This exam is closed book (no books or notes or electronic devices allowed).

Here are some Verilog keywords to remind you:

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 [20 marks]: The function table of a simple combinational circuit is (‘a’, ‘b’, and ‘c’ are inputs, ‘j’ and ‘k’ are outputs):

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>j</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Write a Verilog synthesizable module description of this circuit using:

A **continuous assignment statement** with the **conditional operator** for the ‘j’ output.

An **always** statement with a **case** statement for the ‘k’ output.

Note: use an internal three-bit signal called ‘abc’ to combine the ‘a’, ‘b’ and ‘c’ inputs together

The start of the module description is provided (you do not need to write out again but modify as necessary):

```verilog
module comb(
    input a, b, c,
    output j,
    output k
);
```
```
text{module comb(
input a, b, c,
output j,
output reg k // reg type required
);

wire [2:0] abc;
assign abc = {a, b, c}; // combine three signals into one

// we will use a conditional operator statement for output j
assign j = abc == 3'b001 ? 1'b1 :
        abc == 3'b011 ? 1'b1 :
                    1'b0;

always @(abc) // we will use a case statement for output k
    case (abc)
        3'b000: k = 1'b1;
        3'b001: k = 1'b1;
        3'b011: k = 1'b1;
        3'b100: k = 1'b1;
        3'b101: k = 1'b1;
        default: k = 1'b0;
    endcase
endmodule
```
**Question 2 [20 marks]:** Write the Verilog synthesizable module description of a 5-bit priority encoder. The five inputs are i0 to i4 with i4 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 ‘100’) and the ‘gs’ output should be ‘1’ when at least one of the inputs is ‘1’.

Important: Use an `always` statement to describe the ‘a’ output and a `continuous assignment` statement to describe the ‘gs’ output.

The start of the module is provided (you do not need to write it again but modify as required):

```verilog
module q2_priority(
    input [4:0] i,
    output [2:0] a,
    output   gs
);

always @ (i) // we need to use an IF statement to get the correct priority
if (i[4]) // i4 is highest priority, must check first
    a = 3'b100;
else if (i[3]) // or (i[4] == 1'b1)
    a = 3'b011;
else if (i[2])
    a = 3'b010;
else if (i[1])
    a = 3'b001;
else
    a = 3'b000;

assign gs = i[0] | i[1] | i[2] | i[3] | i[4];
// assign gs = | i; // this is the same as above
endmodule
```
**Question 3 [20 marks]**: Write the Verilog synthesizable module description of a 11-bit linear feedback shift register (LFSR) with a positive edge triggered clock ‘clk_fpga’, an asynchronous ‘reset_n’ (active_low) input to load the LFSR with a starting value (seed) of 0x7FF, and an input ‘lfsr_en’ that allows the shift register to shift when high.

The feedback taps are connected to stages 10, 9, 8, and 1.

The LFSR shifts right and the output is connected to the least significant bit (stage 0).

What is the value of the output bit for the first 13 clock cycles? ___________________________

The start of the module is provided (you do not need to write out again but modify as necessary):

```verilog
module q3_lfsr(
    input  clk_fpga,
    input  reset_n, // load shift register with starting value
    input  lfsr_en, // only shift when enable is high
    output   q // output bit of shift register
);
```

Output sequence of Q will be 1,1,1,1,1,1,1,1,1,1,1,0,1

Simulation (not required):
**Question 4 [40 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 100MHz master clock (clk_100M) that is used to clock all the flip-flops on the positive edge. Write an `always` statement to produce an effective 5MHz clock enable signal (create this internal signal and call it ‘count_enable’).

Write another `always` statement to generate a counter that counts from 0 to 99 at the 5MHz rate using the ‘count_enable’ signal. This counter has an *asynchronous* reset (active high) to start the counter at 0. The counter should increment from 0 to 99 (at the 5MHz rate) and then start again at 0.

An output signal called ‘zero’ should be set to ‘1’ whenever the counter has the value 0.

How many flip-flops will this design require and why? ______________________________
__________________________________________________________________________
__________________________________________________________________________

The start of the module description is provided (modify as required):

```verilog
module counter (    
    input    clk_100M, // 100MHz FPGA clock   
    input    reset,  // Async reset             
    output [6:0] q,   // Output from counter   
    output zero       // Output from counter     
); 
```

**12 flip-flops will be required:**

5 for 5-bit 5MHz counter (count from 0 to 19 for divide by 20)
7 for 7-bit q counter (count from 0 to 99)
module q4(
    input  clk_100M,
    input   reset,
    output reg [6:0] q,
    output   zero
);

parameter MAX_COUNT = 19; // used in divide by 20 counter
reg [4:0] counter_100M; // to count from 0 to 19
wire    count_enable;

// create 0 to 19 counter for 5MHz effective clock rate)
always @(posedge clk_100M, posedge reset)
    if (reset) // or (reset == 1'b1)
        counter_100M <= 5'b0;
    else if (counter_100M == MAX_COUNT) // restart every 20 clocks
        counter_100M <= 5'b0;
    else
        counter_100M <= counter_100M + 1'b1;

assign count_enable = counter_100M == MAX_COUNT;

// create 0 to 99 counter)
always @(posedge clk_100M, posedge reset)
    if (reset)
        q <= 7'b0;
    else if (counter_100M == 19) // counter enable at 5MHz
        if (count_enable) // alternative to above
            if (q == 99)
                q <= 7'b0;
            else
                q <= q + 1'b1;
    else
        assign zero = q == 0; // create zero signal when counter value is 00

endmodule