Lab5--The Design and Utilization of Variable Encoder

姓名：潘子曰 学号：3180105354 专业：计算机科学与技术
课程名称：逻辑与计算机设计基础实验 同组学生姓名：张佳文
试验时间：2019-09-26 实验地点：紫金港东4-509 指导老师：洪奇军

1. Objectives & Requirements

1. Gain the general knowing of the logical structure and functionality of variable encoders.
2. Realize **combining function** based on variable encoders.
3. Master the typical utilization of variable decoder (basic method of **address decoding**).
4. Gain the concept of **memory addressing**.
5. Design the circuit module based on **schematics**.
    - Get familiar with the **Hardware Description Language**.
6. Further knowing the **ISE platform** and **physical verification on experiment box**.

2. Contents & Principles

2.1 Tasks

Basic Tasks

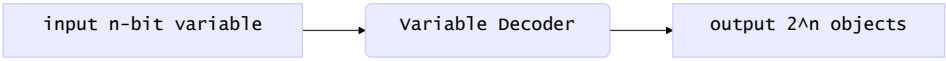
1. Design and realize **74HC138 decoder** module with schematic.
2. Realize the **lamp control** module with 74HC138 encoder.
3. Write the module simulation source code.

Advanced Tasks

4. Design 32×32 bit ROM IP core.
5. Use variable decoder to decode address:
    - Expand 4 same 8×16 ROM IP cores to realize 32×16 bit ROM.

2.2 Principles

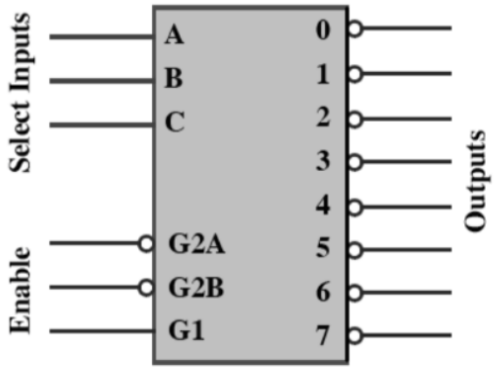
General Decoding



A **binary decoder** is a combinational logic circuit that converts binary information from the n coded inputs to a maximum of 2n unique outputs. They are used in a wide variety of applications, including data multiplexing and data demultiplexing, seven segment displays, and memory address decoding.

3-8 Variable Decoder Detail

Logic Functionality Table of 74LS138



Inputs						Output							
Enable			Select										
G2A	G2B	G1	C	B	A	$\overline{0}$	$\overline{1}$	$\overline{2}$	$\overline{3}$	$\overline{4}$	$\overline{5}$	$\overline{6}$	$\overline{7}$
1	X	X	X	X	X	1	1	1	1	1	1	1	1
X	1	X	X	X	X	1	1	1	1	1	1	1	1
X	X	0	X	X	X	1	1	1	1	1	1	1	1
0	0	1	0	0	0	0	1	1	1	1	1	1	1
0	0	1	0	0	1	1	0	1	1	1	1	1	1
0	0	1	0	1	0	1	1	0	1	1	1	1	1
0	0	1	0	1	1	1	1	1	0	1	1	1	1
0	0	1	1	0	0	1	1	1	1	0	1	1	1
0	0	1	1	0	1	1	1	1	1	1	0	1	1
0	0	1	1	1	0	1	1	1	1	1	1	0	1
0	0	1	1	1	1	1	1	1	1	1	1	1	0

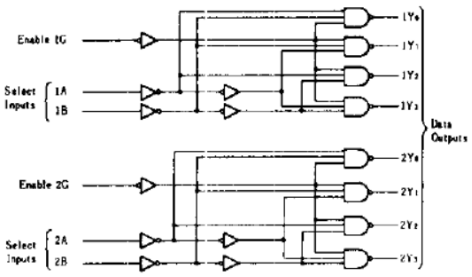
ATT: The logic level of 74LS138 is negative, thus we must slightly adjust out design.

Double 2-4 Variable Decoder

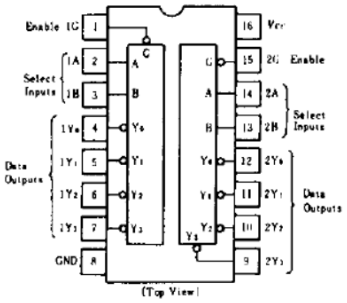
Logic Functionality Table of 74LS139

The HD74LS139 comprises two individual two-line-to-four-line decoder in a single package. The active-low enable input can be used as a data line in demultiplexing applications.

■BLOCK DIAGRAM



■PIN ARRANGEMENT



■FUNCTION TABLE

Inputs			Outputs			
Enable	Select					
G	B	A	Y_0	Y_1	Y_2	Y_3
H	×	×	H	H	H	H
L	L	L	L	H	H	H
L	L	H	H	L	H	H
L	H	L	H	H	L	H
L	H	H	H	H	H	L

H; high level, L; low level, X; irrelevant

ATT: The logic level of 74LS139 is negative, thus we must slightly adjust our design.

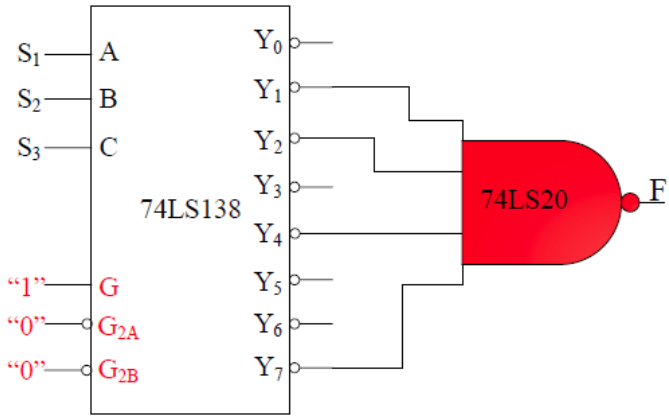
Realize Combinational Function with Variable Decoder

Use **OR** gate to output all the minterms.

*ATT: We must use a **NAND** gate instead of the **OR** gate, because the logic function of the modules we design is negative.*

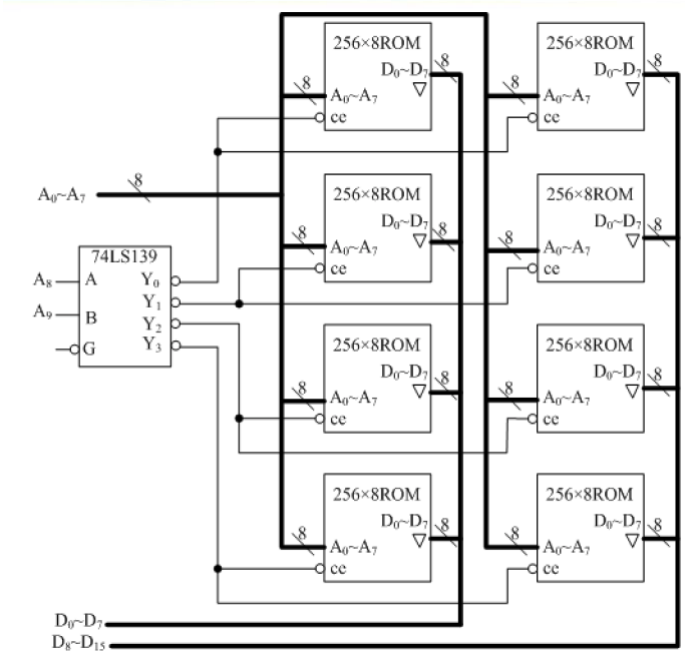
Take the lamp control design in experiment 1 as an example.

```
module lamp_Ctr1(s1, s2, s3, F);
    //Some declarations
    decoder_3_8 Decoder3_8(C, B, A, G, G2A,G2B, Y);
    nand node(F, Y[1], Y[2], Y[4] , Y[7]);
endmodule
```



Example of combinational circuit

Realize Address Decoding by Variable Decoder



Example of memory bank

2.3 Note

- The logic function of the module we use during this experiment is **NEGATIVE**.

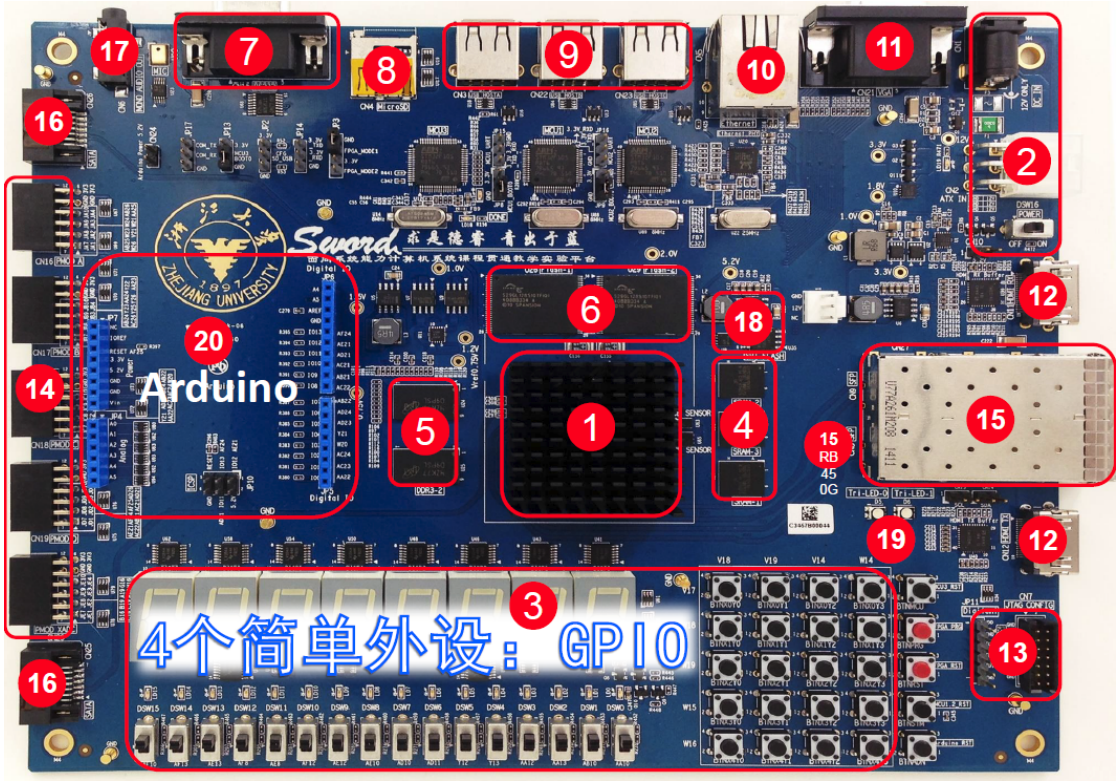
- Naming rules:
  - First character of the file name can **not** be numbers;
  - All the variables within the project must **not** contain any **hyphens**, instead , we can use underline.

3. Major Experiment Instruments

Equipment
Computer (Intel Core i5 or higher, 4GB memory or higher)
Sword circuit design box
Xilinx ISE 14.4 or higher development suit

The detail parameters about *SWORD* box are listed below:

- **Core**: *Xilinx Kintex^{TM-7} Series XC7K160/325*
  - Number: 162,240
  - Slice: 25350
  - Internal storage: 11.7MB
- **Storage architecture**: *Supporting 32-bit storage architecture*
  - 6MB SRAM (Static Random Access Memory): Supporting 32-bit Data, 16-bit TAG
  - 512MB BDDR3 dynamic storage: Supporting 32-bit Data
  - 32MB NOR Flash Storage: Supporting 32-bit Data
- **Basic interface**: *Supporting basic application of microcomputer principle, SOC or microprocessor*
  - 4×5+1 button matric
  - 16-bit slide switch
  - 16-bit LED
  - 8-bit seven-segment LED
- **Standard interface**: *Supporting the realization of basic computer systems*
  - 12-bit VGA interface (RGB656)
  - USB-HID (Keyboard)
- **Communication interface**: *Supporting external storage, multimedia and customized device*
  - MicroSD (TF)
  - PMOD
  - HDMI
  - Arduino



4. Experiment Procedure

Task1: Exp14-HCT138

Create a new project

- `Name`: Exp05-138Decoder
- `Top-level source type`: Schematic or HDL
- `Family`: Kintex7
- `Device`: XC7K150T
- `Package`: FFG676
- `Speed`: -2L

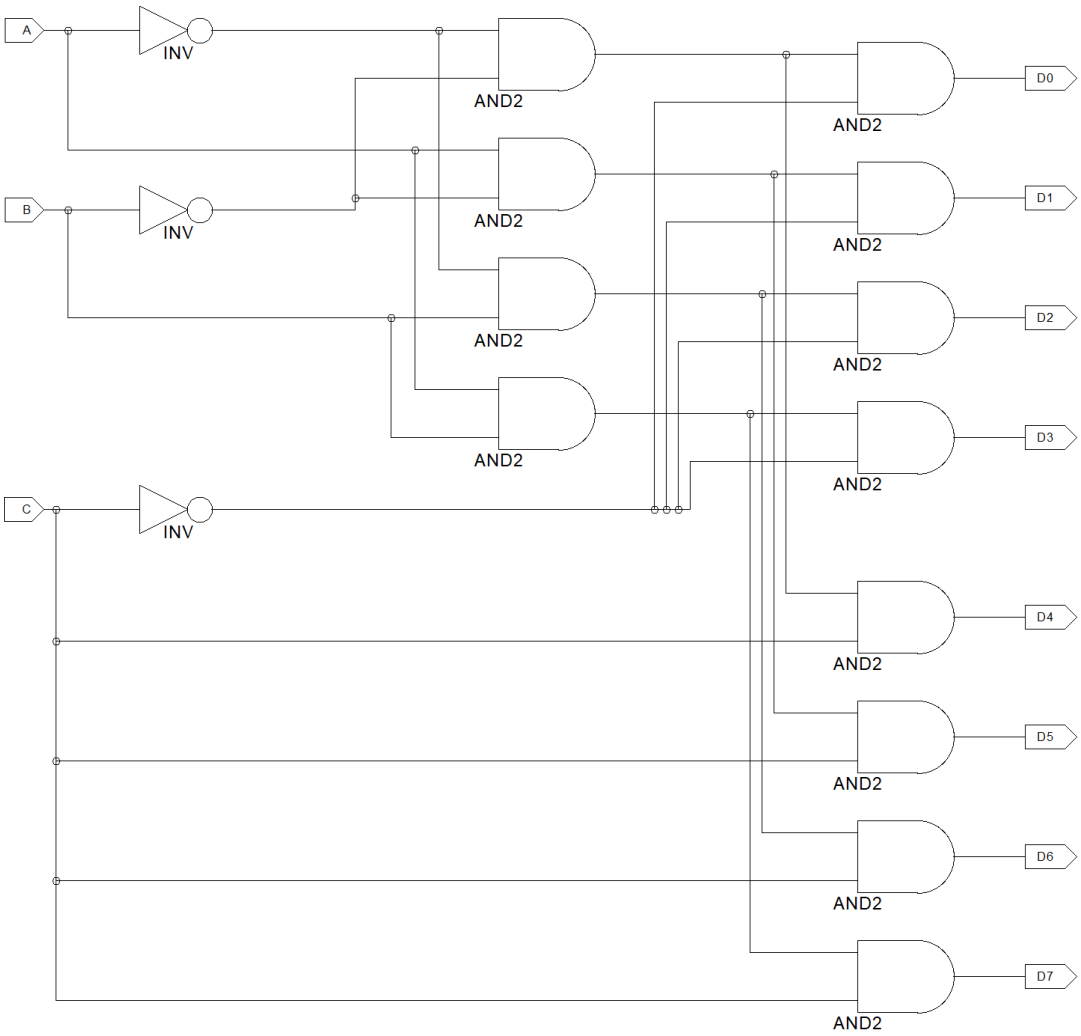
Create Schematic

Enter `File name`: Decoder_38_sch.

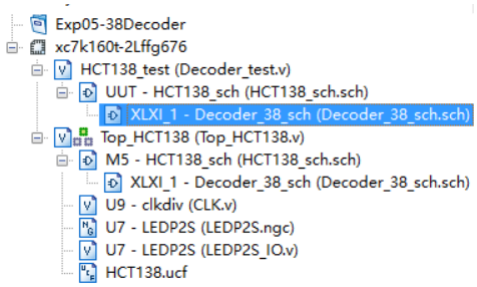
Some basic skills while drawing the schematic:

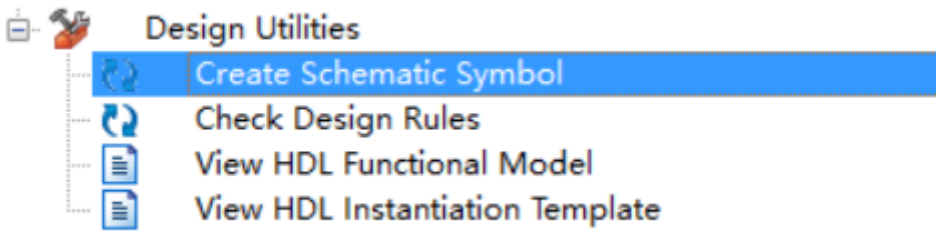
- When an input / output is needed, use this tool  to **add IO marker**.
- Use `Tools --> Check Schematic` to check the connection.
- *ATT: `Check Schematic` will not check the logic functionality*

Finish the schematic like the following:



Create 3-8 Decoder Symbol

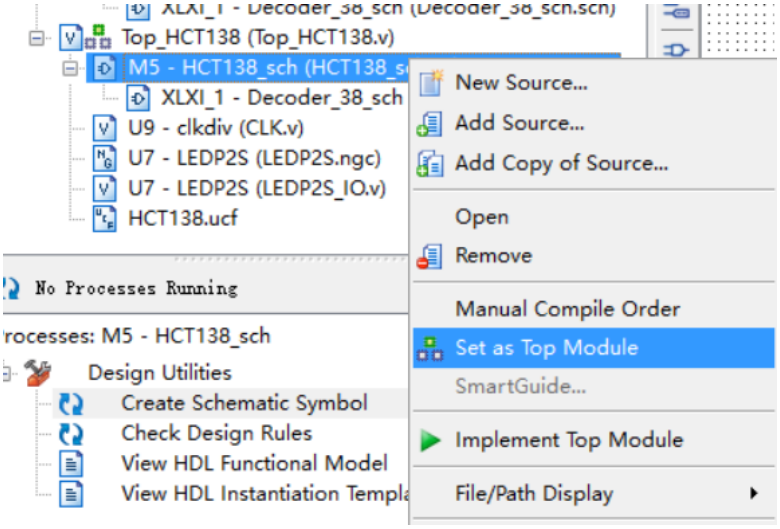
- Select **Decoder_38_sch** file in `Design` panel
  - 

- Double click `Create Schematic Symbol` in `Process` panel.
  - 

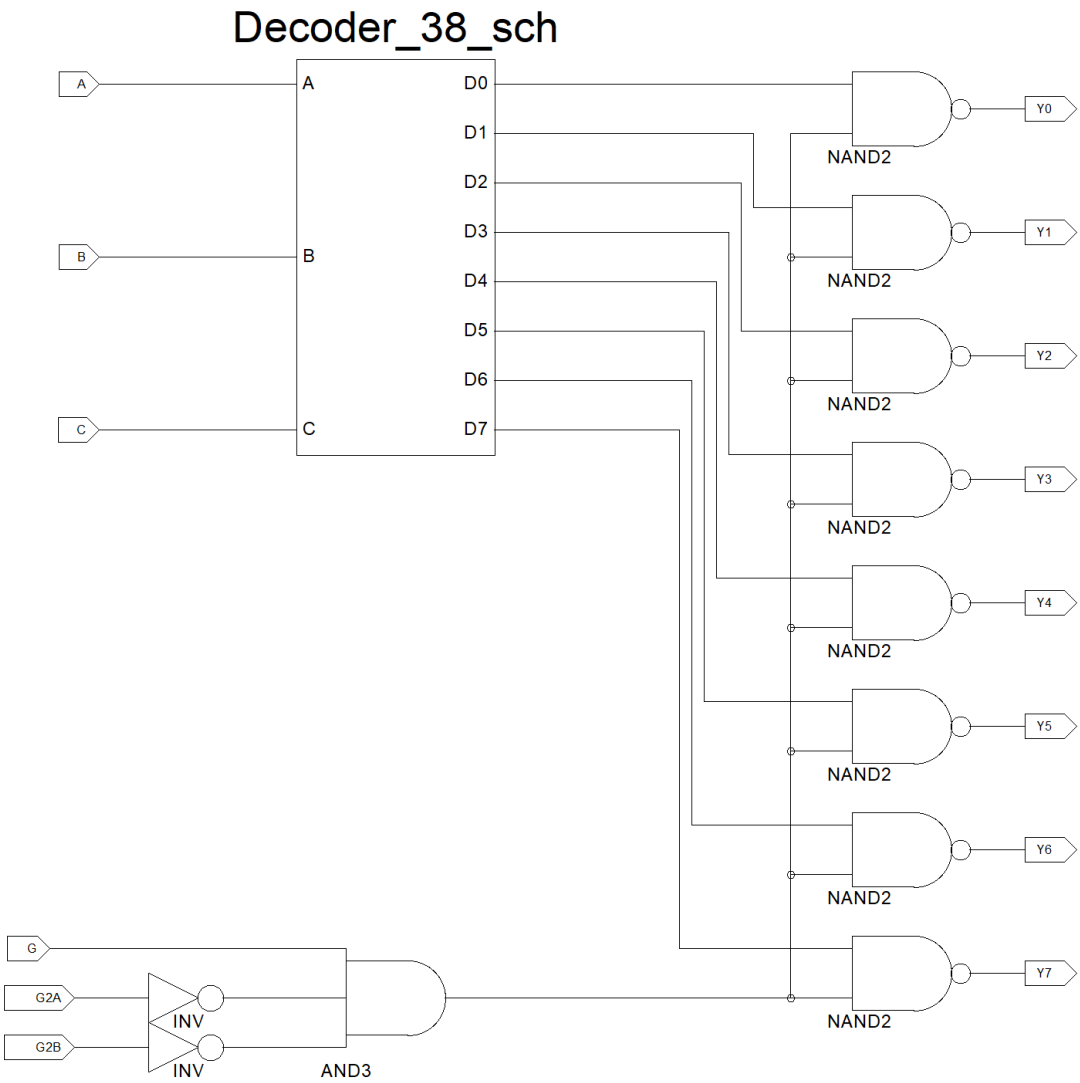
- A file with expansion name `.sym` will be created in the project directory.

Design 74LS138

Create a new schematic file named **HCT138_sch**. Right click it in `work` panel, select `Set as Top Module`.



Draw the schematic like the following:



Again, create a corresponding symbol.

Synthesize

Synthesize is much like what we call **complie** in C language. Select the top module (HCT138_sch.sch), then double click `Synthesize-XST` in `Process` panel.

Simulation

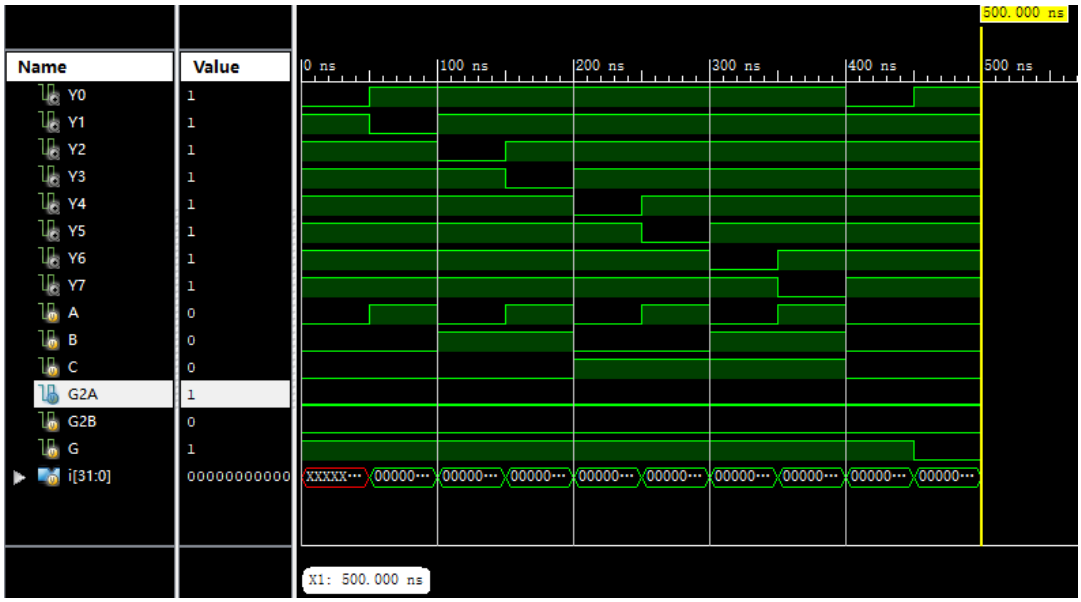
- Right click `Design` panel, select `New Source`.
- Select **Verilog Test Fixture** as source type.
- Enter `File name`: **Decoder_test**.
- Click next or finish till the code page appears.

Enter the code below:

```
`timescale 1ns / 1ps
module HCT138_test();

// Inputs
reg A;
reg B;
reg C;
reg G2A;
reg G2B;
reg G;

// Output
wire Y0;
wire Y1;
wire Y2;
wire Y3;
wire Y4;
wire Y5;
wire Y6;
wire Y7;

// Bidirs

// Instantiate the UUT
HCT138_sch UUT (
        .A(A),
        .B(B),
        .C(C),
        .Y0(Y0),
        .Y1(Y1),
        .Y2(Y2),
        .Y3(Y3),
        .Y4(Y4),
        .Y5(Y5),
        .Y6(Y6),
        .Y7(Y7),
        .G2A(G2A),
        .G2B(G2B),
        .G(G)
);

// Initialize Inputs

    integer i;
    initial begin
    // Add stimulus here
```

```
47      A=0;
48      B=0;
49      C=0;
50
51      G = 1;
52      G2A = 0;
53      G2B = 0;
54
55      #50;
56      for (i=0; i<=7;i=i+1) begin
57      {C,B,A}={C,B,A}+1;
58      #50;
59      end
60      assign G = 0;
61      assign G2A = 0;
62      assign G2B = 0;
63      #50;
64      assign G = 1;
65      assign G2A = 1;
66      assign G2B = 0;
67      #50;
68      assign G = 1;
69      assign G2A = 0;
70      assign G2B = 1;
71      #50;
72      end
73
74  endmodule
```

Then click `Simulate Behavioral Model` in the `Processes` panel. The result should be like this:



Add Top Module

The final test need a top module to realize the design.

Create a new `verilog module` file and then enter the source code below:

```
1  `timescale 1ns / 1ps
```

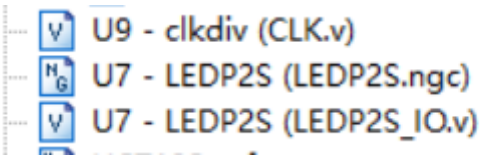
```
module Top_HCT138(input clk_100mhz,

                  input A,B,C,G,G2A,G2B,
                  output wire ledclk,
                  output wire ledsout,
                  output wire ledclrn,
                  output wire LEDEN,
                  output [7:0] Y,
                  );

    wire [31:0] Div,PD;
    wire [7:0] Y;
    wire ny0,ny1,ny2,ny3,ny4,ny5,ny6,ny7;

    assign clk=clk_100mhz;

    HCT138_sch M5(A,B,C,G,G2A,G2B,ny0,ny1,ny2,ny3,ny4,ny5,ny6,ny7);
    assign Y=~{ny7,ny6,ny5,ny4,ny3,ny2,ny1,ny0};

    clkdiv U9(.clk(clk), .clkdiv(Div));
    LEDP2S #(.DATA_BITS(16),.DATA_COUNT_BITS(4),.DIR(0))
        U7(.clk(clk),
            .rst(rst),
            .Start(Div[20]),
            .PData({8'hFF,Y}),
            .sclk(ledclk),
            .sclrn(ledclrn),
            .sout(ledsout),
            .EN(LEDEN)
            );

endmodule
```

Add Auxiliary Module

Create a new `verilog` source named **CLK**, the source code is listed below:

```
`timescale 1ns / 1ps
module clkdiv(input clk,
              output reg [31:0] clkdiv
         );
    always @ (posedge clk) begin
        clkdiv <= clkdiv + 1'b1;
    end

endmodule
```

Then right click the `Design` panel, select `Add Copy of Source`, load file **LEDP2S.ngc** and **LEDP2S_IO.v** from the experiment material.



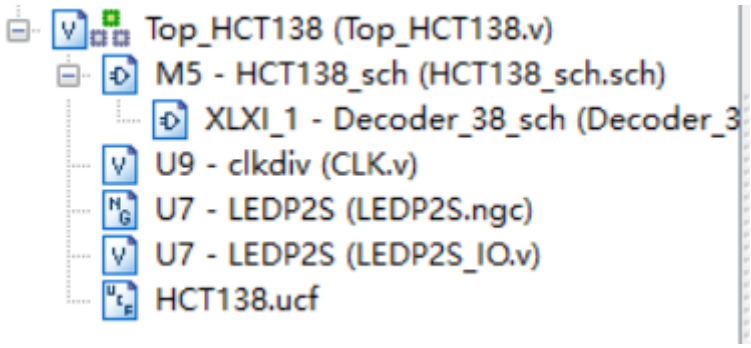
Add constraints File (For Physical Test)

Add `.ucf` file to assign the pinout.

- Right click `Design` panel, select `New Source`
- Select `Implementation Constraints File`.
- Enter name **HCT138**, then step to the source code page
- Enter the source code below:

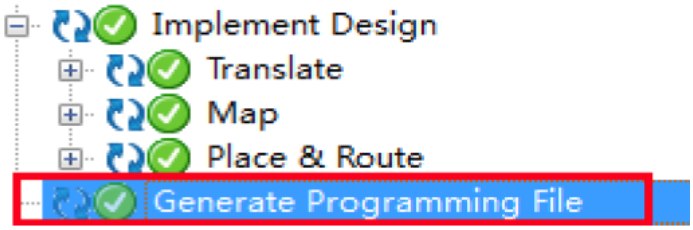
```
#System Clock
NET "clk_100mhz"        LOC = AC18      | IOSTANDARD = LVCMOS18 ;
NET "clk_100mhz"        TNM_NET = TM_CLK ;
TIMESPEC TS_CLK_100M = PERIOD "TM_CLK"  10 ns HIGH 50%;
#74HC138
#SWord
NET "ledclk"            LOC = N26   | IOSTANDARD = LVCMOS33 ;
NET "ledclrn"           LOC = N24   | IOSTANDARD = 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 "G2A"               LOC = AD11  | IOSTANDARD = LVCMOS15 ;#SW[6]
NET "G2B"               LOC = AD10  | IOSTANDARD = LVCMOS15 ;#SW[7]
#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 ;
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]
```

The final hierarchy structure should be like this:

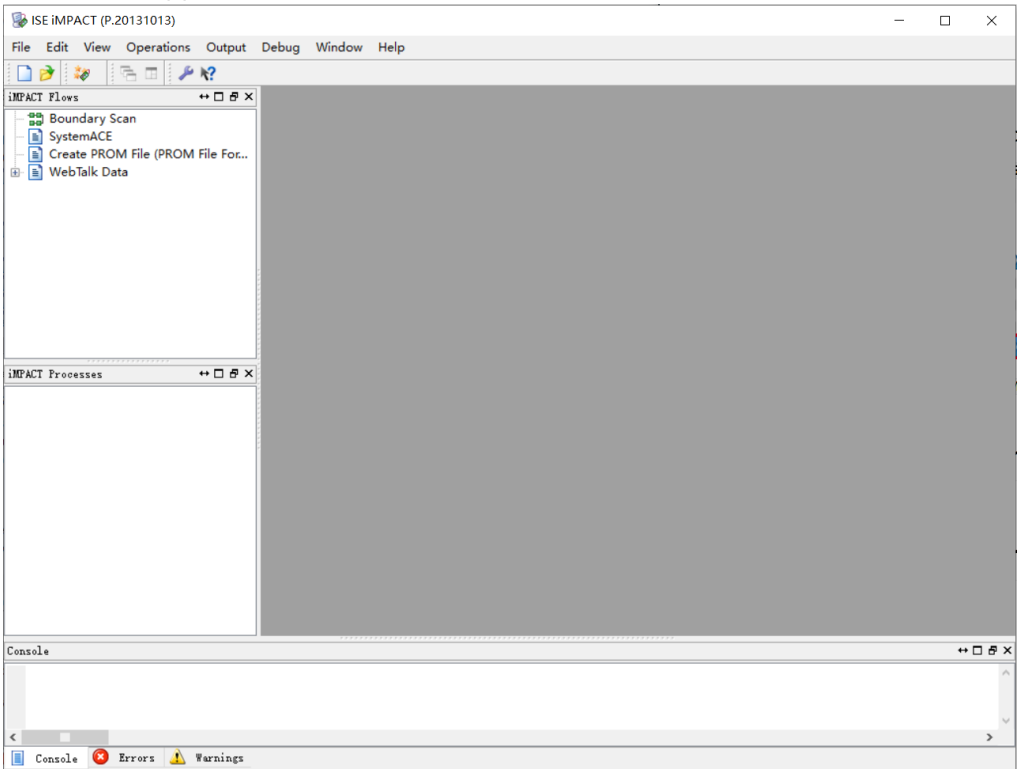


Physical Test

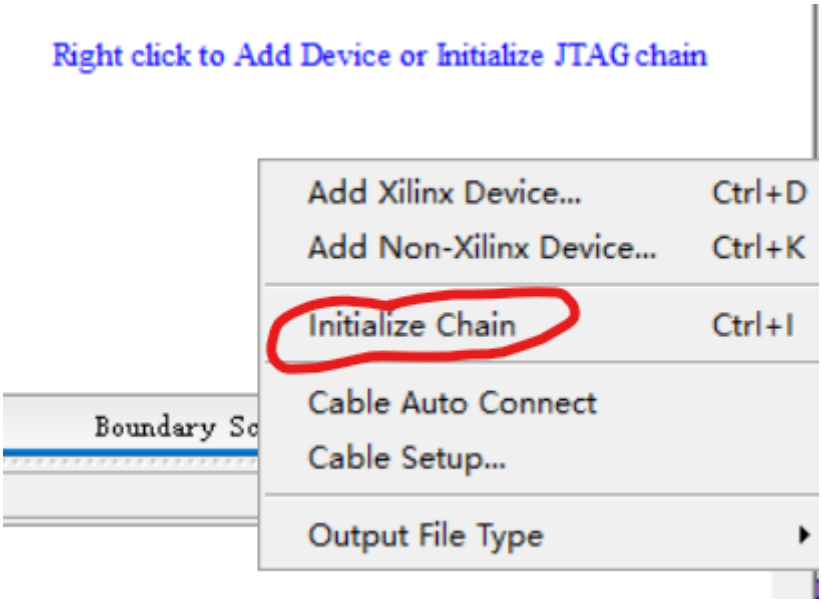
- Select **Top_HCT138** module in `Design` panel.
- Double click `Generate Programming File` in `Processes` Panel, a `.bit` file will be generated after all the steps are completed.



- Double click `Manage Configuration Project (iMPACT)` option, an *ISE iMPACT* window will appear.



- Double click `Boundary Scan` to be ready for searching for available experiment box.
- Right click the white board with some tips: *Right click to Add Device or Initialize JTAG chain*. Then click `Initialize Chain`.



- Right click the experiment device, select `Assign New Configuration File` and select the `.bit` file generated before to download onto the experiment suit.
- Right click the experiment device, select `Program`.
- Cancel all other options emerged.

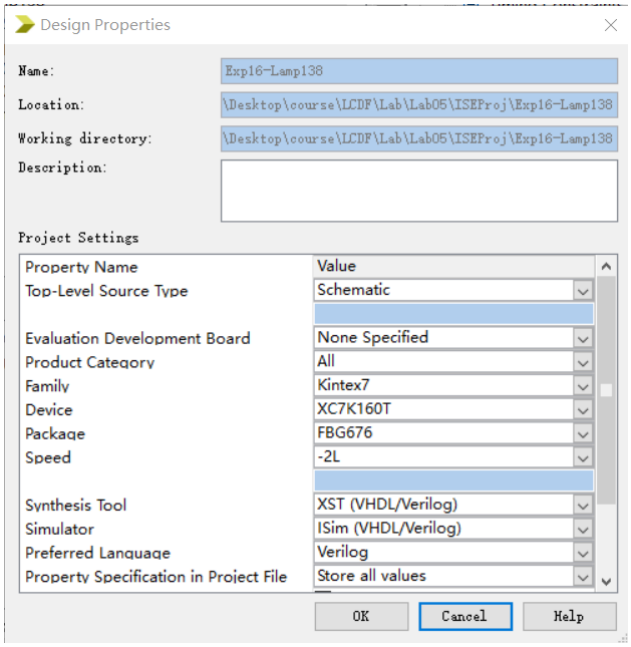
When all are done, check with the design on the experiment device.

Task2: Exp16-Lamp138

In this task, we are going to re-realize the lamp control design in experiment 1 with *74LS138* decoder.

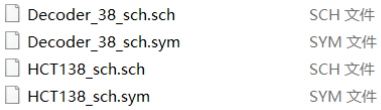
Prerequisite

- Create the project with the likewise procedure before, name the project **Exp16-Lamp138**, select **schematic** as the `Top-Level Source Type`.



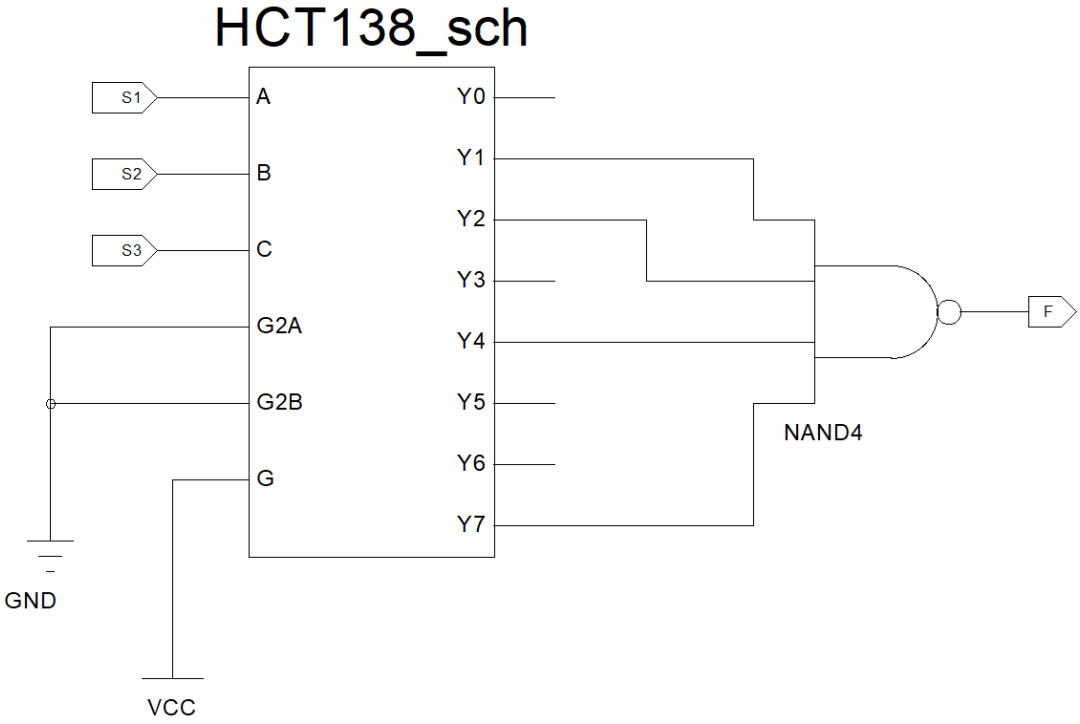
- copy four files from the task1 directory to the directory of this project: **Decoder_38_sch.sch**, **Decoder_38_sch.sym**, **HCT138_sch.sch** and

HCT138_sch.sym. `.sym` files are the symbol items, with which have each `.sch` or `.v` source file corresponds.



Draw the Core Schematic

Draw the schematic as the following image.



Write Pinout Constraints File

Create a `.ucf` file, enter the source code below:

```
NET "F"     LOC = U21   | IOSTANDARD = LVCMOS33;
NET "S1"    LOC = AA10  | IOSTANDARD = LVCMOS15;
NET "S2"    LOC = AB10  | IOSTANDARD = LVCMOS15; #SW[1]
NET "S3"    LOC = AA13  | IOSTANDARD = LVCMOS15; #SW[2]
```

Physical Test

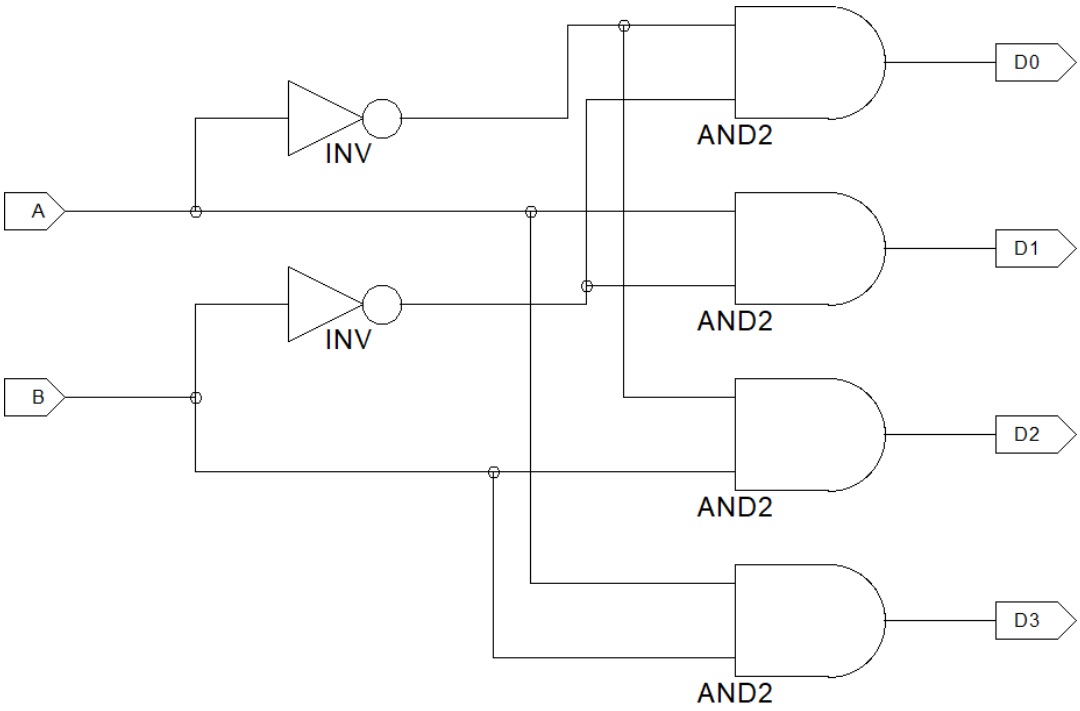
Same with the procedure before, after having downloaded onto the experiment box, check with the logic functionality.

Task3: Design 32×32 bit RAM

ATT: *This is not a complete experiment that can be verified on the circuit board. We shall observe the design procedure and think about what next should be done.*

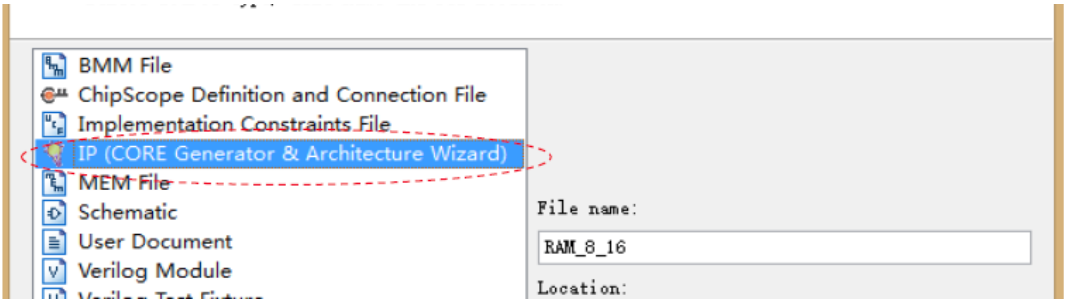
Prerequisite

Create a new project named **Exp-bonus-RAM**. Select **Schematic** as the `Top-level source type`. Then create a **schematic** file to design the **2-4 Decoder** module.

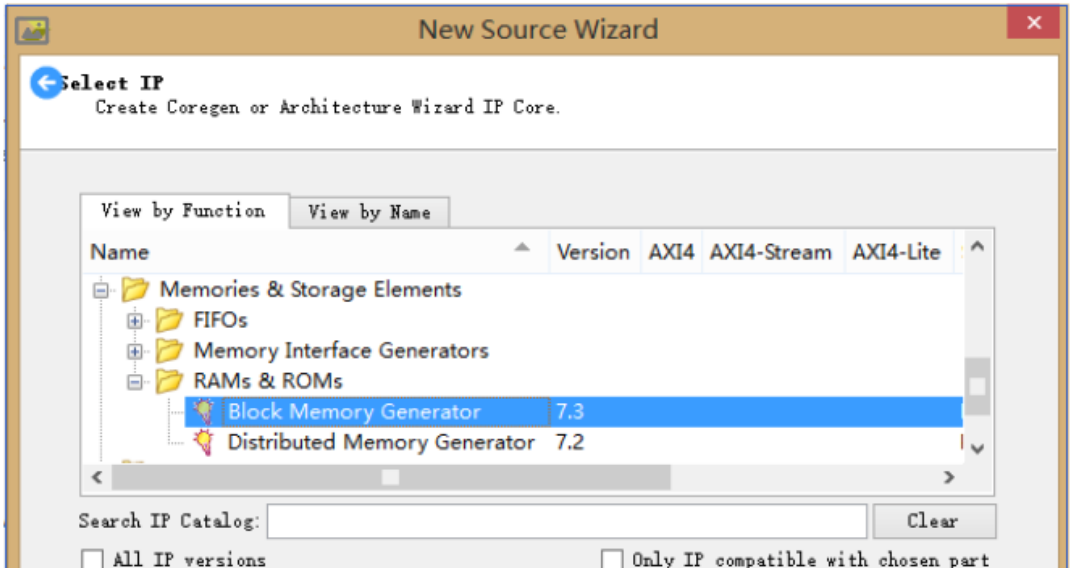


Generate 8×16 bit RAM Core

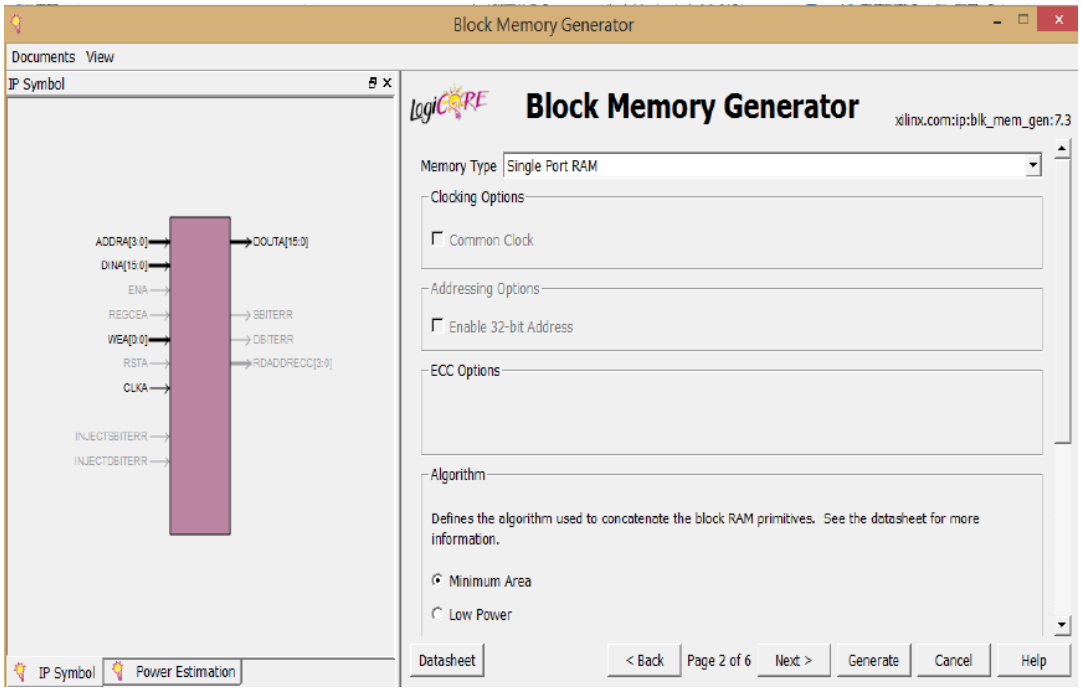
- Right click the `Design` panel, select `New Source`.
- Select `IP (CORE Generator & Architecture Wizard)` as the source type.
- Enter file name **RAM_8_16**.



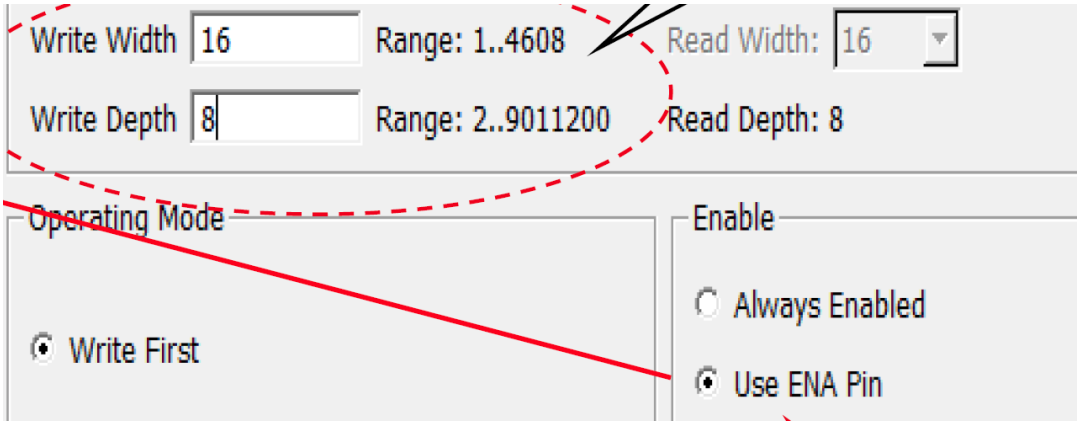
- Click `Next`, then select `Memories & Storage Elements` -> `RAMs & ROMs` -> `Block Memory Generator 7.3`, click `Next`.\



- Configure the core parameter: **Single Port RAM**

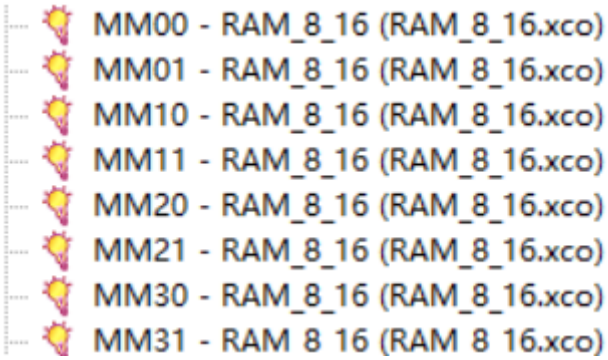


- Enter `Write Width`: **16**, `Write Depth`: **8**. Select `Use ENA Pin`.



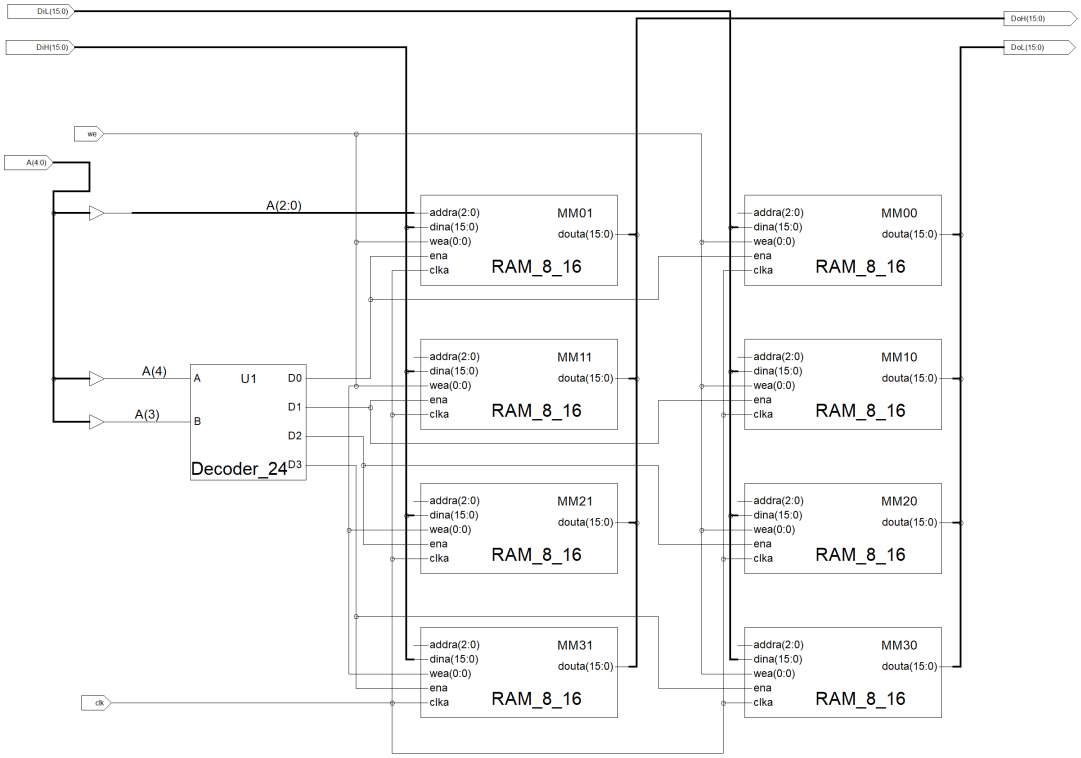
- Click `Next`, then click `Generate` to finish.

The IP core array should be like this:



Draw the 32×32 bit RAM Bank

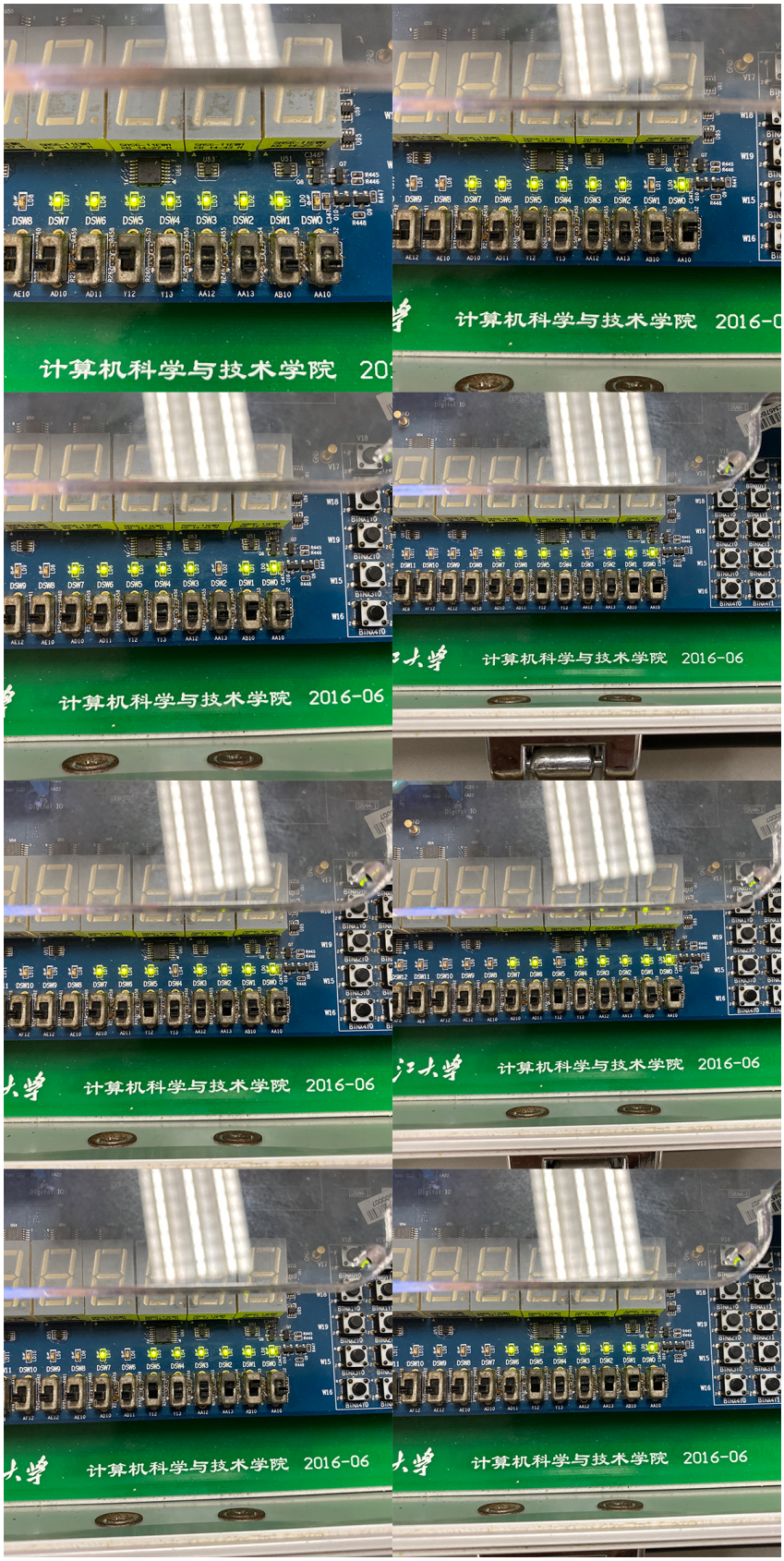
Create a schematic named **MEM_BANK**, and then draw the schematic like this:



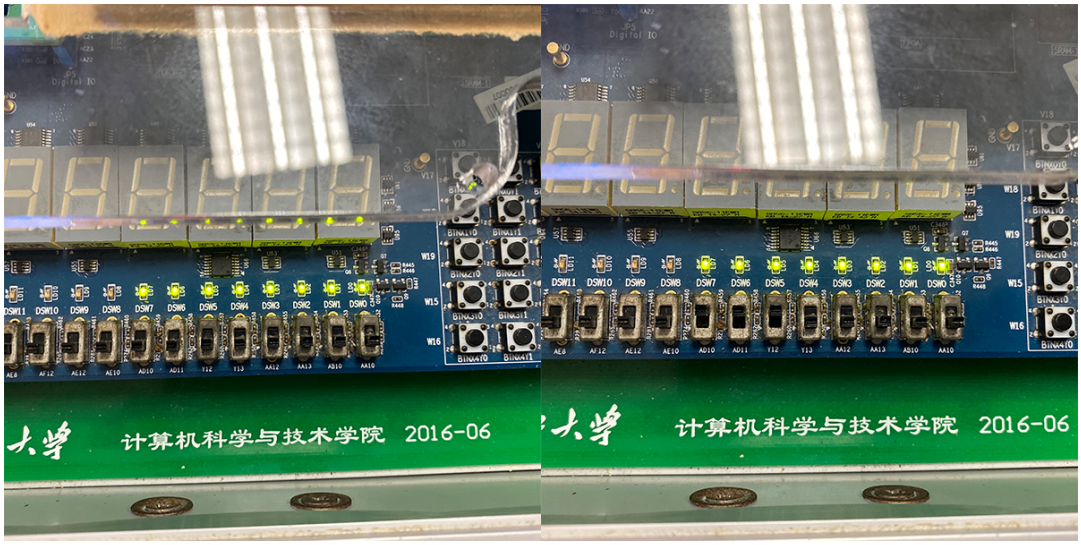
5. Experiment Results and Analyses

Task1: Exp14-HCT138

First set SW[6] at "1" while both SW[7] and SW[8] at "0". With different combination of the states of 3 switches (*SW[1], SW[2], SW[3]*), the LED light numbered with the corresponding decoded result will be turned off. Result will be like the following.



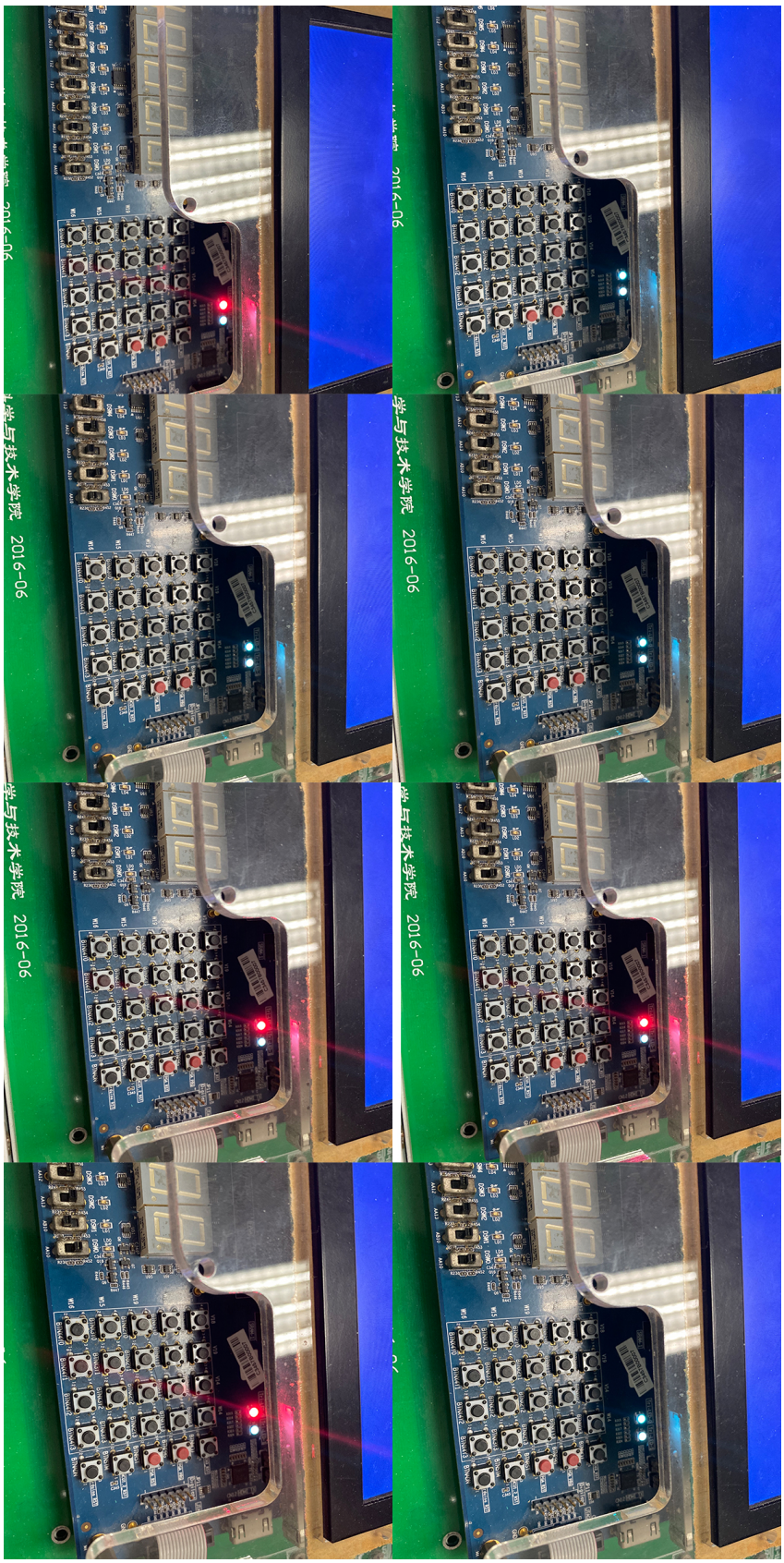
Then if you turn **SW[6] at "0"** or **both SW[7] and SW[8] at "1"**, all eight lights will be turned on.



Thus, we have finished the design of *74LS138* decoder and fully verified the whole logic functionality.

Task2: Exp16-Lamp138

When all three switches SW[1], SW[2] and SW[3] is on, the light bulb looks blue. When exactly one switch is turned off, the light bulb will look red. And still blue when exactly one switch is turned on. Finally, if you shut down all the switches, the light bulb turns red.



6. Discussion and Conclusion

There is nothing special other than getting us more familiar with using *ISE* to design a whole project. We just finish the project with the instruction. During the experiment, I concluded several useful skills:

- Check all the module and source file, ISE will automatically **calculate the module and instantiation** and establish the hierarchy structure.

> - Each `.sym` file need a source file support. For example, `.sch`, `.v` or `.ngc` file. But `.ngc` is a completely encapsulated file which cannot be modified by *ISE*.
>
> Finally, about the addition task to design a memory bank. It cannot work because 4 buses are connect together, which go against the circuit design standard. **What is really needed is a 4-to-1 MUX, which will soon be seen in the future lab tasks.**