复习课

第1章 计算机系统概述与二进制编码

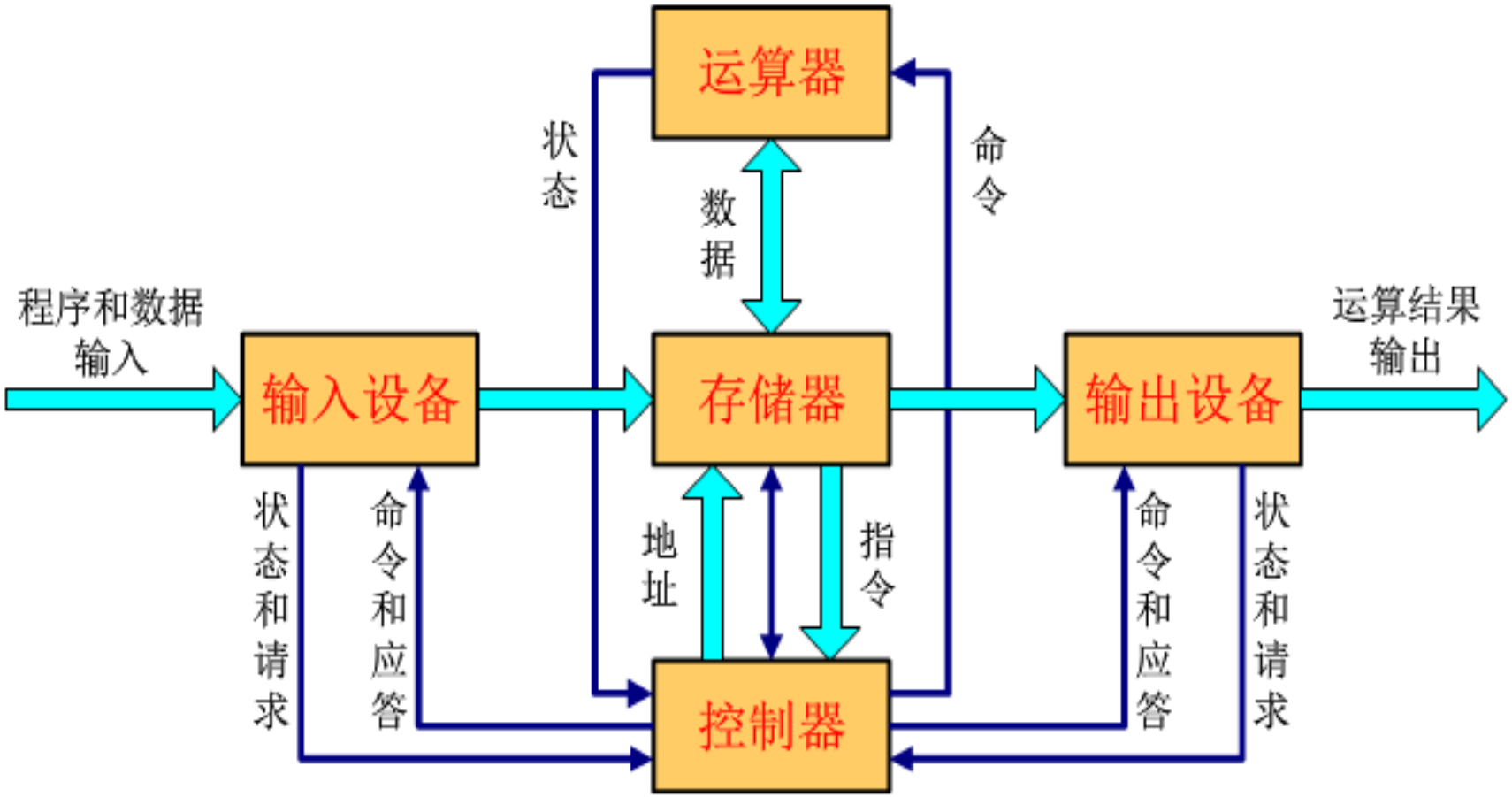
- 第一讲 计算机系统概述
- 第二讲 二进制编码

重要概念

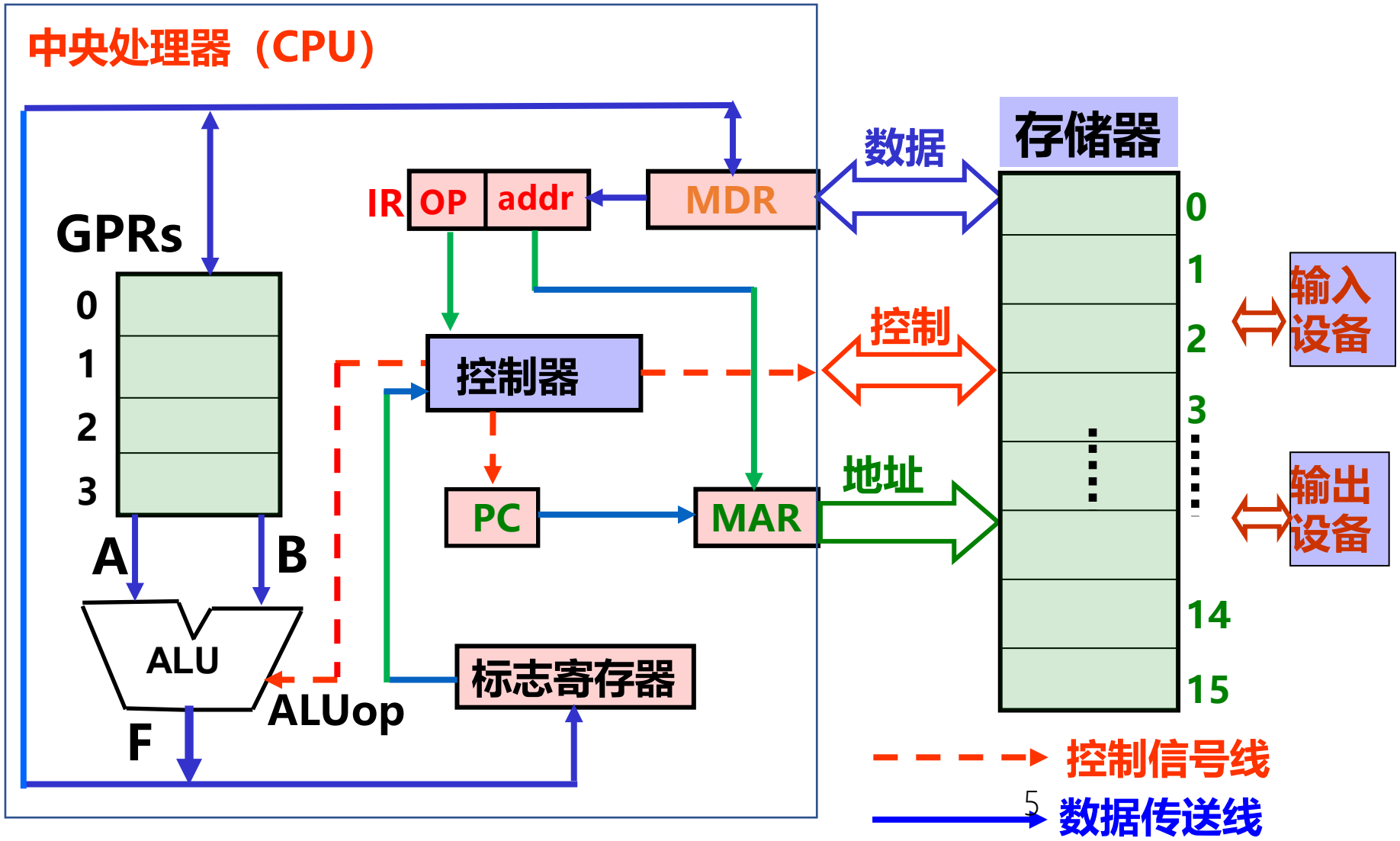
通用电子计算机（general-purpose electronic computer）
冯·诺依曼计算机结构的主要特点、存储程序思想，计算机硬件的基本组成和功能（功能部件、寄存器、控制信号）

计算机系统的层次结构
指令集（instruction set）
指令集体系结构（Instruction Set Architecture，ISA）
编程语言（programming language）、系统软件（system software）、应用软件（application software）

重要概念


运算器
状态
数据
命令
程序和数据
输入
输入设备
存储器
输出设备
运算结果
输出
状态和请求
命令和应答
地址
指令
控制器
命令和应答
状态和请求

重要概念


中央处理器（CPU）
GPRs
0
1
2
3
A
B
ALU
F
ALUop
IR
OP
addr
MDR
控制器
PC
MAR
标志寄存器
数据
控制
地址
存储器
0
1
2
3
14
15
输入设备
输出设备
控制信号线
数据传送线

重要概念

二进制（binary system）、真值、机器数

原码（sign magnitude）
反码（one’s complement）
补码（two’s complement）（定义、特性）
移码（excess notation，biased exponent）

规制、规格、编码
数值数据：无符号整数、带符号整数、浮点数（规格化数、非规格化数、上下溢出、表数特点）、十进制数
非数值数据：逻辑数（包括位串）、西文字符和汉字

机器字长（machine word length）
编址单位（addressing unit）
大端方式（big endian）、小端方式（little endian）

重要内容一：程序执行过程

程序执行过程

假设模型机M中8位指令，格式有两种：R型、M型

格式	4位	2位	2位	功能说明
R型	Op	rt	rs	R[rt] ← R[rt] op R[rs] 或 R[rt] ← R[rs]
M型	Op	Addr		R[0] ← M[addr] 或 M[addr] ← R[0]

rs和rt为通用寄存器编号；addr为主存单元地址
R型：op=0000，寄存器间传送（mov）； op=0001，加（add）
M型：op=1110，取数（load）；op=1111，存数（store）

问题：指令 1110 0111的功能是什么?
答：因为op=1110，故是M型load指令，功能为：
R[0] ←M[0111]，即：将主存地址0111（7号单元）中的
8位数据装入到0号寄存器中。

重要内容二：二进制编码

- 正数：符号位（sign bit）为0，数值部分不变
- 负数：符号位为1，数值部分“各位取反，末位加1”

补码的定义　假定补码有n位，则：

定点整数：$[X]_{补} = 2^n + X \quad (-2^{n-1} \le X < 2^{n-1}, \mod 2^n)$

定点小数：$[X]_{补} = 2 + X \quad (-1 \le X < 1, \mod 2)$

补码（modular运算）：+ 和– 的统一

令：$[A]_{补} = a_{n-1}a_{n-2}\cdots\cdots a_1a_0$

则：$A = -a_{n-1}\cdot 2^{n-1} + a_{n-2}\cdot 2^{n-2} + \cdots\cdots a_1\cdot 2^1 + a_0\cdot 2^0$

重要内容二：二进制编码

进制转换

编码的表示与转换，包括定点数的编码表示、整数的表示、浮点数的表示、十进制数的二进制编码表示等

计算机中数据的宽度以及排列方式

第2章 数字逻辑基础

第一讲 布尔代数

第二讲 逻辑门和逻辑关系描述

第三讲 逻辑函数化简

重要概念

布尔代数（Boolean algebra）
逻辑门（logic gate）、CMOS晶体管、噪声容限（输入输出电平）
数字抽象（digital abstraction）
传输延迟（propagation delay）
逻辑表达式（logical expression）
真值表（truth table）
乘积项（product term）、求和项（sum term）、积之和表达式（sum-of-products expression）、和之积表达式（product-ofsums expression）
最小项（minterm）、最大项（maxterm）、最小项列表（minterm list）、最大项列表（maxterm list）、卡诺图（Karnaugh map）

重要内容一：布尔代数化简

- 二变量和三变量定理
  - 交换律 (T6) X+Y=Y+X　　(T6D) X•Y=Y•X
  - 结合律 (T7) (X+Y)+Z=X+(Y+Z)　　(T7D) (X•Y) •Z=X•(Y•Z)
  - 分配律 (T8) X•Y +X•Z=X •(Y+Z)　　(T8D) (X+Y) •(X+Z)=X+Y•Z
  - 吸收律 (T9) X+ X•Y =X　　(T9D) X•(X+Y)=X
  - 组合律 (T10) X•Y + X•$\bar{Y}$ =X　　(T10D) (X+Y) •(X+$\bar{Y}$) =X
  - 一致律 (T11) X•Y+ $\bar{X}$ • Z+Y • Z =X • Y+ $\bar{X}$ • Z (如何判断能否消掉一项)

    (T11D) (X+Y) •($\bar{X}$ +Z) •(Y+Z)=(X+Y) •($\bar{X}$ +Z)

与算术运算规则不同!

重要内容一：布尔代数化简

n变量定理

- 广义同一律 (T12)　　X+X+…+X=X
  (T12D)　X •X • …•X = X

- 德•摩根定理De Morgan’ s Theorem
  (T13)　$\overline{X_1 \cdot X_2 \cdot \cdots \cdot X_n} = \overline{X_1} + \overline{X_2} + \cdots + \overline{X_n}$
  (T13D)　$\overline{X_1 + X_2 + \cdots + X_n} = \overline{X_1} \cdot \overline{X_2} \cdot \cdots \cdot \overline{X_n}$

- 广义德•摩根定理
  (T14)　$\overline{F(X_1, X_2, \cdots, X_n, +, \cdot)} = F(\overline{X_1}, \overline{X_2}, \cdots, \overline{X_n}, \cdot, +)$

- 香农定理
  (T15)　F(X1,X2,…,Xn)=X1 • F(1,X2, …,Xn)+ $\overline{X_1}$ • F(0,X2, …,Xn)
  (T15D) F(X1,X2,…,Xn)=[X1 + F(0,X2, …,Xn)] •[$\overline{X_1}$ + F(1,X2, …,Xn)]

用于多变量函数的实现

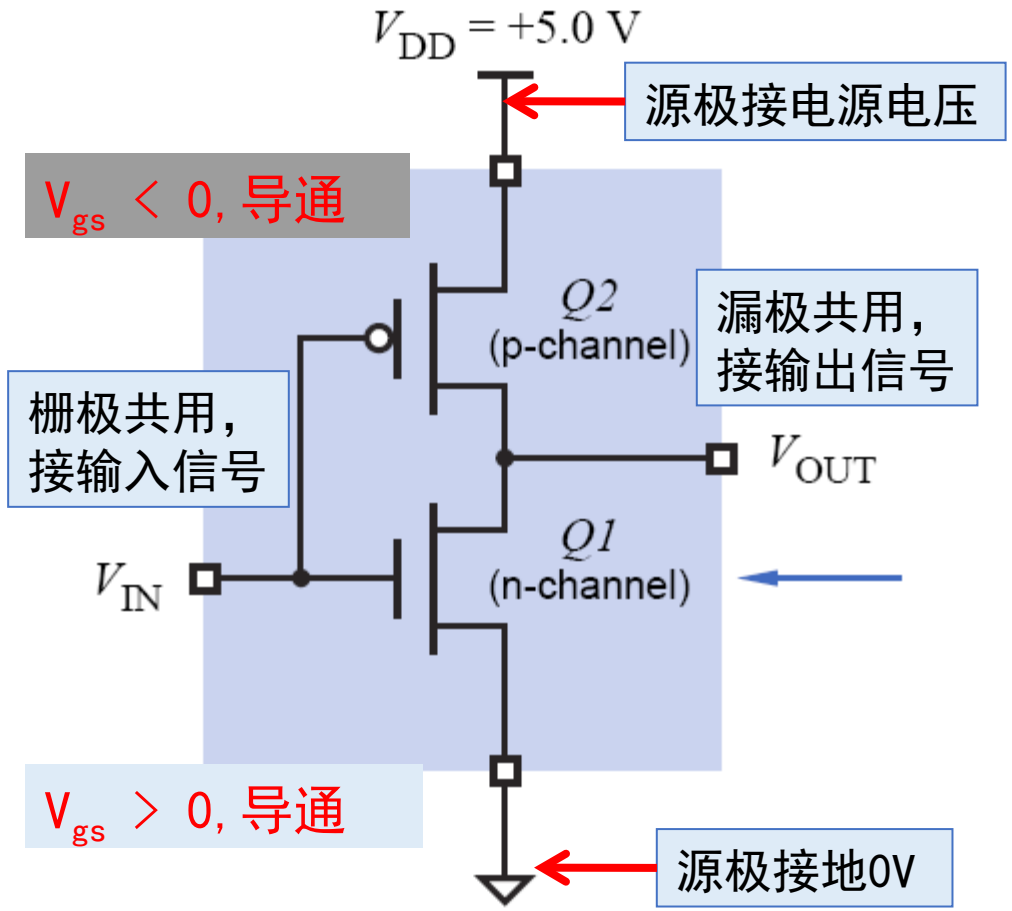
重要内容一：布尔代数化简

- 对于任何一个逻辑表达式Y，若将其中的“·”与“+”互换，“0”和“1”互换，则得到Y的对偶式Y^D，称Y与Y^D互为对偶式。
- 对偶定律：若两个逻辑表达式相等，则它们的对偶式也相等。
  - 在**保持运算优先次序不变**的前提下

$$X+X\cdot Y=X \quad \xrightarrow{\text{对偶式}} \quad X\cdot(X+Y)=X$$

重要内容二：CMOS晶体管

非门使用一对CMOS晶体管实现



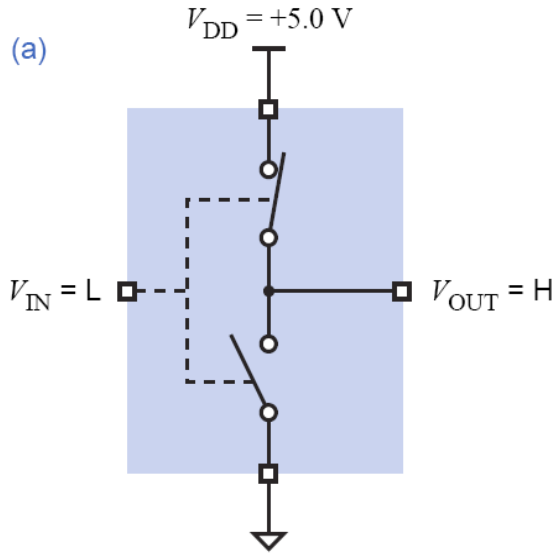
原理图

开关模型

V_{IN}=L

V_{gs1} = 0

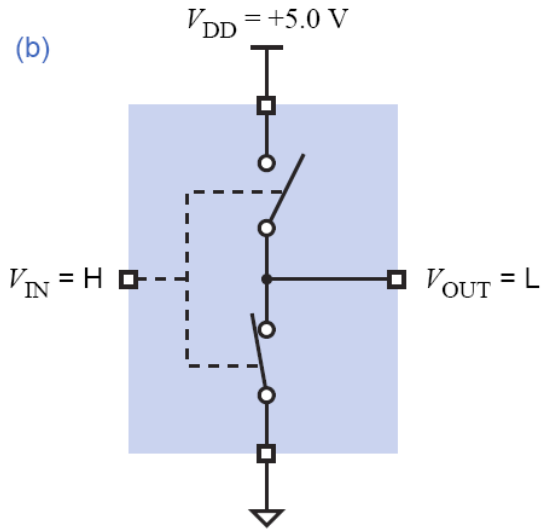
V_{gs2} =−5V



V_{IN}=H

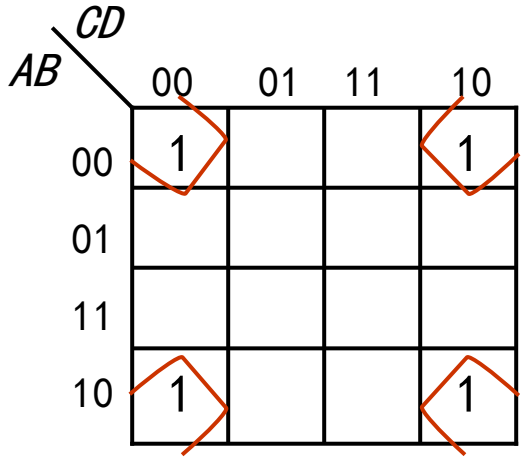
V_{gs1} =5V

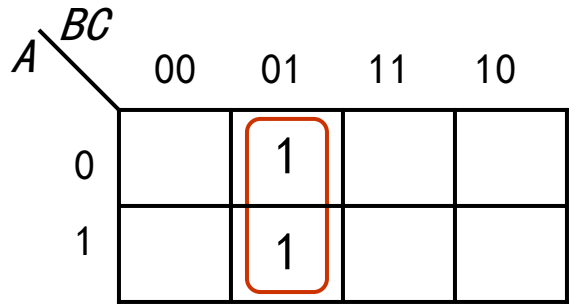
V_{gs2} = 0



重要内容三：卡诺图化简

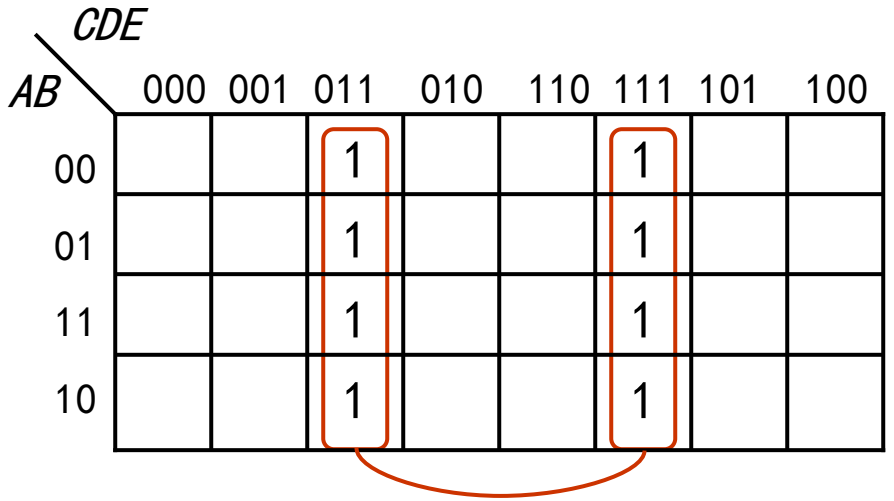
- 卡诺图每个单元对应一个最小项，其中标注该最小项在真值表中的输出值，如果输出为1，则称为“1单元”
- 若两个“1单元”相邻，则表示两个最小项仅1个变量不同，该变量在一个方格中为原变量，在另一个方格中则为反变量。根据T10，这两个最小项可合并为一个乘积项，并消去那个不相同的变量
- 相邻单元数越多可消去的变量数越多
- 相邻2^i个“1单元” 的最小项可以合并成一个乘积项，并消去i个不同的变量
- 使用一个方框来标注可以合并的“1单元”，这个方框称为卡诺圈





$\overline{B} \cdot C$

$\overline{B} \cdot \overline{D}$



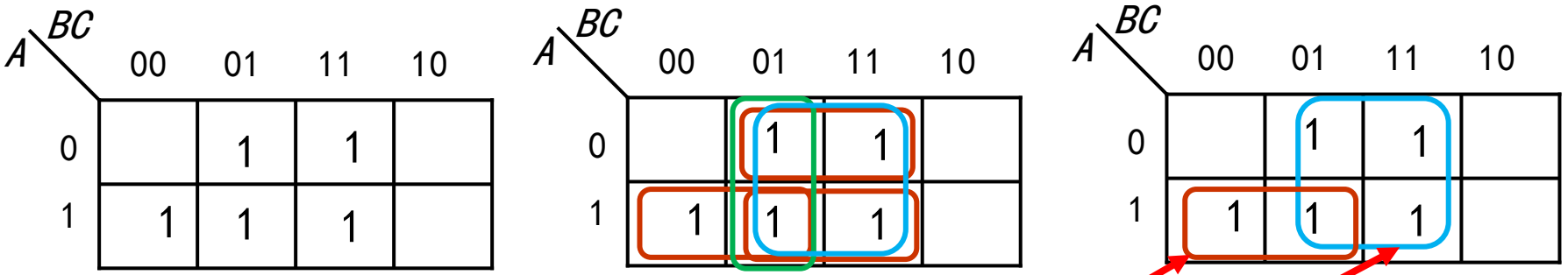
$D \cdot E$

重要内容三：卡诺图化简

- 蕴涵项是一个乘积项，覆盖了逻辑函数的1个或多个最小项。
- 质蕴涵项(prime implicant)：没有被该逻辑函数的其它单个蕴涵项所覆盖的蕴涵项。
- 实质蕴涵项(essential prime implicant)：覆盖的最小项中至少有一个最小项没有被其它所有质蕴涵项所覆盖的质蕴涵项。
  - 质蕴涵项覆盖的最小项越多越可能是实质蕴涵项。
- 如果逻辑函数的所有最小项都被一组的质蕴涵项所覆盖，则该组蕴涵项被称为函数的一个覆盖(Cover)。
  - 一定包含了所有的实质蕴涵项。
- 最小覆盖是包含质蕴涵项数最少的，并且质蕴涵项中的变量总数也是最少的。
  - 逻辑函数化简问题就转化为寻找该函数的最小覆盖问题。

重要内容三：卡诺图化简

- 确定逻辑函数F(A, B, C)=Σm(1, 3, 4, 5, 7)的最小覆盖的方法



蕴涵项：

最小项： $\{\overline{A}\cdot\overline{B}\cdot C,\ \overline{A}\cdot B\cdot C,\ A\cdot\overline{B}\cdot\overline{C},\ A\cdot\overline{B}\cdot C,\ A\cdot B\cdot C\}$

含有两个最小项的蕴涵项： $\{\overline{A}\cdot C,\ A\cdot\overline{B},\ A\cdot C,\ \overline{B}\cdot C,\ B\cdot C\}$

含有四个最小项的蕴涵项： $\{C\}$

质蕴涵项，没有**被覆盖**的蕴涵项： $\{A\cdot\overline{B},\quad C\}$

实质蕴涵项 = $\{A\cdot\overline{B},\quad C\}$

最小覆盖 $=A\cdot\overline{B}+C$

第3章 组合逻辑电路

第一讲 组合逻辑电路与典型部件

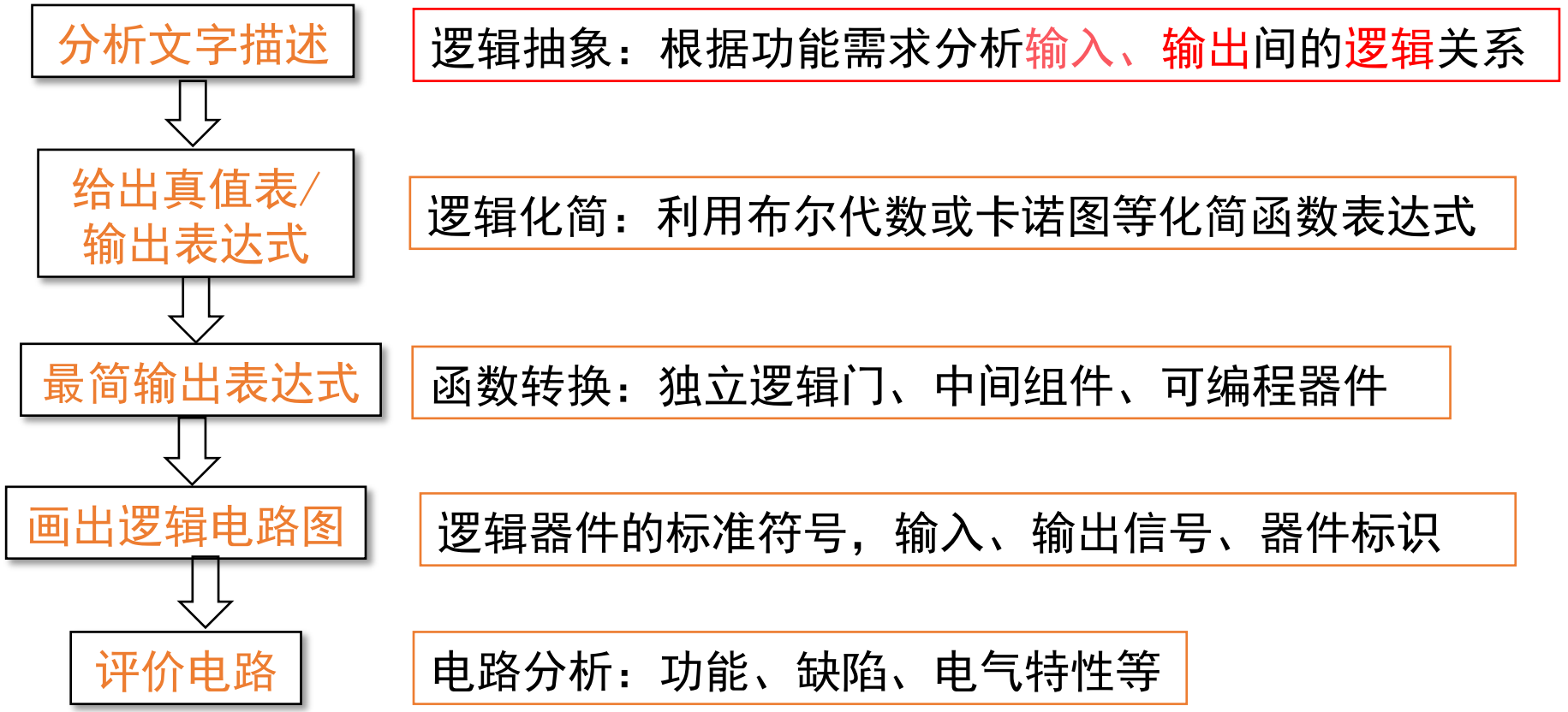
第二讲 组合逻辑电路时序分析

重要概念

组合逻辑电路（combinational logic circuit）
扇入系数（fan-in coeff icient）
扇出系数（fan-out coeff icient）
门延迟（gate delay）
无关项（“ don’ t care”term）
三态门（tri-state gate）
编码器（encoder）
译码器（decoder）
多路选择器（multiplexer）
多路分配器（de-multiplexer）
加法器（adder）
算术逻辑部件（arithmetic logic unit）
竞争冒险

重要内容一：组合逻辑电路设计

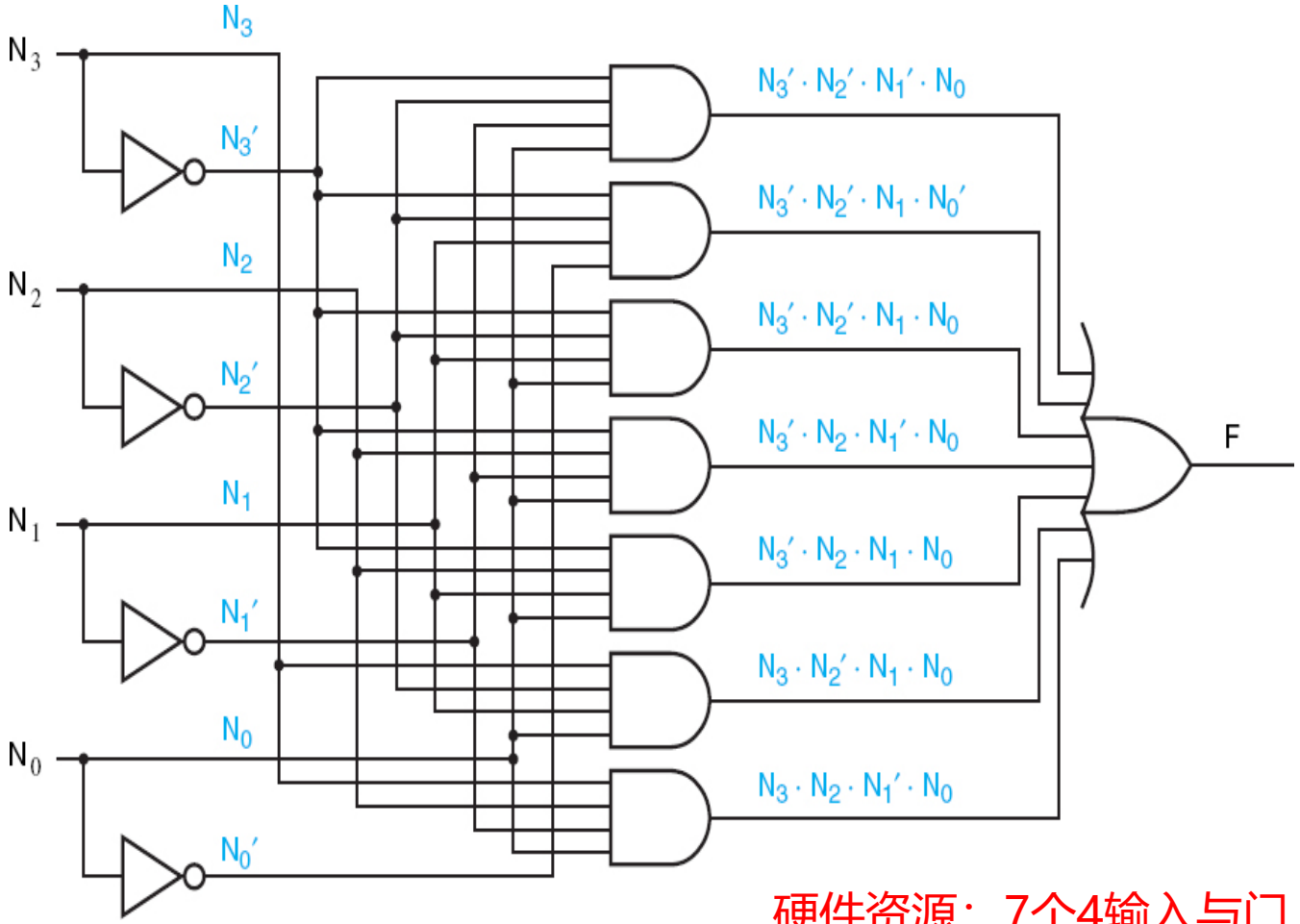
从文字描述到逻辑电路或系统设计的整个过程如下：



重要内容一：组合逻辑电路设计

例1：素数检测器的设计 4-bit input, $N_3N_2N_1N_0$

写出最小项表达式 $F = \Sigma_{N3N2N1N0}(1,2,3,5,7,11,13)$



硬件资源：7个4输入与门、1个7输入或门

列出真值表

row	N_3	N_2	N_1	N_0	F
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	0

重要内容一：组合逻辑电路设计

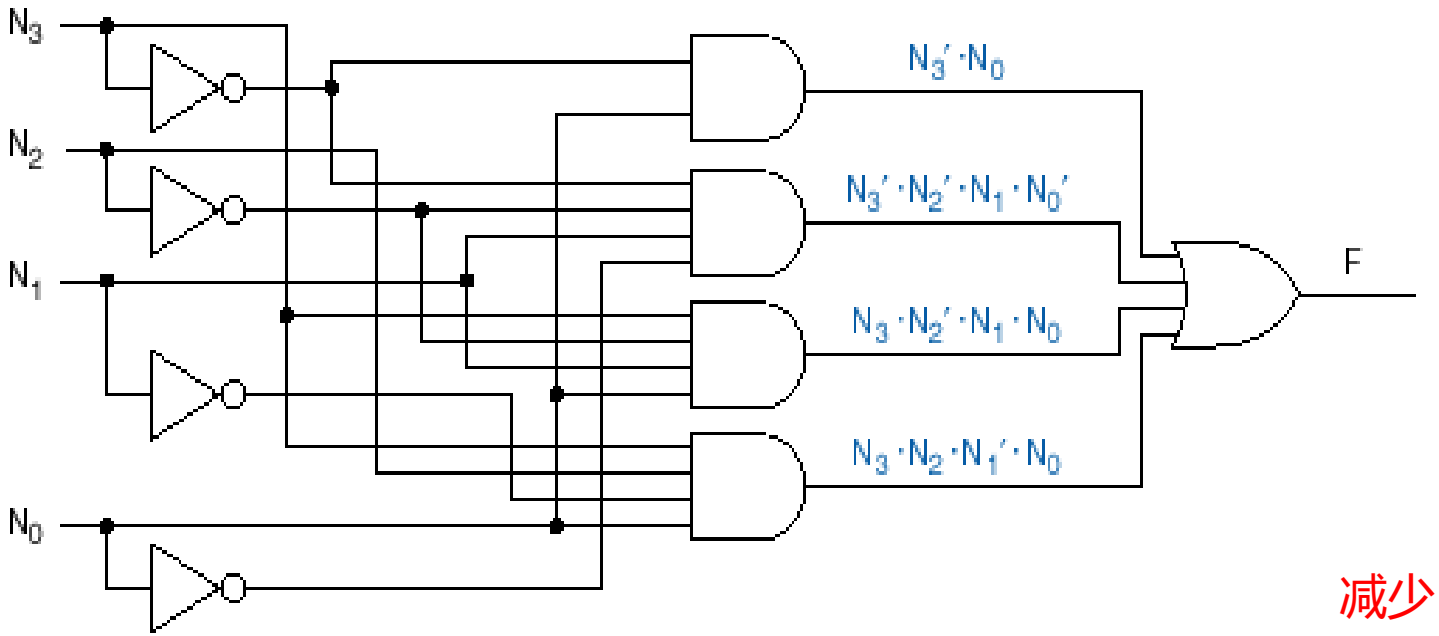
利用布尔代数化简，以减少逻辑门数和输入端数　X·Y+X·Y′ =X

$F = \Sigma_{N3N2N1N0}(1,2,3,5,7,11,13)$

$= N_3' \cdot N_2' N_1' N_0 + N_3' \cdot N_2' \cdot N_1 \cdot N_0 + N_3' \cdot N_2 \cdot N_1' \cdot N_0 + N_3' \cdot N_2 \cdot N_1 \cdot N_0 + \ldots$

$= N_3' N_2' \cdot N_0 + N_3' \cdot N_2 \cdot N_0 + \ldots$

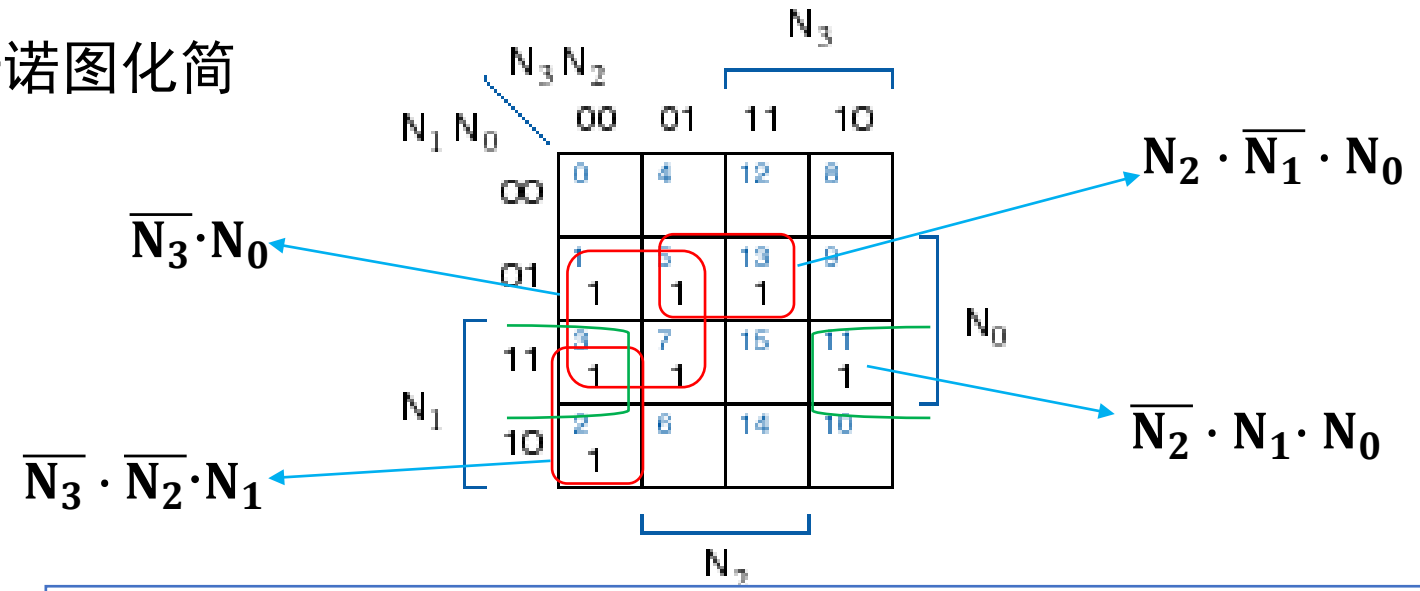
$= N_3' \cdot N_0 + N_3' \cdot N_2' \cdot N_1 \cdot N_0' + N_3 \cdot N_2' \cdot N_1 \cdot N_0 + N_3 \cdot N_2 \cdot N_1' \cdot N_0$



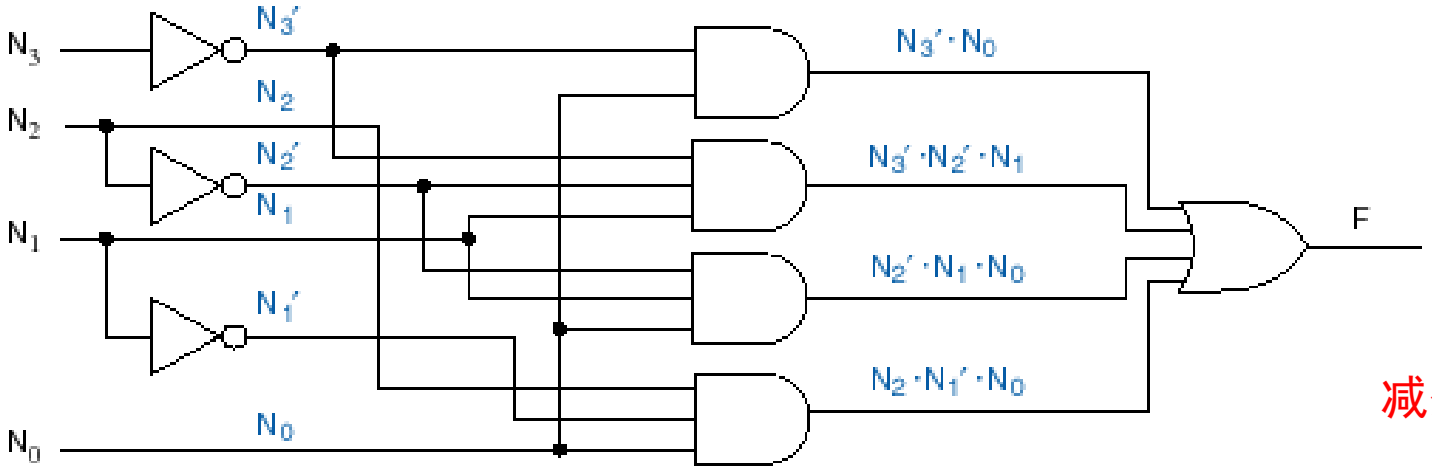
减少：3个与门和17个输入端

重要内容一：组合逻辑电路设计

按卡诺图化简



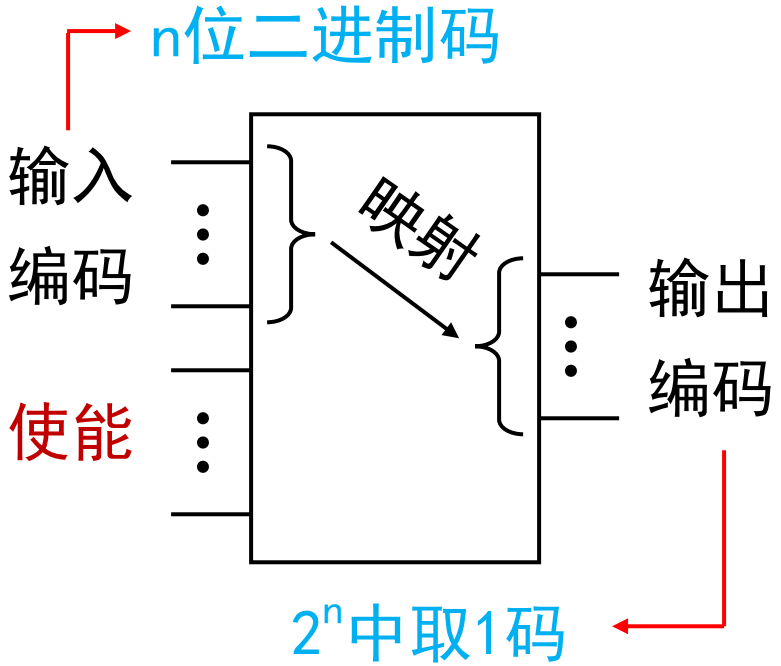
$F=N_3'\cdot N_0+N_3'\cdot N_2'\cdot N_1+N_2'\cdot N_1\cdot N_0+N_2\cdot N_1'\cdot N_0$



减少：3个与门和20个输入端

重要内容二：典型组合逻辑电路的使用

- 译码/编码功能：
  - 信号与信号编码之间的转换，如：n位信号编码←→2^n位信号
- 译码器（decoder）：一种多输入、多输出的组合电路。
  - “编码→信号”的转换，输入端数比输出端数少
  - 通常输出采用2^n中取1码(单热点, one-hot)编码表征信号
  - 可以通过使能端EN来控制电路实现映射功能

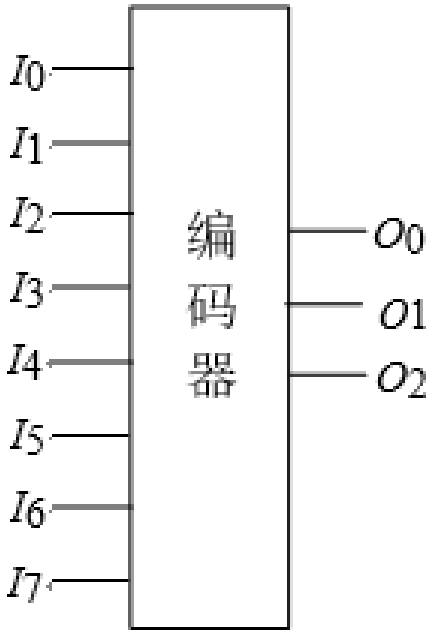


- n-2^n译码器
  - 输入：n位二进制编码
  - 输出：2^n中取1码
- 例如：
  - 2-4译码器
  - 3-8译码器
  - 4-16译码器

重要内容二：典型组合逻辑电路的使用

3位二进制编码器（8-3 编码器）

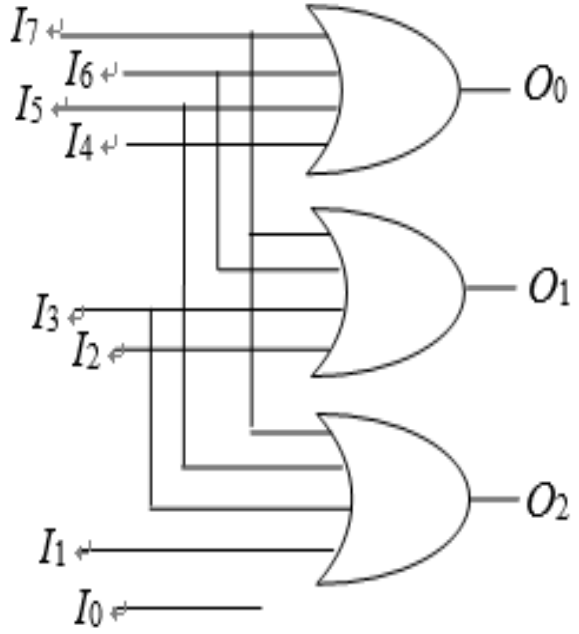
- 输入I_0--I_7是一组互斥变量，每次只有一个输入端I_i为1，其余都为0，输出为i的二进制编码。



a）编码器符号

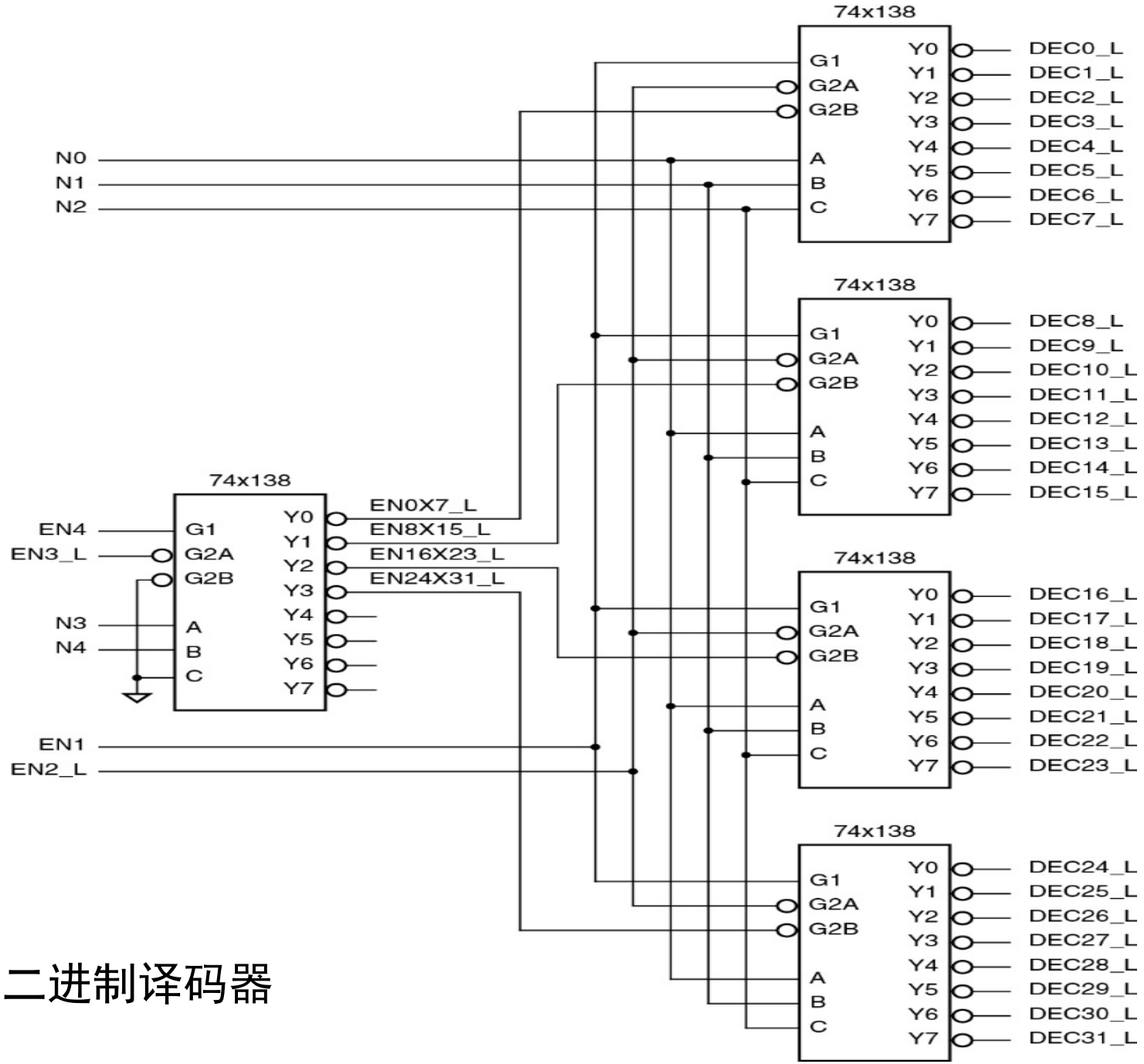
	O_0	O_1	O_2
I_0	0	0	0
I_1	0	0	1
I_2	0	1	0
I_3	0	1	1
I_4	1	0	0
I_5	1	0	1
I_6	1	1	0
I_7	1	1	1

b）编码器真值表



c）编码器电路图

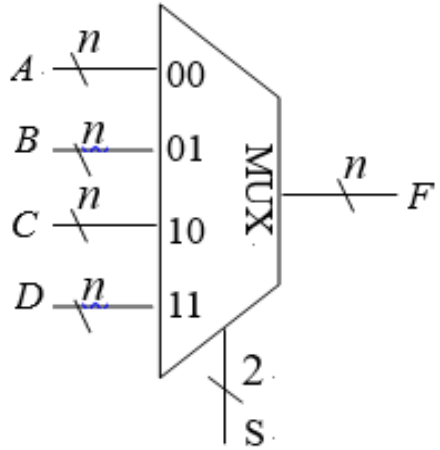
重要内容二：典型组合逻辑电路的使用



3-8译码器级联成5-32二进制译码器

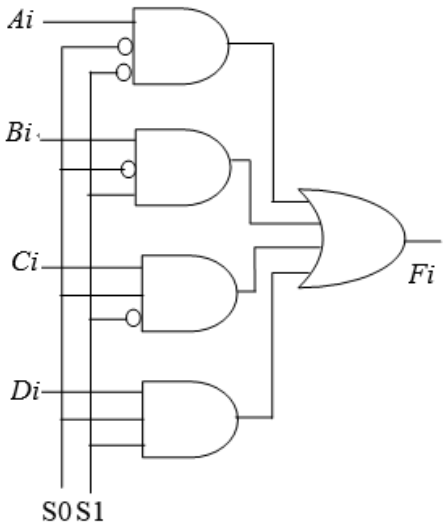
重要内容二：典型组合逻辑电路的使用

4-路选择器

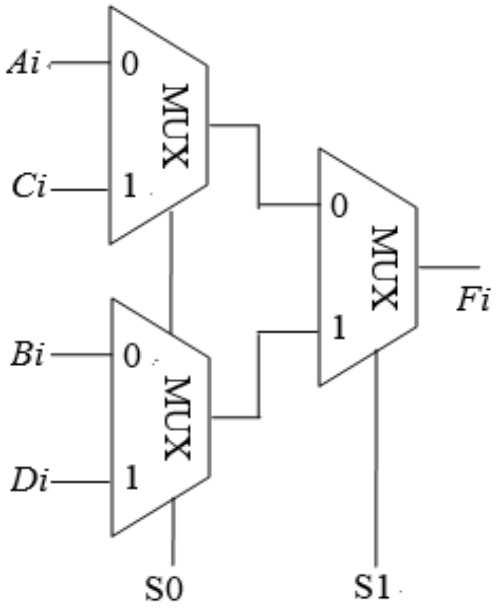


S0 S1	F
0 0	A
0 1	B
1 0	C
1 1	D

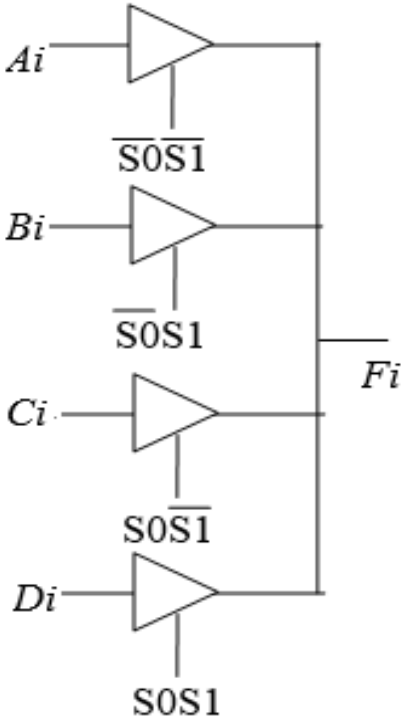
一位4-路选择器的实现



两级门电路



多层次级联



三态门电路

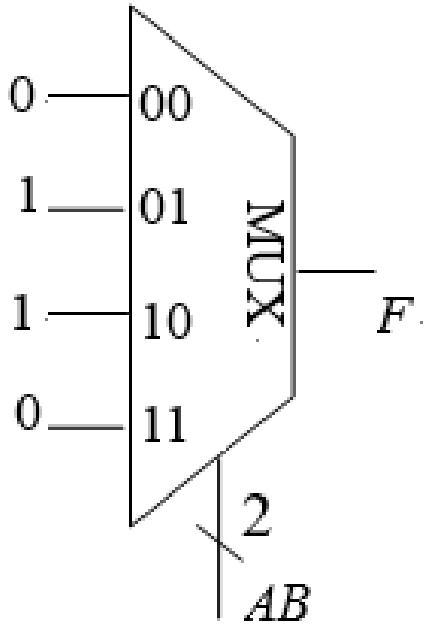
重要内容二：典型组合逻辑电路的使用

可以基于多路选择器实现组合逻辑电路的功能

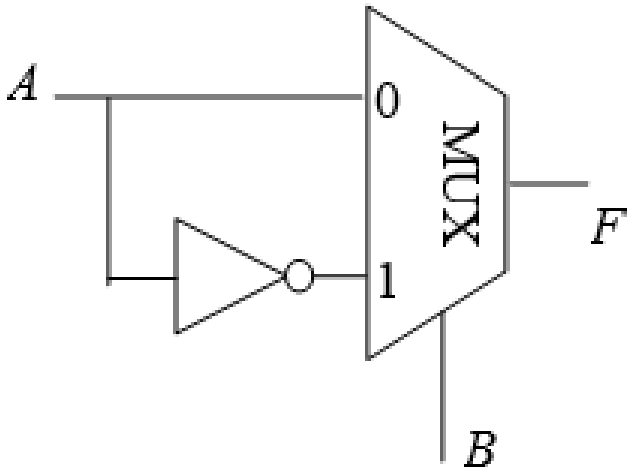
例1：基于多路选择器实现某组合逻辑电路的功能（可用如下真值表描述）

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

真值表



用一个4-路选择器实现

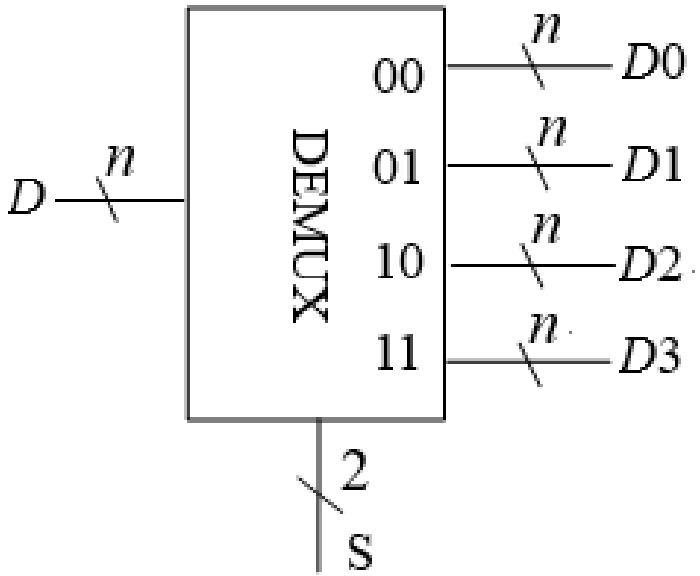


用一个2-路选择器和一个非门实现

重要内容二：典型组合逻辑电路的使用

多路分配器（demultiplexer）：把唯一的输入信号发送到多个输出端中的一个。从哪一个输出端送出输入信号，取决于控制端。简写为DMUX或DEMUX

4-路分配器的符号和真值表



四路分配器的符号

S0	S1	D0	D1	D2	D3
0	0	D	0	0	0
0	1	0	D	0	0
1	0	0	0	D	0
1	1	0	0	0	D

四路分配器真值表

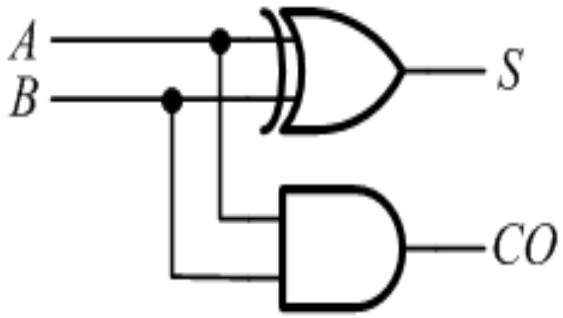
重要内容二：典型组合逻辑电路的使用

- 半加器（Half Adder，简称HA）：仅考虑加数和被加数，不考虑低位来的进位

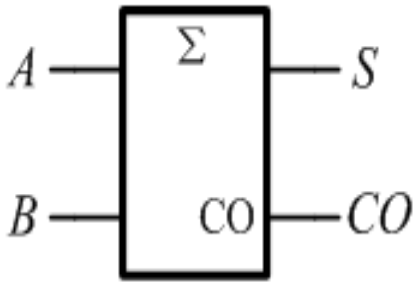
输入		输出	
被加	加数	和数	进位
A	*B*	*S*	*CO*
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$S=\overline{A}B+A\overline{B}=A\oplus B$$

$$CO=A\cdot B$$



电路图



逻辑符号

重要内容二：典型组合逻辑电路的使用

全加器（Full Adder，简称FA）：输入为加数、被加数和低位进位Cin，输出为和F、进位Cout

A	B	Cin	F	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

真值表

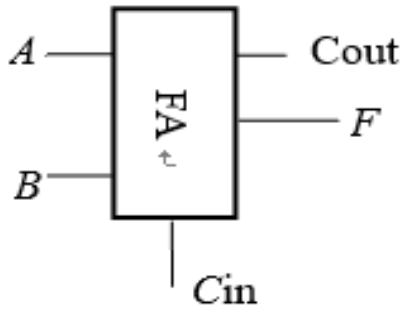
$$F=\overline{A}\cdot\overline{B}\cdot Cin+\overline{A}\cdot B\cdot\overline{Cin}+A\cdot\overline{B}\cdot\overline{Cin}+A\cdot B\cdot Cin$$

$$Cout=\overline{A}\cdot B\cdot Cin+A\cdot\overline{B}\cdot Cin+A\cdot B\cdot\overline{Cin}+A\cdot B\cdot Cin$$

化简后：

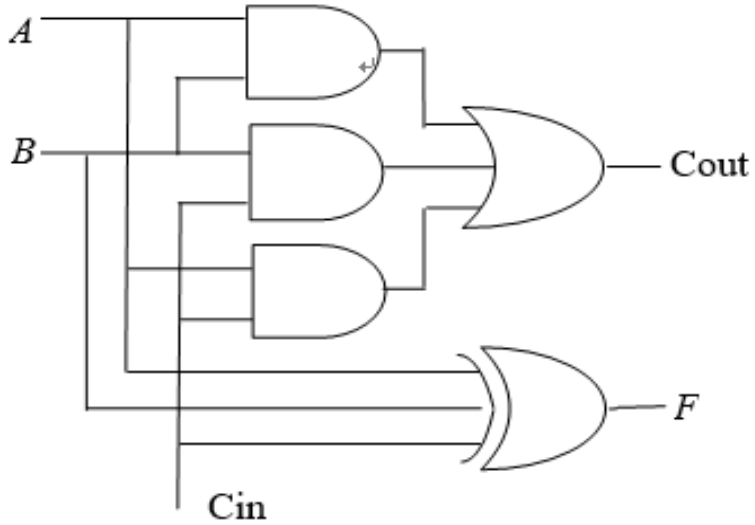
$$F=A\oplus B\oplus Cin$$

$$Cout=A\cdot B+A\cdot Cin+B\cdot Cin$$



逻辑符号

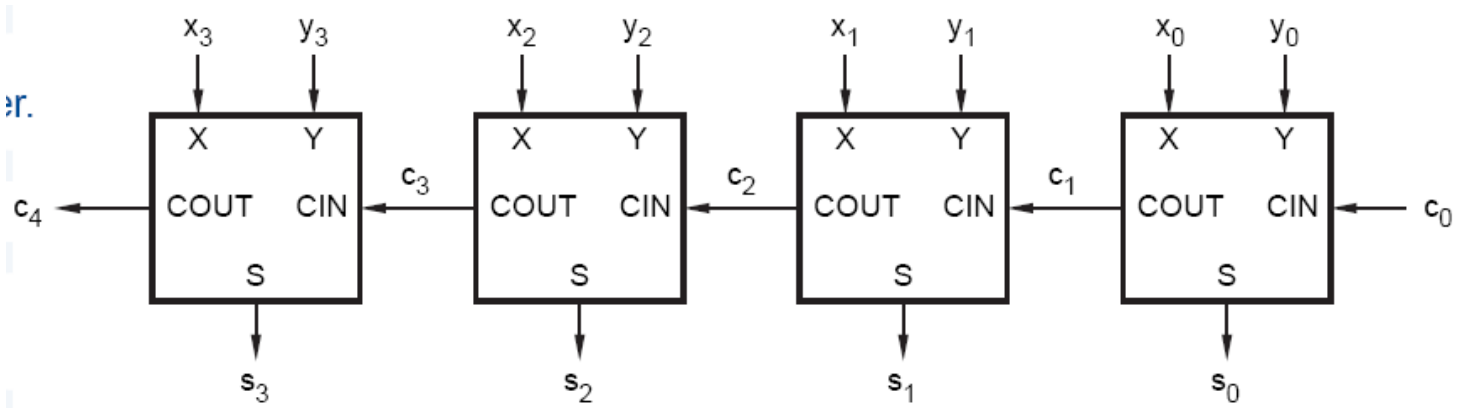
全加器逻辑电路图



重要内容二：典型组合逻辑电路的使用

串行进位加法器：行波进位加法器 ripper adder

- 规格：
  - 两个二进制字，每个n位，相加。
- 方法：
  - n个全加器的级联，属于迭代电路。
- 延迟：
- 特点：简单、速度慢



第4章 时序逻辑电路

第一讲 时序逻辑电路与典型部件

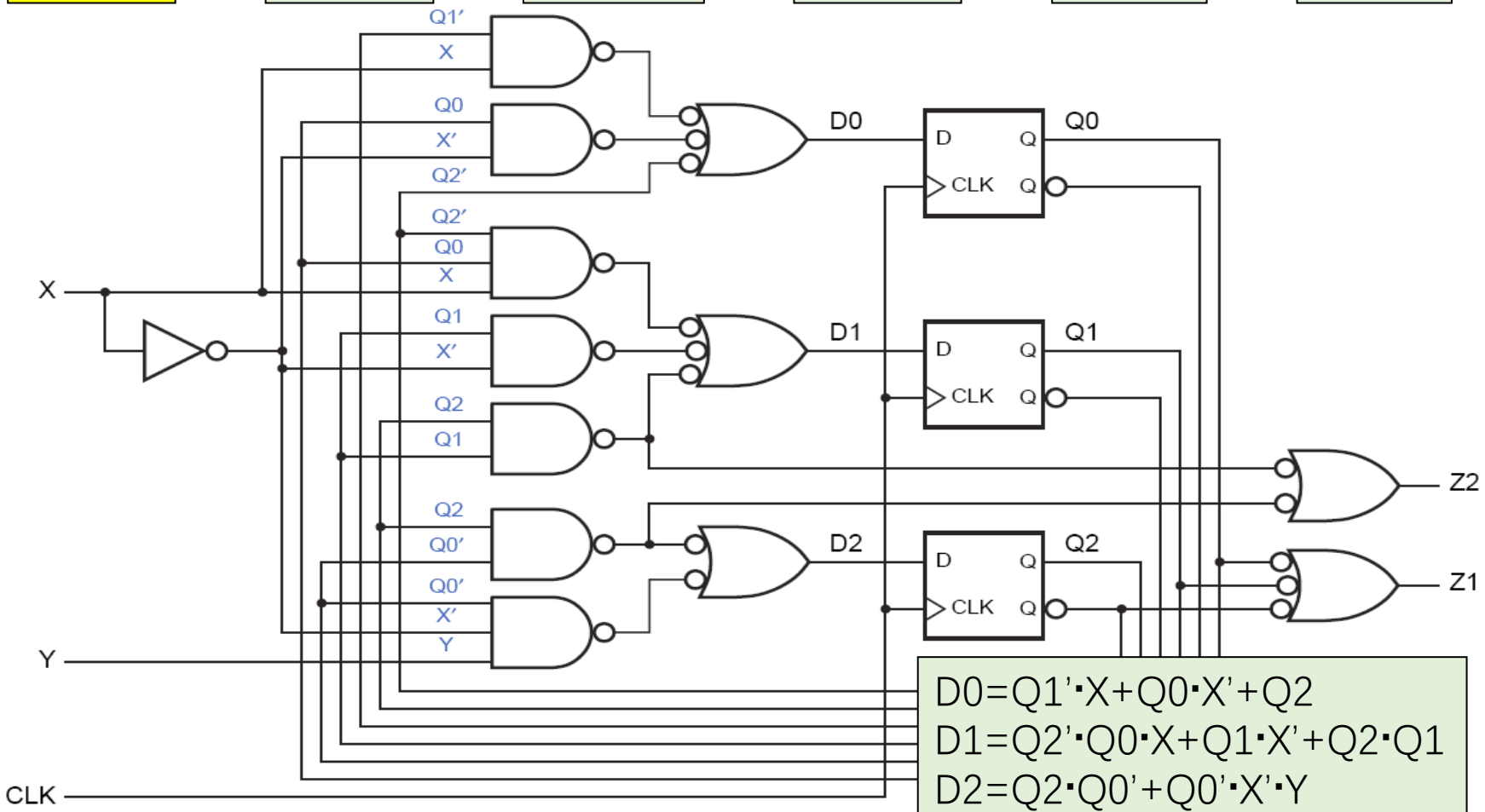
第二讲 时序逻辑电路设计

重要概念

时序逻辑电路（sequential logic circuit）
有限状态机（f inite state machine）
状态图（state diagram）
状态表（state table）
时钟周期（clock cycle）
时钟边沿（clock edge）
锁存器（latch）、触发器（flip-flop）
建立时间（setup time）、
锁存延迟（Clk-to-Q delay）、保持时间（hold time）
同步计数器（synchronous counter）、移位寄存器
（shift register）

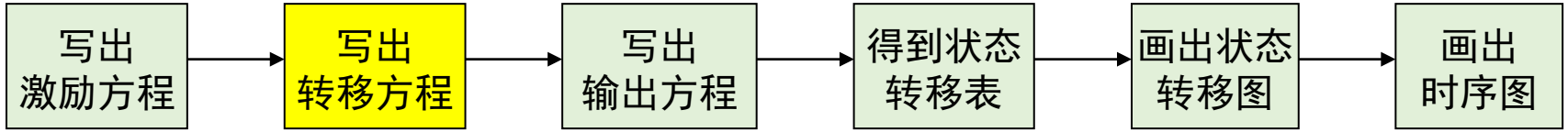
重点内容一：同步时序逻辑电路分析

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



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 $Q^{n+1}=D$

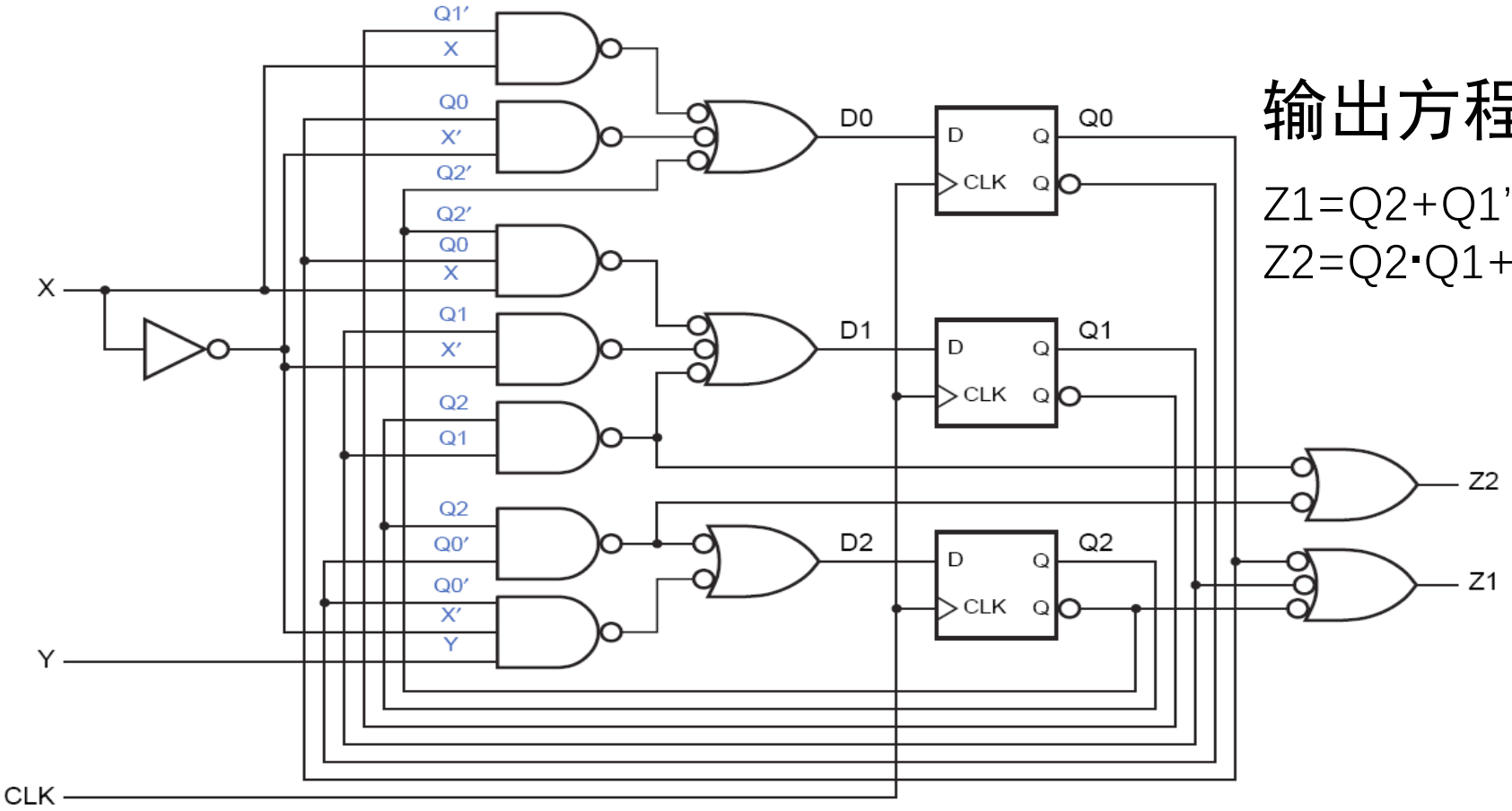
转移方程

$Q0^{n+1}$=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



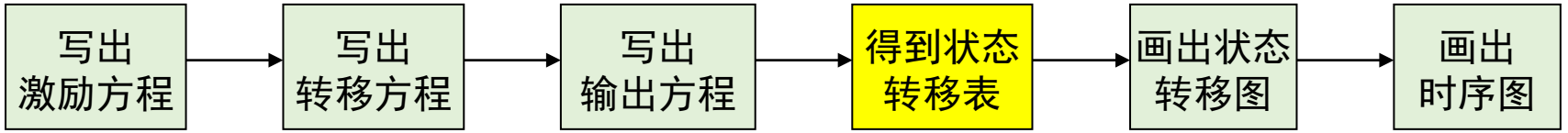
输出方程

$Z1=Q2+Q1'+Q0'$

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

同步时序逻辑电路分析

第四步：构建转移表



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

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

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

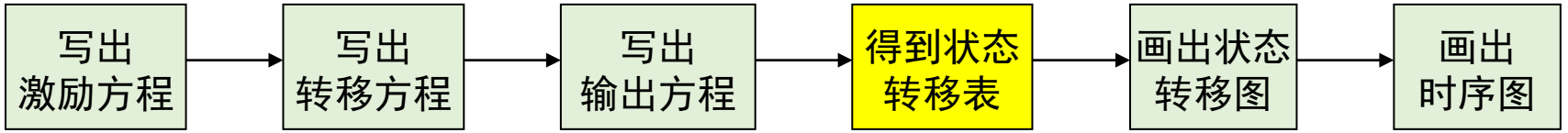
$Z1=Q2+Q1'+Q0'$

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

	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

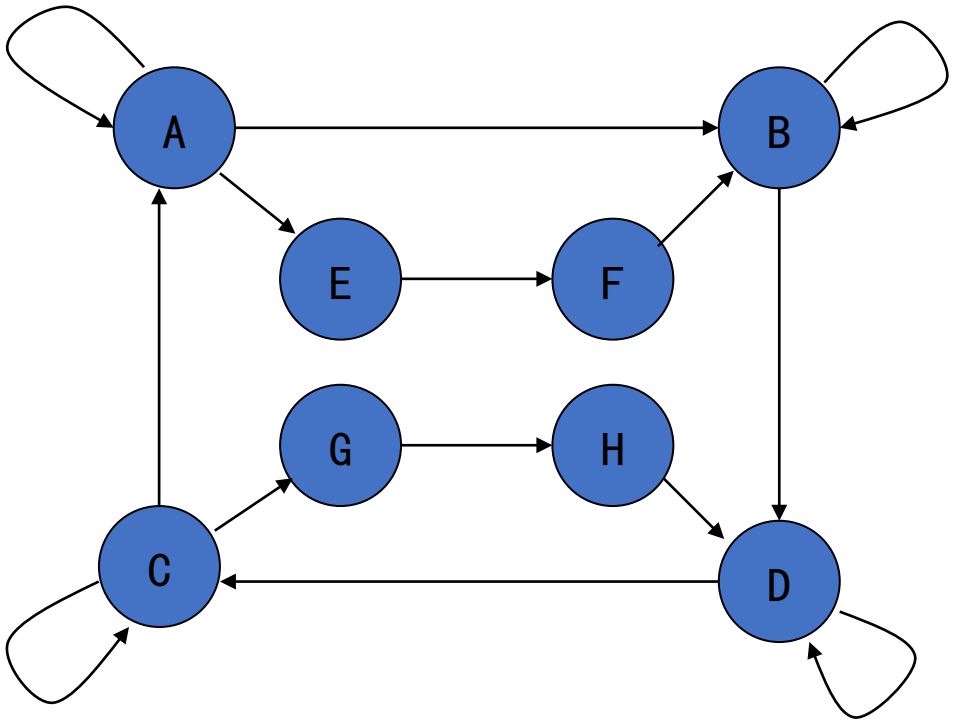
同步时序逻辑电路分析

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

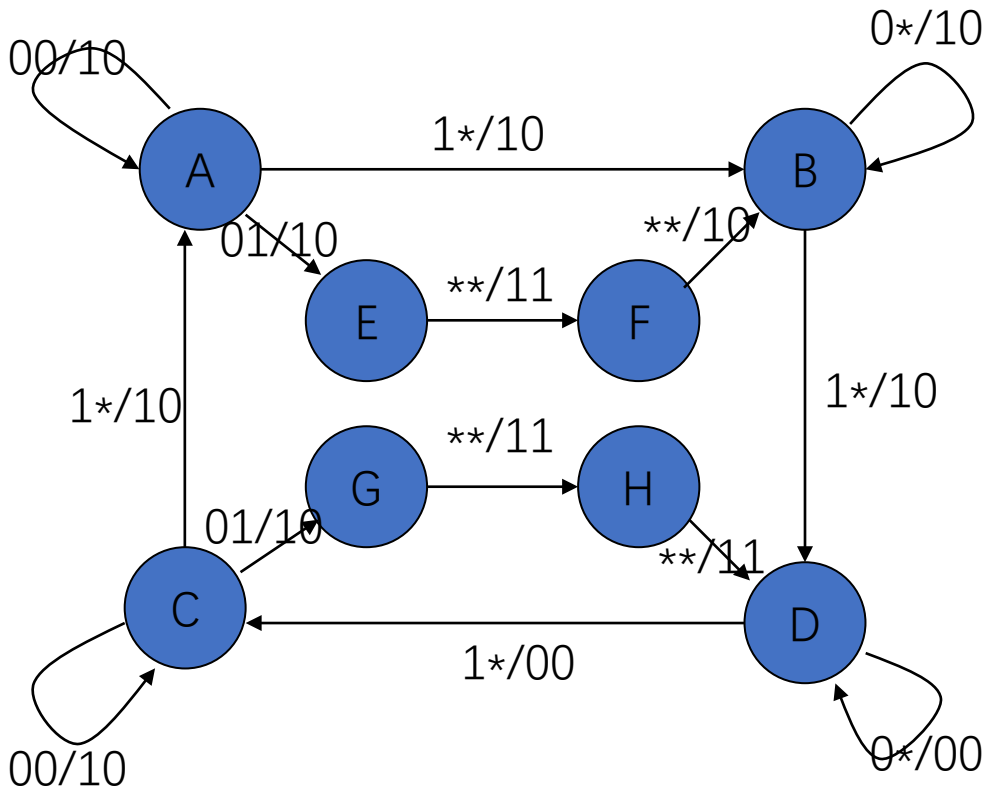
同步时序逻辑电路分析

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。

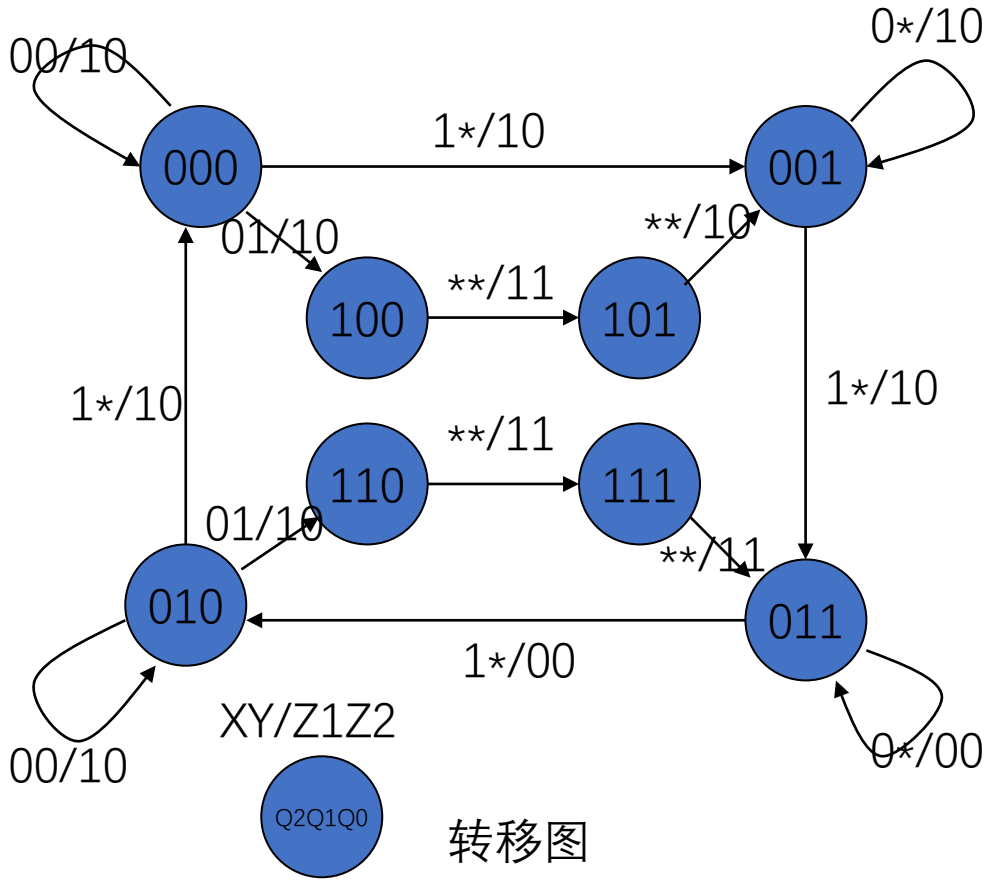
同步时序逻辑电路分析

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

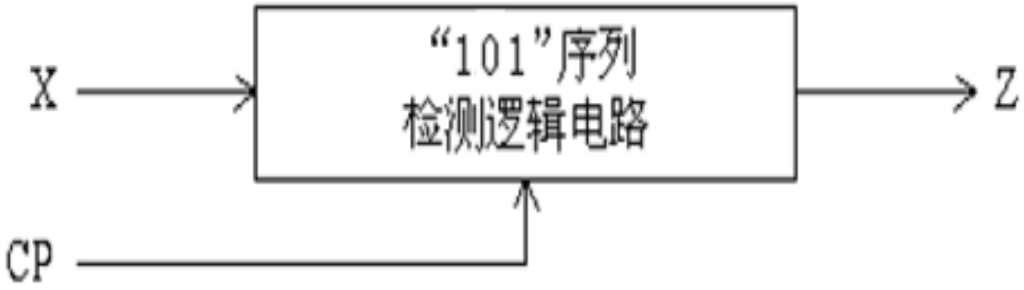
*表示与该输入无关

重点内容二：电路设计与分析-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

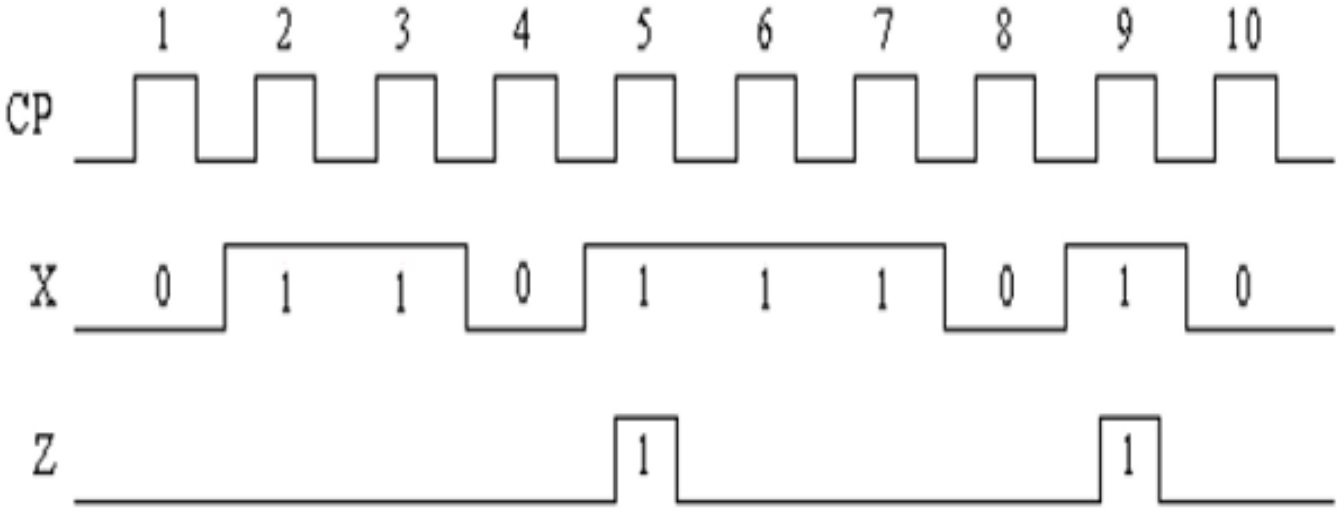
例：设计一个能检测出一连串外部输入中是否出现了0/1序列“101”的状态机。

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。



重点内容二：电路设计与分析-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；

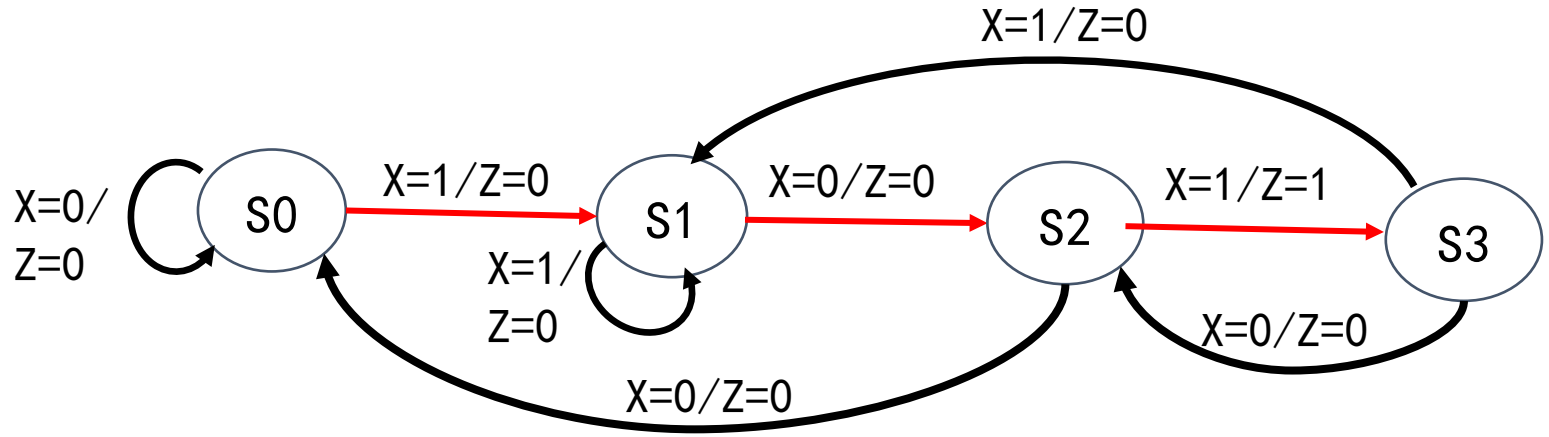
IV. 重复第II、III两步，直到不产生新状态为止。

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



重点内容二：电路设计与分析-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

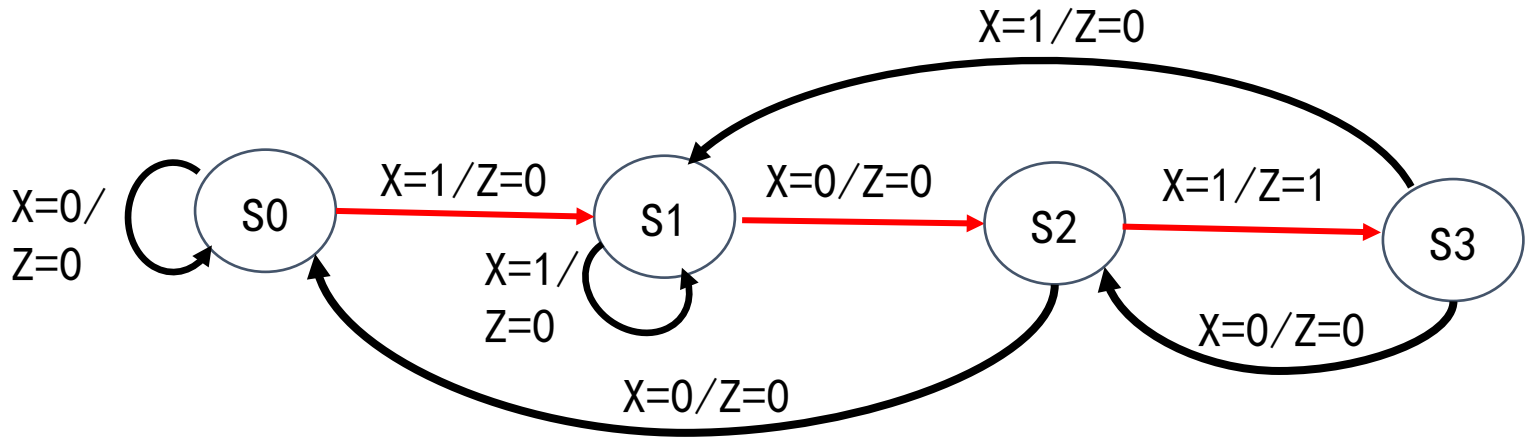
现态S	S*/Z	
	X=0	X=1
S0	S0/0	S1/0
S1	S2/0	S1/0
S2	S0/0	S3/1
S3	S2/0	S1/0

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



重点内容二：电路设计与分析-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。

重点内容二：电路设计与分析-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

重点内容二：电路设计与分析-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

重点内容二：电路设计与分析-状态图/状态表设计

- **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

重要内容三：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

重要内容三：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：

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

重要内容三：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•$\overline{X}$

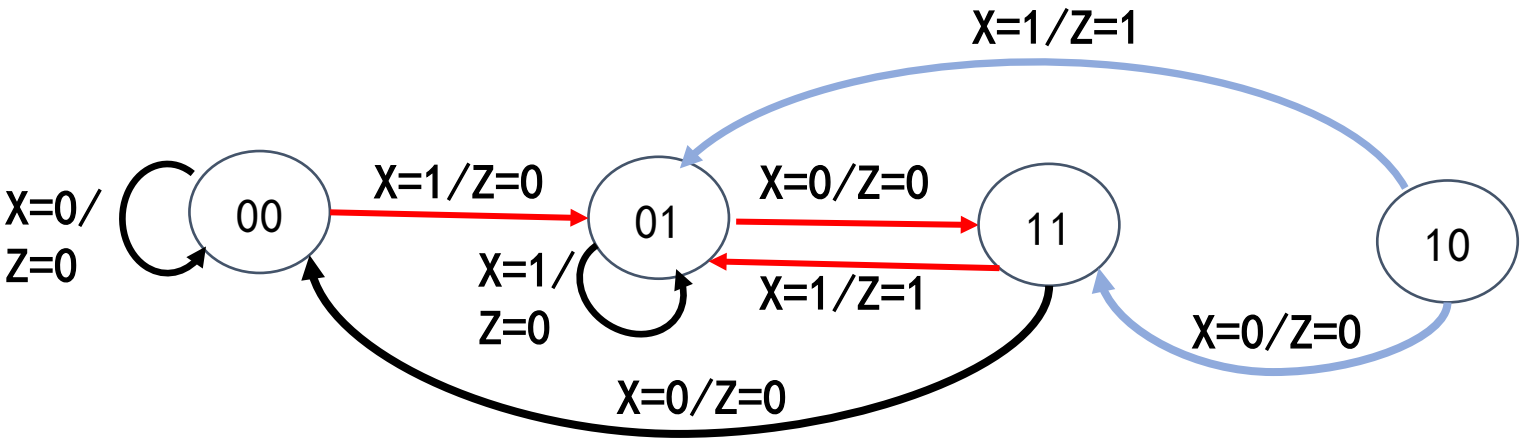
    D0=Y0*= $\overline{Y1}$•Y0+ X
- 输出函数为：Z=Y1 •X

						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

重要内容三：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*=$\overline{Y1}$•Y0•$\overline{X}$

D0=Y0*= $\overline{Y1}$•Y0+ X

Z=Y1 •X

重要内容三：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

重要内容三：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

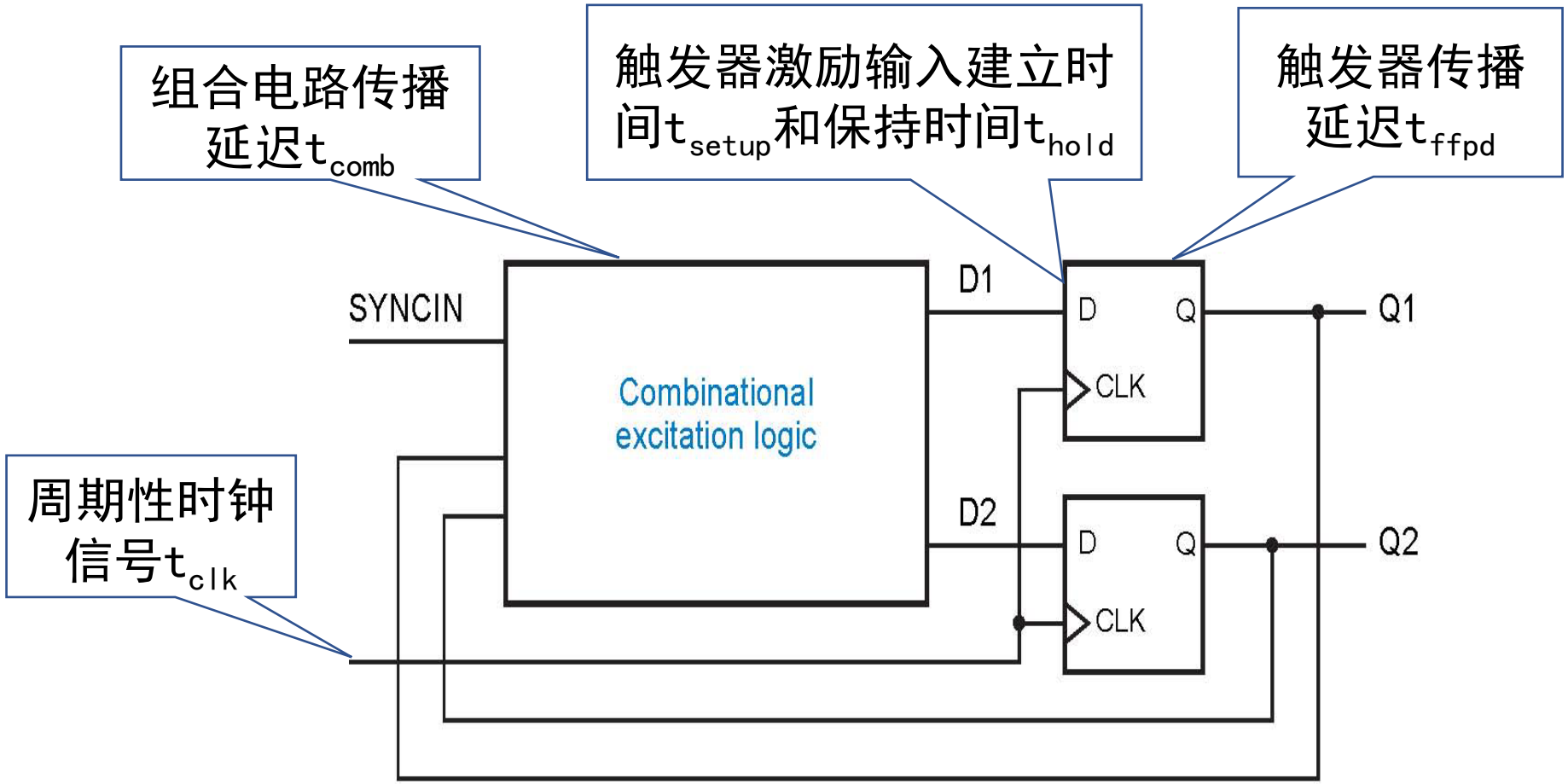
- 重新设计逻辑电路中的输出模块

$Z=Y1\cdot Y0\cdot X$　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

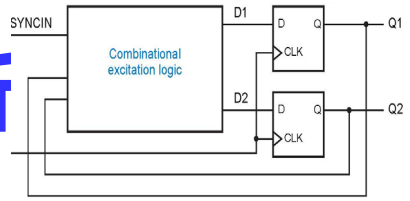
重要内容三：电路设计与分析

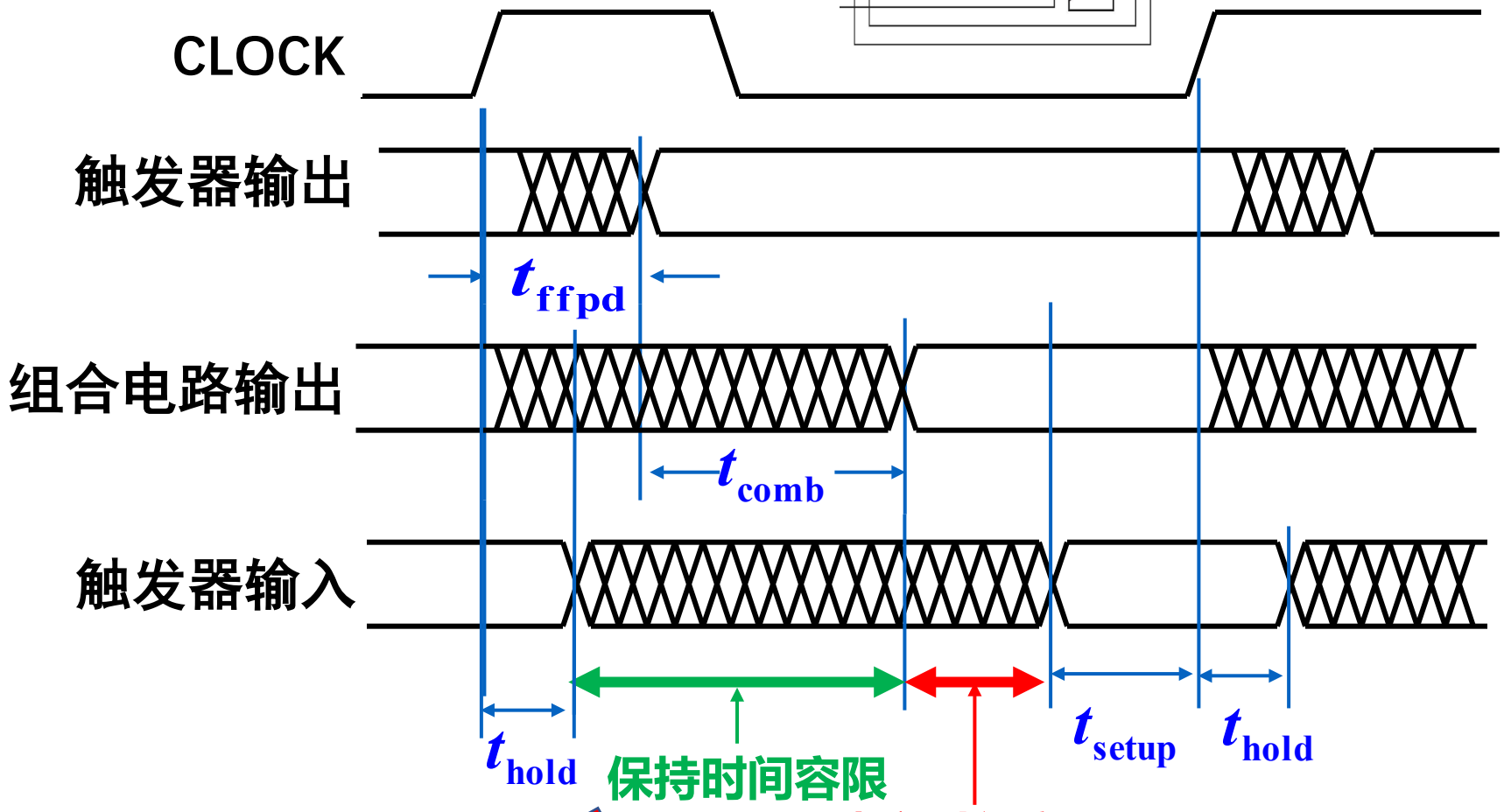
电路定时分析

- 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。



重要内容三：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

重要内容三：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} 。

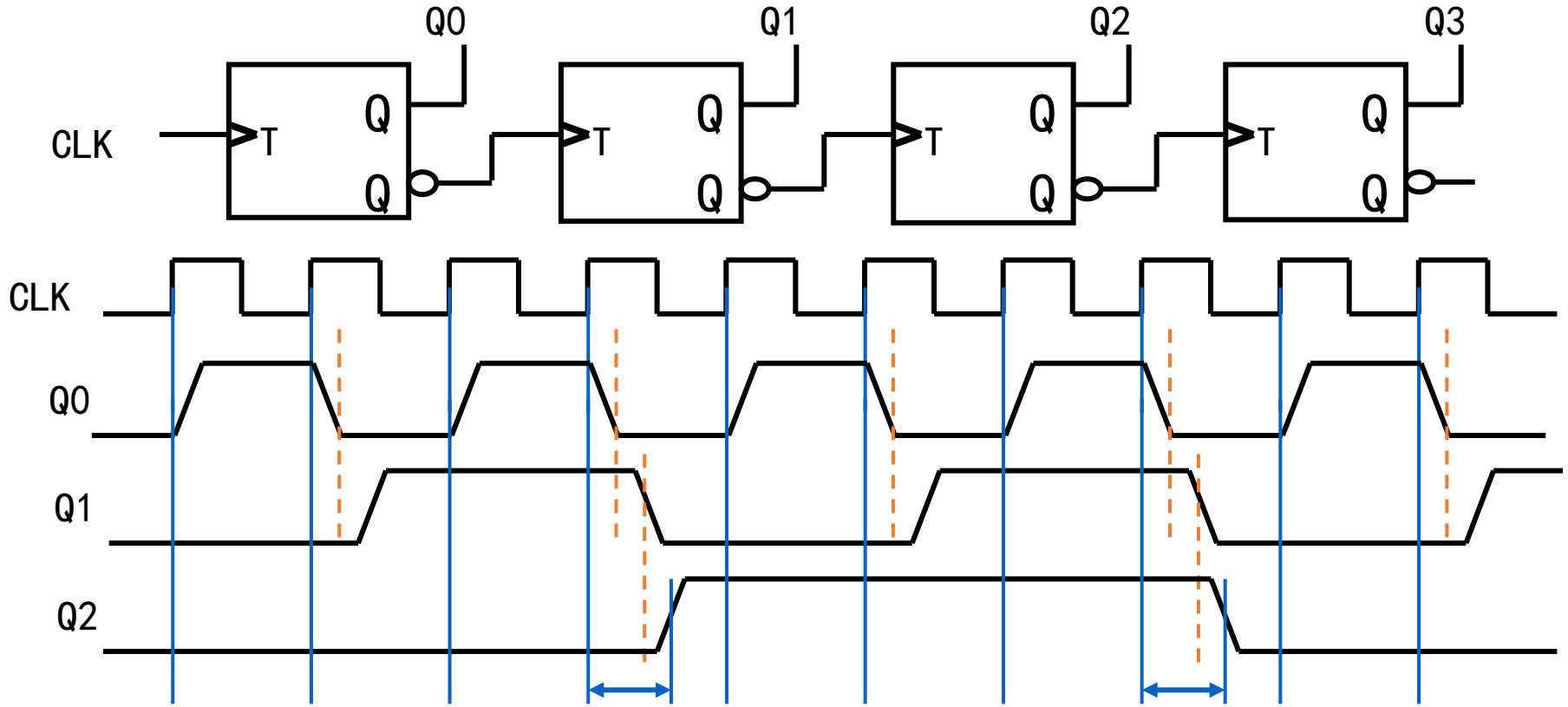
重要内容三：典型时序逻辑电路

- 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

重要内容四：典型时序逻辑电路

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

重要内容四：典型

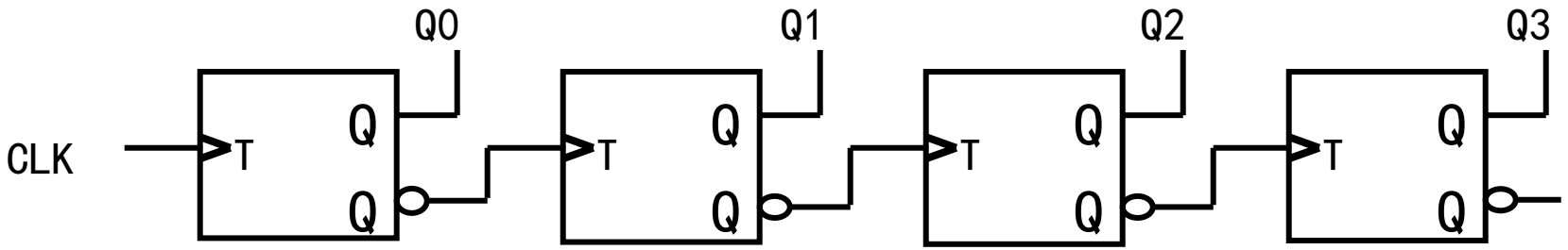
当编码为1111时，下个时钟到达后，经过$n \times t_{TQ}$ 延时，又回到编码0000

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；

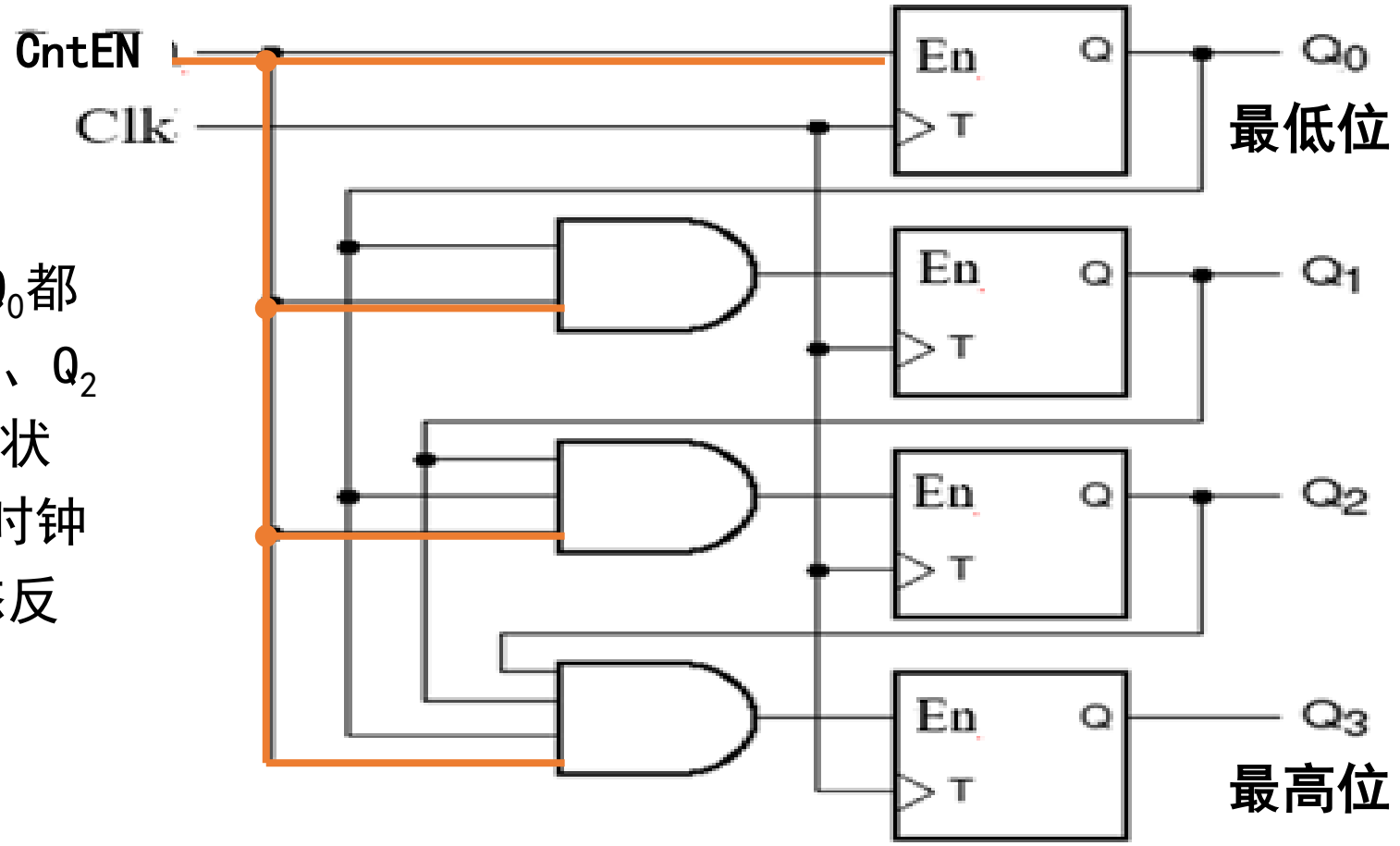
第3个Clk有效信号到来后，Q0从0变成1，此时其他三个状态位不变，因而得到状态编码0011；

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；……。

重要内容四：典型时序逻辑电路

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。

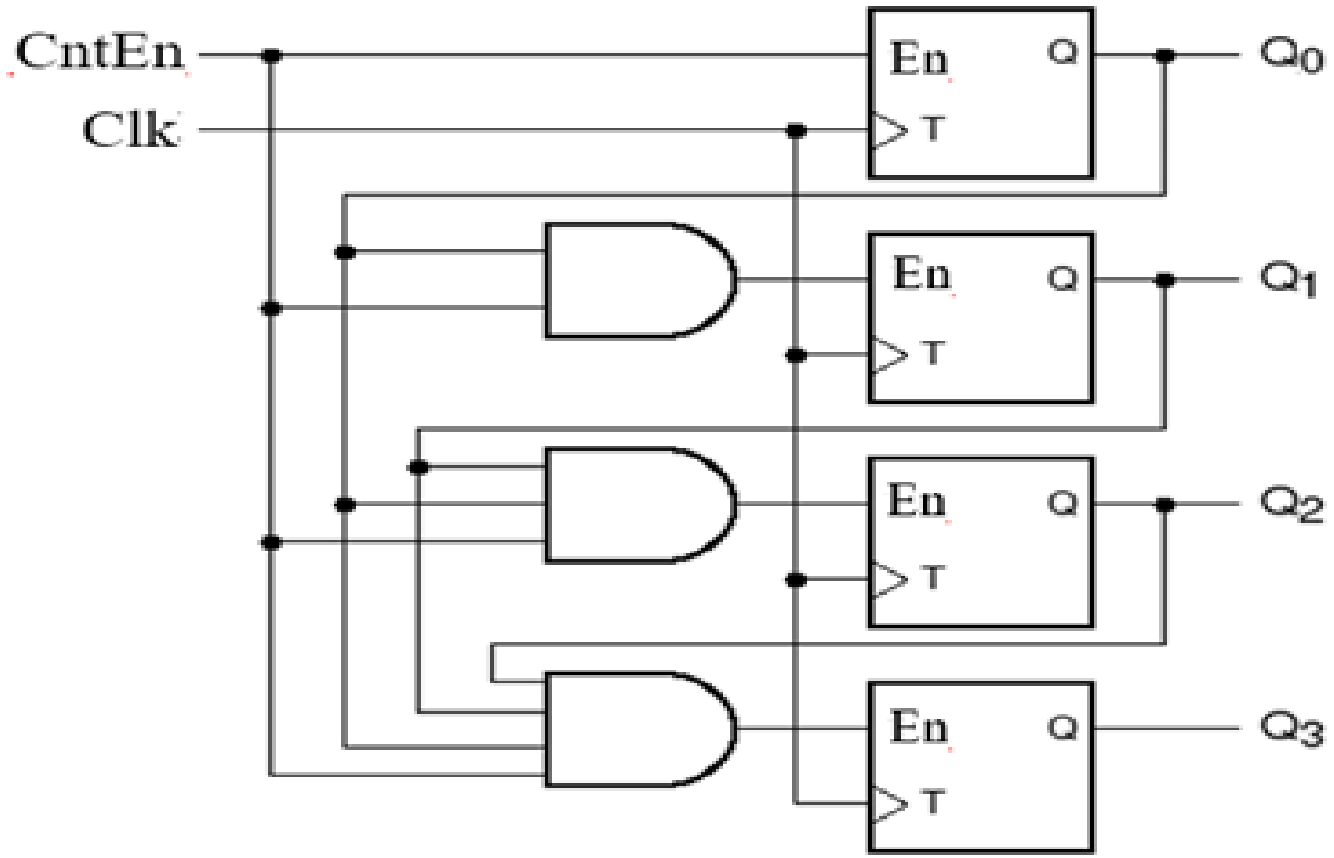
重要内容四：典型时序逻辑

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。



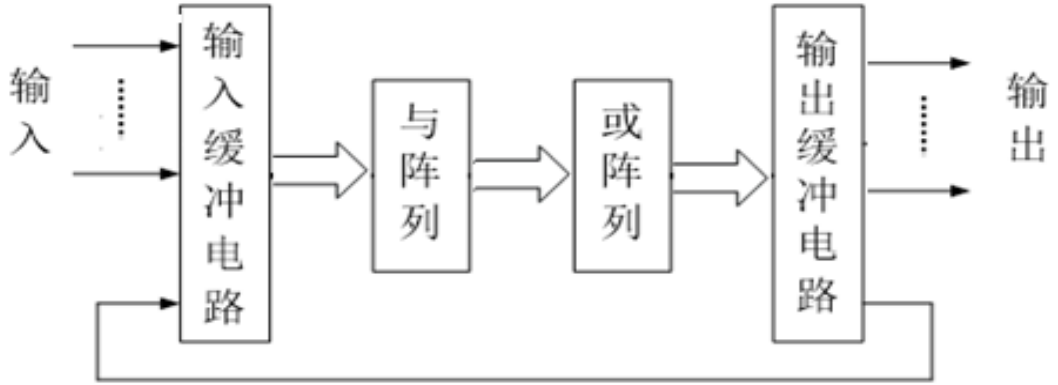
重要内容四：典型时序逻辑电路

寄存器

移位寄存器。。。

第5章 PLD器件

PLD器件



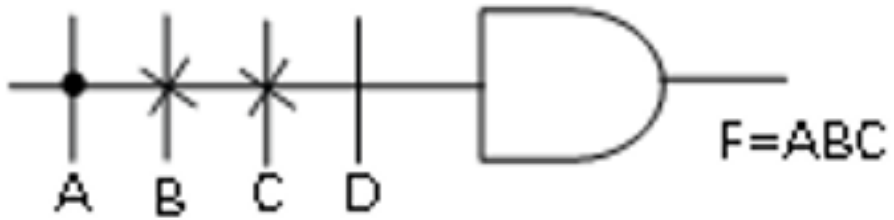
PLD结构框图

PLD中基本电路符号

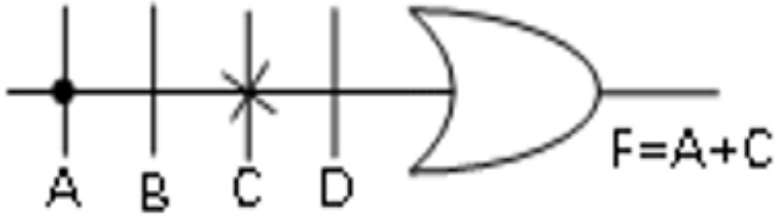
互补缓冲器

阵列连线

与阵列表示

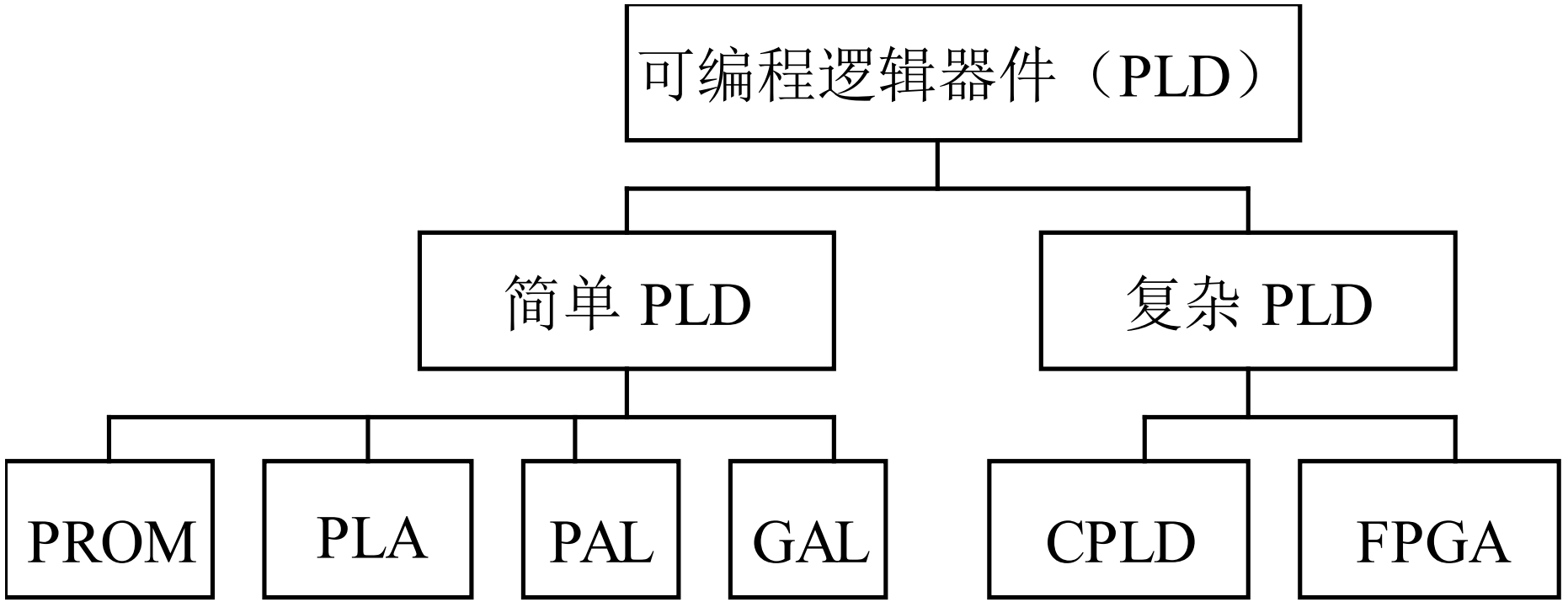


或阵列表示



PLD器件符号基本框架和用法

PLD分类



存储器阵列

- 只读存储器(Read-only Memory，ROM)和随机存取存储器(Random-access Memory，RAM)
- 静态RAM（Static RAM，SRAM）
- 动态RAM（Dynamic RAM，DRAM）

第6章 运算方法和运算部件

第一讲 运算部件

第二讲 运算方法

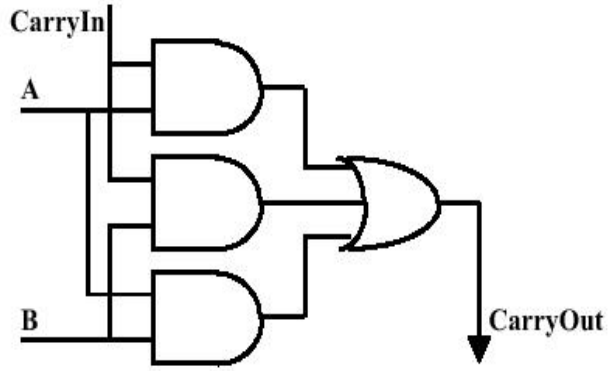
重要概念

逻辑移位（logical shift）、算术移位（arithmetic shift）、
行波进位加法器（Ripple Carry Adder，RCA）
先行进位加法器（Carry Lookahead Adder，CLA）
算术逻辑单元（Arithmetic Logic Unit，ALU）
标志（flag）

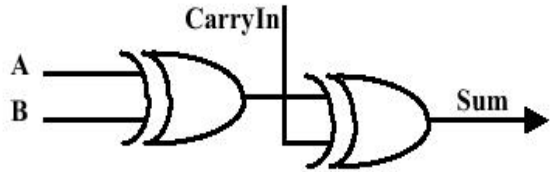
重要内容一：加法器

CarryOut 和 Sum 的逻辑图

- CarryOut = B & CarryIn | A & CarryIn | A & B



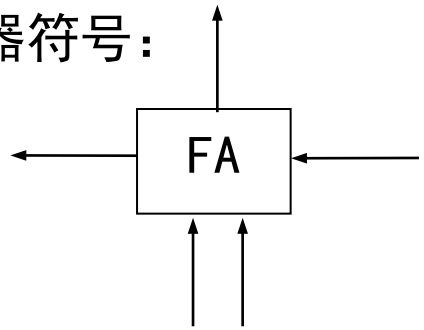
- Sum = A XOR B XOR CarryIn



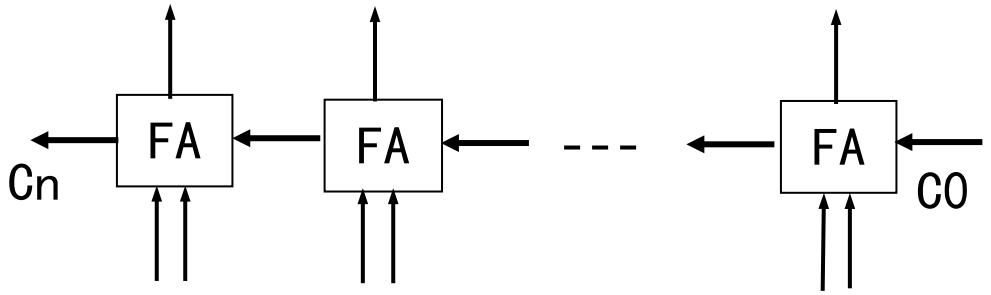
假定与/或门延迟为1ty，异或门3ty，则“和”与“进位”的延迟为多少？

Sum延迟为6ty；Carryout延迟为2ty。

全加器符号：



n位串行(行波)加法器：



串行加法器的缺点：进位按串行方式传递，速度慢！

问题：n位串行加法器从CO到Cn的延迟时间为多少？ 2n级门延迟！

最后一位和数的延迟时间为多少？

2n+1级门延迟！ n>2

重要内容一：加法器

- 为什么用先行进位方式？

  串行进位加法器采用串行逐级传递进位，电路延迟与位数成正比关系，太慢了。因此，现代计算机采用一种先行进位(Carry look ahead)方式。

- 如何产生先行进位？

  定义辅助函数：$G_i=A_iB_i$…进位生成函数

  $P_i=A_i+B_i$…进位传递函数（或 $P_i=A_i \oplus B_i$ ）

  通常把实现上述逻辑的电路称为进位生成/传递部件

  全加逻辑方程：$S_i= A_i \oplus B_i \oplus C_{i-1}$　　$C_i=G_i+P_iC_{i-1}$ （i=1,…n)

  设n=4,则：$C_1=G_0+P_0C_0$

  $C_2=G_1+P_1C_1=G_1+P_1G_0+P_1P_0C_0$

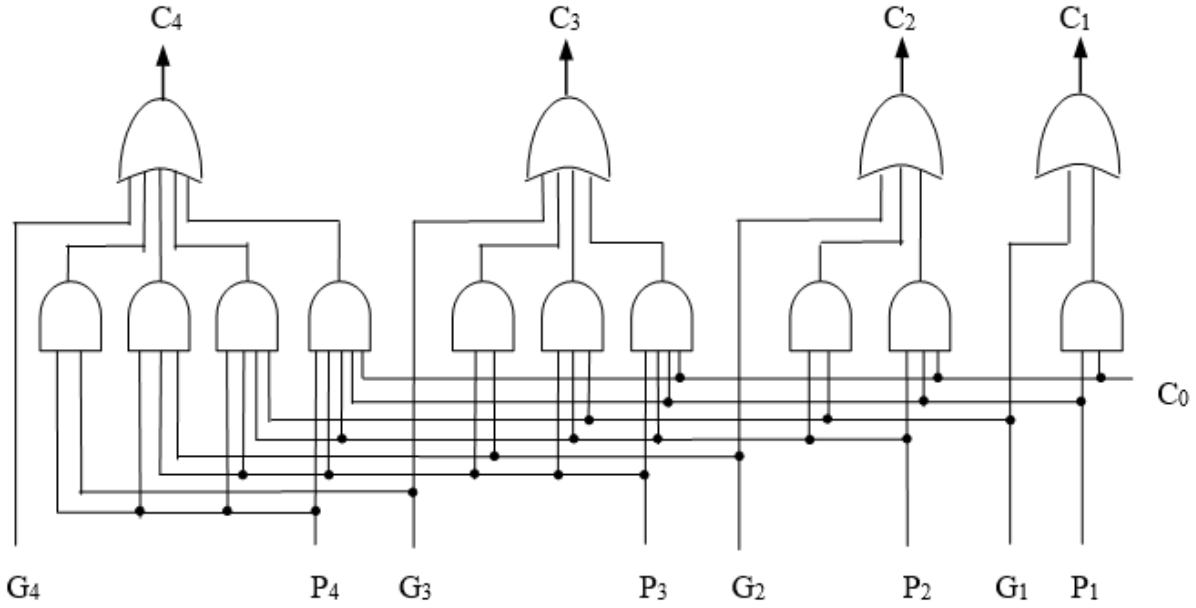
  $C_3=G_2+P_2C_2=G_2+P_2G_1+P_2P_1G_0+P_2P_1P_0C_0$

  $C_4=G_3+P_3C_3=G_3+P_3G_2+P_3P_2G_1+P_3P_2P_1G_0+P_3P_2P_1P_0C_0$

  由上式可知:各进位之间无等待，可以独立并同时产生。

  通常把实现上述逻辑的电路称为4位CLU部件

重要内容一：加法器



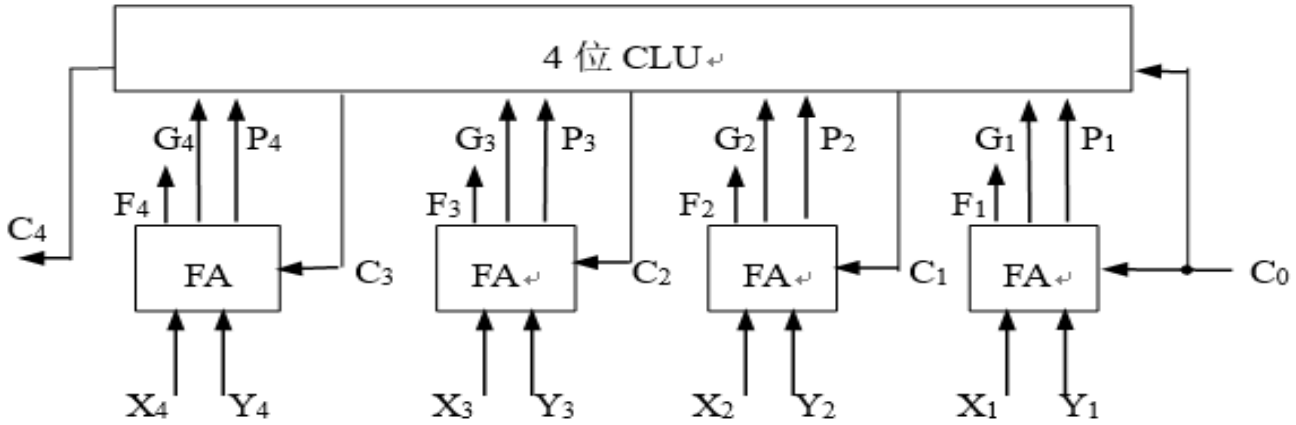
$C_1=G_1+P_1C_0$

$C_2=G_2+P_2C_1=G_2+P_2G_1+P_2P_1C_0$

$C_3=G_3+P_3C_2=G_3+P_3G_2+P_3P_2G_1+P_3P_2P_1C_0$

$C_4=G_4+P_4C_3=G_4+P_4G_3+P_4P_3G_2+P_4P_3P_2G_1+P_4P_3P_2P_1C_0$

4位CLU部件



$G_i=A_iB_i$

$P_i=A_i+B_i$（或 $P_i=A_i\oplus B_i$ ）

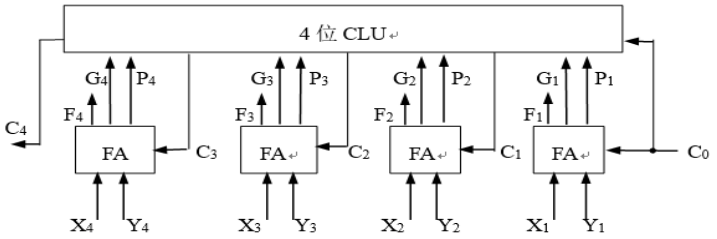
$F_i=A_i\oplus B_i\ \oplus C_{i-1}$

4位CLA加法器

重要内容一：加法器

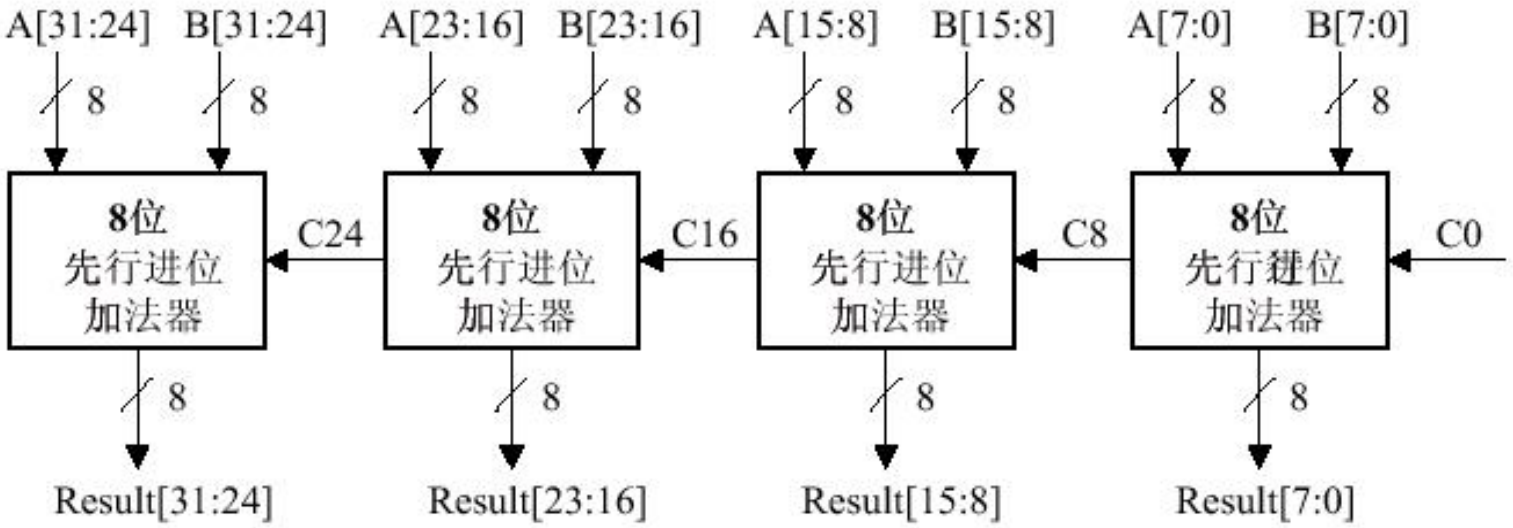
局部先行进位加法器（Partial Carry Lookahead Adder）

或称 单级先行进位加法器



- 实现全先行进位加法器的成本太高
  - 想象 **Cin31**的逻辑方程的长度
- 一般性经验:
  - 连接一些**N**位先行进位加法器，形成一个大加法器
  - 例如: 连接**4**个**8**位进位先行加法器，形成**1**个 **32**位局部先行进位加法器

问题：所有和数产生的延迟为多少？3+2+2+5=12ty



重要内容一：加法器

- 单级(局部)先行进位加法器的进位生成方式：

  “组内并行、组间串行”

- 所以，单级先行进位加法器虽然比行波加法器延迟时间短，但高位组进位依赖低位组进位，故仍有较长的时间延迟
- 通过引入组进位生成/传递函数实现“组内并行、组间并行”进位方式

设n=4, 则：　$C_1=G_1+P_1C_0$

$C_2=G_2+P_2C_1=G_2+P_2G_1+P_2P_1C_0$

$C_3=G_3+P_3C_2=G_3+P_3G_2+P_3P_2G_1+P_3P_2P_1C_0$

$C_4=G_4+P_4C_3=G_4+P_4G_3+P_4P_3G_2+P_4P_3P_2G_1 + P_4P_3P_2P_1C_0$

$G_4^*=G_4+P_4C_3=G_4+P_4G_3+P_4P_3G_2+P_4P_3P_2G_1$

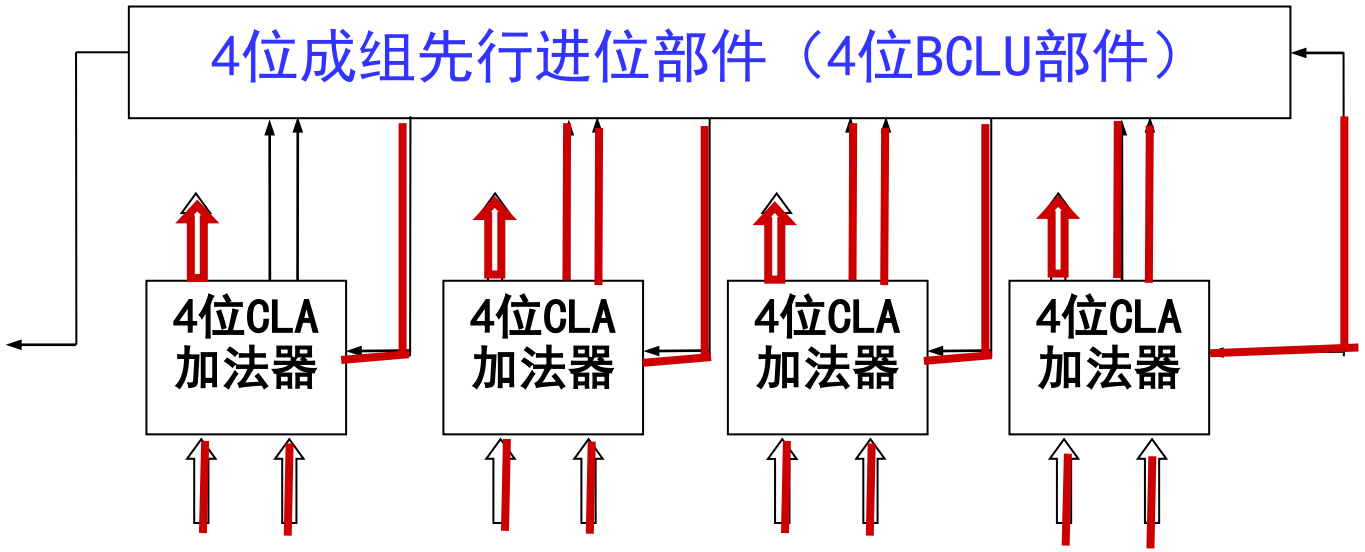
$P_4^*=P_4P_3P_2P_1$

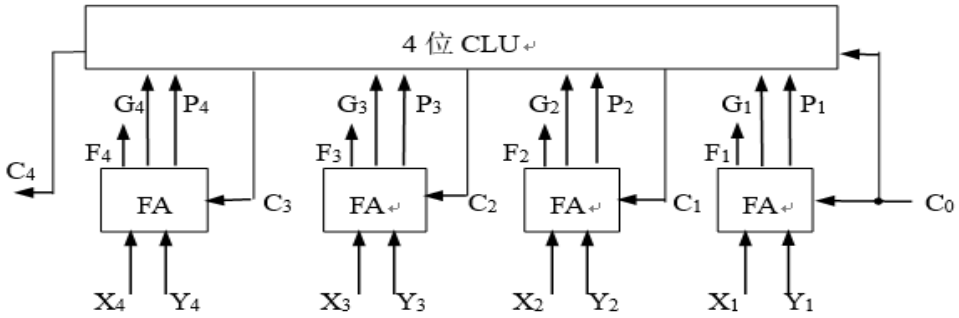
额外的输出信号(CLU的进位生成/传输信号)

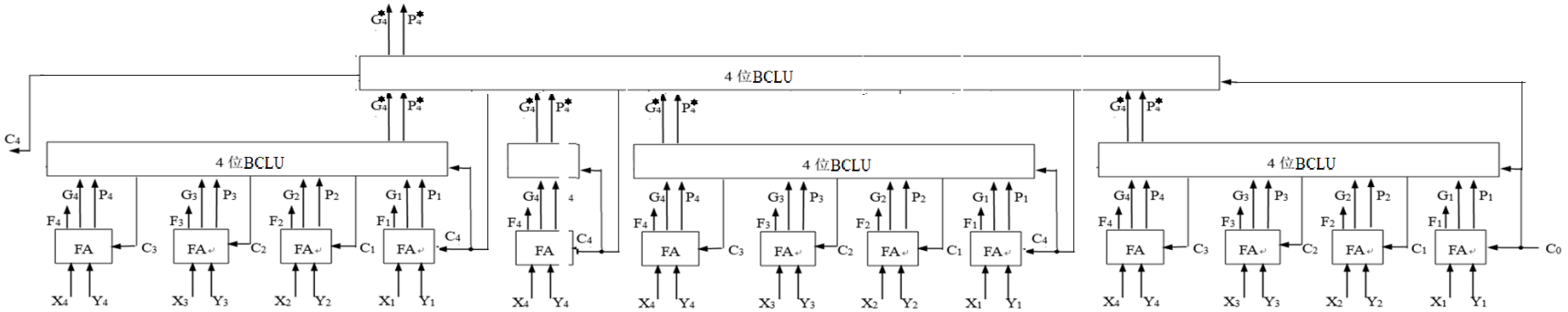
所以$C_4=G_4^*+P_4^*C_0$。把实现上述逻辑的电路称为4位BCLU（Block CLU）部件。

重要内容一：加法器

16位两级先行进位加法器

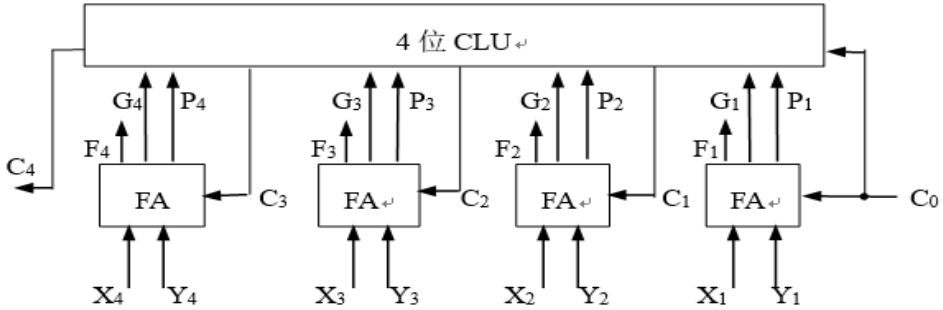


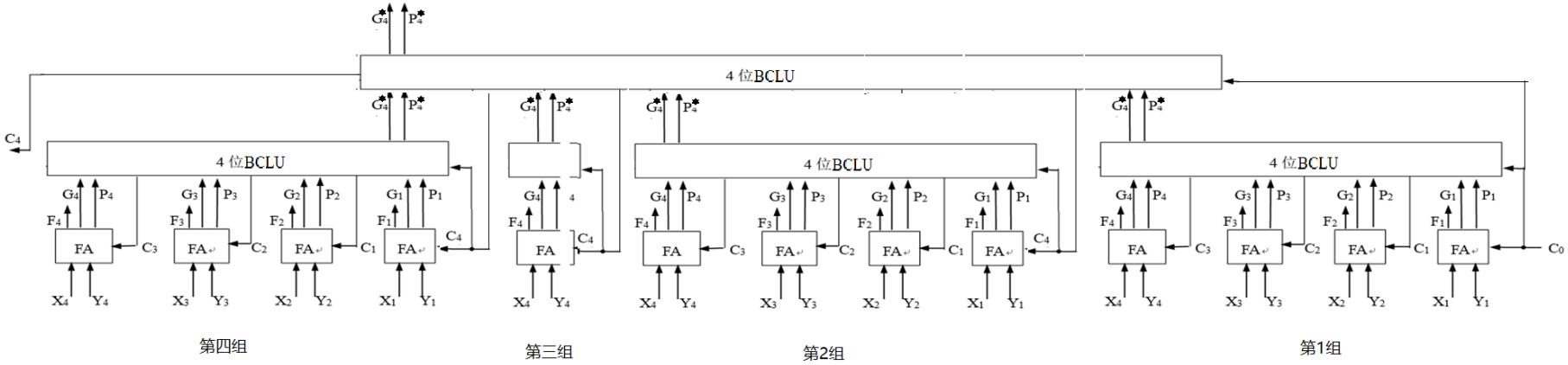




重要内容一：加法器

16位两级先行进位加法器





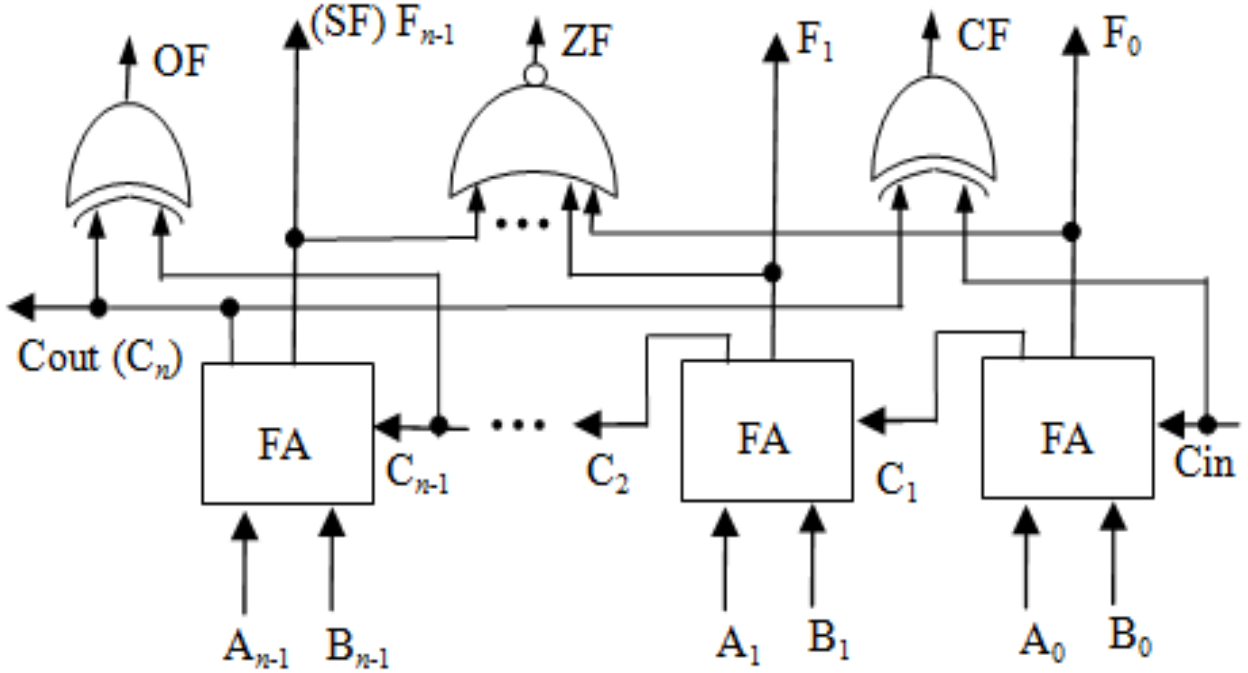
BCLU向下传递进位信号后，下一级CLU计算超前进位需要的延迟

1（gp）+2（clu）+2（clu）+2（clu）+3（xor）=10ty

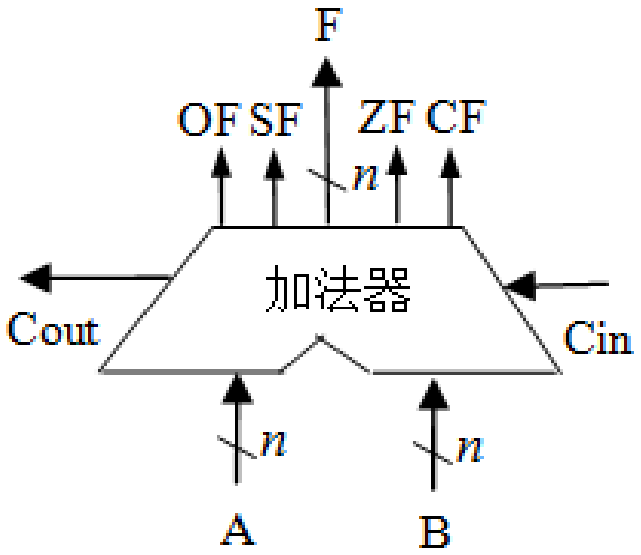
重要内容一：加法器

需求：增加运算结果的标志信息(只针对加法)

- **判断是否溢出**
  - 通盘考虑：n位带符号整数（补码）相加
- **比较大小**
  - 通过（在加法器中）做减法来判断



带标志加法器的逻辑电路



带标志加法器符号

溢出标志OF：

$OF=C_n \oplus C_{n-1}$

符号标志SF：

$SF=F_{n-1}$

零标志ZF=1：

当且仅当F=0；

进位/借位标志CF：

$CF=Cout \oplus Cin$

重要内容二：加减运算

- 补码加减运算公式

  $[A+B]_{补} = [A]_{补} + [B]_{补}\ (\mathrm{mod}\ 2^n)$

  $[A-B]_{补} = [A]_{补} + [-B]_{补}\ (\mathrm{mod}\ 2^n)$

  – 实现减法的关键工作在于：求$[-B]_{补}$

- 利用带标志加法器，可构造整数加/减运算器，进行以下运算：

  无符号整数加、无符号整数减

  带符号整数加、带符号整数减

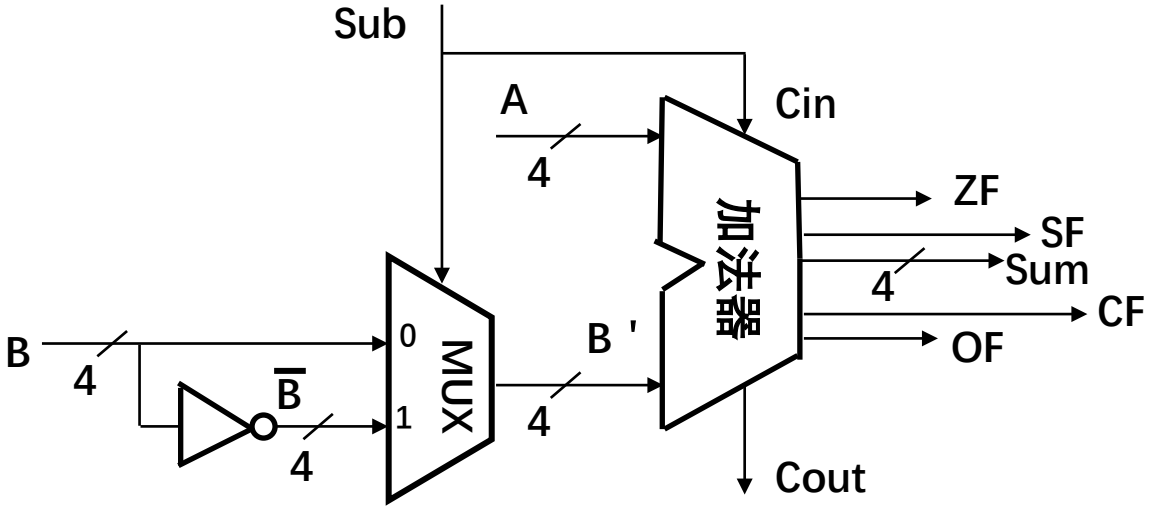
在整数加/减运算部件基础上，加上寄存器、移位器以及控制逻辑，就可实现ALU、乘/除运算以及浮点运算电路

问题：如何求$[-B]_{补}$？

$[-B]_{补}=\overline{[B]_{补}}+1$

当Sub为1时，做减法

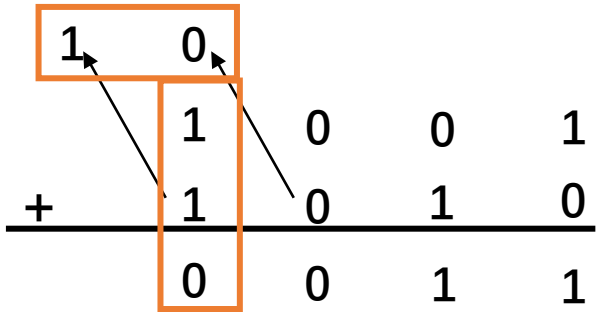
当Sub为0时，做加法



整数加/减运算部件

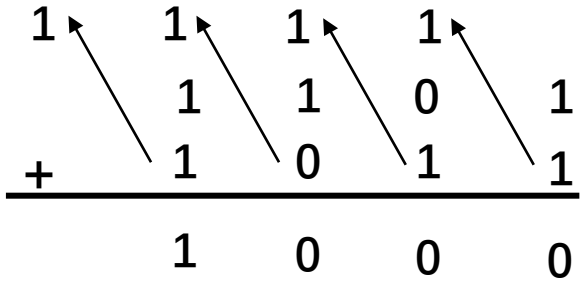
重要内容二：加减运算

-7- 6 = -7 + (-6) = +3 X
9 - 6 = 3 √



OF=1、ZF=0
SF=0、借位CF=0

-3 - 5 = - 3 + (- 5) = - 8 √
13 - 5 = 8 √



OF=0、ZF=0、
SF=1、借位CF=0

带符号溢出：

(1) 最高位和次高位的进位不同,or
(2) 和的符号位和加数的符号位不同

做减法以比较大小，规则：
Unsigned: CF=0时，大于
Signed：OF=SF时，大于

验证：9>6，故CF=0；13>5，故CF=0

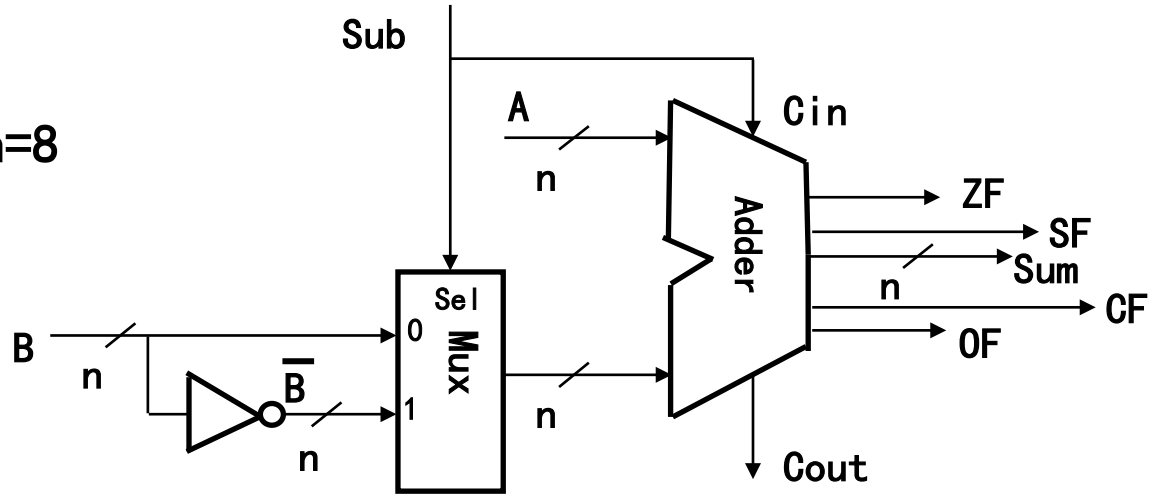
验证：-7<6，故OF≠SF
-3<5，故OF≠SF

重要内容二：加减运算

```
unsigned int x=134;
unsigned int y=246;
int m=x;
int n=y;
unsigned int z1=x-y;
unsigned int z2=x+y;
int k1=m-n;
int k2=m+n;
```

无符号和带符号加减运算都用该部件执行

假定 n=8



x和m的机器数一样：1000 0110，y和n的机器数一样：1111 0110

z1和k1的机器数一样：1001 0000，CF=1，OF=0，SF=1

z1的值为144（=134-246+256，x-y<0），k1的值为-112。

无符号减公式：

$$result=\begin{cases} x-y & (x-y\geq 0) \\ x-y+2^{n} & (x-y<0) \end{cases}$$

带符号减公式：

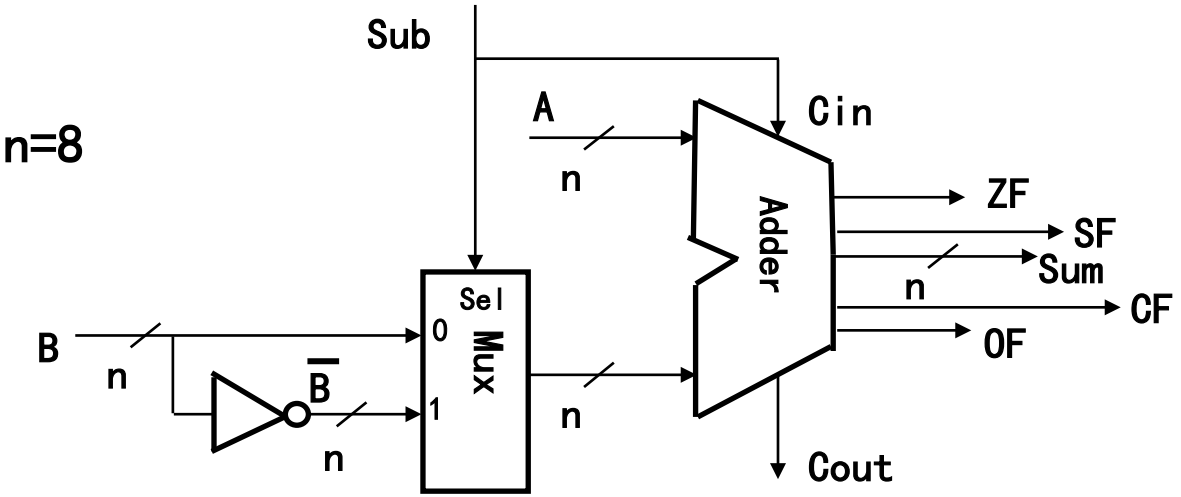
$$result=\begin{cases} x-y-2^{n} & (2^{n-1}\leqslant x-y) & \text{正溢出} \\ x-y & (-2^{n-1}\leqslant x-y<2^{n-1}) & \text{正常} \\ x-y+2^{n} & (x-y<-2^{n-1}) & \text{负溢出} \end{cases}$$

重要内容二：加减运算

```
unsigned int x=134;
unsigned int y=246;
int m=x;
int n=y;
unsigned int z1=x-y;
unsigned int z2=x+y;
int k1=m-n;
int k2=m+n;
```

无符号和带符号加减运算都用该部件执行

假定 n=8



x和m的机器数一样：1000 0110，y和n的机器数一样：1111 0110

z2和k2的机器数一样：1001 0000，CF=1，OF=0，SF=1

z2的值为124（=134-246-256，x+y>256），k1的值为124(=134+246-256,正溢出)。

无符号加公式：

$$result=\begin{cases} x+y & (x+y<2^n) \\ x+y-2^n & (2^n\leqslant x+y<2^{n+1}) \end{cases}$$

带符号加公式：

$$result=\begin{cases} x+y-2^n & (2^{n-1}\leqslant x+y) & \text{正溢出} \\ x+y & (-2^{n-1}\leqslant x+y<2^{n-1}) & \text{正常} \\ x+y+2^n & (x+y<-2^{n-1}) & \text{负溢出} \end{cases}$$

重要内容二：加减运算（移码）

- 用于浮点数阶码运算（符号位和数值部分可以一起处理）
- 运算公式（假定在一个n位ALU中进行加法运算）

$$[E1]_{移}+[E2]_{移}=2^{n-1}+E1+2^{n-1}+E2=2^n+E1+E2=[E1+E2]_{补} \quad (\text{mod } 2^n)$$

$$[E1]_{移}-[E2]_{移}=[E1]_{移}+[-[E2]_{移}]_{补}=2^{n-1}+E1+2^n-[E2]_{移}$$

$$=2^{n-1}+E1+2^n-2^{n-1}-E2$$

$$=2^n+E1-E2=[E1-E2]_{补} \quad (\text{mod } 2^n)$$

结论：移码的和、差等于和、差的补码！（需要转换成移码）

- 运算规则

① 加法：直接将$[E1]_{移}$和$[E2]_{移}$进行模2^n加，然后对结果的符号取反。

② 减法：先将减数$[E2]_{移}$求补（各位取反，末位加1），然后再与被减数 $[E1]_{移}$进行模2^n相加，最后对结果的符号取反。

③ 溢出判断：进行模2^n相加时，如果两个加数的符号相同，并且与和数的符号也相同，则发生溢出。

重要内容三：乘除运算

原码乘、除运算（包括一位、两位）
补码乘法
补码除法（概念性掌握）

重要内容三：乘除运算

被除数： 0000 0111　除数 0011

	A	Q	M=0011
	0000	0111	
←	0000	1110	
+	1101		减（同号）
	1101	1110	
+	0011		恢复(加)商0
	0000	1110	
←	0001	1100	
+	1101		减
	1110	1100	
+	0011		恢复(加)商0
	0001	1100	
←	0011	1000	
+	1101		减
	0000	1000	符同商1
←	0001	0001	
+	1101		减
	1110	0010	
+	0011		恢复(加)商0
	0001	0010	

余:0001/商:0010　　验证：7/3 = 2，余数为1

被除数： 1111 1001　除数 0011

	A	Q	M=0011
	1111	1001	
←	1111	0010	
+	0011		加（异号）
	0010	0010	
+	1101		恢复(减)商0
	1111	0010	
←	1110	0100	
+	0011		加
	0001	0100	
+	1101		恢复(减)商0
	1110	0100	
←	1100	1000	
+	0011		加
	1111	1001	符同商1
←	1111	0010	
+	0011		加
	0010	0010	
+	1101		恢复(减)商0
	1111	0010	

重要内容三：乘除运算

已知 $[X]_{原}$ = 0.1011

$[Y]_{原}$ = 1.1101

用加减交替法计算$[X/Y]_{原}$

解：$[|X|]_{补}$ = 0.1011

$[|Y|]_{补}$ = 0.1101

$[-|Y|]_{补}$ = 1.0011

“加减交替法”的要点：

负、0、加

正、1、减

得到的结果与恢复余数法一样！

余数寄存器R	余数/商寄存器Q	说　明
01011	0000□	开始 $R_0=X$
+10011		$R_1=X-Y$
11110	00000	$R_1<0$，则 $q_4=0$，没有溢出
11100	0000□	$2R_1$（R和Q同时左移，空出一位商）
+01101		$R_2=2R_1+Y$
01001	00001	$R_2>0$，则 $q_3=1$
10010	0001□	$2R_2$（R和Q同时左移，空出一位商）
+10011		$R_3=2R_2-Y$
00101	00011	$R_3>0$，则 $q_2=1$
01010	0011□	$2R_3$（R和Q同时左移，空出一位商）
+10011		$R_3=2R_2-Y$
11101	00110	$R_4<0$，则 $q_1=0$
11010	0110□	$2R_4$（R和Q同时左移，空出一位商）
+01101		$R_5=2R_4+Y$
00111	01101	$R_5>0$，则 $q_0=1$

用被除数（中间余数）减除数试商时，
怎样确定是否“够减”？
中间余数的符号！（正数-正数）

补码除法能否这样来判断呢？

不能，因为符号可能不同！

不恢复余数除法（加减交替法）

恢复余数法可进一步简化为“加减交替法”

根据恢复余数法(设B为除数，R_i为第i次中间余数)，有：

- 若$R_i<0$,则商上“0”，把先做加法恢复余数再移位，改为直接在下一步做加法，即：
  - $R_{i+1}=2(R_i+2^n|B|)-2^n|B|=2R_i + 2^n|B|$ （“负，0，加”）
- 若$R_i>=0$,则商上“1”，不需恢复余数，即：
  - $R_{i+1}=2R_i - 2^n|B|$ （“正，1，减”）

省去了恢复余数的过程

- 注意：最后一次上商为“0”的话，需要“纠余”处理，即把试商时被减掉的除数加回去，恢复真正的余数。
- 不恢复余数法也称为加减交替法

补码除法

补码除法判断是否“够减”的规则

（1）当被除数（或中间余数）与除数同号时，做减法，若新余数的符号与除数符号一致表示够减，否则为不够减；

（2）当被除数（或中间余数）与除数异号时，做加法，若得到的新余数的符号与除数符号一致表示不够减，否则为够减。

上述判断规则归纳如下：

中间余数R的符号	除数Y的符号	同号：新中间余数=R–Y（同号为正商）		异号：新中间余数=R+Y（异号为负商）	
		0	1	0	1
0	0	够减	不够减		
0	1			够减	不够减
1	0			不够减	够减
1	1	不够减	够减		

1011(-5)-1101(-3)=1110(-2)

0100 (4)+1011(-5) =1111(-1)

1011（–5） +0100 （4） =1111（–1）

总结：余数变号不够减，不变号够减

第7、8章

第一讲 指令系统与RISC-V

第二讲 中央处理器设计

重要概念

指令（instruction）、指令集（instruction set）、指令集体系结构（Instruction Set Architecture，ISA）、指令编码、
寻址方式（addressingmode）
通用寄存器（General Purpose Register，GPR）、复杂指令集计算机（Complex Instruction Set Computer，CISC）、精简指令集计算机（Reduced Instruction Set Computer，RISC）
中断（interrupt）
指令周期（instruction cycle）、机器周期（machine cycle）
数据通路（data path）
控制单元（control unit）、执行部件（execute unit）、扩展单元（extension unit）
指令译码器（instruction decoder）
控制信号（control signal）
微程序（microprogram）
指令流水线（instruction pipelining）

重要内容

熟练掌握不同寻址方式以及应用
熟悉指令集基本概念
熟悉不同种类指令操作的执行过程（结合第一张指令执行过程）

熟悉ALU的基本结构与功能（结合第六章运算方法）
熟悉典型指令的数据通路设计
熟悉控制器设计的基本概念
熟悉机器周期与时序分析的基本方法
熟悉多周期与流水的设计原理