# 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：牛庆源

学生学号：PB21111733

完成日期：2022.12.5

【实验题目】

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。
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。
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”。
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。

【实验目的】

1. 进一步熟悉 FPGA 开发的整体流程

2. 掌握几种常见的信号处理技巧

3. 掌握有限状态机的设计方法

4. 能够使用有限状态机设计功能电路

【实验环境】

Vscode

Vlab

Vivado

【实验练习】

## 题目一

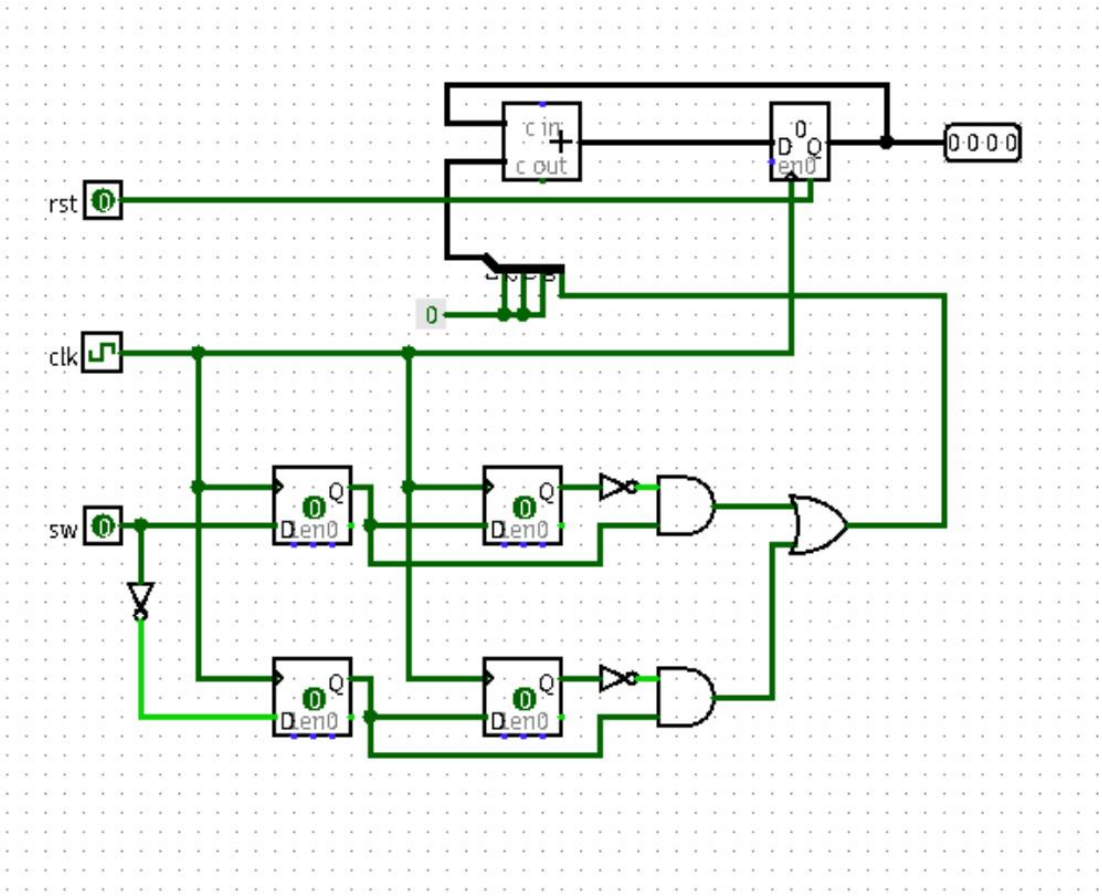
三段式写法第一段生成次态，第二段时序逻辑赋值，第三段组合逻辑输出。

```
module test(
    input clk, rst,
    output led);
    reg [1:0] cur, next;
    // 第一段，组合逻辑生成下一状态
    always @(*) begin
        case(cur)
            0:next = 1;
            1:next = 2;
            2:next = 3;
            3:next = 0;
        endcase
    end
    // 第二段，时序逻辑实现复位或者是变为下一状态
    always @(posedge clk or posedge rst) begin
    if (rst)
        cur <= 0;
    else
        cur <= next;
    end
    // 第三段，生成输出信号
    assign led = (cur == 3) ? 1 : 0;
```

```
endmodule
```

题目二

首先用 step 中给出的信号处理方式存储 sw 信号，高变低低变高均有效于是 sw 和~sw 均处理，最后或门连接到加法器即可，再通过加法器和寄存器寄存每一次加法的结果，rst 用于清空寄存器。




题目三

首先将 btn 按钮信号转换成一个周期的脉冲信号，通过 count 计数，rst 复位值为 1f，sw 控制累加或者累减，最后采用分时复用的方式进行显示。

Xdc 文件将 rst 信号约束在 sw1 上。

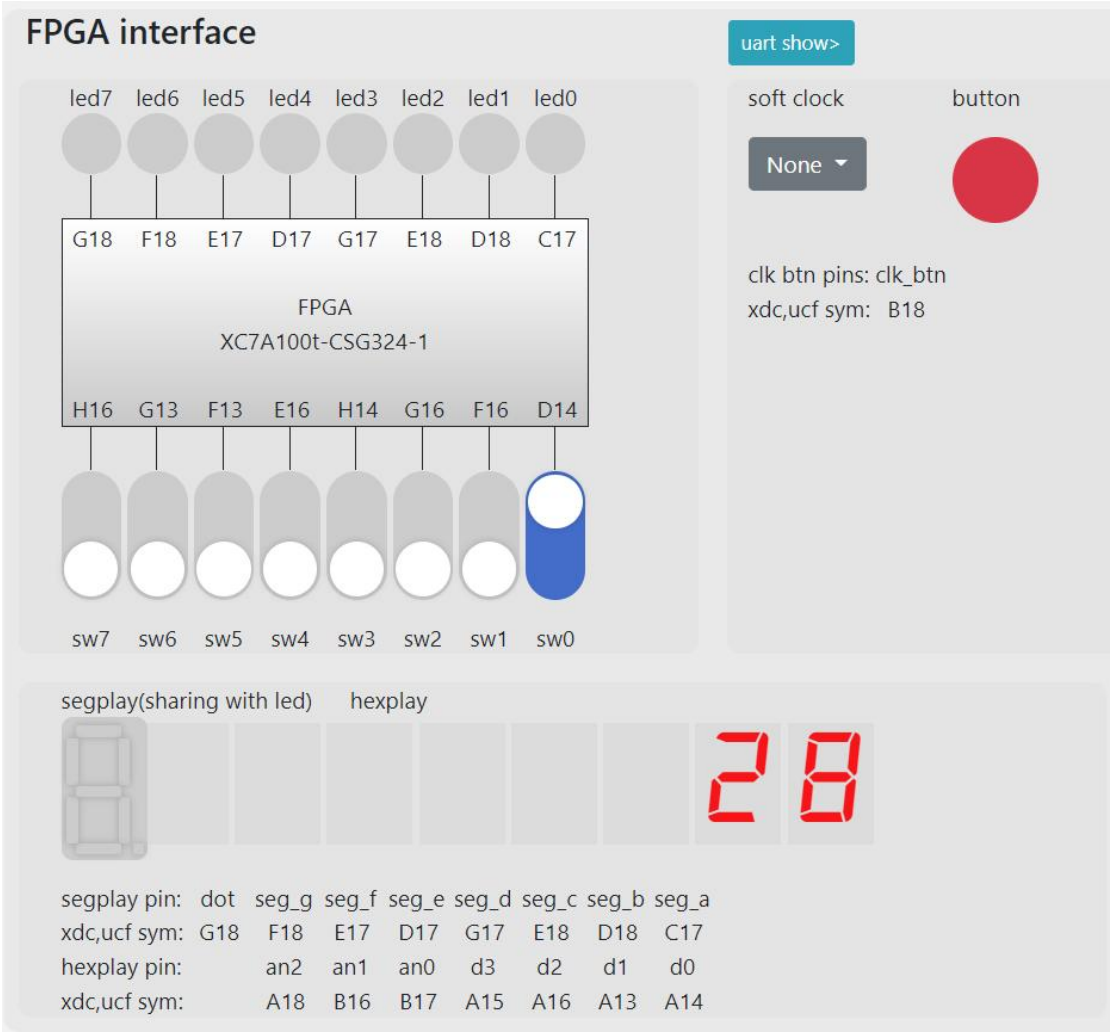
代码如下：

```
module lab8_3(
    input clk, rst, sw, btn,
    output reg [2:0] hexplay_an,
    output reg [3:0] hexplay_data);
    // 按钮信号转为持续一个周期的脉冲信号
    reg temp1, temp2;
    wire btn_pulse;
    always @(posedge clk) temp1 <= btn;
    always @(posedge clk) temp2 <= temp1;
    assign btn_pulse = temp1 & (~temp2);
    // 计数
    reg [7:0] count;
    always @(posedge clk) begin
        if (rst)
            count <= 8'h1f;
        else if (btn_pulse) begin
            if (sw)
                count <= count + 1;
            else
                count <= count - 1;
        end
    end
    // 分时复用
    reg [2:0] cnt;
    always @(posedge clk) begin
        cnt <= cnt + 1;
        case (cnt)
            3'b011: begin
                hexplay_an <= 3'b000;
                hexplay_data <= count[3:0];
            end
            3'b111: begin
                hexplay_an <= 3'b001;
                hexplay_data <= count[7:4];
            end
        endcase
    end
endmodule
```

Xdc 文件如下：

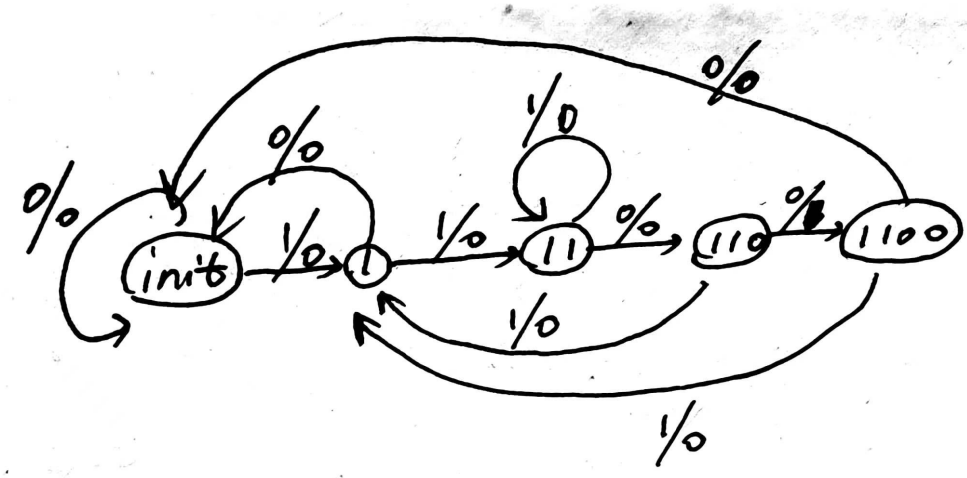
```
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { clk }];
set_property -dict { PACKAGE_PIN B18   IOSTANDARD LVCMOS33 } [get_ports { btn }];
set_property -dict { PACKAGE_PIN D14   IOSTANDARD LVCMOS33 } [get_ports { sw }];
set_property -dict { PACKAGE_PIN F16   IOSTANDARD LVCMOS33 } [get_ports { rst }];

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

烧写 FPGA 结果如下：




题目四

首先根据题目要求画出状态转换图如下：




写出输入输出：

```
module lab8_4(
input clk, rst, sw, btn,
output reg [2:0] hexplay_an,
output reg [3:0] hexplay_data);
```

三段式写法：

第一段是根据状态转换图写出次态（这里将 init 到 1100 分别转变为三位的状态码 000 到 100）。

```
// 第一部分：生成次态
reg [2:0] cur, next;
always @(*) begin
    if (sw)
    case (cur)
        3'b000: next = 1;
        3'b001: next = 2;
        3'b010: next = 2;
        3'b011: next = 1;
        3'b100: next = 1;
        default: next = 0;
    endcase
    else
    case (cur)
        3'b010: next = 3;
        3'b011: next = 4;
        default: next = 0;
    endcase
end
```

第二段是时序逻辑改变状态。（生成最后输入的四位以及当前状态的状态码 0~4），方法是首先生成一个 btn 周期信号，然后在 clk 时序期间 rst 设置复位值，并在 btn 有效时用次态给现态赋值并且将最近输入前移一位并赋自己输入的当前位给最后一位。

```
46  // 第二部分：时序逻辑状态转换
47  reg temp1, temp2;
48  wire btn_pulse;
49  always @(posedge clk) temp1 <= btn;
50  always @(posedge clk) temp2 <= temp1;
51  assign btn_pulse = temp1 & (~temp2);
52  reg [3:0] recent;
53  always @(posedge clk) begin
54      if (rst) begin
55          cur <= 0;
56          recent <= 0;
57      end
58      else if (btn_pulse) begin
59          cur <= next;
60          recent <= recent << 1;
61          recent[0] <= sw;
62      end
63      else begin
64          cur <= cur;
65          recent <= recent;
66      end
67  end
```

第三段是组合逻辑生成输出信号。生成 out 加一的条件是状态码到 4 即三位状态码最高位为 1。

```
68  // 第三部分：输出
69  reg state1, state2;
70  wire state_pulse;
71  always @(posedge clk) state1 <= cur[2];
72  always @(posedge clk) state2 <= state1;
73  assign state_pulse = state1 & (~state2);
74  reg [3:0] out;
75  always @(posedge clk) begin
76      if (rst)
77          out <= 0;
78      else if (state_pulse)
79          out <= out + 1;
80      else
81          out <= out;
82  end
```

最后采用时分复用的方式显示。

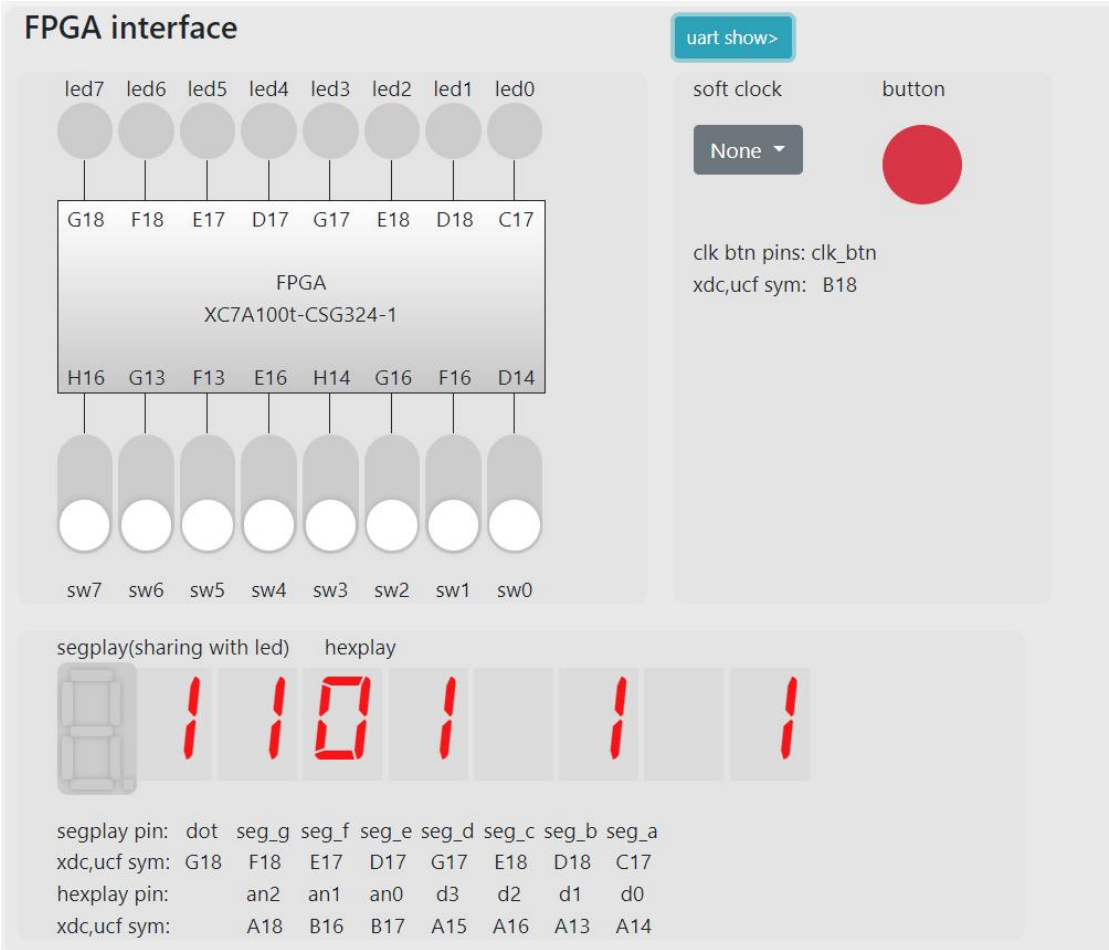
```
83  reg [4:0] cnt;
84  always @(posedge clk) begin
85      if (cnt == 5'b10111)
86          cnt <= 5'b0;
87      else
88          cnt <= cnt + 1;
89      case (cnt[4:2])
90          0: begin
91              hexplay_an <= 3'b000;
92              hexplay_data <= {1'b0, cur};
93          end
94          1: begin
95              hexplay_an <= 3'b010;
96              hexplay_data <= out;
97          end
98          2: begin
99              hexplay_an <= 3'b100;
100             hexplay_data <= {3'b0, recent[0]};
101         end
102         3: begin
103             hexplay_an <= 3'b101;
104             hexplay_data <= {3'b0, recent[1]};
105         end
```

```
            3: begin
                hexplay_an <= 3'b101;
                hexplay_data <= {3'b0, recent[1]};
            end
            4: begin
                hexplay_an <= 3'b110;
                hexplay_data <= {3'b0, recent[2]};
            end
            5: begin
                hexplay_an <= 3'b111;
                hexplay_data <= {3'b0, recent[3]};
            end
            default: hexplay_data <= 4'b0;
        endcase
    end
endmodule
```

Xdc 文件如下：

```
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { clk }];
set_property -dict { PACKAGE_PIN B18   IOSTANDARD LVCMOS33 } [get_ports { btn }];
set_property -dict { PACKAGE_PIN D14   IOSTANDARD LVCMOS33 } [get_ports { sw }];
set_property -dict { PACKAGE_PIN F16   IOSTANDARD LVCMOS33 } [get_ports { rst }];

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

烧写 FPGA 结果如下：




【总结与思考】

1. 熟悉了 fpga 使用，会利用状态转换图写出 verilog 代码，学会生成特定的信号。

2. 难易程度为中等，代码量相较之前提高不少不过难度上来看为中等难度的代码编写。

3. 任务量中上，需要花时间画状态图写代码等。

4. 手册很好解释了问题。无改进建议。