实验报告

曾锦鸿
2021k8009929009

实验目的：

1、熟悉 verilog 编程、调试

2、熟悉状态机的工作原理，能熟练编写状态机程序

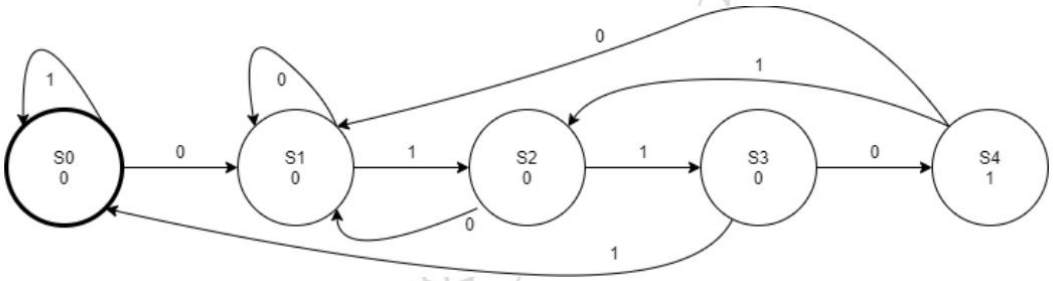
实验环境：

vivado版本号为2019.2

原理说明：

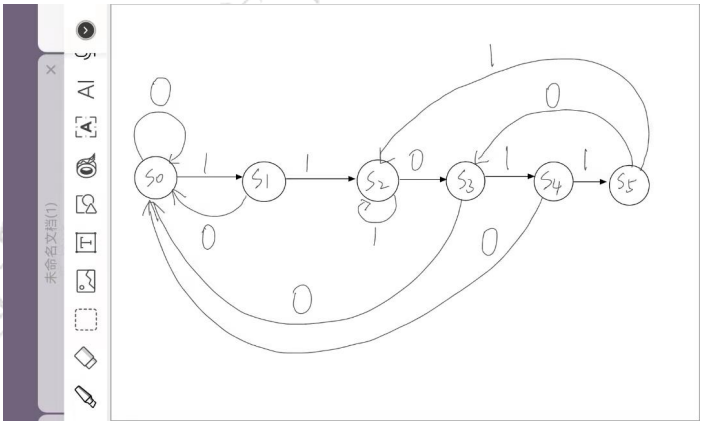
1、检测序列0110的状态机

状态机转移图如下所示：



2、检测序列11011的状态机

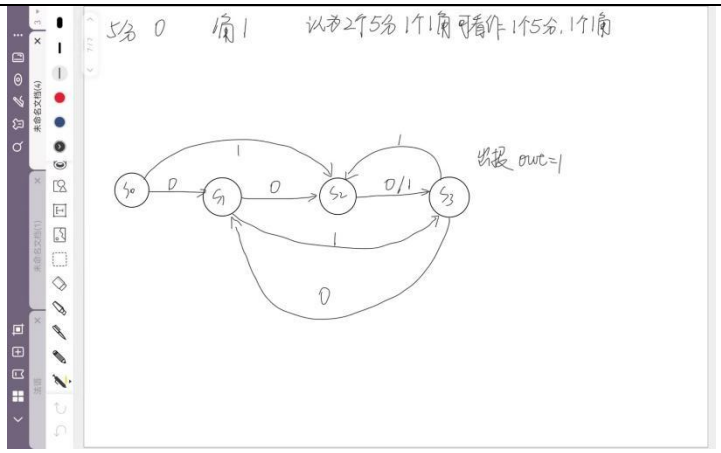
状态机转移图如下所示：



3、周期性输出0010111001001的序列信号发生电路

用一个寄存器存储要输出的序列，每次时钟上升沿输出最高位，将最高位移动到最低位。

4、报纸售卖机



这里认为输入0为5分钱，输入1为1角钱，而且认为2个5分、1个1角并非非法输入。售货机不着钱，而且多出的钱并不会存到下次。

接口定义：

1、检测序列0110的状态机：

名称	方向	位宽	功能描述
clk	IN	1	时钟信号
in	IN	1	输入信号
rstn	IN	1	复位信号的非（当它为0时，复位）
out	OUT	1	状态机的输出

2、检测序列11011的状态机：

名称	方向	位宽	功能描述
clk	IN	1	时钟信号
in	IN	1	输入信号
rstn	IN	1	复位信号的非（当它为0时，复位）
out	OUT	1	状态机的输出

3、周期输出0010111001001的序列信号发生电路：

名称	方向	位宽	功能描述
clk	IN	1	时钟信号
rstn	IN	1	复位信号的非（当它为0时，复位）
out	OUT	1	状态机的输出

4、报纸售货机：

名称	方向	位宽	功能描述
clk	IN	1	时钟信号
in	IN	1	输入信号
rstn	IN	1	复位信号的非（当它为0时，复位）
out	OUT	1	状态机的输出
cur_state	OUT	3	状态机当前状态

调试过程及结果波形：

本次实验较为简单，写完后并未出过bug，所以无调试过程。

1、检测序列0110的状态机：

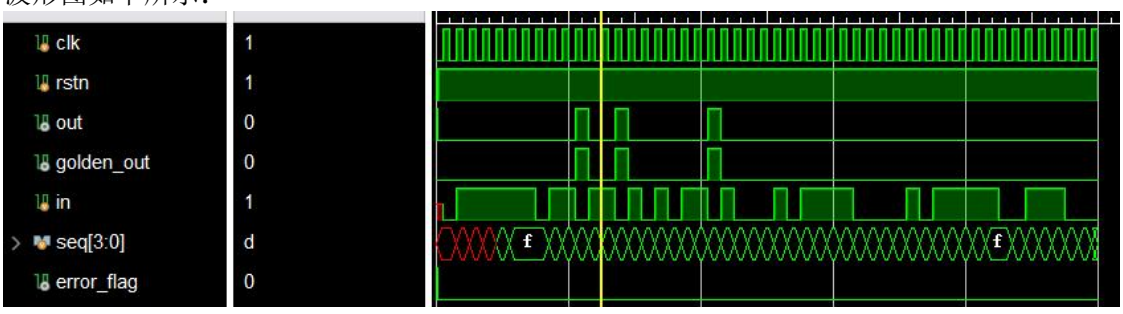
激励文件：

```
module test_fsm0110(

    );
    reg clk;
    reg rstn;
    wire out;
    wire golden_out;
    reg in;
    reg [3:0] seq;
    FSM0110 u_FSM(
        .clk(clk),
        .rstn(rstn),
        .in(in),
        .out(out)
    );
    initial begin
        clk = 0;
        rstn = 1;
        #0.1 rstn = 0;
        #1.1 rstn = 1;
    end
    always begin
        #10 clk = ~clk;
    end
    always @(posedge clk) begin
            in <= $random() % 2;
    end
    always @(negedge rstn or posedge clk) begin
        if(!rstn)begin
            seq <= 4'bxxxx;
        end else begin
            seq <= {seq[2:0],in};
        end
    end
    assign golden_out = {seq[2:0],in} === 4'b0110;
    wire error_flag;
    assign error_flag = golden_out != out;
endmodule
```

在激励文件中，我自行编写了check，这样我们只需要确认error_flag未被拉高过，则说明状态机正确。

波形图如下所示：



2、检测序列11011的状态机：

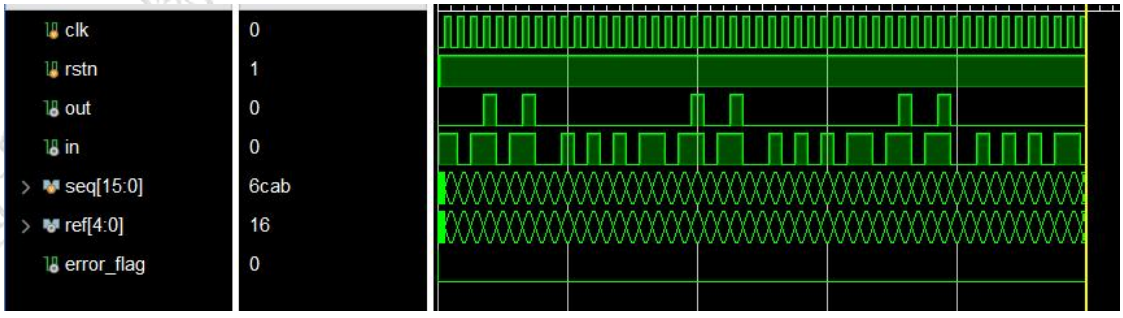
激励文件：

```
module test_FSM11011(

    );
    reg clk;
    reg rstn;
    wire out;
    wire in;
    reg [15:0] seq;
    FSM11011 u_FSM(
        .clk(clk),
        .rstn(rstn),
        .in(in),
        .out(out)
    );
    assign in = seq[15];
    initial begin
        clk = 0;
        rstn = 1;
        seq = 16'b1101101100101010;
        #0.1 rstn = 0;
        #1.1 rstn = 1;
    end
    always begin
        #10 clk = ~clk;
    end
    always @(posedge clk) begin
        seq <= {seq[14:0],seq[15]};
    end
    wire [4:0] ref;
    assign ref = {seq[3:0],seq[15]};
    assign error_flag = ((ref == 5'b11011) && (out == 0)) || ((ref != 5'b11011) && (out == 1));
endmodule
```

题中要求编写的test存在11011的重叠，我们可以周期性地输出一个11011重叠的序列（本例中为1101101100101010），然后让seq的最高位为输入in。由于序列确定，所以out确定，可以自行生成check，检查error_flag未被拉高则说明状态机正确。

波形图如下所示：



3、周期输出0010111001001的序列信号发生电路：

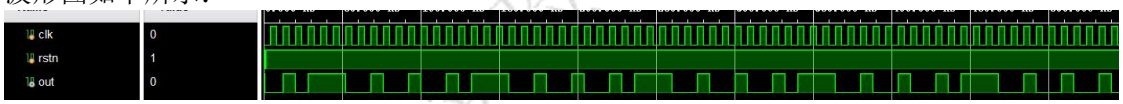
激励文件：

```
module test_seq(

    );
    reg clk;
    reg rstn;
    wire out;
    create_seq u_seq(
        .clk(clk),
        .rstn(rstn),
        .out(out)
    );
    initial begin
        clk = 0;
        rstn = 1;
        #0.1 rstn = 0;
        #1.1 rstn = 1;
    end
    always begin
        #4 clk = ~clk;
    end
endmodule
```

因为这是生成序列，不方便编写check,所以就只对clk、rstn等输入信号做了处理。

波形图如下所示：



它确实可以周期地输出0010111001001。

4、报纸售货机

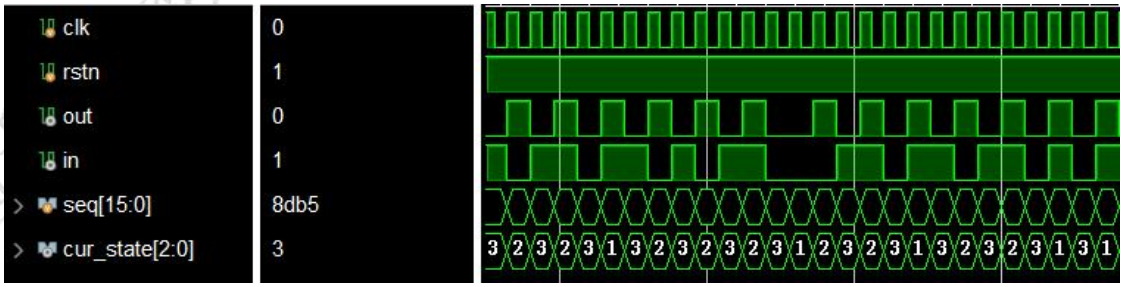
激励文件：

```
module test_sale_v1(

    );
    reg clk;
    reg rstn;
    wire out;
    wire in;
    reg [15:0] seq;
    wire [2:0] cur_state;
    sale_v1 u_sale(
        .clk(clk),
        .rstn(rstn),
        .in(in),
        .state(cur_state),
        .out(out)
    );
    assign in = seq[15];
    initial begin
        clk = 0;
        rstn = 1;
        seq = 16'b0110110110101100;
        #0.1 rstn = 0;
        #1.1 rstn = 1;
    end
    always begin
        #4 clk = ~clk;
    end
    always @(posedge clk) begin
        seq <= {seq[14:0],seq[15]};
    end
endmodule
```

因为这是报纸购买，不方便编写check,所以就只对clk、rstn等输入信号做了处理。

波形图如下所示：



下一个状态为3时，out为1，说明波形图符合预期。

实验总结：

本次实验总体来说并不困难，只需要画出状态转移图，将其转化为verilog代码即可。

源代码：

1、检测序列0110的状态机：

```
module FSM0110(
    input clk,
    input rstn,
    input in,
    output out
    );
    localparam S_0 = 3'd0;
    localparam S_1 = 3'd1;
    localparam S_2 = 3'd2;
    localparam S_3 = 3'd3;
    localparam S_4 = 3'd4;
    reg [2:0] cur_state;
    reg [2:0] nxt_state;
    always @(negedge rstn or posedge clk) begin
        if(!rstn) begin
            cur_state <= S_0;
        end else begin
            cur_state <= nxt_state;
        end
    end
    always @(in or cur_state) begin
        case(cur_state)
            S_0:begin
                if(in == 1) begin
                    nxt_state <= S_0;
                end else begin
                    nxt_state <= S_1;
                end
            end
            S_1:begin
                if(in == 1) begin
                    nxt_state <= S_2;
                end else begin
                    nxt_state <= S_1;
                end
            end
            S_2:begin
                if(in == 1) begin
                    nxt_state <= S_3;
```

```
                end else begin
                    nxt_state <= S_1;
                end
            end
            S_3:begin
                if(in == 1) begin
                    nxt_state <= S_0;
                end else begin
                    nxt_state <= S_4;
                end
            end
            S_4:begin
                if(in == 1)begin
                    nxt_state <= S_2;
                end else begin
                    nxt_state <= S_1;
                end
            end
            default:begin
                nxt_state <= S_0;
            end
        endcase
    end
    assign out = (nxt_state == S_4);
endmodule
```

2、检测序列11011的状态机：

```
23  module FSM11011(
24      input clk,
25      input rstn,
26      input in,
27      output out
28      );
29      localparam S_0 = 3'd0;
30      localparam S_1 = 3'd1;
31      localparam S_2 = 3'd2;
32      localparam S_3 = 3'd3;
33      localparam S_4 = 3'd4;
34      localparam S_5 = 3'd5;
35      reg [2:0] cur_state;
36      reg [2:0] nxt_state;
37      always @(negedge rstn or posedge clk) begin
38          if(!rstn) begin
39              cur_state <= S_0;
40          end else begin
41              cur_state <= nxt_state;
42          end
43      end
44      always @(in or cur_state) begin
45          case(cur_state)
46              S_0:begin
47                  if(in == 1) begin
48                      nxt_state <= S_1;
49                  end else begin
50                      nxt_state <= S_0;
51                  end
52              end
53              S_1:begin
54                  if(in == 1) begin
55                      nxt_state <= S_2;
56                  end else begin
57                      nxt_state <= S_0;
58                  end
59              end
```

```
            S_2:begin
                if(in == 1) begin
                    nxt_state <= S_2;
                end else begin
                    nxt_state <= S_3;
                end
            end
            S_3:begin
                if(in == 1) begin
                    nxt_state <= S_4;
                end else begin
                    nxt_state <= S_0;
                end
            end
            S_4:begin
                if(in == 1)begin
                    nxt_state <= S_5;
                end else begin
                    nxt_state <= S_0;
                end
            end
            S_5:begin
                if(in == 1)begin
                    nxt_state <= S_2;
                end else begin
                    nxt_state <= S_3;
                end
            end
            default:begin
                nxt_state <= S_0;
            end
        endcase
    end
    assign out = (nxt_state == S_5);
endmodule
```

3、周期输出0010111001001的序列信号发生电路：

```
module create_seq(
    input clk,
    input rstn,
    output out
    );
    reg [12:0] seq;
    always @(negedge rstn or posedge clk) begin
        if(!rstn) begin
            seq <= 13'b0010111001001;
        end else begin
            seq <= {seq[11:0],seq[12]};
        end
    end
    assign out = seq[12];
endmodule
```

4、报纸售货机：

```
module sale_v1(
    input clk,
    input rstn,
    input in,
    output [2:0] state,
    output out
    );
    localparam S_0 = 3'd0;
    localparam S_1 = 3'd1;
    localparam S_2 = 3'd2;
    localparam S_3 = 3'd3;
    reg [2:0] cur_state;
    reg [2:0] nxt_state;
    always @(negedge rstn or posedge clk) begin
        if(!rstn) begin
            cur_state <= S_0;
        end else begin
            cur_state <= nxt_state;
        end
    end
    always @(in or cur_state) begin
        case(cur_state)
            S_0:begin
                if(in == 1) begin
                    nxt_state <= S_2;
                end else begin
                    nxt_state <= S_1;
                end
            end
            S_1:begin
                if(in == 1) begin
                    nxt_state <= S_3;
                end else begin
                    nxt_state <= S_2;
                end
            end
            S_2:begin
                nxt_state <= S_3;
            end
            S_3:begin
                if(in == 1) begin
                    nxt_state <= S_2;
                end else begin
                    nxt_state <= S_1;
                end
            end
            default:begin
                nxt_state <= S_0;
            end
        endcase
    end
    assign state = cur_state;
    assign out = (nxt_state == S_3);
endmodule
```