Logic Design Lab 2

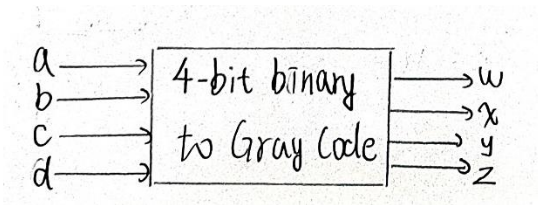
Experiment 1

Emulate exp1 in lab1 (a binary-to-Gray-code converter) in FPGA board with the following parameters.

I/O	*a*	*b*	*c*	*d*	*w*	*x*	*y*	*z*
LOC	W17	W16	V16	V17	V19	U19	E19	U16

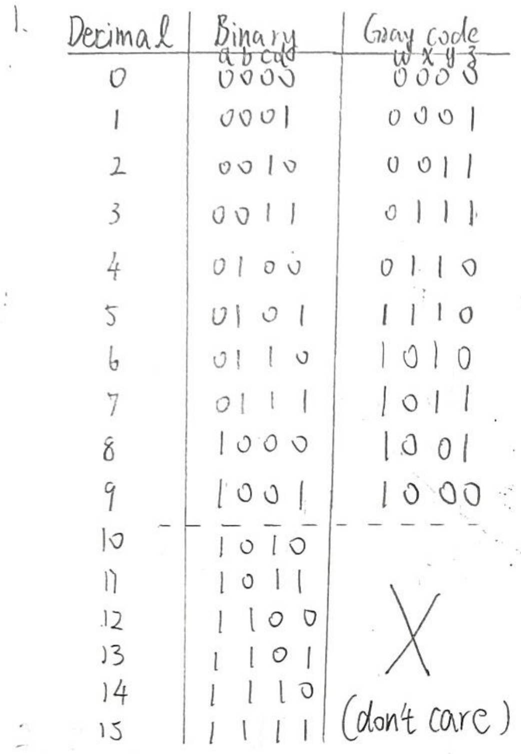
Design Specification：

Input: a,b,c,d.
Output: w,x,y,z.
Block diagram:



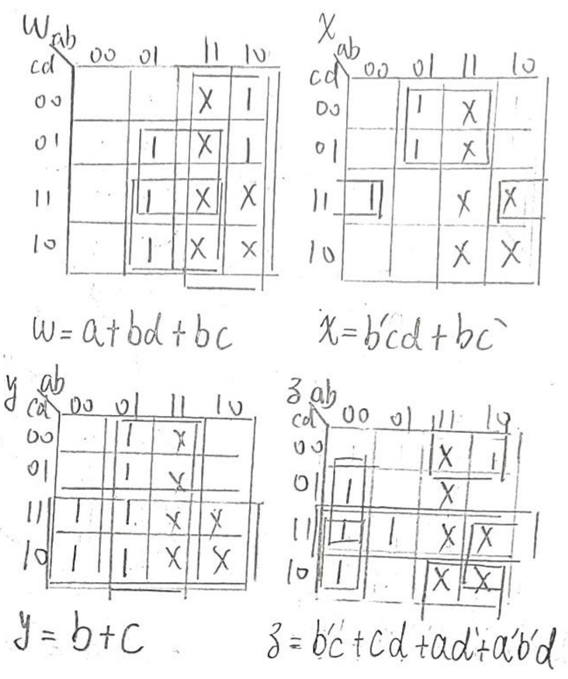
Design Implementation：

Logic equation:



1.

Decimal	Binary a b c d	Gray code w x y z
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0111
4	0100	0110
5	0101	1110
6	0110	1010
7	0111	1011
8	1000	1001
9	1001	1000
10	1010	X (don't care)
11	1011	
12	1100	
13	1101	
14	1110	
15	1111	



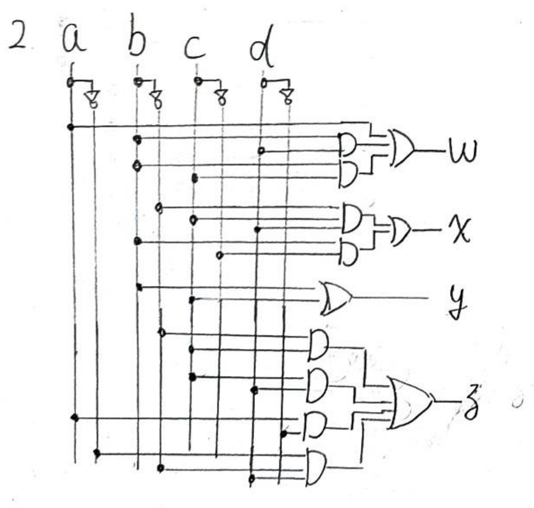
$w = a + bd + bc$

$x = b'cd + bc'$

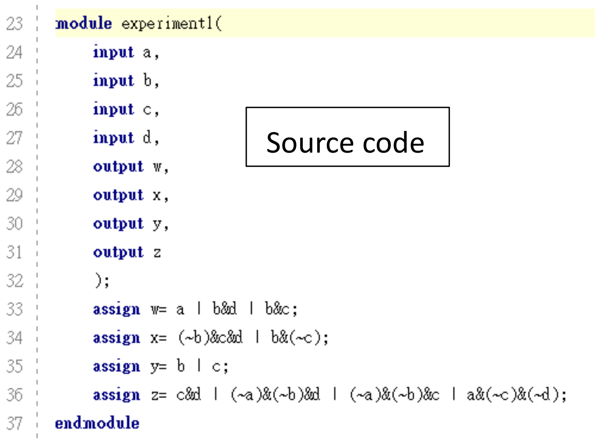
$y = b + c$

$z = b'c' + cd + ad' + a'b'd$

Logic diagram:

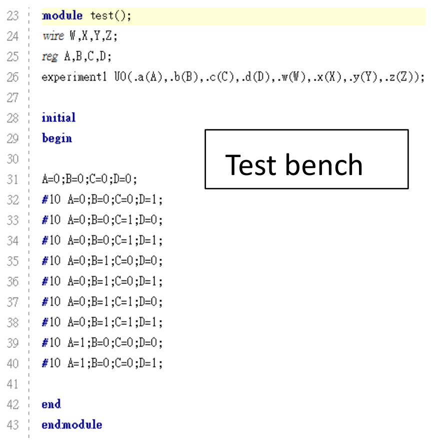


Verilog code:



Source code

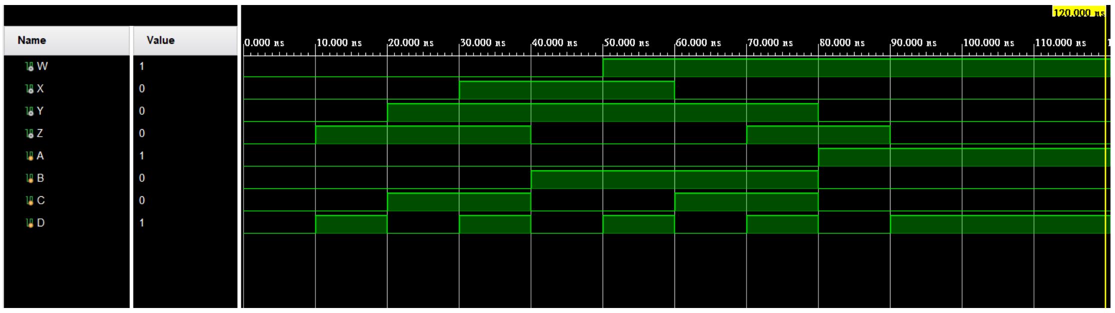
```
module experiment1(
    input a,
    input b,
    input c,
    input d,
    output w,
    output x,
    output y,
    output z
    );
    assign w= a | b&d | b&c;
    assign x= (~b)&c&d | b&(~c);
    assign y= b | c;
    assign z= c&d | (~a)&(~b)&d | (~a)&(~b)&c | a&(~c)&(~d);
endmodule
```



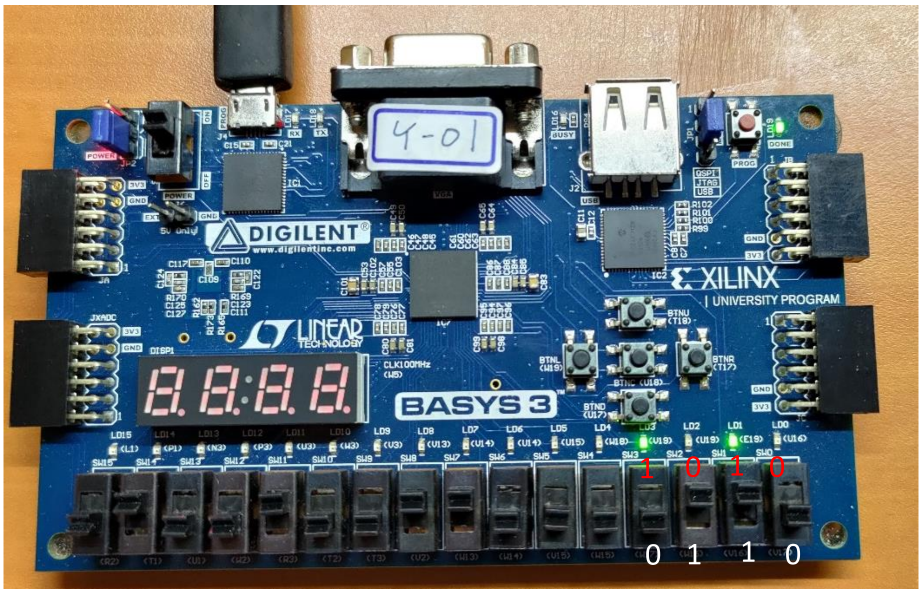
Test bench

```
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:



Example of the results on the FPGA board：



Discussion：

1. The logic diagram and the logic equations are the same as those in lab 1.
2. We should assign the I/O port as what the description mentions.

Conclusion：

This experiment is quite simple and take us less time because what we should do was almost done at lab1. However, it is still a very important experiment to allow us to get familiar to the process of how to program our Varilog code into the FGPA board.

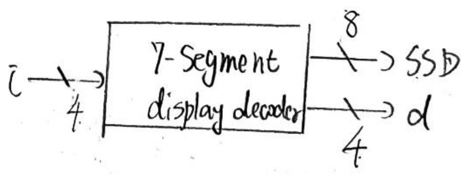
Experiment 2

Design a 4-bit binary (i[3:0], i[3] as MSB) to 7-segment display decoder (SSD[7:0]), and also use four LEDs (d[3:0]) to monitor the 4-bit binary number.

I/O	*i[3]*	*i[2]*	*i[1]*	*i[0]*	*d[3]*	*d[2]*	*d[1]*	*d[0]*
LOC	W17	W16	V16	V17	V19	U19	E19	U16

Design Specification：

Input: i[3:0].
Output: SSD[7:0], d[3:0].
Block diagram:

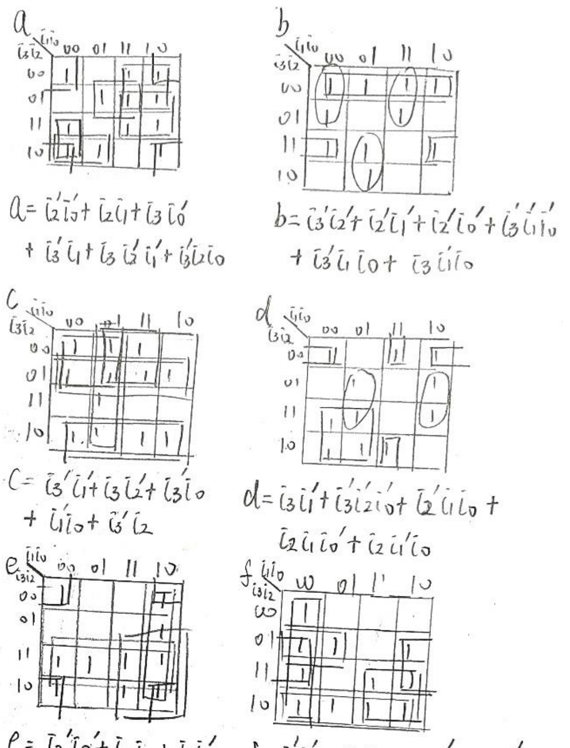


Design Implementation：

1. Logic equations and logic diagrams

	i_3	i_2	i_1	i_0	a (SSD 1)	b (2)	c (3)	d (4)	e (5)	f (6)	g (7)
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1
A	1	0	1	0	1	1	1	0	1	1	1
B	1	0	1	1	0	0	1	1	1	1	1
C	1	1	0	0	1	0	0	1	1	1	0
D	1	1	0	1	0	1	1	1	1	0	1
E	1	1	1	0	1	0	0	1	1	1	1
F	1	1	1	1	1	0	0	0	1	1	1

* 1: on
0: off



$a = i_2'i_0' + i_2i_1 + i_3i_0' + i_3'i_1 + i_3i_2'i_1' + i_3'i_2i_0$

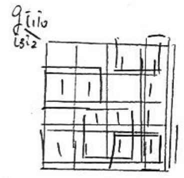
$b = i_3'i_2' + i_2'i_1' + i_2'i_0' + i_3'i_1'i_0' + i_3'i_1i_0 + i_3i_1'i_0$

$c = i_3'i_1' + i_3i_2' + i_3'i_0 + i_1'i_0 + i_3'i_2$

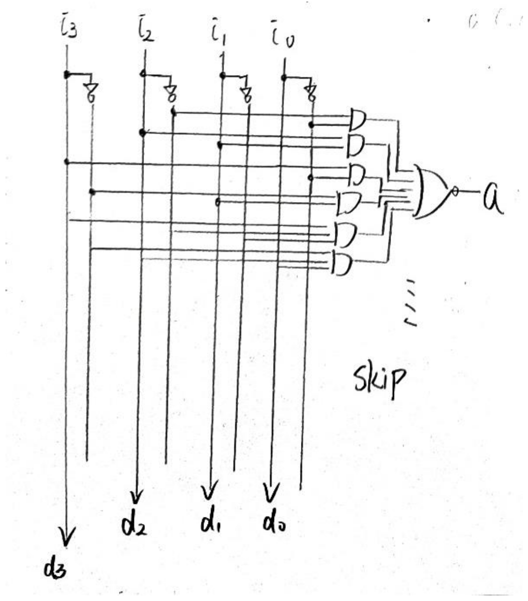
$d = i_3i_1' + i_3'i_2'i_0' + i_2'i_1i_0 + i_2i_1i_0' + i_2i_1'i_0$

$e = i_2'i_0' + i_3i_2 + i_1i_0' + i_3i_1$

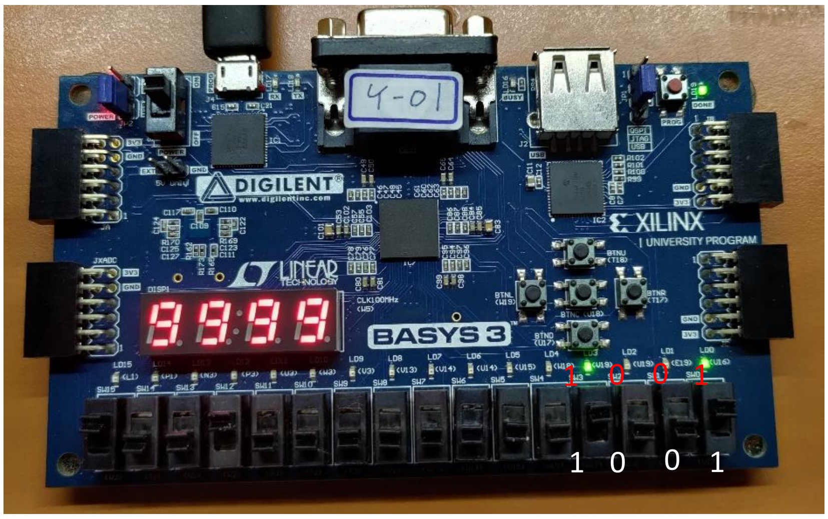
$f = i_1'i_0' + i_3i_1 + i_2i_0' + i_3i_2' + i_3'i_2i_1'$



$g = i_2'i_1 + i_1i_0' + i_3i_0 + i_3i_2' + i_3'i_2i_1'$

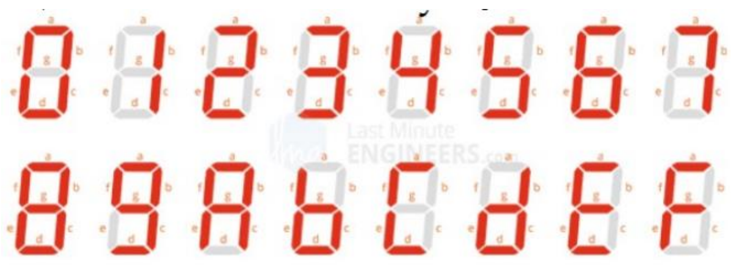


2. Example of the results on the FPGA board：



Discussion：

1. Negative logic of 7-segment displayer：
   First, I assign the segments on '1' while the segments off '0', but we should know that it is the negative logic that the displayer is. Thus, we should complement the result finally.
2. Derive the logic equation：
   According to the picture below, we can find the relation between the segments and the inputs. For example, when the displayer shows '0', the segment a, b, c, d, e, f is on while the g and dot is off. Then, we use K-map to simplify the logic equation and draw the logic diagrams.
   (because we have the logic equations and it is a little bit complicate to draw the logic diagram, I draw the segment 'a' as an example and skip the other ones)



3. The dot on the displayer is always 'off' in this case, so we are encourage to assign 1 to the dot.
4. Finally, we need to check the patten showing on the segment displayer, LED, and on or off of the switch meet our goal.

Conclusion：

At the beginning, I forgot that the 7-segment displayer's operation logic is negative (opposite to what we are used to do) so the patten showing on the displayer was weird. Besides, there are a lot of the logic equations in this lab and the notation of the input(i[3:0]) makes me confused so I had revised the logic equation again and again until I got the correct one, witch took me a lot of time.

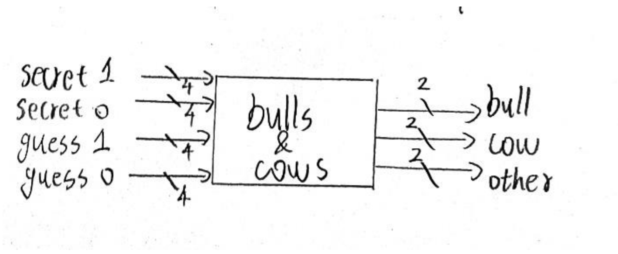
However, this experiment still helps me get familiar to assigning the inputs and outputs to the IO ports and the process of HDL programing. Also, we need to see the handbook or the handout of this board to know the corresponding IO ports of 7-segment displayer. I think it is a essential skill which may be helpful when we do the final project or other things.

Experiment 3

(Bonus) In the Bulls and Cows game, each of the two players writes a two-digit secret number in BCD. The digits must be all different. Then, in turns, the players try to guess their opponent's number and give the number of matches. If the matching digits are in their right positions, they are bulls, and if in different positions, they are cows. In this problem, we want to build the matching process to show the number of bulls and cows.

Design Specification：

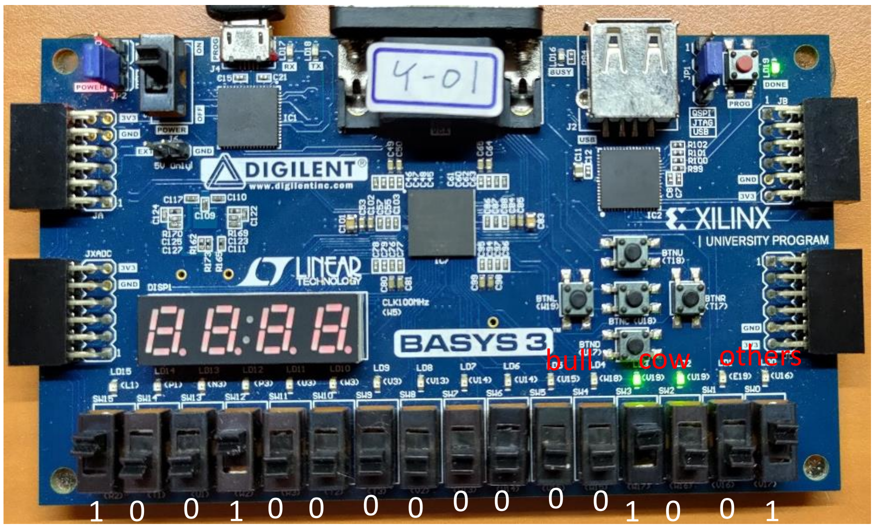
Input: a [3:0], b [3:0].
Output: o [3:0].
Block diagram:



Design Implementation：

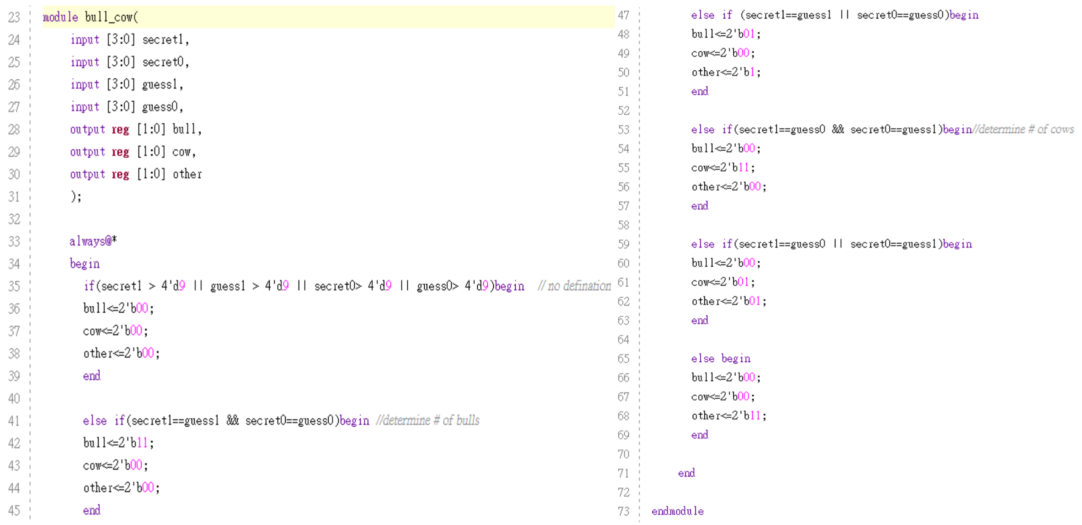
Thought of the implementation:
We can use the high-level language in Verilog to help us do this experiment efficiently. That is to say, we don't use the logic equations and logic diagram including any block diagram to solve the problem.
PS: the details of how we solve the problem will be discussed in the discussion.

Example of the results on the FPGA board：:



Discussion：

1. First of all, with logical analysis, we can conclude that there are only five outcomes that we may get. Those are (2b), (1b,1o), (2c), (1c,1o) and (2o), in which ‘b’ stands for bulls, ‘c’ stands for cows and ‘o’ stands for the other ones.
2. Next, we can use if / else conditions to solve the problem with the same order as the above and we should be careful when we assign our IO ports to meet what TAs requirement. To express my idea clearly, here is the Verilog code that I write.



```verilog
module bull_cow(
    input [3:0] secret1,
    input [3:0] secret0,
    input [3:0] guess1,
    input [3:0] guess0,
    output reg [1:0] bull,
    output reg [1:0] cow,
    output reg [1:0] other
    );

    always@*
    begin
      if(secret1 > 4'd9 || guess1 > 4'd9 || secret0> 4'd9 || guess0> 4'd9)begin  // no defination
      bull<=2'b00;
      cow<=2'b00;
      other<=2'b00;
      end

      else if(secret1==guess1 && secret0==guess0)begin //determine # of bulls
      bull<=2'b11;
      cow<=2'b00;
      other<=2'b00;
      end

      else if (secret1==guess1 || secret0==guess0)begin
      bull<=2'b01;
      cow<=2'b00;
      other<=2'b1;
      end

      else if(secret1==guess0 && secret0==guess1)begin//determine # of cows
      bull<=2'b00;
      cow<=2'b11;
      other<=2'b00;
      end

      else if(secret1==guess0 || secret0==guess1)begin
      bull<=2'b00;
      cow<=2'b01;
      other<=2'b01;
      end

      else begin
      bull<=2'b00;
      cow<=2'b00;
      other<=2'b11;
      end

    end

endmodule
```

3. Finally, we need to check the result meeting our goal.

Conclusion：

In this experiment, though the problem isn’t difficult, we still need to think logically to obtain the concise Verilog code and the correct results. In spite of the fact that we can use high-level language to get the correct answer, I still want to construct/ recall the thought of solving this problem with typical methods. Therefore, I read the homework of the logic design course a year ago to refresh the memory of combination circuit.

Reference：

1. The handout and the assignment of logic design
   To review some basic concept of the combination circuit and the problem bulls and cows.
2. The handout of logic design lab
   To program our FPGA board by following the method on the handout of logic design lab.