中国科学技术大学计算机学院

《数字电路实验》报告

实验题目：实验 08 信号处理及有限状态机

学生姓名：余致远

学生学号：PB18111740

完成日期：2019 年 11 月 29 日

计算机实验教学中心制

2019 年 09 月

【实验目的】

进一步熟悉 FPGA 开发的整体流程

掌握几种常见的信号处理技巧

掌握有限状态机的设计方法

能够使用有限状态机设计功能电路

【实验环境】

PC 一台

Windows 或 Linux 操作系统

Logisim

Vivado 工具

Nexys4DDR 开发板或 FPGAOL 实验平台

vlab.ustc.edu.cn

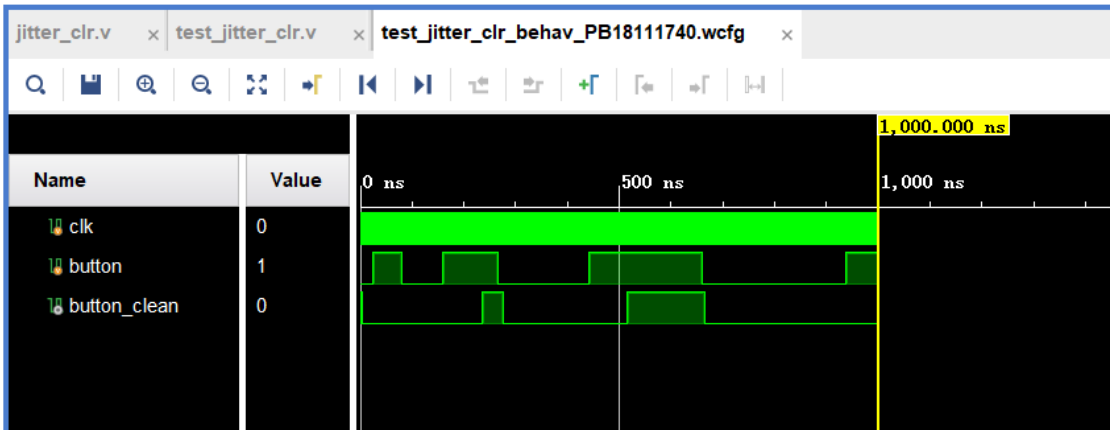
【实验过程】

Step1：信号去抖动

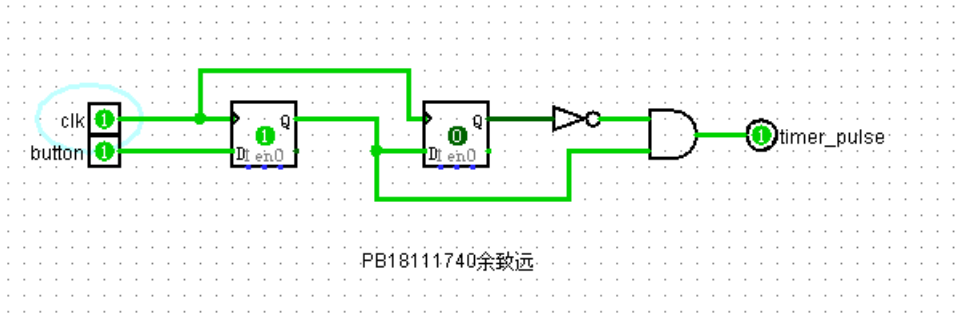
```
//PB18111740余致远
module jitter_clr(
    input clk,
    input button,
    output button_clean
);

reg [3:0] cnt;
always@(posedge clk)
begin
    if(button==1'b0)
        cnt <= 4'h0;
    else if(cnt<4'h8)
        cnt <= cnt + 1'b1;
end
assign button_clean = cnt[3];
endmodule
```

仿真文件：

```
//PB18111740余致远
module test_jitter_clr(

    );
reg clk;
reg button;
wire button_clean;
jitter_clr jitter_clr(clk,button,button_clean);
initial clk=0;
always #5 clk=~clk;
initial
begin
        button = 0;
    #24 button = 1;
    #56 button = 0;
    #80 button = 1;
    #106 button = 0;
    #175 button = 1;
    #218 button = 0;
    #278 button = 1;
    #900 button = 0;
    #918 button = 1;
    #926 button = 0;
    #1000 $finish;
end
endmodule
```

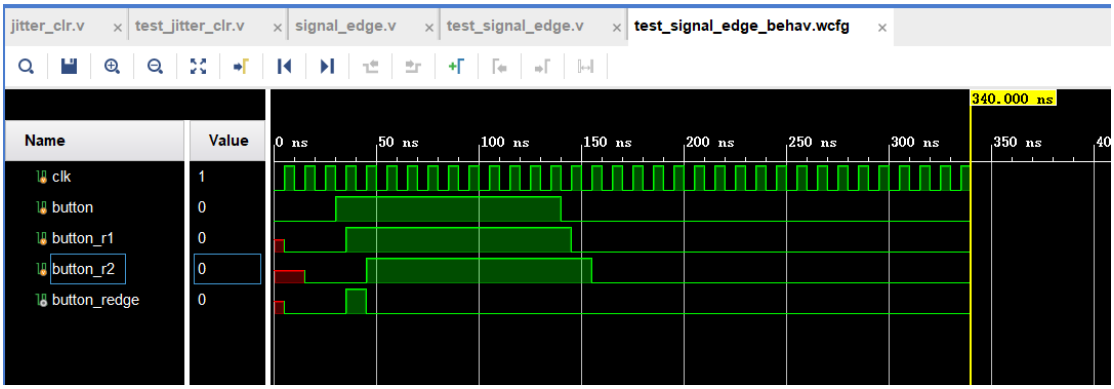


Step2：取信号边沿技巧

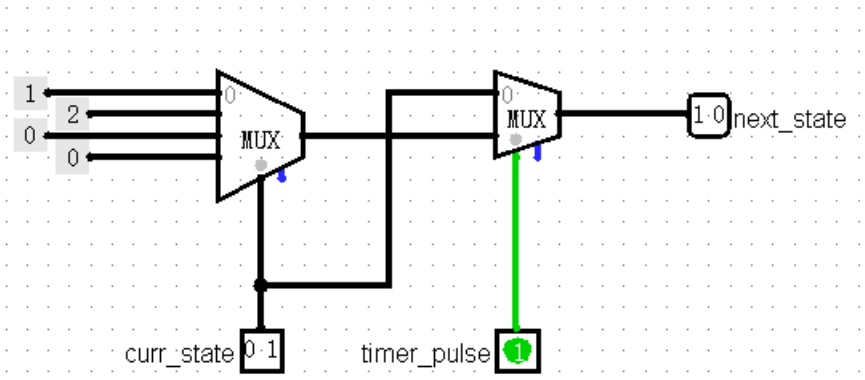



```
//PB18111740余致远
module signal_edge(
    input clk,
    input button,
    output button_redge);

reg button_r1,button_r2;
always@(posedge clk)
    button_r1 <= button;
always@(posedge clk)
    button_r2 <= button_r1;
assign button_redge = button_r1 & (~button_r2);
endmodule
```

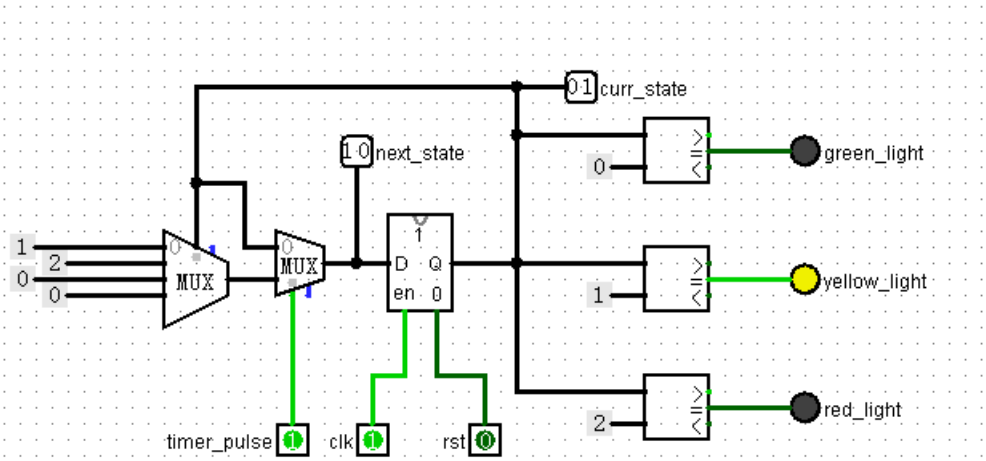
```
//PB18111740余致远
module test_signal_edge(

    );

reg clk;
reg button;
wire button_redge;
signal_edge signal_edge(clk,button,button_redge);
initial clk=0;
always #5 clk=~clk;
initial
begin
        button = 0;
    #30 button = 1;
    #110 button = 0;
    #200 $finish;
end
endmodule
```



Step3：有限状态机介绍



PB18111740余致远

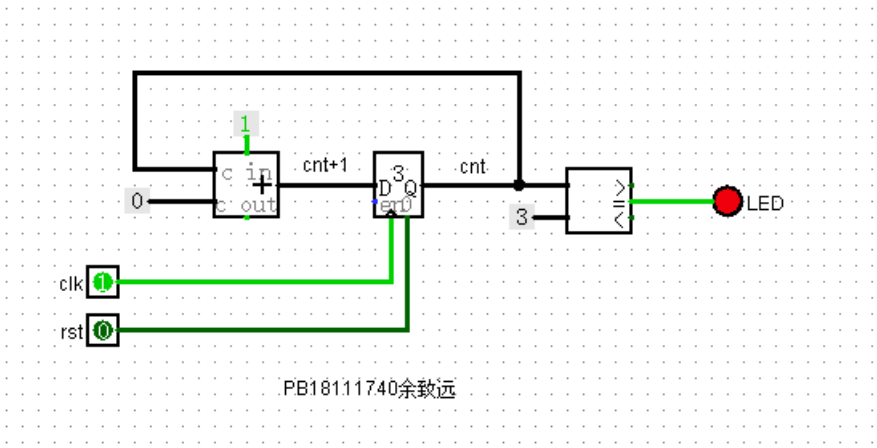


PB18111740余致远

Step4：有限状态机 Verilog 实现

```
module traffic_ctrl(
    input clk,
    input rst,
    input timer_pulse,
    output green_light
);

parameter C_PASS = 2'b00;
parameter C_TRANS = 2'b01;
parameter C_STOP = 2'b10;
reg [1:0] curr_state;
reg [1:0] next_state;
//有限状态机第一部分
always@(*)
begin
    if(timer_pulse)
    begin
        case(curr_state)
            C_PASS: next_state = C_TRANS;
            C_TRANS:next_state = C_STOP;C_STOP: next_state = C_PASS;
            default:next_state = C_PASS;
        endcase
    end
    else
        next_state = curr_state;
end
//有限状态机第二部分
always@(posedge clk or posedge rst)
begin
    if(rst)
    curr_state <= C_PASS;
    else
    curr_state <= next_state;
end
//有限状态机第三部分,各输出信号的赋值都应放在此部分
assign green_light = (curr_state==C_PASS)? 1'b1 : 1'b0;
assign yellow_light = (curr_state==C_TRANS)? 1'b1 : 1'b0;
assign red_light = (curr_state==C_STOP)? 1'b1 : 1'b0;
endmodule
```

Step5：有限状态机的另一形式



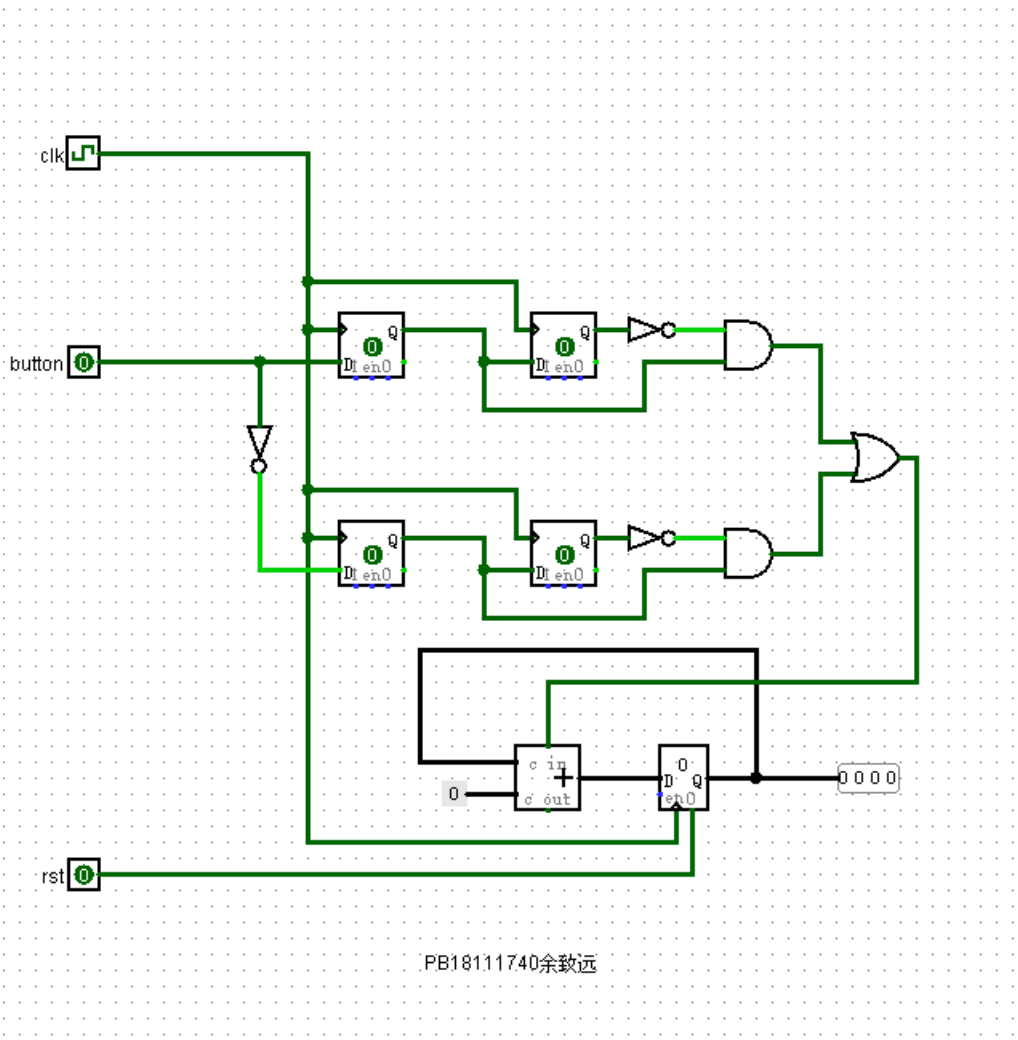
【实验练习】

题目 1.在不改变电路功能和行为的前提下， 将前面 Step5 中的代码改写成三段式有限状态机的形式， 写出完整的 Verilog 代码。

```
22  //PB18111740余致远
23  module T1(
24      input clk,rst,
25      output led
26      );
27  reg [1:0] curr_state;
28  reg [1:0] next_state;
29  parameter C_PASS   = 2'b00;
30  parameter C_TRANS  = 2'b01;
31  parameter C_STOP   = 2'b10;
32  //有限状态机第一部分
33  always@(*)
34  begin
35      case(curr_state)
36          C_PASS:     next_state = C_TRANS;
37          C_TRANS:    next_state = C_STOP;
38          C_STOP:     next_state = C_PASS;
39          default:    next_state = C_PASS;
40      endcase
41  end
42  //有限状态机第二部分
43  always@(posedge clk or posedge rst)
44  begin
45      if(rst)
46      curr_state <= C_PASS;
47      else
48      curr_state <= next_state;
```

```
49  end
50  //有限状态机第三部分
51  assign led = (curr_state == C_STOP) ? 1'b1 : 1'b0;
52  endmodule
53
```

题目 2. 请在 Logisim 中设计一个 4bit 位宽的计数器电路，如下图所示，clk 信号为计数器时钟，复位时（rst==1）计数值为 0，在输入信号 sw 电平发生变化时，计数值 cnt 加 1，即在 sw 信号上升沿时刻和下降沿时刻各触发一次计数操作，其余时刻计数器保持不变。



通过对正、反信号分别进行边沿检测来实现功能。

题目 3. 设计一个 8 位的十六进制计数器，时钟采用板载的 100MHz 时钟，通过按键控制计数模式，一个按键控制累加，每按键一次计数器加一，另一按键控制递减，每按下一次计数器减一，按键按下的瞬间触发操作，一个按键按下未抬起，不会影响到另一按键触发计数器。计数值用数码管显示，其复位值为“1F”。

```
//PB18111740 余致远
module T3(
    input clk,rst,
    input btnl,btnr,
    output reg [7:0]an,
    output [7:0] seg
    );

reg [19:0] cnt;
reg [3:0] data;
reg [7:0] count;
wire btnl_cln,btnr_cln;
wire btnl_redge,btnr_redge;

//边沿检测
jitter_clr jitter_clr(clk,btnl,btnl_cln);
jitter_clr(clk,btnr,btnr_cln);
signal_edge signal_edge(clk,btnl_cln,btnl_redge);
signal_edge(clk,btnr_cln,btnr_redge);

always@(posedge clk) //100MHz 时钟
begin
    if(rst)
    begin
        count <= 8'h1F;
        cnt   <= 20'h0;
    end
    else cnt <= cnt + 20'b1;
    if  (btnl_redge) count <= count + 8'b1;
    else if  (btnr_redge) count <= count - 8'b1;
end
```

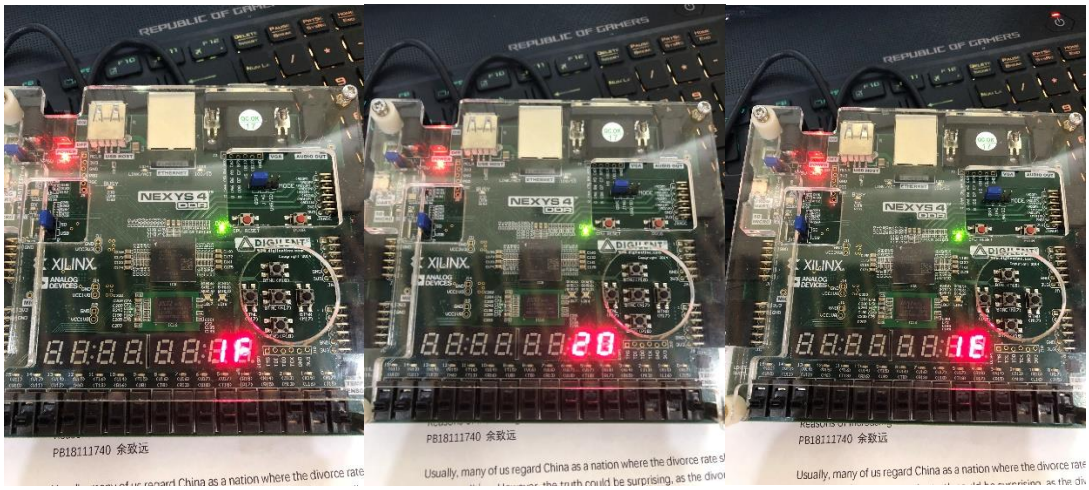
```
always@(posedge clk) //分时复用
begin
    case(cnt[19])
        2'h0:
            begin
                an <= 8'b1111_1110;
                data <= count[3:0];
            end
        2'h1:
            begin
                an <= 8'b1111_1101;
                data <= count[7:4];
            end
        default: an <= 8'b1111_1110;
    endcase
end

//ROM 控制 LED
dist_mem_gen_0 dist_mem_gen_0(
.a (data),
.spo (seg));

Endmodule
```

下面是去毛刺和边沿检测代码

```
//PB18111740余致远
module jitter_clr(
    input clk,
    input button,
    output button_clean
);

reg [17:0] cnt;
always@(posedge clk)
begin
    if(button==1'b0)
        cnt <= 18'h0;
    else if(cnt<18'b1111_1111_1111_1111_11)
        cnt <= cnt + 1'b1;
end
assign button_clean = cnt[17];
endmodule
```

```
//PB18111740余致远
module signal_edge(
    input clk,
    input button,
    output button_redge);

reg button_r1,button_r2;
always@(posedge clk)
    button_r1 <= button;
always@(posedge clk)
    button_r2 <= button_r1;
assign button_redge = button_r1 & (~button_r2);
endmodule
```

下面是具体实现情况。复位值为 1F。



题目 4. 使用有限状态机设计一个序列检测电路，并进行计数，当检测到输入序列为“1100”时，计数器加一，用一个数码管显示当前状态编码，一个数码管显示检测到目标序列的个数，用 4 个数码管显示最近输入的 4 个数值，用开关表示待输入的数值，按键每按下一次将输入一次开关状态，时钟采用板载的 100MHz 时钟。

要求画出状态跳转图，并在 FPGA 开发板上实现电路，例如当输入“0011001110011”时，目标序列个数应为 2，最近输入数值显示“0011”，状态机编码则与具体实现有关。

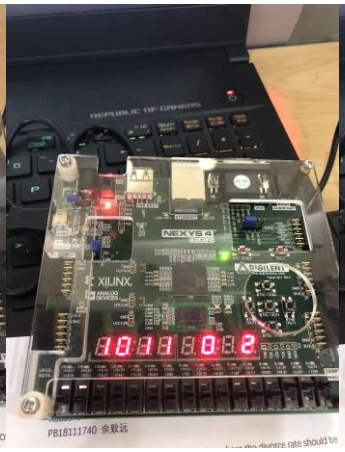
```
//PB18111740 余致远
module T4(
    input clk,rst,
    input btnl,
    input [15:0]sw,
    output reg [7:0]an,
    output [7:0] seg
    );
reg [19:0] cnt;
reg [3:0] data;
reg [3:0] seq;
reg [7:0]count;
```

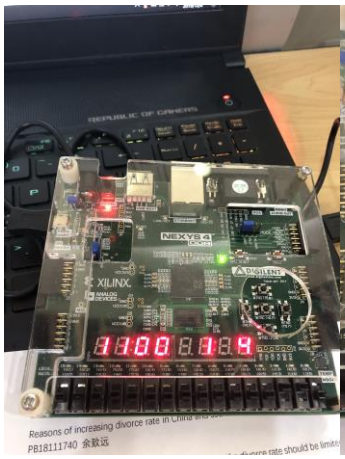
```
wire btnl_cln;
wire btnl_redge;
parameter C_0 = 4'd0;
parameter C_1 = 4'd1;
parameter C_11 = 4'd2;
parameter C_110 = 4'd3;
parameter C_1100 = 4'd4;
reg [3:0] curr_state;
reg [3:0] next_state;

jitter_clr jitter_clr(clk,btnl,btnl_cln);
signal_edge signal_edge(clk,btnl_cln,btnl_redge);

//有限状态机第一部分
always@(*)
begin
    if(btnl_redge)
    begin
        if(sw[0])
        begin
            case(curr_state)
                C_0     : next_state = C_1;
                C_1     : next_state = C_11;
                C_11    : next_state = C_11;
                C_110   : next_state = C_1;
                C_1100  : next_state = C_1;
            endcase
        end
        else
        begin
            case(curr_state)
                C_0     : next_state = C_0;
                C_1     : next_state = C_0;
                C_11    : next_state = C_110;
                C_110   : next_state = C_1100;
                C_1100  : next_state = C_0;
            endcase
        end
    end
end
//有限状态机第二部分
always@(posedge clk or posedge rst)
begin
    if(rst)
```

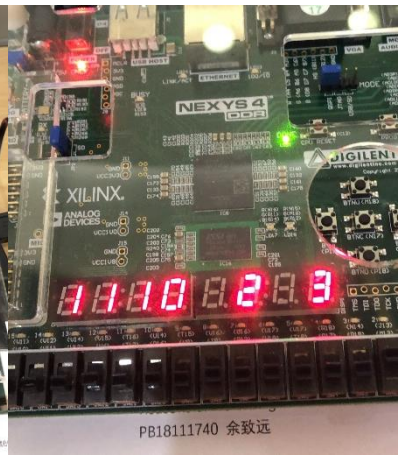
```
    curr_state <= C_0;
    else if(btnl_redge)
    curr_state <= next_state;
end
//有限状态机第三部分
always@(posedge clk or posedge rst)
begin
    if(rst)
    begin
        count <= 4'b0;
        seq <= 4'h0;
    end
    else if(btnl_redge)
    begin
        count <= (next_state == C_1100) ? count+4'b1 : count;
        seq[3:0] <= {seq[2:0],(sw[0]? 1'b1 : 1'b0)};
    end
end


always@(posedge clk) //100MHz 时钟
begin
    if(rst) cnt <= 20'h0;
    else cnt <= cnt + 20'b1;
end

always@(posedge clk) //分时复用
begin
    case(cnt[18:16])
        3'h0:   begin
                    an <= 8'b1111_1110;
                    data <= curr_state;
                end
        3'h1:   begin
                    an <= 8'b1111_1110;
                    data <= curr_state;
                end
        3'h2:   begin
                    an <= 8'b1111_1011;
                    data <= count[3:0];
                end
        3'h3:   begin
                    if (count[7:4])
                    begin
                        an <= 8'b1111_0111;
```

```
                    data <= count[7:4];
                end
            end
        3'h4:   begin
                    an <= 8'b1110_1111;
                    data <= seq[0];
                end
        3'h5:   begin
                    an <= 8'b1101_1111;
                    data <= seq[1];
                end
        3'h6:   begin
                    an <= 8'b1011_1111;
                    data <= seq[2];
                end
        3'h7:   begin
                    an <= 8'b0111_1111;
                    data <= seq[3];
                end
        default:begin
                    an <= 8'b1111_1110;
                    data <= curr_state;
                end
    endcase
end

dist_mem_gen_0 dist_mem_gen_0(
.a (data),
.spo (seg));

endmodule
```

下面是实现情况。左边的数码管是输入序列，最右一位是当前状态，中间的数字是当前检测到的序列数（16 进制，可显示两位）。各状态分别对应：0-前置不为“11”“110”的 0，1-前置“0”的 1，2-“11”，3-“110”，4-“1100”。在状态图中 1100 可以和 0 合并，为了表示检测到序列所以用 4 显示。





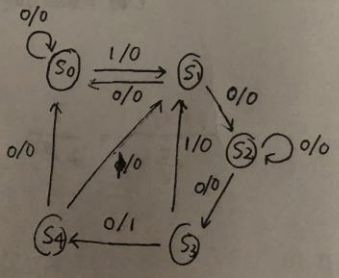


0	S_0
1	S_1
11	S_2
110	S_3
1100	S_4

S_{n+1} / z

S_n	A=0	A=1
S_0	S_0/0	S_1/0
S_1	S_0/0	S_2/0
S_2	S_3/0	S_2/0
S_3	S_4/1	S_1/0
S_4	S_0/0	S_1/0

S_4可以和S_0合并，为了体现出检测到序列，暂不合并



【总结与思考】

1.请总结本次实验的收获

本次实验的重点在于取边沿和去抖动两种信号处理技巧，以及有限状态机的设计。在题目三的操作过程中可以发现按键抖动对于计数的影响十分明显，因此在日常使用中去抖动是很有必要的过程。关于有限状态机的设计在数字电路课程中已经有涉及，在具体编程中则学习了将有限状态机分成比较规范化的三部分进行描述。，完成情况较好。

2.请评价本次实验的难易程度

实验难度适中，需要学习的原理比较复杂。

3.请评价本次实验的任务量

实验任务量较多，主要是调试程序消耗了一定时间。

4.请为本次实验提供改进建议

Step 中为了显示波形需要自行设计仿真文件来体现去抖动和取边沿电路的作用，对电路理解帮助不大。建议可以直接给出仿真文件。