

Introduction

Developing and running tool command language (Tcl) scripts to control the Altera® Quartus® II software allows you to perform a wide range of functions, such as compiling a design or writing procedures to automate common tasks.

You can automate your Quartus II assignments using Tcl scripts so that you do not have to create them individually. Tcl scripts also facilitate project or assignment migration. For example, when using the same prototype or development board for different projects, you can automate reassignment of pin locations in each new project. The Quartus II software can also generate a Tcl script based on all the current assignments in the project, which aids in migrating assignments to another project. You can use Tcl scripts to manage a Quartus II project, make assignments, define design constraints, make device assignments, run compilations, perform timing analysis, import LogicLock™ region assignments, use the Quartus II Chip Editor, and access reports.

The Quartus II software Tcl commands follow the electronic design automation (EDA) industry Tcl application programming interface (API) standards for using command-line options to specify arguments. This simplifies learning and using Tcl commands. If you encounter an error using a command argument, the Tcl interpreter gives help information showing correct usage.

This chapter includes sample Tcl scripts for the Quartus II software. You can modify these example scripts for use with your own designs.

What is Tcl?

Tcl (pronounced tickle) is a popular scripting language that is similar to many shell scripting and high-level programming languages. It provides support for control structures, variables, network socket access, and APIs. Tcl is the EDA industry-standard scripting language used by Synopsys, Mentor Graphics®, and Synplicity software tools. It allows you to create custom commands and works seamlessly across most development platforms. For a list of recommended literature on Tcl, see [“References” on page 3–29](#).

You can create your own procedures by writing scripts containing basic Tcl commands, user-defined procedures, and Quartus II API functions. You can then automate your design flow, run the Quartus II software in batch mode, or execute the individual Tcl commands interactively in the Quartus II Tcl interactive shell.

The Quartus II software version 4.0 supports Tcl/Tk version 8.4, supplied by the Tcl DeveloperXchange at <http://tcl.activestate.com>.

Quartus II Command-Line Executables Supporting Tcl

The Quartus II software version 4.0 command-line executables have reduced runtime memory requirements and provide scripting capability with programs such as **make** and **perl**, and UNIX shells.

The following Quartus II software version 4.0 command-line executables support the Tcl interface:

- **quartus_sh.exe**
- **quartus_tan.exe**
- **quartus_cdb.exe**
- **quartus_sim.exe**

quartus_sh

The **quartus_sh** executable is a Quartus II Tcl interpreter, also known as a shell. The shell may be used as an interactive shell, or as a quick Tcl command evaluator, evaluating the command-line arguments as one (or more) Tcl commands.

This simple Tcl scripting shell allows for easy project assignments and running some basic compiler operations.

quartus_tan

The Quartus II Timing Analyzer (**quartus_tan**) computes delays for the given design and device and annotates them onto the netlist for subsequent use by the Simulator. After delays have been measured, the **quartus_tan** executable's output allows you to analyze the performance of all logic in your design.

quartus_cdb

The Quartus II Compiler Database (**quartus_cdb**) interface allows you to access and modify a project design database from the system command prompt. This is primarily used for back-annotating resources and importing LogicLock assignments.

quartus_sim

The Quartus II Simulator (**quartus_sim**) allows you to simulate a design with a Tcl testbench to verify timing and functionality.



For additional information on these and other Quartus II command-line executables, see the *Command-Line Scripting* chapter in Volume 2 of the *Quartus II Handbook*.

Using Tcl with the Quartus II Command-line Executables

The Quartus II command-line executables give the Quartus II design flow added flexibility by allowing you to use shell and Tcl scripting to automate the Quartus II software. In conjunction with the Quartus II command-line executables, Tcl can be used in three ways:

- Interactively using the Quartus II Tcl shell (**quartus_sh**)
- As a batch file from the system command prompt
- Directly from the system command prompt

Using the Interactive Quartus II Tcl Scripting Shell

Typing the name of a Quartus II command-line executable followed by the `-s` or `--shell` command-line option starts an interactive Tcl shell session that displays `tcl>` prompt. Everything typed in the Tcl shell is directly interpreted by the Quartus II Tcl interpreter. If a command is not recognized by the Tcl shell, it is assumed to be an external command and executed as a system call using Tcl's `exec` command.

The initial output of the Tcl shell is shown below (including the command that starts the shell):

```
C:\>quartus_sh -s
Info: *****
Info: Running Quartus II Shell
Info: Version 4.0 Internal Build 131 10/06/2003 SJ Full Version
Info: Copyright (C) 1991-2003 Altera Corporation. All rights reserved.
Info: Quartus is a registered trademark of Altera Corporation in the
Info: US and other countries. Portions of the Quartus II software
Info: code, and other portions of the code included in this download
Info: or on this CD, are licensed to Altera Corporation and are the
Info: copyrighted property of third parties who may include, without
Info: limitation, Sun Microsystems, The Regents of the University of
Info: California, Softel vdm., and Verific Design Automation Inc.
Info: Warning: This computer program is protected by copyright law
Info: and international treaties. Unauthorized reproduction or
Info: distribution of this program, or any portion of it, may result
Info: in severe civil and criminal penalties, and will be prosecuted
Info: to the maximum extent possible under the law.
Info: Processing started: Sat Jan 10 19:37:42 2004
Info: *****
Info: The Quartus II Shell supports all TCL commands in addition
```

```
Info: to Quartus II Tcl commands. All unrecognized commands are
Info: assumed to be external and are run using Tcl's "exec"
Info: command.
Info: - Type "exit" to exit.
Info: - Type "help" to view a list of Quartus II Tcl packages.
Info: - Type "help -pkg <package name>" to view a list of Tcl commands
Info: available for the specified Quartus II Tcl package.
Info: - Type "help -tcl" to get an overview on Quartus II Tcl usages.
Info: *****
tcl>
```



For more information on creating and compiling a project using Tcl in an interactive shell, see [“Using the Quartus II Tcl Shell in Interactive Mode” on page 3–24.](#)

Using Scripts in Batch From a Shell

Once you create a Tcl script file (.tcl), you can run it by typing the following command in a Tcl shell:

```
source <script_name>.tcl <argument list> ←
```

This runs the Tcl script (contained in the file). Arguments can be read using the `quartus(args)` predefined global variable. See [“Using the Pre-Defined Global Quartus II Tcl Array” on page 3–11](#) for more information. The examples in this chapter are written to be run as batch files.

If you already have a Quartus II version 4.0-compatible Tcl file you can run it at the command line prompt by using the `-t` option. For example, typing `quartus_sh -t <script_name>.tcl` ← runs the Quartus II Tcl shell and uses the Tcl file specified by the `-t` option as the input Tcl script. The Quartus II executable (using the Quartus II Tcl shell) processes the Tcl script file.

Using Tcl Directly from the Command Prompt (--tcl_eval)

You can also access Tcl using Quartus II command-line executables with the `--tcl_eval` option. Using the `--tcl_eval` option causes the Quartus II Tcl shell to directly evaluate the remaining command-line arguments as Tcl commands (and their arguments, if possible). Use semicolons to separate multiple Tcl commands. For example the following code causes the Quartus II software to display “Hello” and “World” on two separate lines:

```
quartus_sh --tcl_eval puts Hello; puts World ←
```

The Tcl evaluation option allows external scripting programs (such as **make**, **perl**, and **sh**) to access information from the Quartus II software. This functionality can be used, for example, to obtain device family information for a targeted part. You can also use the `--tcl_eval` option to get Tcl help information directly from the command-line prompt.

Using the Quartus II Tcl Console Window

You can execute Tcl commands directly in the **Quartus II Tcl Console** window. To open the window, choose **Utility Windows > Tcl Console** (View menu). The Tcl Console is usually docked on the bottom-right corner of Quartus II graphic user interface (GUI). Everything typed in the Tcl Console is interpreted by the Quartus II Tcl shell.



The **Quartus II Tcl Console** window supports the pre-Quartus II software version 3.0 Tcl API for backward compatibility with older designs and EDA tools.

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

Tcl Packages

The Quartus II software version 4.0 Tcl commands are grouped by similarity of function into Tcl packages. For example, one Tcl package contains commands for making and getting assignments, and another package is used to access device family information. Only the appropriate packages are made available to each Quartus II executable. For example, the timing analysis Tcl packages are only available with the **quartus_tan** executable.

By default only the minimum number of packages are loaded with each Quartus II executable. This keeps the memory requirements for each executable as small as possible. This also means that some Tcl commands and help information are not available until you load the required Tcl package.

The available packages address the following functions:

- Project and assignment
- Device
- Advanced device
- Flow
- Timing
- Advanced timing
- Simulator
- Report
- Timing report
- Back annotate

- LogicLock
- Chip Editor
- Atom netlist
- Miscellaneous

Loading a Tcl Package

To load a Tcl package, use the following command:

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

For example, to load the `::quartus::flow` package, type the following command:

```
load_package flow ←
```

For more information about the `load_package` command, type `load_package -help` at a Quartus II Tcl shell prompt.

Packages Loaded for Each Command-Line Executable

To quickly check what packages are loaded by default, run the executable in question with the `--tcl_eval help` option. For example, invoking `quartus_sh` using `--tcl_eval help` produces the following output:

```
C:\>quartus_sh --tcl_eval help
-----
Available Quartus II Tcl Packages:
-----
Loaded                Not Loaded
-----
::quartus::device    ::quartus::flow
::quartus::misc      ::quartus::report
::quartus::project

* Type "help -tcl"
  to get an overview on Quartus II Tcl usages.

* Type "help -pkg <package name>"
  to view a list of Tcl commands available for
  the specified Quartus II Tcl package.
-----
c:\>
```

Type `help` at the Tcl prompt to display the current Tcl packages loaded for the Quartus II executable.

Table 3–1 lists the Quartus II Tcl packages available with each Quartus II executable and whether each package is loaded by default or available to be loaded. A blank space in the table means that the package is not available for that executable.

Packages	Quartus II Executable				
	Quartus_sh	Quartus_tan	Quartus_cdb	Quartus_sim	Tcl Colsole
advanced_device	Not Loaded	Not Loaded	Not Loaded		
advanced_timing		Not Loaded			
backannotate			Not Loaded		Not Loaded
chip_editor			Not Loaded		
device	Loaded	Not Loaded	Loaded	Loaded	Not Loaded
flow	Not Loaded	Not Loaded	Not Loaded		Not Loaded
logiclock		Not Loaded	Not Loaded		Not Loaded
misc	Loaded	Loaded	Loaded	Loaded	Loaded
project	Loaded	Loaded	Loaded	Loaded	Loaded
report	Not Loaded	Not Loaded	Not Loaded	Loaded	Not Loaded
simulator				Loaded	
timing		Loaded			
timing_report		Not Loaded			Loaded
old_api					Loaded
atoms			Not Loaded		

Getting Help on Tcl & Quartus II Tcl APIs

Quartus II Tcl help allows easy access to information on the Quartus II Tcl commands. To access the help information, type `help` at a command prompt, as shown below (with sample output):

```
tcl> help
-----
Available Quartus II Tcl Packages:
-----
Loaded                Not Loaded
-----
::quartus::device    ::quartus::flow
::quartus::misc      ::quartus::report
::quartus::project
```

* Type "help -tcl" to get an overview on Quartus II Tcl usages.

tcl>

Using the `-tcl` option with `help` displays an introduction to the Quartus II Tcl API that focuses on how to get help for Tcl commands (short help and long help) and Tcl packages.

Table 3–2 summarizes the help options available in the Tcl environment.

Table 3–2. Help Options Available in the Quartus II Tcl Environment (Part 1 of 2)	
Help Command	Description
<code>help</code>	To view a list of available Quartus II Tcl packages, loaded and not loaded.
<code>help -tcl</code>	To view a list of commands used to load Tcl packages and access command-line help.
<code>help -pkg <package_name> [-version <version number>]</code>	To view help for a specified Quartus II package that includes the list of available Tcl commands. For convenience, you can omit the <code>::quartus::</code> package prefix, and type <code>help -pkg <package name></code> ←. If you do not specify the <code>-version</code> option, help for the currently loaded package is displayed by default. If the package for which you want help is not loaded, help for the latest version of the package is displayed by default. Examples: <code>help -pkg ::quartus::project</code> ← <code>help -pkg project</code> ← <code>help -pkg project -version 1.0</code> ←
<code><command_name> -h</code> or <code><command_name> -help</code>	To view short help for a Quartus II Tcl command for which the package is loaded. Examples: <code>project_open -h</code> ← <code>project_open -help</code> ←

Table 3–2. Help Options Available in the Quartus II Tcl Environment (Part 2 of 2)

Help Command	Description
<pre>package require ::quartus::<i><package name></i> [<i><version></i>]</pre>	<p>To load a Quartus II Tcl package with the specified version. If <i><version></i> is not specified, the latest version of the package is loaded by default.</p> <p>Example: <pre>package require ::quartus::project 1.0 ↵</pre></p> <p>This command is similar to the <code>load_package</code> command. The advantage of using <code>load_package</code> is that you can alternate freely between different versions of the same package.</p> <pre>Type <package name> [-version <version number>] ↵</pre> <p>to load a Quartus II Tcl package with the specified version. If the "-version" option is not specified, the latest version of the package is loaded by default.</p> <p>Example: <pre>load_package ::quartus::project -version 1.0 ↵</pre></p>
<pre>help -cmd <i><command name></i> [-version <i><version number></i>] or <i><command name></i> -long_help</pre>	<p>To view long help for a Quartus II Tcl command. Only "<i><command name></i> -long_help" requires that the associated Tcl package is loaded. If you do not specify the "-version" option, help for the currently loaded package is displayed by default. If the package for which you want help is not loaded, help for the latest version of the package is displayed by default.</p> <p>Examples: <pre>project_open -long_help ↵ help -cmd project_open ↵ help -cmd project_open -version 1.0 ↵</pre></p>
<pre>help -examples</pre>	<p>To view examples of Quartus II Tcl usage.</p>
<pre>help -quartus</pre>	<p>To view help on the predefined global Tcl array that can be accessed to view information about the Quartus II executable that is currently running.</p>
<pre>quartus_sh --qhelp</pre>	<p>To launch the Tk viewer for Quartus II command-line help and display help for the command-line executables and Tcl API packages. See "The Tcl/Tk GUI Help Interface" on page 3–10 for more information.</p>

There are two types of help for Tcl commands:

- For information on the usage and a brief description of a Tcl command type, use the `-help` option. (The `-h` command-line option may be used instead of `-help`, if preferred.) If the Tcl command is part of a Tcl package that is not loaded, using the `-help` option returns “invalid command name” as an error message.
- For more detailed help on a given Tcl command, use the `-long_help` option or type `help -cmd <Tcl command name>`. If the Tcl command is part of a Tcl package that is not loaded, typing `<command name> -long_help` returns the error message “invalid command name.”



Using the `-cmd` option does not require that the specific Tcl command be loaded. Only the `-long_help` option requires previously loading the relevant Tcl package.

The Tcl/Tk GUI Help Interface

For a complete list of package and commands available with the Quartus II software, launch a help window listing all Quartus II command-line executables and Tcl API packages and their respective commands. To launch the help window, type the following command at a system command prompt:

```
C:\> quartus_sh --qhelp
```

This starts a Tcl/Tk script that provides help for Quartus II Command-line executables and Tcl API packages and commands.



For more information on this utility, see the *Command-Line Scripting* chapter in Volume 2 of the *Quartus II Handbook*.

Running Quartus II Executables from Tcl Scripts

You can run the Quartus II command-line executables from Tcl scripts either with the included `::quartus::flow` package to run the entire Quartus II compilation flow, or by running each executable directly.

The `::quartus::flow` Package

The `::quartus::flow` package includes two commands for running Quartus II command-line executables, either individually or together in standard compilation sequence. The `execute_module` command allows you to run an individual Quartus II command-line executable. The `execute_flow` command allows you to run some or all of the modules in commonly-used combinations.

For an example of a script using `execute_module` to compile a Quartus II project, see [“Importing LogicLock Functions”](#) on page 3–20.

For an example of a script using `execute_flow` to compile a Quartus II project, see [“Creating a Simple Project” on page 3–12](#).

Another way to run a Quartus II command-line executable from the Tcl environment is by using the `qexec Tcl` command, a Quartus II implementation of Tcl's `exec` command. For example, to run the Quartus II technology mapper on a given project, type:

```
qexec "quartus_map <project_name>" ←
```

When you use the `qexec` command to compile a design, assignments made in the Tcl script (or from the Tcl shell) are not exported to the project database and settings file before compilation. Use the `export_assignments` command or compile the project using commands in the `::quartus::flow` package to ensure assignments are exported to the project database and settings file.



You can also use the Tcl `exec` command to perform command-line system calls. However, Altera recommends using the `qexec` command to avoid limitations with Tcl version 8.3. Whether using `exec` or `qexec`, use caution when making system calls.

Altera recommends using the `::quartus::flow` package instead of using system calls to run compiler executables. However, advanced users may choose to use a Tcl script, running each Quartus II executable with a `qexec` command for the additional capabilities provided by command-line options and increased control over execution of each module.

You can also run executables directly in a Tcl shell, using the same syntax as at the system command prompt. For example, to run the Quartus II technology mapper on a given project, type the following at the Tcl shell prompt:

```
quartus_map <project_name> ←
```

Using the Pre-Defined Global Quartus II Tcl Array

The predefined global Tcl array `quartus` contains information about the Quartus II command-line executables. The array includes information about the arguments passed from the command prompt, the version of the Quartus II executable, the location of the Quartus II executable, and the current project name, as well as other related data.



For more information on the Quartus II Tcl array, type `help -quartus` in a Tcl shell.

Accessing Command-Line Arguments in Quartus II Tcl Scripts

The Quartus II software stores command-line arguments in the array `quartus (args)` rather than the `argv` used in standard Tcl scripts. For example, to pass the name of a project to a Tcl script, the syntax is:

```
quartus_tan -t script.tcl <projectname> <frequency> <device>←
```

When executing the above command, *<projectname>* is stored in index 0, *<frequency>* is stored in index one, and *<device>* is stored in index two of `quartus (args)`.

Tcl Examples

The examples in this section demonstrate Tcl scripts for performing the following tasks:

- Create a project
- Select a device family and a specific member of that family
- Compile the design

Creating a Simple Project

The following example is a Tcl script for the **chiptrip** example project in the Quartus II tutorial.

```
# This script uses the quartus_sh command line
executable
# and compiles the chiptrip design
# This script assumes that the project and revision
# have the same name
# Using simple global assignments
# Set the project_name variable to chiptrip
set project_name chiptrip

# Open the project or
# create the project if it's not already created
if [project_exists $project_name] {
    project_open $project_name
} else {
    project_new $project_name
}

#----- Make global assignments -----#
set_global_assignment -name family STRATIX
set_global_assignment -name device EP1S10F484C5

# The Quartus flow package is loaded
load_package flow

execute_flow -compile

project_close
```

Creating a Project Including a Different Revision Name

The following script creates a project `fir_filter` that includes a revision named `filtref`.

Most Quartus II projects have one revision, with the same name as the project. The example shown below creates a revision name that is different from the project name. This technique is used when there are multiple revisions for a project. The `set_current_revision` Tcl command allows you to switch project revisions in a Tcl file.

```
# This script uses the quartus_sh command line executable
# Set project name to fir_filter
set project_name fir_filter
# create revision name
set revision_name filtref

# Create a new project and open it
# Project_name is project name
# Require package ::quartus::project
if {![project_exists $project_name]} {
    project_new -revision $revision_name $project_name;
} else {
    project_open -revision $revision_name $project_name;
}

# setting Fmax timing requirement
set_global_assignment -name "FMAX_REQUIREMENT" "45.0 MHz"

# Compiler Assignments for filtref
set_current_revision filtref

set_global_assignment -name "FAMILY" "APEX20KE"
set_global_assignment -name "DEVICE" "EP20K100EQC208-1"

# The Quartus flow package is loaded
load_package flow

execute_flow -compile

project_close
```

Assignments

Tcl scripts are a convenient way to move assignments between projects that have assignments in common, such as pin and location assignments. In this case, a Tcl script can contain a series of assignments using the following Tcl commands from the `::quartus::project` package:

```
set_global_assignment
set_instance_assignment
set_location_assignment
set_parameter
```

To bring the assignments into a project, source the Tcl script either by specifying the Tcl file as a batch script at the command line using the (-t option) or reading the Tcl file into a project during an interactive Tcl shell session.



The assignments created or modified during an open project are not committed to the Quartus II settings files unless you explicitly call `export_assignments` or `project_close` (unless `-dont_export_assignments` is specified). In some cases, as when running `execute_flow`, the Quartus II software automatically commits the changes.

The following example script is a compilation script for the finite impulse response (FIR) filter example project used and developed in the Quartus II tutorial. It shows how to set global, location, and instance assignments for a project followed by a complete project compilation using the `::quartus::flow` package.

```
# This Tcl file works with quartus_sh.exe
# This Tcl file will compile the Quartus II tutorial fir_filter
# design
# set the project_name to fir_filter
# set revision to filtref
set project_name fir_filter
set revision_name filtref

# Create a new project and open it
# Project_name is project name
# Require package ::quartus::project
if (![project_exists $project_name]) {
    project_new -revision $revision_name $project_name;
} else {
    project_open -revision $revision_name $project_name;
}

#----- Make global assignments -----#

# add design files to project
# When the revision name is the same as the project name
# adding design files can be skipped
#set_global_assignment -name "BDF_FILE" "filtref.bdf"
#set_global_assignment -name "VERILOG_FILE" "acc.v"
#set_global_assignment -name "VERILOG_FILE" "accum.v"
#set_global_assignment -name "VERILOG_FILE" "hvalues.v"
#set_global_assignment -name "VERILOG_FILE" "mult.v"
#set_global_assignment -name "VERILOG_FILE" "state_m.v"
#set_global_assignment -name "VERILOG_FILE" "taps.v"

set_global_assignment -name FAMILY APEX20KE
set_global_assignment -name DEVICE EP20K100EQC208-1

#----- Make instance assignments -----#
```

```

# assign module taps:inst to a location
# These two commands do the same assignment location:
# set_instance_assignment -name LOCATION -to taps:inst MegaLAB_A1
set_location_assignment -to taps:inst MegaLAB_A1

# set an instance assignment to an entity
set_instance_assignment -name APEX20K_TECHNOLOGY_MAPPER -to \
state_m:inst1 "Product Term"

#----- project compilation -----#

# The project is compiled here to see ESB placement following
# what is done in the tutorial
load_package flow
execute_flow -compile

project_close

```

Timing Analysis

The following example script uses the **quartus_tan** executable to perform a timing analysis on the `fir_filter` tutorial design.

The `fir_filter` design is a two-clock design that requires a base clock and a relative clock relationship for timing analysis. This script first does an analysis of the two-clock relationship and checks for t_{SU} slack between `clk` and `clkx2`. The first pass of the timing analysis discovers a negative slack for one of the clocks. The second part of the script adds a multicycle assignment from `clk` to `clkx2` and re-analyzes the design as a multi-clock, multicycle design.

The script does not recompile the design with the new timing assignments, and timing-driven compilation is not used in the synthesis and placement of this design. New timing assignments are added only for the timing analyzer to analyze the design by using the `create_timing_netlist` and `report_timing` Tcl commands.



You must compile the project before running the script example below.

```

# This Tcl file is to be used with quartus_tan.exe
# This Tcl file will do the Quartus II tutorial fir_filter design
# timing analysis portion by making a global timing assignment and
# creating multi-clock assignments and run timing analysis
# for a multi-clock, multi-cycle design

# set the project_name to fir_filter
# set the revision_name to filtref
set project_name fir_filter
set revision_name filtref

# open the project
# project_name is the project name
project_open -revision $revision_name $project_name;

```

```
# Doing TAN tutorial steps this section re-runs the timing
# analysis module with multi-clock and multi-cycle settings
#----- Make timing assignments -----#

#Specifying a global FMAX requirement (tan tutorial)
set_global_assignment -name FMAX_REQUIREMENT 45.0MHZ
set_global_assignment -name CUT_OFF_IO_PIN_FEEDBACK ON

# create a base reference clock "clocka" and specifies the
# following:
#   BASED_ON_CLOCK_SETTINGS = clocka;
#   INCLUDE_EXTERNAL_PIN_DELAYS_IN_FMAX_CALCULATIONS = OFF;
#   FMAX_REQUIREMENT = 50MHZ;
#   DUTY_CYCLE = 50;
# Assigns the reference clocka to the pin "clk"
create_base_clock -fmax 50MHZ -duty_cycle 50 clocka -target clk

# creates a relative clock "clockb" based on reference clock
# "clocka" with the following settings:
#   BASED_ON_CLOCK_SETTINGS = clocka;
#   MULTIPLY_BASE_CLOCK_PERIOD_BY = 1;
#   DIVIDE_BASE_CLOCK_PERIOD_BY = 2;clock period is half the base
#   clk
#   DUTY_CYCLE = 50;
#   OFFSET_FROM_BASE_CLOCK = 500ps;The offset is .5 ns (or 500 ps)
#   INVERT_BASE_CLOCK = OFF;
# Assigns the reference clock to pin "clkx2"
create_relative_clock -base_clock clocka -duty_cycle 50\
-divide 2 -offset 500ps -target clkx2 clockb

# create new timing netlist based on new timing settings
create_timing_netlist

# does an analysis for clkx2
# Limits path listing to 1 path
# Does clock setup analysis for clkx2
report_timing -npaths 1 -clock_setup -file tsu_multiclock.tao

# The output file will show a negative slack for clkx2 when only
# specifying a multi-clock design. The negative slack was created
# by the 500 ps offset from the base clock

# removes old timing netlist to allow for creation of a new timing
# netlist for analysis by report_timing
delete_timing_netlist

# adding a multi-cycle setting corrects the negative slack by
# adding a multicycle assignment to clkx2 to allow for more
# set-up time
set_multicycle_assignment 2 -from clk -to clkx2

# create a new timing netlist based on additional timing
# assignments
create_timing_netlist

# outputs the result to a file for clkx2 only
report_timing -npaths 1 -clock_setup -clock_filter clkx2 \
-file tsu_multicycle.tao
```

```
# The new output file will show a positive slack for the clkx2
project_close
```

EDA Tool Assignments

You can target EDA tools for a project in the Quartus II software in Tcl by using the `set_global_assignment` Tcl command. To use the default tool settings for each EDA tool, you need only specify the EDA tool to be used. The EDA interfaces available for the Quartus II software cover design entry, simulation, timing analysis and board design tools. More advanced EDA tools such as those for formal verification and resynthesis are supported by their own global assignment.

The global options used for each EDA interface in the Quartus II software are:

- EDA_DESIGN_ENTRY_SYNTHESIS_TOOL
- EDA_SIMULATION_TOOL
- EDA_TIMING_ANALYSIS_TOOL
- EDA_BOARD_DESIGN_TOOL
- EDA_FORMAL_VERIFICATION_TOOL
- EDA_RESYNTHESIS_TOOL

By default, these project options are set to <none>. [Table 3–3](#) lists the EDA interface options available in the Quartus II software. Enclose interface assignment options that contain spaces in quotes.

Option	Acceptable Values
Design Entry (EDA_DESIGN_ENTRY_SYNTHESIS_TOOL)	Design Architect Design Compiler FPGA Compiler FPGA Compiler II FPGA Compiler II Altera Edition FPGA Express LeonardoSpectrum LeonardoSpectrum-Altera (Level 1) Synplify Synplify Pro ViewDraw Precision Synthesis Custom
Simulation (EDA_SIMULATION_TOOL)	ModelSim (VHDL output from Quartus II) ModelSim (Verilog HDL output from Quartus II) ModelSim-Altera (VHDL output from Quartus II) ModelSim-Altera (Verilog HDL output from Quartus II) SpeedWave VCS Verilog-XL VSS NC-Verilog (Verilog HDL output from Quartus II) NC-VHDL (VHDL output from Quartus II) Scirocco (VHDL output from Quartus II) Custom Verilog HDL Custom VHDL
Timing Analysis (EDA_TIMING_ANALYSIS_TOOL)	Prime Time (VHDL output from Quartus II) Prime Time (Verilog HDL output from Quartus II) Stamp (board model) Custom Verilog HDL Custom VHDL
Board level tools (EDA_BOARD_DESIGN_TOOL)	Signal Integrity (IBIS) Symbol Generation (ViewDraw)
Formal Verification (EDA_FORMAL_VERIFICATION_TOOL)	Conformal LEC
Resynthesis (EDA_RESYNTHESIS_TOOL)	PALACE Amplify

For example, to generate NC-Sim Verilog simulation output, EDA_SIMULATION_TOOL should be set to target NC-Sim Verilog as the desired output, as shown below:

```
set_global_assignment -name eda_simulation_tool \
"NcSim (Verilog HDL output from Quartus II)"
```

The following example shows compilation of the `fir_filter` design files, generating a VHO file output for NC-Sim Verilog simulation:

```
# This script works with the quartus_sh executable
# Set the project name to filtref
set project_name filtref

# Open the Project. If it does not already exist, create it
if [catch {project_open $project_name}] {project_new \
$project_name}

# Set Family
set_global_assignment -name family APEX 20KE

# Set Device
set_global_assignment -name device ep20k100eqc208-1

# Optimize for speed
set_global_assignment -name optimization_technique speed

# Turn-on Fastfit fitter option to reduce compile times
set_global_assignment -name fast_fit_compilation on

# Generate a NC-Sim Verilog simulation Netlist
set_global_assignment -name eda_simulation_tool "NcSim\
(Verilog HDL output from Quartus II)"

# Create an FMAX=50MHz assignment called clk1 to pin clk
create_base_clock -fmax 50MHz -target clk clk1

# Create a pin assignment on pin clk
set_location -to clk Pin_134

# Compilation option 1
# Always write the assignments to the constraint files before
# doing a system call. Else, stand-alone files will not pick up
# the assignments
#export_assignments
#qexec quartus_map <project_name>
#qexec quartus_fit <project_name>
#qexec quartus_asm <project_name>
#qexec quartus_tan <project_name>
#qexec quartus_eda <project_name>

# Compilation option 2 (better)
# Using the ::quartus::flow package, and execute_flow command will
# export_assignments automatically and be equivalent to the steps
# outlined in compilation option 1
load_package flow
```

```
execute_flow -compile

# Close Project
project_close
```

There are custom options available to target other EDA tools. For custom EDA configurations, you can change the individual EDA interface options by making additional assignments.



For a complete list of each EDA setting line available, see “EDA Tool Setting Section (Settings and Configuration Files)” in Quartus II Help.

Importing LogicLock Functions

The following Tcl script shows how a LogicLock function can be imported into a project. This example is based on the LogicLock tutorial design **topmult**. The script assumes that the Verilog Quartus Mapping file (**.vqm**) named **pipemult.vqm** and the Quartus II Setting File named **pipemult.qsf** have been generated already and placed in the **topmult** project directory. The `get_fmax_from_report` procedure was added to display the result on the design f_{MAX} of importing a LogicLock region. To import LogicLock regions into a project, the **quartus_cdb** executable must be used.

```
# Tcl file created for quartus_cdb to import LogicLock
# pipemult.vqm and pipemult.qsf into the topmult project
# This Tcl script assumes that pipemult.vqm and pipemult.qsf
# have been generated in the lockmult project.

# Since ::quartus::flow is not pre-loaded
# by quartus_cdb, load this package now
# before using the flow Tcl API
# Type "help -pkg flow" to view information
# about the package
load_package flow

proc get_fmax_from_report {} {

    global project_name

    load_package report

    # Load the project report database
    load_report $project_name

    # Find the "Timing Analyzer Summary" panel name containing
    # the Actual Fmax data by traversing the panel names
    # Then set the panel row containing the Actual Fmax
    # information
    set fmax_panel_name "Timing Analyzer Summary"
    foreach panel_name [get_report_panel_names] {
        if { [string match "$fmax_panel_name" "$panel_name"] } {
            {
```

```

        # Fmax is sorted so we just need to go to Row 1
        set fmax_row [get_report_panel_row "$panel_name"\
        -row 1]
    }
}

# Actual Fmax is found on the fourth column
# Index starts at 0
set actual_fmax [lindex $fmax_row 3]

# Now unload the project report database
unload_report $project_name

return $actual_fmax
}

set required_fmax 150.00MHz

set project_name topmult

# $project_name contains the project
# name, in this case fir_filter
# Require package ::quartus::project
load_package project

project_open $project_name

#----- Make global assignments -----#

# remove bdf file from project
set_global_assignment -name "BDF_FILE" "pipemult.bdf" -remove
# add VQM file to project
set_global_assignment -name "VQM_FILE" "pipemult.vqm"

# analyze design with VQM file
execute_module -tool map

# import LogicLock constraints
load_package logiclock
initialize_logiclock

# imports the pipemult.qsf file to the project topmult.qsf
logiclock_import -no_pins

uninitialize_logiclock

# compile entire design
execute_flow -compile

#----- Report Fmax from report -----#
set actual_fmax [get_fmax_from_report]
puts ""
puts "-----"
puts "Required Fmax: $required_fmax Actual Fmax: $actual_fmax"
puts "-----"

project_close

```



For additional information on the LogicLock design methodology, see the *LogicLock Design Methodology* chapter in Volume 2 of the *Quartus II Handbook*.

The Quartus II Chip Editor

Tcl support for Quartus II Chip Editor is primarily used to facilitate Engineering Change Orders (ECO) for a project compiled using the Quartus II command-line executables. From the Quartus II Chip Editor, any ECO changes can be exported as a Tcl file to be used to read back into the project when the project is compiled using the command-line executables. The Quartus II Compiler Database executable **quartus_cdb** uses the ECO Tcl file.

After you generate a Tcl file from the Chip Editor, you can re-import the ECO changes without having to use the Chip Editor. The Quartus II Compiler Database executable must be run to import the ECO changes to the project database using the following command:

```
quartus_cdb -t eco.tcl
```

Once the ECO changes are imported into the project database, run the Quartus II Assembler (**quartus_asm**) to generate the new configuration file and run the Quartus II Timing Analyzer (**quartus_tan**) to extract a new timing netlist. To import an ECO-generated Tcl file, insert the following commands in your Tcl script:

```
gexec "quartus_cdb -t <chipeditor_eco_tcl_output>.tcl"  
gexec "quartus_tan <project_name>"  
gexec "quartus_asm <project_name>"
```



For additional information on using the Quartus II Chip Editor, see the *Design Analysis & Engineering Change Management with Chip Editor* chapter in Volume 3 of the *Quartus II Handbook*.

Using the `foreach_in_collection` Command

Some Tcl functions can return very large sets of elements in a list. To minimize the total amount of memory need for large designs, the Quartus II Tcl API includes a `foreach_in_collection` command. The `foreach_in_collection` command is similar to the `foreach` Tcl command, except that it uses internal Quartus II data structures so that a large amount of data does not have to be passed to the Tcl interpreter.

Some commonly used Tcl commands and packages that return a collection are:

- `get_all_global_assignments`

- `get_all_instance_assignments`
- `get_all_parameters`
- `get_timing_nodes`
- `::quartus::advanced_timing` Tcl package
- `::quartus::project` Tcl package

For example, the `get_all_instance_assignments` Tcl command returns a large amount of data for a fully back-annotated device. Rather than passing the data as a large string to the Tcl interpreter, the `get_all_instance_assignments` command returns a collection ID that can be used with `foreach_in_collection` to iterate through all members of the data set.

If you type the following in the Quartus II Tcl shell (on an open project) the Tcl shell returns a value of `_colxx` (as shown), where `xx` is a sequential number that represents a collection.

```
tcl> get_all_instance_assignments -name *  
_col0
```



For additional details and format information on the `get_all_instance_assignments` collection list generated, see the command-line long help information by typing `get_all_instance_assignments -long_help`.

The following example Tcl script shows how `foreach_in_collection` is used to display a project's instance assignment information.

```
# This script is to be used with quartus_sh  
# This script will run through the compiled fir filter  
# tutorial and print out any instance assignments  
# made in the project  
  
# opens existing project  
project_open -revision filtref fir_filter  
  
# creates a collection  
set collection_of_instance_assignments \  
[get_all_instance_assignments -name *]  
  
# uses the foreach_in_collection to iterate collection  
# The instance assignment output is a list with  
# following format:  
# { {<Section Id>} {<Source>} {<Destination>}  
# {<Assignment name>} {<Assignment value>} }  
# We will only be displaying the Destination,  
# Assignment name and value  
  
foreach_in_collection instance \  
$collection_of_instance_assignments {  
  set dest [lindex $instance 2]
```

```

    set name [lindex $instance 3]
    set value [lindex $instance 4]
    puts "Assignment Instance to ($dest) $name = $value "
  }
project_close

```

The following example shows how to use the `::quartus::advanced_timing` package in `quartus_tan` to extract register information for each timing node using the output from the `get_timing_node` Tcl command:

```

# This script is to be used with quartus_tan
# This script will run through a project and print out all
# the registers in the project and its node names.

project_open fir_filter -revision filtref

# generate the timing netlist
create_timing_netlist

# load the advanced timing package
load_package advanced_timing

# Create a collection of all timing nodes of type reg
# Set this collection to the variable node and start an
# iterative loop through this collection
foreach_in_collection node [get_timing_nodes -type reg] {
    puts "[get_timing_node_info -info type $node] node \
        name is [get_timing_node_info -info name $node]"
}
project_close

```

For more usage examples, type `foreach_in_collection -long_help` in a Tcl shell.

Using the Quartus II Tcl Shell in Interactive Mode

This section presents an example of using the `quartus_sh` interactive shell to make some project assignments and compile the FIR filter tutorial project. This example assumes that you already have the FIR filter tutorial design files in a project directory.

To begin, launch the interactive Tcl shell. The command and initial output are shown below:

```

C:\>quartus_sh -s
Info:*****
Info: Running Quartus II Shell
Info: Version 4.0 Internal Build 131 10/06/2003 SJ Full Version
Info: Copyright (C) 1991-2003 Altera Corporation. All rights reserved.
Info: Quartus is a registered trademark of Altera Corporation in the
Info: US and other countries. Portions of the Quartus II software
Info: code, and other portions of the code included in this download
Info: or on this CD, are licensed to Altera Corporation and are the
Info: copyrighted property of third parties who may include, without

```

```

Info: limitation, Sun Microsystems, The Regents of the University of
Info: California, Softel vdm., and Verific Design Automation Inc.
Info: Warning: This computer program is protected by copyright law
Info: and international treaties. Unauthorized reproduction or
Info: distribution of this program, or any portion of it, may result
Info: in severe civil and criminal penalties, and will be prosecuted
Info: to the maximum extent possible under the law.
Info: Processing started: Thu Nov 20 19:54:12 2003
Info:*****
Info: The Quartus II Shell supports all TCL commands in addition
Info: to Quartus II Tcl commands. All unrecognized commands are
Info: assumed to be external and are run using Tcl's "exec"
Info: command.
Info: - Type "exit" to exit.
Info: - Type "help" to view a list of Quartus II Tcl packages.
Info: - Type "help -pkg <package name>" to view a list of Tcl commands
Info: available for the specified Quartus II Tcl package.
Info: - Type "help -tcl" to get an overview on Quartus II Tcl usages.
Info:*****
tcl>

```

At the Tcl prompt, create a new project called **fir_filter** with a revision name called **filtref** by typing the following:

```
tcl> project_new -revision filtref fir_filter
```



If the project file and project name are the same, the Quartus II software gives the revision the same name as the project.

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

Next, set global assignments for the chip family and device target with the following commands:

```
tcl> set_global_assignment -name family apex20ke
tcl> set_global_assignment -name device EP20K100EQC208-1
```



To learn more about assignment names that can be used with the `-name` option, see “Settings and Configuration Files Introduction” in Quartus II Help.

To make location assignments, use either the `set_location_assignment` Tcl command or the more generic Tcl command `set_instance_assignment`. In the **fir_filter** tutorial, the entity `taps:inst` is assigned to MegaLAB™ location `A1`, as shown in the example below:

```
tcl> set_location_assignment -to taps:inst MegaLAB_A1
```

or

```
tcl> set_instance_assignment -name LOCATION -to taps:\
inst MegaLAB_A1 ←
```

In addition to location, Quartus II logic options can also be assigned to different entities using `set_instance_assignment`. In the `fir_filter` tutorial, the entity `state_m:inst1` is given logic as shown below:

```
tcl> set_instance_assignment -name APEX20K_TECHNOLOGY_MAPPER\
-to state_m:inst1 "Product Term" ←
```

```
entity assignment made
```



For logic option values that contain spaces the value should be enclosed in quotation marks.

To check your instance assignments in a project, use the `get_all_instance_assignments` and the `foreach_in_collection` Tcl commands from the project package. This section shows how this can be done interactively, but since this step involves several steps it is better to do this as a batch file.

First, create a collection named `_col<xx>`, where `xx` is a sequential number generated for each collection created in a Tcl session.

```
tcl>set collection_of_instance_assignments\
[get_all_instance_assignments -name *]
_col4 ←
```

Next, enter the `foreach_in_collection` Tcl command to create the for-loop. An open curly brace (`{`) tells the Tcl session that there are more commands to follow and the Tcl prompt changes from `tcl>` to `>`, indicating that the next command is part of the for-loop being created, as shown in this example:

```
tcl>foreach_in_collection instance\
$collection_of_instance_assignments { ←
>
```

The user-defined variable `instance` is an indexed array of five items for instance assignments. Item 0 contains the Section ID; item 1 contains the Source; Item 2 contains the Destination, item 3 contains the Name of assignments, and item 4 contains the Current Value. In this example, we are only interested in the destination, the name of the assignment, and its value.

Type the following to set the destination (`dest`) variable:

```
> set dest [lindex $instance 2] ←
```

Then, to set the name variable, type the following:

```
> set name [lindex $instance 3] ←
```

Then, to set the value variable, type the following:

```
> set value [lindex $instance 4] ←
```

Finally, the name and the value are displayed on the screen:

```
> puts "Assignment Instance to ($dest) $name = $value " ←
```

A closing brace closes the for loop and starts the `foreach_in_collection` command:

```
> }
```

This sample shows the output from the script:

```
Assignment Instance to (state_m:inst1) APEX20K_TECHNOLOGY_MAPPER
= PRODUCT TERM
Assignment Instance to (taps:inst) LOCATION = MegaLAB_A1
tcl>
```

The `foreach_in_collection` command, when typed, should look like the following:

```
tcl>foreach_in_collection instance\
collection_of_instance_assignments { ←
> set dest [lindex $instance 2] ←
> set name [lindex $instance 3] ←
> set value [lindex $instance 4] ←
> puts "Assignment Instance to ($dest) $name = $value " ←
> } ←
```



For additional information on the `foreach_in_collection` command, see the command-line help.

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

```
tcl> load_package flow ←
1.0
```

For additional help on the `::quartus::flow` package, view the command-line help at the Tcl prompt by typing:

```
tcl> help -pkg ::quartus::flow ←
```

This sample shows an alternative command and the resulting output:

```
tcl> help -pkg flow
-----

-----
Tcl Package and Version:
-----

      ::quartus::flow 1.0

-----
Description:
-----

      This package contains the set of Tcl functions
      for running flows or command-line executables.

-----
Tcl Commands:
-----

      execute_flow
      execute_module

-----
```

tcl>

This help display gives information on the `::quartus::flow` package and what commands are available with the package. To read help on the `execute_flow` Tcl command, short help displays the switch options:

```
tcl> execute_flow -h ↵
```

Long help displays additional information and example usage:

```
tcl> execute_flow -long_help ↵
```

or

```
tcl> help -cmd execute_flow ↵
```

To perform a full compilation of the FIR filter design, use the `execute_flow` command with the `-compile` option, as shown in the following example:

```
tcl> execute_flow -compile ↵
Info:*****
Info: Running Quartus II Analysis & Synthesis
Info: Version 4.0 SJ Full Version
Info: Processing started: Mon Nov 18 09:30:47 2003
Info: Command: quartus_map --import_settings_files=on --
export_settings_files=of fir_filter -c filtref
.
.
.
```

```
Info: Writing report file filtref.tan.rpt
tcl>
```

This script compiles the FIR filter tutorial project, exporting the project assignments and running **quartus_map**, **quartus_fit**, **quartus_asm** and **quartus_tan**. This sequence of events is the same as happens when choosing **Start Compilation** (Processing menu) in the Quartus II GUI.

When you are finished with a project, close it using the `project_close` command:

```
tcl> project_close ←
tcl>
```

Then to exit the interactive Tcl shell, type `exit`.

```
tcl> exit ←
```

Quartus II Legacy Tcl Support

The Quartus II software version 3.0 and later command-line executables do not support the Tcl commands used in previous versions of the Quartus II software. These commands are supported in the GUI by using the Quartus II Tcl console or by using the `quartus_cmd` executable at the command prompt. If you source Tcl scripts developed for an earlier version of the Quartus II software using either of these methods, the project assignments are ported to the project database and settings file. You can then use the command-line executables to process the resulting project. This may be necessary if you create a Tcl file using a third-party EDA tool that does not generate Tcl scripts for the most recent version of the Quartus II software.

Altera recommends creating all new projects and Tcl scripts with the latest version of the Quartus II Tcl API.

References

For more information on using Tcl, see the following sources:

- *Practical Programming in Tcl and Tk*, Brent B. Welch
- *Tcl and the TK Toolkit*, John Ousterhout
- *Effective Tcl/TK Programming*, Michael McLennan and Mark Harrison
- Tcl Developer Xchange at <http://tcl.activestate.com>

