第 1 章　数字逻辑基础

1.1　概　述

模拟信号是指那些在时间上和幅值上都连续取值的电信号，处理模拟信号的电路被称之为模拟电路。

数字信号是指那些在时间上连续，幅值上离散取值的电信号，处理数字信号的电路被称之为数字电路。

与模拟器件相比，数字器件的可靠性、稳定性、抗干扰性、易用性和集成度都高得多。因而在现代电子系统中，信号的处理、数据的存储等主要是采用数字器件。

复杂的数字系统是由各种各样的小规模、中规模和大规模数字集成电路组成的。这些集成电路完成了复杂程度和逻辑功能都各不相同的逻辑运算。

1854年，乔治·布尔(George Boole) 发表了名为“*An Investigation of the Laws of Thought, on Which Are Founded the Mathematical Theories of Logic and Probabilities*”的著作，奠定了逻辑代数(又称为布尔代数)的理论基础。克劳得·香农(Claude Shannon) 把布尔代数用于由继电器组成的开关电路，在 1938 年发表了一篇名为“*A Symbolic Analysis of Relay and Switching Circuit*”的文章，开创了布尔代数在逻辑电路中的应用。

本章是数字逻辑电路的基础，将重点介绍数制、编码、逻辑代数定律和规则、逻辑函数的多种表现形式、最小项表达式、最大项表达式、逻辑函数的代数法化简和卡诺图化简等内容。

1.2　数制与编码

1.2.1　数制(Number System)

(1) 计数方法

数制是人们进行计数的方法，每一种数制都有一组特定的数码和进位规则。

在日常生活中，最常用的数制是十进制（Decimal)。它有十个基本数码(简称基数),它们分别是 0、1、2、3、4、5、6、7、8 和 9，进位规则是逢十进一。

在数字逻辑中，最常用的数制是二进制（Binary）和十六进制(Hexdecimal)。二进制只有 0 和 1 两个数码，进位规则是逢二进一。十六进制有十六个数码，它们分别是 0、1、2、3、4、5、6、7、8、9、A、B、C、D、E 和 F，进位规则是逢十六进一。

在书写数字时，常用的书写方法有位置记数法（Positional Notation）和多项式表示法(Polynomial Notation)。

位置记数法就是将数码有序地排列在一起，构成一个多位数字，且每一个数码所处的位置都有固定的权值。

为了保证不同进制的数字在书写时不发生混淆，除十进制数之外，其他任何进制的数均用括号和基数下标 R来区分。其通用记数形式为：

$$(N)_R = (d_{n-1}d_{n-2}\cdots d_i\cdots d_1d_0.d_{-1}d_{-2}\cdots d_{-m})_R$$

比如，二进制数书写为：

$$(N)_2 = (10011100011)_2$$

十六进制数书写为：

$$(N)_{16} = (5A5.3F)_{16}$$

多项式表示法就是将位于不同数位的不为 0 的数码乘以该位的权值（以十进制数表示），再用加号将各乘积项组合起来，写成多项式的形式。其通用记数式为：

$$(N)_R = d_{n-1}\times R^{n-1} + d_{n-2}\times R^{n-2} + \cdots + d_0\times R^0 + d_{-1}\times R^{-1} + \cdots + d_{-m}\times R^{-m}$$

$$= \sum_{i=-m}^{n-1} d_i \times R^i$$

【例 1.1】 将 $(100110.11)_2$ 和 $(5A5.3F)_{16}$ 用多项式表示。

【解】 $(100110.11)_2 = 1\times 2^5 + 1\times 2^2 + 1\times 2 + 1\times 2^{-1} + 1\times 2^{-2}$

$(5A5.3F)_{16} = 5\times 16^2 + 10\times 16 + 5\times 1 + 3\times 16^{-1} + 15\times 16^{-2}$

(2) 数的转换

在信息交流和数据处理的过程中，经常需要将一种进制的数转换为另一种进制的等值数。在数字逻辑中，主要涉及的是十进制与非十进制数之间的转换和 2 的幂进制数之间的转换。

①十进制与非十进制数之间的转换

Ⅰ. 十进制数转换为非十进制数

根据“两个相等的有理数，它们的整数部分和小数部分必定分别相等”的原理，在把十进制数转换为非十进制数时，应将十进制数的整数部分和小数部分分别转换，再将转换结果合并起来组成与该十进制数等值的非十进制数。将十进制数转换为非十进制数所采用方法是“基数连乘除法”。

整数部分的转换采用“基数连除法”。即把十进制数的整数部分作为被除数，把转换后进制的基数作为除数，用十进制的除法运算规则进行运算，每除一次将余数（包括 0）取出，把商保留作为下一次除法运算的被除数。照此方法连除下去，一直除到商为 0 为止。第一次取出的余数是转换后数的整数部分的最低位，最后得到的余数是整数部分的最高位。

小数部分的转换用“基数连乘法”。即把十进制小数部分作为被乘数，把转换后进制的基数作为乘数，用十进制的乘法运算规则进行运算，每乘一次将乘积的整数部分（包括 0）取出，把乘积的小数部分保留作为下一次乘法运算的被乘数。照此方法连乘下去，一直到乘积的小数部分为 0 或达到与原数相近的精度为止。第一次取出的整数是转换后小数部分的最高位，最后得到的整数是小数部分的最低位。

【例 1.2】 将十进制数 23.25 转换为二进制数。

【解】 将该数的整数部分 23 用 2 连除，一直除到商等于 0 为止。将小数部分 0.25 用 2 连乘，一直乘到乘积的小数部分等于 0 为止。

转换过程如图 1.1 所示。

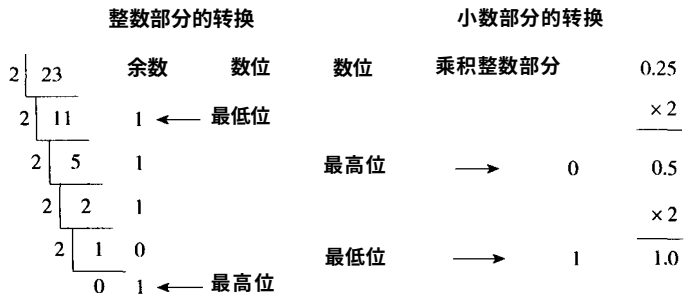


图 1.1 十进制数转换为二进制数

由以上两竖式可得转换结果

$$23.25 = (10111.01)_2$$

【例 1.3】 将十进制数 1012.83 转换为十六进制数。

【解】 将该数的整数部分 1012 用 16 连除，一直除到商等于 0 为止。将小数部分 0.83 用 16 连乘，一直乘到满足精度为止。对连除所得到的余数和连乘所得到的整数部分必须用十六进制数码去表示。

转换过程如图 1.2 所示。

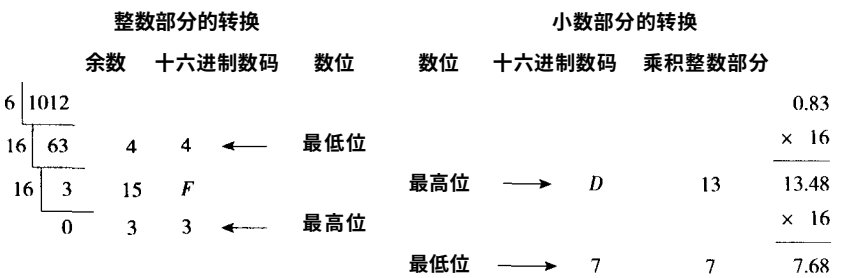


图 1.2 十进制数转换为十六进制数

由以上两竖式可得转换结果

$$1012.83 = (3F4.D7)_{16}$$

Ⅱ. 非十进制数转换为十进制数

要把非十进制数转换为十进制数，应采用“多项式替代法”，就是将非十进制数用多项式表示，然后再用十进制的运算规则，求出该多项式所表示的十进制数。

【例 1.4】 将 $(10011.011)_2$ 和 $(3AF.C)_{16}$ 转换为十进制数。

【解】 $(10011.011)_2 = 1\times2^4+1\times2^1+1\times2^0+1\times2^{-2}+1\times2^{-3} = 19.375$

$$(3AF.C)_{16} = 3\times16^2+10\times16^1+15\times16^0+12\times16^{-1}$$

$$= 3\times256+160+15+12\div16$$

$$= 943.75$$

由以上两式得转换结果

$$(10011.011)_2 = 19.375$$

$$(3AF.C)_{16} = 943.75$$

②2 的幂进制数之间的转换

计算机处理的是二进制数，但长序列的 0、1 给书写和表述带来了诸多不便。为简化起见，常用十六进制数来代表二进制数。

比如在只读存储器中，数据的字长为 8 位。设所存数据为二进制的 00000000,10101101,11110001,00010011,00100100,00111100,01011011,01110100，用十六进制来书写就为 00,*AD*,*F*1,13,24,3*C*,5*B*,74 ，其简洁程度不言而喻。

二进制、四进制、八进制、十六进制都属于 2 的幂进制。2 的幂进制数之间的转换可以采用“分组替代法”来实现。

“分组替代法”的基本原理就是 2 进制数的每个数码，都可用 i 位的二进制数表示。因此，可将二进制数作为中间平台，很方便地完成 2 的幂进制数之间的转换。

【例 1.5】 将 $(47.25)_8$ 转换为十六进制数。

【解】先将 $(47.25)_8$ 的每个数码用二位二进制数替代，得到一个二进制数。然后再将这个二进制数从小数点开始向两边把整数部分和小数部分每 4 位分为一组（位数不够时，可以在整数部分的最高有效数字位前和小数部分的最低效数字位后添 0）。再用十六进制的数码替代每个 4 位二进制数，从而得到与$(47.25)_8$ 等值的十六进制数。

转换过程如图 1.3所示。

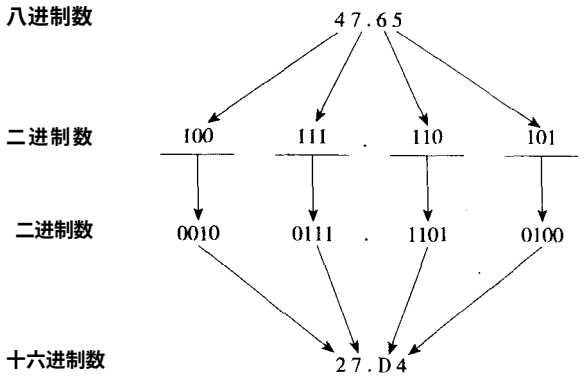


图 1.3 八进制数转换为十六进制数

由此可得转换结果

$$(47.65)_8 = (27.D4)_{16}$$

1.2.2 编码(Coding)

在许多场合，一组数码并不是一个数，而是代表某个事件、某个符号，这种替代过程就被称为编码，该数码也被称为代码。在数字逻辑电路中，常用二进制数码去替代十进制数码，所得编

码被称为 BCD(Binary Coded Decimal)码。

(1)常用 BCD 码

用二进制数对十进制的十个数码进行编码，需要 4 位二进制数中的十个码组。这十个码组被称为有效码，其余的六个码组被称为无效码。

在常用的 BCD 码中，根据编码规律，BCD 码又分为有权 BCD 码和无权 BCD 码。

①有权码

有权码就是码组中的每一位都有固定的权值，把某组代码中每一位的 1 所代表的十进制数相加，就是该代码所代表的十进制数码。

常见的有权 BCD码有 8421 码、5421 码、2421 码等，见表 1.1。

表 1.1 常见 BCD 代码

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

其中，8421 码是最重要、最常用的一种 BCD码，它选取了 4 位二进制码中的前十个码组。将 1位十进制数码转换为 4 位二进制数，就是该十进制数码所对应的 8421 码。

②无权码

无权码就是码组中的每一位没有权值，二进制码组与它所代表的十进制数码之间不存在直接的算术运算关系，仅仅是一种对应关系而已。

常见的无权 BCD码有余 3码(Excess-3 Code)、自补码、余 3格雷码(Gray Code)等，见表 1.1.

其中，余 3码的特点是每组编码比相应的 8421码多 0011，也就是多 3，故被称之为余 3码。自补码的特点是以表中数字 4、5 的代码之间为折叠线对折，对应位置上的两个数码互为反码。格雷码特点是相邻两个码组之间有且仅有 1位不相同。

③十进制数与 BCD 码之间的转换

将十进制数转换为不同的 BCD 码时，要用 4 位一组的二进制数码去替代每一个十进制数码。

【例 1.6】 将十进制数 378.45 转换为 8421 码和余 3 码。

【解】 先用 8421码的相应码组替换十进制数 378.45 的每个数码，再将每组的 8421码加上 0011，即可得到相应的余 3 码。

转换过程如图 1.4 所示。

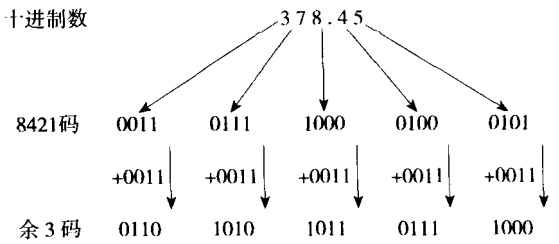


图 1.4　十进制数转换为 BCD码

由此可得转换结果

$$(378.45)_{10} = (001101111000.01000101)_{8421码}$$
$$= (011010101011.01111000)_{余3码}$$

BCD 码转换为十进制数

将不同的 BCD 码十进制数转换为普通的十进制数时，要将 4 位一组的二进制数码转换为一个十进制数码，再将所得的十进制数码按顺序写出即可。

【例 1.7】　将$(100001100011.10011011)_{余3码}$ 转换为 8421 码和十进制数。

【解】在数字逻辑中，减法运算是用加法运算来完成的。减去一个二进制数等于加上这个二进制数的补码。因此，在将余 3 码转换为 8421 码时，要把余 3 码的每个码组加上 1101，且忽略最高位进位溢出，就可得到该数的 8421 码表现形式。再将 8421 码按组转换为十进制数码即可。

转换过程如图 1.5 所示。

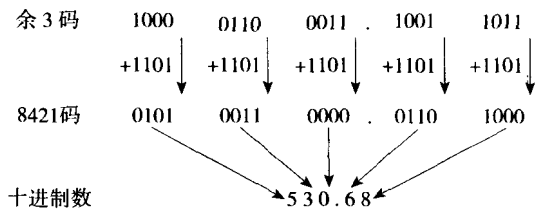


图 1.5　BCD 码转换为十进制数

由此可得转换结果

$$(100001100011.10011011)_{余3码} = (010100110000.01101000)_{8421码}$$
$$= 530.68$$

(2) 格雷码

不是任何编码在任何场合都适用，每一种编码都有适合自己的应用场合。比如，当需要把天线转动的方位角转换为数字量时，就要用特殊结构的编码盘来实现这种转换。在编码盘中，采用合适的编码方式对降低甚至消除误码的出现大有帮助。

编码盘是一组同心圆环，每个圆环又分为若干扇区，每个扇区或者是透明的，或者是不透明的。天线转动时，同心圆环随之转动。

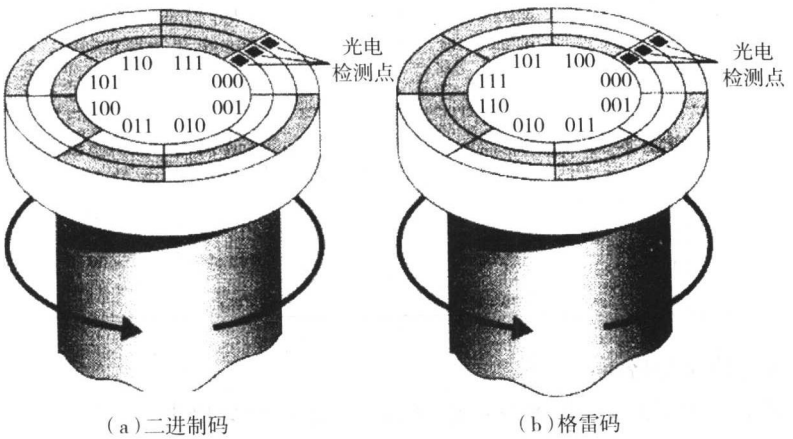
在编码盘的每个圆环上下方安装有光电检测装置。当扇区通过位置固定的光电检测装置

时，透明区将让光线通过，不透明区将阻挡光线通过。如果检测到光线用 0 表示，检测不到光线用 1 表示，则编码盘将把天线的实际方位用相应的一组二进制数码来表示。

为阐述简洁，下面以 3 个圆环、每个圆环分为 8 个扇区为例，讨论编码的实现和不同编码方式的效果。

图 1.6 所示为编码盘结构及 8421 码和格雷码的编码情况。

由图 1.6(a) 可知，该编码盘采用的是二进制编码方式。当所有扇区按逆时针方向顺序通过光电检测装置时，编码盘将依次输出编码 000、001、010、…、111。



(a)二进制码　　(b)格雷码

图 1.6　编码盘及编码方法

如果沿圆环径向排成一列的光电检测装置之间的机械位置发生小的偏移，当由某个扇区转到下一个扇区时，可能出现短暂的误码。比如，若中间检测点位置略滞后于其他两个检测点，由 001 区转到 010 区时,将出现短暂的 011。若外检测点位置略滞后于其他两个检测点，由 001 区转到 010 区时,将出现短暂的 000。

通过观察不难发现，只要相邻扇区的编码有 2 位或更多位不相同时，都可能出现误码，差异位越多，可能出现误码的数量就越多。

图 1.6(b) 是采用另一种编码方式，其编码规则是相邻扇区的编码有且仅有 1 位不相同，这种码被称为格雷码。显然，当编码盘转动时，虽然光电检测装置的机械位置可能发生小的偏移，但在相邻扇区之间切换时，输出的格雷码不会出现误码。

①常见格雷码

常见格雷码如表 1.2 所示。从表中可见格雷码不仅是位置相邻编码相差只有 1 位,而且最上行和最下行也只相差 1 位。由此可推断，格雷码的首尾编码也是相邻的，构成了循环，因而也称格雷码为循环码。

② 二进制码与格雷码之间的转换

设二进制码 B 和格雷码 G 均为 n 位，且表示为

$$B = b_{n-1}b_{n-2}\cdots b_1 b_0$$

$$G = g_{n-1}g_{n-2}\cdots g_1 g_0$$

表 1.2　格雷码

十进制数	2 位格雷码	3 位格雷码	4 位格雷码	余 3 格雷码
0	00	000	0000	0010
1	01	001	0001	0110
2	11	011	0011	0111
3	10	010	0010	0101
4		110	0110	0100
5		111	0111	1100
6		101	0101	1101
7		100	0100	1111
8			1100	1110
9			1101	1010
10			1111	
11			1110	
12			1010	
13			1011	
14			1001	
15			1000	

Ⅰ. 二进制码转换为格雷码

当已知二进制码时，格雷码的各位可根据如下关系式计算出来。

$$g_{n-1} = b_{n-1}$$

$$g_{n-2} = b_{n-1} \oplus b_{n-2}$$

……

……

$$g_1 = b_2 \oplus b_1$$

$$g_0 = b_1 \oplus b_0$$

式中，运算符号“$\oplus$”是模二加运算符号。它是把两个 1 位二进制数相加，忽略进位，取其本位和为所得结果。

模二加的运算式如下所示：

$$0 \oplus 0 = 0, \quad 0 \oplus 1 = 1, \quad 1 \oplus 0 = 1, \quad 1 \oplus 1 = 0$$

二进制码转换为格雷码的图解示例如图 1.7(a)所示。

Ⅱ. 格雷码转换为二进制码

当已知格雷码时，二进制码的各位可根据如下关系式计算出来。

$$b_{n-1} = g_{n-1};$$

$$b_{n-2} = b_{n-1} \oplus g_{n-2};$$

……;

……;

$$b_1 = b_2 \oplus g_1;$$

$$b_0 = b_1 \oplus g_0$$

格雷码转换为二进制码的图解示例如图 1.7(b)所示。

图 1.7　二进制码与格雷码之间的转换示意图

【例 1.8】　将二进制码$(10001110)_B$转换为格雷码，将格雷码$(11101100)_G$转换为二进制码。

【解】转换过程分别如图 1.8(a)和图 1.8(b)所示。

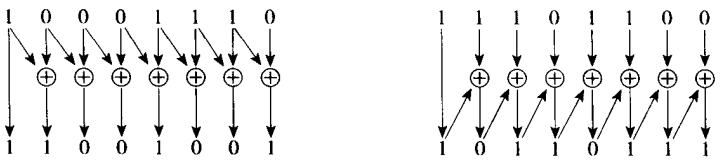


图 1.8　例 1.8转换示例

由此可得转换结果

$$(10001110)_B = (11001001)_G$$

$$(11101100)_G = (10110111)_B$$

1.3　常用逻辑运算

1.3.1　逻辑功能表示方法

在逻辑代数中，输入变量被称之为输入逻辑变量，通常用英文大写字母 A、B、C、……表示。输出结果被称之为输出逻辑变量，通常用英文大写字母 F 表示。在二值逻辑中，变量取值为 0 和 1。这里的 0 和 1 不表示数的大小，而是代表两种互相对立的状态。

为了描述逻辑电路的功能，常用逻辑函数表达式、真值表、逻辑图、波形图、状态转换图和硬件描述语言来描述输出与输入之间的逻辑关系。

在组合逻辑电路中，主要是用前三种表达方式。在时序逻辑电路中，主要是前五种表达方式。在可编程逻辑器件中主要是逻辑图和硬件描述语言。

(1) 逻辑函数表达式(Logic Expression)

逻辑函数表达式是以逻辑代数为基础的数学表达式，它是具体逻辑电路的抽象表示，便于分析、研究和公式推导。

逻辑函数表达式如下所示

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

（2）逻辑图(Logic Circuit)

逻辑图是以图形符号的形式把输入逻辑变量与输出逻辑变量之间的关系表现出来。从逻辑图我们可以直观地了解实际电路的组成概貌。在设计逻辑电路时，通常都是把逻辑函数表达式转化为逻辑图的形式，再选取相应的实际器件去实现设计。与上式相对应的逻辑图如图 1.9 所示。

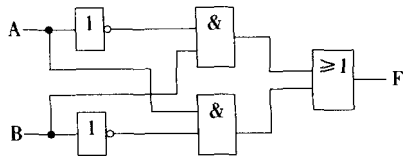


图 1.9　逻辑图

如果已知逻辑图，则根据各图形符号所实现的逻辑功能，从输入开始逐级推导，最后就可得到相应的函数表达式。

如果已知逻辑函数表达式，则可根据表达式的逻辑运算类型和先后关系，依次画出相应的逻辑符号及连线，即可得逻辑图。

（3）真值表(Truth Table)

真值表是把输入逻辑变量的所有取值组合和对应的输出结果用表格的形式详细、准确地列出，为分析逻辑电路某种输入取值组合及某种输出结果之间的关系提供了快捷方式。我们可以根据输入逻辑变量的组合，直接从真值表中查找相应的输出结果。也可根据输出结果，去查找可能的输入变量取值组合。

表 1.3　真值表

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

与上述逻辑函数表达式 $F = A\overline{B} + \overline{A}B$ 相对应的真值表如表 1.3 所示。

在列真值表时，首先是将输入变量的所有取值组合看作为一组二进制数，然后按二进制数的大小从小到大依次列出，然后根据已知逻辑函数表达式，计算出每一个输入取值所对应的输出值,并填入表中 F 栏。

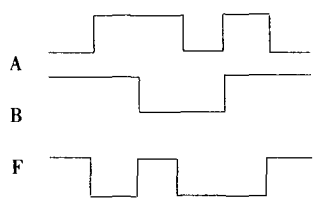


图 1.10　波形图

（4）波形图(Timing Diagram)

波形图是把数字信号的 0和 1 两种状态用确定的电平表示，以波形的形式表示输出逻辑变量与输入逻辑变量的关系。

在数字逻辑电路中，通常是用低电平表示 0 状态，用高电平表示 1 状态，这种逻辑也被称之为正逻辑。

在波形图中，输入变量取值的顺序通常不一定与真值表的顺序相同，但对每一种输入组合，输出结果与真值表的相应栏是一致的，如图 1.10 所示。

（5）状态转换图（States Transfer Diagram）

状态转换图是描述时序逻辑电路的有效工具，它表明了时序逻辑电路在外部时钟和输入信号作用下的状态转移规律。状态转换图示例如图 1.11 所示。

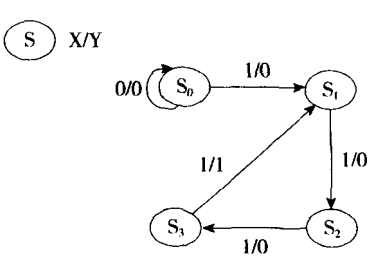


图 1.11　状态转换图

(6) 硬件描述语言(Hardware Description Language)

硬件描述语言是设计可编程逻辑器件的主要工具之一。使用专用的软件，在编辑环境下用硬件描述语言编写程序，然后经过软件编译，把程序翻译成相应的底层文件，再把底层文件固化到可编程逻辑器件中，使原来不具备逻辑功能的器件可以完成特定的逻辑功能。硬件描述语言程序示例如下所示。

```
module repeat_loop(clock);
  input clock;
  initial begin
  repeat(5)
  @(posedge clock);
  $ stop;
  end
endmodule
```

1.3.2 基本逻辑

在实际应用中，遇到的逻辑问题千变万化。有的问题比较简单，有的又很复杂。但复杂的逻辑问题可分解为若干简单逻辑问题的组合。

在逻辑代数中,最基本的逻辑关系有“与”、“或”、“非”三种。

(1) 与(AND)逻辑

与逻辑所表示的逻辑关系是：只有在决定某一事件的条件全部具备时，这一事件才会发生。

与逻辑的函数表达式为

$$F = A \cdot B = AB$$

式中“·”表示“与”运算,可略去不写。

与逻辑的逻辑符号如图 1.12 所示，真值表如表 1.4 所示，波形图如图 1.13 所示。

表 1.4 真值表

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

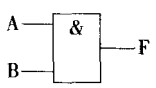


图 1.12 与逻辑符号

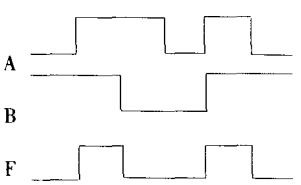


图 1.13 与逻辑波形图

与逻辑也称逻辑乘，它可扩展成任意多个变量相与。如

$$F = ABC\cdots$$

与逻辑的运算规则是：只要输入变量有一个为 0,则输出为 0；只有全部输入变量都为 1,输出才为 1。简述为:见 0 出 0,全 1 出 1。

(2) 或(OR)逻辑

或逻辑所表示的逻辑关系是：只要决定某一事件的某一条件具备，这一事件就会发生。

或逻辑的函数表达式为

$$F = A + B$$

式中“+”表示“或”运算。

或逻辑的逻辑符号如图 1.14 所示，真值表如表 1.5 所示，波形图如图 1.15 所示。

表 1.5 真值表

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

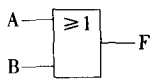


图 1.14 或逻辑符号

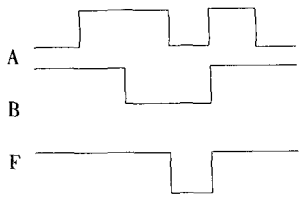


图 1.15 或逻辑波形图

或逻辑也称逻辑加，它可扩展成任意多个变量相或。如

$$F = A + B + C + \cdots$$

或逻辑的运算规则是：只要输入变量有一个为 1，则输出为 1 只有全部输入变量都为 0，输出才为 0。简述为见 1 出 1，全 0 出 0。

(3) 非(NOT)逻辑

非逻辑所表示的逻辑关系是：某一逻辑函数的运算结果是输入逻辑变量的相反状态。

非逻辑的函数表达式为

$$F = \overline{A}$$

式中，输入变量 A 上方的“—”表示取非运算。

非逻辑的逻辑符号如图 1.16 所示，真值表如表 1.6 所示，波形图如图 1.17 所示。

表 1.6 真值表

A	F
0	1
1	0

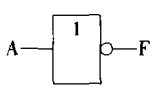


图 1.16 非逻辑符号

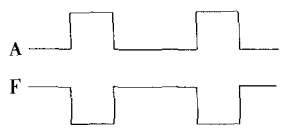


图 1.17 非逻辑波形图

1.3.3 常用复合逻辑

从原理上来说，任意复杂的逻辑功能都能用三种基本逻辑来实现。但相应的逻辑电路就显得特别复杂，增加了电路连线、降低了电路的可靠性。在实际应用中，往往是将三种基本逻辑进行适当的组合，构成功能稍微复杂的逻辑模块。将这些逻辑模块作为基本器件去设计逻辑电路，所得到的逻辑图就比完全用基本逻辑得到的逻辑图要简洁得多。

常用的复合逻辑有“与非”、“或非”、“与或非”、“异或”和“同或”。

(1) 与非 (NAND)逻辑

与非逻辑是在与逻辑的输出基础上增加一个非逻辑所组成的。

与非逻辑的表达式为

$$F = AB$$

与非逻辑的等效构成及逻辑符号如图 1.18所示，真值表如表 1.7所示，波形图如图 1.19所示。

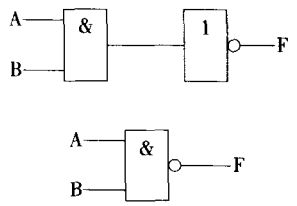


图 1.18 与非逻辑符号

表 1.7 真值表

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

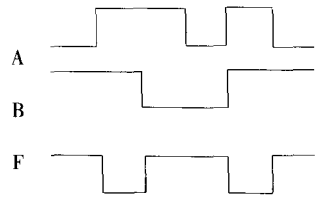


图 1.19 与非逻辑波形图

与非逻辑的运算规则是：只要输入变量有一个为 0,则输出为 1 只有全部输入变量取值都为 1 时,输出才为0。简述为:见 0 出 1,全 1 出 0。

(2)或非(NOR)逻辑

或非逻辑是在或逻辑的输出基础上增加一个非逻辑所组成的。

或非逻辑的表达式为

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

或非逻辑的等效构成及逻辑符号如图 1.20 所示，真值表如表 1.8所示，波形图如图 1.21所示。

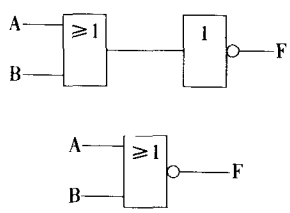


图 1.20 与非逻辑符号

表 1.8 真值表

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

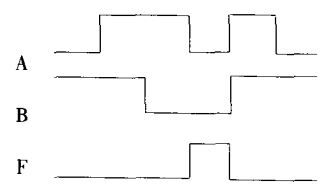


图 1.21 与非逻辑波形图

或非逻辑的运算规则是：只要输入变量有一个为 1,则输出为 0；只有全部输入变量取值都为 0 时,输出才为 1。简述为:见 1 出 0,全 0 出 1。

(3) 与或非(AND - OR - NOT)逻辑

与或非逻辑是在多个与逻辑的输出基础上增加一个或逻辑和一个非逻辑所组成的。

与或非逻辑的表达式为

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

与或非逻辑的等效构成及逻辑符号如图 1.22所示。

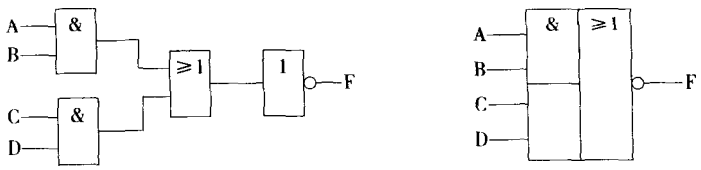


图 1.22　与或非逻辑构成及逻辑符号

与或非逻辑的运算规则是：在多个与门中，只要有一个与门的输入全为 1，则与或非门的输出就为 0；只有每个与门中至少有一个输入变量 0 时，与或非门的输出才为 1。

(4) 异或(Exclusive-OR) 逻辑和同或(Exclusive-NOR)逻辑

异或逻辑的表达式为

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

异或逻辑的逻辑图如图 1.23 所示，真值表如表 1.9 所示，波形图如图 1.24所示,逻辑符号如图 1.25 所示。

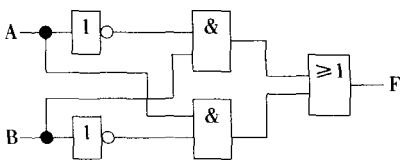


图 1.23　逻辑图

表 1.9　真值表

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

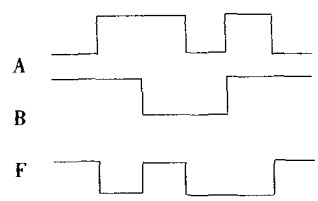


图 1.24　波形图

异或逻辑的运算规则是：只要两个输入变量取值相同，则输出为 0；当两个输入变量取值不相同，则输出为 1。简述为 相同出 0,相异出 1。

同或逻辑是异或逻辑的非，表达式为

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

逻辑符号如图 1.26 所示，真值表如表 1.9 所示。

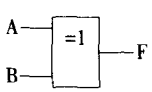


图 1.25　异或逻辑符号

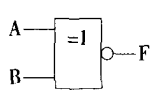


图 1.26　同或逻辑符号

同或逻辑的运算规则是：只要两个输入变量取值相同，则输出为 1；当两个输入变量取值不相同，则输出为 0。简述为 相同出 1,相异出 0.

1.4　基本定律和基本规则

根据基本逻辑和复合逻辑的运算规则，可导出逻辑代数运算中的一些基本定律和基本规则。这些定律和规则在实际逻辑电路的分析和设计中很有用，是必须掌握的内容。

1.4.1 基本定律

(1) 变量与常量的定律

变量与常量定律如表 1.10 所示。

表 1.10 变量与常量定律

名称	表达式	
0—1 律	$A \cdot 0 = 0$	$A + 1 = 1$
自等律	$A \cdot 1 = A$	$A + 0 = A$
重叠律	$A \cdot A = A$	$A + A = A$
互补律	$A \cdot \overline{A} = 0$	$A + \overline{A} = 1$

(2) 类似普通代数的定律

常用定律如表 1.11 所示。

表1.11 常用定律

名称	表达式	
交换律	$A \cdot B = B \cdot A$	$A + B = B + A$
结合律	$A \cdot (B \cdot C) = (A \cdot B) \cdot C$	$A + (B + C) = (A + B) + C$
分配律	$A \cdot (B + C) = A \cdot B + A \cdot C$	$A + B \cdot C = (A + B) \cdot (A + C)$

【例 1.9】 分配律 $A + B \cdot C = (A + B) \cdot (A + C)$ 的证明。

【证】
$$\begin{aligned}(A + B) \cdot (A + C) &= AA + AC + BA + BC \\ &= A + AB + AC + BC \\ &= A \cdot 1 + AB + AC + BC \\ &= A(1 + B + C) + BC \\ &= A + BC\end{aligned}$$

$$A + B \cdot C = (A + B) \cdot (A + C)$$

(3) 逻辑代数中的特殊定律

特殊定律如表 1.12 所示。

表 1.12 特殊定律

名称	表达式	
反演律	$\overline{A \cdot B} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A} \cdot \overline{B}$
合并律	$AB + A\overline{B} = A$	$(A + B)(A + \overline{B}) = A$
吸收律 Ⅰ	$A + AB = A$	$A(A + B) = A$
吸收律 Ⅱ	$A + \overline{A}B = A + B$	$A(\overline{A} + B) = AB$
吸收律 Ⅲ	$AB + \overline{A}C + BC = AB + \overline{A}C$	$(A + B)(\overline{A} + C)(B + C) = (A + B)(\overline{A} + C)$

【例 1.10】 反演律 (狄·摩根定律) $\overline{A \cdot B} = \overline{A} + \overline{B}$ 的证明。

【证】 证明过程如表 1.13 所示。

表 1.13　反演律的证明

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

该证明方法叫做真值表证明法。它是将定律中的输入逻辑变量的所有取值组合一一列出，然后计算出定律两边各自的相应结果。如果定律两边对相同的输入取值组合都有相同的输出结果，那么定律就成立，否则就不成立。

【例 1.11】 吸收律 Ⅱ　$A+\overline{A}B=A+B$ 的证明。

【证】
$$A+AB=(A+\overline{A})(A+B)$$
$$=A+B$$

吸收律 Ⅱ 表明：在一个与或表达式中，如果某一乘积项的部分因子恰好是另一独立乘积项，则该部分因子是多余的。

【例 1.12】 吸收律 Ⅲ(包含律)$AB+\overline{A}C+BC=AB+\overline{A}C$ 的证明。

【证】 证明过程如表 1.14 所示。

表 1.14　包含律的证明

A	B	C	AB	$\overline{A}C$	BC	$AB+\overline{A}C+BC$	$AB+\overline{A}C$
0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	1
0	1	0	0	0	0	0	0
0	1	1	0	1	1	1	1
1	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0
1	1	0	1	0	0	1	1
1	1	1	1	0	1	1	1

从表 1.14 可见，对于相同的输入 $AB+\overline{A}C+BC$ 的值与 $AB+\overline{A}C$ 的值是相同的。且对所有的输入组合均成立，故得证。

包含律表明，在一个与或表达式中，如果两个乘积项中的部分因子互补，而这两个乘积项中的其余因子组成了第三个乘积项或是第三个乘积项的因子，则第三个乘积项是多余的。

异或逻辑与同或逻辑不仅是互为反函数，而且还是互为对偶函数。在可编程器件中，异或逻辑常用来控制输出信号的极性。

异或逻辑、同或逻辑与常量、变量之间的公式如表 1.15 所示。

表 1.15　常用异或逻辑和同或逻辑运算公式

异或逻辑运算式	同或逻辑运算式
$A\oplus 0=A, A\oplus 1=\overline{A}$	$A\odot 0=\overline{A}, A\odot 1=A$
$A\oplus A=0, A\oplus\overline{A}=1$	$A\odot A=1, A\odot\overline{A}=0$
$A\oplus\overline{B}=\overline{A\oplus B}=A\oplus B\oplus 1$	$A\odot\overline{B}=A\oplus B=A\odot B\odot 1$
$A\oplus B=B\oplus A$	$A\odot B=B\odot A$
$A\oplus(B\oplus C)=(A\oplus B)\oplus C$	$A\odot(B\odot C)=(A\odot B)\odot C$
$A(B\oplus C)=AB\oplus AC$	$A(B\odot C)=AB\odot AC$

1.4.2 基本规则

(1)代入规则

代入规则：在任何一个包含变量 A 的逻辑表达式中，若以另一个逻辑表达式代替所有的变量 A，该表达式仍成立。

【例 1.13】用代入规则证明反演定律也适用于多变量的情况。

【证】已知二变量的反演定律为

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

以 BC 代替上式的两边的 B，得

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

因为

$$\overline{BC} = \overline{B} + \overline{C}$$

所以

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

(2)反演规则

对于任何一个逻辑表达式，如果将式中所有的“与”运算符换成“或”运算符，“或”运算符换成“与”运算符，0 换为 1，1 换为 0，原变量换为反变量，反变量换为原变量，则所得到新的表达式就是原表达式的取反。

反演规则是反演律的推广，运用它可以方便地求出一个逻辑函数的反函数。

【例 1.14】 已知 $F = \overline{A + B}(\overline{C} + \overline{D})$，用反演规则求 $\overline{F}$。

【解 1】 对表达式中的单个变量用反演律得

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

【解 2】把表达式中的 $A + B$ 看作单个变量，用反演律得

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

运用反演规则时，要注意两点：

①不能破坏原式的运算顺序；

②在多层非号中，只能对其中的某一层非号用反演规则。

(3)对偶规则

对于任何一个逻辑表达式，如果将式中所有的“与”运算符换成“或”运算符，“或”运算符换成‘与’运算符，0 换为 1，1 换为 0，变量保持不变，则所得到的新表达式就是原表达式的对偶表达式。

在表 1.9、表 1.10 和表 1.11 中，表左栏中的式子与右栏中的式子就是互为对偶关系。

【例 1.15】 已知 $F = \overline{A + B}(\overline{C} + \overline{D})$，求它的对偶表达式 F^*。

【解】

$$\begin{aligned} F^* &= [\overline{A + B}(\overline{C} + \overline{D})]^* \\ &= \overline{AB} + \overline{C}\,\overline{D} \\ &= \overline{A} + \overline{B} + \overline{C}\,\overline{D} \end{aligned}$$