System Verilog入门（四）

实验目的

- 状态机

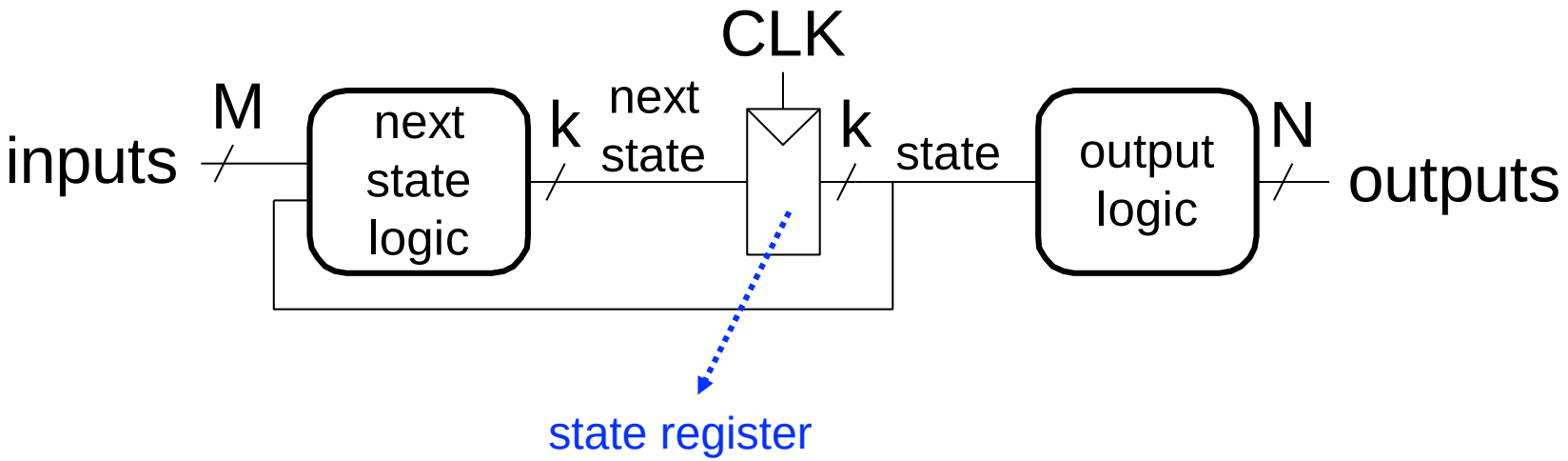
硬件编程注意事项

One of the most common mistakes for beginners is to think of HDL as a computer program rather than as a shorthand for describing digital hardware. If you don't know approximately what hardware your HDL should synthesize into, you probably won't like what you get. You might create far more hardware than is necessary, or you might write code that simulates correctly but cannot be implemented in hardware. Instead, think of your system in terms of blocks of combinational logic, registers, and finite state machines. Sketch these blocks on paper and show how they are connected before you start writing code.

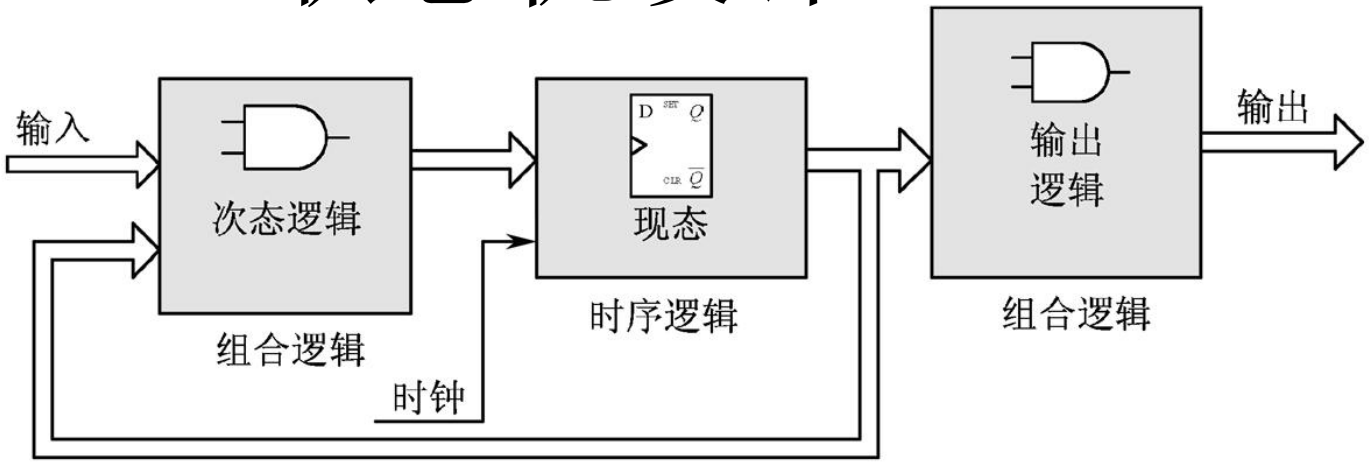
- **初学者通常的错误是认为**HDL是计算机编程语言，而不是数字硬件的描述语言
- 如果不知道被综合成什么样的硬件，那很有可能就是不对的
- **正确的做法：**从组合逻辑，寄存器，有限状态机的角度来考虑系统。可以在写代码之前在纸上绘制模块以及它们之间的连线情况

状态机

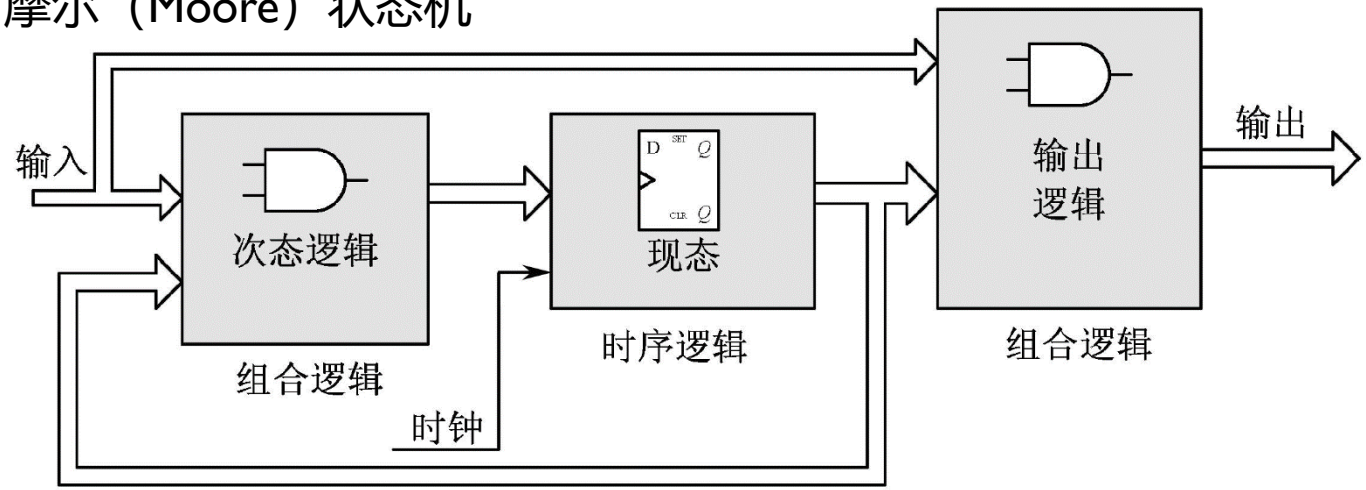
- 每个有限状态机包含三个部分
  - 现态逻辑（时序逻辑）
  - 次态逻辑（组合逻辑）
  - 输出逻辑（组合逻辑）



状态机设计

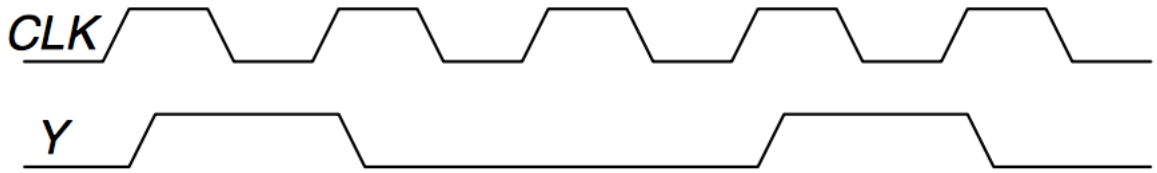


摩尔型状态机（Moore）：输出只和当前状态有关而与输入无关，则称为摩尔（Moore）状态机

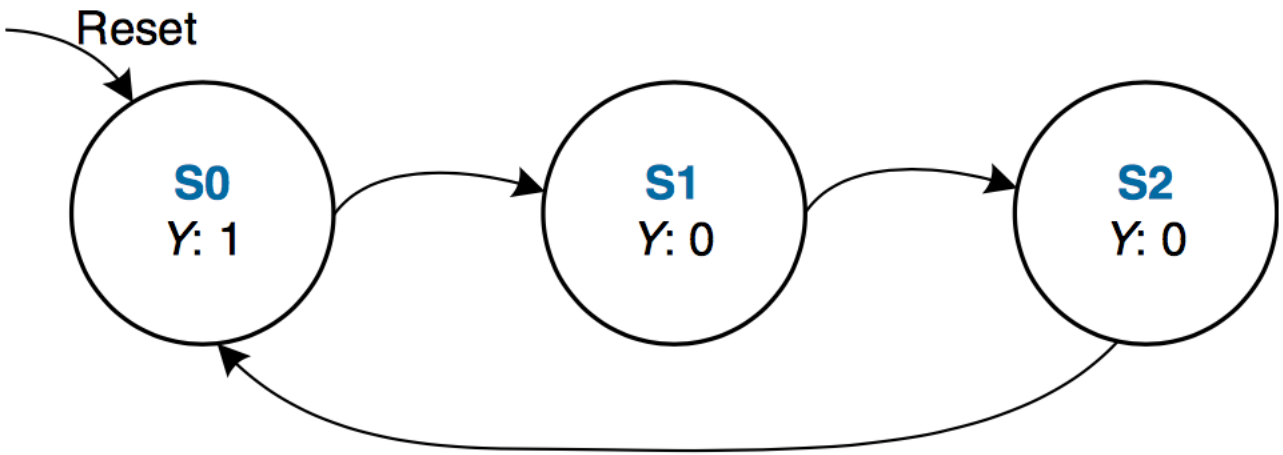


米利型状态机（Mealy）：输出不仅和当前状态有关而且和输入有关，则称为米利（Mealy）状态机

三分频

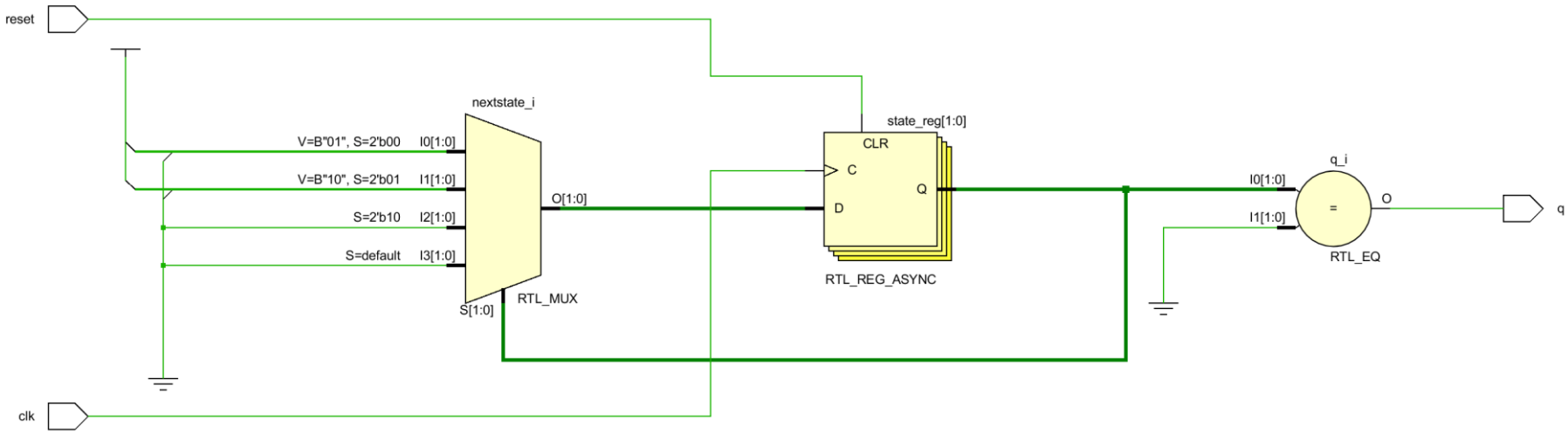


在每3个时钟周期中，输出Y为高电平。换句话说，该输出将时钟的频率除以3。



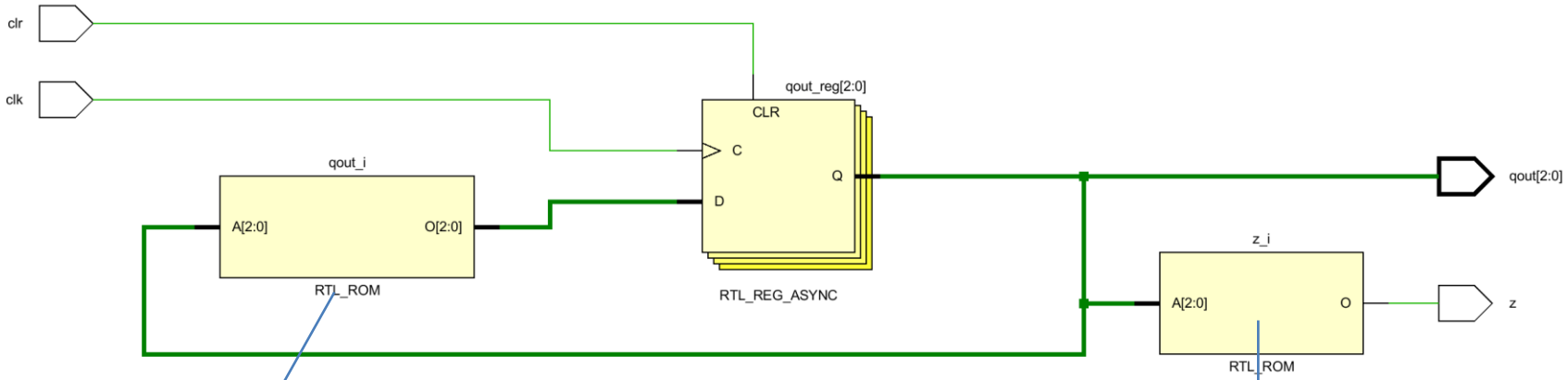
三分频

```
module divideby3FSM (input clk, input reset, output q);
   reg  [1:0] state, nextstate;

   parameter S0 = 2'b00; parameter S1 = 2'b01; parameter S2 = 2'b10;

   always_ff @ (posedge clk, posedge reset) // state register
      if (reset) state <= S0;
      else       state <= nextstate;

   always_comb                              // next state logic
      case (state)
         S0:      nextstate = S1;
         S1:      nextstate = S2;
         S2:      nextstate = S0;
         default: nextstate = S0;
      endcase

   assign q = (state == S0);                // output logic
endmodule
```


reset
clk
nextstate_i
V=B"01", S=2'b00
I0[1:0]
V=B"10", S=2'b01
I1[1:0]
S=2'b10
I2[1:0]
S=default
I3[1:0]
O[1:0]
S[1:0]
RTL_MUX
state_reg[1:0]
CLR
C
D
Q
RTL_REG_ASYNC
q_i
I0[1:0]
I1[1:0]
=
O
RTL_EQ
q

用状态机设计模5计数器

```
module fsm(clk,clr,z,qout);
        input clk,clr;
        output reg z;
        output reg[2:0] qout;

        always_ff @(posedge clk, posedge clr) //此过程定义状态转换
        begin
                if(clr) qout<=0; //异步复位
                else case(qout)
                        3'b000: qout<=3'b001;
                        3'b001: qout<=3'b010;
                        3'b010: qout<=3'b011;
                        3'b011: qout<=3'b100;
                        3'b100: qout<=3'b000;
                        default: qout<=3'b000; /*default语句*/
                endcase
        end

        always_comb/*此过程产生输出逻辑*/
        begin
                case(qout)
                        3'b100: z=1'b1;
                        default:z=1'b0;
                endcase
        end
endmodule
```



Cell Properties

qout_i

INIT	Value
INIT_D...	3'b000
INIT_0	3'b001
INIT_1	3'b010
INIT_2	3'b011
INIT_3	3'b100
INIT_4	3'b000

General | Properties | Nets | Cell Pins | **ROM values**

Cell Properties

z_i

INIT	Value
INIT_D...	1'b0
INIT_4	1'b1

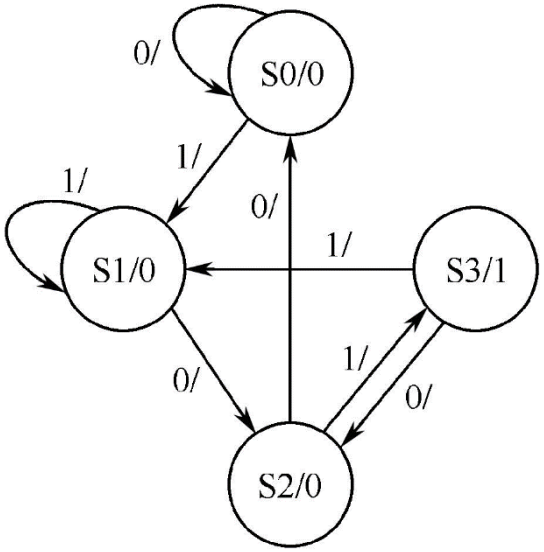
General | Properties | Nets | Cell Pins | **ROM values**

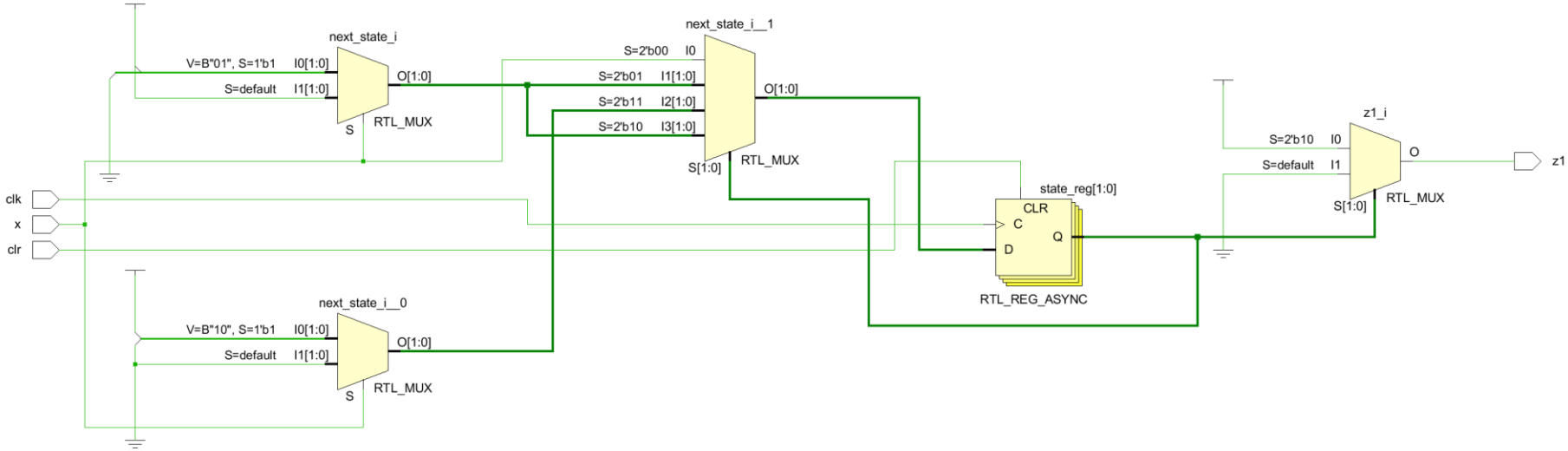
有限状态机的几种描述方式

- 用三个过程描述：
  - 即现态（CS），次态（NS），输出逻辑（OL）各用一个always过程描述。
- 双过程描述（CS+NS，OL双过程描述）：
  - 使用两个always过程来描述有限状态机，一个过程描述现态和次态时序逻辑（CS+NS）；另外一个过程描述输出逻辑（OL）。
- 双过程描述（CS，NS+OL双过程描述）：
  - 一个过程用来描述现态（CS）；另一个过程描述次态和输出逻辑（NS+OL）。
- 单过程描述：
  - 在单过程描述方式中，将状态机的现态、次态和输出逻辑（CS+NS+OL）放在一个always过程中进行描述。

```
module  ttt(clk,clr,x,z1);
   input  clk,clr,x;
   output reg z1;
   reg[1:0]  state,next_state;
   parameter  S0=2'b00,S1=2'b01,S2=2'b11,S3=2'b10;

   always_ff  @(posedge  clk,  posedge  clr)
   begin
      if(clr)
         state<=S0;
      else
         state<=next_state;
   end

   always_comb
   begin
      case(state)
         S0:
            begin
               if(x)  next_state<=S1;
               else  next_state<=S0;
            end
         S1:
            begin
               if(x)  next_state<=S1;
               else  next_state<=S2;
            end
         S2:
            begin
               if(x)  next_state<=S3;
               else  next_state<=S0;
            end
         S3:
            begin
               if(x)  next_state<=S1;
               else  next_state<=S2;
            end
         default:
            next_state<=S0;
      endcase
   end

   always_comb
   begin
      case(state)
         S3:z1=1'b1;
         default:z1=1'b0;
      endcase
   end
endmodule
```

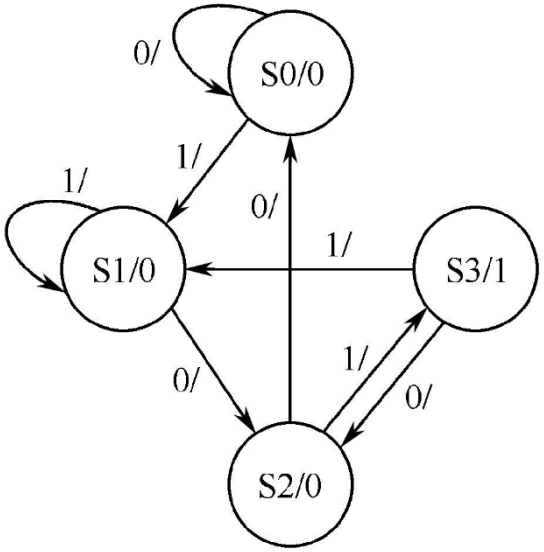
101序列检测器的Verilog描述（三个过程）

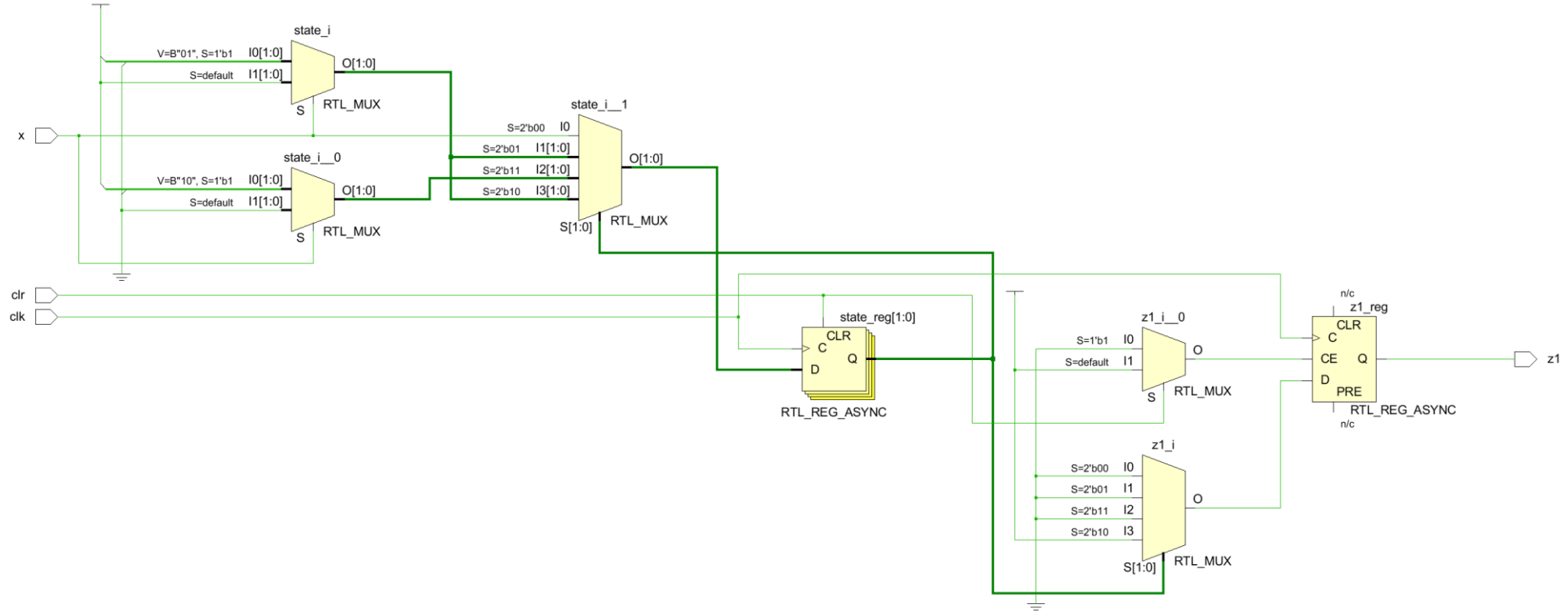



next_state_i
V=B"01", S=1'b1
I0[1:0]
S=default
I1[1:0]
O[1:0]
S
RTL_MUX
next_state_i__1
S=2'b00
I0
S=2'b01
I1[1:0]
S=2'b11
I2[1:0]
S=2'b10
I3[1:0]
O[1:0]
S[1:0]
RTL_MUX
next_state_i__0
V=B"10", S=1'b1
I0[1:0]
S=default
I1[1:0]
O[1:0]
S
RTL_MUX
clk
x
clr
state_reg[1:0]
CLR
C
D
Q
RTL_REG_ASYNC
z1_i
S=2'b10
I0
S=default
I1
O
S[1:0]
RTL_MUX
z1

单过程描述

```
module fsm4_seq101(clk,clr,x,z);
    input clk,clr,x;
    output reg z;
    reg[1:0] state;
    parameter S0=2'b00,S1=2'b01,S2=2'b11,S3=2'b10;

    always_ff @(posedge clk, posedge clr)
    begin
        if(clr) state<=S0;
        else
            case(state)
                S0:
                begin
                    if(x) begin state<=S1;z<=1'b0;end
                    else begin state<=S0;z<=1'b0;end
                end
                S1:
                begin
                    if(x) begin state<=S1;z<=1'b0;end
                    else begin state<=S2;z<=1'b0;end
                end
                S2:
                begin
                    if(x) begin state<=S3;z<=1'b0;end
                    else begin state<=S0;z<=1'b0;end
                end
                S3:
                begin
                    if(x) begin state<=S1;z<=1'b1;end
                    else begin state<=S2;z<=1'b1;end
                end
                default:begin state<=S0;z<=1'b0;end
            endcase
    end
endmodule
```




state_i
V=B"01", S=1'b1
I0[1:0]
S=default
I1[1:0]
O[1:0]
S
RTL_MUX
state_i__0
V=B"10", S=1'b1
I0[1:0]
S=default
I1[1:0]
O[1:0]
S
RTL_MUX
x
state_i__1
S=2'b00
I0
S=2'b01
I1[1:0]
S=2'b11
I2[1:0]
S=2'b10
I3[1:0]
O[1:0]
S[1:0]
RTL_MUX
clr
clk
state_reg[1:0]
CLR
C
D
Q
RTL_REG_ASYNC
z1_i__0
S=1'b1
I0
S=default
I1
O
S
RTL_MUX
z1_i
S=2'b00
I0
S=2'b01
I1
S=2'b11
I2
S=2'b10
I3
O
S[1:0]
RTL_MUX
n/c
z1_reg
CLR
C
CE
D
Q
PRE
RTL_REG_ASYNC
n/c
z1

状态编码

- 常用的编码方式

- 顺序编码
- 格雷编码
- 约翰逊编码
- 一位热码

一般使用case，casez和casex语句来描述状态之间的转换，用case语句表述比用if-else语句更加清晰明了

State Variables

State	One-Hot Code	Binary Code	Gray Code
S0	00001	000	000
S1	00010	001	001
S2	00100	010	011
S3	01000	011	010
S4	10000	100	110

Table 1:An example of state Encoding for a 4 state Machine

有限状态机设计要点

- 起始状态的选择：起始状态是指电路复位后所处的状态，选择一个合理的起始状态将使整个系统简洁，高效。多数EDA软件会自动为基于状态机的设计选择一个最佳的起始状态。
- 有限状态机的同步复位
- 有限状态机的异步复位

多余状态的处理

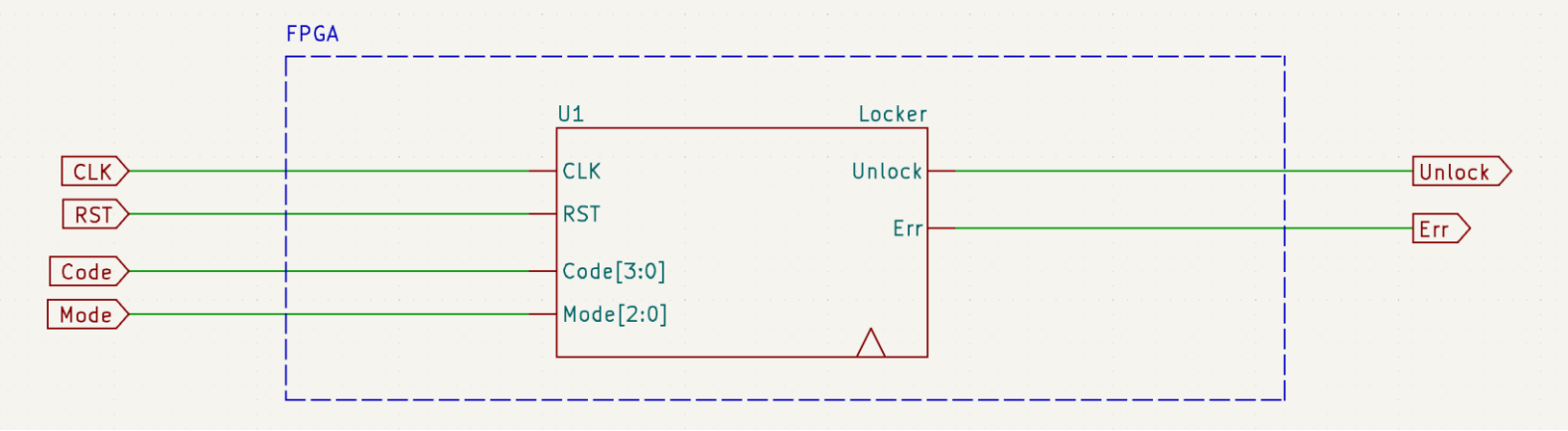
- 一般有如下两种处理多余状态的方法：
  - 在case语句中使用default分支决定如果进入无效状态所采取的措施；
  - 编写必要的Verilog源代码明确定义进入无效状态所采取的行为。

同步电路，异步电路

- 同步电路
  - 时钟源只有一个，所有触发器连接相同的时钟源
  - 触发器的状态与时钟变化同步
  - 有利于静态时序分析
  - 强耦合关系，不利于面积优化和低功耗优化
  - 存在时钟偏斜问题（与时钟源距离不同）
- 异步电路
  - 没有统一的时钟（可以有多个时钟），有的时钟同源不同相，有的时钟不同源
  - 很难对电路进行静态时序分析
  - 弱耦合关系，设计灵活，比同步功耗低
- 只考虑同步电路

实验内容

- 设计一个四位16进制串行电子密码锁，其具体功能如下：
  - 设置密码：用户可串行设置四位16进制密码；
  - 验证密码：用户串行输入密码，如密码符合则点亮开锁灯，若不符合则点亮错误灯；



提高要求

- 密码预置：为管理员创建万用密码以备管理。
- 系统报警：开锁三次失败后点亮报警灯，并锁定密码锁，只有输入管理员密码才可开锁,并解除报警。