第4章 时序逻辑电路

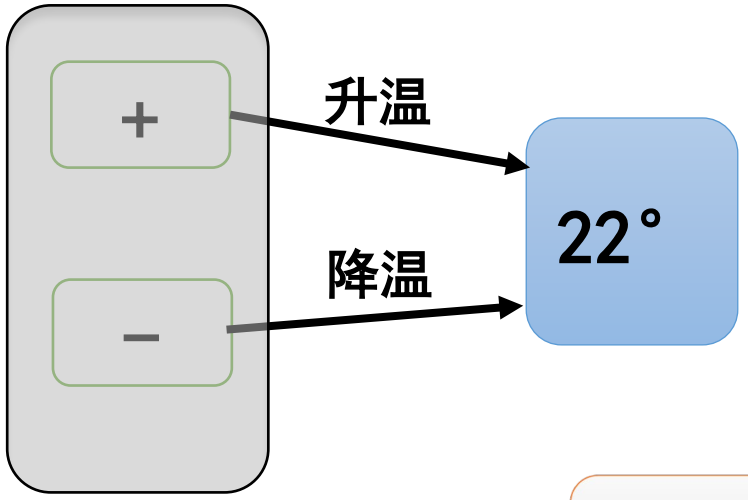
第一讲 时序逻辑电路概述

第二讲 锁存器和触发器

第三讲 同步时序电路设计

第四讲 典型时序逻辑部件设计

时序逻辑电路



+和−两个控制键，可以设置多个不同的温度。用组合逻辑无法实现该功能。

计算需求中时序逻辑

- 决定输出的状态有哪些？
  - 状态如何记录保持？
- 什么时候处于什么状态如何决定？
  - 状态如何流转？

第一讲 时序逻辑电路概述

1. 时序逻辑与有限状态机
2. 时序逻辑电路的基本结构
3. 时序逻辑电路的定时

时序逻辑电路概述

- **组合逻辑：输出结果仅取决于当前的输入信号**
- **时序逻辑：输出结果不仅取决于当前时刻的输入值，而且取决于电路过去时刻的行为（当前状态）**
  - 电路中有存储元件，用于记录电路过去时刻的行为（当前状态）
  - 当电路输入值发生变化时，新的输入值可能使得电路保持当前状态，也可能使得电路状态发生改变，进入新的状态

时序逻辑与有限状态机

- 用有限状态机刻画时序逻辑。
  - 有限状态机(Finite State Machine，FSM)是一种刻画状态及状态转换的理论工具。
- 通常用状态图描述有限状态机。
  - 状态：用包含状态符号的圆圈表示
  - 状态转换方向：用有向边表示，并在边上标注引起状态变换的输入信号值和相应输出响应（若输出响应只与当前状态有关，则把输出响应标注在状态圆圈中）

例：检测输入序列是否为连续4个“1”

A-初始态：若输入1，则转B

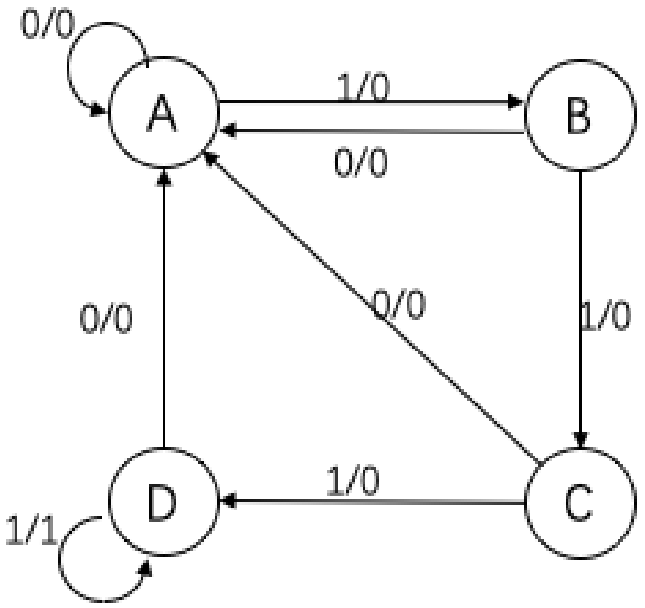
B-连续1个“1”：若输入1，则转C

C-连续2个“1”：若输入1，则转D

D-连续3个“1”：若输入1，则状态不变

并输出为1（表示检测到连续4个1）

任何状态下，输入0都会转到初态A



时序逻辑与有限状态机

- 用数字逻辑实现一个有限状态机（时序逻辑），需要完成工作：
  - 设计实现一种电路能进行状态的记忆。
    - 可使用双稳态器件来记忆状态，如SR锁存器、D触发器、JK触发器等；
    - 也可使用电路的稳态来实现，如反馈时序逻辑电路等。
  - 把状态机的输入、输出以及内部状态都转换成二进制表示。
    - 这里的关键是状态的二进制编码。
  - 设计出符合状态转换逻辑要求的状态记忆器件的激励函数和输出函数，并完成定时分析。

能完成上述工作（实现有限状态机）的数字逻辑电路是时序逻辑电路！

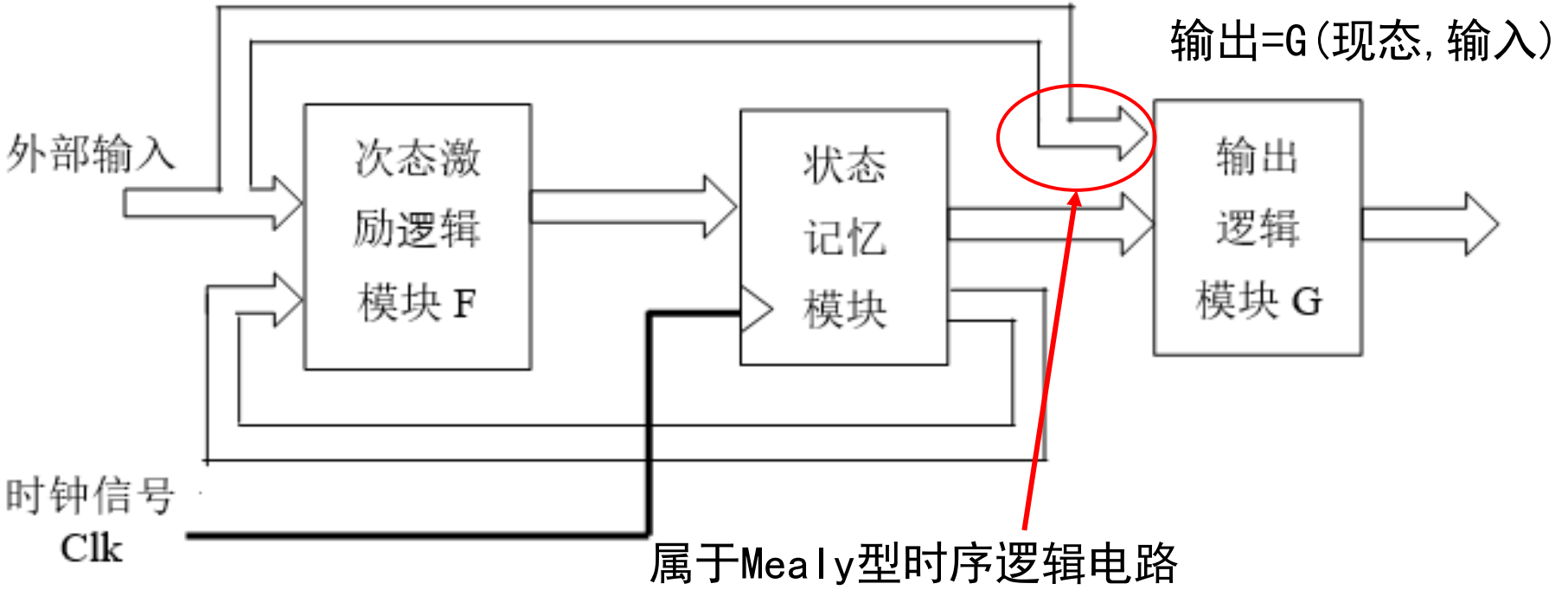
第一讲 时序逻辑电路概述

1. 时序逻辑与有限状态机
2. 时序逻辑电路的基本结构
3. 时序逻辑电路的定时

时序逻辑电路的基本结构

- 时序逻辑电路的一般结构
  - 状态记忆模块：由多个状态记忆单元构成
  - 次态激励逻辑模块F ：激励函数（现态和外部输入的逻辑函数）
  - 输出逻辑模块G ：输出函数（现态和外部输入的逻辑函数）

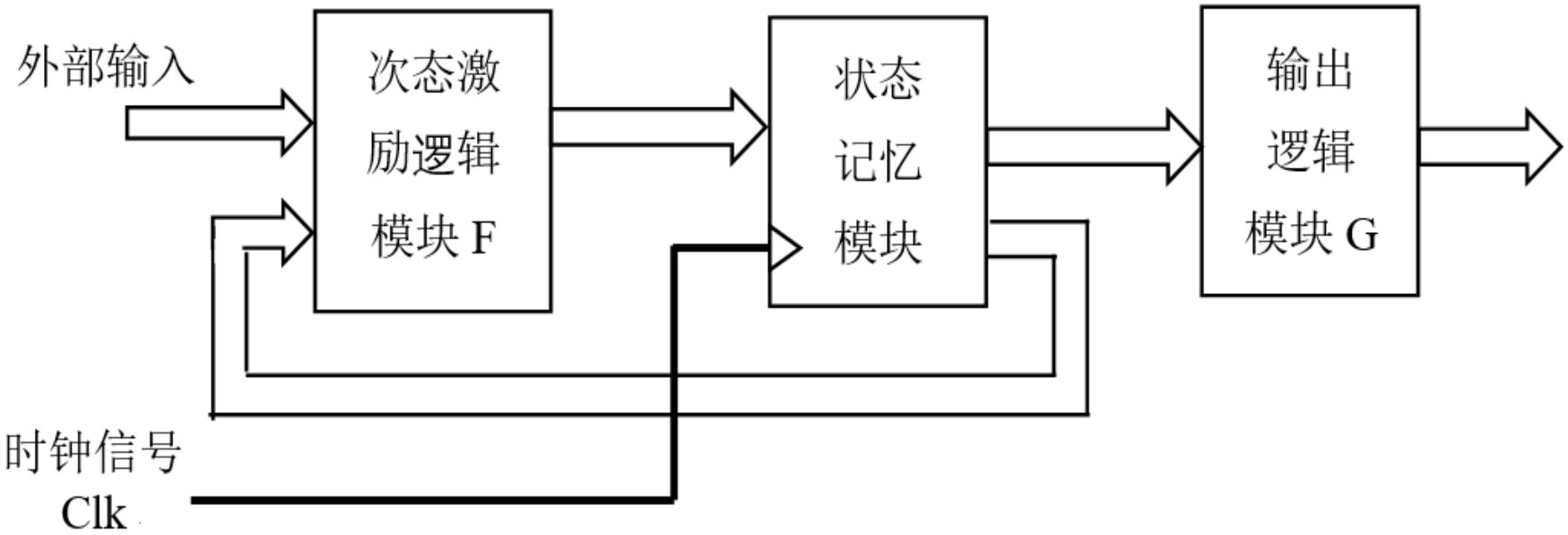
    Mealy型：输出依赖于当前状态和当前输入信号



时序逻辑电路的基本结构

Moore型：输出仅依赖于当前状态，和当前输入信号无关

输出=G(现态)



根据状态转换方式的不同有同步时序逻辑电路和异步时序逻辑电路

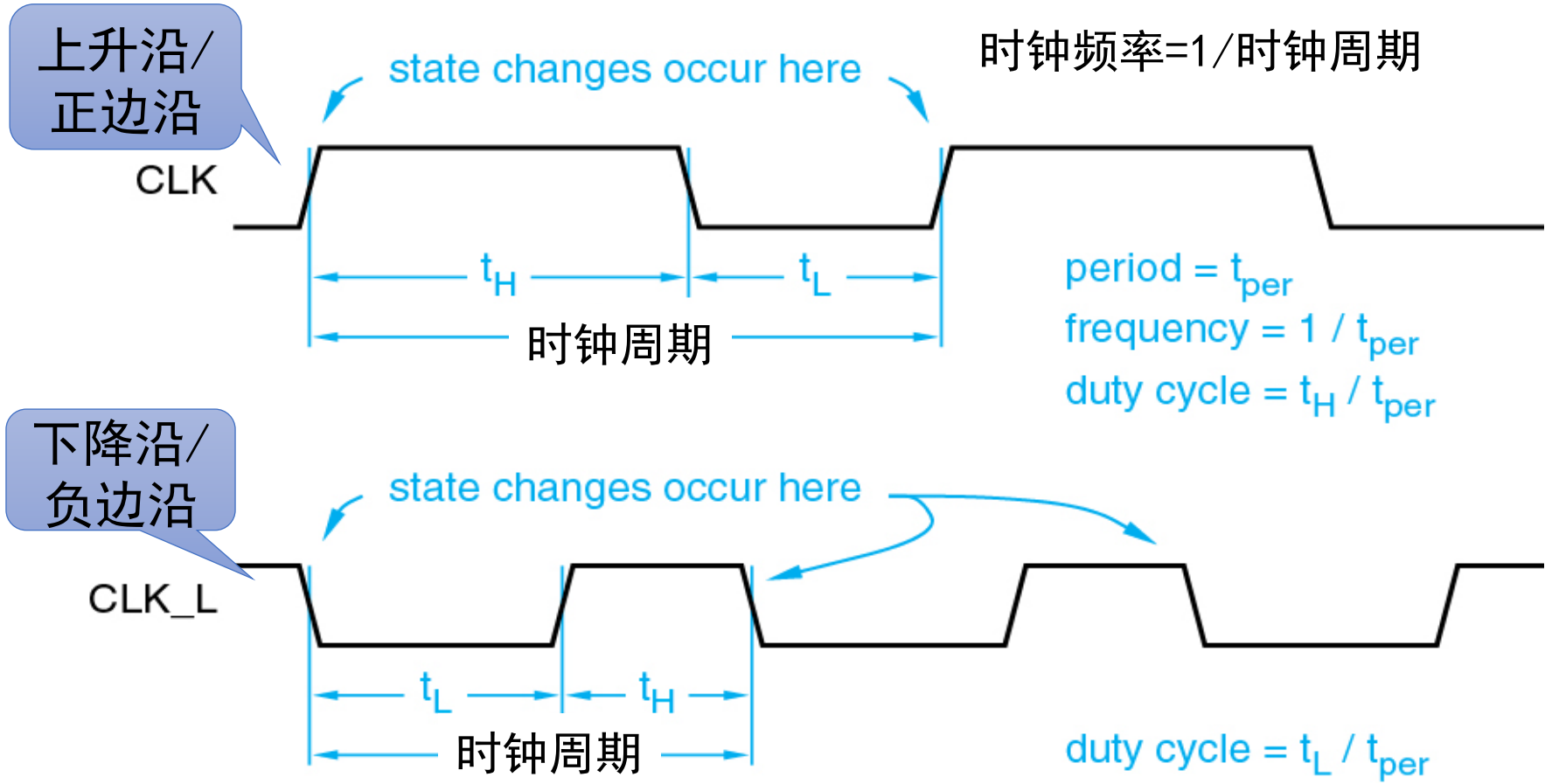
- 同步时序逻辑电路：在统一的时钟信号控制下进行状态转换
- 异步时序逻辑电路：没有统一的时钟信号来控制状态的改变

第一讲 时序逻辑电路概述

1. 时序逻辑与有限状态机
2. 时序逻辑电路的基本结构
3. 时序逻辑电路的定时

时序逻辑电路的定时

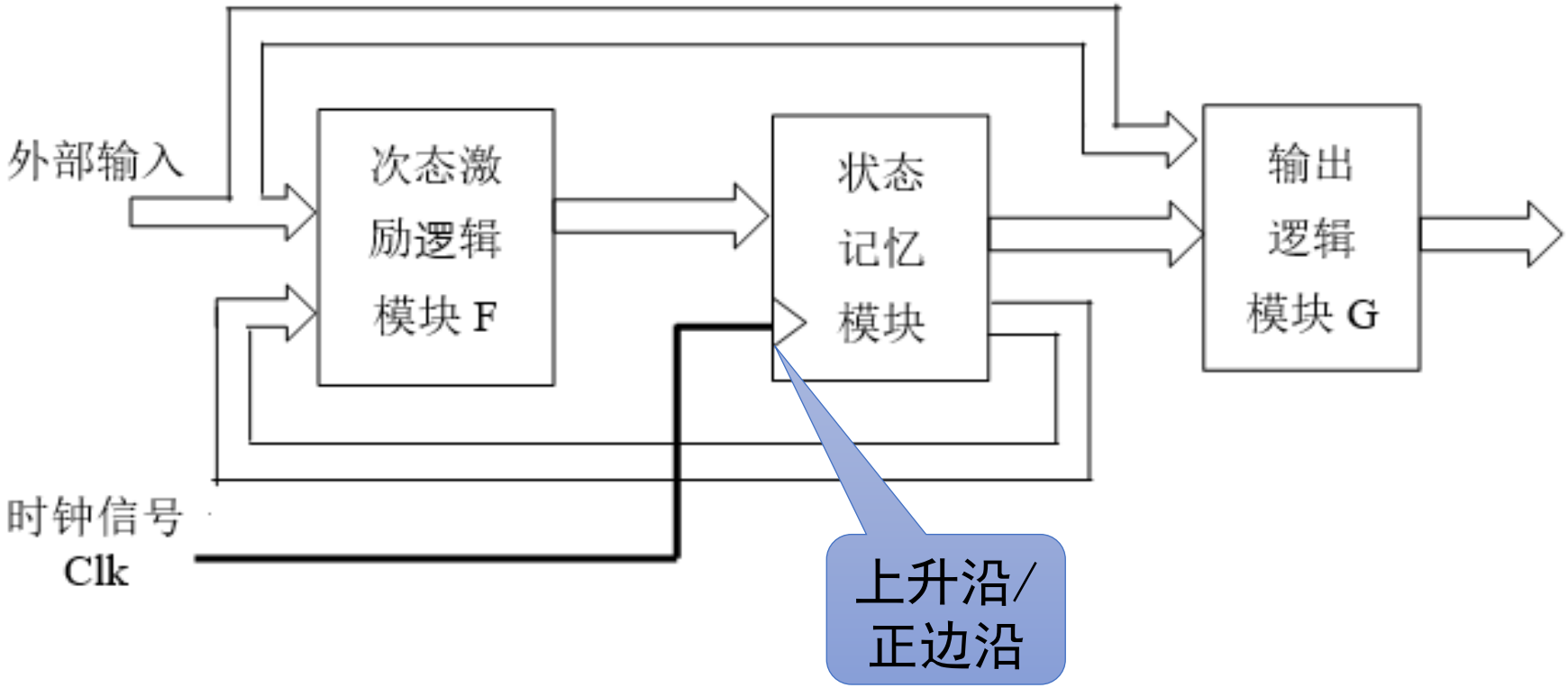
- 什么时候状态会发生变换？电平触发或边沿触发
  - 时序逻辑电路的状态多采用边沿触发方式
  - 边沿触发方式分为上升沿触发和下降沿触发两种类型



时序逻辑电路的定时

什么时候状态会发生变换？电平触发或边沿触发

- 时序逻辑电路的状态多采用边沿触发方式
- 边沿触发方式分为上升沿触发和下降沿触发两种类型



第二讲 锁存器和触发器

1. 双稳态元件
2. SR锁存器
3. D锁存器
4. D触发器
5. T触发器

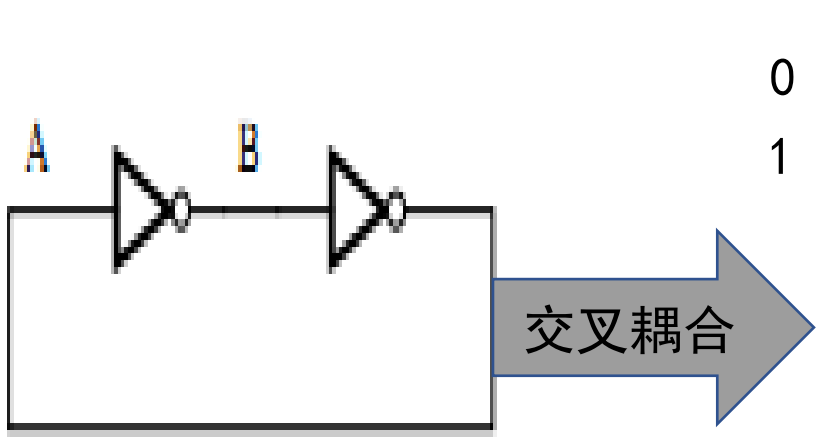
双稳态元件

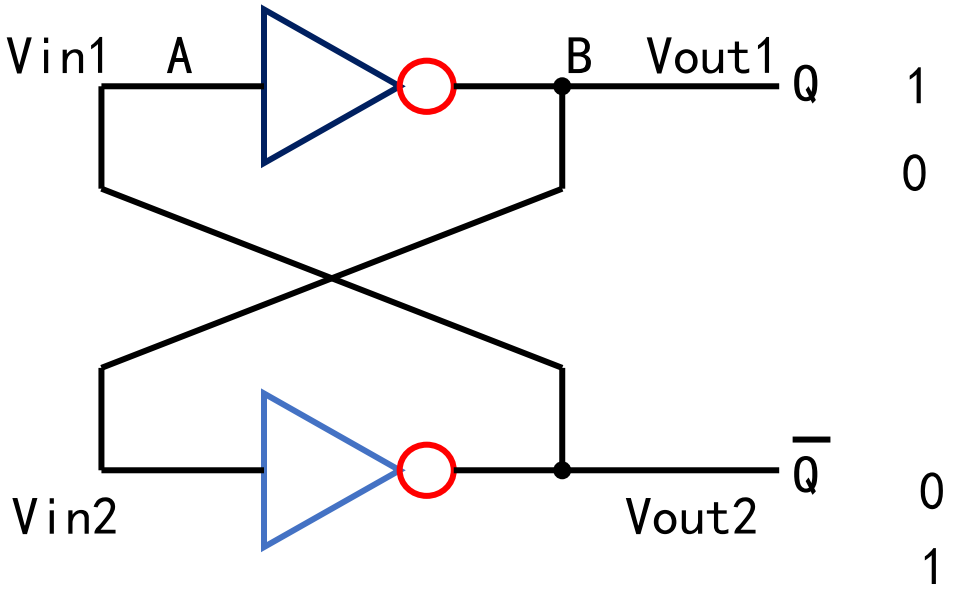
双稳态元件的简单实现

- 串联两个反相器，则反相器的输出状态不同，且保持稳定

用于存储1位二进制数据，有两个互补的输出状态

- 状态 1 ：置位(Set)状态， Q为高电平，表示存储逻辑“1”
- 状态 0 ：复位(Reset)状态，Q为低电平，表示存储逻辑“0”





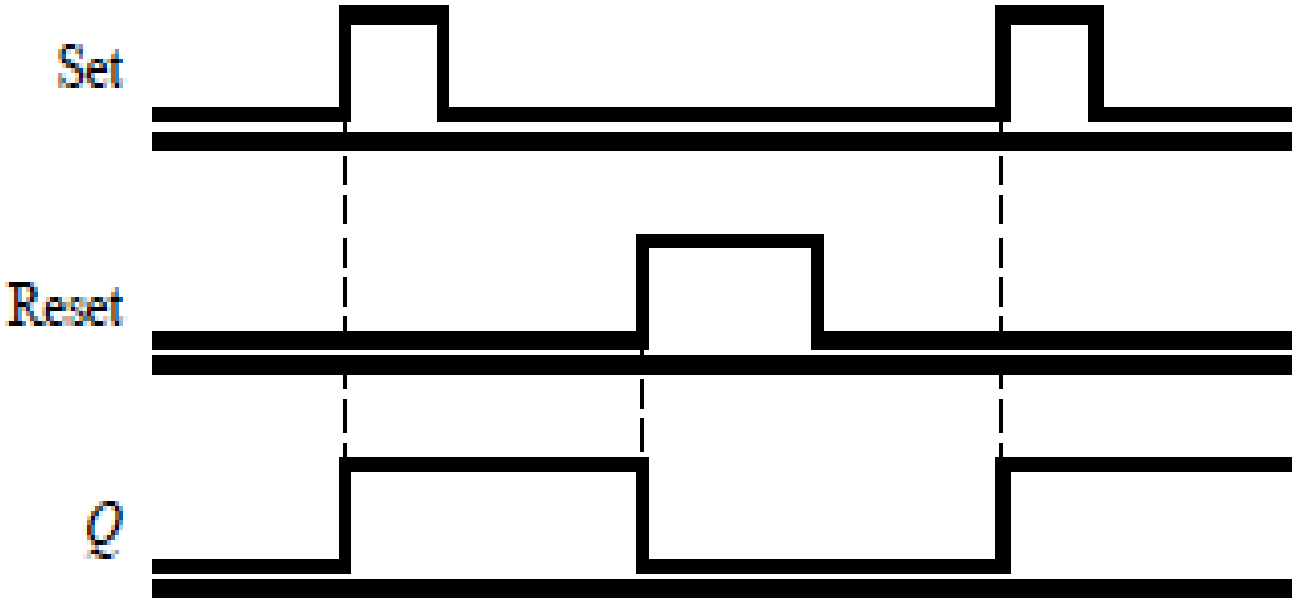
基于简单双稳态元件构建思路，可实现锁存器和触发器

双稳态元件

- 用两个反相器串联构建的双稳态元件，由于无法改变电路的状态，因此功能受到限制。
- 用1个或多个输入信号能驱动双稳态元件进入特定的稳定状态，这样的双稳态元件才能成为实用的存储元件，这些输入信号被称为激励信号或激励输入。
  - 通常根据不同的激励输入信号来命名存储元件，如SR、D、T等不同的激励输入信号。
- 根据存储元件触发方式的不同，分成两种类型：
  - 电平触发：锁存器(latch)
  - 边沿触发：触发器(flip-flop)

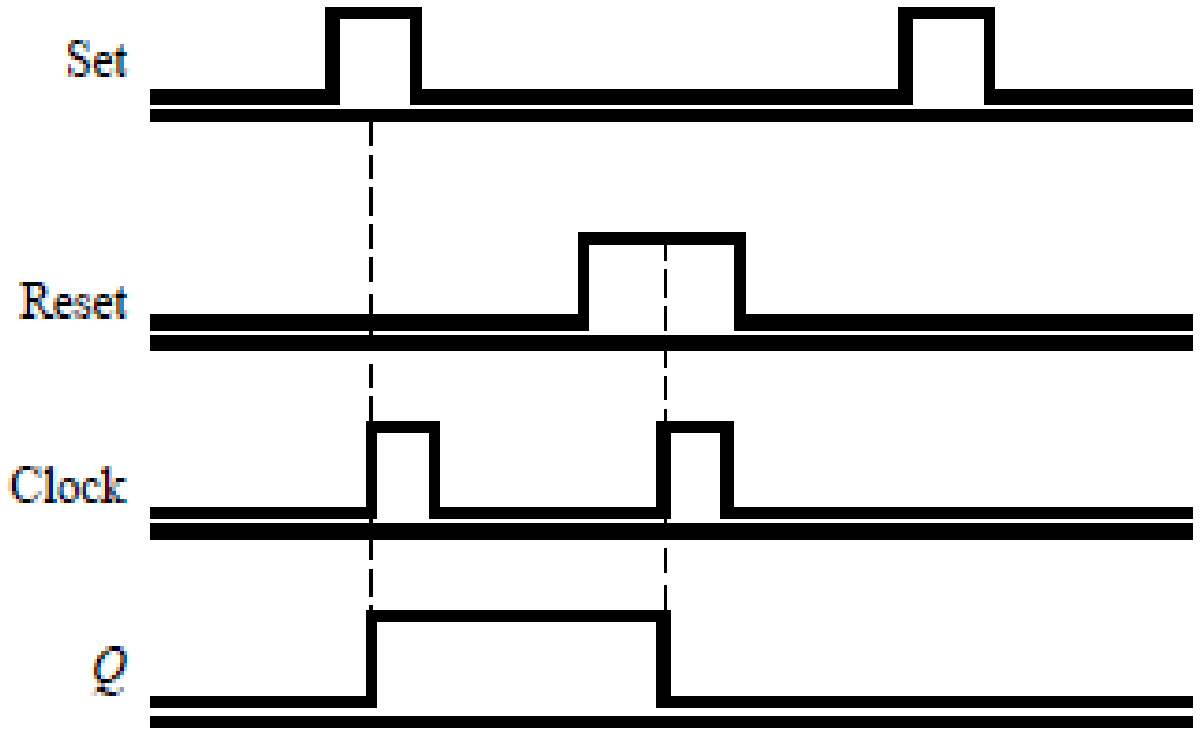
双稳态元件

- 锁存器 （latch）：通过激励输入的电平信号来控制存储元件的状态
- 置位复位锁存器(Set-Reset latch)：具有置位和复位激励信号
  - 置位激励信号Set有效时，强制存储元件的输出Q为1
  - 复位激励信号Reset有效时，强制存储元件的输出Q为0



双稳态元件

- 触发器 flip-flop
  - 具有时钟控制信号(clock)
  - 通过时钟信号的边沿来触发存储元件改变状态

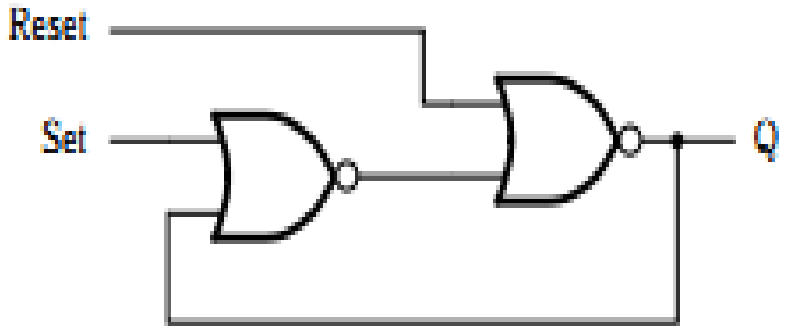


第二讲 锁存器和触发器

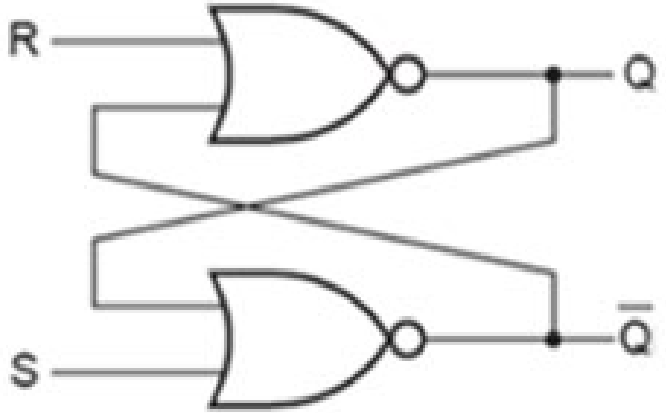
1. 双稳态元件
2. SR锁存器
3. D锁存器
4. D触发器
5. T触发器

SR锁存器

SR锁存器：使用一对交叉耦合的或非门构成双稳态电路，也称为置位-重置（复位）锁存器。S是置位输入端，R是重置输入端

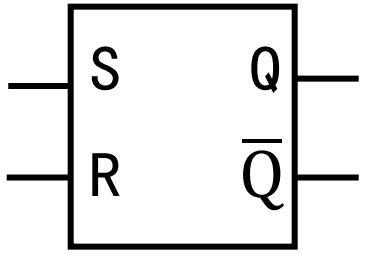


交叉耦合

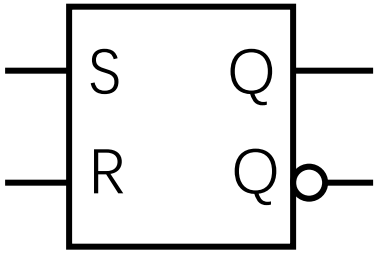


S	R	Q	$\overline{Q}$
0	0	状态不变	
0	1	0	1
1	0	1	0
1	1	禁	止

逻辑符号



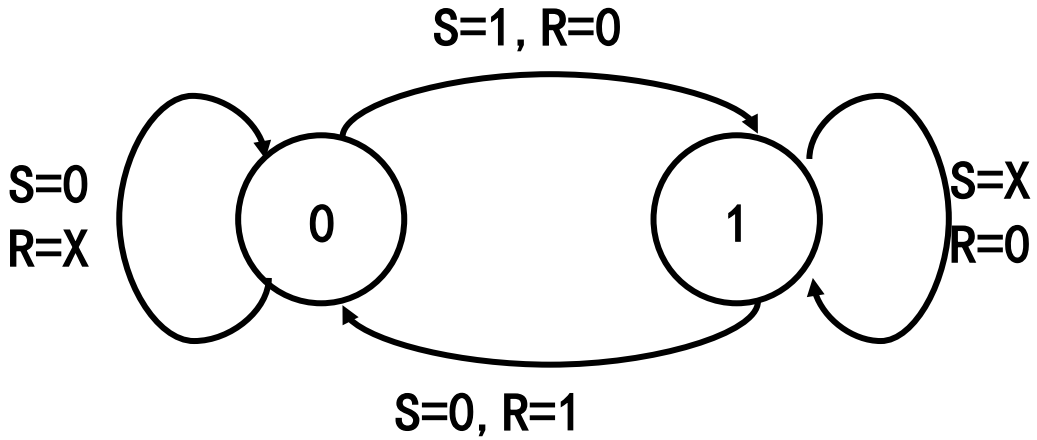
逻辑符号

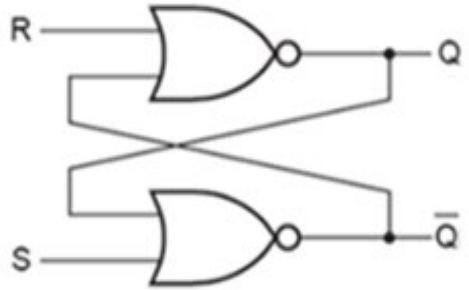


R=S=1时，Q、$\overline{Q}$状态不相反，无效

SR锁存器

- 状态图





状态表

现态 Q	输入RS 00	01	10	11
0	0	1	0	0*
1	1	1	0	0*

- 通过状态变化来描述时序电路
- 构建状态表
  - 顶部为输入信号，左侧为现态Q
  - 右侧填入次态Q*和输出信号
- 在置位态下，若R输入变为高电平，则经过两级门延迟变为复位态
- 从输入驱动信号有效开始，到输出达到稳定为止有一定的延迟，这个延迟称为触发延迟或锁存延迟。

SR锁存器

- 状态表转换成状态转移表

状态表

现态 Q	次态Q* 输入RS 00	01	10	11
0	0	1	0	0
1	1	1	0	0

状态图、状态表、特征方程之间可相互转换！

状态转移表

S	R	现态Q	次态Q*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0* d
1	1	1	0* d

- 特征方程

次态方程

$$\begin{cases} Q^* = S + \bar{R} \cdot Q \\ S \cdot R \neq 1 \quad \text{约束条件} \end{cases}$$

第二讲 锁存器和触发器

1. 双稳态元件
2. SR锁存器
3. D锁存器
4. D触发器
5. T触发器

D锁存器

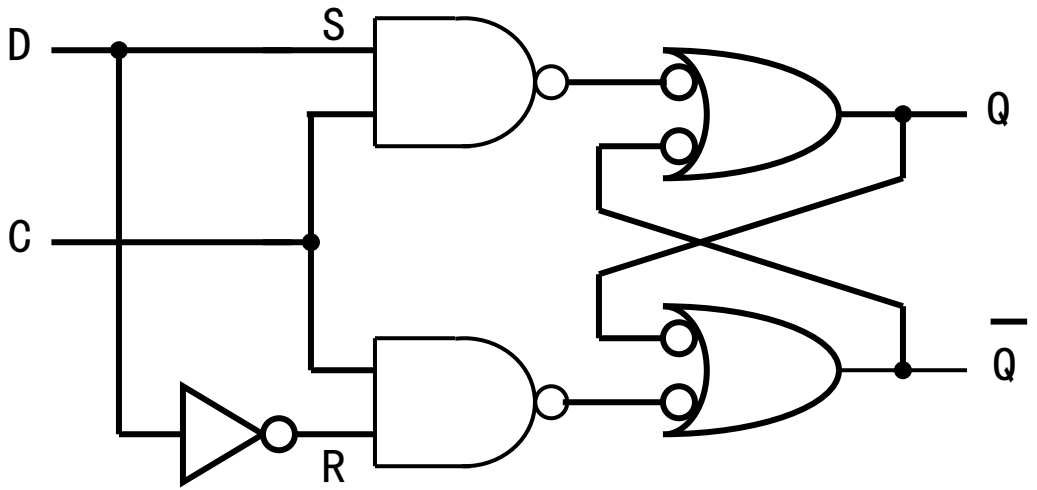
S=1, R=0时， Q=1

S=0, R=1时， Q=0

S、R输入互补

数据输入端

控制端



D锁存器功能表

C	D	Q	$\overline{Q}$
0	X	保	持
1	0	0	1
1	1	1	0

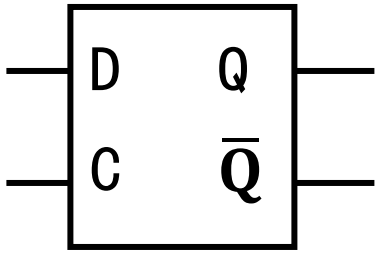
C=0时， 输出状态保持不变

C=1时，
- D = 1时，Q = 1
- D = 0时，Q = 0

} Q = D

输出随输入状态而改变

逻辑符号



只有一个数据输入端D，称为D锁存器，也称为透明锁存器

D锁存器

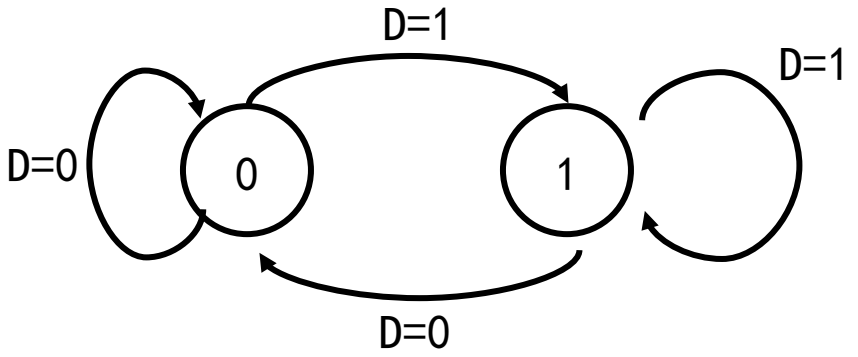
D锁存器状态表、状态图和特征方程

特征方程：Q* = D（C=1）

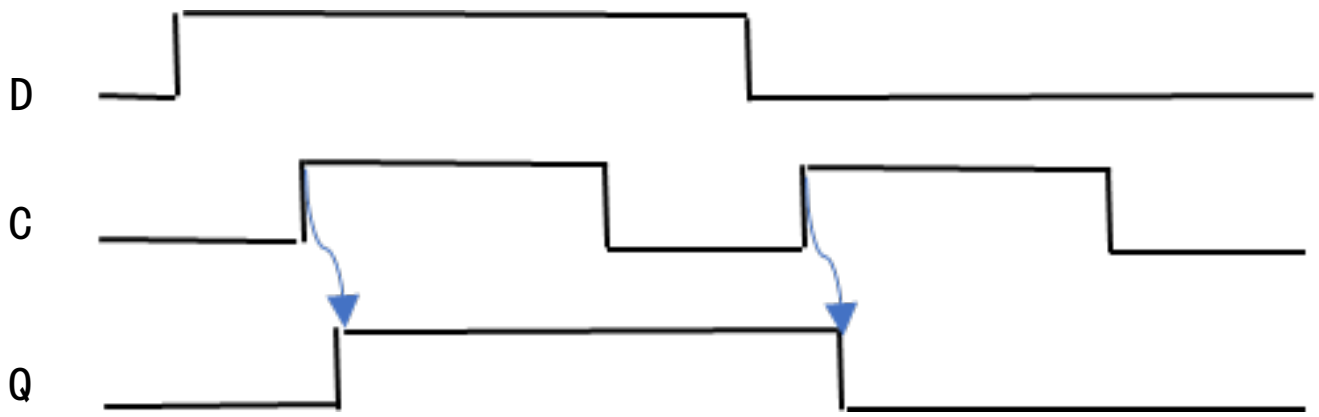
状态转移表

D	Q*
0	0
1	1

状态图



D锁存器的时序图

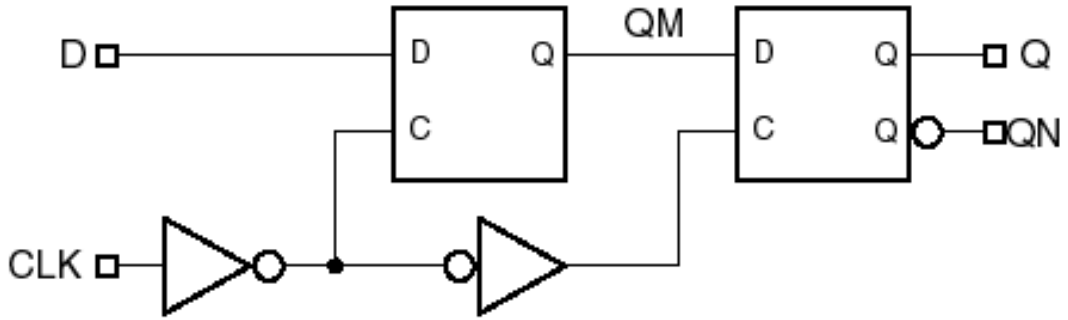


第二讲 锁存器和触发器

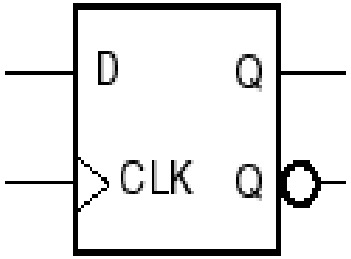
1. 双稳态元件
2. SR锁存器
3. D锁存器
4. D触发器
5. T触发器

D触发器

由一对主、从D锁存器可构成一个D触发器



D触发器符号

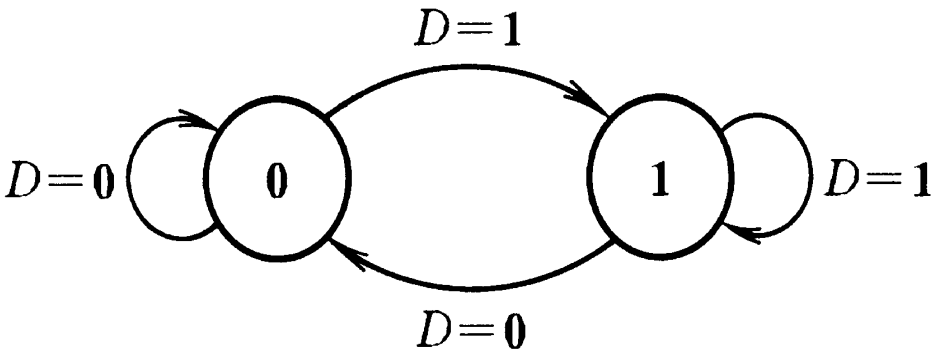


CLK	主锁存器	从锁存器
L	写入	不变
上升沿	锁存	开始写入
H	不变	写入

从锁存器只在时钟CLK的上升沿到来时采样主锁存器的输出QM的值，并确定Q和QN的输出

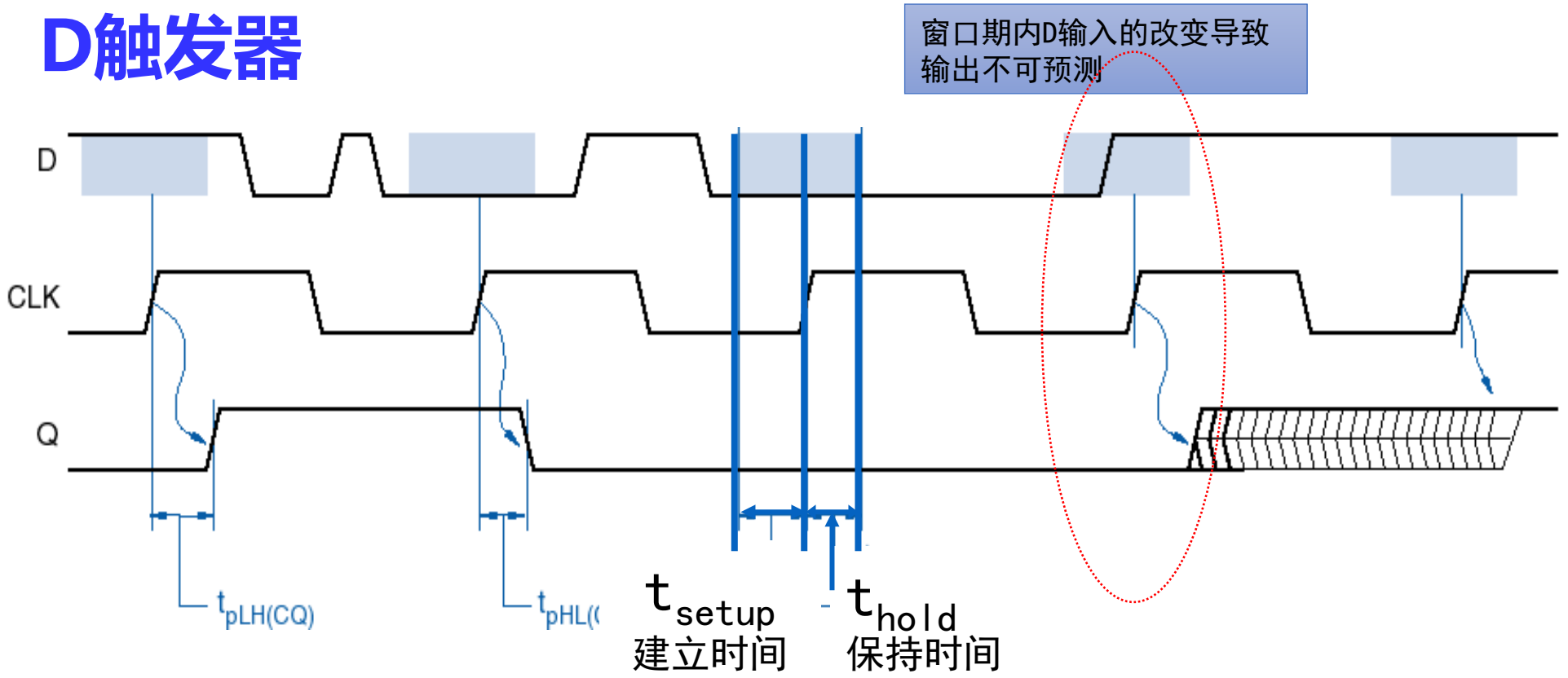
D	CLK	Q	QN
0	↑	0	1
1	↑	1	0
x	0	last Q	last QN
x	1	last Q	last QN

◆状态转移图



◆D触发器特征方程：Q* = D

D触发器



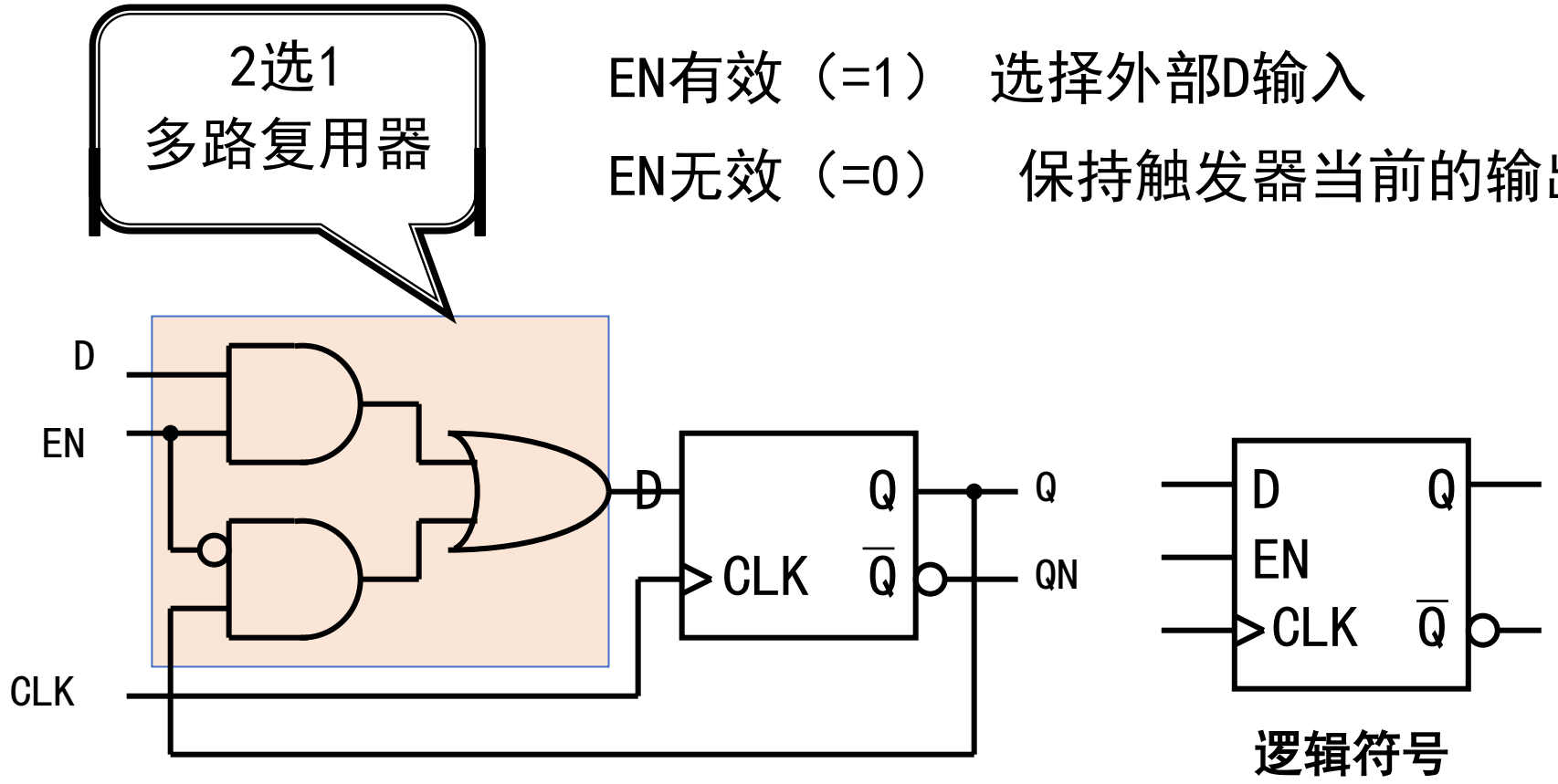
从时钟触发边沿到来，到输出端Q改变为D值的时间称为锁存延迟t_{CQ}（latch prop），即CLK→Q时间，分$t_{pLH(CQ)}$、$t_{pHL(CQ)}$两种时间

建立时间t_{setup}：输入信号D在时钟边沿到达前需稳定的时间

保持时间t_{hold}：输入信号D在时钟边沿到达后需继续稳定的时间

D触发器

带使能端的D触发器：通过使能端EN信号来控制是否在时钟信号的触发边沿进行数据的存储。



EN有效（=1）　选择外部D输入

EN无效（=0）　保持触发器当前的输出

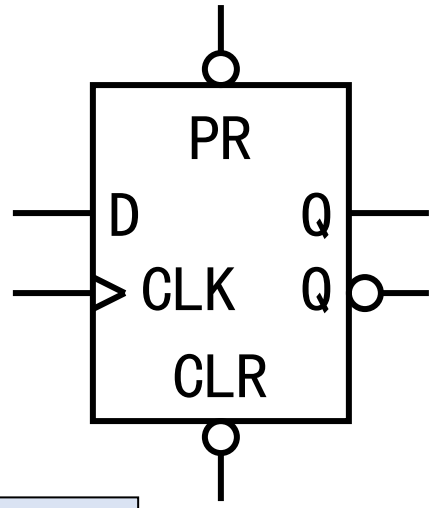
逻辑符号

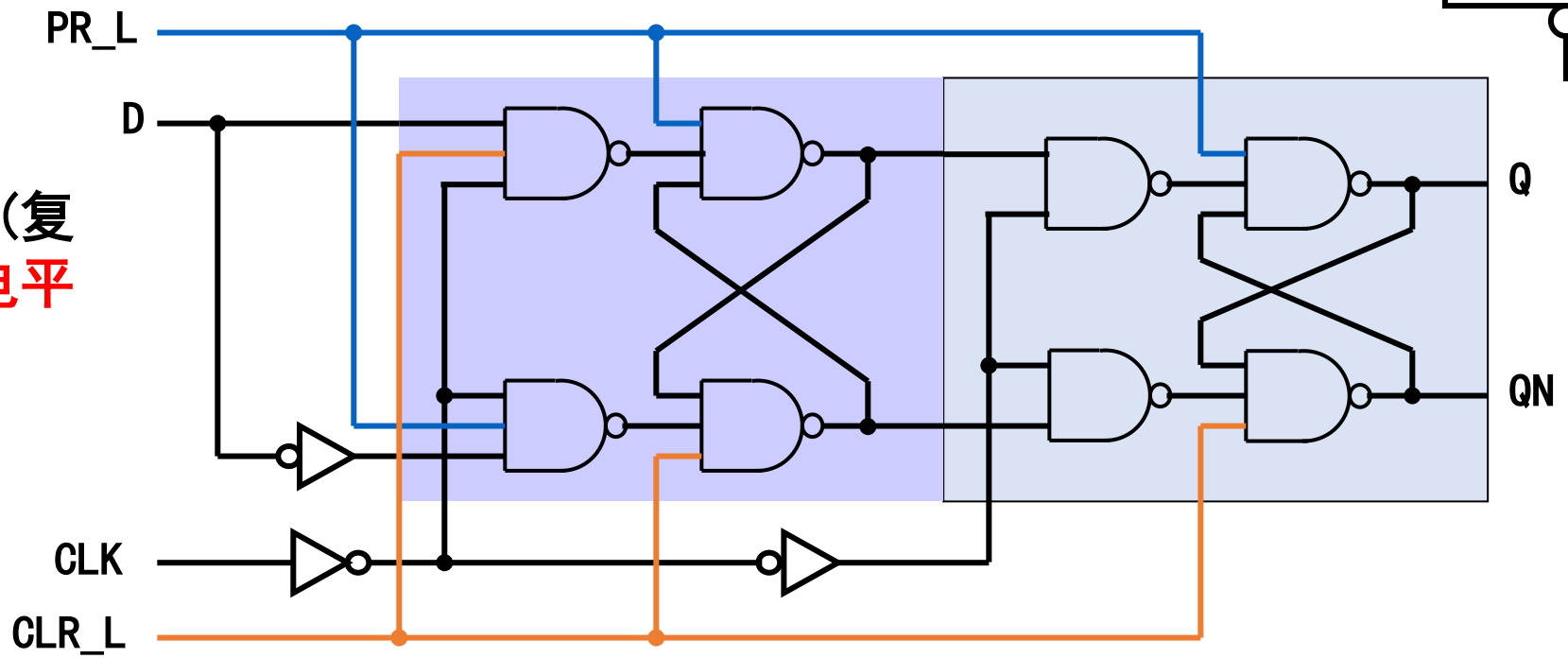
D触发器

具有预置和清零（复位）端的D触发器

- 预置端PR（preset）：将Q置1
- 清零端CLR（clear）：将Q清0

在电路工作的最开始进行置位或清0





预置端和清零(复位)端都是低电平有效信号

预置端和清零端有同步、异步之分。同步方式下只能在CLK的触发边沿进行预置和清零，异步方式下与时钟信号无关

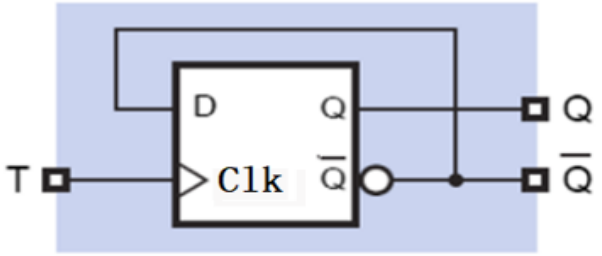
第二讲 锁存器和触发器

1. 双稳态元件
2. SR锁存器
3. D锁存器
4. D触发器
5. T触发器

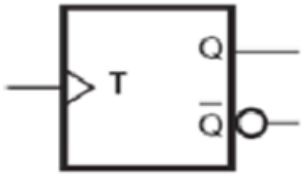
T触发器

T触发器：在每个时钟脉冲的触发边沿都会改变状态

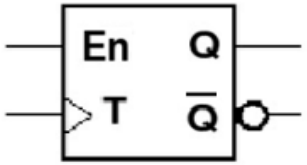
基于D触发器实现；可用于实现计数器、分频器等功能



a）T触发器原理图

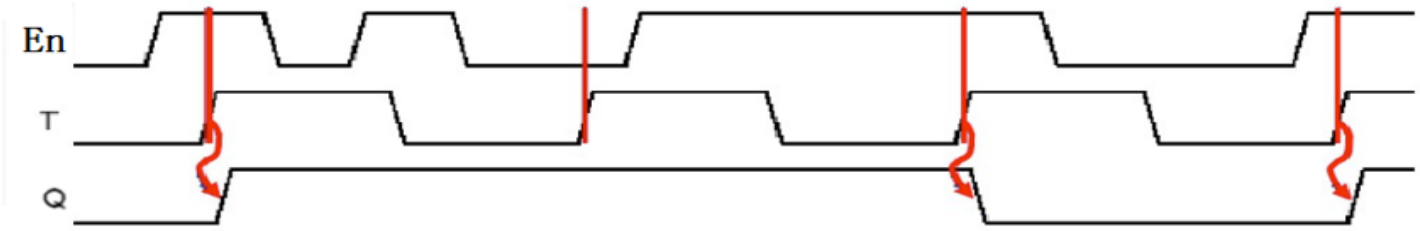


b）T触发器电路符号



C）带使能端T触发器电路符号

d）T触发器波形图



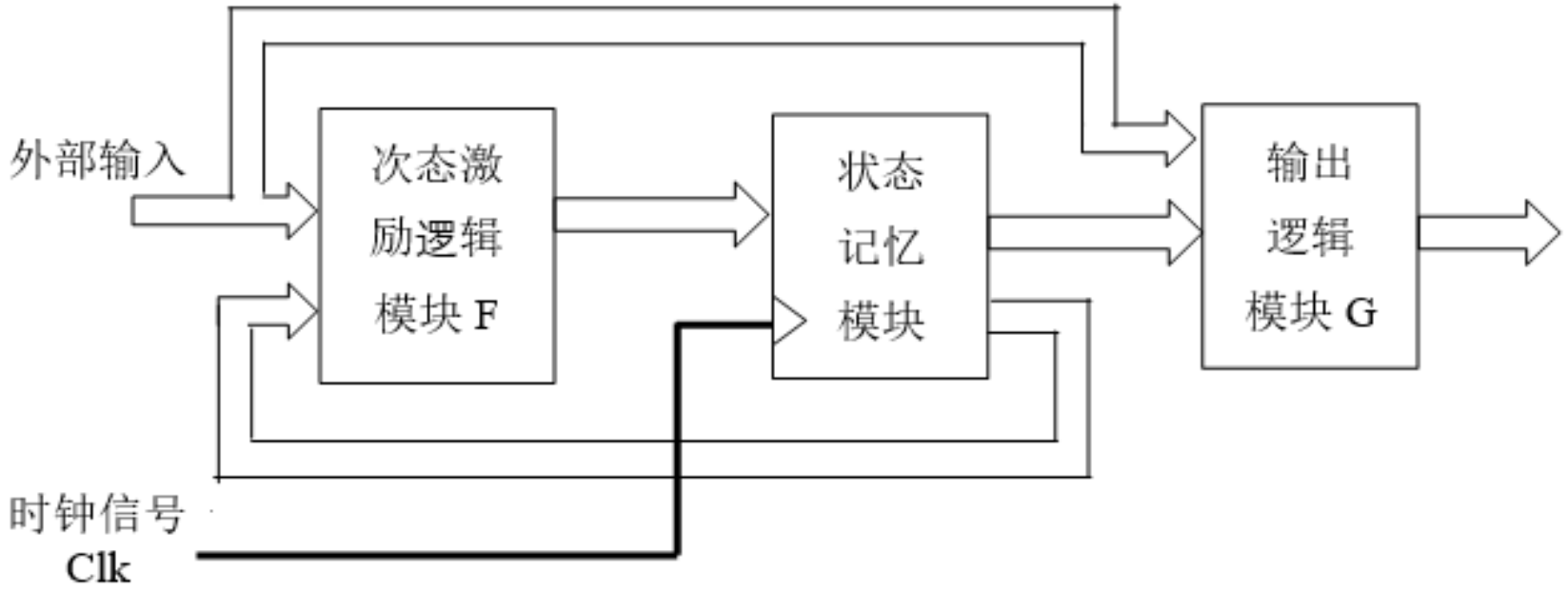
e）带使能端T触发器波形图

第三讲 同步时序逻辑设计

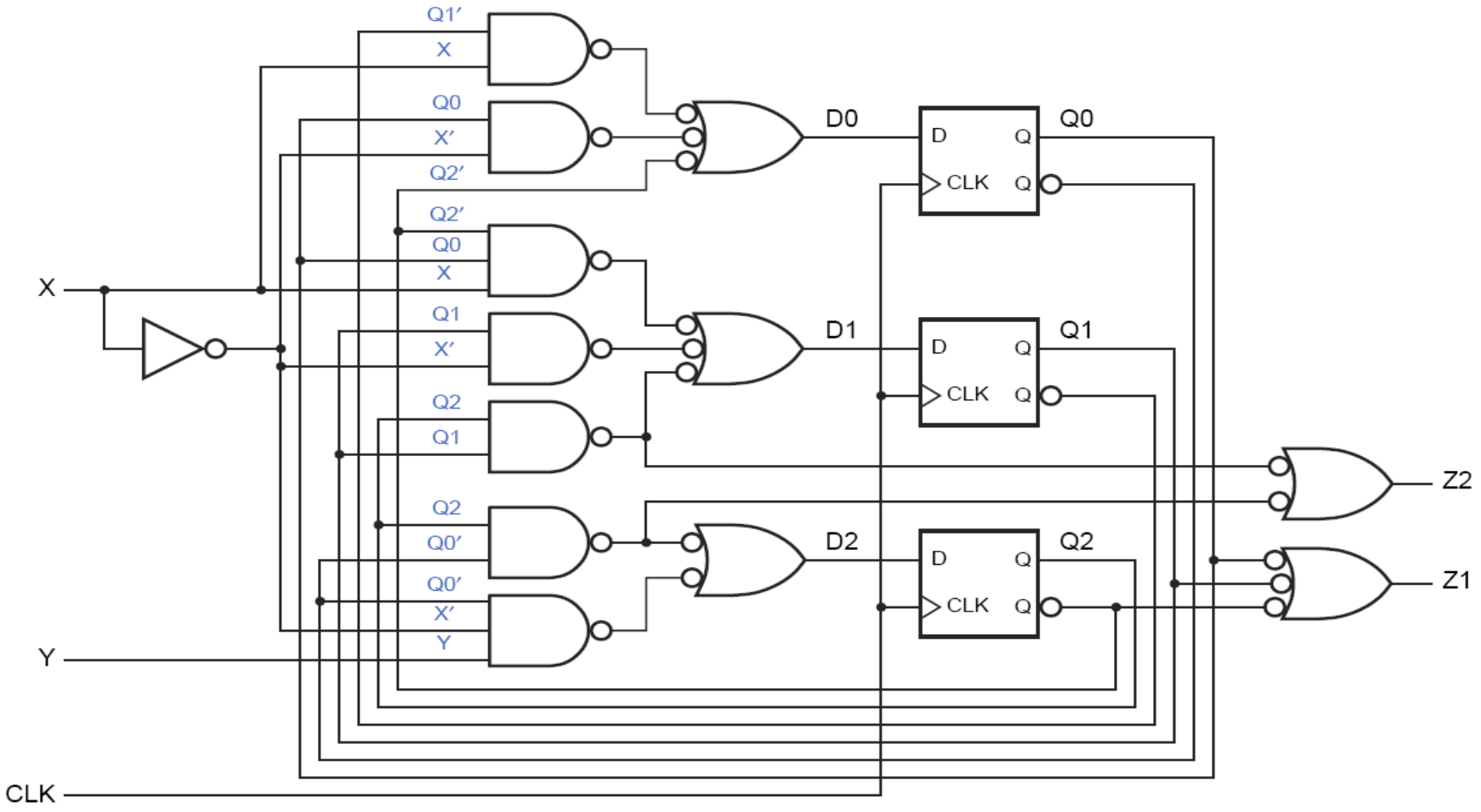
1. 同步时序逻辑设计步骤
2. 状态图/状态表设计
3. 状态化简和状态编码
4. 电路设计和分析

同步时序逻辑设计步骤

- 同步时序逻辑的设计，就是要把F和G函数设计出来。
  - F函数：外部输入+内部状态 → 次态的激励信号
    - 外部输入+内部状态 → 次态
    - 次态 → 次态的激励信号（记忆器件的次态方程）
  - G函数：外部输入+内部状态 → 输出



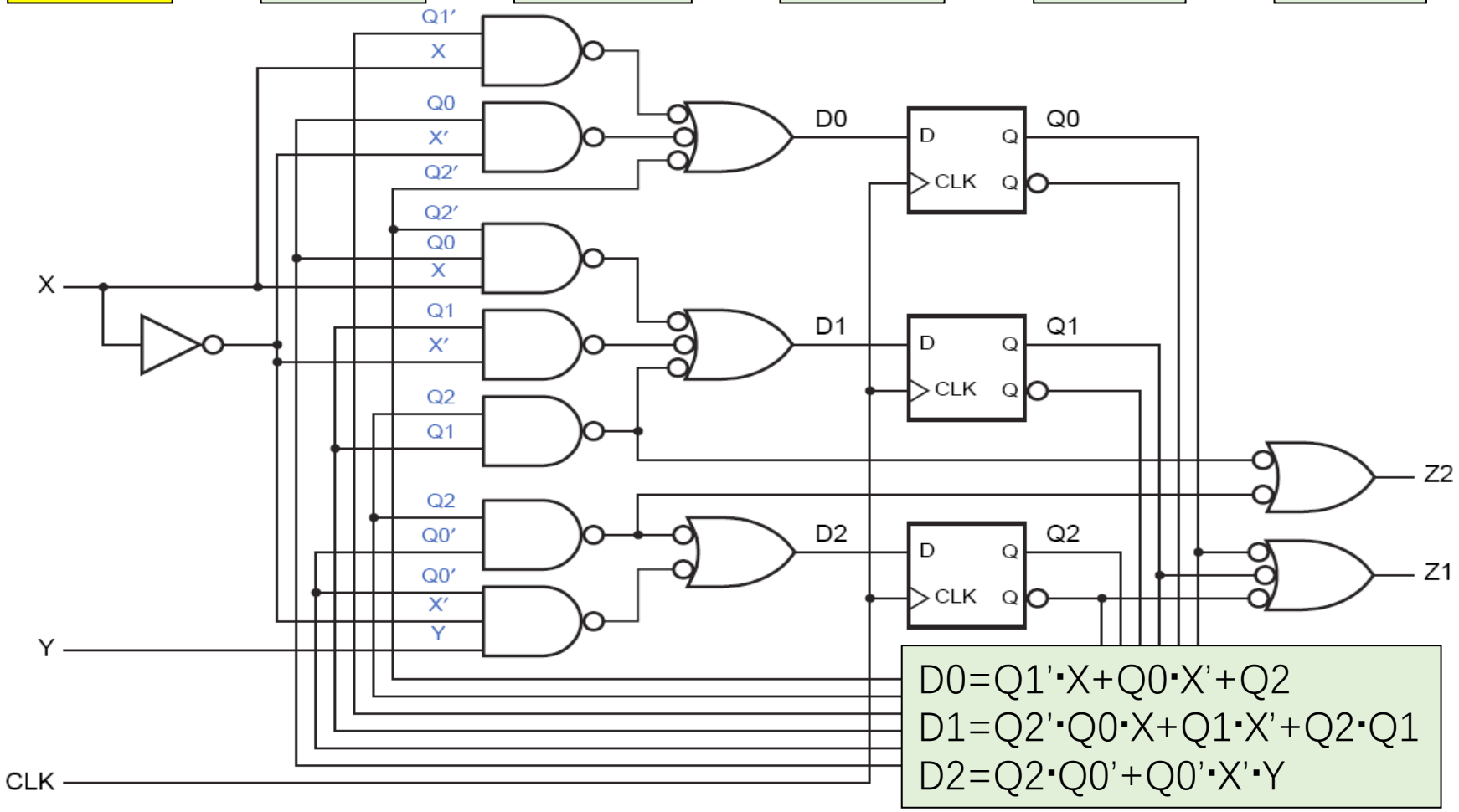
同步时序逻辑电路分析


Q1′
X
Q0
X′
Q2′
Q2′
Q0
X
Q1
X′
Q2
Q1
Q2
Q0′
Q0′
X′
Y
X
Y
CLK
D0
D1
D2
D
Q
CLK
Q
Q0
Q1
Q2
Z2
Z1

同步时序逻辑电路分析

第一步：写出激励方程

写出激励方程 → 写出转移方程 → 写出输出方程 → 得到状态转移表 → 画出状态转移图 → 画出时序图



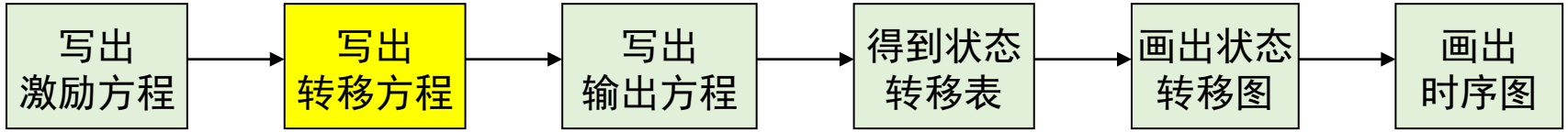
D0=Q1’·X+Q0·X’+Q2

D1=Q2’·Q0·X+Q1·X’+Q2·Q1

D2=Q2·Q0’+Q0’·X’·Y

同步时序逻辑电路分析

第二步：得到转移方程



激励方程

D0=Q1’·X+Q0·X’+Q2
D1=Q2’·Q0·X+Q1·X’+Q2·Q1
D2=Q2·Q0’+Q0’·X’·Y

D触发器的特征方程 $Q^{n+1}=D$

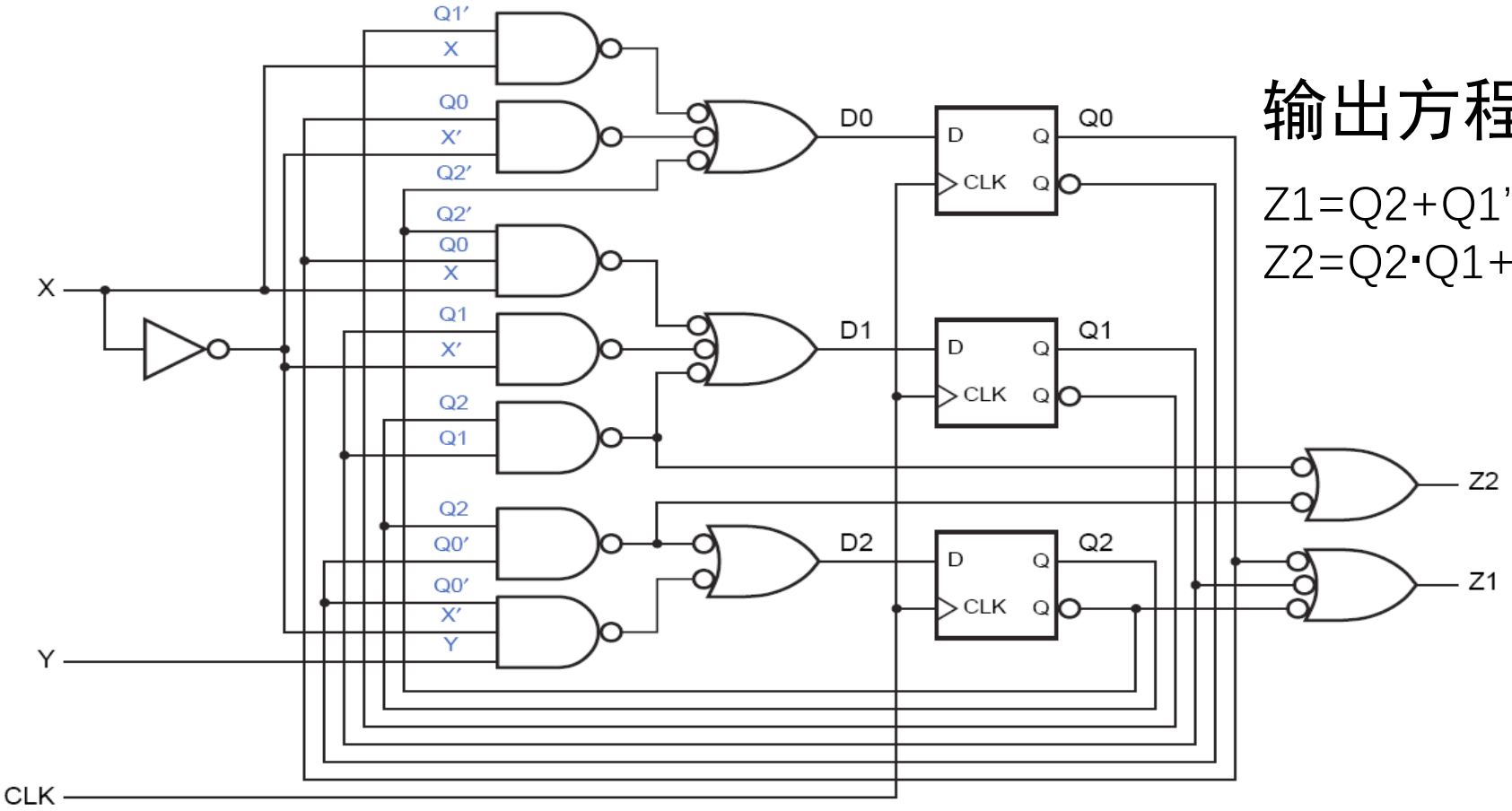
转移方程

$Q0^{n+1}$=Q1’·X+Q0·X’+Q2
$Q1^{n+1}$=Q2’·Q0·X+Q1·X’+Q2·Q1
$Q2^{n+1}$=Q2·Q0’+Q0’·X’·Y

同步时序逻辑电路分析

第三步：得到输出方程

写出激励方程 → 写出转移方程 → 写出输出方程 → 得到状态转移表 → 画出状态转移图 → 画出时序图



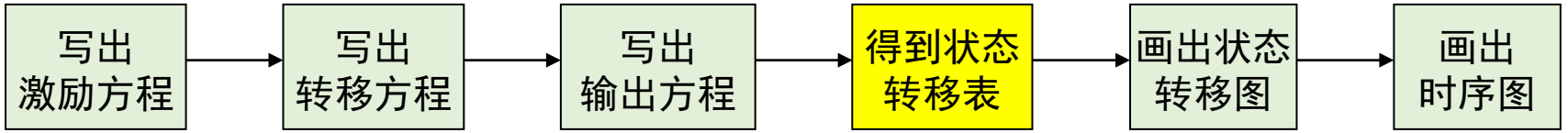
输出方程

$Z1=Q2+Q1'+Q0'$

$Z2=Q2\cdot Q1+Q2\cdot Q0'$

同步时序逻辑电路分析

第四步：构建转移表



$Q0^{n+1}=Q1'\cdot X+Q0\cdot X'+Q2$

$Q1^{n+1}=Q2'\cdot Q0\cdot X+Q1\cdot X'+Q2\cdot Q1$

$Q2^{n+1}=Q2\cdot Q0'+Q0'\cdot X'\cdot Y$

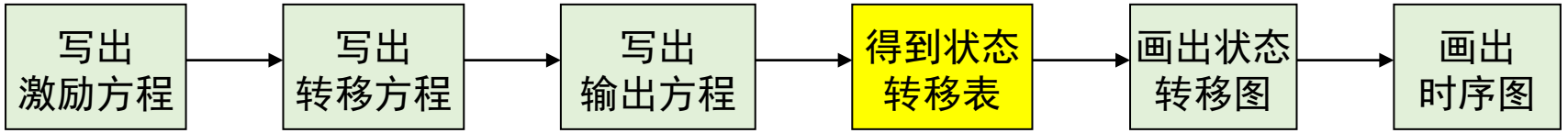
$Z1=Q2+Q1'+Q0'$

$Z2=Q2\cdot Q1+Q2\cdot Q0'$

	X Y				
Q2 Q1 Q0	00	01	10	11	Z1 Z2
000					
001					
010					
011					
100					
101					
110					
111					
	$Q2^{n+1}Q1^{n+1}Q0^{n+1}$				

同步时序逻辑电路分析

第四步：构建转移表



	XY				
Q2 Q1 Q0	00	01	10	11	Z1 Z2
000	000	100	001	001	10
001	001	001	011	011	10
010	010	110	000	000	10
011	011	011	010	010	00
100	101	101	101	101	11
101	001	001	001	001	10
110	111	111	111	111	11
111	011	011	011	011	11
	$Q2^{n+1}Q1^{n+1}Q0^{n+1}$				

转移表

	XY				
S	00	01	10	11	Z1 Z2
A	A	E	B	B	10
B	B	B	D	D	10
C	C	G	A	A	10
D	D	D	C	C	00
E	F	F	F	F	11
F	B	B	B	B	10
G	H	H	H	H	11
H	D	D	D	D	11
	S^{n+1}				

状态表

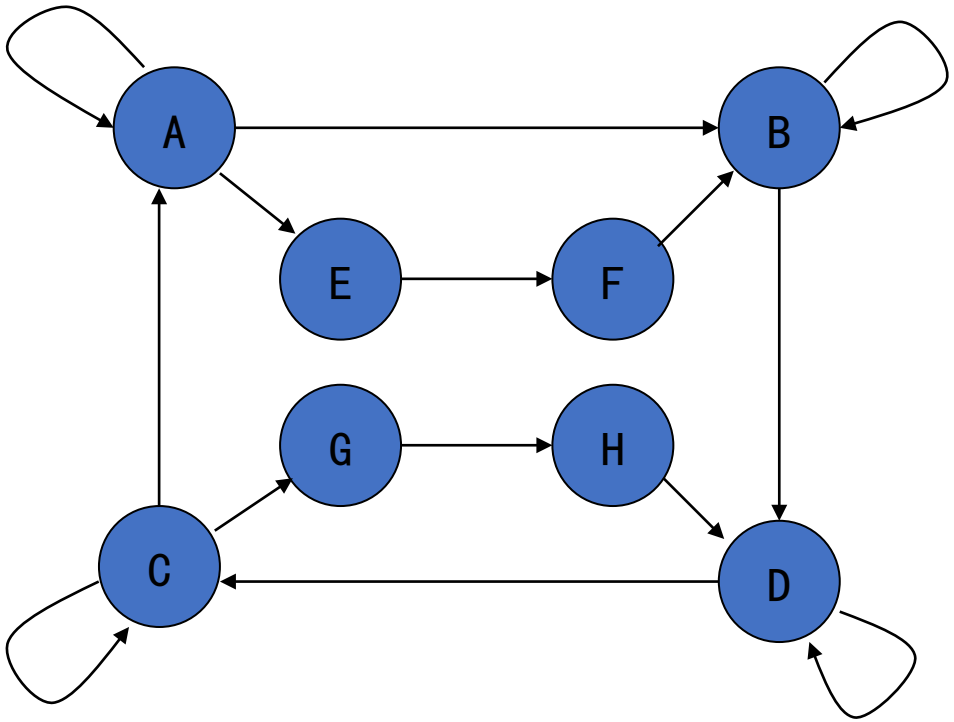
同步时序逻辑电路分析

第五步：画出状态转移图

写出激励方程 → 写出转移方程 → 写出输出方程 → 得到状态转移表 → **画出状态转移图** → 画出时序图

S	XY 00	01	10	11	Z1 Z2
A	A	E	B	B	10
B	B	B	D	D	10
C	C	G	A	A	10
D	D	D	C	C	00
E	F	F	F	F	11
F	B	B	B	B	10
G	H	H	H	H	11
H	D	D	D	D	11
	S^{n+1}				

状态表



转移图

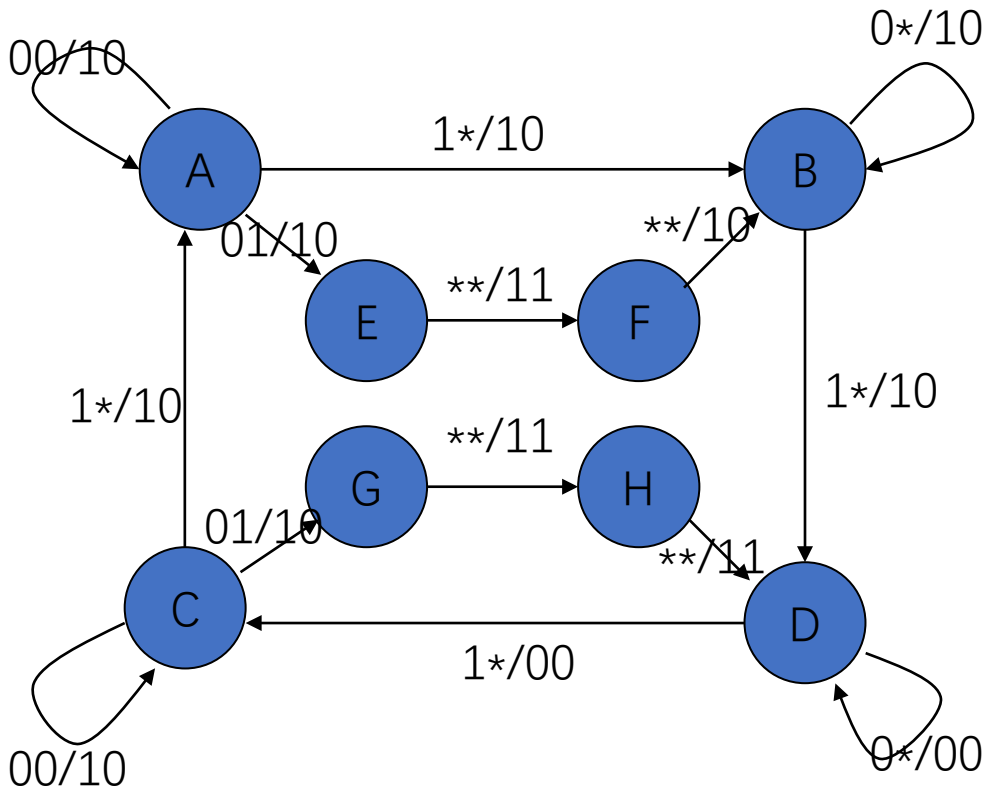
同步时序逻辑电路分析

第五步：画出状态转移图

写出激励方程 → 写出转移方程 → 写出输出方程 → 得到状态转移表 → **画出状态转移图** → 画出时序图

S	XY 00	01	10	11	Z1 Z2
A	A	E	B	B	10
B	B	B	D	D	10
C	C	G	A	A	10
D	D	D	C	C	00
E	F	F	F	F	11
F	B	B	B	B	10
G	H	H	H	H	11
H	D	D	D	D	11
	S^{n+1}				

状态表



转移图

*表示与该输入无关

转移表达式必须是互斥的，并且是完备的。

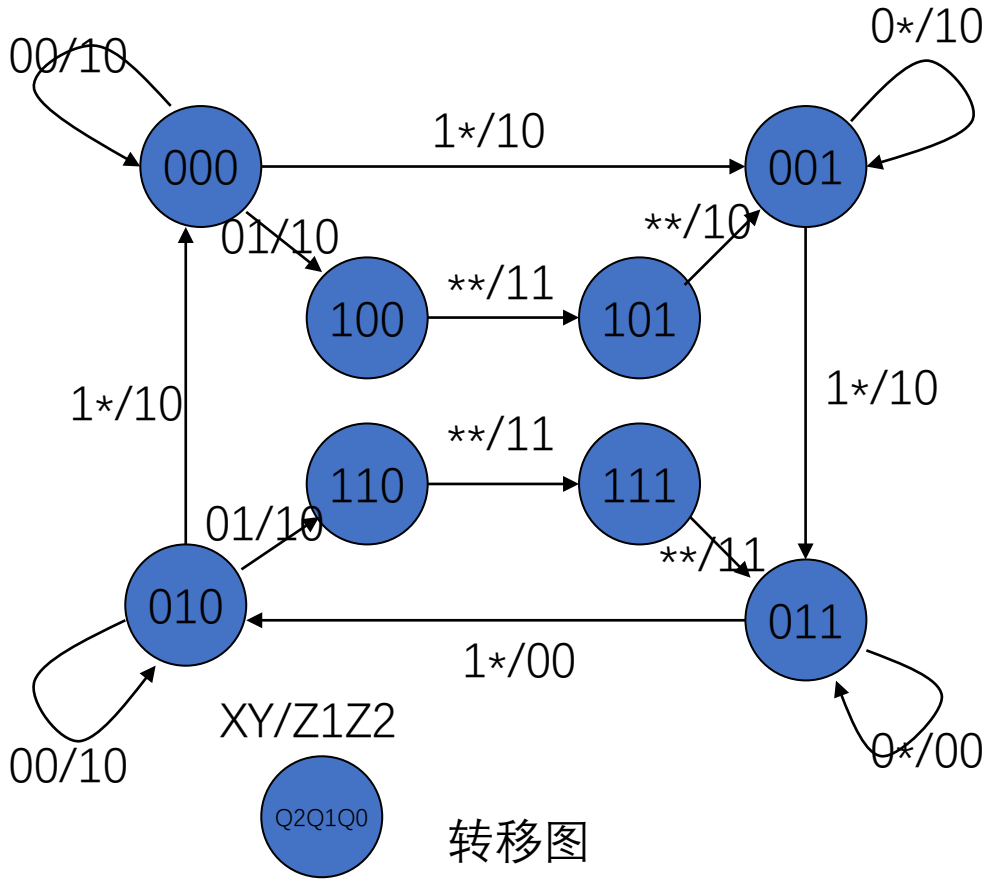
同步时序逻辑电路分析

第五步：画出状态转移图

写出激励方程 → 写出转移方程 → 写出输出方程 → 得到状态转移表 → **画出状态转移图** → 画出时序图

	XY				
Q2 Q1 Q0	00	01	10	11	Z1 Z2
000	000	100	001	001	10
001	001	001	011	011	10
010	010	110	000	000	10
011	011	011	010	010	00
100	101	101	101	101	11
101	001	001	001	001	10
110	111	111	111	111	11
111	011	011	011	011	11
	$Q2^{n+1}Q1^{n+1}Q0^{n+1}$				

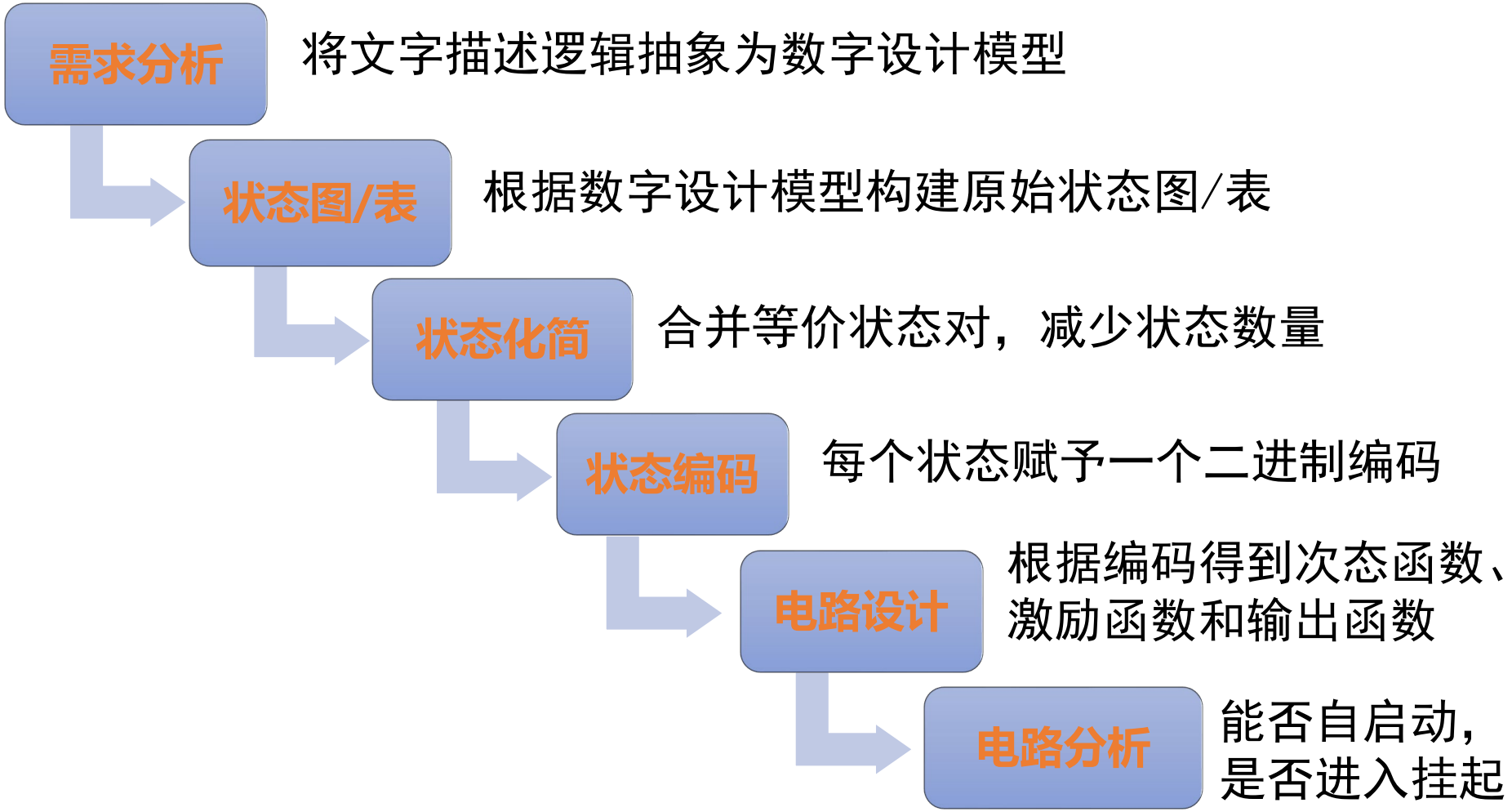
状态表



转移图

*表示与该输入无关

同步时序逻辑设计步骤

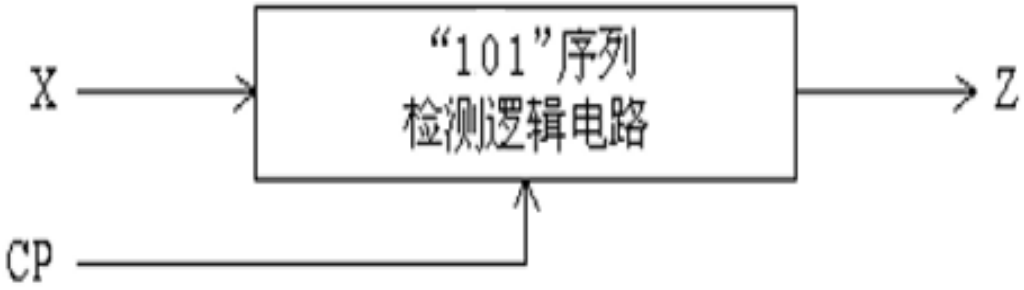

需求分析
将文字描述逻辑抽象为数字设计模型
状态图/表
根据数字设计模型构建原始状态图/表
状态化简
合并等价状态对，减少状态数量
状态编码
每个状态赋予一个二进制编码
电路设计
根据编码得到次态函数、
激励函数和输出函数
电路分析
能否自启动，
是否进入挂起

状态图/状态表设计

状态图/状态表设计：分析系统内部的状态转换关系

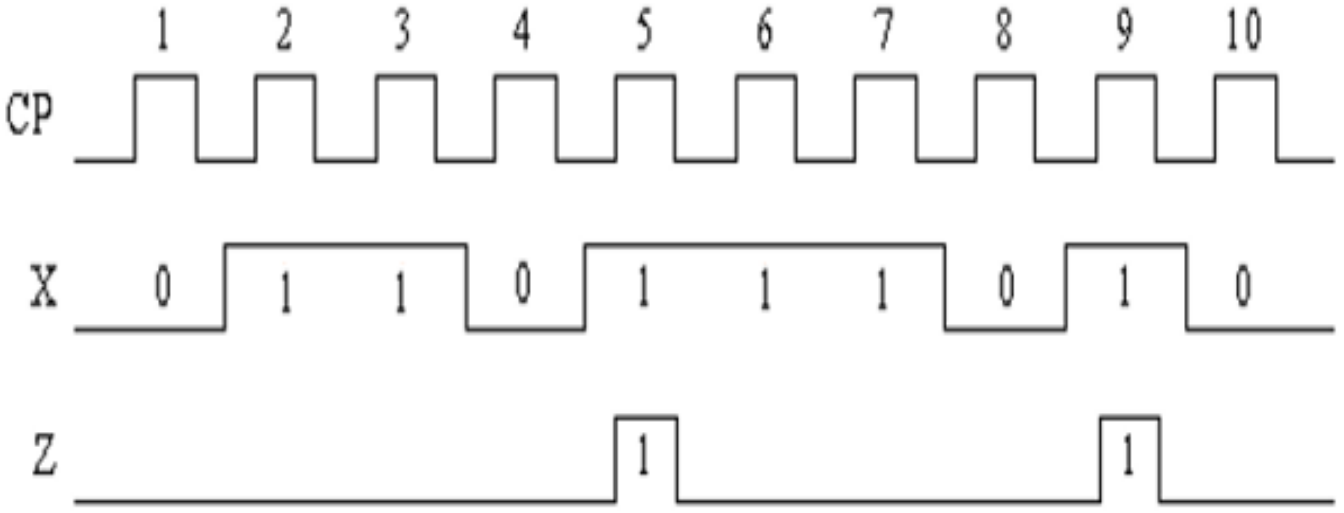
例：设计一个能检测出一连串外部输入中是否出现了0/1序列“101”的状态机。

1. 需求分析



1位输入端X；1位输出端Z。

CP脉冲到来时，根据输入端X的当前输入值，确定输入序列中是否出现“101”。若是，则输出Z为1；否则Z为0。



状态图/状态表设计

2. 构建状态图/表

I. 设定电路初始状态；

II. 从初始状态开始，分析每一个状态在不同输入作用下的状态转移情况和输出取值；

III. 如果某状态下出现的输出响应（次态、输出）不能用已有状态表示，则产生新的状态；

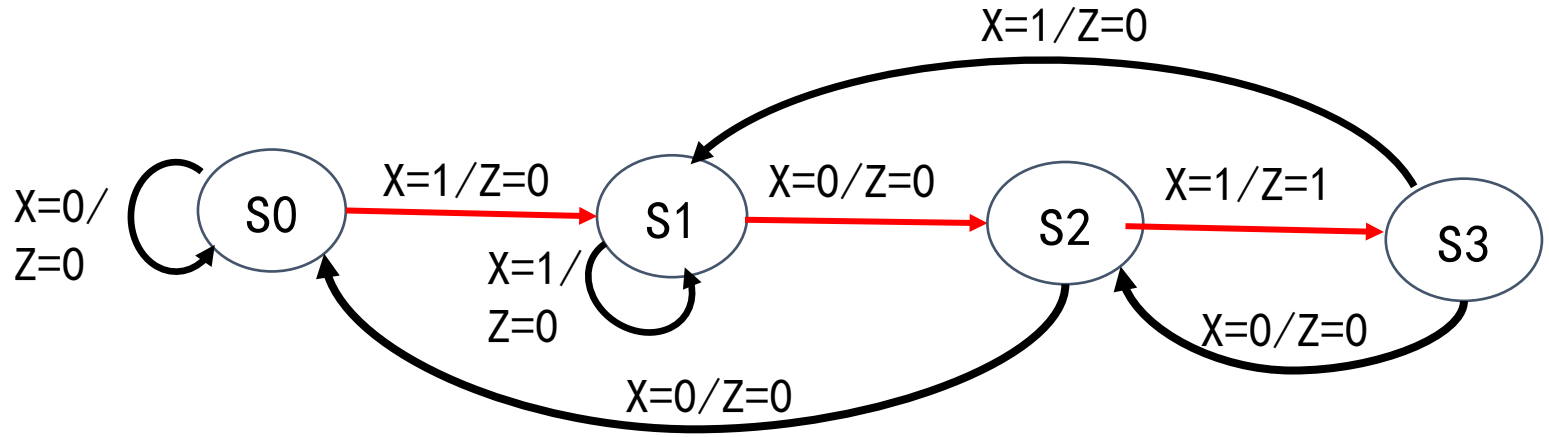
IV. 重复第II、III两步，直到不产生新状态为止。

S0：初始状态，等待接收输入

S1：接收到“101”序列中第一个1

S2：接收到该序列中的10

S3：接收到一个“101”序列



状态图/状态表设计

2. 构建状态图/表
   - 根据状态图构建状态表
     X：输入数据；Z：检测结果
     S：当前状态；S* ：次态

状态表

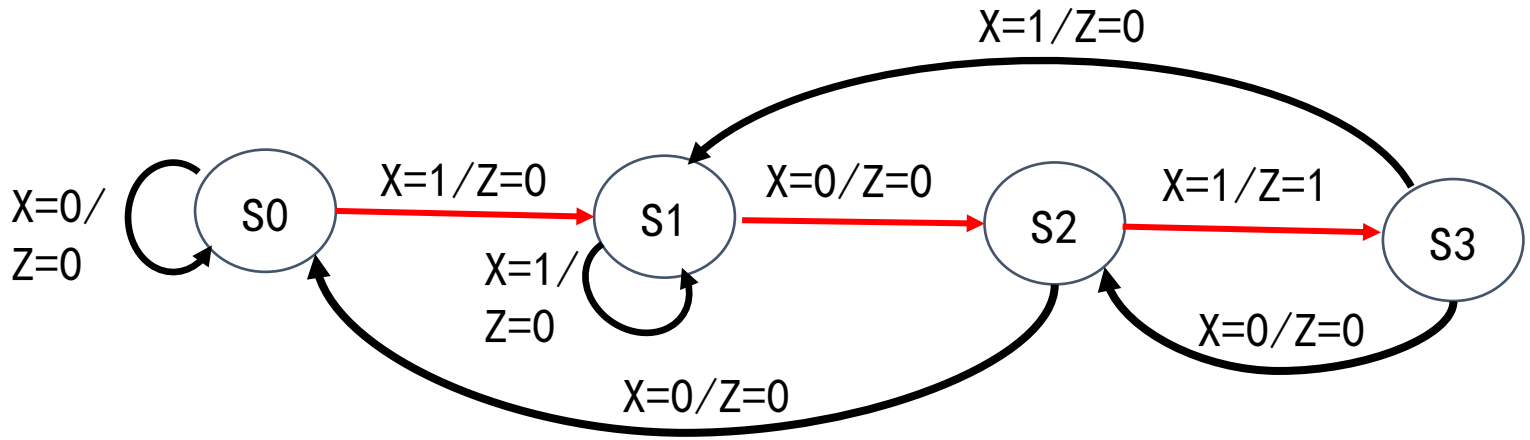
现态S	S*/Z	
	X=0	X=1
S0	S0/0	S1/0
S1	S2/0	S1/0
S2	S0/0	S3/1
S3	S2/0	S1/0

S0：初始状态，等待接收输入

S1：接收到“101”序列中第一个1

S2：接收到该序列中的10

S3：接收到一个“101”序列



状态图/状态表设计

2. 构建状态图/表

- 构建状态图/表时，状态转移需满足下列两个条件：

  互斥性：从每个状态出发的所有状态转换路径上的转换条件都是互斥的，也即任意两个转移表达式的逻辑与等于0。

  完备性：从每个状态出发的所有状态转换路径上的转移表达式的逻辑或等于1（逻辑真）。

  本例中，转移条件分别是X=0和X=1，满足互斥性和完备性。

- 在状态图中，也可以使用逻辑表达式来表示转移条件。本例中，可以使用X和$\overline{X}$分别表示输入X=1和X=0。

状态图/状态表设计

2. 构建状态图/表

- 直接构建状态表

现态	次态/输出	
	X=0	**X=1**
a(初态)	b/0	c/0
b(0)	b/0	c/0
c(1)	f/0	c/0
f(10)	b/0	c/1

现态	次态/输出	
	X=0	**X=1**
a(初态)	b/0	c/0
b(0)	d/0	e/0
c(1)	f/0	g/0
d(00)	d/0	e/0
e(01)	f/0	g/0
f(10)	d/0	e/1
g(11)	f/0	g/0

状态图/状态表设计

2. 构建状态图/表

- 直接构建状态表

现态	次态/输出	
	X=0	X=1
a(初态)	b/0	c/0
b(0)	b/0	c/0
c(1)	f/0	c/0
f(10)	b/0	c/1

现态	次态/输出	
	X=0	X=1
a(初态)	a/0	c/0
c(1)	f/0	c/0
f(10)	a/0	c/1

状态表

现态S	S*/Z	
	X=0	X=1
S0	S0/0	S1/0
S1	S2/0	S1/0
S2	S0/0	S3/1
S3	S2/0	S1/0

状态表

现态S	S*/Z	
	X=0	X=1
S0	S0/0	S1/0
S1	S2/0	S1/0
S2	S0/0	S1/1

状态化简和状态编码

- **状态化简**
  - 合并等价状态，以得到更加精简的状态表
  - 两个状态等价指在所有输入组合下，它们的输出相同且次态相同或次态等价
  - 等价关系具有传递性

    –例如，若状态A和B等价，同时B和C等价，则A和C也等价。状态A、B和C属于一个等价类，可以合并为一个状态。

状态表

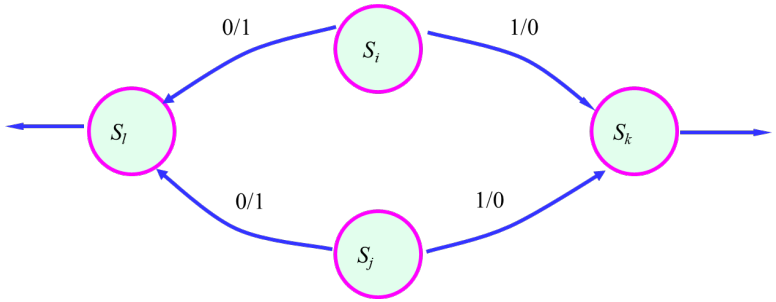
现态S	S*/Z	
	X=0	X=1
S0	S0/0	S1/0
S1	S2/0	S1/0
S2	S0/0	S3/1
S3	S2/0	S1/0

S1和S3构成等价类，可合并化简后，有3个状态

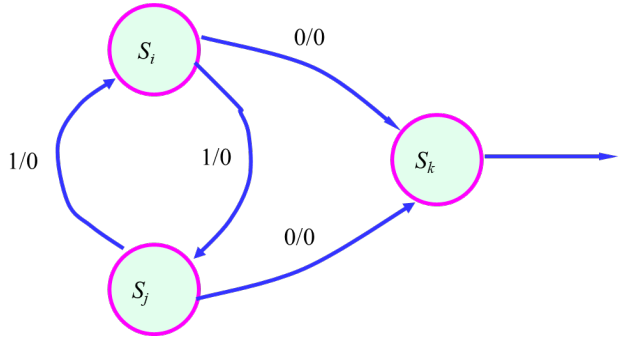
现态S	S*/Z	
	X=0	X=1
S0	S0/0	S1/0
S1	S2/0	S1/0
S2	S0/0	S1/1

状态化简和状态编码

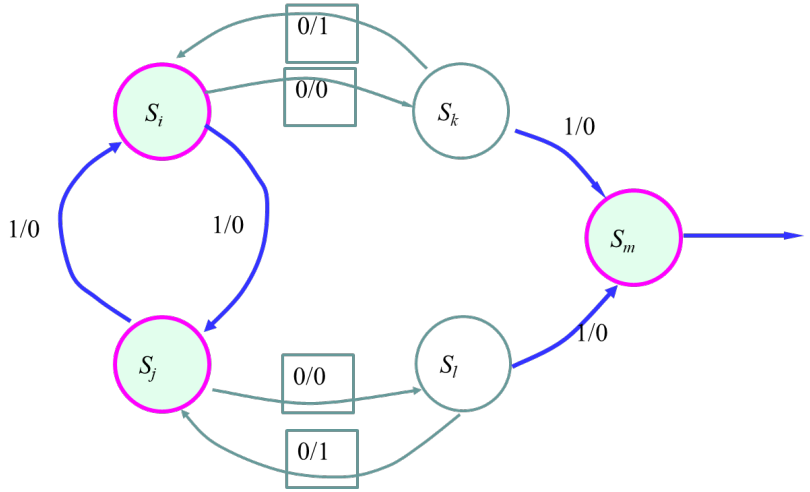
- **状态化简**--等价关系判定(输出相同的前提下)*



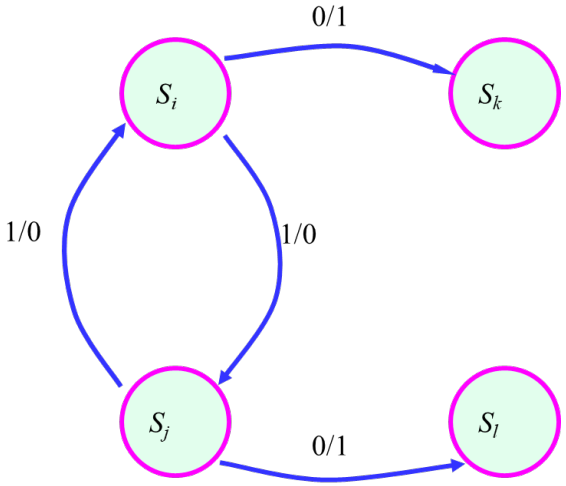
(1) 次态相同;



(2) 次态交错;



(3) 次态循环;



(4) 次态对等价。

状态化简和状态编码

状态编码

- 对状态表中每个状态赋予唯一的二进制编码，也称状态赋值
  - 不同状态编码方案，电路设计不一样，存在优劣之分。
  - 寻找最优编码方案是一个非常复杂的问题
- 通常在具体设计时采用相邻法寻求次优编码方案：

  准则1：若两个状态的次态相同，则其对应编码应尽量相邻

  准则2：同一个现态的各个次态其编码应尽量相邻

  准则3：若两个现态的输出相同，则它们的编码应尽量相邻

现态S	S*/Z	
	X=0	X=1
S0	S0/0	S1/0
S1	S2/0	S1/0
S2	S0/0	S1/1

根据准则1：S0和S2可相邻

根据准则2： S0和S1、S1和S2可相邻

根据准则3： S0和S1可相邻

根据不同的取舍可得到不同编码方案

若S0和S1、S1和S2相邻，则编码方案如下：S0:00，S1:01，S2:11

若S0和S2、S0和S1相邻，则编码方案如下：S0:00，S1:01，S2:10

电路设计与分析

- 在选定的状态编码方案基础上进行电路设计
  - 生成状态转移表

对于前面的例子，若编码方案为S0:00，S1:01，S2:11，则得到右边的状态转移表

Y1Y0	Y1*Y0*/Z	
	X=0	X=1
00	00/0	01/0
01	11/0	01/0
11	00/0	01/1

						Q*=D		Q*=JQ'+K'Q			
X	Y1	Y0	Y1*	Y0*	Z	D1	D0	J1	K1	J0	K0
0	0	0	0	0	0	0	0	0	d	0	d
0	0	1	1	1	0	1	1	1	d	d	0
0	1	0	d	d	d	d	d	d	d	d	d
0	1	1	0	0	0	0	0	d	1	d	1
1	0	0	0	1	0	0	1	0	d	1	d
1	0	1	0	1	0	0	1	0	d	d	0
1	1	0	d	d	d	d	d	d	d	d	d
1	1	1	0	1	1	0	1	d	1	d	0

电路设计与分析

- 在选定的状态编码方案基础上进行电路设计
  - 生成状态转移表

对于前面的例子，若编码方案为S0:00，S1:01，S2:11，则得到右边的状态转移表

X	Y1	Y0	Y1*	Y0*	Z
0	0	0	0	0	0
0	0	1	1	1	0
0	1	0	d	d	d
0	1	1	0	0	0
1	0	0	0	1	0
1	0	1	0	1	0
1	1	0	d	d	d
1	1	1	0	1	1

- 根据状态转移表，推导次态逻辑函数和输出逻辑函数
- 次态函数/次态方程为：

$$Y1^*=\overline{Y1}\cdot Y0\cdot\overline{X}$$

$$Y0^*=\overline{Y1}\cdot Y0\cdot\overline{X}+X\cdot(\overline{Y1}\cdot\overline{Y0}+\overline{Y1}\cdot Y0+Y1\cdot Y0)$$

$$=\overline{Y1}\cdot Y0+X\cdot\overline{Y1}+X\cdot Y0$$

将无关项编码Y1Y0=10引入化简，则$Y0^*=\overline{Y1}\cdot Y0+X$

- 输出函数/输出方程为：

$$Z=Y1\cdot Y0\cdot X+Y1\cdot\overline{Y0}\cdot X=Y1\cdot X$$

电路设计与分析

- **根据次态函数和选择的状态记忆单元（触发器），推导出激励函数**
  - 假设采用D触发器，其特征方程Q* = D，则：

    D1=Y1*=$\overline{Y1}$•Y0•$\overline{X}$

    D0=Y0*= $\overline{Y1}$•Y0+ X
- 输出函数为：Z=Y1 •X

						Q*=D		Q*=JQ'+K'Q			
X	Y1	Y0	Y1*	Y0*	Z	D1	D0	J1	K1	J0	K0
0	0	0	0	0	0	0	0	0	d	0	d
0	0	1	1	1	0	1	1	1	d	d	0
0	1	0	d	d	d	d	d	d	d	d	d
0	1	1	0	0	0	0	0	d	1	d	1
1	0	0	0	1	0	0	1	0	d	1	d
1	0	1	0	1	0	0	1	0	d	d	0
1	1	0	d	d	d	d	d	d	d	d	d
1	1	1	0	1	1	0	1	d	1	d	0

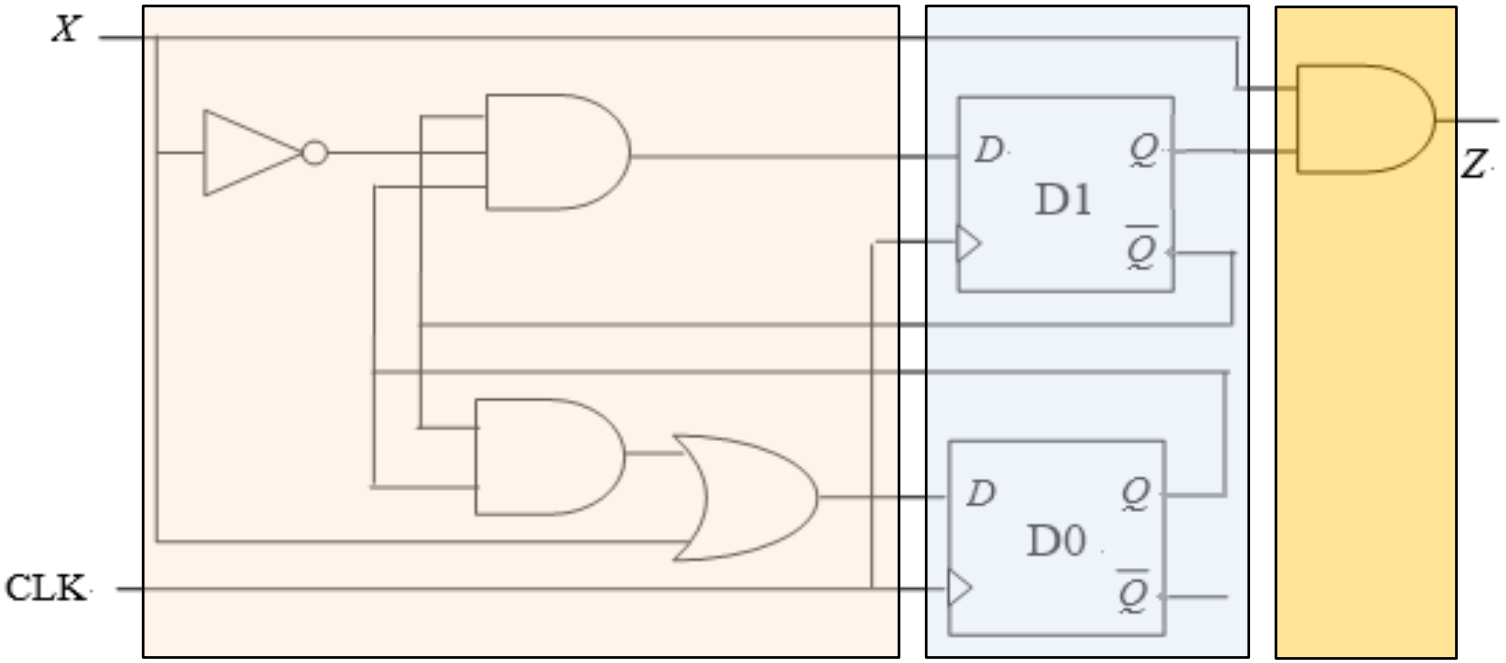
电路设计与分析

- 根据次态函数和选择的状态记忆单元（触发器），推导出激励函数
  - 假设采用D触发器，其特征方程Q* = D，则：

    $D1=Y1^*=\overline{Y1}\cdot Y0\cdot\overline{X}$

    $D0=Y0^*=\overline{Y1}\cdot Y0+X$

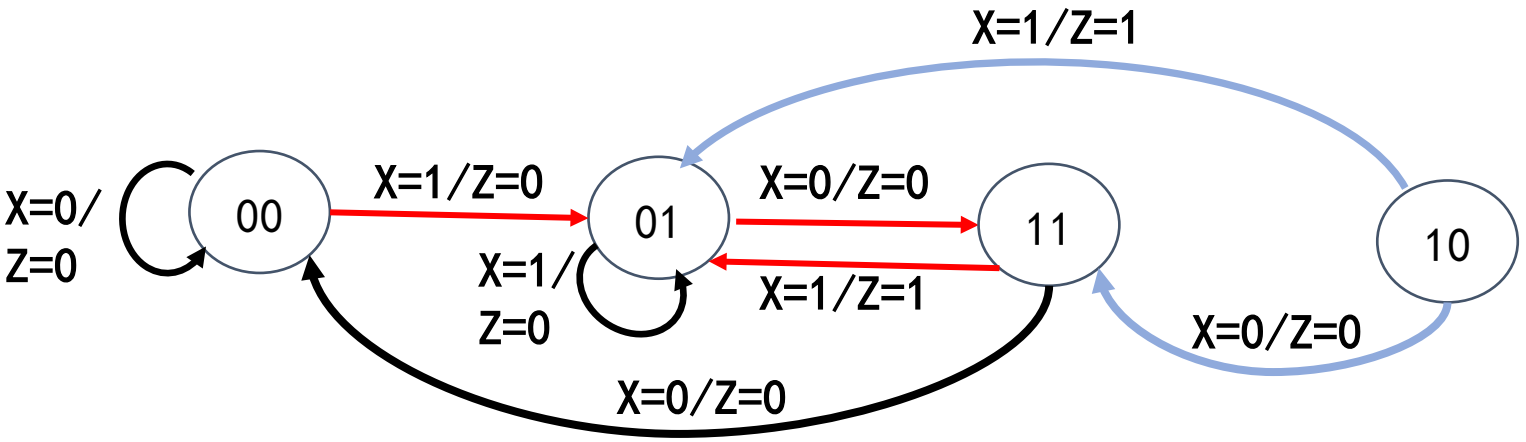
- 输出函数为：$Z=Y1\cdot X$
- 根据激励函数和输出函数，画出逻辑电路图



电路设计与分析

- **电路分析：包括未用状态分析和电路定时分析等**
  - 通常编码空间比状态机的状态集合大，因而存在未用状态

  如前述例子中，编码(2位)空间为4，而实际状态数为3



$D1=Y1^*=\overline{Y1}\cdot Y0\cdot\overline{X}$

$D0=Y0^*=\overline{Y1}\cdot Y0+X$

$Z=Y1\cdot X$

电路设计与分析

- **电路分析：包括未用状态分析和电路定时分析等**
  - 通常编码空间比状态机的状态集合大，因而存在未用状态

    如前述例子中，编码(2位)空间为4，而实际状态数为3
  - 若电路加电后进入未用状态，且在未用状态之间形成循环转换而无法进入工作状态，则称其为“挂起”现象
  - 若时序逻辑电路中的触发器具有预置功能，则可以通过预置处理，使电路进入正常的初始工作状态，从而避免“挂起”
  - 可利用未用状态的无关项进行化简。但需对未用状态进行分析，以判定电路进入未用状态时能否在有限个时钟周期后进入到工作状态。若能，且没有错误输出，则称电路为具有“自启动”能力；若不能，则需调整电路设计

电路设计与分析

未用状态分析举例

- 对于前述的例子，利用未用状态10作为无关项化简后，得到：

$Y1^*=\overline{Y1}\cdot Y0\cdot\overline{X}$

$Y0^*=\overline{Y1}\cdot Y0+X$

$Z=Y1\cdot X$

- 当处于未用状态10时，根据上述逻辑表达式，可知：

若输入X=0，则次态=00，输出Z=0

若输入X=1，则次态=01，输出Z=1

- 分析是否具有“自启动”能力

经过1个时钟周期就能进入正常工作状态，但是，当输入X=1时，输出Z=1，是错误输出，需要调整输出模块的设计

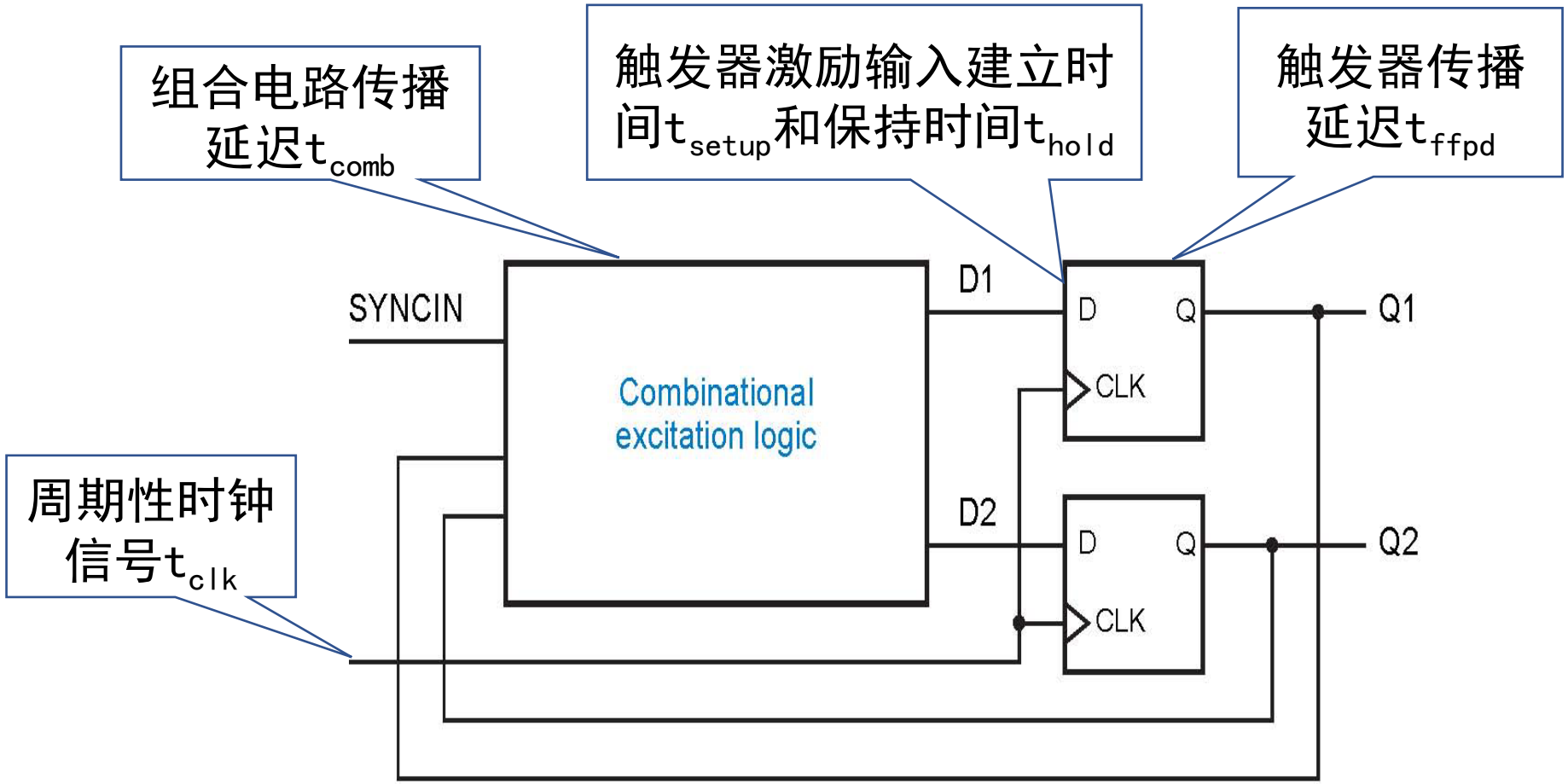
- 重新设计逻辑电路中的输出模块

$Z=Y1\cdot Y0\cdot X$　在未用状态10时，若输入X=1时，则输出Z=0。此时，不会发生误输出

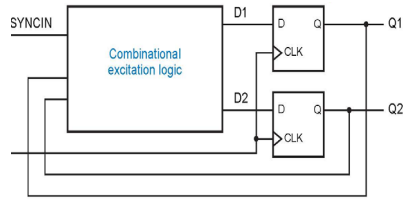
电路设计与分析

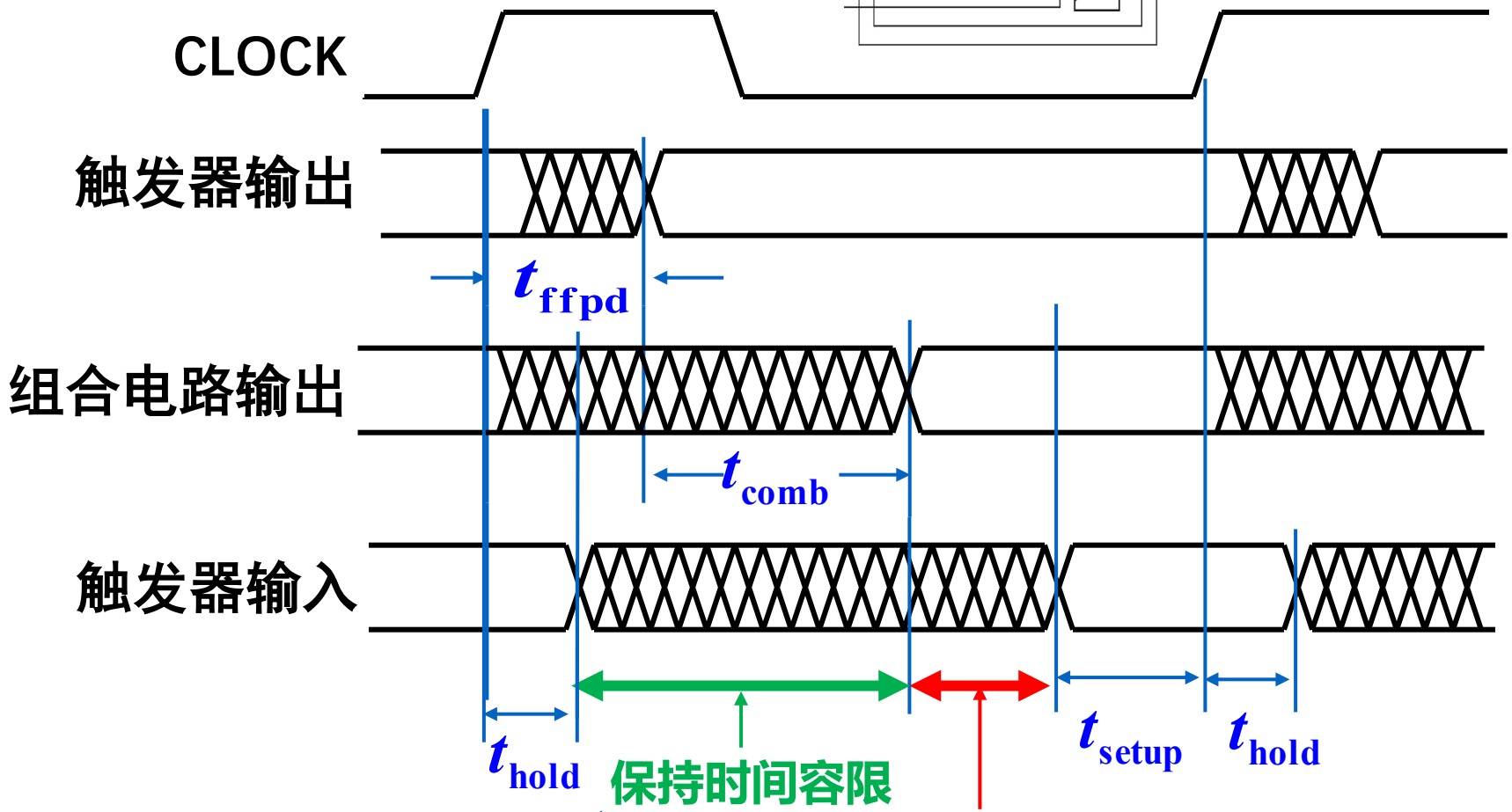
电路定时分析

- 时序逻辑电路的工作频率和组合逻辑电路传输延迟、触发器建立和保持时间、触发器传输延迟等时间密切相关。



电路设计与分析





$$t_{ffpd(min)}+t_{comb(min)}-t_{hold}$$

$$t_{clk}-t_{ffpd(max)}-t_{comb(max)}-t_{setup}$$

时间容限指为保证电路正常工作某信号定时所允许的最大时间范围

电路设计与分析

- 建立时间容限=$t_{clk}-t_{ffpd(max)}-t_{comb(max)}-t_{setup}$，>0
- 保持时间容限=$t_{ffpd(min)}+t_{comb(min)}-t_{hold}$，>0

因此，得到时序约束关系：

(1) $t_{clk} > t_{ffpd(max)} + t_{comb(max)} + t_{setup}$

(2) $t_{hold} < t_{ffpd(min)} + t_{comb(min)}$

（1）为使触发器正常工作，必须保证时钟周期t_{clk}不能小于触发器锁存延迟t_{ffpd} + 次态信号经过激励逻辑延迟t_{comb} + 触发器的建立时间t_{setup} 。

（2）为使触发器正常工作，必须保证外部激励信号在时钟有效边沿到来后的保持时间t_{hold}内能保持稳定不变。这就要求次态信号不能反馈太快，即触发器锁存延迟t_{ffpd} + 次态信号经过激励逻辑延迟t_{comb}不能小于触发器的保持时间t_{hold} 。

第四讲 典型时序逻辑部件设计

1. 计数器
2. 寄存器和寄存器堆
3. 移位寄存器

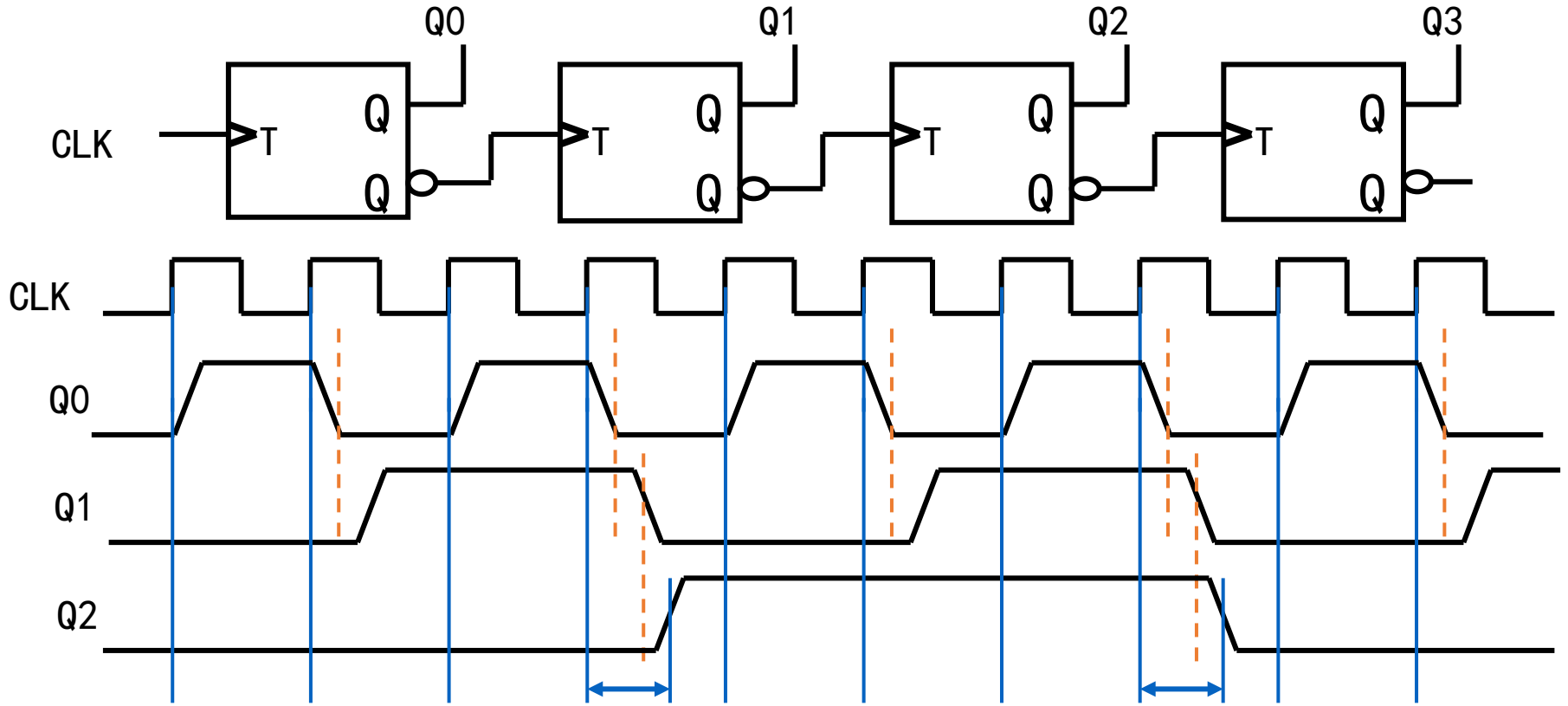
计数器

- 计数器是一种对外部激励信号进行总数统计的时序逻辑元件
- 一般从0开始计数，在达到最大计数值时输出一次计数完成信号，并重新开始计数
- 最大计数值为计数器的模
- 计数器的分类
  - 按时钟：同步、异步
  - 按计数方式：加法、减法、可逆
  - 按编码方式：二进制、十进制BCD码、循环码
  - 按进位方式：行波（串行）进位、并行进位

计数器

异步行波加法计数器

利用 T 触发器实现，激励输入像波浪一样由低位向高位传递，每个时钟周期传送一次。



Q_{i+1}总是在Q_i由1变为0时开始改变状态

第n位状态变换最长要经过 $n \times t_{TQ}$ 的延迟时间

计数器

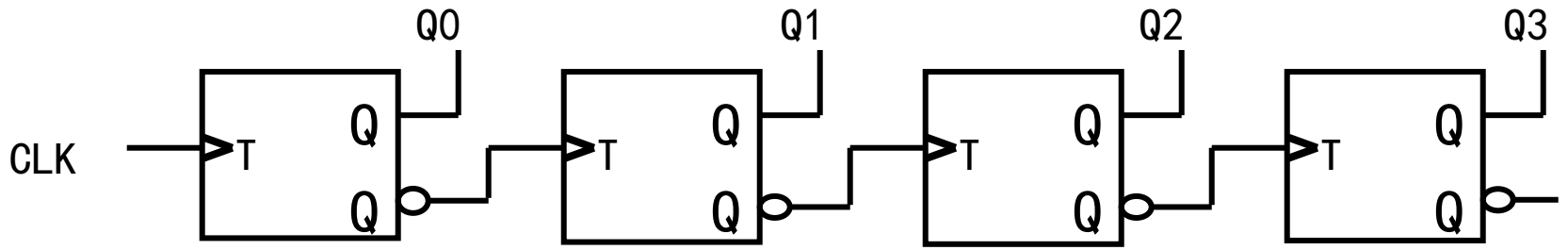
当编码为1111时，下个时钟到达后，经过$n \times t_{TQ}$ 延时，又回到编码0000

异步行波加法计数器

计数器的状态编码$Q_3Q_2Q_1Q_0$从0000开始，转换过程为

0000→0001→0010→0011→0100→0101→0110→0111→…→1111→0000

Q_{i+1}总是在Q_i由1变0时开始改变状态



第1个Clk上升沿到来后，最低位状态Q0从0变成1，此时其他三个状态位不变，因而得到状态编码0001；

第2个Clk到来后，Q0从1变成0，此时Q1从0变成1，而其他两个状态位不变，因而得到状态编码0010；

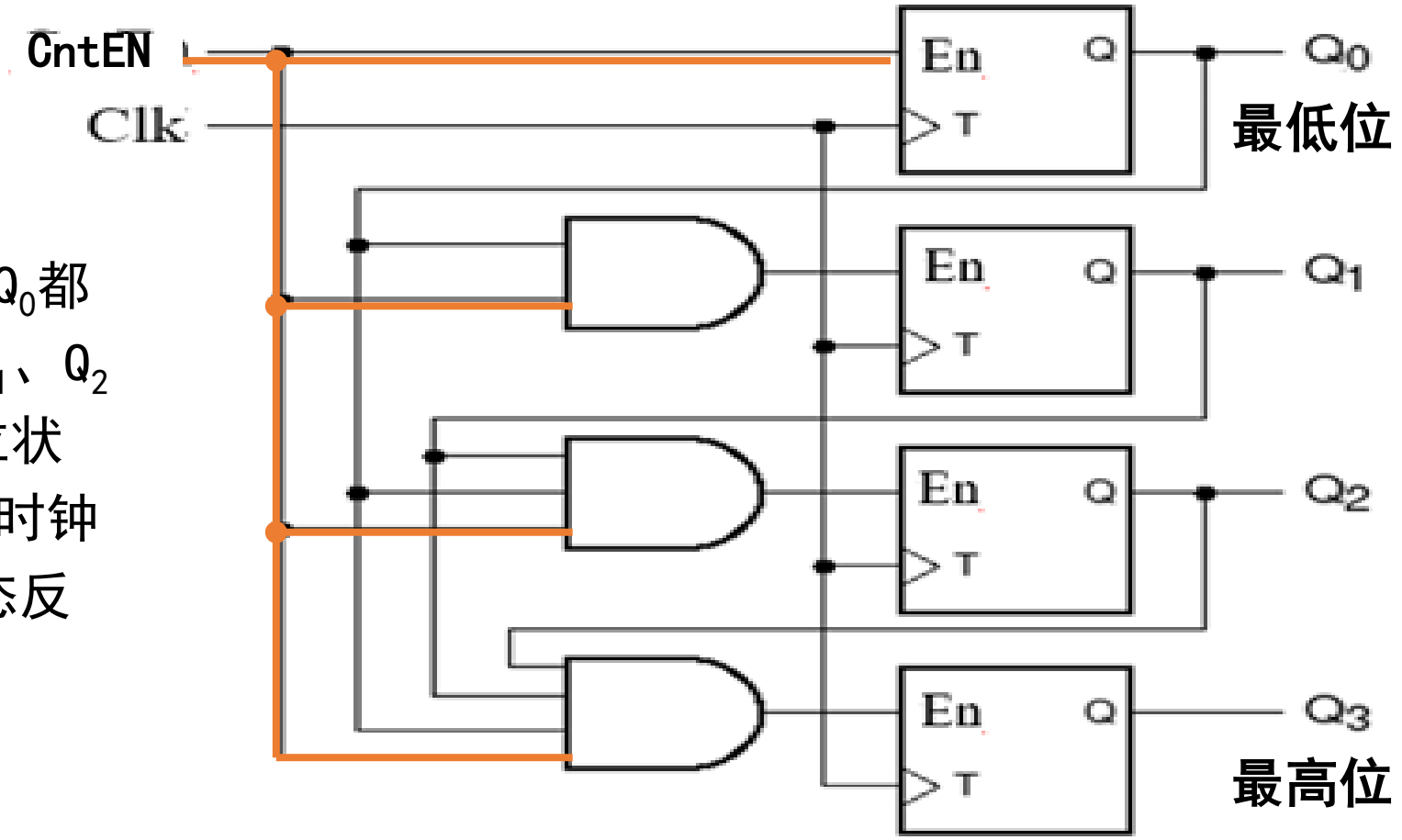
第3个Clk有效信号到来后，Q0从0变成1，此时其他三个状态位不变，因而得到状态编码0011；

第4个Clk有效信号到来后，Q0从1变成0，此时Q1从1变成0，Q2从0变成1，Q3状态位不变，因而得到状态编码0100；……。

计数器

同步并行加法计数器

同步计数器中所有触发器共用同一个时钟信号，在时钟信号边沿到达后，所有触发器的输出同时发生变化。



CntEN有效时，每个时钟Q_0都会发生状态改变；对于Q_1、Q_2和Q_3，只有在其所有低位状态都是1的情况下，下个时钟边沿到来后才会发生状态反转。

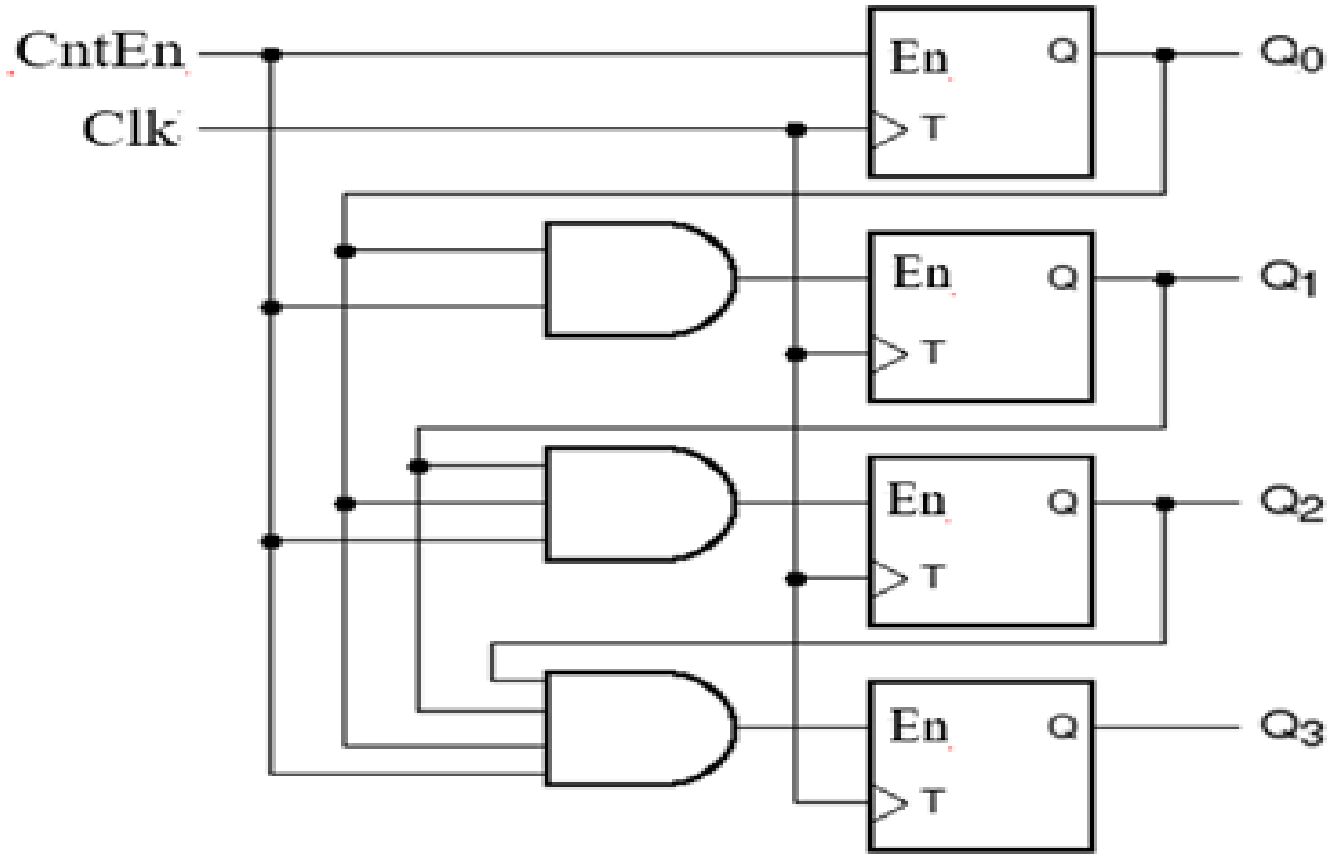
计数器

当编码为1111时，只要经过一个与门+t_{TQ} 延时，就可回到编码0000，比行波（串行）加法计数器快得多！

同步并行加法计数器

- 计数器的状态编码$Q_3Q_2Q_1Q_0$从0000开始，转换过程为
  0000→0001→0010→0011→0100→0101→0110→0111→1000→…→1111

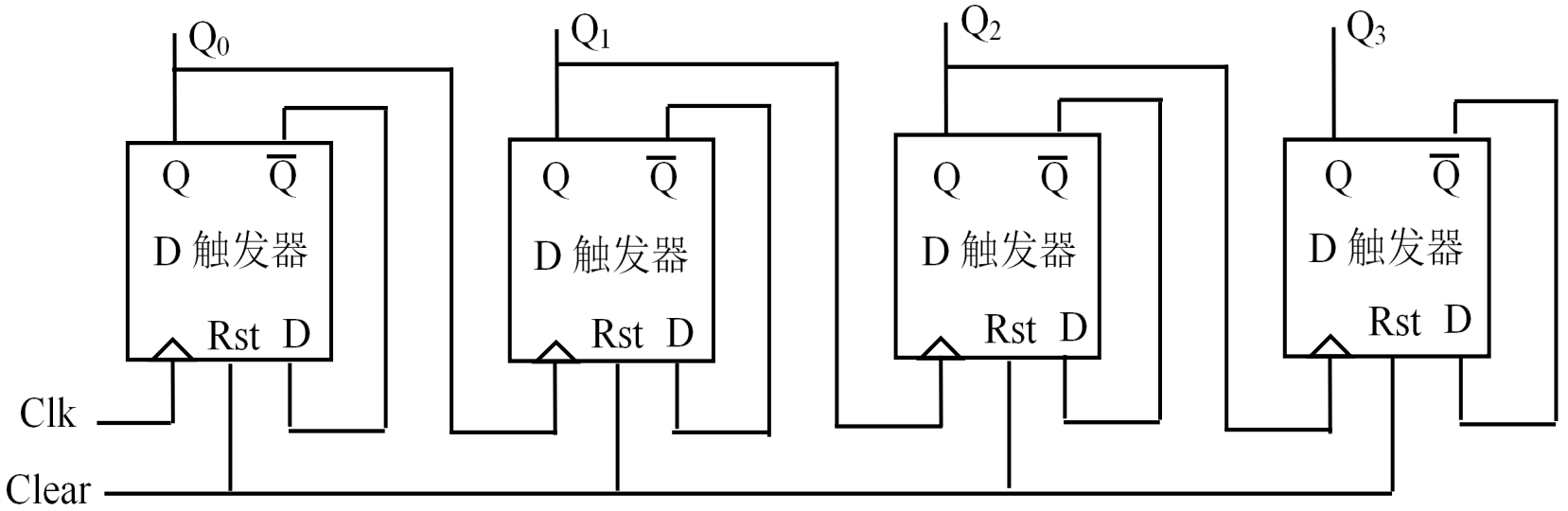
CntEN有效时，每个时钟Q_0都会发生状态改变；对于Q_1、Q_2和Q_3，只有在其所有低位状态都是1的情况下，下个时钟边沿到来后才会发生状态反转。



计数器

二进制异步行波减法计数器

由4个上升沿触发的D触发器组成，Clear为复位（清0）信号



通过Clear信号使所有D触发器清0，得到初始状态编码0000

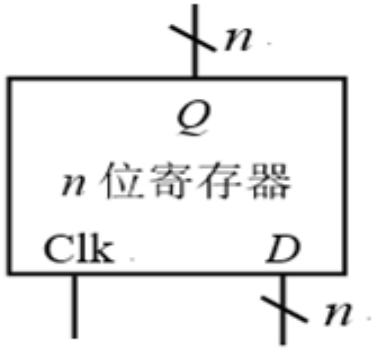
以后每来一个时钟，发生一次状态转换，其过程为

0000→1111→1110→1101→1100→1011→1010→1001→1000→…→0001→0000。

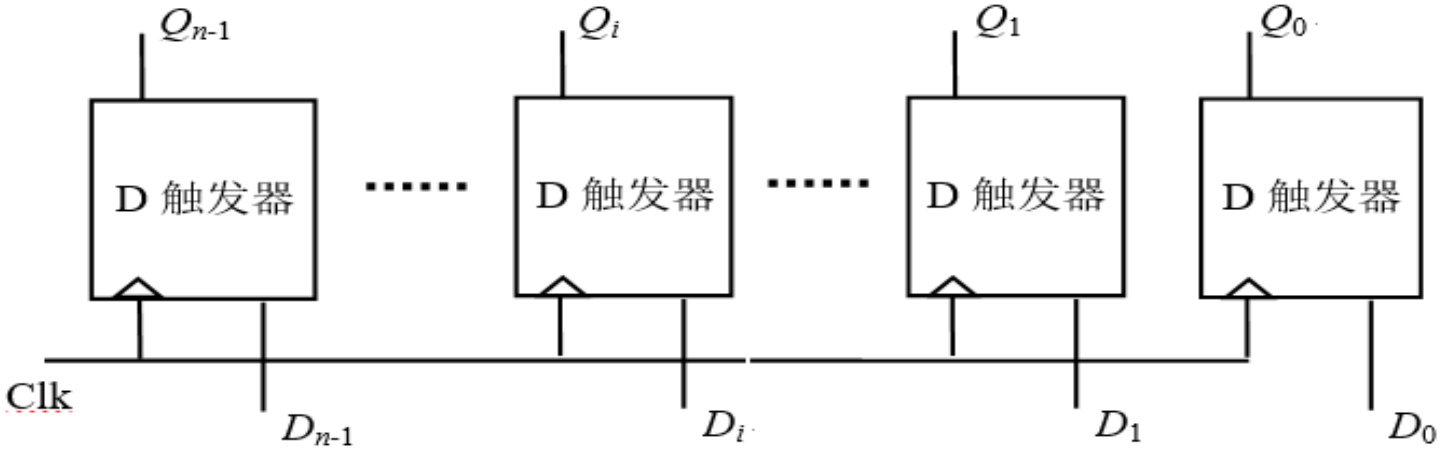
第四讲 典型时序逻辑部件设计

1. 计数器
2. 寄存器和寄存器堆
3. 移位寄存器

寄存器和寄存器堆

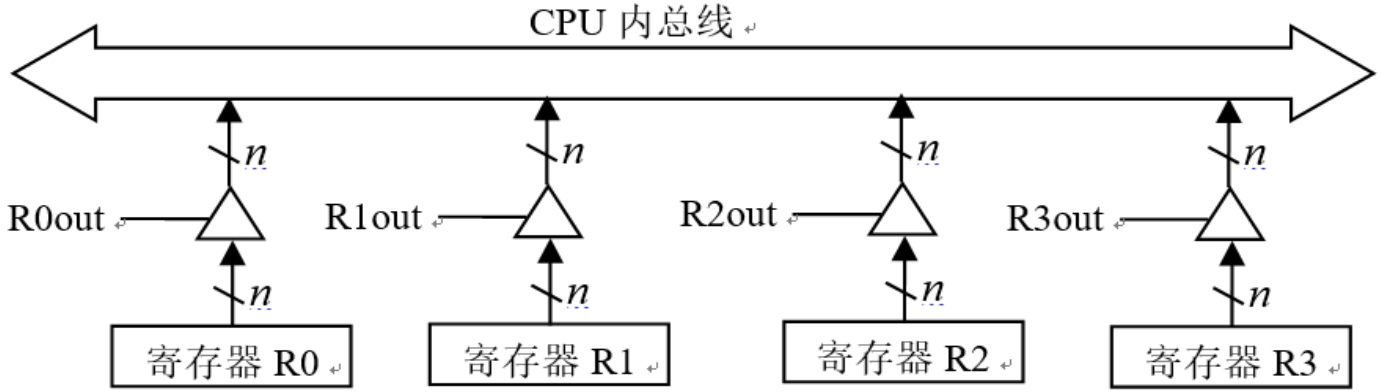


- 寄存器是用来暂存信息的逻辑部件
- 寄存器可直接由若干个触发器组成



- 寄存器通过三态门和总线互连

任何时刻至多只能一个Rout有效

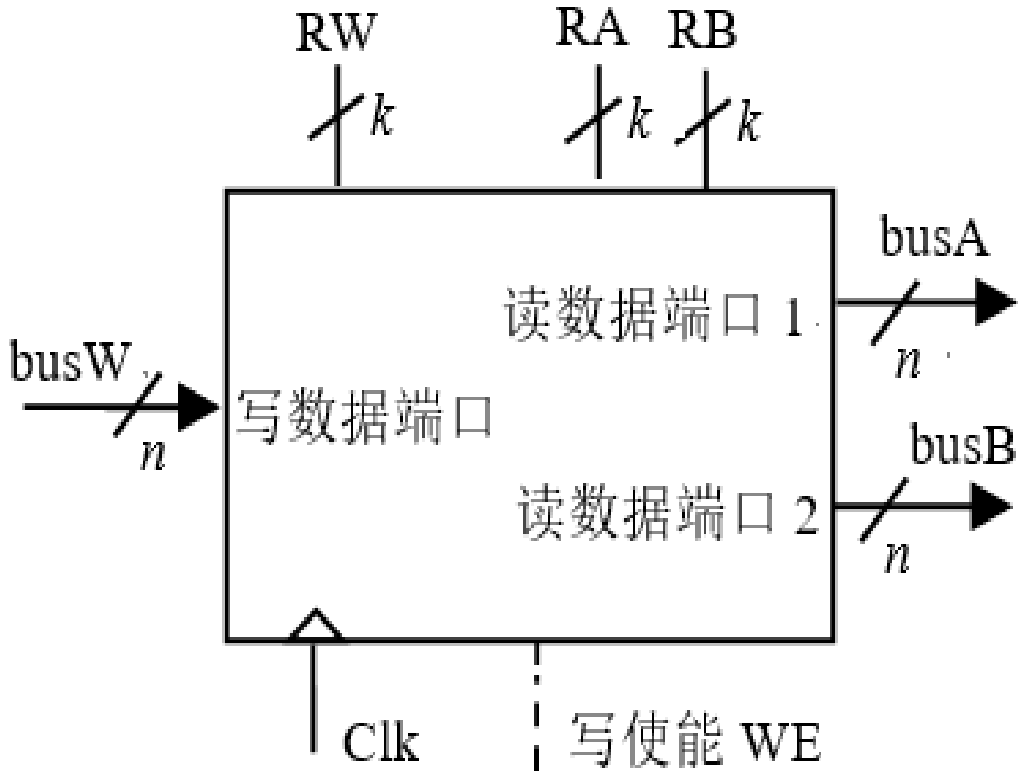


寄存器和寄存器堆

- 寄存器堆(Register File)：CPU内部用于暂存指令执行过程中的中间数据，也称通用寄存器组(General Purpose Register set，GPRs)
- 由许多寄存器组成，每个寄存器有一个编号，CPU可对指定编号的寄存器进行读写

寄存器堆中共有2^k个寄存器，每个寄存器位数为n，RA和RB分别是读口1和读口2的寄存器编号，RW是写口的寄存器编号

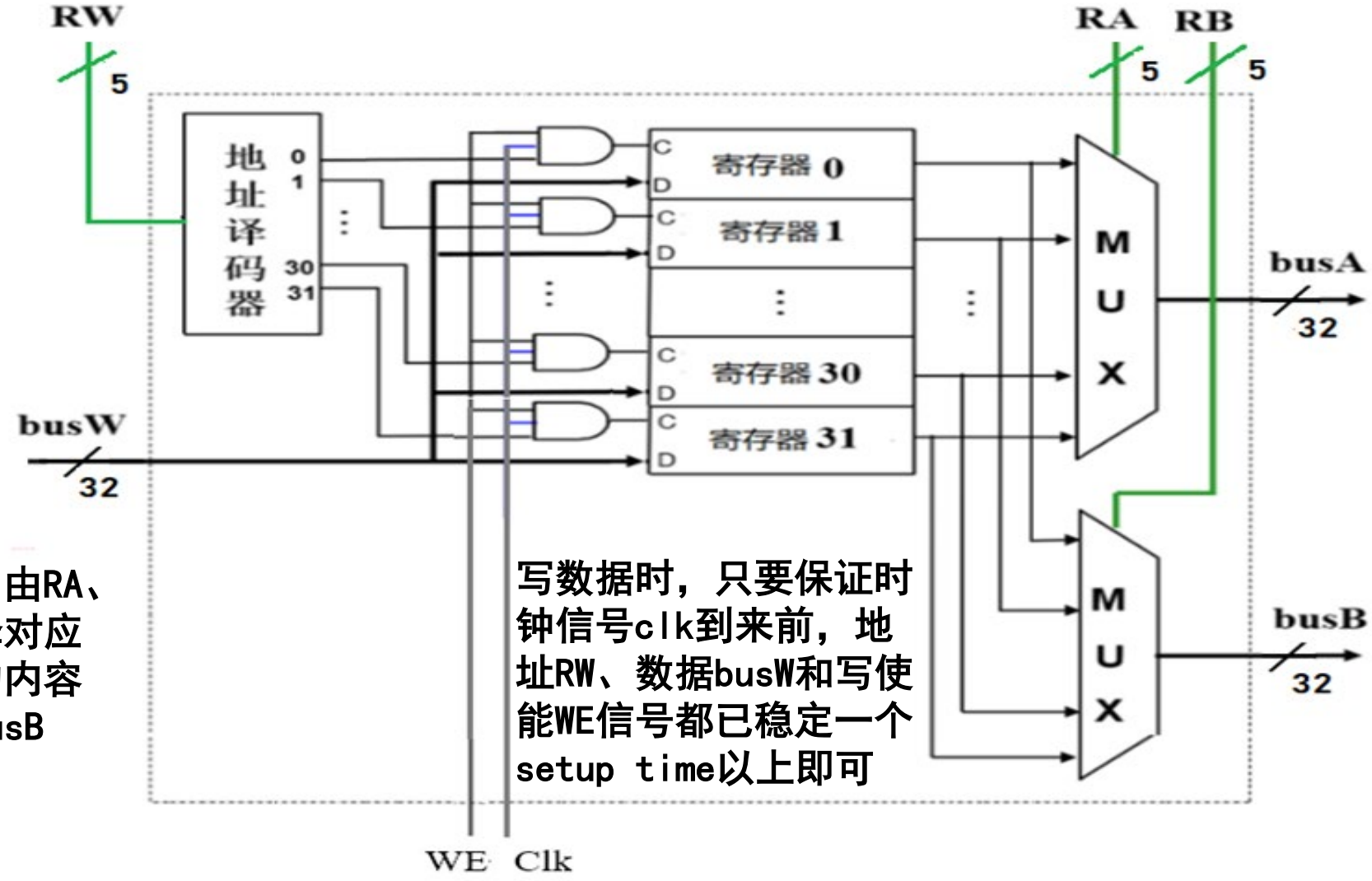
读操作属于组合逻辑操作；写操作属于时序逻辑操作，需要时钟信号Clk和写使能信号WE的控制



寄存器和寄存器堆

写数据时，地址RW必须先有效，然后WE和Clk有效，在WE和Clk结束有效之前，数据busW必须提前有效。

寄存器堆内部结构



读数据时，由RA、RB分别选择对应寄存器中的内容送busA、busB

写数据时，只要保证时钟信号clk到来前，地址RW、数据busW和写使能WE信号都已稳定一个setup time以上即可

第四讲 典型时序逻辑部件设计

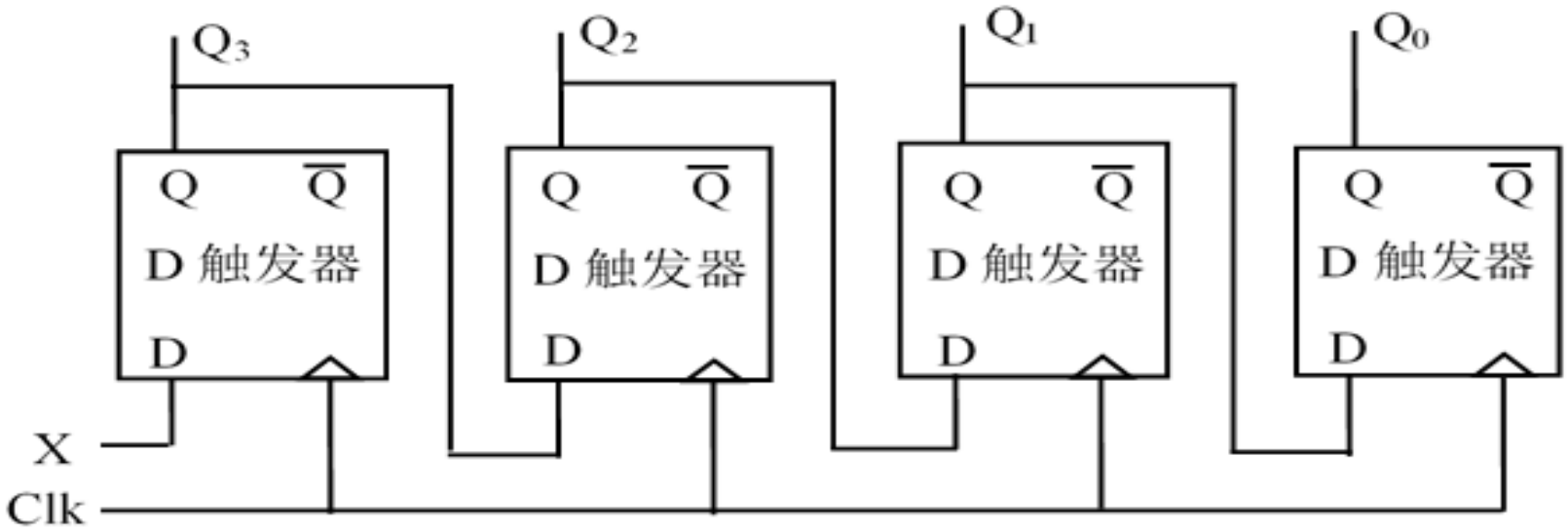
1. 计数器
2. 寄存器和寄存器堆
3. 移位寄存器

移位寄存器

移位寄存器

能够实现暂存信息的左移或右移功能，通常由时钟信号控制

例如：4个D触发器可构成一个右移寄存器



假设初始状态编码$Q_3Q_2Q_1Q_0$=0000，X输入为序列10011011

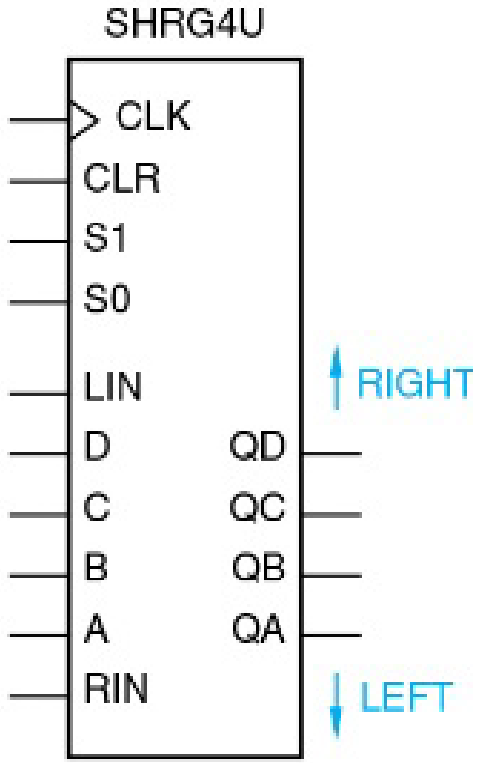
则$Q_3Q_2Q_1Q_0$Q3的输出编码依次为：

0000、1000、0100、0010、1001、1100、0110、1011、1101

移位寄存器

4位通用移位寄存器，如74X194

具有数据左移、数据右移、数据保持和数据载入功能



	Inputs			Next state			
Function	CLR	S1	S0	QA*	QB*	QC*	QD*
Clear	1	x	x	0	0	0	0
Hold	0	0	0	QA	QB	QC	QD
Shift right	0	0	1	RIN	QA	QB	QC
Shift left	0	1	0	QB	QC	QD	LIN
Load	0	1	1	A	B	C	D

左移从QD移到QA向下移↓　　右移从QA移到QD向上移↑

移位寄存器

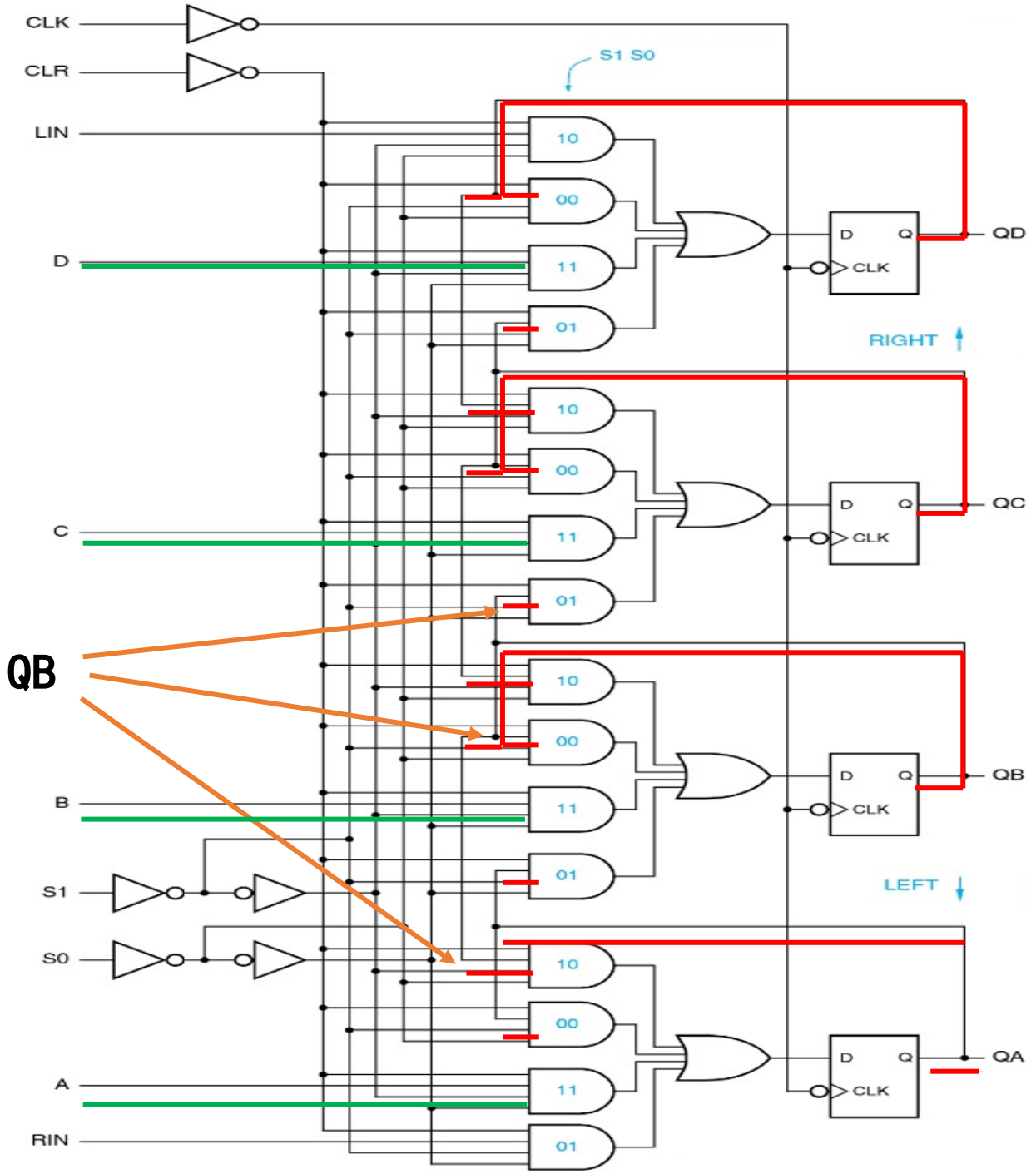
4位通用移位寄存器结构图

S1S0=00：保持

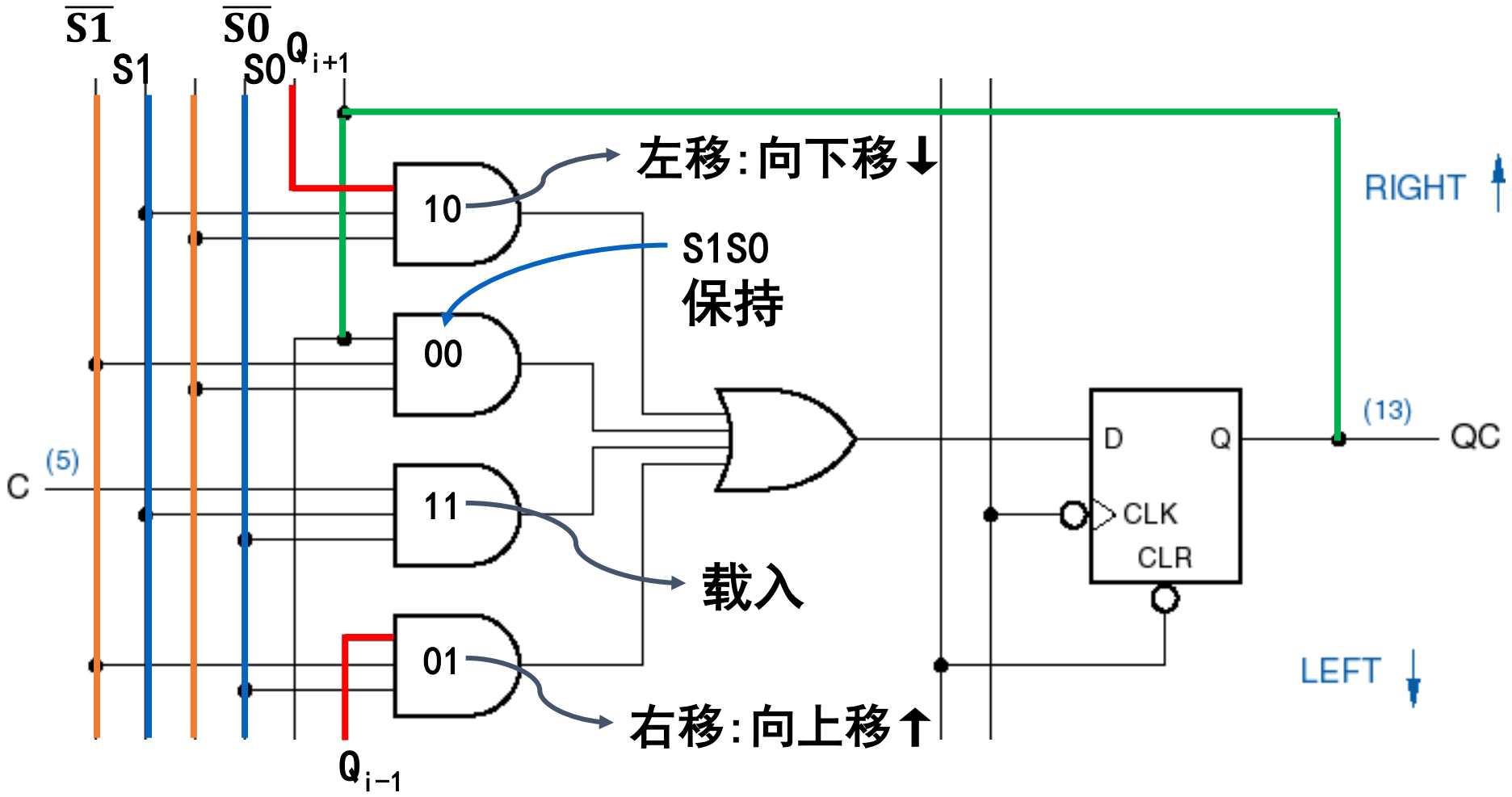
S1S0=01：上移

S1S0=10：下移

S1S0=11：加载



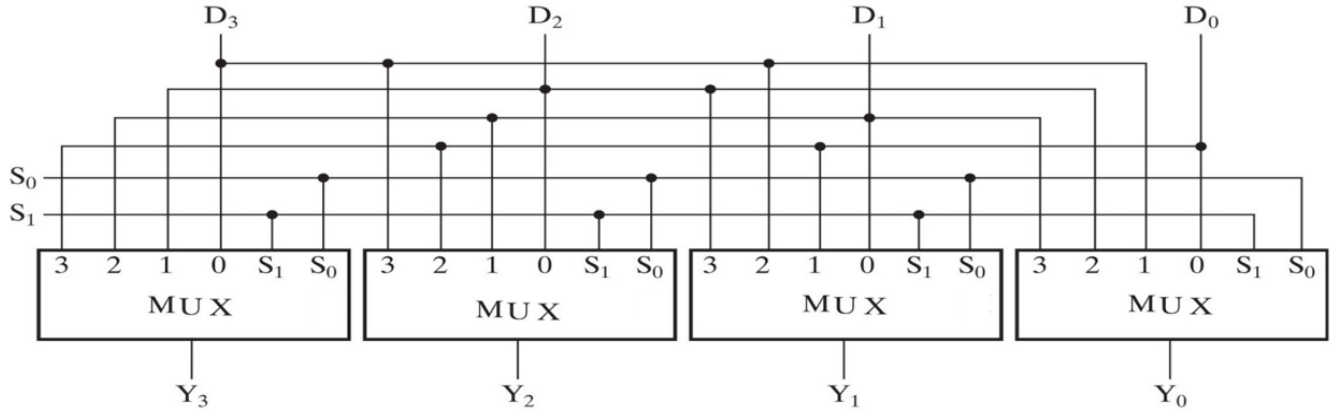
移位寄存器



$$D_i = \overline{S1} \cdot \overline{S0} \cdot Q_i + \overline{S1} \cdot S0 \cdot Q_{i-1} + S1 \cdot \overline{S0} \cdot Q_{i+1} + S1 \cdot S0 \cdot IN_i$$

移位寄存器

- 74x194一个时钟周期只能移动一位，需要移动多位时需要消耗多个时钟周期，效率太低。
- 桶形移位器（Barrel Shifter）：一次可以循环移动多位，可采用多路选择器实现（组合逻辑电路）。



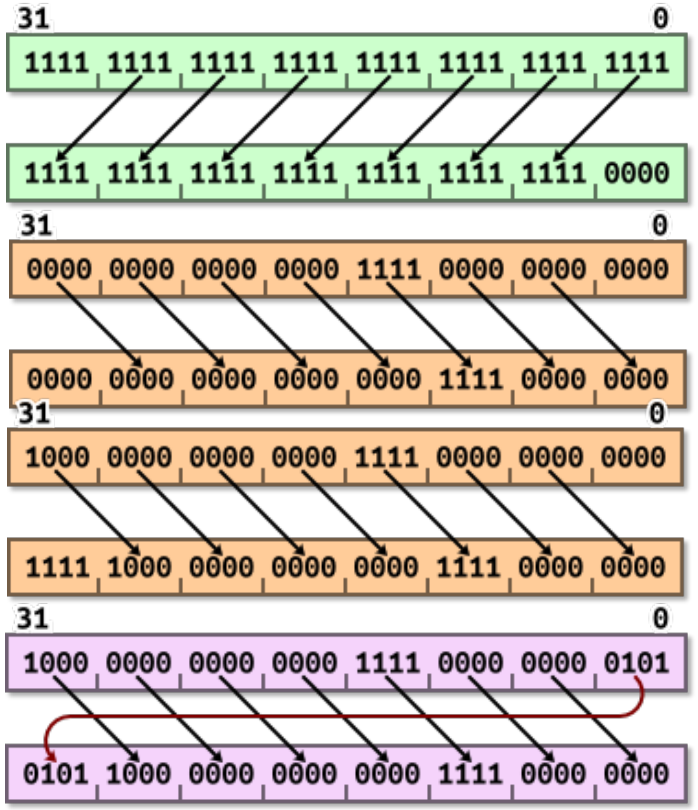
4位桶形移位器的实现和功能

Function Table for 4-Bit Barrel Shifter

Select		Output				
S_1	S_0	Y_3	Y_2	Y_1	Y_0	Operation
0	0	D_3	D_2	D_1	D_0	No rotation
0	1	D_2	D_1	D_0	D_3	Rotate one position
1	0	D_1	D_0	D_3	D_2	Rotate two positions
1	1	D_0	D_3	D_2	D_1	Rotate three positions

移位寄存器

- CPU设计中常常需要实现对数据的多位移动，如：浮点数加减指令需要通过移动尾数部分，来对齐指数；多数指令集都提供直接对数据进行多位左移、右移等算数移位或逻辑移位指令。
- CPU设计中使用综合多种移位功能模式的桶形移位寄存器来支撑上述指令的执行。如：ARM指令集中：
  - LSL - Logical Shift Left
    - 移入0
  - ASR - Arithmetic Shift Right
    - 右移时移入符号位，正数0，负数 1
  - ROR - Rotate Right
    - 循环右移



RISC-V指令集也有类似的指令，扩展集中也提供了循环移位指令。