模块一

基础篇

项目一　数字信号与数字电路

一、模拟信号与数字信号

在自然界中有形形色色的物理量，它们性质各异，就其变化规律的特点而言，可分为两大类：数字量和模拟量。

模拟量是指物理量的变化在时间和数量上都是连续的，这种物理量一般是指模拟真实世界的物理量，如连续发出的声音、持续的光照、我们周围的温度等都在时间和空间上是连续的，处理模拟量和模拟信号的电路称为模拟电路。用波形表示时，模拟信号是一条连续的曲线，处理这类信号时，考虑的是放大倍数、频率失真、非线性失真、相位失真等，着重分析波形的形状、幅度和频率如何变化。

数字量是指物理量的变化在时间和数量上是离散的，也就是说它们的变化在时间上是不连续的，同时它们的数值每次变化都是某一最小数量单位的整数倍，处理数字量和数字信号的电路称为数字电路。用波形表示时，数字信号是一系列高、低电平组成的脉冲波，即信号总是在高电平和低电平之间来回变化，在这里，重要的是能正确区分出信号的高、低电平，并正确反应电路的输出、输入之间关系，至于高、低电平的数值精确为多少则无关紧要。

二、数字电路

（一）数字电路的特点

（1）电路结构简单，稳定可靠。数字电路只要能区分高电平和低电平即可，对元件的精度要求不高，因此有利于实现数字电路集成化。

（2）数字电路抗干扰能力强，不易受外界干扰影响。因为数字信号是采用高、低电平二值信号进行传递的。

（3）数字电路可以完成数值和逻辑两种运算，因此数字电路又称为数字逻辑电路或数字电路与逻辑设计。

（4）数字电路中的元件处于开关状态，功耗较小。

（5）数字电路具有体积小、重量轻、可靠性高、便于集成化、价格便宜等特点。

由于数字电路具有上述特点，故发展十分迅速，在计算机、数字通信、自动控制、数字仪器及家用电器等技术领域中得到广泛的应用。

（二）数字电路的分类

（1）按电路组成结构分为分立元件和集成电路两大类。其中集成电路按集成度（在一块硅片上包含的逻辑门电路或元件的数量）可分为小规模（SSI）、中规模（MSI）、大规模（LSI）和超大规模（VLSI）集成电路，如表 1.1.1 所示。

表 1.1.1　集成电路分类

集成电路分类	集成度	电路规模与范围
小规模集成电路 SSI	1～10 个门/片或 10～100 个元件/片	逻辑单元电路 包括：逻辑门电路、集成触发器
中规模集成电路 MSI	10～100 个门/片或 100～1000 个元件/片	逻辑功能部件 包括：译码器、编码器、选择器、计数器、寄存器、比较器等
大规模集成电路 LSI	>100 个门/片或>1000 个元件/片	数字逻辑系统 包括：中央处理器、存储器、串并行接口电路等
超大规模集成电路 VLSI	>1000 个门/片或>10 万个元件/片	高集成度的数字逻辑系统 例如：在一个硅片上集成一个完整的微型计算机

（2）按电路所用器件分为双极型（如 TTL、ECL、I2L、HTL）和单极型（如 NMOS、PMOS、CMOS）电路。

（3）按电路逻辑功能分为组合逻辑电路和时序逻辑电路。

项目二　数制与码制

一、数制

用数字量表示物理量的大小时，仅用一位数码往往不够，因此经常需要用进位技术的方法组成多位数码使用，把多位数码中每一位的构成方法以及从低位到高位的进位规则称为数制。日常生活中习惯用的数制是十进制，而在数字信号处理系统中进行数字的运算和处理，采用的是二进制数、八进制数、十六进制数。

（一）十进制数

十进制数是人们日常生活中最常使用的计数进位制。在这种计数进位制中，每一位由 0～9 十个数码中的一个组成，计数基数为 10，按照“逢十进一”的规律排列起来，表示数的大小。

例如：$152.25=1\times10^{2}+5\times10^{1}+2\times10^{0}+2\times10^{-1}+5\times10^{-2}$

因此，对任一个十进制的正整数$[N]_{10}$可以表示为

$$[N]_{10}=K_{n-1}\times10^{n-1}+K_{n-2}\times10^{n-2}+\cdots+K_{1}\times10^{1}+K_{0}\times10^{0}=\sum_{i=0}^{n-1}K_{i}\times10^{i} \tag{1.2.1}$$

式中：K_i 为第 i 位的系数，它是 0～9 十个数码中任意一个；10^i 为第 i 位的权，这个表达式也就是数字的加权系数之和的形式。

（二）二进制数

在数字电路中应用最广的则是二进制，因为构成计数电路的基本想法是把电路的状态与数码对应起来，十进制数需要 10 个数码，要找到区分 10 种状态的器件与之对应，是十分困难的，但要找到能够区分两种状态的器件就很多。例如灯泡的亮与灭；开关的接通与断开；晶体管的饱和与截止等。二进制中每位由 0 和 1 两个数码组成，计数的基数是 2，计数规律是“逢二进一”即 1+1=10（读作“壹零”）。二进制数各位的权为 2 的幂。

所以，一个 n 位二进制数$[N]_2$ 的按权展开式为

$$[N]_2 = K_{n-1}\times 2^{n-1} + K_{n-2}\times 2^{n-2} + \cdots + K_1\times 2^1 + K_0\times 2^0 = \sum_{i=0}^{n-1} K_i \times 2^i \qquad (1.2.2)$$

式中：K_i 为第 i 位的系数，它是 0 或 1 两个数码中任意一个；2^i 为第 i 位的权。

用这个方法也就计算出它表示的十进制数的大小。

例如：$[10111.1]_2=[1\times 2^4+0\times 2^3+1\times 2^2+1\times 2^1+1\times 2^0+1\times 2^{-1}]_{10}=(16+0+4+2+1+0.5)_{10}=(23.5)_{10}$

从上例可以看出，采用二进制数便于机器识别和运算，但位数太长，人们既难记忆，又不便于读写。所以在数字系统中为了便于读写，有时用八进制或十六进制数表示二进制数。

（三）八进制数

八进制数的基数是 8，每位采用 0～7 八个数码。计数规律是“逢八进一”。八进制数各位的权为 8 的幂。

所以，一个 n 位八进制数$[N]_8$ 的按权展开式为

$$[N]_8 = K_{n-1}\times 8^{n-1} + K_{n-2}\times 8^{n-2} + \cdots + K_1\times 8^1 + K_0\times 8^0 = \sum_{i=0}^{n-1} K_i \times 8^i \qquad (1.2.3)$$

例如：$[234]_8=[2\times 8^2+3\times 8^1+4\times 8^0]_{10}=[128+24+4]_{10}=[156]_{10}$

（四）十六进制数

十六进制数的基数是 16，采用 0～9 和 A～F 十六个数码。其中，A 到 F 表示 10 到 15。计数规律是“逢十六进一”。十六进制数各位的权为 16 的幂。

所以，一个 n 位十六进制数$[N]_{16}$ 的按权展开式为

$$[N]_{16} = K_{n-1}\times 16^{n-1} + K_{n-2}\times 16^{n-2} + \cdots + K_1\times 16^1 + K_0\times 16^0 = \sum_{i=0}^{n-1} K_i \times 16^i \qquad (1.2.4)$$

例如：$[9C.3]_{16}=[9\times 16^1+12\times 16^0+3\times 16^{-1}]_{10}=[156.1875]_{10}$

二、数制的转换

（一）二进制数、八进制数、十六进制数转换为十进制数

在进行转换时只要将二进制数、八进制数、十六进制数按权展开形式展开，然后把各项的数值按十进制数进行相加，就可以得到等值的十进制数。

例如，二—十进制转换：

$[101.01]_2=[1\times 2^2+0\times 2^1+1\times 2^0+1\times 2^{-2}]_{10}=(4+0+1+0.25)_{10}=(5.25)_{10}$

（二）十进制数转换为二进制数

1. 整数部分的转换（除二取余法）

方法：把十进制数逐次地用 2 除，取余数，一直除到商数为零。然后将每次所得到的余数从后向前排列倒序读出，即为所求的二进制整数。

【例 1.2.1】 将十进制整数$[75]_{10}$转换为二进制数。

解

		余数	
2	75		
2	37	……1	最低位
2	18	……1	
2	9	……0	
2	4	……1	
2	2	……0	
2	1	……0	
	0	……1	最高位

所以$[75]_{10}=[1001011]_2$

2. 小数部分的转换（乘二取整法）

方法：用 2 逐次乘以十进制数小数，取其整数，直到小数为 0 或达到转换所要求的精度为止。然后将所得的整数从高到低正序读出。

【例 1.2.2】 将十进制小数$[0.875]_{10}$转换为二进制数。

解

0.875		
× 2		
1.750	……1	最高位
× 2		
1.500	……1	
× 2		
1.000	……1	最低位

所以，$[0.875]_{10}=[0.111]_2$

十进制数转换为八进制数、十六进制数过程比较繁琐，一般先将十进制数转换为二进制数，再转换为八进制数、十六进制数。

3. 二进制数转换为八进制数

由于 3 位二进制数恰好有 8 个状态，因此把 3 位二进制数看作一个整体，恰好是逢八进一，即可以看作一位八进制数。转换时将二进制数从小数点开始，分别向两侧每 3 位一组，若整数最高位不足一组，在左边加 0 补足一组，小数最低位不足一组，在右边加 0 补足一组，然后将每组二进制数转换为八进制数，组成的数既是该数的八进制数。反之可以把一位八进制数视为 3 位二进制数。

【例 1.2.3】 将二进制数$[1101101010.0110101]_2$转换为八进制数。

解 $[1101101010.0110101]_2=[001/101/101/010.011/010/100]_2=[1552.324]_8$

【例 1.2.4】 将八进制数$[236.74]_8$转换为二进制数。

解 $[236.74]_8=[010011110.111100]_2=[10011110.1111]_2$

4. 二进制数转换为十六进制数

由于 4 位二进制数恰好有 16 个状态，因此把 4 位二进制数看作一个整体，恰好是逢十六进一，即可以看作一位十六进制数。转换时将二进制数从小数点开始，分别向两侧每 4 位一组，若整数最高位不足一组，在左边加 0 补足一组，小数最低位不足一组，在右边加 0 补足一组，然后将每组二进制数转换为十六进制数，组成的数既是该数的十六进制数。反之可以把一位十六进制数视为 4 位二进制数。

【例 1.2.5】 将二进制数$[1101101010.0110101]_2$ 转换为十六进制数。

解 $[1101101010.0110101]_2=[0011/0110/1010.0110/1010]_2=[36A.6A]_{16}$

【例 1.2.6】 将十六进制数$[A6C.63]_{16}$ 转换为二进制数。

解 $[A6C.63]_{16}=[101001101100.01100011]_2$

三、常用编码

不同的数码不仅可以表示数量的不同大小，而且还能用来表示不同的事物。在后一种情况下，这些数码已没有表示数量大小的含意，只是表示不同事物的代号而已，这些数码称之为代码。

例如，在举行长跑比赛时，为便于识别运动员，通常给每个运动员编一个号码。显然这些号码仅仅表示不同的运动员，已失去了数量大小的含意。

在数字电路中，我们常用二进制数表示各种文字、符号等信息，这样的过程叫做编码，用来进行编码之后的二进制数码称为二进制代码。编制代码要遵循一定的规则，规则不同，编码的形式也就很多，这里只介绍常见的二一十进制（BCD）码和格雷码。

（一）二一十进制（BCD）码

二一十进制（BCD）码，是用 4 位二进制数表示一位十进制数的编码方式。因为 4 位二进制代码有 $2^4=16$ 种状态组合，若从中取出 10 种组合表示 0～9 可以有多种方式。因此 BCD 码有多种。表 1.2.1 列出几种常用的二一十进制（BCD）码。

表 1.2.1 几种常用的二一十进制（BCD）码

十进制数	有权码					无权码
	8421 码	5421 码（a）	5421 码（b）	2421 码（a）	2421 码（b）	余 3 码
0	0000	0000	0000	0000	0000	0011
1	0001	0001	0001	0001	0001	0100
2	0010	0010	0010	0010	0010	0101
3	0011	0011	0011	0011	0011	0110
4	0100	0100	0100	0100	0100	0111
5	0101	0101	1000	0101	1011	1000
6	0110	0110	1001	0110	1100	1001
7	0111	0111	1010	0111	1101	1010
8	1000	1011	1011	1110	1110	1011
9	1001	1100	1100	1111	1111	1100

1. 8421 码

8421 码是一种有权代码，是使用最多的一种编码，在用 4 位二进制数码表示一位十进制数时，每一位二进制数的权从高位到低位依次为 8、4、2、1。

【例 1.2.7】 将十进制数$[168]_{10}$用 8421BCD 码表示。

解 $[168]_{10}=[000101101000]_{8421BCD}=[101101000]_{8421BCD}$

【例 1.2.8】 将 8421BCD 码[10000111.0110]用十进制数表示。

解 $[10000111.0110]_{8421BCD}=[87.6]_{10}$

2. 5421 码

5421 码也是一种有权代码，在用 4 位二进制数码表示一位十进制数时，每一位二进制数的权从高位到低位依次为 5、4、2、1。这种编码有两种形式 5421（a）码和 5421（b）码，见表 1.2.1。

【例 1.2.9】 将十进制数 168 用 5421（b）BCD 码表示。

解 $[168]_{10}=[000110011011]_{5421\text{（b）BCD}}$

3. 2421 码

2421 码也是一种有权代码，在用 4 位二进制数码表示一位十进制数时，每一位二进制数的权从高位到低位依次为 2、4、2、1。因此它也有两种编码方式，分为 2421 码（a）和 2421 码（b），见表 1.2.1。

4. 余 3 码

余 3 码是一种无权代码，也是 4 位二进制数码表示，与 8421 码相比，对应同样的十进制数，但多出$[0011]_2$，即$[3]_{10}$，因此称余 3 码。

【例 1.2.10】 将十进制数 168 用余 3 码表示。

解 $[168]_{10}=[010010011011]_{余3码}$

（二）格雷码

格雷码是一种无权码，即各位表示的 0 和 1 已经没有固定的权值，其优点是任意两个相邻的码只有一位不同，其余的各位数码均相同。

一位格雷码与一位二进制数码相同，是 0 和 1。由一位格雷码得到两位格雷码的方法是将一位格雷码的 0 和 1 以虚线为轴折叠，反射出 1、0，然后在虚线上方的数字前面加 0，虚线下方数字前面加 1，便得到了两位格雷码 00、01、11、10，分别表示十进制数 0～3。同样的方法可以得到 3 位、4 位格雷码，如图 1.2.1 所示。以此类推，由此反射循环的过程生成各位的格雷码，因而格雷码又称反射循环码。

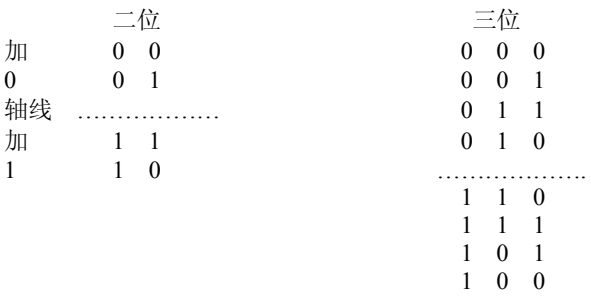


图 1.2.1 格雷码

项目三　逻辑代数基础

一、概述

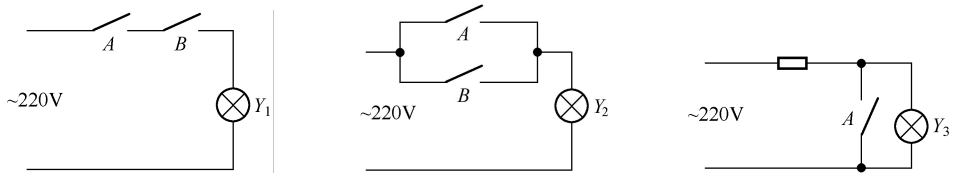
数字电路主要研究电路输入、输出状态之间的相互关系，即逻辑关系。分析和设计数字电路的数学工具是逻辑代数，它是英国数学家布尔于 1849 年提出的，也称布尔代数。

逻辑代数是分析和设计数字电路的一个数学工具，是学习数字电路的基础。因为在数字电路中，一位二进制数码的 0 和 1 不仅可以表示数量的大小，而且可以表示两种不同的逻辑状态。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，这种运算成为算术运算；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所谓的逻辑运算。

1894 年，英国数学家乔·布尔（George Boole）首先提出了描述客观事物逻辑关系的数学方法：布尔代数。后来由于布尔代数被广泛地应用于解决开关电路和数字逻辑电路的分析和设计中，所以又把布尔代数叫做开关代数或逻辑代数，和普通代数有一个共同的特点，就是都用英文字母 *A*、*B*、*C*、…、*X*、*Y*、*Z* 等表示变量，如：*Y*=*F*(*A*、*B*、……)；不同之处在于在逻辑代数中，变量的取值范围仅为 0 和 1 两个值，没有第三种可能，这种变量称为“逻辑变量”。但是在逻辑代数中，0 和 1 不再表示具体的数量大小，而只是表示两种不同的逻辑状态，如灯的亮或灭，开关的开或关，电压的高或低，晶体管的饱和或截止，事件的是或非等。除此以外逻辑代数和普通代数的运算规则也不完全相同。

二、逻辑代数基本运算

逻辑代数的基本逻辑关系有与、或、非三种，为便于理解它们的含意，就以图 1.3.1 中 3 个指示灯的控制电路为例来说明。在图 1.3.1（a）中只有当两个开关同时闭合时，指示灯才会亮；在图 1.3.1（b）电路中，只要有任意一个开关闭合，指示灯就亮；而在图 1.3.1（c）中，开关断开时灯亮，开关闭合时反而不亮。



（a）　（b）　（c）

图 1.3.1　逻辑代数的基本运算举例

如果把开关闭合作为条件（或导致事物结果的原因），把灯亮作为结果，那么图 1.3.1 中的 3 个电路就代表了 3 种不同的因果关系：

图 1.3.1（a）表明，只有决定事物结果的全部条件同时具备时，结果才发生。这种因果关系叫做逻辑与，或者叫逻辑相乘。

图 1.3.1（b）表明，只要决定事物结果的诸多条件中有任何一个满足时，结果就会发生。这种因果关系叫做逻辑或，或者叫逻辑相加。

图 1.3.1（c）表明，只要条件具备了，结果便不会发生，而条件不具备时，结果一定发生。这种因果关系叫做逻辑非，或者叫逻辑求反。

图 1.3.1 对应的功能表如表 1.3.1 所示。

表 1.3.1　图 1.3.1 电路的功能表

开关 A	开关 B	灯 Y_1	灯 Y_2	灯 Y_3
断开	断开	灭	灭	亮
断开	闭合	灭	亮	
闭合	断开	灭	亮	灭
闭合	闭合	亮	亮	

若以 A、B 表示开关的状态，1 表示开关闭合，0 表示开关断开；以 Y 表示指示灯的状态，并以 1 表示灯亮，以 0 表示不亮，则可以列出以 0、1 表示的与或非逻辑关系的图表，如表 1.3.2 所示，这种图表叫做逻辑真值表，简称为真值表。

表 1.3.2　图 1.3.1 电路的真值表

开关 A	开关 B	灯 Y_1	灯 Y_2	灯 Y_3
0	0	0	0	1
0	1	0	1	
1	0	0	1	0
1	1	1	1	

（一）与逻辑、与运算、与门

与运算也称逻辑乘。与运算的逻辑表达式为

$$Y = A \cdot B \tag{1.3.1}$$

或 $Y = AB$ （“·”号可省略）。

与逻辑的运算规律为：输入有 0 得 0，全 1 得 1。

在数字电路中，用来完成与逻辑运算的电路叫与门。逻辑符号如图 1.3.2 所示。一个与门电路有两个或两个以上的输入端，只有一个输出端。符号“&”表示与逻辑运算。

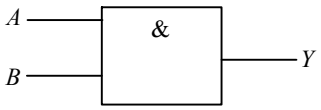


图 1.3.2　与逻辑符号

（二）或逻辑、或运算、或门

或运算也称逻辑加。或运算的逻辑表达式为

$$Y = A + B \tag{1.3.2}$$

或逻辑运算的规律为：有 1 得 1，全 0 得 0。

在数字电路中，用来完成或逻辑运算的电路叫或门。逻辑符号如图 1.3.3 所示。一个或门电路有两个或两个以上的输入端，只有一个输出端。符号“≥1”表示或逻辑运算。

（三）非逻辑、非运算、非门

非运算也称反运算。非运算的逻辑表达式为

$$Y = \overline{A} \tag{1.3.3}$$

非逻辑运算的规律为：0 变 1，1 变 0，即“始终相反”。

实现非逻辑运算的电路叫非门，逻辑符号如图 1.3.4 所示。逻辑符号中用小圆圈代表“非”，“1”表示缓冲。

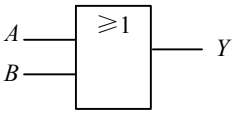


图 1.3.3　或逻辑符号

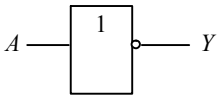


图 1.3.4　非逻辑符号

上述与门、或门、非门 3 种基本电路，是数字电路系统中最基础的逻辑元件。除此之外，将“与”、“或”、“非”3 种基本运算加以组合，还可扩展出“与非”逻辑、“或非”逻辑、“与或非”逻辑、“异或”逻辑、“同或”逻辑等多种逻辑运算。此类扩展的逻辑电路也在数字电路系统中有着广泛的应用。

（四）与非逻辑

“与非”逻辑运算是由“与”和“非”两种逻辑运算复合而成的一种复合逻辑运算，逻辑表达式为

$$Y = \overline{AB} \tag{1.3.4}$$

实现“与非”逻辑运算的电路叫“与非”门，逻辑符号如图 1.3.5 所示。

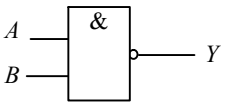


图 1.3.5　与非门逻辑符号

由表 1.3.3 与非逻辑真值表可见，只要输入变量 A、B 中有一个为 0，函数 Y 就为 1，只有输入 A、B 全为 1，输出 Y 才为 0。

表 1.3.3　与非逻辑真值表

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

（五）或非逻辑

“或非”逻辑运算是“或”和“非”两种逻辑运算复合而成的一种复合逻辑运算。其逻辑函数为

$$Y = \overline{A + B} \tag{1.3.5}$$

由真值表 1.3.4 可见，只要变量 A、B 有一个为 1，函数 Y 就为 0，只有 A、B 全部为 0 时，输出 Y 才为 1。

表 1.3.4　或非门逻辑真值表

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

实现“或非”逻辑运算的电路叫“或非”门，逻辑符号如图 1.3.6 所示。

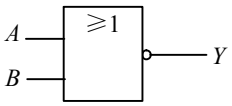
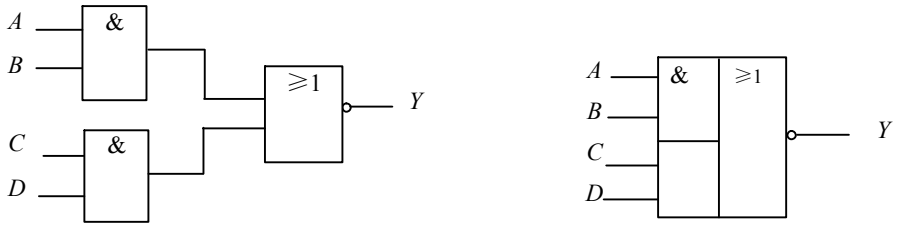


图 1.3.6　或非门逻辑符号

（六）与或非逻辑

“与或非”逻辑运算是“与”、“或”和“非”3 种逻辑运算复合而成的一种复合逻辑运算，实现与或非逻辑运算的门电路称为与或非门，如图 1.3.7 所示。



（a）与或非逻辑图　　（b）逻辑符号

图 1.3.7　与或非门

与或非逻辑的函数表达式为

$$Y=\overline{AB+CD} \tag{1.3.6}$$

与或非逻辑的真值表可由式（1.3.6）得出, 如表 1.3.5 所示。由真值表得出与或非逻辑的逻辑功能是：当输入端的任何一组全为 1 时，输出为 0；只有任何一组输入有 0 时，输出端才为 1。

表 1.3.5　与或非门逻辑真值表

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1

续表

A	B	C	D	Y
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

（七）异或逻辑和同或逻辑

在决定事件发生的各种条件中，有奇数个条件具备时事件就会发生，这种因果关系叫异或逻辑运算；有偶数个条件具备时事件就会发生，这种因果关系叫同或逻辑运算。异或逻辑和同或逻辑互为反函数，因而同或逻辑运算又称为异或非逻辑运算。它们的逻辑符号如图 1.3.8 所示。

（a）异或逻辑符号　　（b）同或逻辑符号

图 1.3.8　同或、异或门电路

异或逻辑函数表达式为

$$Y_1 = A\overline{B} + \overline{A}B\text{，通常写作 } Y_1 = A \oplus B \tag{1.3.7}$$

同或逻辑函数表达式为

$$Y_2 = AB + \overline{A}\,\overline{B}\text{，通常写作 } Y_2 = A \odot B \tag{1.3.8}$$

符号“⊕”表示异或运算，符号“⊙”表示同或运算。根据公式（1.3.7）和（1.3.8），可简单归纳异或逻辑的特性为“输入变量同态为 0、异态为 1”，反之同或逻辑的特性为“输入变量同态为 1、异态为 0”。其真值表如表 1.3.6 所示，查表可更直观的反映此类运算的特性。

表 1.3.6　异或、同或逻辑真值表

A	B	Y_1	Y_2
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

三、案例分析：声光控照明电路

如图 1.3.9 所示电路中，VD_3－VD_6 为整流电路，R_7、C_3、VD_2 组成滤波电路，三极管 VT_1 从 R_2 获得正向偏置。

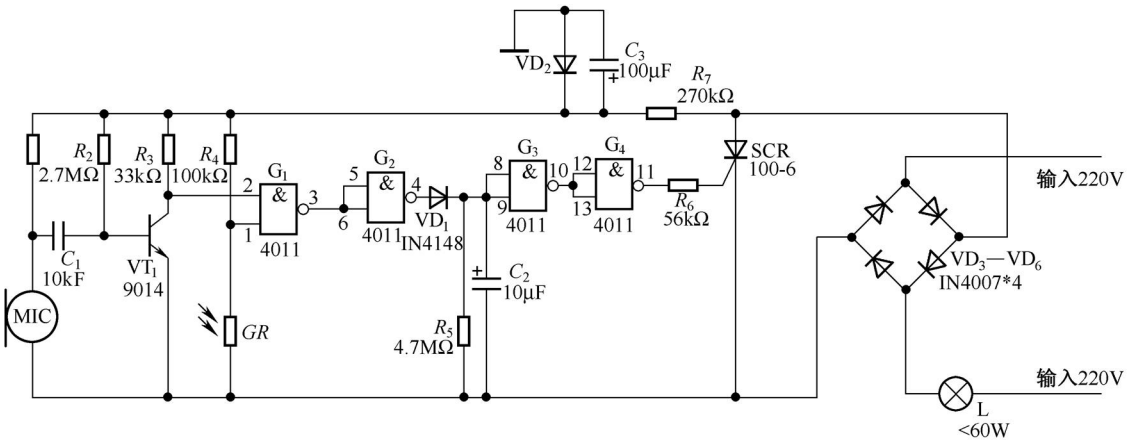


图 1.3.9　声光控照明电路

电路分析：图中 GR 为 MG45 型光敏电阻当有较强光线照射到光敏电阻 *GR* 上时，*GR* 阻值变小，与非门 G_1 的输入端 1 脚处于低电平，此时不管 2 脚输入是什么电平，其输出端 3 脚均为高电平，与非门 G_2 输出端为低电平，此时，与非门 G_3 的两个输入端为低电平，其输出端为高电平，与非门 G_4 输出端为低电平，晶闸管 SCR 是关断的，电灯 L 不亮。

当光线较弱时，光敏电阻 *GR* 阻值变大，近似于断开状态，与非门 G_1 的输入端 1 脚处于高电平状态，但如果没有声音信号，驻极体话筒 MIC 阻值较大，三极管 VT_1 的基极电位较高，VT_1 呈导通状态，与非门 G_1 的输入端 2 脚处于低电平，与非门 G_1 输出端仍高电平，电灯不亮。

只有光线较弱且有声音信号存在时，声音信号使得 MIC 产生电信号。此信号经 C_1 耦合到三极管 VT_1 的基极，使 VT_1 瞬间截止，与非门 G_1 的 2 脚输入为高电平，这时与非门 G_1 的两个输入端都是高电平，输出端为低电平，与非门 G_2 输出为高电平，使 VD_1 导通，C_2 迅速充电，与此同时，与非门 G_3 输入端为高电平，输出端为低电平，与非门 G_4 输出端为高电平，使得晶闸管 SCR 经 R_6 获得高电平而导通，电灯 L 点亮。

一般的声音信号存在时间较短，所以 VT_1 的截止时间较短，但灯 L 要求有一定的点亮时间，这就需要一定的时间延迟，其工作过程是：当声音信号消失后，与非门 G_1 输入端 2 脚恢复到低电平状态，与非门 G_1 的输出端变为高电平，与非门 G_2 的输出为低电平，VD_1 截止，这时充足了电的 C_2 开始时仍有电压，其高电位加到与非门 G_3 的输入端，使与非门 G_3 输出为低电平，与非门 G_4 输出为高电平，L 继续点亮，同时 C_2 通过 R_5 开始放电，随着时间的推移，C_2 两端的电压逐渐降低，当低到一定电平时，促使与非门 G_3 的输入端为低电平而输出变为高电平，SCR 立即关断，电灯 L 熄灭。

项目四　逻辑函数及其化简

一、概述

（一）逻辑函数及其运算顺序

在数字逻辑电路中，如果输入变量 A、B、C…的取值确定之后，输出变量 Y 的值也被唯一地确定了；或者说，如果某逻辑变量 Y 是由其他逻辑变量 A、B、C…经过有限个基本逻辑运算确定的，那么 Y 就称作是 A、B、C…的逻辑函数。逻辑函数的一般表达式可以写为 $Y=f(A、B、C…)$，它的基本运算是与、或、非。但在实际的运用中，往往是多种运算组合构成一种复杂的运算形式。

在一个逻辑函数中，往往包含有几种基本逻辑运算，在执行这些运算时应按照一定的顺序进行。逻辑运算的优先顺序规定如下：当式中有括号时，先进行括号里的运算；没有括号时，按非、与、或的次序依次进行。同或、异或运算的优先级介于与、或之间。

（二）逻辑函数相等的概念

所谓 Y_1 和 Y_2 这两个函数相等是指两个逻辑函数 Y_1 和 Y_2 都含有 n 个变量，对应 n 个输入变量的全部取值组合，输出函数 Y_1 和 Y_2 的值相等；两个相等的函数应当具有相同的真值表，这也是证明逻辑函数相等的一个最简单的方法。

【例 1.4.1】 证明 $Y_1 = A + BC$，$Y_2 = (A + B)(A + C)$ 相等。

解　将 3 个变量 A、B、C 的 8 种取值组合（2^3=8），分别代入逻辑函数表达式中进行计算，求得对应的 Y_1 和 Y_2 的值，即得表 1.4.1 所示的真值表。

表 1.4.1　例 1.4.1 真值表

A	B	C	$A+BC$	$(A+B)(A+C)$
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

由于真值表中两表达式的结果完全相同，所以 $Y_1 = Y_2$。

二、逻辑函数的表示方法

逻辑函数有多种表示方法，通常有逻辑函数真值表、表达式、逻辑图、卡诺图、波形图等 5 种。它们各有特点，而且可以相互转换。

（一）逻辑表达式

逻辑函数表达式是指用与、或、非等基本的和常用的逻辑运算来表示逻辑变量之间关系的代数式。逻辑函数表达式有多种形式，如式（1.3.1）～式（1.3.8）都是最基本的逻辑函数表达式。逻辑函数表达式简称逻辑表达式、或逻辑式、或表达式。

优点是书写简洁、方便，可以用公式和定理十分灵活的进行运算；缺点是在逻辑函数比较复杂时，难以直接从变量取值看出函数的值，没有真值表和卡诺图直观。

（二）真值表

真值表是将输入变量的各种可能取值和对应的函数值，以表格的形式一一列举出来，这种表格就叫做真值表。每一个变量均有 0、1 两种取值，n 个变量共有 2^n 种不同取值，将其按顺序(一般按二进制数递增规律)排列起来，同时在相应位置写上函数的值，便可得到逻辑函数的真值表。如项目三中就列举了很多真值表。

优点是能够直观、明了地反映了变量取值和函数值之间的对应关系。而且从实际的逻辑问题写真值表也比较容易。缺点主要是，变量多时，写真值表比较繁琐。

（三）逻辑电路图

逻辑电路图是用相应的逻辑符号将逻辑函数式的运算关系表示出来得到的图形，简称逻辑图。如图 1.3.2～图 1.3.8 就是最简单的逻辑图。

优点是逻辑符号和实际电路、器件有着明显的对应关系，能方便的按逻辑图构成实际电路图；缺点是不能用公式和定理进行运算和变换，所表示的逻辑关系没有真值表和卡诺图直观。

（四）卡诺图

卡诺图可以说是真值表的一种方块图表示形式，只不过变量取值必须按照格雷码的顺序排列而已，与真值表有严格的一一对应关系，因此也叫做真值方格图。它将在后续章节中详述。

（五）波形图

如果已知输入变量取值随时间变化的波形，就可以根据逻辑函数表达式、真值表、逻辑图表达的逻辑关系，画出输出变量随时间变化的波形。这种能反映输入变量和输出变量随时间变化的图形称为波形图，又叫时序图。如图 1.4.1 所示是表达式 $Y = AB + BC + AC$ 波形图。

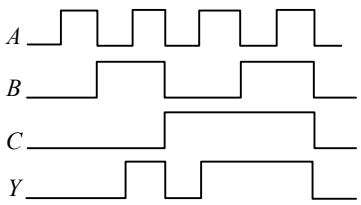


图 1.4.1　$Y = AB + BC + AC$ 波形图

波形图能直观的表达出变量和函数之间随时间变化的规律，可以帮助掌握数字电路的工作情况和诊断电路故障。

三、逻辑函数表示方法的相互转换举例

（一）真值表与表达式的转换

1. 已知逻辑函数表达式列真值表

根据逻辑函数表达式确定输入变量个数（n）以及真值表的行数（有 2^n 行），输入变量取值按

二进制数排列。然后，将每一行的输入变量取值代入逻辑函数表达式，求出输出变量的值，列在真值表的右侧即得到我们要求的真值表。

【例 1.4.2】 列出逻辑函数表达式 $Y = A\overline{B}C + B + \overline{A}C$ 的真值表。

解 因为有 3 个输入变量 A、B、C，则真值表有 $2^3 = 8$ 行，然后，将输入变量的所有组合代入逻辑函数表达式进行计算，得到真值表如表 1.4.2 所示。

表 1.4.2 例 1.4.2 真值表

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

2. 已知真值表求表达式

根据真值表中输入变量和输出变量的对应关系，先确定输出变量等于 1 的行，再将等于 1 的每一行的输入边量写成一个乘积项，每个乘积项中输入变量取值为 1 的，写成原变量，输入变量取值为 0 的，写成反变量。再将输出变量等于 1 的几个乘积项相加即得与或表达式。

【例 1.4.3】 已知真值表如表 1.4.3 所示，试写出其表达式。

表 1.4.3 例 1.4.3 真值表

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

解 $Y = \overline{A}B + A\overline{B}$

（二）逻辑图与表达式的转换

已知逻辑函数表达式画逻辑图，根据逻辑函数的表达式和对应的门电路的关系，画出相应的逻辑图。

【例 1.4.4】 已知逻辑函数表达式 $Y = B \oplus (\overline{A}C)$，试画出其逻辑图。

解 根据逻辑函数表达式，画出逻辑图如图 1.4.2 所示。

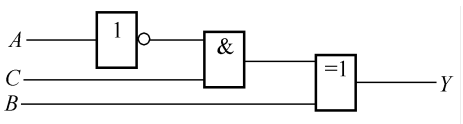


图 1.4.2 例 1.4.4 的图

已知逻辑图求逻辑函数表达式，因每一个逻辑图的输出与输入之间的关系，均可以用相应的逻辑函数表达式来表示，可以根据已知的逻辑图，从输入到输出逐级写出逻辑函数表达式即可。

四、逻辑代数运算规则

（一）逻辑代数基本公式

逻辑代数的基本公式如表 1.4.4 所示，读者可运用前述逻辑函数相等的概念加以证明。

表 1.4.4　逻辑代数基本公式

1．常量和常量的公式	$1\cdot 1=1$　$1\cdot 0=0$　$0\cdot 1=0$　$0\cdot 0=0$ $1+1=1$　$1+0=1$　$0+1=1$　$0+0=0$ $\overline{1}=0$　$\overline{0}=1$
2．常量和变量的公式	$A\cdot 1=A$　$A\cdot 0=0$　$A+0=A$　$A+1=1$
3．变量和变量的公式	（1）互补律　$A\cdot\overline{A}=0$　$A+\overline{A}=1$ （2）重叠律　$A\cdot A=A$　$A+A=A$ （3）交换律　$A\cdot B=B\cdot A$　$A+B=B+A$ （4）结合律　$(A\cdot B)\cdot C=A\cdot(B\cdot C)$　$(A+B)+C=A+(B+C)$ （5）分配律　$A\cdot(B+C)=A\cdot B+A\cdot C$　$A+B\cdot C=(A+B)(A+C)$ （6）非非律　$\overline{\overline{A}}=A$ （7）反演律（德·摩根定律）　$\overline{A\cdot B}=\overline{A}+\overline{B}$　$\overline{A+B}=\overline{A}\cdot\overline{B}$

【例 1.4.5】 证明公式 $A+B\cdot C=(A+B)(A+C)$。

证明　方法一：用真值表证明，见例 1.4.1。

方法二：用公式证明

右边＝$(A+B)(A+C)=A\cdot A+A\cdot C+B\cdot A+B\cdot C=A+B\cdot C=$左边。

【例 1.4.6】 证明反演律。

证明　反演律见表 1.4.4。用真值表证明，如表 1.4.5 所示。

表 1.4.5　例 1.4.6 的真值表

A	B	$\overline{A\cdot B}$	$\overline{A}+\overline{B}$	$\overline{A+B}$	$\overline{A}\cdot\overline{B}$
0	0	1	1	1	1
0	1	1	1	0	0
1	0	1	1	0	0
1	1	0	0	0	0

由表中可以看出，两个等式的左右两边的表达式的真值表完全相同，故反演律等式成立。

（二）逻辑代数的 3 个基本运算规则

逻辑代数中有 3 个重要的基本规则，即代入规则、反演规则和对偶规则。运用这些规则，可以利用已知的基本公式推导出更多的等式（公式）。

1. 代入规则

在任何一个逻辑等式中，如果等式两边所有出现某一变量的地方，都用某一个函数表达式代替，则等式仍然成立。

代入规则在推导公式中用处很大，因为将已知等式中某一变量用任意一个函数代替后，就得到了新的等式，从而扩大了等式的应用范围。

如已知等式 $\overline{A \cdot B} = \overline{A} + \overline{B}$。若用 $Y = BC$ 代替等式中的 B，根据代入规则，等式仍然成立。即：$\overline{A(BC)} = \overline{A} + \overline{BC}$。由此也可见，反演律对任意多个变量均成立。

2. 反演规则

对于任意一个函数表达式 Y，若将 Y 中所有的“·”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，即得原函数表达式 Y 的反函数 $\overline{Y}$。这个规则叫反演规则。

反演规则的意义在于，利用它可以比较容易的求出一个逻辑函数的反函数。

例如函数表达式为 $Y = \overline{A}B + CD$，则它的反函数为 $\overline{Y} = (A + \overline{B})(\overline{C} + \overline{D})$。

3. 对偶规则

对于任意一个函数表达式 Y，若将 Y 中所有的“·”换成“+”，“+”换成“·”；0 换成 1，1 换成 0；而变量保持不变，即得一个新的函数表达式 Y'。我们称函数 Y' 为原函数 Y 的对偶函数。实际上对偶是相互的，故 Y 也是 Y' 的对偶函数。

例如函数 $Y = AB + C$ 的对偶函数 $Y' = (A + B) \cdot C$；

函数 $Y = A + \overline{B} \cdot C$ 的对偶函数 $Y' = A \cdot (\overline{B} + C)$。

对偶规则：如果两个函数相等，则它们各自的对偶函数也相等。

在运用反演规则和对偶规则时应注意以下两点：

（1）为保证逻辑表达式的运算顺序不变，可适当增加或减少括号。

（2）长非号要保持不变。

对偶规则的用途比较广泛。利用对偶规则，可以使需要证明和记忆的公式数目减小一半。

（三）逻辑代数常用公式

运用基本公式和上述 3 个基本规则，可以得到更多的公式，如表 1.4.6 所示。

表 1.4.6　逻辑代数常用公式

1．还原律	$AB + \overline{A}B = B$	$(A + B)(\overline{A} + B) = B$
2．吸收律	$A + AB = A$	$A(A + B) = A$
3．消去律	$A + \overline{A}B = A + B$	$A(\overline{A} + B) = A \cdot B$
4．冗余律	$AB + \overline{A}C + BC = AB + \overline{A}C$ 推论：$AB + \overline{A}C + BCD = AB + \overline{A}C$	$(A + B)(\overline{A} + C)(B + C) = (A + B)(\overline{A} + C)$
5．与或转换律	$AB + \overline{A}C = (A + C)(\overline{A} + B)$	$(A + B)(\overline{A} + C) = AC + \overline{A}B$
6．异或与同或关系	$\overline{A\overline{B} + \overline{A}B} = AB + \overline{A}\,\overline{B}$	$\overline{AB + \overline{A}\,\overline{B}} = A\overline{B} + \overline{A}B$

【例 1.4.7】 证明还原律 $AB + \overline{A}B = B$。

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

【例 1.4.8】 证明吸收律　$A+AB=A$。

证明　$A+AB=A(1+B)=A$

【例 1.4.9】 证明消去律 $A+\overline{A}B=A+B$。

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

【例 1.4.10】 证明冗余律 $AB+\overline{A}C+BC=AB+\overline{A}C$ 及其推论 $AB+\overline{A}C+BCD=AB+\overline{A}C$。

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

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

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

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

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

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

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

【例 1.4.11】 证明与或转换律 $AB+\overline{A}C=(A+C)(\overline{A}+B)$。

证明　右边$=(A+C)(\overline{A}+B)=A\overline{A}+AB+C\overline{A}+BC$

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

利用以上逻辑代数的基本公式、常用公式和 3 个基本规则，可以对逻辑函数进行化简。

五、逻辑函数的公式化简法

（一）化简的意义与标准

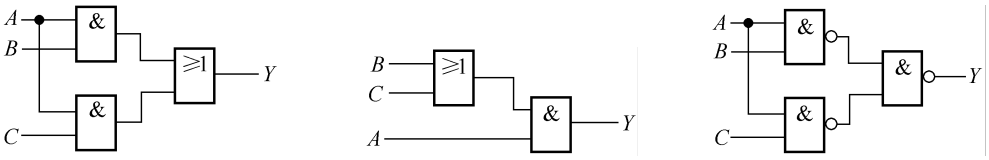
1. 化简的意义

在进行逻辑运算时常常会看到，同一个逻辑函数可以写成不同的逻辑式，而这些逻辑式的繁简程度又相差甚远。逻辑式越是简单，它所表示的逻辑关系越明显，同时也有利于用最少的电子器件实现这个函数。例如一个简单的函数表达式 $Y=AB+AC$ 就有 3 种不同的形式：

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

相应的逻辑电路也就不同，如图 1.4.3 所示。

在图 1.4.3（a）、（b）中采用与门、或门，但图 1.4.3（b）比较简单，图 1.4.3（c）全用与非门。可见，函数表达式的简繁程度不同，实现它所用的逻辑门电路也有简单与否之分。一般而言，如果逻辑函数表达式比较简单，实现它的逻辑门电路也就简单，逻辑电路所用的器件也最节省。所以，逻辑函数的化简是有重要的现实意义的。



（a）用与门实现　（b）用或门实现　（c）用与非门实现

图 1.4.3　逻辑图

2. 逻辑函数的最简表达式

一个逻辑函数的最简表达式，常按照始终变量之间运算关系不同，分成最简与或式、最简与非—与非式、最简或与式、最简或非—或非式、最简与或非式等 5 种。那么，究竟使用哪种形式的表达式，要看组成逻辑电路时使用什么形式的门电路。在进行逻辑函数的化简时，一般以与或表达式的形式为标准进行逻辑函数的化简。最后再转换成我们需要的表达式的形式。其原因有：

（1）任意一个逻辑函数表达式均可展开为与或表达式；

（2）由与或表达式可以方便的得到其他任何形式的表达式。

与或逻辑函数表达式 $Y=AB+\overline{A}C$ 向其他形式的表达式转换的结果为：

$Y=AB+\overline{A}C$（与或表达式）$=(A+C)(\overline{A}+B)$（或与表达式）$=\overline{\overline{AB}\cdot\overline{\overline{A}C}}$（与非与非表达式）

$=\overline{\overline{A+C}+\overline{\overline{A}+B}}$（或非或非表达式）$=\overline{\overline{A}\cdot\overline{C}+A\cdot\overline{B}}$（与或非表达式）

3. 最简与或表达式标准

（1）乘积项的个数最少。

（2）每个乘积项中的变量个数最少。

（二）公式化简法的主要方法

逻辑函数的公式化简法又称代数法，就是利用逻辑代数的基本公式、常用公式和 3 个重要规则，对逻辑函数进行化简。常用的方法有合并项法、吸收法、消去法和配项法。

1. 合并项法

利用公式 $AB+\overline{A}B=B$，将两个乘积项合并成一项，并消去一个变量，其中 A 和 B 都可以是任何复杂的逻辑式。

【例 1.4.12】 化简函数 $Y=A\overline{B}+ACD+\overline{AB}+\overline{A}CD$。

解 $Y=A(\overline{B}+CD)+\overline{A}(\overline{B}+CD)=(A+\overline{A})(\overline{B}+CD)=\overline{B}+CD$

2. 吸收法

利用公式 $A+AB=A$ 消去多余的乘积项 AB，A 和 B 同样也可以是任何一个复杂的逻辑式。

【例 1.4.13】 化简函数 $Y=(\overline{\overline{AB}}+C)ABD+AD$。

解 $Y=[(\overline{\overline{AB}}+C)B]AD+AD=AD$

3. 消去法

利用公式 $A+\overline{A}B=A+B$ 和公式 $AB+\overline{A}C+BC=AB+\overline{A}C$，消去多余的因子，这里边的 A、B 和 C 等都可以是任何一个复杂的逻辑式。

【例 1.4.14】 化简函数 $Y=A\overline{B}+AC+\overline{B+C}$。

解 $Y=A\overline{B}+AC+\overline{B}\,\overline{C}=AC+\overline{B}\,\overline{C}$

4. 配项消项法

（1）利用公式 $A+A=A$，可以在逻辑函数式中重复写入某一项，有时能获得更加简单的化简结果。

【例 1.4.15】 试化简逻辑函数 $Y=\overline{A}B\overline{C}+\overline{A}BC+ABC$。

解 利用 $A+A=A$ 可以将 Y 写成

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

（2）利用公式 $A+\overline{A}=1$，给某个乘积项配项；在函数式中的某一项上乘以 $A+\overline{A}$，然后拆成

两项分别与其他项合并，有时能得到更加简单的化简结果。

【例 1.4.16】 试化简逻辑函数 $Y=A\overline{B}+\overline{A}B+B\overline{C}+\overline{B}C$。

解 利用 $A+\overline{A}=1$ 可以将 Y 写成

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

在化简复杂的逻辑函数时，往往需要灵活、交替地综合运用上述几种方法进行化简，才能得到最简结果。

【例 1.4.17】 化简函数 $Y=AC+\overline{B}C+B\overline{D}+C\overline{D}+A(B+\overline{C})+\overline{A}BC\overline{D}+A\overline{B}DE$。

解 $Y=AC+\overline{B}C+B\overline{D}+A(B+\overline{C})+A\overline{B}DE+C\overline{D}+\overline{A}BC\overline{D}$（吸收法、反演律）

$=AC+B\overline{D}+C\overline{D}+A\overline{B}DE+\overline{B}C+A\overline{\overline{B}C}$　（消去法）

$=\overline{B}C+B\overline{D}+C\overline{D}+AC+A+A\overline{B}DE$　（吸收法）

$=\overline{B}C+B\overline{D}+C\overline{D}+A$　（配项消项法）

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

六、逻辑函数的卡诺图化简法

（一）逻辑函数的最小项

1. 最小项的定义

在 n 个输入变量的逻辑函数中，如果一个乘积项包含 n 个变量，而且每个变量以原变量或反变量的形式出现且仅出现一次，那么该乘积项称为该函数的一个最小项。对 n 个输入变量的逻辑函数，有 2^n 个最小项。

例如：两个变量的逻辑函数 $Y=F(A,B)$ 中，$2^2=4$，有 4 个最小项：$\overline{A}\,\overline{B}$、$\overline{A}B$、$A\overline{B}$、$AB$；3 个变量的逻辑函数 $Y=F(A,B,C)$ 中，$2^3=8$，有 8 个最小项：$\overline{A}\,\overline{B}\,\overline{C}$、$\overline{A}\,\overline{B}C$、$\overline{A}B\overline{C}$、$\overline{A}BC$、$A\overline{B}\,\overline{C}$、$A\overline{B}C$、$AB\overline{C}$、$ABC$。

2. 最小项的性质

（1）在输入变量的任何取值下必有一个最小项，而且仅有一个最小项的值为 1。

（2）任意两个不同的最小项的乘积恒为 0。

（3）对于任意一种取值全体最小项之和恒为 1。

（4）具有相邻性的两个最小项之和可以合并成一项并消去一对因子。若两个最小项之间只有一个变量不同，其余各变量均相同，则称这两个最小项是具有相邻性。对于一个 n 输入变量的函数，每个最小项有 n 个最小项与之相邻。

3. 最小项的编号

n 个输入变量的函数有 2^n 个最小项，为了表达方便，给每个最小项加以编号，记为 m_i，下标 i 即最小项编号。编号的方法：先将最小项的原变量用 1、反变量用 0 表示，构成二进制数；然后将此二进制数转换为相应的十进制数，即为该最小项的编号。

3 个变量的最小项编号如表 1.4.7 所示。