数字逻辑与部件设计

9. 时序电路分析

xgsun@fudan.edu.cn

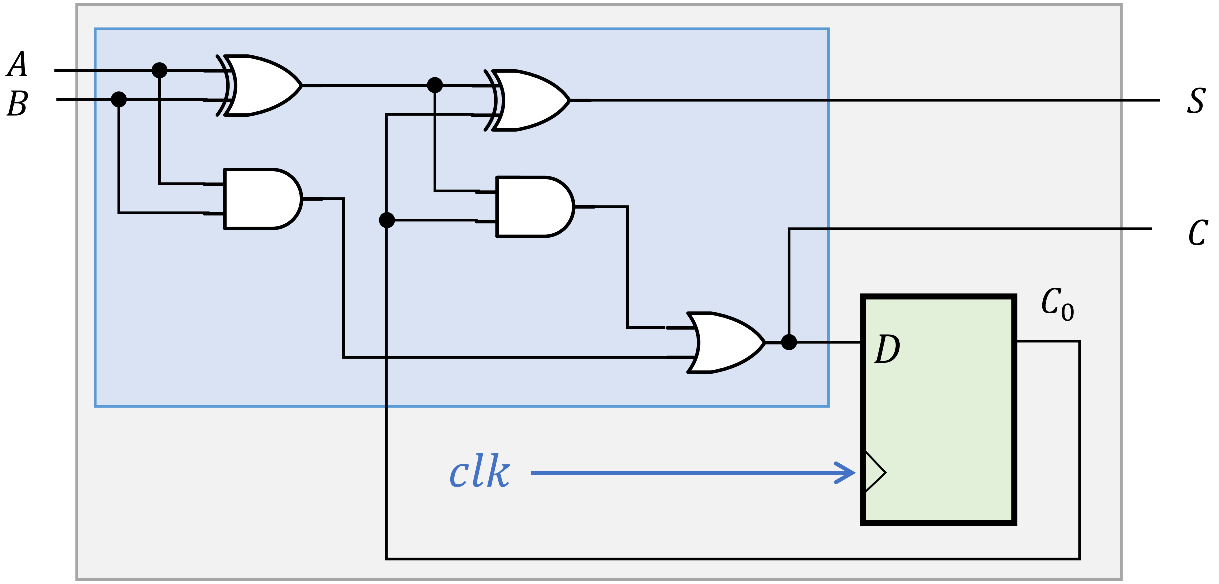
孙晓光

2024-10-27

1

时序电路分析

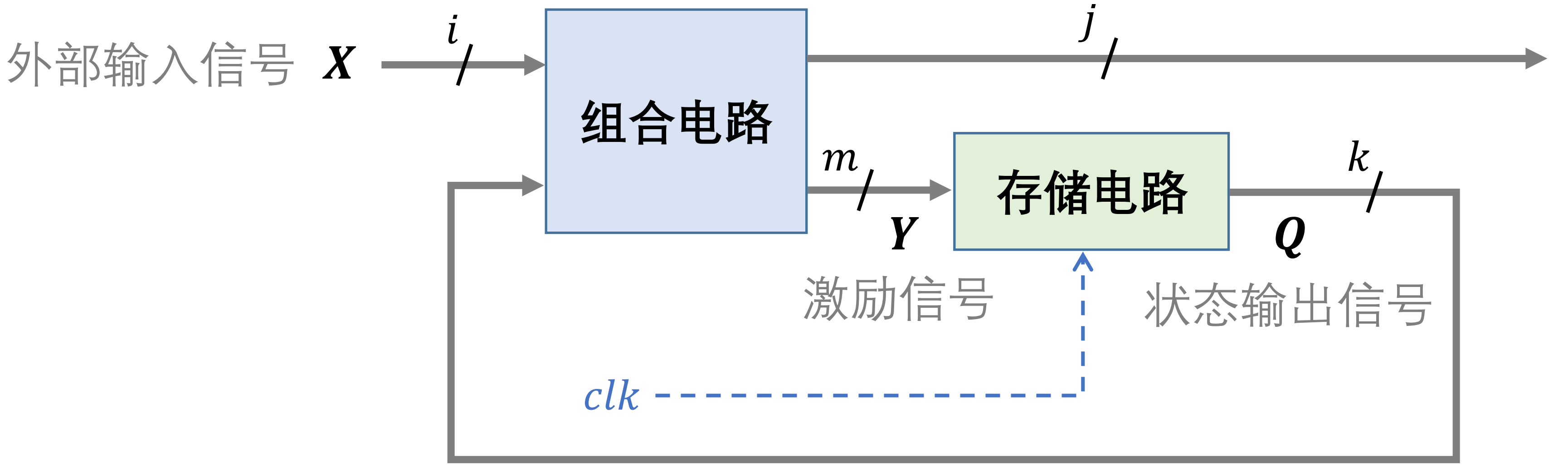
时序逻辑电路 的 **基本结构**



$$\begin{array}{rcccc l}
 & 0 & 1 & 1 & 0 & \\
+ & 0 & 0 & 1 & 1 & \\
\hline
 & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{S} \\
 & 0 & 1 & 1 & 0 & \mathrm{C}
\end{array}$$

←

每个时钟，从低位开始逐位开始计算



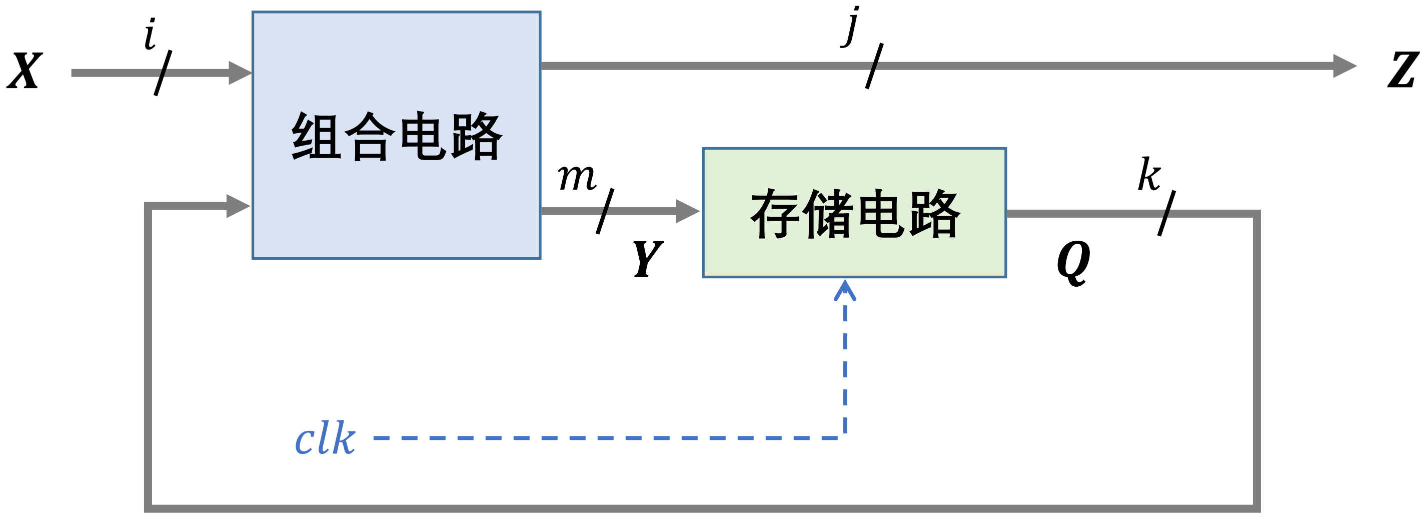
$Z^n = f(X^n, Q^n)$ 输出方程

$Y^n = g(X^n, Q^n)$ 激励方程

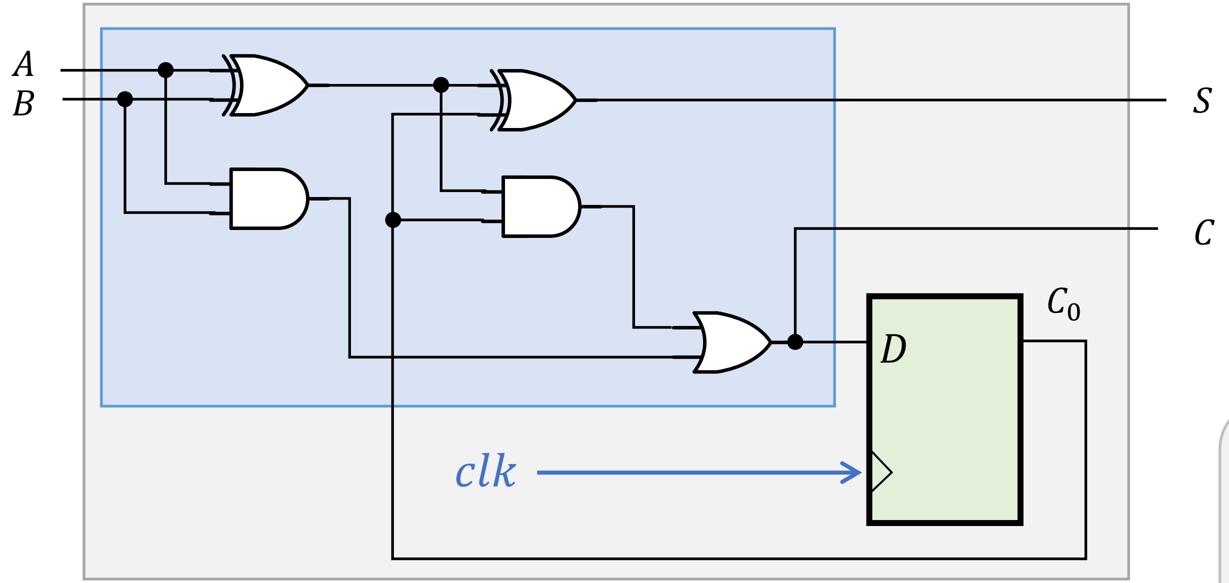
$Q^{n+1} = h(X^n, Q^n)$ 状态方程

组合电路、时序电路

- **组合逻辑电路**：任一时刻的**输出**仅与该时刻**输入**变量的取值有关，而与输入变量的历史情况无关。
- **时序逻辑电路**：任一时刻的**输出**不仅与该时刻**输入**变量的取值有关，而且与电路的**原状态**，即过去的输入情况有关。



时序逻辑电路**分析**



输出方程： $S = A \oplus B \oplus \boldsymbol{C_0}$

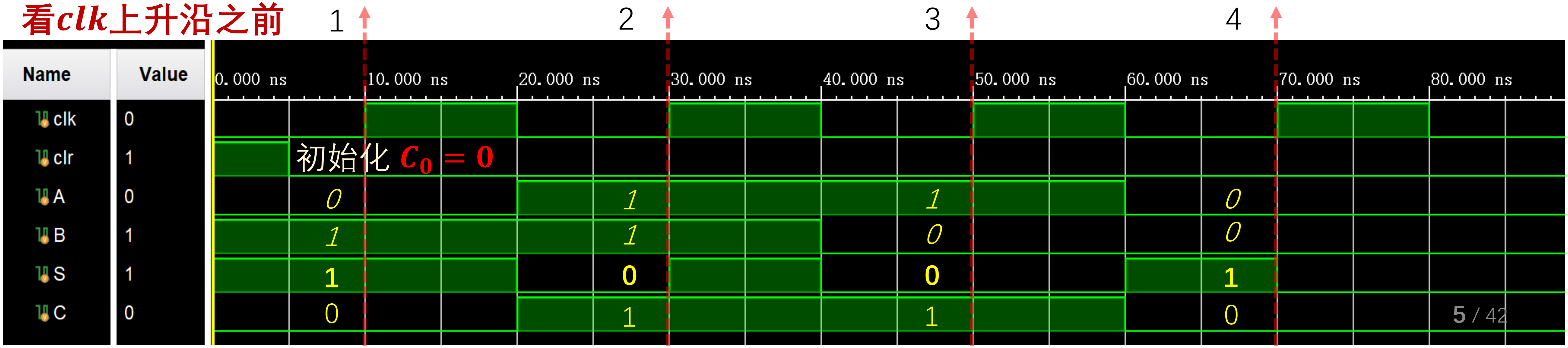
激励方程： $C = (A \oplus B)\,\boldsymbol{C_0} + AB$

状态方程： $\boldsymbol{C_0^{n+1}} = C$　　　$\boldsymbol{Q^{n+1} = D}$

$= (A \oplus B)\,\boldsymbol{C_0} + AB$

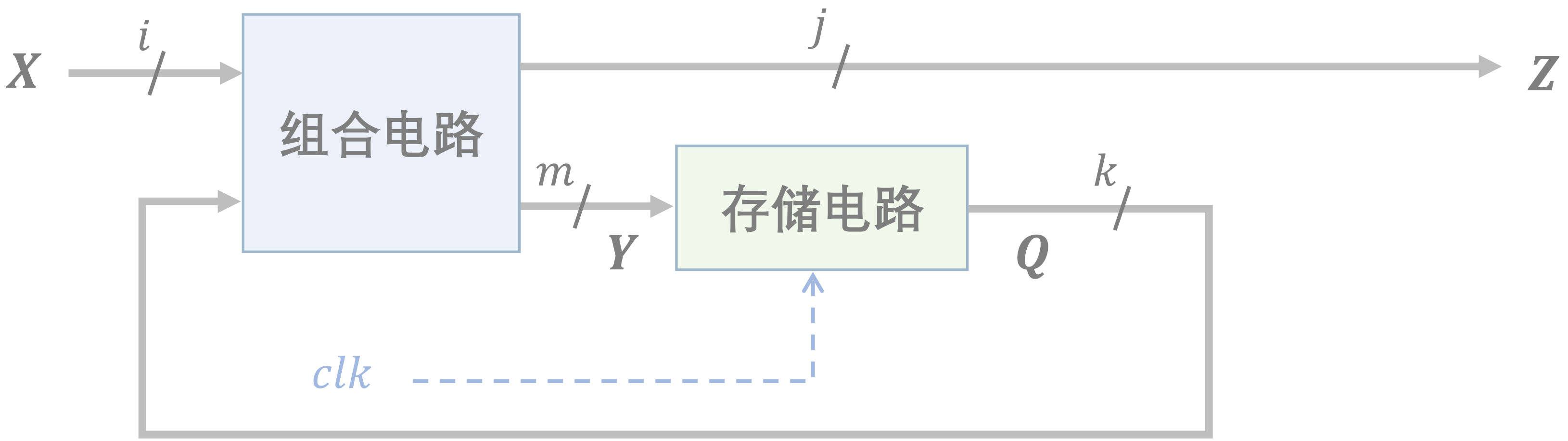
	0	1	1	0	A
+	0	0	1	1	B
	1	**0**	**0**	**1**	**S**
	0	1	1	0	C

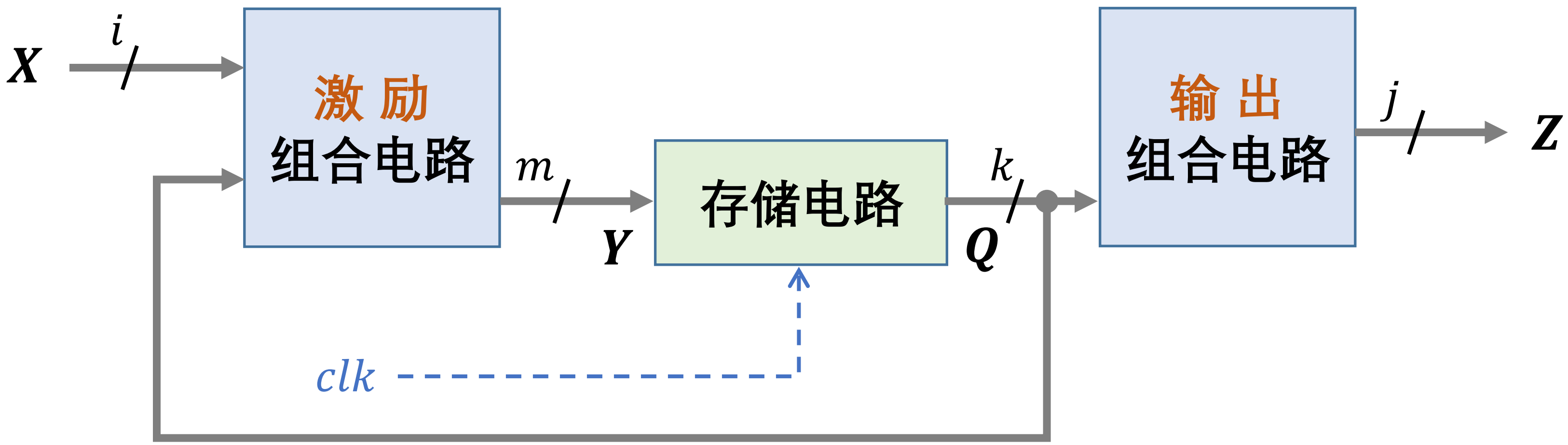
看clk上升沿之前



Name	Value
clk	0
clr	1
A	0
B	1
S	1
C	0

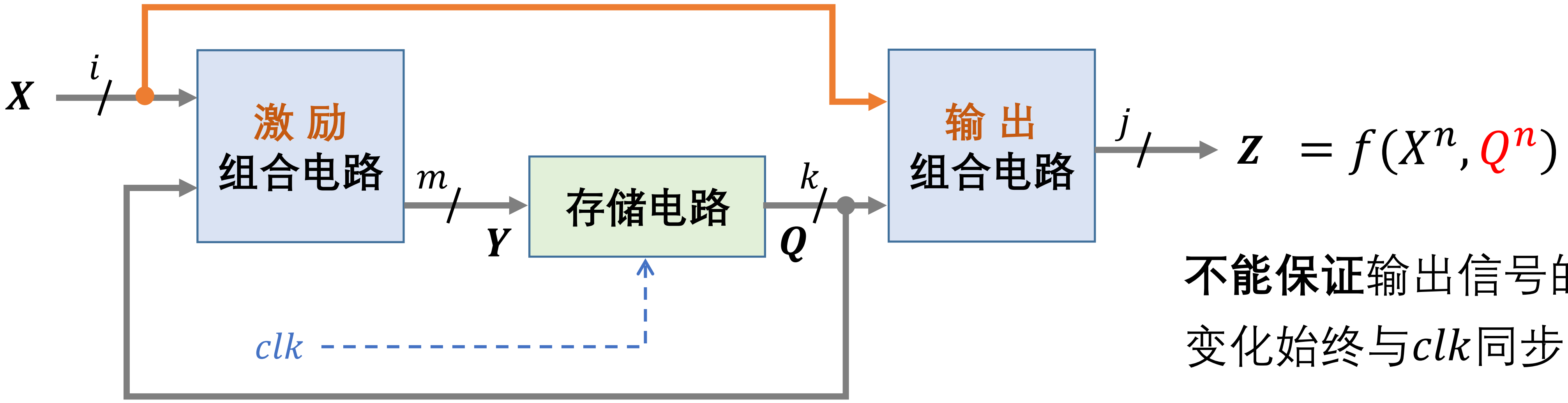
时序逻辑电路 的 **通用结构**





从输出Z看...

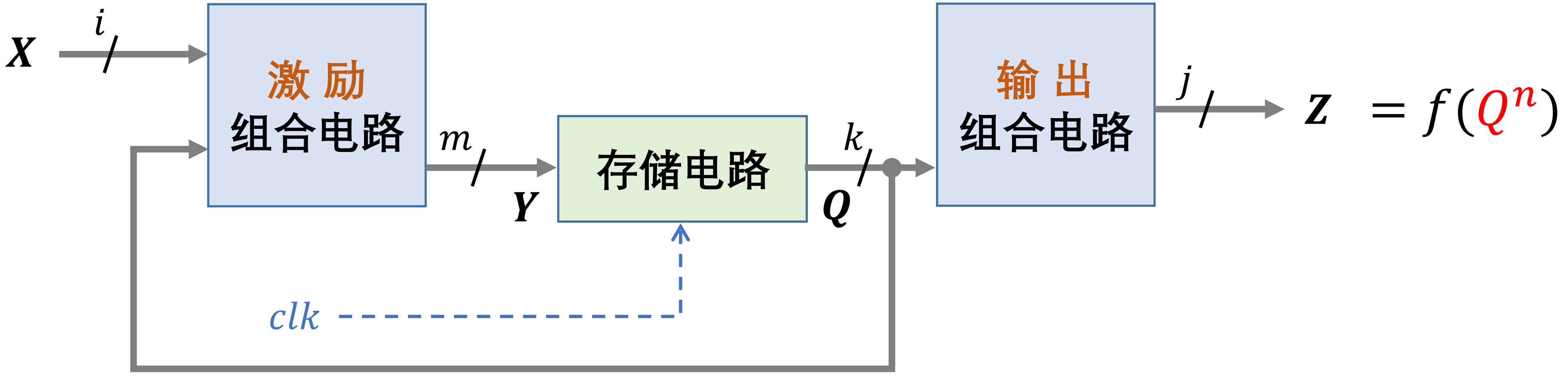
Mealy



$Z = f(X^n, Q^n)$

不能保证输出信号的变化始终与clk同步.

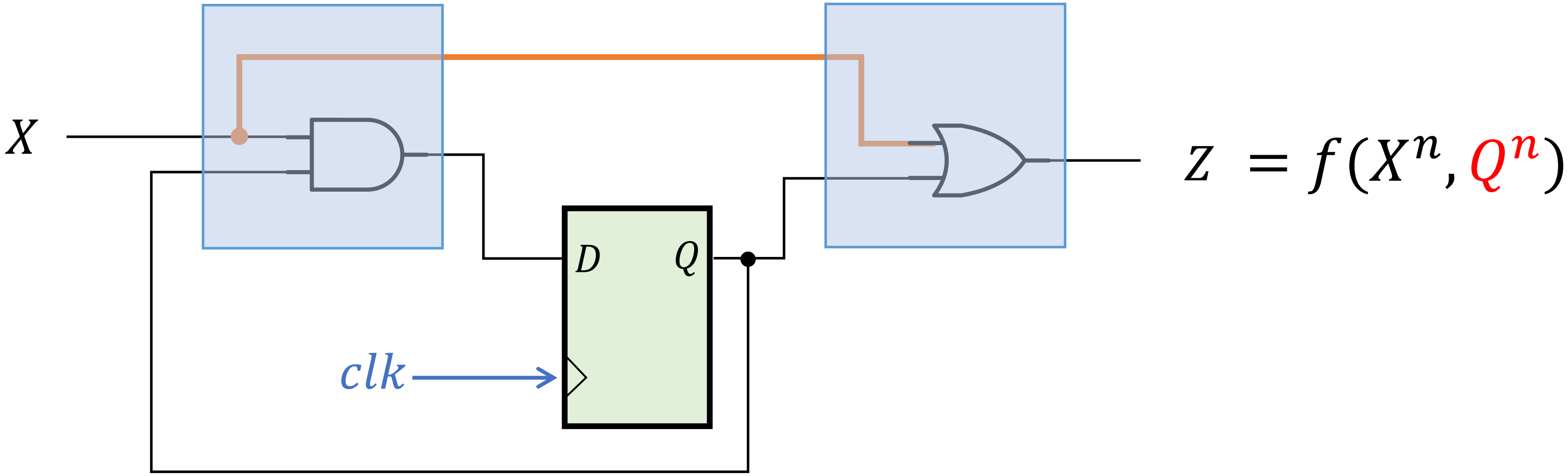
Moore



$Z = f(Q^n)$

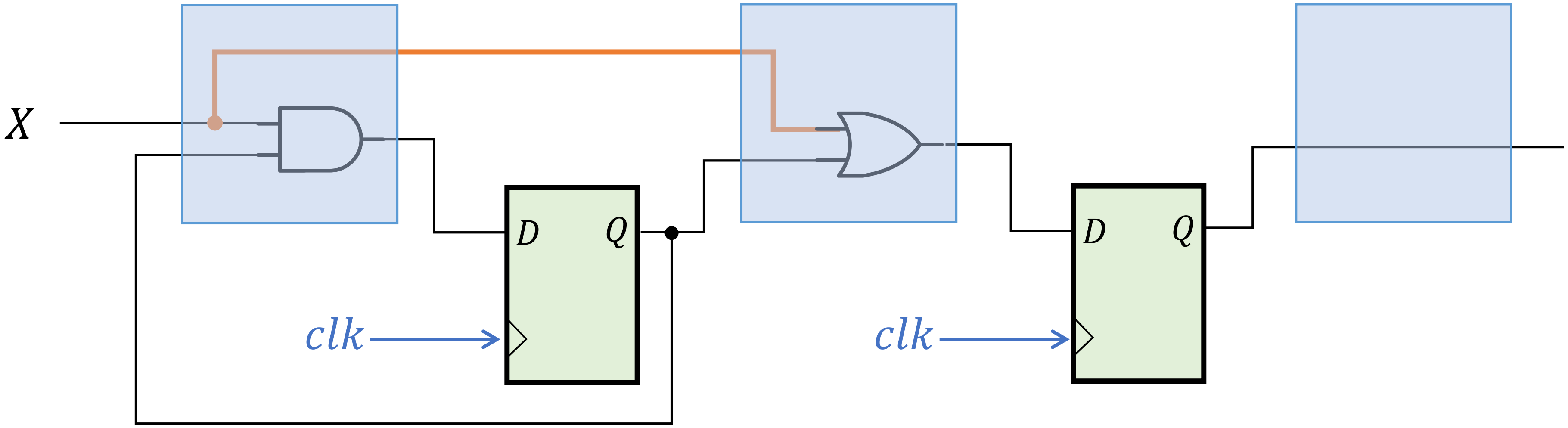
Mealy 变 Moore

Mealy



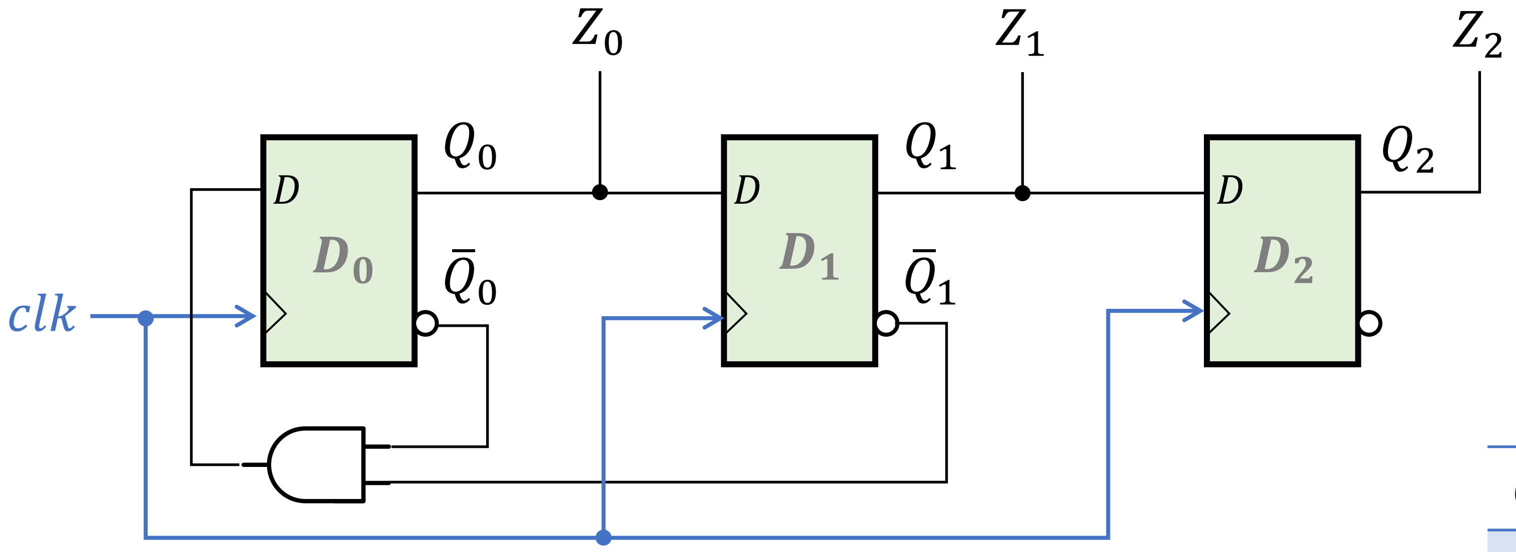
$Z = f(X^n, Q^n)$

Moore



$Z = f(Q^n)$

【例1】同步时序电路分析



激励方程：$D_2 = Q_1$，$D_1 = Q_0$，$D_0 = \bar{Q}_1\bar{Q}_0$

Moore

特征方程：$Q^* = D$

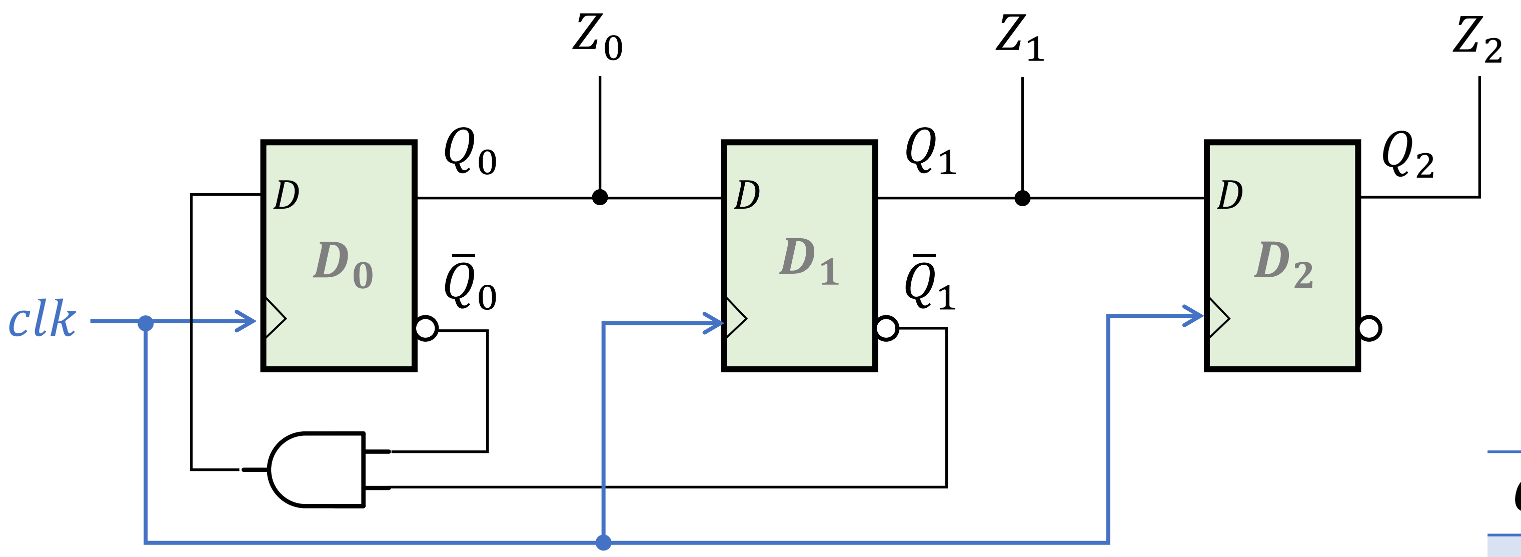
没有输入

状态方程：$Q_2^* = Q_1$，$Q_1^* = Q_0$，$Q_0^* = \bar{Q}_1\bar{Q}_0$

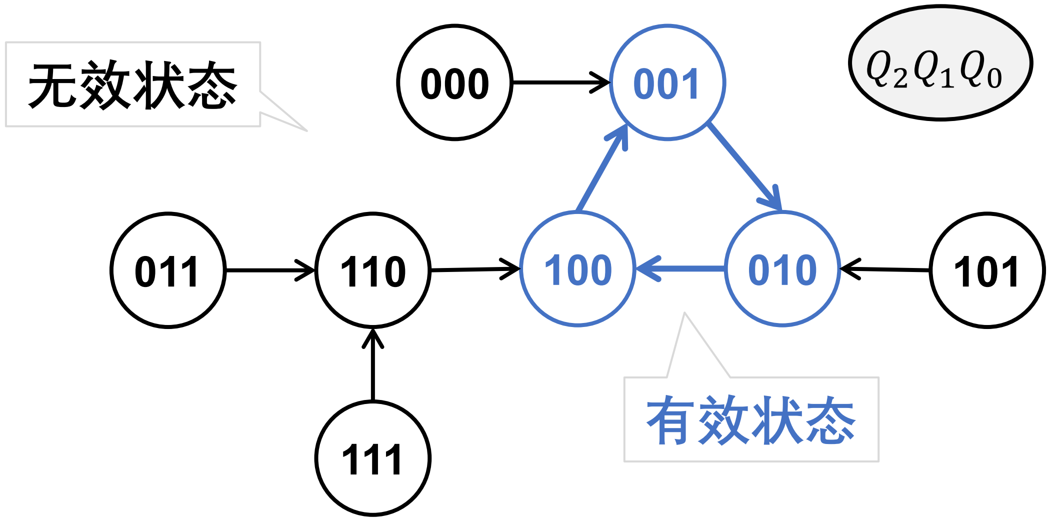
输出方程：$Z_2 = Q_2$，$Z_1 = Q_1$，$Z_0 = Q_0$

$Q_2Q_1Q_0$	$Q_2^*Q_1^*Q_0^*$
0 0 0	**0 0 1**
0 0 1	**0 1 0**
0 1 0	**1 0 0**
0 1 1	**1 1 0**
1 0 0	**0 0 1**
1 0 1	**0 1 0**
1 1 0	**1 0 0**
1 1 1	**1 1 0**

【例1】同步时序电路分析



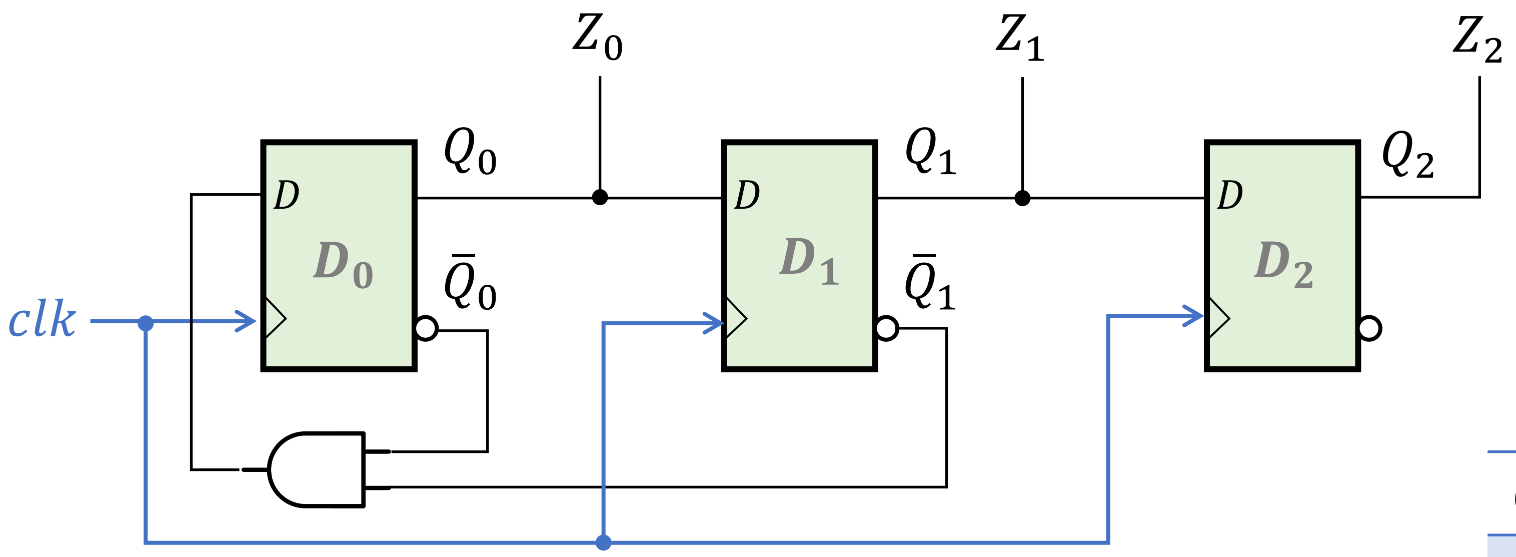
$Q_2Q_1Q_0$	$Q_2^*Q_1^*Q_0^*$
0 0 0	0 0 1
0 0 1	0 1 0
0 1 0	1 0 0
0 1 1	1 1 0
1 0 0	0 0 1
1 0 1	0 1 0
1 1 0	1 0 0
1 1 1	1 1 0

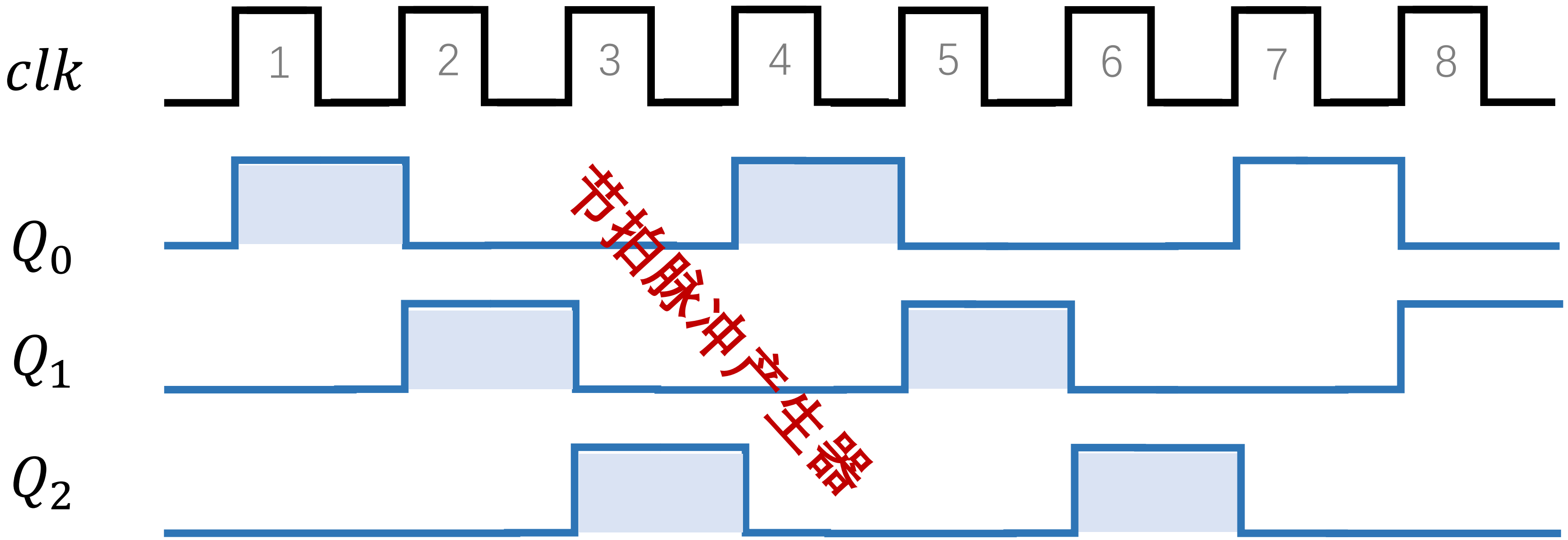


自启动电路

如果电路处于任何一个无效状态，经过若干时钟脉冲后，都可以**自动**进入有效状态的电路。

【例1】同步时序电路分析



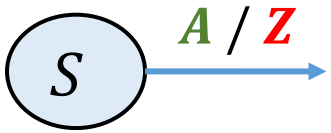


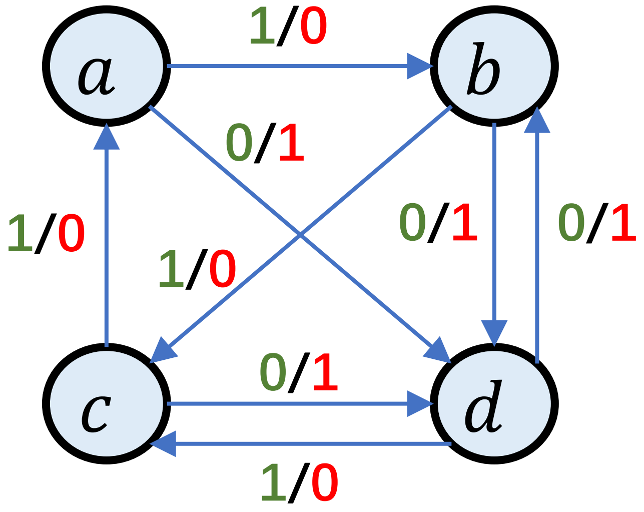
节拍脉冲产生器

$Q_2Q_1Q_0$	$Q_2^*Q_1^*Q_0^*$
0 0 0	0 0 1
0 0 1	0 1 0
0 1 0	1 0 0
0 1 1	1 1 0
1 0 0	0 0 1
1 0 1	0 1 0
1 1 0	1 0 0
1 1 1	1 1 0

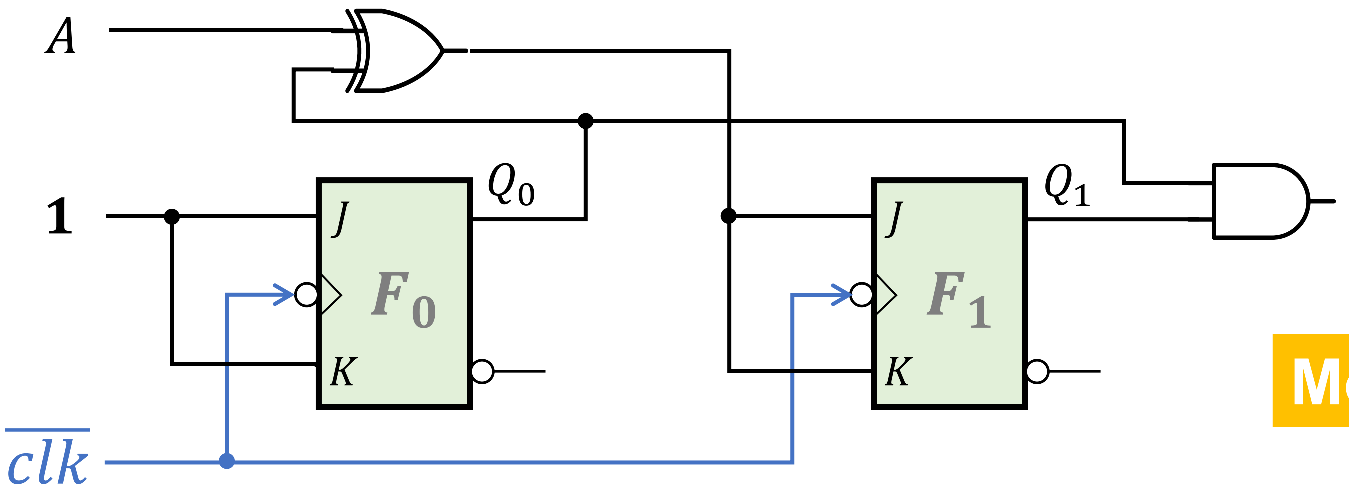
【练习1】状态表 => 状态图

S	S^*/Z	
	$A=0$	$A=1$
a	d / 1	b / 0
b	d / 1	c / 0
c	d / 1	a / 0
d	b / 1	c / 0





【例2】JK触发器电路分析：公式法



Moore

Q_1Q_0	$Q_1^*Q_0^*$		Z
	$A=0$	$A=1$	
0 0	0 1	1 1	0
0 1	1 0	0 0	0
1 0	1 1	0 1	0
1 1	0 0	1 0	1

激励方程：$J_0 = K_0 = 1$

$J_1 = K_1 = A \oplus Q_0$

特征方程：$Q^* = J\bar{Q} + \bar{K}Q$

状态方程：$Q_0^* = J_0\bar{Q}_0 + \bar{K}_0Q_0 = \bar{Q}_0$，$Q_1^* = J_1\bar{Q}_1 + \bar{K}_1Q_1$

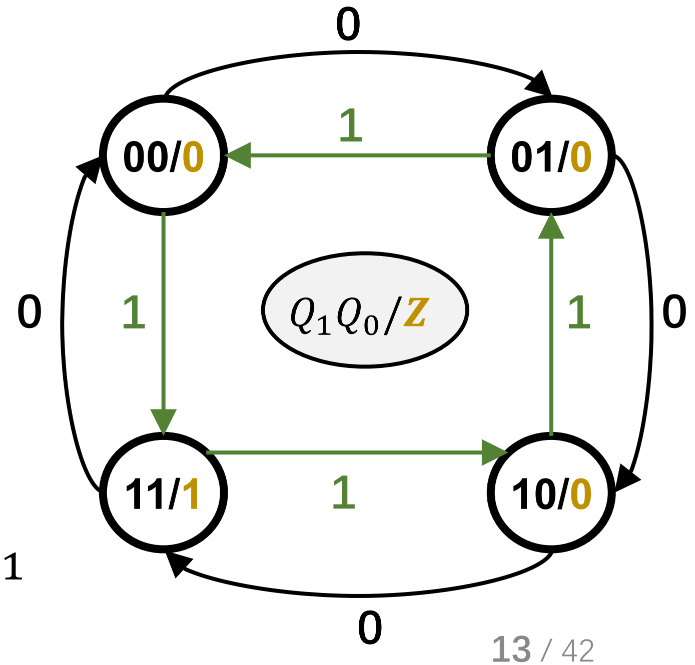
$= (A \oplus Q_0)\bar{Q}_1 + \overline{A \oplus Q_0}Q_1$

$= A \oplus Q_0 \oplus Q_1$

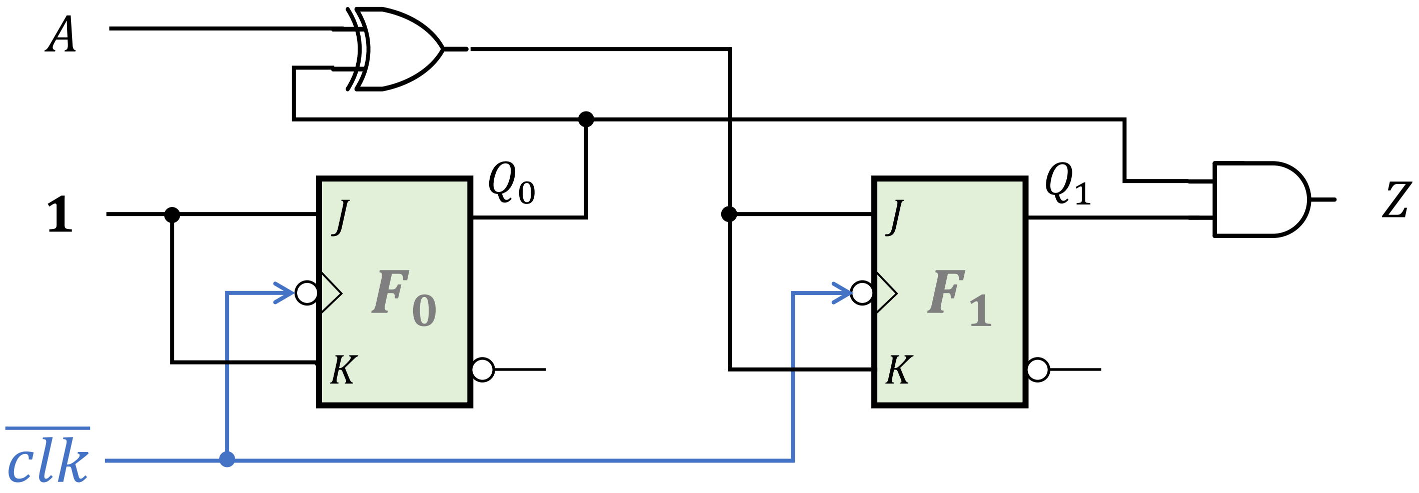
输出方程：$Z = Q_1Q_0$

$A = 0$ 时，递增计数

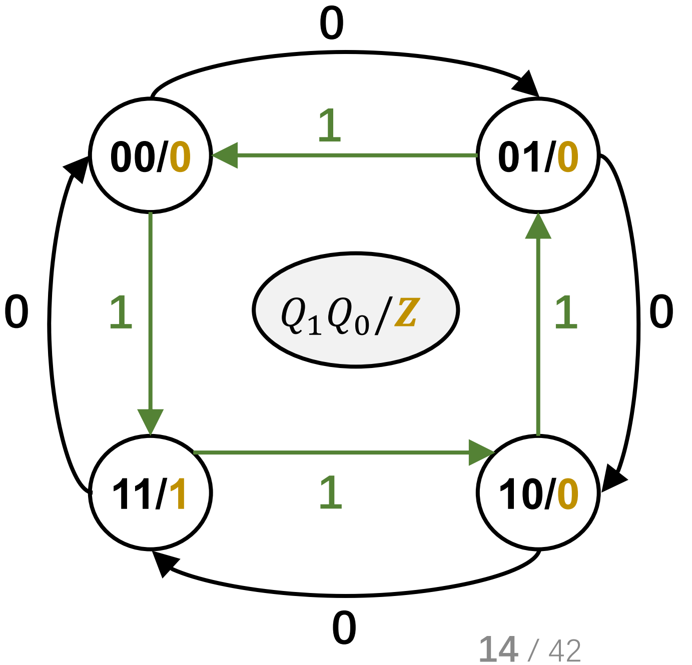
$A = 1$ 时，递减计数



【例2】JK触发器电路分析：特征表法



Q_1Q_0	$Q_1^*Q_0^*$		Z
	$A=0$	$A=1$	
0 0	0 1	1 1	0
0 1	1 0	0 0	0
1 0	1 1	0 1	0
1 1	0 0	1 0	1



激励方程：$J_0 = K_0 = 1$

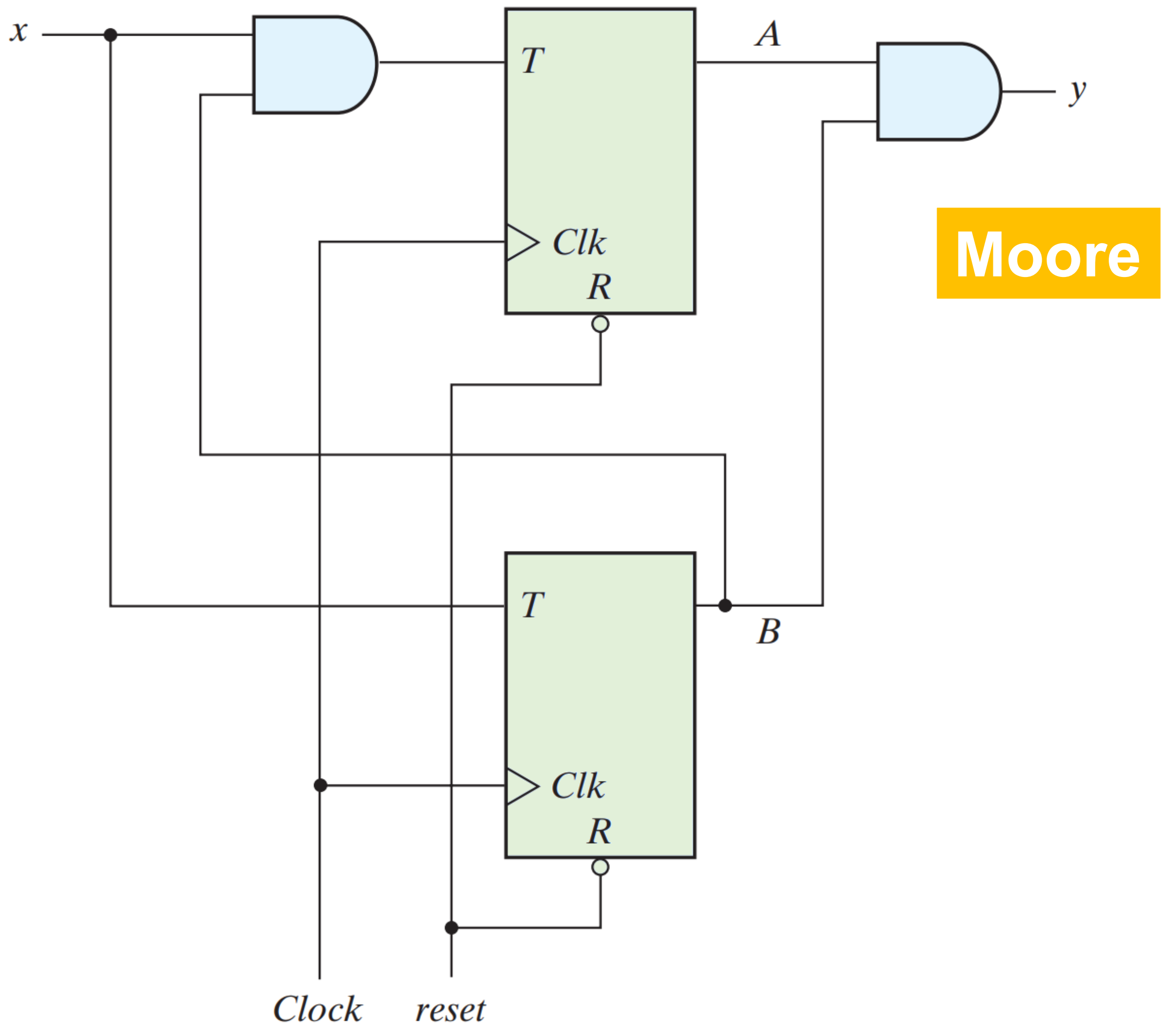
$$J_1 = K_1 = A \oplus Q_0$$

输出方程：$Z = Q_1Q_0$

特征表

J	K	Q^*	说明
0	0	Q	保持
1	0	1	置 1
0	1	0	置 0
1	1	$\overline{Q}$	翻转

【例3】T触发器电路分析



Moore

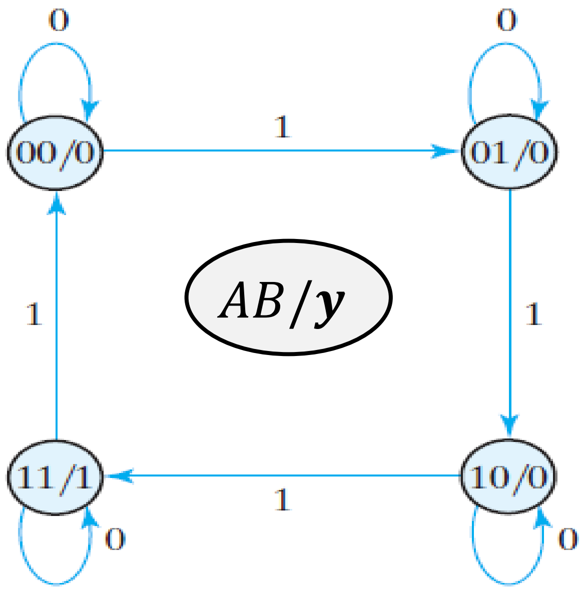
激励方程： $T_A = Bx \qquad T_B = x$

特征方程： $Q^* = T \oplus Q$

状态方程： $A^* = T_A \oplus A = Bx \oplus A$

$B^* = T_B \oplus B = x \oplus B$

输出方程： $y = AB$



时序电路分析步骤

（对同步、异步时序电路都适用）

① 分析**电路组成**（组合？时序？Moore? Mealy?）

② 写出**激励方程**、**输出方程**

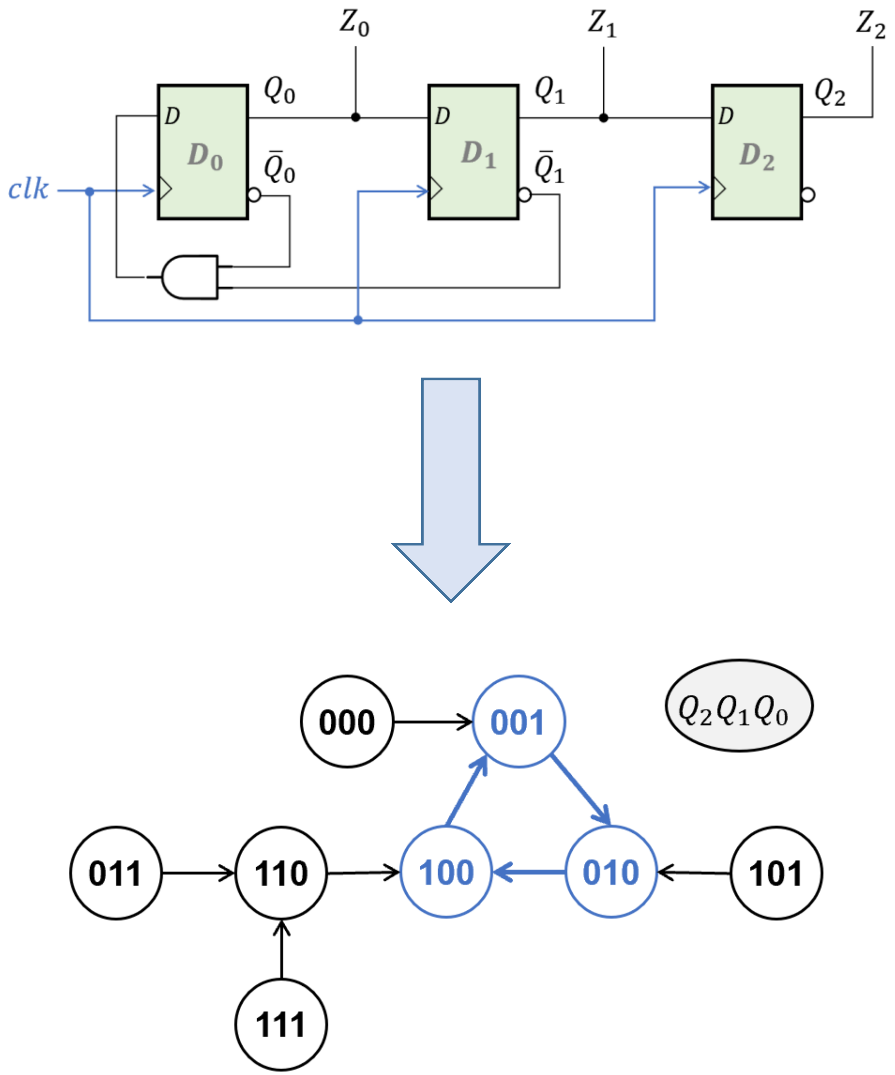
③ 写出**状态方程 / 特征表**

④ 画出**状态表**、**状态图**、**(波形图)**

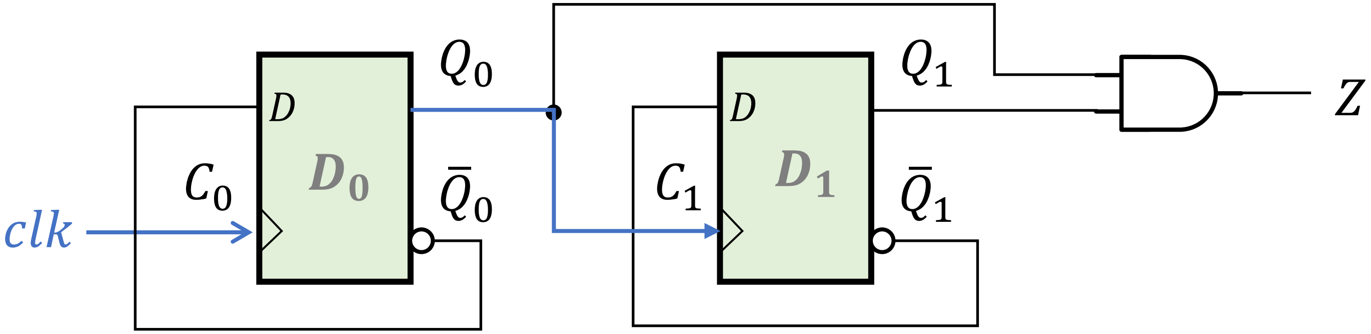
⑤ 分析**输出序列**、**输入序列**的关系，

说明时序电路的逻辑功能。

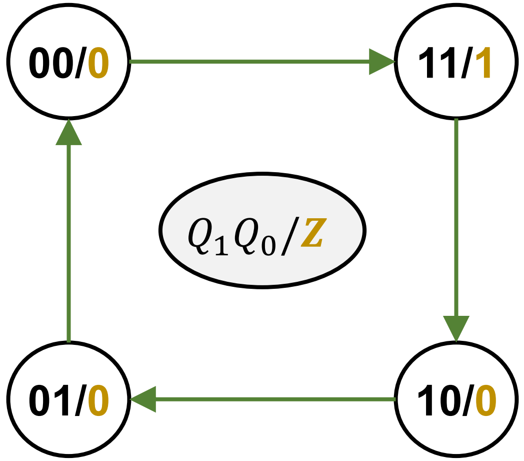
⑥ 评估、改进电路



【例4】异步时序电路分析*



因两个D触发器状态翻转存在延迟，故电路存在短暂的不确定状态。



时钟方程：$C_0 = clk\uparrow$ ，$C_1 = Q_0\uparrow$

即Q_0由0变1

激励方程：$D_0 = \bar{Q}_0$，$D_1 = \bar{Q}_1$

特征方程：$Q^* = D$

上升沿才能转换状态，其余时间保持原态。

状态方程：$Q_0^* = D_0 C_0 + Q_0 \bar{C}_0 = \bar{Q}_0\uparrow + Q_0\downarrow$ $\quad Q_1^* = D_1 C_1 + Q_1 \bar{C}_1 = \bar{Q}_1\uparrow + Q_1\downarrow$

输出方程：$Z = Q_1 Q_0$

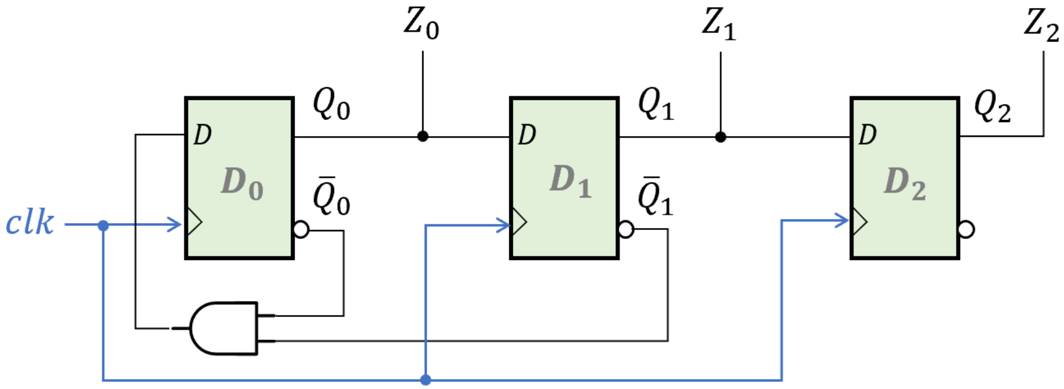
Q_1	Q_0	C_1	C_0	Q_1^*	Q_0^*	Z
0	0	↑	↑	1	1	0
0	1	↓	↑	0	0	0
1	0	↑	↑	0	1	0
1	1	↓	↑	1	0	1

同步时序电路、异步时序电路

时序电路

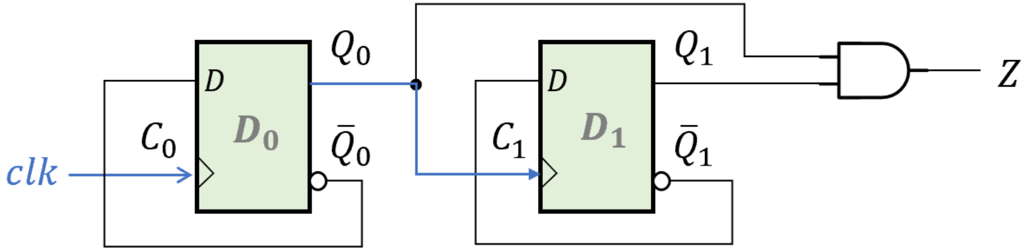
- **同步时序电路**：各个触发器的时钟脉冲相同。一般用触发器实现。

  在非时钟有效沿期间，触发器处于保持状态。



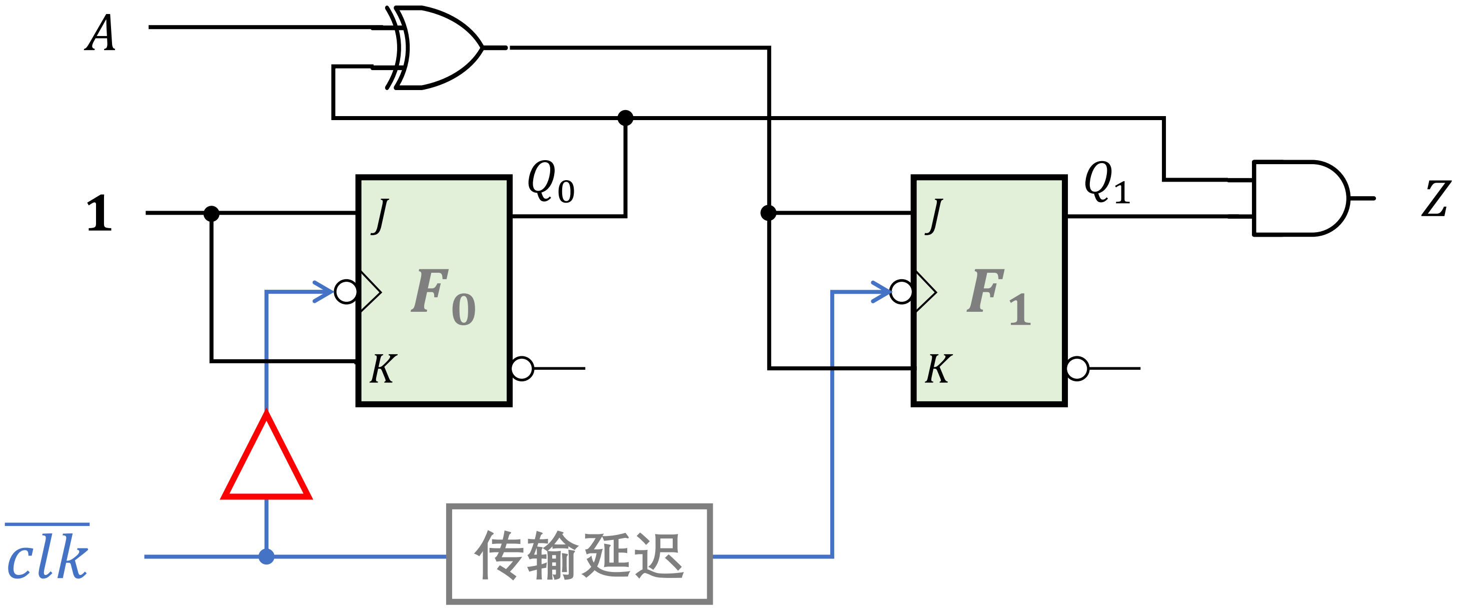
- **异步时序电路**：各个触发器的时钟脉冲不同。既可用触发器、也可用锁存器。

  电路状态的翻转有先有后。



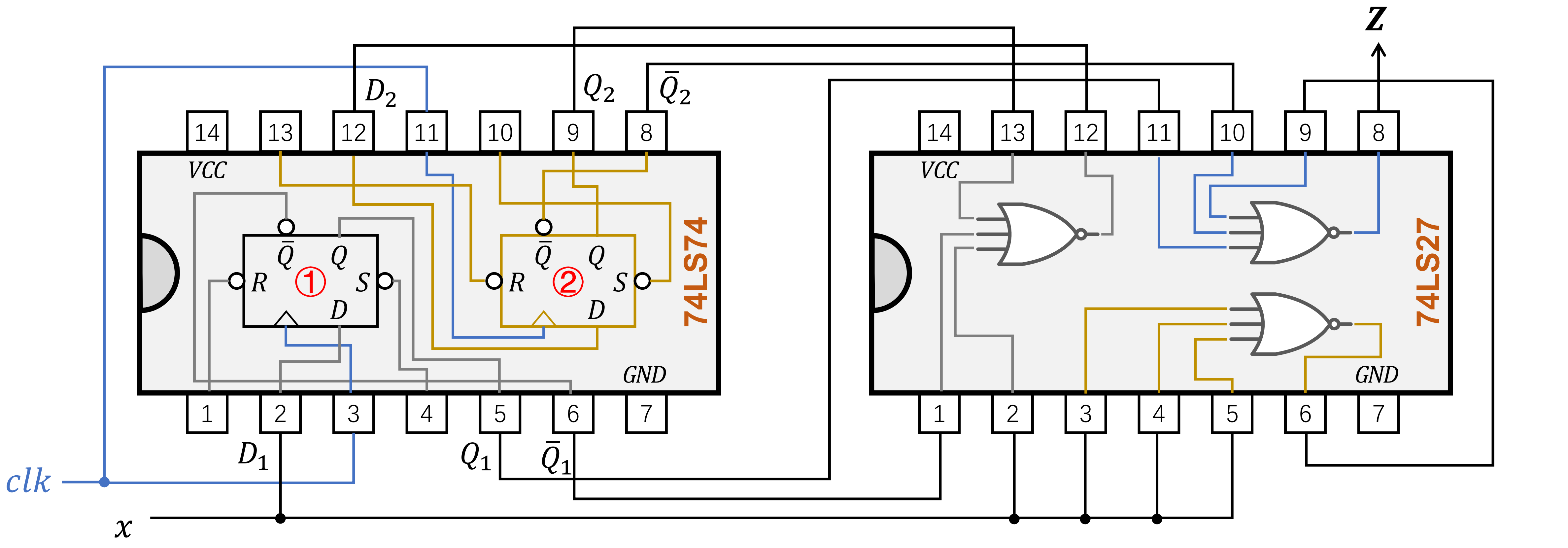
同步时序逻辑电路中的**时钟偏移**

从同一时钟源出发的时钟脉冲，通过不同路径到达每个触发器的时间不同而产生的偏差。



解决方法：增加缓冲器、对称布局……

【例5】电路分析



$$Z = \overline{\bar{x} + \bar{Q}_2 + Q_1} = xQ_2\bar{Q}_1$$

$$Q_1^* = D_1 = x$$

Mealy

$$Q_2^* = D_2 = \overline{x + Q_2 + \bar{Q}_1} = \bar{x}\,\bar{Q}_2\,Q_1$$

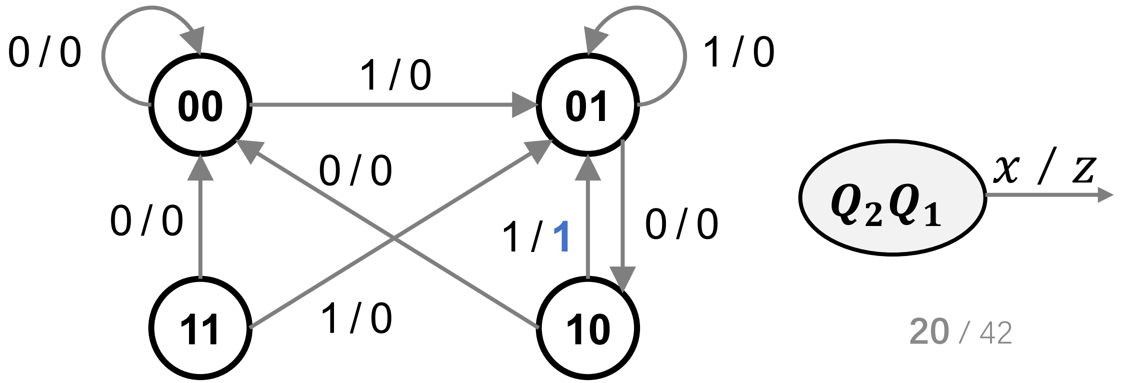
状态表

....

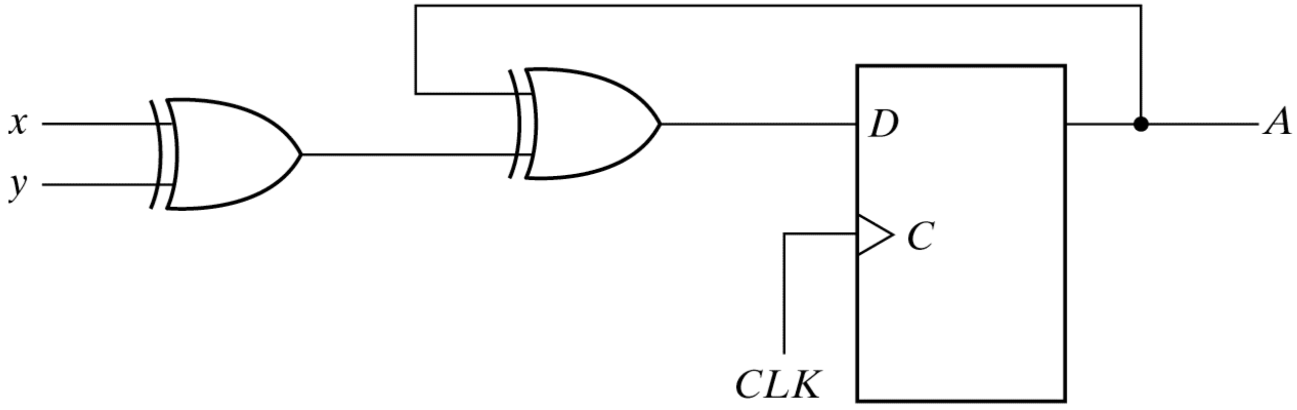
....

....

....



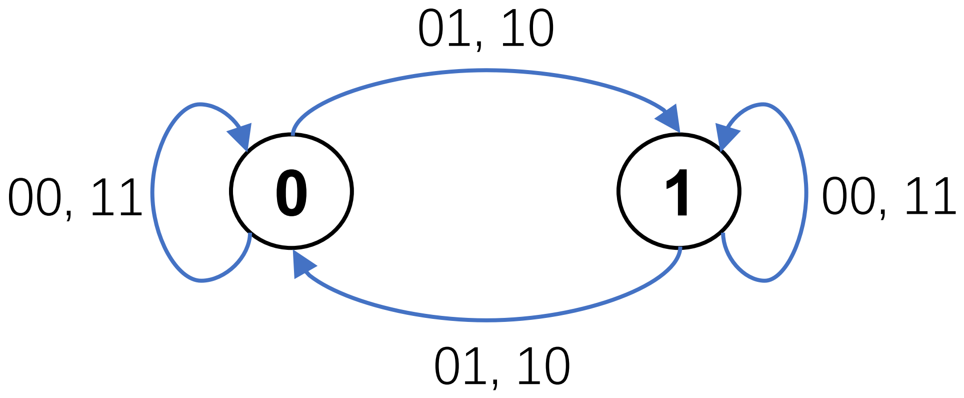
【练习2】时序电路分析



Present state	Inputs		Next state
A	x	y	A^*
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

激励方程：$D = A \oplus x \oplus y$

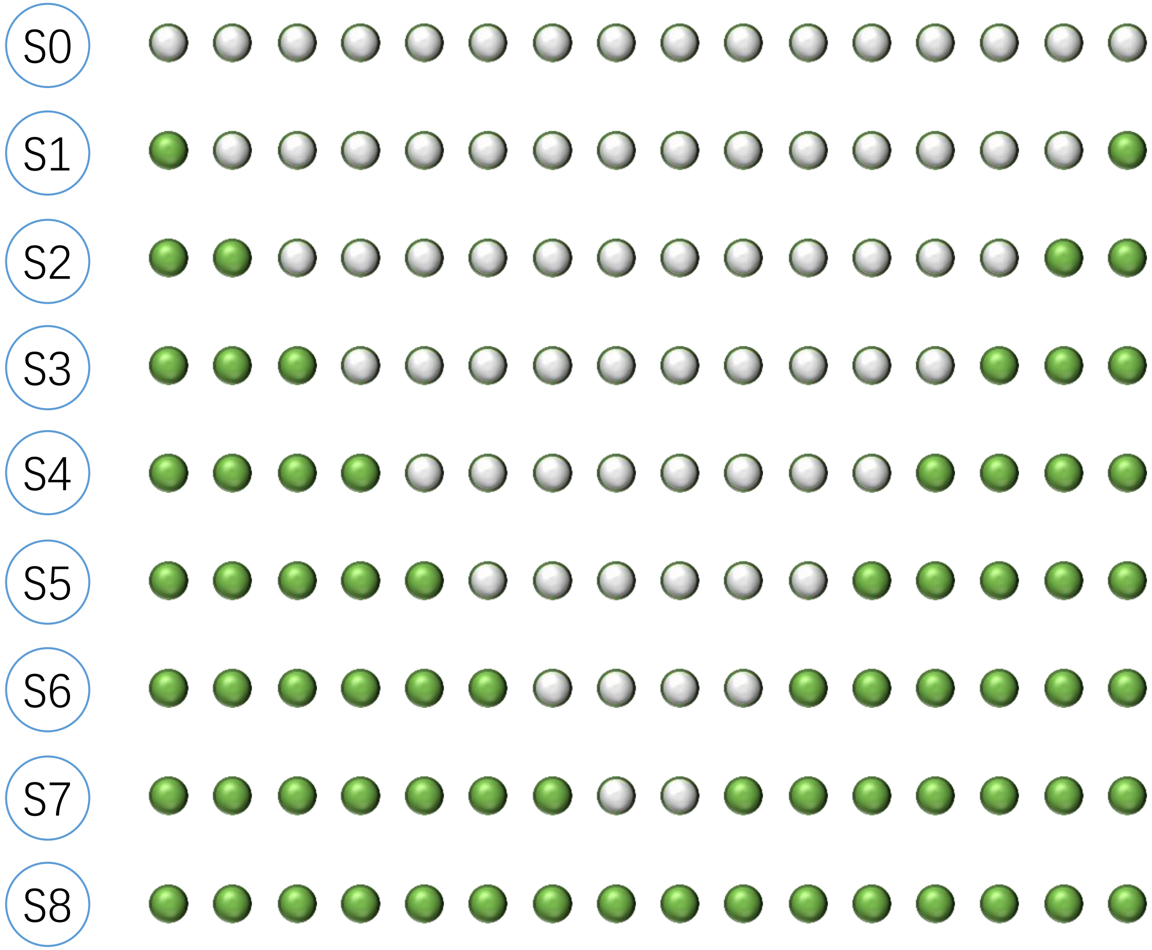
状态方程：$A^* = A \oplus x \oplus y$



2

FSM

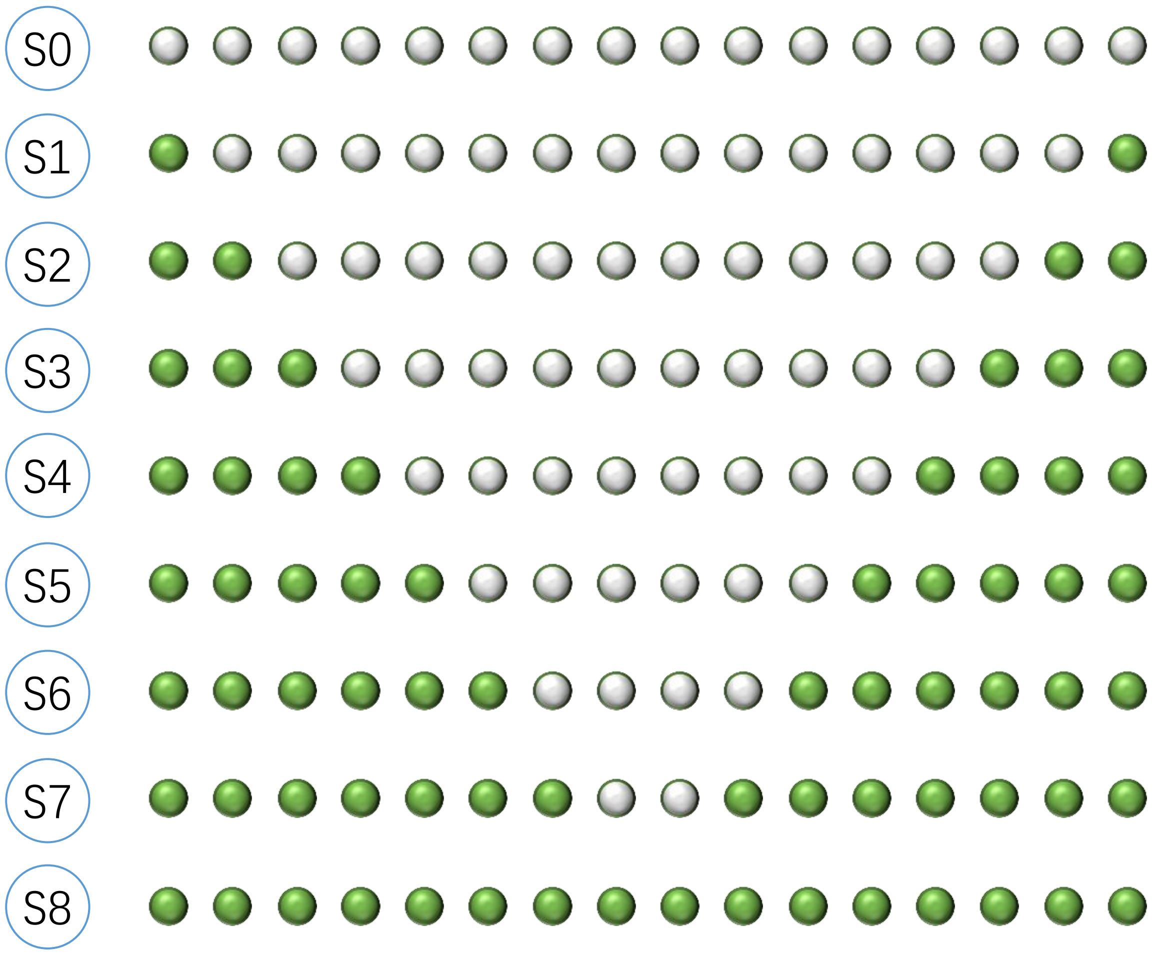
流水灯

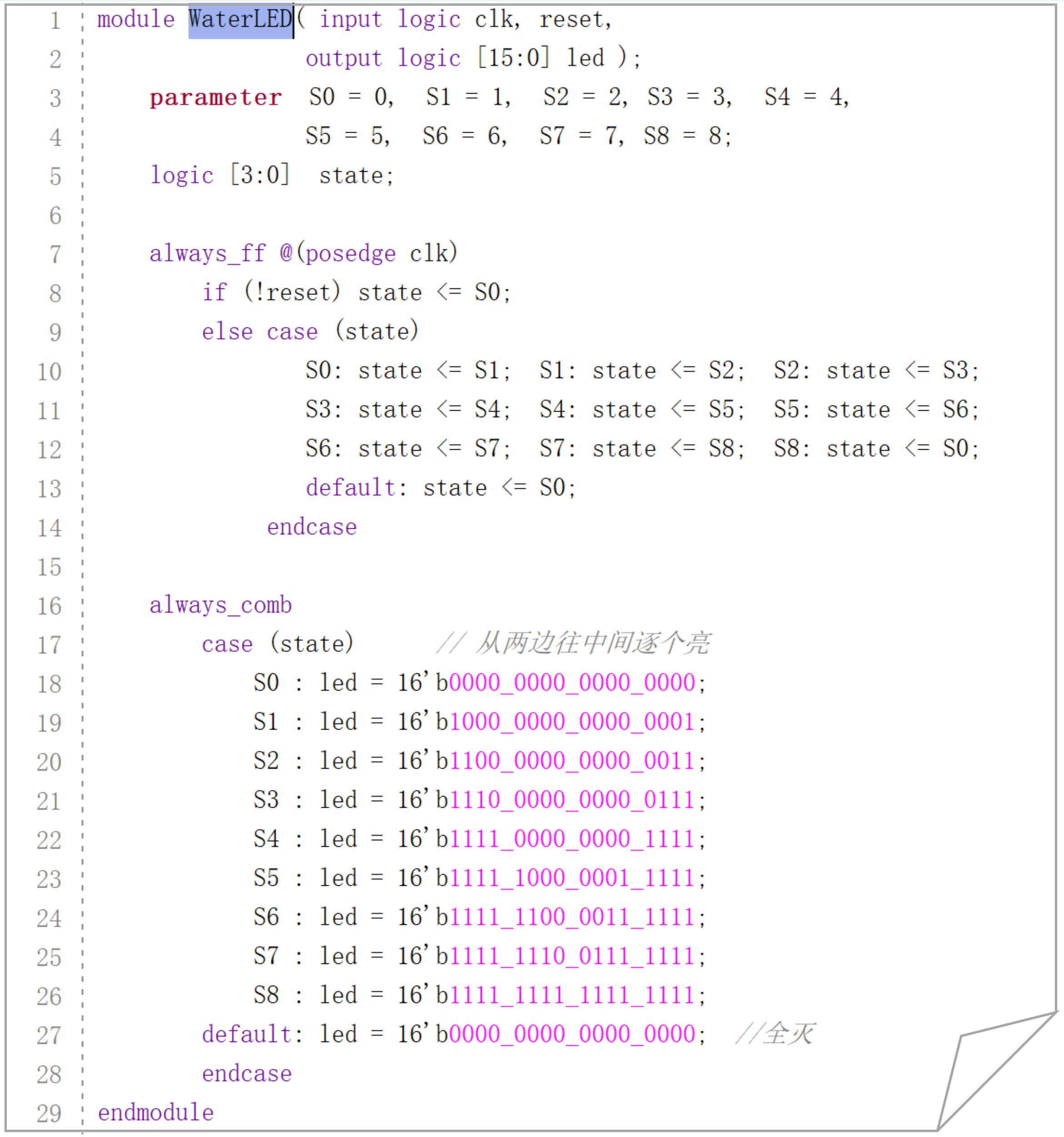


S0	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
S1	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●
S2	●	●	○	○	○	○	○	○	○	○	○	○	○	○	●	●
S3	●	●	●	○	○	○	○	○	○	○	○	○	○	●	●	●
S4	●	●	●	●	○	○	○	○	○	○	○	○	●	●	●	●
S5	●	●	●	●	●	○	○	○	○	○	○	●	●	●	●	●
S6	●	●	●	●	●	●	○	○	○	○	●	●	●	●	●	●
S7	●	●	●	●	●	●	●	○	○	●	●	●	●	●	●	●
S8	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●

WaterLED

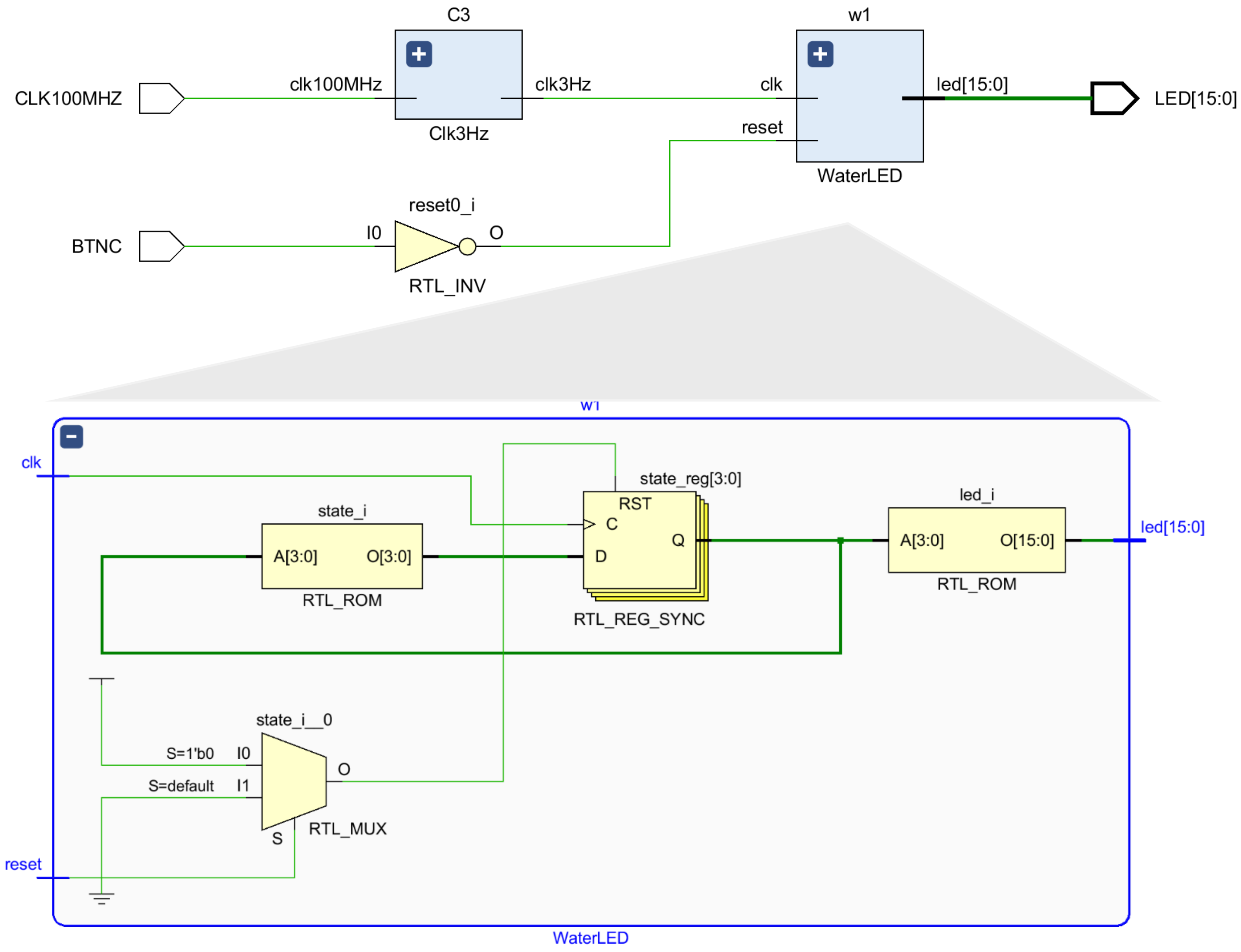
流水灯

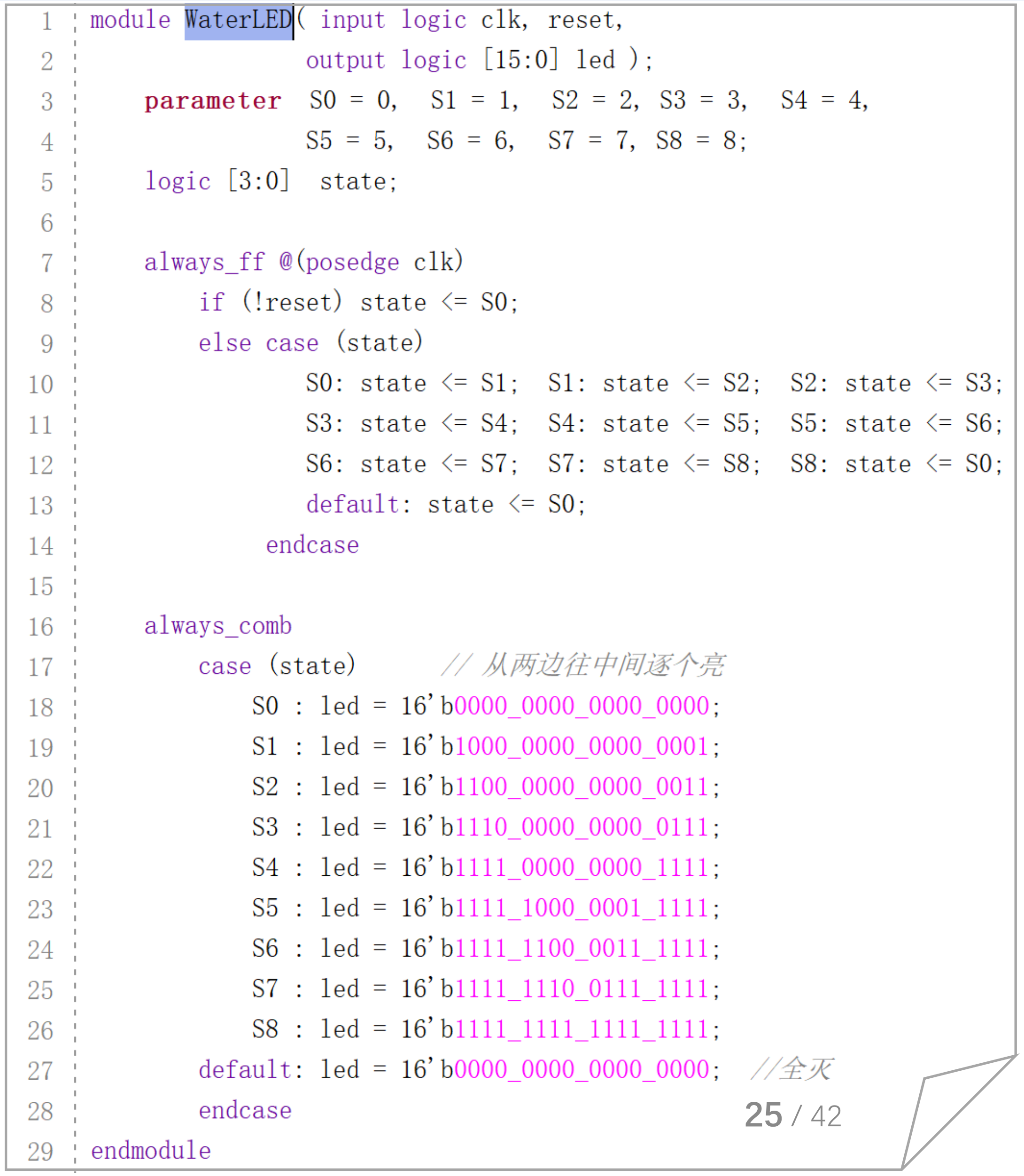




```
module WaterLED( input logic clk, reset,
                output logic [15:0] led );
    parameter  S0 = 0,  S1 = 1,  S2 = 2,  S3 = 3,  S4 = 4,
               S5 = 5,  S6 = 6,  S7 = 7,  S8 = 8;
    logic [3:0]  state;

    always_ff @(posedge clk)
        if (!reset) state <= S0;
        else case (state)
                S0: state <= S1;  S1: state <= S2;  S2: state <= S3;
                S3: state <= S4;  S4: state <= S5;  S5: state <= S6;
                S6: state <= S7;  S7: state <= S8;  S8: state <= S0;
                default: state <= S0;
             endcase

    always_comb
        case (state)      // 从两边往中间逐个亮
            S0 : led = 16'b0000_0000_0000_0000;
            S1 : led = 16'b1000_0000_0000_0001;
            S2 : led = 16'b1100_0000_0000_0011;
            S3 : led = 16'b1110_0000_0000_0111;
            S4 : led = 16'b1111_0000_0000_1111;
            S5 : led = 16'b1111_1000_0001_1111;
            S6 : led = 16'b1111_1100_0011_1111;
            S7 : led = 16'b1111_1110_0111_1111;
            S8 : led = 16'b1111_1111_1111_1111;
        default: led = 16'b0000_0000_0000_0000;  //全灭
        endcase
endmodule
```

流水灯 原理图

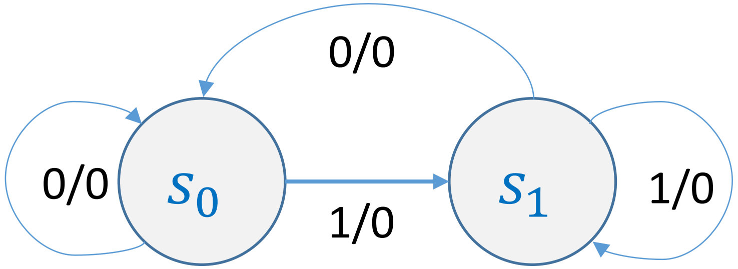




```
module WaterLED( input logic clk, reset,
                 output logic [15:0] led );
   parameter  S0 = 0,  S1 = 1,  S2 = 2, S3 = 3,  S4 = 4,
                 S5 = 5,  S6 = 6,  S7 = 7, S8 = 8;
   logic [3:0]  state;

   always_ff @(posedge clk)
      if (!reset) state <= S0;
      else case (state)
                 S0: state <= S1;  S1: state <= S2;  S2: state <= S3;
                 S3: state <= S4;  S4: state <= S5;  S5: state <= S6;
                 S6: state <= S7;  S7: state <= S8;  S8: state <= S0;
                 default: state <= S0;
              endcase

   always_comb
      case (state)        // 从两边往中间逐个亮
          S0 : led = 16'b0000_0000_0000_0000;
          S1 : led = 16'b1000_0000_0000_0001;
          S2 : led = 16'b1100_0000_0000_0011;
          S3 : led = 16'b1110_0000_0000_0111;
          S4 : led = 16'b1111_0000_0000_1111;
          S5 : led = 16'b1111_1000_0001_1111;
          S6 : led = 16'b1111_1100_0011_1111;
          S7 : led = 16'b1111_1110_0111_1111;
          S8 : led = 16'b1111_1111_1111_1111;
      default: led = 16'b0000_0000_0000_0000;  //全灭
      endcase
endmodule
```

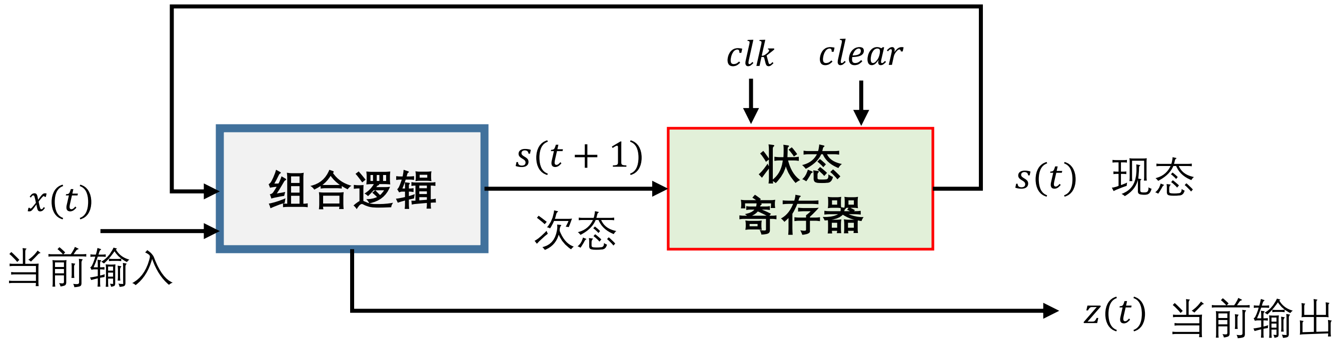
有限状态机 Finite State Machine

- 电路设计**经典方法**，尤其适于设计控制模块，易于FPGA实现。
- 用于：对**有限内部状态相互转换的系统**进行建模。
- 常为同步时序，在时钟信号的触发下完成各状态之间的转换，并产生相应的输出。
- 由**组合逻辑**(*状态译码*、*产生输出信号*) + **时序逻辑**(*存储状态*)构成。
- **状态机**表示方法：**状态图**、状态表、流程图。三者等价。

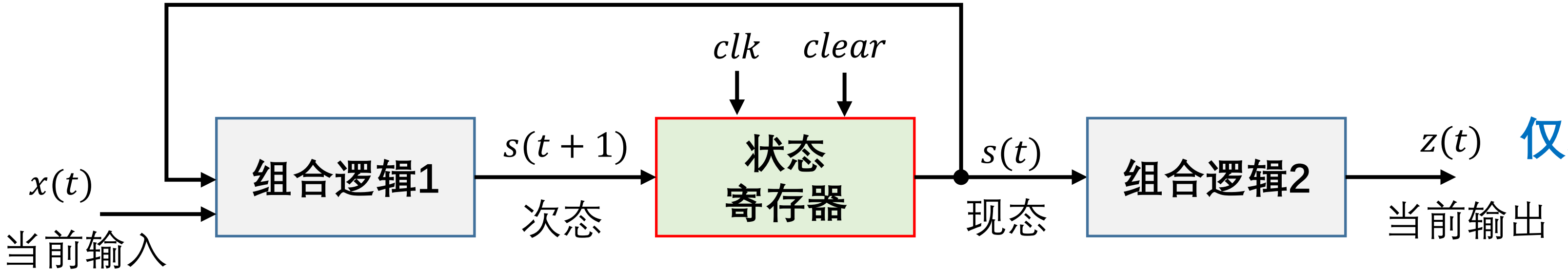


FSM 分类

经典状态机



摩尔状态机



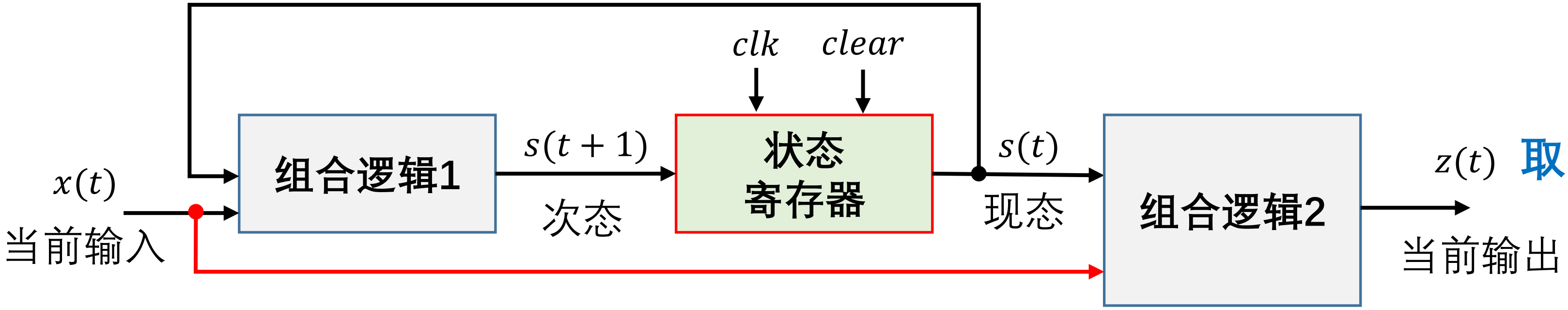
仅取决于：现态$s(t)$

同一系统，

既可以用**摩尔模型**描述

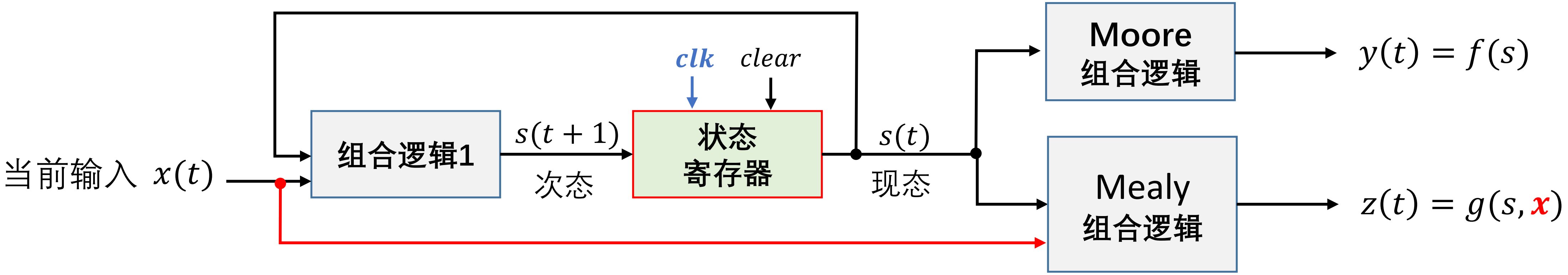
也可以用**米里模型**描述

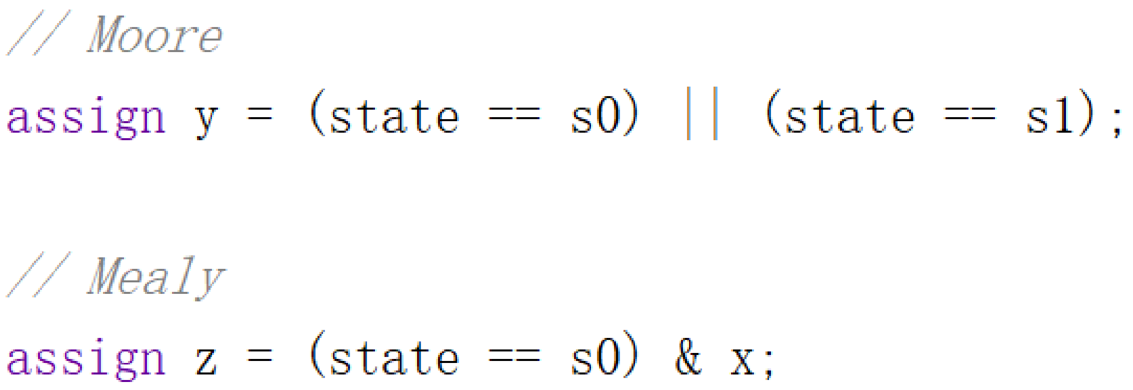
米里状态机



取决于：现态$s(t)$ + 当前输入$x(t)$

复杂系统：混合编程

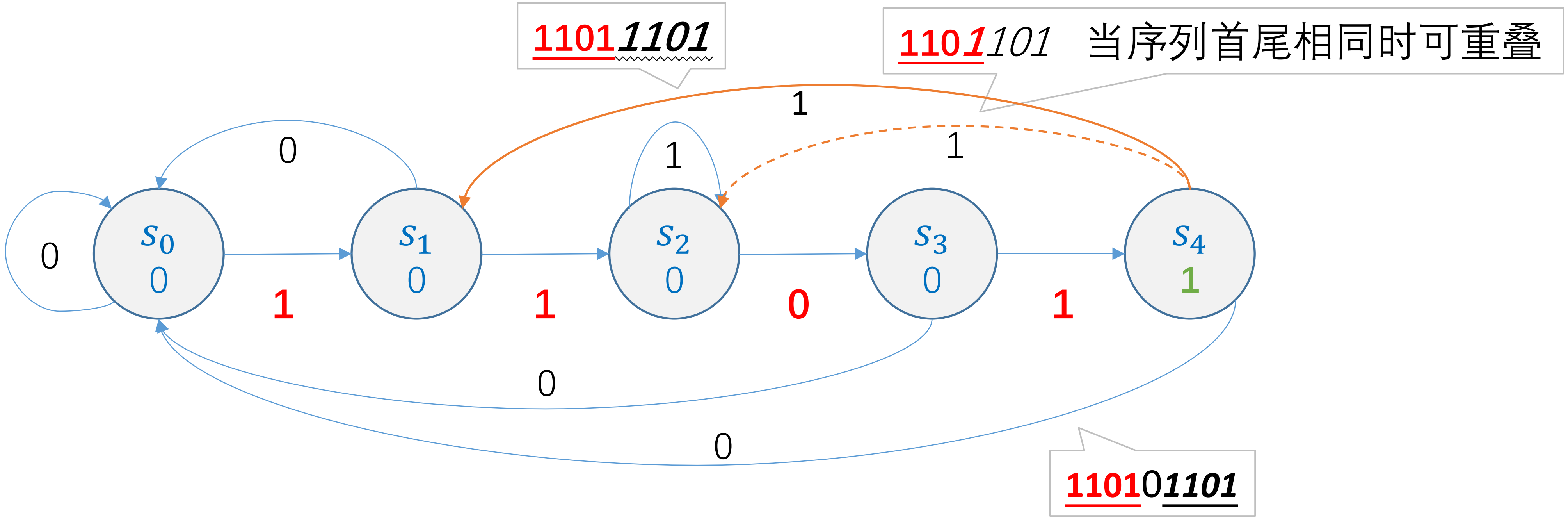




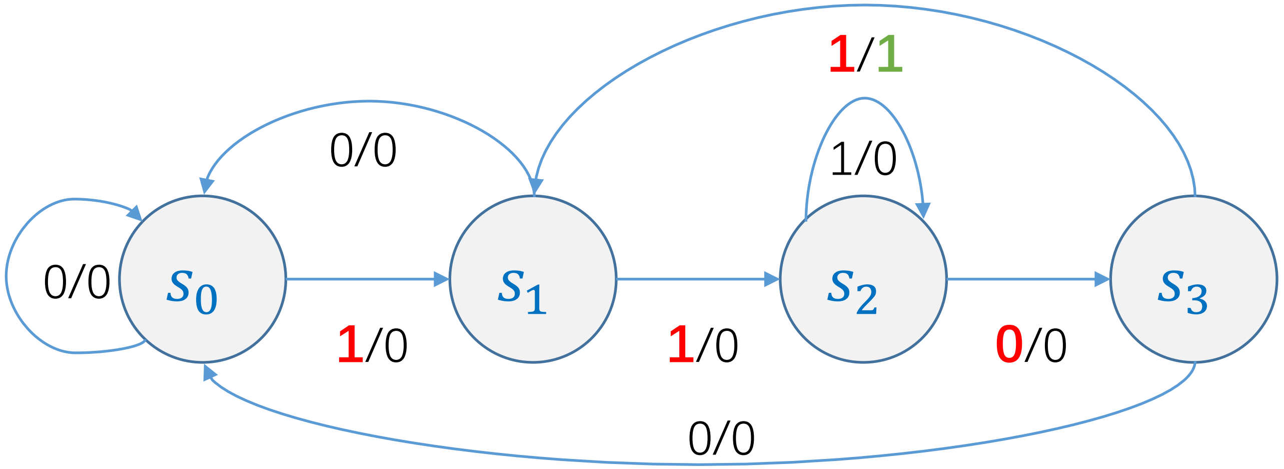
```
// Moore
assign y = (state == s0) || (state == s1);

// Mealy
assign z = (state == s0) & x;
```

1101序列检测器：状态机

摩尔状态机



米里状态机



FSM设计要点

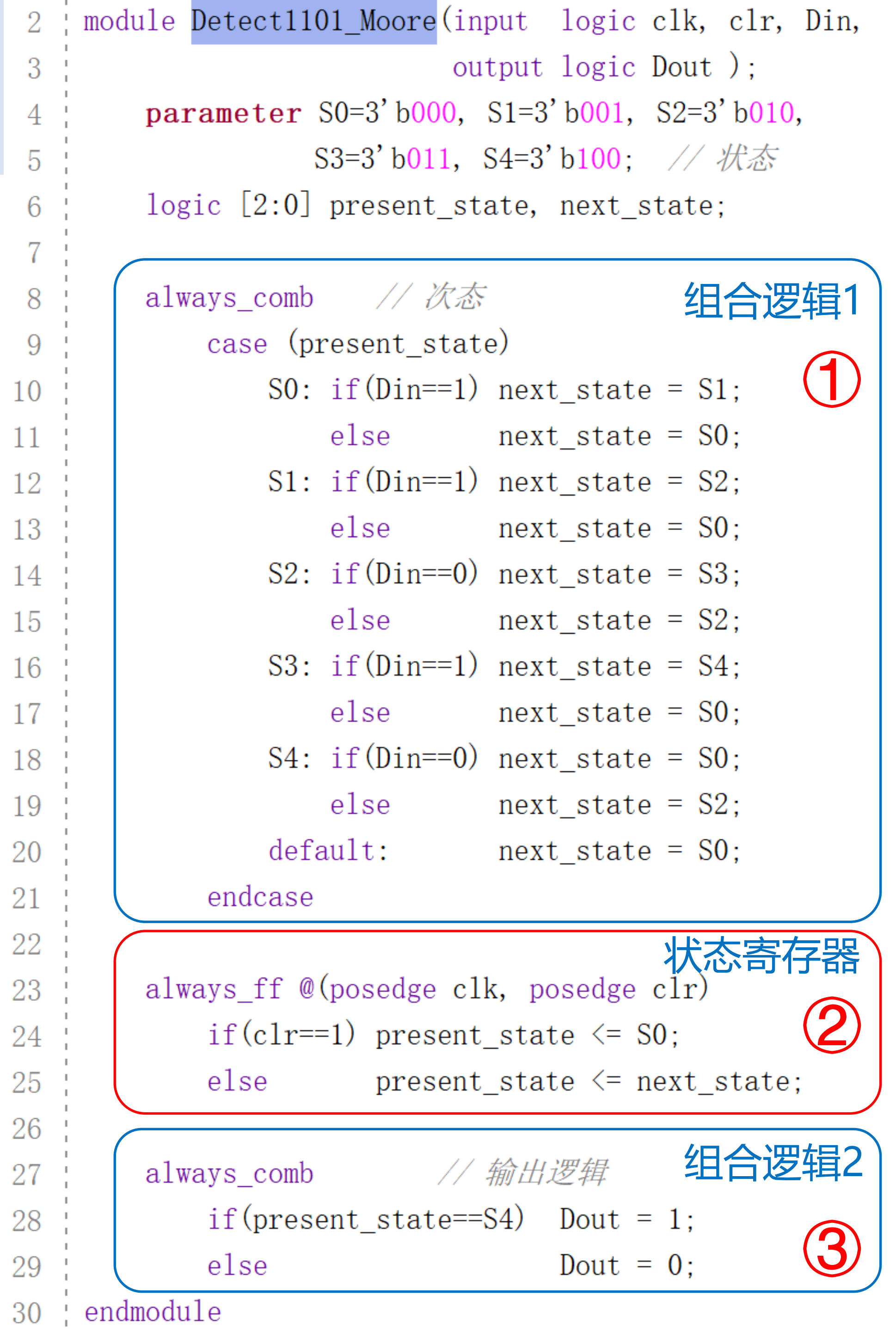
- 状态机有3部分：①**当前状态** PS、②**下一状态** NS、③**输出逻辑** OL。
- Verilog有4种描述方式：
  - **三段式**：①、②、③各用一个 always / assign 描述；
  - **两段式**：①＋②、③ *或* ②、①＋③ 各用一个always；
  - **一段式**：①＋②＋③ 只用一个always。
- **多余状态**要明确定义，或者用case语句中的default。
- **初始状态**：电路复位后所处的状态。

  实用的状态机都应有**复位信号**。
- 异步复位比同步复位占用更少的额外资源。

1101序列检测器 Moore

FSM

三段式描述

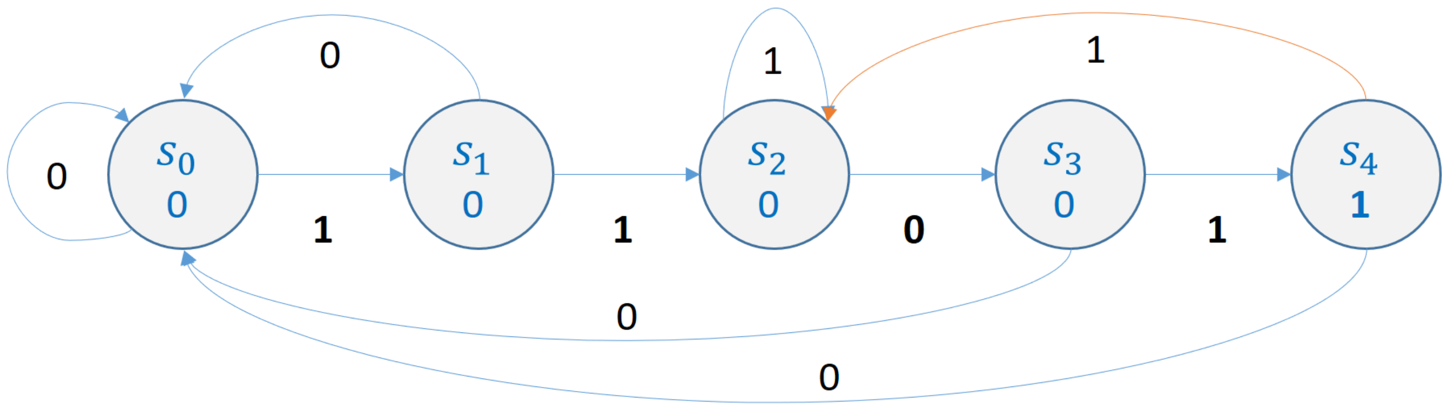


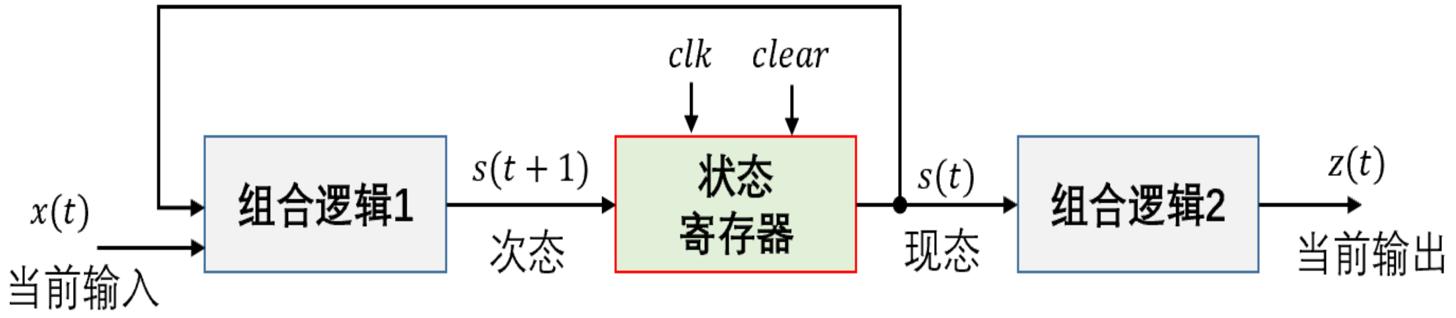
```
module Detect1101_Moore(input  logic clk, clr, Din,
                        output logic Dout );
    parameter S0=3'b000, S1=3'b001, S2=3'b010,
              S3=3'b011, S4=3'b100;  // 状态
    logic [2:0] present_state, next_state;

    always_comb     // 次态
      case (present_state)
        S0: if(Din==1) next_state = S1;
            else       next_state = S0;
        S1: if(Din==1) next_state = S2;
            else       next_state = S0;
        S2: if(Din==0) next_state = S3;
            else       next_state = S2;
        S3: if(Din==1) next_state = S4;
            else       next_state = S0;
        S4: if(Din==0) next_state = S0;
            else       next_state = S2;
        default:       next_state = S0;
      endcase

    always_ff @(posedge clk, posedge clr)
      if(clr==1) present_state <= S0;
      else       present_state <= next_state;

    always_comb           // 输出逻辑
      if(present_state==S4)  Dout = 1;
      else                   Dout = 0;
endmodule
```

组合逻辑1 ①

状态寄存器 ②

组合逻辑2 ③

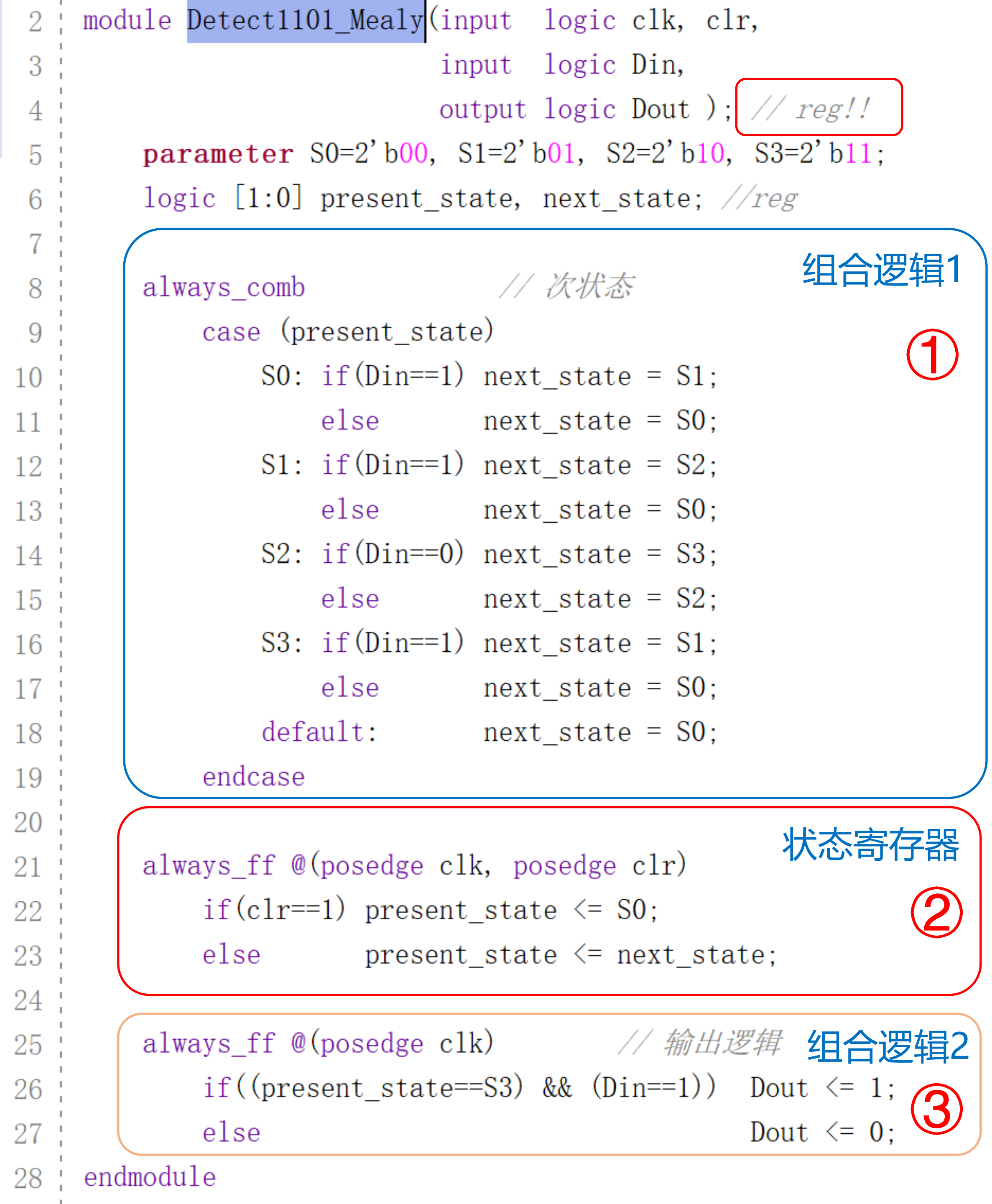




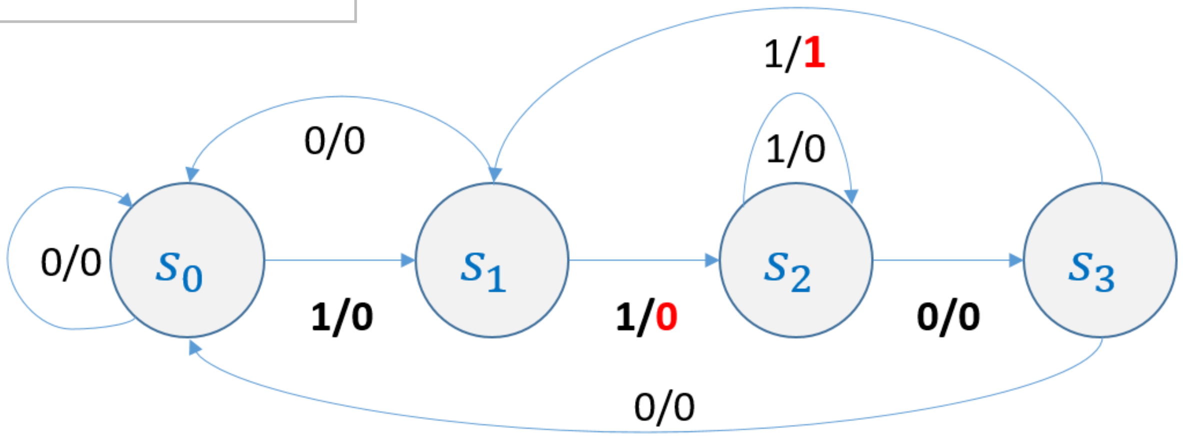
1101序列检测器 Mealy

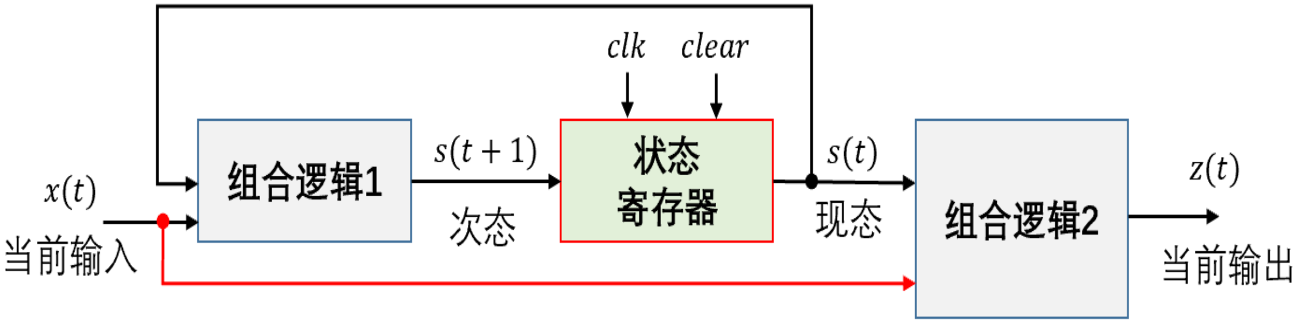
FSM

三段式描述



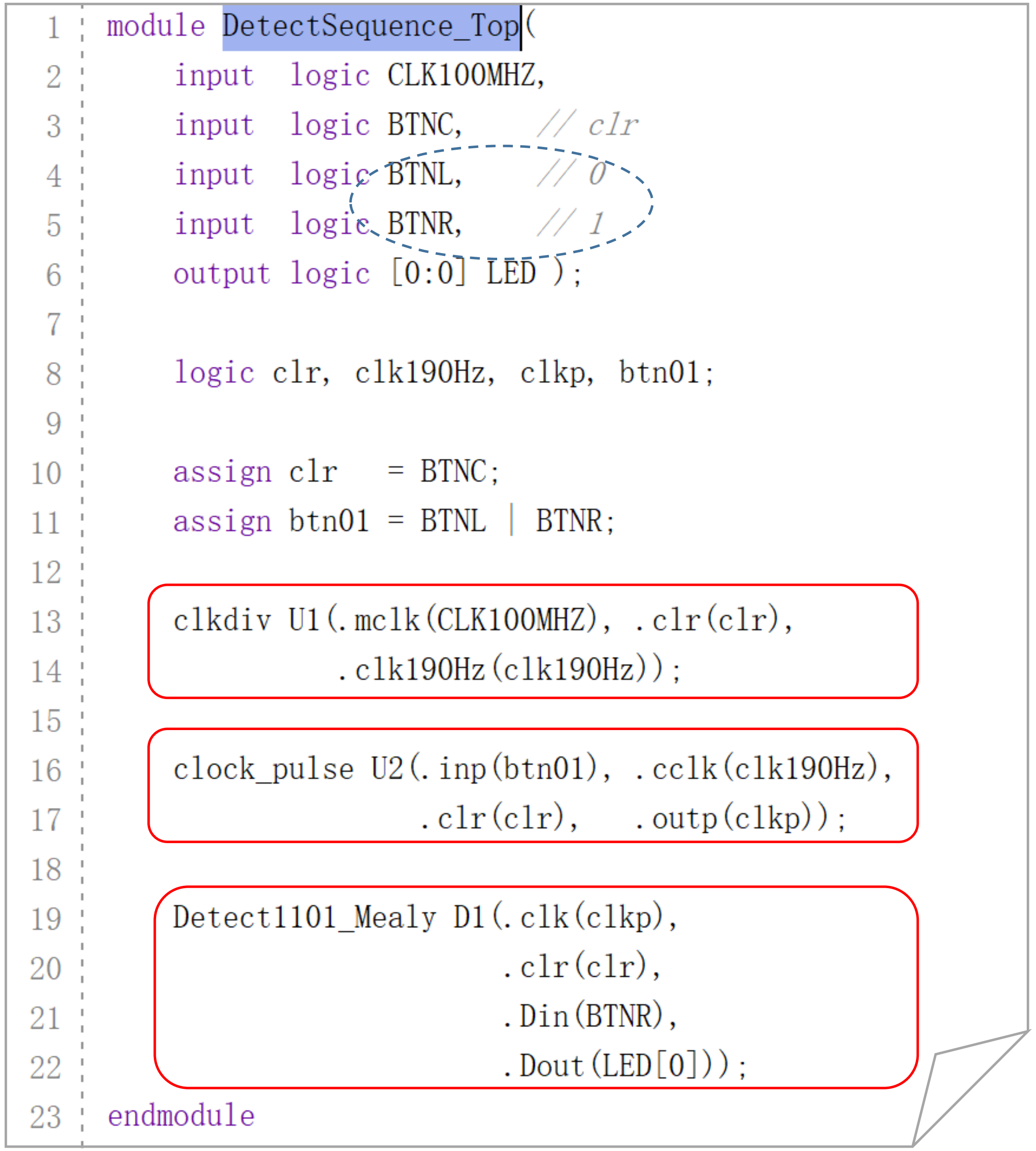
```
module Detect1101_Mealy(input  logic clk, clr,
                        input  logic Din,
                        output logic Dout ); // reg!!
    parameter S0=2'b00, S1=2'b01, S2=2'b10, S3=2'b11;
    logic [1:0] present_state, next_state; //reg

    always_comb                       // 次状态
       case (present_state)
          S0: if(Din==1) next_state = S1;
              else       next_state = S0;
          S1: if(Din==1) next_state = S2;
              else       next_state = S0;
          S2: if(Din==0) next_state = S3;
              else       next_state = S2;
          S3: if(Din==1) next_state = S1;
              else       next_state = S0;
          default:       next_state = S0;
       endcase

    always_ff @(posedge clk, posedge clr)
       if(clr==1) present_state <= S0;
       else       present_state <= next_state;

    always_ff @(posedge clk)          // 输出逻辑
       if((present_state==S3) && (Din==1))  Dout <= 1;
       else                                 Dout <= 0;
endmodule
```



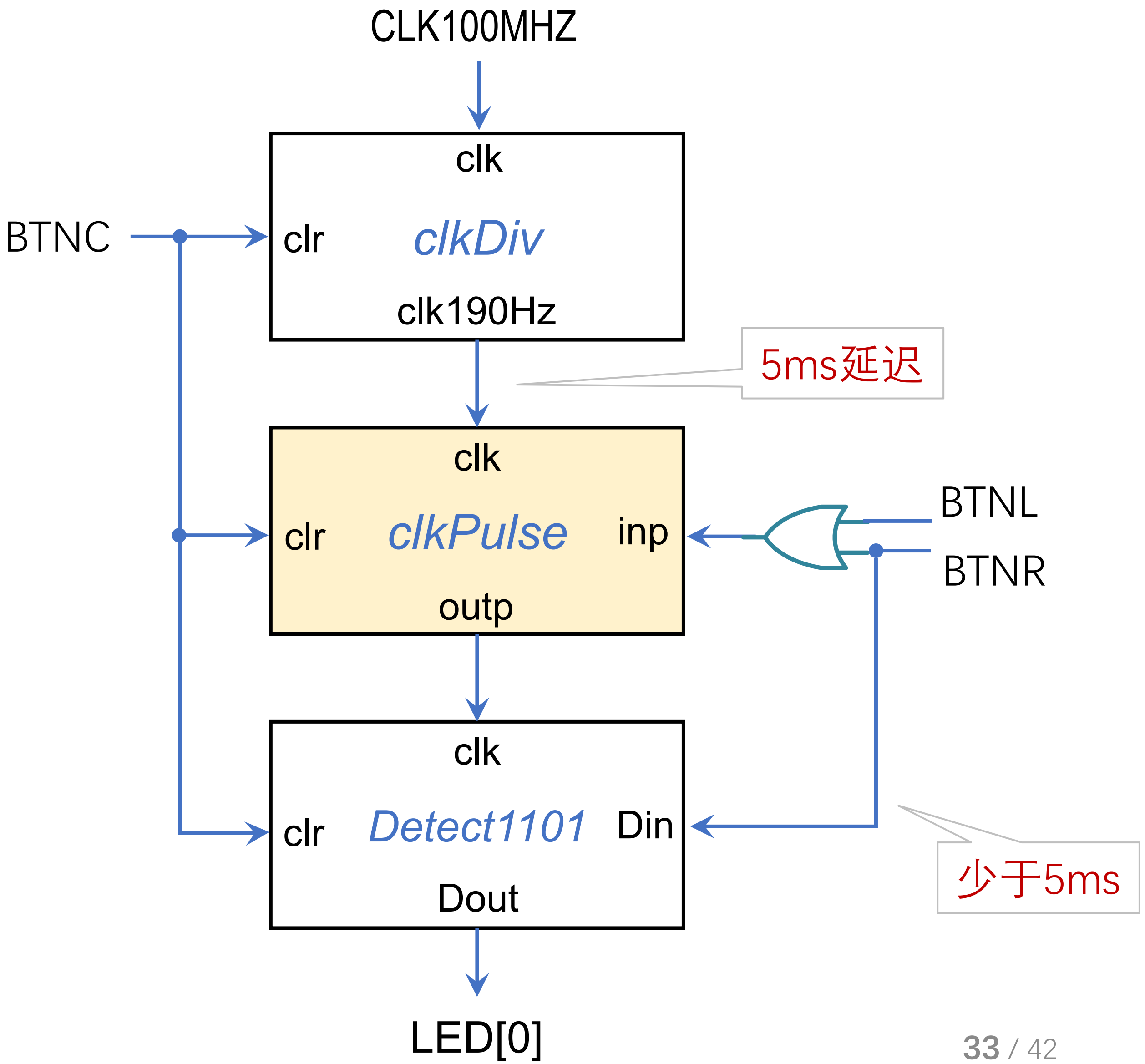


s_3的输出*Dout*需要寄存，故用时序电路！

1101序列检测器：板级验证



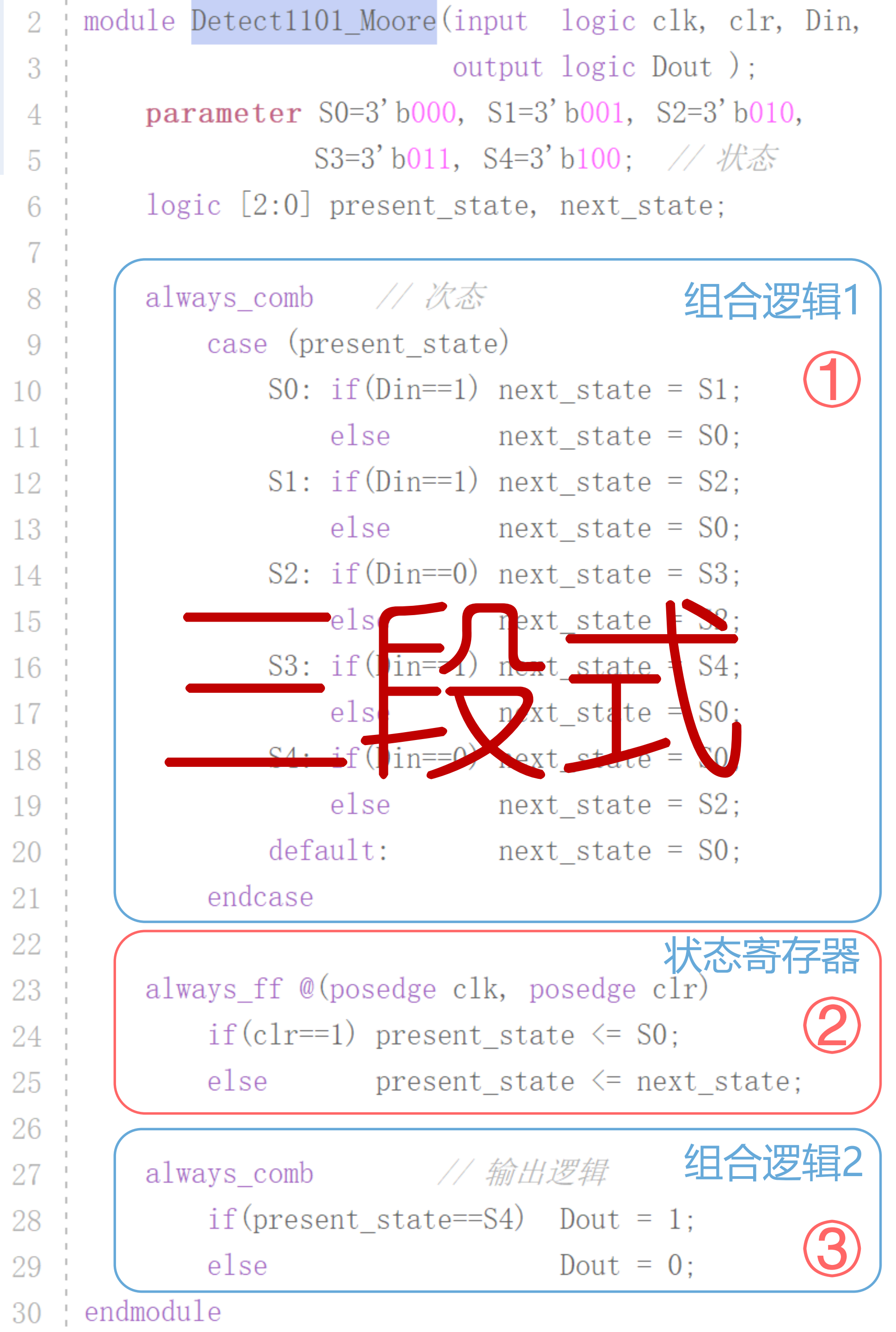
```
module DetectSequence_Top(
    input  logic CLK100MHZ,
    input  logic BTNC,     // clr
    input  logic BTNL,     // 0
    input  logic BTNR,     // 1
    output logic [0:0] LED );

    logic clr, clk190Hz, clkp, btn01;

    assign clr   = BTNC;
    assign btn01 = BTNL | BTNR;

    clkdiv U1(.mclk(CLK100MHZ), .clr(clr),
              .clk190Hz(clk190Hz));

    clock_pulse U2(.inp(btn01), .cclk(clk190Hz),
                   .clr(clr),   .outp(clkp));

    Detect1101_Mealy D1(.clk(clkp),
                        .clr(clr),
                        .Din(BTNR),
                        .Dout(LED[0]));
endmodule
```



5ms延迟

少于5ms

1101序列检测器 Moore-2



```
module Detect1101_Moore(input  logic clk, clr, Din,
                        output logic Dout );
   parameter S0=3'b000, S1=3'b001, S2=3'b010,
             S3=3'b011, S4=3'b100;   // 状态
   logic [2:0] present_state, next_state;

   always_comb     // 次态
      case (present_state)
         S0: if(Din==1) next_state = S1;
             else       next_state = S0;
         S1: if(Din==1) next_state = S2;
             else       next_state = S0;
         S2: if(Din==0) next_state = S3;
             else       next_state = S2;
         S3: if(Din==1) next_state = S4;
             else       next_state = S0;
         S4: if(Din==0) next_state = S0;
             else       next_state = S2;
         default:       next_state = S0;
      endcase

   always_ff @(posedge clk, posedge clr)
      if(clr==1) present_state <= S0;
      else       present_state <= next_state;

   always_comb          // 输出逻辑
      if(present_state==S4)  Dout = 1;
      else                   Dout = 0;
endmodule
```

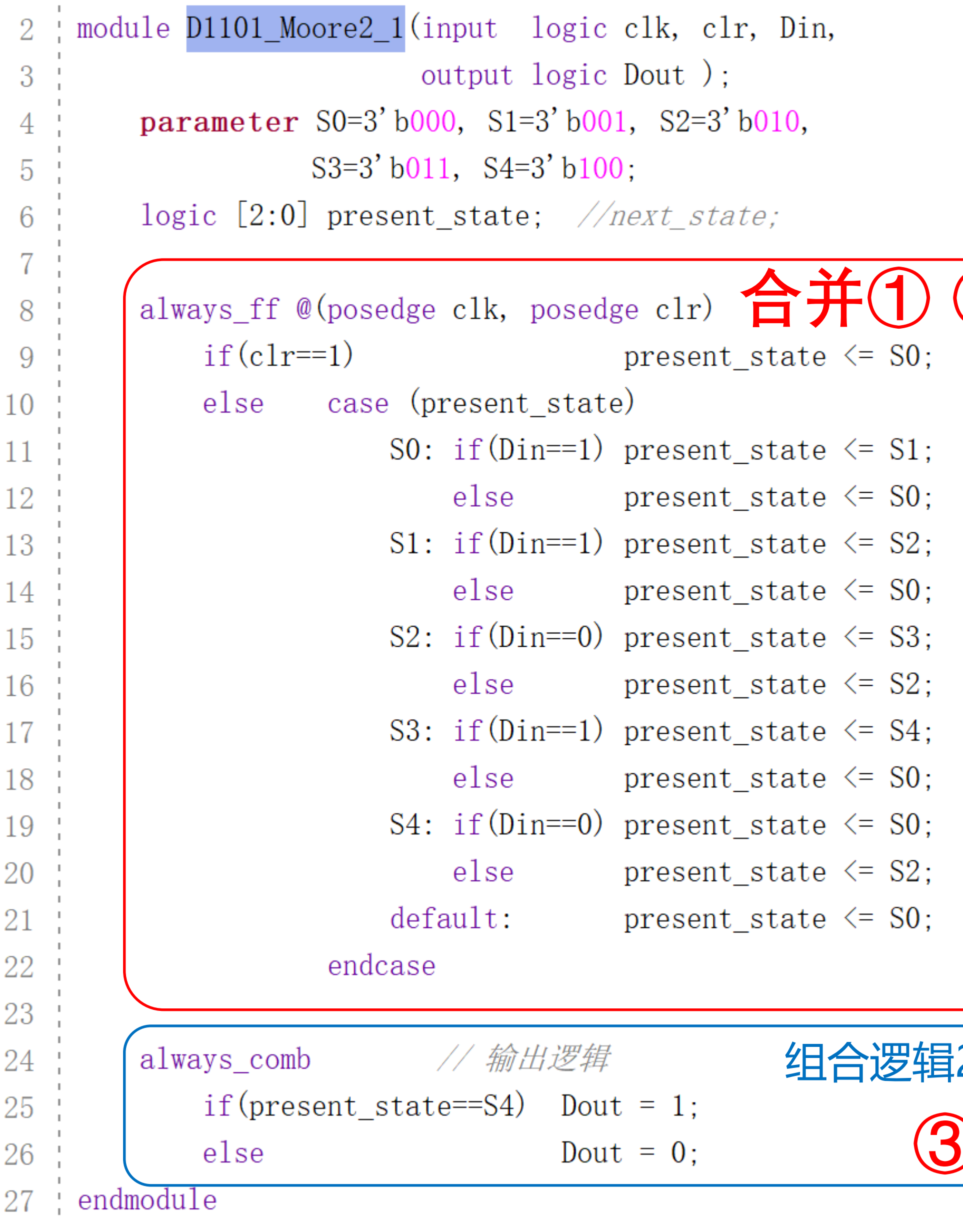
组合逻辑1 ①

三段式

状态寄存器 ②

组合逻辑2 ③

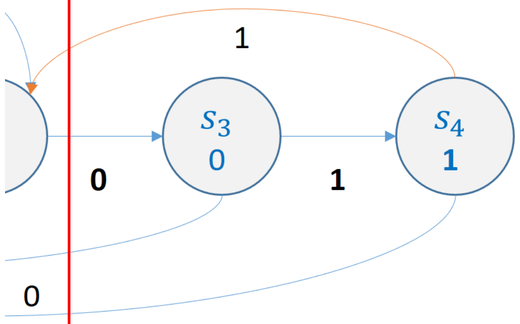
两段式描述



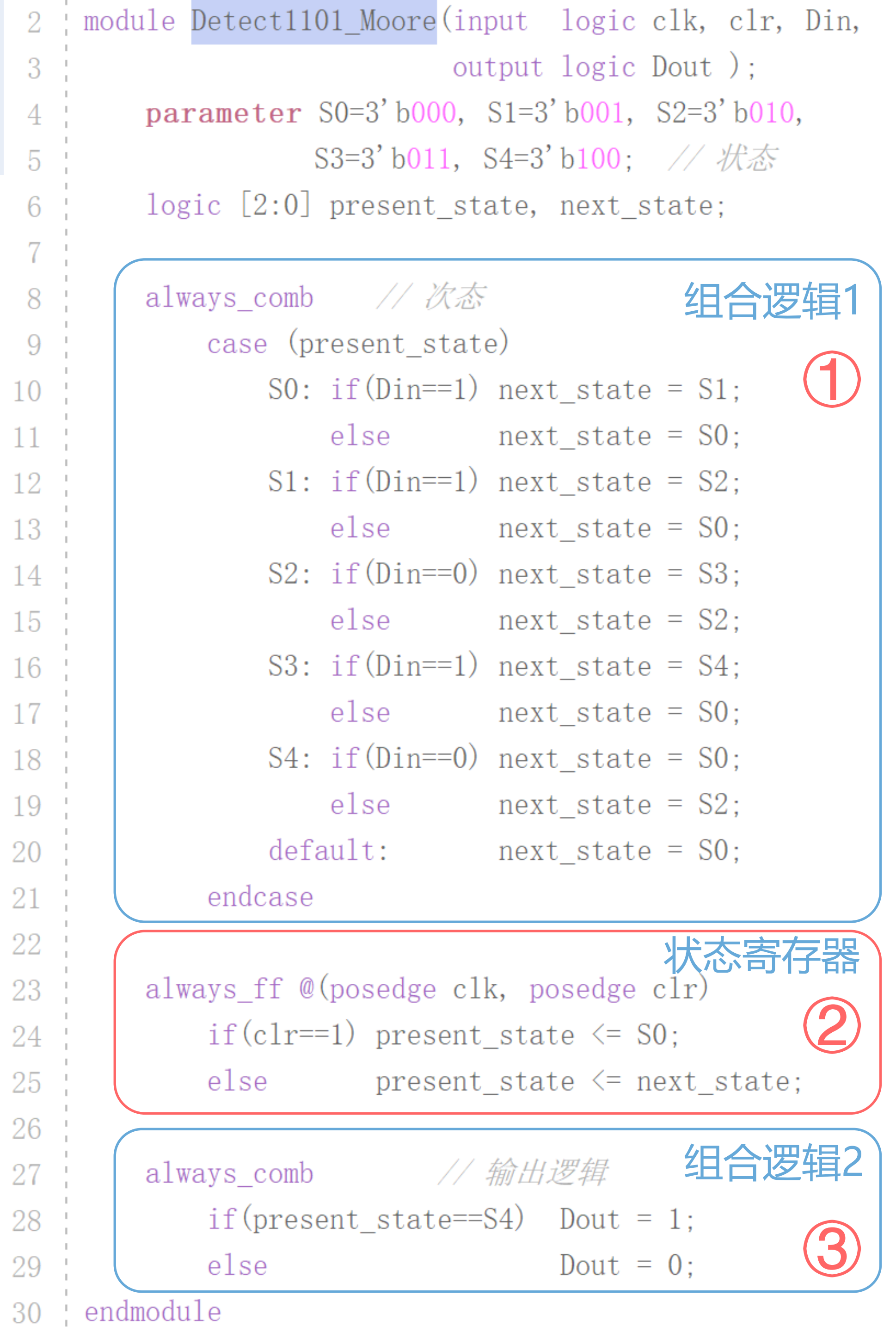
```
module D1101_Moore2_1(input  logic clk, clr, Din,
                      output logic Dout );
   parameter S0=3'b000, S1=3'b001, S2=3'b010,
             S3=3'b011, S4=3'b100;
   logic [2:0] present_state;  //next_state;

   always_ff @(posedge clk, posedge clr)
      if(clr==1)                  present_state <= S0;
      else    case (present_state)
                 S0: if(Din==1) present_state <= S1;
                     else       present_state <= S0;
                 S1: if(Din==1) present_state <= S2;
                     else       present_state <= S0;
                 S2: if(Din==0) present_state <= S3;
                     else       present_state <= S2;
                 S3: if(Din==1) present_state <= S4;
                     else       present_state <= S0;
                 S4: if(Din==0) present_state <= S0;
                     else       present_state <= S2;
                 default:       present_state <= S0;
              endcase

   always_comb          // 输出逻辑
      if(present_state==S4)  Dout = 1;
      else                   Dout = 0;
endmodule
```

合并① ②

组合逻辑2 ③



1101序列检测器 Moore-3

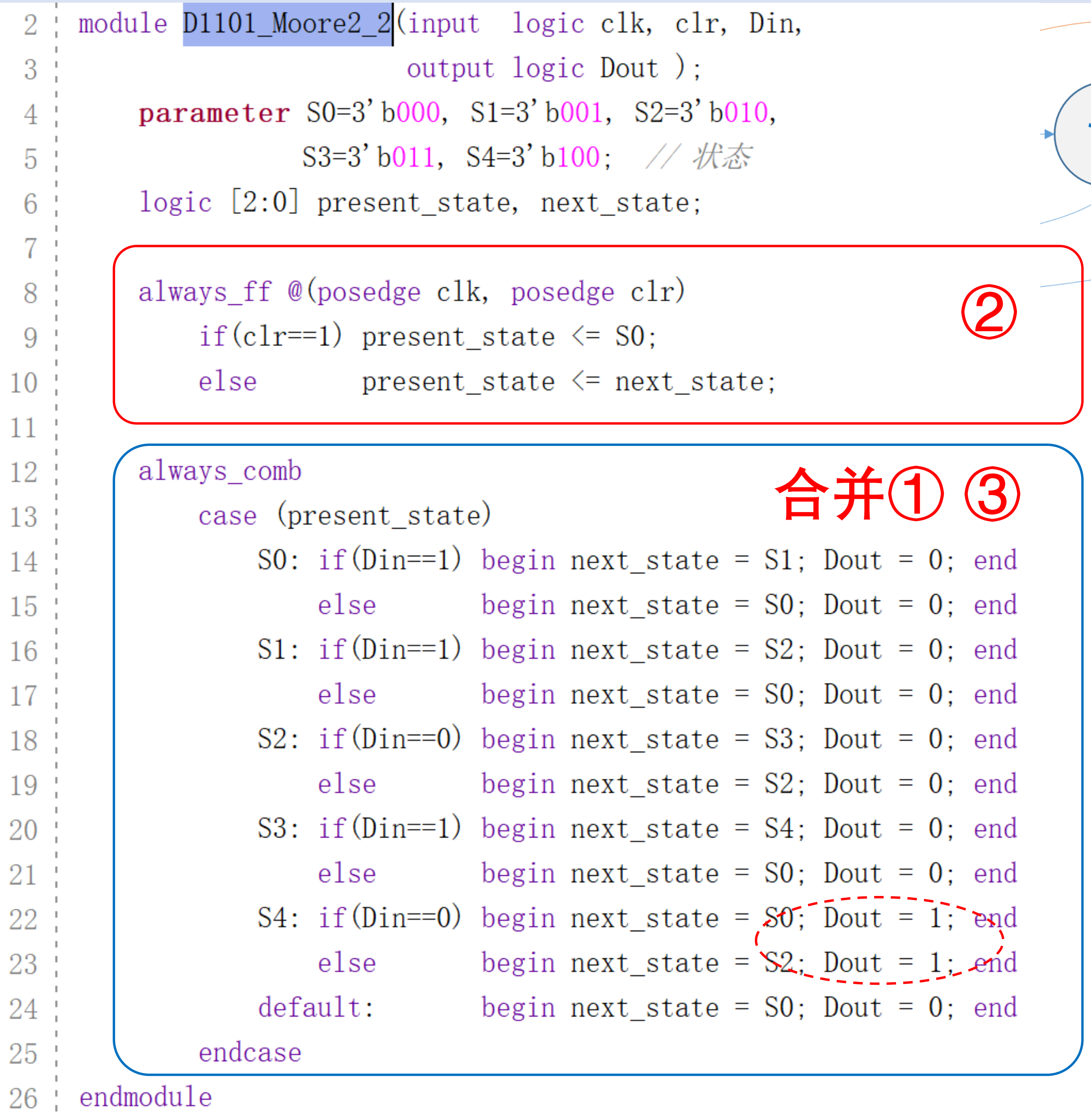


```
module Detect1101_Moore(input  logic clk, clr, Din,
                        output logic Dout );
  parameter S0=3’b000, S1=3’b001, S2=3’b010,
            S3=3’b011, S4=3’b100;  // 状态
  logic [2:0] present_state, next_state;

  always_comb    // 次态
    case (present_state)
      S0: if(Din==1) next_state = S1;
          else       next_state = S0;
      S1: if(Din==1) next_state = S2;
          else       next_state = S0;
      S2: if(Din==0) next_state = S3;
          else       next_state = S2;
      S3: if(Din==1) next_state = S4;
          else       next_state = S0;
      S4: if(Din==0) next_state = S0;
          else       next_state = S2;
      default:       next_state = S0;
    endcase

  always_ff @(posedge clk, posedge clr)
    if(clr==1) present_state <= S0;
    else       present_state <= next_state;

  always_comb          // 输出逻辑
    if(present_state==S4)  Dout = 1;
    else                   Dout = 0;
endmodule
```

组合逻辑1 ①

状态寄存器 ②

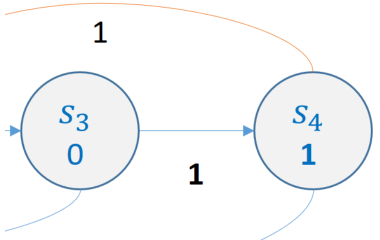
组合逻辑2 ③



```
module D1101_Moore2_2(input  logic clk, clr, Din,
                      output logic Dout );
  parameter S0=3’b000, S1=3’b001, S2=3’b010,
            S3=3’b011, S4=3’b100;  // 状态
  logic [2:0] present_state, next_state;

  always_ff @(posedge clk, posedge clr)
    if(clr==1) present_state <= S0;
    else       present_state <= next_state;

  always_comb
    case (present_state)
      S0: if(Din==1) begin next_state = S1; Dout = 0; end
          else       begin next_state = S0; Dout = 0; end
      S1: if(Din==1) begin next_state = S2; Dout = 0; end
          else       begin next_state = S0; Dout = 0; end
      S2: if(Din==0) begin next_state = S3; Dout = 0; end
          else       begin next_state = S2; Dout = 0; end
      S3: if(Din==1) begin next_state = S4; Dout = 0; end
          else       begin next_state = S0; Dout = 0; end
      S4: if(Din==0) begin next_state = S0; Dout = 1; end
          else       begin next_state = S2; Dout = 1; end
      default:       begin next_state = S0; Dout = 0; end
    endcase
endmodule
```

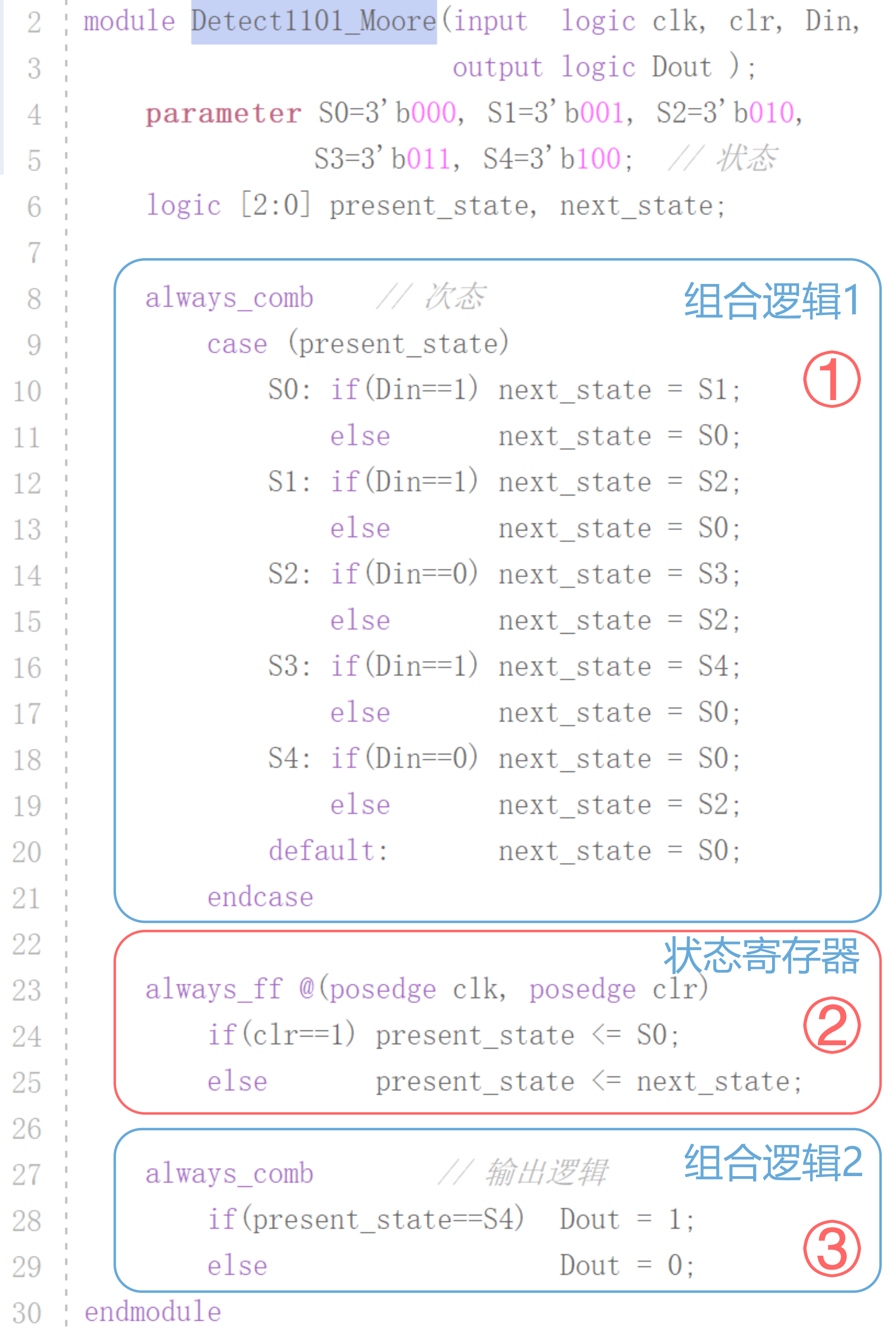
②

合并① ③



两段式描述

1101序列检测器 Moore-4



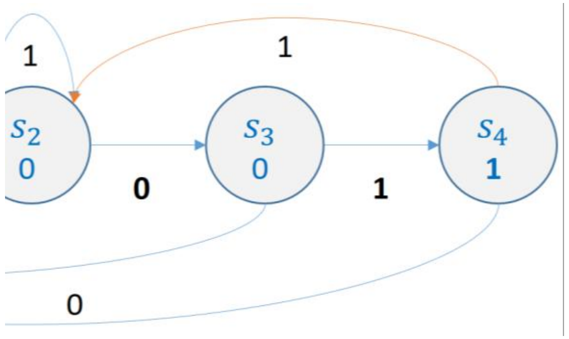
```
module Detect1101_Moore(input  logic clk, clr, Din,
                        output logic Dout );
  parameter S0=3'b000, S1=3'b001, S2=3'b010,
            S3=3'b011, S4=3'b100;  // 状态
  logic [2:0] present_state, next_state;

  always_comb    // 次态
    case (present_state)
        S0: if(Din==1) next_state = S1;
            else       next_state = S0;
        S1: if(Din==1) next_state = S2;
            else       next_state = S0;
        S2: if(Din==0) next_state = S3;
            else       next_state = S2;
        S3: if(Din==1) next_state = S4;
            else       next_state = S0;
        S4: if(Din==0) next_state = S0;
            else       next_state = S2;
        default:       next_state = S0;
    endcase

  always_ff @(posedge clk, posedge clr)
    if(clr==1) present_state <= S0;
    else       present_state <= next_state;

  always_comb          // 输出逻辑
    if(present_state==S4)  Dout = 1;
    else                   Dout = 0;
endmodule
```

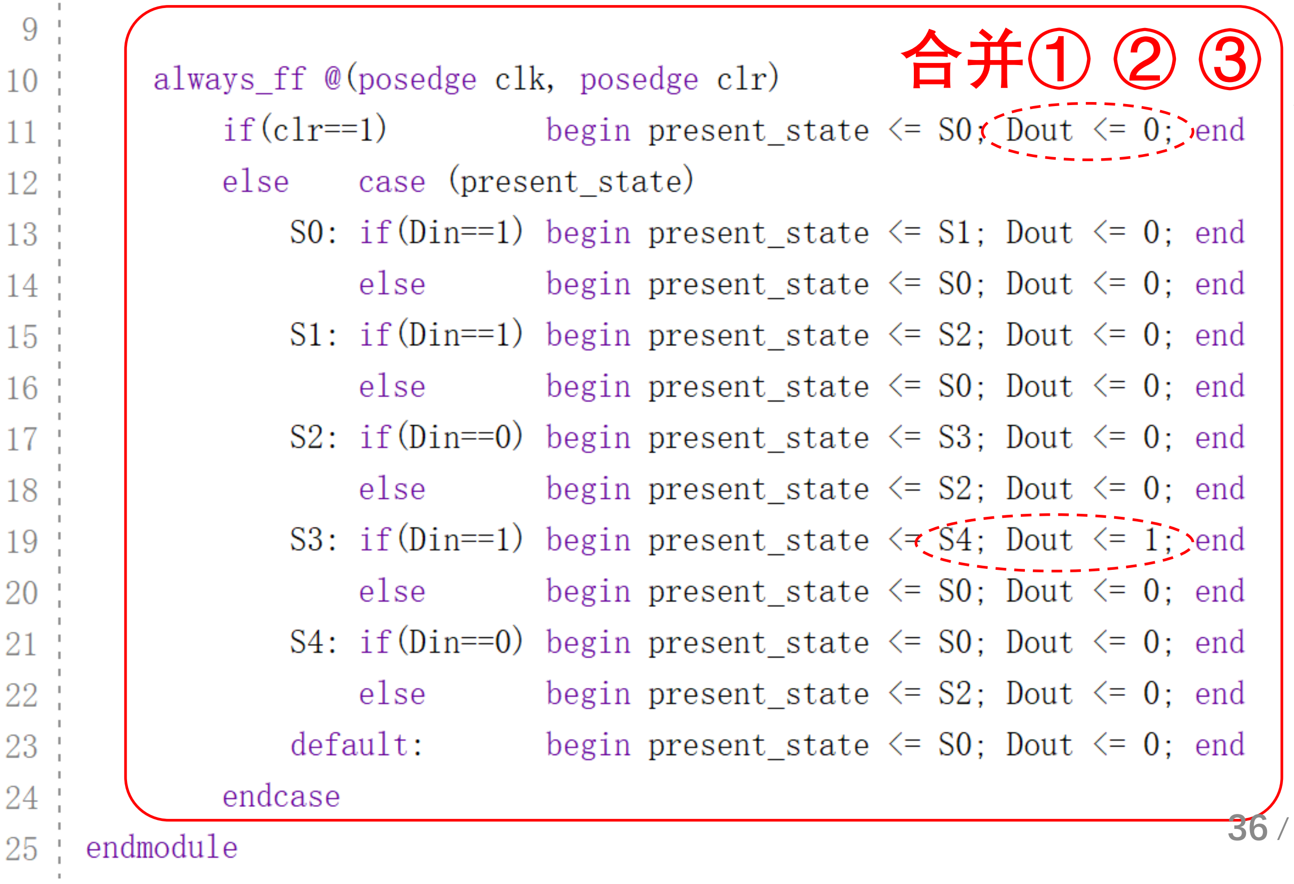
组合逻辑1 ①

状态寄存器 ②

组合逻辑2 ③

```
module D1101_Moore3(input  logic clk, clr, Din,
                    output logic Dout );

  parameter S0=3'b000, S1=3'b001, S2=3'b010,
            S3=3'b011, S4=3'b100;  // 状态

  logic [2:0] present_state; // next_state;
```





```
  always_ff @(posedge clk, posedge clr)
    if(clr==1)         begin present_state <= S0; Dout <= 0; end
    else    case (present_state)
        S0: if(Din==1) begin present_state <= S1; Dout <= 0; end
            else       begin present_state <= S0; Dout <= 0; end
        S1: if(Din==1) begin present_state <= S2; Dout <= 0; end
            else       begin present_state <= S0; Dout <= 0; end
        S2: if(Din==0) begin present_state <= S3; Dout <= 0; end
            else       begin present_state <= S2; Dout <= 0; end
        S3: if(Din==1) begin present_state <= S4; Dout <= 1; end
            else       begin present_state <= S0; Dout <= 0; end
        S4: if(Din==0) begin present_state <= S0; Dout <= 0; end
            else       begin present_state <= S2; Dout <= 0; end
        default:       begin present_state <= S0; Dout <= 0; end
    endcase
endmodule
```

合并① ② ③

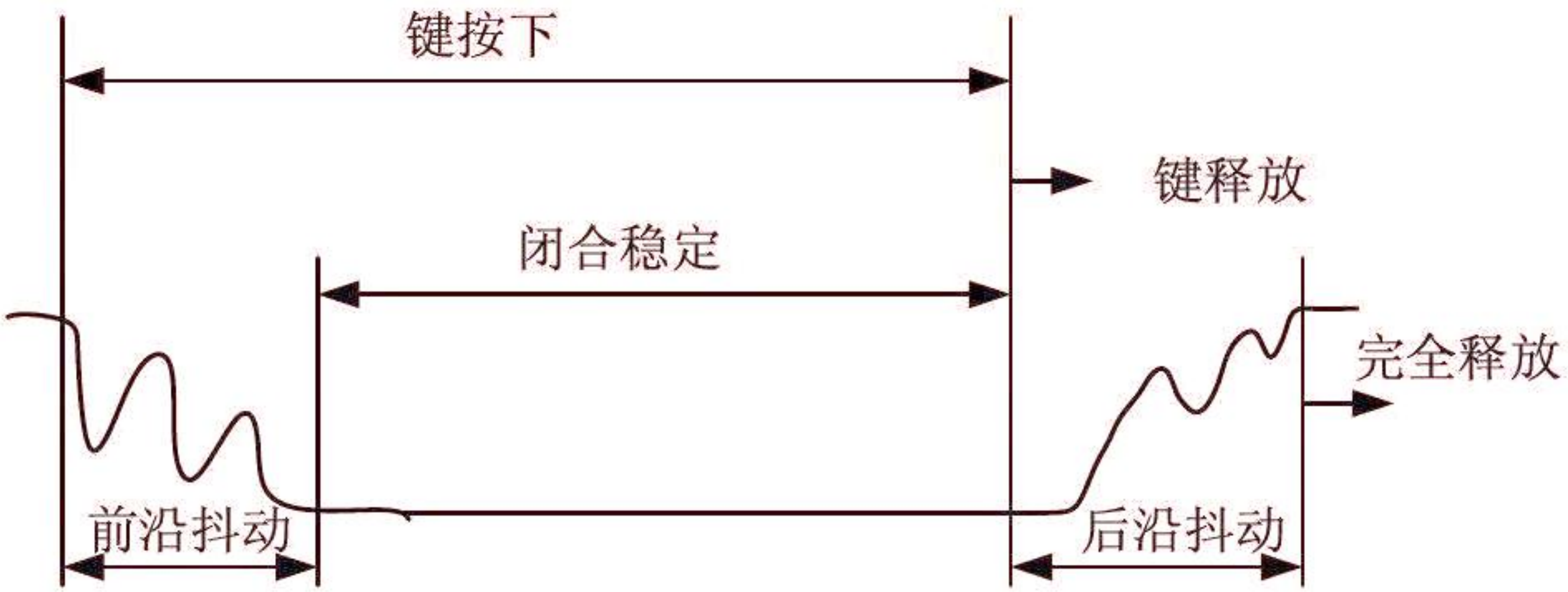
一段式

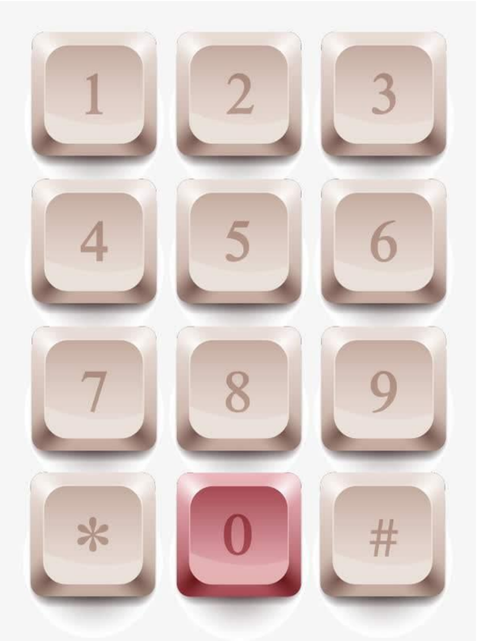
3

去抖电路

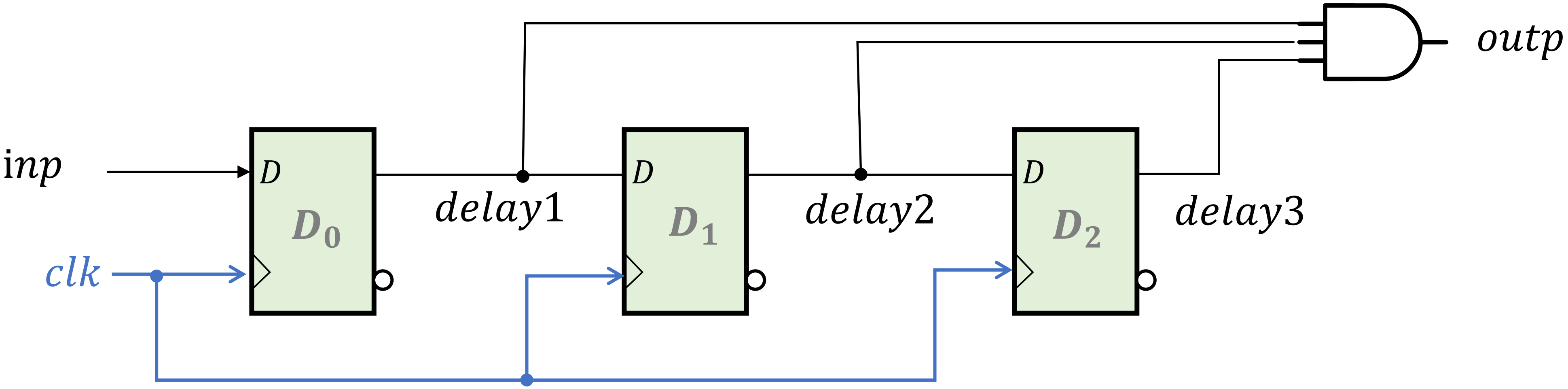
机械抖动

- 任何按钮，按动之后，在稳定下来之前都会有几毫秒的抖动。
- 可能将抖动的错误值锁存到寄存器中!

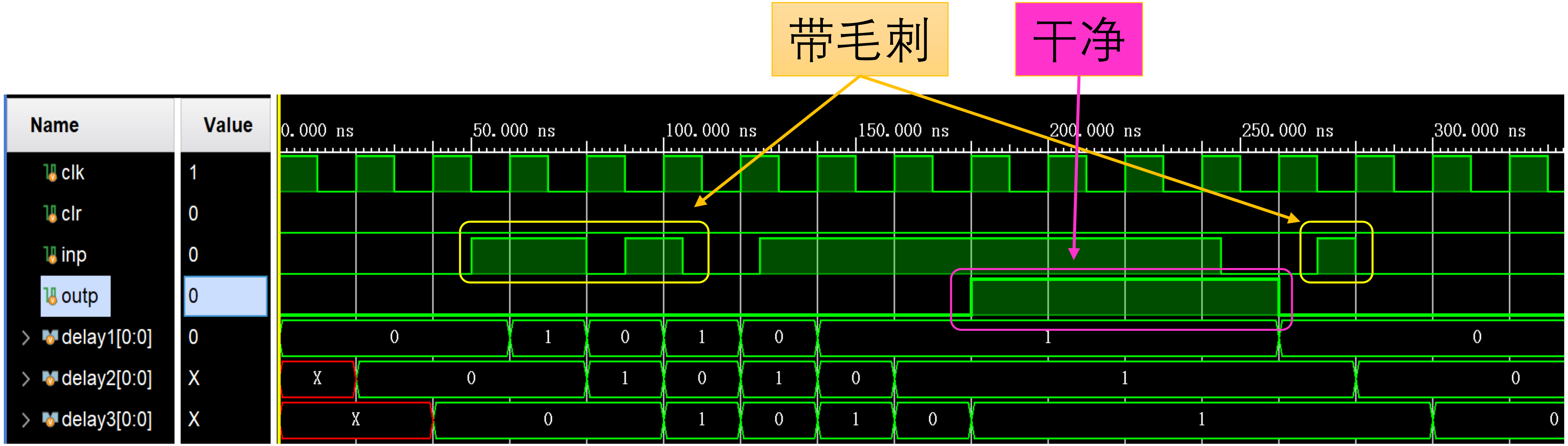




去抖电路



【注】时钟要足够慢。保证抖动在3个时钟周期内！如 clk_190Hz

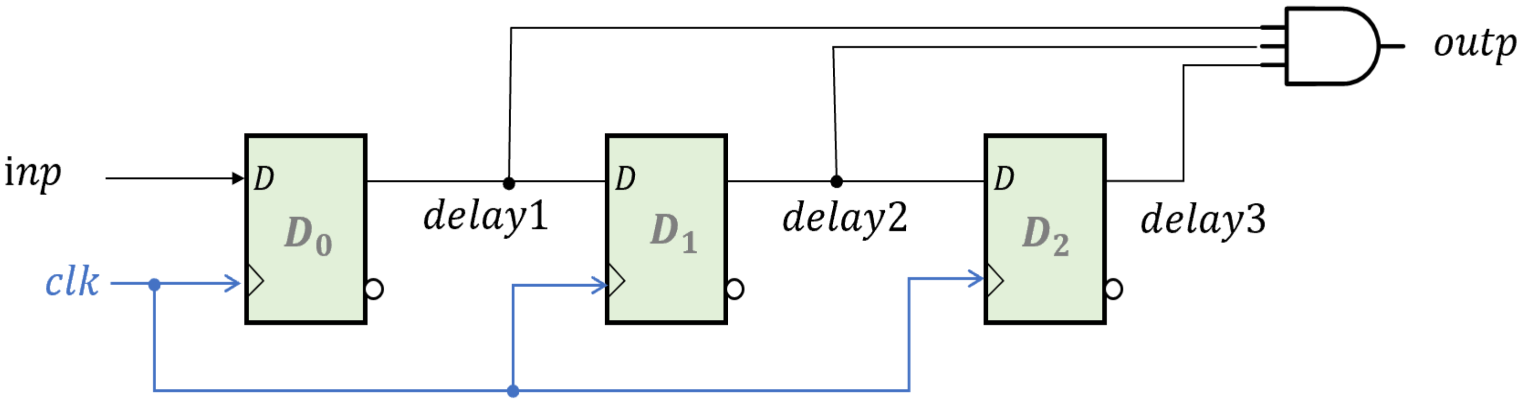


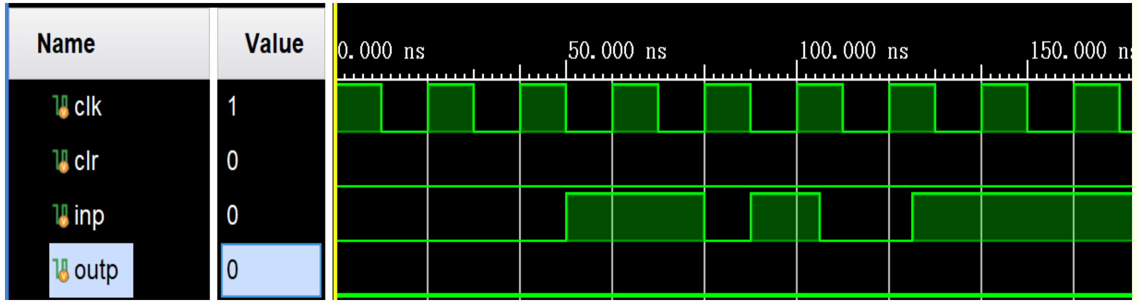
Name	Value
clk	1
clr	0
inp	0
outp	0
delay1[0:0]	0
delay2[0:0]	X
delay3[0:0]	X

Debounce

去抖代码

```
module Debounce  #(parameter N = 1)
    ( input  logic clk, clr,
      input  logic [N-1:0] inp,
      output logic [N-1:0] outp );

    logic [N-1:0] delay1, delay2, delay3;

    always_ff @(posedge clk, posedge clr)
    begin
        if(clr) begin
            delay1 <= 0;
            delay2 <= 0;
            delay3 <= 0;        end
        else begin
            delay1 <= inp;
            delay2 <= delay1;
            delay3 <= delay2;   end
    end

    assign outp = delay1 & delay2 & delay3;
endmodule
```

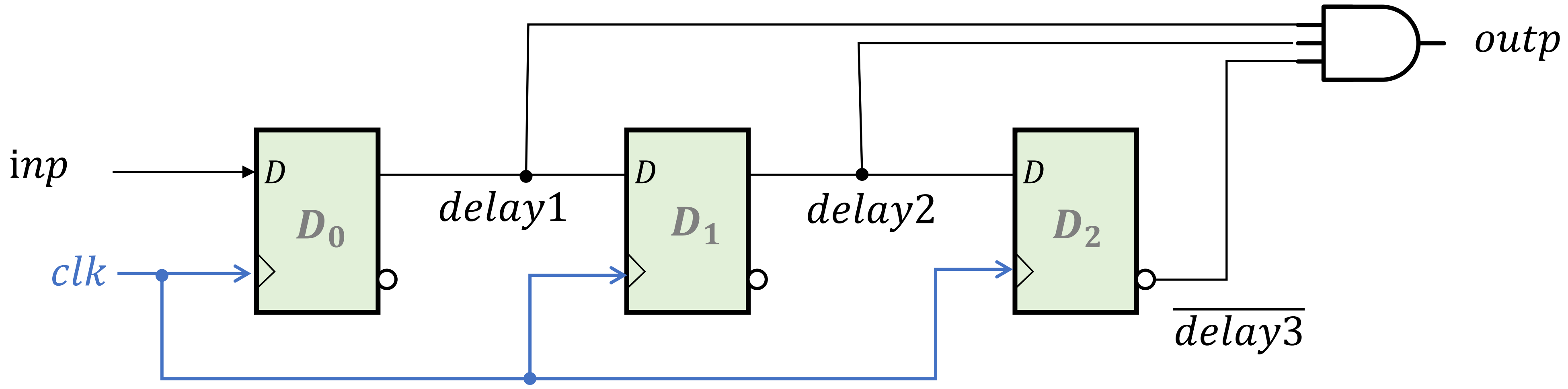


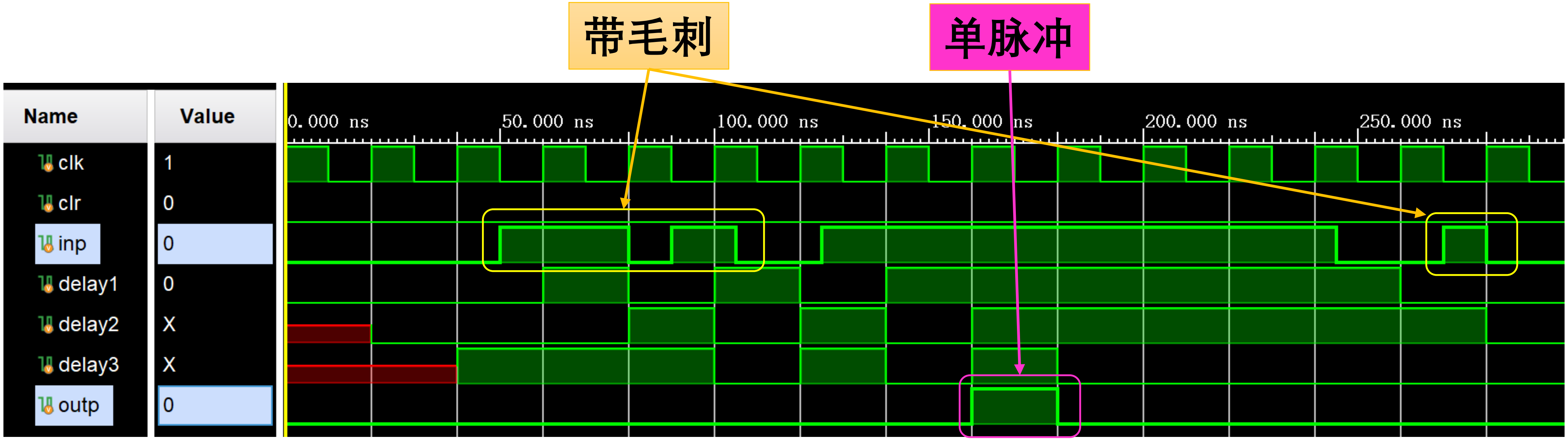


Name	Value
clk	1
clr	0
inp	0
outp	0

```
`timescale 1ns / 1ps
module Debounce_Sim(  );
    logic clk, clr, inp, outp;
    // 实例化
    Debounce D1(clk, clr, inp, outp);

    always  //时钟
    begin
        clk = 1; #10; clk = 0; #10;
    end

    initial  //输入值
    begin
        clr = 0; inp = 0;
        #50; inp = 1; #30; inp = 0;
        #10; inp = 1; #15; inp = 0;
        #20; inp = 1; #120;inp = 0;
        #25; inp = 1; #10; inp = 0;
    end
endmodule
```

时钟脉冲：按一次键，产生一个单脉冲

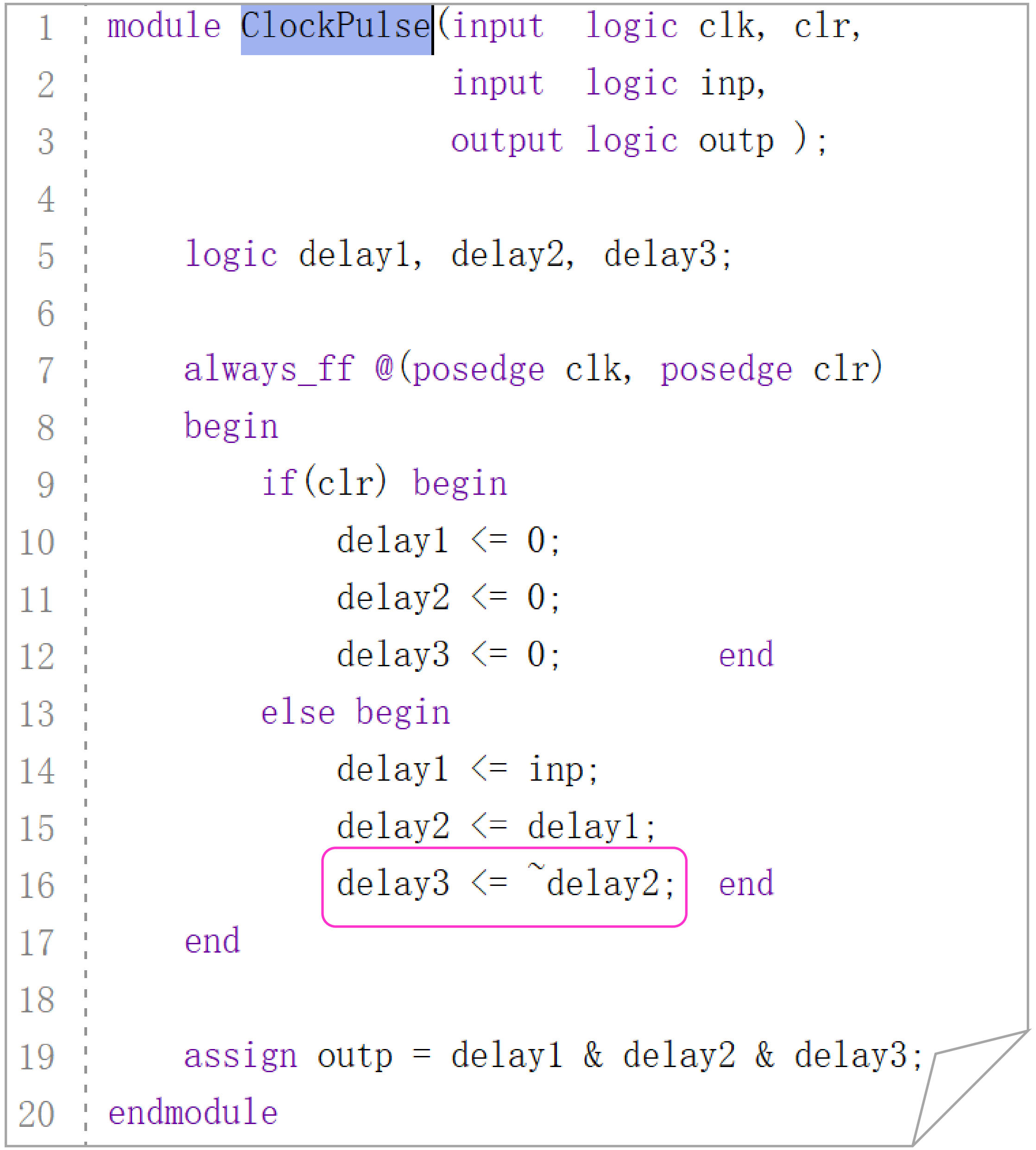




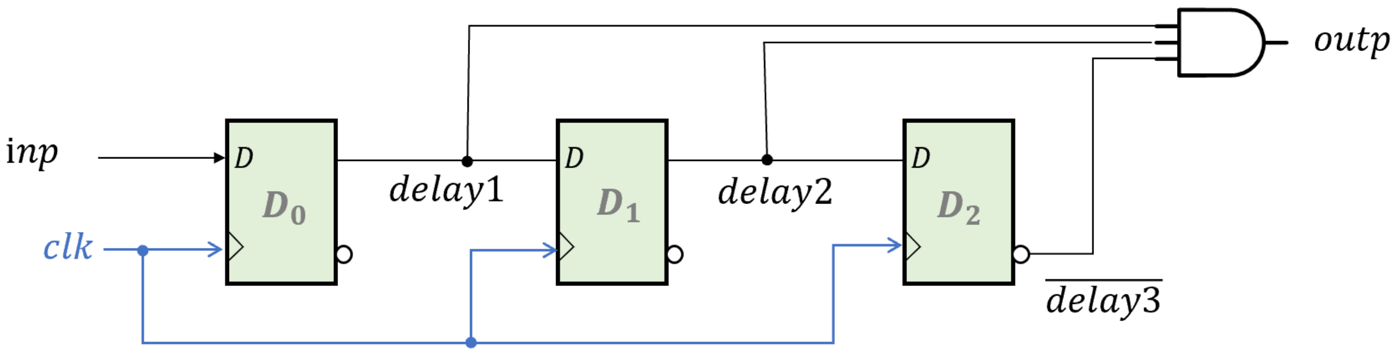
Name	Value
clk	1
clr	0
inp	0
delay1	0
delay2	X
delay3	X
outp	0

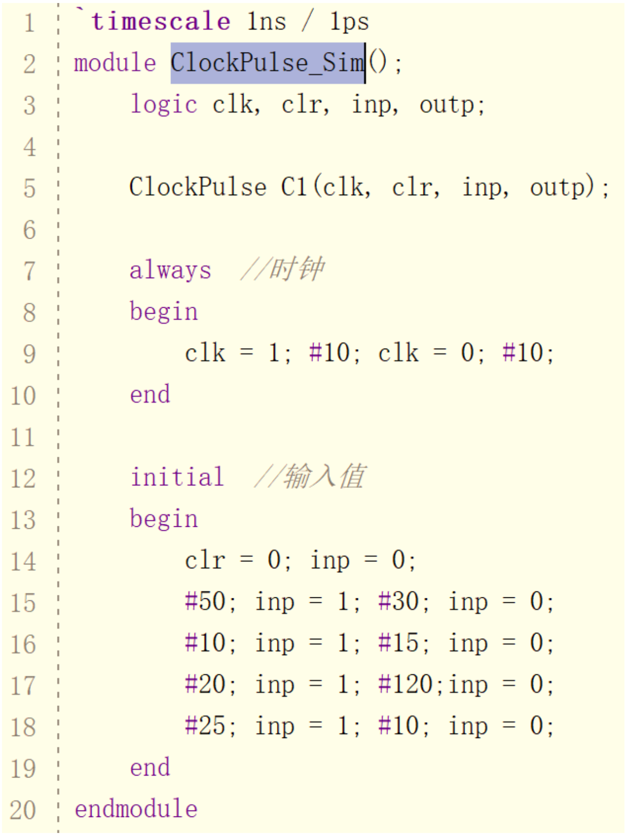
clockPulse

单脉冲代码



```
module ClockPulse(input  logic clk, clr,
                  input  logic inp,
                  output logic outp );

    logic delay1, delay2, delay3;

    always_ff @(posedge clk, posedge clr)
    begin
        if(clr) begin
            delay1 <= 0;
            delay2 <= 0;
            delay3 <= 0;        end
        else begin
            delay1 <= inp;
            delay2 <= delay1;
            delay3 <= ~delay2;  end
    end

    assign outp = delay1 & delay2 & delay3;
endmodule
```





```
`timescale 1ns / 1ps
module ClockPulse_Sim();
    logic clk, clr, inp, outp;

    ClockPulse C1(clk, clr, inp, outp);

    always  //时钟
    begin
        clk = 1; #10; clk = 0; #10;
    end

    initial  //输入值
    begin
        clr = 0; inp = 0;
        #50; inp = 1; #30; inp = 0;
        #10; inp = 1; #15; inp = 0;
        #20; inp = 1; #120;inp = 0;
        #25; inp = 1; #10; inp = 0;
    end
endmodule
```