

实验五（变量译码器设计与应用）报告

姓名: 蒋仕彪 学号: 3170102587 专业: 求是科学班（计算机）1701

课程名称: 逻辑与计算机设计基础实验

实验时间: 2018-10-18

一、实验目的

- 掌握变量译码器的逻辑构成和逻辑功能。
- 用变量译码器实现组合函数
- 掌握变量译码器的典型应用（地址译码的具体方法）
- 了解存储器编址的概念
- 采用原理图设计电路模块
- 进一步熟悉ISE平台及下载实验平台物理验证。

二、实验内容和原理

2.1 实验内容:

1. 原理图设计实现 74LS138 译码器模块
2. 用 74LS138 译码器实现楼道灯控制

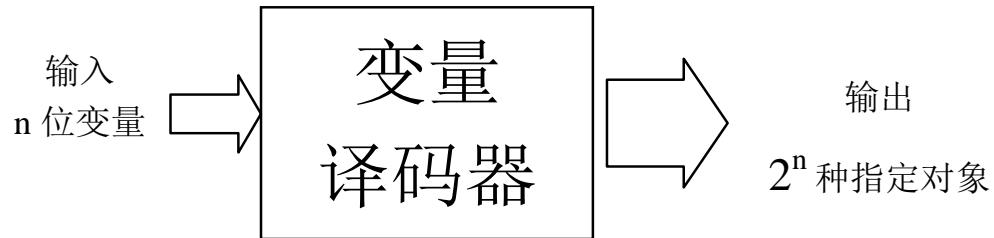
2.2 实验原理:

2.2.1 译码器

- 译码器是将一种输入编码转换成另一种编码的电路，即将给定的代码进行“翻译”并转换成指定的状态或输出信号（脉冲或电平）
- 译码可分为：变量译码、显示译码
 - 变量译码一般是将一种较少位输入变为较多位输出的器件，如 2^n 译码和 8421BCD 码译码。
 - 显示译码主要进行2进制数显示成10进制或16进制数的转换，可分为驱动LED 和LCD两类。

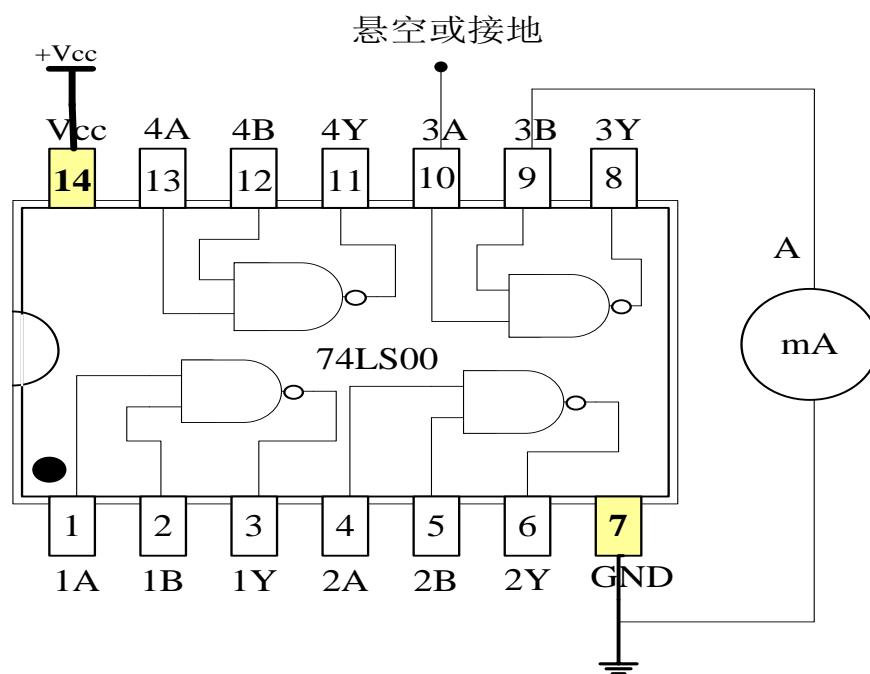
2.2.2 变量译码器

- 变量译码器是一个将 n 个输入变为 2^n 个最小项输出的多输出端的组合逻辑电路。 N 通常在 2^6 ~ 2^{16} 之间。

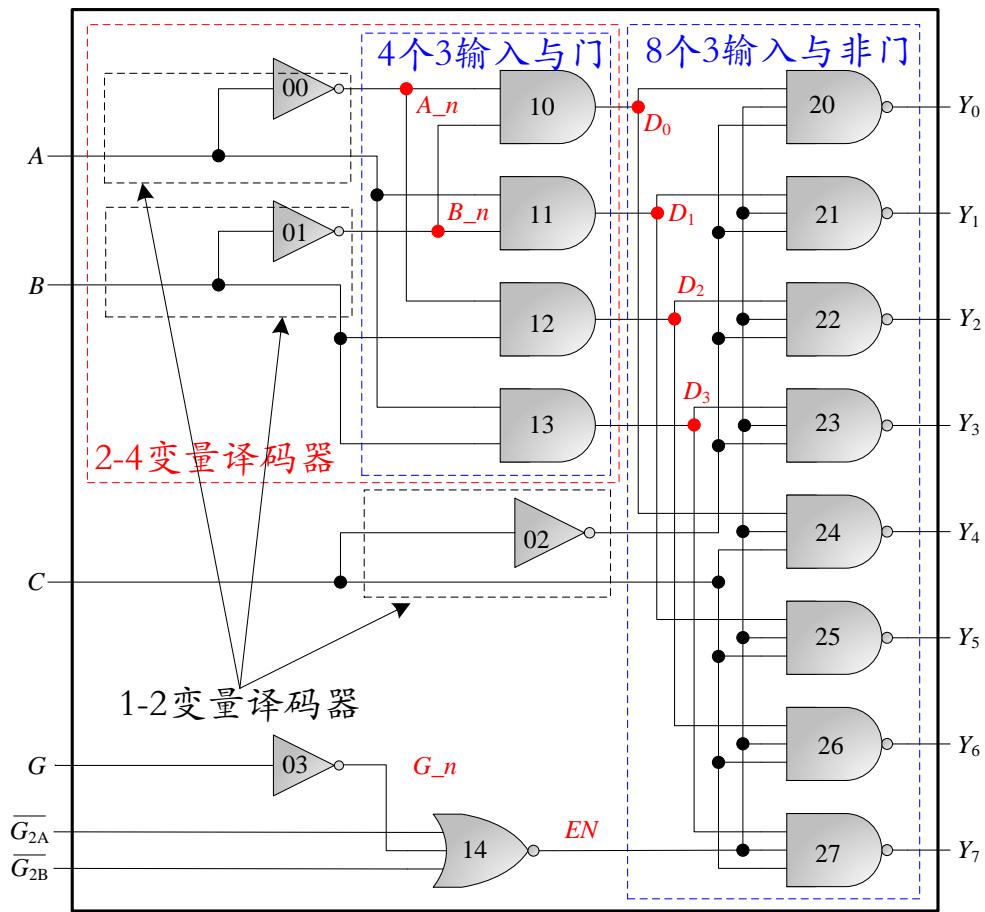


2.2.3 变量译码器74LS138

输入		译码器输出 (低电平有效)								
使能	变 量	CBA	Y ₀	Y ₁	Y ₂	Y ₃	Y ₄	Y ₅	Y ₆	Y ₇
GG _{2A} GG _{2B}			×							
×	X									
×11	X	1	1	1	1	1	1	1	1	1
	X									
0××	X									
	X	1	1	1	1	1	1	1	1	1
	X									
100	000	0	1	1	1	1	1	1	1	1
100	001	1	0	1	1	1	1	1	1	1
100	010	1	1	0	1	1	1	1	1	1
100	011	1	1	1	0	1	1	1	1	1
100	100	1	1	1	1	0	1	1	1	1
100	101	1	1	1	1	1	0	1	1	1
100	110	1	1	1	1	1	1	0	1	1
100	111	1	1	1	1	1	1	1	1	0



□ 带3个使能端的3-8译码器的逻辑结构由三级门电路构成，输出低电平有效。



□ 代码:

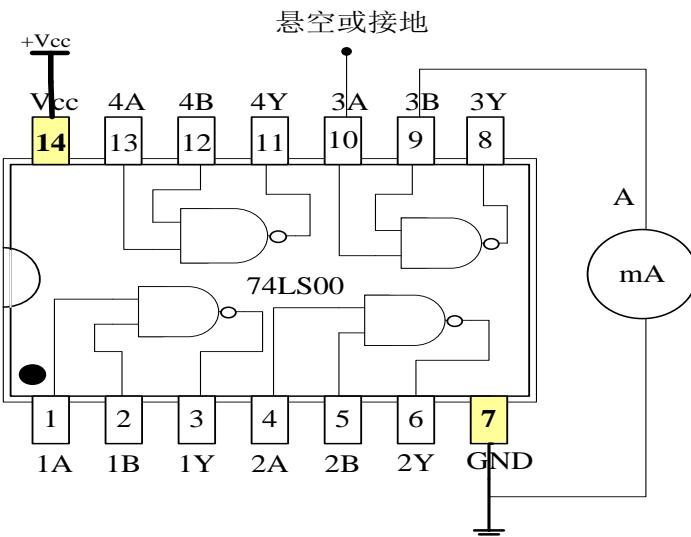
```

module decoder_3_8(C,B,A,G,G2A,G2B,Y
input wire A, B, C, G, G2A, G2B;
output wire [7:0] Y;
not    node_0_0(A_n, A),
       node_0_1(B_n, B),
       node_0_2(C_n, C),
       node_0_3(G_n, G);
and   node_1_0(D0, B_n, A_n),
       node_1_1(D1, B_n, A ),
       node_1_2(D2, B,  A_n),
       node_1_3(D3, B,  A );
nor   node_1_4(EN, G_n, G2A, G2B);
nand  node_2_0(Y[0], EN, D0, C_n),
       node_2_1(Y[1], EN, D1, C_n),
       node_2_2(Y[2], EN, D2, C_n),
       node_2_3(Y[3], EN, D3, C_n),
       node_2_4(Y[4], EN, D0, C ),
       node_2_5(Y[5], EN, D1, C ),
       node_2_6(Y[6], EN, D2, C ),
       node_2_7(Y[7], EN, D3, C );
.endmodule

```

2.2.4 变量译码器74LS139

- 74LS139变量译码器功能表和引脚

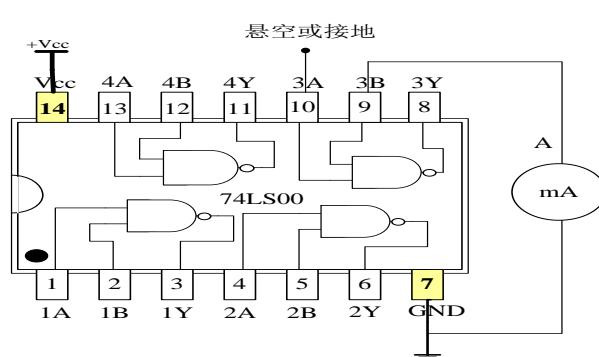


输入		译码器输出 (低电平有效)			
使能	变量	Y_0	Y_1	Y_2	Y_3
1	$\times \times$	1	1	1	1
0	00	0	1	1	1
0	01	1	0	1	1
0	10	1	1	0	1
0	11	1	1	1	0

```
module decoder_2_4(B, A, G, Y);
..... //端口及变量定义
case({B,A})
  2'b00:Y=4'b0001;
  2'b01:Y=4'b0010;
  2'b10:Y=4'b0100;
  2'b11:Y=4'b0001;
endmodule
```

2.2.5 用变量译码器实现组合函数

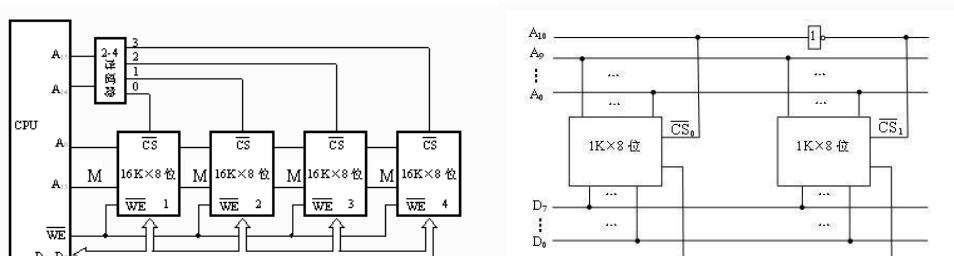
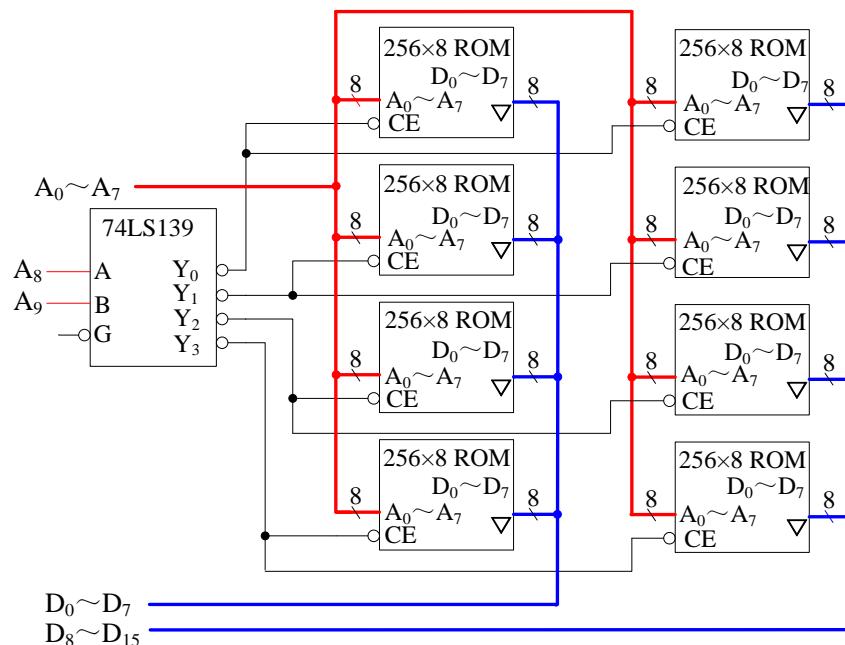
- 变量译码器的输出对应所有输入变量的最小项组合，如果将函数转换成最小项和的形式，则可以用变量译码器实现函数的组合电路：



S_3	S_2	S_1	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

2.2.6 变量译码器实现存储器地址译码

- 存储器电路中地址译码的意义:
 - 在容量扩展时, 将不同的芯片分配到不同地址段, 来达到更大的存储容量(寻址范围)
 - 在容量扩展时, 将不同的芯片分配在同一地址段, 来达到更大的存储单元(存储字)
- 地址译码原理: 将访问存储器的地址线高位作为译码器的输入, 译码器的输出控制各存储器的片选信号。
- 字扩展: 译码器的不同输出连接到不同存储芯片的片选端。
- 位扩展: 译码器的同一输出连接到不同存储芯片的片选端。
- 同时进行字扩展和位扩展。



4片16k*8位芯片组成的64k*8位存储器

图 4.14 由 2 片 1K×8 位的芯片组成的 2K×8 位的存储器

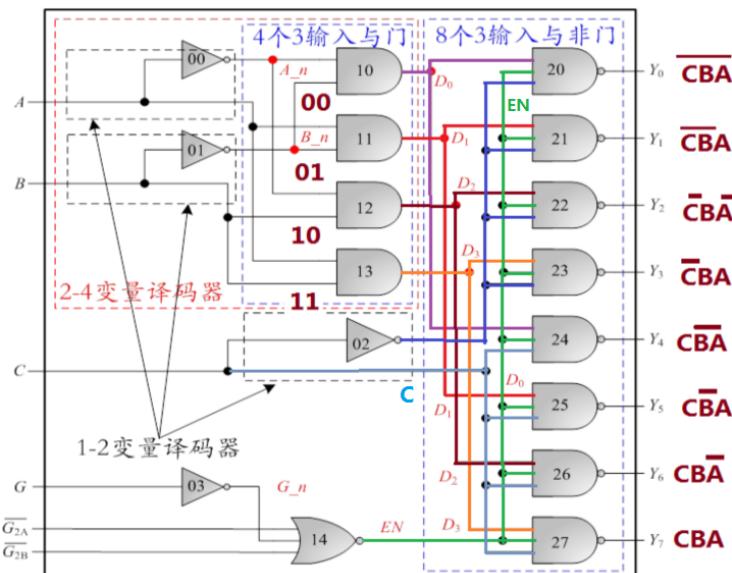
三、主要仪器设备

3. 装有Xilinx ISE 14.7的计算机 1台
4. SWORD开发板 1套

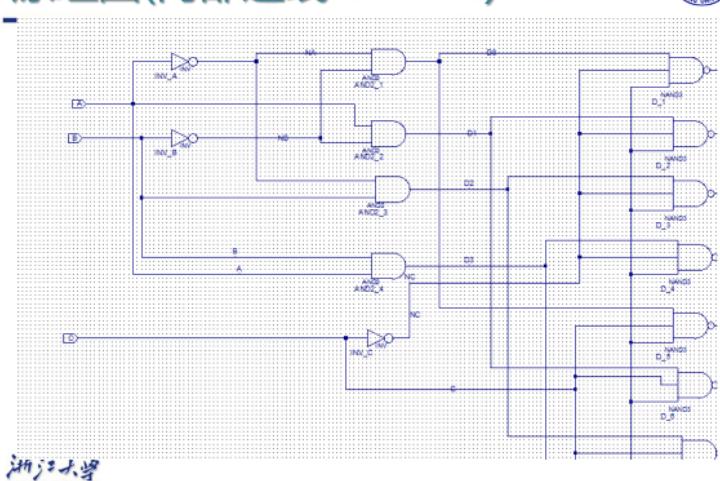
四、操作方法与实验步骤

4.1 设计实现74LS138

- 新建工程，工程名称用D_74LS138_SCH。（第一个工程）
- 新建Schematic源文件，文件名称用D_74LS138。
- 原理图方式进行设计。



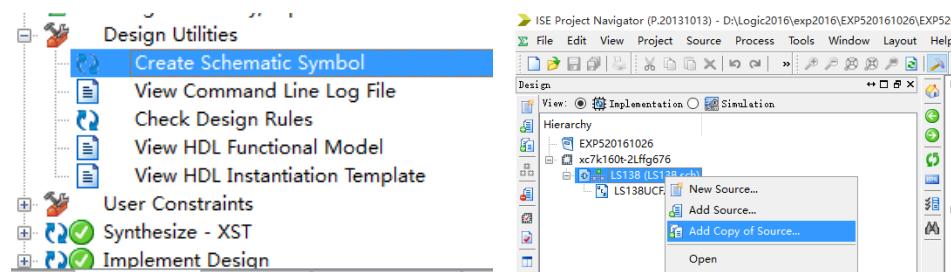
原理图(内部连线rename)



- Check Design Rules, 检查错误
- View HDL Functional Model, 查看并学习 Verilog HDL 代码
- 对 D_74LS138 模块进行仿真。



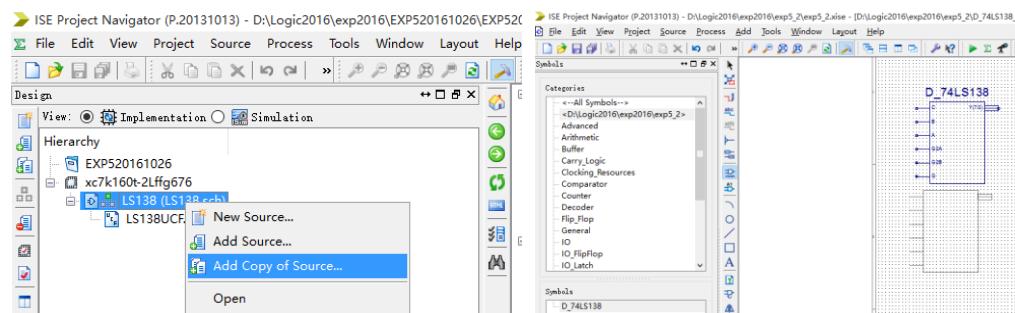
- 生成逻辑符号图和 VF 文件。
- Create Schematic Symbol, 系统生成 D_74LS138 模块的逻辑符号图文件，文件后缀. sym



- 符号图位于工程根目录
 - 自动生成的符号可修改：可以用 Tools 菜单的 Symbol Wizard，也可以打开. sym 文件直接修改。
 - 使用时，把. sym 和. vf {添加到工程中，ADD NEW SCOURSE}（或. v）复制到对应工程目录。

4.2 验证D_74LS138

- 新建工程“D_74LS138_Test”。（第二个工程）
 - 新建 Schematic 文件“D_74LS138_Test”。
 - 复制 D_74LS138. sym 和. vf 到工程目录。
- *. sym, *. vf (或 D_74LS138. sch) {添加到工程中，ADD NEW SCOURSE}。



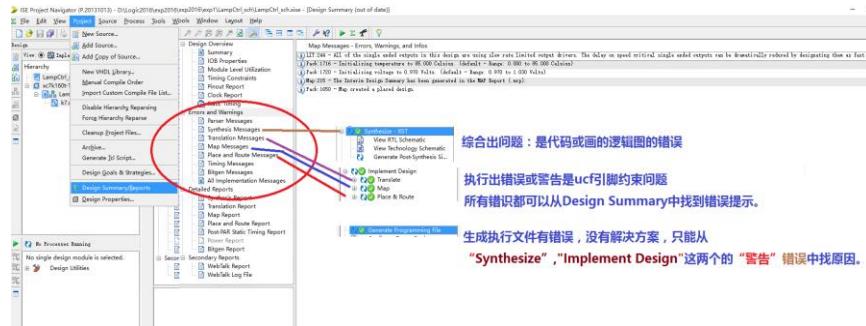
- 下载验证。
 - UCF 引脚定义
 - 输入用 6 个开关

- 3 个译码输入: SW[2:0] AA13, AB10, AA10
 - 3 个使能控制: SW[5:3] Y12, Y13, AA12
 - 输出用 8 个 LED 灯 LED[7:0]

- 输出用8个LED灯：LED[7:0]

□ 设计实现并检查约束结果

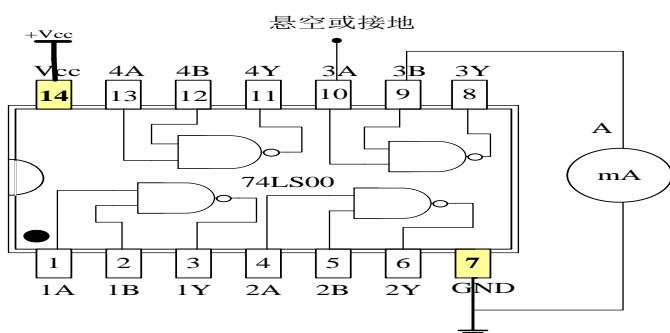
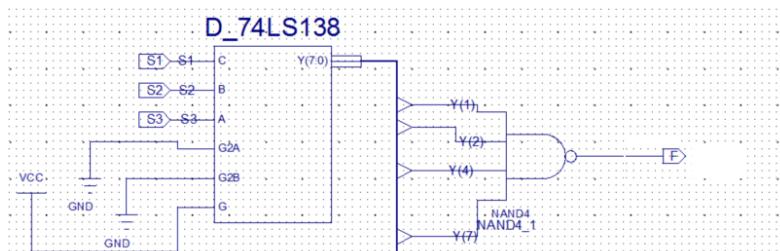
在 Sources 窗口中选择 Synthesis / Implementation，选中 lamp_ctrl；在 Processes 窗口下选择 Implement Design，进行物理转换、平面布图、映射、物理布线等 FPGA 目标格式实现文件生成。最后在设计摘要文档中有如下结果：



- 根据真值表验证。

4.3 实现楼道灯控制

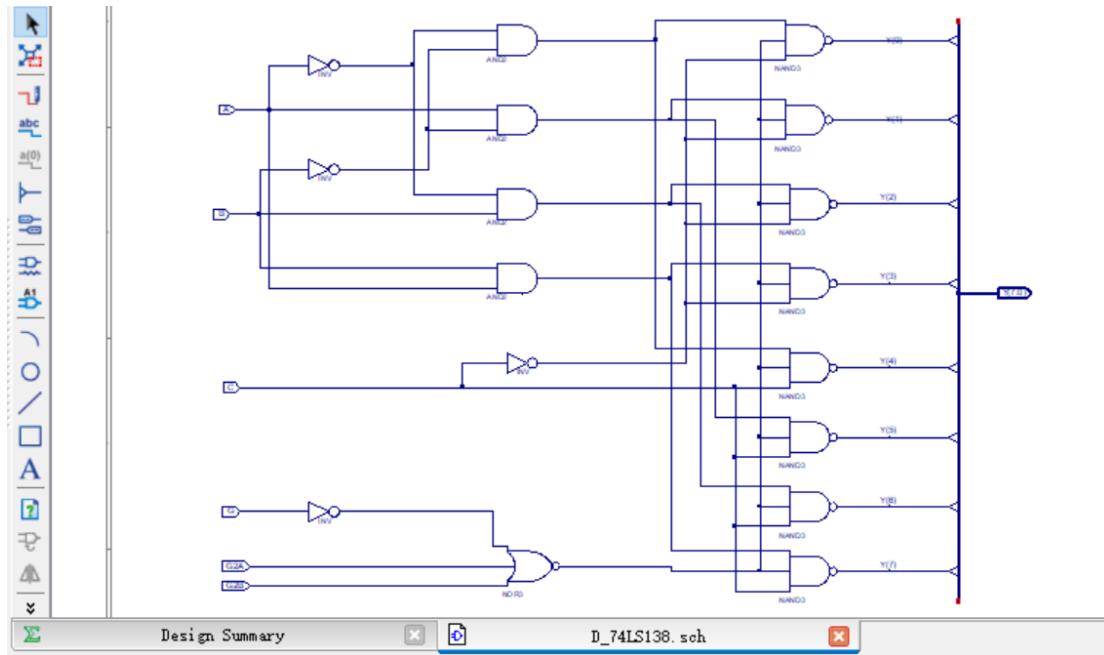
- 新建工程LampCtrl1138。（第三个工程）
 - 复制D_74LS138.sym和.vf文件到工程目录。.vf {添加到工程中，ADD NEW SCOURSE}
 - 在symbols框里的第一个元件，就是D_74LS138。
 - 根据前面原理，用原理图方式输入。
 - 1用VCC，0用GND。



- 仿真并下载。

五、实验结果与分析

5.1 设计实现74LS138

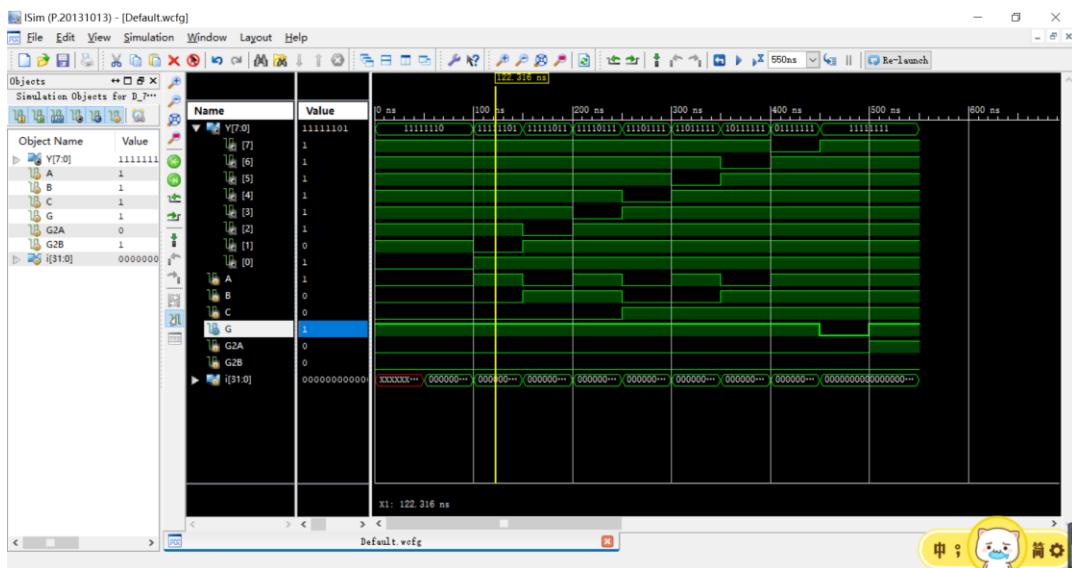


□ 以上为原理图的设计

该截图展示了ISE Project Navigator中的仿真代码。左侧是库浏览器（Libraries），显示了Source Libraries下的work文件夹，其中包含D_74LS138.sch和FangZhen.v。右侧是代码编辑器，显示了VHDL语言编写的测试代码。代码实现了对输入A[2]、A[1]、A[0]和控制信号G2A、G2B、G1的处理，生成输出Y[0]到Y[7]。

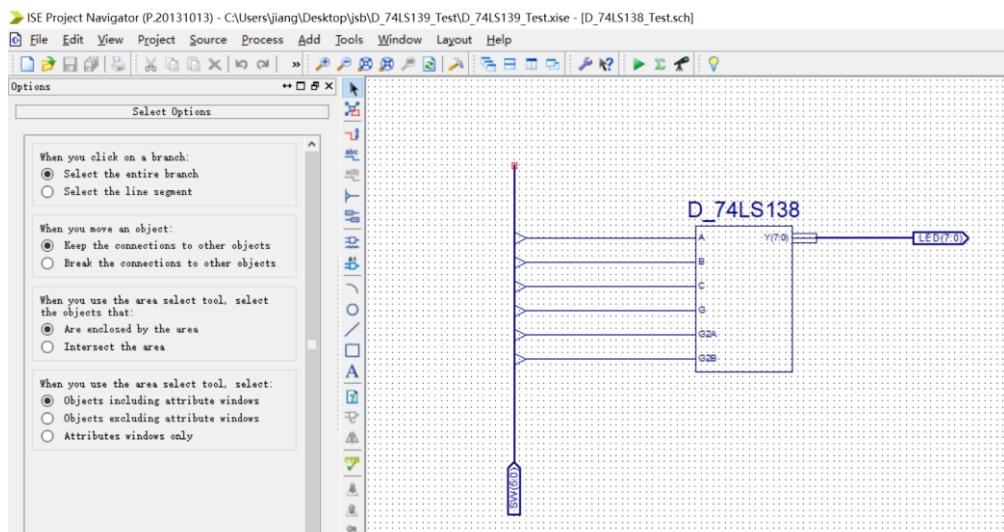
```
26    .G2A(G2A),
27    .G2B(G2B),
28    .Y(Y)
29  );
30  // Initialize Inputs
31  // `ifndef auto_init
32  integer i;
33  initial begin
34    A = 0;
35    B = 0;
36    C = 0;
37    G = 1;
38    G2A = 0;
39    G2B = 0;
40    #50;
41    for (i=0;i<=7;i=i+1) begin
42      {C,B,A}=i;
43      #50;
44    end
45    assign G=0;
46    assign G2A=0;
47    assign G2B=0;
48    #50;
49    assign G=1;
50    assign G2A=1;
51    assign G2B=0;
52    #50;
53    assign G=1;
54    assign G2A=0;
55    assign G2B=1;
56    #50;
57  end
58
```

□ 以上为仿真代码



□ 以上为仿真图像

5.2 验证74LS138



□ 以上为原理图设计

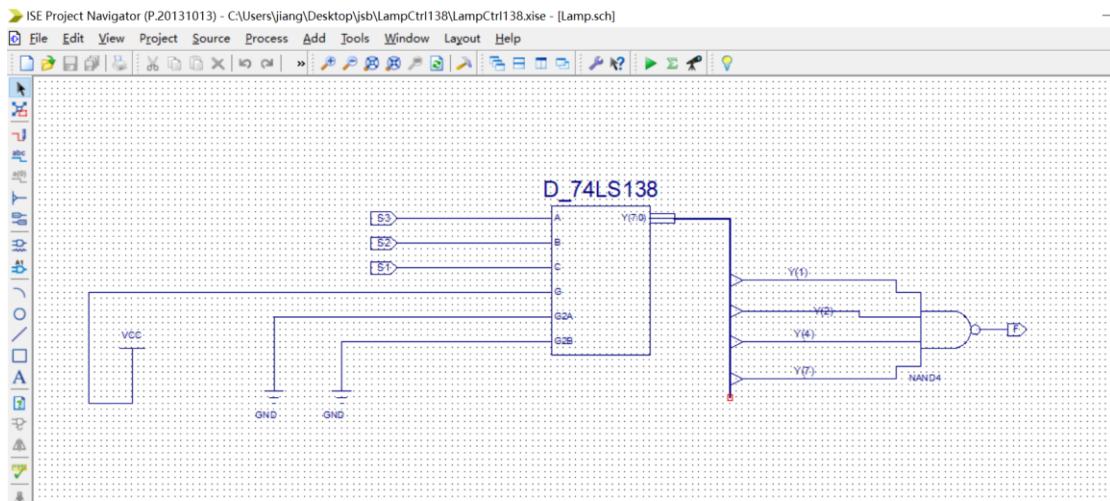
```

1 NET "SW[0]" LOC = AA10 | IOSTANDARD=LVC MOS15;
2 NET "SW[1]" LOC = AB10 | IOSTANDARD=LVC MOS15;
3 NET "SW[2]" LOC = AA13 | IOSTANDARD=LVC MOS15;
4 NET "SW[3]" LOC = AA12 | IOSTANDARD=LVC MOS15;
5 NET "SW[4]" LOC = Y13 | IOSTANDARD=LVC MOS15;
6 NET "SW[5]" LOC = Y12 | IOSTANDARD=LVC MOS15;
7
8 NET "LED[0]" LOC=W23 | IOSTANDARD=LVC MOS33 ;#D1
9 NET "LED[1]" LOC=AB26 | IOSTANDARD=LVC MOS33 ;#D2
10 NET "LED[2]" LOC=Y25 | IOSTANDARD=LVC MOS33 ;#D3
11 NET "LED[3]" LOC=AA23 | IOSTANDARD=LVC MOS33 ;#D4
12 NET "LED[4]" LOC=Y23 | IOSTANDARD=LVC MOS33 ;#D5
13 NET "LED[5]" LOC=Y22 | IOSTANDARD=LVC MOS33 ;#D6
14 NET "LED[6]" LOC=AE21 | IOSTANDARD=LVC MOS33 ;#D7
15 NET "LED[7]" LOC=AF24 | IOSTANDARD=LVC MOS33 ;#D8

```

□ 以上为引脚定义

5.3 实现楼道灯控制



□ 以上为原理图

```
NET "S1" LOC = AA10 | IOSTANDARD=LVCMOS15;
NET "S2" LOC = AB10 | IOSTANDARD=LVCMOS15;
NET "S3" LOC = AA13 | IOSTANDARD=LVCMOS15;
NET "F" LOC=W23 | IOSTANDARD=LVCMOS33 ;#D1
```

□ 以上为引脚定义



□ 以上为实际成果展示

实验六（七段数码管显示译码器设计与应用）

实验报告

姓名: 蒋仕彪 学号: 3170102587 专业: 求是科学班(计算机) 1701

课程名称: 逻辑与计算机设计基础实验

实验时间: 2018-10-25

一、实验目的

- 掌握七段数码管显示原理
- 掌握七段码显示译码设计
- 进一步熟悉 Xilinx ISE 环境及 SWORD 实验平台

二、实验内容和原理

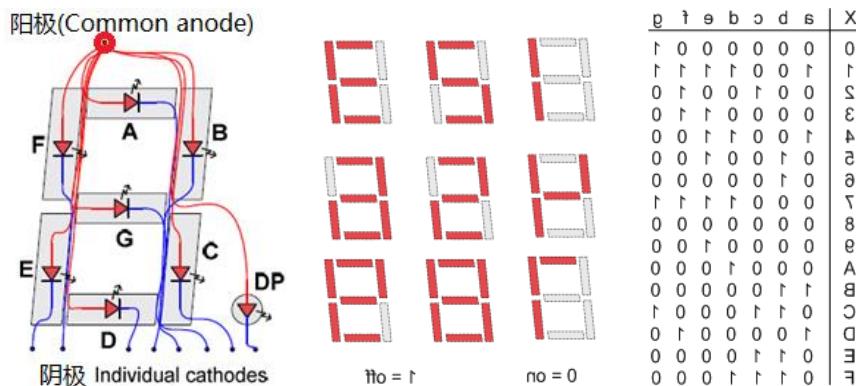
2.1 实验内容

- 任务1: 原理图设计实现显示译码MyMC14495模块
- 任务2: 用MyMC14495模块实现数码管显示

2.2 实验原理

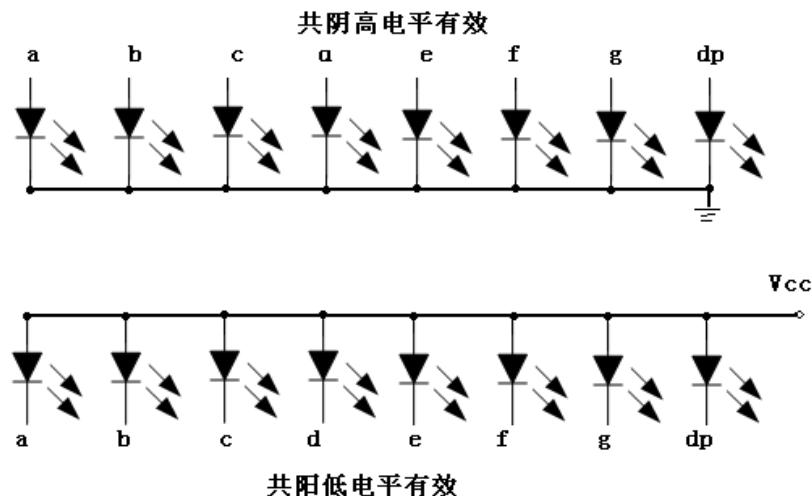
2.2.1 七段数码管

- 由7+1个LED构成的数字显示器件
- 每个LED显示数字的一段，另一个为小数点



2.2.2 共阴（阳）控制

- LED的正极(负极)连在一起，另一端作为点亮的控制
 - 共阳：正极连在一起，负极=0，点亮
 - 共阴：负极连在一起，正极=1，点亮



2.2.3 Hex 7-segment decoder

Hex	D ₃ D ₂ D ₁ D 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	0	1	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

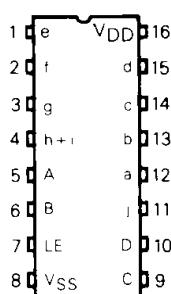
□ 兼容MC14495 略掉：

- Pin11=VCR
- Pin4=h+i

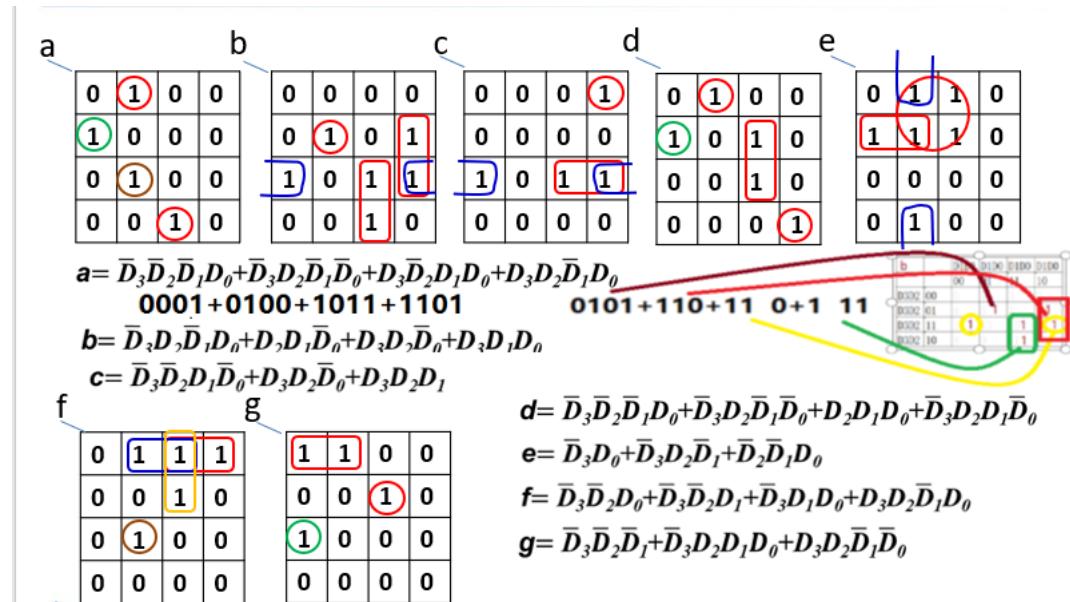
□ 其它

- 共阳：74LS46/47
- 共阴：74LS48/49

CMOS4511

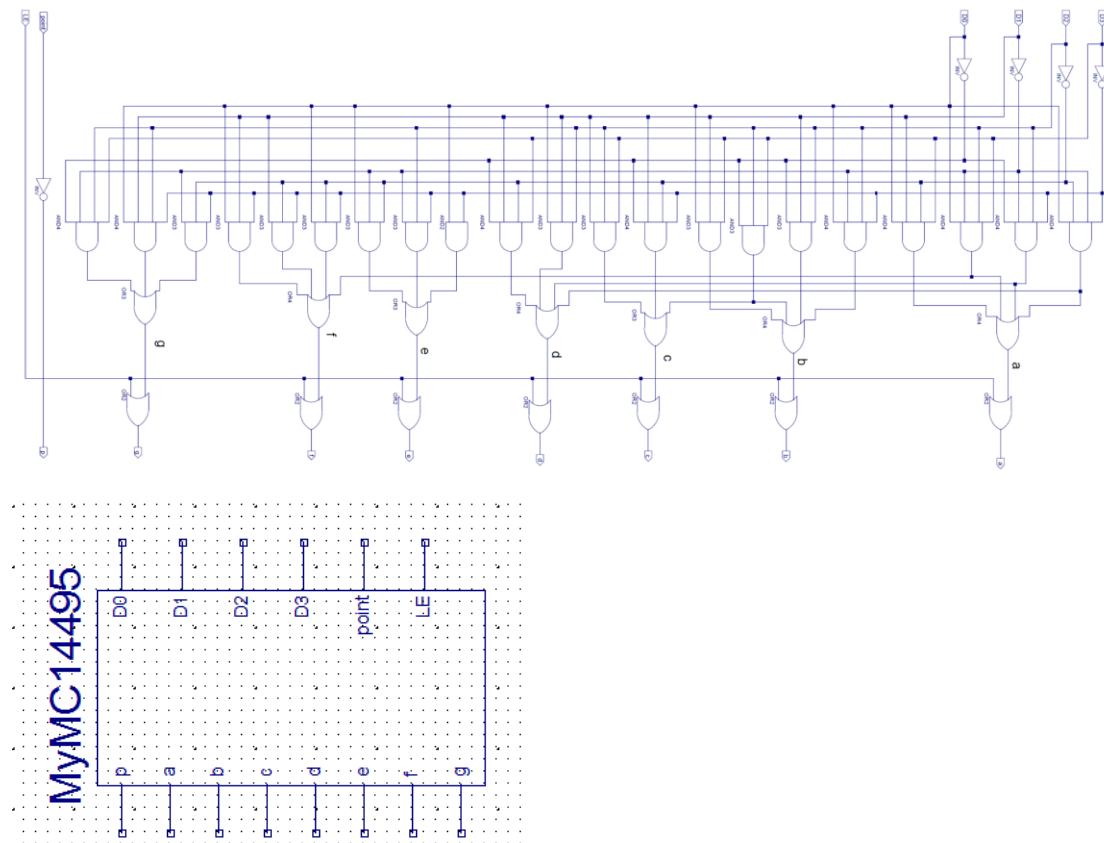


2.2.4 Hex 7-segment decoder



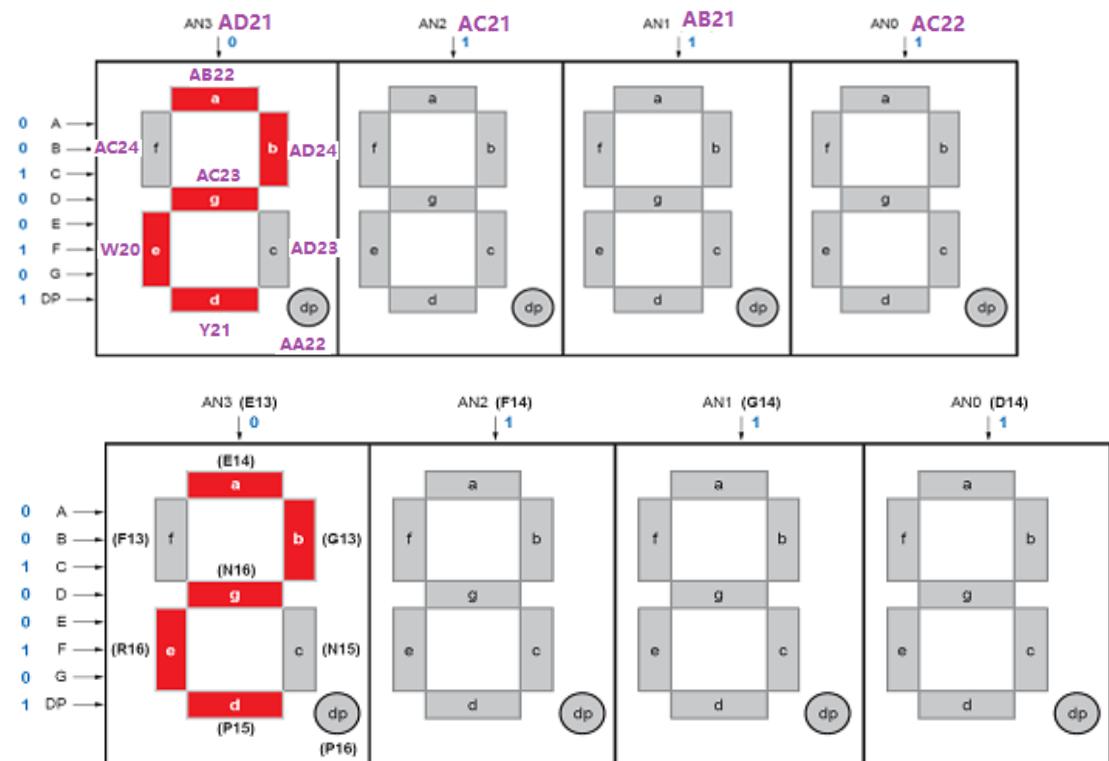
2.2.5 Hex to 7-segment decoder Schematic

- 兼容MC14495
- 省略VCR(Pin11)和h+i(Pin4)功能



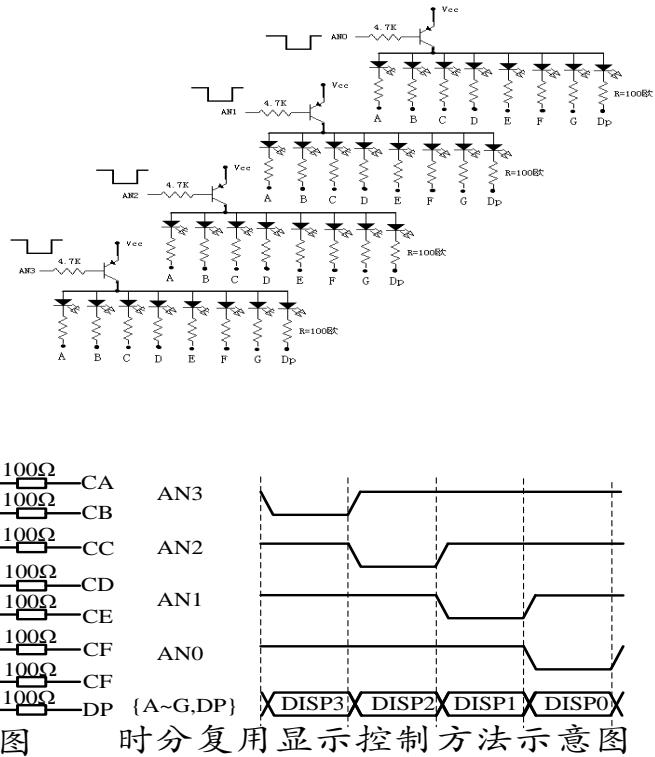
2.2.6 多位七段数码管显示原理

- 静态显示
 - 每个7段码对应一个显示译码电路
- 动态扫描显示：时分复用显示
 - 利用人眼视觉残留
 - 一个7段码译码电路分时为每个7段码提供译码
- 控制时序
 - 用定时计数信号控制公共极，分时输出对应七段码的显示信号： 动态扫描
- 4位七段码结构
 - 正极：公共端
 - 七段信号并联



2.2.7 分时控制示意

- 动态扫描
- 低电平与输入显示对应
- 共阳：低电平控制
- 分时送 $a \sim g, p$
- 可用序列信号控制



三、实验设备与材料

- 装有 Xilinx ISE 14.7 的计算机 1 台
- SWORD 开发板 1 套

四、操作方法与实验步骤

4. 1 设计实现MY_MC14495

- 新建工程，工程名称用MyMC14495。
- 新建源文件，文件名称用MyMC14495。
- 原理图方式进行设计。
- 对MyMC14495模块进行仿真。

4. 2 实现数码管显示

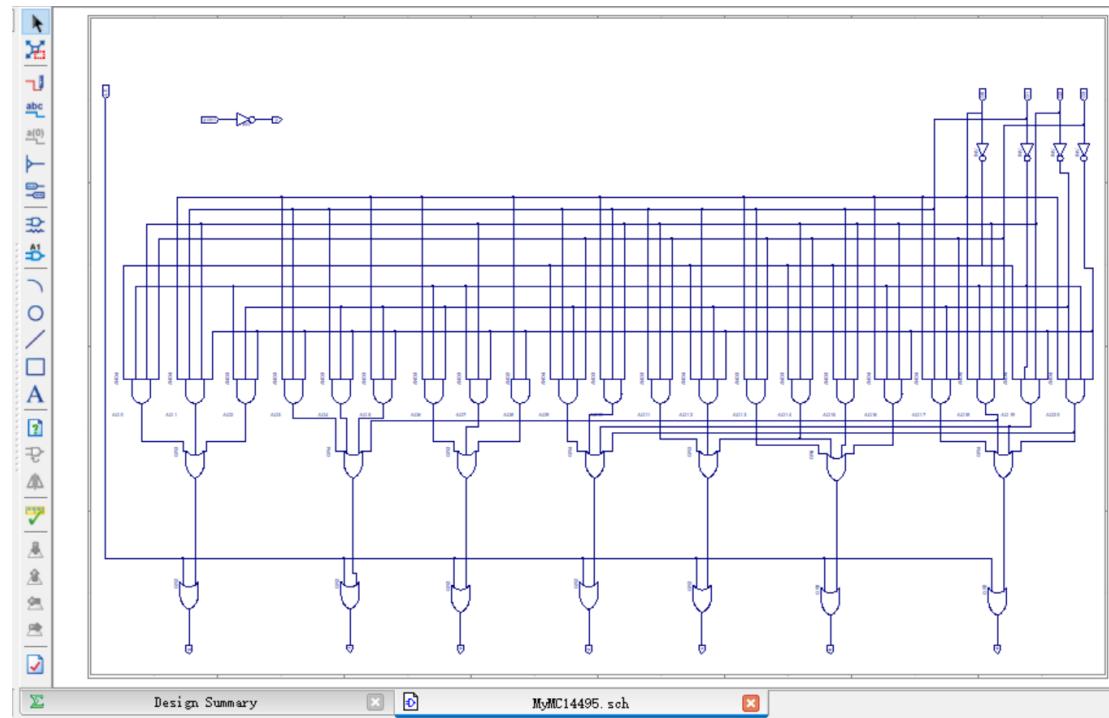
- 新建工程DispNumber_sch
- 新建schematic文件DispNumber_sch
- 复制MyMC14495.sym和.vf到工程根目录 vf {添加到工程中，ADD NEW SCOURSE}
- 在symbols框里的第一个元件，就是MyMC14495

4. 3 下载

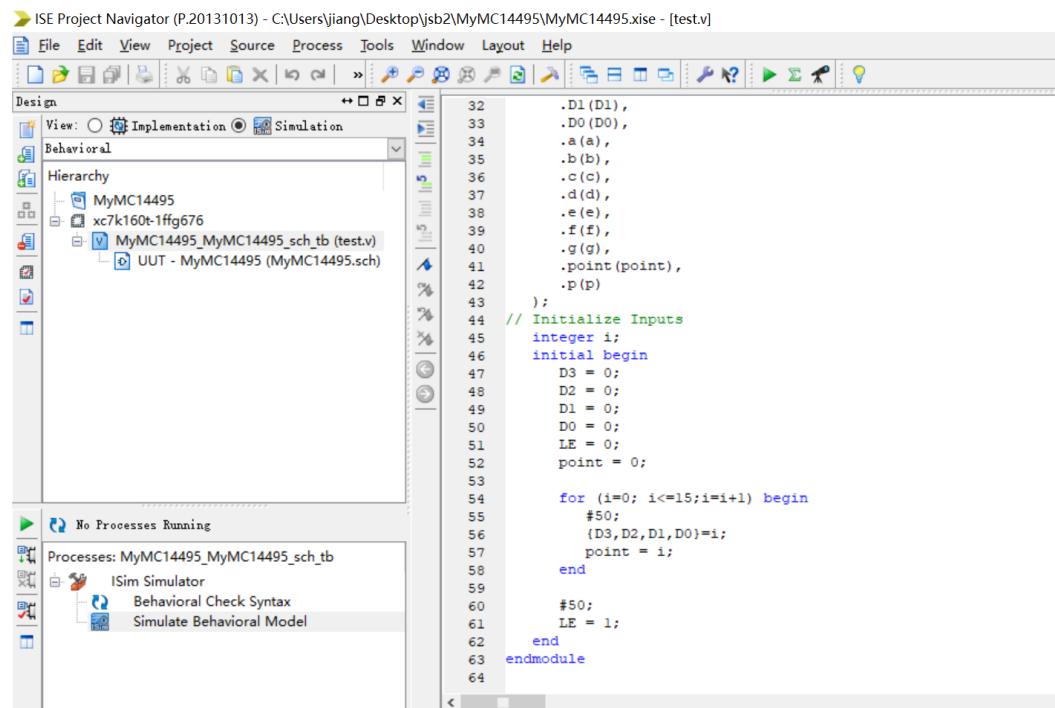
- UCF引脚定义
- 输入
 - SW[7:4]=AN[3:0]
 - SW[3:0]=D3D2D1D0
 - SW[14]=LE
 - SW[15]=point
- 输出
 - a[~]g, p

五、实验结果与分析

5.1 设计实现MY_MC14495



□ 以上为原理图设计



该截图展示了ISE Project Navigator窗口，显示了项目的文件结构和仿真设置。右侧是仿真脚本的内容：

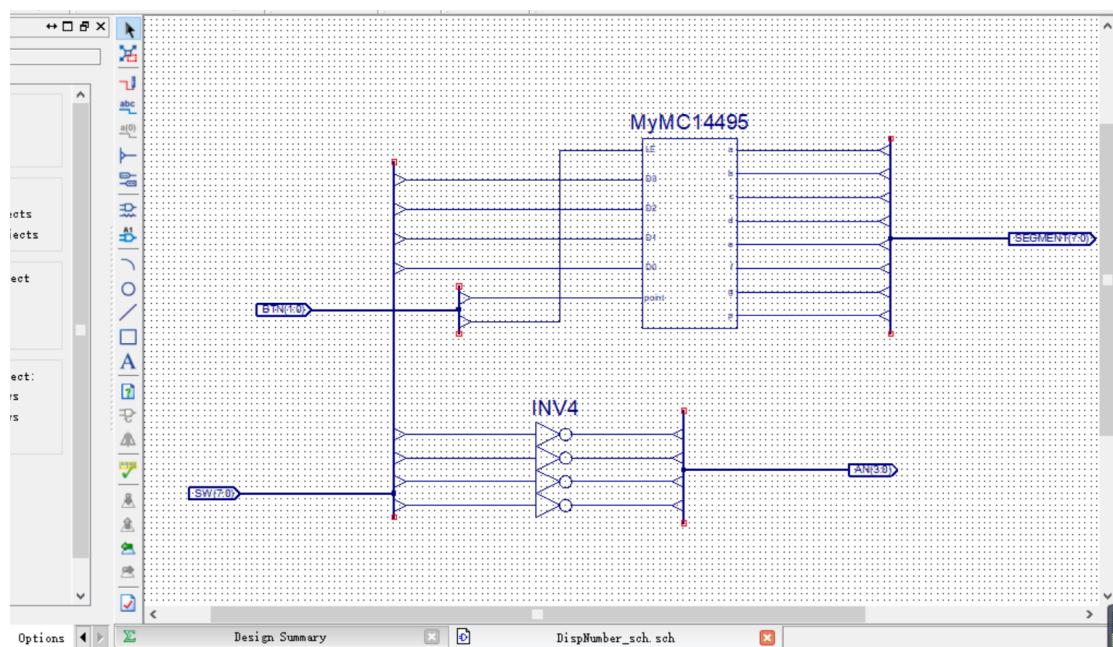
```
32 .D1(D1),
33 .D0(D0),
34 .a(a),
35 .b(b),
36 .c(c),
37 .d(d),
38 .e(e),
39 .f(f),
40 .g(g),
41 .point(point),
42 .p(p)
43 );
44 // Initialize Inputs
45 integer i;
46 initial begin
47   D3 = 0;
48   D2 = 0;
49   D1 = 0;
50   D0 = 0;
51   LE = 0;
52   point = 0;
53
54   for (i=0; i<=15; i=i+1) begin
55     #50;
56     (D3,D2,D1,D0)=i;
57     point = i;
58   end
59
60   #50;
61   LE = 1;
62 end
63 endmodule
64
```

□ 以上为仿真代码



□ 以上为仿真波形

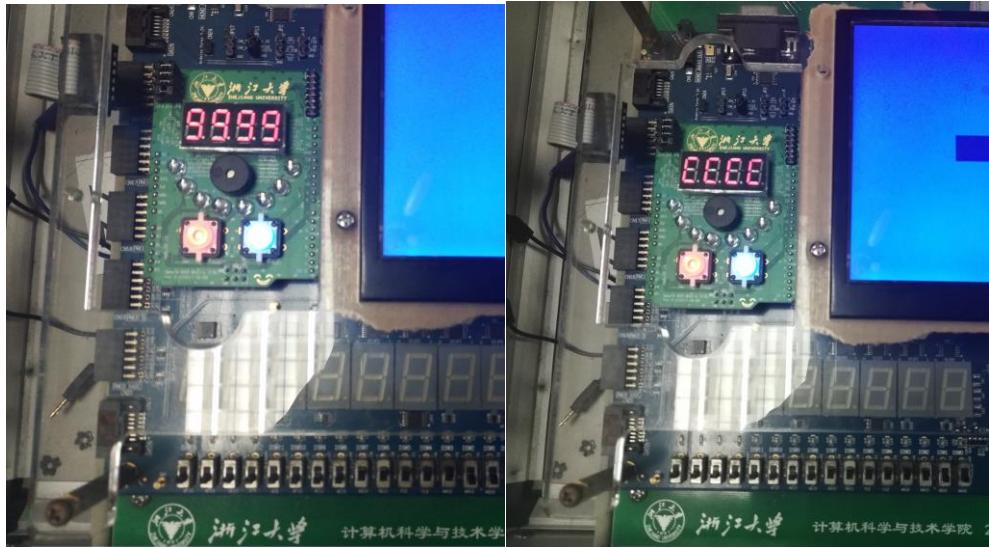
5.2 实现数码管显示



□ 以上为原理图设计

```
1 net "SW[0]" loc = AA10 | IOSTANDARD = LVCMOS15;
2 net "SW[1]" loc = AB10 | IOSTANDARD = LVCMOS15; net "SW[2]" loc = AA13 | IOSTANDARD = LVCMOS15;
3 net "SW[3]" loc = AA12 | IOSTANDARD = LVCMOS15;
4 net "SW[4]" loc = Y13 | IOSTANDARD = LVCMOS15;
5 net "SW[5]" loc = Y12 | IOSTANDARD = LVCMOS15;
6 net "SW[6]" loc = AD11 | IOSTANDARD = LVCMOS15;
7 net "SW[7]" loc = AD10 | IOSTANDARD = LVCMOS15;
8 NET"BTN[0]"LOC = AF13 | IOSTANDARD = LVCMOS15 ;#SW[14]
9 NET"BTN[1]"LOC = AF10 | IOSTANDARD = LVCMOS15 ;#SW[15]
10 NET"SEGMENT[0]"LOC = AB22 | IOSTANDARD = LVCMOS33 ;#a
11 NET"SEGMENT[1]"LOC = AD24 | IOSTANDARD = LVCMOS33 ;#b
12 NET"SEGMENT[2]"LOC = AD23 | IOSTANDARD = LVCMOS33 ;
13 NET"SEGMENT[3]"LOC = Y21 | IOSTANDARD = LVCMOS33 ;
14 NET"SEGMENT[4]"LOC = W20 | IOSTANDARD = LVCMOS33 ;
15 NET"SEGMENT[5]"LOC = AC24 | IOSTANDARD = LVCMOS33 ;
16 NET"SEGMENT[6]"LOC = AC23 | IOSTANDARD = LVCMOS33 ;#g
17 NET"SEGMENT[7]"LOC = AA22 | IOSTANDARD = LVCMOS33 ;#point
18
19 NET"AN[0]"LOC=AD21 | IOSTANDARD=LVCMOS33 ;
20 NET"AN[1]"LOC=AC21 | IOSTANDARD=LVCMOS33 ;
21 NET"AN[2]"LOC=AB21 | IOSTANDARD=LVCMOS33 ;
22 NET"AN[3]"LOC=AC22 | IOSTANDARD=LVCMOS33 ;
23
```

□ 以上为UCF引脚定义



□ 以上为成果展示（9和C）

实验七（多路选择器设计及应用）实验报告

姓名: 蒋仕彪 学号: 3170102587 专业: 求是科学班(计算机) 1701

课程名称: 逻辑与计算机设计基础实验

实验时间: 2018-11-01

一、实验目的

- 掌握数据选择器的工作原理和逻辑功能
- 掌握数据选择器的使用方法
- 掌握4位数码管扫描显示方法
- 4位数码管显示应用—记分板设计

二、实验内容和原理

2.1 实验内容:

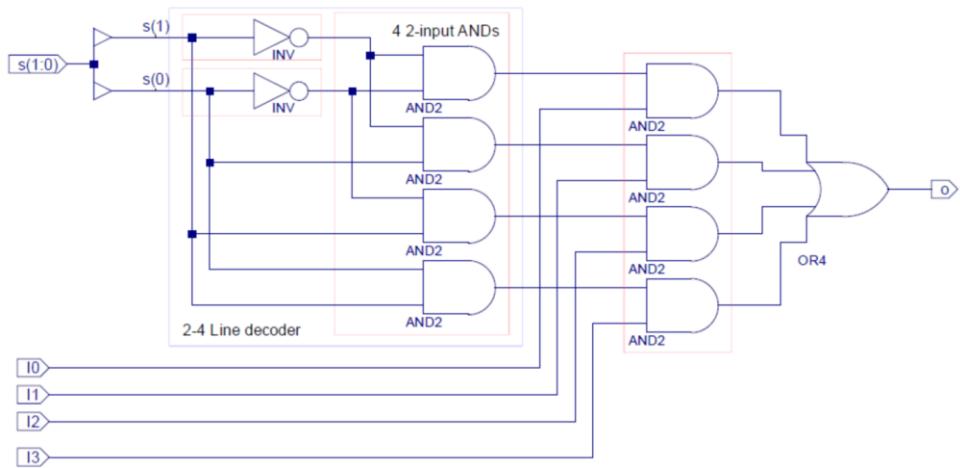
- 任务 1: 数据选择器设计
- 任务 2: 记分板设计

2.2 实验原理:

2.2.1 四选一多路选择器: MUX4to1

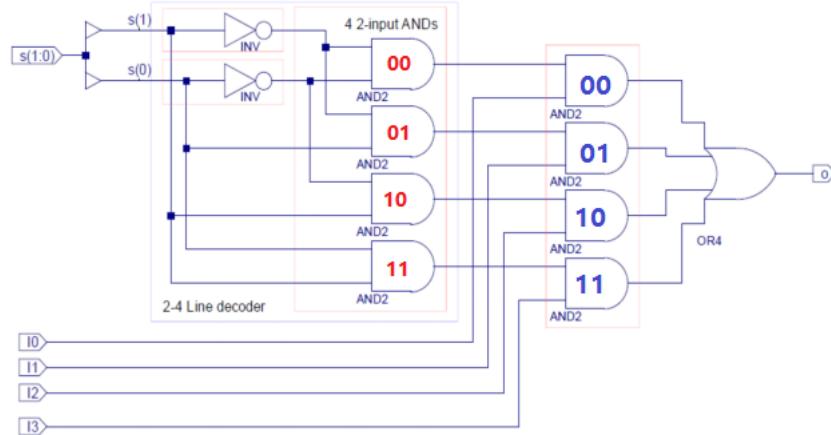
- 根据事件简化真值表
- 输出是控制信号 (全部最小项与或结构)

信息输入	控制端	选择输出		
I0 I1 I2 I3	S1 S0	o 输出项		
I0 I1 I2 I3	0 0	I0	<u>S1</u> <u>S0</u>	I0
I0 I1 I2 I3	0 1	I1	<u>S1</u> <u>S0</u>	I1
I0 I1 I2 I3	1 0	I2	<u>S1</u> <u>S0</u>	I2
I0 I1 I2 I3	1 1	I3	<u>S1</u> <u>S0</u>	I3

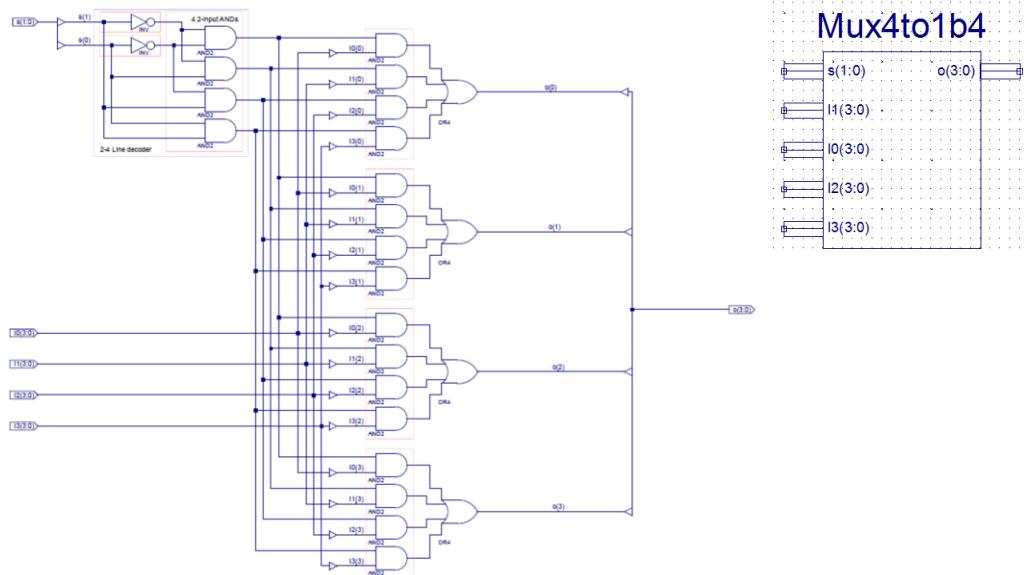


2.2.2 多路选择器位扩展

□ 控制结构不变，每路输入向量化



2.2.3 4位四选一扩展：MUX4to1b4



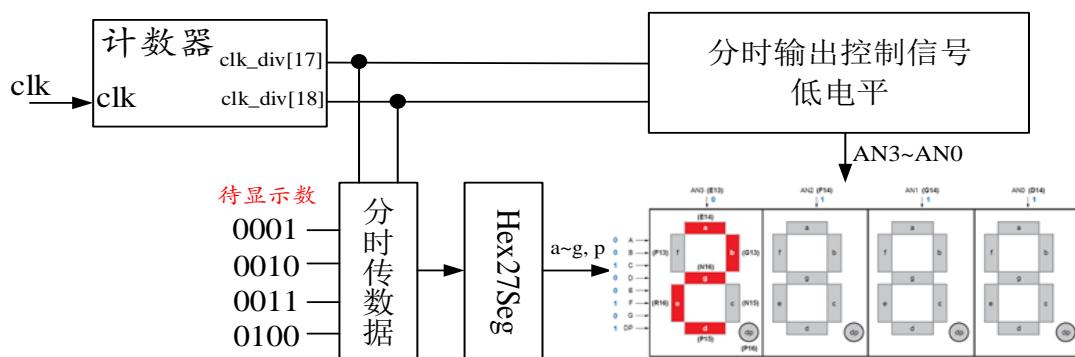
2.2.4 动态扫描显示

□ 动态扫描显示方案

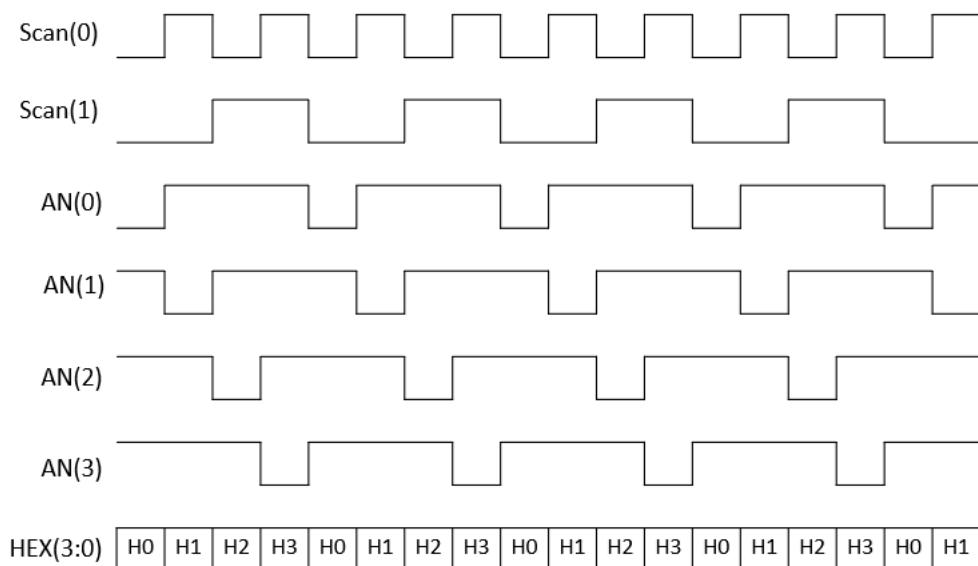
- 扫描信号来自计数器：时序转化为组合电路
- 由板载时钟clk(100MHz)作为计数器时钟，分频后输入到数据选择器的控制端，作为数码管扫描信号
- 计数器的分频系数要适当，眼睛舒适即可

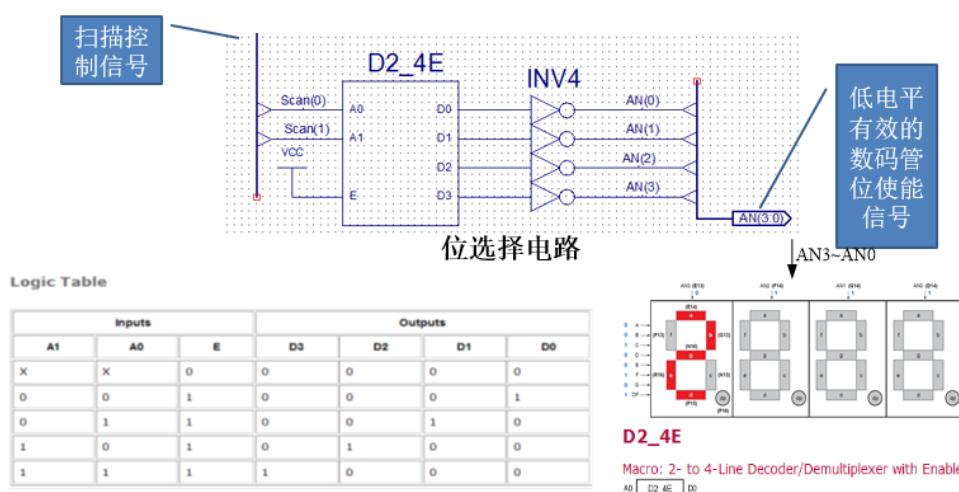
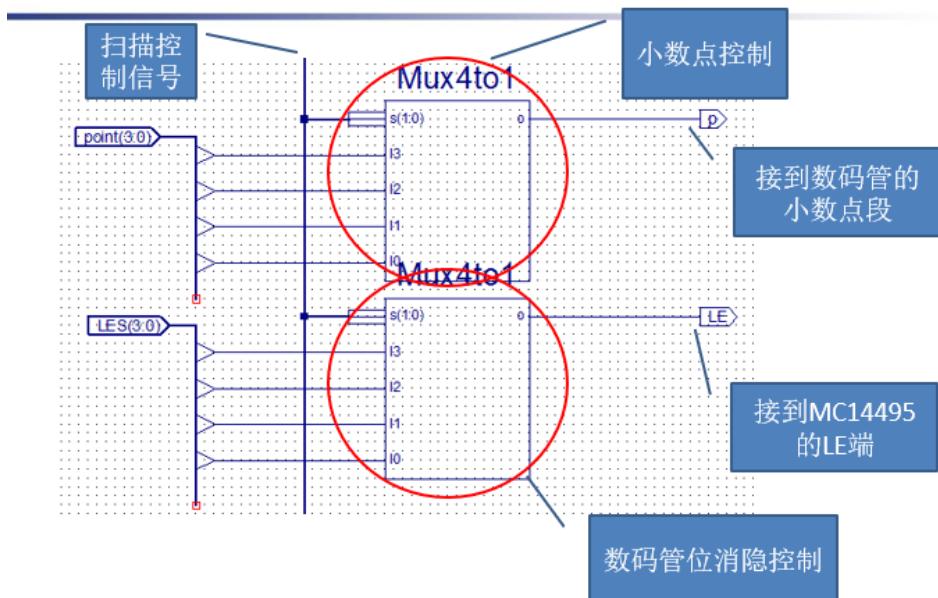
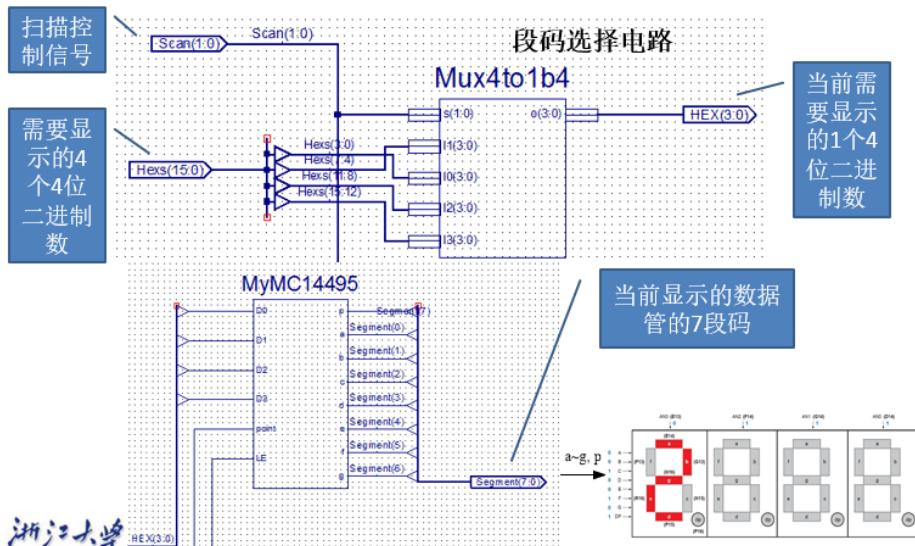
□ 条件语句实现

- if_then 或case语句实现条件输出电路

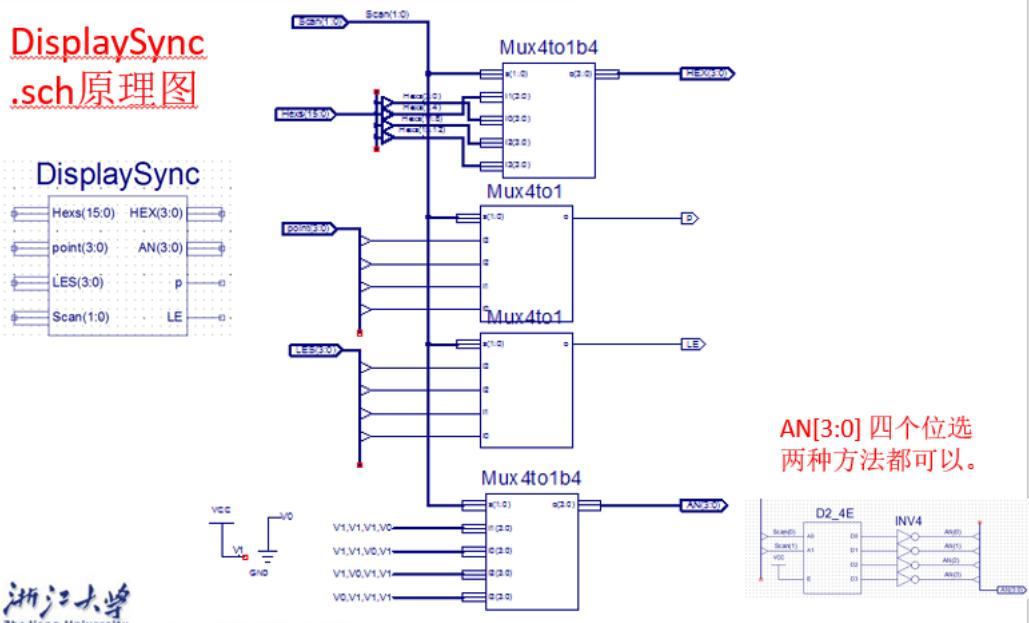


2.2.5 多路选择器应用：4位七段显示扫描控制





浙江大学



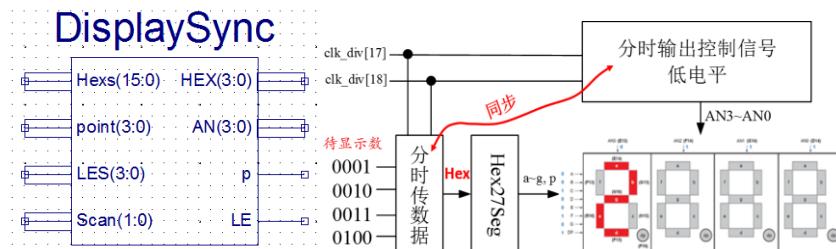
2.2.6 使用 Case 语句实现条件输出 (dispsync)

```

module dispsync(input [15:0] Hexs,           //端口变量说明与定义合并
                 input [1:0] Scan,
                 input [3:0] Point,
                 input [3:0] Les,
                 output reg[3:0] Hex,
                 output reg p,LE,
                 output reg[3:0] AN);

always @* begin                                //信号变化触发 (组合电路不用时钟触发)
    case (Scan)
        2'b00 : begin Hex <= Hexs[3:0];      AN <= 4'b 1110; ... //同步输出
        2'b01 : begin Hex <= Hexs[7:4];      AN <= 4'b 1101; ... //同步输出
        2'b10 : begin Hex <= Hexs[11:8];     AN <= 4'b 1011; ... //同步输出
        2'b11 : begin Hex <= Hexs[15:12];    AN <= 4'b 0111; ... //同步输出
    endcase
end
endmodule

```



2.2.7 辅助模块：时钟计数分频器

□ 32 位时钟计数分频器

- 可输出 $2-2^{32}$ 分频信号，可用于一般非同步类时钟信号
- 延时较高，要求不高的时钟也可以用
- 本实验多位七段显示器动态扫描可用

```
module clkdiv(input clk,
               input rst,
               output reg[31:0] clkdiv
             );
  // Clock divider-时钟分频器

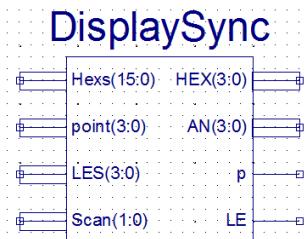
  always @ (posedge clk or posedge rst) begin
    if (rst) clkdiv <= 0;
    else clkdiv <= clkdiv + 1'b1;
  end

endmodule
```

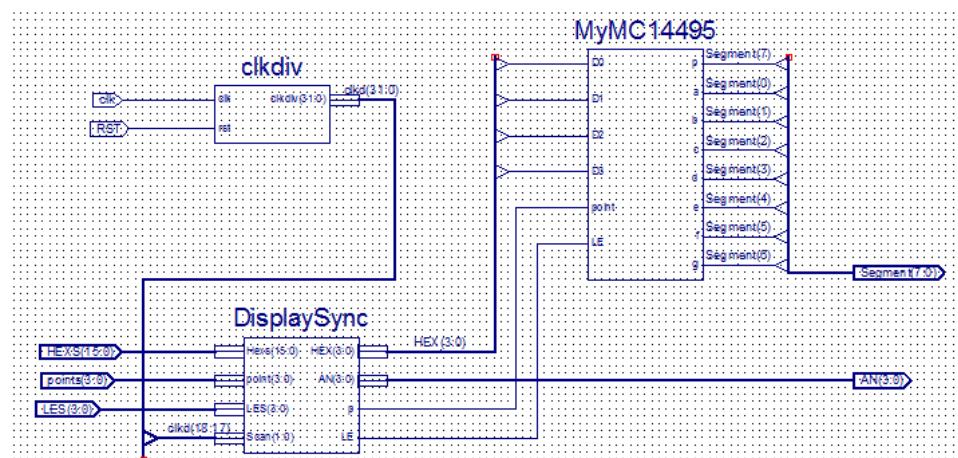
2.2.8 相关模块设计

□ 设计动态扫描同步输出模块

- 模块名: DisplaySync.sch
- 用原理图设计
- 制作逻辑符号并修改: DisplaySync.sym
- 设计通用计数分频模块
- 模块名: clkdiv.v
- 用 Verilog HDL 设计
- 制作逻辑符号并修改: clkdiv.sym



2.2.9 设计 disp_num 显示模块



2.2.10 设计按键数据输入模块

- 使用行为描述设计
 - 四个按键，各按一下，4个4位2进制数分别加1

```
////////////////////////////////////////////////////////////////
module CreateNumber(
    input wire [3:0] btn,
    output reg [15:0] num
);
wire [3:0] A,B,C,D;

initial num <= 16'b1010_1011_1100_1101; // display "AbCd"

assign A = num[ 3: 0] + 4'd1;
assign B = num[ 7: 4] + 4'd1;
assign C = num[11: 8] + 4'd1;
assign D = num[15:12] + 4'd1;

always@ (posedge btn[0]) num[ 3: 0]<= A;
always@ (posedge btn[1]) num[ 7: 4]<= B;
always@ (posedge btn[2]) num[11: 8]<= C;
always@ (posedge btn[3]) num[15:12]<= D;

endmodule
```

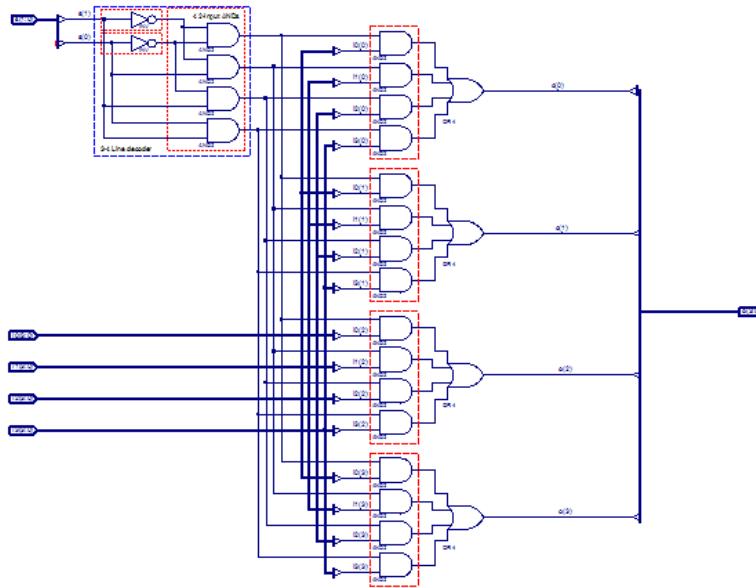
三、主要仪器设备

- 装有 Xilinx ISE 14.7 的计算机 1 台
- SWORD 开发板 1 套

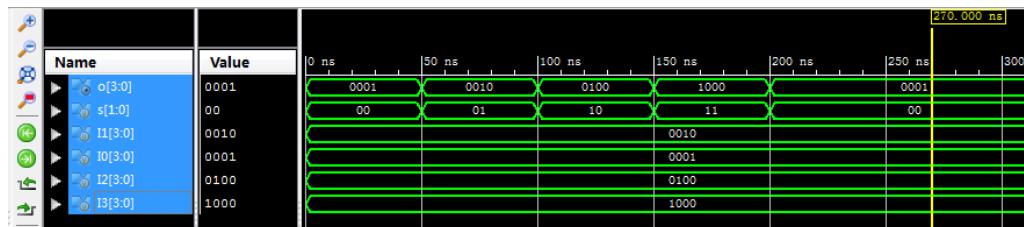
四、操作方法与实验步骤

4.1 数据选择器设计

- 新建工程
 - 工程名称用 Mux4to1b4_sch。
- 新建源文件
 - 类型是 Schematic
 - 文件名称用 Mux4to14b。
- 原理图方式进行设计



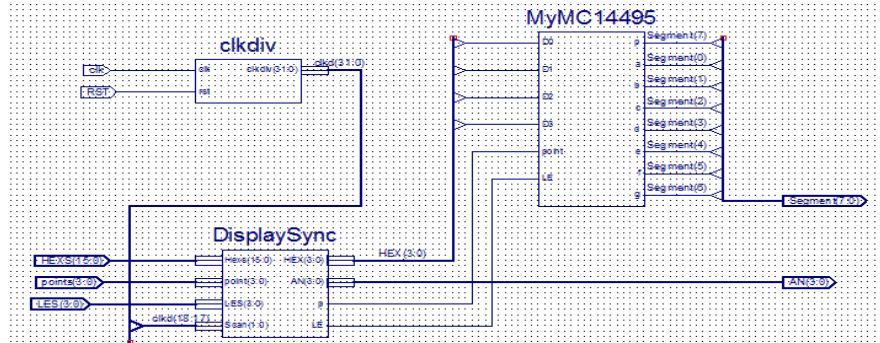
- 仿真波形



4.2 记分板应用设计

- 新建工程
 - 工程名称用 ScoreBoard。
 - Top Level Source Type 用 HDL
 - new source 文件名称: DisplaySync.sch
- 根据原理图设计动态扫描同步输出模块
文件名称: clkdiv.v
- 根据原理图设计通用计数分频模块

- 新建源文件
 - 类型是 Schematic
 - 文件名称用 disp_num。
 - 原理图方式进行设计显示模块



- 新建源文件 top，并右键设为“Top Module”

```

module top(input wire clk,
            input wire [7:0] SW,
            input wire [3:0] btn,
            output wire [3:0] AN,
            output wire [7:0] SEGMENT,
            );
            wire [15:0] num;
            CreateNumber c0(btn,num);

            disp_num d0(clk, num, SW[7:4], SW[3:0], 1'b0, AN, SEGMENT);

endmodule

```

4.3 物理验证

- UCF 引脚定义
 - 输入
 - 时钟: clk
 - 使能控制: sw[7:4] 为 les[3:0]
 - 小数点输入: sw[3:0] 为 point[3:0]
 - 按键输入数字: sw[15:12] 为 btn[3:0]
 - 输出
 - \tilde{a}^g , p=segment
 - an[3:0]
- 根据设计修改 UCF
 - 说明: sym, vf {添加到工程中, ADD NEW SCOUR}
 - 引脚约束 UCF 说明

```

□ NET "clk" LOC = AC18 | IO_STANDARD = LVCMOS18 ;
□ net "btn" loc = "xxx" ;
□ net "btn" clock_dedicated_route = false;
□ NET "AN[0]" LOC = AC21 | IO_STANDARD = LVCMOS33 ;
□ NET "AN[1]" LOC = AD21 | IO_STANDARD = LVCMOS33 ;
□ NET "AN[2]" LOC = AB21 | IO_STANDARD = LVCMOS33 ;
□ NET "AN[3]" LOC = AC22 | IO_STANDARD = LVCMOS33 ;

```

五、实验结果与分析

5.1 数据选择器设计

```

`timescale 1ns / 1ps

module Mux4to1b4(I0,
                  I1,
                  I2,
                  I3,
                  S,
                  O);
    input [3:0] I0;
    input [3:0] I1;
    input [3:0] I2;
    input [3:0] I3;
    input [1:0] S;
    output [3:0] O;

    wire XLXN_5;
    wire XLXN_9;
    wire XLXN_12;
    wire XLXN_13;
    wire XLXN_14;
    wire XLXN_16;
    wire XLXN_17;
    wire XLXN_18;
    wire XLXN_19;
    wire XLXN_20;
    wire XLXN_21;
    wire XLXN_22;
    wire XLXN_23;
    wire XLXN_24;
    wire XLXN_25;
    wire XLXN_26;
    wire XLXN_27;
    wire XLXN_28;
    wire XLXN_75;
    wire XLXN_78;
    wire XLXN_81;
    wire XLXN_83;

    INV XLXI_1 (.I(S[1]),
                 .O(XLXN_5));
    INV XLXI_2 (.I(S[0]),
                 .O(XLXN_9));
    AND2 XLXI_3 (.I0(XLXN_9),
                  .I1(XLXN_5),
                  .O(XLXN_75));
    AND2 XLXI_4 (.I0(S[0]),
                  .I1(XLXN_5),
                  .O(XLXN_78));
    AND2 XLXI_5 (.I0(S[1]),
                  .I1(XLXN_9),

```

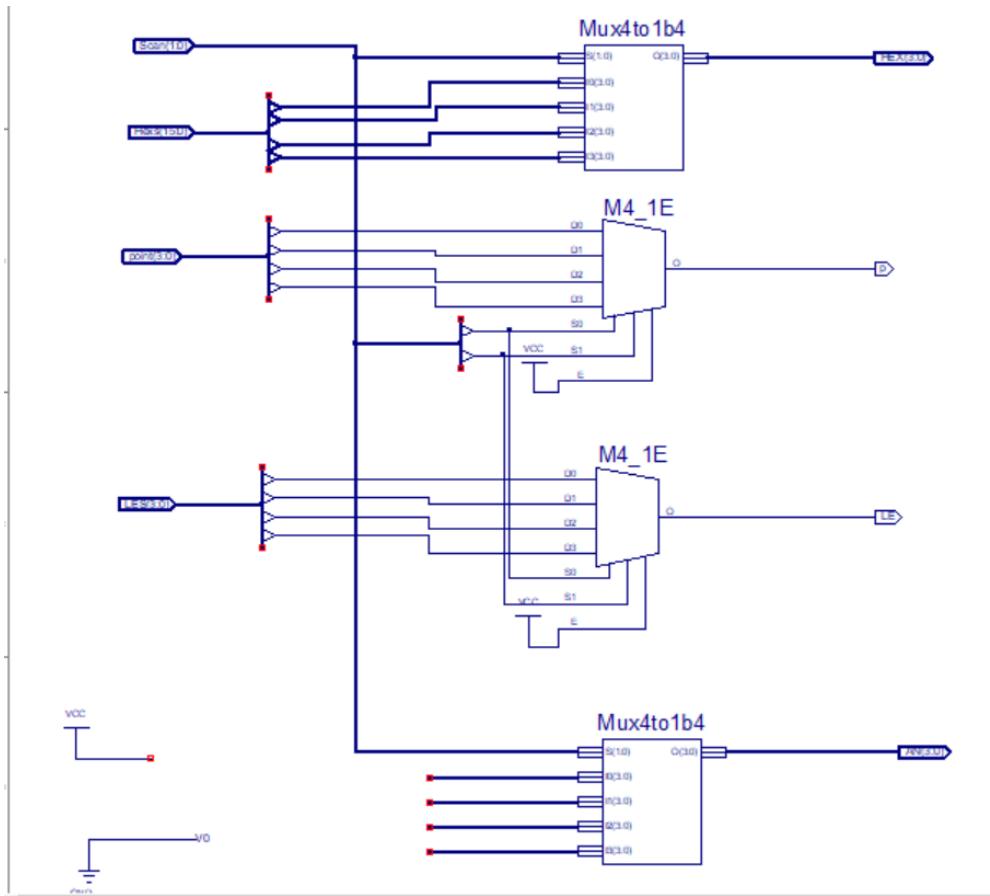
```

        .O(XLXN_81));
AND2 XLXI_6 (.I0(S[1]),
    .I1(S[0]),
    .O(XLXN_83));
AND2 XLXI_7 (.I0(I0[0]),
    .I1(XLXN_75),
    .O(XLXN_12));
AND2 XLXI_8 (.I0(I1[0]),
    .I1(XLXN_78),
    .O(XLXN_13));
AND2 XLXI_9 (.I0(I2[0]),
    .I1(XLXN_81),
    .O(XLXN_14));
AND2 XLXI_10 (.I0(I3[0]),
    .I1(XLXN_83),
    .O(XLXN_16));
AND2 XLXI_11 (.I0(I0[1]),
    .I1(XLXN_75),
    .O(XLXN_17));
AND2 XLXI_12 (.I0(I1[1]),
    .I1(XLXN_78),
    .O(XLXN_18));
AND2 XLXI_13 (.I0(I2[1]),
    .I1(XLXN_81),
    .O(XLXN_19));
AND2 XLXI_14 (.I0(I3[1]),
    .I1(XLXN_83),
    .O(XLXN_20));
AND2 XLXI_15 (.I0(I0[2]),
    .I1(XLXN_75),
    .O(XLXN_21));
AND2 XLXI_16 (.I0(I1[2]),
    .I1(XLXN_78),
    .O(XLXN_22));
AND2 XLXI_17 (.I0(I2[2]),
    .I1(XLXN_81),
    .O(XLXN_23));
AND2 XLXI_18 (.I0(I3[2]),
    .I1(XLXN_83),
    .O(XLXN_24));
AND2 XLXI_47 (.I0(I3[3]),
    .I1(XLXN_83),
    .O(XLXN_28));
AND2 XLXI_48 (.I0(I2[3]),
    .I1(XLXN_81),
    .O(XLXN_27));
AND2 XLXI_49 (.I0(I1[3]),
    .I1(XLXN_78),
    .O(XLXN_26));
AND2 XLXI_50 (.I0(I0[3]),
    .I1(XLXN_75),
    .O(XLXN_25));
OR4 XLXI_99 (.I0(XLXN_16),
    .I1(XLXN_14),
    .I2(XLXN_13),
    .I3(XLXN_12),
    .O(O[0]));
OR4 XLXI_100 (.I0(XLXN_20),
    .I1(XLXN_19),
    .I2(XLXN_18),
    .I3(XLXN_17),
    .O(O[1]));
OR4 XLXI_101 (.I0(XLXN_24),
    .I1(XLXN_23),
    .I2(XLXN_22),
    .I3(XLXN_21),
    .O(O[2]));
OR4 XLXI_102 (.I0(XLXN_28),
    .I1(XLXN_27),
    .I2(XLXN_26),
    .I3(XLXN_25),
    .O(O[3]));
Endmodule

```

□ (因为遗失 sch 的图，只能粘.v 的代码过来作为 Mux4to1b4 原理图)

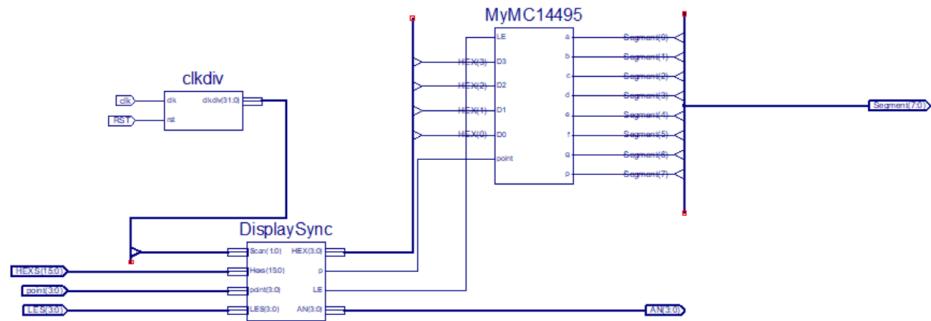
5.2 记分板应用设计



□ 以上为计分板的原理图

```
1 //timescale 1ns / 1ps
2 /////////////////////////////////
3 // Company:
4 // Engineer:
5 //
6 // Create Date: 20:38:16 11/01/2018
7 // Design Name:
8 // Module Name: clkdiv
9 // Project Name:
10 // Target Devices:
11 // Tool versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 ///////////////////////////////
21 module clkdiv(
22     input clk,
23     input rst,
24     output reg[31:0] clkdiv
25 );
26     always @ (posedge clk or posedge rst) begin
27         if (rst) clkdiv<=0;
28         else clkdiv<=clkdiv+1'b1;
29     end
30
31
32 endmodule
33
```

□ 以上为 clk.v 的代码



□ 以上为 dispnum 的原理图

```

1 `timescale 1ns / 1ps
2 ///////////////////////////////////////////////////////////////////
3 // Company:
4 // Engineer:
5 //
6 // Create Date: 21:20:06 11/01/2018
7 // Design Name:
8 // Module Name: top
9 // Project Name:
10 // Target Devices:
11 // Tool versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 ///////////////////////////////////////////////////////////////////
21 module top(input wire clk,
22     input wire [7:0] SW,
23     input wire [3:0] btn,
24     output wire [3:0] AN,
25     output wire [7:0] SEGMENT
26 );
27     wire [15:0] num;
28     CreateNumber c0(btn,num);
29     disp_num d0(clk, num, SW[7:4], SW[3:0], 1'b0, AN, SEGMENT);
30
31 endmodule
32

```

□ 以上为 top.v 的代码

```

7 // Design Name:
8 // Module Name:      CreateNumber
9 // Project Name:
10 // Target Devices:
11 // Tool versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 /////////////////////////////////
21 module CreateNumber(
22     input wire [3:0] btn,
23     output reg [15:0] num
24 );
25     wire [3:0] A,B,C,D;
26     initial num <= 16'b1010_1011_1100_1101;
27
28     assign A=num[3:0]+4'd1;
29     assign B=num[7:4]+4'd1;
30     assign C=num[11:8]+4'd1;
31     assign D=num[15:12]+4'd1;
32
33     always@(posedge btn[0]) num[3:0]<=A;
34     always@(posedge btn[1]) num[7:4]<=B;
35     always@(posedge btn[2]) num[11:8]<=C;
36     always@(posedge btn[3]) num[15:12]<=D;
37
38 endmodule
39

```

□ 以上为 CreateNumber 的代码

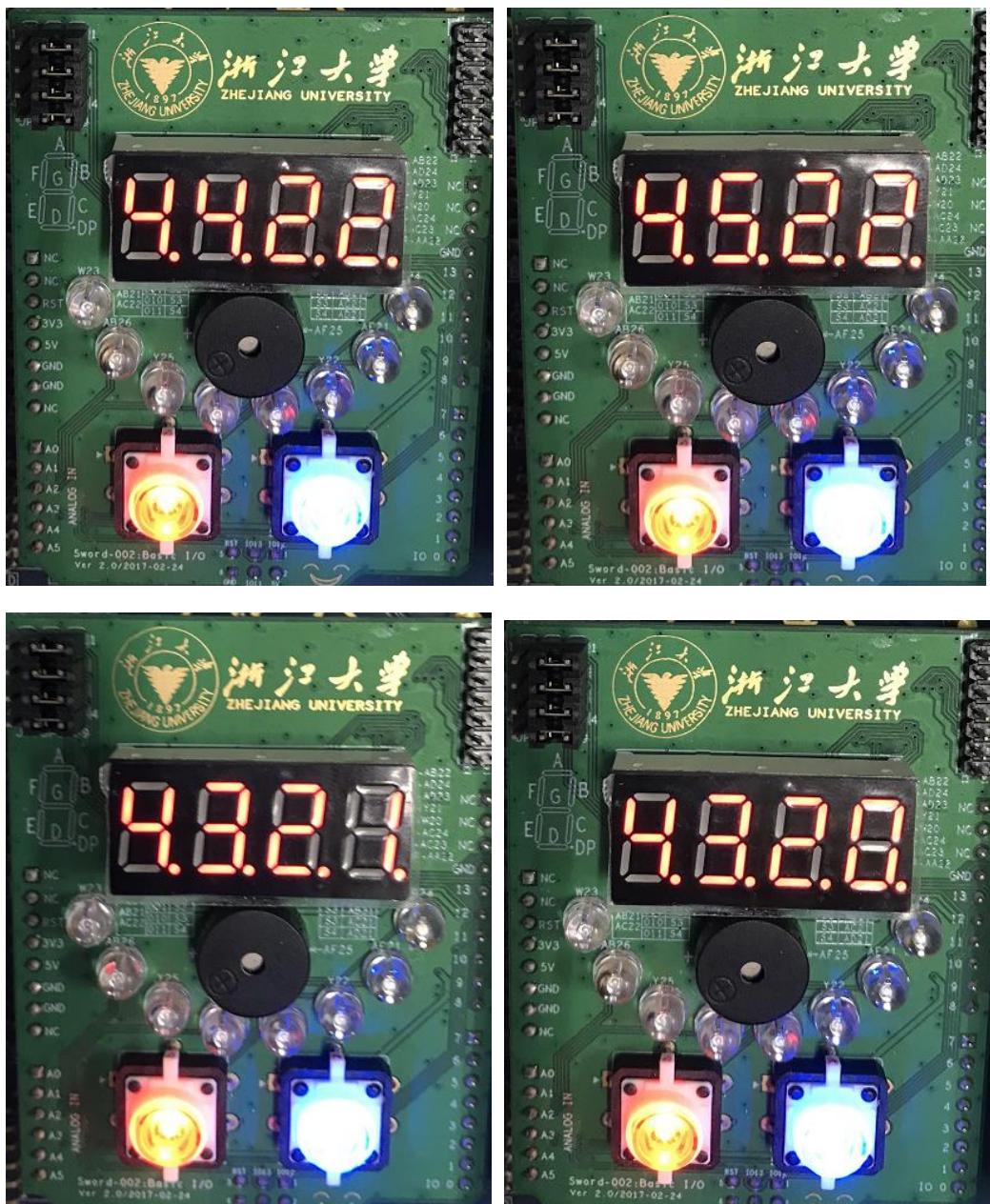
□ 5.3 物理验证

```

1 net "SW[0]" loc = AA10 | IOSTANDARD = LVCMOS15;
2 net "SW[1]" loc = AB10 | IOSTANDARD = LVCMOS15;
3 net "SW[2]" loc = AA13 | IOSTANDARD = LVCMOS15;
4 net "SW[3]" loc = AA12 | IOSTANDARD = LVCMOS15;
5 net "SW[4]" loc = Y13 | IOSTANDARD = LVCMOS15;
6 net "SW[5]" loc = Y12 | IOSTANDARD = LVCMOS15;
7 net "SW[6]" loc = AD11 | IOSTANDARD = LVCMOS15;
8 net "SW[7]" loc = AD10 | IOSTANDARD = LVCMOS15;
9
10 NET "btn[0]" LOC = AF13 | IOSTANDARD = LVCMOS15;#SW[14]
11 net "btn[0]" clock_dedicated_route = false;
12 NET "btn[2]" LOC = AF8 | IOSTANDARD = LVCMOS15;#SW[14]
13 net "btn[2]" clock_dedicated_route = false;
14 NET "btn[3]" LOC = AE13 | IOSTANDARD = LVCMOS15;#SW[14]
15 net "btn[3]" clock_dedicated_route = false;
16 NET "btn[1]" LOC = AF10 | IOSTANDARD = LVCMOS15;#SW[15]
17 net "btn[1]" clock_dedicated_route = false;
18
19 net "clk" LOC = AE8 | IOSTANDARD = LVCMOS15;
20 NET "SEGMENT[0]" LOC = AB22 | IOSTANDARD = LVCMOS33;#a
21 NET "SEGMENT[1]" LOC = AD24 | IOSTANDARD = LVCMOS33;#b
22 NET "SEGMENT[2]" LOC = AD23 | IOSTANDARD = LVCMOS33;
23 NET "SEGMENT[3]" LOC = Y21 | IOSTANDARD = LVCMOS33;
24 NET "SEGMENT[4]" LOC = W20 | IOSTANDARD = LVCMOS33;
25 NET "SEGMENT[5]" LOC = AC24 | IOSTANDARD = LVCMOS33;
26 NET "SEGMENT[6]" LOC = AC23 | IOSTANDARD = LVCMOS33;#g
27 NET "SEGMENT[7]" LOC = AA22 | IOSTANDARD = LVCMOS33;#point
28
29 NET "AN[0]" LOC = AD21 | IOSTANDARD = LVCMOS33;
30 NET "AN[1]" LOC = AC21 | IOSTANDARD = LVCMOS33;
31 NET "AN[2]" LOC = AB21 | IOSTANDARD = LVCMOS33;
32 NET "AN[3]" LOC = AC22 | IOSTANDARD = LVCMOS33;
33

```

□ 以上为 ucf 引脚定义



□ 以上为实际成果