4 时序逻辑电路建模

4.1 锁存器

4.2 时序电路建模基础

4.3 触发器

4.4 寄存器和移位寄存器

4.5 同步计数器

4.6 Verilog HDL函数与任务的使用

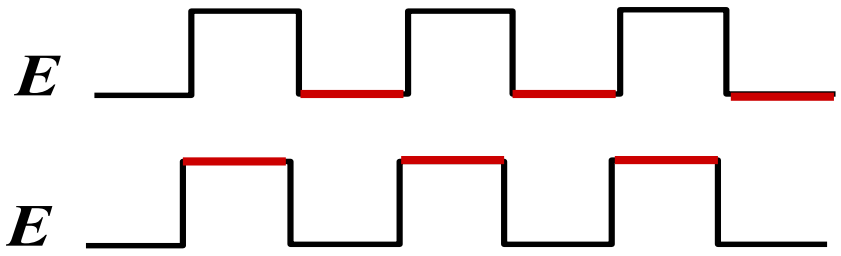
4.7 m序列码产生电路设计

锁存器和触发器的基本特性

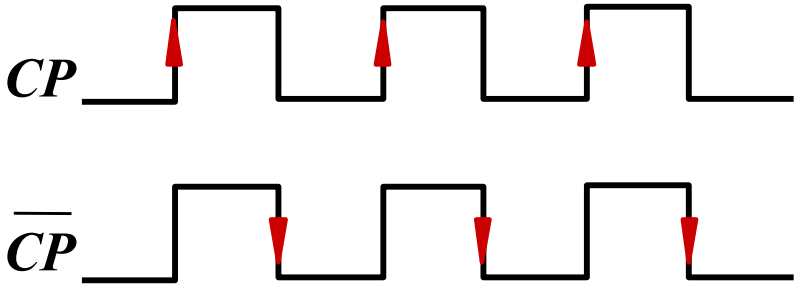
- 锁存器和触发器是构成时序逻辑电路的基本逻辑单元，它们具有存储数据的功能。
- 每个锁存器或触发器都能存储1位二值信息，所以又称为存储单元或记忆单元。
- 若输入信号不发生变化，锁存器和触发器必然处于其中一种状态，且一旦状态被确定，就能自行保持不变，即长期存储1位二进制数。
- 电路在输入信号的作用下，会从一种稳定状态转换成为另一种稳定状态。

锁存器与触发器的区别

锁存器—— 没有时钟输入端，**对脉冲电平敏感的存储电路，在特定输入脉冲电平作用下改变状态。**



触发器——每一个触发器有一个时钟输入端。**对脉冲边沿敏感的存储电路，在时钟脉冲的上升沿或下降沿的变化瞬间改变状态。**



4.1 锁存器

1 0

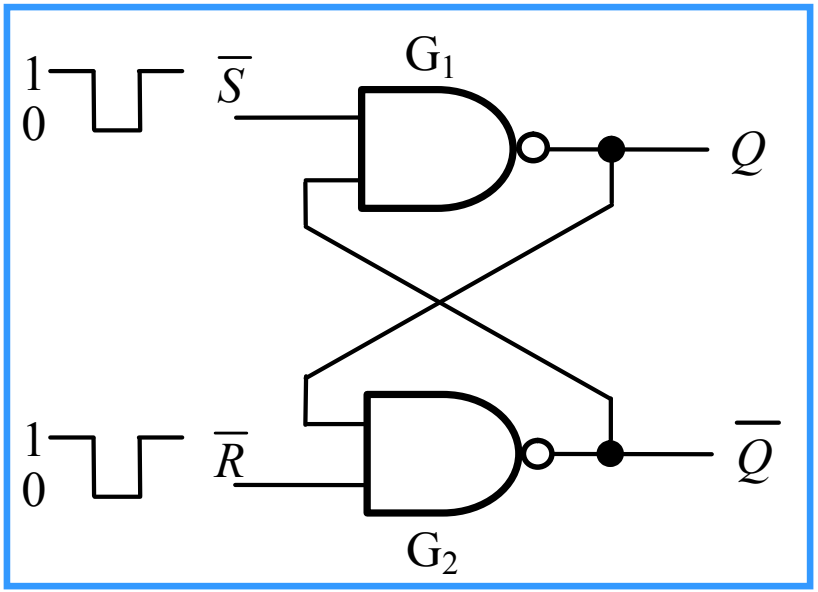
4.1.1 基本SR锁存器

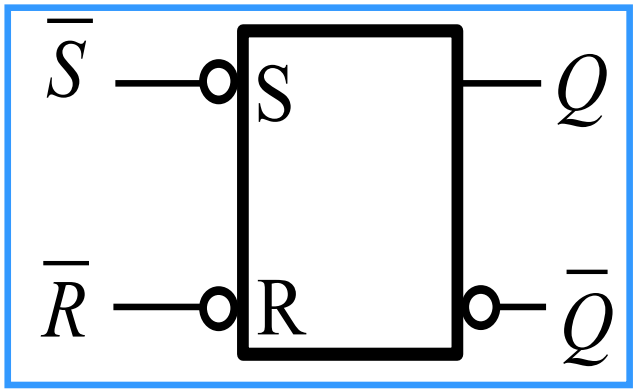
4.1.2 门控D锁存器

4.1.3门控D锁存器的Verilog HDL建模

4.1.1 基本*SR*锁存器

1. 用与非门构成的基本*SR*锁存器



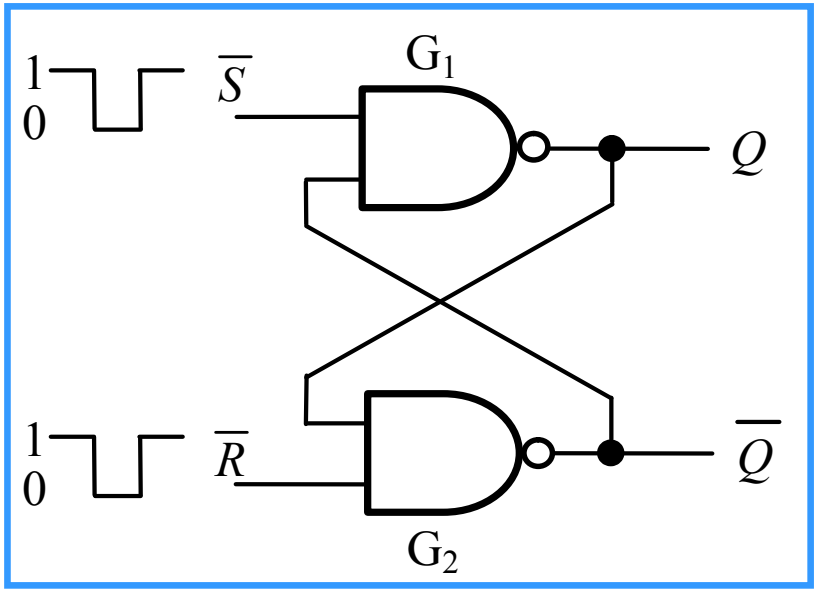


国标逻辑符号

方框外侧输入端的小圆圈和信号名称上面的小横线均表示输入信号是低电平有效的，同时为了区别，这种锁存器有时也称为基本 $\overline{S}\,\overline{R}$ 锁存器。

4.1.1 基本*SR*锁存器

1. 用与非门构成的基本*SR*锁存器

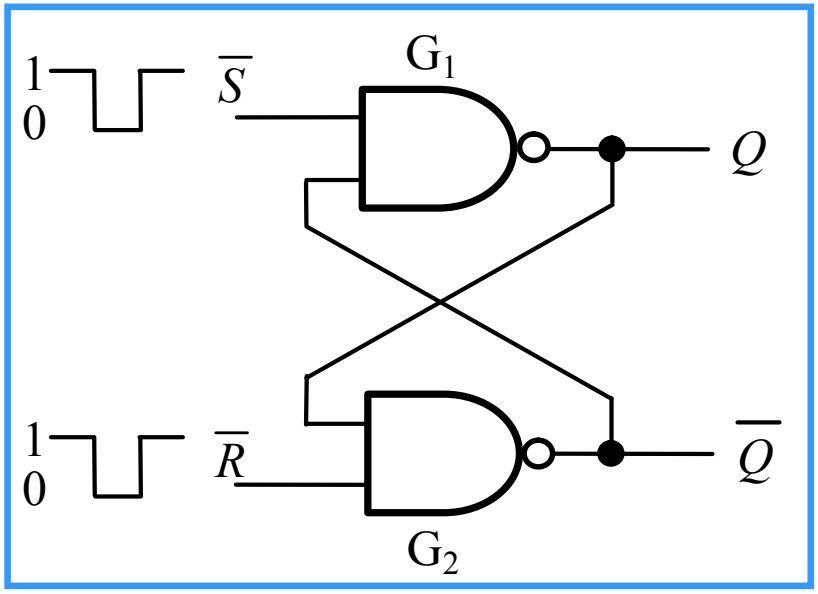


现态：$\overline{R}$、$\overline{S}$信号作用前 Q端的状态，现态用Q^n表示。

次态：$\overline{R}$、$\overline{S}$信号作用后 Q 端的状态，次态用Q^{n+1}表示。

4.1.1 基本*SR*锁存器

a.电路图



b.功能表

$\overline{R}$	$\overline{S}$	Q	$\overline{Q}$
1	1	不变	不变
1	0	1	0
0	1	0	1
0	0	1	1

约束条件： $\overline{S}+\overline{R}=1$

例 当$\overline{S}$、$\overline{R}$的波形如下图虚线上边所示，试画出Q和 $\overline{Q}$对应的波形（假设原始状态$Q=0$）。

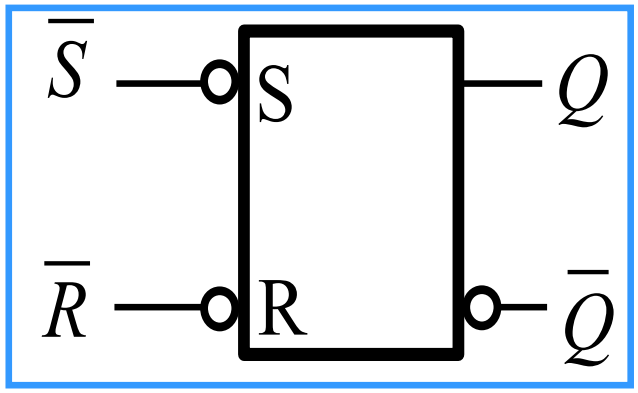
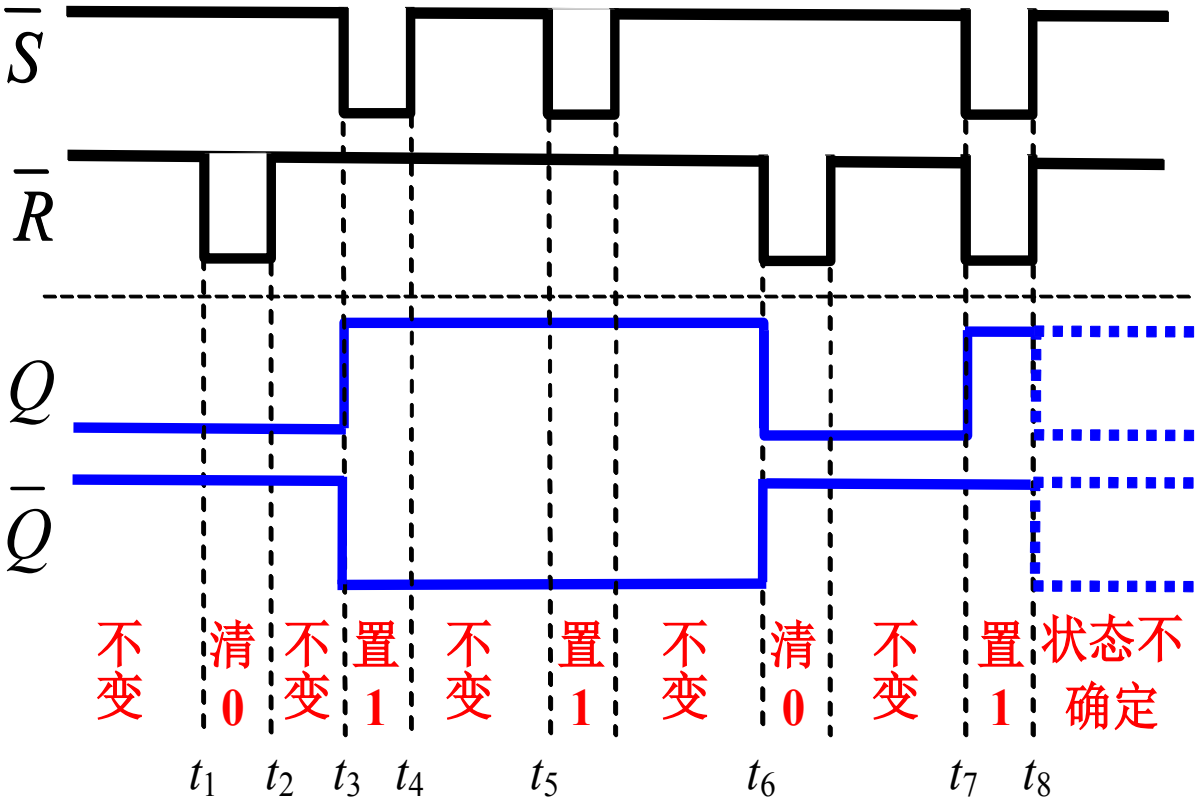


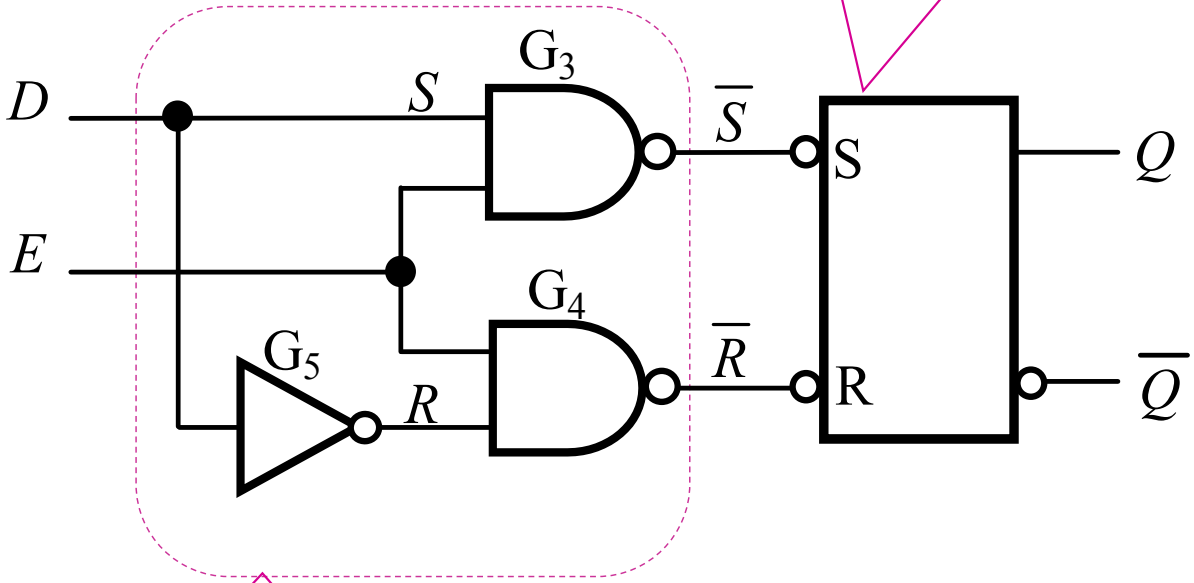
图4.1.1（b）



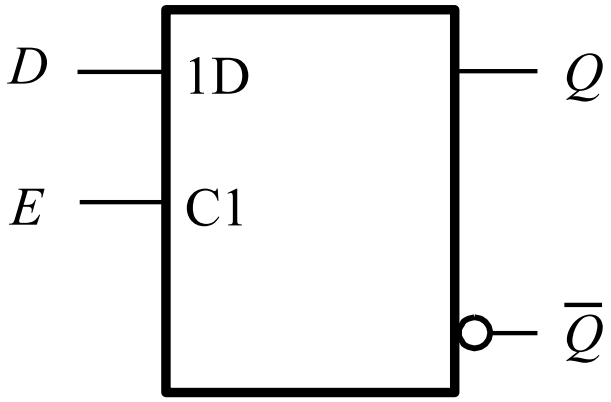
4.1.2 门控D锁存器

1. 电路结构

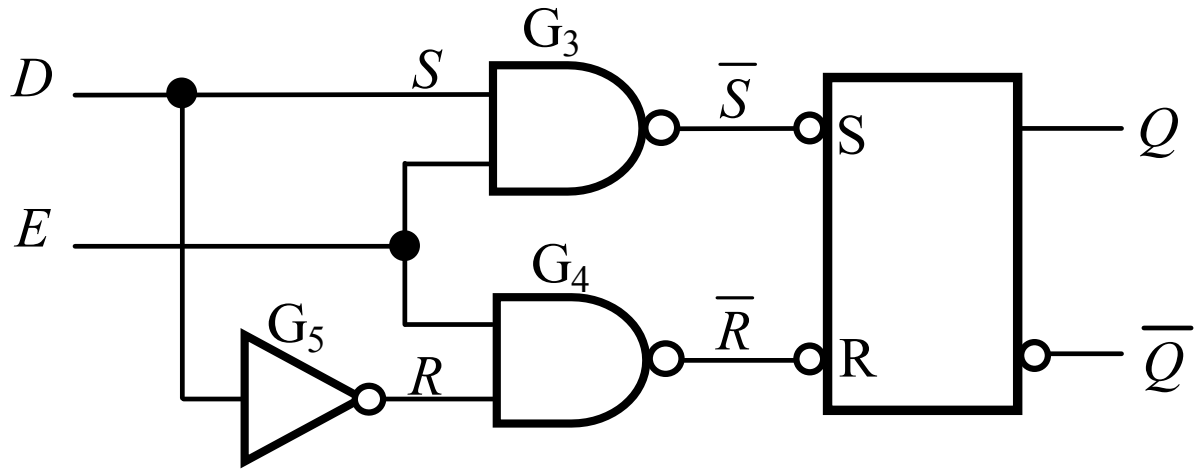
简单$\overline{S}$ $\overline{R}$锁存器



使能信号控制门电路



国标逻辑符号



- 当$E=0$时，$\overline{S}=\overline{R}=1$，无论$D$取什么值，$Q$保持不变。
- 当$E=1$时，
  - D=1时，$\overline{S}$=0，$\overline{R}=1$，Q被置1；
  - D=0时，$\overline{S}$=1，$\overline{R}=0$，Q被置0。
- 在E=1期间，D值将被传输到输出端Q，而当E由1跳变为0时，锁存器将保持跳变之前瞬间D的值。因此，D锁存器常被称为**透明锁存器**（Transparent Latch）。

特性表和特性方程

表 4.2.4　D 锁存器的特性表

E	D	Q^n	Q^{n+1}	功　能
0	×	0	0	保持
0	×	1	1	
1	0	0	0	置 0
1	0	1	0	
1	1	0	1	置 1
1	1	1	1	

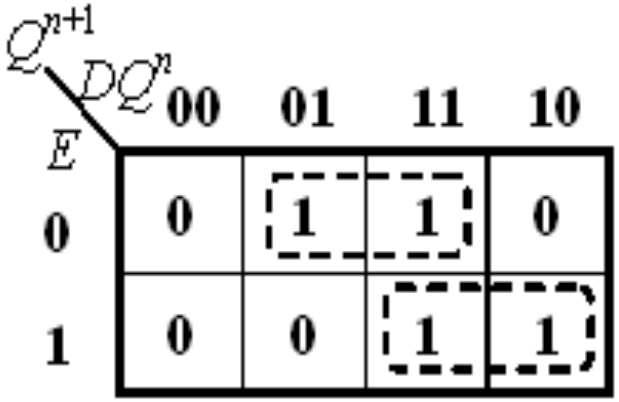
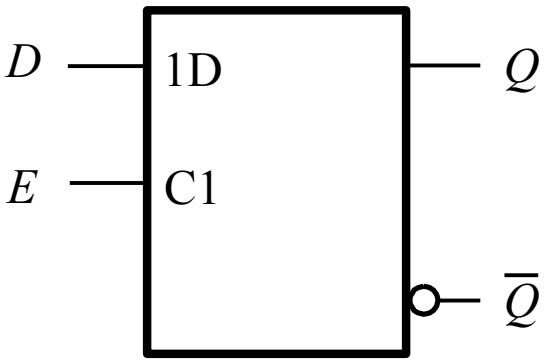


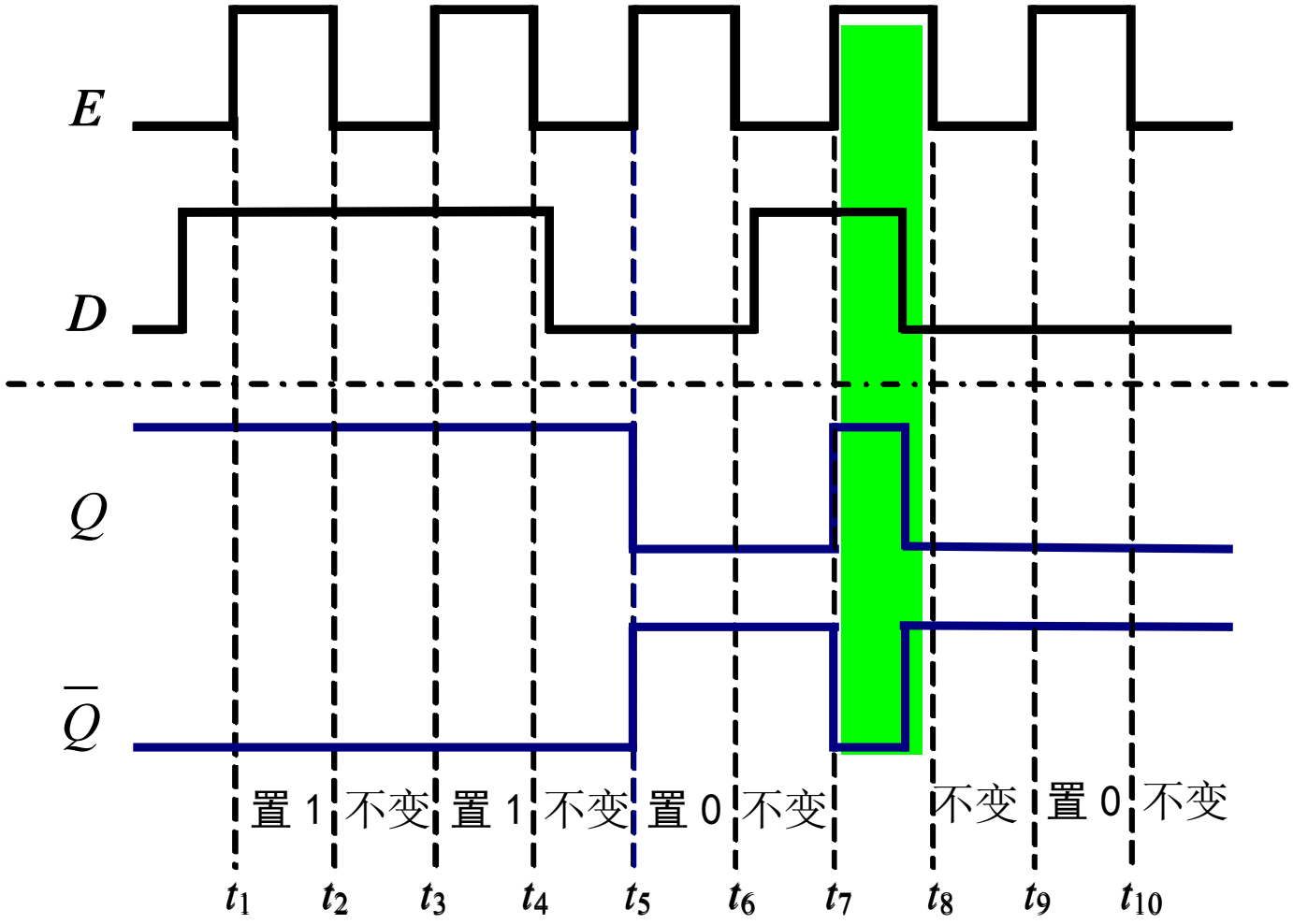
图 4.2.8　卡诺图

$$Q^{n+1} = \overline{E} \cdot Q + E \cdot D$$

波形图


D
1D
Q
E
C1
$\overline{Q}$

初始状态为$Q=1$


E
D
Q
$\overline{Q}$
置 1
不变
置 1
不变
置 0
不变
不变
置 0
不变
t_1
t_2
t_3
t_4
t_5
t_6
t_7
t_8
t_9
t_{10}

4.1.3 门控*D* 锁存器的Verilog HDL建模

1. 试对图4.1.3所示的D锁存器进行建模。

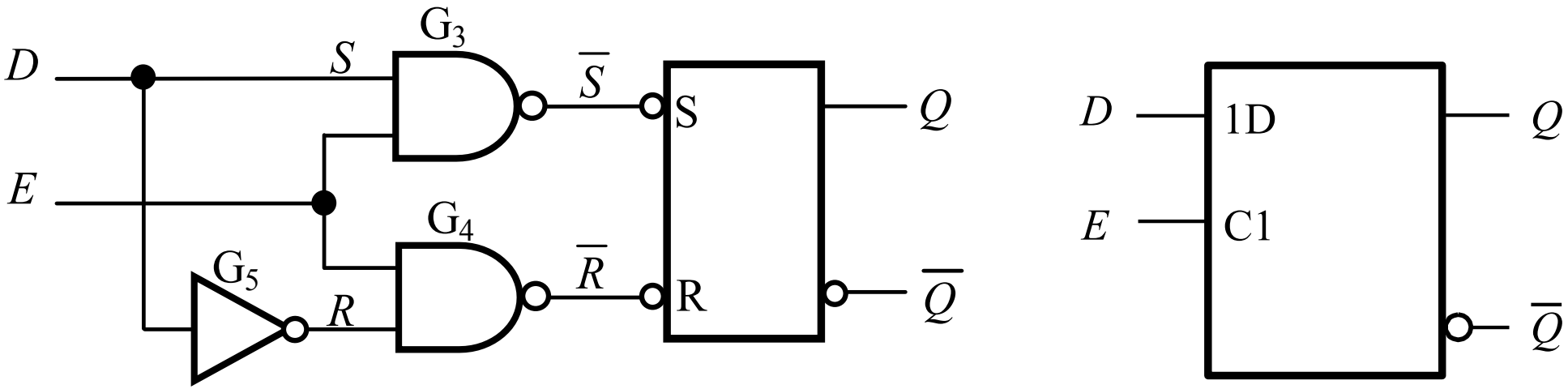


图4.1.3　　D锁存器

```
//版本1: Structural description of a D latch
module Dlatch_Structural (E, D, Q, Q_);
     input E, D;
     output Q, Q_;
     wire R_, S_;
          nand N1(S_, D,E);
          nand N2(R_,~D,E);
          SRlatch_1 N3(S_,R_,Q,Q_);
endmodule
```

```
//Structural description of a SR-latch
module SRlatch_1 (S_,R_, Q, Q_);
     input S_,R_;
     output Q, Q_;
     nand N1(Q, S_,Q_);
     nand N2(Q_,R_,Q);
endmodule
```

版本1的特点：

- 第一个版本根据图4.1.3使用基本的逻辑门元件，采用结构描述风格，编写了两个模块，这两个模块可以放在一个文件中，文件名为Dlatch_Structural.v。
- 在一个文件中可以写多个模块，其中有一个是主模块（或者称为顶层模块）。
- 文件名必须使用顶层模块名。本例中Dlatch_Structural是主模块，它调用SRlatch_1模块。

```
//版本2: Behavioral description of a D latch
module Dlatch_bh (E, D, Q, Q_);
    input E, D;
    output Q,Q_;
    reg Q;
    assign Q_ = ~Q;
    always @(E or D)
        if (E)
            Q <= D;   //当使能有效时，输出跟随输入变化
        else
            Q <= Q;   //保持不变
endmodule
```

版本2的特点：

- 第二个版本采用功能描述风格的代码，不涉及到实现电路的具体结构，靠“算法”实现电路操作。对于不太喜欢低层次硬件逻辑图的人来说，功能描述风格的Verilog HDL是一种最佳选择。其中“<=”为非阻塞赋值符，将在下一节介绍。
- 注意：
  - always内部不能使用assign。
  - 在写可综合的代码时，建议明确地定义if－else中所有可能的条件分支，否则，就会在电路的输出部分增加一个电平敏感型锁存器。

4.2 时序电路建模基础

Verilog行为级描述用关键词initial或always，但initial是面向仿真，不能用于逻辑综合。always是无限循环语句，其用法为：

always@(事件控制表达式（或敏感事件表））

begin

块内局部变量的定义；

过程赋值语句；

end

4.2.1 阻塞型赋值语句与非阻塞型赋值语句

- 在always语句内部的过程赋值语句有两种类型：

  阻塞型赋值语句（Blocking Assignment Statement）

  非阻塞型赋值语句（Non-Blocking Assignment Statement）

- 使用的运算符如下：

  a=b
  ↓
  c=a
  c=b

  •赋值算符(=)： 阻塞型过程赋值算符

  前一条语句没有完成赋值过程之前，后面的语句不可能被执行。

  •赋值算符(<=)： 非阻塞型过程赋值算符

  一条非阻塞型赋值语句的执行，并不会影响块中其它语句的执行

过程赋值语句有阻塞型和非阻塞型:

阻塞型用“=”表示，多条语句顺序执行。

```
begin
    B=A;
    C=B+1;
end
```

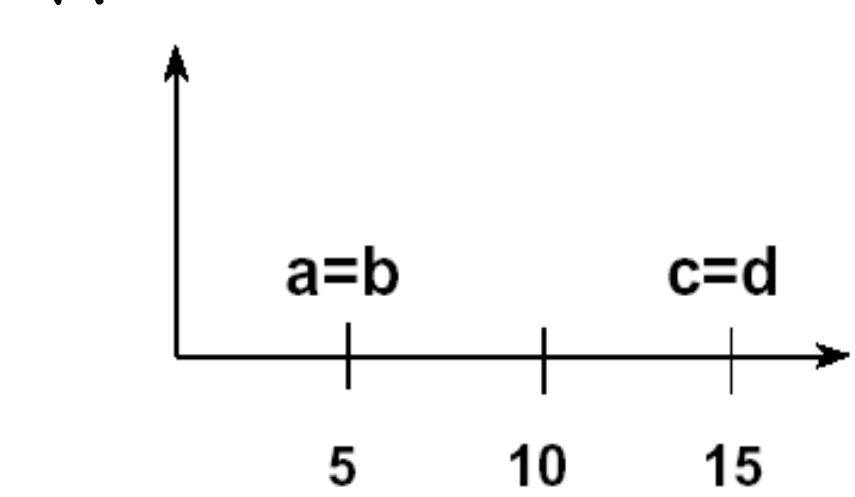
非阻塞型用“<=”表示，语句块内部的语句并行执行。

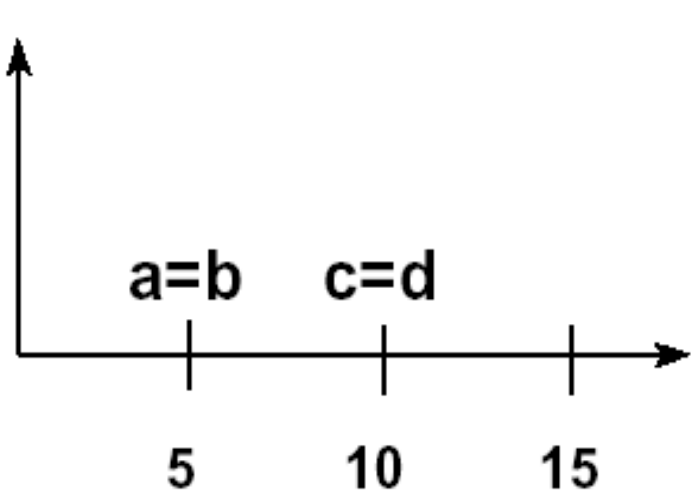
```
begin
    B<=A;
    C<=B+1;
end
```

阻塞型过程赋值与非阻塞型过程赋值

```
//Blocking (=)
initial
begin
    #5   a = b;
    #10  c = d;
end
```

```
//Nonblocking (<=)
initial
begin
    #5   a <= b;
    #10  c <= d;
end
```





注意：

- 在可综合的电路设计中，一个语句块的内部不允许同时出现阻塞型赋值语句和非阻塞型赋值语句。
- 在时序电路的设计中，建议采用非阻塞型赋值语句。

4.2.2 事件控制语句

- 用always语句描述硬件电路的逻辑功能时，在always语句中@符号之后紧跟着“事件控制表达式”。
- 逻辑电路中的敏感事件通常有两种类型：电平敏感事件和边沿触发事件。
- 在组合逻辑电路和锁存器中，输入信号电平的变化通常会导致输出信号变化，在Verilog HDL中，将这种输入信号的电平变化称为电平敏感事件。
- 在同步时序逻辑电路中，触发器状态的变化仅仅发生在时钟脉冲的上升沿或下降沿，Verilog HDL中用关键词posedge（上升沿）和negedge（下降沿）进行说明，这就是边沿触发事件。

敏感事件分为电平敏感事件和边沿触发事件:

电平敏感事件（如锁存器）：

always@(sel or a or b)

sel、a、b中任意一个电平发生变化，后面的过程赋值语句将执行一次。

边沿敏感事件（如触发器）：

always@(posedge CP or negedge CR)

CP的上升沿或CR的下降沿来到，后面的过程语句就会执行。

边沿触发事件

- 在always后面的边沿触发事件中，有一个事件必须是时钟事件，还可以有多个异步触发事件，多个触发事件之间用关键词 or 进行连接，例如，语句

```
always @(posedge CP or negedge Rd_ or negedge Sd_)
```

- 在Verilog 2001标准中，可以使用逗号来代替or。例如，

```
always @(posedge CP, negedge Rd_, negedge Sd_)
```

- posedge CP 是时钟事件， negedge Rd_和negedge Sd_是异步触发事件。如果没有时钟事件，只有异步事件，就会出现语法错误。

4.3 D触发器

4.3.1 D触发器的逻辑功能

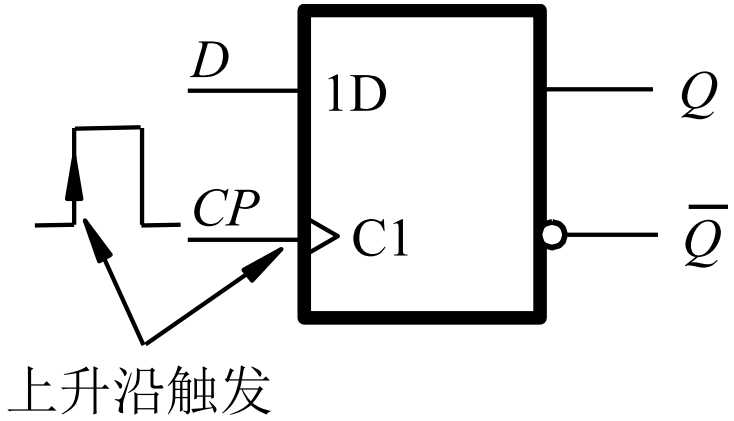
4.3.2 有清零输入和预置输入的D触发器

4.3.3 有使能端的D触发器

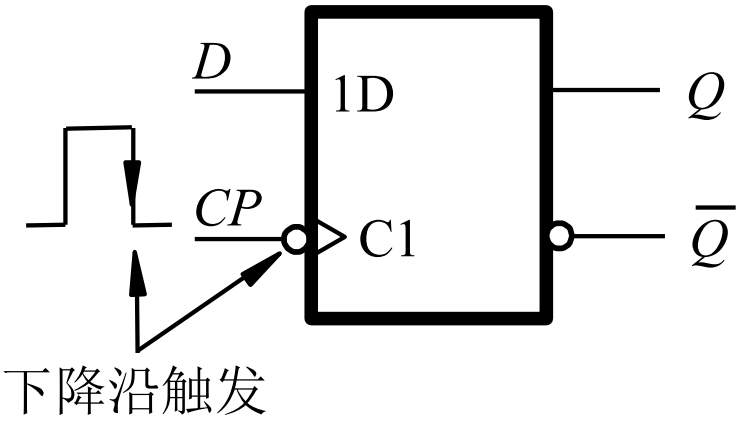
4.3.4 D触发器及其应用电路的Verilog HDL建模

4.3.1 D触发器的逻辑功能

1. D触发器的逻辑符号



（a）上升沿触发



（b）下降沿触发

- 把*CP*有效沿到来之前电路的状态称为**现态**，用Q^n表示。
- 把*CP*有效沿到来之后，电路所进入的新状态称为**次态**，用Q^{n+1}表示。

4.3.1 D触发器的逻辑功能

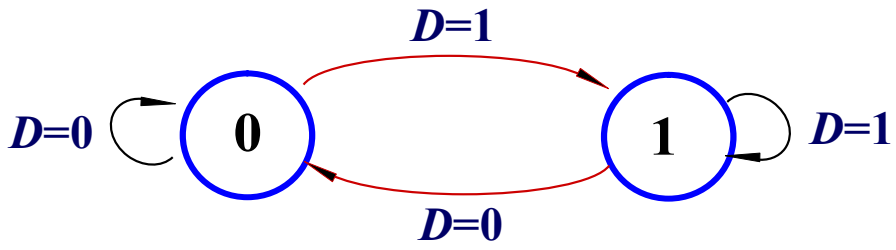
2. 特性表

D	Q^n	Q^{n+1}
0	0	0
0	1	0
1	0	1
1	1	1

3. 特性方程

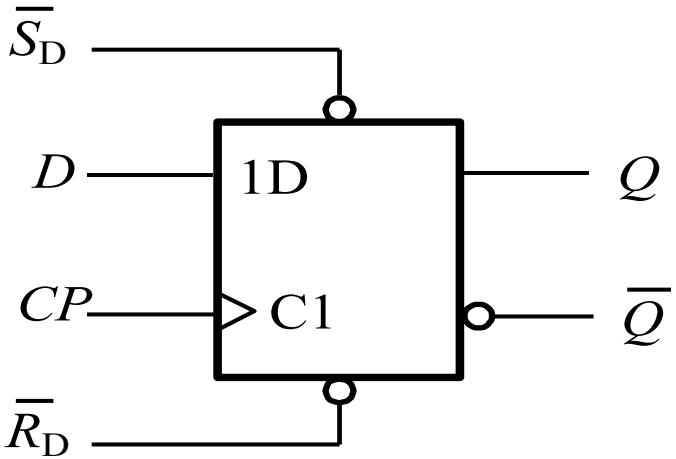
$$Q^{n+1} = D$$

4. 状态图

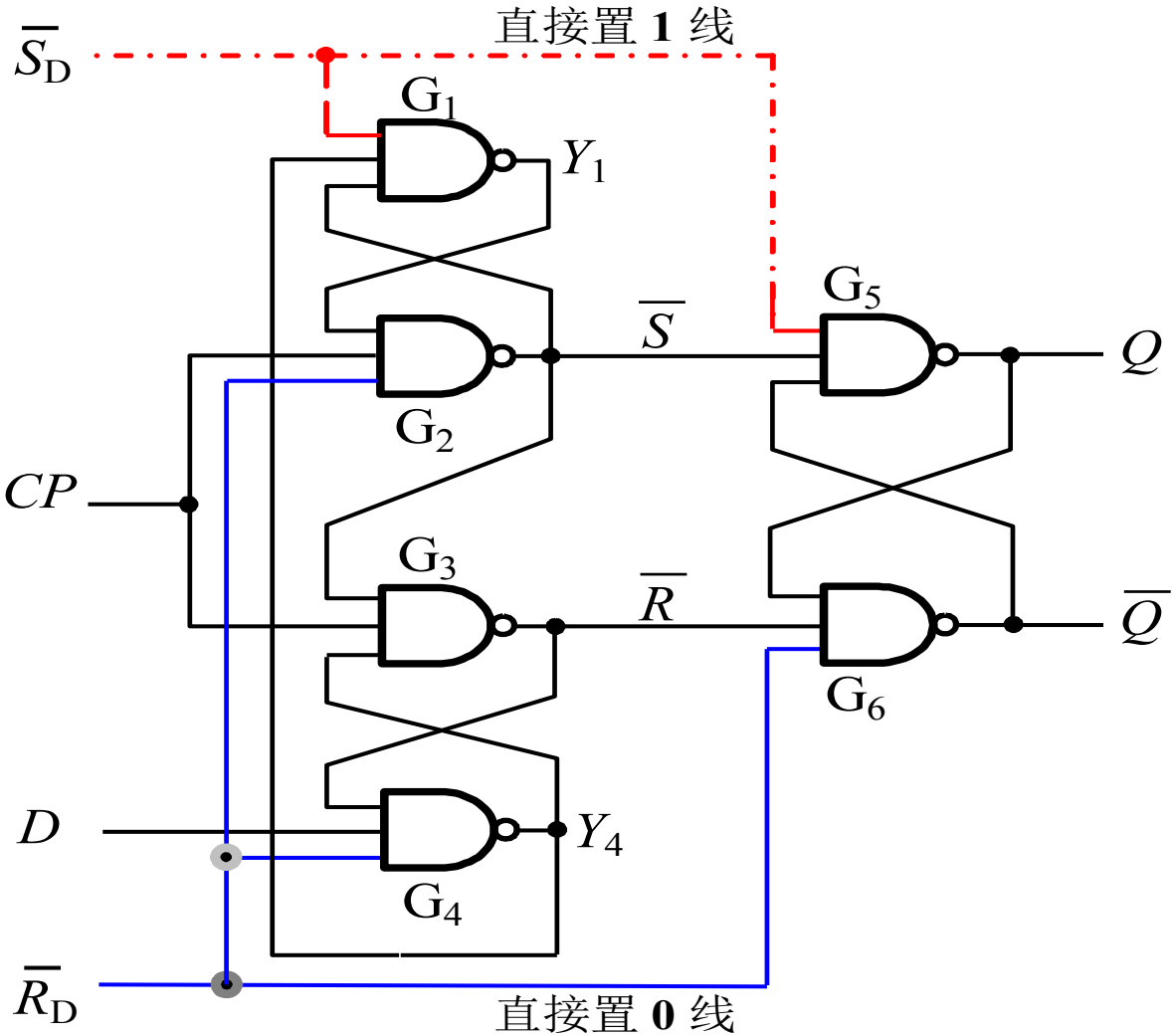


4.3.2 有清零输入和预置输入的D 触发器

由于直接置1和清零时跟*CP*信号无关，所以称置1、清零操作是异步置1和异步清零。



（a）



（b）

4.3.2 有清零输入和预置输入的D 触发器

直接置1和直接清零的过程如下：

(1) 当 $\overline{S}_D=0$，$\overline{R}_D=1$ 时，使得 $Y_1=1$，

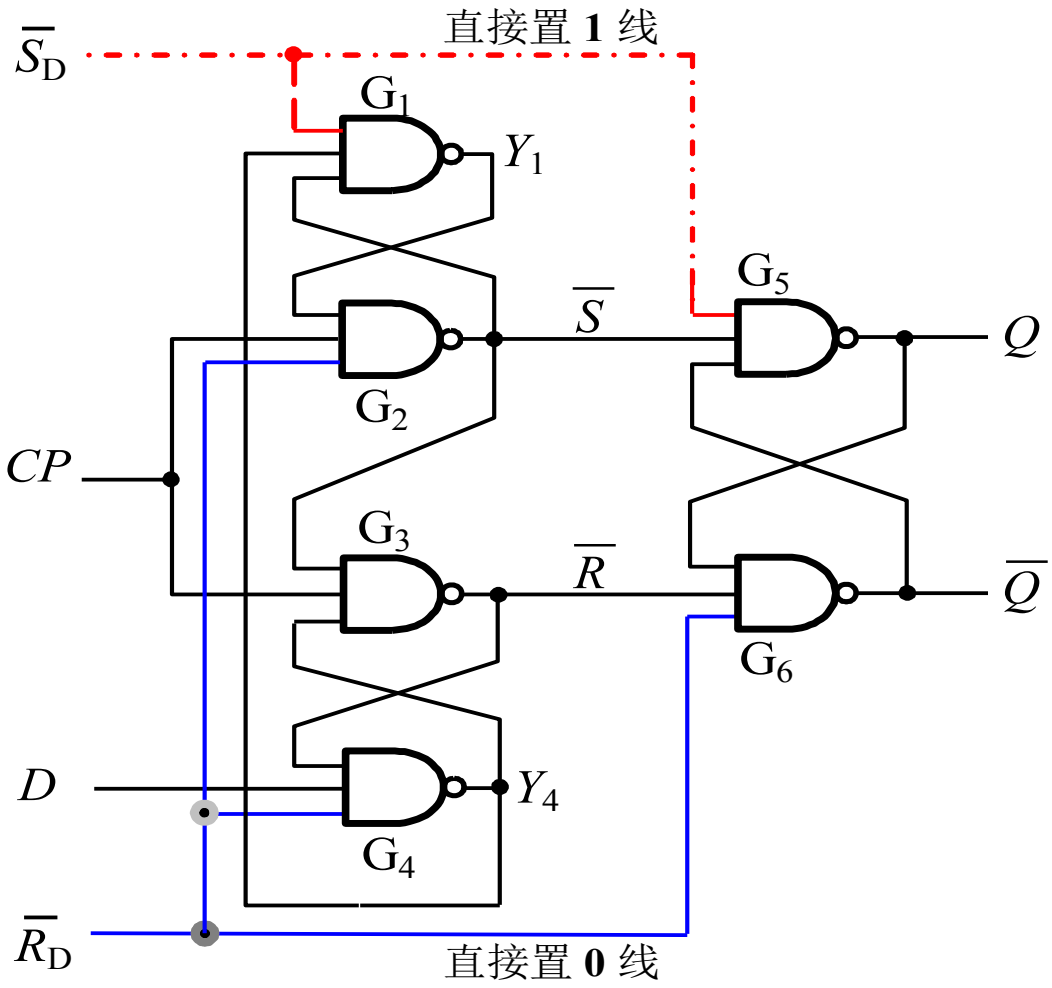
$$\overline{S}=\overline{Y_1\cdot CP\cdot \overline{R}_D}=\overline{CP}，\quad \overline{R}=\overline{\overline{S}\cdot CP\cdot Y_4}=1，$$

于是 $Q=1$，$\overline{Q}=0$，即将输出 Q 直接置1。

(2) 当 $\overline{S}_D=1$，$\overline{R}_D=0$ 时，

使得 $\overline{S}=1$，于是 $Q=0$，$\overline{Q}=1$，

即将输出 Q 直接清零。



有同步清零端的 D 触发器

- 所谓同步清零是指在清零输入信号有效，并且*CP*的有效边沿到来时，才能将触发器清零。

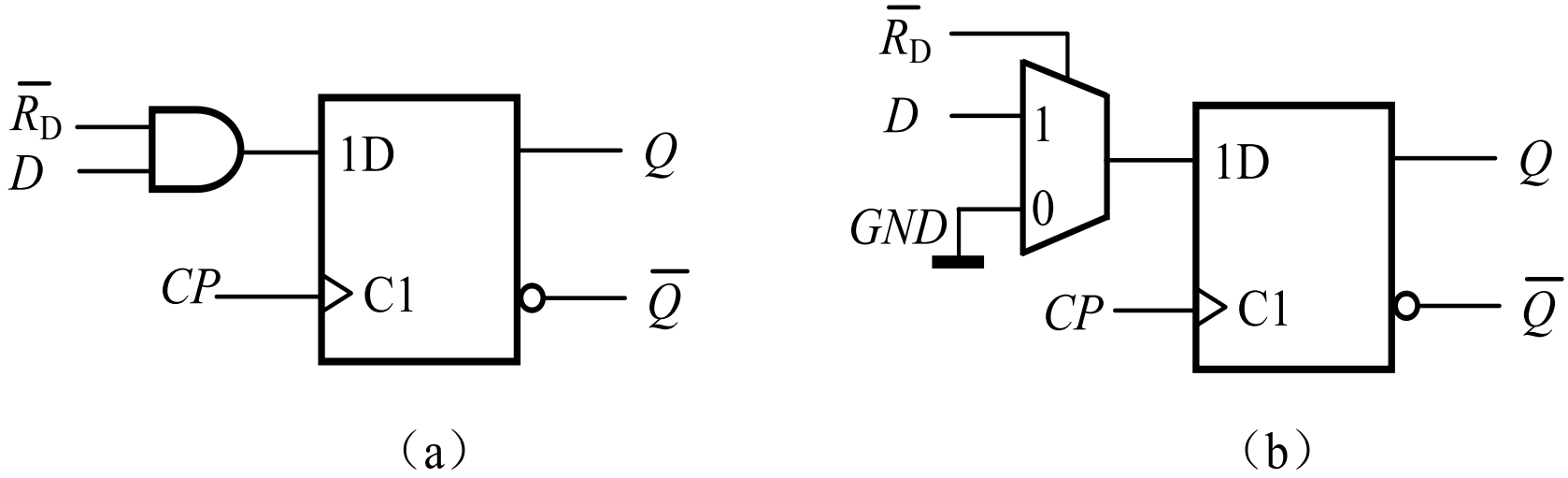


图4.3.4 有同步清零输入端的D触发器

(a) 实现同步清零的方案之一　　**(b)** 实现同步清零的方案之二

4.3.3 有使能端的D 触发器

- **功能：**
  - *En*=**0**，*Q* **保持不变。**
  - *En*=**1**，**在***CP***作用下，**

$$Q = D$$。

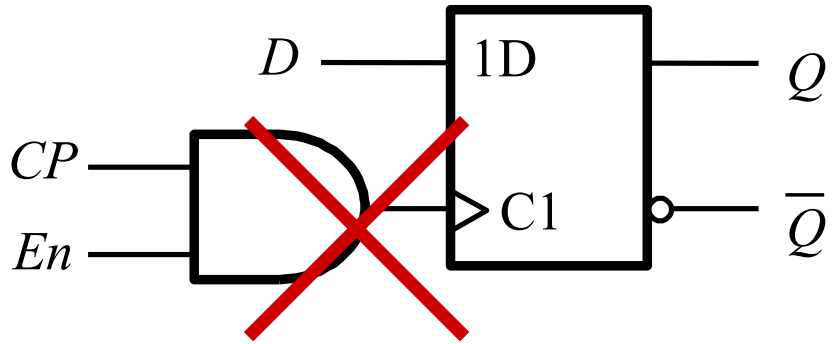
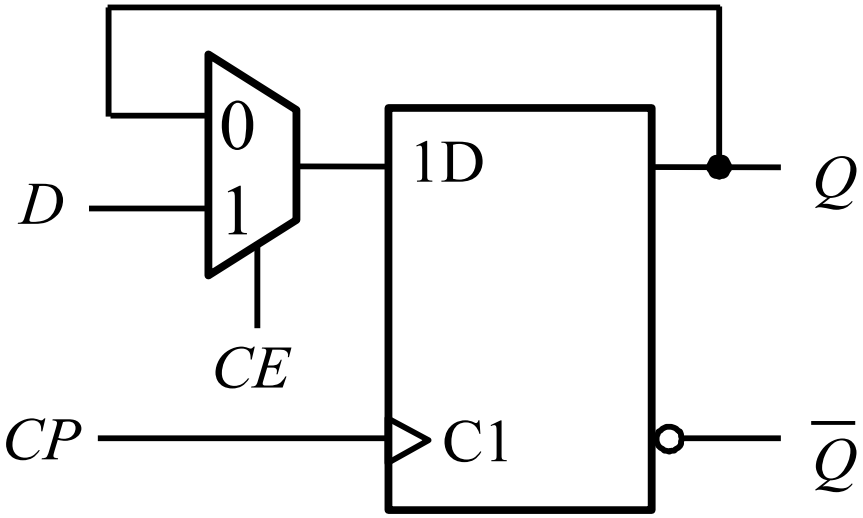
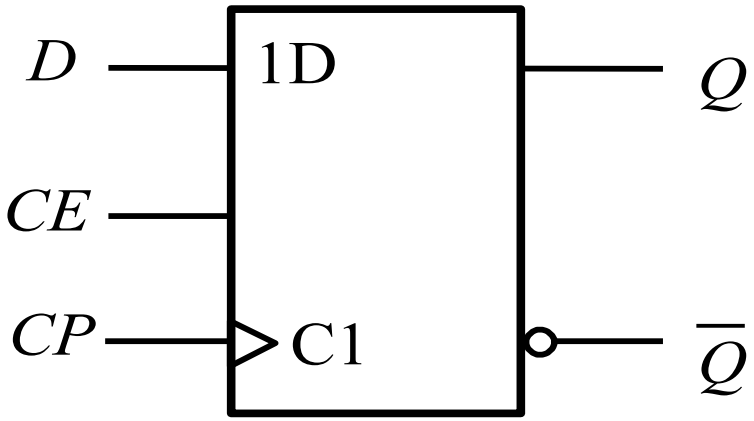


图 4.3.5 用门控制时钟的电路



$$Q^{n+1} = \overline{CE} \cdot Q^n + CE \cdot D$$



逻辑符号

4.3.4 D触发器及其应用电路的Verilog HDL建模

例1.试对图4.3.3所示的带有异步清零和异步置位的边沿D触发器进行建模。

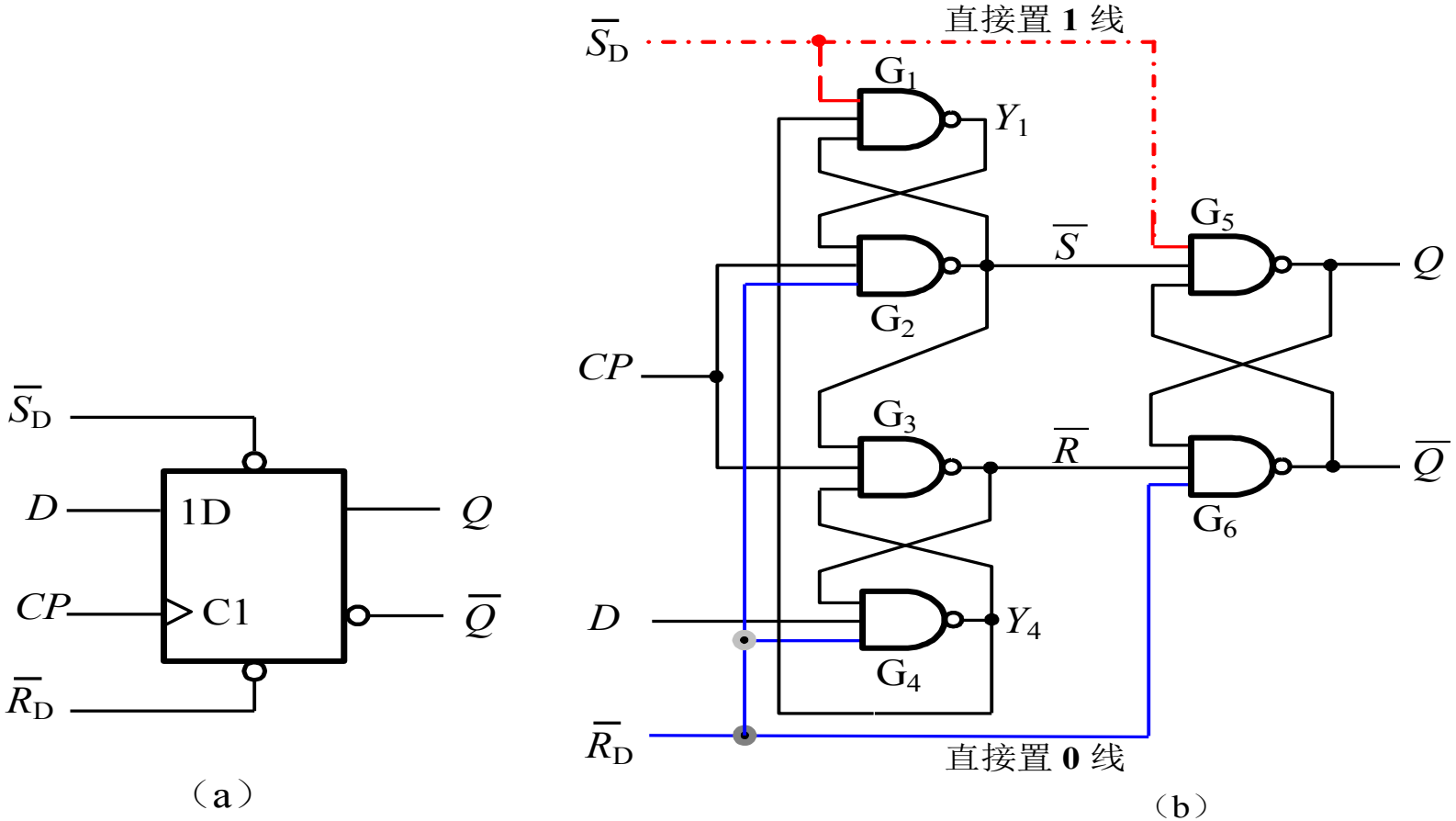


图4.3.3　有异步输入端的D触发器

```
//版本1:
module Set_Rst_DFF (Q,Q_,D,CP,Rd_,Sd_);
   output Q,Q_;
   input D,CP,Rd_,Sd_;
   wire Y1,Y2,Y3,Y4,Y5,Y6;
    assign  #5 Y1 = ~(Sd_ & Y2 & Y4);
    assign  #5 Y2 = ~(Rd_ & CP & Y1);
    assign  #5 Y3 = ~(CP  & Y2 & Y4);
    assign  #5 Y4 = ~(Rd_ & Y3 & D);
    assign  #5 Y5 = ~(Sd_ & Y2 & Y6);
    assign  #5 Y6 = ~(Rd_ & Y3 & Y5);
    assign Q = Y5;
    assign Q_= Y6;
endmodule
```

- **版本1根据该图使用连续赋值语句来建模，在assign语句中的#5表示给每个与非门加5个单位时间的传输延迟。**

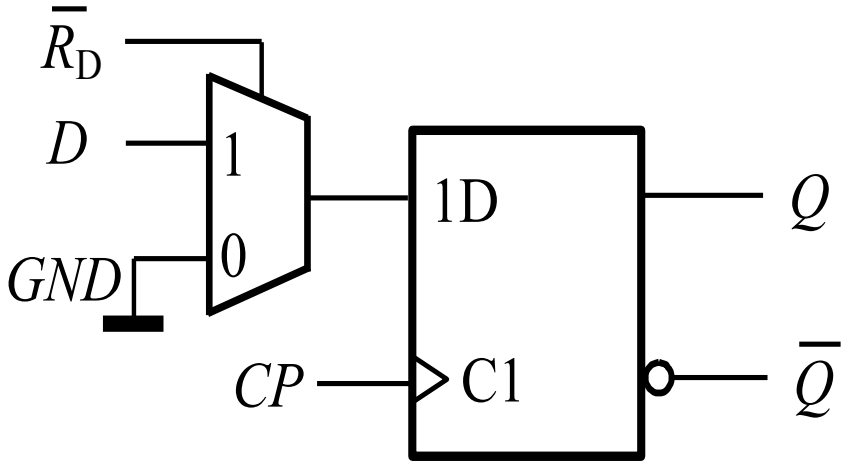
```
//版本2
module Set_Rst_DFF_bh (Q,Q_,D,CP,Rd_,Sd_);
    output reg Q;
    output Q_;
    input D,CP,Rd_,Sd_;

  assign Q_ = ~Q;

  always @(posedge CP or negedge Sd_ or negedge Rd_)
    if (~Sd_)        //等同于: if (Sd_ == 0)
            Q <= 1'b1;
    else if (~Rd_)
            Q <= 1'b0;
    else
             Q <= D;
  endmodule
```

版本2的特点：

- 采用功能描述风格，使用always和if-else对输出变量赋值。
- negedge Sd_是一个异步事件，它与if（~Sd_）必须匹配，negedge Rd_是另一个异步事件，它与if（~Rd_）必须匹配，这是语法规定。
  - 当Sd_为0时，将输出Q置1；
  - 当Sd_=1且Rd_=0时，将输出Q置0；
  - 当Sd_和Rd_均不为0，且时钟CP的上升沿到来时，将输入D传给输出Q。
- 注意，如果置1事件、置0事件和时钟事件同时发生，则置1事件的优先级别最高、置0事件的次之，时钟事件的优先级最低。

例2 具有同步清零功能的上升沿D触发器。

```
module Sync_rst_DFF (Q,D,CP,Rd_);
    output  reg Q;
    input D, CP, Rd_;

  always @(posedge CP)
      if ( !Rd_)
            Q <=0;
      else
            Q <= D;
endmodule
```



例4　试用功能描述风格对图4.3.7所示电路进行建模，并给出仿真结果。

解：（1）设计块：使用always和if-else语句对输出变量赋值，其代码如下。

```
`timescale 1 ns/ 1 ns
module _2Divider (Q,CP,Rd_);
    output reg Q;
    input CP,Rd_;
    wire D;
    assign D = ~Q;
always @(posedge CP or
          negedge Rd_)
     if(~Rd_)   Q <= 1'b0;
     else       Q <= D;
endmodule
```

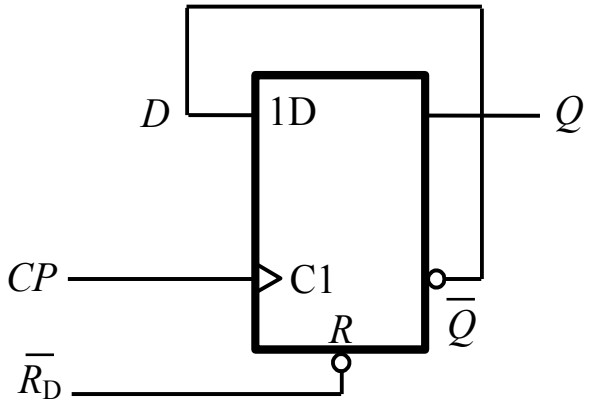
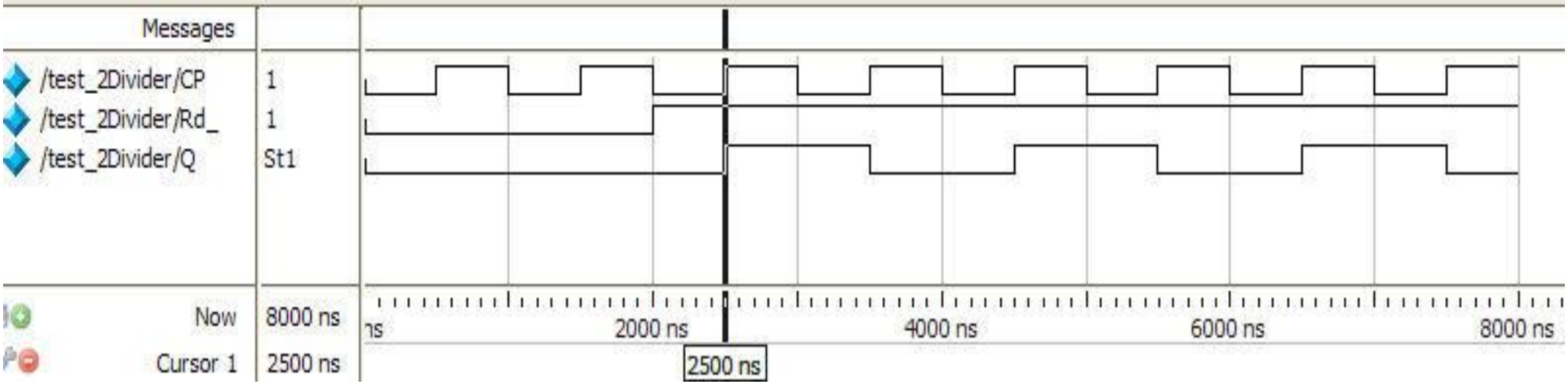


图 4.3.7　2 分频电路

(2) 激励块：给输入变量赋值。

```
`timescale 1 ns/ 1 ns
module test_2Divider();
  reg CP, Rd_;    wire Q;
  //调用设计块
  _2Divider U1 ( .CP(CP), .Q(Q),.Rd_(Rd_) );
  initial begin     //产生复位信号Rd_
          Rd_ = 1'b0;
          Rd_ = #2000 1'b1;
   #8000 $stop;
  end
  always begin        //产生时钟信号CP
          CP = 1'b0;
          CP = #500 1'b1;
          #500;
  end
endmodule
```

（3）仿真波形（用ModelSim）



由图可知，时钟CP的周期为1000ns，在2000ns之前，清零信号Rd_有效，输出Q被清零。在此之后，Rd_=1，在2500ns时，CP上升沿到来，Q=1；到下一个CP上升沿（3500ns）时，Q=0，再到下一个CP上升沿（4500ns）时，Q=1，……，如此重复，直到8000ns时，系统任务$stop被执行，仿真停止。

总之，在不考虑清零信号Rd_的作用时，每当CP上升沿到来时，触发器状态Q翻转一次。输出信号Q的频率正好是CP频率的二分之一，故称该电路为2分频电路。所谓分频电路，是指可将输入的高频信号变为低频信号输出的电路。

例5 试对图4.3.9所示电路进行建模，并给出仿真结果。

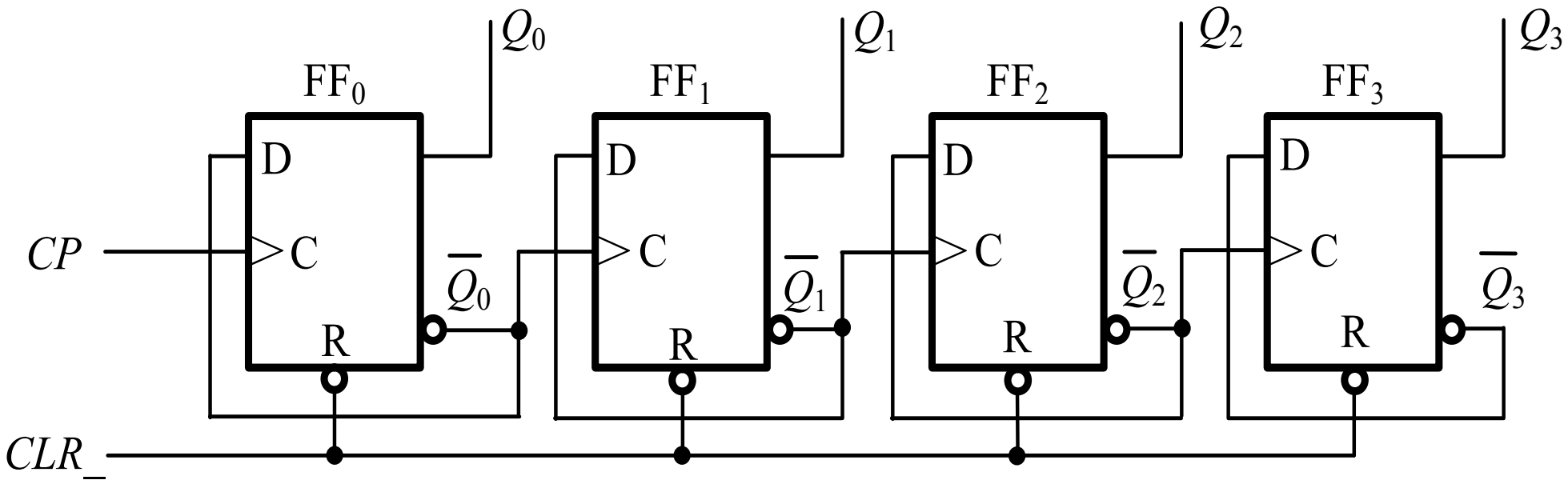


图4.3.9 4位步二进制计数器逻辑图

解：（1）采用结构描述风格的代码如下。编写了两个模块，这两个模块可以放在一个文件中，文件名为**Ripplecounter.v**。

第一个主模块Ripplecounter作为设计的顶层，它实例引用分频器子模块_2Divider1共4次，第二个分频器子模块_2Divider1作为设计的底层。

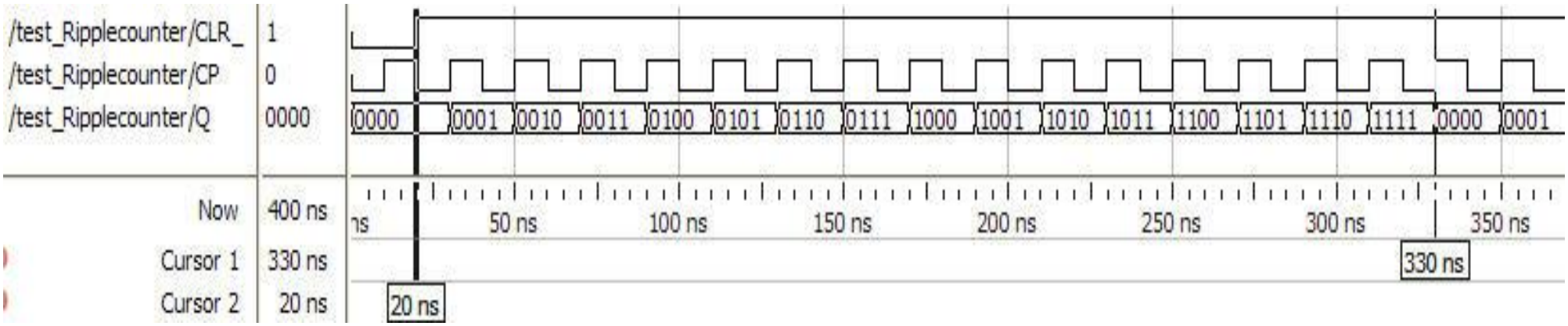
```
/*==== 设计块：Ripplecounter.v ====*/
module Ripplecounter (Q,CP,CLR_);
    output [3:0]Q;
    input CP, CLR_;

        //实例引用分频器模块
    _2Divider1 FF0 (Q[0],CP,CLR_);
        //注意，引用时端口的排列顺序
    _2Divider1 FF1 (Q[1],~Q[0],CLR_);
    _2Divider1 FF2 (Q[2],~Q[1],CLR_);
    _2Divider1 FF3 (Q[3],~Q[2],CLR_);
endmodule
```

```
//分频器子模块
module _2Divider1 (Q,CP,Rd_);
    output reg Q;
    input CP,Rd_;
 always @(posedge CP or negedge Rd_)
    if(!Rd_)
        Q <= 1'b0;
    else
        Q <= ~Q;
endmodule
```

(2) 激励块：给输入变量（CLR_和CP）赋值。

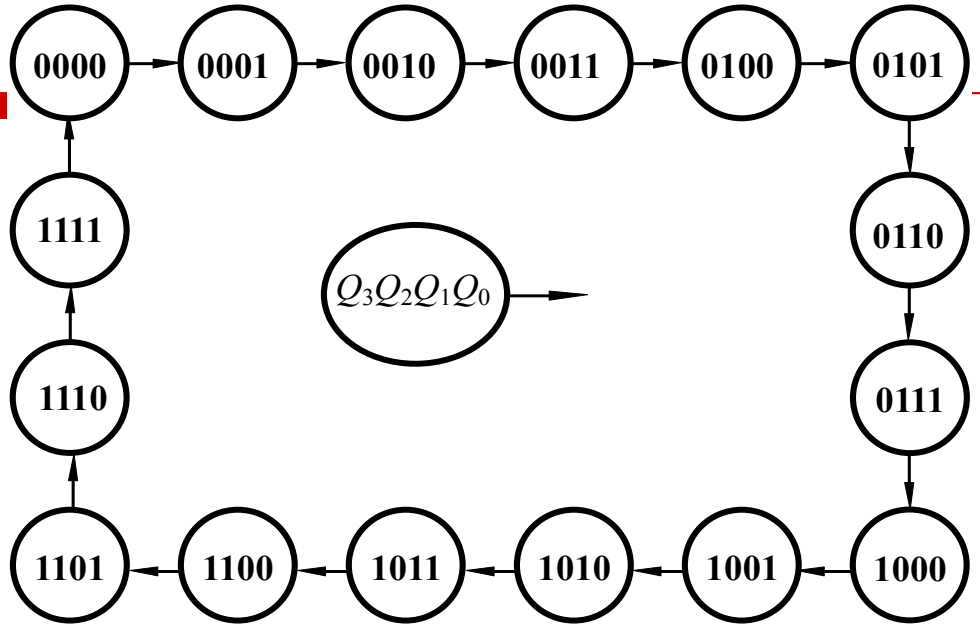
```
/*==== 激励块：test_Ripplecounter.v ====*/
module test_Ripplecounter();
  reg CLR_, CP;
  wire [3:0]Q;

  Ripplecounter i1 (.CLR_(CLR_),.CP(CP),.Q(Q));

  initial begin      // CLR_
     CLR_ = 1'b0;
     CLR_ = #20 1'b1;
     #400 $stop;
  end
  always begin    // CP
    CP = 1'b0;
    CP = #10 1'b1;
    #10;
  end
endmodule
```

（3）仿真波形如图4.3.10所示。



由图可知，

- 时钟CP的周期为20ns。
- 开始时，清零信号CLR_有效（0~20ns），输出Q被清零。
- 20ns之后，CLR_一直为高电平，
- 在30ns时，CP上升沿到来， Q=0001；到下一个CP上升沿（50ns）时，Q=0010，
- 再到下一个CP上升沿（70ns）时，Q=0011，……，如此重复，到310ns时，Q=1111，
- 到330ns时，Q=0000，……，直到系统任务$stop被执行，仿真停止。



电路首先在CLR_的作用下，输出被清零。此后当CLR_=1时，每当CP上升沿到来时，电路状态Q就在原来二进制值的基础上增加1，即符合二进制递增计数的规律，直到计数值为1111时，再来一个CP上升沿，计数值回到0000，重新开始计数。故称该电路为4位二进制递增计数器。

可见，计数器实际上是对时钟脉冲进行计数，每到来一个时钟脉冲触发沿，计数器改变一次状态。

4.4 寄存器和移位寄存器

4.4.1 寄存器及Verilog HDL建模

4.4.2 移位寄存器及Verilog HDL建模

4.4.3 移位寄存器的应用电路

4.4.1 寄存器及Verilog HDL建模

- 图中，PD_3~PD_0是4位数据输入端，
  - 当*Load* = 1时，在*CP*脉冲上升沿到来时，$Q_3 = PD_3$，$Q_2 = PD_2$，$Q_1 = PD_1$，$Q_0 = PD_0$，即输入数据PD_3~PD_0同时存入相应的触发器；
  - 当*Load* = 0时，即使*CP*上升沿到来，输出端的状态将保持不变。可见，电路具有存储输入的二进制数据的功能。

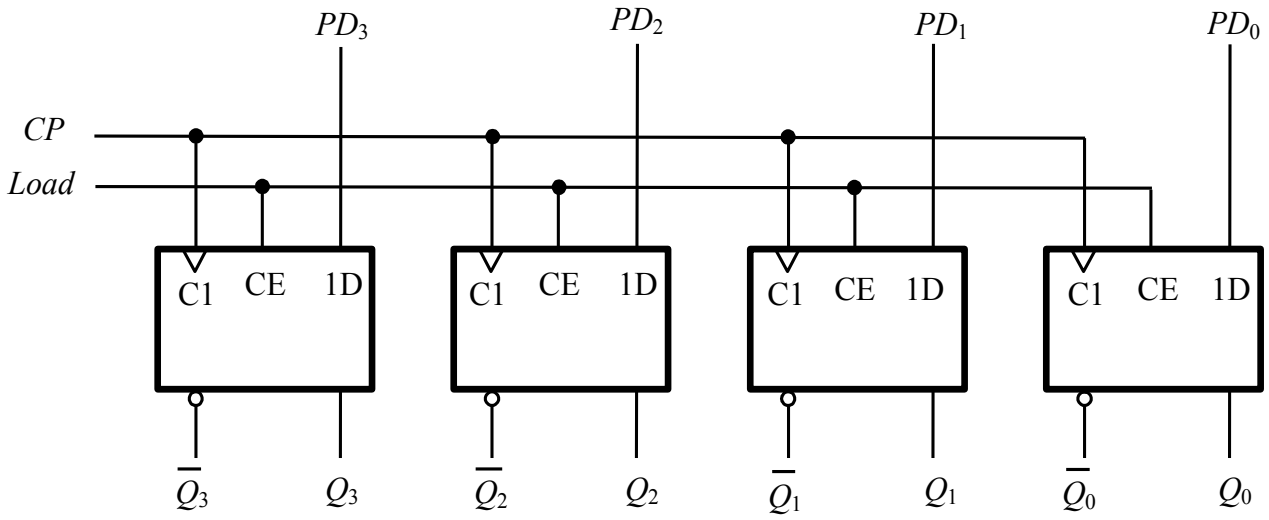


图 4.4.1　4 位寄存器

例4.4.1 试对图4.4.1所示的寄存器进行建模。

```
module Reg4bit (Q,PD,CP,CLR_,Load);
    output reg [3:0]Q;
    input wire [3:0]PD;
    input CP,CLR_,Load;

    always @(posedge CP or posedge CLR_)
      if (!CLR_)
                Q <= 4'b0;
      else if (Load)
                Q <= PD;
endmodule
```

例4.4.2 试对图4.4.2所示任意位数的寄存器进行建模。

```
module Register  //Verilog 2001, 2005 syntax
  #(parameter N = 8)       //定义参数N = 8
   (output reg [N-1:0]Q, //数据输出端口及变量的数据类型声明
    input wire [N-1:0]PD,//并行数据输入
    input CP,CLR_,Load   //输入端口声明
);
 always @(posedge CP or posedge CLR_)
   if (~CLR_)
            Q <= 0;
   else if (Load)
            Q <= PD;
endmodule
```

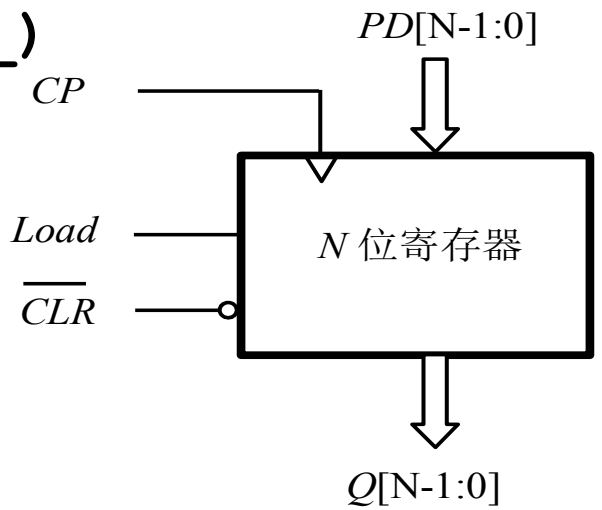


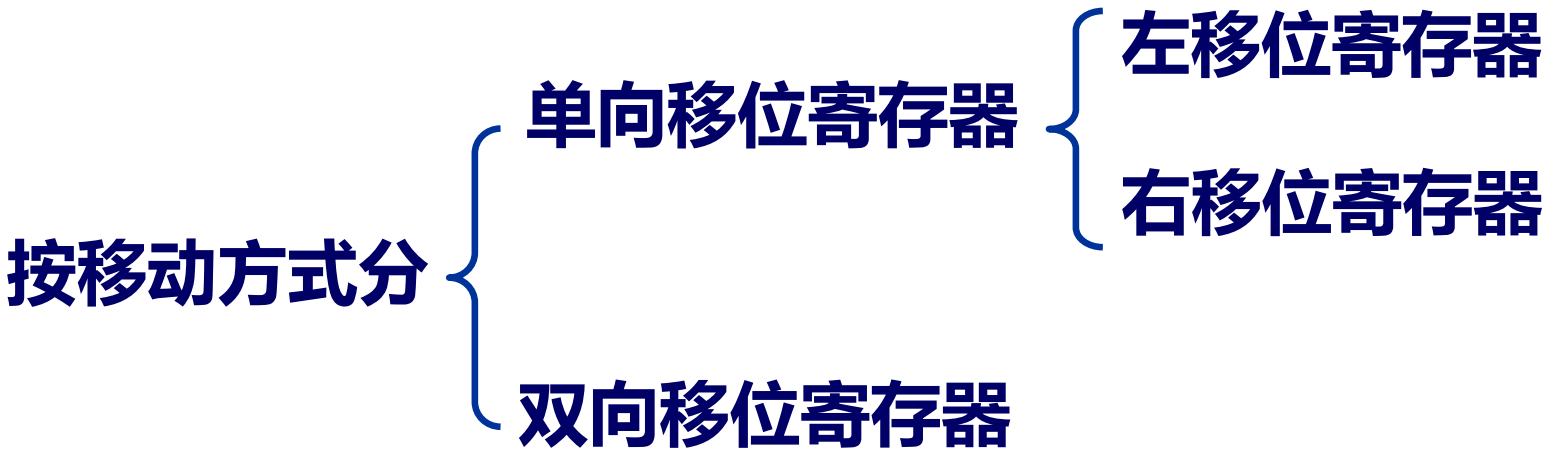
图 4.4.2　N 位寄存器框图

4.4.2 移位寄存器及Verilog HDL建模

(1) 移位寄存器

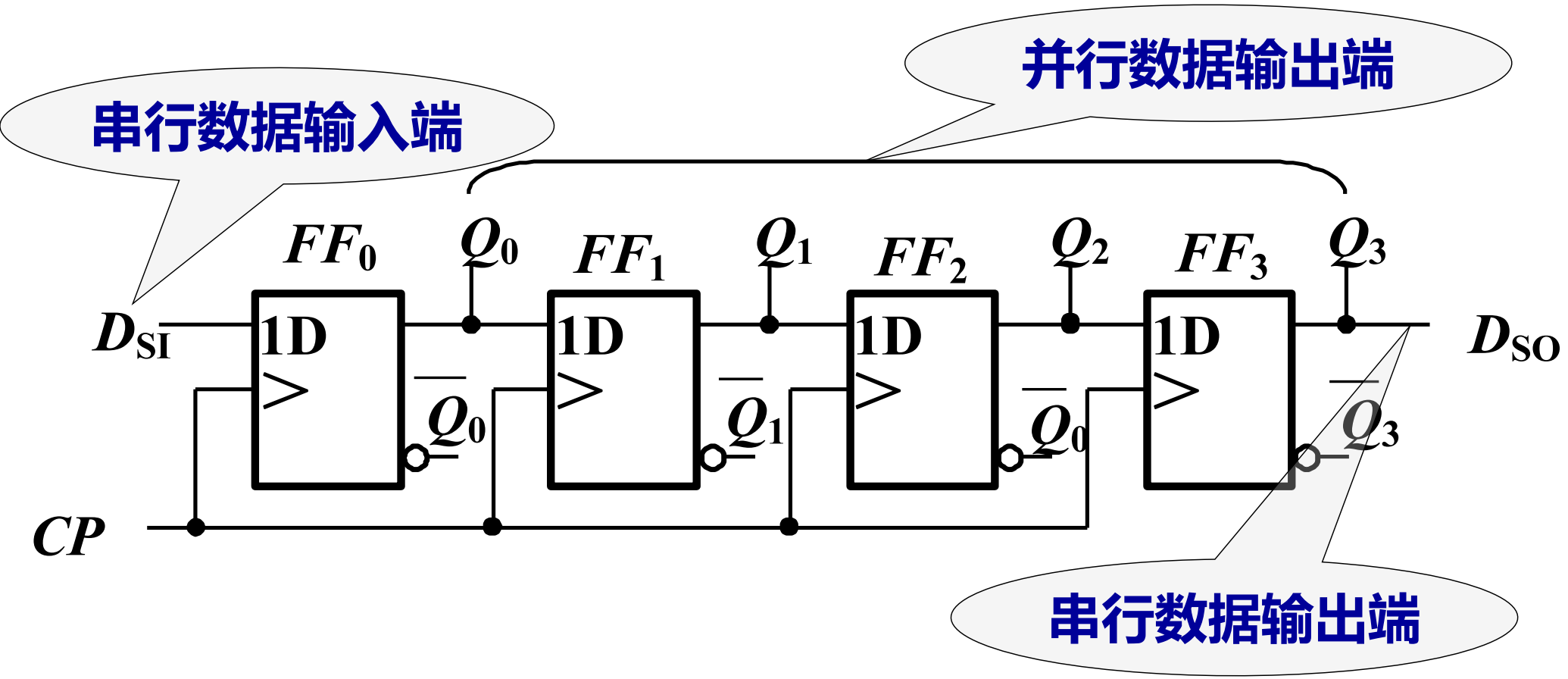
- 将若干个D触发器串接级联在一起构成的具有移位功能的寄存器，叫做移位寄存器。

•移位寄存器的逻辑功能分类



(1) 4位单向右移移位寄存器

（a）电路



(b) 工作原理

写出激励方程：

$D_0=D_{SI}$　$D_1=Q_0^n$　$D_2=Q^n{}_1$　$D_3=Q^n{}_2$

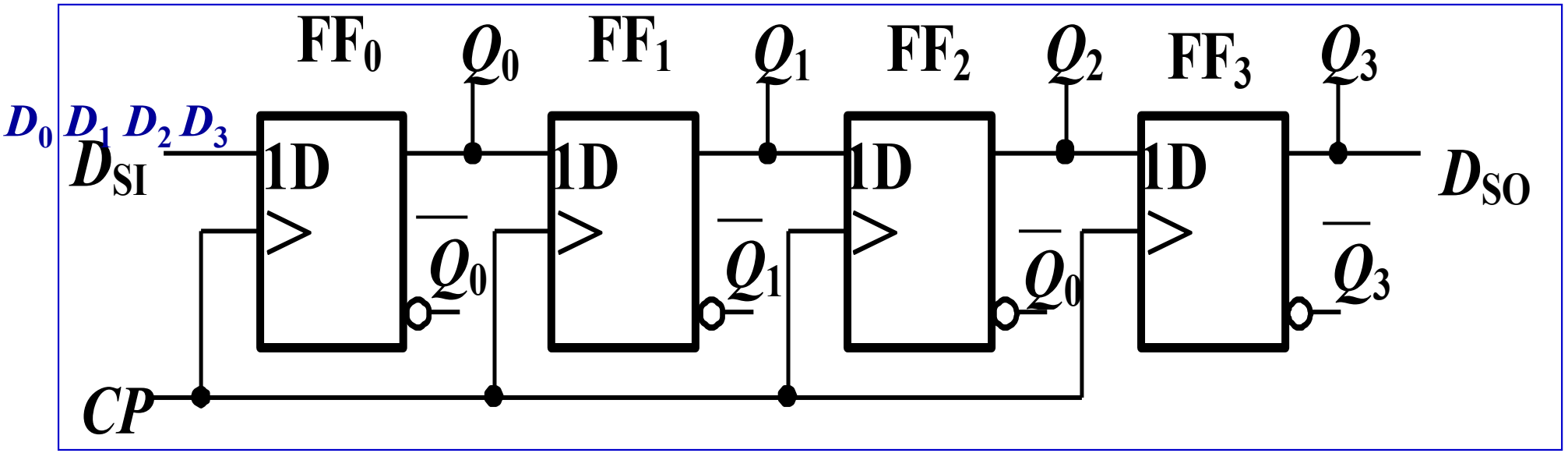
写出状态方程：

$Q_0^{n+1}=D_{SI}$

$Q_1^{n+1}=D_1=Q_0^n$

$Q_2^{n+1}=D_2=Q^n{}_1$

$Q_3^{n+1}=D_3=Q^n{}_2$



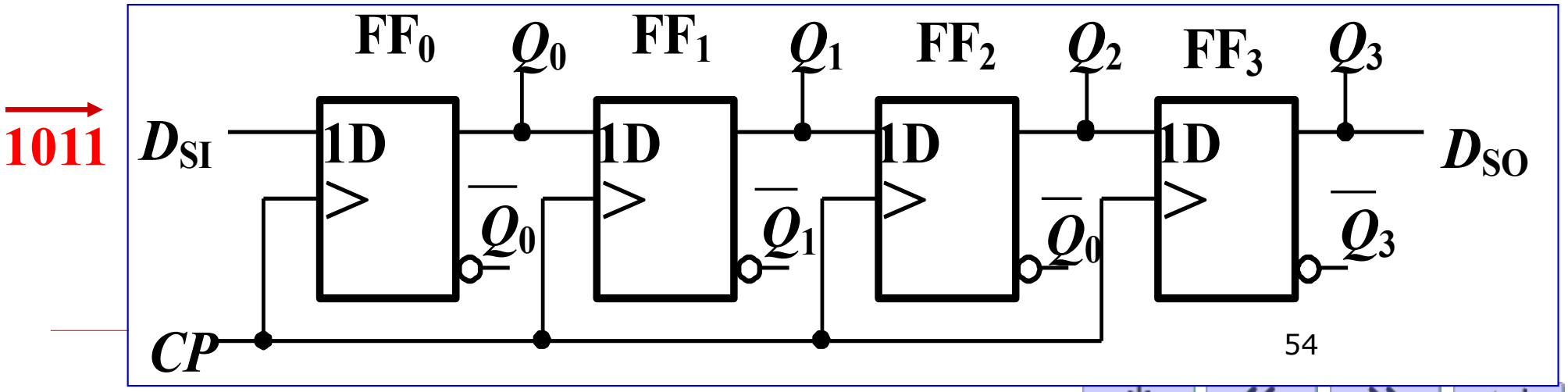
$Q_0^{n+1}=D_{SI}$

$Q_1^{n+1}=Q_0^n$

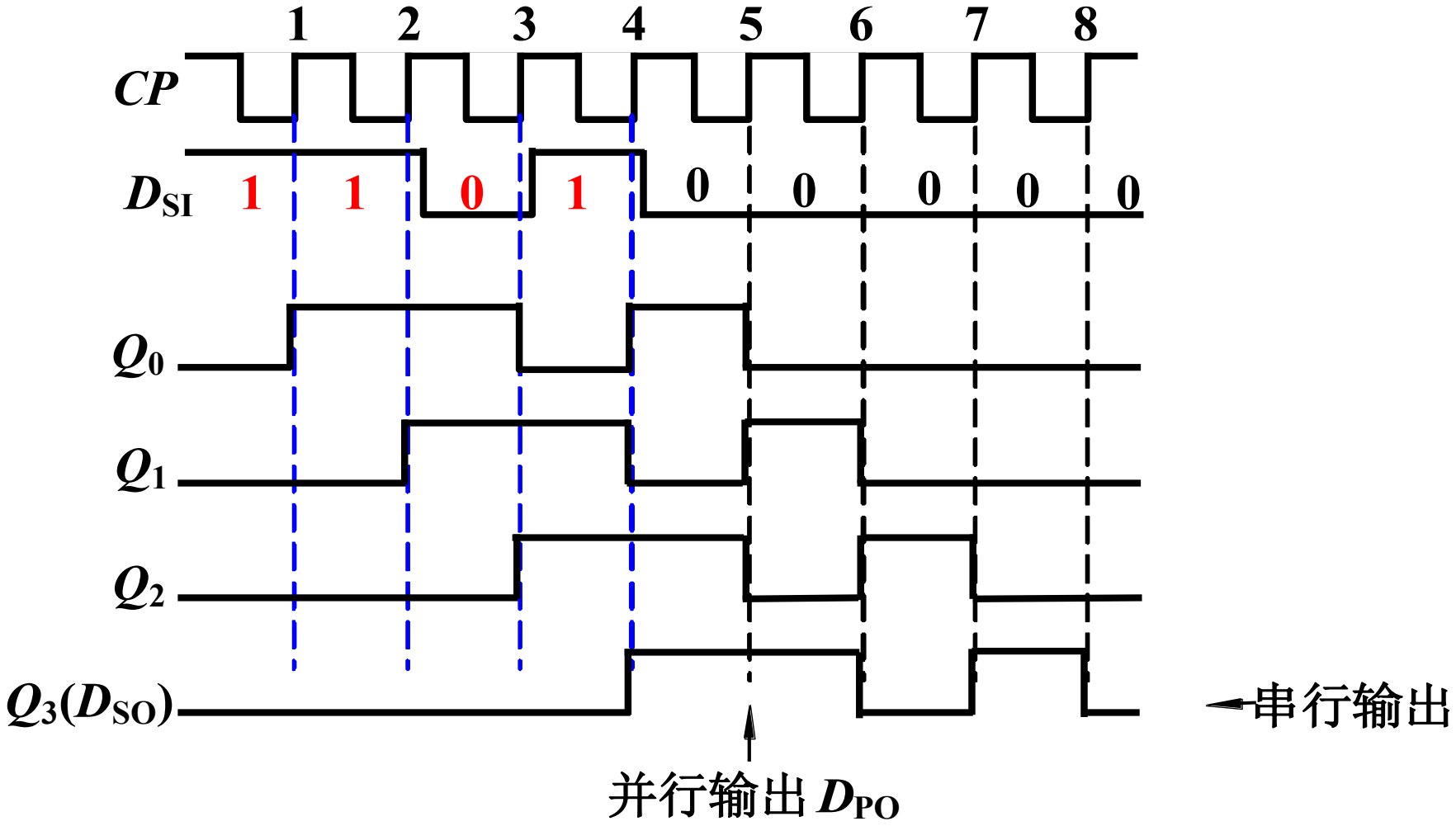
$Q_2^{n+1}=Q^n{}_1$

$Q_3^{n+1}=Q^n{}_2$

	FF_0	FF_1	FF_2	FF_3
	0	0	0	0
1CP 后 1→	1	0	0	0
2CP 后 1→	1	1	0	0
3CP 后 0→	0	1	1	0
4CP 后 1→	1	0	1	1

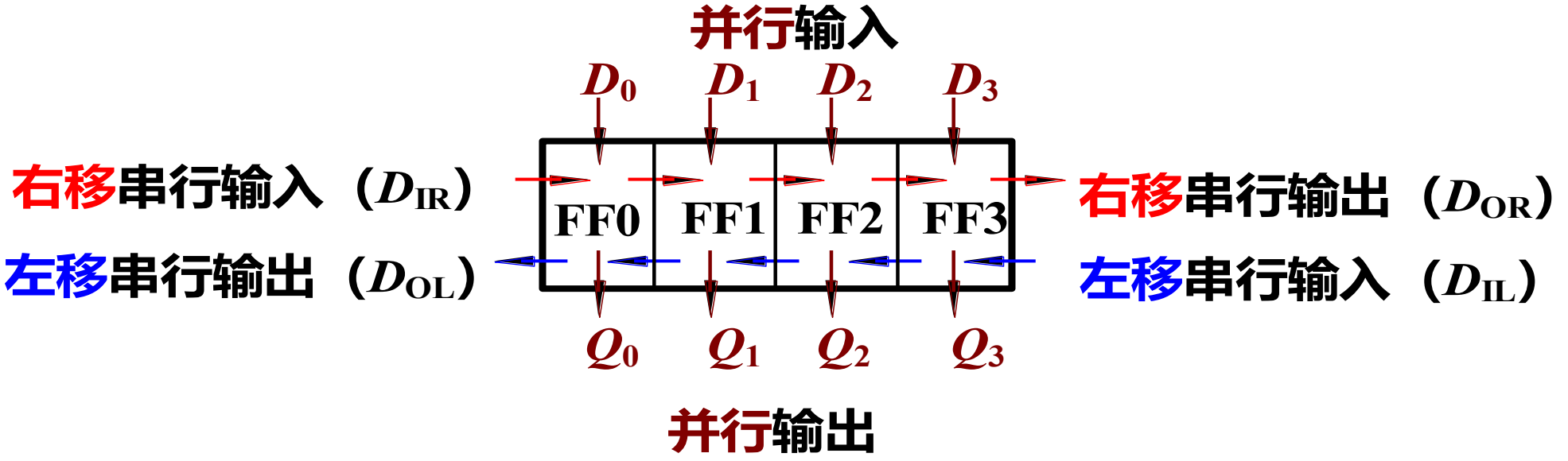


D_{SI} =11010000,从高位开始输入



经过7个*CP*脉冲作用后，从D_{SI}端串行输入的数码就可以从D_O端串行输出。　　串入→串出

（2）多功能移位寄存器工作模式简图



右移：$D_{IR}\rightarrow Q_0\rightarrow Q_1\rightarrow Q_2\rightarrow Q_3\rightarrow D_{OR}$ (左入右出)

左移：$D_{OL}\leftarrow Q_0\leftarrow Q_1\leftarrow Q_2\leftarrow Q_3\leftarrow D_{IL}$ (右入左出)

右移输入并行输出：$D_{IR}\rightarrow Q_0\rightarrow Q_1\rightarrow Q_2\rightarrow Q_3$ (4个CP)

左移输入并行输出：$Q_0\leftarrow Q_1\leftarrow Q_2\leftarrow Q_3\leftarrow D_{IL}$ (4个CP)

并行**输入并行输出**、并行输入串行输出。

- 与普通移位寄存器的连接不同，输入端D连接两个不同的数据源，一个数据源为前级的输出，用于移位寄存器的操作；另一个数据来自于外部输入，作为并行操作的一部分。
- 控制信号Mode用来选择操作的模式，
  - 当Mode＝0时，电路实现移位操作；
  - 当Mode＝1时，则并行数据In3~In0便送到各自的输出端寄存。这两种操作都发生在时钟信号的上升沿时刻。

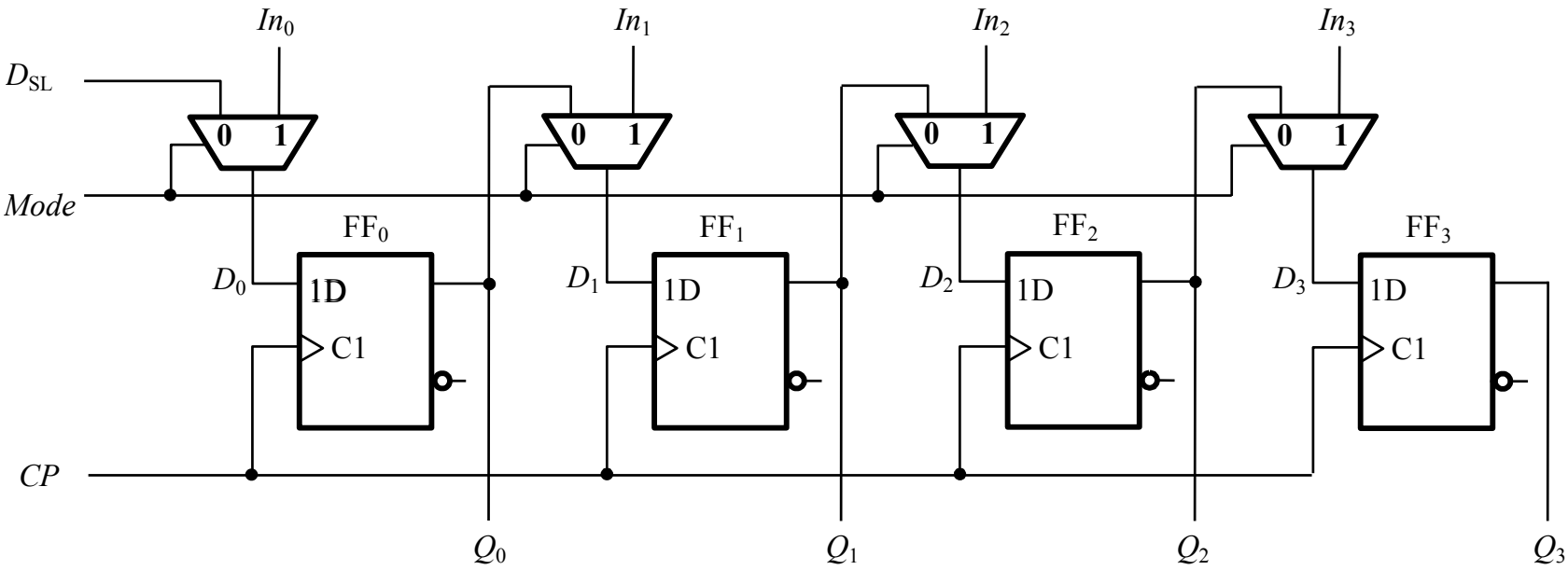


图4.4.4 并行存取的移位寄存器

- 将图4.4.3中移位寄存器的D_{SO}（Q_3）与D_{IN}相连，则构成环形计数器，如图4.4.5所示。

- 若事先通过PE端施加低电平脉冲，将初始数据$Q_0Q_2Q_1Q_3$=1000置入触发器中，

- 在*CP*脉冲用下，$Q_0Q_2Q_1Q_3$将依次为1000→0100→0010→0001→1000→……，即每个触发器经过4个时钟周期输出一个高电平脉冲，并且该高电平脉冲沿环形路径在触发器中传递。可见，4个触发器只有4个计数状态。

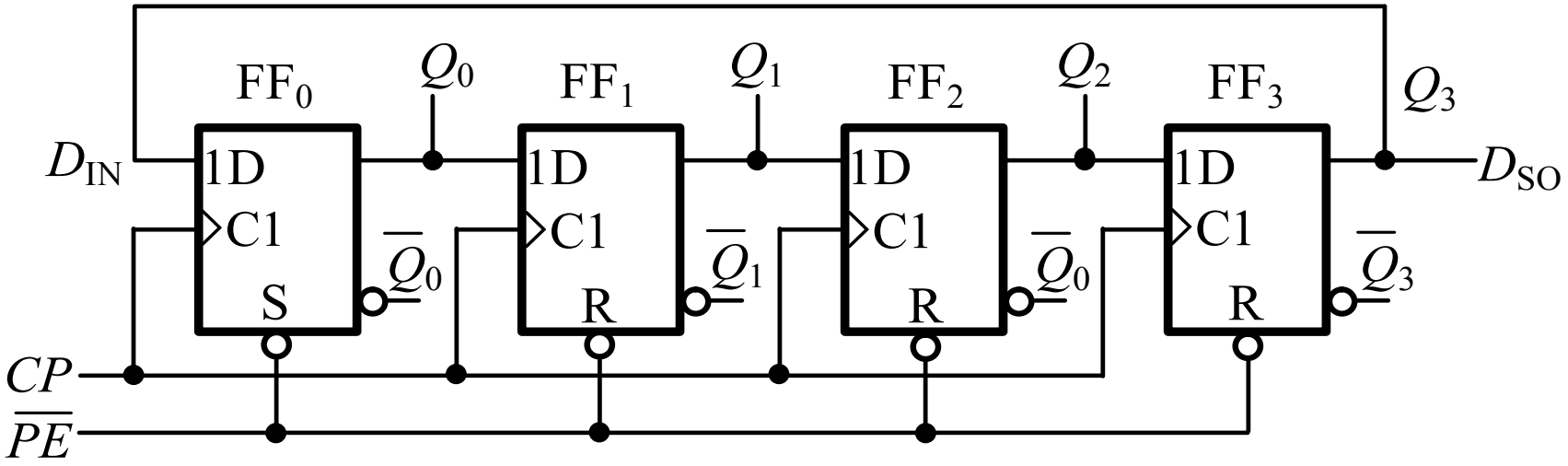


图4.4.5 环形计数器

- 如果将图4.4.3电路中的 与DIN相连，则构成扭环形计数器，亦称为约翰逊计数器（Johnson counter），电路的状态将增加一倍。

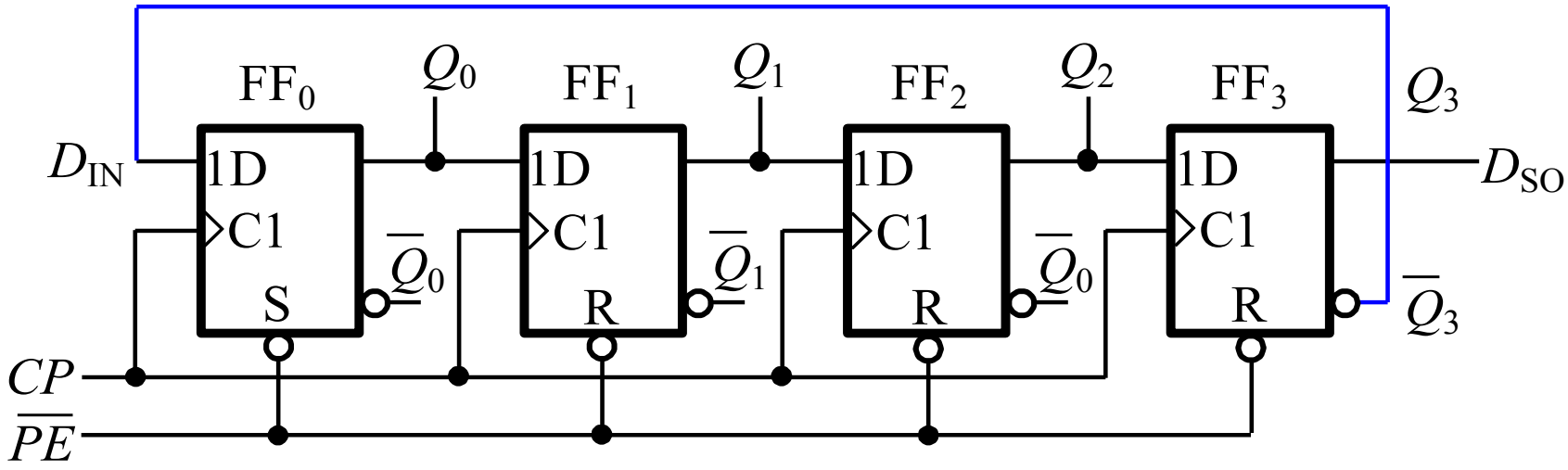
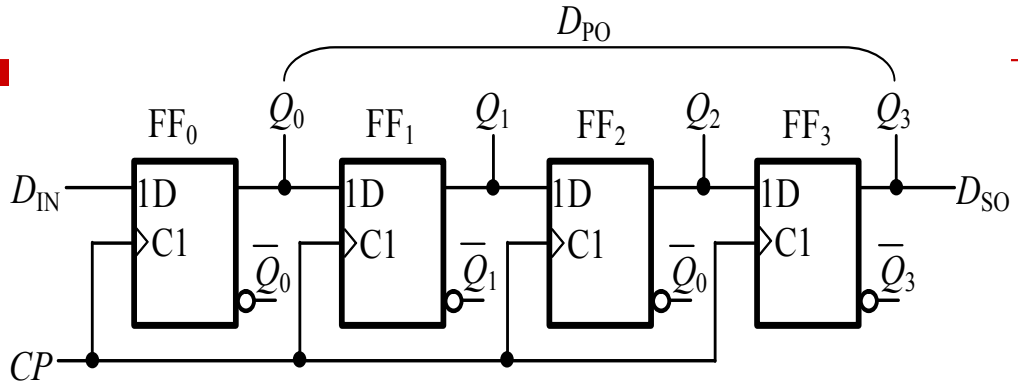


图4.4.6 扭环形计数器

例4.4.3 试对图4.4.3所示的右向移位寄存器进行建模。




```
module ShiftReg (Q,Din,CP,CLR_);
    input Din;                  //Serial Data inputs
    input CP, CLR_;             //Clock and Reset
    output reg [3:0] Q;         //Register output

always @ (posedge CP or negedge CLR_)
    if (~CLR_)
          Q <= 4'b0000;
    else begin                  //Shift right
          Q[0] <= Din;
          Q[3:1] <= Q[2:0];
    end
endmodule
```

例4.4.4一个4位的双向移位寄存器框图如图4.4.6所示。该寄存器有两个控制输入端（S1、S0）、两个串行数据输入端（Dsl、Dsr）、4个并行数据输入端和4个并行输出端，要求实现5种功能：异步置零、同步置数、左移、右移和保持原状态不变，其功能如表4.4.2所示。试用功能描述风格对其建模。

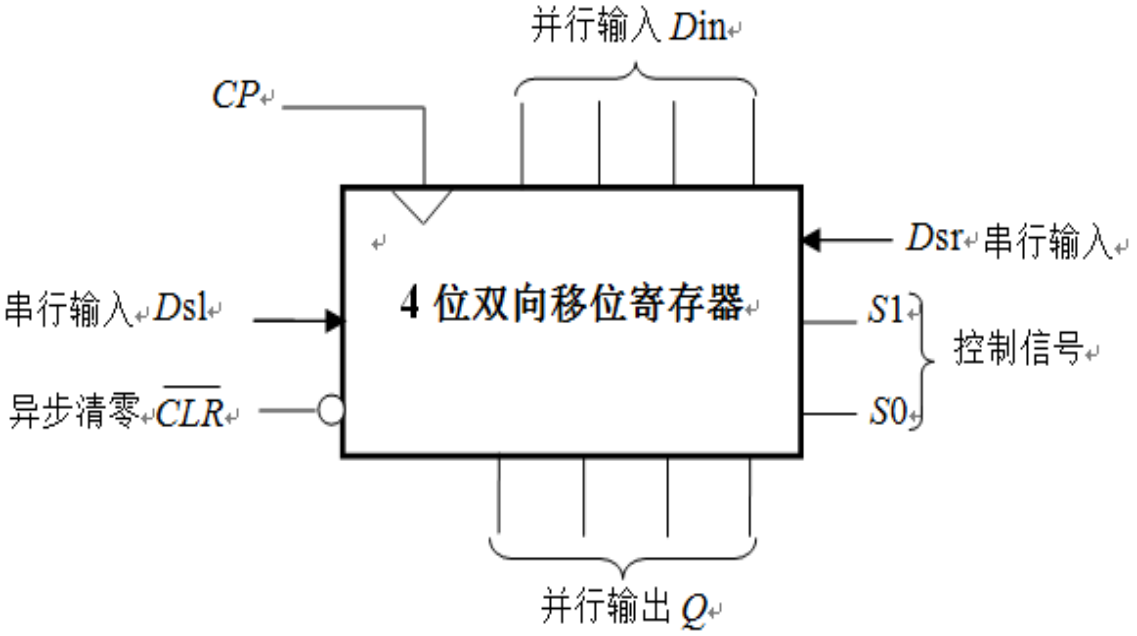


图4.4.6 双向移位寄存器框图

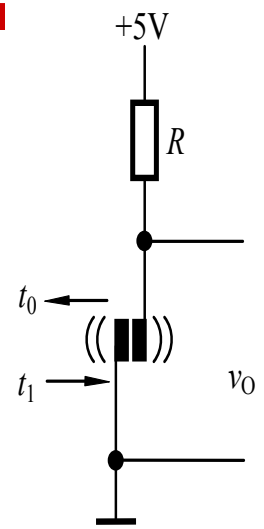
表 4.4.2 双向移位寄存器的功能表

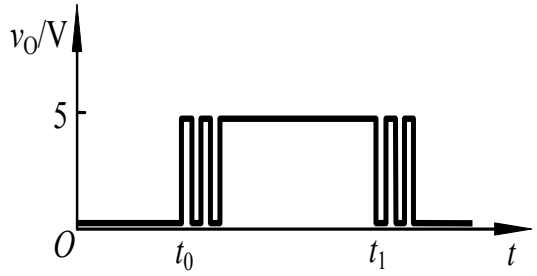
控制信号		功能
S_1	S_0	
0	0	保持
0	1	右移
1	0	左移
1	1	并行输入

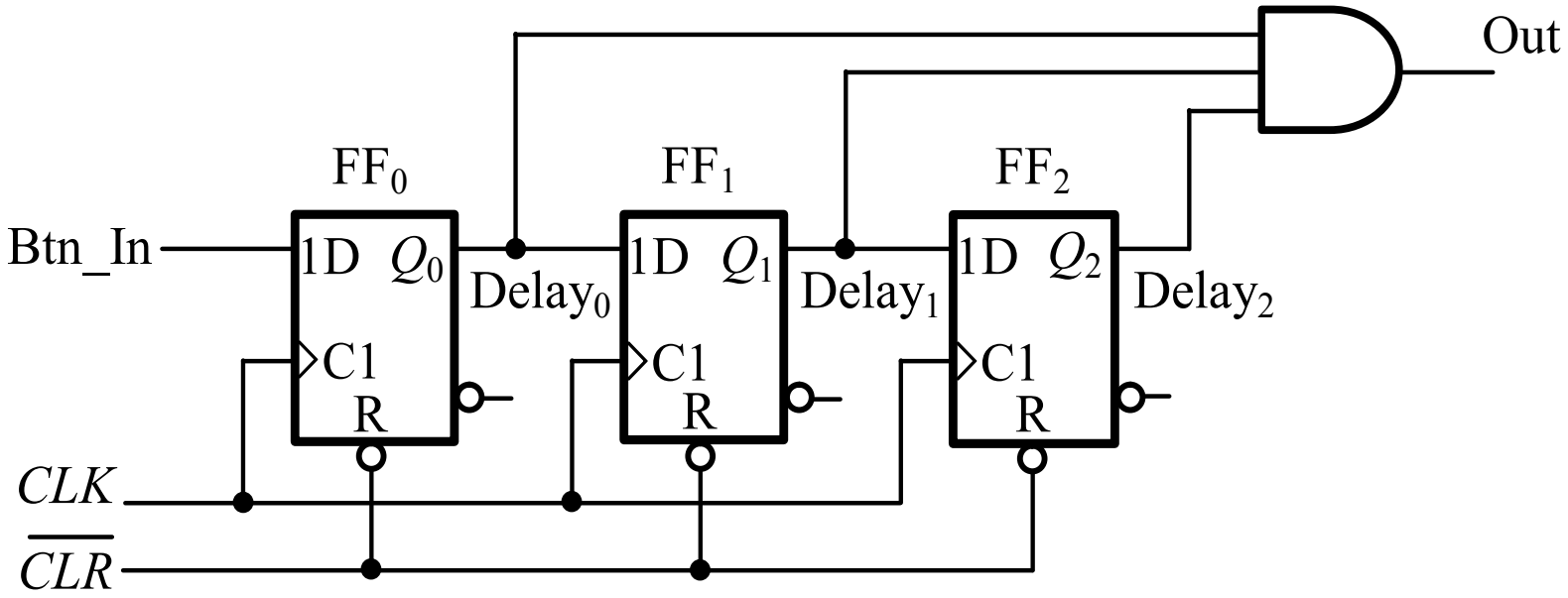
```
module UniversalShift (S1,S0,Din,Dsl,Dsr,Q,CP,CLR_);
   input S1, S0;                 //Select inputs
   input Dsl, Dsr;               //Serial Data inputs
   input CP, CLR_;               //Clock and Reset
   input [3:0] Din;              //Parallel Data input
   output reg [3:0] Q;           //Register output

always @ (posedge CP or negedge CLR_)
   if (~CLR_)
               Q <= 4'b0000;
   else
      case ({S1,S0})
        2'b00: Q <= Q;            //No change
        2'b01: Q <= {Dsr,Q[3:1]}; //Shift right
        2'b10: Q <= {Q[2:0],Dsl}; //Shift left
        2'b11: Q <= Din;          //Parallel load input
      endcase
endmodule
```

4.4.3 移位寄存器的应用电路

(1) 开关去“抖动” 电路



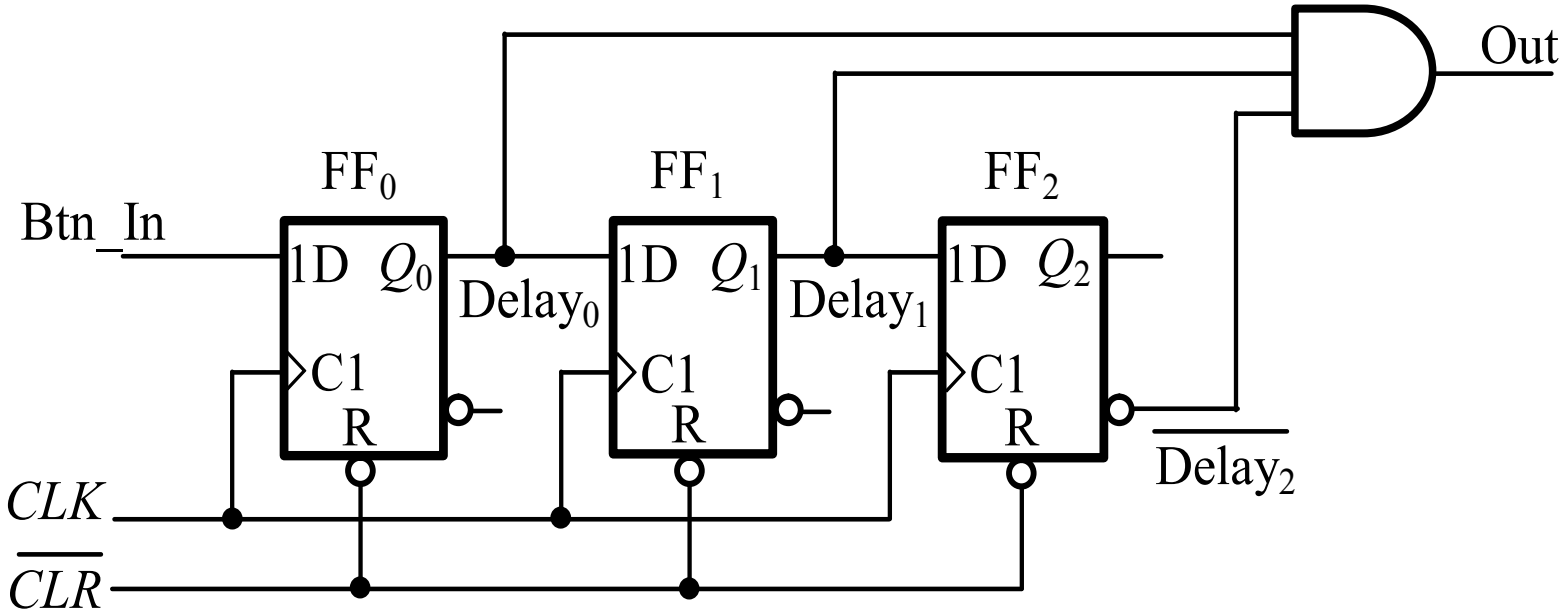




```
module Debounce (Out,Btn_In,CLK,CLR_);
   input [3:0] Btn_In;      //Button inputs
   input CLK, CLR_;         //Clock and Reset
   output [3:0] Out;        //Register output
   reg [3:0] Delay0;
   reg [3:0] Delay1;
   reg [3:0] Delay2;
 always @ (posedge CLK or negedge CLR_)
 begin
   if (~CLR_) begin
           Delay0 <= 4'b0000;
           Delay1 <= 4'b0000;
           Delay2 <= 4'b0000;
     end
   else begin              //Shift right
           Delay0 <= Btn_In;
           Delay1 <= Delay0;
           Delay2 <= Delay1;
     end
 end
 assign Out = Delay0 & Delay1 & Delay2;

endmodule
```

4.4.3 移位寄存器的应用电路

(2) 单脉冲产生电路



（1）设计块：单脉冲产生电路的代码如下：

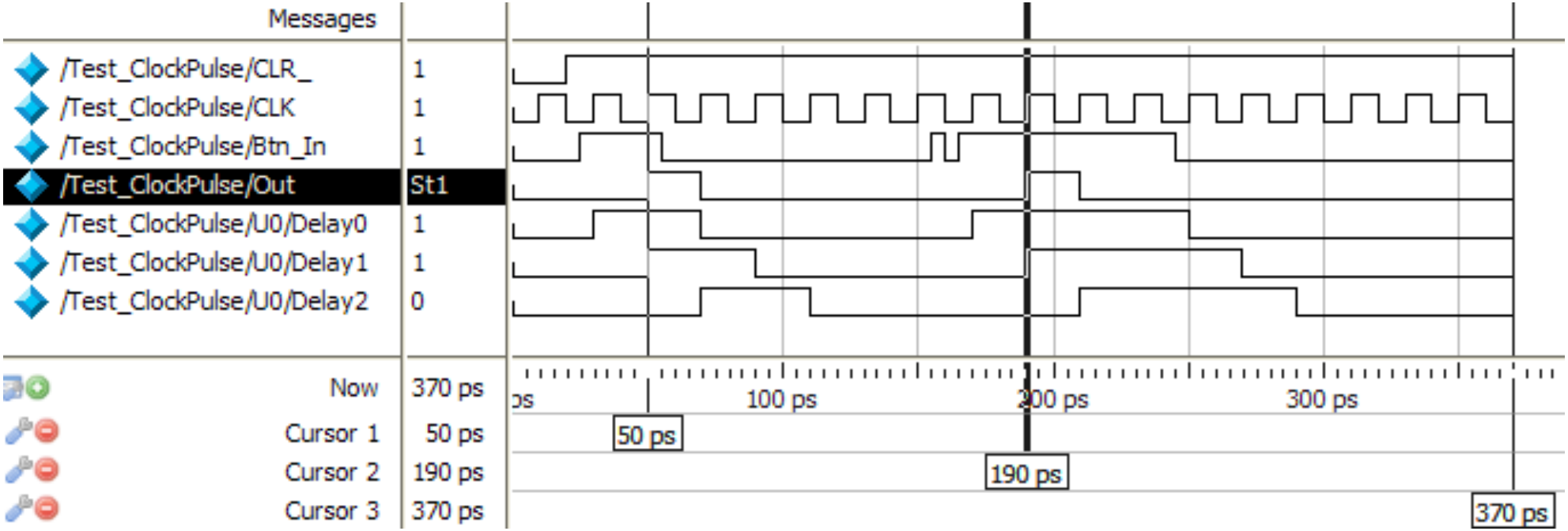
```
module ClockPulse (Out, Btn_In,CLK,CLR_);
   input  Btn_In;      //Button inputs
   input CLK, CLR_;    //Clock and Reset
   output reg  Out;    //Register output
   reg Delay0;
   reg Delay1;
   reg  Delay2;
 always @ (posedge CLK or negedge CLR_)
 begin
   if (~CLR_)
      {Delay0, Delay1, Delay2} <= 3'b000;
   else begin          //Shift right
        Delay0 <= Btn_In;
        Delay1 <= Delay0;
        Delay2 <= Delay1;
    end
 end
 assign Out = Delay0 & Delay1 & ~Delay2;
endmodule
```

（2）激励块：给输入（CLR_、Btn_In和CLK）赋值，产生激励信号。

```
module Test_ClockPulse ;
    reg  Btn_In;       //Button inputs
    reg  CLK, CLR_;    //Clock and Reset
    wire  Out;        //single  clock pulse output

 ClockPulse U0(Out, Btn_In,CLK,CLR_);

 initial begin      // CLR_
    CLR_ = 1'b0;
    CLR_ = #20 1'b1;
    #350 $stop;  //总仿真时间为370
  end

 always begin    // CLK
    CLK = 1'b0;
    CLK = #10 1'b1;
    #10;
    end
```

```
initial begin      // Btn_In
    Btn_In = 1'b0;
    Btn_In = #30 1'b1;
    Btn_In = #5 1'b0;
    Btn_In = #5 1'b1;
    Btn_In = #20 1'b0;
    #100;
    Btn_In = 1'b1;
    Btn_In = #5 1'b0;
    Btn_In = #5 1'b1;
    Btn_In = #80 1'b0;
  end

endmodule
```

(3) 仿真波形


Messages
/Test_ClockPulse/CLR_
/Test_ClockPulse/CLK
/Test_ClockPulse/Btn_In
/Test_ClockPulse/Out
/Test_ClockPulse/U0/Delay0
/Test_ClockPulse/U0/Delay1
/Test_ClockPulse/U0/Delay2
1
1
1
St1
1
1
0
Now
370 ps
Cursor 1
50 ps
Cursor 2
190 ps
Cursor 3
370 ps
100 ps
300 ps
50 ps
190 ps
370 ps

4.5 同步计数器

概 述

(1) 计数器的逻辑功能

计数器的基本功能是对输入时钟脉冲进行计数。它也可用于分频、定时、产生节拍脉冲和脉冲序列及进行数字运算等。

(2) 计数器的分类

- 按脉冲输入方式，分为同步和异步计数器
- 按进位体制，分为二进制、十进制和任意进制计数器
- 按逻辑功能，分为加法、减法和可逆计数器

概 述

- 同步计数器
  - 加计数器
    - 二进制计数器
    - 非二进制计数器
      - 十进制计数器
      - 任意进制计数器
      - ……
  - 减计数器
  - 可逆计数器
- 异步计数器
  - 加计数器
    - 二进制计数器
    - 非二进制计数器
      - 十进制计数器
      - 任意进制计数器
      - ……
  - 减计数器
  - 可逆计数器

•计数器运行时，依次遍历规定的各状态后完成一次循环，它所经过的状态总数称为计数器的"模"（Modulo），通常用*M*表示。

4.5.1 同步计数器的设计

例4.5.1 用D触发器和逻辑门设计一个同步六进制计数器。要求有一个控制信号U，

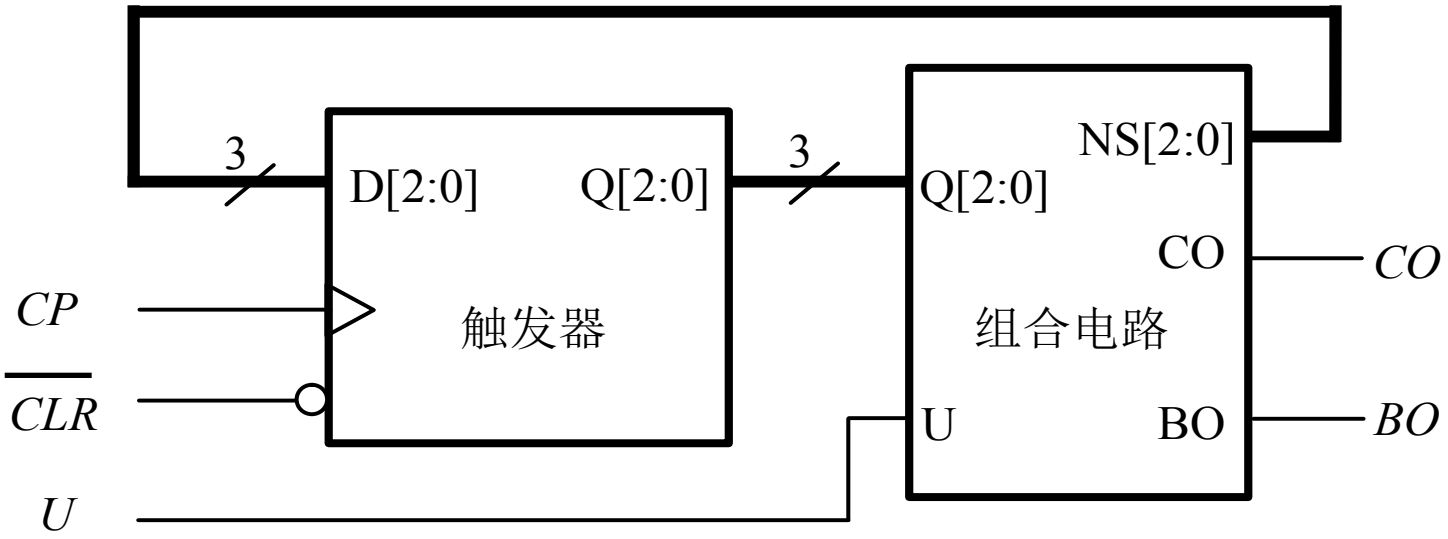
- 当U=1时，计数次序为0,1,2,3,4,5,0,1,2,…;
- 当U=0时，计数次序为5,4,3,2,1,0,5,4,3,…。
- 另外，当递增计数到最大值5时，要求输出一个高电平CO=1；当递减计数到最小值0时，也要求输出一个高电平BO=1。

解：(1) 分析设计要求，画出总体框图。

根据要求，计数器共有6个状态，我们要用D触发器来表示或区分出这6个状态，需要多少个D触发器才够呢？由于3个D触发器能够存储3位二进制数，而3位2进制数能表示2^3=8个状态，即000,001,010,011,100,101,110,111，所以只需要3个触发器就能表示6个状态。

4.5.1 同步计数器的设计

总体电路框图如下：



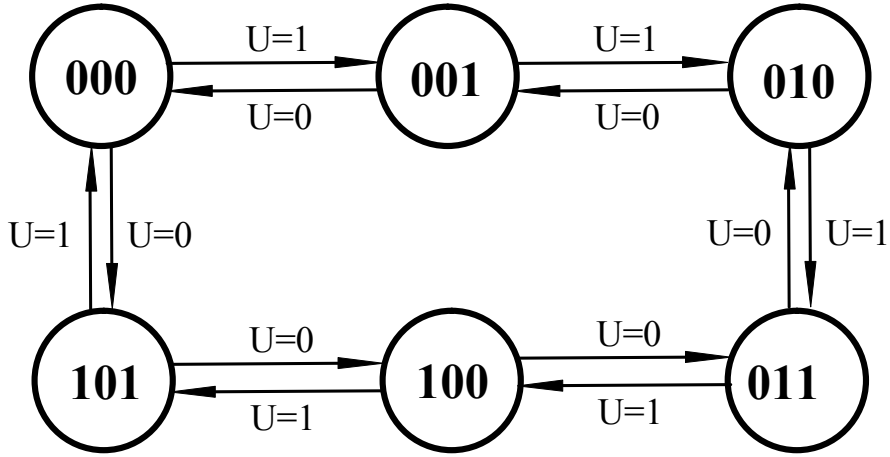
左半部分是3个D触发器，用于记录计数器的当前状态。右半部分是组合逻辑，生成下一个状态信号并产生输出信号。由于下一个状态信号与触发器的D端相连接，因此，该信号也被称为触发器的激励信号。

4.5.1 同步计数器的设计

(2) 画出状态转换图

表 4.5.1 六进制计数器状态分配的一种方案

状态名称	S0	S1	S2	S3	S4	S5
二进制值	000	001	010	011	100	101

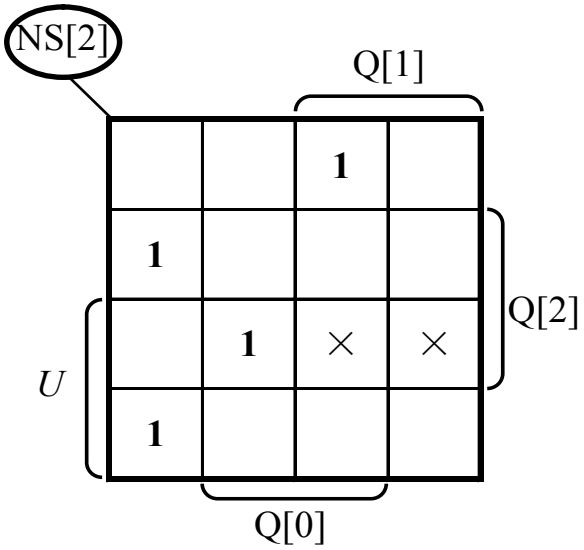


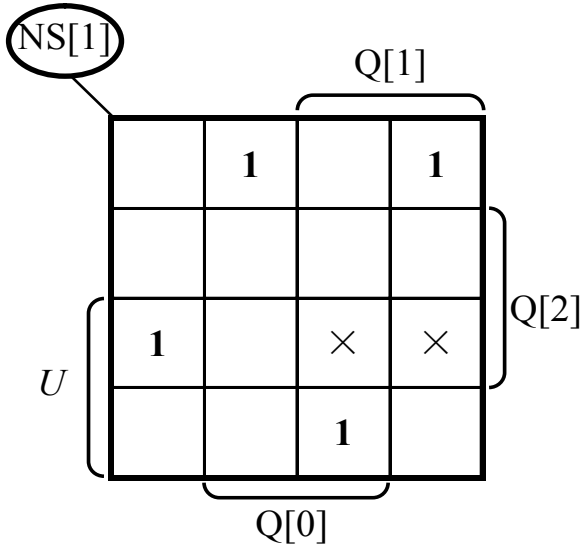
(3)列出转换表

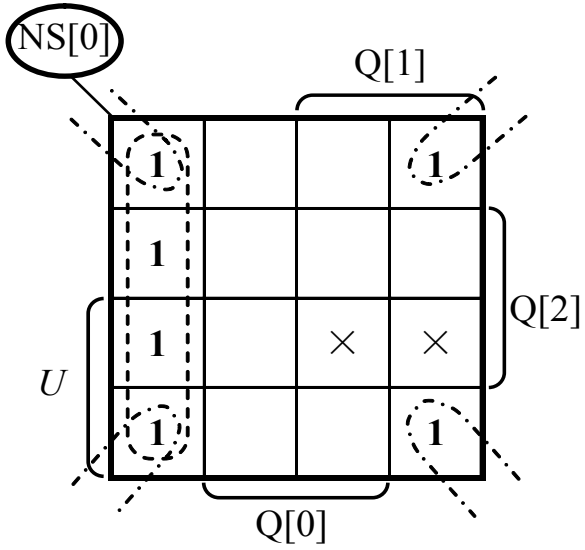
表 4.5.2 六进制计数器的状态转换表

U	Q[2]	Q[1]	Q[0]	NS[2]	NS[1]	NS[0]	CO	BO
0	0	0	0	0	0	1	0	0
0	0	0	1	0	1	0	0	0
0	0	1	0	0	1	1	0	0
0	0	1	1	1	0	0	0	0
0	1	0	0	1	0	1	0	0
0	1	0	1	0	0	0	1	0
1	0	0	0	1	0	1	0	1
1	0	0	1	0	0	0	0	0
1	0	1	0	0	0	1	0	0
1	0	1	1	0	1	0	0	0
1	1	0	0	0	1	1	0	0
1	1	0	1	1	0	0	0	0

(4)确定下一个状态的逻辑表达式







$$NS[0]=\overline{Q[1]}\cdot\overline{Q[0]}+\overline{Q[2]}\cdot\overline{Q[0]}$$

$$NS[1]=\overline{U}\cdot\overline{Q[2]}\cdot\overline{Q[1]}\cdot Q[0]+\overline{U}\cdot\overline{Q[2]}\cdot Q[1]\cdot\overline{Q[0]}+U\cdot\overline{Q[2]}\cdot Q[1]\cdot Q[0]+U\cdot Q[1]\cdot\overline{Q[1]}\cdot\overline{Q[0]}$$

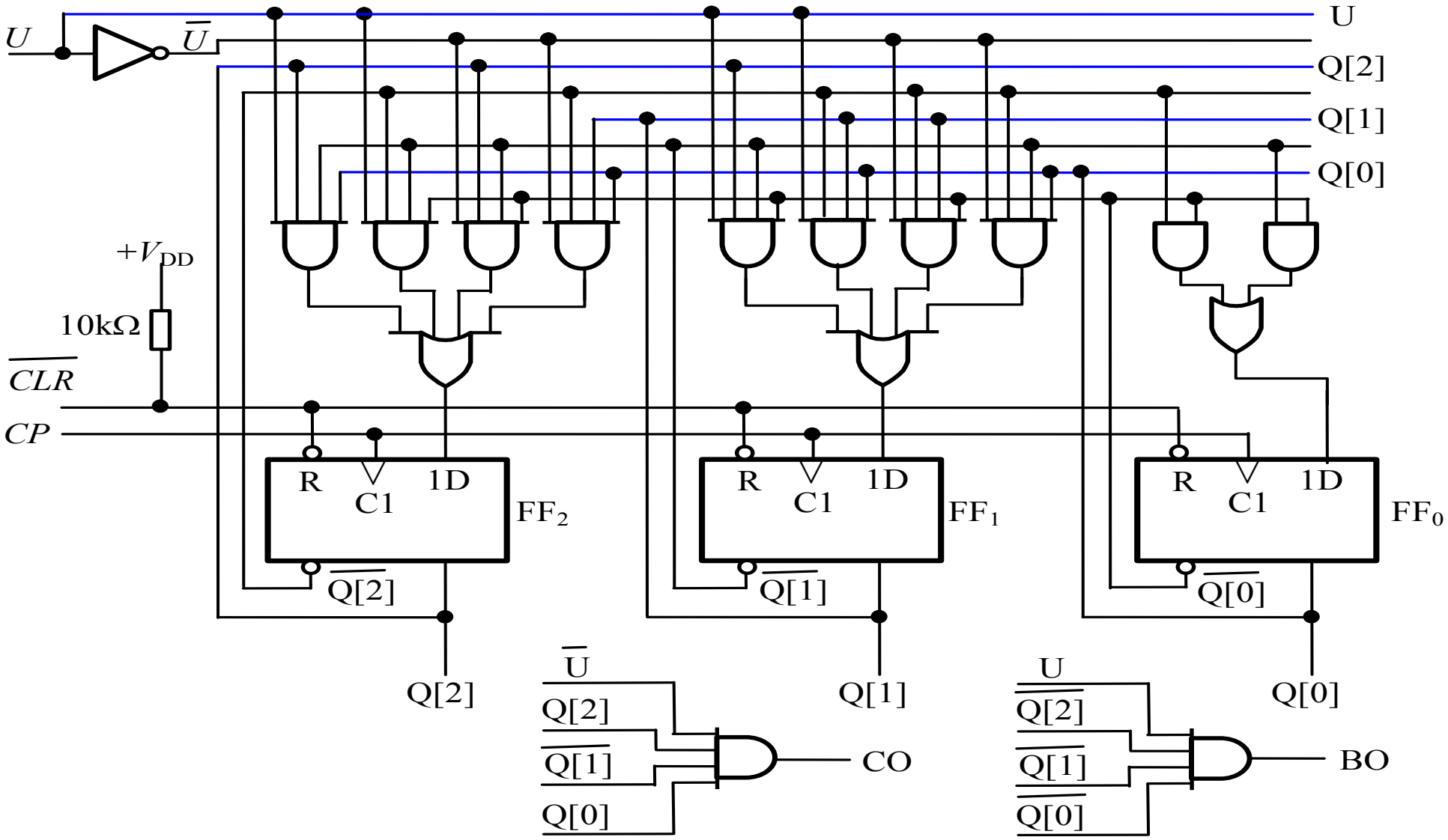
$$NS[2]=\overline{U}\cdot\overline{Q[2]}\cdot Q[1]\cdot Q[0]+\overline{U}\cdot Q[2]\cdot\overline{Q[1]}\cdot\overline{Q[0]}+U\cdot\overline{Q[2]}\cdot\overline{Q[1]}\cdot\overline{Q[0]}+U\cdot Q[2]\cdot\overline{Q[1]}\cdot Q[0]$$

同理，得到

$$CO=\overline{U}\cdot Q[2]\cdot\overline{Q[1]}\cdot Q[0]$$

$$BO=U\cdot\overline{Q[2]}\cdot\overline{Q[1]}\cdot\overline{Q[0]}$$

(5) 画出逻辑图


U
$\overline{U}$
U
Q[2]
Q[1]
Q[0]
$+V_{DD}$
10kΩ
$\overline{CLR}$
CP
R
C1
1D
FF_2
FF_1
FF_0
$\overline{Q[2]}$
$\overline{Q[1]}$
$\overline{Q[0]}$
Q[2]
Q[1]
Q[0]
$\overline{U}$
Q[2]
$\overline{Q[1]}$
Q[0]
CO
U
$\overline{Q[2]}$
$\overline{Q[1]}$
$\overline{Q[0]}$
BO

4.5.2 同步计数器的Verilog HDL建模

例4.5.2 试用Verilog HDL对图4.5.4所示电路建模

（1）设计块：

```
module Counter6 (CP,CLR_,U,Q,CO,BO);
  input CP, CLR_, U;
  output reg [2:0] Q; //Data output
  output CO,BO;
  assign CO = U & (Q == 3'd5);
  assign BO = ~U & (Q == 3'd0) & (CLR_ == 1'b1);
 always @ (posedge CP or negedge CLR_)
   if (~CLR_) Q <= 3'b000; //asynchronous clear
   else if (U==1)          //U=1,Up Counter
         Q <= (Q + 1'b1)%6;
   else  if (Q == 3'b000)
        Q <= 3'd5;
  else                     //U=0,Down Counter
         Q <= (Q - 1'b1)%6;
endmodule
```

4.5.2 同步计数器的Verilog HDL建模

（2）激励块：给输入变量（CLR_、CLK和U）赋值，产生激励信号。

```
module Test_Counter6 ;
    reg  U;              //Up/Down inputs
    reg  CLK, CLR_;      //Clock and Reset
    wire  CO,BO;         //output
    wire [2:0]  Q;       //Register output

  Counter6 U0(CLK,CLR_,U,Q,CO,BO); //实例引用设计块

initial begin      // CLR_
    CLR_ = 1'b0;
    CLR_ = #10 1'b1;
    #360 $stop;
end
```

4.5.2 同步计数器的Verilog HDL建模

```
always begin     // CLK
    CLK = 1'b0;
    CLK = #10 1'b1;
    #10;
end
initial begin    //U
    U = 1'b0;
    #190;
       U = 1'b1;
end
endmodule
```

4.5.2 同步计数器的Verilog HDL建模

(3) 仿真结果:

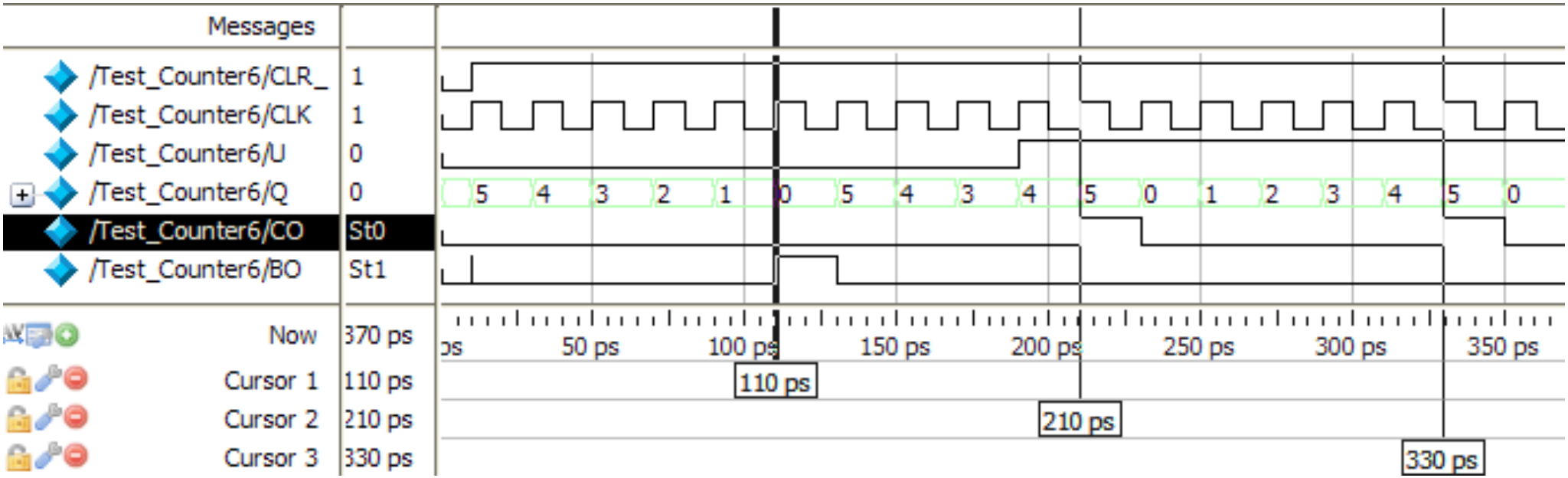


图4.5.5 六进制计数器的仿真波形

4.5.2 同步计数器的Verilog HDL建模

例4.5.4 试用Verilog HDL描述一个带有异步置零功能的十进制同步计数器

```
//Non-Binary counter with enable
module M10_counter (EN,CP,CLR_,Q);
   input EN,CP,CLR_;
   output reg [3:0] Q;    //Data output
always @(posedge CP or negedge CLR_)
  if (!CLR_)              //异步清零
             Q = 4'b0000;
  else if (EN) begin
       if (Q >= 4'b1001)
            Q <= 4'b0000;    //出错处理
       else Q <= Q + 1'b1;   //递增计数
       end
  else
             Q <= Q;          //保持计数值不变
endmodule
```

例：请描述具有异步清零、同步置数的计数器，并要求具有可逆计数和保持的功能。

```
module cntr(q, aclr, clk, func, d);
input aclr, clk;
input [7:0] d;
//Controls the functionality
input [1:0] func;
output [7:0] q;
reg [7:0]q;
```

```
always @(posedge clk or posedge aclr)
begin
  if (aclr)   q <= 8'h00;
  else  case (func)
     2'b00: q <= d; // Loads the counter
     2'b01: q <= q + 1; // Counts up
     2'b10: q <= q - 1; // Counts down
     2'b11: q <= q;
   endcase
end
endmodule
```

例：假设有一个50 MHz时钟信号源，试用Verilog HDL设计一个分频电路，以产生1Hz的秒脉冲输出，要求输出信号的占空比为50%。。

解：设计一个模数为$25*10^6$的二进制递增计数器，其计数范围是0~24999999，每当计数器计到最大值时，输出信号翻转一次，即可产生1Hz的秒脉冲。

```
module Divider50MHz(CR,CLK_50M,
CLK_1HzOut);
  input    CR,CLK_50M;
  output reg CLK_1HzOut;
  reg [24:0] Count_DIV; //内部节点
parameter CLK_Freq = 50000000;
parameter OUT_Freq = 1;
```

```
always @(posedge CLK_50M or negedge CR)
begin
if(!CR)  begin
CLK_1HzOut <= 0;
Count_DIV <= 0;
end
else  begin
if( Count_DIV < (CLK_Freq/(2*OUT_Freq-1)) )
          Count_DIV <= Count_DIV+1'b1;
 else begin
              Count_DIV    <=    0;
              CLK_1HzOut  <=  ~CLK_1HzOut;
       end
 end
end
endmodule
```

```
always @(posedge CLK_50M or negedge CR)
begin
  if(!CR)  begin
                CLK_1HzOut <= 0;
                Count_DIV <= 0;
            end
  else  begin
          if( Count_DIV < (CLK_Freq/2*OUT_Freq-1) )
                 Count_DIV <= Count_DIV+1'b1;
          else begin
                 Count_DIV    <=    0;
                 CLK_1HzOut  <=  ~CLK_1HzOut;
                end
          end
end
endmodule
```

4.6 Verilog HDL函数与任务的使用

4.6.1 函数（function）说明语句

1. 函数的定义

函数定义部分可以出现在模块说明中的任何位置，其语法格式如下：

```
function <返回值类型或位宽> <函数名>;
    <输入参量与类型声明>
    <局部变量声明>
    行为语句;
endfunction
```

4.6.1 函数（function）说明语句

2. 函数的调用

函数调用是表达式的一部分，其格式如下：

<函数名> （<输入表达式1>，.....<输入表达式n>）；

其中输入表达式的排列顺序必须与各个输入端口在函数定义结构中的排列顺序一致。

例4.6.1 用定义function与调用function的方法完成4选1数据选择器设计。

（1）设计块代码如下：

```
`timescale  1ns/1ns     //定义时间单位
module  SEL4to1  ( A, B, C, D, SEL, F );
    input  A, B, C, D;
    input  [1:0] SEL;
    output  F;
  assign F= SEL4to1FUNC ( A, B, C, D, SEL );//调用函数
    //定义函数
    function  SEL4to1FUNC;  //注意此行不需要端口名列表
      input  A1, B1, C1, D1; //函数的输入参量声明
      input  [1:0] SEL1;       //函数的输入参量声明
      case(SEL1)
       2'd0:SEL4to1FUNC = A1;
       2'd1:SEL4to1FUNC = B1;
       2'd2:SEL4to1FUNC = C1;
       2'd3:SEL4to1FUNC = D1;
     endcase
   endfunction
endmodule
```

（2）激励块

```
`timescale  1ns/1ns     //定义时间单位
module Test_SEL4to1();
//declare variables to be connected to inputs
   reg IN0, IN1, IN2, IN3;
   reg [1:0]SEL;
   wire OUT; //Declare output wire

//Instantiate the design block
  SEL4to1 mymux( .A(IN0), .B(IN1), .C(IN2), .D(IN3), .SEL(SEL), .F(OUT) );

//Stimulate the inputs
initial
begin
        IN0 = 1;  IN1 = 0; IN2 = 0; IN3 = 0; //set input lines
   #10  $display ($time, "\t IN0= %b, IN1= %b, IN2= %b, IN3= %b \n", N0, IN1, IN2, IN3);
   #10   SEL = 2'b00;   //choose IN0
   #30   SEL = 2'b01;   //choose IN1
   #30   SEL = 2'b10;   //choose IN2
   #100 SEL = 2'b11;   //choose IN3
   #100 $stop; //总仿真时间为280ns
end
```

```
always begin
      #5 IN2 = ~IN2;  //每隔5ns，IN2改变一次状态
end
always begin
      #10 IN3 = ~IN3; //每隔10ns，IN3改变一次状态
end
//Monitor the outputs
initial
     $monitor ($time,  "\t SEL=%b,  OUT = %b \n", SEL, OUT);
endmodule
```

（3）仿真结果：

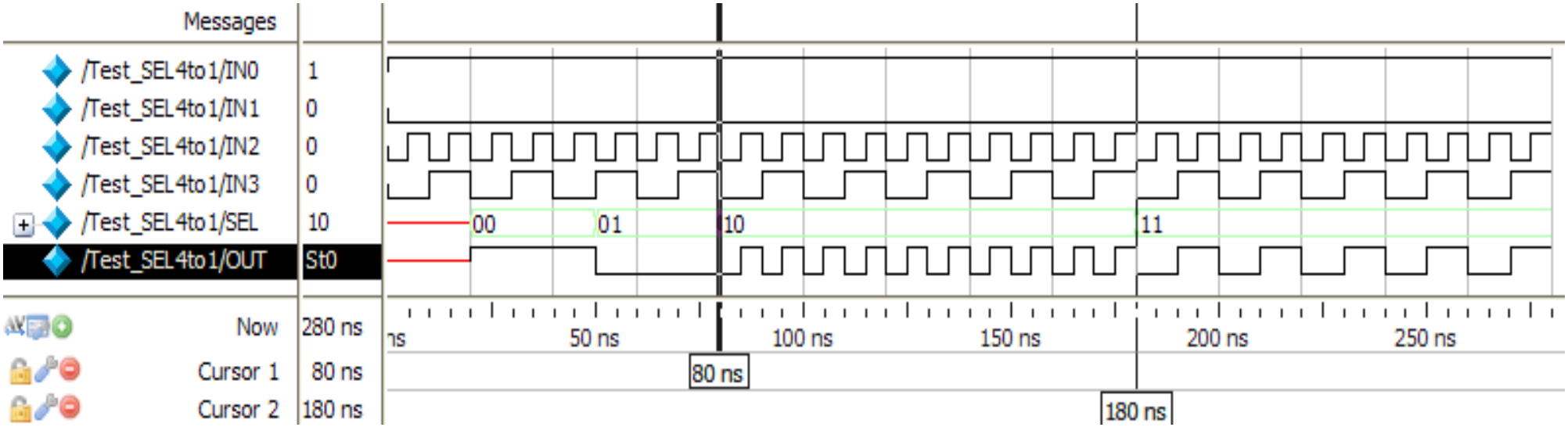


图4.6.1　4选1数据选择器的仿真波形

例：2选1数据选择器

```
module MUX2_1(A,B,SEL,OUT);
  output OUT;
  input A,B,SEL;

  assign OUT = SEL2_1_FUNC(A,B,SEL);

  function SEL2_1_FUNC;
    input A,B,SEL;
          if (SEL==0)
                  SEL2_1_FUNC=A;
           else  SEL2_1_FUNC=B;
   endfunction

endmodule
```

4.6 Verilog HDL函数与任务的使用

4.6.2 任务（task）说明语句

1. 任务的定义

```
task <任务名>;
        端口与类型说明;
        变量声明;
    语句1;
    语句2;
    .....
    语句n;
endtask
```

4.6.2 任务（task）说明语句

2. 任务的调用

一个任务由任务调用语句调用，任务调用语句给出传入任务的参数值和接收结果的变量值，其语法如下：

```
<任务名>  （端口1，端口2，……，端口n）；
```

4.7 m序列码产生电路设计

m 序列又叫做伪随机序列、伪噪声(pseudo noise，PN)码或伪随机码，是一种可以预先确定并可以重复地产生和复制、又具有随机统计特性的二进制码序列。

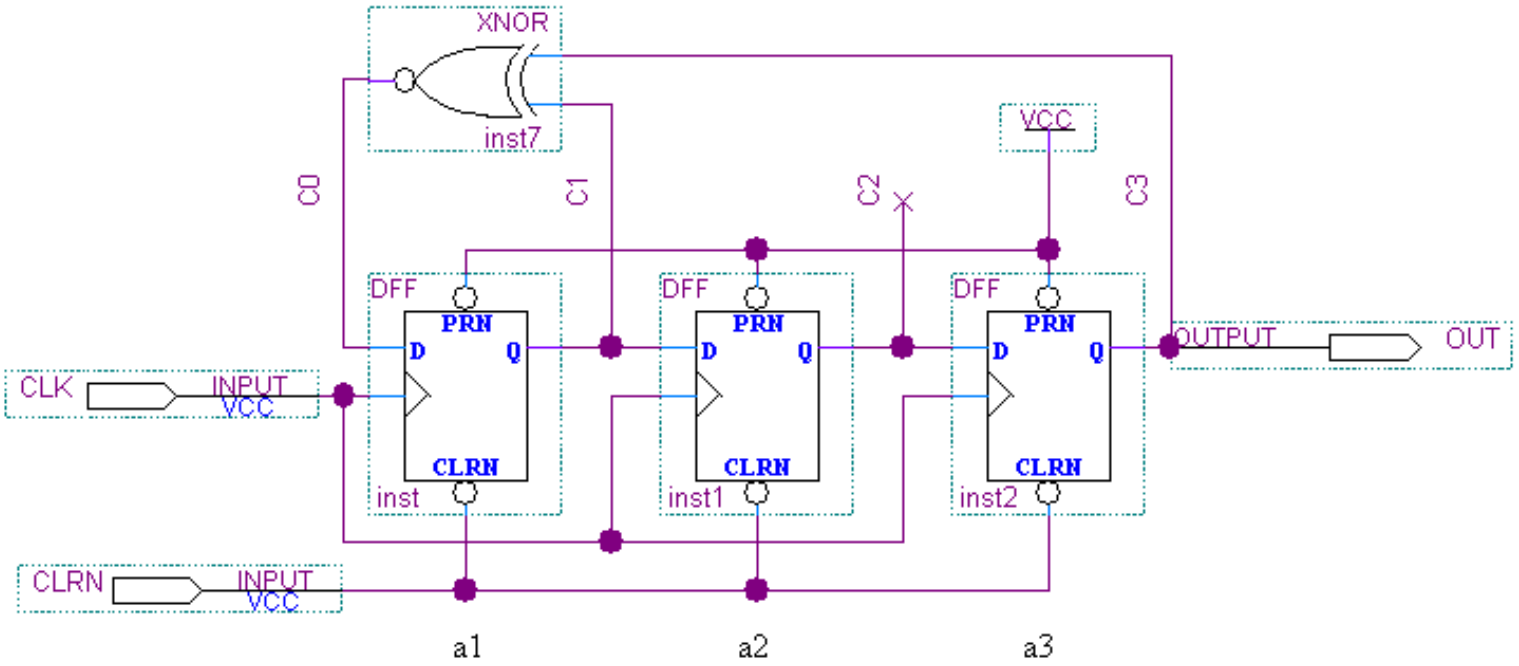
伪随机序列一般用二进制表示，每个码元（即构成m序列的元素）只有“0”或“1”两种取值，分别与数字电路中的低电平或高电平相对应。

m 序列是对最长线性反馈移位寄存器序列的简称，它是一种由带线性反馈的移位寄存器所产生的序列，并且具有最长周期。

4.7 m序列码产生电路设计

图4.7.1所示是一种3位m序列产生器，它将最后两级触发器的输出通过同或门反馈到第一级的输入端。

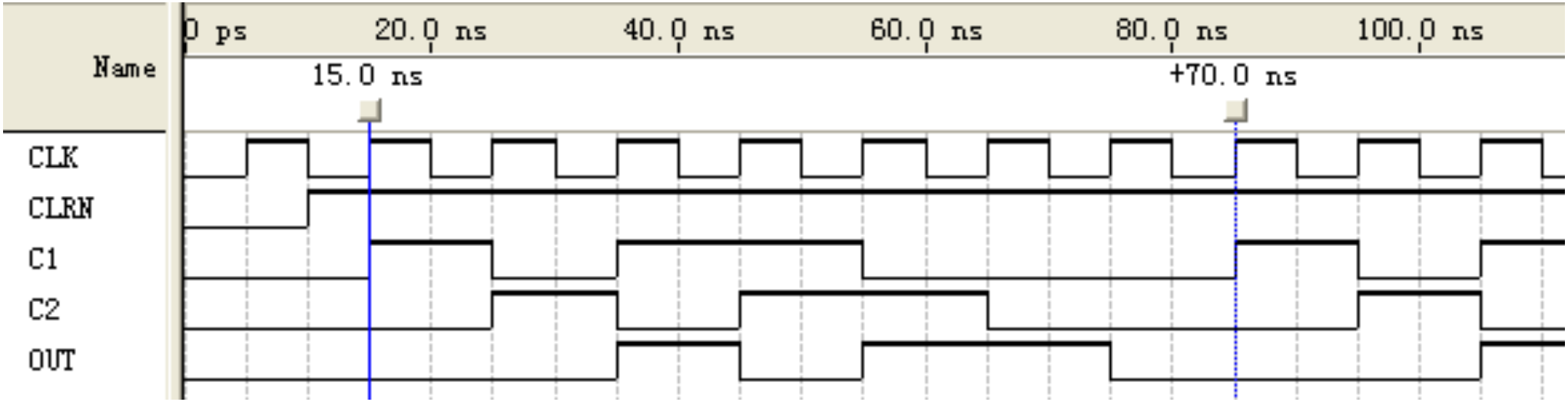
其工作原理是：在清零后，3个触发器的输出均为0，于是同或门的输出为1，在时钟触发下，每次移位后各级寄存器状态都会发生变化。



4.7 m序列码产生电路设计

分析该电路得到如图4.7.2所示的仿真波形图，其中任何一级触发器（通常为末级）的输出都是一个周期序列（或者称为m序列），但各个输出端的m序列的初始相位不同。m序列的周期不仅与移位寄存器的级数有关，而且与线性反馈逻辑和初始状态有关。

此外，在相同级数的情况下，采用不同的线性反馈逻辑所得到的周期长度是不同的。



4.7 m序列码产生电路设计

该电路的状态转换图如图4.7.3所示。

共有$2^3-1=7$个状态

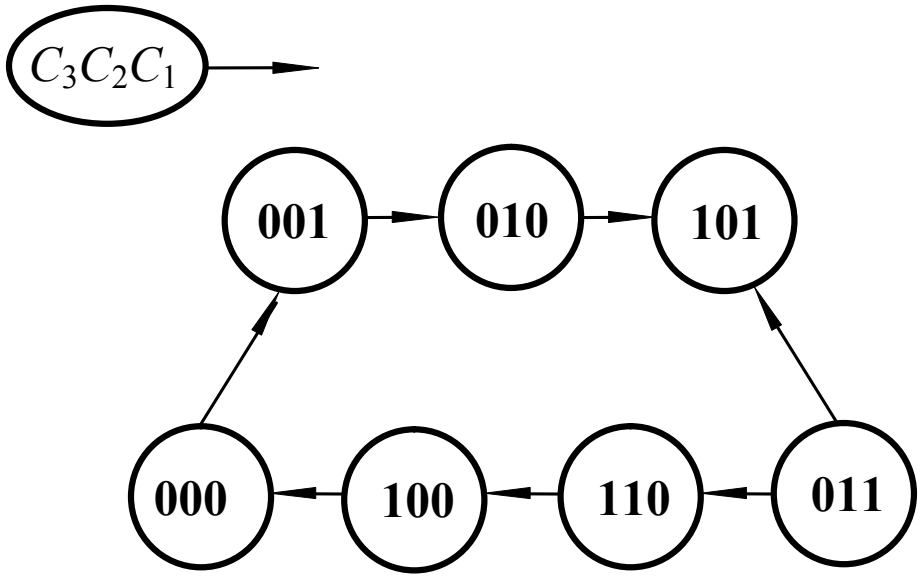


图4.7.3三位*m*序列状态转换图

4.7 m序列码产生电路设计

- 通常，将类似于图4.7.1所示结构的m序列产生器称为简单型码序列发生器（Simple Shift Register Generator, SSRG），它的一般结构如图4.7.4所示。
- 图中，各个触发器a_i（i=1,2，…r）构成移位寄存器， 代表异或运算，C_0、C_1、C_2、……、C_r是反馈系数，也是特征多项式的系数。系数取值为1表示反馈支路连通，0表示反馈支路断开。

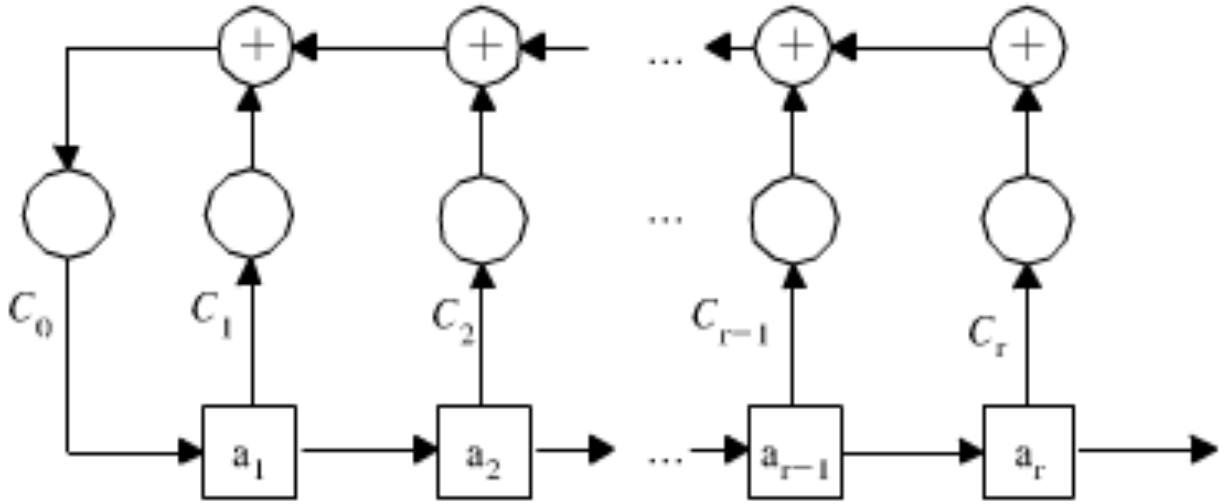


图4.7.4 SSRG电路的结构

4.7 m序列码产生电路设计

- 对于SSRG结构的m序列发生器，其特征多项式的一般表达式为

$$f(x)=C_0x^0+C_1x^1+C_2x^2+\cdots+C_rx^r$$

- 特征多项式系数决定了一个m序列的特征多项式，同时也就决定了一个m序列。

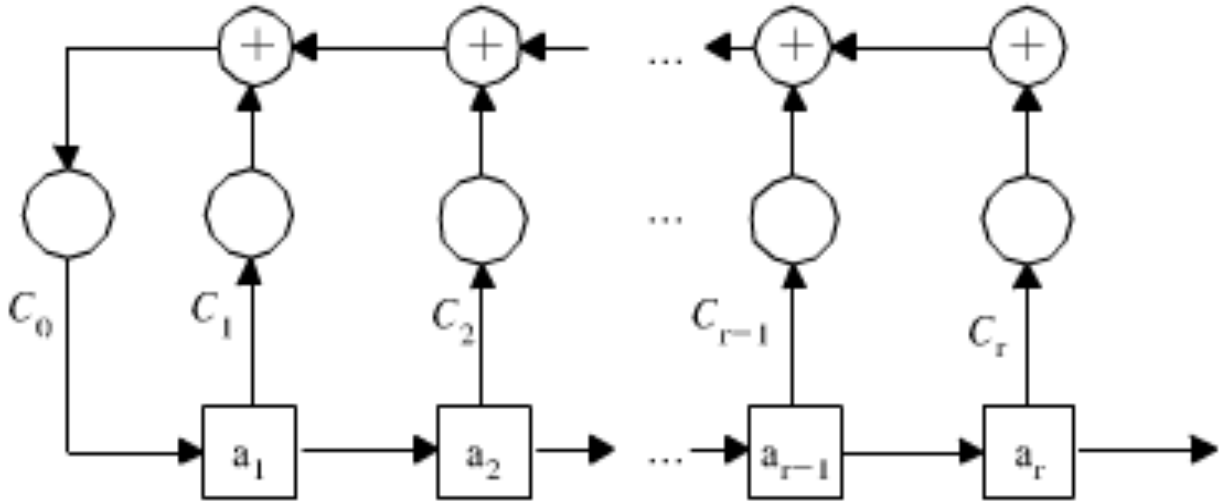


图4.7.4 SSRG电路的结构

表4.7.1给出了部分*m*序列的反馈系数，系数的值是用八进制数表示的。

表 4.7.1　*m* 序列反馈系数表

寄存器级数	*m* 序列长度	*m* 序列产生器反馈系数(八进制数)
2	3	7
3	7	13
4	15	23
5	31	45, 67, 75
6	63	103, 147, 155
7	127	203, 211, 217, 235, 277, 313, 325, 345, 367
8	255	435, 453, 537, 543, 545, 551, 703, 747
9	511	1021, 1055, 1131, 1157, 1167, 1175
10	1023	2011, 2033, 2157, 2443, 2745, 3471
11	2047	4005, 4445, 5023, 5263, 6211,7363
12	4095	10123, 11417, 12515, 13505, 14127, 15053
13	8191	20033, 23261, 24633, 30741, 32535, 37505
14	16383	42103, 51761, 55753, 60153, 71147, 67401
15	32767	100003, 110013, 120265, 133663, 142305,164705
16	65535	210013, 233303, 307572, 311405, 347433,375213
17	131071	400011, 411335, 444257, 527427, 646775, 714303
18	262143	1000201, 1000241, 1025711, 1703601
19	524287	2000047,2020471, 2227023, 2331067, 2570103, 3610353
20	1048575	4000011, 4001051, 4004515, 6000031

4.7 m序列码产生电路设计

- 根据多项式的系数可以产生m序列。
- 例如，想要产生一个码长为31的m序列，寄存器的级数r = 5，从表4.7.1中查到反馈系数有三个，分别为45、67、75，可以从中选择反馈系数45来构成m序列产生器，因为使用45时，反馈线最少，构成的电路最简单。
- **45为八进制数**，写成二进制数为100101，这就是特征多项式的系数，即

$$C5\ C4\ C3\ C2\ C1\ C0=100101$$

表明C5、C2、C0三条反馈支路是连通的，另外三条反馈支路C4、C3、C1是断开的。其电路如图4.7.5所示。

4.7 m序列码产生电路设计

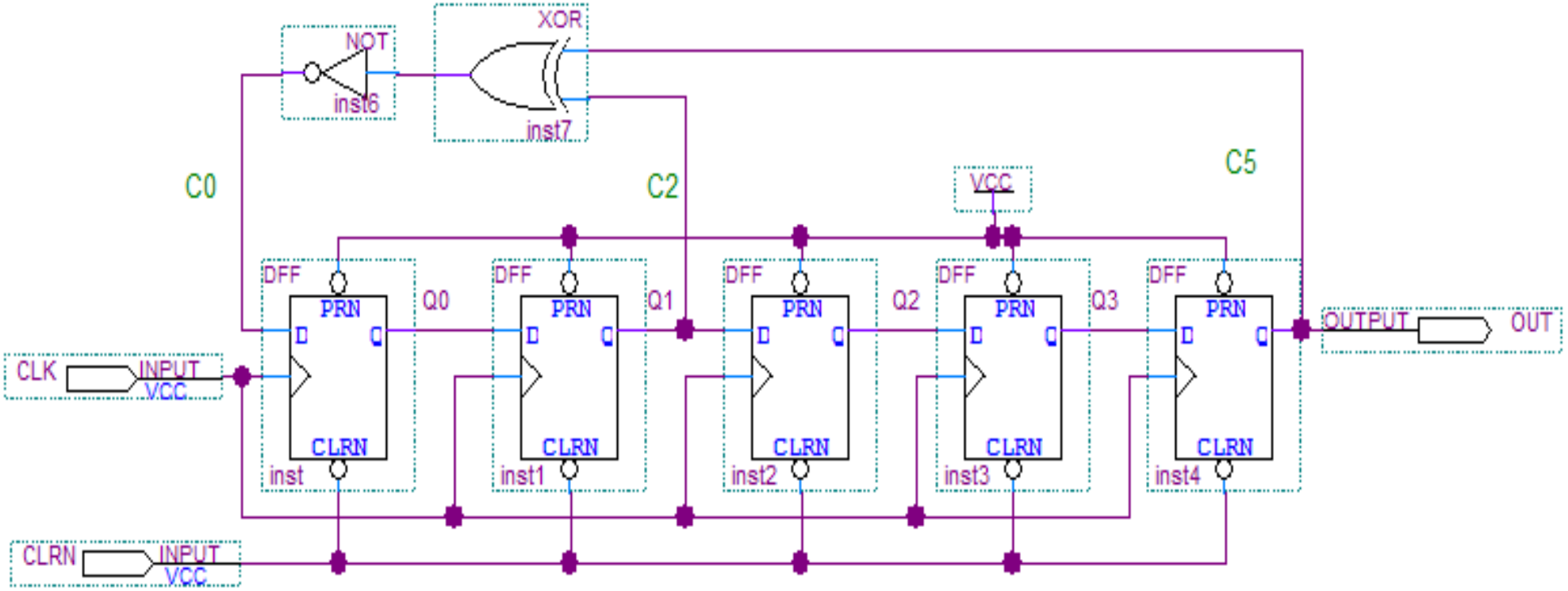


图4.7.5 五位 m 序列产生器

4.7 m序列码产生电路设计

图4.7.5 Verilog HDL程序如下:

```
module m5(CLK, CLRN, OUT);
    input CLK, CLRN;    //输入端口
    output OUT;         //输出端口
    reg[4:0] Q;         //中间节点
    wire C0;
assign C0 = ~(Q[4] ^ Q[1]);   //反馈
assign OUT = Q[4];            //输出信号
always@(posedge CLK or negedge CLRN)
begin
    if(!CLRN )
        Q[4:0] <= 5'b00000;     //异步清零
    else
        Q[4:0] <= {Q[3:0],C0}; //移位
end
endmodule
```

4.7 m序列码产生电路设计

图4.7.5 仿真波形：