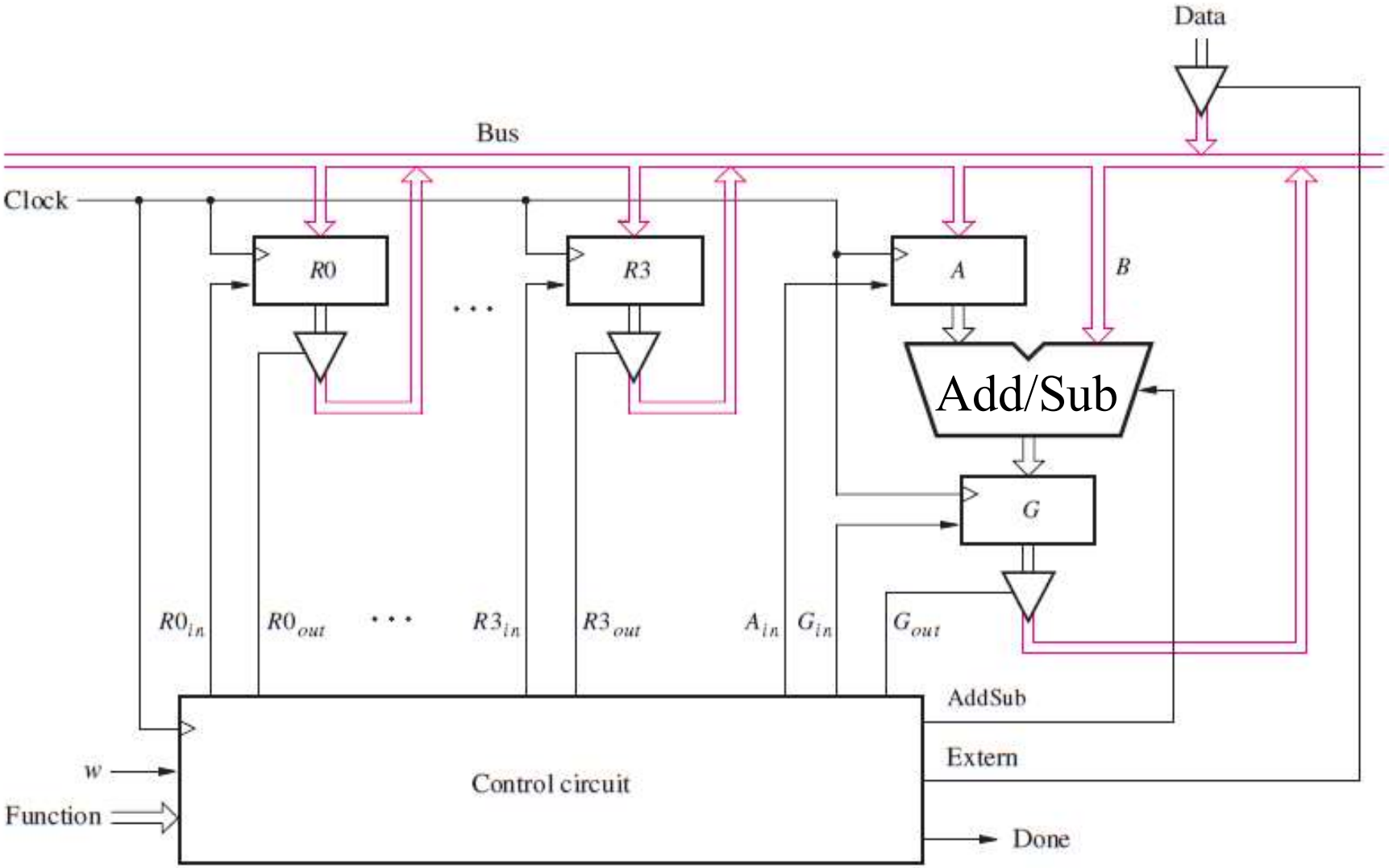
Chapter 6

同步时序电路

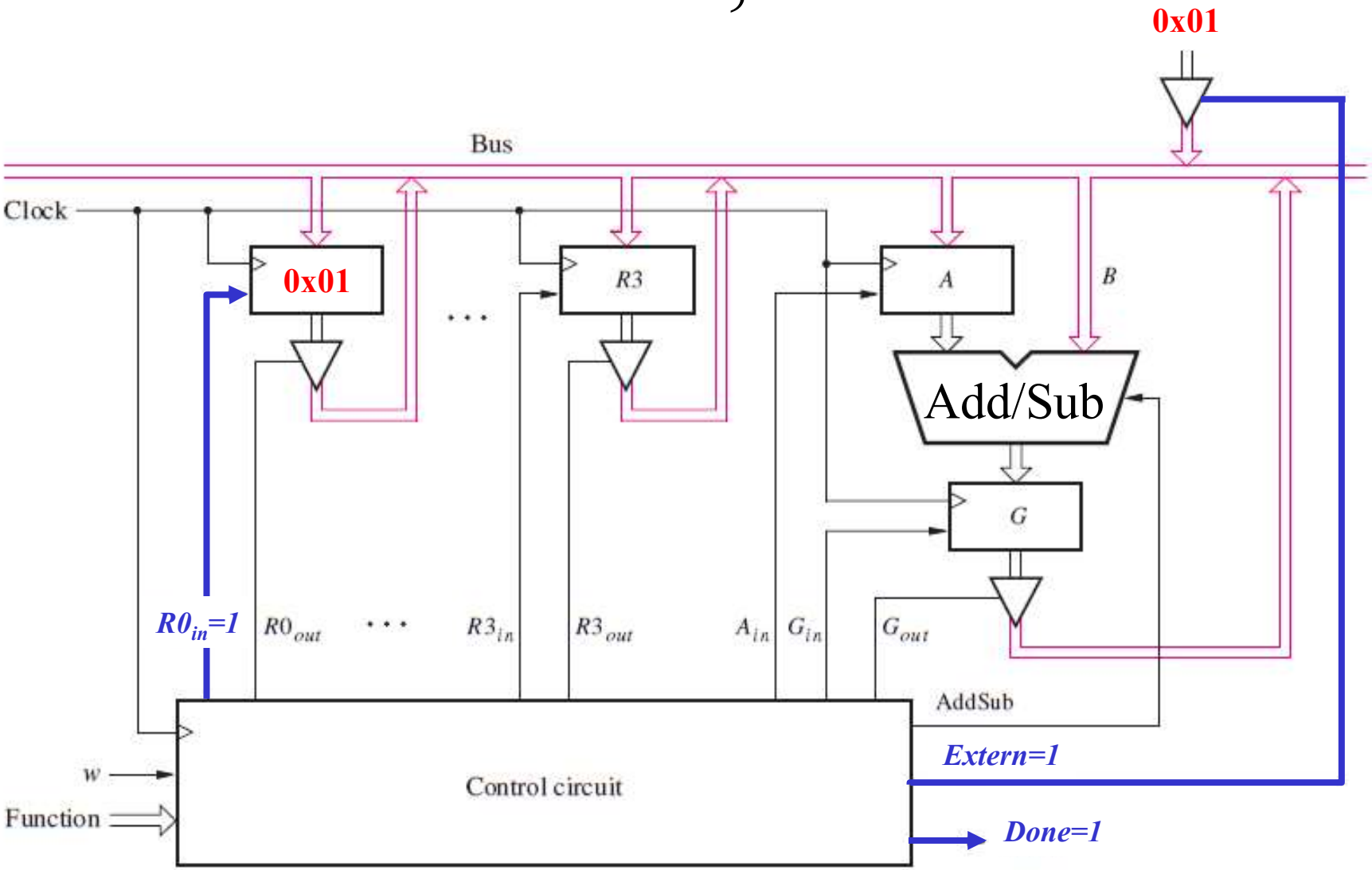
一个简单CPU的指令集

	Operation	Function Performed
单周期	Load *Rx, Data*	*Rx* ← *Data*
单周期	Move *Rx, Ry*	*Rx* ← *[Ry]*
多周期	Add *Rx, Ry*	*Rx* ← *[Rx]+[Ry]*
多周期	Sub *Rx, Ry*	*Rx* ← *[Rx]-[Ry]*

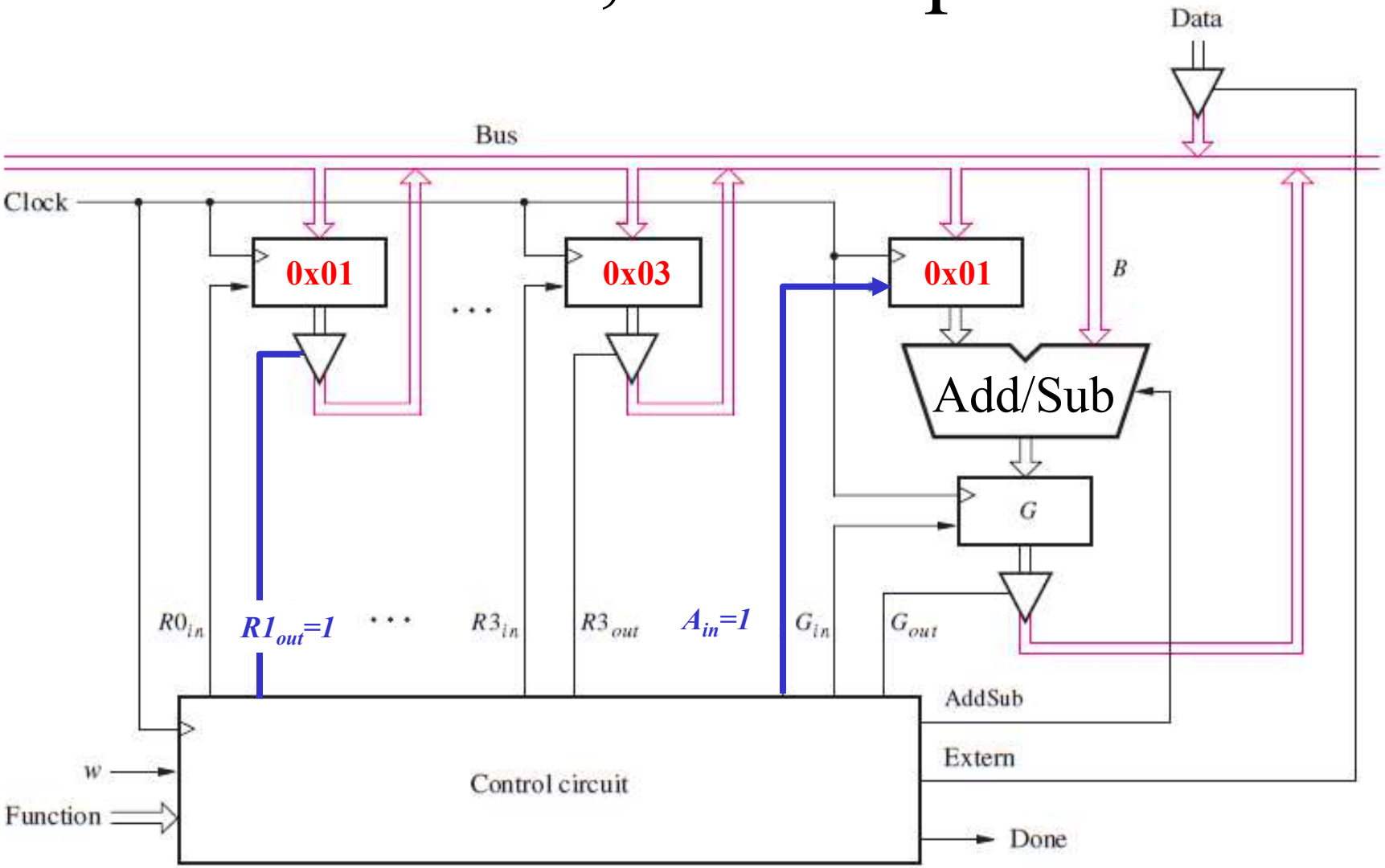
一个简单CPU


Data
Bus
Clock
R0
R3
A
B
Add/Sub
G
$R0_{in}$
$R0_{out}$
$R3_{in}$
$R3_{out}$
A_{in}
G_{in}
G_{out}
AddSub
Extern
w
Function
Control circuit
Done

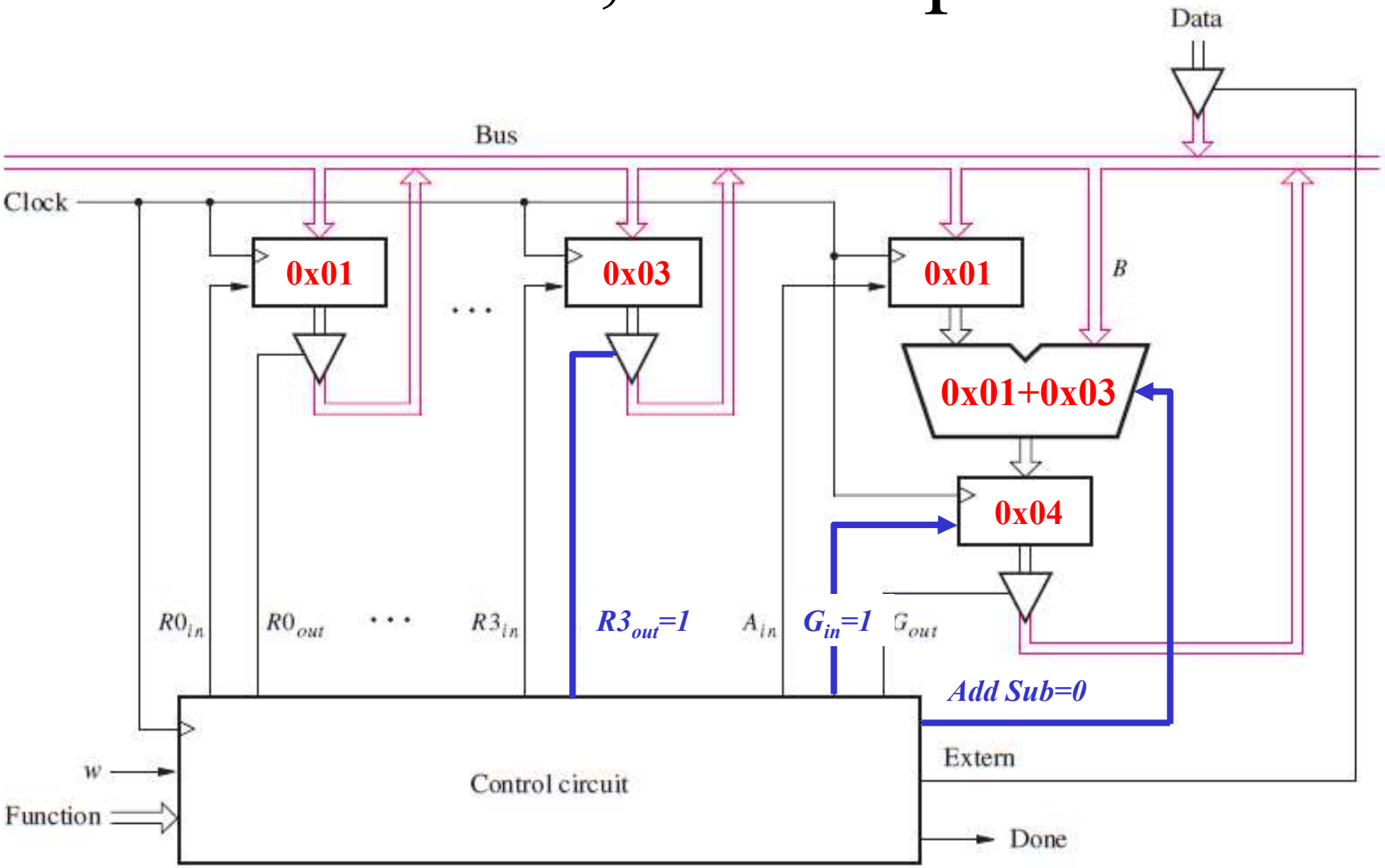
Load R0, Data


0x01
Bus
Clock
0x01
R3
A
B
Add/Sub
G
R0in=1
R0out
R3in
R3out
Ain
Gin
Gout
AddSub
Extern=1
w
Function
Control circuit
Done=1

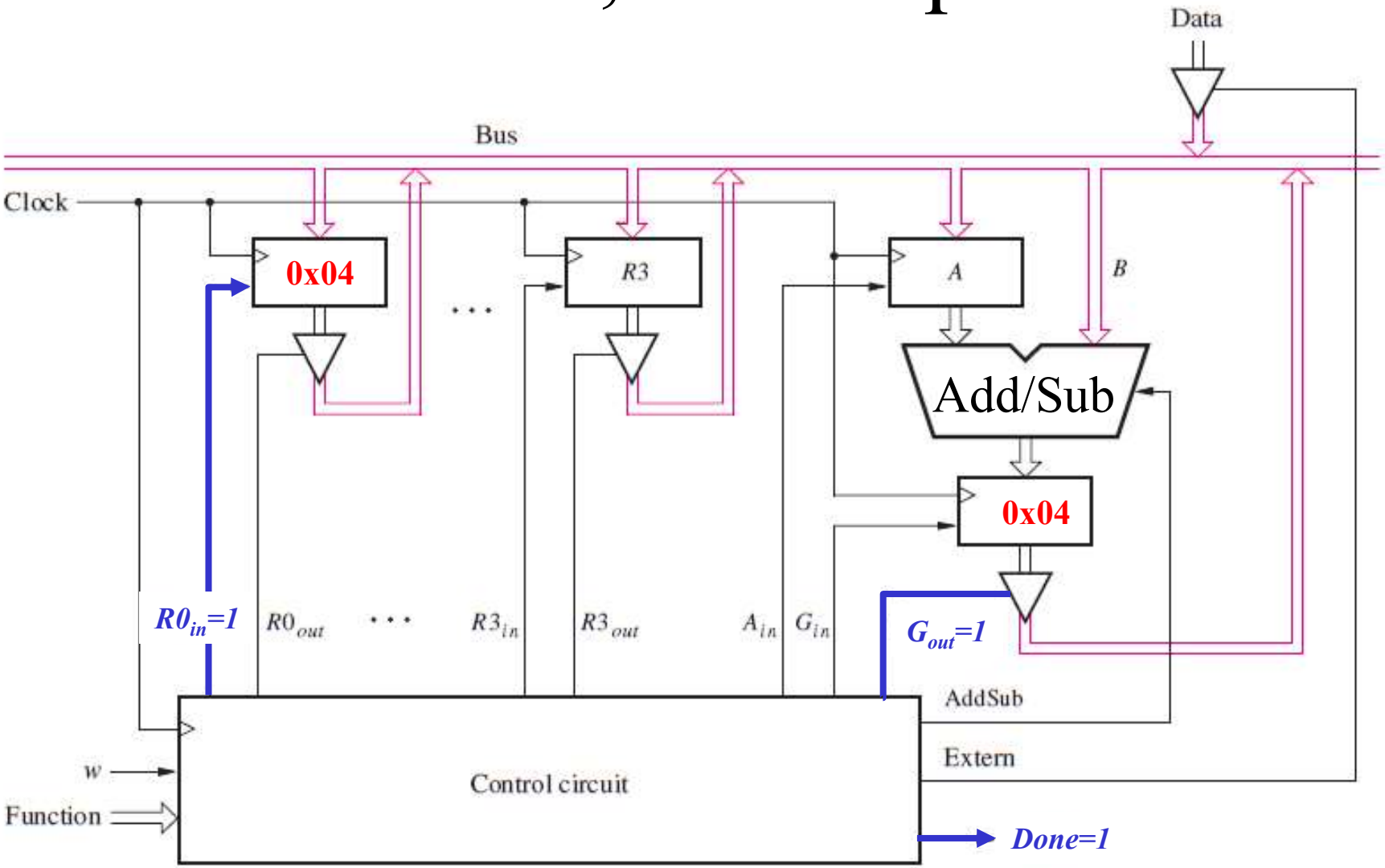
ADD R0, R3: Step1



ADD R0, R3: Step2



ADD R0, R3: Step3



控制逻辑

	T1	T2	T3
(Load):I_0	*Extern, R_{in}=X, Done*		
(Move):I_1	*R_{in}=X, R_{out}=Y, Done*		
(Add):I_2	*R_{out}=X, A_{in}*	*R_{out}=Y, G_{in}, AddSub=0*	*G_{out}, R_{in}=X, Done*
(Sub):I_3	*R_{out}=X, A_{in}*	*R_{out}=Y, G_{in}, AddSub=1*	*G_{out}, R_{in}=X, Done*

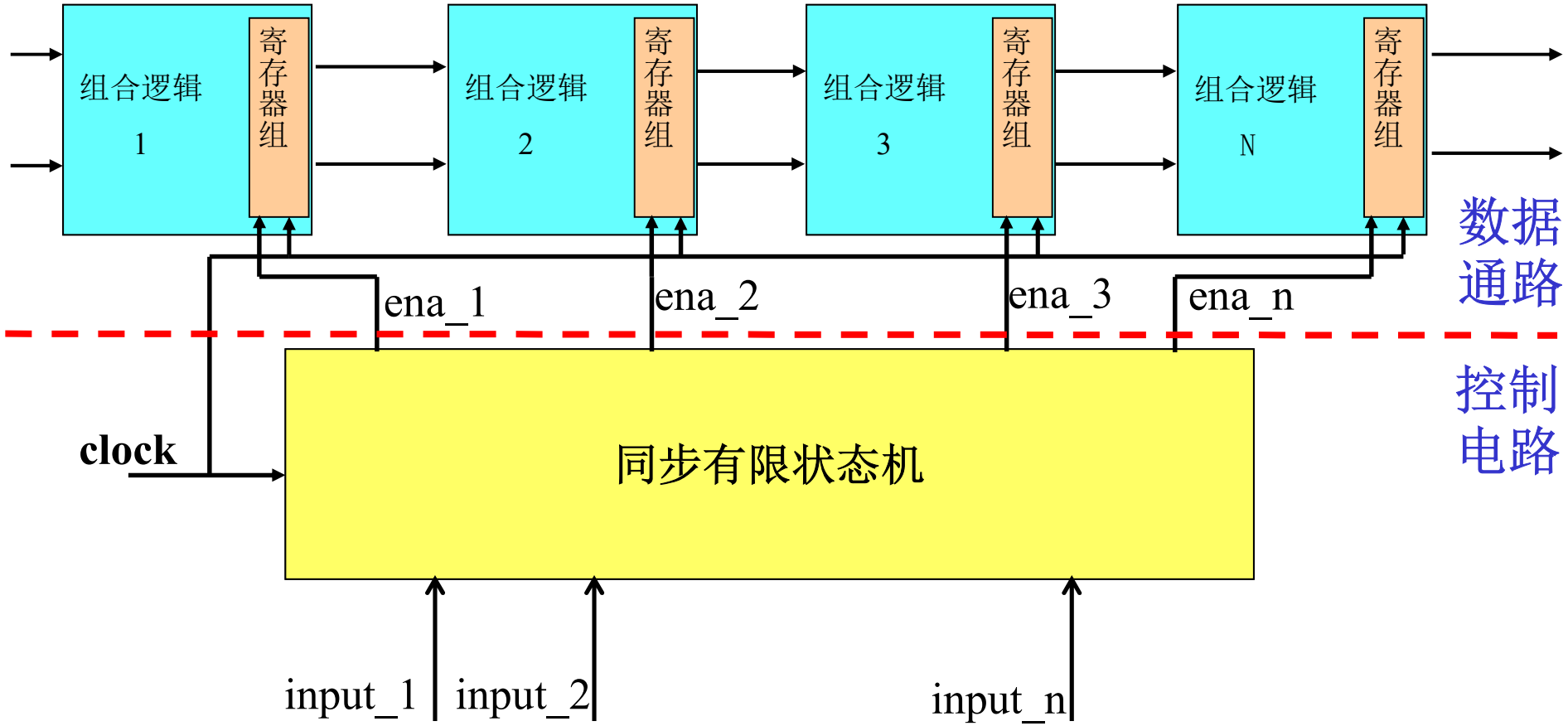
复杂数字系统控制的关键：
生成与时钟精确配合的开关时序

有限状态机的需求

- 如果我们能设计这样一个电路：
  - 能记住自己目前所处的状态；
  - 状态的变化只可能在同一个时钟的跳变沿发生，而不可能发生在任意时刻；
  - 在时钟沿，如输入条件满足，则进入下一状态，否则仍保留在原来的状态；
  - 在进入不同的状态时刻，对系统的开关/功能选择阵列做置“1”或置“0”的操作。

能达到以上要求的电路就是同步有限状态机

复杂数字系统的结构示意图



有限状态机是复杂数字系统控制的核心！

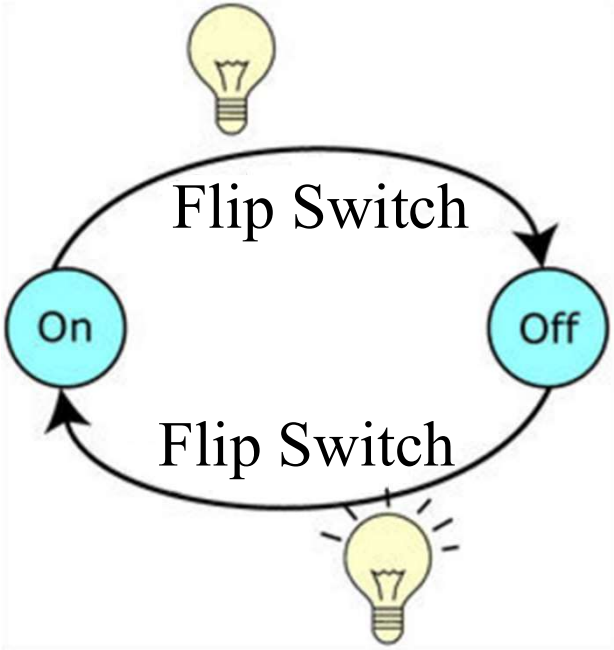
目录

有限状态机

- 时序电路的状态是一个状态变量的集合，这些状态变量在任意时刻的值都包含了为确定电路的未来行为而必须考虑的所有历史信息。
- **状态转移**：系统从一个状态（现态）变成另一个状态（次态）
  - 转移条件：事件驱动
  - 动作： 状态转移过程中，有时需要执行一些操作
  - 状态变量：区分不同状态，位宽有限，状态数有限

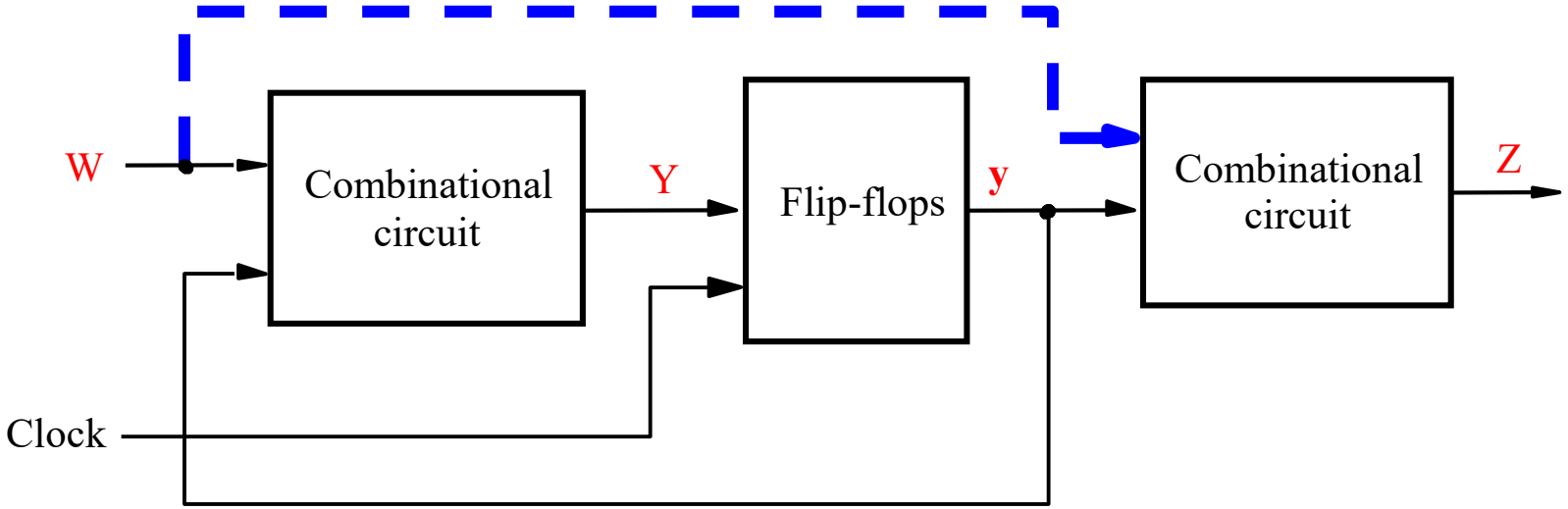
有限状态机

- Finite-state machine, FSM
- 是一种用来进行对象行为建模的工具
- 在计算机科学中，被广泛用于建模应用行为、硬件电路系统设计、软件工程、操作系统、编译器、网络协议、和计算与语言的研究。

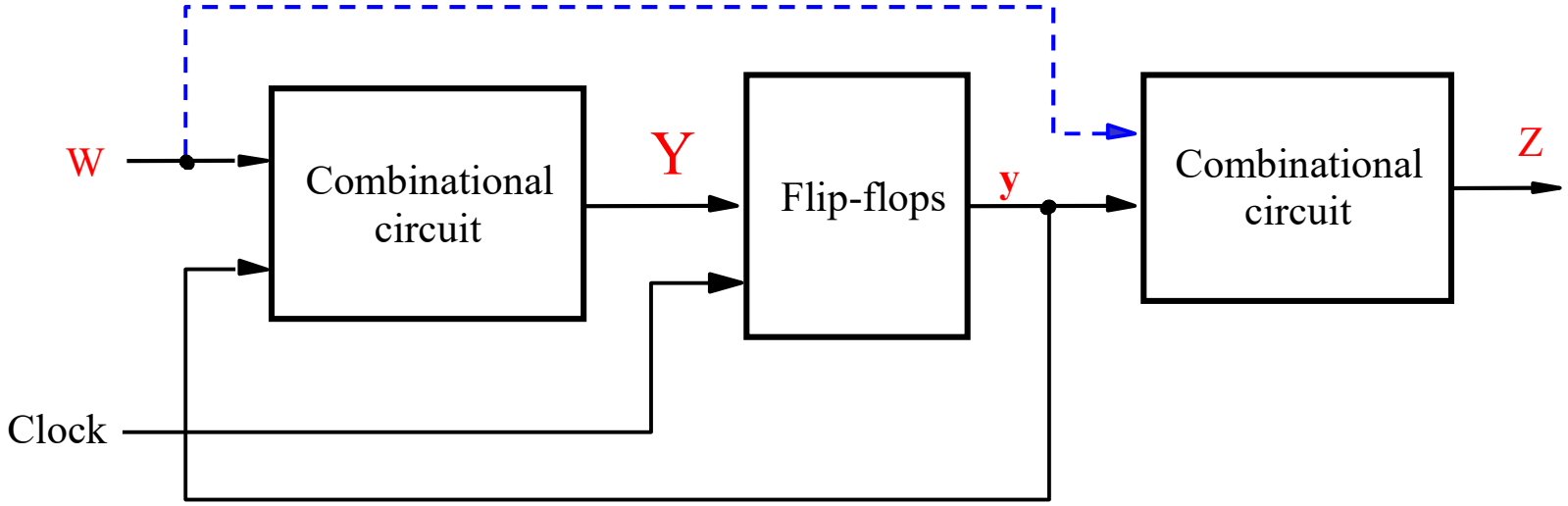


状态机分类

- Moore： 输出仅由电路状态决定
  - 输出可以看成状态的属性
- Mealy：输出由电路状态和输入共同决定
  - 输出可以看成状态转移过程的动作



时序电路模型



逻辑关系：

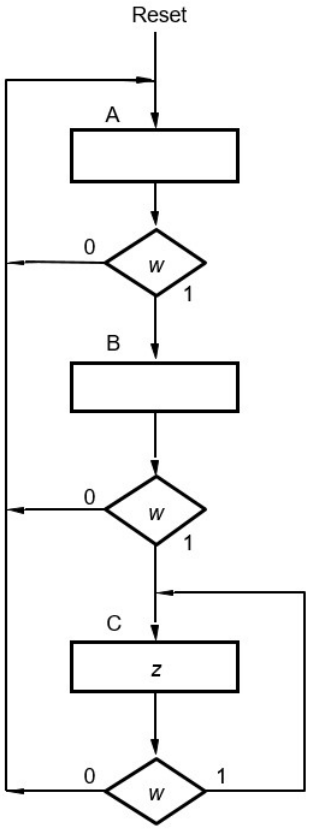
$$\begin{cases} Z = f(w_1, w_2, \cdots, w_n, y_1^n, y_2^n, \cdots, y_j^n) & \text{输出方程} \\ Y = g(w_1, w_2, \cdots, w_n, y_1^n, y_2^n, \cdots, y_j^n) & \text{驱动方程} \\ y = h(Y_1, Y_2, \cdots, Y_k, y_1^n, y_2^n, \cdots, y_j^n) & \text{状态方程} \end{cases}$$

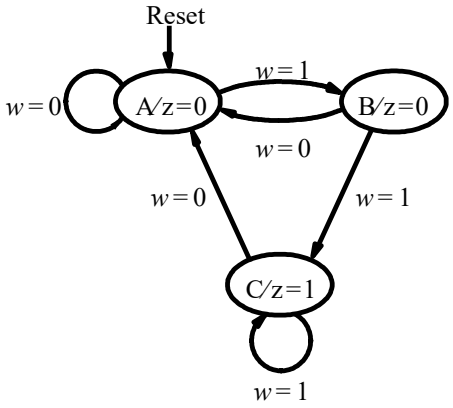
目录

状态机设计方法

- 状态机描述手段
  - 状态图
  - 状态表
  - 直接用HDL语言
  - ASM图

```
module simple (Clock, Resetn, w, z);
    input Clock, Resetn, w;
    output z;
    reg [2:1] y;
    parameter [2:1] A = 2'b00, B = 2'b01, C = 2'b10;

    always @(negedge Resetn, posedge Clock)
        if (Resetn = = 0)  y <= A;
        else
            case (y)
                A: if (w)  y <= B;   else     y <= A;
                B: if (w)  y <= C;   else     y <= A;
                C: if (w)  y <= C;   else     y <= A;
                default:   y <= A;
            endcase

    assign z = (y = = C);
endmodule
```

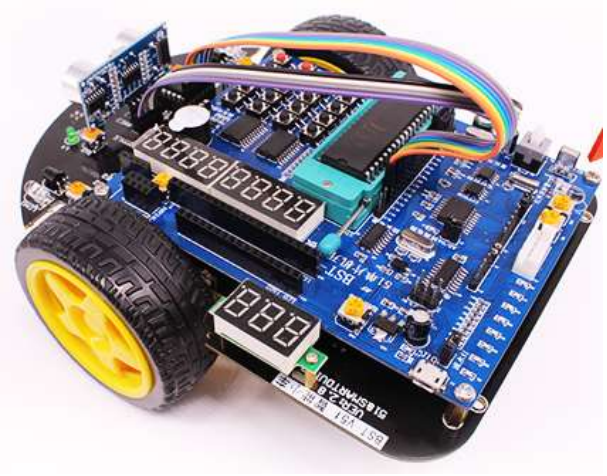




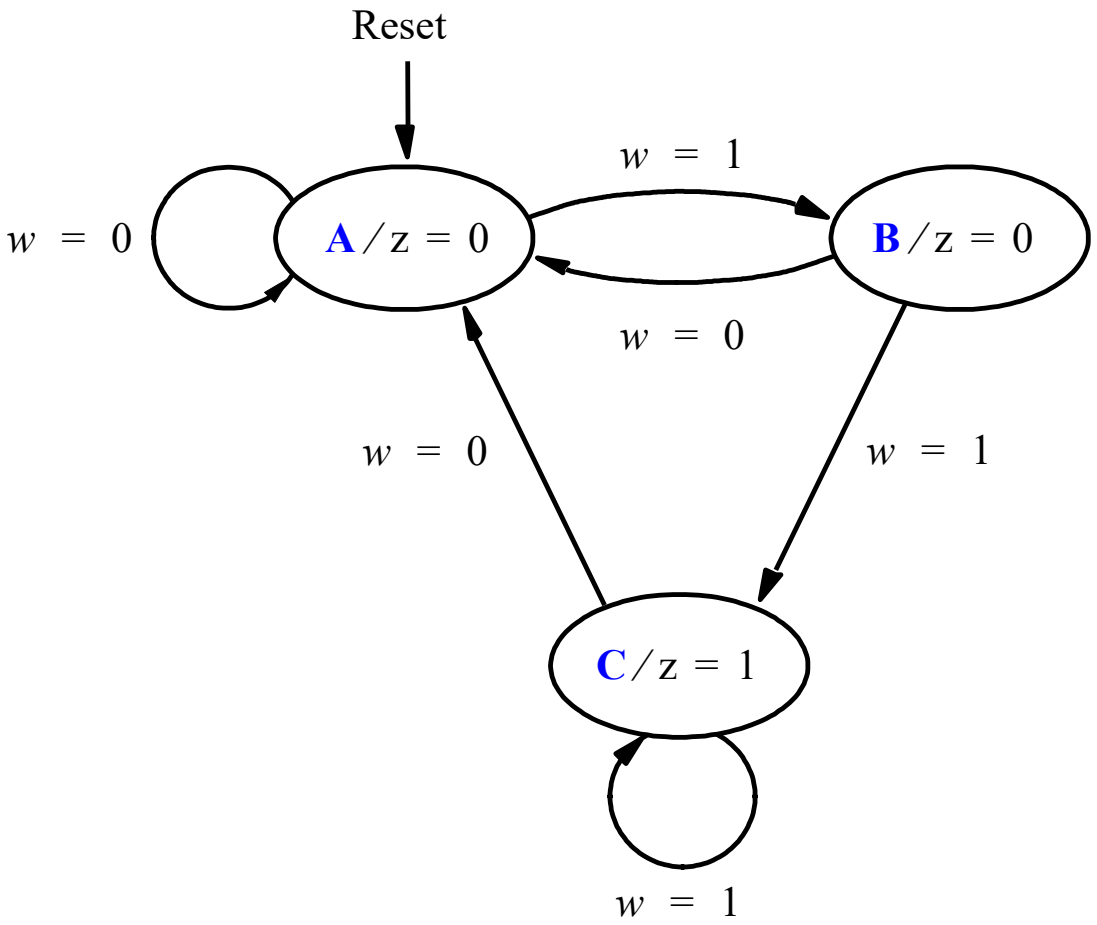
	Present state	Next state		Output
		$w = 0$	$w = 1$	
	y_2y_1	Y_2Y_1	Y_2Y_1	z
A	00	00	01	0
B	01	00	10	0
C	10	00	10	1
	11	*dd*	*dd*	*d*

例1：序列检测器

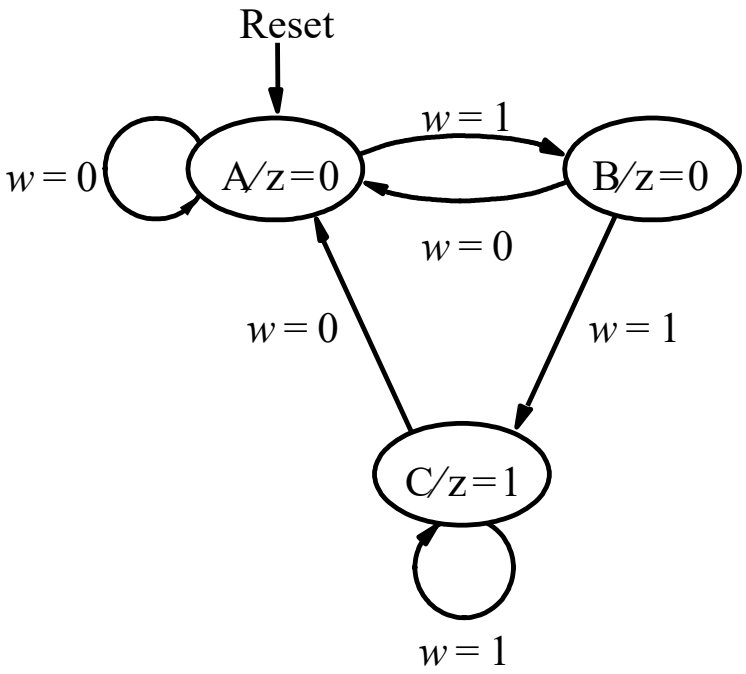
- 功能： 检测特定顺序的数据序列
- 应用举例：自动驾驶汽车的超速检测
  - w：输入信号，1 超速、0 未超速
  - z：输出信号，1 减速、0 保持当前速度
  - 连续输入两个1，则输出z=1 让车辆减速



Moore型状态转移图



状态表



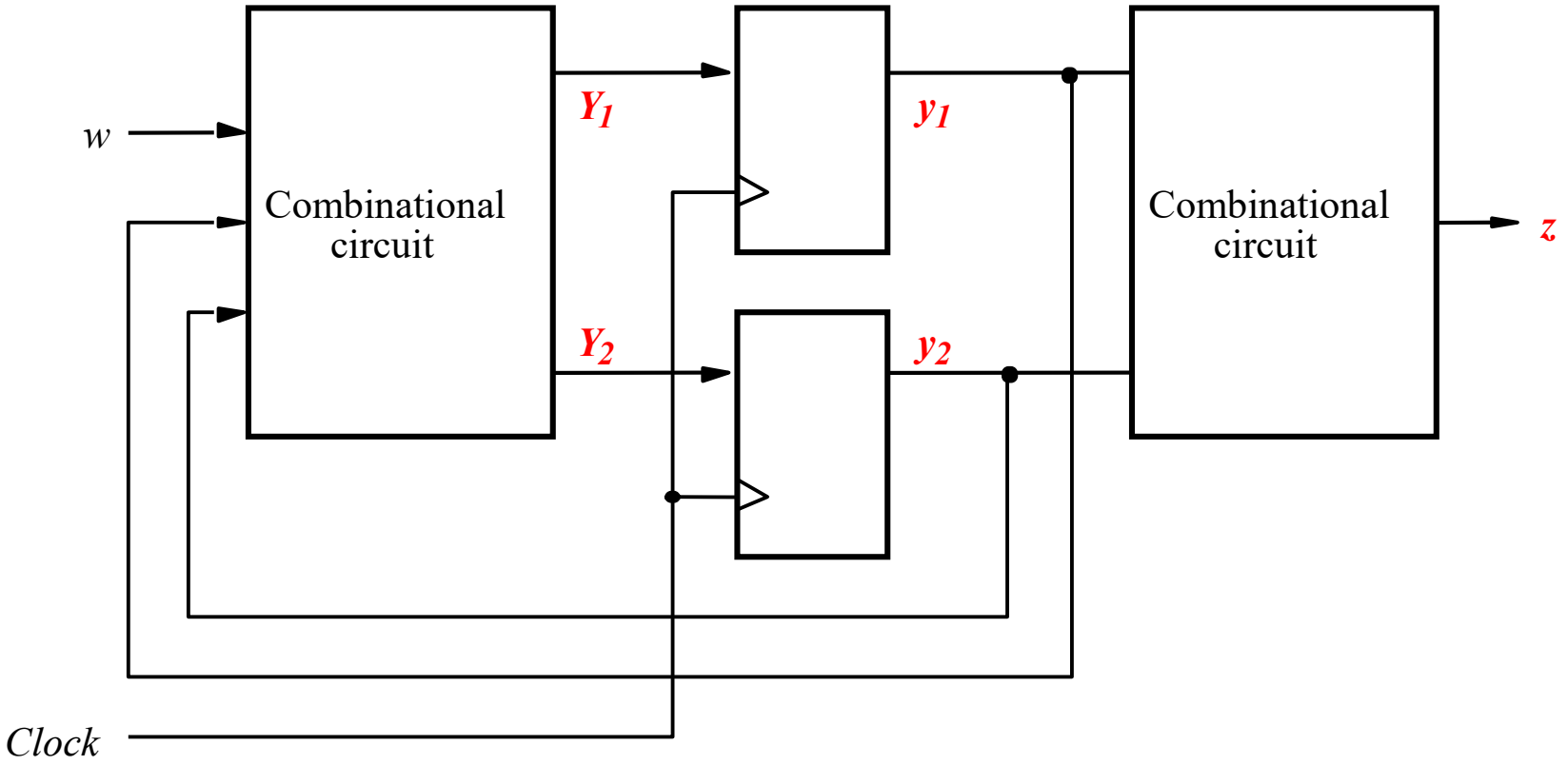
Present state	Next state		Output
	$w = 0$	$w = 1$	z
A	A	B	0
B	A	C	0
C	A	C	1

Figure 6.4. State table for the sequential circuit in Figure 6.3.

状态分配表

	Present state y_2y_1	Next state $w = 0$ Y_2Y_1	Next state $w = 1$ Y_2Y_1	Output z
A	00	00	01	0
B	01	00	10	0
C	10	00	10	1
	11	*dd*	*dd*	*d*

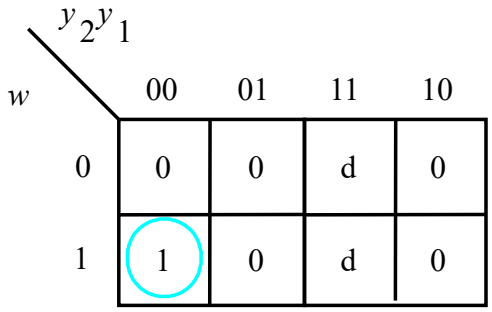
Figure 6.6. State-assigned table for the sequential circuit in Figure 6.4.

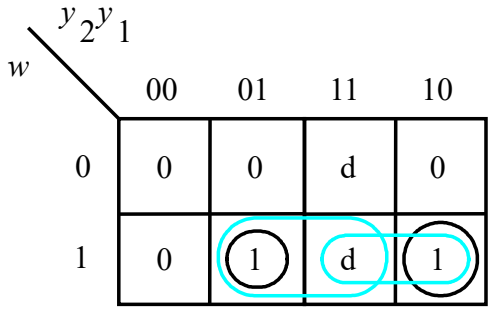


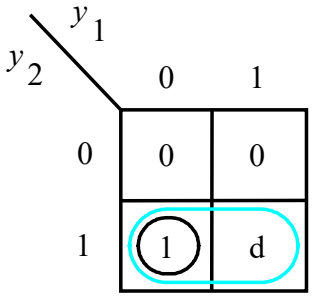
y_1和y_2是现态变量，决定了当前时刻的状态。

Y_1 和 Y_2是次态变量，决定下一个时钟触发边沿后的状态。

逻辑函数推导

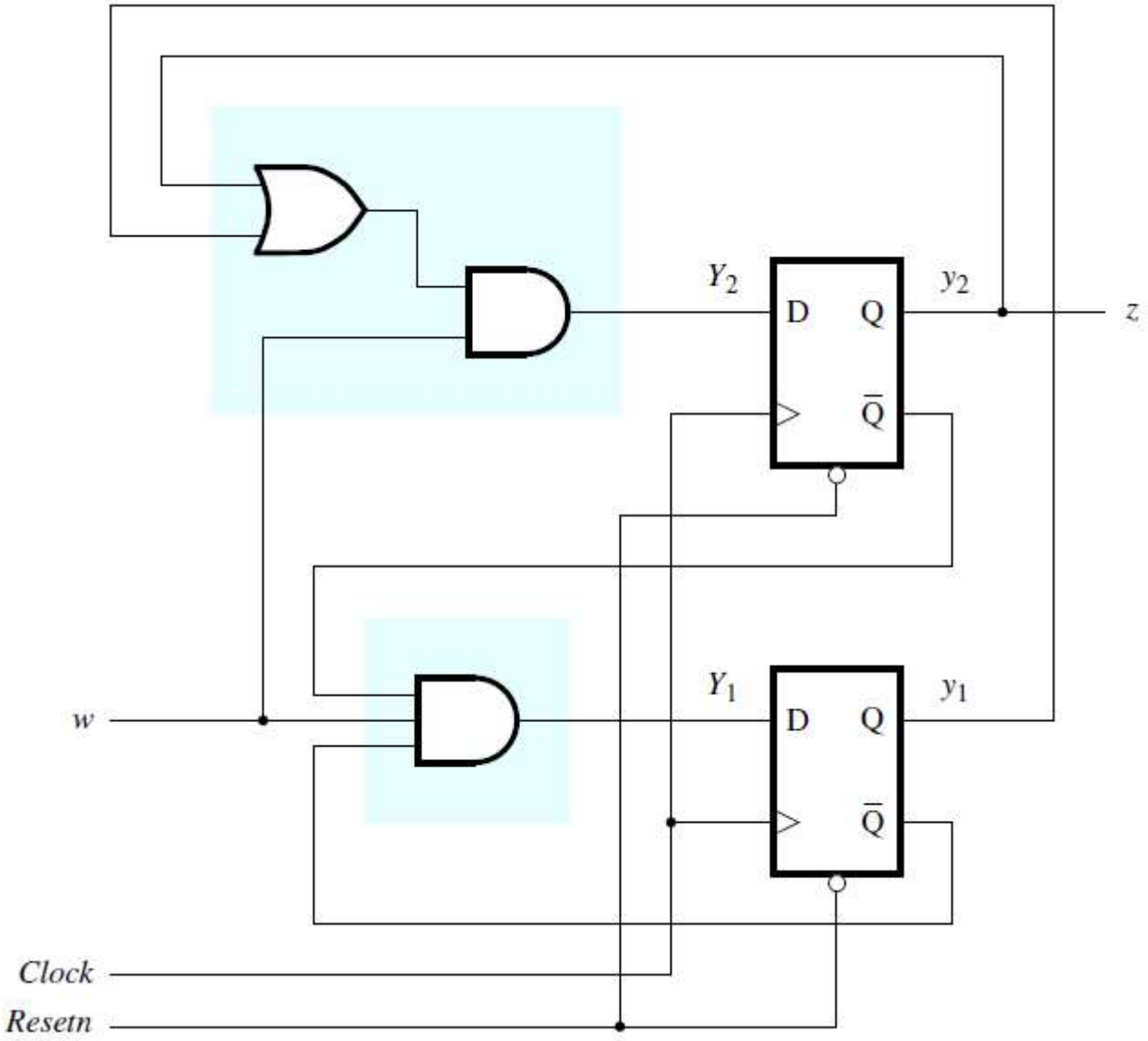






	Ignoring don't cares	Using don't cares
	$Y_1 = w\bar{y}_1\bar{y}_2$	$Y_1 = w\bar{y}_1\bar{y}_2$
	$Y_2 = wy_1\bar{y}_2 + w\bar{y}_1y_2$	$Y_2 = wy_1 + wy_2$ $= w(y_1 + y_2)$
	$z = \bar{y}_1y_2$	$z = y_2$

逻辑电路实现



$$Y_1 = w\bar{y}_1\bar{y}_2$$

$$Y_2 = wy_1 + wy_2$$
$$= w(y_1 + y_2)$$

$$z = y_2$$

仿真波形

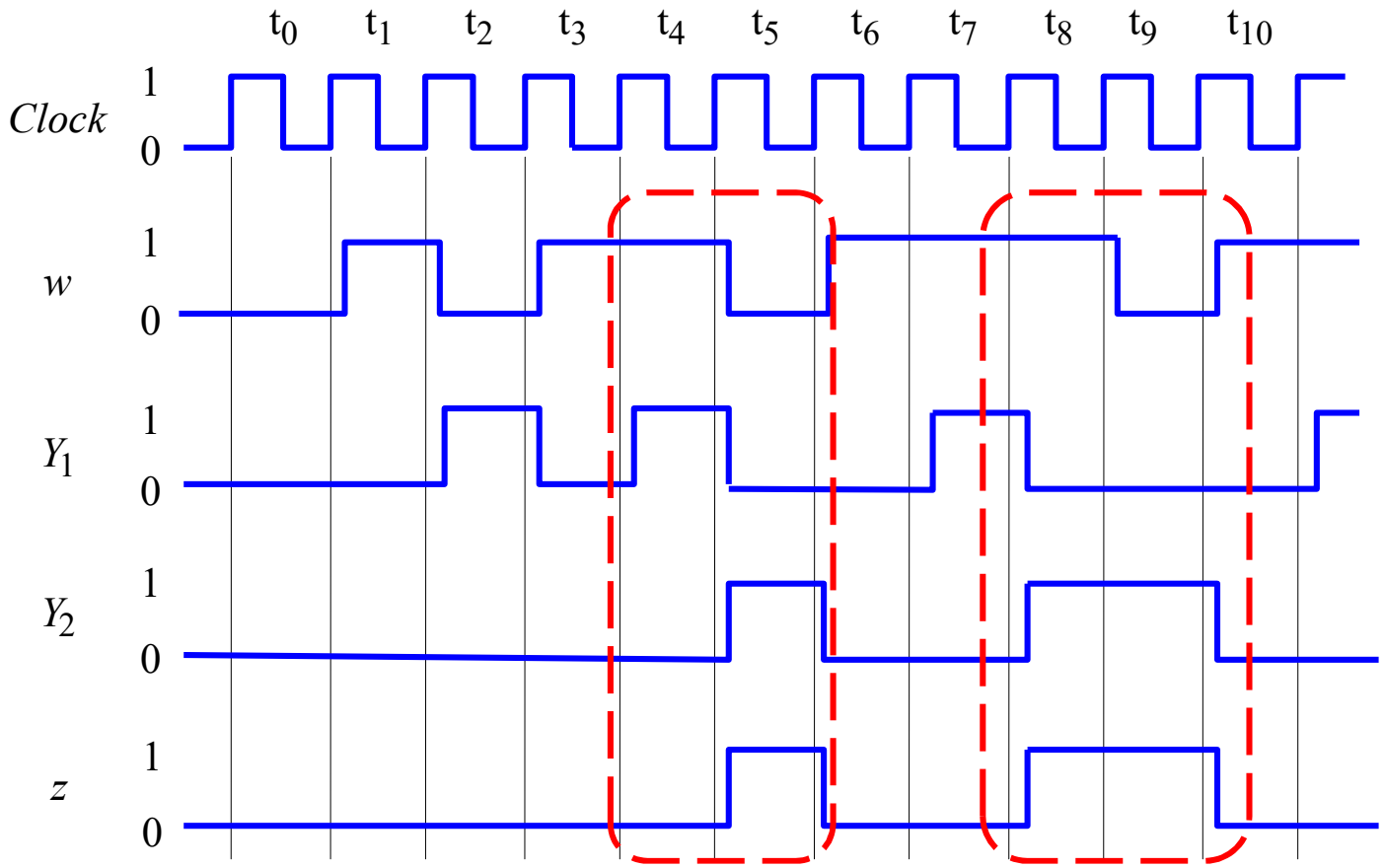
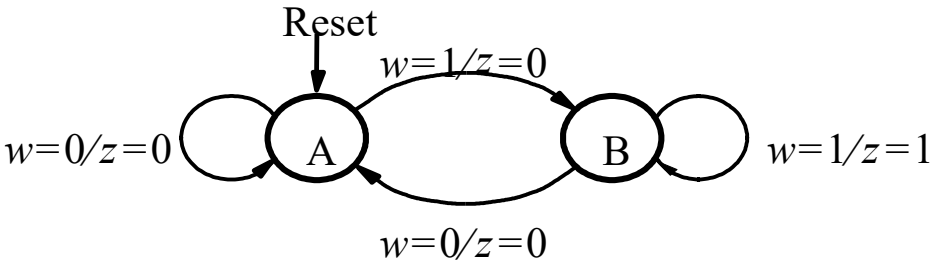


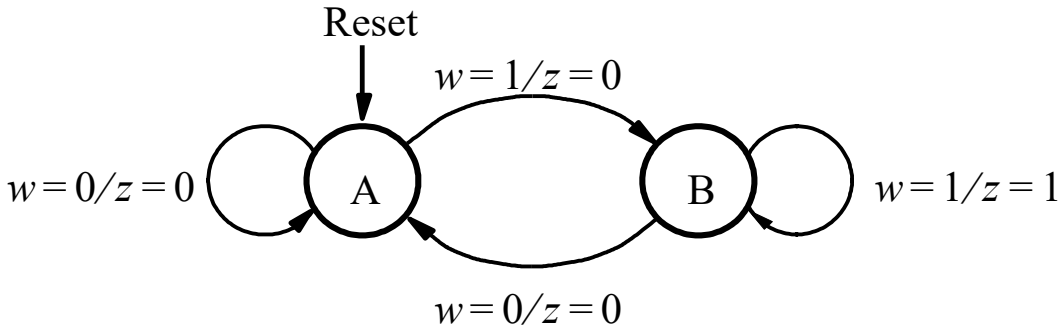
Figure 6.9. Timing diagram for the circuit in Figure 6.8.

Mealy型状态转移图



Mealy型状态表

Present state	Next state		Output z	
	$w = 0$	$w = 1$	$w = 0$	$w = 1$
A	A	B	0	0
B	A	B	0	1

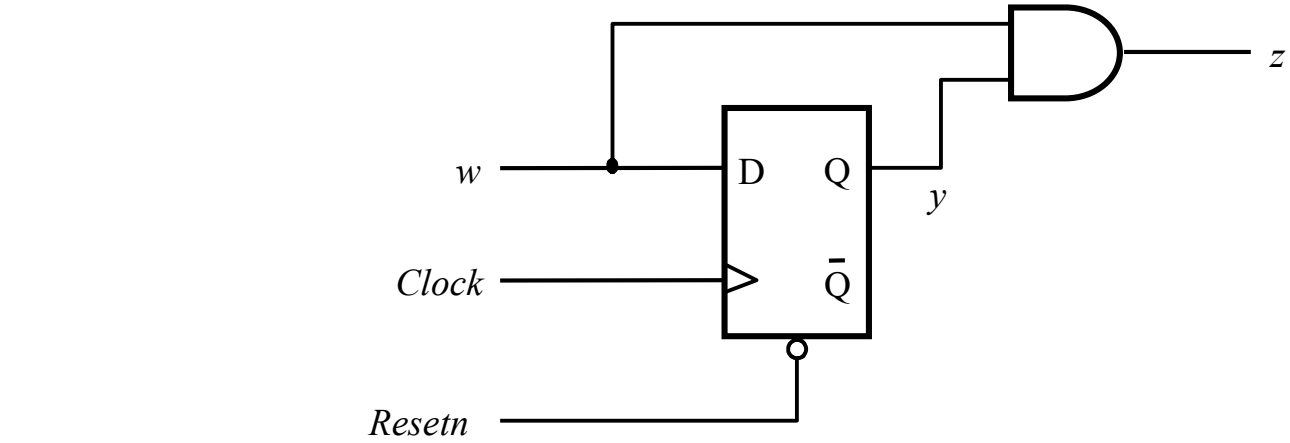


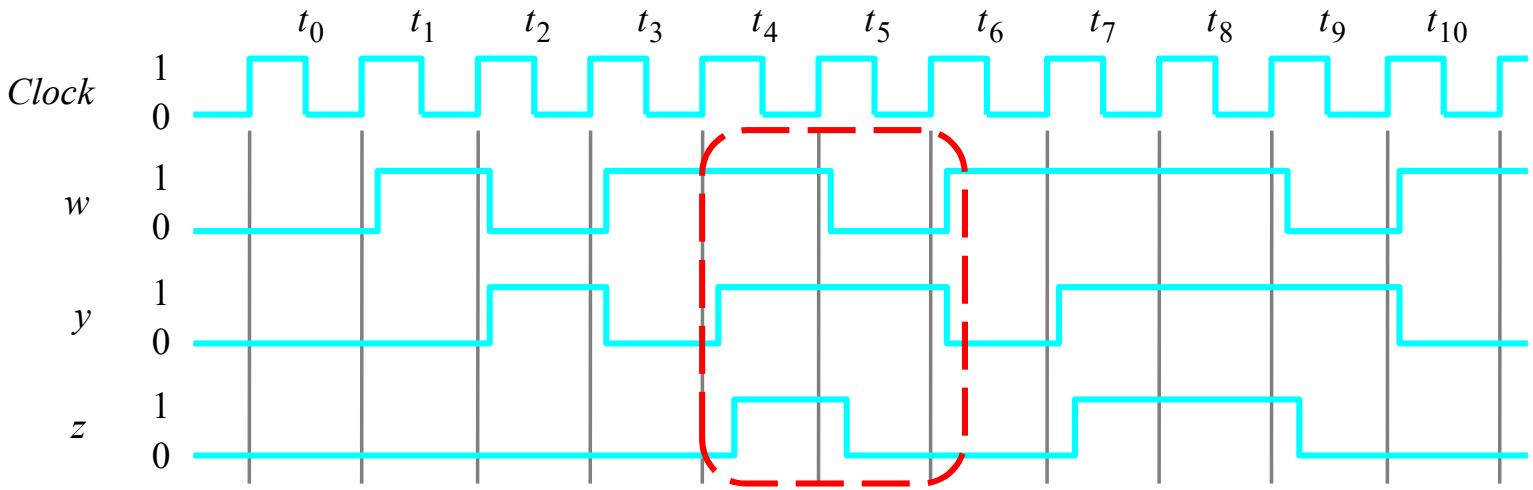
Mealy型状态分配表

	Present state	Next state		Output	
		$w = 0$	$w = 1$	$w = 0$	$w = 1$
	y	Y	Y	z	z
A	0	0	1	0	0
B	1	0	1	0	1

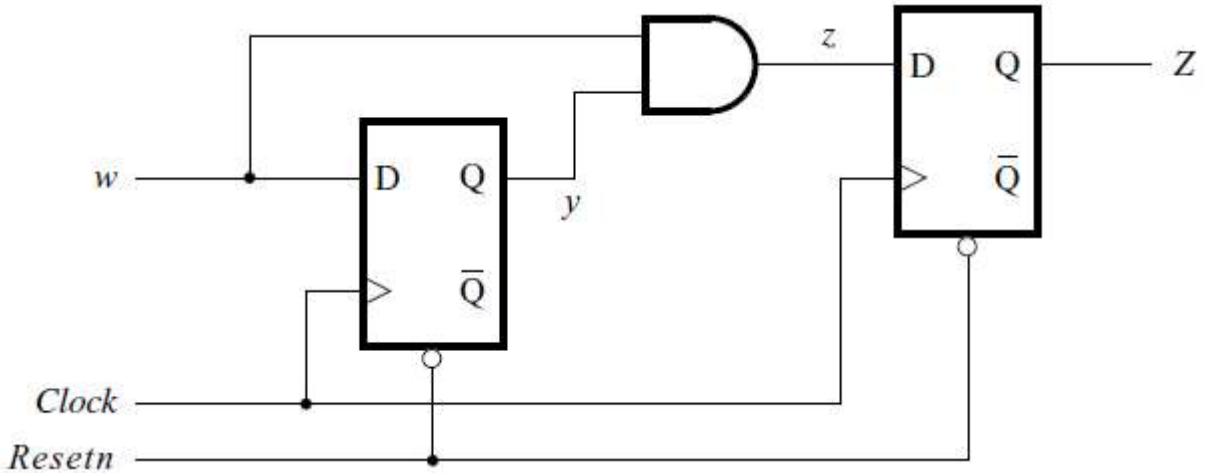
Figure 6.25. State-assigned table for the FSM in Figure 6.24.

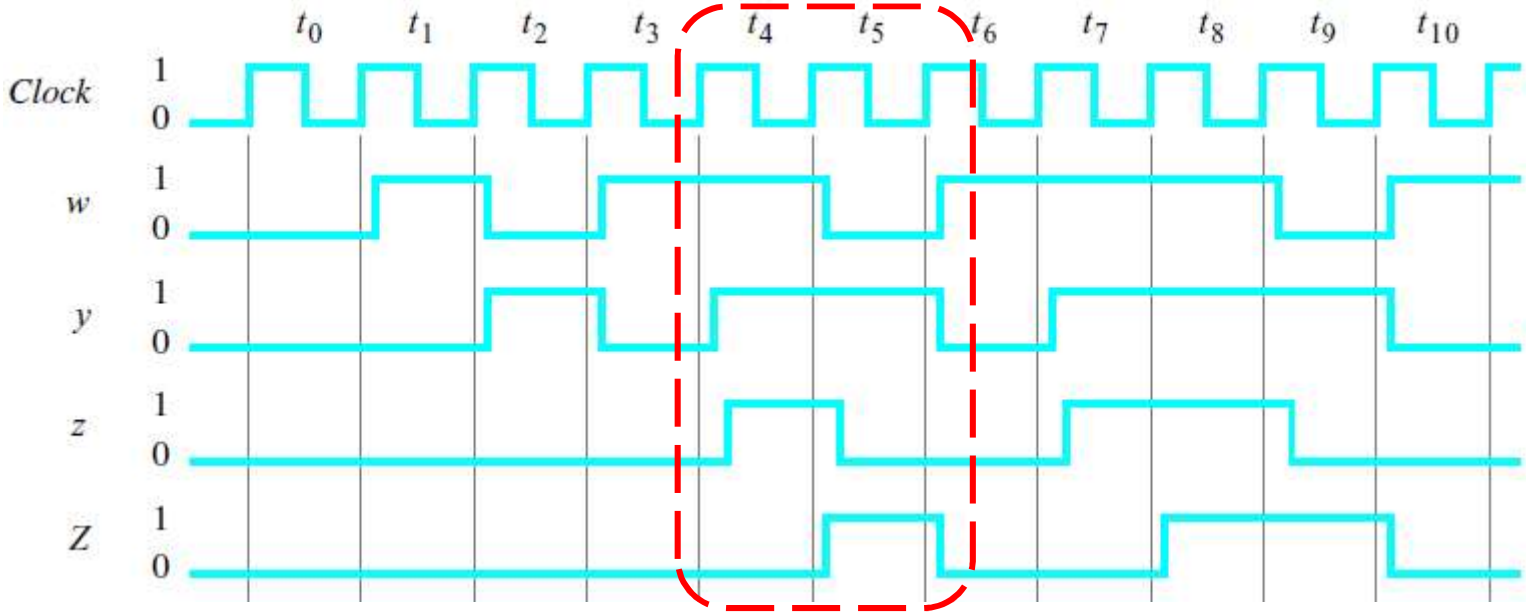
Mealy型状态机的实现





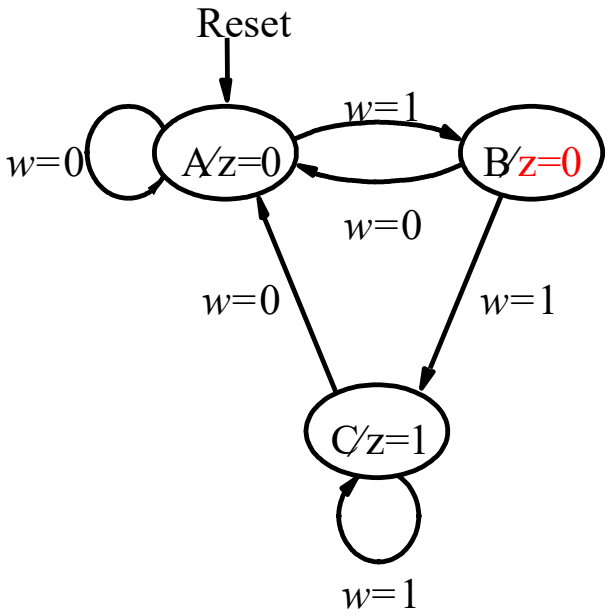
Mealy—>Moore

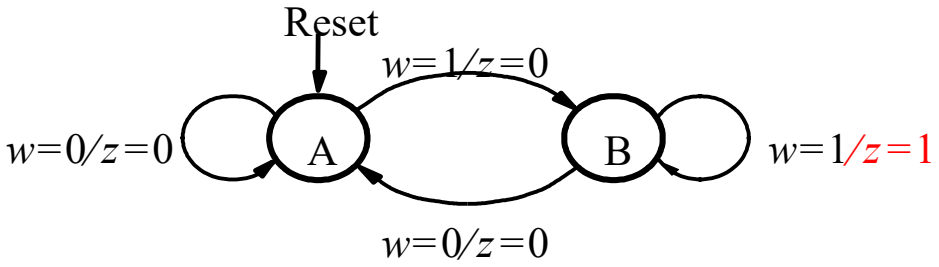

w
D
Q
$\bar{Q}$
y
z
D
Q
$\bar{Q}$
Z
Clock
Resetn


t_0 t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_{10}
Clock
w
y
z
Z
1
0

Moore VS. Mealy

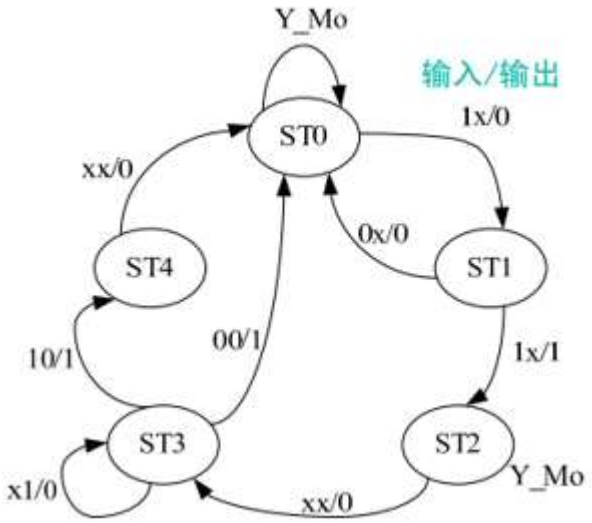
- **Moore:** 输出完全取决于状态，先变状态，后输出
- **Mealy:** 输出取决于状态+输入，先输出，后变状态





状态图设计要点

- 是状态机描述最自然直观的方式
- 离开一个特定状态的弧线上的转移表达式必须是相互排斥且完备的
- 能够复位进入初始态
- 有工具支持状态图作为逻辑设计输入
- 时钟信号是隐含的



状态表设计要点

- 是一种穷举方法，列出所有状态和输入的组合
- 为每个可能的输入组合提供1列表示次态、1列表示输出

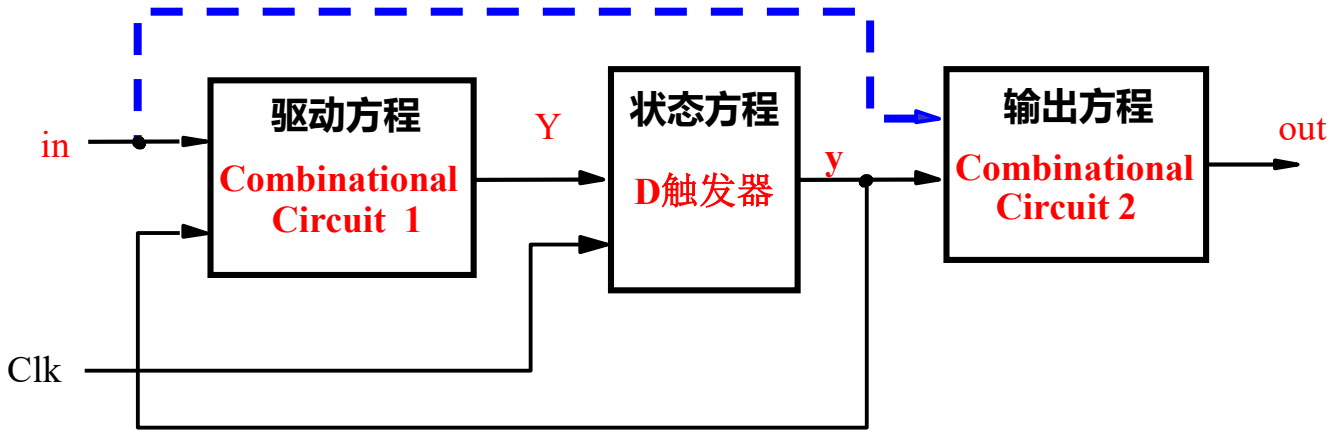
（**Mealy**状态机中可以把输出值和状态值列在一起）

- 状态表的规模随状态数的增加呈指数增长
- 状态表的规模随输入数的增加呈指数增长

	Present state	Next state		Output
		$w = 0$	$w = 1$	
	y_2y_1	Y_2Y_1	Y_2Y_1	z
A	00	00	01	0
B	01	00	10	0
C	10	00	10	1
	11	*dd*	*dd*	*d*

目录

基于D触发器的时序逻辑电路基本模型

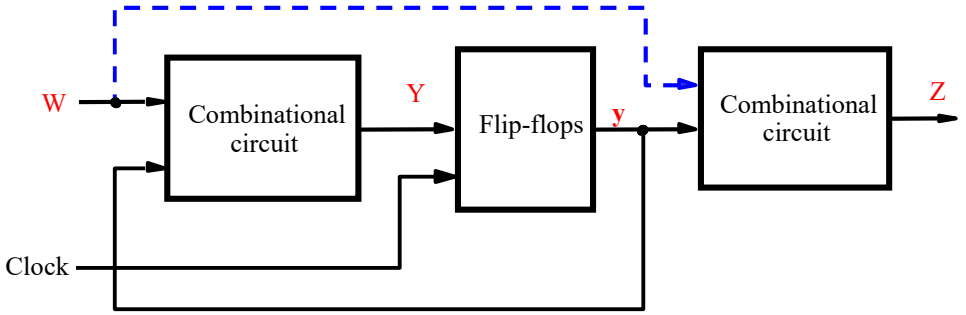


- 状态寄存器：D触发器，保存状态
  - D触发器的输出表示电路的当前状态 y（current_state）
  - D触发器的输入为下一个状态 Y（next_state）
  - 在时钟沿到来后，D触发器输出的即下一个状态 y <= Y
- 组合逻辑电路1：位于D触发器输入端，根据外部输入信号和电路的当前状态计算出下一个状态 Y = f(in, y)
- 组合逻辑电路2：位于D触发器输出端，根据电路当前状态和当前输入计算电路输出 out = f(in, y)

序列检测器：FSM三段式描述方法

```
module simple (Clock, Resetn, w, z);
    input Clock, Resetn, w;
    output z;
    reg [2:1] y, Y;
    parameter [2:1] A = 2'b00, B = 2'b01, C = 2'b10;

    // Define the next state combinational circuit
    always @(w, y)
        case (y)
            A: if (w)   Y = B;
               else     Y = A;
            B: if (w)   Y = C;
               else     Y = A;
            C: if (w)   Y = C;
               else     Y = A;
            default:    Y = A;
        endcase

    // Define the sequential block
    always @(negedge Resetn, posedge Clock)
        if (Resetn = = 0)                y <= A;
        else y <= Y;

    // Define output
    assign z = (y = = C);
endmodule
```

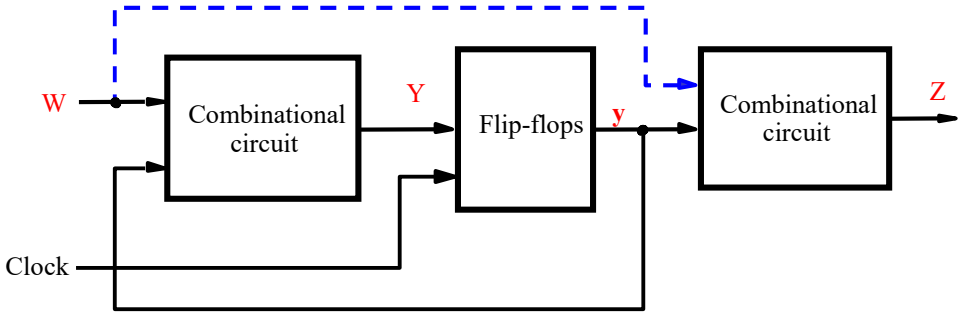
	Present state y_2y_1	Next state $w = 0$ Y_2Y_1	Next state $w = 1$ Y_2Y_1	Output z
A	00	00	01	0
B	01	00	10	0
C	10	00	10	1
	11	*dd*	*dd*	*d*



FSM两段式描述方法

```
module simple (Clock, Resetn, w, z);
    input Clock, Resetn, w;
    output reg z;
    reg [2:1] y, Y;
    parameter [2:1] A = 2'b00, B = 2'b01, C = 2'b10;

    // Define the next state combinational circuit
    always @(w, y)
    begin
        case (y)
            A: if (w)   Y = B;
               else     Y = A;
            B: if (w)   Y = C;
               else     Y = A;
            C: if (w)   Y = C;
               else     Y = A;
            default:    Y = A;
        endcase
        z = (y = = C); //Define output
    end

    // Define the sequential block
    always @(negedge Resetn, posedge Clock)
        if (Resetn = = 0)  y <= A;
        else y <= Y;

endmodule
```

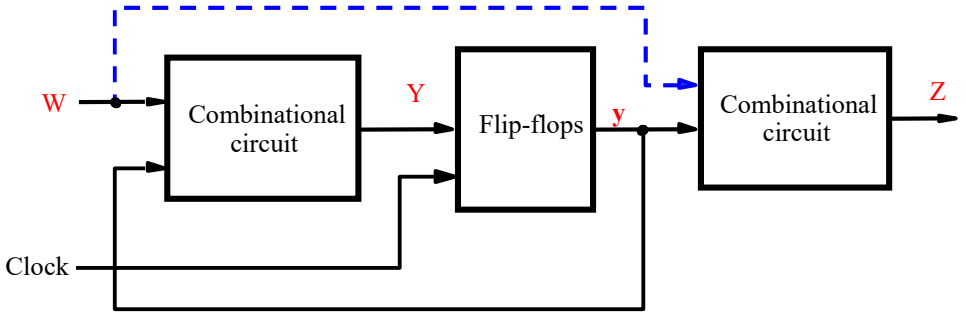
	Present state y_2y_1	Next state $w = 0$ Y_2Y_1	Next state $w = 1$ Y_2Y_1	Output z
A	00	00	01	0
B	01	00	10	0
C	10	00	10	1
	11	*dd*	*dd*	*d*



FSM两段式描述方法

```
module simple (Clock, Resetn, w, z);
    input Clock, Resetn, w;
    output z;
    reg [2:1] y;
    parameter [2:1] A = 2'b00, B = 2'b01, C = 2'b10

    // Define the sequential block
    always @(negedge Resetn, posedge Clock)
        if (Resetn = = 0)   y <= A;
        else
            case (y)
                A: if (w)   y <= B;
                   else     y <= A;
                B: if (w)   y <= C;
                   else     y <= A;
                C: if (w)   y <= C;
                   else     y <= A;
                default:    y <= A;
            endcase

    // Define output
    assign z = (y = = C);

endmodule
```

	Present state y_2y_1	Next state $w = 0$ Y_2Y_1	Next state $w = 1$ Y_2Y_1	Output z
A	00	00	01	0
B	01	00	10	0
C	10	00	10	1
	11	*dd*	*dd*	*d*



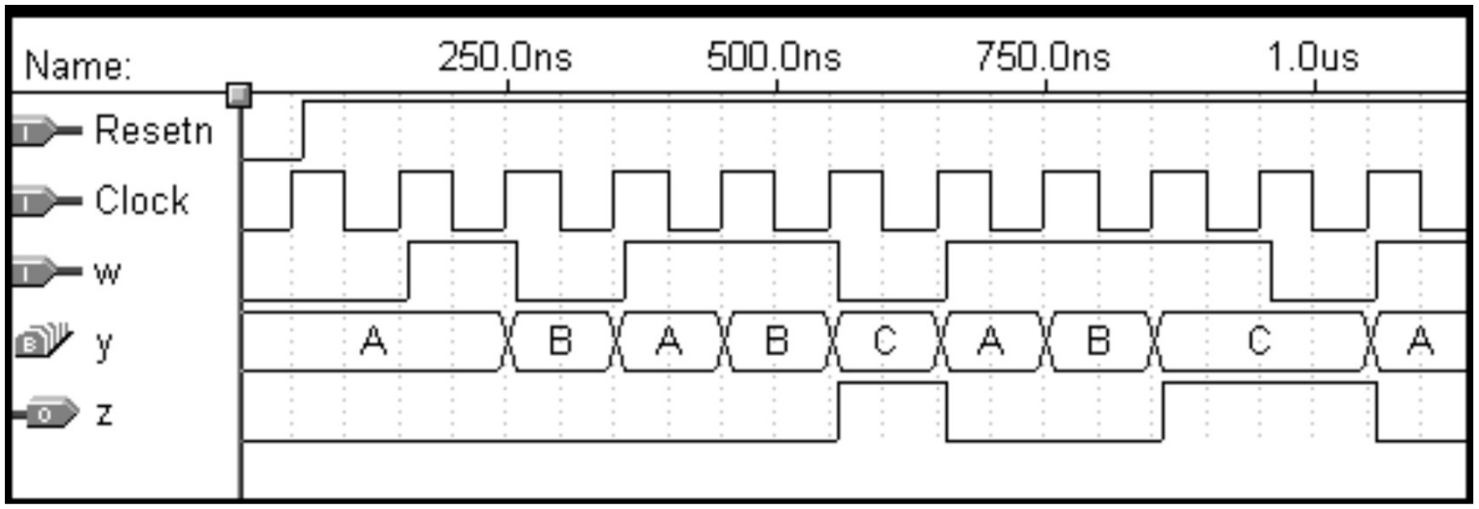
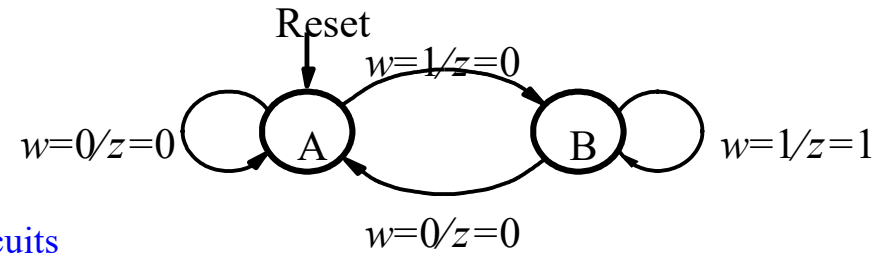


Figure 6.32. Simulation results for the circuit in Figure 6.30.

Mealy状态机描述

```
module mealy (Clock, Resetn, w, z);
    input Clock, Resetn, w;
    output reg z;
    reg y, Y;
    parameter A = 1'b0, B = 1'b1;
    // Define the next state and output combinational circuits
    always @(w, y)
        case (y)
            A:  if (w) begin
                    z = 0;
                    Y = B;
                end
                else  begin
                    z = 0;
                    Y = A;
                end
            B:  if (w) begin
                    z = 1;
                    Y = B;
                end
                else  begin
                    z = 0;
                    Y = A;
                end
        endcase
```



	Present state	Next state		Output	
		$w = 0$	$w = 1$	$w = 0$	$w = 1$
	y	Y	Y	z	z
A	0	0	1	0	0
B	1	0	1	0	1

```
    // Define the sequential block
    always @(negedge Resetn, posedge Clock)
                if (Resetn = = 0) y <= A;
                else              y <= Y;
endmodule
```

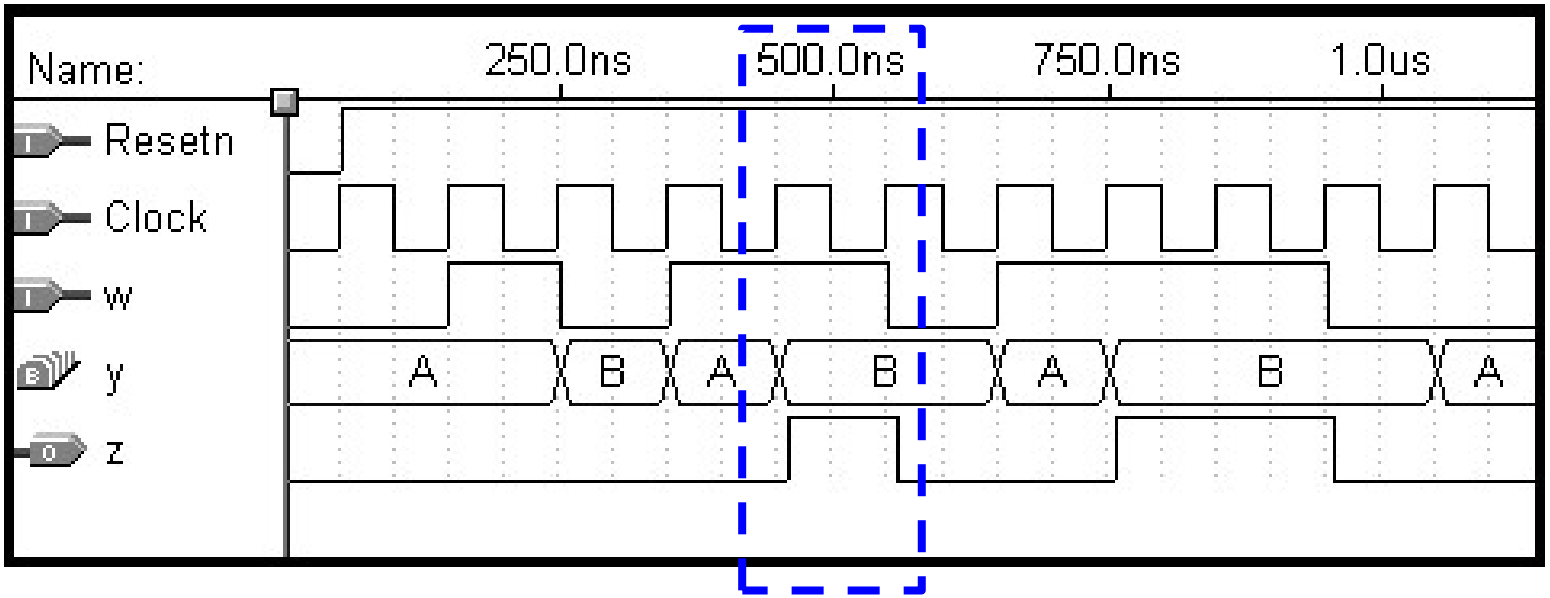
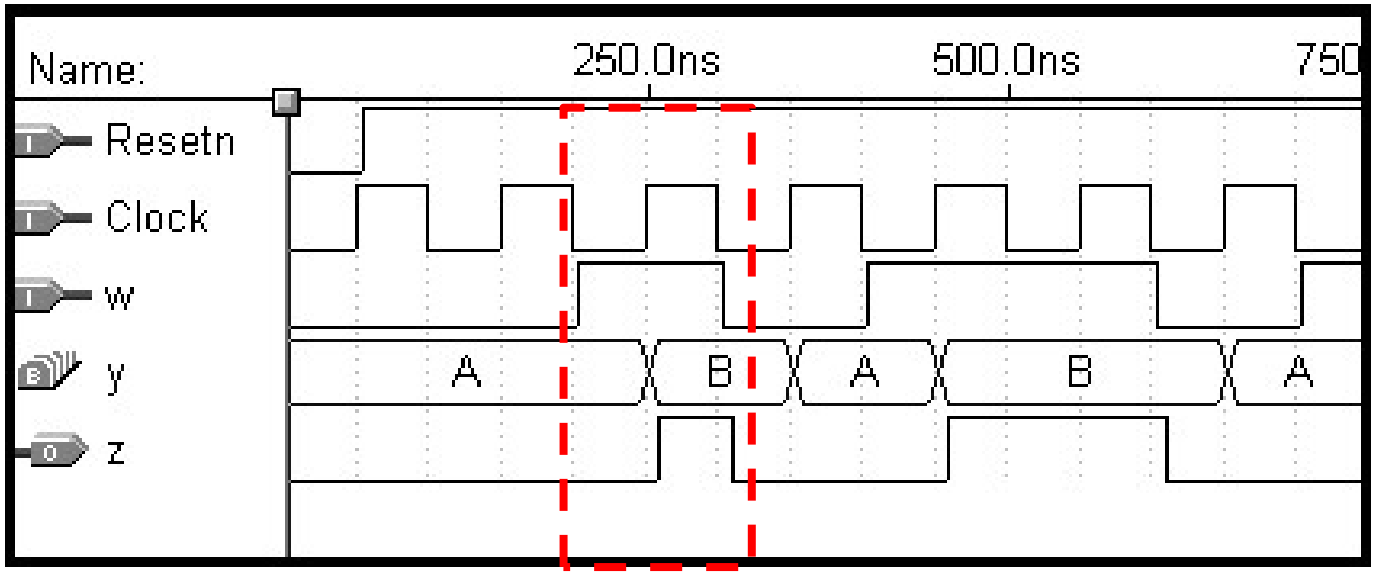


Figure 6.37. Simulation results for the Mealy machine.

异步输入导致Mealy状态机出错



	Present state	Next state		Output	
		$w = 0$	$w = 1$	$w = 0$	$w = 1$
	y	Y	Y	z	z
A	0	0	1	0	0
B	1	0	1	0	1

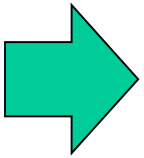
Figure 6.38. Potential problem with asynchronous inputs to a Mealy FSM.

目录

顺序码VS Gray码

顺序码

	Present state y_2y_1	Next state $w=0$ Y_2Y_1	Next state $w=1$ Y_2Y_1	Output z
A	00	00	01	0
B	01	00	10	0
C	**10**	00	10	1
	11	*dd*	*dd*	*d*



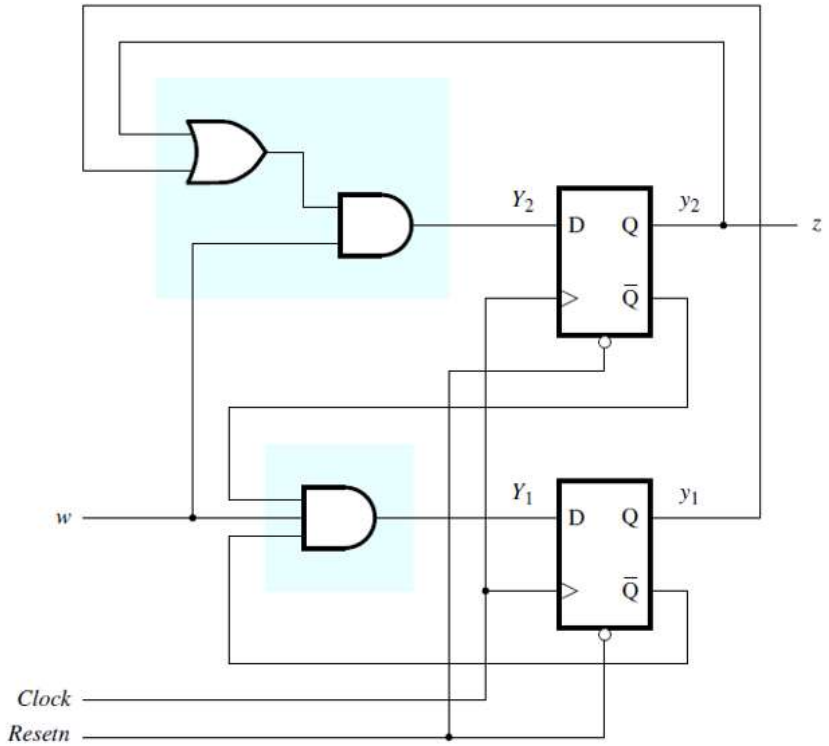
Gray编码

	Present state y_2y_1	Next state $w=0$ Y_2Y_1	Next state $w=1$ Y_2Y_1	Output z
A	00	00	01	0
B	01	00	11	0
C	**11**	00	11	1
	10	*dd*	*dd*	*d*

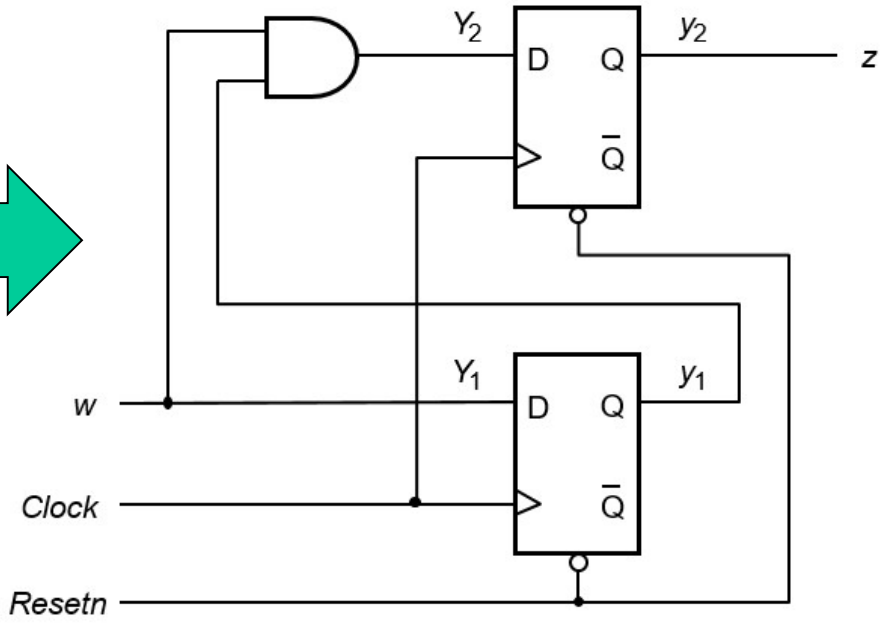
Figure 6.16. Improved state assignment for the sequential circuit in Figure 6.4.

顺序码VS Gray码

顺序码



Gray编码



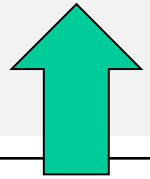
独热编码

$$Y_1 = \bar{w}y_1 + y_4 \qquad R1\text{out} = R2\text{in} = y3$$

$$Y_2 = wy_1 \qquad R1\text{in} = R3\text{out} = Done = y4$$

$$Y3 = y2 \qquad R2\text{out} = R3\text{in} = y2$$

$$Y4 = y3$$



	Present state	Nextstate		Outputs						
		$w=0$	$w=1$							
	$y_4y_3y_2y_1$	$Y_4Y_3Y_2Y_1$	$Y_4Y_3Y_2Y_1$	$R1_{out}$	$R1_{in}$	$R2_{out}$	$R2_{in}$	$R3_{out}$	$R3_{in}$	$Done$
A	0 001	0001	0010	0	0	0	0	0	0	0
B	0 010	0100	0100	0	0	1	0	0	1	0
C	0 100	1000	1000	1	0	0	1	0	0	0
D	1 000	0001	0001	0	1	0	0	1	0	1

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

目录

案例：检测器

- 设计一个具有两输入（A和B）和一个输出（Z）的状态机，Z为1的条件是：
  - 1、在前两个时钟沿，A的值相同；或者
  - 2、从上一次第1个条件为真起，B的值一直为1
- 否则输出为0。

意义	AB					
	S	00	01	11	10	Z
初始状态	INIT	A0	A0	A1	A1	0
在A上收到一个0	A0	OK0	OK0	A1	A1	0
在A上收到一个1	A1	A0	A0	OK1	OK1	0
条件满足，最终A=0	OK0	OK0	OK0	OK1	A1	1
条件满足，最终A=1	OK1	A0	OK0	OK1	OK1	1

三段式FSM

意义	AB					
	S	00	01	11	10	Z
初始状态	INIT	A0	A0	A1	A1	0
在A上收到一个0	A0	OK0	OK0	A1	A1	0
在A上收到一个1	A1	A0	A0	OK1	OK1	0
条件满足，最终A=0	OK0	OK0	OK0	OK1	A1	1
条件满足，最终A=1	OK1	A0	OK0	OK1	OK1	1

```
module ex1 (Clock, A, B, Z);
    input Clock, A, B;
    output reg Z;
    reg [2:0] c_s, n_s;

    parameter [2:0] INIT = 3'b000,
                    A0   = 3'b001,
                    A1   = 3'b010,
                    OK0 = 3'b011,
                    OK1 = 3'b100,

    always @(posedge Clock)
        c_s <= n_s;

    always @(c_s)
        case (c_s)
            INIT, A0, A1:  Z=0;
            OK0,OK1:       Z=1;
            default:       Z=0;
        endcase
```

```
    always @(A, B, c_s) begin
        case (c_s)
          INIT:  if (A==0)              n_s=A0;
                 else                   n_s=A1;
          A0:    if (A==0)              n_s=OK0;
                 else                   n_s=A1;
          A1:    if (A==0)              n_s=A0;
                 else                   n_s=OK1;
          OK0:   if (A==0)              n_s=OK0;
                 else if (( A==1) && (B==0))
                                        n_s=A1;
                 else                   n_s=OK1;
          OK1:   if ((A==0)  && (B==0))
                                        n_s=A0;
                 else if (( A==0) && (B==1))
                                        n_s=OK0;
                 else                   n_s=OK1;
          default:                      n_s=INIT;
        endcase
    end
endmodule
```

- 设计一个具有两输入（A和B）和一个输出（Z）的状态机，Z为1的条件是：
  - 1、在前两个时钟沿，A的值相同；或者
  - 2、从上一次第1个条件为真起，B的值一直为1
- 否则输出为0。

```
module ex1 (Clock, A, B, Z);
    input Clock, A, B;
    output wire Z;
    reg LASTA;
    reg [1:0] c_s, n_s;
    parameter [1:0] INIT = 2'b00,
                    Looking  = 2'b01,
                    OK       = 2'b11,

    always @(posedge Clock) begin
        if (rst)  c_s <= INIT;
        else      c_s <= n_s;
        LASTA <= A;
    end
    assign Z = (c_s==OK)?1:0;
```

```
always @(A, B, LASTA, c_s) begin
    case (c_s)
      INIT:                    n_s=Looking;
      Looking:if (A==LASTA)
                               n_s=OK;
              else             n_s=Looking;
      OK:  if ((A==LASTA) || (B==1))
                               n_s=OK;
           else                n_s=Looking;
      default:                 n_s=INIT;
    endcase
end
endmodule
```

- 次态逻辑更容易理解
- 与状态机的文字描述更相关
- 无需设计状态表

好的FSM设计标准

- 好的**FSM**
  - 可靠、稳定
  - 清晰易懂、易维护
- **HDL**建模考虑的要点
  - 可靠复位：进入初始态/空闲态
  - 组合逻辑避免锁存
  - 看门狗：避免死机
  - 状态编码、三段式风格、注释

基于时序电路的计数器设计

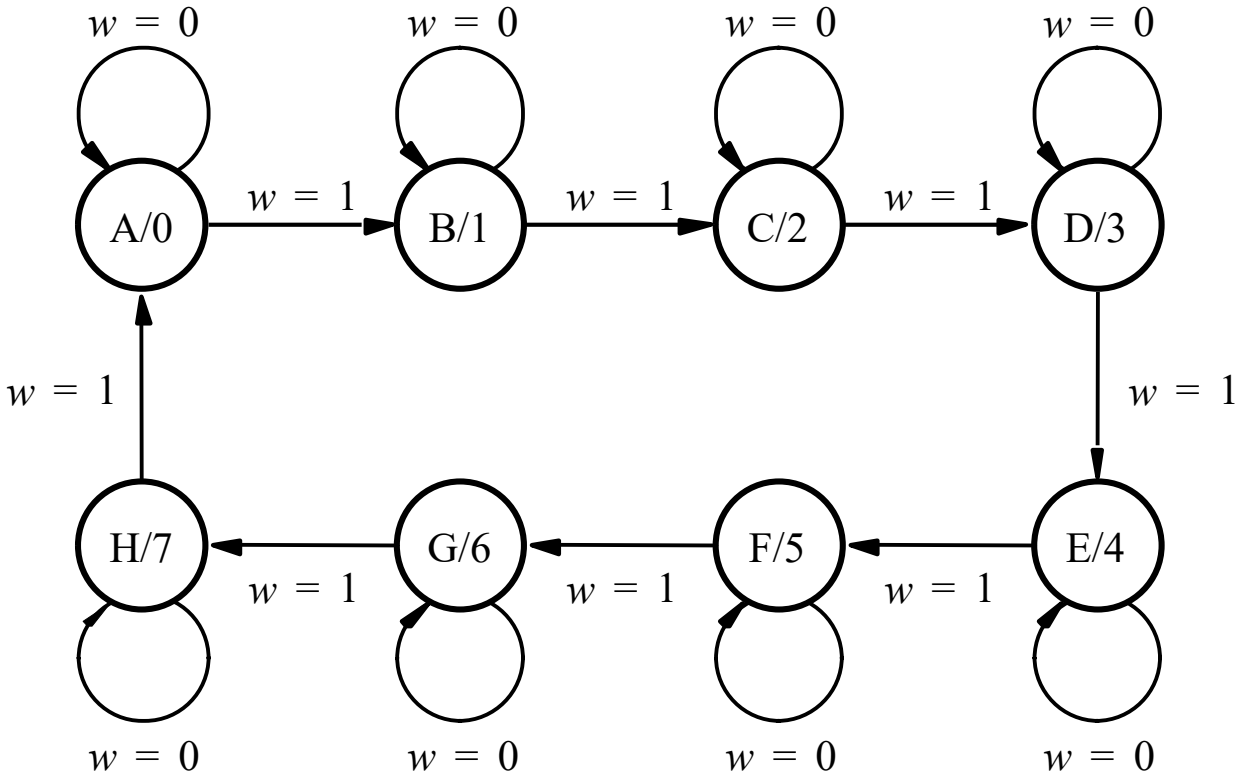
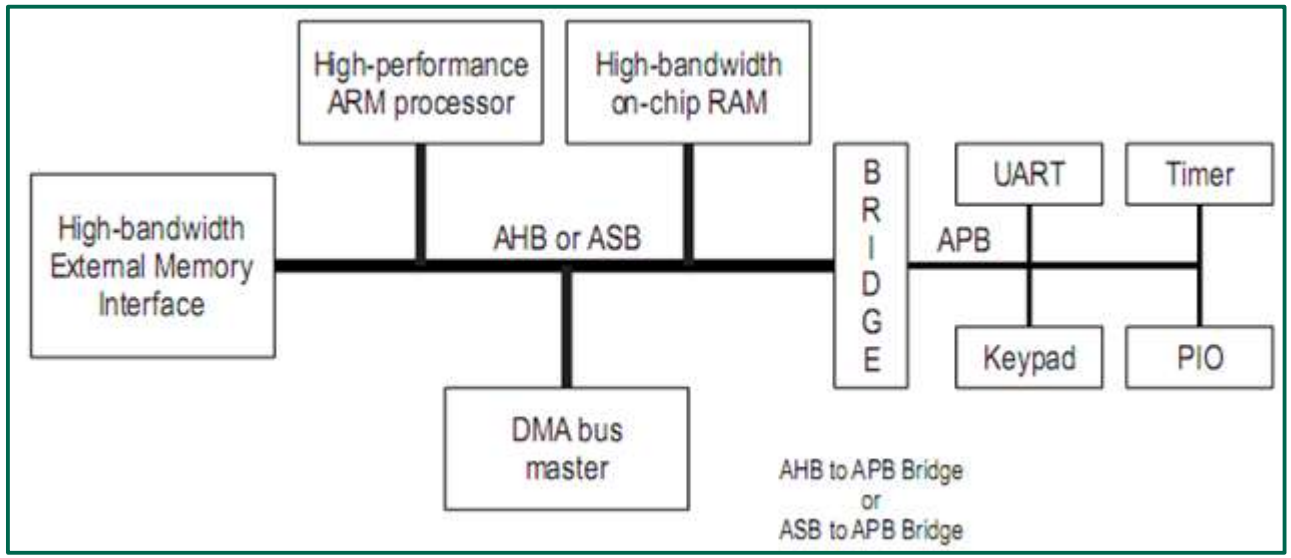


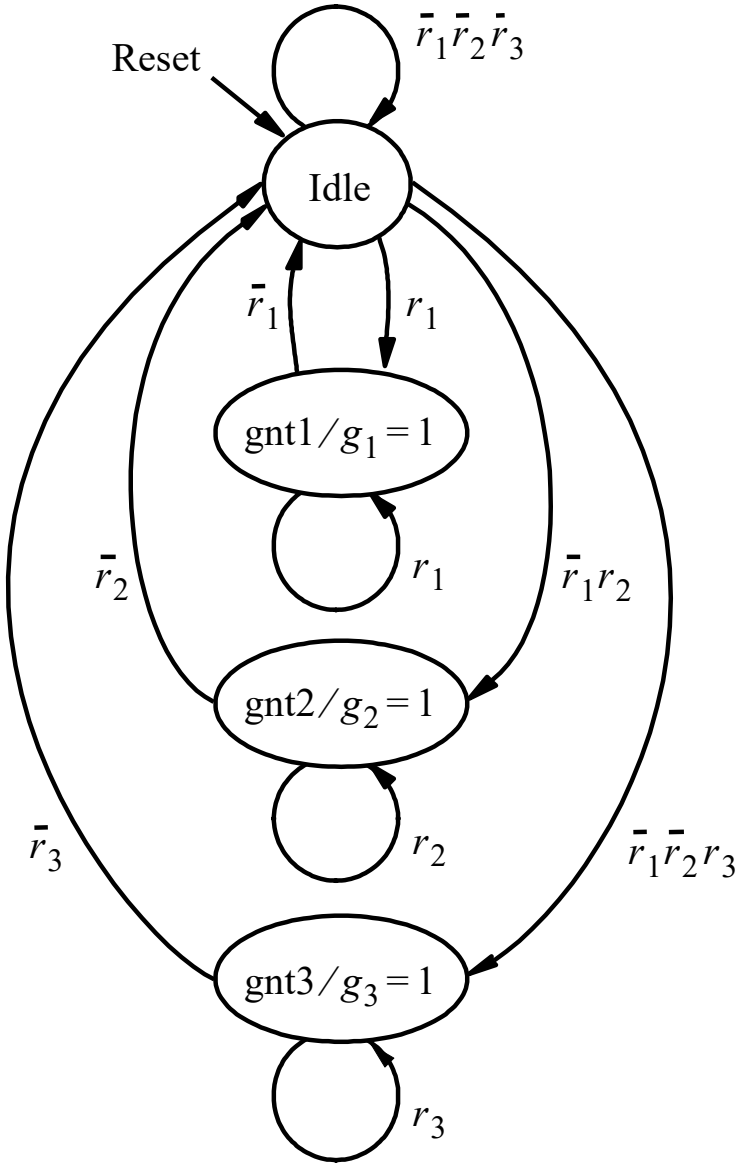
Figure 6.60. State diagram for the counter.

仲裁电路FSM

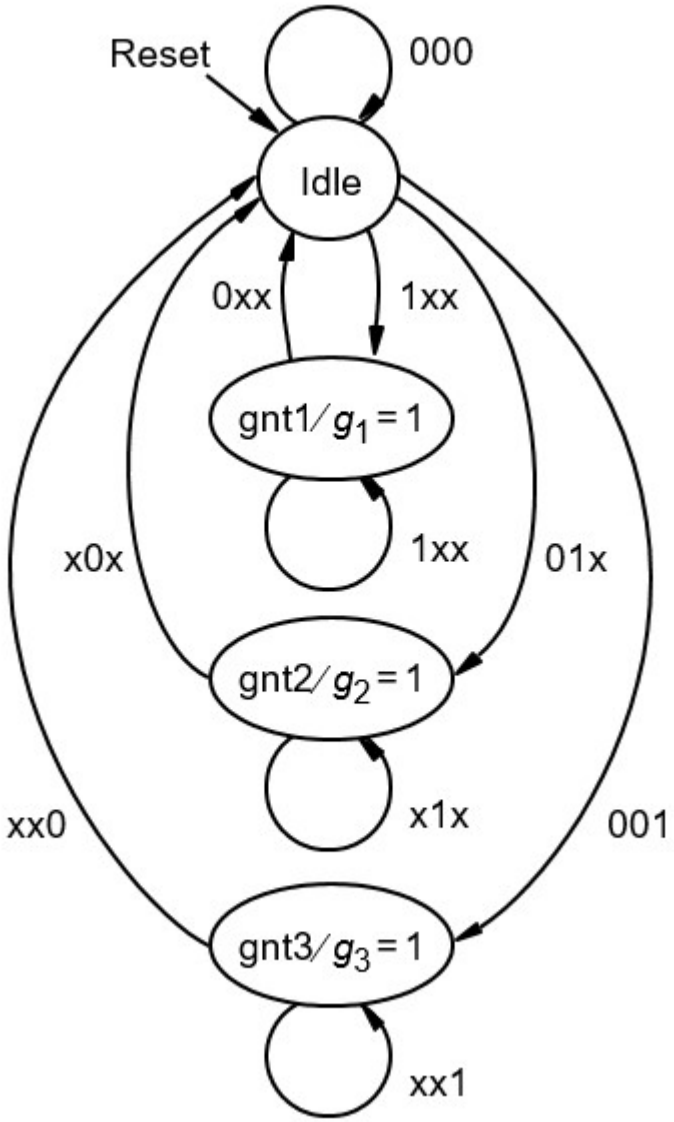
优先级：
1 金卡客户
2 银卡客户
3 普通客户



仲裁电路FSM（Moore）

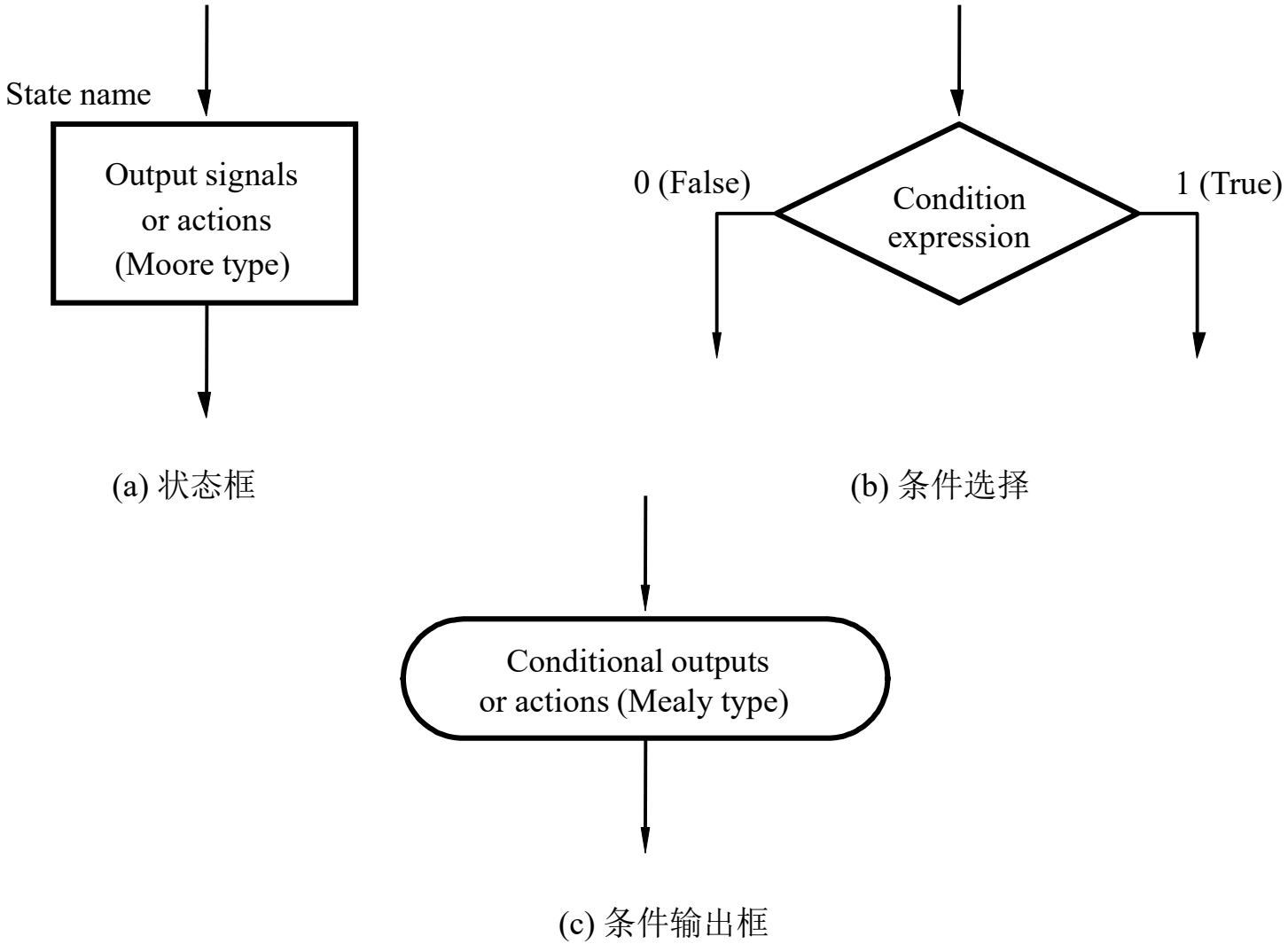


```
module arbiter (r, Resetn, Clock, g);
    input [1:3] r;
    input Resetn, Clock;
    output wire [1:3] g;
    reg [2:1] y, Y;
    parameter Idle = 2'b00, gnt1 = 2'b01, gnt2 = 2'b10, gnt3 = 2'b11;

    // Next state combinational circuit
    always @(r, y)
        case (y)
            Idle:
                if (r[1]) Y = gnt1;
                else if (r[2]) Y = gnt2;
                else if (r[3]) Y = gnt3;
                else Y = Idle;
            gnt1: if (r[1])  Y = gnt1;
                else  Y = Idle;
            gnt2: if (r[2])  Y = gnt2;
                else  Y = Idle;
            gnt3: if (r[3])  Y = gnt3;
                else  Y = Idle;
            default: Y = Idle;
        endcase

    // Sequential block
    always @(posedge Clock)
        if (Resetn == 0)    y <= Idle;
        else     y <= Y;

    //  Define output
    assign g[1] = (y == gnt1);
    assign g[2] = (y == gnt2);
    assign g[3] = (y == gnt3);

endmodule
```



目录

Algorithmic State Machine（ASM）



(a) 状态框

(b) 条件选择

(c) 条件输出框

Figure 6.81. Elements used in ASM charts.

序列检测器ASM（Moore）

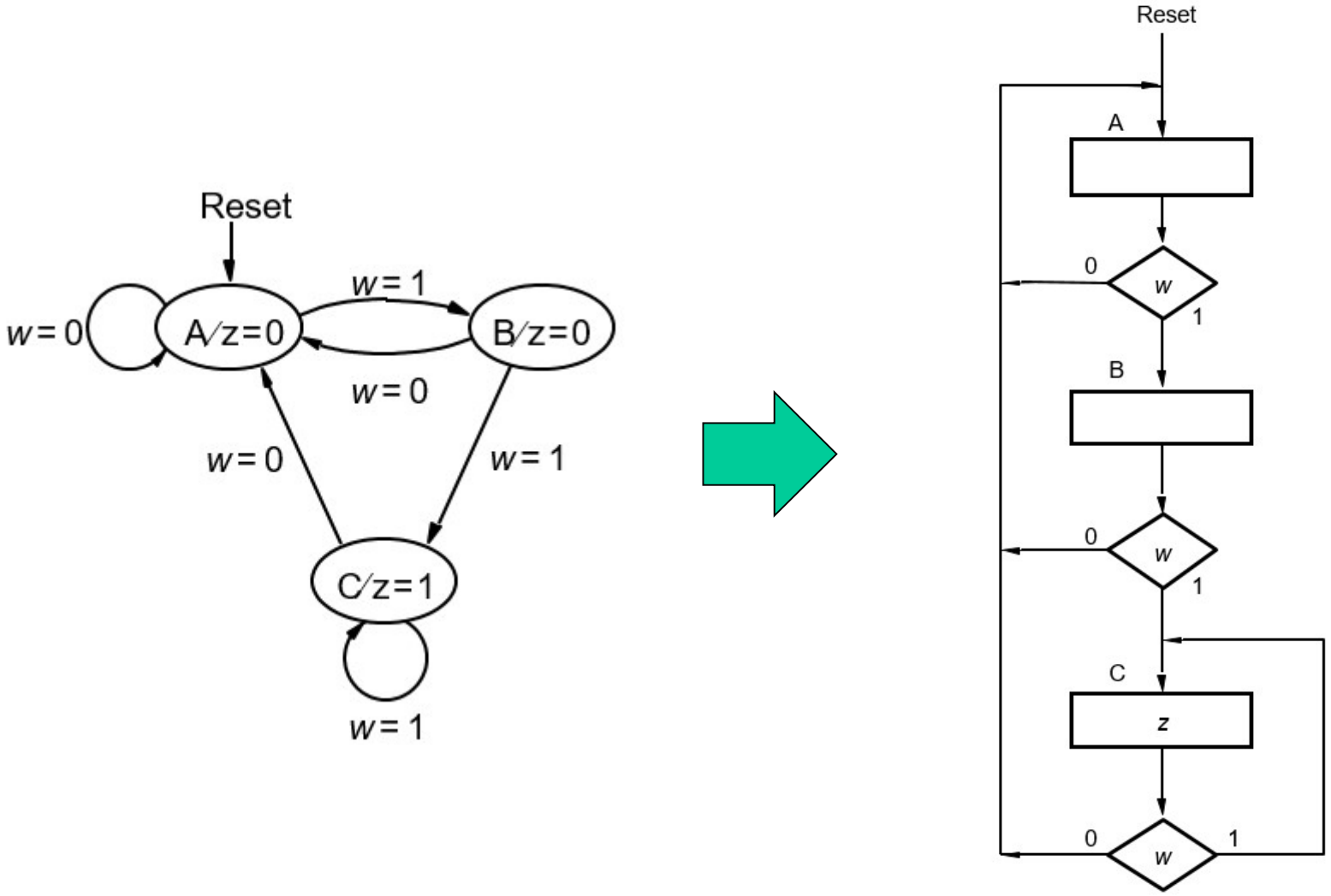


Figure 6.82. ASM chart for the FSM in Figure 6.3.

序列检测器ASM（Mealy）

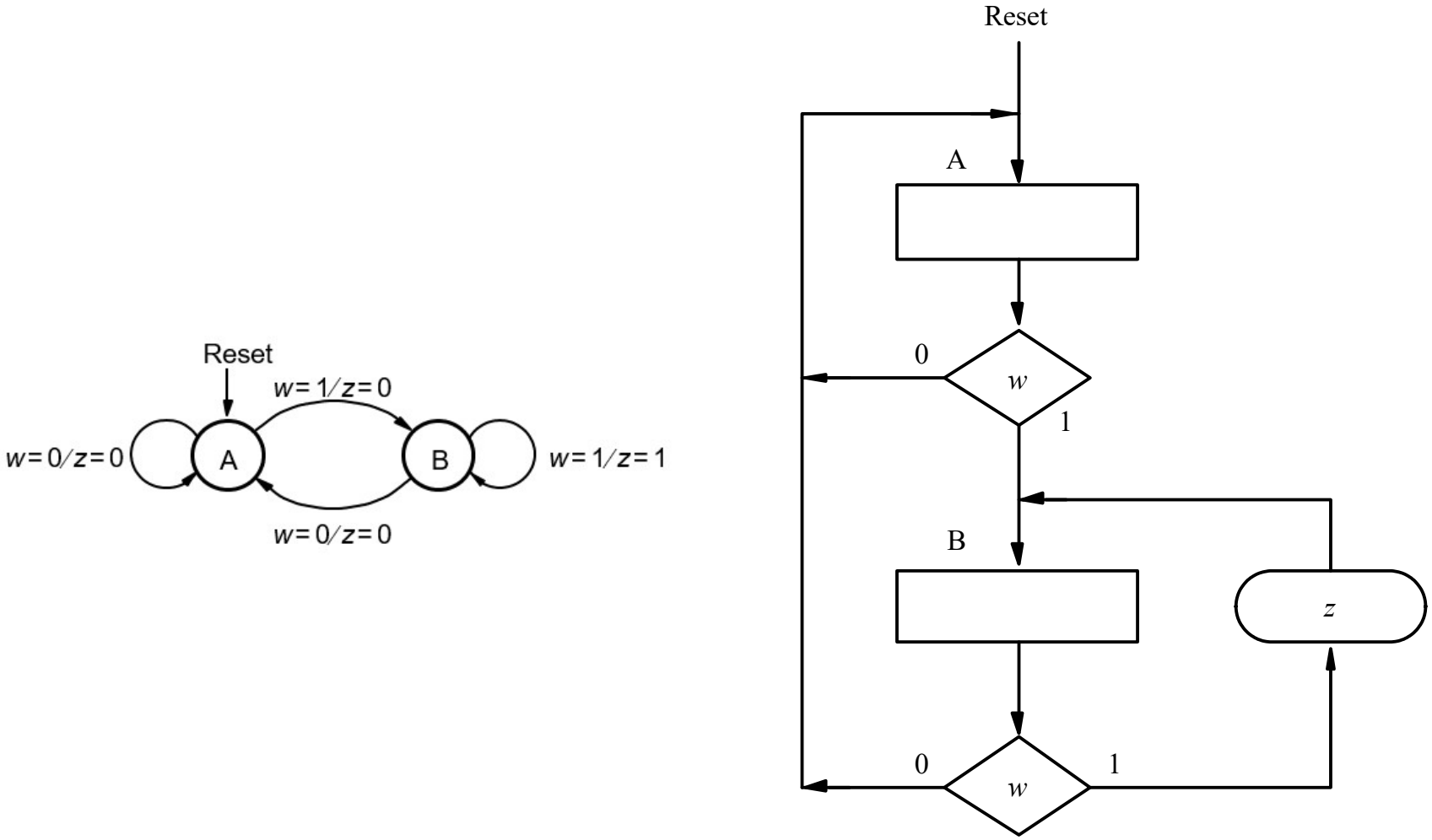


Figure 6.83. ASM chart for the FSM in Figure 6.23.

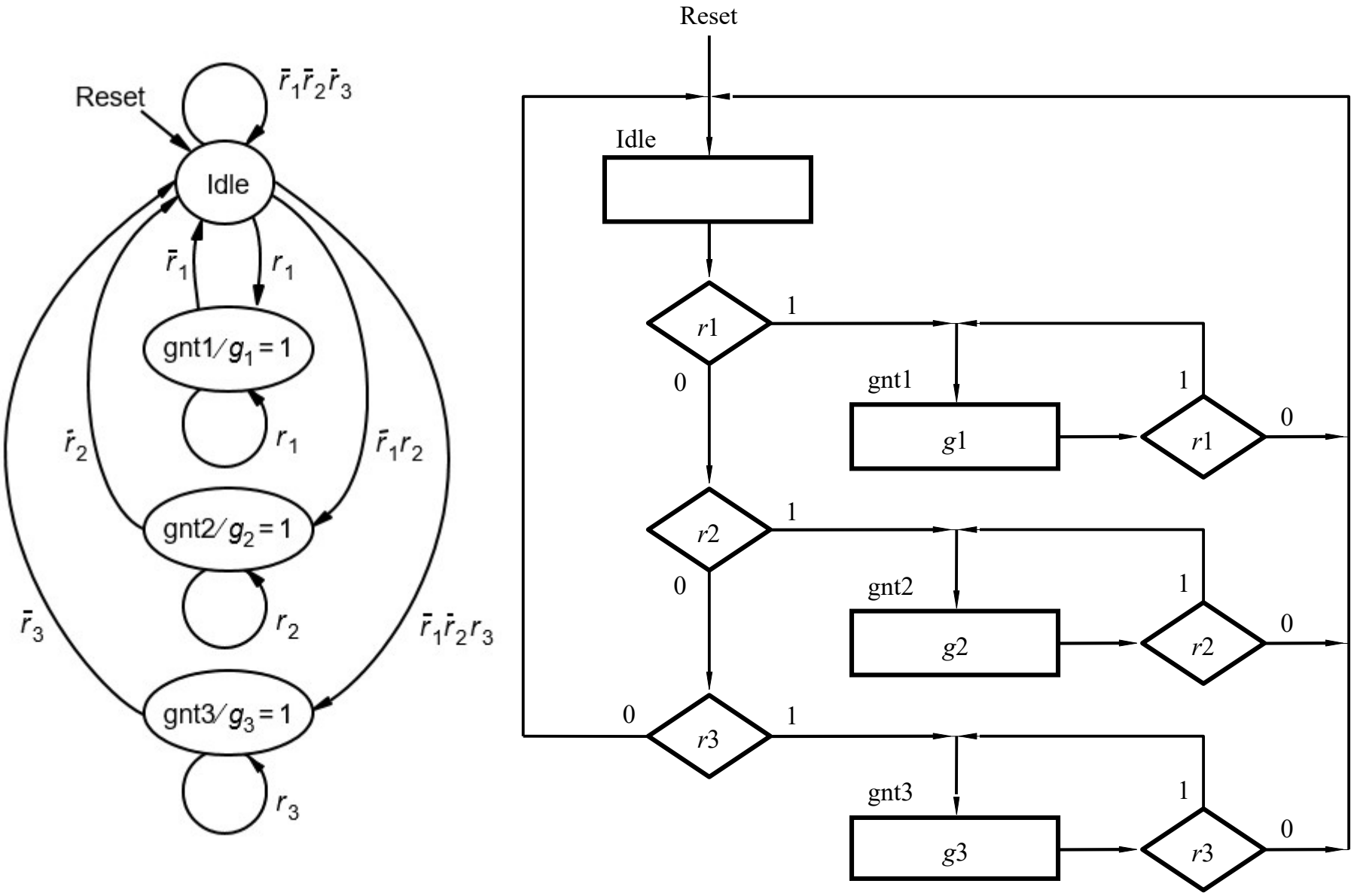


Figure 6.84. ASM chart for the arbiter FSM in Figure 6.73.