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：信号处理及有限状态机

学生姓名：舒文炫

学生学号：PB18000029

完成日期：2021.12.8

计算机实验教学中心制

2020 年 09 月

【实验题目】

信号处理及有限状态机

【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】

- FPGAOL:fpgaol.ustc.edu.cn
- Logisim
- Vivado2020

【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。

三段式即显示组合逻辑部分生成次态，然后是时序逻辑部分进行状态转换，最后是输出部分，改写的 verilog 代码如下

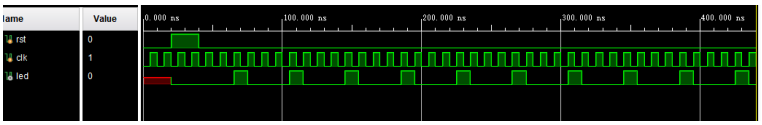
```
module p1(
input clk,rst,
output led
    );
reg [1:0] curr_state;
reg [1:0] next_state;
//the first part
always@(*)
    begin
    case(curr_state)
        2'b00: next_state=2'b01;
        2'b01: next_state=2'b10;
        2'b10: next_state=2'b11;
        2'b11: next_state=2'b00;
     endcase
    end
```

上图是第一段，这个 step5 中的状态就只有 4 个 00，01，10，11，每次根据现态生成次态就是 +1，这里用 case 语句实现

```
    //the second part
    always@(posedge clk or posedge rst)
        begin
            if(rst)
            curr_state <=2'b00;
            else
            curr_state <= next_state;
        end
    //the third part
    assign led=(curr_state==2'b11) ? 1'b1 : 1'b0;

endmodule
```

上图是第二，三段，每当时钟上升沿到来时进行状态转换，并且当前状态为 11 时，输出 1.

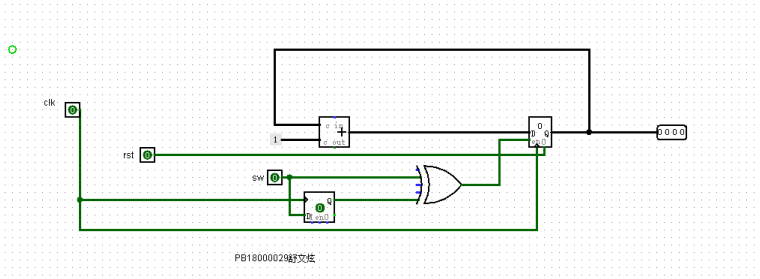


这是跑了仿真之后的波形，可以看到这里每隔四个时钟上升沿输出 led 就会变成 1，这说明这个代码没有问题

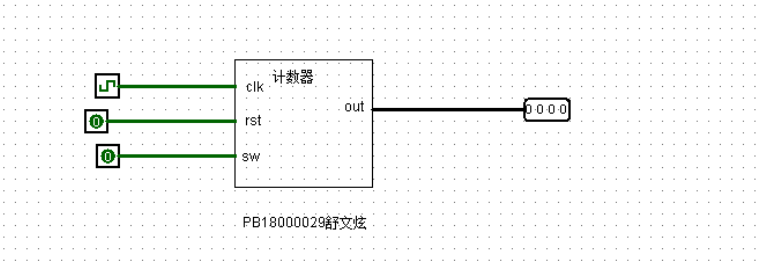
题目 2

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。

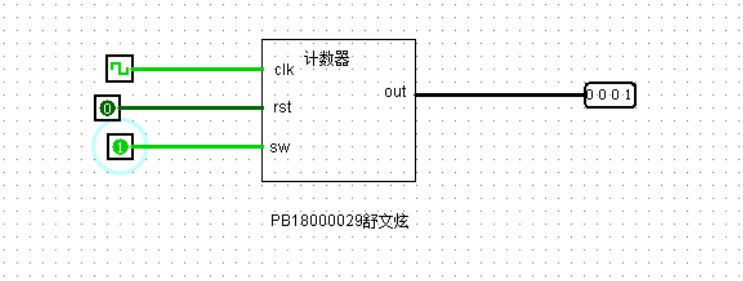
这里主要需要用到取信号边沿的电路，具体电路如下



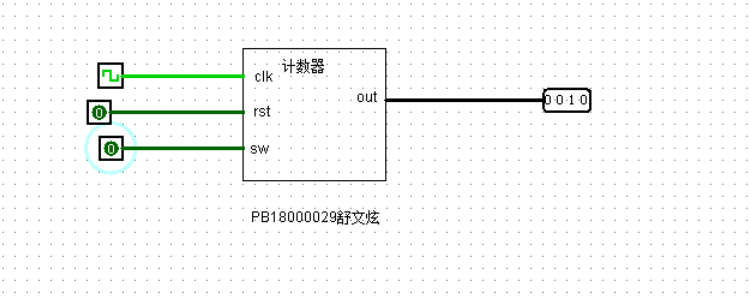
这里用加法器和寄存器组合成计数器，加法器通过寄存器反馈输入然后加 1 这个常数，将这个传给寄存器，这样当时钟上升沿来的时候，就可以将加后的值更新，rst 进行异步复位，下面通过 sw 信号和 D 触发器，外加异或门的方式，取得了 sw 信号的上升沿和下降沿，，将这个输给寄存器的 en 接口，这个接口的意义是，当输入信号为 1 时，才会对寄存器的值更新，否则不变。



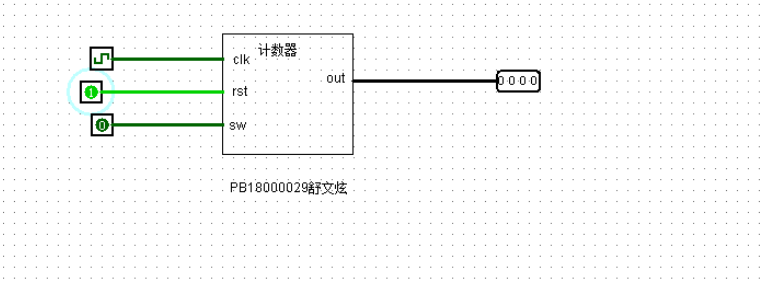
封装一下就成了这样，然后尝试运行



按一下 sw，计数值 +1



再按一下，又 +1



最后按一下 rst 进行复位，可以看到这个电路是满足题目需要的功能的

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 sw[0] 控制计数模式，开关为 1 时为累加模式，为 0 时为递减模式，按键控制计数，按下的瞬间根据开关的状态进行累加或递减计数。计数值用数码管显示，其复位值为 “1F”。

这里用按键控制计数，所以复位信号，我使用了 sw[1] 对应的管脚，计数器的实现十分简单，这里要用到八个数码管，所以需要生成 400Hz 的时钟，这些都在上一次实验中做过，下面放上我的代码

```
module p3(
input sw,clk,rst,button,
output reg [3:0] led,
output reg [2:0] an
    );
wire clk_400hz;
reg [2:0] count=0;
reg [31:0] led0;
wire button_e;
pulse inst_1( clk(clk), .clk_400hz(clk_400hz));
button_edge inst_2( clk(clk), .button(button), .ebutton(button_e));
always@(posedge clk_400hz)
    begin
    case(count)
    3'b000:
    begin
    an=3'b000;led=led0[3:0];
    end
    3'b001:
    begin
    an=3'b001;led=led0[7:4];
    end
    3'b010:
    begin
```

用 pulse 模块生成 400Hz 的时钟，用 button_edge 模块生成按钮的边沿信号，下面就是用 400Hz 的时钟信号来扫描数码管，选择显示哪一个

```
    end
    3'b010:
    begin
    an=3'b010;led=led0[11:8];
    end
    3'b011:
    begin
    an=3'b011;led=led0[15:12];
    end
    3'b100:
    begin
    an=3'b100;led=led0[19:16];
    end
    3'b101:
    begin
    an=3'b101;led=led0[23:20];
    end
    3'b110:
    begin
    an=3'b110;led=led0[27:24];
    end
    3'b111:
    begin
    an=3'b111;led=led0[31:28];
    end
```

因为这里对 8 个数码管，每个都写，所以很长 orz，不过没什么东西在里面

```
    3'b111:
    begin
    an=3'b111;led=led0[31:28];
    end
    endcase
    count=count+1;
    end
always@(posedge clk or posedge rst)
    begin
    if(rst)
        led0=32'h0000001f;
    else
        if(button_e)
            begin
            if(sw)
                led0=led0+1;
            else
                led0=led0-1;
            end
        else
            led0=led0;
    end

endmodule
```

这一部分就是生成次态信号以及状态转换，在按钮边沿到来时，如果 sw[0]=1，就 +1，sw[0]=0，就-1，这里的复位信号 rst 是异步高电平有效的。

```
module pulse(
input clk,
output reg clk_400hz
    );
integer count=0;
wire pulse;
always@(posedge clk)
    begin
    if(count<125000)
        count=count+1;
    else
        count=0;
    end
assign pulse=(count==0);
always@(posedge clk)
begin
    if(pulse)
        clk_400hz = ~clk_400hz;
end
endmodule
```

这是 pulse 模块的代码

```
22
23  module button_edge(
24  input clk,button,
25  output ebutton
26      );
27  reg button1,button2;
28  always@(posedge clk)
29     button1<=button;
30  always@(posedge clk)
31     button2<=button1;
32  assign ebutton=button1&~button2;
33  endmodule
34
```

这是 button_edge 模块的代码

```
set_property -dict { PACKAGE_PIN E3 IOSTANDARD LVCMOS33 } [get_ports { clk }];
set_property -dict { PACKAGE_PIN B18 IOSTANDARD LVCMOS33 } [get_ports { button }];
## leds
set_property -dict { PACKAGE_PIN A15 IOSTANDARD LVCMOS33 } [get_ports { led[3] }];
set_property -dict { PACKAGE_PIN A16 IOSTANDARD LVCMOS33 } [get_ports { led[2] }];
set_property -dict { PACKAGE_PIN A13 IOSTANDARD LVCMOS33 } [get_ports { led[1] }];
set_property -dict { PACKAGE_PIN A14 IOSTANDARD LVCMOS33 } [get_ports { led[0] }];
set_property -dict { PACKAGE_PIN B17 IOSTANDARD LVCMOS33 } [get_ports { an[0] }];
set_property -dict { PACKAGE_PIN B16 IOSTANDARD LVCMOS33 } [get_ports { an[1] }];
set_property -dict { PACKAGE_PIN A18 IOSTANDARD LVCMOS33 } [get_ports { an[2] }];
set_property -dict { PACKAGE_PIN D14 IOSTANDARD LVCMOS33 } [get_ports { sw }];
set_property -dict { PACKAGE_PIN F16 IOSTANDARD LVCMOS33 } [get_ports { rst }];
```

这是对应的约束文件，和实验七用到的没有多少区别，这里就是将 rst 信号连接到了 sw[1] 对应的管脚，多了 button 对应的管脚，下面生成比特文件，在 FPGA 在线开发平台跑的结果

初始复位值 1F

sw[0] 设为 1，按一下按钮 +1，变成 20

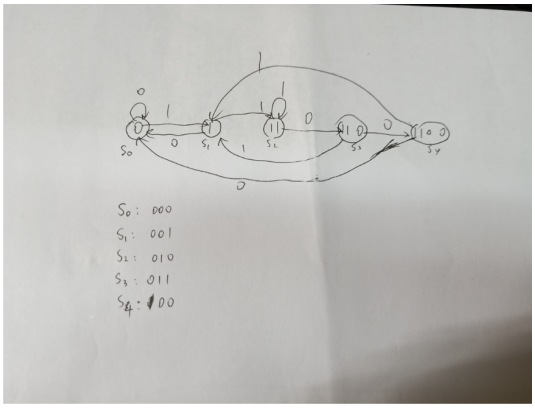
sw[0] 设为 0，按两下按钮，-2，变成 1E

sw[1] 设为 1，直接复位，变成 1F
可以看到这里题目要求的功能都正常实现了

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 sw[0] 进行数据的串行输入，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。

这里我先给出我画的状态转移图



根据题目描述，这里一共有 5 个状态，编码我也写在这张纸上了，放到 FPGA 上，这里的编码直接用其对应的十六进制数表示即可，这样只需要用到一个数码管就可以显示当前状态编码，下面是我的具体 verilog 代码

```
//////////////////////////////////////////////////////////////////////////////////
module p4(
input clk,sw,button,rst,
output reg [3:0] led,
output reg [2:0] an
);
reg [15:0] curr_state;
reg [15:0] next_state;
wire button_e;
wire clk_300hz;
reg [2:0] count=0;
reg [3:0] curr_count,next_count,curr_st,next_st;

parameter s0=4'b0000;
parameter s1=4'b0001;
parameter s2=4'b0010;
parameter s3=4'b0011;
parameter s4=4'b0100;


button_edge inst_1(.clk(clk),.button(button),.ebutton(button_e));
pulse inst_2(.clk(clk),.clk_300hz(clk_300hz));
```

这是需要的变量以及一些参数的定义

```
always@(*)
    begin
    if(button_e)
        begin
        if(sw==0)
            begin
            next_state={curr_state[11:0],4'b0000};
            if(next_state==16'h1100)
                next_count=curr_count+1;
             else
                next_count=curr_count;
            end
         else
            begin
            next_state={curr_state[11:0],4'b0001};
            if(next_state==16'h1100)
                next_count=curr_count+1;
             else
                next_count=curr_count;
            end
        end
     else
     begin
        next_state=curr_state;
        next_count=curr_count;
```

```
67      end
68      end
69
70  always@(*)
71  begin
72  if(button_e)
73  begin
74  case(curr_st)
75  s0:
76  begin
77  if(sw==0) next_st=s0;
78  else next_st=s1;
79  end
80   s1:
81  begin
82  if(sw==0) next_st=s0;
83  else next_st=s2;
84  end
85   s2:
86  begin
87  if(sw==0) next_st=s3;
88  else next_st=s2;
89  end
90   s3:
```

```
89  end
90   s3:
91  begin
92  if(sw==0) next_st=s4;
93  else next_st=s1;
94  end
95   s4:
96  begin
97  if(sw==0) next_st=s0;
98  else next_st=s1;
99  end
100 endcase
101 end
102 else next_st=curr_st;
103 end
```

这一部分就是组合逻辑部分，进行次态的生成，计数器这个用 curr_count,next_count 表示，记录已经出现了多少 1100 序列，这个就遇到对应序列 +1 即可；最近输入的数值，这个用 curr_state,next_state 表示，每次看 sw 信号是什么，进行左移位操作，把 sw 信号移进

来。还需要状态编码的现态和次态，这个我就用 curr_zt,next_zt 来表示了，这个状态用 case 语句比较方便

```
always@(posedge clk or posedge rst)
   begin
   if(rst)
       begin
       curr_state=16'h0000;
       curr_count=4'b0000;
       curr_zt=s0;
       end
   else
       begin
       curr_state=next_state;
       curr_count=next_count;
       curr_zt=next_zt;
       end
    end
```

这一部分是时序逻辑的部分，进行状态的转换，这一部分就很简单，异步复位，复位信号来时，进行复位，没来就将之前生成的次态赋给现态，实现状态的转换

```
always@(posedge clk_300hz)
begin
case(count)
3'b000:
begin
 count=3'b001; an=3'b000;led=curr_state[3:0];
end
3'b001:
begin
 count=3'b010; an=3'b001;led=curr_state[7:4];
end
3'b010:
begin
 count=3'b011; an=3'b010;led=curr_state[11:8];
end
3'b011:
begin
 count=3'b100; an=3'b011;led=curr_state[15:12];
end
3'b100:
begin
 count=3'b101; an=3'b100;led=curr_count;
end
3'b101:
begin
```

```
45  3'b101:
46  begin
47   count=3'b000; an=3'b101;led=curr_zt;
48  end
49  endcase
50  end
51  endmodule
```

这一部分是输出的部分，由于要将输出放到数码管上，这里有 6 个数码管，所以需要给一个 300Hz 的时钟信号进行扫描输出。

```
module pulse(
input clk,
output reg clk_300hz
    );
integer count=0;
wire pulse;
always@(posedge clk)
    begin
    if(count<166667)
        count=count+1;
    else
        count=0;
    end
assign pulse=(count==0);
always@(posedge clk)
begin
    if(pulse)
        clk_300hz = ~clk_300hz;
end
endmodule
```

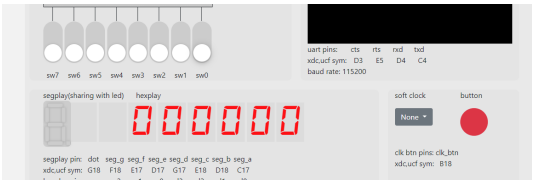
上面的模块用来生成 300Hz 的时钟信号

```
module button_edge(
input clk,button,
output ebutton
    );
reg button1,button2;
always@(posedge clk)
   button1<=button;
always@(posedge clk)
   button2<=button1;
assign ebutton=button1&~button2;
endmodule
```

上面的模块用来取按钮的边沿

```
set_property -dict { PACKAGE_PIN E3 IOSTANDARD LVCMOS33 } [get_ports { clk }];
set_property -dict { PACKAGE_PIN B18 IOSTANDARD LVCMOS33 } [get_ports { button }];
## leds
set_property -dict { PACKAGE_PIN A15 IOSTANDARD LVCMOS33 } [get_ports { led[3] }];
set_property -dict { PACKAGE_PIN A16 IOSTANDARD LVCMOS33 } [get_ports { led[2] }];
set_property -dict { PACKAGE_PIN A13 IOSTANDARD LVCMOS33 } [get_ports { led[1] }];
set_property -dict { PACKAGE_PIN A14 IOSTANDARD LVCMOS33 } [get_ports { led[0] }];
set_property -dict { PACKAGE_PIN B17 IOSTANDARD LVCMOS33 } [get_ports { an[0] }];
set_property -dict { PACKAGE_PIN B16 IOSTANDARD LVCMOS33 } [get_ports { an[1] }];
set_property -dict { PACKAGE_PIN A18 IOSTANDARD LVCMOS33 } [get_ports { an[2] }];
set_property -dict { PACKAGE_PIN D14 IOSTANDARD LVCMOS33 } [get_ports { sw }];
set_property -dict { PACKAGE_PIN F16 IOSTANDARD LVCMOS33 } [get_ports { rst }];
```

这是我用到的约束文件，和前几次使用的没有很多区别，生成比特文件，烧到 FPGA 在线开发板上运行结果如下



初始状态，第一个是当前状态编码，第二个是序列 1100 的数目，后面四个是最近的四个输入

sw[0] 设为 1，按下按钮后，最近输入的数值变为 0001，状态变为 1

再按一次，最近输入变为 0011，状态变为 2

sw[0] 设为 0，按下按钮后，最近输入的数值变为 0110，状态变为 3

再按一次，最近输入变为 1100，状态变为 4，此时 count 变为 1

再将 sw[0] 设为 1，按一下，状态变成 1，最近输入 1001，可以看到题目要求的功能我都正常实现了

【总结与思考】

1. 请总结本次实验的收获

本次实验学习了如何获得边沿的信号，以及如何构建有限状态机，用这种方式去写代码，可以看到逻辑关系十分清晰，一目了然，巩固了我的 verilog 代码能力

2. 请评价本次实验的难易程度

本次实验的复杂程度和实验 7 实际上是一样的，不过因为已经在实验 7 中得到了锻炼，这一次的实验相对而言没有让我感觉到无从下手，我的评价是难度适中

3. 请评价本次实验的任务量

任务量还是有一点的，实验步骤部分实际上讲了很多毕竟实用的知识，实验练习也有 4 道，做起来还是要花一点时间的

4. 请为本次实验提供改进建议

本次实验做起来还是很舒服的，毕竟前面实验步骤部分讲的很详细，实验练习就是将前面的知识进行应用就可以了，总体来说比较满意。