PlanAhead Software Tutorial

Using Tcl and SDC Commands

UG 760 (v 12.2) July 23, 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

The BSD License for the JGoodies Looks

Copyright© 2001-2010 JGoodies Karsten Lentzsch. All rights reserved.

Redistribution and use in source and binary 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 JGoodies Karsten Lentzsch nor 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

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 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. If you have not done so, please familiarize yourself with the basics of PlanAhead by performing the quick front-to-back tutorial located here:

http://www.xilinx.com/support/documentation/dt_planahead_planahead12-2_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. If you have any questions or issues with this tutorial, contact Xilinx Technical Support.

Sample Design Data

This tutorial uses sample design data included with PlanAhead as well as a small reference design zip file that is delivered with this document. The design data delivered with PlanAhead 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 `<Install_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_2/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_2/PlanAhead_UserGuide.pdf
- *Floorplanning Methodology Guide* (UG633) - Provides floorplanning hints. http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_2/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_2/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 with the 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 12.2 desktop icon, or select **Start > All Programs > Xilinx ISE Design Suite 12.2 > PlanAhead > PlanAhead**.
- On Linux, change the directory to `<Install_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.

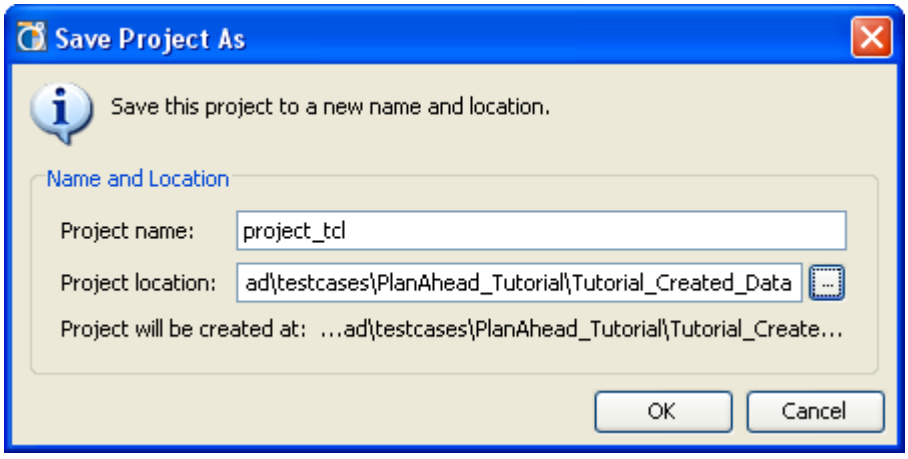

Figure 1: Saving the Project

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:

*<Install_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 the following figure.

The Netlist Design opens and is ready to explore. See Figure 2.

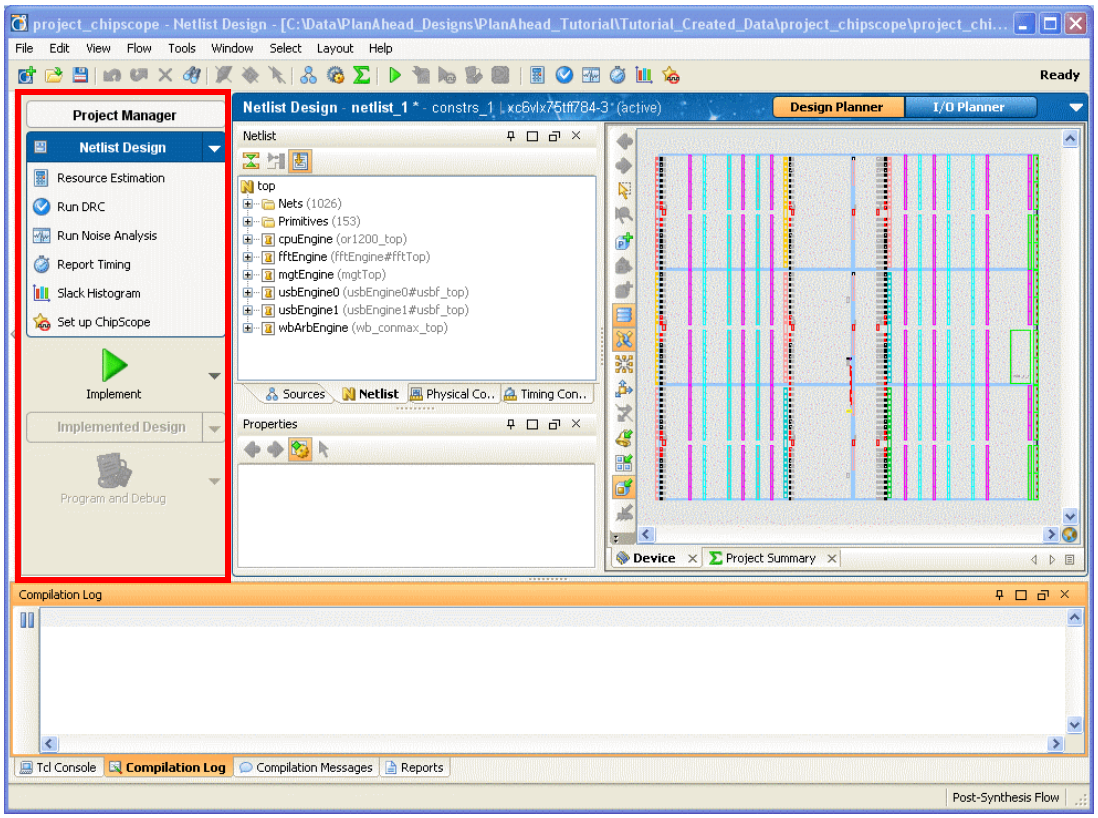

Figure 2: The Project in the Netlist Design Environment

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 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 in 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 command **exec**.

2-5. Examine PlanAhead Log and Journal Files.

Each time you invoke PlanAhead, two files 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 x 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.

For windows users, open a windows explorer and navigate to the `C:\Documents and Settings\<user_name>\Application Data\HDI` directory.

2-5-2. 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. 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. Create a text file with any text editor such as notepad, EMACS, or VI called `step3.tcl` and put the following command in it:

```
> puts "Hello World!"
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.

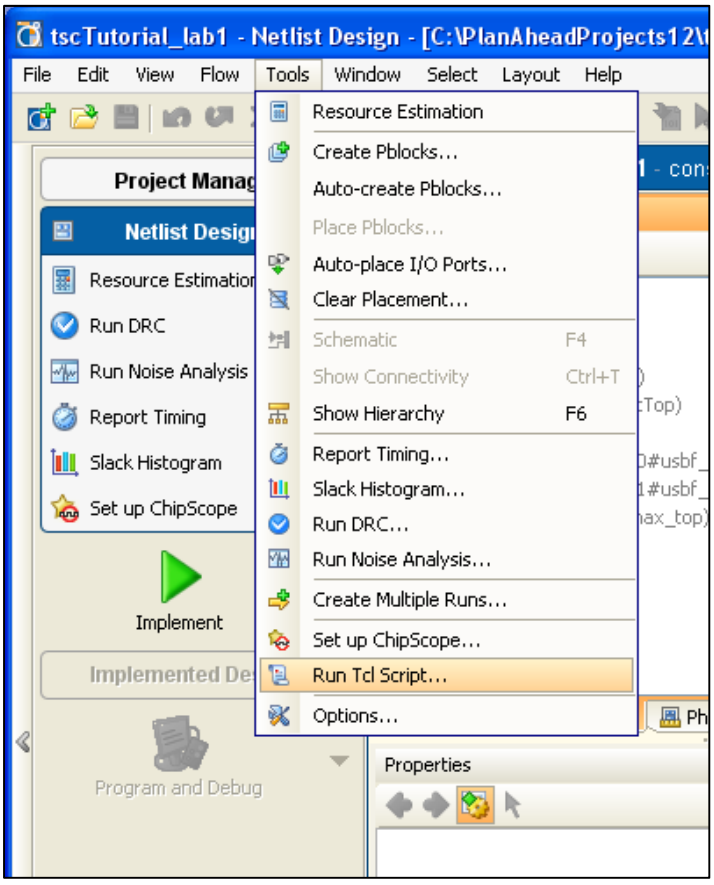

Figure 4: Launching the Run Tcl Script Dialog Box

3-1-3. In *<INSTALL_DIR>*`/src/step3.tcl`, click **Open**.

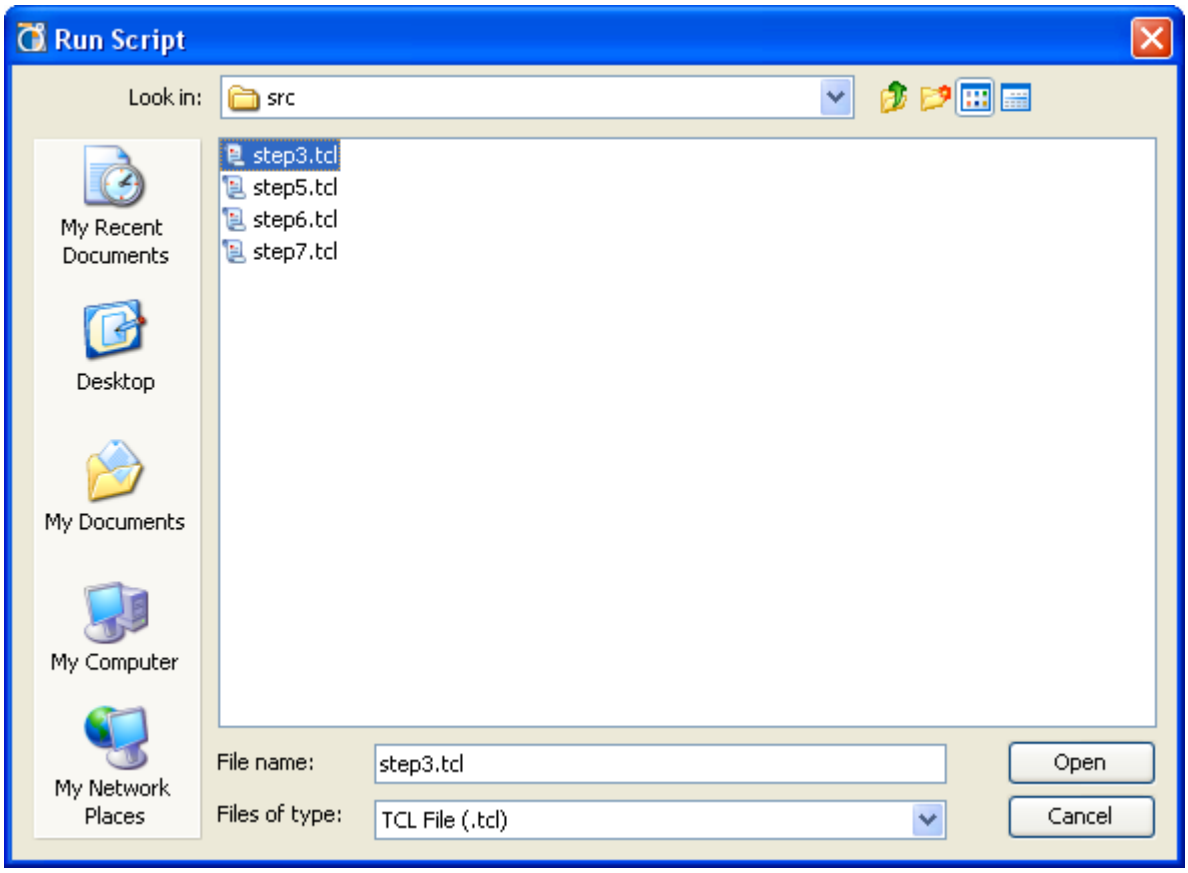

Figure 5: Run Script Dialog

The "Hello World!" prints in the Tcl console.

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

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

3-2-1. Use the `-source` command line option to the PlanAhead command from a DOS terminal (cmd) or Linux shell prompt, 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.

3-3. Use the interactive shell mode.

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

3-3-1. Launch PlanAhead in interactive shell mode, and 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. Type in the help command to view command help in the interactive shell, as follows:

```
>help
```

3-3-3. Exit PlanAhead form 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, and the tool responds 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.

4-2-1. Create a simple string variable by typing:

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

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

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

```
>set var2 1
```

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

```
>set var3 3.14
```

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

4-2-3. 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:

- 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

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

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

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:

```
>set colors "red blue green"
red blue green
```

Alternatively, you can use the explicit list command to build up a list:

```
>set colors2 [list brown black white]
brown black white
```

Tcl contains a number of built-in commands that return useful information about lists, such as the following:

```
>llength $colors
3
```

Returns the number of items in the `$colors` list.

```
>lindex $colors 0
red
```

Returns the number of items in the `$colors` list.

```
>lindex $colors2 end
white
```

Returns the number of items in the `$colors` list.

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 Step 4.5, type the following into the Tcl console:

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

The will print 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. A common way to use nesting of commands is shown by the math command **expr**:

```
>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 up.

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:

```
>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.

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

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

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

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

A small zip file, `labData.zip`, is available from the web site from which you downloaded the tutorial, which contains source code for a small design and some scripts to use as reference in this tutorial.

5-1. Create a project.

5-1-1. Extract the `labData.zip` into an empty directory named `<INSTALL_DIR>`.

5-1-2. Change directories to the location of the tutorial design data as follows:

```
cd INSTALL_DIR
```

5-1-3. Create a new file named `step5.tcl`

5-1-4. Place the following basic variable definitions in the `step5.tcl` file to use later:

```
set projDir [file dirname [info script]]
set srcDir $projDir/src
set projName oneFlopHier
set topName 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.

5-1-5. 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
}
```

5-1-6. Use the `create_project` command to create a new project, using the variables created in Step 5.1.5 for the project name, directory, and the target part, by typing the following:

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

5-1-7. 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 would value would be Netlist instead of RTL.

5-1-8. Define the RTL sources to add to the project, by typing the following:

set verilogSources [glob $srcDir/*.v]

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

5-1-9. 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 are into the project and put those files in "container" objects called a fileset. The default fileset is named `sources_1`, thereby ensuring that the original files are local to the project.

The –force option overwrites any previous sources by the same name, and

the –norecurse option tells PlanAhead not to recursively search every subdirectory and add any additional files it finds.

5-1-10. Set the library name. The work library is the default, so this command is optional:

`set_property library work [get_filesets sources_1]`

5-1-11. Set the UCF constraint files by typing the following command:

set ucfSources [glob $srcDir/*.ucf]

import_files -fileset [get_filesets constrs_1] -force -norecurse $ucfSources

5-1-12. 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]

5-1-13. 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.

Note: After this script runs, PlanAhead exits. You could open PlanAhead normally, and use the **File > Open** project to navigate to the `.ppr` project file you created and use the project a in the GUI.

5-2. Launch Synthesis.

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

5-3. Add commands to synthesize the project using the default synthesis strategy.

5-3-1. 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 next step, which would be to launch implementation:

5-3-2. Enter the command to block execution as follows:

wait_on_run synth_1

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

5-3-4. 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:

5-3-5. 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.

5-3-6. Save the `step5.tcl` file, and 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.

5-4. 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.

5-4-1. 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 runs using the default strategy.

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

5-4-3. 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]

5-4-4. Save the `step5.tcl` file, and 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.

5-5. 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.

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

open_impl_design

Save the `step5.tcl` file, and execute PlanAhead sourcing the script you just created:

planAhead –source step5.tcl

This script deletes the previous project, recreates a new one, runs through implementation flow, and opens the implementation result design for further analysis and operation. This will run the flow and open the GUI with the implemented design open and ready for other commands or analysis using the GUI.

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

In the previous step, you explored creating projects and executing simple flows 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:

```
set projDir [file dirname [info script]]
set srcDir $projDir/src
set projName oneFlopHier
set topName top
set device xc6vlx75tff484-1
# open new project
open_project $projDir/$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.

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 synthesis run `synth_1` should be "**XST Complete!**"

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 0]
>get_property site [lindex $cellList 0]
```

These commands query all cells in the design and do a property report on the first cell in the list. The final command returns the value of a location constraint, if that constraint was applied to the cell.

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 print a report of the property values for the first one in the list. Finally, the type property is queried, which should return a global clock net.

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 print 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.

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 oneFlopInst/dataIn/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. 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 oneFlopInst/dataIn]
>set pin [lindex [get_pins -of $cell -filter "direction == IN"] 0]
>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 in]
>set_property site IOB_X0Y72 [get_ports in]
>set_property is_fixed true  [get_ports in]
>set_property site OLOGIC_X0Y73 [get_cells oneFlopInst/dataIn]
>set_property bel OUTFF [get_cells oneFlopInst/dataIn]
>set_property is_fixed true [get_cells oneFlopInst/dataIn]
```

These commands set physical constraints such as `IOSTANDARDS`, `LOC`, and `BEL` constraints. Most attributes that can be applied in HDL or which would propagate to attributes in an EDIF netlist 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 engine that is separate from TRACE, the sign-off STA engine for ISE. The PlanAhead STA engine is compatible with SDC constraints and supports incremental timing analysis.

In this lab, 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 use the project created in steps 5 and 6, and will save with a different name, so that you can compare the two different projects. In the `/src` directory provided with the lab material is a script to do this conversion for you.

You will copy `step7.tcl` from the `./src` directory with the project data to the directory you were working in during Step 5.

7-1-1. From the Step 5 working directory, type:

```
copy .\src\step7.tcl .
```

Note: the command has back-slashes for windows OS. If you are working in Linux, substitute the appropriate forward slash for your file system copy to work.

7-1-2. Execute the following script in the same directory that you were working in for steps 5 and 6:

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

This command opens the design from the step 5 directory, 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.

The following are UCF commands to constrain the simple design from Step 5. You will convert these commands to SDC equivalent.

```
TIMESPEC TS_clk = PERIOD "clk" 10 ns;
NET "clk" TNM_NET = "clk";

NET "in" OFFSET = IN 3 ns VALID 7 ns BEFORE "clk" RISING;
NET "rst" OFFSET = IN 3 ns VALID 7 ns BEFORE "clk" RISING;

NET "out" OFFSET = OUT 8 ns AFTER "clk" RISING;
```

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 `create_clock` command.

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

```
create_clock -name TS_clk -period 10 -waveform {0 5} [get_ports clk]
```

This command creates a clock with a period of 10ns, with a rising edge at 5ns rooted on the top-level port named clk

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

7-2-4. In the PlanAhead GUI Tcl Console, source top.sdc to create the clock.

```
>source top.sdc
```

7-2-5. See the timing results with your clock by running the `report_timing` command:

```
>report_timing -from [get_clocks TS_clk]
```

You will see a path trace printed in the Tcl console window which equates to the worst critical path on the TS_clk clock domain.

7-3. Input Constraints.

Now, you will set up the input timing relationship that equates to `OFFSET IN` constraints.

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

```
set_input_delay -max -clock [get_clocks TS_clk] 7 [get_ports in]
set_input_delay -add_delay -min -clock [get_clocks TS_clk] 4 [get_ports in]
```

This constraint states that the `in` signal arrives 3 ns prior to the rising edge of the `TS_clk` clock domain. The second constraint, contains the `-add_delay` option that instructs the tool to preserve any prior constraints on the port/pin list, and provides the min-delay (hold analysis) equivalent to the `VALID` window on the `OFFSET IN`.

Next, you set the same input relationships for the reset signal rst.

7-3-2. At the end of top.sdc, add the following constraints:

```
set_input_delay -max -clock [get_clocks TS_clk] 7 [get_ports rst]
set_input_delay -add_delay -min -clock [get_clocks TS_clk] 4 [get_ports rst]
```

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

7-3-4. In the PlanAhead GUI Tcl Console, source `top.sdc` to recreate the constraints.

If you did nothing but source the SDC file, it would attempt to create another clock called `TS_clk` and it would error out because the clock already exists.

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

7-3-5. Enter the following in the Tcl command text box:

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

7-3-6. And to immediately see the timing results with your clock, run the `report_timing` command:

```
>report_timing -from [all_inputs]
```

This command will report the worst critical path from any of the input pins we just constrained.

7-4. Output Constraints.

Finally, you must constrain the output signal requirements according to the specification in the `OFFSET OUT` constraint. The equivalent in SDC is the `set_output_delay` command.

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

```
>set_output_delay -clock [get_clocks TS_clk] 2 [get_ports out]
```

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 your clock immediately, run the report_timing command:

```
>report_timing -to [all_outputs]
```

A path trace displays for the worst critical path to any of the output pins.

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, consider as scenario where you wanted specify a multi-cycle path from the input port in to the one `FlopInst/dataIn` flip-flop. This is the equivalent of a `FROM/TO` constraint in UCF with a `TIMESPEC` multiplier.

To relax timing to this flip-flop 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:

```
> set_multicycle_path -to [get_cells oneFlopInst/dataIn] 2
```

```
> report_timing -to [get_cells oneFlopInst/dataIn]
```

This performs an incremental STA update, and updates only the portions of the timing graph affected by the new constraint. Notice the required time for this path shifted from 7ns to 17ns, giving this path 2 full clock periods to meet timing.

7-6. Use False Paths.

Similarly for false paths, we can incrementally add the SDC equivalent of TIG constraints in 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.

7-6-1. Enter the following constraint directly into the PlanAhead GUI Tcl console:

```
> set_false_path -to [all_outputs]
```

```
> report_timing -to [all_outputs]
```

Notice there is no slack calculation on the output pins. The data path delay is reported, but the critical path in this design is no longer the output path.

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 by the user.

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.