Logic Design Lab 3

Pre-lab 1

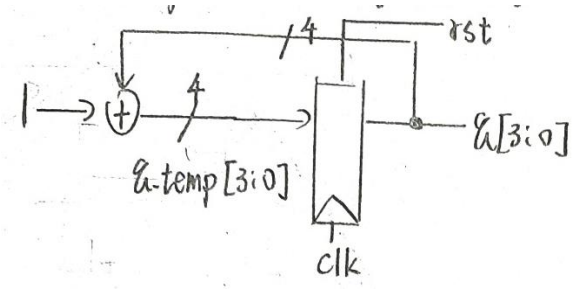
Consider a 4-bit synchronous binary up counter (q3q2q1q0).

Design Specification：

Input: clk,rst.

Output: q [3:0].

Block diagram:



Design Implementation：

Verilog code:

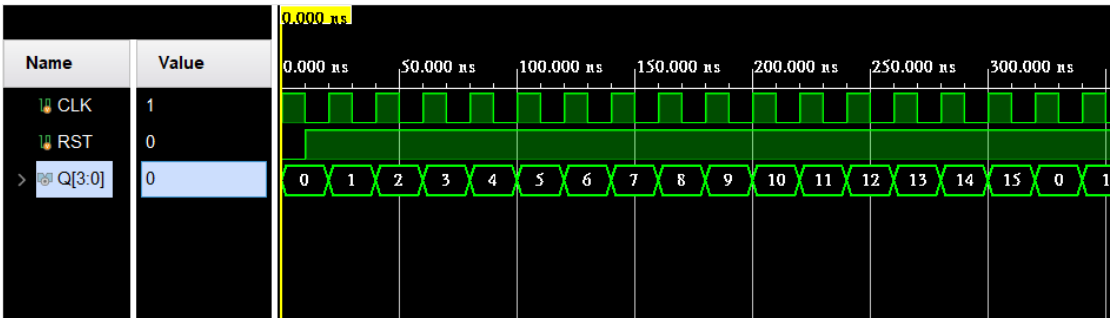
Source code

```
23  module bcdcounter(
24  q, // output
25  clk, // global clock
26  rst // active high reset
27  );
28
29  output [3:0] q; // output
30  input clk; // global clock
31  input rst; // active high reset
32  reg [3:0] q; // output (in always block)
33  reg [3:0] q_tmp; // input to dff (in always block)
34
35  // Combinational logics
36  always @(q)
37      q_tmp = q + 4'b0001;
38
39  // Sequential logics: Flip flops
40  always @(posedge clk or negedge rst)
41      if (~rst)begin
42      q<=4'd0;
43      q_tmp<=4'd0;
44      end
45      else q<=q_tmp;
46
47  endmodule
```

Test bench

```
23  module test();
24  reg CLK,RST;
25  wire [3:0]Q;
26  bcdcounter U0(.q(Q), .clk(CLK),.rst(RST));
27
28  initial
29  begin
30      RST=0;
31      CLK=1;
32      #10RST=1;
33  end
34
35  always
36  begin
37   #10 CLK = ~CLK;
38  end
39
40  endmodule
```

Simulation waveform:



Discussion：

1. First, if we use the commend “if”, we are highly encouraged to add default condition “else” to prevent the uncontrollable condition and debug more effectively.
2. Unlike the traditional logic method, in the Verilog programming, we just assign the input/ output the number of bits we desire, so we can implement the circuit without considering problems of carry.
3. Flowing the Verilog coding style we are used to, we use positive edge trigger for clock while positive edge trigger for reset.
4. To construct Verilog RTL representation, we need to divide our code our idea into two parts. One the combinational logics and the other is the sequential ones.
5. We can easily use #time clock=~clock with always commend to generate a ideal clock in testbench for simulation.

Conclusion：

Through Prelab_1, we have learned how to implement Flip-Flop in Verilog and the concept of RTL level code which means that we divide our code into combinational logics and sequential ones.

Besides, this example gives us the basic idea of how to implement a sequential circuit and it is the fundamental element to help us to move on to the following experiments.

Pre-lab 2

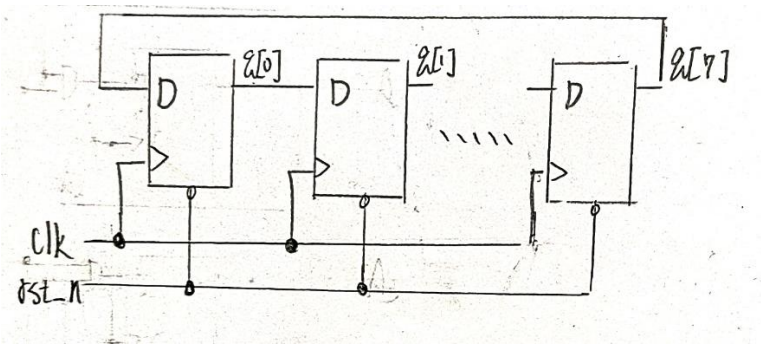
Cascade eight DFFs together as a shift register. Connect the output of the last DFF to the input of the first DFF as a ringer counter. Let the initial value of DFF output after reset be 01010101. Construct the Verilog RTL representation for the logics with verification.

Design Specification：

Input: clk,rst.

Output: q[7:0].

Block diagram:



Design Implementation：

1. Verilog code:

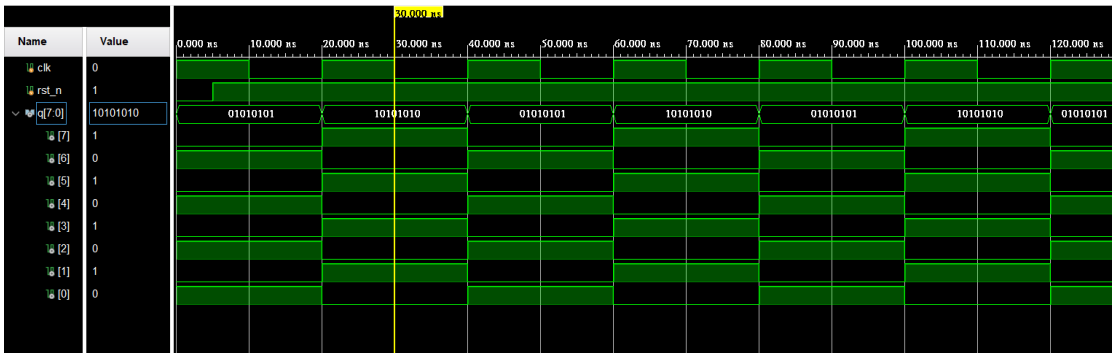
Source code

```
23  module shift_reg(q,clk,rst_n);
24  output [7:0] q; // output
25  input clk; // global clock
26  input rst_n; // active low reset
27  reg [7:0] q; // output
28  integer i;
29
30  // Sequential logics: Flip flops
31  always @(posedge clk or negedge rst_n)
32      if (~rst_n)
33      begin
34      q<=8'b01010101;
35      end
36      else
37          begin
38          q[0]<=q[7];
39          for(i=1;i<8;i=i+1)
40          q[i]<=q[i-1];
41  end
42  endmodule
```

Test bench

```
23  module test_shift_reg();
24  reg clk,rst_n;
25  wire [7:0]q;
26  shift_reg U0(.q(q),.clk(clk),.rst_n(rst_n));
27  initial begin
28  clk=1;
29  rst_n<=0;
30  #5 rst_n<=1;
31  end
32
33  always
34  begin
35  #10 clk<=~clk;
36  end
37  endmodule
```

2. Simulation waveform:



Discussion：

1. The key of how to write sequential circuits with Verilog is discussed in the prelab_1.
2. The only thing I want to discuss in this lab is that the “for” commend in Verilog is pretty different from C language. Things in the “for loop” are carried out at the same time, so it is quite like the “copy” of the circuit with different indexes in Verilog.

Conclusion：

After finishing the prelab_1, I spent little time because there are a lot of implementations sharing same idea. Therefore, I can only copy and paste the code from previous lab and then change some parameter so that we can get a correct one.

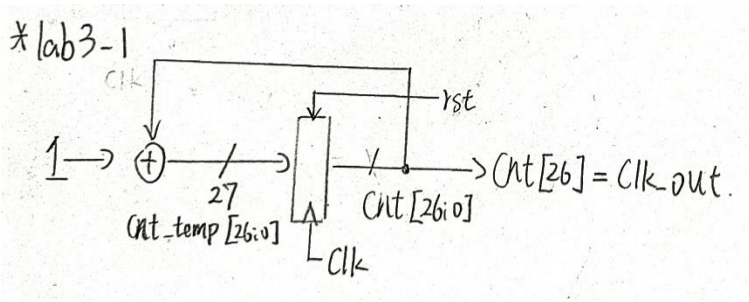
Experiment 1

Frequency Divider: Construct a 27-bit synchronous binary counter. Use the MSB of the counter, we can get a frequency divider which provides a $1/2^{27}$ frequency output (f_{out}) of the original clock ($f_{crystal}$, 100MHz). Construct a frequency divider of this kind.

I/O	$f_{crystal}$	f_{out}	rst
Site	W5	U16	V17

Design Specification：

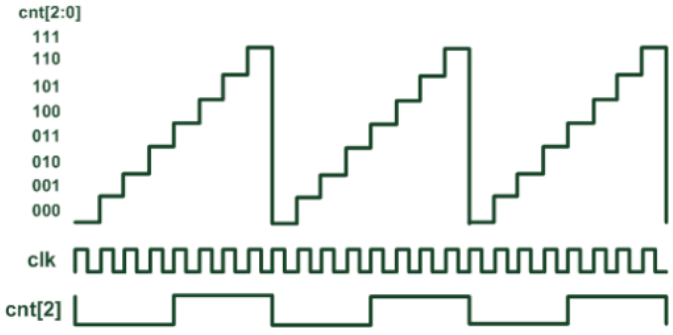
Input: clk($f_{crystal}$),rst.
Output: clk_out(f_{out}).
Block diagram:



Design Implementation：

The specification of the frequency divider:
Like the name of the frequency divider, we can divide the higher frequency(rapid) signal to lower frequency(slower) signal with the frequency divider. In this case, we want to divide 2^{27} to the 100M HZ clock to provide the slower clock.

In this example, we use the 27-bit-binary-up-counter to achieve this goal. The concept is that each time when the faster clock go from 0 to 1, the counter will plus one. Thus, we just focus on the highest bit. It is always 0 in the first half period of the slower clock while it is always 1 in the last half period of the slower clock, so it can be a ideal clock with the function we want.



(copied from the lecture handout)

Discussion：

1. From the block diagram, we can easily observe that lab3_1 is the application of prelab3_1. That is to say, if we know the mechanism of the frequency divider, we can use the same model from prelab3_1 to carry out this experiment.

2. However, there is a limit of the method. The frequency divider could only divide the original clock to the numbers of power of 2. But, in next experiment, we will conduct another method with more flexibility.

Conclusion：

Like the discussion above, this experiment is a well-know application of the prelab_1 which is a synchronous binary counter so I just follow the process of the prelab_1 and change some parameters to meet the goal.

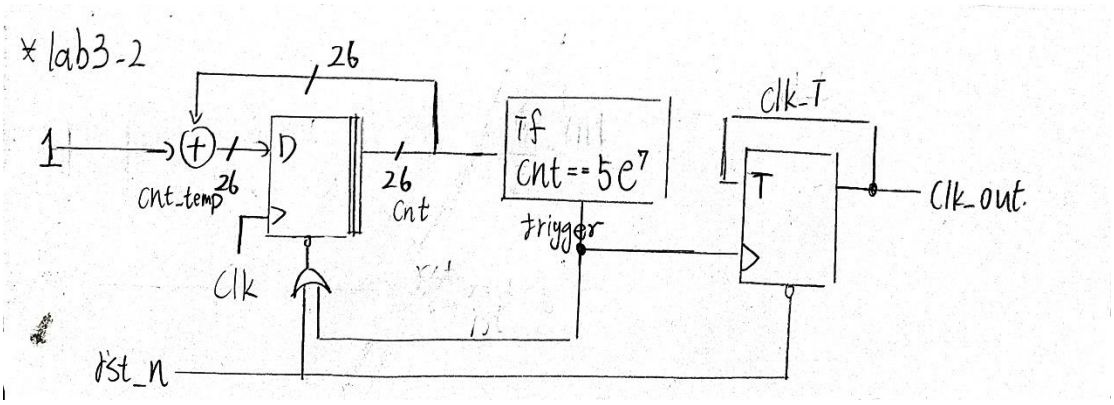
Experiment 2

Frequency Divider: Use a count-for-50M counter and some glue logics to construct a 1 Hz clock frequency. Construct a frequency divider of this kind.

I/O	$f_{crystal}$	f_{out}	rst
Site	W5	U16	V17

Design Specification：

Input: clk($f_{crystal}$),rst.
Output: clk_out(f_{out}).
Block diagram:



Design Implementation：

The specification of the frequency divider:
Like the description in experiment_1, we want to construct a circuit with the function of dividing frequency (let the clock become slower) with more flexibility.

In this example, we are informed the other method to achieve the function of dividing frequency. In this implementation, we use a binary up counter to count numbers. We use “if / else” condition to determine how large the number we want, and the number is related to the clock that we want to put out.

Take this experiment as example, if we want to get a 1 HZ clock from the 100MHZ clock, we can set the counter to count to 50M. the reason is that if we change(complement) the output and reset the counter when the counter count to 50M, we can get the 1 HZ clock directly.

Discussion：

1. Unlike the process showed in handout with MUX, I use the idea of T-Flip Flop to construct my circuit. The reason why I use the method is that it is more straight forward to use T-Flip Flop for me through the specification of the frequency divider.
2. However, by using this idea, I need to remember to the counter when the counter counts to 50M.

Conclusion：

From my perspective, I think this experiment is more interesting than the previous experiments. From comprehending what the mechanism of this frequency divider to build a block diagram to construct the circuit with Verilog, I can start to construct more complicated circuit with various functions instead of getting familiar to the FPGA board and the software.

Experiment 3

Implement pre-lab1 with clock frequency of 1 Hz. Use the following I/O to demonstrate the counter results.

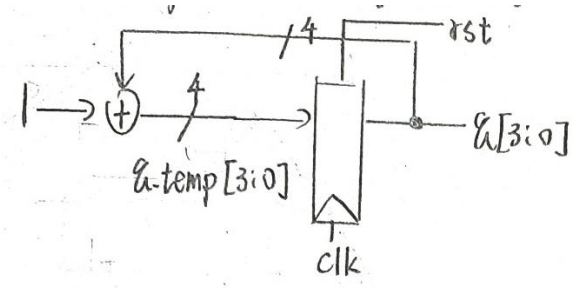
I/O	$f_{crystal}$	f_{out}	rst
Site	W5	U16	V17

Design Specification：

Input: clk,rst.

Output: q [3:0].

Block diagram:



Design Implementation：

The same as prelab3_1.

Discussion：

The same as prelab3_1.

Conclusion：

The same as prelab3_1.

Experiment 4

Implement pre-lab2 with clock frequency of 1 Hz. The I/O pins can be assigned by yourself.

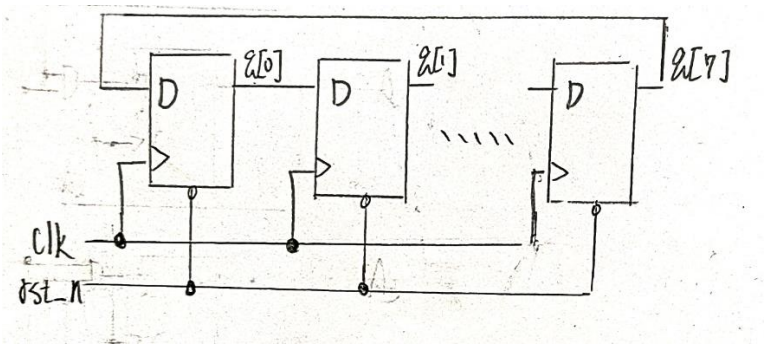
I/O	$f_{crystal}$	q[7]	q[6]	q[5]	q[4]	q[3]	q[2]	q[1]	q[0]	rst
Site	W5	V14	U14	U15	W18	V19	U19	E19	U16	V17

Design Specification：

Input: clk,rst.

Output: q[7:0].

Block diagram:



Design Implementation：

The same as prelab3_2.

Discussion：

The same as prelab3_2.

Conclusion：

The same as prelab3_2.

Experiment 5

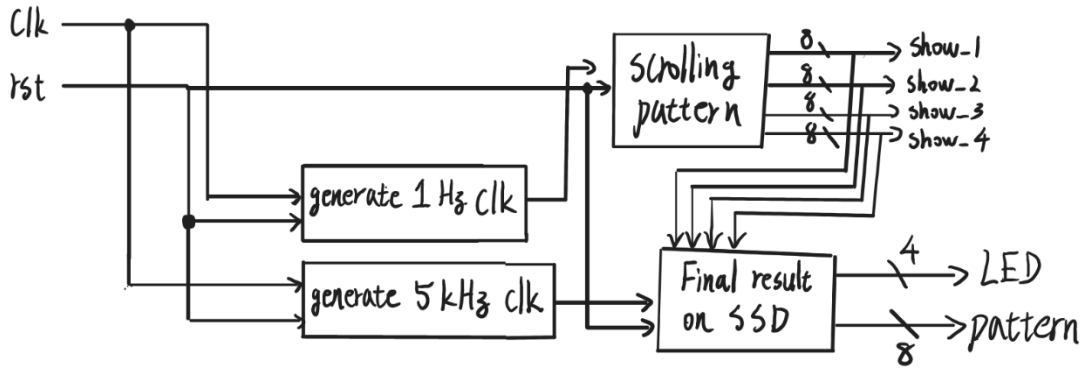
Use the idea from pre-lab2. We can do something on the seven-segment display. Assume we have the pattern of E, H, N, T, U for seven-segment display as shown below. Try to implement the scrolling pre-stored pattern "NTHUEE2023" with the four seven-segment displays.

Design Specification：

Input: clk, rst.

Output: pattern [7:0],LED [3:0].

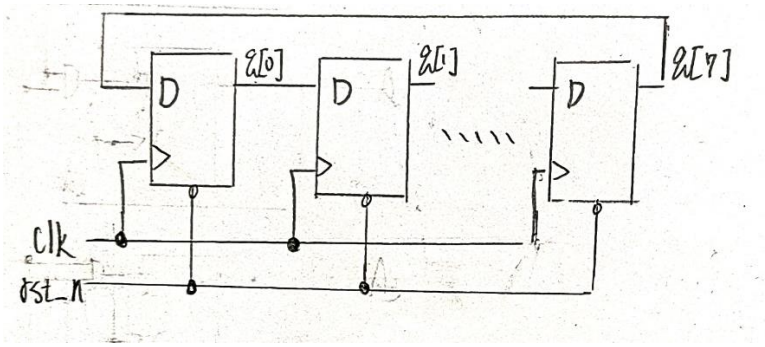
Block diagram:



Design Implementation：

Thought of the implementation:

1. We need two clocks with different frequency to implement this experiment. These two clocks are made from lab3_2(skip discussion).



2. The block diagram of scrolling pattern is to output 4 singles with 8bits (show_1, show_3, show_3, show_4) which assign the patterns showing on the 4 different positions on the SSD.
   (PS: there are another method, and it will be compared to this method in discussion)

```
always@(posedge clk_1HZ or negedge rst) //change pattern
begin
   if (~rst)
    scrolling_pattern<= 10'b0000000001;
    else
        begin
        scrolling_pattern[0]<=scrolling_pattern[9];
        for(i=1;i<10;i=i+1)
        scrolling_pattern[i]<=scrolling_pattern[i-1];
        end

        case(scrolling_pattern)
        10'b0000000001 : begin show_1<=`SS_N show_2<=`SS_T show_3<=`SS_H show_4<=`SS_U  end
        10'b0000000010 : begin show_1<=`SS_T show_2<=`SS_H show_3<=`SS_U show_4<=`SS_E  end
        10'b0000000100 : begin show_1<=`SS_H show_2<=`SS_U show_3<=`SS_E show_4<=`SS_E  end
        10'b0000001000 : begin show_1<=`SS_U show_2<=`SS_E show_3<=`SS_E show_4<=`SS_2  end
        10'b0000010000 : begin show_1<=`SS_E show_2<=`SS_E show_3<=`SS_2 show_4<=`SS_0  end
        10'b0000100000 : begin show_1<=`SS_E show_2<=`SS_2 show_3<=`SS_0 show_4<=`SS_2  end
        10'b0001000000 : begin show_1<=`SS_2 show_2<=`SS_0 show_3<=`SS_2 show_4<=`SS_3  end
        10'b0010000000 : begin show_1<=`SS_0 show_2<=`SS_2 show_3<=`SS_3 show_4<=`SS_N  end
        10'b0100000000 : begin show_1<=`SS_2 show_2<=`SS_3 show_3<=`SS_N show_4<=`SS_T  end
        10'b1000000000 : begin show_1<=`SS_3 show_2<=`SS_N show_3<=`SS_T show_4<=`SS_H  end
        default  : begin show_1<=0; show_2<=0; show_3<=0; show_4<=0;  end
        endcase
end
```

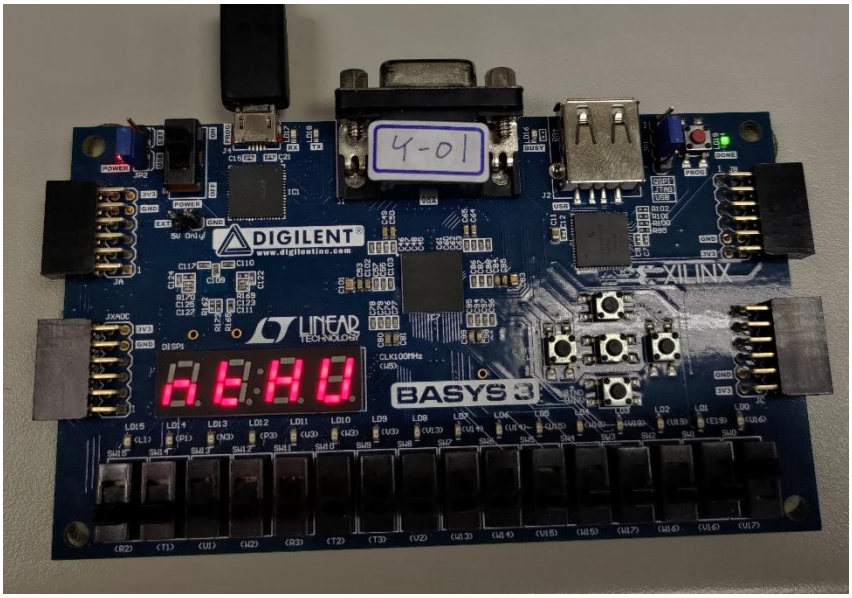
3. The block diagram of scrolling pattern gets the signals from scrolling pattern and then change which displayer is on (LED) as well as which pattern shows on the SSD (pattern) rapidly.

```
always@(posedge clk_5kHZ or negedge rst) //Persistence of vision
begin
    if (~rst) begin
    LED<=4'b1110;
    pattern <= 8'b11111111;
    end
    else
        begin
        LED[0]<=LED[3];
        for(i=1;i<4;i=i+1)
        LED[i]<=LED[i-1];

        case(~LED)
        4'b0001 : pattern <= show_3;
        4'b0010 : pattern <= show_2;
        4'b0100 : pattern <= show_1;
        4'b1000 : pattern <= show_4;
        default : pattern <= 8'b11111111;
        endcase

        end
end
```

Example of the results on the FPGA board：:



Discussion：

1. First, we need two clocks with different frequency to implement this experiment. The faster one (5k HZ) is to allow the seven-segment displayer (SSD) to show different number on corresponding position. The slower one (1 HZ) is let the pattern on the series of SSD scroll from right to left.

2. These two clocks can use the result of lad3_2 to get clocks with different frequencies by changing some parameters.

3. In order to show different patterns on the different positions on SSD synchronously, we must use the phenomenon of “視覺暫留” , which means that we should change the patterns rapidly to let our eye determine the previous patterns still showing.

4. Remember that it is the negative logic in the SSD, so we should set the parameter carefully.

5. In implementation 2., because there are 10 elements in the series (NTHUEE2023), there are 10 combinations of the patterns show on the SSD. Thus, I list all the conditions and assign each condition with the corresponding pattern. However, there are another method that is to store the 10 of 8 bits elements to a shift-register. The latter has more insight on the shit-register. However, if the number of the elements in the series aren’t large, the former may be a good way.

Conclusion：

This experiment is more complicated than the previous experiments and we start to learn to combine modules from previous experiment. However, I think the experiment not only help us get familiar to the sequential circuit design with Verilog but also is interesting. Besides, although while doing the experiment, I debugged the error code from time to time, I got a great sense of achievement when I saw the correct result.

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.