触发器、寄存器和计数器

计算机系　齐悦

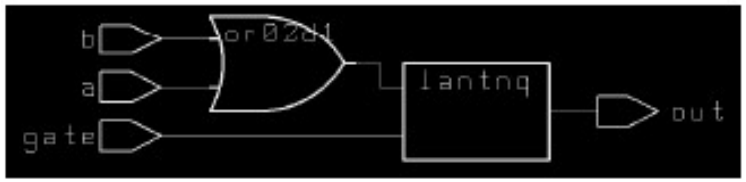
北京科技大学

单选题 2分

设置

你认为下面Verilog代码描述的电路功能（　　）

```
always @(a or b or gate )
begin
  if (gate)
     out = a | b;
end
```



A 正确

B 错误

C 不知道

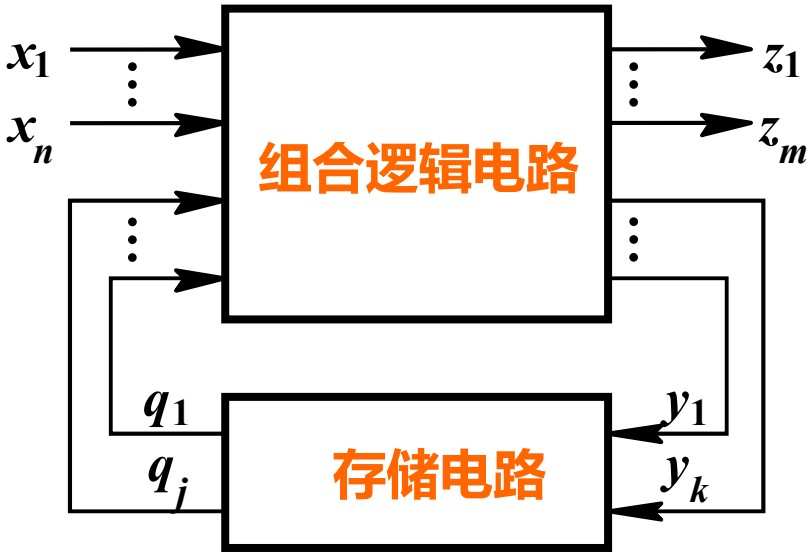
提交

目录

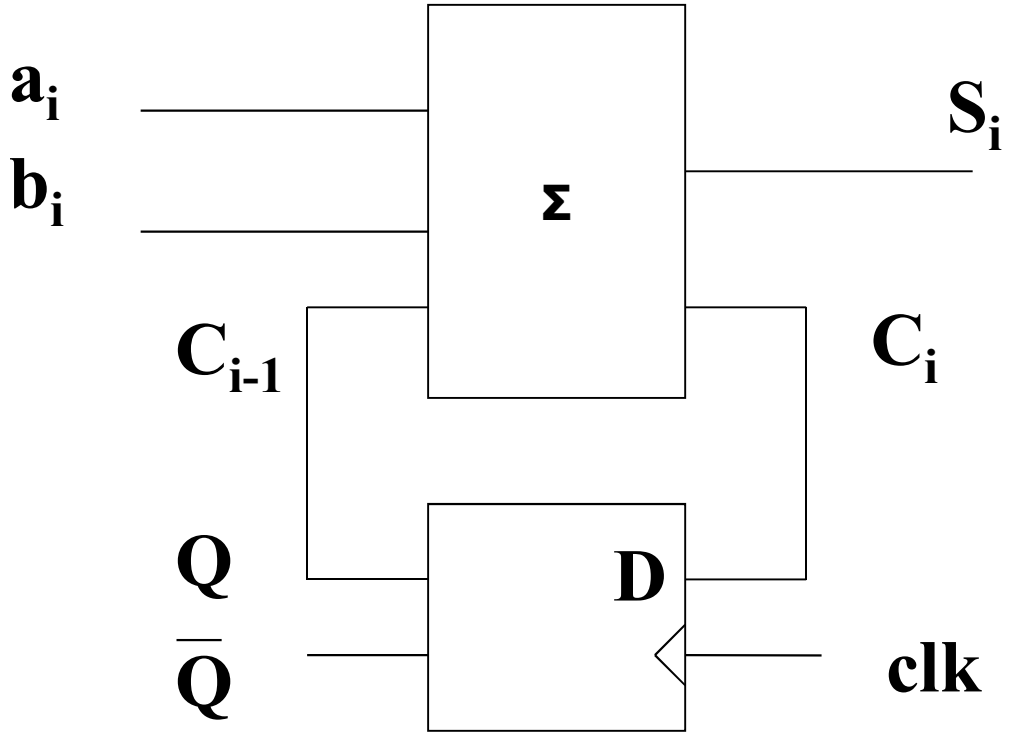
时序逻辑电路——有记忆功能

- 从逻辑上讲， 时序电路在任一时刻的输出不仅取决于该时刻的输入，而且还和电路原来的状态有关。

- 从结构上讲， 时序电路不仅仅由逻辑门组成，还包含存储信息的有记忆能力的电路：触发器、寄存器等



时序电路举例：串行加法器



时序电路的分类

按照触发器的动作特点

同步时序逻辑电路

所有触发器的状态变化都是在同一时钟信号作用下同时发生的。

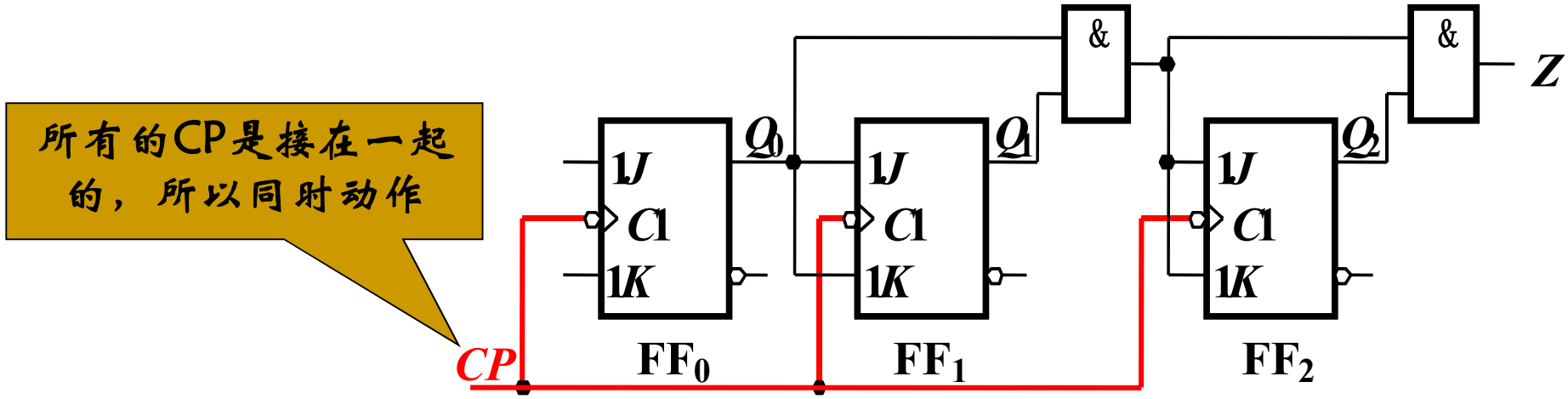
异步时序逻辑电路

没有统一的时钟脉冲信号，各触发器状态的变化不是同时发生，而是有先有后。

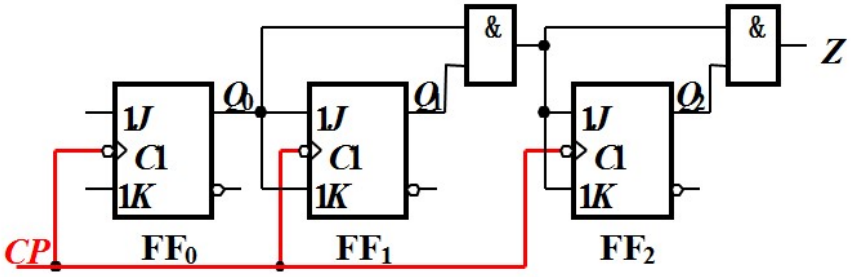
同步时序逻辑电路

- 所有的存储元件都在时钟脉冲(Clock Pulse，CP)统一控制下，用触发器作为存储元件。只有一个“时钟信号”，所有的内部存储器，只会在时钟的边沿时候改变。

  几乎现在所有的时序逻辑都是“同步逻辑”



同步时序逻辑电路

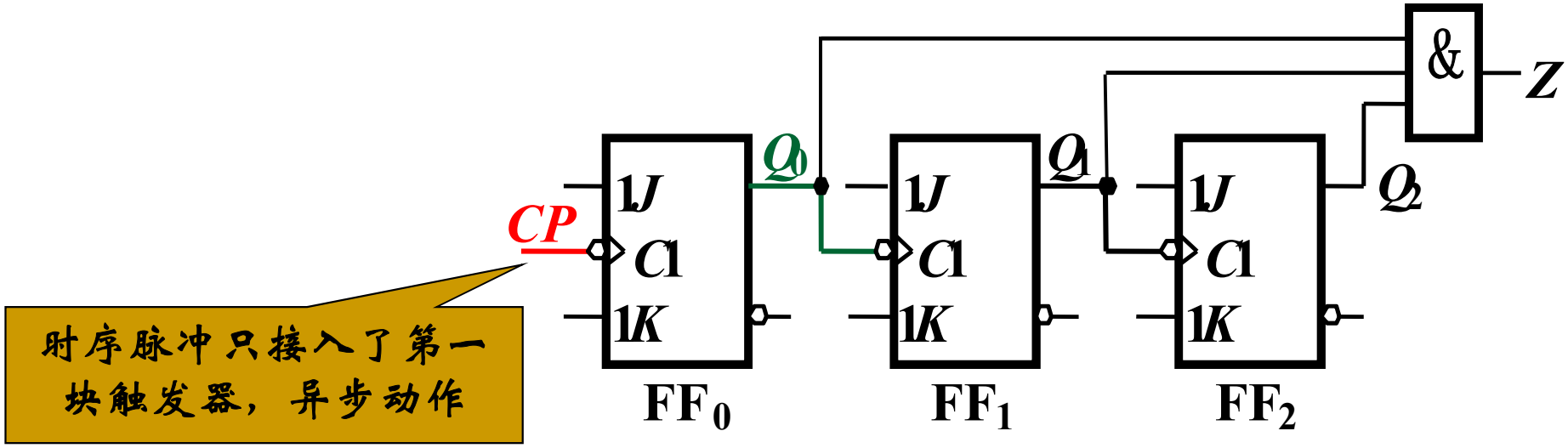


- 优点：
  - 简单。每个电路里的运算必须要在时钟的两个脉冲之间固定的间隔内完成，称为一个时钟周期。满足该条件下的电路是可靠的。
- 缺点：
  - 时钟是高频率信号，而时钟必须分布到各个触发器而不管触发器是否要工作，消耗大量功耗
  - 最快的时钟频率是由电路中最慢的逻辑路径（关键路径）决定的，因此一定程度限制工作的最高频率（流水线）

异步时序逻辑电路

- 没有统一的时钟源，最基本的储存元件是锁存器。锁存器可以在任何时间改变它的状态，依照其它的锁存器信号的变动，它们新的状态就会被产生出来变。

异步电路的复杂度随着逻辑门的增加，而复杂性也快速的增加，因此他们大部分仅仅使用在小规模的应用



同步 VS. 异步

- 同步时序电路
  - 所有存储单元状态变化都由同一时钟信号控制，比较容易满足建立时间和保持时间的要求。
  - 同步时序电路可以很好地避免毛刺
  - 有利于器件移植
  - 有利于静态时序分析（STA）和验证
- 异步时序电路
  - 不存在全局时钟，各触发器翻转的时间不定，设计复杂性增加
  - 电路的核心由组合逻辑实现，比如异步FIFO/RAM的读写信号
  - 最大的问题是容易产生毛刺，影响电路可靠性、稳定性

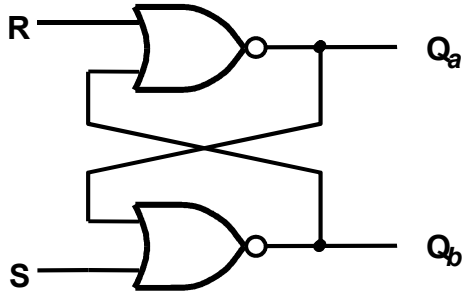
目录

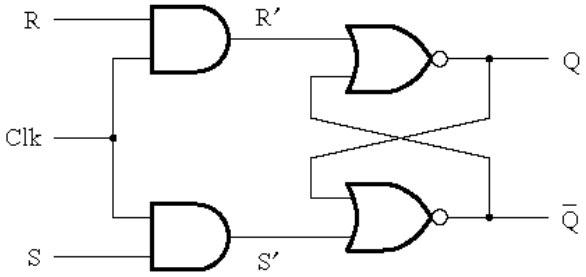
时序电路描述方法

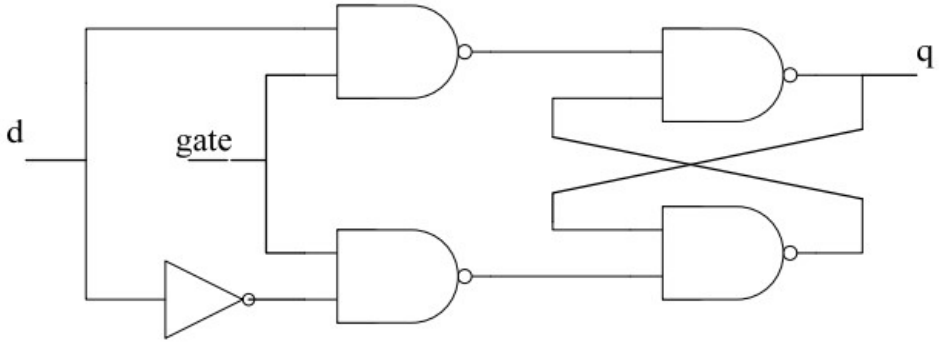
- 定义存储单元
  - Verilog 通过定义寄存器变量为存储单元建模
  - 对于寄存器变量赋值只能使用过程赋值（always）语句
- 考虑存储单元的复位方式、置位条件
- 时钟控制方式
  - always语句的时间控制列表中posedge、negedge、电平触发等
- 组合通路
- 控制通路

时序逻辑电路——锁存器

- 锁存器是一种电平敏感的存储器
  - 优点是占触发器资源少，缺点是容易产生毛刺。
  - 典型的例子有SR(RS)锁存器与D锁存器。

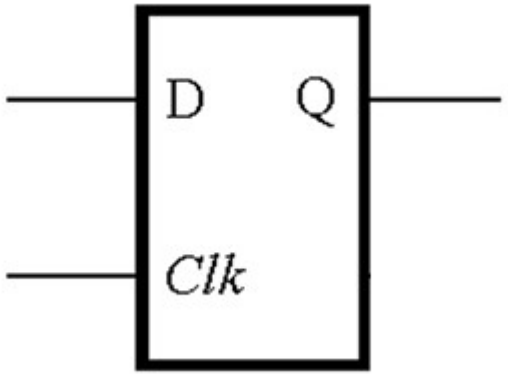






用Verilog描述存储器——D锁存器

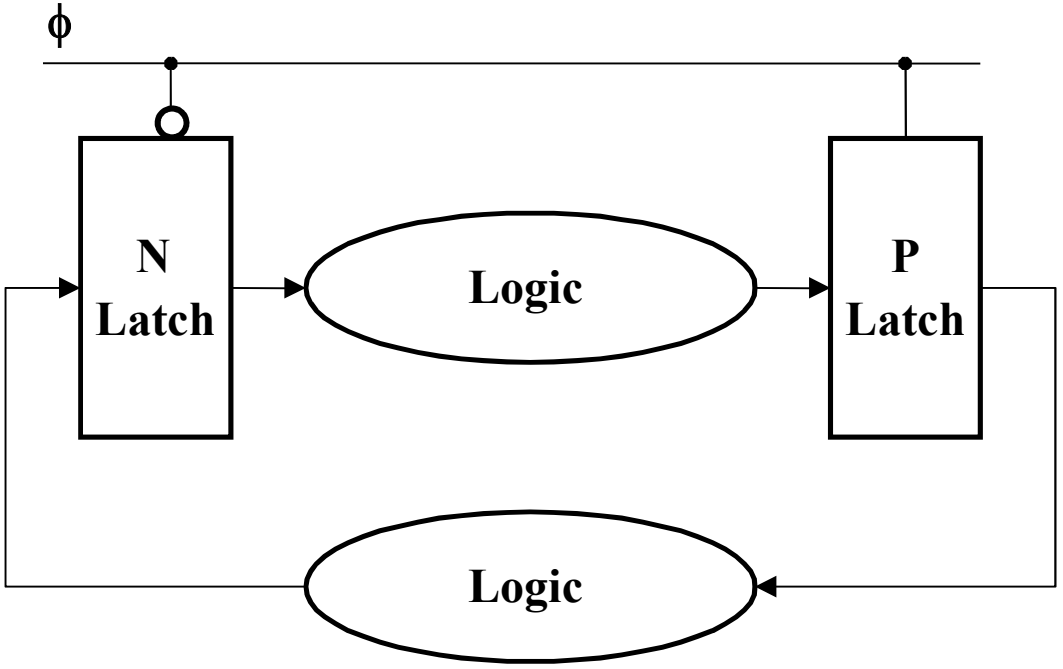
```
module D_latch (D, Clk, Q);
    input D, Clk;
    output reg Q;

    always @(D, Clk)
        if (Clk)
            Q = D;
endmodule
```

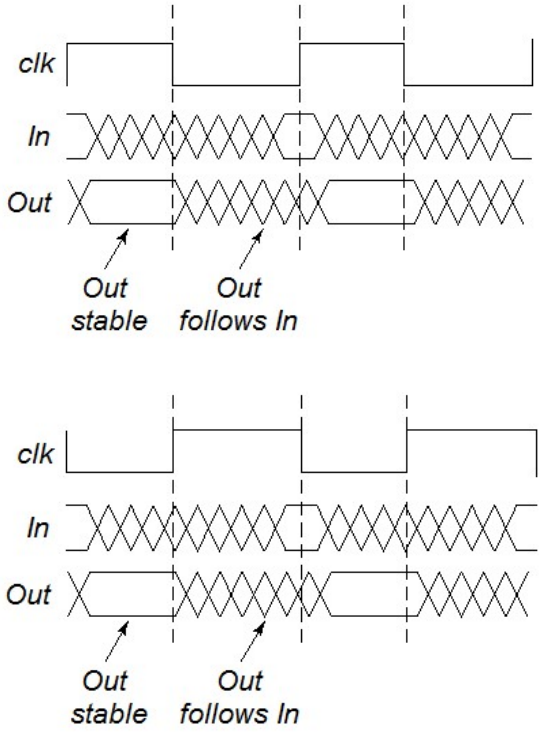


Clk	D	Q(t + 1)
0	x	Q(t)
1	0	0
1	1	1

基于锁存器的设计

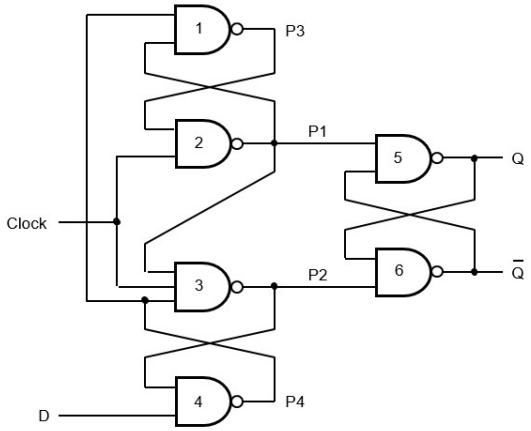
- **N latch is transparent when $\phi = 0$**
- **P latch is transparent when $\phi = 1$**

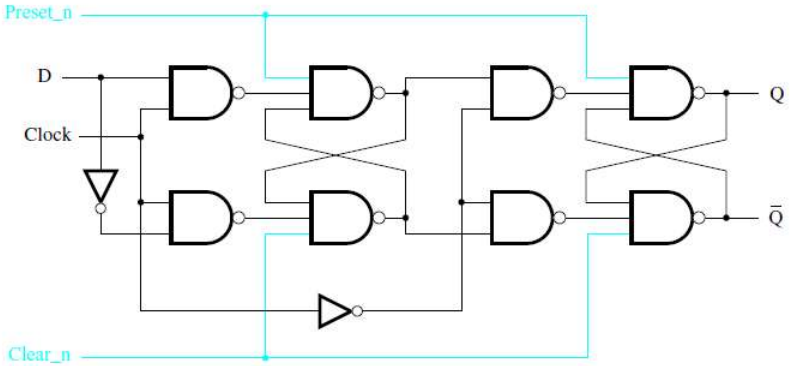


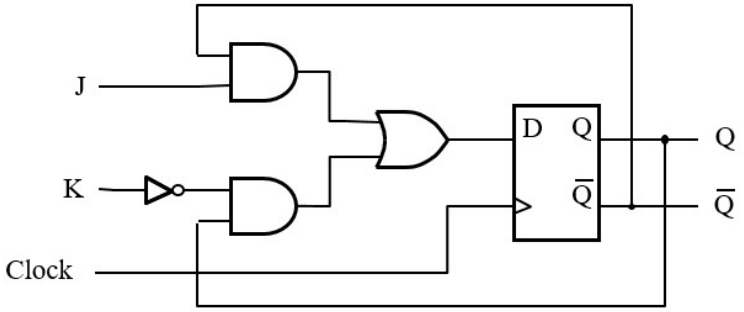


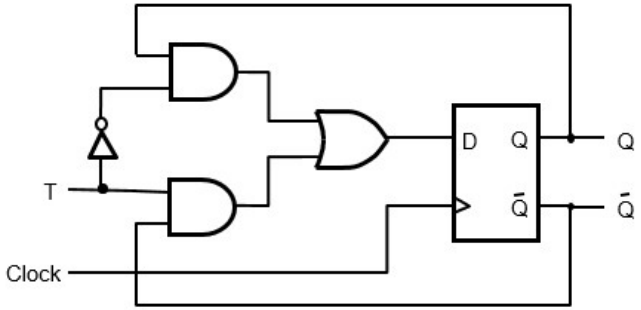
时序逻辑电路——触发器

- 触发器是边沿触发的存储单元，有D型，JK型，T型等



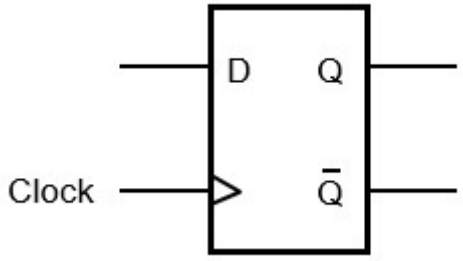


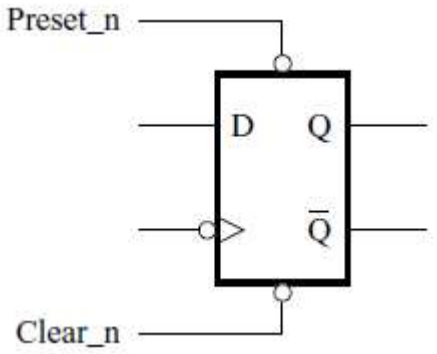


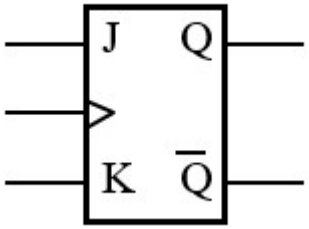


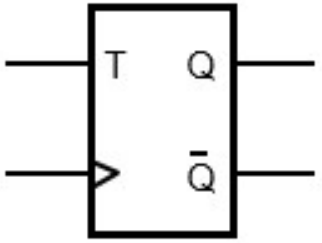
时序逻辑电路——触发器

- **D触发器**是最常用的触发器，几乎所有的逻辑电路都可以描述成D触发器与组合逻辑电路

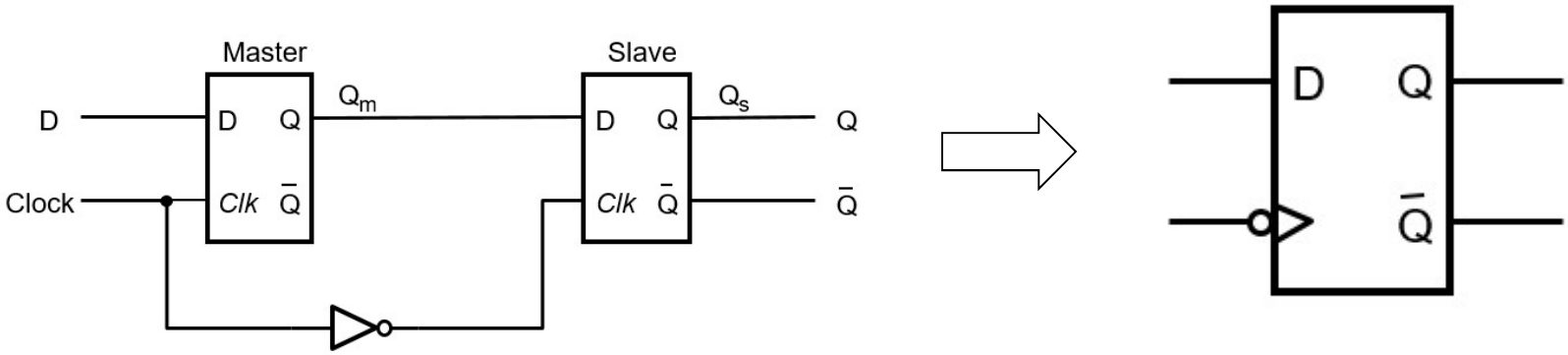




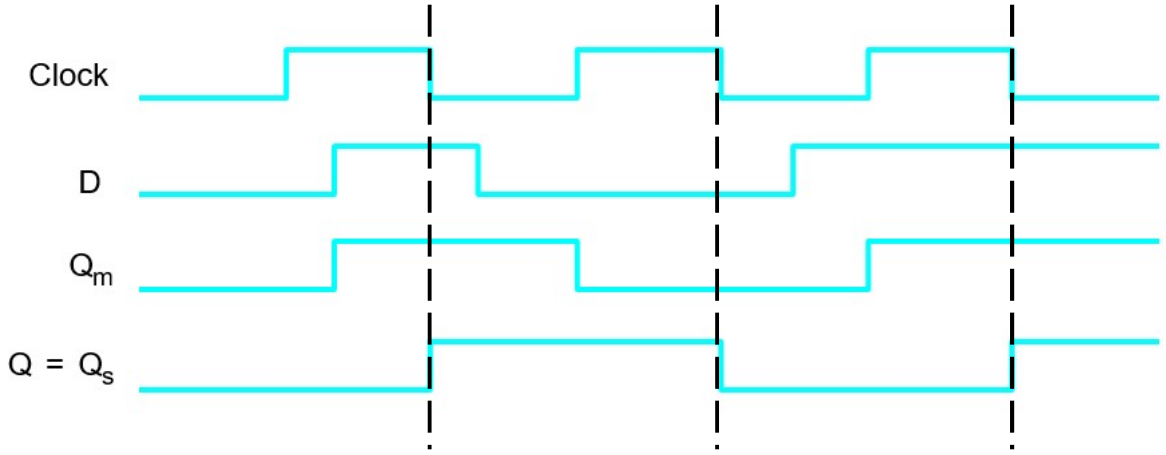




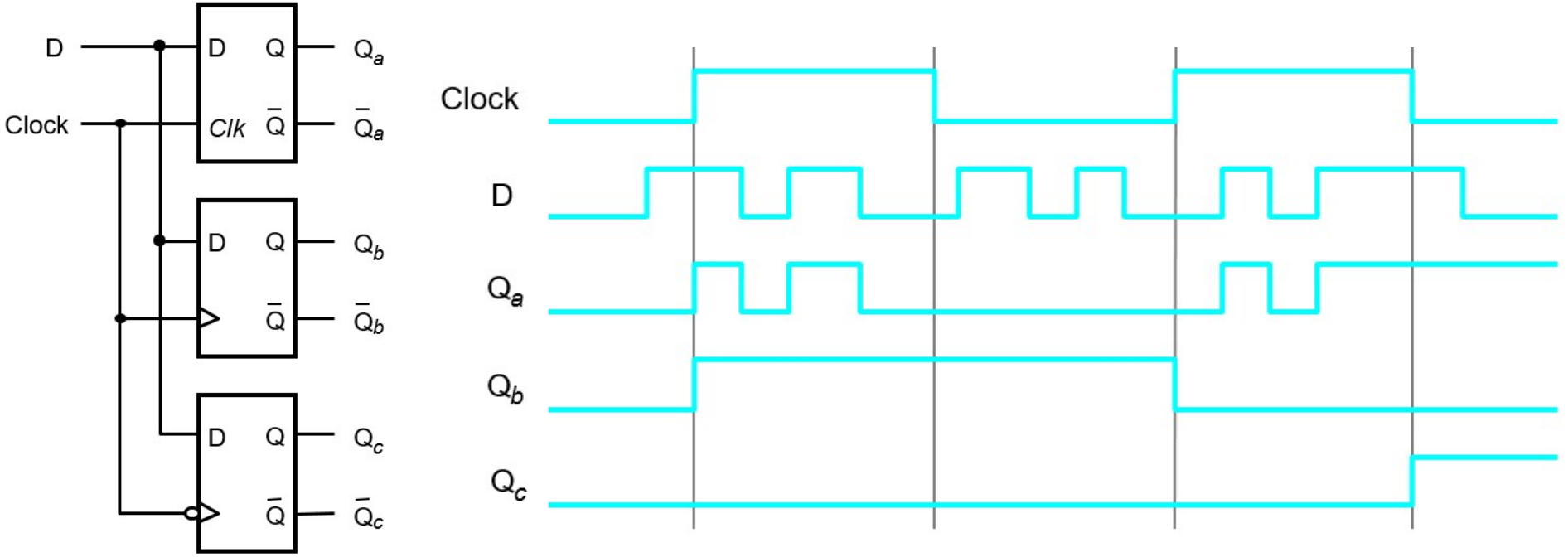
D触发器



- 在时钟沿
  - 把D送到Q
- 在其它时候
  - 保持不变

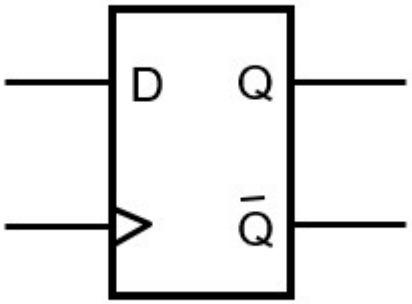


电平触发VS.边沿触发


D
Clock
D
Q
Clk
Q
Q_a
$\overline{Q}_a$
Q_b
$\overline{Q}_b$
Q_c
$\overline{Q}_c$
Clock
D
Q_a
Q_b
Q_c

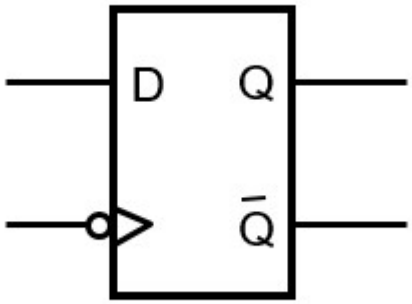
用Verilog描述存储器——D触发器

```
module flipflop (D, Clock, Q);
      input D, Clock;
      output reg Q;

      always @(posedge Clock)
            Q <= D;
endmodule
```



用Verilog描述存储器——D触发器

```
module flipflop (D, Clock, Q);
    input D, Clock;
    output reg Q;

    always @(negedge Clock)
        Q <= D;
endmodule
```



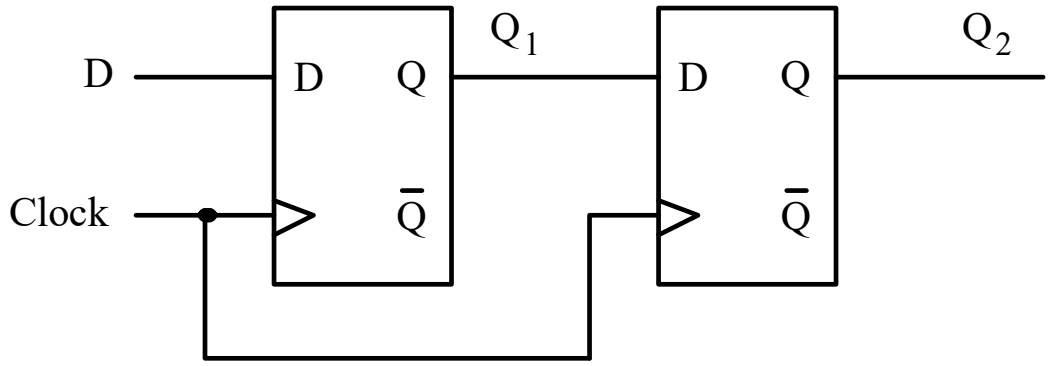
D锁存器 VS. D触发器

```
module D_latch (D, Clk, Q);
    input D, Clk;
    output reg Q;

    always @(D, Clk)
        if (Clk)
            Q = D;
endmodule
```

```
module flipflop (D, Clock, Q);
    input D, Clock;
    output reg Q;

    always @(posedge Clock)
        Q <= D;
endmodule
```

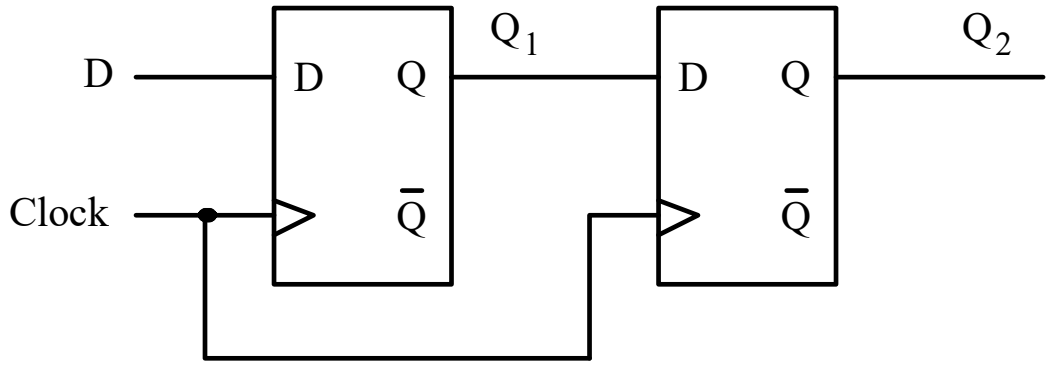
非阻塞赋值实现时序电路

```
module example5_4 (D, Clock, Q1, Q2);
    input D, Clock;
    output reg Q1, Q2;

    always @(posedge Clock)
    begin
        Q1 <= D;
        Q2 <= Q1;
    end

endmodule
```



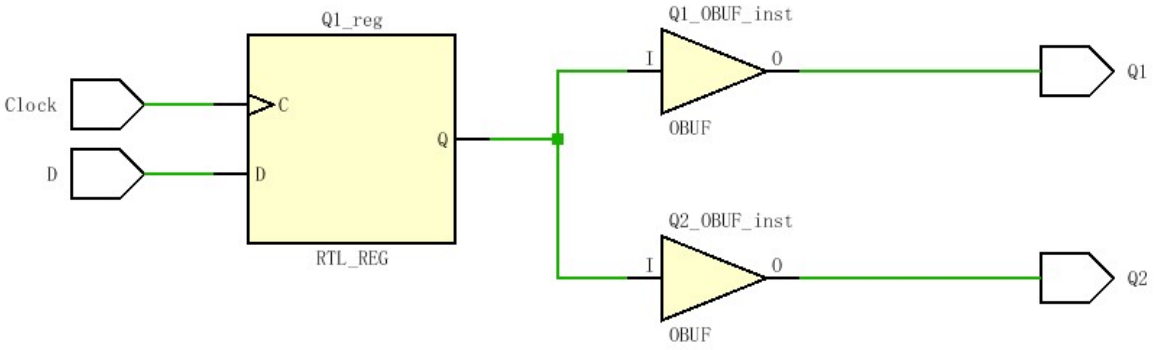
非阻塞赋值实现时序电路

```
module example5_4 (D, Clock, Q1, Q2);
    input D, Clock;
    output reg Q1, Q2;

    always @(posedge Clock)
    begin
        Q1 = D;
        Q2 = Q1;
    end

endmodule
```



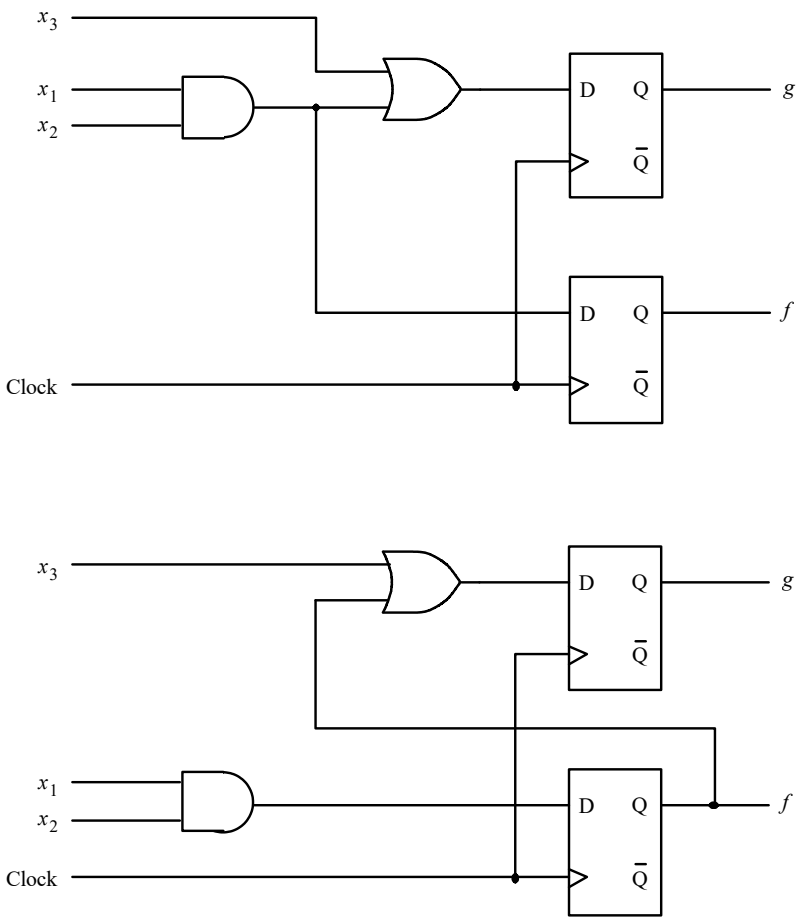
非阻塞赋值实现时序电路

```
module example5_4 (D, Clock, Q1, Q2);
    input D, Clock;
    output reg Q1, Q2;

    always @(posedge Clock)
    begin
        Q1 = D;
        Q2 = Q1;
    end

endmodule
```

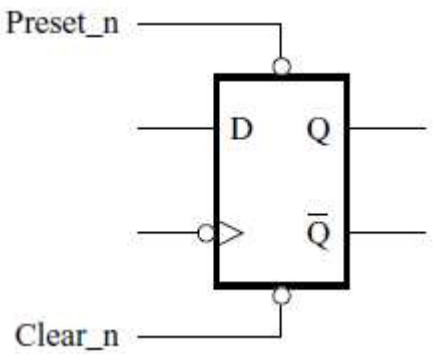


非阻塞赋值实现时序电路

```
module example5_5 (x1, x2, x3, Clock, f, g);
    input x1, x2, x3, Clock;
    output reg f, g;

    always @(posedge Clock)
    begin
        f <= x1 & x2;
        g <= f | x3;
    end

endmodule
```



- D触发器
  - 如果复位有效，输出复位
  - 在时钟沿把D送到Q
  - 在其它时候保持输出不变



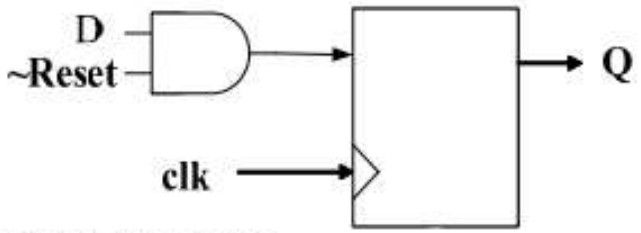
时序逻辑电路的复位操作

- **复位电路的重要性**
  - 仿真时使电路进入预知的初始状态
  - 使真实电路状态进入初始态，可保证电路从错误状态中恢复、可靠工作。
- **复位方式**
  - 同步复位
  - 异步复位

同步复位

- always的敏感表中只有时钟沿信号：

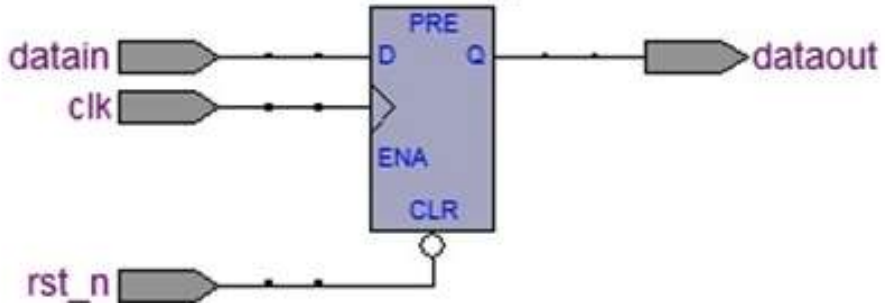
```
always@(posedge clk);
always@(negedge clk);
```



- 可以设计出100%的同步时序电路，有利于时序分析，其综合结果的频率往往较高；
- 有利于基于周期机制的仿真器进行仿真
- 可有效避免因复位电路毛刺造成的亚稳态和错误，增强了电路的稳定性。
- 但使用同步复位常会增加逻辑资源。
- 同步复位仅在时钟的有效沿生效，

复位信号长度大于时钟周期才能保证可靠复位。

异步复位



- 需要在always的事件敏感列表中加入复位信号的有效沿即可，只要复位信号有效沿到达，就可立即复位：

  如 always@(posedge clk or negedge reset);

- 设计简单，节约逻辑资源。

缺点

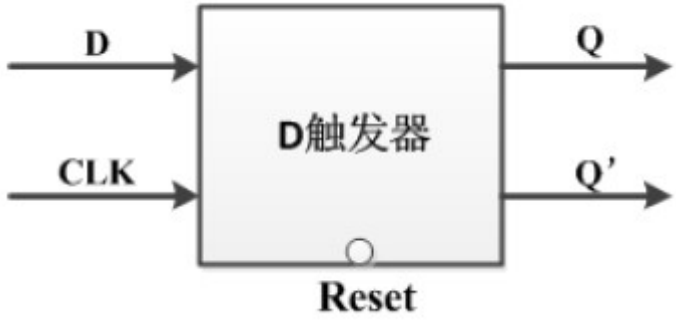
- 异步复位作用与时钟沿无关，如果异步复位释放时间与时钟有效沿到达时间一致，容易造成触发器输出亚稳态。
- 如果异步复位的逻辑树的组合逻辑产生了毛刺，则毛刺的有效沿会使触发器误复位。

带有异步复位的D触发器

```
module flipflop (D, Clock, Resetn, Q);
    input D, Clock, Resetn;
    output reg Q;

    always @(negedge Resetn or posedge Clock)
        if (!Resetn)
          Q <= 0;
        else
          Q <= D;

endmodule
```

n表示低电平有效

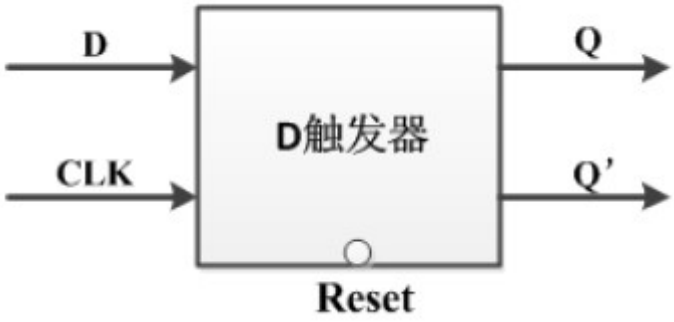
敏感表只有clk和reset



带有同步复位的D触发器

```
module flipflop (D, Clock, Resetn, Q);
    input D, Clock, Resetn;
    output reg Q;

    always @(posedge Clock)
        if (!Resetn)
            Q <= 0;
        else
            Q <= D;

endmodule
```

敏感表只有clk



可综合时序逻辑编码要点

- 在描述组合逻辑的always块中使用阻塞赋值"="，则往往综合成组合逻辑电路；
- 在描述时序逻辑的always块中使用非阻塞赋值"<="，则综合成时序逻辑电路；
- 尽量将组合逻辑时序逻辑分开描述
- 在同一always块中建立时序和组合逻辑电路时，用非阻塞赋值。

目录

寄存器

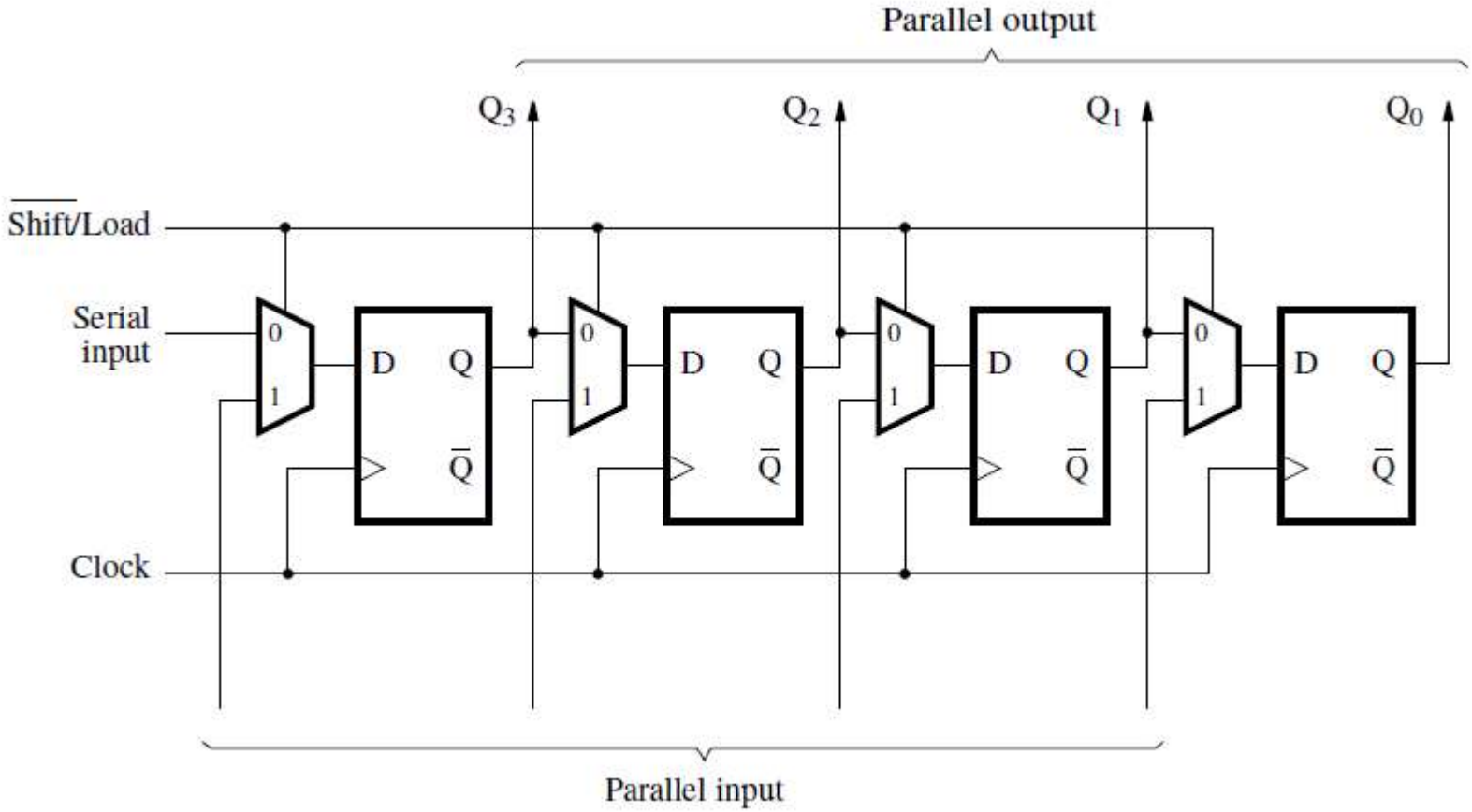
- 寄存器，是集成电路中非常重要的一种存储单元，通常由触发器组成。

- 寄存器也是中央处理器内的组成部分。寄存器是有限存贮容量的高速存贮部件，它们可用来暂存指令、数据和地址。

带有异步清零的n位寄存器

```
module regn (D, Clock, Resetn, Q);
    parameter n = 16;
    input [n-1:0] D;
    input Clock, Resetn;
    output reg [n-1:0] Q;

    always @(negedge Resetn or posedge Clock)
        if (!Resetn)
            Q <= 0;
        else
            Q <= D;
endmodule
```

并行存取移位寄存器


Parallel output
Q_3
Q_2
Q_1
Q_0
$\overline{\text{Shift/Load}}$
Serial input
Clock
0
1
D
Q
$\overline{Q}$
Parallel input

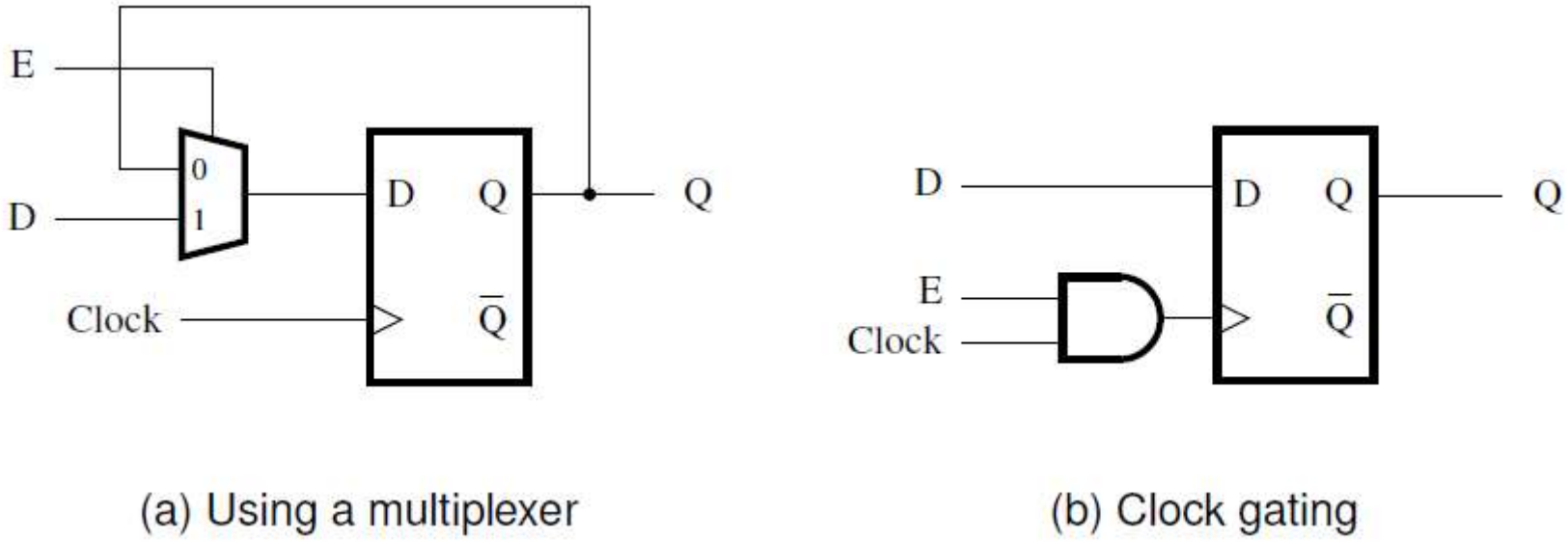
并行存取移位寄存器

```
module shift4 (R, L, w, Clock, Q);
    input [3:0] R;
    input L, w, Clock;
    output reg [3:0] Q;

    always @(posedge Clock)
        if (L)
            Q <= R;
        else  begin
            Q[0] <= Q[1];
            Q[1] <= Q[2];
            Q[2] <= Q[3];
            Q[3] <= w;
        end
endmodule
```

并行存取移位寄存器

```
module shiftn (R, L, w, Clock, Q);
    parameter n = 16;
    input [n-1:0] R;
    input L, w, Clock;
    output reg [n-1:0] Q;
    integer k;

    always @(posedge Clock)
        if (L)
          Q <= R;
        else
        begin
          Q  <=  Q>>1;
          Q[n-1] <= w;
        end
endmodule
```

使能



(a) Using a multiplexer

(b) Clock gating

Figure 5.56. Providing an enable input for a D flip-flop.

带使能输入的D触发器

```
module rege (D, Clock, Resetn, E, Q);
    input D, Clock, Resetn, E;
    output reg Q;

    always @(posedge Clock or negedge Resetn)
        if (Resetn == 0)
          Q <= 0;
        else if (E)
          Q <= D;

endmodule
```

带使能输入的寄存器

```
module regne (R, Clock, Resetn, E, Q);
    parameter n = 8;
    input [n-1:0] R;
    input Clock, Resetn, E;
    output reg [n-1:0] Q;

    always @(posedge Clock or  negedge Resetn)
        if (Resetn == 0)
            Q <= 0;
        else if (E)
            Q <= R;
endmodule
```

使能输入的移位寄存器

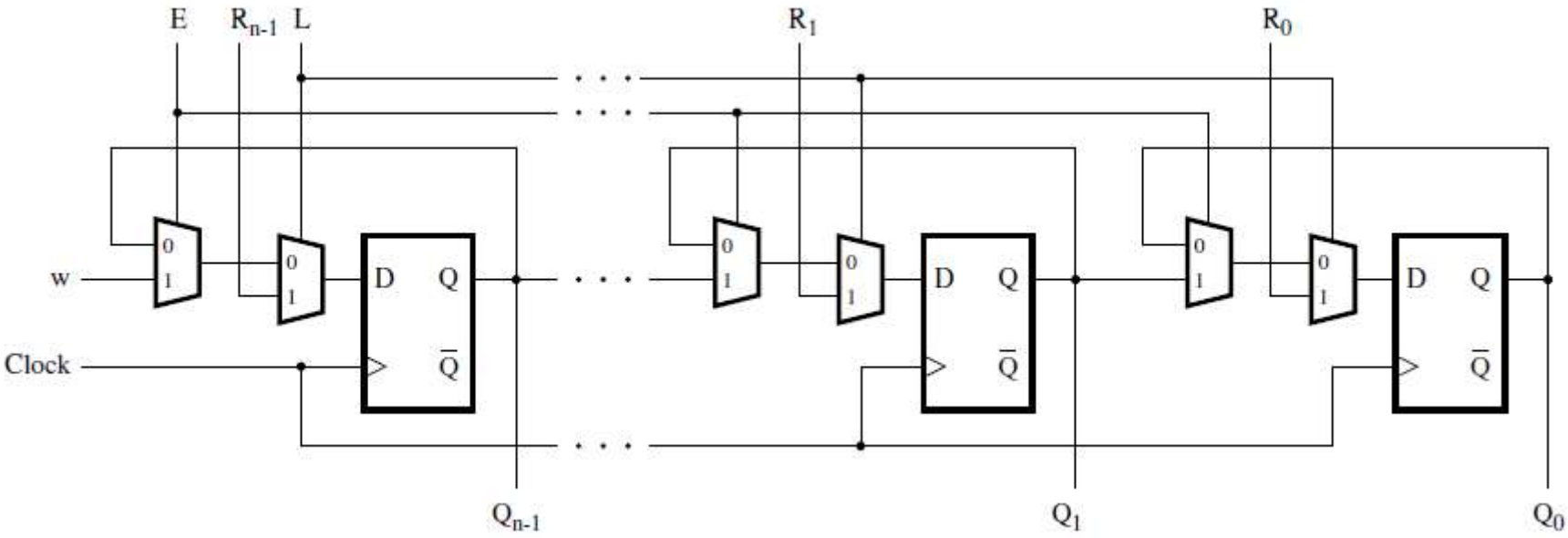


Figure 5.59. A shift register with parallel load and enable control inputs.

```
module shiftrne (R, L, E, w, Clock, Q);
    parameter n = 4;
    input [n-1:0] R;
    input L, E, w, Clock;
    output reg [n-1:0] Q;
    integer k;
    always @(posedge Clock)
    begin
        if (L)
          Q <= R;
        else if (E)
        begin
          Q[n-1] <= w;
          Q[n-2:0] <= Q[n-1:1];
        end
    end
endmodule
```