上海交通大学
Shanghai Jiao Tong University

Verilog HDL 硬件描述语言（4）
状态机设计

上海交通大学微电子学院
蒋剑飞

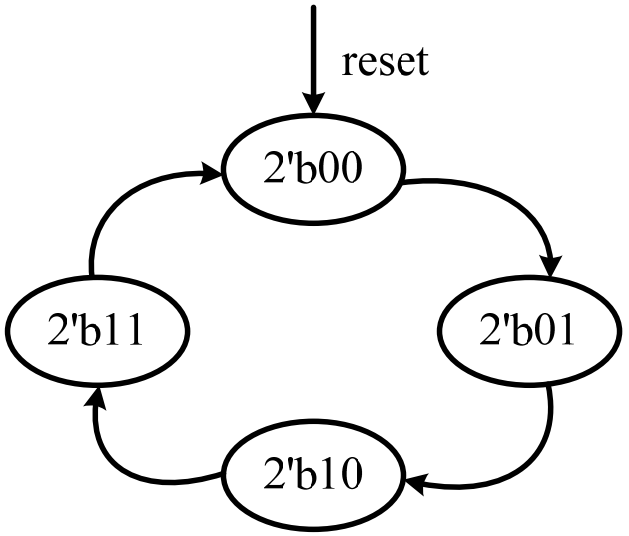
状态机的应用

- 有限状态机（Finite-state machine）是一种计算机科学与数学理论的抽象，将一些问题抽象成一组有限状态之间的切换及其控制，是数字逻辑与程序设计的基础之一。有限状态机在数字逻辑、控制、通讯、编译器设计、甚至生物领域都有着广泛的应用。

- 在数字电路中，控制模块的核心往往由状态机来实现。在日常生活中如红绿灯，自动贩卖机就是典型的应用之一。

简单的状态机——计数器

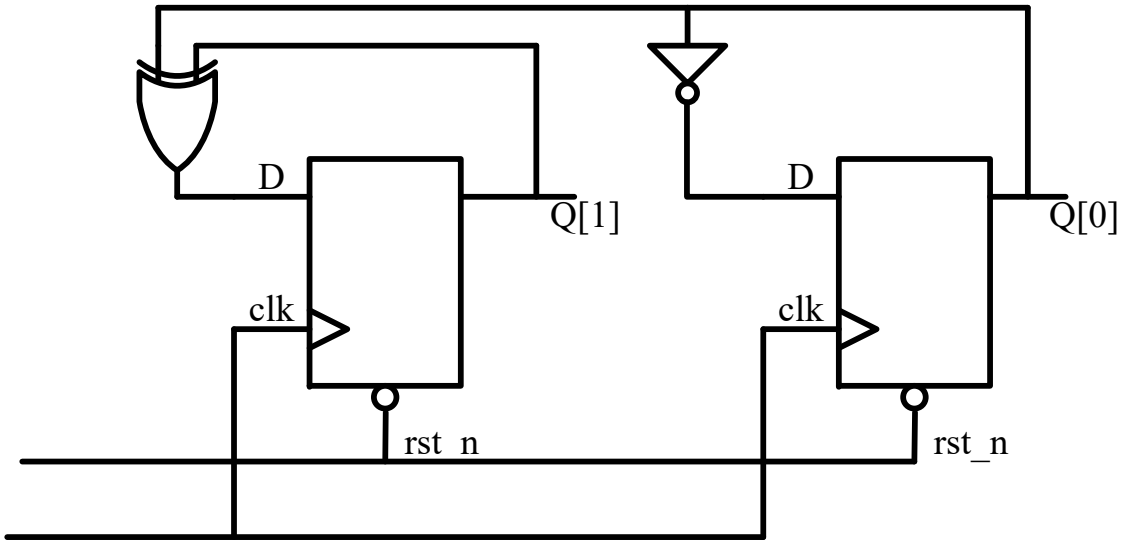
```
module two_bit_counter (clk,rst_n);

input clk,rst_n;
reg [1:0]  cnt;

always @(posedge clk or negedge rst_n)
if(!rst_n)

        cnt<=2'b0;

else

        cnt<=cnt+1'b1;

endmodule
```



组合电路与时序电路分开

```
module two_bit_counter(clk,rst_n);
input clk,rst_n;
wire clk,rst_n;
reg [1:0]  cnt,next_state;
always @(cnt)
case(cnt)
    2'b00:next_state=2'b01;
    2'b01:next_state=2'b10;
    2'b10:next_state=2'b11;
    2'b11:next_state=2'b00;
endcase
always @(posedge clk or negedge rst_n)
if(!rst_n)
            cnt<=2'b0;
else
            cnt<=next_state;
endmodule
```

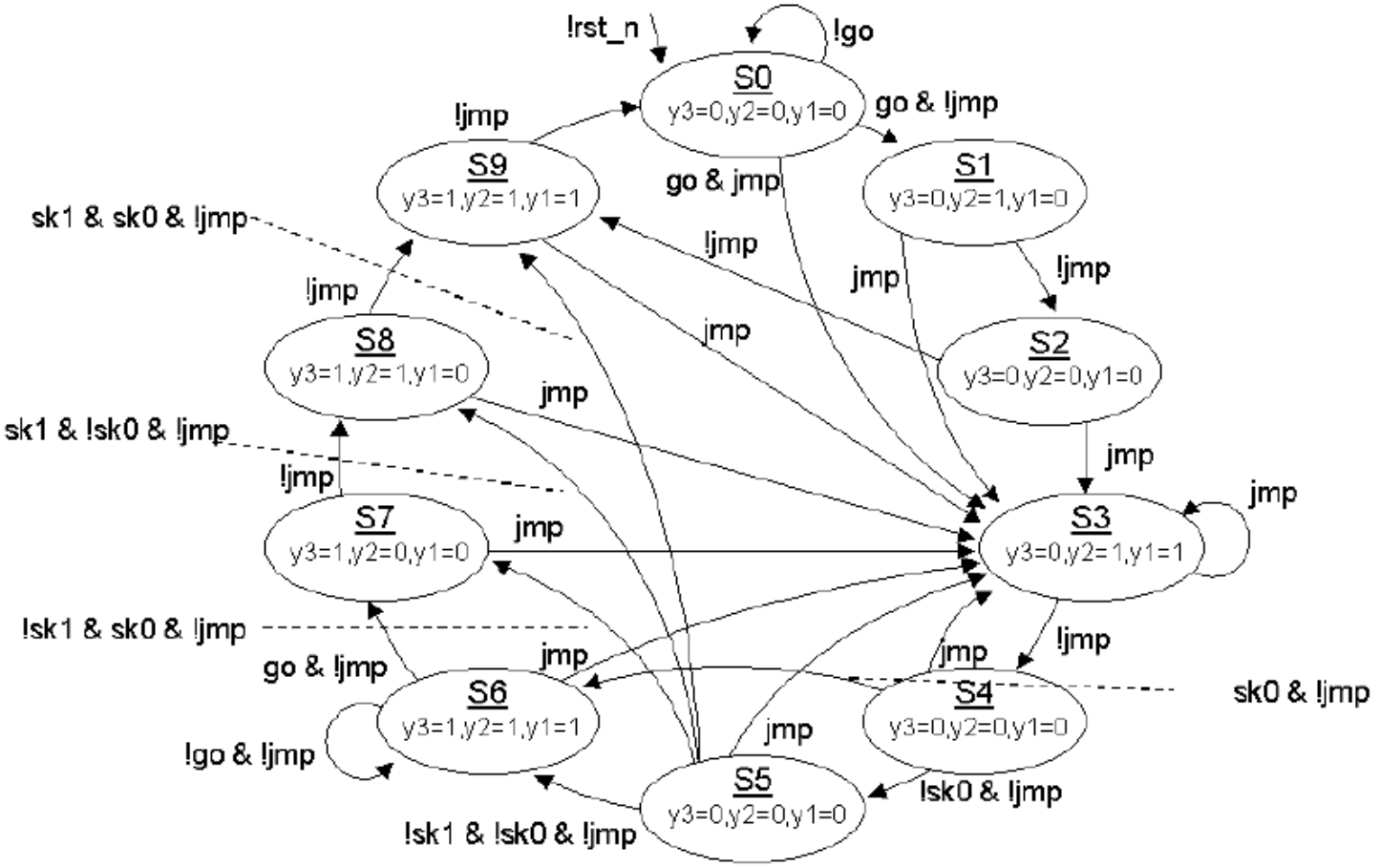
计数器的电路实现



思考：3、4位计数器的
电路实现？？？

复杂状态机

复杂状态机


!rst_n
!go
S0
y3=0,y2=0,y1=0
go & !jmp
!jmp
S9
y3=1,y2=1,y1=1
S1
y3=0,y2=1,y1=0
go & jmp
sk1 & sk0 & !jmp
!jmp
!jmp
jmp
jmp
S8
y3=1,y2=1,y1=0
S2
y3=0,y2=0,y1=0
jmp
sk1 & !sk0 & !jmp
jmp
!jmp
jmp
S7
y3=1,y2=0,y1=0
S3
y3=0,y2=1,y1=1
jmp
!sk1 & sk0 & !jmp
!jmp
go & !jmp
jmp
jmp
S6
y3=1,y2=1,y1=1
S4
y3=0,y2=0,y1=0
sk0 & !jmp
!go & !jmp
jmp
S5
y3=0,y2=0,y1=0
!sk0 & !jmp
!sk1 & !sk0 & !jmp

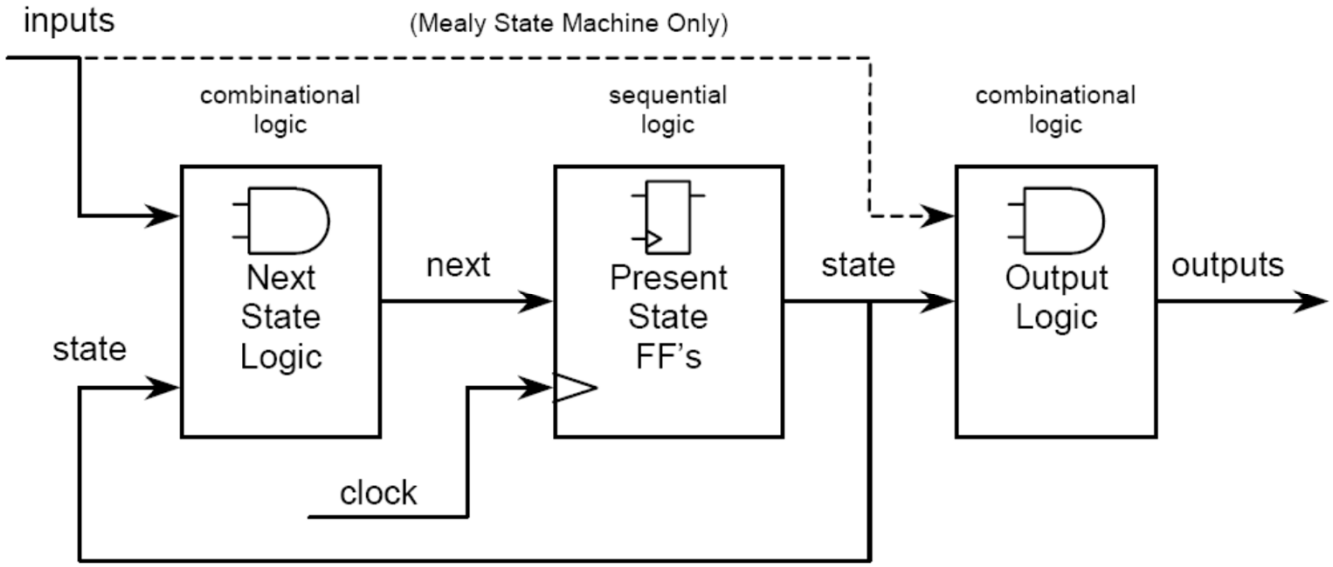
状态机分类

- 摩尔型（Moore）

摩尔型状态机的输出只与当前状态有关，与输入无关。

- 米利型（Mealy ）

米利型状态机的输出不仅与当前状态有关，还与当前输入有关。



怎样设计一个状态机

- 确定状态机的基本行为
- 建立状态、输出表
- 优化状态机状态数目（optional）
- 给状态机编码
- 以状态、输出表为基础建立转换、输出表（图）
- 求输出表达式
- 完成完整的状态机设计

状态机举例

- 设计一个自动售货机的逻辑控制电路（只剩下听装可乐，每听的价格是3.5元）。它有两个投币口分别为一元投币口和五毛投币口，假设每次只能投入一枚一元或五毛硬币，投入三元五毛硬币后机器自动给出一听可乐，投入四元硬币后，在给出一听可乐的同时找回一枚五毛的硬币。

功能抽象

- 取投币信号为输入逻辑变量，投入一枚一元硬币时用A=1表示，未投入时A=0。投入一枚五毛硬币用B=1表示，未投入时B=0。给出可乐和找钱为两个输出变量，分别用O和C表示，给出可乐时O=1，不给时O=0，找回一枚五毛硬币时C=1，不找时C=0。

建立状态与输出表

- 根据上面的功能描述，可用7个状态S0,S1,S2,S3,S4,S5,S6 表示，未投币前的初始状态为S0，投入五毛硬币以后为S1，投入一元硬币后(包括投入一枚一元硬币和投入两枚五毛硬币的情况)为S2。依次类推，S6表示共投入3块，这时新投入的若是5毛，同时输出为O=1，C=0，电路返回S0；如果投入的是一枚一元硬币，则电路也应能返回S0，同时输出为O=1，C=1。

上海交通大学 Shanghai Jiao Tong University

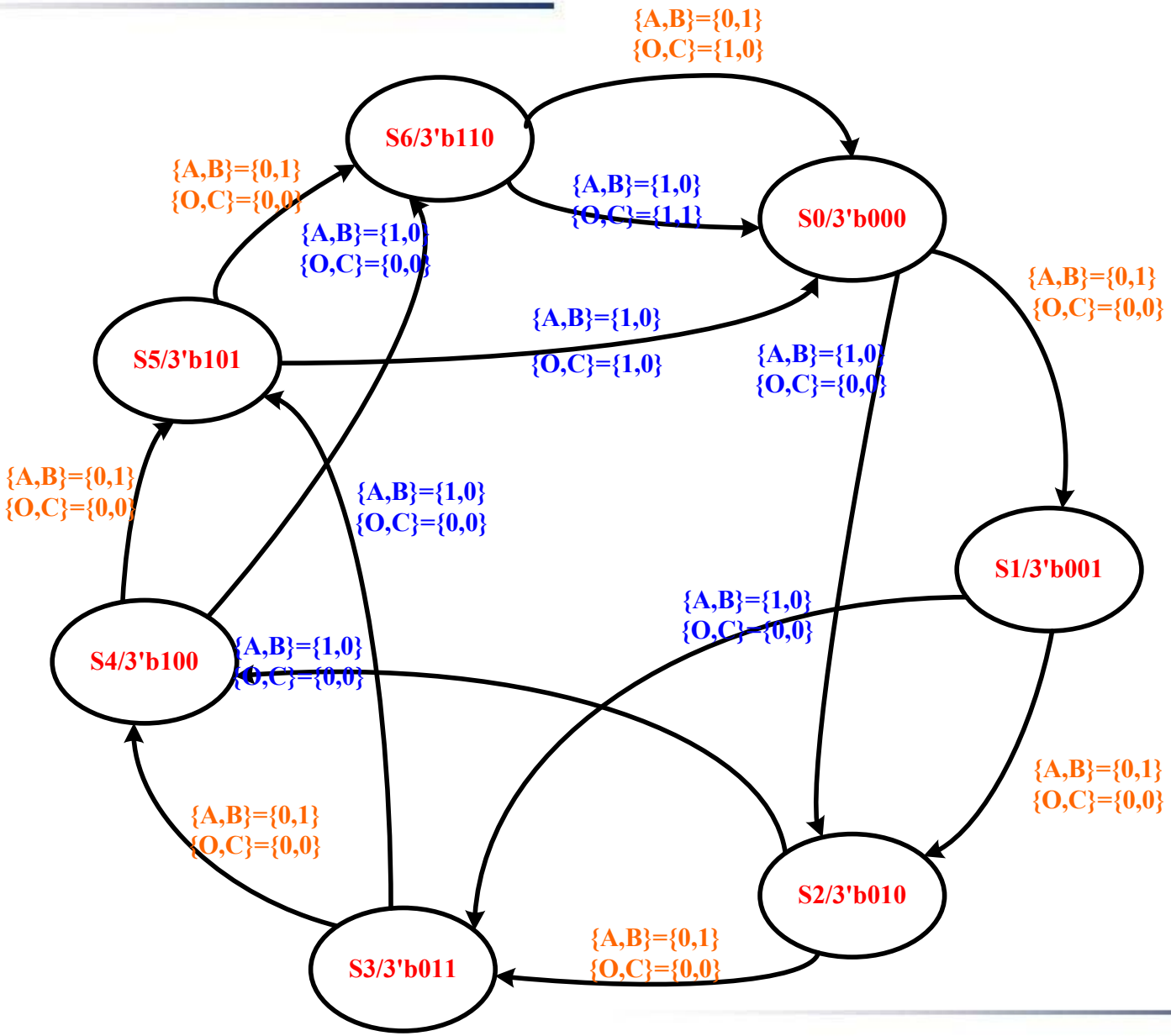
状态输出表

状态	输入{A,B}/{一元/5毛}			
	{1,0}	{0,1}	{0,0}	{1,1}
S0	next_state=S2 {o,c}={0,0}	next_state=S1 {o,c}={0,0}	next_state=S0 {o,c}={0,0}	×
S1	next_state=S3 {o,c}={0,0}	next_state=S2 {o,c}={0,0}	next_state=S1 {o,c}={0,0}	×
S2	next_state=S4 {o,c}={0,0}	next_state=S3 {o,c}={0,0}	next_state=S2 {o,c}={0,0}	×
S3	next_state=S5 {o,c}={0,0}	next_state=S4 {o,c}={0,0}	next_state=S3 {o,c}={0,0}	×
S4	next_state=S6 {o,c}={0,0}	next_state=S5 {o,c}={0,0}	next_state=S4 {o,c}={0,0}	×
S5	next_state=S0 {o,c}={1,0}	next_state=S6 {o,c}={0,0}	next_state=S5 {o,c}={0,0}	×
S6	next_state=S0 {o,c}={1,1}	next_state=S0 {o,c}={1,0}	next_state=S6 {o,c}={0,0}	×

状态机编码

状态	编码
S0	3′ b000
S1	3′ b001
S2	3′ b010
S3	3′ b011
S4	3′ b100
S5	3′ b101
S6	3′ b110

状态转换图


S6/3'b110
S0/3'b000
S5/3'b101
S1/3'b001
S4/3'b100
S2/3'b010
S3/3'b011
{A,B}={0,1}
{O,C}={1,0}
{A,B}={0,1}
{O,C}={0,0}
{A,B}={1,0}
{O,C}={1,1}
{A,B}={1,0}
{O,C}={0,0}
{A,B}={0,1}
{O,C}={0,0}
{A,B}={1,0}
{O,C}={1,0}
{A,B}={1,0}
{O,C}={0,0}
{A,B}={0,1}
{O,C}={0,0}
{A,B}={1,0}
{O,C}={0,0}
{A,B}={1,0}
{O,C}={0,0}
{A,B}={0,1}
{O,C}={0,0}
{A,B}={0,1}
{O,C}={0,0}
{A,B}={0,1}
{O,C}={0,0}

输出表达式

- 输出可乐（O）

```
O=(state==S5 || state==S6) &&  (a==1'b1)  ||
    (state==S6) && (b==1'b1);
```

- 找零（C）

```
C= (state==S6) && (a==1'b1) ;
```

parameter与`define

- parameter 和`define是verilog中用来定义常量的两种方法，常用于定义变量宽度、延时、状态 。

- paramter 参数定义的语法：
  parameter <list_of_assignment>;
  可一次定义多个参数，用逗号隔开。

parameter举例

- 在使用文字的地方都可以使用参数，参数的定义是局部的，只在当前模块中有效。

```
module test (data_in,data_out);

parameter size=4'd8;

wire [size-1:0] datain;
wire [size-1:0] data_out;
……


endmodule
```

```
module fsm (clk, rst_n);


……
parameter    IDLE   = 2'd0;
parameter    BUSY = 2'd1;
parameter    WAIT  = 2'd2;
parameter    FREE  = 2'd3;
……

endmodule
```

parameter 重载（1）

- 使用defparam重载parameter

```
module top;
reg clk;
reg [0:4] in1;
reg [0:9] in2;
wire [0:4] o1;
wire [0:9] o2;
vdff m1 (o1, in1, clk);
vdff m2 (o2, in2, clk);
endmodule
```

```
module annotate;
defparam
    top.m1.size = 5,
    top.m1.delay = 10,
    top.m2.size = 10,
    top.m2.delay = 20;
endmodule
```

```
module vdff (out, in, clk);
parameter size = 1, delay = 1;
input [0:size-1] in;
input clk;
output [0:size-1] out;
reg [0:size-1] out;

always @(posedge clk)
# delay out = in;

endmodule
```

parameter重载（2）

- 在例化时重载parameter

```
module vdff (out, in, clk);

parameter size = 1, delay = 1;

input [0:size-1] in;
input clk;
output [0:size-1] out;

reg [0:size-1] out;

always @(posedge clk)
# delay out = in;

endmodule
```

```
module m;
reg clk;
wire[1:10] out_a, in_a;
wire[1:5] out_b, in_b;

// create an instance and set parameters
vdff #(10,15) mod_a(out_a, in_a, clk);

// create an instance leaving default values
vdff mod_b(out_b, in_b, clk);

endmodule
```

`define

- 编译指导`define提供了一种简单的文本替换的功能

  `define <macro_name> <macro_text>

  在编译时<macro_text>替换<macro_name>。可提高描述的可读性。

```
`define not_delay #1
`define and_delay #2
`define or_delay #1
module MUX2_1 (out, a, b, sel);
output out;
input a, b, sel;
not `not_delay not1( sel_, sel);
and `and_delay and1( a1, a, sel_);
and `and_delay and2( b1, b, sel);
or `or_delay or1( out, a1, b1);
endmodule
```

```
`undef text_macro_name
```

`undef编译指导
撤销宏定义

自动贩卖机代码举例(1)

```
module vending_machine(a,b,clk,rst_n,o,c);
input   a,b,clk,rst_n;
output  o,c;
wire o,c;
parameter S0=3'b000;
parameter S1=3'b001;
parameter S2=3'b010;
parameter S3=3'b011;
parameter S4=3'b100;
parameter S5=3'b101;
parameter S6=3'b110;
reg [2:0]   state,next_state;

always @(posedge clk or negedge rst_n)
if(!rst_n)
   state<=3'b0;
else
   state<=next_state;
……
```

自动贩卖机代码举例(2)

```
always @(a or b or state) begin
   case(state)
      3'b000:if ({a,b}=={1'b1,1'b0})
                next_state=S2;
             else if({a,b}=={1'b0,1'b1})
                next_state=S1;
             else
                next_state=S0;
      3'b001:if ({a,b}=={1'b1,1'b0})
                next_state=S3;
             else if({a,b}=={1'b0,1'b1})
                next_state=S2;
             else
                next_state=S1;
      3'b010:if ({a,b}=={1'b1,1'b0})
                next_state=S4;
             else if({a,b}=={1'b0,1'b1})
                next_state=S3;
             else
                next_state=S2;
3'b011:if ({a,b}=={1'b1,1'b0})
                next_state=S5;
             else if({a,b}=={1'b0,1'b1})
                next_state=S4;
             else
                next_state=S3;
3'b100:if ({a,b}=={1'b1,1'b0})
                next_state=S6;
             else if({a,b}=={1'b0,1'b1})
                next_state=S5;
             else
                next_state=S4;
3'b101:if ({a,b}=={1'b1,1'b0})
                next_state=S0;
             else if({a,b}=={1'b0,1'b1})
                next_state=S6;
             else
                next_state=S5;
3'b110:if ({a,b}=={1'b1,1'b0})
                next_state=S0;
             else if({a,b}=={1'b0,1'b1})
                next_state=S0;
             else
                next_state=S6;
default:next_state=S0;endcase    end
```

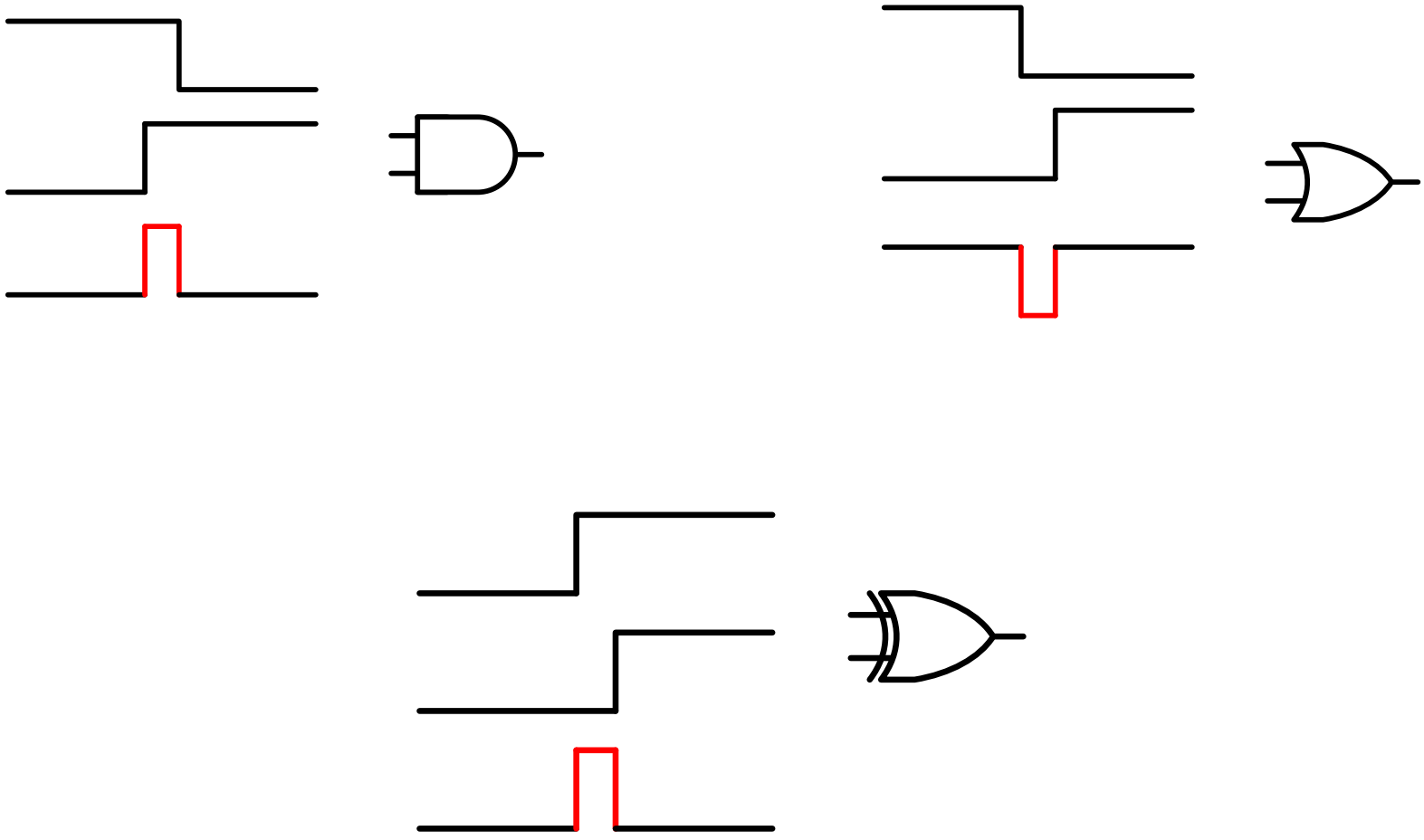
自动贩卖机代码举例(3)

```
assign o=    ((state==S5 || state==S6) &&  a==1'b1)
                ||  (state==S6 && b==1‘b1);

assign c= state==S6 && a==1‘b1;

endmodule
```

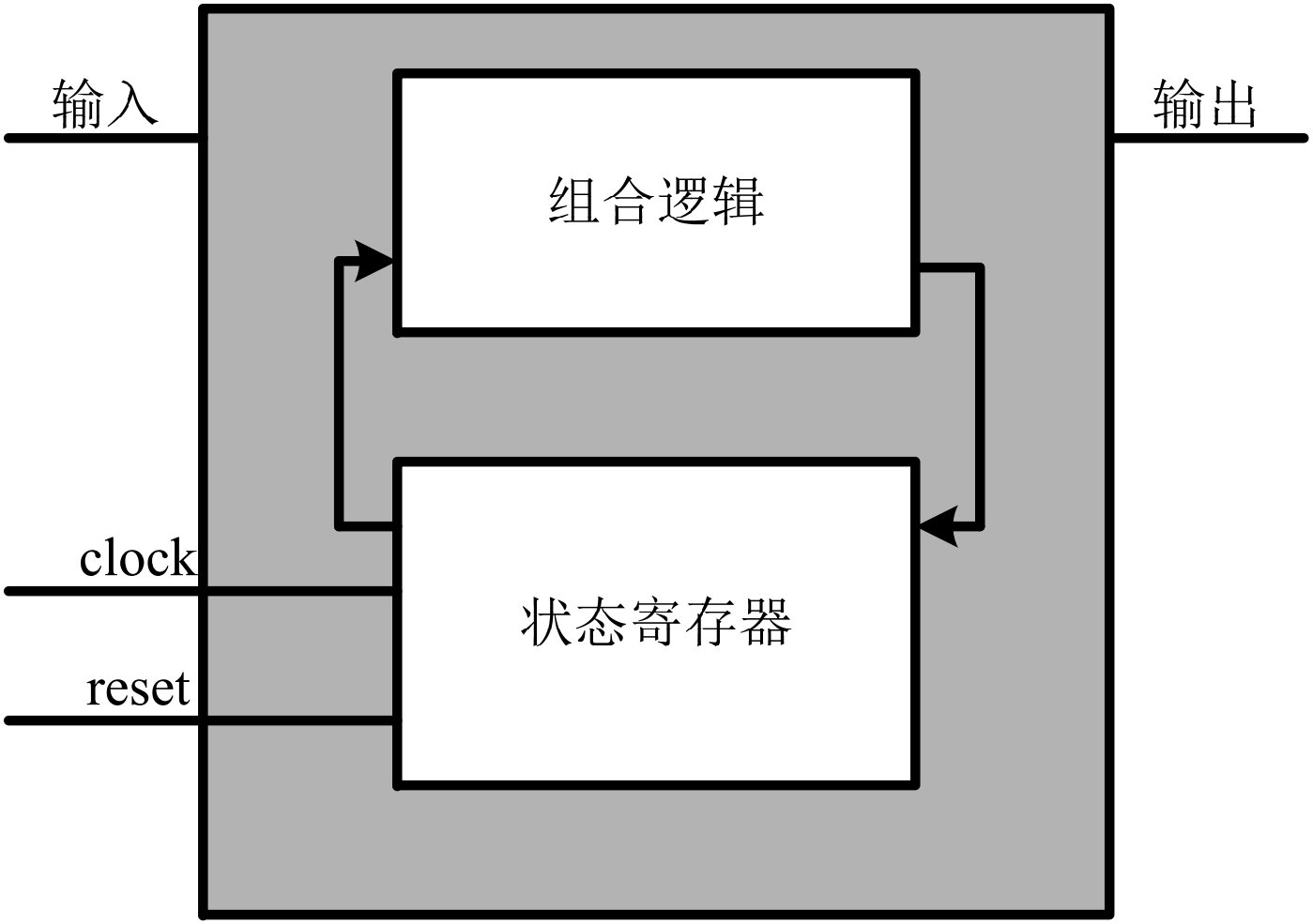
状态机的编码风格

- 稳定性好，稳定性指FSM的输出要无毛刺等异常扰动，且状态机要完备，即使进入异常扰动也能快速恢复到正常状态。
- FSM速度快，满足设计的频率要求。
- FSM面积小，满足设计的面积要求。
- FSM设计要清晰易懂、易维护

组合逻辑的毛刺



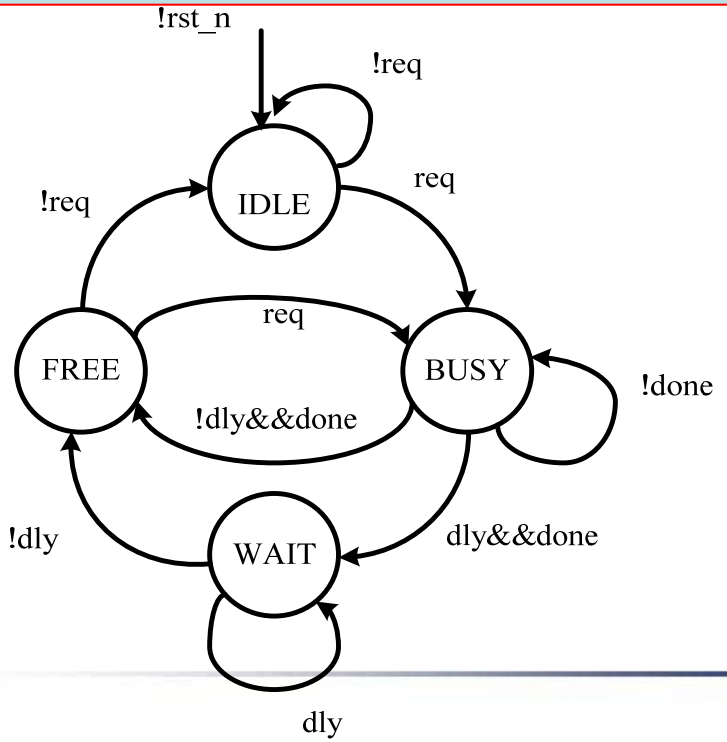
状态机的结构特点



两段式经典状态机设计

```
module fsm(dly, done, req, clk, rst_n);
input dly, done, req, clk, rst_n;
parameter          IDLE   = 2'd0;
parameter          BUSY   = 2'd1;
parameter          WAIT   = 2'd2;
parameter         FREE   = 2'd3;
reg [1:0] current_state,next;
always @(posedge clk or negedge rst_n)
if (!rst_n) current_state <= IDLE;
else        current_state <= next;
always @(current_state or dly or done or
req)
begin
next = 2'bx;
case (current_state)
IDLE : if (req) next = BUSY;
          else next = IDLE;
BUSY: begin
          if (!done) next = BUSY;
          else if ( dly) next = WAIT;
          else next = FREE;  end
```

```
WAIT: begin
          if (!dly) next = FREE;
          else next = WAIT;
          end
FREE: if (req) next = BUSY;
           else next = IDLE;
endcase
end
endmodule
```



两段式状态机的设计风格

- 用两个always块来描述状态机，一个 always块用来描述新状态的产生（组合逻辑），一个always块用来描述状态迁移（时序逻辑）。
- 使用parameter定义状态。
- 时序逻辑使用非阻塞赋值，组合逻辑使用阻塞赋值。
- 组合逻辑的always块中，敏感列表需要包括当前状态和所有输入。
- 在组合逻辑块的开始，指定next state的缺省值。

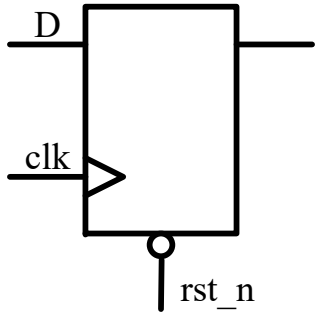
next state的初始值

- 一般来说next state有三种初始值:2′ bx、ilde、其他state。
- 当逻辑中没有出现next state分配的时候，初始值有利于debug。

```
always @(state or dly or done or req) begin
next = 2'bx;
case (state)
IDLE : if (req) next = BUSY;
          else next = IDLE;
BUSY: if (!done) next = BUSY;
          else if ( dly) next = WAIT;
          else next = FREE;
WAIT: if (!dly) next = FREE;
          else next = WAIT;
FREE: if (req) next = BBUSY;
          else next = IDLE;
endcase end
```

电路的初始态

- 在数字芯片中，一般有一个reset过程，使电路有一个初始值。



- 组合逻辑输入主要来自寄存器，还有一些输入是恒0、恒1输入。一般来说，组合逻辑的初始化是没有必要的。

状态的编码

- 普通二进制编码

  3’b000,3’b001,3’b010,3’b011…

  编码紧凑，节省寄存器资源。状态之间翻转多，影响速度。

- 独热码（one-hot）

  3’b001,3’b010,3’b100…

  速度快，设计简单，易于维护，消耗寄存器资源多。

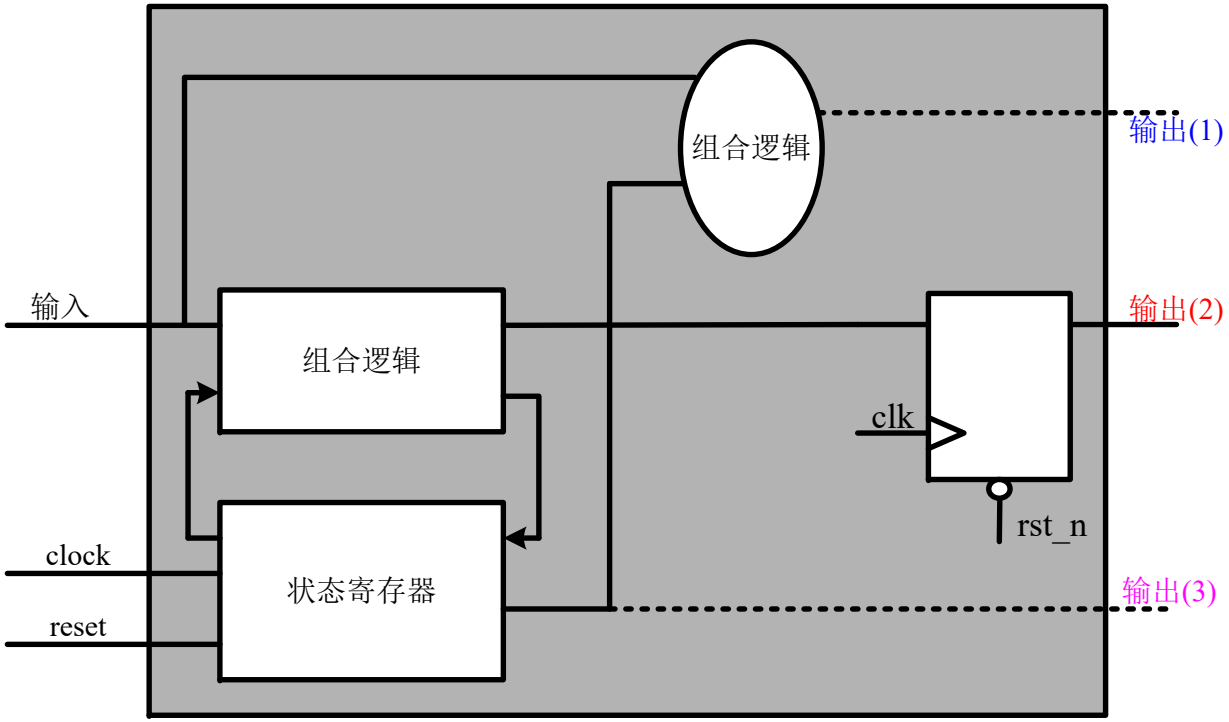
- 格莱码

  3’b000,3’b001,3’b011,3’b010…

  状态之间翻转少，状态的编码分配较困难。

考虑输出的状态设计

- 状态机的常用输出方式

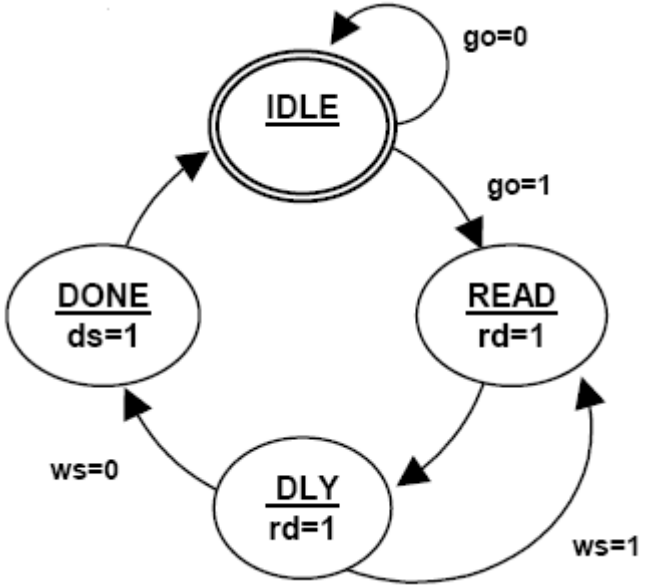


组合逻辑输出 ——容易产生毛刺
寄存器输出 ——减少毛刺、提高电路速度
利用状态编码输出 ——高效

组合逻辑输出(1)

```
module fsm1a (ds, rd, go, ws, clk, rst_n);
output ds, rd;
input go, ws,clk, rst_n;
parameter [1:0]   IDLE = 2'b00,
 READ = 2'b01,DLY = 2'b10,DONE = 2'b11;
reg [1:0] state, next;
always @(posedge clk or negedge rst_n)
if (!rst_n) state <= IDLE;
else        state <= next;
always @(state or go or ws) begin
next = 2'bx;
case (state)
IDLE:     if (go) next = READ;
          else next = IDLE;
READ:   next = DLY;
DLY:      if (ws) next = READ;
           else next = DONE;
DONE: next = IDLE;
endcase    end
```

```
assign rd = (state==READ ||
state==DLY);
assign ds = (state==DONE);
endmodule
```



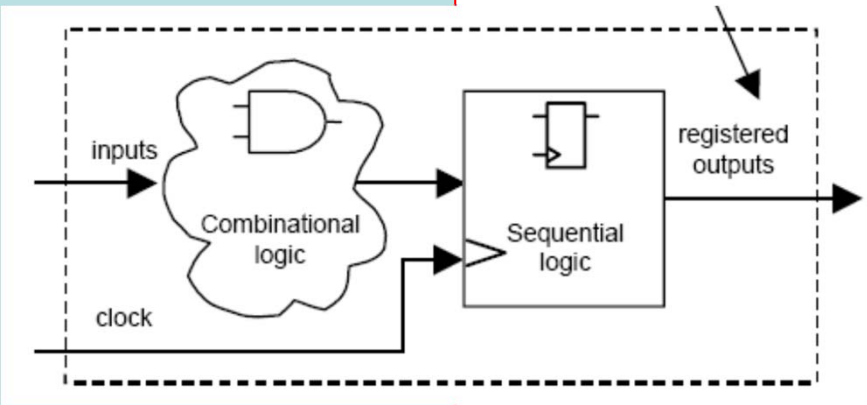
组合逻辑输出（2）

```
module fsm1a (ds, rd, go, ws, clk,
rst_n);
output ds, rd;
input go, ws;
input clk, rst_n;
parameter [1:0]   IDLE = 2'b00,
                  READ = 2'b01,
                  DLY = 2'b10,
                  DONE = 2'b11;
reg [1:0] state, next,ds,rd;

always @(posedge clk or negedge
rst_n)

if (!rst_n)   state <= IDLE;
else          state <= next;
```

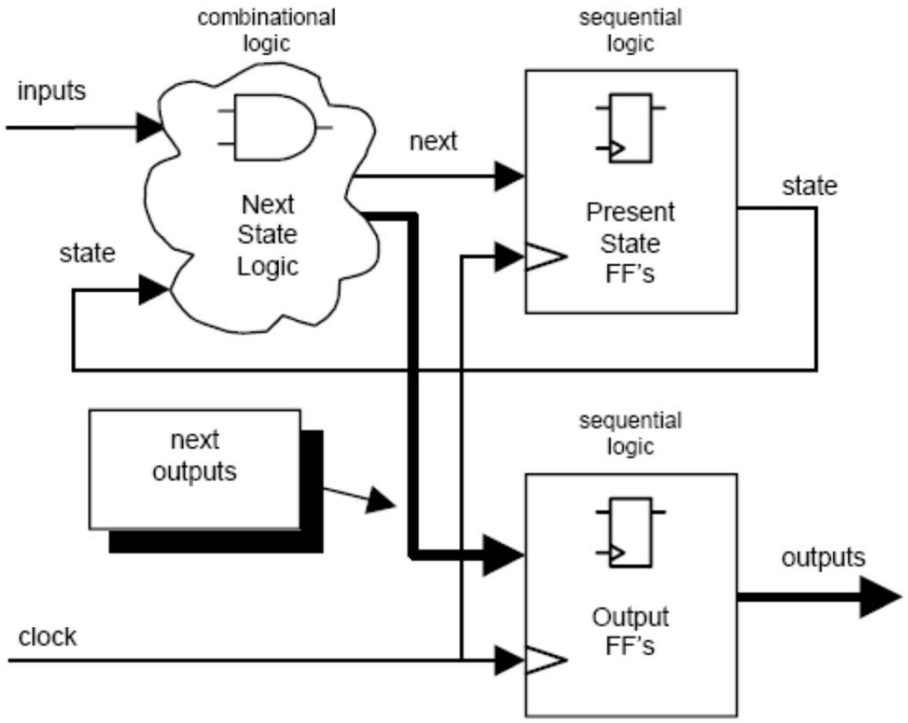
```
always @(state or go or ws) begin
next = 2'bx;  ds = 1'b0;  rd = 1'b0;
case (state)
IDLE: if (go) next = READ;
        else next = IDLE;
READ: begin rd = 1'b1;
         next = DLY; end
DLY: begin rd = 1'b1;
       if (ws) next = READ;
       else next = DONE;
       end
DONE: begin ds = 1'b1;
          next = IDLE; end
endcase
end
endmodule
```

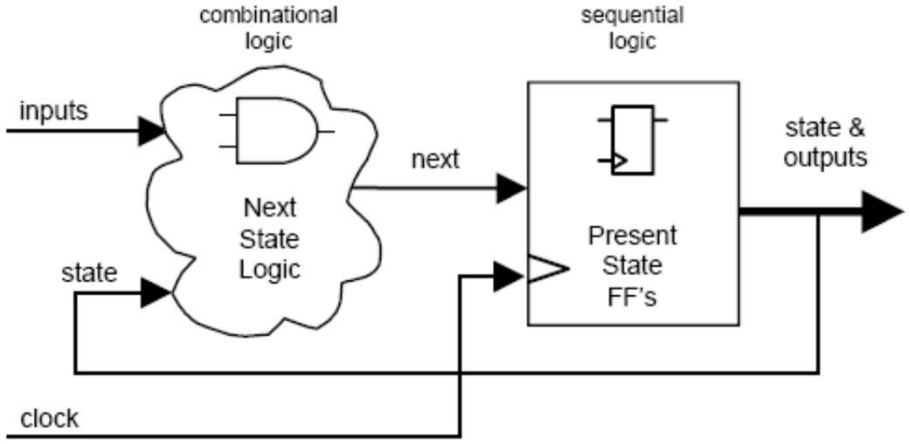
寄存器输出

```
always @(posedge clk or negedge rst_n)
if (!rst_n) begin
ds <= 1'b0;
rd <= 1'b0;
end
else begin
ds <= 1'b0;
rd <= 1'b0;
case (state)
        IDLE:     if (go) rd <= 1'b1;
        READ:    rd <= 1'b1;
        DLY:     if (ws) rd <= 1'b1;
                 else ds <= 1'b1;
endcase
end
```

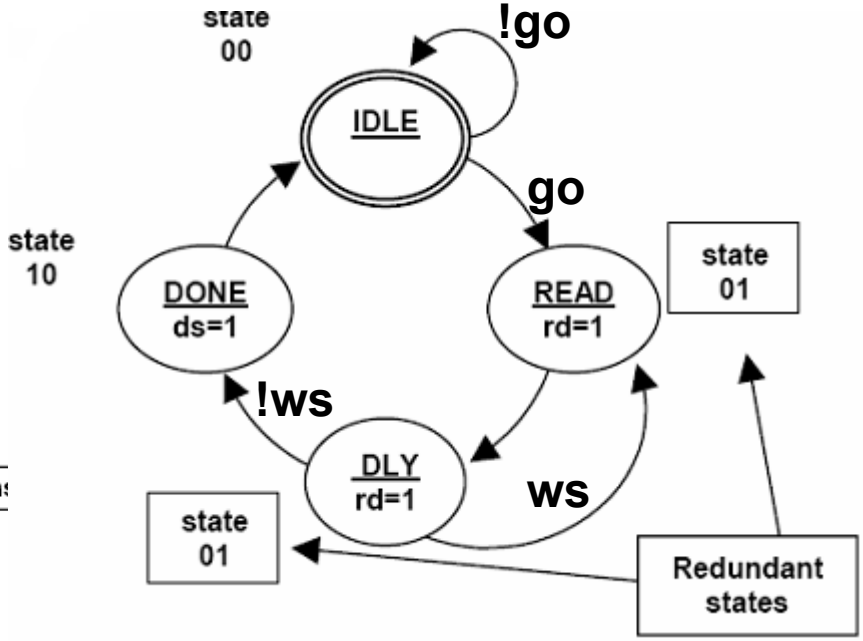


利用状态编码输出


combinational logic
sequential logic
inputs
next
state
Next State Logic
Present State FF's
state
next outputs
sequential logic
outputs
Output FF's
clock


combinational logic
sequential logic
inputs
next
state & outputs
Next State Logic
Present State FF's
state
clock

状态图与状态编码分析



Output columns

State	ds	rd
IDLE	0	0
READ	0	1
DLY	0	1
DONE	1	0

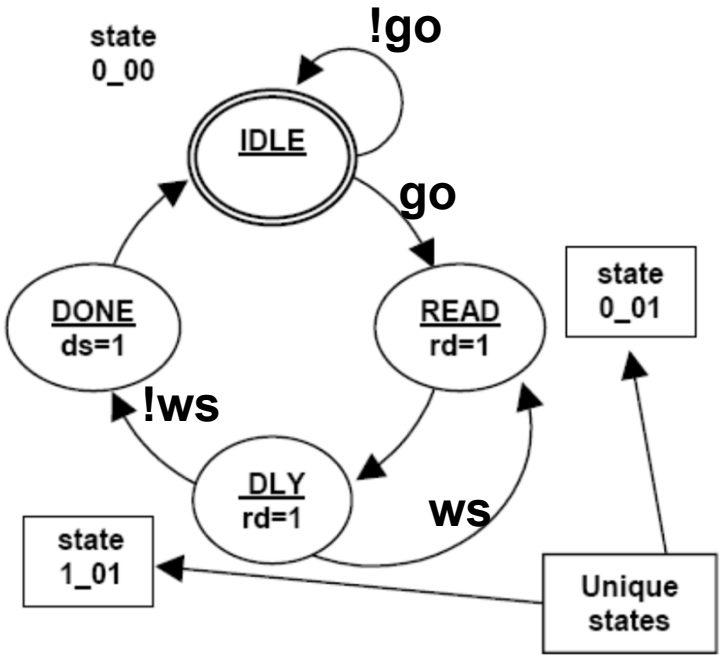
State rows

state
10

修改状态编码

state	x0	ds	rd
IDLE	0	0	0
READ	0	0	1
DLY	1	0	1
DONE	0	1	0

state
0_10



代码举例

```
module fsm (ds, rd, go, ws, clk, rst_n);
output ds, rd;
input go, ws;
input clk, rst_n;
parameter [2:0]   IDLE   = 3'b0_00,
                  READ = 3'b0_01,
                  DLY    = 3'b1_01,
                  DONE = 3'b0_10;
reg [2:0] state, next;
always @(posedge clk or negedge rst_n)
if (!rst_n) state <= IDLE;
else        state <= next;
```

```
always @(state or go or ws) begin
next = 3'bx;
case (state)
IDLE:   if (go) next = READ;
        else next = IDLE;
READ: next = DLY;
DLY:    if (ws) next = READ;
        else next = DONE;
DONE: next = IDLE;
endcase
end
assign {ds,rd} = state[1:0];
endmodule
```