Scripting with Tcl

in the Quartus Software

November 1999, ver. 2.0 Application Note 118

Introduction

Developing and running tool command language (Tcl) scripts in the Quartus™ software allows designers to perform a wide range of simple or complex functions, such as compiling a design or writing procedures to automate common tasks. This application note describes how to develop Tcl scripts for the Quartus software.

The Quartus software supports Tcl version 8.03, which is supplied by Scriptics Corporation (**http://www.scriptics.com**).

What is Tcl?

Tcl 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 application programming interfaces (APIs) for integration.

Tcl is an interpretive language that is easy to learn and use. It allows designers to create custom commands or procedures and can be used for multi-platform programming because it works seamlessly across most development platforms like UNIX and Windows NT. For literature on Tcl, see "References" on page 16.

Using Tcl

The Quartus API details a set of interface functions that can be called while using Tcl. Users with some knowledge of Tcl can use the API to write Tcl scripts that automate tasks within the Quartus software. Designers can execute API functions as if they were Tcl commands, creating a single script that can control the design project, make assignments, start and stop compilation, and run simulations.

The basic syntax for a Tcl command is:

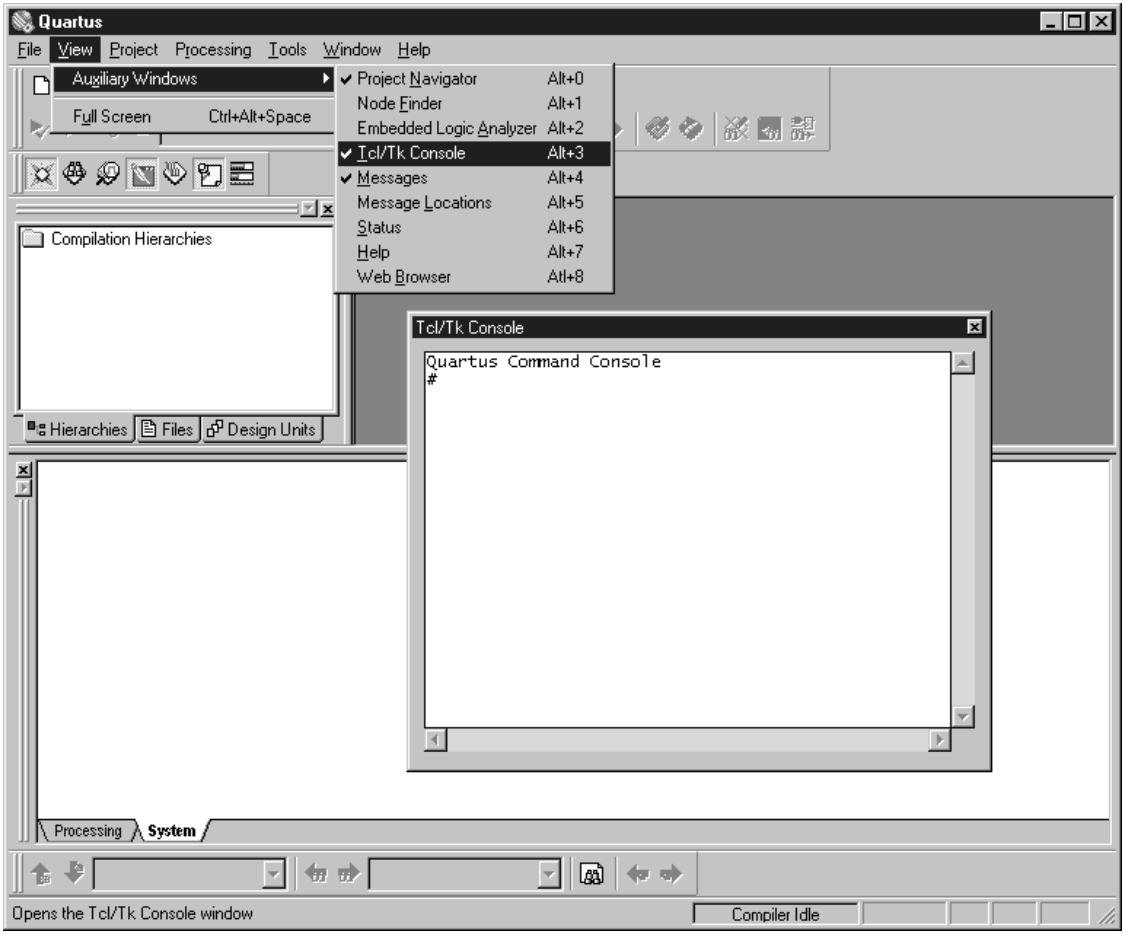
<*command*> [<*argument1*> <*argument2*> <*argument3*>…]

The command syntax is either the name of the built-in command, a procedure, or a set of commands. Spaces separate a command and its arguments, and a new line or semicolon terminates commands. Arguments to commands are passed as strings.

Running Tcl Scripts Interactively

You can execute Tcl commands directly in the Quartus Tcl/Tk Console window. To launch the Tcl/Tk Console window, choose **Auxiliary Windows > Tcl/Tk Console** (View menu). See Figure 1.

Figure 1. Tcl/Tk Console



The Tcl/Tk Console window supports a history, but it does not allow commands to span more than one line. Tcl messages appear in the **System** tab in the Messages window.

Running Tcl Scripts in Batch Mode

Once you create a Tcl Script File (**.tcl**), you can run it by typing the following command in the Tcl/Tk Console window:

`source` *<script file>* ↵

You can also run a script by choosing **Run Script** (Tools menu).

Running Tcl Scripts from DOS or UNIX

The Quartus software also supports `-f` *<script file>* command line arguments. This command is equivalent to choosing **Run Script** (Tools menu). Use the following syntax for running scripts from the DOS or UNIX prompt:

`quartus_cmd –f` *<script file>* ↵

Basic Tcl Commands

Tcl commands in the Quartus software can perform common tasks such as controlling projects and working with the Compiler and Simulator. Table 1 outlines the types of files in which the Tcl interface saves these settings, based on the type of assignment made.

Table 1. Types of Tcl Settings Files

File Type	Description
Quartus File (**.quartus**)	A QUARTUS file contains settings for the entire project.
Project Settings File (**.psf**)	A PSF contains settings for the entire project.
Compiler Settings File (**.csf**)	A CSF contains settings for the Compiler.
Entity Settings File (**.esf**)	An ESF contains parameter settings for individual entities and nodes. *(1)*
Simulator Settings File (**.ssf**)	A SSF contains settings for the Simulator.

Note:
(1) There will be an ESF for every entity in which settings are made.

All commands listed in Tables 2 through 10 apply to the finite impulse response (FIR) filter tutorial included with the Quartus software. For a detailed list of Tcl commands, search for "Application Programming Interface Functions for Tcl" in Quartus Help. To see the following basic Tcl commands in a sample file, go to "Example" on page 10.

Creating Quartus Projects

Table 2 lists the API functions used for creating a project in the Quartus software. You must create a new project or open an existing one before performing any task.

Table 2. Quartus Project Commands

Command	Description
`project exists` <*project name*>	The `project exists` command verifies if a project called <*project name*> already exists to prevent errors during project creation.
`project create` <*project name*>	The `project create` command creates a project called <*project name*> and auxiliary files, including <*project name*>.**quartus**.
`project open` <*project name*>	The `project open` command opens an existing Quartus project called <*project name*>.
`project close`	The `project close` command closes the project that is currently open.

Making Assignments to the Project

After you create a project, you can use Tcl commands to add design files, assign an Altera device, and create project-wide or entity-specific assignments. Table 3 shows the commands that add or remove assignments from a project.

Table 3. Add/Remove Assignments

Command	Description
`project add_assignment` <*entity*> <*section identifier*> <*source*> <*target*> <*variable*> <*value*>	The `project add_assignment` command adds an assignment to the project. Project-wide assignments are written to the PSF or QUARTUS file. Entity-specific assignments are written to the ESF file.
`project remove_assignment` <*entity*> <*section identifier*> <*source*> <*target*> <*variable*> <*value*>	The `project remove_assignment` command removes an existing assignment from a project.

Table 4 defines the arguments for the commands in Table 3.

Table 4. Quartus Project Arguments

Argument	Description
<*entity*>	The <*entity*> argument specifies the entity for the assignment being made.
<*section identifier*>	The <*section identifier*> argument identifies the name of the section in the settings file.
<*source*>	The <*source*> argument specifies the beginning instance name for a range of targets.
<*target*>	The <*target*> argument specifies the ending name for the range of targets started by <*source*>.
<*variable*>	The <*variable*> argument specifies the variable to be added, changed, or removed.
<*value*>	The <*value*> argument specifies the value to be assigned to the variable.

Argument values that do not apply must be passed as empty strings using quotation marks (""). For example, specifying the source file for the project is a project-wide assignment, and thus entity-specific arguments are unnecessary.

If you are unsure of an assignment's scope, open the GUI, make the assignment, and see to which file the Quartus software wrote the assignment. If it is a PSF or QUARTUS file, the assignment is project-wide and needs no entity-specific arguments. If it is an ESF, entity-specific arguments must be included. This method can also aid in checking your syntax.

For more information on functions related to making assignments, including syntax and usage, see Quartus Help.

Creating Compiler & Simulator Settings

Before compiling or simulating a design, you must create Compiler or Simulator settings and make assignments. Compiler settings are saved in a CSF, and Simulator settings are saved in an SSF. Table 5 lists the commands used with these files.

Table 5. Compiler & Simulator Commands

Command	Description
`project cmp_exists` <*settings*>	The `project cmp_exists` command checks to see if CSF, <*settings*>**.csf**, already exists.
`project create_cmp` <*settings*>	The `project create_cmp` command creates a CSF, <*settings*>**.csf**. This function also designates the settings as the current Compiler settings for the project.
`project set_active_cmp` <*settings*>	The `project set_active_cmp` command specifies <*settings*> as the Compiler setting to use for compilation in the Quartus software.
`project sim_exists` <*settings*>	The `project sim_exists` command checks to see if a SSF, <*settings*>**.ssf**, already exists.
`project create_sim` <*settings*>	The `project create_sim` command creates a SSF, <*settings*>**.ssf**.
`project set_active_sim` <*settings*>	The `project set_active_sim` command specifies <*settings*> as the Simulator settings for simulation in the Quartus software.

Table 6 shows the commands that can be used to change the assignments in a CSF or SSF.

Table 6. Making Assignments in a CSF & SSF

Command	Description
`cmp add_assignment` <*section identifier*> <*source*> <*target*> <*variable*> <*value*>	The `cmp add_assignment` command adds an assignment to the current Compiler settings.
`cmp remove_assignment` <*section identifier*> <*source*> <*target*> <*variable*> <*value*>	The `cmp remove_assignment` command removes an assignment from the current Compiler settings.
`sim add_assignment` <*section identifier*> <*source*> <*target*> <*variable*> <*value*>	The `sim add_assignment` command adds an assignment to the current Simulator settings.
`sim remove_assignment` <*section identifier*> <*source*> <*target*> <*variable*> <*value*>	The `sim remove_assignment` command removes an assignment from the current Simulator settings.

Table 7 describes the arguments for the commands in Table 6.

Table 7. Arguments for Compiler & Simulator Settings

Argument	Description
<*section identifier*>	The <*section identifier*> argument identifies the section name of the CSF or SSF that controls the assignment.
<*source*>	The <*source*> argument specifies the beginning instance name for a range of targets.
<*target*>	The <*target*> argument specifies the last name for the range of targets that began with <*source*>.
<*variable*>	The <*variable*> argument specifies the variable to be added, changed, or removed in the CSF or SSF.
<*value*>	The <*value*> argument specifies the value to assign to the CSF or SSF variable, or the value of the variable to remove.

For more information on functions related to making Compiler and Simulator assignments, including syntax and usage, see Quartus Help.

Controlling the Compiler

Table 8 lists the commands that control the Quartus Compiler after you specify compilation settings.

Table 8. Compiler Commands

Command	Description
`cmp start`	The `cmp start` command starts the Compiler for the active Compiler setting.
`cmp stop`	The `cmp stop` command stops the Compiler.
`cmp is_running`	The `cmp is_running` command checks the status of the Compiler and returns a value of 1 if the Compiler is running and a 0 if the Compiler is stopped.

You may also combine commands to perform conditional steps. For example, the code in Figure 2 checks the status of a compilation and stops the compilation if it is still running.

Figure 2. Example of Combining Tcl Commands

```
# check if the Compiler is running, and if so, stop it
if {[cmp is_running] == 1} {
  cmp stop
}
```

The sample `while` loop in Figure 3 prevents further script execution during compilation. When the compilation finishes, the Tcl interpreter will exit this loop. The `after` statement in the `while` loop tells the interpreter how many milliseconds to wait or "sleep" before it flushes messages (`FlushEventQueue` statement) and checks whether the Compiler is still running. Set a low `after` value if you want to see real-time messages; set a high value if you are compiling a large design, and real-time messaging is not important.

Figure 3. Sample While Loop

```
while {[cmp is_running]} {
  after 1000
  FlushEventQueue
}
```

To pipe Quartus messages to stdout or to a file handle, use the procedures in Figure 4 in a library or a calling script.

Figure 4. Example Code to Pipe Quartus Messages

```
proc postMessage {report msg} {
  # this function overrides the postMessage procedure in
  # quartus/bin/ccl_msg.tcl
  split $msg " "
  set line " Q> [lindex $msg 0] : [lindex $msg 3] "
  puts stdout $line
} ; #_____________postMessage_____________#

proc InternalError {report text} {
  # this function overrides the InternalError procedure in
  # quartus/bin/ccl_msg.tcl
  puts stdout "Q>report (IE = $report"
  puts stdout "Q>msg (OE) = $text"
} ; #_____________InternalError_____________"
```

Controlling the Simulator

Table 9 lists the commands that control the Quartus Simulator after you specify settings.

Table 9. Simulator Commands

Command	Description
`sim initialize`	The `sim initialize` command initializes the Simulator to read all netlists and sets the simulation time to zero.
`sim run <time>` *(1)*	The `sim run` command starts the Simulator for the active simulator setting. *(2)*
`sim stop`	The `sim stop` command stops the Simulator.
`sim is_initialized`	The `sim is_initialized` command checks if the Simulator has been initialized.
`sim is_running`	The `sim is_running` command checks if the Simulator is running.

Notes:
(1) For more information on this command, see "sim run" in Quartus Help.
(2) For more information on this command, see "sim start" in Quartus Help.

Performing Interactive Simulations

You can also use Tcl commands to run interactive simulations with the Quartus Simulator. Table 10 lists useful debugging commands.

Table 10. Interactive Simulator Commands

Command	Description
`sim force_value` <*signal name*> <*value*>	The `sim force_value` command forces the value of the `signal_name` signal to 1 or 0 (designate the value as 1 to force the value high, and 0 to force the value low).
`sim release_value` <*signal name*>	The `sim release_value` command releases the value of the `signal_name` signal to revert back to the original state, thus allowing the Simulator to overwrite the current value while simulating at a future time.
`sim get_value` <*signal name*>	The `sim get_value` command checks the value of the `signal_name` signal.
`sim run` <*time*> *(1)*	The `sim run` command specifies the length of time to run the Simulator. *(2)*
`sim run end` *(1)*	The `sim run end` command runs the simulation until completion.
`sim get_time`	The `sim get_time` command checks the current time of the on-going simulation.

Notes:
(1) For more information on this command, see "sim run" in Quartus Help.
(2) For more information on this command, see "sim start" in Quartus Help.

For more information on functions related to running interactive simulations, including syntax and usage, see Quartus Help.

Example

The sample file in Figure 5 shows how to create a project, make assignments, and run a simple compilation using Tcl.

Figure 5. Running a Simple Compilation in Tcl

```
# Change to the working directory
cd D:/qdesigns/tutorial

# check the existence of a project, and if it exists, delete the files
if [project exists filtref] {
  file delete -force filtref.quartus
  file delete -force filtref.psf
  file delete -force filtref.esf
  file delete -force filtref.csf
  file delete -force filtref.ssf
  file delete -force db
}

# create project
project create filtref

# open project
project open filtref

# add source files to current project
project add_assignment "" "" "" "" SOURCE_FILE filtref.bdf
project add_assignment "" "" "" "" SOURCE_FILE acc.v
project add_assignment "" "" "" "" SOURCE_FILE accum.v
project add_assignment "" "" "" "" SOURCE_FILE hvalues.v
project add_assignment "" "" "" "" SOURCE_FILE mult.v
project add_assignment "" "" "" "" SOURCE_FILE state_m.v
project add_assignment "" "" "" "" SOURCE_FILE taps.v

# assign signal clk as a global signal
project add_assignment filref "" "" "" "" |clk" GLOBAL_SIGNAL ON

# create Compiler settings for filtref
project create_cmp filtref

# set the current Compiler settings to filtref
project set_active_cmp filtref

# assign device family
cmp add_assignment "" "" "" "" FAMILY APEX 20K

# assign device
cmp add_assignment filtref "" "" "" DEVICE EP20K100TC144-1

# Start compilation
cmp start
```

Frequently Used Commands

Tcl allows command and procedure customization to fit the functionality of your designs, offering better control and extension of your projects. This section describes and gives examples of frequently used Tcl commands.

Multicycle Path

A multicycle path is a path that intentionally requires more than one cycle to become stable. Declaring a multicycle path tells the timing analyzer to adjust its measurements and allow *x* clock cycles (where *x* equals the number of cycles entered) so that it does not report set-up time violations for the given path.

The sample multicycle path command shown in Figure 6 creates a clock setting, `base_clock`, with an f_{MAX} requirement of 40 MHz in project `project_name`. The signal, `clock`, is then assigned to use the `base_clock` settings. The path from register `start` to register `data_outA_out` is a multicycle path of three cycles.

Figure 6. Sample Multicycle Path

```
project add_assignment "" "base_clock" "" ""
  FMAX_REQUIREMENT 40MHZ
project add_assignment project_name "" "" "|clock"
  USE_CLOCK_SETTINGS base_clock
project add_assignment entity_name "" "|start"
  "|data_outA_out" MULTICYCLE 3
```

Multiclock Domain

A multiclock domain sets up a clock configuration in which two or more clocks exist within a single device.

The multiclock domain example in Figure 7 sets up a multiclock domain in `project_name`. There are two clocks, `clock1` and `clock2`, that are assigned to clock settings `clock_40MHz` and `clock_32MHz`, respectively. `clock_40MHz` is an absolute clock on which `clock_32MHz` is based.

Figure 7. Sample Multiclock Domain

```
project add_assignment "" "clock_40MHz" "" ""
  FMAX_REQUIREMENT 40MHz
project add_assignment "" "clock_32MHz" "" ""
  BASED_ON_CLOCK_SETTINGS clock_40MHz
project add_assignment "" "clock_32MHz" "" ""
  MULTIPLY_BASE_CLOCK_PERIOD_BY 4
project add_assignment "" "clock_32MHz" "" ""
  DIVIDE_BASE_CLOCK_PERIOD_BY 5
project add_assignment project_name "" "" "|clock2"
  USE_CLOCK_SETTINGS clock_40MHz
project add_assignment project_name "" "" "|clock1"
  USE_CLOCK_SETTINGS clock_32MHz
```

Add t_{PD} Assignment

Propagation delay (t_{PD}) is the time required for a signal from an input pin to propagate through combinatorial logic and appear at an output pin. This setting can be specified for the entire project and/or any input, output, or buffer pin.

To set the t_{PD} for the entire design, use the following Tcl command. This example sets the entire design's t_{PD} to 11 ns.

```
project add_assignment "" "" "" "" TPD_REQUIREMENT 11ns
```

To set the t_{PD} from a logic cell, `1cell`, to an output pin, use the following Tcl command. This example sets the `1cell` to output pin t_{PD} to 4 ns.

```
project add_assignment entity_name "" "|1cell"
  "|output_pin" TPD_REQUIREMENT 4ns
```

Add Device

To add a device to your project, use the `cmp add_assignment` command. The following example shows how to add an APEX 20K or APEX 20KE device using Tcl script.

```
cmp add_assignment "" "" "" FAMILY <family name>
cmp add_assignment entity_name "" "" DEVICE <device name>
```

where:
<*family name*> = `APEX20K, APEX20KE`
<*device name*> = any valid device in the specified family

Add Pinout

To add a pin assignment, use the following Tcl command.

```
cmp add_assignment <chip_name> "" "<signal_name>" LOCATION
Pin_<pin>
```

where:
<*chip_name*> = the name of the chip, which is typically the project name
<*signal_name*> = the name of the I/O signal
<*pin*> = the valid I/O pin number to be added

Generating Verilog HDL & VHDL Simulation Files

To generate Verilog HDL and VHDL simulation files with a standard Verilog HDL or VHDL simulator, use the following Tcl command.

```
project add_assignment "" project_name "" ""
  EDA_SIMULATION_TOOL <simulation tool>
```

where:
<*simulation tool*> = `Modelsim, SpeedWave, VCS, Verilog-XL, VSS, Custom Verilog HDL, or Custom`

Using an EDA Synthesis Tool

To interface with an industry-standard EDA tool such as Synopsys Design Complier, Synopsys FPGA Express, Exemplar Leonardo Spectrum, Synplicity Synplify, or Viewlogic ViewDraw, use the following Tcl command.

```
project add_assignment "" project_name "" ""
   EDA_DESIGN_ENTRY_SYNTHESIS_TOOL <EDA tool>
```

where:
<*EDA tool*> = Design Architect, Design Compiler, FPGA Compiler II, FPGA Express, Leonardo Spectrum, Synplify, ViewDraw, Custom

Setting the Technology Mapper

The technology mapper instructs the Compiler to implement hierarchy in the design as ROM, product-term logic, look-up table (LUT), or AUTO (in which the logic type is automatically determined). To set the technology mapper, use the following Tcl command.

```
project add_assignment entity_name "" "" ""
   TECHNOLOGY_MAPPER <technology>
```

where:
<*technology*> = ROM, product_term, LUT, AUTO

Setting the Optimization Technique

To optimize the area and/or timing of a project, set the optimization technique as in the following Tcl example.

```
project add_assignment "" "" "" ""
   OPTIMIZATION_TECHNIQUE <technique>
```

where:
<*technique*> = speed, area

Altera provides extensive documentation to help you design with the Quartus software. For technical support, contact Altera Applications at (800) 800-EPLD. You can also e-mail your technical questions to Altera at **support@altera.com** or launch the Quartus Web Support web site directly from the Quartus software.

References

For more information on using Tcl, refer the following sources.

- *Practical Programming in Tcl and TK*, Brent B. Welch.
- *Tcl and TK Toolkit*, John Ousterhout.
- *Effective Tcl/TK Programming*, Michael McLennan and Mark Harrison.
- **http://www.scriptics.com**

101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
http://www.altera.com
Applications Hotline:
(800) 800-EPLD
Customer Marketing:
(408) 544-7104
Literature Services:
(888) 3-ALTERA