第 1 章　数字逻辑基础

数字系统所处理的信息通常是用二进制数的形式表示的，可采用的符号只有 0 和 1。本章首先介绍数制和编码，并讨论了二进制的数值运算。为了对数字电路进行分析和设计，本章还介绍了逻辑代数的基础知识，包括逻辑代数的基本公式、常用公式和重要定理，并讲述了逻辑函数的表示方法，介绍如何应用逻辑代数的公式和定理来化简逻辑函数；最后讨论了利用卡诺图化简逻辑函数的方法，并介绍了不完全确定逻辑函数的概念。

1.1　数制与数制转换

所谓“数制”是指进位计数制，即用进位的方式来计数。同一个数可以采用不同的进位计数制来计量。在日常生活中，人们习惯于使用十进制，而在数字电路中常采用二进制，这意味着，将十进制数输入到数字系统之前，必须要把它转换为二进制数；同样，在一个数字系统的输出部分，二进制数也必须要转换为十进制数，以方便人们的读取。除了二进制和十进制外，在数字系统中还广泛采用八进制和十六进制，由于八进制和十六进制可以方便地与二进制进行相互转换，因此这两种进制一般可以用来表示数值较大的二进制数。

1.1.1　十进制

十进制是人们最常用的一种数制。它有以下特点。

（1）采用 10 个计数符号（也称数码）：0，1，2，3，4，5，6，7，8，9。就是说，十进制数中的任 1 位，只可能出现这 10 个符号中的某 1 个。

（2）十进制数的进位规则是“逢十进一”。即每位计满十就向高位进 1，进位基数为 10。所谓“基数”，它表示该数制所采用的计数符号的个数及其进位的规则。因此，同一个符号在一个十进制数中的不同位置时，它所代表的数值是不同的。例如，十进制数 1976.5 可写为：

$$1976.5 = 1\times 10^3 + 9\times 10^2 + 7\times 10^1 + 6\times 10^0 + 5\times 10^{-1}$$

我们把一种数制中各位计数符号为 1 时所代表的数值称为该数位的“权”。十进制数中各位的权是基数 10 的整数次幂。

根据上述特点，任何一个十进制数可以表示为：

$$(N)_{10} = \sum_{i=-m}^{n-1} a_i \times 10^i \qquad (1.1)$$

式中，a_i 为基数 10 的 i 次幂的系数，它可以是 0~9 中的任一个计数符号；n 为 $(N)_{10}$ 的整数位个数；m 为 $(N)_{10}$ 的小数位个数；下标 10 为十进制的进位基数；10^i 为 a_i 所在位的权。

通常把式（1.1）的表示形式称为按权展开式或多项式表示法。

从计数电路的角度来看，采用十进制是不方便的。因为要构成计数电路，必须把电路的状态跟计数符号对应起来，十进制有 10 个符号，电路就必须有 10 个能严格区别的状态与之对应，这样将在技术上带来许多困难，而且也不经济，因此在计数电路中一般不直接采用十进制。

1.1.2 二进制

和十进制类似，二进制具有以下特点。

（1）采用两个符号 0 和 1。

（2）二进制的进位规则为“逢二进一”，即 1+1=10（读为“壹零”）。必须注意，这里的“10”和十进制中的“10”是完全不同的，它实际上等值于十进制数“2”。

根据上述特点，任何具有 n 位整数 m 位小数的二进制数的按权展开式可表示为：

$$(N)_2 = \sum_{i=-m}^{n-1} a_i \times 2^i \tag{1.2}$$

式中，系数 a_i 可以是 0 或 1；下标 2 表示为二进制。

例如：$(101.11)_2 = 1\times 2^2 + 0\times 2^1 + 1\times 2^0 + 1\times 2^{-1} + 1\times 2^{-2}$

根据二进制的特点，目前数字电路普遍采用二进制，其原因如下。

（1）二进制的数字装置简单可靠，所用元器件少。二进制只有两个计数符号 0 和 1，因此它的每一位都可以用任何具有两个不同稳定状态的元件来实现。例如，继电器的闭合和断开，晶体管的饱和与截止等。只要规定一种状态代表“1”，另一种状态代表“0”，就可以表示二进制数。这样使数码的存储和传送变得简单而可靠。

（2）二进制的基本运算规则简单。例如：

加法运算　　0+0=0　　1+0=0+1=1　　1+1=10

乘法运算　　0×0=0　　0×1=1×0=0　　1×1=1

因此二进制数的运算操作简便。

1.1.3 十六进制和八进制

用二进制表示一个数，所用的数位要比十进制多很多，例如表示十进制数$(255)_{10}$，只需 3 位，而用二进制表示该数，却需 8 位，即$(11111111)_2$，不便于书写和记忆。为此常采用十六进制和八进制来表示二进制。上述十进制和二进制的表示法可推广到十六进制和八进制。

十六进制中，采用 16 个计数符号：0，1，2，3，4，5，6，7，8，9，A，B，C，D，E，F。符号 A~F 分别对应于十进制数的 10~15。进位规则是“逢十六进一”，进位基数是 16，十六进制数中各位的权是 16 的整数次幂。任何一个十六进制数，按权展开式为：

$$(N)_{16} = \sum_{i=-m}^{n-1} a_i \times 16^i \tag{1.3}$$

例如：$(6D.4B)_{16} = 6\times 16^1 + D\times 16^0 + 4\times 16^{-1} + B\times 16^{-2}$

$$= 6\times 16^1 + 13\times 16^0 + 4\times 16^{-1} + 11\times 16^{-2}$$

八进制中，采用八个计数符号：0, 1, 2, 3, 4, 5, 6, 7，进位规则是“逢八进一”，进位基数是 8。八进制数中各位的权是 8 的整数次幂，任何一个八进制数的按权展开式为：

$$(N)_8 = \sum_{i=-m}^{n-1} a_i \times 8^i \tag{1.4}$$

例如：$(374.6)_8 = 3\times 8^2 + 7\times 8^1 + 4\times 8^0 + 6\times 8^{-1}$

1.1.4 二进制数与十进制数之间的转换

1. 二进制数转换为十进制数

将二进制数转换为等值的十进制数，常用按权展开法和基数连乘、连除法。

（1）按权展开法

这种方法是将二进制数按式（1.2）展开，然后按十进制的运算规则求和，即得等值的十进制数。

【例 1.1】 将二进制数$(1101.101)_2$转换为等值的十进制数。

解： $(1101.101)_2=1\times2^3+1\times2^2+0\times2^1+1\times2^0+1\times2^{-1}+0\times2^{-2}+1\times2^{-3}$

$=8+4+0+1+0.5+0+0.125$

$=(13.625)_{10}$

简化此法，只要将二进制数中数码为 1 的那些位的权值相加即可，而数码为 0 的那些位可以不去管它。

【例 1.2】 将二进制数$(101101.01)_2$转换为等值的十进制数。

解： $(1\ \ 01\ \ \ 1\ 01.01)_2$

$2^5+2^3+2^2+2^0+2^{-2}$

$=32+8+4+1+0.25$

$=(45.25)_{10}$

表 1.1　2 的幂所对应的数

n	2^n	n	2^n	n	2^n	n	2^n
0	1	6	64	12	4 096	18	262 144
1	2	7	128	13	8 192	19	524 288
2	4	8	256	14	16 384	20	1 048 576
3	8	9	512	15	32 768	21	2 097 152
4	16	10	1 024	16	65 536	22	4 194 304
5	32	11	2 048	17	131 072	23	8 388 608

这种方法要求对 2 的各次幂值比较熟悉，才能较快地实现转换。表 1.1 中列出了 2 的 n 次幂所表示的 24 个数。在计算机工作中，2^{10}用 K（kilo）表示，2^{20}用 M（mega）表示，2^{30}用 G（giga）表示，2^{40}用 T（tera）表示。因此，$4K=2^{12}=4096$，$16M=2^{24}=16\ 777\ 216$。计算机的存储容量通常用字节（B）来表示。一个字节等于 8 位二进制信息，可以表示键盘上的一个字符。计算机上一个 4G 的硬盘就能够容纳 $4G=2^{32}$ 字节的数据（大约 40 亿个字节）。

（2）基数连乘、连除法

二进制数（为了简化分析，假设有 4 位整数，4 位小数）的表示形式可以改写成如下连乘、连除的形式：

$$\begin{aligned}N_2&=(a_3a_2a_1a_0a_{-1}a_{-2}a_{-3}a_{-4})_2\\&=a_3\times2^3+a_2\times2^2+a_1\times2^1+a_0\times2^0+a_{-1}\times2^{-1}+a_{-2}\times2^{-2}+a_{-3}\times2^{-3}+a_{-4}\times2^{-4}\\&=\underline{[\underline{(a_3\times2+a_2)}\times2+a_1]}\times2+a_0+\underline{\{a_{-1}+\underline{[a_{-2}+\underline{(a_{-3}+a_{-4}\times2^{-1})}\times2^{-1}]}\times2^{-1}\}}\times2^{-1}\end{aligned}\qquad(1.5)$$

式（1.5）中，为了说明运算次序，在式子下面画上了算法线条。可见，用连乘、连除法把二进制数转换为十进制数时，其整数和小数部分的转换方法不完全相同。

（1）整数部分的转换是从整数部分最高位开始的：① 将最高位数乘以 2，将所得乘积与下 1 位数相加；② 将①所得之和乘以 2，其乘积再与更下 1 位数相加；③ 这样重复做下去，直到加上整数部分的最低位为止，即得转换后的十进制整数部分。

（2）小数部分的转换是从小数部分最低位开始的：① 将最低位数除以 2(即$\times2^{-1}$),将所得结果与高 1 位数相加；② 把①所得结果除以 2，其结果再与更高 1 位数相加；这样重复做下去，直到加上小数部分的最高位后，再除以 2，即得转换后的十进制小数部分。

（3）最后把整数部分和小数部分相加，即得所求十进制数。

【例 1.3】 用基数连乘、连除法将二进制数$(11001.101)_2$转换为等值的十进制数。

解： 分别转换二进制数的整数部分和小数部分，然后把两部分加起来。

整数部分
从最高位开始

1　1　0　0　1

$1\times2+1=3$

$3\times2+0=6$

$6\times2+0=12$

$12\times2+1=25$

整数部分$(11001)_2=(25)_{10}$。

小数部分 0.101，从最低位开始：

$$1\div2+0=0.5\quad 0.5\div2+1=1.25\quad 1.25\div2=0.625$$

即小数部分$(0.101)_2=(0.625)_{10}$。

故$(11001.101)_2=(25.625)_{10}$。

2．十进制数转换为二进制数

将二进制数转换为十进制数的两种方法的运算过程反过来，就可以实现十进制数到二进制数的转换。相应的两种方法为：提取 2 的幂及基数连除、连乘法。

（1）提取 2 的幂

这种方法是前述用按权展开法将二进制数转换为十进制数运算过程的逆过程，即将十进制数分解为 2 的幂之和，然后从该和式求得对应的二进制数。

【例 1.4】 将十进制数$(45)_{10}$转换为等值的二进制数。

解： $(45)_{10}=32+8+4+1=2^5+2^3+2^2+2^0$

$=1\times2^5+0\times2^4+1\times2^3+1\times2^2+0\times2^1+1\times2^0$

$=(101101)_2$

这种方法的关键是要熟悉 2 的各次幂的值。

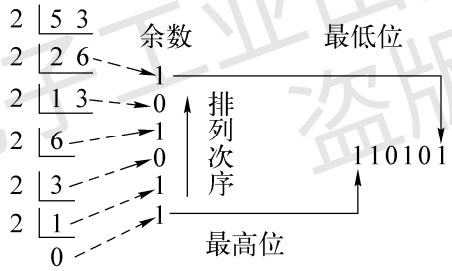
（2）基数连除、连乘法

这种方法也是把十进制数的整数部分和小数部分分别进行转换，然后将结果相加。

整数部分采用“除 2 取余”法转换，即把十进制整数连续除以 2，直到商等于零为止，然后把每次所得余数（1 或者 0）按相反的次序排列，即得转换后的二进制数整数。

【例 1.5】 将十进制数$(53)_{10}$转换为等值的二进制数。

解：

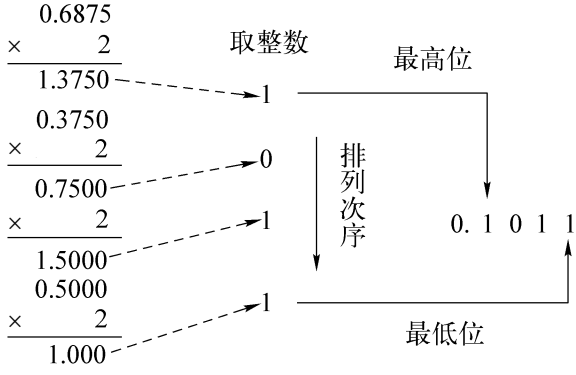


故$(53)_{10}=(110101)_2$。

小数部分采用“乘 2 取整”法转换，即把十进制小数连续乘以 2，直到小数部分为零或者达到规定的位数为止，然后将每次所取整数按序排列，即得转换后的二进制小数。

【例 1.6】 将十进制数$(0.6875)_{10}$转换为等值的二进制数。

解：

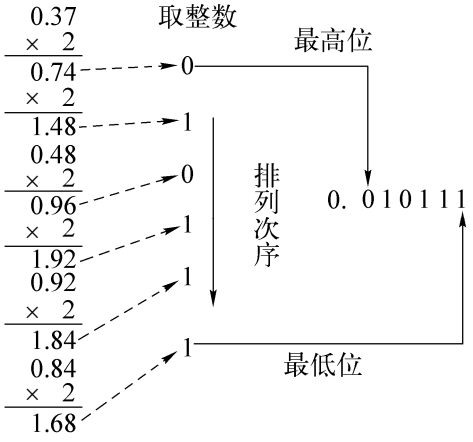


故$(0.6875)_{10}=(0.1011)_2$。

以上每一步都是将前一步所得的小数部分乘以 2。

【例 1.7】 将十进制数$(0.37)_{10}$转换为二进制数（取小数点后六位）。

解：



故$(0.37)_{10}=(0.010111)_2$。

1.1.5 二进制数与十六进制数及八进制数之间的转换

在数字系统中，八进制和十六进制经常用来作为二进制位串简写的方法。二进制的字符串并不总是表示数值，也可以表示非数字信息的某种代码。当处理较多的二进制位数时，用八进制或十六进制书写二进制数更方便且不易出错。但是，必须清楚，数字电路的所有工作都是以二进制形式进行的，八进制和十六进制仅仅是为了人们的使用方便。

十六进制和八进制与二进制之间的转换比较方便，即 4 位二进制数对应 1 位十六进制数，3 位二进制数对应 1 位八进制数。其转换方法为：将二进制数转换为十六进制数（或八进制数）时，从二进制数小数点开始，分别向左、右按 4 位（转换为十六进制）或 3 位（转换为八进制）分组，最后不满 4 位或 3 位的添 0 补位。将每组以对应的十六进制数或八进制数代替，即得等值的十六进制数或八进制数。将十六进制数（或八进制数）转换为二进制数时，方法与上述相反。

【例 1.8】 将二进制数$(111100.101101)_2$转换成十六进制数。

解：首先将$(111100.101101)_2$写成分组形式：$(0011\ 1100.1011\ 0100)_2$，然后将各组 4 位二进制数转换为十六进制数，得：$(3C.B4)_{16}$。

【例 1.9】 将二进制数$(10011101.01)_2$转换成八进制数。

解：将$(10011101.01)_2$写成分组形式：$(010\ 011\ 101.010)_2$，然后将各组 3 位二进制数转换为八进制数，得：$(235.2)_8$。

【例 1.10】 将十六进制数$(6FA.35)_{16}$转化成二进制数。

解：首先分别将 6, F, A, 3, 5 转换为 4 位二进制数，然后按位的高低依次排列，就得到相应的二进制数：$(6FA.35)_{16}=(0110\ 1111\ 1010.0011\ 0101)_2$。

【例 1.11】 将八进制数$(347.12)_8$转换成二进制数。

解：可分别将 3, 4, 7, 1, 2 转换成 3 位二进制数，按位的高低依次排列，就得到相应的二进制数：$(347.12)_8=(011\ 100\ 111.001\ 010)_2$。

1.2 几种简单的编码

编码就是用一组特定的符号表示数字、字母或文字，这一组符号叫做代码。一个 n 位的二进制

码有 2^n 种不同的 0、1 组合，每种组合都可以代表一个编码的元素。尽管给 2^n 个不同的信息元素编码最少需要 n 位二进制数，但对于一组二进制编码来说，它所用的位数是没有最大值的。例如：一个四元素集可以用两位来编码，每个元素唯一对应 00，01，10，11 中的某一个；也可以用 4 位来编码，编码为 0001，0010，0100，1000；或者编码为 1110，1101，1011，0111；只要给每个元素分配一个唯一的数组，在集合中不发生混淆即可。

1.2.1 二-十进制码（BCD 码）

在目前的数字系统中,一般是采用二进制数进行运算的，但是由于人们习惯采用十进制数，因此常需进行十进制数和二进制数之间的转换，其转换方法上面已讨论过了。为了便于数字系统处理十进制数，经常还采用编码的方法，即以若干位二进制码来表示 1 位十进制数，这种代码称为二进制编码的十进制数，简称二-十进制码，或 BCD 码（Binary Coded Decimal Codes）。

因为十进制数有 0~9 共 10 个计数符号，为了表示这 10 个符号中的某一个，至少需要 4 位二进制码。4 位二进制码有 2^4=16 种不同组合，我们可以在 16 种不同的组合代码中任选 10 种表示十进制数的 10 个不同计数符号。根据这种要求可供选择的方法是很多的，选择方法不同，就得到不同的编码形式。常见的有 8421 码、5421 码、2421 码和余 3 码等，如表 1.2 所示。

表 1.2 常用 BCD 码

十进制数	8421 码	5421 码	2421 码	余 3 码
0	0000	0000	0000	0011
1	0001	0001	0001	0100
2	0010	0010	0010	0101
3	0011	0011	0011	0110
4	0100	0100	0100	0111
5	0101	1000	1011	1000
6	0110	1001	1100	1001
7	0111	1010	1101	1010
8	1000	1011	1110	1011
9	1001	1100	1111	1100
无用的位组	1010	0101	0101	0000
	1011	0110	0110	0001
	1100	0111	0111	0010
	1101	1101	1000	1101
	1110	1110	1001	1110
	1111	1111	1010	1111

1．有权 BCD 编码

有权 BCD 编码是以代码的位权值来命名的。在表 1.2 中，8421 码、5421 码和 2421 码为有权码。在这些表示 0~9 共 10 个数字的 4 位二进制代码中，每位数码都有确定的位权，因此可以根据位权展开求得代码所代表的十进制数字。例如，对 8421 码而言，二进制码各位的权从高位到低位依次为 8, 4, 2, 1，如$(0110)_{8421BCD}$ 所代表的十进制数为：$0\times 8+1\times 4+1\times 2+0\times 1=6$。又例如，对 5421 码而言，二进制码各位的权从高位到低位依次为 5, 4, 2, 1，所以$(1010)_{5421BCD}$ 所代表的十进制数为：

$$1\times 5+0\times 4+1\times 2+0\times 1=7$$

在有权码中，8421 码是最常用的，这是由于 8421 码的每位权的规定和二进制数是相同的，因此 8421 码对十进制的 10 个计数符号的表示与普通二进制数是一样的，这样便于记忆。但是要注意，在 8421 码中不允许出现 1010~1111 这六种代码。

对于 5421 和 2421 码，它们的编码形式不是唯一的，表 1.2 中仅列出了其中的一种。例如，数字 6 的 2421 编码可以是 1100 和 0110；数字 7 的 5421 编码可以是 0111 和 1010。一般采用如表 1.2 中所示 5421 和 2421 的编码形式。

【例 1.12】 用 8421BCD 码表示十进制数$(67.58)_{10}$。

解：

十进制数	6	7	·	5	8
	↓	↓		↓	↓
8421BCD 码	0110	0111	·	0101	1000

故$(67.58)_{10}=(01100111.01011000)_{8421BCD}$。

有权的 BCD 码除了上述的 8421 码、5421 码、2421 码外，还有其他多种形式，如 7421 码、5311 码等，甚至还有负权码的形式，如具有两位负权值的 84（−2）（−1）码等。这些有权 BCD

码的编码方式及等值十进制数的计算方法和以上介绍的 8421 码、2421 码的构成方法是一样的。

2．无权 BCD 码

无权码的每位无确定的权，因此不能用按权展开的方法来求它所代表的十进制数。无权码在数字系统中不能进行数值运算。但是这些代码都有其特点，在不同的场合可以根据需要选用。在表 1.2 中，余 3 BCD 码属无权码，它是在每个对应的 8421BCD 代码上加$(3)_{10}=(0011)_2$而得到的。例如，十进制数 6 在 8421BCD 码中为 0110，将它加$(3)_{10}$，得到的 1001 即为十进制数 6 的余 3 码。在余 3 码的编码中，十进制数 0 和 9、1 和 8、2 和 7、3 和 6、4 和 5 对应位的码互为反码（一个是 0，另一个是 1），具有这种特性的代码称为自反代码。在表 1.2 中的 2421 码也是自反代码，但需注意，不是所有的 2421 码都是自反代码。

相对二进制、八进制、十进制、十六进制来说，BCD 码不是一种新的计数体制。事实上，它是将十进制数中的每个数字都用二进制数来进行编码。还要注意的是，BCD 码与直接的二进制数不同，一个二进制数对应的是一个十进制数的整体，而 BCD 码则是分别把每一位十进制数转换为二进制数。例如：

$$(137)_{10}=(10001001)_2$$

$$(137)_{10}=(0001\ 0011\ 0111)_{8421BCD}=(0100\ 0110\ 1010)_{余3BCD}$$

比较上面两个式子可以发现，用 BCD 码表示$(137)_{10}$需要 12 位，而用二进制数表示仅需要 8 位。正如前面指出的那样，由于 BCD 码没有使用所有可能的 4 位编码组合，因此利用率较低，当需要表示的十进制数多于 1 位时，BCD 码比直接用二进制数表示要求有更多的位数。

BCD 码的最大优点是容易实现与十进制数的相互转换，仅需记忆十进制数 0~9 所对应的 4 位二进制编码。在数字系统中，十进制数与 BCD 码的相互转换要依靠逻辑电路来实现。因此，从硬件的角度来看，容易转换是十分重要的。

1.2.2 格雷码

格雷码（Gray 码）是一种常见的无权码，其编码如表 1.3 所示。这种码的特点是：相邻两个代码之间仅有 1 位不同，其余各位均相同。具有这种特点的码称为循环码，故格雷码是一种循环码。格雷码的这个特点使它在代码形成与传输中引起的误差较小。例如，在模拟量到数字量的转换设备中，当模拟量发生微小变化而可能引起数字量发生变化时，格雷码仅改变 1 位，这样与其他码同时改变 2 位或多位的情况相比更为可靠，即减小了出错的可能性。

表 1.3 格雷码与二进制码的关系对照表

十进制数	二进制码	格雷码	十进制数	二进制码	格雷码
	$B_3B_2B_1B_0$	$R_3R_2R_1R_0$		$B_3B_2B_1B_0$	$R_3R_2R_1R_0$
0	0000	0000	8	1000	1100
1	0001	0001	9	1001	1101
2	0010	0011	10	1010	1111
3	0011	0010	11	1011	1110
4	0100	0110	12	1100	1010
5	0101	0111	13	1101	1011
6	0110	0101	14	1110	1001
7	0111	0100	15	1111	1000

假定 n 位二进制码为 $B_n\cdots B_1B_0$，n 位格雷码为 $R_n\cdots R_1R_0$，则两码之间的关系为：

$R_n = B_n$　　$R_i=B_{i+1}\oplus B_i$　　$i\neq n$

$B_n = R_n$　　$B_i=B_{i+1}\oplus R_i$　　$i\neq n$

式中，“$\oplus$”为异或运算符。异或运算的详细内容见 1.4 节。

格雷码无固定的“权”，因而在数字系统中不能直接进行计算；如需要进行计算，则需先把循环码转换成普通的二进制码，这是它的缺点。

1.2.3 奇偶校验码

信息的正确性对数字系统和计算机有极其重要的意义，但在信息的存储与传送过程中，常由于某种随机干扰而发生错误。所以希望在传送代码时能进行某种校验以判断是否发生了错误，甚至能自动纠正错误。

奇偶校验码是一种具有检错能力的代码，它是在原代码（称为信息码）的基础上增加一个码位（称为校验码、校验位或附加位），使代码中含有的 1 的个数均为奇数（称为奇校验）或偶数（称为偶校验），这样通过检查代码中含有的 1 的数目的奇偶性来判别代码的合法性。显然，信号在传送过程中如果代码有两位出错，则这种奇偶校验法是无法检测的，因为两位出错不会改变代码中含 1 码个数的奇偶性。所以，奇偶校验码仅适用于信号出错率很低，且出现成对错误的概率基本为 0 的情况。

表 1.4 给出了由 8421BCD 码变换而得到的奇偶校验码。表中最高位为校验位。

表 1.4 奇偶校验码

十进制数	信息码	奇校验码	偶校验码
0	0000	10000	00000
1	0001	00001	10001
2	0010	00010	10010
3	0011	10011	00011
4	0100	00100	10100
5	0101	10101	00101
6	0110	10110	00110
7	0111	00111	10111
8	1000	01000	11000
9	1001	11001	01001

1.2.4 字符数字码

除了数字数据外，计算机还必须能处理非数字信息。即计算机应能识别表示字母、标点符号和其他特殊符号以及数字的代码。这些代码叫做字符数字码。一个完整的字符数字码应包括 26 个小写英文字母、26 个大写英文字母、10 个数字符号、7 个标点符号，以及其他 20~40 个特殊符号，如+、/、#、%、*等。也就是说，字符数字码能表示计算机键盘上所看到的各种符号和功能键。

美国信息交换的标准代码（简称 ASCII）是应用最为广泛的字符数字码。ASCII 码是 7 位码，因此有 2^7=128 种可能的代码组合。这足以表示标准键盘的字符、回车、换行等控制功能。表 1.5 列出了部分 ASCII 码，对于每一个符号，表中不仅给出了二进制码，而且给出了等值的八进制数和十六进制数。

表 1.5 部分 ASCII 码表

符 号	7 位 ASCII	八 进 制	十六进制	符 号	7 位 ASCII	八进制	十六进制
A	100 0001	101	41	Y	101 1001	131	59
B	100 0010	102	42	Z	101 1010	132	5A
C	100 0011	103	43	0	011 0000	060	30
D	100 0100	104	44	1	011 0001	061	31
E	100 0101	105	45	2	011 0010	062	32
F	100 0110	106	46	3	011 0011	063	33
G	100 0111	107	47	4	011 0100	064	34
H	100 1000	110	48	5	011 0101	065	35
I	100 1001	111	49	6	011 0110	066	36
J	100 1010	112	4A	7	011 0111	067	37
K	100 1011	113	4B	8	011 1000	070	38
L	100 1100	114	4C	9	011 1001	071	39
M	100 1101	115	4D	空格	010 0000	040	20
N	100 1110	116	4E	.	010 1110	056	2E
O	100 1111	117	4F	(	010 1000	050	28
P	101 0000	120	50	+	010 1011	053	2B
Q	101 0001	121	51	$	010 0100	044	24
R	101 0010	122	52	*	010 1010	052	2A
S	101 0011	123	53	)	010 1001	051	29
T	101 0100	124	54	_	010 1101	055	2D
U	101 0101	125	55	/	010 1111	057	2F
V	101 0110	126	56	,	010 1100	054	2C
W	101 0111	127	57	=	011 1101	075	3D
X	101 1000	130	58	<RETURN>	000 1101	015	0D

ASCII 码是 7 位码，但在大多数计算机中 ASCII 字符通常要用一个字节来存储，每个字节中多余的 1 位（最左边 1 位）可以做其他用途，这取决于实际应用。例如，将最高有效位设置为 0，可以认为是 8 位的 ASCII 码；而最高有效位设置成 1，则这 128 个 8 位码可以用来表示希腊字母或斜体字符号等。

1.3 算 术 运 算

当两个二进制数码表示两个数量大小时，它们之间可以进行数值运算，这种运算称为算术运算。数字系统一个重要的功能就是可以对用二进制码表示的数进行各种算术运算，掌握二进制数的基本运算，有助于了解这些运算电路的工作原理和设计过程。

1．二进制加法

任意两个二进制数（不论在什么位置）相加，只可能出现 4 种情况。它们是：

$$0+0=0 \quad 1+0=0+1=1 \quad 1+1=10=0+\text{向高一位的进位 1}$$

$$1+1+1=11=1+\text{向高一位的进位 1}$$

最后一种情况是当本位相加的两个数都为 1，并且从前一位得到一个进位 1 时发生的。

以下是一些二进制数相加的例子（括号里为等值的十进制数）：

$$\begin{array}{rrr} 101(5) & 1001(9) & 101.101(5.625) \\ +\ 011(3) & +\ 1011(11) & +\ 111.011(7.375) \\ \hline 1000(8) & 10100(20) & 1101.000(13) \end{array}$$

现代的数字计算机只需几纳秒便能完成一次加法运算。在所有的数字系统中，电路在同一时间只能进行两个数的相加，所以不用考虑两个以上二进制数同时相加的情况。当多于两个数相加时，只需将前两个数先加到一起，其和再与第三个数相加，以此类推。

加法是数字系统中最重要的一种运算，在绝大多数的现代数字计算机和计算器中所进行的减法、乘法和除法运算，其实都是基于加法运算实现的。

2．有符号数的表示方法

数字系统中，表示二进制数的方法有三种，即原码、反码和补码。在数字计算机中，二进制数的运算常使用的是二进制数的补码系统。二进制数的补码是这样定义的：最高位为符号位，正数为 0，负数为 1；正数的补码和它的原码相同；负数的补码可通过将原码的数值位逐位取反，然后在最低位上加 1 得到。

用补码系统来表示有符号数，可以实现用加法来完成减法的运算，这样数字计算机就可以用相同的电路完成加法和减法运算，从而节省了硬件。

用补码系统表示有符号数的过程如下：如果是正数，数值就用其真实二进制形式，在最高位（MSB）前面的符号位上用 0 表示，如图 1.1 中对$(+45)_{10}$的表示。如果是负数，那么数值就用其补码形式表示，并在最高位前面的符号位上用 1 表示，如图 1.1 中对$(-45)_{10}$的表示。

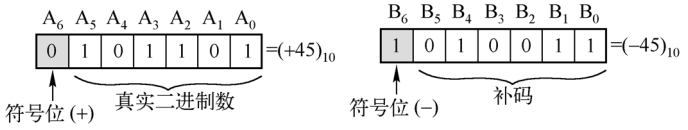


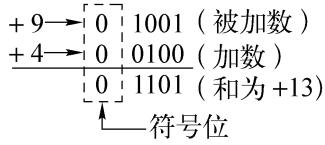
图 1.1 用补码系统表示有符号数

3．补码系统中的加法

这里介绍用补码表示的负数在数字设备中是如何进行加法运算的，在考虑各种不同情况时，要特别注意每个数的符号位同数值位一样要参加运算。

（1）两个正数相加

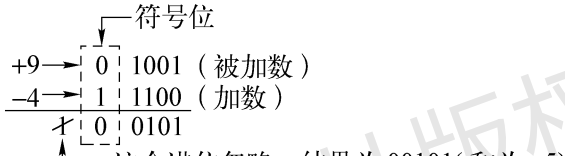
两个正数的加法直接进行。例如，+9 和+4 相加：



注意，其中被加数和加数的符号位都为 0，和数的符号位也是 0，说明和为正数。另外，被加数和加数的位数要一致。这在补码系统中是必需的。

（2）正数与一个比它小的负数相加

例如，+9 和−4 相加。记住：−4 要用补码形式表示，所以+4（00100）必须转换成−4（11100）。



在这种情况下，加数的符号位是 1。注意符号位也参加了加法运算的过程。事实上，在加法的最高位产生了一个进位。这个进位是要被忽略的，所以最后的和为 00101，等于+5。

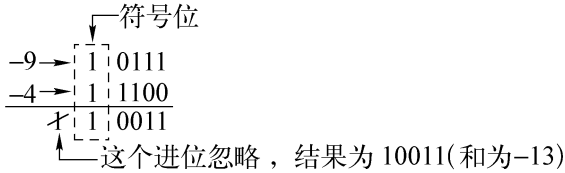
（3）正数与比它大的负数相加

例如，−9 和+4 相加：

这里和的符号位是 1，所以是个负数。因为和是负数，它是补码形式，所以最后的 4 位 1011，其实是和的补码。为了求出和数的真实值，必须对 11011 求相反数（2 的补码）；结果 00101=+5。所以 11011 表示−5。

（4）两个负数相加

例如，−9 和−4 相加：



这个最后的结果也是负数，并且也是符号位为 1 的补码形式。求相反数（求 2 的补码）后，结果为 01101＝+13。

1.4 逻辑代数中的逻辑运算

数字计算机可以进行各种数值运算（如加、减、乘、除等），还可以下棋、翻译、看病等。所有这些功能都说明计算机可以代替人的一部分思维能力。计算机具有这种思维能力的关键在于它具有逻辑判断能力。所谓“逻辑”，简单地说，就是研究前提（或条件）和结论之间的关系，或者说是因果的规律性。所谓“判断”，就是肯定地或是否定地回答一个事物是否具有某种属性。计算机所以具有思维能力，就是因为它在一定的条件下，能体现出一种因果判断关系。

任何矛盾的现象，不管多么复杂，总可以分解成一系列互相对立的简单矛盾。人们判断事物，一般是从简单的判据入手，逐步推理而深入。简单的判据总是二值的，即不是“是”就是“非”，不存在似是而非的情况。这种二值的简单判据称为逻辑命题（在逻辑代数中称为逻辑变量），可以用字母 A，B，C，…表示，并规定用符号 0 和 1 分别表示逻辑命题的两种相反的结果，若“1”表示“肯定”，则“0”就表示“否定”。当然，也可以相反，即“1”表示“错误”，则“0”就表示“正确”。显然，这里的 0 和 1 已经不是数的概念了。

电路上的因果判断关系，体现在数字电路的输入与输出信号之间具有一定的逻辑关系。若把输入信号表示为前提条件，则输出信号就表示所得的结论。我们把这种能根据一定的逻辑关系做出因果判断的数字电路称为“逻辑电路”。逻辑电路的输入和输出通常用电平的高低，或是脉冲的有无来表示，它们也都具有二值性。若用符号 1 表示有脉冲或高电平，则 0 就表示无脉冲或低电平。

为了便于分析和设计逻辑电路，通常用逻辑函数来表示逻辑电路输入与输出信号之间的逻辑关系，逻辑函数的自变量也称为输入变量，因变量也称为输出变量。这样，逻辑电路输入输出信号之间的逻辑关系，通过逻辑函数的自变量和因变量之间的关系得以描述。逻辑电路和逻辑函数是相互对应的。逻辑电路可以用逻辑函数来描述，而逻辑函数也可以用逻辑电路来实现。这样，通过对逻辑函数的分析运算，就可以达到分析和设计逻辑电路的目的。

研究逻辑电路的工具是逻辑代数。它的基本概念是由英国数学家乔治·布尔（George Boole）在 1847 年提出的，故也称为布尔代数。布尔代数是一种较为简单的数学工具，利用布尔代数可以把逻辑电路的输入和输出之间的关系用代数方程（布尔表达式）来描述。

逻辑代数和普通代数也有相同的地方，例如，也用字母 A,B,C,…以及 x,y 等表示变量，但变量的含义及取值范围是不同的。逻辑代数中的变量不表示数值，只表示两种对立的状态，如脉冲的有无，开关的接通和断开，命题的正确和错误等，因此，这些变量的取值只能是 0 或 1，这种变量称为逻辑变量。此外，逻辑代数中对变量的运算和普通代数也有不同的地方。在逻辑代数中只有三种基本的逻辑运算，即“与”、“或”、“非”，以及在基本逻辑运算基础上构成的复合逻辑运算，下面分别予以介绍。

1.4.1 基本逻辑运算

1. “与”逻辑运算

在日常生活中应用“与”逻辑关系的例子很多。例如，一个门上锁着一把明锁和一把暗锁，人们判断“门开”这件事是否成立，就要看前提条件——两把锁的钥匙是否具备。显然，只有同时具备两把锁的钥匙，门才能打开；缺少一把钥匙，门也打不开。这里两把钥匙“具备、不具备”和门“开、不开”之间的因果关系，即为“与”逻辑关系。

又如，如图 1.2 所示，开关 A、B 相串联控制灯 F。显然，只有当开关 A 和 B 均合上时，灯 F

才亮；两个开关中只要有一个断开，灯 F 不亮。这里开关 A、B 的“闭合、断开”和灯 F“亮、不亮”之间的因果关系也是“与”逻辑关系。“与”的含义在此例中就是开关 A、B“闭合”这两个条件同时具备的意思。

通过以上举例分析，可以用文字来叙述“与”逻辑的一般定义，即：只有决定一件事的条件都具备时，这件事才成立；如果有一个（或者一个以上）条件不具备，则这件事不成立。这样的因果关系称为“与”逻辑关系。

“与”逻辑运算也叫逻辑乘。两变量 A、B 的逻辑乘用代数表达式可表示为

$$F = A \cdot B \tag{1.6}$$

式（1.6）称为“与”逻辑函数表达式（简称逻辑函数式，或称逻辑式、函数式）。式中“A·B”读为“A 与 B”或“A 乘 B”。“·”表示“与”运算符号（有些文献中用符号“∧”或“∩”表示），它仅表示“与”的逻辑功能，无数量相乘之意，书写时可把“·”省掉。

图 1.2 中，每个开关的状态只有两种：闭合或断开。灯的状态也有两种：亮或灭。两个开关所有可能的组合状态和灯的状态之间的因果关系如表 1.6 所示。若规定开关合上为 1，开关断开为 0，灯亮为 1，灯灭为 0，则可将表 1.6 转换成表 1.7 的形式。通常把表 1.7 的形式称为真值表。真值表是用表格形式全面、直观地描述所有输入变量（前提条件）取值的各种可能组合和对应的输出变量（结果）值之间的逻辑关系的重要工具。

对 n 个输入变量的情况，其取值的各种可能组合数为 2^n，体现在真值表上则有 2^n 行，每行的排列顺序可按二进制数的正常顺序排列。

由表 1.7 的真值表可知“与”逻辑运算的规则为

$$0 \cdot 0 = 0 \qquad 0 \cdot 1 = 0 \qquad 1 \cdot 0 = 0 \qquad 1 \cdot 1 = 1 \tag{1.7}$$

式（1.7）这组逻辑乘的运算规则是从逻辑推理而来的，故称为公理，它是逻辑代数的基础。

在数字电路中，实现“与”逻辑关系的逻辑电路称为“与”门。“与”门有多个输入端，一个输出端，其逻辑符号如图 1.3 所示。

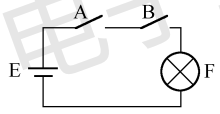


图 1.2 “与”逻辑电路

表 1.6 “与”逻辑电路状态表

开关 A 状态	开关 B 状态	灯 F 状态
断	断	灭
断	合	灭
合	断	灭
合	合	亮

表 1.7 “与”逻辑真值表

A	B	F=AB
0	0	0
0	1	0
1	0	0
1	1	1

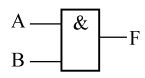


图 1.3 “与”门逻辑符号

“与”门的逻辑功能可以概括为“全 1 出 1，有 0 出 0”。即：只有全部输入均为 1 时，输出才为 1；输入有 0 时，输出为 0。

2. “或”逻辑运算

“或”逻辑关系在日常生活中应用的例子很多。例如，一个门上串起来锁两把锁。人们判断“门开”这件事是否成立，就要看前提条件——两把锁的钥匙是否具备。显然，只要一把锁的钥匙具备，门即可打开；只有两把锁的钥匙都不具备时，门才打不开。这里两把钥匙“具备、不具备”和门“开、不开”之间的因果关系即为“或”逻辑关系。

又如，如图 1.4 所示，开关 A、B 并联控制灯 F。显然，只要有一个开关合上，灯 F 就亮；只有两个开关都断开时，灯 F 才不亮。这里开关 A、B 的“闭合、断开”和灯 F“亮、不亮”之间的因果关系也是“或”逻辑关系。

通过以上举例分析，可以用文字来叙述“或”逻辑的一般定义，即：在决定一件事的各种条件

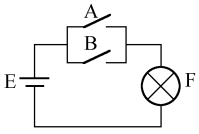


图 1.4 “或”逻辑电路

表 1.8 “或”逻辑电路状态表

开关 A 状态	开关 B 状态	灯 F 状态
断	断	灭
断	合	亮
合	断	亮
合	合	亮

表 1.9 “或”逻辑真值表

A	B	F=A+B
0	0	0
0	1	1
1	0	1
1	1	1

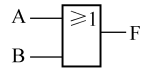


图 1.5 “或”门逻辑符号

中，只要有一个（或者一个以上）条件具备时，这件事就成立；只有所有的条件都不具备时，这件事才不成立。这样的因果关系称为“或”逻辑关系。

“或”逻辑运算也叫逻辑加。两个变量 A、B 的逻辑加用代数表达式可表示为：

$$F=A+B \tag{1.8}$$

式（1.8）中，“A+B”读为“A 或 B”或“A 加 B”。“+”表示“或”运算符号（有些文献中用符号“∨”或“∪”表示），它仅表示“或”的逻辑功能，无数量累加之意。

图 1.4 中两个开关所有可能的组合状态和灯的状态之间的因果关系如表 1.8 所示。与其对应的真值表如表 1.9 所示。

由表 1.6 的真值表可知，“或”逻辑运算的规则为

$$0+0=0 \quad 0+1=1 \quad 1+0=1 \quad 1+1=1 \tag{1.9}$$

式（1.9）这组逻辑加的运算规则，是又一组公理，也是逻辑代数的基础。需要特别注意的是，这里 1+1=1，是表示逻辑加，不是数量的加。从逻辑判断来看，说明对“或”逻辑，具备多个前提条件与具备一个前提条件，所得结论是一样的。

在数字电路中，实现“或”逻辑关系的逻辑电路称为“或”门。“或”门有多个输入端，一个输出端，其逻辑符号如图 1.5 所示。

“或”门的逻辑功能可以概括为“有 1 出 1，全 0 出 0”。

3. “非”逻辑运算

“非”逻辑电路如图 1.6 所示。当开关 A 合上时，灯 F 不亮；而当开关 A 断开时，则灯 F 亮。这里开关 A 的“闭合、断开”与灯 F 的“亮、不亮”之间的因果关系，即为“非”逻辑关系。

用文字叙述“非”逻辑的一般定义，即假定事件 F 成立与否同条件 A 的具备与否有关。若 A 具备，则 F 不成立；若 A 不具备，则 F 成立。F 和 A 之间的这种因果关系称为“非”逻辑关系。

“非”逻辑运算也叫逻辑否定。用代数表达式可表示为：

$$F=\overline{A} \tag{1.10}$$

式（1.10）中 $\overline{A}$ 读为“A 非”，变量 A 上面的短横线表示“非”运算符号。A 叫做原变量，$\overline{A}$ 叫做反变量，$\overline{A}$ 和 A 是一个变量的两种形式。

图 1.6 的“非”逻辑电路的真值表，如表 1.10 所示。由表 1.10 可知，“非”逻辑运算的规则为

$$\overline{0}=1 \qquad \overline{1}=0 \tag{1.11}$$

在数字电路中，实现“非”逻辑关系的逻辑电路称为“非”门（又称反相器）。“非”门只有一个输入端，一个输出端，其逻辑符号如图 1.7 所示。

以上介绍的逻辑代数中的三个基本逻辑运算，即“与”、“或”、“非”，也是人们在进行

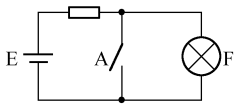


图 1.6 “非”逻辑电路

表 1.10 “非”逻辑真值表

A	$F=\overline{A}$
0	1
1	0

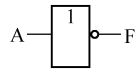


图 1.7 “非”门逻辑符号

逻辑推理时常用的三种基本逻辑关系。和这三种基本逻辑关系相对应的门电路，也是数字电路中最基本的逻辑电路。和这三种门电路相对应的代数表达式，也是逻辑代数中最简单的逻辑函数式。

1.4.2 复合逻辑运算

在实际使用中，并不只是采用“与”门、“或”门、“非”门这三种基本单元电路，而是更广泛地采用“与非”门、“或非”门、“与或非”门、“异或”门及“同或”门等多种复合门电路，它们的逻辑关系可以由“与”、“或”、“非”三种基本逻辑关系推导出，故称为复合逻辑运算。下面分别予以介绍。

1. “与非”逻辑

“与非”逻辑是由“与”逻辑和“非”逻辑组合而成的。其逻辑函数式为：

$$F=\overline{AB} \quad （假定是两个输入变量）$$

“与非”逻辑的真值表如表 1.11 所示，其逻辑功能可概括为“有 0 出 1，全 1 出 0”。

实现“与非”逻辑的逻辑电路称为“与非”门，其逻辑符号如图 1.8 所示。和“与”门逻辑符号相比。“与非”门输出端上多了一个小圆圈，其含义就是“非”。

表 1.11 “与非”逻辑真值表

A	B	$F=\overline{AB}$
0	0	1
0	1	1
1	0	1
1	1	0

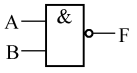


图 1.8 “与非”门逻辑符号

表 1.12 “或非”逻辑真值表

A	B	$F=\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

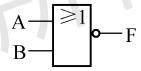


图 1.9 “或非”门逻辑符号

2. “或非”逻辑

“或非”逻辑是由“或”逻辑和“非”逻辑组合而成的。其逻辑函数式为：

$$F=\overline{A+B} \quad （假定是两个输入变量）$$

“或非”逻辑的真值表如表 1.12 所示，其逻辑功能可概括为“全 0 出 1，有 1 出 0”。

实现“或非”逻辑的逻辑电路称为“或非”门，其逻辑符号如图 1.9 所示。

3. “与或非”逻辑

“与或非”逻辑是“与”、“或”、“非”三种逻辑的组合。其逻辑函数式为：

$$F=\overline{AB+CD} \quad （假定有两组“与”输入）$$

“与或非”逻辑的运算次序是：组内先“与”，然后组间相“或”，最后再“非”。其逻辑功能可概括为：“每组有 0 出 1，某组全 1 出 0”。由此不难导出其真值表。

实现“与或非”逻辑的逻辑电路称为“与或非”门，其逻辑符号如图 1.10 所示。

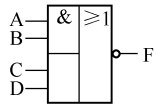


图 1.10 “与或非”门逻辑符号

4. “异或”逻辑

两个变量的异或逻辑函数式为：

$$F=A\overline{B}+\overline{A}B=(\overline{A}+\overline{B})(A+B)=A\oplus B$$

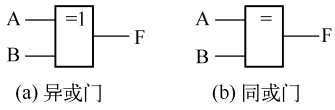
式中，“$\oplus$”表示异或运算符号。

“异或”逻辑的真值表如表 1.13 所示。其逻辑功能可概括为“相异出 1，相同出 0”，这也是“异或”的含义所在。

实现“异或”逻辑的逻辑电路称为异或门，其逻辑符号如图 1.11（a）所示。

表 1.13　“异或”逻辑真值表

A	B	F=A⊕B
0	0	0
0	1	1
1	0	1
1	1	0



(a) 异或门　　(b) 同或门

图 1.11　“异或”门和“同或”门逻辑符号

表 1.14　“同或”逻辑真值表

A	B	F=A⊙B
0	0	1
0	1	0
1	0	0
1	1	1

5. “同或”逻辑

两个变量的“同或”逻辑函数式为：

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

式中，“⊙”表示“同或”运算符号，其逻辑符号如图 1.11（b）所示。

“同或”逻辑的真值表如表 1.14 所示。其逻辑功能可概括为“相同出 1，相异出 0”，故又称为“一致”逻辑或“符合”逻辑。

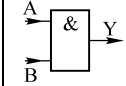
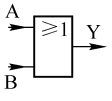
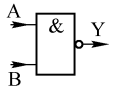
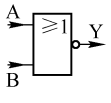
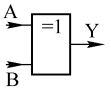
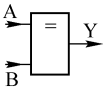
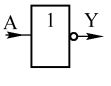
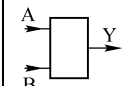
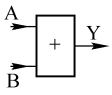
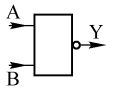
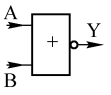
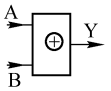
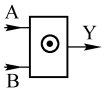
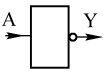
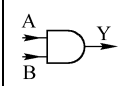
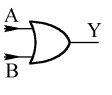
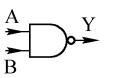
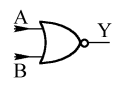
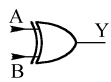
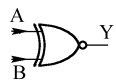
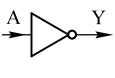
从“异或”逻辑和“同或”逻辑的真值表可以看出，两者互为反函数，即

$$A\oplus B=\overline{A\odot B}\qquad A\odot B=\overline{A\oplus B}$$

在上述的逻辑运算中，运算符的优先级是：① 圆括号，② 非，③ 与，④ 或。换句话说，圆括号内的表达式必须在所有其他的运算前计算，下一个优先的运算是取反，然后是与，最后是或。

为便于查阅，将门的几种表示法列于表 1.15 中。该表中，原部标为过去我国使用的电路符号标准，这些符号在当前出版的许多书籍中还能见到。国外流行的电路符号常见于外文书籍中，在我国引进的一些计算机辅助电路分析和设计软件中，常使用这些符号。

表 1.15　门电路的几种表示法

输出 / 输入 A	B	与 (AND) $Y=A\cdot B$	或 (OR) $Y=A+B$	与非 (NAND) $Y=\overline{AB}$	或非 (NOR) $Y=\overline{A+B}$	异或 (EXOR) $Y=A\oplus B$	异或非 (EXNOR) $Y=\overline{A\oplus B}$	非 (NOT) $Y=\overline{A}$
0	0	0	0	1	1	0	1	1
0	1	0	1	1	0	1	0	1
1	0	0	1	1	0	1	0	0
1	1	1	1	0	0	0	1	0
电路符号 国际								
原部标								
国外流行								

注：表中的国标符号引用 ANSI/IEEE Std.91—1984；国外流行符号引用补充 ANSI/IEEE Std.91a—1991

1.4.3　正逻辑与负逻辑

除了过渡期间外，为了能演示二值逻辑的各种运算（功能），各种逻辑门的输入及输出变量（信号）也必然是二值的。也就是说，它们不是用电压 *V* 的高低来表示，就是用电流 *I* 的大小来说明。实

际上，用电压表征逻辑变量的居多数，通常用较高电平 V_H 代表逻辑 1，较低电平 V_L 代表逻辑 0，这和人们的习惯比较符合，便于测试和观察。至于高、低电平是正是负，具体为何值，这要看所使用的集成电路品种和所加电源电压而定。

通常，将上述高电平 V_H（简写 H）代表逻辑 1，低电平 V_L（简写 L）代表逻辑 0 的约定，简称正逻辑。前面所介绍的与、或、与非及或非等基本门电路的命名，都是在正逻辑约定下的结果。

也可以用相反的约定，即电路的高电平 H 代表逻辑 0，低电平 L 代表逻辑 1，称为负逻辑约定，简称负逻辑。

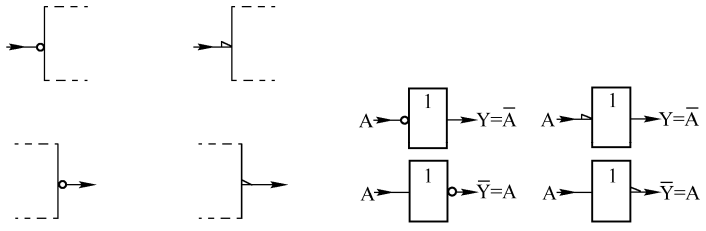
现在以一个具体的门电路为例，分析它在不同逻辑约定下的命名有什么不同。如表 1.16 所示，其中表中(a)是用输入、输出电平列出的真值表；而表中(b)则是正逻辑表示的真值表，显然它应叫做与非门；表中(c)是用负逻辑表示的真值表，很明显它变成了或非门。

表 1.16　正、负逻辑转换举例

（a）电平真值表		
V_{i1}	V_{i2}	V_o
L	L	H
L	H	H
H	L	H
H	H	L
（b）正逻辑与非门		
A	**B**	**Y**
0	0	1
0	1	1
1	0	1
1	1	0
（c）负逻辑或非门		
A	**B**	**Y**
1	1	0
1	0	0
0	1	0
0	0	1

由此可见，正、负逻辑是可以相互转换的。从正逻辑转化到负逻辑就是在一个门的输入和输出端将逻辑 1 变为逻辑 0，将逻辑 0 变为逻辑 1，反之亦然。由于这种运算产生了函数的对偶式，因此，所有终端的极性变化就导致了函数对偶式的采用。这种转化的结果使所有的与运算变为或运算，或运算变为与运算。即正逻辑的与、或、与非及或非门，在负逻辑约定的场合，便可分别用做或、与、或非及与非门，反之亦然。同样，正逻辑的异或门和同或门，分别可转换成负逻辑的同或门和异或门，反之亦然。

在实际的逻辑电路及系统中，如不附加说明，通常都采用正逻辑约定；但也有许多数字设备，其中正、负逻辑往往是混用的，因为这样往往有利于设计的优化和性能的改进。为此，在国际符号中，在需要强调逻辑低电平有效的场合，在逻辑单元框有关的输入、输出处，可标注空心箭头符号，如图 1.12 所示。图中上一排，是标在输入端的记号，下一排是标在输出端的记号。其中图 1.12（a）的小圆圈，表示外部的逻辑 0 相当于单元框内部的逻辑 1，这在前面已经介绍过了；图 1.12（b）用空心箭头记号标注，强调外部的逻辑低电平，相应于内部的逻辑 1；图 1.12（c）就是用这两种记号分别在输入端或输出端时非门的四种表示法，其中上图都表示 $Y=\overline{A}$，下图都表示 $\overline{Y}=A$，它不是说明输入、输出的逻辑状态相反，就是说明输入、输出逻辑电平不同。小圆圈在输入端，强调的是输入逻辑 0，经反相成逻辑 1，作为输出信号；反之，若小圆圈在输出端，则强调的是输入逻辑 1，经反相成逻辑 0，作为输出信号。同理，空心箭头记号在输入端，强调的是输入低电平为有效电平，经反相成高电平，作为输出的有效电平；反之，若空心箭头记号在输出端，则强调的是输入高电平为有效电平，经反相成低电平，作为输出的有效电平。这样的标注方法，在设计或检测逻辑系统时，会带来方便。



(a) 状态记号　　(b) 电平记号　　(c) 非门

图 1.12　外部逻辑的状态及电平记号

从以上叙述还可以看出，在讨论逻辑单元框内的功能时，只有逻辑状态的概念，而且总是采用正逻辑约定；只有在涉及单元框之外的输入、输出线上的逻辑信号时，才会有逻辑状态和逻辑电平两种标注法，也才有采用负逻辑概念的可能性。

1.5 逻辑代数的基本定律和规则

和普通代数相似，逻辑代数中的运算也有一些定律和规则可依。本节介绍逻辑代数的基本定律和几条常用的规则，熟悉这些内容，对数字电路的分析和设计是非常有用的。

1．逻辑函数的相等

逻辑函数和普通代数一样，也有相等的问题。判断两个函数相等，可依照下面的规则：设有两个函数 $F_1=f_1(A_1, A_2, \cdots, A_n)$和 $F_2=f_2(A_1, A_2, \cdots, A_n)$，如果对于 $A_1, A_2, \cdots, A_n$ 的任何一组取值（共 2^n 组），F_1 和 F_2 的值均相等，则称函数 F_1 和 F_2 相等，记为 $F_1=F_2$。换言之，如果 F_1 和 F_2 两函数的真值表相同，则 $F_1=F_2$。反之，如果 $F_1=F_2$，那么这两个函数的真值表一定相同。

【例 1.13】 设有两个函数：$F_1=A+BC$，$F_2=(A+B)(A+C)$。求证：$F_1=F_2$。

解：这两个函数都具有三个变量，有 $2^3=8$ 组逻辑取值，其真值表如表 1.17 所示。由表可见，对应于 A,B,C 的每组取值，函数 F_1 和 F_2 的值均相等，所以 $F_1=F_2$。

表 1.17 F_1 和 F_2 的真值表

A	B	C	F_1	F_2
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

2．基本定律

①0-1 律　$A\cdot 0=0$；　$A+1=1$

②自等律　$A\cdot 1=A$；　$A+0=A$

③重叠律　$A\cdot A=A$；　$A+A=A$

④互补律　$A\cdot \overline{A}=0$；　$A+\overline{A}=1$

⑤交换律　$A\cdot B=B\cdot A$；　$A+B=B+A$

⑥结合律　$A(BC)=(AB)C$；　$A+(B+C)=(A+B)+C$

⑦分配律　$A(B+C)=AB+AC$；　$A+BC=(A+B)(A+C)$

⑧反演律　$\overline{A+B}=\overline{A}\cdot\overline{B}$；　$\overline{AB}=\overline{A}+\overline{B}$

⑨还原律　$\overline{\overline{A}}=A$

其中，反演律也叫德·摩根（De.Morgan）定理，是一个非常有用的定律。

以上逻辑代数的基本定律的正确性，可像例 1.13 那样用真值表加以验证。以基本定律为基础，可以推导出逻辑代数的其他公式，这将在后面介绍。

3．逻辑代数的三条规则

逻辑代数有三条重要规则，即代入规则、反演规则和对偶规则，现分别叙述如下。

（1）代入规则

任何一个含有变量 X 的逻辑等式中，若用逻辑函数式 F 替代等式两边的变量 X，则新的逻辑等式仍然成立。这个规则称为代入规则。

由于任何一个逻辑函数和任何一个逻辑变量一样，只有 0 和 1 两种取值，显然，以上规则是成立的。

利用代入规则，可以将上述基本定律中的变量用任意的逻辑表达式来代替，从而扩大基本定律

的应用范围。

【例 1.14】 试用代入规则证明下式成立：$\overline{A+B+C}=\overline{A}\cdot\overline{B}\cdot\overline{C}$。

解：由德·摩根定理可知等式 $\overline{A+B}=\overline{A}\cdot\overline{B}$，应用代入规则，用 F=B+C 替代等式 $\overline{A+B}=\overline{A}\cdot\overline{B}$ 中所有 B 的位置，则有

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

即

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

可见，应用代入规则可以将两变量的德·摩根定理推广为三变量。同理，可以证明 n 变量的德·摩根定理的成立，即：$\overline{A_1+A_2+\cdots+A_n}=\overline{A}_1\cdot\overline{A}_2\cdot\cdots\cdot\overline{A}_n$。

在使用代入规则时，一定要把所有出现某变量 X 的位置都用同一逻辑函数式替代，否则是不正确的。

（2）反演规则

假设逻辑函数式 $F=f(A_1,A_2,\cdots,A_n)$，则 $\overline{F}=\overline{f(A_1,A_2,\cdots,A_n)}$，称 F 是原函数，$\overline{F}$ 为反函数。若 F 是一个逻辑变量，则称 F 是原变量，$\overline{F}$ 为反变量。由原函数求它的反函数的过程叫做反演或求反。以下的反演规则给出了进行这种运算的一种方法。

设 F 为任意的逻辑表达式，若将 F 中所有的运算符、常量及变量做如下变换

·	+	0	1	原变量	反变量
↓	↓	↓	↓	↓	↓
+	·	1	0	反变量	原变量

则所得的新的逻辑表达式，即为 F 的反函数，记为 $\overline{F}$。这个规则称为反演规则。

反演规则（又称香农定理）实际上是前述反演律的推广，这里就不严格证明了。反演规则为直接求取 $\overline{F}$ 提供了方便。

【例 1.15】 已知 $F=A\overline{B}+\overline{A}B$，求 $\overline{F}$。

解：利用反演规则可得：$\overline{F}=(\overline{A}+B)(A+\overline{B})$。

应用反演规则时，必须注意保持函数的变量间运算次序不要改变。在例 1.15 中 F 的“与”项变成 $\overline{F}$ 的“或”项时，一定要加括号。如果写成 $\overline{F}=\overline{A}+B\cdot A+\overline{B}$，则是错误的。

【例 1.16】 已知 $F=A+\overline{B+\overline{C}+\overline{D+\overline{E}}}$，求 $\overline{F}$。

解：直接应用反演规则可得：$\overline{F}=\overline{A}\cdot\overline{B\cdot C\cdot\overline{D\cdot E}}$。

此例是有多层“非”号的情况。在直接运用反演规则求 $\overline{F}$ 时，注意对不属于单个变量的“非”号保留不变。

（3）对偶规则

若有两个逻辑表达式 F 和 G 相等，则它们的对偶式 F′ 和 G′ 也相等，这就是对偶规则。

所谓“对偶式”是这样定义的：设 F 为任意的逻辑表达式，若将 F 中所有的运算符号和常量做如下变换：

·	+	0	1
↓	↓	↓	↓
+	·	1	0

但变量不变，则所得的新的逻辑表达式，即为 F 的对偶式，记为 F′。

例如：若 $F=A\overline{B}+C\overline{D}$，则 $F'=(A+\overline{B})(C+\overline{D})$；若 $F=A+\overline{B+\overline{C}+\overline{D+\overline{E}}}$，则 $F'=A\cdot\overline{B\cdot\overline{C}\cdot\overline{D\cdot\overline{E}}}$。

实际上对偶是相互的，即 F 和 F′ 互为对偶式。

求对偶式时需要注意：

① 保持原函数变量间运算次序；

② 单变量的对偶式，仍为其自身，如 $F=A$，$F'=A$；

③ 一般情况下，$F'\neq \overline{F}$；在某些特殊情况下，才有 $F'=\overline{F}$。例如，异或表达式 $F=A\overline{B}+\overline{A}B$，$F'=(A+\overline{B})(\overline{A}+B)$，而 $\overline{F}=(\overline{A}+B)(A+\overline{B})$，故 $F'=\overline{F}$。

对偶规则实际上是反演规则和代入规则的应用。因为 F=G，所以 $\overline{F}=\overline{G}$。而 $\overline{F}$ 和 F′，以及 $\overline{G}$ 和 G′ 的区别仅在于，求反函数时要改变变量，而求对偶式时，变量不动。若分别将 $\overline{F}$ 和 $\overline{G}$ 中所有的变量都代之以它们的“非”，则得 F′ 和 G′。根据代入规则，既然 $\overline{F}=\overline{G}$，则有 F′=G′。

回顾前述的基本定律，可以发现，每个定律的两个等式都是互为对偶式。所以有了对偶规则，使得要证明和记忆的公式数目就减少了一半。有时为了证明两个逻辑表达式相等，也可以通过证明它们的对偶式相等来完成。因为在有些情况下，证明它们的对偶式相等更加容易。

例如：已知 A(B+C)=AB+AC，则根据对偶规则必有 A+BC=(A+B)(A+C)。

4．逻辑代数的常用公式

运用逻辑代数的基本定律和三条规则，可以得到更多的公式。常用到的几个公式如下：

消去律 $AB+A\overline{B}=A$

证明： $AB+A\overline{B}=A(B+\overline{B})=A\cdot 1=A$

该公式说明：两个乘积项相加时，若它们只有一个因子不同（如一项中有 B，另一项中有 $\overline{B}$），而其余因子完全相同，则这两项可以合并成一项，且能消去那个不同的因子（即 B 和 $\overline{B}$）。

由对偶规则可得：$(A+B)(A+\overline{B})=A$

吸收律 1 $A+AB=A$

证明： $A+AB=A(1+B)=A\cdot 1=A$

该公式说明：两个乘积项相加时，若其中一项是另一项的因子，则另一项是多余的。

由对偶规则可得：$A(A+B)=A$

吸收律 2 $A+\overline{A}B=A+B$

证明： $A+\overline{A}B=(A+\overline{A})(A+B)=1\cdot(A+B)=A+B$

该公式说明：两个乘积项相加时，若其中一项的非是另一项的因子，则此因子是多余的。

由对偶规则可得：$A(\overline{A}+B)=AB$

包含律 $AB+\overline{A}C+BC=AB+\overline{A}C$

证明： $AB+\overline{A}C+BC=AB+\overline{A}C+(A+\overline{A})BC=AB+\overline{A}C+ABC+\overline{A}BC$

$=AB(1+C)+\overline{A}C(1+B)=AB+\overline{A}C$

该公式说明：三个乘积项相加时，其中两个乘积项中，一项含有原变量 A，另一项含有反变量 $\overline{A}$，而这两项的其余因子都是第三个乘积项的因子，则第三个乘积项是多余的。

该公式可以推广为：$AB+\overline{A}C+BCDE=AB+\overline{A}C$

由对偶规则可得：$(A+B)(\overline{A}+C)(B+C+D+E)=(A+B)(\overline{A}+C)$

5．关于异或（同或）逻辑运算

二输入变量的“异或”和“同或”互为反函数，是仅对偶数个变量而言的，如

$$A\oplus B\oplus C\oplus D=\overline{A\odot B\odot C\odot D}$$

而对奇数个变量，则“异或”等于“同或”，如 $A\oplus B\oplus C=A\odot B\odot C$。

另外，“异或”和“同或”具有如下性质：

异或：	同或：	
$A \oplus 0=A$	$A \odot 1=A$	
$A \oplus 1=\overline{A}$	$A \odot 0=\overline{A}$	
$A \oplus A=0$	$A \odot A=1$	
$A \oplus A \oplus A=A$	$A \odot A \odot A=A$	重叠律
$A \oplus \overline{A}=1$	$A \odot \overline{A}=0$	互补律
$A \oplus B=B \oplus A$	$A \odot B=B \odot A$	交换律
$A \oplus (B \oplus C)=(A \oplus B) \oplus C$	$A \odot (B \odot C)=(A \odot B) \odot C$	结合律
$A \cdot (B \oplus C)=AB \oplus AC$	$A+(B \odot C)=(A+B) \odot (A+C)$	分配律

借助以上介绍的逻辑代数中的基本定律、三个规则和常用公式，可以对复杂的逻辑表达式进行推导、变换和简化，这在分析和设计逻辑电路时是非常有利的。但这些公式反映的是对逻辑变量进行的逻辑运算，而不是对数进行的数值运算。因此，在运用过程中务必注意逻辑代数和普通代数的区别，不能简单地套用普通代数的运算法则。在逻辑代数中，不存在指数、系数、减法和除法。逻辑等式两边相同的项不能随便消去。例如：

$A+A=A$	不能得到	$A+A=2A$
$A \cdot A=A$	不能得到	$A \cdot A=A^2$
$A+\overline{A}=1$	不能得到	$A=1-\overline{A}$
$A\overline{B}+\overline{A}B+AB=A+B+AB$	不能得到	$A\overline{B}+\overline{A}B=A+B$
$A(A+B)=A$	不能得到	$A+B=1$

另外，对于所有公式，要正确理解其含义，避免引起误解。

例如：$A+\overline{A}C=A+C$，这是吸收律第二种表达式。运用代入规则可推广为：$AB+\overline{AB}C=AB+C$，如果认为 $AB+\overline{A}\,\overline{B}C=AB+C$，显然是错误的。这是误认为 $\overline{A}\overline{B}=\overline{AB}$ 了。

1.6 逻辑函数的标准形式

1.6.1 常用的逻辑函数式

逻辑函数式是一种用逻辑代数表达式表示逻辑函数的方法，它是由逻辑变量按照“与”、“或”、“非”等逻辑运算符号组合而成的。例如，$F=(A\overline{B}+\overline{A}B)(A+B)$ 就是以 A、B 为自变量，F 为因变量的一个逻辑函数式。需要再次强调的是，逻辑函数式运算的优先次序是：如式中有括号，则先进行括号内的运算；如无括号，则按“非”、“与”、“或”的次序运算。

逻辑函数式有多种不同的形式。常用的形式有以下五种类型：

例如 $F=AB+\overline{A}C$ （与或式）

$=(A+C)(\overline{A}+B)$ （或与式）

$=\overline{\overline{AB}\cdot\overline{\overline{A}C}}$ （与非与非式）

$=\overline{\overline{A+C}+\overline{\overline{A}+B}}$ （或非或非式）

$=\overline{A\overline{B}+\overline{A}\,\overline{C}}$ （与或非式）

上述五种类型的逻辑函数式是可以相互转换的。

1.6.2 函数的与或式和或与式

利用逻辑函数的基本公式，总可以把任何一个逻辑函数式展开成与或和或与式。其中与或式又叫积之和式，或与式又叫和之积式。

与或式，是指一个函数表达式中包含有若干个“与”项，其中每个“与”项可由一个或多个原变量或反变量组成。这些“与”项的“或”就表示了一个函数。例如，一个四变量函数为：

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

其中，A，$\overline{B}C$，$\overline{A}B\overline{C}D$ 均为“与”项。函数 F 就是一个与或式。

或与式，是指一个函数表达式中包含有若干个“或”项，其中每个“或”项可由一个或多个原变量或反变量组成。这些“或”项的“与”就表示了一个函数。例如，一个三变量函数为：

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

其中，$(A + B)$，$(\overline{A} + C)$，$(\overline{A} + \overline{B} + \overline{C})$ 均为“或”项，这些“或”项的“与”就构成了函数 F。

除上述两种类型外，逻辑函数还可以表示成其他类型。例如，很容易看出，上述三变量函数可写成：

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

这种表示形式既不是与或式，又不是或与式。

另外，即使是同一种类型，写出的函数表达式也不唯一。仍以上面函数为例：

$$\begin{aligned} F(A,B,C) &= (A + B)(\overline{A} + C)(\overline{A} + \overline{B} + \overline{C}) \\ &= (A + B + C\overline{C})(\overline{A} + C)(\overline{A} + \overline{B} + \overline{C}) \\ &= (A + B + C)(A + B + \overline{C})(\overline{A} + C)(\overline{A} + \overline{B} + \overline{C}) \end{aligned}$$

可见，最后一个式子和第一个式子同为或与式，但表示形式不相同。

1.6.3 最小项和最大项

1. 最小项

最小项是一种特殊的乘积项（“与”项），最小项具有以下特点：

（1）n 变量逻辑函数的最小项，一定包含 n 个因子；

（2）在各个最小项中，每个变量以原变量或反变量的形式作为因子仅出现一次。

根据上述特点，容易写出两变量逻辑函数 F(A,B)的最小项为 $\overline{A}\overline{B}$，$\overline{A}B$，$A\overline{B}$，AB；三变量逻辑函数 $F(A,B,C)$ 的所有最小项为 $\overline{A}\overline{B}\overline{C}$，$\overline{A}\overline{B}C$，$\overline{A}B\overline{C}$，$\overline{A}BC$，$A\overline{B}\overline{C}$，$A\overline{B}C$，$AB\overline{C}$，$ABC$，共 8 项。不难看出，$n$ 变量逻辑函数最多有 2^n 个最小项。

为了便于书写和识别，常对最小项进行编号，记为 m_i。这里 m 表示最小项，i 是代号，且 i=0，1，2，…，2^n–1，n 为变量个数。例如，三变量 A, B, C 构成的 8 个最小项之一——$A\overline{B}C$，只有当 A＝1，B＝0，C＝1 时，才使 $A\overline{B}C$＝1。若把 ABC 的取值 101 看成二进制数，那么与之等值的十进制数就是 5，则把 $A\overline{B}C$ 这个最小项记为 m_5。按照类似的方法便可得出三变量最小项的编号如表 1.18 所示。

由最小项的代数形式求其编号还有一个简单的方法，即最小项中的变量若以原变量形式出现的，则记为 1；若以反变量形式出现的，则记为 0。把这些 1 和 0 的有序排列（按最小项中变量排列的顺序）看成二进制数，则与之等值的十进制数，即为最小项编号 m_i 的下标 i。例如，上例中的 $A\overline{B}C = m_5$ 是这样得到的

$$\begin{array}{c} A\overline{B}C = m_5 \\ \downarrow \quad\;\; \uparrow \\ 1\,0\,1 \end{array}$$

反之，由最小项的编号，也可写出相应最小项的代数形式。不过需要注意的是，在提到最小项时，首先要说明变量的数目，以及变量排列的顺序。

例如，三变量 A、B、C 构成的最小项 m_0 和 m_3，分别为 $\overline{A}\overline{B}\overline{C}$ 和 $\overline{A}BC$；四变量 W, X, Y, Z 构成

表 1.18　三变量最小项编号表

最小项	使最小项为 1 的变量取值			对应的十进制数	编号
	A	B	C		
$\bar{A}\bar{B}\bar{C}$	0	0	0	0	m_0
$\bar{A}\bar{B}C$	0	0	1	1	m_1
$\bar{A}B\bar{C}$	0	1	0	2	m_2
$\bar{A}BC$	0	1	1	3	m_3
$A\bar{B}\bar{C}$	1	0	0	4	m_4
$A\bar{B}C$	1	0	1	5	m_5
$AB\bar{C}$	1	1	0	6	m_6
ABC	1	1	1	7	m_7

表 1.19　三变量最小项真值表

A	B	C	m_0 $\bar{A}\bar{B}\bar{C}$	m_1 $\bar{A}\bar{B}C$	m_2 $\bar{A}B\bar{C}$	m_3 $\bar{A}BC$	m_4 $A\bar{B}\bar{C}$	m_5 $A\bar{B}C$	m_6 $AB\bar{C}$	m_7 ABC
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

的最小项 m_0 和 m_3，分别为 $\bar{W}\bar{X}\bar{Y}\bar{Z}$ 和 $\bar{W}\bar{X}YZ$。

三变量 A，B，C 构成的全部最小项的真值表如表 1.19 所示。

由表 1.19 可以看出最小项具有以下主要性质。

（1）每个最小项只有对应的一组变量取值能使其值为 1。例如：最小项 $\bar{A}B\bar{C}$（m_2）只和“010”这组取值对应，即只有当 ABC 取值为 010 时，最小项 $\bar{A}B\bar{C}$ 才为 1。变量取其他各组值时，最小项 $\bar{A}B\bar{C}$ 的值皆为 0。正因为这种“与”函数真值表中 1 的个数最少，所以“最小项”由此得名。

（2）n 个变量的全体最小项（共有 2^n 个）之和恒为 1，即 $\sum_{i=0}^{2^n-1} m_i = 1$。

从表 1.19 中可以看出，变量的每组取值，总有一个相应的最小项为 1，所以全部最小项之和必为 1。

（3）n 个变量的任意两个不同的最小项之积恒为 0，即 $m_i \cdot m_j = 0,\ i \neq j$。这是因为变量的每组取值，对于任何两个不同的最小项不能同时为 1。

（4）相邻的两个最小项之和，可以合并成一项（等于相同因子之积），并消去一个因子。

这里的“相邻”不是几何位置的相邻，而是指逻辑上的相邻。即两个最小项中只有一个因子不同，其余因子完全相同，则称这两个最小项是相邻的，或者称这两个最小项为相邻项。例如，三变量最小项 $\bar{A}B\bar{C}$ 和 $AB\bar{C}$ 是相邻的，因为它们之中只有一个因子不同（前项中有 $\bar{A}$，后项中有 A），其余因子完全相同（两项中都有 B 和 $\bar{C}$）。于是这两项相加可以合并成一项（等于两项中相同因子之积），并消去那个不同的因子，即 $\bar{A}B\bar{C} + AB\bar{C} = B\bar{C}$。

根据上述“相邻”的定义，可知三变量最小项 ABC，将 A 取反得相邻项 $\bar{A}BC$，将 B 取反得相邻项 $A\bar{B}C$，将 C 取反得相邻项 $AB\bar{C}$。即三变量最小项 ABC 有三个相邻项：$\bar{A}BC$，$A\bar{B}C$，$AB\bar{C}$。同理，n 变量最小项有 n 个相邻项。

2．最大项

最大项是一种特殊的和项（“或”项），最大项具有以下特点：

（1）n 个变量构成的每个最大项，一定是包含 n 个因子的和项；

（2）在各个最大项中，每个变量必须以原变量（或反变量）形式作为一个因子仅出现一次。

例如，二变量 A、B 构成的最大项最多有四个，即 A+B，A+$\bar{B}$，$\bar{A}+\bar{B}$，$\bar{A}$+B。n 个变量最多可以构成 2^n 个最大项。

一个函数的最大项记为 M_i，这里 M 表示最大项，i 是代号。例如，三变量 A、B、C 构成的八个最大项之一——（$\overline{A}+B+\overline{C}$），只有当 A = 1，B = 0，C = 1 时，才使 $\overline{A}+B+\overline{C}=0$。若把 ABC 的取值 101 看成二进制数，那么与之等值的十进制数就是 5，则把 $\overline{A}+B+\overline{C}$ 这个最大项记为 M_5。按照类似的方法便可得出三变量最大项的编号如表 1.20 所示。

表 1.20　三变量最大项编号表

最 大 项	使最大项为 0 的变量取值			对应的十进制数	编号
	A	B	C		
A+B+C	0	0	0	0	M_0
A+B+$\overline{C}$	0	0	1	1	M_1
A+$\overline{B}$+C	0	1	0	2	M_2
A+$\overline{B}$+$\overline{C}$	0	1	1	3	M_3
$\overline{A}$+B+C	1	0	0	4	M_4
$\overline{A}$+B+$\overline{C}$	1	0	1	5	M_5
$\overline{A}$+$\overline{B}$+C	1	1	0	6	M_6
$\overline{A}$+$\overline{B}$+$\overline{C}$	1	1	1	7	M_7

由最大项的代数形式求其编号，也有一个简单的方法，即最大项中的变量若以原变量形式出现的，则记为 0；若以反变量形式出现的，则记为 1。把这些 1 和 0 的有序排列（按最大项中变量排列的顺序）看成二进制数，则与之等值的十进制数，即为最大项编号 M_i 的下标 i。例如，上例中的 $\overline{A}+B+\overline{C}=M_5$ 是这样得到的

$$\overline{A}+B+\overline{C}=M_5$$
$$\downarrow\quad\downarrow\quad\uparrow$$
$$1\ 0\ 1$$

反之，由最大项的编号，也可直接写出该最大项的代数形式。

三变量 A、B、C 构成的全部最大项的真值表如表 1.21 所示。

表 1.21　三变量最大项真值表

A	B	C	M_0 $(A+B+C)$	M_1 $(A+B+\overline{C})$	M_2 $(A+\overline{B}+C)$	M_3 $(A+\overline{B}+\overline{C})$	M_4 $(\overline{A}+B+C)$	M_5 $(\overline{A}+B+\overline{C})$	M_6 $(\overline{A}+\overline{B}+C)$	M_7 $(\overline{A}+\overline{B}+\overline{C})$
0	0	0	0	1	1	1	1	1	1	1
0	0	1	1	0	1	1	1	1	1	1
0	1	0	1	1	0	1	1	1	1	1
0	1	1	1	1	1	0	1	1	1	1
1	0	0	1	1	1	1	0	1	1	1
1	0	1	1	1	1	1	1	0	1	1
1	1	0	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	1	0

由表 1.21 可知最大项具有以下性质：

（1）每个最大项只有对应的一组变量取值能使其值为 0。例如，最小项 $\overline{A}+B+\overline{C}$，即 m_5 只和“101”这组取值对应，即只有当 ABC 取值为 101 时，最大项 $\overline{A}+B+\overline{C}$ 才为 0。变量取其他各组值时，最大项 $\overline{A}+B+\overline{C}$ 的值皆为 1。正因为这种“或”函数真值表中 1 的个数最多，所以“最大项”由此得名。

（2）n 个变量的全体最大项（共有 2^n 个）之积恒为 0，即 $\prod_{i=0}^{2^n-1} M_i = 0$。从表 1.21 中看出，变量的每组取值，总有一个相应的最大项为 0，所以全部最小项之积必为 0。

（3）n 个变量的任意两个不同的最大项之和恒为 1，即 $M_i + M_j = 1,\ \ i \neq j$。这是因为变量的每组取值，对于任何两个不同的最大项不能同时为 0。

（4）相邻的两个最大项之积，可以合并成一项（等于相同因子之和），并消去一个因子。例如：

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

3．最小项和最大项的关系

由表 1.19 和表 1.21 可以发现，在相同变量取值的情况下，编号下标相同的最小项和最大项互为反函数，即

$$m_i = \overline{M}_i \qquad M_i = \overline{m}_i \tag{1.12}$$

例如：$m_0 = \overline{A}\ \overline{B}\ \overline{C} = \overline{A+B+C} = \overline{M}_0 \qquad M_0 = A+B+C = \overline{\overline{A}\,\overline{B}\,\overline{C}} = \overline{m}_0$

1.6.4 逻辑函数的标准与或式和标准或与式

1．标准与或式

如果一个逻辑函数式是积之和形式（与或式），而且其中每个乘积项（与项）都是最小项，则称该函数式为标准与或式（也称标准积之和形式，或称最小项之和形式）。

例如，$F(A,B,C) = \overline{A}\,\overline{B}C + \overline{A}BC + A\overline{B}C$ 就是一个标准与或式。为简明起见，该式还可写为：

$$F(A,B,C) = m_1 + m_3 + m_5 = \sum m(1,3,5) = \sum(1,3,5)$$

任何一种逻辑函数式都可以展开为标准与或式，而且是唯一的。

【例 1.17】 试将 $F(A,B,C) = AB + \overline{A}C$ 展开为标准与或式。

解：原式为与或式，但不是标准与或式。其中 AB 和 $\overline{A}C$ 都缺少一个变量，应当补入所缺变量，使之成为最小项，但又不能改变原函数的逻辑功能。方法是：AB 乘（C+$\overline{C}$），$\overline{A}C$ 乘（B+$\overline{B}$），故得：

$$F(A,B,C) = AB + \overline{A}C = AB(C+\overline{C}) + \overline{A}C(B+\overline{B}) = ABC + AB\overline{C} + \overline{A}BC + \overline{A}\,\overline{B}C$$
$$= m_7 + m_6 + m_3 + m_1 = \sum m(1,3,6,7)$$

2．标准或与式

如果一个逻辑函数式是和之积形式（或与式），而且其中每个和项（或项）都是最大项，则称该函数式为逻辑函数的标准或与式（也称标准和之积形式，或称为最大项之积形式）。

例如：$F(A,B,C) = (A+B+C)(A+\overline{B}+C)(\overline{A}+B+C)$

就是一个标准或与式。

为简明起见，上式还可写为

$$F(A,B,C) = M_0 \cdot M_2 \cdot M_4 = \prod M(0,2,4) = \prod(0,2,4)$$

任何一种逻辑函数式也都可以展开为标准或与式，而且是唯一的。

【例 1.18】 试将 $F(A,B,C) = (\overline{A}+C)(B+\overline{C})$ 展开为最大项之积的形式。

解：
$$F(A,B,C) = (\overline{A}+C)(B+\overline{C}) = (\overline{A}+B\cdot\overline{B}+C)(A\cdot\overline{A}+B+\overline{C})$$
$$= (\overline{A}+B+C)(\overline{A}+\overline{B}+C)(A+B+\overline{C})(\overline{A}+B+\overline{C}) \quad \text{（利用分配律）}$$
$$= M_4 \cdot M_6 \cdot M_1 \cdot M_5 = \prod M(1,4,5,6)$$

3．标准与或式和标准或与式的关系

由最小项性质可知：$\sum_{i=0}^{2^n-1} m_i = 1$

而 $F(A_1,A_2,\cdots,A_n) + \overline{F}(A_1,A_2,\cdots,A_n) = 1$

故 $F(A_1,A_2,\cdots,A_n) + \overline{F}(A_1,A_2,\cdots,A_n) = \sum_{i=0}^{2^n-1} m_i$

以三变量函数为例，最多有 $2^3=8$ 个最小项，即 $m_0 \sim m_7$。

若已知 $$F(A,B,C)= m_1+m_3+m_4+m_6+m_7$$

则得 $$\overline{F}(A,B,C)= m_0+m_2+m_5$$

故 $$F(A,B,C)=\overline{m_0+m_2+m_5}=\overline{m}_0\cdot\overline{m}_2\cdot\overline{m}_5=M_0\cdot M_2\cdot M_5$$

对于任意变量的逻辑函数式都存在与上式类似的关系。由此可以得出结论：若已知函数的标准与或式，则可直接写出该函数的标准或与式。在 $0,1,\cdots,(2^n-1)$这 2^n 个编号中，原标准与或式各最小项编号之外的编号，就是标准或与式中最大项的编号；反之，若已知函数的标准或与式，也可以直接写出该函数的标准与或式，在标准与或式中最小项的编号，也就是标准或与式中最大项的编号之外的编号。

【例 1.19】 试将 $F(A,B,C)=AB\overline{C}+BC$ 化为最大项之积的形式。

解：
$$\begin{aligned}F(A,B,C)&=AB\overline{C}+BC=AB\overline{C}+(A+\overline{A})BC=AB\overline{C}+ABC+\overline{A}BC\\&=m_6+m_7+m_3=M_0\cdot M_1\cdot M_2\cdot M_4\cdot M_5\\&=(A+B+C)(A+B+\overline{C})(A+\overline{B}+C)(\overline{A}+B+C)(\overline{A}+B+\overline{C})\end{aligned}$$

【例 1.20】 试求逻辑函数 $F(A,B,C)=(A+\overline{B})(\overline{B}+C)$ 的最小项之和的形式，并求 $\overline{F}$ 和 F' 的最小项之和的形式。

解：
$$\begin{aligned}F(A,B,C)&=(A+\overline{B})(\overline{B}+C)=(A+\overline{B}+C\cdot\overline{C})(A\cdot\overline{A}+\overline{B}+C)\\&=(A+\overline{B}+C)(A+\overline{B}+\overline{C})(A+\overline{B}+C)(\overline{A}+\overline{B}+C)\\&=M_2\cdot M_3\cdot M_6=\sum m(0,1,4,5,7)\end{aligned}$$

则 $$\overline{F}=\sum m(2,3,6)$$

由反演规则和对偶式的定义可知，由原函数 F 来求反函数 $\overline{F}$ 和对偶函数 F' 时，在对 F 进行变化的过程中，运算符和常量的变化是相同的，而变量的变化二者是相反的，所以，$\overline{F}$ 和 F' 除了在相同变量上是取值原变量还是取值反变量不同外，二者的函数表达式是相同的。例如：若已知 $\overline{F}=A\overline{B}C+B\overline{C}$，则 $F'=\overline{A}B\overline{C}+\overline{B}C$。所以，如果在三变量逻辑函数 $\overline{F}$ 的标准与或式中有最小项 $\overline{A}\overline{B}\overline{C}\,(m_0)$，则在 F' 的标准与或式中一定有最小项 $ABC(m_7)$。相同的道理，在 $\overline{F}$ 和 F' 中，m_1 和 m_6 相对应，m_2 和 m_5 相对应，m_3 和 m_4 相对应。所以在本题中由 $\overline{F}=\sum m(2,3,6)$ 可得 $F'=\sum m(5,4,1)=\sum m(1,4,5)$。

1.7 逻辑函数式与真值表

前面曾经提出，真值表和逻辑函数式都是表示逻辑函数的方法。本节介绍这两种表示方法之间的内在联系，以及它们之间的相互转换。

由前述最小项性质可知，最小项只和变量的一组取值相对应，即只有这组变量的取值才能使该最小项为 1。设 $a_1, a_2, \cdots, a_n$ 是变量 $A_1, A_2, \cdots, A_n$ 的一组取值，逻辑函数 $F(A_1, A_2, \cdots, A_n)$是一个标准与或式，m_i 是该函数的一个最小项，则使 $m_i = 1$ 的一组变量取值 $a_1, a_2, \cdots, a_n$，必定有 $F(a_1, a_2, \cdots, a_n)= m_i+0=1+0=1$；反之，如果变量的一组取值 $a_1, a_2, \cdots, a_n$ 使函数 $F(a_1, a_2, \cdots, a_n)=1$，则和 $a_1, a_2, \cdots, a_n$ 对应的项 m_i 必定是 F 的一个最小项。由此可以比较方便地实现逻辑函数式与真值表之间的相互转换。

【例 1.21】 已知逻辑函数式 $F=\overline{A}B+A\overline{B}$，试列出其真值表。

解：原式是二变量函数，而且是最小项之和的形式。使最小项 $\overline{A}B$ 和 $A\overline{B}$ 的值为 1 的变量取值分

别为 01 和 10。也就是说，当 AB 为 01 和 10 时，F＝1；而当 AB 取其他各组值时，F＝0。故可列出真值表如表 1.22 所示。

表 1.22　例 1.21 的真值表

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

表 1.23　例 1.22 的真值表

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	0
1	1	0	1
1	1	1	1

表 1.24　例 1.23 的真值表

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

【例 1.22】　试列出逻辑函数式 F＝AB＋BC 的真值表。

解：这是一个三变量函数，而且是一个非标准与或式。如果将它展开成标准的与或式（即最小项之和的形式），固然可以很方便地列出真值表，但毕竟烦琐。由原式可知，只要 AB＝1 或 BC＝1，就有 F＝1。要使 AB＝1，只要 A、B 同时为 1（不管 C 如何）即可。而要使 BC＝1，只要 B、C 同时为 1（不管 A 如何）即可。因此，在真值表中，只要找出 A、B 取值同时为 1 的行，以及 B、C 取值同时为 1 的行，并将对应的 F 填 1；除此以外，F 填 0 即可。最后所得真值表如表 1.23 所示。

【例 1.23】　已知逻辑函数 F 的真值表如表 1.24 所示，试写出其逻辑函数式。

解：先从真值表找出 F＝1 的各行变量取值：010，100，110，111，将这些变量取值中的 1 写成原变量，0 写成反变量，则得对应的最小项为 $\bar{A}B\bar{C}$，$A\bar{B}\bar{C}$，$AB\bar{C}$，ABC；再将这些最小项相加，即得所求函数 F 的最小项之和形式为

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

从以上举例可以看出，对于一个逻辑函数的与或式和真值表的关系，可以通过函数的最小项之和的形式来联系。最小项之和的形式中各个最小项与真值表中 F =1 的各行变量取值一一对应。具体来说，将真值表中 F =1 的变量取值为 0 的代以反变量，取值为 1 的代以原变量，便得到最小项之和形式中的各个最小项。

类似地，对于一个逻辑函数的或与式和真值表的关系，可以通过函数的最大项之积的形式来联系。最大项之积的形式中各个最大项将与真值表中 F =0 的各行变量取值一一对应，其对应关系正好与上述相反，即 0 对应原变量，1 对应反变量。这里不再一一举例了。

最后需要指出，对同一个逻辑函数的真值表，既可以用最小项之和的形式来表示，也可以用最大项之积的形式来表示。它们所描述的逻辑功能是相同的，可以根据不同的情况来选用这两种表示形式。一般地说，当真值表中 F =1 的行数少时，可选用最小项之和的形式；F=0 的行数少时，可选用最大项之积的形式。因为这样可使所得逻辑函数式简单，从而可能使相应的逻辑电路简单。

1.8　逻辑函数的化简

逻辑函数最终要由逻辑电路来实现，逻辑表达式复杂，得到的逻辑电路就复杂。对逻辑函数进行化简，求得最简逻辑表达式，可以使实现逻辑函数的逻辑电路得到简化。这既有利于节省元器件，降低成本，也利于减小元器件的故障率，提高电路的可靠性；同时简化电路，使元器件之间的连线减少，给制作带来了方便。

由以上讨论知道，同一个逻辑函数，可以用不同类型的表达式表示，与或式是最基本的，其他类型的表达式都可由它转换得到。对于不同的类型，有不同的“最简”标准。这里首先介绍与或表

达式的化简，然后说明如何把与或式转换为其他类型的表达式。

对于与或表达式，假定输入原变量及反变量都可利用，则最简与或式的标准是：

（1）所得与或式中，乘积项（与项）数目最少；

（2）每个乘积项中所含的变量数最少。

逻辑函数常用的化简方法有：公式法、卡诺图法和列表法（Q-M 法）。

1.8.1 公式化简法

所谓公式化简法，是指针对某一逻辑函数式反复运用逻辑代数公式消去多余的乘积项和每个乘积项中多余的因子，使该函数式符合最简标准。利用公式进行化简，无固定步骤可循，全凭化简者的经验和技巧。下面介绍化简中几种常用的方法。

1．并项法

利用公式 $AB+A\overline{B}=A$，将两项合并为一项，并消去因子 B 和 $\overline{B}$。根据代入规则，A 和 B 可以是任何复杂的逻辑式。例如：

$$F_1=AB\overline{C}+\overline{A}B\overline{C}=(A+\overline{A})B\overline{C}=B\overline{C}$$

$$F_2=A\overline{B}C+\overline{A}BC+ABC+\overline{A}\overline{B}C=(A\overline{B}+\overline{A}B)C+(AB+\overline{A}\overline{B})C$$

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

2．吸收法

利用公式 $A+AB=A$，消去多余项。例如：

$$F_1=\overline{B}+A\overline{B}D=\overline{B}$$

$$F_2=\overline{A}+\overline{A\overline{B}C}\ \overline{B+AC+\overline{D}+BC}=(\overline{A}+BC)+(\overline{A}+BC)\overline{B+AC+\overline{D}}=\overline{A}+BC$$

3．消项法

利用公式 $AB+\overline{A}C+BC=AB+\overline{A}C$ 消去多余项。例如：

$$F_1=AB\overline{C}+C\overline{D}+AB\overline{D}=AB\overline{C}+C\overline{D}$$

$$F_2=A\overline{B}C\overline{D}+\overline{A}E+BE+C\overline{D}E=A\overline{B}C\overline{D}+(\overline{A}+B)E+C\overline{D}E$$

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

$$=A\overline{B}C\overline{D}+\overline{A}E+BE$$

4．消因子法

利用公式 $A+\overline{A}B=A+B$，消去多余的变量因子 $\overline{A}$。例如：

$$F_1=\overline{A}+AB+DE=\overline{A}+B+DE$$

$$F_2=AB+\overline{A}C+\overline{B}C=AB+(\overline{A}+\overline{B})C=AB+\overline{AB}C=AB+C$$

5．配项法

利用 $A\cdot 1=A$ 和 $A+\overline{A}=1$，为某项配上一个变量，以便用其他方法进行化简。例如：

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

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

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

还可以利用公式 $A+A=A$，为某项配上其所能合并的项。例如：

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

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

以上介绍了几种常用方法。在实际应用中可能遇到比较复杂的函数式，只要熟练掌握逻辑代数的公式和定理，灵活运用上述方法，总能把函数化成最简。下面是几个综合运用上述方法化简逻辑函数的例子。

$$F_1=A\overline{B}+B\overline{C}+\overline{B}C+\overline{A}B \qquad \text{（配项法）}$$
$$=A\overline{B}+B\overline{C}+(A+\overline{A})\overline{B}C+\overline{A}B(C+\overline{C})$$
$$=(A\overline{B}+A\overline{B}C)+(B\overline{C}+\overline{A}B\overline{C})+(\overline{A}\ \overline{B}C+\overline{A}BC)$$
$$=A\overline{B}+B\overline{C}+\overline{A}C \qquad \text{（吸收法）}$$
$$F_2=AD+A\overline{D}+AB+\overline{A}C+BD+ACEF+\overline{B}EF+DEFG$$
$$=A+AB+\overline{A}C+BD+ACEF+\overline{B}EF+DEFG \qquad \text{（并项法）}$$
$$=A+\overline{A}C+BD+\overline{B}EF+DEFG \qquad \text{（吸收法）}$$
$$=A+C+BD+\overline{B}EF+DEFG \qquad \text{（消因子法）}$$
$$=A+C+BD+\overline{B}EF \qquad \text{（消项法）}$$

由上面的介绍可以看出，公式化简法不仅使用不方便，而且难以判断所得结果是否为最简。由此，公式化简法一般适用于函数表达式较为简单的情况。

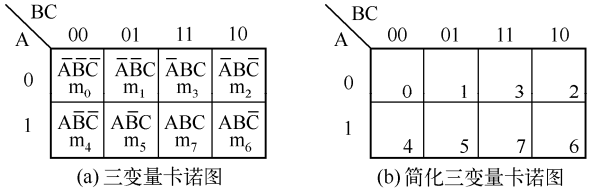
1.8.2 卡诺图化简法

卡诺图化简法是将逻辑函数用一种称为“卡诺图”的图形来表示，然后在卡诺图上进行函数化简的方法。这种方法简单、直观，可很方便地将逻辑函数化成最简。

1．卡诺图的构成

卡诺图是一种包含一些小方块的几何图形。卡诺图中的每一个小方块称为一个单元，每个单元对应一个最小项。当输入变量有 n 个时，最小项有 2^n 个，单元数也是 2^n 个。最小项在卡诺图中的位置不是任意的，它必须满足相邻性规则。所谓相邻性规则，是指任意两个相邻的最小项（两最小项中仅有一个变量不相同），它们在卡诺图中也必须是相邻的。卡诺图中的相邻有两层含义：①几何相邻性，即几何位置上相邻，也就是左右紧挨着或者上下相接；②对称相邻性，即认为图形中两位置对称的单元是相邻的。

图 1.13 为三变量卡诺图，三个输入变量分别为 A,B,C。该图是这样构成的，先画一个含有八个方格（2^3）的矩形图，在图的左上角画一个斜线，将三个变量分为两组，A 为一组，BC 为一组。然后列出每组变量的所有可能的取值，对于单变量 A，可能的取值为 0,1；对于两变量 BC，可能的取值为 00,01,11,10 四种。当变量取值排列顺序确定之后，便可根据图中两组变量的取值组合来确定对应单元的最小项。例如，当 A=0，BC=11 时，它们的组合为 ABC＝011，对应单元的最小项即为 BC＝m_3，因此，可以把变量取值和最小项编号直接对应起来。为简化起见，图 1.13(a)可画成图 1.13(b)的形式。



(a) 三变量卡诺图　　(b) 简化三变量卡诺图

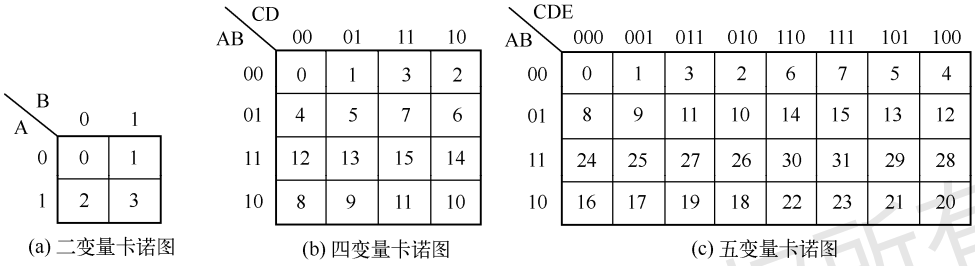
图 1.13　三变量卡诺图

仔细观察图 1.13 可发现，在该图中任何最小项均满足相邻性规则，如 m_1 和 m_0、m_3、m_5 是相邻的，它在几何位置上也满足和 m_0、m_3、m_5 相邻性规则。要注意的是，在三变量卡诺图中，m_0 和 m_2 是相邻的，m_4 和 m_6 也是相邻的，它们分别属于位置对称单元。

在图 1.13 卡诺图中，将变量 BC 的取值按 00, 01, 11, 10 进行排列，这样排列的特点是：任何两组相邻取值，只有 1 位变量取值不同，其余都相同，即符合循环码的排列规则。容易看出，变量取值只有满足这种排列，才能使卡诺图中的任何最小项均符合相邻性规则。

图 1.14 为二变量、四变量、五变量的卡诺图。卡诺图中对称位置相邻在五变量卡诺图中尤其值得注意，如 m_1 和 m_5 属位置对称，它们是相邻的，m_{27} 和 m_{31} 也是相邻的，等等。

当变量数多于六个时，卡诺图就显得很庞大，在实际应用中已失去了它的优越性，一般很少采用。



(a) 二变量卡诺图　(b) 四变量卡诺图　(c) 五变量卡诺图

图 1.14　二、四、五变量卡诺图

2．逻辑函数的卡诺图表示法

先回顾一下逻辑函数的真值表表示法。例如，逻辑函数：

$$F(A,B,C)=\bar{A}BC+A\bar{B}C+ABC=m_3+m_5+m_7$$

当用真值表来表示该函数时，直接根据 ABC 的取值，写出 F 的值。当 ABC 取值分别为 011、101 和 111 时，F=1；否则 F=0。

类似地，用卡诺图来表示逻辑函数时，只需把各组变量取值所对应的逻辑函数 F 的值，填在对应的小方格中，就构成了该逻辑函数的卡诺图。

【例 1.24】　画出 $F(A,B,C)=\bar{A}BC+A\bar{B}C+ABC$ 的卡诺图

解：首先画出三变量卡诺图，然后在 ABC 变量取值为 011、101、111 所对应的三个小方格中填入 1（即在这三种取值时函数 F 的值），在其他位置填入 0，如图 1.15 所示。

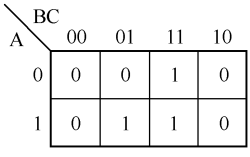


图 1.15　例 1.24 卡诺图

【例 1.25】　画出 $F(A,B,C,D)=\bar{A}\bar{B}\bar{C}\bar{D}+B\bar{C}D+\bar{A}\bar{C}+A$ 的卡诺图。

解：这是一个四变量逻辑函数，式中第一项是最小项，可直接在四变量卡诺图 m_0 的位置填 1；第二项 $B\bar{C}D$ 与变量 A 无关，即只需 BCD＝101，F=1，所以可直接在 CD＝01 的列与 B=1 的行相交的两个小方格（m_5 和 m_{13}）内填 1；第三项 $\bar{A}\bar{C}$ 只含两个变量，说明 AC=00 时，F=1，应在 A=0 的两行和 C=0 的两列相交处（m_0, m_1, m_4, m_5）的小方格内填 1；第四项 A 为单变量，当 A=1 时，F=1，A=1 在卡诺图中的位置为下面两行，即该两行的八个小方格（m_8~m_{15}）均为 1。最后得到的卡诺图如图 1.16 所示。

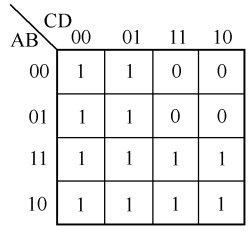


图 1.16　例 1.25 卡诺图

由上两例可知，卡诺图实际上是一种较特殊的真值表，其特殊点在于卡诺图通过几何位置的相邻性，形象地表示出构成逻辑函数的最小项之间在逻

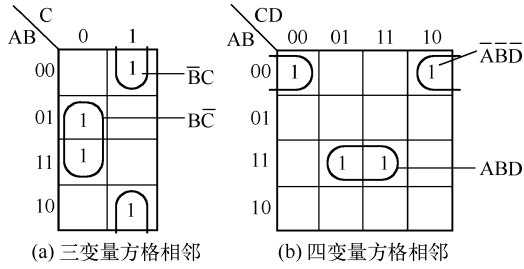
辑上的相邻性。由图 1.15 可知，m_3 和 m_7 相邻，m_5 和 m_7 相邻。而在图 1.16 中，最小项之间的相邻关系就更多了。

3．在卡诺图上合并最小项的规则

在前面讨论最小项的性质时已指出，两相邻的最小项相加，可合并成一项，并可消去一个因子。利用卡诺图化简逻辑函数的基本原理，也就是利用人的直观的阅图能力，去识别卡诺图中最小项之间的相邻关系，并利用合并最小项的规则，将逻辑函数化为最简。

在卡诺图上合并最小项具有下列规则：

（1）卡诺图上任何两个标 1 的方格相邻，可以合为一项，并消去一个变量。例如，在图 1.17（a）中，m_2 和 m_6 相邻，可以合并，即得 $\bar{A}B\bar{C}+AB\bar{C}=B\bar{C}$。所得的简化项中，保留相同的变量 B 和 $\bar{C}$，消去不同的变量 A 和 $\bar{A}$。为表示这两项已合并，在卡诺图中用一小圈将该两项圈在一起。



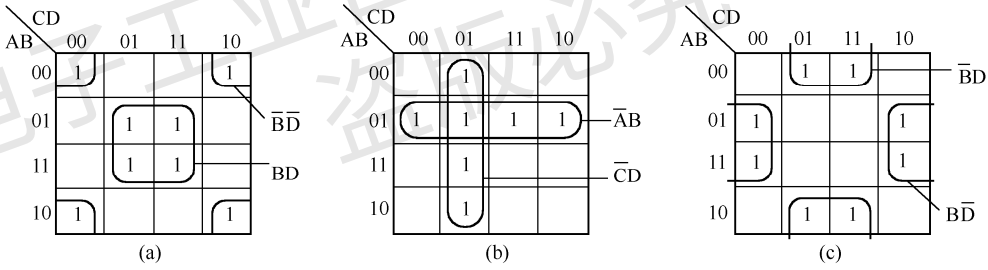
(a) 三变量方格相邻　(b) 四变量方格相邻

图 1.17　两个 1 方格相邻的情况

（2）卡诺图上任何四个标 1 的方格相邻，可以合为一项，并消去两个变量。例如，在图 1.18（a）中，最小项 m_5, m_7, m_{13}, m_{15} 彼此相邻，这四个最小项可以合并，即有

$$(m_5+m_7)+(m_{13}+m_{15})=\bar{A}BD+ABD=BD$$

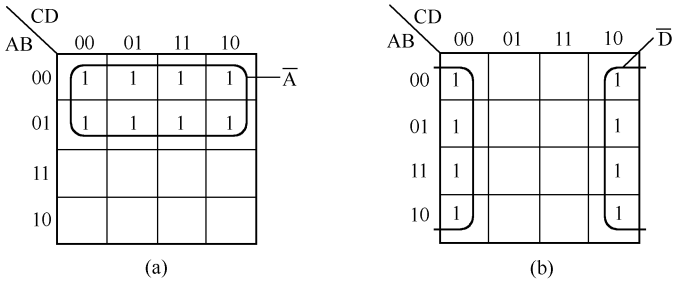
这种合并，在卡诺图中表示为把四个 1 圈在一起。图 1.18 同时列出了四个标 1 方格相邻的几种典型情况。可以看出，四个可合并的相邻最小项在卡诺图中有下列特点：①同在一行或一列；②同在一田字格中。要注意的是：四个角的四个小方格也是符合上面第二个特点的，图 1.18（c）中的两种情况，也属于四个最小项同在一田字格中。



(a)　(b)　(c)

图 1.18　四个 1 方格相邻的情况

（3）卡诺图上任何八个标 1 的方格相邻，可以合并为一项，并可消去三个变量。图 1.19 列出了两种八个标 1 方格相邻的情况，即当相邻两行或相邻两列的方格中均为 1 时，它们可以圈在一起，合并成一项。



(a)　(b)

图 1.19　八个 1 方格相邻的情况

综上所述，在 n 个变量的卡诺图中，只有 2^i（i=0，1，2，…，n）个相邻的标 1 方格（必须排列成方形格或矩形格的形状）才能圈在一起，合并为一项，该项保留了原来各项中 $n-i$ 个相同的变量，消去 i 个不同的变量。

4．用卡诺图化简逻辑函数

首先重点介绍将逻辑函数化为最简的与或式。

前面已介绍过最简与或式的标准，即在所得的与或式中同时满足乘积项数目最少和每一项中的变量数目最少。在用卡诺图化简时，也必须符合这个标准。

（1）化简原则

化简原则是：

① 将所有相邻的标 1 方格圈成尽可能少的圈；

② 在①的条件下，使每个圈中包含尽可能多的相邻标 1 方格。

简言之，即圈子要少，而且圈子要大。因为每个圈对应一个乘积项，圈子少意味着所得与或表达式中的乘积项数目少。圈子大意味着所得乘积项中变量数目少。因此这两条原则是和前述公式法化简的标准是一致的。

【例 1.26】 试用卡诺图将 $F(A,B,C)=\sum m(3,4,5,6,7)$ 化为最简与或式。

解：这是三变量函数，首先画出三变量卡诺图框，再根据构成该函数的各个最小项在卡诺图上找到相应的小方格，并填入 1，如图 1.20（a）所示。显然，图中的 1 方格应圈成两个圈。故得最简式为

$$F(A,B,C)=A+BC$$

如果圈成如图 1.20（b）所示，则得

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

这不是最简式。用公式法可以化简为：

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

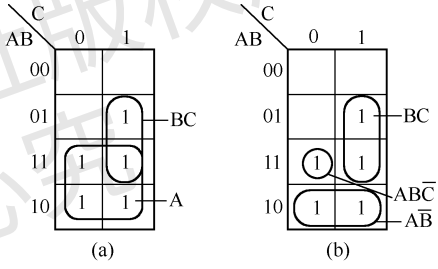


图 1.20　例 1.26 卡诺图

（2）化简步骤

在用卡诺图化简（即圈圈子）的过程中，容易犯的错误是，增加了多余的圈和圈子不是最大。为了避免此问题，现举例说明化简的一般步骤。

【例 1.27】 试用卡诺图将 $F(A,B,C,D)=\sum m(0,1,3,7,8,10,13)$ 化为最简与或式。

解：① 根据函数的变量数，画出相应卡诺图框，再将函数填入卡诺图。

本例为四变量函数，所得卡诺图如图 1.21 所示。

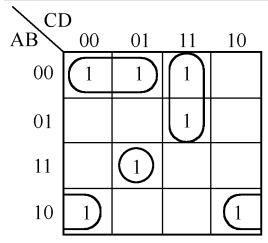


图 1.21　例 1.27 卡诺图

② 圈出孤立的标 1 方格（如果有的话）。所谓孤立，指该 1 方格与其他标 1 方格皆不相邻。

本例的孤立标 1 方格为：$m_{13}=AB\overline{C}D$。

③ 找出只被一个最大的圈所覆盖的标 1 方格（如果有的话），并圈出覆盖该标 1 方格的最大圈。

本例只被一个最大的圈所覆盖的标 1 方格有 m_7 和 m_{10}，覆盖这些标 1 方格的唯一最大圈有：

$$\sum(3,7)=\overline{A}CD,\quad \sum(8,10)=A\overline{B}\,\overline{D}$$

这一步很重要，因为完成这一步后，剩余的标 1 方格少了，再圈圈子就比较直观了。

④ 将剩余的相邻 1 方格，圈成尽可能少，而且尽可能大的圈。

本例剩余的标 1 方格有 m_0、m_1，只能圈成一个最大的圈，即 $\sum m(0,1)=\overline{A}\overline{B}\overline{C}$。

⑤ 最后将各个圈所对应的乘积项相加，即得最简式。

本例中，$F(A,B,C,D)=AB\overline{C}D+\overline{A}CD+A\overline{B}\overline{D}+\overline{A}\overline{B}\overline{C}$。

【例 1.28】 试用卡诺图将 $F=\overline{A}\overline{C}+\overline{A}C\overline{D}+ABD+\overline{B}\overline{C}+\overline{B}C\overline{D}$ 化为最简与或式。

解： ① 将函数填入卡诺图，如图 1.22 所示。

② 本例无孤立的标 1 方格。

③ 本例只被一个最大圈覆盖的标 1 方格有 m_{15}，m_{10}，m_6，覆盖这些标 1 方格的唯一最大圈有：$\sum(13,15)=ABD$，$\sum(0,2,8,10)=\overline{B}\overline{D}$，$\sum(0,2,4,6)=\overline{A}\overline{D}$。

④ 将剩余的标 1 方格 (m_1,m_5,m_9) 圈成一个最大圈，即：$\sum(1,5,9,13)=\overline{C}D$。

⑤ 所得最简式为：$F=ABD+\overline{B}\overline{D}+\overline{A}\overline{D}+\overline{C}D$。

上述化简步骤，对大多数情况都适用。但对特殊情况，就不完全适用。

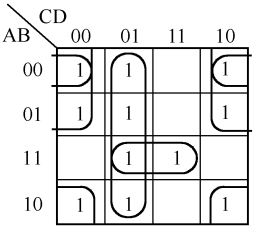


图 1.22 例 1.28 卡诺图

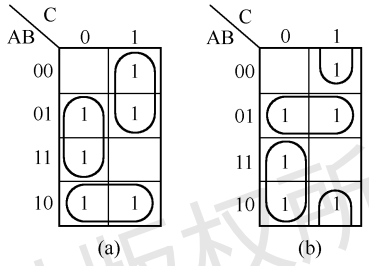


图 1.23 例 1.29 卡诺图

【例 1.29】 试用卡诺图将 $F(A,B,C)=\sum m(1,2,3,4,5,6)$ 化为最简与或式。

解： 此函数的卡诺图如图 1.23 所示。图中每个标 1 方格都被两个最大圈所覆盖。这是一种特殊情况，因此可以得到两种化简结果。

如图 1.23（a）所示，得　　$F=A\overline{B}+B\overline{C}+\overline{A}C$

如图 1.23（b）所示，得　　$F=\overline{A}B+A\overline{C}+\overline{B}C$

（3）化简中注意的问题

除了遵循上述化简原则和化简步骤外，还需注意以下几个问题。

① 所有的圈必须覆盖全部标 1 方格，即每一个标 1 方格必须至少被圈一次。

② 每一个圈中包含的相邻小方格数，必须为 2 的整数次幂。

③ 为了得到尽可能大的圈，圈与圈之间可以重叠一个或 *n* 个标 1 方格。

④ 若某个圈中所有的标 1 方格，已经完全被其他圈所覆盖，则该圈是多余的。即每个圈中至少有一个标 1 方格未被其他圈所覆盖。

⑤ 最简的与或式不一定是唯一的，如例 1.28 就有两种最简结果。

【例 1.30】 试用卡诺图将 $F(A,B,C,D)=\sum m(3,4,5,7,9,13,14,15)$ 化为最简与或式。

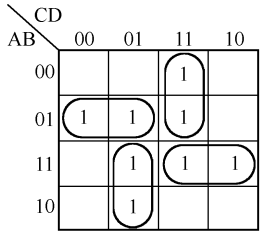


图 1.24 例 1.30 卡诺图

解： 此函数卡诺图如图 1.24 所示。

本例只被一个最大圈覆盖的 1 方格有 m_3,m_4,m_9,m_{14}，覆盖这些标 1 方格的唯一最大圈为：

$\sum(3,7)=\overline{A}CD$，$\sum(4,5)=\overline{A}B\overline{C}$，$\sum(9,13)=A\overline{C}D$，$\sum(14,15)=ABC$

到此所有标 1 方格都被圈过，已经可得最简结果。如果再圈中间四个标 1 方格，将是多余的，故得

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

此例说明，不能孤立地讲“圈子越大越好”，而应当在圈子尽量少的前提下，使圈子尽量大。

（4）用卡诺图求反函数的最简与或式

如果在函数 F 的卡诺图中，合并那些使函数值为 0 的最小项，则可得到 $\overline{F}$ 的最简与或式。

【例 1.31】 试用卡诺图求 $F(A,B,C)=AB+BC+AC$ 的反函数 $\overline{F}$ 的最简与或式。

解： ① 画出 F 的卡诺图，如图 1.25 所示。

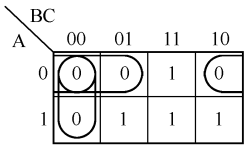


图 1.25 例 1.31 卡诺图

② 合并使函数值为 0 的最小项（图中标 0 的方格）

$$m_0+m_1=\overline{A}\overline{B}\,,\qquad m_0+m_2=\overline{A}\overline{C}\,,\qquad m_0+m_4=\overline{B}\overline{C}$$

③ 写出 $\overline{F}$ 的最简与或式

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

④ 利用还原律，可以由 $\overline{F}$ 的最简与或式得到原函数 F 的最简与或非式

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

（5）用卡诺图对复杂逻辑函数进行运算及化简

【例 1.32】 试用卡诺图求 $F(A,B,C,D)=(\overline{A}\overline{C}\overline{D}+\overline{B}\overline{D}+BD)\oplus(\overline{A}B\overline{D}+\overline{B}D+BC\overline{D})$ 的最简与或式。

解： 将逻辑函数 F 写作：

$$F=F_1\oplus F_2$$

其中

$$F_1=\overline{A}\overline{C}\overline{D}+\overline{B}\overline{D}+BD\,;\quad F_2=\overline{A}B\overline{D}+\overline{B}D+BC\overline{D}$$

将函数 F_1 和 F_2 分别表示在卡诺图上，并在卡诺图上利用“相异出 1，相同出 0”的原则进行异或运算，即观察 F_1 和 F_2 的卡诺图，当 F_1 和 F_2 取值不同时，F 为 1；当 F_1 和 F_2 同为 0 或同为 1 时，则 F 为 0。

得到逻辑函数 F 的卡诺图形式，如图 1.26 所示。

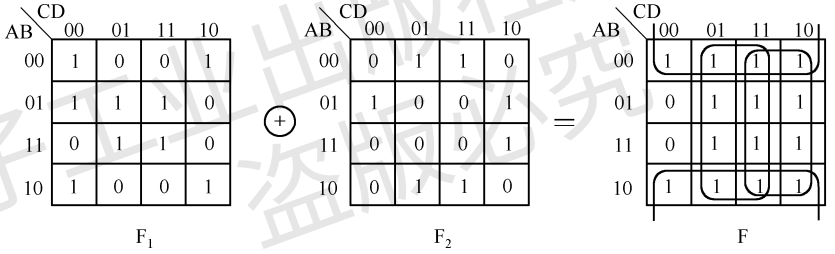


图 1.26 例 1.32 的卡诺图

由 F 的卡诺图可以得到 F 的标准与或式：

$$F(A,B,C,D)=\sum m(0,1,2,3,5,6,7,8,9,10,11,13,14,15)$$

对卡诺图进行化简，求得 F 的最简与或式：

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

读者在利用卡诺图进行逻辑函数化简时，要根据不同的情况灵活运用。例如，在例 1.31 中，观察图 1.26 所示的卡诺图，可以发现在 F 的卡诺图上“0”项比较少，所以可以圈“0”，求得：$\overline{F}(A,B,C,D)=B\overline{C}\overline{D}$，利用反演律也可以求得 F 的最简与或式，即：$F=\overline{\overline{F}}=\overline{B\overline{C}\overline{D}}=\overline{B}+C+D$。

1.8.3 不完全确定的逻辑函数及其化简

前面所讨论的逻辑函数都是属于完全确定的逻辑函数。也就是说，函数的每一组输入变量的取值，都能得到一个完全确定的函数值（0 或 1）。如果逻辑函数有 n 个变量，函数就有 2^n 个最小项，其中每一项都有确定的值。

在某些实际的数字电路中，逻辑函数的输出只与一部分最小项有对应关系，而和余下的最小项

无关。余下的最小项无论是否写入逻辑函数式，都不影响电路的逻辑功能。我们把这些最小项称为无关项。无关项常用英文字母 d 表示，对应的函数值记为“×”（或ϕ）。包含无关项的逻辑函数称为不完全确定的逻辑函数。

发生无关项的情况有两种：一种是由于逻辑变量之间具有一定的约束关系，使得有些变量的取值不可能出现，即所谓“未定义状态”，它所对应的最小项恒等于 0，通常称为约束项；另一种是在某些变量取值下，函数值是 1 还是 0 都可以，并不影响电路的功能，这些变量取值下所对应的最小项称为随意项。

在数字电路的设计中，这种包含有无关项的不完全确定的逻辑函数是经常遇到的。例如，如果逻辑电路的输入是二进制编码的十进制数，4 位二进制输入共有 16 种不同的状态，其中只有 10 种是允许的，有确定的输出；而其余 6 种是不允许的（即存在 6 个无关项），没有确定的输出。在设计中，充分利用无关项可以使设计得到简化。

【例 1.33】 设有一奇偶判别电路，其输入为 1 位十进制数 x 的 8421BCD 码。当输入为偶数时，电路输出为 0；当输入为奇数时，电路输出为 1，如图 1.27 所示。试列出其真值表及卡诺图，并求出最简与或表达式。

解：根据题意，可以列出描述该电路的真值表，如表 1.25 所示。表中，第 10～15 行是不确定的，所以这是一个不完全确定的逻辑函数，其表达式为：

$$F(A,B,C,D)=\sum m(1,3,5,7,9)+\sum d(10,11,12,13,14,15)$$

式中，d 为无关项，即当 ABCD 取 1010～1111 中任一组值时,函数 F 的值既可为 0，也可为 1，故在表中打上“×”。

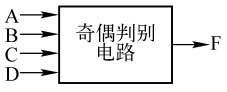


图 1.27　例 1.33 的图

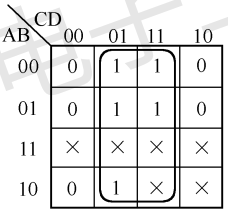


图 1.28　例 1.33 卡诺图

表 1.25　例 1.33 的真值表

十进制数 x	BCD 码 A	B	C	D	F（A, B, C, D）	十进制数 x	BCD 码 A	B	C	D	F（A, B, C, D）
0	0	0	0	0	0	8	1	0	0	0	0
1	0	0	0	1	1	9	1	0	0	1	1
2	0	0	1	0	0	10	1	0	1	0	×
3	0	0	1	1	1	11	1	0	1	1	×
4	0	1	0	0	0	12	1	1	0	0	×
5	0	1	0	1	1	13	1	1	0	1	×
6	0	1	1	0	0	14	1	1	1	0	×
7	0	1	1	1	1	15	1	1	1	1	×

由真值表画出卡诺图，如图 1.28 所示。图中六个无关项对应的小方格打“×”。若不利用无关项化简，即圈中不包含打“×”的方格，结果得：

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

若利用无关项化简，即圈中包含打“×”的方格，以获得尽可能大的圈，如图 1.28 所示，则得：

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

可将函数完整地写为：$\begin{cases}F(A,B,C,D)=D\\ \sum d(10,11,12,13,14,15)=0\end{cases}$ 或 $\begin{cases}F(A,B,C,D)=D\\ AB+AC=0\end{cases}$

即在写出 F 表达式的同时，把约束关系也写上，以便全面地表示逻辑函数的性质。

由例 1.33 可见，充分利用无关项，有可能使逻辑函数进一步简化，从而使逻辑电路更为简单。在卡诺图上化简具有无关项的逻辑函数时，其方法和前述卡诺图化简法基本相同，每个 1 方格都必须至少包含在一个圈中，而无关项（打“×”的方格）却不一定。也就是说，无关项是否包含在某一个或几个圈

中，取决于它是否能使含 1 方格的圈尽可能大（即是否有利于化简）。若能，则该无关项圈入（此时，圈入的“×”当做 1 处理）；若不能，则不圈入（此时，未圈入的“×”当做 0 处理）。

【例 1.34】 化简逻辑函数

$$F(A,B,C,D)=\sum m(0,1,2,3,4,7,8,9)+\sum d(10,11,12,13,14,15)$$

解：画出 F 的卡诺图，如图 1.29 所示，利用无关项化简后得：

$$\begin{cases} F=\overline{C}\overline{D}+CD+\overline{B} \\ AB+AC=0 \end{cases}$$

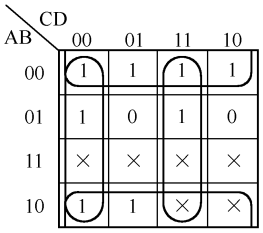


图 1.29　例 1.34 卡诺图

1.8.4　逻辑函数式化简为其他形式

根据上面介绍的方法可知，任何逻辑函数式都可以求得其最简与或式，而有了最简与或式，则可以很方便地得到其他形式的表达式。

下面以 $F=AB+\overline{A}C$ 为例，说明如何将与或式转换为其他类型的表达式。

1．与非与非式

由最简与或式，经过两次求反，可得与非与非式。

$$F=AB+\overline{A}C=\overline{\overline{AB+\overline{A}C}}=\overline{\overline{AB}\cdot\overline{\overline{A}C}}$$

此式即为所求函数的与非与非式。

2．与或非式

求函数的最简与或非式，可以先求出反函数 $\overline{F}$ 的最简与或式，再对 $\overline{F}$ 求反即可。

求反函数的最简与或式可以利用对卡诺图圈 0 方格得到（原则、步骤和圈 1 方格是相同的），如图 1.30 所示。

得到 $\overline{F}$ 的最简与或式为：$\overline{F}=\overline{A}\overline{C}+A\overline{B}$。

再对 $\overline{F}$ 求反便可得到与或非式为：$F=\overline{(\overline{F})}=\overline{A\overline{B}+\overline{A}\overline{C}}$。

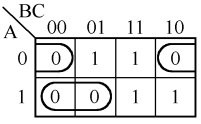


图 1.30　求 $\overline{F}$ 的卡诺图

3．或与式

由最简与或式，运用两次求对偶或两次求反可得或与式。

（1）两次求对偶

例如，对最简与或式：$F=AB+\overline{A}C$，求其对偶式 F' 的最简与或式为

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

再求 F' 的对偶式为

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

此式即为所求函数式的或与式。

（2）两次求反

利用卡诺图圈 0 得到反函数的最简与或式为

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

利用反演规则，再对 $\overline{F}$ 求反，便可得到函数式的或与式为

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

4．或非或非式

由最简或与式，经过两次求反，可得或非或非式

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

以上介绍了由逻辑函数的与或式求得其他形式的表达式的方法。当然，还可以采用其他方法得到转换结果。但需要注意的是，由最简与或式变换为其他形式的表达式，有时不一定是最简的。

1.8.5 奎恩-麦克拉斯基化简法

在对逻辑函数进行化简时，公式化简法的优点是，它的使用不受任何条件的限制。但由于这种方法没有固定的步骤可循，所以在化简一些复杂的逻辑函数时不仅需要熟练地运用各种公式和定理，而且需要有一定的运算技巧和经验。

卡诺图化简法的优点是简单、直观，而且有一定的化简步骤可循，初学者容易掌握这种方法，且化简过程中也易于避免差错。然而在逻辑变量超过 5 个以上时，将失去其简单、直观的优点。

由奎恩（W.V.Quine）和麦克拉斯基（E.J.McCluskey）提出的用列表方式进行化简的方法，克服了公式化简法和卡诺图化简法的的局限性，它有确定的流程，适用于任何复杂逻辑函数的化简，因此适用于编制计算机辅助化简程序。通常将这种化简方法称为奎恩-麦克拉斯基化简法，简称 Q-M 法。

Q-M 法的基本原理仍然是通过合并相邻最小项并消去多余因子而求得逻辑函数的最简与或式。下面结合一个具体的例子简要介绍 Q-M 法的基本原理和化简步骤。

假设 Y 为一个五变量的逻辑函数，其逻辑表达式为：

$$Y(A,B,C,D,E)=A\overline{B}CD\overline{E}+\overline{A}C\overline{D}\overline{E}+\overline{A}\overline{B}\overline{C}D+\overline{A}BD\overline{E}+BCDE+AB\overline{C}DE+AB\overline{C}\overline{D}\overline{E}$$

使用 Q-M 法的化简步骤如下：

（1）将函数化为最小项之和的形式，列出最小项编码表。将 Y 化为最小项之和的形式后，得到

$$Y=\sum m(0,2,3,8,10,14,15,22,24,27,31)$$

用 1 表示最小项中的原变量，用 0 表示反变量，就得到了表 1.26 所示的最小项编码表。

表 1.26　逻辑函数 Y 的最小项编码表

最小项编号	0	2	3	8	10	14	15	22	24	27	31
代码	00000	00010	00011	01000	01010	01110	01111	10110	11000	11011	11111

（2）按包含 1 的个数将最小项分组，如表 1.27 中最左边一列所示。

表 1.27　列表合并最小项

合并前的最小项（$\sum m_i$）							第一次合并结果（含 $n-1$ 个变量的乘积项）							第一次合并结果（含 $n-2$ 个变量的乘积项）						
编号	A	B	C	D	E		编号	A	B	C	D	E		编号	A	B	C	D	E	
0	0	0	0	0	0	√	0, 2	0	0	0	—	0	√	0, 2 8, 10	0	—	0	—	0	P_8
2	0	0	0	1	0	√	0, 8	0	—	0	0	0	√	0, 8 2, 10	0	—	0	—	0	P_8
8	0	1	0	0	0	√	2, 3	0	0	0	1	—	P_2							
3	0	0	0	1	1	√	2, 10	0	—	0	1	0	√							
10	0	1	0	1	0	√	8, 10	0	1	0	—	0	√							
24	1	1	0	0	0	√	8, 24	—	1	0	0	0	P_3							
14	0	1	1	1	0	√	10, 14	0	1	—	1	0	P_4							
22	1	0	1	1	0	P_1	14, 15	0	1	1	1	—	P_5							
15	0	1	1	1	1	√	15, 31	—	1	1	1	1	P_6							
27	1	1	0	1	1	√	27, 31	1	1	—	1	1	P_7							
31	1	1	1	1	1	√														

（3）合并相邻的最小项。将表 1.27 中最左边一列里每一组的每一个最小项与相邻组里所有的最小项逐一比较，若仅有一个因子不同，则可合并，并消去不同的因子。消去的因子用“—”号表示，将合并后的结果列于表 1.27 的第二列中。同时，在第一列中可以合并的最小项右边标以“√”号。

按照同样的方法再将第二列中的乘积项合并，合并后的结果写在第三列中。

如此进行下去，直到不能再合并为止。

（4）选择最少的乘积项。只要将表 1.27 合并过程中没有用过的那些乘积项相加，自然就包含了函数 Y 的全部最小项，故有：

$$Y = P_1 + P_2 + P_3 + P_4 + P_5 + P_6 + P_7 + P_8$$

然而，上式并不一定是最简的与或表达式。为了进一步化简，将 P_1~P_8 各包含的最小项列成表 1.28。因为表中带圆圈的最小项仅包含在一个乘积项中，所以化简结果中一定包含它们所在的乘积项，即 P_1，P_2，P_3，P_7 和 P_8。而且选取了这五项之和以后，已包含了除 m_{14} 和 m_{15} 以外所有 Y 的最小项。

最后就是要确定化简结果中是否应包含 P_4，P_5 和 P_6。为此可将表 1.28 中有关 P_4、P_5 和 P_6 的部分简化成表 1.29 的形式。

由表 1.29 中可以看到，P_4 行中所有的 1 和 P_6 行中所有的 1 皆与 P_5 中的 1 重叠，亦即 P5 中的最小项包含了 P4 和 P6 的所有最小项，故可将表达式中 P_4 和 P_6 两项删掉。从而得到最后的化简结果：

$$\begin{aligned} Y(A,B,C,D,E) &= P_1 + P_2 + P_3 + P_5 + P_7 + P_8 \\ &= A\bar{B}CD\bar{E} + \bar{A}B\bar{C}D + B\bar{C}\bar{D}\bar{E} + \bar{A}BCD + ABDE + \bar{A}\bar{C}\bar{E} \end{aligned}$$

表 1.28　用列表法选择最少的乘积项

P_j \ m_i	0	2	3	8	10	14	15	22	24	27	31
P_1								①			
P_2		1	①								
P_3				1					①		
P_4					1	1					
P_5						1	1				
P_6							1				1
P_7										①	1
P_8	①	1		1	1						

表 1.29　表 1.28 的 P_4，P_5，P_6 部分

P_j \ m_i	14	15
P_4	1	
P_5	1	1
P_6		1

从上面的例子可以看出，虽然 Q-M 法的化简过程看起来比较烦琐，但由于有一定化简步骤，适用于任何复杂逻辑函数的化简，这就为编制计算机辅助化简程序提供了方便。因此，几乎很少有人用手工方法使用 Q-M 法去化简复杂的逻辑函数，而是使用基于该方法的基本原理去编制各种计算机软件，然后在计算机上完成逻辑函数的化简工作。

1.8.6　多输出逻辑函数的化简

前面讨论的逻辑函数化简，都是针对单个函数（相应的逻辑电路只有一个输出端）而言的。实际的数字电路，常常是一个多输出电路，即对应于相同一组输入变量，存在着多个输出函数，根据这些输出函数式，实现为电路时，不是各自分立的，而是相互构成一个整体电路。因此，多输出函数的化简，虽然也是以单个函数的化简方法为基础的，但却不能像单个函数化简那样只顾各个函数本身最简，即不是追求对应各单个函数的局部最简，而必须从整体出发，以使多输出函数整体最简，从而使所实现的整体电路最简。这就是多输出函数化简的指导思想。多输出函数化简，可以利用卡诺图法，也可以利用列表法。这里仅介绍卡诺图法。利用卡诺图法化简多输出函数的关键在于

寻找并恰当地利用全部函数或部分函数的公共项（在卡诺图上即为公共圈）。下面举例说明。

【例 1.35】 化简下列多输出函数

$$\begin{cases} F_1(A,B,C)=\sum m(1,4,5) \\ F_2(A,B,C)=\sum m(1,3,7) \end{cases}$$

解： 作出 F_1 和 F_2 的卡诺图，如图 1.31 所示。

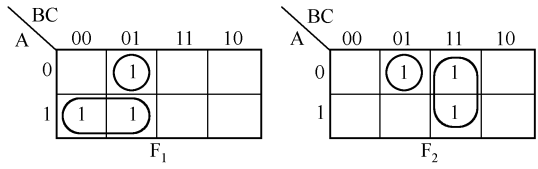


图 1.31　例 1.35 的卡诺图

若按单个函数化简方法，其结果为

$$F_1=A\overline{B}+\overline{B}C \qquad F_2=\overline{A}C+BC$$

相应的电路图如图 1.32 所示（假定输入提供原、反变量）。

现在从整体出发，考虑函数的化简。从 F_1 和 F_2 的卡诺图中看出，m_1 是两个函数的公有乘积项，可先将它单独圈出。其余的 1 方格，仍按一般卡诺图的圈选原则进行圈选（如图 1.31 所示），其结果为：

$$F_1=\overline{A}\overline{B}C+A\overline{B} \qquad F_2=\overline{A}\overline{B}C+BC$$

上式从单个函数看并非最简，但从整体看却是最简的。相应的电路图如图 1.33 所示。比较图 1.32 和图 1.33，显然后者节省了一个门和一条连线。

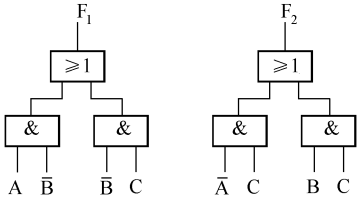


图 1.32　例 1.35 按单个函数化简的逻辑图

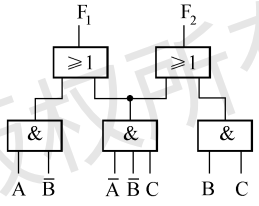


图 1.33　例 1.35 按多输出函数化简的逻辑图

上例中得到的公共圈是有用的，即有利于整体电路最简，但并非任何情况下得到的公共圈都是有用的。

【例 1.36】 化简下列多输出函数

$$\begin{cases} F_1(A,B,C,D)=\sum m(2,3,5,7,8,9,10,11,13,15) \\ F_2(A,B,C,D)=\sum m(2,3,5,6,7,10,11,14,15) \\ F_3(A,B,C,D)=\sum m(6,7,8,9,13,14,15) \end{cases}$$

解： 作出 F_1、F_2 和 F_3 的卡诺图，如图 1.34 所示。

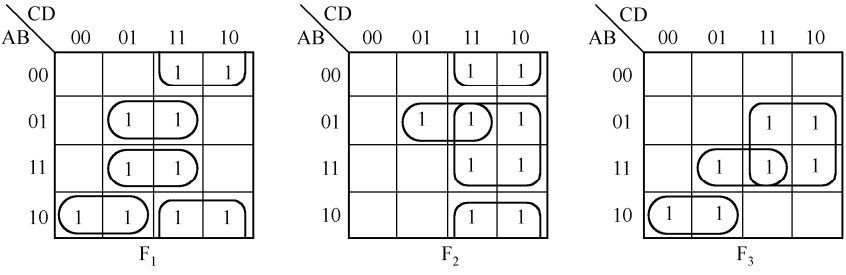


图 1.34　例 1.36 多输出函数化简方案一

如果按照图 1.33 所示来选择公共圈方案，则化简结果为

$$F_1=\overline{B}C+\overline{A}BD+ABD+A\overline{B}\overline{C} \qquad F_2=\overline{B}C+\overline{A}BD+BC \qquad F_3=ABD+A\overline{B}\overline{C}+BC$$

这是一次试圈选，是否为最佳方案还需进一步考察。由图 1.33 看出，m_5 和 m_7、m_8 和 m_9、m_{13} 和 m_{15}

分别圈出来是可以肯定的。因为 F_1 中若把 m_5、m_7、m_{13}、m_{15} 圈在一起，把 m_8、m_9、m_{11}、m_{10} 圈在一起，将使整体电路增加两个门。值得进一步考察的是 F_2 中的 m_2、m_3、m_6、m_7、m_{10}、m_{11}、m_{14}、m_{15} 这八个最小项的圈法，如将这八个最小项圈在一起，合并的结果只剩下一个变量 C，它在电路中不需要逻辑门，而且使 F_2 的或门只有两个输入端。如按图 1.34 的圈法，把 F_2 中这八个最小项拆成两个圈，分别与 F_1、F_3 取得公共圈，这在电路中虽然也不需要增加逻辑门，但 F_2 中因有三个乘积项，使得 F_2 的或门就需要三个输入端。因此可以确定，F_2 中的这八个最小项应当圈在一起，从而得到如图 1.35 所示的圈选方案。化简结果为

$$F_1 = \bar{B}C + \bar{A}BD + ABD + A\bar{B}\bar{C} \qquad F_2 = C + \bar{A}BD \qquad F_3 = ABD + A\bar{B}\bar{C} + BC$$

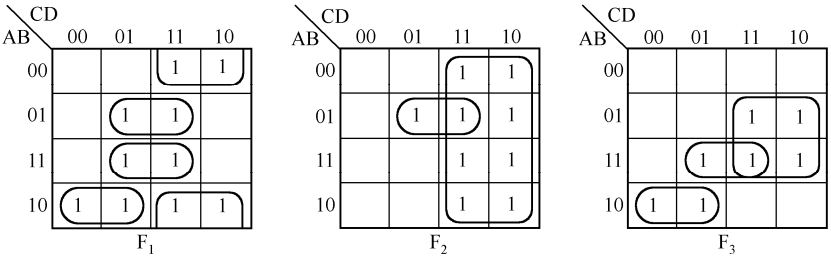


图 1.35　例 1.36 多输出函数化简方案二

相应的逻辑图如图 1.36 所示。

从此例可以得出结论，凡是单个函数中能化简为只含一个变量的积项（1 方格正好占了整张卡诺图的一半）的，不宜再将它拆开来同其他函数构成公共项。总之，从多输出函数卡诺图上找出来的公共圈能否加以利用，完全以能否使整体电路简化为准，如能简化电路，则应充分利用。

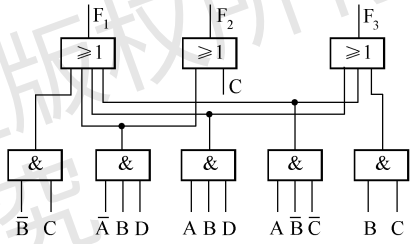


图 1.36　例 1.36 的逻辑图

通过以上举例，可以归纳出用卡诺图化简多输出函数的步骤如下：

（1）分别画出各个函数的卡诺图；

（2）在各个卡诺图中寻找两个或两个以上函数的公共圈，对非共同部分，仍按一般卡诺图的圈选原则进行圈选；

（3）进一步考察步骤（2），考察时着眼点放在公共圈上，因为有些公共圈在某个函数的卡诺图中，有可能扩大合并范围，其结果有可能使整体电路更加简化。这就需要多次修改圈选方案，并做反复比较，从中选出使整体电路最简的方案。

用卡诺图化简多输出函数的步骤也可以反过来，先分别化简各函数，再找公共圈，并做分析比较，找出使整体电路最简的方案。

复习思考题

R1.1　怎样将二进制数转换为 8421BCD 码？

R1.2　试写出 3 位和 5 位格雷码的顺序编码。

R1.3　二进制正、负数的原码、反码和补码三者之间是什么关系？

R1.4　代入定理中对代入逻辑式的形式和复杂程度有无限制？

R1.5　在真值表中，改变输入取值的排列顺序，对逻辑函数的输出有无影响？

R1.6　将一个无关项是否写入逻辑函数式，对函数的输出有无影响？

R1.7　在化简多输出逻辑函数时，怎样才能得到整体电路的最简方案？

习题

1.1 将下列二进制数转换为等值的十进制数。

（1）$(11011)_2$ （2）$(10010111)_2$ （3）$(1101101)_2$ （4）$(11111111)_2$

（5）$(0.1001)_2$ （6）$(0.0111)_2$ （7）$(11.001)_2$ （8）$(101011.11001)_2$

1.2 将下列十进制数转换为等值的二进制数（取小数点后 6 位）。

（1）$(49)_{10}$ （2）$(52.625)_{10}$ （3）$(2.168)_{10}$ （4）$(67.9)_{10}$

1.3 将下列二进制数转换为等值的十六进制数和八进制数。

（1）$(1010111)_2$ （2）$(110011010)_2$ （3）$(10110.011010)_2$ （4）$(101100.110011)_2$

1.4 将下列数转换为等值的二进制数。

（1）$(8C)_{16}$ （2）$(3D.BE)_{16}$ （3）$(8F.FF)_{16}$ （4）$(10.0)_{16}$

（5）$(136.45)_8$ （6）$(372)_8$

1.5 将下列十进制数表示为 8421BCD 码。

（1）$(43)_{10}$ （2）$(95.12)_{10}$ （3）$(67.58)_{10}$ （4）$(932.1)_{10}$

1.6 将下列 BCD 码转换为十进制数。

（1）$(010101111001)_{8421BCD}$ （2）$(10001001.01110101)_{8421BCD}$ （3）$(010011011011)_{2421BCD}$

（4）$(11001101.11100010)_{2421BCD}$ （5）$(010011001000)_{5421BCD}$ （6）$(001110101100.1001)_{5421BCD}$

（7）$(10001011)_{\text{余3BCD}}$ （8）$(10100011.01110110)_{\text{余3BCD}}$

1.7 将下列有符号的十进制数表示成补码形式的有符号二进制数。

（1）+13 （2）−9 （3）+3 （4）−8

1.8 下列是二进制数的补码形式，求它们的十进制值。

（1）01100 （2）11010 （3）10001

1.9 用真值表证明下列各式相等。（查阅本题题解请扫描二维码 1-1）

（1）$A\overline{B}+B+\overline{A}B=A+B$ （2）$A(B\oplus C)=(AB)\oplus(AC)$

（3）$\overline{A\overline{B}+C}=(\overline{A}+B)\overline{C}$ （4）$\overline{AB+\overline{A}\overline{C}}=A\overline{B}+\overline{A}\overline{C}$

二维码 1-1

1.10 写出下列逻辑函数的对偶式 F′ 及反函数 $\overline{F}$。

（1）$F=\overline{A}\overline{B}+CD$ （2）$F=\left[(A\overline{B}+C)D+E\right]G$

（3）$F=\overline{A\overline{B}+C}+\overline{A+\overline{B}\overline{C}}$ （4）$F=\overline{A+B+\overline{C}+\overline{D+E}}$

1.11 将下列逻辑函数化为最小项之和及最大项之积的形式。并求 $\overline{F}$ 和 F′ 的最小项之和的形式。

（1）$F=\overline{A}BC+A$ （2）$F=\overline{A\overline{C}+BC}$ （3）$F=(A+\overline{B})(A+C)$ （4）$F=\overline{(B+\overline{C})(\overline{A}+\overline{B})}$

1.12 用逻辑代数公式将下列逻辑函数化成最简与或表达式。

（1）$F=A\overline{B}+A\overline{C}+BC+A\overline{C}\overline{D}$ （2）$F=(A+\overline{A}C)(A+CD+D)$ （3）$F=\overline{B}\overline{D}+\overline{D}+D(B+C)(\overline{A}\overline{D}+\overline{B})$

（4）$F=\overline{A}\overline{B}\overline{C}+AD+(B+C)D$ （5）$F=\overline{\overline{AC}+\overline{B}\overline{C}+B(A\oplus C)}$ （6）$F=\overline{(A\oplus B)\cdot(B\oplus C)}$

1.13 用卡诺图将下列逻辑函数化成最简与或表达式。（查阅本题题解请扫描二维码 1-2）

（1）$F=ABC+ABD+\overline{C}\overline{D}+A\overline{B}C+\overline{A}C\overline{D}$ （2）$F=A\overline{B}\overline{C}+\overline{A}\overline{B}+\overline{A}D+C+BD$

（3）$F=\overline{B\overline{C}D+A\overline{D}(B+C)}$ （4）$F=\overline{(AB+CD)(BC+AD)}+\overline{A}BC$

（5）$F=(AB+\overline{A}C+\overline{B}D)\oplus(A\overline{B}\overline{C}D+\overline{A}CD+BCD+\overline{B}C)$ （6）$F(A,B,C)=\sum m(0,2,4,6,7)$

（7）$F(A,B,C)=\sum m(0,1,5,6,7)$ （8）$F(A,B,C,D)=\sum m(3,4,5,7,9,13,14,15)$

（9）$F(A,B,C,D)=\sum m(0,1,3,5,6,7,11,13)$ （10）$F(A,B,C,D)=\sum m(0,1,2,3,4,5,6,7,8,9,10,12,)$

（11）$F(A,B,C,D)=\sum m(0,1,2,5,8,9,10,12,14)$ （12）$F(A,B,C,D)=\sum m(0,4,5,7,8,10,14,15)$

二维码 1-2

1.14　用卡诺图将下列逻辑函数化成最简与或表达式。

（1）$F=(A\oplus B)C\overline{D}+\overline{A}B\overline{C}+\overline{A}\overline{C}D$　　且 $AB+CD=0$

（2）$F=\overline{A}C+A\overline{B}$　　且 A,B,C 不能同时为 0 或同时为 1

（3）$F(A,B,C)=\sum m(3,5,6,7)+\sum d(2,4)$

（4）$F(A,B,C,D)=\sum m(0,4,6,8,13)+\sum d(1,2,3,9,10,11)$

（5）$F(A,B,C,D)=\sum m(0,1,8,10)+\sum d(2,3,4,5,11)$

（6）$F(A,B,C,D)=\sum m(3,5,8,9,10,12)+\sum d(0,1,2,13)$

1.15　将下列逻辑函数化简为与非与非式。

（1）$F=AB+BC+AC$

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

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

（4）$F=A\overline{BC}+\overline{\overline{A\overline{B}}+\overline{A}\overline{B}+BC}$

1.16　将下列逻辑函数化简为或非或非式。

（1）$F=A\overline{B}C+B\overline{C}$

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

（3）$F=\overline{(AB\overline{C}+\overline{B}\overline{C})\overline{D}}+\overline{A}\overline{B}D$

（4）$F(A,B,C,D)=\sum m(0,2,3,8,9,10,11,13)$

1.17　用卡诺图将下列逻辑函数化为整体最简的与或表达式。

（1）$\begin{cases}F_1=A\overline{C}+\overline{A}C+\overline{B}C\\F_2=A\overline{B}+B\overline{C}+\overline{A}B\end{cases}$

（2）$\begin{cases}F_1=A\overline{B}\overline{C}D+AC\overline{D}+CD\\F_2=ABC\overline{D}+\overline{A}\overline{B}CD+A\overline{B}\overline{C}+BCD+\overline{C}\overline{D}\end{cases}$

（3）$\begin{cases}F_1(A,B,C)=\sum m(0,1,3)\\F_2(A,B,C)=\sum m(3,5,7)\\F_3(A,B,C)=\sum m(3,4,5,6,7)\end{cases}$