Lab #6: Sequential Traffic Light Control in VHDL

6. Introduction

In this lab, you will develop finite state machines to control traffic lights, starting with a very simple controller in part 1, and then progressively adding features in subsequent parts. Your FSMs will be tested on the real traffic lights in the classroom. The pin assignments for the traffic lights and the control panel are shown in a file in the lab folder on Blackboard (BB).

6.1 Basic Traffic Light Controller (each light pattern = 1 clock)

The aim of this part is to create a finite state machine that will step through the sequence of lights as follows: RG → RY → RR → GR → YR → RR → RG →…. Each pattern should appear for exactly one clock period.

1. Create a new folder called Lab6P1. Download **Lab6P1.zip** from the BB and upzip it to this folder. Create a Quartus **Project** called **Lab6P1**, configure it for the correct **device** and pin **voltages** as usual, and **add** the template file **BasicControl.vhd** (from Lab6P1.zip) to this project and set it as the top-level entity.
2. **Open** the file and edit it to implement your controller, noting the following:
   - Read though the entire given file before developing your state machine.
   - Follow the instruction shown in comments.
   - Add your name on the top comments after "Engineer".
   - Add comments to your own statements as you work out the VHDL code.
   - Compile before moving to the next step.
3. Simulate your VHDL code in ModelSim using the provided Lab6P1.do. Save a picture of the waveform after you have checked the signals carefully and are sure it shows the correct behavior.

   Show your simulation waveform to the TA/instructor.
4. Open **Pin Planner** and assign pins to the signals using the file called "ECE Lab Traffic Lights Pin Assignments for DE10-Standard" on BB. **Compile**, program, and **test** your circuit on the **traffic light station**. The turning knob on the control box can be used to adjust the speed of the clock signal.

   Demonstrate to the TA/instructor.

6.2 Timed Traffic Light Controller

In this part, you are to modify your 6.1 traffic controller to consider an intersection of a farm road facing the left-side lights and a highway facing the right-side lights. We plan to give the highway side more green time.

1. Create a new folder called Lab6P2. Download **Lab6P2.zip** from the BB and upzip it to this folder. Create a Quartus **project** called **Lab6P2**, configure it for the correct **device** and pin **voltages** as usual, and **add** the template file **TrafficControl.vhd** (from Lab6P2.zip) to this project and set it as the top-level entity.
2. **Open** the VHDL file and edit it to implement your controller. You may choose to copy and paste your Part 1 process here and modify it for the new requirement of this part. Note the following:
   - The extra bit in the lights patterns of Part 1 is not needed any more because you will concatenate a count value with the state vector for state sequencing.

- The **RG** pattern now should appear for **seven** clock cycles (to let highway traffic flow), and each of the remaining patterns should last for **two** clock cycles.
- One of the ways to implement different cycles for different states is to use a count value with the state. In this case, define seven constants for counting the cycles of the states. For example, the first two counts can be:
  ```
  constant C0: std_logic_vector(2 downto 0) := "000";
  constant C1: std_logic_vector(2 downto 0) := "001";
  ```
- Then define a state signal with **9 bits** to store the state of the controller. The value of the state signal can be a concatenation of the cycle count and the traffic light pattern to keep track of which cycle the lights patten is in. For example, use the traffic light pattern itself (RG, RY, etc) for **6** of the state bits. Use the other **3 bits** to **count** the time periods for each traffic light pattern and to distinguish between the two RR patterns. For example, the first few cycles of the RG pattern can be C0&RG, C1&RG, C2&RG, etc.

3. Simulate your VHDL code in ModelSim using Lab6P2.do. Save a picture of the waveform after you have checked the signals carefully and are sure it shows the correct behavior.

   Show your simulation waveform to the TA/instructor.

4. **Compile**, program, and **test** your circuit on the **traffic light station**.
5. **Save a copy of your VHDL code for this part in the folder, and name it Lab6P2.vhd. Do not modify this copy for the remaining parts of the lab.**

   Demonstrate to the TA/instructor.

6.3 Timed Traffic Light Controller with Farm Road Request Input.

The controller from part 2 spends longer (7 clocks) in the RG state, allowing traffic to pass on the highway, but it still interrupts this traffic flow and switches to the farm road even when there are no cars waiting on the farm side. To solve this problem, a bright young student suggests using a **sensor input** to detect the presence (or not) of a car on the farm road and re-designing the controller as follows:

a) If the sensor is **not** triggered (sensor = **0,** no car on the farm road), the lights should remain indefinitely in the **RG** state, allowing the highway traffic to flow uninterrupted.

b) When the sensor is **triggered** (sensor = **1**) by a waiting car on the farm road, it should do the following

- If the lights have been in the RG state for **7 or more** clocks, they will immediately start switching to allow the farm road vehicle to pass.
- If the lights have **not** been in the RG state for 7 clocks, they will finish the count of 7 and ***then*** switch to allow the farm road vehicle to pass.

1. To achieve the above function, modify your part2 code to:
   - Add a 1-bit input signal called “treadle” to the entity that acts as the sensor input.
   - Use **if** statements inside the **case** statement to check the status of the **sensor** input and act appropriately.
   - Open Lab6P2.do in Quartus, and add the following line after the clock line:
     ```
     force treadle 0 0 ns, 1 200 ns, 0 360 ns, 1 400 ns, 0 450 ns
     ```
2. Simulate your VHDL code in ModelSim. Save a picture of the waveform after you have checked the signals carefully and are sure it shows the correct behavior.

Show your simulation waveform to the TA/instructor.

3. **Compile**, program, and **test** your circuit on the **traffic light station**.

   Demonstrate to the TA/instructor.

4. **Save a copy of your VHDL code for this part in the folder, and name it Lab6P3.vhd. Do not modify this copy for the remaining parts of the lab.**

6.4 Timed Traffic Light Controller with More Realistic Farm Road Input.

The farmer is unhappy with the new design because there is sometimes a **long line** of cars on the farm road, and they need more time to get across the highway. The bright young student suggests modifying the design to handle the case when the sensor is active and **remains active**. This simulates a line of cars on the farm road. The controller should respond with the additional rule:

When the farm road has the green light (**GR**) for 2 counts then check the sensor:

- If the sensor is no longer active (i.e. only one or two cars), the sequence should **continue** as before.
- However, **if** the sensor is still active, wait an additional **5** counts and only then continue the sequence.

1. Modify your code from part 3 to add the above functionality.
2. Simulate your VHDL code in ModelSim. Save a picture of the waveform after you have checked the signals carefully and are sure it shows the correct behavior.

   Show your simulation waveform to the TA/instructor.

3. **Compile**, program, and **test** your circuit on the **traffic light station**.

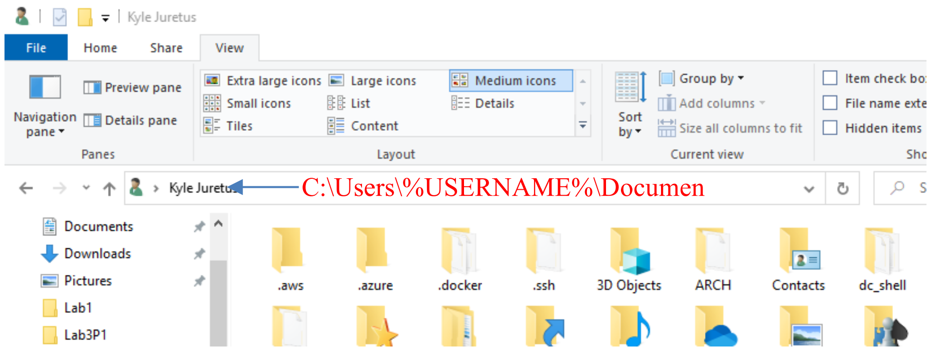
   Demonstrate to the TA/instructor.

4. **Save a copy of your VHDL code for this part in the folder, and name it Lab6P4.vhd.**

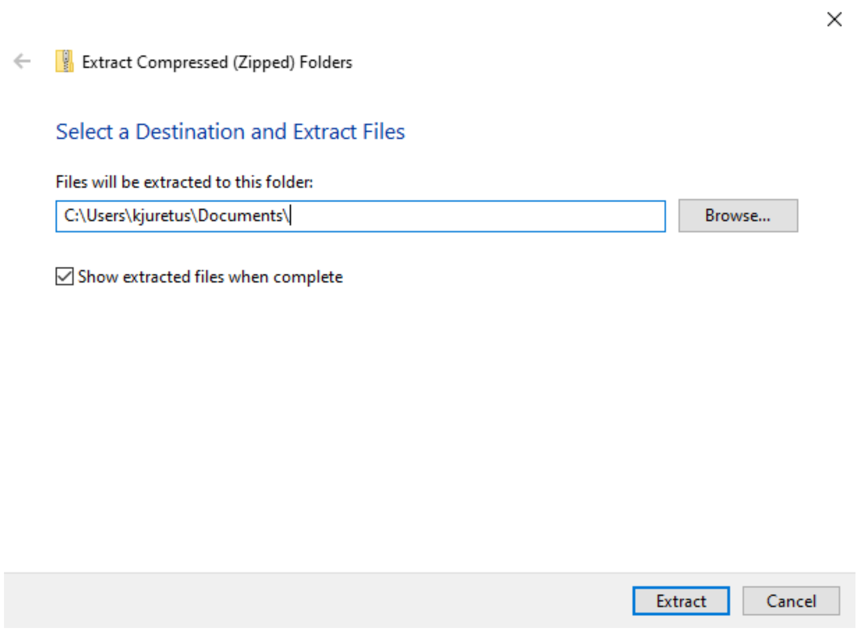
6.5 Synopsys Synthesis of the Timed Traffic Light Controller

The VHDL file you developed for the timed traffic light (Lab6P4.vhd) will now be put through the Synopsys EDA flow to show you what the generated circuit from the VHDL code looks like.

1. Open your Documents directory in Windows File Explorer
   a. Copy “C:\Users\%USERNAME%\Documents” and put the path in the Windows File Explorer Path and press enter. You should now be in your Documents directory in the Windows File Explorer window.



2. Unzip the DC_SHELL_OUT.zip file provided on Blackboard into your Documents directory and go into that directory. **Note: Do not unzip to DC_SHELL_OUT (i.e. a folder within a folder)**. A file named .synopsys_dc.setup should be within the folder, if extracted correctly.


Extract Compressed (Zipped) Folders

Select a Destination and Extract Files

Files will be extracted to this folder:

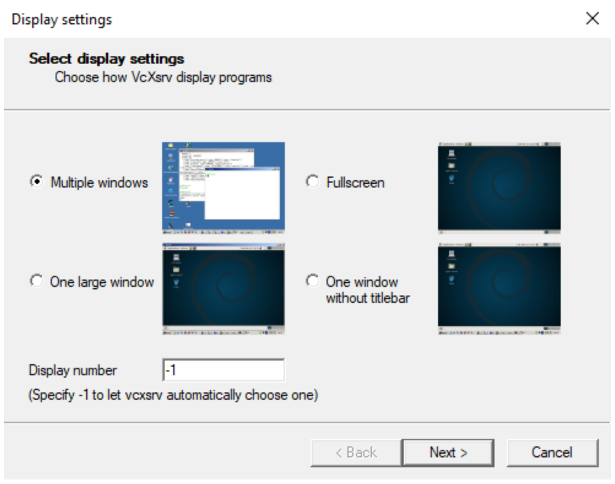
C:\Users\kjuretus\Documents\

☑ Show extracted files when complete

3. Copy your VHDL file from Lab 6 part 4 into the DC_SHELL_OUT directory (should be your current directory).

4. Start the **XLaunch** program on your desktop.
   a. Double click the XLaunch icon on your desktop.

   b. The following window should appear, click **Multiple Windows**, and then click **Next**

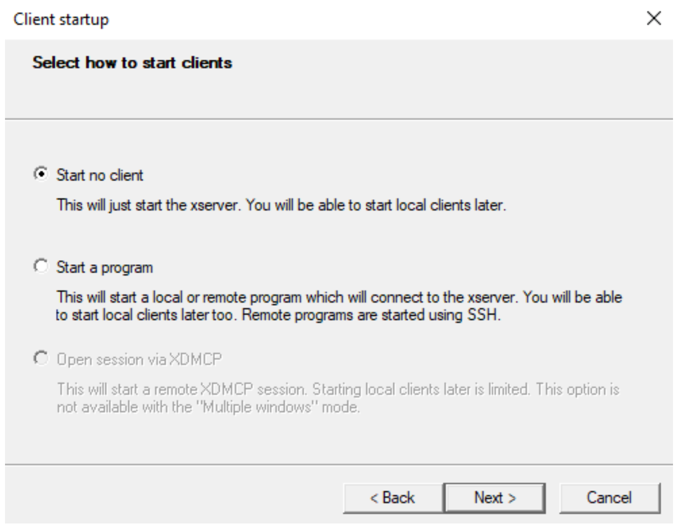

Display settings

Select display settings
Choose how VcXsrv display programs

(●) Multiple windows
( ) Fullscreen
( ) One large window
( ) One window without titlebar

Display number: -1
(Specify -1 to let vcxsrv automatically choose one)

c. Click **Start no client**, and then click **Next**



Client startup

Select how to start clients

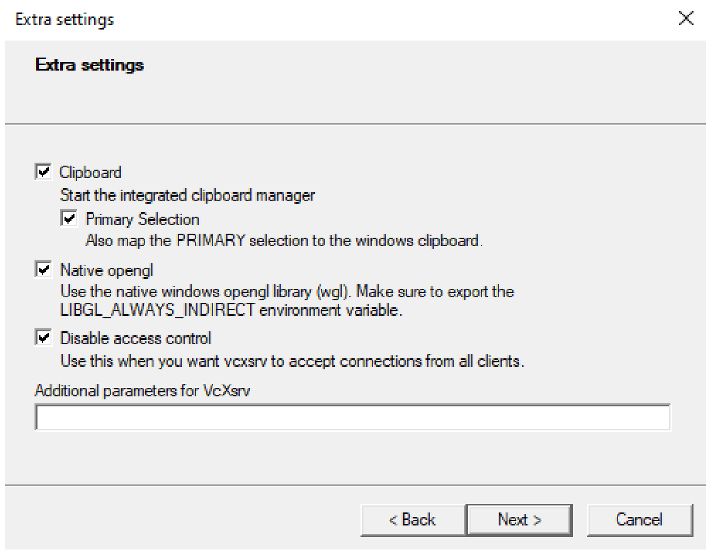
(●) Start no client
This will just start the xserver. You will be able to start local clients later.

( ) Start a program
This will start a local or remote program which will connect to the xserver. You will be able to start local clients later too. Remote programs are started using SSH.

( ) Open session via XDMCP
This will start a remote XDMCP session. Starting local clients later is limited. This option is not available with the "Multiple windows" mode.

< Back | Next > | Cancel

d. On the next window, make sure the following selections are checked, then click **Next**, then **Finish.**



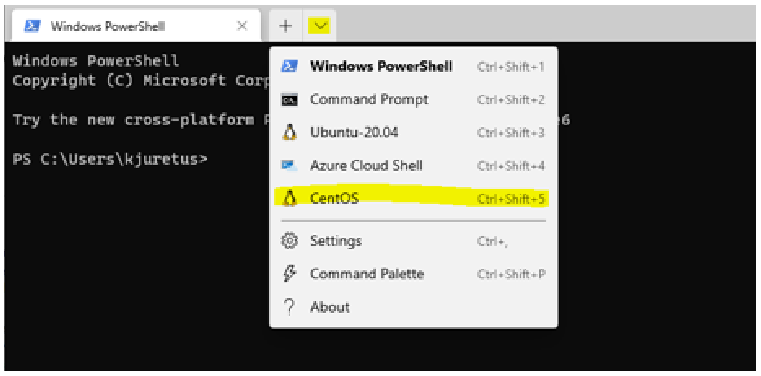
Extra settings

Extra settings

[✓] Clipboard
Start the integrated clipboard manager
[✓] Primary Selection
Also map the PRIMARY selection to the windows clipboard.
[✓] Native opengl
Use the native windows opengl library (wgl). Make sure to export the LIBGL_ALWAYS_INDIRECT environment variable.
[✓] Disable access control
Use this when you want vcxsrv to accept connections from all clients.
Additional parameters for VcXsrv

< Back | Next > | Cancel

5. Run the **WSLStartup** batch file on the desktop. This should open a Windows Terminal instance.
6. In **Windows Terminal**, click the **Down Arrow**, and then select **CentOS.**



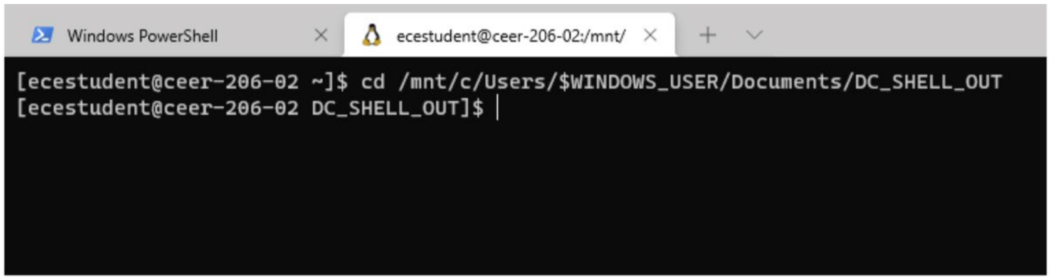
```
Windows PowerShell
Copyright (C) Microsoft Corp

Try the new cross-platform P

PS C:\Users\kjuretus>
```

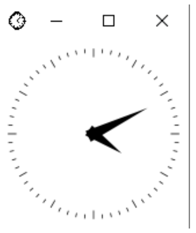
Windows PowerShell
Windows PowerShell Ctrl+Shift+1
Command Prompt Ctrl+Shift+2
Ubuntu-20.04 Ctrl+Shift+3
Azure Cloud Shell Ctrl+Shift+4
CentOS Ctrl+Shift+5
Settings Ctrl+,
Command Palette Ctrl+Shift+P
About

7. Change into the DC_SHELL_OUT directory by entering the following command into the terminal
   `cd /mnt/c/Users/$WINDOWS_USER/Documents/DC_SHELL_OUT`

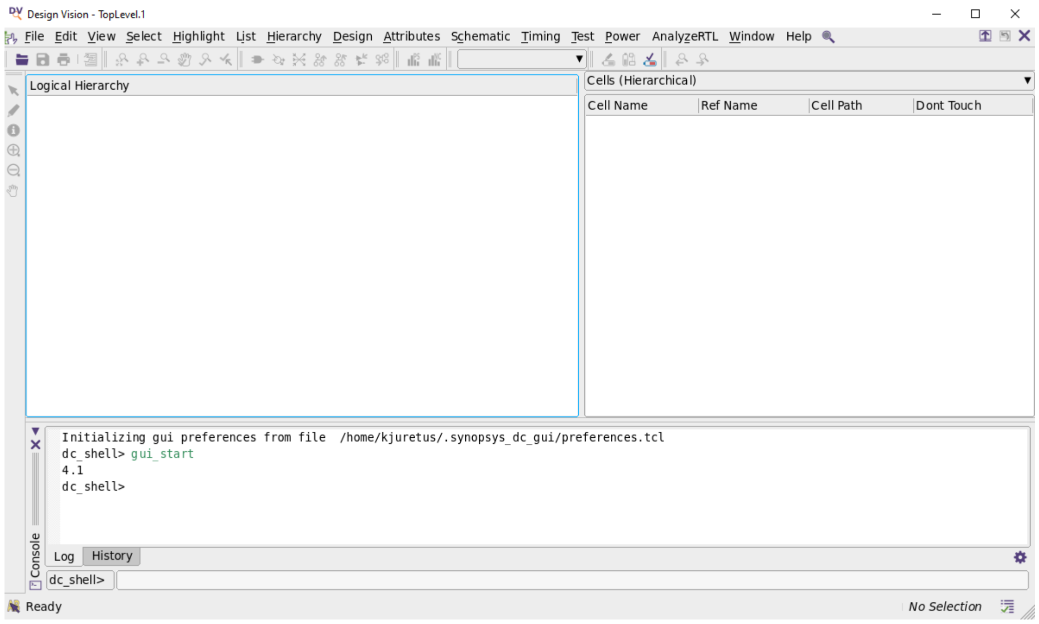


```
[ecestudent@ceer-206-02 ~]$ cd /mnt/c/Users/$WINDOWS_USER/Documents/DC_SHELL_OUT
[ecestudent@ceer-206-02 DC_SHELL_OUT]$
```

8. Test that the graphics are running by typing in **xclock** and pressing enter. If successful, you should see the following pop-up on your screen. (The clock may also be hidden behind other windows.) If successful, close xclock.



9. Type **dc_shell** in the terminal and press enter. The following program should appear on your screen.

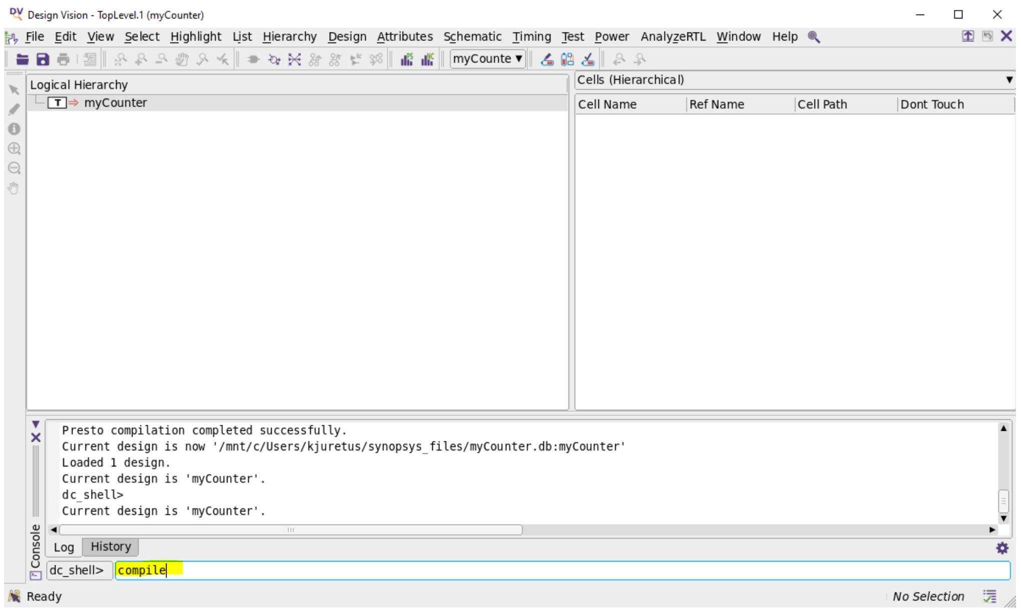

Design Vision - TopLevel.1
File Edit View Select Highlight List Hierarchy Design Attributes Schematic Timing Test Power AnalyzeRTL Window Help
Logical Hierarchy
Cells (Hierarchical)
Cell Name
Ref Name
Cell Path
Dont Touch
Console
Log
History
Ready
No Selection

```
Initializing gui preferences from file  /home/kjuretus/.synopsys_dc_gui/preferences.tcl
dc_shell> gui_start
4.1
dc_shell>
```

```
dc_shell>
```

10. Choose File->Read from the menu and then locate your VHDL file from Lab 6 part 4.

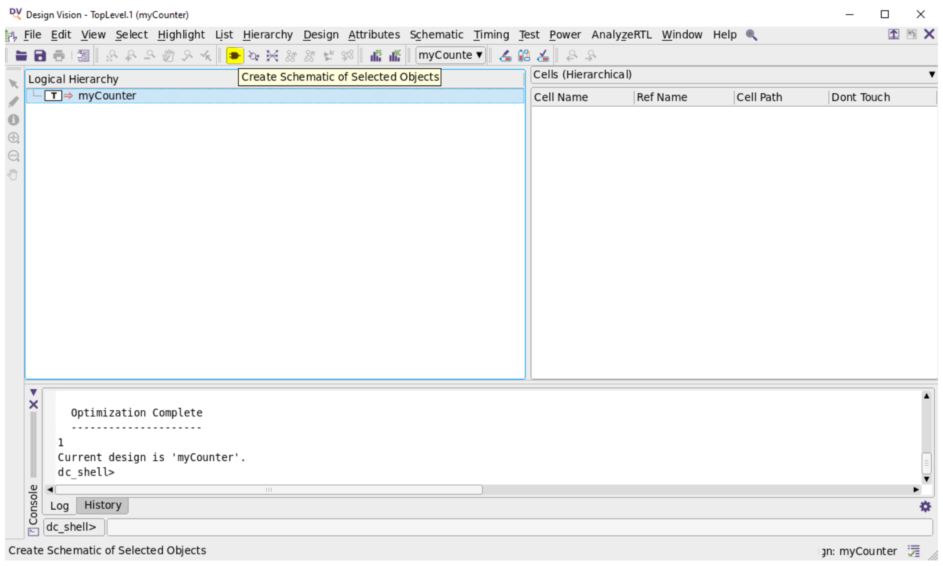
11. Enter **compile** into the dc_shell terminal and press enter.


Design Vision - TopLevel.1 (myCounter)
File Edit View Select Highlight List Hierarchy Design Attributes Schematic Timing Test Power AnalyzeRTL Window Help
myCounte
Logical Hierarchy
myCounter
Cells (Hierarchical)
Cell Name
Ref Name
Cell Path
Dont Touch
Console
Log
History
Ready
No Selection

```
Presto compilation completed successfully.
Current design is now '/mnt/c/Users/kjuretus/synopsys_files/myCounter.db:myCounter'
Loaded 1 design.
Current design is 'myCounter'.
dc_shell>
Current design is 'myCounter'.
```

```
dc_shell> compile
```

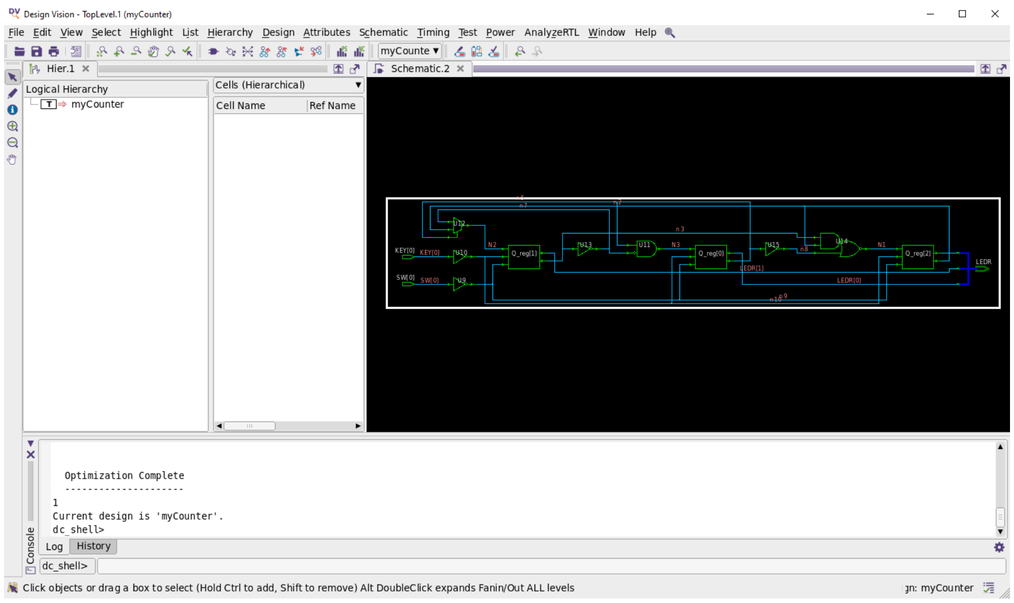
12. Click on your design in the **Logical Hierarchy** list (myCounter as shown in the example) and then click the “Create Schematic of Selected Objects” button (highlighted below).


Design Vision - TopLevel.1 (myCounter)
File Edit View Select Highlight List Hierarchy Design Attributes Schematic Timing Test Power AnalyzeRTL Window Help
myCounte
Logical Hierarchy
Create Schematic of Selected Objects
myCounter
Cells (Hierarchical)
Cell Name
Ref Name
Cell Path
Dont Touch
Console
Log
History
Create Schematic of Selected Objects
gn: myCounter

```
  Optimization Complete
  ---------------------
1
Current design is 'myCounter'.
dc_shell>
```

```
dc_shell>
```

13. A schematic view with the input/output names from your VHDL should be shown as a black box. Double click the schematic view to see the internal cells. Your screen should look similar to the following, but with a different circuit implementation. **Take a screenshot of your implementation and save it to upload with your lab files.**



```
  Optimization Complete
  ---------------------
1
Current design is 'myCounter'.
dc_shell>
```

dc_shell>

Click objects or drag a box to select (Hold Ctrl to add, Shift to remove) Alt DoubleClick expands Fanin/Out ALL levels

gn: myCounter

14. Examine the circuit implementation and determine how the schematic relates to the VHDL implementation. Are there any surprising cells in the implementation? Any differences to how you would implement it via logic?

Show the schematic to your instructor/TA

Submission for Lab 6:

After you have been checked off all the parts by the instructor/TA, make sure the saved VHDL programs and the waveforms are in the folder. The zip the Lab6 project folder and name it Lab6_<your last name>.zip, and submit it in the Blackboard. Please submit only one zip file, and ensure everything is contained in the single zip file.