

Name: Ruiqi Zang

ID: 24075277

Instructions: Only neat, hand-written answers will be accepted (except for the sections 7b and 7c where you can use a computer). This homework assignment is **individual**. Use SystemVerilog for your answers.

1. (5%) Draw a schematic of the logic defined in the following Verilog code.

```
module exercise1(input a, b, c,
                 output y, z);

    assign y = a & b & c | a & b & ~c | a & ~b & c;
    assign z = a & b | ~a & ~b;
endmodule
```

$$y = abc \vee ab\bar{c} \vee a\bar{b}c$$

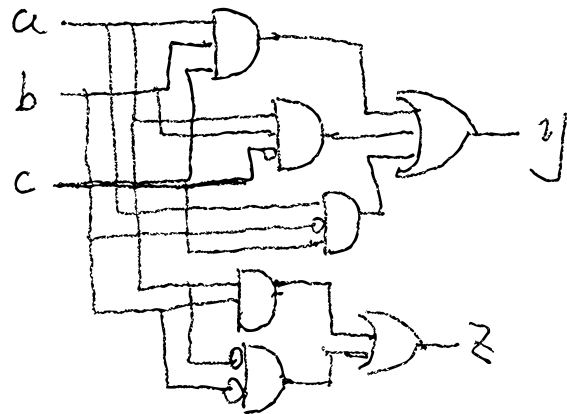
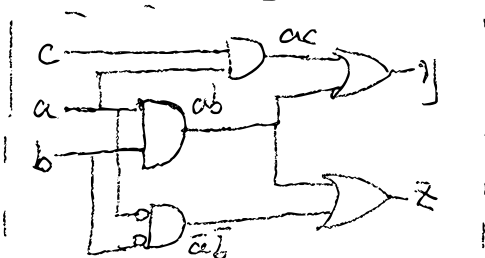
$$= ab(c \vee \bar{c}) \vee a\bar{b}c$$

$$= ab \vee a\bar{b}c$$

$$= a(b \vee \bar{b}c)$$

$$= a(b \vee c)$$

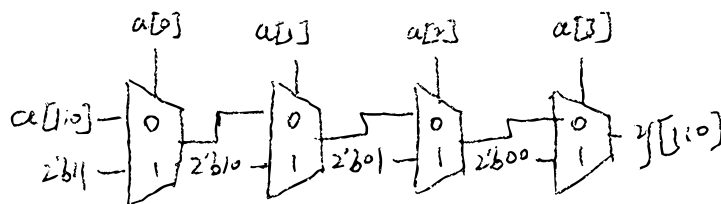
$$= ab \vee ac$$



2. (5%) Draw a schematic of the logic defined in the following Verilog code.

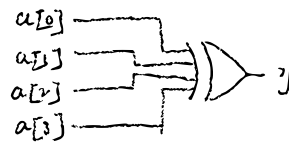
```
module exercise2(input [3:0] a,
                 output reg [1:0] y);

    always @(*)
        if (a[0]) y = 2'b11;
        else if (a[1]) y = 2'b10;
        else if (a[2]) y = 2'b01;
        else if (a[3]) y = 2'b00;
        else y = a[1:0];
endmodule
```



3. (5%) Draw a schematic of the logic defined in the following Verilog code.

```
module ex3(input [3:0] a, output y);
    assign y = ^a;
endmodule
```



4. (10%) Write HDL code that implements a multiplexer, 8 input to 1 output, all 32-bit wide.

```
module ex4( input logic [31:0] a, b, c, d, e, f, g, h,
            input logic [2:0] se , output logic [31:0] out);
```

```
    always_comb
    case2(se)
```

```
        3'b000: out = a ;
        3'b001: out = b ;
        3'b010: out = c ;
        3'b011: out = d ;
        3'b100: out = e ;
        3'b101: out = f ;
        3'b110: out = g ;
        3'b111: out = h ;
        default: out = 0 ;
```

```
    endcase
```

```
endmodule
```

5. (10%) Write HDL code that implements a Priority Encoder with 8 inputs of 1 bit each and 1 output of 3 bits.

```
module ex5 ( input logic a, b, c, d, e, f, g, h,  
             output logic [2:0] out
```

```
             logic [7:0] value ;
```

```
             assign value = {a, b, c, d, e, f, g, h} ;
```

```
             always-comb
```

```
             casez (value)
```

```
                 0'b1xxxxxx: out = 3'b111 ;
```

```
                 0'b01xxxxx: out = 3'b110 ;
```

```
                 0'b00xxxxx: out = 3'b101 ;
```

```
                 0'b000xxxx: out = 3'b100 ;
```

```
                 0'b0000xxx: out = 3'b011 ;
```

```
                 0'b00000xx: out = 3'b010 ;
```

```
                 0'b000000x: out = 3'b001 ;
```

```
                 0'b0000000: out = 3'b000 ;
```

```
                 default: out = 3'b000 ;
```

```
             endcase
```

```
endmodule
```

6. (20%) Write HDL code to synthesize the following circuits:

a. 8-bit register.

```
module ex6a( input logic clk, input logic [7:0] in, output logic [7:0] out );  
    always_ff @(posedge clk)  
        out <= in ;  
endmodule
```

b. 9-bit Register with Asynchronous Reset

```
module ex6b( input logic clk, input logic reset,  
             input logic [8:0] in, output logic [8:0] out );  
  
    always_ff @(posedge clk, posedge reset )  
        if(reset) out <= 9'b0 ;  
        else      out <= in ;  
endmodule
```

c. N-bit Register with Synchronous Reset where N is a parameter

```
module ex6c( input logic clk, input logic reset,  
             input logic [N-1:0] in, output logic [N-1:0] out );  
  
    always_ff @(posedge clk )  
        if(reset) out <= N'b0 ;  
        else      out <= in ;  
endmodule
```

d. N-bit register with Enable and Asynchronous reset
where N is a parameter

```
module exbd ( input logic clk, input logic reset, input logic en,  
              input logic [N-1:0] in, output logic [N-1:0] out );
```

```
    always_ff @(posedge clk, posedge reset)
```

```
        if (reset) out <= N'b0;
```

```
        else if (en) out <= in;
```

```
endmodule
```

e. 8-bit latch

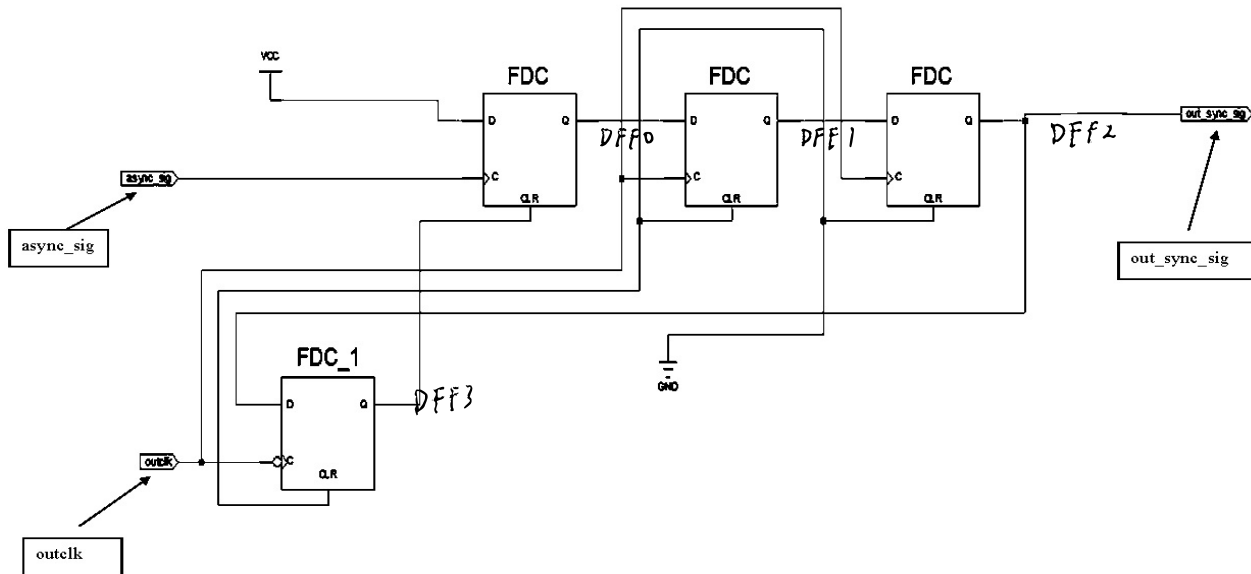
```
module exbe ( input logic clk, input logic [7:0] in, output logic [7:0] out );
```

```
    always_latch
```

```
        if (clk) out <= in;
```

```
endmodule
```

(45%) 7. Look at the diagram below and answer the questions that follow.



The "clr" of Flip-Flops is asynchronous and active high. Note the inversion (circle) on the "C" (clock) input of FDC_1.

(a) (25%) Write HDL code that will result in the synthesis of this circuit.

(b) (20%) (use a computer for this section): In Quartus, create a project and synthesize your code (you do not need to do a full compilation, just analysis and synthesis). Submit a simulation showing that your circuit is operating (if you don't understand what it is supposed to do, look at its inputs and outputs and devise a simulation that will cause those inputs and outputs to toggle).

(c) (10% Bonus) What does the circuit do? What could it be useful for? Explain how it works (an annotated simulation may be helpful, you can use a computer for this section).

a) module q57 (input logic async_sig, input logic outclk,
output logic out_sync_sig);

```
logic DFF3;  
logic DFF;  
logic DFF1;  
logic DFF2;  
logic VCC;  
logic GND;  
assign VCC = 1;  
assign GND = 0;  
assign out_sync_sig = DFF2;  
always_ff @(posedge async_sig, posedge DFF3)  
    if (DFF3) DFF <= 0;  
    else DFF <= VCC;  
  
always_ff @(posedge outclk, posedge GND)  
    if (GND) DFF1 <= 0;  
    else DFF1 <= DFF;  
  
always_ff @(posedge outclk, posedge GND)  
    if (GND) DFF2 <= 0;  
    else DFF2 <= DFF1;  
  
always_ff @(posedge ~outclk, posedge GND)  
    if (GND) DFF3 <= 0;  
    else DFF3 <= DFF2;
```

endmodule

c)

input = 1, the output = 1 when clock is in the second active high,
After two period for clock, the output = 0,
It can be a control light by sound,
make sound as input, the light will light two period,

```
//a)
module QS7( input logic async_sig, input logic outclk, output logic out_sync_sig);

    logic DFF, DFF1, DFF2, DFF3;
    logic VCC, GND;
    assign VCC = 1;
    assign GND = 0;
    assign out_sync_sig = DFF2;

    always_ff@ (posedge async_sig, posedge DFF3)
        if (DFF3) DFF <= 0;
        else      DFF <= VCC;

    always_ff@ (posedge outclk, posedge GND)
        if (GND) DFF1 <= 0;
        else      DFF1 <= DFF;

    always_ff@ (posedge outclk, posedge GND)
        if (GND) DFF2 <= 0;
        else      DFF2 <= DFF1;

    always_ff@ (posedge ~outclk, posedge GND)
        if (GND) DFF3 <= 0;
        else      DFF3 <= DFF2;

endmodule
```

