ECE 3829: Advanced Digital System Design using FPGAs  
C term 2016

Exam 1: February 1, 2016 (11.00 to 11.50)

TEST PREPARATION - SOLUTIONS

This exam is closed book (no books or notes or electronic devices allowed).  
Here are some Verilog keywords to remind you:

```verilog
module, input, output, endmodule, assign, always, if, else, case,  
endcase, posedge, negedge
```

Do not use the wildcard character ‘*’ in the sensitivity list for an always statement – list all the appropriate signals.

Read each question carefully
Use a standard and consistent coding style.

Try and answer each question - if you consider any question to be ambiguous then state any assumptions you make.

NAME: ___________________________
ECE Box: ________________________

**Question 1 [25 marks]:** The function table of a 2-line to 4-line decoder with an enable input is:

<table>
<thead>
<tr>
<th>Enable</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>1</td>
<td>don’t care</td>
<td>don’t care</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Write a Verilog synthesizable module description of this decoder using:

A) A **continuous assignment statement with the conditional operator.**
B) An **always** statement with a **CASE** statement
module decoder(
    input g,
    input a,
    input b,
    output reg [3:0] y;

    wire [1:0] sel;
    assign sel = (b,a);

    // always with case example
    always @(g, sel) // need reg for y
        case ({g, sel})
            0: y = 4'b0001; // or 3'b001: etc
            1: y = 4'b0010;
            2: y = 4'b0100;
            3: y = 4'b1000;
            default: y = 4'b0000;
        endcase

    // always with if example
    // always @(g, b, a) // need reg for y
    // if (g == 1)
    //     y = 4'b0000;
    // else if ( {b, a} == 2'b00)
    //     y = 4'b0001;
    // else if (~b & a)
    //     y = 4'b0010;
    // else if ( {b, a} == 2'b10)
    //     y = 4'b0100;
    // else
    //     y = 4'b1000;
endmodule
**Question 2 [20 marks]:** Write the Verilog synthesizable module description of a 6-bit priority encoder. The six inputs are i0 to i5 with i5 having the highest priority input. There are two outputs, ‘a’, and ‘gs’. The output ‘a’ should encode the address of the highest priority input (‘000’ to ‘101’) and the ‘gs’ output should be ‘1’ when at least one of the inputs is ‘1’.

```verilog
module encoder_6_bit(
    input [5:0] i, // or could have separate inputs 15, 14, 13, 12, 11, 10
    output reg [2:0] a, // reg if always used
    output reg gs
);

// assign a = i[5] == 1'b1 ? 3'b101 :
//     i[4] == 1'b1 ? 3'b100 :
//     i[3] == 1'b1 ? 3'b011 :
//     i[2] == 1'b1 ? 3'b010 :
//     i[1] == 1'b1 ? 3'b001 :
//     3'b000;

always @(t)
    if (i[5] == 1'b1)
        a = 3'b101;
    else if (i[4])
        a = 3'b100;
    else if (i[3])
        a = 3'b011;
    else if (i[2])
        a = 3'b010;
    else if (i[1])
        a = 3'b001;
    else
        a = 3'b000;

// assign gs = | i;
endmodule
```

**Question 3 [20 marks]:** Write the Verilog synthesizable module description of a counter with an asynchronous reset (active high) signal. The counter should count from 8 to 88 on a negative edge of a clock but only when an enable signal (called ‘clk_enable’) is high. Once the counter gets to 88 it should restart at 8.

The Verilog code for the counter is not provided in the image. However, the general idea is to use a counter module with an asynchronous reset and enable signal. The counter should increment on a negative edge of the clock and only when the enable signal is high. The counter should restart at 8 when it reaches 88.
Question 4 [35 marks]: It is important to use a common clock in FPGA logic designs so that the high-speed low-skew dedicated clock routing resources can be used. Keeping this principle in mind, design the following:

Assume that an FPGA has a 50MHz master clock (clk_50M). Write an always statement to generate a 12-bit LFSR that shifts at the 1MHz rate (derived from the 50MHz clock). The LFSR should be loaded with all '1's on reset (active high, asynchronous reset). The output is taken from the least significant stage (bit 0). It has two feedback taps from bits 4 and 8 that feedback an exclusive-or to the most significant stage (bit 11). The LFSR shifts on a positive edge of the clock signal.

The start of the module description is provided:

```verilog
module lfsr (  
    input  clk_50M,  
    input  reset,  
    output  q  
);  
```

See next page for solution – minor variations OK
module lfsr(
    input clk_50M,
    input reset,
    output q
);

reg [5:0] counter_50M; // count 0 to 49
reg [11:0] lfsr; // 12 bit shift register

wire xor_taps;

// create 1MHz clock from 50MHz clock
always @(posedge reset or posedge clk_50M)
    if (reset)
        counter_50M <= 0;
    else if (counter_50M == 50 - 1)
        counter_50M <= 0;
    else
        counter_50M <= counter_50M + 1;

assign xor_taps = lfsr[6] ^ lfsr[4];

// create LFSR that shifts on 1MHz clock
always @(posedge reset or posedge clk_50M)
    if (reset)
        lfsr <= 12'HFFF;
    else if (counter_50M == 0)
        lfsr <= (xor_taps, lfsr[11:1]);

assign q = lfsr[0];
endmodule