PlanAhead Software Tutorial

Using Tcl and SDC Commands

UG 760 (v 12.3) September 21, 2010

Demo Design License

The PlanAhead™ software source code includes the source code for the following programs:

Centerpoint XML

- The initial developer of the original code is CenterPoint – Connective Software
- Software Engineering GmbH. portions created by CenterPoint – Connective Software
- Software Engineering GmbH. are Copyright© 1998-2000 CenterPoint - Connective Software Engineering GmbH. All Rights Reserved. Source code for CenterPoint is available at http://www.cpointc.com/XML/

NLView Schematic Engine

- Copyright© Concept Engineering.

Static Timing Engine by Parallax Software Inc.

- Copyright© Parallax Software Inc.

Java Two Standard Edition

- Includes portions of software from RSA Security, Inc. and some portions licensed from IBM are available at http://oss.software.ibm.com/icu4j/
- Powered By JIDE – http://www.jidesoft.com

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

Free IP Core License

This is the Entire License for all of our Free IP Cores.

Redistribution and use in source, netlist, binary and silicon forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of ASICS World Services, the Authors and/or the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Table of Contents

PlanAhead Software Tutorial

Using Tcl and SDC Commands

Introduction

This tutorial shows you how to use the Xilinx® PlanAhead™ software to write scripts with the Tool Command Language (Tcl) API. It assumes that you are familiar with the PlanAhead software Graphical User Interface (GUI) and project flows. To familiarize yourself with PlanAhead perform the *PlanAhead Tutorial: Quick Front-to-Back Overview* (UG673) before starting this tutorial:

http://www.xilinx.com/support/documentation/dt_planahead_planahead12-3_tutorials.htm

Many PlanAhead features are covered in more detail in other tutorials, and not every command or command option is covered. The tutorial uses the features contained in the PlanAhead software product, which is bundled as a part of the ISE® Design Suite.

Sample Design Data

This tutorial uses sample design data included with PlanAhead software release package. The tutorial design data is located in the following directory:

`<ISE_install_Dir>/PlanAhead/testcases/PlanAhead_Tutorial.zip`

Save and extract the zip files to any write-accessible location. The location of the unzipped `PlanAhead_Tutorial` data is referred to as the `<EXTRACT_DIR>` throughout this document.

The tutorial sample design data is modified while performing this tutorial. A new copy of the original `PlanAhead_Tutorial` data is required each time you run the tutorial. For more information about the example design, see the *Tutorial Description* section.

Xilinx ISE Design Suite and PlanAhead Software

PlanAhead software is installed with ISE Design Suite by default. Before beginning this tutorial, ensure that PlanAhead is operational, and that the sample design data has been installed. For installation instructions and information, see the *ISE Design Suite 12: Installation, Licensing, and Release Notes*:

http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_3/irn.pdf

Required Hardware

Xilinx recommends 2GB or more of RAM for use with PlanAhead on larger devices. For this tutorial, a smaller design was used, with a limited number of designs open at any one time. 1GB of RAM should be sufficient, but it could impact performance.

PlanAhead Documentation and Information

For information about the PlanAhead software, see the following documents, which are available with your software:

- *PlanAhead User Guide* (UG632) - Provides detailed information about the PlanAhead software. http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_3/PlanAhead_UserGuide.pdf
- *Floorplanning Methodology Guide* (UG633) - Provides floorplanning hints. http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_3/Floorplanning_Methodology_Guide.pdf
- *Hierarchical Design Methodology Guide* (UG748) - Provides an overview of the PlanAhead hierarchical design capabilities. http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_3/Hierarchical_Design_Methodology_Guide.pdf
- For additional information about PlanAhead, including video demonstrations, go to http://www.xilinx.com/planahead.

Tcl Syntax

The Tcl syntax is shown with a prefixed right angle symbol **>** in bold, which is to prompt you to type specific commands into the Tcl Console window. An example is:

```
>get_cells cpuEngine
```

```
cpuEngine
```

The **>** character indicates the text to be typed, and below it is the expected response.

Tutorial Description

The sample design used in this tutorial consists of a typical system on a chip design with a RISC CPU core connected to several peripheral cores using a Wishbone bus arbiter. The design targets an xc6vlx75Tff784 device. This tutorial uses a project file which has already synthesized the HDL and is ready to be used. If you have any questions or comments about this tutorial, contact Xilinx Technical Support.

Tutorial Objectives

After completing this tutorial, you will have:

- Used a sample design to explore the Tcl Console, the relationship between the GUI and the Tcl commands, and the log and journal files.
- Opened a PlanAhead project using Tcl and batch scripts.
- Become familiar with the PlanAhead GUI, the Tcl Console, and online help for Tcl commands.
- Learned about the different execution modes of PlanAhead.
- Explored some of the basics of Tcl built-in commands.
- Created batch-mode project creation and flow execution scripts.
- Explored Tcl objects, properties, and physical constraints.
- Performed a simple conversion of UCF timing constraints to Synopsys Design Constraint (SDC) equivalents and explored incremental static timing analysis reporting.

Tutorial Steps

This tutorial is separated into steps, followed by general instructions and supplementary detailed substeps that let you make choices based on your skill level as you progress through the tutorial. This tutorial contains code snippets that you can type into PlanAhead at the Tcl Console. If you need help completing a general instruction, go to the detailed steps below it, or if you are ready, simply skip the step-by-step directions and move on to the next general instruction.

This tutorial has the following primary steps:

Step 1: Open a Project

Step 2: GUI, Tcl Console and online help

Step 3: Run Scripts in GUI, Batch, and Interactive Shell Modes

Step 4: Use Tcl built-ins and basic commands

Step 5: Create New Project and Flow Control

Step 6: Use Netlist Objects, Properties, and Physical Constraints

Step 7: Use Basic Static Timing Analysis and SDC

Step 1: Open a Project Step 1

PlanAhead enables several types of projects to be created depending on the location in the design flow where the software is being used. RTL sources or synthesized netlists can be used to create a Project for development, analysis, or to take all the way through implementation and bit file creation. This tutorial uses a synthesized netlist project which is not yet implemented.

1-1. Open the software.

- On Windows, select the Xilinx PlanAhead 12 desktop icon, or select **Start > All Programs > Xilinx ISE Design Suite 12.3 > PlanAhead > PlanAhead**.
- On Linux, change the directory to `<EXTRACT_DIR>/PlanAhead_Tutorial/Tutorial_Created_Data`, and enter **`planAhead`**.

The PlanAhead Getting Started Help page opens.

1-2. Open the example project_cpu_netlist Project.

You will modify the example design project netlist during this tutorial. You will open the example design and then save the project to a different name so the original example design project can be re-used.

1-2-1. In the Getting Started page, select **Open Example Project > CPU** (Synthesized).

1-2-2. Select **File > Save Project As** to save the project with a different project name.

The **Save Project As** dialog box opens.

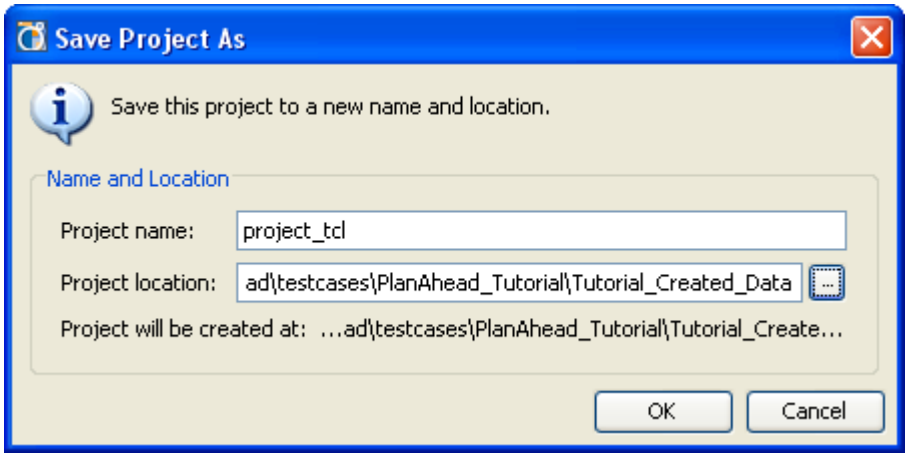

Figure 1: Saving the Project

1-2-3. In the **Project Name** text box, enter a unique name for the project, such as **`project_tcl`**.

1-2-4. Enter the following new Project location:

*<EXTRACT_DIR>***`/PlanAhead_Tutorial/Tutorial_Created_Data/`**

1-2-5. Click **OK**.

The Project Manager opens with the design sources displayed in the Sources view.

1-2-6. In the **Flow Navigator** on the left side of the PlanAhead Environment, click the **Netlist Design** button, which is outlined in red in Figure 2.

The Netlist Design opens and is ready to explore.

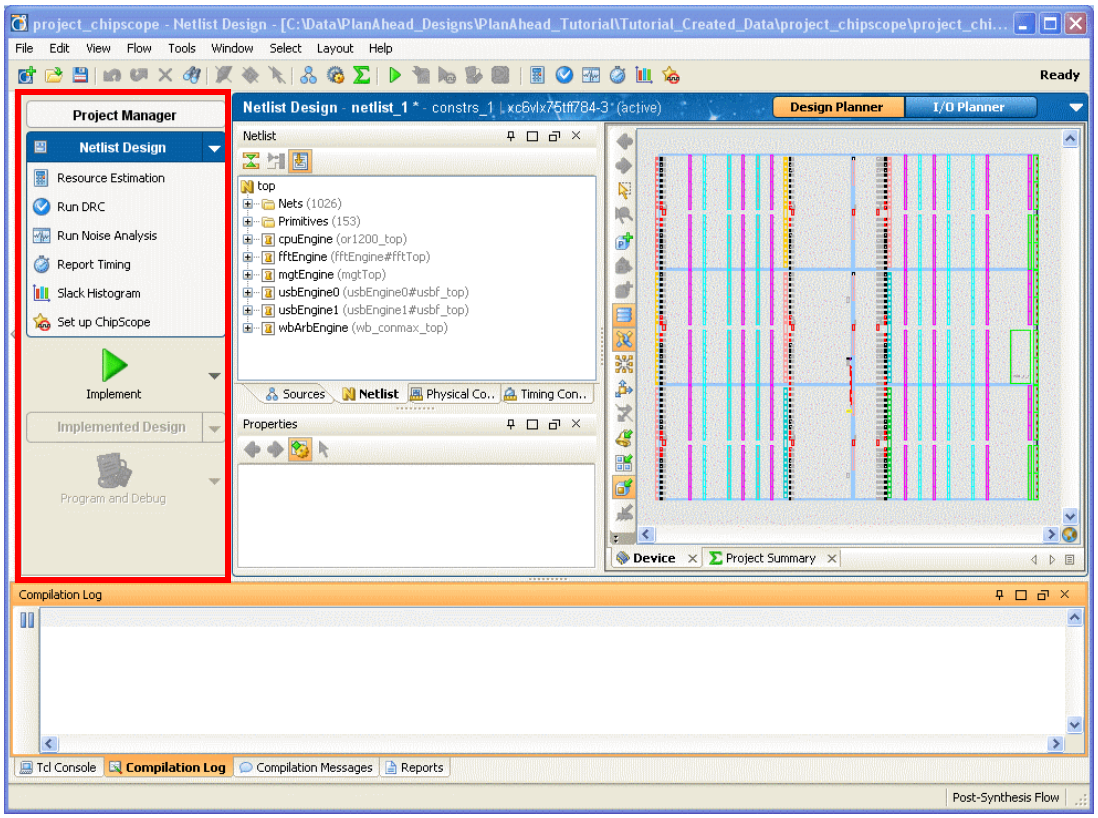

Figure 2: The Project in the Netlist Design Environment

Step 2: Explore the Tcl Console and Online Help Step 2

When you start PlanAhead in the default GUI mode, the bottom of the PlanAhead environment contains the Tcl Console and messages resulting from operations performed in the GUI. The Tcl Console contains a text box that you can type Tcl commands directly into, and a scrollable history view. The command text box is indicated by the red rectangle (Figure 3).

Figure 3: The PlanAhead Tcl Console

2-1. View the Tcl Console and Messages.

2-1-1. Type a Tcl command into the Tcl Console.

Notice the output printed in the Console history. The Tcl built-in `puts` command prints string messages to the Console and to the log files for information purposes. For example:

```
>puts "Hello!"
```

```
"Hello!"
```

2-2. Examine Online Help.

2-2-1. Click into the Tcl Console text box, and type in the following command:

```
>help
```

A complete list of PlanAhead Tcl commands prints to the screen along with a brief description of each command.

2-3. Get help on a specific command.

2-3-1. In the Tcl Console text box, type in the following:

```
>create_project -help
```

The complete help syntax for the `create_project` command is printed.

```
Description:
Create a new project
Syntax:
create_project  [-part <arg>] [-force] [-quiet] <name> <dir>
Returns:
new project object
Usage:
  Name    Optional  Default  Description
  ---------------------------------------
  -part   yes                Set the default Xilinx part for a project
  -force  yes                Overwrite existing project directory
  -quiet  yes                Ignore command errors
  <name>  no                 Project name
  <dir>   no                 Directory where the project file is saved
```

2-3-2. Type the following invalid command into the Tcl Console.

```
>junk
ERROR: invalid command name "junk"
```

The Tcl interpreter issues an error.

Note: Notice that a small red bar was placed next to the scroll bar to the right of the Tcl Console history to indicate an error occurred at this line. You can use this feature to scroll back through command history and quickly see if warnings or errors occurred in the context of any messages. Warnings are colored yellow and errors are red.

When you type commands into the Console, the Tcl interpreter looks for known commands and defined procedures. If none are found that match the command, then it sends the command to the OS shell for execution. If no known command is found, then an error is issued.

2-4. Type an OS Shell command.

2-4-1. In the Tcl Console, type:

- `dir` command (Windows).
- `ls` command (Linux)

```
>dir
```

The complete list of files in the current working directory displays. You can use this feature to access OS-specific commands from directly inside the interpreter, or alternatively use the Tcl built-in `exec` command.

2-5. Examine PlanAhead Log and Journal Files.

Each time you invoke PlanAhead, two files are created that are very useful for understanding the creation of Tcl scripts. These files are the journal file (`.jou`) and the log (`.log`) file.

- The journal file contains the history of all commands executed in your session, either interactively in the GUI, or by using GUI commands that have a Tcl equivalent.
- The log file contains all the journal commands, but also has the information, warning, and error messages as well to provide context for each executed command.

The journal file is useful for learning Tcl syntax for a given command; you can use the GUI and look at this file for the expected Tcl syntax for most operations you perform.

Note: The journal and log files are located in the "start-in" directory, which is the current working directory where you invoked the PlanAhead executable for Linux. For Windows, this directory is defined by the `%APPDATA%` environment variable, under an HDI subdirectory, which is normally mapped to the following location: `C:\Documents and Settings\<user_name>\Application Data\HDI`

2-5-1. View the `planAhead.log` and `planAhead.jou` files.

Notice the commands you executed above and any info, warning, and error messages.

Note: Be aware that each time you invoke PlanAhead, it overwrites the journal and log files. Keep this in mind if you want to save these files for future reference.

Step 3: Running Tcl Scripts Step 3

So far in this tutorial, you have been working with PlanAhead in its default GUI mode. PlanAhead has three operating modes:

- GUI
- Interactive Shell
- Batch Mode

This section explores the different modes for executing Tcl commands and scripts.

3-1. Execute a Tcl script in GUI Mode.

The default mode for PlanAhead is in GUI mode. Tcl commands can be typed directly in the Tcl Console, or may be sourced from the Tools menu pull down.

3-1-1. Open `step3.tcl` in any text editor, such as notepad, EMACS, or VI, and enter the following command in it:

```
puts "Hello World!"
```

3-1-2. In the GUI, click **Tools > Run Tcl Script** to launch the Run Script dialog box as shown in Figure 4.

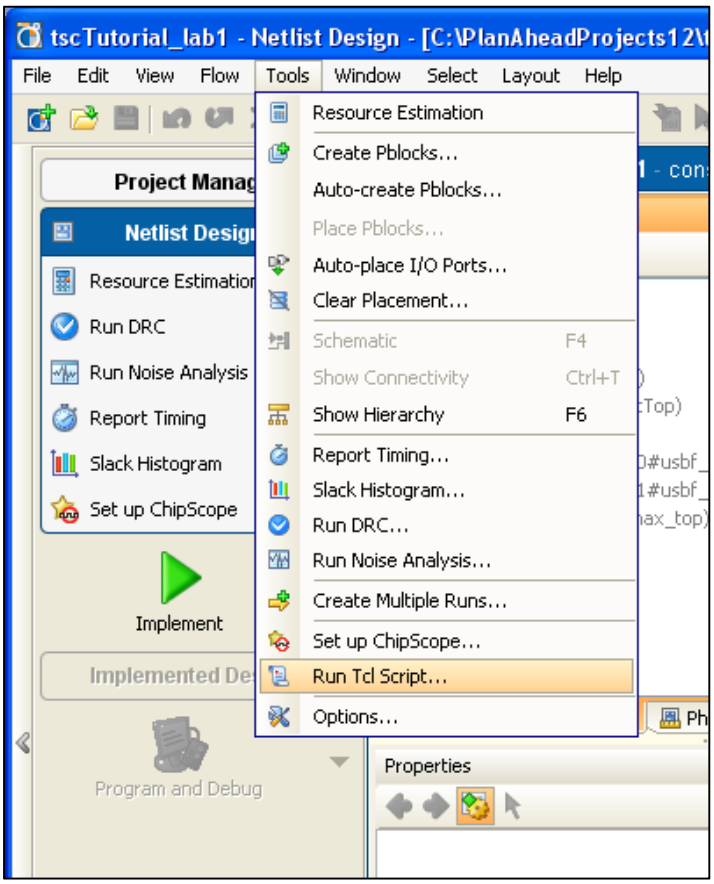

Figure 4: Launching the Run Tcl Script Dialog Box

3-1-3. Navigate to the directory where you created the file `step3.tcl`, select the file, and click **Open**.

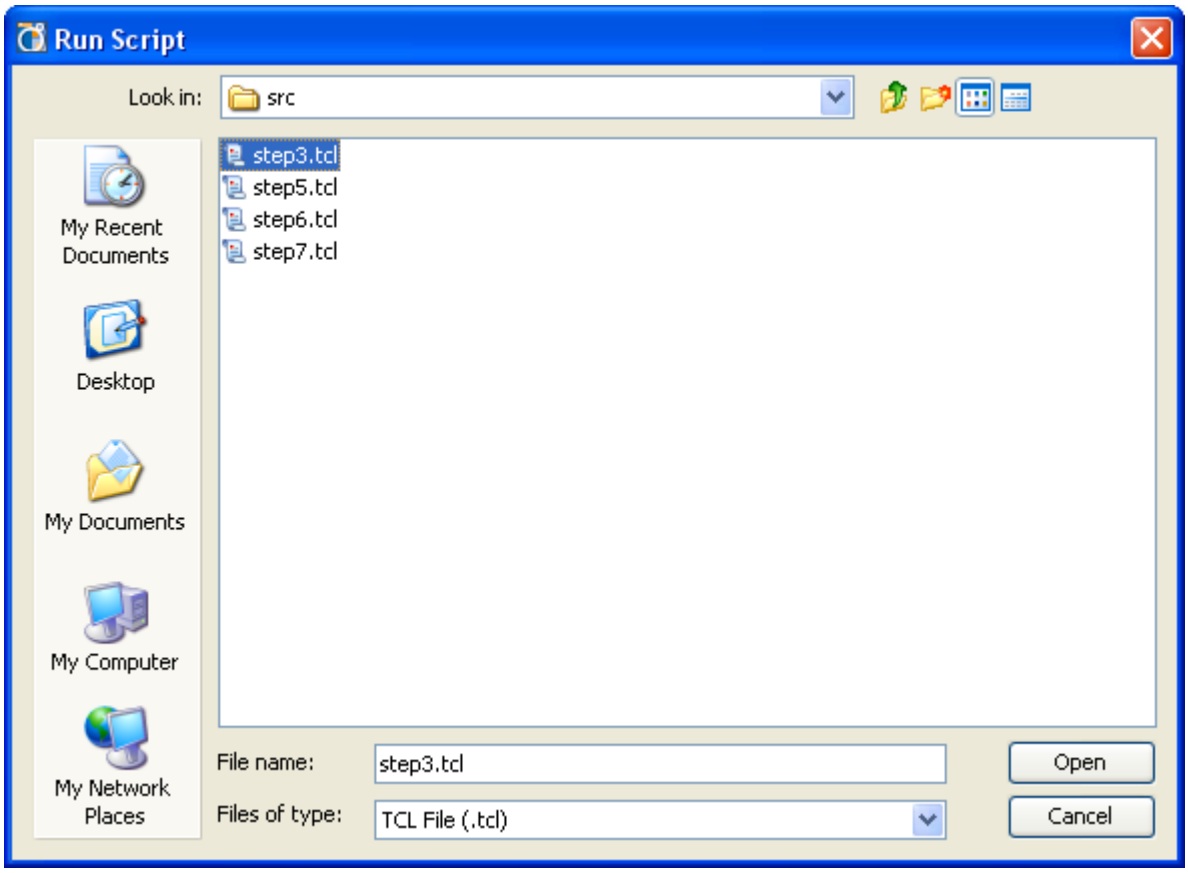

Figure 5: Run Script Dialog

"Hello World!" prints in the Tcl Console.

In addition to invoking Tcl scripts from the GUI, you can also invoke scripts from the command line, even when launching PlanAhead in GUI mode.

3-1-4. Exit PlanAhead by selecting **File > Exit** or typing **`exit`** in the Tcl Console.

3-2. Source a script from the command line.

3-2-1. You can execute Tcl commands when you launch PlanAhead. Enter the `-source` command line option at a DOS terminal (cmd) or Linux shell prompt from the directory you saved your file in, as follows:

```
>planAhead -source step3.tcl
```

This launches PlanAhead in GUI mode and sources the specified Tcl script. When execution completes, GUI control is returned to the user.

Note: This assumes the PlanAhead executable is installed and resides in your command search path ($PATH in Linux, and %Path% in Windows).

If the PlanAhead GUI is not accessible, you can use any of the following options:

- Source the `settings32.bat` or `settings64.bat` (or `.sh` for Linux) environment setup script provided with the ISE installation
- Prefix the PlanAhead command with the fully qualified path to the installation directory for PlanAhead
- Modify the environment variable PATH to make sure the `<PlanAhead_install_dir>/PlanAhead/bin` directory is in the executable search path.
- Notice that you do not see the "Hello World" message that the script prints to the Tcl Console area of the PlanAhead GUI until you open or create a project or until you expand the Tcl Console via the button on the bottom left of the PlanAhead 12.3 main screen.

3-2-2. Exit PlanAhead by selecting **File > Exit** or typing **`exit`** in the Tcl Console.

3-3. Use the interactive shell mode.

PlanAhead offers an interactive Tcl shell mode for non-GUI interactive sessions.

3-3-1. To run in the interactive shell mode invoke PlanAhead with the `-mode` switch, as follows:

```
>planAhead -mode tcl
```

The PlanAhead shell prompt (`PlanAhead%`) displays and is ready for you to type commands directly into the tool, without the GUI. All Tcl commands are available "live," and you can interactively run commands and query projects and designs as you could in the GUI.

3-3-2. Another option to the interactive shell mode (3-3-1) is to run PlanAhead in Tcl mode and also source a starting script. This would be done as follows:

```
>planAhead -mode tcl -source step3.tcl
```

3-3-3. Type in the help command to view command help in the interactive shell, as follows:

```
>help
```

3-3-4. Exit PlanAhead from interactive mode by typing the exit command, as follows:

```
>exit
```

3-4. Use the Batch Mode Command.

The batch mode is similar to the interactive mode in that it does not invoke the GUI, but requires the `-source` option with a valid Tcl script. Batch mode runs the Tcl script, and then exits as if there was an `exit` command on the last line of your script.

3-4-1. Run the `step3.tcl` file by sourcing it in batch mode, as follows:

```
>planAhead -mode batch -source step3.tcl
```

PlanAhead is invoked. It runs the script, then shuts down without offering any interactive shell prompt to type commands.

Step 4: Explore Basic Tcl Built-In Commands and Syntax Step 4

A complete background on all of the capabilities of Tcl is beyond the scope of this tutorial. Here are some of the basics to help you understand the rest of this tutorial.

- For a more complete reference and tutorial on the basics of Tcl, refer to the following web site: http://www.tcl.tk/doc/.
- An introductory tutorial is available at: http://www.tcl.tk/man/tcl8.5/tutorial/tcltutorial.html
- Documentation for specific built-in (non-PlanAhead) commands are available from: http://www.tcl.tk/man/tcl8.5/TclCmd/contents.htm

Note: PlanAhead has integrated the latest release of Tcl, version 8.5.

4-1. Understand the Tcl Interpreter and basic syntax.

Tcl is an interpreted script language; there is no need to compile Tcl code to machine or assembly code. There is an interpreter, which takes Tcl commands as input and evaluates them according to the language semantics. The interpreted nature of the language makes for a very powerful interface to EDA tools, because it allows you to directly type in commands that "ask questions" of the design, which the tool responds to with an answer.

The Tcl Console in the GUI and the "planAhead%" prompt in the interactive shell mode are the interface to the interpreter. You can type commands into these prompts and those command are passed to the interpreter for evaluation. You have been interacting with the interpreter in the previous sections of this tutorial.

The basic syntax for Tcl commands is as follows:

<*command*> <*options*>

Commands are evaluated sequentially (if in a script with multiple commands), and the command is evaluated left to right.

4-2. Understand variables and substitutions.

You can create variables in Tcl with the `set` command. The Tcl language is "loosely typed" which means you can create variables that hold data without undue concern about what type of data the variable contains.

Now lets try some Tcl programming.

4-2-1. Launch PlanAhead in Tcl mode (**`>planAhead -mode tcl`**) or in a Tcl shell if you have that installed (`tclsh`).

4-2-2. Create a simple string variable by typing:

```
>set var1 "Hello World!"
Hello World!
```

The string "Hello World!" echoes to the Console history.

4-2-3. Set variables to strings, integers, or floating point numbers with the `set` command as follows:

```
>set var2 1
1
```

The value to `var2` is set to 1, and echoes on the Tcl Console.

```
>set var3 3.14
3.14
```

The value of `var3` is set to 3.14, and echoes on the Tcl Console.

4-2-4. Set new variables using other variables using the concept of substitution as follows:

```
>set var4 "$var1"
Hello World!
```

In the above example the "$" sign tells the interpreter to look for a variable with the given name and substitute its value before doing the assignment to the var4. The "$" char is a special character in Tcl, and it is important to take note of it. Here is a list of the special characters in Tcl:

Character	Name	Behavior
[ ]	Square Brackets	Nests one command into another
{ }	Curly Braces	Literal string
" "	Double Quotes	String that allows variable and command substitution.
\	Backslash or Escape Character	Use before special characters so the interpreter does not attempt to read
;	Semicolon	End of a command
#	Pound Sign	Comment

Table 1: Special Characters in Tcl

To prevent substitution you can use either of the following mechanisms:

- Literal Strings with curly braces {}
- Back-slashes before special characters

4-2-5. To set a variable with the character for a dollar sign in it without the interpreter trying to substitute a variable, you can do either of the following:

```
>set var5 {$var4}
$var4
```

or

```
>set var6 "\$var4"
$var4
```

4-3. About conditional statements.

Tcl supports conditional execution with `if` statements. To see how this works,

4-3-1. Type the following into the Tcl Console:

```
>if {$var2 == 1} {puts "Yay!"}
Yay!
```

In this example, the curly braces {} indicate the "body" of other statements. The interpreter tests to see if the variable named var2 equals 1, which it does, and it executes all of the commands in the second set of braces.

4-4. Add additional nested ifs with the elseif command.

4-4-1. Type the following into the Tcl Console:

```
>if {$var2 == 2} {puts "Yay!"} elseif {$var3 == 3.14} {puts "Nay!"}
Nay!
```

4-4-2. Provide a final else value by typing the following:

```
>if {$var2 == 2} {puts "Yay!"} else {puts "Nay!"}
Nay!
```

4-5. About lists and looping.

In the same way that Tcl variables are loosely typed, there is a notion of "container" variables that hold more than one value. Lists are also similarly "loose" and can be as simple as a string with values separated by spaces. For example:

Command	Returns	Description
>set colors "red blue green"	red blue green	
>set colors2 [list brown black white]	brown black white	Use the explicit list command to build up a list.
>llength $colors	3	Returns the number of items in the $colors list.
>lindex $colors 0	red	Returns the value of the first item in the $colors list.
>lindex $colors2 end	white	Returns the value of the last item in the $colors2 list.

Table 2: Examples of Tcl Variables

The most common way to iterate through a list is with the `foreach` command.

4-5-1. To operate on each of the colors in the colors list in Table 2, type the following into the Tcl Console:

```
>foreach c $colors {puts $c}
red
blue
green
```

The prints out each color as it loops through the list.

4-6. About nesting commands.

Nesting of commands is embedding multiple commands inside other commands using the square bracket special characters. Nested commands get executed by the interpreter from the inner-most scope of the commands outward. A common way to use nesting of commands is shown by the math `expr` command:

```
>set var7 [expr 1 + 1]
2
>set var8 [expr $var3 / 10]
0.314
>set var9 [expr [expr 2 * 3] + 1]
7
```

You can nest any command, and the square brackets tell the interpreter to evaluate the commands in the enclosed brackets and then substitute the resulting value into the next command.

4-7. Understand error handling.

Errors in Tcl scripts normally halt script execution. There are built-in mechanisms to handle error trapping, so that you can decide to continue execution. The `catch` command takes any command as an argument and returns 1 if there was an error:

4-7-1. Enter the following set of commands:

```
>catch "junk" result
1
>puts $result
invalid command name "junk"
```

The results of a catch command can be combined with if statements to handle errors:

```
>if {[catch "junk" result]} {puts "ERROR_IN_MY_SCRIPT!"}
ERROR_IN_MY_SCRIPT!
```

The catch command traps the error resulting from the unknown "junk" command and executes the commands in the block of code with the puts command in it.

4-7-2. Now exit PlanAhead or the Tcl Shell.

```
>exit
```

Step 5: Create A New Project; Run Synthesis and Implementation Step 5

Basic project creation starts with the `create_project` command. There are a few options required for project creation, and these include:

- Name of the project
- Directory in which to create the project
- Default part the project will target

To do this through in Tcl, you will create a Tcl script to perform batch mode creation of a small project and run RTL synthesis and implementation.

5-1. Create a project.

5-1-1. Create a new file named `step5.tcl`

5-1-2. Type (or copy) the following basic variable definitions in the `step5.tcl` file to use later. Be sure to modify the <*EXTRACT_DIR*> to the correct path:

```
set projDir [file dirname [info script]]

set srcDir  <EXTRACT_DIR>/PlanAhead_Tutorial/Sources/hdl/

set projName usbf

set topName usbf_top

set device xc6vlx75tff484-1
```

In these commands:

- The [`info script`] command returns the full filename of the script executed by the Tcl interpreter.
- The filename is passed to the `file dirname` command, which returns the directory in which the script is located. You are creating the project in the same directory location in which the script resides.

5-1-3. Check to see if a directory already exists, and if it does remove it so there is a clean run, by typing the following :

```
if {[file exists $projDir/$projName]} {
   # if the project directory exists, delete it and create a new clean
one
   file delete -force $projDir/$projName
}
```

5-1-4. Use the `create_project` command to create a new project, using the variables created in Step 5.1.2 for the project name, directory, and the target part, by typing the following:

```
create_project $projName $projDir/$projName -part $device
```

5-1-5. Set the `design_mode` property on the source set:

```
set_property design_mode RTL [get_filesets sources_1]
```

This will sets the project to be an RTL project, which is a container object for all the RTL source files. If you were creating a netlist-based project, based on a post-synthesis netlist, this property value would be Netlist instead of RTL.

5-1-6. Define the RTL sources to add to the project, by typing the following:

```
set verilogSources [glob $srcDir/FifoBuffer.v $srcDir/async_fifo.v
$srcDir/rtlRam.v $srcDir/$projName/*.v]
```

Tcl provides a built-in command to query all files that match a wildcard search, called `glob`.

5-1-7. Use the `import_files` command to add the source files to the project and copy them locally, by typing the following:

```
import_files -fileset [get_filesets sources_1] -force -norecurse
$verilogSources
```

This command string imports the individual files into the project and puts those files in "container" objects called a fileset. The default fileset is named sources_1, doing this copies the original files local to the project, thus preserving the originals.

In this command,

- The `-force` option overwrites any previous sources by the same name
- The `-norecurse` option tells PlanAhead not to recursively search every subdirectory and add any additional files it finds.

5-1-8. Set the name of the top-level module or entity, so the synthesis engine knows what the top-level to synthesize is:

```
set_property top $topName [get_property srcset [current_run]]
```

5-1-9. Save the file, and execute PlanAhead sourcing the step5.tcl script you just created:

```
>planAhead -mode batch -source step5.tcl
```

PlanAhead invokes and creates the project based on the script.

5-2. Add commands to synthesize the project using the default synthesis strategy.

In the previous section, you created a script to create an RTL project. In this step, you will add commands to perform synthesis.

5-2-1. Open `step5.tcl` in a text editor, and add the following commands to the end of the script, after the `set_property` top command:

```
launch_runs -runs synth_1
```

In the PlanAhead GUI, when you launch a synthesis or implementation run, PlanAhead launches the process in a separate thread, so that you can continue to use the GUI to analyze your design.

Tcl is the same, and without doing anything special, synthesis would run. Because you are running in batch mode, you must enter commands to block further execution until the synthesis run completes, so that the next step, which would be to launch implementation, works correctly:

5-2-2. Enter the command to block execution as follows:

```
wait_on_run synth_1
```

5-2-3. Save the `step5.tcl` file.

5-2-4. Execute PlanAhead and source the script you just created:

```
>planAhead -mode batch -source step5.tcl
```

PlanAhead deletes the previous project, re-creates it again, and runs through synthesis using the default synthesis strategy. PlanAhead provides a number of built-in strategies for synthesis, choosing a different one is as simple as setting the strategy property on the synthesis run, and recompiling.

5-2-5. Add the following command to `step5.tcl` before the `launch_runs synth_1` command:

```
set_property strategy PowerOptimization [get_runs synth_1]
```

This chooses the power optimization strategy.

5-2-6. Save the `step5.tcl` file.

5-2-7. Execute PlanAhead, sourcing the script you just created as follows:

```
>planAhead -mode batch -source step5.tcl
```

PlanAhead deletes the prior project, creates a new one, and synthesizes using the power optimization strategy.

5-3. Launch Implementation.

In the previous steps, you created a script to create a project and synthesize it using XST.

Next, you will add to this script, and launch an implementation run.

5-3-1. Open `step5.tcl` in a text editor, and add the following commands to the end of the script, after the `wait_on_run synth_1` command:

```
launch_runs -runs impl_1
wait_on_run impl_1
```

Similar to the synthesis run, these two commands launch the implementation run using the default strategy and block until completion.

5-3-2. Save the `step5.tcl` file.

5-3-3. Execute PlanAhead sourcing the script you just created:

```
>planAhead -mode batch -source step5.tcl
```

This script deletes the previous project, recreates a new one, and runs through implementation with the default strategy.

5-3-4. If you want to use a different implementation strategy such as the timing-driven map flow, add the following command before the `launch_runs -runs impl_1` command:

```
set_property strategy MapTiming [get_runs impl_1]
```

5-3-5. Save the `step5.tcl` file.

5-3-6. Execute PlanAhead sourcing the script you just created:

```
>planAhead -mode batch -source step5.tcl
```

This script deletes the previous project, recreates a new one, and runs through implementation with the `map -timing` flow.

5-4. Open the Implementation Results.

Once a run has completed, opening the design brings the netlist for the design into memory as an active design, so you can perform further Tcl commands and operations.

5-4-1. To open the post-implementation netlist, add the following to the end of `step5.tcl`:

```
open_impl_design
```

5-4-2. Save the `step5.tcl` file.

5-4-3. Execute PlanAhead sourcing the script you just created:

```
>planAhead -mode tcl -source step5.tcl
```

When the **`PlanAhead%`** prompt appears, start the GUI by typing the following:

```
PlanAhead% start_gui
```

The script deletes the previous project, recreates a new one, runs through implementation flow, and then opens the implementation result design for further analysis and operation. Next, you launched the GUI for further interaction.

5-4-4. Exit PlanAhead.

```
>exit
```

Step 6: Explore Netlist Objects, Properties, Physical Constraints Step 6

In the previous step, you created a project and executed a simple flow using Tcl in batch mode. In this step, you will explore a bit more of the netlist access commands using core SDC commands.

6-1. Open the Project.

6-1-1. In a text editor, create a file called `step6.tcl` in the same directory as you used before and type (or copy) the following commands into the file. Remember to replace <*EXTRACT_DIR*> with the correct path.

```
set projDir [file dirname [info script]]
set srcDir <EXTRACT_DIR>/PlanAhead_Tutorial/Projects/
set projName project_cpu_netlist
set topName top
set device xc6vlx75tff484-1
# open existing project
open_project $srcDir/$projName/$projName.ppr
# now open the post-synthesis netlist design
open_netlist_design
```

6-1-2. Save `step6.tcl` and source it to load the project in PlanAhead in the default GUI mode:

```
>planAhead -source step6.tcl
```

This launches PlanAhead, opens the project and loads the netlist design so that it is ready to accept further commands.

6-2. Explore objects and properties.

First, explore a few of the commonly used object types in PlanAhead. For flow control it is helpful to review properties of run objects. This is useful for querying projects to determine their current status. Maximize the Tcl Console as it will make seeing the results of entering commands easier.

6-2-1. Enter the following commands in the PlanAhead GUI Tcl Console to explore properties of run objects:

```
>set runList [get_runs]
>report_property [lindex $runList 0]
>get_property status [lindex $runList 0]
```

These commands illustrate the properties of runs that are able to be queried through Tcl. The status of the implementation run `impl_1` should be "**Not Started**" since we have not yet run implementation.

The most commonly used of the core SDC commands is the `get_cells` command, which provides a way to query instances in the netlist design by name. The `-hierarchical` switch instructs PlanAhead to apply the pattern supplied at each level of hierarchy in the design. The below command has the effect of returning every cell in the design.

6-2-2. Type the following commands into the Tcl Console to see properties of cell objects:

```
>set cellList [get_cells -hierarchical *]
>report_property [lindex $cellList end]
>get_property lib_cell [lindex $cellList end]
```

These commands query all cells in the design and does a property report on the last cell in the list. The final command returns the value of the primitive library cell property.

The `get_nets` command is another commonly used object query command, it takes a hierarchical search pattern also.

6-2-3. Type the following commands in the Tcl Console to experiment with net object properties:

```
>set netList [get_nets *]
>report_property [lindex $netList 0]
>get_property type [lindex $netList 0]
```

These commands query all the net objects at the top-level of hierarchy, and prints a report of the property values for the first one in the list. Finally, the type property is queried, which should return a **`Signal`** type.

Port objects are queries with the **`get_ports`** command.

6-2-4. Type the following commands to experiment with port objects and their properties:

```
>set portList [get_ports *]
>report_property [lindex $portList 0]
>get_property iostandard [lindex $portList 0]
```

These commands return all the top-level ports in the design, and prints out a list of usable properties on each for scripting. Finally, the I/O standard applied to a port can be queried with the `get_property` command, which should return LVCMOS25.

Pin objects can be queried with the **`get_pins`** command.

6-2-5. Type the following commands into the Tcl Console to explore pin object properties:

```
>set pin [get_pins OpMode_pad_0_o_0/D]
>report_property $pin
>get_property setup_slack $pin
```

These commands query a specific pin, the **`D`** input to a flop and report all the properties available on pin objects. Pin objects are tightly coupled to the static timing analysis engine, and you can query pins based on setup or hold slack properties. Quering timing analysis related properties will invoke the timing analysis engine which will build a timing graph. This is a useful debug and analysis feature.

6-3. Filter object queries with properties.

Properties can be combined with the `-filter` option to any `get_` command to filter out the list of returned objects based on specific criteria. Below are some examples of using object access commands with filtering.

6-3-1. Type the following commands into the PlanAhead Tcl Console:

```
>get_nets -filter {type == "Global Clock"}
>set cellList [get_cells * -hierarchical -filter "lib_cell =~ FD*"]
```

These commands query global clock nets and return a list of all cells in the design where the primitive type matches a string pattern that starts with FD. This returns all flip flops in a design, such as FDR primitives. Object `lib_cell` properties map directly to Unisim primitives.

Objects are related to one another through netlist connectivity: ports connect to nets, which connect to pins, which connect to cells.

The `-of` option to the object access commands are an important way to traverse netlist objects.

6-3-2. To explore these commands, type the following commands in the Tcl Console:

```
>set cell [get_cells fftEngine/control_reg_1]
>set pin [lindex [get_pins -of $cell -filter "direction == IN"] 1]
>set net [get_nets -of $pin]
>set driver [get_pins -of $net -filter "direction == OUT"]
>set driverCell [get_cells -of $driver]
```

These commands query connectivity relationships and are used to traverse the netlist based on a variety of properties and connectivity information. This is a powerful capability.

6-4. Explore the physical constraints

Exploring physical constraints is useful, but you need to be able to set these values in a manner consistent with the UCF. The following are command examples of setting physical constraints through Tcl instead of UCF.

6-4-1. Type the following commands in the Tcl Console to see the physical constraints being applied:

```
>set_property IOSTANDARD SSTL15 [get_ports cpuClk]
>set_property site IOB_X1Y72 [get_ports cpuClk]
>set_property is_fixed true  [get_ports cpuClk]
>set_property loc SLICE_X0Y73 [get_cells fftEngine/control_reg_1]
>set_property bel AFF [get_cells fftEngine/control_reg_1]
>set_property is_fixed true [get_cells fftEngine/control_reg_1]
```

These commands set physical constraints such as IOSTANDARDS, LOC, and BEL constraints. Most attributes that can be applied in HDL would propagate to attributes in an EDIF netlist which can be queried with `get_property` and are settable with `set_property`.

6-4-2. Close PlanAhead by either typing **`exit`** in the Tcl Console, by selecting **File > Exit**, or by clicking on the X in the upper right hand corner of the main window.

Step 7: Use Static Timing Analysis with Tcl and SDC Step 7

PlanAhead has a static timing analysis (STA) engine that is separate from TRACE, the ISE STA engine. The PlanAhead STA engine is compatible with SDC constraints and supports incremental timing analysis.

In this step, you will convert a simple UCF timing constraint file to SDC. SDC is currently only supported by PlanAhead, and timing constraints will not port forward to ISE implementation and static timing analysis tools. However, SDC is a very powerful analysis and debug tool for netlist exploration. SDC is used primarily for timing constraints, so in this section you will explore some of the basics of SDC, specifically clocking, IO constraints, and exceptions.

7-1. Setup a UCF Conversion.

In this lab, you will open one of the sample projects, rename the project, save it with a different name, and disable UCF based timing in preparation for applying SDC constraints.

7-1-1. In a text editor, create a file called `step7.tcl` in the same directory as you used before and type (or copy) the following commands into the file. Be sure to replace the <*EXTRACT_DIR*> with the appropriate path to the directory for the `$srcDir` variable.

```
set projDir [file dirname [info script]]
set srcDir <EXTRACT_DIR>PlanAhead_Tutorial/Projects/
set projName project_cpu_netlist
set topName top
set device xc6vlx75tff484-1

# open new project
open_project $srcDir/$projName/$projName.ppr

# rename the project
set projName ${projName}SDC

# save it to a new name and location
if {[file exists $projDir/$projName]} {
   # if the project directory exists, delete it and create a new clean
one
   file delete -force $projDir/$projName}
save_project_as $projName $projDir/$projName

# now disable all ucf files - in preparation for SDC
set_property is_enabled false [get_files *.ucf]

# now open the implementation results
open_netlist_design
```

7-1-2. Save `step7.tcl` and source it to load the project in PlanAhead in the default GUI mode:

```
>planAhead -source step7.tcl
```

This command opens the design, saves it with a new name, and brings up the GUI with the design open. You can alternatively open the project manually with the GUI and rename the project.

7-1-3. The following are some UCF commands to constrain the design. You will convert these commands to SDC equivalent.

```
# Timing Constraints:
TIMESPEC TS_cpuClk = PERIOD "cpuClk" 13 ns;
NET "cpuClk" TNM_NET = "cpuClk";

TIMESPEC TS_wbClk = PERIOD "wbClk" 9 ns;
NET "wbClk" TNM_NET = "wbClk";

TIMESPEC TS_usbClk = PERIOD "usbClk" 5.25 ns;
NET "usbClk" TNM_NET = "usbClk";

TIMESPEC TS_phy_clk_pad_0_i = PERIOD "phy_clk_pad_0_i" 11 ns;
NET "phy_clk_pad_0_i" TNM_NET = "phy_clk_pad_0_i";

TIMESPEC TS_phy_clk_pad_1_i = PERIOD "phy_clk_pad_1_i" 11 ns;
NET "phy_clk_pad_1_i" TNM_NET = "phy_clk_pad_1_i";

TIMESPEC TS_fftClk = PERIOD "fftClk" 7 ns;
NET "fftClk" TNM_NET = "fftClk";
```

7-2. Managing Clocks.

7-2-1. Create a new file called `top.sdc` with your preferred text editor.

You will source this file repeatedly to demonstrate adding and debugging timing constraints with incremental static timing analysis.

The equivalent of a `TIMESPEC PERIOD` constraint from UCF is the SDC `create_clock` command.

7-2-2. Type the following into the `top.sdc` file:

```
create_clock -name cpuClk -period 13 [get_ports cpuClk]
```

The first command creates a clock with a period of 13ns, with a rising edge at 6.5ns rooted on the top-level port named cpuClk.

7-2-3. Add in the other clocks by typing the following into the `top.sdc` file as well:

```
create_clock -name wbClk -period 9 [get_ports wbClk]

create_clock -name usbClk -period 5.25 [get_ports usbClk]

create_clock -name phy_clk_pad_0_i -period 11 [get_ports
phy_clk_pad_0_i]

create_clock -name phy_clk_pad_1_i -period 11 [get_ports
phy_clk_pad_1_i]

create_clock -name fftClk -period 7 [get_ports fftClk]
```

7-2-4. Save `top.sdc` in the current working directory.

7-2-5. In the PlanAhead GUI Tcl Console, source `top.sdc` to create the clocks.

```
>source top.sdc
```

7-2-6. See the timing results with your clock by running the `report_timing` command:

`>report_timing`

You will see a path trace printed in the Tcl Console window which shows the path from the input port specified.

7-3. Input Constraints.

Now, you will set up the input timing relationship that equates to the UCF OFFSET IN constraints. Here is an example:

```
NET "DataIn_pad_0_i[0]" OFFSET = IN 3 ns VALID 7 ns BEFORE "TS_usbClk" RISING;
```

7-3-1. In `top.sdc`, add the following statements to the end of the file:

`set_input_delay -clock usbClk -max 2.25 [get_ports {DataIn_pad_0_i[0]}]`

`set_input_delay -add_delay -clock usbClk -min 4 [get_ports {DataIn_pad_0_i[0]}]`

The first constraint states that the `DataIn_pad_0_i[0]` signal arrives 2.25 ns after the rising edge of the `usbClk` clock domain. The second constraint provides the min-delay (hold analysis) equivalent to the `VALID` window on the `OFFSET IN`. The `-add_delay` option to the second constraint tells the engine to preserve the previously defined max delay when adding the min delay value.

7-3-2. Save `top.sdc` in the current working directory.

Enter the following in PlanAhead provides a command to delete all timing constraints, so you can source the same SDC file over and over.

7-3-3. the Tcl command text box, type:

`>reset_timing`

`>source top.sdc`

7-3-4. And to immediately see the timing results with your clock, run the `report_timing` command:

`>report_timing -from [get_ports {DataIn_pad_0_i[0]}]`

Notice how the timing path changed because of the `set_input_delay` command we added.

7-4. Output Constraints.

Next, you should constrain the output signal requirements according to the specification in the OFFSET OUT constraint. The equivalent in SDC is the `set_output_delay` command. Here is an example from UCF for this project:

```
NET " DataOut_pad_0_o[0]" OFFSET = OUT 4 ns AFTER "TS_usbClk" RISING;
```

7-4-1. At the end of the `top.sdc` file, add the following constraint:

`set_output_delay -clock usbClk 1.25 [get_ports {DataOut_pad_0_o[0]}]`

7-4-2. Save `top.sdc` in the current working directory.

7-4-3. In the PlanAhead GUI Tcl Console, reset timing and source `top.sdc` to recreate the constraints by entering the following in the Tcl command text box:

```
>reset_timing; source top.sdc
```

Note: The semicolon in the previous command is a way to combine multiple commands on the same line. The semicolon instructs the Tcl interpreter to execute the `reset_timing` command first, then source `top.sdc`.

7-4-4. To see the timing results with the new setting immediately, run the `report_timing` command:

```
>report_timing -to [get_ports {DataOut_pad_0_o[0]}]
```

Until this point, you have been deleting the timing graph (with `reset_timing`) and recreating the timing analysis graph each time you modified `top.sdc`. This is not necessary, and you can use incremental STA capabilities by typing constraints and exceptions directly into the Tcl Console.

7-5. Use Multicycle Paths.

Next, you will convert a multi-cycle path from UCF to the SDC equivalent. Multi-cycle paths are equivalent to the FROM/TO constraints in UCF with a TIMESPEC multiplier. Here is an example for this project:

```
# Multi-cycle paths for ALU:
NET "cpuEngine/or1200_cpu/or1200_alu/*" TPTHRU = "GRP_ALU_DATAOUT";
TIMESPEC TS_ALU_MCP = FROM "cpuClk" THRU "GRP_ALU_DATAOUT" TO "cpuClk" TS_cpuClk
* 2;
```

To relax timing to flip-flops and give it two clock cycles to meet setup, you would use the SDC command `set_multicycle_path`.

7-5-1. Type the following into the PlanAhead GUI Tcl Console:

```
>report_timing -through [get_pins cpuEngine/or1200_cpu/or1200_alu/*]
>set_multicycle_path -through [get_pins cpuEngine/or1200_cpu/or1200_alu/*] 2
>report_timing -through [get_pins cpuEngine/or1200_cpu/or1200_alu/*]
```

The first command performs and incremental STA update to generate a timing report through a number of pins of the design.

The second command sets the new constraint on the paths.

The third command updates only the portions of the timing graph affected by the new constraint and performs another incremental STA update. Notice the required time for this path shifted from 13ns to 26ns, giving this path 2 full clock periods to meet timing.

7-6. Use False Paths.

Similar to multicycle paths, for false paths we can incrementally add the SDC equivalent of TIG constraints to the UCF. The command to do this is called `set_false_path`. Suppose for the sake of analysis you wished to ignore all paths to the output pins. No timing between clock domains is the default for timing performed by TRACE with UCF constraints. With SDC-based timing, however, the default assumption is that all clocks are related. To get the same behavior with SDC you must explicitly set false paths between unrelated clock domains.

7-6-1. Enter the following constraints directly into the PlanAhead GUI Tcl Console:

```
>report_timing -from [get_clocks cpuClk]
>set_false_path -from [get_clocks cpuClk] -to [get_clocks wbClk]
>set_false_path -from [get_clocks wbClk] -to [get_clocks cpuClk]
>report_timing -from [get_clocks cpuClk]
```

The first `report_timing` command shows a failing path, but it is between the cpuClk and the wbClk, which is not a valid path. We set two false path commands between these clock domains and rerun the timing. Notice that there is now no timing calculation done between the clock domains.

7-6-2. Exit PlanAhead.

```
>exit
```

Note: As with all timing exceptions, you need to take care with false and multi-cycle path constraints. You must be certain that these commands are correct, because you are overriding the default analysis. It is still possible to have a valid path violation that would be hidden because of a timing exception that was added.

Conclusion

In this tutorial, you:

- Used a sample design to explore the Tcl Console, and explored the relationship between the GUI, the Tcl commands, and the journal file.
- Learned how to open a PlanAhead project using Tcl.
- Became familiar with the Tcl Console, and online help for Tcl commands.
- Learned about the different execution modes in PlanAhead.
- Explored some of the basics of Tcl built-in commands.
- Created batch-mode project creation and flow execution scripts.
- Explored Tcl objects, properties, and physical constraints.
- Performed a simple conversion of UCF timing constraints to SDC equivalents and explored incremental static timing analysis reporting.