第五讲 时序逻辑
Sequential logic

佟冬
tongdong@pku.edu.cn

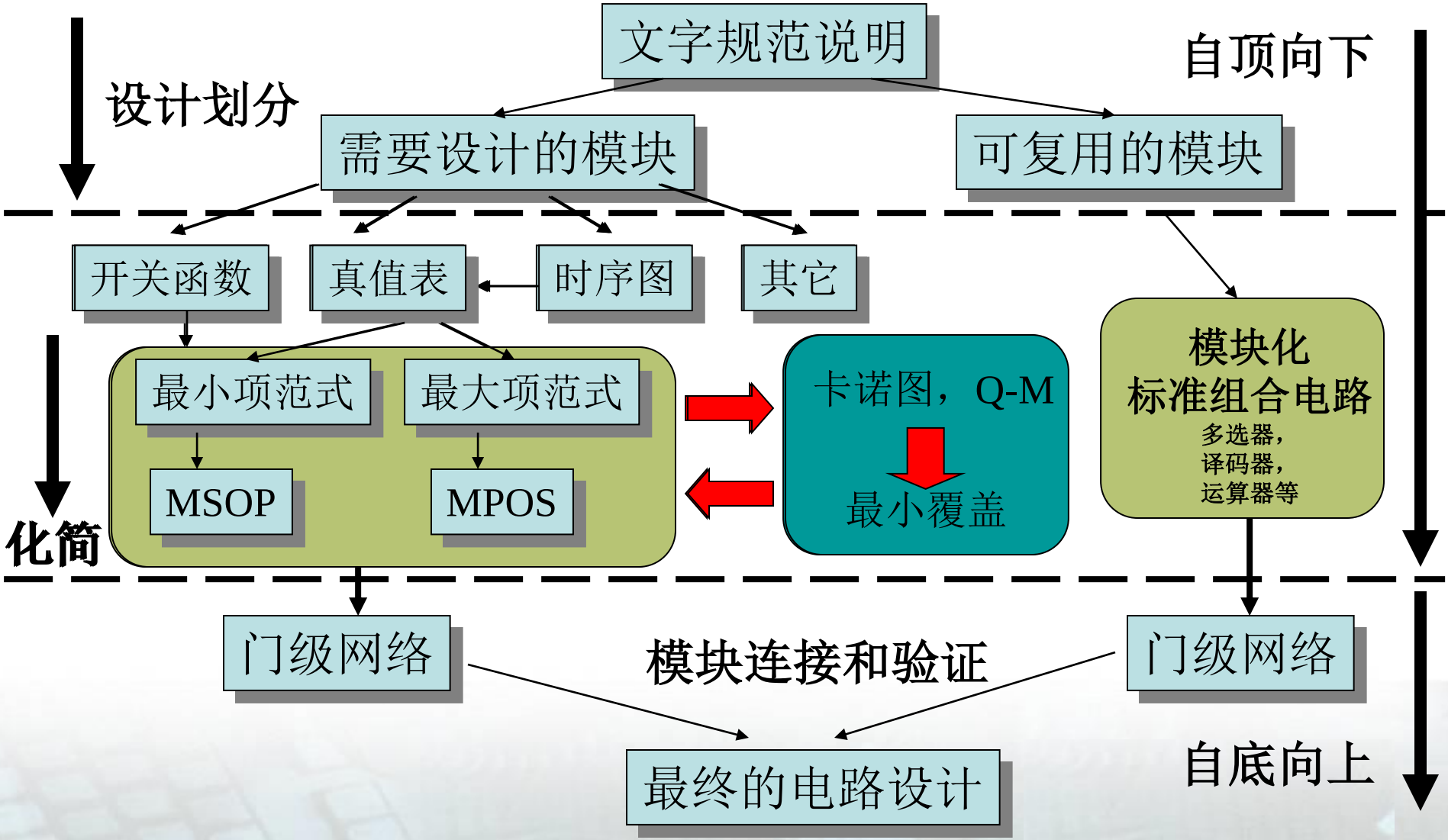
微处理器研究开发中心（MPRC）
计算机科学技术系

北京大学

课程回顾

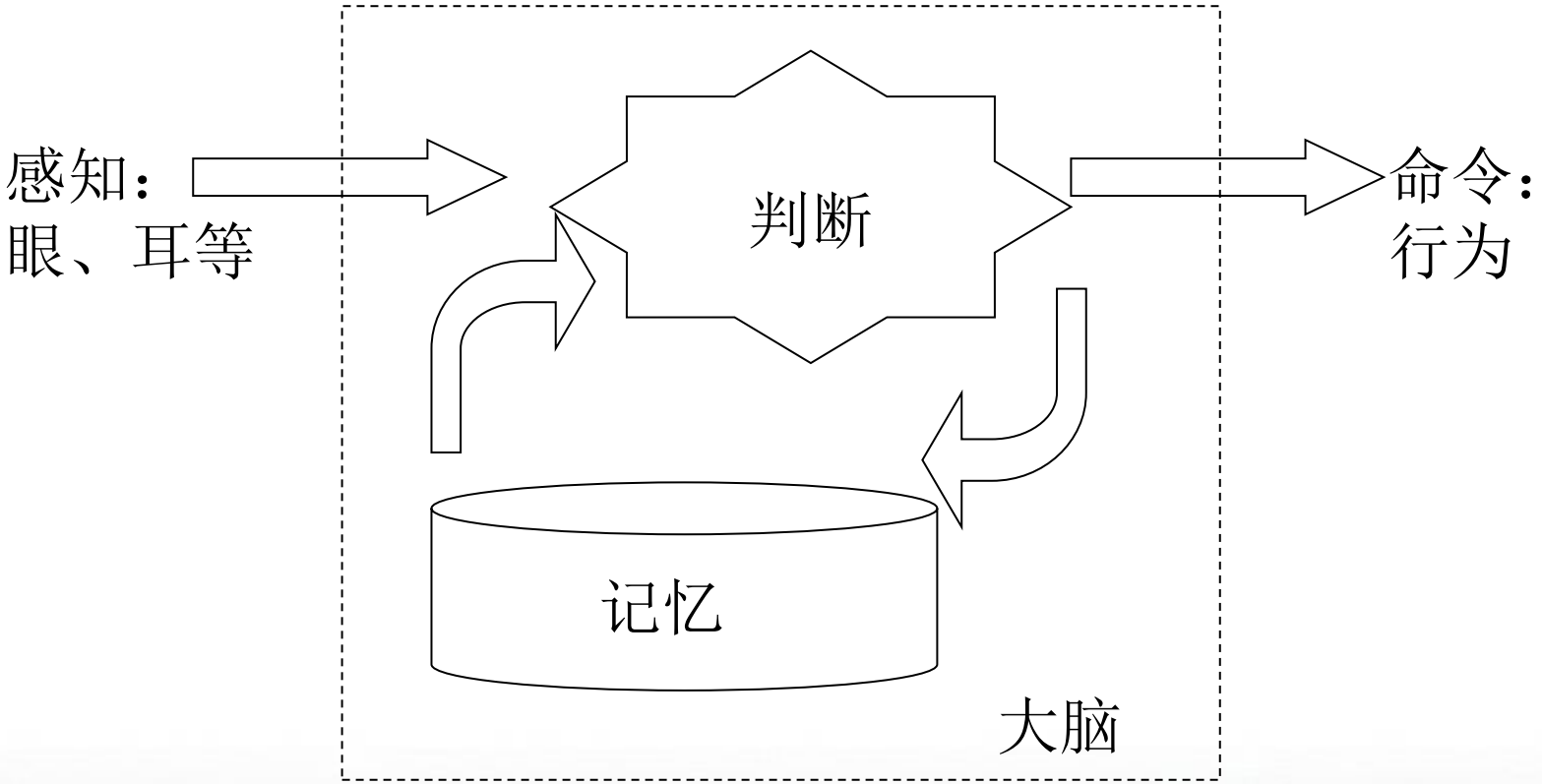
- 二进制及编码
- 布尔代数（共设和定理）
- 开关函数和开关电路
  - 开关函数和开关电路的对应关系
- 组合电路分析与综合
  - 组合电路的刻画：输入、输出、函数公式
  - 原理图、Verilog语言
- 组合电路的化简（门数和扇入）
  - 开关函数的化简、卡诺图法、Q-M方法
- 自顶向下的设计方法
- 标准组合电路模块（多选择/译码器/加法器）

组合逻辑总结

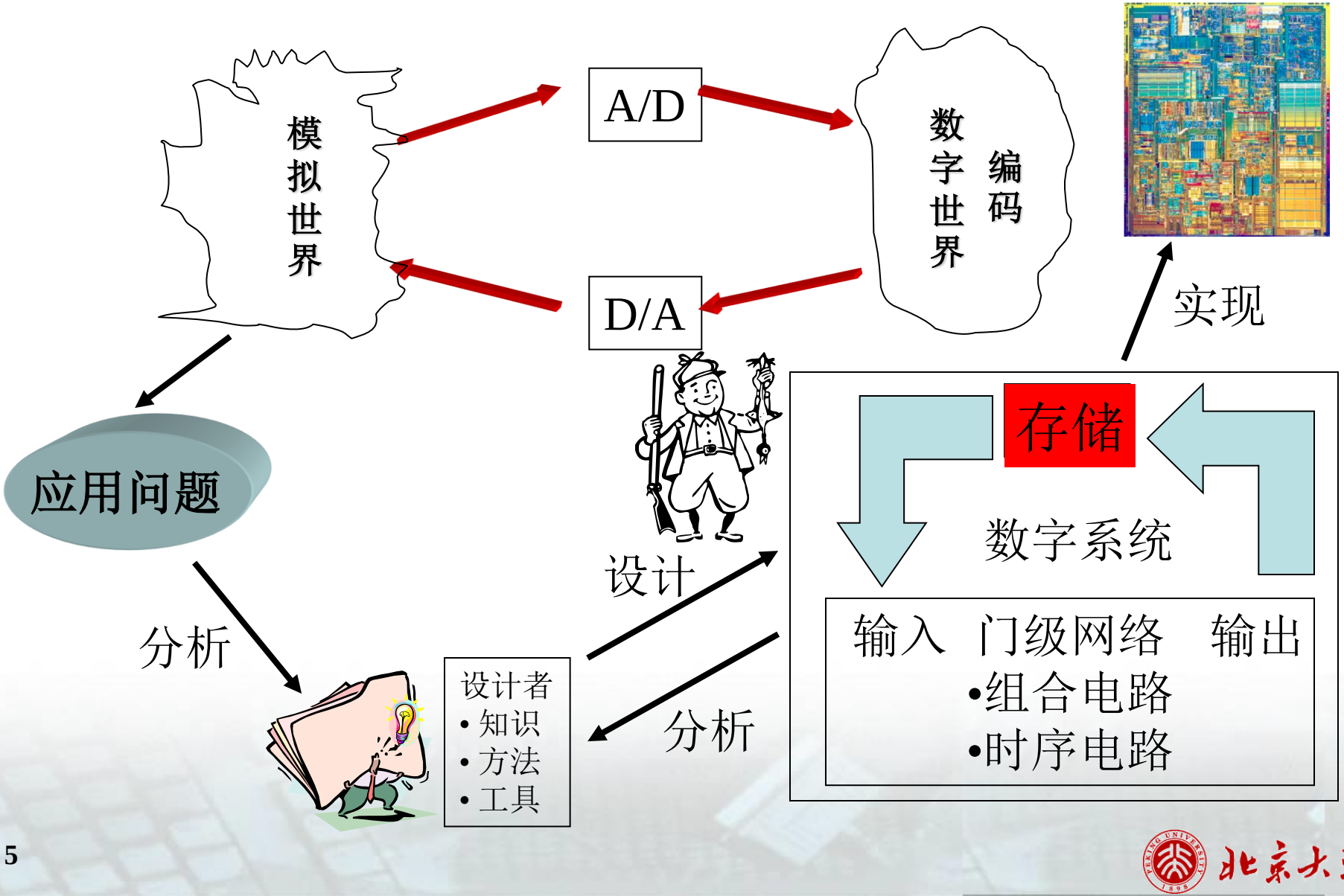

文字规范说明
自顶向下
设计划分
需要设计的模块
可复用的模块
开关函数
真值表
时序图
其它
最小项范式
最大项范式
MSOP
MPOS
卡诺图，Q-M
最小覆盖
模块化
标准组合电路
多选器，
译码器，
运算器等
化简
门级网络
模块连接和验证
门级网络
最终的电路设计
自底向上

人是如何计算的?

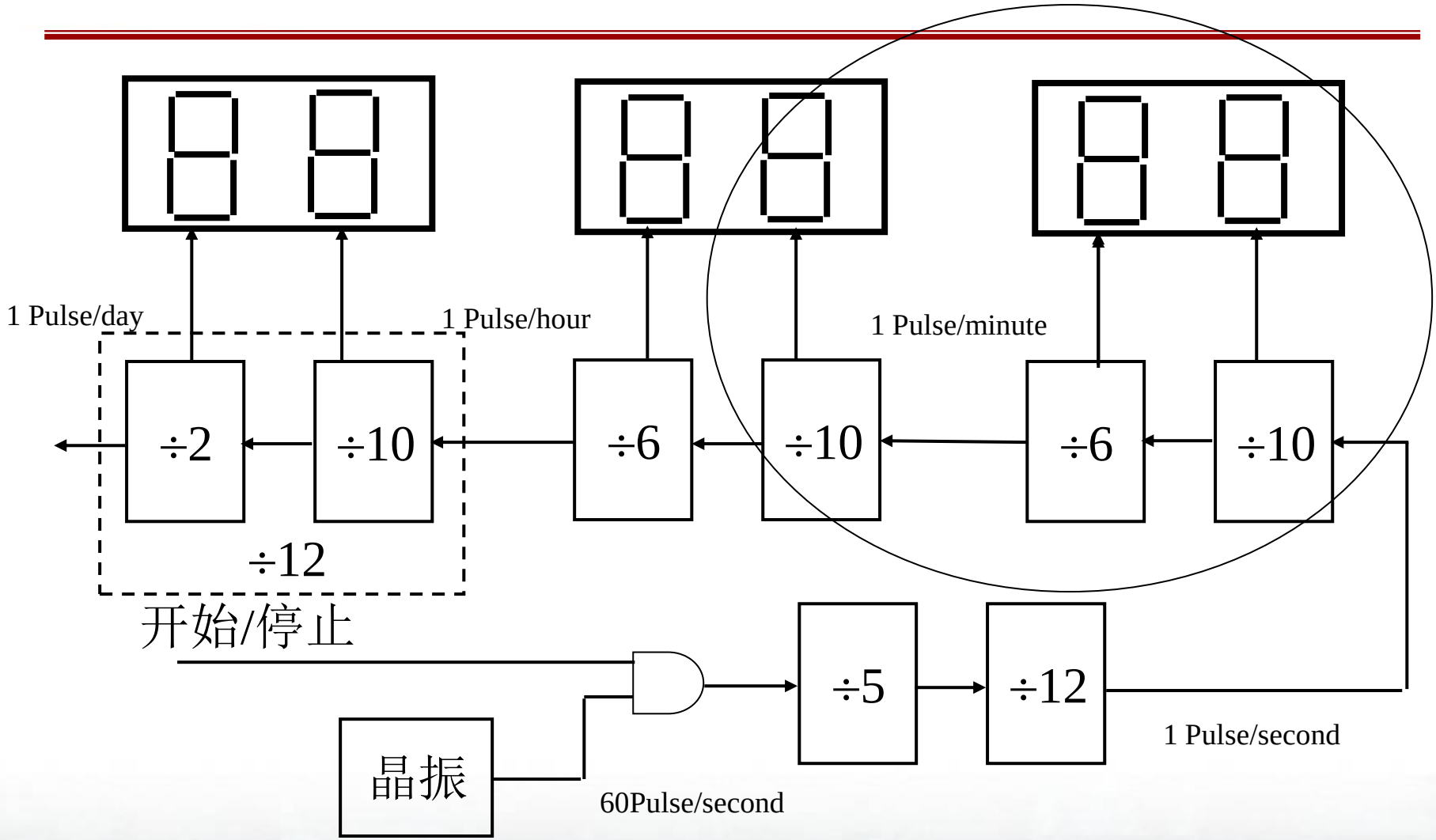
- 简单的模型?



数字电路的分析与设计



Lab：电子秒表

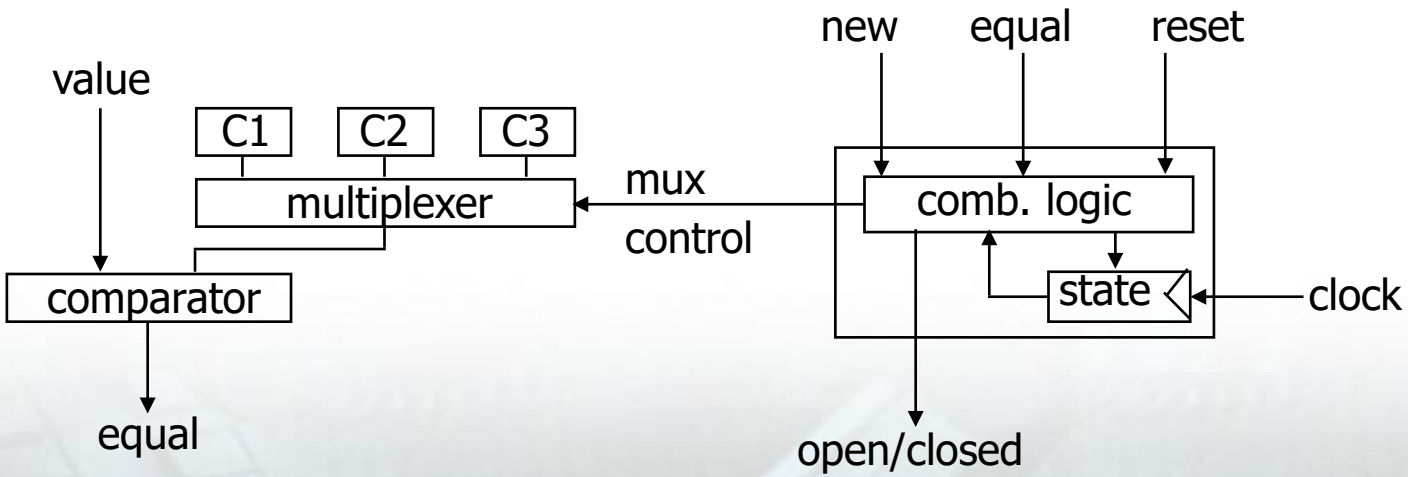


1 时序电路的引入

- 电梯控制电路的设计
- 当前状态(现态，present state)
  - 电梯所在的层数
  - 电梯的运行方向
- 下一个状态(次态，next state)
  - 电梯将要运行的方向
  - 电梯将要去的层数
- 输入(input)
  - 电梯内部的控制按钮
  - 各楼层的电梯控制按钮
- 输出(output)
  - 对电梯电机系统发出的控制信号
- 状态转换(state transition)

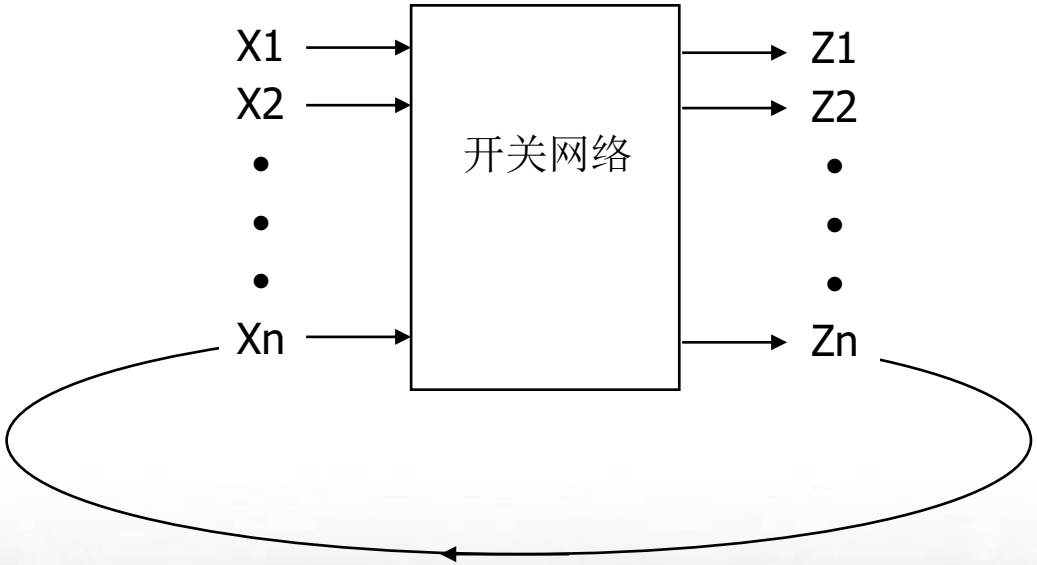
时序电路举例

- 带反馈（feedback）的电路
  - outputs = f(inputs, past inputs, past outputs)
  - 在逻辑电路中构造“记忆”的基础
  - 门组合锁例子
    - 状态是记忆
    - 状态是组合逻辑的输出和输入
    - 存储单元的组合



带反馈的电路

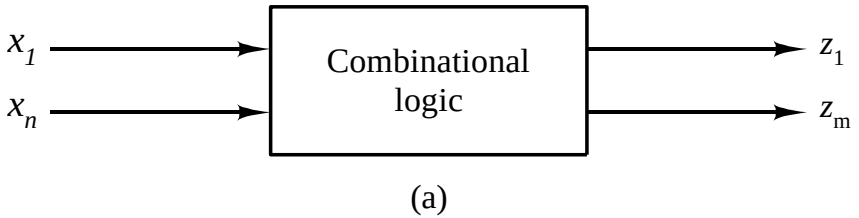
- 如何控制反馈?
  - 如何从无限循环中停止?



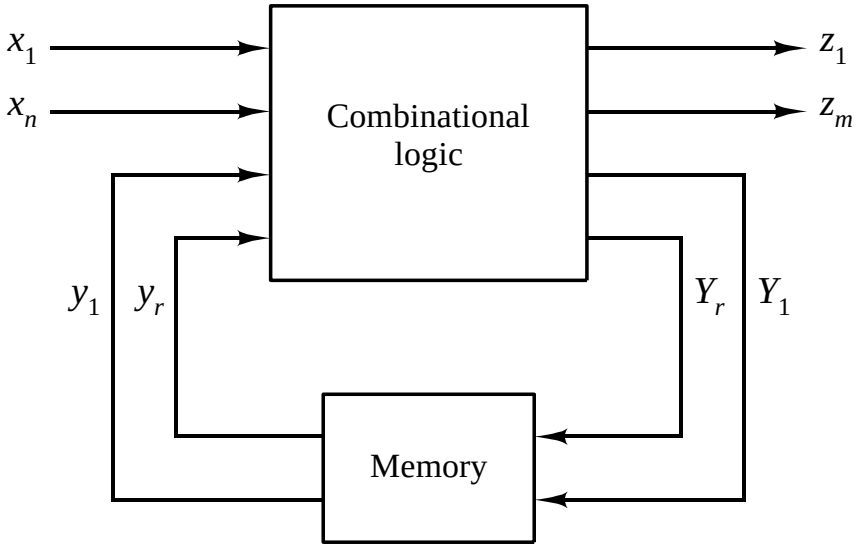
时序电路的基本概念

- 输入$x=(x_1, x_2, \ldots, x_n)$
- 输出$z=(z_1, z_2, \ldots, z_m)$
- 组合电路
  - $z_i=f_i(x_1, x_2, \ldots, x_m)$, $i = 1, \ldots, m$
- 状态 (State)
  - 现态$(y_1, y_2, \ldots, y_r)$
  - 次态$(Y_1, Y_2, \ldots, Y_r)$
- 时序电路函数表示
  - $z_i=g_i(x_1, x_2, \ldots, x_m, y_1, y_2, \ldots, y_r)$, $i = 1, \ldots, m$
  - $Y_i= h_i(x_1, x_2, \ldots, x_m, y_1, y_2, \ldots, y_r)$, $i = 1, \ldots, m$

时序电路的表示—图表示



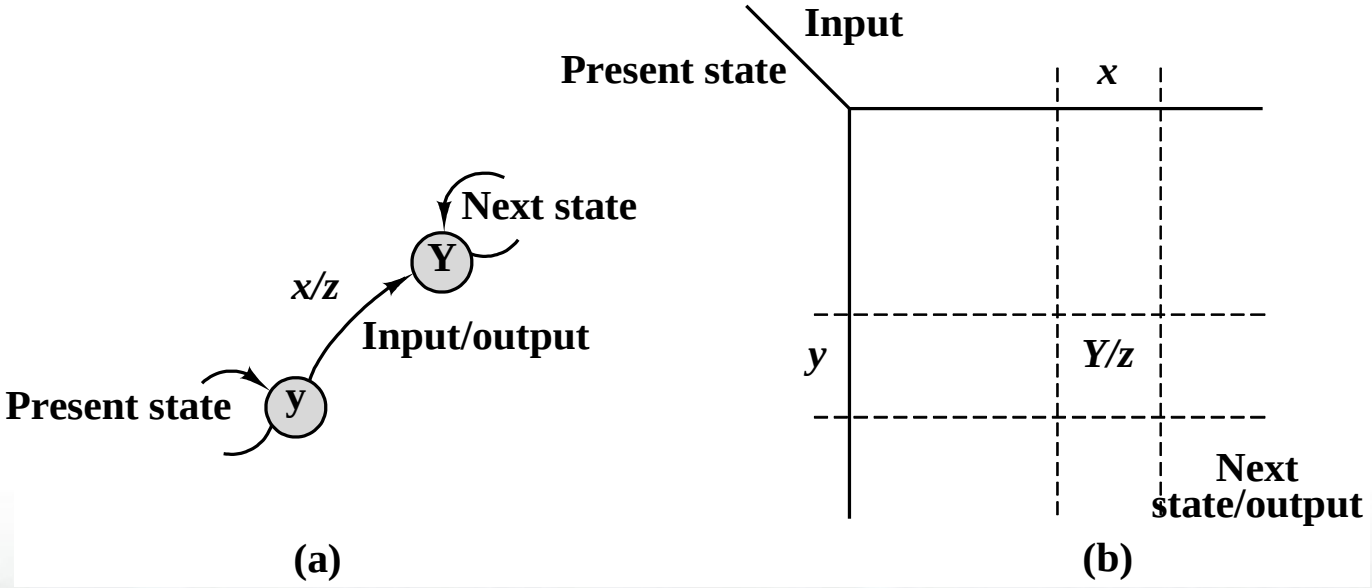
(a)



(b)

状态表和状态图

- 状态图和状态表
  - 圆：状态
  - 线：状态变换
  - 线上标注：产生状态变换的输入和相应输出



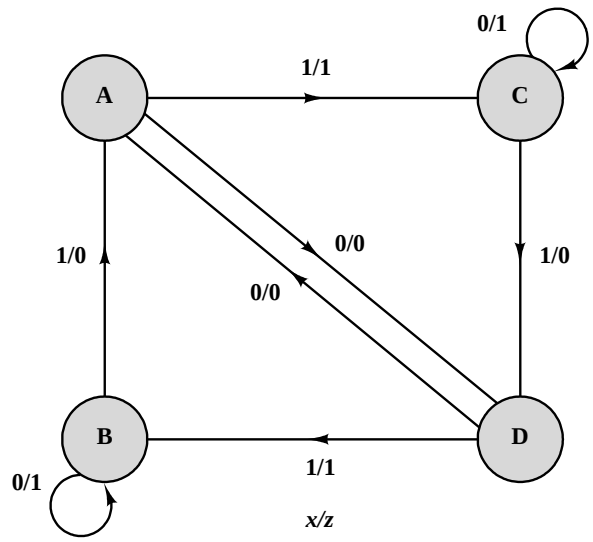
状态变换实例（续）

输入：x = 0110101100
初始状态：A

Present state	Input x = 0	Input x = 1
A	D/0	C/1
B	B/1	A/0
C	C/1	D/0
D	A/0	B/1

(a)

时间：	0	1	2	3	4	5	6	7	8	9	10
现态：	A	D	B	A	D	B	B	A	C	C	C
输入：	0	1	1	0	1	0	1	1	0	0	
次态：	D	B	A	D	B	B	A	C	C	C	
输出：	0	1	0	0	1	1	0	1	1	1	



(b)

输出：z = 0100110111
最后状态：C

2 存储元件

- 存储元件(Memory device)
  - 双稳态(bistable)电子线路
  - 状态 0
  - 状态 1
- 二进制的存储
  - 状态0，表示存储逻辑“0”。
  - 状态1，表示存储逻辑“1”。
  - 输出Q，指示存储元件的现态

存储元件(2)

- 激励输入(excitation inputs)
  - 每个存储元件有多个输入，能激励或者驱动存储元件进入确定的状态的输入，被称为激励输入。
  - 一般的存储元件的命名是根据它与其它存储元件不同的激励输入。
- 存储元件的类型
  - 锁存器(*latch*)
  - 触发器(*flip-flop*)

存储元件(3)

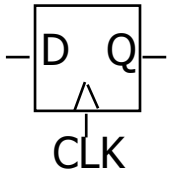
- 锁存器
  - 锁存器的激励输入控制元件的状态。
  - 置位锁存器(set latch)，激励输入强制元件的输出为1。
  - 复位锁存器(reset latch)，激励输入强制元件的输出为0。
  - 置位复位锁存器(set-reset latch)，同时具有置位和复位激励信号的元件。

存储元件(4)

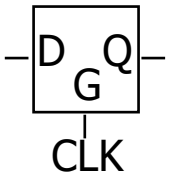
- 触发器
  - 时钟控制信号(*clock*)
  - 时钟信号向触发器发命令，触发器根据激励信号改变状态。
  - 在多触发器的电路中，时钟信号可以使所有的触发器同步(synchronized)的改变状态。
  - 时钟树的概念

存储元件(4)

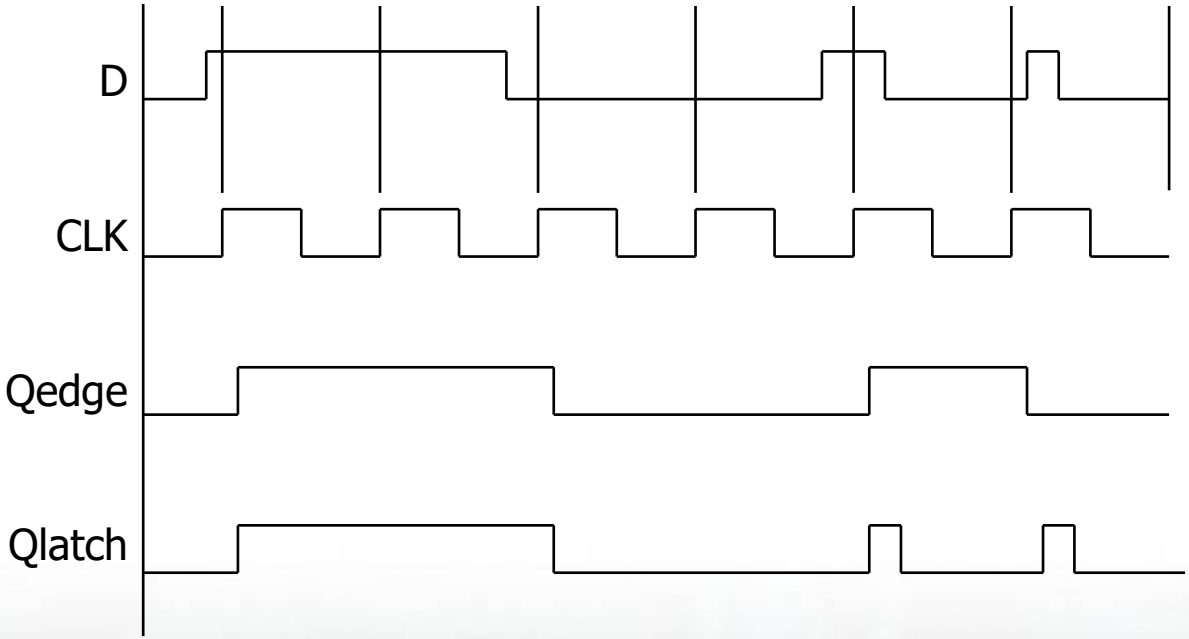
- 锁存器和触发器的操作
  - 锁存器立即响应激励输入
  - 触发器只依赖时钟响应激励输入



触发器



锁存器



存储元件的特征性质

- 锁存器和触发器
- 预置(preset)端和清零(clear)端
  - 同步的预置端和清零端
  - 异步的预置端和清零端
- 脉冲(pulse)触发和边沿(edged)触发
  - 正脉冲触发和负脉冲触发
  - 上升沿触发和下降沿触发

3 锁存器 Latch

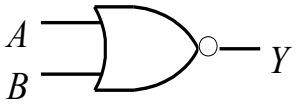
- 或非门(NOR)—复习

a	b	$f_{NOR}(a, b) = \overline{a + b}$
0	0	1
0	1	0
1	0	0
1	1	0

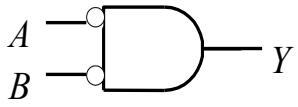
(a)

A	B	Y
L	L	H
L	H	L
H	L	L
H	H	L

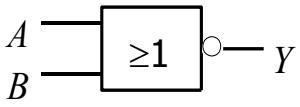
(b)



(c)



(d)



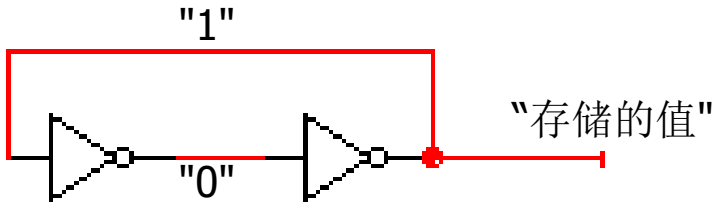
(e)

(a) 或门的逻辑功能
(b) 或门的电子功能
(c)(d) 标准符号表示
(e) IEEE 块符号表示.

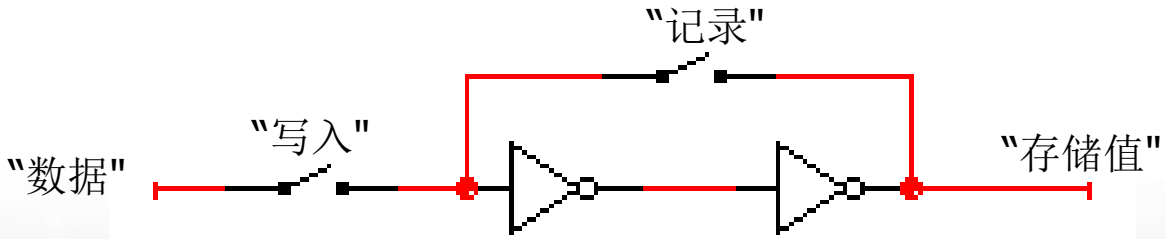
- 功能
  - (a) 正逻辑：输入同时为0，输出为1。任意输入为1，输出为0
  - (b) 0取反的功能

带反馈的简单电路

- 两个反相器形成一个静态记忆单元
  - 上电后永久保留固定值

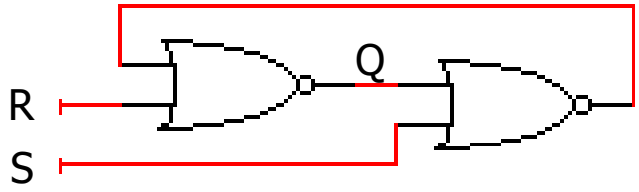


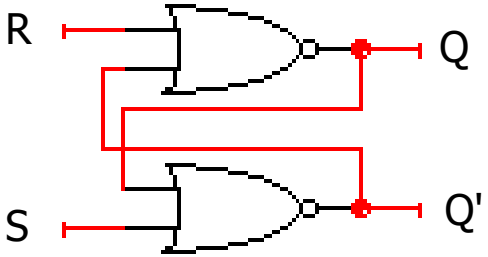
- 如何将新的值存入存储单元？
  - 有选择性的打断反馈路径



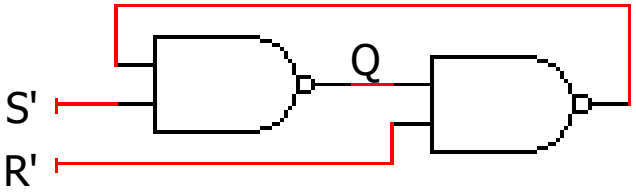
R-S锁存器

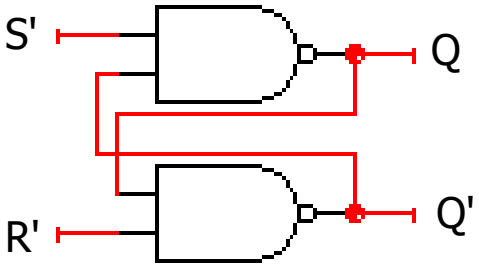
- 或非门(NOR)实现



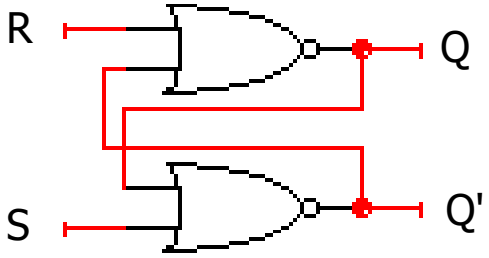


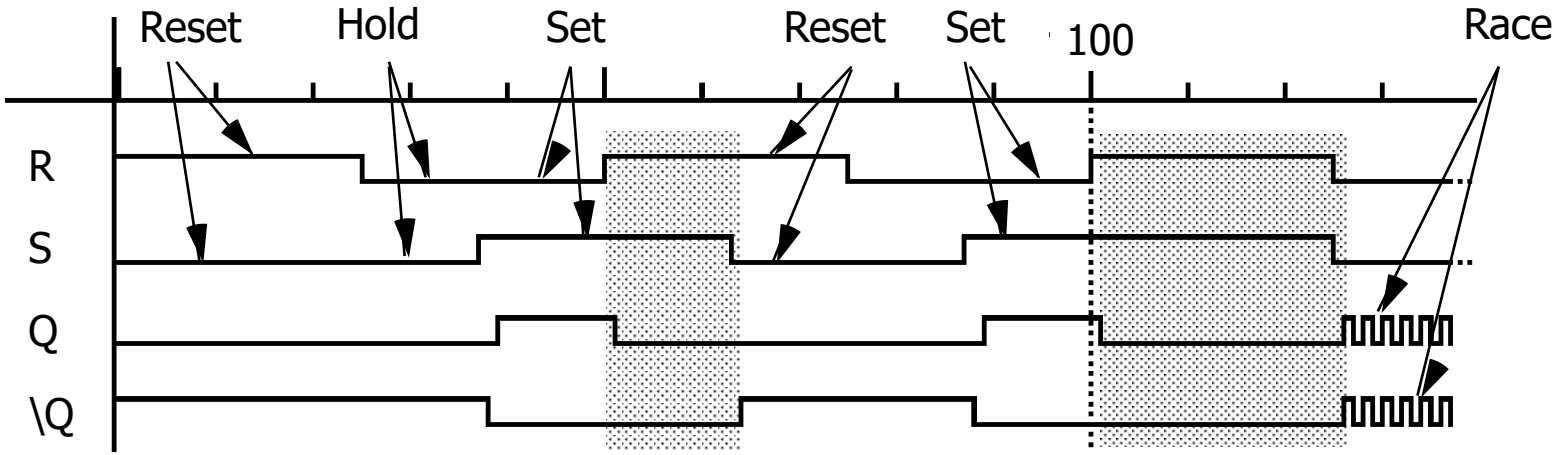
- 与非门(NAND)实现



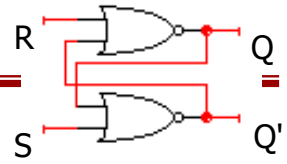


RS锁存器的时序行为Timing behavior



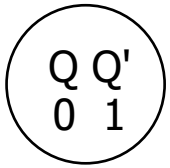


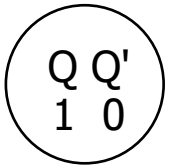
RS锁存器的状态行为

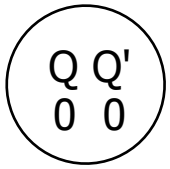


- RS锁存器行为的真值表

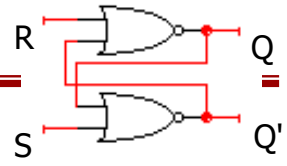
S	R	Q
0	0	hold
0	1	0
1	0	1
1	1	unstable



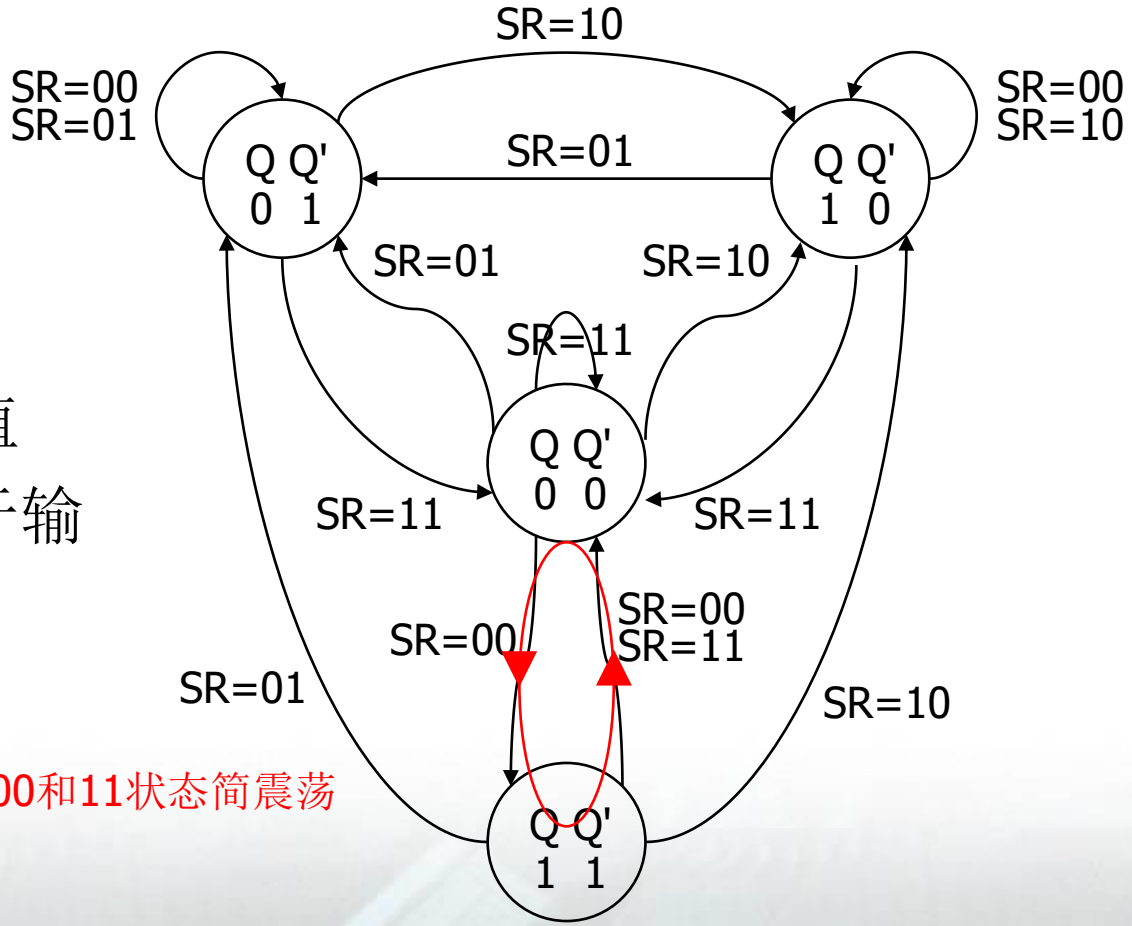




RS锁存器的理论行为

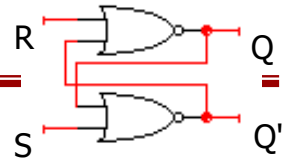


- 状态图
  - 状态：可能的值
  - 状态转换：基于输入改变

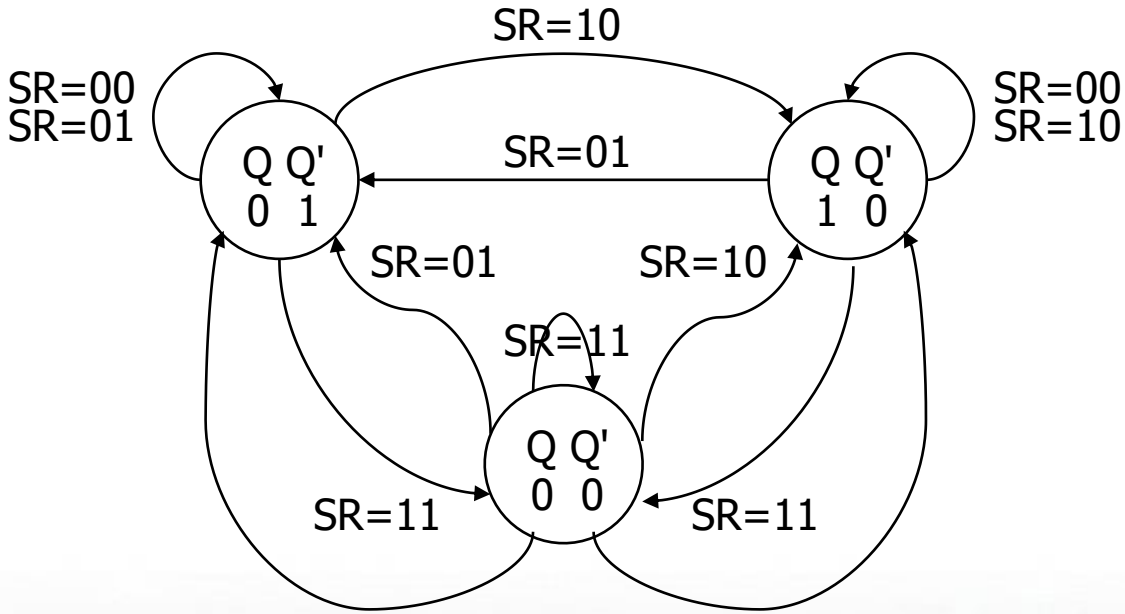


可能在00和11状态简震荡

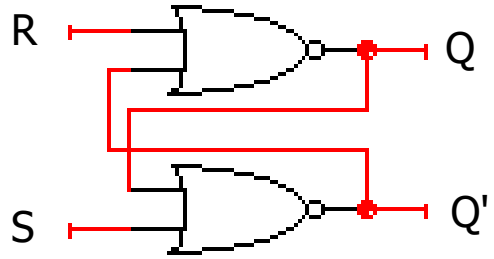
可观测的RS锁存器行为

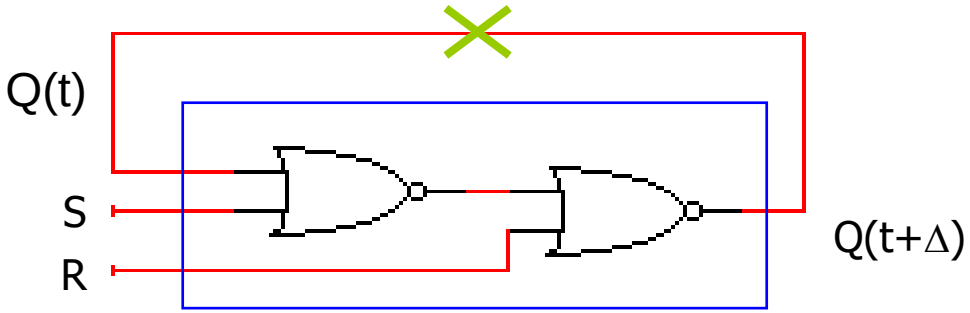


- 观测RS锁存器的11状态非常困难
  - 微观上通常R和S中的必定不同时变化
  - 不确定最终返回状态01还是10



R-S 锁存器分析





S	R	Q(t)	Q(t+Δ)	
0	0	0	0	hold
0	0	1	1	
0	1	0	0	reset
0	1	1	0	
1	0	0	1	set
1	0	1	1	
1	1	0	X	not allowed
1	1	1	X	

Q(t) \ S,R	00	01	11	10
0	0	0	X	1
1	1	0	X	1

特征方程：$Q(t+\Delta) = S + R' Q(t)$

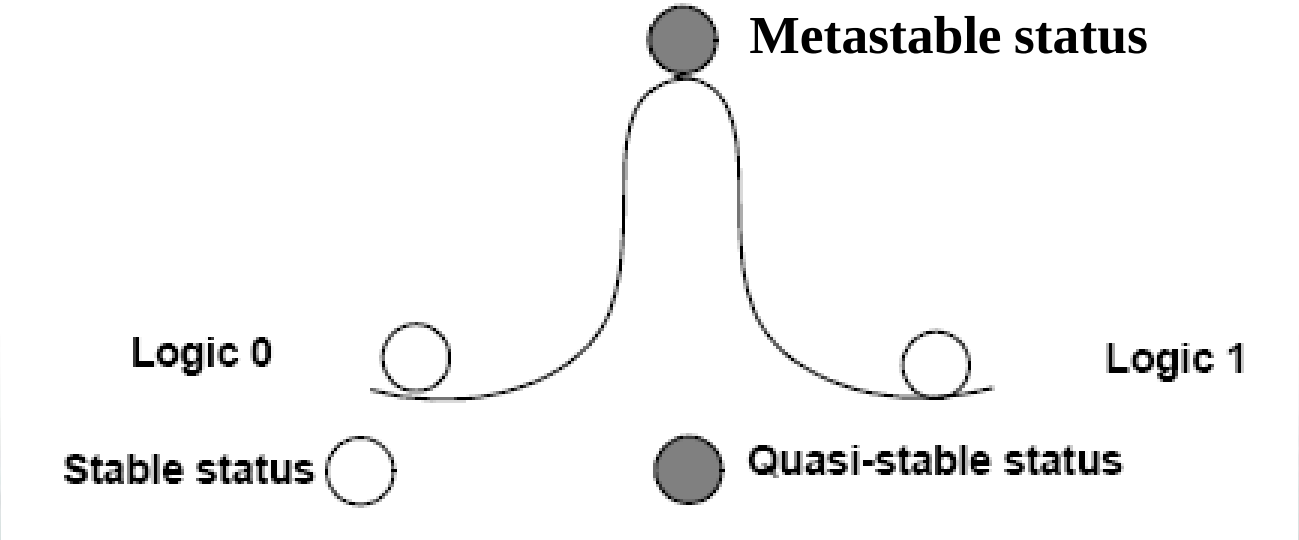
$Q^* = S + R'Q$

SR锁存器的激励输入限制

- 置位端S和复位端R不能同时变为无效
  - 产生信号追逐(race)
  - 输出将产生震荡(oscillate)
  - 最终必有一个门获胜，锁存器达到稳态，但是不能确定输出的结果。
  - 恢复时间(Recovery Time, t_{rec})，复位和置位有效信号间的最小时间
- 置位端S和复位端R的有效脉冲不能太短
  - 进入亚稳态(metastable)状态
  - 脉冲的宽度基本上要大于恢复时间t_{rec}
- 应该在设计电路时，十分注意。

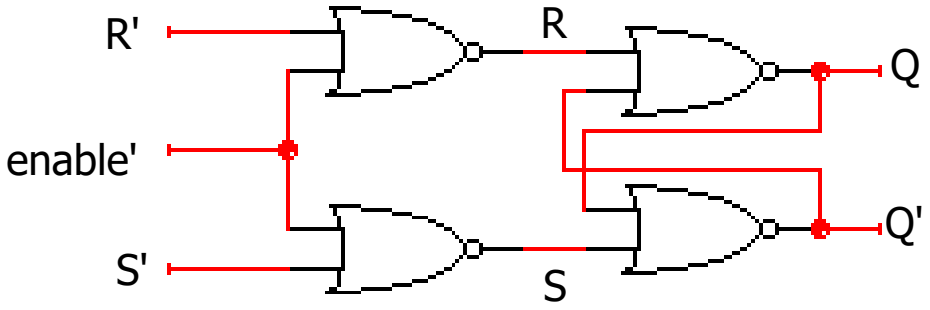
亚稳态(Metastable)

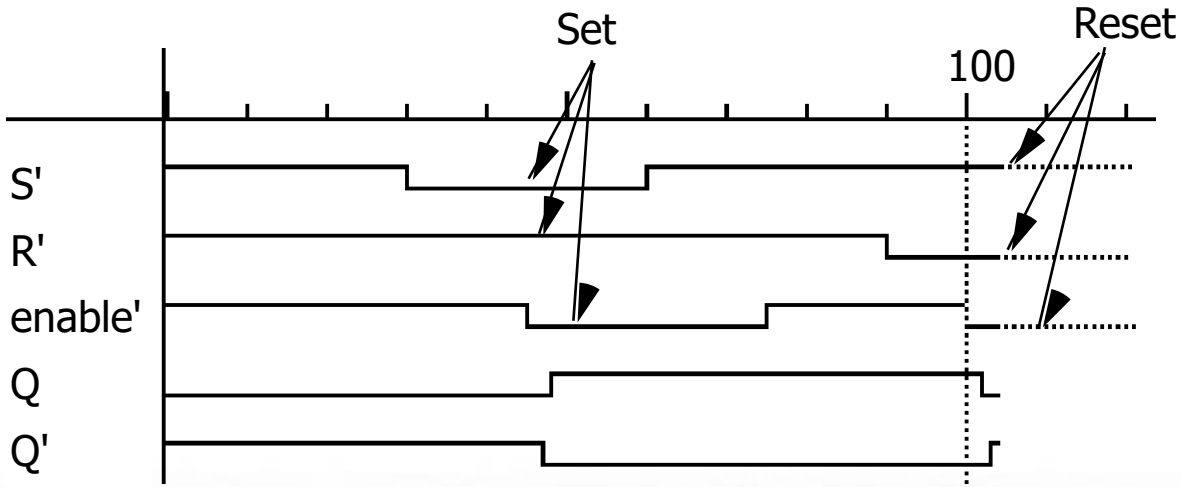
- 锁存器的两个稳态：逻辑0和逻辑1
- 亚稳态为两个稳态之外的第三个平衡的状态。
- 随着噪声的介入，会在比较长的时间内变为稳态，但是并不能确定稳态的类型。
- 亚稳态的危害：不同的门对相同的亚稳态信号地解释，逻辑0或者逻辑1并不一致。



门控RS锁存器（Gated R-S latch）

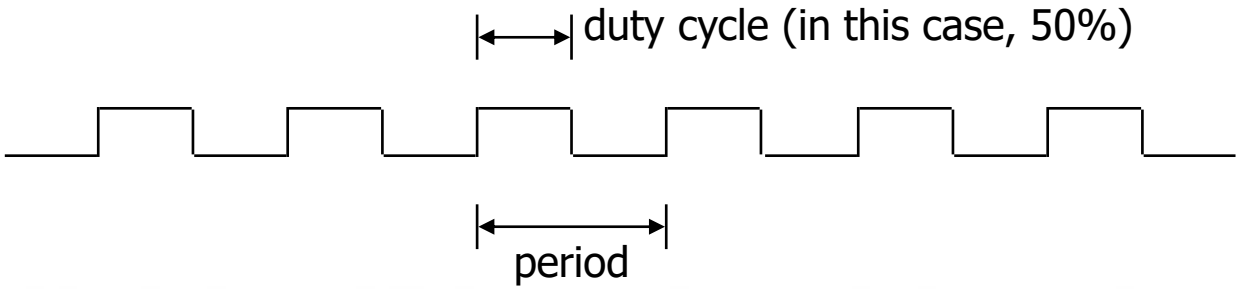
- 控制R和S输入信号是否对存储值产生影响





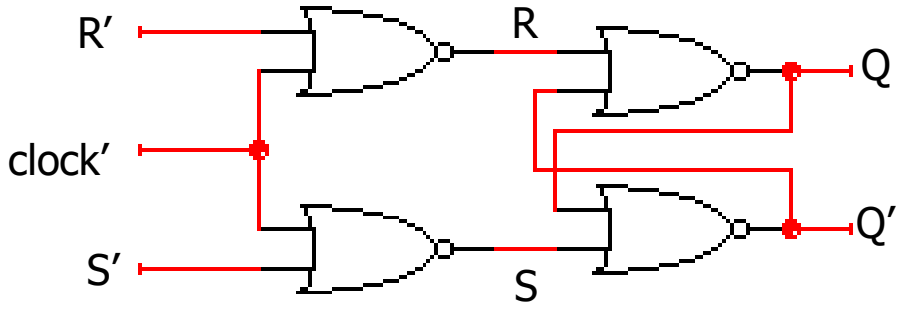
时钟 Clocks

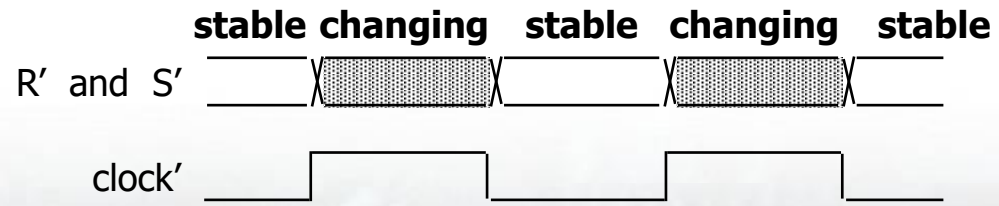
- 用来保证时间
  - 等待足够长的时间使输入信号 (R‘ and S’) 稳定
  - 然后再准许影响存储的值
- 时钟是规整的周期性信号
  - 时钟周期（period）：两个相同跳变的时间间隔
  - 占空比(duty-cycle)：高电平所占时钟周期的比例



时钟（续）

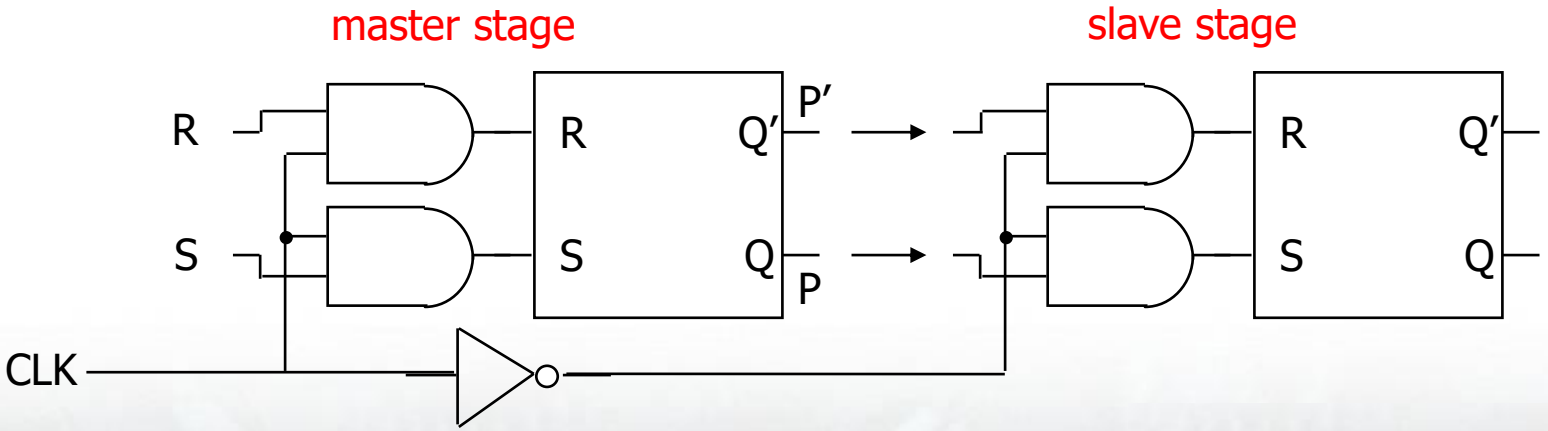
- 用时钟控制RS锁存器
  - 不让R和S在时钟信号有效时变化
  - 只有一半的时钟周期用来信号传播
  - 另一半的时钟周期输出（状态）保持不变





4. 主从触发器 (master-slave flip-flop)

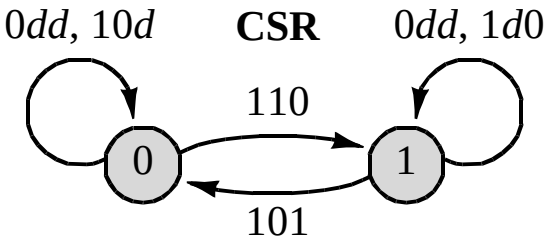
- 将锁存器级联：一个锁存器的输出连接另一个锁存器的输入
  - 用时钟信号的两个电平分别控制两个锁存器
  - 两倍的逻辑门（与锁存器相比）
  - 输出在相同的时钟边沿间（一个时钟周期内）一直保持稳定



钟控SR锁存器的激励表和特征方程

Enable inputs	Excitation inputs		Present state	Next state	
C	*S*	*R*	*Q*	*Q**	
0	×	×	0	0	Hold
0	×	×	1	1	
1	0	0	0	0	No change
1	0	0	1	1	
1	0	1	0	0	Reset
1	0	1	1	0	
1	1	0	0	1	Set
1	1	0	1	1	
1	1	1	0	×	Not allowed
1	1	1	1	×	

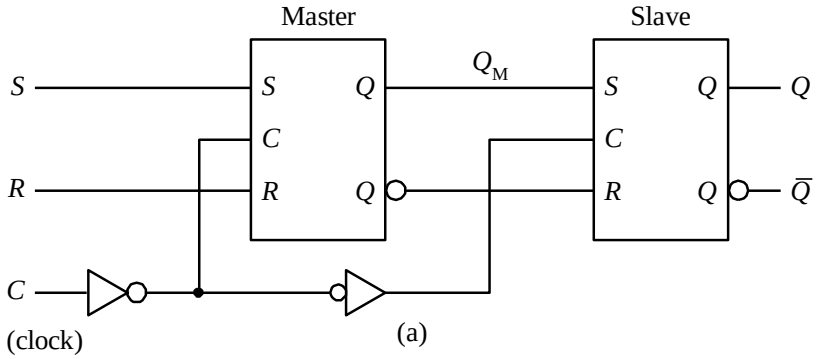
(a)



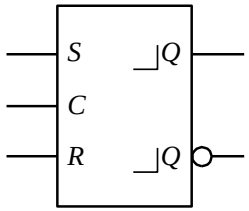
(b)

$$Q^* = SC + R'Q + C'Q$$

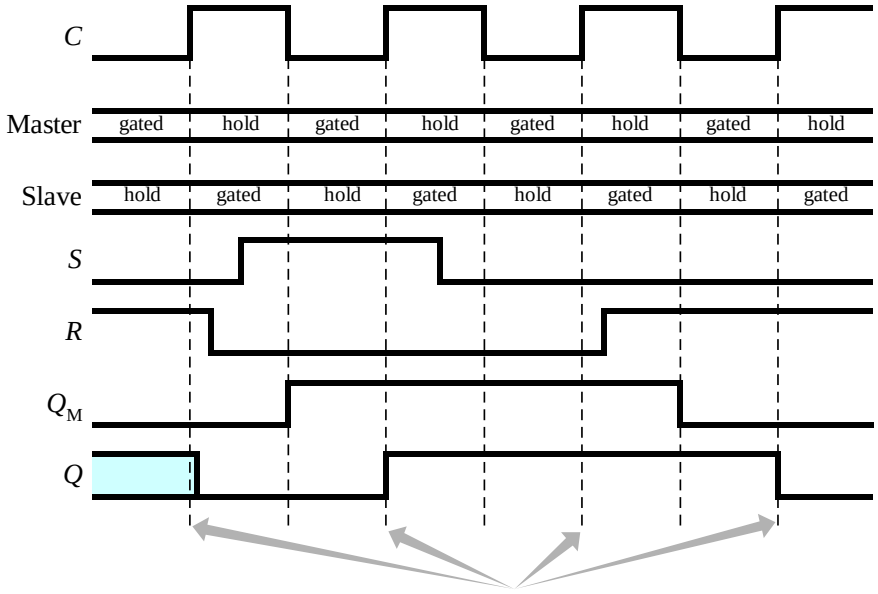
SR主从触发器（上升沿触发）



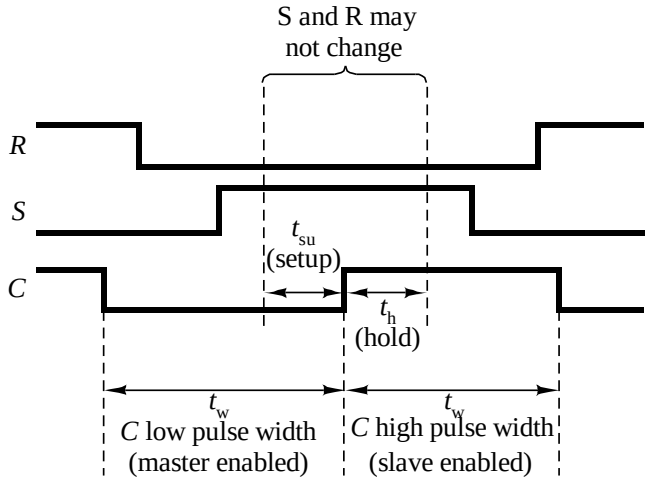
(a)



(b)



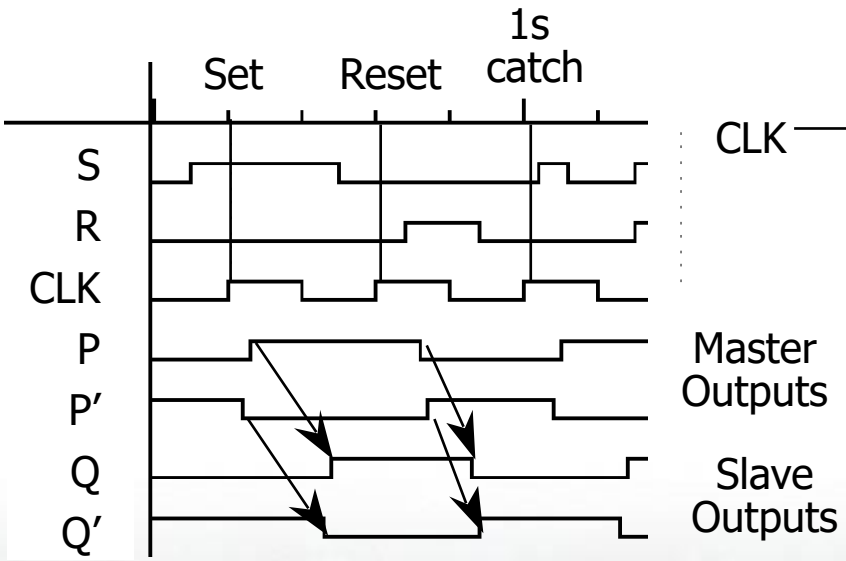
(c)

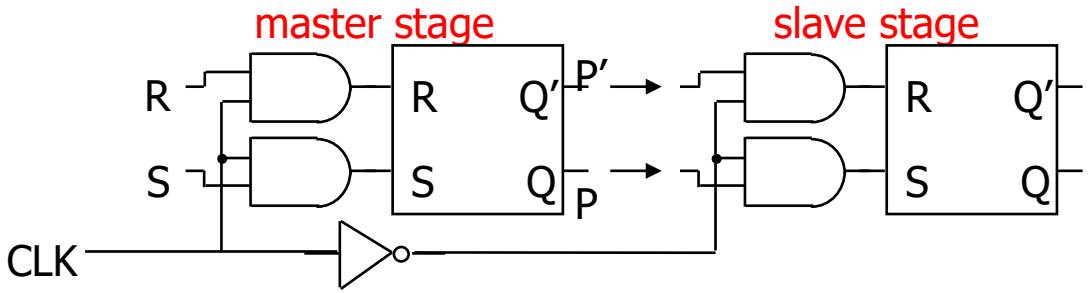


(d)

‘1’捕获问题（‘0’捕获问题）

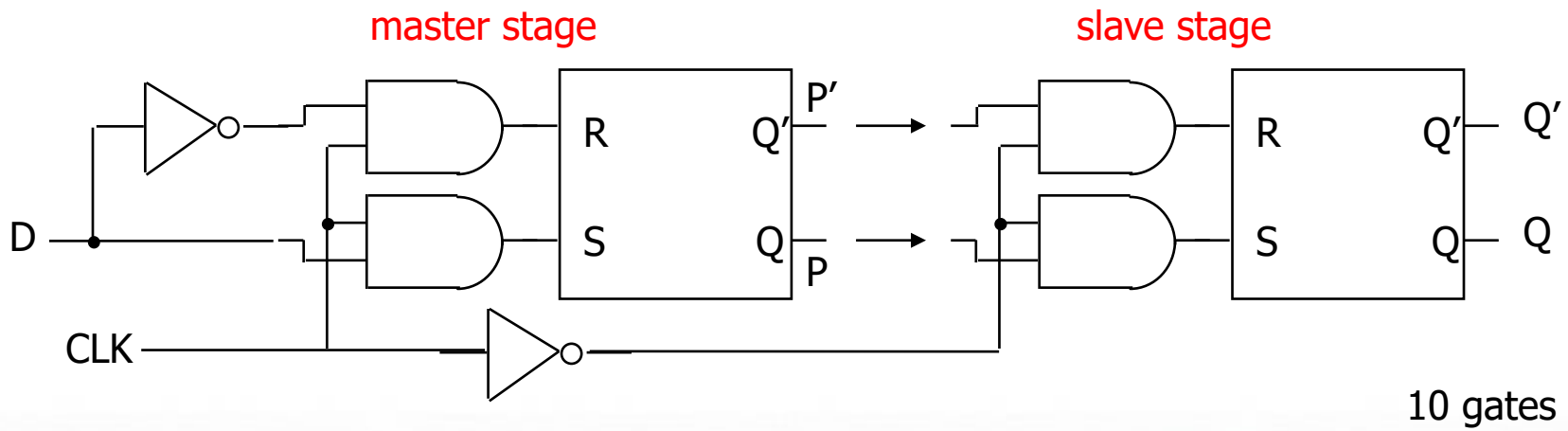
- 主从触发器的第一级RS锁存器
  - 如果时钟电平高时产生一个0静态冒险(static hazard)
  - 导致必须限制输入逻辑是无冒险的电路



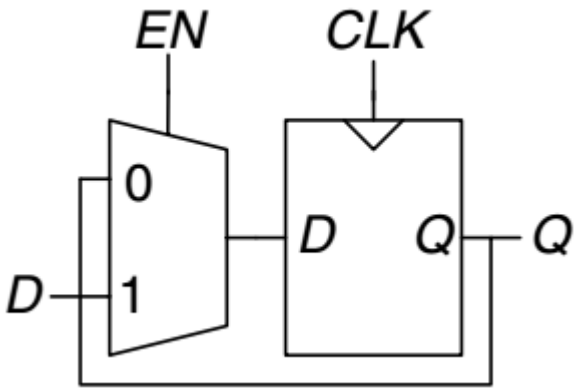


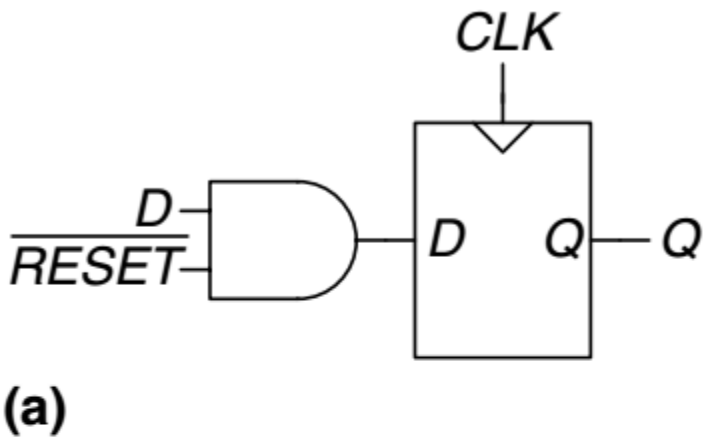
主从D 触发器 (下降沿触发)

- 让S和R互补
  - 可以消除1捕获问题
  - D的值需在时钟变低之前保证是要存储的值
  - D = S + R' Q

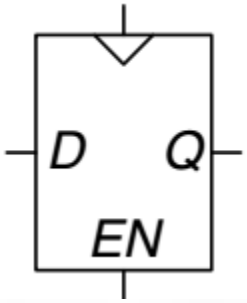


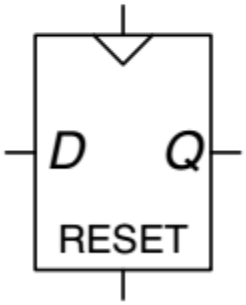
带使能或带同步清零D触发器



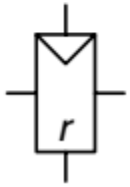


(a)



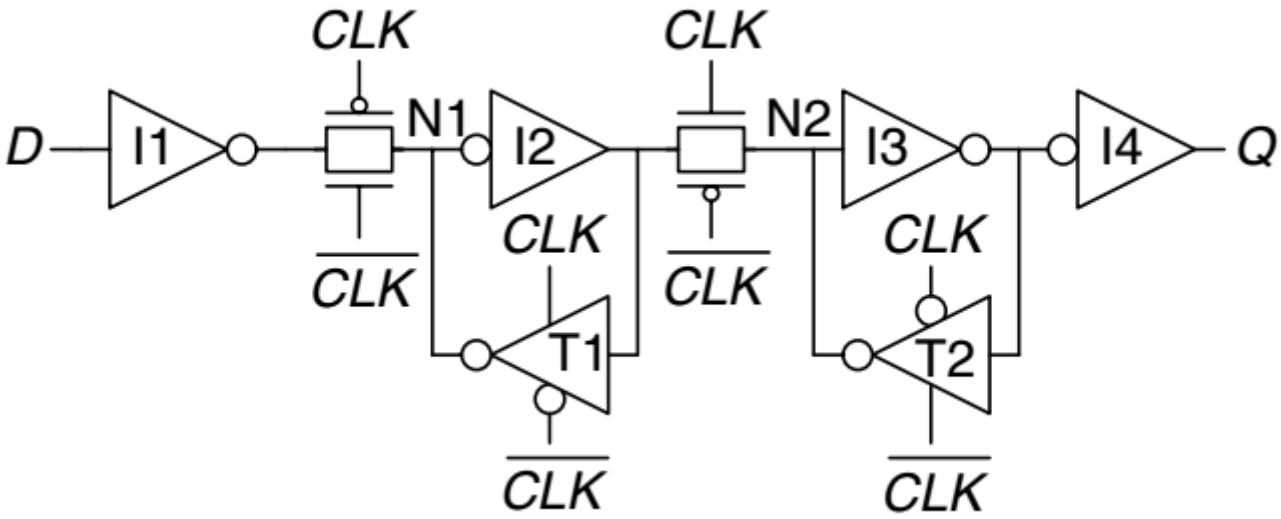


(b)



(c)

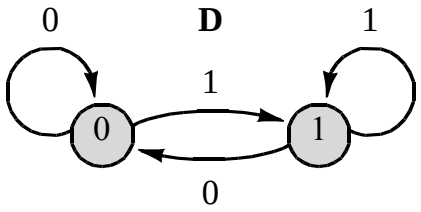
D触发器（CMOS）



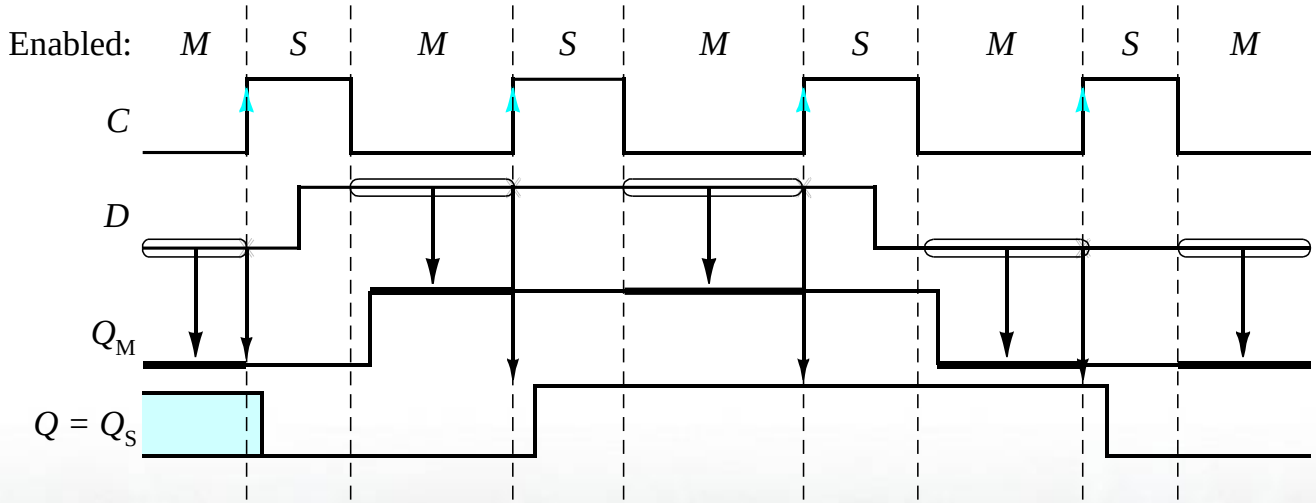
主从D触发器的特性

D	Q	C	Q*	
0	0	⊓	0	Store 0
0	1	⊓	0	
1	0	⊓	1	Store 1
1	1	⊓	1	

(a)



(b)



(c)

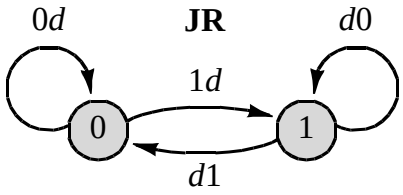
$$Q^* = D$$

5 JK触发器

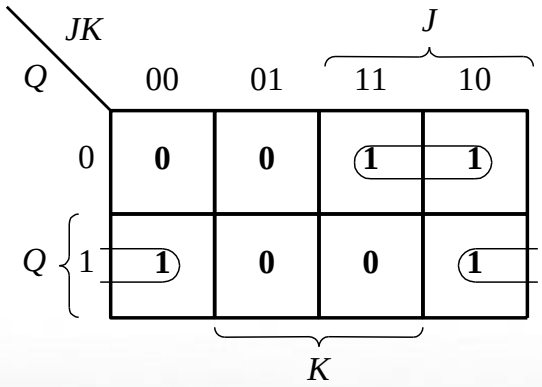
- 在SR锁存器和触发器中，S=1且R=1不被容许。
- 在JK触发器中，J = S，K = R，当J = 1 且K = 1时，JK触发器产生**状态翻转(*toggle*)**

J	*K*	*Q*	*C*	*Q**	
0	0	0	⎍	0	Hold
0	0	1	⎍	1	
0	1	0	⎍	0	Reset
0	1	1	⎍	0	
1	0	0	⎍	1	Set
1	0	1	⎍	1	
1	1	0	⎍	1	Toggle
1	1	1	⎍	0	

(a)



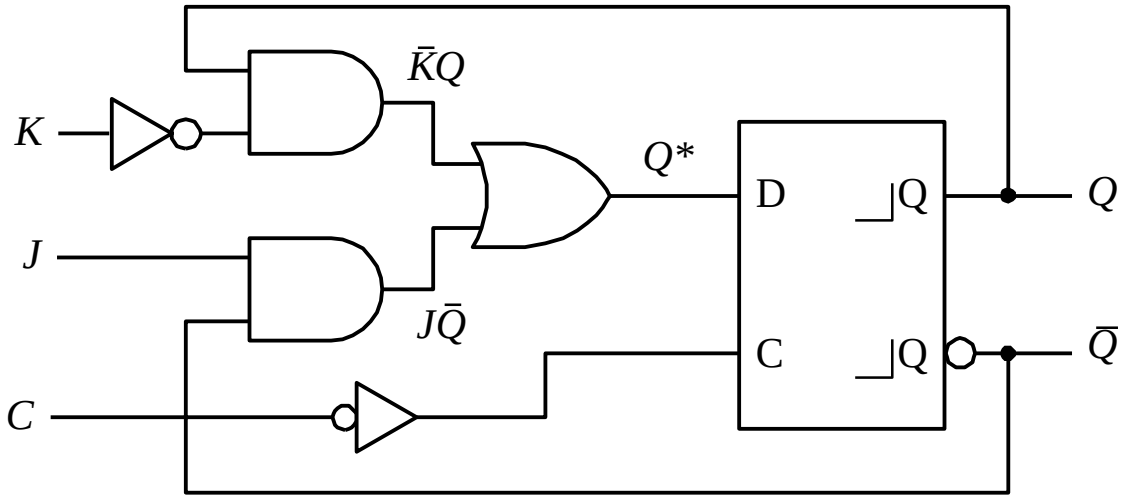
(b)



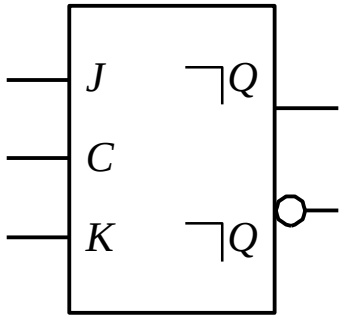
(c)

$$Q^* = K'Q + JQ'$$

主从JK触发器结构



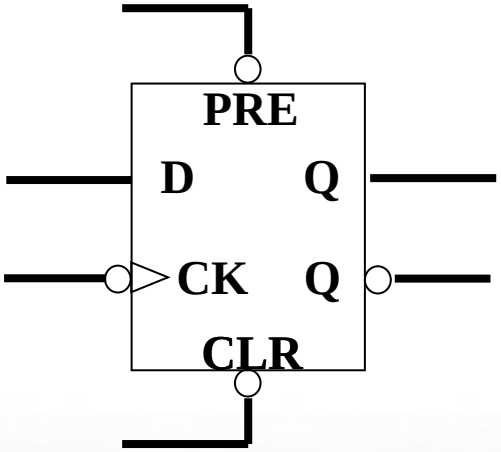
(a)



(b)

两个重要的概念

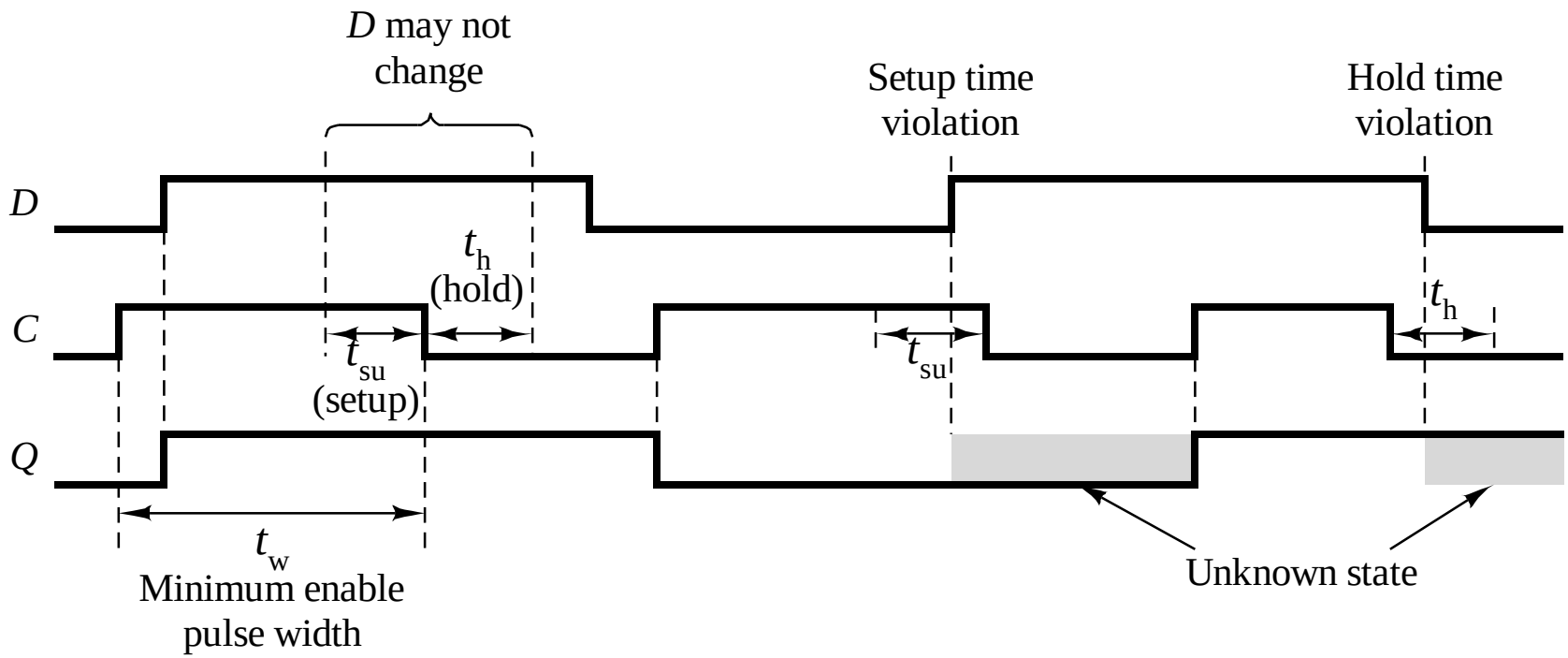
- 异步(asynchronous)信号
  - 在时序电路中，不用时钟参与就能改变电路状态的激励信号，如异步复位信号**clr**
- 同步(synchronous)信号
  - 必须在时钟参与下才能改变电路状态的激励信号



D触发器的时序限制

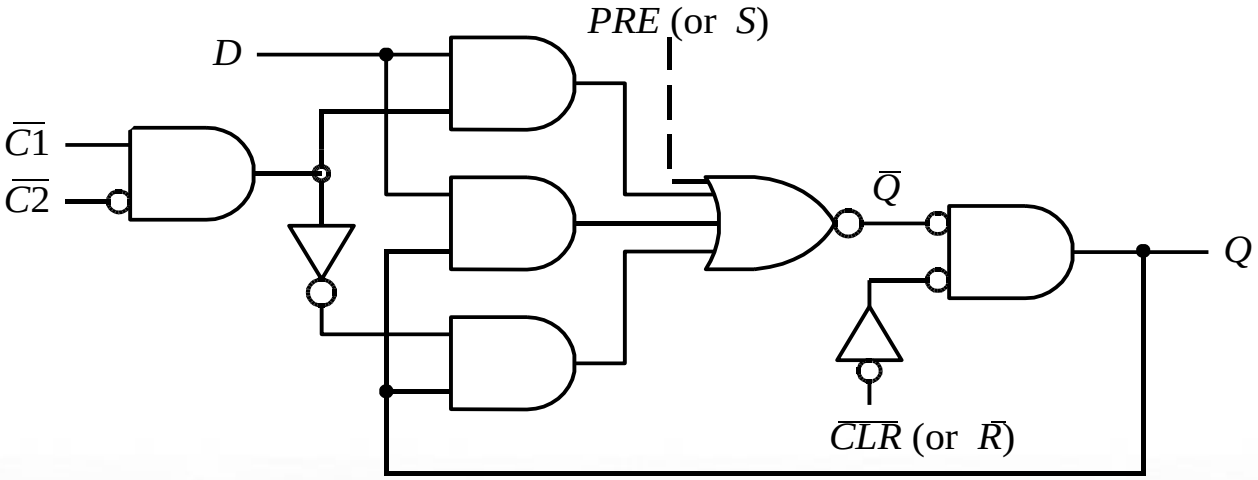
- 建立时间(*setup time*, t_{su})
  - 在使能信号变化**前**，激励信号必须保持的一段时间。
- 保持时间(*hold time*, t_h)
  - 在使能信号变化**后**，激励信号必须保持的一段时间。
- 最小脉冲宽度(t_w)：
  - 为保证状态的稳定，时钟信号需要的最小脉冲宽度。

下降沿D触发器的时序约束

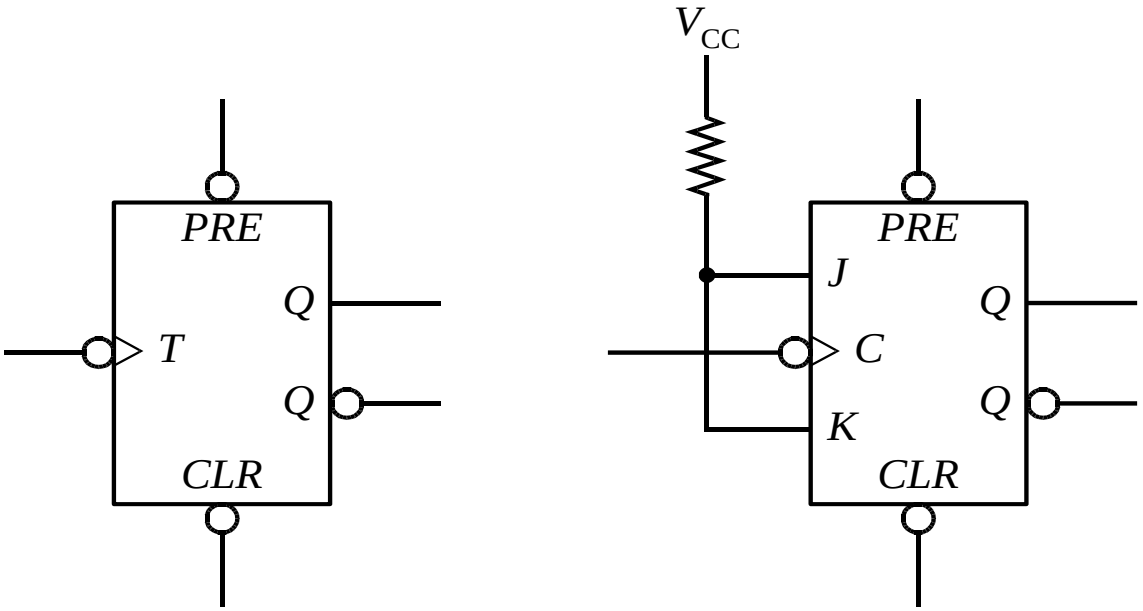


异步控制信号

- 上图中的$\overline{\text{PRE}}$和$\overline{\text{RST}}$
- 异步控制信号对状态的影响，不需要与时钟同步
- 需要注意最后一级锁存器的回复时间限制
- 74116 D Latch中的异步控制信号：



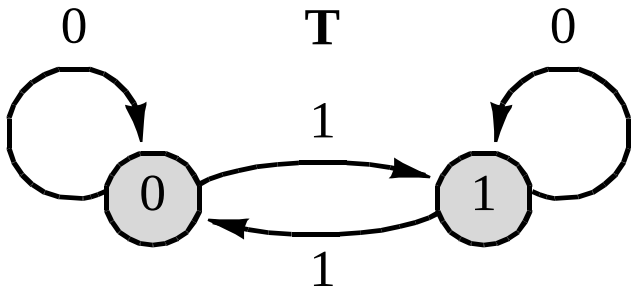
7. T触发器



T触发器特性

T	Q	Q^*	
↓	0	1	Toggle
↓	1	0	Toggle

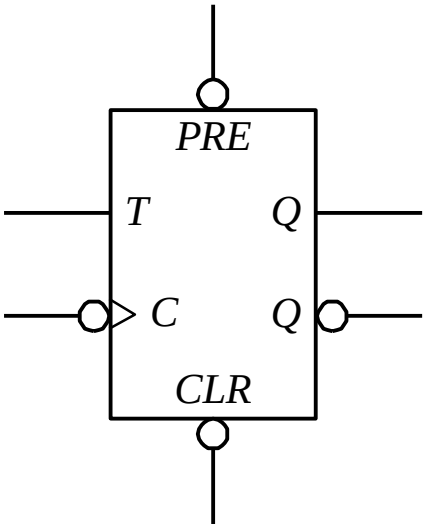
(a)



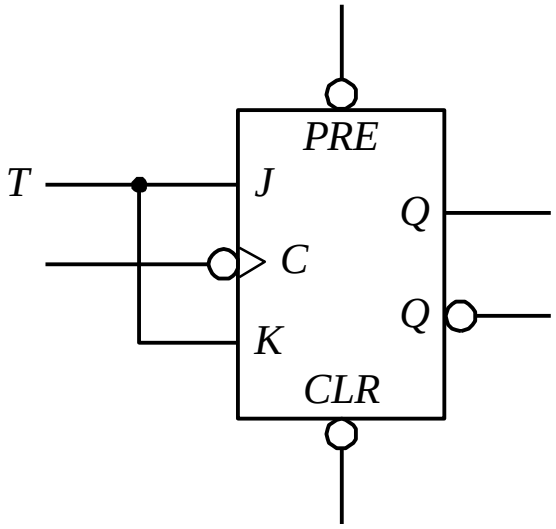
(b)

$$Q^* = Q'$$

钟控T触发器



(a)



(b)

钟控T触发器的激励表

T	Q	C	Q^*	
0	0	↓	0	Hold
0	1	↓	1	
1	0	↓	1	Toggle
1	1	↓	0	

$$Q^* = T'Q + TQ'$$

时序部件总结

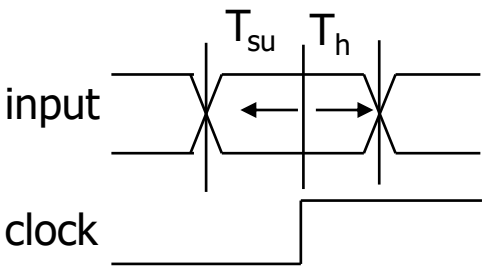
- 状态图、状态表、状态转换
- 锁存器
- 触发器
  - 脉冲触发器（主从触发器）
  - 边沿触发器
- 异步控制信号
  - 异步复位端
  - 异步置位端

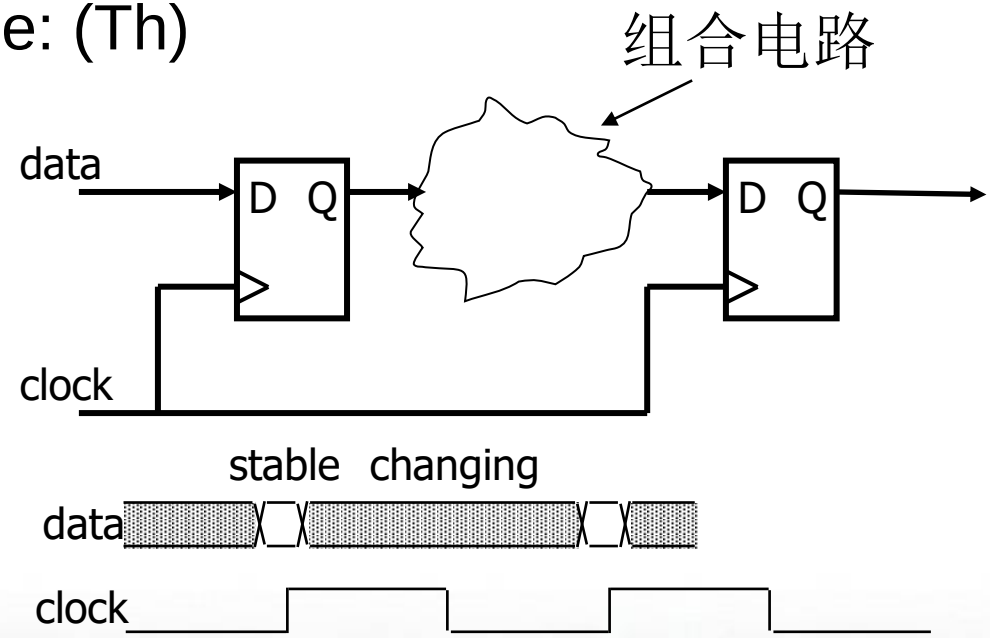
8. 时序电路的定时方法

- 连接器件和时钟的规则
  - 保证系统在最严格的情况下的正确运行
- 方法依赖于所采用的存储部件的性质
  - 主要集中讨论采用边沿触发器的系统
    - 在可编程器件中也能找到
  - 很多定制设计的系统也采用电平触发锁存器
- 正确定时的基本规则：
  - (1) 触发器的正确输入的正确时间
  - (2) 触发器不可能在同一时钟沿变化两次

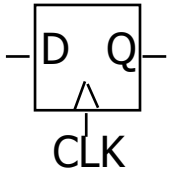
电路定时方法

- 定义
  - 时钟clock:
  - 建立时间setup time: (Tsu)
  - 保持时间hold time: (Th)

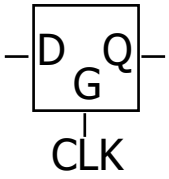




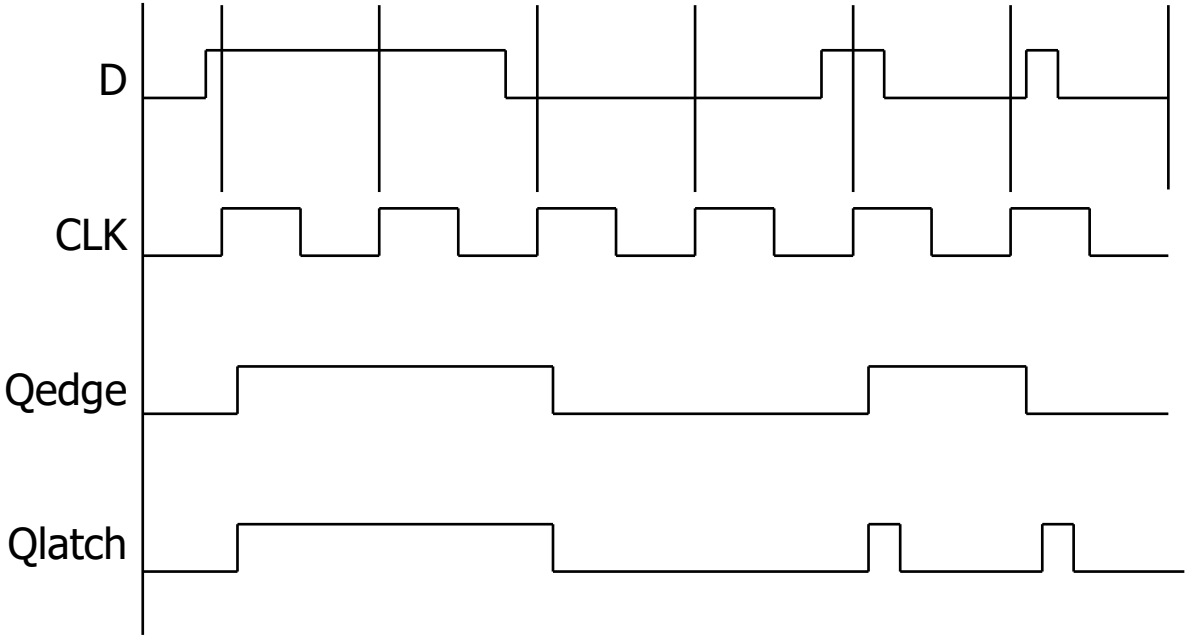
锁存器和触发器的比较



上升沿触发触发器

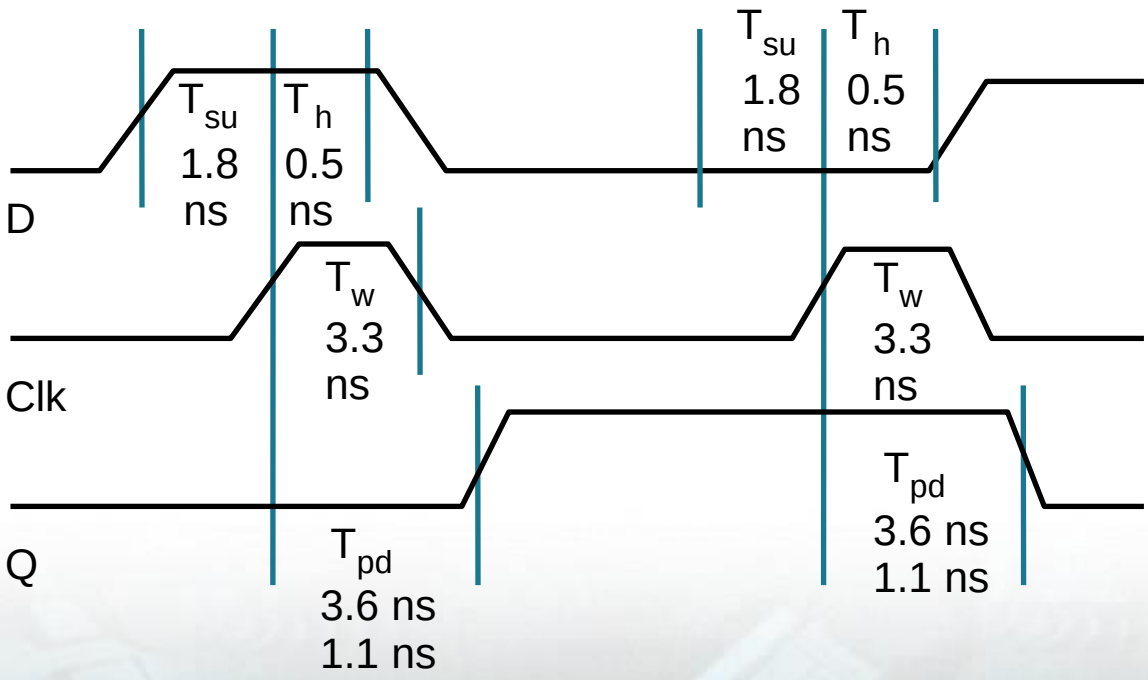


锁存器



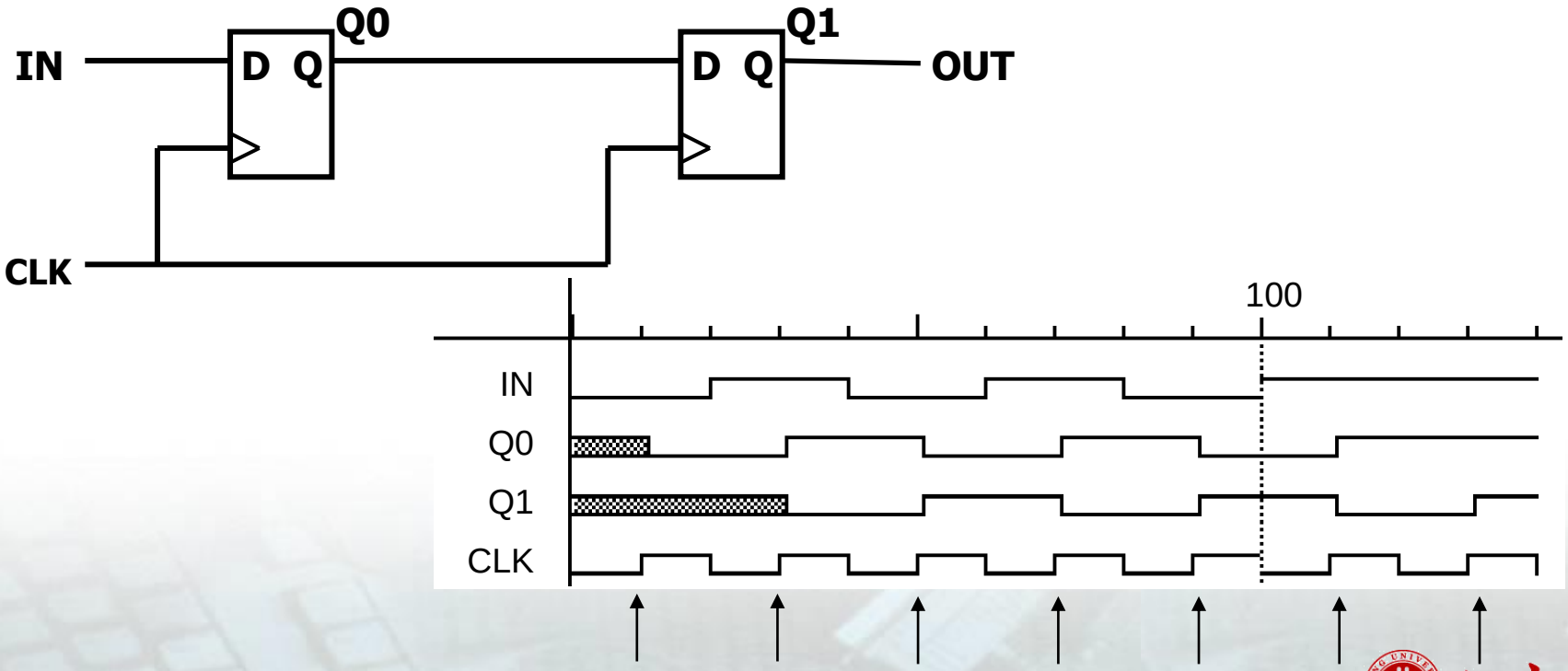
典型的时序参数

- 上升沿 D 触发器
  - 建立/保持时间(setup/hold time)
  - 最小时钟宽度(minimum clock width)
  - 传播延迟propagation delays (low to high, high to low, max and typical)



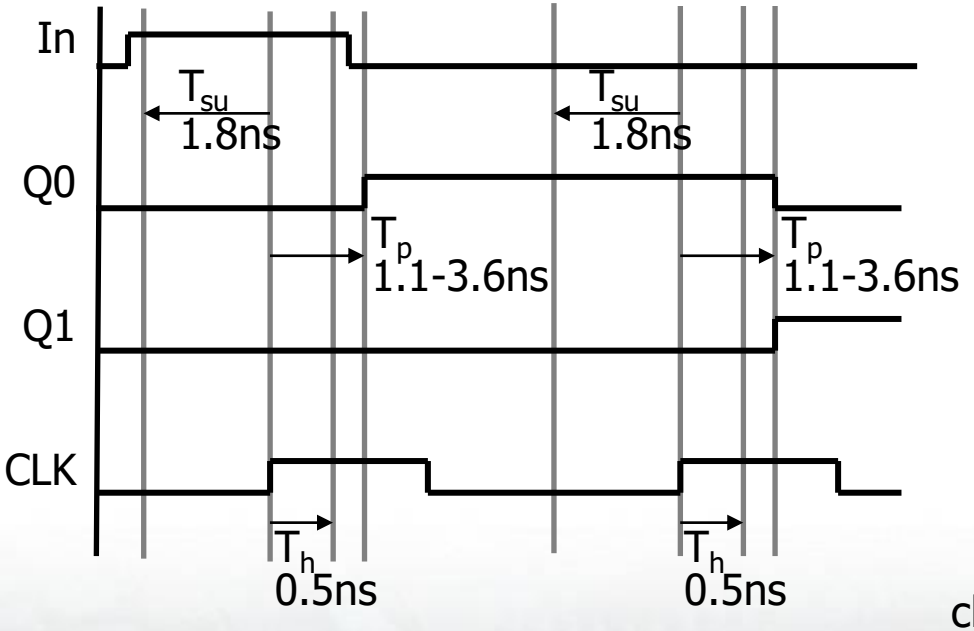
级联边沿触发器

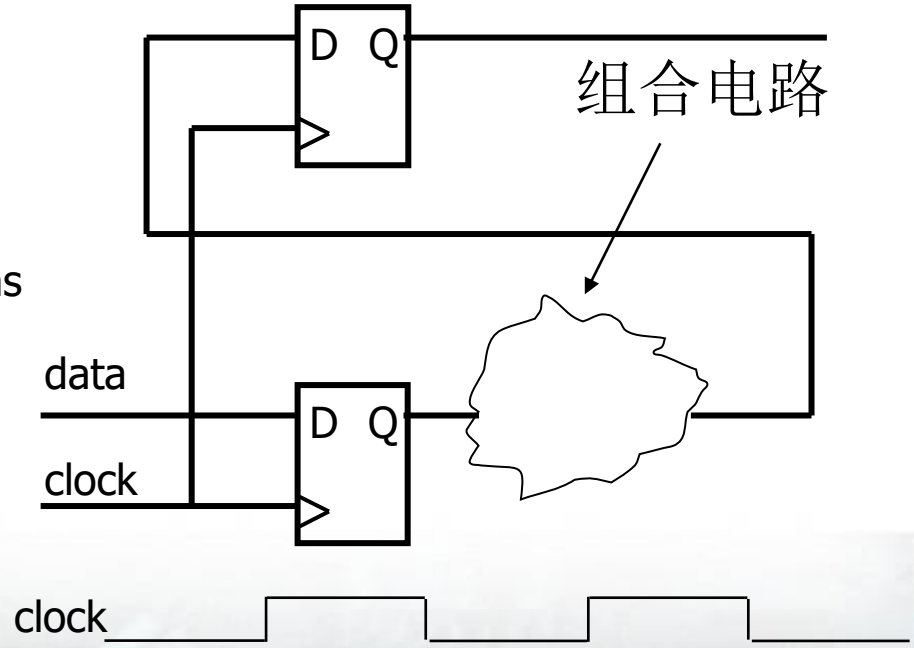
- 移位寄存器
  - 新的输入值进入第一级
  - 同时以前的值进入第二级
  - 考虑建立时间/保持时间/延迟时间



级联边沿触发器

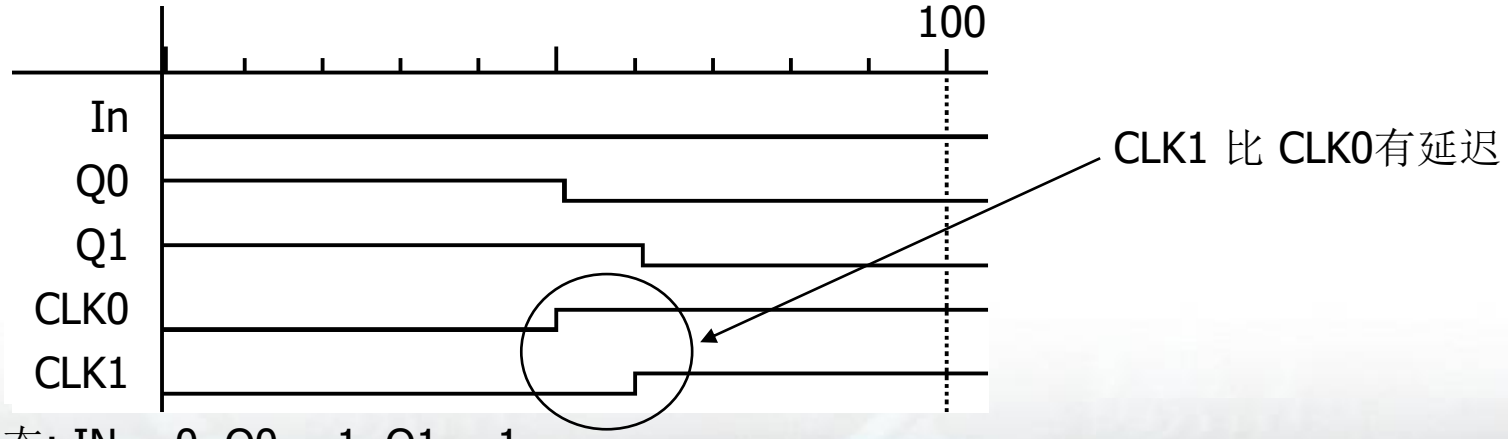
- 为什么可以工作
  - 延迟时间大于保持时间 $T_p > T_h$
  - 时钟周期大于于建立时间和延迟时间 $T_{clock} > T_p + T_{su}$
  - 可以保证后面的触发器锁存正确的值





时钟扭斜(clock skew)

- 问题
  - 正确的电路行为假设所有存储部件的下一个状态在同一个时刻由所有的存储部件测定
  - 在高性能系统中是很难的达到的，因为到达所有触发器的时钟延迟和逻辑延迟是可比的
  - 级联触发器的扭斜影响:
  - $T_{pd} + T_p > T_h + T_{skew}$;($T_{skew}$>0)
  - $T_{clock} + T_{shew} > T_{pd} + T_p + T_{su}$;($T_{skew}$<0)



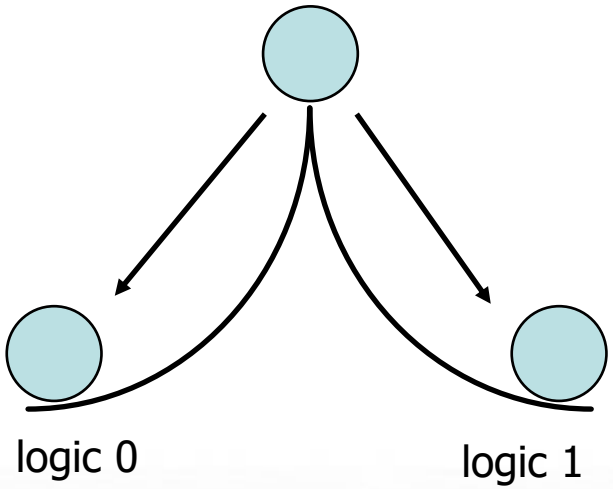
原始状态: IN = 0, Q0 = 1, Q1 = 1
因为扭斜，下一个状态变为Q0 = 0, Q1 = 0而不是 Q0 = 0, Q1 = 1

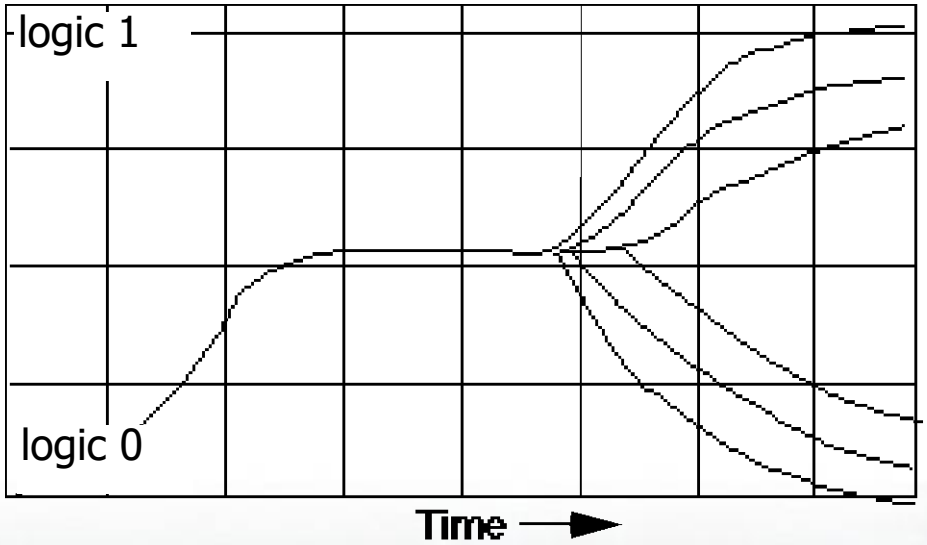
亚稳态(Metastability)和异步输入

- 时钟同步电路
  - 输入、状态和输出的采样或者变化与共同的时钟信号相关
  - 例如主从、边沿触发器
- 异步电路
  - 输入、状态和输出的采样或者变化与共同的时钟信号不相关 （毛刺glitch/冒险hazard）
  - SR锁存器
- 同步电路的异步输入信号
  - 输入可能在任何时刻变化，会造成setup/hold时序违例
  - 十分危险，同步输入是最好的
  - 不可能避免（如复位信号，用户输入，不同时钟域信号等）

同步失败

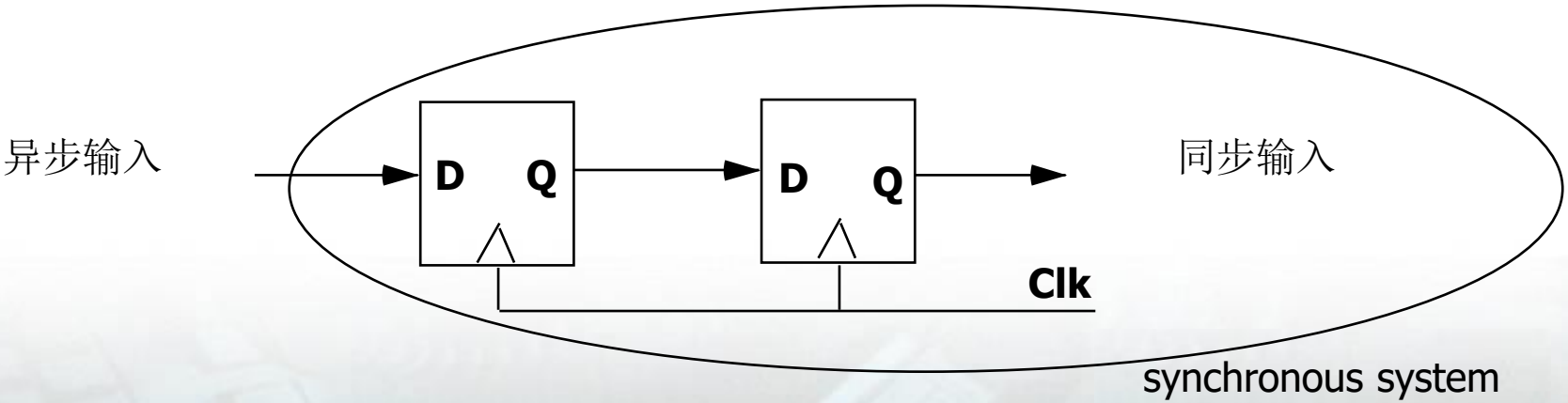
- 发生在触发器的输入在时钟沿附近变化
  - 触发器进入亚稳态状态：既不是0也不是1状态
  - 可能在亚稳态状态持续不确定的时间
  - 可能不经常出现，但是有一定的概率





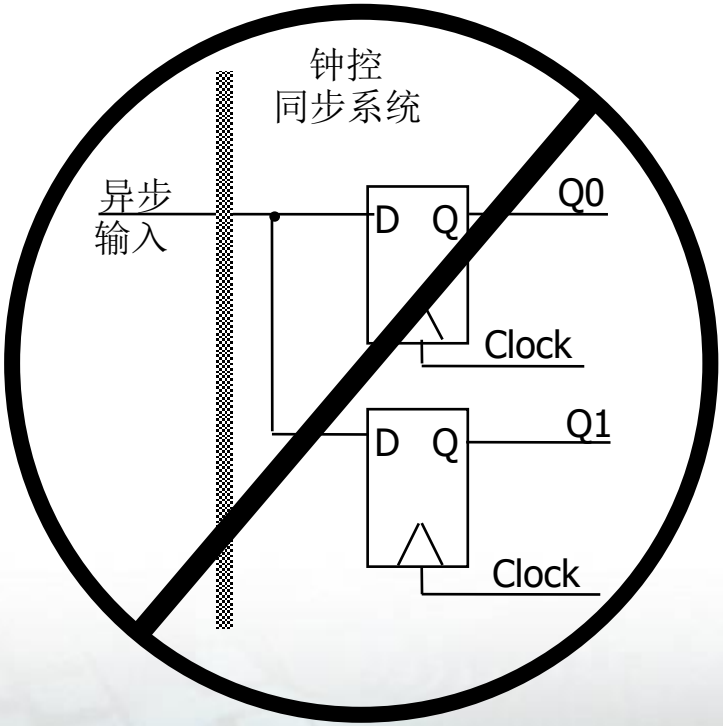
处理同步失败

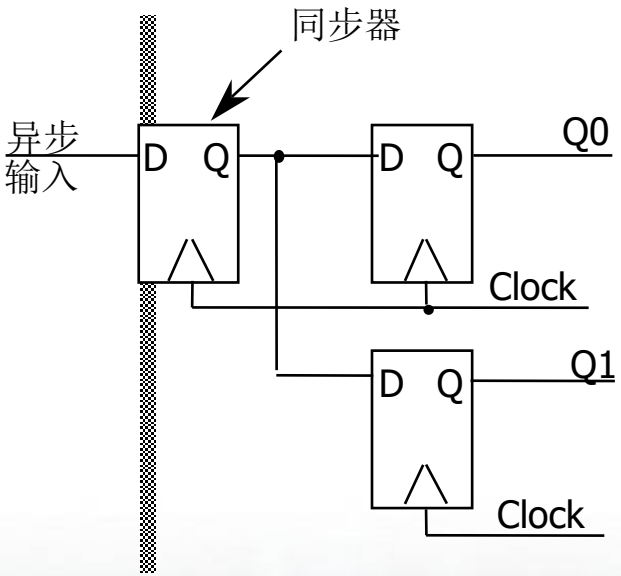
- 同步失败的概率不可能减小到0，但是可以减小的
  - (1) 减慢系统时钟
    这给同步器更多的时间衰变到稳定状态；
    同步器失效成为高速电路的一个大问题
  - (2) 采用最快的逻辑工艺实现同步器
    这可是使“山峰”尽可能的窄来快速打破平衡
  - (3) 级联两个同步器
    这可以非常有效的解决同步问题



处理异步输入

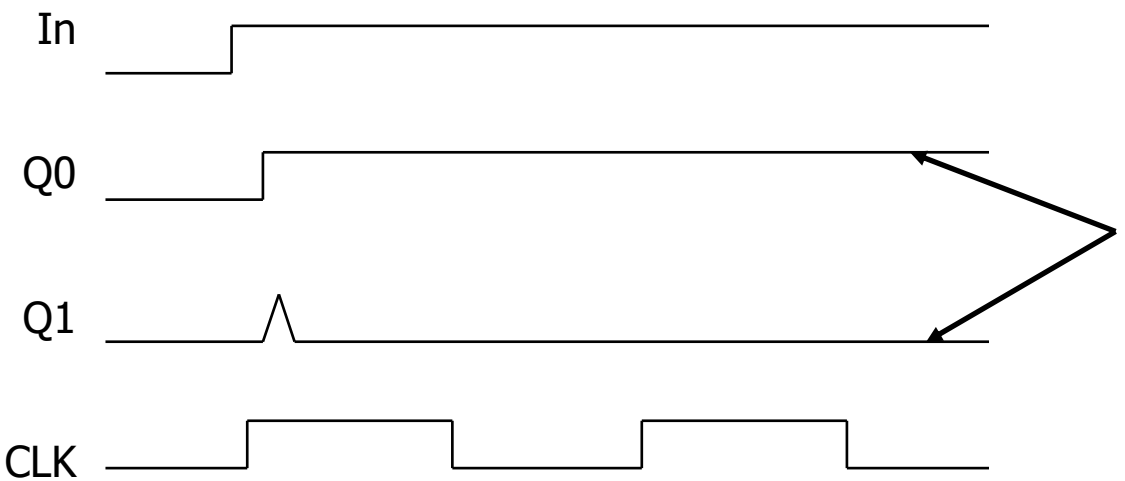
- 一定不要将异步输入驱动多于一个触发器
  - 尽可能同步异步信号然后再同步使用





处理异步输入

- 会产生什么错？
  - 输入变化太靠近时钟沿（保持时间违例）



In是异步信号驱动D0和D1

一个触发器寄存到信号

另一个触发器没有寄存到信号

产生不一致的状态

异步握手协议

图 6.42　独立钟控子系统

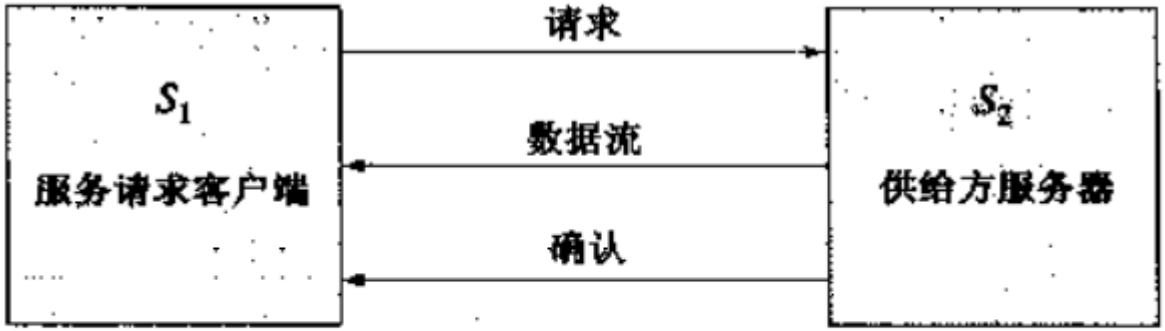


图 6.43　请求/确认信令

异步握手协议

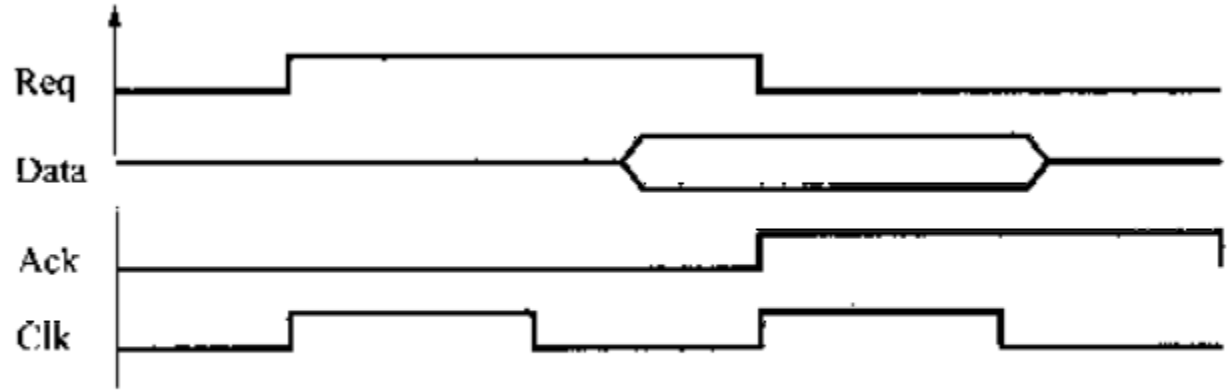
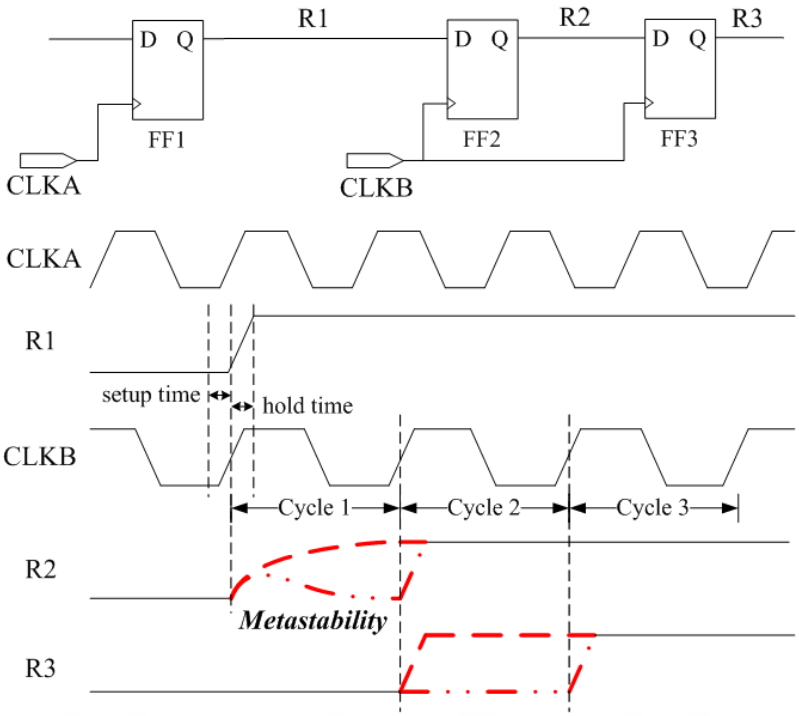


图 6.44　同步请求/确认信令

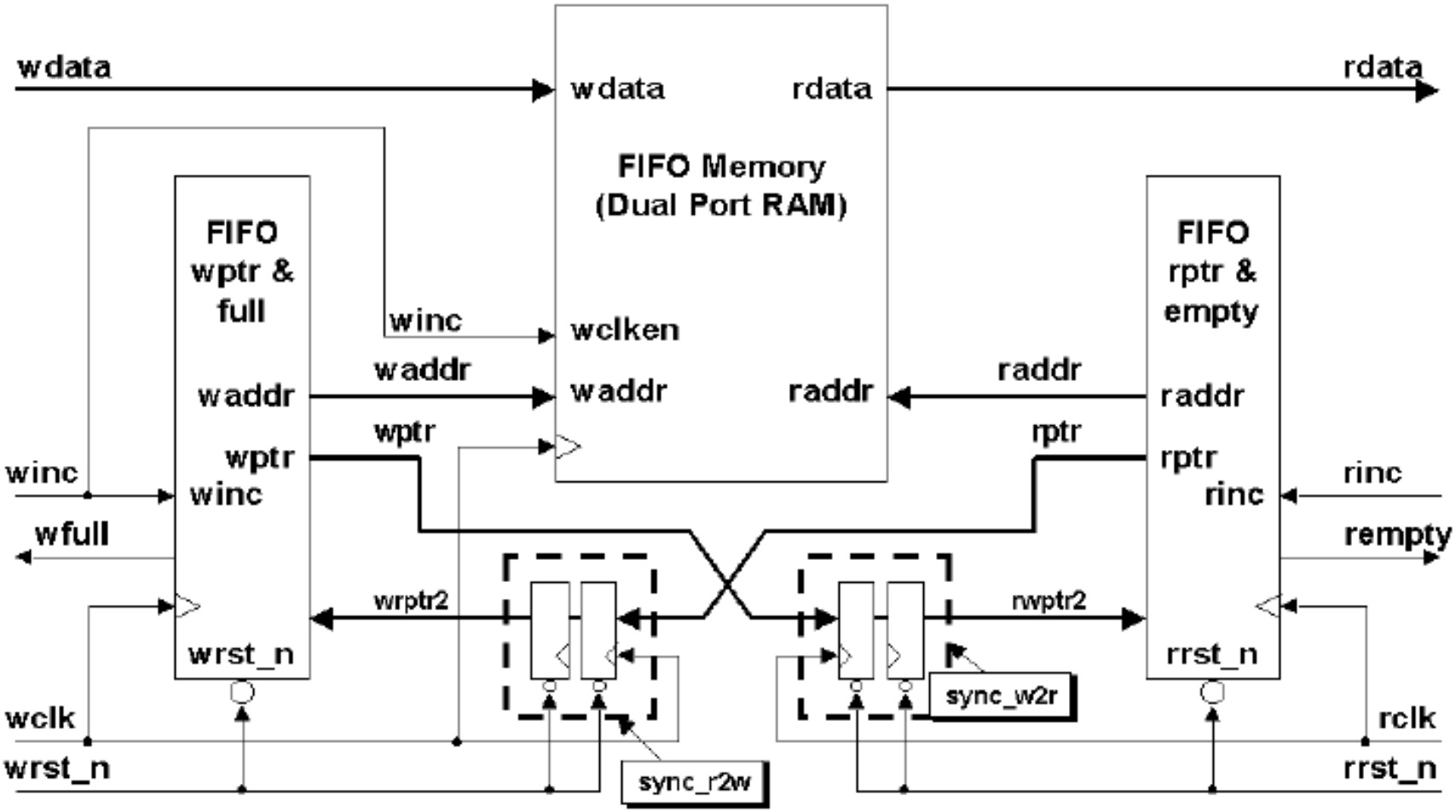
图 6.45　带有等待信令的同步请求

跨时钟域设计

- 跨时钟域设计及亚稳态现象
  - 亚稳定状态
    - 寄存器的建立或保持时间违例（Setup/Hold Timing Violation）将引起该寄存器的输出端进入亚稳定状态
  - 同步器（Synchronizer）
    - 由两级寄存器组成的同步器可以消除亚稳态现象
    - 但同步器输出端体现输入变化的确切时钟周期不可预测
    - 上述不确定性是跨时钟域设计中功能错误的本质原因
  - 异步计数器设计——格雷码计数器



异步FIFO——格雷码计数器和异步握手



3. 计数器

- 计数器的主要功能：
  - 纪录输入脉冲的个数
- 应用：
  - 计时设备
  - 控制电路
  - 信号产生器
- 分类：
  - 二进制/非二进制
  - 异步/同步
  - Up/Down

计数器的控制信号

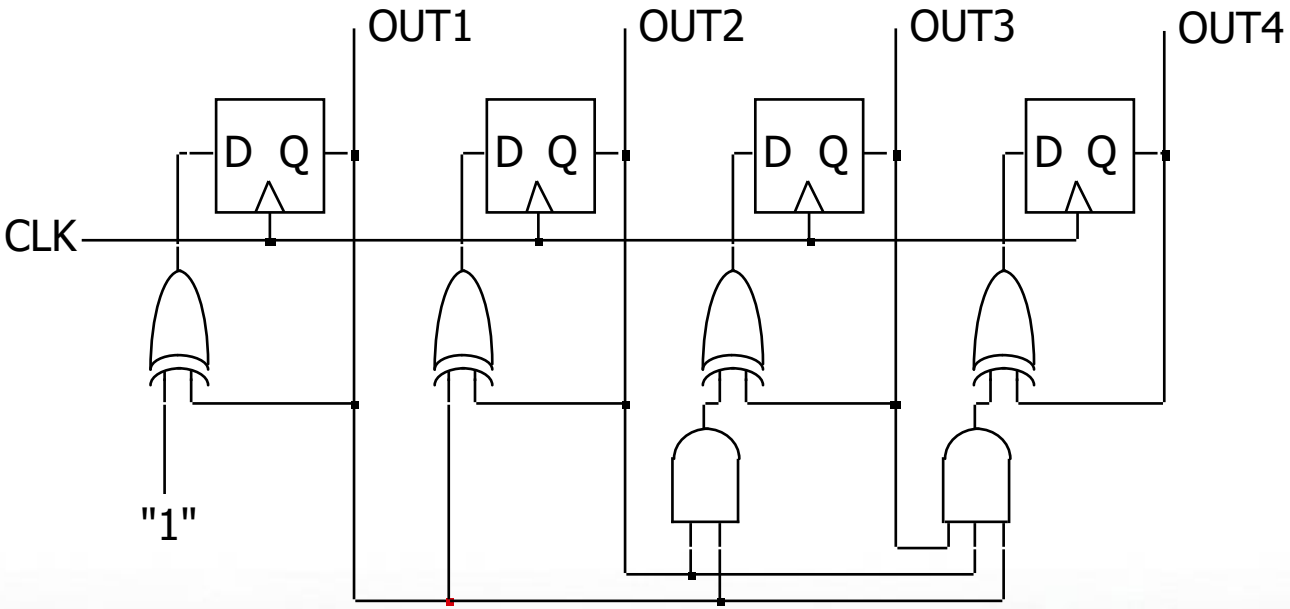
- 同步/异步清‘0’clear
- 使能enable
- 同步异步装载load

同步二进制计数器

- *n*位的同步二进制计数器组成：
  - *n*个钟控的JK触发器
  - 初始状态全‘0’
  - 状态：0, 1, 2, …, 2^n-1, 0
  - 溢出信号
  - 行波进位链

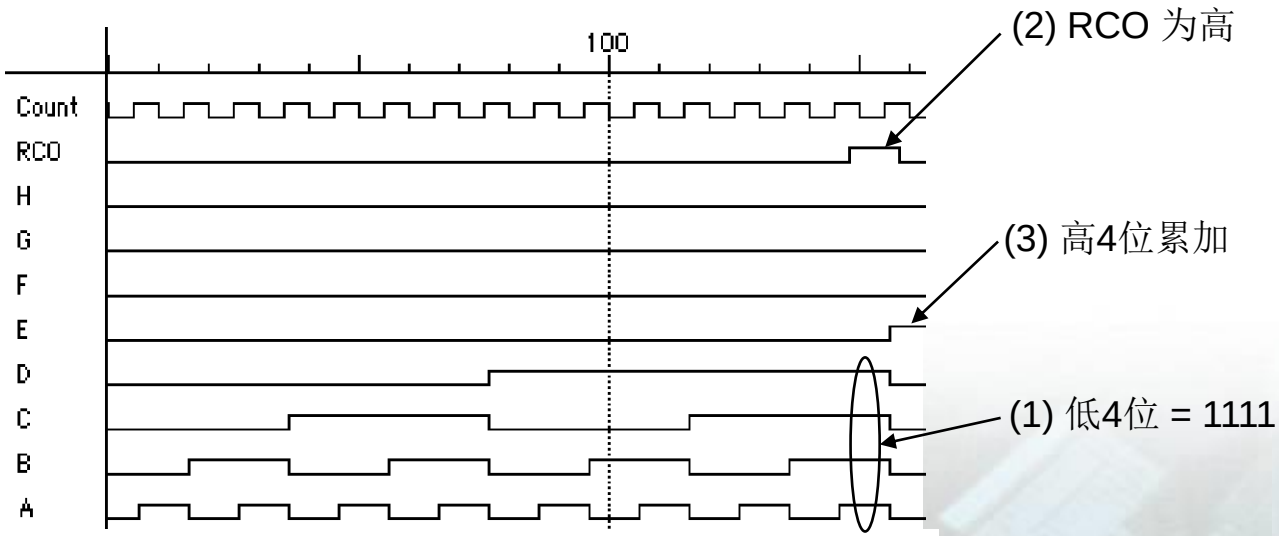
简单二进制计数器

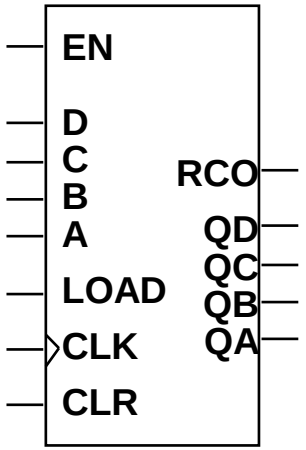
- 寄存器的逻辑（不采用多选器）
  - XOR门决定那一位什么时候反转
  - 低位全部为‘1’时



4位同步计数器

- 许多应用采用的标准模块
  - 上升沿触发器同步装载load和清除clear输入
  - 并行装载数据 D, C, B, A
  - 使能输入enable
  - RCO：形波进位输出可用于级联计数器
    - 高high：当计数器为1111
    - 与门实现

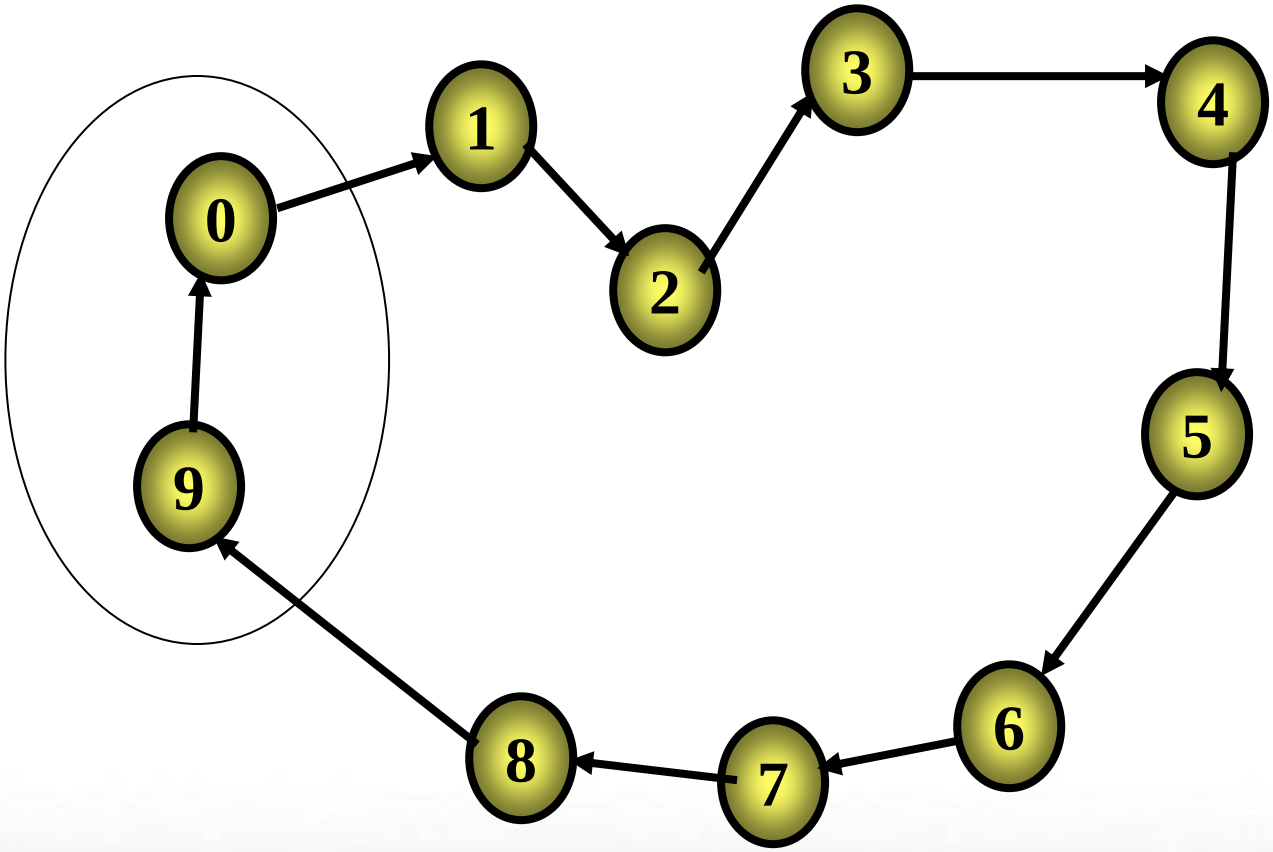




模*N*计数器

- 实际的数字系统设计中需要从0数到*N*-1
- 实现模*N*计数器的要求：
  - 计数器
  - *N*-1到0的翻转
  - 需要设计判断N-1的逻辑电路
- 计数器位数*n*
  - $2^{n-1} < N \leq 2^n$
  - 二进制计数器：模2^n计数器
- 分类：同步、异步

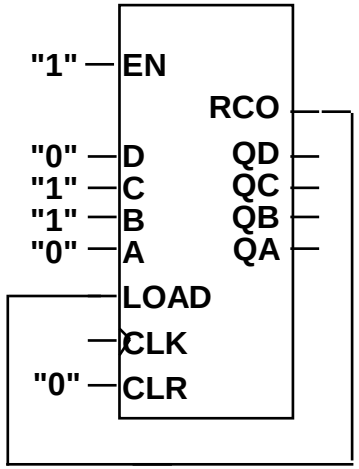
BCD计数器



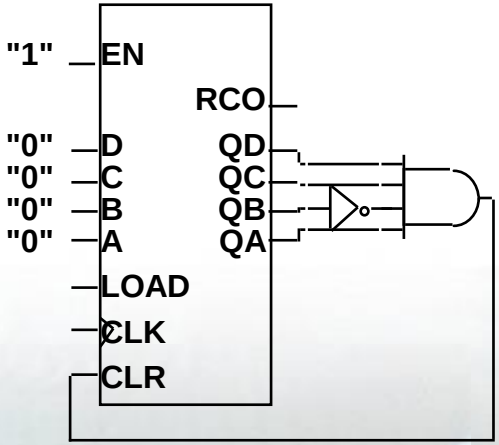
判断状态9的出现，在下一个时钟沿状态状态变为0

偏移计数器offset counter

- 起始偏移计数 – 采用同步装载load
  - e.g., 0110, 0111, 1000, 1001,
    1010, 1011, 1100, 1101, 1111, 0110, . . .



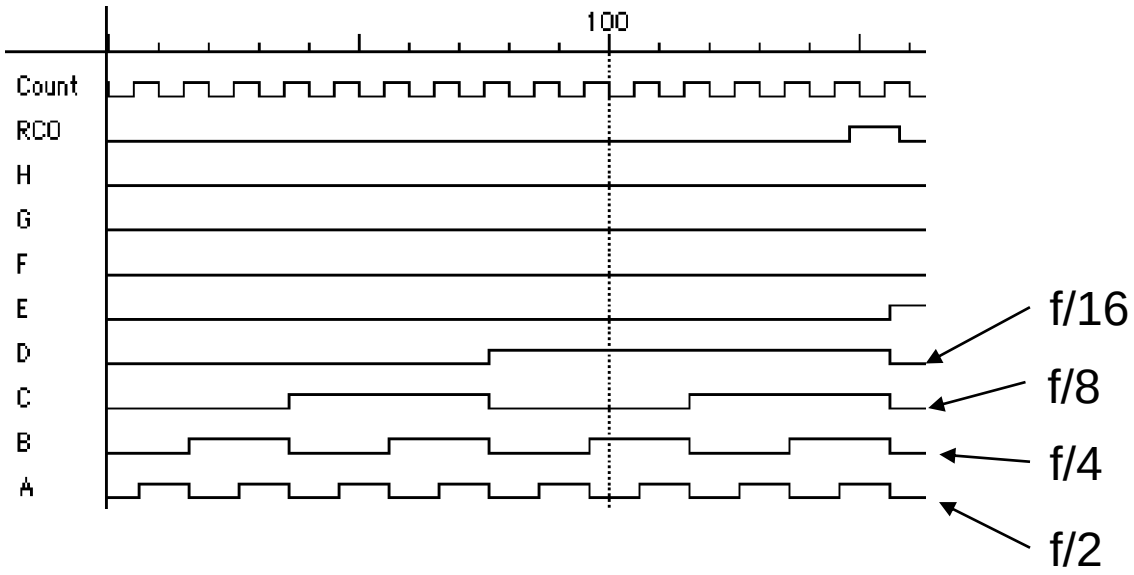
- 截止偏移计数器 – 与截止至比较
  - e.g., 0000, 0001, 0010, ..., 1100, 1101, 0000

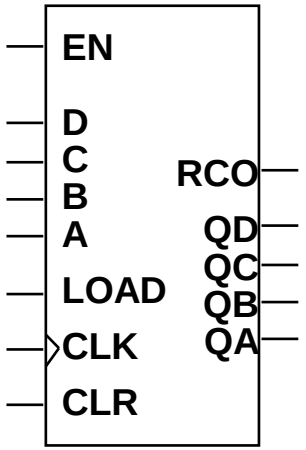


- 两者结合（start/stop counter）

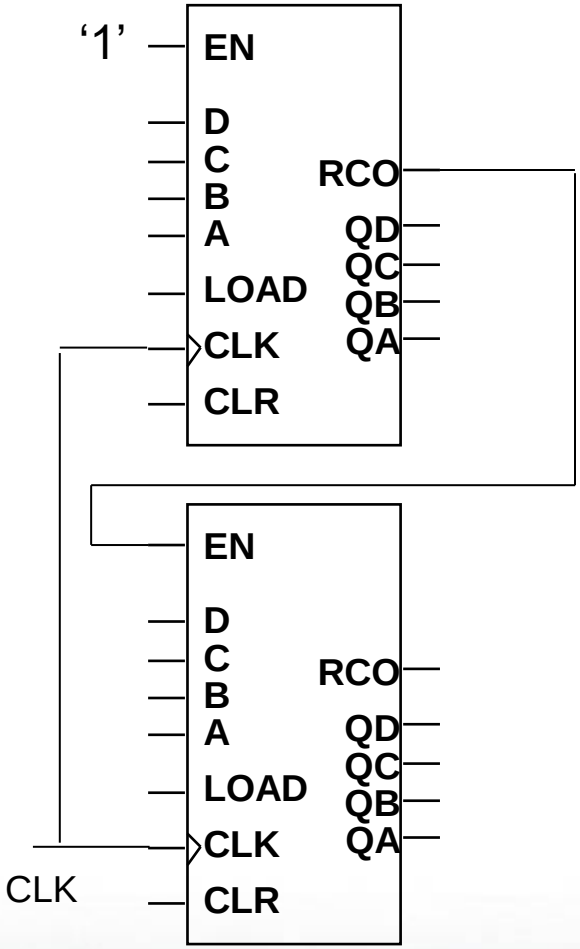
计数器的分频功能

- 计数器的0…n-1为输出频率分别是f/2, f/4, …, $f/2^n$

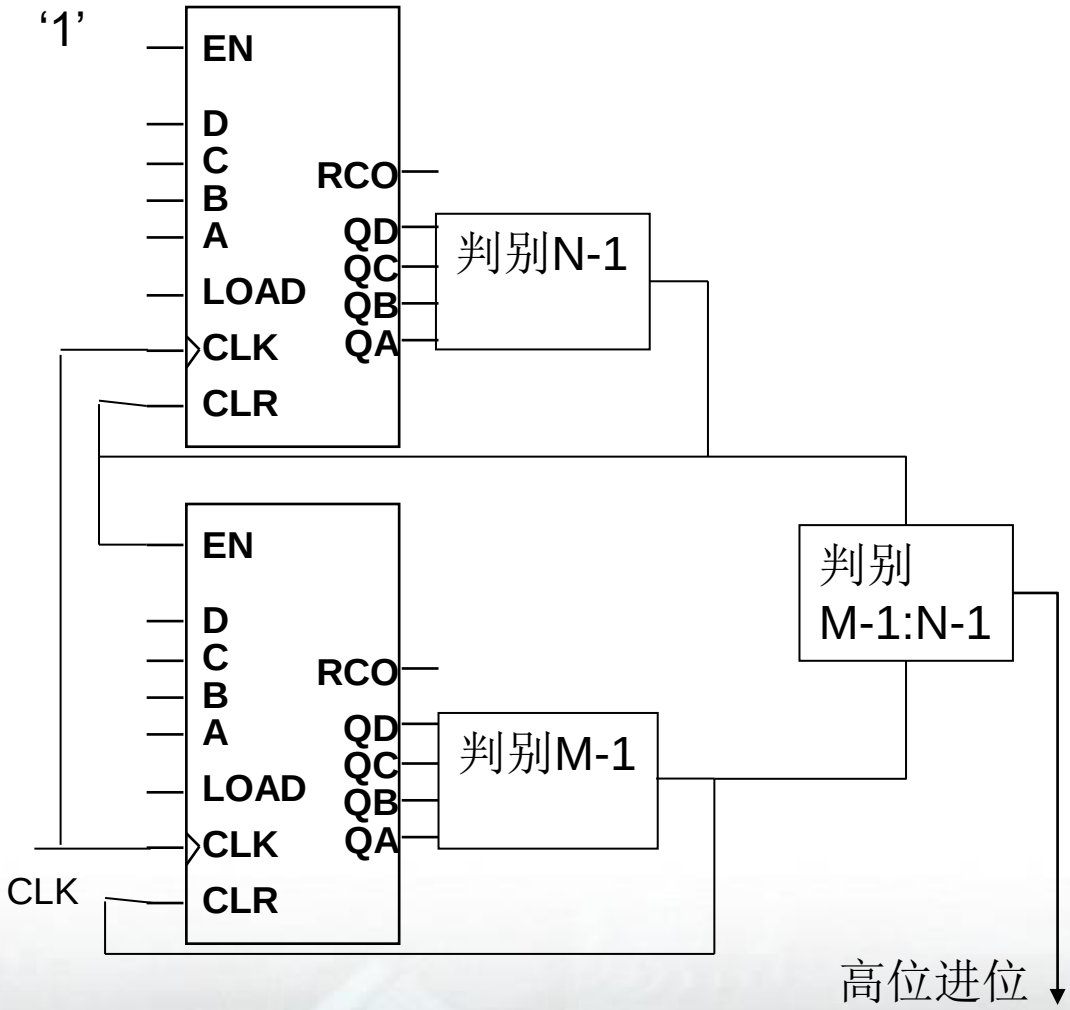




计数器的级联



8位二进制计数器



4. Verilog中的触发器和寄存器

- 用always块中的敏感表(sensitivity list)等待时钟沿事件

```
module dff (clk, d, q);

    input  clk, d;
    output q;
    reg    q;

    always @(posedge clk)
       q = d;

endmodule
```

更多功能的触发器

- 同步/异步的复位/置位
  - 同步：一个线程等待时钟事件
  - 异步：三个并行的线程，只有其中一个等待时钟事件

同步Synchronous

```
module dff (clk, s, r, d, q);
    input  clk, s, r, d;
    output q;
    reg    q;

    always @(posedge clk)
        if (r)       q = 1'b0;
        else if (s)  q = 1'b1;
        else         q = d;

endmodule
```

异步Asynchronous

```
module dff (clk, s, r, d, q);
    input  clk, s, r, d;
    output q;
    reg    q;

    always @(posedge r)
        q = 1'b0;
    always @(posedge s)
        q = 1'b1;
    always @(posedge clk)
        q = d;

endmodule
```

不正确的Verilog触发器描述

- 等待时钟的变化（而不是时钟沿）

```
module dff (clk, d, q);

    input  clk, d;
    output q;
    reg    q;

    always @(clk)
        q = d;

endmodule
```

不正确，q将在时钟的任何变化时变化，而不是时钟沿变化.

阻塞赋值和非阻塞赋值

- 阻塞赋值Blocking assignments (X=A)
  - 在继续下一条语句前赋值
- 非阻塞赋值Non-blocking assignments (X<=A)
  - 在always块完成前不完成赋值

```
always @(posedge CLK)
    begin
        temp = B;
        B = A;
        A = temp;
    end
```

```
always @(posedge CLK)
    begin
        A <= B;
        B <= A;
    end
```

寄存器传输级Register-transfer-level (RTL)

- 非阻塞赋值又称为寄存器传输级赋值
  - 如果在always块中被一个时钟沿触发
  - 所有触发器一起改变值

```
// B,C,D all get the value of A
always @(posedge clk)
   begin
      B = A;
      C = B;
      D = C;
   end
```

```
// implements a shift register too
always @(posedge clk)
   begin
      B <= A;
      C <= B;
      D <= C;
   end
```

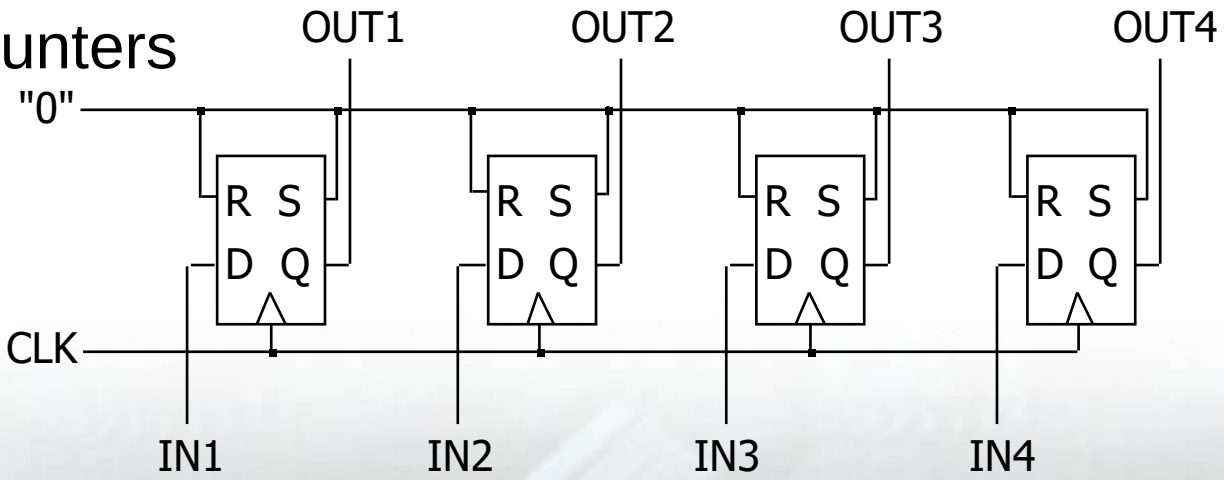
Verilog中的二进制计数器

```
module binary_counter (clk, c8, c4, c2, c1);

   input  clk;
   output c8, c4, c2, c1;

   reg [3:0] count;

   initial begin
      count = 0;
   end

   always @(posedge clk) begin
      count = count + 4'b0001;
   end

   assign c8 = count[3];
   assign c4 = count[2];
   assign c2 = count[1];
   assign c1 = count[0];

endmodule
```

```
module binary_counter (clk, c8, c4, c2, c1, rco);

   input  clk;
   output c8, c4, c2, c1, rco;

   reg [3:0] count;
   reg rco;

   initial begin . . . end

   always @(posedge clk) begin . . . end

   assign c8 = count[3];
   assign c4 = count[2];
   assign c2 = count[1];
   assign c1 = count[0];
   assign rco = (count == 4b'1111);

endmodule
```

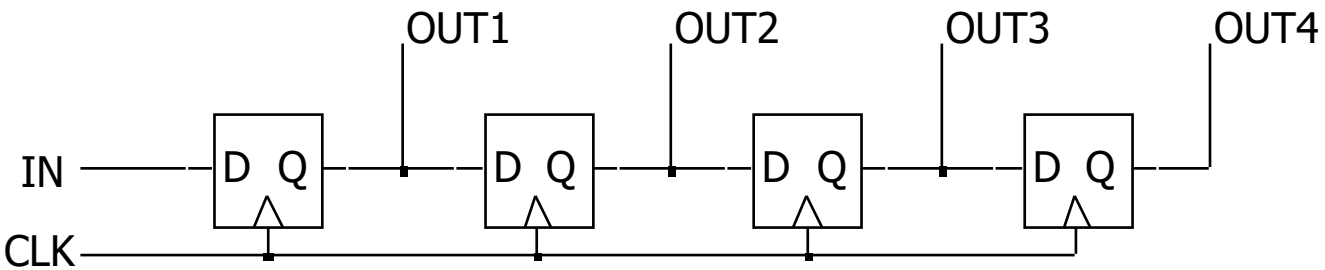
2. 寄存器Registers

- 共用相同控制和逻辑的触发器集合
  - 被存储的值有一定的联系（如形成二进制值）
  - 共用clock、reset和set线
  - 在每一级有相同的逻辑
- 例子
  - 移位寄存器shift registers
  - 计数器counters



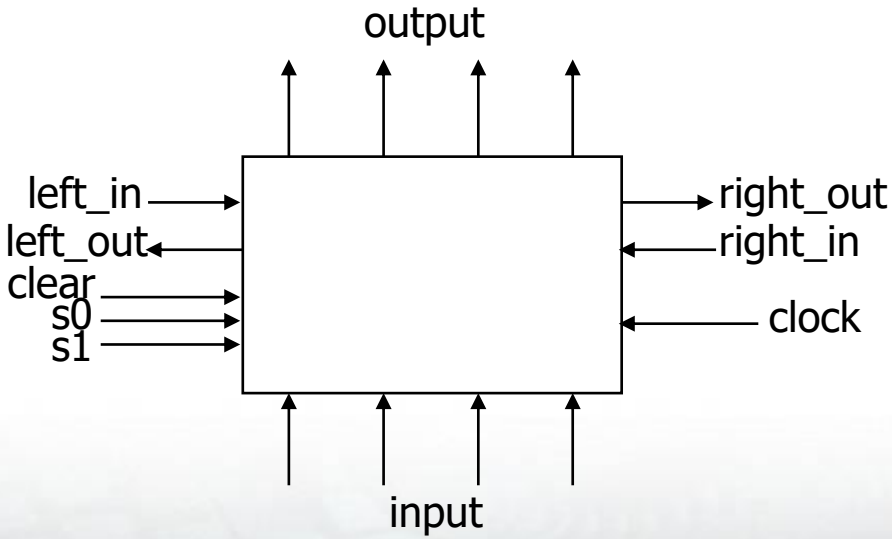
移位寄存器Shift register

- 保存输入的采样值
  - 存储输入序列的最后4个输入值
  - 4位移位寄存器 (4-bit shift register)



通用的移位寄存器

- 保持4个值
  - 串行或并行输入serial or parallel inputs
  - 串行或并行输出serial or parallel outputs
  - 准许左移还是右移permits shift left or right
  - 移入新值得方向是从左还是从右shift in new values from left or right



Clear信号将寄存器的值和输出清0

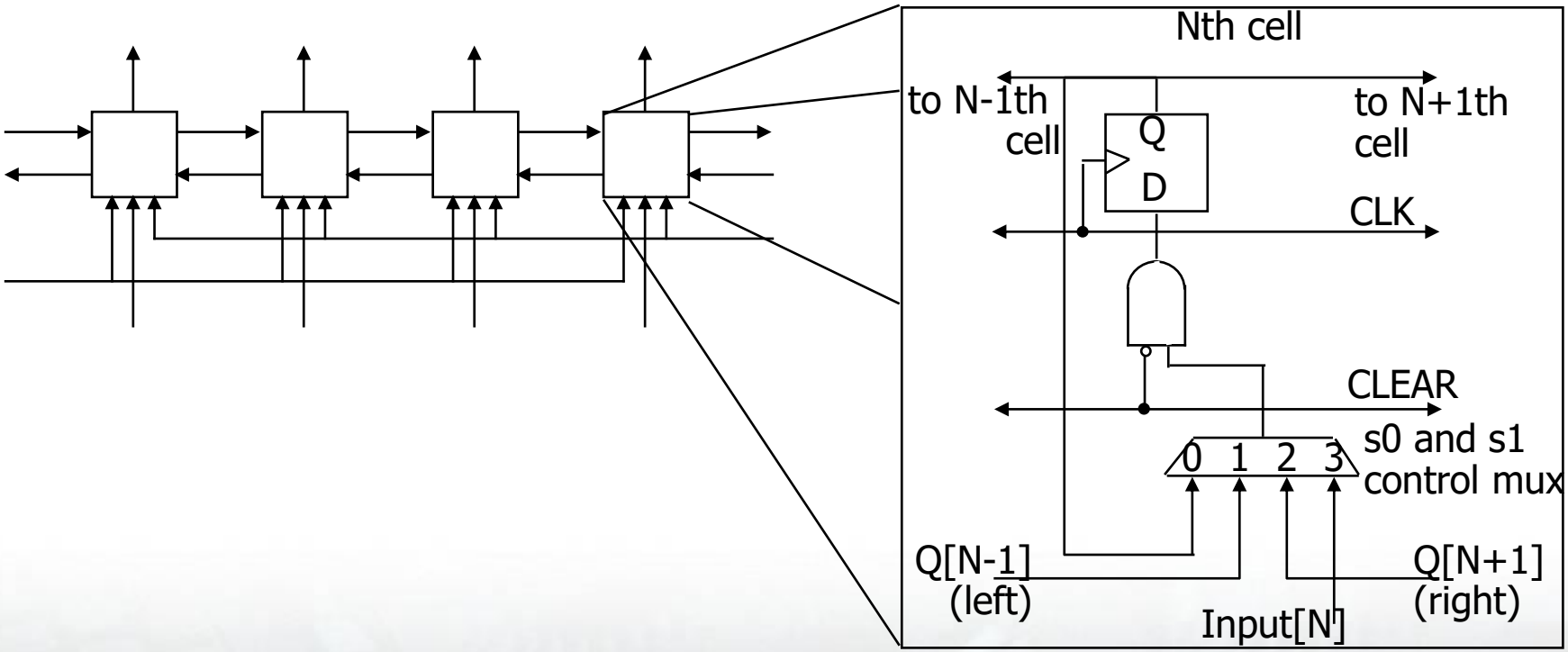
s1和s0 决定移位功能

s0	s1	function
0	0	hold state
0	1	shift right
1	0	shift left
1	1	load new input

通用移位寄存器设计

- 考虑每个触发器
  - 下一个周期的新值：

clear	s0	s1	new value
1	–	–	0
0	0	0	output
0	0	1	output value of FF to left (shift right)
0	1	0	output value of FF to right (shift left)
0	1	1	input

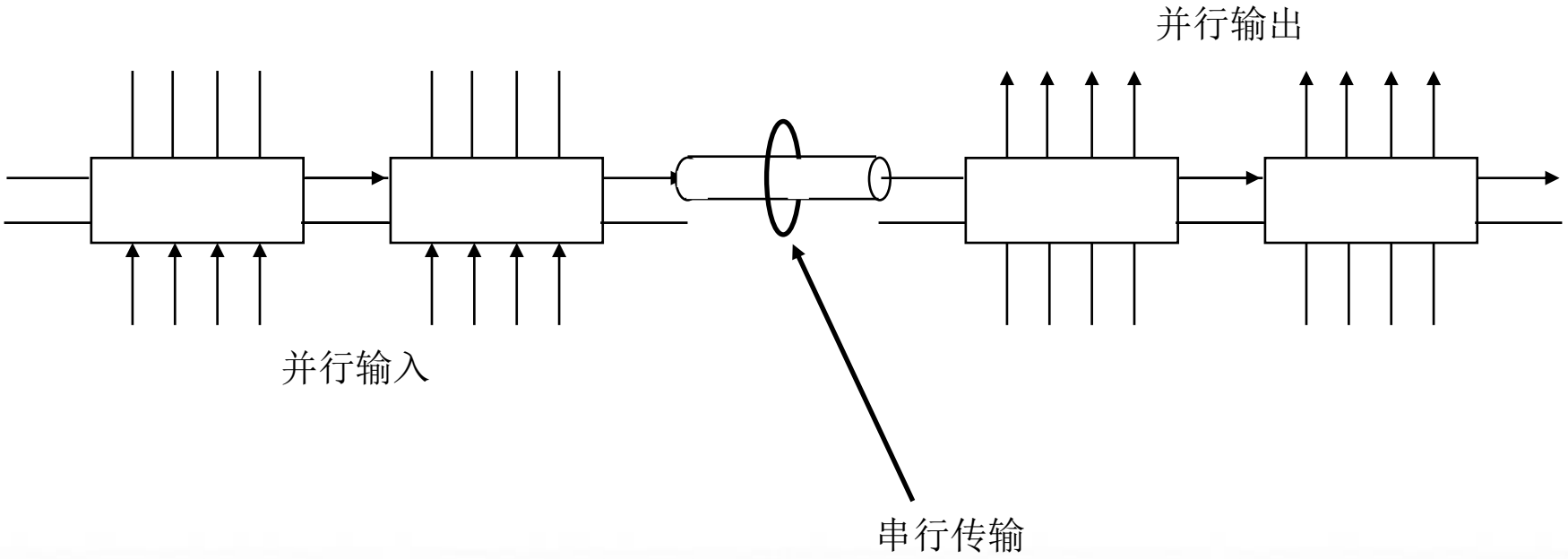


移位寄存器的应用

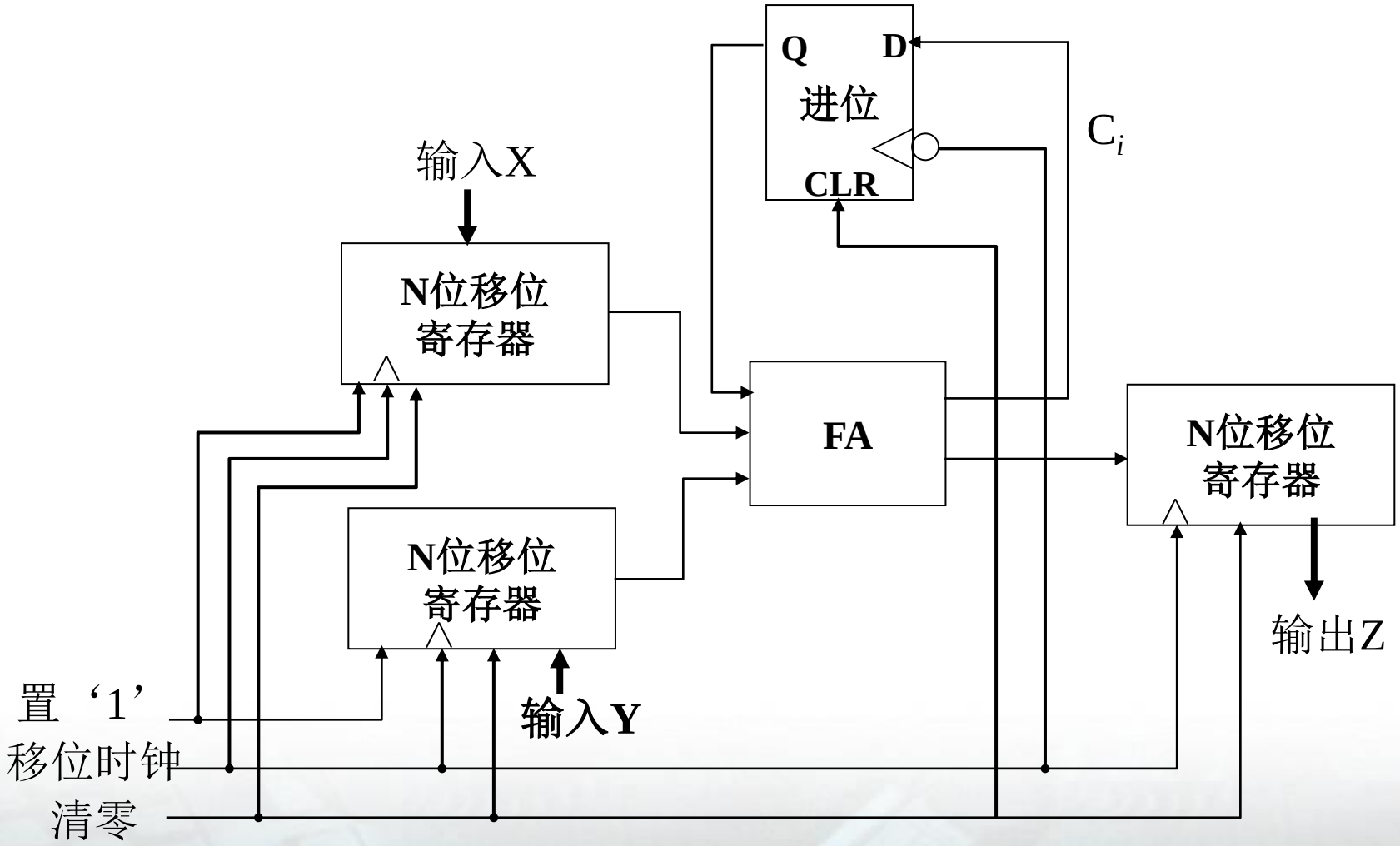
- 串行通信（串并转换）
- 二进制计算
  - 移位的数学意义
  - 左移n位，乘以2^n
  - 右移n位，除以2^n
- 形成特殊的序列
  - 例如生成伪随即数

应用1：串行传输

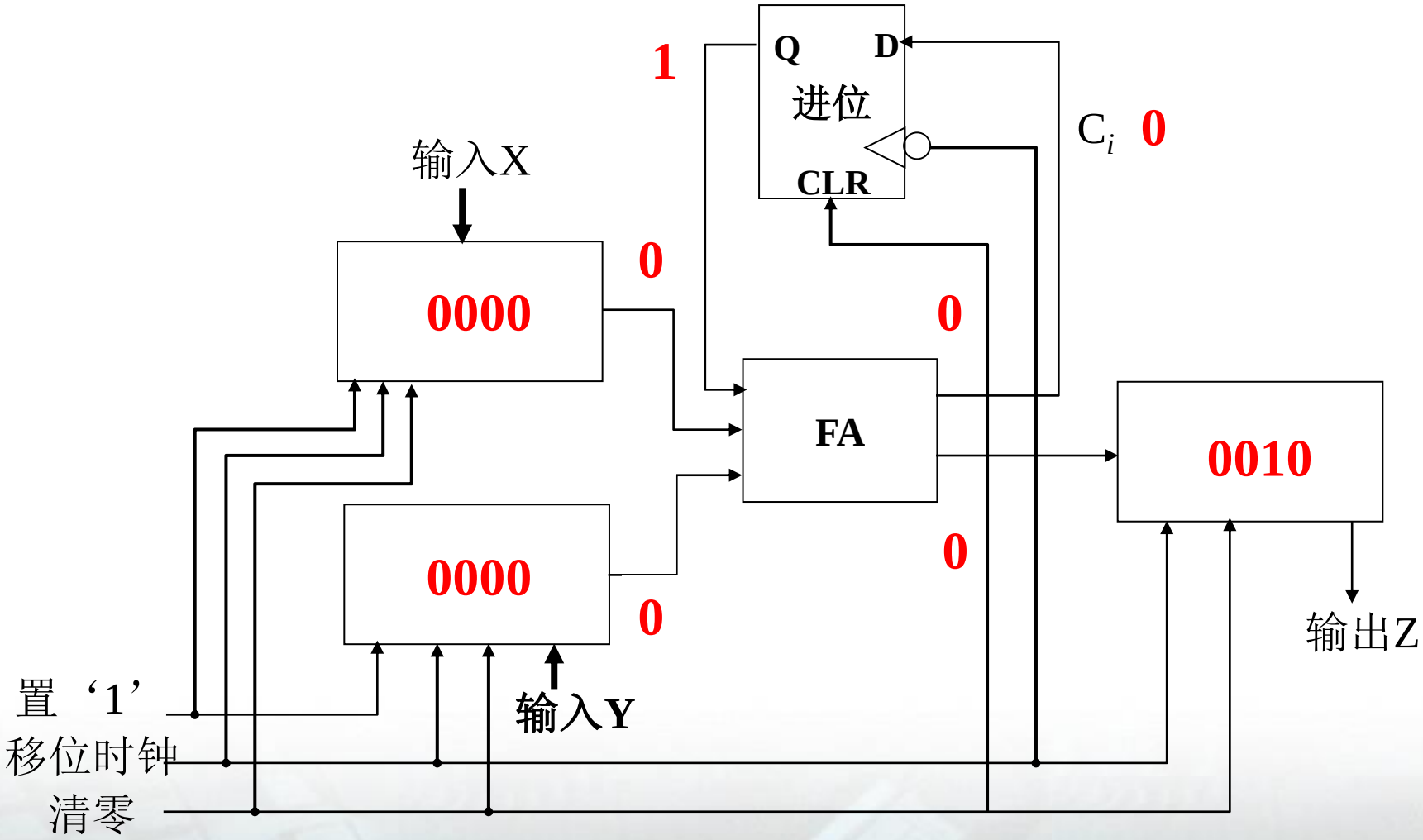
- 串并转换
  - 串转并功能
  - 并转串功能



应用2—串行加法器

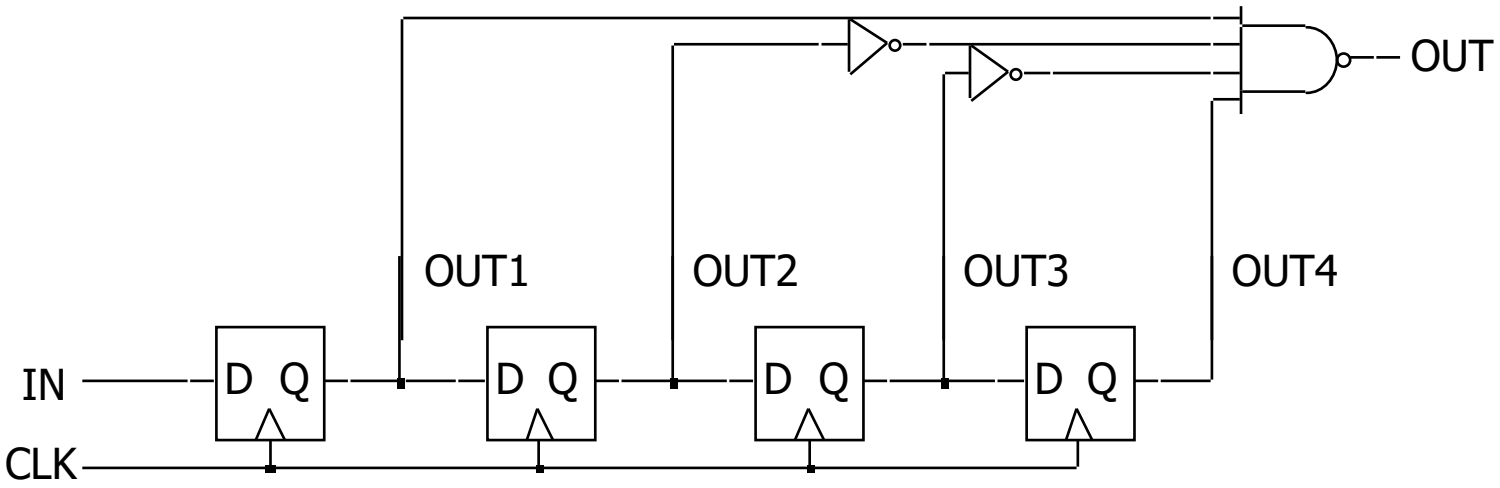


串行加法器（计算过程）



应用3：输入序列特定模式识别器

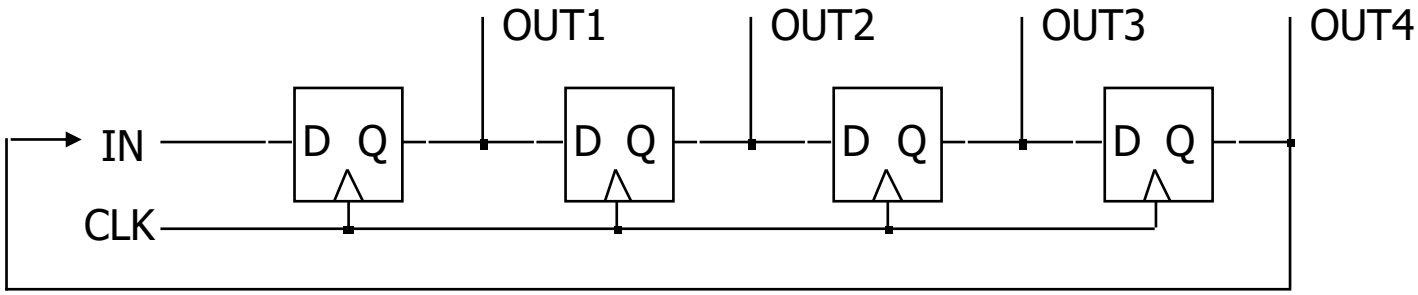
- 输入采样的组合函数
  - 识别1001



- 按钮识别

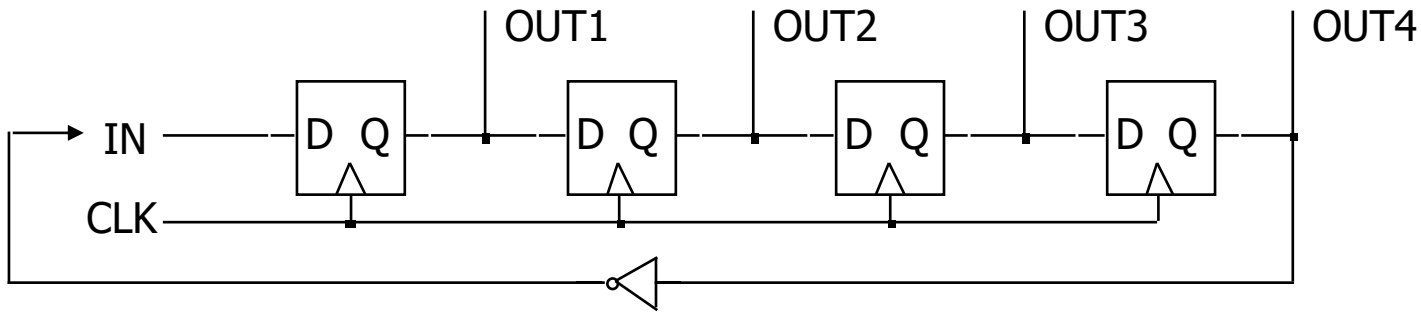
应用4：特殊序列计数器

- 产生固定模式序列的串行输出
  - 1000, 0100, 0010, 0001

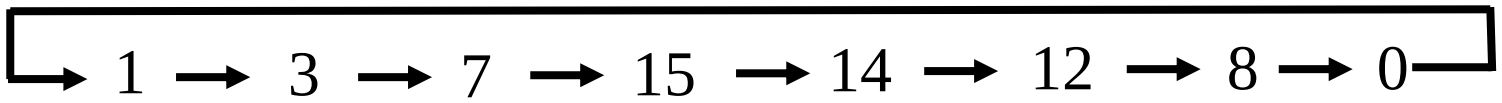


翻转循环计数器

- Mobius计数器
- 这个计数器如何工作？



- 计数序列: 1000, 1100, 1110, 1111, 0111, 0011, 0001, 0000



- 硬件随机数产生器

串行IO接口——SPI

- Serial Peripheral Interface

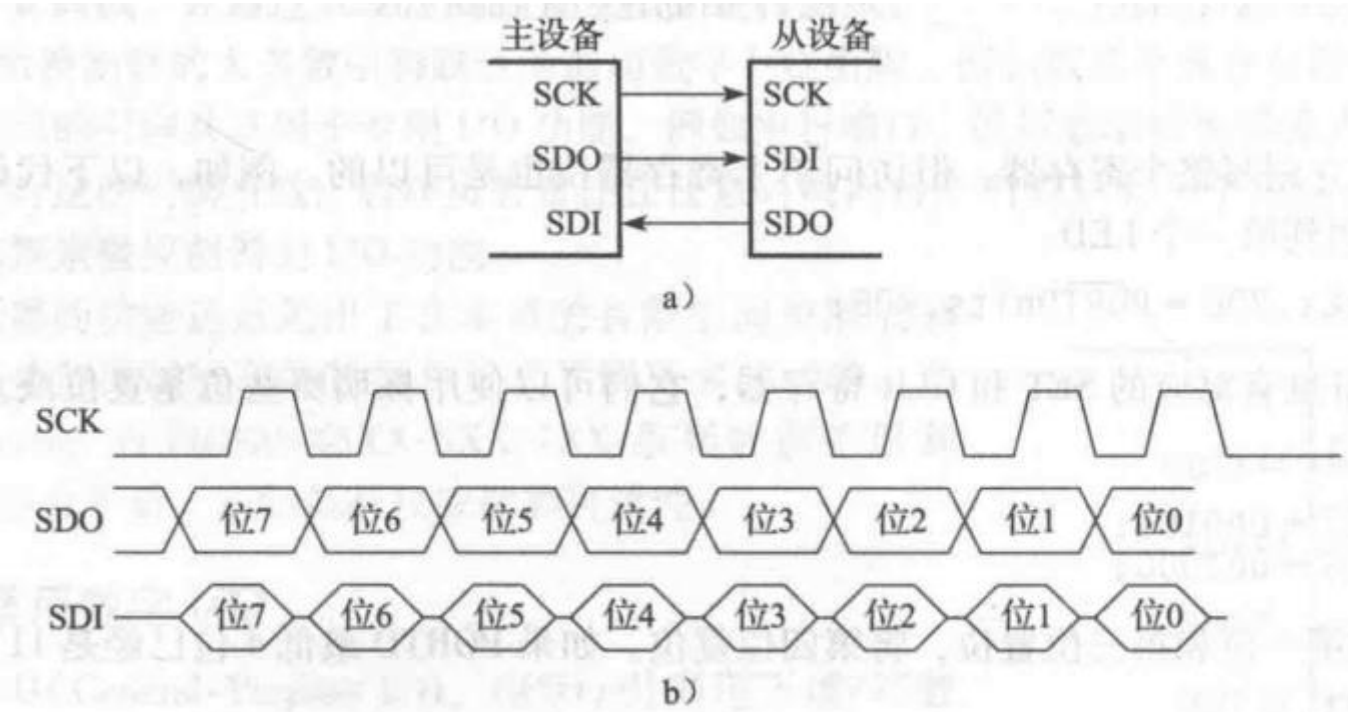


图 8-36　SPI 连接和主设备波形

串行IO接口——SPI

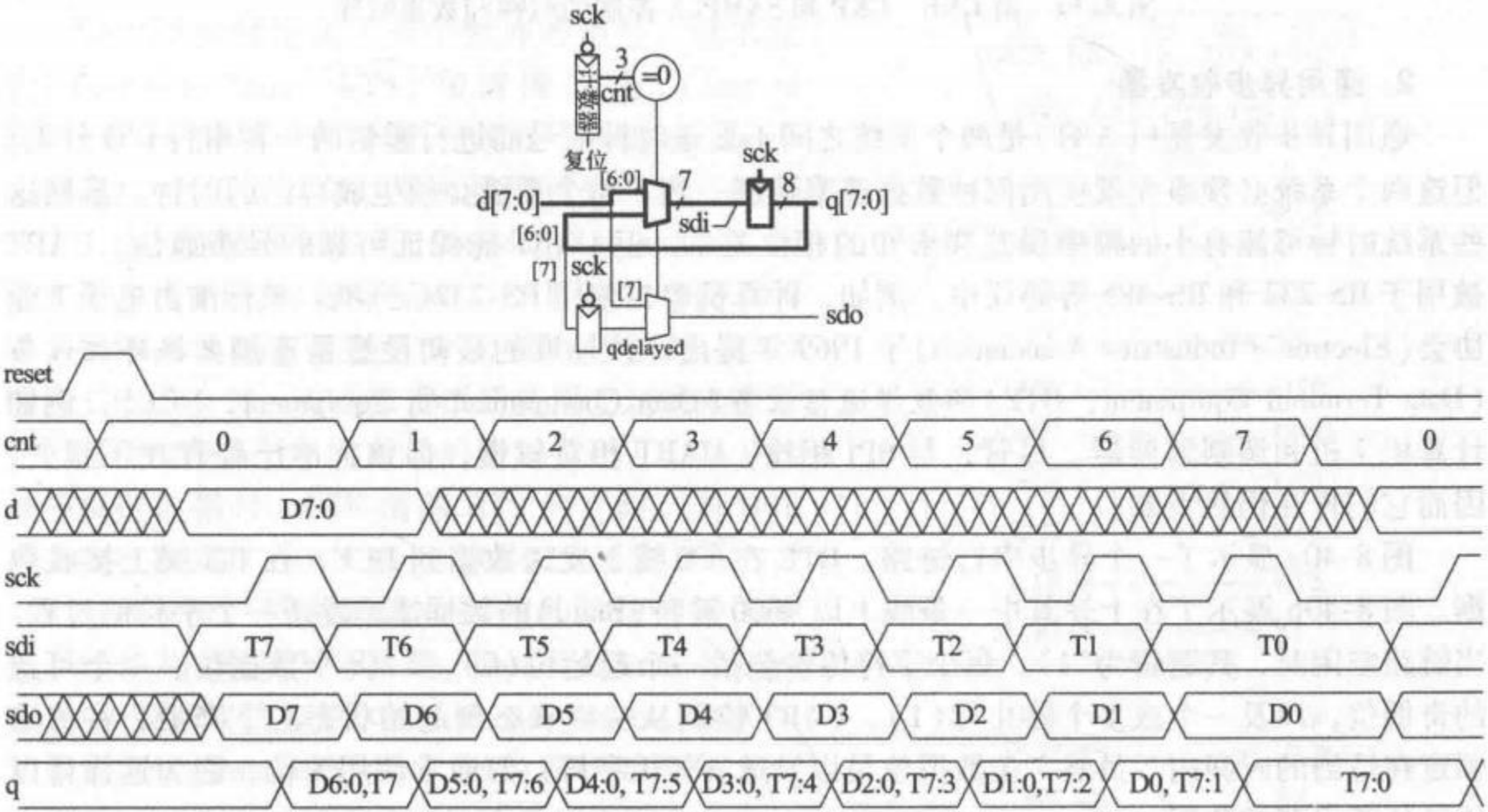
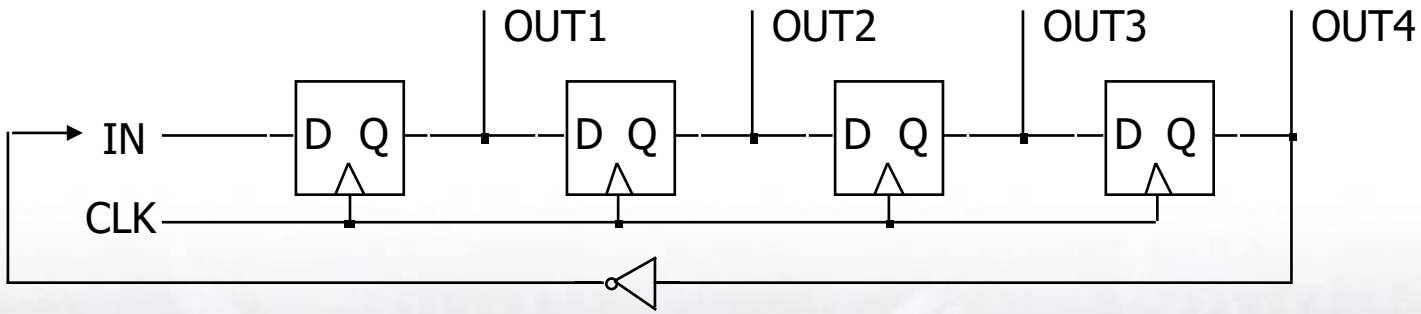


图 8-38　SPI 从设备电路和时序

Verilog中的Mobius计数器

```
initial
   begin
      A = 1'b0;
      B = 1'b0;
      C = 1'b0;
      D = 1'b0;
   end

always @(posedge clk)
   begin
      A <= ~D;
      B <= A;
      C <= B;
      D <= C;
   end
```



组合电路的实现技术

- 标准逻辑门Standard gates
  - 逻辑门封装芯片, 标准单元库
- 规整逻辑Regular logic
  - 多选器multiplexers
  - 译码器decoders
- 两级可编程逻辑Two-level programmable logic
  - PALs, PLAs,
- 存储器实现逻辑
  - ROMs
  - 现场可编程门阵列(Field-Programming Gate Array， FPGA)

存储器和组合逻辑

- 用存储器组合逻辑实现

F0 = A' B' C + A B' C' + A B' C

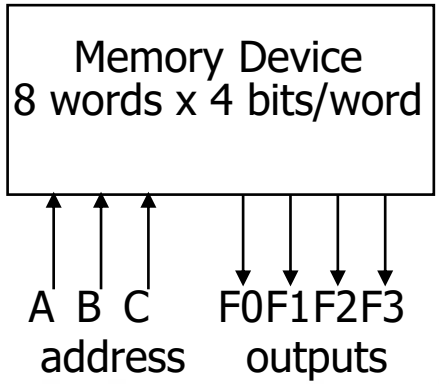
F1 = A' B' C + A' B C' + A B C

F2 = A' B' C' + A' B' C + A B' C'

F3 = A' B C + A B' C' + A B C'

A	B	C	F0	F1	F2	F3
0	0	0	0	0	1	0
0	0	1	1	1	1	0
0	1	0	0	1	0	0
0	1	1	0	0	0	1
1	0	0	1	0	1	1
1	0	1	1	0	0	0
1	1	0	0	0	0	1
1	1	1	0	1	0	0

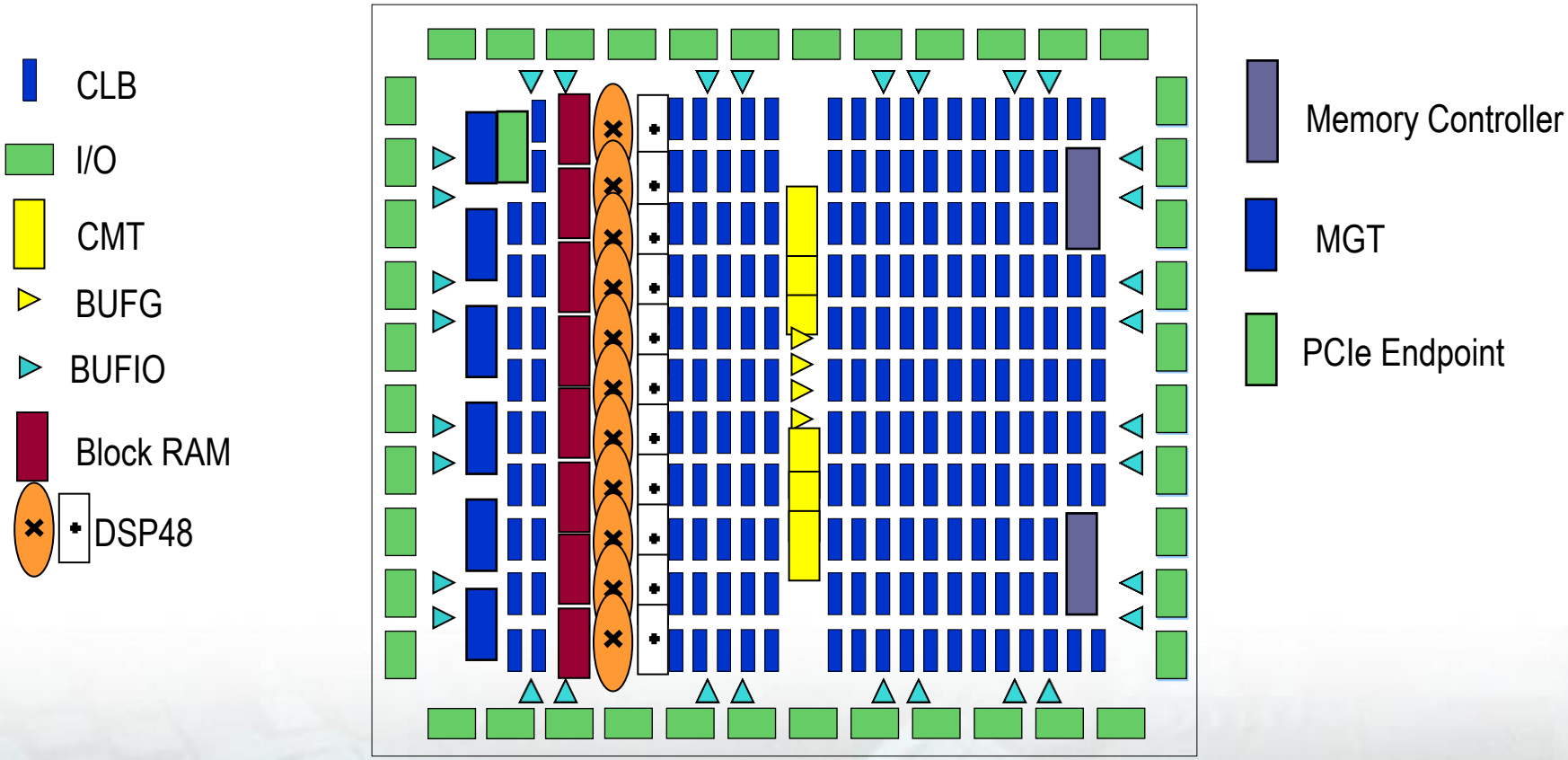
真值表



框图

Xilinx FPGA概述

- FPGA Field Programmable Gate Array
  - 现场可编程门阵列

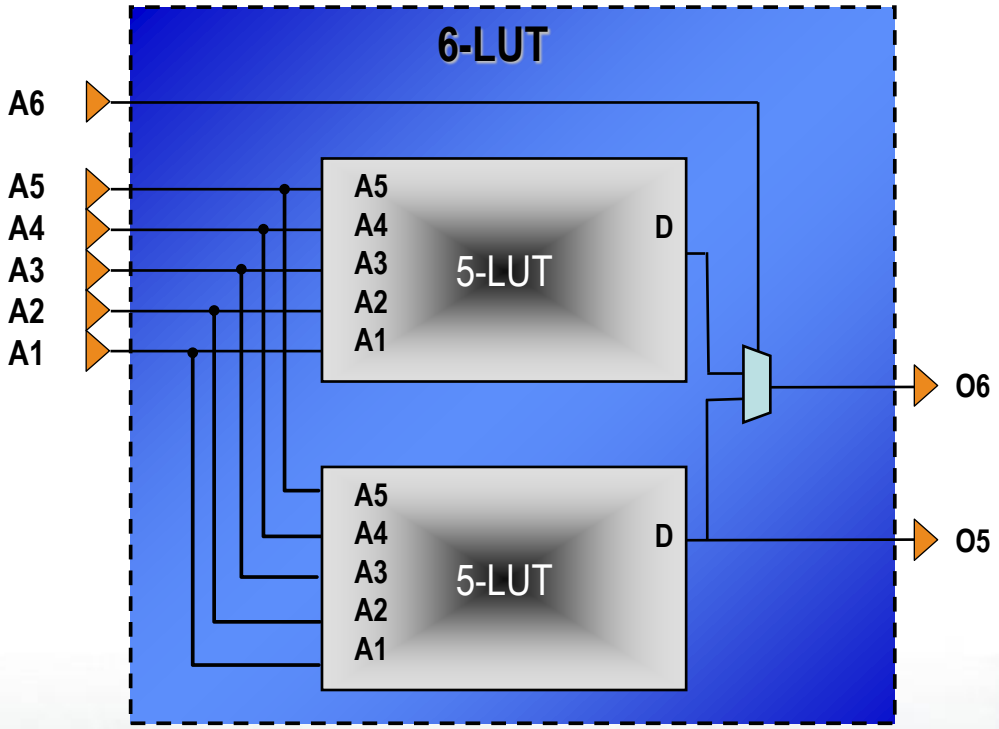


Xilinx FPGA概述

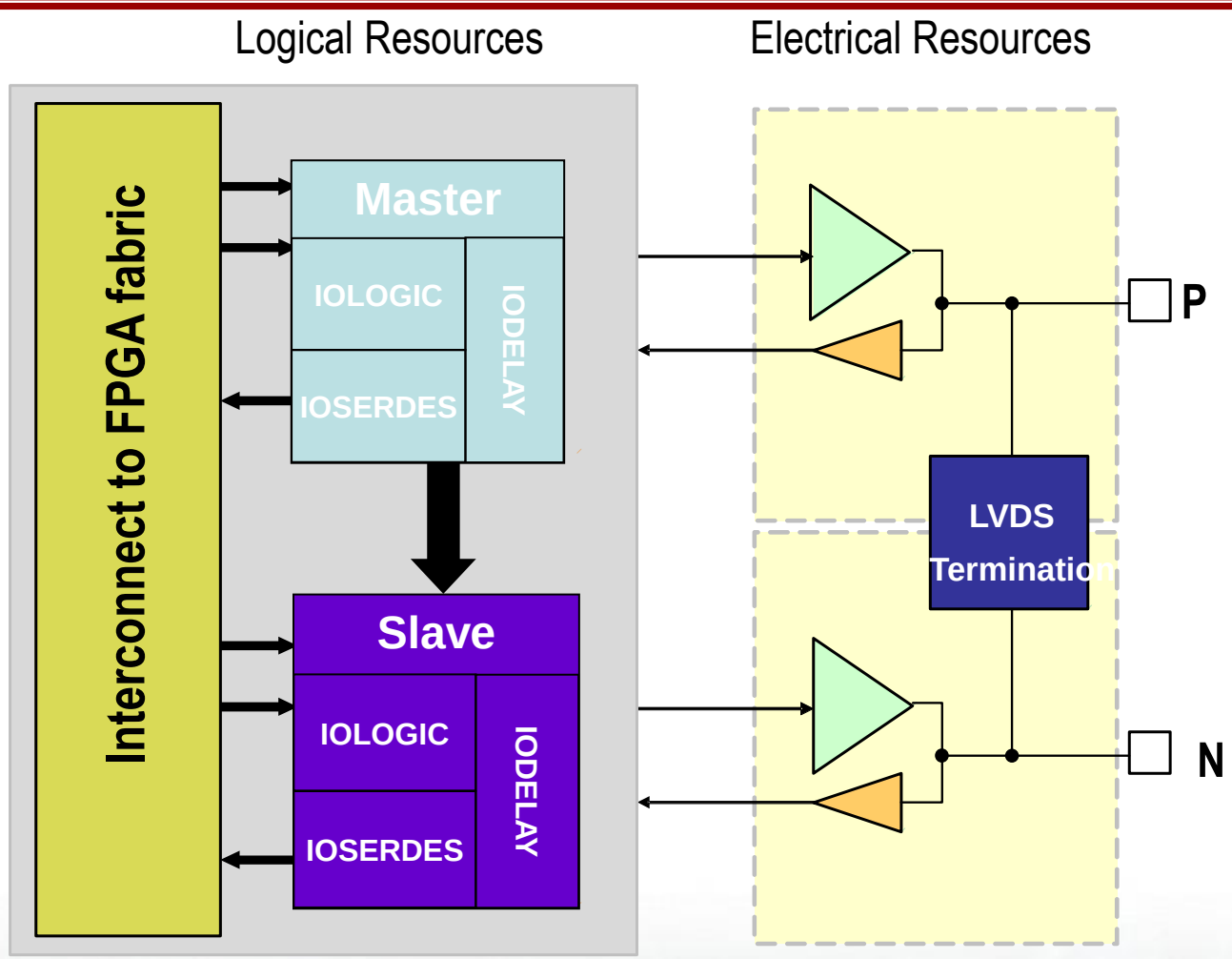
- 所有的Xilinx FPGAs都包含相同的基本资源
  - 逻辑资源Logic Resources
    - 片(Slices)组成CLB(Configurable Logic Block 可配置逻辑块）
      - 包含组合逻辑和寄存器资源
    - 存储器Memory
    - 乘法器Multipliers
  - 互连资源Interconnect Resources
    - 可编程互连资源Programmable interconnect
    - 输入输出块IOBs
      - FPGA与外部的接口
  - 其它资源Other resources
    - 全局时钟缓冲器Global clock buffers
    - 边缘检测逻辑Boundary scan logic

6-Input LUT with Dual Output

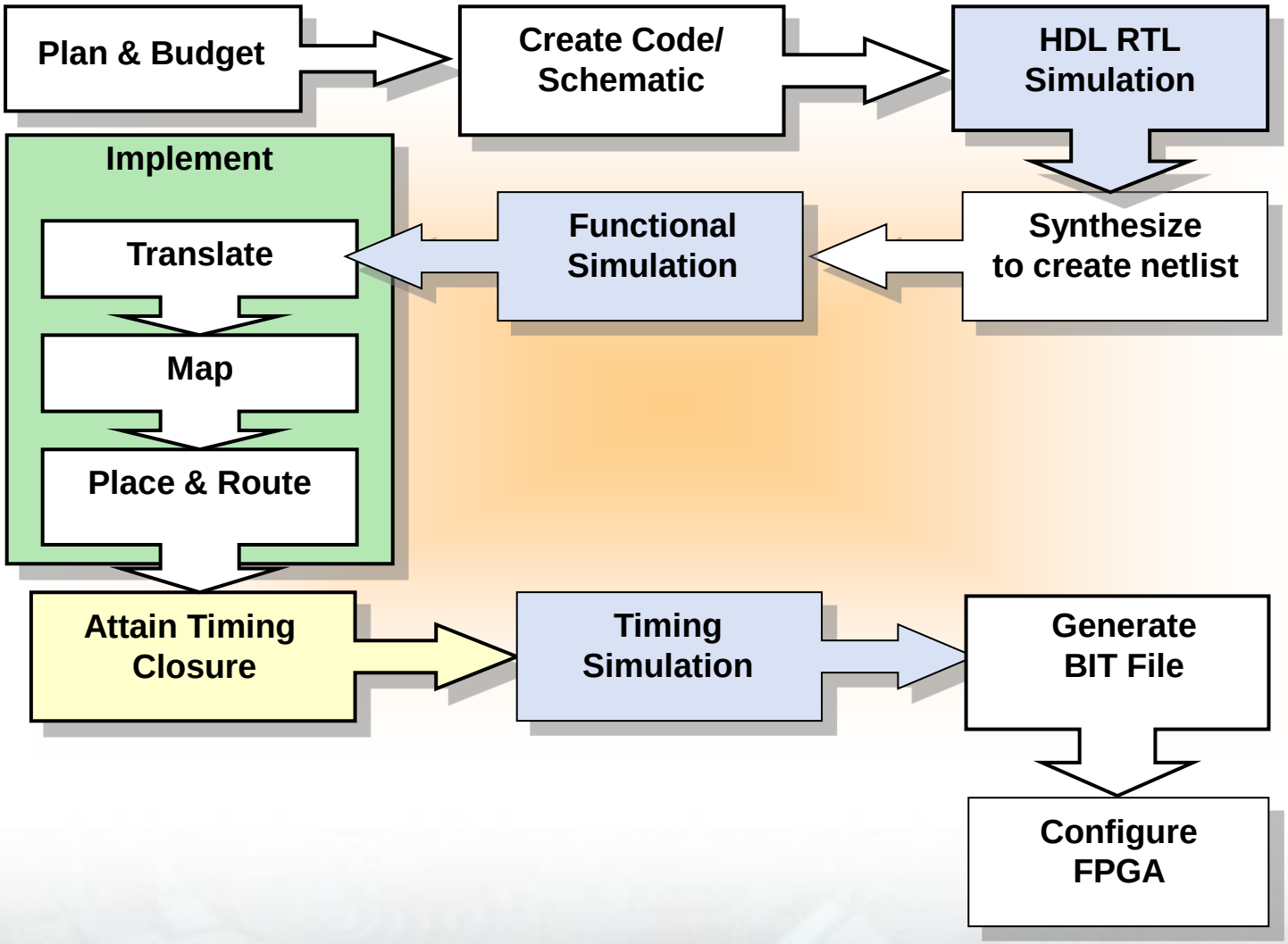
- 6-input LUT 包含两个共用输入的5-input LUTs
  - 影响一个6-input LUT最优速度
  - 一个或者两个输出
  - 任意6变量的函数
  - 两个独立的5变量函数



输入输出块(IOB)的结构图



Xilinx设计流程



电子设计自动化软件

- CAD, Computer-aid Design
- EDA, Electronic Design Automatic
- ESL, Electronic System Level

- 系统设计(C/C++，SystemC）
- 设计输入(Verilog, VHDL, Schematic)
- 设计验证(Verification)
- 设计综合(ASIC, FPGA, Analog)
- 后端(Back-end)设计

1 设计输入 Design Entry

- 原理图输入
- 激励和输出逻辑方程
- 状态表
- 状态图或者ASM图
- 硬件描述语言
  - Verilog
  - VHDL

2 综合 Synthesis

- 状态优化
- 选择状态分配方案
- 选择触发器类型
- 对组合逻辑进行优化

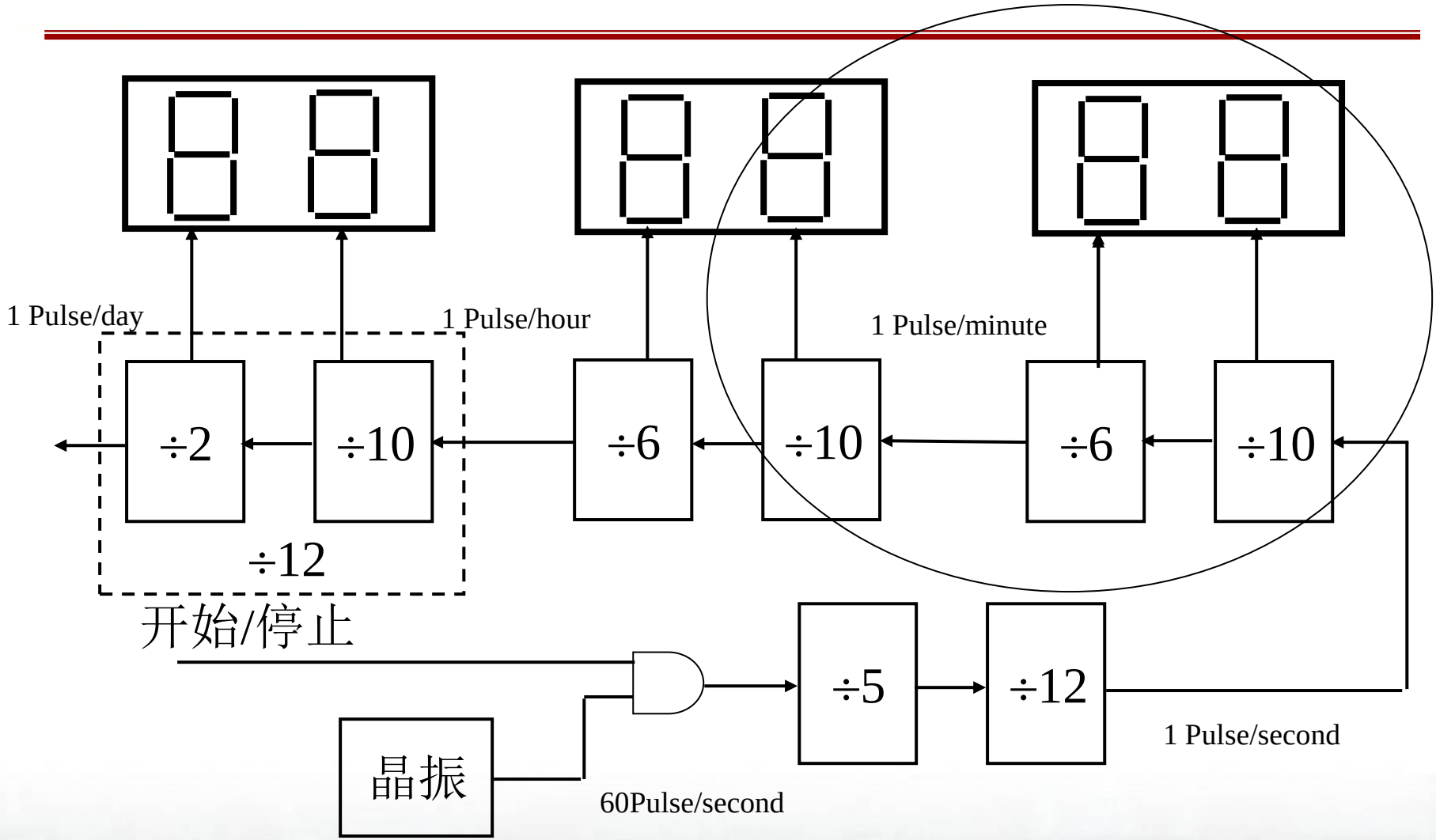
3 设计验证Verification

- 功能分析与验证
  - 模拟 Simulation
  - 形式化验证 Formal Verification
- 时序分析
  - Timing Analysis
  - Timing Methodology

4后端(Back-end)设计

- 时钟树生成
- Place & Route
- 参数提取，后仿真
- 物理验证（信号完整性分析等）
- 形成加工工艺文件
  - FPGA
  - ASIC, GDSII
  - …

Lab4：电子秒表



输入模块和时钟产生

```
reg sync_clr0, sync_clr1;
  always @(posedge clk_10Hz_bufg)
  begin
    sync_clr0 <= btnC;
    sync_clr1 <= sync_clr0;
  end
  assign clr = btnC | sync_clr1;

  reg sync_en0, sync_en1, sync_en2;
  reg en_reg;
  always @(posedge clk_10Hz_bufg)
  begin
    sync_en0 <= btnD;
    sync_en1 <= sync_en0;
    sync_en2 <= sync_en1;
  end

  always @(posedge clk_10Hz_bufg)
  begin
    if (clr)
      en_reg <= 0;
    else if (sync_en2 & ~sync_en1)
      en_reg <= ~en_reg;
  end
  assign en = en_reg;
```

```
//generate 10hz clock
  reg [23:0] cnt;
  reg clk_10Hz; //1KHz
  wire clk_10Hz_bufg;

  parameter PERIOD_CLK = 10000000;

  always @(posedge clk)
    begin
      if (cnt == PERIOD_CLK/2)
      begin
        cnt <= 0;
        clk_10Hz <= ~clk_10Hz;
      end
      else
        cnt <= cnt + 1;
    end

  BUFG  CLK0_BUFG_INST (.I(clk_10Hz),
                        .O(clk_10Hz_bufg));
//generate 10Hz clock
```

秒表和输出显示

```
//stopwatch
  wire [3:0] ms100, sec1, sec10, min1;

  stopwatch u_stopwatch(
  .clk_10Hz(clk_10Hz_bufg),
  .clr(clr),
  .en(en),
  .min1(min1),
  .sec10(sec10),
  .sec1(sec1),
  .ms100(ms100)
  );

  //display
  display_7seg_x4 u_display_7seg_x4(
  .CLK(clk), //100MHz
  .in0(ms100),
  .in1(sec1),
  .in2(sec10),
  .in3(min1),
  .seg(seg),
  .an(an)
  );
```

```
  reg [19:0] count=0;
  reg [1:0] sel=0;
  parameter T1MS = 100000;
  reg [3:0] seg7_in;

always @(seg7_in)
    begin
        case(seg7_in)
                // abc_defg
          0: seg = 7'b000_0001; //111_1110;
          1: seg = 7'b100_1111; //011_0000;
          2: seg = 7'b001_0010; //110_1101;
          3: seg = 7'b000_0110; //111_1001;
          4: seg = 7'b100_1100; //011_0011;
          5: seg = 7'b010_0100; //101_1011;
          6: seg = 7'b010_0000; //101_1111;
          7: seg = 7'b000_1111; //111_0000;
          8: seg = 7'b000_0000; //111_1111;
          9: seg = 7'b000_1100; //111_0011;
          default: seg = 7'b111_1111; //000_0000;
        endcase
    end
```

```
always @(posedge CLK)
    begin
      count<=count+1;
      if(count==T1MS)
        begin
          count<=0;
          sel<=sel+1;
        end
    end

    always @(sel, in0, in1, in2, in3)
    begin
      case(sel)
      0: begin an = 4'b0111; seg7_in = in0; end
      1: begin an = 4'b1011; seg7_in = in1; end
      2: begin an = 4'b1101; seg7_in = in2; end
      3: begin an = 4'b1110; seg7_in = in3; end
      default: begin an = 4'b1111; seg7_in =
4'bxxxx; end
      endcase
    end
```