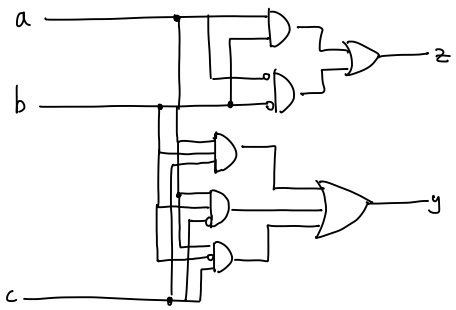
Name: Mercury Mcindoe ID: 88594505

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.

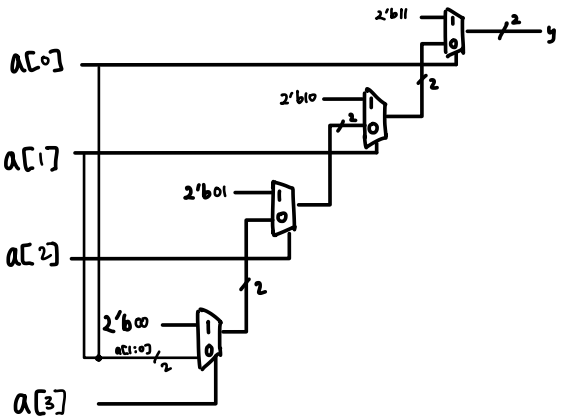
1. (5%) Draw a schematic of the logic defined in the following Verilog code.

```
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
```



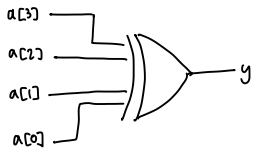
2. (5%) Draw a schematic of the logic defined in the following Verilog code.

```
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 mux8to1 ( input logic [31:0] x1, input logic [31:0] x2, input logic [31:0] x3, input logic [31:0] x4, input logic [31:0] x5, input logic [31:0] x6, input logic [31:0] x7, input logic [31:0] x8
                 input logic [2:0] choose , output logic [31:0] y);

    always_comb begin
        case (choose)
            3'b000 : y = x1;
            3'b001 : y = x2;
            3'b010 : y = x3;
            3'b011 : y = x4;
            3'b100 : y = x5;
            3'b101 : y = x6;
            3'b110 : y = x7;
            3'b111 : y = x8;
            default : y = 32'b0;
        endcase
    end
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 priority_encoder ( input logic x0, input logic x1, input logic x2, input logic x3, input logic x4, input logic x5, input logic x6,
                          input logic x7, output logic [2:0] y );

always_comb begin

    if (x7)  y = 3'b111;
    else if (x6)  y = 3'b110;
    else if (x5)  y = 3'b101;
    else if (x4)  y = 3'b100;
    else if (x3)  y = 3'b011;
    else if (x2)  y = 3'b010;
    else if (x1)  y = 3'b001;
    else  y = 3'b000;

end

endmodule
```

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

a. 8-bit register.

```
module reg8bit (input logic [7:0] in, input logic clk, output logic [7:0] out);

    always_ff @ (posedge clk)
        out <= in;

endmodule
```

b. 9-bit Register with Asynchronous Reset

```
module asynchreg_9 (input logic [8:0] in, input logic clk, input logic reset, output logic [8:0] out);

    always_ff @ (posedge clk or posedge reset) begin
        if (reset) out <= 9'b0;
        else out <= in;
    end
endmodule
```

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

```
module Nbit_synch_reg #(parameter N=2)(input logic [N-1:0] in, input logic clk, input logic reset, output logic
    [N-1:0] out);

    always_ff @ (posedge clk) begin
        if (reset) out <= {N{1'b0}};
        else out <= in;
    end
endmodule
```

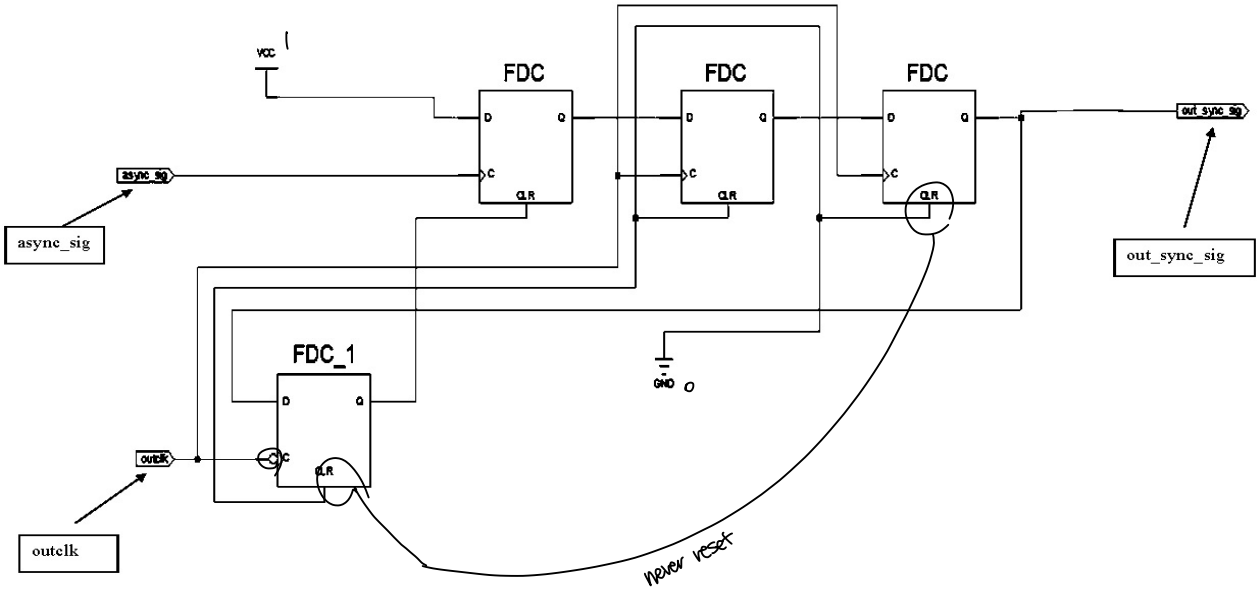
d. N-bit register with Enable and Asynchronous reset where N is a parameter

```
module en_Reg_ff(parameter N=2)( input logic [N-1:0] in, input logic en, input logic clk, input logic reset, output logic [N-1:0] out);

    always_ff @(posedge clk or posedge reset) begin
        if (reset) out <= {N{1'b0}};
        else if (en) out <= in;
        else out <= {N{1'bZ}};
    end
endmodule
```

e. 8-bit latch

```
module latch_8bit( input logic [7:0] in, input logic clk, 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).

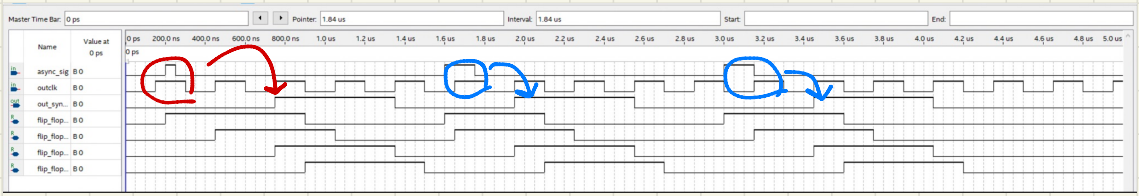
(c) (10% Bonus) What does the circuit do? What could it be useful for? Explain how it works (an annotated simulation may be helpful, you can use a computer for this section).

(a)

```
module flip_flop( input logic D, input logic clr, input logic C, output logic Q);

        always_ff @ (posedge clr or posedge C)
                if (clr)  Q<=0;
                else  Q<=D;

endmodule

module hw_circuit( input logic async_sig, input logic outclk, output logic out_sync_sig);

        logic vcc=1;
        logic gnd=0;

        logic fdc1_2;
        logic fdc2_3;
        logic fdc3_4;
        logic fdc4_1;

        flip_flop flip_flop1( .D(vcc), .clr(fdc4_1), .C(async_sig), .Q(fdc1_2));
        flip_flop flip_flop2( .D(fdc1_2), .clr(gnd), .C(outclk), .Q(fdc2_3));
        flip_flop flip_flop3( .D(fdc2_3), .clr(gnd), .C(outclk), .Q(fdc3_4));
        flip_flop flip_flop( .D(fdc3_4), .clr(gnd), .C(~outclk), .Q(fdc4_1));

        assign out_sync_sig = fdc3_4;

endmodule
```

in-case, file wasn't submitted

(b), (c)



The circuit result above shows that it catches signals that are not even in the clock cycle. Even though a delay might exist it produces synchronous signals despite that they were inputted as an asynchronous signal. The difference shows that if async_sig went high within the cycle of the clock when it is high (check the red circle), the output signal is produced two cycles later. However, when async_sig is high outside of the clock cycle (went the clock goes high) the output signal is produced one cycle later. Therefore, we can see that the circuit produces synchronized signals with an asynchronous input signal by producing a slight delay.

I believe such a signal could be useful in mixed signal systems. Synchronizaion circuits can acts as an interface that allows processing and integration of analog and digital information in a coordinated manner. Also in communication systems, synchronization circuits allow synchronizing data streams between nodes or devices that share different clock domains. This ensures that data is properly aligned, timed and interpreted by systems consisting the whole system.