浙江大学实验报告

课程名称：	数字系统	任课老师：	李宇波
实验名称：	Design an electronic lock	实验日期：	2023/4/21

1 实验目的和要求

1.1 实验目的

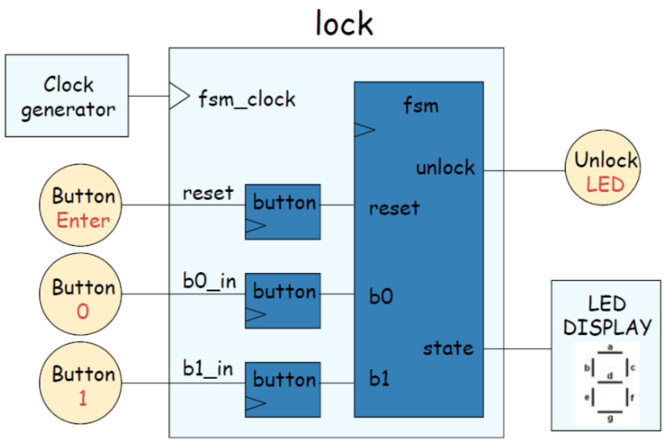
To learn how to implement an Electronic Lock by using FSM.

了解如何用有限状态机来实现密码锁。

1.2 实验要求

Implement the Electronic lock learnt in class with the following function diagram.

用下列函数表实现一个密码锁。



1) Draw the state diagram of the FSM in the above figure.

   绘制上图的有限状态机的状态图。

2) Write the behavior level Verilog module of the FSM.

   编写有限状态机的行为级 Verilog 模型。

3) Write a testbench to simulate the FSM by using the method in Lab 3.

   用 lab3 中学到的方法仿真 FSM。

4) Write the Verilog description of the whole design in the above figure and test it in yourBasys3 board.

   编写整个设计的 Verilog 描述，并用板子测试。

5) Synthesis the FSM by hand to draw the circuit of the FSM with only the D flipflops and AND/OR/NOT gate. Verify the circuit you synthesized and the circuit synthesized by Vivado.

仅用 D 触发器和与或非门设计 FSM 电路。验证你综合的和 vivado 综合的电路。

6) From the synthesized circuit by your own, write the Verilog module of the FSM: For the next-state logic and the output logic, use “assign” statement to describe the combinational logic, and for the D flipflop, use “always” statement to describe it.

   用你自己综合电路写这个 FSM 的 Verilog 语言：对于次态逻辑和输出逻辑，用 assign 语句描述组合逻辑，用 always 语句描述 D 触发器。

7) For the whole design, replace the FSM module designed in step 2) with that designed in step6), and implement the whole design in your board.

   替换 step2 中设计的 FSM 模块，并在板子上实现新的设计。

2 状态图

设计状态图和状态编码如下：

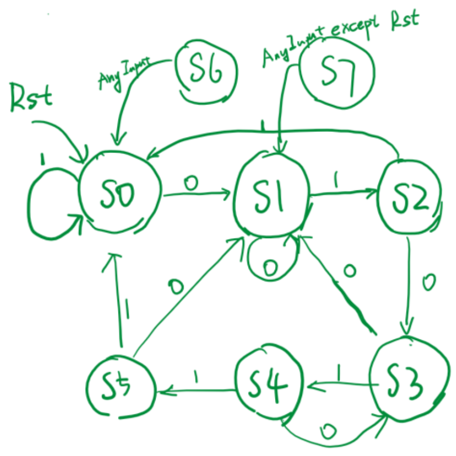


图 1 状态图

Q1Q2Q3	States	Physical Input
000	S0	No input or Reset
001	S1	0
011	S2	01
010	S3	010
110	S4	0101
111	S5	01011

100	S6	Automation
101	S7	Automation

表 1 状态设计

所以对于状态机，可以设计以下输入输出：

$Input\ \ Q_1Q_2Q_3$，key_0，key_1， Rst

$Output\ Q_1^*Q_2^*Q_3^*$， $unlock$

$(Rst = 0)$

$Q_1Q_2Q_3$	$Q_1^*Q_2^*Q_3^*$			$unlock$	LED
	$key\_0 = 1$	$key\_0 = 0\&$ $key\_1 = 1$	$key\_0 = 0\&$ $key\_1 = 0$		
000	001	001	000	0	0000001
001	000	011	001	0	1001111
010	001	110	010	0	0010010
011	010	000	011	0	0000110
100	000	000	000	0	1001100
101	001	001	001	0	0100100
110	010	111	110	0	0100000
111	001	000	111	1	0001111

$(Rst = 1)$

$Q_1Q_2Q_3$ $(Any\ Inputs)$	$Q_1^*Q_2^*Q_3^*$	$unlock$	LED
XXX	000	0	0000001

表 2 真值表

由表格得到输入-输出布尔表达式：

$$unlock = Q_1Q_2Q_3Rst'$$

$$Q_1^* = (key\_0' \cdot key\_1 \cdot Q_2Q_3' + key\_0' \cdot key\_1' \cdot Q_1Q_2)Rst'$$

$$Q_2^* = [key\_0 \cdot Q_2(Q_1 \oplus Q_3) + key\_0' \cdot key\_1 \cdot (Q_1'Q_2'Q_3 + Q_2Q_3') + key\_0' \cdot key\_1' \cdot Q_2]Rst'$$

$$Q_3^* = [key\_0(Q_1'Q_2' + Q_1'Q_2Q_3' + Q_1Q_3) + key\_0' \cdot key\_1 \cdot (Q_2'Q_3 + Q_1Q_2Q_3') + key\_0' \cdot key\_1' \cdot Q_3]Rst'$$

3 行为级模型代码

行为级模型代码设计分为顶层模块、状态机模块和按钮处理模块。完整代码见附录 1 。

4 行为级仿真

编写仿真代码（见附录 2），模拟实际按钮的输入。为了方便仿真测试，把参数 CNT_10MS 改成了 10，即延时 10 个时钟周期。

1) 正确输入：可以看到状态经历了 0-1-3-2-6-7 的变化，符合前述对状态的编码，结果正确。

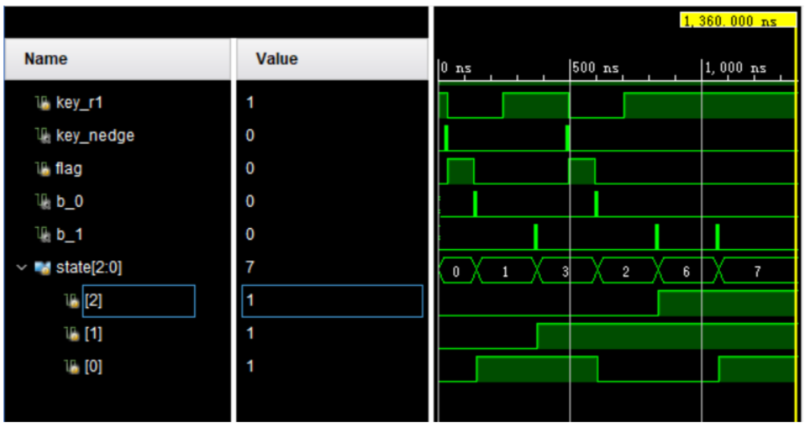
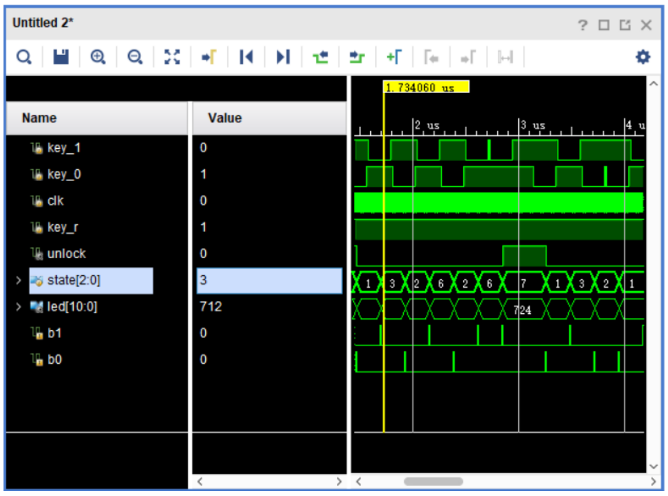


图 2 正确输入

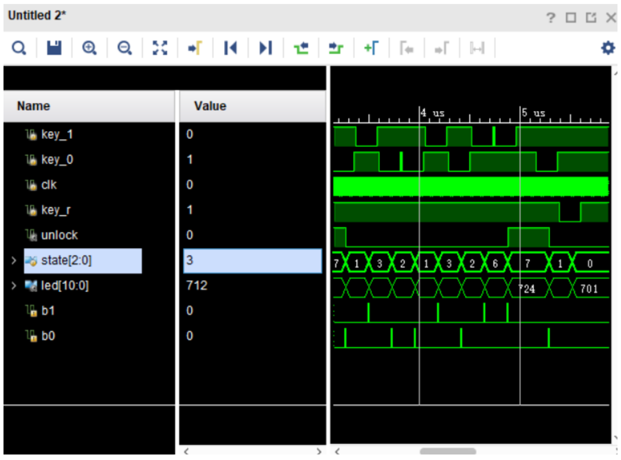
2) 错误输入 1：模拟按下 0101011，可以看到状态经历了 1-3-2-6-2-6-7 的变化，结果正确。



Name	Value
key_1	0
key_0	1
clk	0
key_r	1
unlock	0
state[2:0]	3
led[10:0]	712
b1	0
b0	0

图 3 错误输入 1

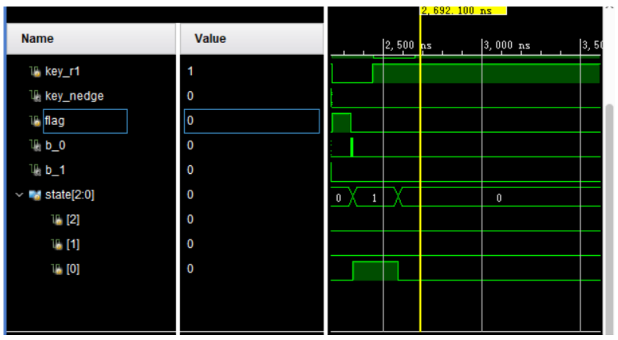
3) 错误输入 2：模拟按下 01001011，可以看到状态经历了 1-3-2-1-3-2-6-7 的变化，结果正确。



Name	Value
key_1	0
key_0	1
clk	0
key_r	1
unlock	0
state[2:0]	3
led[10:0]	712
b1	0
b0	0

图 4 错误输入 2

4) Reset 输入：模拟按下 0 后，按 Reset 按钮，可以看到状态经历了 1-0 的变化，结果正确。



Name	Value
key_r1	1
key_nedge	0
flag	0
b_0	0
b_1	0
state[2:0]	0
[2]	0
[1]	0
[0]	0

图 5 Reset 输入

综合以上仿真结果，认为设计无误。

5 设计及验证 FSM 的门级模型

5.1 设计门级模型

根据上文推导出的布尔表达式，

$$unlock = Q_1Q_2Q_3Rst'$$

$$Q_1^* = (key\_0'\cdot key\_1\cdot Q_2Q_3' + key\_0'\cdot key\_1'\cdot Q_1Q_2)Rst'$$

$$Q_2^* = [key\_0\cdot Q_2(Q_1\oplus Q_3) + key\_0'\cdot key\_1\cdot(Q_1'Q_2'Q_3 + Q_2Q_3') + key\_0'\cdot key\_1'\cdot Q_2]Rst'$$

$$Q_3^* = [key\_0(Q_1'Q_2' + Q_1'Q_2Q_3' + Q_1Q_3) + key\_0'\cdot key\_1\cdot(Q_2'Q_3 + Q_1Q_2Q_3') + key\_0'\cdot key\_1'\cdot Q_3]Rst'$$

FSM 以三个 D 触发器为核心，设计 D 触发器的输入电路如下：

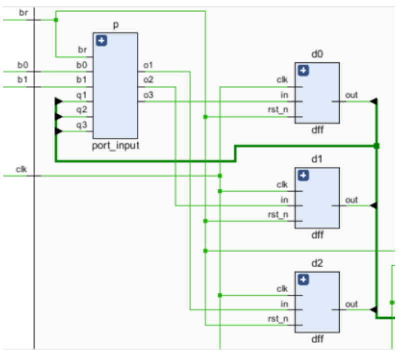


图 6 输入电路（封装后）

将 port_input 模块展开至门级，见下图。Verilog 代码编写时，笔者优化了原先的 FSM 结构，将 FSM 分成若干子模块：D 触发器输入电路模块 port_input、D 触发器模块 dff、输出模块 Output，详细代码见附录 3。

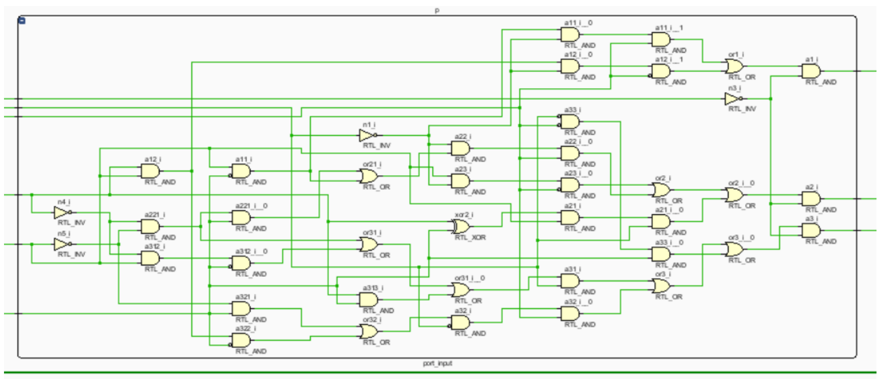
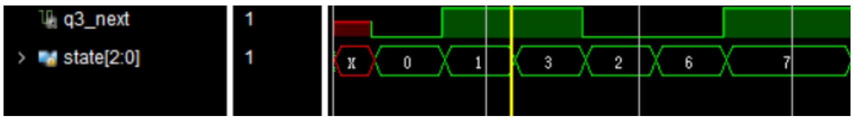


图 7 输入电路（门级）

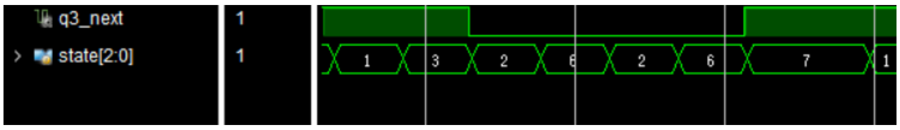
5.2 模型验证

采用附录 2 的仿真文件，得到以下仿真波形：

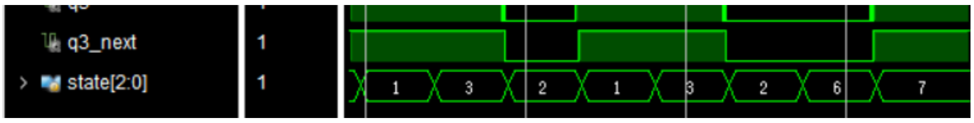
1） 正确输入：状态为 1-3-2-6-7，结果准确。



2） 错误输入 1：状态为 1-3-2-6-2-6-7，结果准确。



3） 错误输入 2：状态为 1-3-2-1-3-2-6-7，结果准确。



4） Reset 输入：状态为 1-0，结果准确。

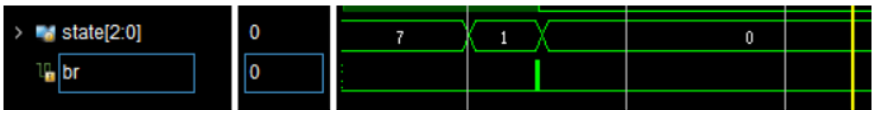


图 8、9、10、11 门级电路仿真波形

因此认为门级模型设计无误。

附录 1

1. 顶层模块

```
module lock(
    input key_1,
    input key_0,
    input clk,
    input key_r,
    output unlock,
    output [10:0] led
);

wire b_1, b_0, rst_n;
//detect effective press
button button_1(.key_in(key_1), .clk(clk), .press(b_1));
button button_0(.key_in(key_0), .clk(clk), .press(b_0));
button button_r(.key_in(key_r), .clk(clk), .press(rst_n));

reg [2:0] state = 3'b000, state_next;
//finite state machine
FSM
lock_fsm(.b0(b_0), .b1(b_1), .br(rst_n), .clk(clk), .unlock(unlock), .l
ed_out(led));
endmodule
```

2. 按钮处理模块

```
//---------------------------------------------
//press button, output a signal.
//---------------------------------------------

module button(
    input key_in, // input
    input clk,     // 10MHz
    output reg press
);

//parameter CNT_10MS = 10;  //for simulation test
parameter CNT_10MS = 100000;

reg     key_r0 = 1'b1;  // last period
```

```verilog
reg     key_r1 = 1'b1;  // this period
wire    key_nedge;      // scan the negedge

reg [19:0]  cnt = 0;
reg         flag = 0;

//cascade two D flip-flop
always @(posedge clk) begin
        key_r0 <= key_in;
        key_r1 <= key_r0;
end
assign key_nedge = ~key_r0 & key_r1;

//debounce & width transformation
always @(posedge clk ) begin
    if(flag) begin
        if(cnt == CNT_10MS - 1) begin
            cnt <= 0;
        end
        else begin
            cnt <= cnt + 1;
        end
    end
end
always @(posedge clk) begin
    if (key_nedge) begin
        flag <= 1'b1;
    end
    else if (cnt == CNT_10MS - 1) begin
        flag <= 1'b0;
    end
end

always @(posedge clk) begin
    if(cnt == CNT_10MS - 1) begin
        press <= ~key_r0;
    end
    else begin
        press <= 1'b0;
    end
end
endmodule
```

3.　状态机模块

```
module FSM(
    input b0,
    input b1,
    input br,
    input clk,
    output unlock,
    output [10:0] led_out
);

parameter STATE_0 = 3'b000;
parameter STATE_1 = 3'b001;
parameter STATE_2 = 3'b011;
parameter STATE_3 = 3'b010;
parameter STATE_4 = 3'b110;
parameter STATE_5 = 3'b111;
parameter STATE_6 = 3'b100;
parameter STATE_7 = 3'b101;

reg [2:0] state = 3'b000, state_next;

//next state logic
always @(*)
begin
    if(br)
    begin
        state_next = STATE_0;
    end
    else
    case(state)
        STATE_0:
        begin
            if(b0)
                state_next = STATE_1;
            else if(b1)
                state_next = STATE_0;
            else
                state_next = state;
        end
        STATE_1:
        begin
            if(b1)
```

```
                state_next = STATE_2;
            else
                state_next = state;
        end
        STATE_2:
        begin
            if(b0)
                state_next = STATE_3;
            else if(b1)
                state_next = STATE_0;
            else
                state_next = state;
        end
        STATE_3:
        begin
            if(b1)
                state_next = STATE_4;
            else if(b0)
                state_next = STATE_1;
            else
                state_next = state;
        end
        STATE_4:
        begin
            if(b1)
                state_next = STATE_5;
            else if(b0)
                state_next = STATE_3;
            else
                state_next = state;
        end
        STATE_5:
        begin
            if(b0)
                state_next = STATE_1;
            else if(b1)
                state_next = STATE_0;
            else
                state_next = state;
            end
        STATE_6:state_next = STATE_0;
        STATE_7:state_next = STATE_1;
    endcase
end
```

```verilog
//state shift
always @(posedge clk)
begin
    state = state_next;
end

//response
reg [10:0] led;
always @(posedge clk or negedge br)
begin
    if(br)
       begin
        led <= 11'b1110_0000001;
       end
    else
    begin
        case(state)
        STATE_0: led <= 11'b1110_0000001;
        STATE_1: led <= 11'b1110_1001111;
        STATE_2: led <= 11'b1110_0010010;
        STATE_3: led <= 11'b1110_0000110;
        STATE_4: led <= 11'b1110_1001100;
        STATE_5: led <= 11'b1110_0100100;
        STATE_6: led <= 11'b1110_0100000;
        STATE_7: led <= 11'b1110_0111100;
        endcase
    end
end

assign unlock = (state == 3'b111);
assign led_out = led;
endmodule
```

附录 2

仿真代码（仅展现有变化的部分）

```
initial
begin
    //To run this testbench successfully, you need to modify parameter
CNT_10MS to 10 in module button
    //1. correct answer 01011
    // state 1 -- 0
    #20     key_0 = 0; #210     key_0 = 1;
    // state 2 -- 01
    #20     key_1 = 0; #210     key_1 = 1;
    // state 3 -- 010
    #20     key_0 = 0; #210     key_0 = 1;
    // state 4 -- 0101
    #20     key_1 = 0; #210     key_1 = 1;
    // state 5 -- 01011
    #20     key_1 = 0; #210     key_1 = 1;

    //2. wrong answer 1 -- 0101011
    // state 1 -- 0
    #200    key_0 = 0; #210     key_0 = 1;
    // state 2 --01
    #20     key_1 = 0; #210     key_1 = 1;
    // state 3 -- 010
    #20     key_0 = 0; #210     key_0 = 1;
    // state 4 -- 0101
    #20     key_1 = 0; #210     key_1 = 1;
    // state 3 -- 01010
    #20     key_0 = 0; #210     key_0 = 1;
    // state 4 -- 010101
    #20     key_1 = 0; #210     key_1 = 1;
    // state 5 -- 0101011
    #20     key_1 = 0; #210     key_1 = 1;

    //3. wrong answer 2 -- 01001011
    // state 1 -- 0
    #200    key_0 = 0; #210     key_0 = 1;
    // state 2 --01
    #20     key_1 = 0; #210     key_1 = 1;
    // state 3 -- 010
    #20     key_0 = 0; #210     key_0 = 1;
    // state 1 -- 0100
    #20    key_0 = 0; #210    key_0 = 1;
    // state 2 -- 01001
    #20    key_1 = 0; #210    key_1 = 1;
    // state 3 -- 010010
    #20    key_0 = 0; #210    key_0 = 1;
    // state 4 -- 0100101
    #20    key_1 = 0; #210    key_1 = 1;
    // state 5 -- 01001011
    #20    key_1 = 0; #210    key_1 = 1;

    //4. input reset
    // state 1 -- 0
    #200   key_0 = 0; #210    key_0 = 1;
    // state 2 -- reset
    #20    key_r = 0; #210    key_r = 1;

//    $finish;
end
```

附录 3

FSM 的门级描述

1. D 触发器

```
module dff (
    input clk,
    input in,
    output reg out
);

always @(posedge clk)
begin
    out = in;
end
endmodule
```

2. D 触发器的输入电路

```
module port_input (
    input b0,    input b1,    input br,
    input q1,    input q2,    input q3,
    output o1,   output o2,   output o3
);

wire nb0, nb1, nbr, nq1, nq2, nq3;
not n1(nb0, b0),    n2(nb1, b1),    n3(nbr, br),
    n4(nq1, q1),    n5(nq2, q2),    n6(nq3, q3);
// construct Q1*
wire s1, s2, s3;
and a11(s1, q2, nq3, nb0, b1),
    a12(s2, q1, q2, nb0, nb1);
or  or1(s3, s1, s2);
and a1(o1, s3, nbr);
// construct Q2*
wire t1, t2, t3, t4, t5, t6, t7, t8;
xor xor2(t1, q1, q3);
and a21(t2, t1, q2, b0),
    a221(t3, nq1, nq2, q3),
    a222(t4, q2, nq3);
or or21(t5, t3, t4);
and a22(t6, nb0, t5, b1),
    a23(t7, q2, nb0, nb1);
or or2(t8, t7, t6, t2);
and a2(o2, t8, nbr);
```

```verilog
//construct Q3*
wire w1, w2, w3, w4, w5, w6, w7, w8, w9, w10, w11;
and a311(w1, nq1, nq2),
    a312(w2, nq1, q2, nq3),
    a313(w3, q1, q3),
    a321(w4, nq2, q3),
    a322(w5, q1, q2, nq3);
or or31(w6, w1, w2, w3);
or or32(w7, w4, w5);
and a31(w8, w6, b0),
    a32(w9, w7, nb0, b1),
    a33(w10, nb0, nb1, q3);
or or3(w11, w8, w9, w10);
and a3(o3, w11, nbr);

endmodule
```

3.　FSM 的输出电路

```verilog
module Output (
    input clk,
    input br,
    input state,
    output led_out,
    output unlock
);

parameter STATE_0 = 3'b000; parameter STATE_1 = 3'b001;
parameter STATE_2 = 3'b011; parameter STATE_3 = 3'b010;
parameter STATE_4 = 3'b110; parameter STATE_5 = 3'b111;
parameter STATE_6 = 3'b100; parameter STATE_7 = 3'b101;
parameter LED_0 = 11'b1110_0000001; parameter LED_1 = 11'b1110_1001111;
parameter LED_2 = 11'b1110_0010010; parameter LED_3 = 11'b1110_0000110;
parameter LED_4 = 11'b1110_1001100; parameter LED_5 = 11'b1110_0100100;
parameter LED_6 = 11'b1110_0100000; parameter LED_7 = 11'b1110_0111100;
reg [10:0] led;

always @(posedge clk or negedge br)
begin
    if(br)
        led = LED_0;
    else
    begin
        case (state)
            STATE_0: led = LED_0;
            STATE_1: led = LED_1;
            STATE_2: led = LED_2;
            STATE_3: led = LED_3;
            STATE_4: led = LED_4;
            STATE_5: led = LED_5;
            STATE_6: led = LED_6;
            STATE_7: led = LED_7;
            default: led = LED_0;
        endcase
    end
end

endmodule
```

4. FSM 顶层模块

```
module FSM (
    input b0,
    input b1,
    input br,
    input clk,
    output unlock,
    output [10:0] led_out
);

reg [2:0] state = 3'b000;
wire
     q1, q2, q3,
     q1_next,
     q2_next,
     q3_next;

port_input
p(.b0(b0), .b1(b1), .br(br), .q1(q1_next), .q2(q2_next), .q3(q3_next),
.o1(q1), .o2(q2), .o3(q3));
dff d2(.clk(clk), .in(q1), .out(q1_next));
dff d1(.clk(clk), .in(q2), .out(q2_next));
dff d0(.clk(clk), .in(q3), .out(q3_next));
always @(posedge clk) begin
    state <= {q1_next,q2_next,q3_next};
end
Output
o(.clk(clk), .br(br), .state(state), .led_out(led_out), .unlock(unlock)
);
endmodule
```