第 3 章　组合逻辑的分析与设计

【学习要求】

本章包括逻辑代数基础、组合逻辑电路的分析与设计和 VHDL 硬件描述语言三方面的内容。逻辑代数是数字逻辑电路的理论基础，学习时应重点掌握逻辑代数的基本公式、定理和规则，逻辑函数的两种基本形式以及逻辑函数的化简方法。应掌握组合逻辑电路分析和设计的基本步骤，能熟练分析所给组合逻辑电路并能按照要求设计所需组合逻辑电路。EDA 是现代逻辑设计的发展方向，硬件描述语言是其重要基础，学习时应重点掌握 VHDL 描述的基本结构、VHDL 语言的主要语法要素、VHDL 主要语句的语法结构，能够读懂 VHDL 程序，会编写简单的 VHDL 程序。

3.1　要 点 指 导

1. 逻辑代数基础

1）逻辑变量及基本逻辑运算

数字电路进行信息处理的理论基础是逻辑代数。英国数学家乔治·布尔在其著作《逻辑的数学分析》及《思维规律的研究》中首先提出了这种代数的基本概念和性质，因此逻辑代数也称布尔代数。

逻辑代数是一个二值代数系统，任何逻辑变量的取值都只有“0”和“1”两种可能性。逻辑代数中定义了“与”、“或”、“非”3 种基本运算，这 3 种基本运算满足表 3-1 所列的运算规律。

表 3-1　逻辑代数的基本运算规律与常用公式

运算规律	“与或”式形式	“或与”式形式
交换律	$A+B=B+A$	$A\cdot B=B\cdot A$
结合律	$(A+B)+C=A+(B+C)$	$(A\cdot B)\cdot C=A\cdot(B\cdot C)$
分配律	$A+B\cdot C=(A+B)\cdot(A+C)$	$A\cdot(B+C)=A\cdot B+A\cdot C$
0-1 律	$A+0=A, A+1=1$	$A\cdot 0=0, A\cdot 1=A$
互补律	$A+\overline{A}=1$	$A\cdot\overline{A}=1$
吸收律	$A+AB=A, A+\overline{A}B=A+B$	$A(A+B)=A, A(\overline{A}+B)=AB$
重叠律	$A+A=A$	$A\cdot A=A$
对合律	$\overline{\overline{A}}=A$	
反演律	$\overline{A+B}=\overline{A}\,\overline{B}$	$\overline{AB}=\overline{A}+\overline{B}$
包含律	$AB+\overline{A}C+BC=AB+\overline{A}C$	$(A+B)(\overline{A}+C)(B+C)=(A+B)(\overline{A}+C)$
常用公式	$AB+A\overline{B}=A$	$(A+B)(A+\overline{B})=A$
异或运算	$A\oplus B=A\overline{B}+\overline{A}B, A\oplus 1=\overline{A}, A\oplus 0=A, \overline{A\oplus B}=A\odot B$	
同或运算	$A\odot B=AB+\overline{A}\,\overline{B}, A\odot 1=A, A\odot 0=\overline{A}, \overline{A\odot B}=A\oplus B$	

2）逻辑代数的主要定理

（1）德·摩根定理。

德·摩根定理是反演律的一般形式，有以下两种形式：

① $\overline{x_1+x_2+\cdots+x_n}=\overline{x_1}\cdot\overline{x_2}\cdot\cdots\cdot\overline{x_n}$

② $\overline{x_1\cdot x_2\cdot\cdots\cdot x_n}=\overline{x_1}+\overline{x_2}+\cdots+\overline{x_n}$

（2）香农定理。

香农定理是德·摩根定理的推广，可以用于直接求任何复杂函数的反函数。公式的形式如下：

$$\overline{f(x_1,x_2,\cdots,x_n,0,1,+,\cdot)}=f(\overline{x_1},\overline{x_2},\cdots,\overline{x_n},1,0,\cdot,+)$$

（3）展开定理。

展开定理提供了一种将一个逻辑函数展开成“与或”式和“或与”式的方法，有以下两种形式：

① $f(x_1,x_2,\cdots,x_i,\cdots,x_n)=x_if(x_1,x_2,\cdots,1,\cdots,x_n)+\overline{x_i}f(x_1,x_2,\cdots,0,\cdots,x_n)$

② $f(x_1,x_2,\cdots,x_i,\cdots,x_n)=[x_i+f(x_1,x_2,\cdots,0,\cdots,x_n)]\cdot[\overline{x_i}+f(x_1,x_2,\cdots,1,\cdots,x_n)]$

展开定理有两组 4 个推理：

① $x_if(x_1,x_2,\cdots,x_i,\cdots,x_n)=x_if(x_1,x_2,\cdots,1,\cdots,x_n)$

② $x_i+f(x_1,x_2,\cdots,x_i,\cdots,x_n)=x_i+f(x_1,x_2,\cdots,0,\cdots,x_n)$

③ $\overline{x_i}f(x_1,x_2,\cdots,x_i,\cdots,x_n)=\overline{x_i}f(x_1,x_2,\cdots,0,\cdots,x_n)$

④ $\overline{x_i}+f(x_1,x_2,\cdots,x_i,\cdots,x_n)=\overline{x_i}+f(x_1,x_2,\cdots,1,\cdots,x_n)$

3）逻辑代数的重要规则

（1）代入规则。

对于逻辑等式中的任何一个变量 x，若以函数 F 代替等式两边的变量 x，则逻辑等式仍然成立。

（2）对偶规则。

对偶式：把逻辑函数 F 中的所有“+”变为“·”，“·”变为“+”，“1”变为“0”，“0”变为“1”，得到的式子称为 F 的对偶式，记为 F_d。

对偶规则：对于两个逻辑函数 F 和 G，若有 F=G，则有 $F_d=G_d$。

（3）反演规则。

把逻辑函数 F 中的所有“+”变为“·”，“·”变为“+”，“1”变为“0”，“0”变为“1”，原变量变为反变量，反变量变为原变量，则可得函数 F 的反函数$\overline{F}$。可见，反演规则实际上就是香农定理。

4）逻辑函数及其表达式

（1）逻辑函数的表示方法。

逻辑函数的常用表示方法有逻辑表达式、真值表和卡诺图 3 种。

逻辑表达式是由逻辑变量和“与”、“或”、“非”3 种基本运算构成的式子，书写时在不影响运算顺序的情况下，可以省略某些括号和“与”运算符号。

真值表和卡诺图是数字电路的分析与设计中常用的重要工具，要熟练掌握。真值表是反映输出和输入取值逻辑关系的表格，由输入和输出两栏组成，左边是输入，右边是输出。

对于有 n 个输入变量的逻辑函数，其真值表应有 2^n 行，对应输入变量的 2^n 种取值组合。为防止遗漏，输入变量的取值组合一般按二进制顺序来列。

卡诺图是由表示输入变量的所有可能取值组合的小方格构成的平面图形，在逻辑函数的化简中非常有用。具体规则将结合卡诺图化简方法进行介绍。

(2) 逻辑函数的两种基本形式。

标准积：包含逻辑函数中的所有变量(每个变量以原变量和反变量的形式出现且仅出现一次)的积项(与项)，也常称为最小项。

标准积之和式：每个与项都是最小项的“与或”式称为标准积之和式，或最小项之和式。

标准和：包含逻辑函数中的所有变量(每个变量以原变量和反变量的形式出现且仅出现一次)的和项(或项)，也常称为最大项。

标准和之积式：每个或项都是最大项的“或与”式称为标准和之积式，或最大项之积式。

最小项的表示方法：通常用 m_i 表示最小项，将最小项中的原变量记为“1”，反变量记为“0”，变量按照一定的顺序排列，得到的二进制数对应的十进制数值即为下标 i。

最大项的表示方法：通常用 M_i 表示最大项，将最大项中的原变量记为“0”，反变量记为“1”，变量按照一定的顺序排列，得到的二进制数对应的十进制数值即为下标 i。

最小项的性质：

① 对于任意一个最小项，有且仅有一组变量取值组合可使其值为 1，该取值即为序号 i 的值。

② 任意两个不同的最小项之积必为 0，即 $m_i \cdot m_j = 0 (i \neq j)$。

③ n 个变量的全部最小项之和必为 1，即 $\sum_{i=0}^{2^n-1} m_i = 1$。

④ n 个变量的任何一个最小项都有 n 个相邻最小项(简称相邻项)。所谓相邻最小项，是指仅有一个变量不同的最小项(一个为原变量，另一个为反变量)。

最大项的性质：

① 对于任意一个最大项，有且仅有一组变量取值组合可使其值为 0，该取值即为序号 i 的值。

② 任意两个不同的最大项之和必为 1，即 $M_i + M_j = 1 (i \neq j)$。

③ n 个变量的全部最大项之积必为 0，即 $\prod_{i=0}^{2^n-1} M_i = 0$。

④ n 个变量的任何一个最大项都有 n 个相邻最大项。

最小项和最大项的关系：

① $m_i = \overline{M_i}$，$M_i = \overline{m_i}$，$m_i + M_i = 1$，$m_i \cdot M_i = 0$。

② 同一函数的最大项与最小项是互斥的，即如果函数的标准积之和式中有最小项 m_i，则最大项 M_i 就不可能出现在同一函数的标准和之积式中。也就是说，一个布尔函数的最小项的集合与它的最大项的集合之间互为补集。

将逻辑函数化成标准积之和式的方法有以下两种。

① 代数演算法：通过反复使用公式 $x + \bar{x} = 1$ 和 $x(y+z) = xy + xz$ 进行演算求得标准和之积的方法。

② 列表法：列出函数的真值表，使函数取值为 1 的那些最小项就构成了函数的标准积

之和式。

将逻辑函数化成标准和之积式的方法有以下两种。

① 代数演算法：通过反复使用公式 $x \cdot \bar{x}=0$ 和 $x+yz=(x+y)(x+z)$ 进行演算求得标准和之积的方法。

② 列表法：列出函数的真值表，使函数取值为 0 的那些最大项就构成了函数的标准和之积式。

2. 逻辑函数的化简

1）逻辑函数的最简标准

最简“与或”式：

（1）表达式中的与项数最少；

（2）每个与项中的变量最少。

最简“或与”式：

（1）表达式中的或项数最少；

（2）每个或项中的变量最少。

2）代数化简法

代数化简法运用逻辑代数的基本公式、定理和规则对逻辑函数进行化简，这种方法建立在对公式的熟练记忆和灵活运用上，常用的方法有并项法、吸收法、消去法和配项法等。该方法的最大优点是灵活方便，不受变量数量的约束；缺点是没有一定的规律和步骤，技巧性较强，对于复杂的逻辑函数究竟化简到什么程度是最简的，还不能进一步化简，有时不太容易判断。

3）卡诺图化简法

卡诺图化简法使用平面图形，根据最小项的相邻关系进行化简，具有直观、不用记忆公式、容易判断是否是最简结果等优点，缺点是变量个数不能太多，一般不超过 6 个。

因卡诺图中的变量是按照格雷码顺序排列的，所以最小项在卡诺图中的相邻关系分为几何相邻、相对相邻和相重相邻 3 种情况，如图 3-1 所示。

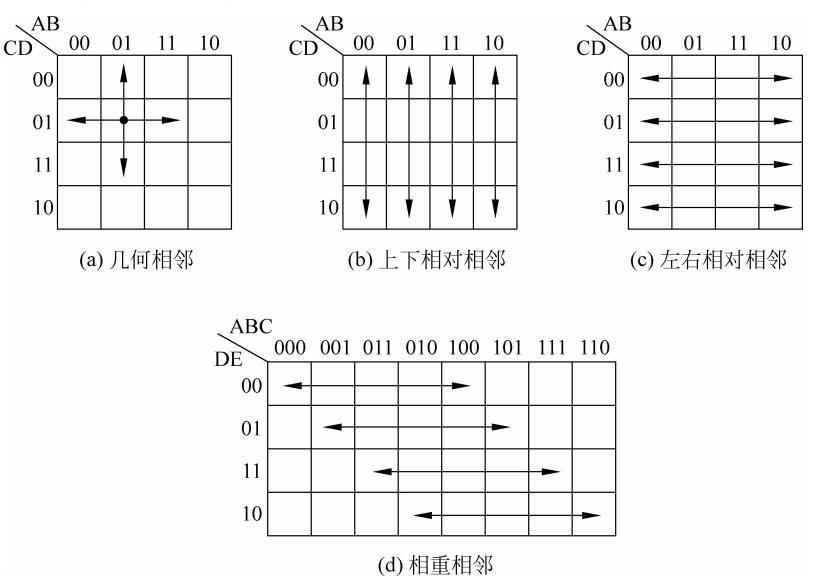

图 3-1　最小项相邻的 3 种情况

图 3-1(a)为几何相邻，是指一个最小项与其上、下、左、右的最小项在几何位置上是相邻的。图 3-1(b)为上下相对相邻，是指每列最下面的一个最小项与其相对的最上面的最小项是相对相邻的。图 3-1(c)为左右相对相邻，是指每行最左边的一个最小项与其相对的最右边的最小项是相对相邻的。图 3-1(d)为相重相邻，是指将卡诺图沿纵向中间线剪开叠放在一起后，相互重叠的最小项是相重相邻的。

用卡诺图进行化简的原则是：在卡诺图上形成最小项的最小覆盖。所谓最小覆盖，包含最小和覆盖两层含义。最小是指选择最少数量的尽可能大的卡诺圈；覆盖是指所选卡诺圈要能把所有的函数中出现的最小项都包含进去，不能有遗漏。要形成最小覆盖，首先要选择所有必要质蕴含项，如果必要质蕴含项不能覆盖所有的最小项，再选择非必要质蕴含项。

这里应注意，逻辑函数的最简形式不一定是唯一的，所以非必要质蕴含项的选择有可能有多种方案。

卡诺图化简法虽然简单、直观，但不适宜机器实现，也不适宜变量个数较多的情况。

这里补充两种用卡诺图将逻辑函数化成最简“或与”式的方法：一种方法是将逻辑函数 F 的反函数$\overline{F}$化成最简“与或”式(对应卡诺图中标 1 之外的最小项)，再应用反演规则对$\overline{F}$一次求反即可；另一种方法是当所给逻辑函数 F 是“或与”式时，可以先求 F 的对偶式 F_d，然后用卡诺图将 F_d化成最简“与或”式，再对 F_d取对偶$(F_d)_d$即可。具体例子见 3.2 节例题精讲部分。

4) 列表化简法

列表化简法的基本思想与卡诺图化简法相同，都是从最小项出发，找出相邻项进行合并。所不同的是化简过程是借助约定的表格形式，按照一定规则进行的。该方法的优点是化简步骤规范，不受变量个数的约束，适合于多变量函数化简和计算机辅助逻辑化简，现代 EDA 软件中的逻辑功能化简模块就是用该方法实现的。

5) 逻辑函数化简中的两个实际问题

(1) 包含无关最小项的逻辑函数的化简。

无关最小项是指对函数取值没有影响的最小项，用“d”或“×”表示。包含无关最小项的逻辑函数用卡诺图法化简的原则是：把对化简有利的最小项当“1”，用卡诺圈包含进去，把对化简不利(需要更多的卡诺圈才能将其覆盖)的最小项当“0”，不需要覆盖。

(2) 多输出逻辑函数的化简。

多输出逻辑函数是指由一组相同的输入变量产生的多个输出函数，对应的逻辑电路为多输出逻辑电路。为使逻辑电路整体上达到最简，应考虑多个输出函数之间尽可能使用公共项，而不是以使每个函数单独最简为标准。

多输出逻辑函数最简的标准是：

- 所有逻辑表达式中包含的不同“与”项总数最少。
- 在满足上述条件的前提下，各不同“与”项中所含的变量总数最少。

3. 组合逻辑电路的分析与设计

组合逻辑电路在任何时刻产生的稳定输出值仅与该时刻的输入值有关，而与电路过去的输入值无关。组合逻辑电路仅由门电路构成，电路中不存在任何输出到输入的反馈回路，没有记忆能力。

对组合逻辑电路的研究包括分析和设计两个方面。分析是对给定的逻辑电路图进行研

究，找出其实现的逻辑功能的过程；设计是用逻辑电路图实现给定逻辑功能的过程，也称为逻辑综合。分析和设计是两个相反的过程。

1）组合逻辑电路的分析与设计的一般步骤

组合逻辑电路的分析过程如图 3-2 所示，设计过程如图 3-3 所示。

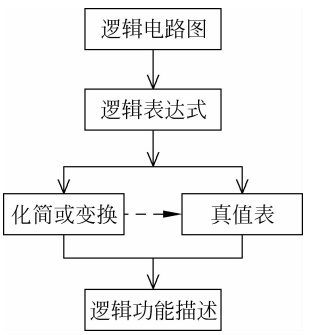

图 3-2　组合逻辑电路的分析过程

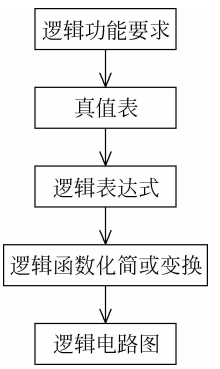

图 3-3　组合逻辑电路的设计过程

2）组合逻辑电路设计中应考虑的问题

（1）逻辑函数形式的变换。

根据逻辑电路图得到的输出函数表达式一般是最简“与或”式，为了满足应用特定的门电路实现等要求，需要将逻辑函数的最简“与或”式进行相应的变换。

① 逻辑函数的“与非”门实现。

需将最简“与或”式变为“与非-与非”式。有两种方法：一种是对 F 两次求反，一次展开；另一种是对$\overline{F}$三次求反，一次展开。

方法一得到的是两级电路，方法二得到的是三级电路，速度稍慢。当原函数比较简单时，方法一可节省门电路；当反函数比较简单时，方法二可节省门电路。

② 逻辑函数的“或非”门实现。

需将最简“与或”式变为“或非-或非”式。有两种方法：一种是对 F 两次求对偶，即先求 F 的对偶式 F_d，并将其化为最简“与非-与非”式，然后再求 F_d的对偶式$(F_d)_d$；另一种是对 F 的最简“或与”式两次取反，一次展开。

③ 逻辑函数的“与或非”门实现。

需将最简“与或”式变换为“与或非”式。有两种方法：一种是对 F 两次求反；另一种是对$\overline{F}$一次求反。

（2）多输出组合逻辑电路的设计。

多输出组合逻辑电路设计的基本步骤与单输出组合逻辑电路的设计基本相同，只是在进行逻辑化简时，应按照多输出函数化简的标准进行。

（3）包含无关项的组合逻辑电路的设计。

包含无关项的组合逻辑电路的设计，应根据功能要求和输入变量的类型，确定哪些最小项是函数的无关项，输出函数化简时应按照包含无关项的逻辑函数的化简原则进行。

（4）考虑级数的组合逻辑电路的设计。

电路的级数直接影响电路的速度，要提高电路的工作速度，就要压缩电路的级数。电路

的级数反映在输出函数表达式中，就是与、或、非运算符号的层数。因此，压缩级数可以通过对输出函数求反或展开来获得。先对 F 求反，并求出$\overline{F}$的最简“与-或”式，再对$\overline{F}$求反来压缩电路级数的方法，称为求反压缩法。

4. 组合逻辑电路的设计举例及其 VHDL 描述

本部分讲述了用硬件描述语言 VHDL 描述组合逻辑电路的方法。为方便教学和学生理解，根据我们多年的教学经验，将 VHDL 语言集中介绍，既枯燥又不便于学生理解掌握，因此教材对这部分内容的安排进行了改革，在具体使用 VHDL 描述电路之前，仅将 VHDL 最基本的语法展示出来，把其他相关的语法都放到具体的实例中介绍，书中几乎所有的实例都配有 VHDL 描述，结合实例再讲述程序描述中出现的语法部分。这样既可以把 VHDL 贯穿于课程的始终，又可以掌握 VHDL 的大部分语法结构，取得了较好的教学效果。但这样安排内容相对分散，对学生期末复习或是平时使用时的语法查询不太方便。因此，本书单独列出一个章节集中介绍 VHDL 的语法，并给出了常用逻辑电路的 VHDL 描述。这部分内容放在本书的第二部分，这里不再对 VHDL 进行复习。

本节讲述的半加器和全加器、BCD 码编码器和七段显示译码器以及代码转换电路都是常用的组合逻辑电路，应掌握每种电路的设计方法以及 VHDL 描述。

5. 组合逻辑电路中的竞争与险象

1）基本概念

竞争与险象是由信号传输中的时间延迟造成的。输入信号经过不同的路径到达输出端的时间有先有后，这种现象称为竞争。由于竞争的存在，当输入信号变化时就有可能引起输出信号出现非预期的错误输出，这种现象称为险象。并不是所有的竞争都会产生错误输出，把不会产生错误输出的竞争称为非临界竞争，会导致错误输出的竞争称为临界竞争。因为一旦延迟时间结束，即可恢复正常的逻辑关系，而延迟时间也是非常短暂的，所以临界竞争产生的错误输出(险象)的表现形式为短暂的尖峰脉冲。

2）险象的分类

按输入变化前后输出是否应该相等，可将险象分为静态险象和动态险象两类。按照错误输出的尖峰脉冲的极性，可将险象分为“0”型险象和“1”型险象两类。根据两种分类方法的组合，可以把险象分为静态“0”型险象、静态“1”型险象、动态“0”型险象、动态“1”型险象 4 类。

3）险象的判断

(1) 险象存在的必要条件。

① 某变量 X 同时以原变量 X 和反变量 $\overline{X}$ 两种形式出现在函数中，并且在一定的条件下可以将函数表达式化简成 $X+\overline{X}$ 或 $X\cdot\overline{X}$形式。

② 因为 $X+\overline{X}=1$，$X\cdot\overline{X}=0$，所以若函数表达式可以化简成 $X+\overline{X}$ 的形式，则可能存在的险象为“0”型险象；若函数表达式可以化简成 $X\cdot\overline{X}$的形式，则可能存在的险象为“1”型险象。

(2) 判断方法。

① 代数法：首先检查函数表达式中是否存在具备竞争条件的变量 X。若有，则将其他变量的各种取值组合依次代入到函数表达式中，使表达式中仅含变量 X。最后，再看函数表达式是否能化成 $X+\overline{X}$或 $X\cdot\overline{X}$的形式，若能，则对应的逻辑电路存在产生险象的可能性。

② 卡诺图法：首先画出“与或”式函数的卡诺图，并画出和函数表达式中各“与”项对应的卡诺圈。然后观察卡诺圈，若发现某两个卡诺圈存在“相切”（两个卡诺圈之间存在不被同一个卡诺圈包含的相邻最小项）关系，则该电路可能产生险象，否则，没有险象。

4）险象的消除

（1）增加冗余项法。

通过在函数表达式中“加”上多余的“与”项或“乘”上多余的“或”项，使原函数不再可能在某种条件下化成 $X+\overline{X}$ 或 $X\cdot\overline{X}$ 的形式，从而将可能产生的险象消除。此处，多余的“与”项和多余的“或”项就是冗余项。冗余项的选择可采用代数法或卡诺图法。

（2）滤波法（增加惯性延时环节法）。

在输出端加低通滤波器（RC 电路，也称惯性延时环节），削掉错误输出的尖峰脉冲。

（3）选通法。

在电路输出端增加选通控制脉冲，避免尖峰脉冲的输出。

3.2　例题精讲

例 3-1　把逻辑函数 $F(A,B,C)=A\overline{B}+B\overline{C}+\overline{B}C+\overline{A}B$ 化成最简“与或”式和最简“或与”式。

解：最简“与或”式：

方法一：

$$
\begin{aligned}
F(A,B,C)&=A\overline{B}+B\overline{C}+\overline{B}C+\overline{A}B\\
&=A\overline{B}+B\overline{C}+(A+\overline{A})\overline{B}C+\overline{A}B(C+\overline{C}) && \text{配项法}\\
&=A\overline{B}+B\overline{C}+A\overline{B}C+\overline{A}\,\overline{B}C+\overline{A}BC+\overline{A}B\,\overline{C} && \text{分配律}\\
&=(A\overline{B}+A\overline{B}C)+(B\overline{C}+\overline{A}B\,\overline{C})+(\overline{A}\,\overline{B}C+\overline{A}BC) && \text{结合律}\\
&=A\overline{B}+B\overline{C}+\overline{A}C && \text{吸收律}
\end{aligned}
$$

方法二：

$$
\begin{aligned}
F(A,B,C)&=A\overline{B}+\overline{B}C+B\overline{C}+\overline{A}B\\
&=A\overline{B}+B\overline{C}+(\overline{B}C+\overline{A}B) && \text{结合律}\\
&=A\overline{B}+B\overline{C}+\overline{B}C+\overline{A}B+\overline{A}C && \text{反向包含律}\\
&=A\overline{B}+\overline{B}C+(B\overline{C}+\overline{A}C+\overline{A}B) && \text{结合律}\\
&=A\overline{B}+\overline{B}C+B\overline{C}+\overline{A}C && \text{包含律}\\
&=(A\overline{B}+\overline{A}C+\overline{B}C)+B\overline{C} && \text{结合律}\\
&=A\overline{B}+\overline{A}C+B\overline{C} && \text{包含律}
\end{aligned}
$$

方法三：函数 F 的卡诺图如图 3-4 所示。由卡诺图得函数 F 的最简“与或”式为：

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

最简“或与”式：

方法一：求上面得到的最简“与或”式的对偶式，并化成最简“与或”式：

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

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

对 F_d再求对偶，即得 F 的最简“或与”式：

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

方法二：函数$\overline{F}$的卡诺图如图 3-5 所示。由卡诺图得函数$\overline{F}$的最简“与或”式为：

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

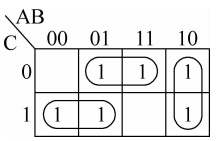

图 3-4　例 3-1 函数 F 的卡诺图

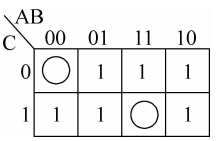

图 3-5　例 3-1 函数$\overline{F}$的卡诺图

使用反演律，可得 F 的最简“或与”式为：

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

由本例可见，同一个逻辑函数的化简方法有多种，应灵活选用。对于公式法，除了牢记基本公式之外，还应能够熟练应用。对公式的记忆，主要是记住其形式特征，具体化简中换成别的逻辑变量之后也应能看出来符合哪个公式。卡诺图法具有不用记忆公式、直观等特点，但要熟练掌握在卡诺图上画卡诺圈形成最小覆盖的方法。建议能用卡诺图化简的尽量使用卡诺图法。

例 3-2　用卡诺图法把逻辑函数 $F(A,B,C,D)=\prod_M(6,7,8,9)$化成最简“与或”式和最简“或与”式。

解：这里应注意，题目里给出的是逻辑函数的标准“或与”式（最大项之积式），要转换成标准“与或”式（最小项之和式）才能用卡诺图化简。因为同一个逻辑函数的最小项与最大项互为补集，所以可直接由最大项之积式写出最小项之和式，即：

$$F(A,B,C,D)=\prod_M(6,7,8,9)=\sum_m(0,1,2,3,4,5,10,11,12,13,14,15)$$

最简“与或”式：函数 F 的卡诺图如图 3-6 所示。由卡诺图得函数 F 的最简“与或”式为：

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

本例也可以按照图 3-7 所示的卡诺图进行化简，得函数 F 的最简“与或”式为：

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

最简“或与”式：

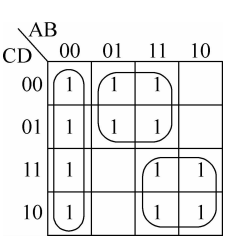

图 3-6　例 3-2 函数 F 的卡诺图

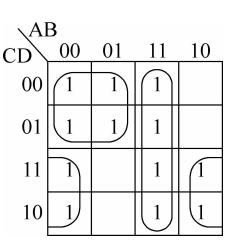

图 3-7　例 3-2 函数 F 的另一种卡诺图

方法一：利用函数 F 的最简“与或”式 $F=\overline{A}\,\overline{B}+B\,\overline{C}+AC$ 求函数 F 的对偶式 F_d，并将其化成最简“与或”式得：

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

对 F_d 再次求对偶，即可得 F 的最简“或与”式：

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

方法二：利用函数 F 的最简“与或”式 $F(A,B,C,D)=AB+\overline{A}\,\overline{C}+\overline{B}C$ 求函数 F 的对偶式 F_d，并将其化成最简“与或”式得：

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

对 F_d 再次求对偶，即可得 F 的最简“或与”式：

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

方法三：函数$\overline{F}$的卡诺图如图 3-8 所示。由卡诺图得函数$\overline{F}$的最简“与或”式为：

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

使用反演律，可得 F 的最简“或与”式为：

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

由本例可见，逻辑函数的最简“与或”式有时候不是唯一的，在卡诺图法中具体体现在选择卡诺圈形成最小覆盖的圈法不同，尽管形成最小覆盖的卡诺圈可能有多种圈法，但每种圈法的卡诺圈的数量和大小应该是相同的，也就是说多种最简“与或”式的复杂程度应该是相同的。同理，逻辑函数的最简“或与”式也有可能不是唯一的。

例 3-3　用卡诺图法把逻辑函数

$$F(A,B,C,D)=\sum_m(4,5,13,14,15)+\sum_d(0,1,2,3,6,7,10,11)$$

化成最简“与或”式和最简“或与”式。

解：最简“与或”式：函数 F 的卡诺图如图 3-9 所示。由卡诺图得函数 F 的最简“与或”式为：

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

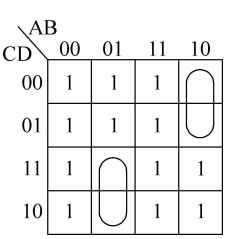

图 3-8　例 3-2 函数$\overline{F}$的卡诺图

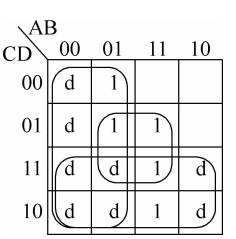

图 3-9　例 3-3 函数 F 的卡诺图