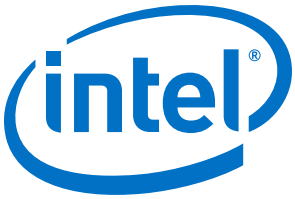


Intel® Quartus® Prime Pro Edition User Guide

Scripting

Updated for Intel® Quartus® Prime Design Suite: **19.1**

Subscribe
Send Feedback

UG-20144 | 2019.06.28
Latest document on the web: PDF | HTML

Contents

1. Tcl Scripting

You can use Tcl scripts to control the Intel® Quartus® Prime software and to perform a wide range of functions, such as compiling a design or scripting common tasks.

For example, use Tcl scripts to perform the following tasks:

- Manage an Intel Quartus Prime project
- Make assignments
- Define design constraints
- Make device assignments
- Compile your design
- Perform timing analysis
- Access reports

Tcl scripts also facilitate project or assignment migration. For example, when designing in different projects with the same prototype or development board, you can write a script to automate reassignment of pin locations in each new project. The Intel Quartus Prime software can also generate a Tcl script based on all the current assignments in the project, which aids in switching assignments to another project.

The Intel Quartus Prime software Tcl commands follow the EDA industry Tcl application programming interface (API) standards for command-line options. This simplifies learning and using Tcl commands. If you encounter an error with a command argument, the Tcl interpreter includes help information showing correct usage.

This chapter includes sample Tcl scripts for automating tasks in the Intel Quartus Prime software. You can modify these example scripts for use with your own designs. You can find more Tcl scripts in the Design Examples section of the Support area on the Intel website.

Related Information

Tcl Design Examples

1.1. Tool Command Language

Tcl (pronounced “tickle”) stands for Tool Command Language, and is the industry-standard scripting language. Tcl supports control structures, variables, network socket access, and APIs.

With Tcl, you can work seamlessly across most development platforms. Synopsys*, Mentor Graphics*, and Intel software products support the Tcl language.

By combining Tcl commands and Intel Quartus Prime API functions, you can create your own procedures and automate your design flow. Run Intel Quartus Prime software in batch mode, or execute individual Tcl commands interactively in the Intel Quartus Prime Tcl shell.

Intel Quartus Prime software supports Tcl/Tk version 8.5, supplied by the Tcl DeveloperXchange.

1.2. Intel Quartus Prime Tcl Packages

The Intel Quartus Prime software groups Tcl commands into packages by function.

Table 1. Intel Quartus Prime Tcl Packages

Package Name	Package Description
chip_planner	Identify and modify resource usage and routing with the Chip Editor
design	Manipulate project databases, including the assignments database, to enable the creation of instance assignments without modifying the `.qsf` file
device	Get device and family information from the device database
external_memif_toolkit	Interact with external memory interfaces and debug components
fif	Contains the set of Tcl functions for using the Fault Injection File (FIF) Driver
flow	Compile a project, run command-line executables, and other common flows
insystem_memory_edit	Read and edit memory contents in Intel devices
insystem_source_probe	Interact with the In-System Sources and Probes tool in an Intel device
iptclgen	Generate memory IP
jtag	Control the JTAG chain
logic_analyzer_interface	Query and modify the Logic Analyzer Interface output pin state
misc	Perform miscellaneous tasks such as enabling natural bus naming, package loading, and message posting
periph	Interact with the interface plans
project	Create and manage projects and revisions, make any project assignments including timing assignments
report	Get information from report tables, create custom reports
rtl	Traverse and query the RTL netlist of your design
sdc	Specify constraints and exceptions to the Timing Analyzer
sdc_ext	Intel-specific SDC commands
simulator	Configure and perform simulations
sta	Contain the set of Tcl functions for obtaining advanced information from the Timing Analyzer
stp	Run the Signal Tap Logic Analyzer
synthesis_report	Contain the set of Tcl functions for the Dynamic Synthesis Report tool
tdc	Obtain information from the Timing Analyzer

To keep memory requirements as low as possible, only the minimum number of packages load automatically with each Intel Quartus Prime executable. To run commands from other packages, load those packages beforehand.

Run your scripts with executables that include the packages you use in the scripts. For example, to use commands in the `sdc_ext` package, you must use the `quartus_sta` executable because `quartus_sta` is the only executable with support for the `sdc_ext` package.

The following command prints lists of the packages loaded or available to load for an executable, to the console:

```
<executable name> --tcl_eval help
```

For example, type the following command to list the packages loaded or available to load by the `quartus_fit` executable:

```
quartus_fit --tcl_eval help
```

1.2.1. Loading Packages

To load an Intel Quartus Prime Tcl package, use the `load_package` command as follows:

```
load_package [-version <version number>] <package name>
```

This command is similar to `package require`, but it allows to alternate between different versions of an Intel Quartus Prime Tcl package.

Related Information

Command Line Scripting on page 37

1.3. Intel Quartus Prime Tcl API Help

Intel Quartus Prime Tcl help allows easy access to information about the Intel Quartus Prime Tcl commands.

- This command opens the Intel Quartus Prime Command-Line and Tcl API help browser, which documents all commands and options in the Intel Quartus Prime Tcl API. At a system command prompt, access the Intel Quartus Prime Tcl API Help by typing:

  ```
  quartus_sh --qhelp
  ```

- The Tcl API Help can be accessed from the Tcl console as well. At a Tcl prompt, type

  ```
  help
  ```

  to access the help information. The output is:

The Tcl console provides help options that display specific information:

Table 2. Help Options Available in the Intel Quartus Prime Tcl Environment

Help Command	Description
`help`	Displays complete list of available Intel Quartus Prime Tcl packages.

continued...

Help Command	Description
`help -tcl`	Explains how to load Tcl packages and access command-line help.

`help -pkg <package_name -[-version <version number>]`	Displays help commands of the Intel Quartus Prime package that you specify, including the list of available Tcl commands.
	• If you do not specify `-version`, the Intel Quartus Prime software loads the latest version of the package.
	• If the package is not loaded, the Intel Quartus Prime software displays the help for the latest version of the package.
	Examples:
	`help -pkg ::quartus::project`
	`help -pkg project`
	`help -pkg project -version 1.0`

`<command_name> -h`
or
`<command_name> -help`
	Displays the short help of a Intel Quartus Prime Tcl command in a loaded package. Examples:
	`project_open -h`
	`project_open -help`

`package require ::quartus::<package name>[<version>]`	Loads a specific version of an Intel Quartus Prime Tcl package. If you do not specify `-version`, the Intel Quartus Prime software loads the latest version of the package. Example: `package require ::quartus::project 1.0` This command is similar to the `load_package` command
`load_package <package name> [-version <version number>]`	Allows you to alternate between different versions of the same package. Example: `load_package ::quartus::project -version 1.0`

`help -cmd <command_name> -[-version <version>]`
or
`<command_name> -long_help`
	Displays the complete help text for an Intel Quartus Prime Tcl command. If you do not specify `-version`, the Intel Quartus Prime software loads the latest version of the package.
	Examples:
	`project_open -long_help`
	`help -cmd project_open`
	`help -cmd project_open -version 1.0`

`help -examples`	Displays examples of Intel Quartus Prime Tcl usage.
`help -quartus`	To view help on the predefined global Tcl array that contains project information and information about the Intel Quartus Prime executable that is currently running.
`quartus_sh --qhelp`	Launches the Tk viewer for Intel Quartus Prime command-line help and display help for the command-line executables and Tcl API packages.

continued...

Help Command	Description
`help -timequestinfo`	To view help on the predefined global `"TimeQuestInfo"` Tcl array that contains delay model information and speed grade information of a Timing Analyzer design that is currently running.

The Tcl API help is also available in Intel Quartus Prime online help. Search for the command or package name to find details about that command or package.

1.3.1. Command-Line Options

You can use any of the following command line options with executables that support Tcl:

Table 3. Command-Line Options Supporting Tcl Scripting

Command-Line Option	Description
`--script=<script file> [<script args>]`	Run the specified Tcl script with optional arguments.
`-t <script file> [<script args>]`	Run the specified Tcl script with optional arguments. The `-t` option is the short form of the `--script` option.
`--shell`	Open the executable in the interactive Tcl shell mode.
`-s`	Open the executable in the interactive Tcl shell mode. The `-s` option is the short form of the `--shell` option.
`--tcl_eval <tcl command>`	Evaluate the remaining command-line arguments as Tcl commands. For example, the following command displays help for the project package: `quartus_sh --tcl_eval help -pkg project`

1.3.1.1. Run a Tcl Script

Running an executable with the `-t` option runs the specified Tcl script. You can also specify arguments to the script. Access the arguments through the `argv` variable, or use a package such as `cmdline`, which supports arguments of the following form:

```
-<argument name> <argument value>
```

The `cmdline` package is included in the *<Intel Quartus Prime directory>*`/common/tcl/packages` directory.

For example, to run a script called `myscript.tcl` with one argument, `Stratix`®, type the following command at a system command prompt:

```
quartus_sh -t myscript.tcl Stratix
```

1.3.1.2. Interactive Shell Mode

Running an executable with the `-s` option starts an interactive Tcl shell. For example, to open the Intel Quartus Prime Timing Analyzer executable in interactive shell mode, type:

```
quartus_sta -s
```

Commands you type in the Tcl shell are interpreted when you press Enter. To run a Tcl script in the interactive shell type:

```
source <script name>
```

If a command is not recognized by the shell, it is assumed to be external and executed with the `exec` command.

1.3.1.3. Evaluate as Tcl

Running an executable with the `--tcl_eval` option causes the executable to immediately evaluate the remaining command-line arguments as Tcl commands. This can be useful if you want to run simple Tcl commands from other scripting languages.

For example, the following command runs the Tcl command that prints out the commands available in the project package.

```
quartus_sh --tcl_eval help -pkg project
```

1.3.2. The Intel Quartus Prime Tcl Console Window

To run Tcl commands directly in the Intel Quartus Prime **Tcl Console** window, click **View**. By default, the **Tcl Console** window is docked in the bottom-right corner of the Intel Quartus Prime GUI. All Tcl commands typed in the **Tcl Console** are interpreted by the Intel Quartus Prime Tcl shell.

Note: Some shell commands such as `cd`, `ls`, and others can be run in the Tcl Console window, with the Tcl `exec` command. However, for best results, run shell commands and Intel Quartus Prime executables from a system command prompt outside of the Intel Quartus Prime software GUI.

Tcl messages appear in the **System** tab (**Messages** window). Errors and messages written to `stdout` and `stderr` also are shown in the Intel Quartus Prime **Tcl Console** window.

1.4. End-to-End Design Flows

You can use Tcl scripts to control all aspects of the design flow, including controlling other software, when the other software also includes a scripting interface.

Typically, EDA tools include their own script interpreters that extend core language functionality with tool-specific commands. For example, the Intel Quartus Prime Tcl interpreter supports all core Tcl commands, and adds numerous commands specific to the Intel Quartus Prime software. You can include commands in one Tcl script to run another script, which allows you to combine or chain together scripts to control different tools. Because scripts for different tools must be executed with different Tcl interpreters, it is difficult to pass information between the scripts unless one script writes information into a file and another script reads it.

Within the Intel Quartus Prime software, you can perform many different operations in a design flow (such as synthesis, fitting, and timing analysis) from a single script, making it easy to maintain global state information and pass data between the operations. However, there are some limitations on the operations you can perform in a single script due to the various packages supported by each executable.

There are no limitations on running flows from any executable. Flows include operations found in

Processing ➤ Start in the Intel Quartus Prime GUI, and are also documented as options for the `execute_flow` Tcl command. If you can make settings in the Intel Quartus Prime software and run a flow to get your desired result, you can make the same settings and run the same flow in a Tcl script.

1.5. Creating Projects and Making Assignments

You can create a script that makes all the assignments for an existing project, and then use the script at any time to restore your project settings to a known state.

Click **Project ➤ Generate Tcl File for Project** to automatically generate a `.tcl` file containing your assignments. You can source this file to recreate your project, and you can add other commands to this file, such as commands for compiling the design. This file is a good starting point to learn about project management and assignment commands.
To commit the assignments you create or modify to the `.qsf` file, you use the `export_assignments` or `project_close` commands. However, when you run the `execute_flow` command, Intel Quartus Prime software automatically commits the assignment changes to the `.qsf` file. To prevent this behavior, specify the `-dont_export_assignments` logic option.

Related Information

- Intel Quartus Prime Pro Edition Settings File Reference Manual
- Interactive Shell Mode on page 8
- Constraining Designs

1.6. Compiling Designs

You can run the Intel Quartus Prime command-line executables from Tcl scripts. Use the included `flow` package to run various Intel Quartus Prime compilation flows, or run each executable directly.

1.6.1. The flow Package

The `flow` package includes two commands for running Intel Quartus Prime command-line executables, either individually or together in standard compilation sequence.

- The `execute_module` command allows you to run an individual Intel Quartus Prime command-line executable.
- The `execute_flow` command allows you to run some or all the executables in commonly-used combinations.

Use the `flow` package instead of system calls to run Intel Quartus Prime executables from scripts or from the Intel Quartus Prime Tcl Console.

1.6.2. Compile All Revisions

You can use a simple Tcl script to compile all revisions in your project. Save the following script in a file called `compile_revisions.tcl` and type the following to run it:

```
quartus_sh -t compile_revisions.tcl <project name>
```

Compile All Revisions

```
load_package flow project_open [lindex $quartus(args) 0] set original_revision
[get_current_revision] foreach revision [get_project_revisions]
{ set_current_revision $revision execute flow -compile } set_current_revision
$original_revision project_close
```

1.7. Reporting

You can extract information from the Compilation Report to evaluate results. The Intel Quartus Prime Tcl API provides easy access to report data so you do not have to write scripts to parse the text report files.

If you know the exact report cell or cells you want to access, use the `get_report_panel_data` command and specify the row and column names (or x and y coordinates) and the name of the appropriate report panel. You can often search for data in a report panel. To do this, use a loop that reads the report one row at a time with the `get_report_panel_row` command.

Column headings in report panels are in row 0. If you use a loop that reads the report one row at a time, start with row 1 to skip column headings. The `get_number_of_rows` command returns the number of rows in the report panel, including the column heading row. Since the number of rows includes the column heading row, continue your loop if the loop index is less than the number of rows.

Report panels are hierarchically arranged and each level of hierarchy is denoted by the string "||" in the panel name. For example, the name of the Fitter Settings report panel is `Fitter||Fitter Settings` because it is in the `Fitter` folder. Panels at the highest hierarchy level do not use the "||" string. For example, the Flow Settings report panel is named `Flow Settings`.

The following Tcl code prints a list of all report panel names in your project. You can run this code with any executable that includes support for the report package.

Print All Report Panel Names

```
load_package report
project_open myproject
load_report
set panel_names [get_report_panel_names]
foreach panel_name $panel_names {
post_message "$panel_name"
}
```

1.7.1. Saving Report Data in csv Format

You can create a Comma Separated Value (`.csv`) file from any Intel Quartus Prime report to open with a spreadsheet editor.

The following Tcl code shows a simple way to create a `.csv` file with data from the Fitter panel in a report.

Create .csv Files from Reports

```
load_package report
project_open my-project
load_report
# This is the name of the report panel to save as a CSV file
set panel_name "Fitter||Fitter Settings"
set csv_file "output.csv"
set fh [open $csv_file w]
set num_rows [get_number_of_rows -name $panel_name]
# Go through all the rows in the report file, including the
# row with headings, and write out the comma-separated data
for { set i 0 } { $i < $num_rows } { incr i } {
    set row_data [get_report_panel_row -name $panel_name \
        -row $i]
    puts $fh [join $row_data ","]
}
close $fh
unload_report
```

You can modify the script to use command-line arguments to pass in the name of the project, report panel, and output file to use. You can run this script example with any executable that supports the report package.

1.8. Timing Analysis

The Intel Quartus Prime Timing Analyzer includes support for industry-standard SDC commands in the `sdc` package.

The Intel Quartus Prime software includes comprehensive Tcl APIs and SDC extensions for the Timing Analyzer in the `sta`, and `sdc_ext` packages. The Intel Quartus Prime software also includes a `tdc` package that obtains information from the Timing Analyzer.

Related Information

Intel Quartus Prime Pro Edition Settings File Reference Manual

1.9. Automating Script Execution

You can configure scripts to run automatically at various points during compilation. Use this capability to automatically run scripts that perform custom reporting, make specific assignments, and perform many other tasks.

The following three global assignments control when a script is run automatically:

- `PRE_FLOW_SCRIPT_FILE` —before a flow starts
- `POST_MODULE_SCRIPT_FILE` —after a module finishes
- `POST_FLOW_SCRIPT_FILE` —after a flow finishes

A module is another term for an Intel Quartus Prime executable that performs one step in a flow. For example, two modules are Analysis and Synthesis (`quartus_syn`), and timing analysis (`quartus_sta`).

A flow is a series of modules that the Intel Quartus Prime software runs with predefined options. For example, compiling a design is a flow that typically consists of the following steps (performed by the indicated module):

1. Analysis and Synthesis (`quartus_syn`)
2. Fitter (`quartus_fit`)
3. Assembler (`quartus_asm`)
4. Timing Analyzer (`quartus_sta`)

Other flows are described in the help for the `execute_flow` Tcl command. In addition, many commands in the **Processing** menu of the Intel Quartus Prime GUI correspond to this design flow.

To make an assignment automatically run a script, add an assignment with the following form to the `.qsf` for your project:

```
set_global_assignment -name <assignment name> <executable>:<script name>
```

The Intel Quartus Prime software runs the scripts.

```
<executable> -t <script name> <flow or module name> <project name> <revision
name>
```

The first argument passed in the `argv` variable (or `quartus(args)` variable) is the name of the flow or module being executed, depending on the assignment you use. The second argument is the name of the project and the third argument is the name of the revision.

The last process, current project, and current revision are passed to the script by the Intel Quartus Prime software and can be accessed by the following commands:

```
set process  [lindex $quartus(args) 0]
set project  [lindex $quartus(args) 1]
set revision [lindex $quartus(args) 2]

project_open $project -revision $revision
```

When you use the `POST_MODULE_SCRIPT_FILE` assignment, the specified script is automatically run after every executable in a flow. You can use a string comparison with the module name (the first argument passed in to the script) to isolate script processing to certain modules.

1.9.1. Execution Example

To illustrate how automatic script execution works in a complete flow, assume you have a project called **top** with a current revision called **rev_1**, and you have the following assignments in the `.qsf` for your project.

```
set_global_assignment -name PRE_FLOW_SCRIPT_FILE quartus_sh:first.tcl
set_global_assignment -name POST_MODULE_SCRIPT_FILE quartus_sh:next.tcl
set_global_assignment -name POST_FLOW_SCRIPT_FILE quartus_sh:last.tcl
```

When you compile your project, the `PRE_FLOW_SCRIPT_FILE` assignment causes the following command to be run before compilation begins:

```
quartus_sh -t first.tcl compile top rev_1
```

Next, the Intel Quartus Prime software starts compilation with analysis and synthesis, performed by the `quartus_syn` executable. After the Analysis and Synthesis finishes, the `POST_MODULE_SCRIPT_FILE` assignment causes the following command to run:

```
quartus_sh -t next.tcl quartus_syn top rev_1
```

Then, the Intel Quartus Prime software continues compilation with the Fitter, performed by the `quartus_fit` executable. After the Fitter finishes, the `POST_MODULE_SCRIPT_FILE` assignment runs the following command:

```
quartus_sh -t next.tcl quartus_fit top rev_1
```

Corresponding commands are run after the other stages of the compilation. When the compilation is over, the `POST_FLOW_SCRIPT_FILE` assignment runs the following command:

```
quartus_sh -t last.tcl compile top rev_1
```

1.9.2. Controlling Processing

The `POST_MODULE_SCRIPT_FILE` assignment causes a script to run after every module. Because the same script is run after every module, you might have to include some conditional statements that restrict processing in your script to certain modules.

For example, if you want a script to run only after timing analysis, use a conditional test like the following example. It checks the flow or module name passed as the first argument to the script and executes code when the module is `quartus_sta`.

Restrict Processing to a Single Module

```
set module [lindex $quartus(args) 0]
if [string match "quartus_sta" $module] {
    # Include commands here that are run
    # after timing analysis
    # Use the post-message command to display
    # messages
    post_message "Running after timing analysis"
}
```

1.9.3. Displaying Messages

Because of the way the Intel Quartus Prime software runs the scripts automatically, you must use the `post_message` command to display messages, instead of the `puts` command. This requirement applies only to scripts that are run by the three assignments listed in "Automating Script Execution".

Related Information

- The post_message Command on page 25
- Automating Script Execution on page 12

1.10. Other Scripting Features

The Intel Quartus Prime Tcl API includes other general-purpose commands and features described in this section.

1.10.1. Natural Bus Naming

The Intel Quartus Prime software supports natural bus naming. Natural bus naming allows you to use square brackets to specify bus indexes in HDL, without including escape characters to prevent Tcl from interpreting the square brackets as containing commands. For example, one signal in a bus named `address` can be identified as `address[0]` instead of `address\[0\]`. You can take advantage of natural bus naming when making assignments.

```
set_location_assignment -to address[10] Pin_M20
```

The Intel Quartus Prime software defaults to natural bus naming. You can turn off natural bus naming with the `disable_natural_bus_naming` command. For more information about natural bus naming, type the following at an Intel Quartus Prime Tcl prompt:

```
enable_natural_bus_naming -h
```

1.10.2. Short Option Names

You can use short versions of command options, if they are unambiguous. For example, the `project_open` command supports two options: `-current_revision` and `-revision`.

You can use any of the following abbreviations of the `-revision` option:

- `-r`
- `-re`
- `-rev`
- `-revi`
- `-revis`
- `-revisio`

You can use an extremely short option such as `-r` because in the case of the `project_open` command no other option starts with the letter r. However, the `report_timing` command includes the options `-recovery` and `-removal`. You cannot use `-r` or `-re` to shorten either of those options, because the abbreviation is not unique.

1.10.3. Collection Commands

Some Intel Quartus Prime Tcl functions return very large sets of data that are inefficient as Tcl lists. These data structures are referred to as collections. The Intel Quartus Prime Tcl API uses a collection ID to access the collection.

There are two Intel Quartus Prime Tcl commands for working with collections, `foreach_in_collection` and `get_collection_size`. Use the `set` command to assign a collection ID to a variable.

1.10.3.1. The foreach_in_collection Command

The `foreach_in_collection` command is similar to the `foreach` Tcl command. Use it to iterate through all elements in a collection. The following example prints all instance assignments in an open project.

foreach_in_collection Example

```
set all_instance_assignments [get_all_instance_assignments -name *]
foreach_in_collection asgn $all_instance_assignments {
    # Information about each assignment is
    # returned in a list. For information
    # about the list elements, refer to Help
    # for the get-all-instance-assignments command.
    set to [lindex $asgn 2]
    set name [lindex $asgn 3]
    set value [lindex $asgn 4]
    puts "Assignment to $to: $name = $value"
}
```

Related Information

foreach_in_collection (::quartus::misc)
In Intel Quartus Prime Help

1.10.3.2. The get_collection_size Command

Use the `get_collection_size` command to get the number of elements in a collection. The following example prints the number of global assignments in an open project.

get_collection_size Example

```
set all_global_assignments [get_all_global_assignments -name *]
set num_global_assignments [get_collection_size $all_global_assignments]
puts "There are $num_global_assignments global assignments in your project"
```

1.10.4. Node Finder Commands

The Node Finder allows you to find any node name in your project's compilation database. You can then perform various actions on found nodes, such as specifying constraints or assignments to those nodes. You can filter the search on various criteria, and also use wildcard characters in the search string.

A complete set of Node Finder Tcl commands that support the equivalent Node Finder filtering options is available for use in the scripted design flow environment.

The filtering options include the following default filters that appear in the filter combo box in the Node Finder:

Design Entry (all names) Filter on page 17

Pins: assigned Filter on page 17

Pins: unassigned Filter on page 18

Pins: input Filter on page 18

Pins: output Filter on page 18

Pins: bidirectional Filter on page 19

1.10.4.1. Design Entry (all names) Filter

This Node Finder filter finds all user-entered names in your design.

The following Tcl command demonstrates the use of the `Design Entry (all names)` filtering option:

```
set name_ids_col [get_names -filter * -node_type all \
-observable_type pre_synthesis]
foreach_in_collection name_id $name_ids_col {
     set name [get_name_info -info full_path -observable_type pre_synthesis \
$name_id]
     append name ","
     append name [get_name_info -info node_type $name_id]
     puts $name
}
```

For more information about the `get_names` command, refer to The get_names Command on page 23.

1.10.4.2. Pins: assigned Filter

This Node Finder filter finds all pin names assigned locations or other pin-related assignments.

The following Tcl command demonstrates the use of the `Pins: assigned` filtering option:

```
set name_ids_col [get_names -filter * -node_type assigned \
-observable_type pre_synthesis]
foreach_in_collection name_id $name_ids_col {
     set name [get_name_info -info full_path -observable_type pre_synthesis \
$name_id]
     append name ","
     append name [get_name_info -info node_type $name_id]
     puts $name
}
```

For more information about the `get_names` command, refer to The get_names Command on page 23.

1.10.4.3. Pins: unassigned Filter

This Node Finder filter finds all pin names unassigned locations or other pin related assignments.

The following Tcl command demonstrates the use of the `Pins: unassigned` filtering option:

```
set name_ids_col [get_names -filter * -node_type unassigned \
-observable_type pre_synthesis]
foreach_in_collection name_id $name_ids_col {
     set name [get_name_info -info full_path -observable_type pre_synthesis \
$name_id]
     append name ","
     append name [get_name_info -info node_type $name_id]
     puts $name
}
```

For more information about the `get_names` command, refer to The get_names Command on page 23.

1.10.4.4. Pins: input Filter

This Node Finder filter finds all input pin names in your design files.

The following Tcl command demonstrates the use of the `Pins: input` filtering option:

```
set name_ids_col [get_names -filter * -node_type input \
-observable_type pre_synthesis]
foreach_in_collection name_id $name_ids_col {
     set name [get_name_info -info full_path -observable_type pre_synthesis \
$name_id]
     append name ","
     append name [get_name_info -info node_type $name_id]
     puts $name
}
```

For more information about the `get_names` command, refer to The get_names Command on page 23.

1.10.4.5. Pins: output Filter

This Node Finder filter finds all output pin names in your design files.

The following Tcl command demonstrates the use of the `Pins: output` filtering option:

```
set name_ids_col [get_names -filter * -node_type output \
-observable_type pre_synthesis]
foreach_in_collection name_id $name_ids_col {
      set name [get_name_info -info full_path -observable_type pre_synthesis \
$name_id]
      append name ","
      append name [get_name_info -info node_type $name_id]
      puts $name
}
```

For more information about the `get_names` command, refer to The get_names Command on page 23.

1.10.4.6. Pins: bidirectional Filter

This Node Finder filter finds all bidirectional pin names in your design files.

The following Tcl command demonstrates the use of the `Pins: bidirectional` filtering option:

```
set name_ids_col [get_names -filter * -node_type bidir \
-observable_type pre_synthesis]
foreach_in_collection name_id $name_ids_col {
      set name [get_name_info -info full_path -observable_type pre_synthesis \
$name_id]
      append name ","
      append name [get_name_info -info node_type $name_id]
      puts $name
}
```

For more information about the `get_names` command, refer to The get_names Command on page 23.

1.10.4.7. Pins: virtual Filter

This Node Finder filter finds names of all I/O elements mapped to logic elements with a Virtual Pin logic option assignment.

The following Tcl command demonstrates the use of the `Pins: virtual` filtering option:

```
set name_ids_col [get_names -filter * -node_type virtual \
-observable_type pre_synthesis]
foreach_in_collection name_id $name_ids_col {
     set name [get_name_info -info full_path -observable_type pre_synthesis \
$name_id]
     append name ","
     append name [get_name_info -info node_type $name_id]
     puts $name
}
```

For more information about the `get_names` command, refer to The get_names Command on page 23.

1.10.4.8. Pins: all Filter

This Node Finder filter finds all pin names in your design files.

The following Tcl command demonstrates the use of the `Pins: all` filtering option:

```
set name_ids_col [get_names -filter * -node_type \
pin -observable_type pre_synthesis]
foreach_in_collection name_id $name_ids_col {
      set name [get_name_info -info full_path -observable_type pre_synthesis \
$name_id]
      append name ","
      append name [get_name_info -info node_type $name_id]
      puts $name
}
```

For more information about the `get_names` command, refer to The get_names Command on page 23.

1.10.4.9. Pins: all & Registers: post-fitting Filter

This Node Finder filter finds all pin names in your design along with all register names from your design files that persist after physical synthesis and fitting. The `Pins: all & Registers: post-fitting` filter is a combination of the `Pins: all` and `Registers: post-fitting` filters.

The following Tcl command demonstrates the use of the `Pins: all & Registers: post-fitting` filtering option:

```
set name_ids_col [get_names -filter * -node_type \
all_reg -observable_type post_fitter]
foreach_in_collection name_id $name_ids_col {
     set name [get_name_info -info full_path -observable_type post_fitter \
$name_id]
     append name ","
     append name [get_name_info -info node_type $name_id]
     puts $name
}
```

For more information about the `get_names` command, refer to The get_names Command on page 23.

1.10.4.10. Ports: partition

This Node Finder filter must be used after running the Fitter, to find nodes for post-fit partition.

Note: When you run this filter before running the Fitter, a "No nodes available. Run Fitter." message displays.

The following Tcl command demonstrates the use of the `Ports: partition` filtering option:

```
set name_ids_col [get_names -filter * -node_type partition \
-observable_type post_fitter]
foreach_in_collection name_id $name_ids_col {
     set name [get_name_info -info full_path -observable_type post_fitter \
$name_id]
     append name ","
     append name [get_name_info -info node_type $name_id]
     puts $name
}
```

For more information about the `get_names` command, refer to The get_names Command on page 23.

1.10.4.11. Entity instance: pre-synthesis Filter

This Node Finder filter finds a list of instances in the logical hierarchy for pre-synthesis netlist.

The following Tcl command demonstrates the use of the `Entity instance: pre-synthesis` filtering option:

```
set name_ids_col [get_names -filter * -node_type hierarchy \
-observable_type pre_synthesis]
foreach_in_collection name_id $name_ids_col {
     set name [get_name_info -info full_path -observable_type pre_synthesis \
$name_id]
     append name ","
```

```
     append name [get_name_info -info node_type $name_id]
     puts $name
}
```

For more information about the `get_names` command, refer to The get_names Command on page 23.

1.10.4.12. Registers: pre-synthesis Filter

This Node Finder filter finds all register names you entered in the design after Analysis and Elaboration, but before physical synthesis performs any synthesis optimizations.

The following Tcl command demonstrates the use of the `Registers: pre-synthesis` filtering option:

```
set name_ids_col [get_names -filter * -node_type reg \
-observable_type pre_synthesis]
foreach_in_collection name_id $name_ids_col {
      set name [get_name_info -info full_path -observable_type pre_synthesis \
$name_id]
      append name ","
      append name [get_name_info -info node_type $name_id]
      puts $name
}
```

For more information about the `get_names` command, refer to The get_names Command on page 23.

1.10.4.13. Registers: post-fitting Filter

This Node Finder filter finds all user-entered register names in your design files that remain after physical synthesis and fitting.

The following Tcl command demonstrates the use of the `Registers: post-fitting` filtering option:

```
set name_ids_col [get_names -filter * -node_type reg \
-observable_type post_fitter]
foreach_in_collection name_id $name_ids_col {
      set name [get_name_info -info full_path -observable_type post_fitter \
$name_id]
      append name ","
      append name [get_name_info -info node_type $name_id]
      puts $name
}
```

For more information about the `get_names` command, refer to The get_names Command on page 23.

1.10.4.14. Post-synthesis Filter

This Node Finder filter finds all user-entered and synthesis-generated names that remain in the design after design elaboration and physical synthesis.

The following Tcl command demonstrates the use of the `Post-Synthesis` filtering option:

```
set name_ids_col [get_names -filter * -node_type all \
-observable_type post_synthesis]
foreach_in_collection name_id $name_ids_col {
     set name [get_name_info -info full_path -observable_type post_synthesis \
```

```
$name_id]
     append name ","
     append name [get_name_info -info node_type $name_id]
     puts $name
}
```

For more information about the `get_names` command, refer to The get_names Command on page 23.

1.10.4.15. Post-Compilation Filter

This Node Finder filter finds all user-centered and Compiler-generated names that remain after fitting and do not have location assignments.

The following Tcl command demonstrates the use of the `Post-Compilation` filtering option:

```
set name_ids_col [get_names -filter * -node_type all \
-observable_type post_fitter]
foreach_in_collection name_id $name_ids_col {
     set name [get_name_info -info full_path -observable_type post_fitter \
$name_id]
     append name ","
     append name [get_name_info -info node_type $name_id]
     puts $name
}
```

For more information about the `get_names` command, refer to The get_names Command on page 23.

1.10.4.16. Signal Tap: pre-synthesis Filter

This Node Finder filter finds all internal device nodes in the pre-synthesis netlist that can be analyzed by the Signal Tap Logic Analyzer.

The following Tcl command demonstrates the use of the `Signal Tap: pre-synthesis` filtering option:

```
set name_ids_col [get_names -filter * -node_type all \
-observable_type pre_synthesis]
foreach_in_collection name_id $name_ids_col {
     set is_signaltap [get_name_info -info signaltapii -observable_type \
pre_synthesis $name_id]
     if {$is_signaltap == 1} {
          set name [get_name_info -use_cached_database -info full_path \
-observable_type pre_synthesis $name_id]
          append name ","
          append name [get_name_info -info node_type -observable_type \
pre_synthesis $name_id]
          puts $name
     }
}
```

For more information about the `get_names` command, refer to The get_names Command on page 23.

1.10.4.17. Signal Tap: post-fitting Filter

This Node Finder filter finds all internal device nodes in the post fit netlist that can be analyzed by the Signal Tap Logic Analyzer.

The following Tcl command demonstrates the use of the `Signal Tap: post-fitting` filtering option:

```
set name_ids_col [get_names -filter * -node_type all \
-observable_type post_fitter]
foreach_in_collection name_id $name_ids_col {
    set is_signaltap [get_name_info -info signaltapii -observable_type \
post_fitter $name_id]
    if {$is_signaltap == 1} {
        set name [get_name_info -use_cached_database -info full_path \
-observable_type post_fitter $name_id]
        append name ","
        append name [get_name_info -info node_type -observable_type \
pre_synthesis $name_id]
        puts $name
    }
}
```

For more information about the `get_names` command, refer to The get_names Command on page 23.

1.10.5. The get_names Command

To query a filtered output collection of all matching node name IDs found in a compiled Intel Quartus Prime project, use the `get_names` command.

To access each element of the output collection, use the Tcl command `foreach_in_collection`. For `get_names` or `foreach_in_collection` command example, type `get_names -long_help` or `foreach_in_collection -long_help`.

- If the `-node_type` option is not specified, the default value is `all`.
- If the `-observable_type` option is not specified, the default value is `all`.
- The node type `pin` includes `input`, `output`, `bidir`, `assigned`, `unassigned`, `virtual`, and `pin`.
- The node type `qsf` include names from the `.qsf` settings file.
- The node type `all` includes all node types.
- The node type `all_reg` includes all node types and registers post-fitting.

The value for `-observable_type` option can be one of the following:

Table 4. Values for `observable_type` Option

Observable Type	Description
`all`	Use post-Fitter information. If it is not available, post-synthesis information is used. Else, pre-synthesis information is used if it exists.
`pre_synthesis`	Use pre-synthesis information.
`post_synthesis`	Use post-synthesis information.

continued...

Observable Type	Description
post_fitter	Use post-Fitter information.
post_asm	Use post-Assembler information. The post-Assembler information is supported only for designs using the HardCopy II device family.
stp_pre_synthesis	Use Signal Tap pre-synthesis information.

Arguments

Following table lists the get_names command arguments:

Table 5. The get_names Command Arguments

Argument	Description
-h | -help	Displays a short help.
-long help	Displays a long help with examples and possible return values.
-entity<wildcard>	Specifies the entity to get names from hierarchies instantiated by the entity.
-filter<wildcard>	Specifies the node's full path name and wildcard characters.
-node_type <all|comb|reg|pin|input|output| bidir|hierarchy|mem|bus|qsf|state_machine| assigned|unassigned|all_reg|partition| virtual>	Filters based on the specified node type.
-observable_type <all|pre_synthesis| post_synthesis|post_fitter| stp_pre_synthesis>]	Filters based on the specified observable type.

Return Values

Following table lists values returned by the get_names command:

Table 6. The get_names Command Return Values

Code Name	Code	String Returned
TCL_OK	0	INFO: operation successful
TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
TCL_ERROR	1	ERROR: Get names cannot return <string> because the name was found in a partition that's not the root partition. Refine your get_names search pattern to exclude child partitions.
TCL_ERROR	1	ERROR: Compiler database does not exist for revision name: <string>. At the minimum, run Analysis & Synthesis with the specified revision name before using this Tcl command.
TCL_ERROR	1	ERROR: Illegal node type: <string>. Specify "all", "comb", "reg", "pin", "hierarchy", or "bus".
TCL_ERROR	1	ERROR: Illegal observable type: <string>. Specify "all", "pre_synthesis", "post_synthesis", or "post_fitter".
TCL_ERROR	1	ERROR: You must open a project before you can use this command.

Example Use

```
# Search for a single post-Fitter pin with the name accel and make assignments
set accel_name_id [get_names -filter accel -node_type pin -observable_type
post_fitter]

foreach_in_collection name_id $accel_name_id {
   # Get the full path name of the node
   set target [get_name_info -info full_path $name_id]
   # Set multicycle assignment
   set_multicycle_assignment -to $target 2
   # Set location assignment
   set_location_assignment -to $target Pin_E22
  }
# Search for nodes of any post-Fitter node type with name length <= 5. The
default node type is "all"
set name_ids [get_names -filter ????? -observable_type post_fitter]
foreach_in_collection name_id $name_ids {
   # Print the name id
   puts $name_id
   # Print the node type
   puts [get_name_info -info node_type $name_id]
   # Print the full path (which excludes the current focus entity from the path)
   puts [get_name_info -info full_path $name_id]
  }
# Search for nodes of any post-Fitter node type that end in "eed".
# The default node type is "all"
  set name_ids [get_names -filter *eed -observable_type post_fitter]
  foreach_in_collection name_id $name_ids {
   # Print the name id
   puts $name_id
   # Print the node type
   puts [get_name_info -info node_type $name_id]
   # Print the full path (which excludes the current
   # focus entity from the path)
   puts [get_name_info -info full_path $name_id]
  }
```

1.10.6. The post_message Command

To print messages that are formatted like Intel Quartus Prime software messages, use the `post_message` command. Messages printed by the `post_message` command appear in the **System** tab of the **Messages** window in the Intel Quartus Prime GUI, and are written to standard output when scripts are run. Arguments for the `post_message` command include an optional message type and a required message string.

The message type can be one of the following:

- `info` (default)
- `extra_info`
- `warning`
- `critical_warning`
- `error`

If you do not specify a type, Intel Quartus Prime software defaults to `info`.

With the Intel Quartus Prime software in Windows, you can color code messages displayed at the system command prompt with the `post_message` command. Add the following line to your `quartus2.ini` file:

```
DISPLAY_COMMAND_LINE_MESSAGES_IN_COLOR = on
```

The following example shows how to use the `post_message` command.

```
post_message -type warning "Design has gated clocks"
```

1.10.7. Accessing Command-Line Arguments

The global variable `quartus(args)` is a list of the arguments typed on the command-line following the name of the Tcl script.

Example 1. Simple Command-Line Argument Access

The following Tcl example prints all the arguments in the `quartus(args)` variable:

```
set i 0
foreach arg $quartus(args) {
    puts "The value at index $i is $arg"
    incr i
}
```

Example 2. Passing Command-Line Arguments to Scripts

If you copy the script in the previous example to a file named `print_args.tcl`, it displays the following output when you type the following at a command prompt.

```
quartus_sh -t print_args.tcl my_project 100MHz
The value at index 0 is my_project
The value at index 1 is 100MHz
```

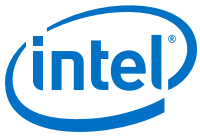
1.10.7.1. The cmdline Package

You can use the `cmdline` package included with the Intel Quartus Prime software for more robust and self-documenting command-line argument passing. The `cmdline` package supports command-line arguments with the form `-<option><value>`.

cmdline Package

```
package require cmdline
variable ::argv0 $::quartus(args)
set options {
   { "project.arg" "" "Project name" }
   { "frequency.arg" "" "Frequency" }
}
set usage "You need to specify options and values"
array set optshash [::cmdline::getoptions ::argv $options $usage]
puts "The project name is $optshash(project)"
puts "The frequency is $optshash(frequency)"
```

If you save those commands in a Tcl script called `print_cmd_args.tcl` you see the following output when you type the following command at a command prompt.



Passing Command-Line Arguments for Scripts

```
quartus_sh -t print_cmd_args.tcl -project my_project -frequency 100MHz
The project name is my_project
The frequency is 100MHz
```

Virtually all Intel Quartus Prime Tcl scripts must open a project. You can open a project, and you can optionally specify a revision name with code like the following example. The example checks whether the specified project exists. If it does, the example opens the current revision, or the revision you specify.

Full-Featured Method to Open Projects

```
package require cmdline
variable ::argv0 $::quartus(args)
set options { \
{ "project.arg" "" "Project Name" } \
{ "revision.arg" "" "Revision Name" } \
}
array set optshash [::cmdline::getoptions ::argv0 $options]
# Ensure the project exists before trying to open it
if {[project_exists $optshash(project)]} {
    if {[string equal "" $optshash(revision)]} {
        # There is no revision name specified, so default
        # to the current revision
        project_open $optshash(project) -current_revision
    } else {
        # There is a revision name specified, so open the
        # project with that revision
        project_open $optshash(project) -revision \
            $optshash(revision)
    }
} else {
    puts "Project $optshash(project) does not exist"
    exit 1
}
# The rest of your script goes here
```

If you do not require this flexibility or error checking, you can use just the `project_open` command.

Simple Method to Open Projects

```
set proj_name [lindex $argv 0]
project_open $proj_name
```

1.10.8. The quartus() Array

The global `quartus()` Tcl array includes other information about your project and the current Intel Quartus Prime executable that might be useful to your scripts. The scripts in the preceding examples parsed command line arguments found in `quartus(args)`. For information on the other elements of the `quartus()` array, type the following command at a Tcl prompt:

```
help -quartus
```

1.11. The Intel Quartus Prime Tcl Shell in Interactive Mode Example

This section presents how to make project assignments and then compile the finite impulse response (FIR) filter tutorial project with the `quartus_sh` interactive shell.

This example assumes you already have the `fir_filter` tutorial design files in a project directory.

1. To run the interactive Tcl shell, type the following at the system command prompt:

```
quartus_sh -s
```

2. Create a new project called `fir_filter`, with a revision called `filtref` by typing:

```
project_new -revision filtref fir_filter
```

Note:
- If the project file and project name are the same, the Intel Quartus Prime software gives the revision the same name as the project.
- If a `.qpf` file for this project already exists, the Intel Quartus Prime software will display an error stating that the project already exists.

Because the revision named `filtref` matches the top-level file, all design files are automatically picked up from the hierarchy tree.

3. Set a global assignment for the device:

```
set_global_assignment -name family <device family name>
```

To learn more about assignment names that you can use with the `-name` option, refer to Intel Quartus Prime Help.

Note: For assignment values that contain spaces, enclose the value in quotation marks.

4. To compile a design, use the `::quartus::flow` package, which properly exports the new project assignments and compiles the design with the proper sequence of the command-line executables. First, load the package:

```
load_package flow
```

It returns:

```
1.1
```

5. To perform a full compilation of the FIR filter design, use the `execute_flow` command with the `-compile` option:

```
execute_flow -compile
```

This command compiles the FIR filter tutorial project, exporting the project assignments and running `quartus_syn`, `quartus_fit`, `quartus_asm`, and `quartus_sta`. This sequence of events is the same as selecting **Processing ➤ Start Compilation** in the Intel Quartus Prime GUI.

6. When you are finished with a project, close it with the `project_close` command.

7. To exit the interactive Tcl shell, type `exit` at a Tcl prompt.

1.12. The tclsh Shell

On the UNIX and Linux operating systems, the tclsh shell included with the Intel Quartus Prime software is initialized with a minimal PATH environment variable. As a result, system commands might not be available within the tclsh shell because certain directories are not in the PATH environment variable.

To include other directories in the path searched by the tclsh shell, set the QUARTUS_INIT_PATH environment variable before running the tclsh shell. Directories in the QUARTUS_INIT_PATH environment variable are searched by the tclsh shell when you execute a system command.

1.13. Tcl Scripting Basics

The core Tcl commands support variables, control structures, and procedures. Additionally, there are commands for accessing the file system and network sockets, and running other programs. You can create platform-independent graphical interfaces with the Tk widget set.

Tcl commands are executed immediately as they are typed in an interactive Tcl shell. You can also create scripts (including the examples in this chapter) in files and run them with the Intel Quartus Prime executables or with the tclsh shell.

1.13.1. Hello World Example

The following shows the basic "Hello world" example in Tcl:

```
puts "Hello world"
```

Use double quotation marks to group the words `hello` and `world` as one argument. Double quotation marks allow substitutions to occur in the group. Substitutions can be simple variable substitutions, or the result of running a nested command. Use curly braces {} for grouping when you want to prevent substitutions.

1.13.2. Variables

Assign a value to a variable with the `set` command. You do not have to declare a variable before using it. Tcl variable names are case-sensitive.

```
set a 1
```

To access the contents of a variable, use a dollar sign ("$") before the variable name. The following example prints `"Hello world"` in a different way.

```
set a Hello
set b world
puts "$a $b"
```

1.13.3. Substitutions

Tcl performs three types of substitution:

- Variable value substitution
- Nested command substitution
- Backslash substitution

1.13.3.1. Variable Value Substitution

Variable value substitution, refers to accessing the value stored in a variable with a dollar sign ("$") before the variable name.

1.13.3.2. Nested Command Substitution

Nested command substitution refers to how the Tcl interpreter evaluates Tcl code in square brackets. The Tcl interpreter evaluates nested commands, starting with the innermost nested command, and commands nested at the same level from left to right. Each nested command result is substituted in the outer command.

```
set a [string length foo]
```

1.13.3.3. Backslash Substitution

Backslash substitution allows you to quote reserved characters in Tcl, such as dollar signs ("$") and braces ("[ ]"). You can also specify other special ASCII characters like tabs and new lines with backslash substitutions. A backslash before a character tells the TCL interpreter to treat the next character as a literal if the character is not the last character on the line.

```
puts "This is a \$ special character"

puts "This is a\
$ special character and line continuation"

puts "This is backslash \is ignored"

puts "This is backslash\
    continued on next line"
```

1.13.4. Arithmetic

Use the `expr` command to perform arithmetic calculations. Use curly braces ("{ }") to group the arguments of this command for greater efficiency and numeric precision.

```
set a 5
set b [expr { $a + sqrt(2) }]
```

The Intel Quartus Prime software supports all standard Tcl boolean and arithmetic operators, such as && (AND), || (OR), ! (NOT), and comparison operators such as < (less than), > (greater than), and == (equal to).

1.13.5. Lists

A Tcl list is a series of values. Supported list operations include creating lists, appending lists, extracting list elements, computing the length of a list, sorting a list, and more.

```
set a { 1 2 3 }
```

You can use the `lindex` command to extract information at a specific index in a list. Indexes are zero-based. You can use the index `end` to specify the last element in the list, or the index `end-<n>` to count from the end of the list. For example, to print the second element (at index 1) in the list stored in a use the following code.

```
puts [lindex $a 1]
```

The `llength` command returns the length of a list.

```
puts [llength $a]
```

The `lappend` command appends elements to a list. If a list does not already exist, the list you specify is created. The list variable name is not specified with a dollar sign ("$").

```
lappend a 4 5 6
```

1.13.6. Arrays

Arrays are similar to lists except that they use a string-based index. Tcl arrays are implemented as hash tables. You can create arrays by setting each element individually or with the `array set` command.

To set an element with an index of `Mon` to a value of `Monday` in an array called `days`, use the following command:

```
set days(Mon) Monday
```

The `array set` command requires a list of index/value pairs. This example sets the array called days:

```
array set days { Sun Sunday Mon Monday Tue Tuesday\
                 Wed Wednesday Thu Thursday Fri Friday Sat Saturday }
```

```
set day_abbreviation Mon
puts $days($day_abbreviation)
```

Use the `array names` command to get a list of all the indexes in a particular array. The index values are not returned in any specified order. The following example is one way to iterate over all the values in an array.

```
foreach day [array names days] {
    puts "The abbreviation $day corresponds to the day name $days($day)"
}
```

Arrays are a very flexible way of storing information in a Tcl script and are a good way to build complex data structures.

1.13.7. Control Structures

Tcl supports common control structures, including if-then-else conditions and `for`, `foreach`, and `while` loops. The position of the curly braces as shown in the following examples ensures the control structure commands are executed efficiently and correctly. The following example prints whether the value of variable a positive, negative, or zero.

If-Then-Else Structure

```
if { $a > 0 } { puts "The value is positive"
} elseif { $a < 0 } {
    puts "The value is negative"
} else {
    puts "The value is zero"
}
```

The following example uses a `for` loop to print each element in a list.

For Loop

```
set a { 1 2 3 }
for { set i 0 } { $i < [llength $a] } { incr i } {
    puts "The list element at index $i is [lindex $a $i]"
}
```

The following example uses a `foreach` loop to print each element in a list.

foreach Loop

```
set a { 1 2 3 }
foreach element $a {
    puts "The list element is $element"
}
```

The following example uses a `while` loop to print each element in a list.

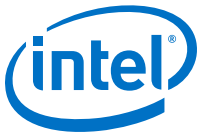
while Loop

```
set a { 1 2 3 }
set i 0
while { $i < [llength $a] } { puts "The list element at index $i is [lindex $a
$i]"
    incr i
}
```

You do not have to use the `expr` command in boolean expressions in control structure commands because they invoke the `expr` command automatically.

1.13.8. Procedures

Use the `proc` command to define a Tcl procedure (known as a subroutine or function in other scripting and programming languages). The scope of variables in a procedure is local to the procedure. If the procedure returns a value, use the `return` command to return the value from the procedure. The following example defines a procedure that multiplies two numbers and returns the result.



Simple Procedure

```
proc multiply { x y } {
    set product [expr { $x * $y }]
    return $product
}
```

The following example shows how to use the `multiply` procedure in your code. You must define a procedure before your script calls it.

Using a Procedure

```
proc multiply { x y } {
    set product [expr { $x * $y }]
    return $product
}
set a 1
set b 2
puts [multiply $a $b]
```

Define procedures near the beginning of a script. If you want to access global variables in a procedure, use the `global` command in each procedure that uses a global variable.

Accessing Global Variables

```
proc print_global_list_element { i } {
    global my_data
    puts "The list element at index $i is [lindex $my_data $i]"
}
set my_data { 1 2 3}
print_global_list_element 0
```

1.13.9. File I/O

Tcl includes commands to read from and write to files. You must open a file before you can read from or write to it, and close it when the read and write operations are done.

To open a file, use the `open` command; to close a file, use the `close` command. When you open a file, specify its name and the mode in which to open it. If you do not specify a mode, Tcl defaults to read mode. To write to a file, specify `w` for write mode.

Open a File for Writing

```
set output [open myfile.txt w]
```

Tcl supports other modes, including appending to existing files and reading from and writing to the same file.

The `open` command returns a file handle to use for read or write access. You can use the `puts` command to write to a file by specifying a file handle.

Write to a File

```
set output [open myfile.txt w]
puts $output "This text is written to the file."
close $output
```

You can read a file one line at a time with the `gets` command. The following example uses the `gets` command to read each line of the file and then prints it out with its line number.

Read from a File

```
set input [open myfile.txt]
set line_num 1
while { [gets $input line] >= 0 } {
    # Process the line of text here
    puts "$line_num: $line"
    incr line_num
}
close $input
```

1.13.10. Syntax and Comments

Arguments to Tcl commands are separated by white space, and Tcl commands are terminated by a newline character or a semicolon. You must use backslashes when a Tcl command extends more than one line. The backslash (\) must be the last character in the line to designate line extension. If the backslash is followed by any other character including a space, that character is treated as a literal.

Tcl uses the hash or pound character (#) to begin comments. The # character must begin a comment. If you prefer to include comments on the same line as a command, be sure to terminate the command with a semicolon before the # character. The following example is a valid line of code that includes a `set` command and a comment.

```
set a 1;# Initializes a
```

Without the semicolon, the command is invalid because the `set` command does not terminate until the new line after the comment.

The Tcl interpreter counts curly braces inside comments, which can lead to errors that are difficult to track down. The following example causes an error because of unbalanced curly braces.

```
# if { $x > 0 } {
if { $y > 0 } {
    # code here
}
```

1.13.11. External References

For more information about Tcl, refer to the following sources:

- Brent B. Welch and Ken Jones, and Jeffery Hobbs, *Practical Programming in Tcl and Tk* (Upper Saddle River: Prentice Hall, 2003)
- John Ousterhout and Ken Jones, *Tcl and the Tk Toolkit* (Boston: Addison-Wesley Professional, 2009)
- Mark Harrison and Michael McLennan, *Effective Tcl/Tk Programming: Writing Better Programs in Tcl and Tk* (Boston: Addison-Wesley Professional, 1997)

Related Information

www.tcl.tk
Tcl Developer Xchange

1.14. Tcl Scripting Revision History

The following revision history applies to this chapter:

Table 7. Document Revision History

Document Version	Intel Quartus Prime Version	Changes
2019.06.28	19.1	Minor correction in *The get_names Command*
2019.04.01	19.1	• Added Node Finder Tcl commands. • Added the `get_names` command. • Rectified code snippet formatting in *Compile All Revisions*, *Arrays*, and *Control Structures* topics.
2018.05.07	18.0.0	• Removed deprecated options. • External reference links updated. • Corrected typos and made minor content fixes.
2016.10.31	16.1.0	• Implemented Intel rebranding.
2015.11.02	15.1.0	• Changed instances of *Quartus II* to *Quartus Prime*. • Updated the list of Tcl packages in the *Quartus Prime Tcl Packages* section. • Updated the *Quartus Prime Tcl API Help* section: — Updated the Tcl Help Output
June 2014	14.0.0	Updated the format.
June 2012	12.0.0	• Removed survey link.
November 2011	11.0.1	• Template update • Updated supported version of Tcl in the section "Tool Command Language." • minor editorial changes
May 2011	11.0.0	Minor updates throughout document.
December 2010	10.1.0	Template update Updated to remove tcl packages used by the Classic Timing Analyzer
July 2010	10.0.0	Minor updates throughout document.
November 2009	9.1.0	• Removed LogicLock example. • Added the incremental_compilation, insystem_source_probe, and rtl packages to Table 3-1 and Table 3-2. • Added quartus_map to table 3-2.
March 2009	9.0.0	• Removed the "EDA Tool Assignments" section • Added the section "Compile All Revisions" on page 3–9 • Added the section "Using the tclsh Shell" on page 3–20
November 2008	8.1.0	Changed to 8½" × 11" page size. No change to content.
May 2008	8.0.0	Updated references.

Related Information

Documentation Archive

For previous versions of the *Intel Quartus Prime Handbook*, search the documentation archives.

2. Command Line Scripting

FPGA design software that easily integrates into your design flow saves time and improves productivity. The Intel Quartus Prime software provides you with a command-line executable for each step of the FPGA design flow to make the design process customizable and flexible.

The command-line executables are completely interchangeable with the Intel Quartus Prime GUI, allowing you to use the exact combination of tools that best suits your needs.

2.1. Benefits of Command-Line Executables

Intel Quartus Prime command-line executables give you precise control over each step of the design flow, reduce memory requirements, and improve performance.

You can group Intel Quartus Prime executable files into a script, batch file, or a makefile to automate design flows. These scripting capabilities facilitate the integration of Intel Quartus Prime software and other EDA synthesis, simulation, and verification software. Automatic design flows can perform on multiple computers simultaneously and easily archive and restore projects.

Command-line executables add flexibility without sacrificing the ease-of-use of the Intel Quartus Prime GUI. You can use the Intel Quartus Prime GUI and command-line executables at different stages in the design flow. For example, you might use the Intel Quartus Prime GUI to edit the floorplan for the design, use the command-line executables to perform place-and-route, and return to the Intel Quartus Prime GUI to perform debugging.

Command-line executables reduce the amount of memory required during each step in the design flow. Since each executable targets only one step in the design flow, the executables themselves are relatively compact, both in file size and the amount of memory used during processing. This memory usage reduction improves performance, and is particularly beneficial in design environments where heavy usage of computing resources results in reduced memory availability.

Related Information

About Command-Line Executables
in Intel Quartus Prime Help

2.2. Command-Line Scripting Help

Help for command-line executables is available through different methods. You can access help built into the executables with command-line options. You can use the Intel Quartus Prime Command-Line and Tcl API Help browser for an easy graphical view of the help information.

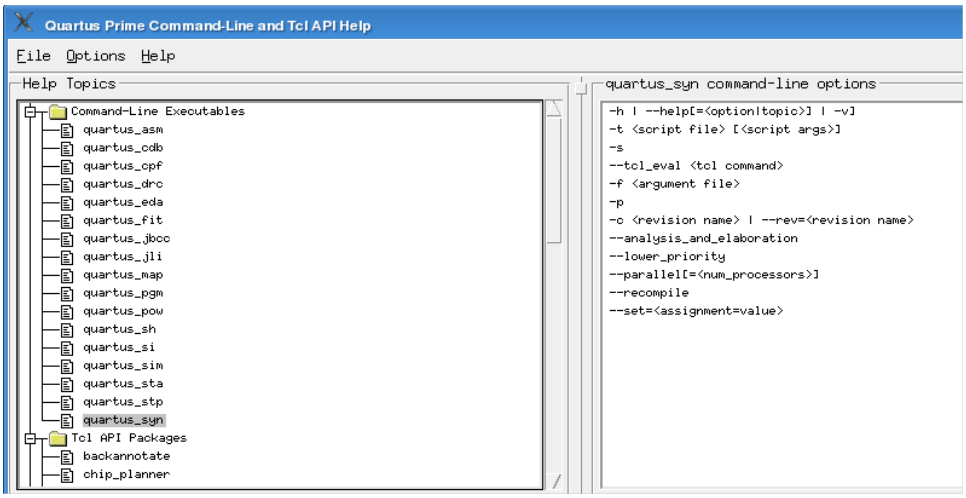
To use the Intel Quartus Prime Command-Line and Tcl API Help browser, type the following command:

```
quartus_sh --qhelp
```

This command starts the Intel Quartus Prime Command-Line and Tcl API Help browser, a viewer for information about the Intel Quartus Prime Command-Line executables and Tcl API.

Use the `-h` option with any of the Intel Quartus Prime Command-Line executables to get a description and list of supported options. Use the `--help=`*<option name>* option for detailed information about each option.

Figure 1. Intel Quartus Prime Command-Line and Tcl API Help Browser



2.3. Project Settings with Command-Line Options

The Intel Quartus Prime software command-line executables accept arguments to set project variables and access common settings.

To make assignments to an individual entity you can use the Intel Quartus Prime Tcl scripting API. On existing projects, you can also open the project in the Intel Quartus Prime GUI, change the assignment, and close the project. The changed assignment is updated in the `.qsf`. Any command-line executables that are run after this update use the updated assignment.

Related Information

- Tcl Scripting on page 4
- Intel Quartus Prime Settings File (.qsf) Definition
  in Intel Quartus Prime Help
- Intel Quartus Prime Pro Edition Settings File Reference Manual

2.3.1. Option Precedence

Project assignments follow a set of precedence rules. Assignments for a project can exist in three places:

- Intel Quartus Prime Settings File (`.qsf`)
- The compiler database
- Command-line options

The `.qsf` file contains all the project-wide and entity-level assignments and settings for the current revision for the project. The compiler database contains the result of the last compilation in the `/db` directory, and reflects the assignments at the moment when the project was compiled. Updated assignments first appear in the compiler database and later in the `.qsf` file.

Command-line options override any conflicting assignments in the `.qsf` file or the compiler database files. To specify whether the `.qsf` or compiler database files take precedence for any assignments not specified in the command-line, use the option `--read_settings_files`.

Table 8. Precedence for Reading Assignments

Option Specified	Precedence for Reading Assignments
`--read_settings_files = on` (Default)	1. Command-line options 2. The `.qsf` for the project 3. Project database (db directory, if it exists) 4. Intel Quartus Prime software defaults
`--read_settings_files = off`	1. Command-line options 2. Project database (db directory, if it exists) 3. Intel Quartus Prime software defaults

The `--write_settings_files` command-line option lists the locations to which assignments are written..

Table 9. Location for Writing Assignments

Option Specified	Location for Writing Assignments
`--write_settings_files = on` (Default)	`.qsf` file and compiler database
`--write_settings_files = off`	Compiler database

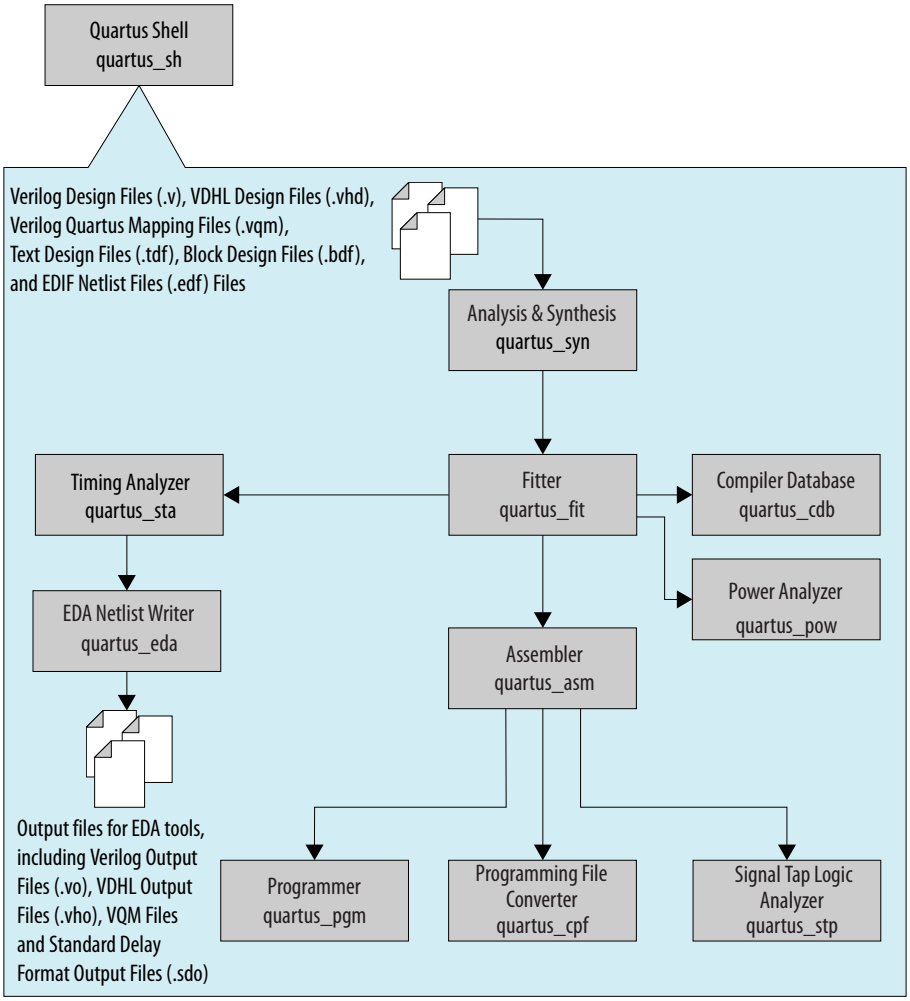
Any assignment not specified as a command-line option or found in the `.qsf` file or compiler database file is set to its default value.

Use the options `--read_settings_files=off` and `--write_settings_files=off` (where appropriate) to optimize the way that the Intel Quartus Prime software reads and updates settings files.

2.4. Compilation with quartus_sh --flow

The figure shows a typical Intel Quartus Prime FPGA design flow using command-line executables.

Figure 2. Typical Design Flow



Use the `quartus_sh` executable with the `--flow` option to perform a complete compilation flow with a single command. The `--flow` option supports the smart recompile feature and efficiently sets command-line arguments for each executable in the flow.

The following example runs compilation, timing analysis, and programming file generation with a single command:

```
quartus_sh --flow compile filtref
```

Tip: For information about specialized flows, type `quartus_sh --help=flow` at a command prompt.

2.5. Text-Based Report Files

Each command-line executable creates a text report file when it is run. These files report success or failure, and contain information about the processing performed by the executable.

Report file names contain the revision name and the short-form name of the executable that generated the report file, in the format `<revision>.<executable>.rpt`. For example, using the `quartus_fit` executable to place and route a project with the revision name **design_top** generates a report file named `design_top.fit.rpt`. Similarly, using the `quartus_sta` executable to perform timing analysis on a project with the revision name **fir_filter** generates a report file named `fir_filter.sta.rpt`.

As an alternative to parsing text-based report files, you can use the `::quartus::report` Tcl package.

Related Information

- Text-Format Report File (.rpt) Definition
  in Intel Quartus Prime Help
- ::quartus::report
  in Intel Quartus Prime Help

2.6. Using Command-Line Executables in Scripts

You can use command-line executables in scripts that control other software, in addition to Intel Quartus Prime software. For example, if your design flow uses third-party synthesis or simulation software, and you can run this other software at the command prompt, you can group those commands with Intel Quartus Prime executables in a single script.

To set up a new project and apply individual constraints, such as pin location assignments and timing requirements, you must use a Tcl script or the Intel Quartus Prime GUI.

Command-line executables are very useful for working with existing projects, for making common global settings, and for performing common operations. For more flexibility in a flow, use a Tcl script. Additionally, Tcl scripts simplify passing data between different stages of the design flow.

For example, you can create a UNIX shell script to run a third-party synthesis software, place-and-route the design in the Intel Quartus Prime software, and generate output netlists for other simulation software.

2.7. The QFlow Script

A Tcl/Tk-based graphical interface called QFlow is included with the command-line executables. You can use the QFlow interface to open projects, launch some of the command-line executables, view report files, and make some global project assignments.

The QFlow interface can run the following command-line executables:

- `quartus_syn` (Analysis and Synthesis)
- `quartus_fit` (Fitter)
- `quartus_sta` (Timing Analyzer)
- `quartus_asm` (Assembler)
- `quartus_eda` (EDA Netlist Writer)

To view floorplans or perform other GUI-intensive tasks, launch the Intel Quartus Prime software.

Start QFlow by typing the following command at a command prompt:

```
quartus_sh -g
```

Tip: The QFlow script is located in the *<Intel Quartus Prime directory>*`/common/tcl/apps/qflow/` directory.

2.8. Command-Line Scripting Revision History

The following revision history applies to this chapter:

Table 10. Document Revision History

Document Version	Intel Quartus Prime Version	Changes
2017.05.08	17.0.0	• Reorganized content on topics: Benefits of Command-Line Executables and Project Settings with Command-Line Options. • Removed mentions to unsupported executables and options. • Removed topics: Introductory Example and Common Scripting Examples
2016.10.31	16.1.0	• Implemented Intel rebranding.
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Quartus Prime*.
2015.05.04	15.0.0	Remove descriptions of makefile support that was removed from software in 14.0.
December 2014	14.1.0	Updated DSE II commands.
June 2014	14.0.0	Updated formatting.
November 2013	13.1.0	Removed information about -silnet `qmegawiz` command
June 2012	12.0.0	Removed survey link.
November 2011	11.0.1	Template update.
May 2011	11.0.0	Corrected quartus_qpf example usage. Updated examples.
December 2010	10.1.0	Template update. Added section on using a script to regenerate megafunction variations. Removed references to the Classic Timing Analyzer (quartus_tan). Removed Qflow illustration.
July 2010	10.0.0	Updated script examples to use quartus_sta instead of quartus_tan, and other minor updates throughout document.

continued...

Document Version	Intel Quartus Prime Version	Changes
November 2009	9.1.0	Updated Table 2–1 to add quartus_jli and quartus_jbcc executables and descriptions, and other minor updates throughout document.
March 2009	9.0.0	No change to content.
November 2008	8.1.0	Added the following sections: • "The MegaWizard Plug-In Manager" on page 2–11 "Command-Line Support" on page 2–12 "Module and Wizard Names" on page 2–13 "Ports and Parameters" on page 2–14 "Invalid Configurations" on page 2–15 "Strategies to Determine Port and Parameter Values" on page 2–15 "Optional Files" on page 2–15 "Parameter File" on page 2–16 "Working Directory" on page 2–17 "Variation File Name" on page 2–17 • "Create a Compressed Configuration File" on page 2–21 • Updated "Option Precedence" on page 2–5 to clarify how to control precedence • Corrected Example 2–5 on page 2–8 • Changed Example 2–1, Example 2–2, Example 2–4, and Example 2–7 to use the EP1C12F256C6 device • Minor editorial updates • Updated entire chapter using 8½" × 11" chapter template
May 2008	8.0.0	• Updated "Referenced Documents" on page 2–20. • Updated references in document.

Related Information

Documentation Archive

For previous versions of the *Intel Quartus Prime Handbook*, search the documentation archives.

3. Intel Quartus Prime Pro Edition User Guide Scripting Archives

If the table does not list a software version, the user guide for the previous software version applies.

Intel Quartus Prime Version	User Guide
19.1	Intel Quartus Prime Pro Edition User Guide Scripting
18.1	Intel Quartus Prime Pro Edition User Guide Scripting
18.0	Scripting User Guide Intel Quartus Prime Pro Edition

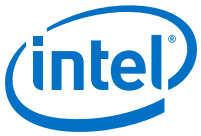
A. Intel Quartus Prime Pro Edition User Guides

Refer to the following user guides for comprehensive information on all phases of the Intel Quartus Prime Pro Edition FPGA design flow.

Related Information

- Intel Quartus Prime Pro Edition User Guide: Getting Started
  Introduces the basic features, files, and design flow of the Intel Quartus Prime Pro Edition software, including managing Intel Quartus Prime Pro Edition projects and IP, initial design planning considerations, and project migration from previous software versions.
- Intel Quartus Prime Pro Edition User Guide: Platform Designer
  Describes creating and optimizing systems using Platform Designer, a system integration tool that simplifies integrating customized IP cores in your project. Platform Designer automatically generates interconnect logic to connect intellectual property (IP) functions and subsystems.
- Intel Quartus Prime Pro Edition User Guide: Design Recommendations
  Describes best design practices for designing FPGAs with the Intel Quartus Prime Pro Edition software. HDL coding styles and synchronous design practices can significantly impact design performance. Following recommended HDL coding styles ensures that Intel Quartus Prime Pro Edition synthesis optimally implements your design in hardware.
- Intel Quartus Prime Pro Edition User Guide: Design Compilation
  Describes set up, running, and optimization for all stages of the Intel Quartus Prime Pro Edition Compiler. The Compiler synthesizes, places, and routes your design before generating a device programming file.
- Intel Quartus Prime Pro Edition User Guide: Design Optimization
  Describes Intel Quartus Prime Pro Edition settings, tools, and techniques that you can use to achieve the highest design performance in Intel FPGAs. Techniques include optimizing the design netlist, addressing critical chains that limit retiming and timing closure, optimizing device resource usage, device floorplanning, and implementing engineering change orders (ECOs).
- Intel Quartus Prime Pro Edition User Guide: Programmer
  Describes operation of the Intel Quartus Prime Pro Edition Programmer, which allows you to configure Intel FPGA devices, and program CPLD and configuration devices, via connection with an Intel FPGA download cable.
- Intel Quartus Prime Pro Edition User Guide: Block-Based Design
  Describes block-based design flows, also known as modular or hierarchical design flows. These advanced flows enable preservation of design blocks (or logic that comprises a hierarchical design instance) within a project, and reuse of design blocks in other projects.

- Intel Quartus Prime Pro Edition User Guide: Partial Reconfiguration
  Describes Partial Reconfiguration, an advanced design flow that allows you to reconfigure a portion of the FPGA dynamically, while the remaining FPGA design continues to function. Define multiple personas for a particular design region, without impacting operation in other areas.
- Intel Quartus Prime Pro Edition User Guide: Third-party Simulation
  Describes RTL- and gate-level design simulation support for third-party simulation tools by Aldec*, Cadence*, Mentor Graphics, and Synopsys that allow you to verify design behavior before device programming. Includes simulator support, simulation flows, and simulating Intel FPGA IP.
- Intel Quartus Prime Pro Edition User Guide: Third-party Synthesis
  Describes support for optional synthesis of your design in third-party synthesis tools by Mentor Graphics, and Synopsys. Includes design flow steps, generated file descriptions, and synthesis guidelines.
- Intel Quartus Prime Pro Edition User Guide: Third-party Logic Equivalence Checking Tools
  Describes support for optional logic equivalence checking (LEC) of your design in third-party LEC tools by OneSpin*. Describes how to verify the logic equivalence between compilation netlists.
- Intel Quartus Prime Pro Edition User Guide: Debug Tools
  Describes a portfolio of Intel Quartus Prime Pro Edition in-system design debugging tools for real-time verification of your design. These tools provide visibility by routing (or "tapping") signals in your design to debugging logic. These tools include System Console, Signal Tap logic analyzer, Transceiver Toolkit, In-System Memory Content Editor, and In-System Sources and Probes Editor.
- Intel Quartus Prime Pro Edition User Guide: Timing Analyzer
  Explains basic static timing analysis principals and use of the Intel Quartus Prime Pro Edition Timing Analyzer, a powerful ASIC-style timing analysis tool that validates the timing performance of all logic in your design using an industry-standard constraint, analysis, and reporting methodology.
- Intel Quartus Prime Pro Edition User Guide: Power Analysis and Optimization
  Describes the Intel Quartus Prime Pro Edition Power Analysis tools that allow accurate estimation of device power consumption. Estimate the power consumption of a device to develop power budgets and design power supplies, voltage regulators, heat sink, and cooling systems.
- Intel Quartus Prime Pro Edition User Guide: Design Constraints
  Describes timing and logic constraints that influence how the Compiler implements your design, such as pin assignments, device options, logic options, and timing constraints. Use the Interface Planner to prototype interface implementations, plan clocks, and quickly define a legal device floorplan. Use the Pin Planner to visualize, modify, and validate all I/O assignments in a graphical representation of the target device.
- Intel Quartus Prime Pro Edition User Guide: PCB Design Tools
  Describes support for optional third-party PCB design tools by Mentor Graphics and Cadence*. Also includes information about signal integrity analysis and simulations with HSPICE and IBIS Models.



- Intel Quartus Prime Pro Edition User Guide: Scripting
  Describes use of Tcl and command line scripts to control the Intel Quartus Prime Pro Edition software and to perform a wide range of functions, such as managing projects, specifying constraints, running compilation or timing analysis, or generating reports.