数 字 电 路

Digital Circuits

12_锁存器和触发器(2)

张俊霞
zjx@ustc.edu.cn

内容提纲

- 触发器逻辑功能
- 触发器功能转换
- 锁存器与触发器动态特性
- 常用集成锁存器和触发器
- Verilog HDL描述锁存器和触发器

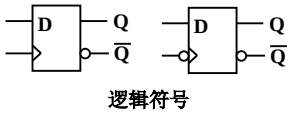
触发器逻辑功能

- **在时钟信号有效边沿(上升沿或下降沿)，触发器根据输入信号更新状态**
  - 现态：时钟信号有效边沿前触发器的状态，记为Q^n
  - 次态：时钟信号有效边沿后触发器的状态，记为Q^{n+1}
- **所谓触发器的逻辑功能，是指次态与现态、输入信号之间的逻辑关系**
  - 描述方法有：特性表(真值表)、特性方程(逻辑表达式)和状态图(状态相互转换图)，三者可以相互转换
- **触发器按逻辑功能分类：D触发器、 T触发器、JK触发器、SR触发器**

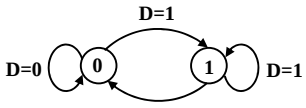
D触发器

特性表

D	Q^n	Q^{n+1}	说明
0	0	0	清0
0	1	0	
1	0	1	置1
1	1	1	



逻辑符号



状态图

特性方程 $Q^{n+1} = D$

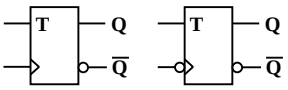
T触发器

特性表

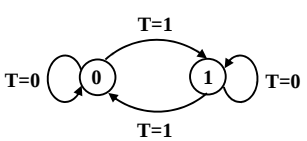
T	Q^n	Q^{n+1}	说明
0	0	0	保持
0	1	1	
1	0	1	翻转
1	1	0	

特性方程

$$Q^{n+1} = \overline{T}Q^n + T\overline{Q^n}$$
$$= T \oplus Q^n$$



逻辑符号

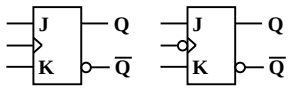


状态图

JK触发器

特性表

J	K	Q^n	Q^{n+1}	说明
0	0	0	0	保持
0	0	1	1	
0	1	0	0	清0
0	1	1	0	
1	0	0	1	置1
1	0	1	1	
1	1	0	1	翻转
1	1	1	0	



逻辑符号

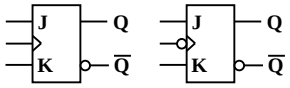
Q^{n+1} JK / Q^n	00	01	11	10
0	0	0	1	1
1	1	0	0	1

特性方程 $Q^{n+1} = J\overline{Q^n} + \overline{K}Q^n$

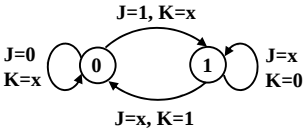
JK触发器 (续)

特性表

J	K	Q^n	Q^{n+1}	说明
0	0	0	0	保持
0	0	1	1	
0	1	0	0	清0
0	1	1	0	
1	0	0	1	置1
1	0	1	1	
1	1	0	1	翻转
1	1	1	0	



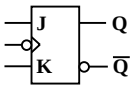
逻辑符号

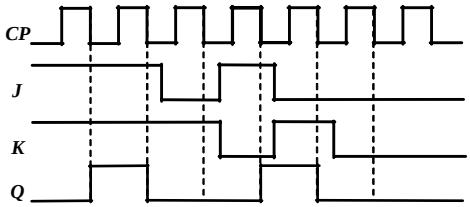


状态图

示例—JK触发器波形图

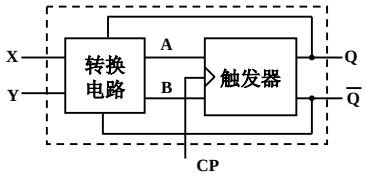
- 画出输出波形(设初态为0)



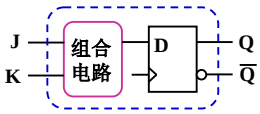


触发器功能转换

- 利用某种功能触发器来构造不同功能的触发器
  - 根据两种触发器的特性方程，求解转换电路的逻辑函数式



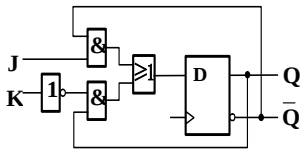
D触发器构成JK触发器

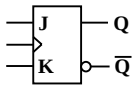


$Q^{n+1} = D$

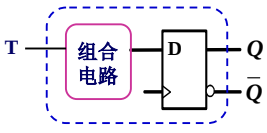
$Q^{n+1} = J\overline{Q^n} + \overline{K}Q^n$

$D = J\overline{Q^n} + \overline{K}Q^n$

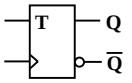




D触发器构成T触发器

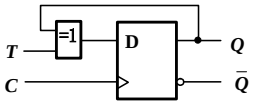


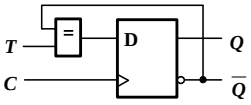
$Q^{n+1} = D$



$Q^{n+1} = T\overline{Q^n} + \overline{T}Q^n$

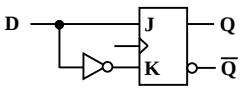
$D = T\overline{Q^n} + \overline{T}Q^n = T \oplus Q^n$



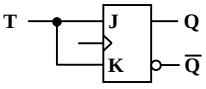


JK触发器构成其他触发器

- 构成D触发器
  - J=D，K=$\overline{D}$　$Q^{n+1} = D$



- 构成T触发器
  - J=K=T　$Q^{n+1} = T \oplus Q^n$



JK触发器特性表

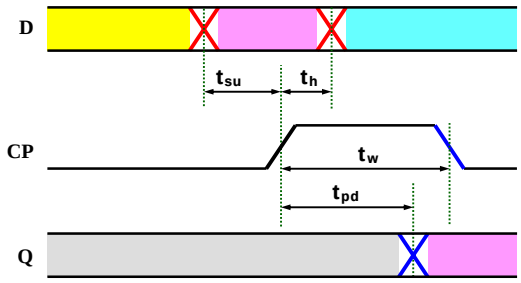
J	K	Q^n	Q^{n+1}	说明
0	0	0	0	保持
0	0	1	1	
0	1	0	0	清0
0	1	1	0	
1	0	0	1	置1
1	0	1	1	
1	1	0	1	翻转
1	1	1	0	

$Q^{n+1} = J\overline{Q^n} + \overline{K}Q^n$

锁存器和触发器动态特性

- 保证锁存器和触发器可靠地更新状态，输入信号与时钟信号之间的时间要求
  - 建立时间：t_{su}
  - 保持时间：t_h
- 锁存器和触发器输出信号对时钟信号响应的延迟时间
  - 传输延迟时间：t_{pd}

触发器定时波形图



示例—分析D触发器动态参数

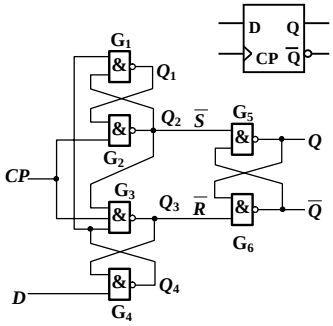
- 设所有门的传输延迟都等于td，忽略导线延迟，求：

  tsu，th，tpd，tw

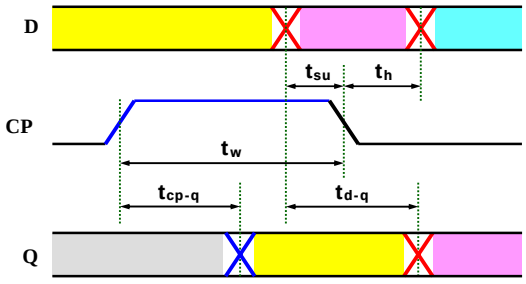
  tsu = 2td

  th = td

  tpd = tw = 3td



锁存器定时波形图

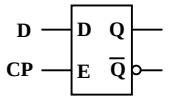


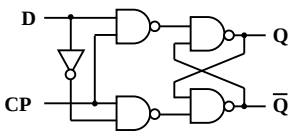
示例—分析D锁存器动态参数

- 设所有门的传输延迟都等于td，忽略导线延迟，求：

  tsu，th，tcp-q，td-q，tw

课后思考题





常用集成锁存器和触发器

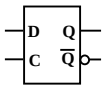
- 集成锁存器
  - RS：279，CD4043，CD4044
  - D： 75， 373，375等
- 集成触发器
  - D：74，174，175，273，374等
  - JK：73，109，112等

Verilog过程赋值语句

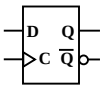
- **在initial和always块中，对reg类型变量赋值**
- **有两种赋值方式**
  - 非阻塞（non-blocking）赋值语句 (<=)
    - 块内的赋值语句并发执行，即同时赋值
  - 阻塞（blocking）赋值语句 (=)
    - 块内的赋值语句顺序执行，完成一条赋值语句后才能做下一句的操作

Verilog描述D锁存器和触发器

```
module D_Latch (Q, Qn,
        D, C);
    input D, C;
    output Q, Qn;
    reg Q;
    assign Qn = ~Q;
    always @(D or C)
      if (C)
        Q <= D;
endmodule
```

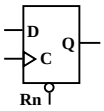



```
module D_FF (Q, Qn, D, C);
    input D, C;
    output Q, Qn;
    reg Q;
    assign Qn = ~Q;
    always @(posedge C)
      Q <= D;
endmodule
```



异步或同步复位D触发器

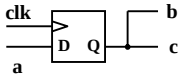
```
module flipflop_a (Q, D,
        Rn, C);
input D, C, Rn;
output Q;
reg Q;

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



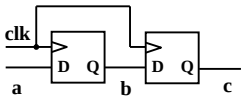

```
module flipflop_s (Q,
        D, Rn, C);
input D, C, Rn;
output Q;
reg Q;

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


示例—阻塞赋值语句

```
always @(posedge
clk)
begin
   b =  a ;
   c = b;
end
```

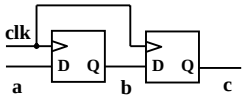



```
always @(posedge
clk)
begin
   c =  b ;
   b = a;
end
```

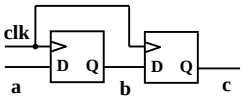


示例—非阻塞赋值语句

```
always @(posedge
clk)
begin
   b <=  a ;
   c <= b;
end
```

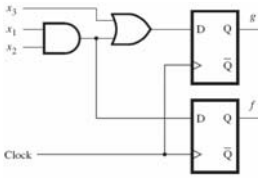



```
always @(posedge
clk)
begin
   c <=  b ;
   b <= a;
end
```



示例—两种赋值语句比较

```
module basgn (x1, x2, x3,
   Clock, f, g);
input x1, x2, x3, Clock;
output f, g;
reg f, g;
always @(posedge Clock)
  begin
    f = x1 & x2;
    g = f | x3;
  end
endmodule
```



示例—两种赋值语句比较(续)

```
module nbasgn (x1, x2, x3,
   Clock, f, g);
input x1, x2, x3, Clock;
output f, g;
reg f, g;
always @(posedge Clock)
   begin
     f <= x1 & x2;
     g <= f | x3;
   end
endmodule
```