lab 1-1：八选一多路选择器

1. 实验环境

- EDA 工具：Ngspice，Logisim-evolution
- 操作系统：Windows 11 24H2，Ubuntu 24.04
- VHDL：Verilog

2. 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

3. 实验报告 50%

3.1 四路选择器原理图 3%

> 与文档略有不同，把两个二路与门合成一个三路与门。

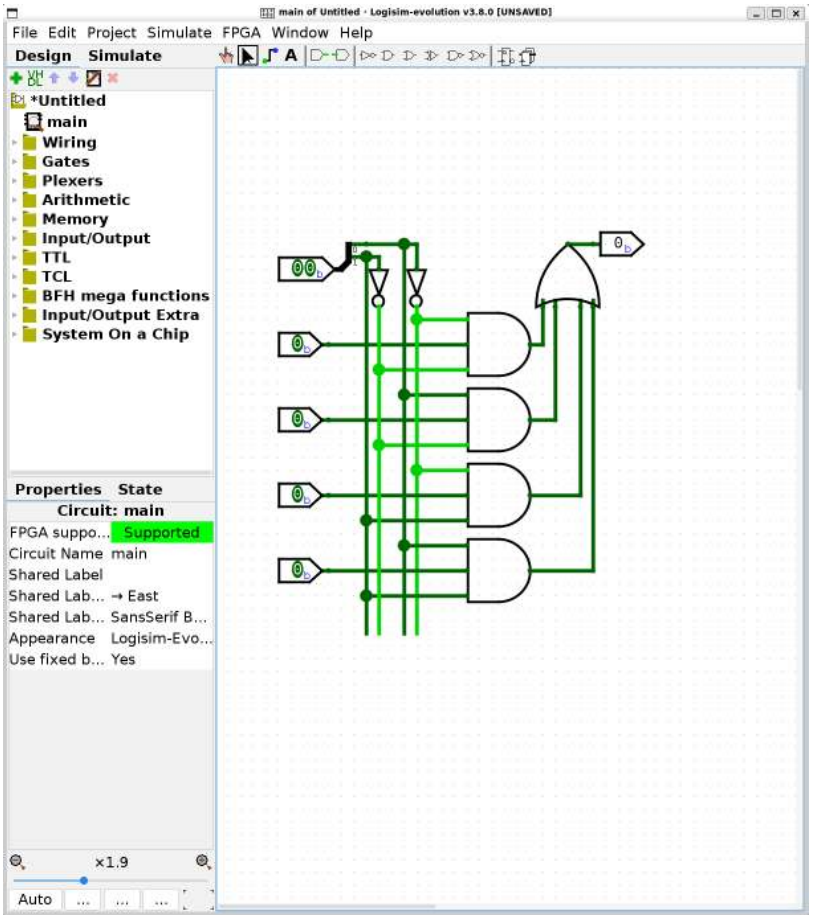


Image 1

3.2 八路选择器代码解释 5%

> 把`mux1`和`mux2`的输出作为`mux0`的输入，`S[2]`作为`mux0`的选择信号，
> 如果是0，选择`o1`，如果是1，选择`o2`，最终输出`O`。
> `s2,(s1,s0)`,先选择`s1`和`s0`选择`I0-I3`或者`I4-I7`，
> 再根据`s2`选择`o1`或者`o2`输出`O`。

```
module Mux8T1_1(
    input [7:0] I,
    input [2:0] S,
    output O
);
    //your verilog code
    wire o1, o2;
    Mux4T1_1 mux1(.I0(I[0]), .I1(I[1]), .I2(I[2]), .I3(I[3]), .s(S[1:0]),
.O(o1));
    Mux4T1_1 mux2(.I0(I[4]), .I1(I[5]), .I2(I[6]), .I3(I[7]), .s(S[1:0]),
.O(o2));
    //实例化两个四路选择器
    //注意大小写！！
    Mux2T1 mux0(.I0(o1), .I1(o2), .S0(S[2]), .O(O));
    //把mux1和mux2的输出作为mux0的输入，S[2]作为mux0的选择信号，
    //如果是0，选择o1，如果是1，选择o2，最终输出O。
    //s2,(s1,s0),先选择s1和s0选择I0-I3或者I4-I7，
    //再根据s2选择o1或者o2输出O。
endmodule

module Mux2T1(
    input I0,
    input I1,
    input S0,
    output O
);
    assign O = S0 ? I1 : I0;
endmodule
//补充定义Mux2T1模块，实现2选1多路选择器。
```

Fence 1

3.3 TestBench 设计 5%

给出$Mux8T1_1$的`TestBench`测试程序。分别选择`I[0]-I[7]`，输入的八位数据为`0b10101010`。

- 根据carry老师的建议，更改了测试脚本。

我试图采用for循环，但是寄存器会溢出，S被重置为0，进而导致死循环。

```
    initial begin
        //your test sample
        I = 8'b10101010;
        S=0; #5;S=1; #5;S=2; #5;S=3; #5;
        S=4; #5;S=5; #5;S=6; #5;S=7; #5;

        I = 8'b11110000;
        S=0; #5;S=1; #5;S=2; #5;S=3; #5;
        S=4; #5;S=5; #5;S=6; #5;S=7; #5;
```

```
10
11          I = 8'b11001100;
12          S=0; #5;S=1; #5;S=2; #5;S=3; #5;
13          S=4; #5;S=5; #5;S=6; #5;S=7; #5;
14          $finish;
15      end
```

Fence 2

3.4 其余实验步骤及截图 10%

3.4.1 仿真 Verilog 设计

- 这是$Mux8T1_1$的波形图。

更改前：

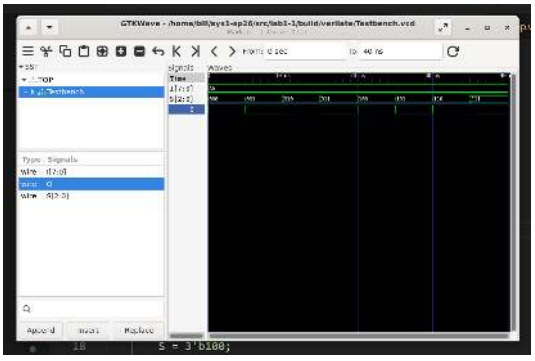


Image 2

更改后：

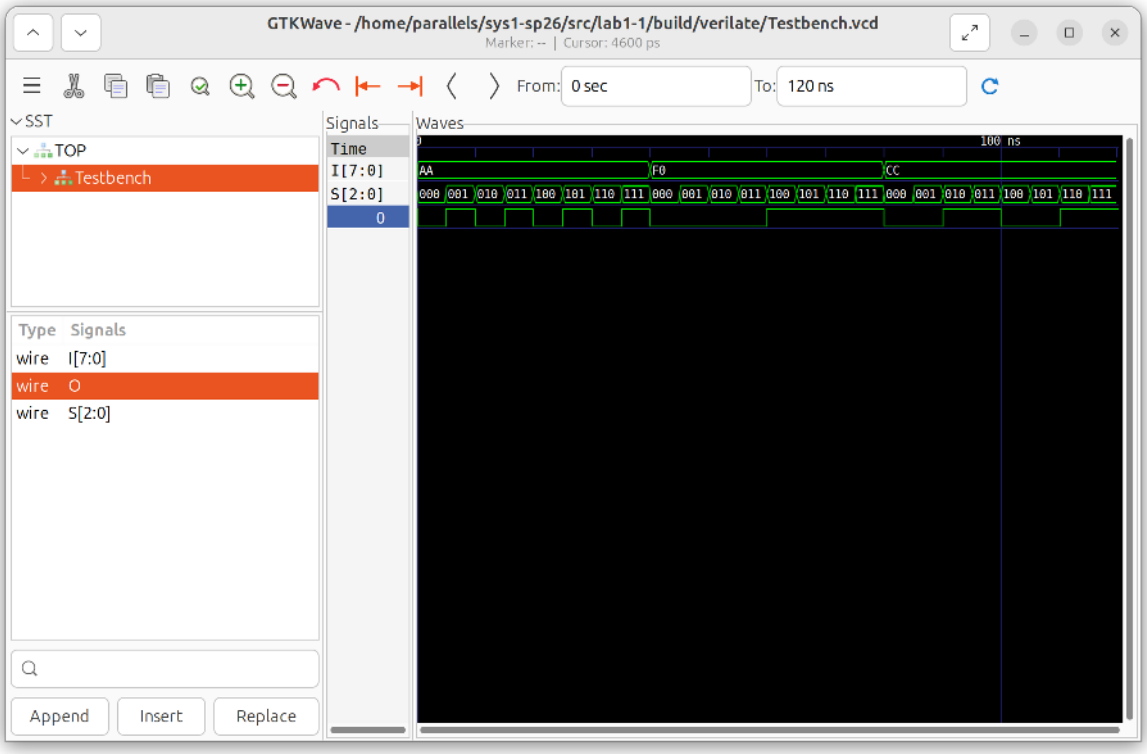


Image 3

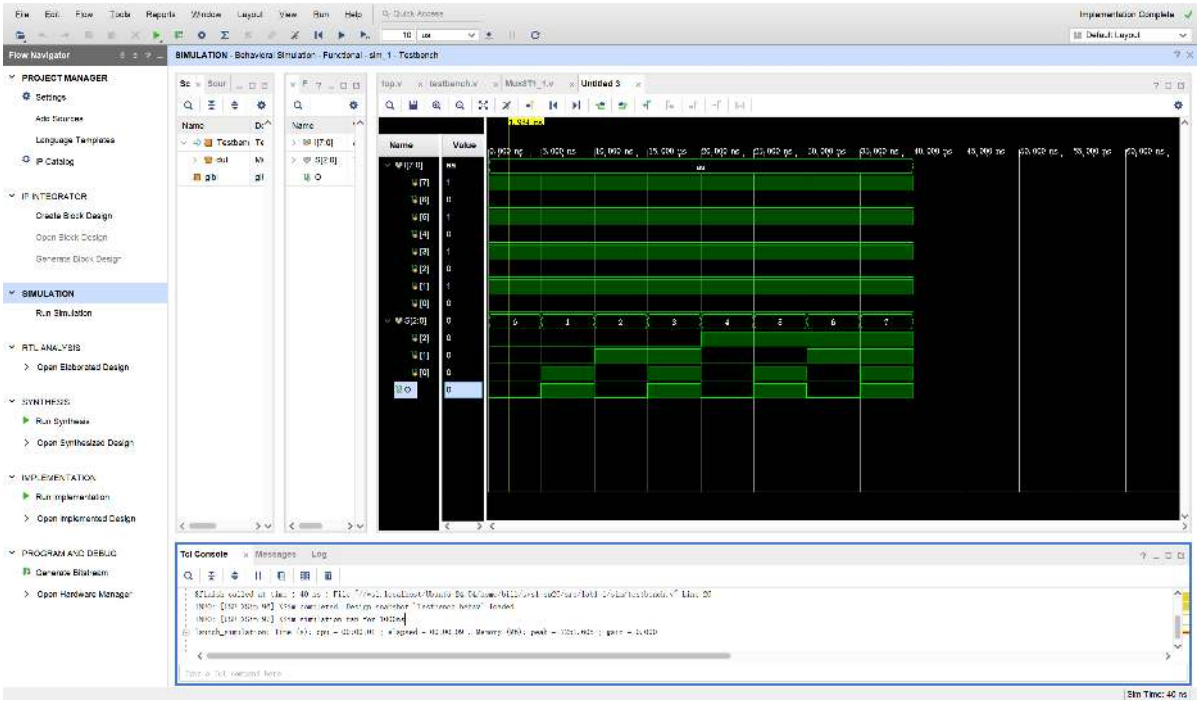


Image 4

- 这是$Mux4T1_4$的波形图。(lab1-2的正文部分)

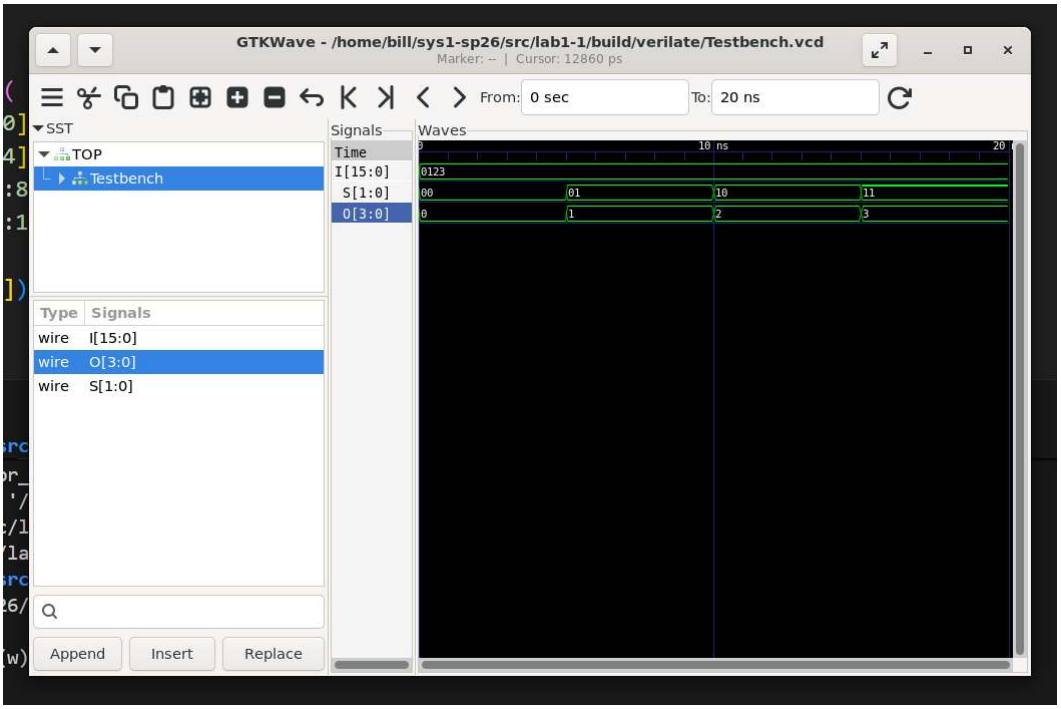


Image 5

3.4.2 Vivado 上板验证

3.4.2.1 xdc 文件（根据lab0的图片中的引脚数据）

```
## IO
set_property PACKAGE_PIN {J15} [get_ports {SW[0]}]
set_property IOSTANDARD {LVCMOS33} [get_ports {SW[0]}]
set_property PACKAGE_PIN {L16} [get_ports {SW[1]}]
set_property IOSTANDARD {LVCMOS33} [get_ports {SW[1]}]
set_property PACKAGE_PIN {M13} [get_ports {SW[2]}]
set_property IOSTANDARD {LVCMOS33} [get_ports {SW[2]}]
# add IO
set_property PACKAGE_PIN {R15} [get_ports {SW[3]}]
set_property IOSTANDARD {LVCMOS33} [get_ports {SW[3]}]
set_property PACKAGE_PIN {R17} [get_ports {SW[4]}]
set_property IOSTANDARD {LVCMOS33} [get_ports {SW[4]}]
set_property PACKAGE_PIN {T18} [get_ports {SW[5]}]
set_property IOSTANDARD {LVCMOS33} [get_ports {SW[5]}]
set_property PACKAGE_PIN {U18} [get_ports {SW[6]}]
set_property IOSTANDARD {LVCMOS33} [get_ports {SW[6]}]
set_property PACKAGE_PIN {R13} [get_ports {SW[7]}]
set_property IOSTANDARD {LVCMOS33} [get_ports {SW[7]}]
set_property PACKAGE_PIN {T8} [get_ports {SW[8]}]
set_property IOSTANDARD {LVCMOS33} [get_ports {SW[8]}]
set_property PACKAGE_PIN {U8} [get_ports {SW[9]}]
set_property IOSTANDARD {LVCMOS33} [get_ports {SW[9]}]
set_property PACKAGE_PIN {R16} [get_ports {SW[10]}]
set_property IOSTANDARD {LVCMOS33} [get_ports {SW[10]}]
set_property PACKAGE_PIN {H17} [get_ports {LD0}]
set_property IOSTANDARD {LVCMOS33} [get_ports {LD0}]
```

Fence 3

3.4.2.2 top 文件

```
module top(
  input [7+3:0] SW,
  output LD0
);
  // add connection from FPGA IO to the main
  //在 src/lab1-1/syn 中补全 top 文件和 xdc 文件,
  //然后用 GUI 模式或者 batch 模式综合下板。其中输入 I0-I7 S0-S2
  //对应板子从右到左的前十一个开关, LD0 对应从右到左的第一个 LED。
  Mux8T1_1 ldut(
      .I(SW[7:0]),
      .S(SW[10:8]),
      .O(LD0)
    );
endmodule
```

Fence 4

3.4.2.3 上板图片（部分）

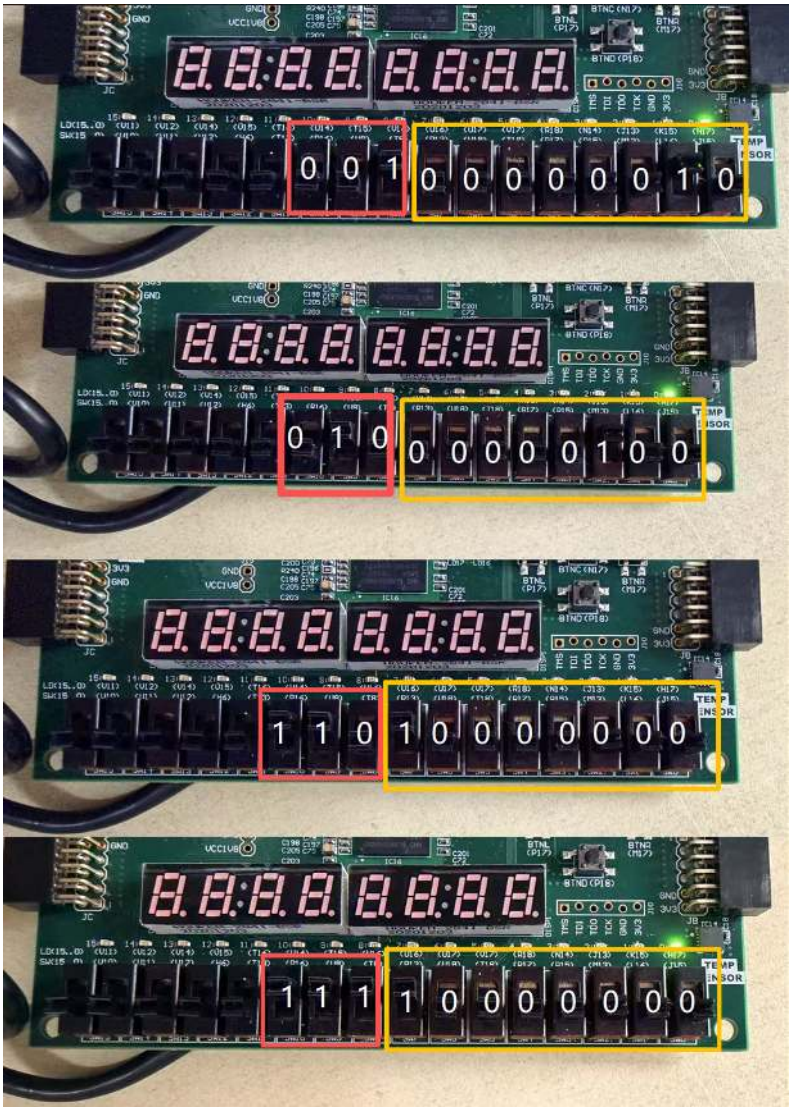


Image 6

3.5 思考题

> 在实验中我们已经拥有了 $Mux4$（控制的信号数）$T1\_1$（输入的位数），实际上它是由如 下图中展示的结构组成，思考如下问题，请在报告中给予解答。

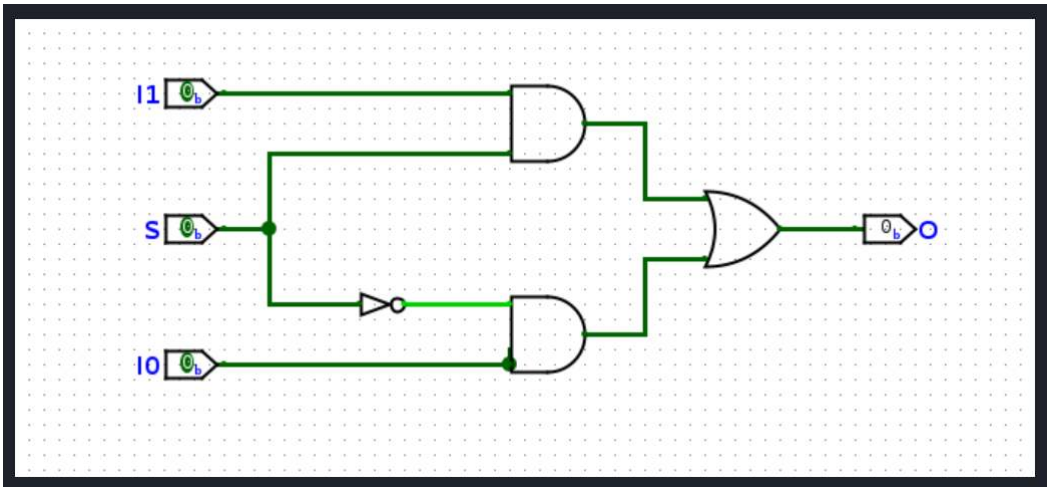
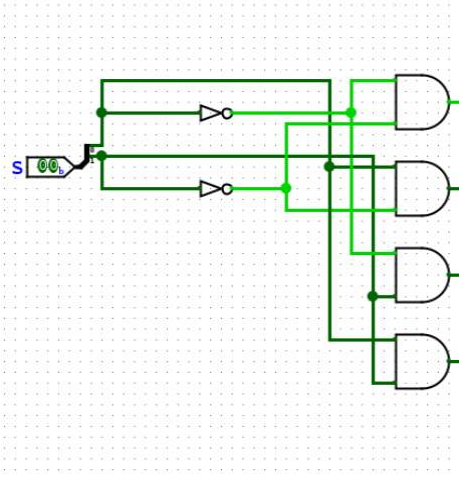


Image 7

- $Mux2T1\_1$ 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%

  > `1-bit decoder`（1位译码器），由输入`S`和一个非门构成。
  >
  > 与AND-OR结构共同构成`Mux2T1_1`

- $Mux4T1\_1$ 是如何组成的 6%

  > 输入`S`的低位分别经过两个`1-bit decoder`，其输出分别作为`1-bit decoder`的输入。`1-bit decoder`的开关是`S`的高位。
  >
  > 也可以根据文档的思路，建立类似“查找表”的结构，当两位分别匹配时选择该路：
  >
  > 
  >
  > Image 8
  >
  > 与AND-OR结构共同构成`Mux4T1_1`

- $Mux8T1\_1$ 是如何组成的 6%

  > “用 Mux4T1_1.v 为基础加一个二路选择器编写 **Mux8T1_1.v**”
  >
  > 输入`S`的低二位分别经过两个`2-bit decoder`，其输出分别作为`1-bit decoder`的输入。`1-bit decoder`的开关是`S`的高位。
  >
  > 与AND-OR结构共同构成`Mux8T1_1`

- 那么 $Mux2^mT1\_n$ 是如何构成的呢 10%

  > 输入`S`的低$(m-1)$位分别经过两个`(m-1)-bit decoder`，其输出分别作为`1-bit decoder`的输入。`1-bit decoder`的开关是`S`的高位。
  >
  > 与AND-OR结构共同构成$Mux2^mT1_1$
  >
  > `n`个$Mux2^mT1_1$ 的组合形成**复合多路选择器**$Mux2^mT1\_n$

lab 1-2：七段数码管译码器设计与应用

1. 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

2. 实验报告 50%

2.1 译码管设计 10%

不要大段贴代码，所以我贴一部分

```
// fill your code for decoder of b-g and p
// 输出为0是通电
wire b_or, b_result;
assign b_or = and_gate[5] | and_gate[6] | and_gate[11] | and_gate[12] | and_gate[14] | 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[14] | and_gate[15];
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;

assign p = ~point | LE;
```

Fence 5

2.2 复合多路选择器及语法分析比较 10%

> **Note**
>
> 使用 ?: 语法
>
> 简单易读。但是嵌套过多时可读性下降

```
module Mux4T1_32(...);
    assign O=S[1] ? (S[0] ? I3 : I2) : (S[0] ? I1 : I0);
endmodule
```

Fence 6

> **Note**
>
> 多个一位多路选择器组合

> 模块化，硬件结构直观。但是需要依赖其他模块。

```
module Mux4T1_32(...);
    Mux4T1_1 m1(I3[0], I2[0], I1[0], I0[0], O[0], S);
    Mux4T1_1 m2(I3[1], I2[1], I1[1], I0[1], O[1], S);
    Mux4T1_1 m3(I3[2], I2[2], I1[2], I0[2], O[2], S);
    Mux4T1_1 m4(I3[3], I2[3], I1[3], I0[3], O[3], S);
    //...
    //显然很麻烦，可以考虑用循环（？）
endmodule
```

Fence 7

> **Note**
>
> if-else语法
>
> 可读性强。需要reg输出。

```
module Mux4T1_32(...
    output reg [31:0] O
);
    always @(*) begin
        if (S[1]) begin
            if (S[0]) O = I3;
            else O = I2;
        end
        else begin
            if (S[0]) O = I1;
            else O = I0;
        end
    end
endmodule
```

Fence 8

2.3 综合实现数码管 10%

- 波形图：

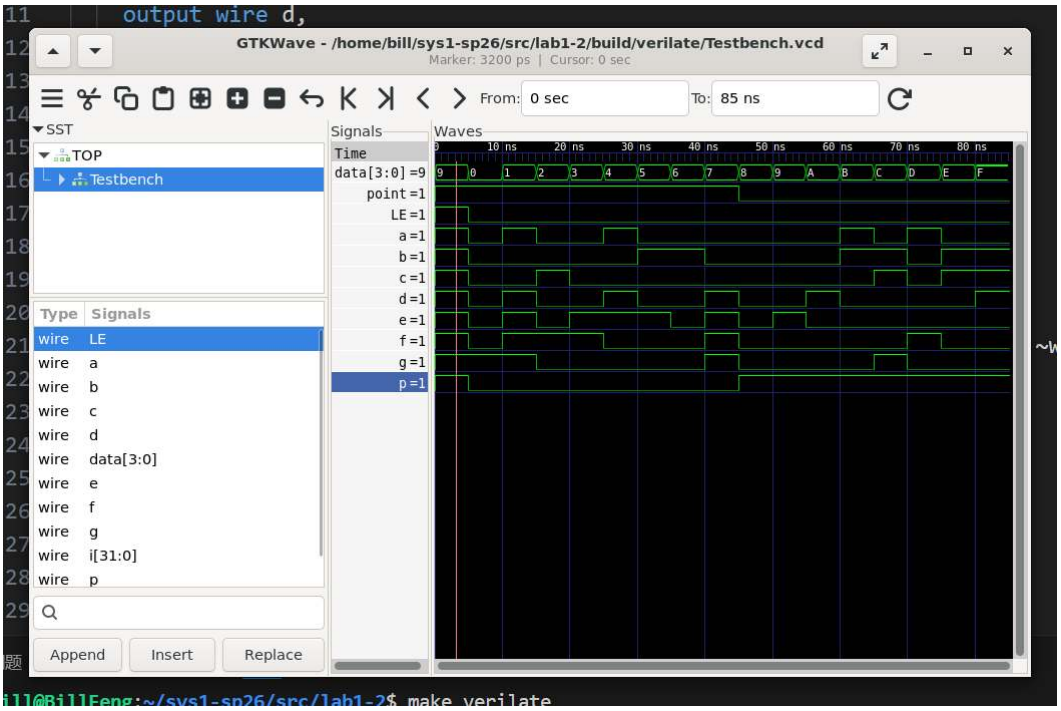


Image 9

- 上板图片（3250101697）：

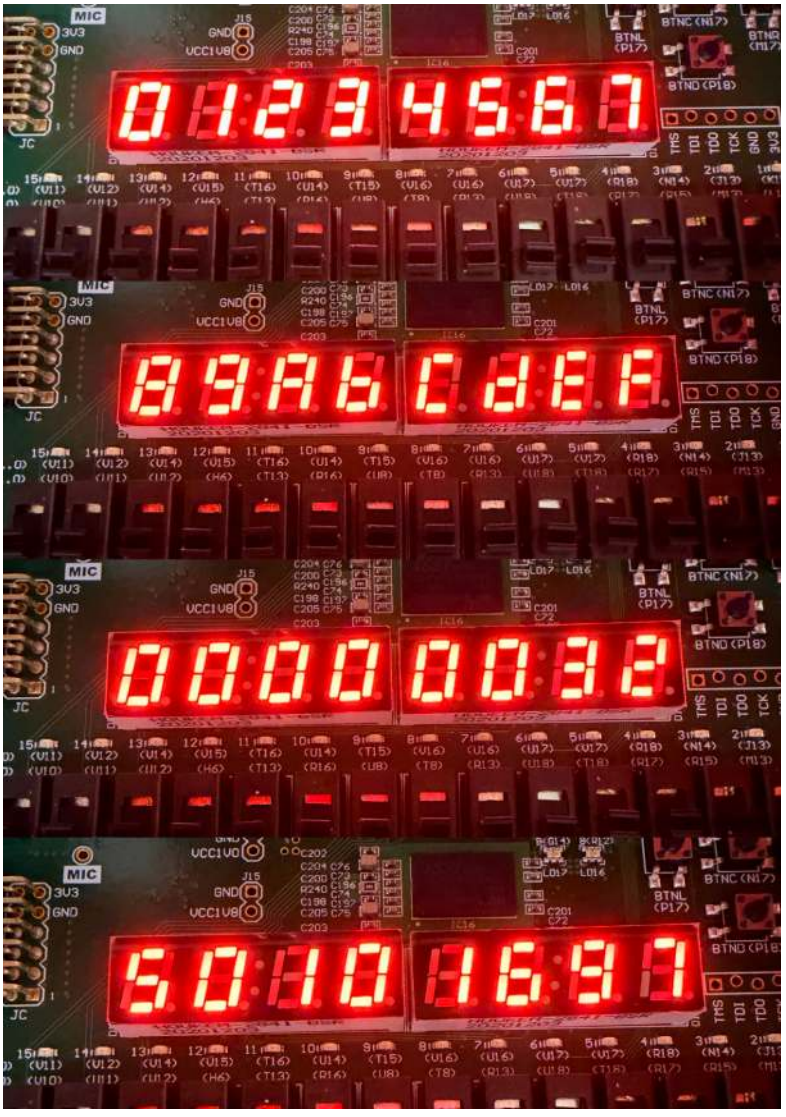


Image 10

2.4 思考题

- 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 10%

```
for(i=0;i<8;i=i+1)begin
    data=i[3:0];
    #5;
end
```

Fence 9

展开后：

```
data=4'h0;
#5;
data=4'h1;
#5;
data=4'h2;
#5;
data=4'h3;
#5;
data=4'h4;
#5;
data=4'h5;
#5;
data=4'h6;
#5;
data=4'h7;
#5;
```

Fence 10

> `for` 是循环语句，可以重复执行一段代码，替代手动赋值。语法与C语言的`for`类似。
>
> 区别：在硬件描述语言中，for 循环的每一次迭代实际上是生成多个并行的硬件电路。

- 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 10%

> 最喜欢的两个：1.级联，因为结构直观，便于理解；2.索引，因为代码简洁。

- 有兴趣的同学请自行阅读 `repo/sys-project/lab1-2/syn` 的代码细节，分享你的心得体会。

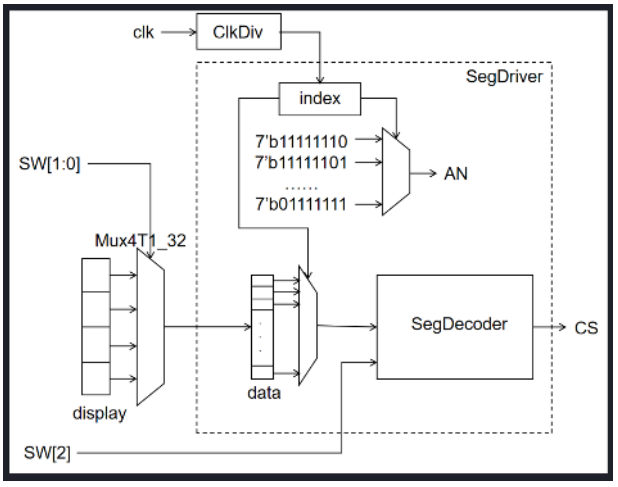


Image 11

心得体会

好玩，做lab1的时候正好过生日，于是做了个字母版本的：

Image 12

另外，vivado成功在mac上的虚拟机跑通了，Batch可以直接下板。

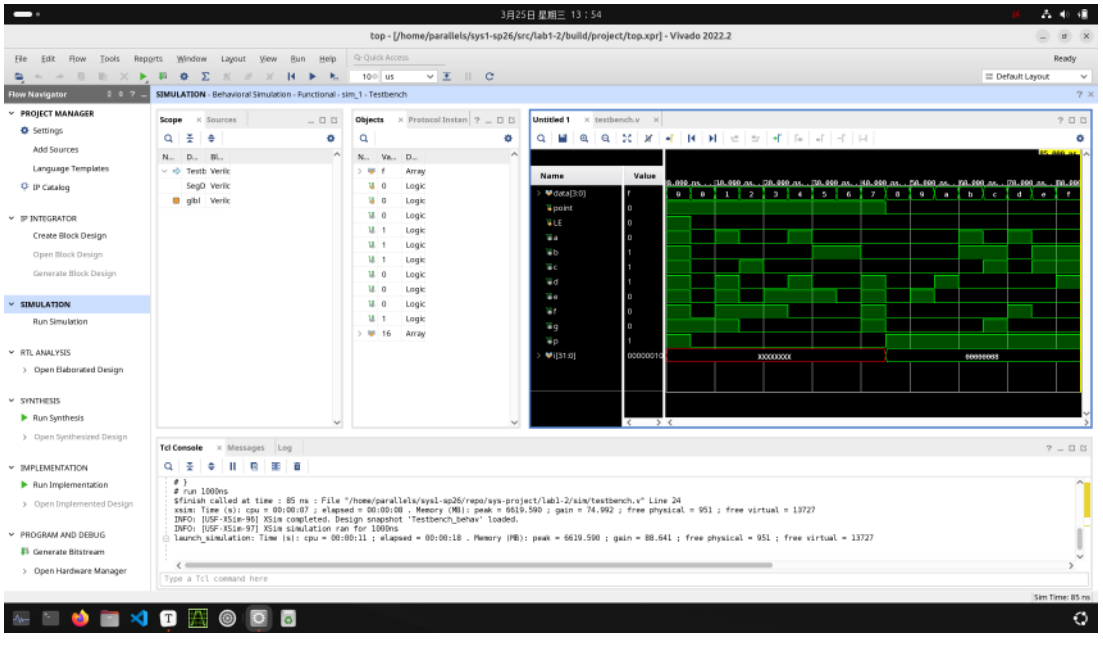


Image 13