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：___宋玮_____

学生学号：_PB20151793_

完成日期：__2021.12.8_

计算机实验教学中心制

【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】

Step1.信号整形及去毛刺

通过一个计数器 对高电平持续时间进行计时，当按键输入信号为 0 时，计数器清零， 当输入信号为高电平时，计数器进行累加计数，计数达到阈值后则停 止计数，如下面代码所示：

```
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
```

Step2.取信号边沿技巧

通过两个寄存器对输入信号进行寄存，寄存器后的信号分别为 button_r1,button_r2，然后将 button_r2 取反并和 button_r1 进行与操作，便得到了一个时钟周期宽度的脉冲信号，该信号在 button 信号的上升沿附近为高电平，其余时间均为低电平。

```
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
```

Step3. 有限状态机介绍

Step4.有限状态机 Verilog 实现

【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。

完整的 Verilog 代码如下：

```
module lab8_1(
  input clk,
  input rst,
  output led
);
    parameter s0 = 2'b00;
    parameter s1 = 2'b01;
    parameter s2 = 2'b10;
     parameter s3 = 2'b11;
```

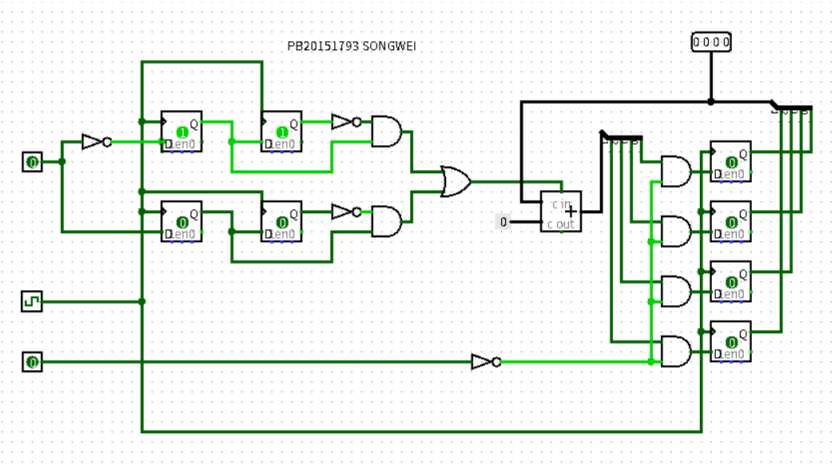
```
    reg [1:0] state;
    reg [1:0] next_state;

    always@(*)
    begin
      case(state)
           s0: next_state = s1;
           s1: next_state = s2;
           s2: next_state = s3;
           s3: next_state = s0;
           default: next_state = s0;
      endcase
    end

    always@(posedge clk or posedge rst)
    begin
       if (rst)
           state <= s0;
       else
           state <= next_state;
    end
  assign led = (state==2'b11) ? 1'b1 : 1'b0;
endmodule
```

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。

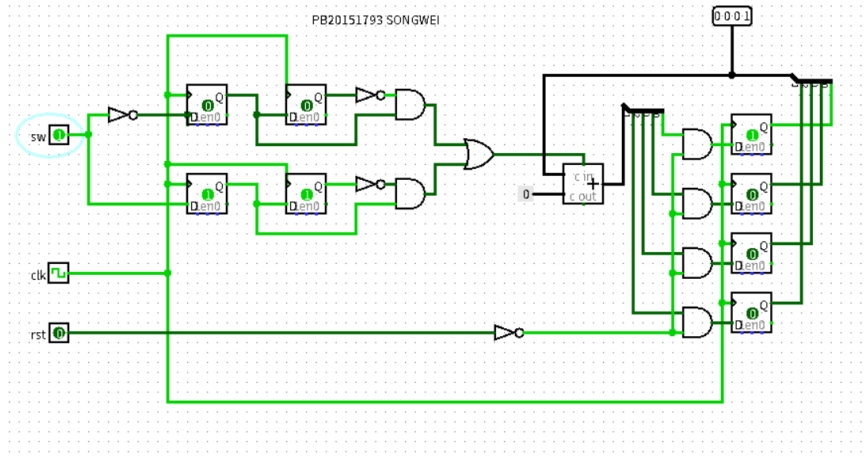
分析：由于题目要求在 sw 信号上升沿时刻和下降沿时刻各触发一次计数操作，因此前面一部分的电路是用于当 sw 从 0->1,或从 1->0 时，触发一次计数操作。后面的电路是保存计数值。

rst 为复位信号。

实验效果图如下：



当 sw 从 0->1 后：



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”。

设计文件：

```
module lab8_3(
input clk1,
input rst,
input sw,
input b,
output reg [3:0] d,
output reg [2:0] a
    );
reg [7:0] count;
wire b_edge;
wire clk;
wire rs=1'b0;

button0 button0(
.clk(clk),
.button(b),
.button_edge(b_edge));

always@(posedge clk)
         begin
            if (a == 1)
                    a <= 0;
            else
                a <= a + 1;

        end

always@(*) begin
        case(a)
            0: d = count[3:0];
            1: d = count[7:4];
        endcase
    end

always@(posedge clk)
     begin
         if(rst == 1)begin
            count <= 8'h1f;
         end
         else
            begin
                if(b_edge == 1)
                begin
```

```
                if (sw)
                begin
                  count <= count + 1;
                end
                else
                begin
                  count <= count - 1;
                end
            end
            else
            begin
               count <= count;
            end
         end
      end

clk_wiz_0 clk_wiz_0(
        .clk_out1(clk),
        .reset(rs),
        .clk_in1(clk1)
        );

endmodule
```

调用模块：

```
module button0(
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
```

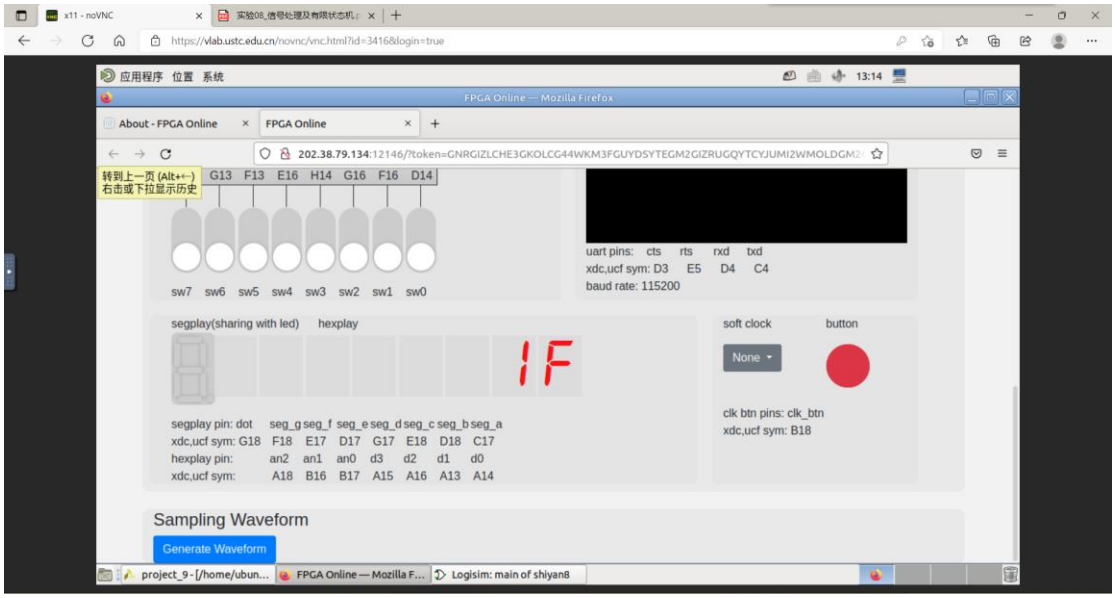
时钟 ip 核输出频率为 10mhz。

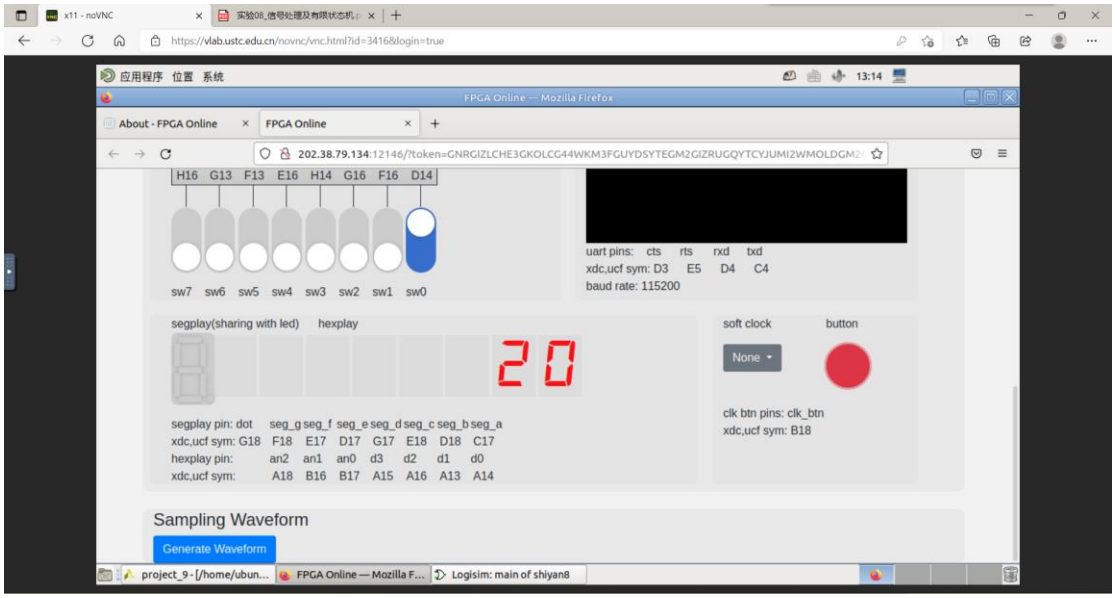
约束文件：

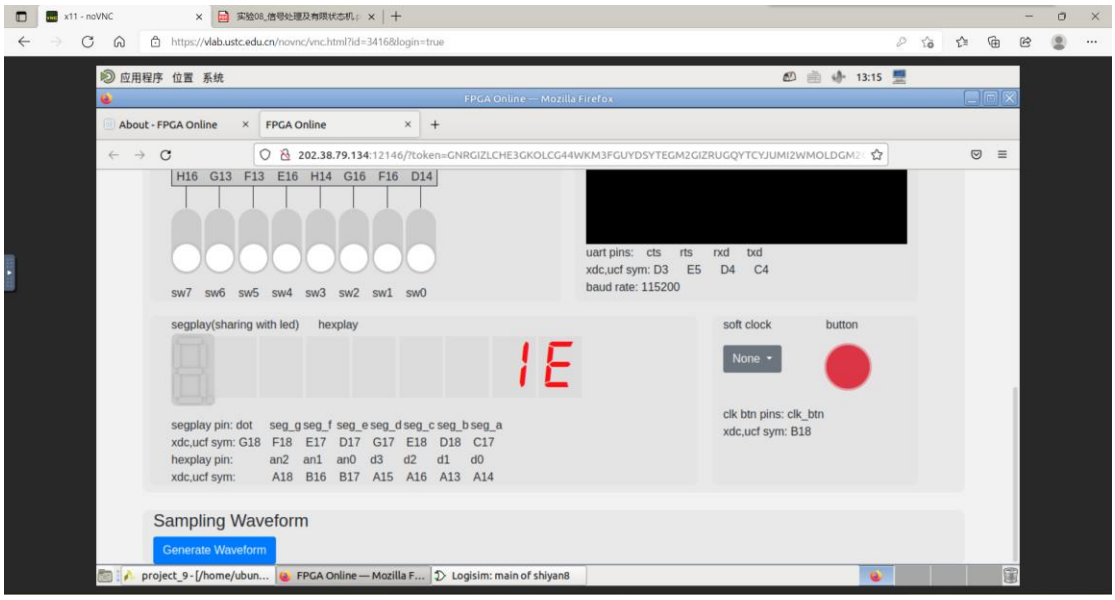
```
set_property -dict { PACKAGE_PIN E3   IOSTANDARD LVCMOS33 } [get_ports {clk1}];
set_property -dict { PACKAGE_PIN B18   IOSTANDARD LVCMOS33 } [get_ports {b}];
```

```
set_property -dict { PACKAGE_PIN A15   IOSTANDARD LVCMOS33 } [get_ports {d[3]}];
set_property -dict { PACKAGE_PIN A16   IOSTANDARD LVCMOS33 } [get_ports {d[2]}];
set_property -dict { PACKAGE_PIN A13   IOSTANDARD LVCMOS33 } [get_ports {d[1]}];
set_property -dict { PACKAGE_PIN A14   IOSTANDARD LVCMOS33 } [get_ports {d[0]}];
set_property -dict { PACKAGE_PIN A18   IOSTANDARD LVCMOS33 } [get_ports {a[2]}];
set_property -dict { PACKAGE_PIN B16   IOSTANDARD LVCMOS33 } [get_ports {a[1]}];
set_property -dict { PACKAGE_PIN B17   IOSTANDARD LVCMOS33 } [get_ports {a[0]}];

set_property -dict { PACKAGE_PIN F16   IOSTANDARD LVCMOS33 } [get_ports {rst}];
set_property -dict { PACKAGE_PIN D14   IOSTANDARD LVCMOS33 } [get_ports {sw}];
```

实验效果图如下：

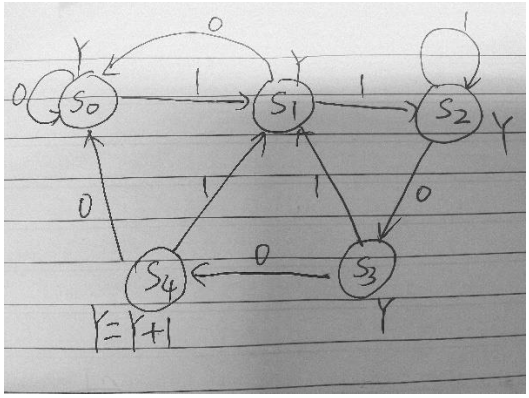






题目 4. 使用有限状态机设计一个序列检测电路，并进行计数，当检测到输入序列为“1100”时，计数器加一，用一个数码管显示当前状态编码，一个数码管显示检测到目标序列的个数，用 4 个数码管显示最近输入的 4 个数值，用 sw[0]进行数据的串行输入，按键每按下一次将输入一次开关状态，时钟采用板载的 100MHz 时钟。 要求画出状态跳转图，并在 FPGA 开发板上实现电路，例如当输入“0011001110011”时，目标序列个数应为 2，最近输入数值显示“0011”， 状态机编码则与具体实现有关。

状态图如下所示：



设计文件：

```
module lab8_4(
input clk1,
input sw,
input b,
output reg [3:0] d,
output reg [2:0] a
    );
        wire clk;
        wire rs=1'b0;
        reg [2:0] state;
        reg [2:0] next_state;
        reg [23:0] data;
        wire b_edge;
        reg [3:0] count;
        reg [3:0] id;

   button_1 button_1(
        .clk(clk),
        .button(b),
        .button_edge(b_edge));

initial
        begin
                count = 0;
        end

        always@(posedge clk)
        begin
            if(b_edge == 1)
            begin
                id <= {id[2:0],sw};
            end
        end


        always@(posedge clk)
        begin
            data[3:0] <= {3'b000,id[0]};
            data[7:4] <= {3'b000,id[1]};
            data[11:8] <= {3'b000,id[2]};
            data[15:12] <= {3'b000,id[3]};
        end
```

```
always@(posedge clk)
                begin
                    if (a == 5)
                            a <= 0;
                    else
                        a <= a + 1;

                end

always@(*)
              begin
                    case(a)
                        0: d = data[3:0];
                        1: d = data[7:4];
                        2: d = data[11:8];
                        3: d = data[15:12];
                        4: d = data[19:16];
                        5: d = data[23:20];
                    endcase
                end

        parameter s0=3'b000;
        parameter s1=3'b001;
        parameter s2=3'b010;
        parameter s3=3'b011;
        parameter s4=3'b100;

        always @(*) begin
            case (state)
                s0:begin
                    if(sw == 1 & b_edge == 1)begin
                        next_state = s1;
                    end
                    if(sw == 0 & b_edge == 1)begin
                        next_state = s0;
                    end
                end
                s1:begin
                    if(sw == 1 & b_edge == 1)begin
                        next_state = s2;
                    end
                    if(sw == 0 & b_edge == 1)begin
                        next_state = s0;
                    end
```

```
            end
            s2:begin
                if(sw == 1 & b_edge == 1)begin
                    next_state = s2;
                end
                if(sw == 0 & b_edge == 1)begin
                    next_state = s3;
                end
            end
            s3:begin
                if(sw == 1 & b_edge == 1)begin
                    next_state = s1;
                end
                if(sw == 0 & b_edge == 1)begin
                    next_state = s4;
                end
            end
            s4:begin
                if(sw == 1 & b_edge == 1)begin
                    next_state = s1;
                end
                if(sw == 0 & b_edge == 1)begin
                    next_state = s0;
                end
            end
            default:begin
                next_state = next_state;
            end
        endcase
    end

    always@(posedge clk)
    begin
            if(b_edge)
            begin
                if(state == s3 & sw == 0)
                begin
                    count <= count + 1;
                end
                state <= next_state;
            end
    end

always@(posedge clk)
```

```
        begin
            data[19:16] <= count;
        end

        always@(posedge clk) begin
                case (state)
                    s0:begin
                        data[23:20] <= 3'b0;
                    end
                    s1:begin
                        data[23:20] <= 3'b1;
                    end
                    s2:begin
                        data[23:20] <= 3'b010;
                    end
                    s3:begin
                        data[23:20] <= 3'b011;
                    end
                    s4:begin
                        data[23:20] <= 3'b100;
                    end
                    default: data[23:20] <= 3'b0;
                endcase
            end
clk_wiz_0 clk_wiz_0(
                     .clk_out1(clk),
                     .reset(rs),
                     .clk_in1(clk1)
                     );

endmodule
```

调用模块：

```
module button_1(
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;
```

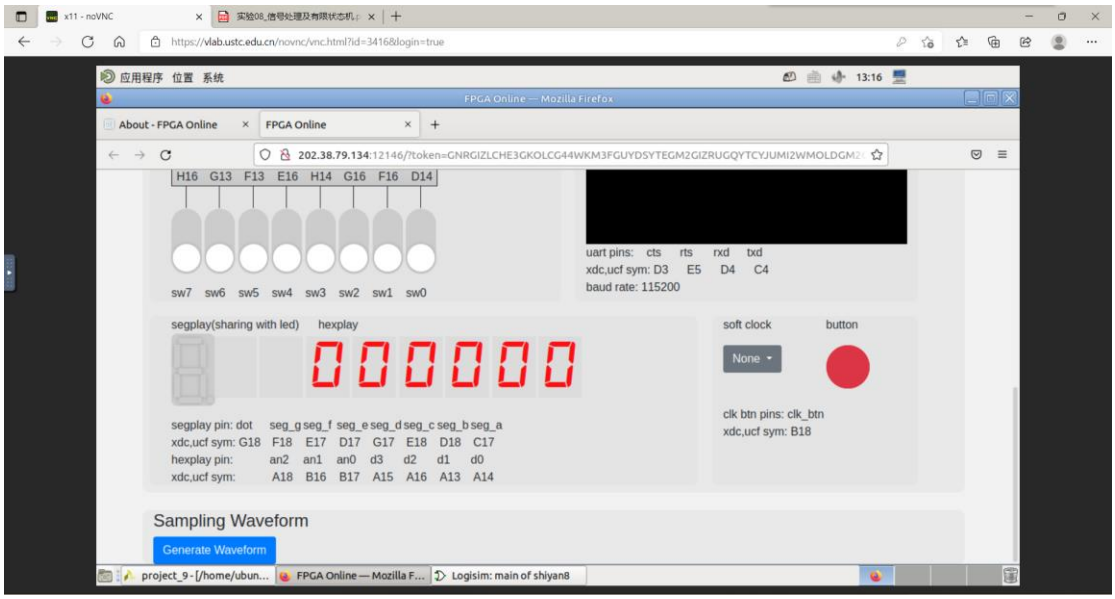
```
assign button_edge = button_r1 & (~button_r2);
endmodule
```

时钟 ip 核输出频率为 10mhz。

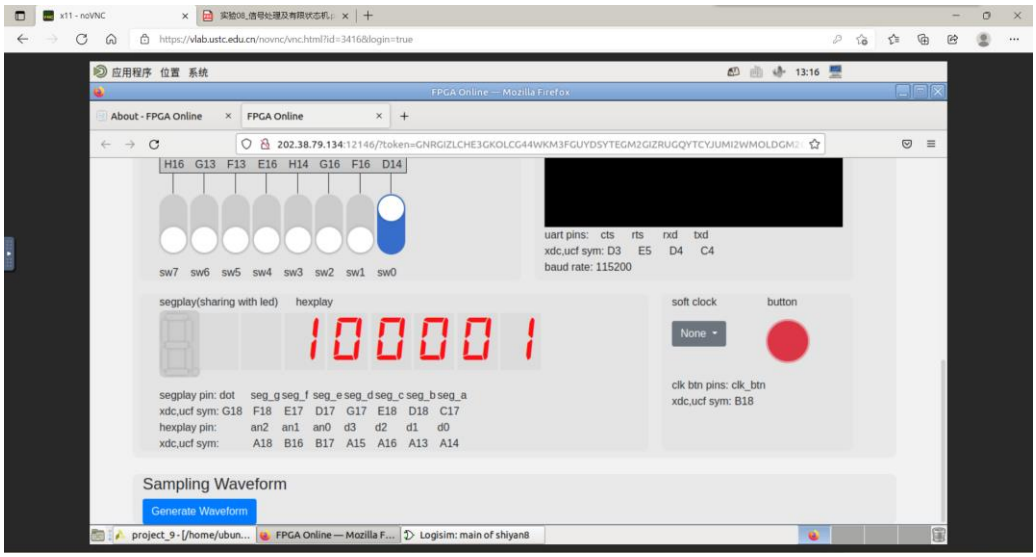
约束文件：

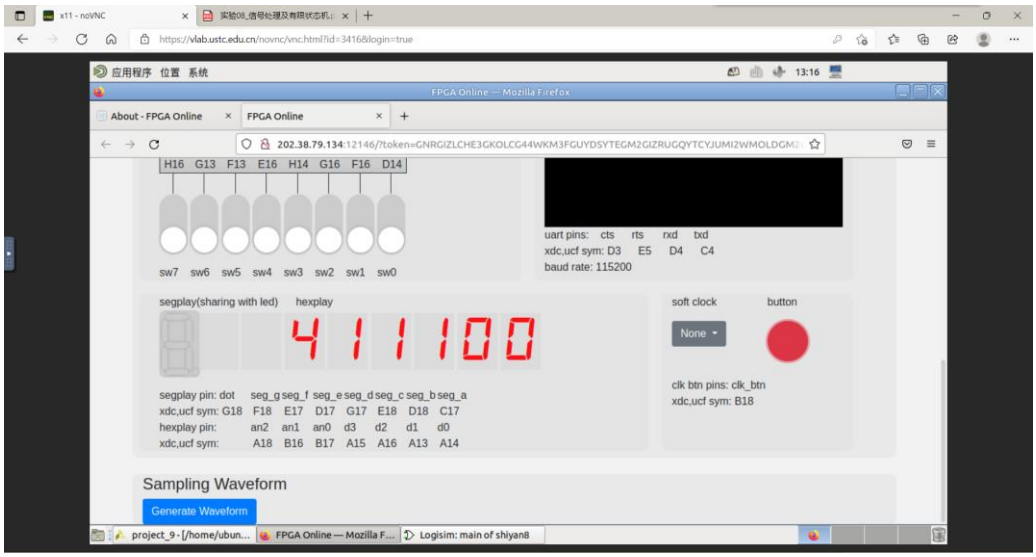
```
set_property -dict { PACKAGE_PIN E3   IOSTANDARD LVCMOS33 } [get_ports {clk1}];
set_property -dict { PACKAGE_PIN B18   IOSTANDARD LVCMOS33 } [get_ports {b}];


set_property -dict { PACKAGE_PIN A15   IOSTANDARD LVCMOS33 } [get_ports {d[3]}];
set_property -dict { PACKAGE_PIN A16   IOSTANDARD LVCMOS33 } [get_ports {d[2]}];
set_property -dict { PACKAGE_PIN A13   IOSTANDARD LVCMOS33 } [get_ports {d[1]}];
set_property -dict { PACKAGE_PIN A14   IOSTANDARD LVCMOS33 } [get_ports {d[0]}];
set_property -dict { PACKAGE_PIN A18   IOSTANDARD LVCMOS33 } [get_ports {a[2]}];
set_property -dict { PACKAGE_PIN B16   IOSTANDARD LVCMOS33 } [get_ports {a[1]}];
set_property -dict { PACKAGE_PIN B17   IOSTANDARD LVCMOS33 } [get_ports {a[0]}];

set_property -dict { PACKAGE_PIN D14   IOSTANDARD LVCMOS33 } [get_ports {sw}];
```

实验效果图如下：

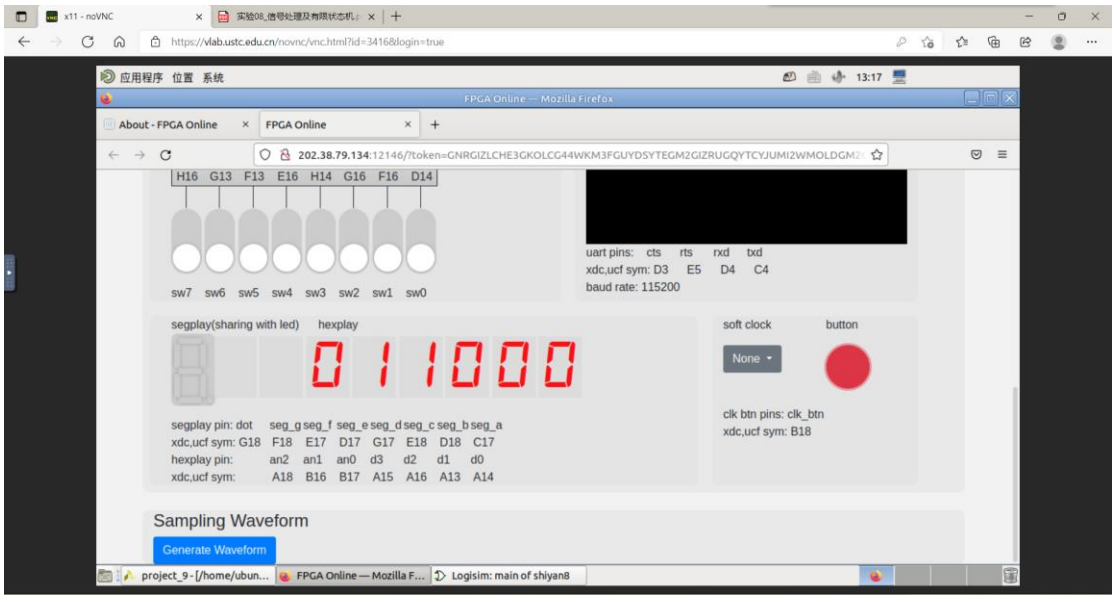


输入 1100：


FPGA Online — Mozilla Firefox
About - FPGA Online
FPGA Online
202.38.79.134:12146/?token=GNRGIZLCHE3GKOLCG44WKM3FGUYDSYTEGM2GIZRUGQYTCYJUMI2WMOLDGM2
H16 G13 F13 E16 H14 G16 F16 D14
sw7 sw6 sw5 sw4 sw3 sw2 sw1 sw0
uart pins: cts rts rxd txd
xdc,ucf sym: D3 E5 D4 C4
baud rate: 115200
segplay(sharing with led)
hexplay
100001
soft clock
button
None
segplay pin: dot seg_g seg_f seg_e seg_d seg_c seg_b seg_a
xdc,ucf sym: G18 F18 E17 D17 G17 E18 D18 C17
hexplay pin: an2 an1 an0 d3 d2 d1 d0
xdc,ucf sym: A18 B16 B17 A15 A16 A13 A14
clk btn pins: clk_btn
xdc,ucf sym: B18
Sampling Waveform
Generate Waveform


FPGA Online — Mozilla Firefox
About - FPGA Online
FPGA Online
202.38.79.134:12146/?token=GNRGIZLCHE3GKOLCG44WKM3FGUYDSYTEGM2GIZRUGQYTCYJUMI2WMOLDGM2
H16 G13 F13 E16 H14 G16 F16 D14
sw7 sw6 sw5 sw4 sw3 sw2 sw1 sw0
uart pins: cts rts rxd txd
xdc,ucf sym: D3 E5 D4 C4
baud rate: 115200
segplay(sharing with led)
hexplay
411100
soft clock
button
None
segplay pin: dot seg_g seg_f seg_e seg_d seg_c seg_b seg_a
xdc,ucf sym: G18 F18 E17 D17 G17 E18 D18 C17
hexplay pin: an2 an1 an0 d3 d2 d1 d0
xdc,ucf sym: A18 B16 B17 A15 A16 A13 A14
clk btn pins: clk_btn
xdc,ucf sym: B18
Sampling Waveform
Generate Waveform

再输入 0：


FPGA Online — Mozilla Firefox
About - FPGA Online
FPGA Online
202.38.79.134:12146/?token=GNRGIZLCHE3GKOLCG44WKM3FGUYDSYTEGM2GIZRUGQYTCYJUMI2WMOLDGM2
H16 G13 F13 E16 H14 G16 F16 D14
sw7 sw6 sw5 sw4 sw3 sw2 sw1 sw0
uart pins: cts rts rxd txd
xdc,ucf sym: D3 E5 D4 C4
baud rate: 115200
segplay(sharing with led)
hexplay
011000
soft clock
button
None
segplay pin: dot seg_g seg_f seg_e seg_d seg_c seg_b seg_a
xdc,ucf sym: G18 F18 E17 D17 G17 E18 D18 C17
hexplay pin: an2 an1 an0 d3 d2 d1 d0
xdc,ucf sym: A18 B16 B17 A15 A16 A13 A14
clk btn pins: clk_btn
xdc,ucf sym: B18
Sampling Waveform
Generate Waveform

【总结与思考】

本次实验学习了有限状态机和它的三段式写法，已经利用 logisim 搭建相关电路。verilog 代码量较大，任务较多，完成时间比较长，不过整体逻辑不是太过复杂（有了实验 7 的基础）。无特别建议。