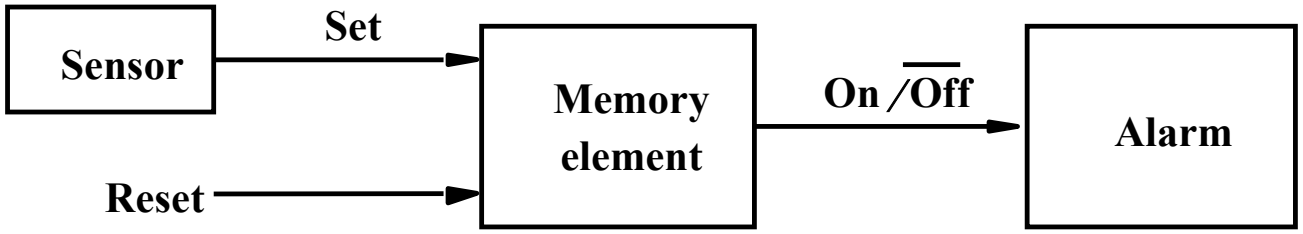
存储元件需求


Sensor
Set
Reset
Memory element
On /Off
Alarm


红外报警(防盗贼)
红外探测器探测到人体温度（36~37度左右）及移动物体，同时满足以上两个条件后即会发出报警信号给网络摄像机进行拍照录像，摄像机会发送报警信息到手机及其它终端。

触发器、寄存器和计数器

计算机系　齐悦

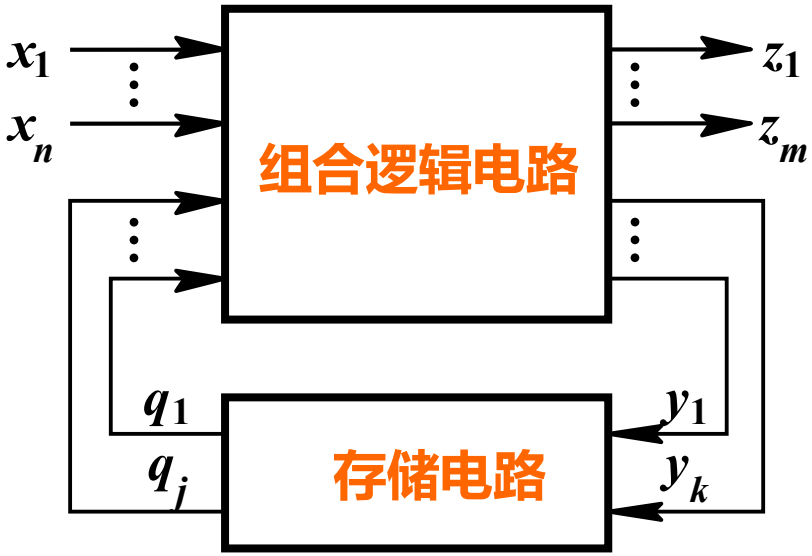
北京科技大学

目录

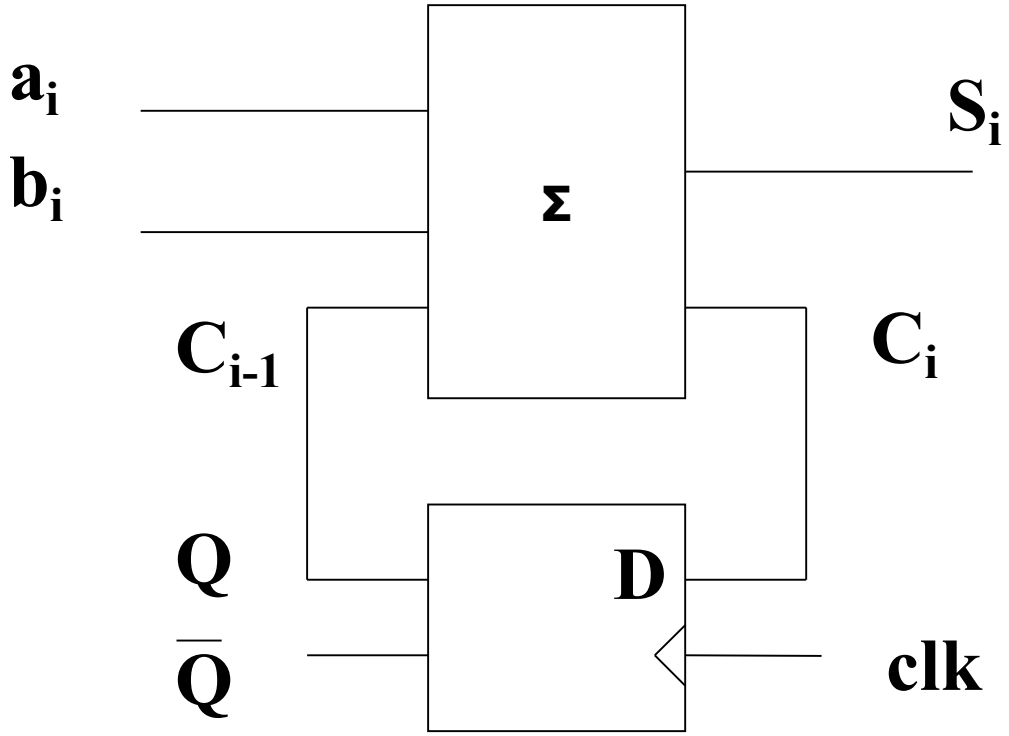
时序逻辑电路——有记忆功能

- 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。

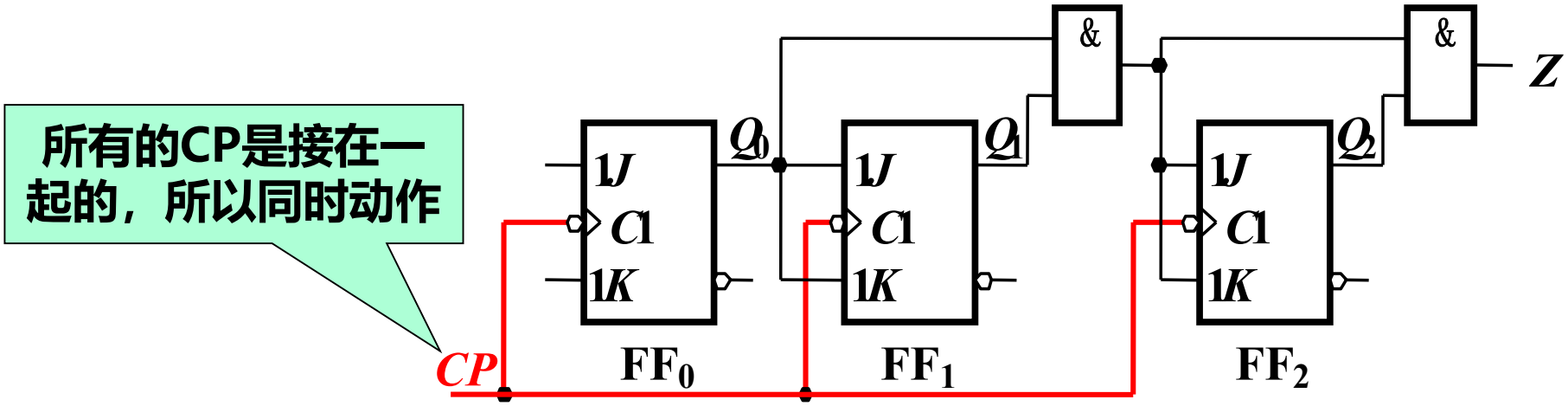
异步时序逻辑电路

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。

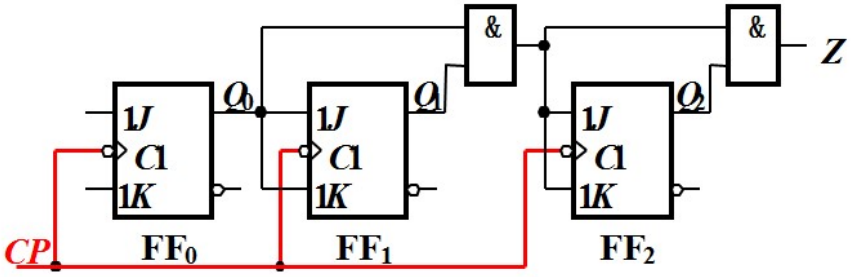
同步时序逻辑电路

- 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

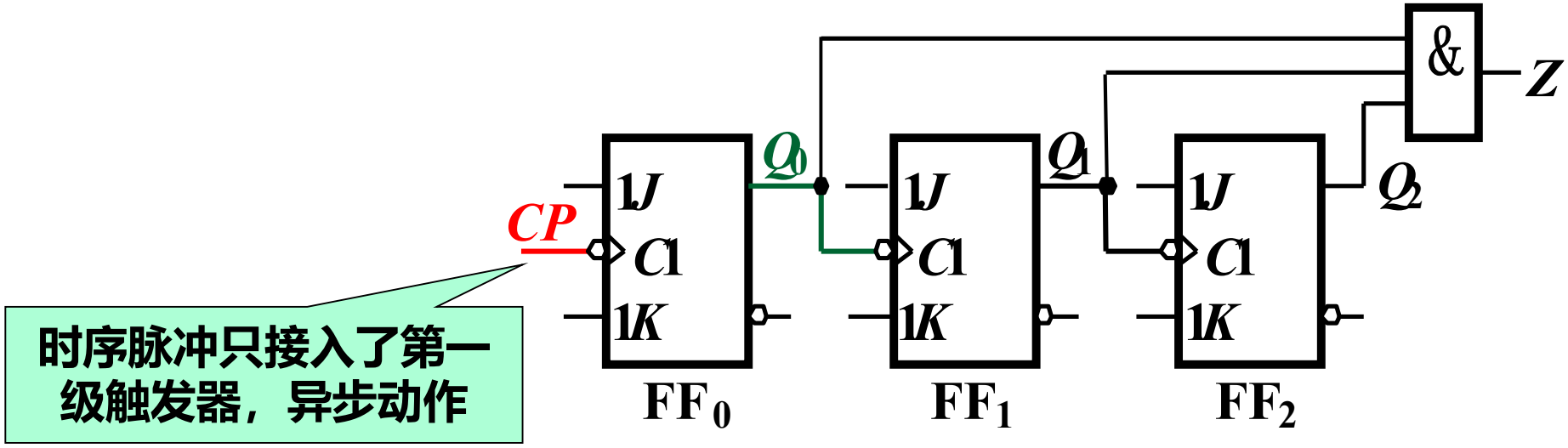


- 优点：
  - 简单。每个电路里的运算必须要在时钟的两个脉冲之间固定的间隔内完成，称为一个时钟周期。满足该条件下的电路是可靠的。
- 缺点：
  - 时钟是高频率信号，而时钟必须分布到各个触发器而不管触发器是否要工作，消耗大量功耗
  - 最快的时钟频率是由电路中最慢的逻辑路径（关键路径）决定的，一定程度限制工作的最高频率（流水线优化）

异步时序逻辑电路

- 没有统一的时钟源，可以有多个时钟源，每个触发器不是同时被触发的，有时间先后。

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



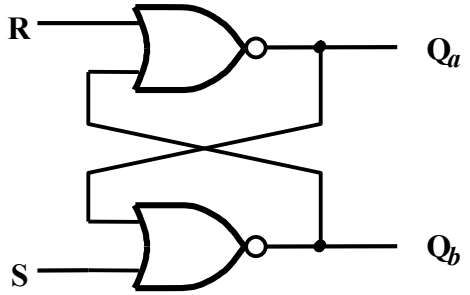
同步 VS. 异步

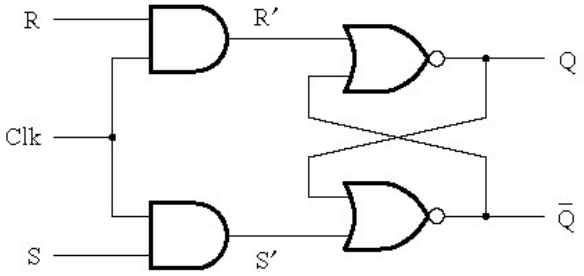
- 同步时序电路
  - 所有存储单元状态变化都由同一时钟信号控制，比较容易满足**建立时间**和**保持时间**的要求。
  - 同步时序电路可以很好地避免毛刺
  - 有利于器件移植
  - 有利于静态时序分析（STA）和验证
- 异步时序电路
  - 不存在全局时钟，各触发器翻转的时间不定，设计复杂性增加
  - 电路的核心由组合逻辑实现，比如异步FIFO/RAM的读写信号
  - 最大的问题是容易**产生毛刺**，影响电路可靠性、稳定性

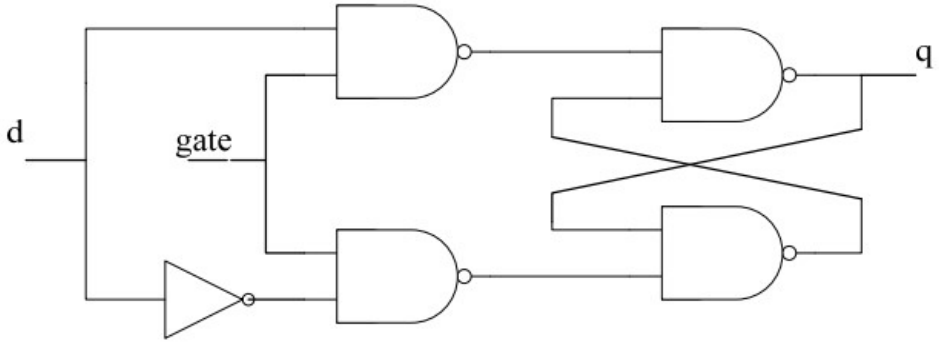
目录

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。

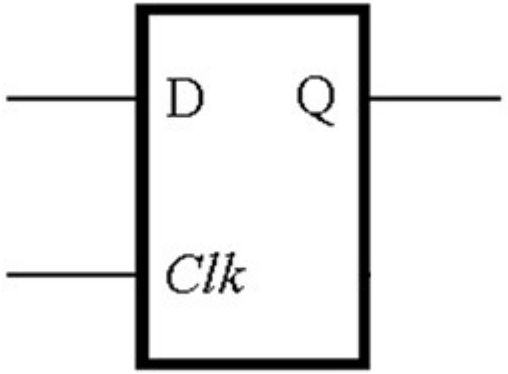






用Verilog描述存储器——D锁存器

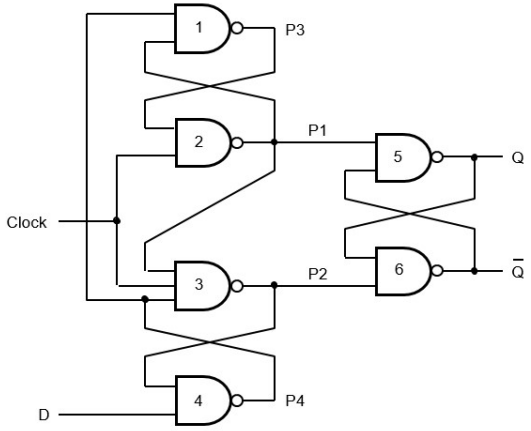
```
module D_latch (D, Clk, Q);
      input D, Clk;
      output reg Q;

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

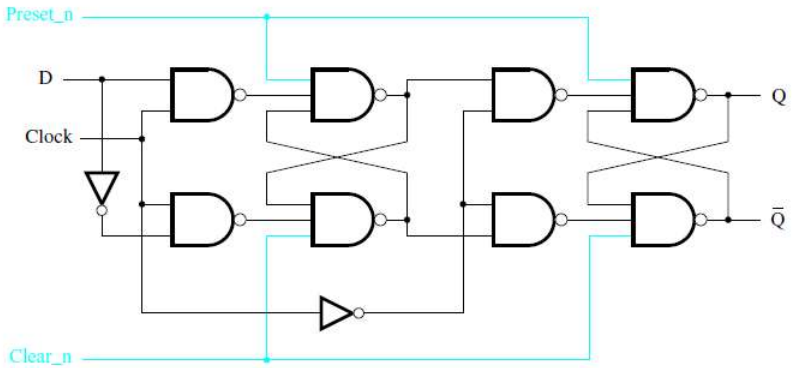


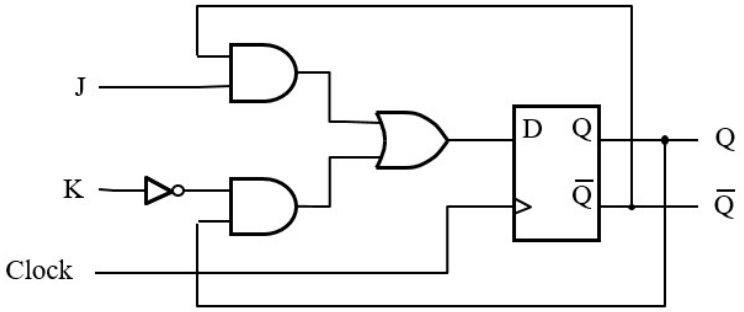
Clk	D	Q(t + 1)
0	x	Q(t)
1	0	0
1	1	1

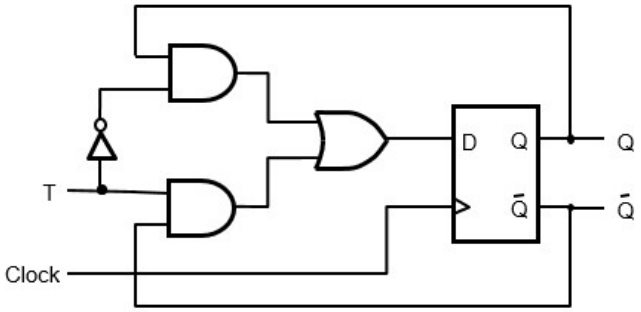
时序逻辑电路——触发器

- 触发器是边沿触发的存储单元，有D型，JK型，T型等



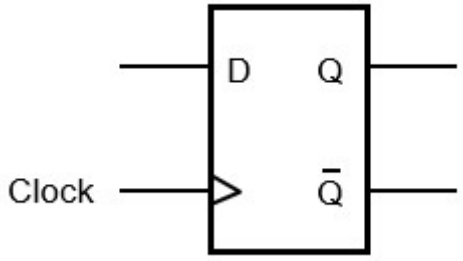


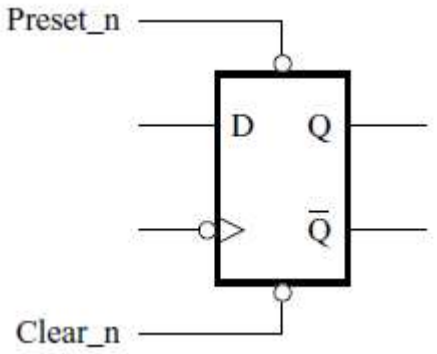


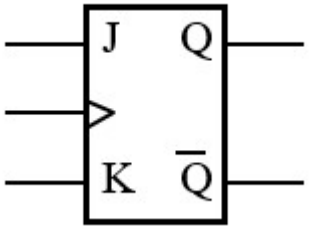


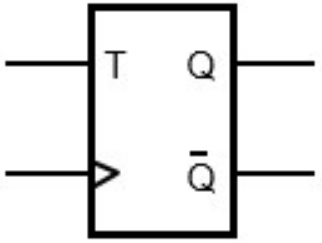
时序逻辑电路——触发器

- **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

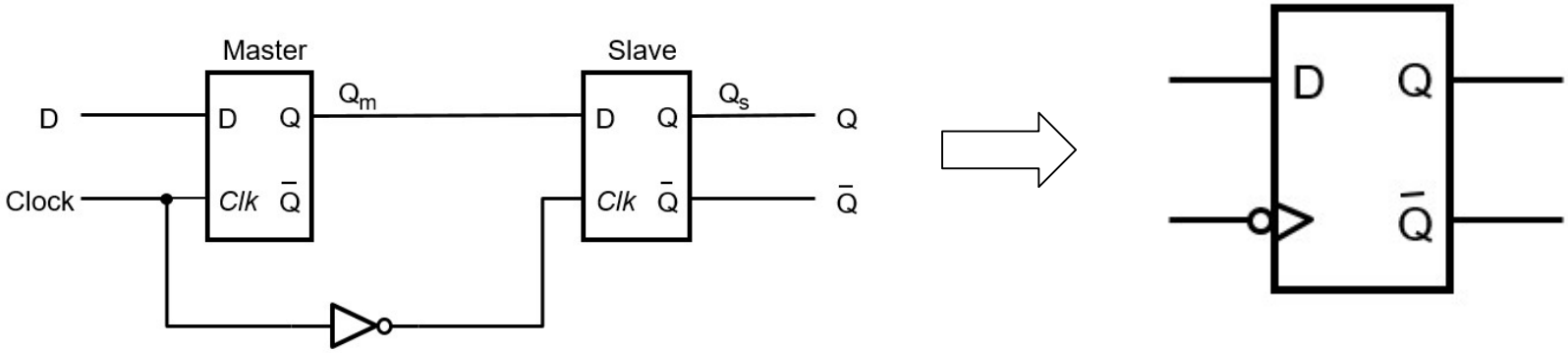




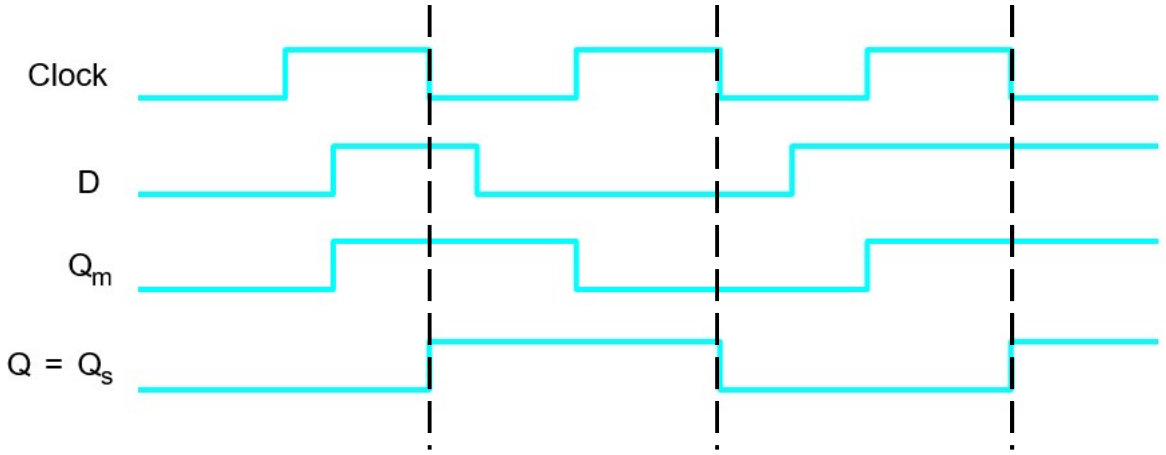




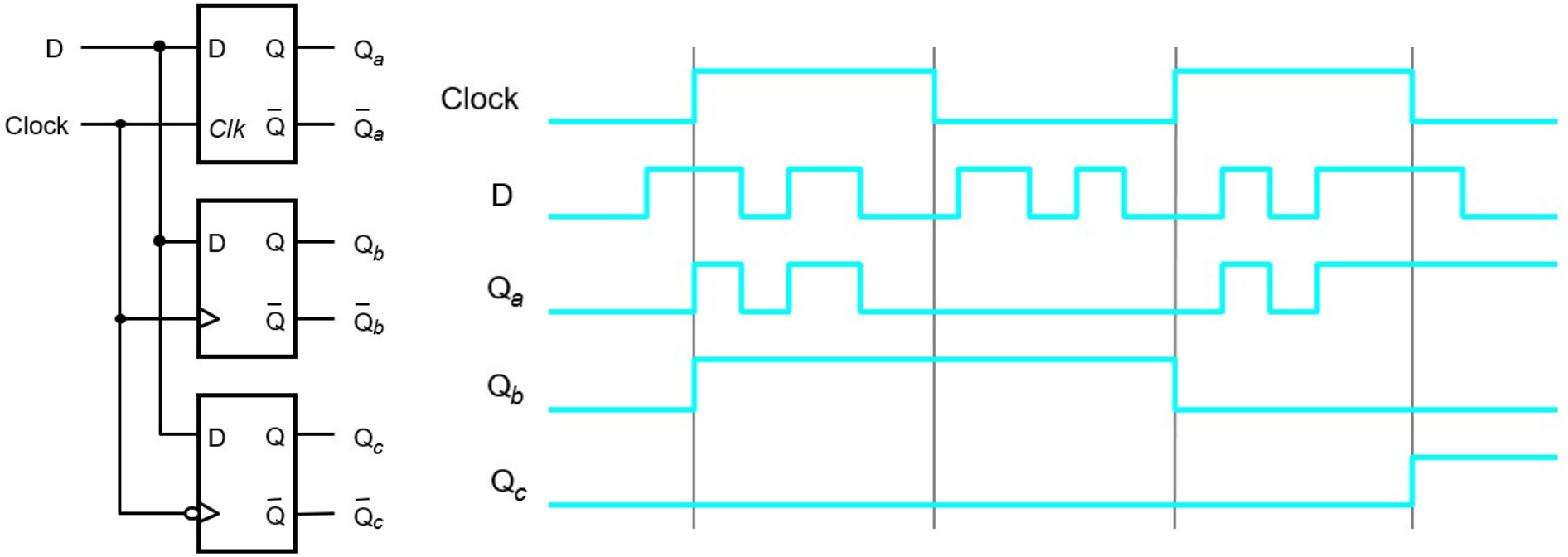
D触发器



- 在时钟沿
  - 把D送到Q
- 在其它时候
  - 保持不变

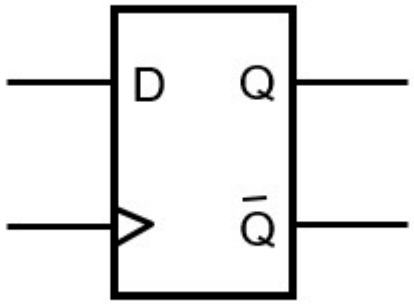


电平触发VS.边沿触发


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

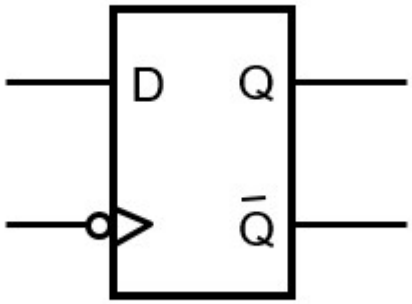
```
module flipflop (D, Clock, Q);
      input D, Clock;
      output reg Q;

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



- **边沿触发**
  - posedge 上升沿触发
  - negedge 下降沿触发
  - 通常与时钟信号搭配使用

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
```

always语句

- **always过程块**
  - 阻塞赋值(=)
  - 非阻塞赋值(<=)
  - **if**语句
  - **case**语句

由输入信号中任意一个电平发生变化所引起

```
always @(a or b or c or …)
begin
    语句块(=, if, case)
end
```

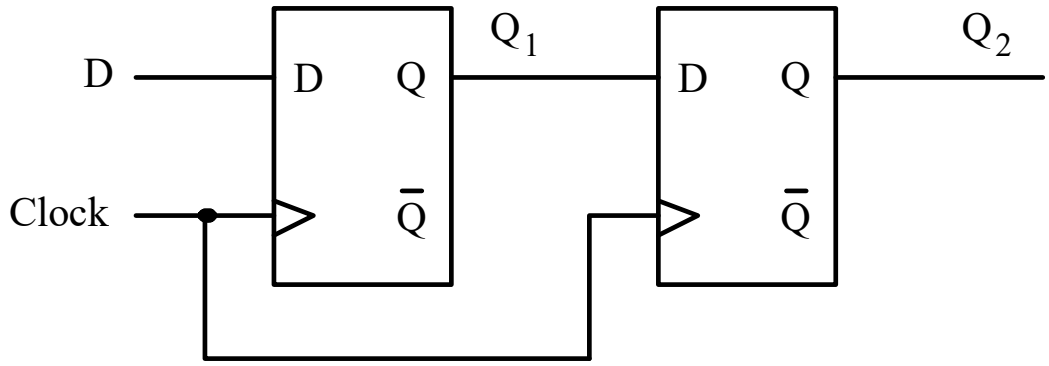
```
always @(posedge/negedge sig or…)
begin
    语句块(<=, if, case)
end
```

由单个跳变沿所引起

- 两个或更多**always**模块是同时执行的
- **always** 模块描述组合逻辑电路时，用阻塞赋值语句
- always 模块描述时序逻辑电路时，用非阻塞赋值语句

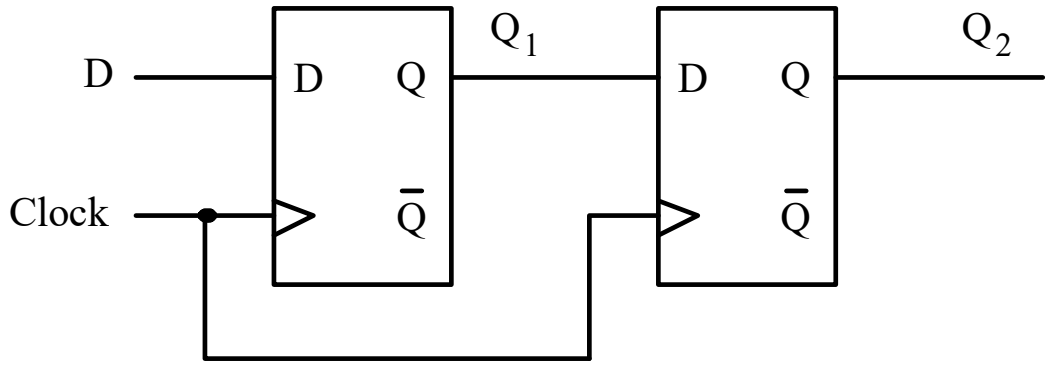
非阻塞赋值实现时序电路

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

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

endmodule
```



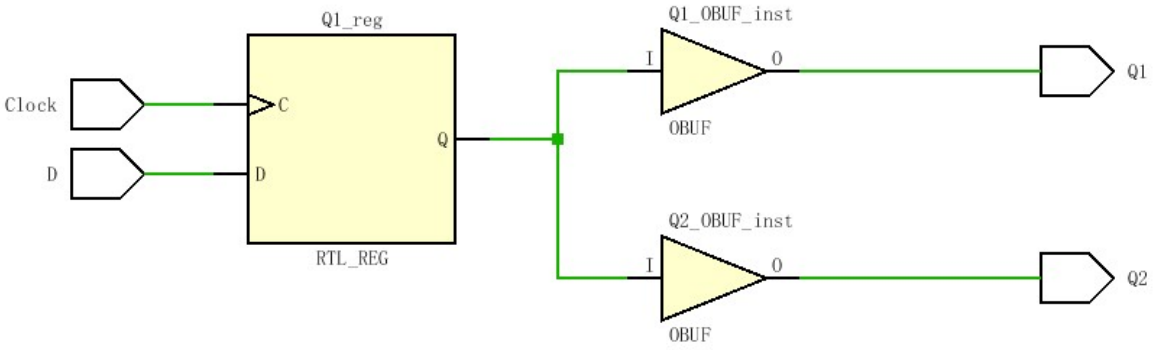
非阻塞赋值实现时序电路

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

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

endmodule
```



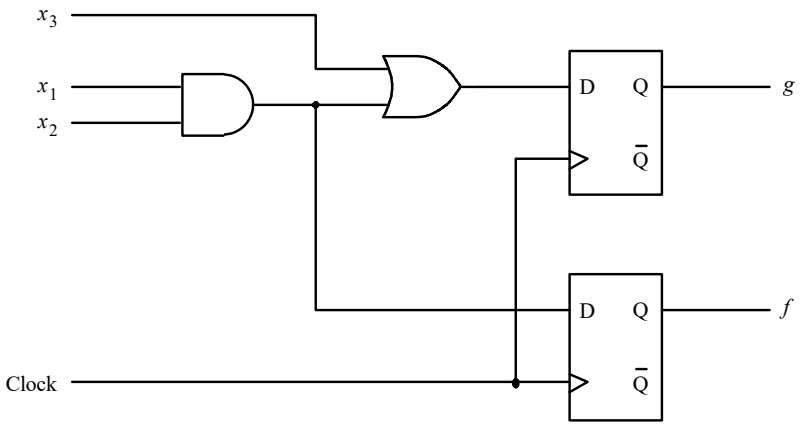
非阻塞赋值实现时序电路

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

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

endmodule
```



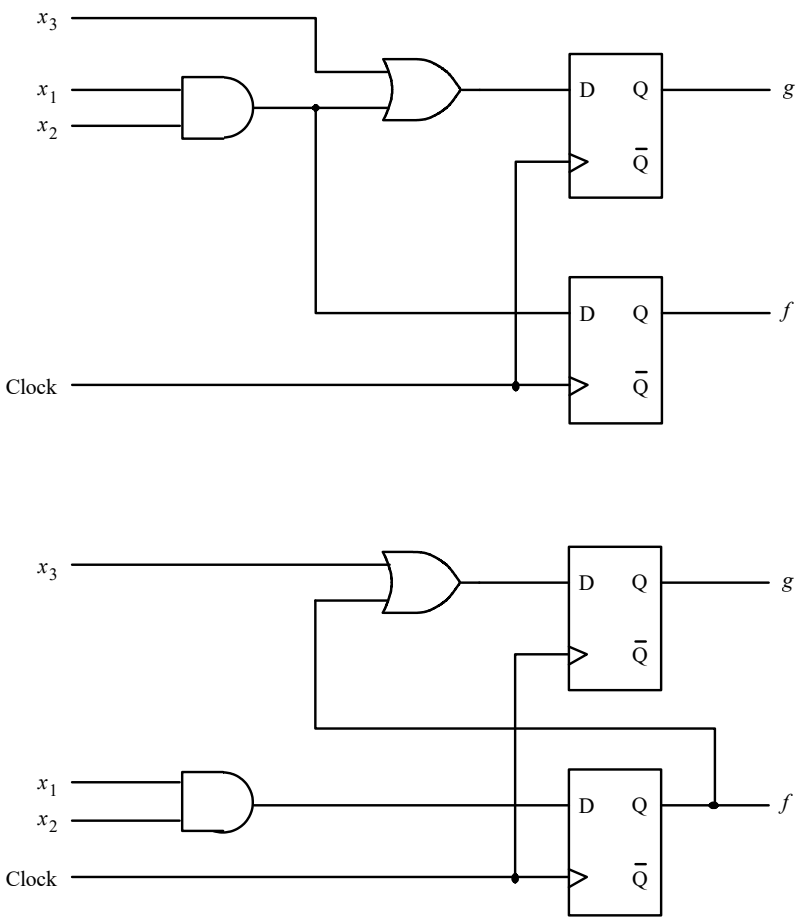
非阻塞赋值实现时序电路

```
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
```



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
```



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

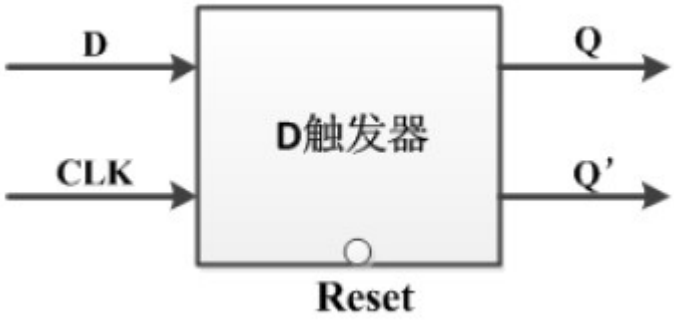
同步复位

- 同步复位仅在时钟有效沿生效
- 有利于时序分析，综合结果的频率通常较高
- 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亚稳态，增强电路稳定性
- 同步复位可能会增加逻辑资源
- 复位信号长度大于时钟周期才能保证可靠复位

带有同步复位的D触发器

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

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

endmodule
```

敏感表只有clk



异步复位

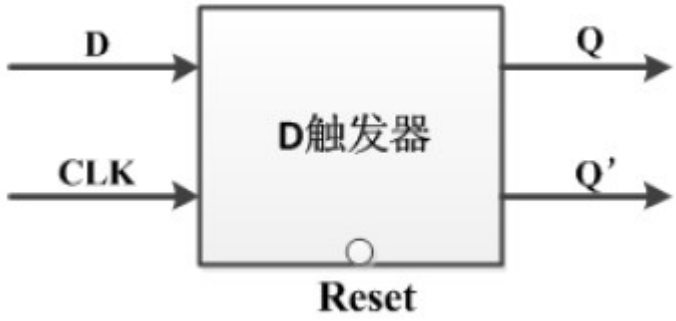
- 异步复位：只要复位信号有效沿到达，立即复位
- 设计简单，可节约逻辑资源
- 缺点
  - 与时钟沿无关，如果异步复位释放时间与时钟有效沿同时到达，可能造成触发器输出亚稳态
  - 如果异步复位的逻辑树的组合逻辑产生了毛刺，则毛刺的有效沿会使触发器误复位

带有异步复位的D触发器

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

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

endmodule
```

n表示低电平有效

敏感表有CLK和Reset



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
- 将组合逻辑和时序逻辑分开描述

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。

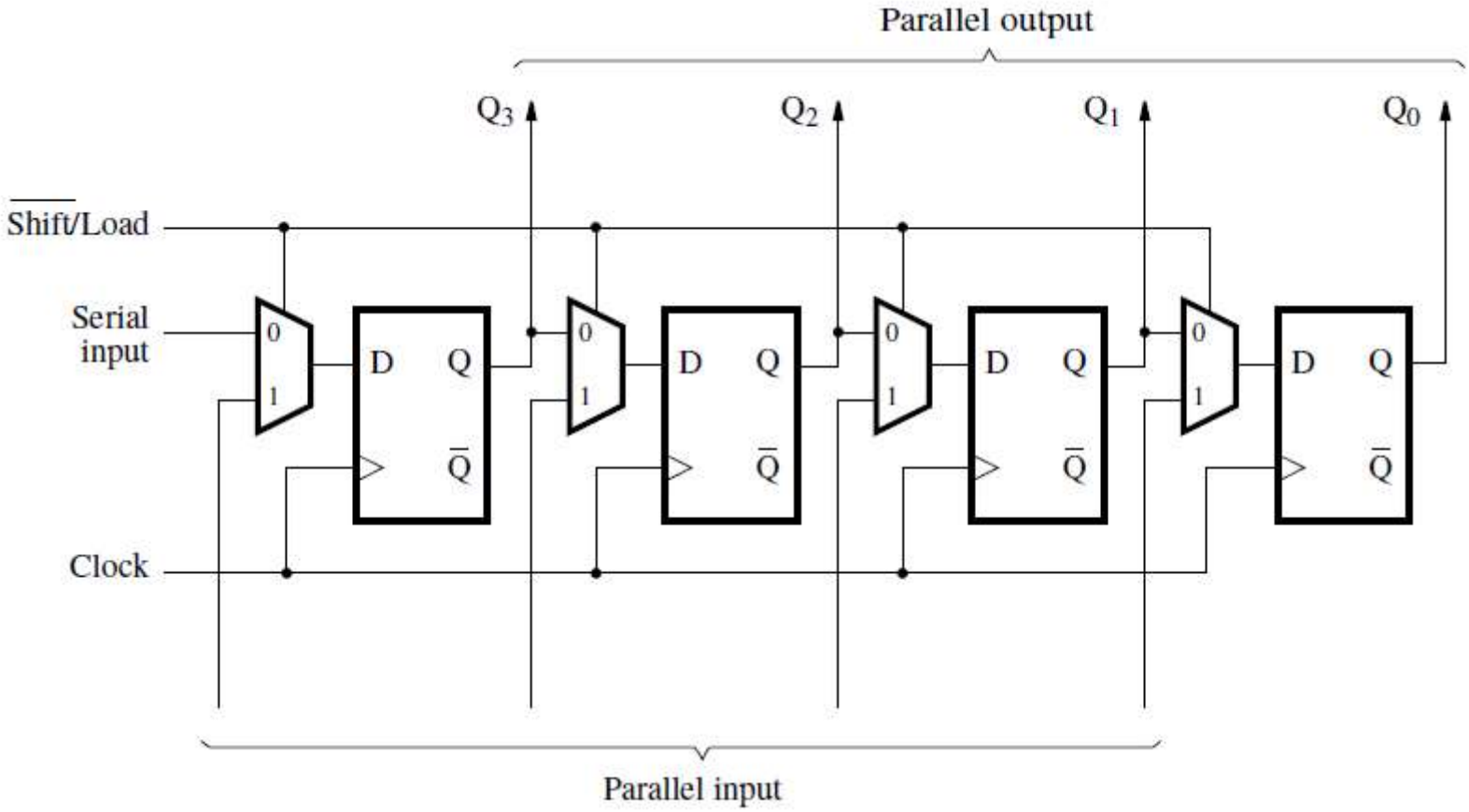
- 寄存器也是中央处理器内的组成部分。寄存器是有限存储容量的高速存储部件，它们可用来暂存指令、数据和地址。

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
```

- **parameter**
  - 用参数声明一个常量，常用于定义延时及宽度变量
  - 语法：
  - parameter 参数名1=表达式1，参数名2=表达式2……;
  - 可一次定义多个参数，用逗号隔开
  - 参数的定义是局部的，只在当前模块中有效
  - 表达式必须是常数表达式

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

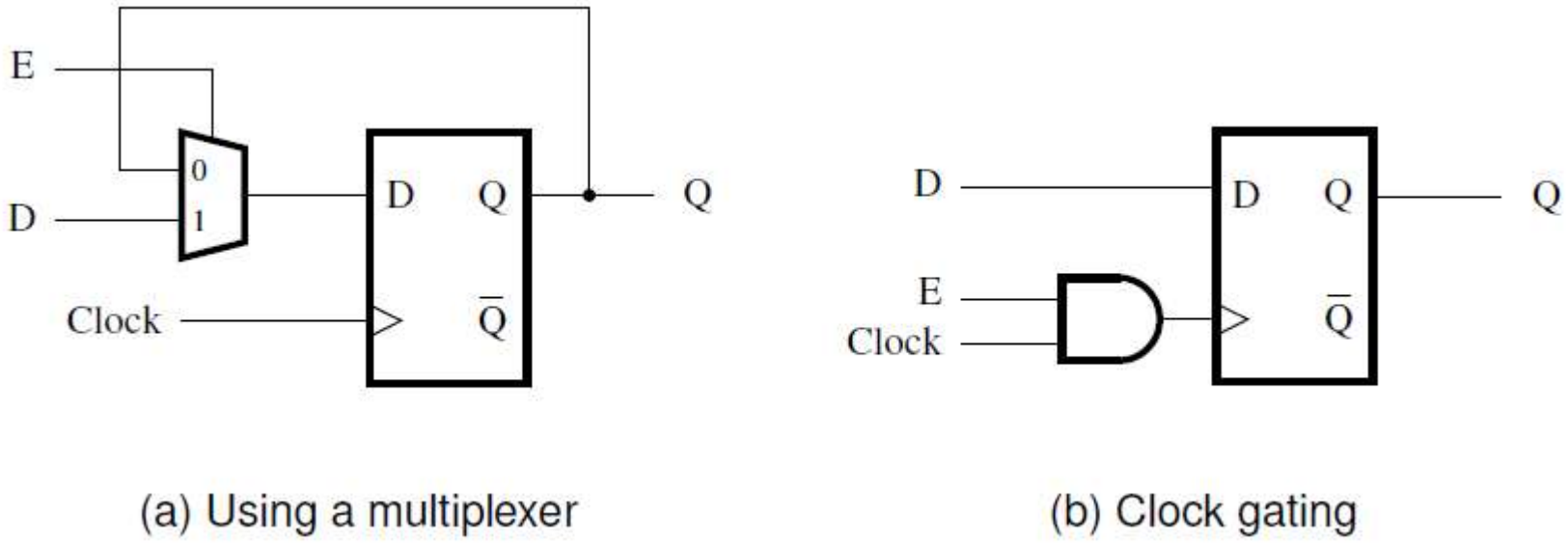
并行存取移位寄存器

```
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
```

使能控制



(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

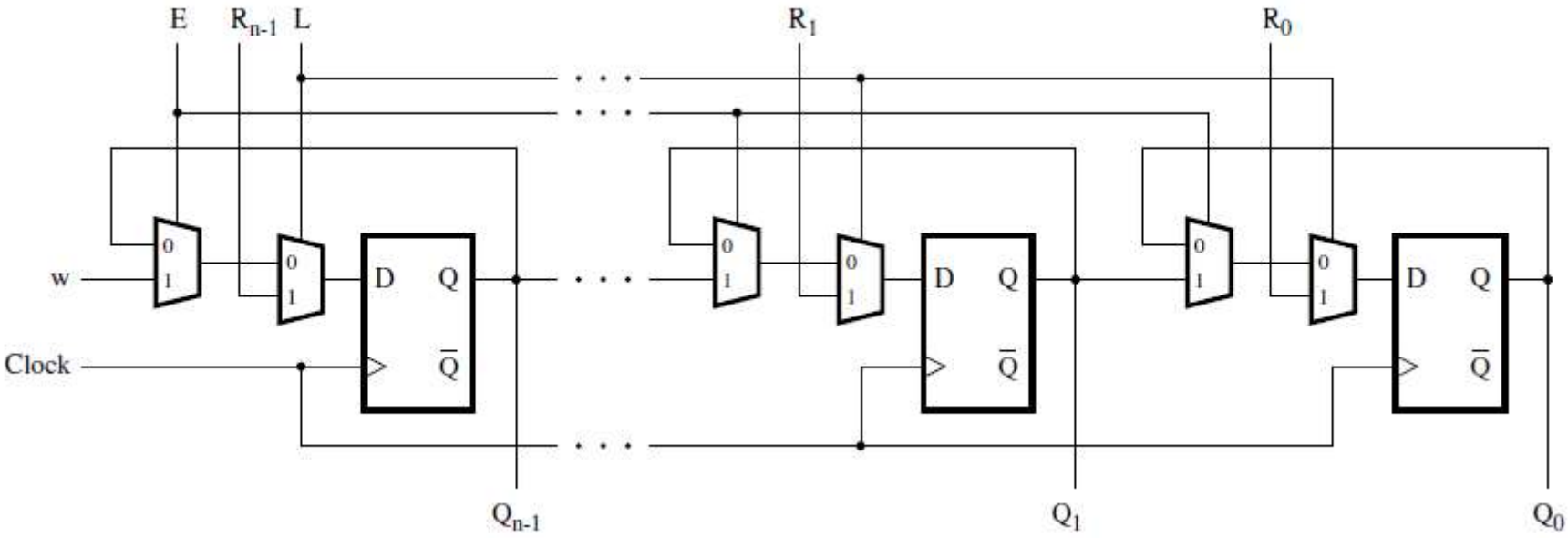


Figure 5.59. A shift register with parallel load and enable control inputs.

```
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
```

计数器

- 功能
  - 计数、定时
  - 时钟分频
  - 时序控制信号
- 分类
  - 同步计数器
  - 异步计数器

同步计数器的实现方法

- 最简单、直观的通用方式：
  - 算术操作 +、-
  - 最终硬件由加法计数器或减法计数器、寄存器实现
- 使用LFSR
  - 同样的功能，使用LFSR能得到速度快、面积小的硬件实现。

具有异步复位的同步递增计数器

- 分析需求
  - 异步复位
  - 计数使能
  - 0-15
- 确定接口信号
- 画出电路结构图
- 描述电路

```
module counter (Resetn, Clock, E, Q);
    input Resetn, Clock, E;
    output reg [3:0] Q;

    always @(negedge Resetn or posedge Clock)
        if (!Resetn)
            Q <= 0;
        else if (E)
            Q <= Q + 1;

endmodule
```

具有并行载入的同步递增计数器

- 分析需求
  - 异步复位
  - 计数使能
  - 0-15
  - 并行载入

```
module counter (R, Resetn, Clock, E, L, Q);
    input [3:0] R;
    input Resetn, Clock, E, L;
    output reg [3:0] Q;

    always @(negedge Resetn or posedge Clock)
        if (!Resetn)
            Q <= 0;
        else if (L)
            Q <= R;
        else if (E)
            Q <= Q + 1;
endmodule
```

具有并行载入的同步递增计数器+

- 分析需求
  - 异步复位
  - 计数使能
  - 0-15
  - 并行载入
  - 计满标志

```
module counter (R, Resetn, Clock, E, L, Q, Rc);
    input [3:0] R;
    input Resetn, Clock, E, L;
    output reg [3:0] Q;
    output Rc;
    always @(negedge Resetn or posedge Clock)
        if (!Resetn)            Q <=4'b0;
        else if (L)             Q <= R;
        else if (E)             Q <= Q + 1;
    always @(Q)
        if (Q==4'd15)           Rc=1;
        else                    Rc=0;
endmodule
```

具有并行载入的同步递增计数器++

- 分析需求
  - 异步复位
  - 计数使能
  - 0-15
  - 并行载入
  - 计满标志
  - 片选

```
module counter (R, Resetn, Clock, CE, E, L, Q, Rc);
    input [3:0] R;
    input Resetn, Clock, CE, E, L;
    output reg [3:0] Q;
    output Rc;
    always @(negedge Resetn or posedge Clock)
        if (!Resetn)                Q <=4'b0;
        else if (CE & L)            Q <= R;
        else if (CE & E)            Q <= Q + 1;
    always @(*)
        if (CE && (Q==4'd15))       Rc=1;
        else                        Rc=0;
endmodule
```

具有并行载入的同步递增计数器+++

- 分析需求
  - 异步复位
  - 计数使能
  - 0-15
  - 并行载入
  - 计满标志
  - 片选
  - 十进制计数

```
module counter (R, Resetn, Clock, CE, E, L, Q, Rc);
    input [3:0] R;
    input Resetn, Clock, CE, E, L;
    output reg [3:0] Q;
    output Rc;
    always @(negedge Resetn or posedge Clock)
        if (!Resetn)                    Q <=4'b0;
        else if (L)                     Q <= R;
        else if (CE && E && (Q==4'd9))  Q <= 4'd0;
        else if (CE & E)                Q <= Q + 1;
    always @(*)
        if (CE && (Q==4'd9))            Rc= 1;
        else                            Rc=0;
endmodule
```

具有并行载入的同步递增计数器++

- 分析需求
  - 异步复位
  - 计数使能
  - 0-15
  - 并行载入
  - 计满标志
  - 递增递减

```
1:  module counter (R, Resetn, Clock, E, UP, L, Q, Rc);
2:      input [3:0] R;
3:      input Resetn, Clock, E, L, UP;
4:      output reg [3:0] Q;
5:      output Rc;
6:      always @(negedge Resetn or posedge Clock)
7:          if (!Resetn)                    Q <=4'b0;
8:          else if (L)                     Q <= R;
9:          else if (E && UP)               Q<=Q+1;
10:         else if (E && !UP)              Q<=Q-1;
11:     always @(*)
12:         if (UP && (Q==4'd15)) || (!UP && (Q==4'd0)) Rc=1;
13:         else                            Rc=0;
14: endmodule
```

时序电路描述方法

- 定义存储单元
  - 通过定义**reg**变量**+**过程赋值（**always**）为存储单元建模
- 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

时钟分频器

- 偶数分频
  - 只需实现一个时钟同步计数器，然后在相应的bit位抽头即可
- 奇数分频
  - 需使用简单的状态机，设计合适的或符合要求的分频时钟的占空比

余3十进制计数器

```
module upcount (R, Resetn, Clock, E, L, Q, Rc);
    input [3:0] R;
    input Resetn, Clock, CE, E, L;
    output reg [3:0] Q;
    output Rc;
    always @(negedge Resetn or posedge Clock)
        if (!Resetn)                    Q <=4'd3;
        else if (L)                     Q <= R;
        else if (E && (Q==4'd12))       Q <= 4'd3;
        else if (E)                     Q <= Q + 1;
    always @(*)
        if (Q==4'd12)                   Rc=1;
        else                            Rc=0;
endmodule
```

目录

设计实例——反应计时器

- 一种测量人对特定事件做出反映时间的电路
- 例如：
  - 测试人员点亮LED灯
  - 被测人员看到LED灯被点亮后，尽可能快地按下开关
  - 该电路测量从LED被点亮时刻到开关被按下时刻的时间长度（单位：1/100 秒）

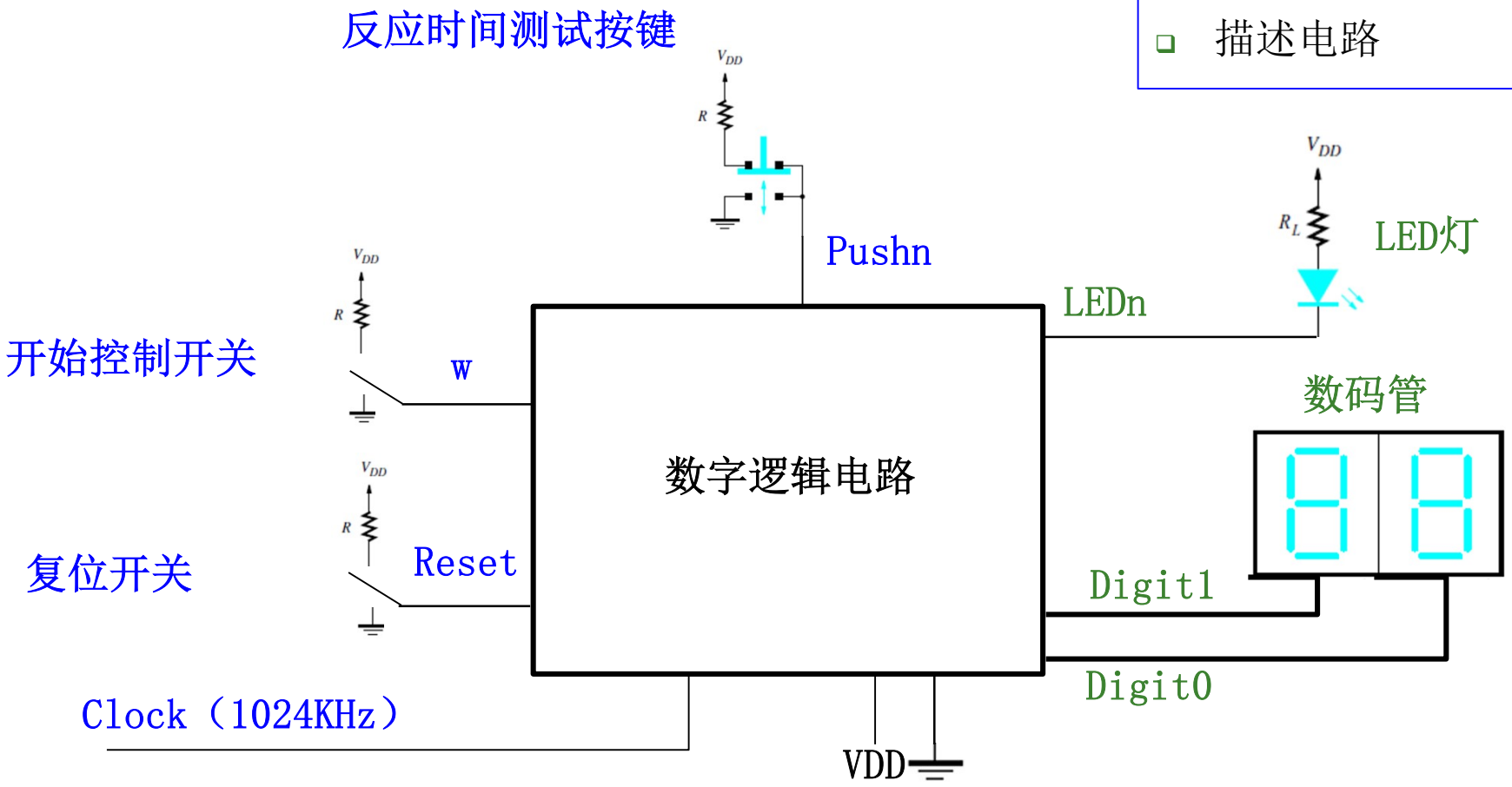
设计实例——系统构成

- 分析需求
- 确定接口信号
- 画出电路结构图
- 描述电路

- 系统输入
  - 输入1：开始控制开关
  - 输入2：反应时间测试按键
  - 输入3：复位开关
- 系统输出信号
  - 输出1：LED灯
  - 输出2：数码管（2个）
- 数字逻辑电路
- 系统时钟源
- 电源：VDD GND

反应计时器系统框图

- 分析需求
- 确定接口信号
- 画出电路结构图
- 描述电路



数字逻辑电路输入输出信号

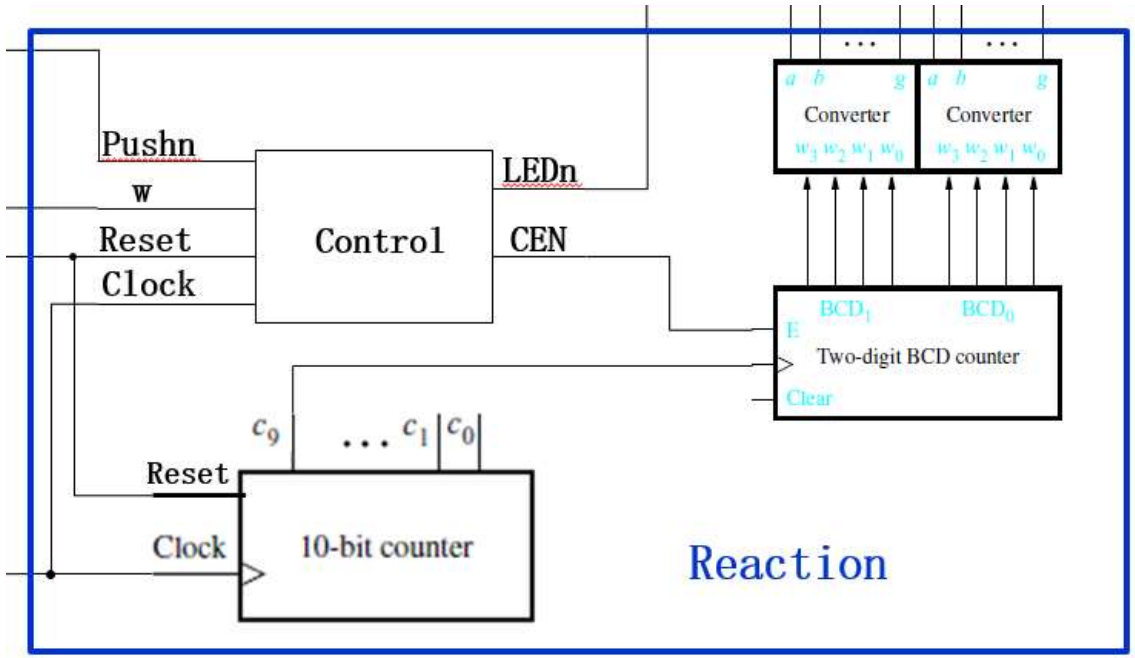
- 系统输入信号
  - Pushn ：测试按键信号, 1 位,低电平有效
    LED灭、计时器停止计时、计时器清零、数码管清零
  - w：开始控制信号,1 位,高电平有效
    LED亮、计时器开始计时、数码管显示计时值
  - Reset：复位信号,1 位,高电平有效
    LED灭、计时器停止计时、数码管显示计时值
  - clk：时钟信号,1 位

- 系统输出信号
  - LEDn：LED灯使能信号　1位（低电平有效）
  - Digit1：数码管控制信号　7 位
  - Digit0：数码管控制信号　7 位

- 分析需求
- 确定接口信号
- 画出电路结构图
- 描述电路

逻辑电路模块划分

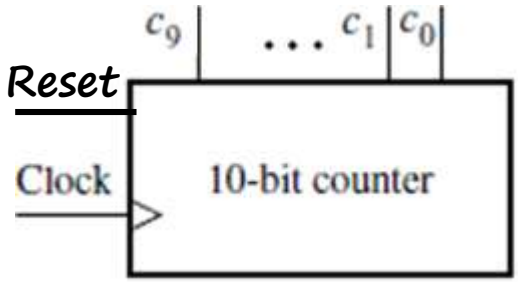
- 分析需求
- 确定接口信号
- 画出电路结构图
- 描述电路



- 1/10分频器模块1个
- BCD-数码管译码器模块2个
- 控制模块1个
- 具有使能和复位清零的BCD计数器模块1个

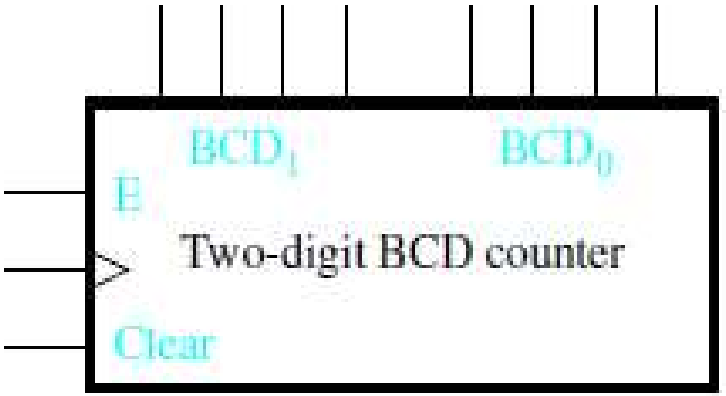
模块输入输出设计

- 分频器模块
  - 输入：
    - Clock 102.4KHz时钟信号 1位
    - Reset 1位
  - 输出：c0,…,c9 分频时钟信号 10位



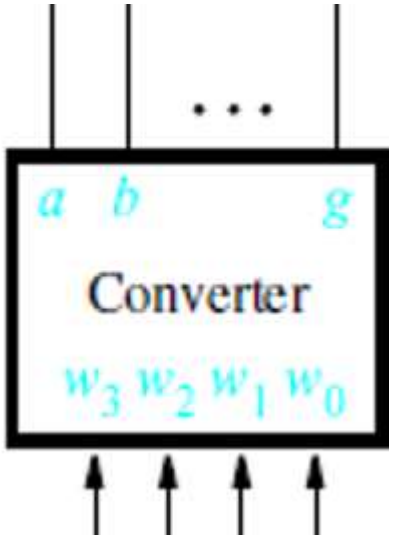
模块输入输出设计

- BCD计数器模块：
  - 输入：
    - Clk 1位 （ 100Hz时钟信号 ）
    - E 1位 （使能信号 ）
    - Clear 1位（复位清零信号 ）
  - 输出：
    - BCD0 4位
    - BCD1 4位



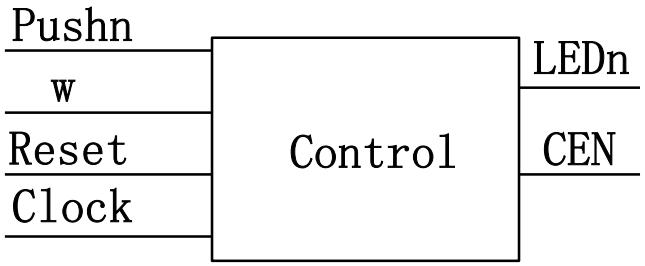
模块输入输出设计

- BCD-数码管译码模块：
  - 输入信号：bcd 4位(w3, w2, w1, w0)
  - 输出：leds 7位 （a,b,…,g）

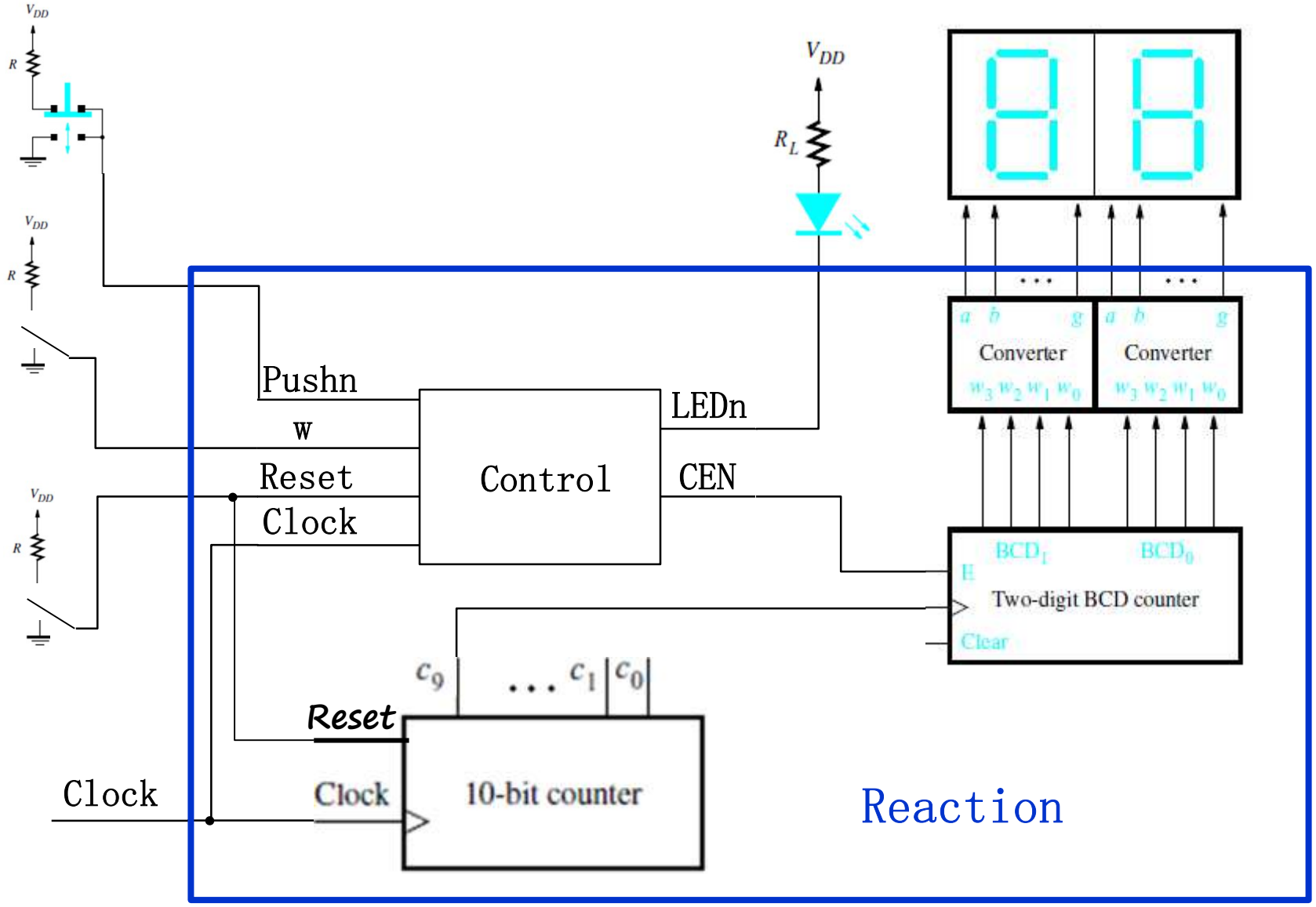


模块输入输出设计

- 控制模块：
  - 输入信号
    - Clock 1位(时钟)
    - w 1位 （测试开始控制信号）
    - Reset 1位（复位信号）
    - Pushn 1位（测试按键信号）
  - 输出信号
    - CEN 1位 （BCD计数器的使能控制信号）
    - LEDn 1位 （LED使能信号，LEDn=~CEN）

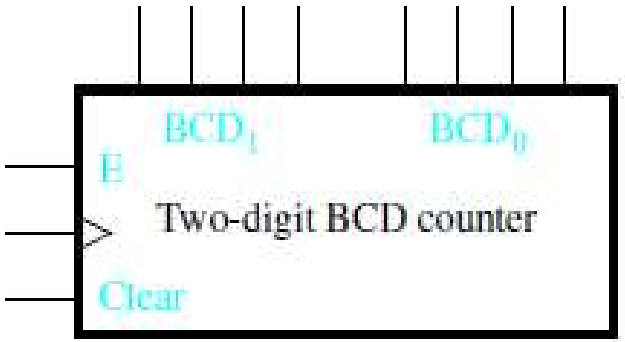


模块链接图

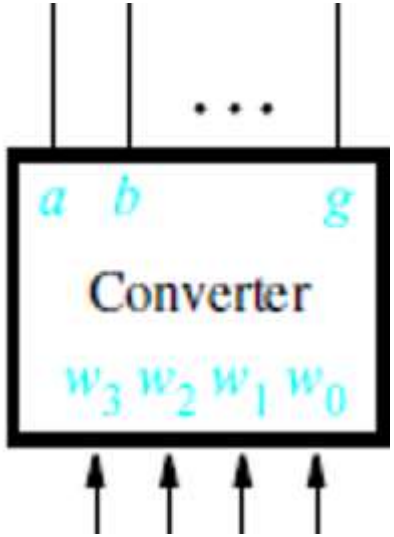

V_{DD}
R
R_L
Pushn
w
Reset
Clock
Control
LEDn
CEN
Converter
Converter
a b g
$w_3 w_2 w_1 w_0$
BCD_1
BCD_0
E
Two-digit BCD counter
Clear
c_9
c_1
c_0
Reset
Clock
10-bit counter
Clock
Reaction

```
module BCDcount (Clock, Clear, E, BCD1, BCD0);
    input Clock, Clear, E;
    output reg [3:0] BCD1, BCD0;

    always @(posedge Clock)
    begin
      if (Clear)
      begin
        BCD1 <= 0;
        BCD0 <= 0;
      end
      else if (E)
        if (BCD0 == 4'b1001)
        begin
            BCD0 <= 0;
            if (BCD1 == 4'b1001)
                BCD1 <= 0;
            else
                BCD1 <= BCD1 + 1;
        end
        else
            BCD0 <= BCD0 + 1;
    end
endmodule
```

- 分析需求
- 确定接口信号
- 画出电路结构图
- 描述电路



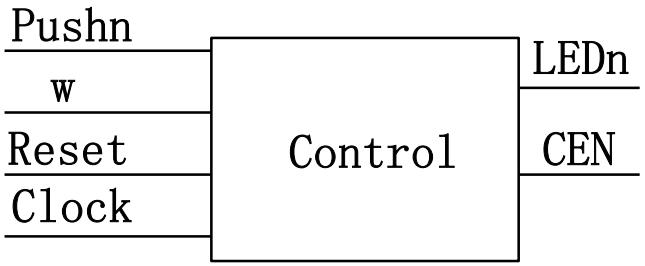
```
module  seg7 (bcd, leds);
    input  [3:0] bcd;
    output  reg  [1:7] leds;

    always @(bcd)
        case (bcd)    //abcdefg
            0: leds = 7'b1111110;
            1: leds = 7'b0110000;
            2: leds = 7'b1101101;
            3: leds = 7'b1111001;
            4: leds = 7'b0110011;
            5: leds = 7'b1011011;
            6: leds = 7'b1011111;
            7: leds = 7'b1110000;
            8: leds = 7'b1111111;
            9: leds = 7'b1111011;
            default: leds = 7'b1111110;
        endcase

endmodule
```



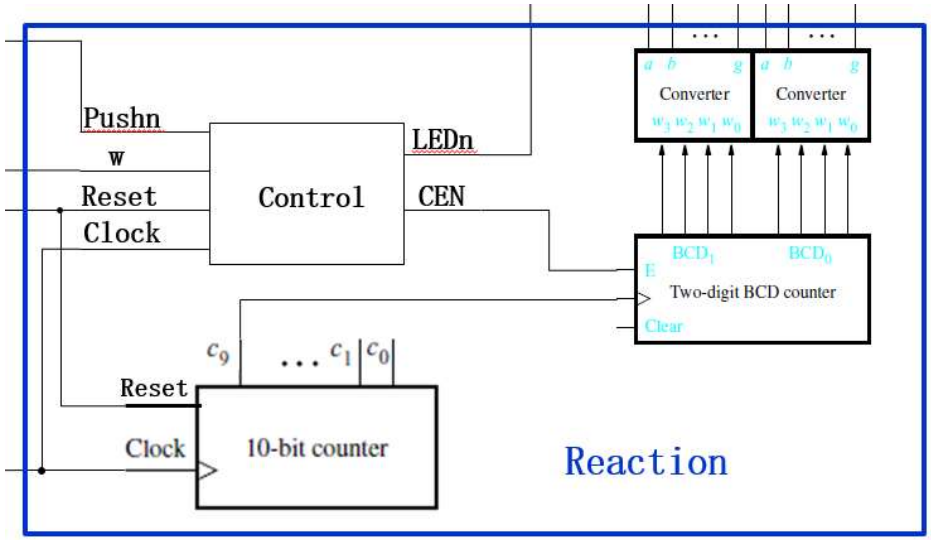
控制器行为描述

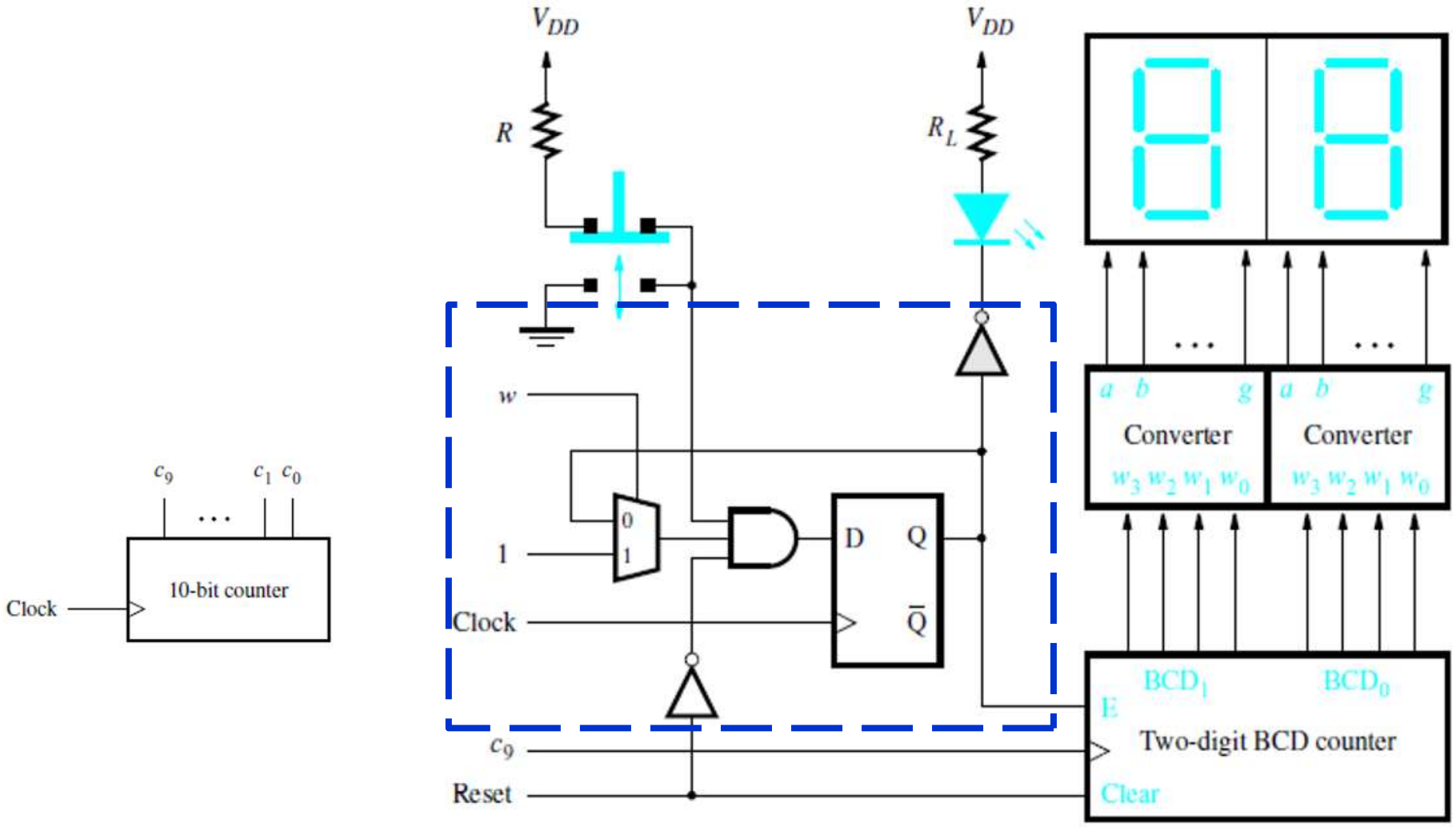
- Reset:
  - 1: LED熄灭、计数器停止计时**(CEN=0, LEDn=1)**
  - 0: 输出不变
- w
  - 1:LED点亮、计数器开始计时**(CEN=1, LEDn=0)**
  - 0:输出不变
- Pushn
  - 1：输出不变
  - 0： LED熄灭、计数器停止计时**(CEN=0, LEDn=1)**

```
module Control (Clock, Reset, w, Pushn, CEN, LEDn);
    input Clock, Reset, w, Pushn;
    output reg CEN;
    output LEDn;

    always @(posedge Clock)
    begin
       if (Reset || ! Pushn)
          CEN<=0;
       else if (w)
          CEN<=1;
    end
   assign LEDn=~CEN;

endmodule
```



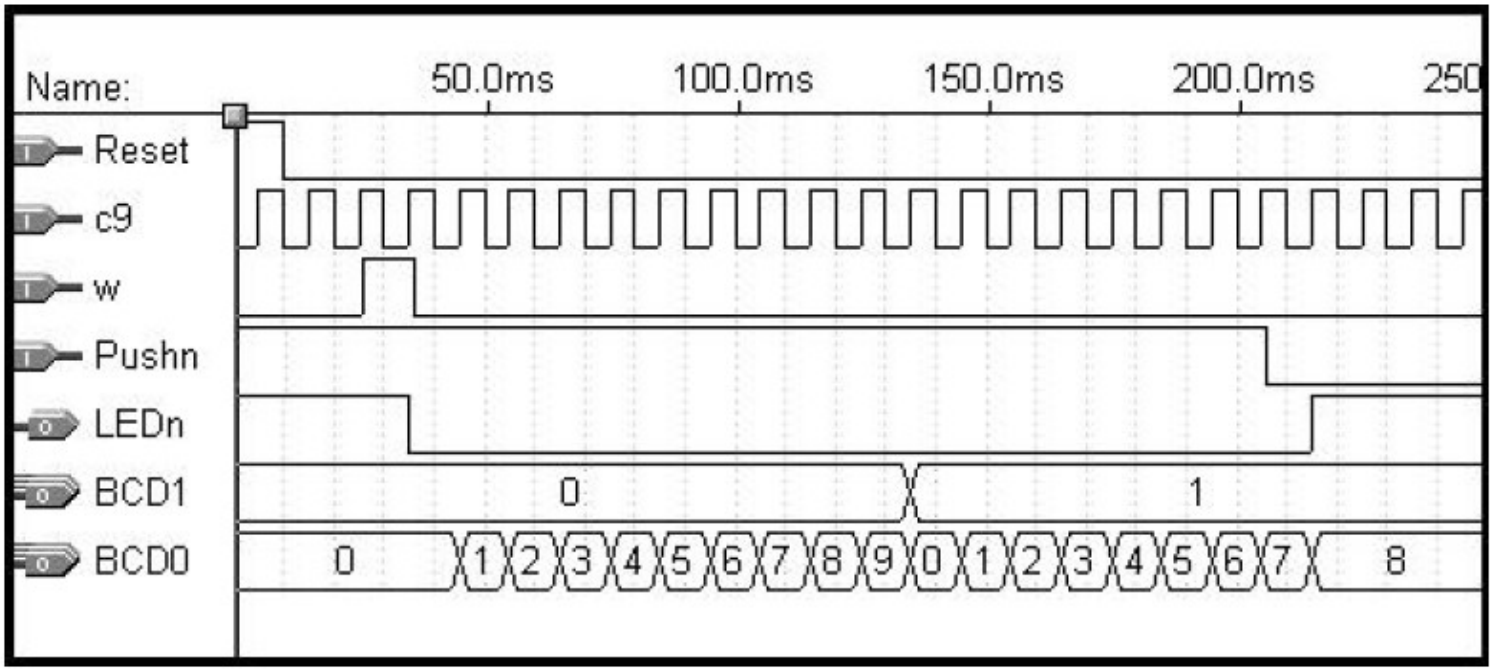
```
module Reaction(Clock, Reset, w, Pushn, LEDn, Digit1, Digit0);
    input Clock, Reset, w, Pushn;
    output wire LEDn;
    output wire [1:7] Digit1, Digit0;
    wire CEN, c0, c1, c2, c3, c4, c5, c6,  c7, c8, c9;
    wire [3:0] BCD1, BCD0;

   Control    controller(Clock, Reset, w, Pushn, CEN, LEDn);
   clk_div_phase     divider(Clock, c0, c1, c2, c3, c4, c5, c6,  c7, c8, c9);
    BCDcount    counter(c9, Reset, CEN, BCD1, BCD0 );
     seg7    seg1(BCD1, Digit1);
     seg7    seg0(BCD0, Digit0);


endmodule
```




V_{DD}
V_{DD}
R
R_L
w
1
Clock
0
1
D
Q
$\bar{Q}$
c_9
Reset
c_9
c_1 c_0
10-bit counter
Clock
a b g
Converter
w_3 w_2 w_1 w_0
a b g
Converter
w_3 w_2 w_1 w_0
BCD_1
BCD_0
E
Two-digit BCD counter
Clear

```
module reaction (Clock, Reset, c9, w, Pushn, LEDn, Digit1, Digit0);
    input Clock, Reset, c9, w, Pushn;
    output wire LEDn;
    output wire [1:7] Digit1, Digit0;
    reg LED;
    wire [3:0] BCD1, BCD0;

    always @(posedge Clock)
    begin
        if (!Pushn || Reset)
            LED <= 0;
        else if (w)
            LED <= 1;
    end

    assign LEDn = ~LED;
    BCDcount counter (c9, Reset, LED, BCD1, BCD0);
    seg7 seg1 (BCD1, Digit1);
    seg7 seg0 (BCD0, Digit0);

endmodule
```

反应计时器仿真结果


Name:
50.0ms
100.0ms
150.0ms
200.0ms
250
Reset
c9
w
Pushn
LEDn
BCD1
BCD0
0
1
0
1
2
3
4
5
6
7
8
9
0
1
2
3
4
5
6
7
8

目录

常用设计约束

- 时序约束：规范设计的时序行为，表达设计者期望满足的时序条件，指导综合和布局布线阶段的优化方法等
  - 作用：提高设计的工作频率；获得正确的时序报告
- 区域与位置约束：主要指芯片I/O引脚位置，以及指导工具在芯片特定的物理区域进行布局布线
- 其他约束：目标芯片型号、电气特性等

几种常见的时序约束的基本概念

1.周期与最高频率：通常指时钟所能达到的最高工作频率。

2.时钟建立时间t_{su}：指时钟到达前，数据和使能信号已经准备好的最小时间间隔。

3.时钟保持时间t_h：指能保证有效时钟沿正确采样的数据和使能信号在时钟沿之后的最小稳定时间间隔。

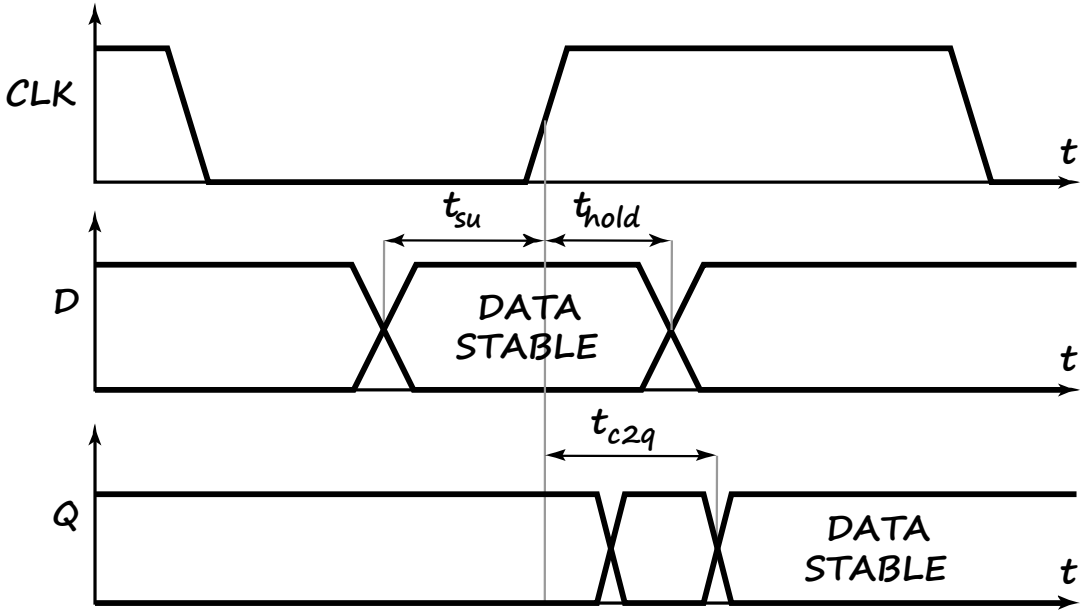
4.时钟输出延时t_{c2}：指从时钟有效沿到数据有效输出的最大时间间隔。

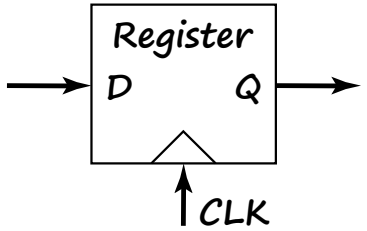
5.引脚到引脚的延时t_{pd}：信号从输入管脚进来到达输出管脚的最大时间间隔。

6.Slack：是否满足时序的称谓。正的Slack表示满足时序，负的Slack表示不满足时序。

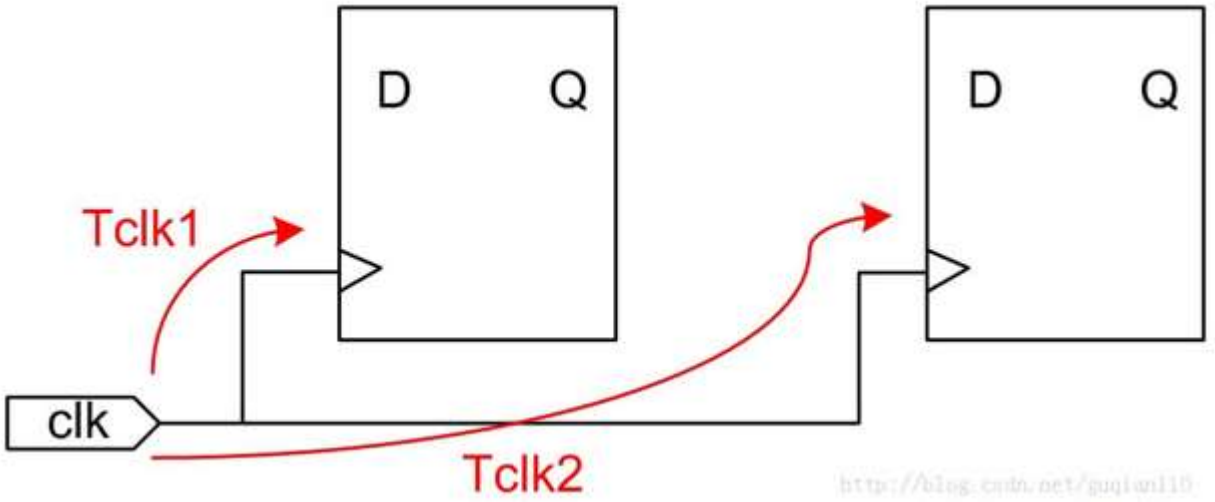
7.时钟偏斜(clock shew)：指一个同源时钟到达两个不同的寄存器时钟端的时间差别。

触发器的时序参数


CLK
t
t_{su}
t_{hold}
D
DATA
STABLE
t
t_{c2q}
Q
DATA
STABLE
t

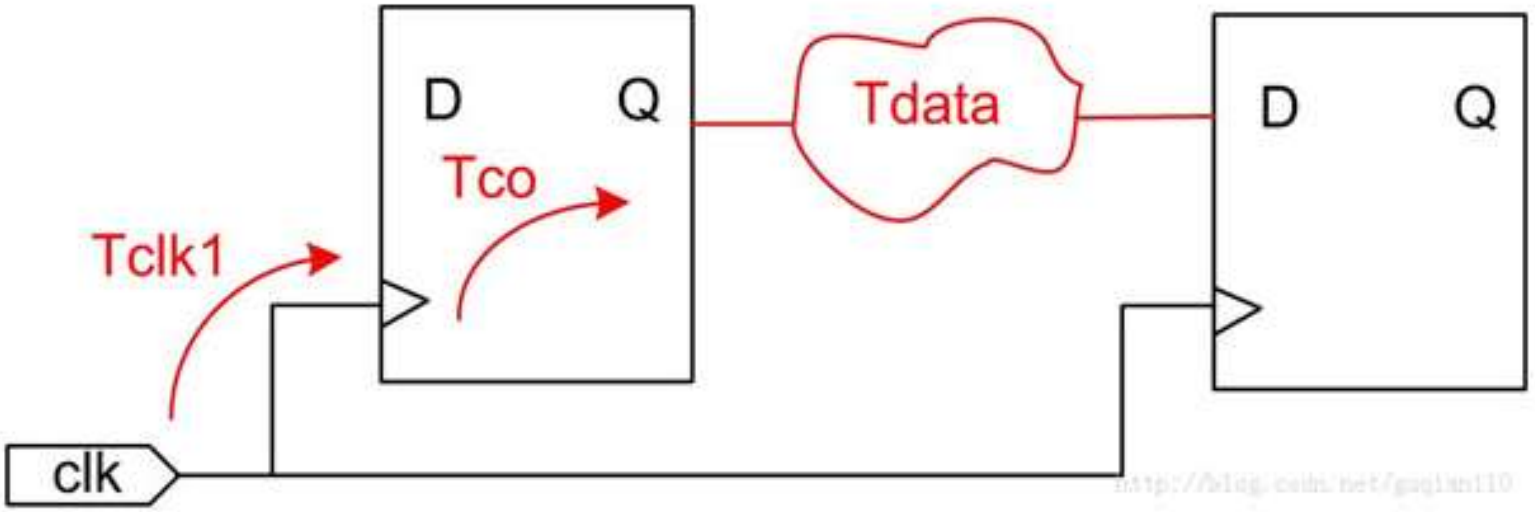

Register
D
Q
CLK

时钟偏斜



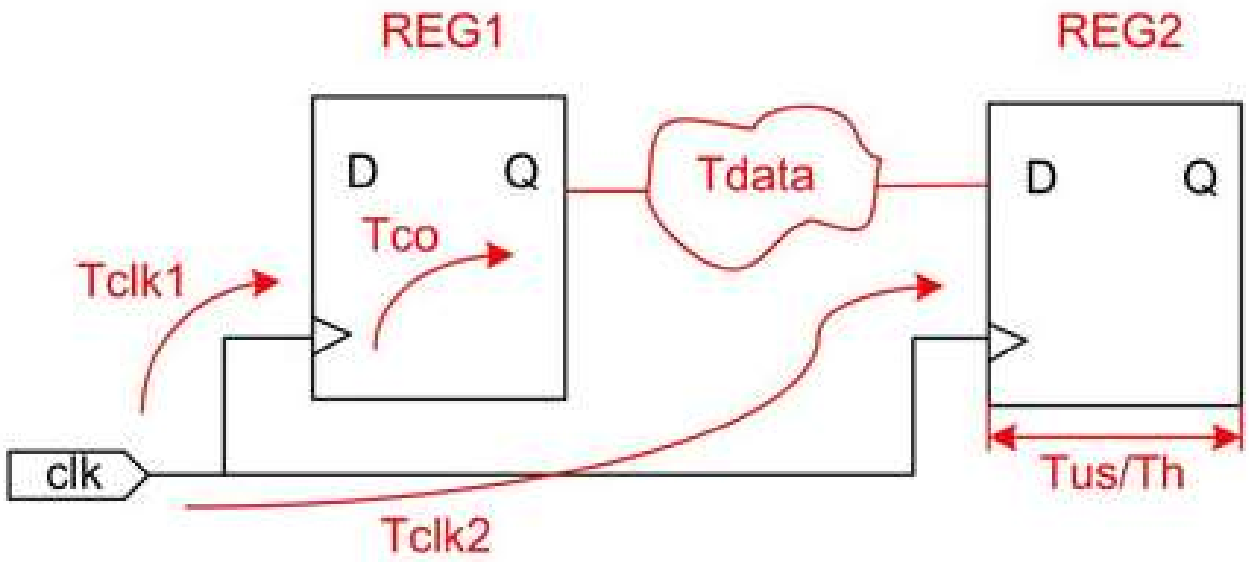
$$Tskew = Tclk2 - Tclk1$$

数据到达时间



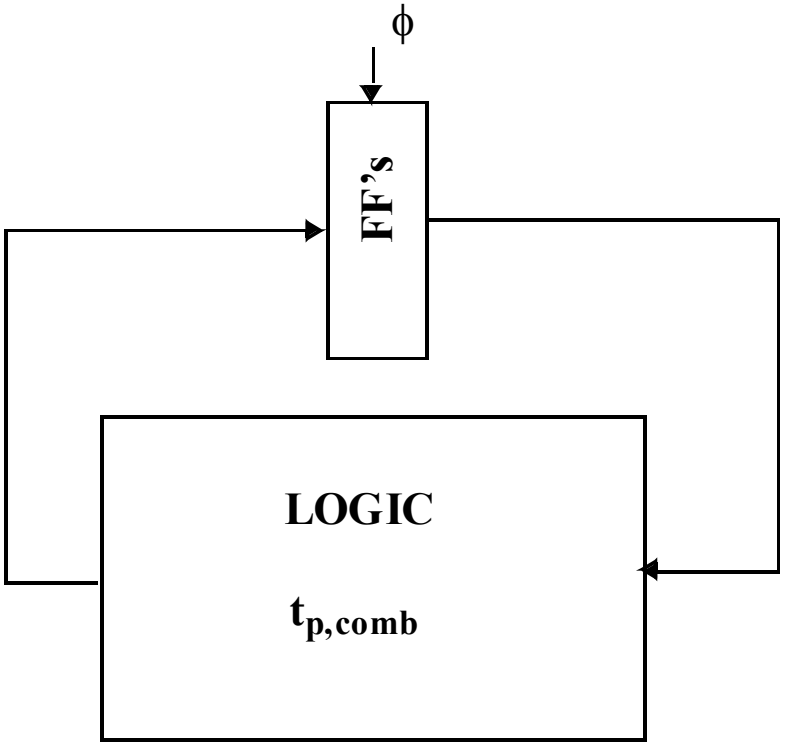
Data Arrival Time = Launch edge + Tclk1 +Tco + Tdata

Setup Slack



Setup slack = Data Required Time - Data Arrival Time

最大时钟频率



$T_{min} = t_{c2Q\_max} + t_{p,comb} + t_{su}$

$F_{max} = 1/T_{min}$

满足：

$t_{c2Q\_min} + t_{p,comb} > t_{hold}$