Saturday **EE 671: VLSI DESIGN 2024/2025** **Due On**

Sep 29, 2024 **ASSIGNMENT-3** **Oct 6, 2024**

Note:

1. This assignment will be done in groups with one submission per group.
2. Submission is **only** through Moodle in the form of a PDF file upload.
3. All plots must be legible/readable in the submission (axes values and units).

Introduction:

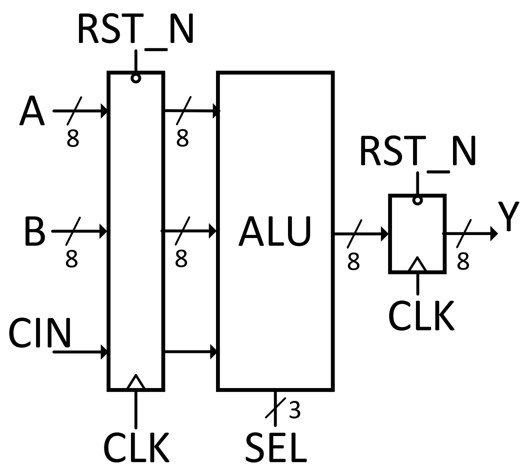
In this assignment you will perform synthesis of a design to obtain the RTL logic. This will be done through OpenLane based open-source IC design flow. The assignment is broken into two stages: (a) perform synthesis of an existing design and (b) design, verify and synthesis of a new logic. This assignment also provides some encouragement to use a unix based system and a few handy commands.

Instructions to login to the VLSI lab machine:

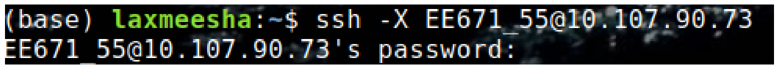
- Each group is assigned a group id for login. Refer to your login id from here. The default password is EE671
- If you are on any unix based system, open a terminal for subsequent steps. If you are on Windows, download MobaXterm and open a terminal via MobaXterm. Ensure you are connected to IITB network.
- To login, type on the terminal: *ssh -X EE671_X@10.107.90.73* (X corresponds to the number from your login id assigned to your group). Enter the password and you are good.
- After the first login, type on the terminal: *passwd* and change your password immediately.

Running synthesis on an existing example:

We consider an example ALU logic, as shown below. It has inputs A, B, CIN, SEL CLK, RST_N (active low reset) and output Y. Follow the sequence of steps below:

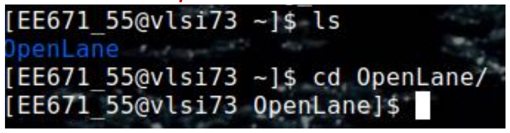


1. Login to your account



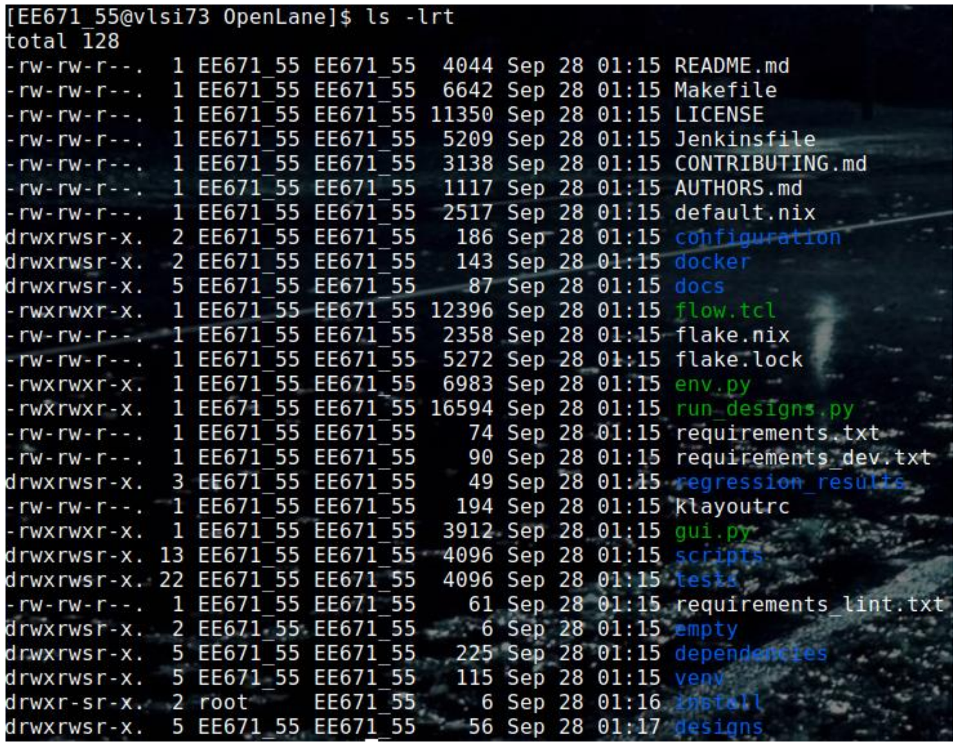
```
(base) laxmeesha:~$ ssh -X EE671_55@10.107.90.73
EE671_55@10.107.90.73's password:
```

2. Type *ls* to list the files. You should see a folder named OpenLane. Go into the directory by typing the command *cd OpenLane*



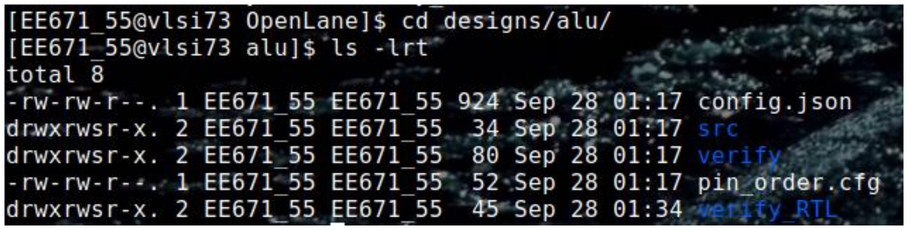
```
[EE671_55@vlsi73 ~]$ ls
OpenLane
[EE671_55@vlsi73 ~]$ cd OpenLane/
[EE671_55@vlsi73 OpenLane]$
```

3. Type *ls -lrt* to list all files with the time index (latest edited file will be on the bottom of the list). The important folder for this assignment is the "designs" folder. You are free to look inside into other files and folders. However, **DO NOT** delete/edit any files in the setup.



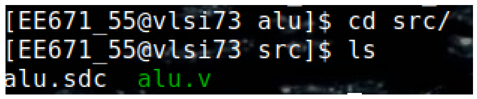
```
[EE671_55@vlsi73 OpenLane]$ ls -lrt
total 128
-rw-rw-r--.  1 EE671_55 EE671_55  4044 Sep 28 01:15 README.md
-rw-rw-r--.  1 EE671_55 EE671_55  6642 Sep 28 01:15 Makefile
-rw-rw-r--.  1 EE671_55 EE671_55 11350 Sep 28 01:15 LICENSE
-rw-rw-r--.  1 EE671_55 EE671_55  5209 Sep 28 01:15 Jenkinsfile
-rw-rw-r--.  1 EE671_55 EE671_55  3138 Sep 28 01:15 CONTRIBUTING.md
-rw-rw-r--.  1 EE671_55 EE671_55  1117 Sep 28 01:15 AUTHORS.md
-rw-rw-r--.  1 EE671_55 EE671_55  2517 Sep 28 01:15 default.nix
drwxrwsr-x.  2 EE671_55 EE671_55   186 Sep 28 01:15 configuration
drwxrwsr-x.  2 EE671_55 EE671_55   143 Sep 28 01:15 docker
drwxrwsr-x.  5 EE671_55 EE671_55    87 Sep 28 01:15 docs
-rwxrwxr-x.  1 EE671_55 EE671_55 12396 Sep 28 01:15 flow.tcl
-rw-rw-r--.  1 EE671_55 EE671_55  2358 Sep 28 01:15 flake.nix
-rw-rw-r--.  1 EE671_55 EE671_55  5272 Sep 28 01:15 flake.lock
-rwxrwxr-x.  1 EE671_55 EE671_55  6983 Sep 28 01:15 env.py
-rwxrwxr-x.  1 EE671_55 EE671_55 16594 Sep 28 01:15 run_designs.py
-rw-rw-r--.  1 EE671_55 EE671_55    74 Sep 28 01:15 requirements.txt
-rw-rw-r--.  1 EE671_55 EE671_55    90 Sep 28 01:15 requirements_dev.txt
drwxrwsr-x.  3 EE671_55 EE671_55    49 Sep 28 01:15 regression_results
-rw-rw-r--.  1 EE671_55 EE671_55   194 Sep 28 01:15 klayoutrc
-rwxrwxr-x.  1 EE671_55 EE671_55  3912 Sep 28 01:15 gui.py
drwxrwsr-x. 13 EE671_55 EE671_55  4096 Sep 28 01:15 scripts
drwxrwsr-x. 22 EE671_55 EE671_55  4096 Sep 28 01:15 tests
-rw-rw-r--.  1 EE671_55 EE671_55    61 Sep 28 01:15 requirements_lint.txt
drwxrwsr-x.  2 EE671_55 EE671_55     6 Sep 28 01:15 empty
drwxrwsr-x.  5 EE671_55 EE671_55   225 Sep 28 01:15 dependencies
drwxrwsr-x.  5 EE671_55 EE671_55   115 Sep 28 01:15 venv
drwxr-sr-x.  2 root     EE671_55     6 Sep 28 01:16 install
drwxrwsr-x.  5 EE671_55 EE671_55    56 Sep 28 01:17 designs
```

4. The Verilog code of the ALU is kept inside the designs folder. Get into the designs folder and inside that get into the alu folder and list all files. You will see three folders (a) src (b) verify (c) verify_RTL and a config.json file. The src file is where all original Verilog files are to be placed. The verify folder is used to verify the design using a test bench and verify_RTL folder is used to perform verification of the RTL.



```
[EE671_55@vlsi73 OpenLane]$ cd designs/alu/
[EE671_55@vlsi73 alu]$ ls -lrt
total 8
-rw-rw-r--. 1 EE671_55 EE671_55 924 Sep 28 01:17 config.json
drwxrwsr-x. 2 EE671_55 EE671_55  34 Sep 28 01:17 src
drwxrwsr-x. 2 EE671_55 EE671_55  80 Sep 28 01:17 verify
-rw-rw-r--. 1 EE671_55 EE671_55  52 Sep 28 01:17 pin_order.cfg
drwxrwsr-x. 2 EE671_55 EE671_55  45 Sep 28 01:34 verify_RTL
```

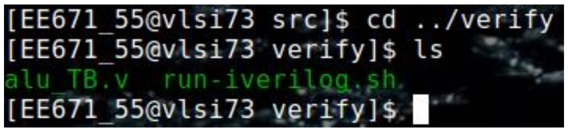
5. Get into the src folder and you should see two files alu.v and alu.sdc. We will discuss about the alu.sdc in the next assignment.



```
[EE671_55@vlsi73 alu]$ cd src/
[EE671_55@vlsi73 src]$ ls
alu.sdc  alu.v
```

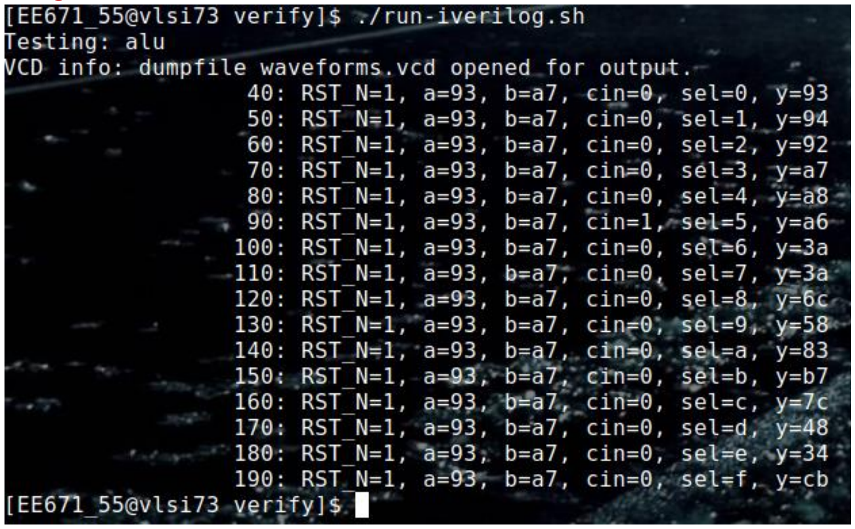
6. Open the alu.v Verilog code using any test editor and check that the design is exactly same as the figure shown above. You can use nedit or gedit text editors. For example, to open using nedit, you can type on the terminal *nedit alu.v &*

7. Now, go to the verify folder to perform a functional simulation. If you list the files under the verify folder, you will see a testbench *alu_TB.v* and a shell script to run iverilog *run-iverilog.sh*



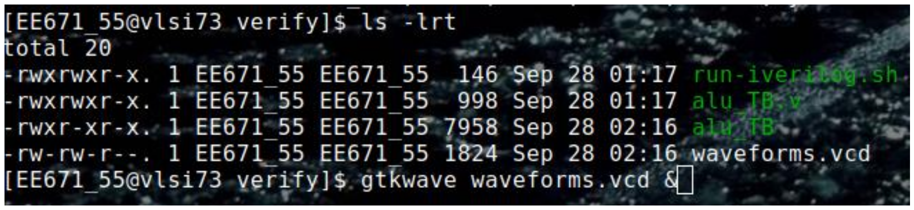
```
[EE671_55@vlsi73 src]$ cd ../verify
[EE671_55@vlsi73 verify]$ ls
alu_TB.v  run-iverilog.sh
[EE671_55@vlsi73 verify]$ 
```

8. Open both the files using nedit/gedit and check the file contents. To run iverilog, simply type *./run-iverilog.sh* on the terminal.



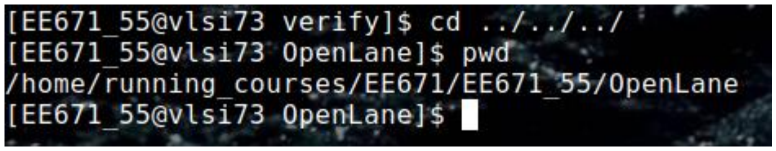
```
[EE671_55@vlsi73 verify]$ ./run-iverilog.sh
Testing: alu
VCD info: dumpfile waveforms.vcd opened for output.
                  40: RST_N=1, a=93, b=a7, cin=0, sel=0, y=93
                  50: RST_N=1, a=93, b=a7, cin=0, sel=1, y=94
                  60: RST_N=1, a=93, b=a7, cin=0, sel=2, y=92
                  70: RST_N=1, a=93, b=a7, cin=0, sel=3, y=a7
                  80: RST_N=1, a=93, b=a7, cin=0, sel=4, y=a8
                  90: RST_N=1, a=93, b=a7, cin=1, sel=5, y=a6
                 100: RST_N=1, a=93, b=a7, cin=0, sel=6, y=3a
                 110: RST_N=1, a=93, b=a7, cin=0, sel=7, y=3a
                 120: RST_N=1, a=93, b=a7, cin=0, sel=8, y=6c
                 130: RST_N=1, a=93, b=a7, cin=0, sel=9, y=58
                 140: RST_N=1, a=93, b=a7, cin=0, sel=a, y=83
                 150: RST_N=1, a=93, b=a7, cin=0, sel=b, y=b7
                 160: RST_N=1, a=93, b=a7, cin=0, sel=c, y=7c
                 170: RST_N=1, a=93, b=a7, cin=0, sel=d, y=48
                 180: RST_N=1, a=93, b=a7, cin=0, sel=e, y=34
                 190: RST_N=1, a=93, b=a7, cin=0, sel=f, y=cb
[EE671_55@vlsi73 verify]$ 
```

9. If you now list the folder contents using *ls* command, you will see a waveforms.vcd file created. We can open this file on the terminal by typing: *gtkwave waveforms.vcd &*



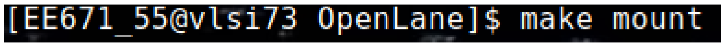
```
[EE671_55@vlsi73 verify]$ ls -lrt
total 20
-rwxrwxr-x. 1 EE671_55 EE671_55  146 Sep 28 01:17 run-iverilog.sh
-rwxrwxr-x. 1 EE671_55 EE671_55  998 Sep 28 01:17 alu_TB.v
-rwxr-xr-x. 1 EE671_55 EE671_55 7958 Sep 28 02:16 alu_TB
-rw-rw-r--. 1 EE671_55 EE671_55 1824 Sep 28 02:16 waveforms.vcd
[EE671_55@vlsi73 verify]$ gtkwave waveforms.vcd &
```

10. Check the waveforms and ensure that all the inputs and ouputs are correct.
11. Now we have a verified Verilog design which is ready for synthesis. To perform synthesis, we need to go to the OpenLane directory. cd to the OpenLane directory on the terminal. Use *pwd* command to check the "present working directory"



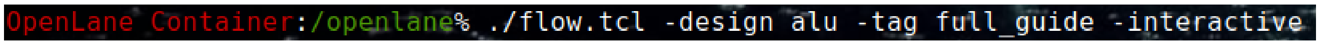
```
[EE671_55@vlsi73 verify]$ cd ../../../
[EE671_55@vlsi73 OpenLane]$ pwd
/home/running_courses/EE671/EE671_55/OpenLane
[EE671_55@vlsi73 OpenLane]$ 
```

11. We use a docker based environment to run OpenLane. To enter the docker container, type *make mount* on the terminal



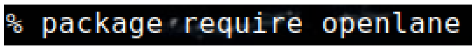
```
[EE671_55@vlsi73 OpenLane]$ make mount
```

12. You are now inside the docker container. From this container, we will run all the tcl scripts. To call the OpenLane flow, we need to run the flow.tcl script with some arguments. On the container terminal, type *./flow.tcl -design alu -tag full_guide -interactive* . The design alu points to the directory "alu" inside the designs folder that you earlier looked at. We ensured that the directory name (**alu** in this case) is same as the top cell design name and Verilog name (module **alu** in file **alu**.v).



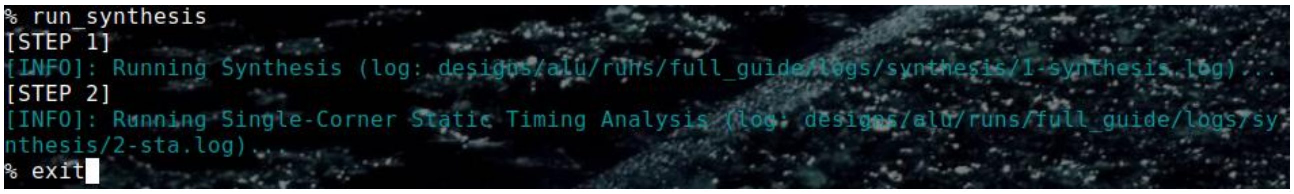
```
OpenLane Container:/openlane% ./flow.tcl -design alu -tag full_guide -interactive
```

13. Next type the following command to import OpenLane tools: *package require openlane*



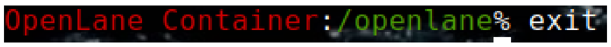
```
% package require openlane
```

14. To run synthesis, type the following: *run_synthesis* . This completes the synthesis and now exit the openlane flow by typing *exit.*



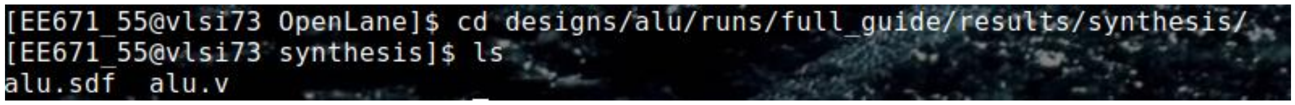
```
% run_synthesis
[STEP 1]
[INFO]: Running Synthesis (log: designs/alu/runs/full_guide/logs/synthesis/1-synthesis.log)...
[STEP 2]
[INFO]: Running Single-Corner Static Timing Analysis (log: designs/alu/runs/full_guide/logs/sy
nthesis/2-sta.log)...
% exit
```

15. Finally, exit the container by typing *exit*. You are now back on the original terminal.



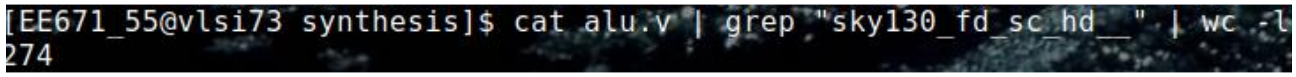
```
OpenLane Container:/openlane% exit
```

16. The RTL from synthesis is generated inside the designs folder. Cd to the *designs/alu/runs/full_guide/results/synthesis/* folder and list the files. You will see an alu.v RTL file.



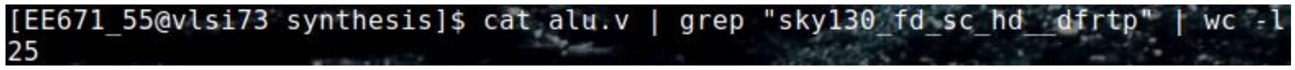
```
[EE671_55@vlsi73 OpenLane]$ cd designs/alu/runs/full_guide/results/synthesis/
[EE671_55@vlsi73 synthesis]$ ls
alu.sdf  alu.v
```

17. Open the RTL file and see that this has only standard cells. You can use nedit or gedit to open the file.
18. You can make use of unix commands to check how many standard cells are used in this RTL. You can use the cat command to read the alu.v file and pipe the output to grep command which will search for patterns. The output of grep can be piped again to a word-count/line-count command.



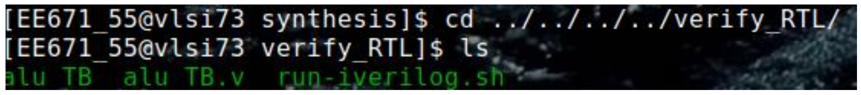
```
[EE671_55@vlsi73 synthesis]$ cat alu.v | grep "sky130_fd_sc_hd__" | wc -l
274
```

This clearly says a total of 274 cells from library "sky130_fd_sc_hd" are used in alu.v RTL. Now, if I want to know how many DFFs are used in the alu.v, only the grep pattern needs to be updated to "sky130_fd_sc_hd__dfrtp" (dfrtp is DFF with positive-edge and reset) – you will see a total of 25 DFFs.



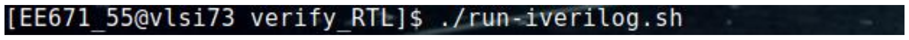
```
[EE671_55@vlsi73 synthesis]$ cat alu.v | grep "sky130_fd_sc_hd__dfrtp" | wc -l
25
```

18. Finally, we need to run the same testbench based verification on the RTL. Cd to the verify_RTL folder and you will see the alu_TB.v (this is the same TB copied from the verify folder) and a run-iverilog.sh script to run iverilog (this script is slightly modified compared to the one in the verify folder).



```
[EE671_55@vlsi73 synthesis]$ cd ../../../../verify_RTL/
[EE671_55@vlsi73 verify_RTL]$ ls
alu_TB  alu_TB.v  run-iverilog.sh
```

19. Run the script to simulate and generate the waveform file. Open using gtkwave and verify the results.



```
[EE671_55@vlsi73 verify_RTL]$ ./run-iverilog.sh
```

Q) In this assignment you will design a 16-bit Brent-Kung tree adder. The inputs A, B (16 bits each), CIN are clocked into a register using CLK signal. The registered signals are given as input to the adder and the output is again registered using a clock (both SUM and COUT are registered). Therefore, your module will have A, B, CIN, CLK, RST_N as inputs and SUM, COUT as outputs. Follow the steps below:

1. In the designs folder copy the alu folder into a new folder called **bkadder** (use the command *cp -r alu bkadder*).
2. Inside the bkadder folder, go to the src folder and rename alu.v to bkadder.v and alu.sdc to bkadder.sdc (use the command *mv alu.v bkadder.v*). Open the bkadder.v file and change the module name from alu to bkadder and put in the logic for Brent-Kung adder.
3. Once you have the code ready, go to the verify folder. Rename alu_TB.v to bkadder_TB.v. Open the testbench file using nedit and modify the TB. Please ensure you have multiple input combinations of A, B in the TB.
4. Open the run-iverilog .sh file and change the design parameter to bkadder.
5. Run the iverilog script and open the waveform using gtkwave. This completes our design verification.
6. Now go to the bkadder folder and open the config.json file using nedit. In this file, search all "alu" and replace them with "bkadder".
7. Run the synthesis using steps mentioned above (point to the new design while running flow.tcl).
8. Once the synthesis is successful, go to the verify_RTL folder. Copy your earlier testbench from the verify folder here. Open the run-iverilog.sh file and replace alu with bkadder.
9. Run iverilog and observe the waveforms.

Your report should have the following:

1. Verilog code (with proper comments, proper indentation) pasted from the original file. You will be penalized if your indentations and comments are not maintained.
2. The Testbench file (with proper comments, proper indentation) pasted from the original file.
3. Screenshot of the gtkwave waveforms (pre-synthesis).
4. Screenshot of the run_synthesis command and the associated dump on the terminal.
5. RTL file generated from synthesis.
6. Screenshot of the gtkwave waveforms (post-synthesis).
7. The following table generated from the RTL file:

Number of combinational cells	
Number of Sequential cells (Flip flops)	
Total number of cells	