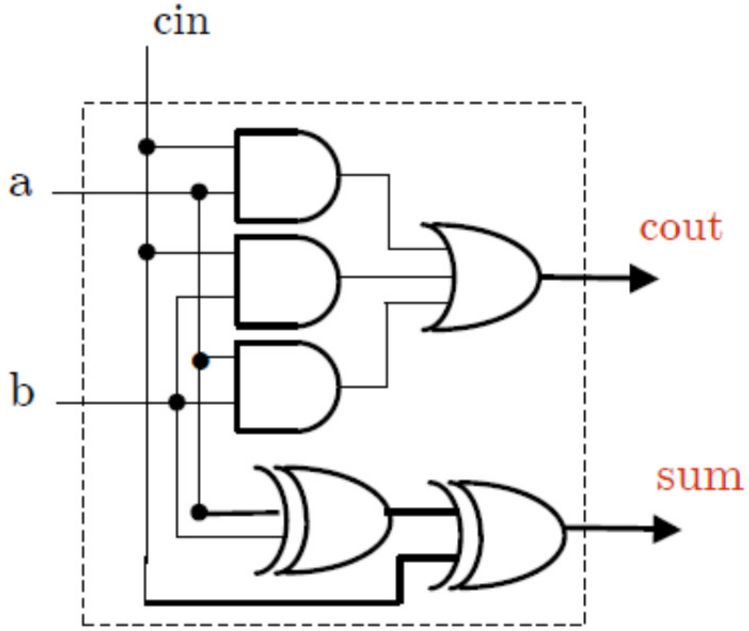
Problem 1: Draw the synthesized circuits described by the Verilog codes below.

(a)

```
module FA (a, b, cin, cout, sum);
        input a, b, cin; //inputs
        output cout, sum; //outputs
        wire w1, w2, w3, w4; // internal nets

        xor #(10) (w1, a, b); //delay time of 10 units
        xor #(10) (sum, w1, cin);
        and #(8) (w2, a, b); //delay time of 8 units
        and #(8) (w3, a, cin);
        and #(8) (w4, b, cin);
        or #(10)(cout, w2, w3, w4);
endmodule
```

Solution:

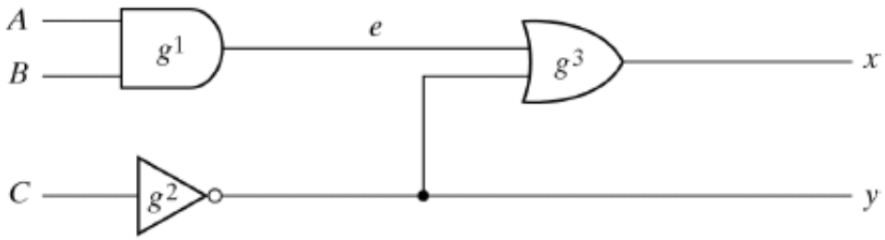


(b)

```
module smpl_circuit (A, B, C, x, y);
        input A, B, C;
        output x, y;
        wire e;

        and #(10) g1(e, A, B);
        not #(5) g2(y, C);
        or #(15) g3(x, e, y);
endmodule
```

Solution:

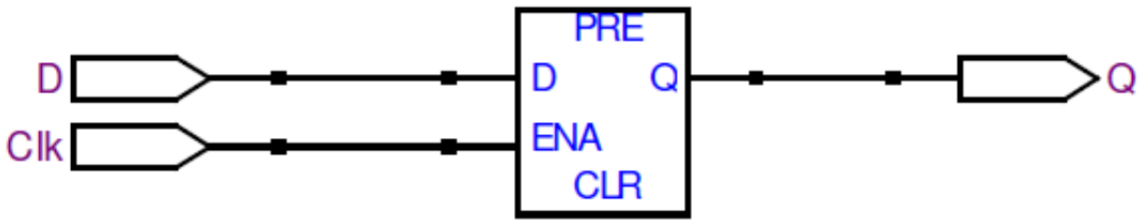


(c)

```
module D_latch (D, Clk, Q);
        input D, Clk;
        output Q;
        reg Q;

        always @(D or posedge Clk)
                if (Clk)
                        Q = D;
endmodule
```

Solution:



Problem 2: Implement an 8-bit ripple carry adder, using as building block the 1-bit full adder from **1(a)**. Use the following module template for your top level design:

```
module adder8 (A, B, cin, S, cout);
        input[7:0] A, B;
        input cin;
        output[7:0] S;
        output cout;
        wire c1, c2, c3, c4, c5, c6, c7;

        //place code here

endmodule
```

Solution:

```
module adder8(A, B, cin, S, cout);
        input[7:0] A, B;
        input cin;
        output[7:0] S;
        output cout;
        wire c1, c2, c3, c4, c5, c6, c7;

        //place code here

        //8 instantiated 1-bit Full Adders
        FA fa0(A[0], B[0], cin, c1, S[0]);
        FA fa1(A[1], B[1], c1, c2, S[1]);
        FA fa2(A[2], B[2], c2, c3, S[2]);
        FA fa3(A[3], B[3], c3, c4, S[3]);

        FA fa4(A[4], B[4], c4, c5, S[4]);
        FA fa5(A[5], B[5], c5, c6, S[5]);
        FA fa6(A[6], B[6], c6, c7, S[6]);
        FA fa7(A[7], B[7], c7, cout, S[7]);

endmodule
```

Problem 3: Implement a 4-bit magnitude comparator using dataflow description (i.e. using continues assignment). Here's the skeleton of the module.

```
module mag_comp4 (A, B, ALB, AGB, AEB);
    input [3:0] A,B;
    output ALB, AGB, AEB;
    //ALB → A<B, AGB → A>B, AEB → A=B

    //place code here

endmodule
```

Solution:

```
module mag_comp (A, B, ALB, AGB, AEB);
    input [3:0] A, B;
    output ALB, AGB, AEB;
    //ALB → A<B, AGB → A>B, AEB → A=B

    //place code here

    assign ALB = (A<B), AGB = (A>B), AEB = (A==B);

endmodule
```

Problem 4: The following code is intended to implement a 1:2 de-MUX. The input is x and two outputs are y and z, and the switch control is s.

```
wire x, y;
reg z;
always @(x or y or z)
   begin
      if (!s) y=x;
      else z=x;
   end
```

(a) Find out all things wrong in this code.

Solution:

— y has to be defined as reg

— y and z must not be in the trigger list

— s must be in the sentivity list

(b) Does the code produce any latches? If no, state why. If yes, how many latches can be produced? And by which statement(s)?

Solution:

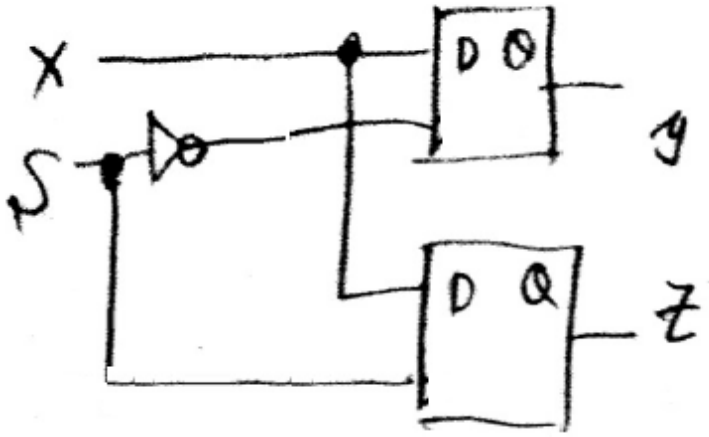
— Yes. 2 latches

— One to store previous value of z, when s = 0 (y = x).

— One to store previous value of y, when s = 1 (z = x).

(c) After all problems you find in **(a)** are fixed, draw the logic diagram of the synthesized hardware from the corrected code.

Solution:

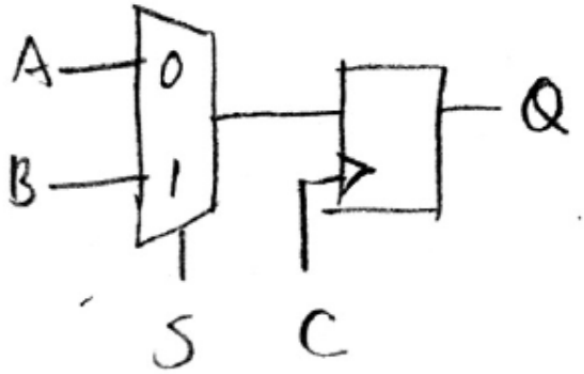


Problem 5: Below is a short snippet of Verilog code of a digital functional block. Draw the logic circuit diagram that is described by the Verilog code.

```
module M(Q, S, A, B, C);
        output Q;
        input S, A, B, C;
        reg Q;

        always @(posedge C)
                Q = (S) ? A : B ;
endmodule;
```

Solution:



Problem 6: In this problem, design a 3-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. For example, a 2-bit gray counter has a count sequence 00, 01, 11, 10 corresponding to decimal count values of 0, 1, 2, and 3 respectively. In the gray count sequence, only one bit changes between adjacent count values, and the right-most bit is changed as long as it does not result in a code word that has been visited earlier.

The following are the ports of the module:

C	1-bit clock input, all actions must be on the positive edge
R	1-bit reset, causes reset on the positive edge
OUT	3-bit result

Solution:

```
module GRAY_CNT (
                 C,
                 R,
               OUT);

     input C;
     input R;
     output [2:0] OUT;

     reg [2:0] OUT;

     always @ (posedge C)
          if (R)
               OUT = 3'b000;
          else
               case (OUT)
                    3'h0:    OUT    =    3'h1;
                    3'h1:    OUT    =    3'h3;
                    3'h3:    OUT    =    3'h2;
                    3'h2:    OUT    =    3'h6;
                    3'h6:    OUT    =    3'h7;
                    3'h7:    OUT    =    3'h5;
                    3'h5:    OUT    =    3'h4;
                    3'h4:   OUT    =    3'h0;
                    default: OUT   =    3'h0;
               endcase
endmodule
```

Problem 7: 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:

S	3-bit select line
I0, I1, I2, I3, I4, I5, I6, and I7	8-bit data inputs
O	8-bit output

Solution:

```
module MUX_8 (
      S,
      I0,
      I1,
      I2,
      I3,
      I4,
      I5,
      I6,
      I7,
      O);

     input   [2:0]   S;
     input   [7:0]   I0;
     input   [7:0]   I1;
     input   [7:0]   I2;
     input   [7:0]   I3;
     input   [7:0]   I4;
     input   [7:0]   I5;
     input   [7:0]   I6;
     input   [7:0]   I7;
     output  [7:0]   O;

     reg O;

     always @ (*)
          case (S)
               3'b000:  O  =  I0;
               3'b001:  O  =  I1;
               3'b010:  O  =  I2;
               3'b011:  O  =  I3;
               3'b100:  O  =  I4;
               3'b101:  O  =  I5;
               3'b110:  O  =  I6;
               3'b111:  O  =  I7;
          endcase
endmodule
```

Problem 8: For this design, combine gray counter (from Problem 6) with the multiplexer (from Problem 7) 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:

C	1-bit clock
R	1-bit reset line
D0, D1, D2, D3, D4, D5, D6, and D7	8-bit data inputs
OUT	8-bit data output

Solution:

```
module TOP (
        C,
        R,
        OUT,
        D0,
        D1,
        D2,
        D3,
        D4,
        D5,
        D6,
        D7);

      input C, R;
      input [7:0]  D0;
      input  [7:0]  D1;
      input  [7:0]  D2;
      input  [7:0]  D3;
      input  [7:0]  D4;
      input  [7:0]  D5;
      input  [7:0]  D6;
      input [7:0]  D7;
      output [7:0] OUT;

      wire [7:0] GRAYOUT;

      GRAY_CNT GRAYCNT_INST1 (C, R, GRAYOUT);
      MUX_8 MUX8_INST1 (GRAYOUT, D0, D1, D2, D3, D4, D5, D6,
      D7, OUT);

endmodule
```