6. Script-Based Design for HardCopy II Devices

Introduction

The Quartus® II software includes a set of command-line executables, many of which support an interactive Tcl shell. Using the Tcl shell, you can perform FPGA or HardCopy® design operations without using the Quartus® II window-based GUI.

This chapter provides an introduction to Tcl operations for script-based HardCopy II design using the interactive Tcl shell. Topics covered in this chapter include:

■ Overview of Tcl scripting features in the Quartus II software
■ HardCopy II design flow
■ Applying location and timing constraints
■ Synthesis, place and route for HardCopy II designs, and Stratix® II prototypes
■ Design verification and analysis

Tcl Support in the Quartus II Software

The Quartus II software provides different ways to execute Tcl commands and scripts, including:

■ A Tcl Console window
■ A Tcl Scripts dialogue box
■ Command-line processing
■ An interactive Tcl shell

The Tcl Console window and **Tcl Scripts** dialogue box both run within the Quartus II GUI and are not described here. Instead, this chapter focuses on the Interactive Tcl shell that you can use with the Quartus II command-line executables.

For more information about command-line processing and the use of Quartus II command-line executables in batchfiles, makefiles, and scripts, refer to the *Command-Line Scripting* chapter in volume 2 of the *Quartus II Handbook*.

For more information on the Quartus II Tcl implementation, refer to the *Tcl Reference Manual* and the *Tcl Scripting* chapter of the *Quartus II Handbook*.

Interactive Tcl Shell

A number of the Quartus II executables can be run with an interactive Tcl shell as the user interface. These executables are identified in Table 6–1. The interactive Tcl shell supports Tcl version 8.4.

Table 6–1. Quartus II Command-Line Executables with Interactive Tcl Support	
Executable Name	**Description**
quartus_sh	A basic Tcl interpreter shell. Supports assignment specification, compile operations, and native operating system commands. For more information, refer to quartus_sh in the *Command-Line Executables* section of the *Quartus II Scripting Reference Manual*.
quartus_sta	The Quartus II TimeQuest timing analyzer engine supports building the timing graph for the design and timing analysis Tcl commands. For more information, refer to quartus_sta in the *Command-Line Executables* section of the *Quartus II Scripting Reference Manual*.
quartus_tan	The Quartus II Classic Timing Analyzer engine supports building the timing graph for the design and timing analysis Tcl commands. For more information, refer to quartus_tan in the *Command-Line Executables* section of the *Quartus II Scripting Reference Manual*.
quartus_cdb	The Quartus II database interface executable. Supports operations related to the design database such as LogicLock, back-annotation, and FPGA-HardCopy comparison for HardCopy II designs. For more information, refer to quartus_cdb in the *Command-Line Executables* section of the *Quartus II Scripting Reference Manual*.
quartus_sim	The Quartus II Simulator. For more information, refer to quartus_sim in the *Command-Line Executables* section of the *Quartus II Scripting Reference Manual*.

The interactive Tcl shell for command-line executables is invoked using the -s command-line switch. For example, to run the basic Quartus shell, type quartus_sh -s at the command prompt:

```
% quartus_sh -s
Info:
Info: ***********************************************************************
Info: Running Quartus II Shell
Info:
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:
Info: ***********************************************************************
tcl>
```

The Quartus II Tcl implementation provides custom Tcl procedures to perform Quartus II operations. These procedures are organized into Tcl packages based on their functionality. Table 6–2 lists these Tcl packages and their availability. Some packages are loaded by default when the executable is invoked. Others must be explicitly loaded before their Tcl procedures are used. To load a particular package, use the load_package Tcl procedure. For example, to load the flow package in the quartus_sh shell, the following Tcl statement is executed:

```
tcl> load_package flow
```

☞ It is important to note that not all executables support all Tcl packages.

Table 6–2. Tcl Package Support in Quartus II Executables		*(Part 1 of 2)*
Executable Name	**Supported Tcl Package**	**Loaded by Default?**
quartus_sta	device	Loaded
	misc	Loaded
	flow	Not loaded
	project	Loaded
	report	Loaded
	sdc	Loaded
	sta	Loaded

Table 6–2. Tcl Package Support in Quartus II Executables   (Part 2 of 2)

Executable Name	Supported Tcl Package	Loaded by Default?
`quartus_sh`	device	Loaded
	flow	Not Loaded
	misc	Loaded
	project	Loaded
	report	Not Loaded
`quartus_tan`	advanced_timing	Not Loaded
	device	Not Loaded
	flow	Not Loaded
	logiclock	Not Loaded
	Misc	Loaded
	project	Loaded
	report	Not Loaded
	timing	Loaded
	timing_report	Not Loaded
`quartus_cdb`	backannotate	Not Loaded
	chip_editor	Not Loaded
	device	Loaded
	flow	Not Loaded
	logiclock	Not Loaded
	misc	Loaded
	project	Loaded
	report	Not Loaded
`quartus_sim`	device	Loaded
	flow	Not Loaded
	misc	Loaded
	project	Loaded
	report	Loaded
	simulator	Loaded

A brief description of each of the Tcl packages referenced in Table 6–2 is given in Table 6–3.

To find out which Tcl packages are loaded, use the command `quartus_??? --tcl_eval help`. For example: `quartus_sta --tcl_eval help`.

Table 6–3. Quartus II Tcl Package Descriptions

Tcl Package	Description
advanced_timing	Traverse the timing netlist and get information about timing modes.
backannotate	Back annotate assignments.
chip_editor	Identify and modify resource usage and routing with the Chip Editor.
database_manager	Manage version-comparable database files.
device	Get device and family information from the device database.
flow	Compile a project, run command-line executables and other common flows.
logiclock	Create and manage LogicLock regions.
misc	Perform miscellaneous tasks.
project	Create and manage projects and revisions and make any project assignments including timing assignments.
report	Get information from report tables and create custom reports.
simulator	Configure and perform simulations.
stp	Operate the SignalTap® II Analyzer.
timing	Annotate timing netlist with delay information, compute and report timing paths.
timing_report	List timing paths.

The Quartus II command-line executables and Tcl shells are supported on all Quartus II operating systems, including Microsoft Windows, Linux, and Unix platforms.

For more information on Quartus II Tcl packages and their available Tcl procedures, refer to the *Tcl Packages and Commands* chapter in the *Quartus II Scripting Reference Manual*.

Command-Line Processing

In addition to the interactive Tcl shell, the Quartus II command-line executables support command-line switches for executing Tcl scripts and commands. When used with these switches, a command-line executable quits when complete. The command-line executables also provide switches for performing specific Quartus II operations. For example, the following c-shell script takes as its argument the top-level design file and entity name and runs it through the entire HardCopy II design flow.

```
!#/bin/csh
quartus_sh --flow compile %1
quartus_cdb %1 --create_companion=%1_hcii
quartus_sh --flow compile %1 -c %1_hcii
quartus_cdb --compare=%1_hcii %1 -c %1
```

This example shows what is, perhaps, the simplest way to execute the HardCopy II design flow. If you have developed and applied the design I/O, location and timing constraints for the project, these constraints are included during script execution.

For more information on the Quartus II executables and command-line options, refer to the *Command-Line Executables* chapter in the *Quartus II Scripting Reference Manual* and the *Command-Line Scripting* section in volume 2 of the *Quartus II Handbook*.

The HardCopy II Design Flow

The Quartus II software supports both HardCopy II first and Stratix II first design flows. The Stratix II first flow involves the following:

- Compiling for the Stratix II FPGA prototype
- Verifying the Stratix II FPGA prototype
- Migrating the prototype design to a HardCopy II design
- Compiling the HardCopy II design
- Transferring your HardCopy II files to the Altera® Design Center

The Hardcopy II first flow is similar, but starts with compiling the HardCopy II target device. Once the HardCopy II compile completes successfully, the design is migrated to the Stratix II target.

The HardCopy II design flow in the Quartus II software is shown in Figure 6–1. To begin a design, create a new project and revision for the Stratix II FPGA prototype. Apply Quartus II settings together with I/O assignments and timing constraints. Compile the Stratix II prototype revision (synthesis, place and route, and assembly) to produce a complete layout, with timing closure and free from errors. You can now perform any additional functional and timing verification necessary and then implement and verify the prototype in hardware.

Once the FPGA prototype is verified, you can compile the HardCopy II design. Begin by creating a HardCopy II companion revision for the FPGA prototype:

1. Create a HardCopy II companion revision for the FPGA prototype. All design settings and constraints are automatically migrated to the new companion revision.

2. Compile the HardCopy II revision. As the compile runs, the Design Assistant checks for errors. When the compile completes, you should correct errors and resolve failures that appear in the Quartus II reports.

3. Run the HardCopy II Companion Revision Comparison tool to compare the HardCopy II design against the FPGA prototype. The comparison tool checks for structural equivalency and consistency between the two revisions.

4. If there are no mismatches, you can prepare the HardCopy II design files for transfer to the Altera Design Center.

☞ In addition to design verification in the Quartus II software, the flow can generate files required to perform Static Timing Analysis (STA) in Synopsys' Primetime.

Figure 6–1. The HardCopy II Design Flow
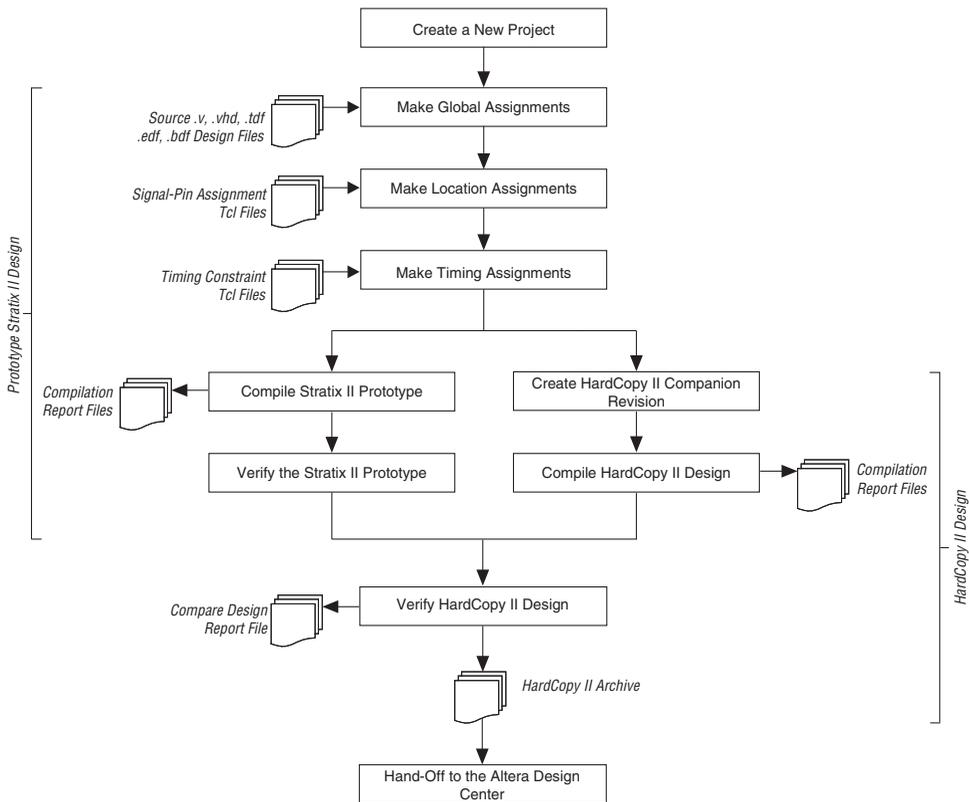
The design flow of Figure 6–1 begins with a Stratix II FPGA prototype design and migrates this design to a HardCopy II device target, or begins with a HardCopy II target and migrates this design to a Stratix II target for FPGA prototyping. The design flow for both cases is shown in Figure 6–1.

For more information on the HardCopy II design flow and alternative methods to complete HardCopy II designs using the Quartus II GUI, refer to the *Quartus II Support for HardCopy II Devices* chapter in the *Quartus II Handbook* or the *HardCopy II Design Considerations* chapter in volume 1 of the *HardCopy Series Handbook*.

The following sections describe each step of the flow shown in Figure 6–1 and explains how each step is completed using the interactive Tcl shell.

Creating a New Project

Both FPGA and HardCopy design in the Quartus II software revolve around the use of projects. You must create a project before you begin working with a new design. A project includes source design files (RTL and schematics), Quartus II tool settings, and a set of pin locations and timing constraints. Although a project can contain many different revisions for a design, each revision can have a unique set of design constraints, target device settings, and Quartus II software settings. You must explicitly open a project before you can perform other operations on the project. You must close the current project to switch to a different project or revision.

This section details the different operations relating to project management using Tcl commands.

Creating a Stratix II Prototype Project

To create a new Stratix II prototype project, use the **project_new** Tcl command. The syntax for this command is:

```
tcl> project_new [-family <family>] [-overwrite] \
    [-part <part>] [-revision <revision_name>] \
    <project_name>
```

The only required argument for this command is the project name, *<project name>*, although the target device family, part code, and revision name can be specified at this time also. By default, the revision name is the same as the project name. The device family and part code can be set later using the **set_global_assignment** command. For example, to create

a project called demo_design with the default revision name of demo_design and an unspecified target device family or part, the following Tcl command is executed:

```
tcl> project_new demo_design
```

Creating a new project creates a quartus settings file (QSF) and a Quartus II Project file (QPF) in the current directory. In addition, a db subdirectory is created that is used to store Quartus II database files. In the case of the demo_design project example, the following files are created in the project directory:

```
demo_design.qpf
demo_design.qsf
db/
    demo_design.db_info
```

Opening a Project

The project created automatically opens when you use the **project_new** command. In future Quartus II sessions, or if you close the project, you must open the project with the Tcl command: **project_open**. The syntax for the **project_open** command is:

```
tcl> project_open [-current_revision] \
    [-revision <revision_name>] <project_name>
```

For example, to open the default revision of project demo_design, execute the following Tcl command:

```
tcl> project_open demo_design
```

☞ It is a good practice to have consistent names for the Stratix II and HardCopy II revisions of your project. This makes it easy to identify which revision is which. For example, naming your revisions *projectname_fpga* and *projectname_hcii* would help you easily identify which revision is the Stratix II revision, and which is the HardCopy II revision.

Closing a Project

Before ending a Quartus II project session, it is good practice to close the Quartus II project using the **project_close** command. This ensures that any changes you have made to your project are written to the Quartus II QSF file. The syntax for the **project_close** command is:

```
tcl> project_close [-dont_export_assignments]
```

New Project Example Script

The following script shows the use of Tcl commands for opening and closing a project called demo_design with the revision name, demo_design_fpga. If the project does not already exist, it is created. This script makes use of the **project_exists** and **project_open** Tcl commands.

```
## Example Tcl Script for opening and closing a project

## Open Project demo_design.  If the Project does not Already
## Exist, Create it
if [is_project_open] project_close
if [project_exists demo_design] {
  project_open demo_design -revision demo_design_fpga
} else {
  project_new demo_design -revision demo_design_fpga
}

## Include Other Tcl Commands Here …

## Close project demo_design and write any changes to settings to
## demo_design.qsf
project_close

## End of script
```

For more information on these and other useful project-related commands, refer to the *Project* section in the *Tcl Packages and Commands* chapter in the *Quartus II Scripting Reference Manual*.

Making Global Assignments

Initializing a HardCopy II Design

For a HardCopy II design, the following key operations are required after a Quartus II project is created:

- Specify design source files (Verilog, VHDL, AHDL, EDIF, and BDF files)
- Specify the Stratix II prototype target family and device name
- Specify the HardCopy II companion revision and migration device
- Enable the Design Assistant
- Make recommended HardCopy II specific Quartus II tool settings

In addition to these, other project settings affecting downstream tools, such as synthesis and place-and-route, can be made at this time.

The operations listed above are performed using the **set_global_assignment** command. The syntax for this command is:

```
tcl> set_global_assignment [-comment<comment>] \
    [-disable] [-entity <entity_name>] -name <name> \
    [-remove] [-section_id <section_id>] <value>
```

The most important parameters for the **set_global_assignment** command are *<name>* and *<value>*. The *<name>* argument specifies the Quartus II global variable to be set and *<value>* is the new value assigned to that variable.

One of the steps in initializing a HardCopy II design is to turn on the Design Assistant. When run in the GUI, the Design Assistant provides a visual checklist for running both the Stratix II and HardCopy II phases of the design. For first-time users, this can provide a powerful guide for successfully completing your HardCopy II project.

The key global variables for a HardCopy II project are listed in Table 6–4.

Table 6–4. Key HardCopy II Design Settings	
Global Variable Name <*name*>	**Value Description <*value*>**
VERILOG_FILE	Verilog file name.
VHDL_FILE	VHDL file name.
AHDL_FILE	Altera HDL file name.
EDIF_FILE	EDIF file name.
BDF_FILE	Altera schematic file name.
FAMILY	Device family name, for example, Stratix II.
DEVICE	Prototype FPGA target device name.
TOP_LEVEL_ENTITY	Top-level design entity or module name.
DEVICE_TECHNOLOGY_MIGRATION_LIST	HardCopy II target device name.
COMPANION_REVISION	HardCopy II design revision name.
ENABLE_DRC_SETTINGS	Turn on the Design Assistant.
USE_TIMEQUEST_TIMING_ANALYZER	Set TimeQuest as the default timing analyzer <ON>.
SDC_FILE	File of TimeQuest constraints <constraint_file.sdc>.
You only need the following settings when using Classic Timing Analyzer. Using Classic Timing Analyzer is not recommended.	
REPORT_IO_PATHS_SEPARATELY	Creates a separate report panel for input and output min and max timing results.
FLOW_ENABLE_TIMING_CONSTRAINT_CHECK	Timing constraints are checked for completeness (all clock domains constraints and minimum and maximum constraints are set for all I/O paths).
DO_COMBINED_ANALYSIS	Timing analysis are run for fast and slow operating conditions and for best and worst-case timing analysis, respectively.
IGNORE_CLOCK_SETTINGS	This must be turned off.
ENABLE_RECOVERY_REMOVAL_ANALYSIS	Verify recovery and removal times on asynchronous control and reset signals.
ENABLE_CLOCK_LATENCY	Clock latency is included in timing analysis to asses clock-insertion timing and clock skew.

The `DEVICE` and `DEVICE_TECHNOLOGY_MIGRATION_LIST` variables are the parts used for the Stratix II prototype design and the HardCopy II design. The selected Stratix II prototype device must be compatible with the selected HardCopy II device to make migration possible. Valid pairings for these devices are listed in Table 6–5.

For the `DEVICE_TECHNOLOGY_MIGRATION_LIST` variable, the HardCopy II part names listed in Table 6–5 are used. For the `DEVICE` variables, the Stratix II part names include the speed grade for the part. The speed grade is a two character code indicating industrial (I) or commercial (C) and the speed indicator (number 3, 4, or 5). For example, a -4 commercial part is denoted using the two character speed grade `C4`. The two-character speed grade is appended to the Stratix II part name to form the value string for the `DEVICE` variable.

Table 6–5. Stratix II Prototype Options for HardCopy II (Part 1 of 2)	
HardCopy II Part	**Stratix II Prototype Part**
HC210F484C HC210W484C	EP2S30F484C3 EP2S30F484C4 EP2S30F484C5 EP2S30F484I4
	EP2S60F484C3 EP2S60F484C4 EP2S60F484C5 EP2S60F484I4
	EP2S90H484C4 EP2S90H484C5
HC220F672C	EP2S60F672C3 EP2S60F672C4 EP2S60F672C5 EP2S60F672I4
HC220F780C	EP2S90F780C4 EP2S90F780C5
	EP2S130F780C4 EP2S130F780C5

Table 6–5. Stratix II Prototype Options for HardCopy II (Part 2 of 2)	
HardCopy II Part	**Stratix II Prototype Part**
HC230F1020C	EP2S90F1020C3 EP2S90F1020C4 EP2S90F1020C5 EP2S90F1020I4
	EP2S130F1020C3 EP2S130F1020C4 EP2S130F1020C5 EP2S130F1020I4
	EP2S180F1020C3 EP2S180F1020C4 EP2S180F1020C5 EP2S180F1020I4
HC2401020C	EP2S180F1020C3 EP2S180F1020C4 EP2S180F1020C5 EP2S180F1020I4
HC240F1508C	EP2S180F1508C3 EP2S180F1508C4 EP2S180F1508C5 EP2S180F1508I4

The following two Tcl commands demonstrate setting the DEVICE and DEVICE_TECHNOLOGY_MIGRATION_LIST variables.

```
tcl> set_global_assignment -name DEVICE EP2S90F1020C4
tcl> set_global_assignment -name \
    DEVICE_TECHNOLOGY_MIGRATION_LIST HC230F1020C
```

The Design Assistant

You should turn on the Design Assistant at the beginning of the design process by turning on the `ENABLE_DRC_SETTINGS` global variable.

```
tcl> set_global_assignment \
    -name ENABLE_DRC_SETTINGS ON
```

The Design Assistant runs concurrently with every step of both the prototype Stratix II and HardCopy II design flows. When the Design Assistant is turned on, the Quartus II software checks to ensure that the project fully complies with all HardCopy II design rules and requirements.

For more information on the Design Assistant, refer to the *Design Guidelines for HardCopy II Devices* chapter in volume 1 of the *HardCopy Series Handbook* and the *Quartus Support for HardCopy II Devices* chapter in the *Quartus II Handbook*.

Example Tcl Script for Making Global Assignments

The example Tcl script below illustrates the application of global constraints for a HardCopy II project.

```
## Example Global Assignments Script for a HardCopy II Design
## This Script Applies Settings for a EP2S90 Stratix II
## prototype FPGA target and a HC230 HardCopy II target

## Source Design File Settings
## ===========================
set_global_assignment -name VERILOG_FILE demo_design.v
set_global_assignment -name VERILOG_FILE example_ram.v

## Stratix II Prototype FPGA Target Settings
## =========================================
set_global_assignment -name FAMILY "Stratix II"
set_global_assignment -name DEVICE EP2S90F1020C4
set_global_assignment -name TOP_LEVEL_ENTITY demo_design

## HardCopy II Companion Revision and Target Settings
## ==================================================
set_global_assignment -name COMPANION_REVISION_NAME \
                                    demo_design_hardcopyii
set_global_assignment -name DEVICE_TECHNOLOGY_MIGRATION_LIST HC230F1020

## Design Assistant Assignments and Settings Required for HardCopy II
##=================================================================
set_global_assignment -name ENABLE_DRC_SETTINGS ON
set_global_assignment -name ERROR_CHECK_FREQUENCY_DIVISOR 1
set_global_assignment -name REPORT_IO_PATHS_SEPARATELY ON

## The following assignments are Classic Timing Analyzer only
## and are not used by TimeQuest.
##============================================================
set_global_assignment -name FLOW_ENABLE_TIMING_CONSTRAINT_CHECK ON
set_global_assignment -name DO_COMBINED_ANALYSIS ON
set_global_assignment -name IGNORE_CLOCK_SETTINGS OFF

set_global_assignment -name ENABLE_RECOVERY_REMOVAL_ANALYSIS ON
set_global_assignment -name ENABLE_CLOCK_LATENCY ON

## End of Script
```

Making I/O Assignments

Because of the complex rules governing the use of programmable I/O cells and their availability for specific pins and packages, Altera highly recommends that I/O assignments are completed using the Pin Planning tool and the Assignment Editor in the Quartus II GUI. These tools ensure that all of the rules regarding each pin and I/O cell are applied correctly. The Quartus II GUI can export a Tcl script containing all I/O assignments and specifications. I/O assignments are described here for information only.

For more information on I/O location and type assignments using the Quartus II Assignment Editor and Pin Planner tools, refer to the *Assignment Editor* chapter in volume 2 of the *Quartus II Handbook*.

In this section, I/O specification is considered in two parts:

■ Pin assignments
■ I/O type assignments

Pin Assignments

Design I/O signals are assigned to package balls using the **set_location_assignment** command. The syntax for this command is given below:

```
tcl> set_location_assignment [-comment <comment>] \
    [-disable] [-remove] -to <destination> <value>
```

Here, <destination> is the package ball name and <value> is the design I/O signal name. For BGA and FBGA packages, the ball name follows the form PIN_<coordinate>. For example, to assign design I/O signal data_out[15] to package ball AL17:

```
tcl> set_location_assignment -to PIN_AL17 data_out[15]
```

Setting I/O Type and Parameters

For I/O type and parameter specification, the **set_instance_assignment** command is used. The syntax for this command is:

```
tcl> set_instance_assignment [-comment <comment>] \
    [-disable] [-entity <entity_name>] \
    [-from <source>] -name <name> [-remove] \
    [-section_id <section_id>] \
    [-to <destination>] <value>
```

The assignment name, *<name>*, should be set to IO_STANDARD to indicate that an I/O specification is being applied. The related I/O signal is specified as -to *<destination>*. The destination argument is a string providing details on the I/O type, such as levels and standards. Table 6–6 lists the strings corresponding to the I/O standards supported in HardCopy II devices.

Table 6–6. Tcl I/O Standard Strings	
I/O Type or *<name>*	**Description**
LVTTL	LVTTL I/O
LVCMOS	LVCMOS I/O
"3.3-V PCI"	3.3-V PCI I/O
"3.3-V PCI-X"	3.3-V PCI X I/O
"1.5 V"	1.5-V I/O
"1.8 V"	1.8-V I/O
"2.5 V"	2.5-V I/O
"1.5-V HSTL CLASS I"	QDRII SRAM 1.5-V I/O
"1.5-V HSTL CLASS II"	QDRII SRAM 1.5-V I/O
"1.8-V HSTL CLASS I"	QDRII SRAM/RLDRAM II 1.8-V I/O
"1.8-V HSTL CLASS II"	QDRII SRAM/RLDRAM II 1.8-V I/O
"DIFFERENTIAL 1.5-V HSTL CLASS I"	Memory clock interface
"DIFFERENTIAL 1.5-V HSTL CLASS II"	Memory clock interface
"DIFFERENTIAL 1.8-V HSTL CLASS I"	Memory clock interface
"DIFFERENTIAL 1.8-V HSTL CLASS II"	Memory clock interface
"DIFFERENTIAL 1.8-V SSTL CLASS I"	DDR2 SDRAM
"DIFFERENTIAL 1.8-V SSTL CLASS II"	DDR2 SDRAM
"DIFFERENTIAL SSTL-2"	DDR SDRAM
"DIFFERENTIAL 2.5-V SSTL CLASS II"	DDR SDRAM
"SSTL-18 CLASS I"	DDR2 SDRAM
"SSTL-18 CLASS II"	DDR2 SDRAM
"SSTL-2 CLASS I"	DDR SDRAM
"SSTL-2 CLASS II"	DDR SDRAM
LVDS	2.5-V differential signaling
HYPERTRANSPORT	2.5-V differential signaling
LVPCL	Differential

You can specify a number of other I/O parameters by using the `set_instance_assignment` command. Some of the more common parameters are listed in Table 6–7.

Table 6–7. Tcl Common I/O Parameter Settings		
<name> **setting**	*<value>* **setting**	**Description**
`weak_pull_up_resistor`	on	Implement a weak pull-up resistor on the pin.
`output_pin_load`	integer	Capacitive load for an output or bidirectional pin. Units of pF.
`fast_output_register`	on	Implements a fast output register in the I/O cell or adjacent LAB.
`fast_output_enable_register`	on	Implement a fast output enable register in the I/O cell or/and adjacent LAB.
`fast_input_register`	on	Implements a fast input register in the I/O cell or adjacent LAB.
`current_strength_new`	2 mA, 4 mA, 8 mA, 10 mA, 12 mA, 16 mA, 18 mA, 20 mA, 24 mA `minimum_current` or `maximum_current`	Drive strength for an output or bidi pin.
`stratixii_termination`	differential "series 25 ohms with calibration" "series 25 ohms without calibration" "series 50 ohms with calibration" "series 50 ohms without calibration"	On-chip termination (or impedance matching) for an I/O pin.

For more information on I/O availability in HardCopy II devices, refer to the *I/O Structures and Features* section in volume 1 of the *HardCopy Series Handbook*.

I/O Assignment Example Script

The following Tcl script example specifies several different I/O constraints.

```
## Signal-Ball Assignments
set_location_assignment PIN_AH5 -to addr_out[0]
set_location_assignment PIN_AH6 -to addr_out[1]
set_location_assignment PIN_AJ5 -to data_in[0]
set_location_assignment PIN_AJ6 -to data_in[1]
set_location_assignment PIN_AJ32 -to resetn
set_location_assignment PIN_AM17 -to ref_clk

# I/O Type and Parameter Assignments
set_instance_assignment -name IO_STANDARD "1.5-V HSTL CLASS II" -to addr_out[0]
set_instance_assignment -name IO_STANDARD "1.5-V HSTL CLASS II" -to addr_out[1]
set_instance_assignment -name IO_STANDARD "1.5-V HSTL CLASS II" -to data_in[0]
set_instance_assignment -name IO_STANDARD "1.5-V HSTL CLASS II" -to data_in[1]
set_instance_assignment -name IO_STANDARD LVDS -to resetn
set_instance_assignment -name IO_STANDARD LVCMOS -to ref_clk

set_instance_assignment -name fast_input_register on -to data_in[0]
set_instance_assignment -name fast_input_register on -to data_in[1]
set_instance_assignment -name fast_output_register on -to addr_out[0]
set_instance_assignment -name fast_output_register on -to addr_out[1]

set_instance_assignment -name output_pin_load 10 -to addr_out[0]
set_instance_assignment -name output_pin_load 10 -to addr_out[1]
set_instance_assignment -name current_strength_new 16mA -to addr_out[0]
set_instance_assignment -name stratixii_termination "series 25 ohms without calibration"\
-to data_in[1]
```

Assigning Timing Constraints

Planning Design Timing Constraints

Timing constraints ensure that a design compiled in the Quartus II software meets specific timing requirements. When you target an FPGA, you may decide not to apply a complete set of timing constraints, choosing instead to fix any timing problems in your prototype system if and when they arise. HardCopy devices, however, cannot be modified using reconfiguration to fix timing problems, so it is critically important that a design is fully constrained. Designs not fully constrained would result in significantly different timing characteristics between the prototype Stratix II FPGA and the HardCopy II device. By fully constraining a design, Altera can guarantee that both the Stratix II FPGA and the HardCopy II device fully complies with your timing specifications.

The minimum set of timing constraints for a HardCopy II design are:

■ Clock settings (F_{MAX}) for each and every clock domain
■ Minimum and maximum delays for all I/O paths, including asynchronous reset and control I/O signals

In addition, it is good design practice to develop timing constraints to cover:

■ Specific cross-clock domain timing requirements
■ False paths
■ Multicycle paths

In TimeQuest, timing constraints are written in TimeQuest SDC format and are read from an SDC file. An example file is *demo_design*.**sdc**. See .

In the Classic Timing Analyzer, timing constraints are applied using dedicated Tcl commands and by assigning timing-specific attributes using the **set_instance_assignment** command.

This section provides an overview of timing constraint development using Tcl commands.

For more information on timing constraints, refer to the *Timing Analysis* section in volume 3 of the *Quartus II Handbook*.

Specifying System Clocks

The most basic constraints that should be applied describe the clock for each clock domain. Parameters usually specified for each clock are:

■ Clock period
■ Latency (LATE_CLOCK_LATENCY/EARLY_CLOCK_LATENCY assignments)
■ Uncertainty (**set_clock_uncertainty** command)

Clock uncertainty specified with the **set_clock_uncertainty** command models any uncertainty in the clock period, including jitter, and is often used to introduce some margin into the target clock frequency. The following example constraints illustrate clock definition for a design with two clock domains, clk_a and clk_b. In this case, both clocks run at 100 MHz, but with different clock latency and skew.

```
## Example TimeQuest SDC Constraints Defining Clocks clk_a and clk_b
create_clock -period 10.0 -name clk_a [get_ports clk_a]
set_clock_latency -source -late 3.0 clk_a
set_clock_latency -source -early 2.0 clk_a
```

```
set_clock_uncertainty -to clk_a 0.25

create_clock -period 10.0 -name clk_b [get_ports clk_b]
set_clock_latency -source -late 4.0 clk_b
set_clock_latency -source -early 3.0 clk_b
set_clock_uncertainty -to clk_b 0.25
```

Input/Output Timing

System clock parameters define the setup and hold timing for register to register paths within each clock domain. I/O timing parameters are used to describe I/O to register, and register to I/O timing.

The **set_input_delay** constraint is used to specify the delay from a source external to the chip to an input pin, relative to a defined clock. The syntax for this command is given below.

```
set_input_delay \
    -clock <clock name> \
    [-clock_fall] \
    [-rise | -fall] \
    [-max | -min] \
    [-add_delay] \
    [-reference_pin <pin or port>] \
    <delay value> \
    <port pin list>
```

The *<clock name>* argument specifies the reference clock for the delay. The *<port pin list>* argument is the top-level input signal for the design, and *<delay value>* is the external delay. The external delay is measured from the positive (rising) edge of *<clock>* unless the -clock_fall argument is specified. The –min and –max arguments are used to specify whether *<delay value>* is the minimum or maximum external delay, respectively.

The **set_output_delay** constraint is similar to the **set_input_delay** constraint except that it specifies the delay from an output pin to its external destination relative to a clock.

```
set_output_delay \
    -clock <clock name> \
    [-clock_fall] \
    [-rise | -fall] \
    [-max | -min] \
    [-add_delay] \
    [-reference_pin <pin or port>] \
    <delay value> \
    <port pin list>
```

As an example, the following Tcl script specifies input and output min and max delays for two I/O signals. Input data_in[0] has minimum and maximum external delays of 3 ns and 7 ns, respectively. Output data_out[0] has minimum and maximum external delays of 4 ns and 8 ns, respectively. The external input delays for data_in[0] are relative to the positive edge of clock ref_clk and the external output delays for data_out[0] are relative to the negative edge of clock ref_clk.

```
# Tcl Script Setting I/O Timing Using set_input_delay and set_output_delay
set_input_delay -clock ref_clk -max 7.0 [get_ports data_in[0]]
set_input_delay -clock ref_clk -min 3.0 [get_ports data_in[0]]
set_output_delay -clock ref_clk -max 8.0 [get_ports data_out[0]]
set_output_delay -clock ref_clk -min 4.0 [get_ports data_out[0]]
```

Creating Timing Exceptions

Timing exceptions are used to correct timing constraints not covered by clock settings and I/O timing settings. The most common of these are multicycle paths and false paths.

In TimeQuest, multicycle paths are described using the **set_multicycle_path** constraint. The syntax for this constraint is:

```
set_multicycle_path [-setup][-hold][-start]
```

In Classic Timing Analyzer, multicycle paths are described using the **set_multicycle_assignment** command. The syntax for this command is:

```
tcl> set_multicycle_assignment [-comment <comment>] \
     [-disable] [-end] [-from <from_list>] \
     [-hold] [-remove] [-setup] [-start] \
     [-to <to_list>] <path_multiplier>
```

In either timing analyzer, multicycle assignments are made with the -setup argument, to specify the maximum number of cycles, or with the -hold argument, to specify the minimum number of cycles for a path.

False paths describe paths that should not be included in timing optimization or analysis operations. In the Quartus II software, there are a number of ways to describe false paths. By default, in Classic Timing Analyzer, feedback from the output to input side of bidirectional I/O, read-while-write paths through memories, and cross-clock domain paths are not timed during optimization or timing analysis. By default, in Time Quest, cross-clock domain paths are timed.

To change these default settings, refer to the *Timing Settings* section in the *Quartus II Support of HardCopy Series Devices* chapter in volume 1 of the *Quartus II Handbook*.

In TimeQuest, the constraint **set_false_path** is used to describe paths that should not be included in timing optimization or analysis. The syntax for this constraint is:

```
tcl> set_false_path \
    [-from <from list>] \
    [-to <to list>] \
    [-thru <thru list>]
```

In Classic Timing Analyzer, the most common command for controlling false paths is the **set_timing_cut_assignment** command. The syntax for this command is:

```
tcl> set_timing_cut_assignment \
    [-comment <comment>] \
    [-disable] \
    [-from <from_pin_list>] \
    [-remove] \
    [-to <to_pin_list>]
```

All paths between nodes in the *<from_pin_list>* to nodes in the *<to_pin_list>* are excluded from timing optimization and analysis operations.

Example of TimeQuest SDC Constraints

```
# Timing Assignments
# ==================
create_clock -period 10.0ns -name ref_clk ref_clk

set_clock_latency -late 3 ref_clk
set_clock_latency -early 2 ref_clk
set_clock_uncertainty -hold -to ref_clk 0.250ns
set_clock_uncertainty -setup -to ref_clk 0.250ns

# Input delay of 6ns (max) & 2ns (min) for bus data_in[1:0]
set_input_delay -clock ref_clk -max 6 data_in
set_input_delay -clock ref_clk -min 2 data_in

# Output delay of 6ns (max) & 2ns (min) for bus data_out[1:0]
set_output_delay -clock ref_clk -max 6 data_out
set_output_delay -clock ref_clk -min 2 data_out

# Don't care about timing on the resetn net. Set as false path
set_false_path -from resetn
```

Example of Classic Timing Analyzer Tcl Script

```
# Timing Assignments
# ==================
create_base_clock –fmax 100 MHz –target ref_clk ref_clk

set_instance_assignment -name LATE_CLOCK_LATENCY 3ns -to ref_clk
set_instance_assignment -name EARLY_CLOCK_LATENCY 2ns -to ref_clk
set_clock_uncertainty –hold –to ref_clk 0.250ns
set_clock_uncertainty –setup –to ref_clk 0.250ns

# Input delay of 6ns (max) & 2ns (min) for bus data_in[1:0]
set_input_delay –clk_ref ref_clk –max –to data_in 6.0ns
set_input_delay –clk_ref ref_clk –min –to data_in 2.0ns

# Output delay of 6ns (max) & 2ns (min) for bus data_out[1:0]
set_output_delay -clk_ref ref_clk -max -to data_out 6.0ns
set_output_delay -clk_ref ref_clk -min -to data_out 2.0ns

# Don't care about timing on the resetn net. Set as false path
set_timing_cut_assignment -from resetn
```

This section has provided an overview of Tcl commands for applying timing constraints.

For more information on the application of timing constraints using Tcl commands, refer to the *Tcl Packages and Commands* chapter in the *Quartus II Scripting Reference Manual*.

Compiling the Stratix II Prototype Design

Once all global assignments, resource assignments, and timing assignments have been specified, the next step in the design process is to compile the Stratix II FPGA prototype design. The execute_flow command is provided for this purpose and supports various arguments affecting the compilation process. The syntax for this command is:

```
tcl> execute_flow \
    [-analysis_and_elaboration] \
    [-attempt_similar_placement] \
    [-check_ios] \
    [-check_netlist] \
    [-compile] \
    [-compile_and_simulate] \
    [-early_timing_estimate] \
    [-eco] [-export_database] \
    [-fast_model] \
    [-generate_functional_sim_netlist] \
    [-import_database]
```

The switches relevant to prototype Stratix II and HardCopy II design are listed in Table 6–8.

Table 6–8. `execute_flow` *Tcl Command Switches*	
Switch	**Description**
`analysis_and_elaboration`	Perform synthesis and mapping to the target Altera technology
`attempt_similar_placement`	Runs Attempt Similar Placement
`check_ios`	Verify I/O assignments
`check_netlist`	Perform syntax checks on the netlist
`compile`	Execute the Quartus II compilation flow
`compile_and_simulate`	As for compile, but also run simulation
`early_timing_estimate`	Runs the early timing estimator
`eco`	Executes a Fitter ECO compilation
`export_database`	Exports a Version-Compatible Database
`fast_model`	Runs Timing Analysis (fast mode analysis)
`generate_functional_sim_netlist`	Generate a Simulation Netlist
`import_database`	Imports a Version-Compatible Database

☞ It is important to note that the HardCopy switches for the `execute_flow` command are for HardCopy Stratix designs, not HardCopy II designs.

The simplest way to run the `execute_flow` command is to use the `-compile` switch.

```
tcl> execute_flow -compile
```

Running the **execute_flow** command in this way executes the four stages of the Quartus II compilation flow with default settings for each stage:

■ Analysis and Synthesis
■ Fitter
■ Timing Analysis
■ Assembler

The Design Assistant and Timing constraint checks are run if they are enabled in the Quartus II Settings file.

You should check I/O assignments to avoid problems in downstream compile operations. To do this, the execute_flow compilation is broken into three steps:

1. tcl> execute_flow -analysis_and_elaboration

2. tcl> execute_flow -check_ios

3. tcl> execute_flow -compile

It should be noted that, in the interests of clarity and brevity, the Tcl fragments given here do not incorporate any error checking. However, it is good practice to include code in your Tcl scripts that checks for success as your design proceeds. In the case of the execute_flow procedure, the return value can be used with the Tcl catch command to handle success or failure. The example below shows one option for doing this.

```
# Determine if compilation was successful and
# print out a personalized message.
if {[catch {execute_flow -compile} result]} {
    puts "\nResult: $result\n"
    puts
      "ERROR: Compilation failed. See report files.\n"
} else {
    puts "\nINFO: Compilation was successful.\n"
}
```

For more information on the **execute_flow** command, refer to the command description in the *Tcl Packages and Commands* chapter in the *Quartus II Scripting Reference Manual*.

Compiling the HardCopy II Design

Once the Stratix II FPGA prototype design is compiled and verified, you can compile the HardCopy II revision of the design. This is a two-step process:

1. Create the HardCopy II companion revision.

2. Compile the HardCopy II companion revision.

To create the HardCopy II version of the design, run the **execute_hardcopyii** Tcl command with the -create_companion option:

```
tcl> execute_hardcopyii -create_companion demo_design_hcii
```

This command initializes the database for the HardCopy II revision and creates a new QSF file (in this example, **demo_design_hcii.qsf**), ensuring that all constraints for the Stratix II FPGA revision are ported over.

Next, the current working revision for the Quartus II project is changed to the HardCopy II revision and the design is compiled for the HardCopy II device target:

```
tcl> set_current_revision demo_design_hcii
tcl> execute_flow -compile
```

As with the prototype Stratix II revision, report files are generated in the project directory for each of the tools that are executed.

Understanding Report Files

The **execute_flow** command generates a number of report files in the project directory. These files summarize messages displayed on the console during compilation and provide additional information about the design. The name of each report file follows the format *<revision><tool short name>***.summary** and *<revision><tool short name>***.rpt**, where *<revision>* is the revision name of the current design. The **.summary** file contains a brief summary of messages and results from the tool while the **.rpt** file contains more detailed messages and information. For a HardCopy II project, two sets of report files are generated: one for the Stratix II prototype FPGA revision and one for the HardCopy II revision. Table 6–9 describes the different report files.

☞ The Tcl report package provides a powerful collection of procedures for customizing and managing report files related to the Quartus II fitter and timing analysis engines.

👣 For more information on customizing and managing report files, refer to the *Tcl Packages and Command*s report section of the *Quartus II Tcl Reference Manual*.

Table 6–9. Stratix II Compile Report File Descriptions   (Part 1 of 2)

Switch	Tool	Description
<revision>.**map.rpt**	Analysis & Synthesis	Synthesis settings, source files, messages, and resource usage.
<revision>.**map.eqn**	Analysis & Synthesis	Implementation equations and device resource instantiations.
<revision>.**fit.rpt**	Fitter	Fitter settings, layout optimizations, resources, pin-out, and messages.
<revision>.**fit.eqn**	Fitter	Implemented equations and device resource instantiations after fitting.
<revision>.**drc.rpt**	Design Assistant	Design rule settings, violations, and messages.

Table 6–9. Stratix II Compile Report File Descriptions   (Part 2 of 2)

Switch	Tool	Description
<revision>.**upc.rpt**	Timing Constraint Checker	Constraint coverage information.
<revision>.**asm.rpt**	Assembler	Assembler settings, **.pof** and **.sof** output file options, and messages.
<revision>.**rec.rpt**	Companion Revision Comparison	A status report on the structural comparison between the HardCopy II revision and the Stratix II Prototype design.
<revision>.**flow.rpt**	Flow	Resource summary and execution time for each tool in the flow. This report is updated as different tools in the flow complete.
<revision>.**sta.rpt**	TimeQuest	TimeQuest timing analysis report.

Comparing FPGA and HardCopy Revisions

Before submitting the HardCopy II project to the Altera Design Center, it should be checked against the Stratix II prototype FPGA revision. To do this, run the **execute_hardcopyii** Tcl command with the -compare option from the quartus_sh shell:

```
tcl> execute_hardcopyii -compare
```

Running this command generates a report file and summary file in the project directory. These files are called *<revision_name>*.**rec.rpt** and *<revision_name>*.**rec.summary**. The command checks to verify that the following items conform to HardCopy II design rules and are consistent between the HardCopy II and Stratix II revisions:

■ Source design files and device netlist files
■ User clock assignments
■ Timing constraints (assignments)
■ I/O location and type assignments
■ PLL parameters
■ Memory implantation parameters
■ DSP implementation parameters
■ Global resource properties
■ Properties of all other device resources used

Any errors or failures in comparison are reported in the **.rec** report files. An example **.rec** file is given below. Note that for this example, the design comparison checks in the HardCopy II Companion Revision Comparison Summary table are all marked passed, indicating that the HardCopy II design in the Quartus II software is finished and ready for hand-off to the back-end engineering team in the Altera Design Center.

You must resolve any failures that show up in the Comparison Summary before you proceed any further with your design.

```
HardCopy II Companion Revision Comparison report for demo_design_hardcopyii
Wed Sep 20 15:30:07 2006
Version 6.0 Build 202 06/20/2006 Service Pack 1 SJ Full Version
--------------------
; Table of Contents ;
--------------------
  1. Legal Notice
  2. HardCopy II Companion Revision Comparison Summary
  3. Atom Netlist Comparison Summary
  4. DSP Information
  5. HardCopy II Companion Revision Comparison Messages


+------------------------------------------------------------------------------------+
; HardCopy II Companion Revision Comparison Summary                                  ;
+---------------------------------------------------+--------------------------------+
; HardCopy II Companion Revision Comparison Status ; Analyzed - Wed Sep 20 15:29:55 2006 ;
; Quartus II Version ; 6.0 Build 202 06/20/2006 SP 1 SJ Full Version ;
; Revision Name demo_dsign_hardcopyii ;
; Top-level Entity Name                            ; demo_design                    ;
; Family                                           ; Stratix II                     ;
; Compare Status                                   ; Passed (14/14)                 ;
; Source Files Compared                            ; Passed (121/121)               ;
; Assignments Compared                             ; Passed                         ;
; User Clocks Compared                             ; Passed (0/0)                   ;
; Resource Counts Compared                         ; Passed (5/5)                   ;
; I/O Structure Compared                           ; Passed (130/130)               ;
; Package Pins Compared                            ; Passed (1020/1020)             ;
; PLL Structure Compared                           ; Passed (1/1)                   ;
; PLL Clocks Compared                              ; Passed (2/2)                   ;
; Timing Constraints Compared                      ; Passed (3/3)                   ;
; RAM Information Compared                          ; Passed (10/10)                 ;
; DSP Information Compared                          ; Passed (100/100)               ;
; Global Resources Compared                        ; Passed (8/8)                   ;
; Atom Compared                                    ; Passed (335084/335084)         ;
; Atom Netlist Compared                            ; Passed (1/1)                   ;
+---------------------------------------------------+--------------------------------+
```

Performing Static Timing Analysis

Static Timing Analysis in the Quartus II Software

The global assignments made for the Stratix II prototype and HardCopy II revisions ensure that Static Timing Analysis (STA) is run for both fast and slow operating conditions and both setup and hold timing is verified.

Using TimeQuest

You can run the timing analysis independent of the compile process in one of two ways:

1. Use the **execute_module -tool sta** Tcl command to run a timing analysis Tcl script in quartus_sta from within the basic quartus shell, quartus_sh.

2. Run the `quartus_sta` interactive Tcl shell independently and execute Tcl commands and scripts at the Tcl prompt.

Using Classic Timing Analyzer

You can run the timing analysis independent of the compile process in one of two ways:

1. Use the **execute_module -tool tan** Tcl command to run a timing analysis Tcl script in `quartus_tan` from within the basic quartus shell, `quartus_sh.`

2. Run the `quartus_tan` interactive Tcl shell independently and execute Tcl commands and scripts at the Tcl prompt.

For more information on running static timing analysis in the Quartus II software, refer to the *Timing Analysis* section in the *Quartus II Handbook.*

For Tcl commands related to static timing analysis, refer to the *Timing* section of the *Tcl Packages and Commands* in the *Quartus II Scripting Reference Manua*l.

Static Timing Analysis in Primetime

The Quartus II software can also generate files required to run STA in Synopsys' PrimeTime. The following example Tcl commands direct the Quartus II software to generate PrimeTime files for STA.

```
## Tcl Script to Generate PrimeTime STA File Output
execute_module -tool sta -args --tq2pt
execute_module -tool eda -args "--tool primetime --format verilog --timing_analysis"
```

The files generated by the Quartus II software are organized in a subdirectory within the project directory. For example, after compiling a Stratix II prototype design (`demo_design`), the following verilog (**.vo**) SDF (**.sdo**) and PrimeTime Tcl script (**.tcl**) are created in the project directory.

```
timing\
      primetime\
         demo_design_v.sdo
         demo_design.pt.tcl
         demo_design.collections.sdc
         demo_design.constraints.sdc
```

The Tcl script includes all timing constraints applied during the Quartus II software compilation.

HardCopy II Example Tcl Script

The following script draws together the Tcl ideas discussed thus far into a top-level Tcl script for the `quartus_sh` Tcl shell. This script implements a HardCopy II design called demo_design. It begins by creating a new project, called `demo_design`, compiling the Stratix II FPGA prototype, creating a HardCopy II companion revision and then compiling the companion revision. Finally, the revision comparison tool is run to verify that both revisions are consistent.

In this example, global, pin, and timing assignment scripts are read into the top-level script using the Tcl **source** command. The sourced scripts are listed after the top-level script listing.

Top-Level Example Script demo_design.tcl

```
## demo_design.tcl
## Top-level script for executing a HardCopy II design in quartus_sh -s
load_package flow

## Open of create the Stratix II FPGA prototype revision
if [is_project_open] project_close
if {[project_exists demo_design]} {
  project_open demo_design

} else {
  project_new demo_design
}

## Apply global design settings
source global_assignments.tcl

## Apply I/O assignments
source pin_assignments.tcl

## Apply FPGA timing constraints
source timing_assignments.tcl

## Compile the Stratix II FPGA prototype design
execute_flow -compile

# #Create and switch to the HardCopy II target revision
execute_hardcopyii -create_companion demo_design_hcii
project_close
project_open demo_design -revision demo_design_hcii

## Compile the HardCopy II design revision
execute_flow -compile

## Check the HardCopy II revision and make sure it matches the FPGA
## design
execute_hardcopyii -compare
```

```
##  Generate a HardCopy II Handoff Report
execute_hardcopyii -handoff_report

##  Archive the HardCopy II Handoff Files into
##  the file named "demo_design_hcii_handoff.qar"
execute_hardcopyii -archive demo_design_hcii_handoff.qar

## Quit quartus_sh -s
qexit

## End of demo_design.tcl
```

Global Assignments Script global_assignments.tcl

The `global_assignments.tcl` script source in the top-level script, `demo_design.tcl` prepares global variables, target devices, and revision names for the HardCopy II project:

```
## global_assignments.tcl

## Source Design File Settings
## ===========================
set_global_assignment -name VERILOG_FILE demo_design.v
set_global_assignment -name VERILOG_FILE example_ram.v

## Constraint File Settings for TimeQuest
## ==============================
set_global_assignment -name USE_TIMEQUEST_TIMING_ANALYZER ON
set_global_assignment -name SDC_FILE demo_design.sdc

## Stratix II Prototype FPGA Target Settings
## =========================================
set_global_assignment -name FAMILY "Stratix II"
set_global_assignment -name DEVICE EP2S90F1020C4
set_global_assignment -name TOP_LEVEL_ENTITY demo_design

## HardCopy II Companion Revision and Target Settings
## ==================================================
set_global_assignment -name COMPANION_REVISION_NAME \
demo_design_hardcopyii
set_global_assignment -name DEVICE_TECHNOLOGY_MIGRATION_LIST HC230F1020

## Design Assistant Assignments and Settings Required for HardCopy II
## =================================================================
set_global_assignment -name ENABLE_DRC_SETTINGS ON
set_global_assignment -name ERROR_CHECK_FREQUENCY_DIVISOR 1
set_global_assignment -name REPORT_IO_PATHS_SEPARATELY ON

## The following assignments are Classic Timing Analyzer only and
## are not used by TimeQuest.
## ==================================================
set_global_assignment -name FLOW_ENABLE_TIMING_CONSTRAINT_CHECK ON
set_global_assignment -name DO_COMBINED_ANALYSIS ON
set_global_assignment -name IGNORE_CLOCK_SETTINGS OFF
```

```
set_global_assignment -name ENABLE_RECOVERY_REMOVAL_ANALYSIS ON
set_global_assignment -name ENABLE_CLOCK_LATENCY ON

## End of global_assignments.tcl
```

Pin Assignments Script pin_assignments.tcl

The `pin_assignments.tcl` script run from the top-level script, `demo_design.tcl`, specifies top-level design signal to package ball assignments and I/O parameters:

```
## pin_assignments.tcl
set_location_assignment PIN_AH5 -to addr_out[0]
set_location_assignment PIN_AH6 -to addr_out[1]
set_location_assignment PIN_AJ5 -to data_in[0]
set_location_assignment PIN_AJ6 -to data_in[1]
set_location_assignment PIN_AJ32 -to resetn
set_location_assignment PIN_AM17 -to ref_clk

## I/O Type and Parameter Assignments
set_instance_assignment -name IO_STANDARD "1.5-V HSTL CLASS II" -to addr_out[0]
set_instance_assignment -name IO_STANDARD "1.5-V HSTL CLASS II" -to addr_out[1]
set_instance_assignment -name IO_STANDARD "1.5-V HSTL CLASS II" -to data_in[0]
set_instance_assignment -name IO_STANDARD "1.5-V HSTL CLASS II" -to data_in[1]
set_instance_assignment -name IO_STANDARD LVDS -to resetn
set_instance_assignment -name IO_STANDARD LVCMOS -to ref_clk

set_instance_assignment -name fast_input_register on -to data_in[0]
set_instance_assignment -name fast_input_register on -to data_in[1]
set_instance_assignment -name fast_output_register on -to addr_out[0]
set_instance_assignment -name fast_output_register on -to addr_out[1]

set_instance_assignment -name output_pin_load 10 -to addr_out[0]
set_instance_assignment -name output_pin_load 10 -to addr_out[1]

## End of pin_assignments.tcl
```

TimeQuest Constraint File demo_design.sdc

TimeQuest reads the SDC file *demo_design*.**sdc** and applies timing constraints for the system clock, `ref_clk`, and I/O-to-core timing specifications.

```
## constraints.sdc
create_clock –period 10.0 MHz -name ref_clk [get_ports ref_clk]

set_clock_latency -late 3 ref_clk
set_clock_latency -early 2 ref_clk
set_clock_uncertainty –hold –to ref_clk 0.250
set_clock_uncertainty -setup –to ref_clk 0.250

# Input delay of 6ns (max) & 2ns (min) for bus data_in[1:0]
set_input_delay -clock ref_clk -max 6 [get_ports data_in]
```

```
set_input_delay –clock ref_clk –min 2 [get_ports data_in]
# Output delay of 6ns (max) & 2ns (min) for bus data_out[1:0]
set_output_delay –clock ref_clk –max 6 [get_ports data_out]
set_output_delay –clock ref_clk –min 2 [get_ports data_out]

# Don't care about timing on the resetn net. Set as false path
set_false_path -from [get_ports resetn]
## End of timing_assignments.tcl
```

Timing Assignments Script timing_assignments.tcl

If you are using Classic Timing Analyzer, the
timing_assignments.tcl script is run from the top-level script,
demo_design.tcl. This script applies timing constraints for the system
clock, ref_clk, and I/O-to-core timing specifications.

```
## timing_assignments.tcl
create_base_clock –fmax 10.0ns –target ref_clk ref_clk

set_instance_assignment -name LATE_CLOCK_LATENCY 3ns -to ref_clk
set_instance_assignment -name EARLY_CLOCK_LATENCY 2ns -to ref_clk
set_clock_uncertainty -hold -to ref_clk 0.250ns
set_clock_uncertainty -setup -to ref_clk 0.250ns

# Input delay of 6ns (max) & 2ns (min) for bus data_in[1:0]
set_input_delay –clk_ref ref_clk –max –to data_in 6.0ns
set_input_delay –clk_ref ref_clk –min –to data_in 2.0ns
# Output delay of 6ns (max) & 2ns (min) for bus data_out[1:0]
set_output_delay –clk_ref ref_clk –max –to data_out 6.0ns
set_output_delay –clk_ref ref_clk –min –to data_out 2.0ns

# Don't care about timing on the resetn net. Set as false path
set_timing_cut_assignment -from resetn
## End of timing_assignments.tcl
```

Summary

This chapter introduced script-based design for HardCopy II devices
using the Quartus II interactive Tcl shell. This approach provides you
with an alternative to GUI-based design for certain situations such as
remote-terminal Quartus II execution, design flow automation, or even if
you are simply more comfortable operating in a scripting environment.

Document Revision History

Table 6–10 shows the revision history for this chapter.

Table 6–10. Document Revision History		
Date and Document Version	**Changes Made**	**Summary of Changes**
September 2008, v1.3	Updated chapter number and metadata.	—
June 2007, v1.2	Minor text edits.	—
December 2006 v1.1	Updates for the Quartus II software version 6.1.0 ● Added information on the Tcl command-line executable quartus_sta, newly available in Quartus II software version 6.1.0, and recommended for use in HardCopy II design timing analysis. ● Updated Figure 6–1. ● Updated Table 6–1, Table 6–2, and Table 6–3. ● Added revision history.	A medium update to the chapter, due to changes in the Quartus II software version 6.1 release.
March 2006	Formerly chapter 15; no content change.	—
October 2005 v1.0	Initial release of *Script-Based Design for Hardcopy II Devices.*	—