数字逻辑与部件设计

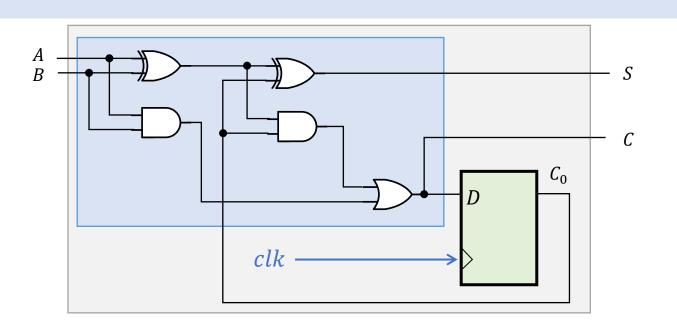
9. 时序电路分析

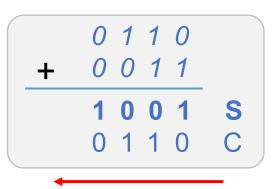




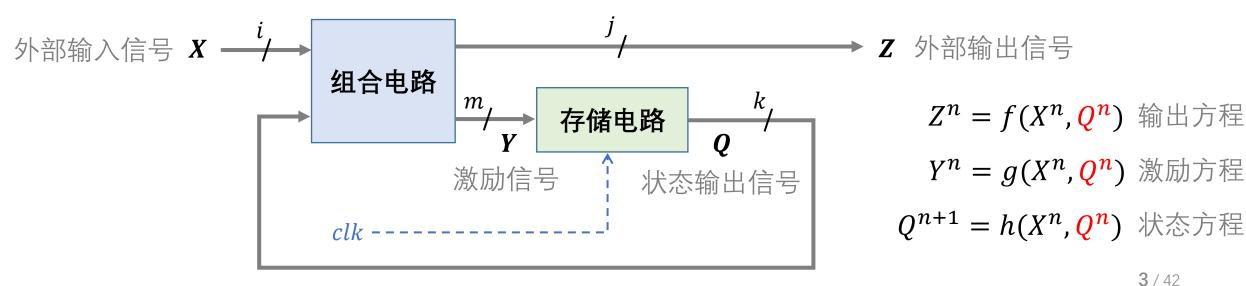
时房电路分析

时序逻辑电路 的 基本结构



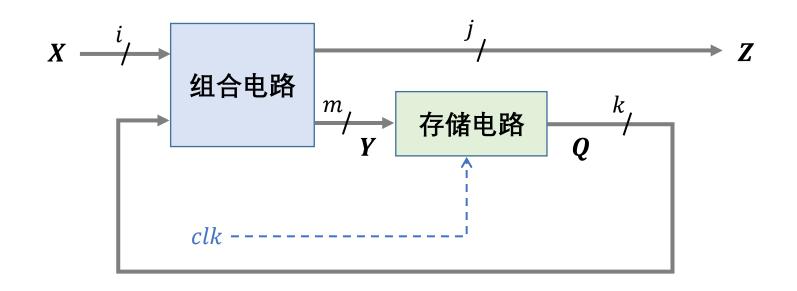


每个时钟, 从低位开始逐位开始计算

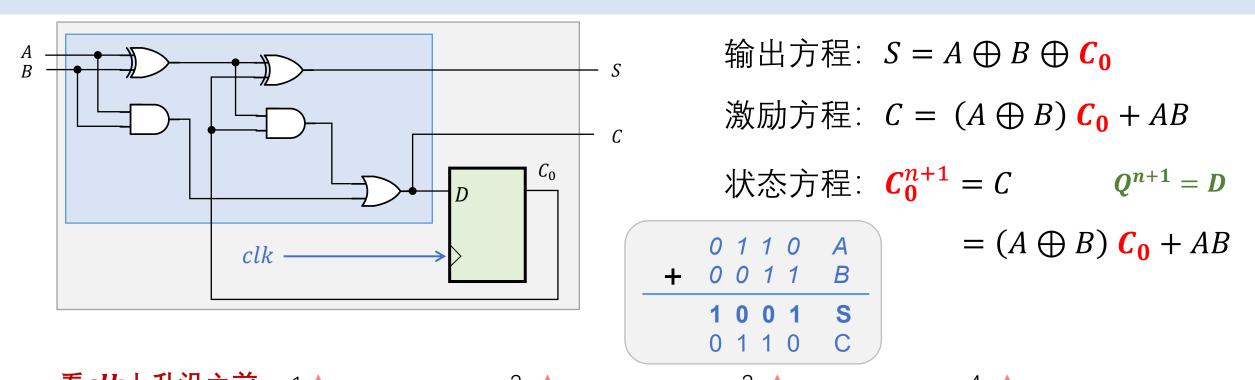


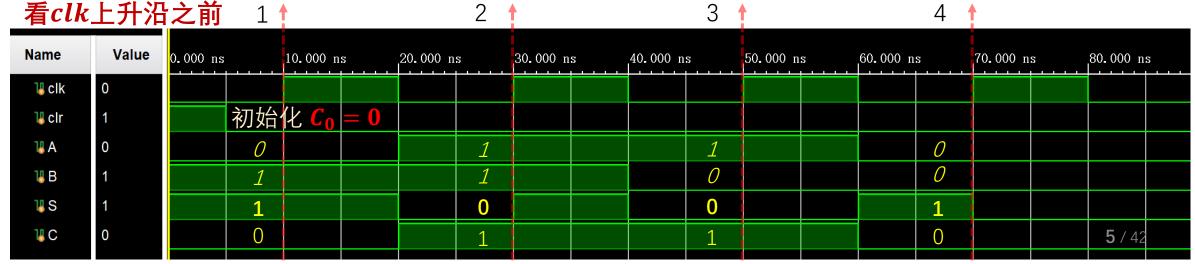
组合电路、时序电路

- **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。

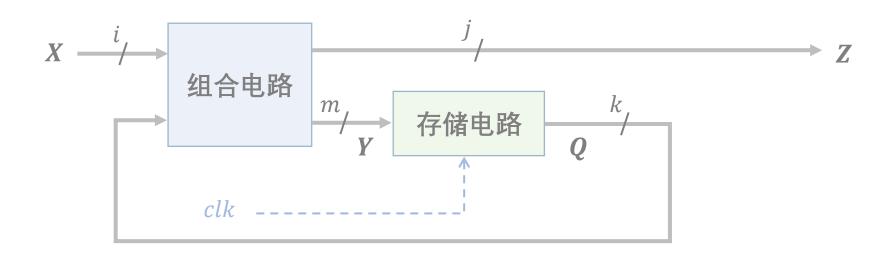


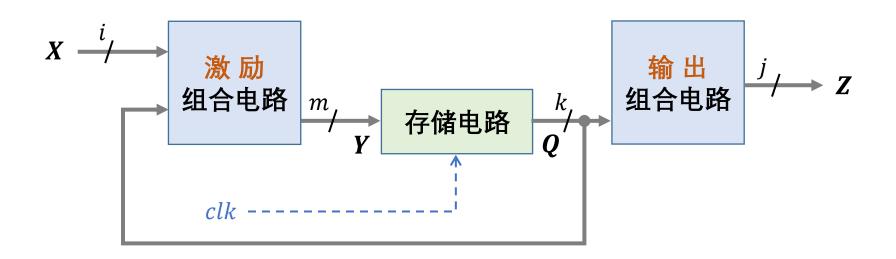
时序逻辑电路分析



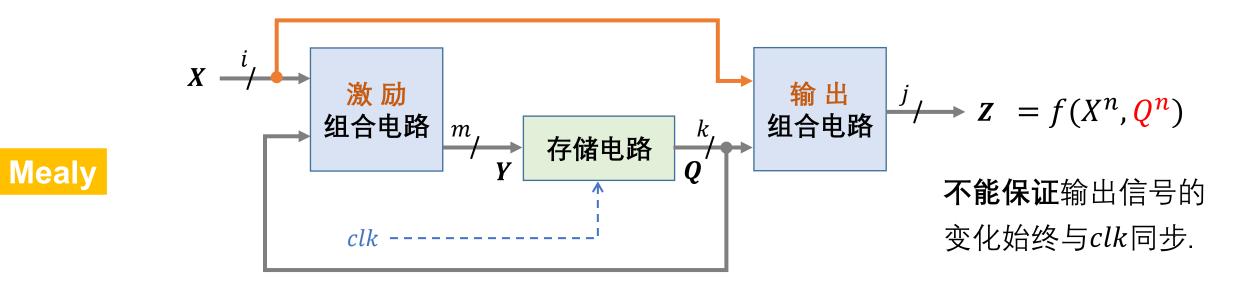


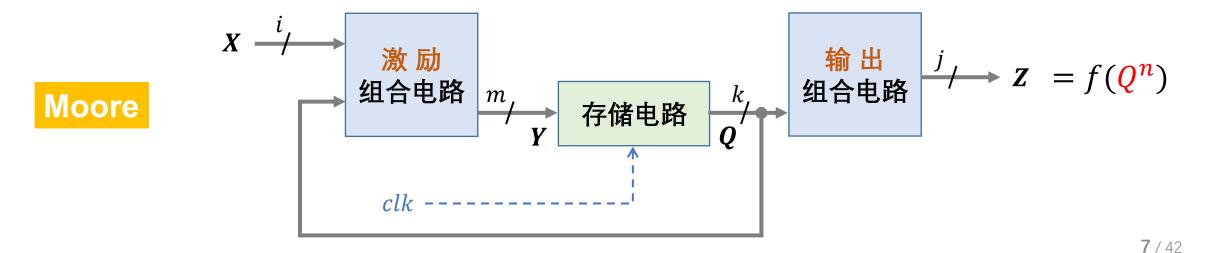
时序逻辑电路 的 通用结构



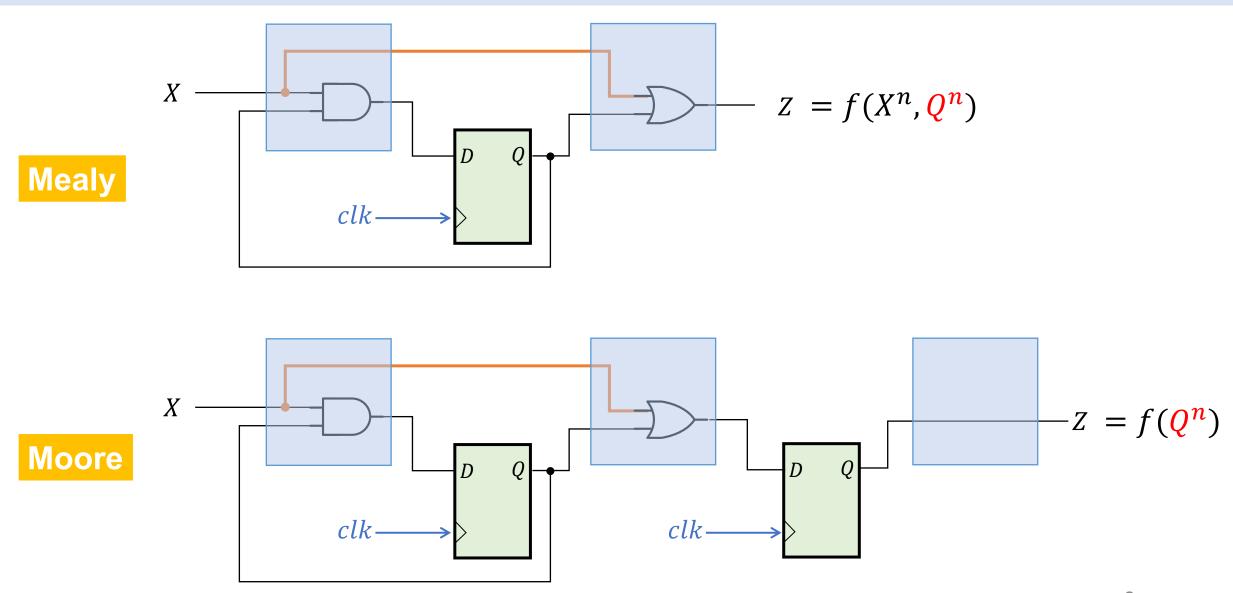


从输出Z看...

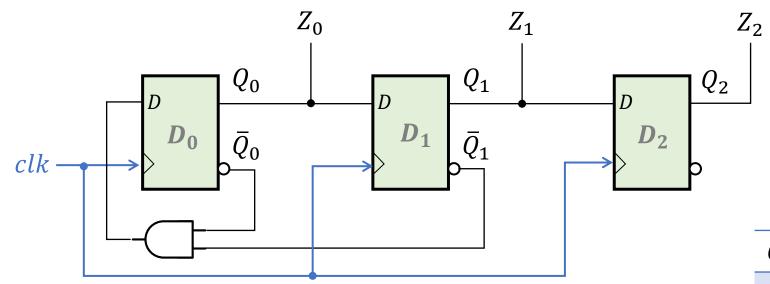




Mealy 变 Moore



【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

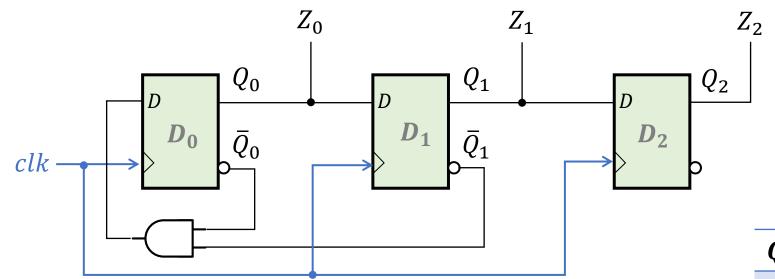
状态方程: $Q_2^* = Q_1$, $Q_1^* = Q_0$, $Q_0^* = \bar{Q}_1\bar{Q}_0$

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

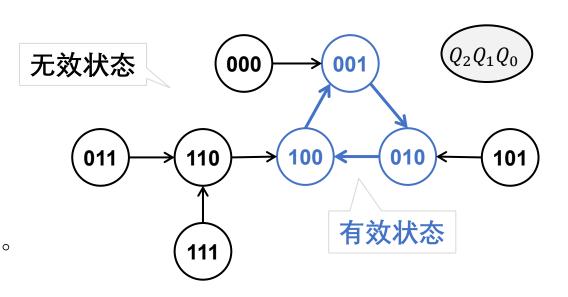
9 / 42

【例1】同步时序电路分析



自启动电路

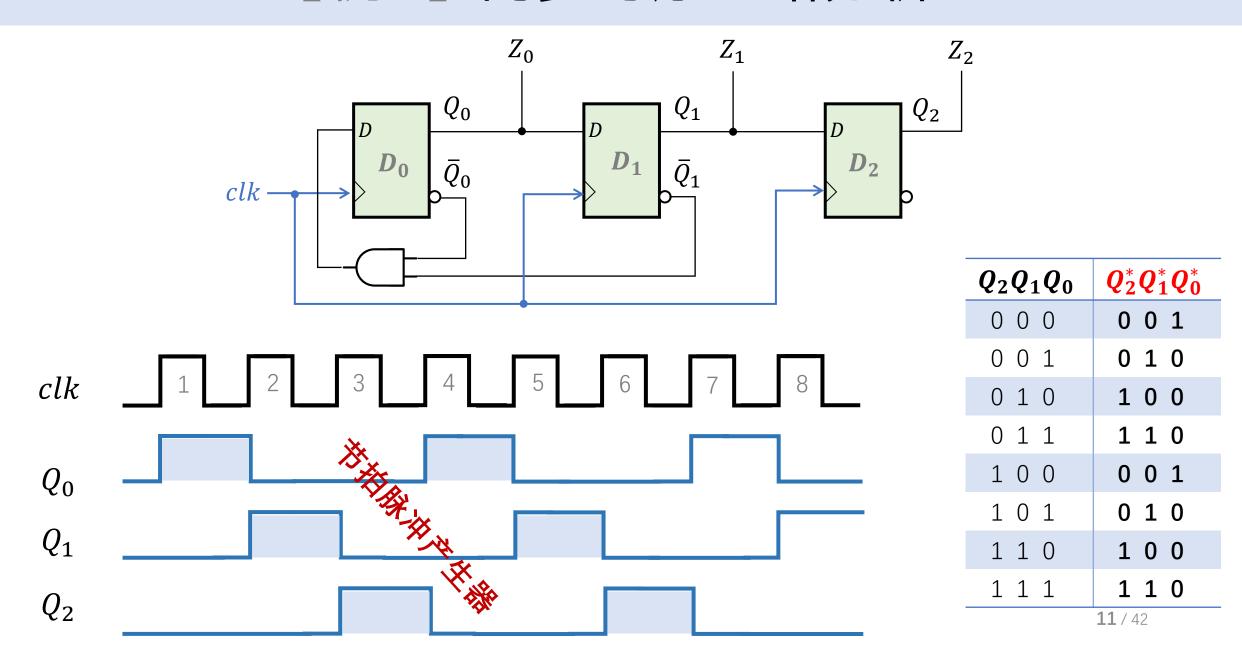
如果电路处于任何一个无效状态, 经过若干时钟脉冲后, 都可以**自动**进入有效状态的电路。



$Q_2Q_1Q_0$	$Q_2^*Q_1^*Q_0^*$
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
	10 / 40

10 / 42

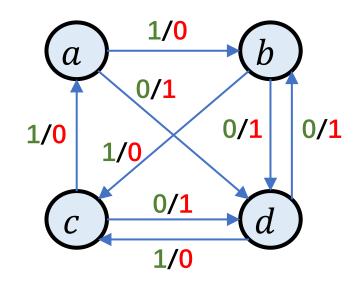
【例1】同步时序电路分析



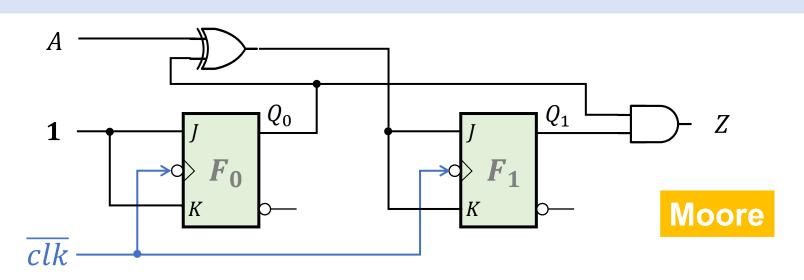
【练习1】状态表 =>状态图

C	S^*/Z	
S	A = 0	A = 1
а	d / 1	b / 0
b	d / 1	c / 0
С	d / 1	a / 0
d	b / 1	c / 0





【例2】JK触发器电路分析:公式法



O	Q_1^*	Z	
Q_1Q_0	A = 0	A = 1	<u>L</u>
0 0	0 1	11	0
0 1	10	0 0	0
10	1 1	0 1	0
1 1	0 0	10	1

激励方程: $J_0 = K_0 = 1$

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

特征方程: $Q^* = J\overline{Q} + \overline{K}Q$

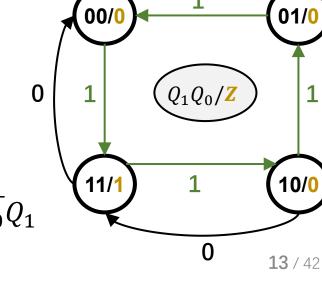
状态方程: $Q_0^* = J_0 \bar{Q}_0 + \bar{K}_0 Q_0 = \bar{Q}_0$, $Q_1^* = J_1 \bar{Q}_1 + \bar{K}_1 Q_1$

输出方程: $Z = Q_1Q_0$

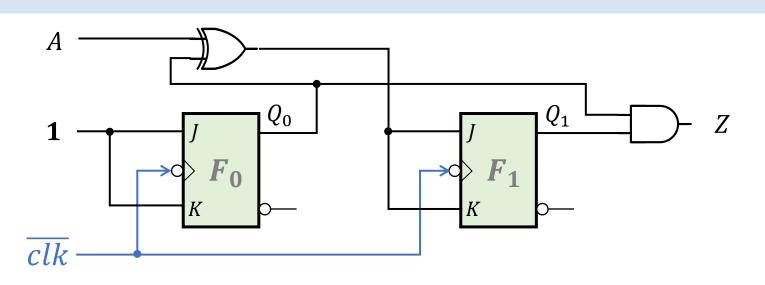
A = 0 时,递增计数 A = 1 时,递减计数

 $= A \oplus Q_0 \oplus Q_1$

$$\begin{aligned} \mathbf{Q}_{1}^{*} &= J_{1} \overline{Q}_{1} + \overline{K}_{1} Q_{1} \\ &= (A \oplus Q_{0}) \overline{Q}_{1} + \overline{A \oplus Q_{0}} Q_{1} \end{aligned}$$



【例2】JK触发器电路分析:特征表法



0.0	Q_1^*	$\boldsymbol{Q_1^*Q_0^*}$	
Q_1Q_0	A = 0	A = 1	Z
0 0	0 1	11	0
0 1	10	0 0	0
10	1 1	0 1	0
1 1	0 0	10	1

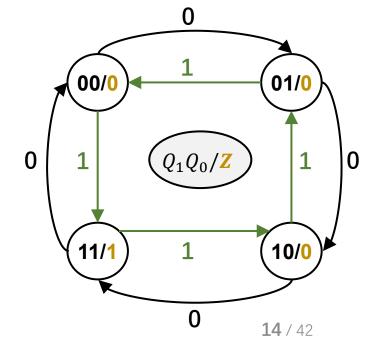
激励方程: $J_0 = K_0 = 1$

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

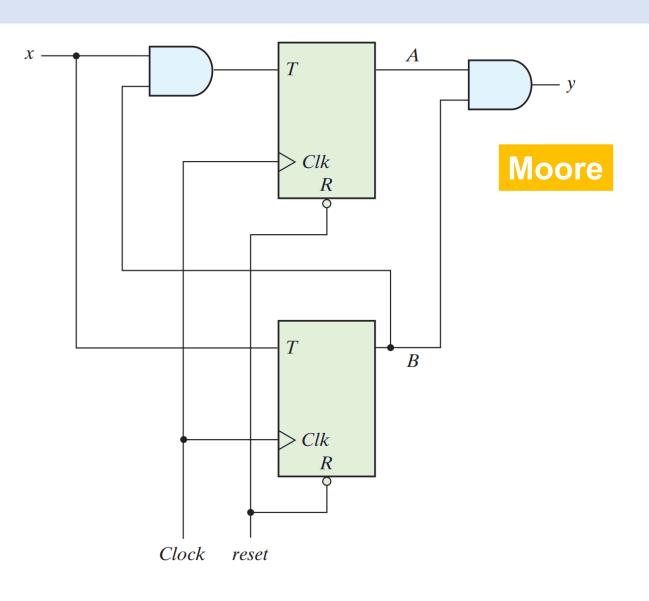
输出方程: $Z = Q_1Q_0$

特征表

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



【例3】**T触发**器电路分析



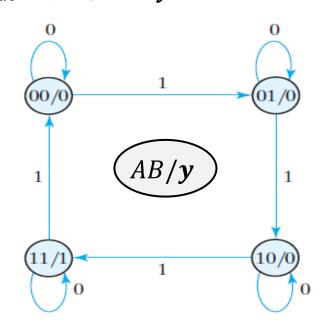
激励方程: $T_A = Bx$ $T_B = x$

特征方程: $Q^* = T \oplus Q$

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

$$\mathbf{B}^* = T_B \oplus B = x \oplus B$$

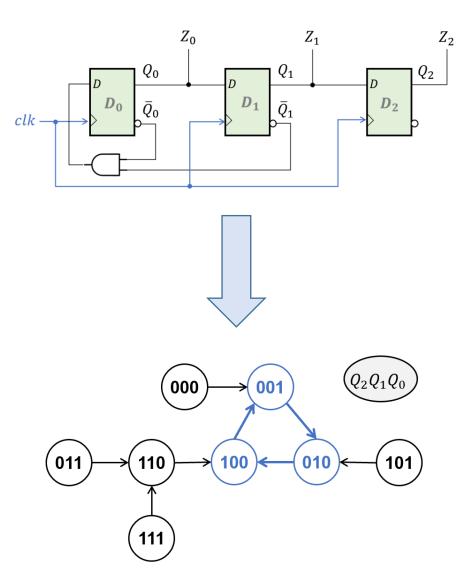
输出方程: y = AB



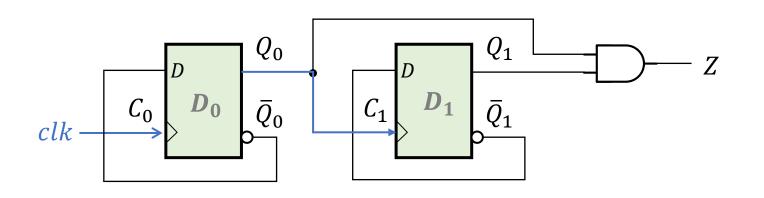
时序电路分析步骤

(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*



时钟方程: $C_0 = clk \uparrow$, $C_1 = Q_0 \uparrow \frown$ 即 Q_0 由0变1

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

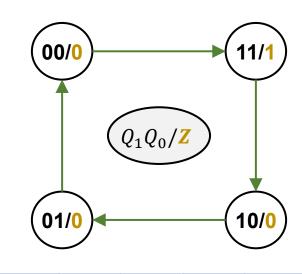
特征方程: $Q^* = D$

上升沿才能转换状态, 其余时间保持原态。

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

输出方程: $Z = Q_1Q_0$

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。

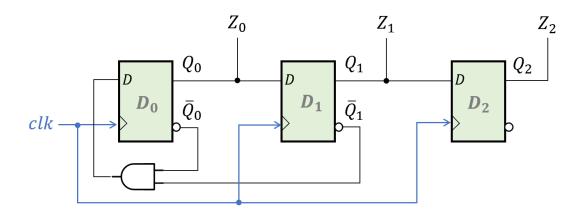


Q_1	Q_0	c_1	$\boldsymbol{\mathcal{C}_0}$	$oldsymbol{Q_1^*}$	$\boldsymbol{Q_0^*}$	Z
0	0	↑	1	1	1	0
0	1	\downarrow	↑	0	0	0
1	0	↑	↑	0	1	0
1	1	+	↑	1	0	1

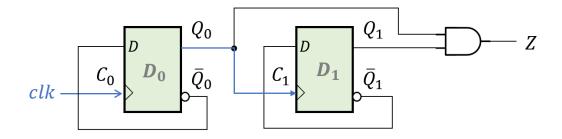
时序电路

同步时序电路、异步时序电路

• 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。

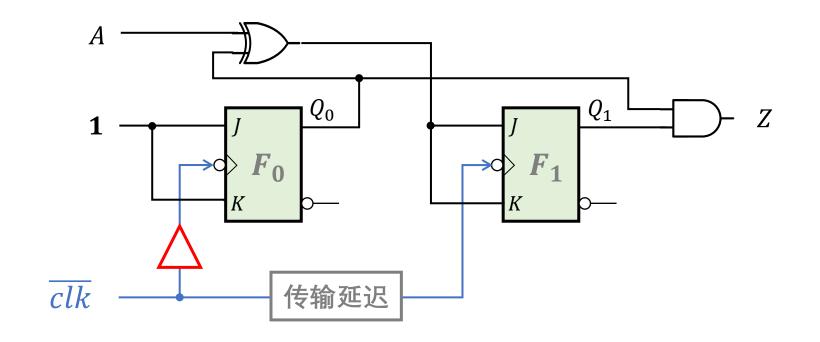


• **异步时序电路**: 各个触发器的时钟脉冲不同。既可用触发器、也可用锁存器。 电路状态的翻转有先有后。



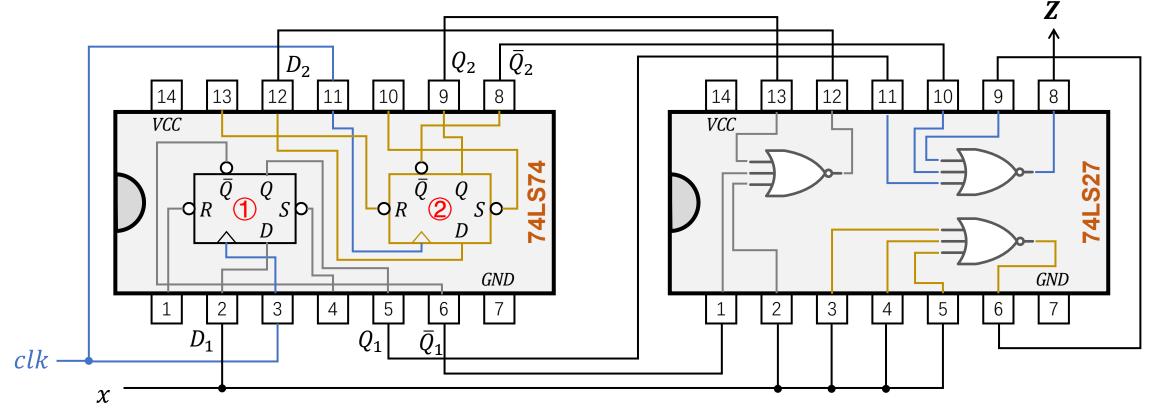
同步时序逻辑电路中的时钟偏移

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

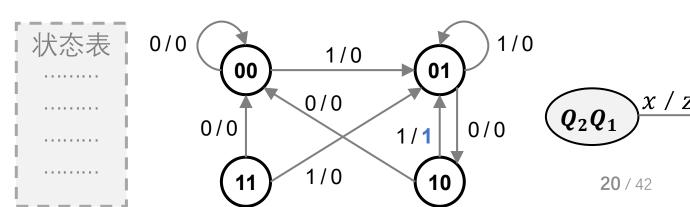


$$Z = \overline{\bar{x} + \bar{Q}_2 + Q_1} = xQ_2\bar{Q}_1$$

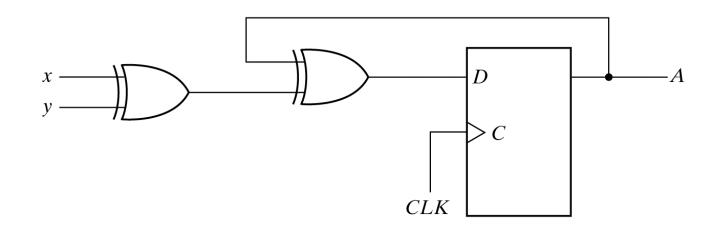
$$\mathbf{Q_1^*} = D_1 = x$$

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

Mealy



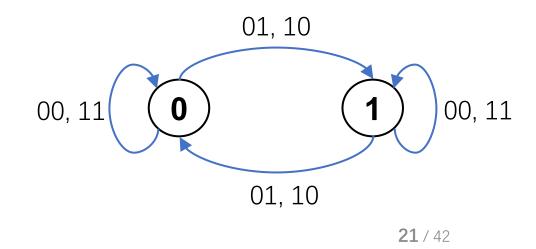
【练习2】时序电路分析



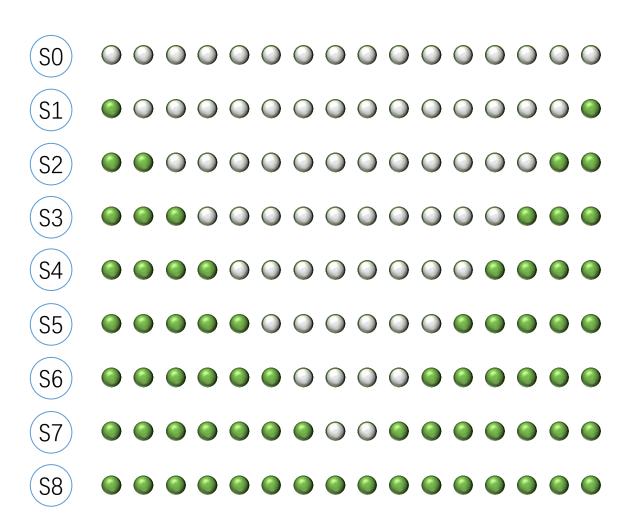
激励方程: $D = A \oplus x \oplus y$

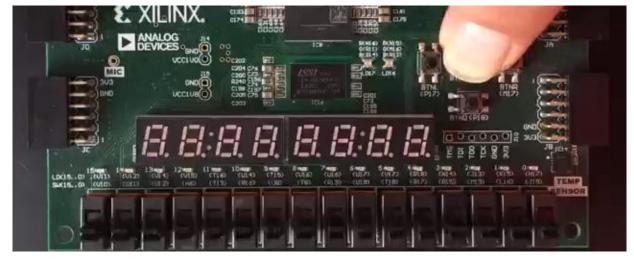
状态方程: $A^* = A \oplus x \oplus y$

Present			Next
state	Inputs		state
A	х	у	A^{\star}
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



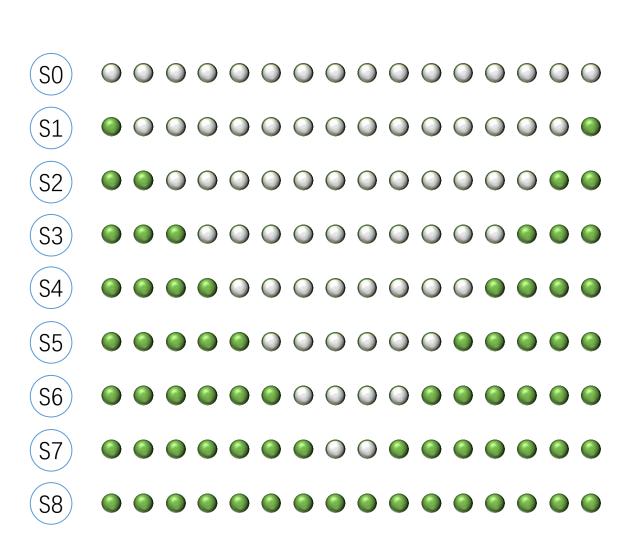
流水灯





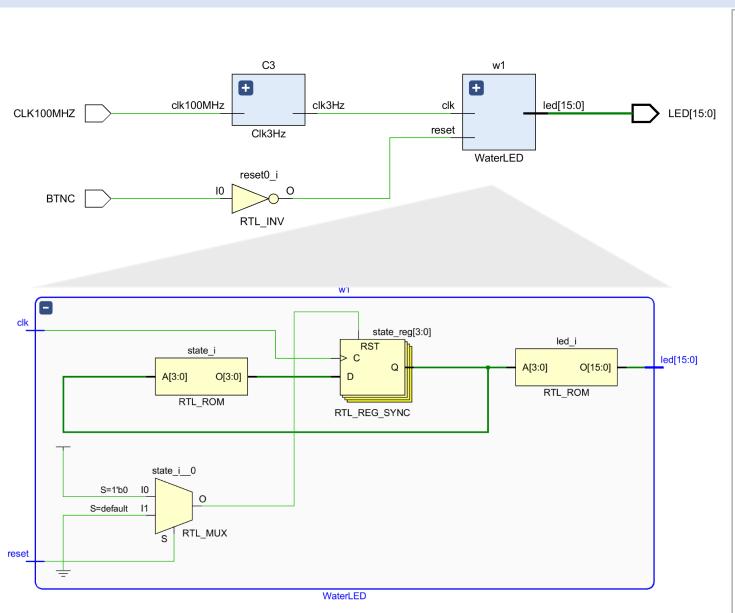
WaterLED

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;
 6
        always ff @(posedge clk)
            if (!reset) state <= S0:
            else case (state)
9
                    S0: state <= S1; S1: state <= S2; S2: state <= S3;
10
                    S3: state <= S4: S4: state <= S5; S5: state <= S6;
11
                    S6: state <= S7; S7: state <= S8; S8: state <= S0;
                    default: state <= S0:
13
14
                 endcase
15
        always comb
16
                             // 从两边往中间逐个亮
            case (state)
17
                S0 : led = 16' b0000 0000 0000 0000;
18
                S1 : led = 16' b1000 0000 0000 0001:
19
                S2 : led = 16' b1100 0000 0000 0011;
20
                S3 : led = 16' b1110 0000 0000 0111:
21
                S4 : led = 16' b1111 0000 0000 1111;
                S5 : led = 16'b1111 1000 0001 1111;
23
                S6 : led = 16'b1111 1100 0011 1111;
24
                S7 : led = 16' b1111 1110 0111 1111:
25
                S8 : led = 16'b1111 1111 1111 1111;
26
            default: led = 16'b0000 0000 0000 0000; //全灭
27
28
            endcase
29 ! endmodule
```

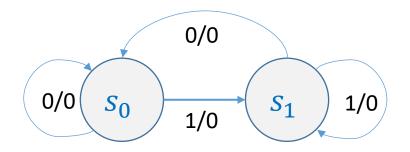
流水灯 原理图



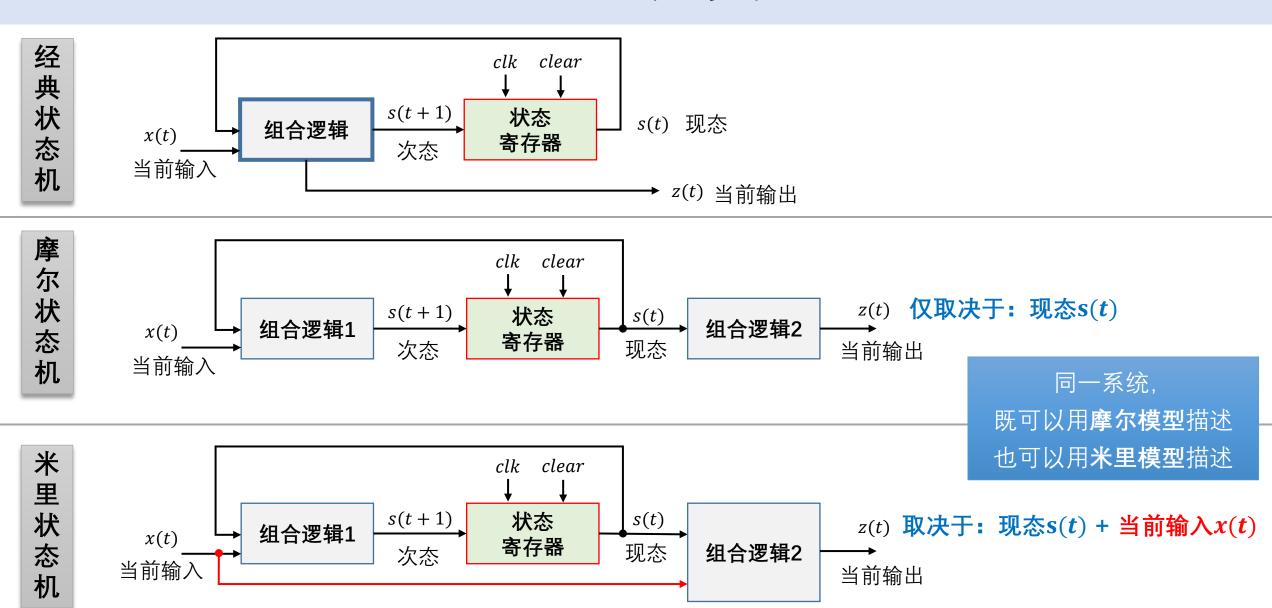
```
module WaterLED (input logic clk, reset,
                    output logic [15:0] led);
        parameter S0 = 0, S1 = 1, S2 = 2, S3 = 3, S4 = 4,
                    S5 = 5, S6 = 6, S7 = 7, S8 = 8;
 5
        logic [3:0] state;
6
7
        always ff @(posedge clk)
            if (!reset) state <= S0;
 8
            else case (state)
 9
                    S0: state <= S1; S1: state <= S2; S2: state <= S3;
10
                    S3: state <= S4; S4: state <= S5; S5: state <= S6;
11
                    S6: state <= S7; S7: state <= S8; S8: state <= S0;
12
                    default: state <= S0:
13
                 endcase
14
15
        always_comb
16
                            // 从两边往中间逐个亮
            case (state)
17 :
                S0 : led = 16' b0000 0000 0000 0000;
18
                S1 : led = 16' b1000 0000 0000 0001;
19
                S2 : led = 16' b1100_0000_0000_0011;
20
                S3 : led = 16' b1110 0000 0000 0111;
21
                S4 : led = 16' b1111 0000 0000 1111:
22
                S5 : led = 16' b1111 1000 0001 1111;
23
                S6 : led = 16' b1111 1100 0011 1111:
24
                S7 : led = 16' b1111 1110 0111 1111;
25
                S8 : led = 16'b1111 1111 1111 1111;
26
            default: led = 16'b0000 0000 0000 0000; //全灭
27
            endcase
28
                                                    25 / 42
29 : endmodule
```

有限状态机 Finite State Machine

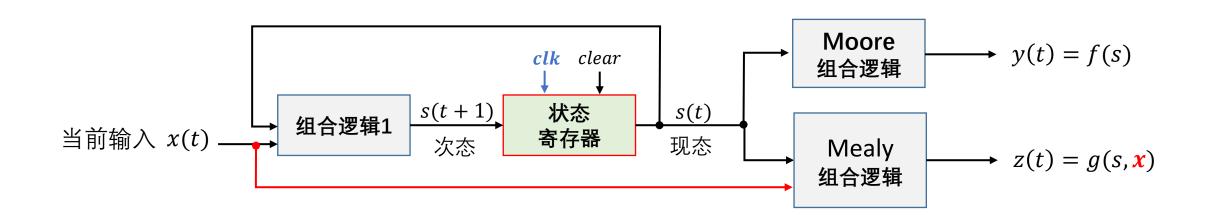
- 电路设计经典方法,尤其适于设计控制模块,易于FPGA实现。
- 用于:对**有限内部状态相互转换的系统**进行建模。
- 常为同步时序, 在时钟信号的触发下完成各状态之间的转换, 并产生相应的输出。
- 由组合逻辑(状态译码、产生输出信号) + 时序逻辑(存储状态)构成。
- **状态机**表示方法: **状态图**、状态表、流程图。三者等价。



FSM 分类



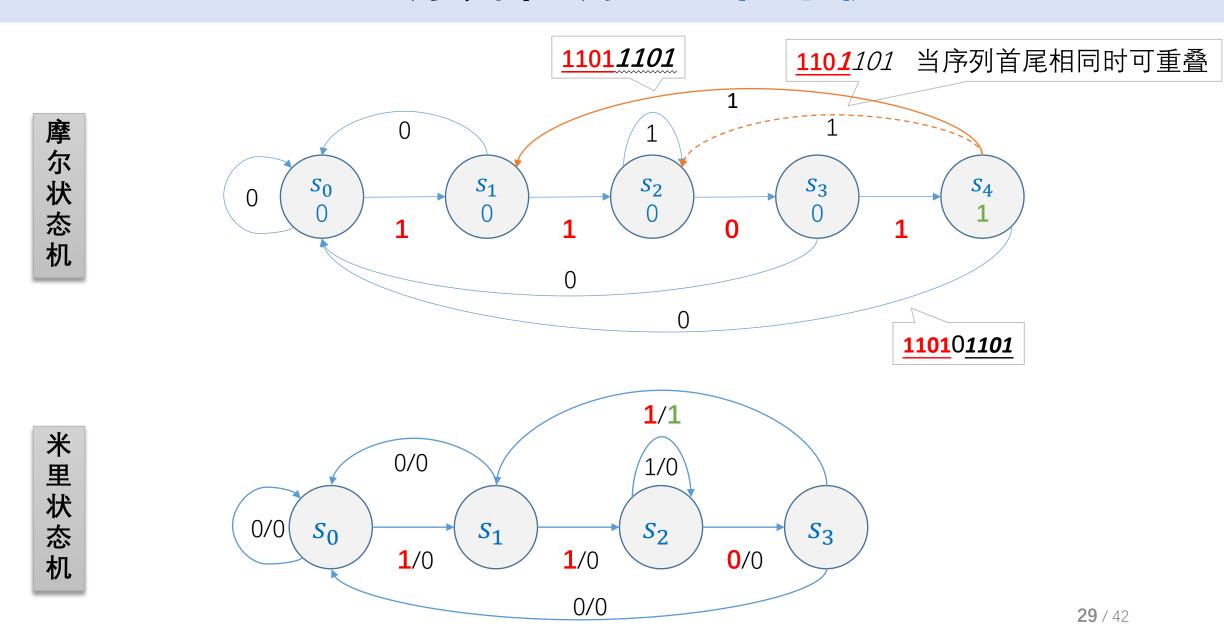
复杂系统:混合编程



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

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

1101序列检测器: 状态机



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 <u>或</u> ② 、① + ③ 各用一个always;
 - **一段式**: <u>1 + 2 + 3</u> 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。

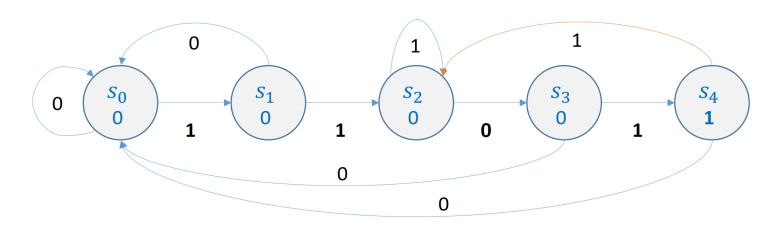
```
output logic Dout );
parameter S0=3' b000, S1=3' b001, S2=3' b010,
           S3=3'b011, S4=3'b100; // 状态
logic [2:0] present state, next state;
                                   组合逻辑1
always comb
              // 次态
    case (present state)
        S0: if(Din==1) next_state = S1;
            else
                       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;</pre>
                                   组合逻辑2
                   // 输出逻辑
always comb
    if(present_state==S4)
                          Dout = 1;
    else
                           Dout = 0:
```

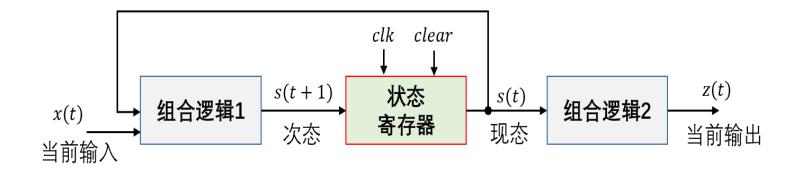
module Detect1101 Moore (input logic clk, clr, Din,

1101序列检测器 Moore

FSM

三段式描述





9

10

13

14

15

16

17

18

19

2021

22

23

24

25

26

27

28

```
module Detect1101_Mealy (input logic clk, clr,
                                                           1101序列检测器 Mealy
                          logic Din,
                     input
                     output logic Dout ); // reg!!
   parameter S0=2'b00, S1=2'b01, S2=2'b10, S3=2'b11;
                                                                                        FSM
   logic [1:0] present state, next_state; //reg
                                          组合逻辑1
                                                          三段式描述
                        // 次状态
   always comb
       case (present state)
                                                 (1)
                                                                                                  1/1
          S0: if(Din==1) next state = S1;
              else
                       next state = S0;
                                                                            0/0
                                                                                                   1/0
          S1: if(Din==1) next state = S2;
              else
                       next_state = S0;
                                                              0/0
                                                                    S_0
                                                                                   S_1
                                                                                                  S_2
                                                                                                                  S_3
          S2: if (Din==0)
                       next state = S3:
                                                                            1/0
                                                                                           1/0
                                                                                                          0/0
              else
                       next state = S2;
          S3: if (Din==1)
                       next state = S1;
                                                                                            0/0
              else
                       next state = S0;
          default:
                       next state = S0;
       endcase
                                                                                      clk clear
                                         状态寄存器
   always_ff @(posedge clk, posedge clr)
                                                                             s(t+1)
                                                                                       状态
                                                                                                 s(t)
                                                                                                                   z(t)
                                                                   组合逻辑1
       if(clr==1) present state <= S0;
                                                          x(t)
                                                                                      寄存器
                                                                                                现态
                                                                                                      组合逻辑2
                                                                              次态
       else
                present state <= next state;</pre>
                                                        当前输入
                                                                                                                 当前输出
   always ff @(posedge clk)
                               // 输出逻辑
                                           组合逻辑2
                                       Dout <= 1;
Dout <= 0;
       if((present state==S3) && (Din==1))
                                                       s_3的输出Dout需要寄存,故用时序电路!
       else
                                                                                                            32 / 42
endmodule
```

8 :

9 :

10

11 :

13 ;

14 ;

15

16

17 ; 18 ;

19 ;

20

21

23

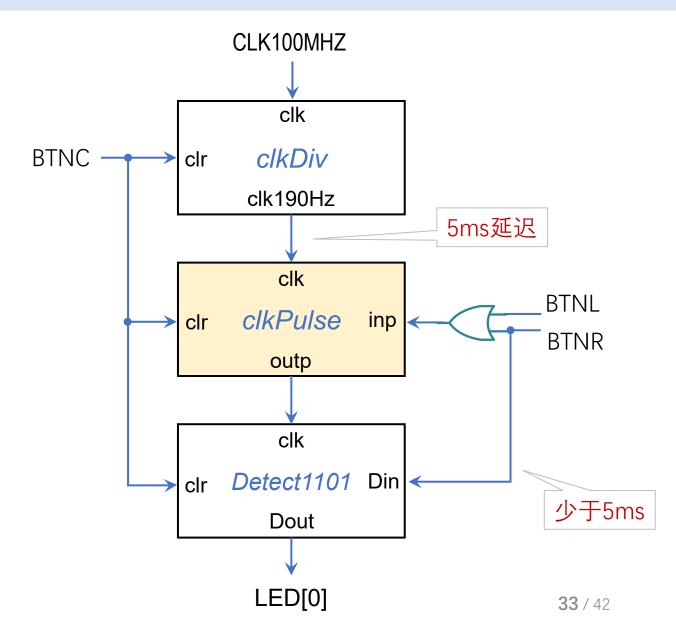
24

25

FSM

1101序列检测器: 板级验证

```
module DetectSequence_Top(
               logic CLK100MHZ,
         input
               logic BTNC,
                             // clr
         input
               logic BTNL,
         input
                logi'e, BTNR,
         input
        output logic [0:0] LED);
         logic clr, clk190Hz, clkp, btn01;
 9
        assign clr = BTNC;
         assign btn01 = BTNL | BTNR;
12
         clkdiv U1(.mclk(CLK100MHZ), .clr(clr),
13
                   .clk190Hz(clk190Hz));
14:
15
         clock_pulse U2(.inp(btn01), .cclk(clk190Hz),
16
                        . clr(clr),
                                     . outp(c1kp));
17
18
         Detect1101 Mealy D1(.clk(clkp),
19
                             .clr(clr),
20
                             . Din (BTNR),
21
                             . Dout (LED[0]));
    endmodule
```



```
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;
                                 ext state = 🔀
                               net state
                S3: if (lin=1)
                                  xt state = S0:
                                next state = S2;
                     else
                default:
                               next state = S0;
            endcase
        always ff @(posedge clk, posedge clr)
            if(clr==1) present state <= S0;
            else
                       present state <= next state;</pre>
                           // 输出逻辑
        always comb
            if(present_state==S4) Dout = 1;
                                   Dout = 0;
            else
30 ; endmodule
```

13

14

15

16 :

17 :

18

19

20

21

23

24

25

26

27

28

29

1101序列检测器 Moore-2

```
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;
                                                                        两段式描述
        logic [2:0] present state; //next state;
        always_ff @(posedge clk, posedge clr) 合并
 8
             if(c1r==1)
 9
                                        present state <= S0;
                    case (present_state)
             else
10
                         S0: if(Din==1) present state <= S1;
11
12
                             else
                                        present state <= S0;</pre>
                         S1: if(Din==1) present state <= S2;
13
                             else
                                        present state <= S0;</pre>
14
                                                                           S_3
                                                                                          S_4
15
                         S2: if (Din==0) present state <= S3;
                                                                            0
                                                                    0
                                        present state <= S2;</pre>
                             else
16
                         S3: if(Din==1) present state <= S4;
17
                             else
                                        present state <= S0;
18
                         S4: if (Din==0) present_state <= S0;
19
                             else
                                        present state <= S2:
20
                         default:
                                        present state <= S0;</pre>
21
                     endcase
23
        always comb
                            // 输出逻辑
                                                  组合逻辑2
24
             if(present state==S4) Dout = 1;
25
                                    Dout = 0:
26
             else
                                                                             34 / 42
```

endmodule

```
module Detect1101_Moore (input logic clk, clr, Din,
                                                                  1101序列检测器 Moore-3
                            output logic Dout );
        parameter S0=3' b000, S1=3' b001, S2=3' b010,
                   S3=3' b011, S4=3' b100: // 状态
                                                                module D1101 Moore2 2 (input logic clk, clr, Din,
        logic [2:0] present state, next state;
                                                                                                                                    1
                                                                                     output logic Dout ):
                                                                    parameter S0=3' b000, S1=3' b001, S2=3' b010,
        always comb // 次态
                                                                              S3=3' b011, S4=3' b100; // 状态
                                                                                                                                   0
                                                            5
            case (present state)
                                                                    logic [2:0] present_state, next_state;
                                                            6
                S0: if(Din==1) next state = S1;
                    else
                              next state = S0;
                                                                    always ff @(posedge clk, posedge clr)
                                                            8
                S1: if(Din==1) next state = S2;
                                                                        if (clr==1) present state \leq S0;
                                                            9
                    else
                              next state = S0;
                                                                                  present_state <= next_state;</pre>
                                                                        else
                                                           10
                S2: if (Din==0) next_state = S3;
                                                           11
                    else
                              next state = S2;
                                                                    always comb
                                                           12
                S3: if(Din==1) next state = S4;
                                                                        case (present state)
                                                           13
                    else
                              next state = S0;
                                                                           S0: if (Din==1) begin next state = S1; Dout = 0; end
                S4: if (Din==0) next state = S0;
                                                           14
                                                                                else
                                                                                          begin next state = S0; Dout = 0; end
                                                           15
                    else
                              next state = S2:
                                                                           S1: if (Din==1) begin next state = S2; Dout = 0; end
                default:
                                                           16
                              next state = S0;
                                                                               else
                                                                                          begin next state = S0; Dout = 0; end
                                                           17
            endcase
                                                                           S2: if (Din==0) begin next state = S3; Dout = 0; end
                                                           18
                                                                                else
                                                                                          begin next state = S2; Dout = 0; end
        always ff @(posedge clk, posedge clr)
                                                           19
                                                                           S3: if (Din==1) begin next state = S4; Dout = 0; end
                                                           20
            if(clr==1) present state <= S0;
                                                                                else
                                                                                          begin next state = S0; Dout = 0; end
                                                           21
                       present state <= next state;</pre>
            else
                                                                           S4: if (Din==0) begin next state = S0; Dout = 1; end
                                                           22
                                                                                else
                                                                                          begin next_state = S2; Dout = 1; end
                           // 输出逻辑
                                                           23
        always comb
                                                                           default:
                                                                                          begin next state = S0; Dout = 0; end
            if(present_state==S4) Dout = 1;
                                                           24
                                                           25
                                                                        endcase
                                  Dout = 0;
            else
                                                                                                                                    35 / 42
                                                                endmodule
30 | endmodule
```

两段式描述

8 :

9

10 11

12

13

14

15

16 :

17 :

18

19

20

21

22

23

24

25

26

27

28

```
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;
                 // 次态
    always comb
        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;</pre>
                       // 输出逻辑
    always comb
        if(present_state==S4) Dout = 1;
                               Dout = 0;
        else
endmodule
```

10

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

1101序列检测器 Moore-4

module D1101 Moore3 (input logic clk, clr, Din,

9

10

12

13

14

15

16

18 :

19

20

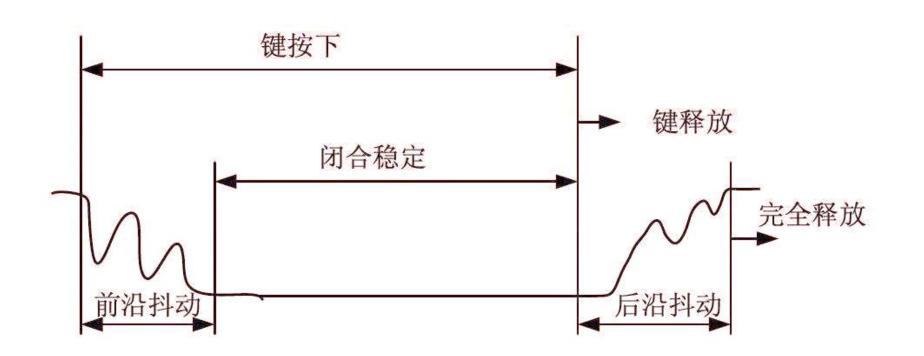
21

23

```
output logic Dout );
    parameter S0=3' b000, S1=3' b001, S2=3' b010,
                                                                   0
                                                            0
               S3=3' b011, S4=3' b100; // 状态
    logic [2:0] present state; // next state;
   always ff @(posedge clk, posedge clr)
                                                                              段式
                           begin present_state <= S0; Dout <= 0; end
        if(c1r==1)
                case (present state)
        else
            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</pre>
            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 < $\frac{1}{2}$ 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
                                                                     36 / 42
endmodule
```

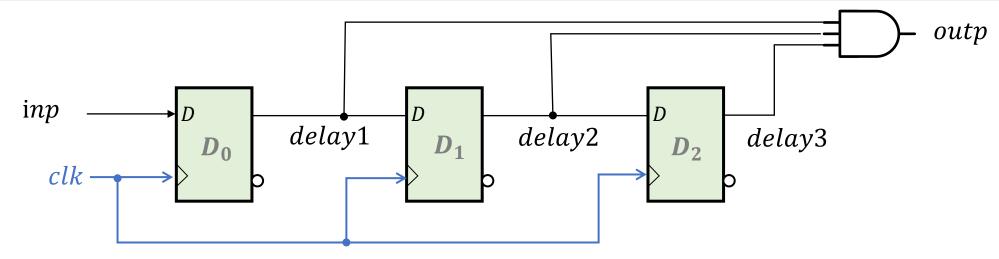
机械抖动

- 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!

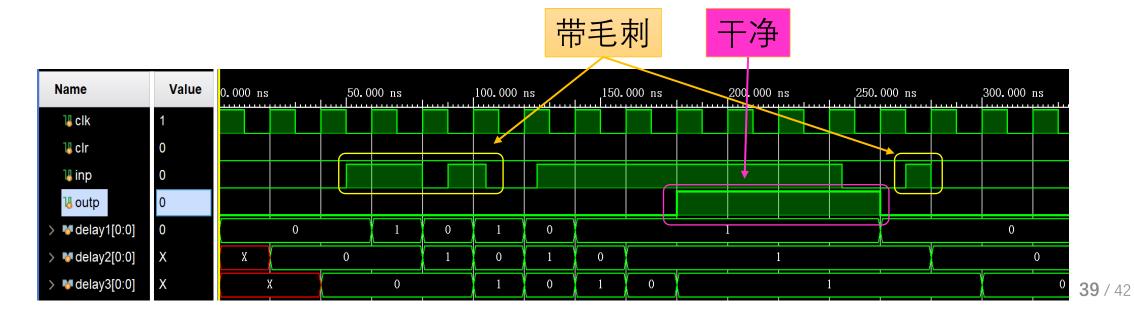




去抖电路



【注】时钟要足够慢。保证抖动在3个时钟周期内!如 clk_190Hz

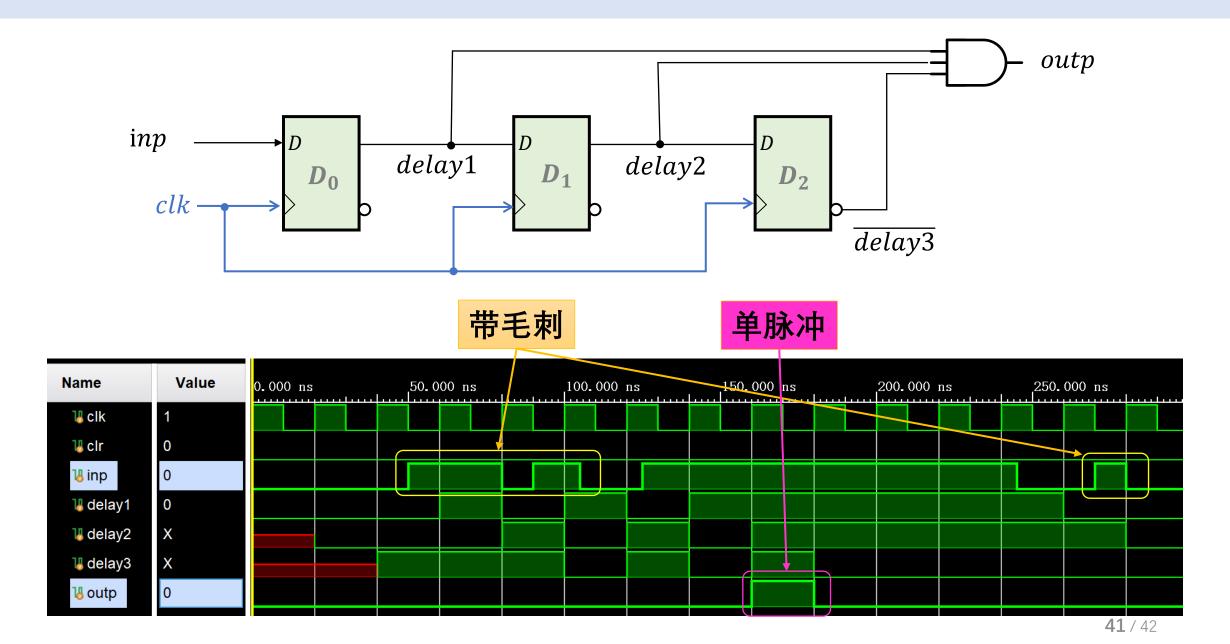


Debounce

去抖代码

```
module Debounce #(parameter N = 1)
                                                                                                                                                      outp
          (input logic clk, clr,
            input logic [N-1:0] inp,
                                                              inp
            output logic [N-1:0] outp);
                                                                                        delay1
                                                                                                             delay2
                                                                                                                                    delay3
                                                                                                                            D_2
                                                                               D_0
          logic [N-1:0] delay1, delay2, delay3;
          always_ff @(posedge clk, posedge clr)
 9
                                                                                                                         1 timescale lns / lps
                                                                                                                            module Debounce_Sim( );
10
          begin
                                                                                                                               logic clk, clr, inp, outp;
               if(clr) begin
11
                                                                                                                               // 实例化
                                                                                                                               Debounce D1(clk, clr, inp, outp);
                    delav1 \le 0:
                                                     Name
                                                                 Value
                                                                       0.000 ns
                                                                                     50.000 ns
                                                                                                    100.000 ns
                                                                                                                  150.000 n
                    delay2 \le 0;
13
                                                                                                                               always //时钟
                                                      clk
                    delay3 \le 0:
                                                                                                                               begin
14
                                            end
                                                                                                                                   c1k = 1; #10; c1k = 0; #10;
                                                      clr
               else begin
15
                                                                                                                        10
                                                                                                                                end
                                                      inp
                    delay1 <= inp;
                                                                                                                        11
16
                                                      ₩ outp
                                                                                                                        12
                                                                                                                               initial //输入值
                    delay2 <= delay1;</pre>
                                                                                                                        13
                                                                                                                               begin
                    delay3 <= delay2;
                                            end
18
                                                                                                                                   clr = 0; inp = 0;
                                                                                                                        14
                                                                                                                                   #50; inp = 1; #30; inp = 0;
                                                                                                                         15
19
          end
                                                                                                                                   #10; inp = 1; #15; inp = 0;
                                                                                                                        16
20
                                                                                                                                   #20; inp = 1; #120; inp = 0;
                                                                                                                        17
                                                                                                                                   #25; inp = 1; #10; inp = 0;
                                                                                                                         18
          assign outp = delay1 & delay2 & delay3;
                                                                                                                        19
                                                                                                                                end
     endmodule
                                                                                                                        20 : endmodule
```

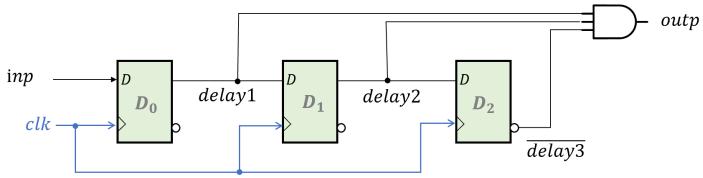
时钟脉冲:按一次键,产生一个单脉冲



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 \le 0;
                delay2 \le 0;
                delav3 \le 0:
                                      end
            else begin
                 delay1 <= inp;
                 delay2 <= delay1;
15
                 delay3 <= ~delay2;
16
                                      end
        end
18
        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
            c1k = 1; #10; c1k = 0; #10;
9
10
        end
11 ;
        initial //輸入值
12 ;
13
        begin
            c1r = 0; inp = 0;
14 :
            #50; inp = 1; #30; inp = 0;
15
            #10; inp = 1; #15; inp = 0;
16
17
            #20; inp = 1; #120; inp = 0;
            #25; inp = 1; #10; inp = 0;
18
        end
19
20 | endmodule
```

42 / 42