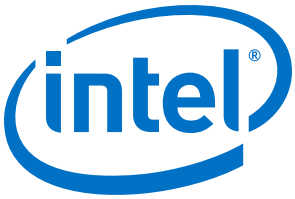


Intel® High Level Synthesis Compiler

Getting Started Guide

Updated for Intel® Quartus® Prime Design Suite: **17.1**

Subscribe
Send Feedback

UG-20036 | 2017.12.22
Latest document on the web: PDF | HTML

Contents

1 Intel® High Level Synthesis (HLS) Compiler Getting Started Guide

The Intel® High Level Synthesis (HLS) Compiler is part of Intel Quartus® Prime design software. The Intel HLS Compiler synthesizes a C++ function into an RTL implementation that is optimized for Intel FPGA products. The compiler is sometimes referred to as the i++ compiler, reflecting the name of the compiler command.

The *Intel High Level Synthesis Compiler Getting Started Guide* describes the procedures to set up the Intel HLS Compiler and to run an HLS design example.

1.1 Intel High Level Synthesis Compiler Prerequisites

To install the Intel HLS Compiler, install Intel Quartus Prime Standard Edition software or the Intel Quartus Prime Pro Edition software. The Intel HLS Compiler is installed as part of the Intel Quartus Prime software installation, but it requires additional software to use. For detailed instructions for installing Intel Quartus Prime software, including system requirements, prerequisites, and licensing requirements, see Intel FPGA Software Installation and Licensing.

Intel Corporation. All rights reserved. Intel, the Intel logo, Altera, Arria, Cyclone, Enpirion, MAX, Nios, Quartus and Stratix words and logos are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2008 Registered

Additional Software Requirements

The Intel HLS Compiler requires the following additional software:

- C++ compiler:

Linux C++ compiler	GCC compiler and C++ Libraries version 4.4.7 *Important:* Newer versions of the GCC compiler are not supported.
Windows C++ compiler	Microsoft Visual Studio 2010 Professional *Important:* Newer versions of Microsoft Visual Studio are not supported.

- Mentor Graphics* ModelSim* software:

  You can install either of the following editions of ModelSim from the Intel Quartus Prime software installer:

  — ModelSim - Intel FPGA Edition

  — ModelSim - Intel FPGA Starter Edition

  You can also use your own independently obtained and licensed version of Mentor Graphics ModelSim software.

  ModelSim software requires the Red Hat development tools packages. You can install the required packages with the following command:

  ```
  yum groupinstall "Development Tools"
  ```

  For 32-bit versions of ModelSim: If you use a 32-bit version of ModelSim (including ModelSim-Intel FPGA Edition software), you must install additional libraries. Run the following command to install the required libraries:

  ```
  yum install -y glibc.i686 glibc-devel.i686 libX11.i686
  libXext.i686 libXft.i686 libgcc.i686 libgcc.x86_64
  libstdc++.i686 libstdc++-devel.i686 ncurses-devel.i686
  qt.i686 qt-x11.i686
  ```

  For version information on all supported ModelSim software, refer to the "EDA Interface Information" section in one of the following documents:

  — Intel Quartus Prime Pro Edition Software and Device Support Release Notes

  — Intel Quartus Prime Standard Edition Software and Device Support Release Notes

Related Links

- Supported Operating Systems
- Mentor Graphics Website
- EDA Interface Information - Quartus Prime Standard Edition Software
- EDA Interface Information - Quartus Prime Pro Edition Software

1.2 Initializing the Intel HLS Compiler Environment

Before you can compile your component with the Intel HLS Compiler `i++` command, a number of environment variables must be set for the `i++` command to run successfully.

The Intel HLS Compiler environment initialization script applies only to the environment variable settings in your *current* terminal or command prompt session. You must initialize the Intel HLS Compiler environment each time that you start a terminal or command prompt session to develop your design.

To set the environment variables permanently, follow your operating system's standard procedure for making persistent changes to environment variable settings. Review the output of the environment initialization script to determine the environment variables to set permanently.

To initialize your current terminal or command prompt session so that you can run the Intel HLS Compiler:

- On Linux systems, initialize your environment as follows:
  a. In your terminal session, change directories to the `hls` directory in your Intel Quartus Prime installation directory.

     For example, `/home/<username>/intelFPGA_pro/17.1/hls`
  b. Run the following command from the `hls` directory to set the environment variables for the `i++` command in the current terminal session:

     ```
     source init_hls.sh
     ```

     The command prints out the modified environment variable settings.

  The environment initialization script shows the environment variables that it set, and you can now run the `i++` command from this terminal session.
- On Windows systems, initialize your environment as follows:
  a. In your command prompt session, change directories to the `hls` directory in your Intel Quartus Prime installation directory.

     For example, `C:\intelFPGA_pro\17.1\hls`
  b. Run the following command from the `hls` directory to set the environment variables for the `i++` command in the current terminal session:

     ```
     init_hls.bat
     ```

     The command prints out the modified environment variable settings.

  The environment initialization script shows the environment variables that it set, and you can now run the `i++` command from this command prompt session.

2 High Level Synthesis (HLS) Design Examples and Tutorials

The Intel High Level Synthesis (HLS) Compiler includes design examples and tutorials to provide you with example implementations and demonstrate ways to model or code your components to get the best results from the Intel HLS Compiler for your design.

High Level Synthesis Design Examples

The high level synthesis (HLS) design examples give you a quick way to see how various algorithms can be effectively implemented to get the best results from the Intel HLS Compiler.

You can find the HLS design examples in the following location:

```
<quartus_installdir>/hls/examples/<design_example_name>
```

Where *<quartus_installdir>* is the directory where you installed the Intel Quartus Prime Design Suite. For example, `/home/<username>/intelFPGA_pro/17.1` or `C:\intelFPGA_pro\17.1`.

For instructions on running the examples, see the following sections:

- Running a High Level Synthesis (HLS) Design Example (Linux) on page 8
- Running a High Level Synthesis (HLS) Design Example (Windows) on page 9

Table 1. HLS design examples

Focus area	Name	Description
Linear algebra	`QRD`	Uses the Modified Gram-Schmidt algorithm for QR factorization of a matrix.
Signal processing	`interp_decim_filter`	Implements a simple and efficient interpolation/decimation filter.
Simple design	`counter`	Implements a simple and efficient 32-bit counter component.
Video processing	`YUV2RGB`	Implements a basic YUV422 to RGB888 color space conversion.
Video processing	`image_downsample`	Implements an image downsampling algorithm to scale an image to a smaller size using bilinear interpolation.

HLS Design Tutorials

The HLS design tutorials show you important concepts to know when programming for the Intel HLS Compiler as well demonstrating good coding practices.

You can find the HLS design tutorials in the following location:

```
<quartus_installdir>/hls/examples/tutorials/<tutorial_name>
```

Intel Corporation. All rights reserved. Intel, the Intel logo, Altera, Arria, Cyclone, Enpirion, MAX, Nios, Quartus and Stratix words and logos are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2008 Registered

Where *<quartus_installdir>* is the directory where you installed the Intel Quartus Prime Design Suite. For example, `/home/<username>/intelFPGA_pro/17.1/hls` or `C:\intelFPGA_pro\17.1\hls`.

Each tutorial has a README file that gives you details about what the tutorial covers and instructions on how to run the tutorial.

Table 2. HLS design tutorials

Focus area	Name	Description
Arbitrary precision types	`ac_fixed_constructor`	Demonstrates the use of the `ac_fixed` constructor where you can get a better QoR by using minor variations in coding style.
	`ac_fixed_math_library`	Demonstrates the use of the Intel HLS Compiler `ac_fixed_math` fixed point math library functions.
	`ac_int_basic_ops`	Demonstrates the operators available for the `ac_int` class.
	`ac_int_overflow`	Demonstrates the usage of the `DEBUG_AC_INT_WARNING` and `DEBUG_AC_INT_ERROR` keywords to help detect overflow during emulation runtime.
Component memories	`bank_bits`	Demonstrates how to control component internal memory architecture for parallel memory access by enforcing which address bits are used for banking.
	`depth_wise_merge`	Demonstrates how to improve resource utilization by implementing two logical memories as a single physical memory with a depth equal to the sum of the depths of the two original memories.
	`static_var_init`	Demonstrates the `hls_init_on_power` and `hls_init_on_reset` flags for static variables and their impact on area and latency.
	`width_wise_merge`	Demonstrates how to improve resource utilization by implementing two logical memories as a single physical memory with a width equal to the sum of the widths of the two original memories.
Interfaces	`overview`	Demonstrates the effects on quality-of-results (QoR) of choosing different component interfaces even when the component algorithm remains the same.
	`explicit_streams_buffer`	Demonstrates how to use explicit `stream_in` and `stream_out` interfaces in the component and testbench.
	`explicit_streams_packets_ready_valid`	Demonstrates how to use the `usesPackets`, `usesValid`, and `usesReady` stream template parameters.
	`mm_master_testbench_operators`	Demonstrates how to invoke a component at different indicies of an Avalon Memory Mapped (MM) Master (`mm_master` class) interface.
	`mm_slaves`	Demonstrates how to create Avalon-MM Slave interfaces (slave registers and slave memories).
	`multiple_stream_call_sites`	Demonstrates the benefits of using multiple stream call sites.
	`pointer-mm_master`	Demonstrates how to create Avalon-MM Master interfaces and control their parameters.
	`stable_arguments`	Demonstrates how to use the `stable` attribute for unchanging arguments to improve resource utilization.

continued...

Focus area	Name	Description
Best practices	const-global	Demonstrates the performance and resource utilization improvements of using const qualified global variables.
	floating_point_ops	Demonstrates the impact of --fpc and --fp-relaxed flags in i++ on floating point operations.
	integer_promotion	Demonstrates how integer promotion rules can influence the behavior of a C or C++ program.
	loop_memory_dependency	Demonstrates breaking loop carried dependencies using the ivdep pragma.
	parameter_aliasing	Demonstrates the use of the restrict keyword on component arguments.
	resource_sharing_filter	Demonstrates the following versions of a 32-tap finite impulse response (FIR) filter design: • optimized-for-throughput variant • optimized-for-area variant
	shift_register	Demonstrates the recommended coding style for implementing shift registers.
	single_vs_double_precision_math	Demonstrates the effect of using single precision literals and functions instead of double precision literals and functions.
	struct_interface	Demonstrates how to use ac_int to implement interfaces with no padding bits.
	swap_vs_copy	Demonstrates the impact of using deep copying with registers on the performance and resource utilization of a component design.
Usability	compiler_interoperability	Demonstrates how to use testbench code compiled with the native compilers for Linux (GCC) or Windows (Microsoft Visual Studio) along with code compile by the i++ command.
	enqueue_call	Demonstrates how to run components asynchronously in the test bench using enqueue functionality.
	qsys_2xclock	Demonstrates the recommended clock and reset generation for a component with a clock2x input.
	qsys_stitching	Demonstrates how to combine multiple components to function as a single cohesive design.

2.1 Running a High Level Synthesis (HLS) Design Example (Linux)

To run an HLS design example, first compile your source code to an x86-64 binary executable and then to a simulated hardware executable.

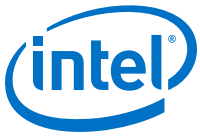
1. Start a terminal session and initialize the Intel HLS Compiler environment.

   For instructions how to initialize the environment, see Initializing the Intel HLS Compiler Environment on page 5.

2. Navigate to the <*quartus_installdir*>/hls/examples/<*design_example_name*> directory, where <*quartus_installdir*> is the directory where you installed Intel Quartus Prime software.

   For example, /home/<*username*>/intelFPGA_pro/17.1.

3. Run the make test-x86-64 command to compile the C++ source code to an x86-64 binary executable and then run the generated executable on the CPU. Expected outcome after you run the make test-x86-64 command:



- The console displays the command it uses to generate the binary (for example, `i++ -march=x86-64 -o test-x86-64 <source_files>`).
- The HLS compiler creates an executable file (for example, `test-x86-64`) in the current working directory.
- The console displays the output of the executable to signify a successful execution.

4. Run the `make test-fpga` command. The command compiles the C++ source code to a hardware executable and then runs a simulation of the generated HDL.
   Expected outcome after you run the `make test-fpga` command:
   - The console displays the command it uses to generate the testbench binary and the contents of the project directory (for example, `i++ -march="<FPGA_family_or_part_number>" -o test-fpga <source_files>`).
   - The HLS compiler creates a `.prj` directory (for example, `test-fpga.prj`) in the current working directory.
   - The console displays the output of the executable to signify a successful execution.

2.2 Running a High Level Synthesis (HLS) Design Example (Windows)

To run an HLS design example, first compile your source code to an x86-64 binary executable and then to a simulated hardware executable.

1. Start a terminal session and initialize the Intel HLS Compiler environment.

   For instructions how to initialize the environment, see Initializing the Intel HLS Compiler Environment on page 5.

2. Navigate to the `<quartus_installdir>\hls\examples\<design_example_name>` directory, where <*quartus_installdir*> is the directory where you installed Intel Quartus Prime software.

   For example, `C:\intelFPGA_pro\17.1`.

3. Run the `build.bat test-x86-64` command to compile the C++ source code to an x86-64 binary executable and then run the generated executable on the CPU.
   Expected outcome after you run the `build.bat test-x86-64` command:
   - The console displays the command it uses to generate the binary (for example, `i++ -march=x86-64 -o test-x86-64 <source_files>`).
   - The HLS compiler creates an executable file (for example, `test-x86-64`) in the current working directory.
   - The console displays the output of the executable to signify a successful execution.

4. Run the `build.bat test-fpga` command. The command compiles the C++ source code to a hardware executable and then runs a simulation of the generated HDL.
   Expected outcome after you run the `build.bat test-fpga` command:

- The console displays the command it uses to generate the testbench binary and the contents of the project directory (for example, `i++ -march="<FPGA_family_or_part_number>" -o test-fpga <source_files>`).
- The HLS compiler creates a `.prj` directory (for example, `test-fpga.prj`) in the current working directory.
- The console displays the output of the executable to signify a successful execution.

3 Troubleshooting the Setup of the Intel HLS Compiler

This section provides information that can help you troubleshoot problems you might encounter when setting up the HLS compiler.

3.1 Intel HLS Compiler Licensing Issues

The Intel High Level Synthesis (HLS) Compiler is licensed as part of your Intel Quartus Prime license. However, the Intel HLS Compiler depends on ModelSim software. In some cases, you might encounter problems with the licensing for this software.

3.1.1 Intel HLS Compiler Licensing Error Messages

The HLS compiler issues error messages if it cannot locate the license for the installed version of ModelSim software.

If the HLS compiler fails to locate the ModelSim software license, it issues the following error message when you compile your design to the FPGA architecture:

```
$ i++ -march="<FPGA_family_or_part_number>" --component dut program.cpp
HLS Elaborate cosim testbench. FAILED.
See ./a.prj/a.log for details.
Error: Missing simulator license. Either:
1) Ensure you have a valid ModelSim license
2) Use the --simulator none flag to skip the verification flow
```

Common causes for these errors include:

- Missing, expired, or invalid licenses
- Incorrect license server name in the `license.dat` file
- Unspecified or incorrectly-specified license location

Note: The running speed of the HLS compiler might decrease if the compiler has to search the network for missing or corrupted licenses. If this problem occurs, correct the license file or license location accordingly.

3.1.2 LM_LICENSE_FILE Environment Variable

Intel and third-party software use the *LM_LICENSE_FILE* environment variable to specify the locations of license files or license servers. For example, both the Intel Quartus Prime software and the ModelSim software use the *LM_LICENSE_FILE* variable to specify the locations of their licenses.

Note: The time it takes for your development machine to communicate with the license server directly affects compilation time. If your *LM_LICENSE_FILE* environment variable setting includes paths to many license servers, or if the license server is hosted in a distant locale, you will notice a significant increase in compilation time.

Intel Corporation. All rights reserved. Intel, the Intel logo, Altera, Arria, Cyclone, Enpirion, MAX, Nios, Quartus and Stratix words and logos are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2008 Registered

On Linux or UNIX systems, insert a colon (:) after each license file or license server location that you append to the *LM_LICENSE_FILE* environment variable.

On Windows systems, insert a semicolon (;) after each license file or license server location that you append to the *LM_LICENSE FILE* environment variable.

Note: When modifying the *LM_LICENSE_FILE* setting to include the locations of your software licenses, do not remove any existing license locations appended to the variable.

3.1.2.1 ModelSim Software License-Specific Considerations

When setting up the ModelSim software license, you need to append the license location to the *LM_LICENSE_FILE* environment variable. However, you can also append the location of the ModelSim software license to the *MGLS_LICENSE_FILE* environment variable.

For Mentor Graphics applications, including the ModelSim software, you can specify the paths to license files and license servers in five different locations. If you specify paths to license files or license servers in multiple locations, the following search order is used to find the first valid path:

- *MGLS_LICENSE_FILE* environment variable you set in the user environment
- *MGLS_LICENSE_FILE* environment variable you set in the registry
- *LM_LICENSE_FILE* environment variable you set in the environment
- *LM_LICENSE_FILE* environment variable you set in the registry
- `<path to FLEXlm>\license.dat`, where *<path to FLEXlm>* is the default location of the FLEXlm license file.

When you install a Mentor Graphics product license on a computer, the *MGLS_LICENSE_FILE* environment variable settings take precedence over the *LM_LICENSE_FILE* environment variable settings. If you set both environment variables, set *LM_LICENSE_FILE* to point to the ModelSim license server and set *MGLS_LICENSE_FILE* to only point to the license server for other Mentor Graphics applications. If you only use the *MGLS_LICENSE_FILE* environment variable, ensure that the ModelSim license server and the license servers for other Mentor Graphics applications are on the same machine.

A Document Revision History

Table 3. Revision History of the Intel High Level Synthesis Compiler Getting Started Guide

Date	Version	Changes
December 2017	2017.12.22	• Added the `interfaces/overview` tutorial to the list of tutorials in High Level Synthesis (HLS) Design Examples and Tutorials on page 6.
December 2017	2017.12.08	• Updated the Mentor Graphics ModelSim software requirements to include the required Red Hat development tools packages.
November 2017	2017.11.06	• The Intel High Level Synthesis (HLS) Compiler is now part of Intel Quartus Prime Design Suite, resulting in the following changes: — Revised to document to reflect that you now get the Intel HLS Compiler by installing Intel Quartus Prime software. — Removed most licensing information. Licensing the Intel HLS Compiler is now covered by your Intel Quartus Prime Design Suite licensing. Some third-party software required by the HLS compiler might continue to require additional licensing. — Removed information about overriding compilers. — Revised prerequisites to reflect only additional prerequisites required by the HLS compiler. — Revised path information to reflect the new file system locations of the Intel HLS Compiler files. • Renamed the following tutorials: — The `explicit_streams` tutorial is now called `explicit_streams_buffer`. — The `explicit_streams_2` tutorial is now called `explicit_streams_packets_ready_valid`.
June 2017	2017.06.23	• Minor changes and corrections.
June 2017	2017.06.09	• Updated High Level Synthesis (HLS) Design Examples and Tutorials on page 6 with information about new examples. • Revised Overriding the Default GCC Compiler for Intel HLS Compiler.
March 2017	2017.03.14	• Removed bit operations (3DES) from list of supplied design examples.
March 2017	2017.03.01	• Added installation of required packages and libraries needed for Linux

continued...

Intel Corporation. All rights reserved. Intel, the Intel logo, Altera, Arria, Cyclone, Enpirion, MAX, Nios, Quartus and Stratix words and logos are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2008 Registered

Date	Version	Changes
February 2017	2017.02.03	• Changed success message in Quick Start sections to PASSED. • Added *HLS Design Examples and Tutorials* section. • Moved *Running an HLS Design Example on Linux* and *Running an HLS Design Example on Windows* to *HLS Design Examples and Tutorials*.
November 2016	2016.11.30	• In *HLS Compiler Prerequisites*, updated software requirements to note that the HLS compiler supports all ModelSim software editions that the Intel Quartus Prime software supports. • In *HLS Compiler Quick Start*, added a note that you must run the `init_hls` script each time you start a shell or terminal to develop your design. • In HLS Compiler Quick Start, separated the Linux and Windows instructions. • In *Running an HLS Design Example*, separated the Linux and Windows instructions. For Linux, run the `make` command; for Windows, run the `build.bat` command. • Changed the `test_x86-64` command option to `test-x86-64`. • Changed the `test_fpga` command option to `test-fpga`. • Removed the instruction to run the `make test_qii` command for Linux and the `build.bat test_qii` command for Windows because it is no longer necessary. • In *HLS Licensing Error Messages*, updated the error message you will see if the HLS compiler fails to locate the ModelSim software license.
September 2016	2016.09.12	Initial release.