Lab1-report

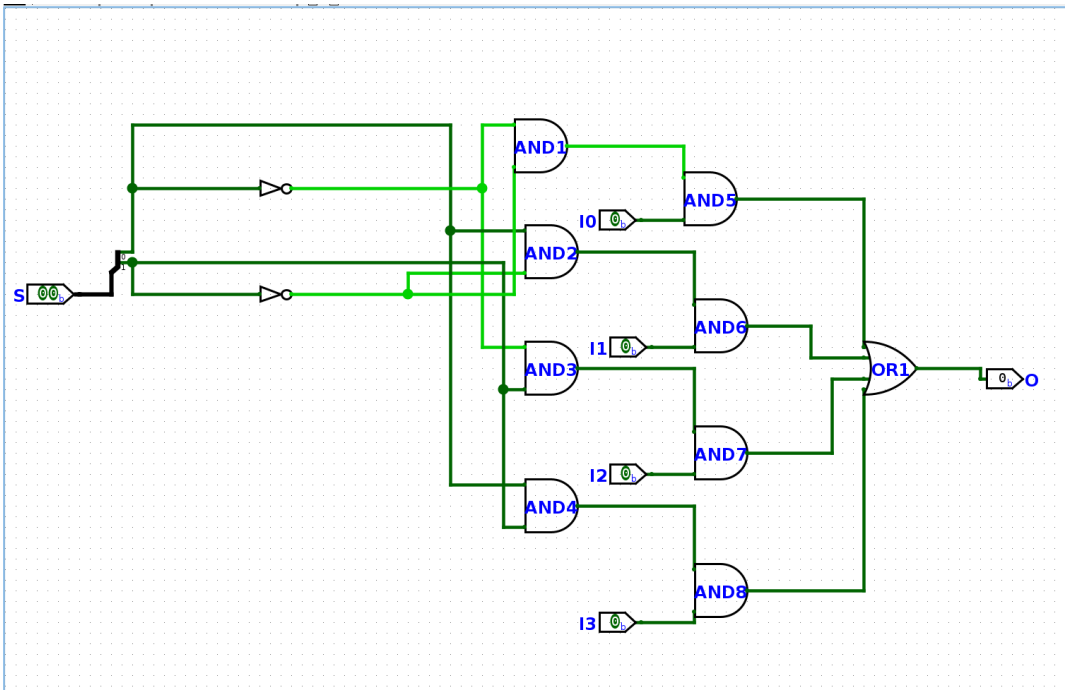
Lab1-1

实验目的

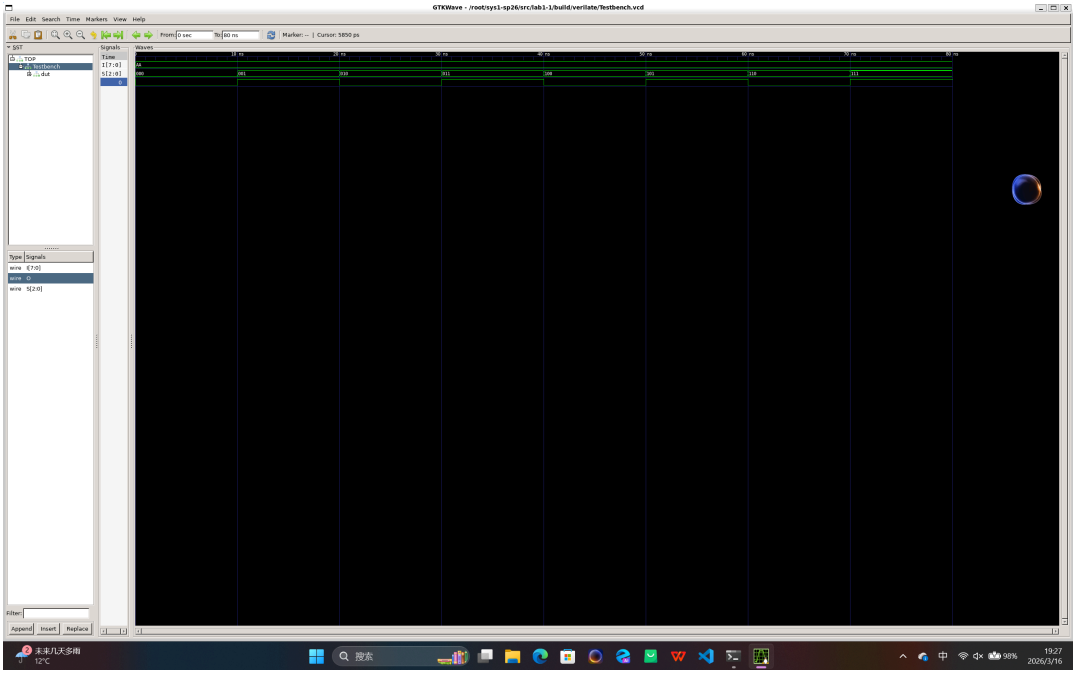
- 掌握用于设计和仿真数字逻辑电路的 Logisim 的基本使用方法，并使用其进行电路设计
- 掌握多路复用器和加法器的内部结构和功能
- 利用原理图所产生的 Verilog 代码进行学习

实验过程

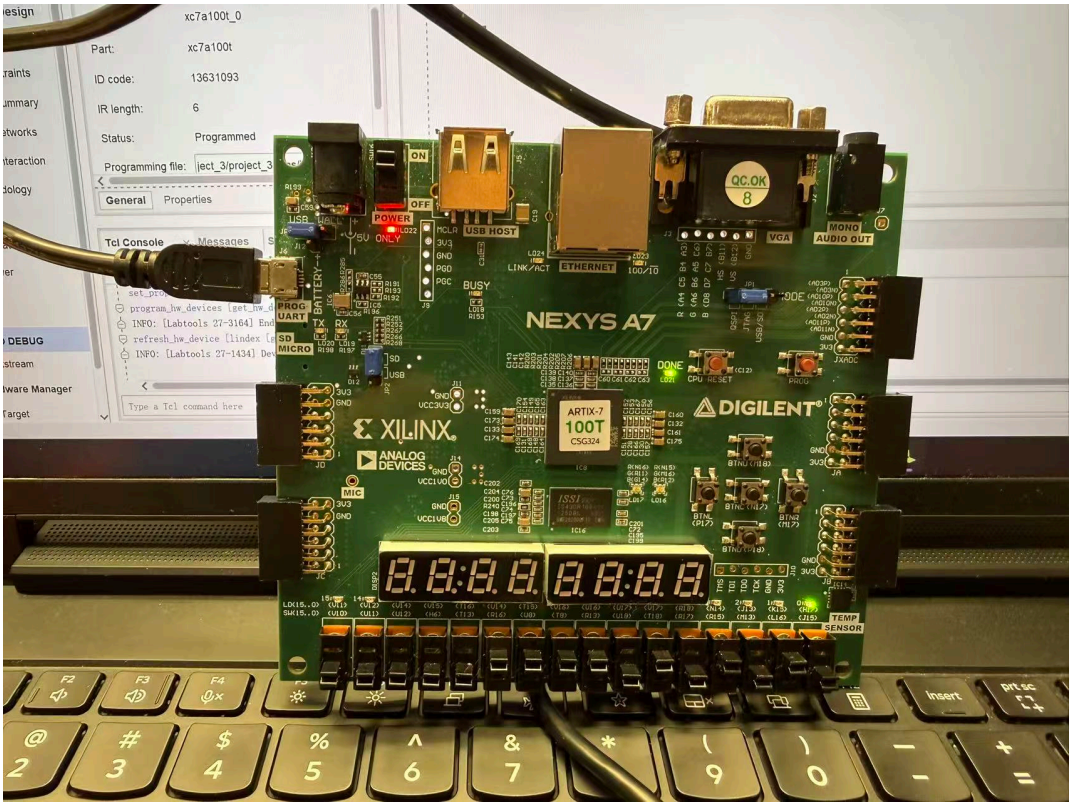
- Debug:我在编写Mux8T1_1.v文件的时候，误以为其模块的名称和c语言的函数名称是一个意思，但其实不然，verilog语言中的名称仅仅指的是模块的名称，其用法和定义方面存在着很大的区别。
- 绘制原理图



- verilate仿真模拟

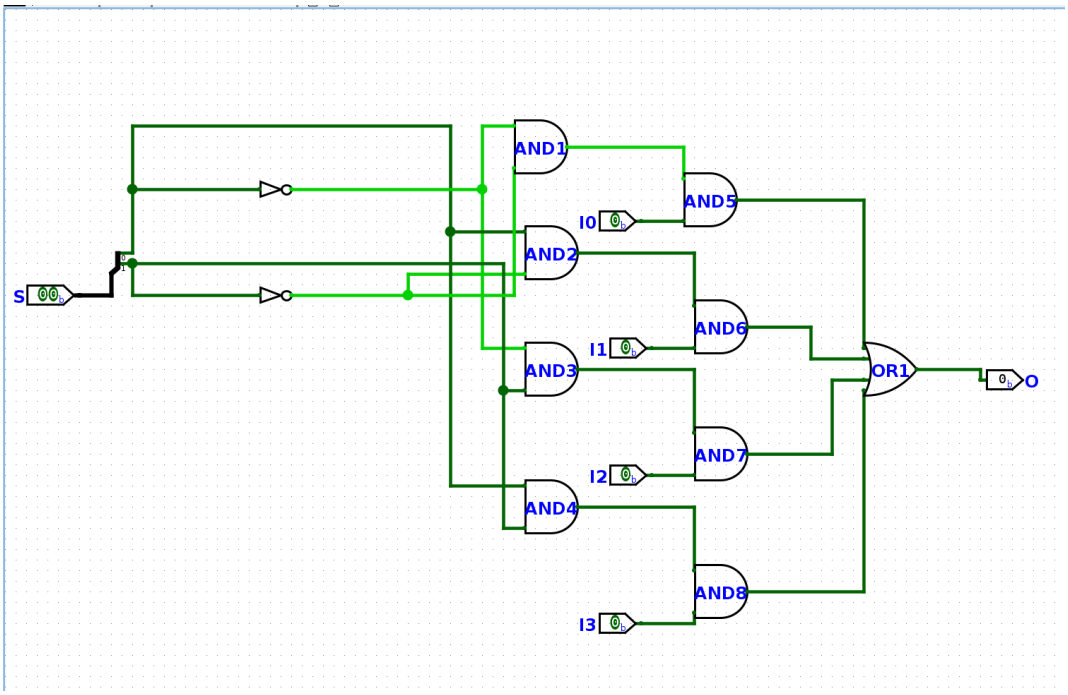


- vivado上板成功



实验结果

四路选择器原理图 3%



- 采用了logisim的软件进行绘制，其中采用了splitter来进行对s的输入元素（2bits）进行分流。

八路选择器代码解释 5%

main.v(Mux8T1_1)

```
module main( //定义声明main模块的输入和输出
    input [7:0] I,  //I[0]~I[7]的向量
    input [2:0] S,  //S[2]~S[0]的向量
    output O
);
    wire net1;
    wire net2;
    Mux4T1_1 gate1(  //定义Mux4T1_1模块的连接
        .I0(I[0]),
        .I1(I[1]),
        .I2(I[2]),
        .I3(I[3]),
        .O(net1),
        .S(S[1:0])

    );
    Mux4T1_1 gate2(//定义Mux4T1_1模块的连接
        .I0(I[4]),
        .I1(I[5]),
        .I2(I[6]),
        .I3(I[7]),
        .O(net2),
```

```
        .S(S[1:0])  //其中S是一个向量，所以要用S[1:0]进行定义

    );
    assign O=S[2]?net1:net2;//采用类似c语言的？三目运算符逻辑表示线路的连接

endmodule
```

Mux4T1_1.v

```
/*********************************************************************************
*
 ** Logisim-evolution goes FPGA automatic generated Verilog code
**
 ** https://github.com/logisim-evolution/
**
 **
**
 ** Component : main
**
 **
**

*********************************************************************************/

module Mux4T1_1( I0,
             I1,
             I2,
             I3,
             O,
             S );


/*********************************************************************************
**
   ** The inputs are defined here
**

*********************************************************************************
*/
   input       I0;
   input       I1;
   input       I2;
   input       I3;
   input [1:0] S;
```

```
/*****************************************************************************
**
   ** The outputs are defined here
**

*****************************************************************************
*/
   output O;


/*****************************************************************************
**
   ** The wires are defined here
**

*****************************************************************************
*/
   wire [1:0] s_logisimBus13;
   wire       s_logisimNet0;
   wire       s_logisimNet1;
   wire       s_logisimNet10;
   wire       s_logisimNet11;
   wire       s_logisimNet12;
   wire       s_logisimNet14;
   wire       s_logisimNet15;
   wire       s_logisimNet16;
   wire       s_logisimNet17;
   wire       s_logisimNet2;
   wire       s_logisimNet3;
   wire       s_logisimNet4;
   wire       s_logisimNet5;
   wire       s_logisimNet6;
   wire       s_logisimNet7;
   wire       s_logisimNet8;
   wire       s_logisimNet9;


/*****************************************************************************
**
   ** The module functionality is described here
**

*****************************************************************************
*/
```

```
/*********************************************************************
**
   ** Here all input connections are defined
**

*********************************************************************
*/
   assign s_logisimBus13[1:0] = S;
   assign s_logisimNet0       = I3;
   assign s_logisimNet15      = I0;
   assign s_logisimNet4       = I1;
   assign s_logisimNet5       = I2;


/*********************************************************************
**
   ** Here all output connections are defined
**

*********************************************************************
*/
   assign O = s_logisimNet1;


/*********************************************************************
**
   ** Here all in-lined components are defined
**

*********************************************************************
*/

   // NOT Gate
   assign s_logisimNet3 = ~s_logisimBus13[0];

   // NOT Gate
   assign s_logisimNet8 = ~s_logisimBus13[1];


/*********************************************************************
**
   ** Here all normal components are defined
**

*********************************************************************
*/
```

```
    AND_GATE #(.BubblesMask(2'b00))
       AND1 (.input1(s_logisimNet3),
             .input2(s_logisimNet8),
             .result(s_logisimNet16));

    AND_GATE #(.BubblesMask(2'b00))
       AND2 (.input1(s_logisimBus13[0]),
             .input2(s_logisimNet8),
             .result(s_logisimNet10));

    AND_GATE #(.BubblesMask(2'b00))
       AND3 (.input1(s_logisimNet3),
             .input2(s_logisimBus13[1]),
             .result(s_logisimNet17));

    AND_GATE #(.BubblesMask(2'b00))
       AND4 (.input1(s_logisimBus13[0]),
             .input2(s_logisimBus13[1]),
             .result(s_logisimNet11));

    AND_GATE #(.BubblesMask(2'b00))
       AND5 (.input1(s_logisimNet16),
             .input2(s_logisimNet15),
             .result(s_logisimNet14));

    AND_GATE #(.BubblesMask(2'b00))
       AND6 (.input1(s_logisimNet10),
             .input2(s_logisimNet4),
             .result(s_logisimNet6));

    AND_GATE #(.BubblesMask(2'b00))
       AND7 (.input1(s_logisimNet17),
             .input2(s_logisimNet5),
             .result(s_logisimNet2));

    AND_GATE #(.BubblesMask(2'b00))
       AND8 (.input1(s_logisimNet11),
             .input2(s_logisimNet0),
             .result(s_logisimNet9));

    OR_GATE_4_INPUTS #(.BubblesMask(4'h0))
       OR1 (.input1(s_logisimNet14),
            .input2(s_logisimNet6),
            .input3(s_logisimNet2),
            .input4(s_logisimNet9),
            .result(s_logisimNet1));
```

```
endmodule
```

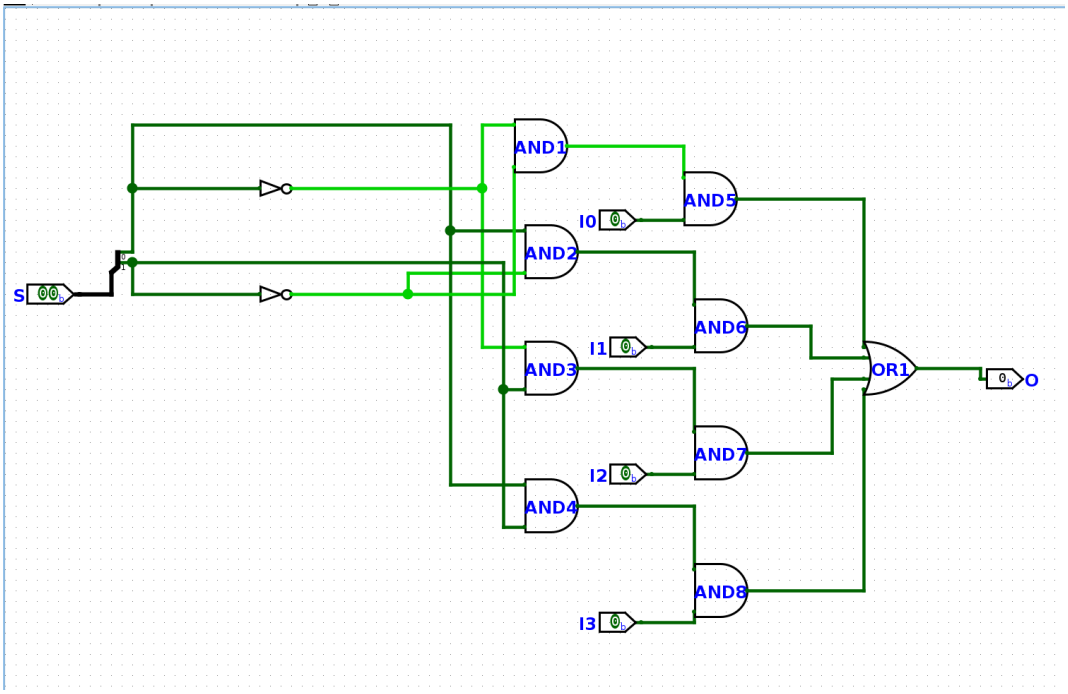
TestBench 设计 5%

```
module Testbench;

    reg [7:0] I;
    reg [2:0] S;
    wire O;

    initial begin
    I=8'b10101010; //我先随便定义了I向量接口的输入，这样可以使得O的波形变得好看一点（bushi）
    S=3'b000; #10
    S=3'b001; #10
    S=3'b010; #10
    S=3'b011; #10
    S=3'b100; #10// 遍历
    S=3'b101; #10
    S=3'b110; #10
    S=3'b111; #10
    $finish;
    end

    Mux8T1_1 dut(  //说明Mux8T1_1模块的接口（我上文的main.v文件将其改成了main模块仅仅只是为了适应vivado的软件，再跑verilate仿真的时候是运用了Mux8T1_1作为模块名
        .I(I),
        .S(S),
        .O(O)
    );

    `ifdef VERILATE
        initial begin
            $dumpfile({`TOP_DIR,"/Testbench.vcd"});
            $dumpvars(0,dut);
            $dumpon;
        end
    `endif

endmodule
```

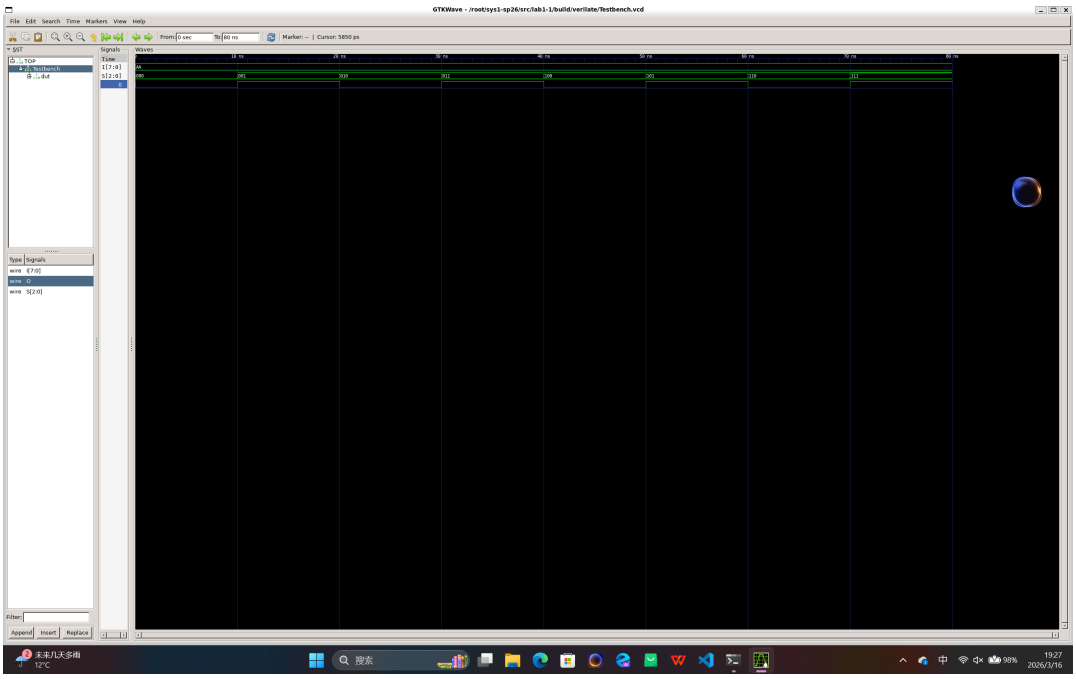
其余实验步骤及截图 10%

其实和上面的实验过程有一部分重复了

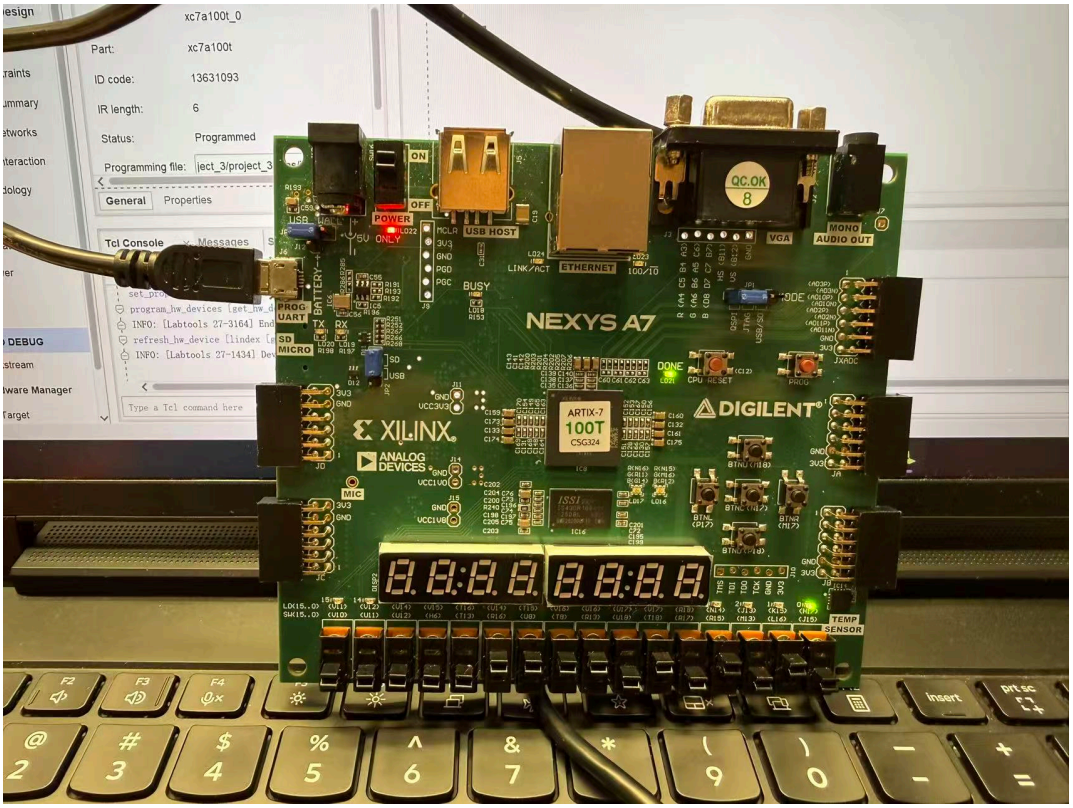
- 绘制原理图



- verilate仿真模拟



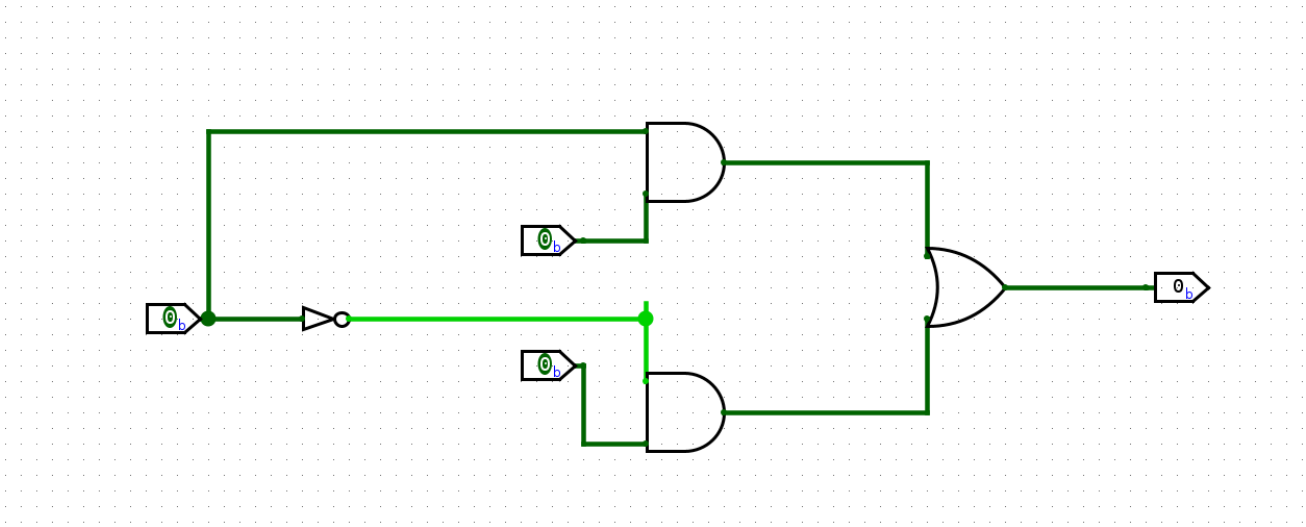
- vivado上板成功



思考题

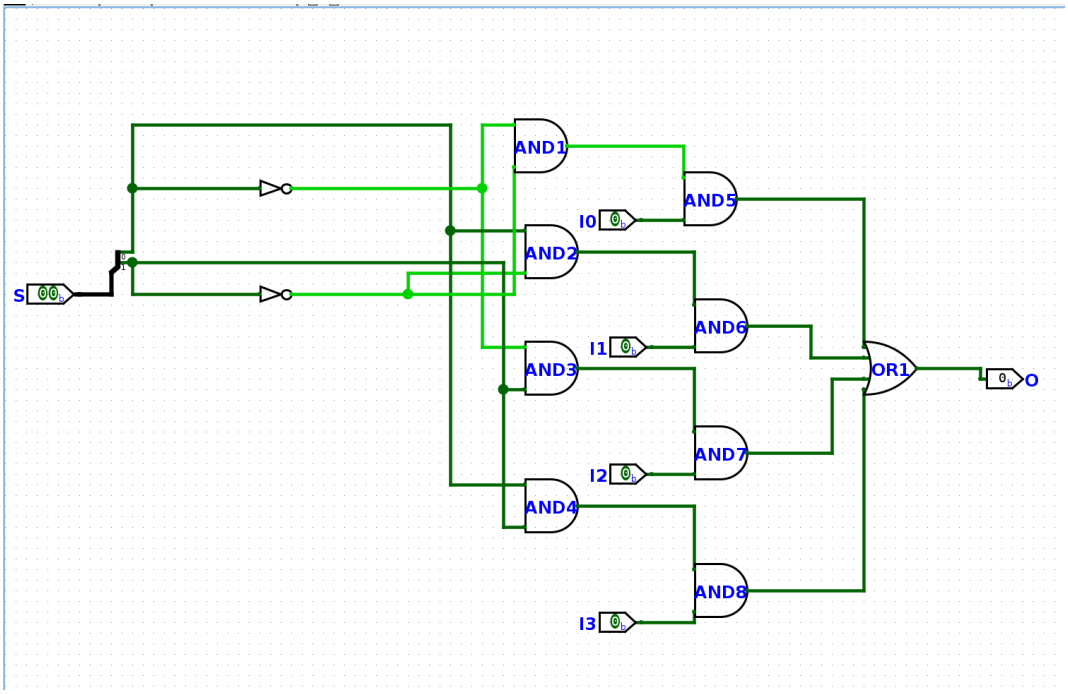
Mux2T1_1 是两个 AND，一个 OR 和一个 NOT 组成的简单结构，它是由哪种 decoder 和 AND-OR 结构组成的 5%

- decoder：NOT门
- AND-OR：两个AND；一个OR
  具体结构如下



- 左边部分即为decode部分，右边为AND-OR结构

Mux4T1_1 是如何组成的 6%



- 是由一个splitter，四个与门，一个四输入或门组成。
- 其中输入信号是两位bit，通过splitter来将其信号分到两根线中。
- 用两个非门来将信号传递给AND中
- AND5-8来判断是否要输出I0-I3之中的其中一个信号
- 之后再通过OR来输出到O

Mux8T1_1 是如何组成的 6%

- Mux8T1_1的输入信号首先通过splitter被分为两组信号，每组的四个输入被接到Mux4T1_1中。两组Mux4T1_1的输出再接一个或门输出。
- 总输入是要三个bit(e.g. 3'b001)。有一个来用于控制是否给两个Mux4T1_1通电。

那么 Mux2 m T1_n 是如何构成的呢 10%

- Mux2^mT1_1的输入信号首先通过splitter被分为$2^(m-1)$组信号，每组的四个输入被接到Mux $2^(m-1)$T1_1中。两组Mux$2^(m-1)$T1_1的输出再接一个或门输出。
- 总输入是要(m-1)个bit(e.g. (m-1)'b00000……01)。有一个来用于控制是否给两个Mux $2^(m-1)$T1_1通电。
- 通过递推就可以推知所有2的幂次的模块。

心得体会

Lab1-2

实验目的

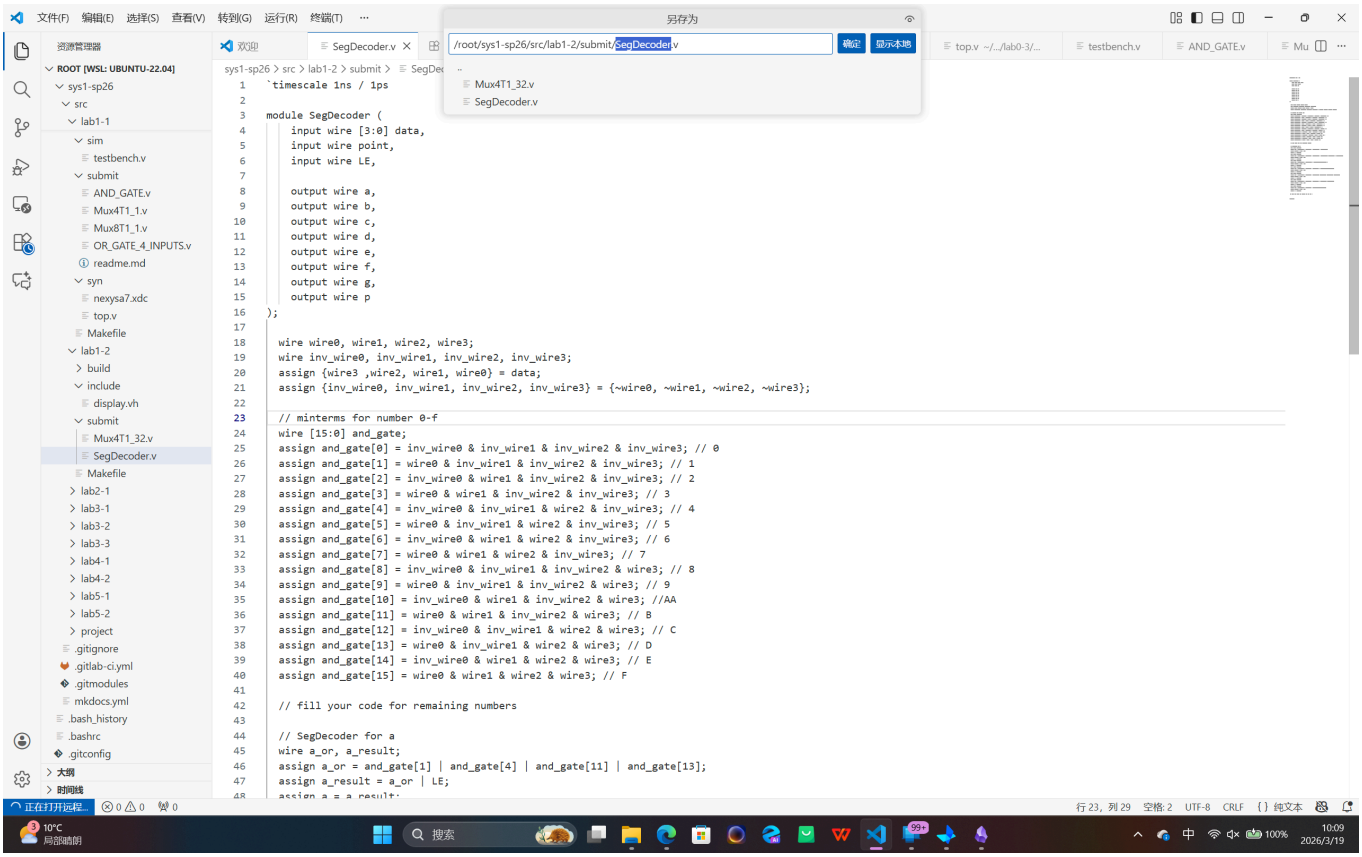
- 掌握使用 Verilog 语言进行电路设计的方法
- 掌握七段数码管显示译码器 (MC14495) 的内部电路结构及其设计
- 实现 Nexys A7 开发板七段数码管十六进制数显示

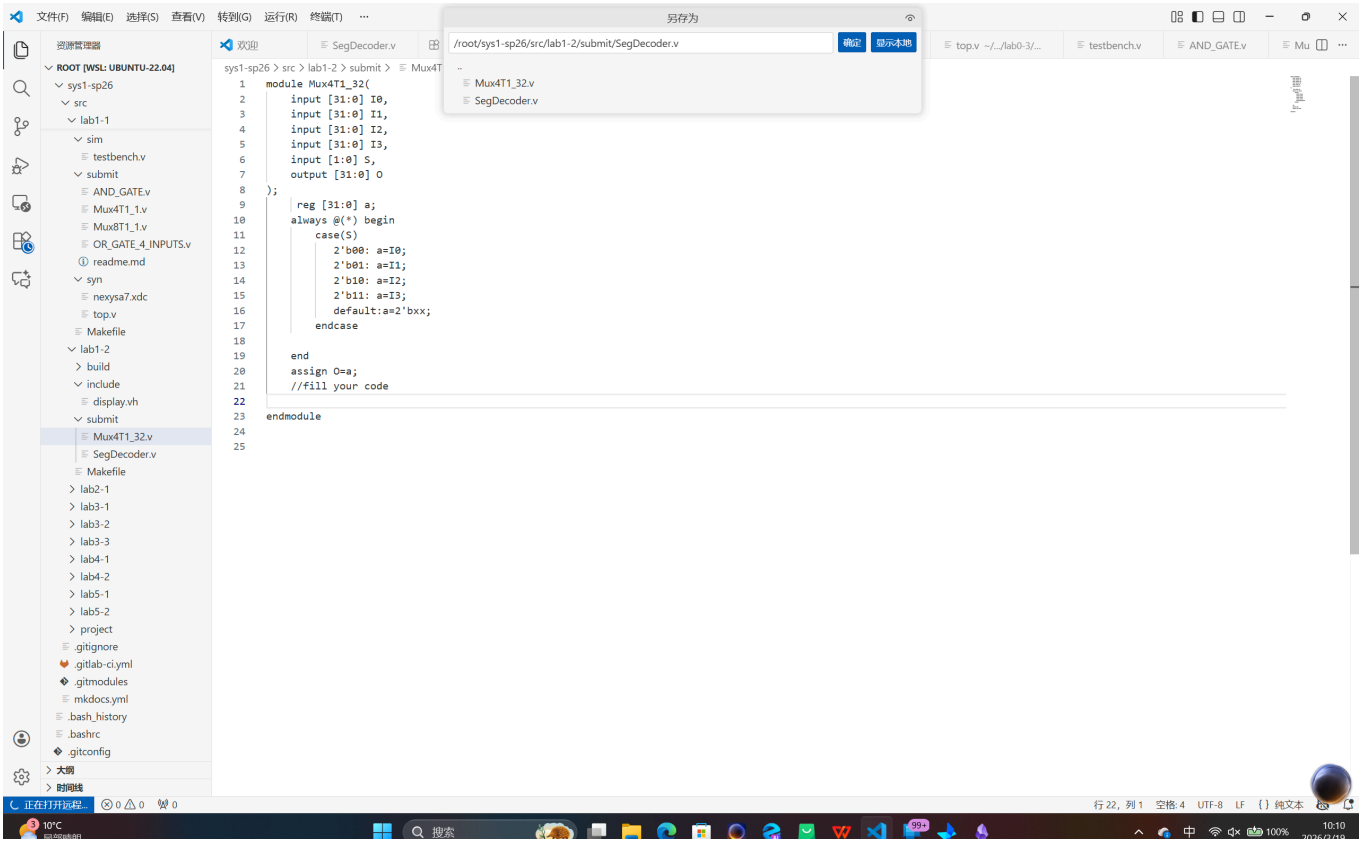
实验过程

debug

- 在使用always@语句的时候，本人发现不能用给定的wire&input来作为输出的结果。要引入一个reg寄存器，来储存其运算的输入和输出。

编写Mux4T1_32.v以及segdecoder.v文件



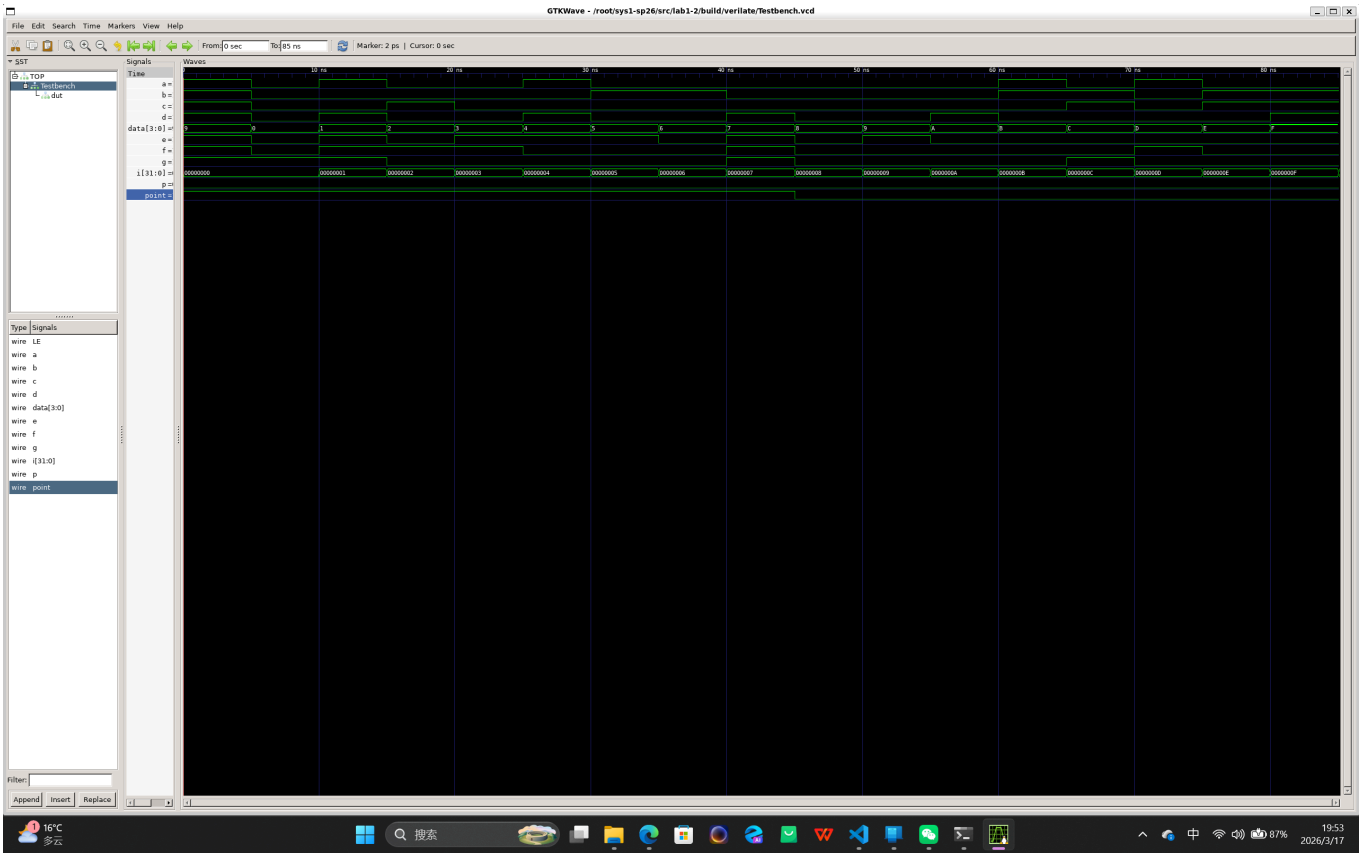


- 上面展示一下我编写程序的环境

实验结果

仿真设计

译码管的设计



- 这个仿真设计是基于补全segdecoder来进行仿真的，图中的数据是根据32位的数据i来进行遍历的。abcdef对应的是LED灯的低引脚，输出低电平的时候是亮灯的。
- 其中，编译码的代码如下：

```
`timescale 1ns / 1ps

module SegDecoder (
    input wire [3:0] data,
    input wire point,
    input wire LE,

    output wire a,
    output wire b,
    output wire c,
    output wire d,
    output wire e,
    output wire f,
    output wire g,
    output wire p
);
```

```
  wire wire0, wire1, wire2, wire3;
  wire inv_wire0, inv_wire1, inv_wire2, inv_wire3;
  assign {wire3 ,wire2, wire1, wire0} = data;
  assign {inv_wire0, inv_wire1, inv_wire2, inv_wire3} = {~wire0, ~wire1,
~wire2, ~wire3};

  // minterms for number 0-f
  wire [15:0] and_gate;
  assign and_gate[0] = inv_wire0 & inv_wire1 & inv_wire2 & inv_wire3; // 0
  assign and_gate[1] = wire0 & inv_wire1 & inv_wire2 & inv_wire3; // 1
  assign and_gate[2] = inv_wire0 & wire1 & inv_wire2 & inv_wire3; // 2
  assign and_gate[3] = wire0 & wire1 & inv_wire2 & inv_wire3; // 3
  assign and_gate[4] = inv_wire0 & inv_wire1 & wire2 & inv_wire3; // 4
  assign and_gate[5] = wire0 & inv_wire1 & wire2 & inv_wire3; // 5
  assign and_gate[6] = inv_wire0 & wire1 & wire2 & inv_wire3; // 6
  assign and_gate[7] = wire0 & wire1 & wire2 & inv_wire3; // 7
  assign and_gate[8] = inv_wire0 & inv_wire1 & inv_wire2 & wire3; // 8
  assign and_gate[9] = wire0 & inv_wire1 & inv_wire2 & wire3; // 9
  assign and_gate[10] = inv_wire0 & wire1 & inv_wire2 & wire3; //AA
  assign and_gate[11] = wire0 & wire1 & inv_wire2 & wire3; // B
  assign and_gate[12] = inv_wire0 & inv_wire1 & wire2 & wire3; // C
  assign and_gate[13] = wire0 & inv_wire1 & wire2 & wire3; // D
  assign and_gate[14] = inv_wire0 & wire1 & wire2 & wire3; // E
  assign and_gate[15] = wire0 & wire1 & wire2 & wire3; // F

  // fill your code for remaining numbers

  // SegDecoder for a
  wire a_or, a_result;
  assign a_or = and_gate[1] | and_gate[4] | and_gate[11] | and_gate[13];
  assign a_result = a_or | LE;
  assign a = a_result;
  wire b_or, b_result;
  assign b_or = and_gate[5] | and_gate[6] | and_gate[14] | and_gate[11]|
and_gate[12] | and_gate[15];
  assign b_result = b_or | LE;
  assign b = b_result;
  wire c_or, c_result;
  assign c_or = and_gate[2] | and_gate[12] | and_gate[15]|and_gate[14] ;
  assign c_result = c_or | LE;
  assign c = c_result;
  wire d_or, d_result;
  assign d_or = and_gate[1] | and_gate[4] | and_gate[7] |
and_gate[10]|and_gate[15];
  assign d_result = d_or | LE;
```

```
  assign d = d_result;
  wire e_or, e_result;
  assign e_or = and_gate[1] | and_gate[3] | and_gate[4]| and_gate[5]|
and_gate[7]| and_gate[9];
  assign e_result = e_or | LE;
  assign e = e_result;
  wire f_or, f_result;
  assign f_or = and_gate[1] | and_gate[2] | and_gate[3] | and_gate[7]|
and_gate[13];
  assign f_result = f_or | LE;
  assign f = f_result;
  wire g_or, g_result;
  assign g_or = and_gate[1] | and_gate[7] | and_gate[0]|and_gate[12];
  assign g_result = g_or | LE;
  assign g = g_result;

 // fill your code for decoder of b-g and p


endmodule
```

- or门的输入端连接的是可能使对应的LED管接入低电平的所有情况，经过了遍历，就可以暴政输出所有的情况。

复合多路选择器及语法分析比较

- segdecoder.v的部分代码

```
 reg [31:0] a;

    always @(*) begin

        case(S)

           2'b00: a=I0;

           2'b01: a=I1;

           2'b10: a=I2;

           2'b11: a=I3;

           default:a=2'bxx;

        endcase
```

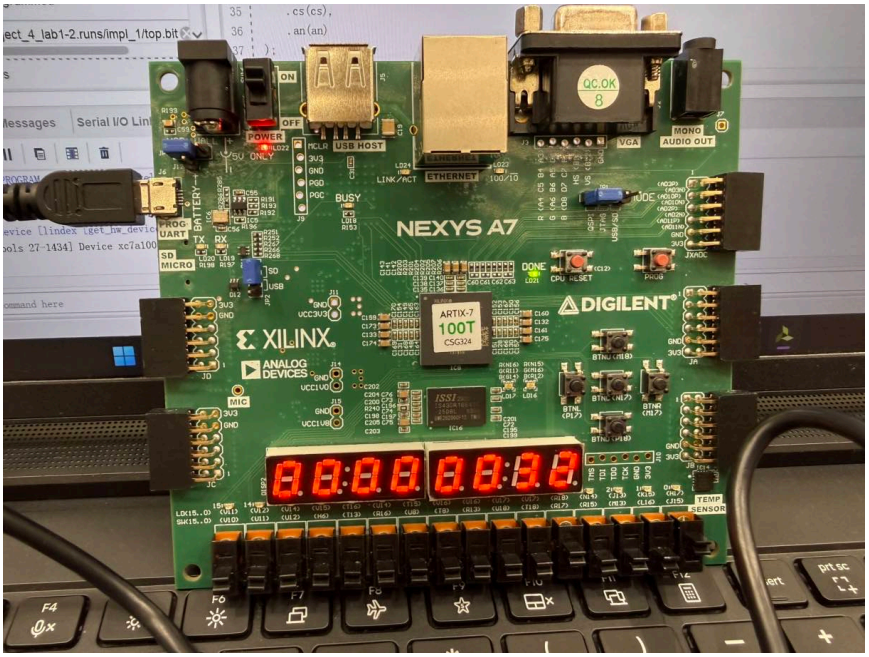
```
    end
```

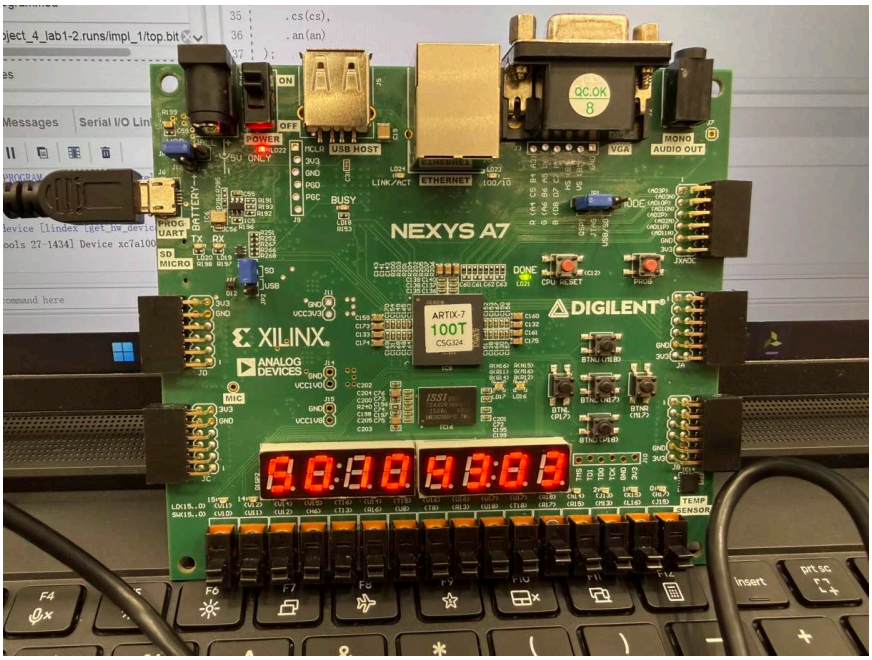
- 我这边采用的是与或的形式，在Mux4T_1.v的文件中，我采用了always+case的语句来遍历所有的S的情况。

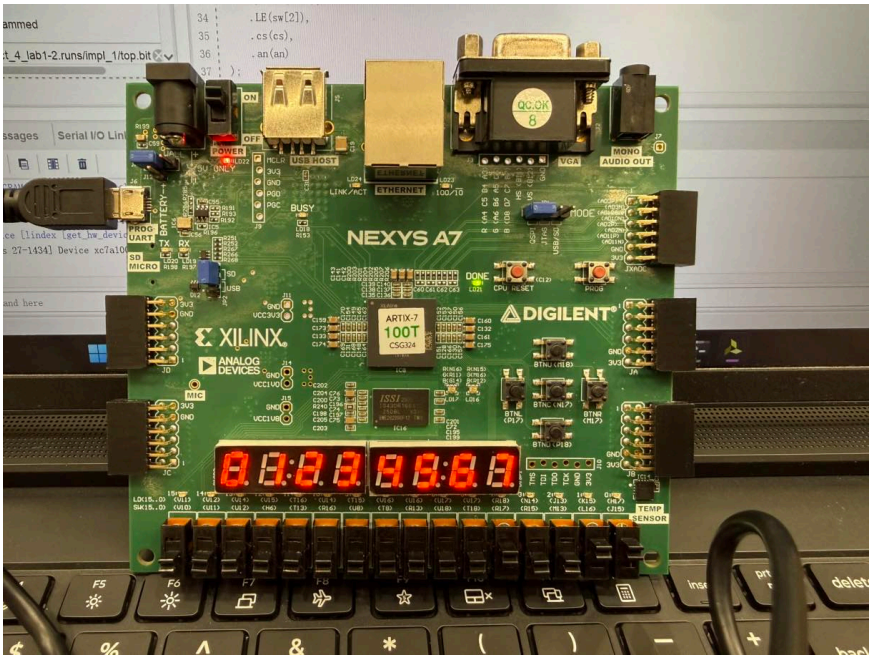
综合实现数码管

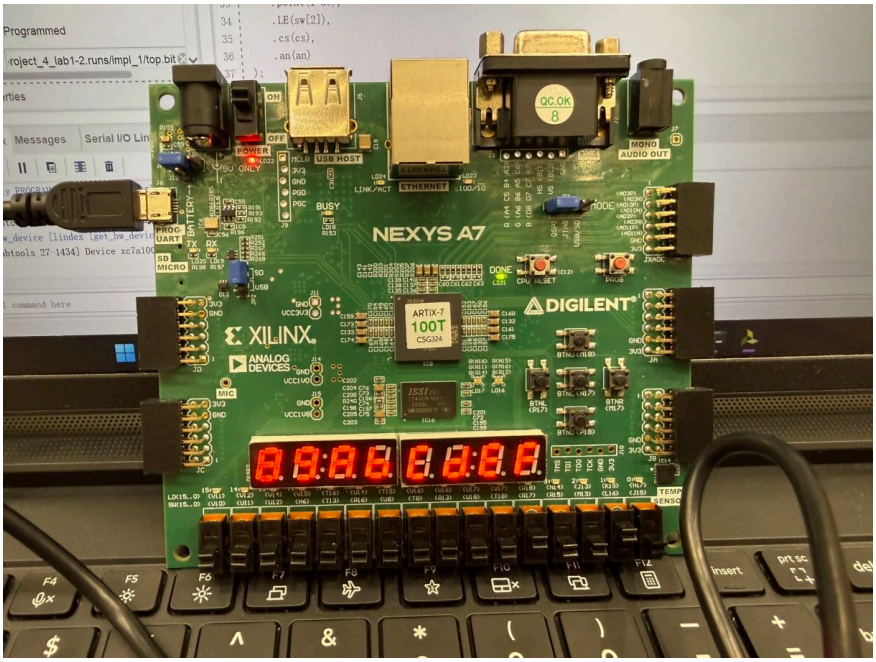
- 我们会给出一个display文件，里面展示了我们可以看到的四种情况的数字和字母的表达式。
- 代码如下

```
    `define DISPLAY 128'h01234567_89abcdef_00000032-50104303//replace xxxxxx
with your student ID
```


NEXYS A7
DIGILENT
XILINX
ARTIX-7
100T
CSG324
QC.OK
8


NEXYS A7
DIGILENT
XILINX
ARTIX-7
100T
CSG324
QC.OK
8





- 上图展示的是上板的过程

思考题

分析阅读 `repo/sys-project/lab1-2/sim/testbench.v` 的测试样例，尝试将 `for` 语句展开为初始化序列，然后写出你对 `for` 语句的理解

```
initial begin

        LE=1'b1;

        point=1'b1;
```

```
        data=4'h9;
        #5;
        LE=1'b0;
        for(i=0;i<8;i=i+1)begin
            data=i[3:0];
            #5;
        end
        point=1'b0;
        for(i=8;i<16;i=i+1)begin
            data=i[3:0];
            #5;
        end
        $finish;
    end
```

- 展开后的初始化序列是

```
data=4b'0000;
data=4b'0001;
data=4b'0010;
data=4b'0011;
data=4b'0100;
data=4b'0101;
data=4b'0110;
data=4b'0111;
data=4b'1000;
data=4b'1001;
data=4b'1010;
data=4b'1011;
data=4b'1100;
data=4b'1101;
```

```
data=4b'1110;
data=4b'1111;
```

- 我所理解的`for`循环语句是遍历了向量I里面的所有排列组合的形式。并从0000一直按照特定的遍历方式来到1111，具体的形式是从低到高来01遍历。

对于各种多路选择器的写法进行比较，请写出你最喜欢的多路选择器语法，并给出理由

- 我目前依据两个角度来说明我最喜欢的两种选择器

1. 简洁：我比较喜欢用向量形式来写多路选择器,如图，I是一个二维的wire，S向量是个2bit的输入`O=I[S]`表示I的第S（向量）的wire组和output进行连接。语言简单明了，符合计算机的逻辑语言。

```
module one-hot(
input [1:0] S,
output [3:0] O
);

wire [3:0] I [3:0];
assign I[0]=4'b0001;
assign I[1]=4'b0010;
assign I[2]=4'b0100;
assign I[3]=4'b1000;

assign O=I[S];

endmodule
```

2. 逻辑明确：我选择了与或的形式，这个形式的代码对于思考的要求较低，代码易懂，和数学的逻辑性强贴近，编写前期准备的工作较少。

```
assign O[0] = (~S[0]&~S[1])&I0[0] | (S[0]&~S[1])&I1[0] | (~S[0]&S[1])&I2[0] |
(S[0]&S[1])&I3[0];
assign O[1] = (~S[0]&~S[1])&I0[1] | (S[0]&~S[1])&I1[1] | (~S[0]&S[1])&I2[1] |
(S[0]&S[1])&I3[1];
assign O[2] = (~S[0]&~S[1])&I0[2] | (S[0]&~S[1])&I1[2] | (~S[0]&S[1])&I2[2] |
(S[0]&S[1])&I3[2];
```

```
 assign O[3] = (~S[0]&~S[1])&I0[3] | (S[0]&~S[1])&I1[3] | (~S[0]&S[1])&I2[3] |
(S[0]&S[1])&I3[3];
```