

计算机组成原理 (2014级)

计算机组成原理课程组

(刘旭东、肖利民、牛建伟、栾钟治)

Tel : 82316285

Mail: liuxd@buaa.edu.cn

liuxd@act.buaa.edu.cn

第二讲：组合逻辑

一. 逻辑代数基础

1. 逻辑代数基本概念
2. 逻辑代数的公理、定理与规则
3. 逻辑函数的表达式
4. 逻辑函数化简（卡诺图）

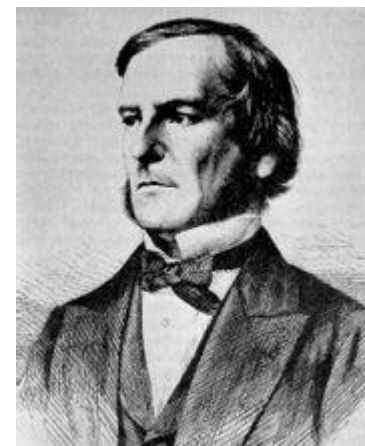
二. 逻辑门电路

1. 晶体管和MOS管
2. 逻辑门电路实现

三. 基本组合逻辑部件设计

逻辑代数的基本概念

- ❖ 所谓“**逻辑**”，指事物间的因果关系。当两个二进制数码表示不同的逻辑状态时，它们之间可以按照指定的某种因果关系进行推理运算，称为**逻辑运算**。
- ❖ 1849年英国数学家乔治·布尔（George Boole）提出了描述客观事物逻辑关系的数学方法——**布尔代数**（Boolean algebra），成功地将形式逻辑问题归结为一种代数运算。
- ❖ 布尔代数后来被广泛用于开关电路和数字逻辑电路的分析和设计，因此也叫做**开关代数**或**逻辑代数**。
 - 布尔代数=逻辑代数
 - 布尔变量=逻辑变量
 - 布尔表达式=逻辑表达式
 - 布尔函数=逻辑函数



乔治·布尔1815.11.2~1864

逻辑代数的基本概念

❖ **逻辑代数**：逻辑代数 L 是一个封闭的代数系统，它由一个逻辑变量集 K ，常量0和1以及“或”、“与”、“非”三种基本运算所构成，记为 $L=\{ K, +, \cdot, -, 0, 1 \}$ 。

➤ **逻辑变量**：在逻辑运算中其值会发生改变的量称为**逻辑变量**，由字母或字母加数字组成。逻辑变量的表示形式如下：

- 原变量： A 、 B 、 C 、 A_1
- 反变量： \bar{A} 、 \bar{B} 、 \bar{C} 、 \bar{A}_1

➤ **逻辑常量**：在逻辑运算中其值不会改变的量称为逻辑常量。

- 逻辑常量是“0”和“1”

➤ **逻辑运算**：

- “或”运算：用“+”或“ \vee ”表示，如 $A+B$ ， $A \vee B$
- “与”运算：用“ \cdot ”或“ \wedge ”表示，如 $A \cdot B$ ， $A \wedge B$
- “非”运算：用“-”或“ \neg ”表示，如 \bar{A} ， $\neg A$

逻辑代数的基本概念

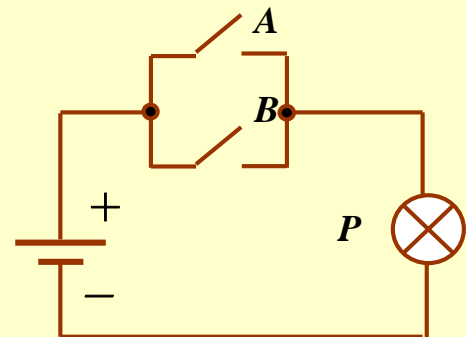
1. **“或”运算**：如果决定某一事件是否发生的多个条件中，只要有一个或一个以上条件成立，事件便可发生，则这种因果关系称之为**“或”逻辑**。

➤ 逻辑代数中，“或”逻辑用“或”运算描述。其运算符号为“+”，有时也用“ \vee ”表示。如： $F = A + B$ 或者 $F = A \vee B$

概念模型

输入条件（ A 、 B ）：闭合 -- “1”，断开 -- “0”

输出结果（灯 P ）：亮 -- “1”，灭 -- “0”



“或”运算表

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

“或”运算的运算法则：

$$0 + 0 = 0 \quad 1 + 0 = 1$$

$$0 + 1 = 1 \quad 1 + 1 = 1$$

实现“或”运算的逻辑电路称为**“或”门**。

逻辑代数的基本概念

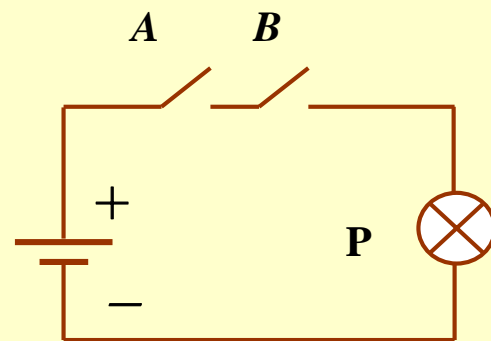
2. “与”运算：如果决定某一事件发生的多个条件必须同时具备，事件才能发生，则这种因果关系称之为“与”逻辑。

➤ 逻辑代数中，“与”逻辑用“与”运算描述。其运算符号为“ \cdot ”，有时也用“ \wedge ”表示。如： $F = A \cdot B$ 或者 $F = A \wedge B$

概念模型

输入条件（ A 、 B ）：闭合 -- “1”，断开 -- “0”

输出结果（灯 P ）：亮 -- “1”，灭 -- “0”



“与”运算表

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

“与”运算的运算法则：

$$0 \cdot 0 = 0 \quad 1 \cdot 0 = 0$$

$$0 \cdot 1 = 0 \quad 1 \cdot 1 = 1$$

实现“与”运算的逻辑电路称为“与”门。

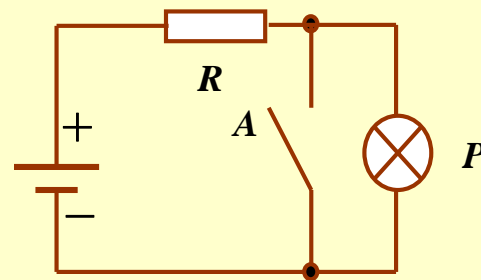
逻辑代数的基本概念

3. “非”运算：如果某一事件的发生取决于条件的否定，即事件与事件发生的条件构成矛盾，这种因果关系称为“非”逻辑。

➤ 在逻辑代数中，“非”逻辑用“非”运算描述。其运算符号为“ \neg ”，有时也用“ \neg ”表示。如： $F = \neg A$ 或者 $F = \neg A$

概念模型

输入条件（开关A）：闭合 -- “1”，断开 -- “0”
输出结果（灯P）：亮 -- “1”，灭 -- “0”



“非”运算的运算法则：

$$\overline{0} = 1 \quad \overline{1} = 0$$

实现“与”运算的逻辑电路称为“与”门。

“非”运算表

A	F
0	1
1	0

逻辑代数的基本概念

4. “异或”运算

$$F=A \oplus B$$

$$F = A \oplus B$$

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

“异或”运算表

$A \ B$	F
0 0	0
0 1	1
1 0	1
1 1	0

5. “同或”运算

$$F = A \odot B$$

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

“同或”运算表

$A \ B$	F
0 0	1
0 1	0
1 0	0
1 1	1

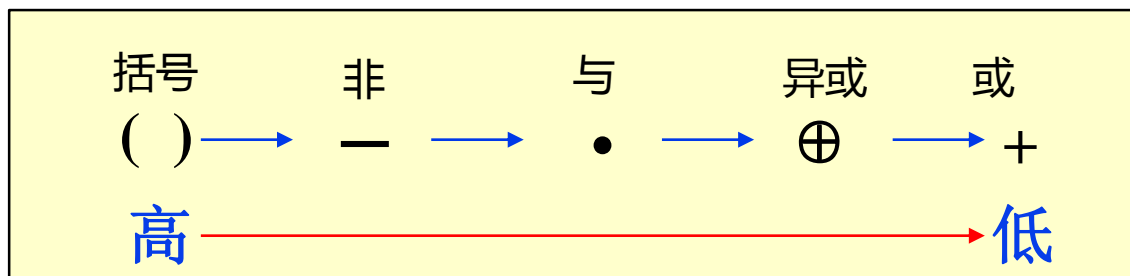
- 异或、同或逻辑只有两个输入；与（与非）、或（或非）逻辑可以有两个以上的输入；非逻辑只有一个输入。
- 异或逻辑与同或逻辑是互非关系：

$$A \oplus B = A \odot B ; \quad A \odot B = A \oplus B \quad \overline{\overline{A}B} + A\overline{B} = \overline{\overline{A}B} + A\overline{B}$$

逻辑代数的基本概念

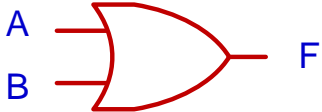
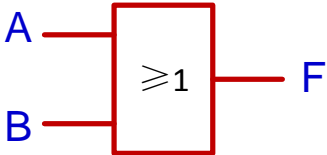
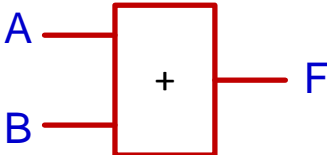

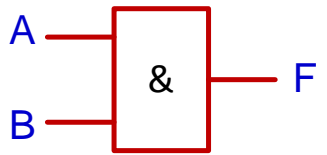
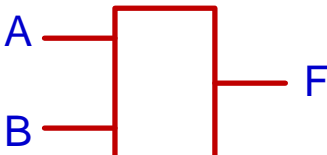
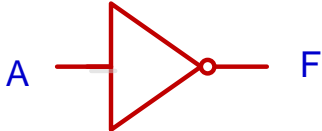
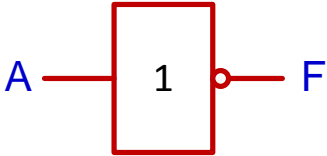
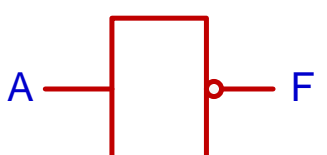
❖ 逻辑运算的顺序

- 在一个表达式中，如果既有“与”运算又有“或”运算，则按先“与”后“或”的规则进行运算，可省去括号，如 $(A \cdot B) + (C \cdot D)$ 可写为 $AB + CD$ 。
- 注意： $(A+B) \cdot (C+D)$ 不能省略括号，即不能写成 $A+B \cdot C+D$
- 运算优先法则：



逻辑代数的基本概念

❖ 逻辑电路的符号表示

逻辑门	IEEE标准符号	国标符号	部标符号	逻辑表达式
或				$F = A + B$ $F = A \vee B$
与				$F = A \cdot B$ $F = A \wedge B$
非				$F = \overline{A}$ $F = \neg A$

第二讲：组合逻辑

一. 逻辑代数基础

1. 逻辑代数基本概念
2. 逻辑代数的公理、定理与规则
3. 逻辑函数的表达式
4. 逻辑函数化简（卡诺图）

二. 逻辑门电路

1. 晶体管和MOS管
2. 逻辑门电路实现

三. 基本组合逻辑部件设计

◆ 逻辑代数的基本公理

① **交换律** : $A + B = B + A$ $A \cdot B = B \cdot A$

② **结合律** : $(A + B) + C = A + (B + C)$
 $(A \cdot B) \cdot C = A \cdot (B \cdot C)$

③ **分配律** : $A \cdot (B + C) = A \cdot B + A \cdot C$
 $A + B \cdot C = (A + B) \cdot (A + C)$

④ **0 - 1律** : $A + 0 = A$ $A \cdot 1 = A$
 $A + 1 = 1$ $A \cdot 0 = 0$

⑤ **互补律** : $A + \bar{A} = 1$ $A \cdot \bar{A} = 0$

公理是一个代数系统的基本出发点，无需加以证明。

逻辑代数基本定理

1. 定理1

$$0 + 0 = 0 \quad 1 + 0 = 1 \quad 0 \cdot 0 = 0 \quad 1 \cdot 0 = 0$$

$$0 + 1 = 1 \quad 1 + 1 = 1 \quad 0 \cdot 1 = 0 \quad 1 \cdot 1 = 1$$

证明：在公理4（0-1律）中，A表示集合K中的任意元素，因而可以是0或1。用0和1代入公理4中的A，即可得到上述关系。

如果以1和0代替公理5中的A，则可得到如下推论：

$$\bar{1} = 0$$

$$\bar{0} = 1$$

逻辑代数基本定理

2. 定理2：重叠律

$$A + A = A \quad ; \quad A \cdot A = A$$

证明：	$A + A = (A + A) \cdot 1$	公理4
	$= (A + A) \cdot (A + \overline{A})$	公理5
	$= A + A \cdot \overline{A}$	公理3
	$= A + 0$	公理5
	$= A$	公理4

仅对其中一个定理进行证明，另一个留给同学自己证明

逻辑代数基本定理

3. 定理3：吸收律

$$A + A \cdot B = A \quad ; \quad A \cdot (A + B) = A$$

证明：	$A + A \cdot B$	$=$	$A \cdot 1 + A \cdot B$	公理4
		$=$	$A \cdot (1 + B)$	公理3
		$=$	$A \cdot (B + 1)$	公理1
		$=$	$A \cdot 1$	公理4
		$=$	A	公理4

逻辑代数基本定理

4. 定理4：吸收律

$$\begin{aligned} A + \bar{A} \cdot B &= A + B \\ A \cdot (\bar{A} + B) &= A \cdot B \end{aligned}$$

$$\begin{aligned} \text{证明: } A + \bar{A} \cdot B &= (A + \bar{A}) \cdot (A + B) && \text{公理3} \\ &= 1 \cdot (A + B) && \text{公理5} \\ &= A + B && \text{公理4} \end{aligned}$$

逻辑代数基本定理

5. 定理5：还原律

$$\overline{\overline{A}} = A$$

证明 令 $\overline{A} = X$

因而 $\overline{A} \cdot X = 0$ $\overline{A} + X = 1$ 公理5

但是 $\overline{A} \cdot A = 0$ $\overline{A} + A = 1$ 公理5

由于X和A都满足公理5。因此，根据公理5的唯一性，得到 $A = X$ 。

逻辑代数基本定理

6. 定理6：反演律（摩根定律）

$$\overline{A + B} = \overline{A} \cdot \overline{B} \quad \overline{A \cdot B} = \overline{A} + \overline{B}$$

证明： 由于 $(\overline{A} \cdot \overline{B}) + (A + B) = (\overline{A} \cdot \overline{B} + A) + B$ 公理2
 $= (\overline{B} + A) + B$ 定理4
 $= A + (B + \overline{B})$ 公理1,2
 $= A + 1$ 公理5
 $= 1$ 公理4
 而且 $(\overline{A} \cdot \overline{B})(A + B) = \overline{A} \cdot \overline{B} \cdot A + \overline{A} \cdot \overline{B} \cdot B$ 公理3
 $= 0 + 0$ 公理1,5
 $= 0$ 定理1

所以，根据公理5的唯一性可得 $\overline{A + B} = \overline{A} \cdot \overline{B}$

逻辑代数基本定理

7. 定理7 :

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

$$(A + B) \cdot (A + \overline{B}) = A$$

证明:
$$\begin{aligned} A \cdot B + A \cdot \overline{B} &= A \cdot (B + \overline{B}) \\ &= A \cdot 1 \\ &= A \end{aligned}$$

公理3

公理5

公理4

逻辑代数基本定理

8. 定理8：包含律

$$A \cdot B + \bar{A} \cdot C + B \cdot C = A \cdot B + \bar{A} \cdot C$$
$$(A + B) \cdot (\bar{A} + C) \cdot (B + C) = (A + B) \cdot (\bar{A} + C)$$

证明：

$$\begin{aligned} A \cdot B + \bar{A} \cdot C + B \cdot C &= A \cdot B + \bar{A} \cdot C + (A + \bar{A}) \cdot B \cdot C && \text{公理5} \\ &= A \cdot B + \bar{A} \cdot C + A \cdot B \cdot C + \bar{A} \cdot B \cdot C && \text{公理3} \\ &= A \cdot B + A \cdot B \cdot C + \bar{A} \cdot C + \bar{A} \cdot B \cdot C && \text{公理1} \\ &= A \cdot B \cdot (1 + C) + \bar{A} \cdot C \cdot (1 + B) && \text{公理3} \\ &= A \cdot B \cdot (C + 1) + \bar{A} \cdot C \cdot (B + 1) && \text{公理1} \\ &= A \cdot B + \bar{A} \cdot C && \text{公理4} \end{aligned}$$

推论：

$$AB + \bar{A}C + BCDEF... = AB + \bar{A}C$$

逻辑代数的规则

1. 代入规则：在任何一个包含某个相同变量的逻辑等式中，用另外一个函数式代入式中所有这个变量的位置，等式仍然成立。

➤ 用途：扩大基本公式和常用公式的使用范围

例：已知 $A + \bar{A} = 1$

则： $ABC + \overline{ABC} = 1$

例：已知 $\overline{A + B} = \bar{A} \cdot \bar{B}$

则： $\overline{AB + CD} = \overline{AB} \cdot \overline{CD}$

逻辑代数的规则

2. 反演规则：将原函数F中的全部“·”换成“+”，“+”换成“·”，“0”换成“1”，“1”换成“0”，原变量换成反变量，反变量换成原变量，所得到的新函数就是原函数的反函数，记作 \bar{F} 。

➤ 用途：直接求已知逻辑函数的反函数，可用于公式的化简

【例2】 试化简函数： $F = (A + B)(\bar{A} + C)(B + C + D)$

$$\begin{aligned}\text{解: } \bar{F} &= \overline{AB} + \overline{AC} + \overline{BCD} \\ &= \overline{AB} + \overline{AC} + (A + \bar{A})\overline{BCD} \\ &= \overline{AB} + \overline{ABCD} + \overline{AC} + \overline{ABCD} \\ &= \overline{AB} + \overline{AC}\end{aligned}$$

$$\text{则: } F = \bar{\bar{F}} = (A + B)(\bar{A} + C)$$

逻辑代数的规则

3. 对偶规则：将原函数F中的全部“ \cdot ”换成“ $+$ ”，“ $+$ ”换成“ \cdot ”，“0”换成“1”，“1”换成“0”，所得的新函数就是原函数的**对偶式**，记作F'或F*。

➤ 用途：已知某公式成立，则可以得到其对偶公式仍成立。若两个逻辑函数表达式F和G相等，则其对偶式F'和G'也相等。

例如：函数 $F_1 = AB + \overline{(C + \overline{D})}B + \overline{BC} + 0$

则对偶式是： $F_1' = (A + B) \cdot \overline{\overline{CD}} + B \cdot \overline{B} + \overline{C} \cdot 1$

又如： $F_2 = A + \overline{B + \overline{C} \cdot \overline{D} + \overline{E}}$

则对偶式是： $F_2' = \overline{A \cdot B \cdot (\overline{C} + \overline{D \cdot E})}$

第二讲：组合逻辑

一. 逻辑代数基础

1. 逻辑代数基本概念
2. 逻辑代数的公理、定理与规则
3. 逻辑函数的表达式
4. 逻辑函数化简（卡诺图）

二. 逻辑门电路

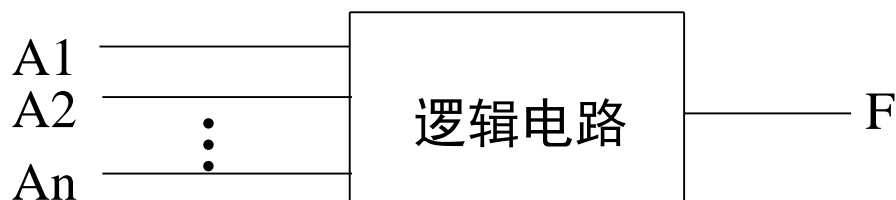
1. 晶体管和MOS管
2. 逻辑门电路实现

三. 基本组合逻辑部件设计

逻辑函数

❖ 逻辑函数

- 设某一逻辑电路的输入逻辑变量为 A_1, A_2, \dots, A_n ，输出逻辑变量为 F ，如下图所示。则称 F 为 A_1, A_2, \dots, A_n 的逻辑函数，记为： $F = f(A_1, A_2, \dots, A_n)$



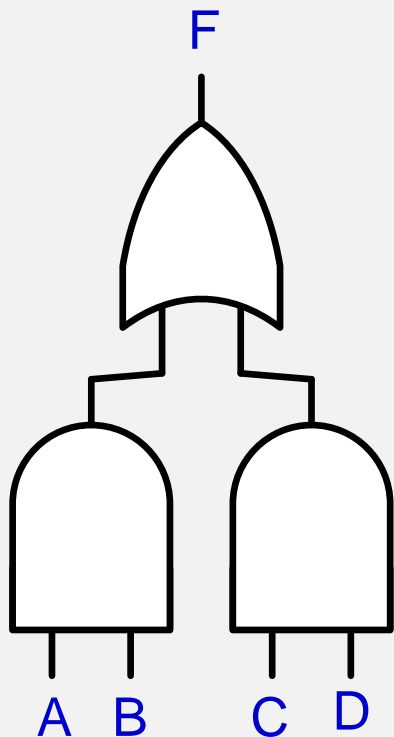
- 逻辑电路输出函数的取值是由逻辑变量的取值和电路本身的结构决定的。
- 任何一个逻辑电路的功能都可由相应的逻辑函数完全描述，因此，能够借助抽象的代数表达式对电路加以分析研究。

逻辑函数的常用表达式

❖ 常用表达式包括：与或式、或与式、与或非式

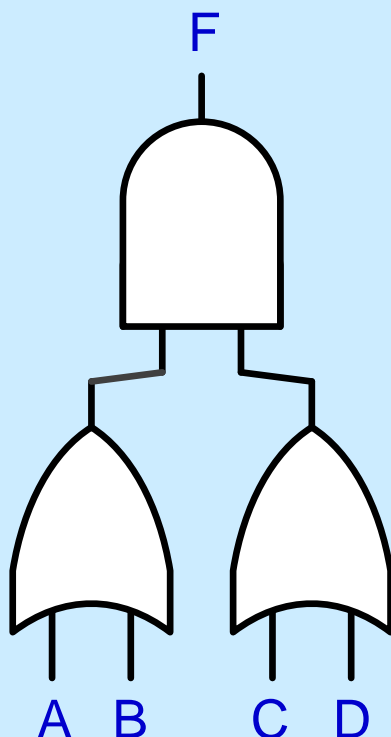
与或式

$$F = AB + CD$$



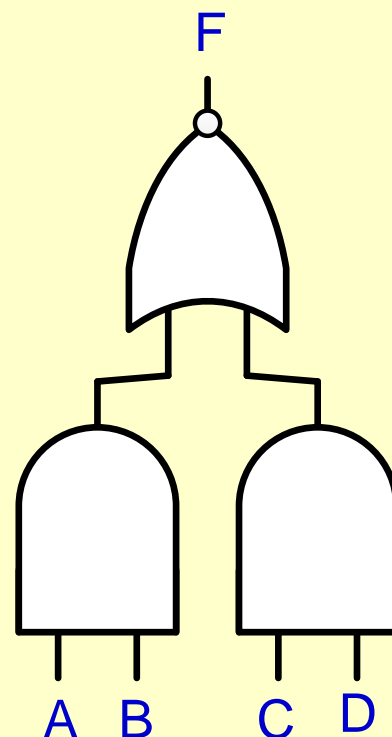
或与式

$$F = (A + B)(C + D)$$



与或非式

$$F = \overline{AB + CD}$$



逻辑函数的标准表达式

- ❖ 逻辑函数的表达形式是不唯一的，在数字电路手工设计技术中，为便于真值表表述、卡诺图表述和逻辑化简等，引入逻辑函数的标准表达式
- ❖ 逻辑函数的标准表达式建立在最小项和最大项概念基础上
- ❖ 标准表达式：
 - **最小项表达式**：全部由最小项构成的 **与或式**（积之和式）
 - **最大项表达式**：全部由最大项构成的 **或与式**（和之积式）

$$F(A, B, C) = \overline{A}\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\overline{C} + ABC \quad \text{最小项表达式}$$

$$F(A, B, C) = (A + B + C) \cdot (A + \overline{B} + \overline{C}) \cdot (\overline{A} + \overline{B} + \overline{C}) \quad \text{最大项表达式}$$

最小项

❖ 最小项定义：

- 设有n个变量，它们所组成的具有n个变量的“与”项（**乘积项**）中，每个变量以原变量或反变量的形式出现且仅出现一次，则这个乘积项称为最小项。
- n个变量有 2^n 个最小项
- 如：3个变量的8个最小项

$$\overline{A}\overline{B}\overline{C}, \overline{A}\overline{B}C, \overline{A}B\overline{C}, \overline{A}BC, A\overline{B}\overline{C}, A\overline{B}C, AB\overline{C}, ABC$$

- 为书写方便，把最小项记做 m_i 。
- 下标i的取值规则：按照变量顺序将最小项中的原变量用1表示、反变量用0表示，得到一个二进制数，与其对应的十进制数即该最小项的编号 i。

最小项的性质

❖ 相邻最小项

➤ 除一个变量互为相反外，其余变量分别相同的两个最小项。

❖ 最小项的性质

- ① 对于任何一个最小项，只有对应的一组变量取值，使其值为1，其余情况下均为0；
- ② 全体最小项之和为1；
- ③ 任意两个最小项的乘积为0；
- ④ 具有相邻性的两个最小项之和可以合并为一个乘积项，消去一个以原变量和反变量形式出现的变量，保留由没有变化的变量构成的乘积项。

$$\text{例: } \overline{A}\overline{B}C + \overline{A}B\overline{C} = \overline{A}\overline{B}$$

最小项编号

最小项	ABC 的取值	编号
$\overline{A}\overline{B}\overline{C}$	000	m_0
$\overline{A}\overline{B}C$	001	m_1
$\overline{A}B\overline{C}$	010	m_2
$\overline{A}BC$	011	m_3
$A\overline{B}\overline{C}$	100	m_4
$A\overline{B}C$	101	m_5
$AB\overline{C}$	110	m_6
ABC	111	m_7

最大项

❖ 最大项

- 设有n个变量，它们所组成的具有n个变量的“或”项（和项）中，每个变量以原变量或反变量的形式出现且仅出现一次，则这个和项称为最大项。
- n个变量有 2^n 个最大项
- 3变量（A,B,C）有8个最大项

$$\begin{array}{cccc}\bar{A} + \bar{B} + \bar{C}, & \bar{A} + \bar{B} + C, & \bar{A} + B + \bar{C}, & \bar{A} + B + C \\ A + \bar{B} + \bar{C}, & A + \bar{B} + C, & A + B + \bar{C}, & A + B + C\end{array}$$

- 为书写方便，把最大项记做 M_i 。
- 下标 i 的取值规则：按照变量顺序将最大项中的原变量用0表示、反变量用1表示，得到一个二进制数，与其对应的十进制数即该最大项的编号i。

最大项的性质

➤ 相邻性

- 若2个最大项中只有1个变量分别以原变量和反变量的形式出现，其余的变量分别相同，则称这2个变量具有相邻性。

➤ 最大项的性质

- ① 对于任何一个最大项，只有对应的一组变量取值，使其值为0，其余情况下均为1；
- ② 全体最大项之积为0；
- ③ 任意两个最大项之和为1。
- ④ 具有相邻性的两个最大项之积可以合并为一个和项，消去一个以原变量和反变量形式出现的变量，保留由没有变化的变量构成的和项。

$$\text{例: } (\bar{A} + B + \bar{C})(A + B + \bar{C}) = [(\bar{A} + (B + \bar{C}))][(A + (B + \bar{C}))] = B + \bar{C}$$

最大项编号

最大项	ABC 的取值	编号
$A+B+C$	000	M_0
$A+B+\bar{C}$	001	M_1
$A+\bar{B}+C$	010	M_2
$A+\bar{B}+\bar{C}$	011	M_3
$\bar{A}+B+C$	100	M_4
$\bar{A}+B+\bar{C}$	101	M_5
$\bar{A}+\bar{B}+C$	110	M_6
$\bar{A}+\bar{B}+\bar{C}$	111	M_7

最小项表达式

❖ 全部由最小项构成的与或式，也称**标准与或式**，可由**最小项推导法**直接从真值表中导出。

❖ 例：三人表决器设计（表决原则：少数服从多数）

➤ A, B, C表示输入，1表示赞成，0表示反对

➤ F表示输出，1表示通过，0表示不通过

$$F = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$$

$$F(A, B, C) = m_3 + m_5 + m_6 + m_7$$

$$F(A, B, C) = \sum m(3,5,6,7)$$

最小项
表达式

真值表

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

❖ **最小项推导法**：从真值表推出逻辑函数表达式的一种方法。

使输出为1的输入组合写成乘积项的形式，其中取值为1的输入用原变量表示，取值为0的输入用反变量表示，然后把这些乘积项加起来。

最大项表达式

❖ 全部由最大项构成的或与式，也称**标准或与式**，可由**最大项推导法**直接从真值表中导出。

【例】：三人表决器设计的输出表达式

$$F = (A + B + C)(A + B + \bar{C})(A + \bar{B} + C)(\bar{A} + B + C)$$

$$F(A, B, C) = M_0 \cdot M_1 \cdot M_2 \cdot M_4$$

$$F(A, B, C) = \prod M(0, 1, 2, 4)$$

真值表

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

❖ **最大项推导法**：把使输出为**0**的输入组合写成和项的形式，其中取值为**0**的输入用原变量表示，取值为**1**的输入用反变量表示，然后把这些和项乘起来。

逻辑函数化简

❖ 设计优化

- **面积优化**——使设计的电路或系统占用的逻辑资源尽量少
- **时间优化**——使设计的电路或系统的输入信号到达输出的路程尽量短

- ❖ 逻辑函数的简化是实现**面积优化**的一种方式。
- ❖ 过去逻辑函数的简化是非常重要而又繁琐的工作；
- ❖ 在现代数字电路或系统的设计中，设计优化主要由**EDA**工具自动完成，一般无须设计者介入。

逻辑函数化简

- ❖ 同一个逻辑函数可以写成不同的逻辑式；
- ❖ 逻辑表达式越简单，实现它的电路越简单，电路工作越稳定可靠；
- ❖ 因此需要通过化简找出最简逻辑式。

【例】化简 $F = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$

➤ 若不化简，需要3个非门、4个3输入与门和1个4输入或门。

解 $F = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$

$= AB(\bar{C} + C) + AC(\bar{B} + B) + BC(\bar{A} + A)$

$= AB + AC + BC$

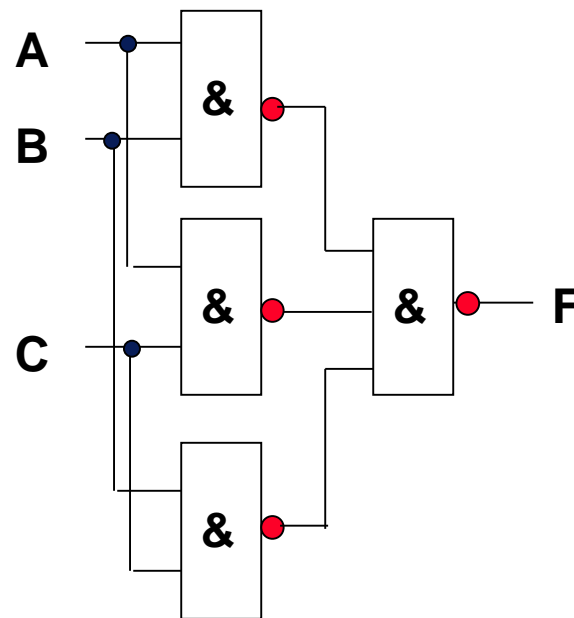
$= \overline{\overline{AB + AC + BC}}$

$= \overline{AB} \cdot \overline{AC} \cdot \overline{BC}$

增加两
项ABC

还原律

反演律



逻辑函数化简

❖ 最简或与表达式

- 1、或项个数最少（或门用的最少）；
- 2、在满足1的条件下，或项中变量数最少（或门的输入端最少）。

➤ 化简方法

- 1、利用对偶规则，将“或与”表达式转换为“与或”表达式。
- 2、实际化简“与或”表达式。
- 3、利用对偶规则将最简“与或”表达式转为最简“或与”表达式。

【例】化简 $F=(A+B)(\bar{A}+C)(B+C)(A+C)$

对偶规则 $F'=AB+\bar{A}C+BC+AC$

$$=AB+\bar{A}C+AC$$

$$=AB+C$$

则： $F=(A+B) \cdot C$

由包含律
 $AB+\bar{A}C+BC=AB+\bar{A}C$

由常用公式1
 $\bar{A}C+AC=C$

逻辑函数化简

❖ 逻辑函数的公式简化常用的方法（以与或表达式的化简为例）有：合并乘积项法、吸收项法、配项法、消除冗余项法

1、合并乘积项法——利用互补律消去1个变量

化简 $F = A(BC + \overline{B}\overline{C}) + AB\overline{C} + A\overline{B}C$

解： $F = ABC + A\overline{B}\overline{C} + AB\overline{C} + A\overline{B}C$

利用分配律展开

$$= (ABC + A\overline{B}C) + (A\overline{B}\overline{C} + AB\overline{C})$$

合并

$$= AC(B + \overline{B}) + A\overline{C}(\overline{B} + B)$$

$$= AC + A\overline{C}$$

互补律

$$= A(C + \overline{C}) = A$$

互补律

逻辑函数化简

2、吸收项法——利用吸收律和包含律减少“与”项

化简 $F = A\bar{B} + \bar{A}B + ABCD + \bar{A}\bar{B}CD$

解:
$$\begin{aligned} F &= (A\bar{B} + \bar{A}B) + (AB + \bar{A}\bar{B})CD \\ &= (A\bar{B} + \bar{A}B) + \overline{A\bar{B} + \bar{A}B} \cdot CD \\ &= A\bar{B} + \bar{A}B + CD \end{aligned}$$

由吸收律3
 $A + \bar{A}B = A + B$

3、配项法——利用互补律，配在乘积项上

化简 $F = AB + \bar{A}\bar{B}C + BC$

解:
$$\begin{aligned} F &= AB + \bar{A}\bar{B}C + BC(A + \bar{A}) \\ &= AB + \bar{A}\bar{B}C + ABC + \bar{A}BC \\ &= (AB + ABC) + (\bar{A}\bar{B}C + \bar{A}BC) \\ &= AB(1 + C) + \bar{A}C(B + \bar{B}) \\ &= AB + \bar{A}C \end{aligned}$$

第二讲：组合逻辑

一. 逻辑代数基础

1. 逻辑代数基本概念
2. 逻辑代数的公理、定理与规则
3. 逻辑函数的表达式
4. 逻辑函数化简（卡诺图）

二. 逻辑门电路

1. 晶体管和**MOS**管
2. 逻辑门电路实现

三. 基本组合逻辑部件设计

卡诺图化简

❖ 基本原理：吸收率 $AB + A\bar{B} = A$

用公式化简

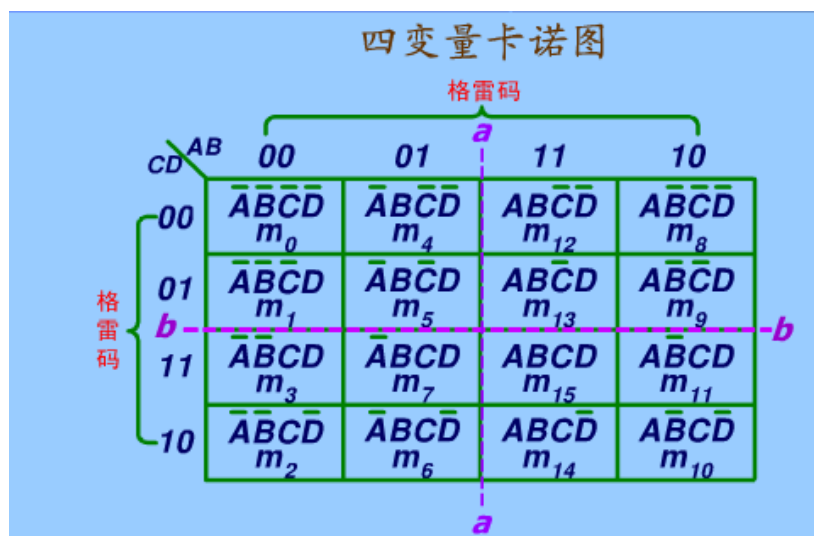
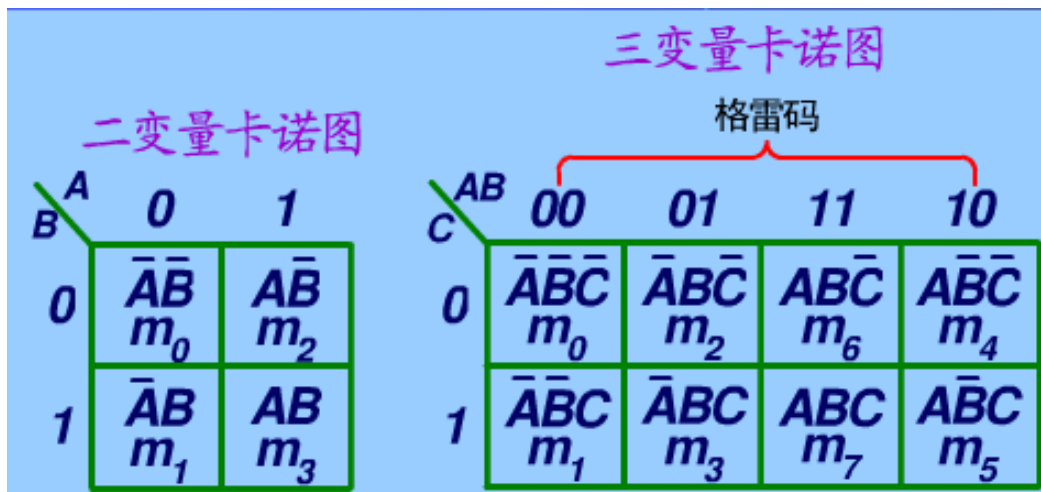
$$\begin{aligned} F &= \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}BC + \bar{A}B\bar{C} + ABC \\ &= \bar{A}\bar{B} + \bar{A}B + BC \\ &= \bar{A} + BC \end{aligned}$$

公式化简：两个逻辑相邻项可以合并为一项，有适合逻辑函数的相邻关系不直观

$$\begin{aligned} F &= \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}BC + \bar{A}B\bar{C} + ABC \\ &= m_0 + m_1 + m_3 + m_7 \\ &= \Sigma(0,1,3,7) \end{aligned}$$

卡诺图化简

❖ 卡诺图结构



卡诺图化简

❖ 逻辑函数卡诺图表示

$$\begin{aligned} F(ABC) &= \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}C + ABC \\ &= \Sigma(0,2,3,5,7) \end{aligned}$$

AB		00	01	11	10
C	0	1	1		
	1		1	1	1

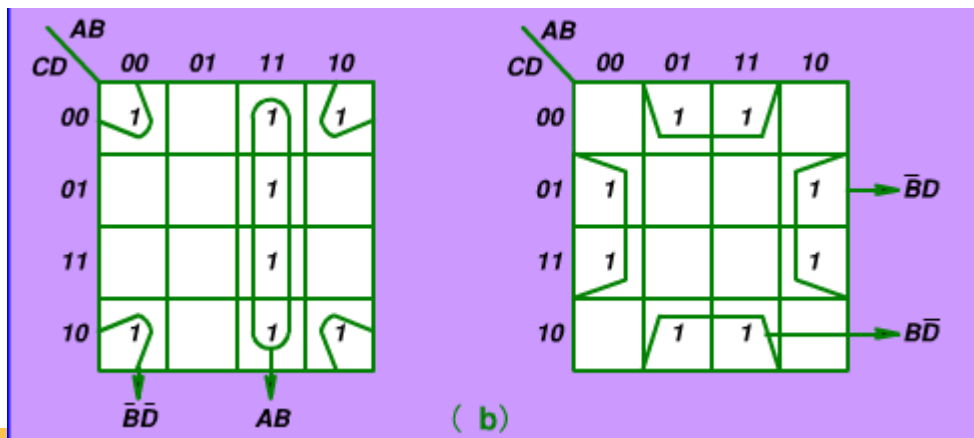
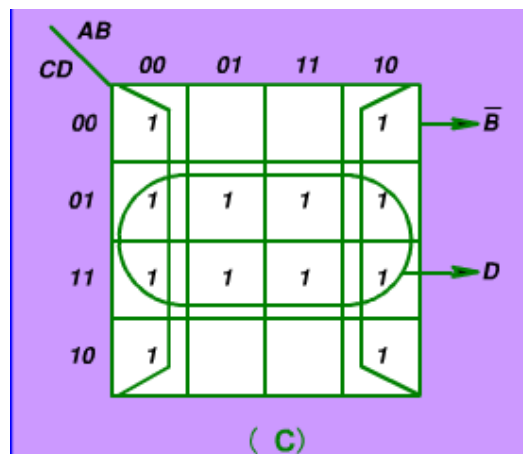
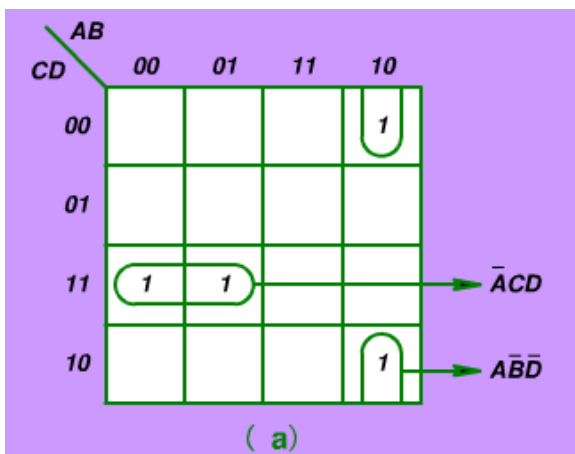
AB		00	01	11	10
CD	00			1	
	01				
	11	1	1	1	
	10		1	1	

$$\begin{aligned} F(ABCD) &= \bar{A}BC + BCD + \bar{A}CD + AB\bar{D} \\ &= \bar{A}BCD + \bar{A}BC\bar{D} + ABCD + \bar{A}\bar{B}CD + ABC\bar{D} + AB\bar{C}\bar{D} \\ &= \Sigma(3,6,7,12,14,15) \end{aligned}$$

卡诺图化简

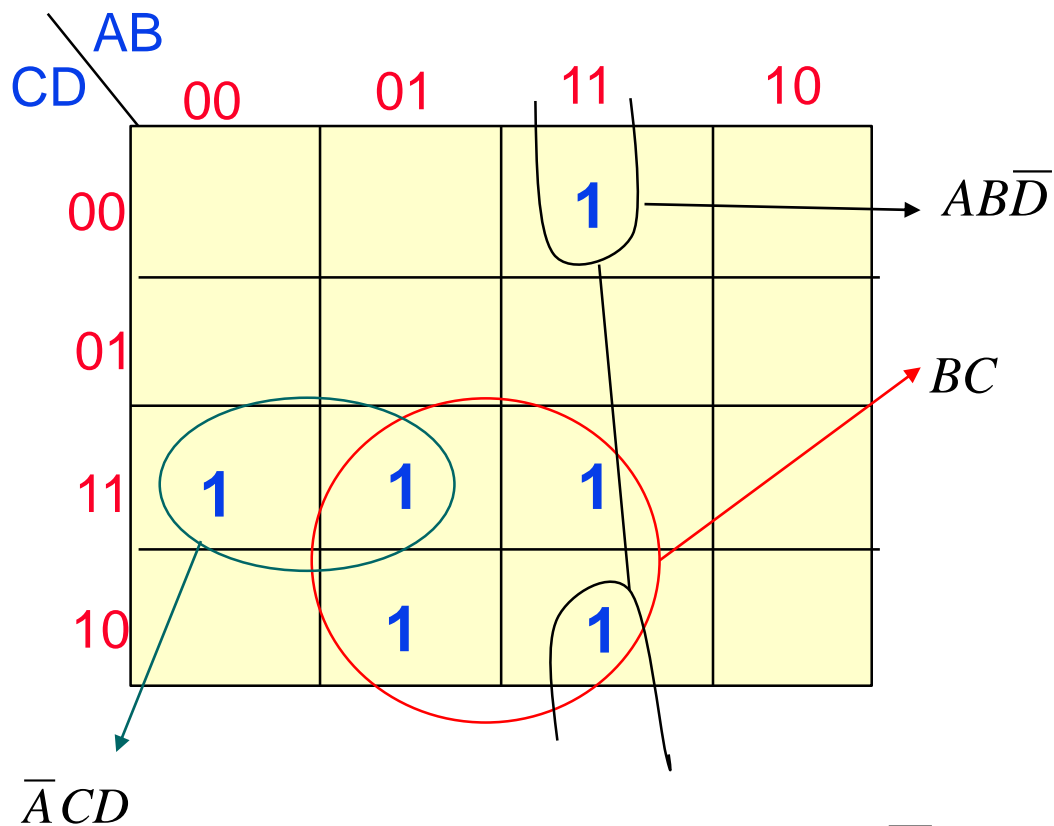
❖ 卡诺图化简基本规则

- 2个相邻项合并为一项，消去1个取值不同的变量
- 4个相邻项合并为一项，消去2个取值不同的变量
- 8个相邻项合并为一项，消去3个取值不同的变量



卡诺图化简

$$\begin{aligned}F(ABCD) &= \bar{A}BC + BCD + \bar{A}CD + AB\bar{D} \\&= \bar{A}BCD + \bar{A}BC\bar{D} + ABCD + \bar{A}\bar{B}CD + ABC\bar{D} + AB\bar{C}\bar{D} \\&= \Sigma(3,6,7,12,14,15)\end{aligned}$$



$$F(ABCD) = AB\bar{D} + \bar{A}CD + BC$$

第二讲：组合逻辑

一. 逻辑代数基础

1. 逻辑代数基本概念
2. 逻辑代数的公理、定理与规则
3. 逻辑函数的表达式
4. 逻辑函数化简（卡诺图）

二. 逻辑门电路

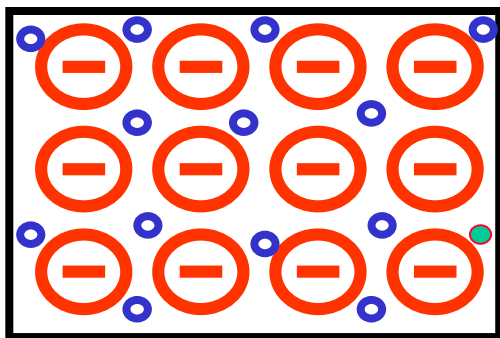
1. 晶体管和**MOS**管
2. 逻辑门电路实现

三. 基本组合逻辑部件设计

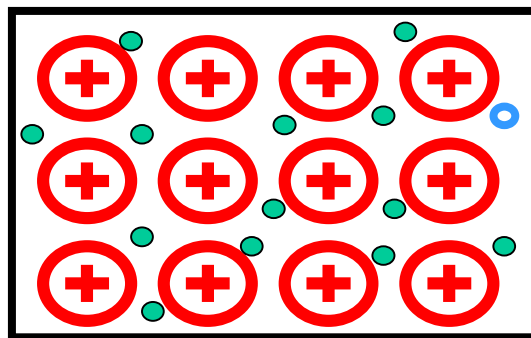
半导体基础知识简介

❖ **半导体**：导电能力介于导体和绝缘体之间的物体称为半导体。如：硅（Si）、锗（Ge）、硒（Se）以及大多数金属氧化物和硫化物。

- **P（Positive）型半导体**：在纯净的硅或锗晶体中掺入微量的三价元素（如硼或铟），导电以**空穴**为主；
- **N（Negative）型半导体**：在纯净的硅或锗晶体中掺入微量五价元素（如磷或砷），导电以**自由电子**为主。
- 半导体中占多数的载流子称为**多子**；占少数的载流子称为**少子**。



P型半导体



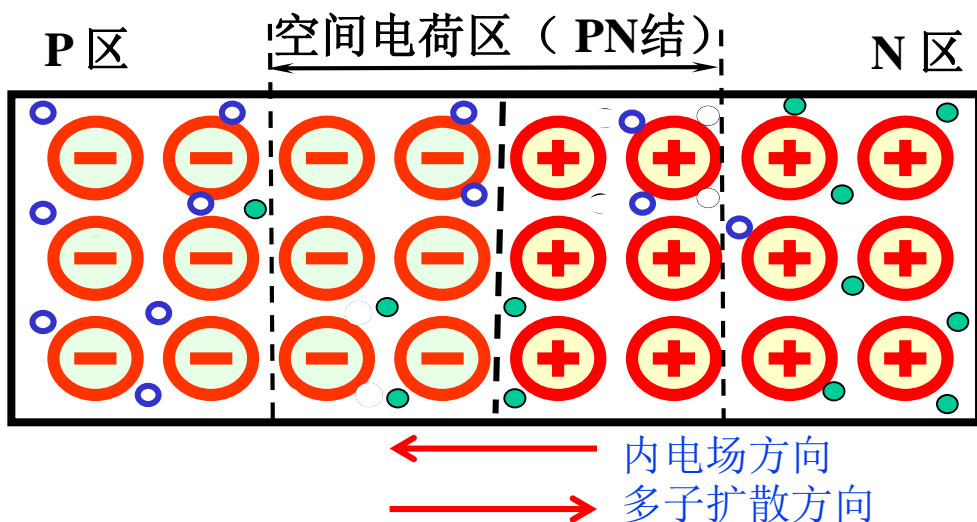
N型半导体

半导体基础知识简介

❖ **PN结**：将P型和N型半导体制作在一起，由于浓度差，P型半导体的空穴扩散进入N区，与N区的电子复合；N型半导体的电子扩散进入P区，与P区空穴复合，在交界面处形成一个PN结。

➤ **漂移电流**：载流子在电场中的定向运动称为漂移，由漂移形成的电流称为漂移电流。半导体中电子逆电场方向定向运动，空穴沿电场方向定向运动，形成半导体总的漂移电流。

➤ **扩散电流**：即使没有电场，由于载流子的浓度分布不均匀，也会发生载流子的定向运动——从浓度高的区域向浓度低的区域扩散，扩散运动形成的电流称为扩散电流，它正比于载流子的浓度梯度。



半导体基础知识简介

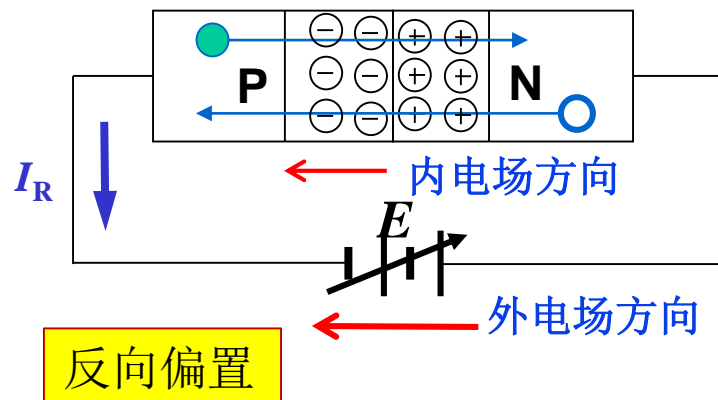
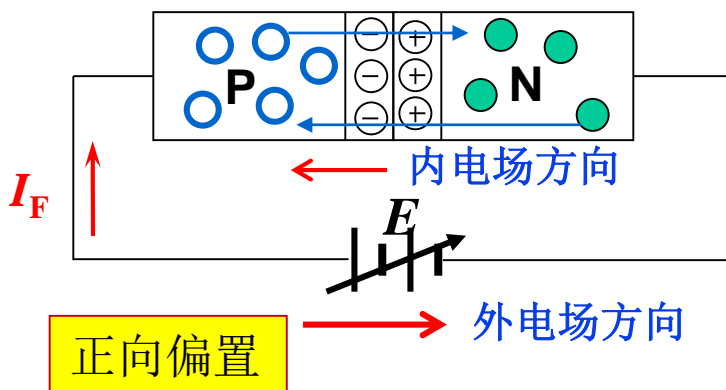
❖ **PN结正向偏置**：外部电压的正极接P区，负极接N区。

➤ 外电场与内电场方向**相反**，空间电荷区**变窄**。多子的扩散运动超过内电场作用下的少子的漂移运动，在PN结内形成了以**扩散电流**为主的**正向**的宏观电流 I_F ；该正向电流较大，PN结处于**导通**状态。

❖ **PN结反向偏置**：外部电压的正极接N区，负极接P区。

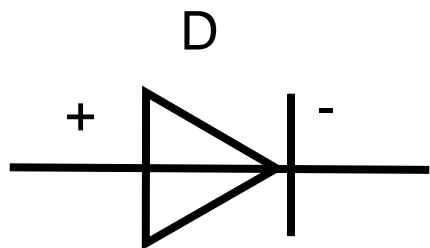
➤ 外电场与内电场方向**一致**，使空间电荷区**变宽**，多子的扩散运动受阻，少子的漂移运动超过多子的扩散运动，在PN结内形成了以**漂移电流**为主的**反向**电流 I_R 。该反向电流很小，约等于0，PN结**截止**。

❖ **无偏置**：PN结是平衡的，多子的扩散电流与少子的漂移电流平衡（大小相等、方向相反），PN结内无宏观电流。

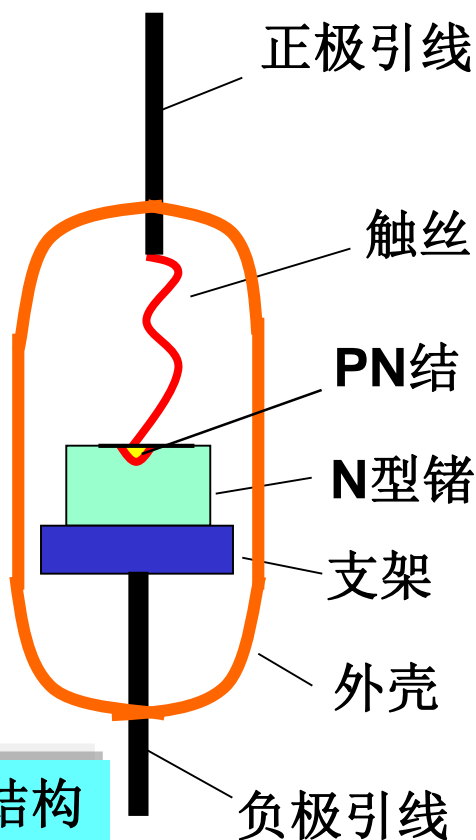


半导体二极管

- ❖ 一个PN结就是一只晶体二极管，记作D。
- ❖ 半导体二极管（晶体二极管）是在PN结两侧的中性区上各引出一个欧姆接触的金属电极构成的。
- ❖ 二极管按结构分为点接触型、面接触型和平面型二极管。
- ❖ 按材料划分为硅管和锗管。



二极管符号



点接触型二极管的结构



产品外形

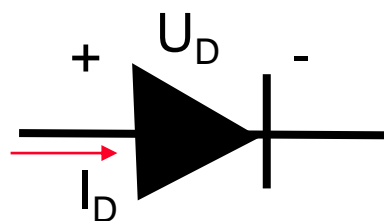
半导体二极管的开关特性

❖ 二极管单向导电性：受外加电压极性控制的开关特性

- 外加正向电压导通，外加反向电压截止；
- 正向开启电压：锗管约为0.2-0.5V；硅管为0.5-0.7V（课程以后默认为0.7V）

➤ 二极管特性

- 正向导通：二极管外加正向电压大于开启电压 V_D ，二极管导通。一旦导通，则 $U_D = V_D$ 不变。因此， V_D 称为钳位电压或正向压降。
- 反向截止：二极管外加反向电压或电压小于开启电压 V_D ，二极管截止。二极管截止，电流 $I_D = 0$
- 击穿：二极管两端外加反向电压超过一个阈值 (V_Z) 时，二极管会被击穿，此时二极管失去单向导电性，压降是 V_Z 。



半导体三极管

❖ 半导体三极管又称**晶体（三极）管**。由两层N型半导体中间夹一层P型半导体（NPN型）或两层P型半导体中间夹一层N型半导体（PNP型）组成。

➤ 半导体三极管的分类

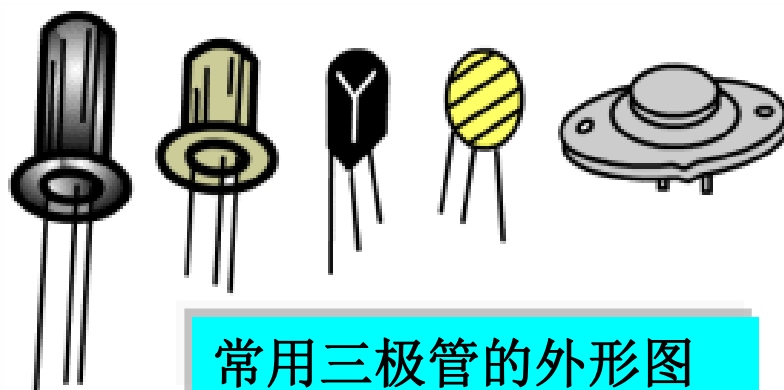
按**结构**划分 { NPN型
PNP型

按**功率**划分 { 大功率管
小功率管

按**用途**划分 { 放大管
开关管

按**材料**划分 { 硅管
锗管

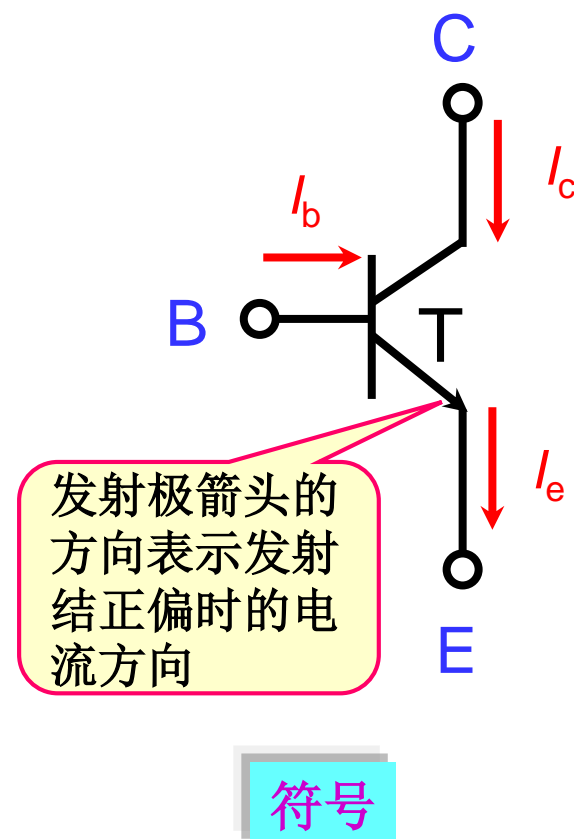
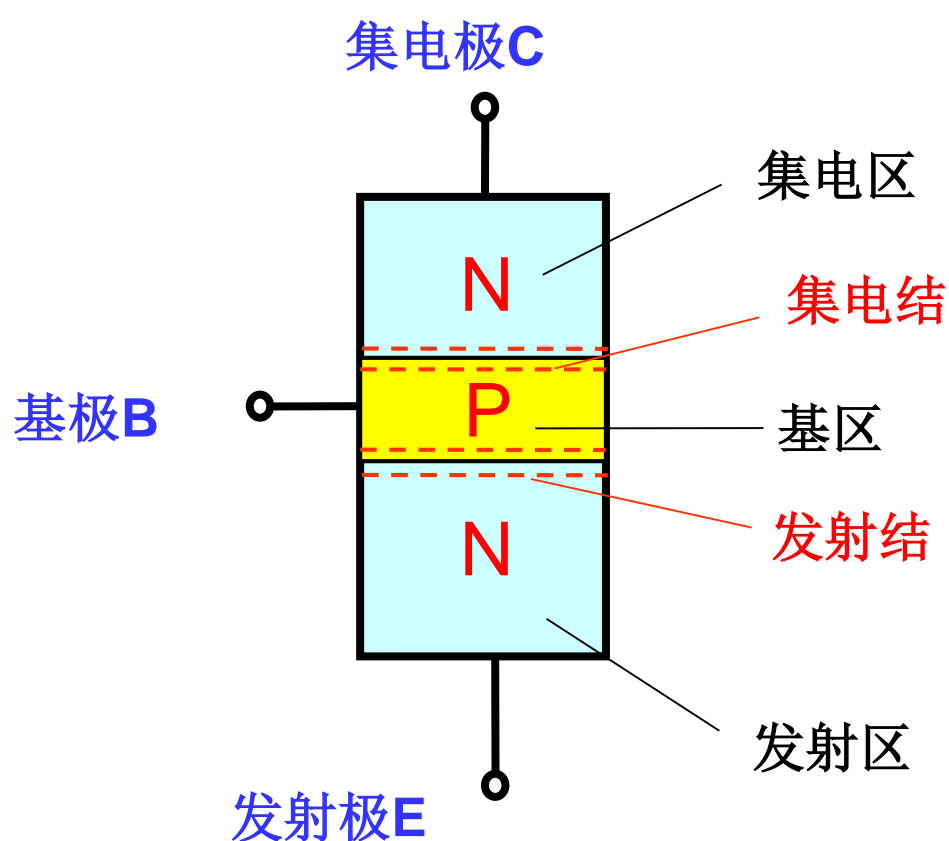
按**频率**划分 { 高频管
低频管



NPN型三极管

❖ 结构

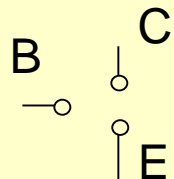
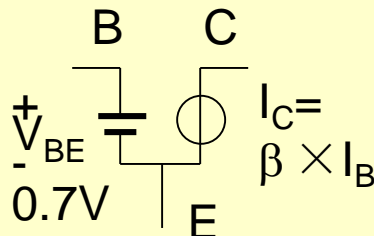
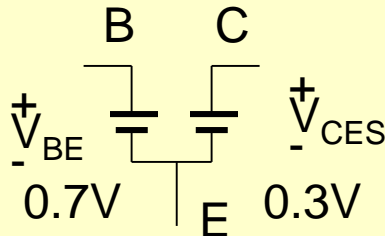
- 有3个电极， 3个区， 两个背向的PN结

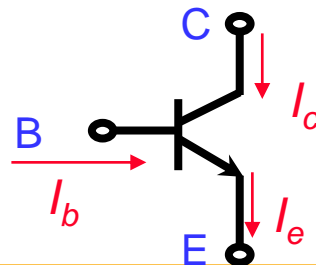


晶体三极管的开关特性

- ❖ 在模拟电路中，晶体三极管主要作为线性放大元件和非线性元件；在数字电路中，主要作为**开关元件**。
- ❖ **晶体管共发射极电路**放大能力强，也即控制能力强，只要在输入端加上两种不同幅值的信号，就可以控制晶体管的**导通或截止**。
- ❖ 作为开关电路，晶体三极管主要工作在**截止区**和**饱和区**。
- ❖ **三极管的稳态开关特性**是指三极管稳定在截止和饱和导通两种状态下的特性。

晶体三极管的稳态开关特性

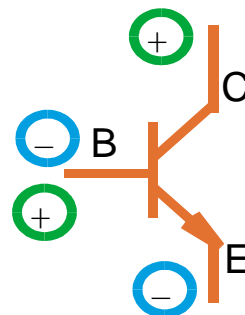
工作区	可靠条件	工程近似	特点	等效电路
<div>截止区</div> <div>(发射结反偏、 集电结反偏)</div>	$V_{BE} \leq 0$ $V_{BC} < 0$	$V_{BE} < 0.7V$ $V_{BC} < 0$	$I_B = I_C = I_E \approx 0$ $V_O = V_{CC}$	
<div>放大区</div> <div>(发射结正偏、 集电结反偏)</div>	$V_{BE} > 0$ $V_{BC} < 0$	$V_{BE} \geq 0.7V$ $V_{BC} < 0$	$I_C = \beta \times I_B$ $V_O = V_{CC} - I_C \times R_C$	
<div>饱和区</div> <div>(发射结正偏、 集电结正偏)</div>	$V_{BE} \geq 0.7V$ $V_{BC} > 0$ $I_B \geq I_{BS} = (V_{CC} - V_{CES}) / \beta R_C$	$V_{BE} \geq 0.7V$ $V_{BC} > 0$	$I_C = I_{CS}$ $= (V_{CC} - V_{CES}) / R_C$ $V_O = V_{CES} = 0.3V$	



晶体三极管三个工作区的特点总结

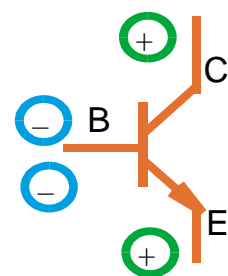
放大区：

- 发射结正偏, 集电结反偏
- 有电流放大作用, $I_C = \beta I_B$
- 输出曲线具有恒流特性



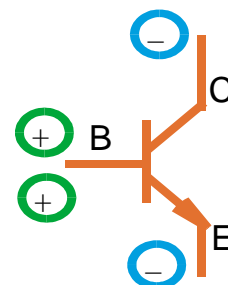
截止区：

- 发射结、集电结处于反偏
- 失去电流放大作用, $I_C \approx 0$
- 晶体管C、E之间相当于开路



饱和区：

- 发射结、集电结处于正偏
- 失去电流放大作用, $I_C = I_{CS}$, 不变
- 晶体管C、E之间相当于短路

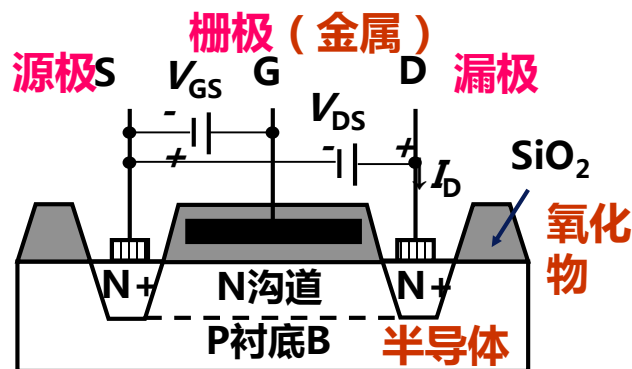


MOS管

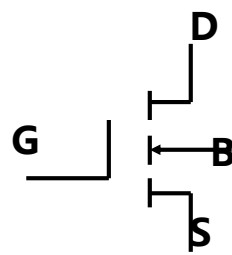
- ❖ MOS (Metal Oxide Semiconductor Field Effect Transistor,金属氧化物半导体场效应管)
 - MOS管属于单极型集成电路，只有一种载流子（自由电子或空穴）参与导电。
 - MOS管分为NMOS管和PMOS管两种类型

1、NMOS

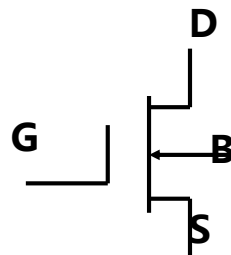
在P型半导体衬底上制作两个高掺杂浓度的N型区，形成源极和漏极。



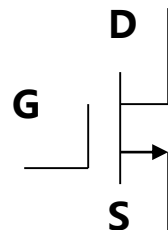
NMOS管有增强型和耗尽型两种类型。



增强型
NMOS管

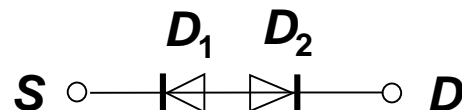
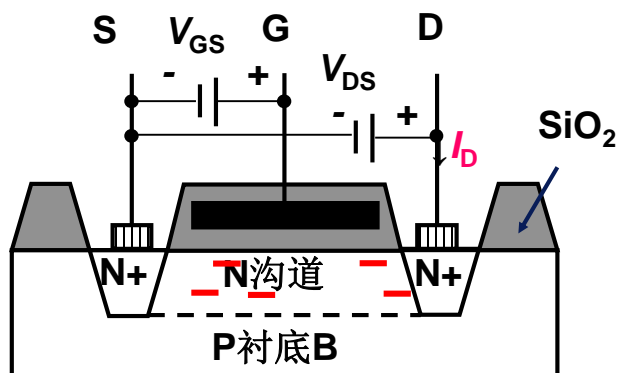


耗尽型
NMOS管



NMOS管
简化符号

NMOS管的工作原理

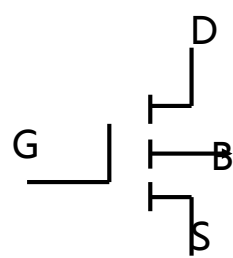
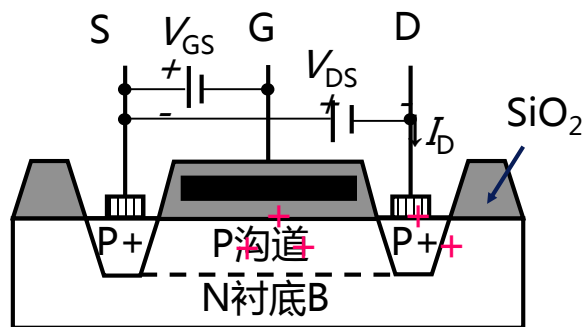


- ◆ 对于增强型NMOS管，如果 $V_{GS}=0$ ，则两个N区和P型底衬形成两个背向的PN结，无论 V_{DS} 为正或负，总有一只PN结反偏，NMOS管都不能导通， $I_D=0$ 。
- ◆ 当 $V_{GS} > V_{GS(TH)}$ （开启电压，1~3V），自由电子在正电场的吸引下，聚集在栅极下的衬底表面，形成N型沟道，把两个N区沟通，在 V_{DS} 作用下，NMOS管导通，形成漏极电流 I_D 。
- ◆ 随着 V_{GS} 升高，导电沟道的截面积将增大， I_D 增加——可以通过改变 V_{GS} 控制 I_D 的大小。

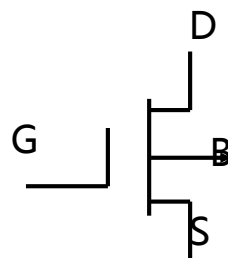
PMOS管

2、PMOS管

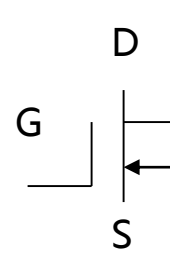
- PMOS管是在N型半导体衬底上制作两个高掺杂浓度的P型区，形成源极和漏极。
- PMOS管也有增强型和耗尽型两种类型。



增强型
PMOS管



耗尽型
PMOS管

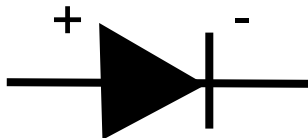
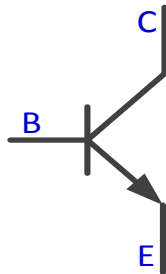
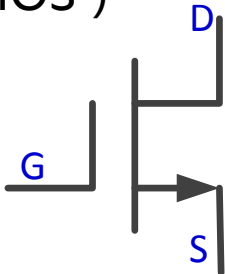


PMOS管
简化符号

- ❖ 对于增强型PMOS管，当 $V_{GS}=0$ 时，则两个P区和N型衬底形成两个背向的PN结，无论 V_{DS} 为正或负，PMOS管都不能导通， $I_D=0$ 。
- ❖ 当 $V_{GS} < -V_{GS(TH)}$ ，自由电子在负电场的排斥下，栅极下的底衬表面的自由电子数量减少、空穴数量大大增加，形成P型沟道，把两个P区沟通，PN结消失，PMOS管导通，形成漏极电流 I_D 。

小结

晶体管与MOS管开关特性

名称	特性
二极管 	<ul style="list-style-type: none">• 导通：正向导通 ($U_D > 0.7V$)• 截止：反向截止 ($U_D < 0.7V$)
三极管 (NPN) 	<ul style="list-style-type: none">• 导通：发射结与集电结正偏 ($U_{BC} > 0, U_{BE} > 0.7V$)• 截止：发射结与集电结反偏 ($U_{BC} < 0, U_{BE} < 0.7V$)
MOS管 (NMOS) 	<ul style="list-style-type: none">• 导通：$V_{GS} > \text{开启电压}$• 截止：$V_{GS} = 0$

第二讲：组合逻辑

一. 逻辑代数基础

1. 逻辑代数基本概念
2. 逻辑代数的公理、定理与规则
3. 逻辑函数的表达式
4. 逻辑函数化简（卡诺图）

二. 逻辑门电路

1. 晶体管和MOS管
2. 逻辑门电路实现

三. 基本组合逻辑部件设计

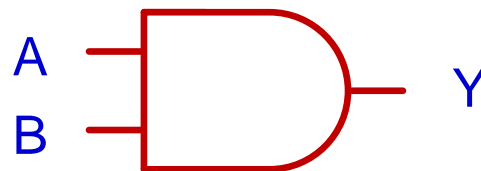
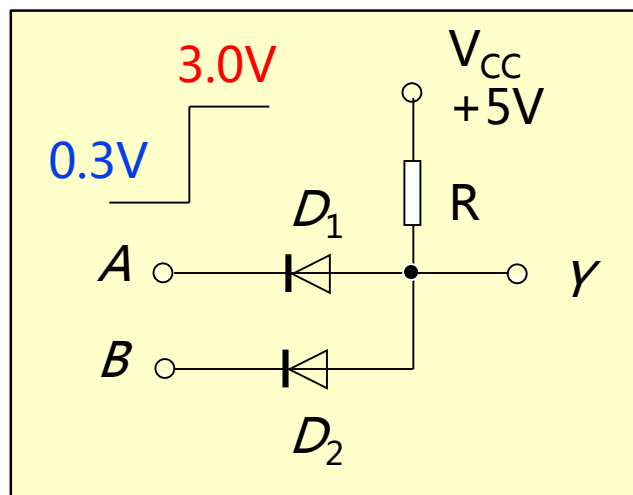
门电路概述

- ❖ “门电路” 是实现某种逻辑关系的电路，它是数字电路的基本逻辑单元电路。基本的逻辑门有与门、或门、非门，复合逻辑门有与非门、或非门、与或非门、异或门等。
- ❖ 逻辑门电路主要分为两类
 - 分立元件门：由电阻、二极管、三极管等分立元件构成
 - 集成门：把构成门电路的基本元件制作在一小片半导体芯片上
 - 集成反相器、缓冲器
 - 集成与门、与非门
 - 集成或门、或非门
 - 集成异或门
 - 集成三态门

“与”门电路（二极管实现）

❖ 实现逻辑与运算的电路称为**与门**。

电路结构



工作原理

- 当 A、B 为高电平（高电平额定值 V_{iH} 为 3.0V）时， D_1 、 D_2 均导通，则输出位高电平（ $Y = 3.0 + 0.7 = 3.7V$ ）。
- 当 A、B 为低电平（低电平额定值 V_{iL} 为 0.3V）时， D_1 、 D_2 均导通，由于二极管导通后钳位电压为 0.7V，则输出 $Y = 0.3 + 0.7 = 1.0V$ ；
- 当 A 为低电平、B 为高电平时， D_1 优先导通，输出 $Y = 0.3 + 0.7 = 1.0V$ ， D_2 被反偏截止；

“与”门电路（二极管实现）

◆ 功能描述

(1) 功能表

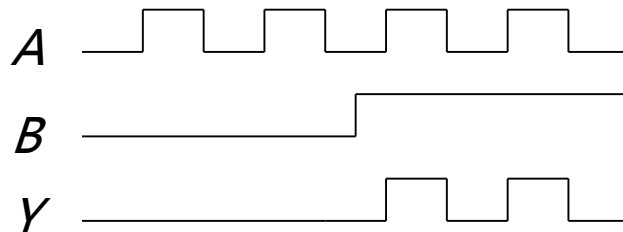
A(V) B(V)	Y(V)
0.3 0.3	1.0
0.3 3.0	1.0
3.0 0.3	1.0
3.0 3.0	3.7

(2) 真值表

A B	Y
0 0	0
0 1	0
1 0	0
1 1	1

(3) 表达式 $Y=AB$

(4) 工作波形图（时序图）



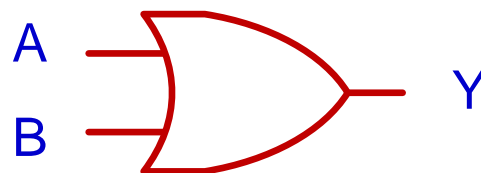
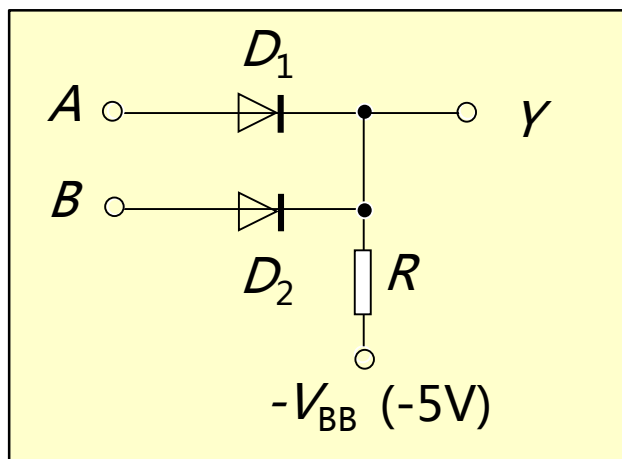
在功能表中，用0代表低电平（输入0.3V，输出1.0V），用1代表高电平（输入3.0V，输出3.7V），则可以得到真值表

- ❖ 与门的时序图体现了“门”的概念！
 - 若A为输入信号，B为控制信号，当B为低电平时，无输出信号，门是“关闭”的；
 - 当B为高电平时，输出信号Y同输入信号，门是“打开”的。

“或”电路（二极管实现）

❖ 实现或逻辑运算的电路称为**或门**。

电路结构



工作原理

- 当A、B为低电平0.3V时， D_1 、 D_2 均导通，由于二极管导通后的钳位电压为0.7V，则输出 $Y=0.3-0.7=-0.4V$ 。
- 当A为0.3V、B为3.0V时， D_2 优先导通，则输出 $Y=3.0-0.7=2.3V$ ；由于A只有0.3V，则 D_1 被反偏截止。
- 当A、B均为高电平3.0V时， D_1 、 D_2 导通，则输出 $Y=3-0.7=2.3V$ 。

“或”门电路（二极管实现）

◆ 功能描述

(1) 功能表

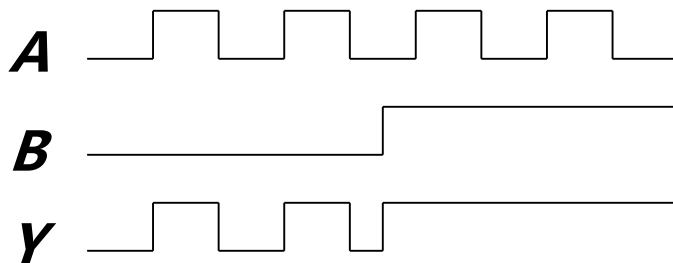
$A(V)$	$B(V)$	$Y(V)$
0.3	0.3	-0.4
0.3	3.0	2.3
3.0	0.3	2.3
3.0	3.0	2.3

(2) 真值表

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

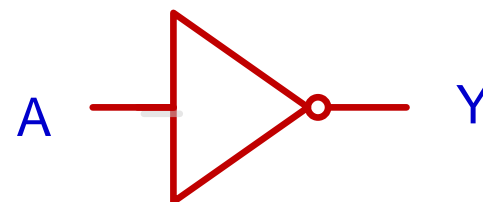
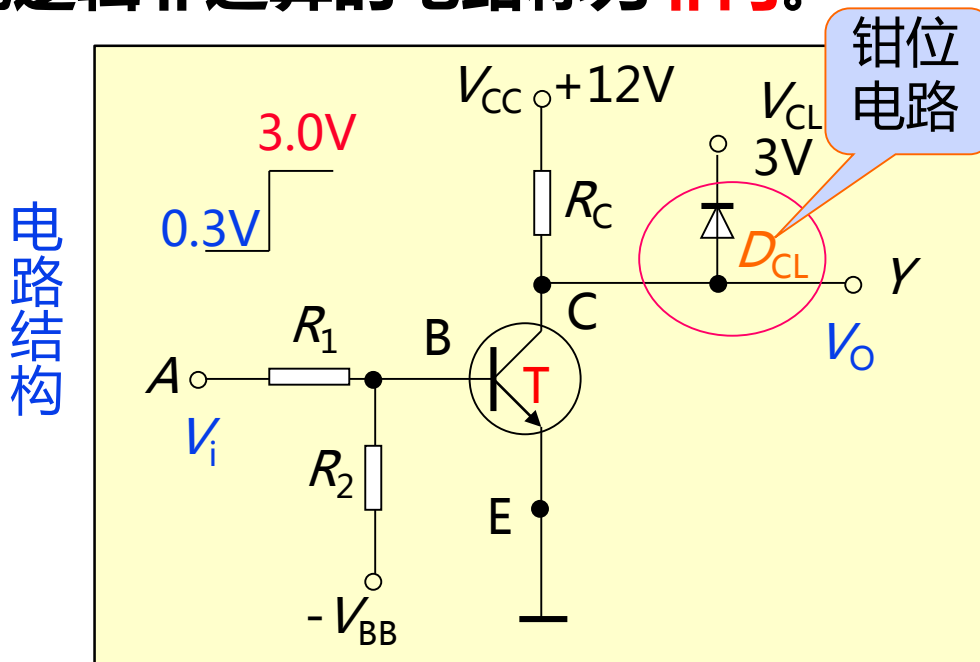
(3) 表达式 $Y = A + B$

(4) 工作波形（时序图）



“非”门电路（三极管实现）

❖ 实现逻辑非运算的电路称为**非门**。



工作原理

- 当 $V_i = 0.3V$ 时， T 截止； D_{CL} 导通，输出 $V_O \approx V_{CL} = 3.0V$ （忽略 D_{CL} 上的电压降时）。
- 当 $V_i = 3V$ 时， T 饱和导通， $V_{CES} \approx 0.3V$ ； D_{CL} 截止，输出 $V_O = V_{CES} = 0.3V$ 。

“非” 门电路（三极管实现）

◆ 功能描述

(1) 功能表

A(V)	Y(V)
0.3	3.0
3.0	0.3

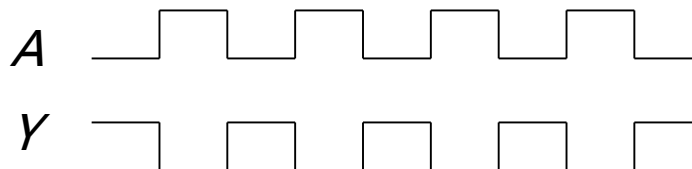
(2) 真值表

A	Y
0	1
1	0

(3) 表达式 $Y = \overline{A}$

输出与输入之间满足“非”逻辑关系，所以此电路称为**非门**。
输出与输入波形有180°的相位差，所以非门也称为**反相器**。

(4) 工作波形（时序图）



“非”门电路（CMOS实现）

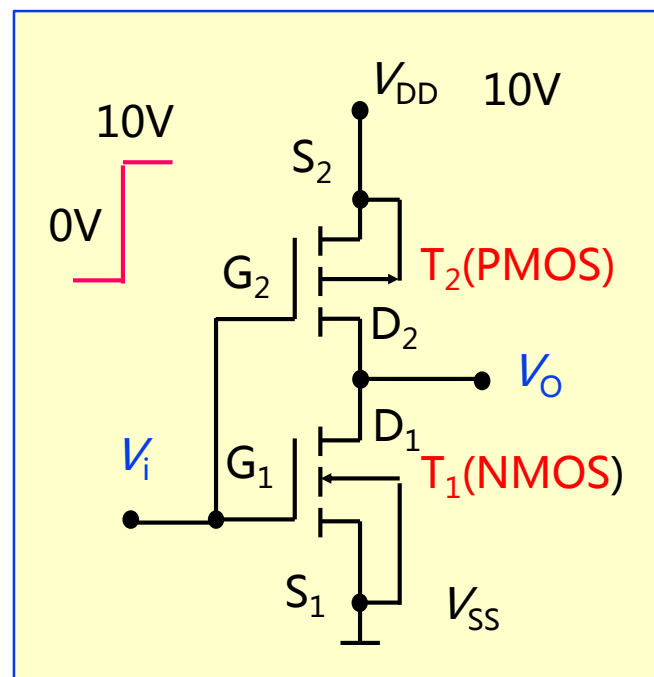
❖ CMOS是由NMOS和PMOS管形成的电路结构，称为**互补MOS**（Complementary Symmetry MOS）

（1）电路结构

驱动管 T_1 ——NMOS
负载管 T_2 ——PMOS } 互补MOS

（2）工作原理

- 当 $V_i = V_{iL} = 0V$ 时， $V_{GS1} < V_{GS(TH)}$ ， T_1 截止， $V_{GS2} = -10V < -V_{GS(TH)}$ ， T_2 导通，输出 $V_o = V_{oH} = 10V$ 。
- 当 $V_i = V_{iH} = 10V$ 时， $V_{GS1} > V_{GS(TH)}$ ， T_1 导通， $V_{GS2} = 0V$ ， T_2 截止，输出 $V_o = V_{oL} = 0V$ 。



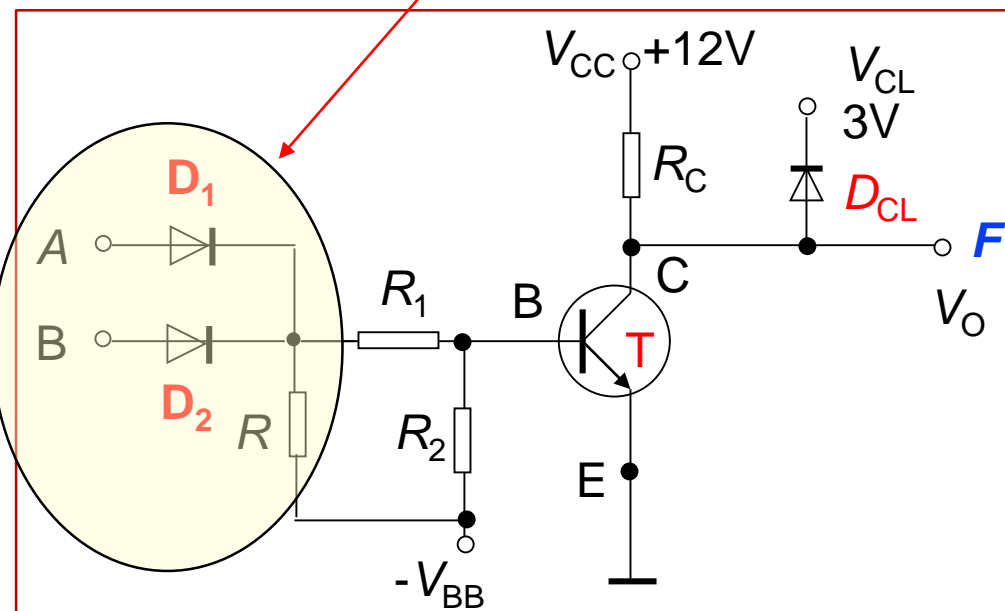
❖ CMOS反相器中，两个MOS管总有一只处于截止状态，使得驱动管和负载管的导通电阻都可以做得很小——使电路**驱动负载能力增强**，同时也使静态功耗极低。

或非门电路（分立元件实现）

二极管或门电路

❖ 或非门由二极管或门和三极管非门复合而成。

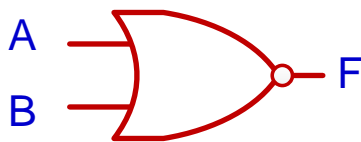
- 只要A或B有一个高电平（3.0V），二极管或门的输出就为高电平，经三极管非门反相后，输出为低电平；
- 只有全部输入为低电平（0.3V）， D_1 、 D_2 均导通，二极管或门的输出才为低电平（-0.4V），T截止，输出 V_O 为高电平（3.7V）。



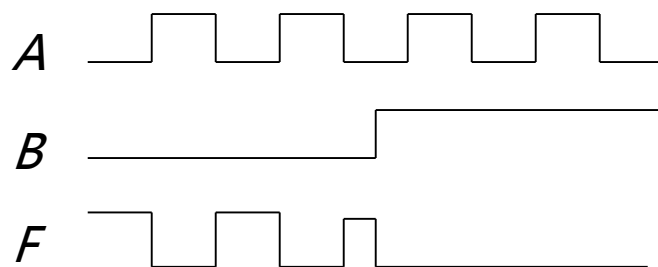
① 真值表

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

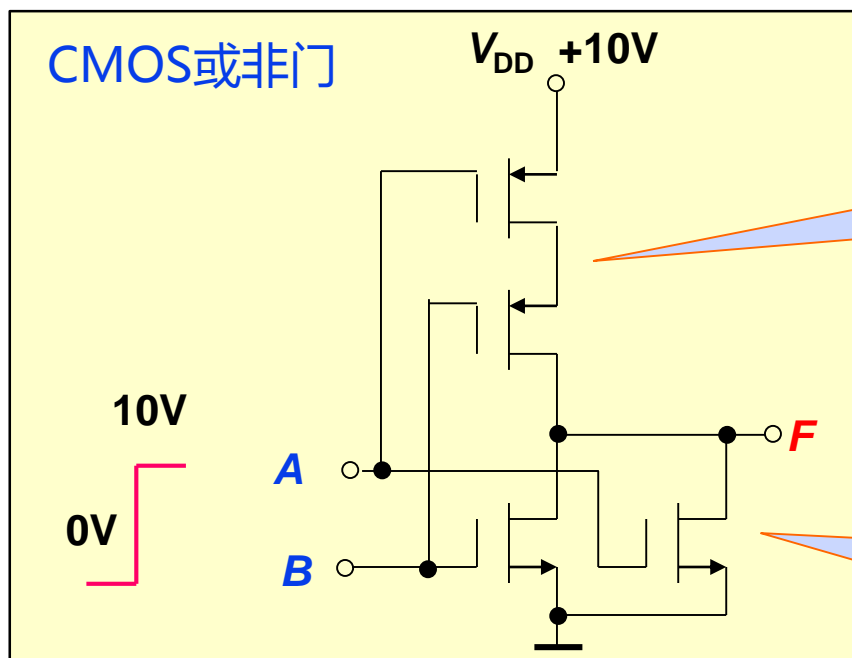
② 表达式 $F = \overline{A+B}$



③ 工作波形（时序图）

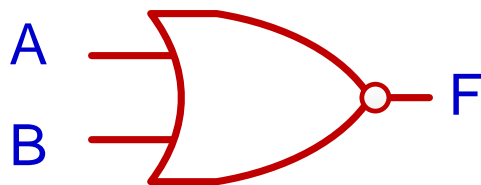


或非门电路（CMOS实现）



增强型
PMOS管

增强型
NMOS管



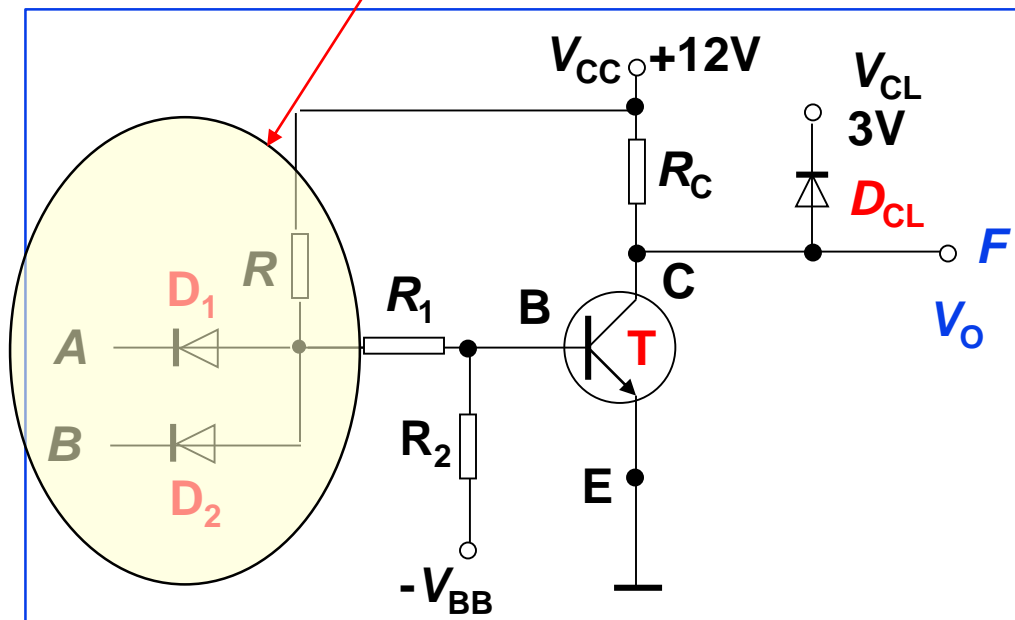
- CMOS或非门：两个CMOS反相器的负载管串联、驱动管并联后得到。
- 当A、B中任一个（或两个）为高电平时，并联支路中至少一只驱动管导通，串联支路中至少一只负载管截止（则串联支路截止），输出 $F \approx 0V$ ，为低电平；
- 只有A、B均为低电平时，并联支路中两只驱动管才全部截止，串联支路中两只负载管才全部导通，输出 $F \approx V_{DD}$ ，为高电平。

与非门电路（分立元件实现）

二极管与门电路

❖ 与非门由二极管与门和三极管非门复合而成。

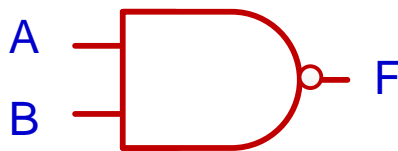
- 当A、B均为高电平（3.0V）时， D_1 、 D_2 均导通，二极管与门的输出为高电平（3.7V），经三极管非门反相后，输出为低电平（0.3V）；
- 其他输入条件下，二极管与门的输出为低电平（1.0V），输出 V_O 为高电平（3.7V）。



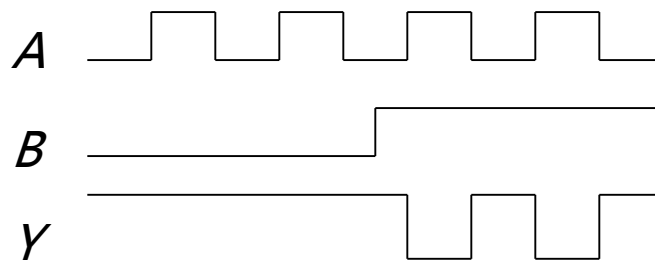
① 真值表

A B	Y
0 0	1
0 1	1
1 0	1
1 1	0

② 表达式 $F = \overline{A \cdot B}$



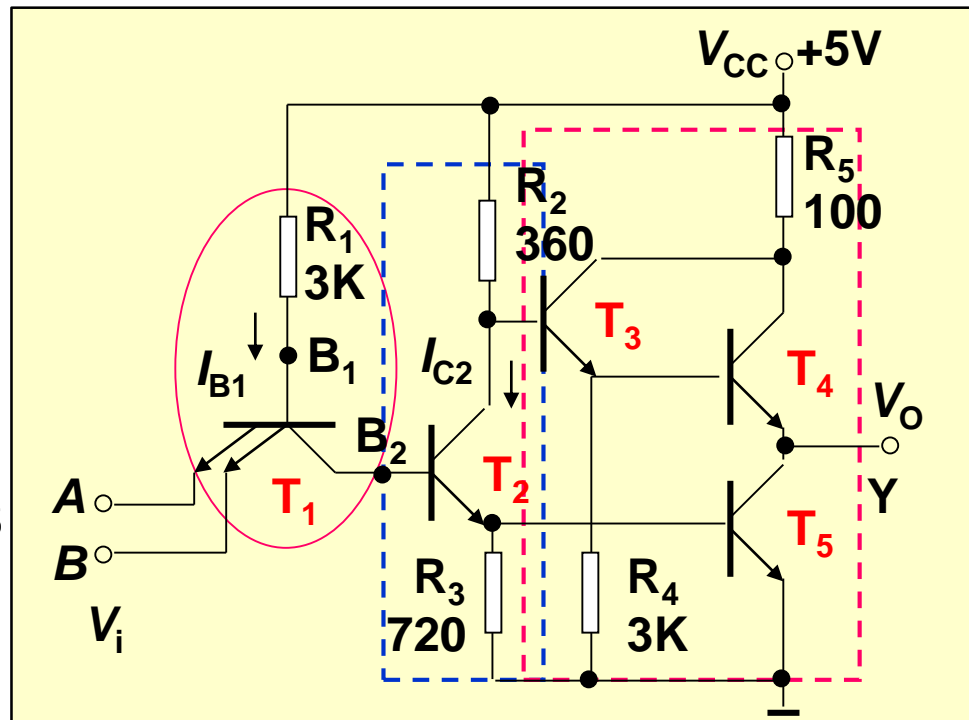
③ 工作波形（时序图）



与非门电路（TTL集成电路实现）

❖ TTL集成电路是双极型集成电路，其输入端和输出端都由晶体三极管构成，称为**晶体管-晶体管逻辑**，简称**TTL**（Transistor-Transistor Logic）。

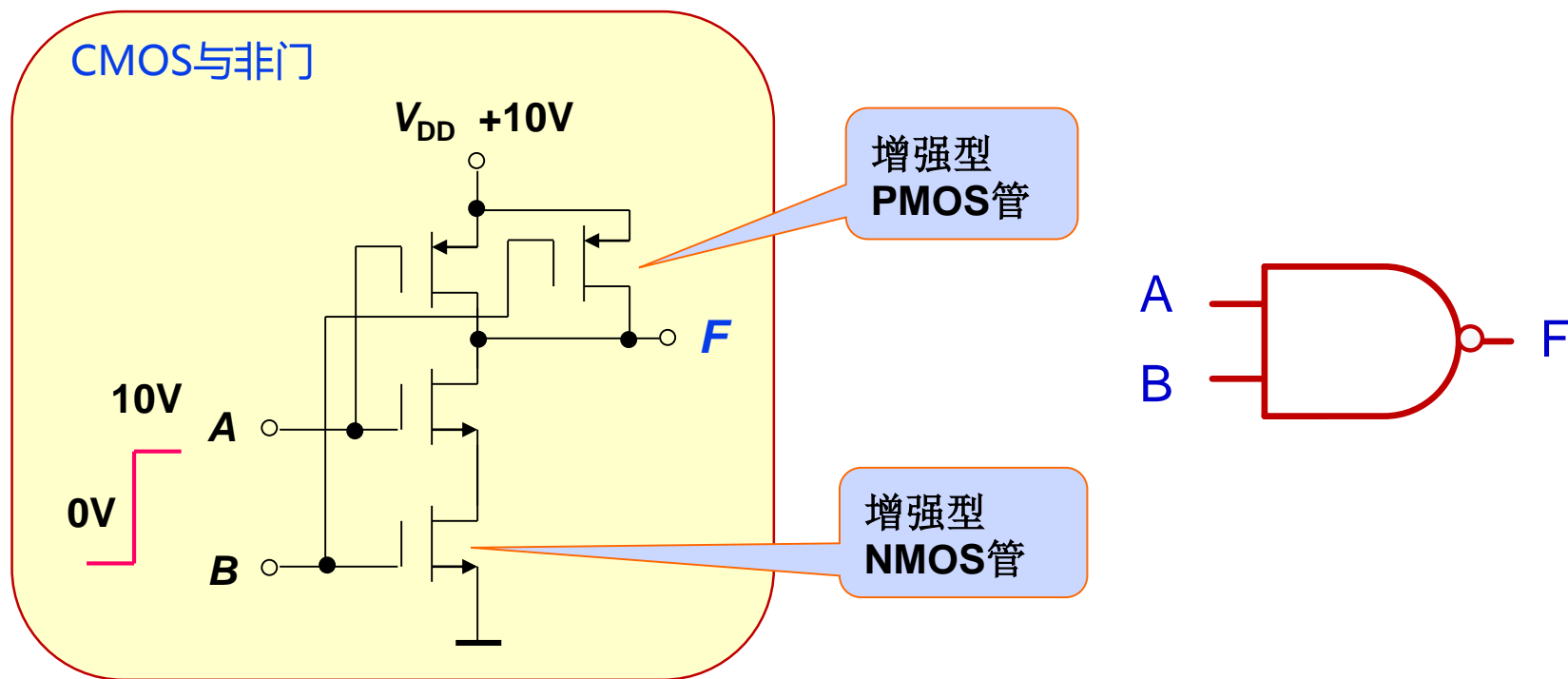
- （1）输入级：多发射极晶体管 T_1 和 R_1 ，完成“与”功能
- （2）中间级： T_2 、 R_2 和 R_3 ，完成“两相驱动”功能
- （3）输出级： T_3 、 T_4 、 T_5 、 R_4 和 R_5 完成“**推拉输出**”功能



TTL与非门电路

电路根据 T_5 的工作状态分为开态和关态： T_5 饱和时输出为 V_{OL} ——开态（On）； T_5 截止时输出为 V_{OH} ——关态（Off）。

与非门电路（CMOS实现）



- CMOS与非门：两个CMOS反相器负载管并联、驱动管串联后得到。
- 当A、B中任一个为低电平时，串联支路中至少一只驱动管截止，并联支路中至少一只负载管导通（ $V_{GS} = -10V$ ），输出 $F \approx V_{DD}$ ，为高电平；
- 只有A、B均为高电平时，串联支路中两只驱动管才全部导通，并联支路中两只负载管才全部截止（ $V_{GS} = -10V$ ），输出 $F \approx 0V$ ，为低电平。

各种集成门电路性能比较

- ◆ CMOS和TTL是两种比较常用的集成电路，各有特色。
CMOS功耗相对低、抗干扰能力相对强、带载能力相对强；
TTL功耗相对高，速度相对快、抗干扰能力相对弱。
- ◆ 与TTL门电路相比，传统的CMOS门电路特点是集成度高、功耗低，但工作速度较慢、抗静电能力差。不过目前新型的CMOS门电路工作速度已经有了很大提高、抗静电能力也大为改善，基本能够与TTL门电路相媲美了。
- ◆ CMOS门电路获得了更为广泛的应用，尤其在大规模集成电路和微处理器中已占据了重要地位。

第二讲：组合逻辑

一. 逻辑代数基础

1. 逻辑代数基本概念
2. 逻辑代数的公理、定理与规则
3. 逻辑函数的表达式
4. 逻辑函数化简（卡诺图）

二. 逻辑门电路

三. 基本组合逻辑部件设计

1. 组合逻辑设计概述
2. 运算单元电路
3. 编码器/译码器
4. 多路选择器

组合逻辑电路的特点

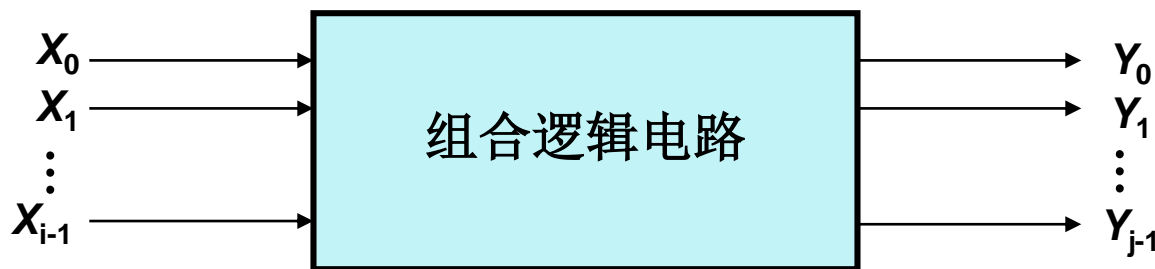
❖ **数字电路分类**：组合逻辑电路和时序逻辑电路

❖ **组合逻辑电路**：将逻辑门以一定的方式组合在一起，使其具有一定逻辑功能的数字电路。

- 组合逻辑电路是一种无记忆电路——任一时刻的输出信号仅取决于该时刻的输入信号，而与信号作用前电路原来所处的状态无关。

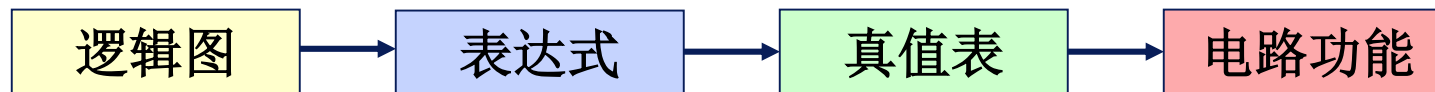
❖ **特点**

- 由逻辑门电路组成
- 没有反馈电路和存储电路
- 当时的输出仅由当时的输入决定——速度快。

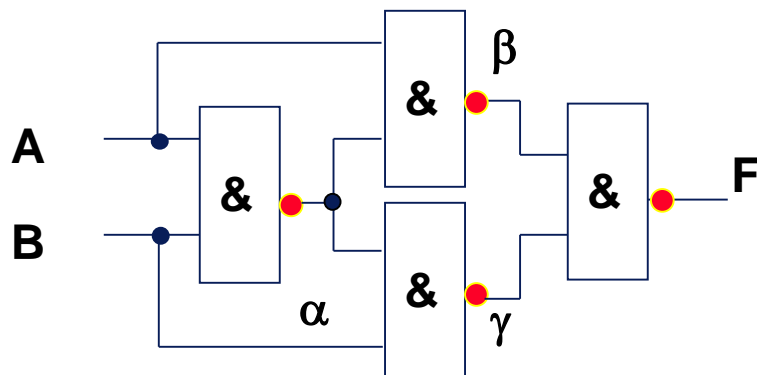


组合逻辑电路的表示方法

❖ 组合逻辑电路的不同表示方法



【例】分析下图电路



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

$$\alpha = \overline{AB}$$

$$\beta = \overline{\alpha A} = \overline{\overline{AB}A}$$

$$\gamma = \overline{\alpha B} = \overline{\overline{AB}B}$$

$$F = \overline{\beta\gamma} = \overline{\overline{\overline{AB}A} \cdot \overline{\overline{AB}B}} = \overline{\overline{AB}A} + \overline{\overline{AB}B} \\ = (\overline{A} + \overline{B})(A + B) = \overline{A}B + A\overline{B}$$

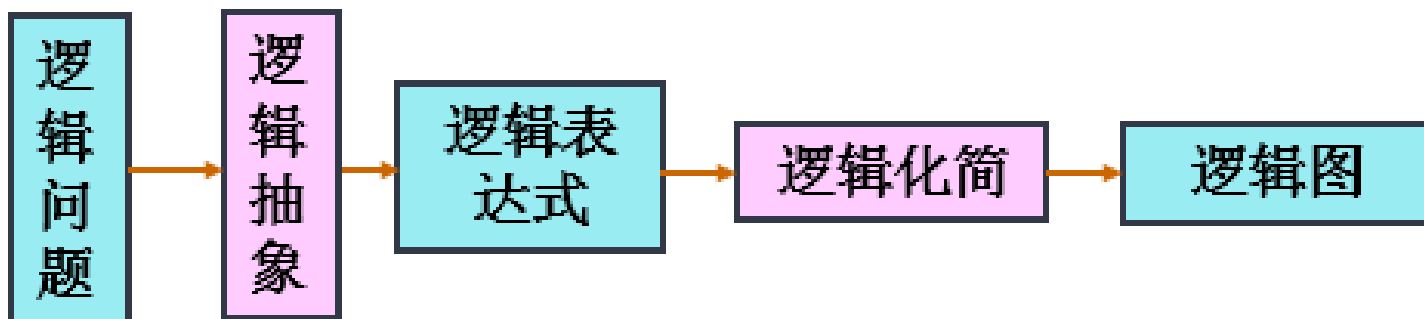
电路功能：异或电路

组合逻辑电路的设计方法

❖ **组合逻辑电路设计**：根据功能要求，采用某种设计方法，得到满足功能要求、且最简单的组合逻辑电路。

❖ 手工设计方法

- **逻辑抽象**：确定输入、输出变量，列出真值表
- **写出逻辑表达式**：根据真值表写出逻辑函数的标准表达式
- **逻辑化简**：用公式化简法或卡诺图化简法化简为最简逻辑函数表达式
- **绘逻辑图**：根据最简逻辑函数表达式画出原理图。



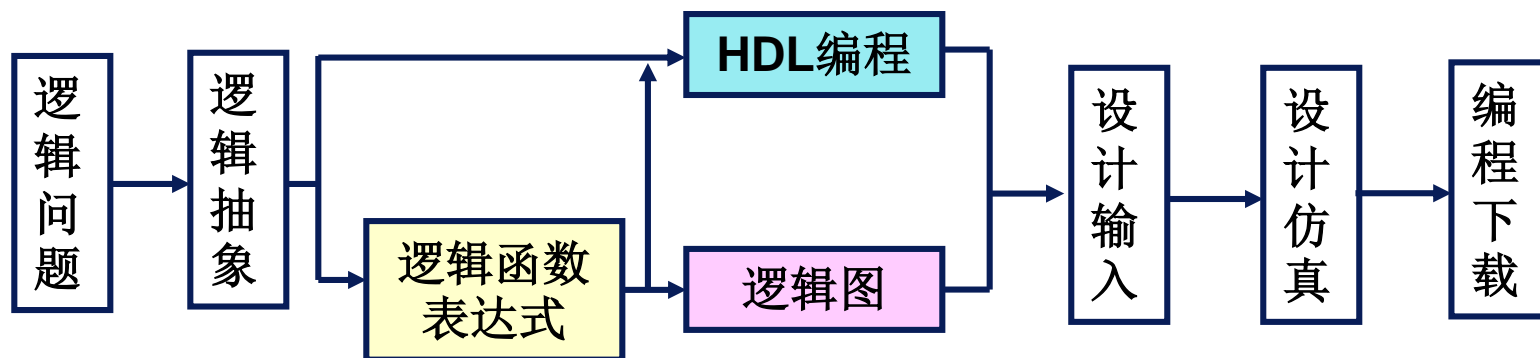
组合逻辑电路的设计方法

❖ **自动设计方法**：基于HDL和EDA工具的组合逻辑电路的设计方法。

- **逻辑抽象**：确定输入、输出变量，列出真值表（复杂系统也可不写出真值表，而直接用HDL的系统级描述方式）
- **写出表达式**：根据真值表写出逻辑函数的标准表达式
- **HDL编程**：如用case语句、if-else语句，assign语句

❖ **三种途径**

1. 逻辑抽象→HDL编程（系统级描述，如用case语句或if-else语句）
2. 逻辑抽象→写出表达式→HDL编程（算法级描述，assign语句）
3. 逻辑抽象→写出表达式→绘逻辑图（适于简单电路）



第二讲：组合逻辑

一. 逻辑代数基础

1. 逻辑代数基本概念
2. 逻辑代数的公理、定理与规则
3. 逻辑函数的表达式
4. 逻辑函数化简（卡诺图）

二. 逻辑门电路

三. 基本组合逻辑部件设计

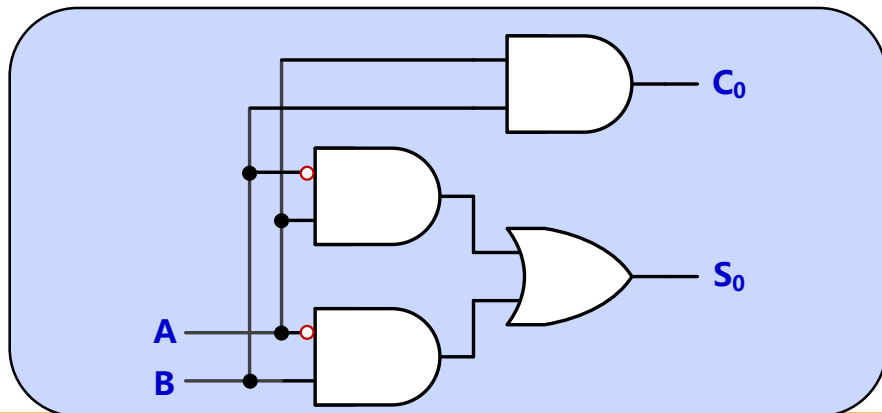
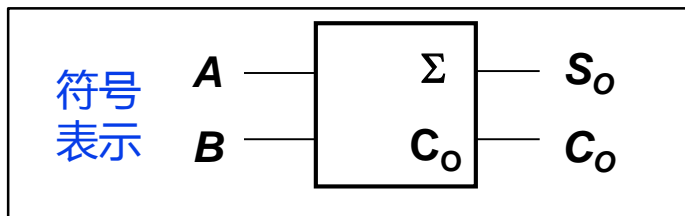
1. 组合逻辑设计概述
2. 运算单元电路
3. 编码器/译码器
4. 多路选择器

3.2.1 加法运算电路

❖ 加法运算电路

- **半加器**：对两个1位二进制数进行相加求和，并向高位进位的逻辑电路，不考虑来自低位的进位。
- **全加器**：对两个1位二进制数进行相加求和，考虑来自低位的进位，并向高位进位的逻辑电路，

1. 半加器



半加器真值表

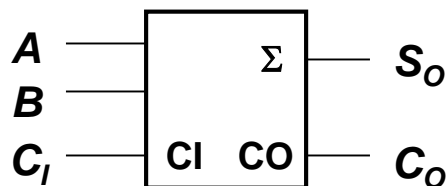
A	B	S_o	C_o
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$C_o = A \cdot B$$

$$S_o = \bar{A}B + A\bar{B} = A \oplus B$$

3.2.1 加法运算电路

2. 全加器

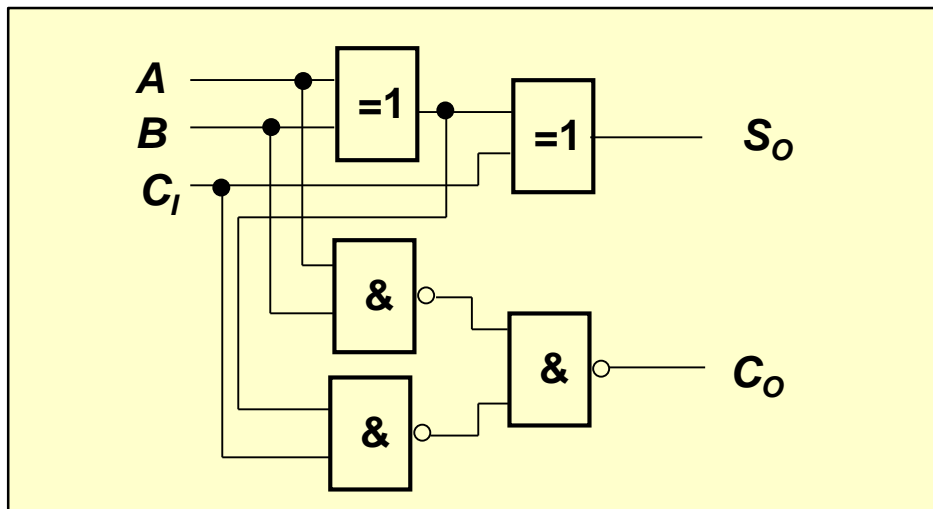


$$\begin{aligned} S_o &= \overline{\overline{A}}\overline{\overline{B}}\overline{C_i} + \overline{\overline{A}}\overline{\overline{B}}\overline{C_i} + \overline{\overline{A}}\overline{\overline{B}}\overline{C_i} + \overline{\overline{A}}\overline{\overline{B}}\overline{C_i} \\ &= A \oplus B \oplus C_i \end{aligned}$$

$$\begin{aligned} C_o &= \overline{\overline{A}}\overline{\overline{B}}\overline{C_i} + \overline{\overline{A}}\overline{\overline{B}}\overline{C_i} + \overline{\overline{A}}\overline{\overline{B}}\overline{C_i} + \overline{\overline{A}}\overline{\overline{B}}\overline{C_i} \\ &= AB + C_i(A \oplus B) \end{aligned}$$

全加器真值表

$A B C_i$	S_o	C_o
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



3.2.1 加法运算电路—全加器

□全加器的Verilog HDL

- 方法一：根据逻辑表达式，用**assign**语句建模（算法级描述）

$$S_o = \overline{\overline{A}}\overline{\overline{B}}\overline{C_i} + \overline{\overline{A}}\overline{\overline{B}}C_i + \overline{\overline{A}}\overline{B}\overline{\overline{C_i}} + \overline{\overline{A}}\overline{B}C_i$$

$$C_o = \overline{\overline{A}}\overline{B}\overline{C_i} + \overline{\overline{A}}\overline{B}C_i + \overline{\overline{A}}\overline{B}\overline{C_i} + \overline{\overline{A}}\overline{B}C_i$$

1位全加器Verilog HDL源程序（assign建模）

```
module adder_1(A,B,CI,SO,CO);  
    input  A,B,CI;  
    output SO,CO;  
    assign SO = (!A&&!B&&CI)//(!A&&B&&!CI)//  
                (A&&!B&&!CI)//(A&&B&&CI);  
    assign CO = (!A&&B&&CI)//(A&&!B&&CI)//  
                (A&&B&&!CI)//(A&&B&&CI);  
endmodule
```

3.2.1 加法运算电路—全加器

□全加器的Verilog HDL

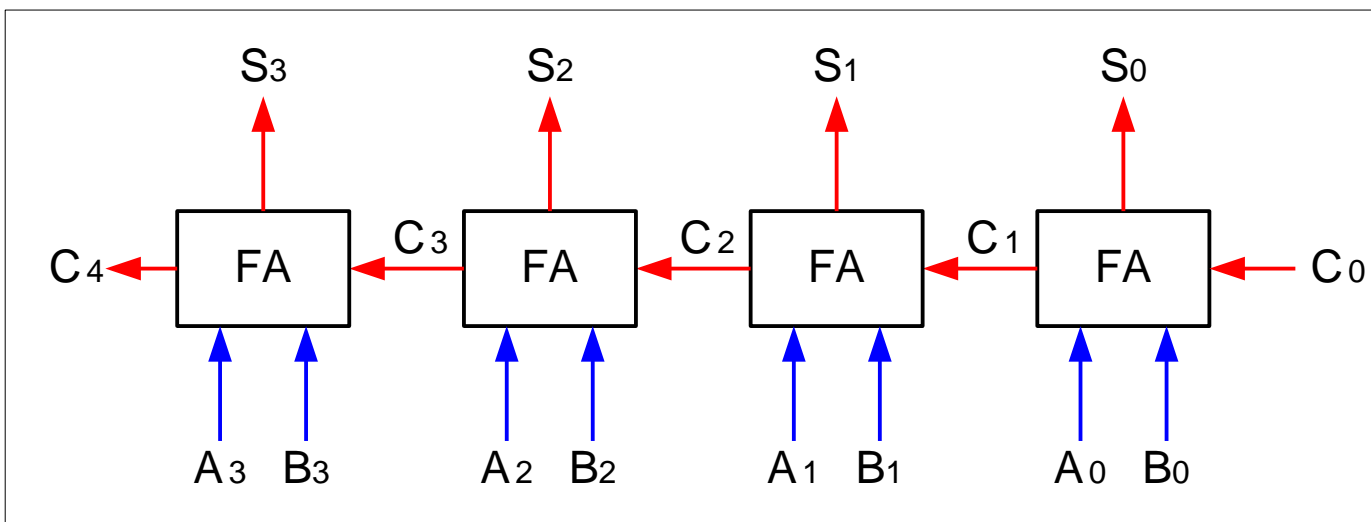
- 方法二：根据逻辑功能定义直接描述，采用行为描述方式的系统级抽象，程序更简洁！

```
module adder_2(A,B,CI,SO,CO);  
  input          A,B,CI;  
  output         SO,CO;  
  assign         {CO,SO} = A+B+CI;  
endmodule
```

- 这里用位拼接运算符 “{ }”将进位与算术和拼接在一起成为一个2位数

3.2.1 加法运算电路—多位加法器

❖ 并行加法器——串行进位



$$C_1 = A_0 B_0 + C_0 (A_0 \oplus B_0)$$

$$C_2 = A_1 B_1 + C_1 (A_1 \oplus B_1)$$

$$C_3 = A_2 B_2 + C_2 (A_2 \oplus B_2)$$

$$C_4 = A_3 B_3 + C_3 (A_3 \oplus B_3)$$

串行进位的特点:

1. 进位串行传递
2. 进位延时较长

3.2.1 加法运算电路—多位加法器

❖ 并行加法器——并行进位（或先行进位）

$$\text{令 } G_i = A_i B_i, P_i = A_i \oplus B_i$$

$$C_1 = G_0 + P_0 C_0$$

$$C_2 = G_1 + C_1 P_1 = G_1 + G_0 P_1 + P_0 P_1 C_0$$

$$C_3 = G_2 + C_2 P_2 = G_2 + G_1 P_2 + G_0 P_1 P_2 + P_0 P_1 P_2 C_0$$

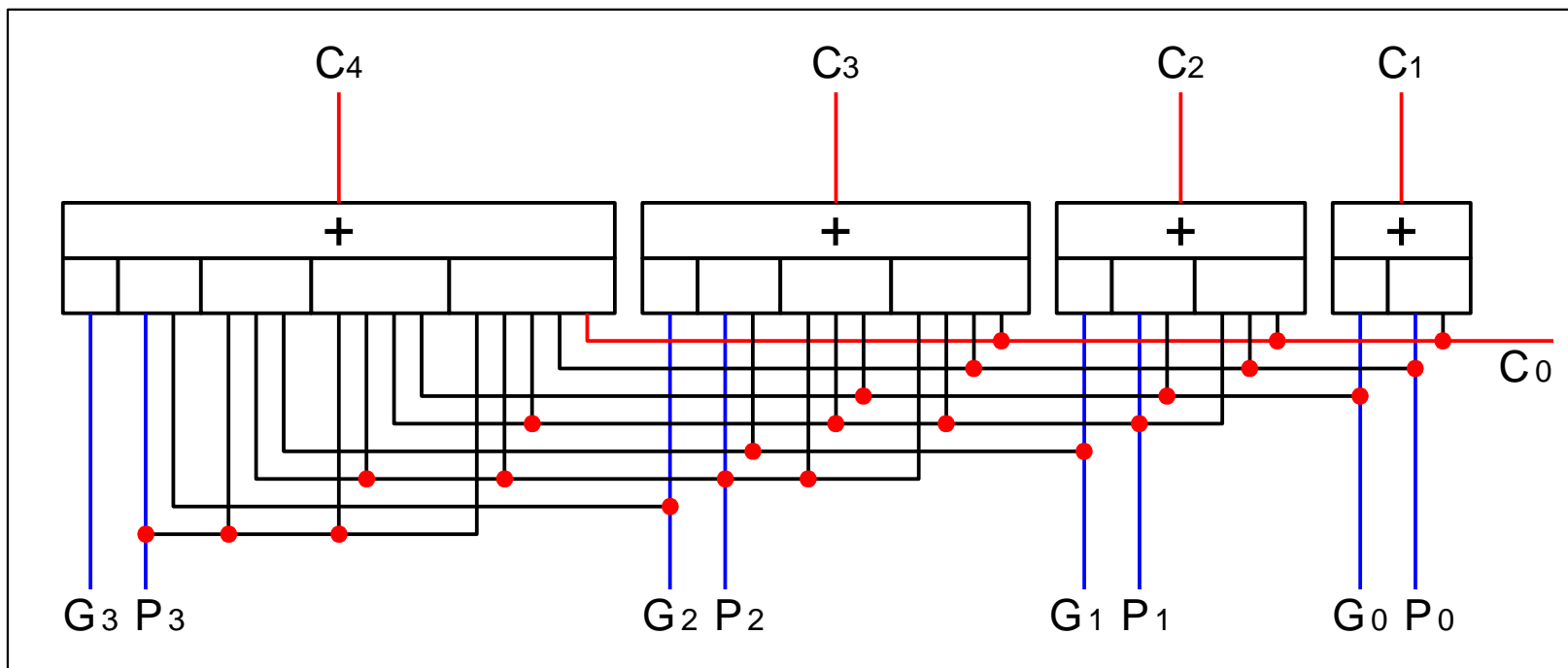
$$C_4 = G_3 + C_3 P_3 = G_3 + G_2 P_3 + G_1 P_2 P_3 + G_0 P_1 P_2 P_3 + P_0 P_1 P_2 P_3 C_0$$

❖ 并行进位的特点

- 同时产生进位
- 加法延时缩短
- 实现相对复杂

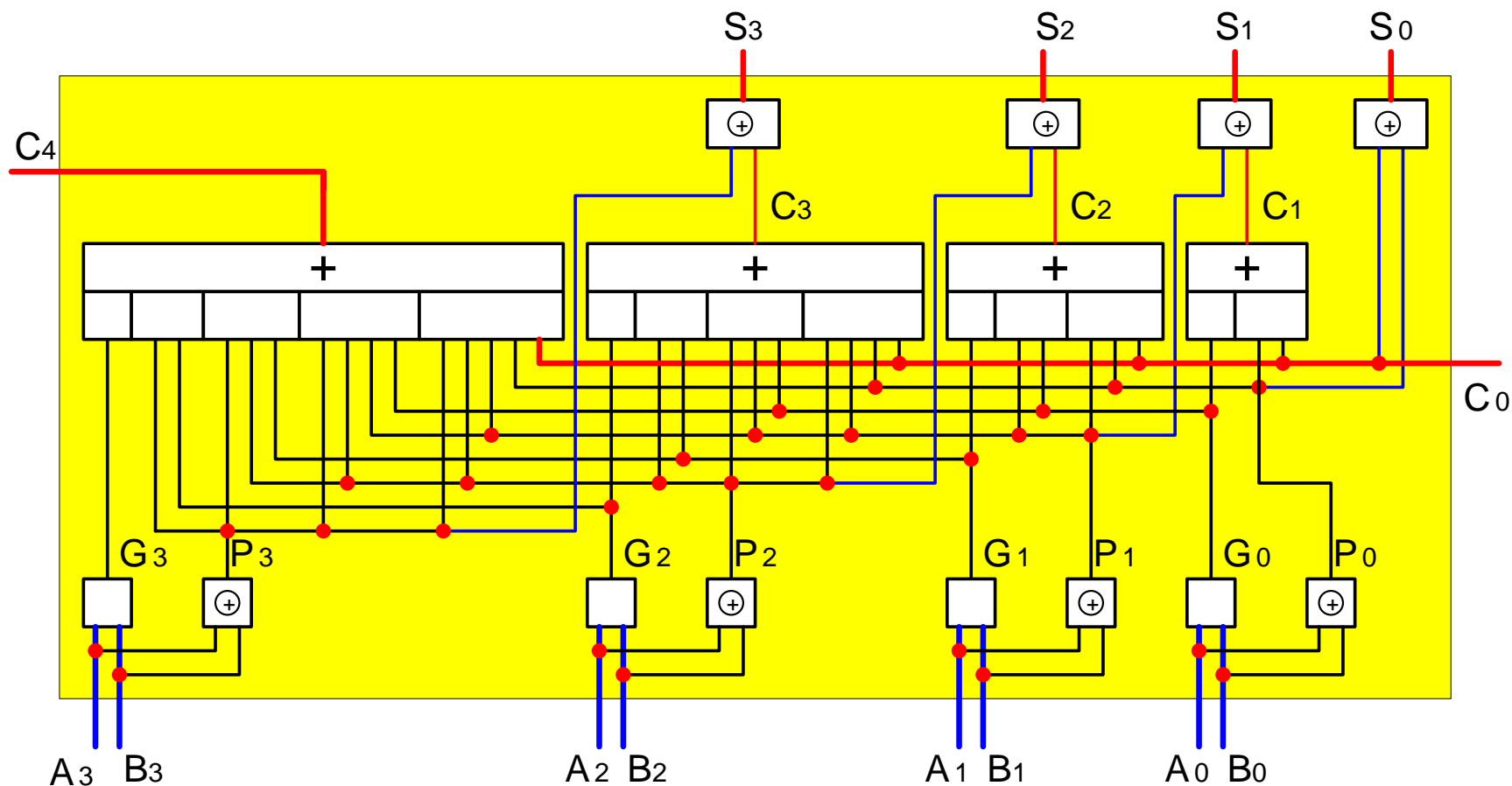
3.2.1 加法运算电路—多位加法器

❖ 并行进位链



3.2.1 加法运算电路—多位加法器

❖ 并行进位加法器



3.2.1 加法运算电路—多位加法器

□ 8位加法器的Verilog HDL

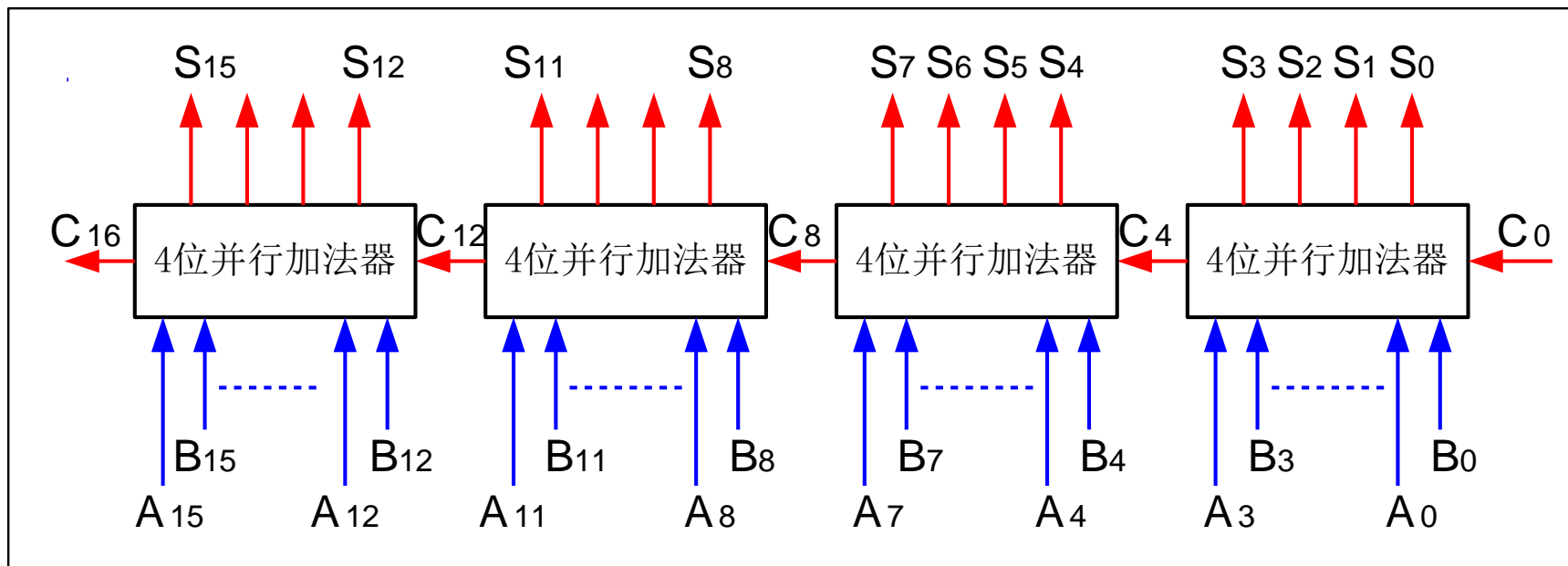
➤ 采用行为描述方式的系统级抽象

```
module adder_8(a,b,cin,sum,cout);  
    parameter                width=8;  
    input [width-1:0]    a,b;  
    input                                cin;  
    output [width-1:0]    sum;  
    output                                cout;  
    assign                {cout,sum} = a+b+cin;  
endmodule
```

❖ 用parameter常量width表示加法器的位数，通过修改width，可以方便地实现不同位宽的加法器。

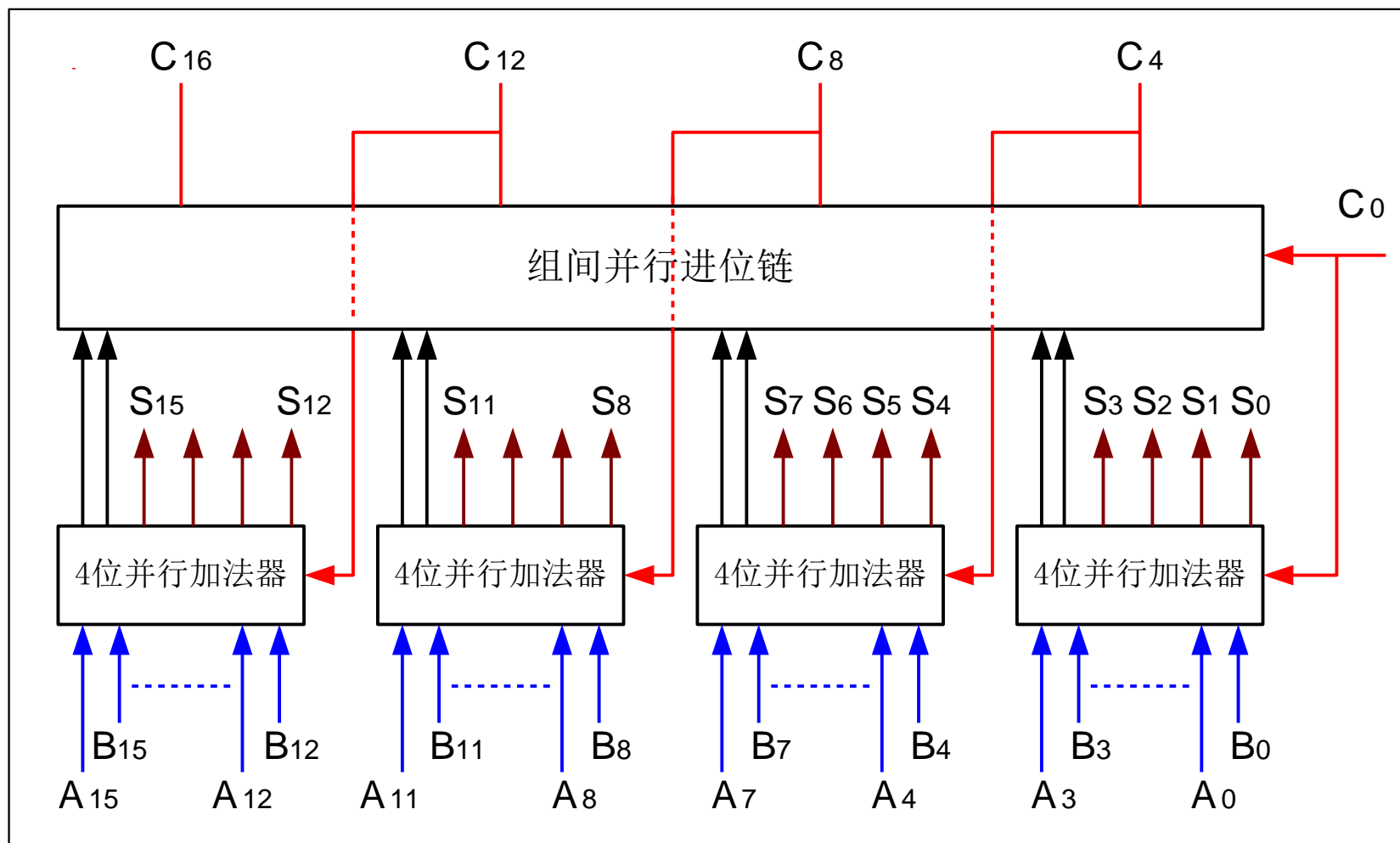
3.2.1 加法运算电路—分组并行进位加法器

❖ 分组并行进位加法器（组内并行，组间传递）



3.2.1 加法运算电路—分组并行进位加法器

❖ 分组并行进位加法器（组内并行，组间并行）



3.2.1 加法运算电路—减法运算

❖ 原则

$$[A + B]_{\text{补}} = [A]_{\text{补}} + [B]_{\text{补}}$$

$$[A - B]_{\text{补}} = [A]_{\text{补}} + [-B]_{\text{补}}$$

❖ $[X]_{\text{补}}$ 与 $[-X]_{\text{补}}$ (以定点整数为例说明)

$$\text{若 } [x]_{\text{补}} = x_0 x_1 x_2 \dots x_{n-1}$$

$$\text{则 } [-x]_{\text{补}} = \overline{x_0} \overline{x_1} \overline{x_2} \dots \overline{x_{n-1}} + 1$$

所以有

$$[A - B]_{\text{补}} = [A]_{\text{补}} + \overline{[B]_{\text{补}}} + 1$$

3.2.2 乘法运算电路：阵列乘法器

❖ 基本思路

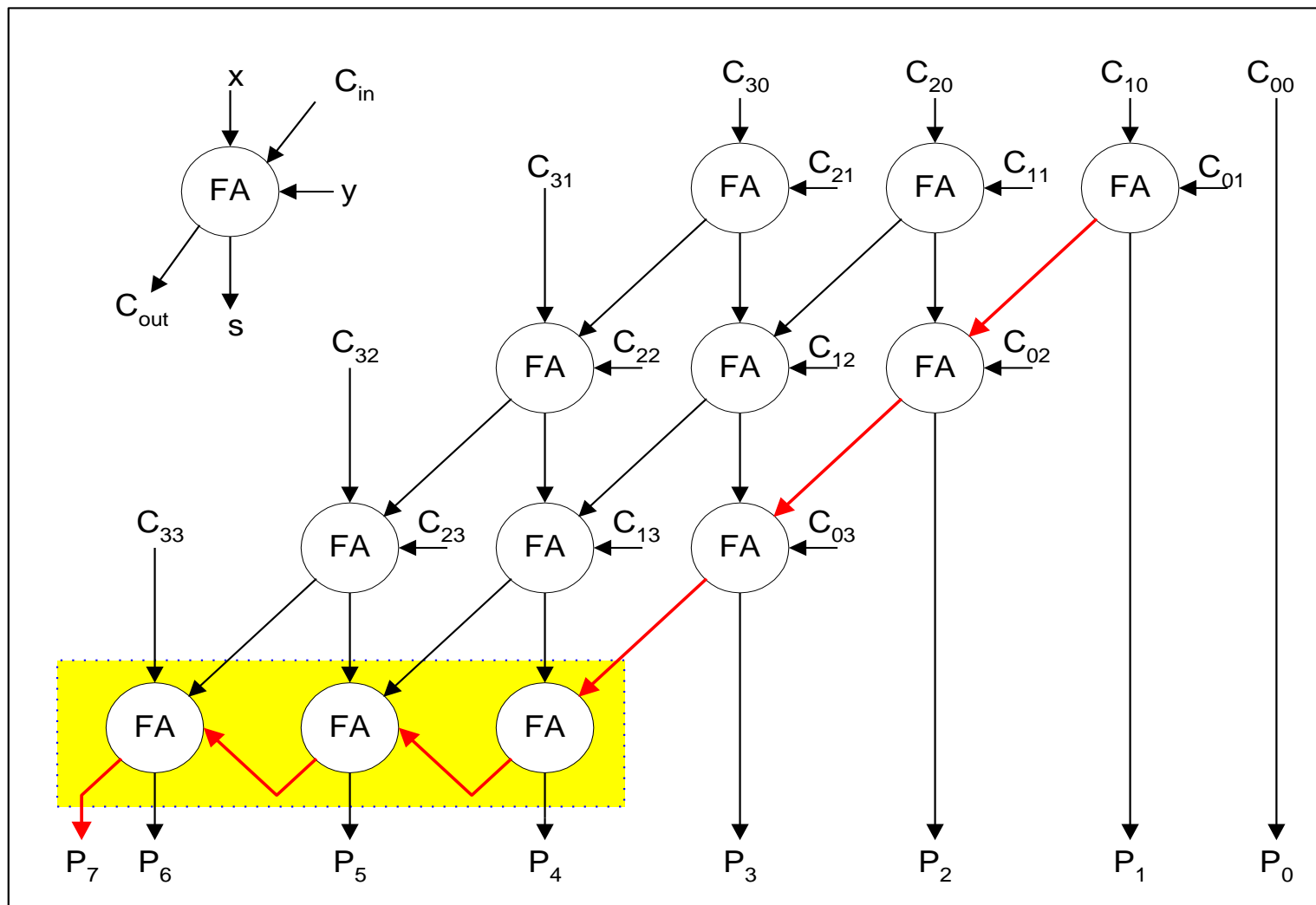
- 利用若干全加器，完全由硬件直接计算乘法结果
- 以 4 位无符号数为例

$$\begin{array}{rcccccccc} & & & & A_3 & A_2 & A_1 & A_0 \\ \times & & & & B_3 & B_2 & B_1 & B_0 \\ \hline & & & & C_{30} & C_{20} & C_{10} & C_{00} \\ & & & C_{31} & C_{21} & C_{11} & C_{01} & \\ & & C_{32} & C_{22} & C_{12} & C_{02} & & \\ + & C_{33} & C_{23} & C_{13} & C_{03} & & & \\ \hline P_7 & P_6 & P_5 & P_4 & P_3 & P_2 & P_1 & P_0 \end{array}$$

其中 $C_{ij} = A_i B_j$

3.2.2 乘法运算电路：阵列乘法器

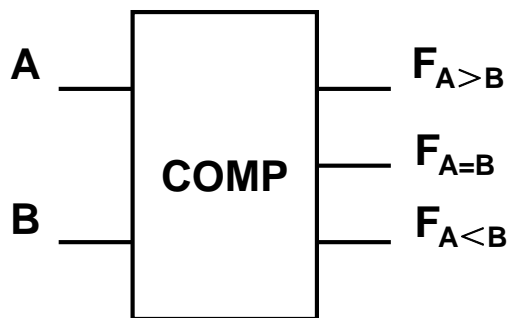
❖ 实现电路



3.2.3 数值比较器

❖ **数值比较器**：一种关系运算电路，它可以对两个二进制数进行比较，得出大于、小于和相等的结果。

❖ **1位比较器**



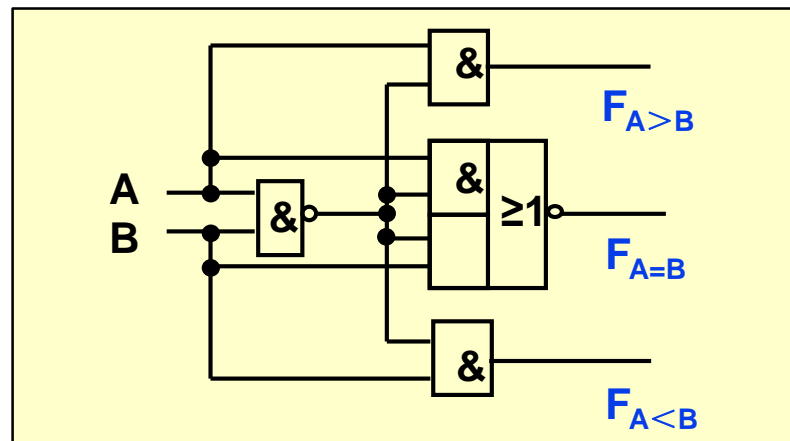
1位比较器真值表

A B	$F_{A>B}$	$F_{A=B}$	$F_{A<B}$
0 0	0	1	0
0 1	0	0	1
1 0	1	0	0
1 1	0	1	0

$$F_{A>B} = \overline{A}\overline{B} = A(\overline{A} + \overline{B}) = \overline{AAB}$$

$$F_{A<B} = \overline{A}B = B(\overline{A} + \overline{B}) = \overline{BAB}$$

$$\begin{aligned} F_{A=B} &= AB + \overline{A}\overline{B} = \overline{\overline{A}\overline{B} + \overline{AB}} \\ &= \overline{AAB} + \overline{BAB} \end{aligned}$$

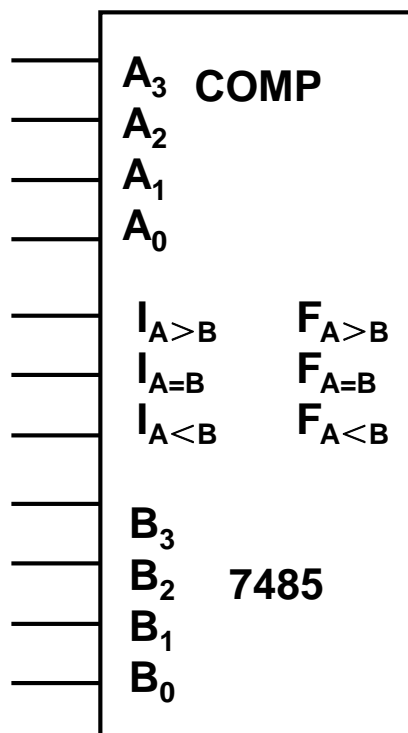


3.2.3 数值比较器

级联输入端，
用于芯片的
扩展

❖ 4位比较器(7485芯片)

(2) 功能表



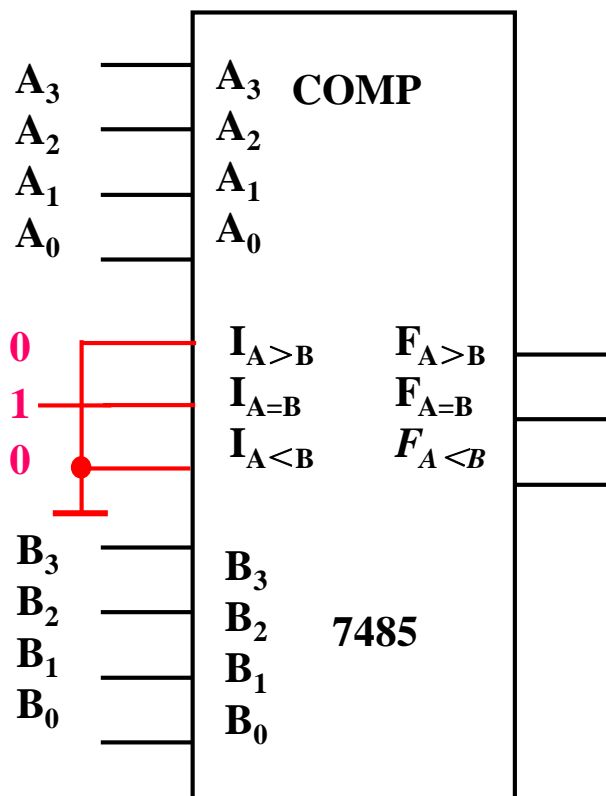
A ₃ B ₃	A ₂ B ₂	A ₁ B ₁	A ₀ B ₀	I _{A>B} I _{A=B} I _{A<B}	F _{A>B} F _{A=B} F _{A<B}
A ₃ >B ₃	X	X	X	X X X	1 0 0
A ₃ <B ₃	X	X	X	X X X	0 0 1
A ₃ =B ₃	A ₂ >B ₂	X	X	X X X	1 0 0
A ₃ =B ₃	A ₂ <B ₂	X	X	X X X	0 0 1
A ₃ =B ₃	A ₂ =B ₂	A ₁ >B ₁	X	X X X	1 0 0
A ₃ =B ₃	A ₂ =B ₂	A ₁ <B ₁	X	X X X	0 0 1
A ₃ =B ₃	A ₂ =B ₂	A ₁ =B ₁	A ₀ >B ₀	X X X	1 0 0
A ₃ =B ₃	A ₂ =B ₂	A ₁ =B ₁	A ₀ <B ₀	X X X	0 0 1
A ₃ =B ₃	A ₂ =B ₂	A ₁ =B ₁	A ₀ =B ₀	a b c	a b c

规则：从高位开始比较，高位不等时，数值的大小由高位决定；若高位相等，则再比较低位，数值的大小由低位比较结果决定。
比如：若A₃>B₃ 则A>B；若A₃<B₃ 则A<B；若A₃=B₃ 则再比较低位

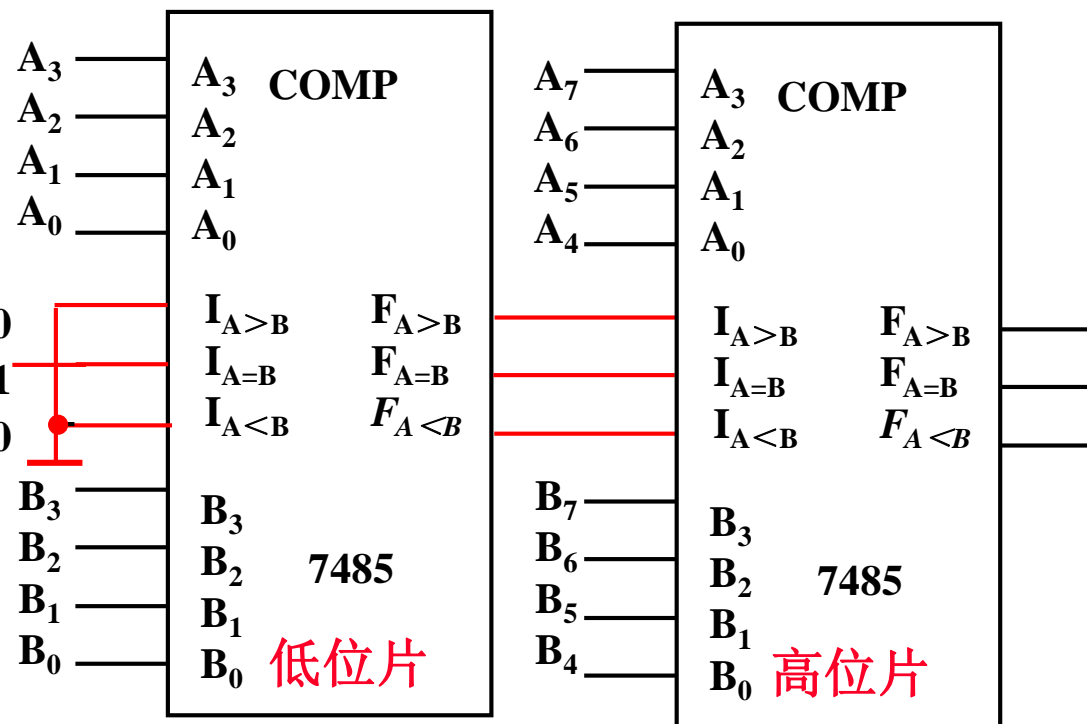
3.2.3 数值比较器

❖ 4位比较器(7485芯片)的使用与扩展

①单片：4位数值比较器



② 2片扩展——8位数值比较器



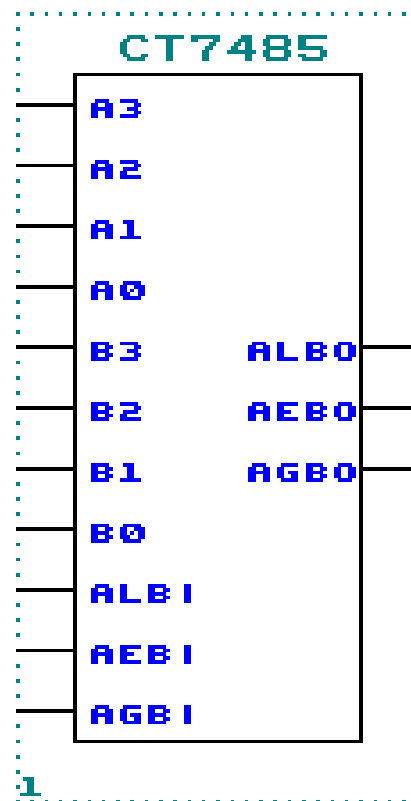
3.2.3 数值比较器

❖ 数值比较器的Verilog HDL设计

- 可以方便地用HDL设计多位数值比较器，而不必用扩展的方法；
- 采用if-else语句

❖ 信号定义

- A3~A0, B3~B0: 两个4位二进制数输入；
- ALBI (即 $I_{A<B}$) : A小于B输入信号；
- AEBI (即 $I_{A=B}$) : A等于B输入信号；
- AGBI (即 $I_{A>B}$) : A大于B输入信号；
- ALBO (即 $F_{A<B}$) : A小于B输出信号；
- AEBO (即 $F_{A=B}$) : A等于B输出信号；
- AGBO (即 $F_{A>B}$) : A大于B输出信号。



3.2.3 数值比较器

❖ 4位比较器的Verilog HDL设计

```
module CT7485(A3,A2,A1,A0,B3,B2,B1,B0,ALBI,AEBI,  
              AGBI,ALBO,AEBO,AGBO);  
  input      A3,A2,A1,A0,B3,B2,B1,B0,ALBI,AEBI,AGBI;  
  output     ALBO,AEBO,AGBO;  
  reg        ALBO,AEBO,AGBO;  
  wire[3:0] A_SIGNAL,B_SIGNAL;  
  assign A_SIGNAL = {A3,A2,A1,A0}; //拼接成4位wire型向量  
  assign B_SIGNAL = {B3,B2,B1,B0}; //拼接成4位wire型向量  
  always  
  begin  
    if (A_SIGNAL > B_SIGNAL)  
      begin ALBO = 0; AEBO = 0; AGBO = 1;end  
    else if (A_SIGNAL < B_SIGNAL)  
      begin ALBO = 1; AEBO = 0; AGBO = 0;end  
    else // if(A_SIGNAL == B_SIGNAL) 可省略  
      begin ALBO = ALBI; AEBO = AEBI; AGBO = AGBI;end  
  end  
endmodule
```

3.2.4 ALU--1位ALU

❖ 可执行1位与、或、或非、与非、加、减运算

➤ AND运算

- Ainvert=0
- Binvert=0
- Operation=0

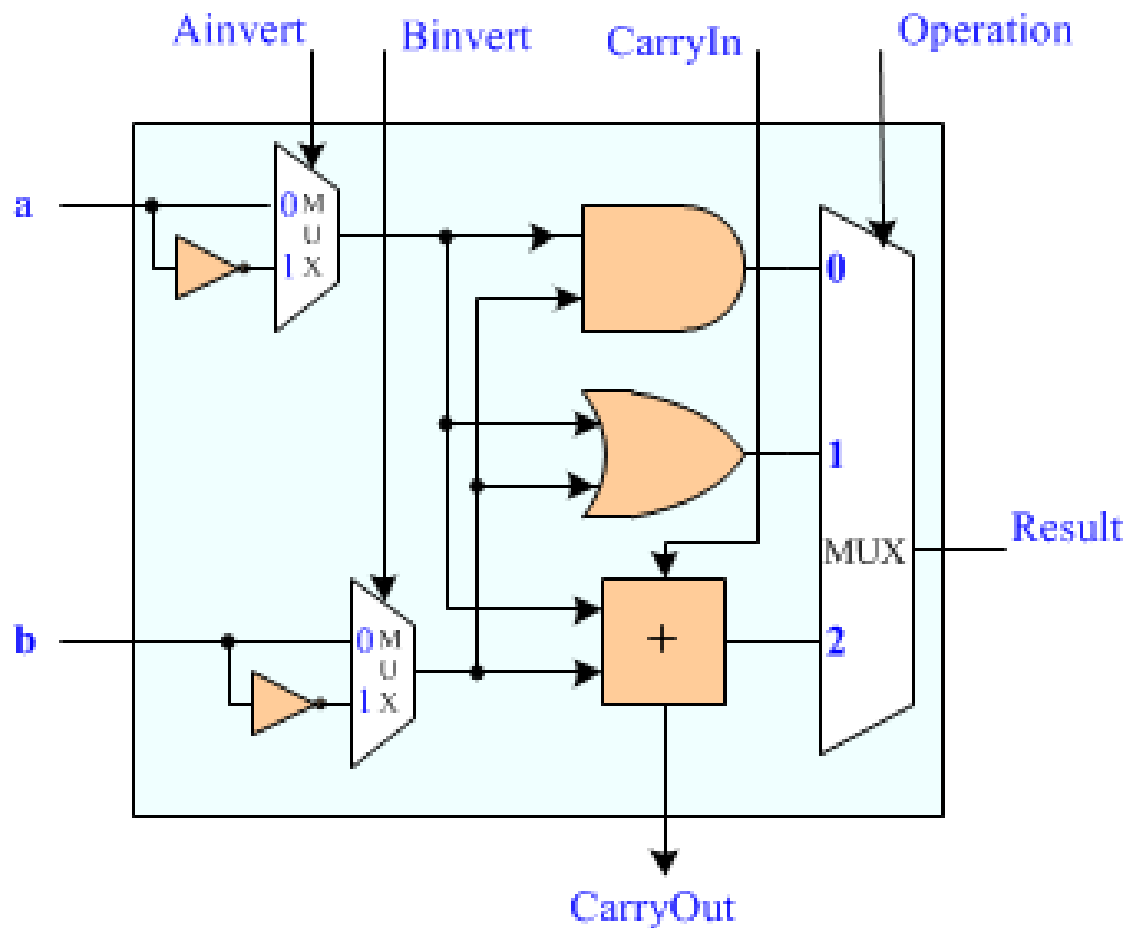
➤ OR运算

- Ainvert=1
- Binvert=1
- Operation=1

➤ SUB运算

- Ainvert=0
- Binvert=1
- CarryIn=1
- Operation=2

➤ . . .



3.2.4 ALU--32位ALU

❖ 加法器采用串行仅为，也可以采用先行进位

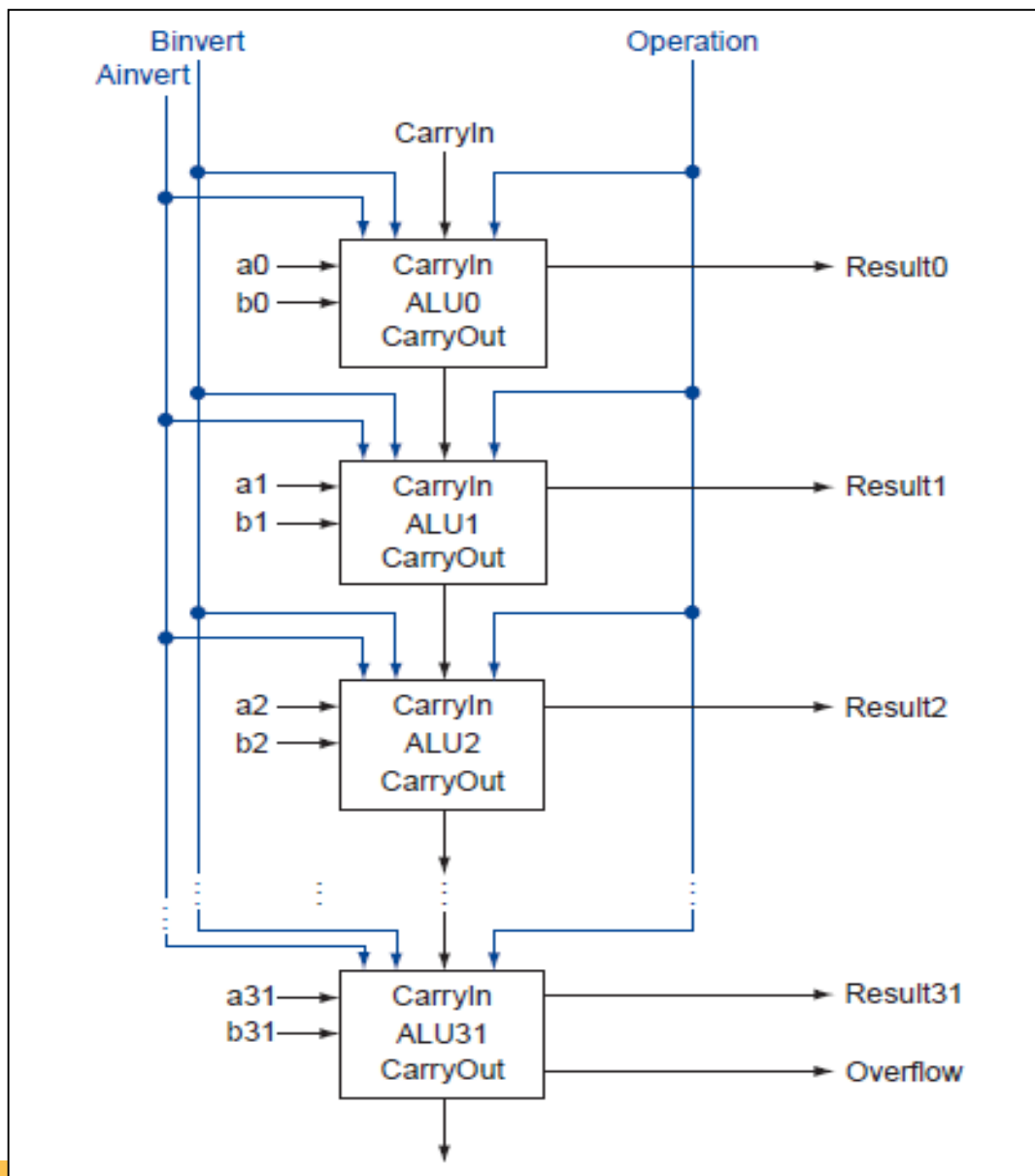
❖ 减法的实现

➢ Ainvert=0

➢ Binvert=1

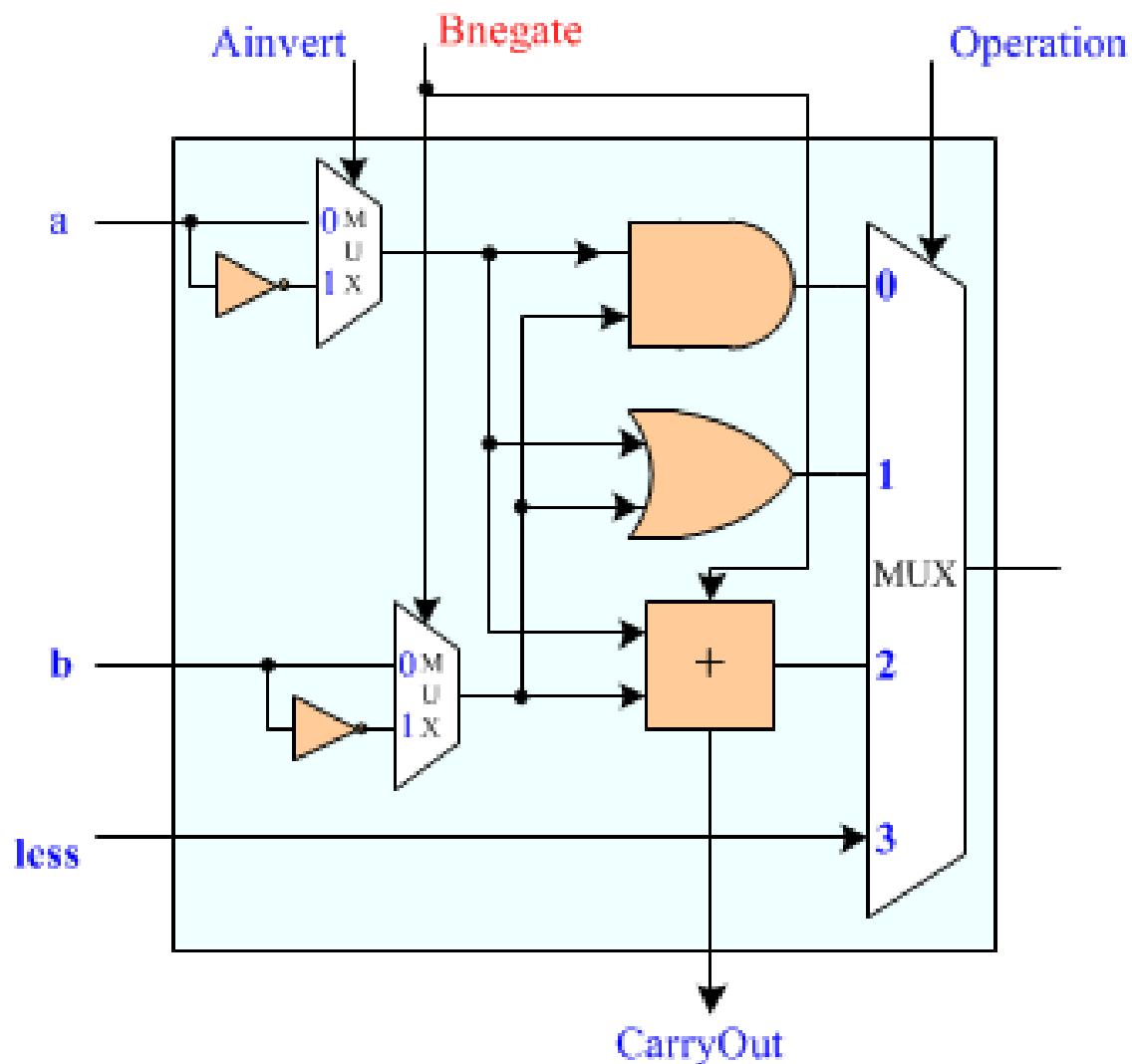
➢ CarryIn=1

❖ Binvert和CarryIn
可以合并成一个信号Bnegate



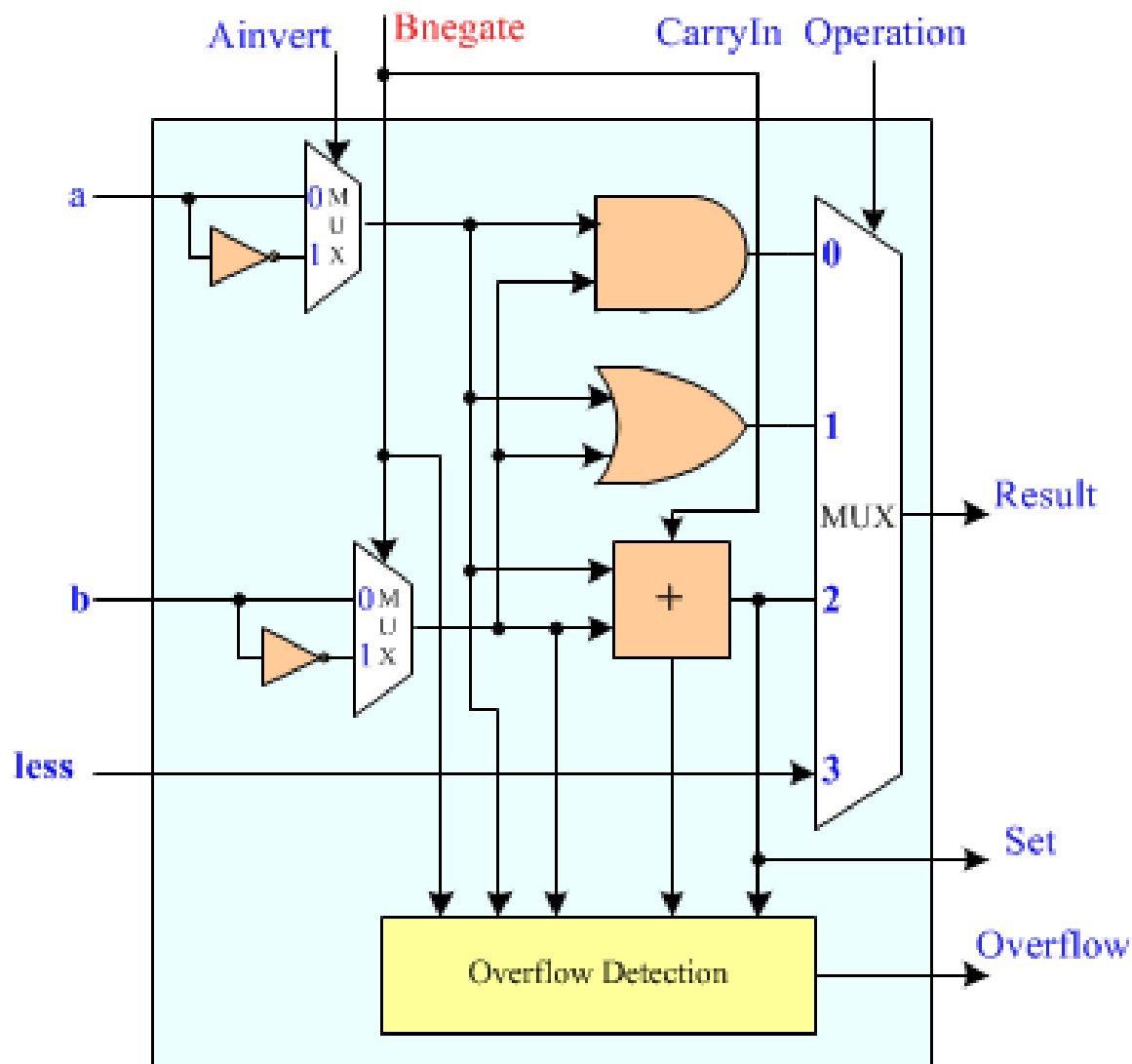
3.2.4 ALU--32位ALU

❖ 为比较功能（**Less**）增加一个输入

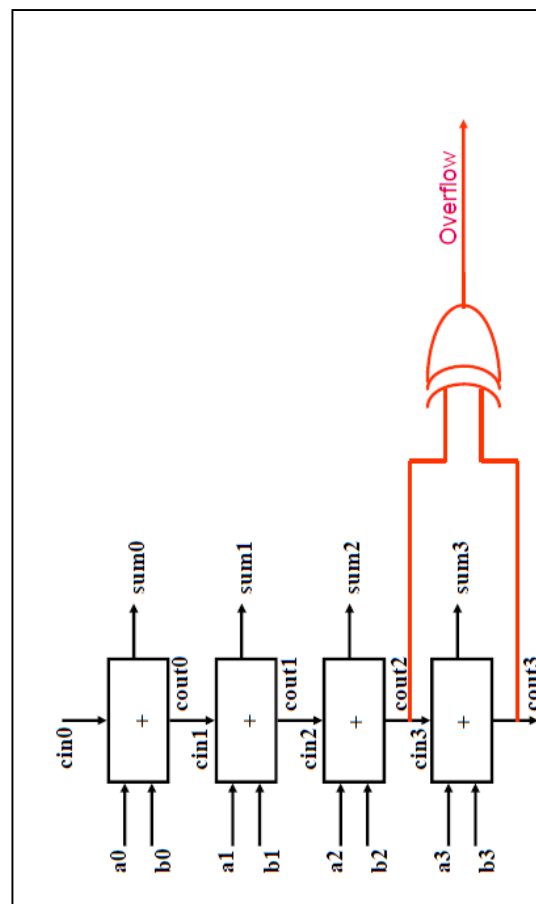


3.2.4 ALU--32位ALU

❖ 最高位（MSB）带溢出判断Overflow

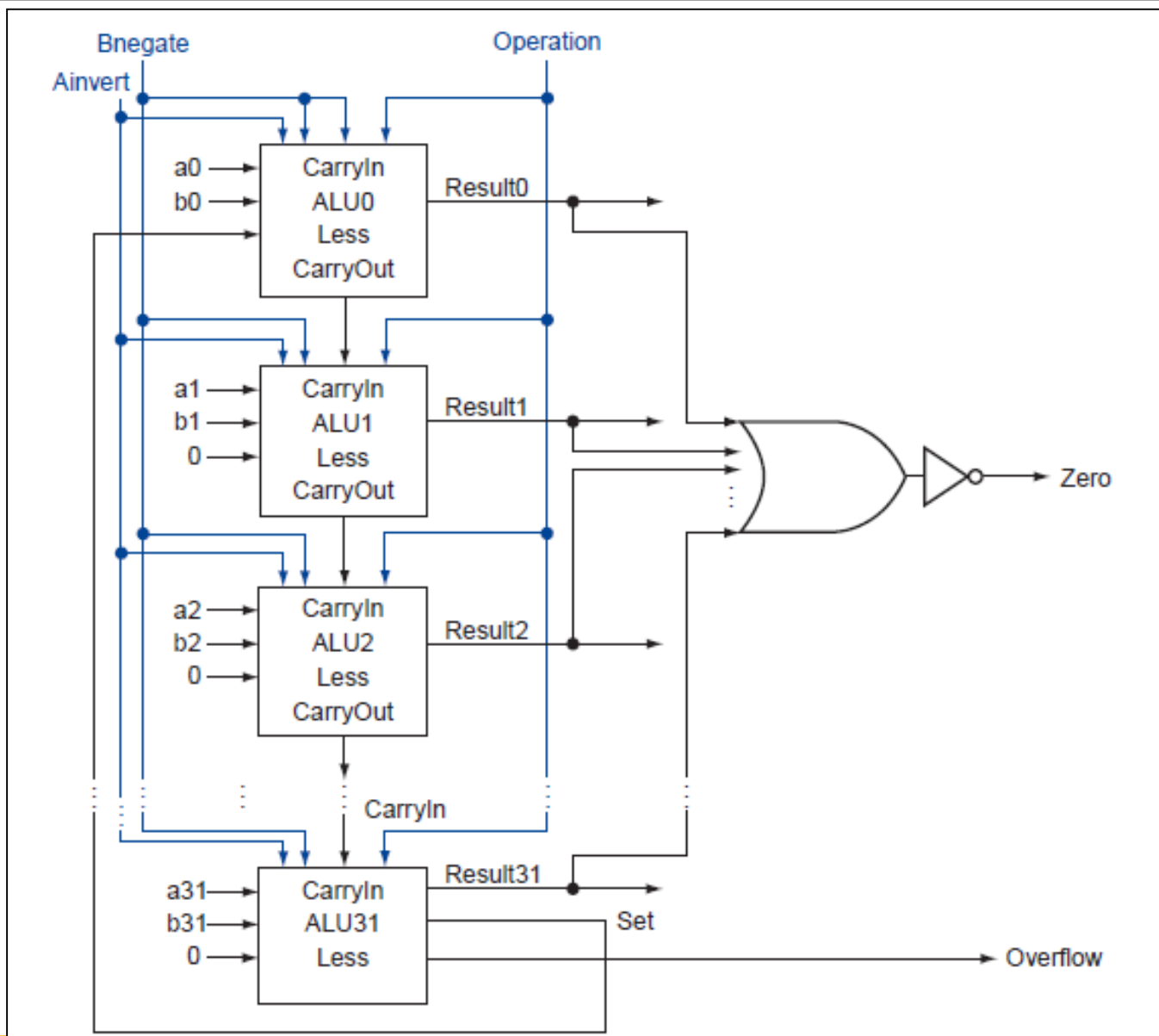


对于4位加法器:
 $\text{Overflow} = \text{Cin3} \text{ XOR } \text{Cout3}$



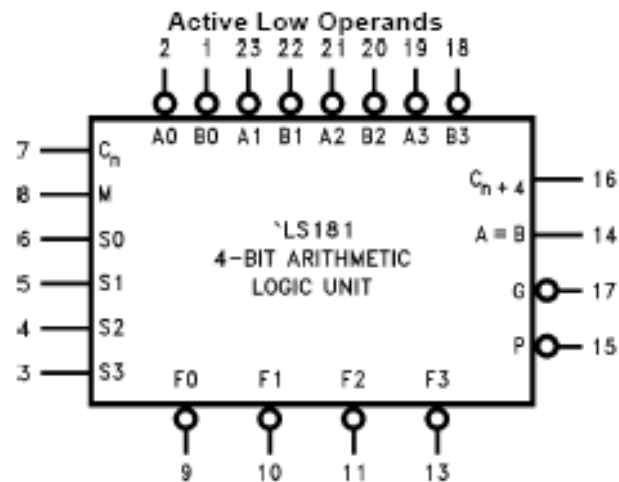
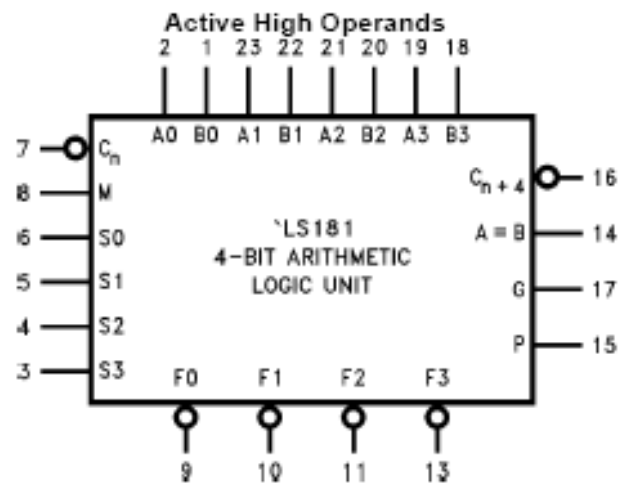
3.2.4 ALU--32位ALU

完整32位 ALU



ALU芯片—DM74LS181N

- 4位ALU
- 提供16种算术逻辑运算
- 两种工作模式：正逻辑和负逻辑



V_{CC} = Pin 24
GND = Pin 12

ALU芯片—DM74LS181N

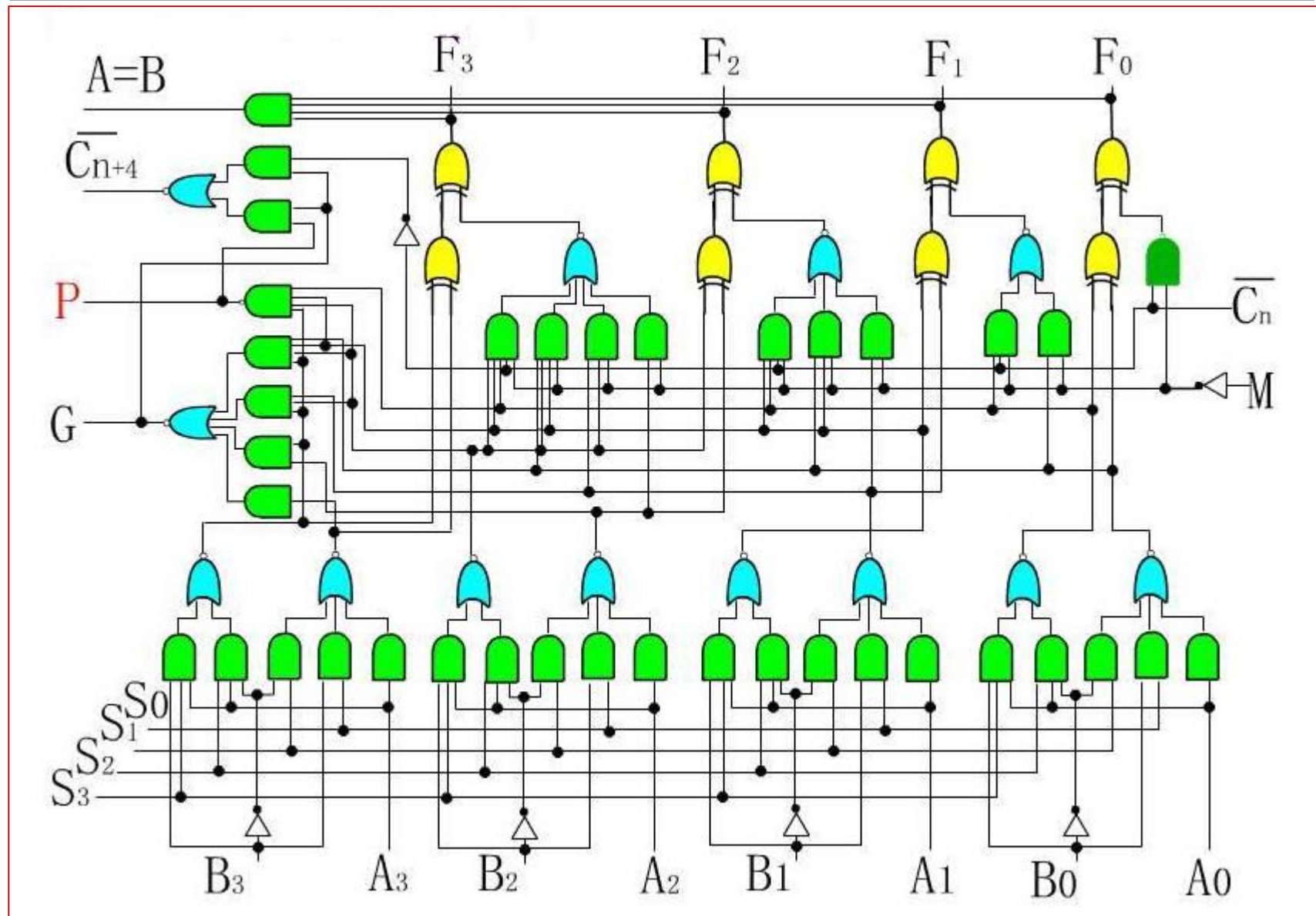
■ Function Table

Function Table							
Mode Select Inputs				Active LOW Operands & F_n Outputs		Active HIGH Operands & F_n Outputs	
				Logic	Arithmetic (Note 2)	Logic	Arithmetic (Note 2)
S3	S2	S1	S0	(M = H)	(M = L) ($C_n = L$)	(M = H)	(M = L) ($C_n = H$)
L	L	L	L	\overline{A}	A minus 1	\overline{A}	A
L	L	L	H	\overline{AB}	AB minus 1	$\overline{A} + \overline{B}$	A + B
L	L	H	L	$\overline{A} + \overline{B}$	\overline{AB} minus 1	$\overline{A} B$	A + \overline{B}
L	L	H	H	Logic 1	minus 1	Logic 0	minus 1
L	H	L	L	$\overline{A} + \overline{B}$	A plus ($A + \overline{B}$)	\overline{AB}	A plus \overline{AB}
L	H	L	H	\overline{B}	AB plus ($A + \overline{B}$)	\overline{B}	(A + B) plus \overline{AB}
L	H	H	L	$\overline{A} \oplus \overline{B}$	A minus B minus 1	$A \oplus B$	A minus B minus 1
L	H	H	H	$A + \overline{B}$	A + \overline{B}	\overline{AB}	AB minus 1
H	L	L	L	$\overline{A} B$	A plus (A + B)	$\overline{A} + B$	A plus AB
H	L	L	H	$A \oplus B$	A plus B	$\overline{A} \oplus \overline{B}$	A plus B
H	L	H	L	B	\overline{AB} plus (A + B)	B	(A + \overline{B}) plus AB
H	L	H	H	A + B	A + B	AB	AB minus 1
H	H	L	L	Logic 0	A plus A (Note 1)	Logic 1	A plus A (Note 1)
H	H	L	H	\overline{AB}	AB plus A	$A + \overline{B}$	(A + B) plus A
H	H	H	L	AB	\overline{AB} minus A	A + B	(A + \overline{B}) plus A
H	H	H	H	A	A	A	A minus 1

Note 1: Each bit is shifted to the next most significant position.

Note 2: Arithmetic operations expressed in 2s complement notation.

ALU芯片—DM74LS181N



第二讲：组合逻辑

一. 逻辑代数基础

1. 逻辑代数基本概念
2. 逻辑代数的公理、定理与规则
3. 逻辑函数的表达式
4. 逻辑函数化简（卡诺图）

二. 逻辑门电路

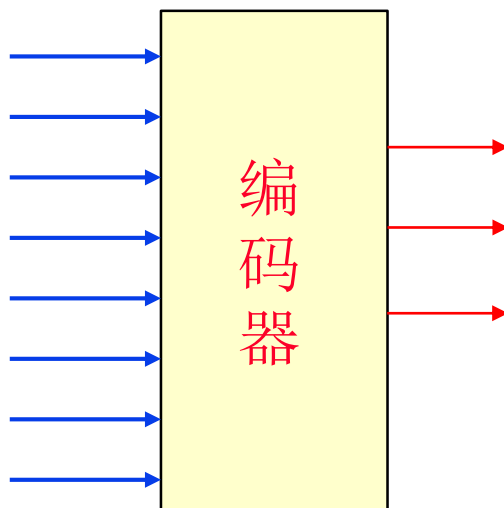
三. 基本组合逻辑部件设计

1. 组合逻辑设计概述
2. 运算单元电路
3. 编码器/译码器
4. 多路选择器

编码器

- ❖ **编码**：为区分一系列不同的事物，对其中的每个事物用一组二值（0或1）代码表示。
- ❖ **编码器**：实现编码功能的数字电路称为**编码器**。
- ❖ 通常有二进制编码器、BCD码编码器及优先编码器。

输入：多个，每一个代表一个不同事物，一般应保证任意时刻只有一个有效，即编码器只对其中一个有效的信号进行编码。



输出：一组二值代码，对应输入信号中有效的那个事物的编码。

二进制编码器

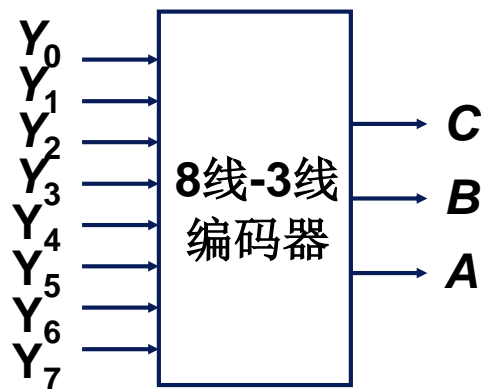
❖ 2^n 线- n 线编码器：用 n 位二进制代码对 2^n 个信号进行编码的电路。

- **输入：** 2^n 个信号
- **输出：** n 位二进制代码
- 任意一时刻只能对其中一个信号进行编码，即**只允许其中一个输入信号有效**（低电平或高电平），而其余信号为无效电平，否则输出将发生混乱。
- **高电平输入有效：**在输入等于1时对输入信号进行编码，则称这类编码器为**高电平输入有效**，此时输出等于对应有效输入的编号的二进制编码；
- **低电平输入有效：**在输入等于0时对输入信号进行编码，则称这类编码器为**低电平输入有效**。

3.3.1 8线-3线编码器

❖ 8线-3线编码器（高电平输入有效）

- 8个输入信号： Y_0, Y_1, \dots, Y_7 。任何时刻最多只有其中一个信号为1（高电平），其余均为0（低电平）。
- 3位输出编码： C, B, A 。
- 当 $Y_0=1$ 时， $CBA=000$ ； $Y_1=1$ 时， $CBA=001$ ； $Y_2=1$ 时， $CBA=010$ ，以此类推。



编码表

输入	CBA
Y_0	0 0 0
Y_1	0 0 1
Y_2	0 1 0
Y_3	0 1 1
Y_4	1 0 0
Y_5	1 0 1
Y_6	1 1 0
Y_7	1 1 1

3.3.1 8线-3线编码器

❖ 8线-3线编码器 (高电平输入有效)

编码器真值表 (高电平输入有效)

1. 利用最小项推导法写出各输出的逻辑函数表达式;

$$C = \bar{Y}_7\bar{Y}_6\bar{Y}_5Y_4\bar{Y}_3\bar{Y}_2\bar{Y}_1\bar{Y}_0 + \bar{Y}_7\bar{Y}_6Y_5\bar{Y}_4\bar{Y}_3\bar{Y}_2\bar{Y}_1\bar{Y}_0 \\ + \bar{Y}_7Y_6\bar{Y}_5\bar{Y}_4\bar{Y}_3\bar{Y}_2\bar{Y}_1\bar{Y}_0 + Y_7\bar{Y}_6\bar{Y}_5\bar{Y}_4\bar{Y}_3\bar{Y}_2\bar{Y}_1\bar{Y}_0$$

2. 利用约束项化简: 任何时刻Y7~Y0中仅有一个为1, 输入变量取值组合仅有8种状态, 其余均为约束项。利用这些约束项化简上式, 得到:

若 $Y_4 = 1$, $\bar{Y}_7\bar{Y}_6\bar{Y}_5\bar{Y}_3\bar{Y}_2\bar{Y}_1\bar{Y}_0 = 1$;

所以有: $C = Y_4 + Y_5 + Y_6 + Y_7$

Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0	C	B	A
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1
0	0	0	0	0	0	1	1	x	x	x
0	0	0	0	0	1	1	1	x	x	x
约束项		
1	1	1	1	1	1	1	1	x	x	x

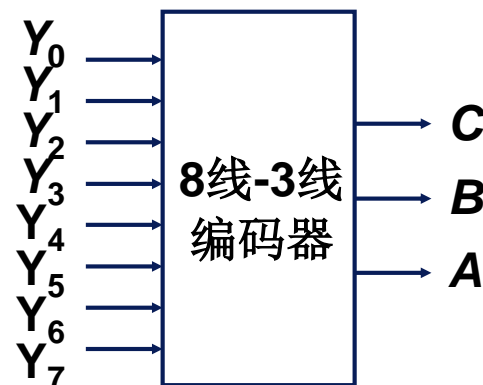
3.3.1 8线-3线编码器

❖ 8线-3线编码器 (高电平输入有效)

$$A = Y_1 + Y_3 + Y_5 + Y_7 = \overline{\overline{Y_1} \cdot \overline{Y_3} \cdot \overline{Y_5} \cdot \overline{Y_7}}$$

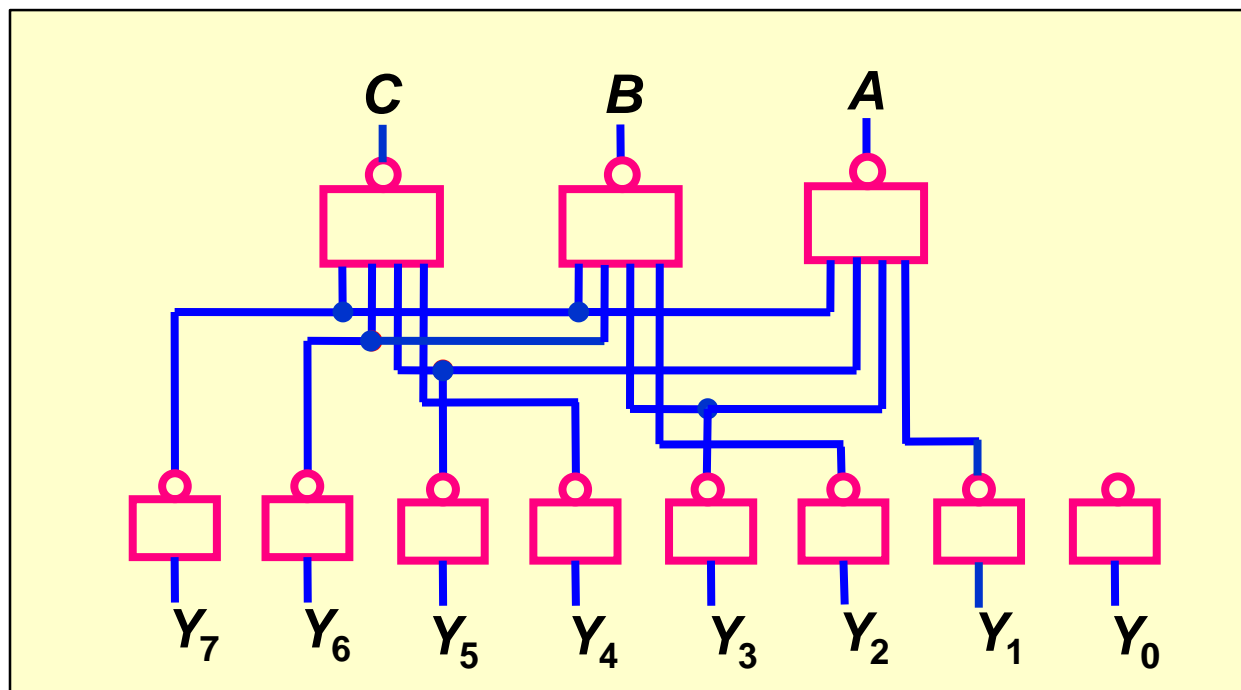
$$B = Y_2 + Y_3 + Y_6 + Y_7 = \overline{\overline{Y_2} \cdot \overline{Y_3} \cdot \overline{Y_6} \cdot \overline{Y_7}}$$

$$C = Y_4 + Y_5 + Y_6 + Y_7 = \overline{\overline{Y_4} \cdot \overline{Y_5} \cdot \overline{Y_6} \cdot \overline{Y_7}}$$



编码表

输入	C	B	A
Y_0	0	0	0
Y_1	0	0	1
Y_2	0	1	0
Y_3	0	1	1
Y_4	1	0	0
Y_5	1	0	1
Y_6	1	1	0
Y_7	1	1	1



3.3.1 8线-3线编码器

❖ 8线-3线编码器（高电平输入有效）的Verilog HDL设计

❖ **方法一**：根据8线-3线编码器的功能列出真值表，由真值表推出输出的逻辑表达式，然后用assign语句建模（算法级描述）

```
module encoder8_3(Y0,Y1,Y2,Y3,Y4,Y5,Y6,Y7, C,B,A);  
  input  Y0,Y1,Y2,Y3,Y4,Y5,Y6,Y7;  
  output C,B,A;  
  assign C =! (!Y4&&!Y5&&!Y6&&!Y7);  
  assign B =! (!Y2&&!Y3&&!Y6&&!Y7);  
  assign A =! (!Y1&&!Y3&&!Y5&&!Y7);  
endmodule
```

3.3.1 8线-3线编码器

❖ 8线-3线编码器（高电平输入有效）的Verilog HDL设计

❖ **方法二**：根据逻辑功能定义，采用case语句直接描述，设计过程更简单

```
module encoder8_3(Y0,Y1,Y2,Y3,Y4,Y5,Y6,Y7, C,B,A);  
    input  Y0, Y1, Y2, Y3, Y4, Y5, Y6, Y7;  
    reg    C, B, A;  
    always  
        case ({Y0,Y1,Y2,Y3,Y4,Y5,Y6,Y7})  
            'b10000000 : {C,B,A} = 0;      'b01000000 : {C,B,A} = 1;  
            'b00100000 : {C,B,A} = 2;      'b00010000 : {C,B,A} = 3;  
            'b00001000 : {C,B,A} = 4;      'b00000100 : {C,B,A} = 5;  
            'b00000010 : {C,B,A} = 6;      'b00000001 : {C,B,A} = 7;  
            default : {C,B,A} = 3'bx;  
        endcase  
endmodule
```

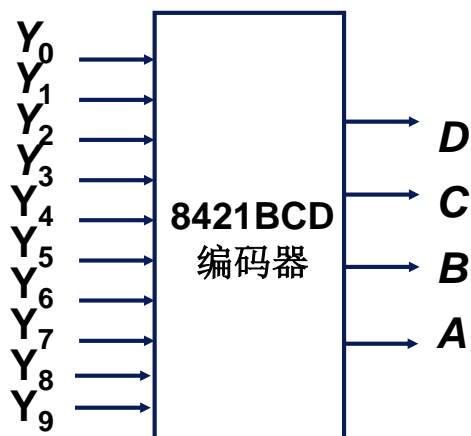
3.3.2 8421BCD码编码器

❖ **8421BCD编码器**：用4位二进制对1十进制数字进行编码。

- 输入：Y0, Y1, ..., Y9, 分别代表十进制数字0~9;
- 输出：4位编码, DCBA=0000, 代表“0”; DCBA=1001, 代表“9”

编码表

输入	DCBA
Y ₀	0 0 0 0
Y ₁	0 0 0 1
Y ₂	0 0 1 0
Y ₃	0 0 1 1
Y ₄	0 1 0 0
Y ₅	0 1 0 1
Y ₆	0 1 1 0
Y ₇	0 1 1 1
Y ₈	1 0 0 0
Y ₉	1 0 0 1



高电平输入有效

$$D = Y_8 + Y_9 = \overline{\overline{Y_8}} \cdot \overline{\overline{Y_9}}$$

$$C = Y_4 + Y_5 + Y_6 + Y_7 = \overline{\overline{Y_4}} \cdot \overline{\overline{Y_5}} \cdot \overline{\overline{Y_6}} \cdot \overline{\overline{Y_7}}$$

$$B = Y_2 + Y_3 + Y_6 + Y_7 = \overline{\overline{Y_2}} \cdot \overline{\overline{Y_3}} \cdot \overline{\overline{Y_6}} \cdot \overline{\overline{Y_7}}$$

$$A = Y_1 + Y_3 + Y_5 + Y_7 + Y_9 = \overline{\overline{Y_1}} \cdot \overline{\overline{Y_3}} \cdot \overline{\overline{Y_5}} \cdot \overline{\overline{Y_7}} \cdot \overline{\overline{Y_9}}$$

3.3.2 8421BCD码编码器

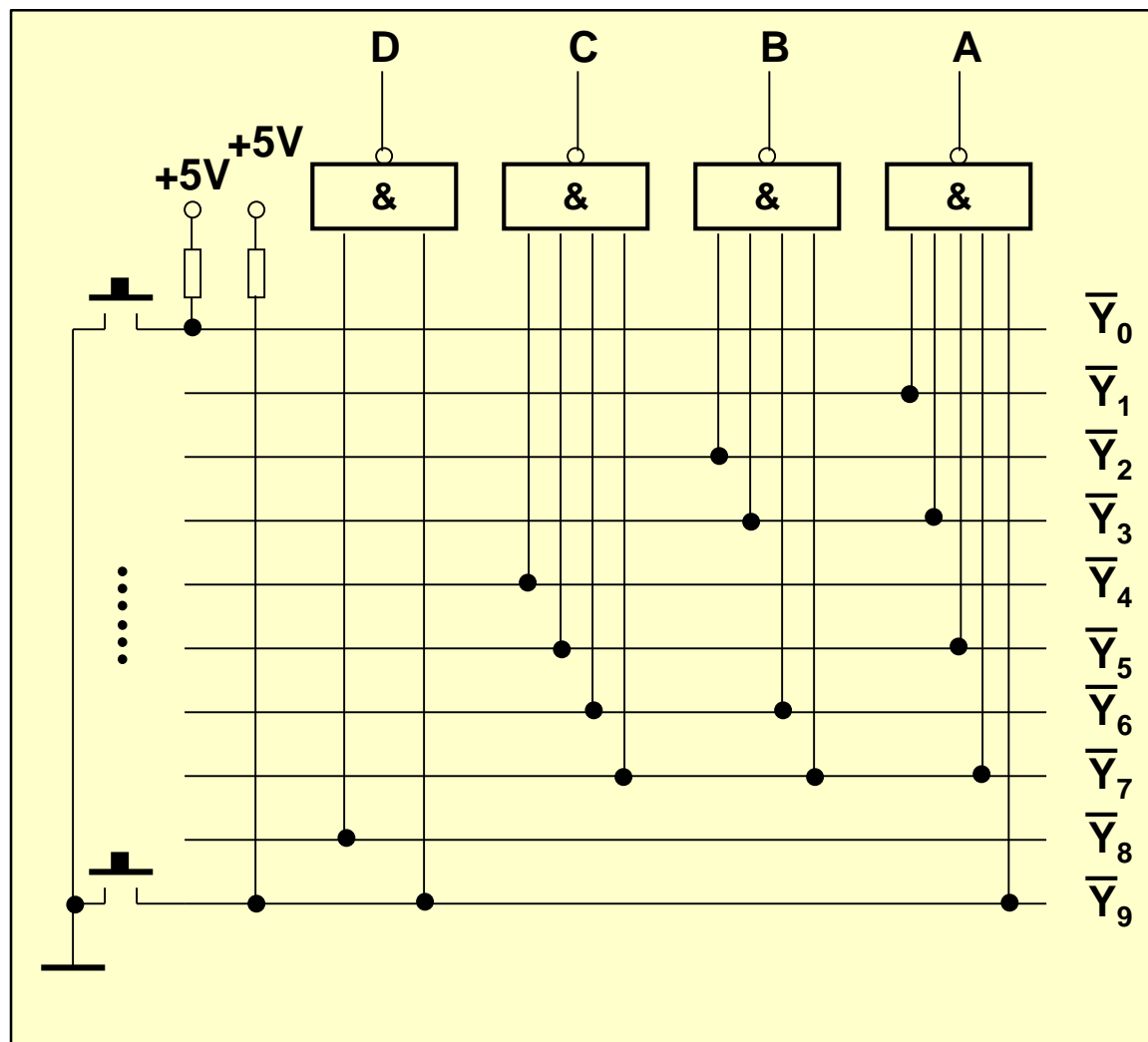
根据逻辑表达式可以直接画出逻辑图

$$D = Y_8 + Y_9 = \overline{\overline{Y_8} \cdot \overline{Y_9}}$$

$$C = Y_4 + Y_5 + Y_6 + Y_7 \\ = \overline{\overline{Y_4} \cdot \overline{Y_5} \cdot \overline{Y_6} \cdot \overline{Y_7}}$$

$$B = Y_2 + Y_3 + Y_6 + Y_7 \\ = \overline{\overline{Y_2} \cdot \overline{Y_3} \cdot \overline{Y_6} \cdot \overline{Y_7}}$$

$$A = Y_1 + Y_3 + Y_5 + Y_7 + Y_9 \\ = \overline{\overline{Y_1} \cdot \overline{Y_3} \cdot \overline{Y_5} \cdot \overline{Y_7} \cdot \overline{Y_9}}$$



3.3.2 8421BCD码编码器

❖ 8421BCD编码器的Verilog HDL设计

```
module bcd8421_3(Y0,Y1,Y2,Y3,Y4,Y5,Y6,Y7,Y8,Y9,D,C,B,A);  
    input          Y0,Y1,Y2,Y3,Y4,Y5,Y6,Y7,Y8,Y9;  
    output D,C,B,A;  
    reg           D,C,B,A;  
    always  
    begin  
        case ({Y9,Y8,Y7,Y6,Y5,Y4,Y3,Y2,Y1,Y0})  
            10'b00_0000_0001: {D,C,B,A} = 0;  10'b00_0000_0010: {D,C,B,A} = 1;  
            10'b00_0000_0100: {D,C,B,A} = 2;  10'b00_0000_1000: {D,C,B,A} = 3;  
            10'b00_0001_0000: {D,C,B,A} = 4;  10'b00_0010_0000: {D,C,B,A} = 5;  
            10'b00_0100_0000: {D,C,B,A} = 6;  10'b00_1000_0000: {D,C,B,A} = 7;  
            10'b01_0000_0000: {D,C,B,A} = 8;  10'b10_0000_0000: {D,C,B,A} = 9;  
            default : {D,C,B,A} = 4'bxxxx;  
        endcase  
    end  
endmodule
```


3.3.3 优先编码器

❖ 优先编码器

- 克服二进制编码器仅允许1个输入信号有限的局限，允许两个以上输入信号同时有效；
- 对所有输入信号进行优先级别排序，任何时刻只对优先级最高的输入信号编码，对优先级别低的输入信号则不响应，以保证编码器可靠工作。
- 优点：当有两个或两个以上的输入有效时，输出不会发生混乱。广泛应用于计算机的优先中断系统、键盘编码系统中。

❖ **74LS148**：8线-3线优先编码器，8个输入信号，低电平有效；3个输出端，反码输出

❖ **74LS147**：10线-4线优先编码器，10个输入信号，低电平有效；4个输出端，反码输出

3.3.3 优先编码器--74147的设计

❖ 74147优先编码器 (低电平输入有效)

输入： $\overline{I}_0 \sim \overline{I}_9$, \overline{I}_9 优先级最高, \overline{I}_0 优先级最低

输出： $\overline{Y}_3 \sim \overline{Y}_0$, 当 $\overline{I}_9 = 0$ (有效)时, $\overline{Y}_3\overline{Y}_2\overline{Y}_1\overline{Y}_0 = 0110$ (“9”的BCD码之反码)

低电平
输入有
效

\overline{I}_9	\overline{I}_8	\overline{I}_7	\overline{I}_6	\overline{I}_5	\overline{I}_4	\overline{I}_3	\overline{I}_2	\overline{I}_1	\overline{I}_0	\overline{Y}_3	\overline{Y}_2	\overline{Y}_1	\overline{Y}_0
0	x	x	x	x	x	x	x	x	x	0	1	1	0
1	0	x	x	x	x	x	x	x	x	0	1	1	1
1	1	0	x	x	x	x	x	x	x	1	0	0	0
1	1	1	0	x	x	x	x	x	x	1	0	0	1
1	1	1	1	0	x	x	x	x	x	1	0	1	0
1	1	1	1	1	0	x	x	x	x	1	0	1	1
1	1	1	1	1	1	0	x	x	x	1	1	0	0
1	1	1	1	1	1	1	0	x	x	1	1	0	1
1	1	1	1	1	1	1	1	0	x	1	1	1	0
1	1	1	1	1	1	1	1	1	0	1	1	1	1

输出为优先
级最高的输
入信号对应
编号的反码

3.3.3 优先编码器--74147的Verilog HDL

Module

CT74147(IN0,IN1,IN2,IN3,IN4,IN5,IN6,IN7,IN8,IN9,YN0,YN1,YN2,YN3);

input IN0,IN1,IN2,IN3, IN4, IN5,IN6,IN7,IN8,IN9;

output YN0,YN1,YN2,YN3;

reg YN0,YN1,YN2,YN3;

reg[3:0] Y; //中间变量

always @(IN0 or IN1 or IN2 or IN3 or IN4 or IN5 or IN6 or IN7 or IN8 or IN9)

begin

if (IN9 == 1'b0) Y = 4'b0110;

else if (IN8 == 1'b0) Y = 4'b0111;

else if (IN7 == 1'b0) Y = 4'b1000;

else if (IN6 == 1'b0) Y = 4'b1001;

.....

else if (IN1 == 1'b0) Y = 4'b1110;

else if (IN0 == 1'b0) Y = 4'b1111;

YN0 = Y[0];

YN1 = Y[1];

YN2 = Y[2];

YN3 = Y[3];

end

endmodule

3.3.4 译码器

- ❖ **译码器**：将二进制代码所表示的信息翻译成对应高低电平信号输出的过程称为**译码**，译码是编码的反操作。实现译码功能的电路称为**译码器（Decoder）**。
- **变量译码器（二进制译码器）**：实现输入变量状态全部组合的译码器，一般称为 **n 线- 2^n 线译码器**。如2线-4线译码器，3线-8线译码器等。
 - **码制变换译码器**：将输入的某个进制代码转换成对应的其他码制输出的译码器。如8421码至十进制码译码器（简称**BCD译码器**）、余3码至十进制码译码器等。
 - **显示译码器**：将输入代码转换成驱动7段数码显示器各段电平信号的译码器。常用的有74××47（低电平输出有效）、74××49（高电平输出有效）、74××48（高电平输出有效）等。

3.3.4 译码器

❖ **二进制译码器**：将每个输入二进制代码译成对应的一根输出线上的高电平（或低电平）信号。因此称为 **n 线- 2^n 线译码器**。

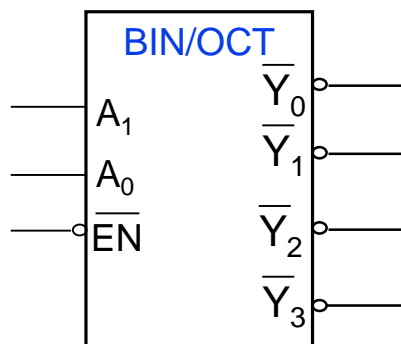
- 二进制译码器功能与二进制编码器正好相反，它是将具有特定含义的不同二进制代码辨别出来，并转换成相应的电平信号。
- **任何时刻最多只允许1个输出有效**（与编码器对应，任何时刻只允许有一个输入为有效电平）；
- 高电平输出有效时，每个输出都是对应的输入变量最小项；低电平输出有效时，每个输出都是对应的输入变量最小项的反，所以二进制译码器也称为**最小项译码器**。

3.3.4 译码器（2线-4线译码器）

❖ 2线-4线译码器（74139）

- 2个输入： A_1, A_0 ；00,01,10,11共4种输入组合。
- 4个输出： Y_3, Y_2, Y_1, Y_0 ，低电平输出有效；任何时刻最多只有一个输出有效。当输入为00时， Y_0 输出有效；当输入为01时， Y_1 输出有效；以此类推。
- 1个使能控制： EN 为使能控制输入，低电平有效。为0时，译码器处于工作状态，为1时，译码器禁止工作。

功能表



\overline{EN}	A_1	A_0	\overline{Y}_3	\overline{Y}_2	\overline{Y}_1	\overline{Y}_0
1	X	X	1	1	1	1
0	0	0	1	1	1	0
0	0	1	1	1	0	1
0	1	0	1	0	1	1
0	1	1	0	1	1	1

当低电平输出有效时，每个输出都是对应的输入变量最小项的反。

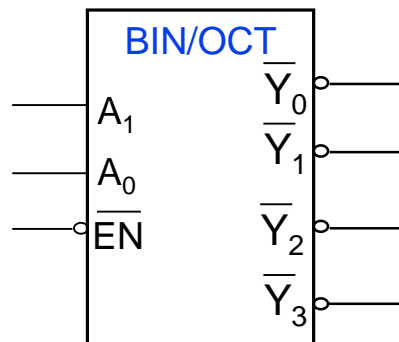
3.3.4 译码器（2线-4线译码器）

❖ 2线-4线译码器（74139）

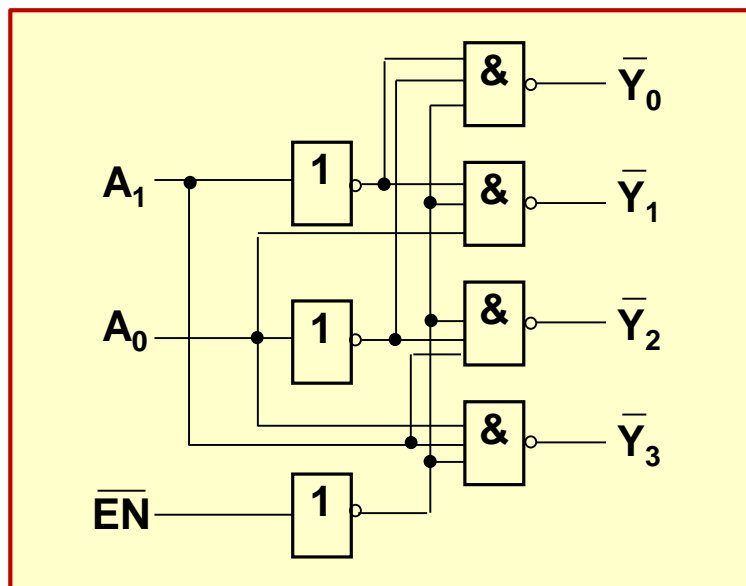
1. 根据真值表推导出表达式（最大项推导）

$$\bar{Y}_0 = A_1 + A_0 = \overline{\bar{A}_1 \bar{A}_0} \quad \bar{Y}_1 = A_1 + \bar{A}_0 = \overline{\bar{A}_1 A_0}$$

$$\bar{Y}_2 = \bar{A}_1 + A_0 = \overline{A_1 \bar{A}_0} \quad \bar{Y}_3 = \bar{A}_1 + \bar{A}_0 = \overline{A_1 A_0}$$



2. 根据逻辑表达式画出逻辑图



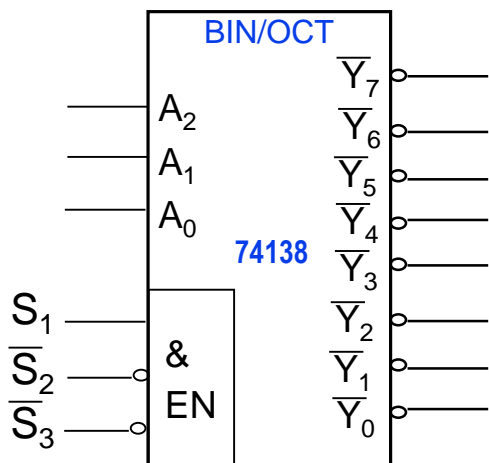
74139译码器真值表

EN	A ₁	A ₀	Y ₃	Y ₂	Y ₁	Y ₀
1	X	X	1	1	1	1
0	0	0	1	1	1	0
0	0	1	1	1	0	1
0	1	0	1	0	1	1
0	1	1	0	1	1	1

3.3.4 译码器（3线-8线译码器）

❖ 3线-8线译码器（74138）

- 3个输入：A₂，A₁，A₀；000~111共8种输入组合。
- 8个输出：Y₇~Y₀，**低电平输出有效**；任何时刻最多只有一个输出有效。当输入为000时，Y₀输出有效；当输入为001时，Y₁输出有效。
- 3个使能控制：S₀，S₁，S₂ 为使能输入，仅当它们分别为1、0、0时，译码器才正常译码；否则禁止工作。



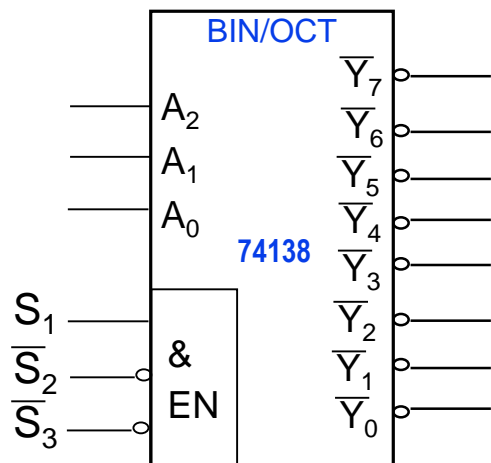
功能表

$S_1 \overline{S_2} \overline{S_3}$	A ₂ A ₁ A ₀	$\overline{Y_7} \overline{Y_6} \overline{Y_5} \overline{Y_4} \overline{Y_3} \overline{Y_2} \overline{Y_1} \overline{Y_0}$
≠100	X X X	1 1 1 1 1 1 1 1
=100	0 0 0	1 1 1 1 1 1 1 0
=100	0 0 1	1 1 1 1 1 1 0 1
=100	0 1 0	1 1 1 1 1 0 1 1
=100	0 1 1	1 1 1 1 0 1 1 1
=100	1 0 0	1 1 1 0 1 1 1 1
=100	1 0 1	1 1 0 1 1 1 1 1
=100	1 1 0	1 0 1 1 1 1 1 1
=100	1 1 1	0 1 1 1 1 1 1 1

3.3.4 译码器（3线-8线译码器）

❖ 3线-8线译码器（74138）

74138真值表



$S_1 \overline{S_2} \overline{S_3}$	$A_2 \ A_1 \ A_0$	$\overline{Y_7} \ \overline{Y_6} \ \overline{Y_5} \ \overline{Y_4} \ \overline{Y_3} \ \overline{Y_2} \ \overline{Y_1} \ \overline{Y_0}$
$\neq 100$	X X X	1 1 1 1 1 1 1 1
$=100$	0 0 0	1 1 1 1 1 1 1 0
$=100$	0 0 1	1 1 1 1 1 1 0 1
$=100$	0 1 0	1 1 1 1 1 0 1 1
$=100$	0 1 1	1 1 1 1 0 1 1 1
$=100$	1 0 0	1 1 1 0 1 1 1 1
$=100$	1 0 1	1 1 0 1 1 1 1 1
$=100$	1 1 0	1 0 1 1 1 1 1 1
$=100$	1 1 1	0 1 1 1 1 1 1 1

根据真值表推导出表达式（最大项推导法）

$$\overline{Y_0} = A_2 + A_1 + A_0 = \overline{\overline{A_2} \overline{A_1} \overline{A_0}}$$

$$\overline{Y_1} = A_2 + A_1 + \overline{A_0} = \overline{\overline{A_2} \overline{A_1} A_0}$$

$$\overline{Y_2} = A_2 + \overline{A_1} + A_0 = \overline{\overline{A_2} A_1 \overline{A_0}}$$

$$\overline{Y_3} = A_2 + \overline{A_1} + \overline{A_0} = \overline{\overline{A_2} A_1 A_0}$$

$$\overline{Y_4} = \overline{A_2} + A_1 + A_0 = \overline{\overline{A_2} \overline{A_1} \overline{A_0}}$$

$$\overline{Y_5} = \overline{A_2} + A_1 + \overline{A_0} = \overline{\overline{A_2} \overline{A_1} A_0}$$

$$\overline{Y_6} = \overline{A_2} + \overline{A_1} + A_0 = \overline{\overline{A_2} A_1 \overline{A_0}}$$

$$\overline{Y_7} = \overline{A_2} + \overline{A_1} + \overline{A_0} = \overline{\overline{A_2} A_1 A_0}$$

3.3.4 译码器（3线-8线译码器Verilog HDL设计）

```
module CT74138(A0,A1,A2,S1,S2N,S3N,YN0,
               YN1,YN2,YN3,YN4,YN5,YN6,YN7);
    input      A0,A1,A2,S1,S2N,S3N;
    output     YN0,YN1,YN2,YN3,YN4,YN5, YN6,YN7;
    reg        YN0,YN1,YN2,YN3,YN4,YN5, YN6,YN7;
    reg[7:0]   Y_SIGNAL
```

always
begin

```
    if (S1 && !S2N && !S3N==1)
```

begin

```
        case ({A2,A1,A0})
```

```
            3 'b000 : Y_SIGNAL =8 'b11111110;
```

```
            3 'b001 : Y_SIGNAL =8 'b11111101;
```

```
            3 'b010 : Y_SIGNAL =8 'b11111011;
```

```
            3 'b011 : Y_SIGNAL =8 'b11110111;
```

```
            3 'b100 : Y_SIGNAL =8 'b11101111;
```

```
            3 'b101 : Y_SIGNAL =8 'b11011111;
```

```
            3 'b110 : Y_SIGNAL =8 'b10111111;
```

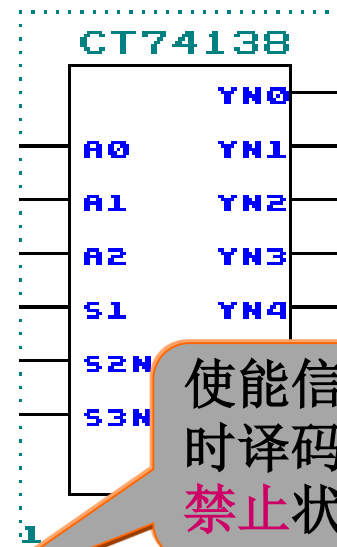
```
            3 'b111 : Y_SIGNAL =8 'b01111111;
```

```
            default : Y_SIGNAL =8'b11111111;
```

endcase

end

使能信号均有效时译码器处于工作状态



使能信号无效时译码器处于禁止状态

```
    else Y_SIGNAL = 8'b11111111;
```

```
    YN0 = Y_SIGNAL[0];
```

```
    YN1 = Y_SIGNAL[1];
```

```
    YN2 = Y_SIGNAL[2];
```

```
    YN3 = Y_SIGNAL[3];
```

```
    YN4 = Y_SIGNAL[4];
```

```
    YN5 = Y_SIGNAL[5];
```

```
    YN6 = Y_SIGNAL[6];
```

```
    YN7 = Y_SIGNAL[7];
```

end

endmodule

3.3.4 译码器（BCD译码器）

❖ **码制变换译码器**：将输入的二进制代码转换成对应的其他码制输出的译码器。

➤ **BCD译码器（4-10线译码器）**：4位8421码转换成十进制数的电路。用于驱动十进制数码显示管、指示灯等

■ **完全译码BCD译码器**：当输入出现伪码0101~1111时，译码器输出 $Y_0 \sim Y_9$ 均为“1”，如74xx42（输出为低电平有效）

■ **不完全译码BCD译码器**：当输入出现伪码0101~1111时， $Y_0 \sim Y_9$ 均为任意值

➤ 余3码至十进制码译码器

➤ 余3循环码至十进制码译码器

3.3.4 译码器（BCD译码器74xx42）

❖ BCD译码器（74XX42）

- 4个输入：D，C，B，A。
- 10个输出：Y9~Y0，**低电平有效**；
- DCBA=0000，Y0输出有效；
DCBA=0001，Y1输出有效，
一次类推。
- DCBA=1010~1111时，
Y9~Y0均为“1”

真值表

	D C B A	Y9	Y8	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
0	0 0 0 0	1	1	1	1	1	1	1	1	1	0
1	0 0 0 1	1	1	1	1	1	1	1	1	0	1
2	0 0 1 0	1	1	1	1	1	1	1	0	1	1
3	0 0 1 1	1	1	1	1	1	1	0	1	1	1
4	0 1 0 0	1	1	1	1	1	0	1	1	1	1
5	0 1 0 1	1	1	1	1	0	1	1	1	1	1
6	0 1 1 0	1	1	1	0	1	1	1	1	1	1
7	0 1 1 1	1	1	0	1	1	1	1	1	1	1
8	1 0 0 0	1	0	1	1	1	1	1	1	1	1
9	1 0 0 1	0	1	1	1	1	1	1	1	1	1
不 用	1 0 1 0	1	1	1	1	1	1	1	1	1	1
	1 0 1 1	1	1	1	1	1	1	1	1	1	1
	1 1 0 0	1	1	1	1	1	1	1	1	1	1
	1 1 0 1	1	1	1	1	1	1	1	1	1	1
	1 1 1 0	1	1	1	1	1	1	1	1	1	1
	1 1 1 1	1	1	1	1	1	1	1	1	1	1

3.3.4 译码器（BCD译码器74xx42的Verilog HDL）

1. 根据真值表推导出表达式（最大项推导法）

$$Y_0 = D + C + B + A = \overline{\overline{D}\overline{C}\overline{B}\overline{A}}$$

$$Y_2 = D + C + \overline{B} + A = \overline{\overline{D}\overline{C}\overline{B}A}$$

$$Y_4 = D + \overline{C} + B + A = \overline{\overline{D}C\overline{B}A}$$

$$Y_6 = D + \overline{C} + \overline{B} + A = \overline{\overline{D}CBA}$$

$$Y_8 = \overline{D} + C + B + A = \overline{\overline{D}\overline{C}\overline{B}\overline{A}}$$

$$Y_1 = D + C + B + \overline{A} = \overline{\overline{D}\overline{C}\overline{B}A}$$

$$Y_3 = D + C + \overline{B} + \overline{A} = \overline{\overline{D}\overline{C}BA}$$

$$Y_5 = D + \overline{C} + B + \overline{A} = \overline{\overline{D}C\overline{B}A}$$

$$Y_7 = D + \overline{C} + \overline{B} + \overline{A} = \overline{\overline{D}CBA}$$

$$Y_9 = \overline{D} + C + B + \overline{A} = \overline{\overline{D}\overline{C}\overline{B}\overline{A}}$$

3.3.4 译码器（BCD译码器74xx42的Verilog HDL）

2. 直接用HDL来描述功能（case语句或assign语句）

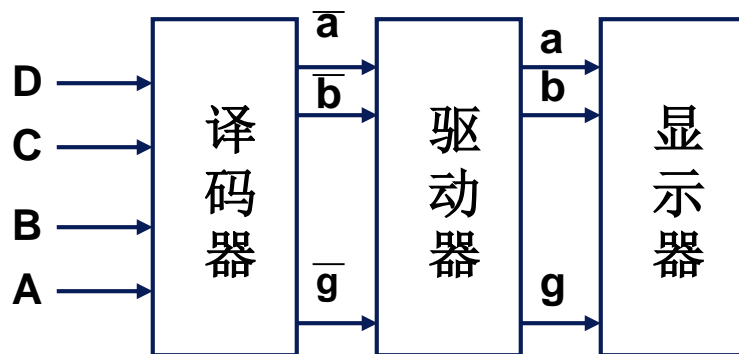
```
module CT7442(D,C,B,A, YN0, YN1,YN2,  
             YN3,YN4,YN5,YN6,YN7,YN8,YN9);  
    input     D,C,B,A;  
    output    YN0,YN1,YN2,YN3,YN4,YN5,  
             YN6,YN7,YN8,YN9;  
    reg       YN0,YN1,YN2,YN3,YN4,YN5,  
             YN6,YN7,YN8,YN9;  
    reg[9:0]  Y_SIGNAL;  
    always  
    begin  
        case ({D,C,B,A})  
            'b0000 : Y_SIGNAL = 'b1111111110;  
            'b0001 : Y_SIGNAL = 'b1111111101;  
            'b0010 : Y_SIGNAL = 'b1111111011;  
            .....  
            'b1000 : Y_SIGNAL = 'b1011111111;  
            'b1001 : Y_SIGNAL = 'b0111111111;  
            default : Y_SIGNAL = 'b1111111111;  
        endcase  
    end
```

```
YN0 = Y_SIGNAL[0];  
YN1 = Y_SIGNAL[1];  
YN2 = Y_SIGNAL[2];  
YN3 = Y_SIGNAL[3];  
YN4 = Y_SIGNAL[4];  
YN5 = Y_SIGNAL[5];  
YN6 = Y_SIGNAL[6];  
YN7 = Y_SIGNAL[7];  
YN8 = Y_SIGNAL[8];  
YN9 = Y_SIGNAL[9];  
end  
endmodule
```

3.3.4 译码器（显示译码器）

❖ **显示译码器：**用于驱动数码显示器，是一种将二进制代码表示的数字、文字、符号用人们习惯的形式直观显示出来的电路。

(1) 电路结构（**8421BCD**译码显示电路）

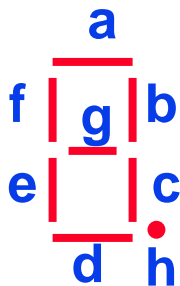


显示器件

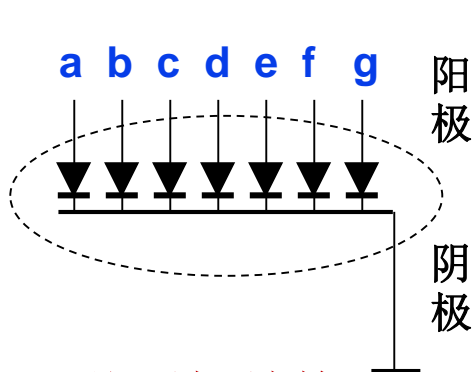
- 辉光数码管、7段荧光数码管、液晶显示器
- 目前广泛使用的显示数字的器件是7段数码显示器（由7段可发光的线段拼合而成），包括半导体数码显示器和液晶显示器两种。
- 数码显示器人们通常又称为数码管。半导体7段数码管实际上是由7个发光二极管（LED）构成的，因此又称为LED数码管。

3.3.4 译码器（显示译码器）

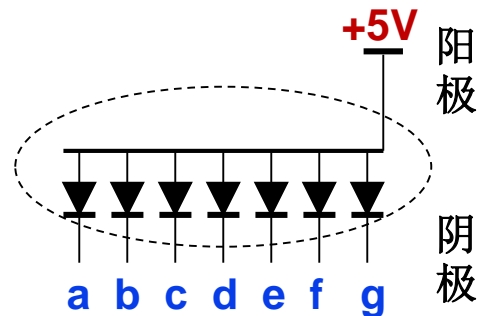
① 半导体7端数码管



半导体7段数码管



共阴极结构



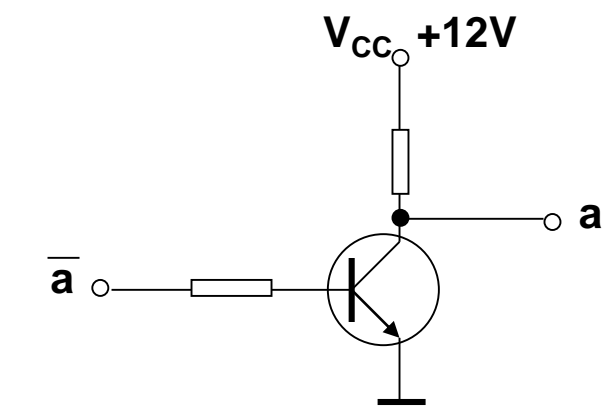
共阳极结构

- 若使用**共阴极LED**数码管，则显示译码器的输出应为**高电平输出有效**；若使用**共阳极LED**数码管，则译码器应为**低电平输出有效**。

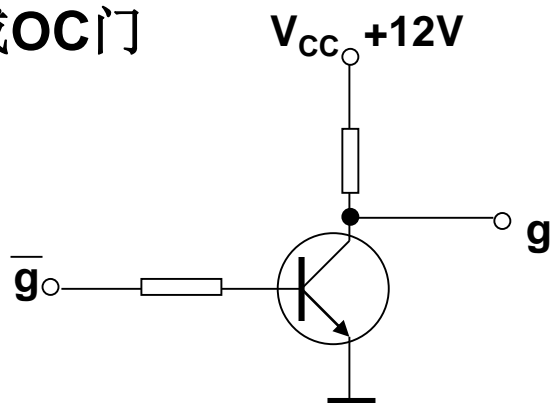
3.3.4 译码器（显示译码器）

③ 驱动电路

使译码器输出反相，并提供大的驱动电流。

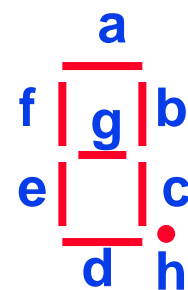


反相器
或OC门



② 译码器（共阳器件）

D C B A	\bar{a} \bar{b} \bar{c} \bar{d} \bar{e} \bar{f} \bar{g}	显示数字
0 0 0 0	0 0 0 0 0 0 1	0
0 0 0 1	1 0 0 1 1 1 1	1
0 0 1 0	0 0 1 0 0 1 0	2
0 0 1 1	0 0 0 0 1 1 0	3
0 1 0 0	1 0 0 1 1 0 0	4
0 1 0 1	0 1 0 0 1 0 0	5
0 1 1 0	1 1 0 0 0 0 0	6
0 1 1 1	0 0 0 1 1 1 1	7
1 0 0 0	0 0 0 0 0 0 0	8
1 0 0 1	0 0 0 1 1 0 0	9
1 0 1 0	X X X X X X X	
1 0 1 1	X X X X X X X	
1 1 0 0	X X X X X X X	
1 1 0 1	X X X X X X X	
1 1 1 0	X X X X X X X	
1 1 1 1	X X X X X X X	



第二讲：组合逻辑

一. 逻辑代数基础

1. 逻辑代数基本概念
2. 逻辑代数的公理、定理与规则
3. 逻辑函数的表达式
4. 逻辑函数化简（卡诺图）

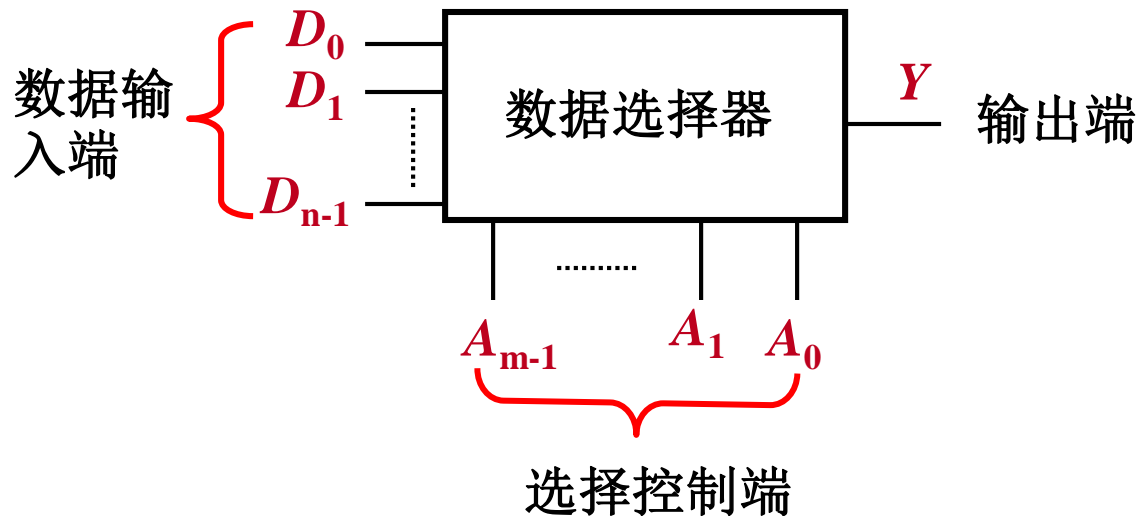
二. 逻辑门电路

三. 基本组合逻辑部件设计

1. 组合逻辑设计概述
2. 运算单元电路
3. 编码器/译码器
4. 多路选择器

3.4.1 多路选择器

- ❖ **数据选择器**：从一组输入数据选出其中一个作为数据输出的电路，又称**多路选择器**（**Multiplexer**，多路器）。
 - 以“与或非”门或以“与或”门为主体，在选择控制信号的作用下，能从多路平行输入数据中任选一路数据作为输出。
 - 常用的集成数据选择器有**2选1**（74××157）、**4选1**（74××153）、**8选1**（74××151）及**16选1**（74××150）等。

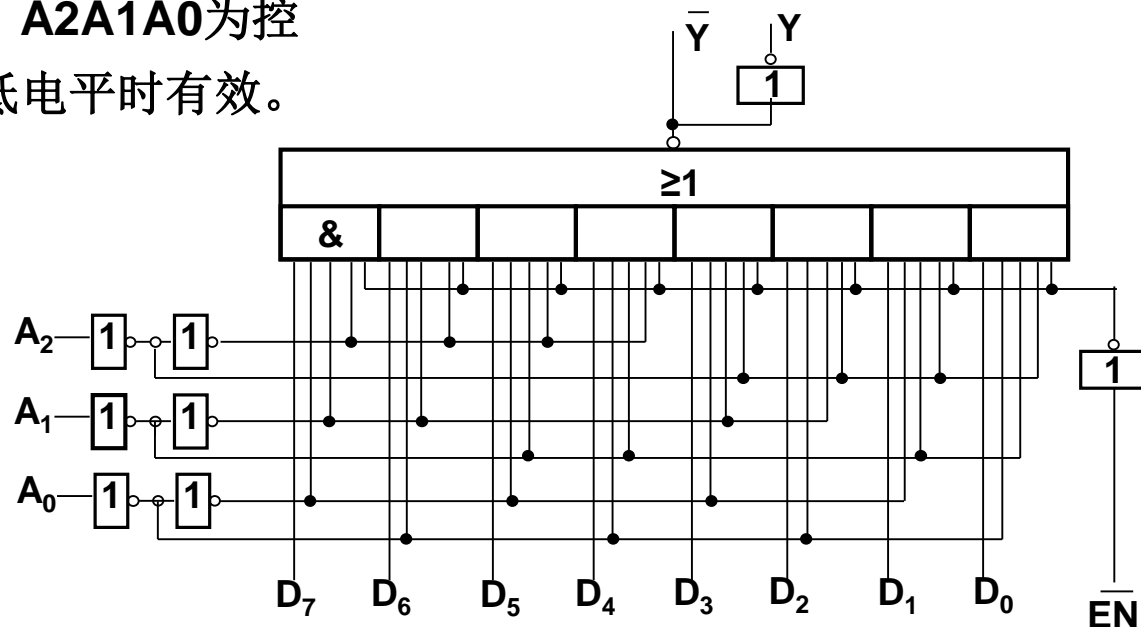
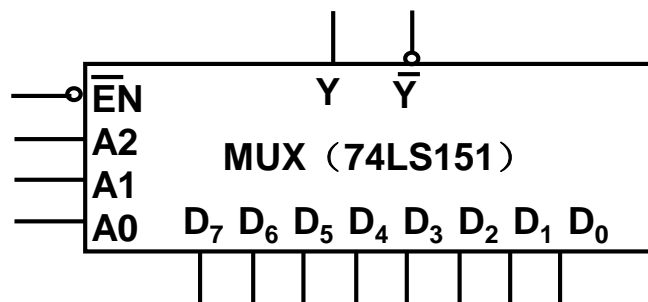


3.4.1 多路选择器（8选1数据选择器74151）

❖ 8选1多路选择器（74151）

功能一： 8选1数据选择器，**A2A1A0**为控制端，使能控制输入**EN**为低电平时有效。

逻辑图与逻辑符号



$$Y = \overline{A_2} \overline{A_1} \overline{A_0} D_0 + \overline{A_2} \overline{A_1} A_0 D_1 + \overline{A_2} A_1 \overline{A_0} D_2 + \overline{A_2} A_1 A_0 D_3 \\ + A_2 \overline{A_1} \overline{A_0} D_4 + A_2 \overline{A_1} A_0 D_5 + A_2 A_1 \overline{A_0} D_6 + A_2 A_1 A_0 D_7$$

3.4.1 多路选择器（8选1数据选择器74151）

功能二：多功能运算电路， $D_7 \sim D_0$ 为控制输入端。

- 通过 $D_7 \sim D_0$ 取不同的值，从输入变量 A_2 、 A_1 、 A_0 的各个最小项中选取某几个最小项输出，实现不同的运算电路。
- 有 $2^8=256$ 种功能，包含3变量的各种最小项表达式，可实现任意组合逻辑电路的设计。

$$Y = D_0 m_0 + D_1 m_1 + D_2 m_2 + D_3 m_3 \\ + D_4 m_4 + D_5 m_5 + D_6 m_6 + D_7 m_7$$

当 $D_7 \sim D_0$ 为0000_0000时， $Y=0$

当 $D_7 \sim D_0$ 为1111_1111时， $Y=1$

当 $D_7 \sim D_0$ 为0000_0001时， $Y=m_0$

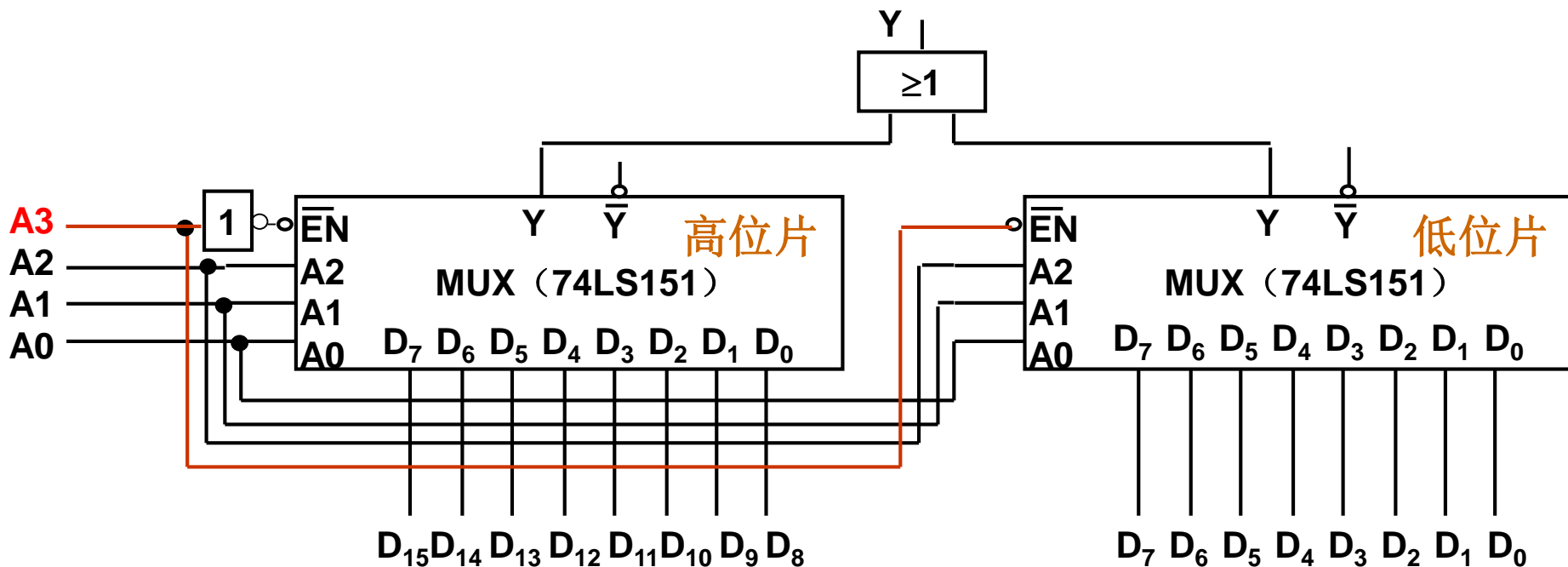
当 $D_7 \sim D_0$ 为1010_0101时， $Y=m_7+m_5+m_2+m_0$

功能表（ $\overline{EN}=0$ ）

A_2	A_1	A_0	Y
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	0	0	D_4
1	0	1	D_5
1	1	0	D_6
1	1	1	D_7

3.4.1 多路选择器（8选1数据选择器74151）

应用使能端将2片74151（8选1）扩展为16选1数据选择器

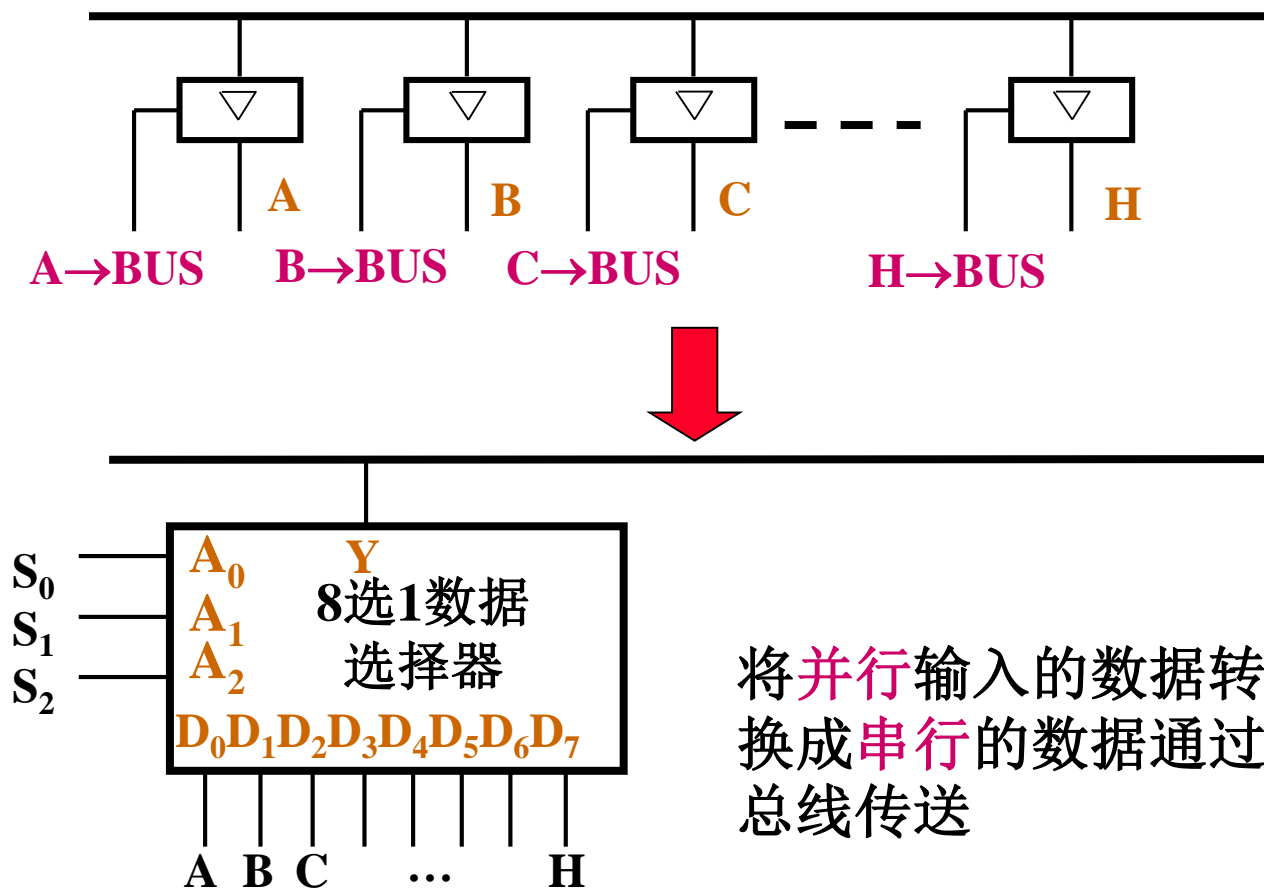


- $A_3=0$ 时，低位片工作，从输入 $D_7 \sim D_0$ 中选择1个输出；
- $A_3=1$ 时，高位片工作，从输入 $D_{15} \sim D_8$ 中选择1个输出。

3.4.1 多路选择器的应用

❖ 数据选择器除选择功能外，还有其他的用途

➤ 代替三态门，实现总线发送控制（并串转换）



$S_2S_1S_0$	Y
0 0 0	A
0 0 1	B
0 1 0	C
0 1 1	D
1 0 0	E
1 0 1	F
1 1 0	G
1 1 1	H

3.4.1 多路选择器的应用

❖ 函数发生器——用数据选择器实现任意组合逻辑函数

数据选择器可以看成是用 N 个控制端 $D_{N-1} \sim D_0$ 选择 2^N 个最小项的某几个组成“与-或”表达式。选择某些控制信号 D_i 为“1”，就是选中这些最小项组成逻辑函数。

❖ 如果给定一个组合逻辑函数，如何用数据选择器实现？

- 利用**互补律**，将组合逻辑函数变换为最小项之和的标准形式；
- 根据该逻辑函数有几个输入变量，确定数据选择器的地址输入为几位，将逻辑函数的输入变量作为数据选择器的地址输入；
- 写出数据选择器的输出表达式；
- 比较逻辑函数的标准形式与数据选择器的输出表达式，推导出数据选择器的 D_i 哪些为1，哪些为0；
- 画出电路图。

3.4.1 多路选择器的应用

利用数据选择器实现逻辑函数：

$$F(A, B, C) = \overline{A}\overline{B}C + \overline{A}B\overline{C} + AB\overline{C} + ABC$$

解：使用8选1数据选择器74151

已知74151的函数表达式：

$$Y = D_0m_0 + D_1m_1 + D_2m_2 + D_3m_3 \\ + D_4m_4 + D_5m_5 + D_6m_6 + D_7m_7$$

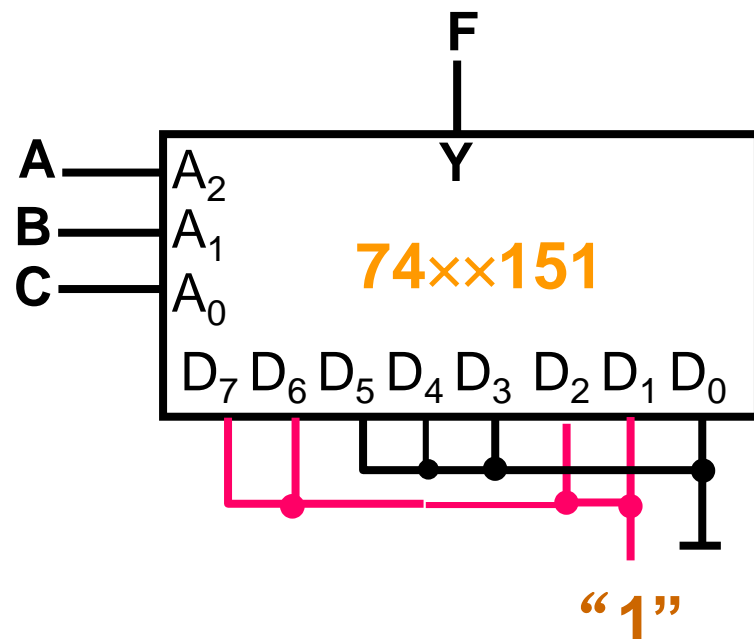
而：

$$F = \overline{A}\overline{B}C + \overline{A}B\overline{C} + AB(\overline{C} + C) \\ = m_1 + m_2 + m_6 + m_7$$

比较 F 和 Y ，则得：

$$D_0=0, D_1=1, D_2=1, D_3=0,$$

$$D_4=0, D_5=0, D_6=1, D_7=1$$

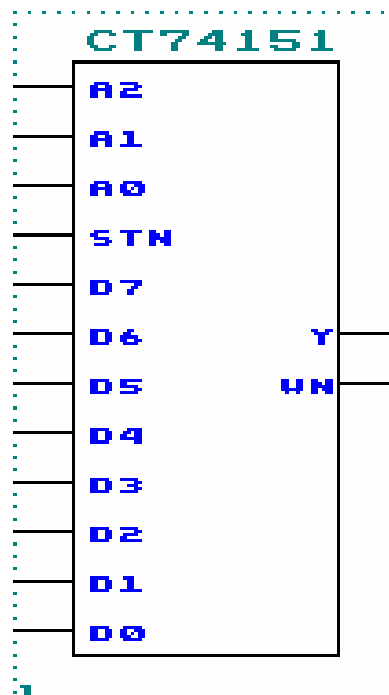


3.4.1 多路选择器（74151的Verilog HDL设计）

❖ 信号定义

- D7~D0: 8位数据输入端
- A2~A0: 地址输入端
- STN: 使能控制端（低电平有效）
- Y: 同相数据输出端
- WN: 反相数据输出端，即WN是Y的反相输出

74151的元件符号



3.4.1 多路选择器（74151的Verilog HDL设计）

```
module CT74151(A2,A1,A0,STN,D7,D6,D5,D4,D3,D2,D1,D0,Y,WN);
    input  A2,A1,A0,STN;
    input  D7,D6,D5,D4,D3,D2,D1,D0;
    output Y,WN;
    reg    Y,WN;
    always
        begin
            if (STN == 0)
                begin
                    case ({A2,A1,A0})
                        3'b000 : Y = D0;
                        3'b001 : Y = D1;
                        3'b010 : Y = D2;
                        3'b011 : Y = D3;
                        3'b100 : Y = D4;
                        3'b101 : Y = D5;
                        3'b110 : Y = D6;
                        3'b111 : Y = D7;
                    endcase
                end
            else Y = 1'b0;
            WN = ~Y;
        end
endmodule
```

74151 Verilog HDL