ECE 551 Homework #4

Due: Nov. 9th @ 11:00 am

This homework assignment is to be completed individually.

Please Note: To receive full credit, you should use the following best practices in your homework assignments**:**

- Follow specific coding guidelines given in lecture (no #0 delays, no mixed blocking/non-blocking, etc.).
- When designing a non-testbench module, your code should be synthesizable.
- Use meaningful names for modules, ports, and wires when possible.
- Use underscores to break up long binary and hex numbers for readability.
- Show your work if you wish to receive partial credit.
- Make port connections *by name* unless using gate primitives.
- Explicitly declare any nets you use.
- All figures & graphs should include a number and caption (handwritten is OK).
- When printing waveforms, ensure that the relevant signal transitions are clearly visible, and label them if necessary.
- *All code should be typed*. Please do not submit handwritten code.
- **You must always turn in a printout of all Verilog code used with each problem unless explicitly told not to do so.**

[1] Improvising with Operators - *(12pts)*

Realize the following Mealy machine in Verilog. Use parameters to define the state encoding with the state names shown in the diagram. Name the input '**A'** and the output '**Y'**. Make the reset signal *asynchronous active-low.*

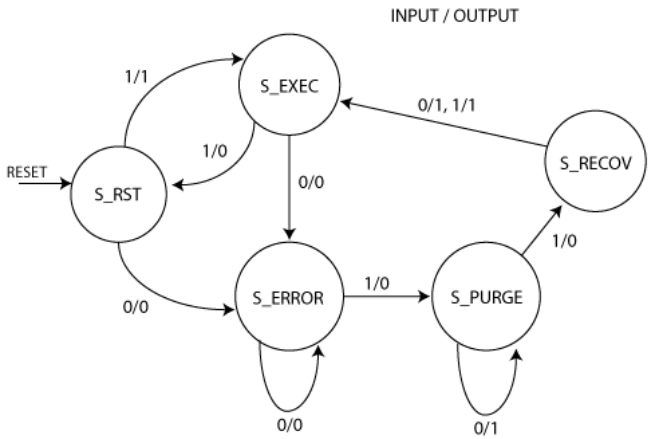

Fig. 1: Mealy Machine for Problem 1

[2] Adder Assembly Line - *(24pts)*

(a) *(8pts)* Solve Problem 8 from Exam 1 - Version A using a **Generate** block and **For** loop to instantiate the 2-bit adder modules.

(b) *(6pts)* Modify your solution from part (a) to add a parameter called BITWIDTH with a default value of 64. The parameterized version should allow your Generate block to create an adder of size BITWIDTH. You may assume that BITWIDTH will be a multiple of 2 and will be >= 4.

(c) *(10pts)* Copy the 120-bit adder from the solution of Problem 8 in Exam 1 – Version B. Write a self-checking testbench that instantiates your module from part (b) and overrides the value of BITWIDTH to 120, and compares the output of your adder with the one from the exam solution, producing an error signal if they do not match. **Submit a printout of the wave window along with your testbench.**

[3] The Road to Synthesis - *(42pts)*

This is a complex, multi-part question, so please read each part of the question carefully to make sure you don't miss any settings or requirements.

(a) *(12pts)* Synthesize your prime detector modules from HW2 and HW3. If you need to correct any design errors to allow your modules to synthesize, do so. If either of your modules did not work properly, you may copy the version from the solution instead. **If you make any changes to the code, submit a copy of the updated version.** Use the same wire load, driving cell, and operating conditions you used in the Design Constraints section of the Design Vision tutorial. Set input delay to 0 and use a 400 MHz clock. Set the max area to **0**. (This will tell the tool to continue trying to make the area smaller until it gives up).

Print the area report for each module. **Comment on the difference (if any) between the area results. Why do you think you do (or do not) see a difference?**

(b) *(10pts)* **Read the Tutorial on Post-Synthesis Validation in ModelSim on the course website.**

After you compile using the steps in part (a) save the post-synthesis netlist for both the HW2 and HW3 versions. Simulate the two post-synthesis netlists using your self-checking testbench from HW3 (Making modifications to the testbench as necessary). Note: you may need to adjust the clock frequency and timescale you are using in your testbench. **Print a copy of the wave output and submit the updated testbench.**

Do you see the same results you saw in HW3? If not, explain why.

(c) *(8pts)* Now modify your testbench to test the *pre-synthesis* Behavioral version from HW3 against the *post-synthesis* netlist you got from synthesizing the HW3 Behavioral module. You should notice that the error signal goes high for a time after each clock edge, then goes low again (if you don't see this you may need to zoom in further on the wave window or adjust your clock period). **Print a copy of the wave window showing this happening.**

(d) *(12pts)* The error seen in part (c) is caused because your pre-synthesis functional model has no delays in its logic, but the synthesized version does. The delay values in the standard cells will cause a delay post-synthesis circuit's output after each clock edge.

This is normal behavior, so we don't want it to cause the error signal in our self-checking testbench to go high. To avoid this, we will alter the error-checking part of the testbench to sample the output value of the modules after a delay.

Add a parameter to your testbench called SAMPLING_DELAY. You may have to experiment to find the right value for SAMPLING DELAY. Remove the continuous assignment that did the error checking in the old testbench. **Replace it with Verilog code that waits #SAMPLING_DELAY after each rising clock edge**, then assigns

```
error = (pre_synth_output != post_synth_output);
```

Submit your updated testbench and print a wave window showing the fixed error signal.

Remember the techniques used in this problem; you will need to compare the output of pre-synthesis and post-synthesis modules when doing the design project. Creating a self-checking testbench like this will make your job a lot easier.

List of things you must submit for this problem:

- Part A - Altered versions of HW2 and HW3 modules (if you made any changes to allow them to synthesize).
- Part A – Area reports for both HW2 and HW3 module.
- Part A – Your comments on the area results
- Part B – Updated testbench
- Part B – Wave output
- Part B – Comments on differences (if any)
- Part C – Wave output showing delay in the synthesized module
- Part D – Updated Testbench
- Part D – Wave output

Make sure each of these is clearly labeled in your homework submission.