# 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：Ouedraogo Ezekiel B.

学生学号：PL19215001

完成日期：12/16/2020

计算机实验教学中心制

2020 年 10 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】

VLAB：vlab.ustc.eud.cn
FPGAOL: fpgaol.ustc.edu.cn
Logisim
Vivado

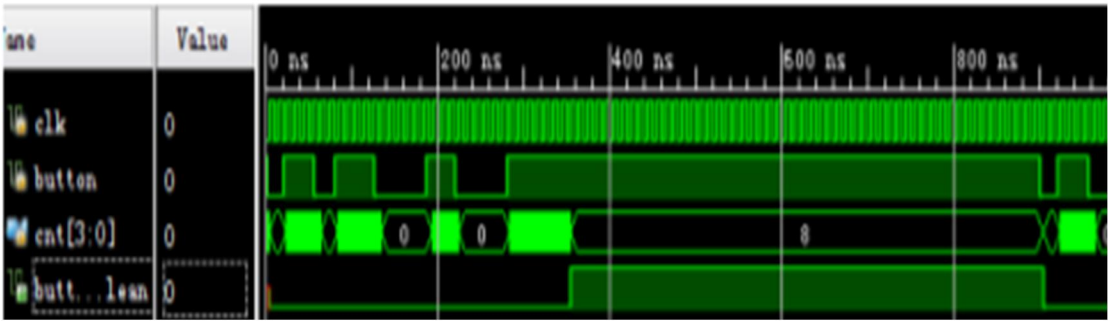
## 【实验过程】

### 1. 信号整形及去毛刺

Verilog 代码

```
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
```

仿真结果



### 2. 取信号边沿技巧

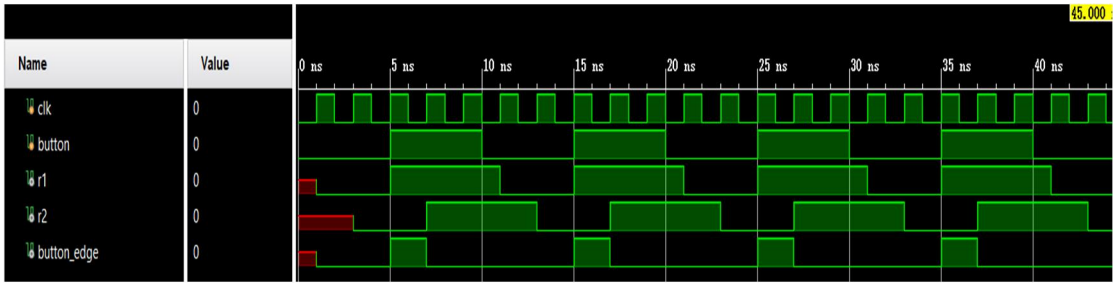
Verilog 代码

```
module signal_edge(
    input clk,
    input button,
    output button_edge);
    reg button_r1,button_r2;
    always@(posedge clk) button_r1 <= button;
    always@(posedge clk) button_r2 <= button_r1;
    assign button_edge = button_r1 & (~button_r2);
endmodule
```

仿真文件

```
module tb();
reg clk,button;
wire button_edge;
signal_edge s(.clk(clk),.button(button),.button_edge(button_edge));
    initial clk <= 0;
    always #1 clk<= ~clk;
    initial
    begin
        button = 0;
        #5 button = 1; #5 button = 0; #5 button = 1; #5 button = 0;
        #5 button = 1; #5 button = 0; #5 button = 1; #5 button = 0;
        #5 $finish;
    end
endmodule
```

结果




## 【实验练习】

### 1. 有限状态机

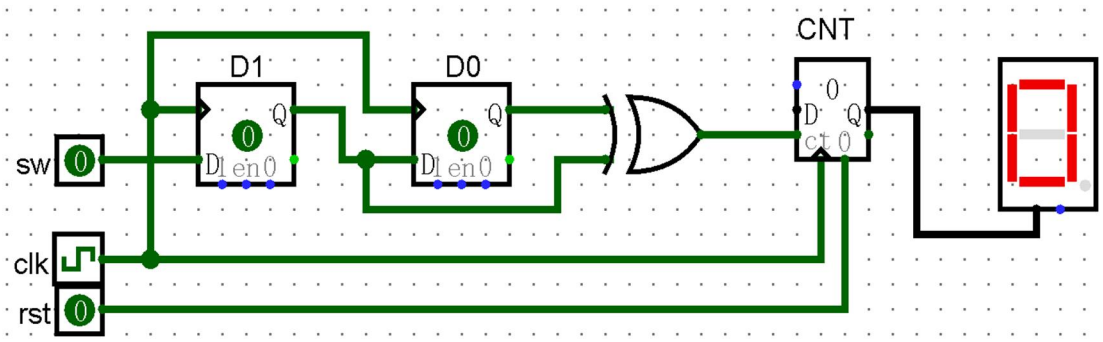
将代码改写成三段式有限状态机的形式

```
module test(
input clk,rst,
output led);
parameter STATE_0 = 2'b00;
```

```
parameter STATE_1 = 2'b01;
parameter STATE_2 = 2'b10;
parameter STATE_3 = 2'b11;
reg [1:0] curr_state;
reg [1:0] next_state;
//有限状态机第一部分
always@(*)
begin
    case(curr_state)
      STATE_0: next_state = STATE_0;
      STATE_1: next_state = STATE_2;
      STATE_2: next_state = STATE_3;
      STATE_3: next_state = STATE_0;
    endcase
end
//有限状态机第二部分
always@(posedge clk or posedge rst)
begin
 if(rst)
 curr_state <= STATE_0;
 else
 curr_state <= next_state;
end
//有限状态机第三部分
assign led = (curr_state==STATE_3);
endmodule
```

### 2. 计数器

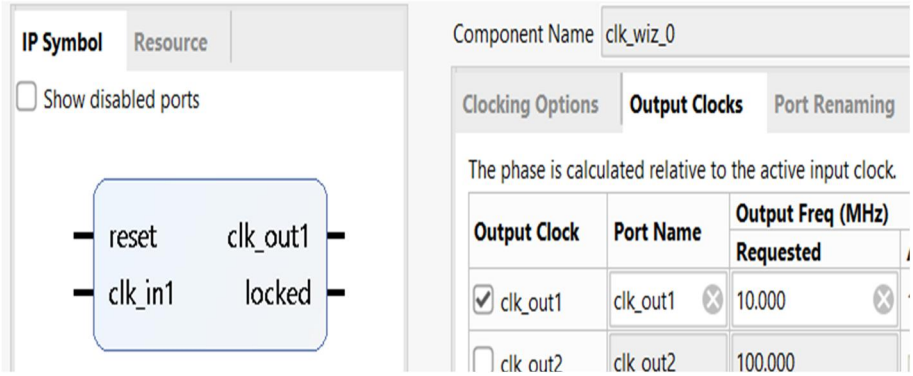
在 Logisim 中设计一个 4bit 位宽的计数器电路，在输入信号 sw 电平发生变化时，计数值 cnt 加 1。




图中 D0,D1 为 D 触发器，CNT 为 4 位宽的计数器。

### 3. 八位的十六进制计数器

例化一个 10Mhz 的时钟



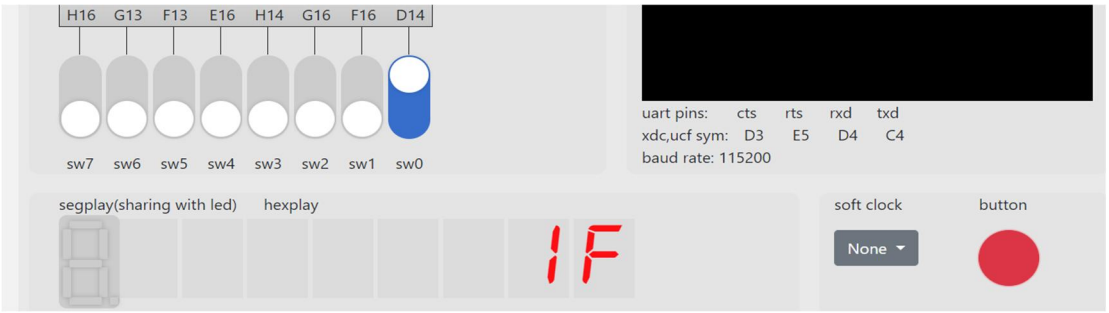

Verilog 代码

```
module signal_edge(
    input clk,
    input button,
    output button_edge);
    reg button_r1,button_r2;
    always@(posedge clk) button_r1 <= button;
    always@(posedge clk) button_r2 <= button_r1;
    assign button_edge = button_r1 & (~button_r2);
endmodule

module count(
input clk, start,
input [1:0] sw,
output reg [3:0]out,
output reg [2:0]an
    );
    wire start_edge;
    signal_edge
signal_edge(.clk(clk),.button(start),.button_edge(start_edge));
    reg [7:0]tmp;
    wire clk_10mz;
    clk_wiz_0 clk_wiz(.clk_in1(clk),.reset(0),.clk_out1(clk_10mz));
    always@(posedge clk)
    begin
        if(sw[1]) tmp <= 8'h1f;
        else if(start_edge)
        begin
            if(sw) tmp <= tmp + 8'h1;
            else   tmp <= tmp - 8'h1;
        end
    end
    always@(*)
    begin
        if(clk_10mz)
```

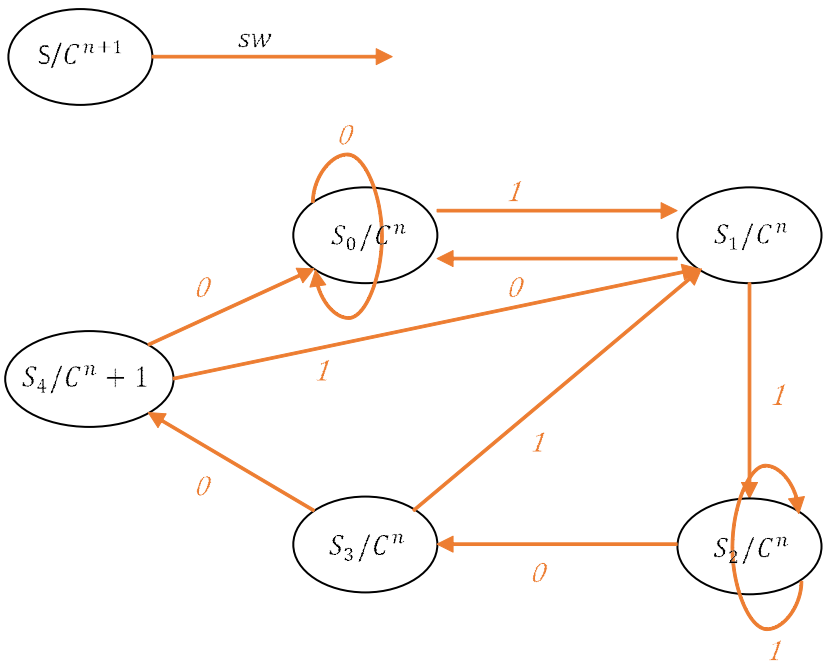
```
        begin
            an <= 3'b0;
            out <= tmp[3:0];
        end
        else
        begin
            an <= 3'b1;
            out <= tmp[7:4];
        end
    end
endmodule
```

结果




## 4. 序列检测电路

### 4.1. 状态跳转图



$S \rightarrow STATE$
$C \rightarrow Count\ (count\ 1100\ numbers)$
$C^n \rightarrow current\ count\ value;\ C^{n+1} \rightarrow next\ count\ value$
$sw \rightarrow input\ (switch)$

### 4.2. 电路实现

代码

```
module counter(
    input clk, rst, start, sw,
    output reg [3:0] curr_state, count, last
    );
    parameter [3:0]  STATE_0 = 4'h0,
                     STATE_1 = 4'h1,
                     STATE_2 = 4'h2,
                     STATE_3 = 4'h3,
                     STATE_4 = 4'h4;
    reg [3:0] next_state, last_tmp;
    wire start_edge;
    signal_edge s(.clk(clk),.button(start),.button_edge(start_edge));
    always@(*)
    begin
        if(start_edge)
        begin
            case(curr_state)
              STATE_0: begin
                           if(sw) next_state <= STATE_1;
                           else   next_state <= curr_state;
                       end
              STATE_1: begin
                           if(sw) next_state <= STATE_2;
                           else   next_state <= STATE_0;
                       end
              STATE_2: begin
                           if(sw) next_state <= curr_state;
                           else   next_state <= STATE_3;
                       end
              STATE_3: begin
                           if(sw) next_state <= STATE_1;
                           else   next_state <= STATE_4;
                       end
              STATE_4: begin
                           if(sw) next_state <= STATE_1;
                           else   next_state <= STATE_0;
                       end
```

```
                default: next_state <= STATE_0;
            endcase
        end//if start_edge
        else next_state <= curr_state;
    end
    reg once1; //use once to make sure a task just run only once
    always@(posedge clk or posedge rst)
    begin
        if(rst) last <= 4'h0;
        else if(start_edge)
        begin
            if(once1)
            begin
                //save last inputs
                last <= {sw,last_tmp[3:1]};
                once1 <= 1'b0;
            end
        end
        else
        begin
            once1 <= 1'b1;
            last_tmp <= last;
        end
    end
    always@(posedge clk or posedge rst)
    begin
        if(rst) curr_state <= STATE_0;
        else curr_state <= next_state;
    end
    reg once2;
    always@(posedge clk)
    begin
        if(rst) count <= 4'h0;
        else if(curr_state==STATE_4)
        begin
            if(once2 )
            begin
                count <= count + 4'b1;
                once2 <= 1'b0;
            end
        end
        else once2 <= 1'b1;
    end
endmodule
```

```
module display(
    input clk, rst, start, sw,
    output reg [3:0] out,
    output reg [2:0] an
    );
    wire [3:0] curr_state, count, last;
    counter
c(.clk(clk),.rst(rst),.start(start),.sw(sw),.curr_state(curr_state),.
count(count),.last(last));
    wire clk_out1;
    clk_wiz_0 clk_wiz_0(.clk_in1(clk),.reset(0),.clk_out1(clk_out1));
    reg [2:0] tmp;
    always@(posedge clk_out1)
    begin
        if(tmp >= 3'h5) tmp <= 3'h0;
        else    tmp <= tmp + 3'h1;
    end
    always@(*)
    begin
        case(tmp)
            3'h0: begin
                      an <= 3'h0;
                      out <= curr_state;
                  end
            3'h1: begin
                      an <= 3'h2;
                      out <= count;
                  end
            3'h2: begin
                      an <= 3'h4;
                      out <= {3'h0,last[0]};
                  end
            3'h3: begin
                      an <= 3'h5;
                      out <= {3'h0,last[1]};
                  end
            3'h4: begin
                      an <= 3'h6;
                      out <= {3'h0,last[2]};
                  end
            3'h5: begin
                      an <= 3'h7;
                      out <= {3'h0,last[3]};
                  end
```

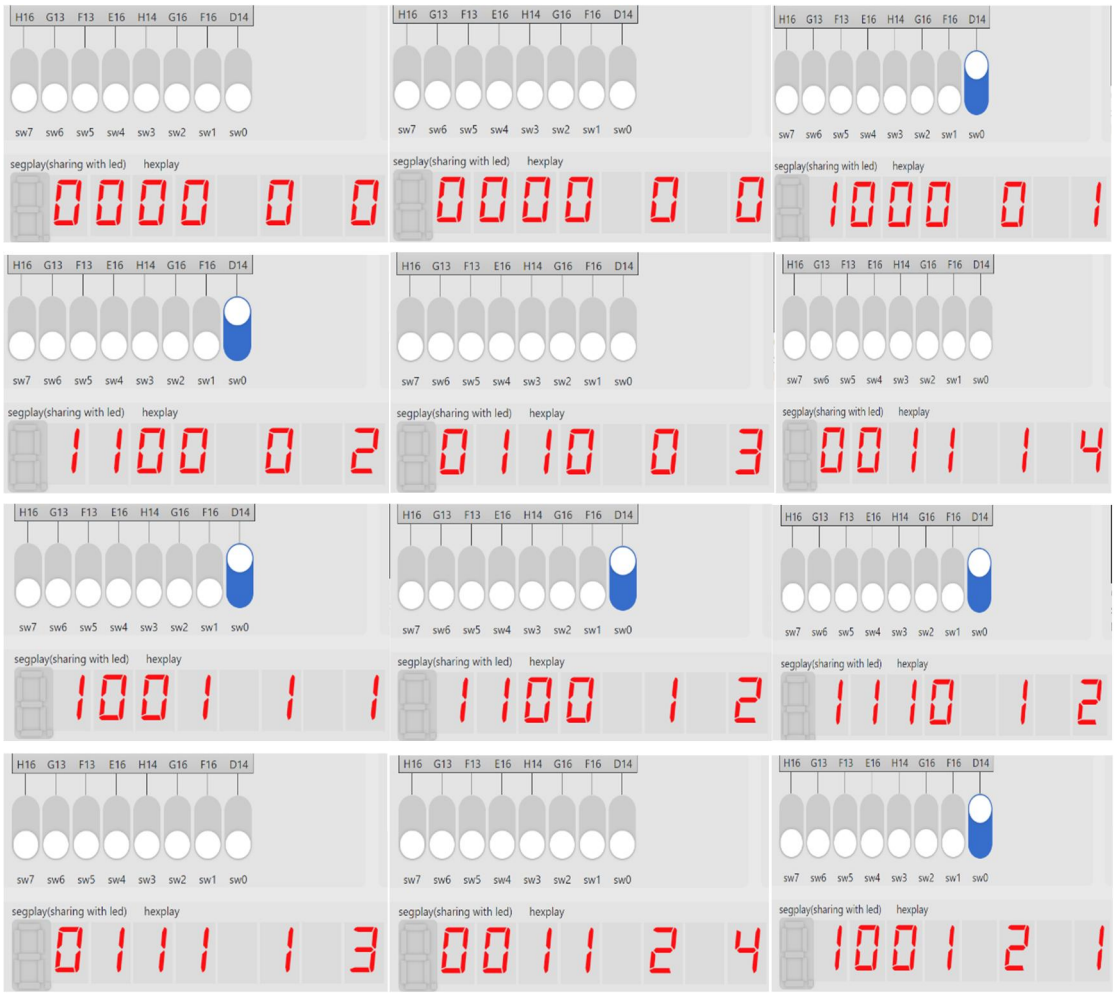
```
        endcase
    end
endmodule
```

XDC 文件

```
## Clock signal
set_property -dict { PACKAGE_PIN E3   IOSTANDARD LVCMOS33 } [get_ports { clk }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
#create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {CLK100MHZ}];
## FPGAOL BUTTON & SOFT_CLOCK
set_property -dict { PACKAGE_PIN B18  IOSTANDARD LVCMOS33 } [get_ports { start }];
## Switches
set_property -dict { PACKAGE_PIN D14  IOSTANDARD LVCMOS33 } [get_ports { sw }];
set_property -dict { PACKAGE_PIN F16  IOSTANDARD LVCMOS33 } [get_ports { rst }];
## Hexa Digit Display
set_property -dict { PACKAGE_PIN A14  IOSTANDARD LVCMOS33 } [get_ports { out[0] }];
set_property -dict { PACKAGE_PIN A13  IOSTANDARD LVCMOS33 } [get_ports { out[1] }];
set_property -dict { PACKAGE_PIN A16  IOSTANDARD LVCMOS33 } [get_ports { out[2] }];
set_property -dict { PACKAGE_PIN A15  IOSTANDARD LVCMOS33 } [get_ports { out[3] }];
##
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] }];
```

结果

下面输入为“0011001110011”,最后清零（rst）




【总结与思考】

在本次实验中咱们掌握几种常见的信号处理技巧如取信号边沿，也掌握有限状态机的设计方法。