

EE4415 Lab Part 1 (Total 5%)

Instructions

You are required to submit a formal report for this lab at the end of semester. In your report, you should include all the answers to the questions listed under each unit and all the required script files except those provided in the manual. The deadline for submission of lab report will be announced later in the semester.

If you need any additional information about Synopsys Design Compiler, please use Synopsys online help. (Linux command: `sdh`)

Synthesizing a 16-bit RISC CPU

Introduction

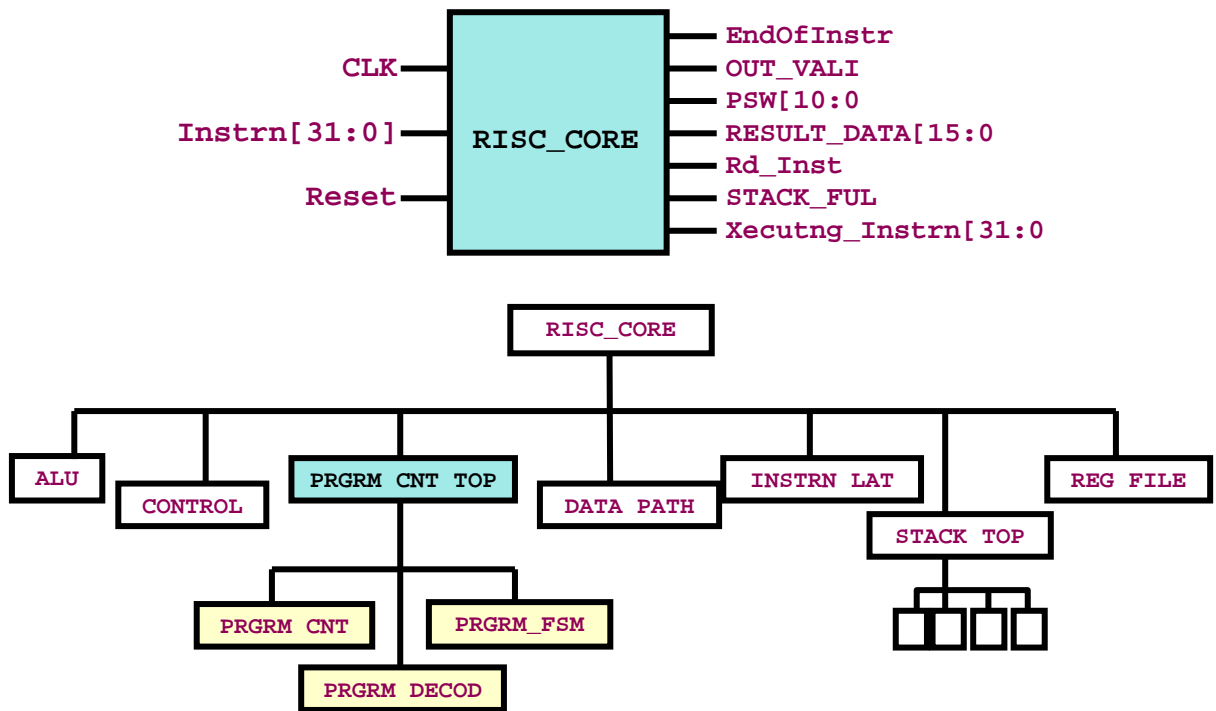
Objectives

You will go through the synthesis process of the core of a 16-bit RISC (Reduced Instruction Set Computer) CPU. Upon completion, you will have a mapped design that meets the stated design specifications as well as a complete script file that automates the synthesis process and documents your work.

Design description

Instruction size is 32-bits and data-size is 16-bits. The microprocessor supports 36 different data, ALU and control-transfer instructions. It follows a Harvard architecture (code and data cache are separate, http://en.wikipedia.org/wiki/Harvard_architecture). The data memory size is 4×16 -bit. The instruction memory has been pushed outside the 'to-be-synthesized' core block. The design size is about 10K gates.

Block diagram for RISC_CORE



Setting up

To login to the workstation, you need to get a user ID and password from the Graduate Assistant. **Once you are logged in to the system, please remember to change your password.**

Please do the following to setup your lab folder.

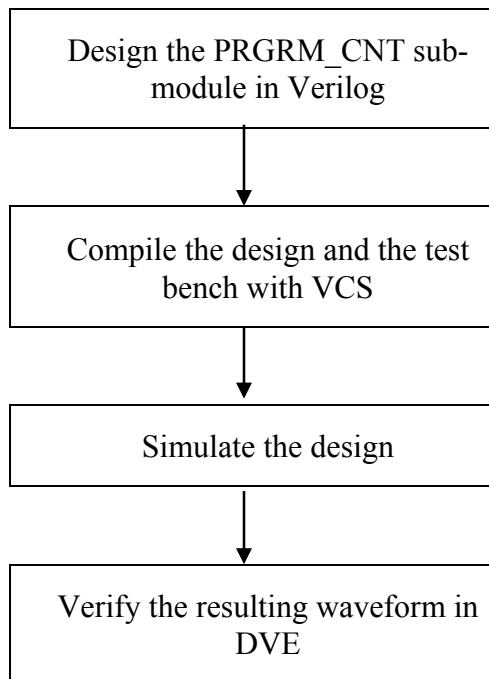
1. After logging in, right click on an empty space in the desktop and click “New Terminal” to open a window for you to enter the commands.
2. In the newly opened window, execute the command “`cp -r /app11/lab-session/ee4415_part1/ ~/ee4415_part1`” to copy the lab materials to your home directory.

Unit 1: Verilog, VCS and DVE

Objectives

In this unit, you'll learn how to design the *PRGRM_CNT* sub-module of the *RISC_CORE* in Verilog, simulate the code using the VCS simulator and view the waveform with DVE.

Lab Flow



More information on VCS and DVE can be found in:

/app11/synopsys/vcs_vJ-2014.12/doc/UserGuide/pdf/vcs.pdf

/app11/synopsys/vcs_vJ-2014.12/doc/UserGuide/pdf/dve_ug.pdf

Lab Instructions

Task 1. Design PRGRM_CNT in Verilog

1. Navigate into the lab folder by typing the following command.

```
cd ~/ee4415_part1
```

2. The folder contains two Verilog files, *prgrm_cnt.v* and *prgrm_cnt_tb.v*. *prgrm_cnt.v* implements the program counter functionality for the *RISC_CORE* while *prgrm_cnt_tb.v* contains the test bench code to verify the functionality of *prgrm_cnt.v*.
3. Open *prgrm_cnt.v* with your favourite editor (eg. Vim, gedit etc). For example, the following command opens the file with gedit.

```
gedit prgrm_cnt.v
```

4. Complete the file with the following specifications for *PRGRM_CNT*, which are listed in decreasing priority. For example, if both *Reset* and *Incrmnt_PC* are 1, *PC* is set to 0 instead of being incremented by 1.
 - a. *PC* is set to 0 when *Reset* is 1.
 - b. *PC* is incremented by 1 when *Incrmnt_PC* is 1.
 - c. *PC* is set to *Return_Addr* when *Ld_Rtn_Addr* is 1.
 - d. *PC* is set to *Imm_Addr* when *Ld_Brnch_Addr* is 1.

Task 2. Compile and simulate *prgrm_cnt.v*

1. Run the following command to compile *prgrm_cnt.v* and *prgrm_cnt_tb.v* to generate the *simv* file. *+lint=all* enables all warning messages. *-debug_all* enables all debug features (eg. setting breakpoints and tracing) in DVE. *-timescale=1ns/10ps* sets the time unit and time precision to 1 ns and 10 ps respectively.

```
vcs -full64 +lint=all -debug_all -timescale=1ns/10ps
prgrm_cnt.v prgrm_cnt_tb.v
```

2. Check that you have the *simv* executable in the folder after executing *vcs* by running the following command.

```
ls
```

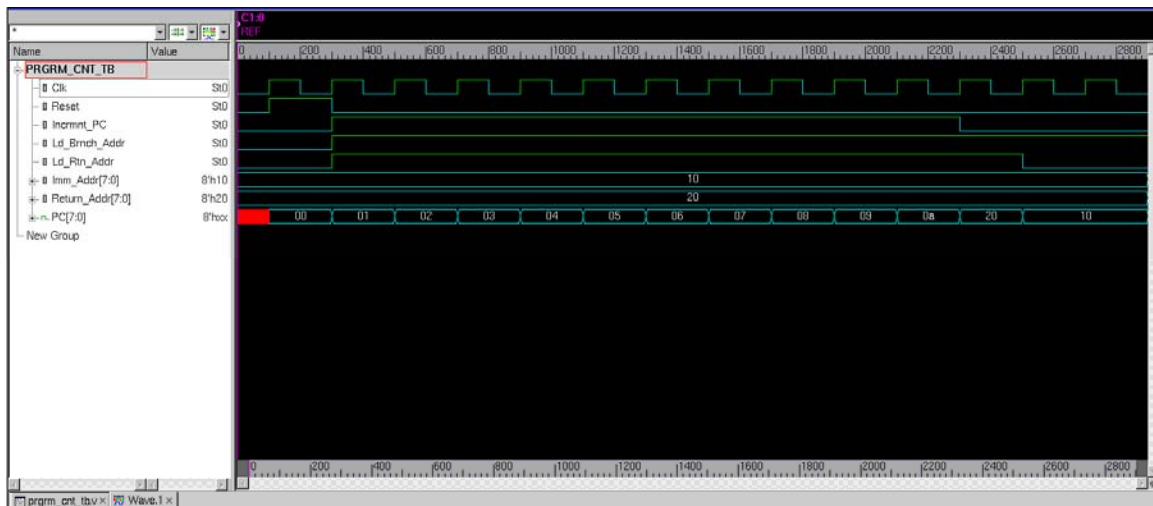
3. Run the *simv* executable by typing “./simv”.

Task 3. Verify the waveform in DVE

1. The test bench in *prgrm_cnt_tb.v* dumps the simulation result in the *results.vpd* file, which can be viewed using DVE by running the following command.

```
dve -full64 -vpd results.vpd &
```

2. View the waveform by clicking selecting all the signals in the signal list and click Signal → Add to Waves → New Wave View. The whole waveform can be set to fit into the window by clicking View → Zoom → Zoom Full. Check that the waveform is identical to the waveform in the figure below. **Please include your completed *prgrm_cnt.v* in your report.**



Unit 2: Introduction to Synopsys Design Flow

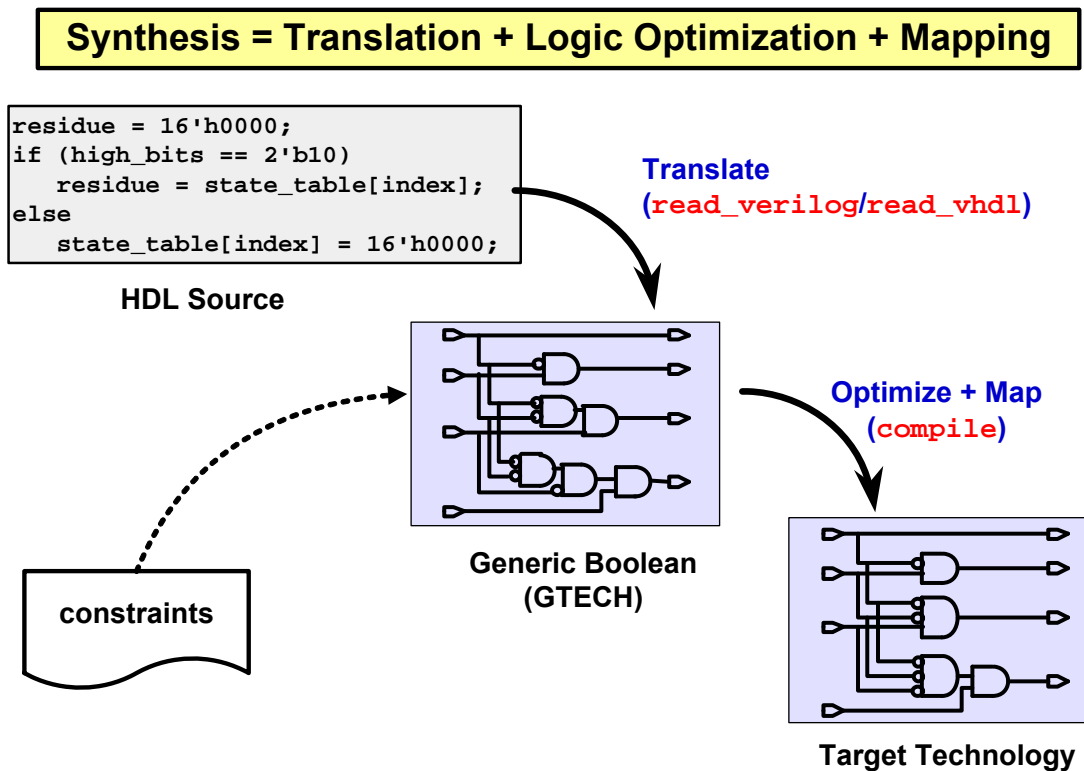
Objectives

This exercise describes the basic steps involved in the synthesis process and how to navigate through a design's hierarchy using Design Vision, the graphical interface to Design Compiler.

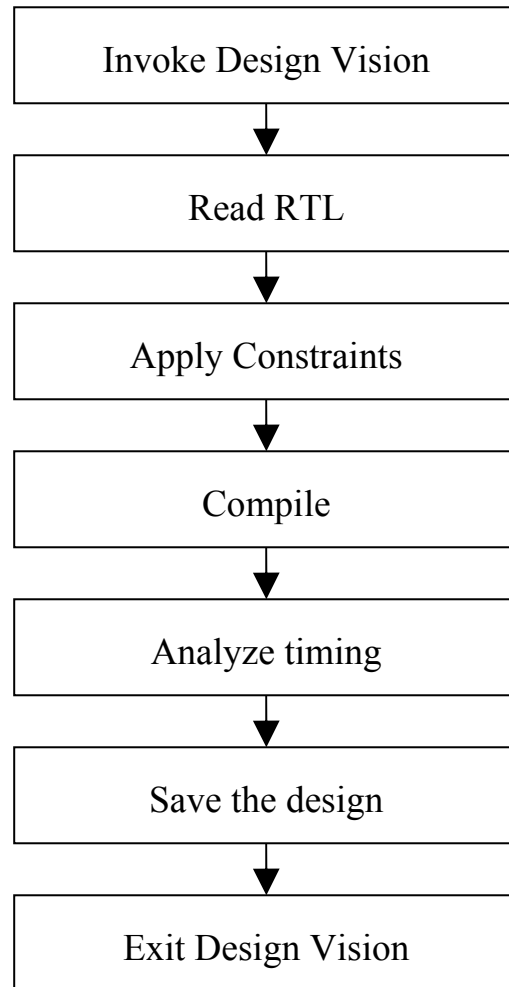
Preliminary Concepts

In the lecture material, the process of synthesis is described as **translation + logic optimization + mapping**.

In terms of the Synopsys tools, **translation** is performed by the “read_vhdl”/“read_verilog” commands. **Logic optimization** and **mapping** are performed by the “compile” command. This process is illustrated in the figure below.



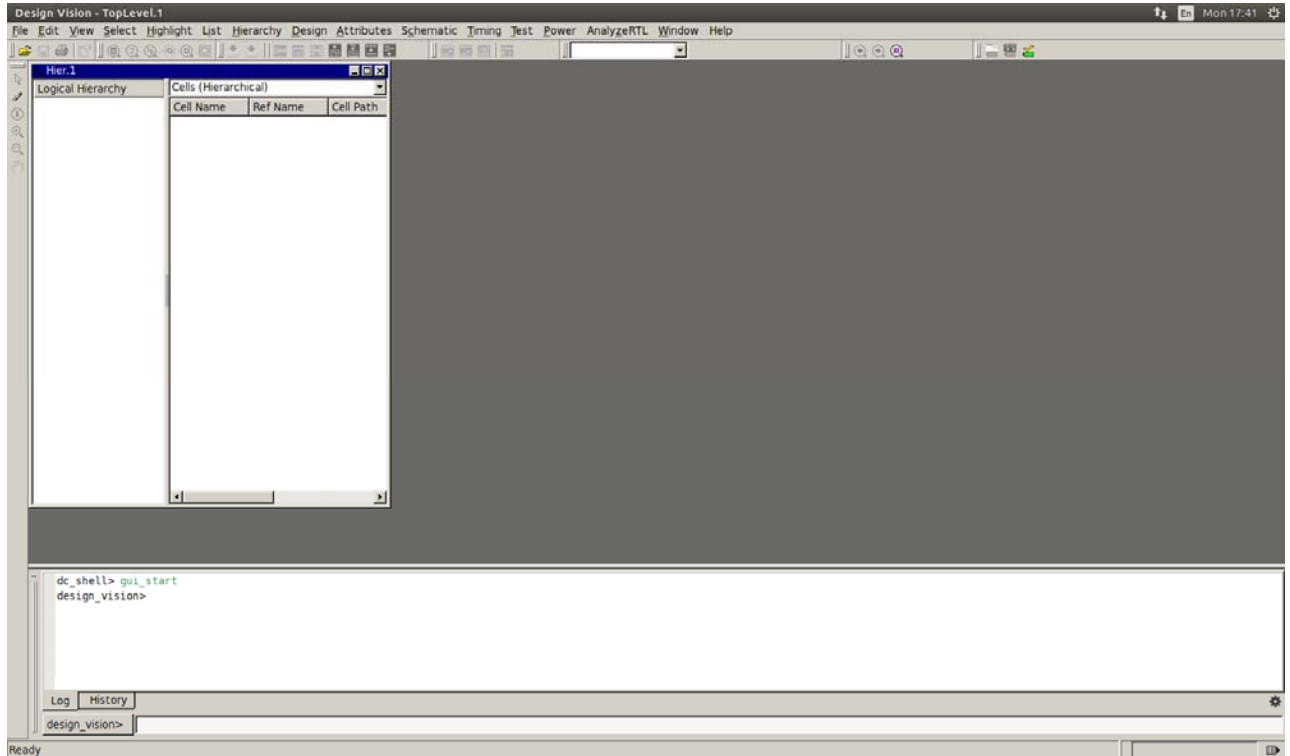
Lab Flow



Lab Instructions

Task 1. Invoke Design Vision

1. Change your current working directory to the lab directory using the “`cd ~/ee4415_part1`” command. Always remember to start Design Vision/Compiler only from this directory due to the `.synopsys_dc.setup` file (more about this file later).
2. Start Design Vision from the UNIX prompt using the “`design_vision-xg`” command.

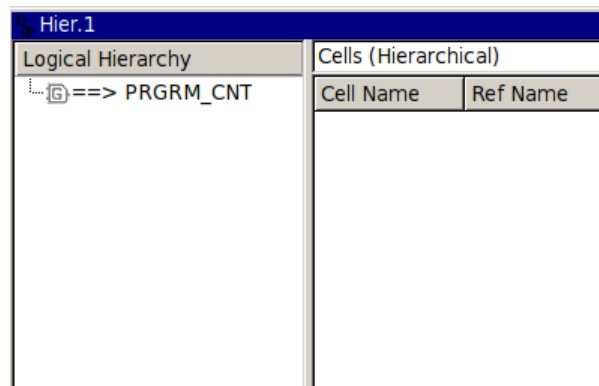



Task 2. Read RTL

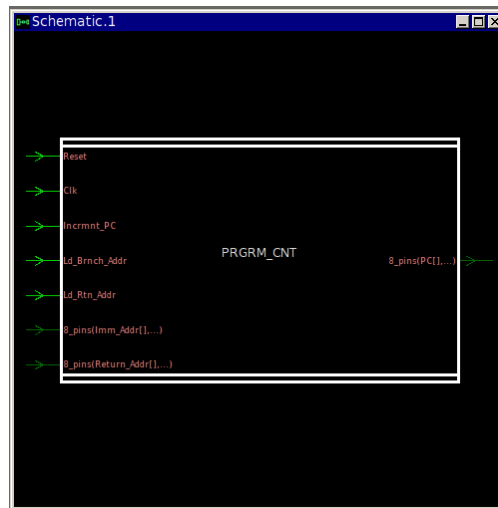
1. Before reading the RTL files, ensure that the environment variables are set properly by going to File → Setup. Check that the Link library and Target library values are the same as the values in the `.synopsys_dc.setup`. The `.synopsys_dc.setup` file is executed everytime Design Vision/Compiler is launched and it is generally used to setup various environment variables and aliases.

2. Click on the **Read** button  at the top left of the GUI.

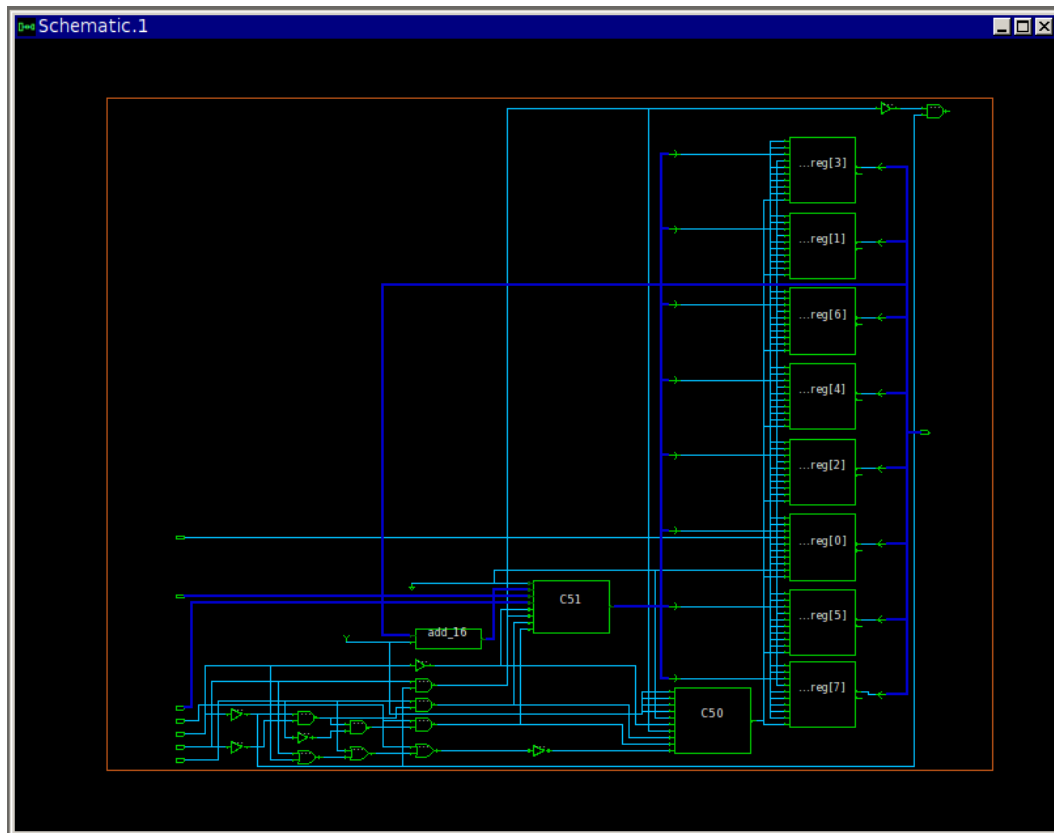
In the dialog box that appears, select the file completed `prgrm_cnt.v` file from Unit 1 then click on **Open**. You will see an icon labeled `PRGRM_CNT` in the **Hier.1** window. The design `PRGRM_CNT` is now loaded in Design Compiler's memory in terms of GTECH components.



3. Click on the label `PRGRM_CNT` in the **Hier.1** window and open the **Schematic** view by clicking the  icon in the toolbar. The **Schematic** view shows a high level picture of the design. The input and output ports shown in the window corresponds exactly to the input and output ports specified in the `prgrm_cnt.v` file.



4. Double click on the `PRGRM_CNT` block in the **Schematic** window to go down one level of the design hierarchy. If you completed your `prgrm_cnt.v` file correctly, you should see something that is similar to the figure below. Notice that the design is implemented using the cells from the GTECH library because the design has not been mapped (compiled) using the target technology library.



5. To go up one level of the hierarchy (back to the high level block), right click on an empty space in the **Schematic** window and click Back.

Task 3. Apply constraints

1. Before the design can be compiled, constraints have to be applied in order to guide the Design Compiler towards the required optimization goal (timing, area, etc). Based on what you have learnt in the lecture, complete the file *constraint.tcl* to set the following constraints. After completing the *constraint.tcl* file, apply the constraints using the “source constraint.tcl” command.

Clock period	1.5 ns
Clock uncertainty	0.1 ns
Max input delay (except Clk port)	1 ns
Max output delay	1 ns
Driving cell and pin at the input	DFFX1_RVT, pin Q
Output load	10 times the capacitance seen at the input port A of the INVX0_RVT cell
Max area	200

Note that Design Compiler assumes that the timing values specified in the commands are normalized to the time unit in the technology library. Therefore, in order to set the timing constraints correctly, use the “report_lib” command (or “view report_lib” because “report_lib” generates a long report) to determine the time unit of the technology library. For example, if the time unit is 0.1 ns, passing a value of 10 as the parameter to “set_input_delay” is equivalent to setting the input delay to $0.1 * 10 = 1$ ns.


2. To check whether constraints are set properly, use the following commands.
 - a. report_clock -skew -attributes
 - b. report_port -verbose

Task 4. Compile

3. After applying the constraints, the design can be compiled using the “compile” command.
4. After compilation, view the schematic of *PRGRM_CNT* again and you should see that the design is now realized using the target technology library.

Task 5. Analyze Timing

There are many commands and tools in Design Compiler to aid you in analyzing the timing and debugging timing problems. For simplicity, you will see one way.

1. Locate the histogram buttons  at the top of the window.
Hover your mouse over the left-most button – a “tool hint” should be displayed indicating “Create Path Slack Histogram”.
2. Click on the **Create Path Slack Histogram** button.
3. In the dialog box that appears, simply select **OK**.
You will see a histogram window displaying several “bins”. Each bin represents a number of timing paths. You can click on the bins, and Design Compiler will list details of the paths contained in this bin (Startpoint, Endpoint, Timing Slack).

In Unit 4, you will learn how to interpret this type of report.

Task 6. Generate a Timing and Area Report

1. Check whether all the constraints specified in *constraint.tcl* are satisfied using the “report_constraint -all_violators” command.
2. Record the following information using the “report_timing” and “report_area” commands:
Max delay and slack: _____
Area and slack: _____
3. Open the **Schematic** of *PRGRM_CNT*.
4. Run **Timing** → **Timing Analysis Driver** and select the critical path (the path with the largest violation). Then click **Highlight** → **Selected**. Notice that the critical path is highlighted. If you can’t see it, go down the design hierarchy by double-clicking the *PRGRM_CNT* block in the Schematic.
5. To undo the highlighting, click **Highlight** → **Clear All**.

Task 7. Save the Mapped Design

1. To save the design, click **File** → **Save As**.
2. Verify that the **Save All Designs in Hierarchy** button is selected. This will save the entire design hierarchy into a single (.ddc) file.
3. Enter *prgrm_cnt.ddc* as the filename.
4. Click **Save**. You have just saved the gate-level netlist (the entire hierarchy) in the DDC format in the current directory. You can verify the created file using the command “ls -l”.

Task 8. Remove Designs from Design Compiler Memory

1. To remove all designs and libraries from Design Vision’s memory, click **File** → **Remove All Designs** or use the “remove_design -all” command.

Task 9. Exit Design Vision

1. To exit design vision, click **File** → **Exit** → **OK**, or type “exit” at the command prompt, and choose **OK** when prompted.

Note:

1. The “history” command will list a history of all commands you have executed since you started Design Vision.
2. If you unintentionally exit out of Design Vision, you can recreate everything you did up to that point by executing the command.log file that has been created in your lab directory by doing the following:

```
unix% cp command.log lab2.log
unix% design_vision-xg -f lab2.log
```

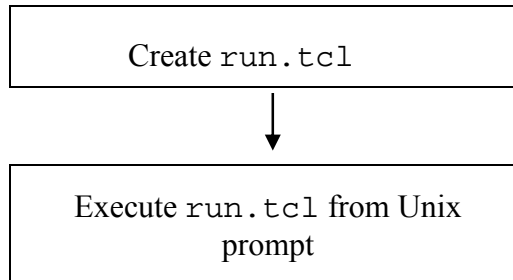
Unit 3 Introduction to DC-Tcl

Objectives

After completing this lab, you should be able to:

- Write a simple DC-Tcl script file to compile a design

Lab Flow



Lab Instructions

1. From the previous Unit, you have learnt that the entire synthesis flow can be done by either clicking the buttons in Design Vision or running individual commands in the command prompt. It is also possible to automate all or part of the synthesis flow by writing a Tcl script and running the script using Design Compiler.
2. In this unit, you are required to complete the script file *run.tcl* to perform the following tasks. As an example, the command to complete the first task has already been provided in the *run.tcl* file.

- Read the file *prgrm_cnt.v*. (Provided as an example in *run.tcl*)
- Set the current design to PRGRM_CNT
- Perform a *link*
- Constrain the design by applying the *constraint.tcl* file
- Compile the design
- Generate and save the results from a constraint report to PRGRM_CNT.rpt using the following command

```
redirect -tee PRGRM_CNT.rpt \  
        {report_constraint -all_violators}
```

- Save the mapped design as PRGRM_CNT.ddc
 - Quit Design Compiler
3. After completing the *run.tcl* file, execute the file using the following command

```
dc_shell-xg-t -f run.tcl | tee -i run.log
```

The tee program is a standard UNIX program that redirects the output of other programs to **both** a file and to a terminal window.

4. Check *run.log* visually or using the command below to verify that there are no errors in the script file.

```
grep Error run.log
```

5. Verify the files PRGRM_CNT.ddc and PRGRM_CNT.rpt are successfully created.

Unit 4 Timing Reports

Objectives

After completing this lab, you should be able to:

- Interpret various timing reports generated by “report_timing”

Lab Instructions

Task 1. Load Mapped PRGRM_CNT Design

1. Execute “dc_shell-xg-t” and read the PRGRM_CNT.ddc file you saved from Unit 3 using the command “read_ddc PRGRM_CNT.ddc”.

Task 2. Generate and Interpret Four Timing Reports

2. Answer the following questions regarding PRGRM_CNT.

Question 1. Are there any unconstrained timing paths in PRGRM_CNT?
(Use the command `check_timing` to verify unconstrained timing paths. Look up a man page for this command.)

Question 2. How many path groups are in PRGRM_CNT? (Use the command `report_path_group`)

3. Use the view utility to generate a default timing report and then answer the following questions. The first command below simply shows the expansion of the alias `vrt`.

If the view command does not work in your lab environment, either use page mode to view long reports or redirect the output of report_timing to a file and use a text editor to read the report.

```
dc_shell-xg-t> alias vrt
dc_shell-xg-t> vrt
```

Question 3. Is this a setup time or hold time timing report?

Question 4. What is the start point (input port or clk pin of internal register?)

Question 5. What is the end point?

Question 6. Under what operating conditions was this timing report generated?

Question 7. Did this timing path meet or violate its constraint?

Question 8. What is the clock period for Clk?

Question 9. What is "input external delay", and where did this number come from?

Question 10. Does the design's partitioning break a combinational path, if so explain?

Question 11. What is the setup time requirement of the capture register?

Question 12. What does the clock uncertainty number represent?

4. Use the view utility to generate a timing report with input pins and with 6 significant digits.

```
dc_shell-xg-t> vrt -input_pins -significant 6
```

Question 13. What is different in this timing report from the default timing report?

5. Use the view utility to generate a timing report with net names and fanout.

```
dc_shell-xg-t> vrt -nets
```

Question 14. What delay is associated with each net and why is the delay zero?

Question 15. What does the "Fanout" column represent?

The incremental timing of the line with the net name is always zero.

6. Use the view utility to generate a timing report for hold.

```
dc_shell-xg-t> vrt -delay min
```

Question 16. Is this a setup or hold time report?

Question 17. What is the start point?

Question 18. What is the end point?

Question 19. Did this timing path meet or violate its constraints?

Question 20. Under what operating conditions was this report generated?

Question 21. Is this an appropriate operating condition for hold time calculations?

Question 22. What is the hold time requirement of the end point?

Question 23. What is the delay through the launching register?

Question 24. Is this delay enough to satisfy the hold time requirement?