

实验二 ——

七段显示部件拓展

姓名： 方晓霖 学号： 3150105184 专业： 混合班

课程名称： 计算机组成实验 同组学生姓名： 无

实验时间： 2017-03-15 实验地点： 紫金港东 4-509 指导老师： 张明敏、洪奇军

一、实验目的和要求

1.1 实验目的

1. 了解设备与接口
2. 了解人机交互
3. 了解计算机通讯
4. 了解最简单的接口 GPIO
5. 了解用 GPIO 实现简单的人机交互

1.2 实验要求

1. 优化逻辑实验输出的显示模块
 - 将原理转化为结构化行为描述
 - 增加七段码文本图形显示
2. 在 Exp01 上增加修改验证

二、主要仪器设备

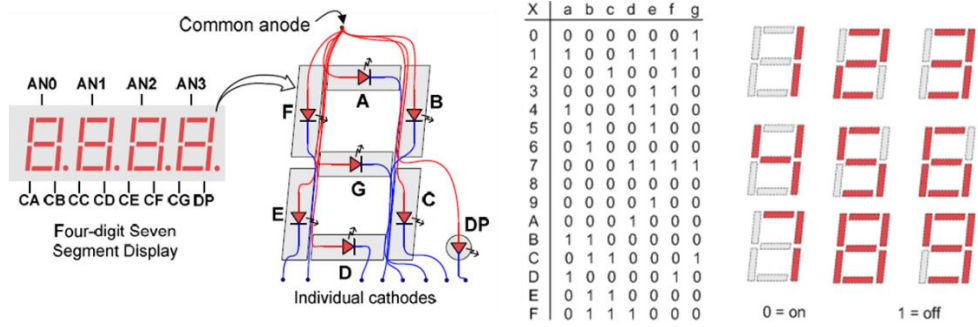
实验设备

装有 Xilinx ISE14.7 的电脑	1 台
Sword 开发板	1 个

三、实验内容和原理

3.1 逻辑实验七段显示模块优化

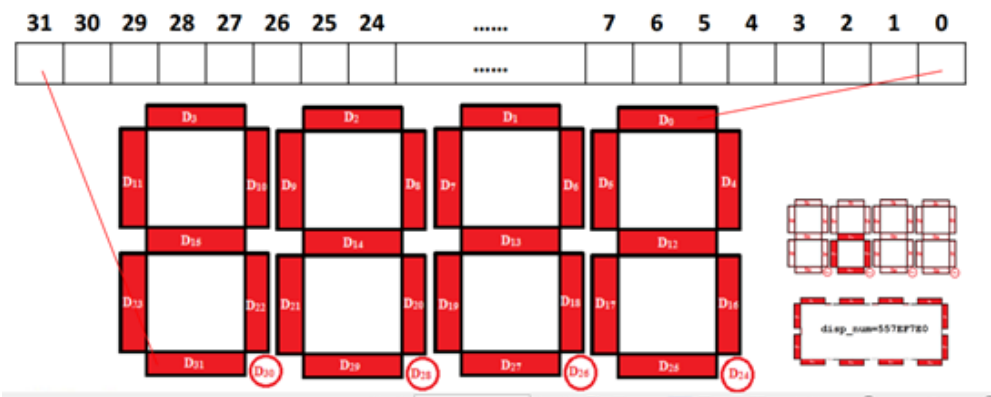
3.1.1 七段译码器文本显示



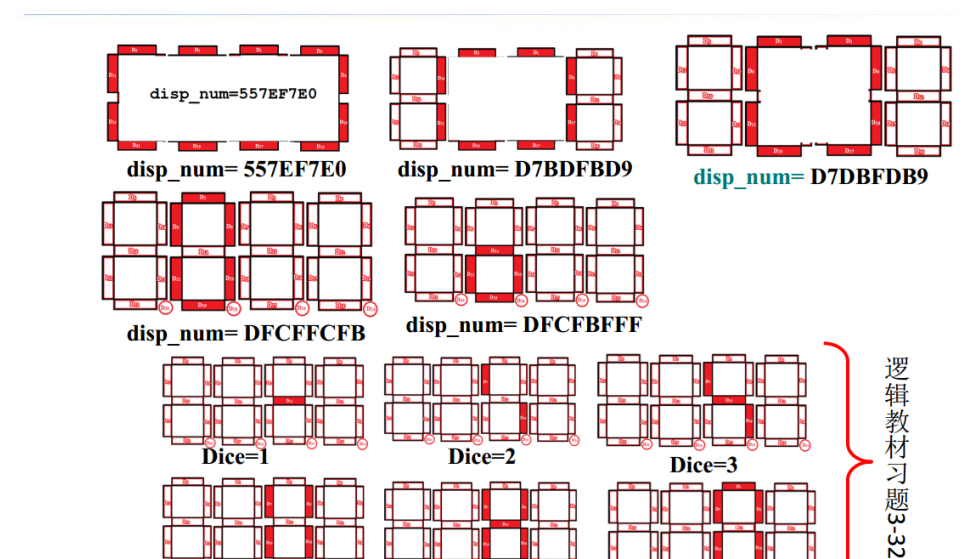
当 SW[0] 为 1，输出为文本显示。
在四位 LED 显示器上需要分为两个 16 位显示，通过 SW[1] 旋转。在高位时显示高 16 为，为 0 显示低 16 位。

3.1.2 七段译码器图形显示

当 SW[0] 为 0，输出为图形显示。



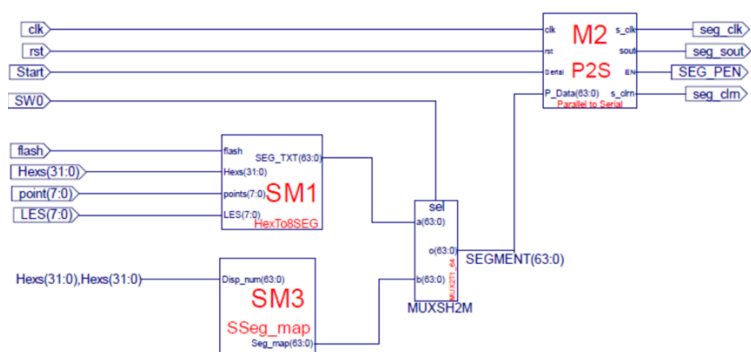
通过图上的对应关系，映射到 32 位字，可通过编程显示不同图形。



这是一些对应的例子。

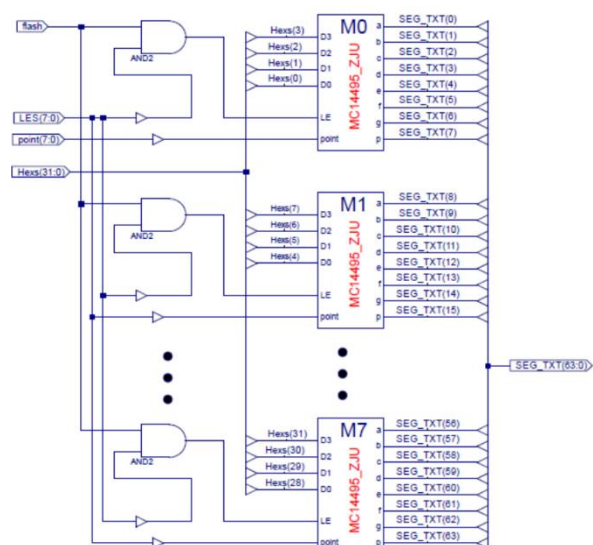
每个 8 位十六进制的数字转换成二进制表示，有 32 位，对应 32 个 LED 灯的亮/灭，从而达到映射图形的目的。

3.1.3 Sword 平台八位七段显示结构



3.1.4 八位七段显示器静态译码结构：HexTo8SEG.v

自行用 HDL 结构化描述



3.1.5 Seg_Map

```
assign Seg_map =
{Disp_num[0], Disp_num[4], Disp_num[16], Disp_num[25], Disp_num[17], Disp_num[5], Disp_num[12], Disp_num[24],
Disp_num[1], Disp_num[6], Disp_num[18], Disp_num[27], Disp_num[19], Disp_num[7], Disp_num[13], Disp_num[26],
Disp_num[2], Disp_num[8], Disp_num[20], Disp_num[29], Disp_num[21], Disp_num[9], Disp_num[14], Disp_num[28],
Disp_num[3], Disp_num[10], Disp_num[22], Disp_num[31], Disp_num[23], Disp_num[11], Disp_num[15], Disp_num[30],

Disp_num[0], Disp_num[4], Disp_num[16], Disp_num[25], Disp_num[17], Disp_num[5], Disp_num[12], Disp_num[24],
Disp_num[1], Disp_num[6], Disp_num[18], Disp_num[27], Disp_num[19], Disp_num[7], Disp_num[13], Disp_num[26],
Disp_num[2], Disp_num[8], Disp_num[20], Disp_num[29], Disp_num[21], Disp_num[9], Disp_num[14], Disp_num[28],
Disp_num[3], Disp_num[10], Disp_num[22], Disp_num[31], Disp_num[23], Disp_num[11], Disp_num[15], Disp_num[30]};
```

可以参照之前那张 LED 灯对应的图，每一行的 8 个 Disp_num 的分量是对应的那位数字的 a, b, c, d, e, f, g, h 管的信号。从第一行到第八行分别为八个数字的信号。

3.2 LED 并行显示模块 U7: SPIO

3.2.1 15 位 LED 指示灯控制

逻辑实验的输出 LED 显示模块，相当于通用输入输出接口：GPIO

基本功能：

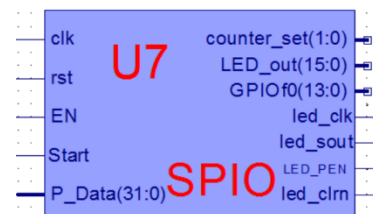
输入 32 位二进制数据：P_Data

- clk=时钟，EN：输出使能，Start：串行扫描启动，rst=复位

串行输出：led_clk=时钟，led_sout=串行输出数据，LED_PEN=使能，led_clrn=清零

并行输出：LED_out、counter_set、GPIOf0

逻辑符号——



3.2.2. 并转串移位寄存核

Sword 平台板七段显示器和 LED

采用串行输出，由 74LS161 串入并出锁存显示

8 位七段显示器共 64 个显示点

- 64 位并串转换：64 位并入串出移位寄存器
- 由 P2S. ngc 核承担

16 个 LED

- 16 位并串转换：16 位并入串出移位寄存器
- 由 LED_P2S. ngc 核承担



```

module      P2S(input wire clk,           //parallel to serial
               input wire rst,
               input wire Serial,
               input wire[DATA_BITS-1:0] P_Data,
               output reg  s_clk,
               output wire s_clrn,
               output wire sout,
               output reg  EN
               );

parameter
    DATA_BITS = 64,           // data length
    DATA_COUNT_BITS = 6;

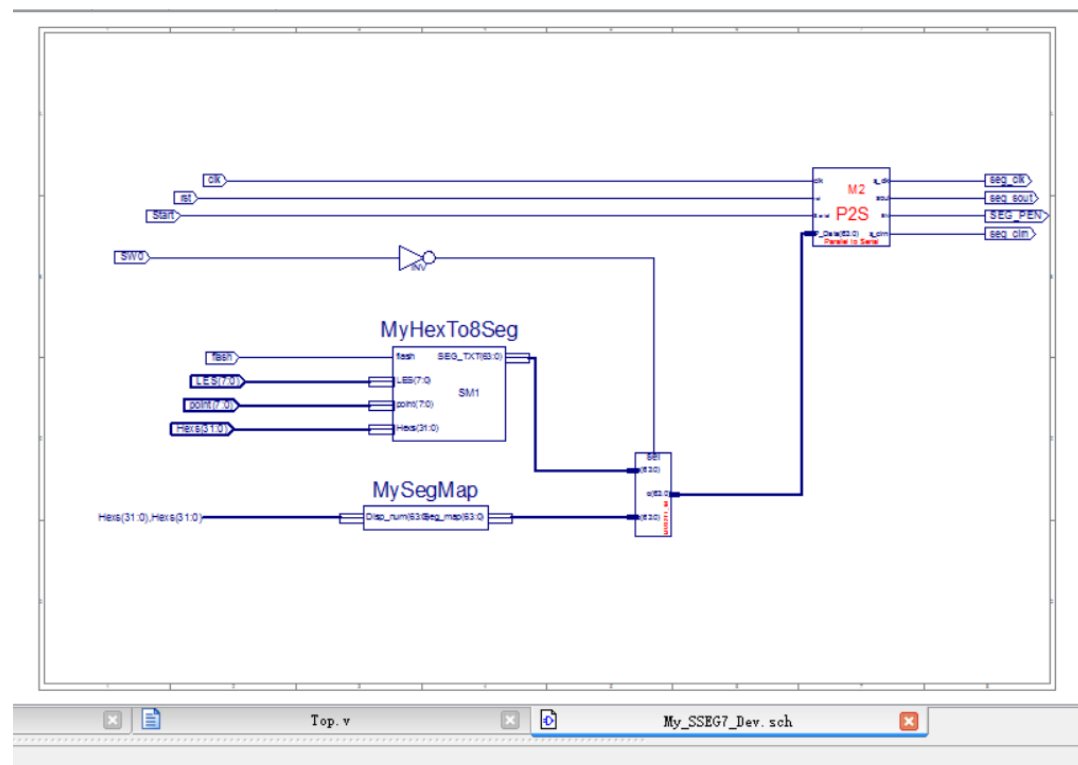
endmodule

```

四、操作方法与实验步骤

4.1. 实现八位七段显示器模块 SSeg7_Dev.v

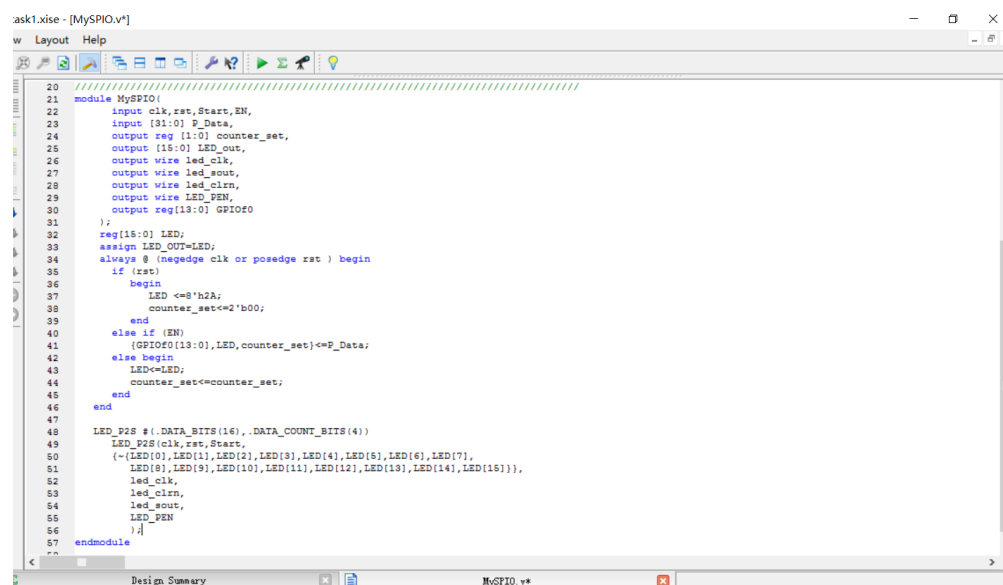
顶层用逻辑图实现，其余用 HDL 结构化描述



这里用到的模块都是自己写的替换过的模块（除了 P2S 用核代替，课外有时会自己试着重新写一下）。在板子上验证时，发现高位显示文字低位显示图形，和小 LED 板子相反，所以这里和课件上给的图不一样，自己加了一个非门。

第一次下板子验证的时候，大的 LED 屏幕图形输出时一直显示全 1，不对开关做出应答，最后发现是 MySegMap 输入信号没有接上，很不容易发现，可能是拖动模块的时候出现了问题。这里要注意可视化编程可能存在的问题。

4.2. 设计实现并行输出兼 LED 显示模块：SPI0.v

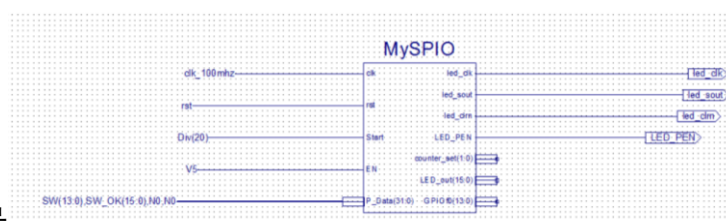


```
20 ////////////////////////////////////////////////////////////////////////////////////
21 module MySPI0(
22     input clk, rst, Start, EN,
23     input [31:0] P_Data,
24     output reg [15:0] counter_set,
25     output [15:0] LED_out,
26     output wire led_clk,
27     output wire led_clrn,
28     output wire led_sout,
29     output wire LED_PEN,
30     output reg [13:0] GPIOf0
31 );
32 reg [15:0] LED;
33 assign LED_OUT=LED;
34 always @ (negedge clk or posedge rst ) begin
35     if (rst)
36         begin
37             LED <= 8'h2A;
38             counter_set <= 2'b00;
39         end
40     else if (EN)
41         {GPIOf0[13:0], LED, counter_set} <= P_Data;
42     else begin
43         LED <= LED;
44         counter_set <= counter_set;
45     end
46 end
47
48 LED_P2S #( .DATA_BITS(16), .DATA_COUNT_BITS(4))
49 LED_P2S(clk, rst, Start,
50     {~LED[0], LED[1], LED[2], LED[3], LED[4], LED[5], LED[6], LED[7],
51     LED[8], LED[9], LED[10], LED[11], LED[12], LED[13], LED[14], LED[15]}},
52     led_clk,
53     led_clrn,
54     led_sout,
55     LED_PEN
56 );
57 endmodule
```

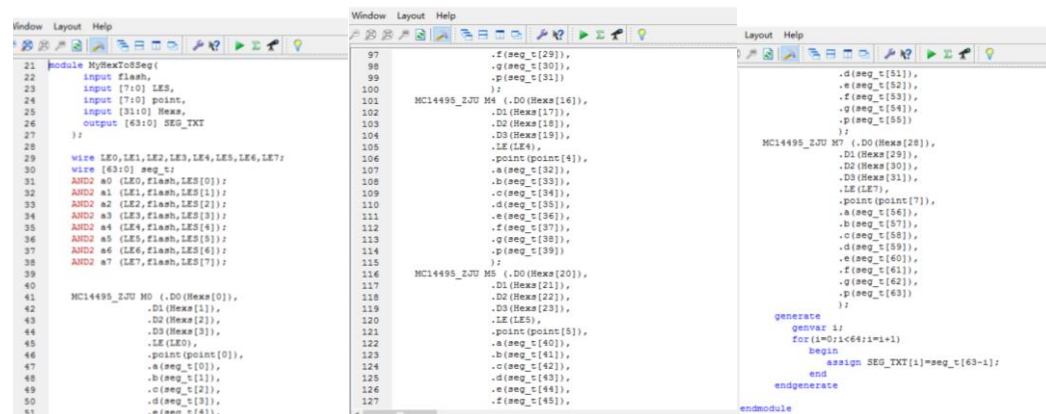
代码如图。

当 rst 复位信号为 1 的时候，复位，LED 信号置为 2A，counter 置为 00。使能信号为 1 时，将 P_Data 信号赋给 {GPIOf0[13:0], LED, counter_set} 这里调用了 LED_P2S 的核

逻辑符号



4.3 设计 8 位七段显示器静态译码模块：HexTo8SEG.v



用结构语言描述。

最后用了一个 generate，将 seg_t 里的值输到寄存器 SEG_TXT 输出。注意这里的信号是反过来的，SEG_TXT[0]对应 seg_t[63]，因为输出的高位和显示信号的高位是反过来的，否则输出文字会错误。

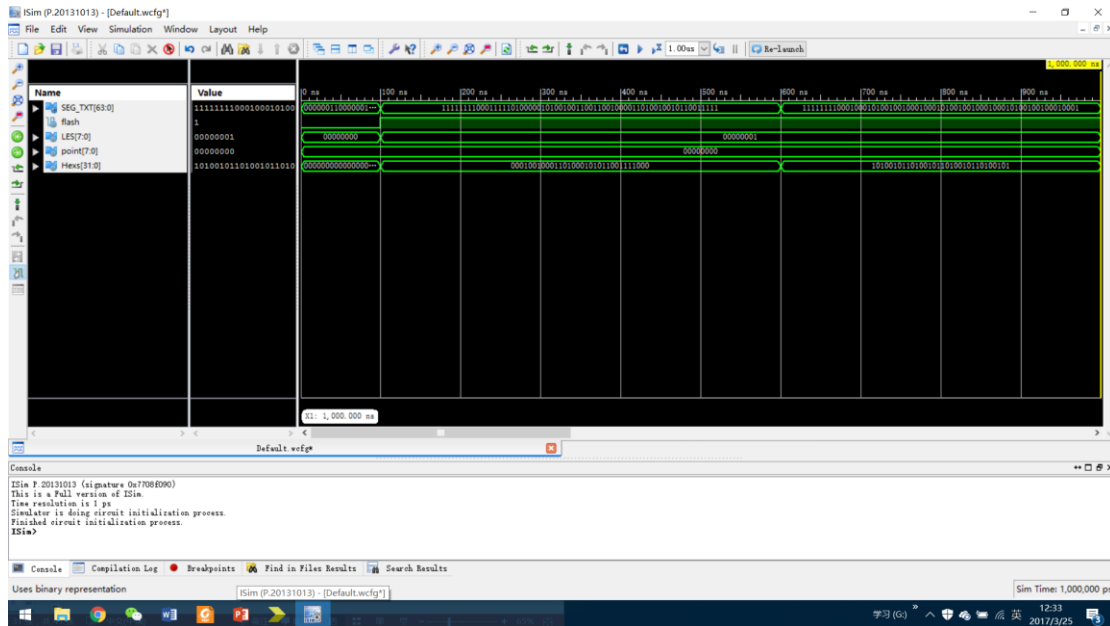
仿真调试验证模块

- 激励输入：Hexs=32'h12345678 和 Hexs=32'hA5A5A5A5
- flash=1, LES=1

激励信号代码

```
initial begin
    // Initialize Inputs
    flash = 0;
    LES = 0;
    point = 0;
    Hexs = 0;

    // Wait 100 ns for global reset to finish
    #100;
    Hexs=32'h12345678;
    flash=1;
    LES=1;
    #500;
    Hexs=32'hA5A5A5A5;
    flash=1;
    LES=1;
end
endmodule
```



Hexs 为 12345678 时

SEG_TXT 低 8 位为 10011111, b, c 管为低位, 显示 1, 正确。

SEG_TXT 第 15 到 8 位为 00100101, a, b, d, e, g 管为低位, 显示 2, 正确。

同理, 1 到 8 的输出均正确无误。

第 600nsHexs 信号改变后, 变为 A5A5A5A5

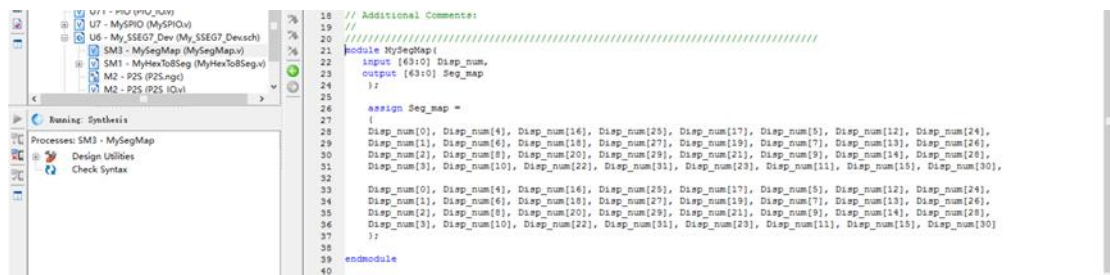
SEG_TXT 第 7 到 0 位为 00010001, a, b, c, e, f, g 管亮, 显示 A, 正确。

同理, 其他输出均正确无误。

以此类推, 文字输出无误, 仿真模拟通过。

4.4 设计 8 位七段显示器点阵映射模块: Seg_Map.v

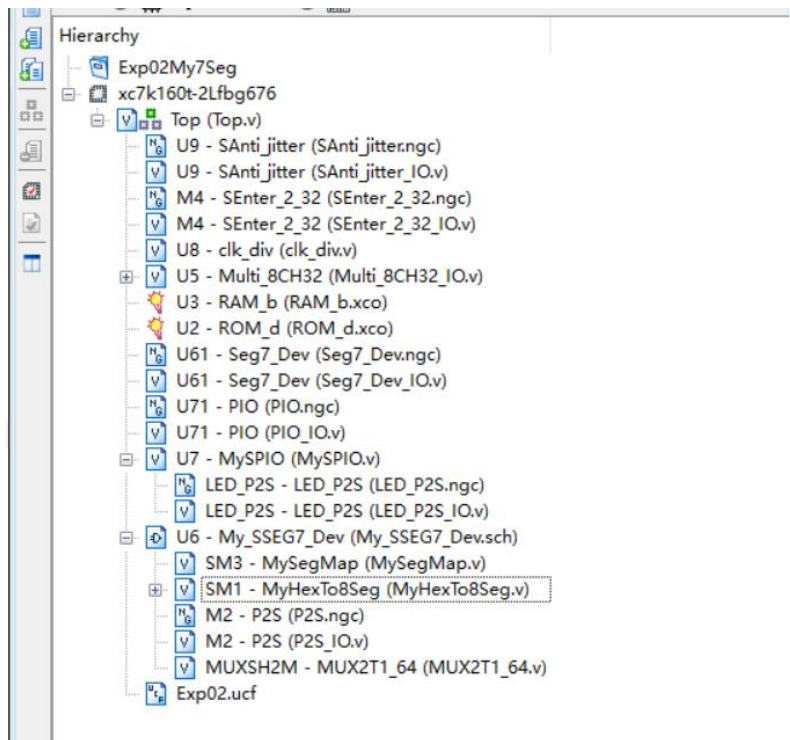
用 HDL 描述



如第一行, Disp_num 的 0, 4, 16, 25, 17, 5, 12, 24, 就是低位数字的 a, b, c, d, e, f, g, h 管。第一列, 0, 1, 2, 3, 0, 1, 2, 3 就是八个数字的 a 管。由于高 4 位和低 4 位的信号是相同的, 因此只用 32 位即可表示。

对应的图已经在原理部分说明。

4.5 替换实验 1 工程中的相应部件



如图，SSEG_Dev, SPIO 模块已被替换。

五、实验结果与分析

根据给出的 UCF 定义，对应按键功能为——

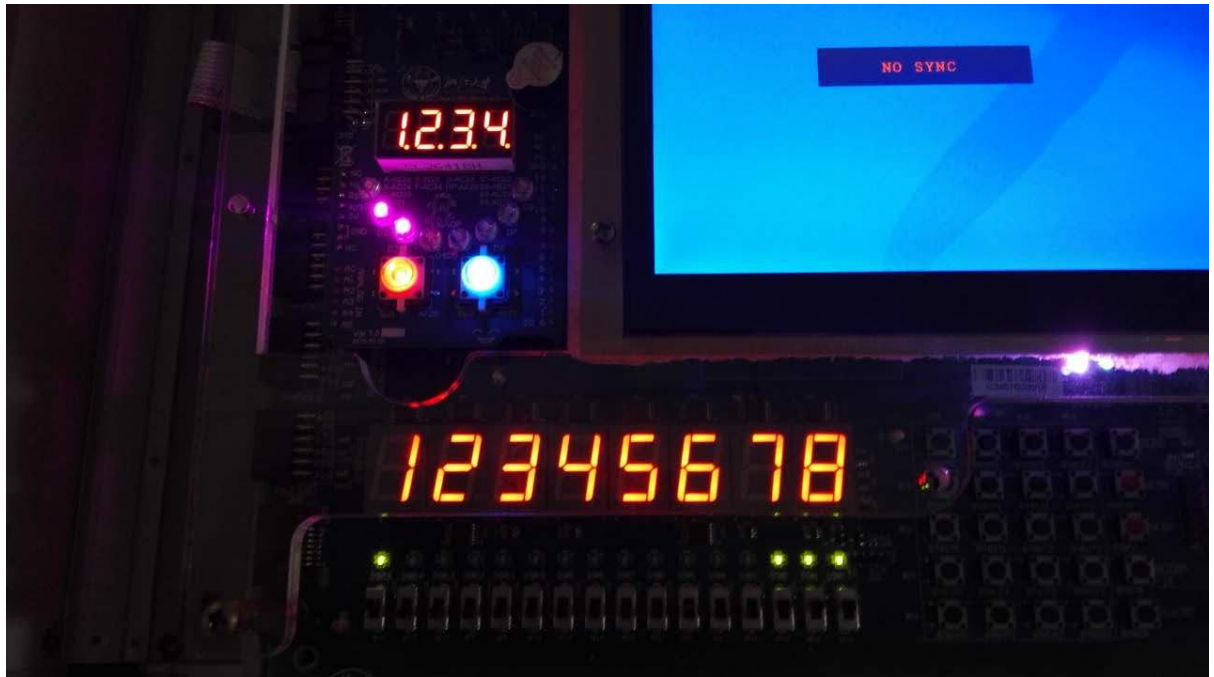
输入设备功能定义



开关定义	=0	=1	备注
SW[0]	图形(七段点阵)	文本(16进制)	
SW[1]	32位二进制高16位	32位二进制低16位	Arduino-Sword 002
SW[2]	CPU全速时钟	CPU单步钟	CPU时钟切换
SW[4]	存储器写禁止	存储器写使能	存储单元写控制
SW[7:5]	=000	通道0	Ai
	=001	通道1	Bi
	=010	通道2	SUM(ALU_Out)
	=011	通道3	Sign extension
	=100	通道4	1 bit Ext. to 32 bits
	=101	通道5	通用分频输出
	=110	通道6	ROM_D输出
	=111	通道7	RAM_B输出D(31:0)
按键定义	=0	=1	备注
BTN[0]		正脉冲左移	SW[15]=0,SW[7:5]<=001
BTN[1]		正脉冲右移	SW[15]=0,SW[7:5]<=001
BTN[2]		正脉冲输入修改	SW[15]=0,SW[7:5]<=001
RSTN			长按复位

SW[0]=1——文本输出

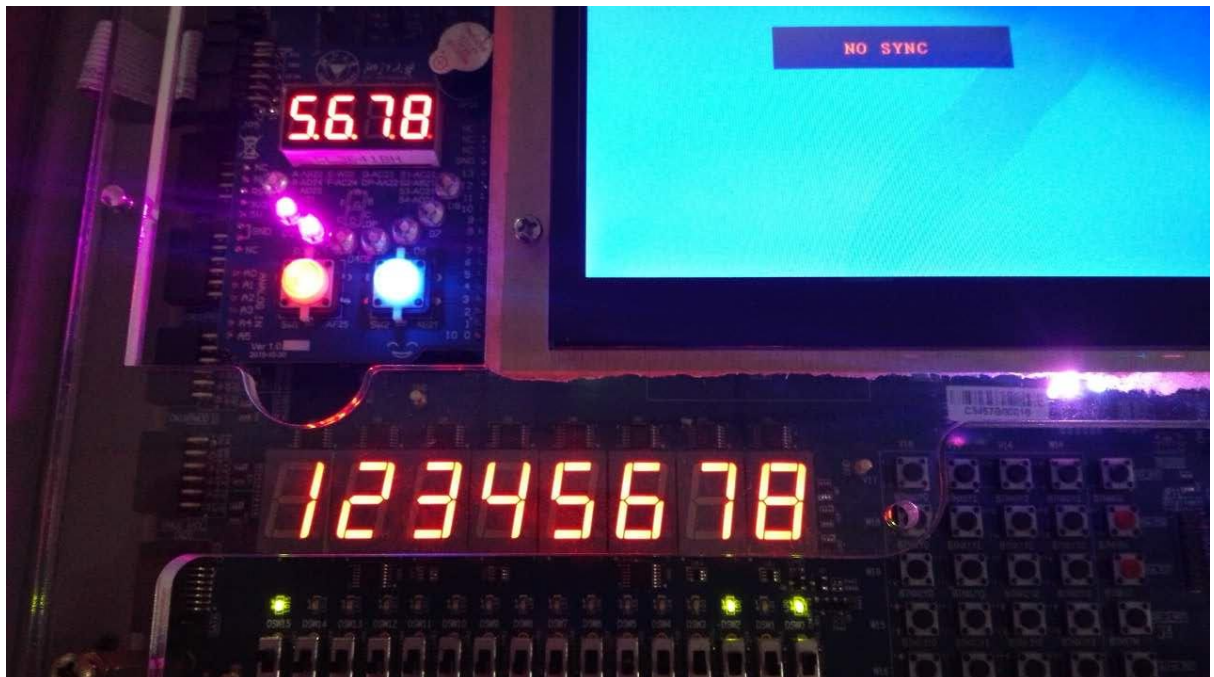
当开关处在高位，文本 16 进制显示数字 1 到 8

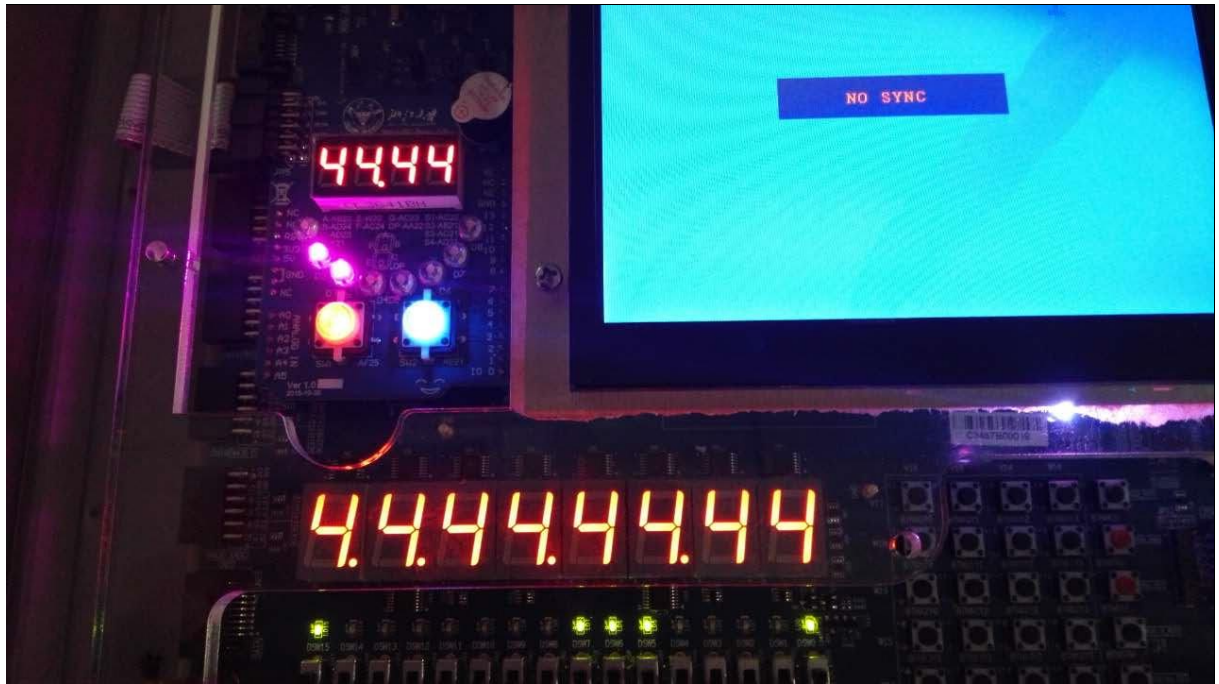


SW[1]——高低 16 位选择

当开关在高位，如上图所示，显示前 4 位高位

当开关在低位，显示后 4 位低位



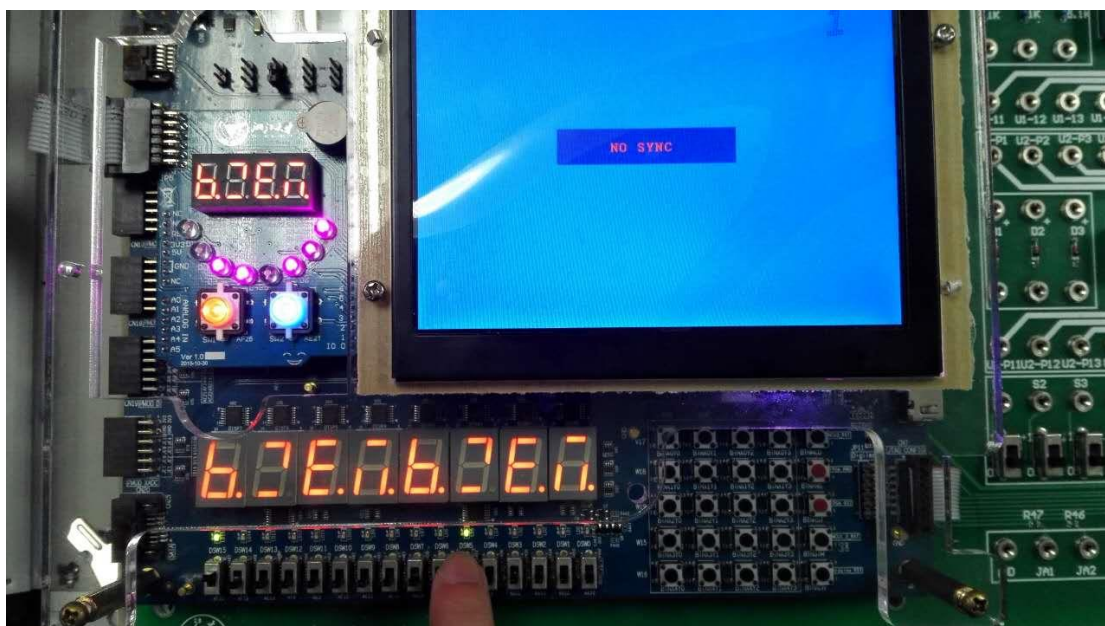


控制 SW[7:5], 选择不同输出模式, 已经在实验 1 中实现, 这里验证无误

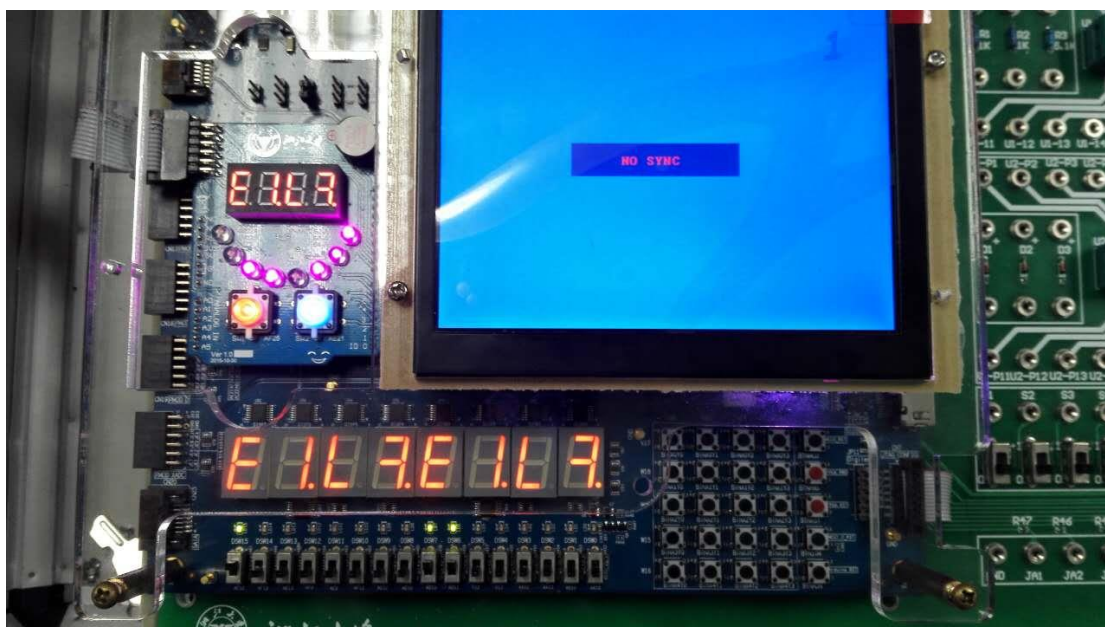
SW[0]=0——图形输出

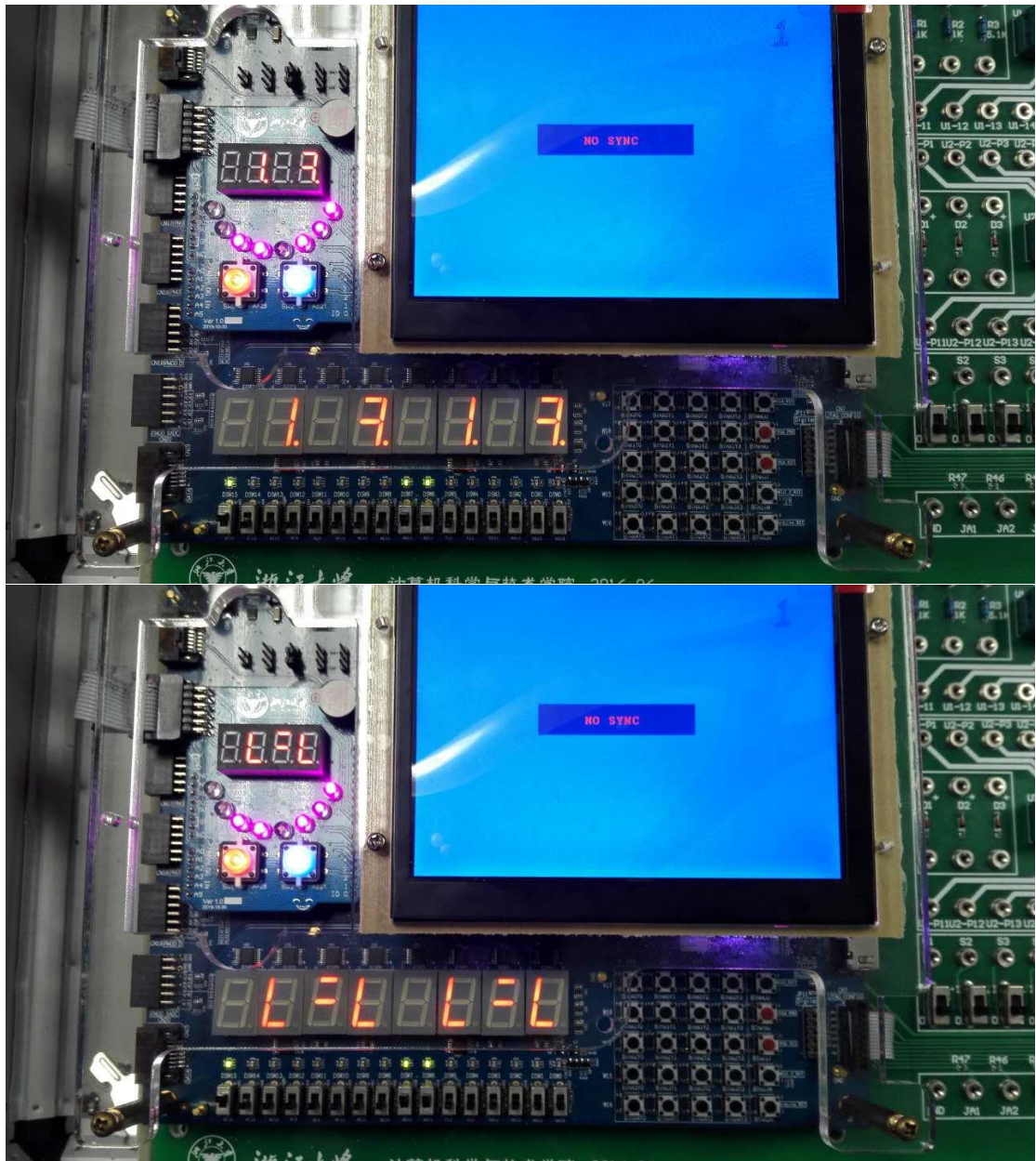
控制 SW[7:5], 选择不同输出模式, 实现诸如图形静态显示、跑马灯等功能。





跑马灯





实现无误

六、讨论、心得

本次实验主要是在上一次实验的基础上，替换两个模块。

在原理图设计实现过程中，连线时可能出现一些 bus 命名上的问题，需要小心注意，否则如果出现 bug，需要很长时间才能找到并修正。比如在 SSEG7_Dev 图中，第一次下板验证时发现八位 LED 显示图形时不动，后来找到 bug 是图中一根导线没接上，信号没有传输过去。这也体现了可视化编程的一些弊端。同时自己以后也要多注意一下。

有一些细节也要注意，比如在 HexTo8Seg 模块中，输出信号时，高低位和寄存器里面时反过来的，前面的数字是 output 的低位，因此要用 for 循环赋值调

整一下，否则输出文字会错误。

我在实验的时候对 P2S 模块很感兴趣。希望课后自己有时间能重新写一下图形显示模块，输出自己想要的图形。这个过程十分有趣。

经过此次试验，日后实验的基本框架。已经基本搭建起来，在 SWORD 实验板上的结果实现无误。如果还有时间，还可以通过修改不同通路的值实现更多功能，如 LED 跑马灯等。

附——代码

MySPI0.v

```
module MySPI0(
    input clk,rst,Start,EN,
    input [31:0] P_Data,
    output reg [1:0] counter_set,
    output [15:0] LED_out,
    output wire led_clk,
    output wire led_sout,
    output wire led_clrn,
    output wire LED_PEN,
    output reg[13:0] GPIOf0
);
    reg[15:0] LED;
    assign LED_out=LED;
    always @ (negedge clk or posedge rst ) begin
        if (rst)
            begin
                LED <=8'h2A;
                counter_set<=2'b00;
            end
        else
            if (EN)
                {GPIOf0[13:0],LED,counter_set}<=P_Data;
            else begin
                LED<=LED;
                counter_set<=counter_set;
            end
        end
    end

    LED_P2S #(.DATA_BITS(16),.DATA_COUNT_BITS(4))
        LED_P2S(clk,rst,Start,
            {~{LED[0],LED[1],LED[2],LED[3],LED[4],LED[5],LED[6],LED[7],
                LED[8],LED[9],LED[10],LED[11],LED[12],LED[13],LED[14],LED[15]}}
        ,
            led_clk,
            led_clrn,
            led_sout,
            LED_PEN
        );
endmodule
```

MySegMap. v

```
module MySegMap(
    input [63:0] Disp_num,
    output [63:0] Seg_map
);

    assign Seg_map =
    {
        Disp_num[0],    Disp_num[4],    Disp_num[16],    Disp_num[25],
        Disp_num[17], Disp_num[5], Disp_num[12], Disp_num[24],
        Disp_num[1],    Disp_num[6],    Disp_num[18],    Disp_num[27],
        Disp_num[19], Disp_num[7], Disp_num[13], Disp_num[26],
        Disp_num[2],    Disp_num[8],    Disp_num[20],    Disp_num[29],
        Disp_num[21], Disp_num[9], Disp_num[14], Disp_num[28],
        Disp_num[3],    Disp_num[10],    Disp_num[22],    Disp_num[31],
        Disp_num[23], Disp_num[11], Disp_num[15], Disp_num[30],

        Disp_num[0],    Disp_num[4],    Disp_num[16],    Disp_num[25],
        Disp_num[17], Disp_num[5], Disp_num[12], Disp_num[24],
        Disp_num[1],    Disp_num[6],    Disp_num[18],    Disp_num[27],
        Disp_num[19], Disp_num[7], Disp_num[13], Disp_num[26],
        Disp_num[2],    Disp_num[8],    Disp_num[20],    Disp_num[29],
        Disp_num[21], Disp_num[9], Disp_num[14], Disp_num[28],
        Disp_num[3],    Disp_num[10],    Disp_num[22],    Disp_num[31],
        Disp_num[23], Disp_num[11], Disp_num[15], Disp_num[30]
    };

endmodule
```

MyHexTo8Seg. v

```
module MyHexTo8Seg(
    input flash,
    input [7:0] LES,
    input [7:0] point,
    input [31:0] Hexs,
    output [63:0] SEG_TXT
);

    wire LE0,LE1,LE2,LE3,LE4,LE5,LE6,LE7;
    wire [63:0] seg_t;
    AND2 a0 (LE0,flash,LES[0]);
    AND2 a1 (LE1,flash,LES[1]);
    AND2 a2 (LE2,flash,LES[2]);
    AND2 a3 (LE3,flash,LES[3]);
    AND2 a4 (LE4,flash,LES[4]);
    AND2 a5 (LE5,flash,LES[5]);
    AND2 a6 (LE6,flash,LES[6]);
    AND2 a7 (LE7,flash,LES[7]);
```

```
MC14495_ZJU M0 (.D0(Hexs[0]),  
    .D1(Hexs[1]),  
    .D2(Hexs[2]),  
    .D3(Hexs[3]),  
    .LE(LE0),  
    .point(point[0]),  
    .a(seg_t[0]),  
    .b(seg_t[1]),  
    .c(seg_t[2]),  
    .d(seg_t[3]),  
    .e(seg_t[4]),  
    .f(seg_t[5]),  
    .g(seg_t[6]),  
    .p(seg_t[7])  
    );
```

```
MC14495_ZJU M1 (.D0(Hexs[4]),  
    .D1(Hexs[5]),  
    .D2(Hexs[6]),  
    .D3(Hexs[7]),  
    .LE(LE1),  
    .point(point[1]),  
    .a(seg_t[8]),  
    .b(seg_t[9]),  
    .c(seg_t[10]),  
    .d(seg_t[11]),  
    .e(seg_t[12]),  
    .f(seg_t[13]),  
    .g(seg_t[14]),  
    .p(seg_t[15])  
    );
```

```
MC14495_ZJU M2 (.D0(Hexs[8]),  
    .D1(Hexs[9]),  
    .D2(Hexs[10]),  
    .D3(Hexs[11]),  
    .LE(LE2),  
    .point(point[2]),  
    .a(seg_t[16]),  
    .b(seg_t[17]),  
    .c(seg_t[18]),  
    .d(seg_t[19]),  
    .e(seg_t[20]),  
    .f(seg_t[21]),  
    .g(seg_t[22]),  
    .p(seg_t[23])
```



```

        );
MC14495_ZJU M3 (.D0(Hexs[12]),
        .D1(Hexs[13]),
        .D2(Hexs[14]),
        .D3(Hexs[15]),
        .LE(LE3),
        .point(point[3]),
        .a(seg_t[24]),
        .b(seg_t[25]),
        .c(seg_t[26]),
        .d(seg_t[27]),
        .e(seg_t[28]),
        .f(seg_t[29]),
        .g(seg_t[30]),
        .p(seg_t[31])
        );
MC14495_ZJU M4 (.D0(Hexs[16]),
        .D1(Hexs[17]),
        .D2(Hexs[18]),
        .D3(Hexs[19]),
        .LE(LE4),
        .point(point[4]),
        .a(seg_t[32]),
        .b(seg_t[33]),
        .c(seg_t[34]),
        .d(seg_t[35]),
        .e(seg_t[36]),
        .f(seg_t[37]),
        .g(seg_t[38]),
        .p(seg_t[39])
        );
MC14495_ZJU M5 (.D0(Hexs[20]),
        .D1(Hexs[21]),
        .D2(Hexs[22]),
        .D3(Hexs[23]),
        .LE(LE5),
        .point(point[5]),
        .a(seg_t[40]),
        .b(seg_t[41]),
        .c(seg_t[42]),
        .d(seg_t[43]),
        .e(seg_t[44]),
        .f(seg_t[45]),
        .g(seg_t[46]),

```

```

        .p(seg_t[47])
    );
MC14495_ZJU M6 (.D0(Hexs[24]),
    .D1(Hexs[25]),
    .D2(Hexs[26]),
    .D3(Hexs[27]),
    .LE(LE6),
    .point(point[6]),
    .a(seg_t[48]),
    .b(seg_t[49]),
    .c(seg_t[50]),
    .d(seg_t[51]),
    .e(seg_t[52]),
    .f(seg_t[53]),
    .g(seg_t[54]),
    .p(seg_t[55])
);
MC14495_ZJU M7 (.D0(Hexs[28]),
    .D1(Hexs[29]),
    .D2(Hexs[30]),
    .D3(Hexs[31]),
    .LE(LE7),
    .point(point[7]),
    .a(seg_t[56]),
    .b(seg_t[57]),
    .c(seg_t[58]),
    .d(seg_t[59]),
    .e(seg_t[60]),
    .f(seg_t[61]),
    .g(seg_t[62]),
    .p(seg_t[63])
);

generate
    genvar i;
    for(i=0;i<64;i=i+1)
        begin
            assign SEG_TXT[i]=seg_t[63-i];
        end
    endgenerate

endmodule

```