EE4415 Integrated Digital Design

Instructions

All lab sessions are conducted at e-ITU Linux Workstation Clusters, WS3(E1-04-12) WS4 (E1-04-11, from Feb 14 to Feb 18), and WS2(E1-04-13, from Feb 28 to Apr 14) on every Tue., Wed., Thur., and Fri. from 2:00-5:00pm. Lab attendance is compulsory and you need to sign in for each lab session. Please sign up for lab session by sending an email to EE4415 GA: Mr. Hong Yibin, email: ***g0900188@nus.edu.sg***. Please get your userid and password from Mr.Hong.

You are required to submit a FORMAL report for this project 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 is on 16 April 2012. Please submit your report to Signal Processing and VLSI Lab at E4-08-33.

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

Synthesizing a 16-bit RISC CPU

Introduction

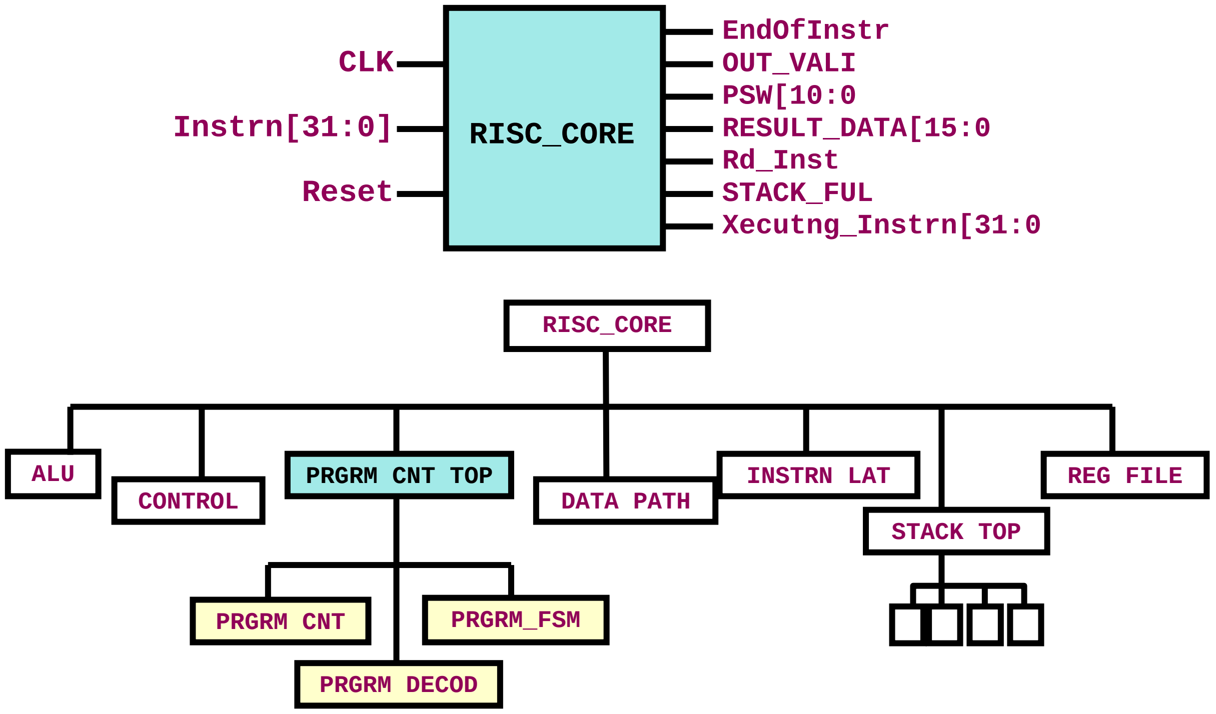
Objectives

You will go through the synthesis process of the core of a 16-bit RISC (Reduced Instruction Set Computer) CPU. The functionally correct and simulated code of the design is available. 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


CLK
Instrn[31:0]
Reset
RISC_CORE
EndOfInstr
OUT_VALI
PSW[10:0
RESULT_DATA[15:0
Rd_Inst
STACK_FUL
Xecutng_Instrn[31:0
RISC_CORE
ALU
CONTROL
PRGRM CNT TOP
DATA PATH
INSTRN LAT
STACK TOP
REG FILE
PRGRM CNT
PRGRM_FSM
PRGRM DECOD

Directory structure

There are total 10 units in this lab manual. You will work in three different directories for the lab, i.e. **Lab_1/** for Unit 1**, rise_design** for Units 2 to 10**,** and **Lab11** for Unit 11. **<u>You should always invoke design compiler from these working directories only.</u>**

The VHDL source code is in the RISC_source directory. The design required by this lab has been analyzed and elaborated and written out in "unmapped" directory using Synopsys internal (.ddc) format. You can read them directly to avoid elaborating each time.

The file structure for the lab
:

```
Lab-1/         risc_design/                            Lab11
                    |                                    |
           ---------------------------       -------------------------------------
           |          |          |               |         |         |          |
       mapped/    scripts/   unmapped/     analyzed/    pass0/   scripts/    tcl_procs/
```

To login to Linux system, you need to get a UserID and Password from Graduate Assistant. ***Once you login to the system, please remember to change your password.***

Listed below are what you need to do:

1. After login, point your mouse at the center of screen then click the right button of mouse to open a menu and select the "New Terminal" to open a window for you to enter the commands.
2. In the newly opened window, copy lab material to your home directory by issuing the commands in bold:
   UNIX% **cd**
   UNIX% **cp -r /app1/lab-session/ee4415 .**
   The first command "cd" is to change directory to your home. The second command "cp" copy files to your home directory.

Note that the UNIX system is case-sensitive and the "." in the "cp" command above is necessary as it represents the current directory you are working with. For more information about UNIX commands, please refer to the document Basic UNIX Commands given to you.

Unit 1: Introduction to Synopsys Chip Synthesis and Design Flow

Objectives

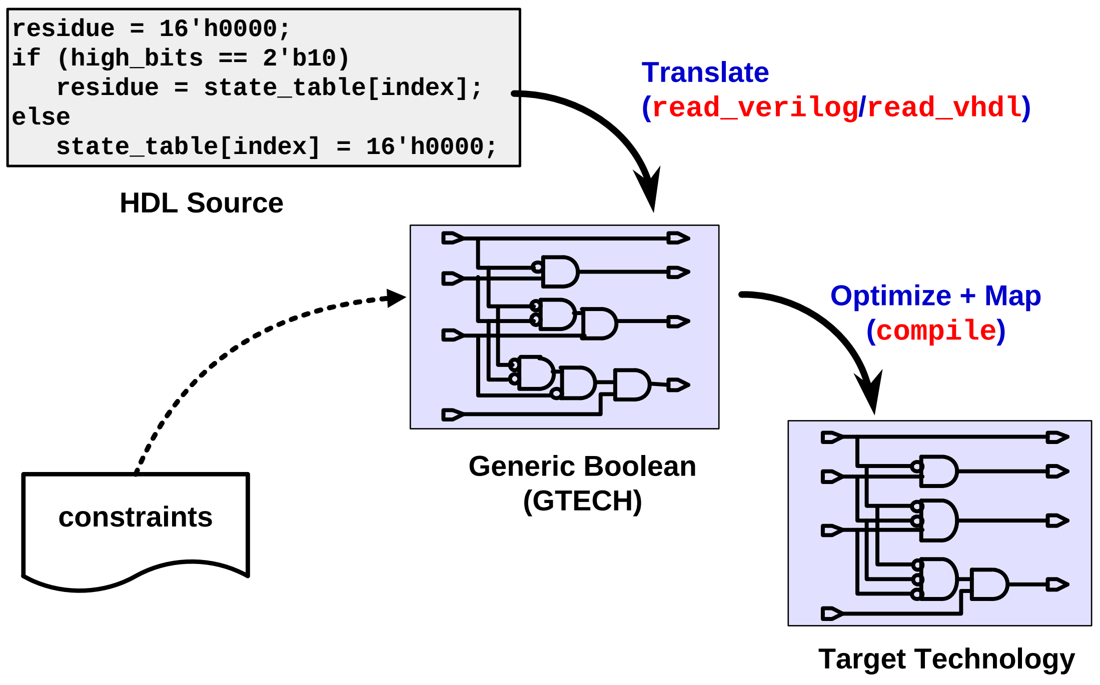
This exercise describes the 6 basic steps involved in the synthesis process and how to navigate through a design's hierarchy using Design Vision, the graphical interface to Design Compiler.

Preliminary Concepts

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

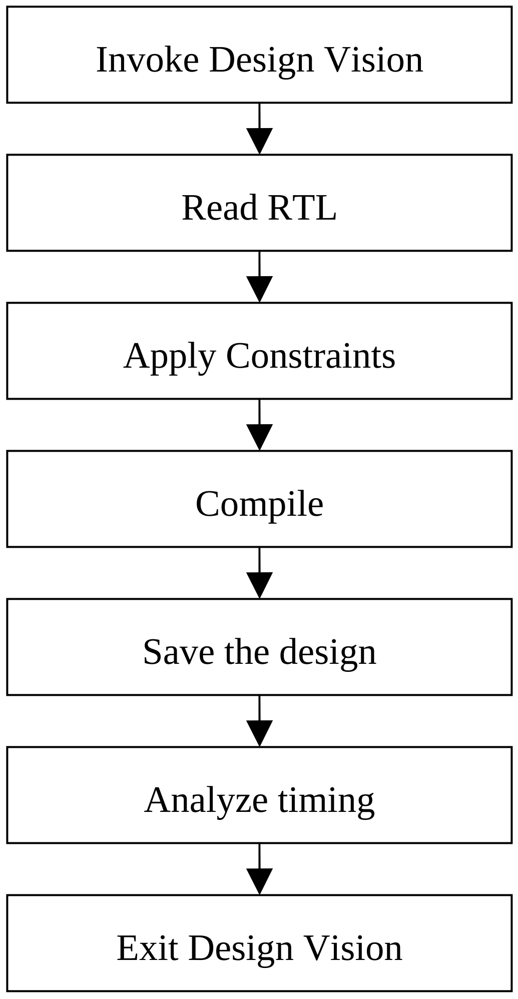
In terms of the Synopsys tools, *translation* is performed by the **`read_vhdl/read_verilog`** commands. *Logic optimization* and *mapping* are performed by the **`compile`** command. This process is illustrated in the figure below.

Synthesis = Translation + Logic Optimization + Mapping



```
residue = 16'h0000;
if (high_bits == 2'b10)
   residue = state_table[index];
else
   state_table[index] = 16'h0000;
```

Lab Flow



Definitions

`read_vhdl / read_verilog:` Reads your VHDL/Verilog files, performs syntax and synthesis-policy checks, then "builds" the design using generic (GTECH) components.

`Constrain:` Is not a Design Compiler command, but a series of steps you perform to tell Design Compiler what your timing and area requirements are.

`Compile:` Optimizes a design and maps it to real gates from your target technology library, producing a circuit that meets your constraints. The unmapped design in Design Compiler memory is overwritten by the new, mapped design.

Lab Instructions

Task 1. Invoke Design Vision

1. Log on to your workstation.

2. Change directories to the project directory **`Lab_1`**:

```
UNIX% cd Lab_1
UNIX% ls -a
```

This is your top-level directory for the `Lab_1` project; notice the file called `.synopsys_dc.setup` and the other files.

Open the file ".synopsys_dc.setup" (in *Lab_1* directory) and make sure that you have the following settings in the file:

lappend search_path ../libs

set target_library core_slow.db
set link_library "* $target_library"
set symbol_library core.sdb

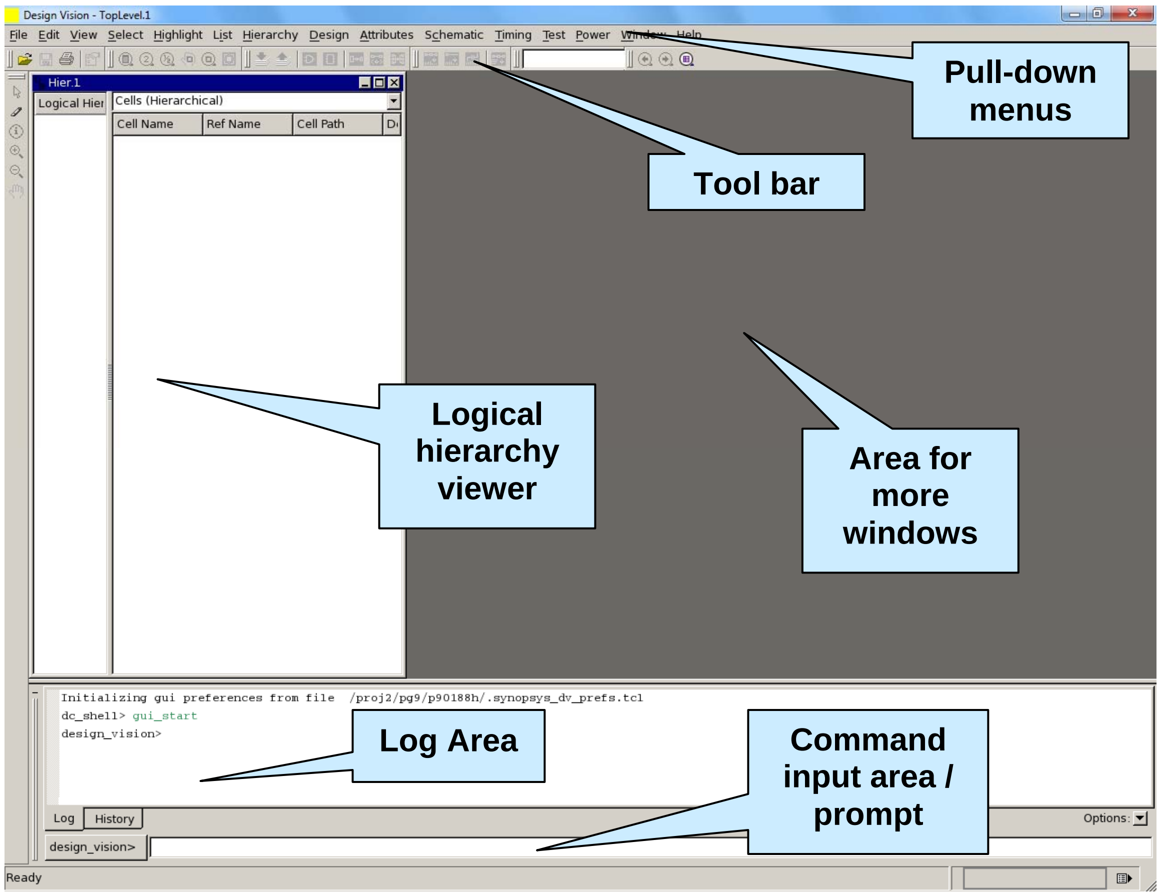
Note: There is an extra line "set synthetic_library "dw_foundation.sldb"". Please delete it. The link_library is "* $target_library dw_foundation.sldb". It should be changed to "* $target_library" as well.

Close the file.

3. Invoke Design Vision from the UNIX prompt:

```
UNIX% design_vision-xg
```

You should see the Design Vision GUI start up.



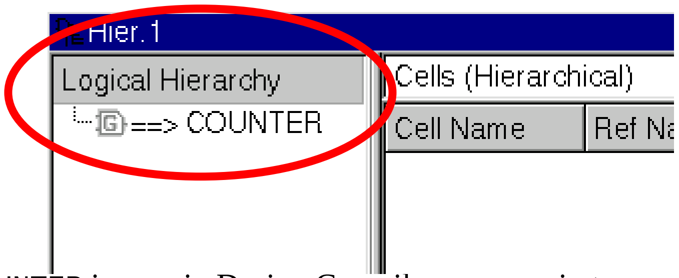
```
Initializing gui preferences from file  /proj2/pg9/p90188h/.synopsys_dv_prefs.tcl
dc_shell> gui_start
design_vision>
```

Task 2. Read RTL

Design Compiler can read VHDL and Verilog, as well as SystemVerilog RTL files.

1. Click on the Read button at the top left of the GUI.
2. In the dialog box that appears, select either one of the RTL files `counter.vhd` or `counter.v`, then click on **Open**. (alternatively you could double click on the file name)

   You will see an icon labeled `COUNTER` in the **Hier.1 window**.

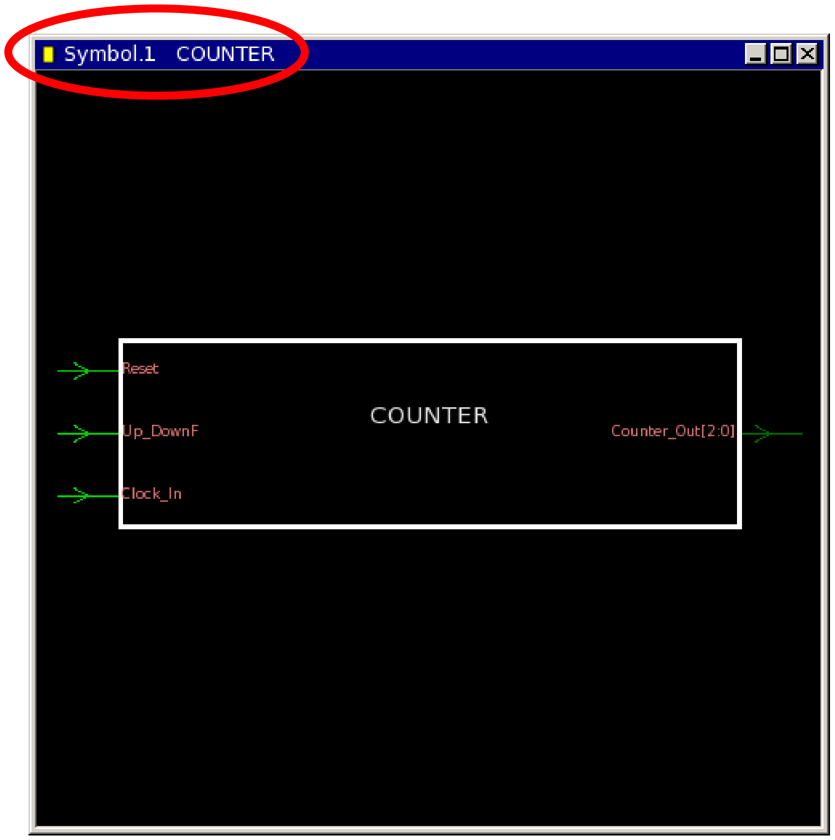


The design `COUNTER` is now in Design Compiler memory in terms of GTECH components.

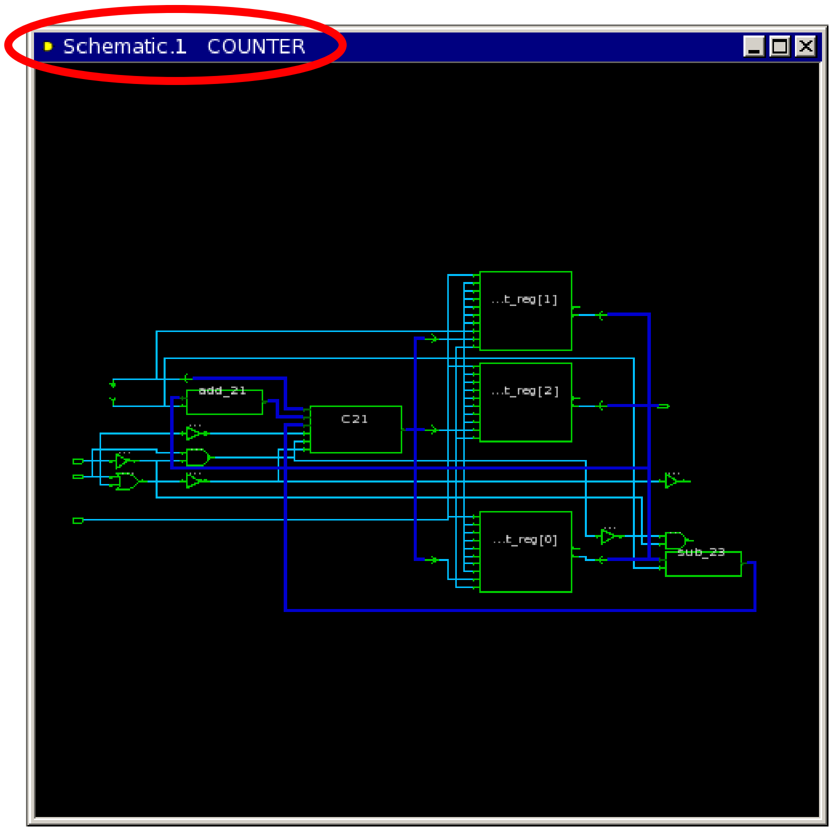
3. Click on the icon labeled `COUNTER` in the **Hier.1 window**, two black icons will appear on the tool bar.

4. Push into the "**Create Symbol View**" by clicking the icon.

You will see a block with input and output ports attached to it. This is referred to as the symbol view of the design, as indicated in the upper left corner of the Design Vision Window. The symbol view shows the block diagram of the design.



5. Push into the "**Create Design Schematic**" by clicking the icon.



See the structure of your design represented in terms of components from your Synopsys GTECH library.

Is this structure similar to what you expected from the (V)HDL code?

Task 3. Apply Constraints

6. Select the menu sequence **File → Execute Script…**

7. In the dialog box that appears, select the option "Echo commands"

8. Double-click on the constraint file "counter.con", or select the file then click on **Open**.

You should see a series of commands echoed in the Log window. Alternatively – you can look at the original Unix window from which you invoked Design Vision. The commands define the clock with a period of 2ns (500 MHz); other constraints (operating conditions, input delays, etc) have also been applied.

You will soon discover how to view and change these constraints.

The design is now constrained and ready for mapping and optimization.

Task 4. Compile

9. To **compile** the design, type the following command at the command prompt on the bottom of the Design Vision window:

```
design_vision > compile -scan
```

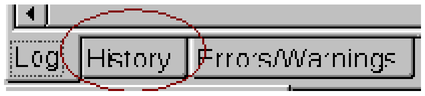
After a few moments, Design Compiler will have mapped and optimized the design.

10. Push into the "**Create Design Schematic**" by clicking the icon again.

You should now see a schematic of your design represented in terms of real components (cells) from the target technology library.

Task 5. Save the Design and Command History

After compile, or after any major steps, it is advisable to save the design. The native DCXG format is DDC – Direct DC.

1. Bring up the Save dialog using the menu sequence **File → Save As…**
2. In the "File Name:" field, specify the name you want the design saved under. Specify the name **counter.ddc**
3. Select the **Save** button. Design Compiler will know what format to save the design under based on the extension you used, ".ddc".
4. Next, select the history tab. 

5. Save the command history by selecting the button "Save Contents As" and specifying the name **run_counter.tcl.**

Task 6. 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 on the paths contained in this bin (Startpoint, Endpoint, Timing Slack).

In later units, you will learn in detail how to interpret this type of report, and many other reports.

Task 7. Exit Design Vision

Choose your way of exiting Design Vision:

1. Use the menu sequence **File → Exit → OK**, or Type **exit** at the command prompt, and choose **OK** when prompted.

Questions:

How would you handle the following last-minute changes to the design's specification?

Question 1. Counter width increases from three bits to six?

Question 2. Clock frequency increases to 1 GHz?

Question 3. You have to switch to a different silicon vendor?

End of Unit 1

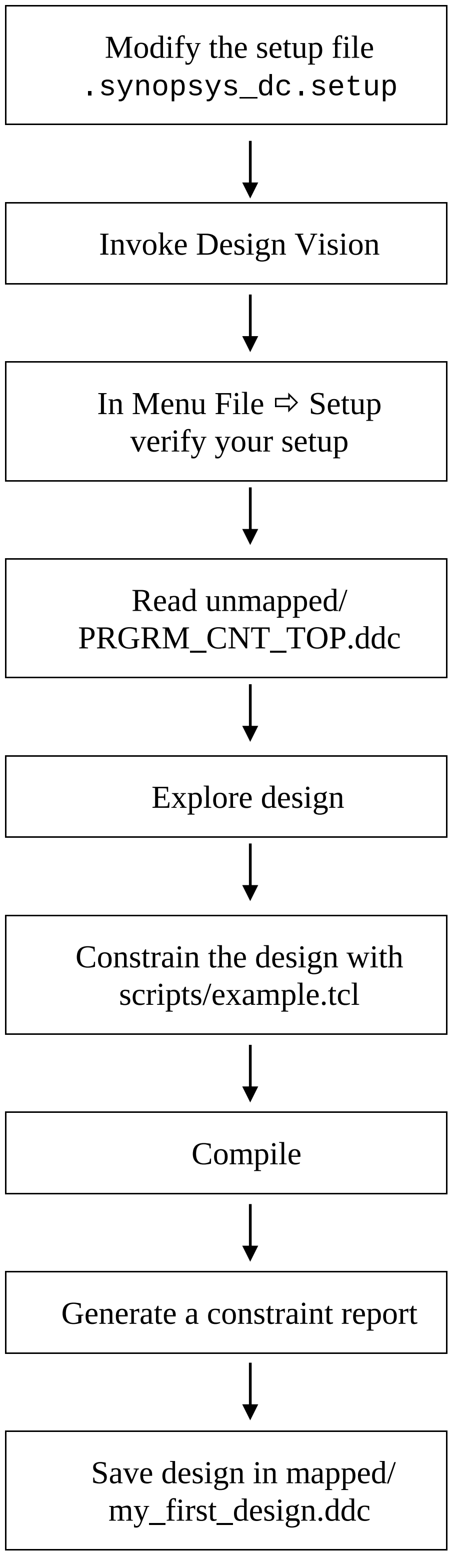
Unit 2: Design Compiler Setup and Synthesis Flow

Objectives

This unit shows you how to setup Design Compile and synthesis flow. After this unit, you should be able to do the followings:

- Use the basic features of Design Vision
- Use the Designs, Symbol and Schematic Views of Design Vision, and select menu, and mouse functions
- Take a design through the basic synthesis steps

Lab Flow


Modify the setup file .synopsys_dc.setup
Invoke Design Vision
In Menu File ⇨ Setup verify your setup
Read unmapped/ PRGRM_CNT_TOP.ddc
Explore design
Constrain the design with scripts/example.tcl
Compile
Generate a constraint report
Save design in mapped/ my_first_design.ddc

Lab Instructions

Task 1. View the .synopsys_dc.setup file

1. Make the `risc_design` directory your working directory.

```
unix% cd risc_design
```

2. Open the `.synopsys_dc.setup` file with a text editor.

3. Check if the following lines are present in the `.synopsys_dc.setup` file:

```
set target_library "core_slow.db"
set link_library "* core_slow.db"
set symbol_library "core.sdb"
```

4. Verify if the following aliases are present in the `.synopsys_dc.setup` file:

```
alias rt {report_timing}
alias ra {report_area}
alias rc {report_constraint –all_violators}
alias h {history}
alias all_gone {remove_design –designs}
```

These commands are used often; creating aliases for them makes typing task easier.

Example: **`all_gone`** removes all designs from Design Compiler memory (instead of having to type: `remove_design -designs`).

5. Verify that line editing has been enabled. This variable can only be modified in the setup file (i.e. it can not be modified after DC has been invoked).

```
set sh_enable_line_editing true
```

6. **Save** the `.synopsys_dc.setup` file, and **Quit** the editor.

Task 2. Invoke Design Vision

1. Invoke Design Vision from the **`risc_design`** directory.

(Use the **`pwd`** command to verify you are in the **`risc_design`** directory)

```
unix% pwd
unix% design_vision–xg
```

2. View the command window at the bottom of the GUI.

It is always good practice to begin by viewing the Command Window. The window displays all the executed commands, their results, and shows any error messages.

3. Choose menu **File ⇨ Setup** and verify that the libraries are set up correctly.

a. What is the Link Library? ______________

b. What is the Target Library? _____________

c. What is the Symbol Library? _____________

If the libraries are called **`your_library.db`*****, (instead of*** **`core_slow.db`*****) this indicates you invoked DC from the wrong Unix directory. Exit, make sure your current directory is*** **`risc_design`** ***and re-invoke Design Vision.***

4. Click **Cancel** to close the Application Setup window.

5. From the original Unix window from which you invoked Design Vision, type the following to confirm the library setup variables from the shell interface.

Note: Command line editing allows for command, option, variable and file completion. Type a few letters and then hit the tab key.

```
design_vision> printvar target_library

design_vision> printvar link_library

design_vision> printvar symbol_library
```

Task 3. Read a Design into DC Memory

1. Choose menu **File ⇨ Read,** double-click on directory **`unmapped/`**, and then on **`PRGRM_CNT_TOP.ddc`**.

In the “Logical Hierarchy” window on the left side of the GUI (you may need to widen the window), there is now an icon for `PRGRM_CNT_TOP`, which is the top-level of a hierarchical design. There are also icons for the lower-level instances: `I_PRGRM_FSM, I_PRGRM_DECODE, and I_PRGRM_CNT`.

2. Select **File** ⇨ **Link Design** ⇨ **OK** to link the design and resolve all references. You should not see any warning or error messages.

3. In the original Unix window from which you invoked Design Vision, type the following to see a list of designs and libraries in memory.

```
design_vision> list_designs
design_vision> list_libs
```

Task 4. Explore Designs-, Symbol-, Schematic-View

1. Explore the **Designs View** by observing all the blocks.

Select `PRGRM_CNT_TOP` (single click via the left mouse button), and look at the lower right corner to verify the selection.
You should see `Design: PGRM_CNT_TOP`

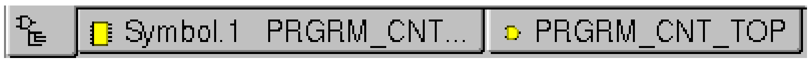
2. **Create Symbol View** of PGRM_CNT_TOP by clicking on the green icon on the Design Vision menu (top banner).

Look at the top left corner of the generated window to verify you are in Symbol View. You should see a yellow icon followed by `Symbol.1`.

3. **Create Design Schematic** of PGRM_CNT_TOP by clicking on the yellow icon on the Design Vision menu.

The schematic of `PRGRM_CNT_TOP` contains instantiations of `PRGRM_FSM`, `PRGRM_DECODE` and `PRGRM_CNT`. This is the “block diagram” of `PRGRM_CNT_TOP`.

4. You now have three windows open in the GUI: The Hierarchical, Symbol and Schematic view windows. Experiment by maximizing one of the windows. All three windows are now maximized and you can bring diferent windows to the foreground by selecting the appropriate tab below the view window. (You will learn how to “zoom” the image in the next task.)



5. Minimize the view windows or select the left-most Hierarchy tab to make the Hierarchical window visible.

6. Explore `PRGRM_CNT_TOP` by visiting the **Symbol** and **Design Schematic Views** of `PRGRM_DECODE`, `PRGRM_FSM`, and `PRGRM_CNT` by selecting each design in the **Logical Hierarchy** window and clicking on the icons and respectively.

   Because you have not compiled these designs yet, you will not see gates from the target technology library. You will see GTECH components. GTECH components are generic Boolean gates and registers that represent the generic, non-technology specific functionality of a design.

Task 5. Explore the Mouse Functions

1. **Click and hold** the **right mouse button** in a **design schematic view** to see the available mouse functions.

2. Select **Zoom Fit All** with the left mouse button to maximize the view. Now repeat **Step 1** and select **Zoom In Tool**. With the left mouse button click and drag the rectangular area you want to zoom into. Press the **[ESC]** key to exit out of the ZOOM mode. Return to **Zoom Fit all** or **Zoom Out Tool** by using the appropriate mouse function.

3. Close all window views except the PRGRM_CNT_TOP schematic window.

4. This time switch to the Design Schematic of `PRGRM_DECODE` by double clicking on the block labelled `PRGRM_DECODE` in the PRGRM_CNT_TOP schematic.

   Note the cell name **`I_PRGRM_DECODE`** in the lower right corner of Design Vision. This signifies that `I_PRGRM_DECODE` is the **Current Instance**.

   Go back to the PRGRM_CNT_TOP design schematic by clicking on the **Move up one level** button from the menu (top banner).

   Repeat **Step 4** for PRGRM_FSM, and PRGRM_CNT.

5. To select multiple objects, experiment with using your **left mouse button** and the **CTRL key**. Use the left mouse button to select the first object, then **left mouse button and CTRL key** to select additional objects. *Selected objects appear to be highlighted in white.* Click in any black area to un-select the selected objects.

Recall the Basic Steps in Synthesis Flow

The following steps will be performed in the upcoming tasks:

- Translate HDL code.

  This is normally done via `read_vhdl/read_verilog`. For this lab, `read_vhdl/verilog` has already been done, and the results were saved in the `unmapped/` directory.

  The translated design `unmapped/PRGRM_CNT_TOP.ddc` has been read into memory.
- Constrain the design
- Compile
- Generate reports
- Save the resulting netlist

Task 6. Constrain PRGRM_CNT_TOP with a Script file

1. Open the **Symbol** view for `PRGRM_CNT_TOP`.

You may also view the Design schematic but the Symbol view gives you a clearer overview of your port names.

2. Type the following at the command prompt on the bottom of the Design Vision window. Remember to take advantage of file completion by hitting the tab key.

```
design_vision> source scripts/example.tcl
```

This will execute a script file, which constrains the `PRGRM_CNT_TOP` design. You will not be able to "see" the constraints in the view window, but they are there. In upcoming labs you will learn how to generate reports to verify constraints that have been applied to a design.

Task 7. Map Program Counter to Vendor-Specific Gates

1. To compile the design, type the following command at the command prompt on the bottom of the Design Vision window:

```
design_vision> compile
```

Monitor the log as compile progresses. This table will be discussed later.

2. Explore the **Design Schematic** of `PRGRM_DECODE`, `PRGRM_FSM` and `PRGRM_CNT`. You will now see gates from the target technology library.

Task 8. Generate a Timing and Area Report

1. Go to the **Symbol View** of `PRGRM_CNT_TOP`.

2. In the original Unix window from which you invoked Design Vision, type: **`rc`**

The `rc` command is an alias that was specified in the `.synopsys_dc.setup` file. It executes the following command:

```
report_constraint -all_violators
```

A report will be generated showing any data paths containing timing and area constraint violations.

Record the following information:

Max Delay: Largest Violation (Slack) ______________

Max Area: Actual Area ______________

Note: You can also obtain max delay and max area by using the aliases **rt** (report_timing) and **ra** (report_area) respectively.

3. Go back to the **Design Schematic** of `PRGRM_CNT_TOP.`

4. Select the critical path (the path with the largest violation) from **Timing ⇨ Timing Analysis Driver**, then switch to the schematic window and choose from menu **Highlight ⇨Selected** (Ctrl+H)**.**

The critical path will be highlighted. Push into the hierarchy to see which gates are contained in this path.

To undo the highlighting select:

Highlight ⇨ Clear All (CTRL-M).

Task 9. Save the Optimized Design

1. Go back to the Symbol View of `PRGRM_CNT_TOP.`
2. Choose menu **File ⇨ Save As.**
3. Double click on the ***mapped*** directory.
4. Verify that the **Save All Designs in Hierarchy** button is selected. This will save the entire design hierarchy into a single (.ddc) file.
5. Enter `my_first_design.ddc` in the File Name field.
6. Click **save.**

   You just saved the gate-level netlist (the entire hierarchy) in 'ddc' format under the `mapped` directory. You can verify the created file in the original Unix window from which you invoked Design Vision, using '`ls -l`'.

Task 10. Remove Designs from Design Compiler Memory

1. Type **all_gone** at the command line prompt.

   Verify that all the icons in Design Vision have been deleted. The "all_gone" alias executes the following command:

```
remove_design -designs
```

   You can also use the pull down menu command **File ⇨ Remove All Designs**.

   This removes all designs and libraries from Design Vision's memory.

2. In the original Unix window from which you invoked Design Vision, type **h.**

   This will list a history of all commands you have executed since you started Design Vision.

   Note: 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 project working directory (`risc_design`), by doing the following:

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

3. Exit from Design Vision.

Questions

Question 1. How do you select multiple objects in Design Vision?

Question 2. What functions are available using your right mouse button?

Question 3. Numerically order the following steps to show the basic synthesis flow:

A. Compile
B. Read in the unmapped design
C. Generate a constraint report
D. Apply a constraint script file
E. Save the mapped design
F. Determine if constraints are met
G. Set up library variables

Question 4. How do you optimize and map a design with Design Vision?

Question 5. Which Design Vision menu item saves a design?

Question 6. What are the benefits of using synthesis in a design flow?

Question 7. What is the difference between `design_vision-xg` and `dc_shell-xg-t`?

Question 8. Why should you create a `.synopsys_dc.setup` file?

Question 9. How do you verify the library variables are set up correctly?

Question 10. How do you “read” VHDL or Verilog code into Design Vision?

Question 11. What are two optimization goals you can set on a design?

Question 12. What is the function of the `target_library` variable?

End of Unit 2

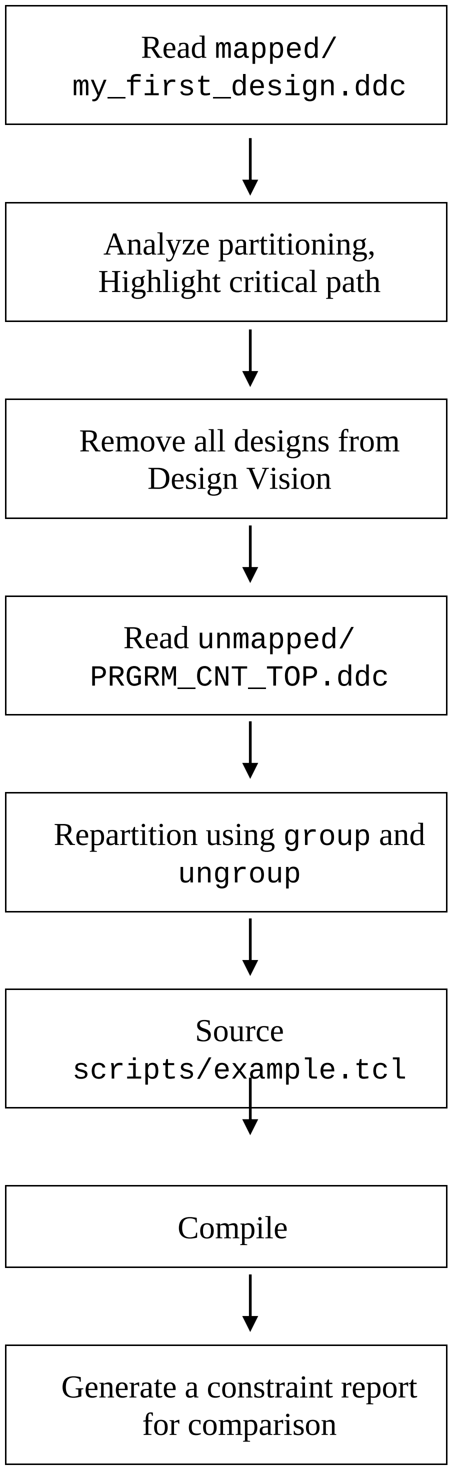
Unit 3 Partitioning for Better Synthesis Results

Objectives

After completing this lab, you should be able to:

- Improve a design’s QoR (Quality of Results = Timing and/or Area) by repartitioning a design using the **`group`** and **`ungroup`** commands

Lab Flow


Read mapped/ my_first_design.ddc
Analyze partitioning, Highlight critical path
Remove all designs from Design Vision
Read unmapped/ PRGRM_CNT_TOP.ddc
Repartition using group and ungroup
Source scripts/example.tcl
Compile
Generate a constraint report for comparison

Task 1. Analyze Partitioning of PRGRM_CNT_TOP

1. Invoke Design Vision from the **risc_design** directory.
2. **Read** and **link** the design from `my_first_design.ddc` located in the `mapped` directory (that you completed in Lab 2)
3. Generate the default "End Point Slack" histogram within DesignVision:

   `Timing -> Endpoint Slack` → **`OK`** (Accept defaults)

   Write down the "Worst" slack amount from the histogram (Look under the Left most bar (Red = Negative slack) of the histogram :

   Max Delay: Largest Violation (Slack) ______________

   Write down the Total Cell area by using **ra** (report_area)

   Total Cell Area (Max Area): Actual Area ______________

   Note: Recall that in Lab 2 Task-8, you used **rc** (report_constraint –all_violators) to obtain answers to the above questions

4. Go to the Design Schematic of `PRGRM_CNT_TOP` and highlight the critical path.

   Push into the subblocks to see the critical path start and end points. Notice that the path traverses a purely combinational block.

5. In the space below, draw the block diagram for `PRGRM_CNT_TOP`, and indicate where the critical path start and end points are. Think about which partitioning guidelines the design violates.

   How can you improve the partitioning of this design?

Original Partitioning	After Re-Partitioning

6. **Remove** all the designs from DesignVision memory (File -> Remove All Designs in GUI (or) all_gone from the command line)

Task 2. Repartition using *group* and *ungroup*

1. From Task-1 (or from Lab 2, Task-8), record the Delay and Area information into Table 3-1 below:

Task	**Compiled Design**	**Timing Slack**	**Actual (Total Cell) Area**
Task 1	Initial partitioning		
Task 3	After partitioning (group + ungroup)		

Table 3-1: Improving QoR with Partitioning

2. Read and link the unmapped design `unmapped/PRGRM_CNT_TOP.ddc`. (Re-invoke Design Vision incase you exited the tool in the previous task)

3. From the Logical Hierarchy View, select both sub-designs with the instance names `I_PRGRM_DECODE` and `I_PRGRM_CNT` (**CTRL** + **Left** mouse click).

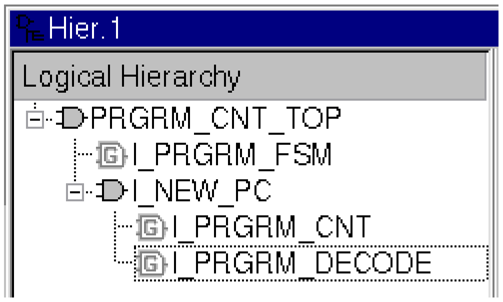
4. To confirm your selection, type the following at the command prompt.

```
design_vision> get_selection
→ {I_PRGRM_CNT I_PRGRM_DECODE}
```

5. Group these two designs together with a new design name, **NEW_PC** and new instance name, **I_NEW_PC**, by typing the following.

```
design_vision> group -design NEW_PC \
        -cell I_NEW_PC [get_selection]
```

6. Validate this last step using any of the following methods. In the Logical Hierarchy View, you should see the following.



Open up a Design Schematic of PRGRAM_CNT_TOP to see the new instance I_NEW_PC that was just created. Or type the following to report the design hierarchy.

```
  design_vision> report_hierarchy -noleaf
```

7. The next step is to ungroup everything below level two in the hierarchy for the new design **NEW_PC**. Do this by typing the following.

```
  design_vision> ungroup –start_level 2 I_NEW_PC
```

Confirm this step using the same techniques as before. Your design hierarchy should now be the desired, optimum hierarchy for synthesis.

Task 3. Compile and Analyze Results

1. Go to the Symbol or Schematic View of `PRGRM_CNT_TOP`.

Look at the bottom of the Design Vision window to make sure the current_design is `PRGRM_CNT_TOP`.

2. Execute the script file `scripts/example.tcl`.

3. Perform a default `compile` on `PRGRM_CNT_TOP`.

4. Generate a constraint report for All Violations (**rc**)

If no timing violations are reported, the design meets its timing constraints.

Record the following information and transfer them into Table 3-1 for a comparison:

Max Delay: Largest Violation (Slack) ______________

Max Area: Actual Area ______________

5. Highlight the critical path using (Ctrl-H).

Does the critical path cross any purely combinational block ? __________

6. Quit Design Vision**.** There is no need to save this design, you will not be using it again.

7. Using the results noted in Table 3-1, Did repartitioning (group + ungroup)

a) Improve Timing ? __________ By how much ? __________

b) Improve Area ? __________ By how much ? __________

Note that the results **are** design and constraints **dependent**.

Questions

Question 1. Why is it important to partition a design correctly in the source code?

Question 2. What is one reason for <u>not</u> ungrouping the entire hierarchy and compiling a “flattened” design?

Question 3. What are 3 synthesis benefits you gain from good partitioning?

Question 4. How do you implement partitioning in the RTL code?

Question 5. List two partitioning guidelines that will help reduce compile run times.

Question 6. Name one partitioning guideline that will help simplify setting constraints on a design.

End of Unit 3

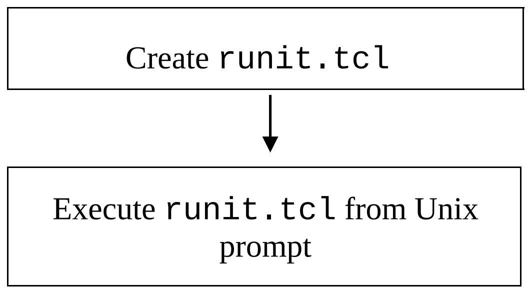
Unit 4 Introduction to DC-Tcl

Objectives

After completing this lab, you should be able to:

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

Lab Flow



Design Specifications

Script File: runit.tcl

The script file should perform the following actions:

- Read the unmapped design netlist `unmapped/PRGRM_CNT_TOP.ddc`
- Set the current design to PRGRM_CNT_TOP
- Perform a `link`
- Constrain the design by applying the `./scripts/example.tcl` file
- Perform a default compile
- Generate and save the results from a constraint report to `reports/PRGRM_CNT_TOP.rpt`

```
# Executes the report and echos the output to the screen and
# saves the results in the file reports/PRGRM_CNT_TOP.rpt
redirect –tee reports/PRGRM_CNT_TOP.rpt \
                       {report_constraint –all_violators}
```

- Save the mapped design as `mapped/PRGRM_CNT_TOP.ddc`
- Quit Design Compiler

Task 4. Create and Test *runit.tcl*

1. Under the `scripts` directory, create a `runit.tcl` file following the requirements on the previous page. Use your experience in the last few labs plus the job aid for assistance.
2. Save `runit.tcl`.
3. Return to a Unix shell, and type the following from the `risc_design` directory:

```
unix% dc_shell-xg-t -f scripts/runit.tcl | tee -i runit.log
```

The tee program is a standard UNIX program (/usr/bin/tee), which redirects the output of other programs to both a file and to a terminal window.

4. Check `runit.log` to verify that there are no errors in the script file.

```
unix% grep Error runit.log
```

5. Verify the mapped design is saved under the `mapped` directory.
6. Verify the constraint report is saved under the `reports` directory.

Task 5. Explore Useful Commands

Take a moment to explore useful commands when using Design Compiler in the shell interface.

1. Invoke Design Compiler from the `risc_design` Unix directory.

```
unix% dc_shell-xg-t
```

2. Read the mapped design created in the last task. Take advantage of command and file completion by using the tab key.

```
dc_shell-xg-t> read_ddc mapped/PRGRM_CNT_TOP.ddc
```

3. Explore other key bindings that are available and the current edit mode (the choices are emacs or vi).

```
dc_shell-xg-t> help *key*
dc_shell-xg-t> sh_list_key_bindings
```

4. Explore page mode which will display long reports or man pages one page at a time. Use two aliases that have been defined in the .synopsys_dc.setup file to turn page mode on and off. Recall the alias **rc** generates a constraint report which is several pages long.

Type "q" to quit from a long report in page mode to return to the DC prompt without reading the entire report.

```
dc_shell-xg-t> page_off
dc_shell-xg-t> rc
dc_shell-xg-t> page_on
dc_shell-xg-t> rc
```

Note: The following will display the expansions for the given aliases.

```
dc_shell-xg-t> alias page_on
dc_shell-xg-t> alias page_off
```

5. Generate man pages and take advantage of the view utility. The view utility is a script that is available on SolvNet, Doc ID 014947, and allows you to display long reports in a pop-up window with a scroll bar.

```
dc_shell-xg-t> man sh_enable_page_mode
dc_shell-xg-t> view man sh_enable_page_mode
```

Note: Alternatively, take advantage of two aliases that have been created in the setup file to generate timing reports and man pages using view.

```
dc_shell-xg-t> vman sh_enable_page_mode
dc_shell-xg-t> vrt
```

End of Unit 4

Unit 5 Apply Timing Constraints to PRGRM_CNT_TOP

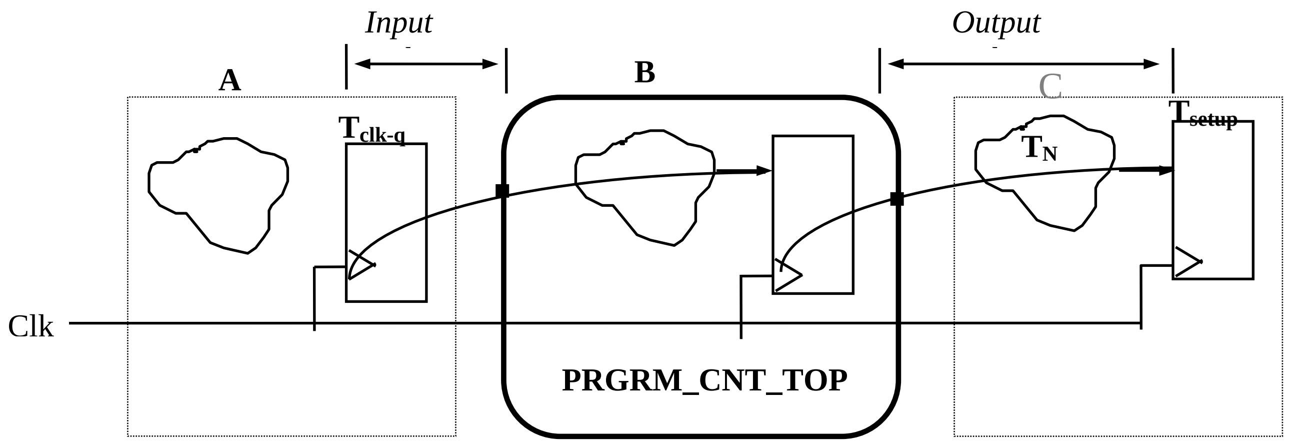
Objectives

After completing this lab, you should be able to:

- Apply timing constraints to a design

Getting Started

Use the drawing below to determine the input and output delays for setup time calculations.



Specifications

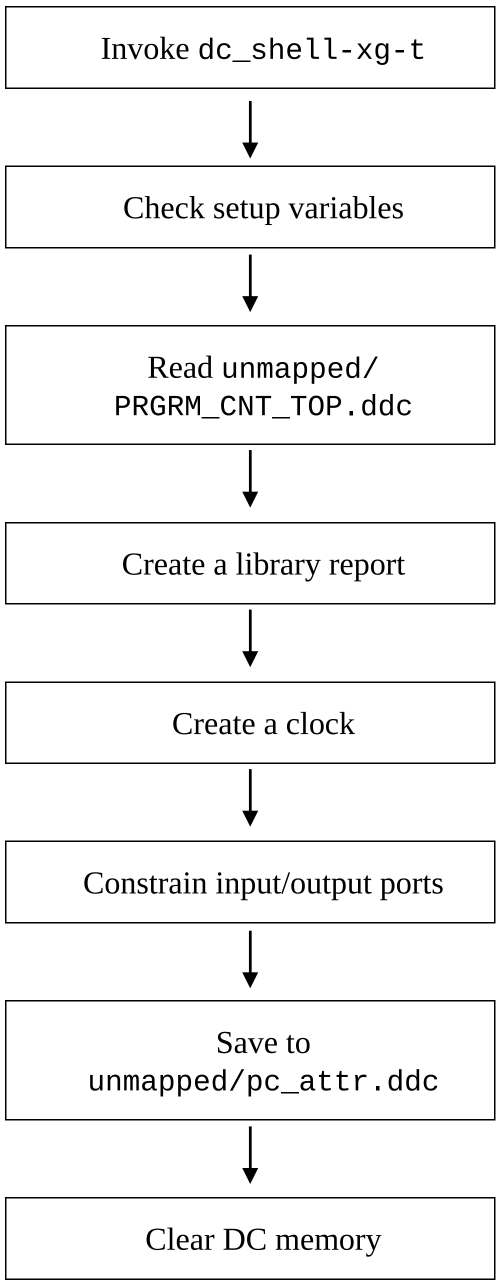
Timing Goal	Period = 4ns 50% duty cycle
Clock Skew	0.25 ns
Input Ports	T_{clk-q} = 1ns
Output Ports	All output ports are registered
Area Goal	None

Question 1. From the specifications above, what are the input/output delays?

Input Delay = ________________

Output Delay = _______________

Lab Flow


Invoke dc_shell-xg-t
Check setup variables
Read unmapped/PRGRM_CNT_TOP.ddc
Create a library report
Create a clock
Constrain input/output ports
Save to unmapped/pc_attr.ddc
Clear DC memory

Task 1. Invoke dc_shell and read PRGRM_CNT_TOP

1. Invoke Design Compiler in the `risc_design` directory.

```
unix% cd risc_design
unix% dc_shell-xg-t
```

2. Check the definition of the setup variables:

```
# Take advantage of command and variable completion with the tab key
printvar target_library
printvar link_library
```

Note: Make sure target_library is "core_slow.db" and link_library is "* core_slow.db." If the library variables are not correctly set, then DC was not invoked from the correct directory. Quit DC and invoke it from the correct directory. Remember, the library variables were assigned in the setup file located under the `risc_design` directory.

3. Read the design **unmapped/PRGRM_CNT_TOP.ddc**.

Note: Use `help read*` to determine which command to use, or use your job aid.

4. Link the design PRGRM_CNT_TOP.

Task 2. Examine the core_slow Library

1. Generate a report for the target library.

The name of the technology library may be different from the name of the library file name. Use the following to list the libraries in memory, the file name for each and the library names.

```
dc_shell-xg-t> list_libs
```

Question 2. What libraries are in DC memory?

....

Question 3. What is the technology library name for the target library?

....

2. Use that library name for `lib_name` in the following report.

Note: Because this is a long report, either turn on page mode or use the view utility, as shown below.

```
dc_shell-xg-t> page_on
dc_shell-xg-t> report_lib lib_name
OR
dc_shell-xg-t> view report_lib lib_name
```

The file name for this library is “`core_slow.db`”. This name is NOT used in the above command!

At the top of the generated report, you will find the library units. The remainder of the report shows the available cells.

Question 4. What is the Time Unit?

....

Question 5. What is the Capacitive Load Unit?

....

Task 3. Constrain PRGRM_CNT_TOP

In this task you will constrain the design for timing. In order to keep track of all the constraints set on the design and to easily reapply or modify these constraints later, you will create a file called **`constraints.tcl`**. After manually entering and successfully executing the commands in the following steps you will copy & paste them into the **`constraints.tcl`** file.

1. Use a text editor to create a new file called **`constraints.tcl`**.

Note the location of this file. You will need it the next lab.

2. Enter the following commands at the `dc_shell-xg-t` prompt. After you successfully execute each command, copy & paste the command into the **`constraints.tcl`** file.

3. “reset” the design.

This will clear any existing constraints, which may have been placed on the design.

```
dc_shell-xg-t> reset_design
```

4. Create a 250Mhz clock object on the port `Clk`.

Use the `all_inputs` command to see all input port names (optional step).

Execute the following command to create a clock, named **my_clk**, with a period of 4 ns (the name of the clock port is **`Clk`**):

```
dc_shell-xg-t> create_clock –period 4 -name my_clk [get_ports Clk]
```

5. Execute the following command to model clock skew for **`my_clk`**:

```
dc_shell-xg-t> set_clock_uncertainty 0.25 [get_clocks my_clk]
```

6. Constrain the input ports for timing.

Apply the `set_input_delay` command on all the input ports **except Clk**:

```
dc_shell-xg-t> set_input_delay 1 –max –clock my_clk \
              [remove_from_collection [all_inputs] [get_ports Clk]]
```

The command `remove_from_collection` removes objects in the second argument from the first. This will result in the desired collection.

7. Constrain the output ports.

Since the outputs are registered, the output delay constraint should be ($T_{period} – T_{clk-Q}$).

```
dc_shell-xg-t> set_output_delay 3 –max –clock my_clk [all_outputs]
```

8. Validate the following in the constraints Tcl file you have created.

```
# ./constraints.tcl
reset_design
create_clock –period 4 -name my_clk [get_ports Clk]
set_clock_uncertainty 0.25 [get_clocks my_clk]
set_input_delay 1 –max –clock my_clk \
  [remove_from_collection [all_inputs] [get_ports Clk]]
set_output_delay 3 –max –clock my_clk [all_outputs]
```

Task 4. Check your work and Save the Results

1. Generate a report for the clocks.

```
dc_shell-xg-t> report_clock –skew -attribute
```

Verify that the clock period is 4, the skew is 0.25 and the source of the clock waveform is `Clk`.

2. Generate a report for all the ports of your design. Use the view utility as this is a long report

```
dc_shell-xg-t> view report_port -verbose
```

In the Input Delay section, verify the Max Rise and Fall numbers for input delay are set for all input ports (except *Clk*) and that they are related to *my_clk.*

In the Output Delay section, verify the Max Rise and Fall numbers for output delay are set for all output ports and that they are related to *my_clk.*

Enter the following command to see all contraints on one port:

```
dc_shell-xg-t> report_port -v Clk
```

This command displays all constraints and attributes of the clock port only, and you do not have to scroll up and down.

3. All constraints are set, **save** the design.
You can use this saved version next time and not have to reenter the above commands.

Be sure to use **`–hierarchy`** to save all design hierarchies:

```
dc_shell-xg-t> write –format ddc –hierarchy \
                        –output unmapped/pc_attr.ddc
```

Saving the design in a ddc format also saves the timing and area constraints.

4. Remove designs from DC memory.

```
remove_design -designs
```

To clear the DC memory of all designs and ***libraries****, you would use the following command. (****No need to enter this!****)*

```
remove_design -all
```

Questions

Question 6. After reading the unmapped `ddc` file for `PRGRM_CNT_TOP`, why should you reset the design before applying design contraints?

Question 7. Why was it important to check the library time units before setting constraints?

Question 8. Write the `dc_shell-xg-t` command for setting a max area goal of 500.

Question 9. Which has a higher priority—timing or area goals?

Question 10. The command `set_max_area` places an attribute on a design. What is the name of this attribute?

Question 11. How do you check what area goal has been placed on a design?

End of Unit 5

Unit 6 Apply Environmental Attributes

Objectives

After completing this lab, you should be able to:

- Apply operating conditions, a wire load model, and port environment attributes on a design
- Save the attributes and constraints set on the `current_design` using the `write_script` command

Specifications

Voltage and Temperature Variation	1.8V +/- 0.18V, 0°C to 125°C
Default Register Driving input ports (Except for Clk port)	cell "**fdef1a1**", pin "**Q**" **<u>Note</u>:** fdef1a1 contains two of the letter "**one**" character, and **<u>no</u>** **"L"** characters
Wire Load Model	Auto Wire Load Selection
Max Capacitance Allowed on an input ports: (except for Clk port)	5 "**and2a1**" cells, pin "**A**"
Number of blocks each output port must be able to drive:	3
Assumed load on output ports	5 x 3 "**and2a1**" cells, pin "**A**"

To Get You Started

Question 1. Under what temperature and voltage conditions will the worst case (slowest) operating condition occur?

Generate a library report for the `core_slow.db` library file.

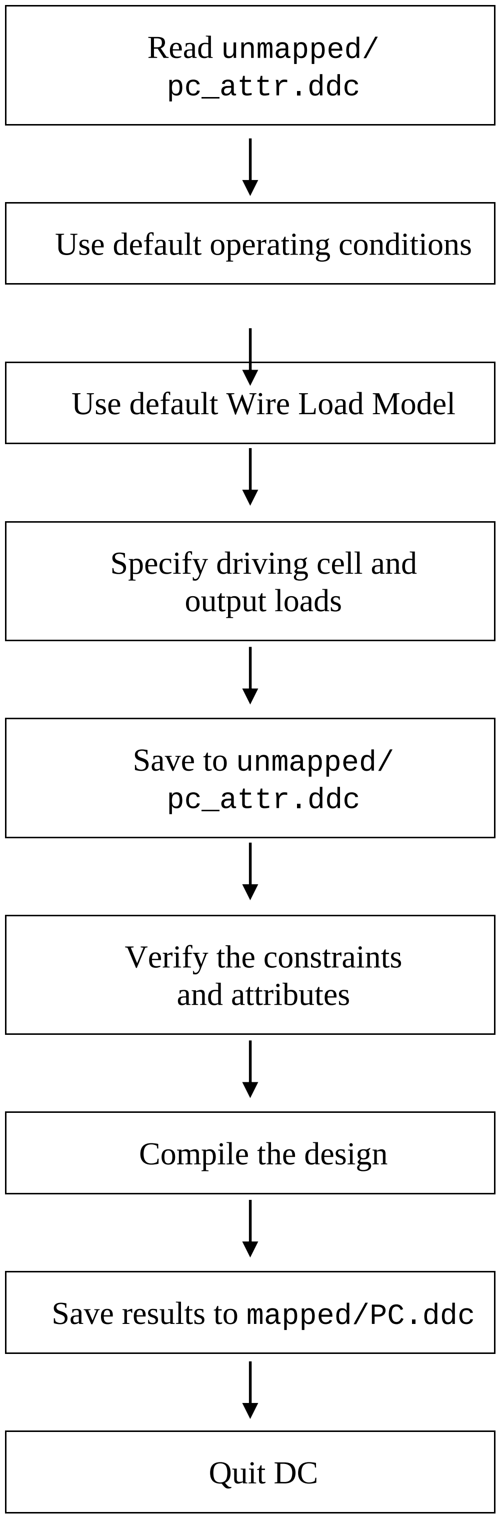
Question 2. Which operating conditions are available?

Question 3. From the library report for the `core_slow.db` library file, which wire load model will be used?

Question 4. Name two DC commands you can use to check your work for this lab?

Question 5. If you do not model the port environment accurately, what is likely to happen?

Lab Flow


Read unmapped/ pc_attr.ddc
Use default operating conditions
Use default Wire Load Model
Specify driving cell and output loads
Save to unmapped/ pc_attr.ddc
Verify the constraints and attributes
Compile the design
Save results to mapped/PC.ddc
Quit DC

Task 1. Apply Environmental Constraints and Attributes to PRGRM_CNT_TOP

After you apply a new constraint and verify that it works correctly, add the command to your constraint script constraints.tcl. This will simplify future reapplication of constraints.

1. If you closed the `dc_shell-xg-t` session, re-start `dc_shell-xg-t` and read the design "`Program Counter`" that you constrained for timing in the previous lab.

```
dc_shell-xg-t> read_ddc unmapped/pc_attr.ddc
dc_shell-xg-t> link
```

2. Operating conditions: You will be using the default operating condition that is specified in the technology library `core_slow.db`. Do not execute any command.

Note: If you wanted to modify the operating conditions for your design you would use the **`set_operating_conditions`** command if your library contained operating condition models, or you would modify the name of your `target_` and `link_library` to point to the appropriately characterized technology library.

Question 6. What is the default operating condition?

....

```
dc_shell-xg-t> get_attribute ssc_core_slow \
                           default_operating_conditions
```

3. Wire load model: The library you are using has automatic wire load model selection, which means that DC automatically picks the wire load model based on the area of the block.

Note: If your library does not use automatic wire load selection you would specify the wire load model (WLM) using the command **`set_wire_load_model`**.

Create a library report to answer the following questions.

```
dc_shell-xg-t> view report_lib ssc_core_slow
```

Question 7. How many wire load models are defined in core_slow.db?

....

Question 8. What WLM would DC pick if the block had a size of 200,000?

....

Question 9. What is the base unit for resistance in that WLM?

....

4. Model the port environment.

Use the **`set_driving_cell`** command to model a flip-flop driving the non-clock input ports. Use the flip-flop specified in the table at the beginning of the lab.

The format for the command is:

```
set_driving_cell –lib_cell <cellname> \
                 -pin <pin name> <port list>
```

Remember, this will allow DC to use the characteristics of the "driving cell" to more accurately calculate the delays of the input gates.

If you entered `[all_inputs]` as your port list, you need to remove and verify the driving cell from the clock port by executing the following two commands:

```
remove_driving_cell [get_ports Clk]
report_port -v Clk
```

5. Report the attributes of the driving cell.

Using the command `set_driving_cell` without the –no_design_rule option produces a warning UID-401, telling you that you have applied a design rule at the input ports. The UID-401 message is, however, suppressed in the .synopsys_dc.setup file. Use the following command to display all attributes of the pins for the driving cell:

```
report_attribute [get_pins ssc_core_slow/fdef1a1/*]
```

Question 10. What are the pin names of the cell fdef1a1?

....

Question 11. What is the **max_capacitance** design rule value?

....

6. Use the `set_max_capacitance` command to limit the input port capacitance as described by the specification.

The format for the command is:

```
set_max_capacitance <max_load> <port list>
```

You will need to specify the load using the `load_of` command as shown in lecture.

7. Use the **`set_load`** command to specify the worst possible output capacitance. Use the load value as specified in the table at the beginning of the lab.

   The format for the command is:

```
set_load <capacitance value> <port list>
```

   Use the `load_of` command to specify the capacitance value, to more accurately calculate the delay of the gates driven by the output ports.

8. New constraints have been applied; you should have added the following or equivalent lines to your **`constraints.tcl`** file:

```
# ./constraints.tcl
set all_in_ex_clk \
  [remove_from_collection [all_inputs] [get_ports Clk]]
set_driving_cell –lib_cell fdef1a1 \
              -pin Q $all_in_ex_clk
set max_cap [expr [load_of ssc_core_slow/and2a1/A] * 5]
set_max_capacitance $max_cap $all_in_ex_clk
set_load [expr 3 * $max_cap] [all_outputs]
```

Task 2. Check your work and Save the Design

1. Generate a design report using the view utility:

```
dc_shell-xg-t> view report_design
```

   Verify that the `slow_125_1.62` operating condition model and the `5KGATES` wire load model (which is the default when design area is not known) are applied to design `PRGRM_CNT_TOP`.

2. Generate a verbose port report using the view utility:

```
dc_shell-xg-t> view report_port -verbose
```

   Make sure that a Pin Load of 0.03 is applied to all output ports. Confirm the input and output delays from the previous exercise are still in effect. Verify that the `fdef1a1` register is driving all the input ports except `Clk`.

3. Save your work to **unmapped/pc_attr.ddc**. Make sure you save all designs in the hierarchy. There will be more constraints to add-in the next lab.

4. Enter the following command to save all constraints and attributes of your design for future reference.

```
dc_shell-xg-t> write_script -output scripts/pc_w.tcl
```

   Look at the newly created `pc_w.tcl`. The script file will contain all the constraints and attributes you set on the design `PRGRM_CNT_TOP`.

5. Perform a test-ready compile.

```
dc_shell-xg-t> compile -scan
```

   The **–scan** option insures that DC maps to scannable flip-flops instead of regular D-flip-flops. This topic will be further discussed during lecture.

6. Generate a constraint report.

   Question 12. Did you meet timing?

   ....

7. **Save** the design as `mapped/PC.ddc`. Make sure you save all designs in the hierarchy.

8. **Quit** Design Compiler.

Questions

Question 13. Why would you want to model pin capacitive load on the output ports in addition to the timing constraints?

Question 14. What 3 pieces of information can the wire load model provide?

Question 15. List at least five `dc_shell-xg-t` commands that you have used in this and the previous lab to place constraints and attributes on a design:

End of Unit 6

Unit 7 Design Rules and Min Timing

Objectives

After completing this lab, you should be able to:

- Apply design rules and hold time constraints
- Fix design rule violations
- Fix hold time violations

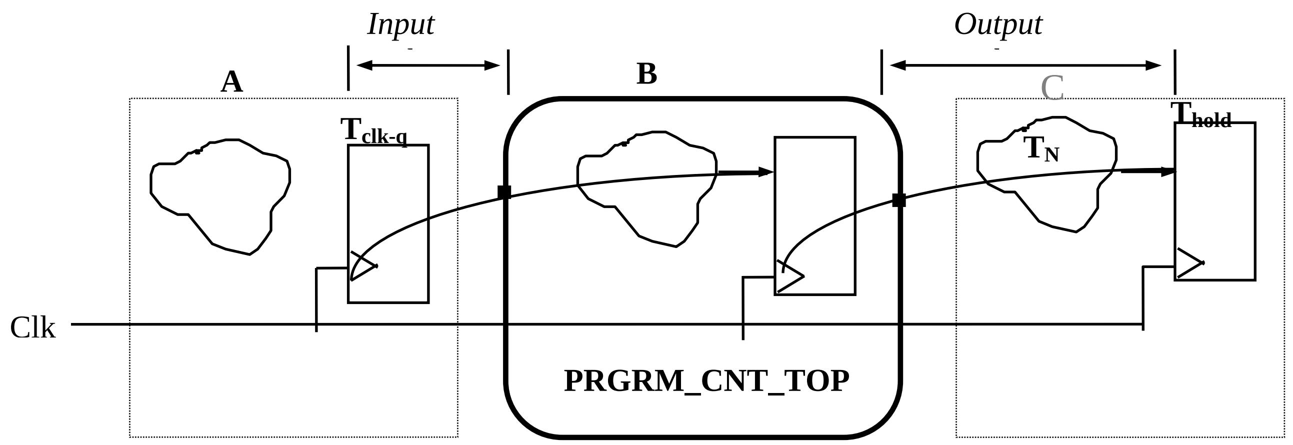
Design Specifications

The table below is a summary of constraints from Units 5 and 6. The design rule and hold time constraints shown in the second table are additional constraints that will be applied during this lab.

Clock Frequency	250 Mhz (4 ns) 50% duty cycle
Clock Skew	0.25 ns
Input Ports (worst case input delay)	$T_{clk\text{-}q}$ = 1ns
Output Ports (worst case output delay)	All output ports are registered
Area Goal	None
Voltage and Temperature Variation	1.8V +/- 0.18V, 0°C to 125°C
Default Register Driving input ports (except for Clk port)	cell "**fdef1a1**", pin "**Q**"
Wire Load Model	**Auto Wire Load Model**
Wire Load Mode	**enclosed**
Max Capacitance Allowed on an input ports (except for Clk port)	5 "**and2a1**" cells, pin "**A**"
Number of blocks each output port must be able to drive	3
Assumed load on output ports	5 x 3 "**and2a1**" cells, pin "**A**"
Max Transition driving input ports(except for Clk port)	**0.25 ns**
Input Ports (best case input delay)	**$T_{clk\text{-}q}$ = 0.2 ns**
Output Ports (best case output delay)	FF hold requirement = 0.5 ns External logic delay T_n = 0.2 ns

To Get You Started

Use the drawing below to determine the input and output delays for hold time calculations.

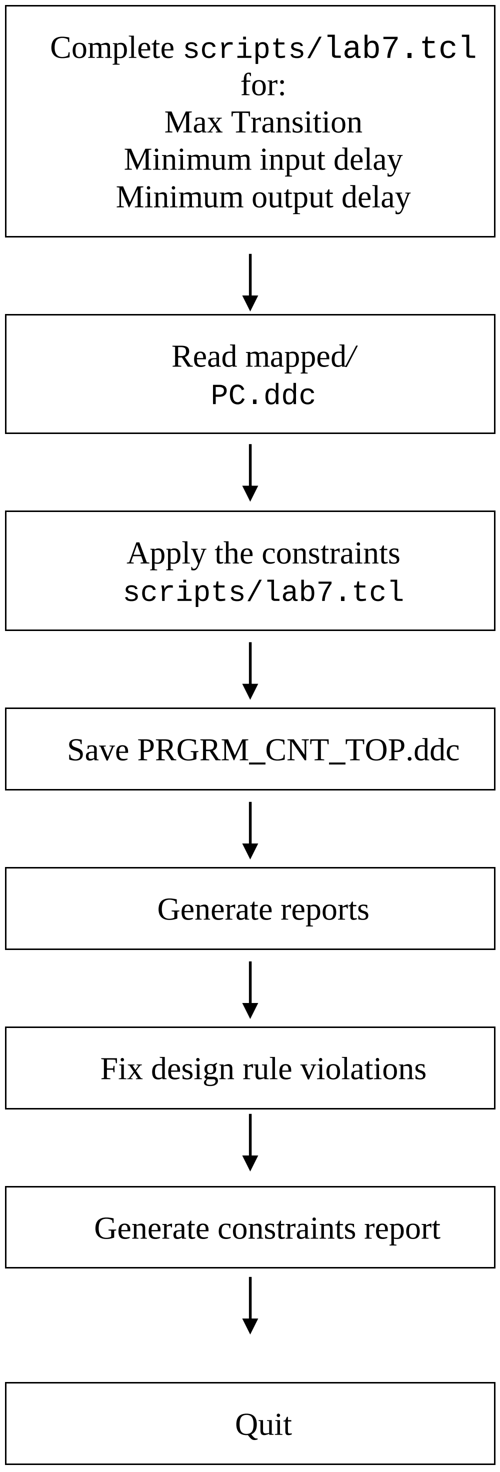


Question 1. From the specifications above, complete the commands below by filling in the blanks:

```
set ALL_INS_EX_CLK [remove_from_collection \
                      [all_inputs] [get_ports Clk]]

set_max_transition ____________________________________

set_input_delay –min ______ -clock my_clk $ALL_INS_EX_CLK

          set_output_delay –min ______ -clock my_clk [all_outputs]
```

Question 2. If you do not constrain the ports accurately, what is likely to happen?

Lab Flow


Complete scripts/lab7.tcl for:
Max Transition
Minimum input delay
Minimum output delay
Read mapped/ PC.ddc
Apply the constraints scripts/lab7.tcl
Save PRGRM_CNT_TOP.ddc
Generate reports
Fix design rule violations
Generate constraints report
Quit

Task 1. Complete `lab7.tcl` Script file

1. Complete the existing `scripts/lab7.tcl` file with a text editor.

Task 2. Read Mapped PC.ddc

2. Start `dc_shell-xg-t` and read the previously mapped design PC.ddc.

3. Apply `scripts/lab7.tcl` to the PRGRM_CNT_TOP design.

Remember that PC.ddc was previously mapped and saved with applied constraints. Sourcing lab7.tcl will just add more constraints to the PRGRM_CNT_TOP design.

4. Save PRGRM_CNT_TOP.ddc for use in the next lab.

```
write –f ddc –hier –output mapped/PRGRM_CNT_TOP.ddc
```

Task 3. Generate Reports

5. Generate a constraints report for all violations using the: `report_constraint` command.

PRGRM_CNT_TOP now violates both the hold time and the design rule max_transition, because of the design rule and hold time constraints you just added.

6. Generate a detailed hold time report using the `report_timing` command.

```
dc_shell-xg-t> report_timing –delay min
```

Notice how clock skew (0.25 ns) and output constraint (0.30 ns) contribute to data required time of 0.55 ns. In order to meet this hold time requirement, the shortest delay on output ports cannot be faster than 0.55 ns. A detailed discussion on timing reports will be covered during lecture.

Task 4. Fix Design Rule Violations and Hold

7. Execute the following command to fix the design rules and hold time violations.

```
dc_shell-xg-t> set_fix_hold [all_clocks]
dc_shell-xg-t> compile –inc –map_effort high
```

8. Generate a constraints report for all violations. The report should not have any violations.

9. **Quit** Design Compiler. (Do not overwrite mapped/PC.ddc file.)

End of Unit 7

Unit 8 Timing Reports

Objectives

After completing this lab, you should be able to:

- Interpret various timing reports generated by `report_timing`

Timing Reports Exercise

Task 1. Load Mapped PRGRM_CNT_TOP Design

1. Start `dc_shell-xg-t`, and read the mapped/`PRGRM_CNT_TOP.ddc` file you saved from the previous lab.

Task 2. Generate and Interpret Four Timing Reports

2. Answer the following questions regarding `PRGRM_CNT_TOP`.

Question 1. Are there any unconstrained timing paths in `PRGRM_CNT_TOP`? (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_TOP`? (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?

End of Unit 8

Optional Unit 9 Multiple Clocks and Timing Exceptions

Objectives

The goal of this lab is to give you a better understanding of how static timing analysis works and how timing exceptions are properly applied.

After completing this lab, you should be able to:

- Fully constrain and analyze a design using virtual clocks, false paths and multicycle paths.

Background

The design you will be working with, called "test", contains parallel paths shown in the figure below (resets not shown). Between inputs `adr_i` and `coeff`, and output `dout` there is a purely combinatorial path as well as a sequential path.

The combinatorial and sequential paths have different constraints. When you perform timing analysis, you will discover that there are initially many violations – all due to incomplete or incorrect constraints.

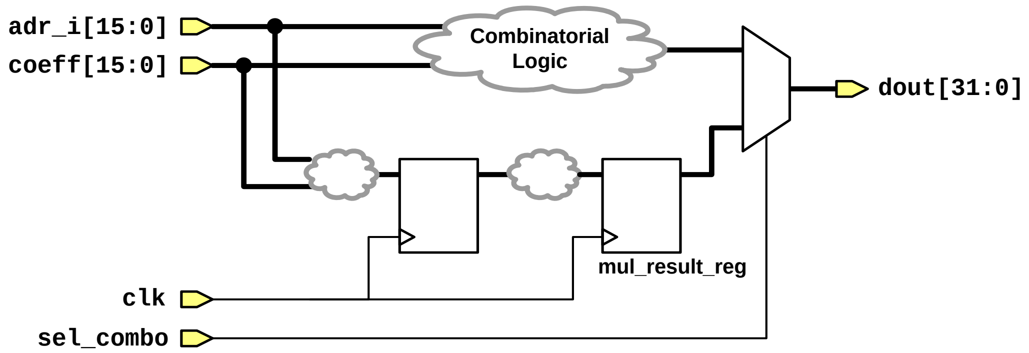


Figure 1. Design "test"

Task 1. Read a Mapped Design

1. Make sure your current working directory is `risc_design` and invoke `dc_shell-xg-t`.

2. Read the design "test" into memory – it has already been compiled for you.

```
dc_shell-xg-t> read_ddc mapped/test.ddc
dc_shell-xg-t> current_design test
dc_shell-xg-t> link
```

3. Use the view utility to generate a default timing report.

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

Question 1. How large is the worst negative slack in comparison with the data required time?

....

Question 2. Describe the start and end points of this violating timing path?

....

Question 3. What is the maximum path delay constraint for this critical path (= Clock Period – Input Delay – Output Delay)?

....

This path is over constrained - it cannot meet a –1 ns maximum delay!

This design was given improper constraints for demonstration purposes. *The maximum propagation delay through the combinational path should be constrained to 10 ns.* The current input and output delay constraints are appropriate for the sequential paths (from the input ports to a flip-flop and from flip-flops to the output ports).

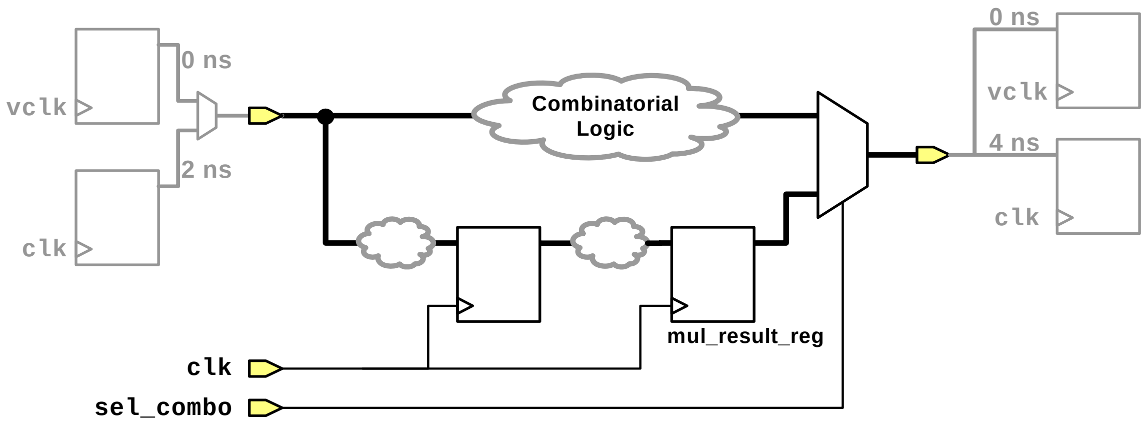
The constraints must be expanded such that the different paths are constrained separately.

Task 2. Use Virtual Clocks

Use virtual clocks to constrain the combinatorial path separately from the sequential timing paths.

4. Apply a constraint to the combinatorial logic that is independent of the constraints that apply to the sequential logic paths.

To accomplish this, constrain the combinatorial paths with respect to a different clock (**vclk**) than the sequential paths; in this way multiple constraints can be overlaid without interfering with each other. The following schematic illustrates this concept:



The diagram shows that two clocks, **clk** and **vclk** are now clocking the design. The sequential path is constrained using **`clk`**, and the combinatorial path using the virtual clock **`vclk`**.

The combinatorial path must have a maximum delay of 10 ns so **`vclk`** will have a period of 10 ns and the input and output delays will be zero.

5. Execute the following commands:

```
dc_shell-xg-t> create_clock –name vclk –period 10
dc_shell-xg-t> set in_ports [get_ports "coeff* adr_i*"]
dc_shell-xg-t> set_input_delay 0 –clock vclk –add_delay $in_ports
dc_shell-xg-t> set_output_delay 0 –clock vclk –add_delay \
                            [all_outputs]
```

6. Generate a timing report for a timing path constrained by the virtual clock.

The switch **–group** will generate a timing report for the path group **vclk** (i.e. the timing path with the worst slack for setup captured by the clock **vclk**).

```
dc_shell-xg-t> vrt –group vclk
```

Question 4. Does the path violate or meet timing?

....

Question 5. Describe the launch and capture clocks and does this match with your expectations?

....

7. The clocks **clk** and **vclk** should not interact. Use false paths to turn off timing analysis between these clocks.

```
dc_shell-xg-t> set_false_path –from [get_clocks clk] \
                              –to [get_clocks vclk]
dc_shell-xg-t> set_false_path –from [get_clocks vclk] \
                              -to [get_clocks clk]
dc_shell-xg-t> vrt –group vclk
```

Question 6. Does this path violate or meet timing?

....

Question 7. Describe the launch and capture clocks and does this match your expectations?

....

8. This constraining situation is not quite over! Try the following report and answer the following questions.

```
dc_shell-xg-t> vrt –group clk
```

Question 8. Does this path violate or meet timing?

....

Question 9. Describe the start and end points of this timing path and does this match your expectations?

....

9. As the final step, turn off timing analysis from the clock **clk** to **clk** on these combinatorial timing paths using false paths.

This final timing report will confirm that the combinatorial paths are constrained by **vclk** and these paths meet timing.

```
dc_shell-xg-t> set_false_path –from [get_clocks clk] \
                -through $in_ports \
                -through [all_outputs] –to [get_clocks clk]
dc_shell-xg-t> vrt –from $in_ports –to [all_outputs]
```

Task 3. Constrain a Multicycle Path

The path to the `mul_result_reg` is a multicycle path, which is allowed to take up to 3 clock cycles (instead of 1, the default) for setup. The hold checks should be done at zero. These paths are not yet correctly constrained.

10. Execute a timing report for the clock **clk**.

```
dc_shell-xg-t> vrt –group clk
```

Question 10. How large is this violation in comparison to the clock period?

....

11. Apply a multicycle path of 3 cycles for setup to all paths that end at the `D0` pin (a zero, not an O) of `mul_result_reg`:

```
dc_shell-xg-t> set_multicycle_path 3 –setup –to mul_result_reg*/D0
```

12. Verify the multicycle path exception was applied correctly:

```
dc_shell-xg-t> vrt –to mul_result_reg*/D0
```

Question 11. What is the effective clock period for this timing path (the clock capture edge – the clock launch edge)?

....

13. Generate a hold timing report to these same end points.

```
dc_shell-xg-t> !! –delay min
```

Question 12. What are the launch and capture clock edges and does this match the specification described at the beginning of this task?

....

14. Complete the multicycle path constraint by including a constraint for hold.

```
dc_shell-xg-t> set_multicycle_path 2 –hold –to mul_result_reg*/D0
dc_shell-xg-t> !vrt
```

Question 13. What are the launch and capture clock edges and are they now correct for hold ?

....

15. **Quit** Design Compiler.

End of Unit 9

Optional Unit 10 Automated Chip Synthesis

Objectives

During this lab, you will explore the results created by **acs_compile_design**. You will then prepare to do an incremental compile based on module-level constraints generated from the first-pass gate-level netlist.

Lab Instructions

Your goal is to explore the results created by an automatic 'bottom-up' tool. You will start by exploring the results of an initial ACS run – then will prepare to make a subsequent run.

Task 1. Explore Results of Pass0

The first **acs_compile_design** has been completed, resulting in a pass0 directory structure. This was done for you to save classroom time.

If you are interested in browsing, the run script used to create the initial results is at Lab11/scripts/run_acs.tcl.

1. Change Unix directories to **Lab11**. You may have to go up one level to see this directory.
2. Invoke a web browser from a Unix prompt.

```
unix% mozilla &
# Or alternatively
unix% netscape &
```

3. Use the appropriate menu options on your web browser to open the file **Lab11/ACS_results_RISC_CORE.html**. When the page opens, select **pass0** to see the results from the initial compile.

   Use the information on this page, as well as the links to **report_timing** and **report_area** to answer the following questions.

   Question 1. What is the critical path slack for the path group 'my_clk'?

   ....

   Question 2. What is the total cell area for the top-level design?

   ....

   Question 3. How many compile partitions were automatically defined by ACS?

   ....

Task 2. Read the Compiled Design; Prepare a new Run

After the first run, prepare to do an incremental compile.

4. Invoke Design Compiler in XG mode from the **Lab11** Unix directory.

5. Execute the following command.

This command will do the following: read the RISC_CORE modules generated in pass0; prepare new constraints and scripts based on the pass0 results; put those new constraints and scripts in a directory called 'pass9'.

This command will <u>not</u> compile the design due to the switch **–prepare_only**.

```
dc_shell-xg-t> acs_refine_design RISC_CORE -source pass0 \
                          -destination pass9 –prepare_only
```

6. Quit Design Compiler.

7. Change directory to **./pass9** and answer the following questions.

Question 4. Which of the **pass9** subdirectories are currently populated?

....

Question 5. Where is the makefile?

....

Question 6. If the preparations for the compile meet your approval, how would you execute this compile to further refine the gate-level netlist?

....

ACS uses Makefiles, which check timestamps and results before executing commands; thus, if you do anything in this lab more than once, ACS may tell you that the step is not necessary, and not do it. So, if you want to try this lab more than once, remove the relevant pass0 or passn directories (or use the –force switch).

8. If time permits, fully execute the ACS recompile command.

```
dc_shell-xg-t> acs_refine_design RISC_CORE –force –source pass0 \
                        -destination pass9
```

End of Unit 10