Mechatronics Department
Computer Organization, CSE315
Third Year – 2014/2015

Sheet 3, Verilog & Logic Design

Question 1:

Given the following Verilog code, what value of "a" is displayed?

```
always @(clk)
begin
    a = 0;
    a <= 1;
    $display(a);
end
```

Question 2:

What is the difference between the following two lines of Verilog code?

```
#5 a = b;
a = #5 b;
```

Question 3:

What is the difference between:

```
c = foo ? a : b;
```

and

```
if (foo) c = a; else c = b;
```

Question 4:

Using the given, draw the waveforms for the following versions of a (each version is separate, i.e. not in the same run):

```
reg clk;
reg a;
always #10 clk = ~clk;

(1) always @(clk) a = #5 clk;
(2) always @(clk) a = #10 clk;
(3) always @(clk) a = #15 clk;

Now, change a to wire, and draw for:
(4) assign #5 a = clk;
(5) assign #10 a = clk;
(6) assign #15 a = clk;
```

Question 5:

Draw the state diagram to output a "1" for one cycle if the sequence "0110" shows up (the leading 0s cannot be used in more than one sequence). Write a Verilog code for the machine using one-hot state encoding.

Question 6:

Read each of the following statements carefully. If the statement is true circle T, otherwise circle F.

T F 1. Verilog is case sensitive.

T F 2. Verilog synthesizers treat the white space ' ' and carriage returns differently.

T F 3. "beginmodule" and "endmodule" are reserved words in Verilog.

T F 4. "2'b1x == 2'b1x;" has a "true" return value.

T F 5. An "if" statement must always be inside of an "always" block.

T F 6. The left argument of a statement in an initial or always block may be a wire.

T F 7. One must always declare the data type of a signal before using that signal within a design.

Question 7:

For each of the following statements, write the letter corresponding to the best answer in the space provided.

1. _____ Consider the following choices below. To comment Verilog Code, one may use:
   - I A "Double-slash" // for a single-line comment.
   - II Multiple "Double-slashes" (one per line) for a multiple-line comment.
   - III A "Block-comment" /* */ for a single-line comment.
   - IV A "Block-comment" /* */ for a multiple-line comment.

   A. I and II
   B. I and IV
   C. II and III
   D. I, II, and IV
   E. I, II, III and IV

2. _____ Which of the following is (are) equivalent to logic level 1?
   A. 1
   B. 1'b1
   C. 1 'b 1
   D. A and B
   E. A, B and C

3. _____ Which of the following is true about the always block?
   A. There may be exactly one always block in a design.
   B. There may be exactly one always block in a module.
   C. Execution of an always block occurs exactly once per simulation run.
   D. An always block may be used to generate a periodic signal.

4. _____ Which of the following is not true about operators?
   A. A logical "or" is performed by writing "C = A || B;"
   B. '!' performs logical negation while '~' performs bitwise negation.
   C. The two types of "or" operators are "logical" and "bitwise."
   D. The "shift right" (>>) operator inserts zeros on the left end of its argument.

Question 8:

In this question five lower-level modules are written and then two of these are combined to create a top-level design. For each module, write a complete Verilog program with correct declaration, etc.

Module 1: Gray Counter.

In this module, create a three-bit gray counter with positive reset. When reset, the count value becomes "000". Recall that a Gray Counter changes only one-bit at a time. (000, 001, 011, 010, 110, 111, 101, 100).
The following are the ports of the module:

CLK	1-bit clock input, all actions on positive edge
RESET	1-bit reset, causes reset on positive edge
GRAY_OUT	3-bit result

Module 2: Parallel-in, Serial-out Shift Register.

In this module, create a register that loads data in parallel but shift data out serially, MSB first. The following are the ports of the module:

CLK	1-bit clock, all operations must be on the rising edge
LD	1-bit input, when high, inputs are loaded into the shift register
SHIFT	1-bit shift enable input, when high, contents of shift register are shifted out on to the serial output Q
PI	8-bit parallel data input
Q	1-bit serial output

Module 3: Up-Down, Loadable Counter.

In this module, create a counter that counts in both the up and down directions. The preset is to be set to decimal value 3. That is, upon asserting the reset signal low, the register value should be reset to 3. Also, upon asserting the load signal LD, the register value should be set to the input value DIN. Both the reset and the load are synchronous and the module should count on the rising edge of the clock.
The following are the ports of the module:

CLK	1-bit clock input, all actions performed on rising edge
RESET_N	1-bit preset (synchronous)
UP_DNN	1-bit input (if '1', then count up, if '0', then count down)
LD	1-bit load enable input, loads synchronized with CLK rising edge
DIN	3-bit input data for loading counter value
Q	3-bit result

Module 4: Magnitude Comparator.

In this module, two inputs are treated as signed, 2's complement numbers and compared. The output should be a 1 if the AD_IN is greater than or equal to VS_IN. The following are the ports of the module:

AD_IN	8-bit data input
VS_IN	5-bit input
GTE	1-bit output

Module 5: Multiplexer.
In this module, create a byte-wide 8 to 1 multiplexer. In this case, the value on the 3-bit select line will route 1 of 8 inputs to the output. This module is purely combinatorial. The following are the ports of the module:

SEL	3-bit select line
D0, D1, D2, D3, D4, D5, D6, and D7	8-bit data inputs
O	8-bit output

Top-Level Design
For this design, combine Gray Counter (Module 1) with the Multiplexer (Module 5) to create a circuit such that the output of the Gray counter controls the select lines of the multiplexer. The top-level design has the following port definitions:

CLK	1-bit clock
RESET	1-bit reset line
OUT_DATA	8-bit data output
IN0, IN1, IN2, ... IN7	8-bit data inputs

Question 9:
The following Verilog code is supposed to describe a circuit that has an 8-bit data input, an 8-bit data output, plus two additional single-bit outputs. The over_flow output is strictly combinatorial. You may assume that the operations to produce data_out, carry_out and over_flow are correct (i.e. the functional specification for these values is correct). However, syntax errors for these statements do exist.

Examine the code below and identify any errors that would prevent this code from compiling and synthesizing properly. These errors may include syntax errors, logical design errors or needed lines that are missing from the code. Indicate the modifications you would make to the code to make it work. Include comments beside these modifications or the portion of the code that is incorrect. There are 20 errors in the code below.

```
module bad_module is (clk,
            reset,
            [7:0] data_in,
            data_out;
            carry out,
            over_flow,
            );
input clk;
input reset;
input carry_in;
input data_in;
output data_out;
output carry_out;
output over_flow;
reg [7:0] data_out;
reg over_flow;
always @ (posedge clk | posedge reset)
if reset then
```

```
        data_out = 0;
        carry_out = 0;
else
        data_out = (data_in * 2) + carry_in;
        carryout = ~& data_in[7|6];
end
assign over_flow <= data_out7;
end module
```