北京科技大学 计算机与通信工程学院

实 验 报 告

实验名称： 实验四 状态机实验

一、实验目的与实验要求（可以有多个目标及要求，对应多个实验内容）

1、实验目的

本实验的目的是学习 Moore 型和 Mealy 型状态机的设计、状态机的编写以及在数字系统设计中的应用。

2、实验要求

1） 在实验报告中提交状态图或者算法流程状态图，系统级模块图、设计代码、仿真程序、仿真结果截图、实测验证结果照片。

2） 提交实验报告和每个实验的完整工程文件。

二、实验设备（环境）及要求

（1） Xilinx Ego1 实验平台。

（2） OS：Win7 64 位

（3） Software：Vivado15.4 开发工具

三、实验内容与步骤

1、实验 1

（1）实验内容：

验证 1101 序列（Mealy）：按照“5 实验步骤——验证 1101 序列（Mealy 型）”完成状态机的设计与仿真验证；

a) 状态机：

状态机由状态寄存器和组合逻辑电路构成，能够根据控制信号按照预先设定的状态进行状态转移，是协调相关信号动作、完成特定操作的控制中心。状态机简写为 FSM（Finite State Machine），主要分为两大类：

第一类，若输出只和状态有关而与输入无关，则称为 Moore 状态机

第二类，输出不仅和状态有关而且和输入有关系，则称为 Mealy 状态机

b) 状态图：

当需要定义一个状态机时，首先要绘制一张状态图。状态图可用来显示状态、状态间的转换和状态机的输出。图 1 显示了 Moore 状态机的状态图（左）和 Mealy 状态机的状态图（右）。

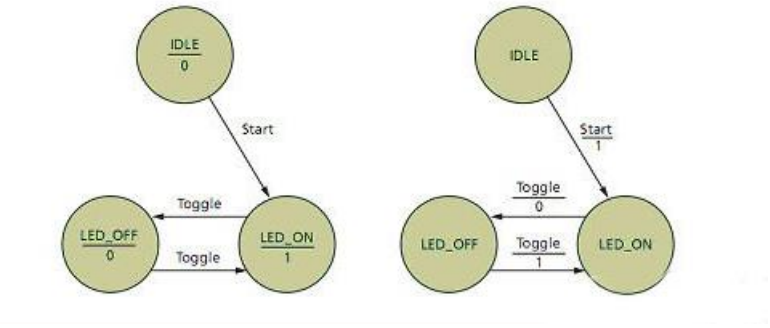

图 1 用于开/关 LED 的 Moore 状态机（左）和 Mealy 状态机（右）的状态图。

如果您要在物理组件中实现这些状态图（工程师在 FPGA 问世之前就是这么做的），首先就得生成当前状态和后续状态表，然后生成实现状态机所需的逻辑。不过由于我们将使用 FPGA 来实现设计，因此我们可以直接从状态转换图开始工作。

c) 算法状态图：

虽然有许多状态机是使用图 1 所示的状态图方法进行设计的，但另外还有一种描述状态机行为的方法，这就是算法状态图法。ASM 图（图 2）在外观上更加接近软件工程流程图。它由三个基本部分构成：

1. 状态框。它与状态名称有关，并包含 Moore 状态输出列表。
2. 决策框。如果检验某条件为真，则进行下一状态的判断。
3. 条件输出框。让状态机根据当前状态和输入描述 Mealy 输出。

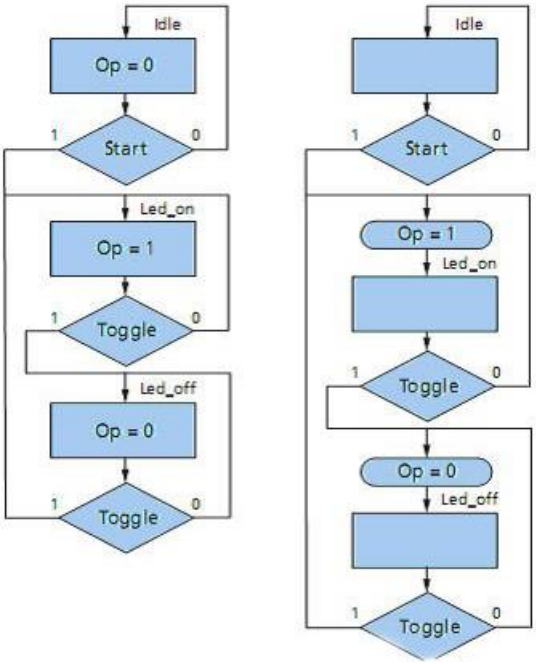

图 2 用于图 1 所示的状态机（Moore 状态机（左），Mealy 状态机（右））的算法状态图。

（2）实验步骤：

a) 新建工程， fsm_mealy 即为 Mealy 型状态机工程文件

b) Fsm_mealy 状态机模块代码：

```
`timescale 1ns / 1ps
module fsm_mealy(
    input wire clk,
    input wire clr,
    input wire din,
    output reg dout
     );
     reg [1:0]present_state,next_state;
     parameter S0=3'b00,S1=3'b01,S2=3'b10,S3=3'b11;
     always @(posedge clk or posedge clr)
     begin
      if(clr)
       present_state<=S0;
      else
      present_state<=next_state;
      end
      always @(*)
         begin
          case(present_state)
             S0:if(din)
                      next_state<=S1;
                  else
                      next_state<=S0;
             S1:if(din)
                          next_state<=S2;
                  else
                              next_state<=S0;
             S2:if(din)
                          next_state<=S2;
                 else
                       next_state<=S3;
            S3:if(din)
                              next_state<=S1;
                              else
                              next_state<=S0;
               default: next_state<=S0;
          endcase
         end
         always@(posedge clk or posedge clr)
            if(clr)
                dout<=0;
            else
                if((present_state==S3)&&din)
                 dout<=1;
                 else
                  dout<=0;
endmodule
```

c）测试激励代码：

```
`timescale 1ns / 1ps
module fsm_mealy_tb( );
reg clk;
reg clr;
reg din;
wire dout;
  parameter PERIOD=40;
  fsm_mealy U1(clk,clr,din,dout);
  initial
     begin
        clk=0;
        forever
          begin
          #(PERIOD/2) clk=1;
          #(PERIOD/2) clk=0;
          end

        end
  initial
     begin
       clr=1;
       forever
         #50 clr=0;
     end
     initial
       begin
         din=1;
         #350
           din=0;
           #50
           din=1;
           #100
           din=0;
           #50
           din=1;
       end
endmodule
```

d）状态机 RTL 级结构图

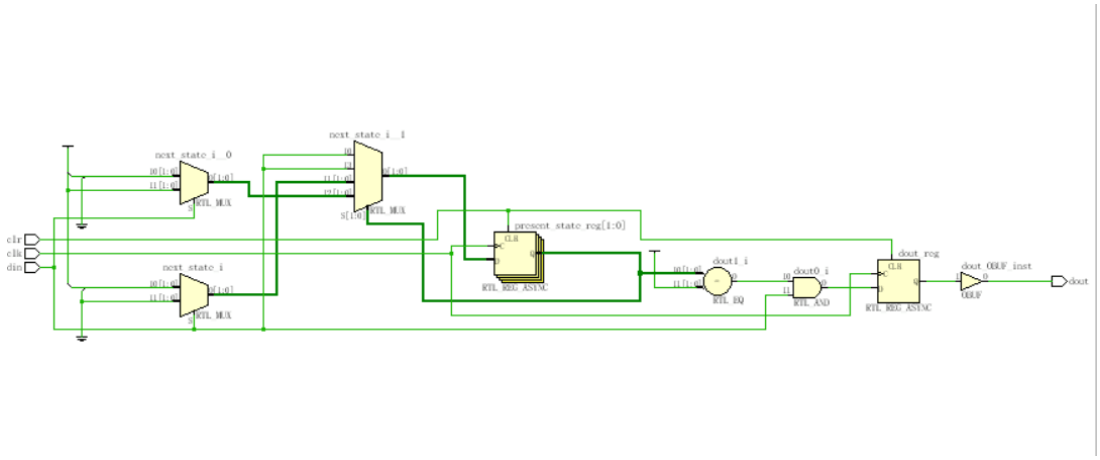

图 3 Mealy 型状态机 RTL 结构图

2、实验 2

（1） 实验内容：

按照“6 实验步骤——验证 1101 序列（Moore 型）”完成状态机的设计与仿真验证

（2）实验步骤：

a) 新建工程，即 Moore 型验证 1101 工程

b) Moore 序列验证机模块代码：

```
`timescale 1ns / 1ps
module fsm_moore(
    input wire clk,
        input wire clr,
        input wire din,
        output reg dout
         );
         reg [2:0]present_state,next_state;
         parameter S0=3'b000,S1=3'b001,S2=3'b010,S3=3'b011,S4=3'b100;
         always @(posedge clk or posedge clr)
         begin
          if(clr)
            present_state<=S0;
```

```
else
present_state<=next_state;
end
always @(*)
        begin
         case(present_state)
            S0:if(din)
                    next_state<=S1;
                else
                    next_state<=S0;
            S1:if(din)
                    next_state<=S2;
                else
                    next_state<=S0;
            S2:if(din)
                    next_state<=S2;
                else
                    next_state<=S3;
           S3:if(din)
                    next_state<=S4;
                    else
                    next_state<=S0;
             S4:if(din)
                    next_state<=S2;
                    else
                    next_state<=S0;
              default: next_state<=S0;
          endcase
        end
        always@(*)
```

```
                        if(present_state==S4)
                          dout<=1;
                          else
                            dout<=0;
endmodule
```

c) 测试激励代码：

```
`timescale 1ns / 1ps
module fsm_moore_tb();
        reg clk;
        reg clr;
        reg din;
        wire dout;
         parameter PERIOD=40;
         fsm_moore U1(clk,clr,din,dout);
         initial
            begin
               clk=0;
               forever
                 begin
                 #(PERIOD/2) clk=1;
                 #(PERIOD/2) clk=0;
                 end

               end
        initial
            begin
              clr=1;
              forever
                #50 clr=0;
```

```
        end
        initial
          begin
            din=1;
            #400
              din=0;
              #50
              din=1;
              #100
              din=0;
              #50
              din=1;
          end
endmodule
```

d) Moore 型 RTL 级结构图：

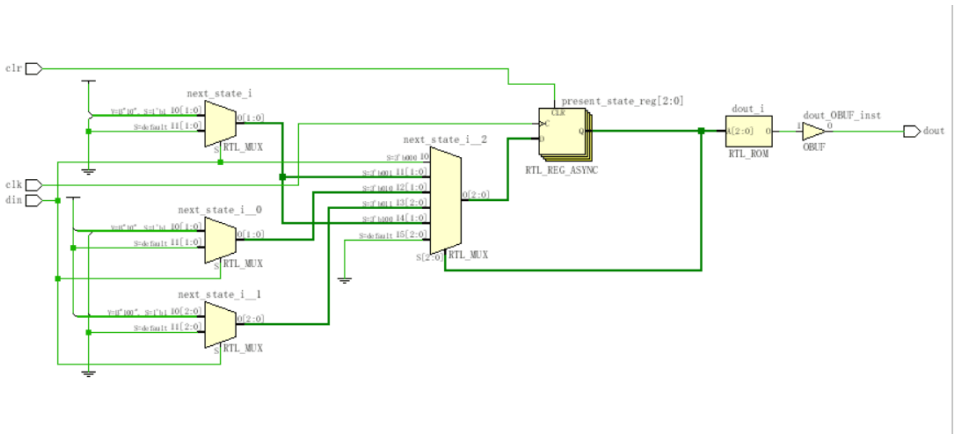

图 4： Moore 型状态机 RTL 级结构图

3、实验 3

（1）实验内容：修改源文件和仿真文件，编写两种状态机以检测序列 1000，并进行仿真验证结果是否正确。

（2）实验步骤：

a) Mealy 序列检测机模块代码：

```
`timescale 1ns / 1ps
module fsm_mealy(
    input wire clk,
    input wire clr,
    input wire din,
    output reg dout);
     reg [1:0]present_state,next_state;
     parameter S0=3'b00,S1=3'b01,S2=3'b10,S3=3'b11;
     always @(posedge clk or posedge clr)
     begin
      if(clr) present_state<=S0;
      else present_state<=next_state;
      end
      always @(*) begin
          case(present_state)
            S0:if(din)
                    next_state<=S1;
                else
                    next_state<=S0;
            S1:if(din)
                        next_state<=S0;
                else
                            next_state<=S2;
            S2:if(din)
                        next_state<=S1;
                else
                        next_state<=S3;
            S3:if(din)
                            next_state<=S1;
                            else
                            next_state<=S0;
              default: next_state<=S0;
          endcase end
        always@(posedge clk or posedge clr)
            if(clr) dout<=0;
            else
                if((present_state==S3)&&(din==0)) dout<=1; else dout<=0;
endmodule
```

b）Mealy 序列检测机测试激励代码

```
`timescale 1ns / 1ps
module fsm_mealy_tb( );
reg clk;
reg clr;
reg din;
wire dout;
  parameter PERIOD=40;
  fsm_mealy U1(clk,clr,din,dout);
  initial
     begin
        clk=0;
        forever
          begin
          #(PERIOD/2) clk=1;
          #(PERIOD/2) clk=0;
          end
        end
  initial
     begin
       clr=1;
       forever
         #50 clr=0;
     end
     initial
       begin
         din=1;
         #350
           din=0;
           #50
           din=1;
           #100
           din=0;
           #50
           din=1;
       end
endmodule
```

Moore:

c）Moore 序列检测机模块代码

```
`timescale 1ns / 1ps
module fsm_moore(
input wire clk,
    input wire clr,
    input wire din,
    output reg dout
     );
     reg [2:0]present_state,next_state;
     parameter S0=3'b000,S1=3'b001,S2=3'b010,S3=3'b011,S4=3'b100;
     always @(posedge clk or posedge clr)
     begin
      if(clr) present_state<=S0;
      else present_state<=next_state;
      end
      always @(*)
                begin
                 case(present_state)
                    S0:if(din)
                                next_state<=S1;
                            else
                                next_state<=S0;
                    S1:if(din)
                                    next_state<=S0;
                            else
                                        next_state<=S2;
                    S2:if(din)
                                    next_state<=S1;
                          else
                                next_state<=S3;
                     S3:if(din)
                                        next_state<=S1;
                                        else
                                        next_state<=S4;
                       S4:if(din)
                                    next_state<=S1;
                                    else
                                    next_state<=S0;
                        default: next_state<=S0;
                  endcase
                end
                always@(*)
                        if(present_state==S4) dout<=1;
                          else dout<=0;
endmodule
```

d）Moore 序列检测机测试激励代码

```
`timescale 1ns / 1ps
module fsm_moore_tb();
        reg clk;
        reg clr;
        reg din;
        wire dout;
          parameter PERIOD=40;
          fsm_moore U1(clk,clr,din,dout);
          initial
             begin
                clk=0;
                forever
                  begin
                  #(PERIOD/2) clk=1;
                  #(PERIOD/2) clk=0;
                  end
                end
          initial
             begin
               clr=1;
               forever
                 #50 clr=0;
             end
             initial
               begin
                 din=1;
                 #400
                   din=0;
                   #50
                   din=1;
                   #100
                   din=0;
                   #50
                   din=1;
               end
endmodule
```

e）两种状态机各自的 RTL 结构图：

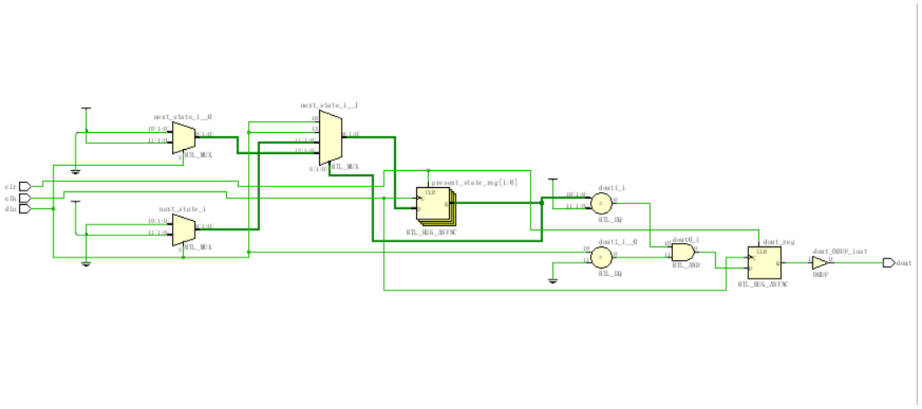

图 5 Mealy 状态机 RTL 结构图

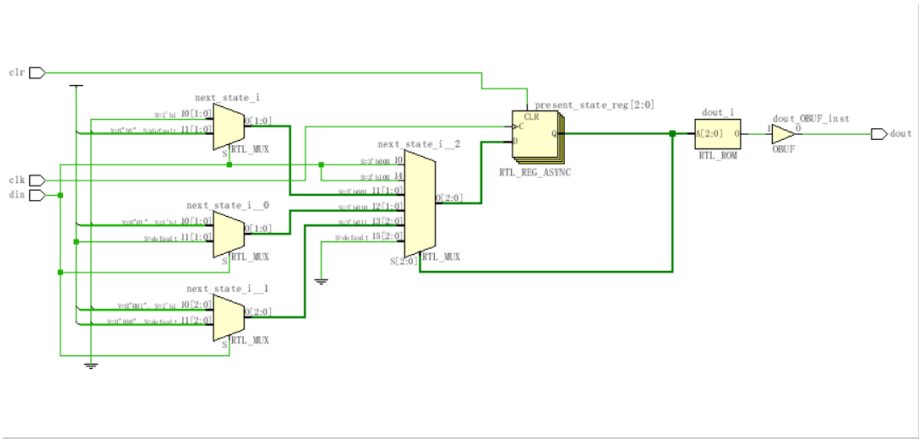

图 6 Moore 状态机 RTL 结构图

4、实验 4

（1） 实验内容：

根据在验证实验和练习实验中学到的知识，自己设计并实现状态机和整个控制系统的数字逻辑电路，设计电梯控制系统：通过仿真和实验板的现象验证设计的正确性。

（2） 实验步骤：

a）模块设计代码：

```
`timescale 1ns / 1ps
module clkdiv(
input clk100mhz,
output clk1s,
output clk190hz,
output clk200ms
);
	reg[31:0] count=0;
	assign clk1s = count[26];
	assign clk190hz = count[17];
	assign clk200ms = count[24];
	always @(posedge clk100mhz)
	count<=count+1;
	endmodule

`timescale 1ns / 1ps
module dianti(
input clk1s,
input clk190hz,
input clk200ms,
input reset,
input up1,input up2,input up3,input dn2,input dn3,input dn4,
input d1,input d2,input d3,input d4,
output reg [7:0]seg,
output reg [3:0]pos);
parameter idle_state = 3'b001;
parameter up_state = 3'b010;
parameter down_state=3'b011;
parameter open_state=3'b100;
parameter close_state=3'b101;
parameter f1=4'b0001;
parameter f2=4'b0010;
parameter f3=4'b0100;
parameter f4=4'b1000;
parameter up_f=2'b01;
parameter dn_f=2'b10;
parameter idle=2'b00;
```

```
reg [1:0]ud_f;
reg [3:0]count;
reg [3:0] led;
reg [3:0] dataP;
reg [2:0] state;
reg [3:0] next_state;
reg [3:0] now_f;
reg [3:0] up,dn,d;
reg [2:0] posC;
reg up1_f,up2_f,up3_f,dn2_f,dn3_f,dn4_f,
d1_f,d2_f,d3_f,d4_f;
always @(posedge clk1s or posedge reset)
      if(reset)
                begin
                        state<=idle_state;
                end
      else
      state<=next_state;
      always @(*)
      case(state)
      open_state:
                 begin
                 if(count<10)
                    next_state=open_state;
                 else
                    next_state=close_state;
                 end
      idle_state:
                begin
                  if(d>0)
                     begin
                        if((d&now_f)>0)
                           next_state=open_state;
                        else if(d>now_f)
                           next_state=up_state;
                        else
                                next_state=down_state;
                           end
      else if((up&now_f)||(dn&now_f))
                           next_state=open_state;
                      else if((up>now_f)||(dn>now_f))
                           next_state=up_state;
                      else if(up||dn)
                            next_state=down_state;
                       else
                            next_state=idle_state;
```

```
up_state:
                    begin
                      if((d&now_f)||(up&now_f))
                        next_state=open_state;
                      else if((d>now_f)||(up>now_f))
                        next_state=up_state;
                      else if(d||up)
                        next_state=down_state;
                      else if(dn>0)
                          begin
                            if(dn>now_f)
                              next_state=up_state;
                            else if((dn&now_f)||(now_f<f4))
                              next_state=open_state;
                            else if((dn&now_f)&&(now_f==f4))
                              next_state=open_state;
                            else
                              next_state=down_state;
                          end
                        else
                          next_state=idle_state;
                      end

        down_state:
                      begin
                        if((d&now_f)||(dn&now_f))
                          next_state=open_state;
                        else
if(((d<now_f)&&(d!=4'b0001))||((dn<now_f)&&(dn!=4'b0001)))
                          next_state=down_state;
                        else if((d>now_f)||(dn>now_f))
                            next_state=up_state;
                        else if(up>0)
                              begin
                                if(up<now_f)
                                  next_state=down_state;
                                else if((up&now_f)&&(now_f>f1))
                                  next_state=down_state;
                                else if((up&now_f)&&(now_f==f1))
                                  next_state=open_state;
                                 else
                                  next_state=up_state;
                              end
                            else
                              next_state=idle_state;
                         end
```

```
close_state:
                        begin
                          if(ud_f==up_f)
                            begin
                              if((d&now_f)||(up&now_f))
                                next_state=open_state;
                              else if((d>now_f)||(up>now_f))
                                next_state=up_state;
                              else if(d||up)
                                next_state=down_state;
                               else if(dn>0)
                                 begin
                                   if(dn>now_f)
                                     next_state=up_state;
                                   else if((dn&now_f)>0)
                                     next_state=open_state;
                                    else
                                     next_state=down_state;
                                 end
                                else
                                 next_state=idle_state;
                                end
                            else if(ud_f==dn_f)
                              begin
                                  if((d&now_f)||(dn&now_f))
                                    next_state=open_state;
                                  else
if(((d<now_f)&&(d!=4'b0000))||((dn<now_f)&&(dn!=4'b0000)))
                                    next_state=down_state;
                                  else if(d||dn)
                                      next_state=up_state;
                                  else if(up>0)
                                  begin
                                      if(up<now_f)
                                        next_state=down_state;
                                      else if((up&now_f)>0)
                                        next_state=open_state;
                                      else
                                        next_state=up_state;
                                  end
                                 else
                                   next_state=idle_state;
                          end
```

```
                    else
                       begin
                          if(d>0)
                           begin
                              if((d&now_f)>0)
                                 next_state=open_state;
                              else if(d>now_f)
                                 next_state=up_state;
                              else
                                 next_state=down_state;
                            end
                          else if((up&now_f)||(dn&now_f))
                             next_state=open_state;
                          else if((up>now_f)||(dn>now_f))
                             next_state=up_state;
                          else if(up||dn)
                             next_state=down_state;
                          else
                             next_state=idle_state;
                       end
                 end

  default:
              next_state=idle_state;

endcase

always @(posedge clk1s or posedge reset)
if(reset)
   begin
      now_f<=f1;
      ud_f<=idle;
         led<=1;
      end
  else
      begin
        now_f<=now_f;
        case(next_state)
           idle_state:
              begin
                 now_f<=now_f;
                 ud_f<=idle;
                 led<=led;
              end
```

```
up_state:
                    begin
                        now_f<=now_f<<1;
                        ud_f<=up_f;
                        led<=led+1;
                    end
                down_state:
                    begin
                        now_f<=now_f>>1;
                        ud_f<=dn_f;
                        led<=led-1;
                    end
                open_state:
                    begin
                        now_f<=now_f;
                        ud_f<=ud_f;
                        led<=led;
                    end
                close_state:
                    begin
                        now_f<=now_f;
                        ud_f<=ud_f;
                        led<=led;
                    end
                default:
                    begin
                        now_f<=f1;
                        ud_f<=idle;
                        led<=led;
                    end
        endcase
    end
    always@(dataP)
        case(dataP)
            0:seg=8'b0011_1111;
            1:seg=8'b0000_0110;
            2:seg=8'b0101_1011;
            3:seg=8'b0100_1111;
            4:seg=8'b0110_0110;
            5:seg=8'b0110_1101;
            6:seg=8'b0111_1101;
            7:seg=8'b0000_0111;
            8:seg=8'b0111_1111;
            9:seg=8'b0110_1111;
            10:seg=8'b0100_0000;
            11:seg=8'b0000_0000;
            default:seg = 8'b0000_1000;
        endcase
```

```
always @(posedge clk190hz)
                    begin
                    case(posC)
                      0: begin
                           pos<=4'b0001;
                           dataP<=count;
                           end
                      1: begin
                           pos<=4'b0010;
                           dataP<=led;
                           end
                      2: begin
                           pos<=4'b0100;
                           dataP<=ud_f;
                           end
                      3: begin
                           pos<=4'b1000;
                           dataP<=state;
                           end
                       endcase
                       posC = posC+1;
                       end
always@(posedge clk1s)
            begin

up1_f=up1;up2_f=up2;up3_f=up3;dn2_f=dn2;dn3_f=dn3;dn4_f=dn4;
            d1_f=d1;d2_f=d2;d3_f=d3;d4_f=d4;
            end
always @(up1_f or up2_f or up3_f    )
            up={1'b0, up3_f, up2_f, up1_f};

       always @(dn2_f or dn3_f or dn4_f )
            dn={dn4_f, dn3_f, dn2_f, 1'b0};

       always @(d1_f or d2_f or d3_f or d4_f )
            d={d4_f, d3_f, d2_f, d1_f};
       always @(posedge clk1s or posedge reset)
                    if(reset)
                          count<=0;
                    else if((next_state==open_state)&&(count<10))
                          count<=count+1;
         else
                          count<=0;
endmodule
```

b) FPGA 板级验证示意图：

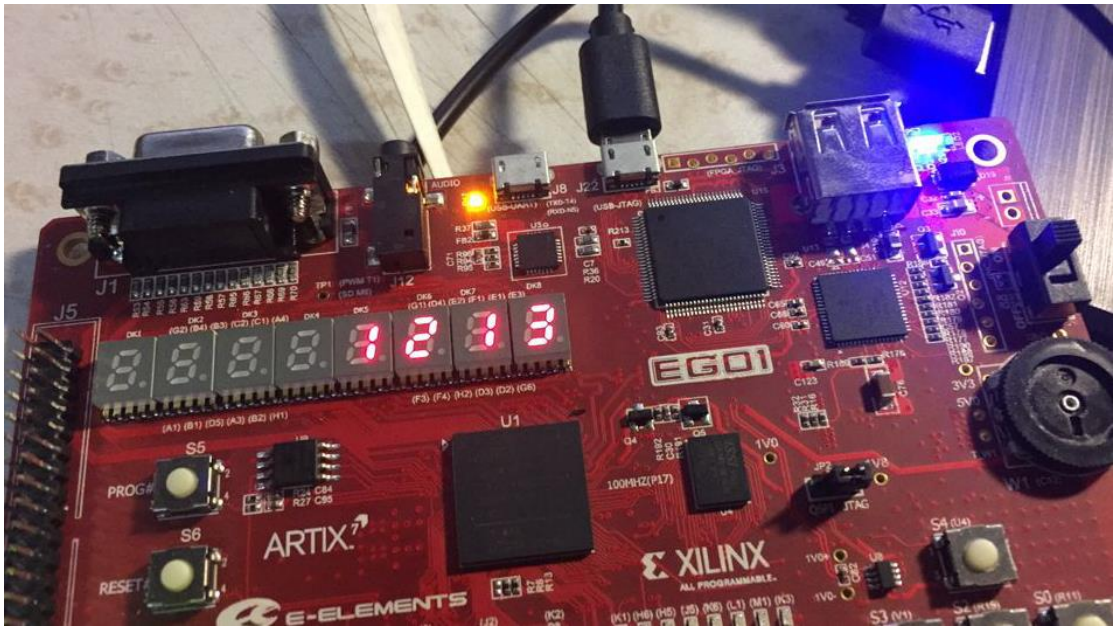

图 7 电梯实验板级验证

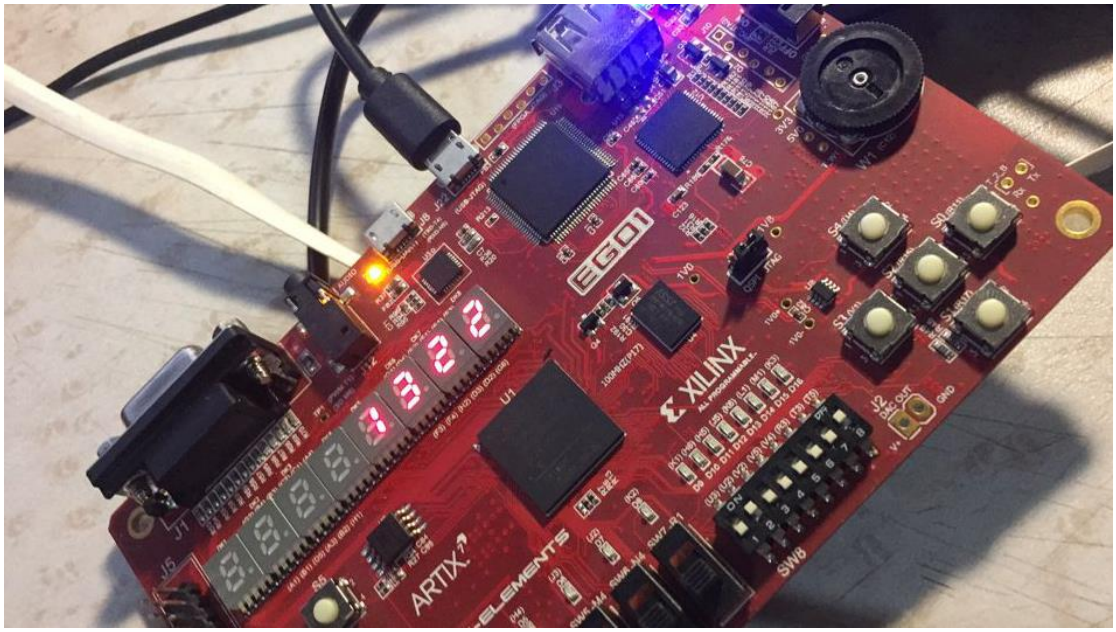

图 8 电梯实验板级验证

四：实验结果与分析

a) 使用 mearly 型状态机验证 1101 得到的仿真波形图

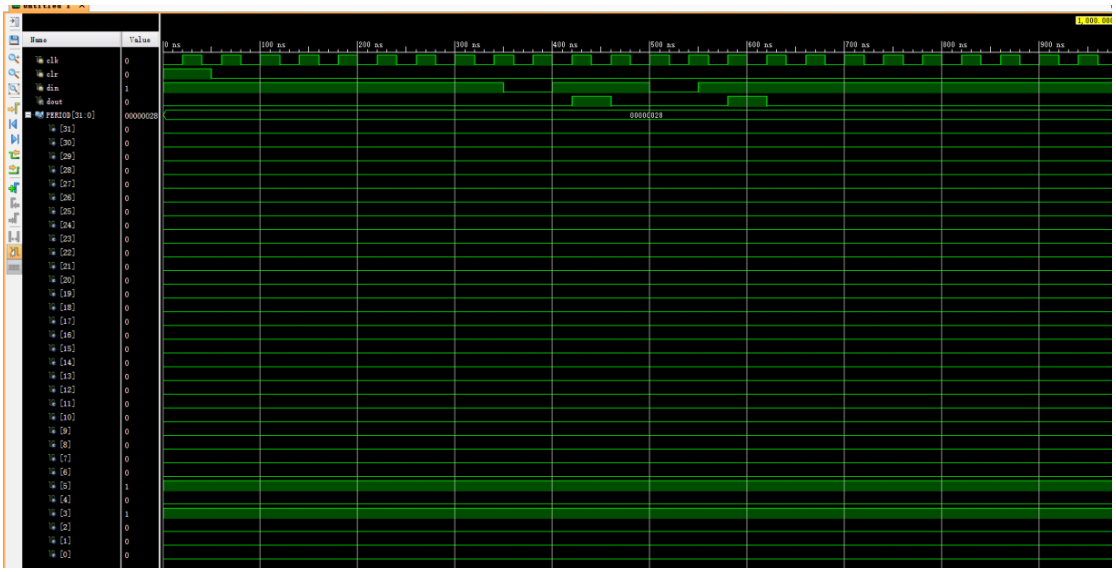

图 9 验证 1101 仿真波形图

b) 使用 moore 型状态机验证 1101 得到的仿真波形图

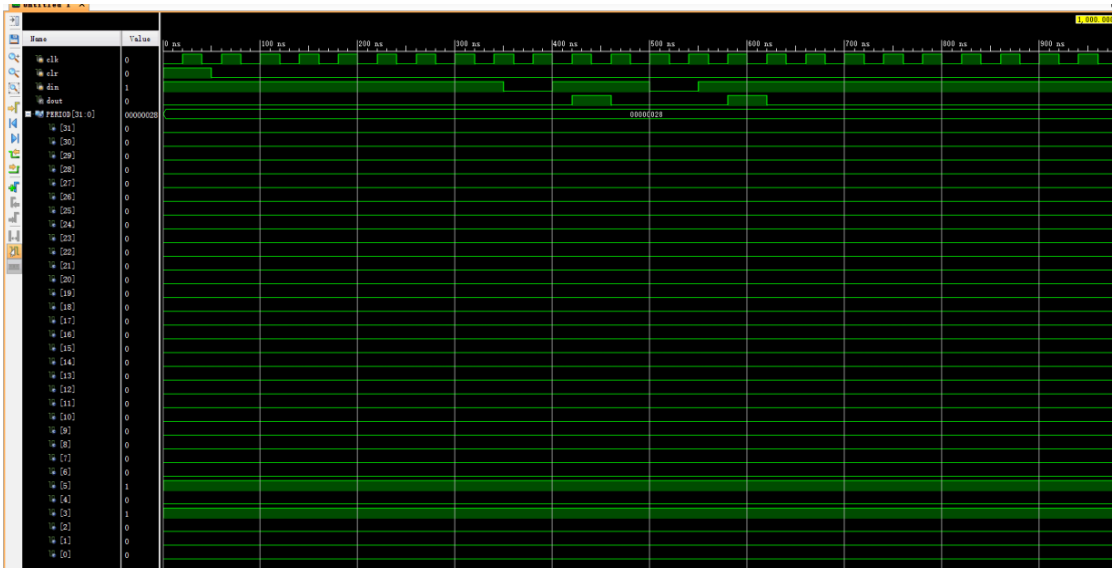

图 10 验证 1101 仿真波形图

c) 使用 mearly 型状态机验证 1000 得到的仿真波形图

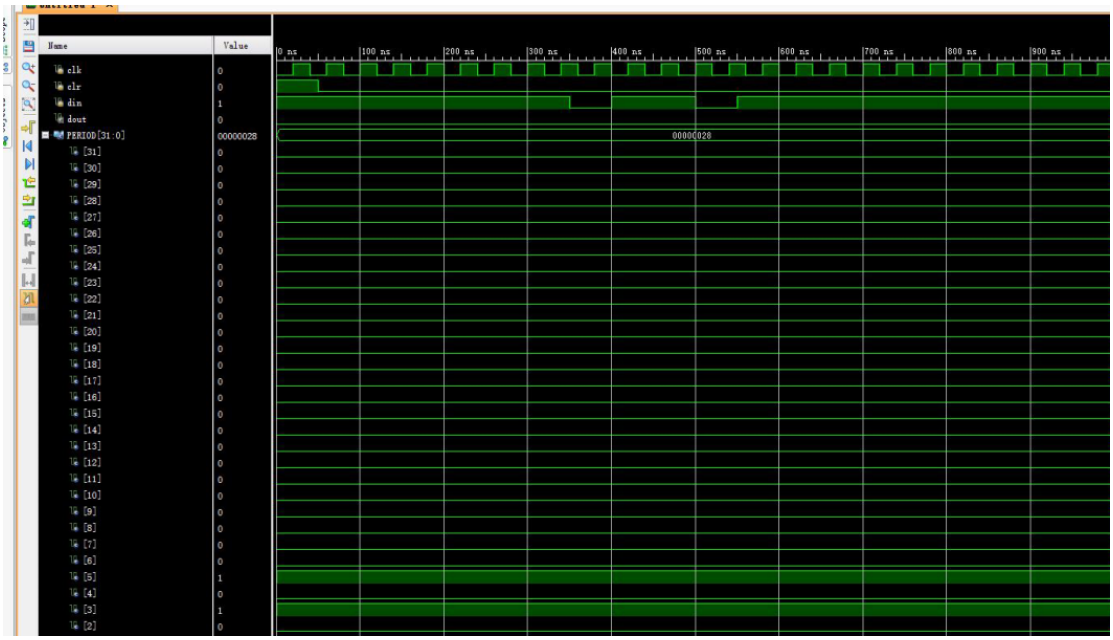

图 11 Mealy 型验证 1000 仿真波形图

d) 使用 moore 型状态机验证 1000 得到的仿真波形图

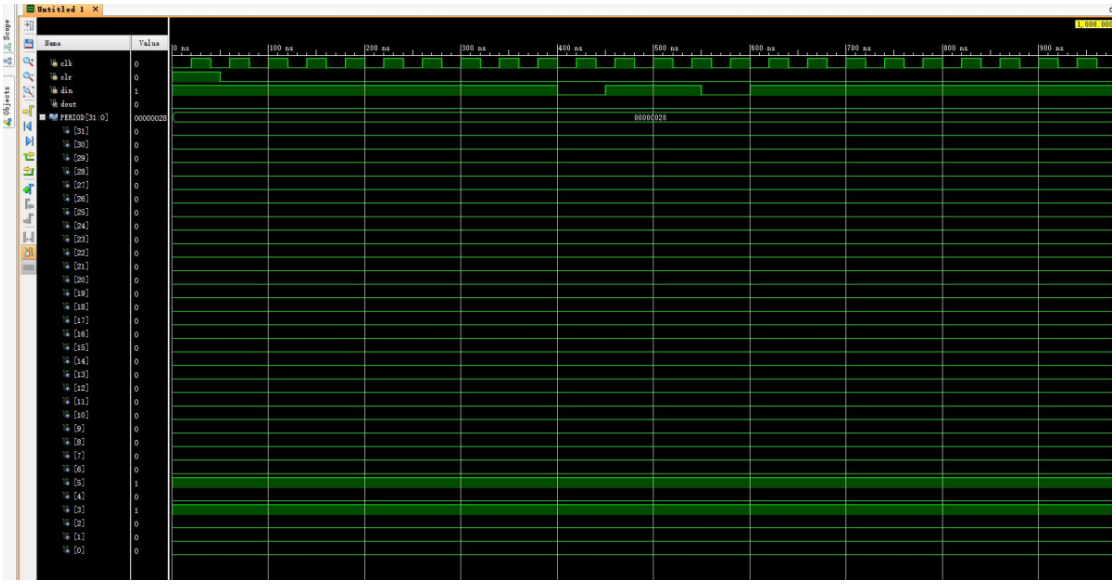

图 12 Moore 型验证 1000 仿真波形图

e) Mealy 状态机序列检测器状态转移图

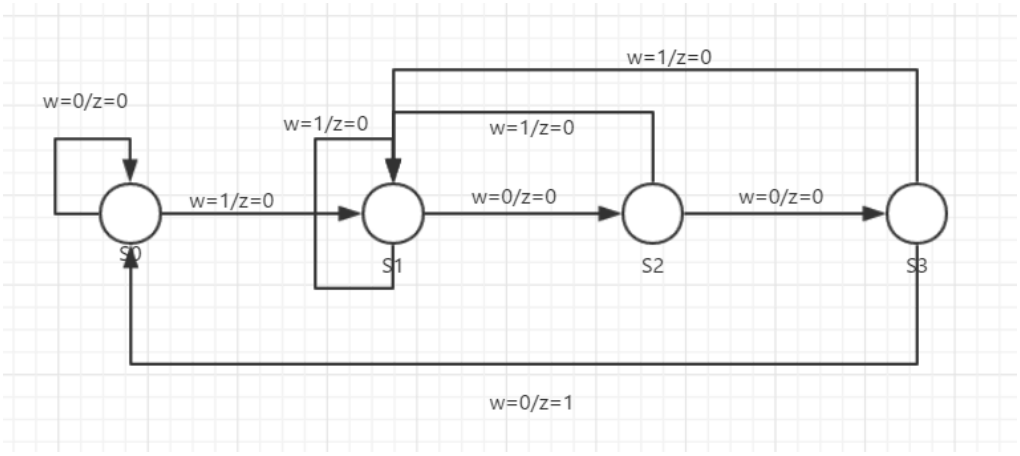

图 13 状态转移图

五：结论（讨论）

1、 实验结论

Moore 型状态机与 Mealy 型状态机类型不同 Moore 型状态机的转向只取决于当前的状态，Mealy 型不同。Moore 型状态机的输出信号是直接由状态寄存器译码得到，而 Mealy 型状态机则是以现时的输入信号结合即将变成次态的现态，编码成输出信号

Moore 状态机的输出只与有限状态自动机的当前状态有关，与输入信号的当前值无关。Moore 有限状态机在时钟 CLOCK 脉冲的有效边沿后的有限个门延后，输个出达到稳定值。即使在一时钟周期内输入信号发生变化,输出也会在一个完整的时钟周期内保持稳定值而不变。输入对输出的影响要到下一个时钟周期才能反映出来。Moore 有限状态机最重要的特点就是将输入与输出信号隔离开来。

一般而言，在设计中，Mealy 型状态机状态数量是少于 Moore 型的，使用的触发器也相对是最少的。而且，Mealy 型在我看来更符合实际应用时的情况，更加贴合我们思维，设计起来也是较为容易的。

2、讨论

实验中，经常会遇到输出结果与自己想的相差一个 clk 周期，其实是赋值的方式问题。在课程中，老师讲述了 FSM 的三段论方法，在我经分析看来，如果赋值语句放在 Always 模块外面，输入的变化会立刻反应到输出，如果放在了 always 语句里面，输出会产生一个独立的触发器，使得输出的变化相对于状态延迟一个时间周期。

经过分析 FSM 的 3 种风格的 Verilog 代码，意识到，给定的逻辑函数能有多种实现的方式，每个版本的代码通过 Verilog 编译器产生的电路略有不同。但是，这 3 个版本的代码产生的电路具有相同的功能。

六、教师评审

教师评语	实验成绩
签名： 日期：	