**ECE 551 Homework #4**

**Due: Nov. 9th @ 11:00 am**

**[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.*

**Attached as Figure A: p1\_mealy.v**

**Attached as Figure B: p1\_mealy\_tb.v**

**Attached as Figure C: p1\_mealy\_tb Wave**

**[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.

**Attached as Figure D: fulladd\_2bit.v**

**Attached as Figure E: p2ab\_adder.v**

(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.

**Attached as Figure F: p2ab\_adder.v**

(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.**

**Attached as Figure G: fulladd\_120bit.v**

**Attached as Figure H: p2\_tb.v**

**Attached as Figure I: p2\_tb Wave**

**[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?**

**Attached as Figure J: prime\_detector\_structural.v**

**Attached as Figure K: prime\_detector\_behavioral.v**

**Attached as Figure L: dff\_async\_rst.v**

**Attached as Figure M: area\_report\_prime\_detector\_structural.txt**

**Attached as Figure N: area\_report\_prime\_detector\_behavioral.txt**

**The area of the structural is slightly smaller. I believe this is due to the very explicit way in which the structural design was done. This keeps the area quite low. They are pretty close however.**

(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.**

**Attached as Figure O: prime\_detector\_structural\_netlist.v**

**Attached as Figure P: prime\_detector\_behavioral\_netlist.v**

**Attached as Figure Q: prime\_detector\_behavioral\_structural\_netlist\_tb.v**

**Attached as Figure R: prime\_detector\_behavioral\_structural\_netlist\_tb Wave**

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

**The result that is seen is the same, except some minor glitches in the error signal at transitions. This is fixed in part d.**

(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.**

**Attached as Figure S: prime\_detector\_behavioral\_tb.v**

**Attached as Figure T: prime\_detector\_behavioral\_tb Wave**

(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.**

**Attached as Figure U: prime\_detector\_behavioral\_noglitch\_tb.v**

**Attached as Figure V: prime\_detector\_behavioral\_noglitch\_tb Wave**