ECEN 651: Microprogrammed Control of Digital Systems
Department of Electrical and Computer Engineering
Texas A&M University

Prof. Mi Lu
TA: Ye Wang

Laboratory Exercise #1

Logic Level Modeling in Verilog

Objective

The objective of this first lab is to review how to do logic level modeling. To do so, we will make use of the simulator built into the Integrated Software Environment (ISE) provided by Xilinx.

Procedure

1. Launch the ISE Project Navigator and create a new design project.

   (a) Open a terminal window in the CentOS workstation and run the following command:
   >/softwares/setup/xilinx/ise-13.1

   This sets up the environment in order to run ISE and invokes the ISE toolset.

   (b) Create a new project called 'lab1' and ensure 'Isim' is set as the default simulator.

   (c) Leave the device properties as default. Since we will not be synthesizing our design just yet, we do not need to concern ourselves with these settings at the moment.

2. Simulate a Ripple Carry Counter

(a) Examine the code below. While attempting to understand what it is doing, draw a block diagram of the top level module. Please use standard digital logic symbols.

```
 ------------------------------------------------------------------------
****** Ripple Carry counter Top Block *******
------------------------------------------------------------------------
module RCC (q,clk,reset);
output [3:0] q;
input clk, reset;
T_FF tff0(q[0], clk, reset);
T_FF tff1(q[1], q[0], reset);
T_FF tff2(q[2], q[1], reset);
T_FF tff3(q[3], q[2], reset);
endmodule
-------------------------------------------------
****** Flip-flop T_FF *******
-------------------------------------------------
module T_FF(q, clk, reset);
output q;
input clk, reset;
wire d;
D_FF dff0(q, d, clk, reset);
not n1(d, q); // not is a verilog-provided primitive
endmodule
-------------------------------------------------
****** Flip_flop D_FF *******
-------------------------------------------------
module D_FF(q, d, clk, reset);
output q;
input d, clk, reset;
reg q;
// lots of construction here....
always @(posedge reset or negedge clk)
if (reset)
q = 1'b0;
else
```

```
q = d;
endmodule
-------------------------------------------------
****** Stimulus Block *******
-------------------------------------------------
module stimulus;
reg clk;
reg reset;
wire [3:0] q;
// instantiate the design block
RCC r1(q, clk, reset);
// control the clock signal that drives the design block. Cycle time = 10
initial
clk = 1'b0;
always
#5 clk=~clk; // togle clk every 5 time units
// control the reset signal that drives the design block
// reset is asserted from 0 to 20 and from 200 to 220
initial
begin
reset = 1'b1;
#15 reset = 1'b0;
#180 reset = 1'b1;
#10 reset = 1'b0;
#20 $finish; // teriminate the simulation
end
// Monitor the outputs
initial
$monitor($time, " Clk %b, Output q = %d",clk,q);
endmodule
```

(b) Open an editor (you may use the one provided in ISE or your favorite editor provided by Linux) and enter the code above. Be sure and put each individual module in its own source file and name each file appropriately.

(c) Within ISE simulate the test bench provided above. Demonstrate your progress to the TA.

(d) Take a screen capture of the waveform from your simulation and include it in your write-up.

3. Design a 4-to-1 Multiplexer

(a) Draw the logic necessary to build a 4:1 Mux with select lines S0 and S1.
Note: There are many different ways to accomplish this. You may use hierarchy in your design, but your diagrams must reflect this.

(b) Create the Verilog to describe the 4:1 Mux.

(c) Create the testbench to completely test all combinations of input.

(d) Simulate the testbench in ISE. Demonstrate your progress to the TA and be sure to include the resultant waveform in your lab write-up.

1 Deliverables

1. Submit a lab report that captures your efforts in lab.

2. Include all Verilog source files with appropriate comments.

   Note: Code submitted without adequate comments will not be graded!

3. Be sure to include all material requested in lab.

4. Answer the following review questions:

   (a) What does the *$monitor* statement in the provided test bench do? What about *$finish*?

   (b) Similarly, what do the # symbols signify in the provided test bench? What about *$time*?

   (c) The 4:1 multiplexer above can be built directly using gates, or it can be constructed using three 2:1 multiplexers, of which are constructed from gates. Provide some intuition as to how the delay through the 4:1 multiplexer would differ for both of the aforementioned cases.