Lab 5 – Design and Application of Variable Decoder

Name: 余若涵 Student ID: 3180105412 Major: Computer Science and Technology
Course: Logic and Computer Design Fundamentals Groupmate: 吴晗晗
Date: 2019-10-17 Laboratory: East4-509, Zijingang Instructor: 洪奇军

1. Purpose and Requirements

1.1 Master the logical structure and logic functions of the variable decoder.
1.2 Implementing a combination function with a variable decoder
1.3 Master the typical application of variable decoder (the specific method of address decoding)
1.4 Understand the concept of memory addressing
1.5 Schematic design circuit module
Familiar with hardware description language by principle
1.6 Further familiar with ISE platform and download experimental platform physical verification

2. Experiment Tasks and Theory

2.1 Experiment Tasks

Basic

2.1.1 Schematic design to achieve 74HC138 decoder module.
2.1.2 Realize the corridor light controller with 74HC138 decoder.
2.1.3 System test stimulus code for designing timing simulation.

Advanced

2.1.4 Design 32×32bit ROM IP core
2.1.5 Variable Decoder Address Decoding Application

2.2 Experiment Principle

2.2.1 Variable Decoder Principle: Universal Decoding

Variable decoder is a general purpose decoder. Its function is to translate all the minimum term states composed of input variables into state information or control signals corresponding to them one-to-one.

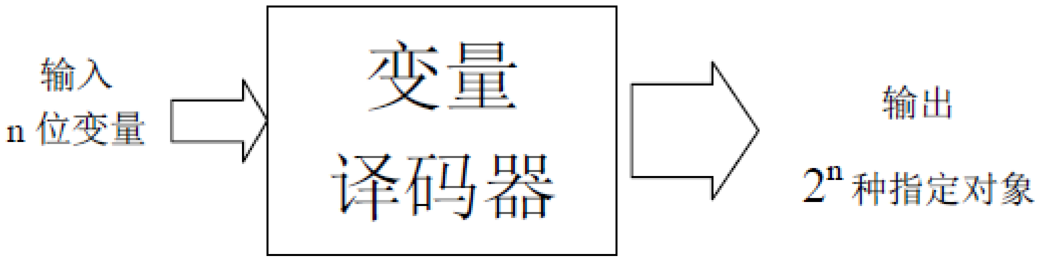
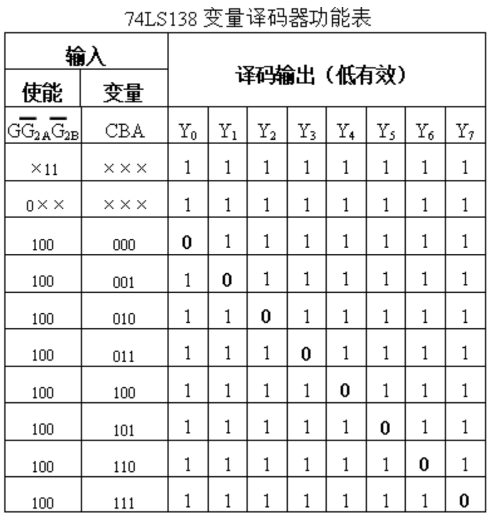


Figure 1 Principle of Variable Decoder

2.2.2 3-8 Variable Decoder



74LS138 变量译码器功能表

输入		译码输出（低有效）							
使能	变量								
$G\overline{G}_{2A}\overline{G}_{2B}$	CBA	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
×11	×××	1	1	1	1	1	1	1	1
0××	×××	1	1	1	1	1	1	1	1
100	000	**0**	1	1	1	1	1	1	1
100	001	1	**0**	1	1	1	1	1	1
100	010	1	1	**0**	1	1	1	1	1
100	011	1	1	1	**0**	1	1	1	1
100	100	1	1	1	1	**0**	1	1	1
100	101	1	1	1	1	1	**0**	1	1
100	110	1	1	1	1	1	1	**0**	1
100	111	1	1	1	1	1	1	1	**0**

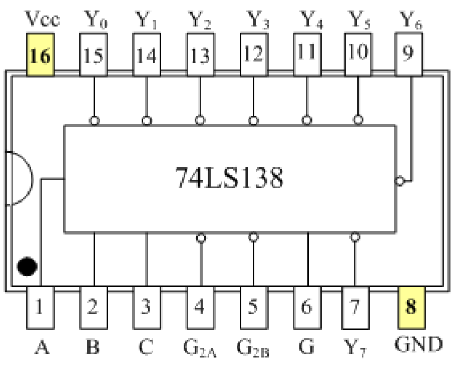
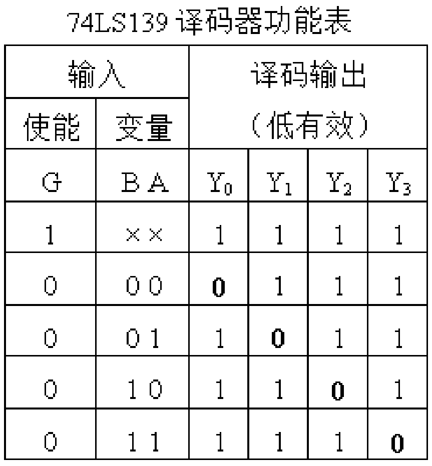


Figure 2 Function of 3-8 Variable Decoder Figure 3 Pins of 3-8 Variable Decoder

2.2.3 Double 2-4 variable decoder



74LS139 译码器功能表

输入		译码输出（低有效）			
使能	变量				
G	B A	Y_0	Y_1	Y_2	Y_3
1	× ×	1	1	1	1
0	0 0	**0**	1	1	1
0	0 1	1	**0**	1	1
0	1 0	1	1	**0**	1
0	1 1	1	1	1	**0**

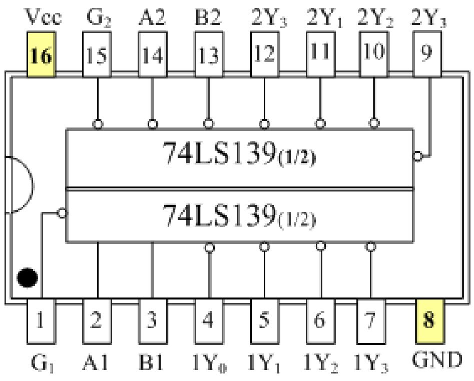


Figure 4 Function of 2-4 Variable Decoder   Figure 5 Pins of 2-4 Variable Decoder

2.2.4 Implementing a combination function with a variable decoder

The output of the variable decoder corresponds to the minimum combination of all input variables. If the function is converted to the form of the minimum term sum, the variable decoder can be used to implement the combined circuit of the functions.

2.2.5 Variable Decoder Implements Memory Address Decoding

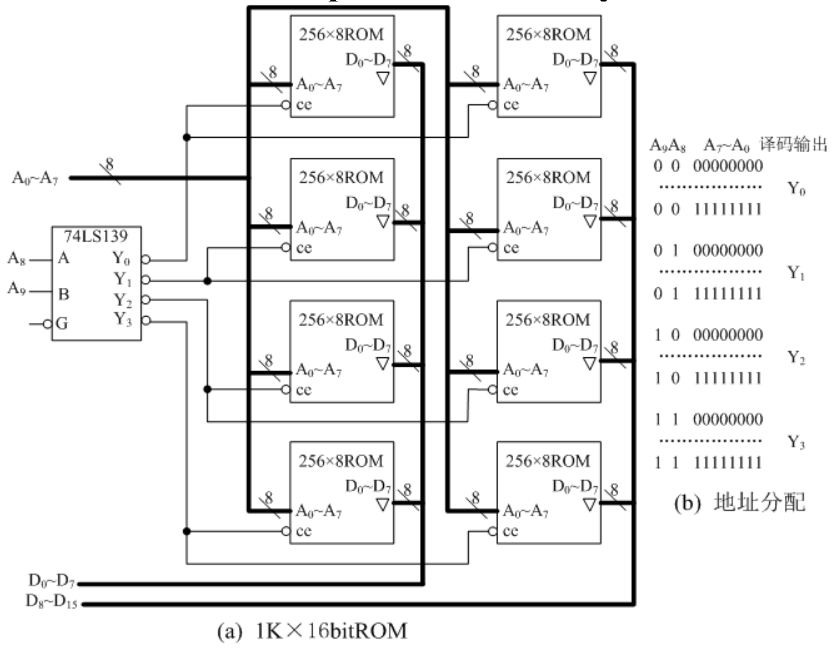


Figure 5 Chip select or output enable for controlling memory

3. Experiment Instruments and Materials

1. Computer with Xilinx ISE 14.4 or above   1 unit

2. Sword experiment system 1 unit

4. Experiment Procedure and Operations

4.1 Design Project1: Exp14-HCT138

Design to implement 74LS138/74HC138/HCT138

Note: (1) Schematic file names cannot begin with a number

(2) Variable name cannot use overline.

Step 1: Create a new FPGA project named Exp05-38Decoder.

Step 2: Create a Schematic file named Decoder_38_sch.

Design Decoder_38_sch according to the principle diagram

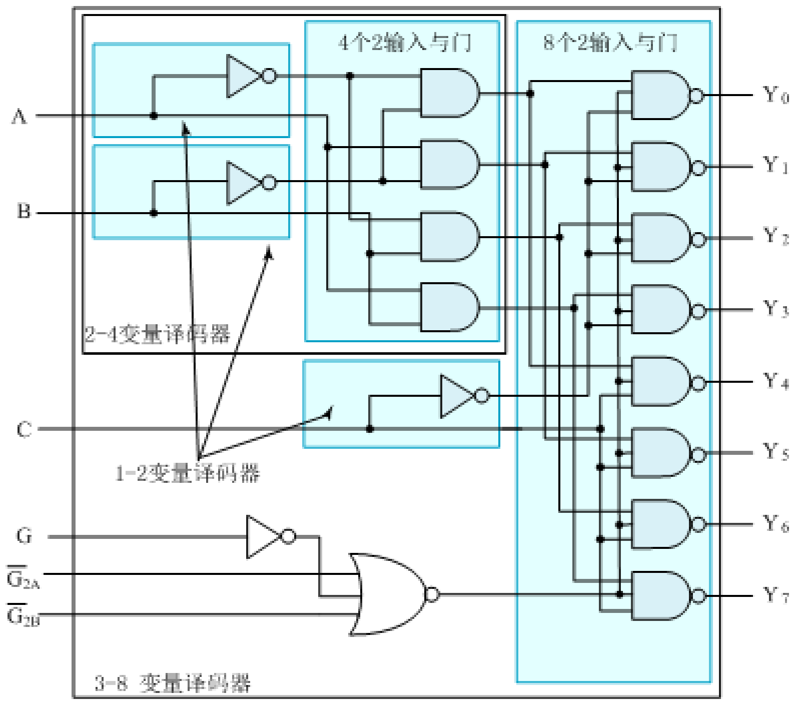


Figure 6 Principle diagram of Decoder_38_sch

Check Design Rules and View HDL Functional Model

Create Schematic Symbol

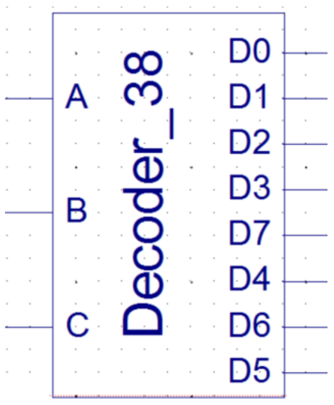


Figure 7 Symbol of Decoder_38_sch

Step 3: Create a Schematic file named HCT138.

Design HCT138 according to the principle diagram

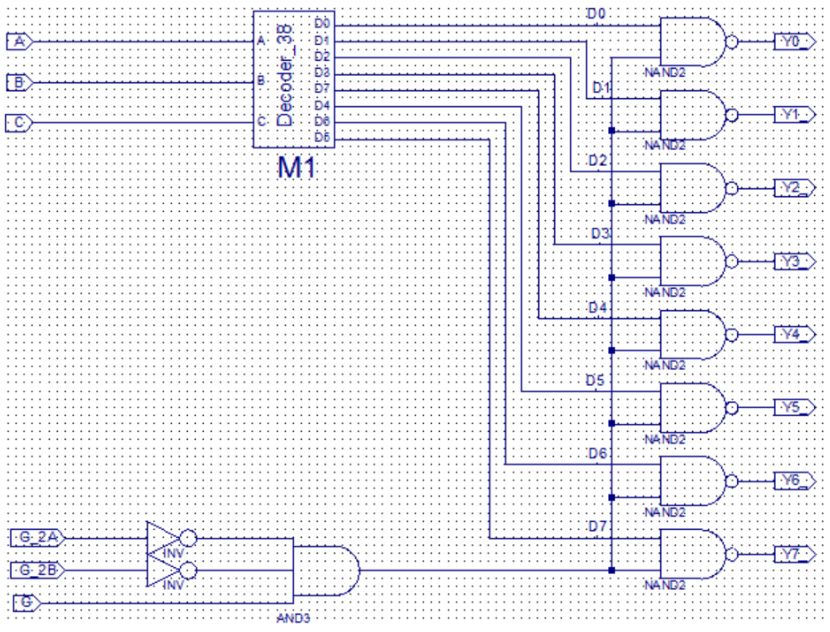


Figure 8 Principle diagram of HCT138

Check Design Rules and View HDL Functional Model

Create Schematic Symbol

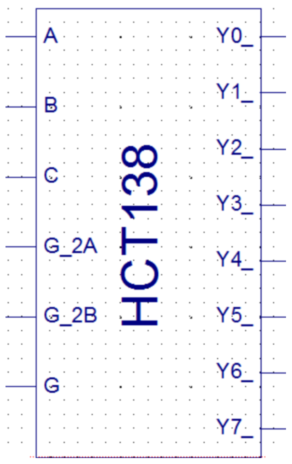
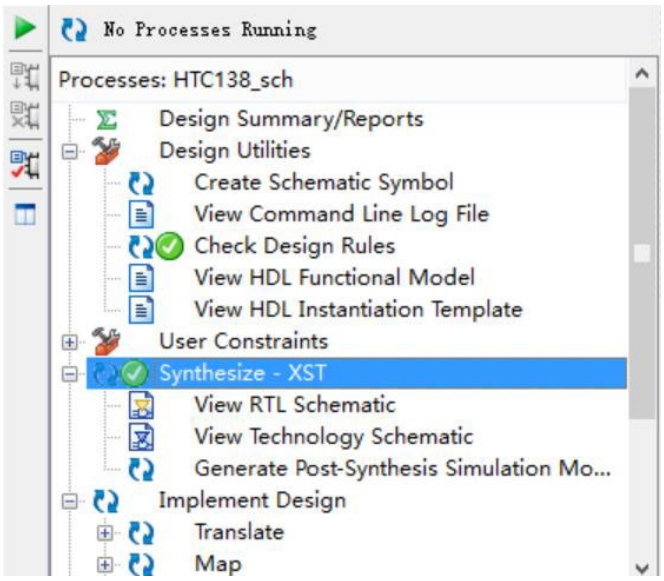


Figure 9 Symbol of HCT138

Step 4: Synthesize – XST


No Processes Running

Processes: HTC138_sch

- Design Summary/Reports
- Design Utilities
  - Create Schematic Symbol
  - View Command Line Log File
  - Check Design Rules
  - View HDL Functional Model
  - View HDL Instantiation Template
- User Constraints
- Synthesize - XST
  - View RTL Schematic
  - View Technology Schematic
  - Generate Post-Synthesis Simulation Mo...
- Implement Design
  - Translate
  - Map

Figure 10 Synthesize button

Step 5: Simulation of logic module

Create a Verilog Test Fixture file named Decoder_test.

Input the simulation code.

```
integer i; //数值变量定义
initial begin //这个是模板生成的……
// Add stimulus here
G = 1;
G2A = 0;
G2B = 0;
#50;
for (i=0; i<=7;i=i+1) begin
{C,B,A}={C,B,A}+1;
#50;
end
assign G = 0;
assign G2A = 0;
assign G2B = 0;
#50;
assign G = 1;
assign G2A = 1;
assign G2B = 0;
#50;
assign G = 1;
assign G2A = 0;
assign G2B = 1;
#50;
end
```

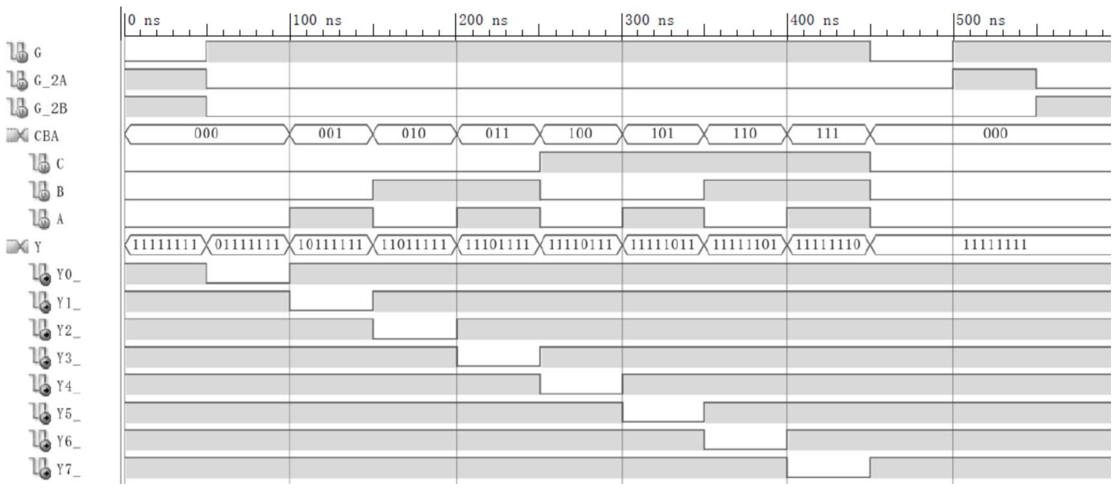
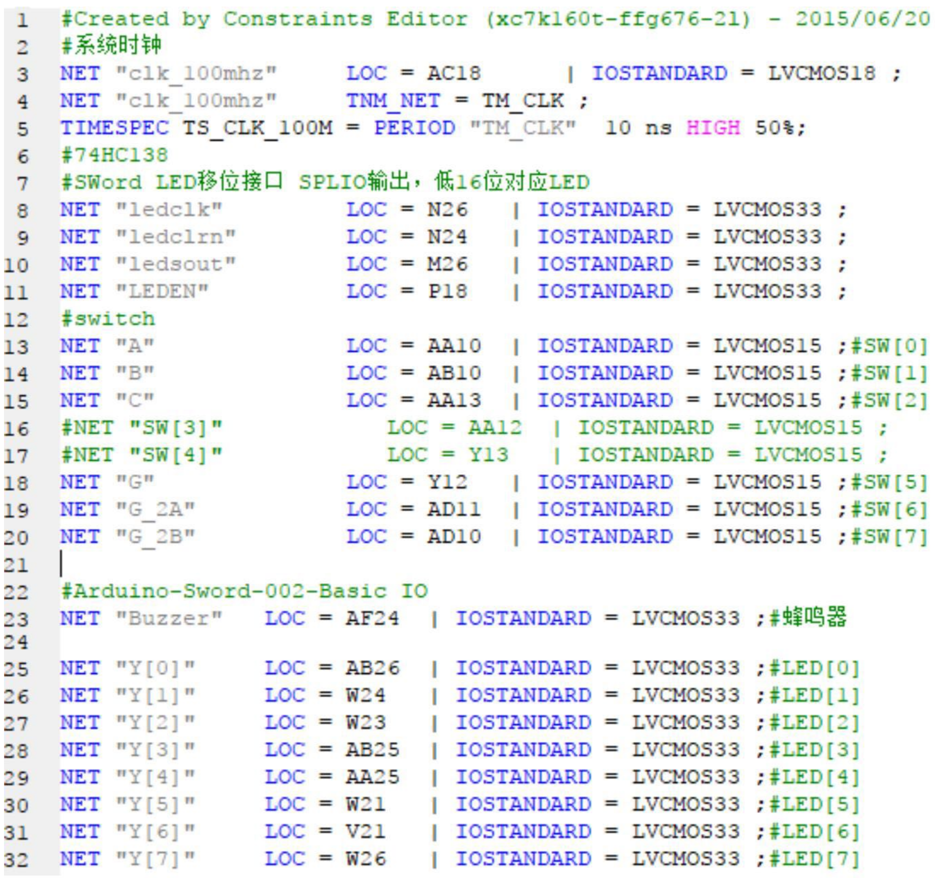


Figure 11 Decoder Simulation

Step 6: Constraint and implementation

Create a Implementation Constraints File named HCT138.ucf associated with Top_HCT138.

Input the UCF file.



```
#Created by Constraints Editor (xc7k160t-ffg676-21) - 2015/06/20
#系统时钟
NET "clk_100mhz"      LOC = AC18      | IOSTANDARD = LVCMOS18 ;
NET "clk_100mhz"      TNM_NET = TM_CLK ;
TIMESPEC TS_CLK_100M = PERIOD "TM_CLK"  10 ns HIGH 50%;
#74HC138
#SWord LED移位接口 SPLIO输出，低16位对应LED
NET "ledclk"          LOC = N26    | IOSTANDARD = LVCMOS33 ;
NET "ledclrn"         LOC = N24    | IOSTANDARD = LVCMOS33 ;
NET "ledsout"         LOC = M26    | IOSTANDARD = LVCMOS33 ;
NET "LEDEN"           LOC = P18    | IOSTANDARD = LVCMOS33 ;
#switch
NET "A"               LOC = AA10   | IOSTANDARD = LVCMOS15 ;#SW[0]
NET "B"               LOC = AB10   | IOSTANDARD = LVCMOS15 ;#SW[1]
NET "C"               LOC = AA13   | IOSTANDARD = LVCMOS15 ;#SW[2]
#NET "SW[3]"             LOC = AA12   | IOSTANDARD = LVCMOS15 ;
#NET "SW[4]"             LOC = Y13    | IOSTANDARD = LVCMOS15 ;
NET "G"               LOC = Y12    | IOSTANDARD = LVCMOS15 ;#SW[5]
NET "G_2A"            LOC = AD11   | IOSTANDARD = LVCMOS15 ;#SW[6]
NET "G_2B"            LOC = AD10   | IOSTANDARD = LVCMOS15 ;#SW[7]

#Arduino-Sword-002-Basic IO
NET "Buzzer"    LOC = AF24   | IOSTANDARD = LVCMOS33 ;#蜂鸣器

NET "Y[0]"      LOC = AB26   | IOSTANDARD = LVCMOS33 ;#LED[0]
NET "Y[1]"      LOC = W24    | IOSTANDARD = LVCMOS33 ;#LED[1]
NET "Y[2]"      LOC = W23    | IOSTANDARD = LVCMOS33 ;#LED[2]
NET "Y[3]"      LOC = AB25   | IOSTANDARD = LVCMOS33 ;#LED[3]
NET "Y[4]"      LOC = AA25   | IOSTANDARD = LVCMOS33 ;#LED[4]
NET "Y[5]"      LOC = W21    | IOSTANDARD = LVCMOS33 ;#LED[5]
NET "Y[6]"      LOC = V21    | IOSTANDARD = LVCMOS33 ;#LED[6]
NET "Y[7]"      LOC = W26    | IOSTANDARD = LVCMOS33 ;#LED[7]
```

Figure 12 Code of UCF file

Step 7: Implement Design and Generate Programming File.

Step 8: Download FPGA programing file

4.2 Design Project 2: Exp16-Lamp138

Design to implement corridor lamp control

Step 1: Create a new project

Step 2: Add a copy of file HCT138.sym and Decoder_38.sym, Decoder_38.sch and HTC138_sch.sch

Step 3: Create a Schematic file and design as follows.

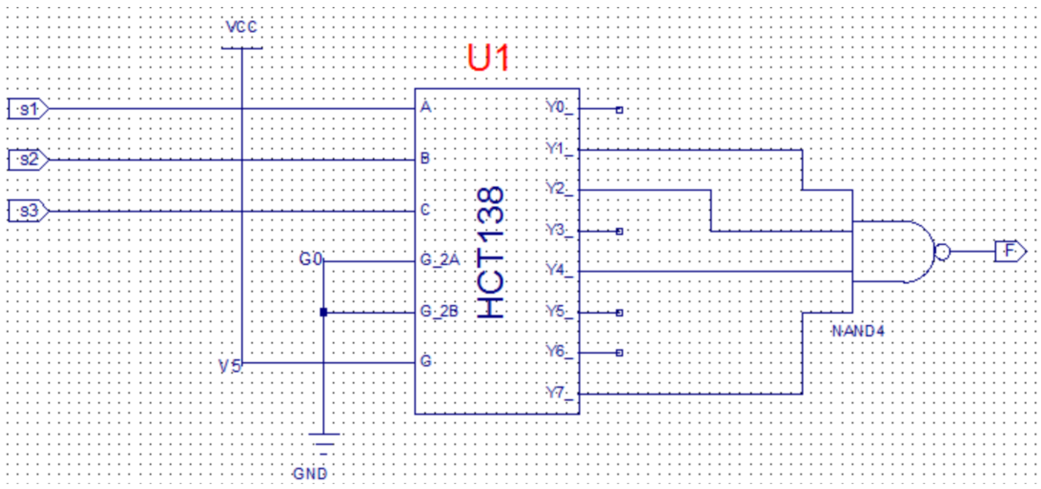
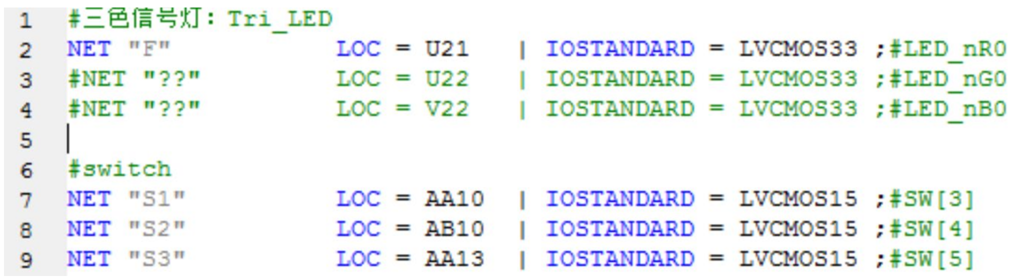


Figure 13 Principle diagram

Step 4: Modify UCF according to the new design



```
#三色信号灯: Tri_LED
NET "F"              LOC = U21    | IOSTANDARD = LVCMOS33 ;#LED_nR0
#NET "??"            LOC = U22    | IOSTANDARD = LVCMOS33 ;#LED_nG0
#NET "??"            LOC = V22    | IOSTANDARD = LVCMOS33 ;#LED_nB0

#switch
NET "S1"             LOC = AA10   | IOSTANDARD = LVCMOS15 ;#SW[3]
NET "S2"             LOC = AB10   | IOSTANDARD = LVCMOS15 ;#SW[4]
NET "S3"             LOC = AA13   | IOSTANDARD = LVCMOS15 ;#SW[5]
```

Figure 14 Modify UCF file

Step 5: Synthesize, Constraint and Implement, Generate Programming File as project 1.

5. Results and Analysis

5.1 Variable Decoder 74LS138

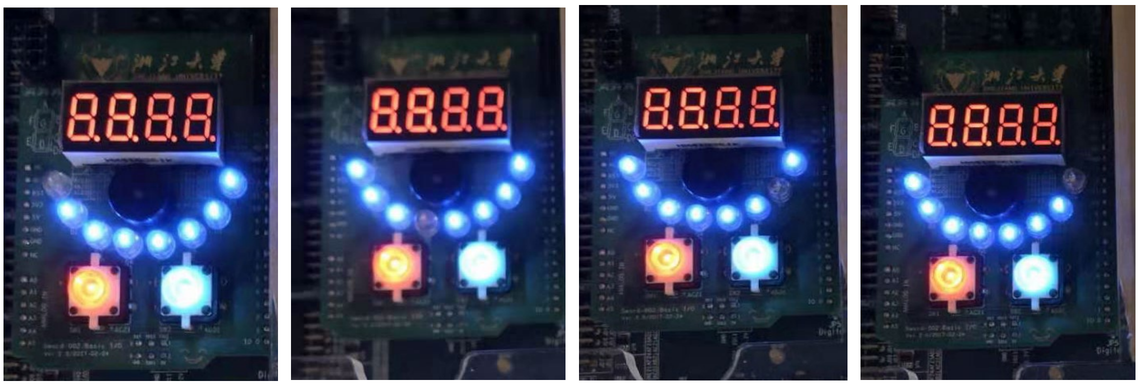
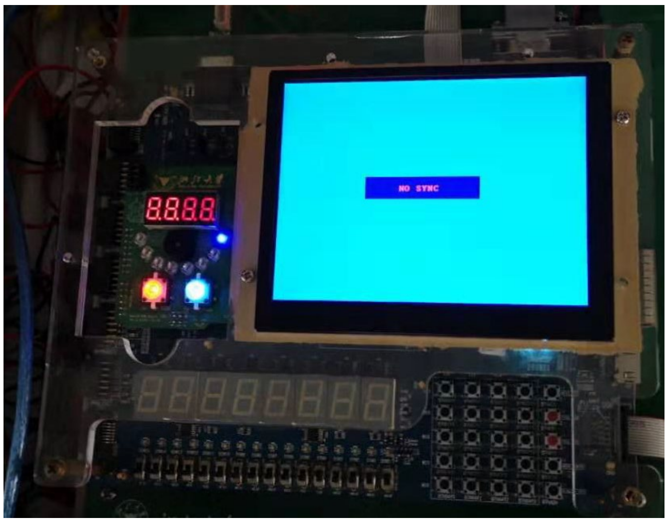


Figure 15 Four examples of Decoder 74LS138

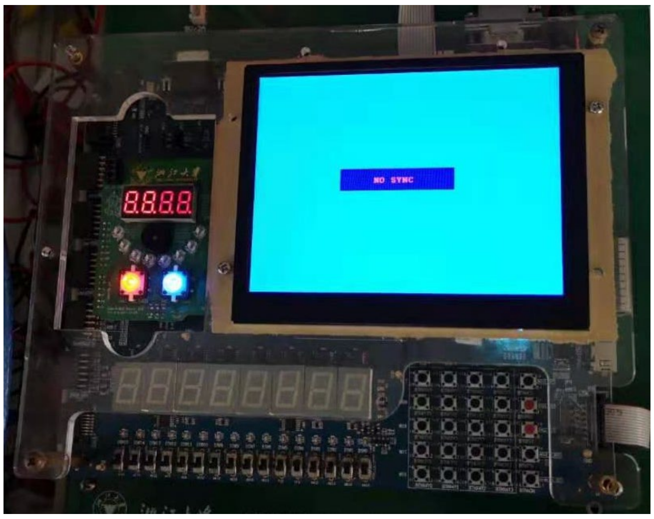
In the pictures above, light 0, 3, 6 and 7 is off respectively, when the switch is 000, 011, 110, 111.

5.2 Lamp Control

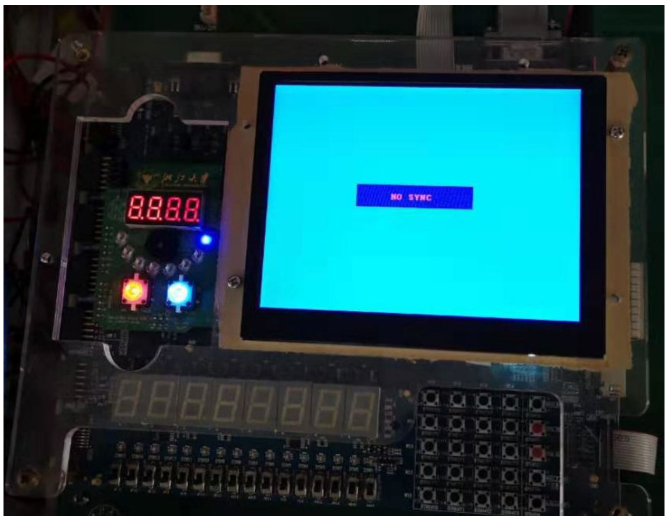
(1) 1 switch on, light on



(2) 2 switches on, light off



(3) 3 switches on, light on



6. Discussion and Revision

This experiment is the first one I completed on the development board with ISE. With the fourth experiment as the basis, this experiment is not very difficult. But because of the unfamiliarity with the software, I spent a lot of time on drawing schematic and simulation. In addition, during the experiment, I relied too much on the guidance on the slides, so I did not have a general concept of the whole experiment. I hope in the future experiments, I will be able to use the ISE software more skillfully and at the same time deepen my understanding of the experimental ideas.

Lab 6 – Design of 7-segment Code Display Decoder

Name: 余若涵 Student ID: 3180105412 Major: Computer Science and Technology
Course: Logic and Computer Design Fundamentals Groupmate: 吴晗晗
Date: 2019-10-24 Laboratory: East4-509, Zijingang Instructor: 洪奇军

1. Purpose and Requirements

1.1 Master the display principle of seven digital tubes.
1.2 Master the design of seven segment code display and decoding.
1.3 Master the scanning and display control of multi position nixie tube.
1.4 More familiar with ISE platform and use schematic diagram to learn Verilog HDL language comprehensively

2. Experiment Tasks and Theory

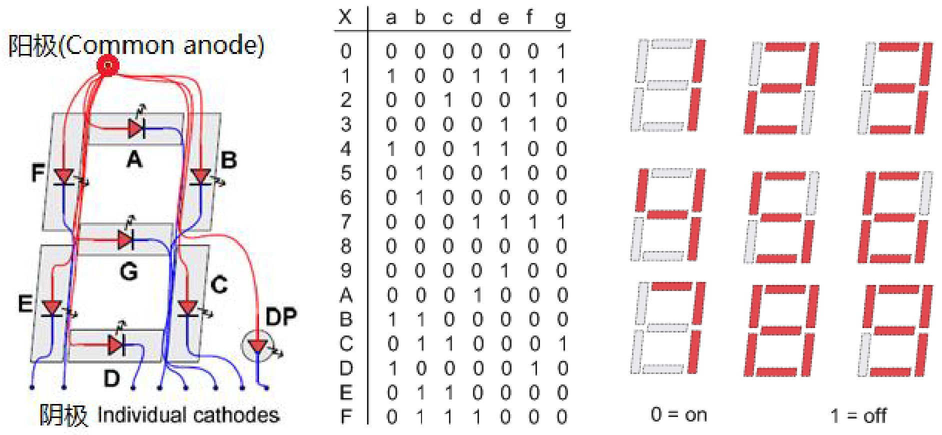
2.1 Experiment Tasks

2.1.1 Design a general hex seven segment display decoding circuit.
2.1.2 Simulate test and package mc14495 compatible display decoder.
2.1.3 Design and implement dynamic scanning display of four hexadecimal numbers.
2.1.4 Design and implement the static display of octal hexadecimal number*.
2.1.5 Learn HDL description method of display decoding circuit.

2.2 Experiment Principle

2.2.1 Structure of 7-segment display

Digital display device composed of 7+1 LEDs. Each LED shows a segment of the number and the other is a decimal point. The positive (negative) of the LED is connected together, and the other end is used as the lighting control.



X	a	b	c	d	e	f	g
0	0	0	0	0	0	0	1
1	1	0	0	1	1	1	1
2	0	0	1	0	0	1	0
3	0	0	0	0	1	1	0
4	1	0	0	1	1	0	0
5	0	1	0	0	1	0	0
6	0	1	0	0	0	0	0
7	0	0	0	1	1	1	1
8	0	0	0	0	0	0	0
9	0	0	0	0	1	0	0
A	0	0	0	1	0	0	0
B	1	1	0	0	0	0	0
C	0	1	1	0	0	0	1
D	1	0	0	0	0	1	0
E	0	1	1	0	0	0	0
F	0	1	1	1	0	0	0

Figure 1 Principle of 7-segment display

2.2.2 Hex 7-segment Decoder

In this experiment, the seven-segment digital tube module showing the number is controlled by common yang, the positive poles of the LEDs are connected together, and the other end is used as the lighting control. When the negative polarity = 0, the light is illuminated.

Hex	$D_3D_2D_1D_0$	BI/LE	a	b	c	d	e	f	g	p
0	0 0 0 0	1	0	0	0	0	0	0	1	p
1	0 0 0 1	1	1	0	0	1	1	1	1	p
2	0 0 1 0	1	0	0	1	0	0	1	0	p
3	0 0 1 1	1	0	0	0	0	1	1	0	p
4	0 1 0 0	1	1	0	0	1	1	0	0	p
5	0 1 0 1	1	0	1	0	0	1	0	0	p
6	0 1 1 0	1	0	1	0	0	0	0	0	p
7	0 1 1 1	1	0	0	0	1	1	1	1	p
8	1 0 0 0	1	0	0	0	0	0	0	0	P
9	1 0 0 1	1	0	0	0	0	1	0	0	P
A	1 0 1 0	1	0	0	0	1	0	0	0	P
B	1 0 1 1	1	1	1	0	0	0	0	0	P
C	1 1 0 0	1	0	1	1	0	0	0	1	P
D	1 1 0 1	1	1	0	0	0	0	1	0	P
E	1 1 1 0	1	0	1	1	0	0	0	0	P
F	1 1 1 1	1	0	1	1	1	0	0	0	P
X	x x x x	0	1	1	1	1	1	1	1	1

Figure 2 Truth table of MC14495

According to the truth table, we can get the corresponding logic circuit expression, and then draw the circuit diagram.

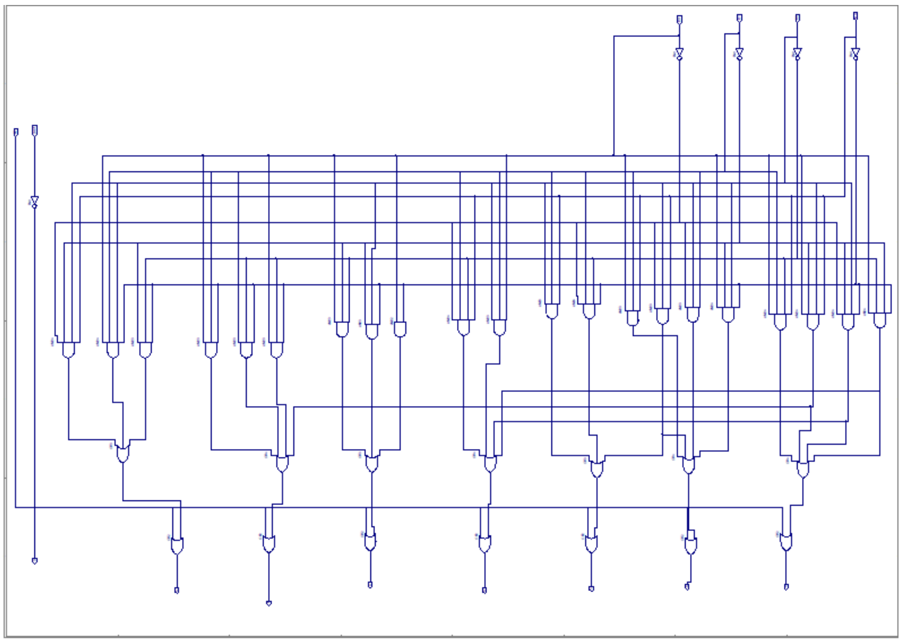


Figure 3 Circuit diagram of MC14495

2.2.3 Four-digit seven-segment dynamic display control

To implement dynamic scan display, we used the onboard clock: clk(50MHz) as the counter clock, which is divided into the control terminal of the data selector and used as the digital tube scan signal. The crossover coefficient of the counter should be appropriate and the eyes can be comfortable. Implement conditional output circuit with if_then of case statement.

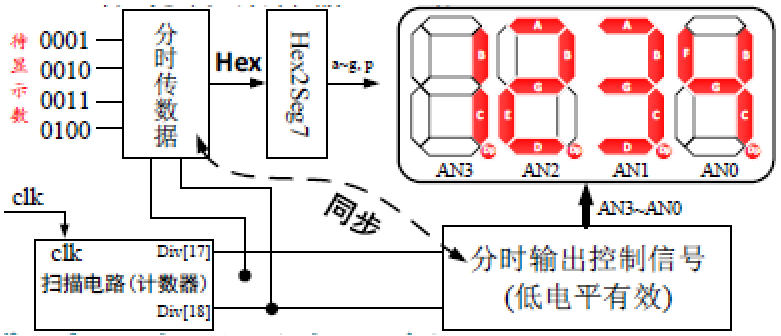


Figure 4 Function of 2-4 Variable Decoder Figure 5 Pins of 2-4 Variable Decoder

2.2.4 Auxiliary Module: Clock Count Divider

A 32-bit clock count divider that outputs a 2-232 divide signal that can be used for general asynchronous clock signals. It can be used in this experiment as a dynamic scanner.

```
` module   clkdiv(input clk,
                  input rst,
                  output reg[31:0]clkdiv
                  );
//clkck divider-时钟分频器
   always @ (posedge clk or posedge rst) begin
      if(rst) clkdiv <= 0;
      else clkdiv <= clkdiv + 1'b1;
   end

endmodule
```

Figure 5 Code of clkdiv

3. Experiment Instruments and Materials

1. Computer with Xilinx ISE 14.4 or above	1 unit
2. Sword experiment system	1 unit

4. Experiment Procedure and Operations

4.1 Design Project1: Hex27Seg

Design to implement Hex to 7-segment display decoder

Step 1: Create a new FPGA project named Hex27Seg.

Step 2: Create a Schematic file named MC14495_ZJU.

Design MC14495_ZJU according to the principle diagram in chapter 2.

Check Design Rules and View HDL Functional Model.

Step 3: Simulation.

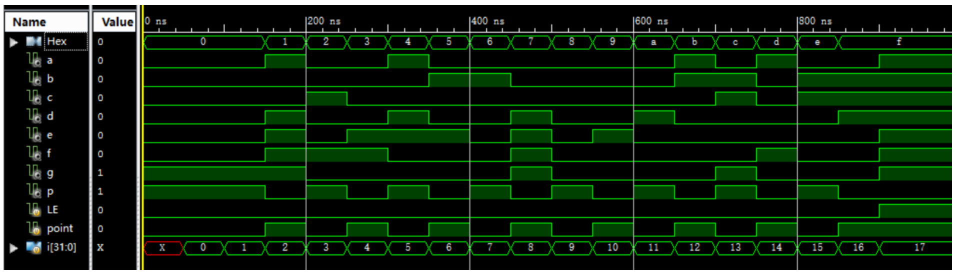


Figure 6 Result of simulation of MC14495_ZJU.

Create Schematic Symbol

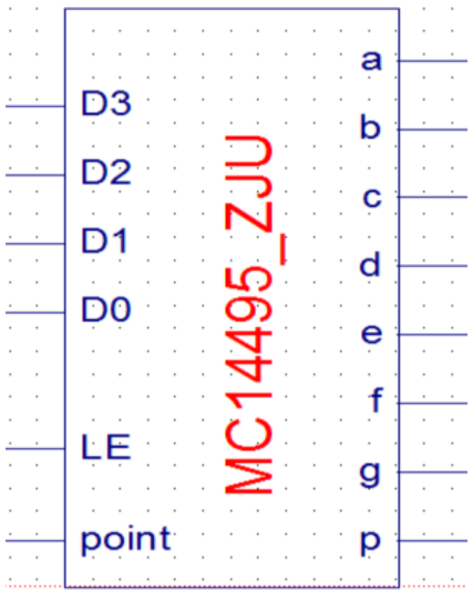


Figure 7 Symbol of MC14495_ZJU

Step 4: Create a Schematic file named Hex27Seg_sch and set it as top module.

Design Hex27Seg_sch with module MC14495_ZJU

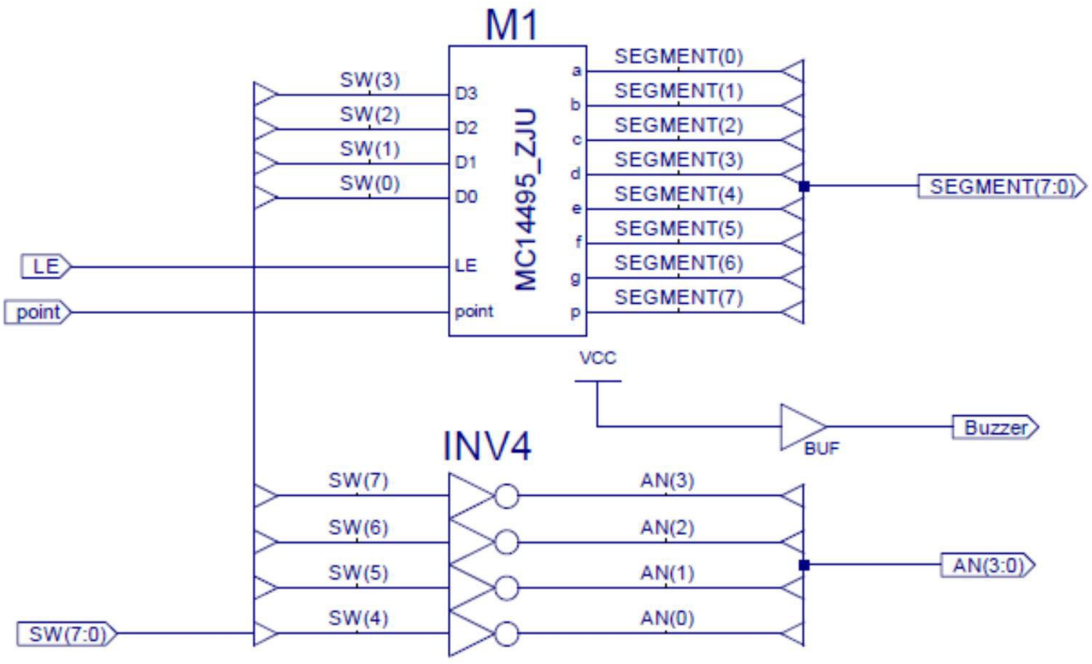


Figure 8 Diagram of Hex27Seg_sch

Check Design Rules and View HDL Functional Model

Step 5: Constraint and implementation

Create a Implementation Constraints File associated with

Hex27Seg_sch. Input the UCF file.

```
NET"SW[0]"LOC=AA10 |IOSTANDARD=LVCMOS15;
NET"SW[1]"LOC=AB10 | IOSTANDARD=LVCMOS15;
NET"SW[2]"LOC=AA13 | IOSTANDARD=LVCMOS15;
NET"SW[3]"LOC=AA12 | IOSTANDARD=LVCMOS15;
NET"SW[4]"LOC=Y13 | IOSTANDARD=LVCMOS15;
NET"SW[5]"LOC=Y12 | IOSTANDARD=LVCMOS15;
NET"SW[6]"LOC=AD11 | IOSTANDARD=LVCMOS15;
NET"SW[7]"LOC=AD10 | IOSTANDARD=LVCMOS15;

NET"BTN[0]"LOC=AF13 |IOSTANDARD=LVCMOS15;
NET"BTN[1]"LOC=AF10 |IOSTANDARD=LVCMOS15;

NET"SEGMENT[0]"LOC=AB22 |IOSTANDARD=LVCMOS33;#a
NET"SEGMENT[1]"LOC=AD24 |IOSTANDARD=LVCMOS33;#b
NET"SEGMENT[2]"LOC=AD23 |IOSTANDARD=LVCMOS33;#c
NET"SEGMENT[3]"LOC=Y21 |IOSTANDARD=LVCMOS33;#d
NET"SEGMENT[4]"LOC=W20 |IOSTANDARD=LVCMOS33;#e
NET"SEGMENT[5]"LOC=AC24 |IOSTANDARD=LVCMOS33;#f
NET"SEGMENT[6]"LOC=AC23 |IOSTANDARD=LVCMOS33;#g
NET"SEGMENT[7]"LOC=AA22 |IOSTANDARD=LVCMOS33;#point

NET"AN[0]"LOC=AD21 |IOSTANDARD=LVCMOS33;
NET"AN[1]"LOC=AC21 |IOSTANDARD=LVCMOS33;
NET"AN[2]"LOC=AB21 |IOSTANDARD=LVCMOS33;
NET"AN[3]"LOC=AC22 |IOSTANDARD=LVCMOS33;
```

Figure 9 Code of UCF file

Step 7: Implement Design and Generate Programming File.

Step 8: Download FPGA programing file

4.2 Design Project 2: Heg427Seg

Step 1: Create a new project and a Schematic file named Hex427Seg_sch.

Step 2: Add a copy of file MC14495_ZJU.sym and MC14495.sch.

Step 3: Design dispsync.v and clkdiv.v with Verilog HDL and create Schematic Symbol.

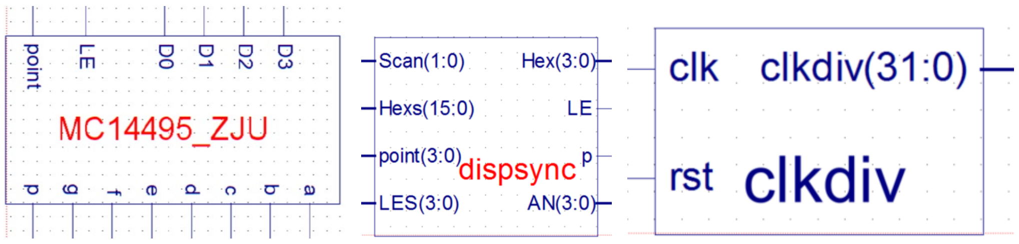


Figure 10 Symbol of MC14495_ZJU, dispsync and clkdiv.

Step 4: Design Hex427Seg_sch.sch as the following circuit diagram.

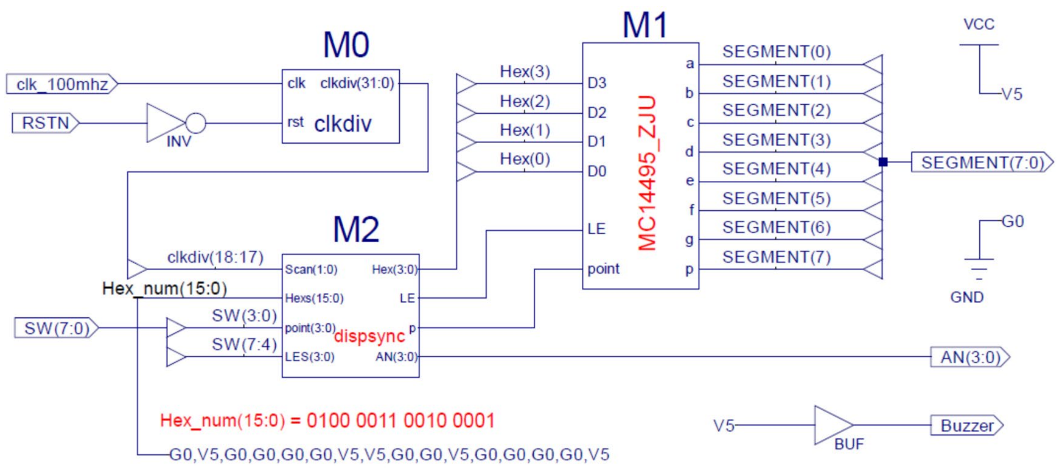


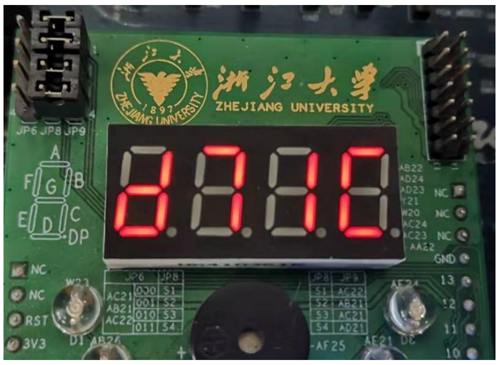
Figure 11 Circuit of Hex427Seg_sch

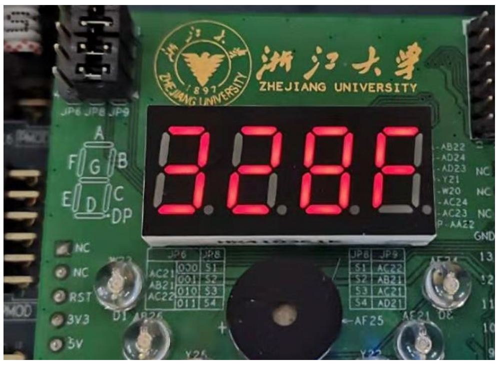
Step 5: Modify the UCF file.

Step 6: Synthesize, Constraint and Implement, Generate Programming File and Download FPGA programing file.

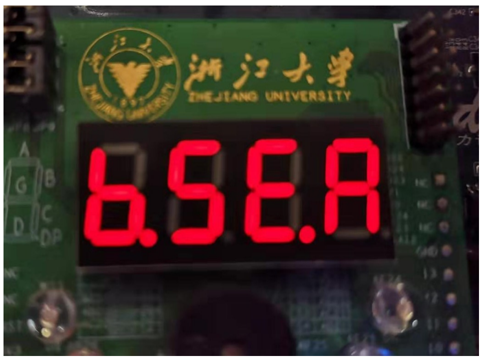
5. Results and Analysis

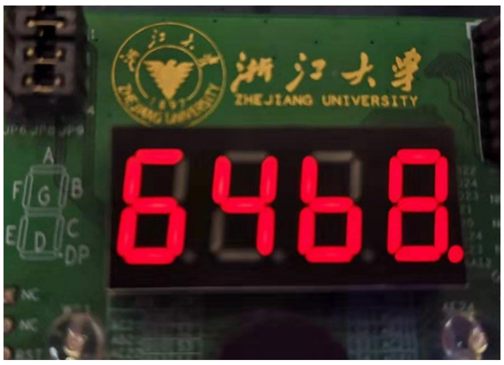
(1) 4 different numbers (Hex included)





(2) Decimal point included





(3) 2 numbers displayer

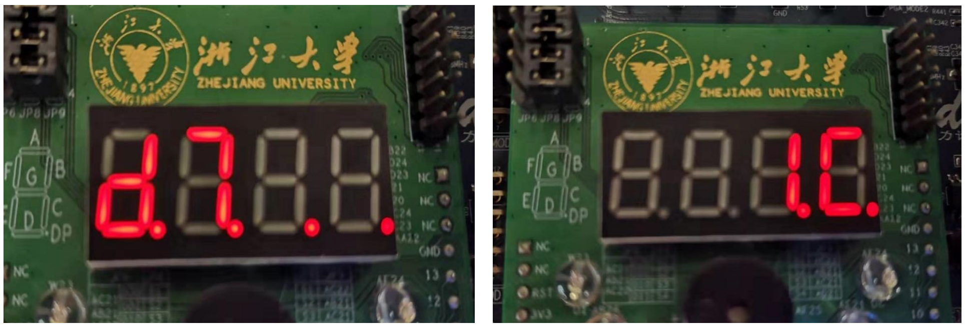


Figure 12 Result of Hex427Seg

In the pictures above, the function of Hex427Seg is implemented. By controlling AN, each bit is individually displayed with a visual residue, and different numbers can be displayed respectively. At the same time, it can be controlled whether each digit is displayed and whether each decimal point is displayed.

6. Discussion and Revision

This experiment is based on the previous one, so it’s not very difficult. Looking back at this experiment, I found that the following parts need more attention. The first one is the connection of the MC14495 component. Although the logic expression is not complicated, the actual connection is very cumbersome. For the first time, I connected a wrong wire, which brought great trouble in later experiment. The second is the auxiliary module clkdiv.v, which is my first module written in the Verilog language. The third is the UCF file. The definition of the pins is very complicated, and the code is long and dazzling. I hope I will become more proficient in the future, and don't spend too much time and energy on the basic parts of connecting wires and typing code.

Lab 7 – Design and Application of Data Selector

Name: 余若涵 Student ID: 3180105412 Major: Computer Science and Technology
Course: Logic and Computer Design Fundamentals Groupmate: 吴晗晗
Date: 2019-10-31 Laboratory: East4-509, Zijingang Instructor: 洪奇军

1. Purpose and Requirements

1.1 Master the logical structure and logic of the data selector.
1.2 Master the use of data selectors.
1.3 Using a data selector to implement a combined circuit module.
1.4 Establish an input and output environment for subsequent experiments.
1.5 Understand the principle of button debounce and implementation method.
1.6 Further familiar with the ISE platform, learn Verlog-HDL.

2. Experiment Tasks and Theory

2.1 Experiment Tasks

2.1.1 Design a 4-bit four-select data selector based on the schematic diagram and simulate the test.
2.1.2 Extended design 8/32 bit eight-select data selector and simulation test.
2.1.3 Data selector application (optional).
2.1.4 Data input real-time display.
Design data input module, package experiment four-digit seven-segment display Seg7_Dev16.
2.1.5 Build a follow-up experimental input and output environment (after class practice).
Extended display device Seg7_Dev16 is 32-bit.
Design 32-bit octal data channel.

2.2 Experiment Principle

2.2.1 4 -choice multiplexer: MUX4T1

According to its truth table, we can get its circuit diagram.

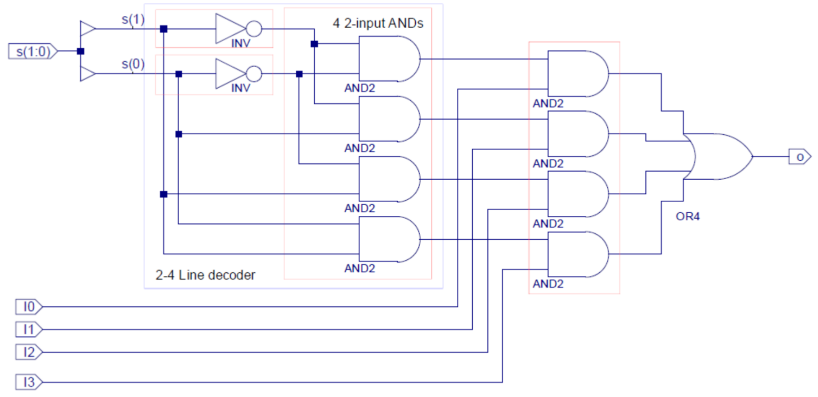


Figure 1 Circuit Diagram of MUX4T1

2.2.2 4-bit 4 -choice multiplexer extension: MUX441

It is a extended module of MUX4T1.

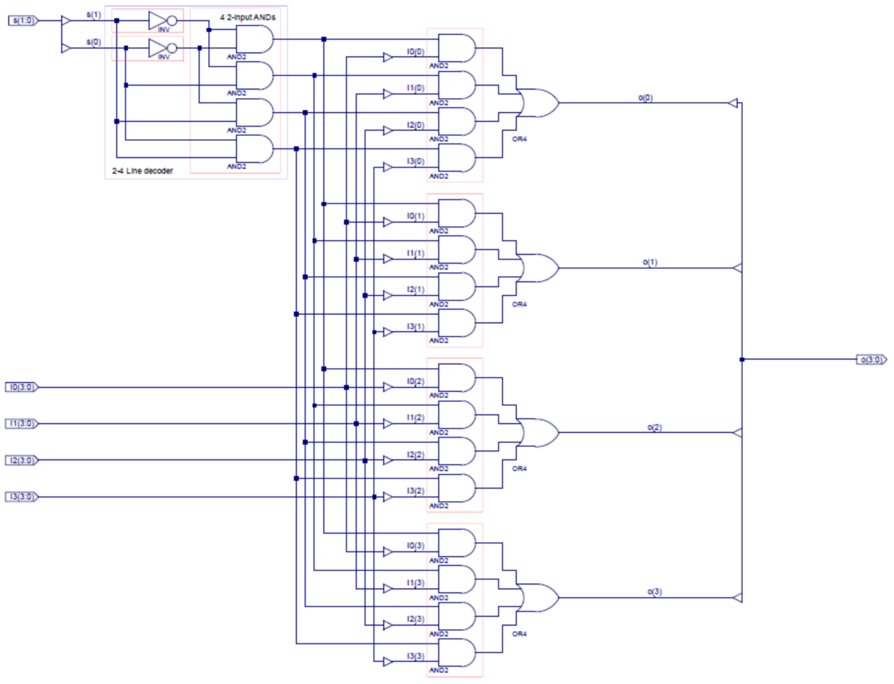


Figure 2 Circuit Diagram of MUX441

2.2.3 8-bit 8-choice multiplexer extension

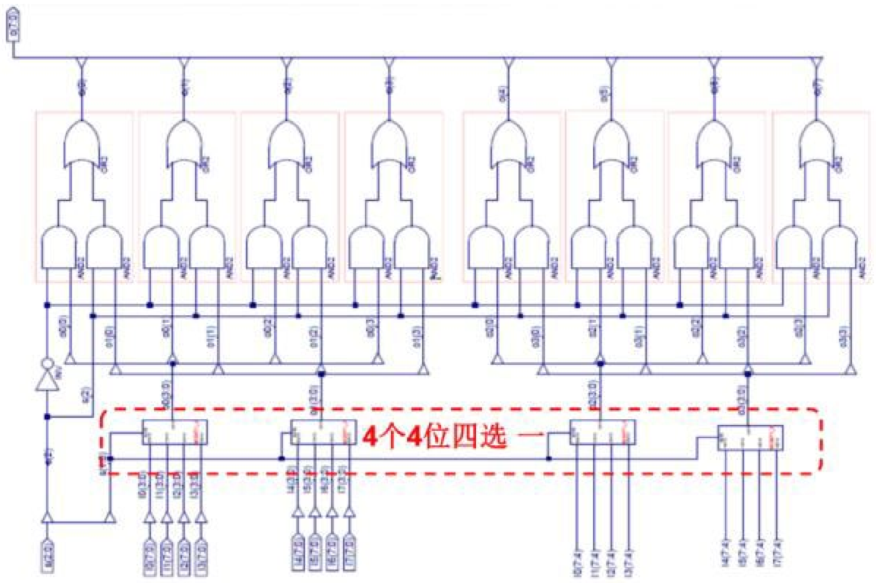


Figure 3 Circuit Diagram of 8bit MUX8to1

2.2.4 32-bit 8-choice multiplexer extension

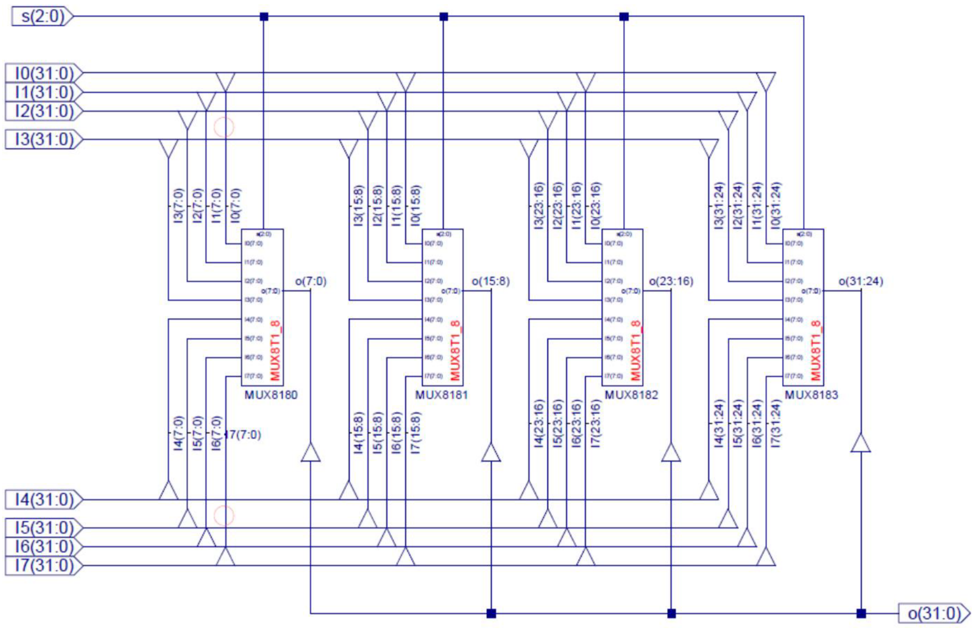


Figure 4 Circuit Diagram of 32bit MUX8to1

2.2.5 Multiple Selector Application: 4-bit display seven-segment display

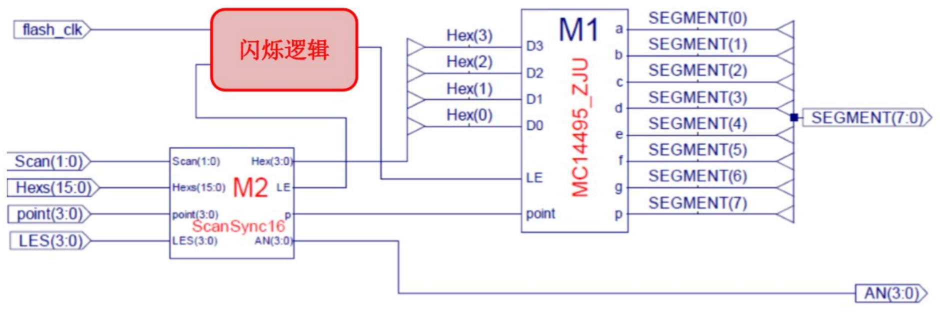


Figure 5 Packaged 4-bit 7-segment display

2.2.6 Top Logic

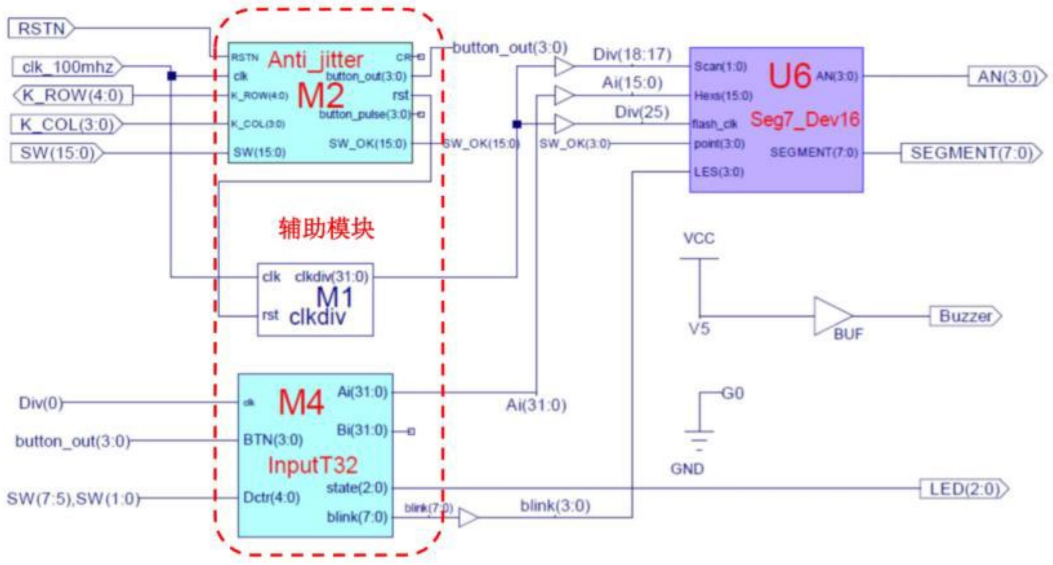


Figure 6 Schematic file of Top logic

2.2.7 Auxiliary Module:

Clock Count Divider: clkdiv

A 32-bit clock count divider that outputs a 2-232 divide signal that can be used for general asynchronous clock signals. It can be used in this experiment as a dynamic scanner.

2.2.8 Button anti jitter module: anti_jitter

```
`module anti_jitter(input wire clk,
                    input wire RSTN,
                    input wire [3:0] K_COL,
                    input wire [15:0] SW,
                    output reg [3:0] button_out,
                    output reg [3:0] button_pulse,
                    output reg [15:0] SW_OK,
                    output [4:0] K_ROW,
                    output reg CR,
                    output reg rst

                    );

      reg [31:0] counter, rst_counter;
      reg [4:0] btn_temp;
      reg [15:0] sw_temp;
      reg pulse;

      wire [4:0] button = {~RSTN,~K_COL[3:0]};
      assign K_ROW = {SW[15:11]};

      always @(posedge clk) begin
         btn_temp <= button;
         sw_temp <= SW;
         if (btn_temp != button || sw_temp != SW) begin
               counter <= 32'h00000000;
               rst_counter <= 0; pulse <= 0;
               end
         else if (counter < 100000)
               counter <= counter + 1;
            else begin
               button_out <= button[3:0];
               CR <= ~RSTN; SW_OK <= SW;
               pulse <= 1;
                  if(!pulse)
                        button_pulse <= button;
                  else
                        button_pulse <= 0;
                  if (button[4] && rst_counter < 200000000)
                     rst_counter <= rst_counter + 1;
                  else
                     rst <= ~RSTN;
            end
      end
endmodule
```

2.2.9 Real-time data input module: Input32

```
` module   InputT32(input clk,
                 input [3:0] BTN,
                 input [4:0] Dctr,
                 output reg[31:0] Ai=32'h87654321,
                 output reg[31:0] Bi=32'h12345678,
                 output reg[2:0] state=100,
                 output reg[7:0] blink
                 );

   reg [3:0] samp;
   reg [1:0] get_01, get_23;

   assign push01 = BTN[0] | BTN[1];
   assign push23 = BTN[2] | BTN[3];

   always@(posedge clk) begin
      get_01 <= {get_01[0], push01};
      get_23 <= {get_23[0], push23};
   end

   always@(posedge clk) begin
      if (Dctr[4:2]<=2'b01 && get_01==2'b01)
```

```
          if (BTN[0]) state <= state+1;
                 else   state <= state-1;
       else state <= state;
    end

    assign HexUP16 = 0;
    always
    blink <= 4'b00000000;

    always@(posedge clk) begin
       if (Dctr[4:2] == 3'b000 && get_23==2'b01) begin
          case({HexUP16, state[1:0]})
       3'b000: if(BTN[2])   Ai[3:0] <= Ai[3:0] + 1;
                 else   Ai[3:0] <= Ai[3:0] - 1;
       3'b001: if(BTN[2])   Ai[7:4] <= Ai[7:4] + 1;
                 else   Ai[7:4] <= Ai[7:4] - 1;
       3'b010: if(BTN[2])
                 Ai[11:8] <= Ai[11:8] + 1;
              else
                 Ai[11:8] <= Ai[11:8] - 1;
       3'b011: if(BTN[2])
                 Ai[15:12] <= Ai[15:12] + 1;
              else
                 Ai[15:12] <= Ai[15:12] - 1;
       3'b100: if(BTN[2])
                 Ai[19:16] <= Ai[19:16] + 1;
              else
                 Ai[19:16] <= Ai[19:16] - 1;
       3'b101:
              if(BTN[2])
                 Ai[23:20] <= Ai[23:20] + 1;
              else
                 Ai[23:20] <= Ai[23:20] - 1;
       3'b110:
              if(BTN[2])
                 Ai[27:24] <= Ai[27:24] + 1;
              else
                 Ai[27:24] <= Ai[27:24] - 1;
       3'b111:
              if(BTN[2])
                 Ai[31:28] <= Ai[31:28] + 1;
              else
                 Ai[31:28] <= Ai[31:28] - 1;
       endcase
    end

else if (Dctr[4:2] == 4'b001 && get_23==2'b01) begin
          case({HexUP16, state[1:0]})
       3'b000: if(BTN[2])   Bi[3:0] <= Bi[3:0] + 1;
                 else   Bi[3:0] <= Bi[3:0] - 1;
       3'b001: if(BTN[2])   Bi[7:4] <= Bi[7:4] + 1;
                 else   Bi[7:4] <= Bi[7:4] - 1;
       3'b010: if(BTN[2])
                 Bi[11:8] <= Bi[11:8] + 1;
              else
                 Bi[11:8] <= Bi[11:8] -   1;
       3'b011: if(BTN[2])
                 Bi[15:12] <= Bi[15:12] + 1;
              else
                 Bi[15:12] <= Bi[15:12] - 1;
       3'b100: if(BTN[2])
                 Bi[19:16] <= Bi[19:16] + 1;
              else
                 Bi[19:16] <= Bi[19:16] -1;
       3'b101:
              if(BTN[2])
                 Bi[23:20] <= Bi[23:20] + 1;
              else
                 Bi[23:20] <= Bi[23:20] - 1;
       3'b110:
              if(BTN[2])
                 Bi[27:24] <= Bi[27:24] + 1;
              else
                 Bi[27:24] <= Bi[27:24] - 1;
```

```
        3'b111:
              if(BTN[2])
                 Bi[31:28] <= Bi[31:28] + 1;
              else
                 Bi[31:28] <= Bi[31:28] - 1;
           endcase
        end
     end

 endmodule
```

3. Experiment Instruments and Materials

1. Computer with Xilinx ISE 14.4 or above	1 unit
2. Sword experiment system	1 unit

4. Experiment Procedure and Operations

4.1 Design Project1: Ex07-MUX441

Design to implement 4-bit 4-choice multiplexer

Step 1: Create a new FPGA project named Ex07-MUX441.

Step 2: Create a Schematic file named MUX441_sch.

Design MUX441_sch according to the principle diagram in chapter 2.

Check Design Rules and View HDL Functional Model.

Step 3: Simulation. Create Schematic Symbol of MUX441_sch.

Step 4: Create a Schematic file named MUX4T1_32b_sch.

Design MUX4T1_32b_sch according to the principle diagram in chapter 2.

Check Design Rules and View HDL Functional Model.

Step 5: Simulation. Create Schematic Symbol of MUX4T1_32b_sch.

4.2 Design Project 2: I2Disp

Step 1: Create a new project named I2Disp and a Schematic file named Seg7_Dev16.

Step 2: Design clkdiv.v and anti_jitter.v and Input32.v with Verilog HDL according to the code in Chapter 2 and create Schematic Symbol.

Step 3: Design the top module Seg7_Dev16 according to the circuit diagram in chapter2. Use the clkdiv, anti_jitter and Input32 module

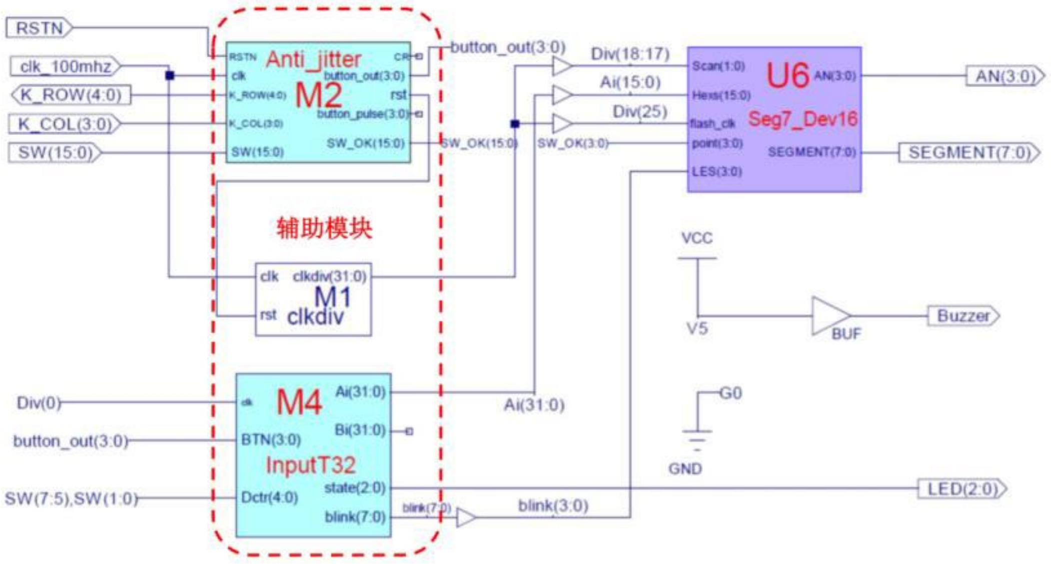


Figure 7 Top logic of project I2Disp.

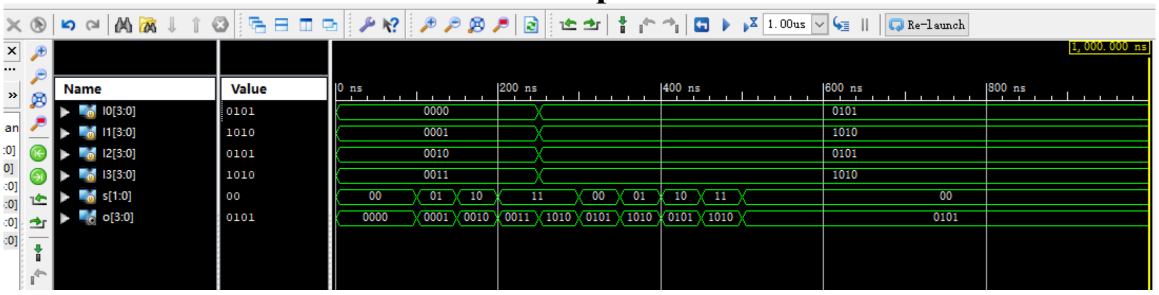
Step 4: Edit the UCF file.

```
#Hex4i7Seg
#SYSTEM CLOCK
NET "clk_100mhz" LOC = AC18 | IOSTANDARD = LVCMOS18;
NET "clk_100mhz" TNM_NET = TM_CLK;
TIMESPEC TS_CLK_100M = PERIOD "TM_CLK" 10ns HIGH 50%;

#switch
NET "SW[0]"  LOC = AA10 | IOSTANDARD = LVCMOS15;
NET "SW[1]"  LOC = AB10 | IOSTANDARD = LVCMOS15;
NET "SW[2]"  LOC = AA13 | IOSTANDARD = LVCMOS15;
NET "SW[3]"  LOC = AA12 | IOSTANDARD = LVCMOS15;
NET "SW[4]"  LOC = Y13  | IOSTANDARD = LVCMOS15;
NET "SW[5]"  LOC = Y12  | IOSTANDARD = LVCMOS15;
NET "SW[6]"  LOC = AD11 | IOSTANDARD = LVCMOS15;
NET "SW[7]"  LOC = AD10 | IOSTANDARD = LVCMOS15;
NET "SW[8]"  LOC = AE10 | IOSTANDARD = LVCMOS15;
NET "SW[9]"  LOC = AE12 | IOSTANDARD = LVCMOS15;
NET "SW[10]" LOC = AF12 | IOSTANDARD = LVCMOS15;
NET "SW[11]" LOC = AE8  | IOSTANDARD = LVCMOS15;
NET "SW[12]" LOC = AF8  | IOSTANDARD = LVCMOS15;
NET "SW[13]" LOC = AE13 | IOSTANDARD = LVCMOS15;
NET "SW[14]" LOC = AF13 | IOSTANDARD = LVCMOS15;
NET "SW[15]" LOC = AF10 | IOSTANDARD = LVCMOS15;

#Reset or CR
NET "RSTN" LOC = W13 | IOSTANDARD = LVCMOS18;

#Keyboard Row
NET "K_ROW[0]" LOC = V17 | IOSTANDARD = LVCMOS18;
NET "K_ROW[1]" LOC = W18 | IOSTANDARD = LVCMOS18;
NET "K_ROW[2]" LOC = W19 | IOSTANDARD = LVCMOS18;
NET "K_ROW[3]" LOC = W15 | IOSTANDARD = LVCMOS18;
NET "K_ROW[4]" LOC = W16 | IOSTANDARD = LVCMOS18;

#Keyboard Column
NET "K_COL[0]" LOC = V18 | IOSTANDARD = LVCMOS18;
NET "K_COL[1]" LOC = V19 | IOSTANDARD = LVCMOS18;
NET "K_COL[2]" LOC = V14 | IOSTANDARD = LVCMOS18;
NET "K_COL[3]" LOC = W14 | IOSTANDARD = LVCMOS18;

#3-Color LED
NET "LED[0]" LOC = U21 | IOSTANDARD = LVCMOS33;
NET "LED[1]" LOC = U22 | IOSTANDARD = LVCMOS33;
NET "LED[2]" LOC = V22 | IOSTANDARD = LVCMOS33;

#Arduino-Sword-002-Basic IO
NET "SEGMENT[0]" LOC = AB22 | IOSTANDARD = LVCMOS33;
NET "SEGMENT[1]" LOC = AD24 | IOSTANDARD = LVCMOS33;
NET "SEGMENT[2]" LOC = AD23 | IOSTANDARD = LVCMOS33;
NET "SEGMENT[3]" LOC = Y21  | IOSTANDARD = LVCMOS33;
NET "SEGMENT[4]" LOC = W20  | IOSTANDARD = LVCMOS33;
NET "SEGMENT[5]" LOC = AC24 | IOSTANDARD = LVCMOS33;
NET "SEGMENT[6]" LOC = AC23 | IOSTANDARD = LVCMOS33;
NET "SEGMENT[7]" LOC = AA22 | IOSTANDARD = LVCMOS33;

NET "AN[0]" LOC = AD21 | IOSTANDARD = LVCMOS33;
NET "AN[1]" LOC = AC21 | IOSTANDARD = LVCMOS33;
NET "AN[2]" LOC = AB21 | IOSTANDARD = LVCMOS33;
NET "AN[3]" LOC = AC22 | IOSTANDARD = LVCMOS33;endmodule
```

Step 5: Synthesize, Constraint and Implement, Generate Programming File and Download FPGA programing file.

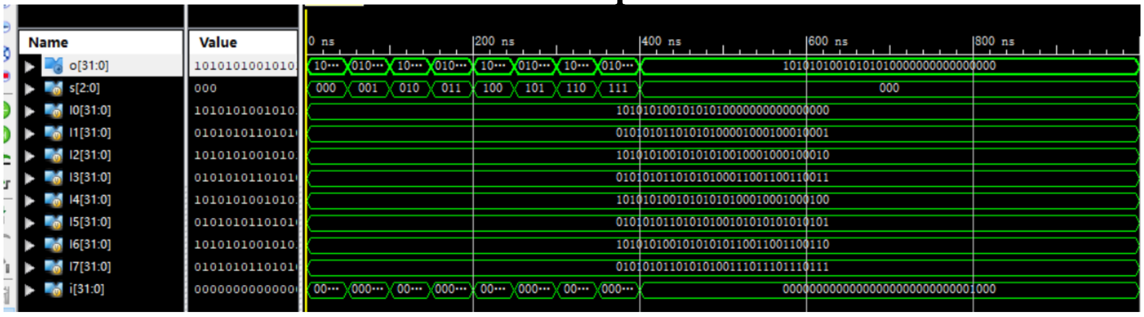
5. Results and Analysis

5.1.1 Simulation of 4-bit 4-choice multiplexer



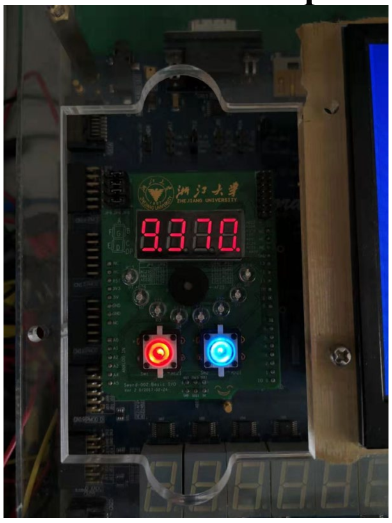
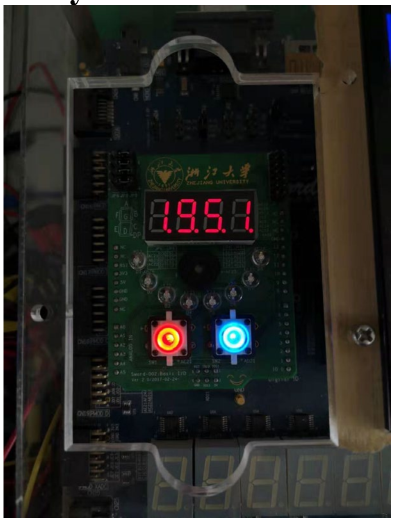
It can be seen from the picture that when s=00, I0 is selected, when s=01, I1 is selected, when s==10, I2 is selected, when s=11, I3 is selected. So, the module functions successfully.

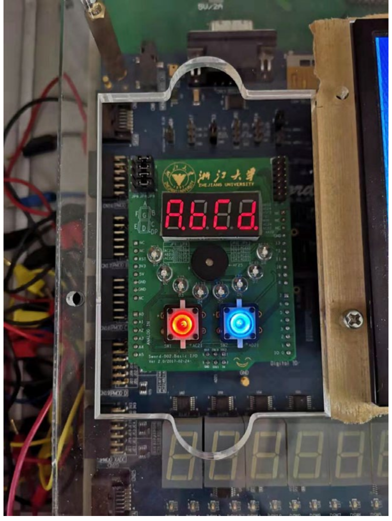
5.1.2 Simulation of 32-bit 8-choice multiplexer

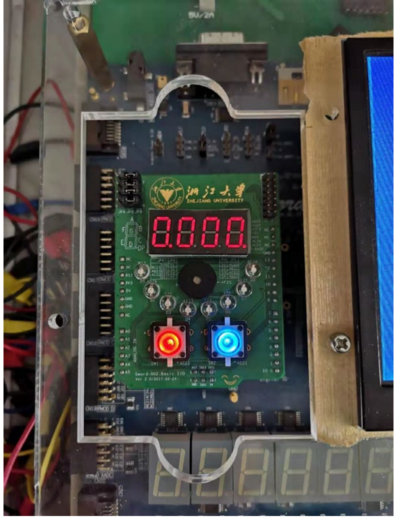


According to the simulation result shown in the picture, we can conclude that this module functions successfully.

5.1.3 Result in Sword experiment system.





In the pictures above, the function of I2Disp is implemented. Different number and displaying induvial have been implemented in experiment 6. Its different functions are shown in the chart below.

SW[7:5]=000, SW[15:11]=00000				Result			
BTN[3]	BTN[2]	BTN[1]	BTN[0]	SEG[3]	SEG[2]	SEG[1]	SEG[0]
Press				Selected digit+1			
	Press			Selected digit-1			
		Press		Select left digit			
			Press	Select right digit			
SW[7:5]=001, SW[15:11]=00000							
Press				No change			
	Press						
		Press					
			Press				
SW[3:0]=0000 -> SW[3:0]=1111				Selected point on			
SW[3:0]=1111 -> SW[3:0]=0000				Selected point off			

6. Discussion and Revision

Experiment 7 is an extension version based on experiment 6. In this experiment, we can implement the function of +1 and -1 of the selected digit by press corresponding button. The logic of module Input32 and anti_jitter is not difficult to understand. However, typing them and connect the wires took a great effort.