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：傅申

学生学号：PB20000051

完成日期：2021 年 12 月 9 日

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】

- Windows PC
- Microsoft Visual Studio Code
- Logisim
- Xilinx Design Tools Vivado HL Design Edition 2019.1

【实验练习】

题目 1: Step 5. 中的代码共有 4 个状态: 0, 1, 2, 3, 分别编码为 00, 01, 10, 11. 其状态跳转图如图 1.

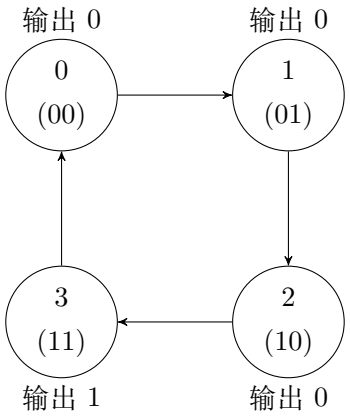


图 1: Step 5. 的状态跳转图

只有当状态为 3 时才输出 1, 其他状态都输出 0. 由此写出有限状态机的代码如下

代码 1: Step 5. 中代码改写为有限状态机的形式

```
1 module test(
2     input clk, rst,
3     output led
```

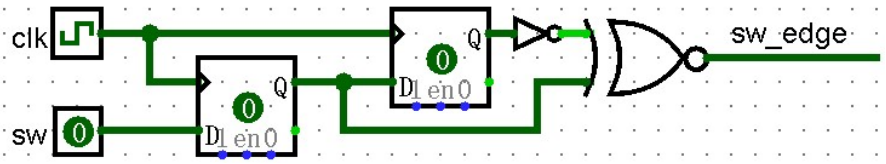
```
);
parameter STATE_0 = 2'h0;
parameter STATE_1 = 2'h1;
parameter STATE_2 = 2'h2;
parameter STATE_3 = 2'h3;
reg [1:0] curr_state, next_state;
// FSM Part 1
always @(*) case (curr_state)
    STATE_0: next_state = STATE_1;
    STATE_1: next_state = STATE_2;
    STATE_2: next_state = STATE_3;
    STATE_3: next_state = STATE_0;
    default: next_state = STATE_0;
endcase
// FSM Part 2
always @(posedge clk or posedge rst) begin
    if (rst) curr_state <= STATE_0;
    else curr_state <= next_state;
end
// FSM Part 3
assign led = (curr_state == STATE_3);
endmodule
```

题目 2: 首先, 采用 Step 2. 中的方式取 sw 信号的边缘, 将最后的与门换成同或门, 即可同时取到 sw 的上升沿与下降沿, 将 sw 的信号边缘连接到计数器电路加法器的进位输入, 然后设计一个位宽为 4 bit 的 D 触发器, 其复位值为 `4'b0000`. 将计数器电路连接到触发器 D 输入. 最后得到的电路如图 2(c). 电路设计过程如图 2.

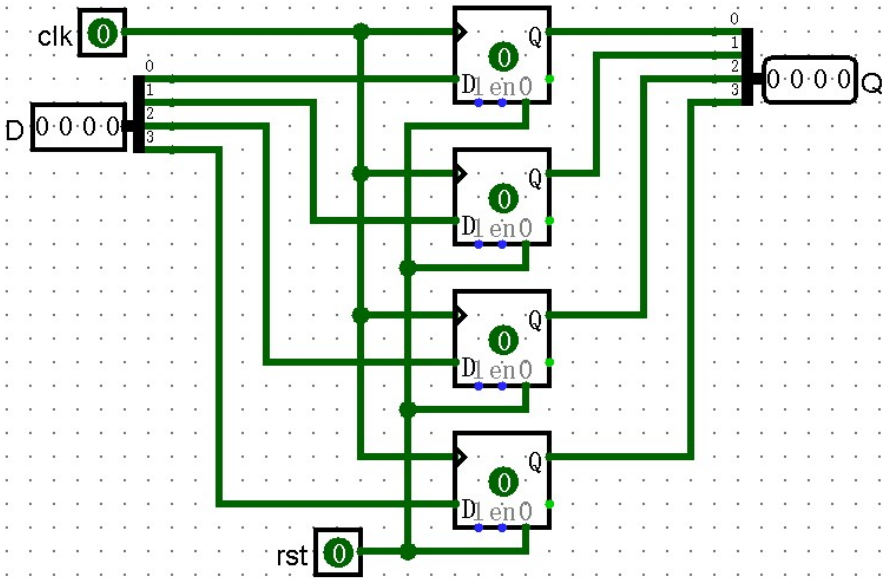
题目 3: 因为要根据按钮按下的瞬间进行计数, 所以首先需要取 `btn` 信号的边缘. 取信号边缘的模块如下

代码 2: 取信号边缘的模块

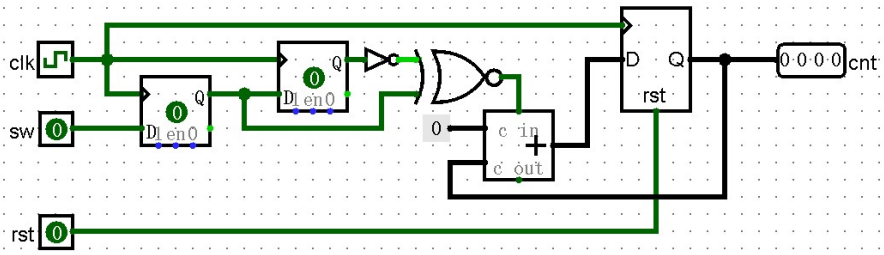
```
// ./Prob.3/Prob.3.srcs/sources_1/new/SignalEdge.v
module signal_edge(
    input clk,
    input signal,
    output signal_edge
    );
reg signal_r1, signal_r2;
always @(posedge clk) signal_r1 <= signal;
always @(posedge clk) signal_r2 <= signal_r1;
assign signal_edge = signal_r1 & (~signal_r2);
endmodule
```



(a) 取 sw 信号边缘



(b) 位宽为 4 bit 的 D 触发器



(c) 最终电路

图 2: 计数器电路

在 btn_edge 信号有效时根据 sw 的情况进行计数. 因为需要用到 2 个数码管, 所以我们需要生成一个 100 Hz 的时钟信号, 并采用时分复用的方式进行显示. 最后的 Verilog 代码如代码 3.

代码 3: 题目 3. 的 Verilog 代码

```
// ./Prob.3/Prob.3.srcs/sources_1/new/Counter.v
module counter(
    input clk_100mhz,
    input sw,
    input btn,
    input rst,
    output reg hexplay_an,
    output reg [3:0] hexplay_data
    );
// generate a 100 hz clock
wire clk_100hz;
reg [19:0] clk_cnt;
assign clk_100hz = (clk_cnt >= 500000);
always @(posedge clk_100mhz)
begin
    if (clk_cnt >= 1000000) clk_cnt = 0;
    else clk_cnt = clk_cnt + 20'h00001;
end
// get the edge of btn signal
wire btn_edge;
signal_edge getBtnEdge(.clk(clk_100mhz),
                       .signal(btn),
                       .signal_edge(btn_edge));
// counter
reg [7:0] cnt;
always @(posedge clk_100mhz)
begin
    if (rst) cnt <= 8'h1f;
    else if (btn_edge)
    begin
        if (sw)
        begin
            if (cnt >= 8'hff) cnt <= 8'h00;
            else cnt <= cnt + 8'h01;
        end
        else
        begin
            if (cnt == 8'h00) cnt <= 8'hff;
```

```
            else cnt <= cnt - 8'h01;
        end
    end
end
// segplay
always @(posedge clk_100mhz)
begin
    if (clk_100hz)
    begin
        hexplay_an = 1'b1;
        hexplay_data = cnt[7:4];
    end
    else
    begin
        hexplay_an = 1'b0;
        hexplay_data = cnt[3:0];
    end
end
endmodule
```

部分 xdc 约束文件代码如下

代码 4: 题目 3. 的 xdc 文件

```
## Clock signal
#IO_L12P_T1_MRCC_35 Sch=clk100mhz
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { clk_100mhz }];

## FPGAOL SWITCH

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

## FPGAOL HEXPLAY

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 }];

## FPGAOL BUTTON & SOFT_CLOCK

set_property -dict { PACKAGE_PIN B18   IOSTANDARD LVCMOS33 } [get_ports { btn }];
```

最后生成比特流后烧写在 FPGAOL 平台, 效果如图 3.

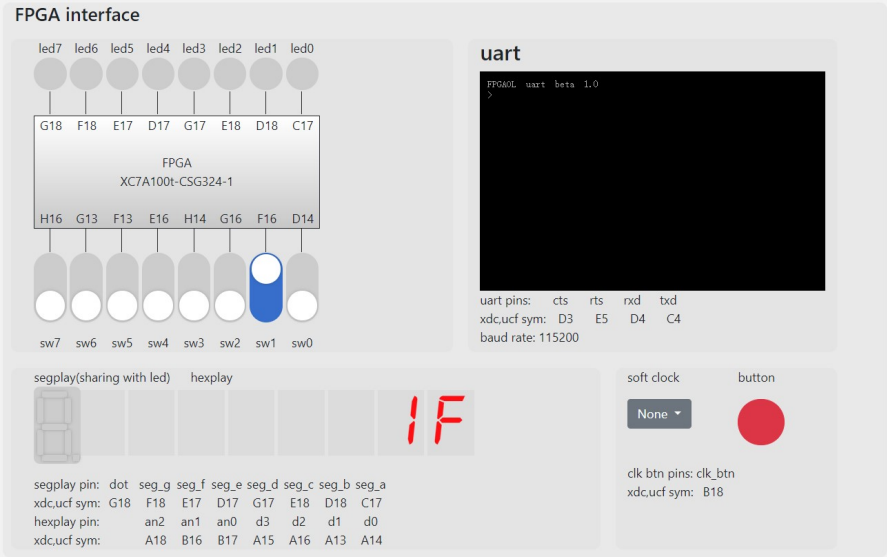


图 3: 题目 3. 在 FPGA 上的显示效果

题目 4: 由题意可知, 有限状态机有 4 个状态, 对应输入序列以 `0`, `1`, `11`, `110` 结尾, 分别编码为 00, 01, 10, 11. 只有在状态 11 且输入的 `sw` 为 0 时, 计数器才加 1. 状态转移图如图 4. 由题意可知, 我们仍然需要取 `btn` 信号的边缘, 所以仍需要用到代码 2 的模

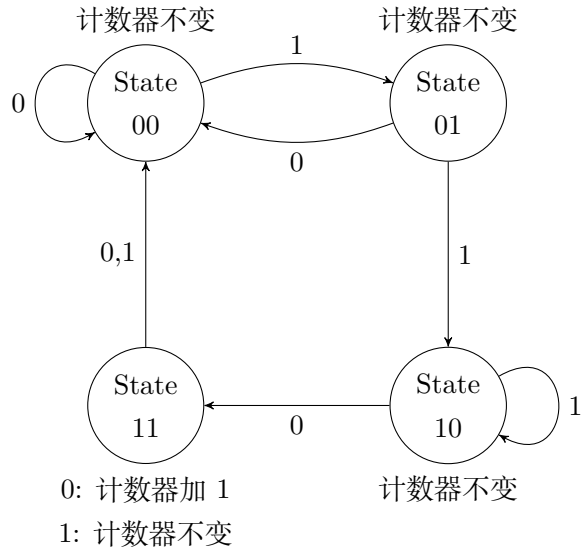


图 4: 题目 4. 的状态转移图

块. 而需要驱动 6 个数码管, 我们则需要 300 Hz 的时钟/脉冲信号, 但是 300 Hz 不好实现, 因此这里使用 400 Hz 的脉冲信号进行驱动. 最后的 Verilog 模块代码如下

代码 5: 题目 4. 的 Verilog 模块

```
1 module array_detect(
2     input clk_100mhz,
3     input btn,
4     input sw,
5     output reg [3:0] hexplay_data,
```

```
6       output reg [2:0] hexplay_an
7       );
8   // generate a 400 Hz pulse
9   wire pulse_400hz;
10  reg [18:0] pulse_cnt;
11
12  assign pulse_400hz = (pulse_cnt == 19'h00001);
13  always @(posedge clk_100mhz)
14  begin
15      if (pulse_cnt >= 19'h3d090) pulse_cnt <= 19'h00000;
16      else pulse_cnt <= pulse_cnt + 19'h00001;
17  end
18
19  // get btn signal edge
20  wire btn_edge;
21  signal_edge getBtnEdge(.clk(clk_100mhz),
22                         .signal(btn),
23                         .signal_edge(btn_edge));
24
25  // input array process
26  reg [3:0] input_array;
27  initial input_array = 4'h0;
28  always @(posedge clk_100mhz)
29  begin
30          if (btn_edge) input_array <= {input_array[2:0], sw};
31  end
32
33  // FSM
34  parameter STATE_0 = 2'b00;
35  parameter STATE_1 = 2'b01;
36  parameter STATE_2 = 2'b10;
37  parameter STATE_3 = 2'b11;
38
39  reg [1:0] curr_state, next_state;
40  reg [3:0] cnt;
41  initial cnt <= 4'h0;
42  // Part 1
43  always @(*)
44  begin
45      if (sw)
46      begin
47          case(curr_state)
```

```
48              STATE_0: next_state = STATE_1;
49              STATE_1: next_state = STATE_2;
50              STATE_2: next_state = STATE_2;
51              STATE_3: next_state = STATE_0;
52              default: next_state = STATE_0;
53          endcase
54      end
55      else
56      begin
57          case(curr_state)
58              STATE_0: next_state = STATE_0;
59              STATE_1: next_state = STATE_0;
60              STATE_2: next_state = STATE_3;
61              STATE_3: next_state = STATE_0;
62              default: next_state = STATE_0;
63          endcase
64      end
65  end
66  // Part 2
67  always @(posedge clk_100mhz)
68      if (btn_edge) curr_state <= next_state;
69  // Part 3
70  always @(posedge clk_100mhz)
71  begin
72      if (btn_edge)
73      begin
74          if (curr_state == STATE_3 && sw == 0)
75          begin
76              if (cnt >= 4'hf) cnt <= 4'h0;
77              else cnt <= cnt + 4'h1;
78          end
79      end
80  end
81
82  // Segplay
83  always @(posedge clk_100mhz)
84  begin
85      if (pulse_400hz)
86      begin
87          if (hexplay_an >= 3'h7)
88              hexplay_an <= 3'h0;
89          else if (hexplay_an == 3'h0 ||
```

```
                    hexplay_an == 3'h5)
                hexplay_an <= hexplay_an + 3'h2;
        else hexplay_an <= hexplay_an + 3'h1;
    end
end
always @(posedge clk_100mhz)
begin
    case (hexplay_an)
        3'h0: hexplay_data <= cnt;
        3'h2: hexplay_data <= {3'b000, input_array[0]};
        3'h3: hexplay_data <= {3'b000, input_array[1]};
        3'h4: hexplay_data <= {3'b000, input_array[2]};
        3'h5: hexplay_data <= {3'b000, input_array[3]};
        3'h7: hexplay_data <= {2'b00, curr_state};
        default: hexplay_data <= 4'h0;
    endcase
end
endmodule
```

可以看出, 我们使用了两端的两个数码管以及中间的四个数码管, 从左到右依次为: 状态编码, 最近输入四位数值, 目标序列个数. 工程的 xdc 约束文件代码如下

代码 6: 题目 4. 的 xdc 文件

```
## Clock signal
#IO_L12P_T1_MRCC_35 Sch=clk100mhz
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { clk_100mhz }];
## FPGAOL SWITCH

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

## FPGAOL HEXPLAY

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] }];

## FPGAOL BUTTON & SOFT_CLOCK

set_property -dict { PACKAGE_PIN B18   IOSTANDARD LVCMOS33 } [get_ports { btn }];
```

生成比特流后烧写如 FPGAOL 平台, 当输入序列为 “0011001110011” 时, 显示效果如图 5.

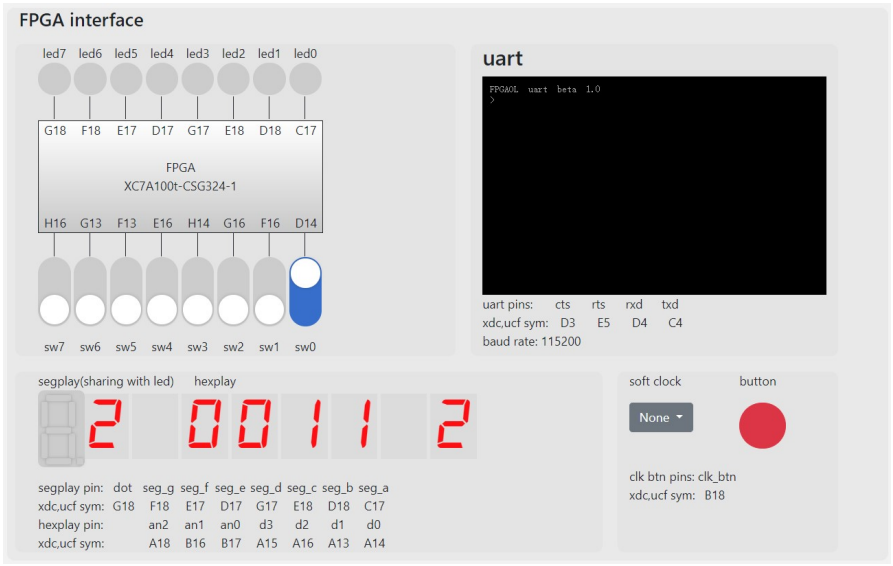


图 5: 题目 4. 在 FPGA 上的显示效果

【总结与思考】

收获 学习了如何对输入信号进行处理, 能够构造简单的有限状态机.

难易程度 较难

任务量 中等

建议 暂无