Logic Design Lab 1

Experiment 1

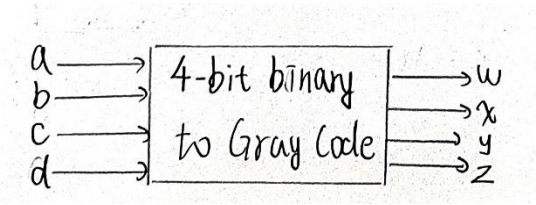
Design and verify a 4-bit binary-to-Gray-code converter for a Gray code sequence with 10 code words (input: abcd, output: wxyz, a and w are the MSB).

Design Specification：

Input: a,b,c,d.

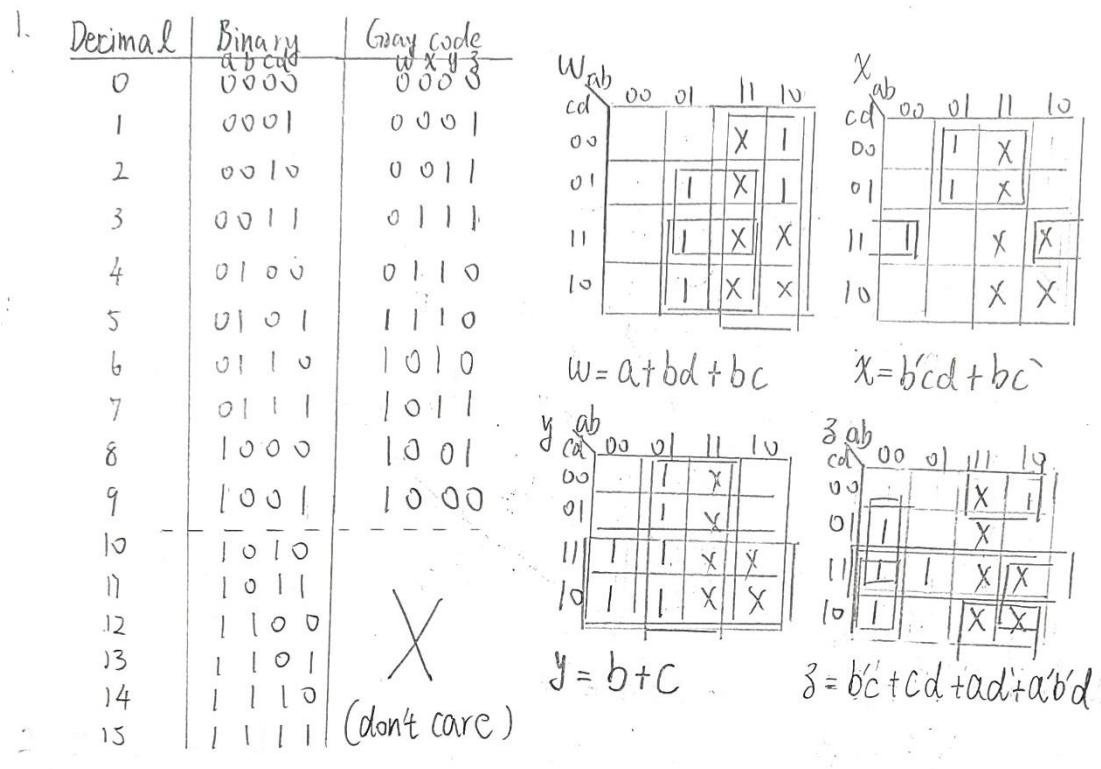
Output: w,x,y,z.

Block diagram:

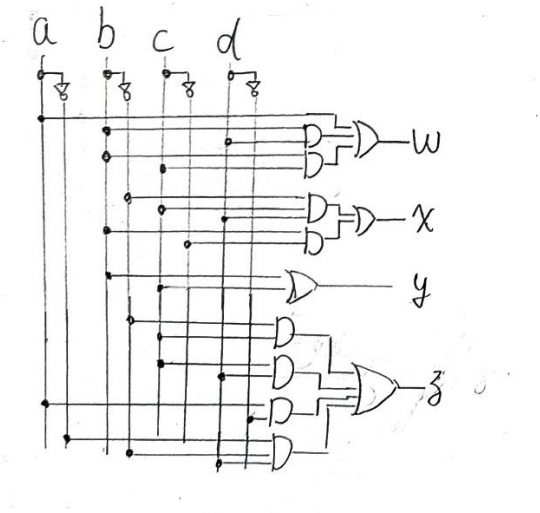


Design Implementation：

Logic equation:



Logic diagram:



Verilog code:

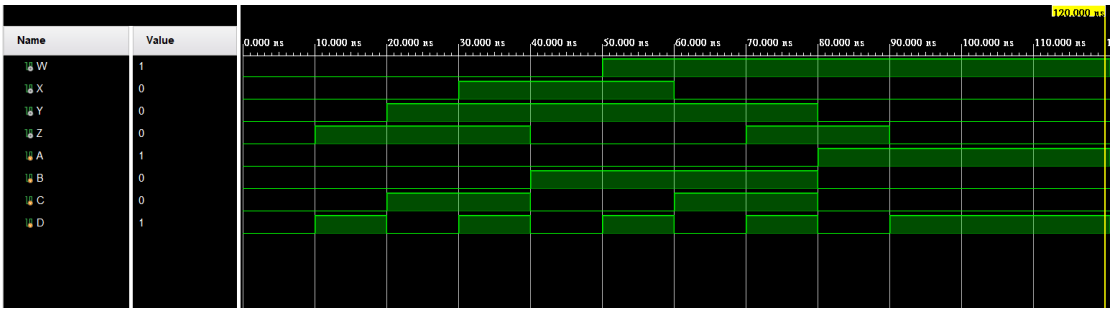
Source code

```
module experiment1(
    input a,
    input b,
    input c,
    input d,
    output w,
    output x,
    output y,
    output z
    );
    assign w= a | b&d | b&c;
    assign x= (~b)&c&d | b&(~c);
    assign y= b | c;
    assign z= c&d | (~a)&(~b)&d | (~a)&(~b)&c | a&(~c)&(~d);
endmodule
```

Test bench

```
module test();
 wire W,X,Y,Z;
reg A,B,C,D;
experiment1 U0(.a(A),.b(B),.c(C),.d(D),.w(W),.x(X),.y(Y),.z(Z));

initial
begin

A=0;B=0;C=0;D=0;
#10 A=0;B=0;C=0;D=1;
#10 A=0;B=0;C=1;D=0;
#10 A=0;B=0;C=1;D=1;
#10 A=0;B=1;C=0;D=0;
#10 A=0;B=1;C=0;D=1;
#10 A=0;B=1;C=1;D=0;
#10 A=0;B=1;C=1;D=1;
#10 A=1;B=0;C=0;D=0;
#10 A=1;B=0;C=0;D=1;

end
endmodule
```

Simulation waveform:



Discussion：

1. First of all, we need to clarify the meaning of this sentence:” a Gray code sequence with 10 code words”, which means that there are 10 codes of the sequence of this gray code. According to the website attached by TA on the platform eeclass, we can find out the gray code only with 10 codes (the first code and the last code still fit the property of the gray code).

2. However, there are $2^4 = 16$ codes in the typical 4-bit binary code, so we are encouraged to assign the remained 6 conditions as the don't care conditions to simplify the logic circuit to the simplest.

3. Next, from the truth table of this case, we use K-Map to gain the logic equations and the logic diagram. Then, we move on to write the Verilog code on Vivado and run the simulation to ensure that the logic circuit can function well.

Conclusion：

Because I'm the sophomore, the logic design course was token in my freshman semester. Definitely, I forgot some basic skills to construct the logic diagram and the standard operation process to deal with the problem. After finishing this experiment, I reconstruct the memory of how to use the K-Map method to simplify the logic equation and the fundamental skills to solve the problem.

Besides, when I took the logic design course, the lecturer didn't demonstrate how to write Verilog and introduce the coding style of Verilog so that the first experiment helps us to get familiar with Verilog and the software Vavido effectively.

Experiment 2

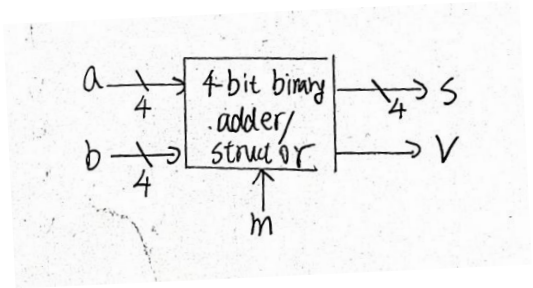
Design a signed 4-bit binary adder/subtractor with input a (a3a2a1a0), b (b3b2b1b0), m as the operator control (0 for addition and 1 for subtraction); output s (s3s2s1s0), v as overflow indicator.

Design Specification：

Input: a[3:0], b[3:0],m.

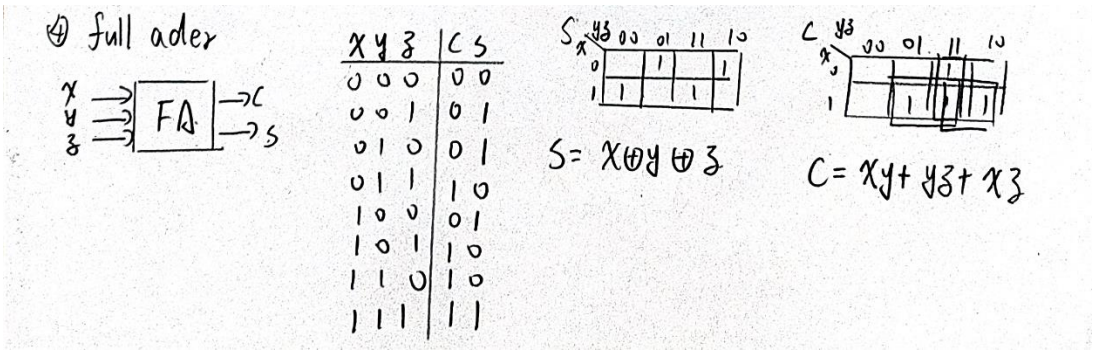
Output: s[3:0],v.

Block diagram:

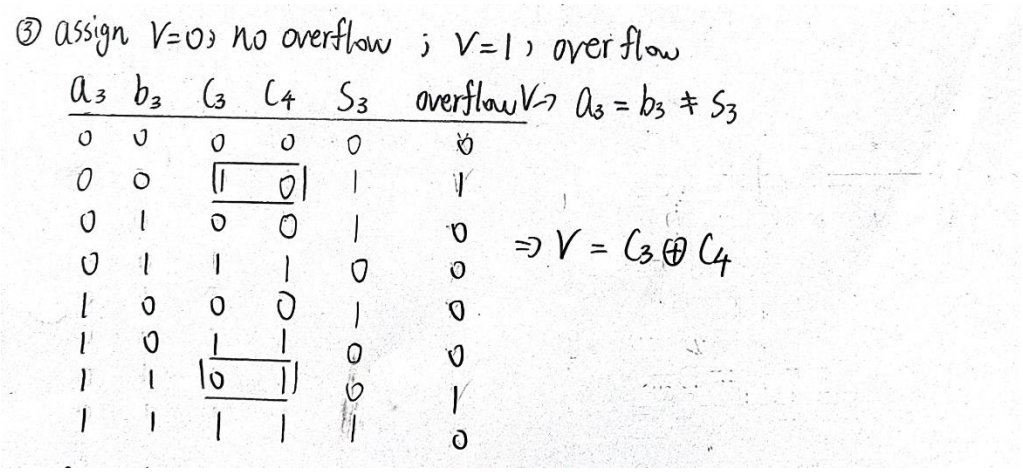


Design Implementation：

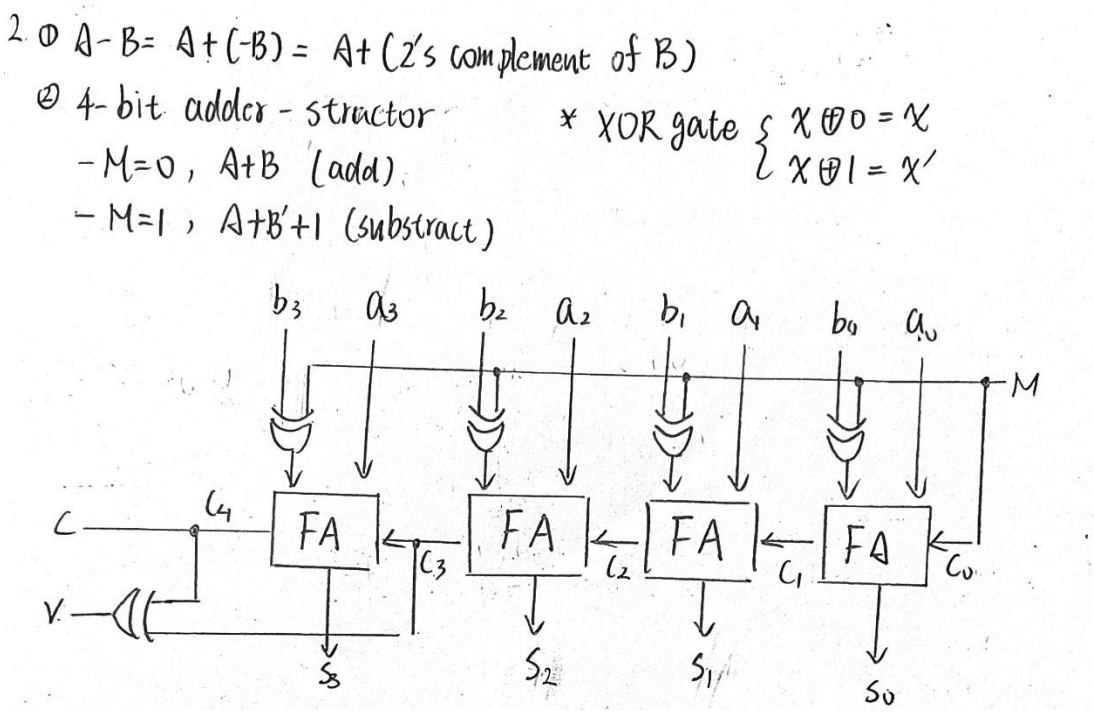
1. Logic equation and diagram for Full Adder：



2. Logic equation for overflow indicator：



3. Logic equation and diagram for the whole circuit



4. Verilog code:

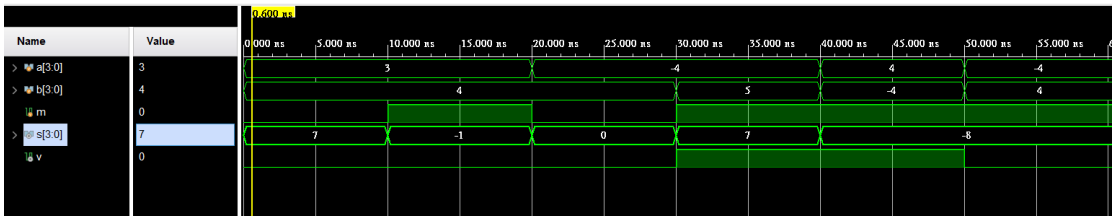
Source code

```
module lab1_2(
    input [3:0]a,
    input [3:0]b,
    output [3:0]s,
    input m,
    output v,
    wire [4:0]c
    );
    assign s[0]=(b[0]^m)^a[0]^c[0];
    assign s[1]=(b[1]^m)^a[1]^c[1];
    assign s[2]=(b[2]^m)^a[2]^c[2];
    assign s[3]=(b[3]^m)^a[3]^c[3];


    assign c[0]=m;
    assign c[1]=(a[0]& (b[0]^m)) | (a[0]& c[0]) | ((b[0]^m) & c[0]);
    assign c[2]=(a[1]& (b[1]^m)) | (a[1]& c[1]) | ((b[1]^m) & c[1]);
    assign c[3]=(a[2]& (b[2]^m)) | (a[2]& c[2]) | ((b[2]^m) & c[2]);
    assign c[4]=(a[3]& (b[3]^m)) | (a[3]& c[3]) | ((b[3]^m) & c[3]);

    assign v= c[3]^c[4];


endmodule
```

Test bench

```
module test_lab1_2;

    reg [3:0] a;
    reg [3:0] b;
    reg m;
    wire [3:0] s;
    wire v;

    lab1_2 U0(
        .s(s),
        .v(v),
        .a(a),
        .b(b),
        .m(m)
    );

    initial begin
            m = 1'b0; a = 4'sd3;  b = 4'sd4;     // s = 7, v = 0
        #10 m = 1'b1; a = 4'sd3;  b = 4'sd4;     // s = -1, v = 0
        #10 m = 1'b0; a = -4'sd4; b = 4'sd4;     // s = 0, v = 0
        #10 m = 1'b1; a = -4'sd4; b = 4'sd5;     // s = 7, v = 1
        #10 m = 1'b1; a = 4'sd4;  b = -4'sd4;    // s = -8, v = 1
        #10 m = 1'b1; a = -4'sd4; b = 4'sd4;     // s = -8, v = 0
    end

endmodule
```

5. Simulation waveform:



Discussion：

1. Full adder implementation：

   (1) “s” serves as the units digit. Therefore, when there is odd “1” in the 3 of the input (x, y, z), the output “s” is 1.

Thus, s = (x) XOR (y) XOR (z) = s ^ y ^ z.

(2) “c” serves as the carry out, Therefore, when there are two or above “1” in the 3 of the input (x, y, z), the output “c” is 1.
Thus, c = x · y + x · z + y · z.

2. Overflow indicator implementation：
   (1) From the class of Logic design, we have learned that whether the condition is overflow or not depends on the highest two bits (a_3 b_3) as well as their sum(s_3). If the condition is $a_3 = b_3 \neq s_3$, which means that a positive number plus a positive one but get the negative one and vice versa, so the result is overflow
   (2) However, through observing the truth table, we can get the overflow indicator “1” easily with different c_3 and c_4. Thus, the logic equation of the indicator is v = (c_3) XOR (c_4) = c_3 ^ c_4.

3. Finally, from the discussion above, we use block diagram to specify the logic diagram. Then, we move on to write the Verilog code on Vivado and run the simulation to ensure that the logic circuit can function well.

Conclusion：

In my opinion, a signed 4-bit binary adder/subtractor is a typical example to build a concept of how to deal with the combination circuit logically. In this experiment, I have learned how to use manager to switch the model of adding or subtracting and find out the input-output relation by observing the truth table in order to meet our goal.

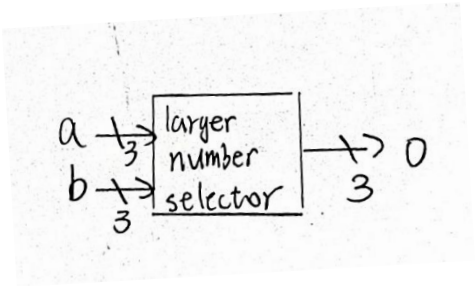
As for the signed numbers, it is a little bit confusing for me because as I have mentioned, I forgot a lot of the basic concept of the logic design. So, through this example, I read the handout of the logic design again to help me clarify the idea of various ways to indicate the signed numbers.

Experiment 3

(Bonus) For two 3-bit signed numbers a (a2a1a0) and b (b2b1b0), build a logic circuit to output o(o2o1o0) as the larger number and use a given testbench for verification.

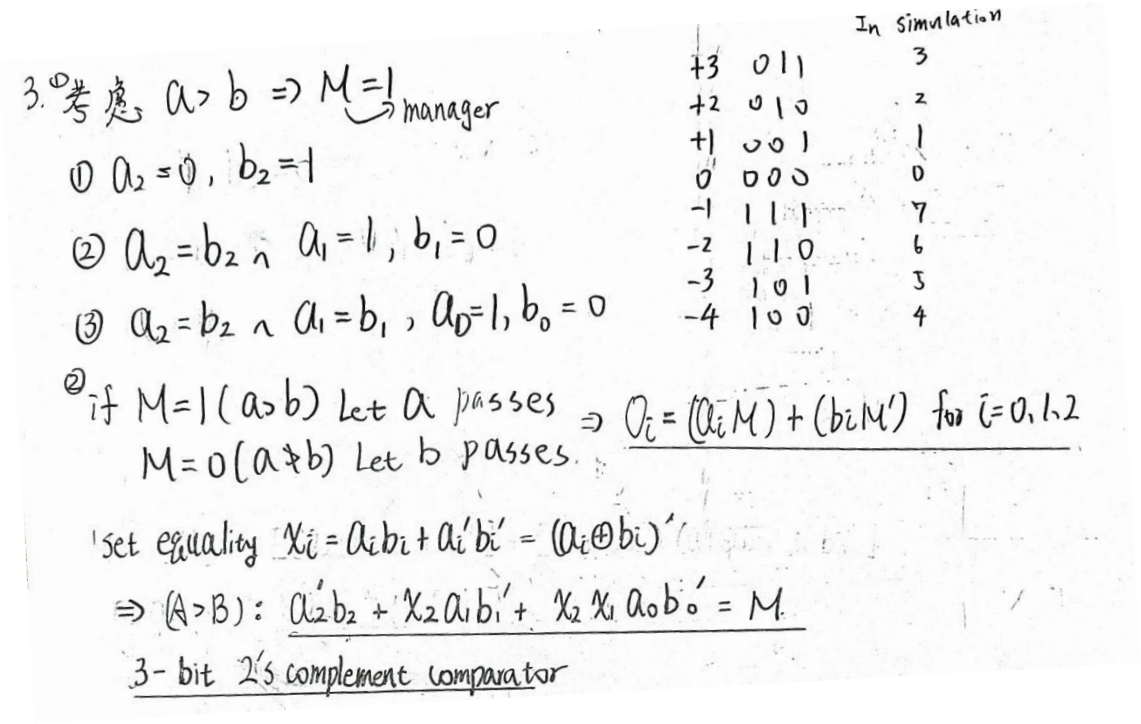
Design Specification：

Input: a [3:0], b [3:0].
Output: o [3:0].
Block diagram:

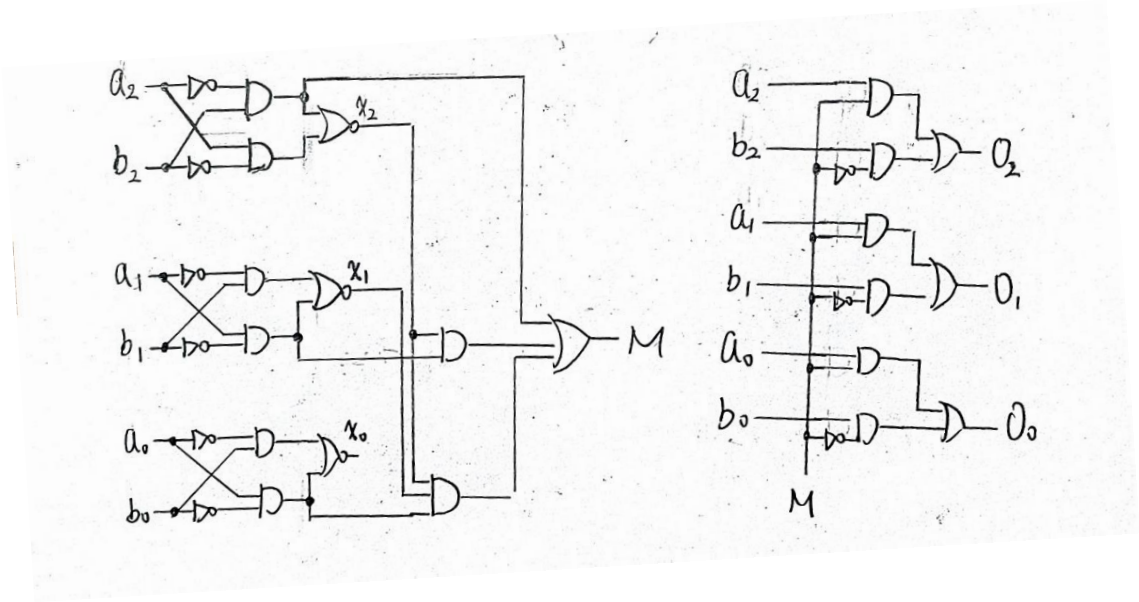


Design Implementation：

Logic equation:



Logic diagram:



Verilog code:

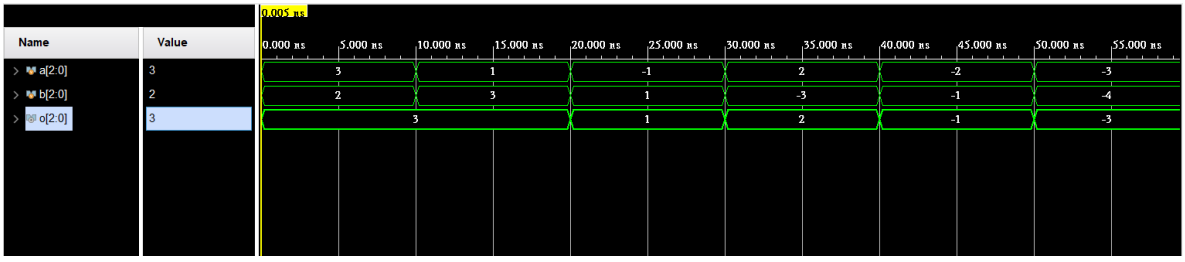
Source code

```
module lab1_3(
    input [2:0] a,
    input [2:0] b,
    output [2:0] o,
    wire m
    );

    assign m =( ~a[2]& b[2]) | ((a[2]==b[2])&a[1]& ~b[1]) | ((a[2]==b[2]) & (a[1]==b[1]) & a[0]& ~b[0]);

    assign o[0]=(a[0]& m) | (b[0]& ~m);
    assign o[1]=(a[1]& m) | (b[1]& ~m);
    assign o[2]=(a[2]& m) | (b[2]& ~m);


endmodule
```

Test bench

```
module test_lab1_3;

    reg [2:0] a;
    reg [2:0] b;
    wire [2:0] o;

    lab1_3 U0(
        .o(o),
        .a(a),
        .b(b)
    );

    initial begin
            a = 3'sd3;   b = 3'sd2;   //o=3
        #10 a = 3'sd1;   b = 3'sd3;   //o=3
        #10 a = -3'sd1;  b = 3'sd1;   //o=1
        #10 a = 3'sd2;   b = -3'sd3;  //o=2
        #10 a = -3'sd2;  b = -3'sd1;  //o=-1
        #10 a = -3'sd3;  b = -3'sd4;  //o=-3
    end

endmodule
```

Simulation waveform:



Discussion：

1. First of all, it dawns on me that we can use the mux-like method to select the larger number, so I review the Quadruple Two-to-One-Line Multiplexer and assign the “m” as a manager/ select bit to decide to let either a or b go through the mux. And, if a is greater than b and then “m” shows 1 and let a pass the mux. Otherwise, if a isn’t greater than b and then “m” shows 0 and let b pass the mux.

2. However, a typical comparator couldn't deal with the signed number but the pure magnitude number. Thus, I used the ''half'' comparator to get m. The difference between ''half'' comparator and the typical comparator only depends on the MSB through observation of the signed number. Thus, we only need to reverse the equation of the MSB part in the typical comparator and then we can get a correct logic diagram as well as logic equation of "m".

3. Finally, I refer to the model of Quadruple Two-to-One-Line Multiplexer to construct the logic diagram to select larger number and write Verilog code to check the idea is correct.

Conclusion：

In this experiment, although the problem isn't difficult, we still need to think logically to obtain the perfect logic circuit to meet the requirement. Furthermore, this example still helps us to realize the observation of the truth table and so forth can benefit a lot.

Like the above experiment, this example still let me get familiar with the combination circuit and Verilog writing.

Reference：

1. https:// www.edn.com/how-to-generate-gray-codes-for-non-power-of-2-sequences/
To get the Gary code with exactly ten codes and learn how to generate Gray Codes for non-power-of-2 sequences.

2. Textbook and handout of logic design
To review some basic concept of the combination circuit and the mechanism of the 2'complement signed number.