Justin Liang

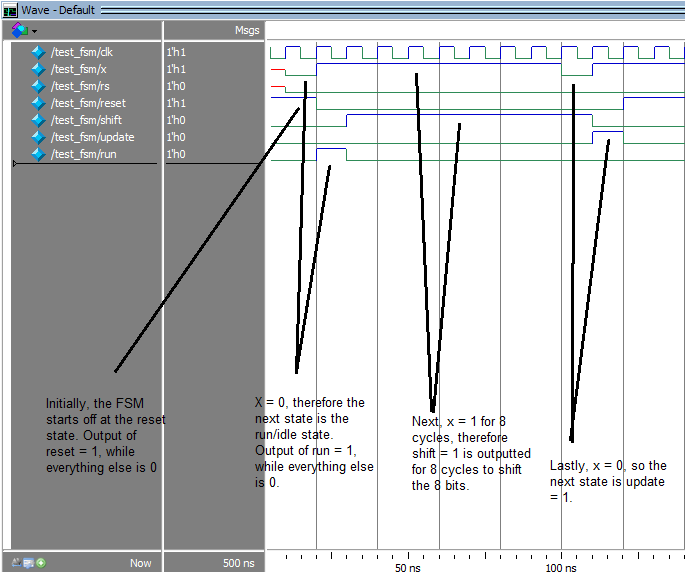
ECE 156A

Due: 12/6/13

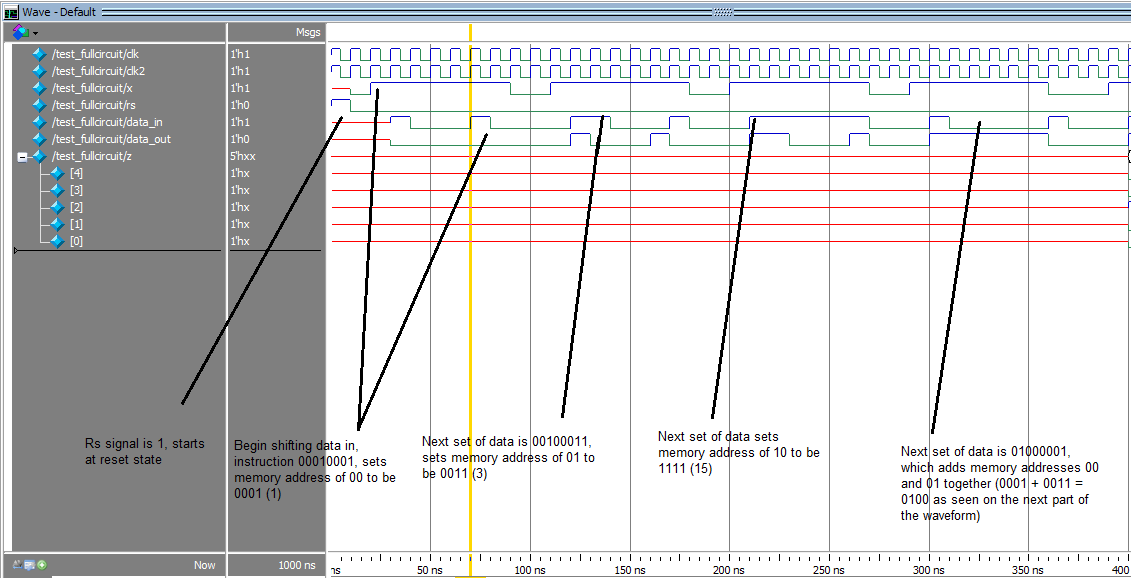
**Homework 5**

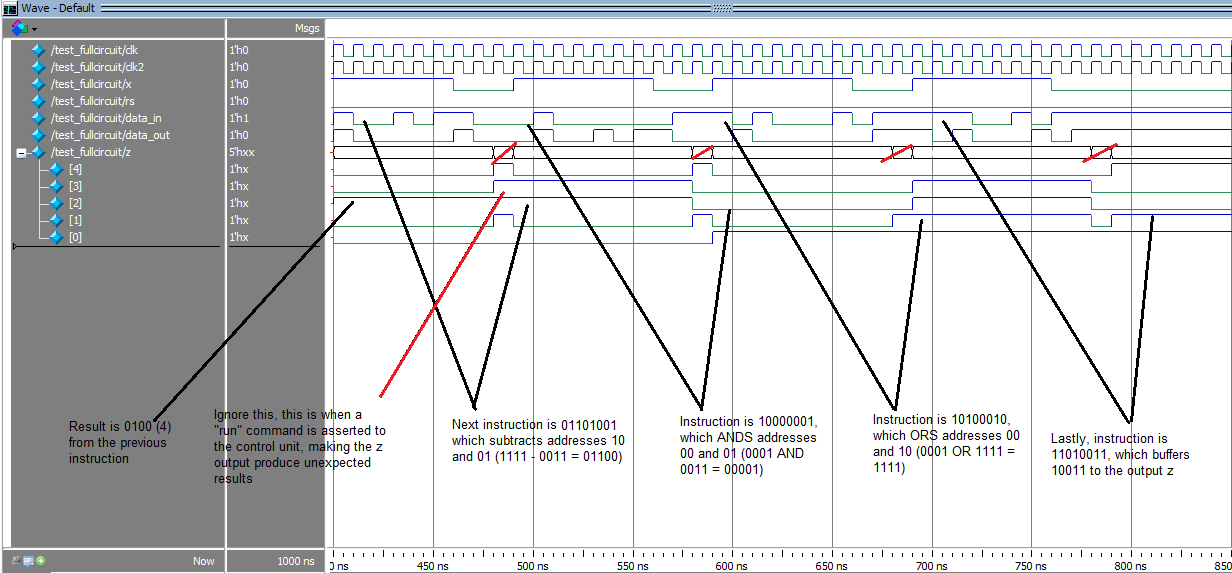
**Part 1: Design of a Simple Processor**

In this first part, we are asked to design a finite state machine and a control unit for a simple processor. To verify functional correctness, I first designed and tested the FSM individually, then designed and tested the control unit individually, and finally put the FSM and control unit together and tested the full circuit. By verifying each part individually for correct functionality, I am able to find bugs in my code more effectively and more quickly than verifying the full circuit. For the finite state machine, I simply implemented the Verilog code to model the state diagram using a series of case statements. Below is the following waveform for the FSM.



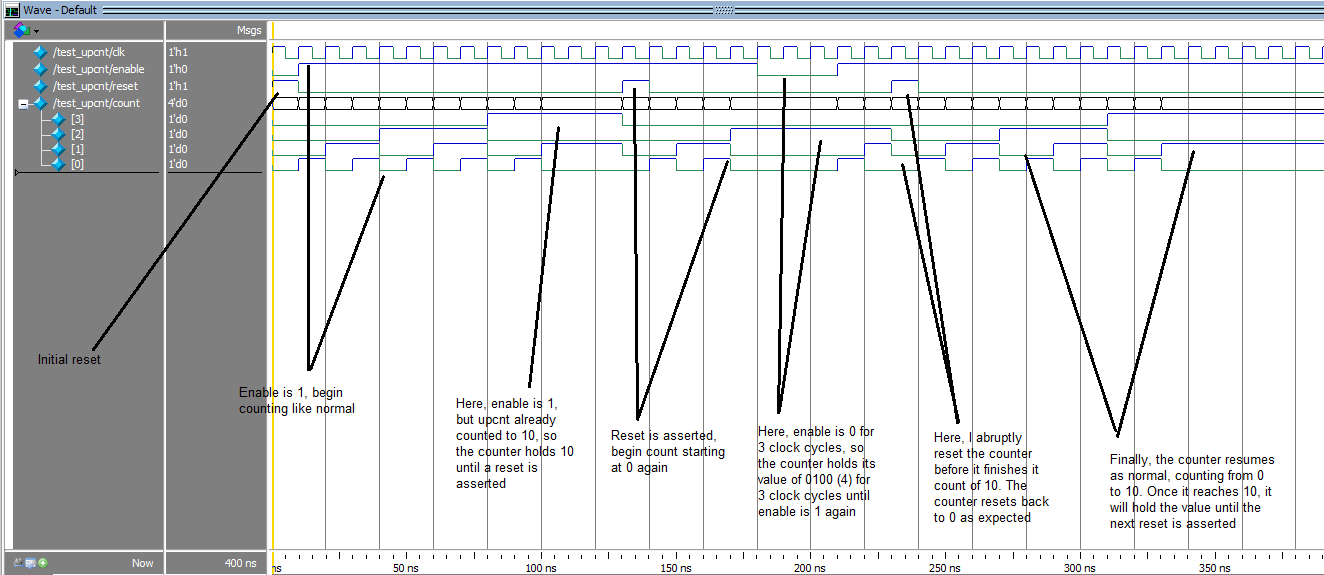
Next, the control unit is implemented in Verilog using simple if/else if statements. The control unit’s memory, shadow register, and shift register are all initialized to 0. When the FSM output is reset, the control unit will set all memory, shadow register, and shift register to 0. When the FSM output is run, the control unit will run the current operation given by what’s stored in the shadow register. The current operation and how it’s handled is determined by if/else if statements as shown in my Verilog code. When the FSM output is shift, the control unit will shift each bit to the right (from MSB to LSB). This is done by assigning data\_in to shift\_reg[7], shift\_reg[7] to shift\_reg[6], shift\_reg[6] to shift\_reg[5] and so on. Data\_out is the bit that is being discarded which is shift\_reg[0]. When the FSM output is update, the shift register’s contents will latch onto the shadow register for stability. In order to test the control unit, I had to make sure the FSM’s functionality was working correctly by itself. I connected the FSM to the control unit using the module fullcircuit. Below are the following waveforms split into two parts (the second waveform is a continuation of the first waveform).





**Part 2: Design Compiler**

Here, we are asked to design a counter that counts from 0 to 10 without looping back to 0 when it reaches 10. This was designed in Verilog using simple case statements with a behavioral implementation. Below is the following waveform.



After creating the behavioral Verilog of the up-counter, I used design compiler to synthesize the code to a gate-level implementation. First, I set up dc\_shell as instructed. Next, I placed .synopsys\_DC.setup into the same folder as upcnt.v. In the dc\_shell, ran the following commands:

1. read\_verilog upcnt.v
2. compile
3. write -hierarchy -format verilog -output upcnt\_structural.v

This produces the gate-level implementation in the file upcnt\_structural. Below is the following area report using the command “report\_area.”

dc\_shell> report\_area

Information: Updating graph... (UID-83)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Report : area

Design : upcnt

Version: D-2010.03-SP1

Date : Sun Dec 8 17:39:11 2013

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Information: Updating design information... (UID-85)

Library(s) Used:

class (File: /ece/synopsys/synthesis/D-2010.03-SP1/libraries/syn/class.db)

Number of ports: 7

Number of nets: 24

Number of cells: 18

Number of references: 9

Combinational area: 24.000000

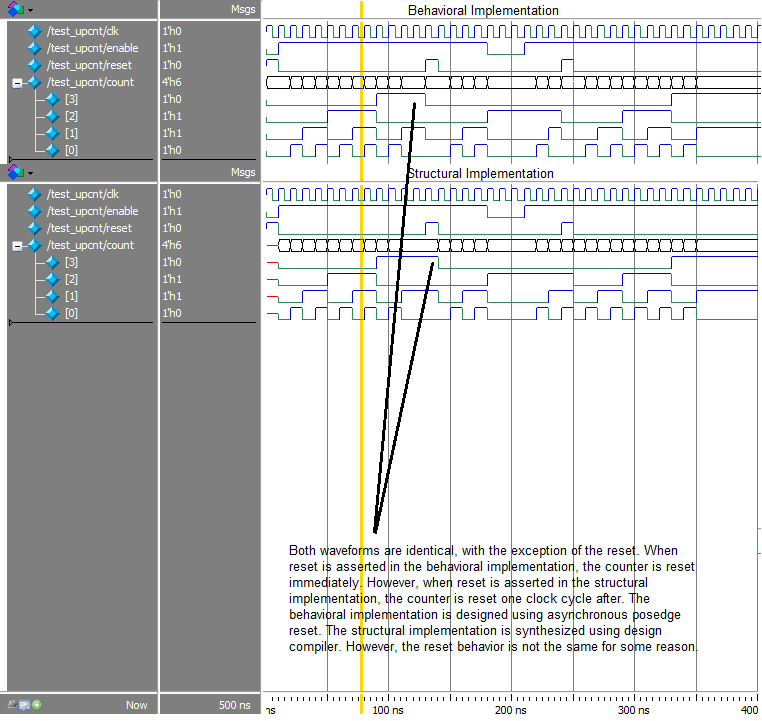
Noncombinational area: 36.000000

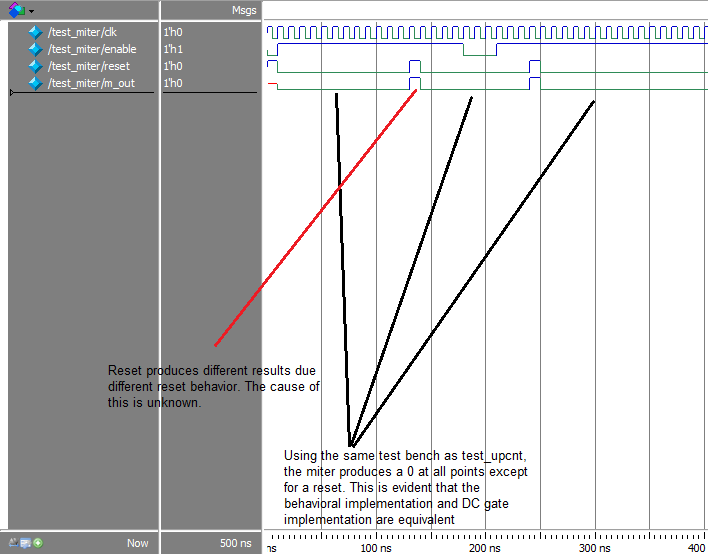
Net Interconnect area: undefined (Wire load has zero net area)

Total cell area: 60.000000

Total area: undefined

Next, I used a miter to compare the two implementations to verify functional equivalence. Below are the following waveforms. The first picture is a comparison between the waveform generated by upcnt.v compared to upcnt\_structural.v, using the testbench test\_upcnt. The second picture is the waveform generated by the miter.





**Conclusion**

Overall, this homework was fairly difficult and time consuming. I encountered two minor difficulties during this homework. In the first part of the homework, the processor’s memory wrote the address incorrectly. In order to write to address N, I had to specify address N+1. For example, in order to write to address 00, I had to use the instruction 0001XXXX instead of 0000XXXX. This was a very minor bug that did not get in the way of functionality. In the second part of the homework, the synthesized gate-level implementation of the up-counter did not mimic the reset functionality exactly of the behavioral implementation. I tried using an asynchronous positive edge triggered reset, negative edge triggered reset, and just reset for the always block parameters. However, none of these produced the correct reset behavior for the gate level implementation after synthesis. The reset of the behavioral implementation resets the counter immediately, while the reset of the synthesized gate-level implementation resets the counter after one clock cycle. Again, this is a minor unknown bug due to the design compiler which does not affect the overall functionality of the upcounter.