计数器

```
module Counter8(
    input CLK,//时钟信号，上升沿有效
    input rst_n,//异步复位信号，低电平有效
    output[2:0]oQ,//二进制计数器输出
    output[6:0]oDisplay//七段数码管显示输出   );
    //Divider #(100000000) divider (CLK, ~rst_n, clk);
    JK_FF JK1(CLK,1'b1,1'b1,rst_n,oQ[0]);
    JK_FF JK2(CLK,oQ[0],oQ[0],rst_n,oQ[1]);
    JK_FF JK3(CLK,oQ[1]&oQ[0],oQ[1]&oQ[0],rst_n,oQ[2]);
    display7 utt({1‘b0,oQ}, oDisplay);
endmodule
```

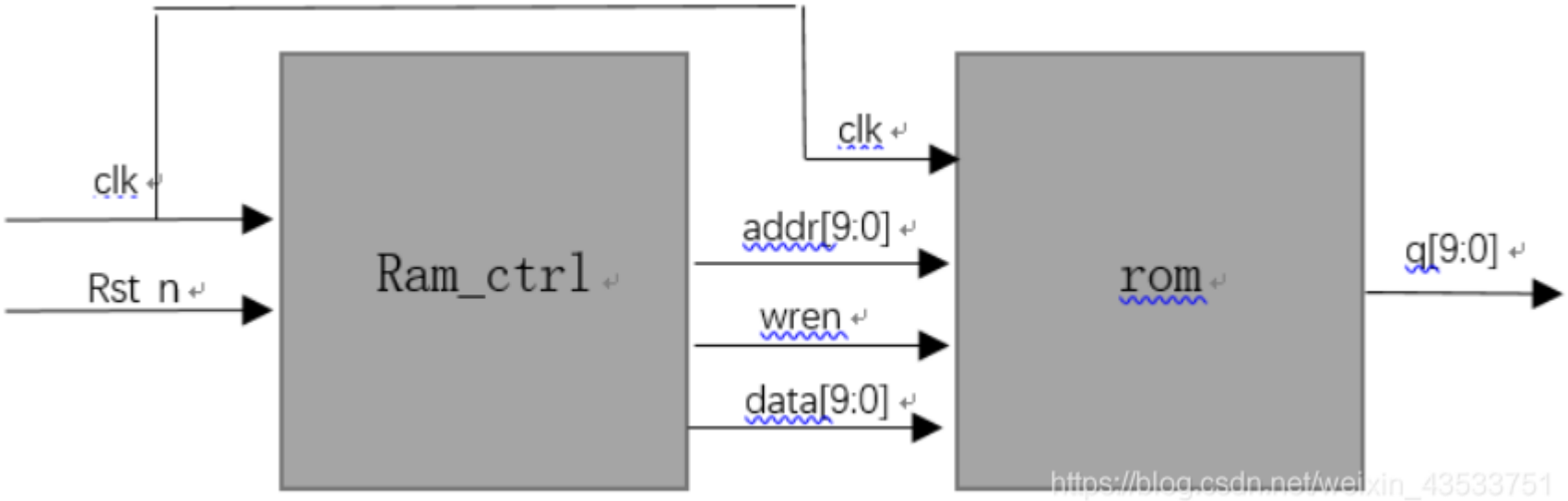
计数器

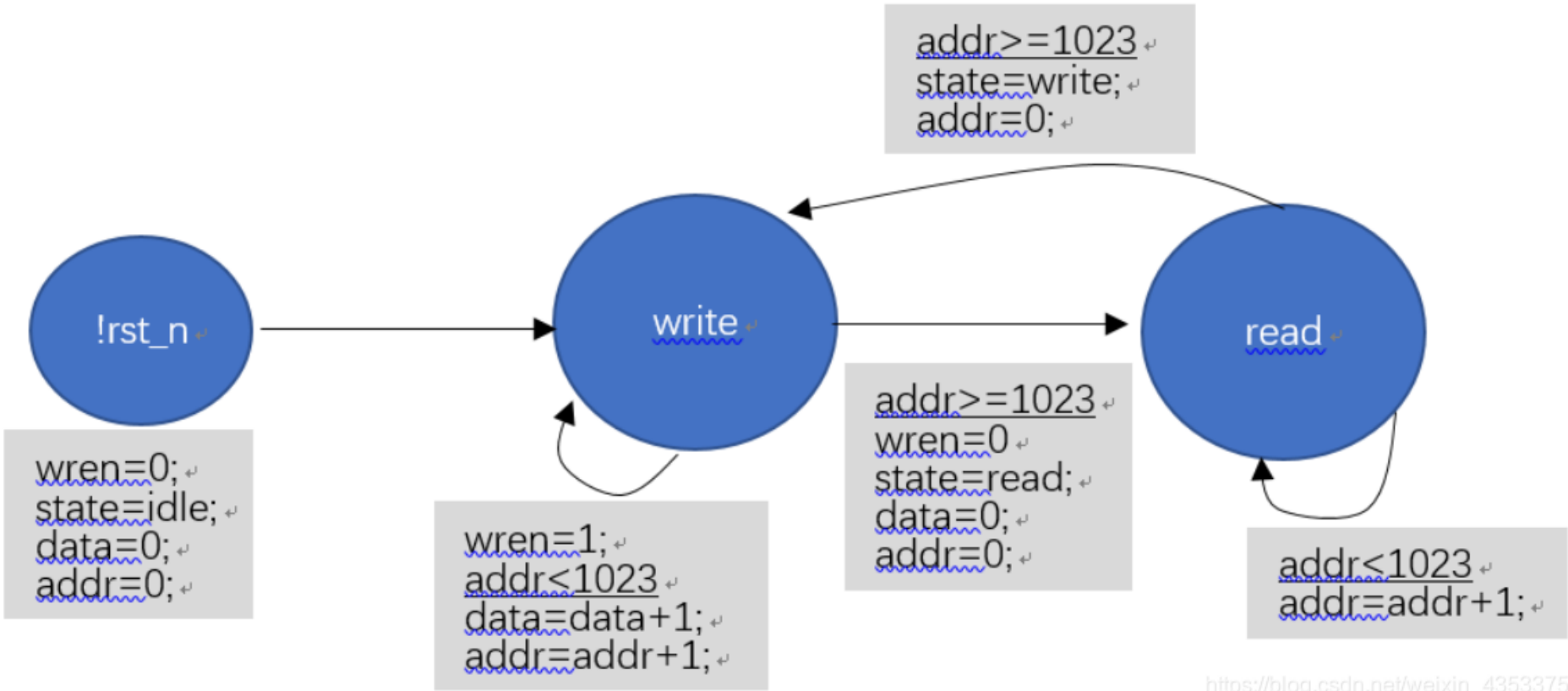
```
module Divider(I_CLK,rst,O_CLK);
    input I_CLK; //输入时钟信号，上升沿有效
    input rst;  //同步复位信号，高电平有效
    output reg O_CLK;//输出时钟
    parameter N=20;
    integer count=0;
        always@(posedge I_CLK or posedge rst)
        begin
            if(rst)
                begin
                    O_CLK<=0;
                    count<=0;
                end
            else
                begin
                    if(count==N/2-1)
                        begin
                            O_CLK<=~O_CLK;
                            count<=0;
                        end
                    else
                        count<=count+1;
                end
        end
endmodule
```

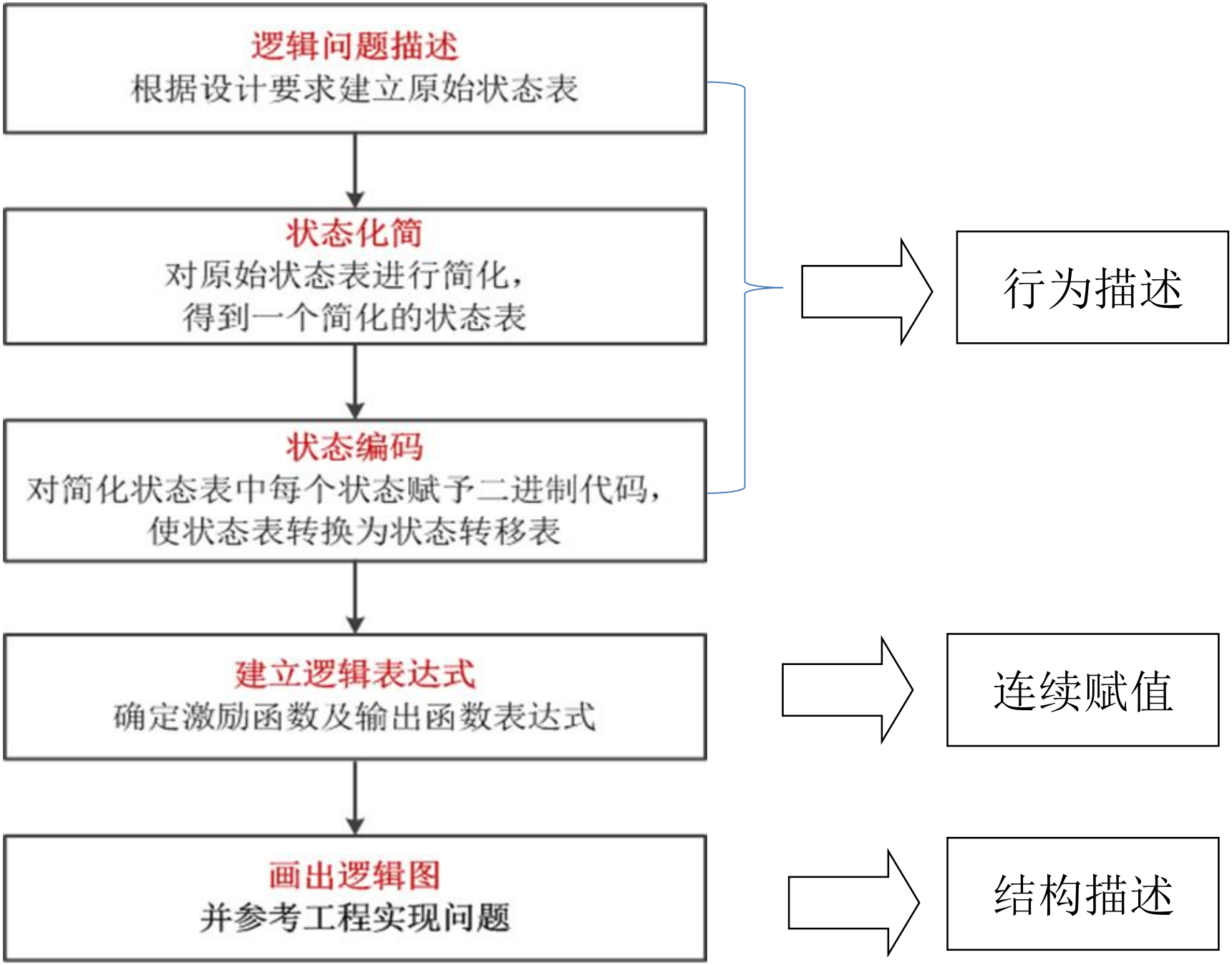
ram

```
Module  ram(clk,ena,wena,addr,data_in,data_out);
        input clk;    input ena;   input wena;    input [4:0]addr;
         input [31:0]data_in;    output reg [31:0] data_out;
         reg [31:0] mem[0:31];
        always@(posedge clk)  begin  //write to ram
            if(ena)
              begin
                    if(wena)begin          //write to ram
                         mem[addr]<=data_in;
                    end
                    else begin              //read from ram
                         data_out<=mem[addr];
                   end
              end
              else
                    begin
                          data_out<=32'bz;
                    end
              End
endmodule
```

```
always@(posedge clk or negedge rst_n)
    if(!rst_n)
        begin
            state<=write;
            addr<=10'd0;
            data<=10'd0;
            wren<=0;
        end
    else begin
        case(state)
            write:begin
                        wren<=1;
                        if(addr<10'd1023)
                            begin
                                addr<=addr+1'b1;
                                data<=data+1'b1;
                            end
                        else
                            begin
                                wren<=0;
                                addr<=1'b0;
                                data<=1'b0;
                                state<=read;
                            end
                    end
            read :begin
                        if(addr<10'd1023)
                            addr<=addr+1'b1;
                        else
                            begin
                                addr<=10'd1023;
                            end
                    end
            default:state<=write;
        endcase
    end
```






逻辑问题描述
根据设计要求建立原始状态表
状态化简
对原始状态表进行简化，
得到一个简化的状态表
状态编码
对简化状态表中每个状态赋予二进制代码，
使状态表转换为状态转移表
建立逻辑表达式
确定激励函数及输出函数表达式
画出逻辑图
并参考工程实现问题
行为描述
连续赋值
结构描述

状 态 机

Finite State Machine

定义

- 有限状态机（Finite-State Machine，FSM），简称状态机，是表示有限个状态以及在这些状态之间的转移和动作等行为的数学模型。状态机不仅是一种电路的描述工具，而且也是一种思想方法，在电路设计的**系统级和 RTL 级**有着广泛的应用。
- 同步时序逻辑的设计，能够在有限个状态之间按一定要求和规律切换时序电路的状态。状态的切换方向不但取决于各个输入值，还取决于当前所在状态。

同步状态机

● 同步状态机的结构

- 由状态寄存器（触发器）作为状态记忆部件（常用正跳边沿触发的D触发器）
- 仅当触发信号到达时刻才可能发生状态改变
- n个触发器最多有2^n 个状态
- 两种同步状态机：

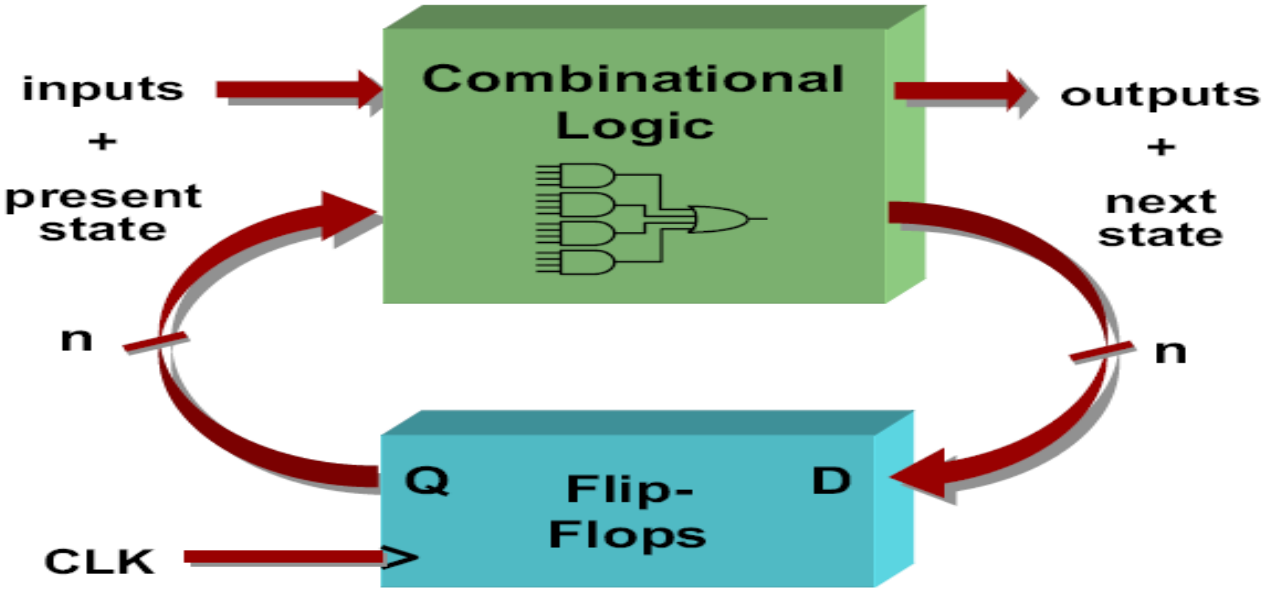
Mealy型----- 下一个输出是当前状态和输入的函数

下一个状态是当前状态和输入的函数

Moore型 -----下一个输出是当前状态的函数

下一个状态是当前状态和输入的函数

FSM的结构

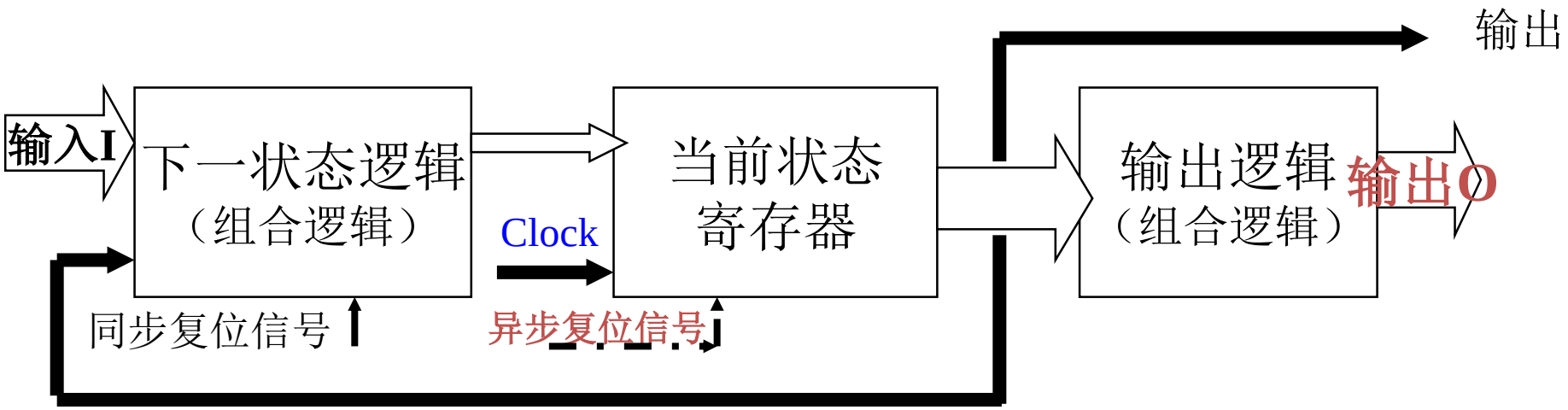


- *Current State Register*
- *Next State Logic：* $NS = f(PS, I)$
- *Output Logic*

FSM的分类

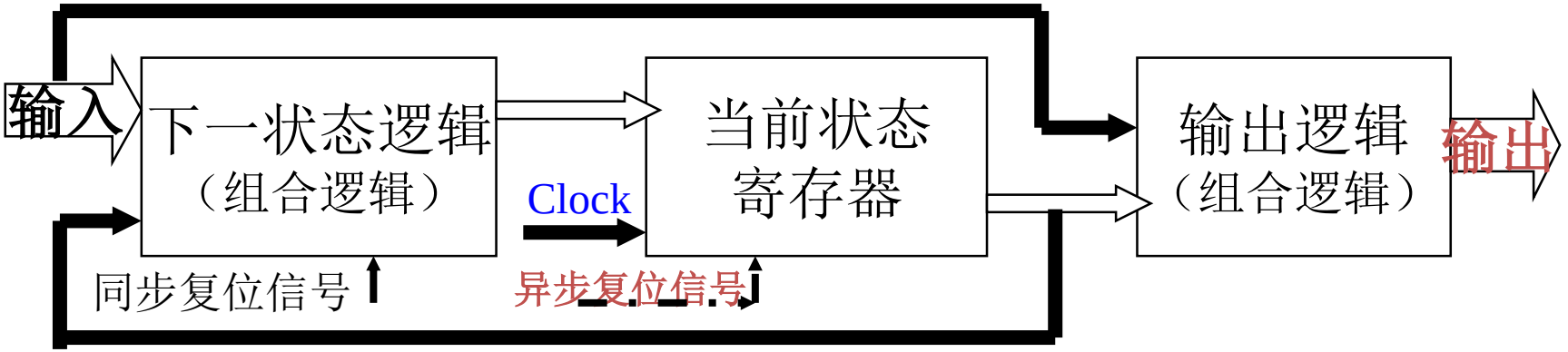
- Moore型
- Mealy型
- Mealy/Moore混合型

Moore有限状态机模型



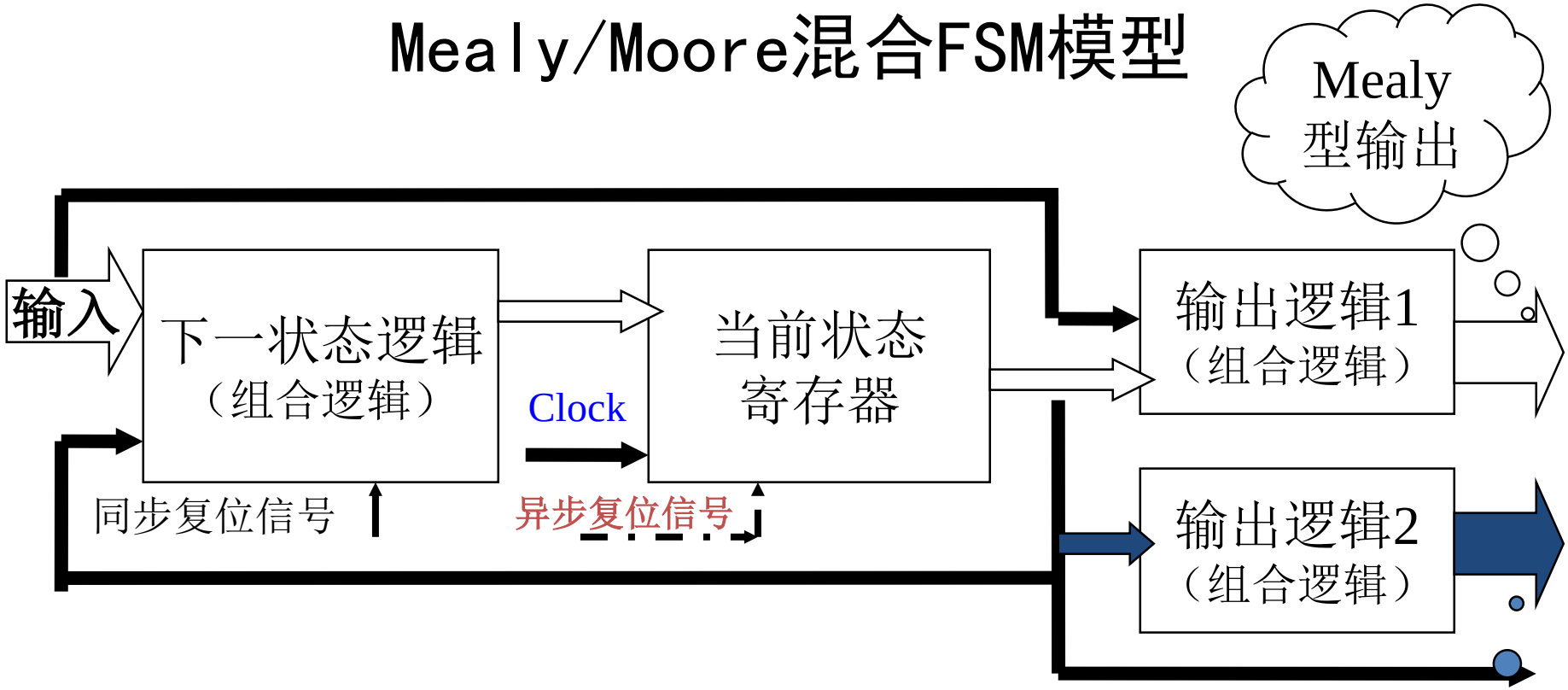
$$
\begin{cases}
I：\ \text{输入信号} & O：\ \text{输出信号} \\
PS：\ \text{FSM当前状态} & NS：\ \text{FSM下一个状态} \\
NS = f(PS,\ I) & \\
O = h(PS) & \\
PS = \begin{cases} \text{FSM初始状态S0} & \text{异步复位信号有效} \\ NS & \text{Clock有效跳变沿} \end{cases}
\end{cases}
$$

Mealy有限状态机模型



$$
\begin{cases}
I：\ \text{输入信号} \qquad\qquad O：\ \text{输出信号} \\
PS：\ \text{FSM当前状态} \qquad NS：\ \text{FSM下一个状态} \\
NS = f(PS,\ I) \\
O = h(PS,\ I) \\
PS = \begin{cases} \text{FSM初始状态S0} & \text{异步复位信号有效} \\ NS & \text{Clock有效跳变沿} \end{cases}
\end{cases}
$$

Mealy/Moore混合FSM模型



I ： 输入信号　　O＝{Omoore Omealy }： 输出信号

PS： FSM当前状态　　NS： FSM下一个状态

NS ＝ f(PS, I)

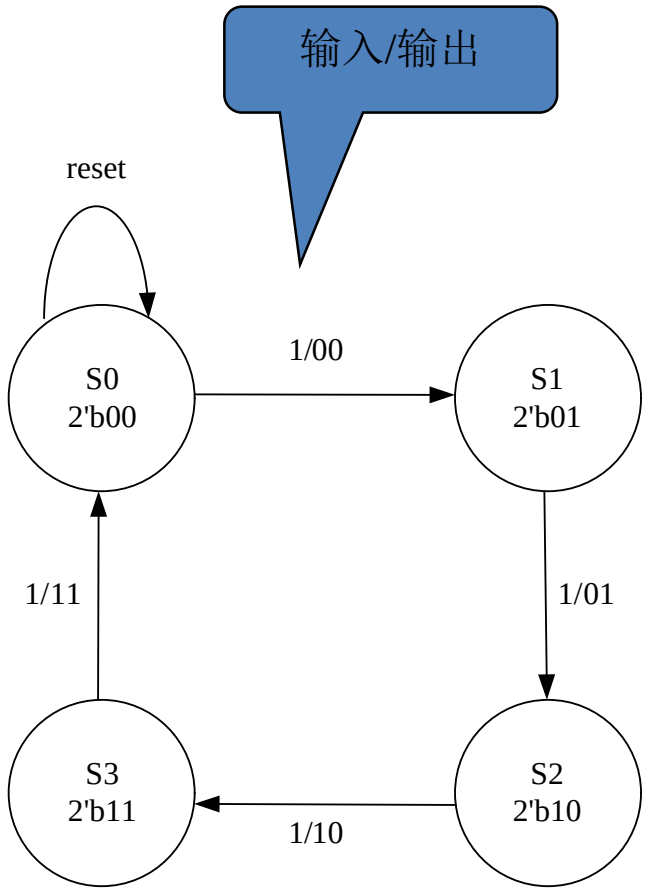
Omoore ＝ h1(PS)　Omealy ＝ h2（PS, I）

PS ＝ { *FSM初始状态S0　异步复位信号有效*；*NS　Clock有效跳变沿*

Moore
型输
出

FSM设计流程

- FSM电路有哪些状态，给出这些状态的编码方式；
- 明确初始复位状态；
- 明确这些状态的转换关系，和转换条件，
- 画出状态转换图
- **RTL编码**



FSM状态的存储方式

- 用什么器件来存储FSM的状态？
  - 时钟沿跳变触发器（Flip－Flop）
  - 电平触发的锁存器（Latch）
- N个状态的FSM，需要多少个Flip－Flop来存储其状态？
  - 最少需 $\log_2 N$个；
  - 最多使用N个
  - 具体由状态编码方式决定

FSM状态编码方式

- 顺序编码
- Gray编码
- One-hot编码
- Johnson编码
- 自定义方式

FSM的状态编码方式不同，实现FSM所需的Flip-Flop数目不同，FSM的状态转换逻辑、输出逻辑的复杂程度差别很大。

FSM的常用编码实例

No	顺序编码	Gray码	Johnson编码	One－Hot编码
0	0000	0000	00000000	0000000000000001
1	0001	0001	00000001	0000000000000010
2	0010	0011	00000011	0000000000000100
3	0011	0010	00000111	0000000000001000
4	0100	0110	00001111	0000000000010000
5	0101	0111	00011111	0000000000100000
6	0110	0101	00111111	0000000001000000
7	0111	0100	01111111	0000000010000000
8	1000	1100	11111111	0000000100000000
9	1001	1101	11111110	0000001000000000
10	1010	1111	11111100	0000010000000000
11	1011	1110	11111000	0000100000000000
12	1100	1010	11110000	0001000000000000
13	1101	1011	11100000	0010000000000000
14	1110	1001	11000000	0100000000000000
15	1111	1000	10000000	1000000000000000

FSM各种状态编码的特点

- FSM的编码方式、决定了FSM所需要的Flip-Flop数目及FSM的输出逻辑、状态转换逻辑的复杂性
- 顺序编码、Gray码使用最少的Flip－Flop；
- One-Hot编码使用最多的Flip－Flop；
- 采用顺序编码、Gray码、Johnson编码，FSM的状态转换逻辑、输出逻辑较复杂，逻辑延时级数较多；
- 采用One-Hot编码，FSM的转态转换逻辑、输出逻辑相对简单，速度快，但Flip－Flop的使用数量最多；

FSM三部分逻辑

- 当前状态寄存逻辑（次态方程或状态转移）

次态方程 $Q_i^{n+1} = h_i(X_1, X_2 \ldots X_n; Q_1^n, Q_2^n \ldots Q_k^n)$ $i=1, 2, \ldots k$ （3）

- 下一状态逻辑（激励方程）

激励方程 $Y_i = g_i(X_1, X_2 \ldots X_n; Q_1^n, Q_2^n \ldots Q_k^n)$ $i=1, 2, \ldots k$ （2）

- 输出逻辑（输出方程）

输出方程 $Z_i = f_i(X_1, X_2 \ldots X_n; Q_1^n, Q_2^n \ldots Q_k^n)$ $i=1, 2, \ldots m$ （1）

下一状态逻辑比较简单时

常用方式：时序逻辑和组合逻辑分开描述

FSM设计描述的HDL编码形式（1）

——当前状态逻辑

设FSM：当前状态为PS、输入信号为I、下一状态为NS，输出为O

FSM时序块（初始状态设置、状态转移）描述形式：

可以采用always @(posedge clk or negedge Reset) 形式

```
always  @(posedge clk or negedge Reset)
   begin
     if  (~Reset)
          PS <= FSM_Initial_State;
     else   PS <= NS;
   end
```

FSM初始状态设置

FSM在时钟控制下的状态转移情形

次态方程 $Q_i^{n+1} = h_i(X_1, X_2 \ldots X_n; Q_1^n, Q_2^n \ldots Q_k^n)$ $i=1, 2, \ldots k$ （3）

FSM设计描述的HDL编码形式（2）

——下一状态逻辑

```
always  @( PS or I )
    begin
        case (PS)
            ST0:
               描述FSM的函数 NS = f(ST0, I);
            ST1：
               描述FSM的函数 NS = f(ST1, I);
          ……
            STn-1：
               描述FSM的函数 NS = f(STn, I);
            default：
               描述FSM在缺省情形下的NS取值情形；
        end
```

也可以在case语句前加一条语句：NS＝ST0；

不可省略！

FSM设计描述的HDL编码形式（3）

——输出逻辑

FSM输出逻辑O的描述形式

```
always  @( PS or I )
    begin
       case (PS)
           ST0:
              描述FSM的函数 O =h(ST0, I);
           ST1：
              描述FSM的函数 O = h(ST1, I);
          ……
           STn-1：
              描述FSM的函数 O =h(STn, I);
           default：
              描述FSM在缺省情形下的NS取值情形;
    end
```

Moore机模型中，输出逻辑只与当前状态PS有关，与输入信号I无关。

always @(PS)

O = h(Sti)

FSM设计描述的HDL编码形式（4）

——下一状态+输出

- FSM的下一状态逻辑、输出逻辑也可以放在同一个always描述块中加以描述，例如：

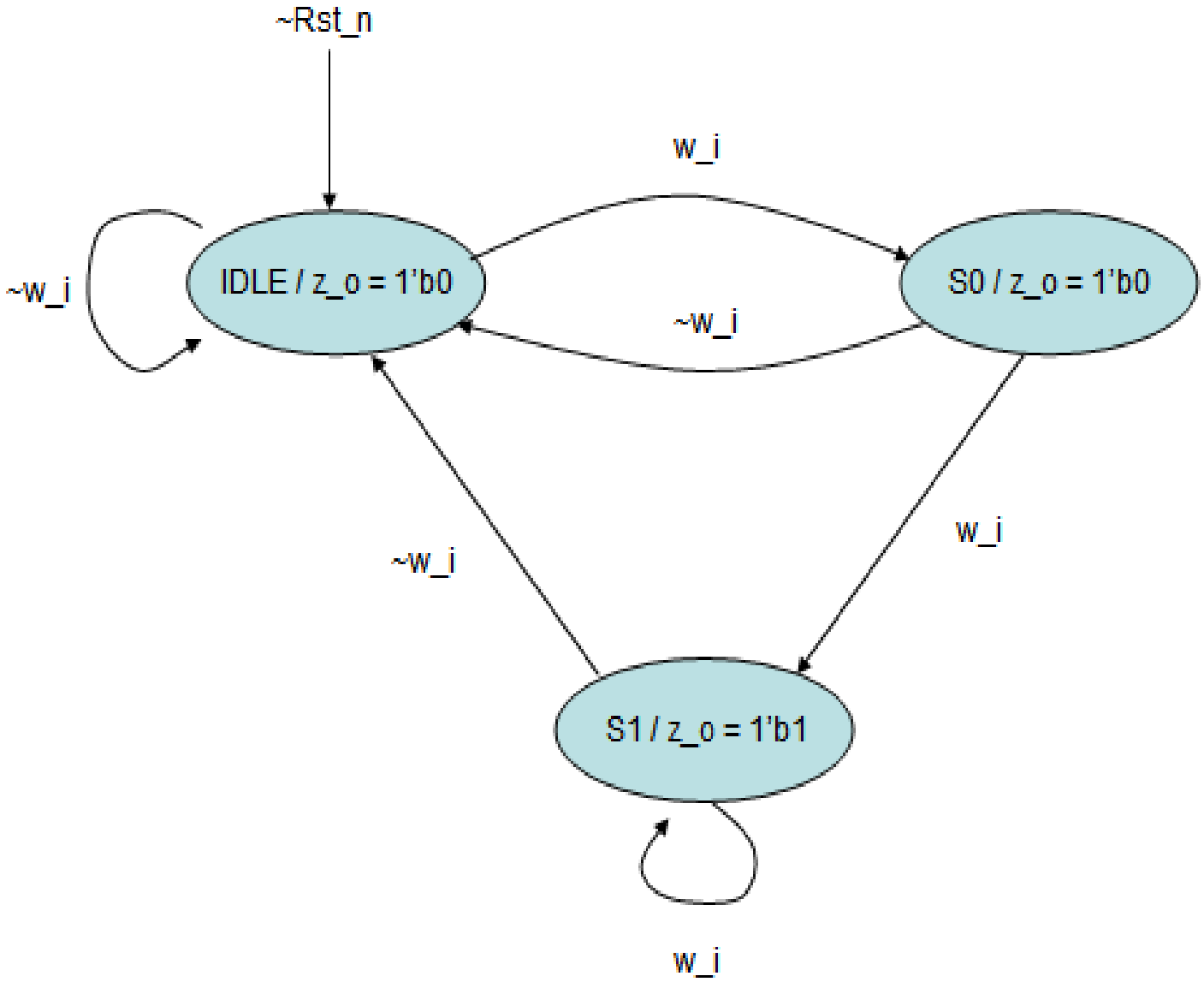
```
always  @( PS or I )
    begin
      case (PS)
          ST0:
            描述FSM的下一状态NS、输出逻辑O;
          ST1：
            描述FSM的下一状态NS、输出逻辑O;
         ……
          STn-1：
            描述FSM的下一状态NS、输出逻辑O;
          default：
             描述FSM在缺省情形下的NS、O的取值情形；
    end
```

```
always  @(posedge clk or negedge
Reset)
   begin
     if  (~Reset)
          PS <= FSM_Initial_State;
     else   PS <= NS;
   end
```

```
always  @( PS or I )
   begin
     case (PS)
       ST0:
         描述FSM的下一状态NS、输出逻辑O;
       ST1：
         描述FSM的下一状态NS、输出逻辑O;
       ……
       STn-1：
         描述FSM的下一状态NS、输出逻辑O;
       default：
          描述FSM在缺省情形下的NS、O的取值情形;
   end
```

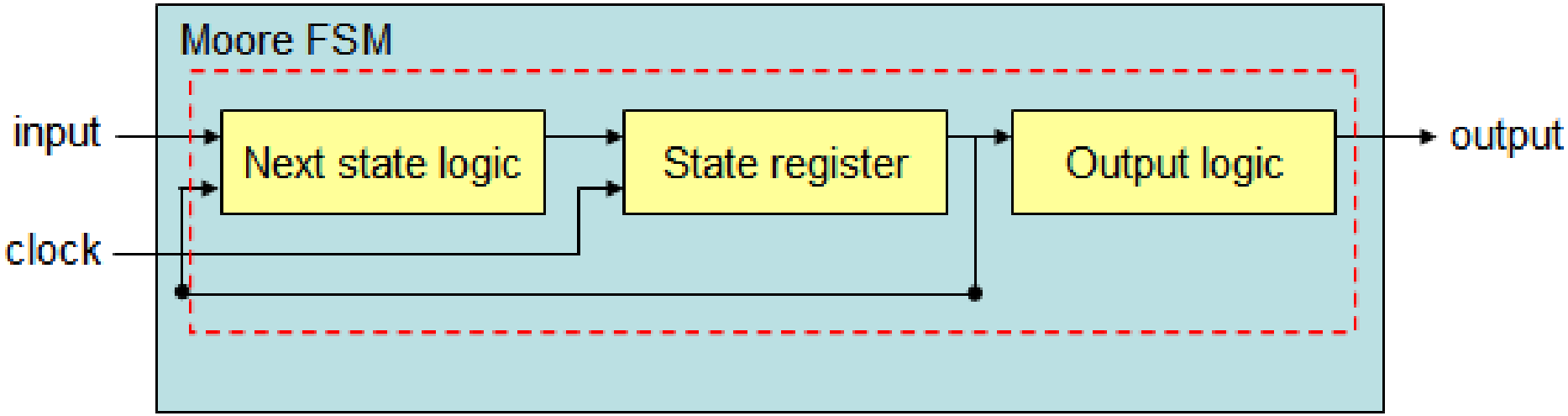
```
always  @( PS or I )
   begin
     case (PS)
       ST0:
         描述FSM的函数 O = f(ST0
       ST1：
         描述FSM的函数 O = f(ST1
       ……
       STn-1：
         描述FSM的函数 O = f(STn
       default：
          描述FSM在缺省情形下的
取值情形;
   end
```

- 有限状态机的描述风格：
  - **One-always风格 — 一段式**
  - **Two-always风格 — 二段式**
  - **Three-always风格 — 三段式**

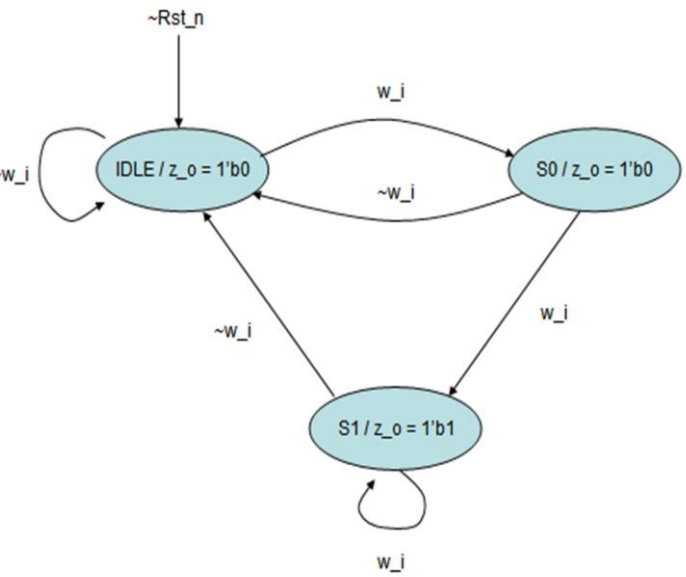

~Rst_n
w_i
~w_i
IDLE / z_o = 1'b0
S0 / z_o = 1'b0
~w_i
~w_i
w_i
S1 / z_o = 1'b1
w_i

一段式

- 一段式的状态机描述方法是指将整个状态机写到一个always 模块里，该模块描述当前状态转移，又描述状态的输入和输出。


Moore FSM
input
clock
Next state logic
State register
Output logic
output

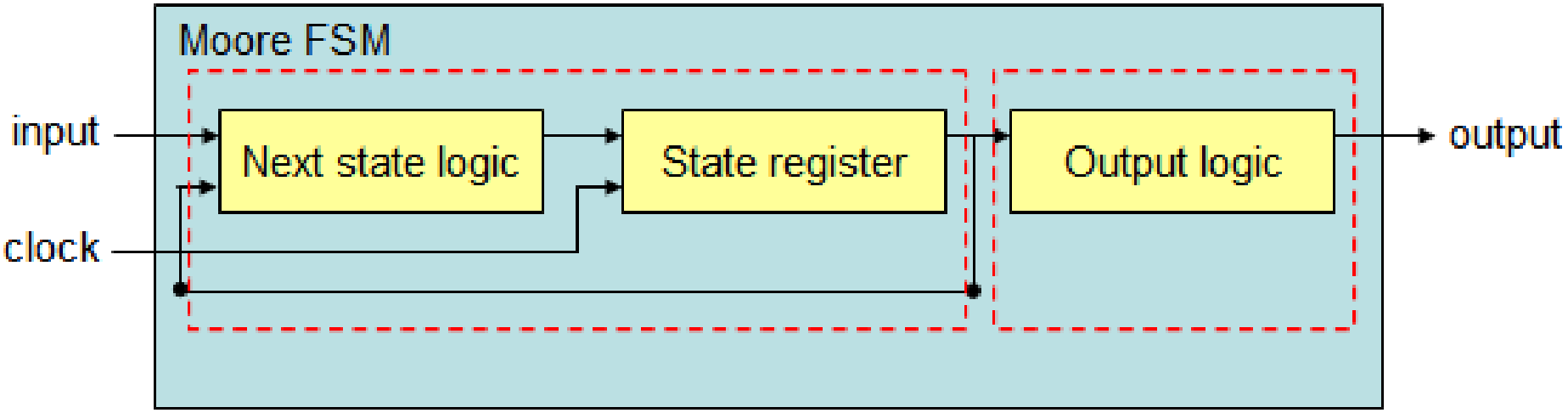
```
1 /*
2 (C) OOMusou 2011 http://oomusou.cnblogs.com
3
4 Filename    : simple_fsm_moore_1_always_ng.v
5 Synthesizer : Quartus II 8.1
6 Description : 1 always block for moore fsm
7 Release     : Jun.05,2011 1.0
8 */
9
10 module simple_fsm (
11   clk,
12   rst_n,
13   w_i,
14   z_o
15 );
16
```

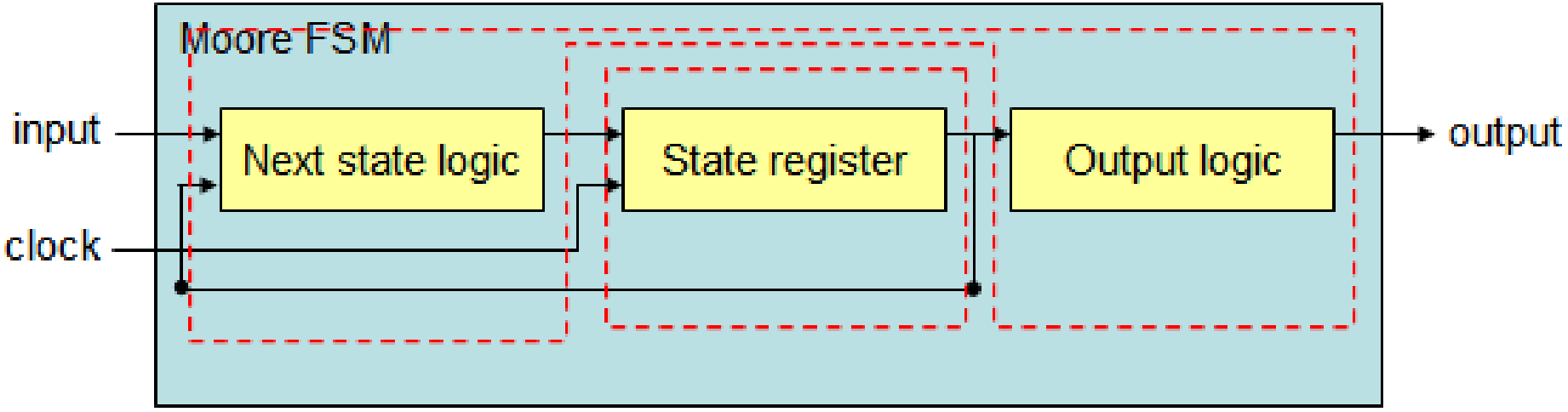



```
17 input  clk;
18 input  rst_n;
19 input  w_i;
20 output z_o;
21
22 parameter IDLE = 2'b00;
23 parameter S0   = 2'b01;
24 parameter S1   = 2'b10;
25
26 reg [1:0] curr_state;
27 reg z_o;
28
29 always@(posedge clk or negedge rst_n)
30   if (~rst_n) {curr_state, z_o} <= {IDLE, 1'b0};
31   else
32     case (curr_state)
33       IDLE : if (w_i) {curr_state, z_o} <= {S0,   1'b0};
34              else     {curr_state, z_o} <= {IDLE, 1'b0};
35       S0   : if (w_i) {curr_state, z_o} <= {S1,   1'b1};
36              else     {curr_state, z_o} <= {IDLE, 1'b0};
37       S1   : if (w_i) {curr_state, z_o} <= {S1,   1'b1};
38              else     {curr_state, z_o} <= {IDLE, 1'b0};
39       default :       {curr_state, z_o} <= {IDLE, 1'b0};
40     endcase
41
42 endmodule
```

二段式

- 二段式的状态机描述方法是指使用两个always模块
- 一个always模块采用同步时序的方式描述当前状态和下一状态的转移,
- 一个always模块采用组合逻辑来描述下一状态和输出向量的赋值.


Moore FSM
input
clock
Next state logic
State register
Output logic
output


Moore FSM
input
clock
Next state logic
State register
Output logic
output

```
28 
29 // state reg + next state logic
30 always@(posedge clk or negedge rst_n)
31    if (~rst_n) curr_state <= IDLE;
32    else
33       case (curr_state)
34          IDLE    : if (w_i) curr_state <= S0;
35                    else     curr_state <= IDLE;
36          S0      : if (w_i) curr_state <= S1;
37                    else     curr_state <= IDLE;
38          S1      : if (w_i) curr_state <= S1;
39                    else     curr_state <= IDLE;
40          default :          curr_state <= IDLE;
41       endcase
42     
```

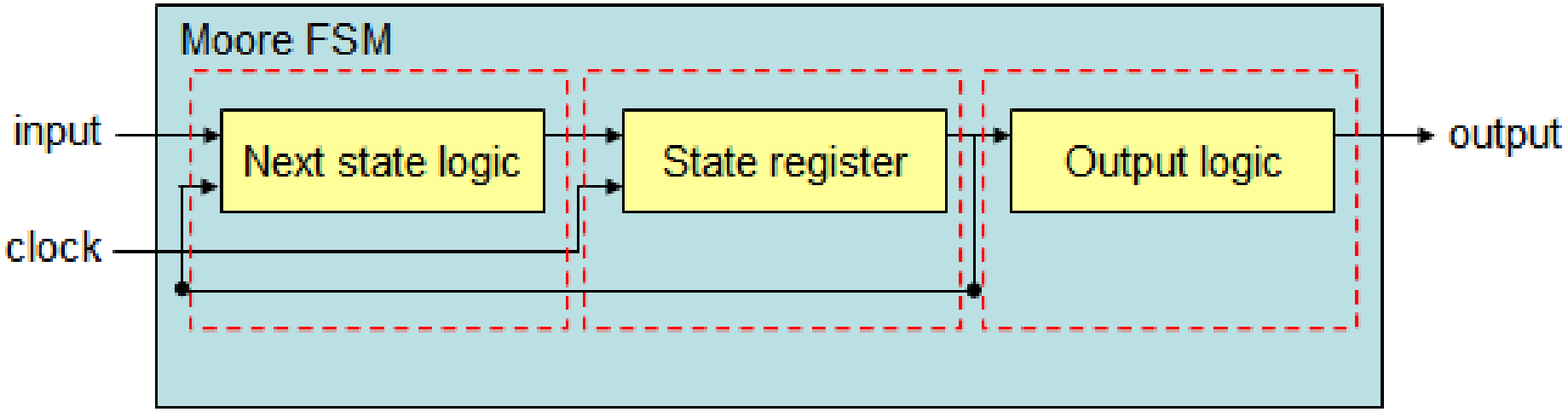
```
43 // output logic
44 always@(*)
45    case (curr_state)
46       IDLE    : z_o = 1'b0;
47       S0      : z_o = 1'b0;
48       S1      : z_o = 1'b1;
49       default : z_o = 1'b0;
50    endcase
51
```

```
// state reg
always@(posedge clk or negedge rst_n)
  if (~rst_n) curr_state <= IDLE;
  else        curr_state <= next_state;
    
// next state logic
always@(*)
  case (curr_state)
    IDLE : if (w_i) next_state = S0;
           else     next_state = IDLE;
    S0   : if (w_i) next_state = S1;
           else     next_state = IDLE;
    S1   : if (w_i) next_state = S1;
           else     next_state = IDLE;
    default :       next_state = IDLE;
  endcase
```

```
28
29 // state reg + next st
30 always@(posedge clk or
31   if (~rst_n) curr_sta
32   else
33     case (curr_state)
34       IDLE : if (w_i
35              else
36       S0   : if (w_i
37              else
38       S1   : if (w_i
39              else
40       default :
41     endcase
42
```

三段式

- 三段式的状态机描述方法是由二段式发展而来的,在二段式两个always 的基础上又增加了一个always 块来描述每个状态的输出。


Moore FSM
input
clock
Next state logic
State register
Output logic
output

```
// state reg
always@(posedge clk or negedge rst_n)
  if (~rst_n) curr_state <= IDLE;
  else        curr_state <= next_state;

// next state logic
always@(*)
  case (curr_state)
    IDLE : if (w_i) next_state = S0;
                else next_state = IDLE;
    S0   : if (w_i) next_state = S1;
                else next_state = IDLE;
    S1   : if (w_i) next_state = S1;
                else next_state = IDLE;
    default :       next_state = IDLE;
  endcase
```

```
// output logic
always@(*)
  case (curr_state)
    IDLE : z_o = 1'b0;
    S0   : z_o = 1'b0;
    S1   : z_o = 1'b1;
    default : z_o = 1'b0;
  endcase
```

使用3个always写法的优点

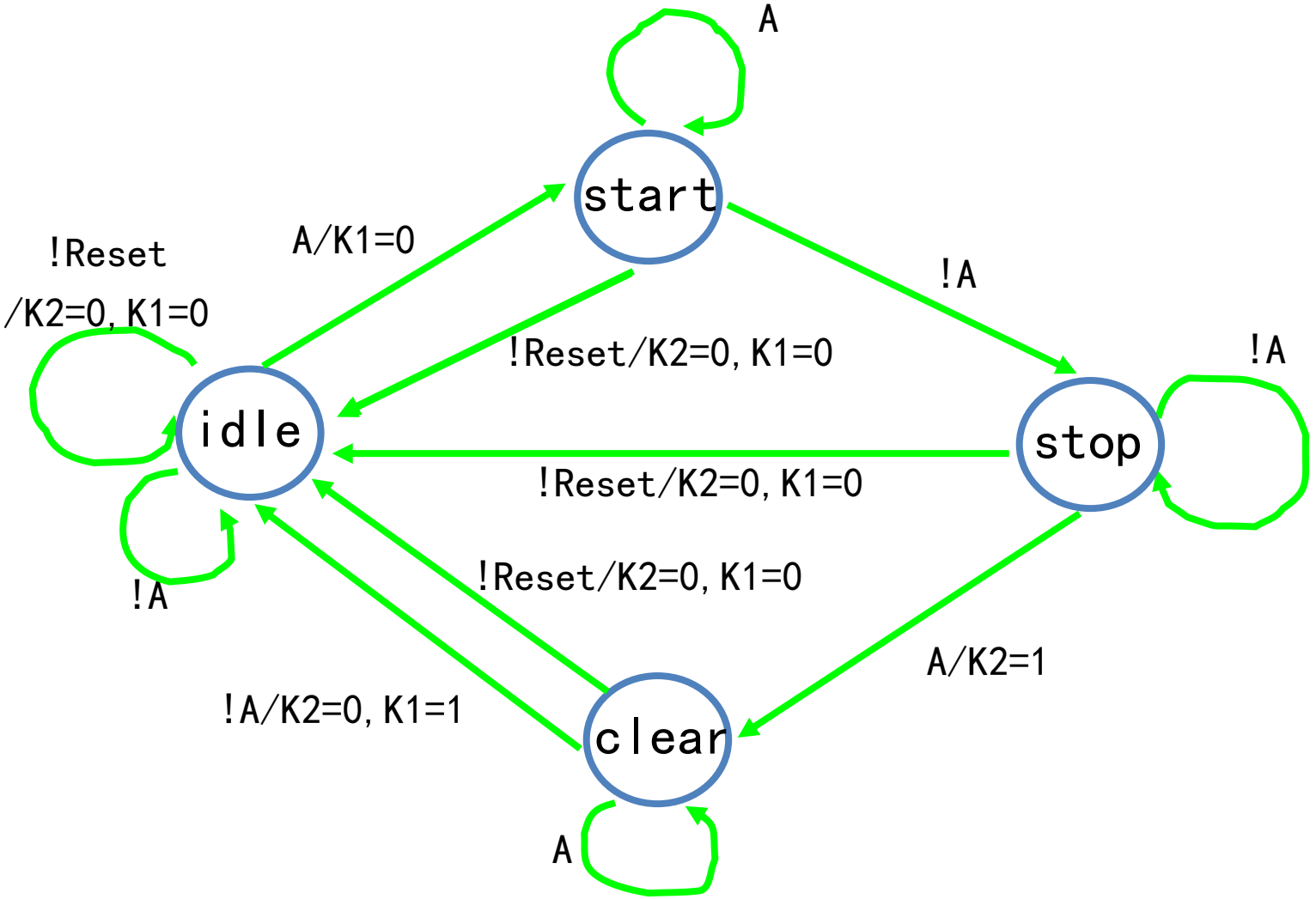
- 1.可忠实地反映出原本的Moore FSM硬件架构
- 2.可轻易地将state diagram改用Verilog表示
- 3.将Next state logic与output logic分开，可降低code的复杂，便于日后维护；

● 同步状态机的实现

状态分配：

- 二进制码表示状态的状态机
- 格雷（Gray）码表示状态的状态机
- 独热（One-hot）码表示状态的状态

设计举例


A
start
!Reset
/K2=0, K1=0
A/K1=0
!A
!Reset/K2=0, K1=0
!A
idle
stop
!Reset/K2=0, K1=0
!A
!Reset/K2=0, K1=0
A/K2=1
!A/K2=0, K1=1
clear
A

用可综合Verilog模块设计状态机的典型方法

```
module fsm(clock,reset,a,k2,k1);
    input clock,reset,a;output k2,k1;reg k2,k1;reg[1:0] state;

    parameter idle=2'b00,start=2'b01,stop=2'b10,clear=2'b11;

    always @(posedge clock)
    if(!reset)
        begin
          state<=idle;k2<=0;k1<=0;
        end
    else
```

```
case(state)
    idle:begin
            if(a)
                    begin
                            state<=start;
                            k1<=0;
                    end
                    else
                            state<=idle;
          end
    start:begin
            if(!a)  state<=stop;
            else    state<=start;
          end
```

```
stop:begin
            if(a)
            begin
                        state<=clear;
                        k2<=1;
            end
            else      state<=stop;
        end
clear:begin
            if(!a)
            begin
                        state<=idle;
                        k2<=0;k1<=1;
            end
            else      state<=clear;
         end
    endcase
endmodule
```

用可综合Verilog模块设计、用独热码表示状态的状态机

```
module fsm(clock,reset,a,k2,k1);

input clock,reset,a;output k2,k1;reg k2,k1;reg[3:0] state;

parameter
    idle=4'b1000,start=4'b0100,stop=4'b0010,clear=4'b0001;
     always @(posedge clock)
          if(!reset)
          begin
            state<=idle;k2<=0;k1<=0;
          end
     else
```

```
case(state)
    idle:begin
            if(a)
            begin
                    state<=start;
                    k1<=0;
            end
            else
                    state<=idle;
           end
    start:begin
            if(!a)  state<=stop;
            else    state<=start;
          end
```

```
stop:begin
        if(a)
        begin
                state<=clear;
                k2<=1;
        end
        else    state<=stop;
      end
   clear:begin
        if(!a)
        begin
                state<=idle;
                k2<=0;k1<=1;
        end
        else    state<=clear;
       end
   default:state<=idle;
   endcase
endmodule
```

用可综合Verilog模块设计的多输出状态机时常用的方法

```
module fsm(clock,reset,a,k2,k1);
input clock,reset,a;
output k2,k1;
reg k2,k1;
reg[1:0] state;

parameter idle=2'b00,start=2'b01,stop=2'b10,clear=2'b11;
      always @(posedge clock)
      if(!reset)
      begin
           state<=idle;
      end
else
```

```
case(state)                                    //状态机
    idle:begin
            if(a)
            begin
                        state<=start;
            end
            else
                        state<=idle;
        end
    start:begin
            if(!a)  state<=stop;
            else    state<=start;
         end
stop:begin
            if(a)
            begin
                        state<=clear;
            end
            else     state<=stop;
        end
```

```
clear:begin
          if(!a)
          begin
                    state<=idle;
          end
          else      state<=clear;
        end
    default:state<=2'bxx;
    endcase
always @(state or reset or a)    //组合逻辑
      if(!reset) k2=0;
      else if((state==clear)&&a)
          k2=1;
      else k2=0;
always @(state or reset)         //组合逻辑
      if(!reset) k1=0;
      else if((state==idle)&&!a)
          k1=1;
      else k1=0;
endmodule
```

状态/输出图

状态/输出表

S	S*/C
S0	S1 /0
S1	S2 /0
S2	S3 /0
S3	S4 /0
S4	S0 /1

实现格雷码、独热码（一段、二段、三段）

- https://blog.csdn.net/chenchen410/article/details/38759381