

# Digital Logic

## Assignment 2

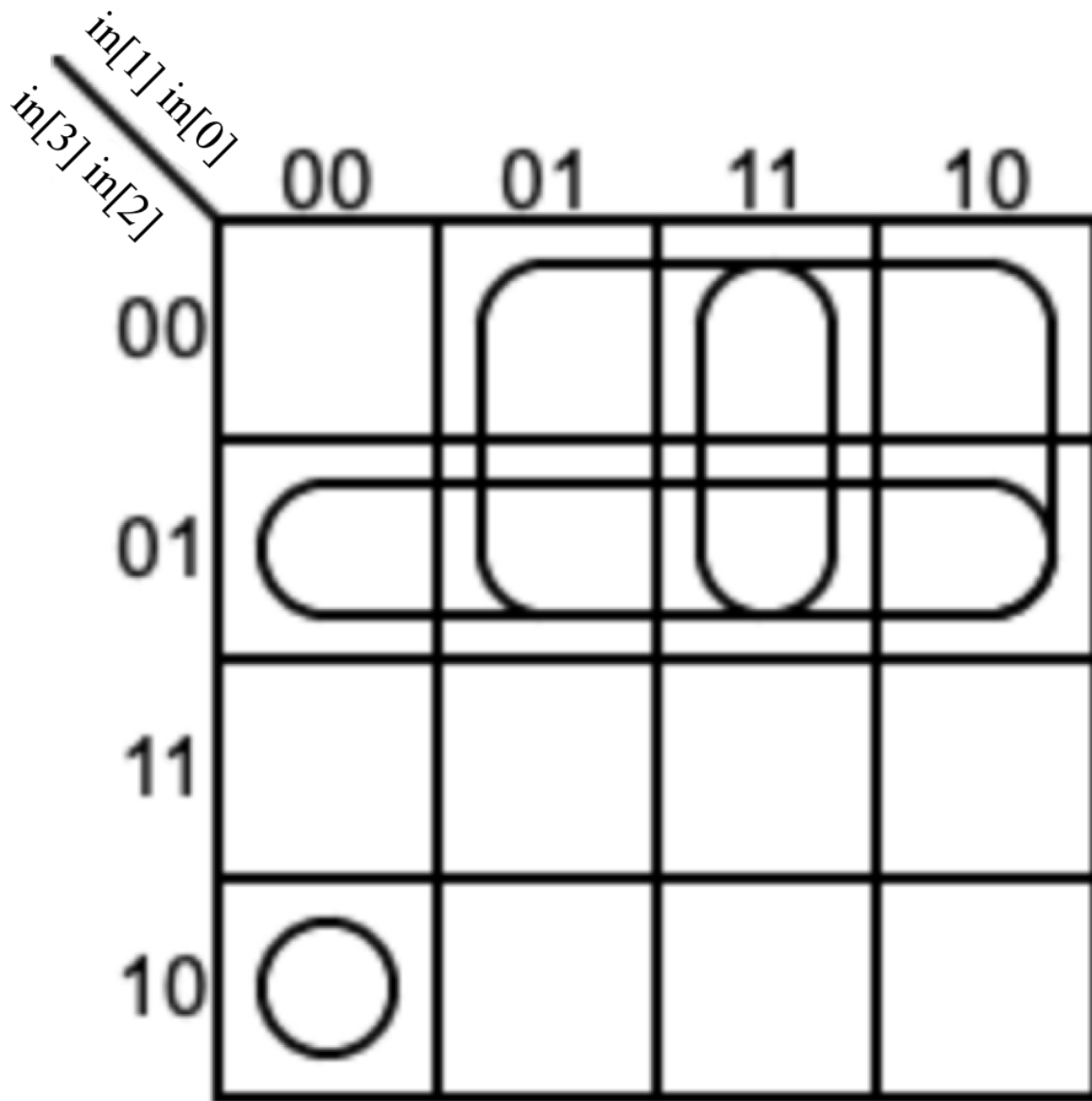
12111448 侯芳旻

### Part 1

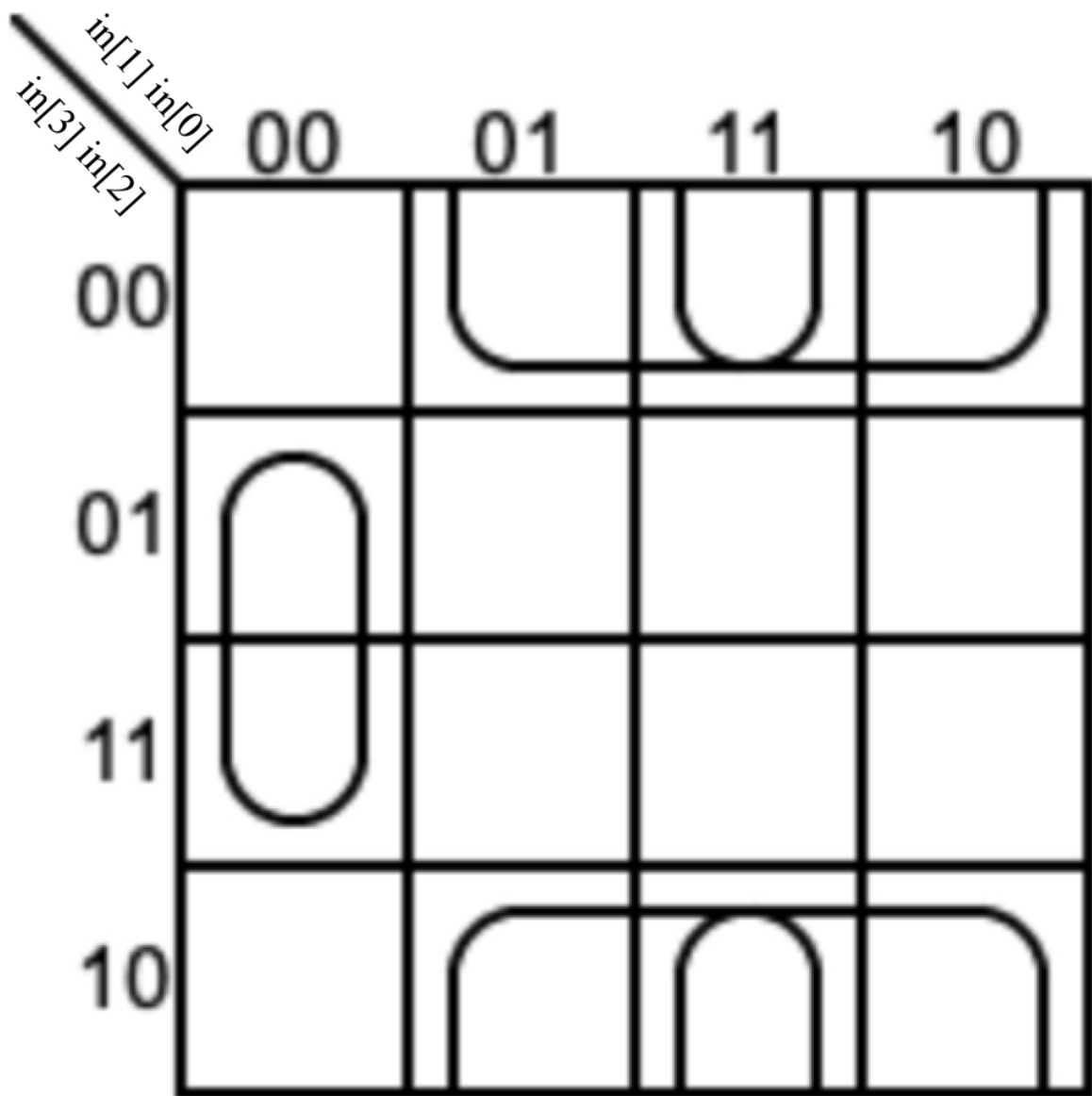
#### Question 1

- (10 points) Design a four-bit 2's complemener with only OR and XOR gates. Draw the logic diagram.  
补充说明:需要写出真值表。每个输出的表达式需要化简。 电路限定用异或和或门，可以用反相器。逻辑电路图需要标注输入/输出。

in[3]	in[2]	in[1]	in[0]	out[3]	out[2]	out[1]	out[0]
0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1
0	0	1	0	1	1	1	0
0	0	1	1	1	1	0	1
0	1	0	0	1	1	0	0
0	1	0	1	1	0	1	1
0	1	1	0	1	0	1	0
0	1	1	1	1	0	0	1
1	0	0	0	1	0	0	0
1	0	0	1	0	1	1	1
1	0	1	0	0	1	1	0
1	0	1	1	0	1	0	1
1	1	0	0	0	1	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	0	1	0
1	1	1	1	0	0	0	1



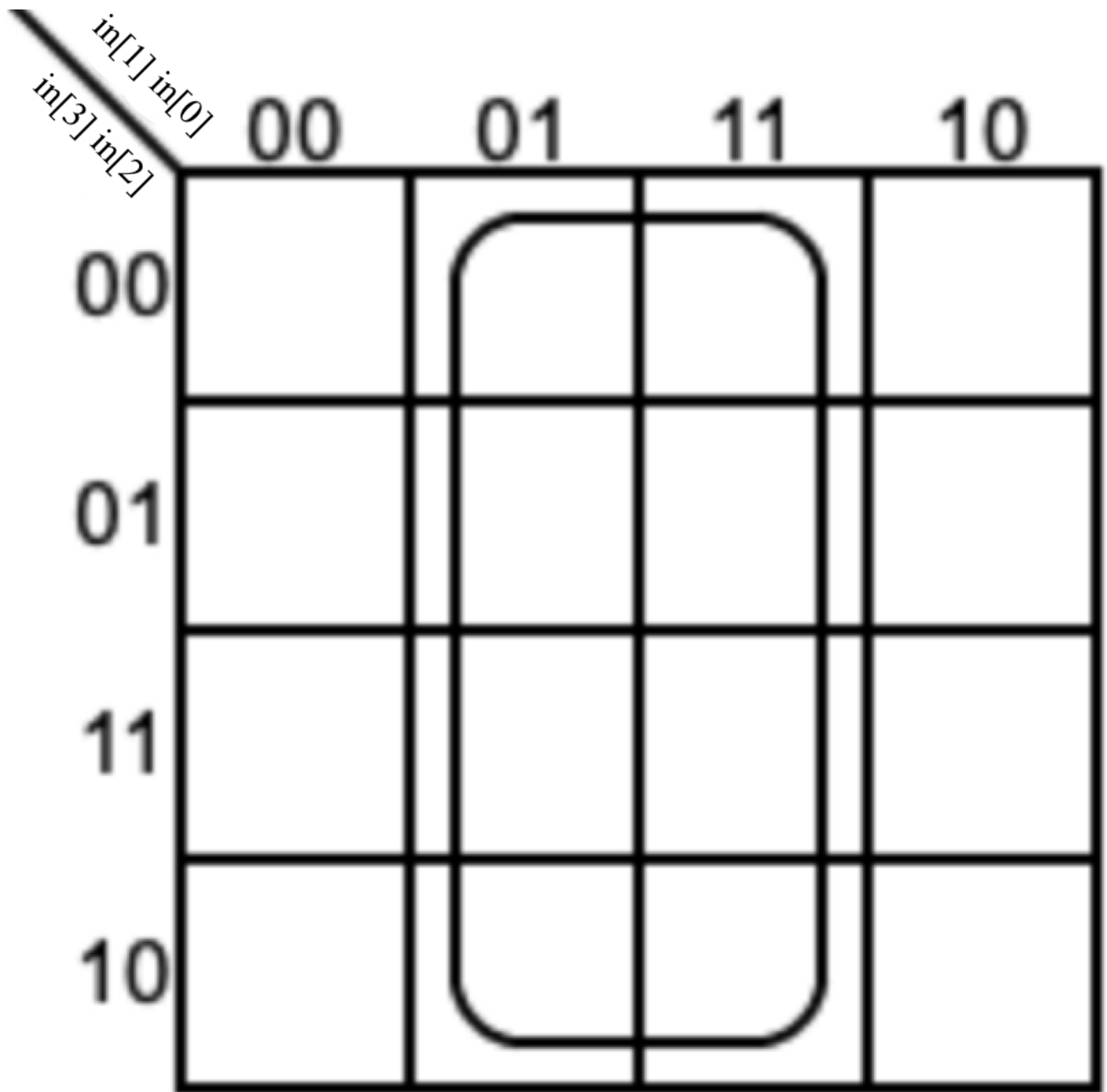
$$\begin{aligned}
 out[3] &= in[3]' * in[2] + in[3]' * in[1] + in[3]' * in[0] + in[3] * in[2]' * in[1]' * in[0]' \\
 &= in[3]' * (in[2] + in[1] + in[0]) + in[3] * (in[2] + in[1] + in[0])'
 \end{aligned}$$



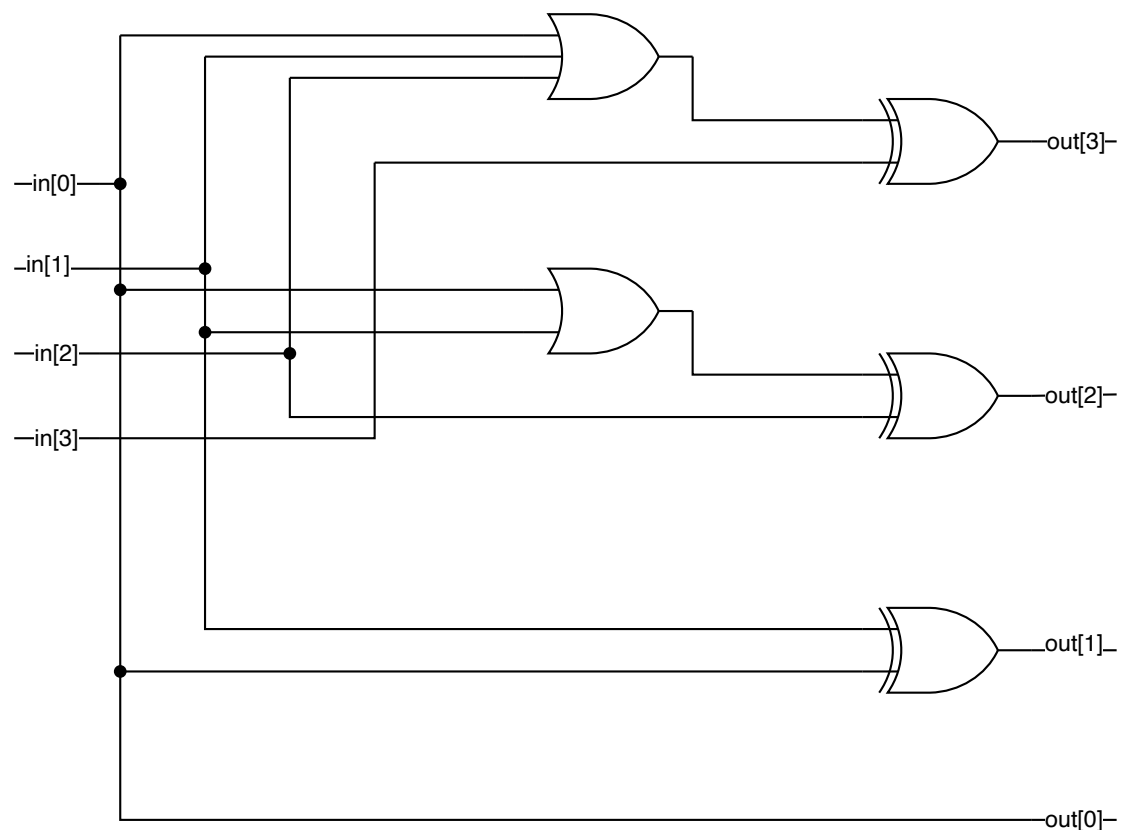
$$\begin{aligned}
 out[2] &= in[2] * in[1]' * in[0]' + in[2]' * in[0] + in[2]' * in[1] \\
 &= in[2] * (in[1] + in[0])' + in[2]' * (in[1] + in[0])
 \end{aligned}$$

$in[1] in[0]$ $in[3] in[2]$		00	01	11	10
00					
01					
11					
10					

$$out[1] = in[1] * in[0]' + in[0] * in[1]'$$



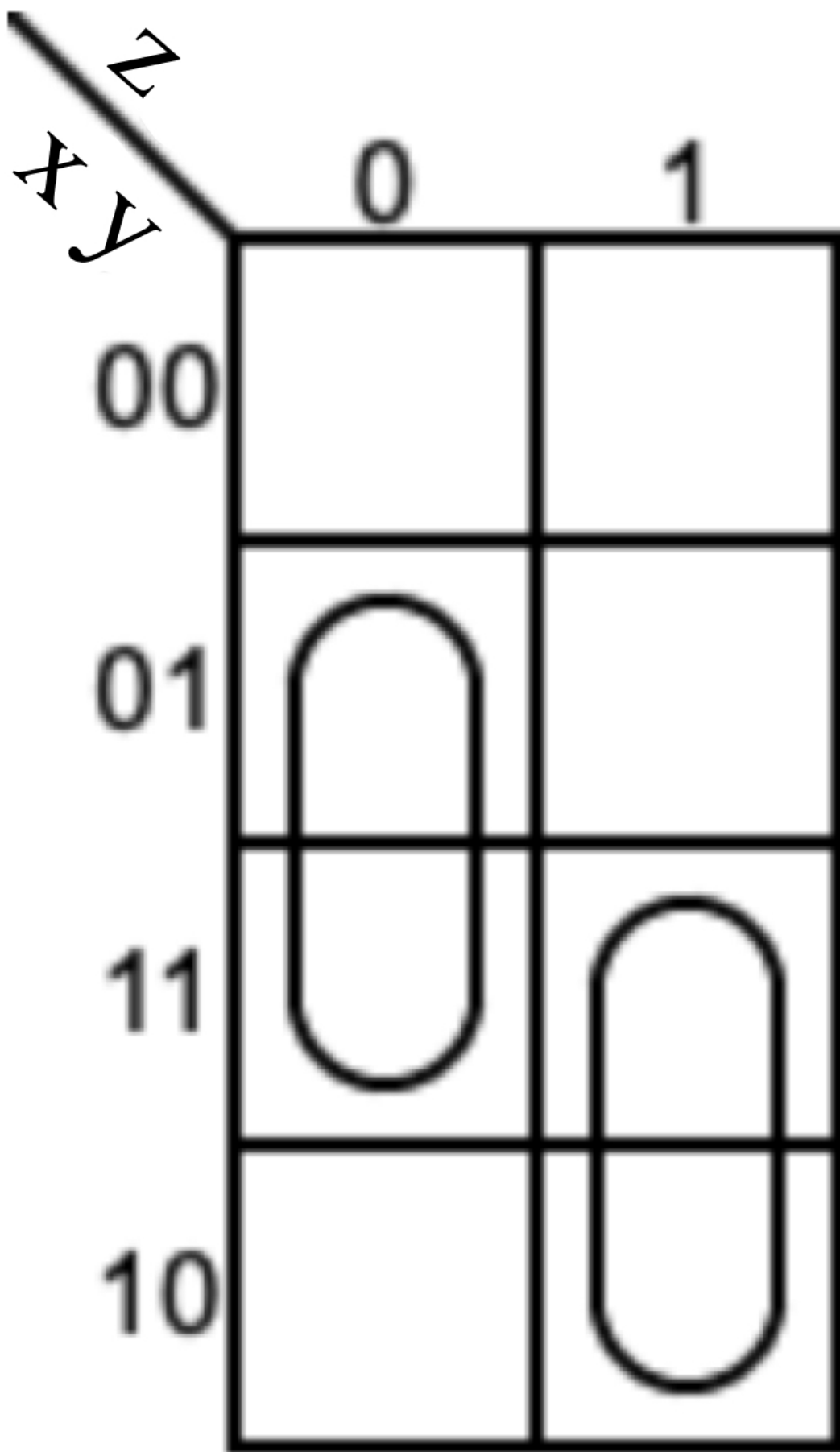
$$out[0] = in[0]$$



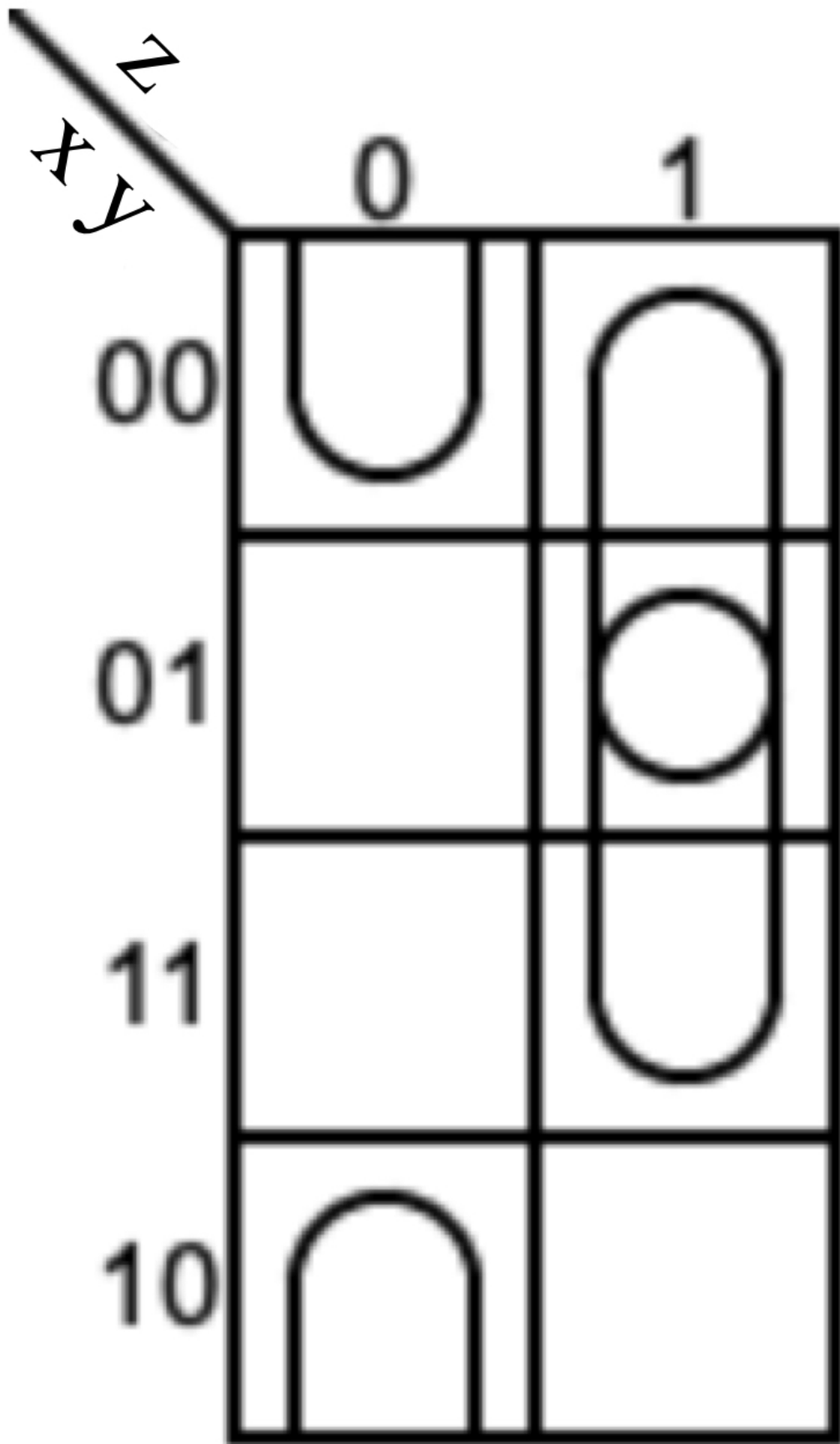
## Question2

- (20 points) Design a combinational circuit with three inputs,  $x$ ,  $y$ , and  $z$ , and three outputs,  $A$ ,  $B$ , and  $C$ . When the binary input is 0, 1, or 2, the binary output is two greater than the input. When the binary input is 3, 4, 5, 6, or 7, the binary output is one less than the input. Draw the logic diagram. 补充说明:输入从最高位到最低位是 $x$ ,  $y$ ,  $z$ ;输出从最高位到最低位是 $A$ ,  $B$ ,  $C$ 。需要写出真值表。每个输出的表达式需要化简(但不一定是最简)。电路没有限定门电路类型,但必须是组合电路。逻辑电路图需要标注输入/输出。不能对于每个输出单独画一个完全独立的电路,需要是一个电路整体。

$x$	$y$	$z$	$A$	$B$	$C$
0	0	0	0	1	0
0	0	1	0	1	1
0	1	0	1	0	0
0	1	1	0	1	0
1	0	0	0	1	1
1	0	1	1	0	0
1	1	0	1	0	1
1	1	1	1	1	0





$$A = yz' + xz$$

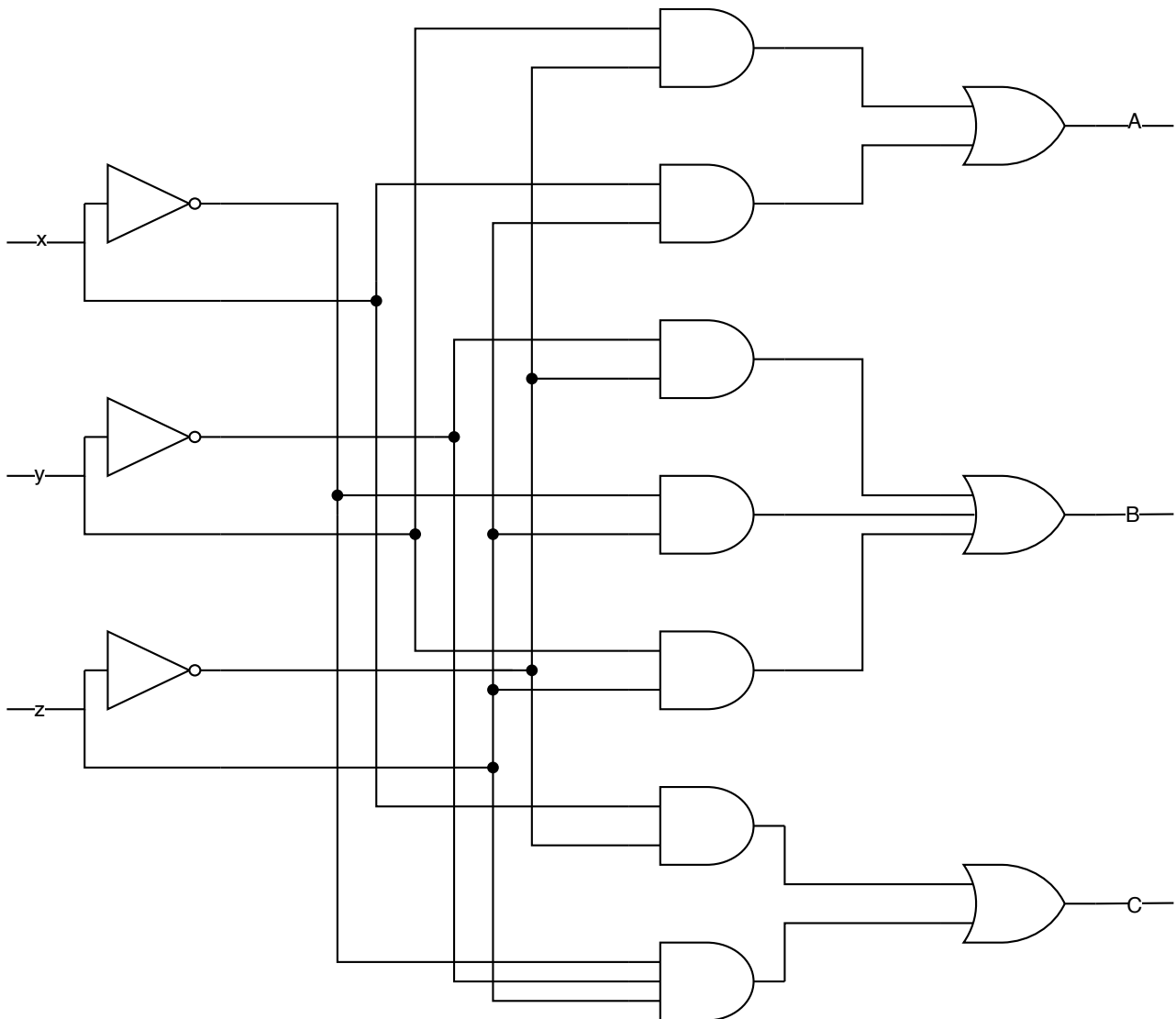


$$B = y'z' + x'z + yz$$



<div> <div> <div></div> <div></div> </div> <div> <div></div> <div></div> </div> </div>		0	1
00			
01			
11			
10			

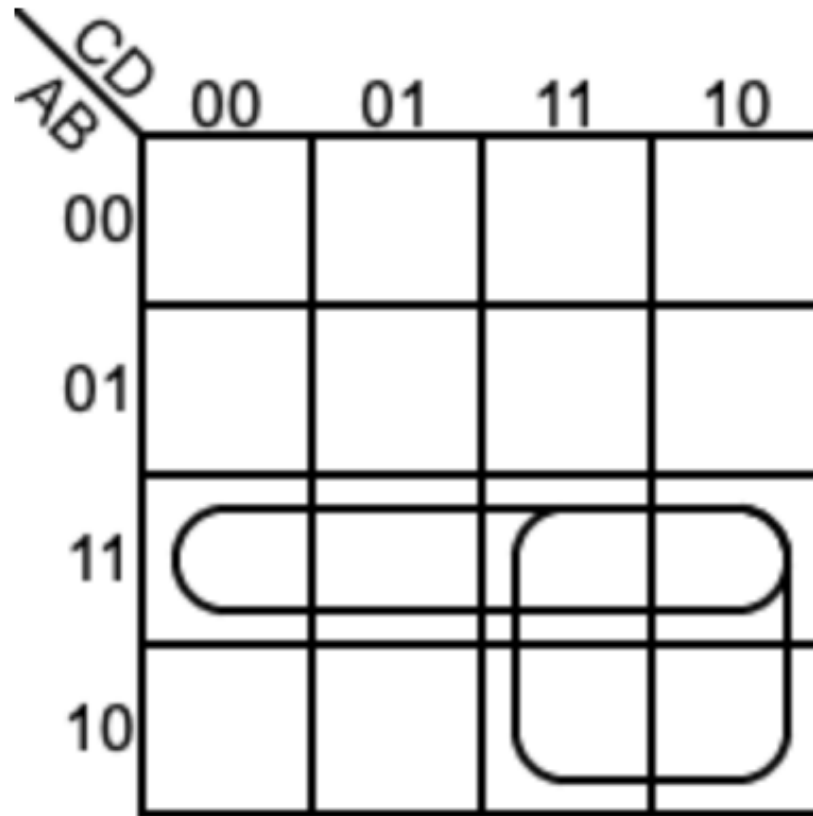
$$C = x * z' + x' * y' * z$$



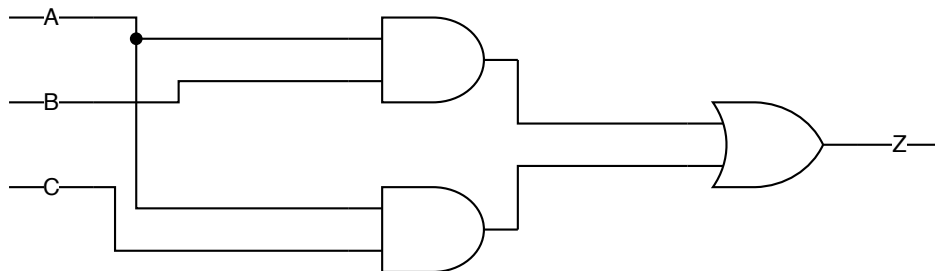
### Question3

- (20 points) A combinational circuit has four inputs A, B, C, and D, and one output Z. The output is 0 if the input combination is a valid BCD coded decimal digit. Otherwise, the output is 1. Draw the logic diagram. 补充说明:从最高位到最低位是A, B, C, D, 输出为Z。需要写出真值表。每个输出的表达式需要化简。电路没有限定门电路类型,但必须是组合电路。逻辑电路图需要标注输入/输出。不能对于每个输出单独画一个完全独立的电路,需要是一个电路整体。

A	B	C	D	Z
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1



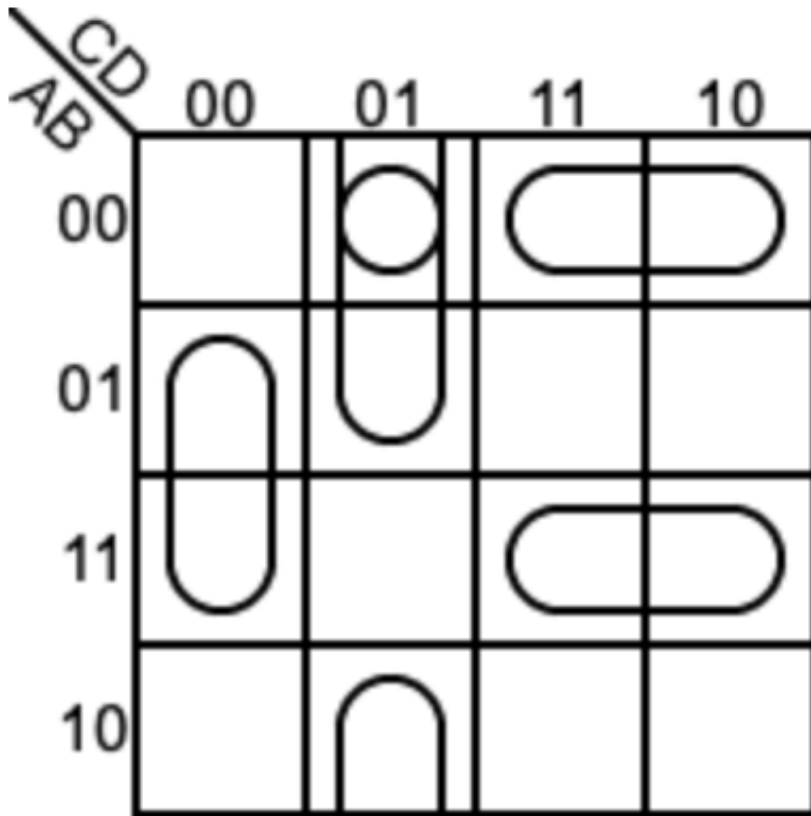
$$Z = AB + AC$$



## Question4

- (20 points) An 8-to-1 MUX has inputs A, B, and C connected to selection lines S2, S1, and S0, respectively. The data inputs I0 to I7 are connected as I1 = I2 = I7 = 1, I3 = I5 = 0, I0 = I4 = D, and I6 = D'. Determine the Boolean expression of the MUX output. 补充说明:从最高位到最低位是A, B, C, 分别连接 S2, S1, S0。输出的表达式需要化简。

A	B	C	output
0	0	0	D
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	D
1	0	1	0
1	1	0	D'
1	1	1	1



$$\begin{aligned} output &= (A'B'C' + AB'C')D + (A'B'C + A'BC' + ABC) + ABC'D' \\ &= BC'D' + ABC + A'C'D + A'B'C + B'C'D \end{aligned}$$

## Question5

- (20points) Implement the Boolean function  $F(A,B,C,D) = \sum(1,2,4,9,12,13,15)$  using
  1. decoder and external gates, and
  2. 8-to-1 MUX and external gates. Draw the logic diagram. 补充说明: Decoder 使用 74154, 需要标 74154 的输入和输出引脚。8-to-1 MUX 需要标输入和输出引脚, 需要解题过程。

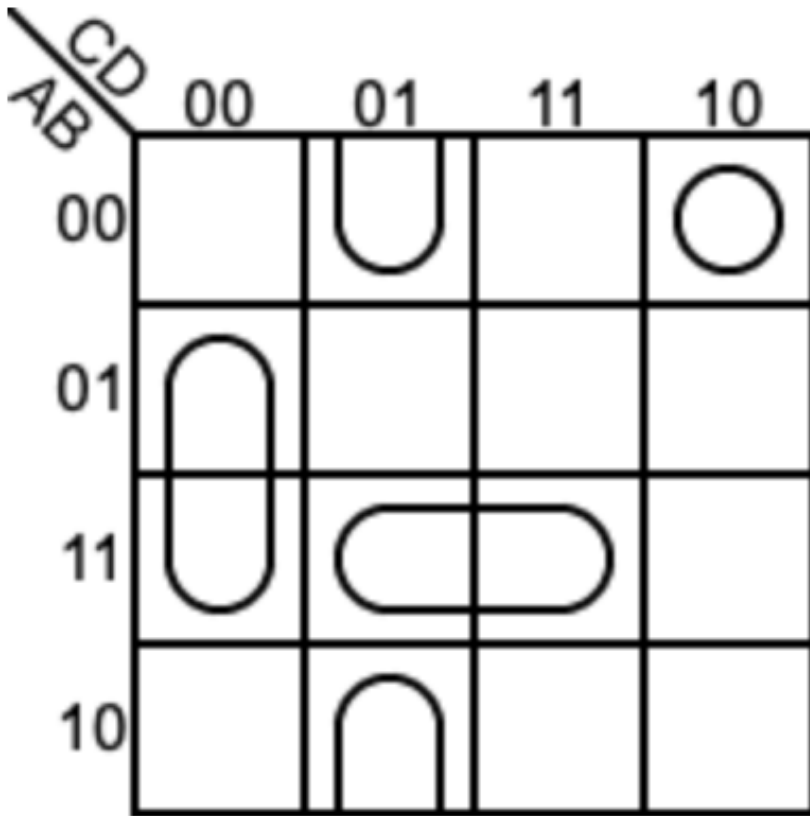
A	B	C	D	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

} D

} D'

} D

} D



$$F = BC'D' + ABD + B'C'D + A'B'CD'$$

$$A = 0 \ B = 0 \ C = 0 \ F = D$$

$$A = 0 \ B = 0 \ C = 1 \ F = D'$$

$$A = 0 \ B = 1 \ C = 0 \ F = D'$$

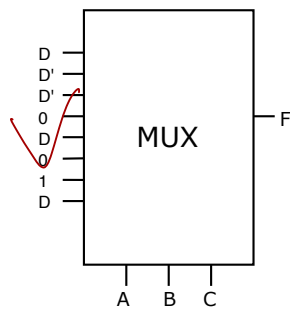
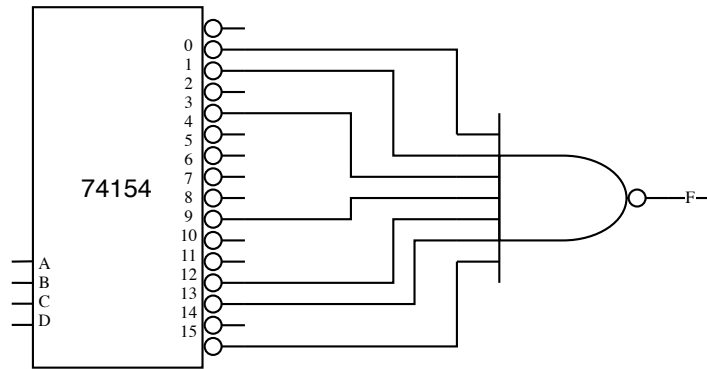
$$A = 0 \ B = 1 \ C = 1 \ F = 0$$

$$A = 1 \ B = 0 \ C = 0 \ F = D$$

$$A = 1 \ B = 0 \ C = 1 \ F = 0$$

$$A = 1 \ B = 1 \ C = 0 \ F = 1$$

$$A = 1 \ B = 1 \ C = 1 \ F = D$$



## Question6

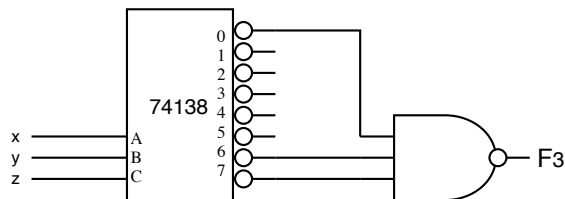
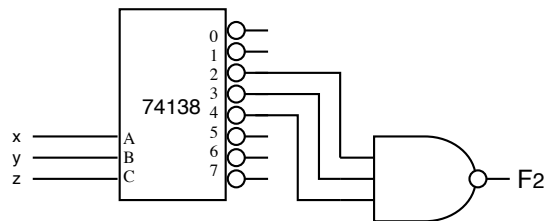
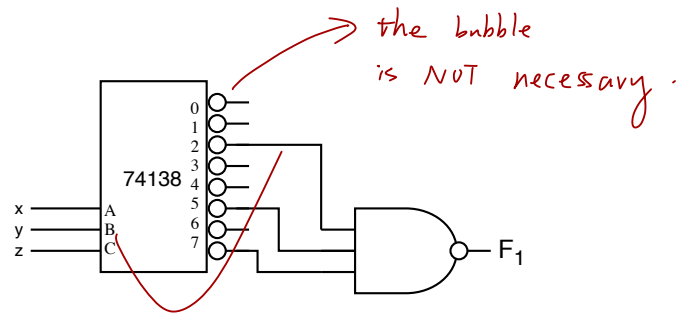
- (10 points) Using decoders and external gates, design and draw three combinational circuits defined by the following three Boolean functions:  $F_1 = x'yz' + xz$ ,  $F_2 = xy'z' + x'y$ ,  $F_3 = x'y'z' + xy$ . 补充说明: Decoder使用74154, 需要标74154的输入和输出引脚。需要解题过程。

$$F_1 = x'yz' + xz(y + y') = x'yz' + \overset{4}{x} \overset{2}{y} \overset{1}{z'} + xy'z = \sum(2, 5, 7)$$

$$F_2 = xy'z' + \overset{2}{x'y} = xy'z' + x'y(z + z') = xy'z' + x'yz + x'y'z' = \sum(2, \overset{3}{3}, 4)$$

$$F_3 = x'y'z' + xy = x'y'z' + xy(z + z') = x'y'z' + xyz + xyz' = \sum(0, 6, 7)$$





## Part2

**My simulation result may look different from others.**

As I am a MacOS user, it is impossible for me to use Vivado. So I used an online platform called EDA playground to get the simulation results for convenience. And I have got the permission of TA.

### task1

**Design in structure descriptions styles by using xor and nand gates respectively.**

- xor gates

```

module a2t1_xor (input [3:0]x,
                output y);
wire y2,y1,y3;
xor xor1(y1,x[3],x[2]);
xor xor2(y2,x[1],x[0]);
xor xor3(y3,y1,y2);
assign y = y3;
// assign = continual assignment to wire outside an always statement.
// value of LHS is updated when RHS changes.
endmodule

```

- nand gates

```

module a2t1_nand (input [3:0]x,
                output y);

wire [8:0] y1;
wire y2;
wire [3:0]nx;

not nn0(nx[0],x[0]);
not nn1(nx[1],x[1]);
not nn2(nx[2],x[2]);
not nn3(nx[3],x[3]);
// the ^ in primitive gates will not be calculated
// although you use a ^, the compiler will not calculate it
// so the outpt of n1-n8 are the same

nand n1(y1[1],nx[3],nx[2],nx[1],x[0]);
nand n2(y1[2],nx[3],nx[2],x[1],nx[0]);
nand n3(y1[3],nx[3],x[2],nx[1],nx[0]);
nand n4(y1[4],nx[3],x[2],x[1],x[0]);
nand n5(y1[5],x[3],nx[2],nx[1],nx[0]);
nand n6(y1[6],x[3],nx[2],x[1],x[0]);
nand n7(y1[7],x[3],x[2],nx[1],x[0]);
nand n8(y1[8],x[3],x[2],x[1],nx[0]);
nand n9(y2,y1[1],y1[2],y1[3],y1[4],y1[5],y1[6],y1[7],y1[8]);

assign y = y2;

endmodule

```

## Test bench in Verilog, simulation result and its description

- test bench

```
// Code your testbench here
// or browse Examples
module a2t1_sim (

);

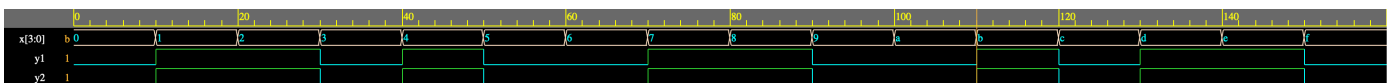
reg [3:0]x = 4'b0000;
wire y1,y2;

// forget to initialize modules
a2t1_nand sim1(x,y1);
a2t1_xor sim2(x,y2);
// nand and xor are like the reserved words
// so you cannot name a module as it
// and the purple color also reminds you of it

// you cannot let y1 and y2 be the same wire y
// as the variables shown in the EPWave are the variables
// that was in the testbench
// if you let y connect to bothe sim1 and sim2,
// it will go wrong

initial begin
    repeat(15) #10 x = x+1;
    #10 $finish();
end
endmodule
```

- simulation result



- description

As x is the input, y1 is the output of design using nand gates, y2 is the output of design using xor gates, judging from the simulation result, it is clear to see that when the number of "1" in x is odd, the output is 1; when it is even, the output is 0. So the design meets the requirement.

## task2

## Design in structure descriptions styles by using primitive gates on Sum-of-Minterms and Product-of-Maxterms respectively.(5\*2 marks)

- Sum-of-Minterms

```
module a2t2_somin (input [3:0]x,
                  output y);

    wire [9:0]min;
    wire [3:0]xn;
    wire ywire;

    not n0(xn[0],x[0]);
    not n1(xn[1],x[1]);
    not n2(xn[2],x[2]);
    not n3(xn[3],x[3]);

    and a0(min[0],xn[3],xn[2],xn[1],xn[0]);
    and a1(min[1],xn[3],xn[2],xn[1],x[0]);
    and a2(min[2],xn[3],xn[2],x[1],xn[0]);
    and a3(min[3],xn[3],xn[2],x[1],x[0]);
    and a4(min[4],xn[3],x[2],xn[1],xn[0]);
    and a5(min[5],xn[3],x[2],xn[1],x[0]);
    and a6(min[6],xn[3],x[2],x[1],xn[0]);
    and a7(min[7],xn[3],x[2],x[1],x[0]);
    and a8(min[8],x[3],xn[2],xn[1],xn[0]);
    and a9(min[9],x[3],xn[2],xn[1],x[0]);

    or o0(ywire,min[0],min[1],min[2],min[3],min[4],min[5],min[6],min[7],min[8],min[9]);

    assign y = ywire;
    // do not forget to assign

endmodule
```

- Product-of-Maxterms

```
module a2t2_pomax(input [3:0]x,
                  output y);

    /*
    ^x[3]+^x[2]+x[1]+x[0]
    ^x[3]+^x[2]+x[1]+^x[0]
    ^x[3]+^x[2]+^x[1]+^x[0]
    ^x[3]+^x[2]+^x[1]+x[0]
    ^x[3]+x[2]+^x[1]+^x[0]
    ^x[3]+x[2]+^x[1]+x[0]
    */

    wire [5:0]max;
```

```

wire [3:0]xn;
wire ywire;

not n0(xn[0],x[0]);
not n1(xn[1],x[1]);
not n2(xn[2],x[2]);
not n3(xn[3],x[3]);

or o0(max[0],xn[3],xn[2],x[1],x[0]);
or o1(max[1],xn[3],xn[2],x[1],xn[0]);
or o2(max[2],xn[3],xn[2],xn[1],xn[0]);
or o3(max[3],xn[3],xn[2],xn[1],x[0]);
or o4(max[4],xn[3],x[2],xn[1],xn[0]);
or o5(max[5],xn[3],x[2],xn[1],x[0]);

and a0(ywire,max[0],max[1],max[2],max[3],max[4],max[5]);

assign y = ywire;

endmodule

```

## Test bench in Verilog, simulation result and its description

- test bench

```

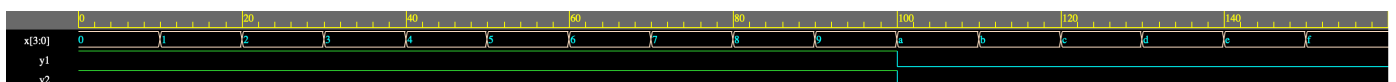
module a2t2_sim();
    reg [3:0]x = 4'b0000;
    wire y1,y2;

    a2t2_pomax sim2(x,y2);
    a2t2_somin sim1(x,y1);

    initial
    begin
        repeat(15) #10 x = x+1;
        #10 $finish();
    end
endmodule

```

- simulation result



- description

x is the input, y1 is the output of design by product of maxterm and y2 is the output of design by sum of minterm. As the simulation result shows, when x is valid BCD code ( $x \leq 9$ ), the output is 1, otherwise the output is 0. So the design meets the requirement.

## task3

### Design in Verilog in data flow style and behavioral description style

- data flow

```
module a2t3_df(input [3:0]x,
              output [3:0] y);

assign y = 5'b01111-x+5'b00001;

endmodule
```

- behavioral description

```
module a2t3_bd(input [3:0]x,
              output [3:0]y);
    // using behaviour description styles

    reg [3:0]ywire;
    // it is illegal to use y on the left side in the initial
    // and as y is more than 1 bit, when defining it
    //do not forget to declare ist length
    always @(*) begin
        ywire = 5'b10000-x;
    end

    assign y = ywire;
endmodule
```

### Test bench in Verilog, simulation result and its description

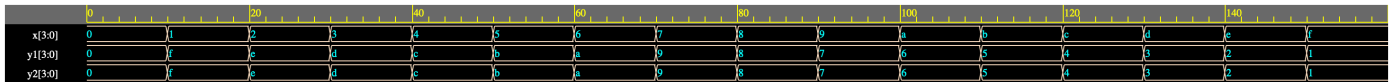
- test bench

```
module a2t3_sim();
    reg [3:0]x = 4'b0000;
    wire [3:0]y1;
    wire [3:0]y2;

    a2t3_df sim1(x,y1);
    a2t3_bd sim2(x,y2);

    initial begin
        repeat(15) #10 x=x+1;
        #10 $finish();
    end
endmodule
```

- simulation result



- description

x is the input, y1 is the output of design using data flow style and y2 is the output using behavioral description. As we can see from the simulation result, the output is the 2' complement of input. So the design meets the demand.

## task4

### Design the Decoder(74139) and 16-to-1 MUX which would be used in the circuit

- 74139

```
module d74139(input ne,
              input  [1:0]x,
              output [3:0]y);
    reg [3:0] yreg;
    // do not assign y in the always block

    always @(*)
    begin
        if (ne)
            yreg = 4'hf;
        else
            case(x)
                2'b00: yreg = 4'b1110;
                2'b01: yreg = 4'b1101;
                2'b10: yreg = 4'b1011;
                2'b11: yreg = 4'b0111;
            endcase
        end
        assign y = yreg;
    endmodule
```

- 16-to-1 MUX

```
module mux16to1(input [3:0]sel,
                input  [15:0]x,
                output y);
    reg regy;
    // variables that are assigned in initial or always should be reg type
    // it is not okay to assign outputs in initial or always
    always @(*) begin
        case(sel)
            4'h0: regy = x[0];
            4'h1: regy = x[1];
```

```

        4'h2: regy = x[2];
        4'h3: regy = x[3];
        4'h4: regy = x[4];
        4'h5: regy = x[5];
        4'h6: regy = x[6];
        4'h7: regy = x[7];
        4'h8: regy = x[8];
        4'h9: regy = x[9];
        4'ha: regy = x[10];
        4'hb: regy = x[11];
        4'hc: regy = x[12];
        4'hd: regy = x[13];
        4'he: regy = x[14];
        4'hf: regy = x[15];
    endcase
    // case should be end with "endcase" statement
end

    assign y = regy;
endmodule

```

## Design the circuit in structure description style by using the Decoder(74139) and 16-to-1 MUX

- using 74139

```

module a2t4_dc(input [4:0]x,
               output y);

    reg [3:0]y1,y2,y3;
    reg a1,a2,a3,a4,a5;
    reg [3:0]ny1,ny2,ny3;
    reg regy;

    d74139 d1(1'b0,{x[4],x[3]},y1);
    d74139 d2(1'b0,{x[2],x[1]},y2);
    d74139 d3(1'b0,{x[0],1'b0},y3);

    not n10(ny1[0],y1[0]);
    not n11(ny1[1],y1[1]);
    not n12(ny1[2],y1[2]);
    not n20(ny2[0],y2[0]);
    not n21(ny2[1],y2[1]);
    not n22(ny2[2],y2[2]);
    not n32(ny3[2],y3[2]);
    not n30(ny3[0],y3[0]);

    and a1(a1,y1[3],y1[2],y1[1],ny1[0],y2[3],y2[2],y2[1],ny2[0],y3[3],ny3[2],y3[1],y3[0]);

```



```

and a2(a2,y1[3],y1[2],y1[1],ny1[0],y2[3],y2[2],ny2[1],y2[0],y3[3],y3[2],y3[1],ny3[0]);
and a3(a3,y1[3],y1[2],y1[1],ny1[0],y2[3],ny2[2],y2[1],y2[0],y3[3],y3[2],y3[1],ny3[0]);
and a4(a4,y1[3],y1[2],ny1[1],y1[0],y2[3],y2[2],y2[1],ny2[0],y3[3],y3[2],y3[1],ny3[0]);
and a5(a5,y1[3],ny1[2],y1[1],y1[0],y2[3],y2[2],y2[1],ny2[0],y3[3],y3[2],y3[1],ny3[0]);

or o1(regy,a1,a2,a3,a4,a5);

assign y = regy;
endmodule

```

- using 16-to-1 MUX

```

module a2t4_mux(input [4:0]x,
               output y);

wire x0n;
wire regy;

not (x0n,x[0]);

mux16to1 mux({x[4],x[3],x[2],x[1]},
{1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,x0n,1'b0,1'b0,1'b0,x0n,1'b0,x0n,x0n,x[0]},
regy);

// 1'b0 instead of 0
// the left side in the {} represent more significant bits
assign y = regy;

endmodule

```

## Test bench to verify the function of the Decoder(74139) and 16-to-1 MUX in Verilog, simulation result and its description

- test bench

```

module decoder_mux_sim();

reg ne=1'b0;
reg [1:0]dx = 2'b00;
wire [3:0]dy;
reg [3:0]sel = 4'h0;
reg [15:0]mx = 16'b0000_0000_0000_0001;
wire my;

mux16to1 mux(sel,mx,my);
d74139 de(ne,dx,dy);
// do not forget semicolon

```

```

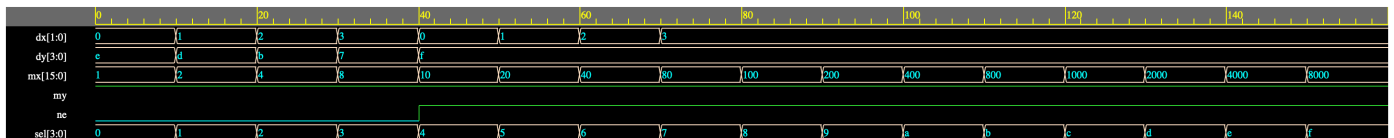
initial begin
    repeat(4) #10 dx=dx+1;
    ne=ne+1;
    repeat(3) #10 dx=dx+1;
end

initial begin
    repeat(15) #10
    begin
        sel=sel+1;
        mx=mx*2;
    end
    #10 $finish();
end
endmodule

// when designing a module, do not forget to end it

```

- simulation result



- description As dx, ne is the input of decoder(74139), dy is the output of decoder(74139), when ne=1, the output is always 4'hf, and when ne=0, the output changes as input x changes, which meets the demand of design. As sel, mx is the input of 16-to-1 MUX, my is the output, judging from the simulation result, it is clear that the design meets the demand.

## Test bench to verify the function of the circuit, simulation result and its description

- test bench

```

module a2t4_sim ();

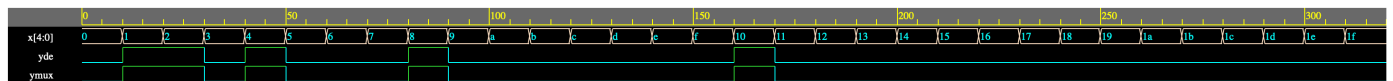
    reg [4:0] x = 5'b0;
    wire ymux,yde;

    a2t4_dc dc(x,yde);
    a2t4_mux mux(x,ymux);

    initial begin
        repeat (31) #10 x = x+1;
        #10 $finish();
    end
endmodule

```

- simulation result



- description

x is the input, yde is the output of circuit using decoder, ymux is the circuit using MUX. When x is one-hot code, both output is 1, otherwise they are both 0, which meets the demand.