "CLIQUE" -type string "fast1" ↵
```

Related Topics:

- Go to the following topics in MAX+PLUS II Help for related information:
    - Assigning a Clique
    - Guidelines for Achieving Maximum Speed Performance

Assigning Logic Options

Logic options and logic synthesis style assignments allow you to guide logic synthesis with logic optimization features that are specific to Altera® devices. You can assign logic options and styles to individual logic functions in your design. The MAX+PLUS® II Compiler also uses a device family-specific default logic synthesis style for each project.

To make pin, logic cell, and chip assignments, use the `set_attribute` command at a **dc_shell** prompt. Before using the `set_attribute` command, add the following line to your **.synopsys_dc.setup** file:

```
edifout_write_properties_list = {LOGIC_OPTION, CLIQUE, CHIP_PIN_LC} ↵
```

✔ To assign a logic option or a logic synthesis style, type the following command at a **dc_shell** prompt:

```
set_attribute find(<design object>, (<instance name>)) "LOGIC_OPTION"
-type string "<logic option>=<value>" ↵
```

For example:

```
set_attribute find (cell, (U1)) "LOGIC_OPTION" -type string
"STYLE=FAST" ↵
```

To specify multiple logic options, use commas as separators.

For example:

```
set_attribute find (cell, (U1))"LOGIC_OPTION" -type string "STYLE=FAST,
CARRY_CHAIN=MANUAL" ↵
```

Related Topics:

- Go to "Resource Assignments in EDIF Input Files" and "Assigning Resources in a Third-Party Design Editor" in MAX+PLUS II Help for complete and up-to-date information on logic option and logic synthesis style assignments, including definitions and syntax of these assignments.

Modifying the Assignment & Configuration File with the setacf Utility

Altera provides the **setacf** utility to help you modify a project's Assignment & Configuration File (**.acf**) from the command line, without opening the file with a text editor. Type `setacf -h ↵` at a UNIX or DOS prompt to get help on this utility.

Converting Synopsys Timing Constraints into MAX+PLUS II-Compatible Format with the

syn2acf Utility

Altera provides the **syn2acf** utility, which is an interface program that converts Synopsys timing constraints from non-hierarchical designs into the MAX+PLUS® II Assignment & Configuration File (**.acf**) format. For information on converting timing constraints from hierarchical designs, refer to Converting Synopsys Hierarchical Timing Constraints into MAX+PLUS II-Compatible Format with the **gen_iacf** and **gen_hacf** Utilities.

The **syn2acf** utility requires the following input files:

- Flattened EDIF netlist file
- **dc_shell** script file
- Standard Delay Format (SDF) constraints construct
- SDF timing delay construct

To use the **syn2acf** utility, follow these steps:

1. Set the timing constraints by using one of the following methods:

   ✔ Start the Synopsys Design Analyzer and specify timing constraints by choosing appropriate menu commands.

   or:

   ✔ Create the <*design name*>**.cmd** file for use with a **dc_shell** script. See Figure 1.

☞ The **syn2acf** utility does not support `set_arrival` timing constraints for internal nodes.

Figure 1. Sample Command File (.cmd) for Setting Timing Constraints

```
create_clock -period 50 -waveform {0 25} CLK
set_clock_skew -delay 2 CLK
set_input_delay 10 IN2
set_input_delay 5 -clock CLK IN1
set_output_delay 20 OUT2
set_output_delay 5 -clock CLK OUT1
set_max_delay 25 -to OUT1
set_max_delay 35 -to OUT2
set_multicycle_path 2 -to n20_reg
```

1. Compile the design and run the **syn2acf** utility either from the command line or with a Design Compiler **dc** script:

   ✔ Compile the design, then type the following command from the UNIX prompt to start the **syn2acf** utility:

   `/usr/maxplus2/synopsys/bin/syn2acf` <*design name*> ↵

   or:

   ✔ Run a **dc** script inside the **dc_shell** script that reads the VHDL design, compiles it, and runs the **syn2acf** utility. Figure 2 shows a sample **dc** script.

The **syn2acf** utility uses the `ALT_HOME` environment variable, if it has been specified, to determine the MAX+PLUS II system directory; otherwise, it uses the **/usr/maxplus2** directory. To specify a different MAX+PLUS II system directory with the `ALT_HOME` environment variable, you can either edit the **.cshrc** file to specify the correct directory or type the following command at the UNIX prompt:

```
setenv ALT_HOME <MAX+PLUS II system directory> ↵
```

Figure 2. Sample Script for Running the syn2acf Utility

```
/* dc_script example to interface with syn2acf */
dc_shell <<!
read -f vhdl <design name>.vhd
include <design name>.cmd                          /*set timing constraints*/
compile
current_design=<design name>
include /usr/maxplus2/synopsys/bin/syn2acf.cmd     /*generate required files*/
sh /usr/maxplus2/synopsys/bin/syn2acf <design name> /*invoke syn2acf utility*/
quit
!
```

The **syn2acf** utility cannot support maximum Clock frequency ($\mathbf{f_{MAX}}$) correctly if more than one Clock skew is specified in the **dc_shell** command script. This problem occurs because the Synopsys `write_script` command drops the Clock skew information for the registers. The **syn2acf** utility will use the last Clock skew number to calculate $\mathbf{f_{MAX}}$.

The sample **dc** script includes the Altera® -provided **syn2acf.cmd** file, shown in Figure 3, to generate the required input files for the **syn2acf** utility.

Figure 3. Altera-Provided syn2acf.cmd File

```
ungroup -flatten -all
write -f edif
write_script > altsyn.dc
write_constraints -format sdf -cover_design
write_timing -format sdf
```

All timing assignments generated by the **syn2acf** utility are written to the Timing Requirement Assignments Section of the project's ACF, with the assignment source identifier `{synopsys}` at the end of each line. Figure 4 shows a sample ACF excerpt that contains Synopsys timing constraints generated by the **syn2acf** utility.

Figure 4. Sample ACF Excerpt with Synopsys Timing Constraints

```
TIMING_POINT
BEGIN
   "|OUT2"    : TCO = 15.00ns {synopsys};
   "|IN1"     : TPD = 10.00ns {synopsys};
   "|IN2"     : TPD = 5.00ns {synopsys};
   "|OUT1"    : TCO = 20.00ns {synopsys};
   "|IN1"     : TSU = 20.00ns {synopsys};
   "|IN2"     : TSU = 117.00ns {synopsys};
   "|CLK"     : FREQUENCY = 50.00ns {synopsys};
   "|n10_reg" : FREQUENCY = 100.00ns {synopsys};
END;
```

Altera provides sample files for these utilities in the **/usr/maxplus2/synopsys/bin** directory.

Performing a Pre-Routing or Functional Simulation with VSS Software

After you have synthesized and optimized a VHDL or Verilog HDL design with the Design Compiler or FPGA Compiler software, you can perform a pre-routing or functional simulation with the Synopsys VHDL System Simulator (VSS) software.

To perform a pre-routing/functional simulation, follow these steps:

1. Be sure to set up the working environment correctly, as described in the following topics:
    - Setting Up the MAX+PLUS II/Synopsys Working Environment
    - Setting Up Design Compiler & FPGA Compiler Configuration Files
    - Setting Up the DesignWare Interface
    - Setting Up VSS Configuration Files
2. Create a VHDL or Verilog HDL design file that follows the guidelines described in one of the following topics:
    - Creating VHDL Designs for Use with MAX+PLUS II Software
    - Creating Verilog HDL Designs for Use with MAX+PLUS II Software
3. Synthesize and optimize your design with the Design Compiler or FPGA Compiler, as described in Synthesizing & Optimizing VHDL & Verilog HDL Files with Design Compiler or FPGA Compiler Software.
4. Save your design as a VHDL Design File (**.vhd**).

    ☞ VSS requires each architecture/entity pair in a VHDL Design File to have a configuration. The Configuration Declaration is necessary for simulation, but not for synthesis.
5. Use VSS and one of the Altera pre-routing functional simulation libraries to simulate the design.
6. When you are ready to compile your project with MAX+PLUS II software, save the design as an EDIF netlist file (**.edf**), then process it as described in Compiling Projects with MAX+PLUS II Software.

Related Topics:

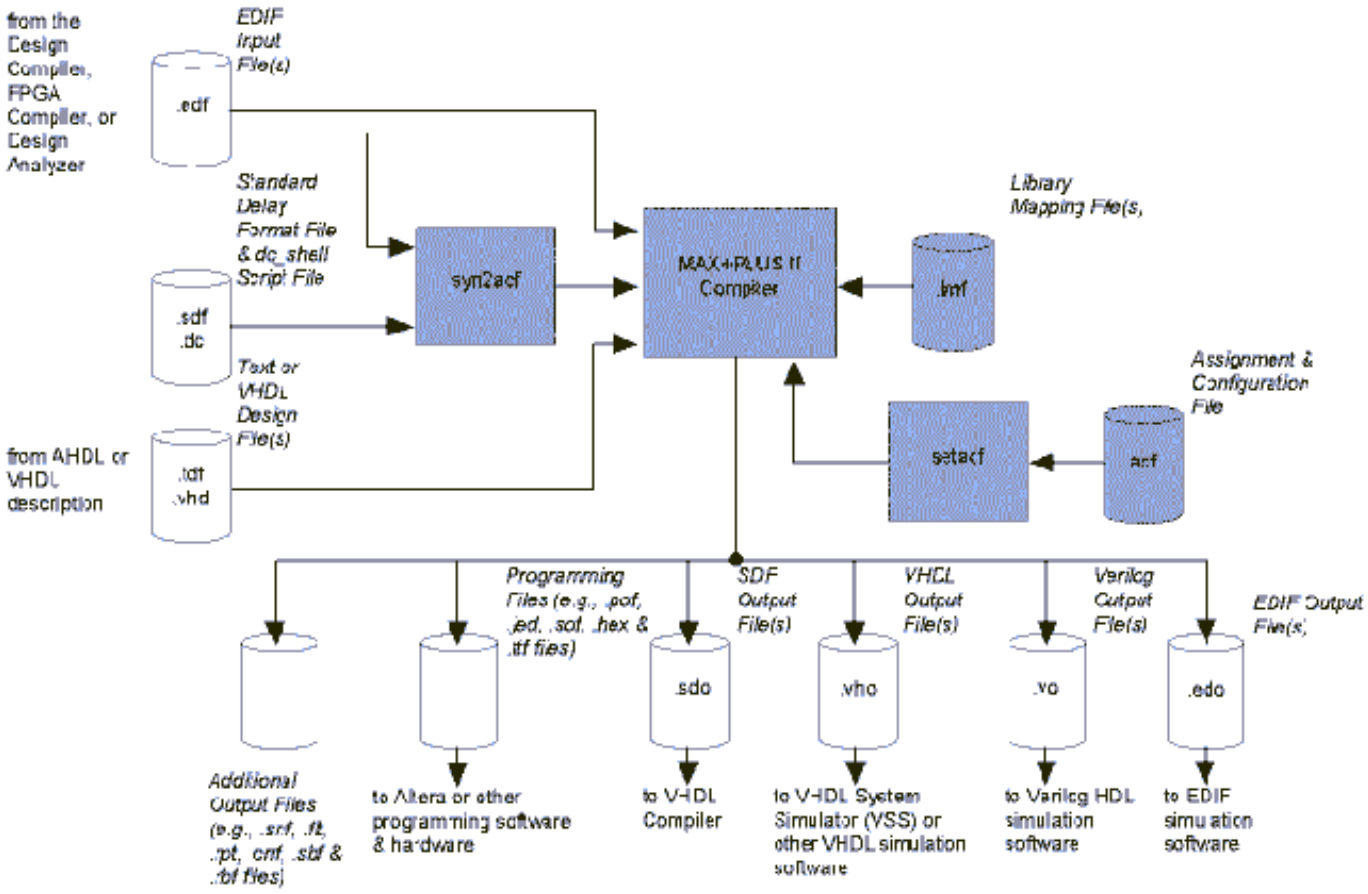
- Refer to the following sources for related information:
    - ***VHDL System Simulator Core Programs Manual*** for more information about VSS
    - Performing a Timing Simulation with VSS Software

Altera/Synopsys Project Compilation Flow

The following figure shows the MAX+PLUS® II /Synopsys project compilation flow.

Figure 1. MAX+PLUS II/Synopsys Project Compilation Flow

Altera-provided items are shown in blue.



Compiling Projects with MAX+PLUS II Software

The MAX+PLUS® II Compiler can process design files in a variety of formats. This topic describes how to use MAX+PLUS II software to compile projects in which the top-level design file is an EDIF Input File (with the extension **.edf**).

Related Topics:

- Refer to the following sources for additional information:
  - Go to MAX+PLUS II Help for information on compiling VHDL and Verilog HDL, design files directly with the MAX+PLUS II Compiler.
  - Go to Running Synopsys Compilers from MAX+PLUS II Software for information on running the Synopsys Design Compiler or FPGA Compiler software on a VHDL or Verilog HDL design from within the MAX+PLUS II Compiler window.

To compile a design (also called a "project") with MAX+PLUS II software, go through the following steps:

1. Create design files that are compatible with the MAX+PLUS II software and convert them into EDIF Input Files with the extension **.edf**. Specific instructions for some tools are described in these MAX+PLUS II ACCESS℠ Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your design entry or synthesis and optimization tool.

2. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, create a mapping for each function in a Library Mapping File (**.lmf**) to map the custom symbol to the corresponding EDIF Input File, AHDL Text Design File (**.tdf**), or other MAX+PLUS II-supported design file. These custom functions are represented in design files as hollow-body symbols or "black box" HDL descriptions.

   ☞ Go to "Library Mapping Files (**.lmf**)" in MAX+PLUS II Help for more information.

3. Open MAX+PLUS II and specify the name of your top-level design file as the project name with the **Project Name** command (File menu). If you open an HDL file in the MAX+PLUS II Text Editor, you can choose the **Project Set Project to Current File** command (File menu) instead.

   ☞ You can also compile a project from a command line. However, the first time you compile a project, the settings you need to specify are easier to specify from within the MAX+PLUS II software. After you have run the graphical user interface for the MAX+PLUS II software at least once, you can more easily use the command-line **setacf** utility to modify options in the Assignment & Configuration File (**.acf**) for the project. Type `setacf -h` ↵ and `maxplus2 -h` ↵ for descriptions of **setacf** and MAX+PLUS II command-line syntax.

4. Choose **Device** (Assign menu) and select the target Altera device family in the *Device Family* drop-down list box. If you wish to implement the design logic in a specific device, select it in the *Devices* box. Otherwise, select *AUTO* to allow the MAX+PLUS II Compiler to choose the best device(s) in the current device family. If your design entry or synthesis and optimization tool required you to specify a target family and/or device, specify the same information in this dialog box. For information on partitioning logic among multiple devices, go to MAX+PLUS II Help. Choose **OK**.

5. Open the Compiler window by choosing the **Compiler** command (MAX+PLUS II menu). Go through the following steps to specify the options necessary to compile the design file(s) in your project:

   1. Ensure that all EDIF netlist files have the extension **.edf** and choose **EDIF Netlist Reader Settings** (Interfaces menu).

   2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor. This name should be the name of the vendor whose tool(s) you used to create the EDIF netlist files. If your vendor name does not appear, select *Custom* instead.

      ☞ If you are compiling a design created with Synopsys FPGA Express software, select *Synopsys*, choose the **Customize** button, enter <*project name*>**.lmf** in the *LMF #1* box, choose **OK**, and skip to step 6.

   3. If you selected an existing vendor name in the *Vendor* box and your project contains design files that require custom LMF mappings, choose the **Customize** button to expand the dialog box to show all settings. Turn on the *LMF #2* checkbox and type your custom LMF's filename in the corresponding text box, or select a name from the *Files* box. The selection in the *Vendor* box will change to *Custom* and all settings will be retained until you change them again.

   4. If you selected *Custom* in the *Vendor* box, choose the **Customize** button to expand the dialog box to show all settings. Any previously defined custom settings will be displayed. Under *Signal Names*, type one or more names with up to 20 total name characters in the *VCC* or *GND* box if your EDIF Input File(s) use one or more names other than `VCC` or `GND` for the global high or low signals. Multiple signal names must be separated by either a comma (,) or a space. Under *Library Mapping Files*, turn on the *LMF #1* checkbox and type a filename in the text box following it, or select a name from the *Files* box. If necessary, specify another LMF name in the *LMF #2* box. Go to MAX+PLUS II Help for detailed information on the settings available in the **EDIF Netlist Reader Settings** dialog box.

5. Choose **OK**.

6. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, you may need to ensure that all files are present in your project directory, i.e., the same directory as the top-level design file. Otherwise, you must specify the directories containing these files as user libraries with the **User Libraries** command (Options menu).

7. Follow all guidelines that apply to your design entry or synthesis and optimization tool:

   - Exemplar Logic Galileo Extreme-Specific Compiler Settings
   - Synopsys DesignWare-Specific Compiler Settings
   - Converting Synopsys FPGA Compiler & Design Compiler Timing Constraints into MAX+PLUS II-Compatible Format with the **syn2acf** Utility
   - Synplicity Synplify-Specific Compiler Settings

8. If you wish to generate EDIF, VHDL, or Verilog HDL output files for post-compilation simulation or timing analysis with another EDA tool, go through the following steps:

   1. (Optional) Turn on the **Optimize Timing SNF** command (Processing menu) to reduce the size of the output file(s). Turning on this command can reduce the size of output netlists by up to 30%.

      This command does not create optimized timing SNFs on UNIX workstations. However, a non-optimized timing SNF provides the same functional and timing information as an optimized timing SNF.

   2. If you wish to generate EDIF Output Files (**.edo**), go through these steps:

      1. Turn on the **EDIF Netlist Writer** command (Interfaces menu). Then choose the **EDIF Netlist Writer Settings** command (Interfaces menu).

      2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor and choose **OK**. If your vendor name does not appear, select *Custom* instead and specify the settings that are appropriate for your simulation or timing analysis tool. Go to MAX+PLUS II Help for detailed information on the options available in the **EDIF Netlist Writer Settings** dialog box.

      3. To generate an optional Standard Delay Format (SDF) Output File (**.sdo**), choose the **Customize** button to expand the dialog box to show all settings. Select one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**.

      The filenames of the EDIF Output File(s) and optional SDF Output File(s) are the same as the user-defined chip name(s) for the project; if no chip names exist, the Compiler assigns filenames that are based on the project name. For a multi-device project, the Compiler also generates a top-level EDIF Output File that is uniquely identified by "**_t**" appended to the project name. In addition, the Compiler automatically generates a VHDL Memory Model Output File, *<project name>*.**vmo**, when it generates an EDIF Output File that contains memory (RAM or ROM).

   3. If you wish to generate VHDL Output Files (**.vho**), turn on the **VHDL Netlist Writer** command (Interfaces menu). Then choose **VHDL Netlist Writer Settings** command (Interfaces menu). Select *VHDL Output File (.vho)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF ver. 2.1 files contain timing delay information that allows you to perform back-annotation simulation in VHDL with VITAL-compliant simulation libraries. The VHDL Output Files generated by the Compiler have the extension **.vho**, but are otherwise named in the same way as the EDIF Output Files described above.

   4. If you wish to generate Verilog HDL Output Files (**.vo**), turn on the **Verilog Netlist Writer**

command (Interfaces menu). Then choose **Verilog Netlist Writer Settings** command (Interfaces menu). Select *Verilog Output File (.vo)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF Output Files contain timing delay information that allows you to perform back-annotation simulation in Verilog HDL. The Verilog Output Files generated by the Compiler have the extension **.vo**, but are otherwise named in the same way as the EDIF Output Files described above.

9. To run the MAX+PLUS II Compiler, choose the **Project Save & Compile** command (File menu) or choose the **Start** button in the Compiler window.

   ☞ See step 3 for information on running MAX+PLUS II software from the command line.

10. Once you have compiled the project with the MAX+PLUS II Compiler, you can use the VHDL, Verilog HDL, or EDIF output file(s), and the optional SDF Output File(s) (**.sdo**) to perform timing analysis or timing simulation with another EDA tool. Specific instructions for some tools are described in these MAX+PLUS II ACCESS Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your EDA tool.

The MAX+PLUS II Compiler also generates a Report File (**.rpt**), a Pin-Out File (**.pin**), and one or more of the following files for device programming or configuration:

- JEDEC Files (**.jed**)
- Programmer Object Files (**.pof**)
- SRAM Object Files (**.sof**)
- Hexadecimal (Intel-format) Files (**.hex**)
- Tabular Text Files (**.ttf**)

Related Topics:

- Refer to the following sources for additional information:
  - Go to *Compiler Procedures* in MAX+PLUS II Help for information on other available Compiler settings.
  - Go to *Programmer Procedures* in MAX+PLUS II Help for instructions on creating other types of programming files and on programming or configuring Altera devices.
  - Go to Back-Annotating MAX+PLUS II Pin Assignments to Design Architect Symbols for information on back-annotating pin assignments in Mentor Graphics Design Architect schematics.
  - Go to Programming Altera Devices for information on the different programming hardware options for Altera device families.
- Go to the following topics, which are available on the web, for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware

Synopsys DesignWare-Specific Compiler Settings

If you are compiling a design that was created with the Synopsys DesignWare software, follow these additional steps:

1. Choose **Global Project Logic Synthesis** (Assign menu).
2. Select the desired style in the *Global Project Synthesis Style* box.
3. Choose the **Define Synthesis Style** button to check and/or edit the selected style.
4. In the **Define Synthesis Style** dialog box, select *Manual* in the *Carry Chain* box and also in the *Cascade Chain* box.
5. Choose the **Advanced Options** button in the **Define Synthesis Style** dialog box.
6. Turn on the *SOFT Buffer Insertion* logic option in the **Advanced Options** dialog box if it is not on

already. This option should be turned on in all Synopsys designs.

7. Choose **OK** three times to close all dialog boxes.

Running Synopsys Compilers from the MAX+PLUS II Software

With MAX+PLUS® II software, you can automatically process Verilog HDL and VHDL designs with the Synopsys Design Compiler or FPGA Compiler by following these steps:

1. Create a project directory under your login directory.

2. Add the following environment variables to your **.cshrc** file:

```
setenv ALT_HOME /<MAX+PLUS II system directory> ↵
setenv SYNOPSYS /<Synopsys system directory> ↵
```

3. Add the **$ALT_HOME/synopsys/bin** and **$SYNOPSYS/$ARCH/syn/bin** directories to the PATH environment variable in your **.cshrc** file. The $ARCH environment variable specifies the platform on which the Synopsys Design Compiler is running. Valid platform names are `sparc`, `sparcOS5`, `rs6000`, and `hp700`.

4. Source your **.cshrc** file to update the environment variables.

If you use additional custom libraries, you must specify them in a **.synopsys_dc.setup** file, and verify that it contains the correct library settings for mapping to the target family. See Setting Up the Synopsys/MAX+PLUS II Working Environment for more information about the **.synopsys_dc.setup** file.

5. Create your project in Verilog HDL or VHDL using the MAX+PLUS II Text Editor or another standard text editor. You must save Verilog HDL files with the extension **.v** and VHDL files with the extension **.vhd**.

If you use the MAX+PLUS II Text Editor to create your design, you can insert templates for Verilog HDL or VHDL constructs with the **Verilog Template** and **VHDL Template** commands (Templates menu). The MAX+PLUS II Text Editor also provides syntax coloring for Verilog HDL and VHDL files to improve file readability.

6. In the MAX+PLUS II software, specify the project to be compiled with **Project Name** (File menu). Make sure the project name specified in MAX+PLUS II software matches both the name of the top-level design file and the Entity Declaration name specified in the top-level design file.

7. Click Button 1 on the **Compiler** toolbar button or choose the **Compiler** command (MAX+PLUS II menu) to open the Compiler.

8. In the MAX+PLUS II Compiler, turn on the **Synopsys Compiler** command (Interfaces menu).

9. Open the **Synopsys Compiler Settings** dialog box by choosing **Synopsys Compiler Settings** (Interfaces menu). Specify the appropriate options:

    1. Select either *Design Compiler or FPGA Compiler* in the *Compiler* box to specify which Synopsys compiler you want to process the design.

    2. If you wish to use the DesignWare interface and libraries, turn on the *DesignWare (FLEX® devices only)* option (FLEX 6000, FLEX 8000, and FLEX 10K devices only).

    3. To preserve the design hierarchy during Synopsys compilation, turn on the *Hierarchical*

Compilation option. Turning off this option allows the Synopsys compiler to flatten the design.

4. To allow the Synopsys compiler to optimize across all hierarchical boundaries, turn on the *Boundary Optimization* option.

5. Select the *Low, Medium,* or *High* option for *Mapping Effort*.

6. Choose **OK** to save all changes.

10. If you have turned on the *DesignWare (FLEX devices only)* option in the **Synopsys Compiler Settings** dialog box, ensure that the global project synthesis style uses the correct settings. Refer Compiling Projects with MAX+PLUS II Software for more information.

11. Specify the device(s) and output file(s) for the project. If you do not specify a device, the MAX+PLUS II Compiler automatically selects one or more devices from the current device family. Refer to Creating VHDL Design Files for Use with MAX+PLUS II Software for more information.

12. Choose the **Start** button to compile the project. The MAX+PLUS II software converts the EDIF Input File, flattens the project, fits it into one or more Altera® devices, and generates the selected output files, including programming files. The MAX+PLUS II Message Processor notifies you when one of the Synopsys compilers is processing your design. When it has finished processing the design file(s), the Synopsys compiler generates an EDIF netlist file for each design in the hierarchy, and the MAX+PLUS II software immediately compiles the EDIF Input File(s).

Altera provides the **mp2dc_ana** and **mp2dc_cmp** shell scripts, which specify Synopsys Design Compiler or FPGA Compiler settings automatically. These scripts read the settings you have specified in the **Device** (Assign menu) and **Synopsys Compiler Settings** (Interfaces menu) dialog boxes for the project device(s), search path, link library, target library, synthetic library options (if you have turned on the *DesignWare* option in the **Synopsys Compiler Settings** dialog box), and other optimization options. You do not need to provide your own **.synopsys_dc.setup** file unless you use libraries other than Altera libraries. See Setting Up Synopsys Configuration Files for more information.

The MAX+PLUS II software runs both the **mp2dc_ana** and **mp2dc_cmp** shell scripts automatically when you compile a VHDL or Verilog HDL design file with the **Synopsys Compiler** command (Interfaces menu) turned on. The **mp2dc_ana** shell script analyzes your designs and generates a single hierarchical **.db** database file. The analysis output information is recorded in the <*project name*>**.log** file. If the Design Compiler or FPGA Compiler generates errors or warning messages during processing, the messages appear in the MAX+PLUS II Message Processor window. You can select a message that includes a line number and click Button 1 on the **Locate** button to locate the source of a message in the MAX+PLUS II Text Editor. If no errors occur during analysis, the MAX+PLUS II software then starts the **mp2dc_cmp** shell script to read the **.db** file, compile the design, and generate an EDIF netlist file for each design file in the hierarchy, which the MAX+PLUS II software then processes as an EDIF Input File (**.edf**).

The the **mp2dc_ana** and **mp2dc_cmp** shell scripts are located in the **/usr/maxplus2/synopsys/bin** directory. You can copy the **mp2dc_cmp** shell script to your project directory and specify custom settings for your design, such as Clock frequency or timing constraints settings. Alternatively, you can create your own custom **dc_shell** script and name the file **my_mp2dc.scr**. The **mp2dc_cmp** shell script will then use the commands in the **my_mp2dc.scr** file and ignore the current settings or default settings for Synopsys compilation options. Figure 1 shows an excerpt of the Altera-provided **mp2dc_cmp** shell script.

Figure 1. Excerpt from Altera-Provided mp2dc_cmp Shell Script

```
read -f db $proj.db >> $proj.log
   if (dc_shell_status == {}) {
   quit
   }
```

```
current_design=$design
uniquify
set_max_area 0



designs= find(design, "*")
foreach (dsgn, designs) {
        current_design= dsgn
        edfout_file = ""
        edfout_file = dsgn
        edfout_file = edfout_file + ".edf"
        set_max_area 0



/* If you do not use my_mp2dc.scr to customize your compilation, the */
/* customizable settings in the following section are used. You can  */
/* customize these settings only if the mp2dc_cmp file is located in */
/* your project directory.                                           */

   if ( "$use_my_cmd" == "true") {
      include my_mp2dc.scr
   } else {
   /* if no hierarchical compilation, then flatten the design  */
      if ( "$hierarchical_compile" == "OFF") {
            set_structure false
            set_flatten -effort low
            ungroup -all
      }



/* test compile options */
      if ( "$boundary_opt" == "ON") {
            compile -boundary_optimization -map_effort $map_effort
      } else {
          compile  -map_effort $map_effort
      }



/* If you use FPGA Compiler for FLEX devices, the LUT equation is output.*/
      /* If you use Design Compiler for FLEX devices, a TBL cell is output.    */
      if ( ("$family" == "flex8000") || ("$family" == "flex10k") ) {



if ( "$synopsys_compiler" == "FPGA" ){
            edifout_write_properties_list = {"lut_function"}
         }  else {
            replace_fpga
         }
      }
 }    /* End of customizable compilation settings section */



write  -f edif current_design  -o edfout_file



if (dc_shell_status == {}) {
            quit
      }
}
```

Resynthesizing a Design Using the alt_vtl Library & a MAX+PLUS II SDF Output File

Altera provides the **alt_vtl.db** post-synthesis library for technology mapping or resynthesis. You can use this library with the MAX+PLUS® II -generated Standard Delay Format (SDF) Output File (**.sdo**) to retarget and resynthesize your design for another device family by performing the following steps:

To retarget and resynthsize a design, follow these steps:

1. Generate an EDIF Output File (**.edo**) and an SDF Output File (**.sdo**), as described in Compiling Projects with MAX+PLUS II Software.

2. Modify your **.synopsys_dc.setup** file to include the following lines:

```
search_path = {./usr/maxplus2/synopsys/library/alt_post/syn/lib
<target library path>}; ↵
target_library = {<target library path>}; ↵
symbol_library = {<target library symbol file>}; ↵
link_library = {alt_vtl.db}; ↵
```

3. In the Design Compiler or FPGA Compiler software, type the following commands to read in the EDIF and SDF output files:

```
read -f edif <design name>.edo ↵
read_timing -load_delay net <design name>.sdo ↵
```

4. Type the following commands to compile your design, report the timing information, and create an EDIF netlist file (**.edf**) that can be processed with the MAX+PLUS II Compiler.

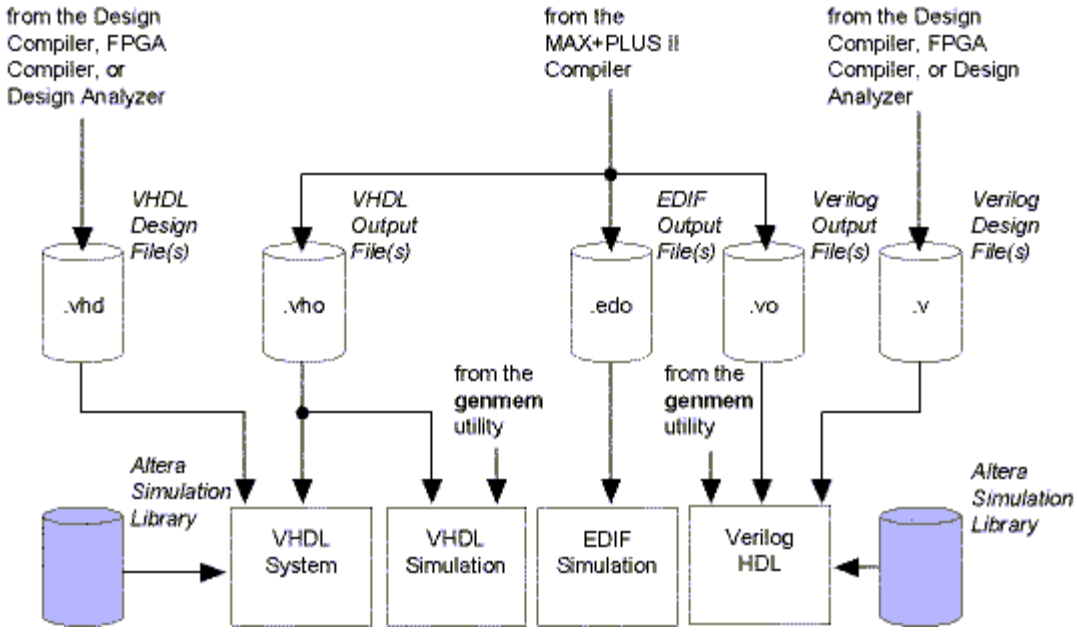
```
compile ↵
report_timing ↵
write -f edif -hierarchy -o <design name>.edf ↵
```

Project Simulation Flow

Figure 1 shows the project simulation flow for the MAX+PLUS® II /Synopsys interface.

Figure 1. MAX+PLUS II/Synopsys Project Simulation Flow

Altera-provided items are shown in blue.



The MAX+PLUS II/Synopsys design environment fully supports design verification with the Synopsys VHDL System Simulator (VSS). For pre-route simulation, you can simulate a design that has been compiled with one of the Synopsys compilers. For post-route simulation, you can simulate the VHDL Output File (**.vho**) that MAX+PLUS II® software generates during project compilation.

Performing a Timing Simulation with VSS Software

Once the MAX+PLUS® II software has compiled a project and generated a VHDL Output File (**.vho**) and an optional Standard Delay Format (SDF) Output File (**.sdo**), you can perform timing simulation with the Synopsys VHDL Simulator Software (VSS).

To simulate a VHDL Output File with VSS, follow these steps:

Be sure to set up the working environment correctly, as described in the following topics:

- Setting Up the MAX+PLUS II/Synopsys Working Environment
- Setting Up Design Compiler & FPGA Compiler Configuration Files
- Setting Up the DesignWare Interface
- Setting Up VSS Configuration Files

1. Generate a VHDL Output File (**.vho**) and an optional SDF Output File (**.sdo**), as described in Compiling Projects with MAX+PLUS II Software.

2. (Optional) Analyze the VITAL 95-compliant **alt_vtl** library , then back-annotate timing information through the SDF Output File:

    1. Use the **analyze_vss** script to analyze the **alt_vtl** Post-Routing Timing Simulation library, as described in Setting Up VSS Configuration Files.

    2. Enter the following command to back-annotate timing information through the SDF Output File:

        `vhdlsim -sdf_top /`*<design name>*`/`*<design name>* `-sdf` *<design name>*`.sdo` ↵

3. Simulate the VHDL Output File with the VSS software.

Related Topics:

- Go to the ***VSS User's Guide*** for more details on post-routing simulation.

Using Viewlogic Fusion/VCS & MAX+PLUS II Software

The following topics describe how to use the Viewlogic Fusion/VCS software with MAX+PLUS® II software. Click on one of the following topics for information:

This file is suitable for printing only. It does not contain hypertext links that allow you to jump from topic to topic.

Setting Up the MAX+PLUS II/Viewlogic Powerview Working Environment

- Software Requirements
- MAX+PLUS II/Viewlogic Powerview Interface File Organization
- Viewlogic Powerview **viewdraw.ini** Configuration File
- MAX+PLUS II/Viewlogic Powerview Project File Structure
- Altera-Provided Logic & Symbol Libraries
- The **vdpath** & **mega_lpm** Libraries

Simulation

- Project Simulation Flow
- Initializing Registers in VHDL & Verilog Output Files for Power-Up before Simulation
- Performing a Timing Simulation with Fusion/VCS for Powerview Software

Related Topics:

- Go to the following MAX+PLUS II ACCESS℠ Key topics for related information:
  - Compiling Projects with MAX+PLUS II Software
  - Programming Altera® Devices
- Go to the following topics, which are available on the web, for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware
  - Viewlogic web site (http://www.viewlogic.com)

Setting Up the MAX+PLUS II/Viewlogic Powerview Working Environment

To use the MAX+PLUS® II software with Viewlogic's Powerview software, you must install the MAX+PLUS II software, familiarize yourself with the Altera® Toolbox in the Powerview Cockpit, and then establish an environment that facilitates entering and processing designs. The MAX+PLUS II /Viewlogic Powerview interface is installed automatically when you install the MAX+PLUS II software on your workstation.

To set up your working environment for the MAX+PLUS II/Viewlogic Powerview interface, follow these steps:

1. Ensure that you have correctly installed the MAX+PLUS II and Viewlogic software versions described in MAX+PLUS II/Viewlogic Powerview Software Requirements.

2. Add the following environment variable to your **.cshrc** file to specify **/usr/maxplus2** as the MAX+PLUS II system directory:

   ```
   setenv ALT_HOME /usr/maxplus2 ↵
   ```

3. Add the **$ALT_HOME/viewlogic/standard**, **$ALT_HOME/bin**, and **$ALT_HOME/viewlogic/bin** directories to the `PATH` environment variable in your **.cshrc** file.

4. Add the **$ALT_HOME/viewlogic/standard** directory to the `WDIR` environment variable in your **.cshrc** file using the following syntax:

   `setenv WDIR $ALT_HOME/viewlogic/standard:/<`*Powerview system directory*`>/standard ↵`

   ☞ Make sure the **$ALT_HOME/viewlogic/standard** directory is the first directory in your `WDIR` path.

5. Source your **.cshrc** file by typing `source .cshrc ↵` at the UNIX prompt.

6. Create the Viewlogic Powerview **viewdraw.ini** configuration file.

7. Copy the **/usr/maxplus2/maxplus2.ini** file to your **$HOME** directory:

   ```
   cp /usr/maxplus2/maxplus2.ini $HOME ↵

   chmod u+w $HOME/maxplus2.ini ↵
   ```

   The **maxplus2.ini** file contains both Altera- and user-specified initialization parameters that control the MAX+PLUS II software, such as MAX+PLUS II symbol and logic function library paths and the current project name. The MAX+PLUS II installation procedure creates and copies the **maxplus2.ini** file to the **/usr/maxplus2** directory.

   ☞ Normally, you do not have to edit your local copy of **maxplus2.ini**, because the MAX+PLUS II software updates the file automatically whenever you change any parameters or settings. However, if you move the **max2lib** and **max2inc** library subdirectories, you must update the file. Go to "Creating & Using a Local Copy of the **maxplus2.ini** File" in MAX+PLUS II Help for more information.

8. If you plan to instantiate Library of Parameterized Modules (LPM) functions in ViewDraw schematics, you must create a new file with the name **vdraw.vs**. The **vdraw.vs** file must include the following line:

   ```
   load ("vdpath")
   ```

   You must also make sure that you specify the **vdraw.vs** file in your `WDIR` path.

9. Set up a directory structure that facilitates working with the MAX+PLUS II/Viewlogic Powerview interface. Refer to MAX+PLUS II/Viewlogic Powerview Project File Structure.

Related Topics:

- Go to *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual for more information on installation and details on the directories that are created during MAX+PLUS II installation. Go to MAX+PLUS II/Viewlogic Powerview Interface File Organization for information about the MAX+PLUS II/Viewlogic Powerview directories that are created during MAX+PLUS II installation.

- Go to the following topics, which are available on the web, for additional information:
  - ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
  - This manual is also available in 4 parts:
    - Preface & Section 1: MAX+PLUS II Installation
    - Section 2: MAX+PLUS II - A Perspective
    - Section 3: MAX+PLUS II Tutorial
    - Appendices, Glossary & Index

MAX+PLUS II/Viewlogic Powerview Software Requirements

The following applications and utilities are used to generate, process, synthesize, and verify a project with MAX+PLUS® II and Viewlogic Powerview software.

Viewlogic		Altera
ViewDraw	ViewGen	MAX+PLUS II version 9.4
VHDL Analyzer	ViewPath (optional)	
Vantage VHDL Analyzer	ViewTrace	
VHDL -> sym	ViewData Path	
edifneto	MOTIVE version 5.1.6 *Note (1)*	
edifneti	MOTIVE for Powerview version 3.2.1 (optional) *Note (1)*	
EEDIF (optional)	SDF2MTV (optional)	
MMP (optional)	Fusion/VCS	
vsm		

Note:

(1) MOTIVE for Powerview, a wrapper application for MOTIVE, provides a graphical user interface for the utilities (i.e., EEDIF, SDF2MTV, and MMP) used during a static timing verification with MOTIVE. MOTIVE alone does not accept EDIF files through the Setup Advisor.

☞ The MAX+PLUS II **read.me** file provides up-to-date information on which versions of Viewlogic Powerview applications the current version of the MAX+PLUS II software supports. It also provides information on installation and operating requirements. You should read the **read.me** file on the CD-ROM before installing the MAX+PLUS II software. After installation, you can open the **read.me** file from the MAX+PLUS II Help menu.

MAX+PLUS II/Viewlogic Powerview Interface File Organization

Table 1 shows the MAX+PLUS® II/Viewlogic Powerview interface subdirectories that are created in the MAX+PLUS II system directory (by default, the **/usr/maxplus2** directory) during MAX+PLUS II installation.

☞ For information on the other directories that are created during MAX+PLUS II installation, see "MAX+PLUS II File Organization" in *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual.

Table 1. MAX+PLUS II Directory Organization

Directory	Description
./lmf	Contains the Altera-provided Library Mapping File, **vwlogic.lmf**, that maps Viewlogic logic functions to equivalent MAX+PLUS II logic functions.
./viewlogic	Contains the **alt_edif.cfg** EDIF configuration file that is used with the **edifneti** utility. Also contains the library and sample subdirectories.

./viewlogic/examples	Contains the sample Viewlogic designs.
./viewlogic/library/max2sim	Contains the MAX+PLUS II simulation model library (**max2_sim**) for use in ViewSim software.
./viewlogic/library/alt_max2	Contains MAX+PLUS II primitives (`EXP`, `GLOBAL`, `LCELL`, `SOFT`, `CARRY`, `CASCADE`, `DFFE`, `DFFE6K`, and `OPNDRN`), macrofunctions (`a_8fadd`, `a_8mcomp`, `a_8count`, `a_81mux`), and megafunctions (`clklock`) for use in ViewDraw schematics. These logic functions support specific architectural features of Altera® devices. The **alt_max2** library also contains modified versions of the ViewDraw primitives that use tri-state buffers, because these primitives require special handling in the MAX+PLUS II /Viewlogic Powerview interface.
./viewlogic/library/synlib	Contains the Altera-provided synthesis library **altera**, which includes MAX+PLUS II primitives, the **altera.sml** file, a **sym** directory, and a **wir** directory for use with ViewSynthesis software.
./viewlogic/library/alt_mf	Contains the VHDL models for the MAX+PLUS II primitives (`EXP`, `GLOBAL`, `LCELL`, `SOFT`, `CARRY`, `CASCADE`, `DFFE`, and `OPNDRN`), macrofunctions (`clklock`) for use with ViewSynthesis software, the Vantage VHDL Analyzer software, and the VHDL source files. These logic functions are used to maintain portability to other architectures.
./viewlogic/library/alt_time	Contains MOTIVE timing models for MAX+PLUS II logic functions (**motive.lib**), including the `clklock` megafunction, and MAX+PLUS II driver models (**motive.drv**).
./viewlogic/library/alt_vtl	Contains the VHDL source files for the VITAL 3.0-compliant library. This library is available for ViewSim software.
./viewlogic/bin	Contains all MAX+PLUS II, Viewlogic, and interface-related scripts.
./viewlogic/standard	Contains all standard **.ini** files and standard tools.

Related Topics:

- Go to the following topics, which are available on the web, for additional information:
  - ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
  - This manual is also available in 4 parts:
    - Preface & Section 1: MAX+PLUS II Installation
    - Section 2: MAX+PLUS II - A Perspective
    - Section 3: MAX+PLUS II Tutorial
    - Appendices, Glossary & Index

Viewlogic Powerview viewdraw.ini Configuration File

Each Powerview project is configured with the **viewdraw.ini** file that resides in the project directory. The `DIR` statements at the end of **viewdraw.ini** are paths to library directories that are used by the various Powerview applications. Figure 1 shows a sample of the `DIR` statements that are required to use the libraries.

Figure 1. Excerpt from viewdraw.ini

```
DIR [pw] .
DIR [r]  /usr/maxplus2/vwlogic/library/alt_max2 (alt_max2)
DIR [r]  /usr/maxplus2/vwlogic/library/max2sim (max2_sim)
DIR [r]  /usr/maxplus2/vwlogic/library/synlib (altera)
DIR [r]  /usr/maxplus2/vwlogic/library/alt_mf (alt_mf)
DIR [r]  /usr/maxplus2/vwlogic/library/alt_vtl (alt_vtl)
DIR [rm] /<Powerview system directory>/lib/builtin (builtin)
DIR [rm] /<Powerview system directory>/simmods/vl/dip/74ls (vl74ls)
```

DIR [rm] /<*Powerview system directory*>/symsets/vl/dip/74ls (vl74ls)

DIR [r] /<*Powerview system directory*>/lib/vdpath (vdpath)

When you add the libraries to the **/usr/maxplus2/vwlogic/standard/viewdraw.ini** file, they are automatically set when you create a new project. Powerview tools search these libraries sequentially, so it is important to add them in the order in which they are listed in Figure 1.

Table 1 shows the libraries that must be specified in the DIR statements in the **viewdraw.ini** file.

Table 1. Powerview Application Libraries

Library	Library Alias	Source	Topics
alt_max2	alt_max2	Altera	Graphical elements for ViewDraw
max2sim	max2_sim	Altera	Models for project simulation
synlib	altera	Altera	VHDL synthesis library for the MAX+PLUS ® II software
alt_mf	alt_mf	Altera	VHDL models of MAX+PLUS II logic functions
alt_vtl	alt_vtl	Altera	VITAL-compliant primitives
builtin	builtin	Altera	Basic primitives such as INPUT pins, OUTPUT pins, AND gates, OR gates, etc.
74ls	vl74ls	Viewlogic	74-series macrofunctions
vdpath	vdpath	Viewlogic	Standard library of parameterized modules (LPM) functions

The Altera-provided libraries must be listed before the Viewlogic-provided libraries in the **viewdraw.ini** file to ensure that the correct versions of the megafunctions, macrofunctions, and primitives are used.

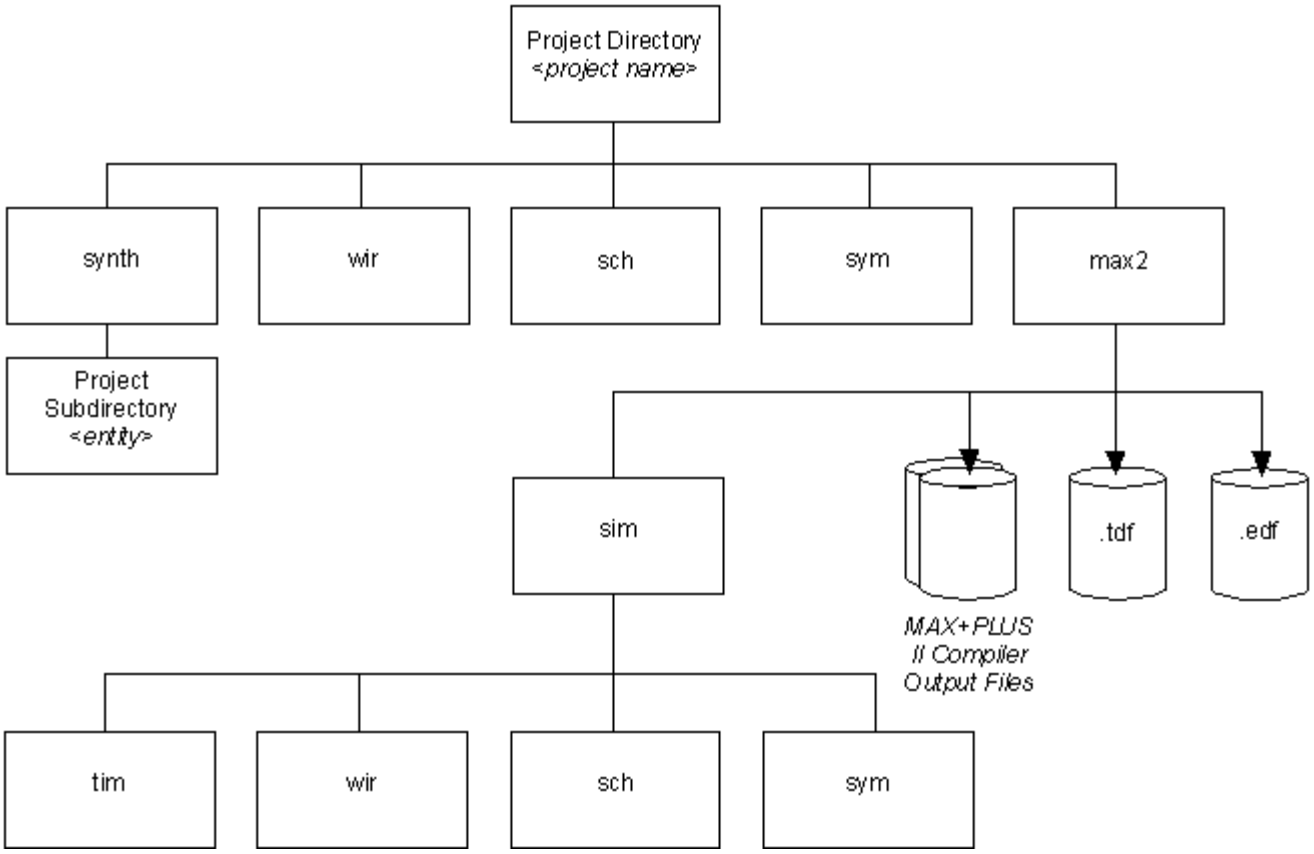
Related Topics:

- Go to Altera-Provided Logic & Symbol Libraries for more information on Altera-supplied libraries. Refer to the Powerview documentation for more information on setting up the **viewdraw.ini** file.

MAX+PLUS II/Viewlogic Powerview Project File Structure

In the MAX+PLUS® II software, a project name is the name of a top-level design file, without the filename extension. This design file can be an EDIF, Verilog HDL, or VHDL netlist file; an Altera® Hardware Description Language (AHDL) TDF; or any other MAX+PLUS II- supported design file. The EDIF netlist file must be created by Powerview and imported into the MAX+PLUS II software as an EDIF Input File (**.edf**). Figure 1 shows an example of MAX+PLUS II project directory structure that includes Powerview-generated files.

Figure 1. Sample MAX+PLUS II Project Organization



The MAX+PLUS II software stores the connectivity data on the links between design files in a hierarchical project in a Hierarchy Interconnect File (**.hif**), but refers to the entire project only by its project name. The MAX+PLUS II Compiler uses the HIF to build a single, fully flattened project database that integrates all the design files in a project hierarchy.

Unlike Powerview, the MAX+PLUS II software does not automatically create a project directory when you create a project. A single directory can contain several MAX+PLUS II design files, and you can specify any one of the designs in the directory as a project in the MAX+PLUS II software.

Viewlogic Powerview Local Work Area Structure

When you create a project with the Powerview Cockpit's **Create** command (Project menu), the project directory is created. You should generate design files and functional simulation files under this directory. A **max2** subdirectory is automatically created under your current project directory when you generate an EDIF file from your schematic or VHDL file. The ***<project name>*.edf** file is stored in the **max2** subdirectory. All MAX+PLUS® II Compiler output files are created in the **/*<project name>*/max2** subdirectory.

☞ ViewDraw files are identified by their directories and not by their extensions, so it is easy to overwrite files unintentionally. To avoid overwriting files, Altera recommends that you create a new project directory, ***<project name>*/max2/sim**, where you can generate all the files needed for simulation.

ViewDraw Project File Structure

Each ViewDraw project directory contains three subdirectories: **wir**, **sch**, and **sym**. See Table 1.

Table 1. ViewDraw Subdirectories

Directory	Topics

./wir	Wirelist files that contain connectivity information for a particular logic block
./sch	Schematics that contain logic
./sym	Symbol files that are the ViewDraw graphical representation of the logic blocks

Each file type uses the filename extension **.1**. Different file types are distinguished only by their directory: **/lib/wir/**<*project name*>**.1** is a wirelist file; **/lib/sch/**<*project name*>**.1** is the corresponding schematic file; and **/lib/sym/**<*project name*>**.1** is the corresponding symbol.

VHDL Project File Structure

Each VHDL project directory contains three subdirectories. See Table 2.

Table 2. VHDL Subdirectories

Directory	Topics
./synth	All synthesis-related files and directories
./synth/<*entity*>	Four types of files: <*entity*>**.pdf**, <*entity*>**.opt**, <*entity*>**.sta**, and <*entity*>**.gnl**
./wir	Wirelist for synthesized VHDL modules

☞ For each VHDL entity in the design, there is a corresponding **./synth/<*entity*>** directory.

Altera-Provided Logic & Symbol Libraries

The MAX+PLUS® II/Viewlogic Powerview environment provides libraries for compiling, synthesizing, and simulating designs.

☞ You can create your own libraries of custom symbols and logic functions for use in ViewDraw schematics and VHDL design files. You can use custom symbols (and functions) to incorporate an EDIF Input File, TDF, or any other MAX+PLUS II-supported design file into a project. The MAX+PLUS II software uses the **vwlogic.lmf** Library Mapping File to map ViewDraw symbols to equivalent MAX+PLUS II megafunctions, macrofunctions, or primitives. To use custom symbols and functions, you can create a custom LMF that maps your custom functions to equivalent EDIF Input Files, TDFs, or other MAX+PLUS II-supported design files. Go to "Library Mapping File" and "Viewlogic Library Mapping File" in MAX+PLUS II Help for more information.

Logic symbols used in ViewDraw software are available from the MAX+PLUS II **alt_max2** library, the ViewDraw **builtin** and **74ls** libraries, and the ViewDatapath **vdpath** library. VHDL models of MAX+PLUS II logic functions are available from the Altera-provided **alt_mf** library.

The alt_max2 Library

The **alt_max2** library provides MAX+PLUS II-specific logic functions that can be used to take advantage of special architectural features in each Altera® device family. See Table 1. Symbols and functional simulation models are available for all of these elements.

The alt_mf Library

The Altera-provided **alt_mf** library, which supports the Viewlogic Vantage VHDL Analyzer software, contains VHDL simulation models for all logic functions listed in the following table. The library is configured so that these functions pass untouched through the EDIF netlist file to the MAX+PLUS II Compiler, providing you with optimal control over design processing. Altera also provides models for all of the logic functions that you can synthesize and simulate. These models allow you to perform functional VHDL simulation while maintaining an architecture-independent VHDL description.

Table 1. Architecture Control Logic Functions

Name *Note (1), Note (2)*	Description	Name	Description	Name	Description
8fadd	8-bit full adder macrofunction	LCELL	Logic cell buffer primitive	EXP	MAX® 5000, MAX 7000, and MAX 9000 Expander buffer primitive
8mcomp	8-bit magnitude comparator macrofunction	GLOBAL	Global input buffer primitive	SOFT	Soft buffer primitive
8count	8-bit up/down counter macrofunction	CASCADE	FLEX® 6000, FLEX 8000, and FLEX 10K cascade buffer primitive	OPNDRN	Open-drain buffer primitive
81mux	8-to-1 multiplexer macrofunction	CARRY	FLEX 6000, FLEX 8000, and FLEX 10K cascade buffer primitive	DFFE *Note (2)*	D-type flipflop with Clock Enable primitive
clklock	Phase-locked loop megafunction				

Notes:

1. Logic function names that begin with a number must be prefixed with "a_" in VHDL designs. For example, 8fadd must be specified as a_8fadd.
2. For designs that are targeted to FLEX 6000 devices, you should use the DFFE primitive only if the design contains either a Clear or Preset signal, but not both. If your design contains both a Clear and a Preset signal, you must use the DFFE6K primitive.

Related Topics:

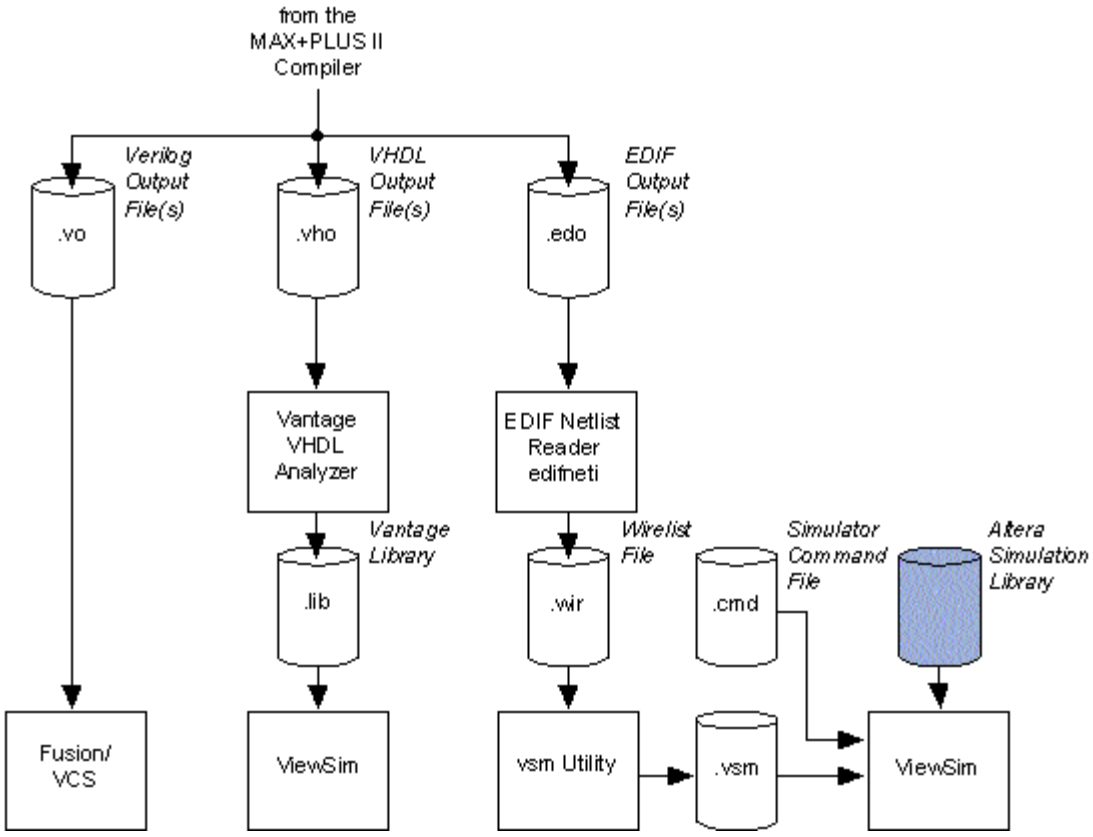
- Choose **Old-Style Macrofunctions**, **Primitives**, or **Megafunctions/LPM** from the MAX+PLUS II Help menu for detailed information on these functions.
- Go to the following topics, which are available on the web, for additional information:
  - FLEX Devices
  - MAX Devices
  - Classic Device Family

MAX+PLUS II/Viewlogic Powerview Simulation Flow

Figure 1 shows the project simulation flow for the MAX+PLUS® II/Viewlogic Powerview interface.

Figure 1. MAX+PLUS II/Viewlogic Powerview Project Simulation Flow

Altera-provided items are shown in blue.



Performing a Timing Simulation with Fusion/VCS for Powerview Software

After you have compiled a project with the MAX+PLUS® II software to generate a VHDL Output File (**.vho**) and a Standard Delay Format (SDF) Output File (**.sdo**), you can perform a timing simulation with Fusion/VCS software.

To simulate a project with Fusion/VCS software, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II /Viewlogic Powerview Working Environment.
2. Generate a VHDL Output File (**.vdo**) and an SDF ver 2.1 or 1.0 Output File (**.sdo**) for your project, as described in Compiling Projects with MAX+PLUS II Software.
3. Create a new **sim** directory under your **max2** directory to contain your Fusion/VCS simulation-related files.
4. To use the SDF Output File with the Fusion/VCS software, create a PLI table file (**.tab**) in the ***<project name>*/max2/sim** directory that contains the following line:

   `$sdf_annotate call=sdf_annotate_call acc=tchk, mp:`*<project name>* ↵
5. Open the **Fusion/VCS** dialog box by choosing the **max2_VCS** button from the Altera® Design Tools Drawer in the Powerview Cockpit.
   1. Type *<project name>*`/max2/`*<project name>*`.vo` in the *Verilog Design and Object Files* box.
   2. Type *<project name>*`.tab` in the *PLI Table File* box.
   3. Type *<project name>*`/max2/alt_max2.vo` in the *Verilog Library File 1* box.
   4. (Optional) To use a command file or to set stimuli during simulation, select the *Debug* option in the

VCS box and type the name of the command file in the *Simulation Command-file* box.

☞ When using a command file or setting stimuli, include the signal scope as part of the signal name. For example, to manipulate clk, a top-level signal in the **fadd** project, name the signal as `fadd.clk`.

5. Choose **OK**.

Related Topics:

- Go to Performing a Timing Simulation with ViewSim Software in these MAX+PLUS II ACCESS℠ Key topics for related information:

Compiling Projects with MAX+PLUS II Software

The MAX+PLUS® II Compiler can process design files in a variety of formats. This topic describes how to use MAX+PLUS II software to compile projects in which the top-level design file is an EDIF Input File (with the extension **.edf**).

☞ Refer to the following sources for additional information:

- Go to MAX+PLUS II Help for information on compiling VHDL and Verilog HDL, design files directly with the MAX+PLUS II Compiler.
- Go to Running Synopsys Compilers from MAX+PLUS II Software for information on running the Synopsys Design Compiler or FPGA Compiler software on a VHDL or Verilog HDL design from within the MAX+PLUS II Compiler window.

To compile a design (also called a "project") with MAX+PLUS II software, go through the following steps:

1. Create design files that are compatible with the MAX+PLUS II software and convert them into EDIF Input Files with the extension **.edf**. Specific instructions for some tools are described in these MAX+PLUS II ACCESS℠ Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your design entry or synthesis and optimization tool.

2. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, create a mapping for each function in a Library Mapping File (**.lmf**) to map the custom symbol to the corresponding EDIF Input File, AHDL Text Design File (**.tdf**), or other MAX+PLUS II-supported design file. These custom functions are represented in design files as hollow-body symbols or "black box" HDL descriptions.

   ☞ Go to "Library Mapping Files (**.lmf**)" in MAX+PLUS II Help for more information.

3. Open MAX+PLUS II and specify the name of your top-level design file as the project name with the **Project Name** command (File menu). If you open an HDL file in the MAX+PLUS II Text Editor, you can choose the **Project Set Project to Current File** command (File menu) instead.

   ☞ You can also compile a project from a command line. However, the first time you compile a project, the settings you need to specify are easier to specify from within the MAX+PLUS II software. After you have run the graphical user interface for the MAX+PLUS II software at least once, you can more easily use the command-line **setacf** utility to modify options in the Assignment & Configuration File (**.acf**) for the project. Type `setacf -h` ↵ and `maxplus2 -h` ↵

for descriptions of **setacf** and MAX+PLUS II command-line syntax.

4. Choose **Device** (Assign menu) and select the target Altera device family in the *Device Family* drop-down list box. If you wish to implement the design logic in a specific device, select it in the *Devices* box. Otherwise, select *AUTO* to allow the MAX+PLUS II Compiler to choose the best device(s) in the current device family. If your design entry or synthesis and optimization tool required you to specify a target family and/or device, specify the same information in this dialog box. For information on partitioning logic among multiple devices, go to MAX+PLUS II Help. Choose **OK**.

5. Open the Compiler window by choosing the **Compiler** command (MAX+PLUS II menu). Go through the following steps to specify the options necessary to compile the design file(s) in your project:

    1. Ensure that all EDIF netlist files have the extension **.edf** and choose **EDIF Netlist Reader Settings** (Interfaces menu).

    2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor. This name should be the name of the vendor whose tool(s) you used to create the EDIF netlist files. If your vendor name does not appear, select *Custom* instead.

        If you are compiling a design created with Synopsys FPGA Express software, select *Synopsys*, choose the **Customize** button, enter <*project name*>**.lmf** in the *LMF #1* box, choose **OK**, and skip to step 6.

    3. If you selected an existing vendor name in the *Vendor* box and your project contains design files that require custom LMF mappings, choose the **Customize** button to expand the dialog box to show all settings. Turn on the *LMF #2* checkbox and type your custom LMF's filename in the corresponding text box, or select a name from the *Files* box. The selection in the *Vendor* box will change to *Custom* and all settings will be retained until you change them again.

    4. If you selected *Custom* in the *Vendor* box, choose the **Customize** button to expand the dialog box to show all settings. Any previously defined custom settings will be displayed. Under *Signal Names*, type one or more names with up to 20 total name characters in the *VCC* or *GND* box if your EDIF Input File(s) use one or more names other than `VCC` or `GND` for the global high or low signals. Multiple signal names must be separated by either a comma (,) or a space. Under *Library Mapping Files*, turn on the *LMF #1* checkbox and type a filename in the text box following it, or select a name from the *Files* box. If necessary, specify another LMF name in the *LMF #2* box. Go to MAX+PLUS II Help for detailed information on the settings available in the **EDIF Netlist Reader Settings** dialog box.

    5. Choose **OK**.

6. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, you may need to ensure that all files are present in your project directory, i.e., the same directory as the top-level design file. Otherwise, you must specify the directories containing these files as user libraries with the **User Libraries** command (Options menu).

7. Follow all guidelines that apply to your design entry or synthesis and optimization tool:

    - Exemplar Logic Galileo Extreme-Specific Compiler Settings
    - Synopsys DesignWare-Specific Compiler Settings
    - Converting Synopsys FPGA Compiler & Design Compiler Timing Constraints into MAX+PLUS II-Compatible Format with the **syn2acf** Utility
    - Synplicity Synplify-Specific Compiler Settings

8. If you wish to generate EDIF, VHDL, or Verilog HDL output files for post-compilation simulation or timing analysis with another EDA tool, go through the following steps:

1. (Optional) Turn on the **Optimize Timing SNF** command (Processing menu) to reduce the size of the output file(s). Turning on this command can reduce the size of output netlists by up to 30%.

   This command does not create optimized timing SNFs on UNIX workstations. However, a non-optimized timing SNF provides the same functional and timing information as an optimized timing SNF.

2. If you wish to generate EDIF Output Files (**.edo**), go through these steps:

   1. Turn on the **EDIF Netlist Writer** command (Interfaces menu). Then choose the **EDIF Netlist Writer Settings** command (Interfaces menu).

   2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor and choose **OK**. If your vendor name does not appear, select *Custom* instead and specify the settings that are appropriate for your simulation or timing analysis tool. Go to MAX+PLUS II Help for detailed information on the options available in the **EDIF Netlist Writer Settings** dialog box.

   3. To generate an optional Standard Delay Format (SDF) Output File (**.sdo**), choose the **Customize** button to expand the dialog box to show all settings. Select one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**.

   The filenames of the EDIF Output File(s) and optional SDF Output File(s) are the same as the user-defined chip name(s) for the project; if no chip names exist, the Compiler assigns filenames that are based on the project name. For a multi-device project, the Compiler also generates a top-level EDIF Output File that is uniquely identified by "**_t**" appended to the project name. In addition, the Compiler automatically generates a VHDL Memory Model Output File, *<project name>*.**vmo**, when it generates an EDIF Output File that contains memory (RAM or ROM).

3. If you wish to generate VHDL Output Files (**.vho**), turn on the **VHDL Netlist Writer** command (Interfaces menu). Then choose **VHDL Netlist Writer Settings** command (Interfaces menu). Select *VHDL Output File (.vho)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF ver. 2.1 files contain timing delay information that allows you to perform back-annotation simulation in VHDL with VITAL-compliant simulation libraries. The VHDL Output Files generated by the Compiler have the extension **.vho**, but are otherwise named in the same way as the EDIF Output Files described above.

4. If you wish to generate Verilog HDL Output Files (**.vo**), turn on the **Verilog Netlist Writer** command (Interfaces menu). Then choose **Verilog Netlist Writer Settings** command (Interfaces menu). Select *Verilog Output File (.vo)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF Output Files contain timing delay information that allows you to perform back-annotation simulation in Verilog HDL. The Verilog Output Files generated by the Compiler have the extension **.vo**, but are otherwise named in the same way as the EDIF Output Files described above.

9. To run the MAX+PLUS II Compiler, choose the **Project Save & Compile** command (File menu) or choose the **Start** button in the Compiler window.

   See step 3 for information on running MAX+PLUS II software from the command line.

10. Once you have compiled the project with the MAX+PLUS II Compiler, you can use the VHDL, Verilog HDL, or EDIF output file(s), and the optional SDF Output File(s) (**.sdo**) to perform timing analysis or timing simulation with another EDA tool. Specific instructions for some tools are described in these MAX+PLUS II ACCESS Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your EDA tool.

The MAX+PLUS II Compiler also generates a Report File (**.rpt**), a Pin-Out File (**.pin**), and one or more of the following files for device programming or configuration:

- JEDEC Files (**.jed**)
- Programmer Object Files (**.pof**)
- SRAM Object Files (**.sof**)
- Hexadecimal (Intel-format) Files (**.hex**)
- Tabular Text Files (**.ttf**)

Related Topics:

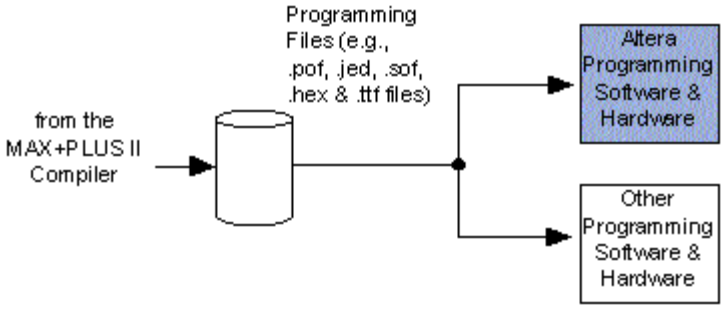
- Refer to the following sources for additional information:
  - Go to *Compiler Procedures* in MAX+PLUS II Help for information on other available Compiler settings.
  - Go to *Programmer Procedures* in MAX+PLUS II Help for instructions on creating other types of programming files and on programming or configuring Altera devices.
  - Go to Back-Annotating MAX+PLUS II Pin Assignments to Design Architect Symbols for information on back-annotating pin assignments in Mentor Graphics Design Architect schematics.
  - Go to Programming Altera Devices for information on the different programming hardware options for Altera device families.
- Go to the following topics, which are available on the web, for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware

Programming Altera Devices

Once you have successfully compiled and simulated a project with the MAX+PLUS® II software, you can program an Altera® device and test it in the target circuit. Figure 1 shows the device programming flow for MAX+PLUS II software.

Figure 1. MAX+PLUS II Device Programming Flow

Altera-provided items are shown in blue.



You can program devices with Altera programming hardware and MAX+PLUS II Programmer software installed on a 486- or Pentium-based PC or a UNIX workstation, or with programming hardware and software available from other manufacturers. Table 1 shows the available Altera programming hardware options on PCs and UNIX workstations.

Table 1. Altera Programming Hardware

Programming Hardware Option	PCs	UNIX Work-stations	MAX® 3000A Devices	Classic® & MAX 5000 Devices	MAX 7000 & MAX 7000E Devices	MAX 7000A, MAX 7000AE, MAX 7000B, MAX 7000S MAX 9000 & MAX 9000A Devices	FLEX® 6000, FLEX 6000A, FLEX 8000, FLEX 10K, FLEX 10KA, FLEX 10KB, & FLEX 10KE Devices	In-System Programming/ Configuration
Logic Programmer card, PL-MPU Master Programming Unit, and device-specific adapters	✓		✓	✓	✓	✓		
BitBlaster Download Cable	✓	✓	✓			✓	✓	✓
ByteBlasterMV Download Cable	✓		✓			✓	✓	✓
MasterBlaster Download Cable	✓	✓	✓			✓	✓	✓

If you wish to transfer programming files from a UNIX workstation to a PC over a network with File Transfer Protocol (FTP) or other similar transfer programs, be sure to select binary transfer mode.

Programming hardware from other manufacturers varies, but typically consists of a device connected to one of the serial ports on the workstation. Various vendors, such as Data I/O and BP Microsystems, supply hardware and software for programming Altera devices.

Related Topics:

- Go to Compiling Projects with MAX+PLUS II Software for information on creating programming files.
- Go to the following topics, which are available on the web, for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware
  - FLEX Devices
  - MAX Devices
  - Classic Device Family

Using Viewlogic Fusion/VCS & MAX+PLUS® II Software

The following topics describe how to use the Viewlogic Fusion/VCS software with MAX+PLUS® II software. Choose one of the following topics for information:

Open a printable version of all topics listed on this page.

Setting Up the MAX+PLUS II/Viewlogic Powerview Working Environment

- Software Requirements
- MAX+PLUS II/Viewlogic Powerview Interface File Organization
- Viewlogic Powerview **viewdraw.ini** Configuration File
- MAX+PLUS II/Viewlogic Powerview Project File Structure
- Altera-Provided Logic & Symbol Libraries
- The **vdpath** & **mega_lpm** Libraries

Simulation

- Project Simulation Flow
- Initializing Registers in VHDL & Verilog Output Files for Power-Up before Simulation
- Performing a Timing Simulation with Fusion/VCS for Powerview Software

Related Links

- Compiling Projects with MAX+PLUS II Software
- Programming Altera® Devices
- MAX+PLUS II Development Software
- Altera Programming Hardware
- Viewlogic web site (http://www.viewlogic.com)

Performing a Timing Simulation with Fusion/VCS for Powerview Software

After you have compiled a project with the MAX+PLUS® II software to generate a VHDL Output File (**.vho**) and a Standard Delay Format (SDF) Output File (**.sdo**), you can perform a timing simulation with Fusion/VCS software.

To simulate a project with Fusion/VCS software, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II /Viewlogic Powerview Working Environment.
2. Generate a VHDL Output File (**.vdo**) and an SDF ver 2.1 or 1.0 Output File (**.sdo**) for your project, as described in Compiling Projects with MAX+PLUS II Software.
3. Create a new **sim** directory under your **max2** directory to contain your Fusion/VCS simulation-related files.
4. To use the SDF Output File with the Fusion/VCS software, create a PLI table file (**.tab**) in the ***<project name>*/max2/sim** directory that contains the following line:

   `$sdf_annotate call=sdf_annotate_call acc=tchk, mp:`*<project name>*↵
5. Open the **Fusion/VCS** dialog box by choosing the **max2_VCS** button from the Altera® Design Tools Drawer in the Powerview Cockpit.
   1. Type *<project name>*`/max2/`*<project name>*`.vo` in the *Verilog Design and Object Files* box.
   2. Type *<project name>*`.tab` in the *PLI Table File* box.
   3. Type *<project name>*`/max2/alt_max2.vo` in the *Verilog Library File 1* box.
   4. (Optional) To use a command file or to set stimuli during simulation, select the *Debug* option in the *VCS* box and type the name of the command file in the *Simulation Command-file* box.

      ☞ When using a command file or setting stimuli, include the signal scope as part of the signal name. For example, to manipulate clk, a top-level signal in the **fadd** project, name the signal as `fadd.clk`.
   5. Choose **OK**.

Related Links:

- Go to Performing a Timing Simulation with ViewSim Software in these MAX+PLUS II ACCESS℠ Key topics for related information:

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

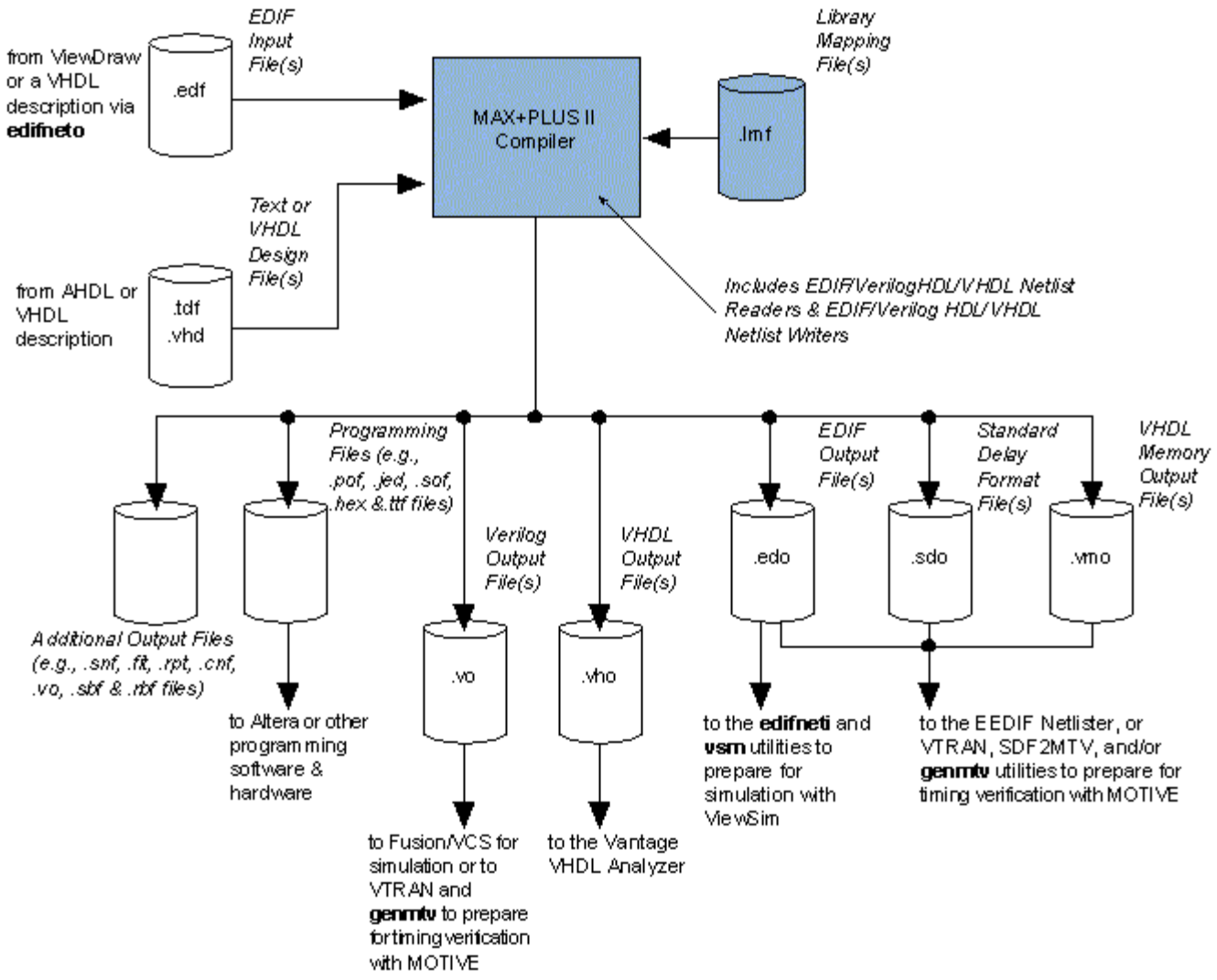
Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

MAX+PLUS II/Viewlogic Powerview Compilation Flow

Figure 1 shows the project compilation flow for the MAX+PLUS® II/Viewlogic Powerview interface.

Figure 1. MAX+PLUS II/Viewlogic Powerview Project Compilation Flow

Altera-provided items are shown in blue.



Feedback

Did this information help you?

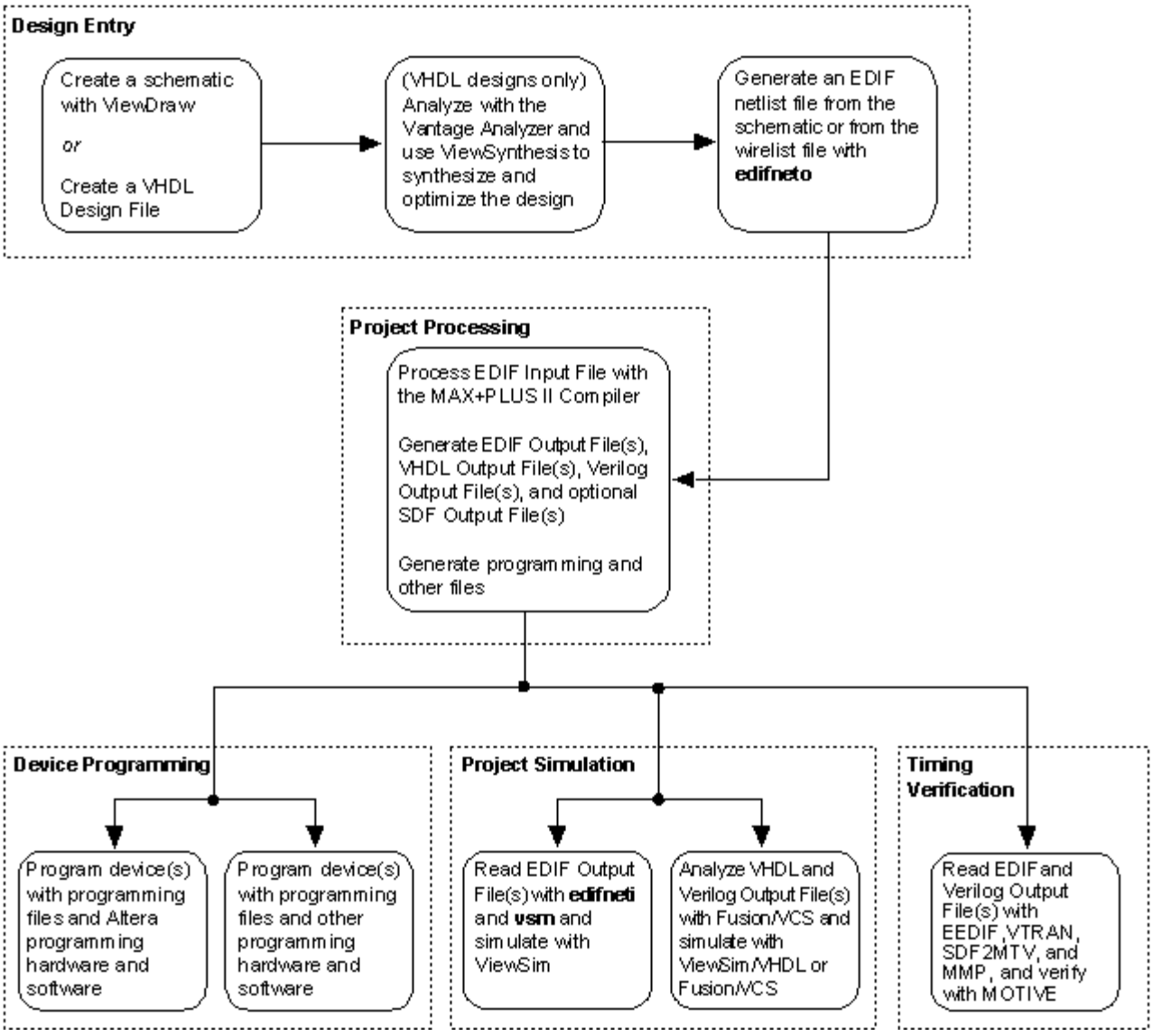
If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Design Flow for All Viewlogic Powerview Tools

Figure 1 shows the typical design flow for logic circuits created and processed with Viewlogic Powerview and MAX+PLUS® II software. Design Entry Flow, Project Compilation Flow, Project Simulation Flow, Timing Verification Flow, and Device Programming Flow show detailed diagrams for each stage of the design flow.

Figure 1. Viewlogic Powerview & MAX+PLUS II Design Flow



Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

MAX+PLUS II/Viewlogic Powerview Design Entry Flow

Figure 1 shows the design entry flow for the MAX+PLUS® II/Viewlogic Powerview interface.

Figure 1. MAX+PLUS II/Viewlogic Powerview Design Entry Flow

Altera-provided items are shown in blue.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

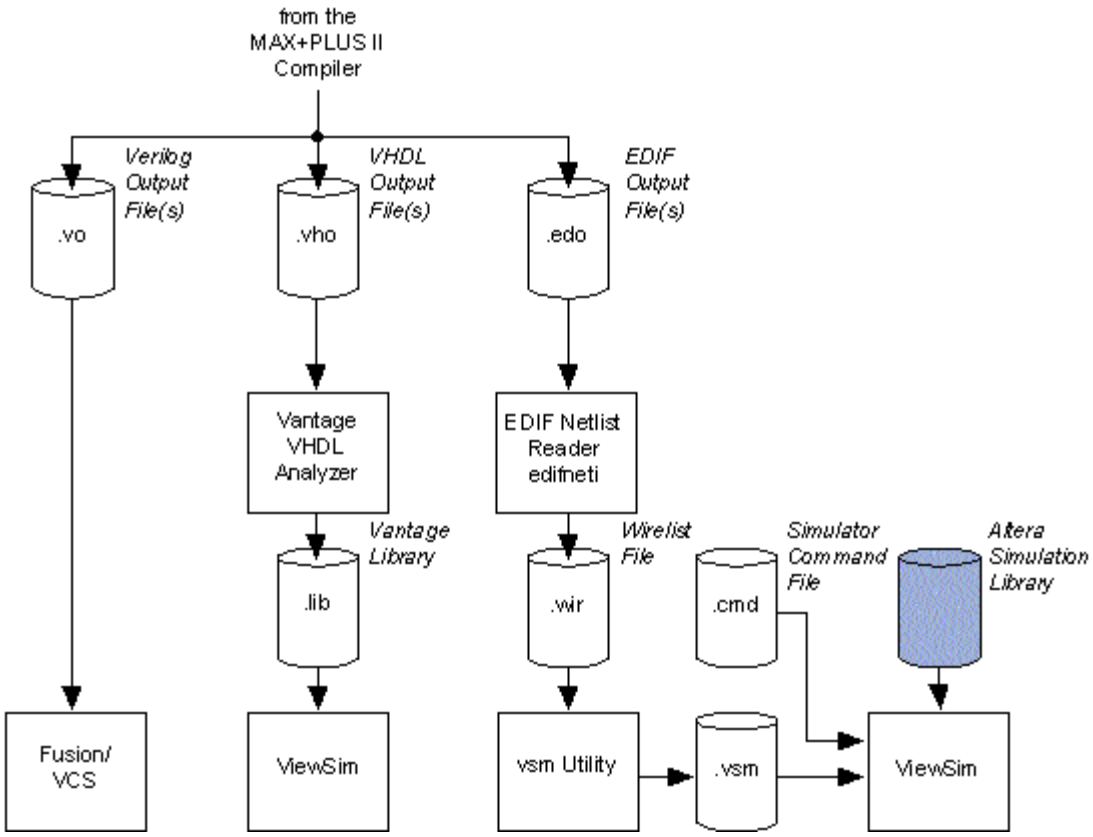
Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

MAX+PLUS II/Viewlogic Powerview Simulation Flow

Figure 1 shows the project simulation flow for the MAX+PLUS® II/Viewlogic Powerview interface.

Figure 1. MAX+PLUS II/Viewlogic Powerview Project Simulation Flow

Altera-provided items are shown in blue.



Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Using Viewlogic Powerview Tools with MAX+PLUS II Software

The following topics describe how to use Viewlogic Powerview tools as part of a complete design flow that includes the MAX+PLUS® II software. If you use only one Viewlogic Powerview tool, click List by Tool and select the tool name to view the list of topics only for that tool. Choose one of the following topics for information:

Open a printable version of all topics listed on this page.

Setting Up the MAX+PLUS II/Viewlogic Powerview Working Environment

- Software Requirements
- MAX+PLUS II/Viewlogic Powerview Interface File Organization
- Viewlogic Powerview **viewdraw.ini** Configuration File
- Viewlogic Powerview Graphical User Interface & the Altera Toolbox
- MAX+PLUS II/Viewlogic Powerview Project File Structure
- Altera-Provided Logic & Symbol Libraries
- The **vdpath** & **mega_lpm** Libraries

Design Flow for All Viewlogic Powerview Tools

Design Entry

- Design Entry Flow
- **ViewDraw**
  - Creating ViewDraw Schematics for Use with MAX+PLUS II Software
    - Instantiating LPM Functions in ViewDraw Schematics
    - Instantiating RAM & ROM Functions in Viewlogic Powerview Designs
  - Creating Hierarchical Projects in ViewDraw Schematics
  - Entering Resource Assignments
    - Assigning Pins, Logic Cells & Chips
    - Assigning Cliques
    - Assigning Logic Options
    - Modifying the Assignment & Configuration File with the **setacf** Utility
  - Performing a Functional Simulation with ViewSim Software
  - Converting ViewDraw Schematics or VHDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the **edifneto** Utility
- **VHDL**
  - Creating VHDL Designs for Use with MAX+PLUS II Software
    - Instantiating the clklock Megafunction in VHDL or Verilog HDL
    - Instantiating RAM & ROM Functions in Viewlogic Powerview Designs
  - Entering Resource Assignments
    - Modifying the Assignment & Configuration File with the **setacf** Utility
  - Analyzing VHDL Files with the Vantage VHDL Analyzer Software

    - Performing a Functional Simulation with ViewSim Software
    - Converting ViewDraw Schematics or VHDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the **edifneto** Utility

Synthesis & Optimization

- Synthesizing & Optimizing VHDL Designs with ViewSynthesis Software

Compilation

- Project Compilation Flow
- Compiling Projects with MAX+PLUS II Software
- Using the Max2 Express Drawer's **SCH <-> max2** Utility
- Using the Max2 Express Drawer's **VHDL <-> max2** Utility

Simulation

- Project Simulation Flow
- **ViewSim**
    - Initializing Registers in VHDL & Verilog Output Files for Power-Up before Simulation
    - Analyzing VHDL Files with the Vantage VHDL Analyzer Software
    - Performing a Functional Simulation with ViewSim Software
    - Performing a Timing Simulation with ViewSim Software
        - Using ViewDraw & ViewGen Software to Prepare for Multi-Device Board-Level Simulation with ViewSim Software
- **Fusion/VCS**
    - Performing a Timing Simulation with Fusion/VCS for Powerview Software

Timing Verification

- Timing Verification Flow
- Performing Timing Verification of EDIF Output Files (.edo) with MOTIVE & MOTIVE for Powerview Software
- Performing Timing Verification of Verilog Output Files (.vo) with MOTIVE Software

Device Programming

- Programming Altera® Devices

Related Links

- Powerview Command-Line Syntax
- MAX+PLUS II Development Software
- Altera Programming Hardware
- Viewlogic web site (http://www.viewlogic.com)

Using Viewlogic Powerview Tools with MAX+PLUS II Software

The following topics describe how to use Viewlogic Powerview tools as part of a complete design flow that includes the MAX+PLUS® II software. If you use only one Viewlogic Powerview tool, click List by Tool and select the tool name to view the list of topics only for that tool. Click on one of the following topics for information:

This file is suitable for printing only. It does not contain hypertext links that allow you to jump from topic to topic.

Setting Up the MAX+PLUS II/Viewlogic Powerview Working Environment

- Software Requirements
- MAX+PLUS II/Viewlogic Powerview Interface File Organization
- Viewlogic Powerview **viewdraw.ini** Configuration File
- Viewlogic Powerview Graphical User Interface & the Altera Toolbox
- MAX+PLUS II/Viewlogic Powerview Project File Structure
- Altera-Provided Logic & Symbol Libraries
- The **vdpath** & **mega_lpm** Libraries

Design Flow for All Viewlogic Powerview Tools

Design Entry

- Design Entry Flow
- **ViewDraw**
    - Creating ViewDraw Schematics for Use with MAX+PLUS II Software
        - Instantiating LPM Functions in ViewDraw Schematics
        - Instantiating RAM & ROM Functions in Viewlogic Powerview Designs
    - Creating Hierarchical Projects in ViewDraw Schematics
    - Entering Resource Assignments
        - Assigning Pins, Logic Cells & Chips
        - Assigning Cliques
        - Assigning Logic Options
        - Modifying the Assignment & Configuration File with the **setacf** Utility
    - Performing a Functional Simulation with ViewSim Software
    - Converting ViewDraw Schematics or VHDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the **edifneto** Utility
- **VHDL**

Creating VHDL Designs for Use with MAX+PLUS II Software
    - Instantiating the clklock Megafunction in VHDL or Verilog HDL
    - Instantiating RAM & ROM Functions in Viewlogic Powerview Designs
  - Entering Resource Assignments
    - Modifying the Assignment & Configuration File with the **setacf** Utility
  - Analyzing VHDL Files with the Vantage VHDL Analyzer Software
  - Performing a Functional Simulation with ViewSim Software
  - Converting ViewDraw Schematics or VHDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the **edifneto** Utility

Synthesis & Optimization

- Synthesizing & Optimizing VHDL Designs with ViewSynthesis Software

Compilation

- Project Compilation Flow
- Compiling Projects with MAX+PLUS II Software
- Using the Max2 Express Drawer's **SCH <-> max2** Utility
- Using the Max2 Express Drawer's **VHDL <-> max2** Utility

Simulation

- Project Simulation Flow
- **ViewSim**
  - Initializing Registers in VHDL & Verilog Output Files for Power-Up before Simulation
  - Analyzing VHDL Files with the Vantage VHDL Analyzer Software
  - Performing a Functional Simulation with ViewSim Software
  - Performing a Timing Simulation with ViewSim Software
    - Using ViewDraw & ViewGen Software to Prepare for Multi-Device Board-Level Simulation with ViewSim Software
- **Fusion/VCS**
  - Performing a Timing Simulation with Fusion/VCS for Powerview Software

Timing Verification

- Timing Verification Flow
- Performing Timing Verification of EDIF Output Files (.edo) with MOTIVE & MOTIVE for Powerview Software
- Performing Timing Verification of Verilog Output Files (.vo) with MOTIVE Software

Device Programming

- Programming Altera® Devices

Related Topics:

- Go to Powerview Command-Line Syntax in these MAX+PLUS II ACCESS℠ Key topics for related information.
- Go to the following topics, which are available on the web, for additional information:

- MAX+PLUS II Development Software
- Altera Programming Hardware
- Viewlogic web site (http://www.viewlogic.com)

Setting Up the MAX+PLUS II/Viewlogic Powerview Working Environment

To use the MAX+PLUS® II software with Viewlogic's Powerview software, you must install the MAX+PLUS II software, familiarize yourself with the Altera® Toolbox in the Powerview Cockpit, and then establish an environment that facilitates entering and processing designs. The MAX+PLUS II /Viewlogic Powerview interface is installed automatically when you install the MAX+PLUS II software on your workstation.

To set up your working environment for the MAX+PLUS II/Viewlogic Powerview interface, follow these steps:

1. Ensure that you have correctly installed the MAX+PLUS II and Viewlogic software versions described in MAX+PLUS II/Viewlogic Powerview Software Requirements.

2. Add the following environment variable to your **.cshrc** file to specify **/usr/maxplus2** as the MAX+PLUS II system directory:

   ```
   setenv ALT_HOME /usr/maxplus2 ↵
   ```

3. Add the **$ALT_HOME/viewlogic/standard**, **$ALT_HOME/bin**, and **$ALT_HOME/viewlogic/bin** directories to the `PATH` environment variable in your **.cshrc** file.

4. Add the **$ALT_HOME/viewlogic/standard** directory to the `WDIR` environment variable in your **.cshrc** file using the following syntax:

   ```
   setenv WDIR $ALT_HOME/viewlogic/standard:/<Powerview system directory>/standard ↵
   ```

   ☞ Make sure the **$ALT_HOME/viewlogic/standard** directory is the first directory in your `WDIR` path.

5. Source your **.cshrc** file by typing `source .cshrc ↵` at the UNIX prompt.

6. Create the Viewlogic Powerview **viewdraw.ini** configuration file.

7. Copy the **/usr/maxplus2/maxplus2.ini** file to your **$HOME** directory:

   ```
   cp /usr/maxplus2/maxplus2.ini $HOME ↵

   chmod u+w $HOME/maxplus2.ini ↵
   ```

   The **maxplus2.ini** file contains both Altera- and user-specified initialization parameters that control the MAX+PLUS II software, such as MAX+PLUS II symbol and logic function library paths and the current project name. The MAX+PLUS II installation procedure creates and copies the **maxplus2.ini** file to the **/usr/maxplus2** directory.

   ☞ Normally, you do not have to edit your local copy of **maxplus2.ini**, because the MAX+PLUS II software updates the file automatically whenever you change any parameters or settings. However, if you move the **max2lib** and **max2inc** library subdirectories, you must update the file. Go to "Creating & Using a Local Copy of the **maxplus2.ini** File" in MAX+PLUS II Help for more information.

8. If you plan to instantiate Library of Parameterized Modules (LPM) functions in ViewDraw schematics, you must create a new file with the name **vdraw.vs**. The **vdraw.vs** file must include the following line:

```
load ("vdpath")
```

You must also make sure that you specify the **vdraw.vs** file in your `WDIR` path.

9. Set up a directory structure that facilitates working with the MAX+PLUS II/Viewlogic Powerview interface. Refer to MAX+PLUS II/Viewlogic Powerview Project File Structure.

Related Topics:

- Go to *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual for more information on installation and details on the directories that are created during MAX+PLUS II installation. Go to MAX+PLUS II/Viewlogic Powerview Interface File Organization for information about the MAX+PLUS II/Viewlogic Powerview directories that are created during MAX+PLUS II installation.
- Go to the following topics, which are available on the web, for additional information:
  - ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
  - This manual is also available in 4 parts:
    - Preface & Section 1: MAX+PLUS II Installation
    - Section 2: MAX+PLUS II - A Perspective
    - Section 3: MAX+PLUS II Tutorial
    - Appendices, Glossary & Index

MAX+PLUS II/Viewlogic Powerview Software Requirements

The following applications and utilities are used to generate, process, synthesize, and verify a project with MAX+PLUS® II and Viewlogic Powerview software.

	Viewlogic	**Altera**
ViewDraw	ViewGen	MAX+PLUS II version 9.4
VHDL Analyzer	ViewPath (optional)	
Vantage VHDL Analyzer	ViewTrace	
VHDL -> sym	ViewData Path	
edifneto	MOTIVE version 5.1.6 *Note (1)*	
edifneti	MOTIVE for Powerview version 3.2.1 (optional) *Note (1)*	
EEDIF (optional)	SDF2MTV (optional)	
MMP (optional)	Fusion/VCS	
vsm		

Note:

(1) MOTIVE for Powerview, a wrapper application for MOTIVE, provides a graphical user interface for the utilities (i.e., EEDIF, SDF2MTV, and MMP) used during a static timing verification with MOTIVE. MOTIVE alone does not accept EDIF files through the Setup Advisor.

☞ The MAX+PLUS II **read.me** file provides up-to-date information on which versions of Viewlogic Powerview applications the current version of the MAX+PLUS II software supports. It also provides information on installation and operating requirements. You should read the **read.me** file on the CD-ROM before installing the MAX+PLUS II software. After installation, you can open the **read.me** file from the MAX+PLUS II Help menu.

MAX+PLUS II/Viewlogic Powerview Interface File Organization

Table 1 shows the MAX+PLUS® II/Viewlogic Powerview interface subdirectories that are created in the

MAX+PLUS II system directory (by default, the **/usr/maxplus2** directory) during MAX+PLUS II installation.

☞ For information on the other directories that are created during MAX+PLUS II installation, see "MAX+PLUS II File Organization" in *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual.

Table 1. MAX+PLUS II Directory Organization

Directory	Description
./lmf	Contains the Altera-provided Library Mapping File, **vwlogic.lmf**, that maps Viewlogic logic functions to equivalent MAX+PLUS II logic functions.
./viewlogic	Contains the **alt_edif.cfg** EDIF configuration file that is used with the **edifneti** utility. Also contains the library and sample subdirectories.
./viewlogic/examples	Contains the sample Viewlogic designs.
./viewlogic/library/max2sim	Contains the MAX+PLUS II simulation model library (**max2_sim**) for use in ViewSim software.
./viewlogic/library/alt_max2	Contains MAX+PLUS II primitives (`EXP`, `GLOBAL`, `LCELL`, `SOFT`, `CARRY`, `CASCADE`, `DFFE`, `DFFE6K`, and `OPNDRN`), macrofunctions (`a_8fadd`, `a_8mcomp`, `a_8count`, `a_81mux`), and megafunctions (`clklock`) for use in ViewDraw schematics. These logic functions support specific architectural features of Altera® devices. The **alt_max2** library also contains modified versions of the ViewDraw primitives that use tri-state buffers, because these primitives require special handling in the MAX+PLUS II /Viewlogic Powerview interface.
./viewlogic/library/synlib	Contains the Altera-provided synthesis library **altera**, which includes MAX+PLUS II primitives, the **altera.sml** file, a **sym** directory, and a **wir** directory for use with ViewSynthesis software.
./viewlogic/library/alt_mf	Contains the VHDL models for the MAX+PLUS II primitives (`EXP`, `GLOBAL`, `LCELL`, `SOFT`, `CARRY`, `CASCADE`, `DFFE`, and `OPNDRN`), macrofunctions (`clklock`) for use with ViewSynthesis software, the Vantage VHDL Analyzer software, and the VHDL source files. These logic functions are used to maintain portability to other architectures.
./viewlogic/library/alt_time	Contains MOTIVE timing models for MAX+PLUS II logic functions (**motive.lib**), including the `clklock` megafunction, and MAX+PLUS II driver models (**motive.drv**).
./viewlogic/library/alt_vtl	Contains the VHDL source files for the VITAL 3.0-compliant library. This library is available for ViewSim software.
./viewlogic/bin	Contains all MAX+PLUS II, Viewlogic, and interface-related scripts.
./viewlogic/standard	Contains all standard **.ini** files and standard tools.

Related Topics:

- Go to the following topics, which are available on the web, for additional information:
  - ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
  - This manual is also available in 4 parts:
    - Preface & Section 1: MAX+PLUS II Installation
    - Section 2: MAX+PLUS II - A Perspective
    - Section 3: MAX+PLUS II Tutorial
    - Appendices, Glossary & Index

Viewlogic Powerview viewdraw.ini Configuration File

Each Powerview project is configured with the **viewdraw.ini** file that resides in the project directory. The `DIR` statements at the end of **viewdraw.ini** are paths to library directories that are used by the various Powerview applications. Figure 1 shows a sample of the `DIR` statements that are required to use the libraries.

Figure 1. Excerpt from viewdraw.ini

```
DIR [pw] .
DIR [r]  /usr/maxplus2/vwlogic/library/alt_max2 (alt_max2)
DIR [r]  /usr/maxplus2/vwlogic/library/max2sim (max2_sim)
DIR [r]  /usr/maxplus2/vwlogic/library/synlib (altera)
DIR [r]  /usr/maxplus2/vwlogic/library/alt_mf (alt_mf)
DIR [r]  /usr/maxplus2/vwlogic/library/alt_vtl (alt_vtl)
DIR [rm] /<Powerview system directory>/lib/builtin (builtin)
DIR [rm] /<Powerview system directory>/simmods/vl/dip/74ls (vl74ls)
DIR [rm] /<Powerview system directory>/symsets/vl/dip/74ls (vl74ls)
DIR [r]  /<Powerview system directory>/lib/vdpath (vdpath)
```

☞ When you add the libraries to the **/usr/maxplus2/vwlogic/standard/viewdraw.ini** file, they are automatically set when you create a new project. Powerview tools search these libraries sequentially, so it is important to add them in the order in which they are listed in Figure 1.

Table 1 shows the libraries that must be specified in the `DIR` statements in the **viewdraw.ini** file.

Table 1. Powerview Application Libraries

Library	Library Alias	Source	Topics
alt_max2	`alt_max2`	Altera	Graphical elements for ViewDraw
max2sim	`max2_sim`	Altera	Models for project simulation
synlib	`altera`	Altera	VHDL synthesis library for the MAX+PLUS ® II software
alt_mf	`alt_mf`	Altera	VHDL models of MAX+PLUS II logic functions
alt_vtl	`alt_vtl`	Altera	VITAL-compliant primitives
builtin	`builtin`	Altera	Basic primitives such as `INPUT` pins, `OUTPUT` pins, `AND` gates, `OR` gates, etc.
74ls	`vl74ls`	Viewlogic	74-series macrofunctions
vdpath	`vdpath`	Viewlogic	Standard library of parameterized modules (LPM) functions

☞ The Altera-provided libraries must be listed before the Viewlogic-provided libraries in the **viewdraw.ini** file to ensure that the correct versions of the megafunctions, macrofunctions, and primitives are used.

Related Topics:

- Go to Altera-Provided Logic & Symbol Libraries for more information on Altera-supplied libraries. Refer to the Powerview documentation for more information on setting up the **viewdraw.ini** file.

Viewlogic Powerview Graphical User Interface & the Altera Toolbox

You use the Powerview graphical interface manager, the Cockpit, and the Altera® Toolbox to start all Powerview and Altera tools. Within the Altera Toolbox, you can specify the Max2 Express Drawer or the Design Tools Drawer to work with the Altera/Viewlogic Powerview interface.

The Max2 Express Drawer provides a quick and seamless way to transfer designs created in Powerview to the MAX+PLUS® II software for compilation, then return the compiled designs to Powerview for simulation and timing verification. Table 1 describes the Max2 Express Drawer tools.

Table 1. Max2 Express Drawer Tools

Tool	Description
max2_VDraw	Launches the Powerview ViewDraw schematic entry tool.
VHDL<->max2	Launches all tools necessary to synthesize a VHDL design, compile for an Altera device, and generate a **.vsm** file for simulation with the Powerview ViewSim simulator.
SCH<->max2	Launches all tools necessary to compile a schematic design entered with Powerview ViewDraw software for an Altera device and to generate a **.vsm** file for simulation with Powerview ViewSim and **.edo**, **.sdo**, and **.vmo** files for timing analysis with MOTIVE for Powerview.
max2_VSim	Launches the Powerview ViewSim simulator.
max2_VTrace	Launches the Powerview ViewTrace simulation waveform editor.
max2_MOTIVE	Launches the MOTIVE for Powerview ViewDraw static timing verification tool.

The Design Tools Drawer provides tools that enable you to create a design with the Powerview tools, compile the design in the MAX+PLUS II software, and simulate and verify the design with Powerview software. Table 2 describes the Design Tools Drawer tools.

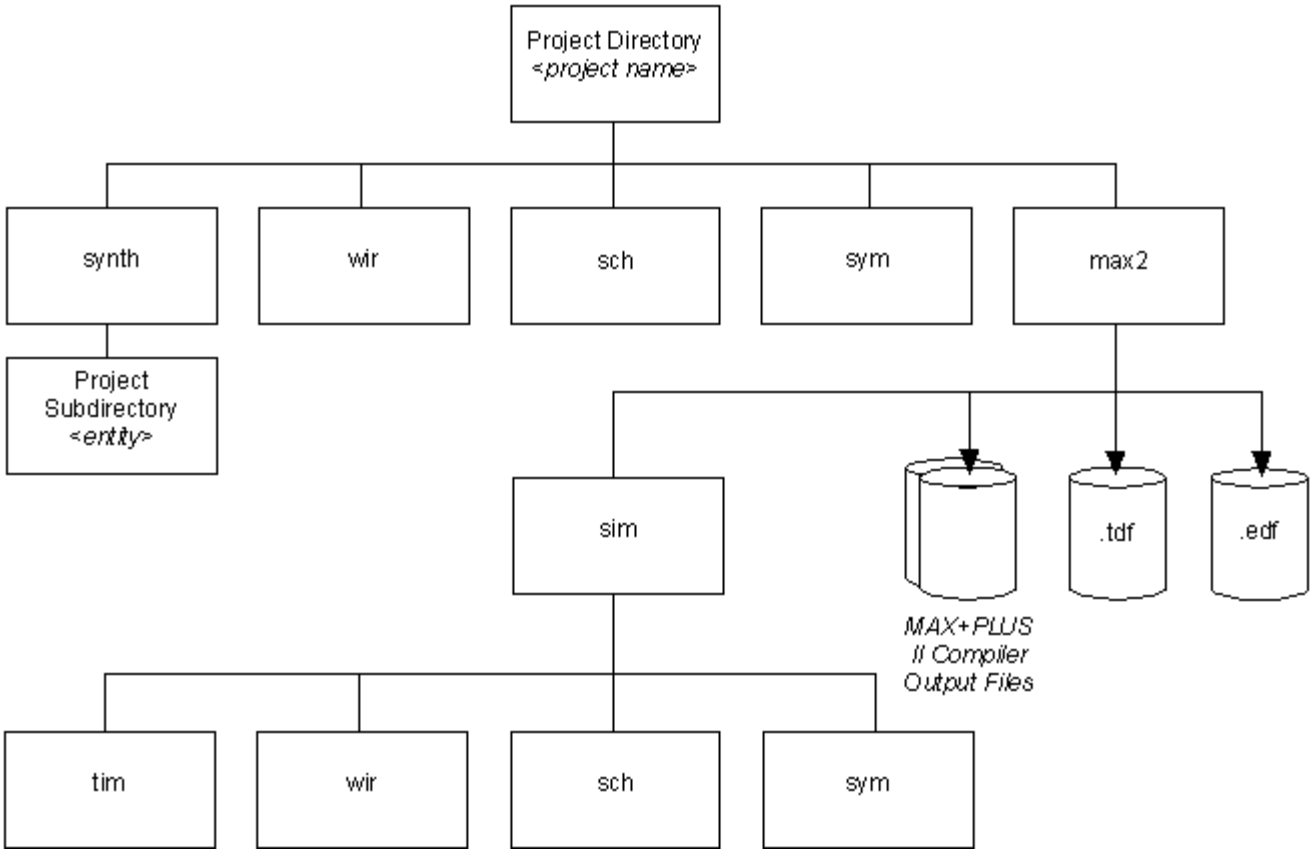
Table 2. Design Tools Drawer Tools

Tool	Description
max2_VDraw	Launches the Powerview ViewDraw schematic entry tool.
max2_analyzer	Launches the Powerview VHDL Analyzer software.
max2_syn	Launches the Powerview VHDL synthesis tool.
max2_chk	Launches the Powerview schematic verification tool.
max2_vsmnet	Launches the Powerview **vsm** utility that converts a wirelist file into a **.vsm** file.
max2_VSim	Launches the Powerview ViewSim simulator.
max2_VTrace	Launches the Powerview ViewTrace simulator.
max2_edifo	Launches the Powerview EDIF netlist writer, **edifneto**.
max2_VGen	Launches the Powerview ViewGen utility that generates a schematic from a wirelist file.
max2	Launches the MAX+PLUS II Compiler.
max2_edifi	Launches the Powerview EDIF Netlist Reader, **edifneti**.
max2_vhdl2sym	Launches the Powerview **vhdl2sym** utility that generates a symbol from a VHDL file.
max2_VantgMgr	Launches the Powerview Vantage VHDL Library Manager tool.
max2_VantgAnlz	Launches the Vantage VHDL Analyzer software.
max2_VCS	Launches the Fusion/VCS Simulator.
max2_MOTIVE	Launches the MOTIVE for Powerview static timing verification tool.

MAX+PLUS II/Viewlogic Powerview Project File Structure

In the MAX+PLUS® II software, a project name is the name of a top-level design file, without the filename extension. This design file can be an EDIF, Verilog HDL, or VHDL netlist file; an Altera® Hardware Description Language (AHDL) TDF; or any other MAX+PLUS II- supported design file. The EDIF netlist file must be created by Powerview and imported into the MAX+PLUS II software as an EDIF Input File (**.edf**). Figure 1 shows an example of MAX+PLUS II project directory structure that includes Powerview-generated files.

Figure 1. Sample MAX+PLUS II Project Organization



The MAX+PLUS II software stores the connectivity data on the links between design files in a hierarchical project in a Hierarchy Interconnect File (**.hif**), but refers to the entire project only by its project name. The MAX+PLUS II Compiler uses the HIF to build a single, fully flattened project database that integrates all the design files in a project hierarchy.

Unlike Powerview, the MAX+PLUS II software does not automatically create a project directory when you create a project. A single directory can contain several MAX+PLUS II design files, and you can specify any one of the designs in the directory as a project in the MAX+PLUS II software.

Viewlogic Powerview Local Work Area Structure

When you create a project with the Powerview Cockpit's **Create** command (Project menu), the project directory is created. You should generate design files and functional simulation files under this directory. A **max2** subdirectory is automatically created under your current project directory when you generate an EDIF file from your schematic or VHDL file. The ***<project name>*.edf** file is stored in the **max2** subdirectory. All MAX+PLUS® II Compiler output files are created in the **/*<project name>*/max2** subdirectory.

☞ ViewDraw files are identified by their directories and not by their extensions, so it is easy to overwrite files unintentionally. To avoid overwriting files, Altera recommends that you create a new project directory, ***<project name>*/max2/sim**, where you can generate all the files needed for simulation.

ViewDraw Project File Structure

Each ViewDraw project directory contains three subdirectories: **wir**, **sch**, and **sym**. See Table 1.

Table 1. ViewDraw Subdirectories

Directory	Topics

./wir	Wirelist files that contain connectivity information for a particular logic block
./sch	Schematics that contain logic
./sym	Symbol files that are the ViewDraw graphical representation of the logic blocks

Each file type uses the filename extension **.1**. Different file types are distinguished only by their directory: **/lib/wir/**<*project name*>**.1** is a wirelist file; **/lib/sch/**<*project name*>**.1** is the corresponding schematic file; and **/lib/sym/**<*project name*>**.1** is the corresponding symbol.

VHDL Project File Structure

Each VHDL project directory contains three subdirectories. See Table 2.

Table 2. VHDL Subdirectories

Directory	Topics
./synth	All synthesis-related files and directories
./synth/<*entity*>	Four types of files: <*entity*>**.pdf**, <*entity*>**.opt**, <*entity*>**.sta**, and <*entity*>**.gnl**
./wir	Wirelist for synthesized VHDL modules

☞ For each VHDL entity in the design, there is a corresponding **./synth/<*entity*>** directory.

Altera-Provided Logic & Symbol Libraries

The MAX+PLUS® II/Viewlogic Powerview environment provides libraries for compiling, synthesizing, and simulating designs.

☞ You can create your own libraries of custom symbols and logic functions for use in ViewDraw schematics and VHDL design files. You can use custom symbols (and functions) to incorporate an EDIF Input File, TDF, or any other MAX+PLUS II-supported design file into a project. The MAX+PLUS II software uses the **vwlogic.lmf** Library Mapping File to map ViewDraw symbols to equivalent MAX+PLUS II megafunctions, macrofunctions, or primitives. To use custom symbols and functions, you can create a custom LMF that maps your custom functions to equivalent EDIF Input Files, TDFs, or other MAX+PLUS II-supported design files. Go to "Library Mapping File" and "Viewlogic Library Mapping File" in MAX+PLUS II Help for more information.

Logic symbols used in ViewDraw software are available from the MAX+PLUS II **alt_max2** library, the ViewDraw **builtin** and **74ls** libraries, and the ViewDatapath **vdpath** library. VHDL models of MAX+PLUS II logic functions are available from the Altera-provided **alt_mf** library.

The alt_max2 Library

The **alt_max2** library provides MAX+PLUS II-specific logic functions that can be used to take advantage of special architectural features in each Altera® device family. See Table 1. Symbols and functional simulation models are available for all of these elements.

The alt_mf Library

The Altera-provided **alt_mf** library, which supports the Viewlogic Vantage VHDL Analyzer software, contains VHDL simulation models for all logic functions listed in the following table. The library is configured so that these functions pass untouched through the EDIF netlist file to the MAX+PLUS II Compiler, providing you with optimal control over design processing. Altera also provides models for all of the logic functions that you can synthesize and simulate. These models allow you to perform functional VHDL simulation while maintaining an architecture-independent VHDL description.

Table 1. Architecture Control Logic Functions

Name *Note (1), Note (2)*	Description	Name	Description	Name	Description
8fadd	8-bit full adder macrofunction	LCELL	Logic cell buffer primitive	EXP	MAX® 5000, MAX 7000, and MAX 9000 Expander buffer primitive
8mcomp	8-bit magnitude comparator macrofunction	GLOBAL	Global input buffer primitive	SOFT	Soft buffer primitive
8count	8-bit up/down counter macrofunction	CASCADE	FLEX® 6000, FLEX 8000, and FLEX 10K cascade buffer primitive	OPNDRN	Open-drain buffer primitive
81mux	8-to-1 multiplexer macrofunction	CARRY	FLEX 6000, FLEX 8000, and FLEX 10K cascade buffer primitive	DFFE *Note (2)*	D-type flipflop with Clock Enable primitive
clklock	Phase-locked loop megafunction				

Notes:

1. Logic function names that begin with a number must be prefixed with "a_" in VHDL designs. For example, 8fadd must be specified as a_8fadd.
2. For designs that are targeted to FLEX 6000 devices, you should use the DFFE primitive only if the design contains either a Clear or Preset signal, but not both. If your design contains both a Clear and a Preset signal, you must use the DFFE6K primitive.

Related Topics:

- Choose **Old-Style Macrofunctions**, **Primitives**, or **Megafunctions/LPM** from the MAX+PLUS II Help menu for detailed information on these functions.
- Go to the following topics, which are available on the web, for additional information:
  - FLEX Devices
  - MAX Devices
  - Classic Device Family

The vdpath & mega_lpm Libraries

The library of parameterized modules (LPM) 2.1.0 standard defines a set of parameterized functions and their corresponding representations in an EDIF netlist file. These logic functions allow you to create and functionally simulate an LPM-based design without targeting a specific device family. After the design is completed, you can target the design to any device family.

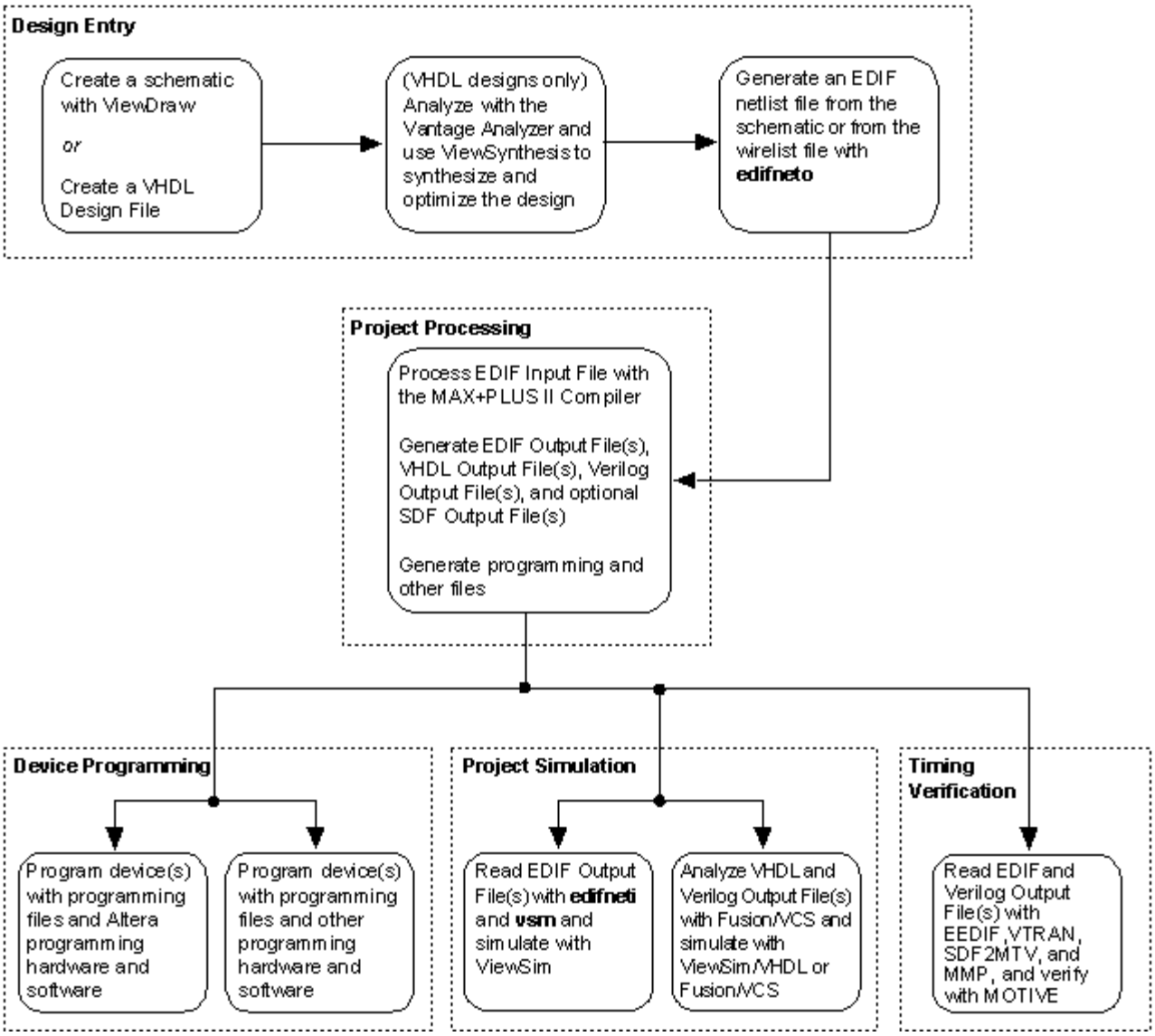
When the MAX+PLUS® II software processes projects that include Viewlogic-provided **vdpath** LPM functions, it uses functions from the Altera-provided **mega_lpm** library. This library includes all standard LPM functions except the truth table, finite state machine, and pad functions. Altera does not directly support the lpm_ram_dq, lpm_ram_io, and lpm_rom functions. Refer to Instantiating RAM & ROM Functions in Viewlogic Powerview Designs for instructions on instantiating RAM and ROM functions.

☞ Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu for more information about LPM functions.

Design Flow for All Viewlogic Powerview Tools

Figure 1 shows the typical design flow for logic circuits created and processed with Viewlogic Powerview and MAX+PLUS® II software. Design Entry Flow, Project Compilation Flow, Project Simulation Flow, Timing Verification Flow, and Device Programming Flow show detailed diagrams for each stage of the design flow.

Figure 1. Viewlogic Powerview & MAX+PLUS II Design Flow



MAX+PLUS II/Viewlogic Powerview Design Entry Flow

Figure 1 shows the design entry flow for the MAX+PLUS® II/Viewlogic Powerview interface.

Figure 1. MAX+PLUS II/Viewlogic Powerview Design Entry Flow

Altera-provided items are shown in blue.

Creating ViewDraw Schematics for Use with MAX+PLUS II Software

You can create ViewDraw schematics and convert them into EDIF Input Files (**.edf**) that can be processed with the MAX+PLUS® II software.

To create a ViewDraw schematic for use with the MAX+PLUS II software, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II /Viewlogic Powerview Working Environment.
2. Start Powerview by typing `powerview` ↵ at a UNIX prompt.
3. In the Cockpit window, select *Altera* in the *Current ToolBox* drop-down list box, and select the drawer you want to use, i.e., *Design Tools* or *Max2 Express*, in the *Current Drawer* drop-down list box.
4. Choose **Create** (Project menu) from your working directory to create your project directory. Choose **OK**.
5. Choose **SearchOrder** (Project menu) to add the appropriate library directories and aliases to your **viewdraw.ini** file in the appropriate search order. Refer to Viewlogic Powerview **viewdraw.ini** Configuration File for more information on Powerview application libraries.
6. Start ViewDraw by double-clicking Button 1 on the **max2_VDraw** icon in the drawer that you selected in step 3, type the name of the schematic, and choose **OK**. You can also start the ViewDraw software by typing `viewdraw` ↵ at the UNIX prompt.
7. Choose **Comp** (Add menu) to add components to the schematic. You can use functions from the **alt_max2**, **builtin**, and **74ls** libraries. For information on Altera-provided libraries, go to Altera-Provided Logic & Symbol Libraries.

   Instructions for instantiating specific functions are provided in the following MAX+PLUS II ACCESS℠ Key topics:

   - Instantiating LPM Functions in ViewDraw Schematics
   - Instantiating RAM & ROM Functions in Viewlogic Powerview Designs

   ☞ You can instantiate MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP℠). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.
8. If you instantiate a `clklock` megafunction, choose the **Dialog** command (Attr menu), then choose the **All** command (Dialog menu) to specify the values of the `INPUT_FREQUENCY` and `CLOCKBOOST` parameters. For detailed information on the `clklock` megafunction, choose **Megafunctions/LPM** from the MAX+PLUS II Help menu.
9. If you wish to create a hierarchical design that contains symbols representing other design files, such as Altera® Hardware Description Language (AHDL) Text Design Files (**.tdf**), go to Creating Hierarchical Projects in ViewDraw Schematics.
10. Choose **Net** (Add menu) to add nets to the schematic.
11. Choose **Bus** (Add menu) to add buses to the schematic.

12. Choose **Label** (Add menu) to attach labels to nets and buses. When you are naming and labeling buses, make sure you use the format <*bus name*>[<*most significant bit*>:<*least significant bit*>], and that you label both the net and the pin.

13. (Optional) To enter resource assignments in your schematic, select a symbol or a net that feeds an output and use the **Attr** command (Add menu) to add the assignments. For more information, go to Entering Resource Assignments. You can also enter resource assignments from the MAX+PLUS II software.

14. Choose **Write** (File menu) to check and save both the schematic with the name **.sch/<*design name*>.1** and the wirelist with the name **./wir/<*design name*>.1**.

15. (Optional) Perform a functional simulation, as described in Performing a Functional Simulation with ViewSim Software.

16. Once you have created a schematic, you can generate an EDIF netlist file that can be imported into the MAX+PLUS II software with either of the following methods:

    - You can create an EDIF netlist file, as described in Converting ViewDraw Schematics or VHDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the **edifneto** Utility You must use this method if your ViewDraw schematic instantiates Library of Parameterized Modules (LPM) functions.
    - You can use the **SCH <-> max2** utility in the Max2 Express drawer to automatically create an EDIF netlist file, compile it with the MAX+PLUS II Compiler, generate an EDIF Output File (**.edo**), and generate a **.vsm** file for simulation, as described in Using the Max2 Express Drawer's **SCH <-> max2** Utility.

Installing the Altera-provided MAX+PLUS II/Viewlogic interface on your computer automatically creates the following sample ViewDraw schematic files:

- **/usr/maxplus2/examples/viewlogic/example1/fadd**
- **/usr/maxplus2/examples/viewlogic/example3/fadd2**
- **/usr/maxplus2/examples/viewlogic/example4/fadd2mpp**
- **/usr/maxplus2/examples/viewlogic/example7/fifo**

Related Topics:

- Go to Powerview Command-Line Syntax in these MAX+PLUS II ACCESS Key topics for related information.

Instantiating LPM Functions in ViewDraw Schematics

You can instantiate library of parameterized modules (LPM) functions from the **vdpath** library in ViewDraw schematics.

☞ Altera does not directly support the `lpm_ram_dq`, `lpm_ram_io`, and `lpm_rom` functions. Go to Instantiating RAM & ROM Functions in Viewlogic Powerview Designs for information on instantiating these functions.

To instantiate an LPM function, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Viewlogic Powerview Working Environment. Make sure that you have created a **vdraw.vs** file, as described in step 8 of that topic.

2. Choose **Cell** (Add menu).

3. Choose an <*LPM function name*> to open the <*LPM function name*> dialog box. Specify a symbol name for *Symbol Prefix* and specify appropriate parameters. Choose **OK**.

   The ViewDraw software generates the specified symbol name symbol according to your specifications. It also generates a corresponding VHDL simulation model, but it is compiled only after you save the schematic. If you want to change the settings for the symbol, select the instance and choose **Cell** (Change menu) to re-open the appropriate dialog box.

4. Continue with the steps necessary to complete your schematic, as described in Creating ViewDraw Schematics for Use with MAX+PLUS II Software.

Related Topics:

- Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu for detailed information on LPM functions.

Instantiating RAM & ROM Functions in Viewlogic Powerview Designs

The MAX+PLUS® II /Viewlogic Powerview interface offers full support for the memory capabilities of the FLEX® 10K device family, including synchronous and asynchronous RAM and ROM, cycle-shared dual-port RAM, dual-port RAM, single-Clock FIFO, and dual-Clock FIFO functions. You can use the Altera-provided **genmem** utility to generate functional simulation models and timing models for these functions. Type `genmem` ↵ at the UNIX prompt to display information on how to use this utility, as well as a list of the functions you can generate. RAM and ROM can be instantiated in both ViewDraw schematics and VHDL designs.

☞ Refer to Viewlogic documentation for information on simulating projects that contain RAM functions. The procedure for reading an EDIF Output File and preparing it for simulation with ViewSim requires additional steps when the project contains RAM functions.

When you instantiate a RAM or ROM function, follow these general guidelines:

- For ROM functions, you must specify an initial memory content file in the Intel hexadecimal format (**.hex**) or the Altera® Memory Initialization File (**.mif**) format. The filename must be the same as the instance name; e.g., the instance name must be unique throughout the whole project, and must contain only valid name characters. The initialization file must reside in the directory containing the project's design files.
- For RAM functions, specifying a memory initialization file is optional.
- For VHDL designs, specify the name of the initial memory content file in the Generic Map Clause of the instance, with the specified type `LPM_FILE`. If you do not use an initial memory content file (e.g., for a RAM function), you should not declare or use the Generic Clause.
- Do not synthesize the **genmem**-generated VHDL file: it is intended for simulation only.

☞ The MIF format is supported only for specifying initial memory content when compiling designs within the MAX+PLUS II software. You cannot use a MIF to perform simulation with Viewlogic tools prior to MAX+PLUS II compilation.

To instantiate RAM or ROM in a ViewDraw schematic, follow these steps:

1. Use the **genmem** utility to generate a memory model by typing the following command at the UNIX prompt:

   `genmem` <*memory type*> <*memory size*> `-vwlogic` ↵

For example: `genmem asynrom 256x15 -vwlogic ↵`

2. Start the VHDL-to-symbol utility, **vhdl2sym**, by double-clicking Button 1 on the **max2_vhdl2sym** icon in the Altera® Toolbox Design Tools Drawer.

3. Specify the following options in the **vhdl2sym** dialog box and choose **OK** to create a symbol. For example, to create the symbol for a 256x15 asynchronous ROM, enter the following settings:

Option:	**Setting:**
VHDL Source Filename	*asyn_rom_256x15.vhd*
Add LEVEL attribute	*On*

4. Choose **Comp** (Add menu), type <*design name*> in the *Enter Name* box, and choose **OK**.

To instantiate a RAM or ROM function in VHDL, follow these steps:

1. Repeat step 1 above.

2. Create a VHDL design that incorporates the text from the **genmem**-generated Component Declaration, <*memory name*>**.cmp**, and instantiate the <*memory name*> function.

Figure 1 shows a VHDL design that instantiates asyn_rom_256x15.vhd, a 256 x 15 ROM function.

Figure 1. VHDL Design File with ROM Instantiation (tstrom.vhd)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;



ENTITY tstrom IS
        PORT (
          addr    : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
          memenab : IN STD_LOGIC;
          q       : OUT STD_LOGIC_VECTOR (14 DOWNTO 0));
END tstrom;



ARCHITECTURE behavior OF tstrom IS

COMPONENT asyn_rom_256x15
     GENERIC (LPM_FILE : string);



PORT (Address : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
           MemEnab : IN STD_LOGIC;
           Q       : OUT STD_LOGIC_VECTOR(14 DOWNTO 0)
     );
END COMPONENT;



BEGIN

   u1: asyn_rom_256x15
        GENERIC MAP (LPM_FILE => "u1.hex")
   PORT MAP (Address => addr, MemEnab => memenab, Q =>q);
END behavior;
```

Related Topics:

- Go to FLEX 10K Device Family, which is available on the web, for additional information.

Creating Hierarchical Projects in ViewDraw Schematics

You can incorporate any MAX+PLUS® II-supported design file, such as an Altera® Hardware Description Language (AHDL) Text Design File (**.tdf**), into a project hierarchy that consists of both schematic and text files. To incorporate a non-ViewDraw design file into a higher-level schematic design, you must create a hollow-body symbol for it in the ViewDraw software. During compilation, the MAX+PLUS II software recognizes the symbol as an identifier for the design file, and inserts the correct logic and connections. You can incorporate any number of design files into a project hierarchy.

To create a hierarchical project in your ViewDraw schematic, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Viewlogic Powerview Working Environment.
2. Create a ViewDraw schematic and save it in your working directory, as described in Creating ViewDraw Schematics for Use with MAX+PLUS II Software.

   You can instantiate MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.
3. Create a design file that uses all uppercase letters for the function name and all lowercase letters for the file extension, e.g., **DECODE.tdf**. This naming convention is required to prevent conflicts when the file is incorporated into a hierarchical design. When the **edifneto** utility generates an EDIF netlist file from the ViewDraw schematic, it copies the name of the hollow-body symbol in uppercase letters, regardless of the case that appears in the schematic.
4. Double-click Button 1 on the **max2_VDraw** icon in the Altera Toolbox Design Tools Drawer to start ViewDraw.
5. In the **File Open** dialog box, type <*design name*>, i.e., the name of the hollow-body symbol you want to create. Turn on the *Symbol* option and choose **OK**. The Symbol Editor is displayed.
6. Choose **Block Size Z-WxH** (Change menu) and select a symbol size.
7. Choose **Graphics-Box** (Add menu) to draw the symbol body.
8. Choose **Pin** (Add menu) to enter pinstubs.
9. Select a pin and choose **Label** (Add menu) to label the pin names.
10. (Optional) **Choose Graphics-Text** (Add menu) to label the symbol.
11. Choose **Block Type Module** (Change menu). You must choose **Block Type Module** to specify that no Viewlogic schematic is available to represent the functionality of the symbol.
12. Choose **Write** (File menu) to save the symbol.
13. In the top-level ViewDraw schematic, choose **Comp** (Add menu), select the name of the symbol, and choose

OK.

14. The MAX+PLUS II software uses Library Mapping Files (**.lmf**) to map standard ViewDraw symbols to equivalent MAX+PLUS II megafunctions, macrofunctions, or primitives. To use custom symbols, you can create a custom LMF that maps your custom symbols to the equivalent EDIF Input File, TDF, or other design file.

    ☞ You will also need to specify a Library Mapping File (.lmf) in the EDIF Netlist Reader Settings dialog box before compiling with the MAX+PLUS II Software. Go to Compiling Projects with MAX+PLUS II Software for more information.

15. Continue with the steps necessary to complete your ViewDraw schematic, as described in Creating ViewDraw Schematics for Use with MAX+PLUS II Software.

Related Topics:

- Go to Creating AHDL Designs for Use with MAX+PLUS II Software in these MAX+PLUS II ACCESS℠ Key topics for related information.

Entering Resource Assignments

The MAX+PLUS® II software allows you to enter a variety of resource and device assignments for your projects. Resource assignments are used to assign logic functions to a particular pin, logic cell, I/O cell, embedded cell, row, column, Logic Array Block (LAB), Embedded Array Block (EAB), chip, clique, local routing, logic option, timing requirement, or connected pin group. In the MAX+PLUS II software, you can enter all types of resource and device assignments with Assign menu commands. You can also enter pin, logic cell, I/O cell, embedded cell, LAB, EAB, row, and column assignments in the MAX+PLUS II Floorplan Editor. The Assign menu commands and the Floorplan Editor all save assignment information in the ASCII Assignment & Configuration File (**.acf**) for the project. In addition, you can edit ACFs manually in any standard text editor.

ViewDraw Schematics

In ViewDraw schematics, you can assign a limited subset of these resource assignments by assigning properties to symbols. These properties are incorporated into the EDIF netlist file(s). The MAX+PLUS II software automatically converts assignment information from the EDIF Input File (**.edf**) into the ACF format. For information on making MAX+PLUS II-compatible resource assignments, go to the following topics:

- Assigning Pins, Logic Cells & Chips
- Assigning Cliques
- Assigning Logic Options
- Modifying the Assignment & Configuration File with the **setacf** Utility

Installing the Altera-provided MAX+PLUS II/Viewlogic interface on your computer automatically creates the following sample ViewDraw schematic file, which includes resource assignments:

- **/usr/maxplus2/examples/viewlogic/example4/fadd2mpp**

Related Topics:

- Go to Viewlogic documentation for information on how to assign properties. Go to "Resource Assignments in EDIF Input Files" and "Assigning Resources in a Third-Party Design Editor" in MAX+PLUS II Help for more information on assignments or properties that can be assigned in ViewDraw.

VHDL Design Files

For VHDL-based designs, you must use the MAX+PLUS II software or the **setacf** utility to enter resource assignments. For information on using the **setacf** utility, go to Modifying the Assignment & Configuration File with the **setacf** Utility.

Related Topics:

- For information on entering assignments in the MAX+PLUS II software with Assign menu commands or in an ACF, go to "resource assignments" or "ACF, format" in MAX+PLUS II Help using **Search for Help on** (Help menu).

Assigning Pins, Logic Cells & Chips

You can assign a single logic function to a specific pin or logic cell (including I/O cells and embedded cells) within a chip, and assign one or more functions to a specific chip. A chip is a group of logic functions defined as a single, named unit, which can be assigned to a specific device.

You can assign a signal to a particular pin to ensure that the signal is always associated with that pin, regardless of future changes to the project. If you wish to set and maintain the performance of your project, assigning logic to a specific logic cell within a chip can minimize timing delays. In a project that is partitioned among multiple devices, you can assign logic functions that must be kept together in the same device to a chip. Chip assignments allow you to split a project so that only a minimum number of signals travel between devices, and to ensure that no unnecessary device-to-device delays exist on critical timing paths. You can assign a chip to a device in some EDA tools or in the MAX+PLUS® II software.

Use the following syntax for chip, pin, and logic cell assignments:

- To assign a logic function to a chip:

  `CHIP_PIN_LC=<`*chip name*`>`

  For example: `CHIP_PIN_LC=chip1`

- To assign a pin number within a chip:

  `CHIP_PIN_LC=<`*chip name*`>@<`*pin number*`>`

  For example: `CHIP_PIN_LC=chip1@K2`

- To assign a logic cell, I/O cell, or embedded cell number:

  `CHIP_PIN_LC=<`*chip name*`>@LC<`*logic cell number*`>`

  `CHIP_PIN_LC=<`*chip name*`>@IOC<`*I/O cell number*`>`

  `CHIP_PIN_LC=<`*chip name*`>@EC<`*embedded cell number*`>`

  For example: `CHIP_PIN_LC=chip1@LC44`

Related Topics:

- Refer to the following sources for additional information:
  - Go to "Devices & Adapters" and "Assigning a Device" in MAX+PLUS II Help for information on

device pin-outs and assigning devices, respectively, in the MAX+PLUS II software.
- Go to Back-Annotating MAX+PLUS II Pin Assignments to Design Architect Symbols for information on back-annotating pin assignments in Mentor Graphics Design Architect schematics.

Assigning Cliques

You can define a group of logic functions as a single, named unit, called a clique. The MAX+PLUS® II Compiler attempts to place all logic in the clique in the same logic array block (LAB) to ensure optimum speed. If the project does not use multi-LAB devices, or if it is not possible to fit all clique members into a single LAB, the clique assignment ensures that all members of a clique are placed in the same device. In FLEX® 6000, FLEX 8000, FLEX 10K, and MAX® 9000 devices the Compiler also attempts to place the logic in LABs in the same row. Cliques therefore allow you to partition a project so that only a minimum number of signals travel between LABs, and to ensure that no unnecessary LAB-to-LAB or device-to-device delays exist on critical timing paths.

To assign a clique, use the following syntax:

✔ `CLIQUE=`<*clique name*>

For example: `CLIQUE=fast1`

Related Topics:

- Go to the following topics in MAX+PLUS II Help for related information:
  - Assigning a Clique
  - Guidelines for Achieving Maximum Speed Performance

Assigning Logic Options

Logic option and logic synthesis style assignments allow you to guide logic synthesis with logic optimization features that are specific to Altera® devices. You can assign logic options and styles to individual logic functions in your design. The MAX+PLUS® II Compiler also uses a device-family-specific default logic synthesis style for each project.

Related Topics:

- Go to "Resource Assignments in EDIF Input Files" and "Assigning Resources in a Third-Party Design Editor" in MAX+PLUS II Help for complete and up-to-date information on logic option and logic synthesis style assignments, including definitions and syntax of these assignments.

Modifying the Assignment & Configuration File with the setacf Utility

Altera provides the **setacf** utility to help you modify a project's Assignment & Configuration File (**.acf**) from the command line, without opening the file with a text editor. Type `setacf -h` ↵ at a UNIX or DOS prompt to get help on this utility.

Performing a Functional Simulation with ViewSim Software

You can use Viewlogic ViewSim software to perform a functional simulation of a ViewDraw schematic or a VHDL Design File (**.vhd**) before compiling your project with the MAX+PLUS II Compiler. Follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II /Viewlogic Powerview Working Environment.

2. Create a ViewDraw schematic that follows the guidelines in Creating ViewDraw Schematics for Use with MAX+PLUS II Software. Then go to step 3.

   or:

   Create a VHDL Design File <*design name*>**.vhd** and analyze it, as described in the following MAX+PLUS II ACCESS^{SM} Key topics:

   - Creating VHDL Designs for Use with MAX+PLUS II Software
   - Analyzing VHDL Files with the Vantage VHDL Analyzer Software

   Then go to step 7.

3. With the schematic open in the ViewDraw editor, add `CLR` and `PRE` inputs to any flipflops in your design, or tie the `CLR` and `PRE` ports of the flipflops to VCC. (Use the `PWR` primitive from the **builtin** library.)

4. Choose **Write To** (File menu) and save the schematic as <*design name*>**_funct**.

5. Start the **vsm** utility by double-clicking Button 1 on the **max2_vsmnet** icon in the Altera® Toolbox Design Tools Drawer.

6. Specify the following options in the **vsm** dialog box and choose **OK** to generate the <*design name*>**_funct.vsm** file:

Option:	**Setting:**
Design Name	*<design name>_funct*
Level	(blank)

7. Create a simulation command file (**.cmd**) for simulation with ViewSim software. Alternatively, you can enter commands at the prompt in the ViewSim window. Refer to your Viewlogic documentation for more information on creating ViewSim command files.

8. Start the ViewSim simulation tool by double-clicking Button 1 on the **max2_VSim** icon in the Design Tools Drawer.

9. If you wish to simulate a ViewDraw schematic, specify the following options in the **ViewSim** dialog box, then go to step 11.

Option:	**Setting:**
Design Name	*<design name>_funct*
Command File	*<design name>_funct.cmd*
VHDL Source Window	*OFF*
VHDL Debugging	*OFF*

10. If you wish to simulate a VHDL design, specify the following options in the **ViewSim** dialog box:

Option:	**Setting:**
Design Name	*<design name>*

Command File	*<design name>.cmd*
Graphical Interface	*ON*
VHDL Source Window	*OFF* or *ON*
VHDL Debugging	*OFF* or *ON*

11. Choose **OK** to simulate the design. ViewSim software simulates the design and starts the ViewTrace waveform editor to allow you to observe the simulation results.

12. Use the **edifneto** utility to generate an EDIF Netlist File (**.edf**) that can be imported into the MAX+PLUS II software, as described in Converting ViewDraw Schematics or VHDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the **edifneto** Utility.

Related Topics:

- Go to ViewSim documentation for complete details on simulating a project and using ViewTrace to observe waveform output results.

Converting ViewDraw Schematics or VHDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the edifneto Utility

You can use the **edifneto** utility to generate an EDIF netlist file from a ViewDraw schematic or VHDL Design File (**.vhd**). This file can be imported into the MAX+PLUS® II software as an EDIF Input File with the extension **.edf**. To generate an EDIF netlist file, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Viewlogic Powerview Working Environment.

2. Create a ViewDraw schematic and save it in your working directory, as described in Creating ViewDraw Schematics for Use with MAX+PLUS II Software.

   or:

   Create a VHDL Design File, analyze it, and synthesize and optimize it, as described in the following topics:

   - Creating VHDL Designs for Use with MAX+PLUS II Software
   - Analyzing VHDL Files with the Vantage VHDL Analyzer Software
   - Synthesizing & Optimizing VHDL Designs with ViewSynthesis Software

3. Start the **edifneto** utility by double-clicking Button 1 on the **max2_edifo** icon in the Design Tools Drawer or the Max2 Express Drawer in the Altera Toolbox. You can also start the **edifneto** utility by typing `edifneto` ↵ at the UNIX prompt.

4. If you are converting a ViewDraw schematic, specify the *<design name>* for the *Wire File Name* option in the **edifneto** dialog box. If you are not using the Altera® toolbox, do not specify *Altera* for the *Level* option in the **edifneto** dialog box.

5. If you are converting a VHDL Design File, or if your ViewDraw schematic instantiates Library of Parameterized Modules (LPM) functions, specify *Altera* and *VHDL* as the *Level* in the **edifneto** dialog box.

6. Choose **OK** to generate the EDIF netlist file. The **edifneto** utility creates the **max2** subdirectory under your working directory. The **max2** subdirectory contains the EDIF netlist file for your design.

When the **edifneto** utility generates an EDIF netlist file from a design that instantiates LPM functions, the EDIF netlist file may contain parameters with incorrect parameter names. To correct this problem, go to the **/usr/maxplus2/viewlogic/bin** directory and type `chlpmpty` <*design name*>`.edf` ↵ at the UNIX prompt to run the Altera-provided **chlpmpty** script, which converts all of the parameters to their correct names.

7. Process the <*design name*>**.edf** with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Related Topics:

- Go to the following MAX+PLUS II ACCESS℠ Key topics for related information:
  - Using the Max2 Express Drawer's **SCH <-> max2** Utility
  - Using the Max2 Express Drawer's **VHDL <-> max2** Utility

---

Creating VHDL Designs for Use with MAX+PLUS II Software

You can create VHDL design files with the MAX+PLUS® II Text Editor or another standard text editor and save them in the appropriate directory for your project. The MAX+PLUS II Text Editor offers the following advantages:

- VHDL templates are available with the **VHDL Templates** command (Templates menu). These templates are also available in the ASCII **vhdl.tmp** file, which is located in the **/usr/maxplus2** directory.
- If you use the MAX+PLUS II Text Editor to create your VHDL design, you can use the **Syntax Coloring** command (Options menu). The Syntax Coloring feature displays keywords and other elements in text files in different colors to distinguish them from other forms of syntax.

To create a VHDL design that can be synthesized and optimized with ViewSynthesis software, follow these steps:

1. You can instantiate the following Altera-provided logic functions in your VHDL design:
   - The **alt_mf** library contains the Altera® VHDL logic function library, which includes MAX+PLUS II-specific primitives and the `a_8count`, `a_8mcomp`, `a_8fadd`, and `a_81mux` macrofunctions. If you wish to instantiate **alt_mf** logic functions in your VHDL design, you must first analyze all functions in the **alt_mf/src** directory. See Analyzing VHDL Files with the Vantage VHDL Analyzer Software for details.
   - The `clklock` megafunction, which enables the phase-locked loop, or ClockLock , circuitry available on selected Altera FLEX® 10K devices. Go to Instantiating the clklock Megafunction in VHDL or Verilog HDL for information.
   - MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.
2. (Optional) To enter resource assignments in your VHDL design, go to Entering Resource Assignments. You can also enter resource assignments from within the MAX+PLUS II software.

Once you have created a VHDL design, you can analyze it, synthesize it, and generate an EDIF netlist file that can be imported into the MAX+PLUS II software with either of the following methods:

- You can analyze, functionally simulate, and synthesize the VHDL design, then generate an EDIF netlist file by following the steps in these topics:
  - Analyzing VHDL Files with the Vantage VHDL Analyzer Software
  - Performing a Functional Simulation with ViewSim Software
  - Synthesizing & Optimizing VHDL Designs with ViewSynthesis Software
  - Converting ViewDraw Schematics or VHDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the **edifneto** Utility
- You can use the **VHDL <-> max2** utility in the Max2 Express Drawer to automatically analyze and synthesize the VHDL design, compile it with the MAX+PLUS II Compiler, generate an EDIF Output File (**.edo**), and create a **.vsm** file for simulation. See Using the Max2 Express Drawer's **VHDL <-> max2** Utility in these MAX+PLUS II ACCESS^{SM} Key topics for details.

Installing the Altera-provided MAX+PLUS II/Viewlogic Powerview interface on your computer automatically creates the following sample VHDL files:

- **/usr/maxplus2/examples/viewlogic/example5/count4.vhd**
- **/usr/maxplus2/examples/viewlogic/example5/count8.vhd**

Related Topics:

- Go to FLEX 10K Device Family, which is available on the web, for additional information.

Instantiating the clklock Megafunction in VHDL or Verilog HDL

MAX+PLUS® II interfaces to other EDA tools support the `clklock` phase-locked loop megafunction, which can be used with some FLEX® 10K devices, with the **genckllk** utility, which is available in the MAX+PLUS II system directory. Type `genckllk -h` ↵ at the DOS or UNIX prompt to display information on how to use this utility. The **genckllk** utility generates VHDL or Verilog HDL functional simulation models and a VHDL Component Declaration template file (**.cmp**).

The **genckllk** utility allows parameters for the `clklock` function to be passed from the VHDL or Verilog HDL file to EDIF netlist format. The **genckllk** utility embeds the parameter values in the `clklock` function name; therefore, the values do not need to be declared explicitly.

To instantiate the `clklock` megafunction in VHDL or Verilog HDL, go through the following steps:

1. Type the following command at the DOS or UNIX prompt to generate the `clklock_`*x*`_`*y* function, where *x* is the ClockBoost value and *y* is the input frequency in MHz:

   ✔ Type `genckllk` <*ClockBoost*> <*input frequency*> `-vhdl` ↵ for VHDL designs.

   or:

   ✔ Type `genckllk` <*ClockBoost*> <*input frequency*> `-verilog` ↵ for Verilog HDL designs.

   Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu for more information on the `clklock` megafunction.

2. Create a design file that instantiates the `clklock_`*x*`_`*y***.vhd** or `clklock_`*x*`_`*y***.v** file. The **genckllk** utility automatically generates a VHDL Component Declaration template in the **clklock_*x*_*y*.cmp** file that you can incorporate into a VHDL design file.

In MAX+PLUS II version 8.3 and lower, running `genclklk` on a PC always creates files named as **clklock.vhd**, **clklock.cmp**, and **clklock.v**, regardless of the ClockBoost and input frequency values you specify.

Figures 1 and 2 show a `clklock` function with <*ClockBoost*> = 2 and <*input frequency*> = 40 MHz instantiated in VHDL and Verilog HDL design files, respectively.

Figure 1. VHDL Design File with clklock Instantiation (count8.vhd)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY altera;
USE altera.maxplus2.all;     --  Include Altera Component Declarations



ENTITY count8 IS
   PORT (a    : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
         ldn  : IN STD_LOGIC;
         gn   : IN STD_LOGIC;



dnup : IN STD_LOGIC;
         setn : IN STD_LOGIC;
         clrn : IN STD_LOGIC;
         clk  : IN STD_LOGIC;



co   : OUT STD_LOGIC;
         q    : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END count8;



ARCHITECTURE structure OF count8 IS
   signal clk2x : STD_LOGIC;



COMPONENT clklock_2_40
   PORT (
      INCLK : IN STD_LOGIC;
      OUTCLK : OUT STD_LOGIC
   );
END COMPONENT;



BEGIN
   u1: clklock_2_40
   PORT MAP (inclk=>clk, outclk=>clk2x);



u2: a_8count
   PORT MAP (a=>a(0), b=>a(1), c=>a(2), d=>a(3),
             e=>a(4), f=>a(5), g=>a(6), h=>a(7),
             clk=>clk2x,
             ldn=>ldn,
             gn=>gn,



dnup=>dnup,
             setn=>setn,
             clrn=>clrn,



qa=>q(0), qb=>q(1), qc=>q(2), qd=>q(3),
```

```
            qe=>q(4), qf=>q(5), qg=>q(6), qh=>q(7),
            cout=>co);
END structure;
```

Figure 2. Verilog HDL Design File with clklock Instantiation (count8.v)

```
`timescale 1ns / 10ps
module count8 (a, ldn, gn, dnup, setn, clrn, clk, co, q);
output          co;
output[7:0]     q;



input[7:0]      a;
input           ldn, gn,dnup, setn, clrn, clk;
wire            clk2x;



clklock_2_40 u1 (.inclk(clk), .outclk(clk2x) );
A_8COUNT u2 (.A(a[0]), .B(a[1]), .C(a[2]), .D(a[3]), .E(a[4]), .F(a[5]),



.G(a[6]), .H(a[7]), .LDN(ldn), .GN(gn), .DNUP(dnup),
            .SETN(setn), .CLRN(clrn), .CLK(clk2x), .QA(q[0]), .QB(q[1]),
            .QC(q[2]), .QD(q[3]), .QE(q[4]), .QF(q[5]), .QG(q[6]),
            .QH(q[7]), .COUT(co) );



endmodule
```

Related Topics:

- Go to FLEX 10K Device Family, which is available on the web, for additional information.

Instantiating RAM & ROM Functions in Viewlogic Powerview Designs

The MAX+PLUS®II /Viewlogic Powerview interface offers full support for the memory capabilities of the FLEX® 10K device family, including synchronous and asynchronous RAM and ROM, cycle-shared dual-port RAM, dual-port RAM, single-Clock FIFO, and dual-Clock FIFO functions. You can use the Altera-provided **genmem** utility to generate functional simulation models and timing models for these functions. Type `genmem` ↵ at the UNIX prompt to display information on how to use this utility, as well as a list of the functions you can generate. RAM and ROM can be instantiated in both ViewDraw schematics and VHDL designs.

Refer to Viewlogic documentation for information on simulating projects that contain RAM functions. The procedure for reading an EDIF Output File and preparing it for simulation with ViewSim requires additional steps when the project contains RAM functions.

When you instantiate a RAM or ROM function, follow these general guidelines:

- For ROM functions, you must specify an initial memory content file in the Intel hexadecimal format (**.hex**) or the Altera® Memory Initialization File (**.mif**) format. The filename must be the same as the instance name; e.g., the instance name must be unique throughout the whole project, and must contain only valid name characters. The initialization file must reside in the directory containing the project's design files.

- For RAM functions, specifying a memory initialization file is optional.
- For VHDL designs, specify the name of the initial memory content file in the Generic Map Clause of the instance, with the specified type `LPM_FILE`. If you do not use an initial memory content file (e.g., for a RAM function), you should not declare or use the Generic Clause.
- Do not synthesize the **genmem**-generated VHDL file: it is intended for simulation only.

The MIF format is supported only for specifying initial memory content when compiling designs within the MAX+PLUS II software. You cannot use a MIF to perform simulation with Viewlogic tools prior to MAX+PLUS II compilation.

To instantiate RAM or ROM in a ViewDraw schematic, follow these steps:

1. Use the **genmem** utility to generate a memory model by typing the following command at the UNIX prompt:

   `genmem` <*memory type*> <*memory size*> `-vwlogic` ↵

   For example: `genmem asynrom 256x15 -vwlogic` ↵

2. Start the VHDL-to-symbol utility, **vhdl2sym**, by double-clicking Button 1 on the **max2_vhdl2sym** icon in the Altera® Toolbox Design Tools Drawer.
3. Specify the following options in the **vhdl2sym** dialog box and choose **OK** to create a symbol. For example, to create the symbol for a 256x15 asynchronous ROM, enter the following settings:

Option:	**Setting:**
VHDL Source Filename	*asyn_rom_256x15.vhd*
Add LEVEL attribute	*On*

4. Choose **Comp** (Add menu), type <*design name*> in the *Enter Name* box, and choose **OK**.

To instantiate a RAM or ROM function in VHDL, follow these steps:

1. Repeat step 1 above.
2. Create a VHDL design that incorporates the text from the **genmem**-generated Component Declaration, <*memory name*>**.cmp**, and instantiate the <*memory name*> function.

Figure 1 shows a VHDL design that instantiates **asyn_rom_256x15.vhd**, a 256 x 15 ROM function.

Figure 1. VHDL Design File with ROM Instantiation (tstrom.vhd)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;


ENTITY tstrom IS
        PORT (
          addr    : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
          memenab : IN STD_LOGIC;
          q       : OUT STD_LOGIC_VECTOR (14 DOWNTO 0));
END tstrom;


ARCHITECTURE behavior OF tstrom IS
```

```
COMPONENT asyn_rom_256x15
     GENERIC (LPM_FILE : string);



PORT (Address : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
           MemEnab : IN STD_LOGIC;
           Q       : OUT STD_LOGIC_VECTOR(14 DOWNTO 0)
     );
END COMPONENT;



BEGIN

   u1: asyn_rom_256x15
        GENERIC MAP (LPM_FILE => "u1.hex")
   PORT MAP (Address => addr, MemEnab => memenab, Q =>q);
END behavior;
```

Related Topics:

- Go to FLEX 10K Device Family, which is available on the web, for additional information.

Entering Resource Assignments

The MAX+PLUS® II software allows you to enter a variety of resource and device assignments for your projects. Resource assignments are used to assign logic functions to a particular pin, logic cell, I/O cell, embedded cell, row, column, Logic Array Block (LAB), Embedded Array Block (EAB), chip, clique, local routing, logic option, timing requirement, or connected pin group. In the MAX+PLUS II software, you can enter all types of resource and device assignments with Assign menu commands. You can also enter pin, logic cell, I/O cell, embedded cell, LAB, EAB, row, and column assignments in the MAX+PLUS II Floorplan Editor. The Assign menu commands and the Floorplan Editor all save assignment information in the ASCII Assignment & Configuration File (**.acf**) for the project. In addition, you can edit ACFs manually in any standard text editor.

ViewDraw Schematics

In ViewDraw schematics, you can assign a limited subset of these resource assignments by assigning properties to symbols. These properties are incorporated into the EDIF netlist file(s). The MAX+PLUS II software automatically converts assignment information from the EDIF Input File (**.edf**) into the ACF format. For information on making MAX+PLUS II-compatible resource assignments, go to the following topics:

- Assigning Pins, Logic Cells & Chips
- Assigning Cliques
- Assigning Logic Options
- Modifying the Assignment & Configuration File with the **setacf** Utility

Installing the Altera-provided MAX+PLUS II/Viewlogic interface on your computer automatically creates the following sample ViewDraw schematic file, which includes resource assignments:

- **/usr/maxplus2/examples/viewlogic/example4/fadd2mpp**

Related Topics:

- Go to Viewlogic documentation for information on how to assign properties. Go to "Resource

Assignments in EDIF Input Files" and "Assigning Resources in a Third-Party Design Editor" in MAX+PLUS II Help for more information on assignments or properties that can be assigned in ViewDraw.

VHDL Design Files

For VHDL-based designs, you must use the MAX+PLUS II software or the **setacf** utility to enter resource assignments. For information on using the **setacf** utility, go to Modifying the Assignment & Configuration File with the **setacf** Utility.

Related Topics:

- For information on entering assignments in the MAX+PLUS II software with Assign menu commands or in an ACF, go to "resource assignments" or "ACF, format" in MAX+PLUS II Help using **Search for Help on** (Help menu).

Modifying the Assignment & Configuration File with the setacf Utility

Altera provides the **setacf** utility to help you modify a project's Assignment & Configuration File (**.acf**) from the command line, without opening the file with a text editor. Type `setacf -h` ↵ at a UNIX or DOS prompt to get help on this utility.

Analyzing VHDL Files with the SpeedWave VHDL Analyzer Software

You can use the SpeedWave VHDL Analyzer software to analyze VHDL Design Files (**.vhd**) prior to functional (or gate-level) simulation with ViewSim software, or to synthesis and optimization with ViewSynthesis software. You can also use the SpeedWave VHDL Analyzer to analyze a MAX+PLUS® II -generated VHDL Output File (**.vho**) prior to post-compilation timing simulation with ViewSim software. The **max2_VantgMgr** and **max2_VantgAnlz** tools are located in the Altera® Toolbox Design Tools Drawer.

To analyze a VHDL file with the SpeedWave VHDL Analyzer, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II /Viewlogic Powerview Working Environment.

2. If you wish to analyze a VHDL Design File (**.vhd**), create a VHDL file <*design name*>**.vhd** using the MAX+PLUS II Text Editor or another standard text editor and save it in a working directory. Go to Creating VHDL Designs for Use with MAX+PLUS II Software for more information.

3. If you wish to analyze a MAX+PLUS II-generated VHDL Output File (**.vho**), be sure to select *VHDL 1987* for the *VHDL Version* option and *VHDL Output File (.vho)* for the *Write Delay Constructs To* option in the **VHDL Netlist Writer Settings** dialog box (Interfaces menu) when you set up the MAX+PLUS II Compiler to generate a VHDL Output File. See Compiling Projects with MAX+PLUS II Software for more information on generating VHDL Output Files.

4. If your VHDL file contains functions from the **alt_mf** library, follow these steps:

    1. Start the Vantage Manager by double-clicking Button 1 on the **max2_VantgMgr** icon in the Design Tools Drawer.

    2. Use the Vantage VHDL Library Manager to create an **alt_mf.lib** library file with the symbolic name `ALT_MF`.

3. Make **alt_mf** the working library with the **Set Working** command (Edit menu).

4. Start the VHDL Analyzer by double-clicking Button 1 on the **max2_VantgAnlz** icon in the Design Tools Drawer.

5. Analyze each VHDL file in the **alt_mf/src** directory into the **alt_mf.lib** working library. Source files are located in the **/usr/maxplus2/vwlogic/library/alt_mf/src** directory that is created by installing the Altera/Viewlogic interface.

5. If it is not already running, start the Vantage VHDL Library Manager, as described in step 4b, to create a Vantage library.

6. Choose the **List system libs** button.

7. Add the **ieee.lib** and **synopsys.lib** system libraries to your project:

    1. Select the **ieee.lib** and **synopsys.lib** libraries from the *Available Libraries* window and choose **Add lib**. Choose the **ieee** library from the **libs_syn** directory, which is located at ***/<Powerview system directory>/*** **standard/van_vss/pgm/libs_syn**. The **ieee** library contains Synopsys package files.

    2. If your project uses functions from the **alt_mf** library, also select the **alt_mf.lib** file from the *Available Libraries* window and choose **Add lib**.

    3. Choose **Create Library** (File menu, type the project directory name in the *Symbolic Name* field, and choose **OK**.

8. Specify the project directory as the working directory by choosing **Set Working** (Edit menu).

9. Choose **Save INI File** (File menu).

10. Choose **Dismiss Window** (Powerview Red-Box menu).

11. Specify the appropriate path and file name in the **Analyzer VHDL Source File** dialog box and choose **OK** to analyze the VHDL file.

12. Once you have analyzed the file, perform one or more of the following tasks, as appropriate:

    - Performing a Functional Simulation with ViewSim Software
    - Synthesizing & Optimizing VHDL Designs with ViewSynthesis Software
    - Performing a Timing Simulation with ViewSim Software

Related Topics:

- Refer to the following sources for related information:
    - The ***Viewlogic ViewSim/VHDL User's Guide*** and ***ViewSim/VHDL Tutorial*** for information on using the Vantage VHDL Analyzer software or Vantage VHDL Library Manager
    - Powerview Command-Line Syntax in these MAX+PLUS II ACCESS^{SM} Key topics

---

Performing a Functional Simulation with ViewSim Software

You can use Viewlogic ViewSim software to perform a functional simulation of a ViewDraw schematic or a VHDL Design File (**.vhd**) before compiling your project with the MAX+PLUS II Compiler. Follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II /Viewlogic Powerview Working Environment.

2. Create a ViewDraw schematic that follows the guidelines in Creating ViewDraw Schematics for Use with MAX+PLUS II Software. Then go to step 3.

   or:

   Create a VHDL Design File <*design name*>**.vhd** and analyze it, as described in the following MAX+PLUS II ACCESS^{SM} Key topics:

   - Creating VHDL Designs for Use with MAX+PLUS II Software
   - Analyzing VHDL Files with the Vantage VHDL Analyzer Software

   Then go to step 7.

3. With the schematic open in the ViewDraw editor, add `CLR` and `PRE` inputs to any flipflops in your design, or tie the `CLR` and `PRE` ports of the flipflops to VCC. (Use the `PWR` primitive from the **builtin** library.)

4. Choose **Write To** (File menu) and save the schematic as <*design name*>**_funct**.

5. Start the **vsm** utility by double-clicking Button 1 on the **max2_vsmnet** icon in the Altera® Toolbox Design Tools Drawer.

6. Specify the following options in the **vsm** dialog box and choose **OK** to generate the <*design name*>**_funct.vsm** file:

Option:	**Setting:**
Design Name	*<design name>_funct*
Level	(blank)

7. Create a simulation command file (**.cmd**) for simulation with ViewSim software. Alternatively, you can enter commands at the prompt in the ViewSim window. Refer to your Viewlogic documentation for more information on creating ViewSim command files.

8. Start the ViewSim simulation tool by double-clicking Button 1 on the **max2_VSim** icon in the Design Tools Drawer.

9. If you wish to simulate a ViewDraw schematic, specify the following options in the **ViewSim** dialog box, then go to step 11.

Option:	**Setting:**
Design Name	*<design name>_funct*
Command File	*<design name>_funct.cmd*
VHDL Source Window	*OFF*
VHDL Debugging	*OFF*

10. If you wish to simulate a VHDL design, specify the following options in the **ViewSim** dialog box:

Option:	**Setting:**
Design Name	*<design name>*
Command File	*<design name>.cmd*
Graphical Interface	*ON*

VHDL Source Window *OFF* or *ON*
VHDL Debugging *OFF* or *ON*

11. Choose **OK** to simulate the design. ViewSim software simulates the design and starts the ViewTrace waveform editor to allow you to observe the simulation results.

12. Use the **edifneto** utility to generate an EDIF Netlist File (**.edf**) that can be imported into the MAX+PLUS II software, as described in Converting ViewDraw Schematics or VHDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the **edifneto** Utility.

Related Topics:

- Go to ViewSim documentation for complete details on simulating a project and using ViewTrace to observe waveform output results.

Converting ViewDraw Schematics or VHDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the edifneto Utility

You can use the **edifneto** utility to generate an EDIF netlist file from a ViewDraw schematic or VHDL Design File (**.vhd**). This file can be imported into the MAX+PLUS® II software as an EDIF Input File with the extension **.edf**. To generate an EDIF netlist file, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Viewlogic Powerview Working Environment.

2. Create a ViewDraw schematic and save it in your working directory, as described in Creating ViewDraw Schematics for Use with MAX+PLUS II Software.

   or:

   Create a VHDL Design File, analyze it, and synthesize and optimize it, as described in the following topics:

   - Creating VHDL Designs for Use with MAX+PLUS II Software
   - Analyzing VHDL Files with the Vantage VHDL Analyzer Software
   - Synthesizing & Optimizing VHDL Designs with ViewSynthesis Software

3. Start the **edifneto** utility by double-clicking Button 1 on the **max2_edifo** icon in the Design Tools Drawer or the Max2 Express Drawer in the Altera Toolbox. You can also start the **edifneto** utility by typing `edifneto` ↵ at the UNIX prompt.

4. If you are converting a ViewDraw schematic, specify the <*design name*> for the *Wire File Name* option in the **edifneto** dialog box. If you are not using the Altera® toolbox, do not specify *Altera* for the *Level* option in the **edifneto** dialog box.

5. If you are converting a VHDL Design File, or if your ViewDraw schematic instantiates Library of Parameterized Modules (LPM) functions, specify *Altera* and *VHDL* as the *Level* in the **edifneto** dialog box.

6. Choose **OK** to generate the EDIF netlist file. The **edifneto** utility creates the **max2** subdirectory under your working directory. The **max2** subdirectory contains the EDIF netlist file for your design.

   When the **edifneto** utility generates an EDIF netlist file from a design that instantiates LPM functions, the EDIF netlist file may contain parameters with incorrect parameter names. To

correct this problem, go to the **/usr/maxplus2/viewlogic/bin** directory and type `chlpmpty` <*design name*>`.edf` ↵ at the UNIX prompt to run the Altera-provided **chlpmpty** script, which converts all of the parameters to their correct names.

7. Process the <*design name*>**.edf** with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Related Topics:

- Go to the following MAX+PLUS II ACCESS℠ Key topics for related information:
  - Using the Max2 Express Drawer's **SCH <-> max2** Utility
  - Using the Max2 Express Drawer's **VHDL <-> max2** Utility

---

Synthesizing & Optimizing VHDL Designs with ViewSynthesis Software

You can create and process VHDL files and convert them into Altera® Hardware Description Language (AHDL) Text Design Files (**.tdf**) or EDIF Input Files (**.edf**) that can be processed by the MAX+PLUS® II Compiler. The MAX+PLUS II Compiler can process a VHDL file that has been synthesized by ViewSynthesis software, saved as an AHDL TDF or an EDIF netlist file, and imported into the MAX+PLUS II software. The information presented here describes only how to use VHDL files that have been processed by ViewSynthesis software. For information on direct MAX+PLUS II support for VHDL Design Files, go to MAX+PLUS II VHDL Help.

To synthesize and optimize a VHDL design, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Viewlogic Powerview Working Environment.

2. Create a VHDL file <*design name*>.**vhd** using the MAX+PLUS II Text Editor or another standard text editor and save it in a working directory. Go to Creating VHDL Designs for Use with MAX+PLUS II Software for more information.

3. Start Powerview by typing `powerview` ↵ at a UNIX prompt.

4. In the Cockpit window, select *Altera* in the *Current ToolBox* drop-down list box, and select the drawer you want to use, i.e., *Design Tools* or *Max2 Express*, in the *Current Drawer* drop-down list box.

5. Choose **Create** (Project menu) from your working directory to create your project directory. Choose **OK**.

6. Choose **SearchOrder** (Project menu) to add the appropriate library directories and aliases to your **viewdraw.ini** file. Refer to Viewlogic Powerview **viewdraw.ini** Configuration File for more information on Powerview application libraries.

   When you add libraries to the **/usr/maxplus2/vwlogic/standard/viewdraw.ini** file, they are automatically set when you create a new project. Powerview tools search these libraries sequentially, so it is important to add them in the order in which they are listed.

7. Analyze the VHDL design, as described in Analyzing VHDL Files with the Vantage VHDL Analyzer Software.

8. (Optional) Perform a functional simulation, as described in Performing a Timing Simulation with ViewSim Software.

9. In Powerview 5.3.2 and previous versions, start ViewSynthesis software by double-clicking Button 1 on the **max2_syn** icon in the Altera Toolbox Design Tools Drawer.

   In Powerview 6.0, ViewSynthesis software is available only for the SunOS, and only as a command-line version. If you are using Powerview 6.0, read ***/<Powerview system directory>*/README/vsyn.doc** to learn how to synthesize a design with ViewSynthesis software. You can create the **synth.ini** file, a one-line text file that contains the text `technology altera`. Then type the following commands at the UNIX prompt to analyze and synthesize your VHDL design:

   ```
   vsyn -vhdl <design name> ↵

   vsyn -synth "*" ↵
   ```

10. Choose **Target Technology** (Setup menu) and select *altera:altera* in the **Specify Target Technology** dialog box. Choose **OK**.

11. Choose **Compile VHDL** (Setup menu) and select *<design name>.vhd* in the *VHDL Files* list box. Choose **OK**.

    If more than one VHDL Design File (**.vhd**) exists for the project, you must compile the lower-level design files before compiling the top-level file.

12. Press Button 3 on the ***<design name>*** icon in the ViewSynthesis window, choose **Synthesize** from the pop-up menu, then choose **OK**.

13. (Optional) To generate a synthesis report file for the design, press Button 3 on the ***<design name>*** icon and choose **View Report** from the pop-up menu.

14. (Optional) To create a schematic representation of the gate-level netlist file, press Button 3 on the ***<design name>*** icon and choose **View Schematic** from the pop-up menu.

15. Generate an EDIF netlist file that can be compiled with the MAX+PLUS II Compiler, as described in Converting ViewDraw Schematics or VHDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the **edifneto** Utility.

16. Process the *<design name>*.**edf** file with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Installing the Altera-provided MAX+PLUS II/Viewlogic interface on your computer automatically creates the following sample ViewDraw schematic files:

- **/usr/maxplus2/examples/viewlogic/example5/count4.vhd**
- **/usr/maxplus2/examples/viewlogic/example5/count8.vhd**

Related Topics:

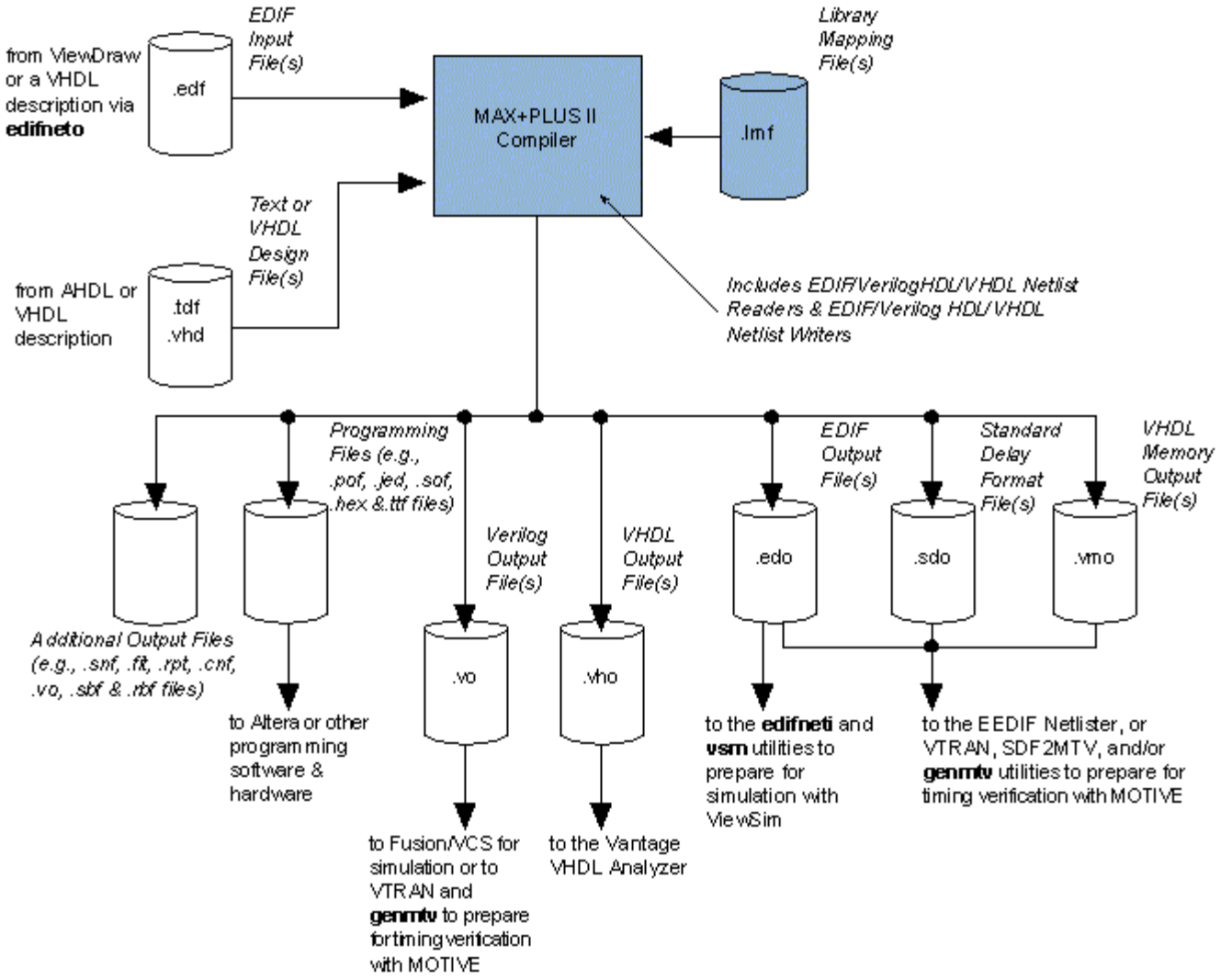
- Go to Powerview Command-Line Syntax in these MAX+PLUS II ACCESS^{SM} Key topics for related information.

MAX+PLUS II/Viewlogic Powerview Compilation Flow

Figure 1 shows the project compilation flow for the MAX+PLUS® II/Viewlogic Powerview interface.

Figure 1. MAX+PLUS II/Viewlogic Powerview Project Compilation Flow

Altera-provided items are shown in blue.



Compiling Projects with MAX+PLUS II Software

The MAX+PLUS® II Compiler can process design files in a variety of formats. This topic describes how to use MAX+PLUS II software to compile projects in which the top-level design file is an EDIF Input File (with the extension **.edf**).

☞ Refer to the following sources for additional information:

- Go to MAX+PLUS II Help for information on compiling VHDL and Verilog HDL, design files directly with the MAX+PLUS II Compiler.

- Go to Running Synopsys Compilers from MAX+PLUS II Software for information on running the Synopsys Design Compiler or FPGA Compiler software on a VHDL or Verilog HDL design from within the MAX+PLUS II Compiler window.

To compile a design (also called a "project") with MAX+PLUS II software, go through the following steps:

1. Create design files that are compatible with the MAX+PLUS II software and convert them into EDIF Input Files with the extension **.edf**. Specific instructions for some tools are described in these MAX+PLUS II ACCESS^{SM} Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your design entry or synthesis and optimization tool.

2. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, create a mapping for each function in a Library Mapping File (**.lmf**) to map the custom symbol to the corresponding EDIF Input File, AHDL Text Design File (**.tdf**), or other MAX+PLUS II-supported design file. These custom functions are represented in design files as hollow-body symbols or "black box" HDL descriptions.

   Go to "Library Mapping Files (**.lmf**)" in MAX+PLUS II Help for more information.

3. Open MAX+PLUS II and specify the name of your top-level design file as the project name with the **Project Name** command (File menu). If you open an HDL file in the MAX+PLUS II Text Editor, you can choose the **Project Set Project to Current File** command (File menu) instead.

   You can also compile a project from a command line. However, the first time you compile a project, the settings you need to specify are easier to specify from within the MAX+PLUS II software. After you have run the graphical user interface for the MAX+PLUS II software at least once, you can more easily use the command-line **setacf** utility to modify options in the Assignment & Configuration File (**.acf**) for the project. Type `setacf -h` ↵ and `maxplus2 -h` ↵ for descriptions of **setacf** and MAX+PLUS II command-line syntax.

4. Choose **Device** (Assign menu) and select the target Altera device family in the *Device Family* drop-down list box. If you wish to implement the design logic in a specific device, select it in the *Devices* box. Otherwise, select *AUTO* to allow the MAX+PLUS II Compiler to choose the best device(s) in the current device family. If your design entry or synthesis and optimization tool required you to specify a target family and/or device, specify the same information in this dialog box. For information on partitioning logic among multiple devices, go to MAX+PLUS II Help. Choose **OK**.

5. Open the Compiler window by choosing the **Compiler** command (MAX+PLUS II menu). Go through the following steps to specify the options necessary to compile the design file(s) in your project:

   1. Ensure that all EDIF netlist files have the extension **.edf** and choose **EDIF Netlist Reader Settings** (Interfaces menu).

   2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor. This name should be the name of the vendor whose tool(s) you used to create the EDIF netlist files. If your vendor name does not appear, select *Custom* instead.

      If you are compiling a design created with Synopsys FPGA Express software, select *Synopsys*, choose the **Customize** button, enter <*project name*>**.lmf** in the *LMF #1* box, choose **OK**, and skip to step 6.

3. If you selected an existing vendor name in the *Vendor* box and your project contains design files that require custom LMF mappings, choose the **Customize** button to expand the dialog box to show all settings. Turn on the *LMF #2* checkbox and type your custom LMF's filename in the corresponding text box, or select a name from the *Files* box. The selection in the *Vendor* box will change to *Custom* and all settings will be retained until you change them again.

4. If you selected *Custom* in the *Vendor* box, choose the **Customize** button to expand the dialog box to show all settings. Any previously defined custom settings will be displayed. Under *Signal Names*, type one or more names with up to 20 total name characters in the *VCC* or *GND* box if your EDIF Input File(s) use one or more names other than `VCC` or `GND` for the global high or low signals. Multiple signal names must be separated by either a comma (,) or a space. Under *Library Mapping Files*, turn on the *LMF #1* checkbox and type a filename in the text box following it, or select a name from the *Files* box. If necessary, specify another LMF name in the *LMF #2* box. Go to MAX+PLUS II Help for detailed information on the settings available in the **EDIF Netlist Reader Settings** dialog box.

5. Choose **OK**.

6. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, you may need to ensure that all files are present in your project directory, i.e., the same directory as the top-level design file. Otherwise, you must specify the directories containing these files as user libraries with the **User Libraries** command (Options menu).

7. Follow all guidelines that apply to your design entry or synthesis and optimization tool:

   - Exemplar Logic Galileo Extreme-Specific Compiler Settings
   - Synopsys DesignWare-Specific Compiler Settings
   - Converting Synopsys FPGA Compiler & Design Compiler Timing Constraints into MAX+PLUS II-Compatible Format with the **syn2acf** Utility
   - Synplicity Synplify-Specific Compiler Settings

8. If you wish to generate EDIF, VHDL, or Verilog HDL output files for post-compilation simulation or timing analysis with another EDA tool, go through the following steps:

   1. (Optional) Turn on the **Optimize Timing SNF** command (Processing menu) to reduce the size of the output file(s). Turning on this command can reduce the size of output netlists by up to 30%.

      This command does not create optimized timing SNFs on UNIX workstations. However, a non-optimized timing SNF provides the same functional and timing information as an optimized timing SNF.

   2. If you wish to generate EDIF Output Files (**.edo**), go through these steps:

      1. Turn on the **EDIF Netlist Writer** command (Interfaces menu). Then choose the **EDIF Netlist Writer Settings** command (Interfaces menu).

      2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor and choose **OK**. If your vendor name does not appear, select *Custom* instead and specify the settings that are appropriate for your simulation or timing analysis tool. Go to MAX+PLUS II Help for detailed information on the options available in the **EDIF Netlist Writer Settings** dialog box.

      3. To generate an optional Standard Delay Format (SDF) Output File (**.sdo**), choose

the **Customize** button to expand the dialog box to show all settings. Select one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**.

The filenames of the EDIF Output File(s) and optional SDF Output File(s) are the same as the user-defined chip name(s) for the project; if no chip names exist, the Compiler assigns filenames that are based on the project name. For a multi-device project, the Compiler also generates a top-level EDIF Output File that is uniquely identified by "**_t**" appended to the project name. In addition, the Compiler automatically generates a VHDL Memory Model Output File, *<project name>***.vmo**, when it generates an EDIF Output File that contains memory (RAM or ROM).

3. If you wish to generate VHDL Output Files (**.vho**), turn on the **VHDL Netlist Writer** command (Interfaces menu). Then choose **VHDL Netlist Writer Settings** command (Interfaces menu). Select *VHDL Output File (.vho)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF ver. 2.1 files contain timing delay information that allows you to perform back-annotation simulation in VHDL with VITAL-compliant simulation libraries. The VHDL Output Files generated by the Compiler have the extension **.vho**, but are otherwise named in the same way as the EDIF Output Files described above.

4. If you wish to generate Verilog HDL Output Files (**.vo**), turn on the **Verilog Netlist Writer** command (Interfaces menu). Then choose **Verilog Netlist Writer Settings** command (Interfaces menu). Select *Verilog Output File (.vo)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF Output Files contain timing delay information that allows you to perform back-annotation simulation in Verilog HDL. The Verilog Output Files generated by the Compiler have the extension **.vo**, but are otherwise named in the same way as the EDIF Output Files described above.

9. To run the MAX+PLUS II Compiler, choose the **Project Save & Compile** command (File menu) or choose the **Start** button in the Compiler window.

   See step 3 for information on running MAX+PLUS II software from the command line.

10. Once you have compiled the project with the MAX+PLUS II Compiler, you can use the VHDL, Verilog HDL, or EDIF output file(s), and the optional SDF Output File(s) (**.sdo**) to perform timing analysis or timing simulation with another EDA tool. Specific instructions for some tools are described in these MAX+PLUS II ACCESS Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your EDA tool.

The MAX+PLUS II Compiler also generates a Report File (**.rpt**), a Pin-Out File (**.pin**), and one or more of the following files for device programming or configuration:

- JEDEC Files (**.jed**)
- Programmer Object Files (**.pof**)
- SRAM Object Files (**.sof**)
- Hexadecimal (Intel-format) Files (**.hex**)
- Tabular Text Files (**.ttf**)

Related Topics:

- Refer to the following sources for additional information:
  - Go to *Compiler Procedures* in MAX+PLUS II Help for information on other available Compiler settings.
  - Go to *Programmer Procedures* in MAX+PLUS II Help for instructions on creating other types of programming files and on programming or configuring Altera devices.
  - Go to Back-Annotating MAX+PLUS II Pin Assignments to Design Architect Symbols for

information on back-annotating pin assignments in Mentor Graphics Design Architect schematics.
    - Go to Programming Altera Devices for information on the different programming hardware options for Altera device families.
- Go to the following topics, which are available on the web, for additional information:
    - MAX+PLUS II Development Software
    - Altera Programming Hardware

Using the Max2 Express Drawer's SCH <-> max2 Utility

Once you have created a ViewDraw schematic, you can use the **SCH <-> max2** utility in the Max2 Express drawer to generate an EDIF netlist file from the schematic; process the EDIF Input File (**.edf**) with the MAX+PLUS ® II software to generate an EDIF Output File (**.edo**); and generate a **.vsm** file for simulation. The **SCH <-> max2** utility creates all necessary subdirectories and copies all of the files to the correct locations.

To use the **SCH <-> max2** utility, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II /Viewlogic Powerview Working Environment.

2. Create a ViewDraw schematic that follows the guidelines described in Creating ViewDraw Schematics for Use with MAX+PLUS II Software.

3. Start the **SCH <-> max2** utility by double-clicking Button 1 on the **SCH <-> max2** icon in the Max2 Express Drawer.

4. Specify the *Input Schematic*, *Family*, *Max2 Synthesis Style*, and *Choose project direction* options in the **SCH <-> max2** dialog box and choose **OK** to generate the <*design name*>**.vsm** file for simulation in ViewSim. The **SCH <-> max2** utility generates the <*design name*>**.vsm** file in the **sim** subdirectory of the **max2** directory.

5. If necessary, correct any errors in the ViewDraw schematic and recompile the project.

6. Simulate your project, as described in Performing a Timing Simulation with ViewSim Software.

Related Topics:

- Go to Performing Timing Verification for EDIF Output Files (.edo) with MOTIVE & MOTIVE for Powerview Software or Performing Timing Verification of Verilog Output Files (.vo) with MOTIVE Software in these MAX+PLUS II ACCESS℠ Key topics for related information.

Using the Max2 Express Drawer's VHDL <-> max2 Utility

Once you have created a VHDL Design File (**.vhd**) for your project, you can use the **VHDL <-> max2** utility in the Max2 Express drawer to synthesize and optimize the design; generate an EDIF netlist file; and process the EDIF netlist file with the MAX+PLUS II Compiler to generate an EDIF Output File (**.edo**) for simulation. The **VHDL <-> max2** utility creates all necessary subdirectories and copies all files to the correct locations.

To use the **VHDL <-> max2** utility, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the

MAX+PLUS II /Viewlogic Powerview Working Environment.

2. Create a VHDL Design File that follows the guidelines described in Creating VHDL Designs for Use with MAX+PLUS II Software.

3. Start the **VHDL <-> max2** utility by double-clicking Button 1 on the **VHDL <-> max2** icon in the Max2 Express Drawer.

4. Specify the *Input VHDL file*, *Viewlogic Optimize Style*, *Viewlogic Timing Constraint File*, *Altera Device Family*, *Max2 Synthesis Style*, and the *Process Direction* options in the **VHDL <-> max2** dialog box and choose **OK**. The **VHDL <-> max2** utility generates the <*design name*>**.vsm** file for simulation with ViewSim in the **sim** subdirectory of the **max2** directory.

5. If necessary, correct any errors in the VHDL Design File and recompile the project.

6. Simulate your project, as described in Performing a Timing Simulation with ViewSim Software.

Related Topics:

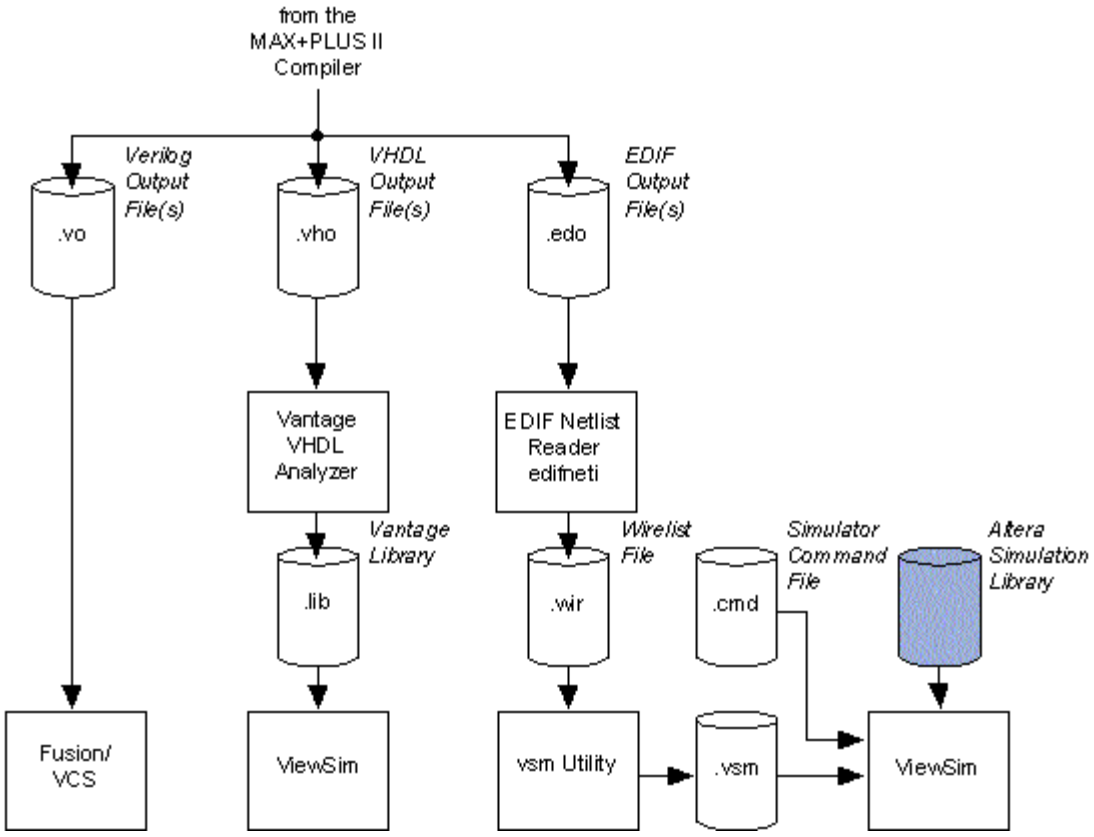
- Go to Performing Timing Verification for EDIF Output Files (.edo) with MOTIVE & MOTIVE for Powerview Software or Performing Timing Verification of Verilog Output Files (.vo) with MOTIVE Software in these MAX+PLUS II ACCESS^{SM} Key topics for related information.

MAX+PLUS II/Viewlogic Powerview Simulation Flow

Figure 1 shows the project simulation flow for the MAX+PLUS® II/Viewlogic Powerview interface.

Figure 1. MAX+PLUS II/Viewlogic Powerview Project Simulation Flow

Altera-provided items are shown in blue.



Analyzing VHDL Files with the SpeedWave VHDL Analyzer Software

You can use the SpeedWave VHDL Analyzer software to analyze VHDL Design Files (**.vhd**) prior to functional (or gate-level) simulation with ViewSim software, or to synthesis and optimization with ViewSynthesis software. You can also use the SpeedWave VHDL Analyzer to analyze a MAX+PLUS® II -generated VHDL Output File (**.vho**) prior to post-compilation timing simulation with ViewSim software. The **max2_VantgMgr** and **max2_VantgAnlz** tools are located in the Altera® Toolbox Design Tools Drawer.

To analyze a VHDL file with the SpeedWave VHDL Analyzer, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II /Viewlogic Powerview Working Environment.

2. If you wish to analyze a VHDL Design File (**.vhd**), create a VHDL file <*design name*>**.vhd** using the MAX+PLUS II Text Editor or another standard text editor and save it in a working directory. Go to Creating VHDL Designs for Use with MAX+PLUS II Software for more information.

3. If you wish to analyze a MAX+PLUS II-generated VHDL Output File (**.vho**), be sure to select *VHDL 1987* for the *VHDL Version* option and *VHDL Output File (.vho)* for the *Write Delay Constructs To* option in the **VHDL Netlist Writer Settings** dialog box (Interfaces menu) when you set up the MAX+PLUS II Compiler to generate a VHDL Output File. See Compiling Projects with MAX+PLUS II Software for more information on generating VHDL Output Files.

4. If your VHDL file contains functions from the **alt_mf** library, follow these steps:

    1. Start the Vantage Manager by double-clicking Button 1 on the **max2_VantgMgr** icon in the Design Tools Drawer.

2. Use the Vantage VHDL Library Manager to create an **alt_mf.lib** library file with the symbolic name `ALT_MF`.

3. Make **alt_mf** the working library with the **Set Working** command (Edit menu).

4. Start the VHDL Analyzer by double-clicking Button 1 on the **max2_VantgAnlz** icon in the Design Tools Drawer.

5. Analyze each VHDL file in the **alt_mf/src** directory into the **alt_mf.lib** working library. Source files are located in the **/usr/maxplus2/vwlogic/library/alt_mf/src** directory that is created by installing the Altera/Viewlogic interface.

5. If it is not already running, start the Vantage VHDL Library Manager, as described in step 4b, to create a Vantage library.

6. Choose the **List system libs** button.

7. Add the **ieee.lib** and **synopsys.lib** system libraries to your project:

    1. Select the **ieee.lib** and **synopsys.lib** libraries from the *Available Libraries* window and choose **Add lib**. Choose the **ieee** library from the **libs_syn** directory, which is located at **/<*Powerview system directory*>/ standard/van_vss/pgm/libs_syn**. The **ieee** library contains Synopsys package files.

    2. If your project uses functions from the **alt_mf** library, also select the **alt_mf.lib** file from the *Available Libraries* window and choose **Add lib**.

    3. Choose **Create Library** (File menu, type the project directory name in the *Symbolic Name* field, and choose **OK**.

8. Specify the project directory as the working directory by choosing **Set Working** (Edit menu).

9. Choose **Save INI File** (File menu).

10. Choose **Dismiss Window** (Powerview Red-Box menu).

11. Specify the appropriate path and file name in the **Analyzer VHDL Source File** dialog box and choose **OK** to analyze the VHDL file.

12. Once you have analyzed the file, perform one or more of the following tasks, as appropriate:

    - Performing a Functional Simulation with ViewSim Software
    - Synthesizing & Optimizing VHDL Designs with ViewSynthesis Software
    - Performing a Timing Simulation with ViewSim Software

Related Topics:

- Refer to the following sources for related information:
    - The ***Viewlogic ViewSim/VHDL User's Guide*** and ***ViewSim/VHDL Tutorial*** for information on using the Vantage VHDL Analyzer software or Vantage VHDL Library Manager
    - Powerview Command-Line Syntax in these MAX+PLUS II ACCESS^{SM} Key topics

---

Performing a Functional Simulation with ViewSim Software

You can use Viewlogic ViewSim software to perform a functional simulation of a ViewDraw schematic or a VHDL Design File (**.vhd**) before compiling your project with the MAX+PLUS II Compiler. Follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II /Viewlogic Powerview Working Environment.

2. Create a ViewDraw schematic that follows the guidelines in Creating ViewDraw Schematics for Use with MAX+PLUS II Software. Then go to step 3.

   or:

   Create a VHDL Design File <*design name*>**.vhd** and analyze it, as described in the following MAX+PLUS II ACCESS^{SM} Key topics:

   - Creating VHDL Designs for Use with MAX+PLUS II Software
   - Analyzing VHDL Files with the Vantage VHDL Analyzer Software

   Then go to step 7.

3. With the schematic open in the ViewDraw editor, add `CLR` and `PRE` inputs to any flipflops in your design, or tie the `CLR` and `PRE` ports of the flipflops to VCC. (Use the `PWR` primitive from the **builtin** library.)

4. Choose **Write To** (File menu) and save the schematic as <*design name*>**_funct**.

5. Start the **vsm** utility by double-clicking Button 1 on the **max2_vsmnet** icon in the Altera® Toolbox Design Tools Drawer.

6. Specify the following options in the **vsm** dialog box and choose **OK** to generate the <*design name*>**_funct.vsm** file:

Option:	**Setting:**
Design Name	*<design name>_funct*
Level	(blank)

7. Create a simulation command file (**.cmd**) for simulation with ViewSim software. Alternatively, you can enter commands at the prompt in the ViewSim window. Refer to your Viewlogic documentation for more information on creating ViewSim command files.

8. Start the ViewSim simulation tool by double-clicking Button 1 on the **max2_VSim** icon in the Design Tools Drawer.

9. If you wish to simulate a ViewDraw schematic, specify the following options in the **ViewSim** dialog box, then go to step 11.

Option:	**Setting:**
Design Name	*<design name>_funct*
Command File	*<design name>_funct.cmd*
VHDL Source Window	*OFF*
VHDL Debugging	*OFF*

10. If you wish to simulate a VHDL design, specify the following options in the **ViewSim** dialog box:

Option:	Setting:
Design Name	*<design name>*
Command File	*<design name>.cmd*
Graphical Interface	*ON*
VHDL Source Window	*OFF* or *ON*
VHDL Debugging	*OFF* or *ON*

11. Choose **OK** to simulate the design. ViewSim software simulates the design and starts the ViewTrace waveform editor to allow you to observe the simulation results.

12. Use the **edifneto** utility to generate an EDIF Netlist File (**.edf**) that can be imported into the MAX+PLUS II software, as described in Converting ViewDraw Schematics or VHDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the **edifneto** Utility.

Related Topics:

- Go to ViewSim documentation for complete details on simulating a project and using ViewTrace to observe waveform output results.

Initializing Registers in VHDL & Verilog Output Files for Power-Up before Simulation

Altera provides the **add_dc** script, which is availiable in the MAX+PLUS II system directory, to allow you to process MAX+PLUS II-generated Verilog Output Files (**.vo**) and VHDL Output Files (**.vho**) to prepare these files for simulation with another EDA tool. The add_dc script runs the **add_dclr** utility, which inserts a `device_clear` signal that is used for power-up initialization of all registers or flipflops in the design.

The script adds in a top-level signal named `device_clear` and connects it to the `CLRN` pin in all flipflops that should initialize to 0, and to the `PRN` pin of all flipflops that should initialize to 1. If the `CLRN` or `PRN` pin of a flipflop is already being used (i.e., is already connected to a signal), the script modifies the Verilog Output File or VHDL Output File so that the `AND` of the original signal and the `device_clear` pin feed the `CLRN` or `PRN` pin.

To use the **add_dc** script to process Verilog Output Files and VHDL Output Files before simulation with another EDA tool, follow these steps:

1. Make sure that your design file is located in the current directory, or change to the directory in which the design file is located.

2. Type the following command at the command prompt:

   Â¥*<path name of **add_dc.bat** file>*Â¥add_dc *<design name>* *<path name of **add_dclr.exe** file>* ↵

For example, if the both the **add_dc.bat** and the **add_dclr.exe** files are located in the **d:Â¥maxplus2Â¥exew** directory, and the **d:Â¥maxplus2Â¥exew** directory is specified in the search path, you can type the following command at a command prompt to add a `device_clear` signal to a design named `myfifo` in the file **myfifo.vo**:

```
add_dc myfifo d:Â¥maxplus2Â¥exew ↵
```

1. The **add_dc** script gives a message if the directory contains both a VHDL Output File and

a Verilog Output File with the same name (<*design name*>**.vo** and <*design*>**.vho**). You should delete or rename whichever of those files should not have the `device_clear` signal added. The **add_dc** script can modify only one design file at a time.

2. When the **add_dc** script processes the Verilog Output File or VHDL Output File, it creates a backup copy of the original file, with the extension **.ori**.

3. The **add_dc** script works only for Verilog Output Files and VHDL Output Files that are generated by MAX+PLUS II.

After you have used the **add_dc** script and are ready to simulate the resulting Verilog Output File or VHDL Output File with another EDA tool, you should assert the active low `device_clear` pin for a period of time that is long enough for the design to initialize. You can then de-assert the pin, and apply simulation vectors to the design.

Performing a Timing Simulation with ViewSim Software

After you have entered a design and compiled it with the MAX+PLUS® II Compiler, you can simulate a MAX+PLUS II-generated EDIF Output File (**.edo**) or VHDL Output File (**.vho**) with ViewSim software. ViewSim software can simulate both the functionality and the timing of your design. It also checks setup time, hold time, and Clock duty cycle timing requirements on registers.

To simulate a design with ViewSim software, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II /Viewlogic Powerview Working Environment.

2. Compile the design with the MAX+PLUS II software and generate an EDIF Output File (**.edo**) or VHDL Output File (**.vho**), as described in Compiling Projects with MAX+PLUS II Software.

3. In the Viewlogic Cockpit window, choose **Create** (Project menu) to open the **Create Project** dialog box. Type the name of your working directory and choose **OK**. You must create this new directory to avoid overwriting your original files when you generate new files for simulation.

4. Choose **SearchOrder** (Project menu) and add the appropriate directories and aliases to your **viewdraw.ini** file if you have not already done so. Go to Viewlogic Powerview **viewdraw.ini** Configuration File for more information.

   Refer to Viewlogic documentation for information on simulating projects that contain RAM functions. The procedure for reading an EDIF Output File and preparing it for simulation with ViewSim requires additional steps when the project contains RAM functions.

5. If you used the **SCH <-> max2** or **VHDL <-> max2** utility in the Max2 Express drawer to process your project, skip to step 8.

6. If you wish to simulate a VHDL Output File, follow the steps in Analyzing VHDL Files with the Vantage VHDL Analyzer then skip to step 7d.

7. If you are using the Altera® Toolbox Design Tools Drawer, follow these steps:

   1. To generate a Powerview wirelist from the EDIF Output File, double-click Button 1 on the **max2_edifi** icon in the Design Tools Drawer. The **Netlist In** dialog box is displayed.

2. In the **Netlist In** dialog box, specify *../<design name>* for the *EDIF Netlist File* option, then choose **OK** to process the EDIF netlist file.

3. If your project is implemented in multiple devices, repeat steps a and b for each EDIF Output File generated by the MAX+PLUS II Compiler, and ensure that the Altera-provided **alt_edif.cfg** file is specified for the *Attribute Swap Configuration File* option. In a multi-device project, the MAX+PLUS II Compiler generates a separate file for each device, plus a top-level file that is identified by "**_t**" appended to the project name. You must also follow the steps in Using ViewDraw & ViewGen Software to Prepare for Multi-Device Board-Level Simulation with ViewSim Software.

4. Start the **vsm** utility by double-clicking Button 1 on the **max2_vsmnet** icon in the Design Tools Drawer.

5. Specify your design name for the *Design Name* option in the **vsm** dialog box and choose **OK** to generate the *<design name>***.vsm** file.

8. Create a simulation command file (**.cmd**) for simulation with ViewSim software. Alternatively, you can enter commands at the prompt in the ViewSim window. Refer to your Viewlogic documentation for more information on creating ViewSim command files.

   ☞ The Altera simulation model library, **max2_sim**, allows you to use the `alt_grst` signal to asynchronously clear all flipflops (`DFFE` primitives).

9. Start the ViewSim simulation tool by double-clicking Button 1 on the **max2_VSim** icon in the Design Tools Drawer or the Max2 Express Drawer.

10. Specify the following options in the **ViewSim** dialog box and choose **OK** to simulate the design:

Option:	**Setting:**
Design Name	*<design name>*
Command File	*<design name>.cmd*
VHDL Source Window	*OFF*
VHDL Debugging	*OFF*

ViewSim software simulates the design and starts the ViewTrace waveform editor to allow you to observe the simulation results.

Related Topics:

- Refer to the following sources for related information:
  - ViewSim documentation for complete details on simulating a project and using ViewTrace to observe waveform output results
  - Using ViewDraw & ViewGen Software to Prepare for Multi-Device Board-Level Simulation with ViewSim Software

---

Using ViewDraw & ViewGen Software to Prepare for Multi-Device Board-Level Simulation with ViewSim Software

In order to perform board-level simulation with ViewSim software, you must generate symbols that represent each MAX+PLUS® II -generated EDIF Output File (**.edo**) and incorporate them into a top-level ViewDraw schematic. You can use ViewGen to generate hollow-body symbols to represent each EDIF Output File, and connect them to other system components in the top-level schematic. You must

also edit the wirelist files (**.wir**) created by the **edifneti** utility.

To prepare for multi-device board-level simulation with ViewSim software, follow these steps:

1. Perform steps 1 through 6c in Performing a Timing Simulation with ViewSim Software.
2. Start ViewGen by double-clicking Button 1 on the **max2_VGen** icon in the Design Tools Drawer.
3. Specify the filename of one of the EDIF Output Files <*filename*>.**edf** in the *Name* box in the **ViewGen** dialog box and choose **OK** to generate a corresponding <*filename*> symbol.
4. Repeat step 3 to generate other symbols as needed. You do not need to generate a symbol for the <*filename*>**_t.edf** file.
5. Eliminate the two extra pins for VDD and GND connections from the top-level wirelist file **./wir/**<*design name*>**_t.1**:
    1. Open the **./wir/**<*design name*>**_t.1** wirelist file with a standard text editor and delete the following lines:

        ```
        P IN GND
        I GND IN GND
        P IN VDD
        I VDD IN VDD
        ```

    2. Add the following two lines to the file to ensure global ground and power connections for simulation:

        ```
        G VDD↵
        G GND↵
        ```

    3. Save the top-level wirelist file with your changes.
6. Continue with the steps necessary to perform timing simulation, as described in Performing a Timing Simulation with ViewSim Software.

Performing a Timing Simulation with Fusion/VCS for Powerview Software

After you have compiled a project with the MAX+PLUS® II software to generate a VHDL Output File (**.vho**) and a Standard Delay Format (SDF) Output File (**.sdo**), you can perform a timing simulation with Fusion/VCS software.

To simulate a project with Fusion/VCS software, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II /Viewlogic Powerview Working Environment.
2. Generate a VHDL Output File (**.vdo**) and an SDF ver 2.1 or 1.0 Output File (**.sdo**) for your project, as described in Compiling Projects with MAX+PLUS II Software.
3. Create a new **sim** directory under your **max2** directory to contain your Fusion/VCS simulation-related files.
4. To use the SDF Output File with the Fusion/VCS software, create a PLI table file (**.tab**) in the ***<project name>*/max2/sim** directory that contains the following line:

`$sdf_annotate call=sdf_annotate_call acc=tchk, mp:`*<project name>*↵

5. Open the **Fusion/VCS** dialog box by choosing the **max2_VCS** button from the Altera® Design Tools Drawer in the Powerview Cockpit.
    1. Type *<project name>*`/max2/`*<project name>*`.vo` in the *Verilog Design and Object Files* box.
    2. Type *<project name>*`.tab` in the *PLI Table File* box.
    3. Type *<project name>*`/max2/alt_max2.vo` in the *Verilog Library File 1* box.
    4. (Optional) To use a command file or to set stimuli during simulation, select the *Debug* option in the *VCS* box and type the name of the command file in the *Simulation Command-file* box.

        ☞ When using a command file or setting stimuli, include the signal scope as part of the signal name. For example, to manipulate clk, a top-level signal in the **fadd** project, name the signal as `fadd.clk`.

    5. Choose **OK**.

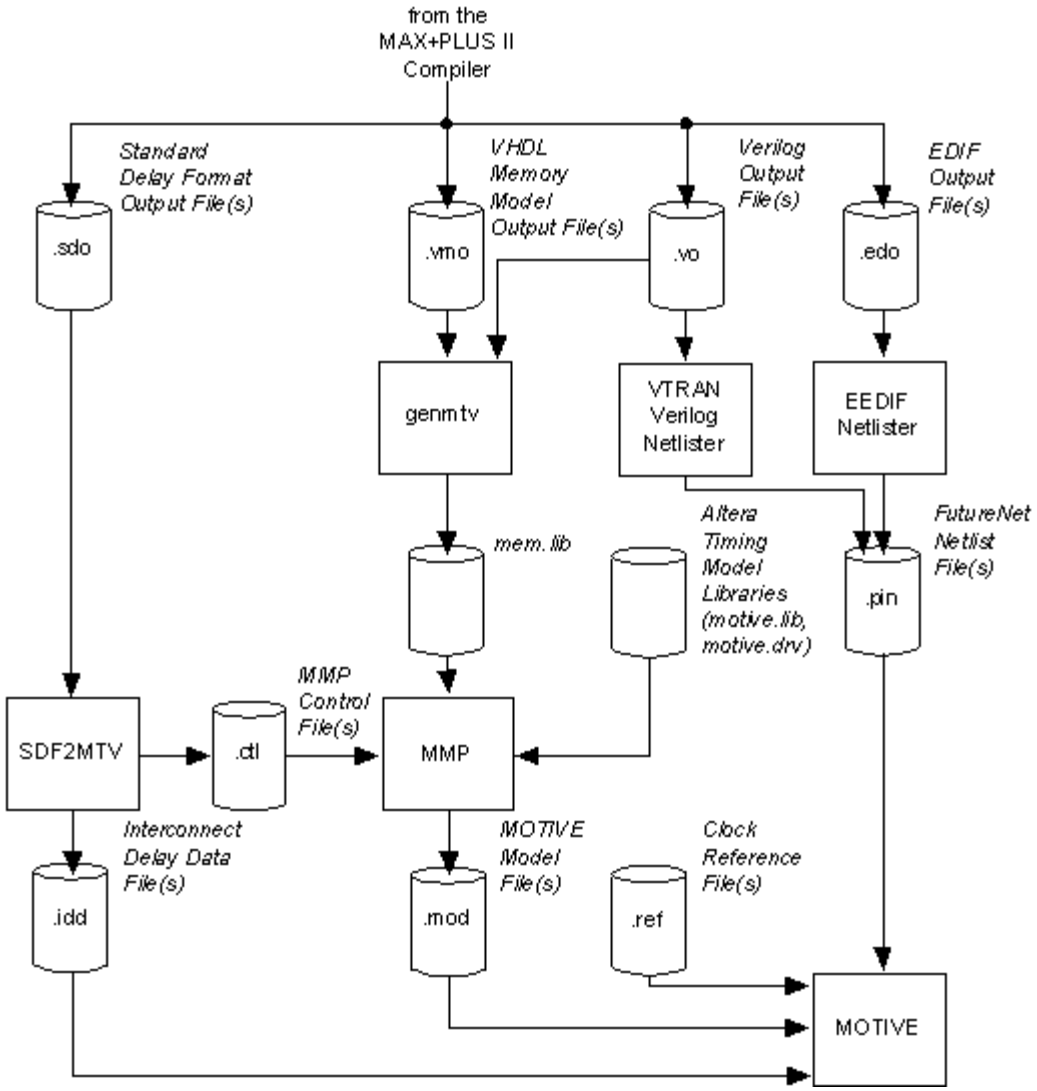
Related Topics:

- Go to Performing a Timing Simulation with ViewSim Software in these MAX+PLUS II ACCESS℠ Key topics for related information:

MAX+PLUS II/Viewlogic Powerview Timing Verification Flow

Figure 1 shows the timing verification flow for the MAX+PLUS® II/Viewlogic Powerview interface.

Figure 1. MAX+PLUS II/Viewlogic Powerview Project Timing Verification Flow



Performing Timing Verification of EDIF Output Files (.edo) with MOTIVE & MOTIVE for Powerview Software

After you have compiled a project and generated an EDIF Output File (**.edo**) with the MAX+PLUS® II software, you can use Viewlogic MOTIVE or MOTIVE for Powerview software to perform timing verification. The **max2_MOTIVE** tool is located in both the Altera® Toolbox Design Tools Drawer and the Altera Toolbox Max2 Express Drawer. The MOTIVE timing model library, **motive.lib**, provides models of basic primitives and the clklock megafunction for timing verification.

To perform timing verification for EDIF Output Files with MOTIVE or MOTIVE for Powerview software, follow these steps:

1. Set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Viewlogic Powerview Working Environment.

2. Generate an EDIF Output File (**.edo**) by compiling your design with the MAX+PLUS II software, as described in Compiling Projects with MAX+PLUS II Software.

3. Start the MOTIVE for Powerview software by double-clicking Button 1 on the **max2_MOTIVE** icon in the Altera Toolbox Design Tools Drawer. The MOTIVE for

Powerview Control Panel opens.

4. Choose **Setup Environment** (File menu) to open the **Environment Parameters** dialog box, and specify the following options:

    1. Specify the directory for the *Project Directory* option.

    2. Specify */usr/maxplus2/vwlogic/library/alt_time/motive.lib* for the *Model Library Search Path* option.

    3. Select *EDIF* for the *Netlist Input Format* option.

    4. Choose **Accept**. The MOTIVE for Powerview software automatically creates a **tim** subdirectory, which contains MOTIVE design cases and related files, in the current working directory.

5. Choose **Save Parameters** (File menu) to save your customized project setup.

6. To specify the project name, choose the **New Design** button to open the **Adding a New Design** dialog box. Type the design name in the *New Design* box. Choose **Accept**, then **Dismiss**.

7. To specify the case name, choose the **New Case** button to open the **Adding a New Case** dialog box. Type the case name in the *New Case* box. Select *Default* as the *New Case Type.* Choose **Accept**, then **Dismiss**.

8. Choose **Browse Cases** (File menu) to open the **Case Display** dialog box. In the **Case Display** dialog box, double-click Button 1 on the field that contains the case for the project. Double-clicking on the field opens a file manager listing all the project files located under that case. Choose **Dismiss** in the **Case Display** dialog box.

    1. Choose the **Get File** button from the file manager to display the *Get File* box at the bottom of the window. This box allows you to specify which file(s) you would like to add to the list of files for the current case.

    2. Type /<*working directory*>/<*project name*>`.edo` in the *Get File* box and choose **Copy**. The new file appears in the list of design files.

    3. Type /<*working directory*>/<*project name*>`.sdo` in the *Get File* box and choose **Copy**.

    4. Type /<*working directory*>/<*project name*>`.ref` in the *Get File* box and choose **Copy**.

    5. If your project contains memory functions, such as `ram`, `rom`, `dpram`, `scfifo`, `dcfifo`, `altdpram`, or `clklock`, type <*project name*>**.vmo** in the *Get File* box and choose **Copy** to add the MAX+PLUS II-generated VHDL Memory Model Output File (**.vmo**) to the list of files for the case. The MAX+PLUS II Compiler automatically generates this file for a project that contains memory functions.

Every MOTIVE analysis requires a MOTIVE Clock Reference File (**.ref**). If the project is simple, you can create the file in the Setup Advisor. Otherwise, you must create the file in a text editor using MOTIVE syntax. For more information on the purpose, function, and syntax of MOTIVE Clock Reference Files, see the ***MOTIVE System Reference***.

6. Choose **Dismiss**.

9. Choose the **Netlister** button in the MOTIVE for Powerview Control Panel to open the **EDIF Netlist Parameters** dialog box. To create a FutureNet Format Netlist File (**.pin**) with the EEDIF Netlister for your design, follow these steps:

    1. Choose the **Select Design** button to open the **Select Design** dialog box.

    2. Double-click Button 1 on the project name to open the **Select Case** dialog box.

    3. Double-click Button 1 on the case name in the **Select Case** dialog box to open the **Select File** dialog box.

    4. Double-click Button 1 on the EDIF Output File, <*project name*>**.edo**, in the **Select File** dialog box.

    5. Select *Keep* for all *Case Sensitivity* options in the *EDIF Netlist Parameters* dialog box.

    6. Choose **Accept**, then **Dismiss** to close the **EDIF Netlist Parameters** dialog box.

10. Choose the **SDF2MTV** button in the Control Panel to open the **SDF2MTV (MOTIVE SDF Reader) Parameters** dialog box and specify the following options:

    1. Choose the **Select** button next to the *SDF Filename* box to open the **Select File** dialog box.

    2. Double-click Button 1 on the project's Standard Delay Format (SDF) Output File, <*project name*>**.sdo**, in the **Select File** dialog box. The **SDF2MTV** utility creates a MOTIVE Model Pre-Processor (MMP) Control File (**.ctl**) that allows you to annotate the parameterized library, and an Interconnect Delay Data File (**.idd**).

    3. Choose **Accept**, then **Dismiss** to close the **Select File** dialog box.

11. If your project contains `ram`, `rom`, `dpram`, `scfifo`, `dcfifo`, `altdpram`, or `clklock` megafunctions, use the **genmtv** utility to back-annotate the MMP Control File and to allow the MMP Control File to recognize the function. The input to the **genmtv** utility is the VHDL Memory Model Output File (**.vmo**) described above. From the ***/<working directory>/<project name>/<case name>*** directory, type the following command at the UNIX prompt:

    `genmtv` <*project name*> ↵

12. If your project contains RAM or ROM functions and you turned on the *Flatten Bus* option in the MAX+PLUS II Compiler's **EDIF Netlist Writer Settings** dialog box when you compiled your project, you must edit the **mem.lib** file, i.e., the MOTIVE Model Pre-Processor timing library file created with the **genmtv** utility. You must remove bracket `[ ]` characters from all occurrences of the address bus, e.g., change `A[0]` to `A0`, in both the `INPUTS` and `MIXED` sections of every RAM and ROM cell definition in **mem.lib**.

13. Choose the **MMP** button from the Control Panel to open the **MOTIVE Model Pre-processor (MMP) Parameters** dialog box and specify the following options:

    1. Choose the **Select** button next to the *MMP Ctl File* box to open the **Select File** dialog box.

    2. Double-click Button 1 on the project's MMP Control File, <*project name*>**.ctl**, in the **Select File** dialog box.

3. In the **MOTIVE Model Pre-processor (MMP) Parameters** dialog box, choose the **Setup Model Libraries** button to display boxes on the right side of the dialog box that allow you to list additional source model libraries. In one of these boxes, type the following path and filename:

   `/usr/maxplus2/vwlogic/library/alt_time/motive.drv` ↵

4. If your project contains RAM or ROM functions, repeat step 13c but specify the pathname of the **mem.lib** file created in step 12. For example:

   `/usr/maxplus2/`*<working directory>***`/..../`***<case name>***`/mem.lib`** ↵

5. In the **MOTIVE Model Pre-processor (MMP) Parameters** dialog box, choose **Accept**, then **Dismiss**. The **MMP** utility creates a design-specific Timing Model Library File (**.mod**).

14. Choose the **Analyze** button from the Control Panel to expand the Control Panel.

15. Double-click Button 1 on the project name in the *Select Design* box in the Control Panel to open the *Select Case* box.

16. Select the specific case of the project in the *Select Case* box and double-click Button 1 on the case name to open MOTIVE software and its Setup Advisor. The Setup Advisor helps guide you through the following steps to set up and configure a case analysis:

    1. In the Setup Advisor window, choose the **Continue** button to open the **Project Name Selection** dialog box, which displays the project name.

    2. Choose the **Begin analysis** button to open the **Checking for existing project** dialog box.

    3. Choose **Continue** to open the **Design Specific Flow(s)** dialog box and set up the project through the Setup Advisor. The *Design Name* option lists the project filename.

    4. Choose **Continue** to open the **Flow and Translation Selection** dialog box.

    5. Select the *Manual Translation Flow* option to specify input files and the steps to perform in the timing verification flow for MOTIVE software. Choose **Continue** to open the **Manual Flow Selection** dialog box and specify the following options:

Option:	**Setting:**
Netlist/Pinlist	*FutureNet (.pin)*
Parametric	*OVI Verilog (.sdf)*

       In the *Other* box, select *Use available MOTIVE files* to use the input files you created in previous steps. Choose **Continue** to open the **FutureNet Pinlist Preparation** dialog box.

    6. Type the project name in the *Root Block* box. Choose **Continue** to open the **OVI Standard Parametric Back-annotation** dialog box.

    7. Type <*project name*>**.sdo** in the *OVI (SDF) back-annotation file* box. Choose **Continue** to open the **MOTIVE Model Compilation** dialog box.

    8. Replace the entry in the *Control file(s)* box with <*project name*>`.ctl`. Type the

following two filenames, which must be separated by a space, in the *Libraries(s)* box:

```
/usr/maxplus2/vwlogic/library/alt_time/motive.lib
/usr/maxplus2/vwlogic/library/alt_time/motive.drv
```

9. If your project contains RAM or ROM functions, add the **mem.lib** file to the directories specified in step 16h.

10. Choose **Continue** to open the **Quick Definition of Existing MOTIVE Files** dialog box. The <*project name*>**.ref** filename appears in the *Clock Reference File (.ref)* box.

11. Replace the entry in the *Design's (pre-compiled) Model File (.mod)* box with <*project name*>*.mod.* Choose **Continue** to open the **Congratulations** dialog box.

12. Choose **Continue** to open the **Cleaning up** dialog box after completing the Setup Advisor interview. Select *Save under Project name* to save your setup, and choose **Continue** to close the Setup Advisor window.

17. In the MOTIVE window, choose **Verify** (Analyze menu) and then choose **Execute** to start verification. To view the output files, choose **Output Files** (View menu).

Performing Timing Verification of Verilog Output Files (.vo) with MOTIVE Software

After you have compiled a project and generated a Verilog Output File (**.vo**) with the MAX+PLUS II Software, you can use Viewlogic MOTIVE to perform timing verification. The MOTIVE timing model library, **motive.lib**, provides basic primitives and the `clklock` megafunction for timing verification.

To perform timing verification for Verilog Output Files with MOTIVE software, follow these steps:

1. Set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Viewlogic Powerview Working Environment.

2. Generate a Verilog Output File by compiling your design with the MAX+PLUS II software, as described in Compiling Projects with MAX+PLUS II Software.

3. Start the MOTIVE software by typing `motive` ↵ at the UNIX prompt. The MOTIVE Session Log and Setup Advisor windows are displayed. Choose **OK**.

4. Choose **Project** on the vertical menubar in the Setup Advisor, then choose the **Name** (Select project name) tab and specify the name of the project for *Project name*. The directory in which you started MOTIVE will be selected automatically for *Current directory*. Choose **Accept**. MOTIVE then searches for the <*project name*>**.stm** file. If this is a new file, a message will appear in the Session Log window that mentions that MOTIVE found a license and the message `could not open the <project name>.stm file -- assuming a new design.`

5. Choose **Flow** from the vertical menubar, then choose the **Type** (Select flow type) tab. Select the *Using Verilog and SDF* option and choose **Accept**.

6. Choose **Options** from the vertical menubar, then choose the **Options** (Miscellaneous usage options) tab. If desired, specify a different value for the *MOTIVE analysis cycle time*

option. Choose **Accept**.

7. Choose **Verilog** on the vertical menubar and specify the following Verilog HDL input options:

    1. Choose the **Translate** (Translate Verilog netlist file) tab. Specify the name of the MAX+PLUS II-generated Verilog Output File (**.vo**) for the *Verilog netlist* option. Choose the **Common Options** button to display the **Common Options** dialog box. Select the *Special Options* option and turn on the *Skip Behavioral Constructs* option. Type either `pinlist` or a period (`.`) for the *Generated pin files* option. Choose **OK** to close the **Common Options** dialog box and return to the **Translate** tab.

    2. Specify the location of the MAX+PLUS II-generated **alt_max2.vo** file for the *Vendor module definition* option. Choose the **Translate** button. The **Process Execution Log & Tips** dialog box displays the current status of the translation to **.pin** files. Choose **OK** after successful translation.

    3. Choose the **Import** (Confirm Adding hierarchical blocks) tab. Choose the **Import Blocks** button. The **MOTIVE Interaction Log & Tips** dialog displays the current import status. Choose **OK** after a successful completion.

    4. Select the **Hierarchy** (Configure hierarchy options) tab. Type the name of the rootblock for the *Rootblock of design* option, or choose the **Find Rootblock** button to display the rootblock name. Choose **Accept**.

8. Choose the **Check** (Review and/or build the netlist database) tab. Choose the **Incremental Build** button. The **MOTIVE Interaction Log & Tips** dialog displays the current build status. Choose **OK** after a successful completion.

9. Select **SDF** on the vertical menubar, then select the **Translate** (SDF model preparation) tab. Type <*project name*>`.sdo` for the *SDF file* option, making sure that you specify the **.sdo** extension. Type <*project name*>`.ctl` for the *MPP control file name*, and <*project name*>`.idd` for the *IDD file name*.

10. Choose the **Process SDF File** button.

11. If your project contains the `clklock` megafunction, use the **genmtv** utility to back-annotate the MPP Control File and to allow the MPP Control File to recognize the `clklock` function. The input to the **genmtv** utility is the Verilog netlist file (**.vo**). From the /<*working directory*>/<*project name*>/<*case name*> directory, type the following command at the UNIX prompt:

    `genmtv -v` <*project name*> ↵

12. If your project contains RAM or ROM functions and you turned on the *Flatten Bus* option the MAX+PLUS II Compiler's **Verilog Netlist Writer Settings** dialog box when you compiled your project, you must edit the **mem.lib** file, i.e., the MOTIVE Model Pre-Processor timing library file generated with the **genmtv** utility. You must remove the bracket `[ ]` characters from all occurrences of the address bus, e.g., change `A[0]` to `A0`, in both the `INPUTS` and `MIXED` sections of every RAM and ROM cell definition in **mem.lib**.

13. Select the **MPP** (MOTIVE model compilation) tab. Type <*project name*>`.ctl` for the *Control file* option. Type `/usr/maxplus2/viewlogic/library/alt_time/motive.lib /usr/maxplus2/viewlogic/library/alt_time/motive.drv` for the *Libraries* option. If the project contains memory functions, you should also specify the location of the **mem.lib** file for the *Libraries* option. Type <*project name*>`.mod` for the *Generated model file* option and <*project name*>`.rcf` for the *Revised control file* option. Choose the **RUN**

MMP button. The **MOTIVE Execution Log & Tips** dialog displays and shows the current status. Choose **OK** after a successful completion.

14. Select **Save** from the File menu in the Setup Advisor to write all the selections made so far to the <*project name*>**.stm** file.

15. Select **Clock** on the vertical menubar, then choose the **File** (Check reference file and timebase options) tab. The correct name of the Clock Reference File (**.ref**) should be displayed for the *Clock reference file* option. Choose **Accept**.

    ☞ Every MOTIVE analysis requires a MOTIVE Clock Reference File. If the project is simple, you can create the file in the Setup Advisor. Otherwise, you must create the file with a text editor using MOTIVE syntax. For more information on the purpose, function, and syntax of MOTIVE Clock Reference Files, see the ***MOTIVE System* Reference**.

16. Choose the **Edit** (Simple clock reference generation) tab. Specify the names for the *Clock reference* and *Clock net name* options. Choose **Generate**.

17. Choose the **Check** (Choose incremental definitions) tab, then choose the **Load Clock** button.

18. Choose **Finish** from the vertical menubar, then choose the **Build** button. The **MOTIVE Interaction Log & Tips** dialog displays the current status. Choose **OK** after a successful completion.

19. Select **Save** from the File menu in the Setup Advisor.

20. In the MOTIVE Session Log window, choose **Verify** (Analyze menu) and then choose the **Execute** button to start verification. To view the output files, choose **Output Files** (View menu).

Alternatively, you can run MOTIVE analysis on the command line by following these steps:

1. Type the following commands at the UNIX prompt:

   `vtran` <*project name*>`.vo -b -h -u alt_max2.vo` ↵ (generates **.pin** files)

   `sdf2mtv` <*project name*>`.sdfo` ↵ (generates **.ctl** files)

2. If your project contains `ram`, `rom`, `dpram`, or `clklock` functions, you should also type the following commands at the UNIX prompt:

   `genmtv -v` <*project name*> ↵

   `mmp` <project name>`.ctl -l`
   `/usr/maxplus2/viewlogic/library/alt_time/motive.lib -l`
   `/usr/maxplus2/viewlogic/library/alt_time/motive/drv -l mem.lib` ↵

3. Type the following command at the UNIX prompt:

   `amtv` <*project name*>

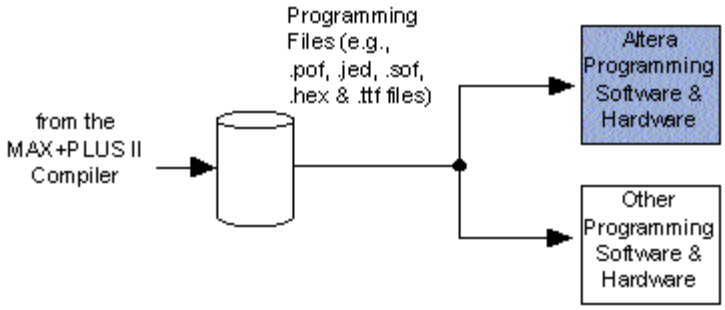
Programming Altera Devices

Once you have successfully compiled and simulated a project with the MAX+PLUS® II

software, you can program an Altera® device and test it in the target circuit. Figure 1 shows the device programming flow for MAX+PLUS II software.

Figure 1. MAX+PLUS II Device Programming Flow

Altera-provided items are shown in blue.



You can program devices with Altera programming hardware and MAX+PLUS II Programmer software installed on a 486- or Pentium-based PC or a UNIX workstation, or with programming hardware and software available from other manufacturers. Table 1 shows the available Altera programming hardware options on PCs and UNIX workstations.

Table 1. Altera Programming Hardware

Programming Hardware Option	PCs	UNIX Work-stations	MAX® 3000A Devices	Classic® & MAX 5000 Devices	MAX 7000 & MAX 7000E Devices	MAX 7000A, MAX 7000AE, MAX 7000B, MAX 7000S MAX 9000 & MAX 9000A Devices	FLEX® 6000, FLEX 6000A, FLEX 8000, FLEX 10K, FLEX 10KA, FLEX 10KB, & FLEX 10KE Devices	In-System Programming/ Configuration
Logic Programmer card, PL-MPU Master Programming Unit, and device-specific adapters	✓		✓	✓	✓	✓		
BitBlaster Download Cable	✓	✓	✓			✓	✓	✓
ByteBlasterMV	✓		✓			✓	✓	✓

Download Cable							
MasterBlaster Download Cable	✓	✓	✓		✓	✓	✓

If you wish to transfer programming files from a UNIX workstation to a PC over a network with File Transfer Protocol (FTP) or other similar transfer programs, be sure to select binary transfer mode.

Programming hardware from other manufacturers varies, but typically consists of a device connected to one of the serial ports on the workstation. Various vendors, such as Data I/O and BP Microsystems, supply hardware and software for programming Altera devices.

Related Topics:

- Go to Compiling Projects with MAX+PLUS II Software for information on creating programming files.
- Go to the following topics, which are available on the web, for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware
  - FLEX Devices
  - MAX Devices
  - Classic Device Family

Powerview Command-Line Syntax

Table 1 shows the command-line syntax for using Powerview functions.

Table 1. Powerview Command-Line Syntax

Action	Command
Start VHDL Analyzer software	`vhdl -v` <*project name*>
Start ViewSynthesis software	`vhdldes`
Load Altera® technology library	`vhdldes> technology altera`
Compile a VHDL design	`vhdldes> vhdl` <*project name*>
Synthesize a design	`vhdldes> synthesize`
Generate wirelist file	`vhdldes> wir`
Create a schematic representation	`vhdldes> viewgen`
Generate a synthesis report file	`vhdldes> report`
Start the graphical user interface for ViewSynthesis	`vhdldes> vdesgui`
Start the VHDL-to-symbol utility	`vhdl2sym` <*project name*>
Start **vsm**	`vsm` <*project name*>
Start ViewSim simulator	`viewsim` <*project name*> -<*project name*>`.cmd`
Start **edifneto**	`edifneto -f` <*project name*>-`l` (`std` *or* `altera`) <*project name*>`.edf`
Start Vantage VHDL Analyzer software	`analyze -src` <*design file*>
Start MOTIVE for Powerview software	`mfp`

Entering Resource Assignments

The MAX+PLUS® II software allows you to enter a variety of resource and device assignments for your projects. Resource assignments are used to assign logic functions to a particular pin, logic cell, I/O cell, embedded cell, row, column, Logic Array Block (LAB), Embedded Array Block (EAB), chip, clique, local routing, logic option, timing requirement, or connected pin group. In the MAX+PLUS II software, you can enter all types of resource and device assignments with Assign menu commands. You can also enter pin, logic cell, I/O cell, embedded cell, LAB, EAB, row, and column assignments in the MAX+PLUS II Floorplan Editor. The Assign menu commands and the Floorplan Editor all save assignment information in the ASCII Assignment & Configuration File (**.acf**) for the project. In addition, you can edit ACFs manually in any standard text editor.

ViewDraw Schematics

In ViewDraw schematics, you can assign a limited subset of these resource assignments by assigning properties to symbols. These properties are incorporated into the EDIF netlist file(s). The MAX+PLUS II software automatically converts assignment information from the EDIF Input File (**.edf**) into the ACF format. For information on making MAX+PLUS II-compatible resource assignments, go to the following topics:

- Assigning Pins, Logic Cells & Chips
- Assigning Cliques
- Assigning Logic Options
- Modifying the Assignment & Configuration File with the **setacf** Utility

Installing the Altera-provided MAX+PLUS II/Viewlogic interface on your computer automatically creates the following sample ViewDraw schematic file, which includes resource assignments:

- **/usr/maxplus2/examples/viewlogic/example4/fadd2mpp**

Related Topics:

- Go to Viewlogic documentation for information on how to assign properties. Go to "Resource Assignments in EDIF Input Files" and "Assigning Resources in a Third-Party Design Editor" in MAX+PLUS II Help for more information on assignments or properties that can be assigned in ViewDraw.

VHDL Design Files

For VHDL-based designs, you must use the MAX+PLUS II software or the **setacf** utility to enter resource assignments. For information on using the **setacf** utility, go to Modifying the Assignment & Configuration File with the **setacf** Utility.

Related Topics:

- For information on entering assignments in the MAX+PLUS II software with Assign menu commands or in an ACF, go to "resource assignments" or "ACF, format" in MAX+PLUS II Help using **Search for Help on** (Help menu).

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Using ViewDraw & ViewGen Software to Prepare for Multi-Device Board-Level Simulation with ViewSim Software

In order to perform board-level simulation with ViewSim software, you must generate symbols that represent each MAX+PLUS® II -generated EDIF Output File (**.edo**) and incorporate them into a top-level ViewDraw schematic. You can use ViewGen to generate hollow-body symbols to represent each EDIF Output File, and connect them to other system components in the top-level schematic. You must also edit the wirelist files (**.wir**) created by the **edifneti** utility.

To prepare for multi-device board-level simulation with ViewSim software, follow these steps:

1. Perform steps 1 through 6c in Performing a Timing Simulation with ViewSim Software.
2. Start ViewGen by double-clicking Button 1 on the **max2_VGen** icon in the Design Tools Drawer.
3. Specify the filename of one of the EDIF Output Files <*filename*>.**edf** in the *Name* box in the **ViewGen** dialog box and choose **OK** to generate a corresponding <*filename*> symbol.
4. Repeat step 3 to generate other symbols as needed. You do not need to generate a symbol for the <*filename*>**_t.edf** file.
5. Eliminate the two extra pins for VDD and GND connections from the top-level wirelist file **./wir/**<*design name*>**_t.1**:
    1. Open the **./wir/**<*design name*>**_t.1** wirelist file with a standard text editor and delete the following lines:

        ```
        P IN GND
        I GND IN GND
        P IN VDD
        I VDD IN VDD
        ```

    2. Add the following two lines to the file to ensure global ground and power connections for simulation:

        ```
        G VDD↵
        G GND↵
        ```

    3. Save the top-level wirelist file with your changes.
6. Continue with the steps necessary to perform timing simulation, as described in Performing a Timing Simulation with ViewSim Software.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Creating VHDL Designs for Use with MAX+PLUSÂ II Software

You can create VHDL design files with the MAX+PLUS$^{Â®}$ Â II Text Editor or another standard text editor and save them in the appropriate directory for your project. The MAX+PLUSÂ II Text Editor offers the following advantages:

- VHDL templates are available with the **VHDL Templates** command (Templates menu). These templates are also available in the ASCII **vhdl.tmp** file, which is located in the **/usr/maxplus2** directory.
- If you use the MAX+PLUSÂ II Text Editor to create your VHDL design, you can use the **Syntax Coloring** command (Options menu). The Syntax Coloring feature displays keywords and other elements in text files in different colors to distinguish them from other forms of syntax.

To create a VHDL design that can be synthesized and optimized with ViewSynthesis software, follow these steps:

1. You can instantiate the following Altera-provided logic functions in your VHDL design:
   - The **alt_mf** library contains the Altera$^{Â®}$ VHDL logic function library, which includes MAX+PLUSÂ II-specific primitives and the `a_8count`, `a_8mcomp`, `a_8fadd`, and `a_81mux` macrofunctions. If you wish to instantiate **alt_mf** logic functions in your VHDL design, you must first analyze all functions in the **alt_mf/src** directory. See Analyzing VHDL Files with the Vantage VHDL Analyzer Software for details.
   - The `clklock` megafunction, which enables the phase-locked loop, or ClockLock™ , circuitry available on selected Altera FLEX$^{Â®}$ Â 10K devices. Go to Instantiating the clklock Megafunction in VHDL or VerilogÂ HDL for information.
   - MegaCore™ functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP™). The OpenCore™ feature in the MAX+PLUSÂ II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.
2. (Optional) To enter resource assignments in your VHDL design, go to Entering Resource Assignments. You can also enter resource assignments from within the MAX+PLUSÂ II software.

Once you have created a VHDL design, you can analyze it, synthesize it, and generate an EDIF netlist file that can be imported into the MAX+PLUSÂ II software with either of the following methods:

- You can analyze, functionally simulate, and synthesize the VHDL design, then generate an EDIF netlist file by following the steps in these topics:
  - Analyzing VHDL Files with the Vantage VHDL Analyzer Software
  - Performing a Functional Simulation with ViewSim Software
  - Synthesizing & Optimizing VHDL Designs with ViewSynthesis Software
  - Converting ViewDraw Schematics or VHDL Designs into MAX+PLUSÂ II-Compatible EDIF Netlist Files with the **edifneto** Utility
- You can use the **VHDL <-> max2** utility in the Max2 Express Drawer to automatically analyze and synthesize the VHDL design, compile it with the MAX+PLUSÂ II Compiler, generate an EDIF Output File (**.edo**), and create a **.vsm** file for simulation. See Using the Max2 Express Drawer's **VHDL <-> max2** Utility in these MAX+PLUSÂ II ACCESS^{SM} Key topics for details.

Installing the Altera-provided MAX+PLUSÂ II/Viewlogic Powerview interface on your computer automatically creates the following sample VHDL files:

- **/usr/maxplus2/examples/viewlogic/example5/count4.vhd**

- **/usr/maxplus2/examples/viewlogic/example5/count8.vhd**

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Analyzing VHDL Files with the Vantage VHDL Analyzer Software

You can use the Vantage VHDL Analyzer software to analyze VHDL Design Files (**.vhd**) prior to functional (or gate-level) simulation with ViewSim software, or to synthesis and optimization with ViewSynthesis software. You can also use the Vantage VHDL Analyzer to analyze a MAX+PLUS® II -generated VHDL Output File (**.vho**) prior to post-compilation timing simulation with ViewSim software. The **max2_VantgMgr** and **max2_VantgAnlz** tools are located in the Altera® Toolbox Design Tools Drawer.

To analyze a VHDL file with the Vantage VHDL Analyzer, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II /Viewlogic Powerview Working Environment.
2. If you wish to analyze a VHDL Design File (**.vhd**), create a VHDL file <*design name*>**.vhd** using the MAX+PLUS II Text Editor or another standard text editor and save it in a working directory. Go to Creating VHDL Designs for Use with MAX+PLUS II Software for more information.
3. If you wish to analyze a MAX+PLUS II-generated VHDL Output File (**.vho**), be sure to select *VHDL 1987* for the *VHDL Version* option and *VHDL Output File (.vho)* for the *Write Delay Constructs To* option in the **VHDL Netlist Writer Settings** dialog box (Interfaces menu) when you set up the MAX+PLUS II Compiler to generate a VHDL Output File. See Compiling Projects with MAX+PLUS II Software for more information on generating VHDL Output Files.
4. If your VHDL file contains functions from the **alt_mf** library, follow these steps:
    1. Start the Vantage Manager by double-clicking Button 1 on the **max2_VantgMgr** icon in the Design Tools Drawer.
    2. Use the Vantage VHDL Library Manager to create an **alt_mf.lib** library file with the symbolic name `ALT_MF`.
    3. Make **alt_mf** the working library with the **Set Working** command (Edit menu).
    4. Start the VHDL Analyzer by double-clicking Button 1 on the **max2_VantgAnlz** icon in the Design Tools Drawer.
    5. Analyze each VHDL file in the **alt_mf/src** directory into the **alt_mf.lib** working library. Source files are located in the **/usr/maxplus2/vwlogic/library/alt_mf/src** directory that is created by installing the Altera/Viewlogic interface.
5. If it is not already running, start the Vantage VHDL Library Manager, as described in step 4b, to create a Vantage library.
6. Choose the **List system libs** button.
7. Add the **ieee.lib** and **synopsys.lib** system libraries to your project:
    1. Select the **ieee.lib** and **synopsys.lib** libraries from the *Available Libraries* window and choose **Add lib**. Choose the **ieee** library from the **libs_syn** directory, which is located at **/<*Powerview system directory*>/ standard/van_vss/pgm/libs_syn**. The **ieee** library contains Synopsys package files.

2. If your project uses functions from the **alt_mf** library, also select the **alt_mf.lib** file from the *Available Libraries* window and choose **Add lib**.

3. Choose **Create Library** (File menu, type the project directory name in the *Symbolic Name* field, and choose **OK**.

8. Specify the project directory as the working directory by choosing **Set Working** (Edit menu).

9. Choose **Save INI File** (File menu).

10. Choose **Dismiss Window** (Powerview Red-Box menu).

11. Specify the appropriate path and file name in the **Analyzer VHDL Source File** dialog box and choose **OK** to analyze the VHDL file.

12. Once you have analyzed the file, perform one or more of the following tasks, as appropriate:

    - Performing a Functional Simulation with ViewSim Software
    - Synthesizing & Optimizing VHDL Designs with ViewSynthesis Software
    - Performing a Timing Simulation with ViewSim Software

Related Links:

- Refer to the following sources for related information:
  - The ***Viewlogic ViewSim/VHDL User's Guide*** and ***ViewSim/VHDL Tutorial*** for information on using the Vantage VHDL Analyzer software or Vantage VHDL Library Manager
  - Powerview Command-Line Syntax in these MAX+PLUS II ACCESS^{SM} Key topics

---

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Using Viewlogic ViewSim & MAX+PLUS® II Software

The following topics describe how to use the Viewlogic ViewSim software with MAX+PLUS® II software. Choose one of the following topics for information:

Open a printable version of all topics listed on this page.

Setting Up the MAX+PLUS II/Viewlogic Powerview Working Environment

- Software Requirements
- MAX+PLUS II/Viewlogic Powerview Interface File Organization
- Viewlogic Powerview **viewdraw.ini** Configuration File
- MAX+PLUS II/Viewlogic Powerview Project File Structure
- Altera-Provided Logic & Symbol Libraries
- The **vdpath** & **mega_lpm** Libraries

Functional Simulation

- Performing a Functional Simulation with ViewSim Software
  - Analyzing VHDL Files with the Vantage VHDL Analyzer Software

Timing Simulation

- Project Simulation Flow
- Initializing Registers in VHDL & Verilog Output Files for Power-Up before Simulation
- Performing a Timing Simulation with ViewSim Software
  - Analyzing VHDL Files with the Vantage VHDL Analyzer Software
  - Using ViewDraw & ViewGen Software to Prepare for Multi-Device Board-Level Simulation with ViewSim Software

Related Links

- Viewlogic Powerview Graphical User Interface & the Altera Toolbox
- Powerview Command-Line Syntax
- Compiling Projects with MAX+PLUS II Software
- Programming Altera® Devices
- MAX+PLUS II Development Software
- Altera Programming Hardware
- Viewlogic web site (http://www.viewlogic.com)

Using Viewlogic SpeedWave VHDL Analyzer & MAX+PLUS II Software

---

The following topics describe how to use the Viewlogic SpeedWave VHDL Analyzer software with MAX+PLUS® II software. Click on one of the following topics for information:

This file is suitable for printing only. It does not contain hypertext links that allow you to jump from topic to topic.

Setting Up the MAX+PLUS II/Viewlogic Powerview Working Environment

- Software Requirements
- MAX+PLUS II/Viewlogic Powerview Interface File Organization
- Viewlogic Powerview **viewdraw.ini** Configuration File
- MAX+PLUS II/Viewlogic Powerview Project File Structure
- Altera-Provided Logic & Symbol Libraries
- The **vdpath** & **mega_lpm** Libraries

VHDL Design Entry

- Design Entry Flow
- Creating VHDL Designs for Use with MAX+PLUS II Software
  - Instantiating the clklock Megafunction in VHDL or Verilog HDL
  - Instantiating RAM & ROM Functions in Viewlogic Powerview Designs
- Entering Resource Assignments
  - Modifying the Assignment & Configuration File with the **setacf** Utility

VHDL Analysis

- Analyzing VHDL Files with the SpeedWave VHDL Analyzer Software

Related Topics:

- Go to the following MAX+PLUS II ACCESS℠ Key topics for related information:
  - Powerview Command-Line Syntax
  - Performing a Functional Simulation with ViewSim Software
  - Initializing Registers in VHDL & Verilog Output Files for Power-Up before Simulation
  - Performing a Timing Simulation with ViewSim Software
  - Compiling Projects with MAX+PLUS II Software
  - Programming Altera® Devices
- Go to the following topics, which are available on the web, for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware

- Viewlogic web site (http://www.viewlogic.com)

Setting Up the MAX+PLUS II/Viewlogic Powerview Working Environment

To use the MAX+PLUS® II software with Viewlogic's Powerview software, you must install the MAX+PLUS II software, familiarize yourself with the Altera® Toolbox in the Powerview Cockpit, and then establish an environment that facilitates entering and processing designs. The MAX+PLUS II /Viewlogic Powerview interface is installed automatically when you install the MAX+PLUS II software on your workstation.

To set up your working environment for the MAX+PLUS II/Viewlogic Powerview interface, follow these steps:

1. Ensure that you have correctly installed the MAX+PLUS II and Viewlogic software versions described in MAX+PLUS II/Viewlogic Powerview Software Requirements.

2. Add the following environment variable to your **.cshrc** file to specify **/usr/maxplus2** as the MAX+PLUS II system directory:

   ```
   setenv ALT_HOME /usr/maxplus2 ↵
   ```

3. Add the **$ALT_HOME/viewlogic/standard**, **$ALT_HOME/bin**, and **$ALT_HOME/viewlogic/bin** directories to the `PATH` environment variable in your **.cshrc** file.

4. Add the **$ALT_HOME/viewlogic/standard** directory to the `WDIR` environment variable in your **.cshrc** file using the following syntax:

   ```
   setenv WDIR $ALT_HOME/viewlogic/standard:/<Powerview system directory>/standard ↵
   ```

   ☞ Make sure the **$ALT_HOME/viewlogic/standard** directory is the first directory in your `WDIR` path.

5. Source your **.cshrc** file by typing `source .cshrc ↵` at the UNIX prompt.

6. Create the Viewlogic Powerview **viewdraw.ini** configuration file.

7. Copy the **/usr/maxplus2/maxplus2.ini** file to your **$HOME** directory:

   ```
   cp /usr/maxplus2/maxplus2.ini $HOME ↵

   chmod u+w $HOME/maxplus2.ini ↵
   ```

   The **maxplus2.ini** file contains both Altera- and user-specified initialization parameters that control the MAX+PLUS II software, such as MAX+PLUS II symbol and logic function library paths and the current project name. The MAX+PLUS II installation procedure creates and copies the **maxplus2.ini** file to the **/usr/maxplus2** directory.

   ☞ Normally, you do not have to edit your local copy of **maxplus2.ini**, because the MAX+PLUS II software updates the file automatically whenever you change any parameters or settings. However, if you move the **max2lib** and **max2inc** library subdirectories, you must update the file. Go to "Creating & Using a Local Copy of the **maxplus2.ini** File" in MAX+PLUS II Help for more information.

8. If you plan to instantiate Library of Parameterized Modules (LPM) functions in ViewDraw schematics, you must create a new file with the name **vdraw.vs**. The **vdraw.vs** file must include the following line:

   ```
   load ("vdpath")
   ```

You must also make sure that you specify the **vdraw.vs** file in your `WDIR` path.

9. Set up a directory structure that facilitates working with the MAX+PLUS II/Viewlogic Powerview interface. Refer to MAX+PLUS II/Viewlogic Powerview Project File Structure.

Related Topics:

- Go to *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual for more information on installation and details on the directories that are created during MAX+PLUS II installation. Go to MAX+PLUS II/Viewlogic Powerview Interface File Organization for information about the MAX+PLUS II/Viewlogic Powerview directories that are created during MAX+PLUS II installation.
- Go to the following topics, which are available on the web, for additional information:
  - ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
  - This manual is also available in 4 parts:
    - Preface & Section 1: MAX+PLUS II Installation
    - Section 2: MAX+PLUS II - A Perspective
    - Section 3: MAX+PLUS II Tutorial
    - Appendices, Glossary & Index

MAX+PLUS II/Viewlogic Powerview Software Requirements

The following applications and utilities are used to generate, process, synthesize, and verify a project with MAX+PLUS® II and Viewlogic Powerview software.

	Viewlogic	**Altera**
ViewDraw	ViewGen	MAX+PLUS II version 9.4
VHDL Analyzer	ViewPath (optional)	
Vantage VHDL Analyzer	ViewTrace	
VHDL -> sym	ViewData Path	
edifneto	MOTIVE version 5.1.6 *Note (1)*	
edifneti	MOTIVE for Powerview version 3.2.1 (optional) *Note (1)*	
EEDIF (optional)	SDF2MTV (optional)	
MMP (optional)	Fusion/VCS	
vsm		

Note:

(1) MOTIVE for Powerview, a wrapper application for MOTIVE, provides a graphical user interface for the utilities (i.e., EEDIF, SDF2MTV, and MMP) used during a static timing verification with MOTIVE. MOTIVE alone does not accept EDIF files through the Setup Advisor.

☞ The MAX+PLUS II **read.me** file provides up-to-date information on which versions of Viewlogic Powerview applications the current version of the MAX+PLUS II software supports. It also provides information on installation and operating requirements. You should read the **read.me** file on the CD-ROM before installing the MAX+PLUS II software. After installation, you can open the **read.me** file from the MAX+PLUS II Help menu.

MAX+PLUS II/Viewlogic Powerview Interface File Organization

Table 1 shows the MAX+PLUS® II/Viewlogic Powerview interface subdirectories that are created in the MAX+PLUS II system directory (by default, the **/usr/maxplus2** directory) during MAX+PLUS II installation.

☞ For information on the other directories that are created during MAX+PLUS II installation, see "MAX+PLUS II File Organization" in *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual.

Table 1. MAX+PLUS II Directory Organization

Directory	Description
./lmf	Contains the Altera-provided Library Mapping File, **vwlogic.lmf**, that maps Viewlogic logic functions to equivalent MAX+PLUS II logic functions.
./viewlogic	Contains the **alt_edif.cfg** EDIF configuration file that is used with the **edifneti** utility. Also contains the library and sample subdirectories.
./viewlogic/examples	Contains the sample Viewlogic designs.
./viewlogic/library/max2sim	Contains the MAX+PLUS II simulation model library (**max2_sim**) for use in ViewSim software.
./viewlogic/library/alt_max2	Contains MAX+PLUS II primitives (`EXP`, `GLOBAL`, `LCELL`, `SOFT`, `CARRY`, `CASCADE`, `DFFE`, `DFFE6K`, and `OPNDRN`), macrofunctions (`a_8fadd`, `a_8mcomp`, `a_8count`, `a_81mux`), and megafunctions (`clklock`) for use in ViewDraw schematics. These logic functions support specific architectural features of Altera® devices. The **alt_max2** library also contains modified versions of the ViewDraw primitives that use tri-state buffers, because these primitives require special handling in the MAX+PLUS II /Viewlogic Powerview interface.
./viewlogic/library/synlib	Contains the Altera-provided synthesis library **altera**, which includes MAX+PLUS II primitives, the **altera.sml** file, a **sym** directory, and a **wir** directory for use with ViewSynthesis software.
./viewlogic/library/alt_mf	Contains the VHDL models for the MAX+PLUS II primitives (`EXP`, `GLOBAL`, `LCELL`, `SOFT`, `CARRY`, `CASCADE`, `DFFE`, and `OPNDRN`), macrofunctions (`clklock`) for use with ViewSynthesis software, the Vantage VHDL Analyzer software, and the VHDL source files. These logic functions are used to maintain portability to other architectures.
./viewlogic/library/alt_time	Contains MOTIVE timing models for MAX+PLUS II logic functions (**motive.lib**), including the `clklock` megafunction, and MAX+PLUS II driver models (**motive.drv**).
./viewlogic/library/alt_vtl	Contains the VHDL source files for the VITAL 3.0-compliant library. This library is available for ViewSim software.
./viewlogic/bin	Contains all MAX+PLUS II, Viewlogic, and interface-related scripts.
./viewlogic/standard	Contains all standard **.ini** files and standard tools.

Related Topics:

- Go to the following topics, which are available on the web, for additional information:
  - ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
  - This manual is also available in 4 parts:
    - Preface & Section 1: MAX+PLUS II Installation
    - Section 2: MAX+PLUS II - A Perspective
    - Section 3: MAX+PLUS II Tutorial
    - Appendices, Glossary & Index

Viewlogic Powerview viewdraw.ini Configuration File

Each Powerview project is configured with the **viewdraw.ini** file that resides in the project directory. The `DIR` statements at the end of **viewdraw.ini** are paths to library directories that are used by the various Powerview applications. Figure 1 shows a sample of the `DIR` statements that are required to use the libraries.

Figure 1. Excerpt from viewdraw.ini

```
DIR [pw] .
DIR [r]  /usr/maxplus2/vwlogic/library/alt_max2 (alt_max2)
DIR [r]  /usr/maxplus2/vwlogic/library/max2sim (max2_sim)
DIR [r]  /usr/maxplus2/vwlogic/library/synlib (altera)
DIR [r]  /usr/maxplus2/vwlogic/library/alt_mf (alt_mf)
DIR [r]  /usr/maxplus2/vwlogic/library/alt_vtl (alt_vtl)
DIR [rm] /<Powerview system directory>/lib/builtin (builtin)
DIR [rm] /<Powerview system directory>/simmods/vl/dip/74ls (vl74ls)
DIR [rm] /<Powerview system directory>/symsets/vl/dip/74ls (vl74ls)
DIR [r]  /<Powerview system directory>/lib/vdpath (vdpath)
```

☞ When you add the libraries to the **/usr/maxplus2/vwlogic/standard/viewdraw.ini** file, they are automatically set when you create a new project. Powerview tools search these libraries sequentially, so it is important to add them in the order in which they are listed in Figure 1.

Table 1 shows the libraries that must be specified in the `DIR` statements in the **viewdraw.ini** file.

Table 1. Powerview Application Libraries

Library	Library Alias	Source	Topics
alt_max2	`alt_max2`	Altera	Graphical elements for ViewDraw
max2sim	`max2_sim`	Altera	Models for project simulation
synlib	`altera`	Altera	VHDL synthesis library for the MAX+PLUS ® II software
alt_mf	`alt_mf`	Altera	VHDL models of MAX+PLUS II logic functions
alt_vtl	`alt_vtl`	Altera	VITAL-compliant primitives
builtin	`builtin`	Altera	Basic primitives such as `INPUT` pins, `OUTPUT` pins, `AND` gates, `OR` gates, etc.
74ls	`vl74ls`	Viewlogic	74-series macrofunctions
vdpath	`vdpath`	Viewlogic	Standard library of parameterized modules (LPM) functions

☞ The Altera-provided libraries must be listed before the Viewlogic-provided libraries in the **viewdraw.ini** file to ensure that the correct versions of the megafunctions, macrofunctions, and primitives are used.

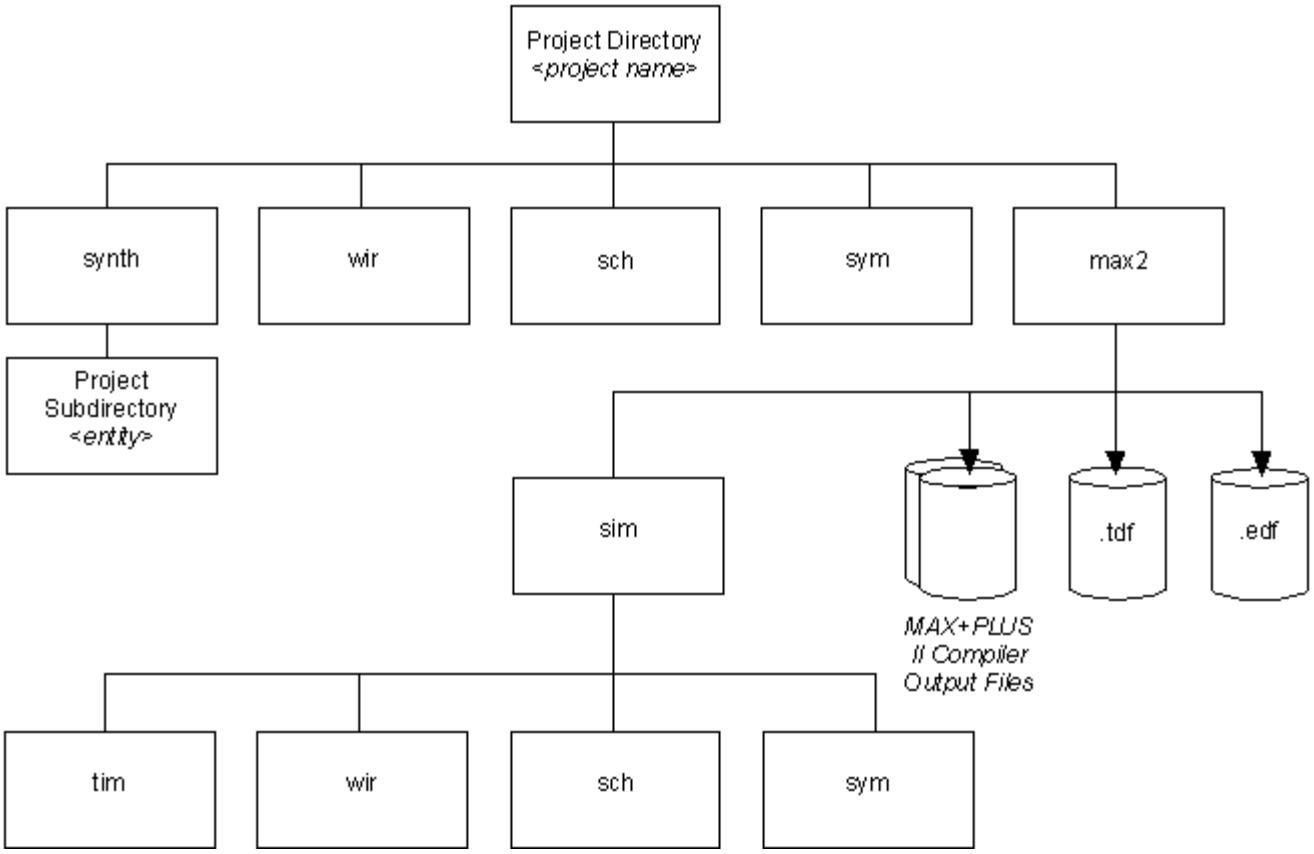
Related Topics:

- Go to Altera-Provided Logic & Symbol Libraries for more information on Altera-supplied libraries. Refer to the Powerview documentation for more information on setting up the **viewdraw.ini** file.

MAX+PLUS II/Viewlogic Powerview Project File Structure

In the MAX+PLUS® II software, a project name is the name of a top-level design file, without the filename extension. This design file can be an EDIF, Verilog HDL, or VHDL netlist file; an Altera® Hardware Description Language (AHDL) TDF; or any other MAX+PLUS II- supported design file. The EDIF netlist file must be created by Powerview and imported into the MAX+PLUS II software as an EDIF Input File (**.edf**). Figure 1 shows an example of MAX+PLUS II project directory structure that includes Powerview-generated files.

Figure 1. Sample MAX+PLUS II Project Organization



The MAX+PLUS II software stores the connectivity data on the links between design files in a hierarchical project in a Hierarchy Interconnect File (**.hif**), but refers to the entire project only by its project name. The MAX+PLUS II Compiler uses the HIF to build a single, fully flattened project database that integrates all the design files in a project hierarchy.

Unlike Powerview, the MAX+PLUS II software does not automatically create a project directory when you create a project. A single directory can contain several MAX+PLUS II design files, and you can specify any one of the designs in the directory as a project in the MAX+PLUS II software.

Viewlogic Powerview Local Work Area Structure

When you create a project with the Powerview Cockpit's **Create** command (Project menu), the project directory is created. You should generate design files and functional simulation files under this directory. A **max2** subdirectory is automatically created under your current project directory when you generate an EDIF file from your schematic or VHDL file. The ***<project name>*.edf** file is stored in the **max2** subdirectory. All MAX+PLUS® II Compiler output files are created in the **/*<project name>*/max2** subdirectory.

☞ ViewDraw files are identified by their directories and not by their extensions, so it is easy to overwrite files unintentionally. To avoid overwriting files, Altera recommends that you create a new project directory, ***<project name>*/max2/sim**, where you can generate all the files needed for simulation.

ViewDraw Project File Structure

Each ViewDraw project directory contains three subdirectories: **wir**, **sch**, and **sym**. See Table 1.

Table 1. ViewDraw Subdirectories

Directory	Topics

./wir	Wirelist files that contain connectivity information for a particular logic block
./sch	Schematics that contain logic
./sym	Symbol files that are the ViewDraw graphical representation of the logic blocks

Each file type uses the filename extension **.1**. Different file types are distinguished only by their directory: **/lib/wir/**<*project name*>**.1** is a wirelist file; **/lib/sch/**<*project name*>**.1** is the corresponding schematic file; and **/lib/sym/**<*project name*>**.1** is the corresponding symbol.

VHDL Project File Structure

Each VHDL project directory contains three subdirectories. See Table 2.

Table 2. VHDL Subdirectories

Directory	Topics
./synth	All synthesis-related files and directories
./synth/<*entity*>	Four types of files: <*entity*>**.pdf**, <*entity*>**.opt**, <*entity*>**.sta**, and <*entity*>**.gnl**
./wir	Wirelist for synthesized VHDL modules

☞ For each VHDL entity in the design, there is a corresponding **./synth/<*entity*>** directory.

Altera-Provided Logic & Symbol Libraries

The MAX+PLUS® II/Viewlogic Powerview environment provides libraries for compiling, synthesizing, and simulating designs.

☞ You can create your own libraries of custom symbols and logic functions for use in ViewDraw schematics and VHDL design files. You can use custom symbols (and functions) to incorporate an EDIF Input File, TDF, or any other MAX+PLUS II-supported design file into a project. The MAX+PLUS II software uses the **vwlogic.lmf** Library Mapping File to map ViewDraw symbols to equivalent MAX+PLUS II megafunctions, macrofunctions, or primitives. To use custom symbols and functions, you can create a custom LMF that maps your custom functions to equivalent EDIF Input Files, TDFs, or other MAX+PLUS II-supported design files. Go to "Library Mapping File" and "Viewlogic Library Mapping File" in MAX+PLUS II Help for more information.

Logic symbols used in ViewDraw software are available from the MAX+PLUS II **alt_max2** library, the ViewDraw **builtin** and **74ls** libraries, and the ViewDatapath **vdpath** library. VHDL models of MAX+PLUS II logic functions are available from the Altera-provided **alt_mf** library.

The alt_max2 Library

The **alt_max2** library provides MAX+PLUS II-specific logic functions that can be used to take advantage of special architectural features in each Altera® device family. See Table 1. Symbols and functional simulation models are available for all of these elements.

The alt_mf Library

The Altera-provided **alt_mf** library, which supports the Viewlogic Vantage VHDL Analyzer software, contains VHDL simulation models for all logic functions listed in the following table. The library is configured so that these functions pass untouched through the EDIF netlist file to the MAX+PLUS II Compiler, providing you with optimal control over design processing. Altera also provides models for all of the logic functions that you can synthesize and simulate. These models allow you to perform functional VHDL simulation while maintaining an architecture-independent VHDL description.

Table 1. Architecture Control Logic Functions

Name *Note (1), Note (2)*	Description	Name	Description	Name	Description
8fadd	8-bit full adder macrofunction	LCELL	Logic cell buffer primitive	EXP	MAX® 5000, MAX 7000, and MAX 9000 Expander buffer primitive
8mcomp	8-bit magnitude comparator macrofunction	GLOBAL	Global input buffer primitive	SOFT	Soft buffer primitive
8count	8-bit up/down counter macrofunction	CASCADE	FLEX® 6000, FLEX 8000, and FLEX 10K cascade buffer primitive	OPNDRN	Open-drain buffer primitive
81mux	8-to-1 multiplexer macrofunction	CARRY	FLEX 6000, FLEX 8000, and FLEX 10K cascade buffer primitive	DFFE *Note (2)*	D-type flipflop with Clock Enable primitive
clklock	Phase-locked loop megafunction				

Notes:

1. Logic function names that begin with a number must be prefixed with "a_" in VHDL designs. For example, 8fadd must be specified as a_8fadd.
2. For designs that are targeted to FLEX 6000 devices, you should use the DFFE primitive only if the design contains either a Clear or Preset signal, but not both. If your design contains both a Clear and a Preset signal, you must use the DFFE6K primitive.

Related Topics:

- Choose **Old-Style Macrofunctions**, **Primitives**, or **Megafunctions/LPM** from the MAX+PLUS II Help menu for detailed information on these functions.
- Go to the following topics, which are available on the web, for additional information:
  - FLEX Devices
  - MAX Devices
  - Classic Device Family

The vdpath & mega_lpm Libraries

The library of parameterized modules (LPM) 2.1.0 standard defines a set of parameterized functions and their corresponding representations in an EDIF netlist file. These logic functions allow you to create and functionally simulate an LPM-based design without targeting a specific device family. After the design is completed, you can target the design to any device family.

When the MAX+PLUS® II software processes projects that include Viewlogic-provided **vdpath** LPM functions, it uses functions from the Altera-provided **mega_lpm** library. This library includes all standard LPM functions except the truth table, finite state machine, and pad functions. Altera does not directly support the lpm_ram_dq, lpm_ram_io, and lpm_rom functions. Refer to Instantiating RAM & ROM Functions in Viewlogic Powerview Designs for instructions on instantiating RAM and ROM functions.

☞ Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu for more information about LPM functions.

MAX+PLUS II/Viewlogic Powerview Design Entry Flow

Figure 1 shows the design entry flow for the MAX+PLUS® II/Viewlogic Powerview interface.

Figure 1. MAX+PLUS II/Viewlogic Powerview Design Entry Flow

Altera-provided items are shown in blue.

Creating VHDL Designs for Use with MAX+PLUS II Software

You can create VHDL design files with the MAX+PLUS® II Text Editor or another standard text editor and save them in the appropriate directory for your project. The MAX+PLUS II Text Editor offers the following advantages:

- VHDL templates are available with the **VHDL Templates** command (Templates menu). These templates are also available in the ASCII **vhdl.tmp** file, which is located in the **/usr/maxplus2** directory.
- If you use the MAX+PLUS II Text Editor to create your VHDL design, you can use the **Syntax Coloring** command (Options menu). The Syntax Coloring feature displays keywords and other elements in text files in different colors to distinguish them from other forms of syntax.

To create a VHDL design that can be synthesized and optimized with ViewSynthesis software, follow these steps:

1. You can instantiate the following Altera-provided logic functions in your VHDL design:
    - The **alt_mf** library contains the Altera® VHDL logic function library, which includes MAX+PLUS II-specific primitives and the `a_8count`, `a_8mcomp`, `a_8fadd`, and `a_81mux` macrofunctions. If you wish to instantiate **alt_mf** logic functions in your VHDL design, you must first analyze all functions in the **alt_mf/src** directory. See Analyzing VHDL Files with the Vantage VHDL Analyzer Software for details.
    - The `clklock` megafunction, which enables the phase-locked loop, or ClockLock , circuitry available on selected Altera FLEX® 10K devices. Go to Instantiating the clklock Megafunction in VHDL or Verilog HDL for information.
    - MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.
2. (Optional) To enter resource assignments in your VHDL design, go to Entering Resource Assignments. You can also enter resource assignments from within the MAX+PLUS II software.

Once you have created a VHDL design, you can analyze it, synthesize it, and generate an EDIF netlist file that can be imported into the MAX+PLUS II software with either of the following methods:

- You can analyze, functionally simulate, and synthesize the VHDL design, then generate an EDIF netlist file

by following the steps in these topics:

  - Analyzing VHDL Files with the Vantage VHDL Analyzer Software
  - Performing a Functional Simulation with ViewSim Software
  - Synthesizing & Optimizing VHDL Designs with ViewSynthesis Software
  - Converting ViewDraw Schematics or VHDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the **edifneto** Utility

- You can use the **VHDL <-> max2** utility in the Max2 Express Drawer to automatically analyze and synthesize the VHDL design, compile it with the MAX+PLUS II Compiler, generate an EDIF Output File (**.edo**), and create a **.vsm** file for simulation. See Using the Max2 Express Drawer's **VHDL <-> max2** Utility in these MAX+PLUS II ACCESS^{SM} Key topics for details.

Installing the Altera-provided MAX+PLUS II/Viewlogic Powerview interface on your computer automatically creates the following sample VHDL files:

- **/usr/maxplus2/examples/viewlogic/example5/count4.vhd**
- **/usr/maxplus2/examples/viewlogic/example5/count8.vhd**

Related Topics:

- Go to FLEX 10K Device Family, which is available on the web, for additional information.

Instantiating the clklock Megafunction in VHDL or Verilog HDL

MAX+PLUS® II interfaces to other EDA tools support the `clklock` phase-locked loop megafunction, which can be used with some FLEX® 10K devices, with the **genckllk** utility, which is available in the MAX+PLUS II system directory. Type `genclklk -h` ↵ at the DOS or UNIX prompt to display information on how to use this utility. The **genckllk** utility generates VHDL or Verilog HDL functional simulation models and a VHDL Component Declaration template file (**.cmp**).

The **genckllk** utility allows parameters for the `clklock` function to be passed from the VHDL or Verilog HDL file to EDIF netlist format. The **genckllk** utility embeds the parameter values in the `clklock` function name; therefore, the values do not need to be declared explicitly.

To instantiate the `clklock` megafunction in VHDL or Verilog HDL, go through the following steps:

1. Type the following command at the DOS or UNIX prompt to generate the `clklock_`*x*`_`*y* function, where *x* is the ClockBoost value and *y* is the input frequency in MHz:

   ✔ Type `genclklk` <*ClockBoost*> <*input frequency*> `-vhdl` ↵ for VHDL designs.

   or:

   ✔ Type `genclklk` <*ClockBoost*> <*input frequency*> `-verilog` ↵ for Verilog HDL designs.

   Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu for more information on the `clklock` megafunction.

2. Create a design file that instantiates the `clklock_`*x*`_`*y***.vhd** or `clklock_`*x*`_`*y***.v** file. The **genckllk** utility automatically generates a VHDL Component Declaration template in the **clklock_*x*_*y*.cmp** file that you can incorporate into a VHDL design file.

In MAX+PLUS II version 8.3 and lower, running `genclklk` on a PC always creates files named as **clklock.vhd**,

clklock.cmp, and **clklock.v**, regardless of the ClockBoost and input frequency values you specify.

Figures 1 and 2 show a `clklock` function with <*ClockBoost*> = 2 and <*input frequency*> = 40 MHz instantiated in VHDL and Verilog HDL design files, respectively.

Figure 1. VHDL Design File with clklock Instantiation (count8.vhd)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY altera;
USE altera.maxplus2.all;     --  Include Altera Component Declarations



ENTITY count8 IS
   PORT (a    : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
         ldn  : IN STD_LOGIC;
         gn   : IN STD_LOGIC;



dnup : IN STD_LOGIC;
         setn : IN STD_LOGIC;
         clrn : IN STD_LOGIC;
         clk  : IN STD_LOGIC;



co   : OUT STD_LOGIC;
         q    : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END count8;



ARCHITECTURE structure OF count8 IS
   signal clk2x : STD_LOGIC;



COMPONENT clklock_2_40
   PORT (
      INCLK : IN STD_LOGIC;
      OUTCLK : OUT STD_LOGIC
   );
END COMPONENT;



BEGIN
   u1: clklock_2_40
   PORT MAP (inclk=>clk, outclk=>clk2x);



u2: a_8count
   PORT MAP (a=>a(0), b=>a(1), c=>a(2), d=>a(3),
             e=>a(4), f=>a(5), g=>a(6), h=>a(7),
             clk=>clk2x,
             ldn=>ldn,
             gn=>gn,



dnup=>dnup,
             setn=>setn,
             clrn=>clrn,



qa=>q(0), qb=>q(1), qc=>q(2), qd=>q(3),
             qe=>q(4), qf=>q(5), qg=>q(6), qh=>q(7),
             cout=>co);
```

```
 END structure;
```

Figure 2. Verilog HDL Design File with clklock Instantiation (count8.v)

```
`timescale 1ns / 10ps
module count8 (a, ldn, gn, dnup, setn, clrn, clk, co, q);
output          co;
output[7:0]     q;



input[7:0]      a;
input           ldn, gn,dnup, setn, clrn, clk;
wire            clk2x;



clklock_2_40 u1 (.inclk(clk), .outclk(clk2x) );
A_8COUNT u2 (.A(a[0]), .B(a[1]), .C(a[2]), .D(a[3]), .E(a[4]), .F(a[5]),



.G(a[6]), .H(a[7]), .LDN(ldn), .GN(gn), .DNUP(dnup),
             .SETN(setn), .CLRN(clrn), .CLK(clk2x), .QA(q[0]), .QB(q[1]),
             .QC(q[2]), .QD(q[3]), .QE(q[4]), .QF(q[5]), .QG(q[6]),
             .QH(q[7]), .COUT(co) );



endmodule
```

Related Topics:

- Go to FLEX 10K Device Family, which is available on the web, for additional information.

Instantiating RAM & ROM Functions in Viewlogic Powerview Designs

The MAX+PLUS®II /Viewlogic Powerview interface offers full support for the memory capabilities of the FLEX® 10K device family, including synchronous and asynchronous RAM and ROM, cycle-shared dual-port RAM, dual-port RAM, single-Clock FIFO, and dual-Clock FIFO functions. You can use the Altera-provided **genmem** utility to generate functional simulation models and timing models for these functions. Type `genmem` ↵ at the UNIX prompt to display information on how to use this utility, as well as a list of the functions you can generate. RAM and ROM can be instantiated in both ViewDraw schematics and VHDL designs.

☞ Refer to Viewlogic documentation for information on simulating projects that contain RAM functions. The procedure for reading an EDIF Output File and preparing it for simulation with ViewSim requires additional steps when the project contains RAM functions.

When you instantiate a RAM or ROM function, follow these general guidelines:

- For ROM functions, you must specify an initial memory content file in the Intel hexadecimal format (**.hex**) or the Altera® Memory Initialization File (**.mif**) format. The filename must be the same as the instance name; e.g., the instance name must be unique throughout the whole project, and must contain only valid name characters. The initialization file must reside in the directory containing the project's design files.
- For RAM functions, specifying a memory initialization file is optional.
- For VHDL designs, specify the name of the initial memory content file in the Generic Map Clause of the

instance, with the specified type `LPM_FILE`. If you do not use an initial memory content file (e.g., for a RAM function), you should not declare or use the Generic Clause.

- Do not synthesize the **genmem**-generated VHDL file: it is intended for simulation only.

☞ The MIF format is supported only for specifying initial memory content when compiling designs within the MAX+PLUS II software. You cannot use a MIF to perform simulation with Viewlogic tools prior to MAX+PLUS II compilation.

To instantiate RAM or ROM in a ViewDraw schematic, follow these steps:

1. Use the **genmem** utility to generate a memory model by typing the following command at the UNIX prompt:

   `genmem <memory type> <memory size> -vwlogic` ↵

   For example: `genmem asynrom 256x15 -vwlogic` ↵

2. Start the VHDL-to-symbol utility, **vhdl2sym**, by double-clicking Button 1 on the **max2_vhdl2sym** icon in the Altera® Toolbox Design Tools Drawer.

3. Specify the following options in the **vhdl2sym** dialog box and choose **OK** to create a symbol. For example, to create the symbol for a 256x15 asynchronous ROM, enter the following settings:

Option:	**Setting:**
VHDL Source Filename	*asyn_rom_256x15.vhd*
Add LEVEL attribute	*On*

4. Choose **Comp** (Add menu), type <*design name*> in the *Enter Name* box, and choose **OK**.

To instantiate a RAM or ROM function in VHDL, follow these steps:

1. Repeat step 1 above.

2. Create a VHDL design that incorporates the text from the **genmem**-generated Component Declaration, <*memory name*>**.cmp**, and instantiate the <*memory name*> function.

Figure 1 shows a VHDL design that instantiates asyn_rom_256x15.vhd, a 256 x 15 ROM function.

Figure 1. VHDL Design File with ROM Instantiation (tstrom.vhd)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;



ENTITY tstrom IS
        PORT (
          addr    : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
          memenab : IN STD_LOGIC;
          q       : OUT STD_LOGIC_VECTOR (14 DOWNTO 0));
END tstrom;



ARCHITECTURE behavior OF tstrom IS

COMPONENT asyn_rom_256x15
     GENERIC (LPM_FILE : string);
```

```
PORT (Address : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
          MemEnab : IN STD_LOGIC;
          Q       : OUT STD_LOGIC_VECTOR(14 DOWNTO 0)
    );
END COMPONENT;



BEGIN

   u1: asyn_rom_256x15
        GENERIC MAP (LPM_FILE => "u1.hex")
   PORT MAP (Address => addr, MemEnab => memenab, Q =>q);
END behavior;
```

Related Topics:

- Go to FLEX 10K Device Family, which is available on the web, for additional information.

Entering Resource Assignments

The MAX+PLUS® II software allows you to enter a variety of resource and device assignments for your projects. Resource assignments are used to assign logic functions to a particular pin, logic cell, I/O cell, embedded cell, row, column, Logic Array Block (LAB), Embedded Array Block (EAB), chip, clique, local routing, logic option, timing requirement, or connected pin group. In the MAX+PLUS II software, you can enter all types of resource and device assignments with Assign menu commands. You can also enter pin, logic cell, I/O cell, embedded cell, LAB, EAB, row, and column assignments in the MAX+PLUS II Floorplan Editor. The Assign menu commands and the Floorplan Editor all save assignment information in the ASCII Assignment & Configuration File (**.acf**) for the project. In addition, you can edit ACFs manually in any standard text editor.

ViewDraw Schematics

In ViewDraw schematics, you can assign a limited subset of these resource assignments by assigning properties to symbols. These properties are incorporated into the EDIF netlist file(s). The MAX+PLUS II software automatically converts assignment information from the EDIF Input File (**.edf**) into the ACF format. For information on making MAX+PLUS II-compatible resource assignments, go to the following topics:

- Assigning Pins, Logic Cells & Chips
- Assigning Cliques
- Assigning Logic Options
- Modifying the Assignment & Configuration File with the **setacf** Utility

Installing the Altera-provided MAX+PLUS II/Viewlogic interface on your computer automatically creates the following sample ViewDraw schematic file, which includes resource assignments:

- **/usr/maxplus2/examples/viewlogic/example4/fadd2mpp**

Related Topics:

- Go to Viewlogic documentation for information on how to assign properties. Go to "Resource Assignments in EDIF Input Files" and "Assigning Resources in a Third-Party Design Editor" in MAX+PLUS II Help for more information on assignments or properties that can be assigned in ViewDraw.

VHDL Design Files

For VHDL-based designs, you must use the MAX+PLUS II software or the **setacf** utility to enter resource assignments. For information on using the **setacf** utility, go to Modifying the Assignment & Configuration File with the **setacf** Utility.

Related Topics:

- For information on entering assignments in the MAX+PLUS II software with Assign menu commands or in an ACF, go to "resource assignments" or "ACF, format" in MAX+PLUS II Help using **Search for Help on** (Help menu).

Modifying the Assignment & Configuration File with the setacf Utility

Altera provides the **setacf** utility to help you modify a project's Assignment & Configuration File (**.acf**) from the command line, without opening the file with a text editor. Type `setacf -h` ↵ at a UNIX or DOS prompt to get help on this utility.

Analyzing VHDL Files with the SpeedWave VHDL Analyzer Software

You can use the SpeedWave VHDL Analyzer software to analyze VHDL Design Files (**.vhd**) prior to functional (or gate-level) simulation with ViewSim software, or to synthesis and optimization with ViewSynthesis software. You can also use the SpeedWave VHDL Analyzer to analyze a MAX+PLUS® II -generated VHDL Output File (**.vho**) prior to post-compilation timing simulation with ViewSim software. The **max2_VantgMgr** and **max2_VantgAnlz** tools are located in the Altera® Toolbox Design Tools Drawer.

To analyze a VHDL file with the SpeedWave VHDL Analyzer, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II /Viewlogic Powerview Working Environment.
2. If you wish to analyze a VHDL Design File (**.vhd**), create a VHDL file <*design name*>**.vhd** using the MAX+PLUS II Text Editor or another standard text editor and save it in a working directory. Go to Creating VHDL Designs for Use with MAX+PLUS II Software for more information.
3. If you wish to analyze a MAX+PLUS II-generated VHDL Output File (**.vho**), be sure to select *VHDL 1987* for the *VHDL Version* option and *VHDL Output File (.vho)* for the *Write Delay Constructs To* option in the **VHDL Netlist Writer Settings** dialog box (Interfaces menu) when you set up the MAX+PLUS II Compiler to generate a VHDL Output File. See Compiling Projects with MAX+PLUS II Software for more information on generating VHDL Output Files.
4. If your VHDL file contains functions from the **alt_mf** library, follow these steps:
    1. Start the Vantage Manager by double-clicking Button 1 on the **max2_VantgMgr** icon in the Design Tools Drawer.
    2. Use the Vantage VHDL Library Manager to create an **alt_mf.lib** library file with the symbolic name `ALT_MF`.
    3. Make **alt_mf** the working library with the **Set Working** command (Edit menu).
    4. Start the VHDL Analyzer by double-clicking Button 1 on the **max2_VantgAnlz** icon in the Design Tools Drawer.

5. Analyze each VHDL file in the **alt_mf/src** directory into the **alt_mf.lib** working library. Source files are located in the **/usr/maxplus2/vwlogic/library/alt_mf/src** directory that is created by installing the Altera/Viewlogic interface.

5. If it is not already running, start the Vantage VHDL Library Manager, as described in step 4b, to create a Vantage library.

6. Choose the **List system libs** button.

7. Add the **ieee.lib** and **synopsys.lib** system libraries to your project:

    1. Select the **ieee.lib** and **synopsys.lib** libraries from the *Available Libraries* window and choose **Add lib**. Choose the **ieee** library from the **libs_syn** directory, which is located at ***/<Powerview system directory>/*** **standard/van_vss/pgm/libs_syn**. The **ieee** library contains Synopsys package files.

    2. If your project uses functions from the **alt_mf** library, also select the **alt_mf.lib** file from the *Available Libraries* window and choose **Add lib**.

    3. Choose **Create Library** (File menu, type the project directory name in the *Symbolic Name* field, and choose **OK**.

8. Specify the project directory as the working directory by choosing **Set Working** (Edit menu).

9. Choose **Save INI File** (File menu).

10. Choose **Dismiss Window** (Powerview Red-Box menu).

11. Specify the appropriate path and file name in the **Analyzer VHDL Source File** dialog box and choose **OK** to analyze the VHDL file.

12. Once you have analyzed the file, perform one or more of the following tasks, as appropriate:

    - Performing a Functional Simulation with ViewSim Software
    - Synthesizing & Optimizing VHDL Designs with ViewSynthesis Software
    - Performing a Timing Simulation with ViewSim Software

Related Topics:

- Refer to the following sources for related information:
    - The ***Viewlogic ViewSim/VHDL User's Guide*** and ***ViewSim/VHDL Tutorial*** for information on using the Vantage VHDL Analyzer software or Vantage VHDL Library Manager
    - Powerview Command-Line Syntax in these MAX+PLUS II ACCESS℠ Key topics

---

Powerview Command-Line Syntax

Table 1 shows the command-line syntax for using Powerview functions.

Table 1. Powerview Command-Line Syntax

Action	Command
Start VHDL Analyzer software	`vhdl -v` <*project name*>
Start ViewSynthesis software	`vhdldes`
Load Altera® technology library	`vhdldes> technology altera`

Compile a VHDL design	`vhdldes> vhdl` *<project name>*
Synthesize a design	`vhdldes> synthesize`
Generate wirelist file	`vhdldes> wir`
Create a schematic representation	`vhdldes> viewgen`
Generate a synthesis report file	`vhdldes> report`
Start the graphical user interface for ViewSynthesis	`vhdldes> vdesgui`
Start the VHDL-to-symbol utility	`vhdl2sym` *<project name>*
Start **vsm**	`vsm` *<project name>*
Start ViewSim simulator	`viewsim` *<project name>* `-`*<project name>*`.cmd`
Start **edifneto**	`edifneto -f` *<project name>*`-l` (`std` *or* `altera`) *<project name>*`.edf`
Start Vantage VHDL Analyzer software	`analyze -src` *<design file>*
Start MOTIVE for Powerview software	`mfp`

Performing a Functional Simulation with ViewSim Software

You can use Viewlogic ViewSim software to perform a functional simulation of a ViewDraw schematic or a VHDL Design File (**.vhd**) before compiling your project with the MAX+PLUS II Compiler. Follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II /Viewlogic Powerview Working Environment.

2. Create a ViewDraw schematic that follows the guidelines in Creating ViewDraw Schematics for Use with MAX+PLUS II Software. Then go to step 3.

   or:

   Create a VHDL Design File <*design name*>**.vhd** and analyze it, as described in the following MAX+PLUS II ACCESS^{SM} Key topics:

   - Creating VHDL Designs for Use with MAX+PLUS II Software
   - Analyzing VHDL Files with the Vantage VHDL Analyzer Software

   Then go to step 7.

3. With the schematic open in the ViewDraw editor, add `CLR` and `PRE` inputs to any flipflops in your design, or tie the `CLR` and `PRE` ports of the flipflops to VCC. (Use the `PWR` primitive from the **builtin** library.)

4. Choose **Write To** (File menu) and save the schematic as <*design name*>**_funct**.

5. Start the **vsm** utility by double-clicking Button 1 on the **max2_vsmnet** icon in the Altera® Toolbox Design Tools Drawer.

6. Specify the following options in the **vsm** dialog box and choose **OK** to generate the <*design name*>**_funct.vsm** file:

Option:	**Setting:**
Design Name	<*design name*>*_funct*
Level	(blank)

7. Create a simulation command file (**.cmd**) for simulation with ViewSim software. Alternatively, you can enter

commands at the prompt in the ViewSim window. Refer to your Viewlogic documentation for more information on creating ViewSim command files.

8. Start the ViewSim simulation tool by double-clicking Button 1 on the **max2_VSim** icon in the Design Tools Drawer.

9. If you wish to simulate a ViewDraw schematic, specify the following options in the **ViewSim** dialog box, then go to step 11.

Option:	**Setting:**
Design Name	*<design name>_funct*
Command File	*<design name>_funct.cmd*
VHDL Source Window	*OFF*
VHDL Debugging	*OFF*

10. If you wish to simulate a VHDL design, specify the following options in the **ViewSim** dialog box:

Option:	**Setting:**
Design Name	*<design name>*
Command File	*<design name>.cmd*
Graphical Interface	*ON*
VHDL Source Window	*OFF* or *ON*
VHDL Debugging	*OFF* or *ON*

11. Choose **OK** to simulate the design. ViewSim software simulates the design and starts the ViewTrace waveform editor to allow you to observe the simulation results.

12. Use the **edifneto** utility to generate an EDIF Netlist File (**.edf**) that can be imported into the MAX+PLUS II software, as described in Converting ViewDraw Schematics or VHDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the **edifneto** Utility.

Related Topics:

- Go to ViewSim documentation for complete details on simulating a project and using ViewTrace to observe waveform output results.

Performing a Timing Simulation with ViewSim Software

After you have entered a design and compiled it with the MAX+PLUS® II Compiler, you can simulate a MAX+PLUS II-generated EDIF Output File (**.edo**) or VHDL Output File (**.vho**) with ViewSim software. ViewSim software can simulate both the functionality and the timing of your design. It also checks setup time, hold time, and Clock duty cycle timing requirements on registers.

To simulate a design with ViewSim software, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II /Viewlogic Powerview Working Environment.

2. Compile the design with the MAX+PLUS II software and generate an EDIF Output File (**.edo**) or VHDL Output File (**.vho**), as described in Compiling Projects with MAX+PLUS II Software.

3. In the Viewlogic Cockpit window, choose **Create** (Project menu) to open the **Create Project** dialog

box. Type the name of your working directory and choose **OK**. You must create this new directory to avoid overwriting your original files when you generate new files for simulation.

4. Choose **SearchOrder** (Project menu) and add the appropriate directories and aliases to your **viewdraw.ini** file if you have not already done so. Go to Viewlogic Powerview **viewdraw.ini** Configuration File for more information.

   ☞ Refer to Viewlogic documentation for information on simulating projects that contain RAM functions. The procedure for reading an EDIF Output File and preparing it for simulation with ViewSim requires additional steps when the project contains RAM functions.

5. If you used the **SCH <-> max2** or **VHDL <-> max2** utility in the Max2 Express drawer to process your project, skip to step 8.

6. If you wish to simulate a VHDL Output File, follow the steps in Analyzing VHDL Files with the Vantage VHDL Analyzer then skip to step 7d.

7. If you are using the Altera® Toolbox Design Tools Drawer, follow these steps:

   1. To generate a Powerview wirelist from the EDIF Output File, double-click Button 1 on the **max2_edifi** icon in the Design Tools Drawer. The **Netlist In** dialog box is displayed.

   2. In the **Netlist In** dialog box, specify *../<design name>* for the *EDIF Netlist File* option, then choose **OK** to process the EDIF netlist file.

   3. If your project is implemented in multiple devices, repeat steps a and b for each EDIF Output File generated by the MAX+PLUS II Compiler, and ensure that the Altera-provided **alt_edif.cfg** file is specified for the *Attribute Swap Configuration File* option. In a multi-device project, the MAX+PLUS II Compiler generates a separate file for each device, plus a top-level file that is identified by "**_t**" appended to the project name. You must also follow the steps in Using ViewDraw & ViewGen Software to Prepare for Multi-Device Board-Level Simulation with ViewSim Software.

   4. Start the **vsm** utility by double-clicking Button 1 on the **max2_vsmnet** icon in the Design Tools Drawer.

   5. Specify your design name for the *Design Name* option in the **vsm** dialog box and choose **OK** to generate the *<design name>***.vsm** file.

8. Create a simulation command file (**.cmd**) for simulation with ViewSim software. Alternatively, you can enter commands at the prompt in the ViewSim window. Refer to your Viewlogic documentation for more information on creating ViewSim command files.

   ☞ The Altera simulation model library, **max2_sim**, allows you to use the `alt_grst` signal to asynchronously clear all flipflops (`DFFE` primitives).

9. Start the ViewSim simulation tool by double-clicking Button 1 on the **max2_VSim** icon in the Design Tools Drawer or the Max2 Express Drawer.

10. Specify the following options in the **ViewSim** dialog box and choose **OK** to simulate the design:

Option:	**Setting:**
Design Name	*<design name>*
Command File	*<design name>.cmd*
VHDL Source Window	*OFF*

VHDL Debugging *OFF*

ViewSim software simulates the design and starts the ViewTrace waveform editor to allow you to observe the simulation results.

Related Topics:

- Refer to the following sources for related information:
  - ViewSim documentation for complete details on simulating a project and using ViewTrace to observe waveform output results
  - Using ViewDraw & ViewGen Software to Prepare for Multi-Device Board-Level Simulation with ViewSim Software

---

Compiling Projects with MAX+PLUS II Software

The MAX+PLUS® II Compiler can process design files in a variety of formats. This topic describes how to use MAX+PLUS II software to compile projects in which the top-level design file is an EDIF Input File (with the extension **.edf**).

☞ Refer to the following sources for additional information:

- Go to MAX+PLUS II Help for information on compiling VHDL and Verilog HDL, design files directly with the MAX+PLUS II Compiler.
- Go to Running Synopsys Compilers from MAX+PLUS II Software for information on running the Synopsys Design Compiler or FPGA Compiler software on a VHDL or Verilog HDL design from within the MAX+PLUS II Compiler window.

To compile a design (also called a "project") with MAX+PLUS II software, go through the following steps:

1. Create design files that are compatible with the MAX+PLUS II software and convert them into EDIF Input Files with the extension **.edf**. Specific instructions for some tools are described in these MAX+PLUS II ACCESS℠ Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your design entry or synthesis and optimization tool.

2. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, create a mapping for each function in a Library Mapping File (**.lmf**) to map the custom symbol to the corresponding EDIF Input File, AHDL Text Design File (**.tdf**), or other MAX+PLUS II-supported design file. These custom functions are represented in design files as hollow-body symbols or "black box" HDL descriptions.

   ☞ Go to "Library Mapping Files (**.lmf**)" in MAX+PLUS II Help for more information.

3. Open MAX+PLUS II and specify the name of your top-level design file as the project name with the **Project Name** command (File menu). If you open an HDL file in the MAX+PLUS II Text Editor, you can choose the **Project Set Project to Current File** command (File menu) instead.

   ☞ You can also compile a project from a command line. However, the first time you compile a project, the settings you need to specify are easier to specify from within the MAX+PLUS II software. After you have run the graphical user interface for the MAX+PLUS II software at least once, you can more easily use the command-line **setacf** utility to modify options in the Assignment & Configuration File (**.acf**) for the project. Type `setacf -h` ↵ and `maxplus2 -h` ↵

for descriptions of **setacf** and MAX+PLUS II command-line syntax.

4. Choose **Device** (Assign menu) and select the target Altera device family in the *Device Family* drop-down list box. If you wish to implement the design logic in a specific device, select it in the *Devices* box. Otherwise, select *AUTO* to allow the MAX+PLUS II Compiler to choose the best device(s) in the current device family. If your design entry or synthesis and optimization tool required you to specify a target family and/or device, specify the same information in this dialog box. For information on partitioning logic among multiple devices, go to MAX+PLUS II Help. Choose **OK**.

5. Open the Compiler window by choosing the **Compiler** command (MAX+PLUS II menu). Go through the following steps to specify the options necessary to compile the design file(s) in your project:

    1. Ensure that all EDIF netlist files have the extension **.edf** and choose **EDIF Netlist Reader Settings** (Interfaces menu).

    2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor. This name should be the name of the vendor whose tool(s) you used to create the EDIF netlist files. If your vendor name does not appear, select *Custom* instead.

        If you are compiling a design created with Synopsys FPGA Express software, select *Synopsys*, choose the **Customize** button, enter <*project name*>**.lmf** in the *LMF #1* box, choose **OK**, and skip to step 6.

    3. If you selected an existing vendor name in the *Vendor* box and your project contains design files that require custom LMF mappings, choose the **Customize** button to expand the dialog box to show all settings. Turn on the *LMF #2* checkbox and type your custom LMF's filename in the corresponding text box, or select a name from the *Files* box. The selection in the *Vendor* box will change to *Custom* and all settings will be retained until you change them again.

    4. If you selected *Custom* in the *Vendor* box, choose the **Customize** button to expand the dialog box to show all settings. Any previously defined custom settings will be displayed. Under *Signal Names*, type one or more names with up to 20 total name characters in the *VCC* or *GND* box if your EDIF Input File(s) use one or more names other than `VCC` or `GND` for the global high or low signals. Multiple signal names must be separated by either a comma (,) or a space. Under *Library Mapping Files*, turn on the *LMF #1* checkbox and type a filename in the text box following it, or select a name from the *Files* box. If necessary, specify another LMF name in the *LMF #2* box. Go to MAX+PLUS II Help for detailed information on the settings available in the **EDIF Netlist Reader Settings** dialog box.

    5. Choose **OK**.

6. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, you may need to ensure that all files are present in your project directory, i.e., the same directory as the top-level design file. Otherwise, you must specify the directories containing these files as user libraries with the **User Libraries** command (Options menu).

7. Follow all guidelines that apply to your design entry or synthesis and optimization tool:

    - Exemplar Logic Galileo Extreme-Specific Compiler Settings
    - Synopsys DesignWare-Specific Compiler Settings
    - Converting Synopsys FPGA Compiler & Design Compiler Timing Constraints into MAX+PLUS II-Compatible Format with the **syn2acf** Utility
    - Synplicity Synplify-Specific Compiler Settings

8. If you wish to generate EDIF, VHDL, or Verilog HDL output files for post-compilation simulation or timing analysis with another EDA tool, go through the following steps:

1. (Optional) Turn on the **Optimize Timing SNF** command (Processing menu) to reduce the size of the output file(s). Turning on this command can reduce the size of output netlists by up to 30%.

   This command does not create optimized timing SNFs on UNIX workstations. However, a non-optimized timing SNF provides the same functional and timing information as an optimized timing SNF.

2. If you wish to generate EDIF Output Files (**.edo**), go through these steps:

   1. Turn on the **EDIF Netlist Writer** command (Interfaces menu). Then choose the **EDIF Netlist Writer Settings** command (Interfaces menu).

   2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor and choose **OK**. If your vendor name does not appear, select *Custom* instead and specify the settings that are appropriate for your simulation or timing analysis tool. Go to MAX+PLUS II Help for detailed information on the options available in the **EDIF Netlist Writer Settings** dialog box.

   3. To generate an optional Standard Delay Format (SDF) Output File (**.sdo**), choose the **Customize** button to expand the dialog box to show all settings. Select one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**.

   The filenames of the EDIF Output File(s) and optional SDF Output File(s) are the same as the user-defined chip name(s) for the project; if no chip names exist, the Compiler assigns filenames that are based on the project name. For a multi-device project, the Compiler also generates a top-level EDIF Output File that is uniquely identified by "**_t**" appended to the project name. In addition, the Compiler automatically generates a VHDL Memory Model Output File, *<project name>*.**vmo**, when it generates an EDIF Output File that contains memory (RAM or ROM).

3. If you wish to generate VHDL Output Files (**.vho**), turn on the **VHDL Netlist Writer** command (Interfaces menu). Then choose **VHDL Netlist Writer Settings** command (Interfaces menu). Select *VHDL Output File (.vho)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF ver. 2.1 files contain timing delay information that allows you to perform back-annotation simulation in VHDL with VITAL-compliant simulation libraries. The VHDL Output Files generated by the Compiler have the extension **.vho**, but are otherwise named in the same way as the EDIF Output Files described above.

4. If you wish to generate Verilog HDL Output Files (**.vo**), turn on the **Verilog Netlist Writer** command (Interfaces menu). Then choose **Verilog Netlist Writer Settings** command (Interfaces menu). Select *Verilog Output File (.vo)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF Output Files contain timing delay information that allows you to perform back-annotation simulation in Verilog HDL. The Verilog Output Files generated by the Compiler have the extension **.vo**, but are otherwise named in the same way as the EDIF Output Files described above.

9. To run the MAX+PLUS II Compiler, choose the **Project Save & Compile** command (File menu) or choose the **Start** button in the Compiler window.

   See step 3 for information on running MAX+PLUS II software from the command line.

10. Once you have compiled the project with the MAX+PLUS II Compiler, you can use the VHDL, Verilog HDL, or EDIF output file(s), and the optional SDF Output File(s) (**.sdo**) to perform timing analysis or timing simulation with another EDA tool. Specific instructions for some tools are described in these MAX+PLUS II ACCESS Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your EDA tool.

The MAX+PLUS II Compiler also generates a Report File (**.rpt**), a Pin-Out File (**.pin**), and one or more of the following files for device programming or configuration:

- JEDEC Files (**.jed**)
- Programmer Object Files (**.pof**)
- SRAM Object Files (**.sof**)
- Hexadecimal (Intel-format) Files (**.hex**)
- Tabular Text Files (**.ttf**)

Related Topics:

- Refer to the following sources for additional information:
  - Go to *Compiler Procedures* in MAX+PLUS II Help for information on other available Compiler settings.
  - Go to *Programmer Procedures* in MAX+PLUS II Help for instructions on creating other types of programming files and on programming or configuring Altera devices.
  - Go to Back-Annotating MAX+PLUS II Pin Assignments to Design Architect Symbols for information on back-annotating pin assignments in Mentor Graphics Design Architect schematics.
  - Go to Programming Altera Devices for information on the different programming hardware options for Altera device families.
- Go to the following topics, which are available on the web, for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware

Initializing Registers in VHDL & Verilog Output Files for Power-Up before Simulation

Altera provides the **add_dc** script, which is availiable in the MAX+PLUS II system directory, to allow you to process MAX+PLUS II-generated Verilog Output Files (**.vo**) and VHDL Output Files (**.vho**) to prepare these files for simulation with another EDA tool. The add_dc script runs the **add_dclr** utility, which inserts a `device_clear` signal that is used for power-up initialization of all registers or flipflops in the design.

The script adds in a top-level signal named `device_clear` and connects it to the `CLRN` pin in all flipflops that should initialize to 0, and to the `PRN` pin of all flipflops that should initialize to 1. If the `CLRN` or `PRN` pin of a flipflop is already being used (i.e., is already connected to a signal), the script modifies the Verilog Output File or VHDL Output File so that the `AND` of the original signal and the `device_clear` pin feed the `CLRN` or `PRN` pin.

To use the **add_dc** script to process Verilog Output Files and VHDL Output Files before simulation with another EDA tool, follow these steps:

1. Make sure that your design file is located in the current directory, or change to the directory in which the design file is located.
2. Type the following command at the command prompt:

   `Â¥`*<path name of* ***add_dc.bat*** *file>*`Â¥add_dc` *<design name>* *<path name of* ***add_dclr.exe*** *file>* ↵

For example, if the both the **add_dc.bat** and the **add_dclr.exe** files are located in the **d:Â¥maxplus2Â¥exew** directory, and the **d:Â¥maxplus2Â¥exew** directory is specified in the search path, you can type the following command at a command prompt to add a `device_clear` signal to a design named `myfifo` in the file **myfifo.vo**:

```
add_dc myfifo d:Â¥maxplus2Â¥exew ↵
```

1. The **add_dc** script gives a message if the directory contains both a VHDL Output File and a Verilog Output File with the same name (<*design name*>**.vo** and <*design*>**.vho**). You should delete or rename whichever of those files should not have the `device_clear` signal added. The **add_dc** script can modify only one design file at a time.

2. When the **add_dc** script processes the Verilog Output File or VHDL Output File, it creates a backup copy of the original file, with the extension **.ori**.

3. The **add_dc** script works only for Verilog Output Files and VHDL Output Files that are generated by MAX+PLUS II.

After you have used the **add_dc** script and are ready to simulate the resulting Verilog Output File or VHDL Output File with another EDA tool, you should assert the active low `device_clear` pin for a period of time that is long enough for the design to initialize. You can then de-assert the pin, and apply simulation vectors to the design.

Using Viewlogic SpeedWave VHDL Analyzer & MAX+PLUS II Software

The following topics describe how to use the Viewlogic SpeedWave VHDL Analyzer software with MAX+PLUS® II software. Click on one of the following topics for information:

This file is suitable for printing only. It does not contain hypertext links that allow you to jump from topic to topic.

Setting Up the MAX+PLUS II/Viewlogic Powerview Working Environment

- Software Requirements
- MAX+PLUS II/Viewlogic Powerview Interface File Organization
- Viewlogic Powerview **viewdraw.ini** Configuration File
- MAX+PLUS II/Viewlogic Powerview Project File Structure
- Altera-Provided Logic & Symbol Libraries
- The **vdpath** & **mega_lpm** Libraries

VHDL Design Entry

- Design Entry Flow
- Creating VHDL Designs for Use with MAX+PLUS II Software
  - Instantiating the clklock Megafunction in VHDL or Verilog HDL
  - Instantiating RAM & ROM Functions in Viewlogic Powerview Designs
- Entering Resource Assignments
  - Modifying the Assignment & Configuration File with the **setacf** Utility

VHDL Analysis

- Analyzing VHDL Files with the SpeedWave VHDL Analyzer Software

Related Topics:

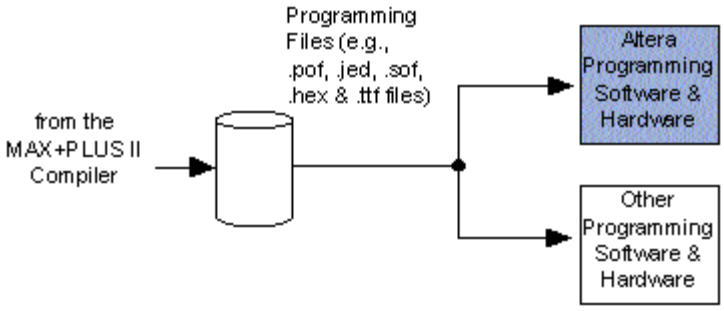
- Go to the following MAX+PLUS II ACCESS℠ Key topics for related information:
  - Powerview Command-Line Syntax
  - Performing a Functional Simulation with ViewSim Software
  - Initializing Registers in VHDL & Verilog Output Files for Power-Up before Simulation
  - Performing a Timing Simulation with ViewSim Software
  - Compiling Projects with MAX+PLUS II Software
  - Programming Altera® Devices
- Go to the following topics, which are available on the web, for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware
  - Viewlogic web site (http://www.viewlogic.com)

Programming Altera Devices

Once you have successfully compiled and simulated a project with the MAX+PLUS® II software, you can program an Altera® device and test it in the target circuit. Figure 1 shows the device programming flow for MAX+PLUS II software.

Figure 1. MAX+PLUS II Device Programming Flow

Altera-provided items are shown in blue.



You can program devices with Altera programming hardware and MAX+PLUS II Programmer software installed on a 486- or Pentium-based PC or a UNIX workstation, or with programming hardware and software available from other manufacturers. Table 1 shows the available Altera programming hardware options on PCs and UNIX workstations.

Table 1. Altera Programming Hardware

Programming Hardware	PCs	UNIX Work-	MAX® 3000A	Classic® & MAX	MAX 7000 &	MAX 7000A, MAX 7000AE, MAX 7000B, MAX	FLEX® 6000, FLEX 6000A, FLEX 8000, FLEX 10K, FLEX	In-System Programming/

Option	stations	Devices	5000 Devices	MAX 7000E Devices	7000S MAX 9000 & MAX 9000A Devices	10KA, FLEX 10KB, & FLEX 10KE Devices	Configuration
Logic Programmer card, PL-MPU Master Programming Unit, and device-specific adapters	✓	✓	✓	✓	✓		
BitBlaster Download Cable	✓	✓	✓		✓	✓	✓
ByteBlasterMV Download Cable	✓		✓		✓	✓	✓
MasterBlaster Download Cable	✓	✓	✓		✓	✓	✓

If you wish to transfer programming files from a UNIX workstation to a PC over a network with File Transfer Protocol (FTP) or other similar transfer programs, be sure to select binary transfer mode.

Programming hardware from other manufacturers varies, but typically consists of a device connected to one of the serial ports on the workstation. Various vendors, such as Data I/O and BP Microsystems, supply hardware and software for programming Altera devices.

Related Topics:

- Go to Compiling Projects with MAX+PLUS II Software for information on creating programming files.
- Go to the following topics, which are available on the web, for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware
  - FLEX Devices
  - MAX Devices
  - Classic Device Family

Creating AHDL Designs for Use with MAX+PLUS II Software

The Altera® Hardware Description Language (AHDL) is a high-level language that supports design entry with Boolean equations, conditional logic, truth tables, arithmetic operators, and parameterized functions, including Library of Parameterized Modules (LPM) functions. AHDL provides a compact and efficient syntax for state machines, decoders, and comparators. The MAX+PLUS® II software can read and compile AHDL Text Design Files (**.tdf**) directly.

AHDL TDFs must be in ASCII file format. If you enter a file with a word processor, you must save it in text-only format. For complete information on AHDL, refer to MAX+PLUS II Help.

You can also use AHDL TDFs to combine EDIF netlist files generated from ViewDraw schematics with other design files to create complex, hierarchical circuits.

AHDL templates are available with the MAX+PLUS II Text Editor's **AHDL Template** command (Templates menu) or in the ASCII **ahdl.tap** file, which is located in the **/usr/maxplus2** directory. Go to "Inserting an AHDL Template" in MAX+PLUS II Help for information on using templates in the Text Editor.

Installing the Altera-provided MAX+PLUS II/Viewlogic interface on your computer automatically creates the following sample AHDL design files:

- **/usr/maxplus2/examples/viewlogic/example2/decode.tdf**
- **/usr/maxplus2/examples/viewlogic/example3/fadd2**

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Creating ViewDraw Schematics for Use with MAX+PLUS II Software

You can create ViewDraw schematics and convert them into EDIF Input Files (**.edf**) that can be processed with the MAX+PLUS® II software.

To create a ViewDraw schematic for use with the MAX+PLUS II software, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II /Viewlogic Powerview Working Environment.
2. Start Powerview by typing `powerview` ↵ at a UNIX prompt.
3. In the Cockpit window, select *Altera* in the *Current ToolBox* drop-down list box, and select the drawer you want to use, i.e., *Design Tools* or *Max2 Express*, in the *Current Drawer* drop-down list box.
4. Choose **Create** (Project menu) from your working directory to create your project directory. Choose **OK**.
5. Choose **SearchOrder** (Project menu) to add the appropriate library directories and aliases to your **viewdraw.ini** file in the appropriate search order. Refer to Viewlogic Powerview **viewdraw.ini** Configuration File for more information on Powerview application libraries.
6. Start ViewDraw by double-clicking Button 1 on the **max2_VDraw** icon in the drawer that you selected in step 3, type the name of the schematic, and choose **OK**. You can also start the ViewDraw software by typing `viewdraw` ↵ at the UNIX prompt.
7. Choose **Comp** (Add menu) to add components to the schematic. You can use functions from the **alt_max2**, **builtin**, and **74ls** libraries. For information on Altera-provided libraries, go to Altera-Provided Logic & Symbol Libraries.

   Instructions for instantiating specific functions are provided in the following MAX+PLUS II ACCESS℠ Key topics:

   - Instantiating LPM Functions in ViewDraw Schematics
   - Instantiating RAM & ROM Functions in Viewlogic Powerview Designs

   ☞ You can instantiate MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP℠). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.
8. If you instantiate a `clklock` megafunction, choose the **Dialog** command (Attr menu), then choose the **All** command (Dialog menu) to specify the values of the `INPUT_FREQUENCY` and `CLOCKBOOST` parameters. For detailed information on the `clklock` megafunction, choose **Megafunctions/LPM** from the MAX+PLUS II Help menu.
9. If you wish to create a hierarchical design that contains symbols representing other design files, such as Altera® Hardware Description Language (AHDL) Text Design Files (**.tdf**), go to Creating Hierarchical Projects in ViewDraw Schematics.

10. Choose **Net** (Add menu) to add nets to the schematic.

11. Choose **Bus** (Add menu) to add buses to the schematic.

12. Choose **Label** (Add menu) to attach labels to nets and buses. When you are naming and labeling buses, make sure you use the format <*bus name*>[<*most significant bit*>:<*least significant bit*>], and that you label both the net and the pin.

13. (Optional) To enter resource assignments in your schematic, select a symbol or a net that feeds an output and use the **Attr** command (Add menu) to add the assignments. For more information, go to Entering Resource Assignments. You can also enter resource assignments from the MAX+PLUS II software.

14. Choose **Write** (File menu) to check and save both the schematic with the name **.sch/<*design name*>.1** and the wirelist with the name **./wir/<*design name*>.1**.

15. (Optional) Perform a functional simulation, as described in Performing a Functional Simulation with ViewSim Software.

16. Once you have created a schematic, you can generate an EDIF netlist file that can be imported into the MAX+PLUS II software with either of the following methods:

    - You can create an EDIF netlist file, as described in Converting ViewDraw Schematics or VHDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the **edifneto** Utility You must use this method if your ViewDraw schematic instantiates Library of Parameterized Modules (LPM) functions.
    - You can use the **SCH <-> max2** utility in the Max2 Express drawer to automatically create an EDIF netlist file, compile it with the MAX+PLUS II Compiler, generate an EDIF Output File (**.edo**), and generate a **.vsm** file for simulation, as described in Using the Max2 Express Drawer's **SCH <-> max2** Utility.

Installing the Altera-provided MAX+PLUS II/Viewlogic interface on your computer automatically creates the following sample ViewDraw schematic files:

- **/usr/maxplus2/examples/viewlogic/example1/fadd**
- **/usr/maxplus2/examples/viewlogic/example3/fadd2**
- **/usr/maxplus2/examples/viewlogic/example4/fadd2mpp**
- **/usr/maxplus2/examples/viewlogic/example7/fifo**

Related Links:

- Go to Powerview Command-Line Syntax in these MAX+PLUS II ACCESS Key topics for related information.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Using Viewlogic ViewDraw & MAX+PLUS II Software

The following topics describe how to use the Viewlogic ViewDraw software with MAX+PLUS® II software. Click on one of the following topics for information:

CLICK HERE to open a printable version of all topics listed on this page.

Setting Up the MAX+PLUS II/Viewlogic Powerview Working Environment

- Software Requirements
- MAX+PLUS II/Viewlogic Powerview Interface File Organization
- Viewlogic Powerview **viewdraw.ini** Configuration File
- MAX+PLUS II/Viewlogic Powerview Project File Structure
- Altera-Provided Logic & Symbol Libraries
- The **vdpath** & **mega_lpm** Libraries

Schematic Design Entry

- Design Entry Flow
- Creating ViewDraw Schematics for Use with MAX+PLUS II Software
  - Instantiating LPM Functions in ViewDraw Schematics
  - Instantiating RAM & ROM Functions in Viewlogic Powerview Designs
- Creating Hierarchical Projects in ViewDraw Schematics
- Entering Resource Assignments
  - Assigning Pins, Logic Cells & Chips
  - Assigning Cliques
  - Assigning Logic Options
  - Modifying the Assignment & Configuration File with the **setacf** Utility
- Performing a Functional Simulation with ViewSim Software
- Converting ViewDraw Schematics or VHDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the **edifneto** Utility

Related Links

- Viewlogic Powerview Graphical User Interface & the Altera Toolbox
- Programming AlteraÂ® Devices

Converting ViewDraw Schematics or VHDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the edifneto Utility

You can use the **edifneto** utility to generate an EDIF netlist file from a ViewDraw schematic or VHDL Design File (**.vhd**). This file can be imported into the MAX+PLUS® II software as an EDIF Input File with the extension **.edf**. To generate an EDIF netlist file, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Viewlogic Powerview Working Environment.

2. Create a ViewDraw schematic and save it in your working directory, as described in Creating ViewDraw Schematics for Use with MAX+PLUS II Software.

   or:

   Create a VHDL Design File, analyze it, and synthesize and optimize it, as described in the following topics:

   - Creating VHDL Designs for Use with MAX+PLUS II Software
   - Analyzing VHDL Files with the Vantage VHDL Analyzer Software
   - Synthesizing & Optimizing VHDL Designs with ViewSynthesis Software

3. Start the **edifneto** utility by double-clicking Button 1 on the **max2_edifo** icon in the Design Tools Drawer or the Max2 Express Drawer in the Altera Toolbox. You can also start the **edifneto** utility by typing `edifneto` ↵ at the UNIX prompt.

4. If you are converting a ViewDraw schematic, specify the <*design name*> for the *Wire File Name* option in the **edifneto** dialog box. If you are not using the Altera® toolbox, do not specify *Altera* for the *Level* option in the **edifneto** dialog box.

5. If you are converting a VHDL Design File, or if your ViewDraw schematic instantiates Library of Parameterized Modules (LPM) functions, specify *Altera* and *VHDL* as the *Level* in the **edifneto** dialog box.

6. Choose **OK** to generate the EDIF netlist file. The **edifneto** utility creates the **max2** subdirectory under your working directory. The **max2** subdirectory contains the EDIF netlist file for your design.

   ☞ When the **edifneto** utility generates an EDIF netlist file from a design that instantiates LPM functions, the EDIF netlist file may contain parameters with incorrect parameter names. To correct this problem, go to the **/usr/maxplus2/viewlogic/bin** directory and type `chlpmpty` <*design name*>`.edf` ↵ at the UNIX prompt to run the Altera-provided **chlpmpty** script, which converts all of the parameters to their correct names.

7. Process the <*design name*>**.edf** with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Related Links:

- Go to the following MAX+PLUS II ACCESS℠ Key topics for related information:

Using the Max2 Express Drawer's **SCH <-> max2** Utility
- Using the Max2 Express Drawer's **VHDL <-> max2** Utility

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Creating Hierarchical Projects in ViewDraw Schematics

You can incorporate any MAX+PLUS® II-supported design file, such as an Altera® Hardware Description Language (AHDL) Text Design File (**.tdf**), into a project hierarchy that consists of both schematic and text files. To incorporate a non-ViewDraw design file into a higher-level schematic design, you must create a hollow-body symbol for it in the ViewDraw software. During compilation, the MAX+PLUS II software recognizes the symbol as an identifier for the design file, and inserts the correct logic and connections. You can incorporate any number of design files into a project hierarchy.

To create a hierarchical project in your ViewDraw schematic, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Viewlogic Powerview Working Environment.
2. Create a ViewDraw schematic and save it in your working directory, as described in Creating ViewDraw Schematics for Use with MAX+PLUS II Software.

   You can instantiate MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.
3. Create a design file that uses all uppercase letters for the function name and all lowercase letters for the file extension, e.g., **DECODE.tdf**. This naming convention is required to prevent conflicts when the file is incorporated into a hierarchical design. When the **edifneto** utility generates an EDIF netlist file from the ViewDraw schematic, it copies the name of the hollow-body symbol in uppercase letters, regardless of the case that appears in the schematic.
4. Double-click Button 1 on the **max2_VDraw** icon in the Altera Toolbox Design Tools Drawer to start ViewDraw.
5. In the **File Open** dialog box, type <*design name*>, i.e., the name of the hollow-body symbol you want to create. Turn on the *Symbol* option and choose **OK**. The Symbol Editor is displayed.
6. Choose **Block Size Z-WxH** (Change menu) and select a symbol size.
7. Choose **Graphics-Box** (Add menu) to draw the symbol body.
8. Choose **Pin** (Add menu) to enter pinstubs.
9. Select a pin and choose **Label** (Add menu) to label the pin names.
10. (Optional) **Choose Graphics-Text** (Add menu) to label the symbol.
11. Choose **Block Type Module** (Change menu). You must choose **Block Type Module** to specify that no Viewlogic schematic is available to represent the functionality of the symbol.
12. Choose **Write** (File menu) to save the symbol.
13. In the top-level ViewDraw schematic, choose **Comp** (Add menu), select the name of the symbol, and choose **OK**.

14. The MAX+PLUS II software uses Library Mapping Files (**.lmf**) to map standard ViewDraw symbols to equivalent MAX+PLUS II megafunctions, macrofunctions, or primitives. To use custom symbols, you can create a custom LMF that maps your custom symbols to the equivalent EDIF Input File, TDF, or other design file.

    ☞ You will also need to specify a Library Mapping File (.lmf) in the EDIF Netlist Reader Settings dialog box before compiling with the MAX+PLUS II Software. Go to Compiling Projects with MAX+PLUS II Software for more information.

15. Continue with the steps necessary to complete your ViewDraw schematic, as described in Creating ViewDraw Schematics for Use with MAX+PLUS II Software.

Related Links:

- Go to Creating AHDL Designs for Use with MAX+PLUS II Software in these MAX+PLUS II ACCESS^{SM} Key topics for related information.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Instantiating LPM Functions in ViewDraw Schematics

You can instantiate library of parameterized modules (LPM) functions from the **vdpath** library in ViewDraw schematics.

☞ Altera does not directly support the `lpm_ram_dq`, `lpm_ram_io`, and `lpm_rom` functions. Go to Instantiating RAM & ROM Functions in Viewlogic Powerview Designs for information on instantiating these functions.

To instantiate an LPM function, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Viewlogic Powerview Working Environment. Make sure that you have created a **vdraw.vs** file, as described in step 8 of that topic.
2. Choose **Cell** (Add menu).
3. Choose an <*LPM function name*> to open the <*LPM function name*> dialog box. Specify a symbol name for *Symbol Prefix* and specify appropriate parameters. Choose **OK**.

   The ViewDraw software generates the specified symbol name symbol according to your specifications. It also generates a corresponding VHDL simulation model, but it is compiled only after you save the schematic. If you want to change the settings for the symbol, select the instance and choose **Cell** (Change menu) to re-open the appropriate dialog box.
4. Continue with the steps necessary to complete your schematic, as described in Creating ViewDraw Schematics for Use with MAX+PLUS II Software.

Related Topics:

- Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu for detailed information on LPM functions.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Using the Max2 Express Drawer's SCH <-> max2 Utility

Once you have created a ViewDraw schematic, you can use the **SCH <-> max2** utility in the Max2 Express drawer to generate an EDIF netlist file from the schematic; process the EDIF Input File (**.edf**) with the MAX+PLUS ® II software to generate an EDIF Output File (**.edo**); and generate a **.vsm** file for simulation. The **SCH <-> max2** utility creates all necessary subdirectories and copies all of the files to the correct locations.

To use the **SCH <-> max2** utility, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II /Viewlogic Powerview Working Environment.
2. Create a ViewDraw schematic that follows the guidelines described in Creating ViewDraw Schematics for Use with MAX+PLUS II Software.
3. Start the **SCH <-> max2** utility by double-clicking Button 1 on the **SCH <-> max2** icon in the Max2 Express Drawer.
4. Specify the *Input Schematic*, *Family*, *Max2 Synthesis Style*, and *Choose project direction* options in the **SCH <-> max2** dialog box and choose **OK** to generate the <*design name*>**.vsm** file for simulation in ViewSim. The **SCH <-> max2** utility generates the <*design name*>**.vsm** file in the **sim** subdirectory of the **max2** directory.
5. If necessary, correct any errors in the ViewDraw schematic and recompile the project.
6. Simulate your project, as described in Performing a Timing Simulation with ViewSim Software.

Related Links:

- Go to Performing Timing Verification for EDIF Output Files (.edo) with MOTIVE & MOTIVE for Powerview Software or Performing Timing Verification of Verilog Output Files (.vo) with MOTIVE Software in these MAX+PLUS II ACCESS℠ Key topics for related information.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Using Viewlogic ViewDraw & MAX+PLUS II Software

The following topics describe how to use the Viewlogic ViewDraw software with MAX+PLUS® II software. Click on one of the following topics for information:

This file is suitable for printing only. It does not contain hypertext links that allow you to jump from topic to topic.

Setting Up the MAX+PLUS II/Viewlogic Powerview Working Environment

- Software Requirements
- MAX+PLUS II/Viewlogic Powerview Interface File Organization
- Viewlogic Powerview **viewdraw.ini** Configuration File
- MAX+PLUS II/Viewlogic Powerview Project File Structure
- Altera-Provided Logic & Symbol Libraries
- The **vdpath** & **mega_lpm** Libraries

Schematic Design Entry

- Design Entry Flow
- Creating ViewDraw Schematics for Use with MAX+PLUS II Software
  - Instantiating LPM Functions in ViewDraw Schematics
  - Instantiating RAM & ROM Functions in Viewlogic Powerview Designs
- Creating Hierarchical Projects in ViewDraw Schematics
- Entering Resource Assignments
  - Assigning Pins, Logic Cells & Chips
  - Assigning Cliques
  - Assigning Logic Options
  - Modifying the Assignment & Configuration File with the **setacf** Utility
- Performing a Functional Simulation with ViewSim Software
- Converting ViewDraw Schematics or VHDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the **edifneto** Utility

Related Topics:

- Go to the following MAX+PLUS II ACCESS℠ Key topics for related information:
  - Viewlogic Powerview Graphical User Interface & the Altera Toolbox
  - Compiling Projects with MAX+PLUS II Software
  - Using ViewDraw & ViewGen Software to Prepare for Multi-Device Board-Level Simulation with ViewSim Software
  - Programming Altera® Devices

Setting Up the MAX+PLUS II/Viewlogic Powerview Working Environment

To use the MAX+PLUS® II software with Viewlogic's Powerview software, you must install the MAX+PLUS II software, familiarize yourself with the Altera® Toolbox in the Powerview Cockpit, and then establish an environment that facilitates entering and processing designs. The MAX+PLUS II /Viewlogic Powerview interface is installed automatically when you install the MAX+PLUS II software on your workstation.

To set up your working environment for the MAX+PLUS II/Viewlogic Powerview interface, follow these steps:

1. Ensure that you have correctly installed the MAX+PLUS II and Viewlogic software versions described in MAX+PLUS II/Viewlogic Powerview Software Requirements.

2. Add the following environment variable to your **.cshrc** file to specify **/usr/maxplus2** as the MAX+PLUS II system directory:

   ```
   setenv ALT_HOME /usr/maxplus2 ↵
   ```

3. Add the **$ALT_HOME/viewlogic/standard**, **$ALT_HOME/bin**, and **$ALT_HOME/viewlogic/bin** directories to the `PATH` environment variable in your **.cshrc** file.

4. Add the **$ALT_HOME/viewlogic/standard** directory to the `WDIR` environment variable in your **.cshrc** file using the following syntax:

   `setenv WDIR $ALT_HOME/viewlogic/standard:/<`*Powerview system directory*`>/standard ↵`

   ☞ Make sure the **$ALT_HOME/viewlogic/standard** directory is the first directory in your `WDIR` path.

5. Source your **.cshrc** file by typing `source .cshrc ↵` at the UNIX prompt.

6. Create the Viewlogic Powerview **viewdraw.ini** configuration file.

7. Copy the **/usr/maxplus2/maxplus2.ini** file to your **$HOME** directory:

   ```
   cp /usr/maxplus2/maxplus2.ini $HOME ↵

   chmod u+w $HOME/maxplus2.ini ↵
   ```

   The **maxplus2.ini** file contains both Altera- and user-specified initialization parameters that control the MAX+PLUS II software, such as MAX+PLUS II symbol and logic function library paths and the current project name. The MAX+PLUS II installation procedure creates and copies the **maxplus2.ini** file to the **/usr/maxplus2** directory.

   ☞ Normally, you do not have to edit your local copy of **maxplus2.ini**, because the MAX+PLUS II software updates the file automatically whenever you change any parameters or settings. However, if you move the **max2lib** and **max2inc** library subdirectories, you must update the file. Go to "Creating & Using a Local Copy of the **maxplus2.ini** File" in MAX+PLUS II Help for more information.

8. If you plan to instantiate Library of Parameterized Modules (LPM) functions in ViewDraw schematics, you must create a new file with the name **vdraw.vs**. The **vdraw.vs** file must include the following line:

   ```
   load ("vdpath")
   ```

   You must also make sure that you specify the **vdraw.vs** file in your `WDIR` path.

9. Set up a directory structure that facilitates working with the MAX+PLUS II/Viewlogic Powerview interface. Refer to MAX+PLUS II/Viewlogic Powerview Project File Structure.

Related Topics:

- Go to *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual for more information on installation and details on the directories that are created during MAX+PLUS II installation. Go to MAX+PLUS II/Viewlogic Powerview Interface File Organization for information about the MAX+PLUS II/Viewlogic Powerview directories that are created during MAX+PLUS II installation.
- Go to the following topics, which are available on the web, for additional information:
  - ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
  - This manual is also available in 4 parts:
    - Preface & Section 1: MAX+PLUS II Installation
    - Section 2: MAX+PLUS II - A Perspective
    - Section 3: MAX+PLUS II Tutorial
    - Appendices, Glossary & Index

MAX+PLUS II/Viewlogic Powerview Software Requirements

The following applications and utilities are used to generate, process, synthesize, and verify a project with MAX+PLUS® II and Viewlogic Powerview software.

Viewlogic		Altera
ViewDraw	ViewGen	MAX+PLUS II version 9.4
VHDL Analyzer	ViewPath (optional)	
Vantage VHDL Analyzer	ViewTrace	
VHDL -> sym	ViewData Path	
edifneto	MOTIVE version 5.1.6 *Note (1)*	
edifneti	MOTIVE for Powerview version 3.2.1 (optional) *Note (1)*	
EEDIF (optional)	SDF2MTV (optional)	
MMP (optional)	Fusion/VCS	
vsm		

Note:

(1) MOTIVE for Powerview, a wrapper application for MOTIVE, provides a graphical user interface for the utilities (i.e., EEDIF, SDF2MTV, and MMP) used during a static timing verification with MOTIVE. MOTIVE alone does not accept EDIF files through the Setup Advisor.

☞ The MAX+PLUS II **read.me** file provides up-to-date information on which versions of Viewlogic Powerview applications the current version of the MAX+PLUS II software supports. It also provides information on installation and operating requirements. You should read the **read.me** file on the CD-ROM before installing the MAX+PLUS II software. After installation, you can open the **read.me** file from the MAX+PLUS II Help menu.

MAX+PLUS II/Viewlogic Powerview Interface File Organization

Table 1 shows the MAX+PLUS® II/Viewlogic Powerview interface subdirectories that are created in the MAX+PLUS II system directory (by default, the **/usr/maxplus2** directory) during MAX+PLUS II installation.

☞ For information on the other directories that are created during MAX+PLUS II installation, see "MAX+PLUS II File Organization" in *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual.

Table 1. MAX+PLUS II Directory Organization

Directory	Description
./lmf	Contains the Altera-provided Library Mapping File, **vwlogic.lmf**, that maps Viewlogic logic functions to equivalent MAX+PLUS II logic functions.
./viewlogic	Contains the **alt_edif.cfg** EDIF configuration file that is used with the **edifneti** utility. Also contains the library and sample subdirectories.
./viewlogic/examples	Contains the sample Viewlogic designs.
./viewlogic/library/max2sim	Contains the MAX+PLUS II simulation model library (**max2_sim**) for use in ViewSim software.
./viewlogic/library/alt_max2	Contains MAX+PLUS II primitives (`EXP`, `GLOBAL`, `LCELL`, `SOFT`, `CARRY`, `CASCADE`, `DFFE`, `DFFE6K`, and `OPNDRN`), macrofunctions (`a_8fadd`, `a_8mcomp`, `a_8count`, `a_81mux`), and megafunctions (`clklock`) for use in ViewDraw schematics. These logic functions support specific architectural features of Altera® devices. The **alt_max2** library also contains modified versions of the ViewDraw primitives that use tri-state buffers, because these primitives require special handling in the MAX+PLUS II /Viewlogic Powerview interface.
./viewlogic/library/synlib	Contains the Altera-provided synthesis library **altera**, which includes MAX+PLUS II primitives, the **altera.sml** file, a **sym** directory, and a **wir** directory for use with ViewSynthesis software.
./viewlogic/library/alt_mf	Contains the VHDL models for the MAX+PLUS II primitives (`EXP`, `GLOBAL`, `LCELL`, `SOFT`, `CARRY`, `CASCADE`, `DFFE`, and `OPNDRN`), macrofunctions (`clklock`) for use with ViewSynthesis software, the Vantage VHDL Analyzer software, and the VHDL source files. These logic functions are used to maintain portability to other architectures.
./viewlogic/library/alt_time	Contains MOTIVE timing models for MAX+PLUS II logic functions (**motive.lib**), including the `clklock` megafunction, and MAX+PLUS II driver models (**motive.drv**).
./viewlogic/library/alt_vtl	Contains the VHDL source files for the VITAL 3.0-compliant library. This library is available for ViewSim software.
./viewlogic/bin	Contains all MAX+PLUS II, Viewlogic, and interface-related scripts.
./viewlogic/standard	Contains all standard **.ini** files and standard tools.

Related Topics:

- Go to the following topics, which are available on the web, for additional information:
  - ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
  - This manual is also available in 4 parts:
    - Preface & Section 1: MAX+PLUS II Installation
    - Section 2: MAX+PLUS II - A Perspective
    - Section 3: MAX+PLUS II Tutorial
    - Appendices, Glossary & Index

Viewlogic Powerview viewdraw.ini Configuration File

Each Powerview project is configured with the **viewdraw.ini** file that resides in the project directory. The `DIR` statements at the end of **viewdraw.ini** are paths to library directories that are used by the various Powerview applications. Figure 1 shows a sample of the `DIR` statements that are required to use the libraries.

Figure 1. Excerpt from viewdraw.ini

```
DIR [pw] .
DIR [r]  /usr/maxplus2/vwlogic/library/alt_max2 (alt_max2)
DIR [r]  /usr/maxplus2/vwlogic/library/max2sim (max2 sim)
```

```
DIR [r]  /usr/maxplus2/vwlogic/library/synlib (altera)
DIR [r]  /usr/maxplus2/vwlogic/library/alt_mf (alt_mf)
DIR [r]  /usr/maxplus2/vwlogic/library/alt_vtl (alt_vtl)
DIR [rm] /<Powerview system directory>/lib/builtin (builtin)
DIR [rm] /<Powerview system directory>/simmods/vl/dip/74ls (vl74ls)
DIR [rm] /<Powerview system directory>/symsets/vl/dip/74ls (vl74ls)
DIR [r]  /<Powerview system directory>/lib/vdpath (vdpath)
```

☞ When you add the libraries to the **/usr/maxplus2/vwlogic/standard/viewdraw.ini** file, they are automatically set when you create a new project. Powerview tools search these libraries sequentially, so it is important to add them in the order in which they are listed in Figure 1.

Table 1 shows the libraries that must be specified in the `DIR` statements in the **viewdraw.ini** file.

Table 1. Powerview Application Libraries

Library	Library Alias	Source	Topics
alt_max2	`alt_max2`	Altera	Graphical elements for ViewDraw
max2sim	`max2_sim`	Altera	Models for project simulation
synlib	`altera`	Altera	VHDL synthesis library for the MAX+PLUS ® II software
alt_mf	`alt_mf`	Altera	VHDL models of MAX+PLUS II logic functions
alt_vtl	`alt_vtl`	Altera	VITAL-compliant primitives
builtin	`builtin`	Altera	Basic primitives such as `INPUT` pins, `OUTPUT` pins, `AND` gates, `OR` gates, etc.
74ls	`vl74ls`	Viewlogic	74-series macrofunctions
vdpath	`vdpath`	Viewlogic	Standard library of parameterized modules (LPM) functions

☞ The Altera-provided libraries must be listed before the Viewlogic-provided libraries in the **viewdraw.ini** file to ensure that the correct versions of the megafunctions, macrofunctions, and primitives are used.

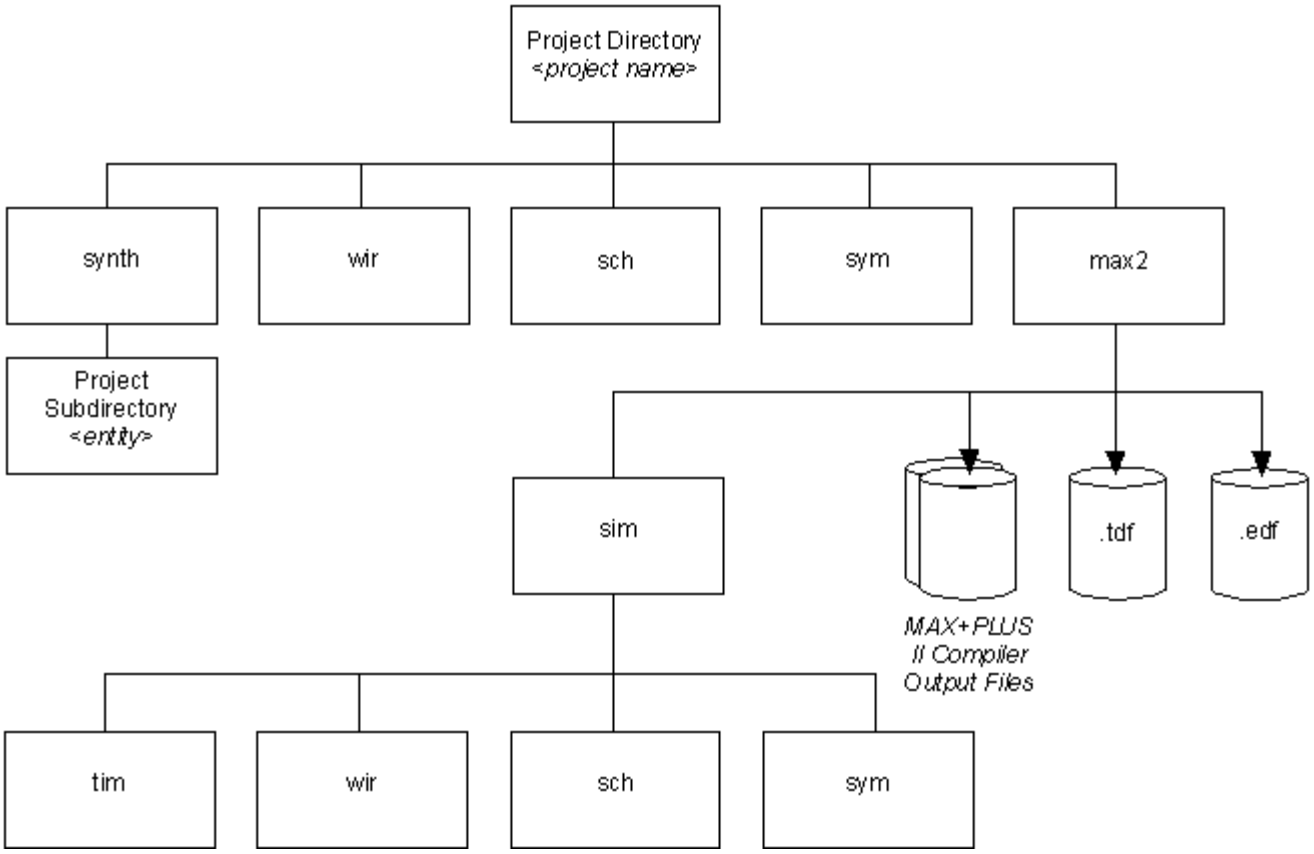
Related Topics:

- Go to Altera-Provided Logic & Symbol Libraries for more information on Altera-supplied libraries. Refer to the Powerview documentation for more information on setting up the **viewdraw.ini** file.

MAX+PLUS II/Viewlogic Powerview Project File Structure

In the MAX+PLUS® II software, a project name is the name of a top-level design file, without the filename extension. This design file can be an EDIF, Verilog HDL, or VHDL netlist file; an Altera® Hardware Description Language (AHDL) TDF; or any other MAX+PLUS II- supported design file. The EDIF netlist file must be created by Powerview and imported into the MAX+PLUS II software as an EDIF Input File (**.edf**). Figure 1 shows an example of MAX+PLUS II project directory structure that includes Powerview-generated files.

Figure 1. Sample MAX+PLUS II Project Organization



The MAX+PLUS II software stores the connectivity data on the links between design files in a hierarchical project in a Hierarchy Interconnect File (**.hif**), but refers to the entire project only by its project name. The MAX+PLUS II Compiler uses the HIF to build a single, fully flattened project database that integrates all the design files in a project hierarchy.

Unlike Powerview, the MAX+PLUS II software does not automatically create a project directory when you create a project. A single directory can contain several MAX+PLUS II design files, and you can specify any one of the designs in the directory as a project in the MAX+PLUS II software.

Viewlogic Powerview Local Work Area Structure

When you create a project with the Powerview Cockpit's **Create** command (Project menu), the project directory is created. You should generate design files and functional simulation files under this directory. A **max2** subdirectory is automatically created under your current project directory when you generate an EDIF file from your schematic or VHDL file. The ***<project name>*.edf** file is stored in the **max2** subdirectory. All MAX+PLUS® II Compiler output files are created in the **/*<project name>*/max2** subdirectory.

☞ ViewDraw files are identified by their directories and not by their extensions, so it is easy to overwrite files unintentionally. To avoid overwriting files, Altera recommends that you create a new project directory, ***<project name>*/max2/sim**, where you can generate all the files needed for simulation.

ViewDraw Project File Structure

Each ViewDraw project directory contains three subdirectories: **wir**, **sch**, and **sym**. See Table 1.

Table 1. ViewDraw Subdirectories

Directory	Topics

./wir	Wirelist files that contain connectivity information for a particular logic block
./sch	Schematics that contain logic
./sym	Symbol files that are the ViewDraw graphical representation of the logic blocks

Each file type uses the filename extension **.1**. Different file types are distinguished only by their directory: **/lib/wir/**<*project name*>**.1** is a wirelist file; **/lib/sch/**<*project name*>**.1** is the corresponding schematic file; and **/lib/sym/**<*project name*>**.1** is the corresponding symbol.

VHDL Project File Structure

Each VHDL project directory contains three subdirectories. See Table 2.

Table 2. VHDL Subdirectories

Directory	Topics
./synth	All synthesis-related files and directories
./synth/<*entity*>	Four types of files: <*entity*>**.pdf**, <*entity*>**.opt**, <*entity*>**.sta**, and <*entity*>**.gnl**
./wir	Wirelist for synthesized VHDL modules

☞ For each VHDL entity in the design, there is a corresponding **./synth/<*entity*>** directory.

Altera-Provided Logic & Symbol Libraries

The MAX+PLUS® II/Viewlogic Powerview environment provides libraries for compiling, synthesizing, and simulating designs.

☞ You can create your own libraries of custom symbols and logic functions for use in ViewDraw schematics and VHDL design files. You can use custom symbols (and functions) to incorporate an EDIF Input File, TDF, or any other MAX+PLUS II-supported design file into a project. The MAX+PLUS II software uses the **vwlogic.lmf** Library Mapping File to map ViewDraw symbols to equivalent MAX+PLUS II megafunctions, macrofunctions, or primitives. To use custom symbols and functions, you can create a custom LMF that maps your custom functions to equivalent EDIF Input Files, TDFs, or other MAX+PLUS II-supported design files. Go to "Library Mapping File" and "Viewlogic Library Mapping File" in MAX+PLUS II Help for more information.

Logic symbols used in ViewDraw software are available from the MAX+PLUS II **alt_max2** library, the ViewDraw **builtin** and **74ls** libraries, and the ViewDatapath **vdpath** library. VHDL models of MAX+PLUS II logic functions are available from the Altera-provided **alt_mf** library.

The alt_max2 Library

The **alt_max2** library provides MAX+PLUS II-specific logic functions that can be used to take advantage of special architectural features in each Altera® device family. See Table 1. Symbols and functional simulation models are available for all of these elements.

The alt_mf Library

The Altera-provided **alt_mf** library, which supports the Viewlogic Vantage VHDL Analyzer software, contains VHDL simulation models for all logic functions listed in the following table. The library is configured so that these functions pass untouched through the EDIF netlist file to the MAX+PLUS II Compiler, providing you with optimal control over design processing. Altera also provides models for all of the logic functions that you can synthesize and simulate. These models allow you to perform functional VHDL simulation while maintaining an architecture-independent VHDL description.

Table 1. Architecture Control Logic Functions

Name *Note (1), Note (2)*	Description	Name	Description	Name	Description
8fadd	8-bit full adder macrofunction	LCELL	Logic cell buffer primitive	EXP	MAX® 5000, MAX 7000, and MAX 9000 Expander buffer primitive
8mcomp	8-bit magnitude comparator macrofunction	GLOBAL	Global input buffer primitive	SOFT	Soft buffer primitive
8count	8-bit up/down counter macrofunction	CASCADE	FLEX® 6000, FLEX 8000, and FLEX 10K cascade buffer primitive	OPNDRN	Open-drain buffer primitive
81mux	8-to-1 multiplexer macrofunction	CARRY	FLEX 6000, FLEX 8000, and FLEX 10K cascade buffer primitive	DFFE *Note (2)*	D-type flipflop with Clock Enable primitive
clklock	Phase-locked loop megafunction				

Notes:

1. Logic function names that begin with a number must be prefixed with "a_" in VHDL designs. For example, 8fadd must be specified as a_8fadd.
2. For designs that are targeted to FLEX 6000 devices, you should use the DFFE primitive only if the design contains either a Clear or Preset signal, but not both. If your design contains both a Clear and a Preset signal, you must use the DFFE6K primitive.

Related Topics:

- Choose **Old-Style Macrofunctions**, **Primitives**, or **Megafunctions/LPM** from the MAX+PLUS II Help menu for detailed information on these functions.
- Go to the following topics, which are available on the web, for additional information:
  - FLEX Devices
  - MAX Devices
  - Classic Device Family

The vdpath & mega_lpm Libraries

The library of parameterized modules (LPM) 2.1.0 standard defines a set of parameterized functions and their corresponding representations in an EDIF netlist file. These logic functions allow you to create and functionally simulate an LPM-based design without targeting a specific device family. After the design is completed, you can target the design to any device family.

When the MAX+PLUS® II software processes projects that include Viewlogic-provided **vdpath** LPM functions, it uses functions from the Altera-provided **mega_lpm** library. This library includes all standard LPM functions except the truth table, finite state machine, and pad functions. Altera does not directly support the lpm_ram_dq, lpm_ram_io, and lpm_rom functions. Refer to Instantiating RAM & ROM Functions in Viewlogic Powerview Designs for instructions on instantiating RAM and ROM functions.

☞ Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu for more information about LPM functions.

MAX+PLUS II/Viewlogic Powerview Design Entry Flow

Figure 1 shows the design entry flow for the MAX+PLUS® II/Viewlogic Powerview interface.

Figure 1. MAX+PLUS II/Viewlogic Powerview Design Entry Flow

Altera-provided items are shown in blue.

Creating ViewDraw Schematics for Use with MAX+PLUS II Software

You can create ViewDraw schematics and convert them into EDIF Input Files (**.edf**) that can be processed with the MAX+PLUS® II software.

To create a ViewDraw schematic for use with the MAX+PLUS II software, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II /Viewlogic Powerview Working Environment.
2. Start Powerview by typing `powerview` ↵ at a UNIX prompt.
3. In the Cockpit window, select *Altera* in the *Current ToolBox* drop-down list box, and select the drawer you want to use, i.e., *Design Tools* or *Max2 Express*, in the *Current Drawer* drop-down list box.
4. Choose **Create** (Project menu) from your working directory to create your project directory. Choose **OK**.
5. Choose **SearchOrder** (Project menu) to add the appropriate library directories and aliases to your **viewdraw.ini** file in the appropriate search order. Refer to Viewlogic Powerview **viewdraw.ini** Configuration File for more information on Powerview application libraries.
6. Start ViewDraw by double-clicking Button 1 on the **max2_VDraw** icon in the drawer that you selected in step 3, type the name of the schematic, and choose **OK**. You can also start the ViewDraw software by typing `viewdraw` ↵ at the UNIX prompt.
7. Choose **Comp** (Add menu) to add components to the schematic. You can use functions from the **alt_max2**, **builtin**, and **74ls** libraries. For information on Altera-provided libraries, go to Altera-Provided Logic & Symbol Libraries.

   Instructions for instantiating specific functions are provided in the following MAX+PLUS II ACCESS℠ Key topics:

   - Instantiating LPM Functions in ViewDraw Schematics
   - Instantiating RAM & ROM Functions in Viewlogic Powerview Designs

   ☞ You can instantiate MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP℠). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.

8. If you instantiate a `clklock` megafunction, choose the **Dialog** command (Attr menu), then choose the **All** command (Dialog menu) to specify the values of the `INPUT_FREQUENCY` and `CLOCKBOOST` parameters. For detailed information on the `clklock` megafunction, choose **Megafunctions/LPM** from the MAX+PLUS II Help menu.

9. If you wish to create a hierarchical design that contains symbols representing other design files, such as Altera® Hardware Description Language (AHDL) Text Design Files (**.tdf**), go to Creating Hierarchical Projects in ViewDraw Schematics.

10. Choose **Net** (Add menu) to add nets to the schematic.

11. Choose **Bus** (Add menu) to add buses to the schematic.

12. Choose **Label** (Add menu) to attach labels to nets and buses. When you are naming and labeling buses, make sure you use the format <*bus name*>[<*most significant bit*>:<*least significant bit*>], and that you label both the net and the pin.

13. (Optional) To enter resource assignments in your schematic, select a symbol or a net that feeds an output and use the **Attr** command (Add menu) to add the assignments. For more information, go to Entering Resource Assignments. You can also enter resource assignments from the MAX+PLUS II software.

14. Choose **Write** (File menu) to check and save both the schematic with the name **.sch/<*design name*>.1** and the wirelist with the name **./wir/<*design name*>.1**.

15. (Optional) Perform a functional simulation, as described in Performing a Functional Simulation with ViewSim Software.

16. Once you have created a schematic, you can generate an EDIF netlist file that can be imported into the MAX+PLUS II software with either of the following methods:

    - You can create an EDIF netlist file, as described in Converting ViewDraw Schematics or VHDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the **edifneto** Utility You must use this method if your ViewDraw schematic instantiates Library of Parameterized Modules (LPM) functions.
    - You can use the **SCH <-> max2** utility in the Max2 Express drawer to automatically create an EDIF netlist file, compile it with the MAX+PLUS II Compiler, generate an EDIF Output File (**.edo**), and generate a **.vsm** file for simulation, as described in Using the Max2 Express Drawer's **SCH <-> max2** Utility.

Installing the Altera-provided MAX+PLUS II/Viewlogic interface on your computer automatically creates the following sample ViewDraw schematic files:

- **/usr/maxplus2/examples/viewlogic/example1/fadd**
- **/usr/maxplus2/examples/viewlogic/example3/fadd2**
- **/usr/maxplus2/examples/viewlogic/example4/fadd2mpp**
- **/usr/maxplus2/examples/viewlogic/example7/fifo**

Related Topics:

- Go to Powerview Command-Line Syntax in these MAX+PLUS II ACCESS Key topics for related information.

Instantiating LPM Functions in ViewDraw Schematics

You can instantiate library of parameterized modules (LPM) functions from the **vdpath** library in ViewDraw

schematics.

Altera does not directly support the `lpm_ram_dq`, `lpm_ram_io`, and `lpm_rom` functions. Go to Instantiating RAM & ROM Functions in Viewlogic Powerview Designs for information on instantiating these functions.

To instantiate an LPM function, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Viewlogic Powerview Working Environment. Make sure that you have created a **vdraw.vs** file, as described in step 8 of that topic.
2. Choose **Cell** (Add menu).
3. Choose an *<LPM function name>* to open the *<LPM function name>* dialog box. Specify a symbol name for *Symbol Prefix* and specify appropriate parameters. Choose **OK**.

   The ViewDraw software generates the specified symbol name symbol according to your specifications. It also generates a corresponding VHDL simulation model, but it is compiled only after you save the schematic. If you want to change the settings for the symbol, select the instance and choose **Cell** (Change menu) to re-open the appropriate dialog box.
4. Continue with the steps necessary to complete your schematic, as described in Creating ViewDraw Schematics for Use with MAX+PLUS II Software.

Related Topics:

- Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu for detailed information on LPM functions.

Instantiating RAM & ROM Functions in Viewlogic Powerview Designs

The MAX+PLUS® II /Viewlogic Powerview interface offers full support for the memory capabilities of the FLEX® 10K device family, including synchronous and asynchronous RAM and ROM, cycle-shared dual-port RAM, dual-port RAM, single-Clock FIFO, and dual-Clock FIFO functions. You can use the Altera-provided **genmem** utility to generate functional simulation models and timing models for these functions. Type `genmem` ↵ at the UNIX prompt to display information on how to use this utility, as well as a list of the functions you can generate. RAM and ROM can be instantiated in both ViewDraw schematics and VHDL designs.

Refer to Viewlogic documentation for information on simulating projects that contain RAM functions. The procedure for reading an EDIF Output File and preparing it for simulation with ViewSim requires additional steps when the project contains RAM functions.

When you instantiate a RAM or ROM function, follow these general guidelines:

- For ROM functions, you must specify an initial memory content file in the Intel hexadecimal format (**.hex**) or the Altera® Memory Initialization File (**.mif**) format. The filename must be the same as the instance name; e.g., the instance name must be unique throughout the whole project, and must contain only valid name characters. The initialization file must reside in the directory containing the project's design files.
- For RAM functions, specifying a memory initialization file is optional.
- For VHDL designs, specify the name of the initial memory content file in the Generic Map Clause of the instance, with the specified type `LPM_FILE`. If you do not use an initial memory content file (e.g., for a RAM

function), you should not declare or use the Generic Clause.

- Do not synthesize the **genmem**-generated VHDL file: it is intended for simulation only.

The MIF format is supported only for specifying initial memory content when compiling designs within the MAX+PLUS II software. You cannot use a MIF to perform simulation with Viewlogic tools prior to MAX+PLUS II compilation.

To instantiate RAM or ROM in a ViewDraw schematic, follow these steps:

1. Use the **genmem** utility to generate a memory model by typing the following command at the UNIX prompt:

   `genmem` <*memory type*> <*memory size*> `-vwlogic` ↵

   For example: `genmem asynrom 256x15 -vwlogic` ↵

2. Start the VHDL-to-symbol utility, **vhdl2sym**, by double-clicking Button 1 on the **max2_vhdl2sym** icon in the Altera® Toolbox Design Tools Drawer.

3. Specify the following options in the **vhdl2sym** dialog box and choose **OK** to create a symbol. For example, to create the symbol for a 256x15 asynchronous ROM, enter the following settings:

Option:	**Setting:**
VHDL Source Filename	*asyn_rom_256x15.vhd*
Add LEVEL attribute	*On*

4. Choose **Comp** (Add menu), type <*design name*> in the *Enter Name* box, and choose **OK**.

To instantiate a RAM or ROM function in VHDL, follow these steps:

1. Repeat step 1 above.

2. Create a VHDL design that incorporates the text from the **genmem**-generated Component Declaration, <*memory name*>**.cmp**, and instantiate the <*memory name*> function.

Figure 1 shows a VHDL design that instantiates asyn_rom_256x15.vhd, a 256 x 15 ROM function.

Figure 1. VHDL Design File with ROM Instantiation (tstrom.vhd)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;



ENTITY tstrom IS
        PORT (
          addr    : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
          memenab : IN STD_LOGIC;
          q       : OUT STD_LOGIC_VECTOR (14 DOWNTO 0));
END tstrom;



ARCHITECTURE behavior OF tstrom IS

COMPONENT asyn_rom_256x15
     GENERIC (LPM_FILE : string);
```

```
PORT (Address : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
           MemEnab : IN STD_LOGIC;
           Q       : OUT STD_LOGIC_VECTOR(14 DOWNTO 0)
     );
END COMPONENT;



BEGIN

   u1: asyn_rom_256x15
        GENERIC MAP (LPM_FILE => "u1.hex")
   PORT MAP (Address => addr, MemEnab => memenab, Q =>q);
END behavior;
```

Related Topics:

- Go to FLEX 10K Device Family, which is available on the web, for additional information.

Creating Hierarchical Projects in ViewDraw Schematics

You can incorporate any MAX+PLUS® II-supported design file, such as an Altera® Hardware Description Language (AHDL) Text Design File (**.tdf**), into a project hierarchy that consists of both schematic and text files. To incorporate a non-ViewDraw design file into a higher-level schematic design, you must create a hollow-body symbol for it in the ViewDraw software. During compilation, the MAX+PLUS II software recognizes the symbol as an identifier for the design file, and inserts the correct logic and connections. You can incorporate any number of design files into a project hierarchy.

To create a hierarchical project in your ViewDraw schematic, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Viewlogic Powerview Working Environment.
2. Create a ViewDraw schematic and save it in your working directory, as described in Creating ViewDraw Schematics for Use with MAX+PLUS II Software.

   ☞ You can instantiate MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.

3. Create a design file that uses all uppercase letters for the function name and all lowercase letters for the file extension, e.g., **DECODE.tdf**. This naming convention is required to prevent conflicts when the file is incorporated into a hierarchical design. When the **edifneto** utility generates an EDIF netlist file from the ViewDraw schematic, it copies the name of the hollow-body symbol in uppercase letters, regardless of the case that appears in the schematic.
4. Double-click Button 1 on the **max2_VDraw** icon in the Altera Toolbox Design Tools Drawer to start ViewDraw.
5. In the **File Open** dialog box, type <*design name*>, i.e., the name of the hollow-body symbol you want to create. Turn on the *Symbol* option and choose **OK**. The Symbol Editor is displayed.
6. Choose **Block Size Z-WxH** (Change menu) and select a symbol size.

7. Choose **Graphics-Box** (Add menu) to draw the symbol body.
8. Choose **Pin** (Add menu) to enter pinstubs.
9. Select a pin and choose **Label** (Add menu) to label the pin names.
10. (Optional) **Choose Graphics-Text** (Add menu) to label the symbol.
11. Choose **Block Type Module** (Change menu). You must choose **Block Type Module** to specify that no Viewlogic schematic is available to represent the functionality of the symbol.
12. Choose **Write** (File menu) to save the symbol.
13. In the top-level ViewDraw schematic, choose **Comp** (Add menu), select the name of the symbol, and choose **OK**.
14. The MAX+PLUS II software uses Library Mapping Files (**.lmf**) to map standard ViewDraw symbols to equivalent MAX+PLUS II megafunctions, macrofunctions, or primitives. To use custom symbols, you can create a custom LMF that maps your custom symbols to the equivalent EDIF Input File, TDF, or other design file.

    ☞ You will also need to specify a Library Mapping File (.lmf) in the EDIF Netlist Reader Settings dialog box before compiling with the MAX+PLUS II Software. Go to Compiling Projects with MAX+PLUS II Software for more information.

15. Continue with the steps necessary to complete your ViewDraw schematic, as described in Creating ViewDraw Schematics for Use with MAX+PLUS II Software.

Related Topics:

- Go to Creating AHDL Designs for Use with MAX+PLUS II Software in these MAX+PLUS II ACCESS℠ Key topics for related information.

Entering Resource Assignments

The MAX+PLUS® II software allows you to enter a variety of resource and device assignments for your projects. Resource assignments are used to assign logic functions to a particular pin, logic cell, I/O cell, embedded cell, row, column, Logic Array Block (LAB), Embedded Array Block (EAB), chip, clique, local routing, logic option, timing requirement, or connected pin group. In the MAX+PLUS II software, you can enter all types of resource and device assignments with Assign menu commands. You can also enter pin, logic cell, I/O cell, embedded cell, LAB, EAB, row, and column assignments in the MAX+PLUS II Floorplan Editor. The Assign menu commands and the Floorplan Editor all save assignment information in the ASCII Assignment & Configuration File (**.acf**) for the project. In addition, you can edit ACFs manually in any standard text editor.

ViewDraw Schematics

In ViewDraw schematics, you can assign a limited subset of these resource assignments by assigning properties to symbols. These properties are incorporated into the EDIF netlist file(s). The MAX+PLUS II software automatically converts assignment information from the EDIF Input File (**.edf**) into the ACF format. For information on making MAX+PLUS II-compatible resource assignments, go to the following topics:

- Assigning Pins, Logic Cells & Chips
- Assigning Cliques
- Assigning Logic Options

- Modifying the Assignment & Configuration File with the **setacf** Utility

Installing the Altera-provided MAX+PLUS II/Viewlogic interface on your computer automatically creates the following sample ViewDraw schematic file, which includes resource assignments:

- **/usr/maxplus2/examples/viewlogic/example4/fadd2mpp**

Related Topics:

- Go to Viewlogic documentation for information on how to assign properties. Go to "Resource Assignments in EDIF Input Files" and "Assigning Resources in a Third-Party Design Editor" in MAX+PLUS II Help for more information on assignments or properties that can be assigned in ViewDraw.

VHDL Design Files

For VHDL-based designs, you must use the MAX+PLUS II software or the **setacf** utility to enter resource assignments. For information on using the **setacf** utility, go to Modifying the Assignment & Configuration File with the **setacf** Utility.

Related Topics:

- For information on entering assignments in the MAX+PLUS II software with Assign menu commands or in an ACF, go to "resource assignments" or "ACF, format" in MAX+PLUS II Help using **Search for Help on** (Help menu).

Assigning Pins, Logic Cells & Chips

You can assign a single logic function to a specific pin or logic cell (including I/O cells and embedded cells) within a chip, and assign one or more functions to a specific chip. A chip is a group of logic functions defined as a single, named unit, which can be assigned to a specific device.

You can assign a signal to a particular pin to ensure that the signal is always associated with that pin, regardless of future changes to the project. If you wish to set and maintain the performance of your project, assigning logic to a specific logic cell within a chip can minimize timing delays. In a project that is partitioned among multiple devices, you can assign logic functions that must be kept together in the same device to a chip. Chip assignments allow you to split a project so that only a minimum number of signals travel between devices, and to ensure that no unnecessary device-to-device delays exist on critical timing paths. You can assign a chip to a device in some EDA tools or in the MAX+PLUS® II software.

Use the following syntax for chip, pin, and logic cell assignments:

- To assign a logic function to a chip:

  `CHIP_PIN_LC=<`*chip name*`>`

  For example: `CHIP_PIN_LC=chip1`

- To assign a pin number within a chip:

  `CHIP_PIN_LC=<`*chip name*`>@<`*pin number*`>`

  For example: `CHIP_PIN_LC=chip1@K2`

- To assign a logic cell, I/O cell, or embedded cell number:

CHIP_PIN_LC=<*chip name*>@LC<*logic cell number*>

CHIP_PIN_LC=<*chip name*>@IOC<*I/O cell number*>

CHIP_PIN_LC=<*chip name*>@EC<*embedded cell number*>

For example: CHIP_PIN_LC=chip1@LC44

Related Topics:

- Refer to the following sources for additional information:
  - Go to "Devices & Adapters" and "Assigning a Device" in MAX+PLUS II Help for information on device pin-outs and assigning devices, respectively, in the MAX+PLUS II software.
  - Go to Back-Annotating MAX+PLUS II Pin Assignments to Design Architect Symbols for information on back-annotating pin assignments in Mentor Graphics Design Architect schematics.

Assigning Cliques

You can define a group of logic functions as a single, named unit, called a clique. The MAX+PLUS® II Compiler attempts to place all logic in the clique in the same logic array block (LAB) to ensure optimum speed. If the project does not use multi-LAB devices, or if it is not possible to fit all clique members into a single LAB, the clique assignment ensures that all members of a clique are placed in the same device. In FLEX® 6000, FLEX 8000, FLEX 10K, and MAX® 9000 devices the Compiler also attempts to place the logic in LABs in the same row. Cliques therefore allow you to partition a project so that only a minimum number of signals travel between LABs, and to ensure that no unnecessary LAB-to-LAB or device-to-device delays exist on critical timing paths.

To assign a clique, use the following syntax:

✔ CLIQUE=<*clique name*>

For example: CLIQUE=fast1

Related Topics:

- Go to the following topics in MAX+PLUS II Help for related information:
  - Assigning a Clique
  - Guidelines for Achieving Maximum Speed Performance

Assigning Logic Options

Logic option and logic synthesis style assignments allow you to guide logic synthesis with logic optimization features that are specific to Altera® devices. You can assign logic options and styles to individual logic functions in your design. The MAX+PLUS® II Compiler also uses a device-family-specific default logic synthesis style for each project.

Related Topics:

- Go to "Resource Assignments in EDIF Input Files" and "Assigning Resources in a Third-Party Design Editor" in MAX+PLUS II Help for complete and up-to-date information on logic option and logic synthesis

style assignments, including definitions and syntax of these assignments.

Modifying the Assignment & Configuration File with the setacf Utility

Altera provides the **setacf** utility to help you modify a project's Assignment & Configuration File (**.acf**) from the command line, without opening the file with a text editor. Type `setacf -h` ↵ at a UNIX or DOS prompt to get help on this utility.

Performing a Functional Simulation with ViewSim Software

You can use Viewlogic ViewSim software to perform a functional simulation of a ViewDraw schematic or a VHDL Design File (**.vhd**) before compiling your project with the MAX+PLUS II Compiler. Follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II /Viewlogic Powerview Working Environment.

2. Create a ViewDraw schematic that follows the guidelines in Creating ViewDraw Schematics for Use with MAX+PLUS II Software. Then go to step 3.

   or:

   Create a VHDL Design File <*design name*>**.vhd** and analyze it, as described in the following MAX+PLUS II ACCESS^{SM} Key topics:

   - Creating VHDL Designs for Use with MAX+PLUS II Software
   - Analyzing VHDL Files with the Vantage VHDL Analyzer Software

   Then go to step 7.

3. With the schematic open in the ViewDraw editor, add `CLR` and `PRE` inputs to any flipflops in your design, or tie the `CLR` and `PRE` ports of the flipflops to VCC. (Use the `PWR` primitive from the **builtin** library.)

4. Choose **Write To** (File menu) and save the schematic as <*design name*>**_funct**.

5. Start the **vsm** utility by double-clicking Button 1 on the **max2_vsmnet** icon in the Altera® Toolbox Design Tools Drawer.

6. Specify the following options in the **vsm** dialog box and choose **OK** to generate the <*design name*>**_funct.vsm** file:

Option:	**Setting:**
Design Name	<*design name*>*_funct*
Level	(blank)

7. Create a simulation command file (**.cmd**) for simulation with ViewSim software. Alternatively, you can enter commands at the prompt in the ViewSim window. Refer to your Viewlogic documentation for more information on creating ViewSim command files.

8. Start the ViewSim simulation tool by double-clicking Button 1 on the **max2_VSim** icon in the Design Tools Drawer.

9. If you wish to simulate a ViewDraw schematic, specify the following options in the **ViewSim** dialog box, then go to step 11.

Option:	Setting:
Design Name	*<design name>_funct*
Command File	*<design name>_funct.cmd*
VHDL Source Window	*OFF*
VHDL Debugging	*OFF*

10. If you wish to simulate a VHDL design, specify the following options in the **ViewSim** dialog box:

Option:	Setting:
Design Name	*<design name>*
Command File	*<design name>.cmd*
Graphical Interface	*ON*
VHDL Source Window	*OFF* or *ON*
VHDL Debugging	*OFF* or *ON*

11. Choose **OK** to simulate the design. ViewSim software simulates the design and starts the ViewTrace waveform editor to allow you to observe the simulation results.

12. Use the **edifneto** utility to generate an EDIF Netlist File (**.edf**) that can be imported into the MAX+PLUS II software, as described in Converting ViewDraw Schematics or VHDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the **edifneto** Utility.

Related Topics:

- Go to ViewSim documentation for complete details on simulating a project and using ViewTrace to observe waveform output results.

Converting ViewDraw Schematics or VHDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the edifneto Utility

You can use the **edifneto** utility to generate an EDIF netlist file from a ViewDraw schematic or VHDL Design File (**.vhd**). This file can be imported into the MAX+PLUS® II software as an EDIF Input File with the extension **.edf**. To generate an EDIF netlist file, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Viewlogic Powerview Working Environment.

2. Create a ViewDraw schematic and save it in your working directory, as described in Creating ViewDraw Schematics for Use with MAX+PLUS II Software.

   or:

   Create a VHDL Design File, analyze it, and synthesize and optimize it, as described in the following topics:

   - Creating VHDL Designs for Use with MAX+PLUS II Software
   - Analyzing VHDL Files with the Vantage VHDL Analyzer Software
   - Synthesizing & Optimizing VHDL Designs with ViewSynthesis Software

3. Start the **edifneto** utility by double-clicking Button 1 on the **max2_edifo** icon in the Design Tools Drawer or the Max2 Express Drawer in the Altera Toolbox. You can also start the **edifneto** utility by

typing `edifneto` at the UNIX prompt.

4. If you are converting a ViewDraw schematic, specify the <*design name*> for the *Wire File Name* option in the **edifneto** dialog box. If you are not using the Altera® toolbox, do not specify *Altera* for the *Level* option in the **edifneto** dialog box.

5. If you are converting a VHDL Design File, or if your ViewDraw schematic instantiates Library of Parameterized Modules (LPM) functions, specify *Altera* and *VHDL* as the *Level* in the **edifneto** dialog box.

6. Choose **OK** to generate the EDIF netlist file. The **edifneto** utility creates the **max2** subdirectory under your working directory. The **max2** subdirectory contains the EDIF netlist file for your design.

   ☞ When the **edifneto** utility generates an EDIF netlist file from a design that instantiates LPM functions, the EDIF netlist file may contain parameters with incorrect parameter names. To correct this problem, go to the **/usr/maxplus2/viewlogic/bin** directory and type `chlpmpty` <*design name*>`.edf` ↵ at the UNIX prompt to run the Altera-provided **chlpmpty** script, which converts all of the parameters to their correct names.

7. Process the <*design name*>**.edf** with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Related Topics:

- Go to the following MAX+PLUS II ACCESS℠ Key topics for related information:
  - Using the Max2 Express Drawer's **SCH <-> max2** Utility
  - Using the Max2 Express Drawer's **VHDL <-> max2** Utility

---

Viewlogic Powerview Graphical User Interface & the Altera Toolbox

You use the Powerview graphical interface manager, the Cockpit, and the Altera® Toolbox to start all Powerview and Altera tools. Within the Altera Toolbox, you can specify the Max2 Express Drawer or the Design Tools Drawer to work with the Altera/Viewlogic Powerview interface.

The Max2 Express Drawer provides a quick and seamless way to transfer designs created in Powerview to the MAX+PLUS® II software for compilation, then return the compiled designs to Powerview for simulation and timing verification. Table 1 describes the Max2 Express Drawer tools.

Table 1. Max2 Express Drawer Tools

Tool	Description
max2_VDraw	Launches the Powerview ViewDraw schematic entry tool.
VHDL<->max2	Launches all tools necessary to synthesize a VHDL design, compile for an Altera device, and generate a **.vsm** file for simulation with the Powerview ViewSim simulator.
SCH<->max2	Launches all tools necessary to compile a schematic design entered with Powerview ViewDraw software for an Altera device and to generate a **.vsm** file for simulation with Powerview ViewSim and **.edo**, **.sdo**, and **.vmo** files for timing analysis with MOTIVE for Powerview.
max2_VSim	Launches the Powerview ViewSim simulator.
max2_VTrace	Launches the Powerview ViewTrace simulation waveform editor.
max2_MOTIVE	Launches the MOTIVE for Powerview ViewDraw static timing verification tool.

The Design Tools Drawer provides tools that enable you to create a design with the Powerview tools, compile the design in the MAX+PLUS II software, and simulate and verify the design with Powerview software. Table 2 describes the Design Tools Drawer tools.

Table 2. Design Tools Drawer Tools

Tool	Description
max2_VDraw	Launches the Powerview ViewDraw schematic entry tool.
max2_analyzer	Launches the Powerview VHDL Analyzer software.
max2_syn	Launches the Powerview VHDL synthesis tool.
max2_chk	Launches the Powerview schematic verification tool.
max2_vsmnet	Launches the Powerview **vsm** utility that converts a wirelist file into a **.vsm** file.
max2_VSim	Launches the Powerview ViewSim simulator.
max2_VTrace	Launches the Powerview ViewTrace simulator.
max2_edifo	Launches the Powerview EDIF netlist writer, **edifneto**.
max2_VGen	Launches the Powerview ViewGen utility that generates a schematic from a wirelist file.
max2	Launches the MAX+PLUS II Compiler.
max2_edifi	Launches the Powerview EDIF Netlist Reader, **edifneti**.
max2_vhdl2sym	Launches the Powerview **vhdl2sym** utility that generates a symbol from a VHDL file.
max2_VantgMgr	Launches the Powerview Vantage VHDL Library Manager tool.
max2_VantgAnlz	Launches the Vantage VHDL Analyzer software.
max2_VCS	Launches the Fusion/VCS Simulator.
max2_MOTIVE	Launches the MOTIVE for Powerview static timing verification tool.

Compiling Projects with MAX+PLUS II Software

The MAX+PLUS® II Compiler can process design files in a variety of formats. This topic describes how to use MAX+PLUS II software to compile projects in which the top-level design file is an EDIF Input File (with the extension **.edf**).

☞ Refer to the following sources for additional information:

- Go to MAX+PLUS II Help for information on compiling VHDL and Verilog HDL, design files directly with the MAX+PLUS II Compiler.
- Go to Running Synopsys Compilers from MAX+PLUS II Software for information on running the Synopsys Design Compiler or FPGA Compiler software on a VHDL or Verilog HDL design from within the MAX+PLUS II Compiler window.

To compile a design (also called a "project") with MAX+PLUS II software, go through the following steps:

1. Create design files that are compatible with the MAX+PLUS II software and convert them into EDIF Input Files with the extension **.edf**. Specific instructions for some tools are described in these MAX+PLUS II ACCESS℠ Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your design entry or synthesis and optimization tool.
2. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, create a mapping for each function in a Library Mapping File (**.lmf**) to map the custom symbol to the corresponding EDIF Input File, AHDL Text Design File (**.tdf**), or other MAX+PLUS II-

supported design file. These custom functions are represented in design files as hollow-body symbols or "black box" HDL descriptions.

Go to "Library Mapping Files (**.lmf**)" in MAX+PLUS II Help for more information.

3. Open MAX+PLUS II and specify the name of your top-level design file as the project name with the **Project Name** command (File menu). If you open an HDL file in the MAX+PLUS II Text Editor, you can choose the **Project Set Project to Current File** command (File menu) instead.

You can also compile a project from a command line. However, the first time you compile a project, the settings you need to specify are easier to specify from within the MAX+PLUS II software. After you have run the graphical user interface for the MAX+PLUS II software at least once, you can more easily use the command-line **setacf** utility to modify options in the Assignment & Configuration File (**.acf**) for the project. Type `setacf -h` ↵ and `maxplus2 -h` ↵ for descriptions of **setacf** and MAX+PLUS II command-line syntax.

4. Choose **Device** (Assign menu) and select the target Altera device family in the *Device Family* drop-down list box. If you wish to implement the design logic in a specific device, select it in the *Devices* box. Otherwise, select *AUTO* to allow the MAX+PLUS II Compiler to choose the best device(s) in the current device family. If your design entry or synthesis and optimization tool required you to specify a target family and/or device, specify the same information in this dialog box. For information on partitioning logic among multiple devices, go to MAX+PLUS II Help. Choose **OK**.

5. Open the Compiler window by choosing the **Compiler** command (MAX+PLUS II menu). Go through the following steps to specify the options necessary to compile the design file(s) in your project:

    1. Ensure that all EDIF netlist files have the extension **.edf** and choose **EDIF Netlist Reader Settings** (Interfaces menu).

    2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor. This name should be the name of the vendor whose tool(s) you used to create the EDIF netlist files. If your vendor name does not appear, select *Custom* instead.

If you are compiling a design created with Synopsys FPGA Express software, select *Synopsys*, choose the **Customize** button, enter <*project name*>**.lmf** in the *LMF #1* box, choose **OK**, and skip to step 6.

    3. If you selected an existing vendor name in the *Vendor* box and your project contains design files that require custom LMF mappings, choose the **Customize** button to expand the dialog box to show all settings. Turn on the *LMF #2* checkbox and type your custom LMF's filename in the corresponding text box, or select a name from the *Files* box. The selection in the *Vendor* box will change to *Custom* and all settings will be retained until you change them again.

    4. If you selected *Custom* in the *Vendor* box, choose the **Customize** button to expand the dialog box to show all settings. Any previously defined custom settings will be displayed. Under *Signal Names*, type one or more names with up to 20 total name characters in the *VCC* or *GND* box if your EDIF Input File(s) use one or more names other than `VCC` or `GND` for the global high or low signals. Multiple signal names must be separated by either a comma (,) or a space. Under *Library Mapping Files*, turn on the *LMF #1* checkbox and type a filename in the text box following it, or select a name from the *Files* box. If necessary, specify another LMF name in the *LMF #2* box. Go to MAX+PLUS II Help for detailed information on the settings available in the **EDIF Netlist Reader Settings** dialog box.

    5. Choose **OK**.

6. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level

logic functions, you may need to ensure that all files are present in your project directory, i.e., the same directory as the top-level design file. Otherwise, you must specify the directories containing these files as user libraries with the **User Libraries** command (Options menu).

7. Follow all guidelines that apply to your design entry or synthesis and optimization tool:

   - Exemplar Logic Galileo Extreme-Specific Compiler Settings
   - Synopsys DesignWare-Specific Compiler Settings
   - Converting Synopsys FPGA Compiler & Design Compiler Timing Constraints into MAX+PLUS II-Compatible Format with the **syn2acf** Utility
   - Synplicity Synplify-Specific Compiler Settings

8. If you wish to generate EDIF, VHDL, or Verilog HDL output files for post-compilation simulation or timing analysis with another EDA tool, go through the following steps:

   1. (Optional) Turn on the **Optimize Timing SNF** command (Processing menu) to reduce the size of the output file(s). Turning on this command can reduce the size of output netlists by up to 30%.

      This command does not create optimized timing SNFs on UNIX workstations. However, a non-optimized timing SNF provides the same functional and timing information as an optimized timing SNF.

   2. If you wish to generate EDIF Output Files (**.edo**), go through these steps:

      1. Turn on the **EDIF Netlist Writer** command (Interfaces menu). Then choose the **EDIF Netlist Writer Settings** command (Interfaces menu).

      2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor and choose **OK**. If your vendor name does not appear, select *Custom* instead and specify the settings that are appropriate for your simulation or timing analysis tool. Go to MAX+PLUS II Help for detailed information on the options available in the **EDIF Netlist Writer Settings** dialog box.

      3. To generate an optional Standard Delay Format (SDF) Output File (**.sdo**), choose the **Customize** button to expand the dialog box to show all settings. Select one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**.

      The filenames of the EDIF Output File(s) and optional SDF Output File(s) are the same as the user-defined chip name(s) for the project; if no chip names exist, the Compiler assigns filenames that are based on the project name. For a multi-device project, the Compiler also generates a top-level EDIF Output File that is uniquely identified by "**_t**" appended to the project name. In addition, the Compiler automatically generates a VHDL Memory Model Output File, *<project name>*.**vmo**, when it generates an EDIF Output File that contains memory (RAM or ROM).

   3. If you wish to generate VHDL Output Files (**.vho**), turn on the **VHDL Netlist Writer** command (Interfaces menu). Then choose **VHDL Netlist Writer Settings** command (Interfaces menu). Select *VHDL Output File (.vho)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF ver. 2.1 files contain timing delay information that allows you to perform back-annotation simulation in VHDL with VITAL-compliant simulation libraries. The VHDL Output Files generated by the Compiler have the extension **.vho**, but are otherwise named in the same way as the EDIF Output Files described above.

   4. If you wish to generate Verilog HDL Output Files (**.vo**), turn on the **Verilog Netlist Writer** command (Interfaces menu). Then choose **Verilog Netlist Writer Settings** command (Interfaces menu). Select *Verilog Output File (.vo)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF Output Files contain timing delay information that allows

you to perform back-annotation simulation in Verilog HDL. The Verilog Output Files generated by the Compiler have the extension **.vo**, but are otherwise named in the same way as the EDIF Output Files described above.

9. To run the MAX+PLUS II Compiler, choose the **Project Save & Compile** command (File menu) or choose the **Start** button in the Compiler window.

   ☞ See step 3 for information on running MAX+PLUS II software from the command line.

10. Once you have compiled the project with the MAX+PLUS II Compiler, you can use the VHDL, Verilog HDL, or EDIF output file(s), and the optional SDF Output File(s) (**.sdo**) to perform timing analysis or timing simulation with another EDA tool. Specific instructions for some tools are described in these MAX+PLUS II ACCESS Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your EDA tool.

The MAX+PLUS II Compiler also generates a Report File (**.rpt**), a Pin-Out File (**.pin**), and one or more of the following files for device programming or configuration:

- JEDEC Files (**.jed**)
- Programmer Object Files (**.pof**)
- SRAM Object Files (**.sof**)
- Hexadecimal (Intel-format) Files (**.hex**)
- Tabular Text Files (**.ttf**)

Related Topics:

- Refer to the following sources for additional information:
  - Go to *Compiler Procedures* in MAX+PLUS II Help for information on other available Compiler settings.
  - Go to *Programmer Procedures* in MAX+PLUS II Help for instructions on creating other types of programming files and on programming or configuring Altera devices.
  - Go to Back-Annotating MAX+PLUS II Pin Assignments to Design Architect Symbols for information on back-annotating pin assignments in Mentor Graphics Design Architect schematics.
  - Go to Programming Altera Devices for information on the different programming hardware options for Altera device families.
- Go to the following topics, which are available on the web, for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware

Using ViewDraw & ViewGen Software to Prepare for Multi-Device Board-Level Simulation with ViewSim Software

In order to perform board-level simulation with ViewSim software, you must generate symbols that represent each MAX+PLUS® II -generated EDIF Output File (**.edo**) and incorporate them into a top-level ViewDraw schematic. You can use ViewGen to generate hollow-body symbols to represent each EDIF Output File, and connect them to other system components in the top-level schematic. You must also edit the wirelist files (**.wir**) created by the **edifneti** utility.

To prepare for multi-device board-level simulation with ViewSim software, follow these steps:

1. Perform steps 1 through 6c in Performing a Timing Simulation with ViewSim Software.

2. Start ViewGen by double-clicking Button 1 on the **max2_VGen** icon in the Design Tools Drawer.

3. Specify the filename of one of the EDIF Output Files <*filename*>.**edf** in the *Name* box in the **ViewGen** dialog box and choose **OK** to generate a corresponding <*filename*> symbol.

4. Repeat step 3 to generate other symbols as needed. You do not need to generate a symbol for the <*filename*>**_t.edf** file.

5. Eliminate the two extra pins for VDD and GND connections from the top-level wirelist file **./wir/**<*design name*>**_t.1**:

    1. Open the **./wir/**<*design name*>**_t.1** wirelist file with a standard text editor and delete the following lines:

        ```
        P IN GND
        I GND IN GND
        P IN VDD
        I VDD IN VDD
        ```

    2. Add the following two lines to the file to ensure global ground and power connections for simulation:

        ```
        G VDD ↵
        G GND ↵
        ```

    3. Save the top-level wirelist file with your changes.

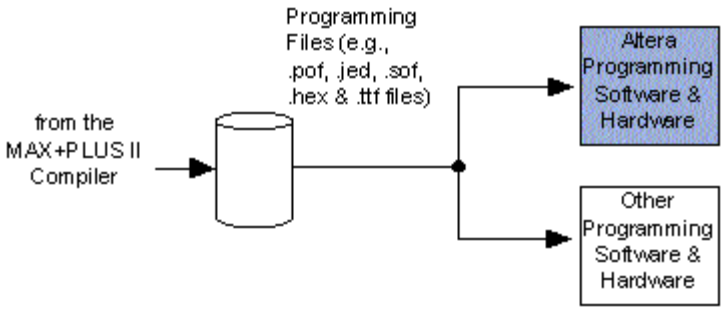
6. Continue with the steps necessary to perform timing simulation, as described in Performing a Timing Simulation with ViewSim Software.

Programming Altera Devices

Once you have successfully compiled and simulated a project with the MAX+PLUS® II software, you can program an Altera® device and test it in the target circuit. Figure 1 shows the device programming flow for MAX+PLUS II software.

Figure 1. MAX+PLUS II Device Programming Flow

Altera-provided items are shown in blue.



You can program devices with Altera programming hardware and MAX+PLUS II Programmer software installed on a 486- or Pentium-based PC or a UNIX workstation, or with programming hardware and software available from other manufacturers. Table 1 shows the available Altera programming hardware options on PCs and UNIX workstations.

Table 1. Altera Programming Hardware

Programming Hardware Option	PCs	UNIX Work-stations	MAX® 3000A Devices	Classic® & MAX 5000 Devices	MAX 7000 & MAX 7000E Devices	MAX 7000A, MAX 7000AE, MAX 7000B, MAX 7000S MAX 9000 & MAX 9000A Devices	FLEX® 6000, FLEX 6000A, FLEX 8000, FLEX 10K, FLEX 10KA, FLEX 10KB, & FLEX 10KE Devices	In-System Programming/ Configuration
Logic Programmer card, PL-MPU Master Programming Unit, and device-specific adapters	✓		✓	✓	✓	✓		
BitBlaster Download Cable	✓	✓	✓			✓	✓	✓
ByteBlasterMV Download Cable	✓		✓			✓	✓	✓
MasterBlaster Download Cable	✓	✓	✓			✓	✓	✓

If you wish to transfer programming files from a UNIX workstation to a PC over a network with File Transfer Protocol (FTP) or other similar transfer programs, be sure to select binary transfer mode.

Programming hardware from other manufacturers varies, but typically consists of a device connected to one of the serial ports on the workstation. Various vendors, such as Data I/O and BP Microsystems, supply hardware and software for programming Altera devices.

Related Topics:

- Go to Compiling Projects with MAX+PLUS II Software for information on creating programming files.
- Go to the following topics, which are available on the web, for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware
  - FLEX Devices
  - MAX Devices
  - Classic Device Family

Performing a Functional Simulation with ViewSim Software

You can use Viewlogic ViewSim software to perform a functional simulation of a ViewDraw schematic or a VHDL Design File (**.vhd**) before compiling your project with the MAX+PLUS II Compiler. Follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II /Viewlogic Powerview Working Environment.

2. Create a ViewDraw schematic that follows the guidelines in Creating ViewDraw Schematics for Use with MAX+PLUS II Software. Then go to step 3.

   or:

   Create a VHDL Design File <*design name*>**.vhd** and analyze it, as described in the following MAX+PLUS II ACCESS^{SM} Key topics:

   - Creating VHDL Designs for Use with MAX+PLUS II Software
   - Analyzing VHDL Files with the Vantage VHDL Analyzer Software

   Then go to step 7.

3. With the schematic open in the ViewDraw editor, add `CLR` and `PRE` inputs to any flipflops in your design, or tie the `CLR` and `PRE` ports of the flipflops to VCC. (Use the `PWR` primitive from the **builtin** library.)

4. Choose **Write To** (File menu) and save the schematic as <*design name*>**_funct**.

5. Start the **vsm** utility by double-clicking Button 1 on the **max2_vsmnet** icon in the Altera® Toolbox Design Tools Drawer.

6. Specify the following options in the **vsm** dialog box and choose **OK** to generate the <*design name*>**_funct.vsm** file:

Option:	**Setting:**
Design Name	<*design name*>*_funct*
Level	(blank)

7. Create a simulation command file (**.cmd**) for simulation with ViewSim software. Alternatively, you can enter commands at the prompt in the ViewSim window. Refer to your Viewlogic documentation for more information on creating ViewSim command files.

8. Start the ViewSim simulation tool by double-clicking Button 1 on the **max2_VSim** icon in the Design Tools Drawer.

9. If you wish to simulate a ViewDraw schematic, specify the following options in the **ViewSim** dialog box, then go to step 11.

Option:	**Setting:**
Design Name	<*design name*>*_funct*

Command File	*<design name>_funct.cmd*
VHDL Source Window	*OFF*
VHDL Debugging	*OFF*

10. If you wish to simulate a VHDL design, specify the following options in the **ViewSim** dialog box:

Option:	**Setting:**
Design Name	*<design name>*
Command File	*<design name>.cmd*
Graphical Interface	*ON*
VHDL Source Window	*OFF* or *ON*
VHDL Debugging	*OFF* or *ON*

11. Choose **OK** to simulate the design. ViewSim software simulates the design and starts the ViewTrace waveform editor to allow you to observe the simulation results.

12. Use the **edifneto** utility to generate an EDIF Netlist File (**.edf**) that can be imported into the MAX+PLUS II software, as described in Converting ViewDraw Schematics or VHDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the **edifneto** Utility.

Related Links:

- Go to ViewSim documentation for complete details on simulating a project and using ViewTrace to observe waveform output results.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Performing a Timing Simulation with ViewSim Software

After you have entered a design and compiled it with the MAX+PLUS® II Compiler, you can simulate a MAX+PLUS II-generated EDIF Output File (**.edo**) or VHDL Output File (**.vho**) with ViewSim software. ViewSim software can simulate both the functionality and the timing of your design. It also checks setup time, hold time, and Clock duty cycle timing requirements on registers.

To simulate a design with ViewSim software, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II /Viewlogic Powerview Working Environment.

2. Compile the design with the MAX+PLUS II software and generate an EDIF Output File (**.edo**) or VHDL Output File (**.vho**), as described in Compiling Projects with MAX+PLUS II Software.

3. In the Viewlogic Cockpit window, choose **Create** (Project menu) to open the **Create Project** dialog box. Type the name of your working directory and choose **OK**. You must create this new directory to avoid overwriting your original files when you generate new files for simulation.

4. Choose **SearchOrder** (Project menu) and add the appropriate directories and aliases to your **viewdraw.ini** file if you have not already done so. Go to Viewlogic Powerview **viewdraw.ini** Configuration File for more information.

   ☞ Refer to Viewlogic documentation for information on simulating projects that contain RAM functions. The procedure for reading an EDIF Output File and preparing it for simulation with ViewSim requires additional steps when the project contains RAM functions.

5. If you used the **SCH <-> max2** or **VHDL <-> max2** utility in the Max2 Express drawer to process your project, skip to step 8.

6. If you wish to simulate a VHDL Output File, follow the steps in Analyzing VHDL Files with the Vantage VHDL Analyzer then skip to step 7d.

7. If you are using the Altera® Toolbox Design Tools Drawer, follow these steps:

   1. To generate a Powerview wirelist from the EDIF Output File, double-click Button 1 on the **max2_edifi** icon in the Design Tools Drawer. The **Netlist In** dialog box is displayed.

   2. In the **Netlist In** dialog box, specify *../<design name>* for the *EDIF Netlist File* option, then choose **OK** to process the EDIF netlist file.

   3. If your project is implemented in multiple devices, repeat steps a and b for each EDIF Output File generated by the MAX+PLUS II Compiler, and ensure that the Altera-provided **alt_edif.cfg** file is specified for the *Attribute Swap Configuration File* option. In a multi-device project, the MAX+PLUS II Compiler generates a separate file for each device, plus a top-level file that is identified by "**_t**" appended to the project name. You must also follow the steps in Using ViewDraw & ViewGen Software to Prepare for Multi-Device Board-Level Simulation with ViewSim Software.

   4. Start the **vsm** utility by double-clicking Button 1 on the **max2_vsmnet** icon in the Design Tools

Drawer.

5. Specify your design name for the *Design Name* option in the **vsm** dialog box and choose **OK** to generate the <*design name*>**.vsm** file.

8. Create a simulation command file (**.cmd**) for simulation with ViewSim software. Alternatively, you can enter commands at the prompt in the ViewSim window. Refer to your Viewlogic documentation for more information on creating ViewSim command files.

   ☞ The Altera simulation model library, **max2_sim**, allows you to use the `alt_grst` signal to asynchronously clear all flipflops (`DFFE` primitives).

9. Start the ViewSim simulation tool by double-clicking Button 1 on the **max2_VSim** icon in the Design Tools Drawer or the Max2 Express Drawer.

10. Specify the following options in the **ViewSim** dialog box and choose **OK** to simulate the design:

Option:	**Setting:**
Design Name	*<design name>*
Command File	*<design name>.cmd*
VHDL Source Window	*OFF*
VHDL Debugging	*OFF*

    ViewSim software simulates the design and starts the ViewTrace waveform editor to allow you to observe the simulation results.

Related Links:

- Refer to the following sources for related information:
  - ViewSim documentation for complete details on simulating a project and using ViewTrace to observe waveform output results
  - Using ViewDraw & ViewGen Software to Prepare for Multi-Device Board-Level Simulation with ViewSim Software

---

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Using Viewlogic ViewSim & MAX+PLUS II Software

The following topics describe how to use the Viewlogic ViewSim software with MAX+PLUS® II software. Choose one of the following topics for information:

Open a printable version of all topics listed on this page.

Setting Up the MAX+PLUS II/Viewlogic Powerview Working Environment

- Software Requirements
- MAX+PLUS II/Viewlogic Powerview Interface File Organization
- Viewlogic Powerview **viewdraw.ini** Configuration File
- MAX+PLUS II/Viewlogic Powerview Project File Structure
- Altera-Provided Logic & Symbol Libraries
- The **vdpath** & **mega_lpm** Libraries

Functional Simulation

- Performing a Functional Simulation with ViewSim Software
  - Analyzing VHDL Files with the Vantage VHDL Analyzer Software

Timing Simulation

- Project Simulation Flow
- Initializing Registers in VHDL & Verilog Output Files for Power-Up before Simulation
- Performing a Timing Simulation with ViewSim Software
  - Analyzing VHDL Files with the Vantage VHDL Analyzer Software
  - Using ViewDraw & ViewGen Software to Prepare for Multi-Device Board-Level Simulation with ViewSim Software

Related Links

- Viewlogic Powerview Graphical User Interface & the Altera Toolbox
- Powerview Command-Line Syntax
- Compiling Projects with MAX+PLUS II Software
- Programming AlteraÂ® Devices
- MAX+PLUS II Development Software
- Altera Programming Hardware
- Viewlogic web site (http://www.viewlogic.com)

Using Viewlogic ViewSim & MAX+PLUS II Software

The following topics describe how to use the Viewlogic ViewSim software with MAX+PLUS® II software. Click on one of the following topics for information:

This file is suitable for printing only. It does not contain hypertext links that allow you to jump from topic to topic.

Setting Up the MAX+PLUS II/Viewlogic Powerview Working Environment

- Software Requirements
- MAX+PLUS II/Viewlogic Powerview Interface File Organization
- Viewlogic Powerview **viewdraw.ini** Configuration File
- MAX+PLUS II/Viewlogic Powerview Project File Structure
- Altera-Provided Logic & Symbol Libraries
- The **vdpath** & **mega_lpm** Libraries

Functional Simulation

- Performing a Functional Simulation with ViewSim Software
  - Analyzing VHDL Files with the Vantage VHDL Analyzer Software

Timing Simulation

- Project Simulation Flow
- Initializing Registers in VHDL & Verilog Output Files for Power-Up before Simulation
- Performing a Timing Simulation with ViewSim Software
  - Analyzing VHDL Files with the Vantage VHDL Analyzer Software
  - Using ViewDraw & ViewGen Software to Prepare for Multi-Device Board-Level Simulation with ViewSim Software

Related Topics:

- Go to the following MAX+PLUS II ACCESS℠ Key topics for related information:
  - Viewlogic Powerview Graphical User Interface & the Altera Toolbox
  - Powerview Command-Line Syntax
  - Compiling Projects with MAX+PLUS II Software
  - Programming Altera® Devices
- Go to the following topics, which are available on the web, for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware
  - Viewlogic web site (http://www.viewlogic.com)

Setting Up the MAX+PLUS II/Viewlogic Powerview Working Environment

To use the MAX+PLUS® II software with Viewlogic's Powerview software, you must install the MAX+PLUS II software, familiarize yourself with the Altera® Toolbox in the Powerview Cockpit, and then establish an environment that facilitates entering and processing designs. The MAX+PLUS II /Viewlogic Powerview interface is installed automatically when you install the MAX+PLUS II software on your workstation.

To set up your working environment for the MAX+PLUS II/Viewlogic Powerview interface, follow these steps:

1. Ensure that you have correctly installed the MAX+PLUS II and Viewlogic software versions described in MAX+PLUS II/Viewlogic Powerview Software Requirements.

2. Add the following environment variable to your **.cshrc** file to specify **/usr/maxplus2** as the MAX+PLUS II system directory:

   ```
   setenv ALT_HOME /usr/maxplus2 ↵
   ```

3. Add the **$ALT_HOME/viewlogic/standard**, **$ALT_HOME/bin**, and **$ALT_HOME/viewlogic/bin** directories to the `PATH` environment variable in your **.cshrc** file.

4. Add the **$ALT_HOME/viewlogic/standard** directory to the `WDIR` environment variable in your **.cshrc** file using the following syntax:

   ```
   setenv WDIR $ALT_HOME/viewlogic/standard:/<Powerview system directory>/standard ↵
   ```

   ☞ Make sure the **$ALT_HOME/viewlogic/standard** directory is the first directory in your `WDIR` path.

5. Source your **.cshrc** file by typing `source .cshrc ↵` at the UNIX prompt.

6. Create the Viewlogic Powerview **viewdraw.ini** configuration file.

7. Copy the **/usr/maxplus2/maxplus2.ini** file to your **$HOME** directory:

   ```
   cp /usr/maxplus2/maxplus2.ini $HOME ↵

   chmod u+w $HOME/maxplus2.ini ↵
   ```

   The **maxplus2.ini** file contains both Altera- and user-specified initialization parameters that control the MAX+PLUS II software, such as MAX+PLUS II symbol and logic function library paths and the current project name. The MAX+PLUS II installation procedure creates and copies the **maxplus2.ini** file to the **/usr/maxplus2** directory.

   ☞ Normally, you do not have to edit your local copy of **maxplus2.ini**, because the MAX+PLUS II software updates the file automatically whenever you change any parameters or settings. However, if you move the **max2lib** and **max2inc** library subdirectories, you must update the file. Go to "Creating & Using a Local Copy of the **maxplus2.ini** File" in MAX+PLUS II Help for more information.

8. If you plan to instantiate Library of Parameterized Modules (LPM) functions in ViewDraw schematics, you must create a new file with the name **vdraw.vs**. The **vdraw.vs** file must include the following line:

   ```
   load ("vdpath")
   ```

   You must also make sure that you specify the **vdraw.vs** file in your `WDIR` path.

9. Set up a directory structure that facilitates working with the MAX+PLUS II/Viewlogic Powerview interface.

Refer to MAX+PLUS II/Viewlogic Powerview Project File Structure.

Related Topics:

- Go to *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual for more information on installation and details on the directories that are created during MAX+PLUS II installation. Go to MAX+PLUS II/Viewlogic Powerview Interface File Organization for information about the MAX+PLUS II/Viewlogic Powerview directories that are created during MAX+PLUS II installation.
- Go to the following topics, which are available on the web, for additional information:
  - ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
  - This manual is also available in 4 parts:
    - Preface & Section 1: MAX+PLUS II Installation
    - Section 2: MAX+PLUS II - A Perspective
    - Section 3: MAX+PLUS II Tutorial
    - Appendices, Glossary & Index

MAX+PLUS II/Viewlogic Powerview Software Requirements

The following applications and utilities are used to generate, process, synthesize, and verify a project with MAX+PLUS® II and Viewlogic Powerview software.

	Viewlogic	**Altera**
ViewDraw	ViewGen	MAX+PLUS II version 9.4
VHDL Analyzer	ViewPath (optional)	
Vantage VHDL Analyzer	ViewTrace	
VHDL -> sym	ViewData Path	
edifneto	MOTIVE version 5.1.6 *Note (1)*	
edifneti	MOTIVE for Powerview version 3.2.1 (optional) *Note (1)*	
EEDIF (optional)	SDF2MTV (optional)	
MMP (optional)	Fusion/VCS	
vsm		

Note:

(1) MOTIVE for Powerview, a wrapper application for MOTIVE, provides a graphical user interface for the utilities (i.e., EEDIF, SDF2MTV, and MMP) used during a static timing verification with MOTIVE. MOTIVE alone does not accept EDIF files through the Setup Advisor.

☞ The MAX+PLUS II **read.me** file provides up-to-date information on which versions of Viewlogic Powerview applications the current version of the MAX+PLUS II software supports. It also provides information on installation and operating requirements. You should read the **read.me** file on the CD-ROM before installing the MAX+PLUS II software. After installation, you can open the **read.me** file from the MAX+PLUS II Help menu.

MAX+PLUS II/Viewlogic Powerview Interface File Organization

Table 1 shows the MAX+PLUS® II/Viewlogic Powerview interface subdirectories that are created in the MAX+PLUS II system directory (by default, the **/usr/maxplus2** directory) during MAX+PLUS II installation.

☞ For information on the other directories that are created during MAX+PLUS II installation, see "MAX+PLUS II File Organization" in *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual.

Table 1. MAX+PLUS II Directory Organization

Directory	Description
./lmf	Contains the Altera-provided Library Mapping File, **vwlogic.lmf**, that maps Viewlogic logic functions to equivalent MAX+PLUS II logic functions.
./viewlogic	Contains the **alt_edif.cfg** EDIF configuration file that is used with the **edifneti** utility. Also contains the library and sample subdirectories.
./viewlogic/examples	Contains the sample Viewlogic designs.
./viewlogic/library/max2sim	Contains the MAX+PLUS II simulation model library (**max2_sim**) for use in ViewSim software.
./viewlogic/library/alt_max2	Contains MAX+PLUS II primitives (`EXP`, `GLOBAL`, `LCELL`, `SOFT`, `CARRY`, `CASCADE`, `DFFE`, `DFFE6K`, and `OPNDRN`), macrofunctions (`a_8fadd`, `a_8mcomp`, `a_8count`, `a_81mux`), and megafunctions (`clklock`) for use in ViewDraw schematics. These logic functions support specific architectural features of Altera® devices. The **alt_max2** library also contains modified versions of the ViewDraw primitives that use tri-state buffers, because these primitives require special handling in the MAX+PLUS II /Viewlogic Powerview interface.
./viewlogic/library/synlib	Contains the Altera-provided synthesis library **altera**, which includes MAX+PLUS II primitives, the **altera.sml** file, a **sym** directory, and a **wir** directory for use with ViewSynthesis software.
./viewlogic/library/alt_mf	Contains the VHDL models for the MAX+PLUS II primitives (`EXP`, `GLOBAL`, `LCELL`, `SOFT`, `CARRY`, `CASCADE`, `DFFE`, and `OPNDRN`), macrofunctions (`clklock`) for use with ViewSynthesis software, the Vantage VHDL Analyzer software, and the VHDL source files. These logic functions are used to maintain portability to other architectures.
./viewlogic/library/alt_time	Contains MOTIVE timing models for MAX+PLUS II logic functions (**motive.lib**), including the `clklock` megafunction, and MAX+PLUS II driver models (**motive.drv**).
./viewlogic/library/alt_vtl	Contains the VHDL source files for the VITAL 3.0-compliant library. This library is available for ViewSim software.
./viewlogic/bin	Contains all MAX+PLUS II, Viewlogic, and interface-related scripts.
./viewlogic/standard	Contains all standard **.ini** files and standard tools.

Related Topics:

- Go to the following topics, which are available on the web, for additional information:
  - ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
  - This manual is also available in 4 parts:
    - Preface & Section 1: MAX+PLUS II Installation
    - Section 2: MAX+PLUS II - A Perspective
    - Section 3: MAX+PLUS II Tutorial
    - Appendices, Glossary & Index

Viewlogic Powerview viewdraw.ini Configuration File

Each Powerview project is configured with the **viewdraw.ini** file that resides in the project directory. The `DIR` statements at the end of **viewdraw.ini** are paths to library directories that are used by the various Powerview applications. Figure 1 shows a sample of the `DIR` statements that are required to use the libraries.

Figure 1. Excerpt from viewdraw.ini

```
DIR [pw] .
```

```
DIR [r]  /usr/maxplus2/vwlogic/library/alt_max2 (alt_max2)
DIR [r]  /usr/maxplus2/vwlogic/library/max2sim (max2_sim)
DIR [r]  /usr/maxplus2/vwlogic/library/synlib (altera)
DIR [r]  /usr/maxplus2/vwlogic/library/alt_mf (alt_mf)
DIR [r]  /usr/maxplus2/vwlogic/library/alt_vtl (alt_vtl)
DIR [rm] /<Powerview system directory>/lib/builtin (builtin)
DIR [rm] /<Powerview system directory>/simmods/vl/dip/74ls (vl74ls)
DIR [rm] /<Powerview system directory>/symsets/vl/dip/74ls (vl74ls)
DIR [r]  /<Powerview system directory>/lib/vdpath (vdpath)
```

When you add the libraries to the **/usr/maxplus2/vwlogic/standard/viewdraw.ini** file, they are automatically set when you create a new project. Powerview tools search these libraries sequentially, so it is important to add them in the order in which they are listed in Figure 1.

Table 1 shows the libraries that must be specified in the `DIR` statements in the **viewdraw.ini** file.

Table 1. Powerview Application Libraries

Library	Library Alias	Source	Topics
alt_max2	`alt_max2`	Altera	Graphical elements for ViewDraw
max2sim	`max2_sim`	Altera	Models for project simulation
synlib	`altera`	Altera	VHDL synthesis library for the MAX+PLUS ® II software
alt_mf	`alt_mf`	Altera	VHDL models of MAX+PLUS II logic functions
alt_vtl	`alt_vtl`	Altera	VITAL-compliant primitives
builtin	`builtin`	Altera	Basic primitives such as `INPUT` pins, `OUTPUT` pins, `AND` gates, `OR` gates, etc.
74ls	`vl74ls`	Viewlogic	74-series macrofunctions
vdpath	`vdpath`	Viewlogic	Standard library of parameterized modules (LPM) functions

The Altera-provided libraries must be listed before the Viewlogic-provided libraries in the **viewdraw.ini** file to ensure that the correct versions of the megafunctions, macrofunctions, and primitives are used.

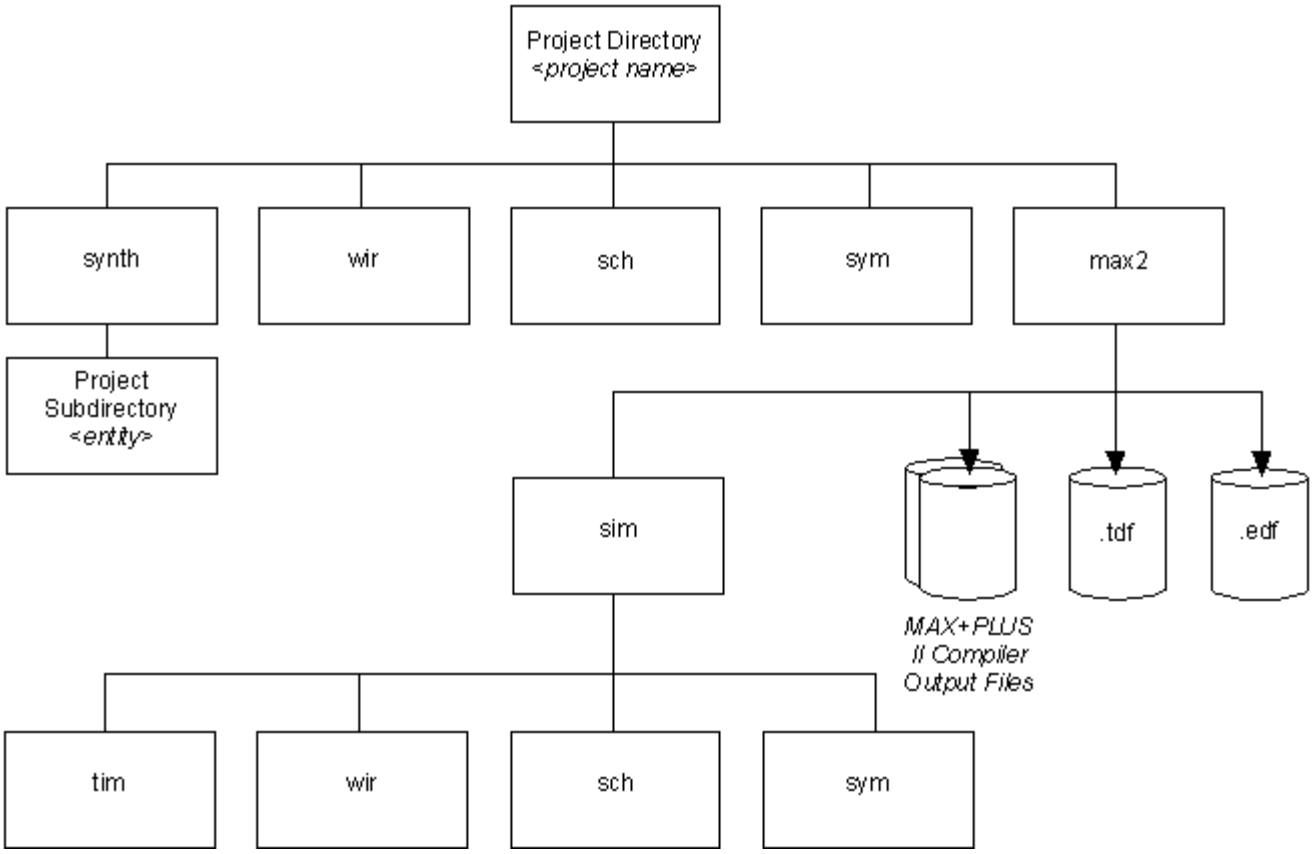
Related Topics:

- Go to Altera-Provided Logic & Symbol Libraries for more information on Altera-supplied libraries. Refer to the Powerview documentation for more information on setting up the **viewdraw.ini** file.

MAX+PLUS II/Viewlogic Powerview Project File Structure

In the MAX+PLUS® II software, a project name is the name of a top-level design file, without the filename extension. This design file can be an EDIF, Verilog HDL, or VHDL netlist file; an Altera® Hardware Description Language (AHDL) TDF; or any other MAX+PLUS II- supported design file. The EDIF netlist file must be created by Powerview and imported into the MAX+PLUS II software as an EDIF Input File (**.edf**). Figure 1 shows an example of MAX+PLUS II project directory structure that includes Powerview-generated files.

Figure 1. Sample MAX+PLUS II Project Organization



The MAX+PLUS II software stores the connectivity data on the links between design files in a hierarchical project in a Hierarchy Interconnect File (**.hif**), but refers to the entire project only by its project name. The MAX+PLUS II Compiler uses the HIF to build a single, fully flattened project database that integrates all the design files in a project hierarchy.

Unlike Powerview, the MAX+PLUS II software does not automatically create a project directory when you create a project. A single directory can contain several MAX+PLUS II design files, and you can specify any one of the designs in the directory as a project in the MAX+PLUS II software.

Viewlogic Powerview Local Work Area Structure

When you create a project with the Powerview Cockpit's **Create** command (Project menu), the project directory is created. You should generate design files and functional simulation files under this directory. A **max2** subdirectory is automatically created under your current project directory when you generate an EDIF file from your schematic or VHDL file. The ***<project name>*.edf** file is stored in the **max2** subdirectory. All MAX+PLUS® II Compiler output files are created in the **/*<project name>*/max2** subdirectory.

☞ ViewDraw files are identified by their directories and not by their extensions, so it is easy to overwrite files unintentionally. To avoid overwriting files, Altera recommends that you create a new project directory, ***<project name>*/max2/sim**, where you can generate all the files needed for simulation.

ViewDraw Project File Structure

Each ViewDraw project directory contains three subdirectories: **wir**, **sch**, and **sym**. See Table 1.

Table 1. ViewDraw Subdirectories

Directory	Topics

./wir	Wirelist files that contain connectivity information for a particular logic block
./sch	Schematics that contain logic
./sym	Symbol files that are the ViewDraw graphical representation of the logic blocks

Each file type uses the filename extension **.1**. Different file types are distinguished only by their directory: **/lib/wir/**<*project name*>**.1** is a wirelist file; **/lib/sch/**<*project name*>**.1** is the corresponding schematic file; and **/lib/sym/**<*project name*>**.1** is the corresponding symbol.

VHDL Project File Structure

Each VHDL project directory contains three subdirectories. See Table 2.

Table 2. VHDL Subdirectories

Directory	Topics
./synth	All synthesis-related files and directories
./synth/<*entity*>	Four types of files: <*entity*>**.pdf**, <*entity*>**.opt**, <*entity*>**.sta**, and <*entity*>**.gnl**
./wir	Wirelist for synthesized VHDL modules

☞ For each VHDL entity in the design, there is a corresponding **./synth/<*entity*>** directory.

Altera-Provided Logic & Symbol Libraries

The MAX+PLUS® II/Viewlogic Powerview environment provides libraries for compiling, synthesizing, and simulating designs.

☞ You can create your own libraries of custom symbols and logic functions for use in ViewDraw schematics and VHDL design files. You can use custom symbols (and functions) to incorporate an EDIF Input File, TDF, or any other MAX+PLUS II-supported design file into a project. The MAX+PLUS II software uses the **vwlogic.lmf** Library Mapping File to map ViewDraw symbols to equivalent MAX+PLUS II megafunctions, macrofunctions, or primitives. To use custom symbols and functions, you can create a custom LMF that maps your custom functions to equivalent EDIF Input Files, TDFs, or other MAX+PLUS II-supported design files. Go to "Library Mapping File" and "Viewlogic Library Mapping File" in MAX+PLUS II Help for more information.

Logic symbols used in ViewDraw software are available from the MAX+PLUS II **alt_max2** library, the ViewDraw **builtin** and **74ls** libraries, and the ViewDatapath **vdpath** library. VHDL models of MAX+PLUS II logic functions are available from the Altera-provided **alt_mf** library.

The alt_max2 Library

The **alt_max2** library provides MAX+PLUS II-specific logic functions that can be used to take advantage of special architectural features in each Altera® device family. See Table 1. Symbols and functional simulation models are available for all of these elements.

The alt_mf Library

The Altera-provided **alt_mf** library, which supports the Viewlogic Vantage VHDL Analyzer software, contains VHDL simulation models for all logic functions listed in the following table. The library is configured so that these functions pass untouched through the EDIF netlist file to the MAX+PLUS II Compiler, providing you with optimal control over design processing. Altera also provides models for all of the logic functions that you can synthesize and simulate. These models allow you to perform functional VHDL simulation while maintaining an architecture-independent VHDL description.

Table 1. Architecture Control Logic Functions

Name *Note (1), Note (2)*	Description	Name	Description	Name	Description
8fadd	8-bit full adder macrofunction	LCELL	Logic cell buffer primitive	EXP	MAX® 5000, MAX 7000, and MAX 9000 Expander buffer primitive
8mcomp	8-bit magnitude comparator macrofunction	GLOBAL	Global input buffer primitive	SOFT	Soft buffer primitive
8count	8-bit up/down counter macrofunction	CASCADE	FLEX® 6000, FLEX 8000, and FLEX 10K cascade buffer primitive	OPNDRN	Open-drain buffer primitive
81mux	8-to-1 multiplexer macrofunction	CARRY	FLEX 6000, FLEX 8000, and FLEX 10K cascade buffer primitive	DFFE *Note (2)*	D-type flipflop with Clock Enable primitive
clklock	Phase-locked loop megafunction				

Notes:

1. Logic function names that begin with a number must be prefixed with "a_" in VHDL designs. For example, 8fadd must be specified as a_8fadd.
2. For designs that are targeted to FLEX 6000 devices, you should use the DFFE primitive only if the design contains either a Clear or Preset signal, but not both. If your design contains both a Clear and a Preset signal, you must use the DFFE6K primitive.

Related Topics:

- Choose **Old-Style Macrofunctions**, **Primitives**, or **Megafunctions/LPM** from the MAX+PLUS II Help menu for detailed information on these functions.
- Go to the following topics, which are available on the web, for additional information:
  - FLEX Devices
  - MAX Devices
  - Classic Device Family

The vdpath & mega_lpm Libraries

The library of parameterized modules (LPM) 2.1.0 standard defines a set of parameterized functions and their corresponding representations in an EDIF netlist file. These logic functions allow you to create and functionally simulate an LPM-based design without targeting a specific device family. After the design is completed, you can target the design to any device family.

When the MAX+PLUS® II software processes projects that include Viewlogic-provided **vdpath** LPM functions, it uses functions from the Altera-provided **mega_lpm** library. This library includes all standard LPM functions except the truth table, finite state machine, and pad functions. Altera does not directly support the lpm_ram_dq, lpm_ram_io, and lpm_rom functions. Refer to Instantiating RAM & ROM Functions in Viewlogic Powerview Designs for instructions on instantiating RAM and ROM functions.

☞ Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu for more information about LPM functions.

Performing a Functional Simulation with ViewSim Software

You can use Viewlogic ViewSim software to perform a functional simulation of a ViewDraw schematic or a VHDL Design File (**.vhd**) before compiling your project with the MAX+PLUS II Compiler. Follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II /Viewlogic Powerview Working Environment.

2. Create a ViewDraw schematic that follows the guidelines in Creating ViewDraw Schematics for Use with MAX+PLUS II Software. Then go to step 3.

   or:

   Create a VHDL Design File <*design name*>**.vhd** and analyze it, as described in the following MAX+PLUS II ACCESS^{SM} Key topics:

   - Creating VHDL Designs for Use with MAX+PLUS II Software
   - Analyzing VHDL Files with the Vantage VHDL Analyzer Software

   Then go to step 7.

3. With the schematic open in the ViewDraw editor, add `CLR` and `PRE` inputs to any flipflops in your design, or tie the `CLR` and `PRE` ports of the flipflops to VCC. (Use the `PWR` primitive from the **builtin** library.)

4. Choose **Write To** (File menu) and save the schematic as <*design name*>**_funct**.

5. Start the **vsm** utility by double-clicking Button 1 on the **max2_vsmnet** icon in the Altera® Toolbox Design Tools Drawer.

6. Specify the following options in the **vsm** dialog box and choose **OK** to generate the <*design name*>**_funct.vsm** file:

Option:	**Setting:**
Design Name	*<design name>_funct*
Level	(blank)

7. Create a simulation command file (**.cmd**) for simulation with ViewSim software. Alternatively, you can enter commands at the prompt in the ViewSim window. Refer to your Viewlogic documentation for more information on creating ViewSim command files.

8. Start the ViewSim simulation tool by double-clicking Button 1 on the **max2_VSim** icon in the Design Tools Drawer.

9. If you wish to simulate a ViewDraw schematic, specify the following options in the **ViewSim** dialog box, then go to step 11.

Option:	**Setting:**
Design Name	*<design name>_funct*
Command File	*<design name>_funct.cmd*
VHDL Source Window	*OFF*
VHDL Debugging	*OFF*

10. If you wish to simulate a VHDL design, specify the following options in the **ViewSim** dialog box:

Option:	Setting:
Design Name	*<design name>*
Command File	*<design name>.cmd*
Graphical Interface	*ON*
VHDL Source Window	*OFF* or *ON*
VHDL Debugging	*OFF* or *ON*

11. Choose **OK** to simulate the design. ViewSim software simulates the design and starts the ViewTrace waveform editor to allow you to observe the simulation results.

12. Use the **edifneto** utility to generate an EDIF Netlist File (**.edf**) that can be imported into the MAX+PLUS II software, as described in Converting ViewDraw Schematics or VHDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the **edifneto** Utility.

Related Topics:

- Go to ViewSim documentation for complete details on simulating a project and using ViewTrace to observe waveform output results.

Analyzing VHDL Files with the SpeedWave VHDL Analyzer Software

You can use the SpeedWave VHDL Analyzer software to analyze VHDL Design Files (**.vhd**) prior to functional (or gate-level) simulation with ViewSim software, or to synthesis and optimization with ViewSynthesis software. You can also use the SpeedWave VHDL Analyzer to analyze a MAX+PLUS® II - generated VHDL Output File (**.vho**) prior to post-compilation timing simulation with ViewSim software. The **max2_VantgMgr** and **max2_VantgAnlz** tools are located in the Altera® Toolbox Design Tools Drawer.

To analyze a VHDL file with the SpeedWave VHDL Analyzer, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II /Viewlogic Powerview Working Environment.

2. If you wish to analyze a VHDL Design File (**.vhd**), create a VHDL file <*design name*>**.vhd** using the MAX+PLUS II Text Editor or another standard text editor and save it in a working directory. Go to Creating VHDL Designs for Use with MAX+PLUS II Software for more information.

3. If you wish to analyze a MAX+PLUS II-generated VHDL Output File (**.vho**), be sure to select *VHDL 1987* for the *VHDL Version* option and *VHDL Output File (.vho)* for the *Write Delay Constructs To* option in the **VHDL Netlist Writer Settings** dialog box (Interfaces menu) when you set up the MAX+PLUS II Compiler to generate a VHDL Output File. See Compiling Projects with MAX+PLUS II Software for more information on generating VHDL Output Files.

4. If your VHDL file contains functions from the **alt_mf** library, follow these steps:

    1. Start the Vantage Manager by double-clicking Button 1 on the **max2_VantgMgr** icon in the Design Tools Drawer.

    2. Use the Vantage VHDL Library Manager to create an **alt_mf.lib** library file with the symbolic name `ALT_MF`.

    3. Make **alt_mf** the working library with the **Set Working** command (Edit menu).

    4. Start the VHDL Analyzer by double-clicking Button 1 on the **max2_VantgAnlz** icon in the

Design Tools Drawer.

5. Analyze each VHDL file in the **alt_mf/src** directory into the **alt_mf.lib** working library. Source files are located in the **/usr/maxplus2/vwlogic/library/alt_mf/src** directory that is created by installing the Altera/Viewlogic interface.

5. If it is not already running, start the Vantage VHDL Library Manager, as described in step 4b, to create a Vantage library.

6. Choose the **List system libs** button.

7. Add the **ieee.lib** and **synopsys.lib** system libraries to your project:

    1. Select the **ieee.lib** and **synopsys.lib** libraries from the *Available Libraries* window and choose **Add lib**. Choose the **ieee** library from the **libs_syn** directory, which is located at ***/<Powerview system directory>/*** **standard/van_vss/pgm/libs_syn**. The **ieee** library contains Synopsys package files.

    2. If your project uses functions from the **alt_mf** library, also select the **alt_mf.lib** file from the *Available Libraries* window and choose **Add lib**.

    3. Choose **Create Library** (File menu, type the project directory name in the *Symbolic Name* field, and choose **OK**.

8. Specify the project directory as the working directory by choosing **Set Working** (Edit menu).

9. Choose **Save INI File** (File menu).

10. Choose **Dismiss Window** (Powerview Red-Box menu).

11. Specify the appropriate path and file name in the **Analyzer VHDL Source File** dialog box and choose **OK** to analyze the VHDL file.

12. Once you have analyzed the file, perform one or more of the following tasks, as appropriate:

    - Performing a Functional Simulation with ViewSim Software
    - Synthesizing & Optimizing VHDL Designs with ViewSynthesis Software
    - Performing a Timing Simulation with ViewSim Software

Related Topics:

- Refer to the following sources for related information:
    - The ***Viewlogic ViewSim/VHDL User's Guide*** and ***ViewSim/VHDL Tutorial*** for information on using the Vantage VHDL Analyzer software or Vantage VHDL Library Manager
    - Powerview Command-Line Syntax in these MAX+PLUS II ACCESS^{SM} Key topics

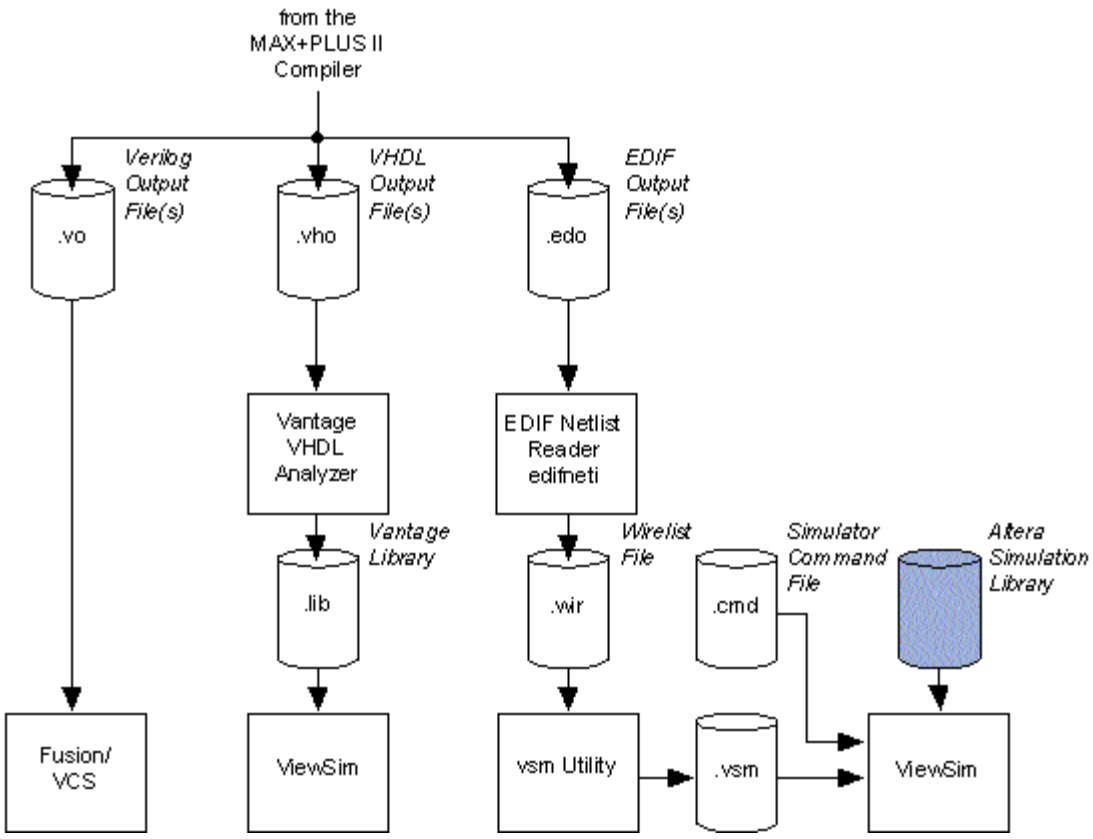
---

MAX+PLUS II/Viewlogic Powerview Simulation Flow

Figure 1 shows the project simulation flow for the MAX+PLUS$^{®}$ II/Viewlogic Powerview interface.

Figure 1. MAX+PLUS II/Viewlogic Powerview Project Simulation Flow

Altera-provided items are shown in blue.



Performing a Timing Simulation with ViewSim Software

After you have entered a design and compiled it with the MAX+PLUS® II Compiler, you can simulate a MAX+PLUS II-generated EDIF Output File (**.edo**) or VHDL Output File (**.vho**) with ViewSim software. ViewSim software can simulate both the functionality and the timing of your design. It also checks setup time, hold time, and Clock duty cycle timing requirements on registers.

To simulate a design with ViewSim software, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II /Viewlogic Powerview Working Environment.

2. Compile the design with the MAX+PLUS II software and generate an EDIF Output File (**.edo**) or VHDL Output File (**.vho**), as described in Compiling Projects with MAX+PLUS II Software.

3. In the Viewlogic Cockpit window, choose **Create** (Project menu) to open the **Create Project** dialog box. Type the name of your working directory and choose **OK**. You must create this new directory to avoid overwriting your original files when you generate new files for simulation.

4. Choose **SearchOrder** (Project menu) and add the appropriate directories and aliases to your **viewdraw.ini** file if you have not already done so. Go to Viewlogic Powerview **viewdraw.ini** Configuration File for more information.

   ☞ Refer to Viewlogic documentation for information on simulating projects that contain RAM functions. The procedure for reading an EDIF Output File and preparing it for simulation with ViewSim requires additional steps when the project contains RAM functions.

5. If you used the **SCH <-> max2** or **VHDL <-> max2** utility in the Max2 Express drawer to process

your project, skip to step 8.

6. If you wish to simulate a VHDL Output File, follow the steps in Analyzing VHDL Files with the Vantage VHDL Analyzer then skip to step 7d.

7. If you are using the Altera® Toolbox Design Tools Drawer, follow these steps:

    1. To generate a Powerview wirelist from the EDIF Output File, double-click Button 1 on the **max2_edifi** icon in the Design Tools Drawer. The **Netlist In** dialog box is displayed.

    2. In the **Netlist In** dialog box, specify *../<design name>* for the *EDIF Netlist File* option, then choose **OK** to process the EDIF netlist file.

    3. If your project is implemented in multiple devices, repeat steps a and b for each EDIF Output File generated by the MAX+PLUS II Compiler, and ensure that the Altera-provided **alt_edif.cfg** file is specified for the *Attribute Swap Configuration File* option. In a multi-device project, the MAX+PLUS II Compiler generates a separate file for each device, plus a top-level file that is identified by "**_t**" appended to the project name. You must also follow the steps in Using ViewDraw & ViewGen Software to Prepare for Multi-Device Board-Level Simulation with ViewSim Software.

    4. Start the **vsm** utility by double-clicking Button 1 on the **max2_vsmnet** icon in the Design Tools Drawer.

    5. Specify your design name for the *Design Name* option in the **vsm** dialog box and choose **OK** to generate the *<design name>***.vsm** file.

8. Create a simulation command file (**.cmd**) for simulation with ViewSim software. Alternatively, you can enter commands at the prompt in the ViewSim window. Refer to your Viewlogic documentation for more information on creating ViewSim command files.

    The Altera simulation model library, **max2_sim**, allows you to use the `alt_grst` signal to asynchronously clear all flipflops (`DFFE` primitives).

9. Start the ViewSim simulation tool by double-clicking Button 1 on the **max2_VSim** icon in the Design Tools Drawer or the Max2 Express Drawer.

10. Specify the following options in the **ViewSim** dialog box and choose **OK** to simulate the design:

Option:	**Setting:**
Design Name	*<design name>*
Command File	*<design name>.cmd*
VHDL Source Window	*OFF*
VHDL Debugging	*OFF*

    ViewSim software simulates the design and starts the ViewTrace waveform editor to allow you to observe the simulation results.

Related Topics:

- Refer to the following sources for related information:
    - ViewSim documentation for complete details on simulating a project and using ViewTrace to observe waveform output results
    - Using ViewDraw & ViewGen Software to Prepare for Multi-Device Board-Level Simulation with ViewSim Software

Initializing Registers in VHDL & Verilog Output Files for Power-Up before Simulation

Altera provides the **add_dc** script, which is availiable in the MAX+PLUS II system directory, to allow you to process MAX+PLUS II-generated Verilog Output Files (**.vo**) and VHDL Output Files (**.vho**) to prepare these files for simulation with another EDA tool. The add_dc script runs the **add_dclr** utility, which inserts a `device_clear` signal that is used for power-up initialization of all registers or flipflops in the design.

The script adds in a top-level signal named `device_clear` and connects it to the `CLRN` pin in all flipflops that should initialize to 0, and to the `PRN` pin of all flipflops that should initialize to 1. If the `CLRN` or `PRN` pin of a flipflop is already being used (i.e., is already connected to a signal), the script modifies the Verilog Output File or VHDL Output File so that the `AND` of the original signal and the `device_clear` pin feed the `CLRN` or `PRN` pin.

To use the **add_dc** script to process Verilog Output Files and VHDL Output Files before simulation with another EDA tool, follow these steps:

1. Make sure that your design file is located in the current directory, or change to the directory in which the design file is located.
2. Type the following command at the command prompt:

   `Â¥`*<path name of **add_dc.bat** file>*`Â¥add_dc` *<design name>* *<path name of **add_dclr.exe** file>* ↵

For example, if the both the **add_dc.bat** and the **add_dclr.exe** files are located in the **d:Â¥maxplus2Â¥exew** directory, and the **d:Â¥maxplus2Â¥exew** directory is specified in the search path, you can type the following command at a command prompt to add a `device_clear` signal to a design named `myfifo` in the file **myfifo.vo**:

```
add_dc myfifo d:Â¥maxplus2Â¥exew ↵
```

1. The **add_dc** script gives a message if the directory contains both a VHDL Output File and a Verilog Output File with the same name (*<design name>***.vo** and *<design>***.vho**). You should delete or rename whichever of those files should not have the `device_clear` signal added. The **add_dc** script can modify only one design file at a time.

2. When the **add_dc** script processes the Verilog Output File or VHDL Output File, it creates a backup copy of the original file, with the extension **.ori**.
3. The **add_dc** script works only for Verilog Output Files and VHDL Output Files that are generated by MAX+PLUS II.

After you have used the **add_dc** script and are ready to simulate the resulting Verilog Output File or VHDL Output File with another EDA tool, you should assert the active low `device_clear` pin for a period of time that is long enough for the design to initialize. You can then de-assert the pin, and apply simulation vectors to the design.

Analyzing VHDL Files with the SpeedWave VHDL Analyzer Software

You can use the SpeedWave VHDL Analyzer software to analyze VHDL Design Files (**.vhd**) prior to functional (or gate-level) simulation with ViewSim software, or to synthesis and optimization with

ViewSynthesis software. You can also use the SpeedWave VHDL Analyzer to analyze a MAX+PLUS® II - generated VHDL Output File (**.vho**) prior to post-compilation timing simulation with ViewSim software. The **max2_VantgMgr** and **max2_VantgAnlz** tools are located in the Altera® Toolbox Design Tools Drawer.

To analyze a VHDL file with the SpeedWave VHDL Analyzer, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II /Viewlogic Powerview Working Environment.

2. If you wish to analyze a VHDL Design File (**.vhd**), create a VHDL file <*design name*>**.vhd** using the MAX+PLUS II Text Editor or another standard text editor and save it in a working directory. Go to Creating VHDL Designs for Use with MAX+PLUS II Software for more information.

3. If you wish to analyze a MAX+PLUS II-generated VHDL Output File (**.vho**), be sure to select *VHDL 1987* for the *VHDL Version* option and *VHDL Output File (.vho)* for the *Write Delay Constructs To* option in the **VHDL Netlist Writer Settings** dialog box (Interfaces menu) when you set up the MAX+PLUS II Compiler to generate a VHDL Output File. See Compiling Projects with MAX+PLUS II Software for more information on generating VHDL Output Files.

4. If your VHDL file contains functions from the **alt_mf** library, follow these steps:

    1. Start the Vantage Manager by double-clicking Button 1 on the **max2_VantgMgr** icon in the Design Tools Drawer.

    2. Use the Vantage VHDL Library Manager to create an **alt_mf.lib** library file with the symbolic name `ALT_MF`.

    3. Make **alt_mf** the working library with the **Set Working** command (Edit menu).

    4. Start the VHDL Analyzer by double-clicking Button 1 on the **max2_VantgAnlz** icon in the Design Tools Drawer.

    5. Analyze each VHDL file in the **alt_mf/src** directory into the **alt_mf.lib** working library. Source files are located in the **/usr/maxplus2/vwlogic/library/alt_mf/src** directory that is created by installing the Altera/Viewlogic interface.

5. If it is not already running, start the Vantage VHDL Library Manager, as described in step 4b, to create a Vantage library.

6. Choose the **List system libs** button.

7. Add the **ieee.lib** and **synopsys.lib** system libraries to your project:

    1. Select the **ieee.lib** and **synopsys.lib** libraries from the *Available Libraries* window and choose **Add lib**. Choose the **ieee** library from the **libs_syn** directory, which is located at **/<*Powerview system directory*>/ standard/van_vss/pgm/libs_syn**. The **ieee** library contains Synopsys package files.

    2. If your project uses functions from the **alt_mf** library, also select the **alt_mf.lib** file from the *Available Libraries* window and choose **Add lib**.

    3. Choose **Create Library** (File menu, type the project directory name in the *Symbolic Name* field, and choose **OK**.

8. Specify the project directory as the working directory by choosing **Set Working** (Edit menu).

9. Choose **Save INI File** (File menu).

10. Choose **Dismiss Window** (Powerview Red-Box menu).

11. Specify the appropriate path and file name in the **Analyzer VHDL Source File** dialog box and choose **OK** to analyze the VHDL file.

12. Once you have analyzed the file, perform one or more of the following tasks, as appropriate:

    - Performing a Functional Simulation with ViewSim Software
    - Synthesizing & Optimizing VHDL Designs with ViewSynthesis Software
    - Performing a Timing Simulation with ViewSim Software

Related Topics:

- Refer to the following sources for related information:
  - The ***Viewlogic ViewSim/VHDL User's Guide*** and ***ViewSim/VHDL Tutorial*** for information on using the Vantage VHDL Analyzer software or Vantage VHDL Library Manager
  - Powerview Command-Line Syntax in these MAX+PLUS II ACCESS^{SM} Key topics

---

Using ViewDraw & ViewGen Software to Prepare for Multi-Device Board-Level Simulation with ViewSim Software

In order to perform board-level simulation with ViewSim software, you must generate symbols that represent each MAX+PLUS® II -generated EDIF Output File (**.edo**) and incorporate them into a top-level ViewDraw schematic. You can use ViewGen to generate hollow-body symbols to represent each EDIF Output File, and connect them to other system components in the top-level schematic. You must also edit the wirelist files (**.wir**) created by the **edifneti** utility.

To prepare for multi-device board-level simulation with ViewSim software, follow these steps:

1. Perform steps 1 through 6c in Performing a Timing Simulation with ViewSim Software.

2. Start ViewGen by double-clicking Button 1 on the **max2_VGen** icon in the Design Tools Drawer.

3. Specify the filename of one of the EDIF Output Files <*filename*>.**edf** in the *Name* box in the **ViewGen** dialog box and choose **OK** to generate a corresponding <*filename*> symbol.

4. Repeat step 3 to generate other symbols as needed. You do not need to generate a symbol for the <*filename*>**_t.edf** file.

5. Eliminate the two extra pins for `VDD` and `GND` connections from the top-level wirelist file **./wir/**<*design name*>**_t.1**:

    1. Open the **./wir/**<*design name*>**_t.1** wirelist file with a standard text editor and delete the following lines:

       ```
       P IN GND
       I GND IN GND
       P IN VDD
       I VDD IN VDD
       ```

    2. Add the following two lines to the file to ensure global ground and power connections for simulation:

       ```
       G VDD↵
       G GND↵
       ```

3. Save the top-level wirelist file with your changes.

6. Continue with the steps necessary to perform timing simulation, as described in Performing a Timing Simulation with ViewSim Software.

Viewlogic Powerview Graphical User Interface & the Altera Toolbox

You use the Powerview graphical interface manager, the Cockpit, and the Altera® Toolbox to start all Powerview and Altera tools. Within the Altera Toolbox, you can specify the Max2 Express Drawer or the Design Tools Drawer to work with the Altera/Viewlogic Powerview interface.

The Max2 Express Drawer provides a quick and seamless way to transfer designs created in Powerview to the MAX+PLUS® II software for compilation, then return the compiled designs to Powerview for simulation and timing verification. Table 1 describes the Max2 Express Drawer tools.

Table 1. Max2 Express Drawer Tools

Tool	Description
max2_VDraw	Launches the Powerview ViewDraw schematic entry tool.
VHDL<->max2	Launches all tools necessary to synthesize a VHDL design, compile for an Altera device, and generate a **.vsm** file for simulation with the Powerview ViewSim simulator.
SCH<->max2	Launches all tools necessary to compile a schematic design entered with Powerview ViewDraw software for an Altera device and to generate a **.vsm** file for simulation with Powerview ViewSim and **.edo**, **.sdo**, and **.vmo** files for timing analysis with MOTIVE for Powerview.
max2_VSim	Launches the Powerview ViewSim simulator.
max2_VTrace	Launches the Powerview ViewTrace simulation waveform editor.
max2_MOTIVE	Launches the MOTIVE for Powerview ViewDraw static timing verification tool.

The Design Tools Drawer provides tools that enable you to create a design with the Powerview tools, compile the design in the MAX+PLUS II software, and simulate and verify the design with Powerview software. Table 2 describes the Design Tools Drawer tools.

Table 2. Design Tools Drawer Tools

Tool	Description
max2_VDraw	Launches the Powerview ViewDraw schematic entry tool.
max2_analyzer	Launches the Powerview VHDL Analyzer software.
max2_syn	Launches the Powerview VHDL synthesis tool.
max2_chk	Launches the Powerview schematic verification tool.
max2_vsmnet	Launches the Powerview **vsm** utility that converts a wirelist file into a **.vsm** file.
max2_VSim	Launches the Powerview ViewSim simulator.
max2_VTrace	Launches the Powerview ViewTrace simulator.
max2_edifo	Launches the Powerview EDIF netlist writer, **edifneto**.
max2_VGen	Launches the Powerview ViewGen utility that generates a schematic from a wirelist file.
max2	Launches the MAX+PLUS II Compiler.
max2_edifi	Launches the Powerview EDIF Netlist Reader, **edifneti**.
max2_vhdl2sym	Launches the Powerview **vhdl2sym** utility that generates a symbol from a VHDL file.

max2_VantgMgr	Launches the Powerview Vantage VHDL Library Manager tool.
max2_VantgAnlz	Launches the Vantage VHDL Analyzer software.
max2_VCS	Launches the Fusion/VCS Simulator.
max2_MOTIVE	Launches the MOTIVE for Powerview static timing verification tool.

Powerview Command-Line Syntax

Table 1 shows the command-line syntax for using Powerview functions.

Table 1. Powerview Command-Line Syntax

Action	Command
Start VHDL Analyzer software	`vhdl -v` <*project name*>
Start ViewSynthesis software	`vhdldes`
Load Altera® technology library	`vhdldes> technology altera`
Compile a VHDL design	`vhdldes> vhdl` <*project name*>
Synthesize a design	`vhdldes> synthesize`
Generate wirelist file	`vhdldes> wir`
Create a schematic representation	`vhdldes> viewgen`
Generate a synthesis report file	`vhdldes> report`
Start the graphical user interface for ViewSynthesis	`vhdldes> vdesgui`
Start the VHDL-to-symbol utility	`vhdl2sym` <*project name*>
Start **vsm**	`vsm` <*project name*>
Start ViewSim simulator	`viewsim` <*project name*> -<*project name*>`.cmd`
Start **edifneto**	`edifneto -f` <*project name*>-`l` (`std` *or* `altera`) <*project name*>`.edf`
Start Vantage VHDL Analyzer software	`analyze -src` <*design file*>
Start MOTIVE for Powerview software	`mfp`

Compiling Projects with MAX+PLUS II Software

The MAX+PLUS® II Compiler can process design files in a variety of formats. This topic describes how to use MAX+PLUS II software to compile projects in which the top-level design file is an EDIF Input File (with the extension **.edf**).

☞ Refer to the following sources for additional information:

- Go to MAX+PLUS II Help for information on compiling VHDL and Verilog HDL, design files directly with the MAX+PLUS II Compiler.
- Go to Running Synopsys Compilers from MAX+PLUS II Software for information on running the Synopsys Design Compiler or FPGA Compiler software on a VHDL or Verilog HDL design from within the MAX+PLUS II Compiler window.

To compile a design (also called a "project") with MAX+PLUS II software, go through the following steps:

1. Create design files that are compatible with the MAX+PLUS II software and convert them into EDIF Input Files with the extension **.edf**. Specific instructions for some tools are described in these MAX+PLUS II ACCESS^{SM} Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your design entry or synthesis and optimization tool.

2. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, create a mapping for each function in a Library Mapping File (**.lmf**) to map the custom symbol to the corresponding EDIF Input File, AHDL Text Design File (**.tdf**), or other MAX+PLUS II-supported design file. These custom functions are represented in design files as hollow-body symbols or "black box" HDL descriptions.

   ☞ Go to "Library Mapping Files (**.lmf**)" in MAX+PLUS II Help for more information.

3. Open MAX+PLUS II and specify the name of your top-level design file as the project name with the **Project Name** command (File menu). If you open an HDL file in the MAX+PLUS II Text Editor, you can choose the **Project Set Project to Current File** command (File menu) instead.

   ☞ You can also compile a project from a command line. However, the first time you compile a project, the settings you need to specify are easier to specify from within the MAX+PLUS II software. After you have run the graphical user interface for the MAX+PLUS II software at least once, you can more easily use the command-line **setacf** utility to modify options in the Assignment & Configuration File (**.acf**) for the project. Type `setacf -h` ↵ and `maxplus2 -h` ↵ for descriptions of **setacf** and MAX+PLUS II command-line syntax.

4. Choose **Device** (Assign menu) and select the target Altera device family in the *Device Family* drop-down list box. If you wish to implement the design logic in a specific device, select it in the *Devices* box. Otherwise, select *AUTO* to allow the MAX+PLUS II Compiler to choose the best device(s) in the current device family. If your design entry or synthesis and optimization tool required you to specify a target family and/or device, specify the same information in this dialog box. For information on partitioning logic among multiple devices, go to MAX+PLUS II Help. Choose **OK**.

5. Open the Compiler window by choosing the **Compiler** command (MAX+PLUS II menu). Go through the following steps to specify the options necessary to compile the design file(s) in your project:

   1. Ensure that all EDIF netlist files have the extension **.edf** and choose **EDIF Netlist Reader Settings** (Interfaces menu).

   2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor. This name should be the name of the vendor whose tool(s) you used to create the EDIF netlist files. If your vendor name does not appear, select *Custom* instead.

      ☞ If you are compiling a design created with Synopsys FPGA Express software, select *Synopsys*, choose the **Customize** button, enter <*project name*>**.lmf** in the *LMF #1* box, choose **OK**, and skip to step 6.

   3. If you selected an existing vendor name in the *Vendor* box and your project contains design files that require custom LMF mappings, choose the **Customize** button to expand the dialog box to show all settings. Turn on the *LMF #2* checkbox and type your custom LMF's filename in the corresponding text box, or select a name from the *Files* box. The selection in the *Vendor* box will change to *Custom* and all settings will be retained until you change them again.

   4. If you selected *Custom* in the *Vendor* box, choose the **Customize** button to expand the dialog box to show all settings. Any previously defined custom settings will be displayed. Under *Signal Names*, type one or more names with up to 20 total name characters in the *VCC* or *GND* box if your EDIF Input File(s) use one or more names other than `VCC` or `GND` for the global high or low signals. Multiple signal names must be separated by either a comma (,) or a space. Under

Library Mapping Files, turn on the *LMF #1* checkbox and type a filename in the text box following it, or select a name from the *Files* box. If necessary, specify another LMF name in the *LMF #2* box. Go to MAX+PLUS II Help for detailed information on the settings available in the **EDIF Netlist Reader Settings** dialog box.

5. Choose **OK**.

6. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, you may need to ensure that all files are present in your project directory, i.e., the same directory as the top-level design file. Otherwise, you must specify the directories containing these files as user libraries with the **User Libraries** command (Options menu).

7. Follow all guidelines that apply to your design entry or synthesis and optimization tool:

   - Exemplar Logic Galileo Extreme-Specific Compiler Settings
   - Synopsys DesignWare-Specific Compiler Settings
   - Converting Synopsys FPGA Compiler & Design Compiler Timing Constraints into MAX+PLUS II-Compatible Format with the **syn2acf** Utility
   - Synplicity Synplify-Specific Compiler Settings

8. If you wish to generate EDIF, VHDL, or Verilog HDL output files for post-compilation simulation or timing analysis with another EDA tool, go through the following steps:

   1. (Optional) Turn on the **Optimize Timing SNF** command (Processing menu) to reduce the size of the output file(s). Turning on this command can reduce the size of output netlists by up to 30%.

      This command does not create optimized timing SNFs on UNIX workstations. However, a non-optimized timing SNF provides the same functional and timing information as an optimized timing SNF.

   2. If you wish to generate EDIF Output Files (**.edo**), go through these steps:

      1. Turn on the **EDIF Netlist Writer** command (Interfaces menu). Then choose the **EDIF Netlist Writer Settings** command (Interfaces menu).

      2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor and choose **OK**. If your vendor name does not appear, select *Custom* instead and specify the settings that are appropriate for your simulation or timing analysis tool. Go to MAX+PLUS II Help for detailed information on the options available in the **EDIF Netlist Writer Settings** dialog box.

      3. To generate an optional Standard Delay Format (SDF) Output File (**.sdo**), choose the **Customize** button to expand the dialog box to show all settings. Select one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**.

      The filenames of the EDIF Output File(s) and optional SDF Output File(s) are the same as the user-defined chip name(s) for the project; if no chip names exist, the Compiler assigns filenames that are based on the project name. For a multi-device project, the Compiler also generates a top-level EDIF Output File that is uniquely identified by "**_t**" appended to the project name. In addition, the Compiler automatically generates a VHDL Memory Model Output File, *<project name>***.vmo**, when it generates an EDIF Output File that contains memory (RAM or ROM).

   3. If you wish to generate VHDL Output Files (**.vho**), turn on the **VHDL Netlist Writer** command (Interfaces menu). Then choose **VHDL Netlist Writer Settings** command (Interfaces menu). Select *VHDL Output File (.vho)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF ver. 2.1 files contain timing delay information that allows

you to perform back-annotation simulation in VHDL with VITAL-compliant simulation libraries. The VHDL Output Files generated by the Compiler have the extension **.vho**, but are otherwise named in the same way as the EDIF Output Files described above.

4. If you wish to generate Verilog HDL Output Files (**.vo**), turn on the **Verilog Netlist Writer** command (Interfaces menu). Then choose **Verilog Netlist Writer Settings** command (Interfaces menu). Select *Verilog Output File (.vo)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF Output Files contain timing delay information that allows you to perform back-annotation simulation in Verilog HDL. The Verilog Output Files generated by the Compiler have the extension **.vo**, but are otherwise named in the same way as the EDIF Output Files described above.

9. To run the MAX+PLUS II Compiler, choose the **Project Save & Compile** command (File menu) or choose the **Start** button in the Compiler window.

   ☞ See step 3 for information on running MAX+PLUS II software from the command line.

10. Once you have compiled the project with the MAX+PLUS II Compiler, you can use the VHDL, Verilog HDL, or EDIF output file(s), and the optional SDF Output File(s) (**.sdo**) to perform timing analysis or timing simulation with another EDA tool. Specific instructions for some tools are described in these MAX+PLUS II ACCESS Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your EDA tool.

The MAX+PLUS II Compiler also generates a Report File (**.rpt**), a Pin-Out File (**.pin**), and one or more of the following files for device programming or configuration:

- JEDEC Files (**.jed**)
- Programmer Object Files (**.pof**)
- SRAM Object Files (**.sof**)
- Hexadecimal (Intel-format) Files (**.hex**)
- Tabular Text Files (**.ttf**)

Related Topics:

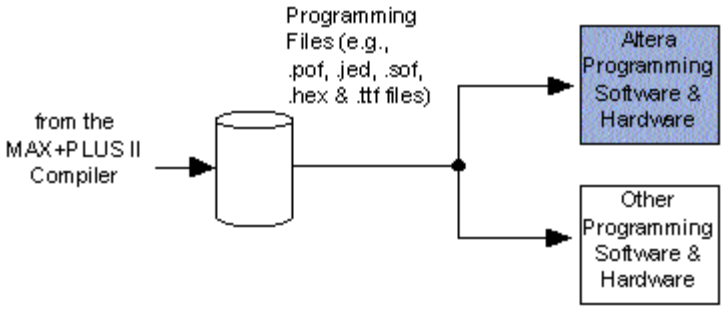
- Refer to the following sources for additional information:
  - Go to *Compiler Procedures* in MAX+PLUS II Help for information on other available Compiler settings.
  - Go to *Programmer Procedures* in MAX+PLUS II Help for instructions on creating other types of programming files and on programming or configuring Altera devices.
  - Go to Back-Annotating MAX+PLUS II Pin Assignments to Design Architect Symbols for information on back-annotating pin assignments in Mentor Graphics Design Architect schematics.
  - Go to Programming Altera Devices for information on the different programming hardware options for Altera device families.
- Go to the following topics, which are available on the web, for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware

Programming Altera Devices

Once you have successfully compiled and simulated a project with the MAX+PLUS® II software, you can program an Altera® device and test it in the target circuit. Figure 1 shows the device programming flow for MAX+PLUS II software.

Figure 1. MAX+PLUS II Device Programming Flow

Altera-provided items are shown in blue.



You can program devices with Altera programming hardware and MAX+PLUS II Programmer software installed on a 486- or Pentium-based PC or a UNIX workstation, or with programming hardware and software available from other manufacturers. Table 1 shows the available Altera programming hardware options on PCs and UNIX workstations.

Table 1. Altera Programming Hardware

Programming Hardware Option	PCs	UNIX Work-stations	MAX® 3000A Devices	Classic® & MAX 5000 Devices	MAX 7000 & MAX 7000E Devices	MAX 7000A, MAX 7000AE, MAX 7000B, MAX 7000S MAX 9000 & MAX 9000A Devices	FLEX® 6000, FLEX 6000A, FLEX 8000, FLEX 10K, FLEX 10KA, FLEX 10KB, & FLEX 10KE Devices	In-System Programming/ Configuration
Logic Programmer card, PL-MPU Master Programming Unit, and device-specific adapters	✓		✓	✓	✓	✓		
BitBlaster Download Cable	✓	✓	✓			✓	✓	✓
ByteBlasterMV Download Cable	✓		✓			✓	✓	✓
MasterBlaster Download Cable	✓	✓	✓			✓	✓	✓

If you wish to transfer programming files from a UNIX workstation to a PC over a network with File Transfer Protocol (FTP) or other similar transfer programs, be sure to select binary transfer mode.

Programming hardware from other manufacturers varies, but typically consists of a device connected to one of the serial ports on the workstation. Various vendors, such as Data I/O and BP Microsystems, supply hardware and software for programming Altera devices.

Related Topics:

- Go to Compiling Projects with MAX+PLUS II Software for information on creating programming files.
- Go to the following topics, which are available on the web, for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware
  - FLEX Devices
  - MAX Devices
  - Classic Device Family

Using Viewlogic ViewSynthesis & MAX+PLUS® II Software

The following topics describe how to use the Viewlogic ViewSynthesis software with MAX+PLUS® II software. Choose one of the following topics for information:

Open a printable version of all topics listed on this page.

Setting Up the MAX+PLUS II/Viewlogic Powerview Working Environment

- Software Requirements
- MAX+PLUS II/Viewlogic Powerview Interface File Organization
- Viewlogic Powerview **viewdraw.ini** Configuration File
- MAX+PLUS II/Viewlogic Powerview Project File Structure
- Altera-Provided Logic & Symbol Libraries
- The **vdpath** & **mega_lpm** Libraries

VHDL Design Entry

- Design Entry Flow
- Creating VHDL Designs for Use with MAX+PLUS II Software
  - Instantiating the clklock Megafunction in VHDL or Verilog HDL
  - Instantiating RAM & ROM Functions in Viewlogic Powerview Designs
- Entering Resource Assignments
  - Modifying the Assignment & Configuration File with the **setacf** Utility
- Performing a Functional Simulation with ViewSim Software
- Analyzing VHDL Files with the Vantage VHDL Analyzer Software

Synthesis & Optimization

- Synthesizing & Optimizing VHDL Designs with ViewSynthesis Software

Related Links

- Powerview Command-Line Syntax
- Compiling Projects with MAX+PLUS II Software
- Programming Altera® Devices
- MAX+PLUS II Development Software
- Altera Programming Hardware
- Viewlogic web site (http://www.viewlogic.com)

Using the Max2 Express Drawer's VHDL <-> max2 Utility

Once you have created a VHDL Design File (**.vhd**) for your project, you can use the **VHDL <-> max2** utility in the Max2 Express drawer to synthesize and optimize the design; generate an EDIF netlist file; and process the EDIF netlist file with the MAX+PLUS II Compiler to generate an EDIF Output File (**.edo**) for simulation. The **VHDL <-> max2** utility creates all necessary subdirectories and copies all files to the correct locations.

To use the **VHDL <-> max2** utility, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II /Viewlogic Powerview Working Environment.
2. Create a VHDL Design File that follows the guidelines described in Creating VHDL Designs for Use with MAX+PLUS II Software.
3. Start the **VHDL <-> max2** utility by double-clicking Button 1 on the **VHDL <-> max2** icon in the Max2 Express Drawer.
4. Specify the *Input VHDL file*, *Viewlogic Optimize Style*, *Viewlogic Timing Constraint File*, *Altera Device Family*, *Max2 Synthesis Style*, and the *Process Direction* options in the **VHDL <-> max2** dialog box and choose **OK**. The **VHDL <-> max2** utility generates the <*design name*>**.vsm** file for simulation with ViewSim in the **sim** subdirectory of the **max2** directory.
5. If necessary, correct any errors in the VHDL Design File and recompile the project.
6. Simulate your project, as described in Performing a Timing Simulation with ViewSim Software.

Related Links:

- Go to Performing Timing Verification for EDIF Output Files (.edo) with MOTIVE & MOTIVE for Powerview Software or Performing Timing Verification of Verilog Output Files (.vo) with MOTIVE Software in these MAX+PLUS II ACCESS^{SM} Key topics for related information.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Synthesizing & Optimizing VHDL Designs with ViewSynthesis Software

You can create and process VHDL files and convert them into Altera® Hardware Description Language (AHDL) Text Design Files (**.tdf**) or EDIF Input Files (**.edf**) that can be processed by the MAX+PLUS® II Compiler. The MAX+PLUS II Compiler can process a VHDL file that has been synthesized by ViewSynthesis software, saved as an AHDL TDF or an EDIF netlist file, and imported into the MAX+PLUS II software. The information presented here describes only how to use VHDL files that have been processed by ViewSynthesis software. For information on direct MAX+PLUS II support for VHDL Design Files, go to MAX+PLUS II VHDL Help.

To synthesize and optimize a VHDL design, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Viewlogic Powerview Working Environment.

2. Create a VHDL file <*design name*>.**vhd** using the MAX+PLUS II Text Editor or another standard text editor and save it in a working directory. Go to Creating VHDL Designs for Use with MAX+PLUS II Software for more information.

3. Start Powerview by typing `powerview` ↵ at a UNIX prompt.

4. In the Cockpit window, select *Altera* in the *Current ToolBox* drop-down list box, and select the drawer you want to use, i.e., *Design Tools* or *Max2 Express*, in the *Current Drawer* drop-down list box.

5. Choose **Create** (Project menu) from your working directory to create your project directory. Choose **OK**.

6. Choose **SearchOrder** (Project menu) to add the appropriate library directories and aliases to your **viewdraw.ini** file. Refer to Viewlogic Powerview **viewdraw.ini** Configuration File for more information on Powerview application libraries.

   ☞ When you add libraries to the **/usr/maxplus2/vwlogic/standard/viewdraw.ini** file, they are automatically set when you create a new project. Powerview tools search these libraries sequentially, so it is important to add them in the order in which they are listed.

7. Analyze the VHDL design, as described in Analyzing VHDL Files with the Vantage VHDL Analyzer Software.

8. (Optional) Perform a functional simulation, as described in Performing a Timing Simulation with ViewSim Software.

9. In Powerview 5.3.2 and previous versions, start ViewSynthesis software by double-clicking Button 1 on the **max2_syn** icon in the Altera Toolbox Design Tools Drawer.

   In Powerview 6.0, ViewSynthesis software is available only for the SunOS, and only as a command-line version. If you are using Powerview 6.0, read **/<*Powerview system*

***directory*>/README/vsyn.doc** to learn how to synthesize a design with ViewSynthesis software. You can create the **synth.ini** file, a one-line text file that contains the text `technology altera`. Then type the following commands at the UNIX prompt to analyze and synthesize your VHDL design:

```
vsyn -vhdl <design name>↵

vsyn -synth "*"↵
```

10. Choose **Target Technology** (Setup menu) and select *altera:altera* in the **Specify Target Technology** dialog box. Choose **OK**.
11. Choose **Compile VHDL** (Setup menu) and select *<design name>.vhd* in the *VHDL Files* list box. Choose **OK**.

    If more than one VHDL Design File (**.vhd**) exists for the project, you must compile the lower-level design files before compiling the top-level file.

12. Press Button 3 on the ***<design name>*** icon in the ViewSynthesis window, choose **Synthesize** from the pop-up menu, then choose **OK**.
13. (Optional) To generate a synthesis report file for the design, press Button 3 on the ***<design name>*** icon and choose **View Report** from the pop-up menu.
14. (Optional) To create a schematic representation of the gate-level netlist file, press Button 3 on the ***<design name>*** icon and choose **View Schematic** from the pop-up menu.
15. Generate an EDIF netlist file that can be compiled with the MAX+PLUS II Compiler, as described in Converting ViewDraw Schematics or VHDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the **edifneto** Utility.
16. Process the *<design name>*.**edf** file with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Installing the Altera-provided MAX+PLUS II/Viewlogic interface on your computer automatically creates the following sample ViewDraw schematic files:

- **/usr/maxplus2/examples/viewlogic/example5/count4.vhd**
- **/usr/maxplus2/examples/viewlogic/example5/count8.vhd**

Related Links:

- Go to Powerview Command-Line Syntax in these MAX+PLUS II ACCESS℠ Key topics for related information.

Feedback

Did this information help you?

If no, please log onto mySupport to file a technical request or enhancement.

Altera does not warrant that this solution will work for the customer's intended purpose and disclaims all liability for use of or reliance on the solution.

Using Viewlogic ViewSynthesis & MAX+PLUS II Software

The following topics describe how to use the Viewlogic ViewSynthesis software with MAX+PLUS® II software. Click on one of the following topics for information:

This file is suitable for printing only. It does not contain hypertext links that allow you to jump from topic to topic.

Setting Up the MAX+PLUS II/Viewlogic Powerview Working Environment

- Software Requirements
- MAX+PLUS II/Viewlogic Powerview Interface File Organization
- Viewlogic Powerview **viewdraw.ini** Configuration File
- MAX+PLUS II/Viewlogic Powerview Project File Structure
- Altera-Provided Logic & Symbol Libraries
- The **vdpath** & **mega_lpm** Libraries

VHDL Design Entry

- Design Entry Flow
- Creating VHDL Designs for Use with MAX+PLUS II Software
  - Instantiating the clklock Megafunction in VHDL or Verilog HDL
  - Instantiating RAM & ROM Functions in Viewlogic Powerview Designs
- Entering Resource Assignments
  - Modifying the Assignment & Configuration File with the **setacf** Utility
- Performing a Functional Simulation with ViewSim Software
- Analyzing VHDL Files with the Vantage VHDL Analyzer Software

Synthesis & Optimization

- Synthesizing & Optimizing VHDL Designs with ViewSynthesis Software

Related Topics:

- Go to the following MAX+PLUS II ACCESS℠ Key topics for related information:
  - Powerview Command-Line Syntax
  - Compiling Projects with MAX+PLUS II Software
  - Programming Altera® Devices
- Go to the following topics, which are available on the web, for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware
  - Viewlogic web site (http://www.viewlogic.com)

Setting Up the MAX+PLUS II/Viewlogic Powerview Working Environment

To use the MAX+PLUS® II software with Viewlogic's Powerview software, you must install the MAX+PLUS II software, familiarize yourself with the Altera® Toolbox in the Powerview Cockpit, and then establish an environment that facilitates entering and processing designs. The MAX+PLUS II /Viewlogic Powerview interface is installed automatically when you install the MAX+PLUS II software on your workstation.

To set up your working environment for the MAX+PLUS II/Viewlogic Powerview interface, follow these steps:

1. Ensure that you have correctly installed the MAX+PLUS II and Viewlogic software versions described in MAX+PLUS II/Viewlogic Powerview Software Requirements.

2. Add the following environment variable to your **.cshrc** file to specify **/usr/maxplus2** as the MAX+PLUS II system directory:

   ```
   setenv ALT_HOME /usr/maxplus2 ↵
   ```

3. Add the **$ALT_HOME/viewlogic/standard**, **$ALT_HOME/bin**, and **$ALT_HOME/viewlogic/bin** directories to the `PATH` environment variable in your **.cshrc** file.

4. Add the **$ALT_HOME/viewlogic/standard** directory to the `WDIR` environment variable in your **.cshrc** file using the following syntax:

   ```
   setenv WDIR $ALT_HOME/viewlogic/standard:/<Powerview system directory>/standard ↵
   ```

   ☞ Make sure the **$ALT_HOME/viewlogic/standard** directory is the first directory in your `WDIR` path.

5. Source your **.cshrc** file by typing `source .cshrc ↵` at the UNIX prompt.

6. Create the Viewlogic Powerview **viewdraw.ini** configuration file.

7. Copy the **/usr/maxplus2/maxplus2.ini** file to your **$HOME** directory:

   ```
   cp /usr/maxplus2/maxplus2.ini $HOME ↵

   chmod u+w $HOME/maxplus2.ini ↵
   ```

   The **maxplus2.ini** file contains both Altera- and user-specified initialization parameters that control the MAX+PLUS II software, such as MAX+PLUS II symbol and logic function library paths and the current project name. The MAX+PLUS II installation procedure creates and copies the **maxplus2.ini** file to the **/usr/maxplus2** directory.

   ☞ Normally, you do not have to edit your local copy of **maxplus2.ini**, because the MAX+PLUS II software updates the file automatically whenever you change any parameters or settings. However, if you move the **max2lib** and **max2inc** library subdirectories, you must update the file. Go to "Creating & Using a Local Copy of the **maxplus2.ini** File" in MAX+PLUS II Help for more information.

8. If you plan to instantiate Library of Parameterized Modules (LPM) functions in ViewDraw schematics, you must create a new file with the name **vdraw.vs**. The **vdraw.vs** file must include the following line:

   ```
   load ("vdpath")
   ```

   You must also make sure that you specify the **vdraw.vs** file in your `WDIR` path.

9. Set up a directory structure that facilitates working with the MAX+PLUS II/Viewlogic Powerview interface. Refer to MAX+PLUS II/Viewlogic Powerview Project File Structure.

Related Topics:

- Go to *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual for more information on installation and details on the directories that are created during MAX+PLUS II installation. Go to MAX+PLUS II/Viewlogic Powerview Interface File Organization for information about the MAX+PLUS II/Viewlogic Powerview directories that are created during MAX+PLUS II installation.
- Go to the following topics, which are available on the web, for additional information:
  - ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
  - This manual is also available in 4 parts:
    - Preface & Section 1: MAX+PLUS II Installation
    - Section 2: MAX+PLUS II - A Perspective
    - Section 3: MAX+PLUS II Tutorial
    - Appendices, Glossary & Index

MAX+PLUS II/Viewlogic Powerview Software Requirements

The following applications and utilities are used to generate, process, synthesize, and verify a project with MAX+PLUS® II and Viewlogic Powerview software.

	Viewlogic	**Altera**
ViewDraw	ViewGen	MAX+PLUS II version 9.4
VHDL Analyzer	ViewPath (optional)	
Vantage VHDL Analyzer	ViewTrace	
VHDL -> sym	ViewData Path	
edifneto	MOTIVE version 5.1.6 *Note (1)*	
edifneti	MOTIVE for Powerview version 3.2.1 (optional) *Note (1)*	
EEDIF (optional)	SDF2MTV (optional)	
MMP (optional)	Fusion/VCS	
vsm		

Note:

(1) MOTIVE for Powerview, a wrapper application for MOTIVE, provides a graphical user interface for the utilities (i.e., EEDIF, SDF2MTV, and MMP) used during a static timing verification with MOTIVE. MOTIVE alone does not accept EDIF files through the Setup Advisor.

☞ The MAX+PLUS II **read.me** file provides up-to-date information on which versions of Viewlogic Powerview applications the current version of the MAX+PLUS II software supports. It also provides information on installation and operating requirements. You should read the **read.me** file on the CD-ROM before installing the MAX+PLUS II software. After installation, you can open the **read.me** file from the MAX+PLUS II Help menu.

MAX+PLUS II/Viewlogic Powerview Interface File Organization

Table 1 shows the MAX+PLUS® II/Viewlogic Powerview interface subdirectories that are created in the MAX+PLUS II system directory (by default, the **/usr/maxplus2** directory) during MAX+PLUS II installation.

☞ For information on the other directories that are created during MAX+PLUS II installation, see "MAX+PLUS II

File Organization" in *MAX+PLUS II Installation* in the ***MAX+PLUS II Getting Started*** manual.

Table 1. MAX+PLUS II Directory Organization

Directory	Description
./lmf	Contains the Altera-provided Library Mapping File, **vwlogic.lmf**, that maps Viewlogic logic functions to equivalent MAX+PLUS II logic functions.
./viewlogic	Contains the **alt_edif.cfg** EDIF configuration file that is used with the **edifneti** utility. Also contains the library and sample subdirectories.
./viewlogic/examples	Contains the sample Viewlogic designs.
./viewlogic/library/max2sim	Contains the MAX+PLUS II simulation model library (**max2_sim**) for use in ViewSim software.
./viewlogic/library/alt_max2	Contains MAX+PLUS II primitives (`EXP`, `GLOBAL`, `LCELL`, `SOFT`, `CARRY`, `CASCADE`, `DFFE`, `DFFE6K`, and `OPNDRN`), macrofunctions (`a_8fadd`, `a_8mcomp`, `a_8count`, `a_81mux`), and megafunctions (`clklock`) for use in ViewDraw schematics. These logic functions support specific architectural features of Altera® devices. The **alt_max2** library also contains modified versions of the ViewDraw primitives that use tri-state buffers, because these primitives require special handling in the MAX+PLUS II /Viewlogic Powerview interface.
./viewlogic/library/synlib	Contains the Altera-provided synthesis library **altera**, which includes MAX+PLUS II primitives, the **altera.sml** file, a **sym** directory, and a **wir** directory for use with ViewSynthesis software.
./viewlogic/library/alt_mf	Contains the VHDL models for the MAX+PLUS II primitives (`EXP`, `GLOBAL`, `LCELL`, `SOFT`, `CARRY`, `CASCADE`, `DFFE`, and `OPNDRN`), macrofunctions (`clklock`) for use with ViewSynthesis software, the Vantage VHDL Analyzer software, and the VHDL source files. These logic functions are used to maintain portability to other architectures.
./viewlogic/library/alt_time	Contains MOTIVE timing models for MAX+PLUS II logic functions (**motive.lib**), including the `clklock` megafunction, and MAX+PLUS II driver models (**motive.drv**).
./viewlogic/library/alt_vtl	Contains the VHDL source files for the VITAL 3.0-compliant library. This library is available for ViewSim software.
./viewlogic/bin	Contains all MAX+PLUS II, Viewlogic, and interface-related scripts.
./viewlogic/standard	Contains all standard **.ini** files and standard tools.

Related Topics:

- Go to the following topics, which are available on the web, for additional information:
  - ***MAX+PLUS II Getting Started*** version 8.1 (5.4 MB)
  - This manual is also available in 4 parts:
    - Preface & Section 1: MAX+PLUS II Installation
    - Section 2: MAX+PLUS II - A Perspective
    - Section 3: MAX+PLUS II Tutorial
    - Appendices, Glossary & Index

Viewlogic Powerview viewdraw.ini Configuration File

Each Powerview project is configured with the **viewdraw.ini** file that resides in the project directory. The `DIR` statements at the end of **viewdraw.ini** are paths to library directories that are used by the various Powerview applications. Figure 1 shows a sample of the `DIR` statements that are required to use the libraries.

Figure 1. Excerpt from viewdraw.ini

```
DIR [pw] .
DIR [r]  /usr/maxplus2/vwlogic/library/alt_max2 (alt_max2)
DIR [r]  /usr/maxplus2/vwlogic/library/max2sim (max2_sim)
DIR [r]  /usr/maxplus2/vwlogic/library/synlib (altera)
DIR [r]  /usr/maxplus2/vwlogic/library/alt_mf (alt_mf)
DIR [r]  /usr/maxplus2/vwlogic/library/alt_vtl (alt_vtl)
DIR [rm] /<Powerview system directory>/lib/builtin (builtin)
DIR [rm] /<Powerview system directory>/simmods/vl/dip/74ls (vl74ls)
DIR [rm] /<Powerview system directory>/symsets/vl/dip/74ls (vl74ls)
DIR [r]  /<Powerview system directory>/lib/vdpath (vdpath)
```

When you add the libraries to the **/usr/maxplus2/vwlogic/standard/viewdraw.ini** file, they are automatically set when you create a new project. Powerview tools search these libraries sequentially, so it is important to add them in the order in which they are listed in Figure 1.

Table 1 shows the libraries that must be specified in the `DIR` statements in the **viewdraw.ini** file.

Table 1. Powerview Application Libraries

Library	Library Alias	Source	Topics
alt_max2	`alt_max2`	Altera	Graphical elements for ViewDraw
max2sim	`max2_sim`	Altera	Models for project simulation
synlib	`altera`	Altera	VHDL synthesis library for the MAX+PLUS ® II software
alt_mf	`alt_mf`	Altera	VHDL models of MAX+PLUS II logic functions
alt_vtl	`alt_vtl`	Altera	VITAL-compliant primitives
builtin	`builtin`	Altera	Basic primitives such as `INPUT` pins, `OUTPUT` pins, `AND` gates, `OR` gates, etc.
74ls	`vl74ls`	Viewlogic	74-series macrofunctions
vdpath	`vdpath`	Viewlogic	Standard library of parameterized modules (LPM) functions

The Altera-provided libraries must be listed before the Viewlogic-provided libraries in the **viewdraw.ini** file to ensure that the correct versions of the megafunctions, macrofunctions, and primitives are used.

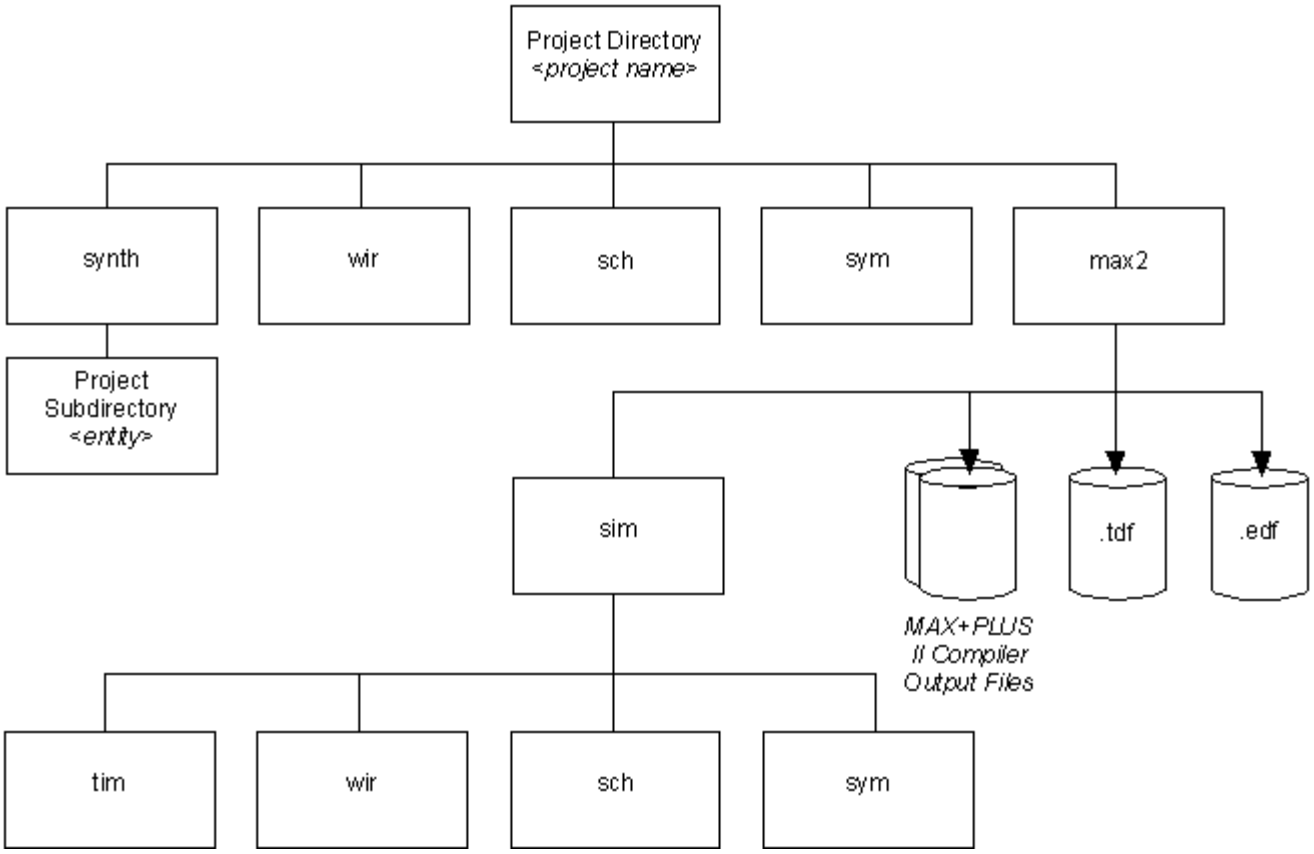
Related Topics:

- Go to Altera-Provided Logic & Symbol Libraries for more information on Altera-supplied libraries. Refer to the Powerview documentation for more information on setting up the **viewdraw.ini** file.

MAX+PLUS II/Viewlogic Powerview Project File Structure

In the MAX+PLUS® II software, a project name is the name of a top-level design file, without the filename extension. This design file can be an EDIF, Verilog HDL, or VHDL netlist file; an Altera® Hardware Description Language (AHDL) TDF; or any other MAX+PLUS II- supported design file. The EDIF netlist file must be created by Powerview and imported into the MAX+PLUS II software as an EDIF Input File (**.edf**). Figure 1 shows an example of MAX+PLUS II project directory structure that includes Powerview-generated files.

Figure 1. Sample MAX+PLUS II Project Organization



The MAX+PLUS II software stores the connectivity data on the links between design files in a hierarchical project in a Hierarchy Interconnect File (**.hif**), but refers to the entire project only by its project name. The MAX+PLUS II Compiler uses the HIF to build a single, fully flattened project database that integrates all the design files in a project hierarchy.

Unlike Powerview, the MAX+PLUS II software does not automatically create a project directory when you create a project. A single directory can contain several MAX+PLUS II design files, and you can specify any one of the designs in the directory as a project in the MAX+PLUS II software.

Viewlogic Powerview Local Work Area Structure

When you create a project with the Powerview Cockpit's **Create** command (Project menu), the project directory is created. You should generate design files and functional simulation files under this directory. A **max2** subdirectory is automatically created under your current project directory when you generate an EDIF file from your schematic or VHDL file. The ***<project name>*.edf** file is stored in the **max2** subdirectory. All MAX+PLUS® II Compiler output files are created in the **/*<project name>*/max2** subdirectory.

☞ ViewDraw files are identified by their directories and not by their extensions, so it is easy to overwrite files unintentionally. To avoid overwriting files, Altera recommends that you create a new project directory, ***<project name>*/max2/sim**, where you can generate all the files needed for simulation.

ViewDraw Project File Structure

Each ViewDraw project directory contains three subdirectories: **wir**, **sch**, and **sym**. See Table 1.

Table 1. ViewDraw Subdirectories

Directory	Topics

./wir	Wirelist files that contain connectivity information for a particular logic block
./sch	Schematics that contain logic
./sym	Symbol files that are the ViewDraw graphical representation of the logic blocks

Each file type uses the filename extension **.1**. Different file types are distinguished only by their directory: **/lib/wir/**<*project name*>**.1** is a wirelist file; **/lib/sch/**<*project name*>**.1** is the corresponding schematic file; and **/lib/sym/**<*project name*>**.1** is the corresponding symbol.

VHDL Project File Structure

Each VHDL project directory contains three subdirectories. See Table 2.

Table 2. VHDL Subdirectories

Directory	Topics
./synth	All synthesis-related files and directories
./synth/<*entity*>	Four types of files: <*entity*>**.pdf**, <*entity*>**.opt**, <*entity*>**.sta**, and <*entity*>**.gnl**
./wir	Wirelist for synthesized VHDL modules

☞ For each VHDL entity in the design, there is a corresponding **./synth/<*entity*>** directory.

Altera-Provided Logic & Symbol Libraries

The MAX+PLUS® II/Viewlogic Powerview environment provides libraries for compiling, synthesizing, and simulating designs.

☞ You can create your own libraries of custom symbols and logic functions for use in ViewDraw schematics and VHDL design files. You can use custom symbols (and functions) to incorporate an EDIF Input File, TDF, or any other MAX+PLUS II-supported design file into a project. The MAX+PLUS II software uses the **vwlogic.lmf** Library Mapping File to map ViewDraw symbols to equivalent MAX+PLUS II megafunctions, macrofunctions, or primitives. To use custom symbols and functions, you can create a custom LMF that maps your custom functions to equivalent EDIF Input Files, TDFs, or other MAX+PLUS II-supported design files. Go to "Library Mapping File" and "Viewlogic Library Mapping File" in MAX+PLUS II Help for more information.

Logic symbols used in ViewDraw software are available from the MAX+PLUS II **alt_max2** library, the ViewDraw **builtin** and **74ls** libraries, and the ViewDatapath **vdpath** library. VHDL models of MAX+PLUS II logic functions are available from the Altera-provided **alt_mf** library.

The alt_max2 Library

The **alt_max2** library provides MAX+PLUS II-specific logic functions that can be used to take advantage of special architectural features in each Altera® device family. See Table 1. Symbols and functional simulation models are available for all of these elements.

The alt_mf Library

The Altera-provided **alt_mf** library, which supports the Viewlogic Vantage VHDL Analyzer software, contains VHDL simulation models for all logic functions listed in the following table. The library is configured so that these functions pass untouched through the EDIF netlist file to the MAX+PLUS II Compiler, providing you with optimal control over design processing. Altera also provides models for all of the logic functions that you can synthesize and simulate. These models allow you to perform functional VHDL simulation while maintaining an architecture-independent VHDL description.

Table 1. Architecture Control Logic Functions

Name *Note (1), Note (2)*	Description	Name	Description	Name	Description
8fadd	8-bit full adder macrofunction	LCELL	Logic cell buffer primitive	EXP	MAX® 5000, MAX 7000, and MAX 9000 Expander buffer primitive
8mcomp	8-bit magnitude comparator macrofunction	GLOBAL	Global input buffer primitive	SOFT	Soft buffer primitive
8count	8-bit up/down counter macrofunction	CASCADE	FLEX® 6000, FLEX 8000, and FLEX 10K cascade buffer primitive	OPNDRN	Open-drain buffer primitive
81mux	8-to-1 multiplexer macrofunction	CARRY	FLEX 6000, FLEX 8000, and FLEX 10K cascade buffer primitive	DFFE *Note (2)*	D-type flipflop with Clock Enable primitive
clklock	Phase-locked loop megafunction				

Notes:

1. Logic function names that begin with a number must be prefixed with "a_" in VHDL designs. For example, 8fadd must be specified as a_8fadd.
2. For designs that are targeted to FLEX 6000 devices, you should use the DFFE primitive only if the design contains either a Clear or Preset signal, but not both. If your design contains both a Clear and a Preset signal, you must use the DFFE6K primitive.

Related Topics:

- Choose **Old-Style Macrofunctions**, **Primitives**, or **Megafunctions/LPM** from the MAX+PLUS II Help menu for detailed information on these functions.
- Go to the following topics, which are available on the web, for additional information:
  - FLEX Devices
  - MAX Devices
  - Classic Device Family

The vdpath & mega_lpm Libraries

The library of parameterized modules (LPM) 2.1.0 standard defines a set of parameterized functions and their corresponding representations in an EDIF netlist file. These logic functions allow you to create and functionally simulate an LPM-based design without targeting a specific device family. After the design is completed, you can target the design to any device family.

When the MAX+PLUS® II software processes projects that include Viewlogic-provided **vdpath** LPM functions, it uses functions from the Altera-provided **mega_lpm** library. This library includes all standard LPM functions except the truth table, finite state machine, and pad functions. Altera does not directly support the lpm_ram_dq, lpm_ram_io, and lpm_rom functions. Refer to Instantiating RAM & ROM Functions in Viewlogic Powerview Designs for instructions on instantiating RAM and ROM functions.

☞ Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu for more information about LPM functions.

MAX+PLUS II/Viewlogic Powerview Design Entry Flow

Figure 1 shows the design entry flow for the MAX+PLUS® II/Viewlogic Powerview interface.

Figure 1. MAX+PLUS II/Viewlogic Powerview Design Entry Flow

Altera-provided items are shown in blue.

Creating VHDL Designs for Use with MAX+PLUS II Software

You can create VHDL design files with the MAX+PLUS® II Text Editor or another standard text editor and save them in the appropriate directory for your project. The MAX+PLUS II Text Editor offers the following advantages:

- VHDL templates are available with the **VHDL Templates** command (Templates menu). These templates are also available in the ASCII **vhdl.tmp** file, which is located in the **/usr/maxplus2** directory.
- If you use the MAX+PLUS II Text Editor to create your VHDL design, you can use the **Syntax Coloring** command (Options menu). The Syntax Coloring feature displays keywords and other elements in text files in different colors to distinguish them from other forms of syntax.

To create a VHDL design that can be synthesized and optimized with ViewSynthesis software, follow these steps:

1. You can instantiate the following Altera-provided logic functions in your VHDL design:
    - The **alt_mf** library contains the Altera® VHDL logic function library, which includes MAX+PLUS II-specific primitives and the `a_8count`, `a_8mcomp`, `a_8fadd`, and `a_81mux` macrofunctions. If you wish to instantiate **alt_mf** logic functions in your VHDL design, you must first analyze all functions in the **alt_mf/src** directory. See Analyzing VHDL Files with the Vantage VHDL Analyzer Software for details.
    - The `clklock` megafunction, which enables the phase-locked loop, or ClockLock , circuitry available on selected Altera FLEX® 10K devices. Go to Instantiating the clklock Megafunction in VHDL or Verilog HDL for information.
    - MegaCore functions offered by Altera or by members of the Altera Megafunction Partners Program (AMPP). The OpenCore feature in the MAX+PLUS II software allows you to instantiate, compile, and simulate MegaCore functions before deciding whether to purchase a license for full device programming and post-compilation simulation support.
2. (Optional) To enter resource assignments in your VHDL design, go to Entering Resource Assignments. You can also enter resource assignments from within the MAX+PLUS II software.

Once you have created a VHDL design, you can analyze it, synthesize it, and generate an EDIF netlist file that can be imported into the MAX+PLUS II software with either of the following methods:

- You can analyze, functionally simulate, and synthesize the VHDL design, then generate an EDIF netlist file by following the steps in these topics:

    - Analyzing VHDL Files with the Vantage VHDL Analyzer Software
    - Performing a Functional Simulation with ViewSim Software
    - Synthesizing & Optimizing VHDL Designs with ViewSynthesis Software
    - Converting ViewDraw Schematics or VHDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the **edifneto** Utility

- You can use the **VHDL <-> max2** utility in the Max2 Express Drawer to automatically analyze and synthesize the VHDL design, compile it with the MAX+PLUS II Compiler, generate an EDIF Output File (**.edo**), and create a **.vsm** file for simulation. See Using the Max2 Express Drawer's **VHDL <-> max2** Utility in these MAX+PLUS II ACCESS^{SM} Key topics for details.

Installing the Altera-provided MAX+PLUS II/Viewlogic Powerview interface on your computer automatically creates the following sample VHDL files:

- **/usr/maxplus2/examples/viewlogic/example5/count4.vhd**
- **/usr/maxplus2/examples/viewlogic/example5/count8.vhd**

Related Topics:

- Go to FLEX 10K Device Family, which is available on the web, for additional information.

Instantiating the clklock Megafunction in VHDL or Verilog HDL

MAX+PLUS® II interfaces to other EDA tools support the `clklock` phase-locked loop megafunction, which can be used with some FLEX® 10K devices, with the **gencklk** utility, which is available in the MAX+PLUS II system directory. Type `gencklk -h` ↵ at the DOS or UNIX prompt to display information on how to use this utility. The **gencklk** utility generates VHDL or Verilog HDL functional simulation models and a VHDL Component Declaration template file (**.cmp**).

The **gencklk** utility allows parameters for the `clklock` function to be passed from the VHDL or Verilog HDL file to EDIF netlist format. The **gencklk** utility embeds the parameter values in the `clklock` function name; therefore, the values do not need to be declared explicitly.

To instantiate the `clklock` megafunction in VHDL or Verilog HDL, go through the following steps:

1. Type the following command at the DOS or UNIX prompt to generate the `clklock_`*x*`_`*y* function, where *x* is the ClockBoost value and *y* is the input frequency in MHz:

   ✔ Type `gencklk` <*ClockBoost*> <*input frequency*> `-vhdl` ↵ for VHDL designs.

   or:

   ✔ Type `gencklk` <*ClockBoost*> <*input frequency*> `-verilog` ↵ for Verilog HDL designs.

   Choose **Megafunctions/LPM** from the MAX+PLUS II Help menu for more information on the `clklock` megafunction.

2. Create a design file that instantiates the `clklock_`*x*`_`*y***.vhd** or `clklock_`*x*`_`*y***.v** file. The **gencklk** utility automatically generates a VHDL Component Declaration template in the **clklock_*x*_*y*.cmp** file that you can incorporate into a VHDL design file.

☞ In MAX+PLUS II version 8.3 and lower, running `genclklk` on a PC always creates files named as **clklock.vhd**, **clklock.cmp**, and **clklock.v**, regardless of the ClockBoost and input frequency values you specify.

Figures 1 and 2 show a `clklock` function with <*ClockBoost*> = 2 and <*input frequency*> = 40 MHz instantiated in VHDL and Verilog HDL design files, respectively.

Figure 1. VHDL Design File with clklock Instantiation (count8.vhd)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY altera;
USE altera.maxplus2.all;     --  Include Altera Component Declarations



ENTITY count8 IS
   PORT (a    : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
         ldn  : IN STD_LOGIC;
         gn   : IN STD_LOGIC;



dnup : IN STD_LOGIC;
         setn : IN STD_LOGIC;
         clrn : IN STD_LOGIC;
         clk  : IN STD_LOGIC;



co   : OUT STD_LOGIC;
         q    : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END count8;



ARCHITECTURE structure OF count8 IS
   signal clk2x : STD_LOGIC;



COMPONENT clklock_2_40
   PORT (
      INCLK : IN STD_LOGIC;
      OUTCLK : OUT STD_LOGIC
   );
END COMPONENT;



BEGIN
   u1: clklock_2_40
   PORT MAP (inclk=>clk, outclk=>clk2x);



u2: a_8count
   PORT MAP (a=>a(0), b=>a(1), c=>a(2), d=>a(3),
             e=>a(4), f=>a(5), g=>a(6), h=>a(7),
             clk=>clk2x,
             ldn=>ldn,
             gn=>gn,



dnup=>dnup,
             setn=>setn,
             clrn=>clrn,



qa=>q(0), qb=>q(1), qc=>q(2), qd=>q(3),
             qe=>q(4), qf=>q(5), qg=>q(6), qh=>q(7),
             cout=>co);
 END structure;
```

Figure 2. Verilog HDL Design File with clklock Instantiation (count8.v)

```
`timescale 1ns / 10ps
module count8 (a, ldn, gn, dnup, setn, clrn, clk, co, q);
output          co;
output[7:0]     q;



input[7:0]      a;
input           ldn, gn,dnup, setn, clrn, clk;
wire            clk2x;



clklock_2_40 u1 (.inclk(clk), .outclk(clk2x) );
A_8COUNT u2 (.A(a[0]), .B(a[1]), .C(a[2]), .D(a[3]), .E(a[4]), .F(a[5]),



.G(a[6]), .H(a[7]), .LDN(ldn), .GN(gn), .DNUP(dnup),
             .SETN(setn), .CLRN(clrn), .CLK(clk2x), .QA(q[0]), .QB(q[1]),
             .QC(q[2]), .QD(q[3]), .QE(q[4]), .QF(q[5]), .QG(q[6]),
             .QH(q[7]), .COUT(co) );



endmodule
```

Related Topics:

- Go to FLEX 10K Device Family, which is available on the web, for additional information.

Instantiating RAM & ROM Functions in Viewlogic Powerview Designs

The MAX+PLUS®II /Viewlogic Powerview interface offers full support for the memory capabilities of the FLEX® 10K device family, including synchronous and asynchronous RAM and ROM, cycle-shared dual-port RAM, dual-port RAM, single-Clock FIFO, and dual-Clock FIFO functions. You can use the Altera-provided **genmem** utility to generate functional simulation models and timing models for these functions. Type `genmem` ↵ at the UNIX prompt to display information on how to use this utility, as well as a list of the functions you can generate. RAM and ROM can be instantiated in both ViewDraw schematics and VHDL designs.

Refer to Viewlogic documentation for information on simulating projects that contain RAM functions. The procedure for reading an EDIF Output File and preparing it for simulation with ViewSim requires additional steps when the project contains RAM functions.

When you instantiate a RAM or ROM function, follow these general guidelines:

- For ROM functions, you must specify an initial memory content file in the Intel hexadecimal format (**.hex**) or the Altera® Memory Initialization File (**.mif**) format. The filename must be the same as the instance name; e.g., the instance name must be unique throughout the whole project, and must contain only valid name characters. The initialization file must reside in the directory containing the project's design files.
- For RAM functions, specifying a memory initialization file is optional.
- For VHDL designs, specify the name of the initial memory content file in the Generic Map Clause of the instance, with the specified type `LPM_FILE`. If you do not use an initial memory content file (e.g., for a RAM function), you should not declare or use the Generic Clause.

- Do not synthesize the **genmem**-generated VHDL file: it is intended for simulation only.

☞ The MIF format is supported only for specifying initial memory content when compiling designs within the MAX+PLUS II software. You cannot use a MIF to perform simulation with Viewlogic tools prior to MAX+PLUS II compilation.

To instantiate RAM or ROM in a ViewDraw schematic, follow these steps:

1. Use the **genmem** utility to generate a memory model by typing the following command at the UNIX prompt:

   `genmem` <*memory type*> <*memory size*> `-vwlogic` ↵

   For example: `genmem asynrom 256x15 -vwlogic` ↵

2. Start the VHDL-to-symbol utility, **vhdl2sym**, by double-clicking Button 1 on the **max2_vhdl2sym** icon in the Altera® Toolbox Design Tools Drawer.

3. Specify the following options in the **vhdl2sym** dialog box and choose **OK** to create a symbol. For example, to create the symbol for a 256x15 asynchronous ROM, enter the following settings:

Option:	**Setting:**
VHDL Source Filename	*asyn_rom_256x15.vhd*
Add LEVEL attribute	*On*

4. Choose **Comp** (Add menu), type <*design name*> in the *Enter Name* box, and choose **OK**.

To instantiate a RAM or ROM function in VHDL, follow these steps:

1. Repeat step 1 above.

2. Create a VHDL design that incorporates the text from the **genmem**-generated Component Declaration, <*memory name*>**.cmp**, and instantiate the <*memory name*> function.

Figure 1 shows a VHDL design that instantiates asyn_rom_256x15.vhd, a 256 x 15 ROM function.

Figure 1. VHDL Design File with ROM Instantiation (tstrom.vhd)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;



ENTITY tstrom IS
        PORT (
          addr    : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
          memenab : IN STD_LOGIC;
          q       : OUT STD_LOGIC_VECTOR (14 DOWNTO 0));
END tstrom;



ARCHITECTURE behavior OF tstrom IS

COMPONENT asyn_rom_256x15
     GENERIC (LPM_FILE : string);



PORT (Address : IN STD LOGIC VECTOR(7 DOWNTO 0);
```

```
            MemEnab : IN STD_LOGIC;
            Q       : OUT STD_LOGIC_VECTOR(14 DOWNTO 0)
    );
END COMPONENT;



BEGIN

   u1: asyn_rom_256x15
        GENERIC MAP (LPM_FILE => "u1.hex")
   PORT MAP (Address => addr, MemEnab => memenab, Q =>q);
END behavior;
```

Related Topics:

- Go to FLEX 10K Device Family, which is available on the web, for additional information.

Entering Resource Assignments

The MAX+PLUS® II software allows you to enter a variety of resource and device assignments for your projects. Resource assignments are used to assign logic functions to a particular pin, logic cell, I/O cell, embedded cell, row, column, Logic Array Block (LAB), Embedded Array Block (EAB), chip, clique, local routing, logic option, timing requirement, or connected pin group. In the MAX+PLUS II software, you can enter all types of resource and device assignments with Assign menu commands. You can also enter pin, logic cell, I/O cell, embedded cell, LAB, EAB, row, and column assignments in the MAX+PLUS II Floorplan Editor. The Assign menu commands and the Floorplan Editor all save assignment information in the ASCII Assignment & Configuration File (**.acf**) for the project. In addition, you can edit ACFs manually in any standard text editor.

ViewDraw Schematics

In ViewDraw schematics, you can assign a limited subset of these resource assignments by assigning properties to symbols. These properties are incorporated into the EDIF netlist file(s). The MAX+PLUS II software automatically converts assignment information from the EDIF Input File (**.edf**) into the ACF format. For information on making MAX+PLUS II-compatible resource assignments, go to the following topics:

- Assigning Pins, Logic Cells & Chips
- Assigning Cliques
- Assigning Logic Options
- Modifying the Assignment & Configuration File with the **setacf** Utility

Installing the Altera-provided MAX+PLUS II/Viewlogic interface on your computer automatically creates the following sample ViewDraw schematic file, which includes resource assignments:

- **/usr/maxplus2/examples/viewlogic/example4/fadd2mpp**

Related Topics:

- Go to Viewlogic documentation for information on how to assign properties. Go to "Resource Assignments in EDIF Input Files" and "Assigning Resources in a Third-Party Design Editor" in MAX+PLUS II Help for more information on assignments or properties that can be assigned in ViewDraw.

VHDL Design Files

For VHDL-based designs, you must use the MAX+PLUS II software or the **setacf** utility to enter resource

assignments. For information on using the **setacf** utility, go to Modifying the Assignment & Configuration File with the **setacf** Utility.

Related Topics:

- For information on entering assignments in the MAX+PLUS II software with Assign menu commands or in an ACF, go to "resource assignments" or "ACF, format" in MAX+PLUS II Help using **Search for Help on** (Help menu).

Modifying the Assignment & Configuration File with the setacf Utility

Altera provides the **setacf** utility to help you modify a project's Assignment & Configuration File (**.acf**) from the command line, without opening the file with a text editor. Type `setacf -h` ↵ at a UNIX or DOS prompt to get help on this utility.

Performing a Functional Simulation with ViewSim Software

You can use Viewlogic ViewSim software to perform a functional simulation of a ViewDraw schematic or a VHDL Design File (**.vhd**) before compiling your project with the MAX+PLUS II Compiler. Follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II /Viewlogic Powerview Working Environment.

2. Create a ViewDraw schematic that follows the guidelines in Creating ViewDraw Schematics for Use with MAX+PLUS II Software. Then go to step 3.

   or:

   Create a VHDL Design File <*design name*>**.vhd** and analyze it, as described in the following MAX+PLUS II ACCESS^{SM} Key topics:

   - Creating VHDL Designs for Use with MAX+PLUS II Software
   - Analyzing VHDL Files with the Vantage VHDL Analyzer Software

   Then go to step 7.

3. With the schematic open in the ViewDraw editor, add `CLR` and `PRE` inputs to any flipflops in your design, or tie the `CLR` and `PRE` ports of the flipflops to VCC. (Use the `PWR` primitive from the **builtin** library.)

4. Choose **Write To** (File menu) and save the schematic as <*design name*>**_funct**.

5. Start the **vsm** utility by double-clicking Button 1 on the **max2_vsmnet** icon in the Altera® Toolbox Design Tools Drawer.

6. Specify the following options in the **vsm** dialog box and choose **OK** to generate the <*design name*>**_funct.vsm** file:

Option:	**Setting:**
Design Name	<*design name*>*_funct*
Level	(blank)

7. Create a simulation command file (**.cmd**) for simulation with ViewSim software. Alternatively, you can enter

commands at the prompt in the ViewSim window. Refer to your Viewlogic documentation for more information on creating ViewSim command files.

8. Start the ViewSim simulation tool by double-clicking Button 1 on the **max2_VSim** icon in the Design Tools Drawer.

9. If you wish to simulate a ViewDraw schematic, specify the following options in the **ViewSim** dialog box, then go to step 11.

Option:	**Setting:**
Design Name	*<design name>_funct*
Command File	*<design name>_funct.cmd*
VHDL Source Window	*OFF*
VHDL Debugging	*OFF*

10. If you wish to simulate a VHDL design, specify the following options in the **ViewSim** dialog box:

Option:	**Setting:**
Design Name	*<design name>*
Command File	*<design name>.cmd*
Graphical Interface	*ON*
VHDL Source Window	*OFF* or *ON*
VHDL Debugging	*OFF* or *ON*

11. Choose **OK** to simulate the design. ViewSim software simulates the design and starts the ViewTrace waveform editor to allow you to observe the simulation results.

12. Use the **edifneto** utility to generate an EDIF Netlist File (**.edf**) that can be imported into the MAX+PLUS II software, as described in Converting ViewDraw Schematics or VHDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the **edifneto** Utility.

Related Topics:

- Go to ViewSim documentation for complete details on simulating a project and using ViewTrace to observe waveform output results.

Analyzing VHDL Files with the SpeedWave VHDL Analyzer Software

You can use the SpeedWave VHDL Analyzer software to analyze VHDL Design Files (**.vhd**) prior to functional (or gate-level) simulation with ViewSim software, or to synthesis and optimization with ViewSynthesis software. You can also use the SpeedWave VHDL Analyzer to analyze a MAX+PLUS® II - generated VHDL Output File (**.vho**) prior to post-compilation timing simulation with ViewSim software. The **max2_VantgMgr** and **max2_VantgAnlz** tools are located in the Altera® Toolbox Design Tools Drawer.

To analyze a VHDL file with the SpeedWave VHDL Analyzer, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II /Viewlogic Powerview Working Environment.

2. If you wish to analyze a VHDL Design File (**.vhd**), create a VHDL file <*design name*>**.vhd** using the MAX+PLUS II Text Editor or another standard text editor and save it in a working directory. Go to Creating VHDL Designs for Use with MAX+PLUS II Software for more information.

3. If you wish to analyze a MAX+PLUS II-generated VHDL Output File (**.vho**), be sure to select *VHDL 1987* for the *VHDL Version* option and *VHDL Output File (.vho)* for the *Write Delay Constructs To* option in the **VHDL Netlist Writer Settings** dialog box (Interfaces menu) when you set up the MAX+PLUS II Compiler to generate a VHDL Output File. See Compiling Projects with MAX+PLUS II Software for more information on generating VHDL Output Files.

4. If your VHDL file contains functions from the **alt_mf** library, follow these steps:

    1. Start the Vantage Manager by double-clicking Button 1 on the **max2_VantgMgr** icon in the Design Tools Drawer.

    2. Use the Vantage VHDL Library Manager to create an **alt_mf.lib** library file with the symbolic name `ALT_MF`.

    3. Make **alt_mf** the working library with the **Set Working** command (Edit menu).

    4. Start the VHDL Analyzer by double-clicking Button 1 on the **max2_VantgAnlz** icon in the Design Tools Drawer.

    5. Analyze each VHDL file in the **alt_mf/src** directory into the **alt_mf.lib** working library. Source files are located in the **/usr/maxplus2/vwlogic/library/alt_mf/src** directory that is created by installing the Altera/Viewlogic interface.

5. If it is not already running, start the Vantage VHDL Library Manager, as described in step 4b, to create a Vantage library.

6. Choose the **List system libs** button.

7. Add the **ieee.lib** and **synopsys.lib** system libraries to your project:

    1. Select the **ieee.lib** and **synopsys.lib** libraries from the *Available Libraries* window and choose **Add lib**. Choose the **ieee** library from the **libs_syn** directory, which is located at ***/<Powerview system directory>/*** **standard/van_vss/pgm/libs_syn**. The **ieee** library contains Synopsys package files.

    2. If your project uses functions from the **alt_mf** library, also select the **alt_mf.lib** file from the *Available Libraries* window and choose **Add lib**.

    3. Choose **Create Library** (File menu, type the project directory name in the *Symbolic Name* field, and choose **OK**.

8. Specify the project directory as the working directory by choosing **Set Working** (Edit menu).

9. Choose **Save INI File** (File menu).

10. Choose **Dismiss Window** (Powerview Red-Box menu).

11. Specify the appropriate path and file name in the **Analyzer VHDL Source File** dialog box and choose **OK** to analyze the VHDL file.

12. Once you have analyzed the file, perform one or more of the following tasks, as appropriate:

    - Performing a Functional Simulation with ViewSim Software
    - Synthesizing & Optimizing VHDL Designs with ViewSynthesis Software
    - Performing a Timing Simulation with ViewSim Software

Related Topics:

- Refer to the following sources for related information:
  - The ***Viewlogic ViewSim/VHDL User's Guide*** and ***ViewSim/VHDL Tutorial*** for information on using the Vantage VHDL Analyzer software or Vantage VHDL Library Manager
  - Powerview Command-Line Syntax in these MAX+PLUS II ACCESS^{SM} Key topics

---

Synthesizing & Optimizing VHDL Designs with ViewSynthesis Software

You can create and process VHDL files and convert them into Altera® Hardware Description Language (AHDL) Text Design Files (**.tdf**) or EDIF Input Files (**.edf**) that can be processed by the MAX+PLUS® II Compiler. The MAX+PLUS II Compiler can process a VHDL file that has been synthesized by ViewSynthesis software, saved as an AHDL TDF or an EDIF netlist file, and imported into the MAX+PLUS II software. The information presented here describes only how to use VHDL files that have been processed by ViewSynthesis software. For information on direct MAX+PLUS II support for VHDL Design Files, go to MAX+PLUS II VHDL Help.

To synthesize and optimize a VHDL design, follow these steps:

1. Be sure to set up your working environment correctly, as described in Setting Up the MAX+PLUS II/Viewlogic Powerview Working Environment.

2. Create a VHDL file <*design name*>.**vhd** using the MAX+PLUS II Text Editor or another standard text editor and save it in a working directory. Go to Creating VHDL Designs for Use with MAX+PLUS II Software for more information.

3. Start Powerview by typing `powerview` ↵ at a UNIX prompt.

4. In the Cockpit window, select *Altera* in the *Current ToolBox* drop-down list box, and select the drawer you want to use, i.e., *Design Tools* or *Max2 Express*, in the *Current Drawer* drop-down list box.

5. Choose **Create** (Project menu) from your working directory to create your project directory. Choose **OK**.

6. Choose **SearchOrder** (Project menu) to add the appropriate library directories and aliases to your **viewdraw.ini** file. Refer to Viewlogic Powerview **viewdraw.ini** Configuration File for more information on Powerview application libraries.

   ☞ When you add libraries to the **/usr/maxplus2/vwlogic/standard/viewdraw.ini** file, they are automatically set when you create a new project. Powerview tools search these libraries sequentially, so it is important to add them in the order in which they are listed.

7. Analyze the VHDL design, as described in Analyzing VHDL Files with the Vantage VHDL Analyzer Software.

8. (Optional) Perform a functional simulation, as described in Performing a Timing Simulation with ViewSim Software.

9. In Powerview 5.3.2 and previous versions, start ViewSynthesis software by double-clicking Button 1 on the **max2_syn** icon in the Altera Toolbox Design Tools Drawer.

   In Powerview 6.0, ViewSynthesis software is available only for the SunOS, and only as a command-line version. If you are using Powerview 6.0, read **/<*Powerview system*

***directory*>/README/vsyn.doc** to learn how to synthesize a design with ViewSynthesis software. You can create the **synth.ini** file, a one-line text file that contains the text `technology altera`. Then type the following commands at the UNIX prompt to analyze and synthesize your VHDL design:

```
vsyn -vhdl <design name> ↵

vsyn -synth "*" ↵
```

10. Choose **Target Technology** (Setup menu) and select *altera:altera* in the **Specify Target Technology** dialog box. Choose **OK**.
11. Choose **Compile VHDL** (Setup menu) and select *<design name>.vhd* in the *VHDL Files* list box. Choose **OK**.

    If more than one VHDL Design File (**.vhd**) exists for the project, you must compile the lower-level design files before compiling the top-level file.

12. Press Button 3 on the ***<design name>*** icon in the ViewSynthesis window, choose **Synthesize** from the pop-up menu, then choose **OK**.
13. (Optional) To generate a synthesis report file for the design, press Button 3 on the ***<design name>*** icon and choose **View Report** from the pop-up menu.
14. (Optional) To create a schematic representation of the gate-level netlist file, press Button 3 on the ***<design name>*** icon and choose **View Schematic** from the pop-up menu.
15. Generate an EDIF netlist file that can be compiled with the MAX+PLUS II Compiler, as described in Converting ViewDraw Schematics or VHDL Designs into MAX+PLUS II-Compatible EDIF Netlist Files with the **edifneto** Utility.
16. Process the *<design name>*.**edf** file with the MAX+PLUS II Compiler, as described in Compiling Projects with MAX+PLUS II Software.

Installing the Altera-provided MAX+PLUS II/Viewlogic interface on your computer automatically creates the following sample ViewDraw schematic files:

- **/usr/maxplus2/examples/viewlogic/example5/count4.vhd**
- **/usr/maxplus2/examples/viewlogic/example5/count8.vhd**

Related Topics:

- Go to Powerview Command-Line Syntax in these MAX+PLUS II ACCESS^{SM} Key topics for related information.

Powerview Command-Line Syntax

Table 1 shows the command-line syntax for using Powerview functions.

Table 1. Powerview Command-Line Syntax

Action	Command

Start VHDL Analyzer software	`vhdl -v` *<project name>*
Start ViewSynthesis software	`vhdldes`
Load Altera® technology library	`vhdldes> technology altera`
Compile a VHDL design	`vhdldes> vhdl` *<project name>*
Synthesize a design	`vhdldes> synthesize`
Generate wirelist file	`vhdldes> wir`
Create a schematic representation	`vhdldes> viewgen`
Generate a synthesis report file	`vhdldes> report`
Start the graphical user interface for ViewSynthesis	`vhdldes> vdesgui`
Start the VHDL-to-symbol utility	`vhdl2sym` *<project name>*
Start **vsm**	`vsm` *<project name>*
Start ViewSim simulator	`viewsim` *<project name>* `-`*<project name>*`.cmd`
Start **edifneto**	`edifneto -f` *<project name>*`-l` (`std` *or* `altera`) *<project name>*`.edf`
Start Vantage VHDL Analyzer software	`analyze -src` *<design file>*
Start MOTIVE for Powerview software	`mfp`

Compiling Projects with MAX+PLUS II Software

The MAX+PLUS® II Compiler can process design files in a variety of formats. This topic describes how to use MAX+PLUS II software to compile projects in which the top-level design file is an EDIF Input File (with the extension **.edf**).

☞ Refer to the following sources for additional information:

- Go to MAX+PLUS II Help for information on compiling VHDL and Verilog HDL, design files directly with the MAX+PLUS II Compiler.
- Go to Running Synopsys Compilers from MAX+PLUS II Software for information on running the Synopsys Design Compiler or FPGA Compiler software on a VHDL or Verilog HDL design from within the MAX+PLUS II Compiler window.

To compile a design (also called a "project") with MAX+PLUS II software, go through the following steps:

1. Create design files that are compatible with the MAX+PLUS II software and convert them into EDIF Input Files with the extension **.edf**. Specific instructions for some tools are described in these MAX+PLUS II ACCESS℠ Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your design entry or synthesis and optimization tool.
2. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, create a mapping for each function in a Library Mapping File (**.lmf**) to map the custom symbol to the corresponding EDIF Input File, AHDL Text Design File (**.tdf**), or other MAX+PLUS II-supported design file. These custom functions are represented in design files as hollow-body symbols or "black box" HDL descriptions.

   ☞ Go to "Library Mapping Files (**.lmf**)" in MAX+PLUS II Help for more information.
3. Open MAX+PLUS II and specify the name of your top-level design file as the project name with the

Project Name command (File menu). If you open an HDL file in the MAX+PLUS II Text Editor, you can choose the **Project Set Project to Current File** command (File menu) instead.

You can also compile a project from a command line. However, the first time you compile a project, the settings you need to specify are easier to specify from within the MAX+PLUS II software. After you have run the graphical user interface for the MAX+PLUS II software at least once, you can more easily use the command-line **setacf** utility to modify options in the Assignment & Configuration File (**.acf**) for the project. Type `setacf -h` ↵ and `maxplus2 -h` ↵ for descriptions of **setacf** and MAX+PLUS II command-line syntax.

4. Choose **Device** (Assign menu) and select the target Altera device family in the *Device Family* drop-down list box. If you wish to implement the design logic in a specific device, select it in the *Devices* box. Otherwise, select *AUTO* to allow the MAX+PLUS II Compiler to choose the best device(s) in the current device family. If your design entry or synthesis and optimization tool required you to specify a target family and/or device, specify the same information in this dialog box. For information on partitioning logic among multiple devices, go to MAX+PLUS II Help. Choose **OK**.

5. Open the Compiler window by choosing the **Compiler** command (MAX+PLUS II menu). Go through the following steps to specify the options necessary to compile the design file(s) in your project:

    1. Ensure that all EDIF netlist files have the extension **.edf** and choose **EDIF Netlist Reader Settings** (Interfaces menu).

    2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor. This name should be the name of the vendor whose tool(s) you used to create the EDIF netlist files. If your vendor name does not appear, select *Custom* instead.

        If you are compiling a design created with Synopsys FPGA Express software, select *Synopsys*, choose the **Customize** button, enter <*project name*>**.lmf** in the *LMF #1* box, choose **OK**, and skip to step 6.

    3. If you selected an existing vendor name in the *Vendor* box and your project contains design files that require custom LMF mappings, choose the **Customize** button to expand the dialog box to show all settings. Turn on the *LMF #2* checkbox and type your custom LMF's filename in the corresponding text box, or select a name from the *Files* box. The selection in the *Vendor* box will change to *Custom* and all settings will be retained until you change them again.

    4. If you selected *Custom* in the *Vendor* box, choose the **Customize** button to expand the dialog box to show all settings. Any previously defined custom settings will be displayed. Under *Signal Names*, type one or more names with up to 20 total name characters in the *VCC* or *GND* box if your EDIF Input File(s) use one or more names other than `VCC` or `GND` for the global high or low signals. Multiple signal names must be separated by either a comma (,) or a space. Under *Library Mapping Files*, turn on the *LMF #1* checkbox and type a filename in the text box following it, or select a name from the *Files* box. If necessary, specify another LMF name in the *LMF #2* box. Go to MAX+PLUS II Help for detailed information on the settings available in the **EDIF Netlist Reader Settings** dialog box.

    5. Choose **OK**.

6. If your design files contain symbols (or HDL instantiations) representing your own custom lower-level logic functions, you may need to ensure that all files are present in your project directory, i.e., the same directory as the top-level design file. Otherwise, you must specify the directories containing these files as user libraries with the **User Libraries** command (Options menu).

7. Follow all guidelines that apply to your design entry or synthesis and optimization tool:

    - Exemplar Logic Galileo Extreme-Specific Compiler Settings

- Synopsys DesignWare-Specific Compiler Settings
- Converting Synopsys FPGA Compiler & Design Compiler Timing Constraints into MAX+PLUS II-Compatible Format with the **syn2acf** Utility
- Synplicity Synplify-Specific Compiler Settings

8. If you wish to generate EDIF, VHDL, or Verilog HDL output files for post-compilation simulation or timing analysis with another EDA tool, go through the following steps:

    1. (Optional) Turn on the **Optimize Timing SNF** command (Processing menu) to reduce the size of the output file(s). Turning on this command can reduce the size of output netlists by up to 30%.

        This command does not create optimized timing SNFs on UNIX workstations. However, a non-optimized timing SNF provides the same functional and timing information as an optimized timing SNF.

    2. If you wish to generate EDIF Output Files (**.edo**), go through these steps:

        1. Turn on the **EDIF Netlist Writer** command (Interfaces menu). Then choose the **EDIF Netlist Writer Settings** command (Interfaces menu).

        2. Select a vendor name in the *Vendor* drop-down list box to activate the default settings for that vendor and choose **OK**. If your vendor name does not appear, select *Custom* instead and specify the settings that are appropriate for your simulation or timing analysis tool. Go to MAX+PLUS II Help for detailed information on the options available in the **EDIF Netlist Writer Settings** dialog box.

        3. To generate an optional Standard Delay Format (SDF) Output File (**.sdo**), choose the **Customize** button to expand the dialog box to show all settings. Select one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**.

        The filenames of the EDIF Output File(s) and optional SDF Output File(s) are the same as the user-defined chip name(s) for the project; if no chip names exist, the Compiler assigns filenames that are based on the project name. For a multi-device project, the Compiler also generates a top-level EDIF Output File that is uniquely identified by "**_t**" appended to the project name. In addition, the Compiler automatically generates a VHDL Memory Model Output File, *<project name>*.**vmo**, when it generates an EDIF Output File that contains memory (RAM or ROM).

    3. If you wish to generate VHDL Output Files (**.vho**), turn on the **VHDL Netlist Writer** command (Interfaces menu). Then choose **VHDL Netlist Writer Settings** command (Interfaces menu). Select *VHDL Output File (.vho)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF ver. 2.1 files contain timing delay information that allows you to perform back-annotation simulation in VHDL with VITAL-compliant simulation libraries. The VHDL Output Files generated by the Compiler have the extension **.vho**, but are otherwise named in the same way as the EDIF Output Files described above.

    4. If you wish to generate Verilog HDL Output Files (**.vo**), turn on the **Verilog Netlist Writer** command (Interfaces menu). Then choose **Verilog Netlist Writer Settings** command (Interfaces menu). Select *Verilog Output File (.vo)* or one of the SDF Output File options under *Write Delay Constructs To*, and choose **OK**. SDF Output Files contain timing delay information that allows you to perform back-annotation simulation in Verilog HDL. The Verilog Output Files generated by the Compiler have the extension **.vo**, but are otherwise named in the same way as the EDIF Output Files described above.

9. To run the MAX+PLUS II Compiler, choose the **Project Save & Compile** command (File menu) or choose the **Start** button in the Compiler window.

☞ See step 3 for information on running MAX+PLUS II software from the command line.

10. Once you have compiled the project with the MAX+PLUS II Compiler, you can use the VHDL, Verilog HDL, or EDIF output file(s), and the optional SDF Output File(s) (**.sdo**) to perform timing analysis or timing simulation with another EDA tool. Specific instructions for some tools are described in these MAX+PLUS II ACCESS Key Guidelines. Otherwise, refer to MAX+PLUS II Help or the product documentation for your EDA tool.

The MAX+PLUS II Compiler also generates a Report File (**.rpt**), a Pin-Out File (**.pin**), and one or more of the following files for device programming or configuration:

- JEDEC Files (**.jed**)
- Programmer Object Files (**.pof**)
- SRAM Object Files (**.sof**)
- Hexadecimal (Intel-format) Files (**.hex**)
- Tabular Text Files (**.ttf**)

Related Topics:

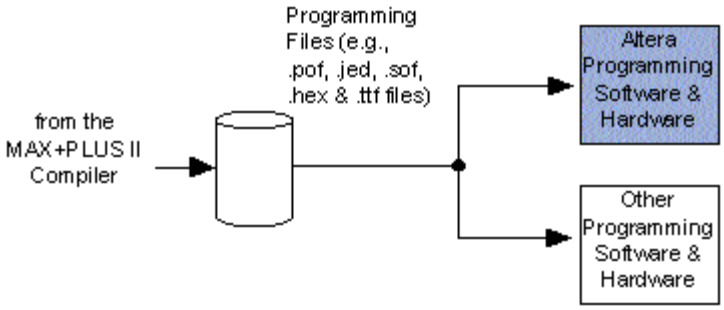
- Refer to the following sources for additional information:
  - Go to *Compiler Procedures* in MAX+PLUS II Help for information on other available Compiler settings.
  - Go to *Programmer Procedures* in MAX+PLUS II Help for instructions on creating other types of programming files and on programming or configuring Altera devices.
  - Go to Back-Annotating MAX+PLUS II Pin Assignments to Design Architect Symbols for information on back-annotating pin assignments in Mentor Graphics Design Architect schematics.
  - Go to Programming Altera Devices for information on the different programming hardware options for Altera device families.
- Go to the following topics, which are available on the web, for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware

Programming Altera Devices

Once you have successfully compiled and simulated a project with the MAX+PLUS® II software, you can program an Altera® device and test it in the target circuit. Figure 1 shows the device programming flow for MAX+PLUS II software.

Figure 1. MAX+PLUS II Device Programming Flow

Altera-provided items are shown in blue.



You can program devices with Altera programming hardware and MAX+PLUS II Programmer software installed on a 486- or Pentium-based PC or a UNIX workstation, or with programming hardware and software available from other manufacturers. Table 1 shows the available Altera programming hardware options on PCs and UNIX workstations.

Table 1. Altera Programming Hardware

Programming Hardware Option	PCs	UNIX Work-stations	MAX® 3000A Devices	Classic® & MAX 5000 Devices	MAX 7000 & MAX 7000E Devices	MAX 7000A, MAX 7000AE, MAX 7000B, MAX 7000S MAX 9000 & MAX 9000A Devices	FLEX® 6000, FLEX 6000A, FLEX 8000, FLEX 10K, FLEX 10KA, FLEX 10KB, & FLEX 10KE Devices	In-System Programming/ Configuration
Logic Programmer card, PL-MPU Master Programming Unit, and device-specific adapters	✓		✓	✓	✓	✓		
BitBlaster Download Cable	✓	✓	✓			✓	✓	✓
ByteBlasterMV Download Cable	✓		✓			✓	✓	✓
MasterBlaster Download Cable	✓	✓	✓			✓	✓	✓

If you wish to transfer programming files from a UNIX workstation to a PC over a network with File Transfer Protocol (FTP) or other similar transfer programs, be sure to select binary transfer mode.

Programming hardware from other manufacturers varies, but typically consists of a device connected to one of the serial ports on the workstation. Various vendors, such as Data I/O and BP Microsystems, supply hardware and software for programming Altera devices.

Related Topics:

- Go to Compiling Projects with MAX+PLUS II Software for information on creating programming files.
- Go to the following topics, which are available on the web, for additional information:
  - MAX+PLUS II Development Software
  - Altera Programming Hardware
  - FLEX Devices
  - MAX Devices
  - Classic Device Family