实验七

课程名称：数字逻辑电路设计 实验类型：综合

实验项目名称：多路选择器设计及应用

学生姓名：段皞一 学号：3190105359 同组学生姓名：无

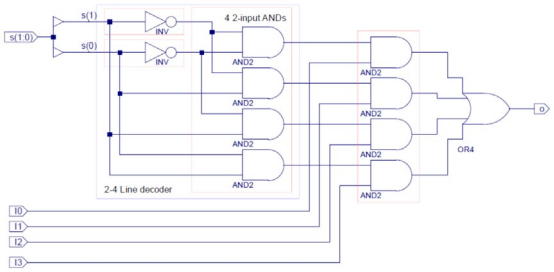
实验地点：紫金港东四 509 室 实验日期：2020 年 10 月 26 日

一、操作方法与实验步骤

1.1 实验原理

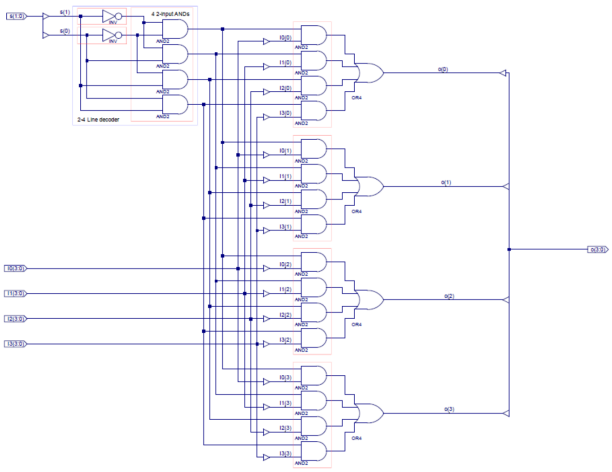
1.1.1 4 选 1 多路选择器原理

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	S1S0 I0
I0 I1 I2 I3	0 1	I1	S1S0 I1
I0 I1 I2 I3	1 0	I2	S1S0 I2
I0 I1 I2 I3	1 1	I3	S1S0 I3



4 选 1 多路选择器可以根据事件简化真值表，输出是控制信号全部最小项。控制结构不变，每路输入向量化。

1.1.2 4 位四选一扩展：MUX4to1b4



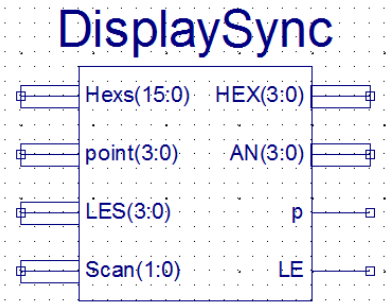
1.1.3 使用 Case 语句实现条件输出

```
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
```

1.1.4 相关模块设计

1.1.4.1 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



1.1.4.2 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

1.2 任务一——数据选择器设计

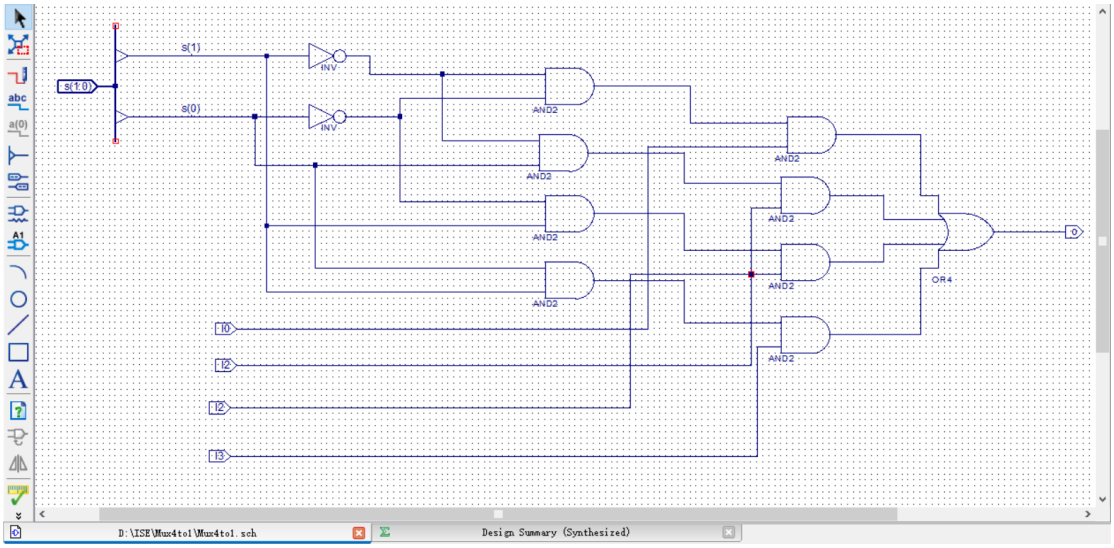
1.2.1 Mux4to1 模块

1.2.1.1 用画图的方式设计 Mux4to1 模块

1.新建工程，命名为 Mux4to1_sch。

2.在工程下新建源文件，类型 Schematic,文件名称为 Mux4to1。

3.按照原理图，进行连线。



4.连接完成后，双击 Processes 窗口下的 Check Design Rules，成功后点击 View HDL Functional Model,观看并学习生成的 Verilog 源代码。

```
//////////////////////////////////////////////////////////////////////////////
module Mux4to1(I0,
               I1,
               I2,
               I3,
               s,
               o);

    input I0;
    input I1;
    input I2;
    input I3;
    input [1:0] s;
   output o;

   wire XLXN_1;
   wire XLXN_2;
   wire XLXN_3;
   wire XLXN_4;
   wire XLXN_6;
   wire XLXN_7;
   wire XLXN_8;
   wire XLXN_9;
   wire XLXN_10;
   wire XLXN_13;

   AND2  XLXI_1 (.I0(I0),
                 .I1(XLXN_6),
                 .O(XLXN_1));
   AND2  XLXI_2 (.I0(I1),
                 .I1(XLXN_7),
                 .O(XLXN_2));
   AND2  XLXI_3 (.I0(I2),
                 .I1(XLXN_8),
                 .O(XLXN_3));
   AND2  XLXI_4 (.I0(I3),
                 .I1(XLXN_9),
                 .O(XLXN_4));
   OR4  XLXI_5 (.I0(XLXN_4),
                .I1(XLXN_3),
                .I2(XLXN_2),
                .I3(XLXN_1),
```

```
            .O(o));
  AND2  XLXI_6 (.I0(XLXN_13),
            .I1(XLXN_10),
            .O(XLXN_6));
  AND2  XLXI_7 (.I0(s[0]),
            .I1(XLXN_10),
            .O(XLXN_7));
  AND2  XLXI_8 (.I0(s[1]),
            .I1(XLXN_13),
            .O(XLXN_8));
  AND2  XLXI_9 (.I0(s[1]),
            .I1(s[0]),
            .O(XLXN_9));
  INV  XLXI_10 (.I(s[1]),
            .O(XLXN_10));
  INV  XLXI_11 (.I(s[0]),
            .O(XLXN_13));
endmodule
```

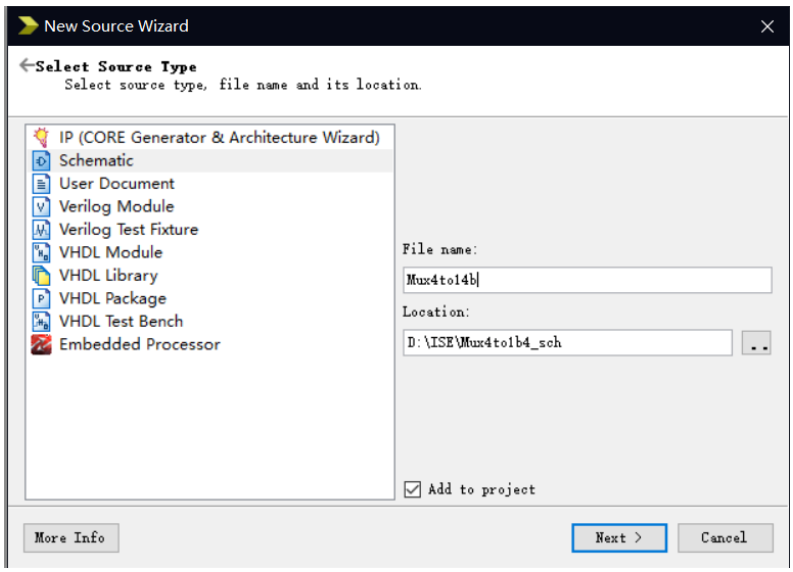
1.2.1.2 建立.vf 文件，.sym 文件

1.2.2 Mux4to1b4

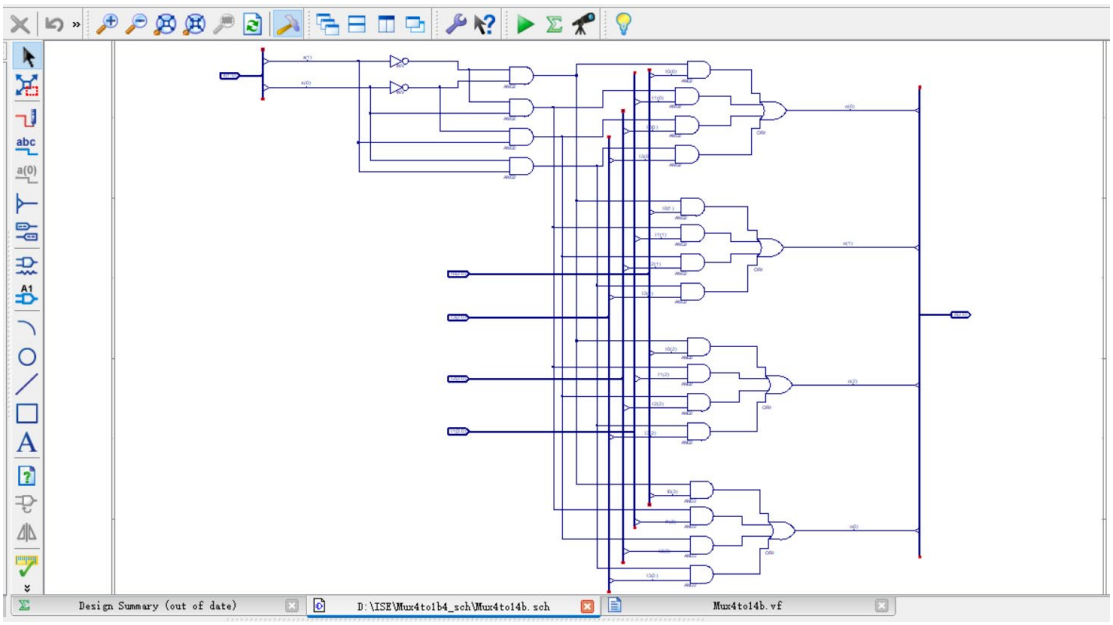
1.2.2.1 用画图的方法设计 Mux4to1b4 位扩展模块

1.新建工程，命名为 Mux4to1b4_sch。

2.在工程下新建源文件，类型 Schematic,文件名称为 Mux4to1b4。



3.以原理图的方式进行电路的设计。



4.连接完成后，双击 Processes 窗口下的 Check Design Rules，成功后点击 View HDL Functional Model,观看并学习生成的 Verilog 源代码。

```
//////////////////////////////////////////////////////////////////////////////////
module Mux4to14b(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_1;
   wire XLXN_2;
   wire XLXN_3;
   wire XLXN_4;
   wire XLXN_5;
   wire XLXN_6;
   wire XLXN_7;
   wire XLXN_8;
   wire XLXN_9;
   wire XLXN_10;
   wire XLXN_11;
```

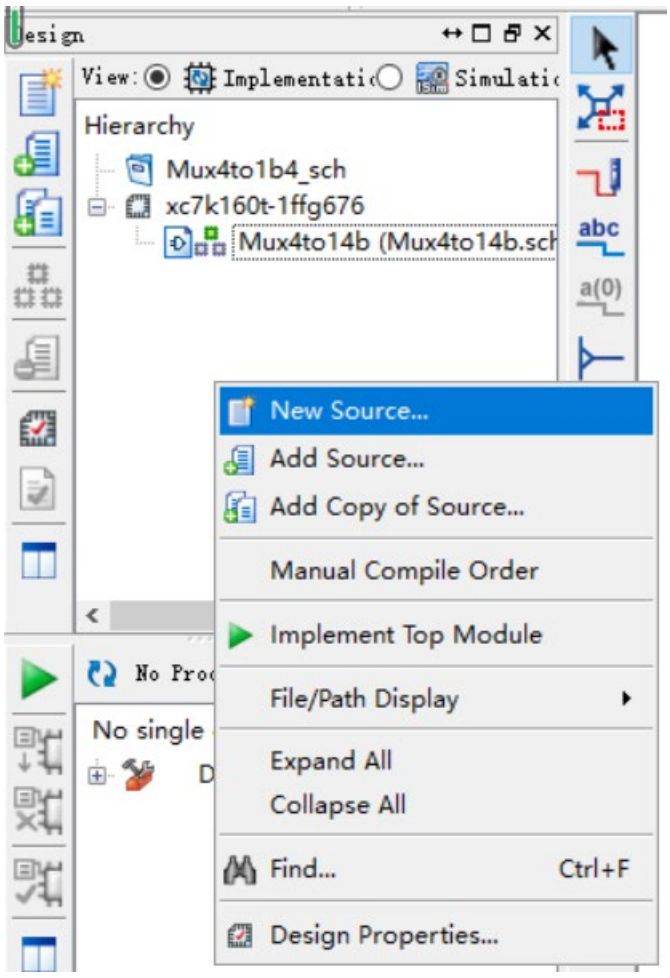
```
wire XLXN_12;
wire XLXN_13;
wire XLXN_14;
wire XLXN_15;
wire XLXN_16;
wire XLXN_22;
wire XLXN_23;
wire XLXN_24;
wire XLXN_47;
wire XLXN_66;
wire XLXN_67;

AND2  XLXI_1 (.I0(I0[0]),
              .I1(XLXN_47),
              .O(XLXN_1));
AND2  XLXI_2 (.I0(I1[0]),
              .I1(XLXN_22),
              .O(XLXN_2));
AND2  XLXI_3 (.I0(I2[0]),
              .I1(XLXN_23),
              .O(XLXN_3));
AND2  XLXI_4 (.I0(I3[0]),
              .I1(XLXN_24),
              .O(XLXN_4));
AND2  XLXI_5 (.I0(I0[1]),
              .I1(XLXN_47),
              .O(XLXN_5));
AND2  XLXI_6 (.I0(I1[1]),
              .I1(XLXN_22),
              .O(XLXN_6));
AND2  XLXI_7 (.I0(I2[1]),
              .I1(XLXN_23),
              .O(XLXN_7));
AND2  XLXI_8 (.I0(I3[1]),
              .I1(XLXN_24),
              .O(XLXN_8));
AND2  XLXI_9 (.I0(I0[2]),
              .I1(XLXN_47),
              .O(XLXN_9));
AND2  XLXI_10 (.I0(I1[2]),
               .I1(XLXN_22),
               .O(XLXN_10));
AND2  XLXI_11 (.I0(I2[2]),
               .I1(XLXN_23),
```

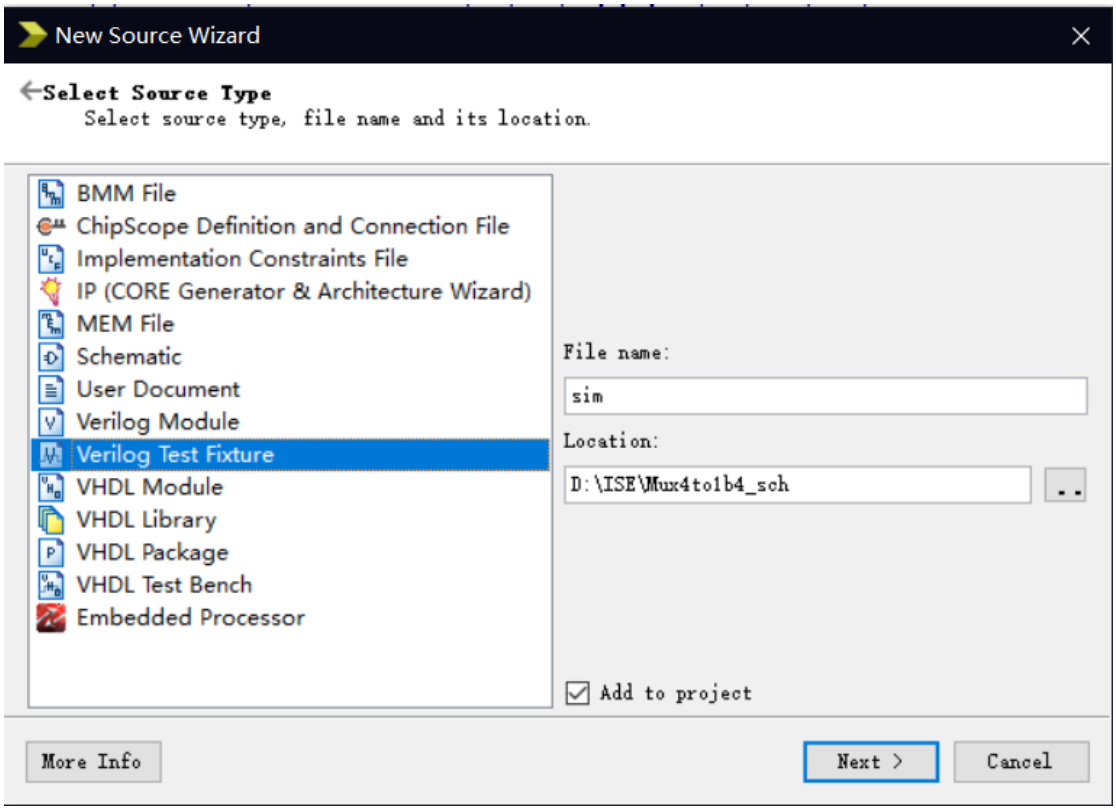
```
            .O(XLXN_11));
AND2  XLXI_12 (.I0(I3[2]),
            .I1(XLXN_24),
            .O(XLXN_12));
AND2  XLXI_13 (.I0(I0[3]),
            .I1(XLXN_47),
            .O(XLXN_13));
AND2  XLXI_14 (.I0(I1[3]),
            .I1(XLXN_22),
            .O(XLXN_14));
AND2  XLXI_15 (.I0(I2[3]),
            .I1(XLXN_23),
            .O(XLXN_15));
AND2  XLXI_16 (.I0(I3[3]),
            .I1(XLXN_24),
            .O(XLXN_16));
OR4  XLXI_33 (.I0(XLXN_4),
           .I1(XLXN_3),
           .I2(XLXN_2),
           .I3(XLXN_1),
           .O(o[0]));
OR4  XLXI_34 (.I0(XLXN_8),
           .I1(XLXN_7),
           .I2(XLXN_6),
           .I3(XLXN_5),
           .O(o[1]));
OR4  XLXI_35 (.I0(XLXN_12),
           .I1(XLXN_11),
           .I2(XLXN_10),
           .I3(XLXN_9),
           .O(o[2]));
OR4  XLXI_36 (.I0(XLXN_16),
           .I1(XLXN_15),
           .I2(XLXN_14),
           .I3(XLXN_13),
           .O(o[3]));
AND2  XLXI_37 (.I0(XLXN_66),
            .I1(XLXN_67),
            .O(XLXN_47));
AND2  XLXI_38 (.I0(s[0]),
            .I1(XLXN_67),
            .O(XLXN_22));
AND2  XLXI_39 (.I0(s[1]),
            .I1(XLXN_66),
```

```
               .O(XLXN_23));
  AND2  XLXI_40 (.I0(s[1]),
               .I1(s[0]),
               .O(XLXN_24));
  INV  XLXI_41 (.I(s[1]),
              .O(XLXN_67));
  INV  XLXI_42 (.I(s[0]),
              .O(XLXN_66));
endmodule
```

1.2.2.2 仿真波形检验模块设计的正确性

新建一个 Verilog Test Fixture。

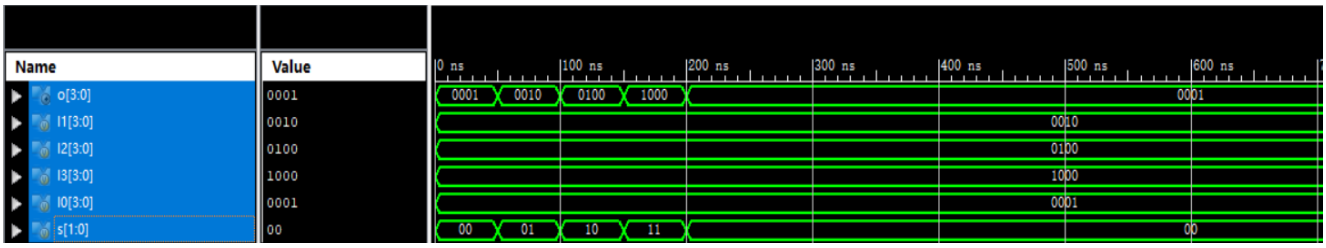




对 Mux4to14b 模块进行仿真，激励代码如下（initial 主要代码）：

```
initial begin
        s = 0;
        I1 = 0;
        I2 = 0;
        I3 = 0;
        I0 = 0;

        I0[0] = 1;
        I1[1] = 1;
        I2[2] = 1;
        I3[3] = 1;
        #50;

       s[0] = 1; #50;
        s[0] = 0; s[1] = 1; #50;
        s[0] = 1; #50;
        s[0] = 0;
        s[1] = 0;
        end
endmodule
```

仿真结果如下：



在模拟仿真中，当输入 s 为 00 时，只有 O[0]为 1，当输入 s 为 01 时，只有 O[1]为 1，当输入 s 为 10 时，只有 O[2]为 1，当输入 s 为 11 时，只有 O[3]位 1，符合预期结果，成功实现了 Mux4tob4 模块的功能。

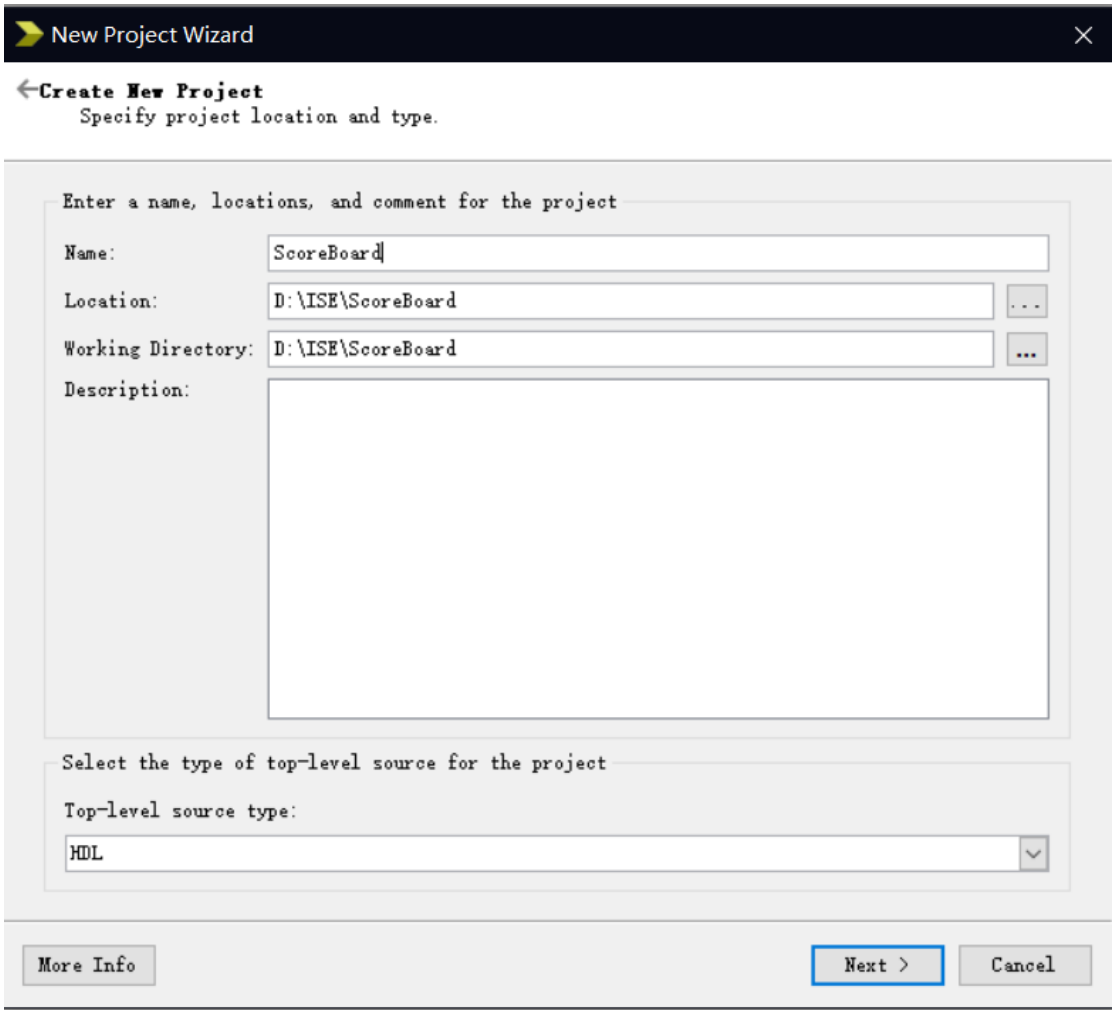
1.2.2.3 生成.sym 文件

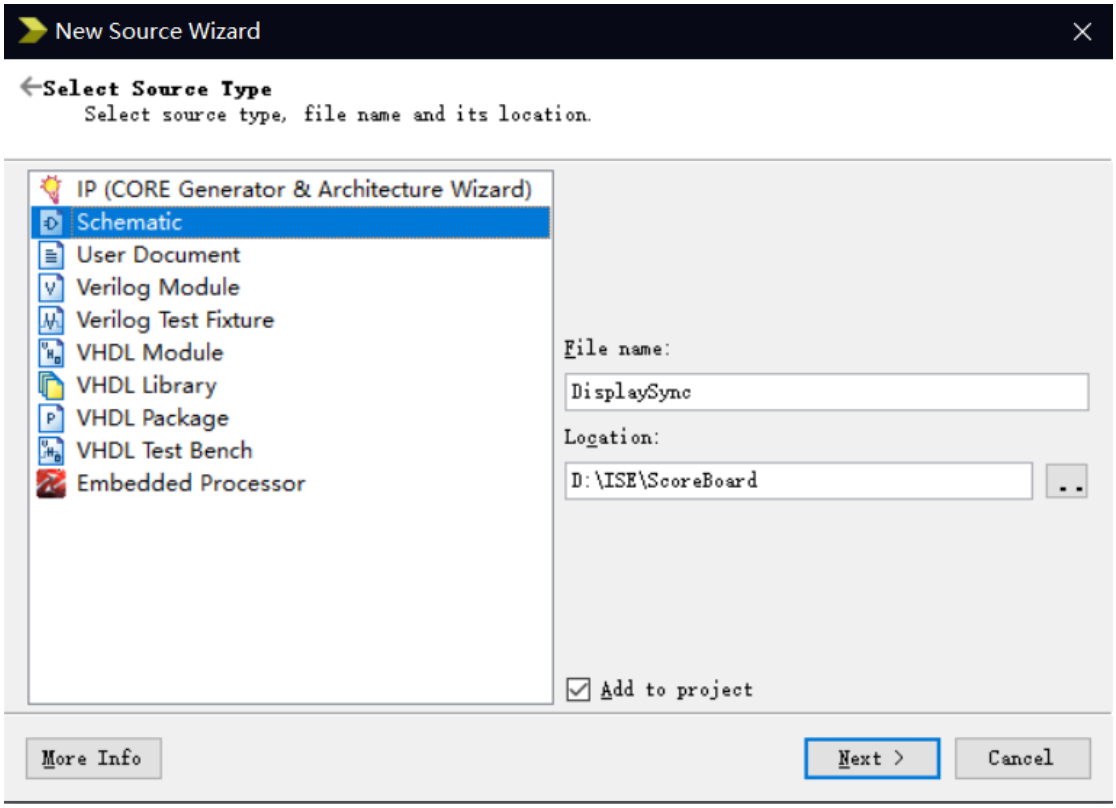
双击 Processes 窗口下的 Create Schematic Symbol 按钮，生成相关的.sym 文件。

1.3 任务二——计分板应用设计

1.3.1DisplaySync 元件的设计

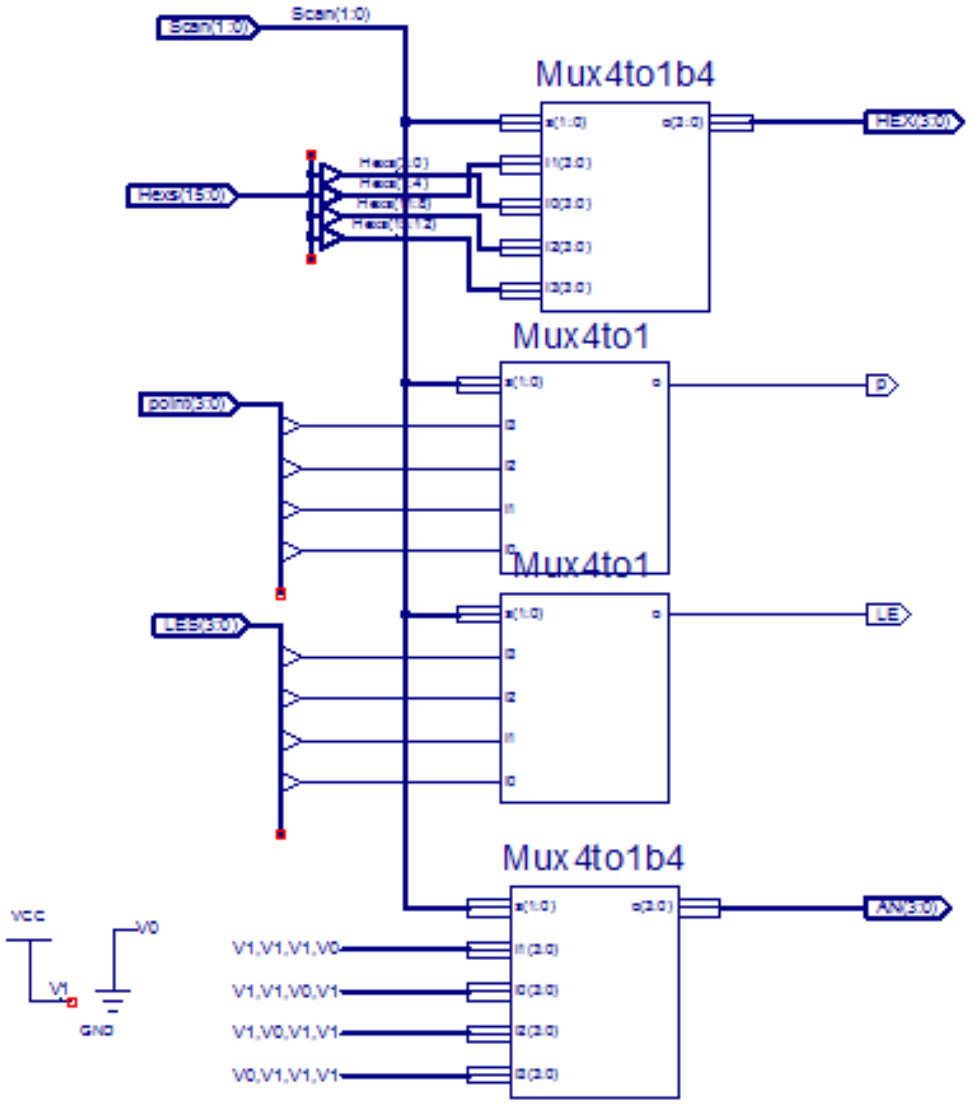
1.新建工程，工程名称为 ScoreBoard。并且新建一个文件名为 DisplaySync 的 Schematic 文件。



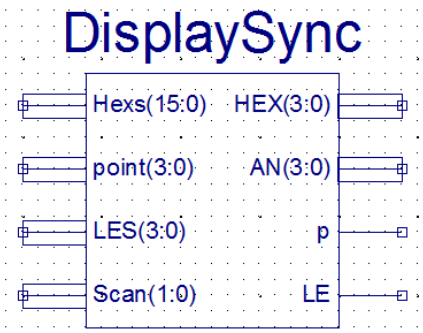


2.把之前生成的 Mux4to1.sym 和 Mux4to1.vf 文件复制到当前的工程目录下。

3.按照原理进行连线。



4.连好逻辑图后，双击 Processes 窗口下的 Check Design Rules 按钮，确认无误后，生成相应的.sym 文件，以供后续操作使用。



1.3.2 clkdiv 原件的设计

1.新建工程，命名为 clkdiv，新建命名为 clkdiv 的 Verilog Module 文件，输入时钟计数分频器的相关代码作为计分板的辅助模块。

相关代码如下：

```
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
```

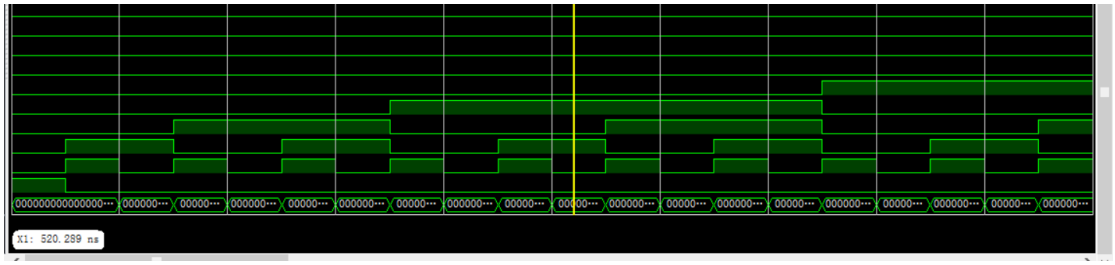
2.仿真模拟

代码如下：

```
module sim;

    // Inputs
    reg clk;
    reg rst;

    // Outputs
    wire [31:0] clkdiv;

    // Instantiate the Unit Under Test (UUT)
    clkdiv uut (
        .clk(clk),
        .rst(rst),
        .clkdiv(clkdiv)
    );

    integer i = 0;
    initial begin
        // Initialize Inputs
        clk = 0;
        rst = 1;
        #50;

        // Wait 100 ns for global reset to finish
        for (rst = 0; i < 128; i = i+1) begin
            clk = clk + 1;
            #50;
        end
```

```
        rst = 1;
    end

endmodule
```

仿真波形图如下：

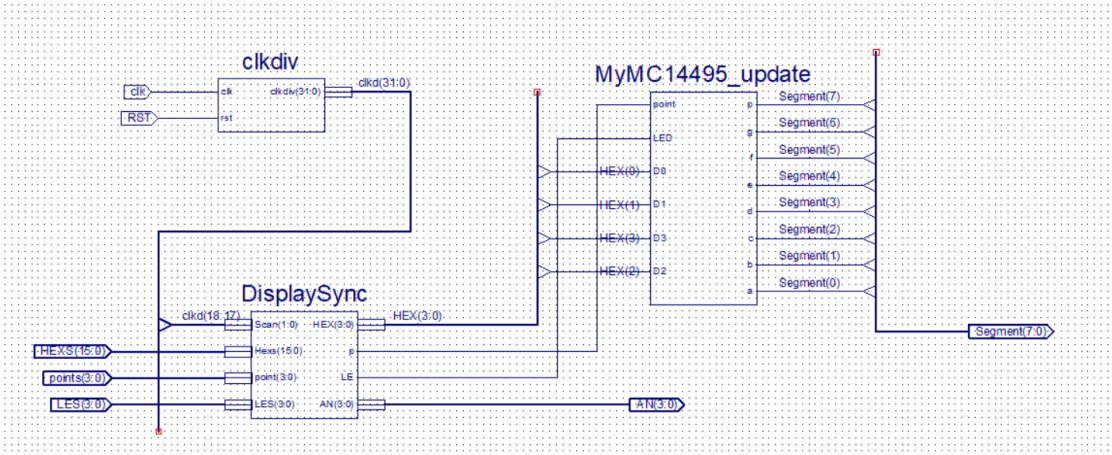


1.3.3 设计 disp_num 模块

1.按要求新建一个工程，命名为 disp_num,并且在该工程目录下新建一个命名为 disp_num 的 Schematic 类型的文件。

2.把 DisplaySync 原件和实验六设计的 MyMC14495 原件所涉及到的相关.sym .vf 文件复制到新建的工程目录下，同时把 clkdiv 原件的.sym .v 文件 y 也复制到该新建的工程目录下。

3.按照原理进行原件 disp_num 的连线。



4.新建 Verilog Module 文件，并且命名为 top。作为本实验的顶层模块的代码。

相关代码如下：

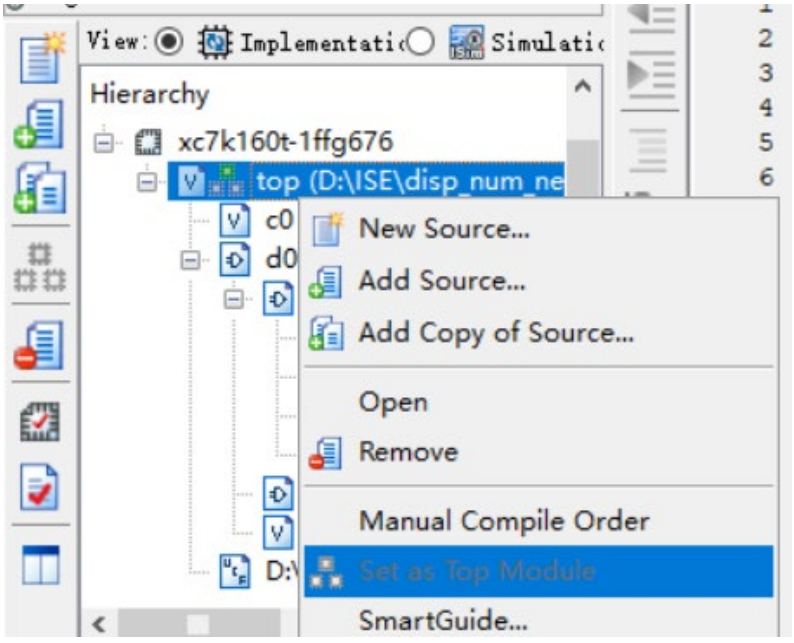
```
module top(
    input wire clk,
    input wire [7:0] SW,
    input wire [3:0] btn,
    output wire [3:0] AN,
    output wire [7:0] Segment
);
    wire [15:0] num;

    CreateNumber c0(btn, num);

    disp_num_new d0(clk, num, SW[7:4], SW[3:0], 1'b0, AN, Segment);
```

```
endmodule
```

CreateNumber 模块的相关代码如下：

```
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;

    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
```

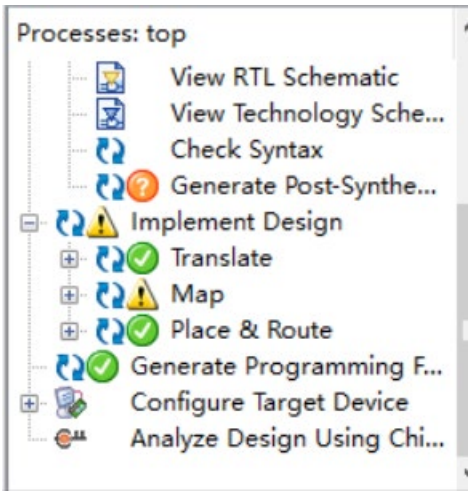
5.将鼠标移动至 top 文件，右键弹出窗口，点击 Set as Top Module,做为本实验的顶层模块。



6.进行引脚的分配。
本实验的引脚分配如下：

```
NET"clk"LOC=AC18 | IOSTANDARD=LVCMOS18;

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

NET"btn[0]"LOC=AF8 | IOSTANDARD=LVCMOS15;
NET"btn[0]"clock_dedicated_route=false;
NET"btn[1]"LOC=AE13 | IOSTANDARD=LVCMOS15;
NET"btn[1]"clock_dedicated_route=false;
NET"btn[2]"LOC=AF13 | IOSTANDARD=LVCMOS15;
NET"btn[2]"clock_dedicated_route=false;
NET"btn[3]"LOC=AF10 | IOSTANDARD=LVCMOS15;
NET"btn[3]"clock_dedicated_route=false;

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;
```

7.配置完毕，双击 Processes 窗口下的 Generate Programming File，成功后即可在数字逻辑实验板上进行操作。



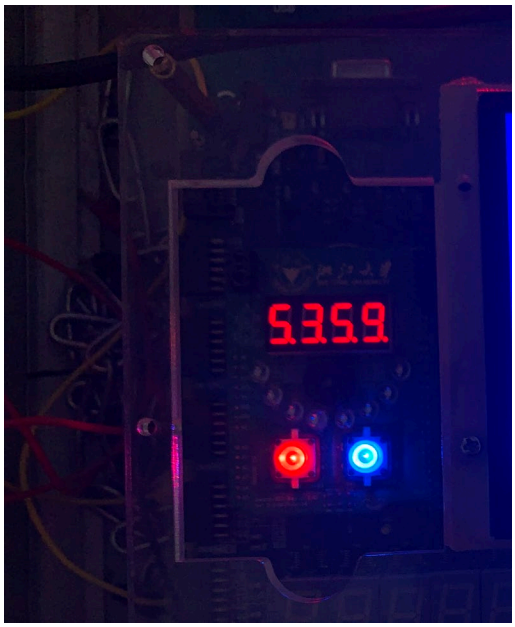
注意：本实验由于部分元件的部分引脚没有得到使用等等诸多系统自身存在的问题，在编译

运行的过程中会产生 Warnings，这是正常的现象，不影响后续生成可执行文件，不影响后续的实际操作。

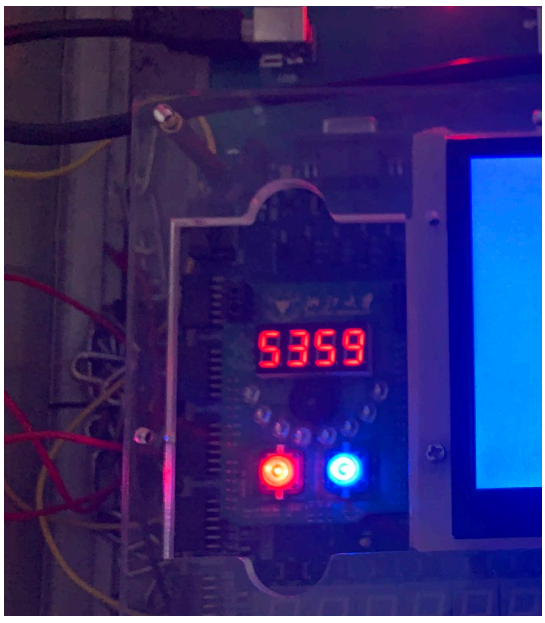
二、实验结果与分析

2.1 实验结果

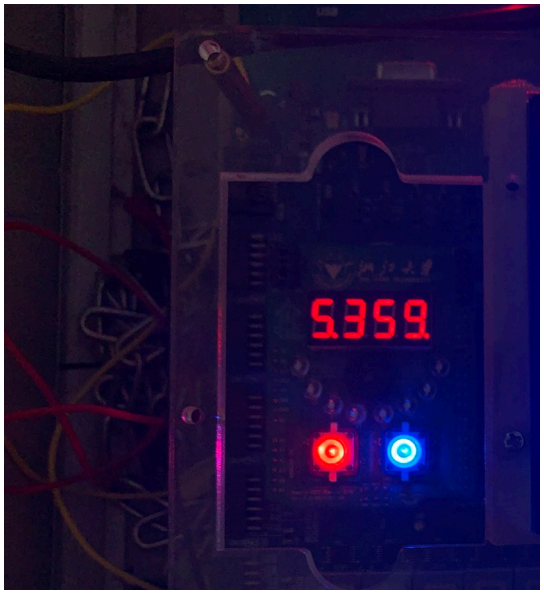
通过开关，可以在四位七段数码管显示数字的基础上，每一位显示不同的数字，如下图所示：



显示数码管的开关都打开，显示小数点的使能开关打开，通过调节各自的递增开关调出数字5359.

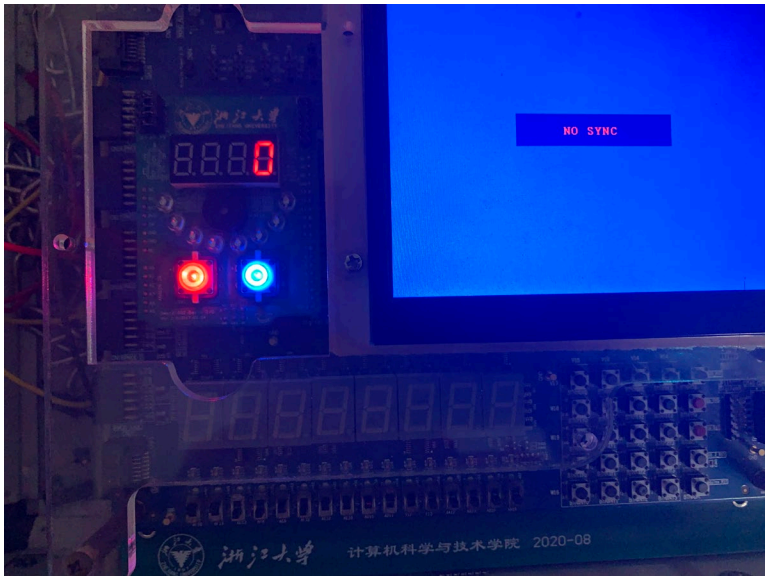


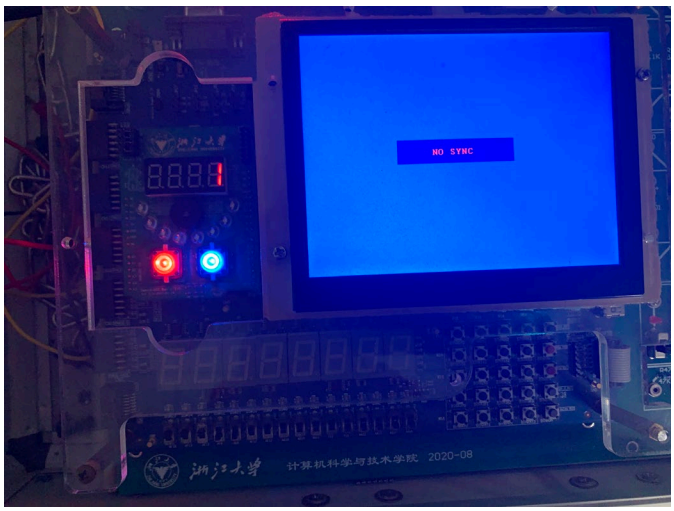
在前面的基础上，关闭显示小数点的使能开关。

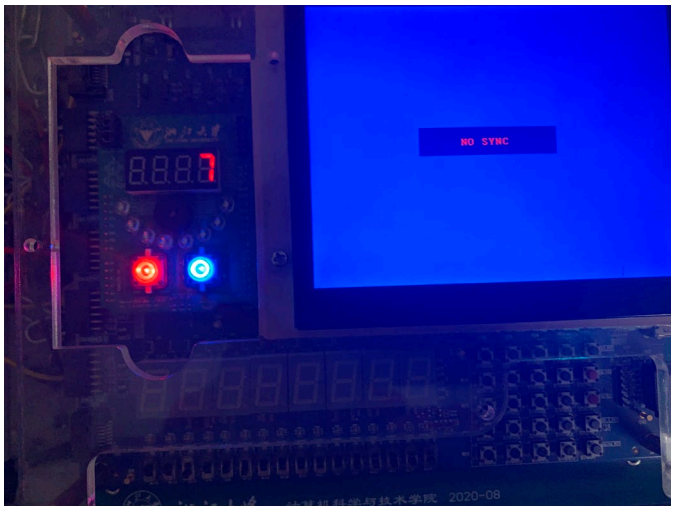


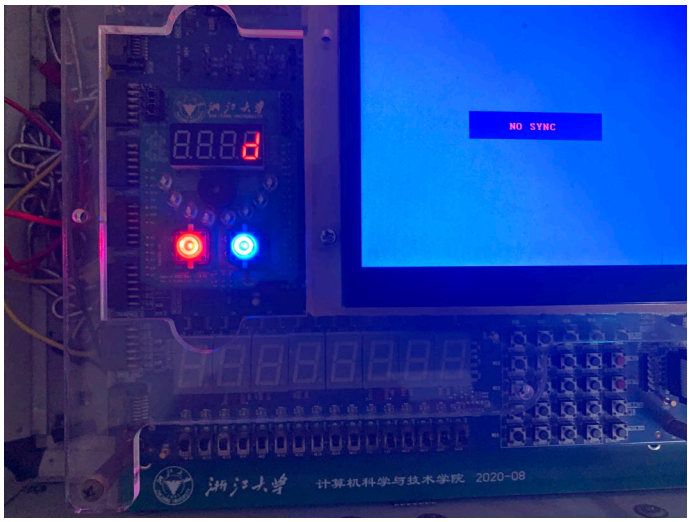
选择性地关闭小数点的使能开关，这表明小数点的使能开关可以独立控制四个数码管的小数点的亮与暗。

下面检验数字递增的实现是否正常：









经检验，开关实现数字递增的功能是正常的。

2.2 实验分析

通过实验结果的验证分析，成功实现了计数板的基本功能，即实现了用动态扫描来实现四位七段数码管显示不同的数字，并且能够利用开关使得数字逻辑实验板上能够显示任意的数字。同时，可以通过开关控制四个数码管各自的亮与暗、小数点各自的亮与暗。

三、讨论、心得

本次实验是在实验六的基础上，从之前的四位数码管只能同时显示一个数字，到通过动态扫描手段实现了四个数码管上可以显示不同的数字，故本次实验相比之前要复杂了不少。由于我是采用画图的方式进行本次实验的，所以画图的工作量是较大的。实验中，我的第一次实验到了 generating programing file 这一步，由于未知的错误出现了问题。通过排查，我认为是因为实验中的某一个模块在连接的时候出现了问题所导致的。我把该模块做了重新的连线以及实例化，最终通过了实验。在排查错误的过程中，我对于实验的原理，元件的功能作用以及输入、输出的引脚有了更加深刻的认识。

另外，本实验也是第一次采用 Verilog Module 文件而不是 Schematic 原理图设计实现顶层模块，这使得我对于 Verilog 语言的语法也有了更进一步的了解。而 Verilog Module 模块与 Schematic 模块的相互连接，也让我对于两者的关系有了更加清晰的认识。

本实验有一个值得注意的地方，就是 top 文件中引用到 disp_num 模块，命名为 d0.

```
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_new d0(clk, num, SW[7:4], SW[3:0], 1'b0, AN, Segment);
endmodule
```

而其中的变量(clk, num, SW[7:4], SW[3:0], 1'b0, AN, Segment)是有顺序的，应该与自己生成的 dip_num 模块工程目录下的.vf 文件中相应变量的顺序对应。之后的模块引用参变量，也应该十分注意顺序这一个问题。