至简设计系列_矩阵按键检测

--作者：肖肖肖

1.1 总体设计

1.1.1 概述

在键盘中按键数量较多时，为了减少 I/O 口的占用，通常将按键排列成矩阵形式。在矩阵式键盘中，每条水平线和垂直线在交叉处不直接连通，而是通过一个按键加以连接。这样，一个端口（如 P1 口）就可以构成 4*4=16 个按键，比之直接将端口线用于键盘多出了一倍，而且线数越多，区别越明显，比如再多加一条线就可以构成 20 键的键盘，而直接用端口线则只能多出一键（9 键）。由此可见，在需要的键数比较多时，采用矩阵法来做键盘是合理的。

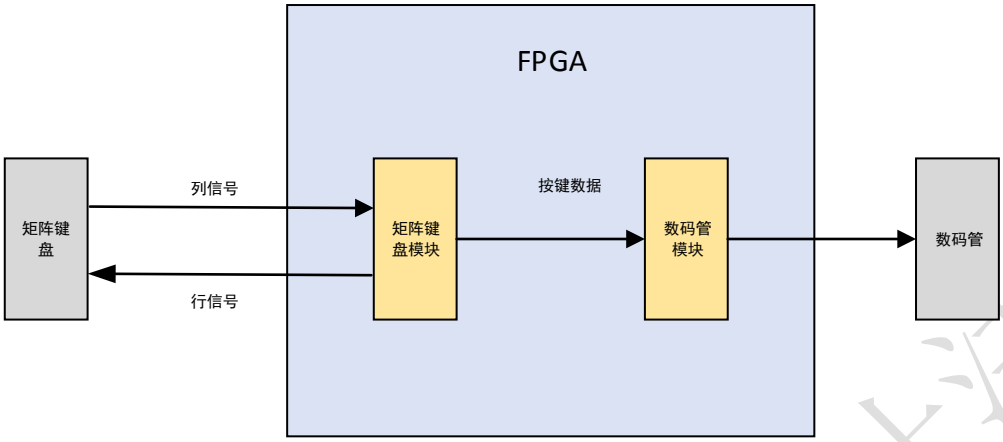
1.1.2 设计目标

完成矩阵键盘的扫描检测程序，具体功能要求如下：

1. 运用逐行扫描的方法进行按键监测；
2. 每行扫描的时间不少于 20ms，滤除抖动；
3. 检测到有按键按下之后，消抖时间 20ms；
4. 输出信号 key_vld 持续一拍即可；
5. 输出信号 key_out 表示 16 个按键，并在数码管上显示对应数值；

1.1.3 系统结构框图

系统结构框图如下图一所示：



图一

1.1.4 模块功能

- **矩阵键盘模块实现功能**

1、将外来异步信号打两拍处理，将异步信号同步化。
2、实现 20ms 按键消抖功能。
3、实现矩阵键盘的按键检测功能，并输出有效按键信号。

- **数码管显示模块实现功能**

1、 对接收到的按键数据进行译码并显示。

1.1.5 顶层信号

信号名	I/O	位宽	定义
clk	I	1	系统工作时钟 50M
rst_n	I	1	系统复位信号，低电平有效
Key_col	I	4	4 位矩阵键盘列信号，最高位表示矩阵键盘往右数第四列，默认高电平
Key_row	O	4	4 位矩阵键盘行信号，最高位表示矩阵键盘往下数第四行
Segment	O	8	8 位数码管段选信号

Seg_sel	O	2	2 位数码管位选信号

1.1.6 参考代码

下面是使用工程的顶层代码：

```
1.  module top(
2.      clk         ,
3.      rst_n       ,
4.      key_col     ,
5.      key_row     ,
6.      segment     ,
7.      seg_sel
8.      );
9.
10.     parameter      DATA_W =       4;
11.
12.     input              clk    ;
13.     input              rst_n  ;
14.     input   [3:0]      key_col;
15.
16.     output  [3:0]      key_row;
17.     output  [7:0]      segment;
18.     output  [1:0]      seg_sel;
19.
20.     wire     [3:0]      key_row;
21.     wire     [7:0]      segment;
22.     wire     [1:0]      seg_sel;
23.
24.     wire    [DATA_W-1:0]    key_out      ;
25.     wire                    key_vld      ;
26.     wire    [DATA_W-1:0]    segment_data ;
27.
28. key_scan    u1(
29.     .clk        (   clk              ),
30.     .rst_n      (   rst_n            ),
31.     .key_col    (   key_col          ),
32.     .key_row    (   key_row          ),
33.     .key_out    (   key_out          ),
34.     .key_vld    (   key_vld          )
```

```
35.     );
36.
37. seg_disp    u3(
38.     .clk              (  clk            ),
39.     .rst_n            (  rst_n          ),
40.     .data_in          (  key_out          ),
41.     .key_en           (  key_vld          ),
42.     .segment          (  segment          ),
43.     .seg_sel          (  seg_sel          )
44. );
45.
46.     endmodule
```

1.2 矩阵键盘模块设计

1.2.1 接口信号

信号名	I/O	位宽	定义
clk	I	1	系统工作时钟 50M
rst_n	I	1	系统复位信号，低电平有效
key_col	I	4	4 位矩阵键盘列信号，最高位表示矩阵键盘往右数第四列，默认高电平
key_row	O	4	4 位矩阵键盘行信号，最高位表示矩阵键盘往下数第四行
key_out	O	4	矩阵按键数据
key_vld	O	1	按键有效指示信号

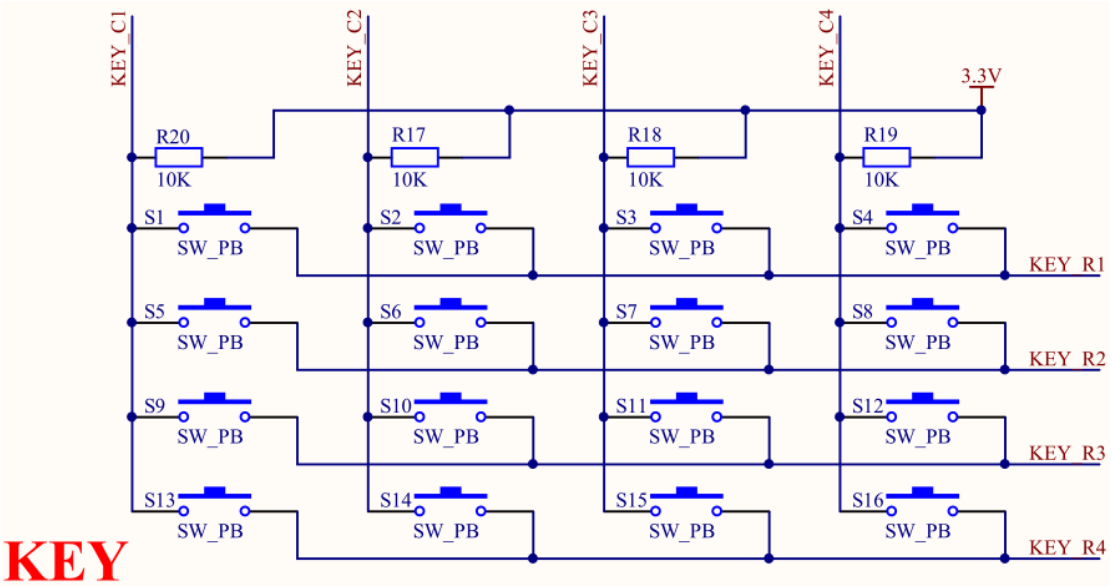
1.2.2 设计思路

- **行扫描法原理**

开发板上为 4*4 矩阵键盘：默认 4 条列线上来高电平，4 条行线默认接高电平。列线 KEY_C1 ~ KEY_C4 分别接有 4 个上拉电阻到正电源 +3.3 V，并把列线 KEY_C1~KEY_C4 设置为输入线，

行线 KEY_R1~KEY_R4 设置为输出线。4 根行线和 4 根列线形成 16 个相交点。

如下图所示：



确认矩阵键盘上哪个按键被按下有多同方法，其中行扫描法又称为逐行（或列）扫描查询法，是一种最常用的按键识别方法。

1. 判断键盘中有无键按下：将全部行线 KEY_R1~KEY_R4 置低电平，然后检测列线 KEY_C1~KEY_C4 的状态。只要有一列的电平为低，则表示键盘中有键被按下，而且闭合的键位于低电平线与 4 根行线相交叉的 4 个按键之中。若所有列线均为高电平，则键盘中无键按下。
2. 判断闭合键所在的位置：在确认有键按下后，即可进入确定具体闭合键的过程。其方法是：依次将行线置为低电平，即在置某根行线为低电平时，其它线为高电平。在确定某根行线位置为低电平后，再逐行检测各列线的电平状态。若某列为低，则该列线与置为低电平的行线交叉处的按键就是闭合的按键。

➢ 打拍操作

输入的 key_col 是异步信号，所以要进行打两拍操作，将异步信号 key_col 同步化，并防止亚稳态。

设计代码如下：

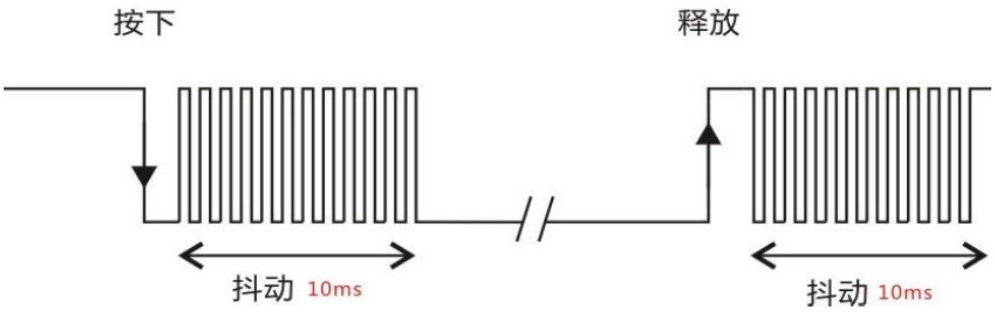
```
input   [3:0]          key_col     ;

reg     [3:0]          key_col_ff0      ;
reg     [3:0]          key_col_ff1      ;

always  @(posedge clk or negedge rst_n)begin
    if(rst_n==1'b0)begin
        key_col_ff0 <= 4'b1111;
        key_col_ff1 <= 4'b1111;
    end
```

```
11.     else begin
12.         key_col_ff0 <= key_col    ;
13.         key_col_ff1 <= key_col_ff0;
14.     end
15. end
```

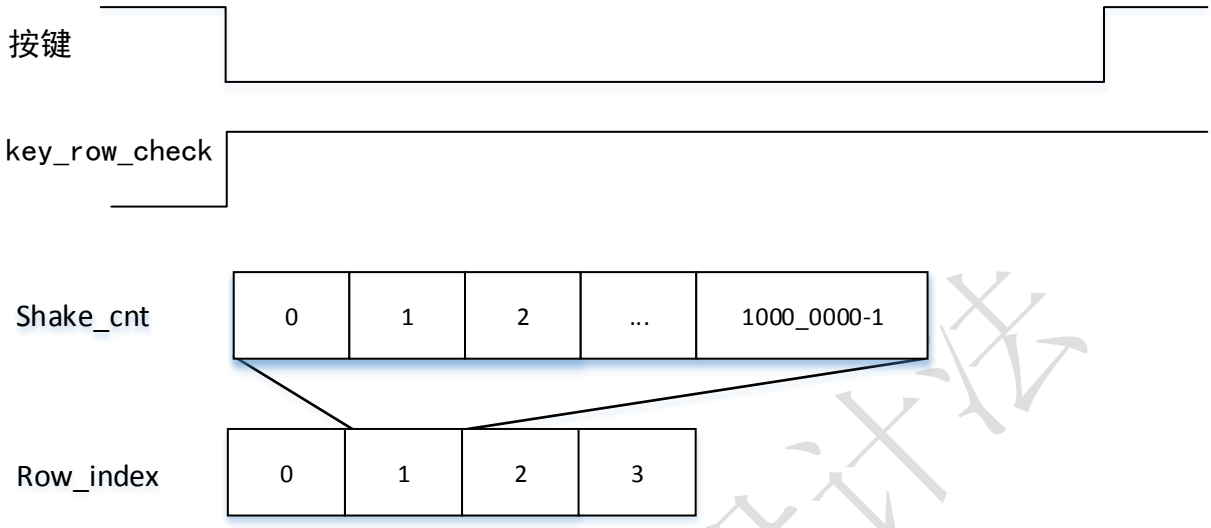
➢ 按键消抖

对于按键和触摸屏等机械设备来说，都存在一个固有问题，那就是“抖动”，按键从最初接通到稳定接通要经过数毫秒，其间可能发生多次“接通-断开”这样的毛刺。如果不进行处理，会使系统识别到抖动信号而进行不必要的反应，导致模块功能不正常，为了避免这种现象的产生，需要进行按键消抖的操作。

软件方法消抖，即检测出键闭合后执行一个延时程序，抖动时间的长短由按键的机械特性决定，一般为 5ms～20ms，让前沿抖动消失后再一次检测键的状态，如果仍保持闭合状态电平，则确认按下按键操作有效。当检测到按键释放后，也要给 5ms～20ms 的延时，待后沿抖动消失后才能转入该键的处理程序。



由于按键按下去的时间一般都会大于 20ms，为了达到不管按键按下多久，都视为按下一次的效果，提出以下计数器架构，如下图所示：



消抖计数器 shake_cnt：用于计算 20ms 的时间，加一条件为 key_col_ff1 != 4'hf || key_row_check==1，表示当某个按键按下或者进行行扫描时就开始计数；数到 1,000,000 下，表示数到 20ms 就结束。

行扫描计数器 row_index：用于区分扫描的行，加一条件为 key_row_check && end_shake_cnt，表示当处于行扫描状态并且每行消抖 20ms 后，开始扫描下一行；数到 4 下，表示 4 行按键扫描完了。

按键：表示有无按键按下，没被按下时为高电平，按下后为低电平。

行扫描指示信号 key_row_check：该信号为高电平，指示当前处于行扫描状态。

矩阵键盘列信号 key_col_ff1：4bit 位宽的矩阵键盘列信号，最高位表示矩阵键盘往右数第四列，默认信号为 key_col_ff1 = 4'hf，否则表示该信号低电平对应位的列有按键按下。

1.2.3 参考代码

```
16. module key_scan(
17.     clk    ,
18.     rst_n  ,
19.     key_col,
20.     key_row,
21.     key_out,
22.     key_vld
23.     );
24.
25.     parameter       DATA_W      =   4           ;
26.     parameter       TIME_20MS   =   1_000_000   ;
27.
```

```
28.     input                   clk         ;
29.     input                   rst_n       ;
30.     input   [3:0]           key_col     ;
31.
32.     output  [3:0]           key_row     ;
33.     output                  key_vld     ;
34.     output  [DATA_W-1:0]    key_out     ;
35.
36.     reg     [3:0]           key_row     ;
37.     reg                     key_vld     ;
38.     reg     [DATA_W-1:0]    key_out     ;
39.
40.     reg     [3:0]           key_col_ff0     ;
41.     reg     [3:0]           key_col_ff1     ;
42.     reg                     key_col_check   ;
43.     reg     [21:0]          shake_cnt       ;
44.     wire                    add_shake_cnt   ;
45.     wire                    end_shake_cnt   ;
46.     reg     [1:0]           key_col_get     ;
47.     reg                     key_row_check   ;
48.     reg     [1:0]           row_index       ;
49.     wire                    add_row_index   ;
50.     wire                    end_row_index   ;
51.
52.
53. always  @(posedge clk or negedge rst_n)begin
54.     if(rst_n==1'b0)begin
55.         key_col_ff0 <= 4'b1111;
56.         key_col_ff1 <= 4'b1111;
57.     end
58.     else begin
59.         key_col_ff0 <= key_col    ;
60.         key_col_ff1 <= key_col_ff0;
61.     end
62. end
63.
64. always  @(posedge clk or negedge rst_n)begin
65.     if(rst_n==1'b0)begin
66.         key_col_check <= 1'b0;
67.     end
68.     else if(key_col_ff1 !=4'hf && end_shake_cnt)begin
69.         key_col_check <= 1'b1;
70.     end
```

```
71.     else if(key_col_ff1==4'hf)begin
72.         key_col_check <= 1'b0;
73.     end
74. end
75.
76.
77. always @(posedge clk or negedge rst_n) begin
78.     if (rst_n==0) begin
79.         shake_cnt <= 0;
80.     end
81.     else if(add_shake_cnt) begin
82.         if(end_shake_cnt)
83.             shake_cnt <= 0;
84.         else
85.             shake_cnt <= shake_cnt+1 ;
86.    end
87. end
88. assign add_shake_cnt = key_col_ff1 !=4'hf || key_row_check==1;
89. assign end_shake_cnt = add_shake_cnt  && shake_cnt == TIME_20MS-1 ;
90.
91.
92. always  @(posedge clk or negedge rst_n)begin
93.     if(rst_n==1'b0)begin
94.         key_col_get <= 0;
95.     end
96.     else if(key_col_check) begin
97.         if(key_col_ff1==4'b1110)
98.             key_col_get <= 0;
99.         else if(key_col_ff1==4'b1101)
100.            key_col_get <= 1;
101.        else if(key_col_ff1==4'b1011)
102.            key_col_get <= 2;
103.        else if(key_col_ff1==4'b0111)
104.            key_col_get <= 3;
105.    end
106.end
107.
108.always  @(posedge clk or negedge rst_n)begin
109.    if(rst_n==1'b0)begin
110.        key_row_check <= 0;
111.    end
112.    else if(key_col_check)begin
113.        key_row_check <= 1;
```

```
114.    end
115.    else if(key_vld)begin
116.        key_row_check <= 0;
117.    end
118.end
119.
120.
121.always @(posedge clk or negedge rst_n) begin
122.    if (rst_n==0) begin
123.        row_index <= 0;
124.    end
125.    else if(add_row_index) begin
126.        if(end_row_index)
127.            row_index <= 0;
128.        else
129.            row_index <= row_index+1 ;
130.   end
131.end
132.assign add_row_index = key_row_check && end_shake_cnt;
133.assign end_row_index = add_row_index  && row_index == 4-1 ;
134.
135.
136.always  @(posedge clk or negedge rst_n)begin
137.    if(rst_n==1'b0)begin
138.        key_row = 4'b0;
139.    end
140.    else if(key_row_check)begin
141.        key_row = ~(4'b0001 << row_index);
142.    end
143.    else begin
144.        key_row = 4'b0;
145.    end
146.end
147.
148.
149.always  @(posedge clk or negedge rst_n)begin
150.    if(rst_n==1'b0)begin
151.        key_vld <= 1'b0;
152.    end
153.    else if(key_row_check && key_col_ff1[key_col_get]==1'b0 &&
    key_col_check==0 )begin
154.        key_vld <= 1'b1;
155.    end
```

```
156.    else begin
157.        key_vld <= 1'b0;
158.    end
159.end
160.
161.
162.always  @(posedge clk or negedge rst_n)begin
163.    if(rst_n==1'b0)begin
164.        key_out <= 4'd0;
165.    end
166.    else if(key_vld)begin
167.        key_out <= {row_index,key_col_get};
168.    end
169.    else begin
170.        key_out <= 4'd0;
171.    end
172.end
173.
174.endmodule
```

1.3 数码管显示模块设计

1.3.1 接口信号

信号名	I/O	位宽	定义
clk	I	1	系统工作时钟 50M
rst_n	I	1	系统复位信号，低电平有效
data_in	I	4	矩阵按键数据
key_en	I	1	按键有效指示信号
segment	O	8	8 位数码管段选信号
seg_sel	O	2	2 位数码管位选信号

1.3.2 设计思路

在前面的案例中已经有数码管显示的介绍，所以这里不在过多介绍，详细介绍请看下方链接：
http://fpgabbs.com/forum.php?mod=viewthread&tid=399

1.3.3 参考代码

```
1.  module seg_disp(
2.      clk         ,
3.      rst_n       ,
4.      data_in     ,
5.      key_en      ,
6.      segment     ,
7.      seg_sel
8.  );
9.
10. parameter   ZERO          =   8'b0000_0011        ;
11. parameter   ONE           =   8'b1001_1111        ;
12. parameter   TWO           =   8'b0010_0101        ;
13. parameter   THREE         =   8'b0000_1101        ;
14. parameter   FOUR          =   8'b1001_1001        ;
15. parameter   FIVE          =   8'b0100_1001        ;
16. parameter   SIX           =   8'b0100_0001        ;
17. parameter   SEVEN         =   8'b0001_1111        ;
18. parameter   EIGHT         =   8'b0000_0001        ;
19. parameter   NINE          =   8'b0000_1001        ;
20.
21. input            clk           ;
22. input            rst_n         ;
23. input    [3:0]    data_in       ;
24. input            key_en        ;
25. output   [7:0 ]   segment       ;
26. output   [1:0 ]   seg_sel       ;
27.
28. reg      [7:0 ]   segment       ;
29. reg      [1:0 ]   seg_sel       ;
30. reg      [10:0]   delay         ;
31. reg      [1:0 ]   delay_time    ;
32. wire             add_delay_time ;
33. wire             end_delay_time ;
34. wire             add_delay      ;
```

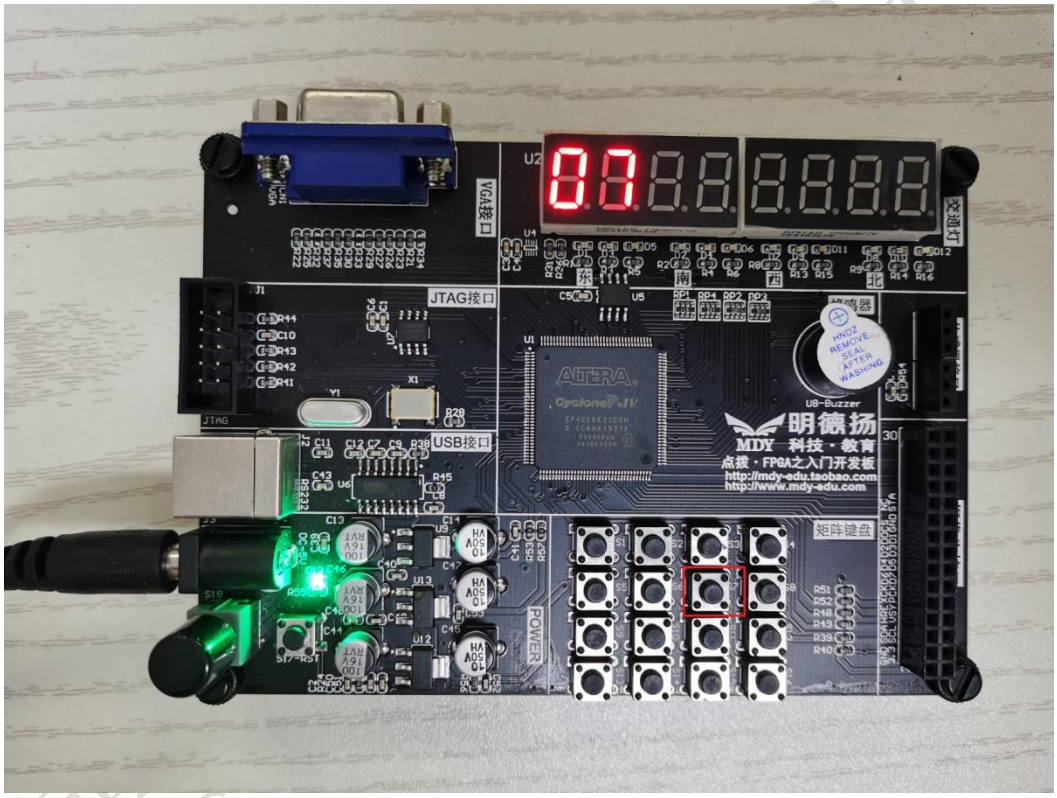
```
35. wire             end_delay      ;
36. reg      [4:0 ]   segment_tmp    ;
37. reg      [3:0 ]   segment_data   ;
38.
39.
40. always @(posedge clk or negedge rst_n) begin
41.     if (rst_n==0) begin
42.         delay <= 0;
43.     end
44.     else if(add_delay) begin
45.         if(end_delay)
46.             delay <= 0;
47.         else
48.             delay <= delay+1 ;
49.    end
50. end
51. assign add_delay = 1;
52. assign end_delay = add_delay  && delay == 2000-1 ;
53.
54.
55. always @(posedge clk or negedge rst_n) begin
56.     if (rst_n==0) begin
57.         delay_time <= 0;
58.     end
59.     else if(add_delay_time) begin
60.         if(end_delay_time)
61.             delay_time <= 0;
62.         else
63.             delay_time <= delay_time+1 ;
64.    end
65. end
66. assign add_delay_time = end_delay;
67. assign end_delay_time = add_delay_time  && delay_time == 2-1 ;
68.
69.
70. always  @(posedge clk or negedge rst_n)begin
71.     if(rst_n==1'b0)begin
72.         segment_data <= 4'd0;
73.     end
74.     else if(key_en)begin
75.         segment_data <= data_in;
76.     end
77. end
```

```
78.
79. always  @(posedge clk or negedge rst_n)begin
80.     if(rst_n==1'b0)begin
81.         segment_tmp <= 4'd0;
82.     end
83.     else if(add_delay_time  && delay_time == 1-1)begin
84.         segment_tmp <= (segment_data+1)%10;
85.     end
86.     else if(end_delay_time)begin
87.         segment_tmp <= (segment_data+1)/10;
88.     end
89. end
90.
91.
92. always  @(posedge clk or negedge rst_n)begin
93.     if(rst_n==1'b0)begin
94.         segment <= ZERO;
95.     end
96.     else begin
97.         case(segment_tmp)
98.             4'd0:segment <= ZERO;
99.             4'd1:segment <= ONE  ;
100.            4'd2:segment <= TWO  ;
101.            4'd3:segment <= THREE;
102.            4'd4:segment <= FOUR ;
103.            4'd5:segment <= FIVE ;
104.            4'd6:segment <= SIX  ;
105.            4'd7:segment <= SEVEN;
106.            4'd8:segment <= EIGHT;
107.            4'd9:segment <= NINE ;
108.            default:begin
109.                segment <= segment;
110.            end
111.        endcase
112.    end
113.end
114.
115.
116.always  @(posedge clk or negedge rst_n)begin
117.    if(rst_n==1'b0)begin
118.        seg_sel <= 2'b11;
119.    end
120.    else begin
```

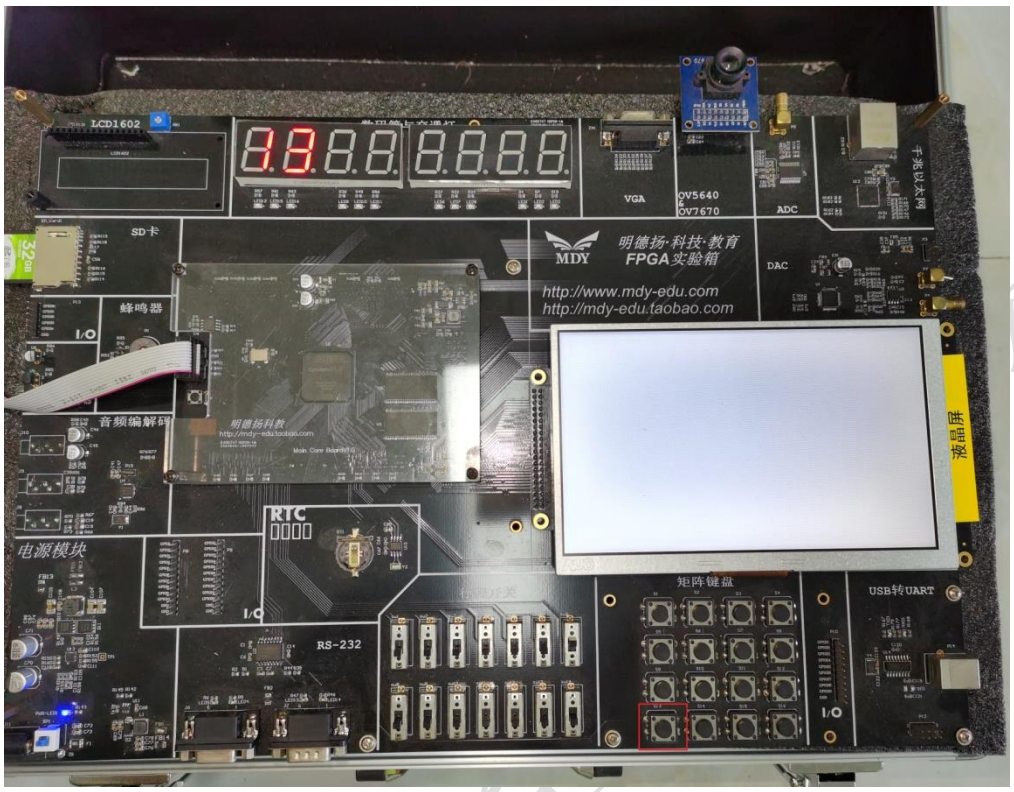
```
121.        seg_sel <= ~(2'b1<<delay_time);
122.    end
123.end
124.
125.
126.endmodule
```

1.4 效果和总结

- 下图是该工程在 **db603** 开发板上的现象——按下按键 **s7**



➢ 下图是该工程在 ms980 试验箱上的现象——按下按键 s13



由于该项目的上板现象是按下矩阵按键，数码管显示对应的按键号，想观看完整现象的朋友可以看一下现象演示的视频。

感兴趣的朋友也可以访问明德扬论坛（http://www.fpgabbs.cn/）进行 FPGA 相关工程设计学习，也可以看一下我们往期的文章：

《基于 FPGA 的密码锁设计》
《波形相位频率可调 DDS 信号发生器》
《基于 FPGA 的曼彻斯特编码解码设计》
《基于 FPGA 的出租车计费系统》
《数电基础与 Verilog 设计》
《基于 FPGA 的频率、电压测量》
《基于 FPGA 的汉明码编码解码设计》
《关于锁存器问题的讨论》
《阻塞赋值与非阻塞赋值》
《参数例化时自动计算位宽的解决办法》

1.5 公司简介

明德扬是一家专注于 FPGA 领域的专业性公司，公司主要业务包括开发板、教育培训、项目承

接、人才服务等多个方向。

点拨开发板——学习 FPGA 的入门之选。

MP801 开发板——千兆网、ADDA、大容量 SDRAM 等，学习和项目需求一步到位。

网络培训班——不管时间和空间，明德扬随时在你身边，助你快速学习 FPGA。

周末培训班——明天的你会感激现在的努力进取，升职加薪明德扬来助你。

就业培训班——七大企业级项目实训，获得丰富的项目经验，高薪就业。

专题课程——高手修炼课：提升设计能力；实用调试技巧课：提升定位和解决问题能力；FIFO 架构设计课：助你快速成为架构设计师；时序约束、数字信号处理、PCIE、综合项目实践课等你来选。

项目承接——承接企业 FPGA 研发项目。

人才服务——提供人才推荐、人才代培、人才派遣等服务。