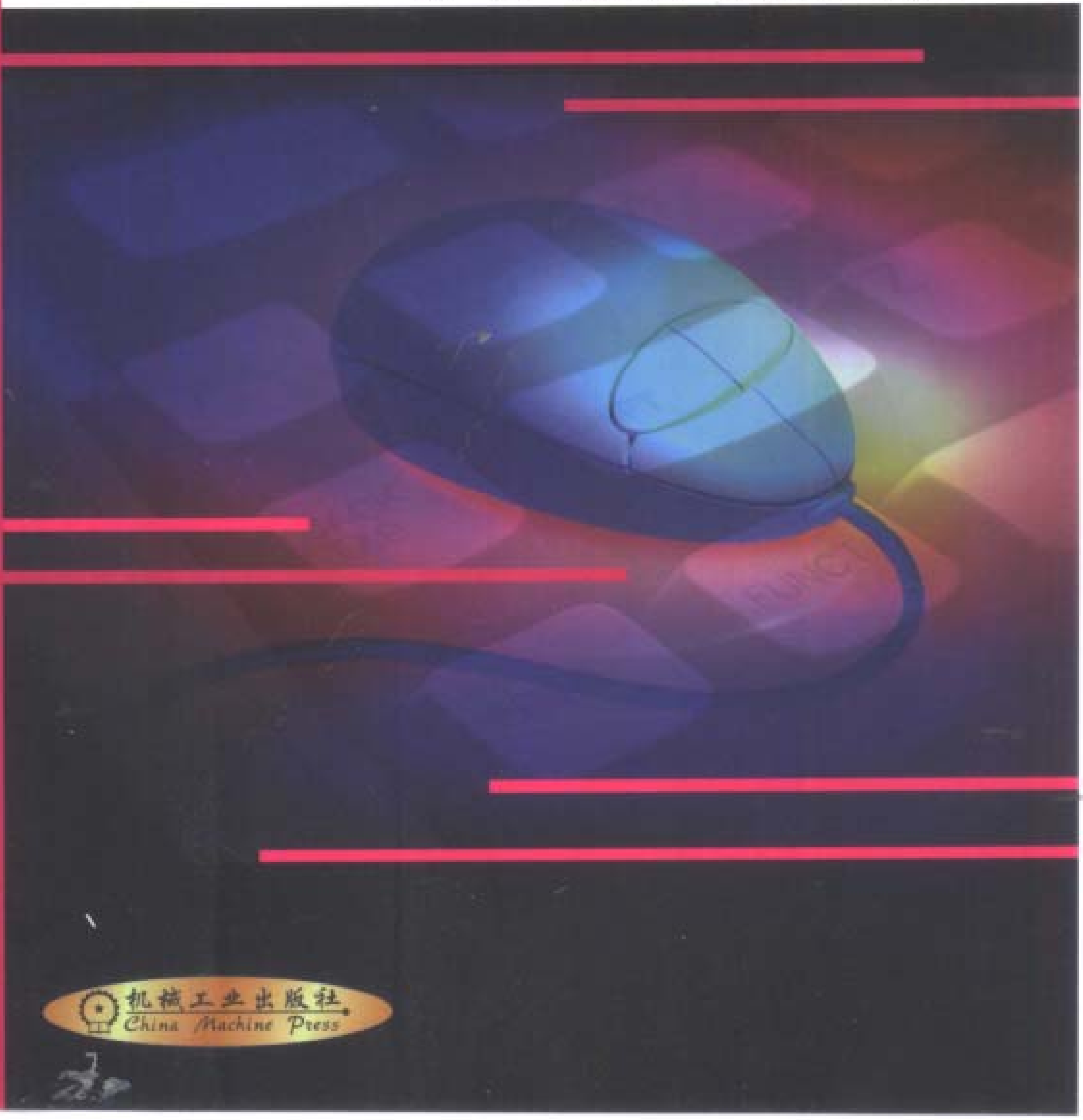
DSP控制器及其应用

章 云 谢莉萍 熊红艳 编著



机械工业出版社
China Machine Press

目　　录

前言

第1章　概述 …… 1

1.1　引言 …… 1
1.2　二进制、补码及其运算 …… 3
1.2.1　数制 …… 3
1.2.2　补码与加、乘运算 …… 4
1.3　DSP控制器的基本原理 …… 9

第2章　总体结构 …… 11

2.1　总线结构 …… 12
2.2　中央处理单元 …… 14
2.2.1　输入比例部分 …… 15
2.2.2　中央算术逻辑部分 …… 15
2.2.3　乘法部分 …… 16
2.3　辅助寄存器算术单元 …… 16
2.4　状态寄存器ST0和ST1 …… 17
2.5　存储器与I/O空间 …… 19
2.5.1　与外部存储器和I/O空间接口的信号 …… 19
2.5.2　程序存储器 …… 20
2.5.3　局部数据存储器 …… 21
2.5.4　全局数据存储器 …… 24
2.5.5　I/O空间 …… 25
2.6　程序控制 …… 26
2.7　时钟源模块 …… 28
2.8　系统复位 …… 32

第3章　片内外设 …… 35

3.1　事件管理模块 …… 35
3.1.1　通用定时器 …… 37
3.1.2　比较单元与PWM发生器 …… 50
3.1.3　捕获单元 …… 61
3.1.4　正交编码脉冲电路 …… 65
3.2　模/数转换模块 …… 66
3.2.1　结构概述 …… 67
3.2.2　模/数转换控制与操作 …… 70
3.3　SCI串行通信接口模块 …… 71
3.3.1　串行通信的工作原理 …… 71
3.3.2　串行通信接口模块SCI的结构 …… 74
3.3.3　多机通信 …… 83
3.4　SPI串行外设接口模块 …… 84
3.4.1　串行外设接口结构与工作原理 …… 85
3.4.2　SPI的多机通信 …… 91
3.4.3　SPI引脚功能的选择 …… 94
3.5　数字I/O端口 …… 95
3.5.1　数字I/O端口概述 …… 95
3.5.2　数字I/O端口寄存器 …… 97
3.6　看门狗与实时时钟 …… 98
3.7　中断管理系统 …… 102
3.7.1　DSP内核中断 …… 102
3.7.2　事件管理模块的中断 …… 106
3.7.3　系统模块中断 …… 110

第4章　指令系统 …… 112

4.1　寻址方式 …… 112
4.1.1　立即寻址 …… 112
4.1.2　直接寻址 …… 113
4.1.3　间接寻址 …… 114
4.2　句法格式 …… 117
4.2.1　汇编句法格式 …… 117
4.2.2　指令分类表 …… 119
4.3　传送指令 …… 123
4.4　算术操作指令 …… 142
4.5　逻辑运算指令 …… 155
4.6　分支转移指令 …… 161

第5章　应用实例 …… 170

5.1　基于空间矢量的通用变

频器 ………………………………… 170
5.2 快速傅里叶变换（FFT） ……… 173
5.2.1 快速傅里叶变换的基本原理 ………………………………… 173
5.2.2 快速傅里叶变换的DSP实现 ………………………………… 177

附录 ………………………………………… 182

附录A DSP240引脚说明 ……………… 182
附录B 可编程寄存器汇总……………… 189
附录C 指令汇总 ……………………… 191
参考文献………………………………… 197

第1章 概　述

1.1 引言

随着1946年第一台电子计算机的诞生，一场数字化的技术革命悄然地引发。如果说当初计算机的出现纯粹是为了解决日益复杂的计算问题，那么现在计算机已无处不在。自动控制与计算机几乎是同步地发展着。自动控制系统的核心问题是如何寻找和实现最佳的控制律。在A/D、D/A以及I/O技术出现并成熟之后，最佳控制律的实现问题就变成了最佳控制律的运算（代数、微分、积分等运算）问题，从而计算机作为自动控制系统的核心部分就是自然之事。

计算机真正在自动控制系统中发挥重要作用还得缘起20世纪70年代微处理器（Microprocessor）的出现，它使得计算机在体积、价格上得以突破，为计算机在各种技术领域的应用提供了可能。微处理器即计算机的中央处理单元（CPU）和控制单元的集成，它配上一定的存储器、I/O接口和其它外设，就可构成自动控制系统的通用控制器。70年代Intel公司的8080，Motorola公司的M6800和Zilog公司的Z80是当时三个著名的8位微处理器。

随着大规模集成电路技术不断改进，一方面微处理器由8位向16位、32位甚至64位发展，在配上外围设备后便形成单板机或微型机（也称为个人计算机，Personal Computer），使得计算机不但在计算、控制中应用，而且使得计算机逐步地走人了家庭；另一方面将微处理器与外围设备集成到一块芯片形成单片机（Single _ chip Microcomputer），以适应控制器体积越来越小的工程要求。

正是由于单片机的出现，计算机在控制领域的应用又得到了一次突破。单片机不但小巧、成本低，而且由于众多设备集成到了一块芯片上带来了功耗小和抗干扰能力强的优点。另外，它可以方便地组成各种智能式控制设备和仪器，做到机电仪一体化；也可以方便地实现多机和分布式控制，使整个控制系统的效率和可靠性大为提高。单片机有许多类型，有低档的1位、4位和8位单片机，也有高档的8位、16位单片机。其中Intel公司的MCS-51系列、Motorola公司的68系列和Zilog公司的Z8系列为大家所知。

DSP（Digital Signal Processor）实际上也是一种单片机，它同样是将中央处理单元、控制单元和外围设备集成到一块芯片上。DSP最早是针对数字信号处理，特别是语音、图像信号的各种处理而开发的。由于这类信号处理的算法复杂，要求DSP必须具有强大快速的运算能力。因此，DSP有别于普通的单片机，它采用了多组总线技术实现并行运行机制，从而极大地提高了运算速度，也提供了非常灵活的指令系统。近些年来，各种集成化单片DSP的性能不断得以改进，相应的软件和开发工具日臻完善，价格迅速下降，使得DSP在控制领域的应用倍受关注。

在DSP领域中，德州仪器（TI）公司的产品及其配套技术与开发工具最有强大的竞争力，其中TMS320 DSP是它的代表系列，本书以TMS320C24X进行介绍。TMS320C24X也称为DSP控制器，是TI公司专门针对电机、逆变器、机器人、数控机床等控制而设计的。它以

C2XLP 16bit 定点 DSP CPU 为内核，配置了完善的外围设备，包括事件管理模块（EV）、A/D 转换模块（ADC）、串行通信接口模块（SCI）、串行外设接口模块（SPI）、中断管理系统和系统监视模块，其中事件管理模块（EV）含有通用定时器、比较器、PWM 发生器、捕获器。

DSP 控制器的结构和主要特性如下：

□中央处理单元

●32 位中央算术逻辑单元（CALU）。

●32 位累加器。

●16 位×16 位乘法器。

●3 个比例移位器。

●间接寻址用的 8 个 16 位辅助寄存器和它的辅助算术单元（ARAU）。

□存储器

●544 字片内双口 RAM，其中 288 字用于数据，256 字用于程序/数据。

●16K 字片内 ROM 或 FLASH EEPROM，用作程序存储器。

●244K 字可寻址空间，程序存储空间 64K 字，数据存储空间 64K 字，I/O 空间 64K 字，还有 32K 字全局存储空间。

●外部有 16 位地址总线，16 位数据总线，支持软件、硬件等待状态。

□程序控制

●4 级流水线操作。

●8 级硬件堆栈。

●6 个外部中断：电源保护、复位、不可屏蔽中断（NMI）和 3 个可屏蔽中断。

□指令集

●源代码与定点 TMS320C2X、C2XX、C5X 兼容。

●单周期相乘/累加指令。

●单指令重复操作。

●程序/数据存储器中的块移动。

●丰富的变址寻址能力。

●具有基于 2 的 FFT 倒位序变址寻址能力。

□事件管理模块

●3 个 16 位通用定时器。

●3 个全比较/PWM 单元。

●3 个简单比较/PWM 单元。

●4 个捕获单元。

□2 个 8 通道 10 位 A/D 转换器

□串行异步数字通信接口模块（SCI）

□串行外设接口模块（SPI）

□中断管理系统

□由看门狗和实时中断定时器组成的系统监视模块

□28 个可独立编程的 I/O 引脚

□速度

●单周期指令执行时间为 50ns（20MIPS）。

□电源

● 5V 或 3.3V 静态 CMOS 工艺。

● 4 种降低功耗的方式。

1.2 二进制、补码及其运算

DSP 控制器就是一款高性能的单片机。运算是单片机（计算机）的主要任务，为此首先简单介绍二进制、补码及其运算的实现方法。

二进制数是计算机原理看似最简单的内容，但却是最容易弄混淆的。对于计算机，它的基本任务是：从某个“地方”取数，按某种规律运算，再把结果放到某个“地方”。这里“地方”是指存储器、I/O 接口等。因此，地址与数据是计算机中两个基本概念。由于计算机只存在“0”、“1”两种状态，因此在计算机中地址、数据都是二进制数表示。另外，数据存在着正数与负数、整数与小数，为了区别它们还必须产生用二进制数表示的编码系统。这样一来，在计算机中给出的一个二进制数到底是地址还是数据、是正数还是负数、是整数还是小数，常常让人眼花缭乱。本节着重介绍二进制数的编码系统及其特点，至于寻址问题将在第 4 章介绍。

1.2.1 数制

在日常生活中经常使用的是十进制数，由 0～9 十个符号组成，按照逢十进一的规则运算。由于计算机只存在“0”、“1”两种状态，故计算机中的数用二进制表示。二进制数只有 0 和 1 两个符号，按照逢二进一的规则运算。与二进制密切相关的是十六进制，它由 0～9、A～F 共 16 个符号组成，按照逢十六进一的规则运算。为了以示区别，二进制数以后缀“B”表示，十六进制数以后缀“H”表示。二进制数与十进制数、十六进制数与十进制数的关系如下：

$$[b_{n-1}b_{n-2}\cdots b_0 \cdot b_{-1}b_{-2}\cdots b_{-m}]_B = b_{n-1}\times 2^{n-1} + b_{n-2}\times 2^{n-2} + \cdots + b_0\times 2^0 + b_{-1}\times 2^{-1} + b_{-2}\times 2^{-2} + \cdots + b_{-m}\times 2^{-m}$$

$$[h_{n-1}h_{n-2}\cdots h_0 \cdot h_{-1}h_{-2}\cdots h_{-m}]_H = h_{n-1}\times 16^{n-1} + h_{n-2}\times 16^{n-2} + \cdots + h_0\times 16^0 + h_{-1}\times 16^{-1} + h_{-2}\times 16^{-2} + \cdots + h_{-m}\times 16^{-m}$$

例 1.1 将下列二进制数、十六进制数转换为十进制。

$$1011.101_B = 1\times 2^3 + 0\times 2^2 + 1\times 2^1 + 1\times 2^0 + 1\times 2^{-1} + 0\times 2^{-2} + 1\times 2^{-3} = 11.625$$

$$1011.101_H = 1\times 16^3 + 0\times 16^2 + 1\times 16^1 + 1\times 16^0 + 1\times 16^{-1} + 0\times 16^{-2} + 1\times 16^{-3} = 4113.0627$$

由于 $2^4=16$，因此 1 位十六进制数可用 4 位二进制数表示，它们之间的关系见表 1.1。这样一来，二进制数与十六进制数的转换非常方便。

例 1.2 二进制数与十六进制数的转换。

$$[1AB.3E]_H = [0001\quad 1010\quad 1011.0011\quad 1110]_B$$

$$[10101101.11]_B = [1010\quad 1101.1100]_B = [AD.C]_H$$

计算机除了受“0”、“1”两种状态的限制外，还受到数据长度的限制。在计算机中数据总线的位数是有限的，一般有 8 位、16 位或 32 位。用 8 位二进制数表示的数据称为一个字节，

可以表示 0～255 的无符号整数；用 16 位二进制数表示的数据称为一个字，可以表示 0～65535 的无符号整数；用 32 位二进制数表示的数据称为一个双字，可以表示 $0\sim2^{32}-1$ 的无符号整数。一般情况下，用 n 位的二进制数表示无符号整数的范围是：$0\sim2^{n}-1$。由于数据位数的限制，在进行数据运算时要注意数据表示的范围，否则将会发生溢出错误。

表 1.1 十进制与二进制、十六进制的关系

十进制	十六进制	二进制	十进制	十六进制	二进制
0	0	0000	8	8	1000
1	1	0001	9	9	1001
2	2	0010	10	A	1010
3	3	0011	11	B	1011
4	4	0100	12	C	1100
5	5	0101	13	D	1101
6	6	0110	14	E	1110
7	7	0111	15	F	1111

除了要注意数据表示的范围外，还要关注数据的符号。数据有正、负两种，正好可用"0"、"1"、两种状态来表示，以"0"表示正数，以"1"表示负数。一般把符号位放在最高位，其余位表示数据。在计算机中，以这种格式表示的二进制数称为原码，n 位的原码表示有符号整数的范围是：$-2^{n-1}\sim2^{n-1}-1$。

定义 1（二进制原码）：整数 X 用 n 位二进制原码表示，其规则如下：

1. 若 $X\geqslant0$，它的最高位为 0，其余位是 X 的二进制数；

2. 若 $X<0$，它的最高位为 1，其余位是 $|X|$ 的二进制数。

例 1.3 用二进制数表示无符号、有符号整数。

无符号整数：$[0101\ 1101]_B=93$　　$[1101\ 1101]_B=221$

有符号整数：$[0101\ 1101]_{原}=93$　　$[1101\ 1101]_{原}=-93$

在计算机中，无符号整数的运算比较容易，因为一般情况下计算机中的加法器与乘法器都是按无符号整数设计的，这样一来只需注意数据的范围，不要发生溢出错误便可。对于有符号整数的运算就要麻烦一些，由于最高位是符号标志，如果直接参与运算，会带来不正确的结果。因此，对于有符号整数的运算分两步进行：先按规则运算符号位并寄存；然后屏蔽掉符号位，按无符号数进行运算，再把结果与寄存的符号位拼接成最后的结果。

可以看出，在计算机中直接进行有符号整数的运算是不方便的，有没有可能经过一定变换，使得有符号整数的运算能够一步完成？下面介绍的二进制补码就可以做到这一点。

1.2.2 补码与加、乘运算

1. 补码与整数加法 在计算机中加、乘运算是最基本的运算，对于有符号整数以及小数的加、乘运算能否有一套统一的编码系统来完成是一件非常有意义的事。下面首先从有符号整数的加法谈起，引入补码并同时介绍其特点，然后分别介绍补码形式的整数与小数的乘法运算。

前面说到，计算机的数据长度是有限的。对于 8 位计算机，超过 2^8 就会溢出，这跟数学

中的同余（mod）运算是一样的，即

$$X=[X+2^8]\quad \bmod 2^8$$

为了讨论方便，设 X 和 Y 是正整数，N 是大于 X 和 Y 的正整数，那么

$$\begin{aligned}X+(-Y)&=[N+X+(-Y)]\quad \bmod N\\&=[X+(N-Y)]\quad \bmod N\end{aligned}$$

从上式可以看出，如果用 $N-Y$（这是大于 0 的数）代表（$-Y$），则有符号的加法运算转换成了无符号的加法运算。下面给出二进制补码表示的定义：

定义 2（二进制补码）：整数 X 用 n 位二进制补码表示，其规则如下：

1）若 $X\geqslant 0$，它的原码就是它的补码；

2）若 $X<0$，它的补码为 $2^n-|X|$。

原码和补码都可以表示有符号数，它们之间的区别在于：对于原码表示，其最高位仅仅是一个符号标志，不能直接参与数据的加法运算，否则会引起错误的结果；对于补码表示，则不一样，其最高位既是符号标志，也是一位可以参与运算的数据。按二进制补码表示的数据，无论正数还是负数可以直接进行加法运算，但其结果按二进制补码解释。

例 1.4 设 $n=8$，$X=66$，$Y=-24$，写出它们的原码和补码，并求 $X+Y=$？

1）按原码方式直接相加，其结果是错误的。

$$[X]_{原}=[0100\ 0010]_{原}\quad [Y]_{原}=[1001\ 1000]$$

$$\begin{aligned}[X]_{原}+[Y]_{原}&=[0100\ 0010]_{原}+[1001\ 1000]_{原}\\&=[1101\ 1010]_{原}=-90\end{aligned}$$

2）按补码方式直接相加，其结果是正确的。

$$[X]_{补}=[0100\ 0010]_{补}\quad [Y]_{补}=2^8-24=232=[1110\ 1000]_{补}$$

$$\begin{aligned}[X]_{补}+[Y]_{补}&=[0100\ 0010]_{补}+[1110\ 1000]_{补}\\&=[1\ 0000\ 0000]+[0010\ 1010]_{补}\quad \bmod\ 2^8\\&=[0010\ 1010]_{补}=42\end{aligned}$$

上面的讨论说明了采用二进制补码有益于有符号的加法运算，但是按定义 2 求负数的补码需要做一次减法，这与采用补码的初衷是相悖的，有无其它简易的方法求解补码呢？仔细的推敲可得到与定义 2 等价的另一种定义：

定义 3（二进制补码）：整数 X 用 n 位二进制补码表示，其规则如下：

1）若 $X\geqslant 0$，它的原码就是它的补码；

2）若 $X<0$，它的补码为：其原码的符号位不变，数据位求反加 1。

按照定义 3 可以很容易得到有符号整数的补码，从而避免了定义 2 求补码要做减法的困难。

2. 补码运算的溢出 要注意二进制补码表示也有一个数据范围的问题，n 位二进制补码可以表示 $-2^{n-1}\sim 2^{n-1}-1$ 范围内的数。在做二进制补码加法的时候，要特别注意溢出的问题。下面先看一个例题。

例 1.5 按二进制补码求 $X+Y=$？并注意符号位与数据的最高位的进位情况。

1）设 $n=8$，$X=110$，$Y=94$

$$[X]_{补}=[0110\ 1110]_{补}\quad [Y]_{补}=[0101\ 1110]_{补}$$

$$\begin{array}{r} 0110\ 1110 \\ +\ \ 0101\ 1110 \\ \hline 1100\ 1100 \end{array}$$ 符号位没有进位，数据的最高位有进位。

$[X]_补+[Y]_补=[1100\ 1100]_补=-52$

2)设 $n=8, X=-92, Y=-102$

$[X]_补=[1010\ 0100]_补\quad [Y]_补=[1001\ 1010]_补$

$$\begin{array}{r} 1010\ 0100 \\ +\ \ 1001\ 1010 \\ \hline 1\ 0011\ 1110 \end{array}$$ 符号位有进位，数据的最高位没有进位。

$[X]_补+[Y]_补=[0011\ 1110]_补=62$

3)设 $n=8, X=44, Y=82$

$[X]_补=[0010\ 1100]_补\quad [Y]_补=[0101\ 0010]_补$

$$\begin{array}{r} 0010\ 1100 \\ +\ \ 0101\ 0010 \\ \hline 0111\ 1100 \end{array}$$ 符号位没有进位，数据的最高位没有进位。

$[X]_补+[Y]_补=[0111\ 1100]_补=126$

4)设 $n=8, X=-22, Y=-44$

$[X]_补=[1110\ 1010]_补\quad [Y]_补=[1101\ 0100]_补$

$$\begin{array}{r} 1110\ 1010 \\ +\ \ 1101\ 0100 \\ \hline 1\ 1011\ 1110 \end{array}$$ 符号位有进位，数据的最高位有进位。

$[X]_补+[Y]_补=[1011\ 1110]_补=-66$

从前两个算式看，最后的结果是不正确的，因为两个正数相加不可能为负数，两个负数相加也不可能为正数。引起这个结果的原因是相加的结果超出了 8 位二进制补码表示的范围，发生了溢出。所以二进制补码的溢出，不能只看最高位是否有进位，而需要看最高两位（符号位与数据最高位）的进位情况。如果最高两位都有进位或都无进位，则不发生溢出，否则将产生溢出。不少计算机的状态标志寄存器专门有一个溢出位 OV，就是为补码运算所设。

3. 数据扩展与符号扩展 在计算机的计算中，为了提高计算精度，常常需要用多字节来表示一个数，这就存在数据扩展的问题。对于无符号整数，这种扩展比较容易，在前面补 0 即可。对于有符号整数仅仅补 0 是不够的。下面以一个实例说明补码形式的数据扩展并总结出它的规律。

例 1.6 设 $n=8$，$X=66$，$Y=-24$，写出它们的补码：

$[X]_补=[0100\ 0010]_补\quad [Y]_补=[1110\ 1000]_补$

设 $n=16$，$X=66$，$Y=-24$，写出它们的补码：

$[X]_补=[0000\ 0000\ 0100\ 0010]_补$

$[Y]_补=[1111\ 1111\ 1110\ 1000]_补$

仔细分析例 1.6 可以看出，采用二进制补码表示的数据扩展，实际上就是符号扩展，即正整数前面全部补 0，负整数前面全部补 1。正因为如此，在有的计算机中专门提供了符号扩展方式，就是为了方便实现二进制补码形式的数据扩展。

4. 整数补码的乘法 下面探讨二进制补码的乘法。设 X、Y 是 n 位有符号的整数，其范

围为-2^{n-1}到$2^{n-1}-1$，$N=2^n$。二进制补码的乘法有下面三种情况：

- $X\geqslant 0$，$Y\geqslant 0$。这与无符号乘法一样。
- $X=0$，$Y<0$。这是一种平凡的情况，按补码相乘的结果肯定是 0。
- $X>0$，$Y<0$。用补码实现乘法的过程为：

$$\begin{aligned}\{X\times(N-|Y|)\} \quad \bmod N &= \{NX-N|Y|\} \quad \bmod N\\ &= \{N(X-1)+(N-X|Y|)\} \quad \bmod N\\ &= N-X|Y|\end{aligned}$$

注意到整数 $X>0$，则 $X-1\geqslant 0$。因此，上面的运算表明按照补码相乘其结果也是相应的补码。

- $X<0$，$Y<0$。用补码实现乘法的过程为：

$$\begin{aligned}\{(N-|X|)\times(N-|Y|)\} \quad \bmod N &= \{N^2-N(|X|+|Y|)+|X||Y|\} \quad \bmod N\\ &= \{N\{N-(|X|+|Y|)\}+XY\} \quad \bmod N\\ &= XY\end{aligned}$$

由于$|X|\leqslant 2^{n-1}$，$|Y|\leqslant 2^{n-1}$，所以 $N-(|X|+|Y|)\geqslant 0$。因此，对于二个负的整数按照补码相乘其结果也是相应的补码（正整数的补码是其本身）。

综上所述，按照二进制补码进行的有符号整数的乘法毋须关注它的符号，其结果按补码进行解释。在进行二进制乘法运算时有一点需要注意，二个 n 位二进制整数相乘可以得到 $2n$ 位二进制整数的乘积结果。为了得到正确的 $2n$ 位二进制整数的乘积结果，需首先将 n 位二进制补码的乘数和被乘数扩展到 $2n$ 位，然后再按二进制补码的方式相乘，根据上面的讨论，其结果肯定是正确的。总的一句话，要得到 N 位二进制补码的乘积，其乘数与被乘数应为 N 位二进制补码。

例 1.7 设 $X=21=[0001\ 0101]_{补}$，$Y=-3=[1111\ 1101]_{补}$，求 $XY=?$

若只需 8 位二进制乘积结果，则

```
                  11111101
      ×           00010101
      --------------------
                  11111101
              11  111101
            1111  1101
      --------------------
        00010100  11000001
```

所以 $XY=[1100\ 0001]_{补}=-63$

若需要 16 位二进制乘积结果，则先要将 X、Y 进行扩展，即

$X=21=[0000\ 0000\ 0001\ 0101]_{补}$，$Y=-3=[1111\ 1111\ 1111\ 1101]_{补}$

```
                           11111111 11111101
              ×            00000000 00010101
              ------------------------------
                           11111111 11111101
                        11 11111111 111101
                      1111 11111111 1101
              ------------------------------
     00000000 00010100     11111111 11000001
```

所以 $XY=[11111111\ 1100\ 0001]_{补}=-63$

5. 小数的补码与加、乘运算 前面较详细讨论了采用二进制补码的有符号整数的加、乘运算，那么对于有符号的小数，能否移植前面的结论？下面给出详细的讨论。

首先，有符号小数原码的表示可以从整数原码进行自然推广。即最高位为符号位为“0”表示正数，为“1”表示负数；其余位为数据位。不失一般性，设 n 位二进制小数由 m 位整数位（包括符号位）和 k 位小数位组成（$n=m+k$）。这种表示方式也成为 Q_k 方式。若 $k=0$，则退化为完全整数（Q_0）；若 $m=1$，则为完全小数，或称标准小数（若 $n=16$，为 Q_{15}）。

例 1.8 用 8 位二进制原码表示下面的十进制数（$m=4$，$k=4$）：

$$5.625=[0101.1010]_{原}$$

$$-3.25=[1011.0100]_{原}$$

有符号小数补码的定义可以从整数二进制补码的定义进行推广。对于定义 3 可以完全照搬，即对于正数，其原码就是它的补码；对于负数，将其原码的符号位不变，其余位求反加 1。而对于定义 2 有如下自然的推广。

定义 4（一般性的二进制补码）：设 x 是由 m 位整数位（包括符号位）和 k 位小数位组成的 n 位二进制数，$n=m+k$。它的补码表示如下：

1）若 $x\geqslant 0$，它的原码就是它的补码；

2）若 $x<0$，它的补码为 $2^m-|x|$。

例 1.9 用 8 位二进制补码表示下面的十进制数（$m=4$，$k=4$）。

采用定义 3 求：$5.625=[0101.1010]_{补}$ $\quad -3.25=[1100.1100]_{补}$

采用定义 4 求：$-3.25=2^4-[0011.0100]_B=[10000.0000]_B-[0011.0100]_B$

$$=[1100.1100]_{补}$$

定义 3 和定义 4 都是二进制补码的一般性定义，它们是完全等效的，有兴趣的读者可自行证明。根据定义 3 或定义 4 给出的带小数二进制补码，其加法、乘法运算与整数补码的加法、乘法运算是否同样简单呢？答案是肯定的。

对于带小数的加法，首先要注意小数点位置要对齐，即需要相加的两个小数的格式要一致（m 与 k 的值要一样），不然的话要通过移位操作使它们一致。另外，在计算机中并没有小数点的硬件设置，小数点的位置是一种“假想”，也就是说计算机中的加法器（也包括乘法器）都是按无符号整数来运算的。基于这两点，下面简单推导有符号小数二进制补码加法实现的原理。

设 x、y 是由 m 位整数位（包括符号位）和 k 位小数位组成的 n（$n=m+k$）位二进制数，取 $X=x2^k$，$Y=y2^k$，即 X、Y 变为完全整数，也就是将小数点右移 k 位。按照前面讨论的结果知，完全整数可以采用补码形式直接相加；然后，将相加结果的小数点左移 k 位便是 x 与 y 相加的结果。不失一般性，设 $x>0$，$y<0$，则

$$[x+y]_{补}=[X+(2^n-|Y|)]\times 2^{-k}=[x2^k+(2^{m+k}-|y|)2^k]\times 2^{-k}$$

$$=[x+(2^m-|y|)]$$

这个推导说明，只要 x、y 的补码按照定义 3 或定义 4 给出，其加法可以直接相加。注意对于由 m 位整数位（包括符号位）和 k 位小数位组成的 n（$n=m+k$）位二进制数所表示的范围是 -2^{m-1} 到 $2^{m-1}-2^{-k}$，超出此范围将发生溢出。

有符号小数的乘法实现原理与有符号小数的加法是类似的。思路都是先把它看成是完全整数（相当于将小数点右移 k 位），然后按完全整数进行相乘，最后再确定小数点的位置。如

果相乘的两个数据的小数位都是 k 位，则最后的结果要将小数点左移 $2k$ 位；如果相乘两个数据的小数位分别是 k_1 位和 k_2 位（这一点与加法不同，两个数据可以有不同的数据格式），则最后的结果要将小数点左移 k_1+k_2 位。

不失一般性，设 $x>0$ 由 m_1 位整数位（包括符号位）和 k_1 位小数位组成，$y<0$ 由 m_2 位整数位（包括符号位）和 k_2 位小数位组成，$n=m_1+k_1=m_2+k_2$。取 $X=x2^{k_1}$，$Y=y2^{k_2}$，那么

$$
\begin{aligned}
[x\times y]_{补} &= 2^{-(k_1+k_2)}[X\times Y]_{补}=[X\times(2^n-|Y|)]2^{-(k_1+k_2)}\\
&=[x2^{k_1}\times(2^n-|y|2^{k_2})]\times 2^{-(k_1+k_2)}\\
&=x\times(2^{m_2}-|y|)
\end{aligned}
$$

例 1.10 设 $x=5.25=[000101.01]_{补}$，$y=-0.75=[111111.01]_{补}$，求 $xy=?$

若只需 8 位二进制乘积结果，则

$$
\begin{array}{r|l}
 & 11111101 \\
\times & 00010101 \\
\hline
 & 11111101 \\
11 & 111101 \\
1111 & 1101 \\
\hline
00010100 & 11000001
\end{array}
$$

所以 $xy=[1100.0001]_{补}=-3.9375$

若需要 16 位二进制乘积结果，则先要将 x、y 进行扩展，即

$x=5.25=[0000\ 0000\ 0001\ 01.01]_{补}$，$y=-0.75=[11111111\ 1111\ 11.01]_{补}$

$$
\begin{array}{r|l}
 & 11111111\ 11111101 \\
\times & 00000000\ 00010101 \\
\hline
 & 11111111\ 11111101 \\
11 & 11111111\ 111101 \\
1111 & 11111111\ 1101 \\
\hline
00000000\ 00010100 & 11111111\ 11000001
\end{array}
$$

所以 $xy=[11111111\ 1100\ 00.01]_{补}=-3.9375$

上面讨论了二进制补码的整数、小数与有符号数的加乘运算原理，可以看出采用二进制补码带来了许多便利。因此，在实际应用时应尽可能转换成二进制补码的形式存储数据。在 DSP 控制器中有一个 16 位×16 位的乘法器，其结果是 32 位。由于该乘法器自动实现乘数与被乘数的数据扩展（16 位到 32 位），因此不需要用户编程干预。

1.3 DSP 控制器的基本原理

无论是微处理器、单片机还是 DSP 控制器，它们的工作原理是基本一致的。不外乎要做的工作都是：从存储器、I/O 接口等地方取数，按某种规律运算，再把结果放到存储器、I/O 接口等地方。因此，在其工作过程中数据流与地址流占统治地位。为了实现数据流、地址流有序地管理和控制，采用数据总线和地址总线是一种最佳的结构方式。数据总线和地址总线就像两条高速公路，数据信息与地址信息分别在其上快速地流动。中央处理单元（CPU）、程序存储器、数据存储器和内部外设等功能模块分别挂接在数据总线和地址总线上。中央处理

单元是控制中心，由它指挥当前时刻谁可以占用数据总线或地址总线，同时它还可以进行有关的运算；程序存储器是物理芯片与人的交接面，由人编写程序指令并写入到程序存储器中，体现了人的意志，中央处理单元只能根据程序的流程进行指挥不能随意发挥；数据存储器用于记录工作过程中的原始数据、中间结果和最后结论；内部外设是集成在芯片内部的与外部世界进行信息交换的功能模块，一般包含I/O、A/D、串行通信等。另外，数据总线和地址总线一般情况下都延伸到芯片外部（到引脚上）。图1.1说明了上述情况。

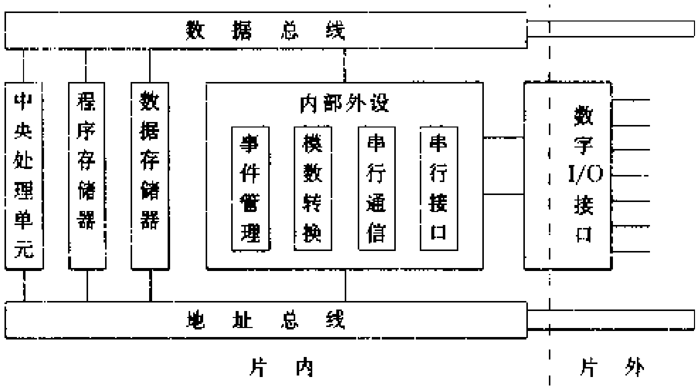


图1.1 DSP控制器的基本原理

一般的微处理器的数据总线和地址总线是单总线方式，相当于一辆车在只有一条道的高速公路上跑，这辆车分时地为大家服务。DSP控制器与此不同，采用多总线方式，相当于是多条道的高速公路，这样一来多辆车可以同时在其上行驶，极大地加快了运行速度。这实质上是一种并行机制。

数据和地址是贯穿任何一种微处理器设计、编程的两个基本概念，特别是地址，它就是数据源、专用寄存器、I/O的代表。每一个存储器、寄存器都有地址，这个容易为大家接受。对于可编程的功能模块（片内的或片外的）它也有地址，有时让人费神。准确地说，对可编程的功能模块的操作，实际上是对它的寄存器（控制的、数据的等）进行操作，这些寄存器必须有唯一地址，否则会引起工作混乱。对于片内外设的功能模块各寄存器的地址是由芯片厂家确定的，应仔细查看手册，不可更改。片外设功能模块各寄存器的地址与所连接的外部地址总线有关，这是设计者一个重要的设计任务，即给每个功能模块分配地址，一旦完成设计做好印制电路也就被固定了下来。

当设计者明确了存储器地址空间、I/O地址空间、片内或片外设功能模块各寄存器的地址后，程序设计的工作就有了一个明晰的轮廓，剩下的任务就是如何组织数据，采用什么控制律或算法，以何种流程来实现。

DSP控制器在工作时要注意如下问题：

●加电后，中央处理单元自动从复位地址（0000H）取出首条指令。

●指令处理周期分为：取指令、指令译码、取操作数、执行指令四个阶段。

●一条指令执行完后，顺序处理下一条指令，除非遇到分支指令或中断响应。

●数据的地址由指令的寻址方式和相应的操作数确定，这一点非常重要。

●对外部世界的访问有两种方式：通过I/O端口或者特定的映射寄存器，前者使用I/O传送指令，后者使用存储器传送指令。

第2章　总体结构

DSP 控制器是一款高性能的单片机。DSP 控制器的总体结构有许多独特的地方：一是采用多组总线结构实现并行处理机制，允许 CPU 同时进行程序指令和存储数据的访问；二是采

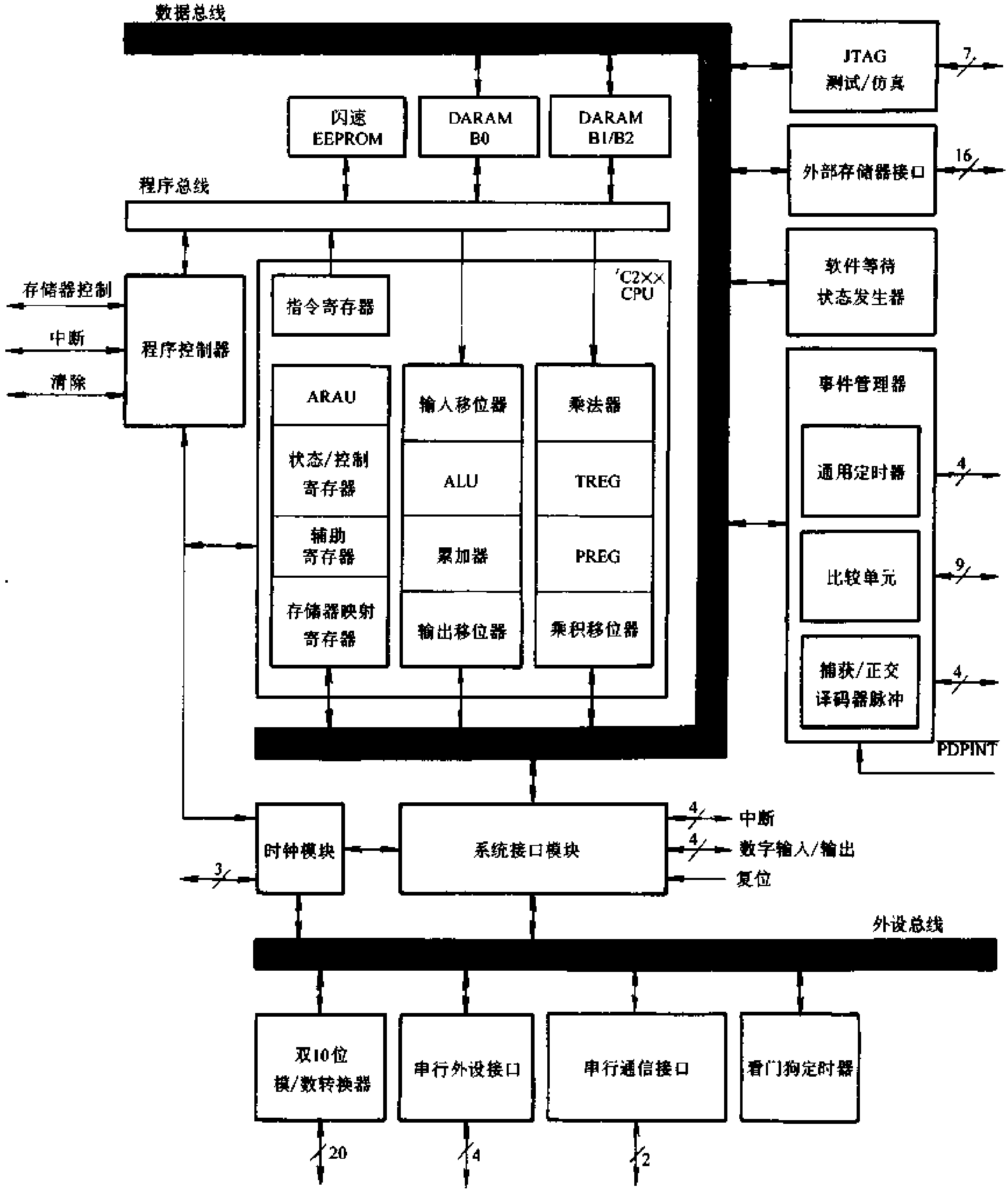


图 2.1　DSP 控制器的总体结构图

用独立的累加器和乘法器，使得复杂的乘法运算能快速进行；三是累加器和乘法器分别连接了比例移位器，使得许多复杂运算或者运算后的定标能在一条指令中完成；四是有丰富的寻址方式，可方便灵活的编程；五是有完善的片内外设，可以构成完整的单片系统。本章主要介绍DSP控制器的总体结构，包括总线结构、中央处理单元、存储器与I/O空间以及系统的复位。图2.1是DSP控制器的总体结构图。

2.1 总线结构

目前，在控制领域使用的各种微处理器芯片（各类CPU、单片机等）的基本任务，就是从某个地方（内、外部存储器或外部接口）取得数据，经过算术或逻辑运算，然后放到相应的地方上。为了区别不同的数据源，需要给其赋予一个独立的地址。数据和地址是任何微处理器都要面对的两个基本要素。因此，在微处理器芯片中采用基于数据/地址总线的结构是最佳的选择。在总线上可以挂接中央算术逻辑单元（CALU）、存储器、定时器等功能模块。通过地址总线和某些控制信号线（与指令密切相关），使得在某一时刻仅仅让某个数据源占用数据总线。这样一来，在地址总线和控制总线的共同作用下，使得数据总线的数据得以有序的流动。

总线结构是计算机体系结构中最基本的结构，它提供了一种标准的接口方式。功能模块之间的信息交换，都可解释为“在什么地址存放数据”或“从什么地址取回数据”。数据与地址成为密不可分的一对伙伴。具备数据与地址接口方式的功能模块都可以挂接到数据/地址总线上。数据/地址总线是双向的，为了保证数据通畅流动，要在中央处理单元统一指挥下按节拍进行工作。

总线结构是各种微处理器芯片的总干道，它的性能（响应速度、位宽、负载能力等）在很大程度上决定了微处理器芯片的性能。为了提高处理速度，一方面可以通过新的工艺使得微处理器芯片能够采用更高频率的晶振以加快响应的速度；另一方面可以加宽数据总线（32位或64位）以增加高精度复杂运算的指令。除此之外，加快处理速度的最佳方案是采用并行机制。一般情况下，总线的操作时序分为四个独立的阶段：取指令、指令译码、取操作数和执行指令，这四个阶段分别面向程序读、数据读和数据写。如果将数据/地址总线分开为三组数据/地址总线，分别对应程序读、数据读和数据写三种情况，这样一来就可以使总线操作时序的四个独立阶段并行处理，从而极大地加快微处理器芯片的处理速度。可以形象地理解，单总线方式就像是一辆车在只有一条道的高速公路上跑；而多组总线方式就像是多辆车在多条道的高速公路上跑。后者的运行速度和效率肯定要超过前者。

DSP控制器就是采用了多组总线的结构，图2.2是DSP控制器的总线结构图。其中内部地址总线分为了三条总线：

- 程序读地址总线（PAB），提供读程序的地址；
- 数据读地址总线（DRAB），提供读数据存储器的地址；
- 数据写地址总线（DWAB），提供写数据存储器的地址。

内部数据总线也对应分为三条总线：

- 程序读数据总线（PRDB），将指令代码中的立即数以及表信息传送到CPU；
- 数据读数据总线（DRDB），将数据存储器的数据传送到CPU；
- 数据写数据总线（DWDB），将处理后的数据传送到数据存储器和程序存储器。

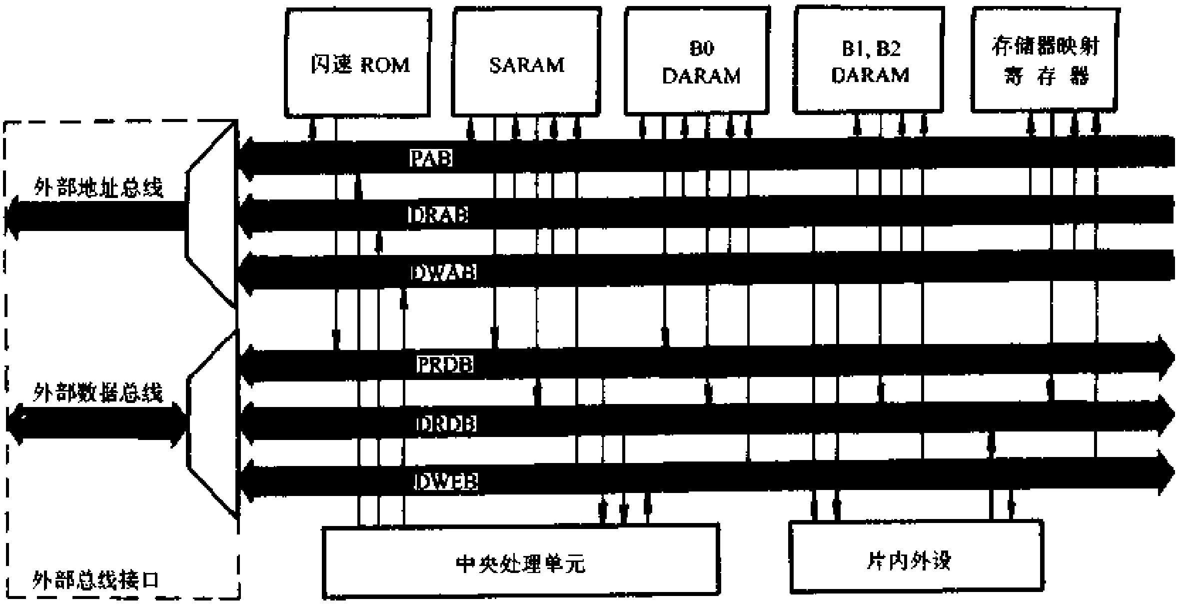


图 2.2 DSP 控制器的总线结构图

外部数据/地址总线仍为单一形式，这使得众多的外围芯片可与其兼容。

每条指令的执行过程可以分为四个阶段：取指令（P）、指令译码（T）、取操作数（D）和执行指令（E）。由于 DSP 控制器采用了多组总线的结构，这将允许 CPU 同时进行程序指令和存储数据的访问，因而在其内部可以实现四级逻辑流水线，如图 2.3 所示。在某一时刻，第一条流水线上在做取指令操作时，第二条流水线可同时进行上一条指令的指令译码的操作，第三条流水线可同时进行再上一条指令的取操作数的操作，第四条流水线可同时进行再上一条指令的执行指令的操作。由于这四种操作在同一时刻是分别使用内部的六条总线，因此不会发生冲突，就像多辆车在多条道的高速公路上行驶一样，从而实现了一种并行处理的机制。

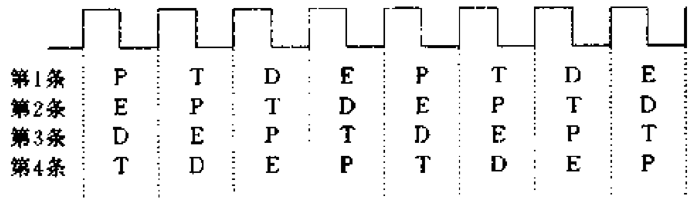


图 2.3 四级逻辑流水线

这种并行机制可使得 4 条指令同时在一个周期内处于激活状态，而在任一周期都有执行指令（E）的操作，就好像一个周期就可以完成一条指令。

四级流水线是逻辑上的，大部分情况下对用户来说也是不可见的。在下面两种情况下，四级流水线将会暂停：

- 紧跟在修改全局存储器分配寄存器（GREG）后的单字单周期指令使用先前的全局映射。
- NORM 指令修改辅助寄存器指针（ARP）而且在流水线执行阶段要使用修改后 ARP 指出的那个寄存器，如果后面两个指令字在执行 NORM 前会改变 ARP 的值或者是辅助寄存器的值，这就会使 NORM 使用错误的辅助寄存器值，并使后续指令使用错误的 ARP 值。

2.2 中央处理单元

中央处理单元（CPU）是挂接在总线上的核心模块，它的任务是从程序读数据总线或数据读数据总线上获取数据，经过加、乘、移位等算术逻辑运算，再经数据写数据总线将结果送出，如图 2.4 所示。中央处理单元分为三个部分：

●输入比例部分。

●中央算术逻辑部分。

●乘法部分。

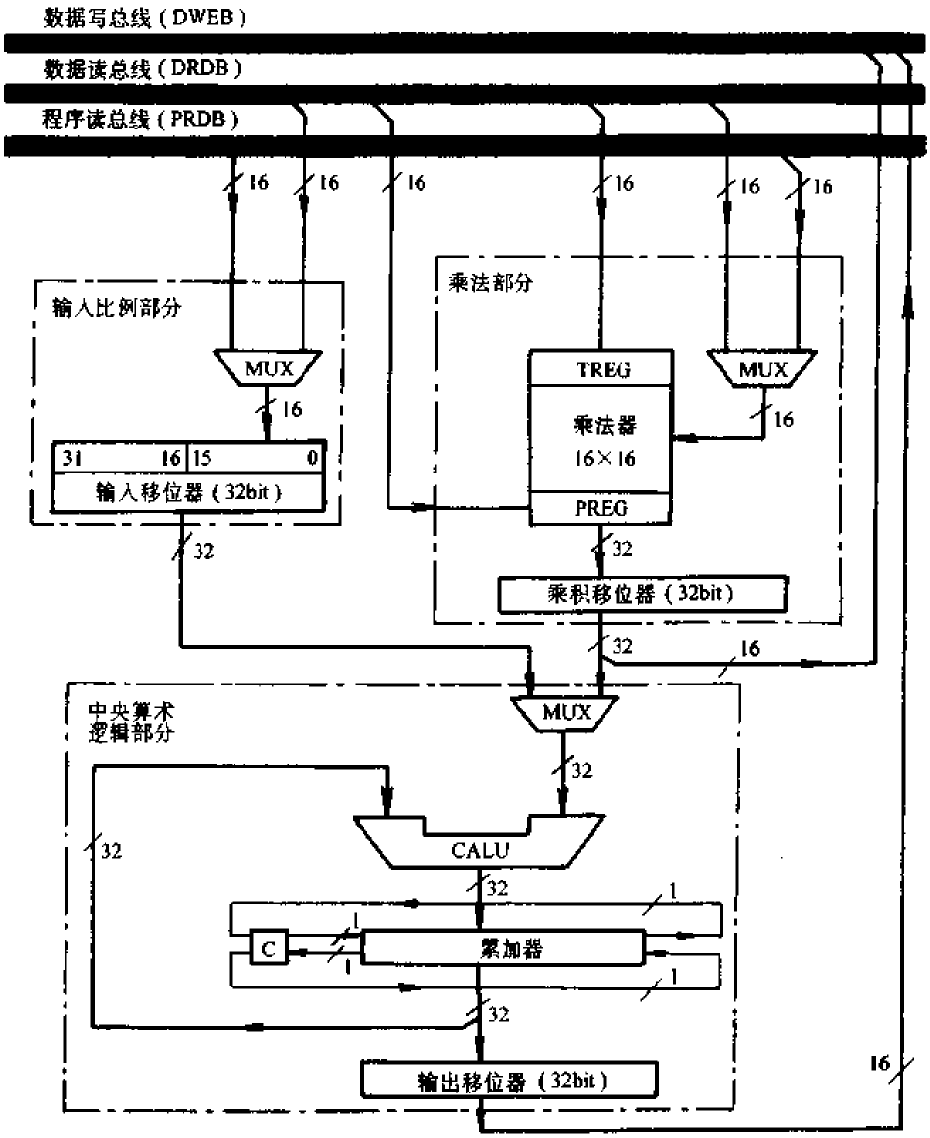


图 2.4 中央处理单元

输入比例部分是将程序读数据总线或数据读数据总线上的 16 位数据与 32 位的中央算术逻辑单元（CALU）的数据对齐；中央算术逻辑部分完成加、减、移位等算术逻辑运算；乘法部分实现 16×16 的乘法运算。下面分别加以介绍。

2.2.1 输入比例部分

由于程序读数据总线或数据读数据总线是16位，而中央算术逻辑单元是32位。为了使程序读数据总线或数据读数据总线上的数据能正确地参与到中央算术逻辑单元中运算，需要对程序读数据总线或数据读数据总线上的数据进行移位以及扩展，以便与32位中央算术逻辑单元中的数据对齐。输入移位器进行的操作不需要CPU时钟开销。

16位的程序读数据总线或数据读数据总线上的数据信号通过多路转换器接至32位的输入移位器的输入端，程序读数据总线上的数据是指令操作数给出的常数值（立即寻址），数据读数据总线上的数据是指令操作所需引用的数据存储单元中的数值（直接寻址、间接寻址），如图2.4所示。

输入移位器可将输入值左移0～16位，移位次数可由两种来源获得：

●指令字中的常数。把移位次数放在指令字中，允许用户为程序代码使用特定的数据比例（左移1次相当于乘2）。

●临时寄存器（TREG）的低4位。根据TREG的值移位，允许动态调整数据的比例系数，从而适应不同的系统性能。

另外，输入移位器也可以进行符号扩展。在微处理器中常常需要进行有符号数的运算，有符号数常以二进制补码来表示。因此，对于16位的有符号数送入到32位中央算术逻辑单元时，需要进行符号扩展（参见1.2.2）。DSP控制器中状态寄存器ST1的第10位是符号扩展模式位（SXM）。若SXM=0，则不进行符号扩展；若SXM=1，则进行符号扩展。

●当SXM=0时，输入移位器对输入值左移后，其低端中未使用的LSB填0，高端未使用的MSB填0。

●当SXM=1时，移位时需要进行符号扩展。若输入值为正数，高端未使用的MSB填0，低端中未使用的LSB填0；若输入值为负数，高端未使用的MSB填1，而低端中未使用的LSB仍填0。

2.2.2 中央算术逻辑部分

中央算术逻辑部分含有：

● 32位的中央算术逻辑单元（CALU）；

● 32位的累加器（ACC）；

● 32位输出移位器。

中央算术逻辑单元有两个输入：一个输入总是来自累加器（所有的加减法指令都隐含累加器作为一个操作数）；另一个输入来自输入移位器的输出或者乘积移位器的输出。中央算术逻辑单元可实现加减算术运算、与或等逻辑运算和位测试等功能；累加器接收中央算术逻辑单元的输出，可与进位位（C）一起进行移位操作，ACCH和ACCL分别是累加器的高位字和低位字。

输出移位器把累加器的内容拷贝过来，并可对其高位字或低位字进行移位操作，然后送到16位的数据写数据总线上，进入到数据存储器中。输出移位器可以左移0～7位，移位时高位丢失，低位补0。然后经指令SACH或SACL将移位器中的高位字或低位字保存到数据存储器，累加器的内容保持不变。

2.2.3 乘法部分

DSP 控制器含有一个 16 位×16 位的硬件乘法器，可在一个周期内完成有符号或无符号数乘法，乘积为 32 位。如图 2.4 所示，乘法部分包括：

- 16 位的临时寄存器（TREG），它含有一个乘数；
- 乘法器，它把临时寄存器的值与来自数据存储器或程序存储器的被乘数相乘；
- 32 位的乘积寄存器（PREG），它接收相乘运算的结果；
- 乘积移位器，使乘积寄存器的值在送到中央算术逻辑单元或数据存储器前进行移位定标。

乘法器接收两个 16 位的输入：一个输入（乘数）总是来自临时寄存器 TREG，在乘法之前把数据读数据总线的值加载到临时寄存器；另一输入（被乘数）来自数据存储器或程序存储器。

乘积移位器可进行四种形式的移位，由状态寄存器 ST1 中的乘积移位模式位（PM）确定，如表 2.1 所示。乘积移位器可把乘积结果送到中央算术逻辑单元或者经指令 SPH（SPL）将乘积移位器的高位字（低位字）送到数据存储器。

表 2.1 乘积移位器中乘积移位模式

PM	移位	说　明
00	不移	乘积送到 CLAU 或数据写数据总线，不移位
01	左移 1 位	移去二进制补码乘法产生的额外符号位，产生 Q_{31} 格式的乘积
10	左移 4 位	移去 16 位×13 位（常数）二进制补码乘法产生的额外 4 位符号位，产生 Q_{31} 格式的乘积
11	右移 6 位	把乘积定标，使最多作 128 次乘法累加而不使累加器溢出。不论 ST1 中 SXM 为何值，右移总是要进行符号扩展

注：Q_{31} 格式是 32 位有符号小数的标准格式，即最高位是符号位，小数点紧接其后。可参见 1.2.2。

2.3 辅助寄存器算术单元

中央处理单元（CPU）还有一个与中央算术逻辑单元（CALU）无关的辅助寄存器算术单元（ARAU）。它的主要功能是与中央算术逻辑单元中进行的操作并行地实现对 8 个辅助寄存器（AR0～AR7）的算术运算，如图 2.5 所示。

DSP 控制器的指令系统提供了丰富、灵活、有效的间接寻址方式的指令。这些间接寻址方式由 8 个辅助寄存器（AR0～AR7）来实现。当前时刻由哪个辅助寄存器进行间接寻址取决于状态寄存器 ST0 中的辅助寄存器指针（ARP），利用 MAR、LST 指令可以修改辅助寄存器指针的值。辅助寄存器指针所指出的寄存器称为当前辅助寄存器或当前 AR。在处理一条间接寻址的指令时，当前辅助寄存器的内容用作访问数据存储器的地址。如果指令需要从数据存储器读出，辅助寄存器算术单元（ARAU）就把这个地址送到数据读地址总线（DRAB）。如果指令要向数据存储器写入，则将地址送到数据写地址总线（DWAB）。

辅助寄存器算术单元（ARAU）完成以下运算：

- 将辅助寄存器的内容增、减 1，或者增、减一个变址量（取决于间接寻址的指令）。
- 使辅助寄存器的内容增、减一常数（ADRK、SBRK 指令），该常数值是指令字的低 8 位。
- 把 AR0 的内容与当前 AR 的内容进行比较（CMPR 指令），并把结果经数据写数据总线（DWDB）放入到状态寄存器 ST1 的测试/控制位（TC）。

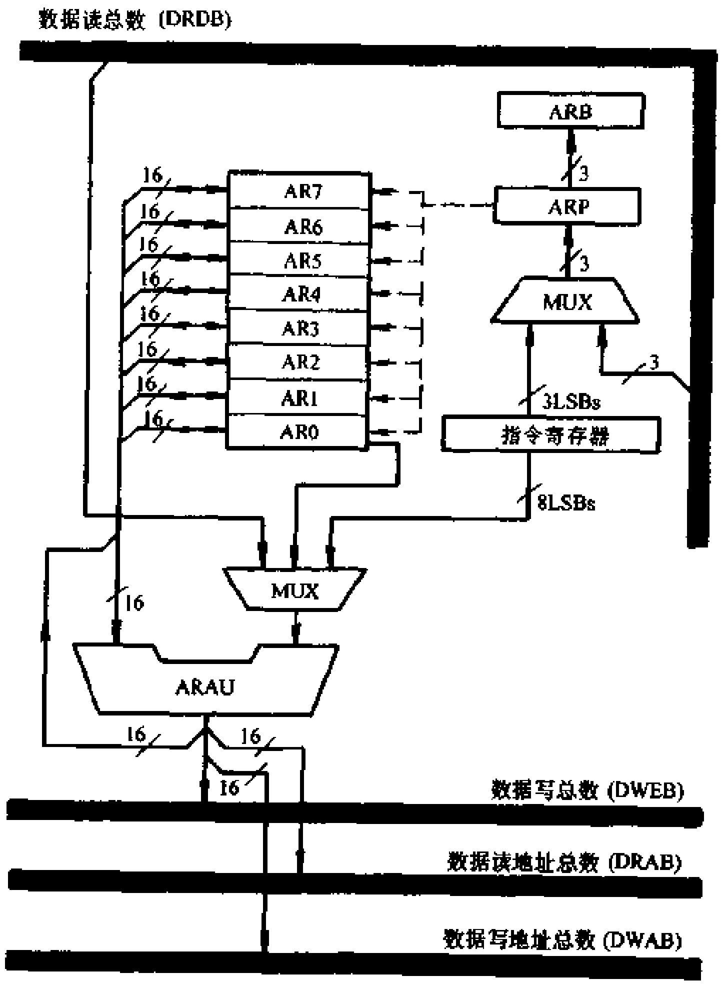


图 2.5　辅助寄存器算术单元

通常，辅助寄存器算术单元在流水线的译码阶段进行它的算术运算。这就能在下条指令译码之前产生地址。有一个例外情况，就是在处理NORM指令时，在流水线的执行阶段修改辅助寄存器或辅助寄存器指针。

除进行间接寻址外，辅助寄存器还有其它用途。例如：

●通过CMPR指令，使辅助寄存器支持条件分支、调用和返回。

●用辅助寄存器作暂存单元，利用LAR指令将数值加载到该寄存器，再用SAR指令把AR值保存到数据存储器。

●利用辅助寄存器做软件计数器，按需要对其加、减。

2.4　状态寄存器ST0和ST1

DSP控制器有两个状态寄存器——ST0和ST1，它们含有状态和控制位。状态寄存器ST0和ST1的内容可以保存到数据存储器（SST指令）也可以从数据存储器中加载（LST指令）。因此，可以保存和恢复子程序的机器状态。

状态寄存器ST0和ST1中有很多位可用SETC和CLRC指令单独置1和清0。例如，利用SETC SXM指令将符号扩展模式位置1，而CLRC SXM指令则将其清0。表2.2示出了状态寄存器ST0和ST1各位的定义。其中有些位是保留的，读出时为逻辑1。表2.3是各位的

详细说明。

表 2.2 状态寄存器 ST0 和 ST1

<table>
<tr><td>寄存器</td><td colspan="8">位 数</td></tr>
<tr><td rowspan="3">ST0</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8-0</td></tr>
<tr><td colspan="3">ARP</td><td>OV</td><td>OVM</td><td rowspan="2">1</td><td>INTM</td><td>DP</td></tr>
<tr><td colspan="3">RW-x</td><td>RW-0</td><td>RW-x</td><td>RW-1</td><td>RW-x</td></tr>
<tr><td rowspan="6">ST1</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td></tr>
<tr><td colspan="3">ARB</td><td>CNF</td><td>TC</td><td>SXM</td><td>C</td><td rowspan="2">1</td></tr>
<tr><td colspan="3">RW-x</td><td>RW-0</td><td>RW-x</td><td>RW-1</td><td>RW-1</td></tr>
<tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td rowspan="2">1</td><td rowspan="2">1</td><td rowspan="2">1</td><td>XF</td><td rowspan="2">1</td><td rowspan="2">1</td><td colspan="2">PM</td></tr>
<tr><td>RW-1</td><td colspan="2">RW-0</td></tr>
</table>

注：R——可读；W——可写；x——复位时为任意值；0——复位时为 0；1——复位时为 1。（以下同）

表 2.3 状态寄存器 ST0 和 ST1 各位的定义与说明

名称	说明
ARP	辅助寄存器指针。该指针用于选择间接寻址中使用的辅助寄存器 AR。可用 LST、MAR 指令修改它的值
ARB	辅助寄存器指针缓存器。除 LST 指令外，每当加载 ARP 时，ARP 原来的值就拷贝到 ARB 中。当用 LST 加载 ARB 时，同样的值也拷贝到 ARP 中
C	进位位。如果相加结果产生进位，该位置 1，否则清 0；如果减法结果产生借位，该位清 0，否则置 1。但是移位 16 位的 ADD 和 SUB 指令除外，这时 ADD 指令只在进位时使 C 置 1，SUB 只在借位时使 C 清 0，否则不影响 C。复位时 C 为 1
CNF	片内 DARAM 配置位。该位决定可重新配置的双口 RAM 块映射到数据空间还是程序空间。CNF 为 0 则配置到数据空间，CNF 为 1 配置到程序空间。SETC CNF、CLRC CNF、LSF 指令可以修改 CNF。复位时 CNF 为 0
DP	数据页面指针
INTM	中断总屏蔽位。INTM 为 0 允许没有屏蔽的中断使能，INTM 为 1 则禁止所有可屏蔽中断。LST 指令不影响 INTM。该位对非屏蔽中断 RS、NMI 和软件中断不起作用
OV	溢出标志。进行补码运算时，数据超出范围将发生溢出，该位置 1。复位或 LST、(无)溢出条件分支指令将使该位清 0
OVM	溢出模式位。OVM 为 0，不对溢出结果进行调整；OVM 为 1，对溢出结果进行调整，正溢出时累加器的结果调整为 7FFF FFFFH，负溢出时累加器的结果调整为 8000 0000H
PM	乘积移位模式。PM 决定乘积结果在送出前怎样进行移位 PM=00 不移位 PM=01 左移 1 位，低位填 0 PM=10 左移 4 位，低位填 0 PM=11 右移 6 位，进行符号扩展 复位时 PM 清 0。SPM、LST 指令可以修改 PM
SXM	符号扩展模式位。SXM 为 0 不进行符号扩展；SXM 为 1 进行符号扩展
TC	测试/控制标志位。如果 BIT 或 BITT 测试结果为 1、CMPB 测试当前 AR 和 AR0 的比较条件成立、NOMR 测试累加器最高 2 位异或结果为 1，将使 TC 置 1
XF	XF 引脚状态位。复位时该位为 1。SET XF、CLR XF、LST 指令都可以修改 XF

2.5 存储器与I/O空间

存储器结构有两大类：冯诺曼结构（Von Neumann）和哈佛结构（Harvard）。前者将程序与数据合用一个存储空间，通过地址分段来存放程序与数据，像Intel 8086系列。后者将程序存储空间与数据存储空间分离开来，二者是不同的物理存储器，可以拥有相同的地址，通过不同的控制线（读、写、片选等信号）来对它们进行访问，像Intel 8051系列。一般情况下，控制系统需要的程序存储容量较大而数据存储容量较小。这样一来，采用哈佛结构就可以单独将小容量的数据存储器以高速的RAM形式实现并集成到芯片内，以加快数据处理的速度。目前，大部分单片机和DSP控制器都采用哈佛结构。

I/O端口与存储器一样，都可看作为数据源，从逻辑上讲二者没有本质的差异。有的微处理器芯片，其存储空间与I/O空间是相互分离的，可以拥有相同的地址，它们的访问通过控制线来区分（对应不同类型的指令），像Intel 8086系列；有的微处理器芯片，其存储空间与I/O空间是同一个地址空间，也就是把I/O空间映射到存储空间，二者通过地址来区分，这样一来对存储器和I/O端口的访问使用同类型的指令，像Intel 8051系列。

DSP控制器采用独立的程序存储器、数据存储器和I/O空间，即它们可以有相同的地址，而它们的访问通过控制线来区分。除此之外，数据存储器还分为局部数据存储器和全局数据存储器，这二者也可共用相同的地址空间，它们的访问除了通过不同的控制线来区分以外，还受全局存储器分配寄存器（GREG）的控制。

DSP控制器使用16位的地址总线，可访问的四种独立的选择空间是（共224K字）：

● 64K字程序存储器空间，包含要执行的指令及程序执行时使用的数据。

● 64K字局部数据存储器空间，保存指令使用的数据。

● 32K字全局数据存储器空间，保存与其它处理器共用的数据，或者作为一个附加的数据空间。局部数据空间中的高32K字地址（8000h～FFFFh）可用作全局数据空间。局部数据存储器与全局数据存储器可以是两个不同的物理存储器，通过修改全局存储器分配寄存器（GREG）的内容来控制当前使用何种数据存储器。但全局数据寄存器的最大空间是32K字（8000h～FFFFh）。

● 64K字的I/O空间，用于外设接口，包括一些片内外设的寄存器。

2.5.1 与外部存储器和I/O空间接口的信号

由于DSP控制器四种独立选择空间都占用相同的16位的地址空间，因此对它们的访问必须通过控制线来区分。表2.4描述了与外部存储器和I/O空间接口的信号，这些信号主要有四类：

●外部数据/地址总线，D0～D15是数据总线，A0～A15是地址总线。

●片选信号线，外部器件利用这些信号来区分是内部访问还是外部访问？是访问程序存储空间、局部数据空间、全局数据空间还是I/O空间？

●读/写信号，这些信号向外部器件指出数据传送的方向。

●请求/控制信号，通过这些信号可实现一些特殊操作，例如，存储器直接访问方式的实现。

表 2.4 与外部存储器和 I/O 空间接口的信号

名 称	引 脚	说 明
外部总线	A15-A0	16 条外部地址总线，可寻址 64K 字的外部存储器或 I/O 空间
	D15-D0	16 条双向外部数据总线
选择信号	$\overline{DS}$	数据存储器选择引脚。当执行访问外部(局部或全局)数据存储器指令时该引脚有效
	$\overline{BR}$	总线请求引脚。当执行访问外部全局数据存储器指令时该引脚有效
	$\overline{PS}$	程序存储器选择引脚。当从外部程序存储器取指令时该引脚有效
	$\overline{IS}$	I/O 空间选择引脚，当执行 I/O 指令时该引脚有效
	$\overline{STRB}$	外部访问有效选通引脚。当访问外部程序、数据或 I/O 时该引脚有效
读/写信号	R/$\overline{W}$	读/写引脚
	$\overline{R}$/W	R/$\overline{W}$ 引脚的逆
	$\overline{WE}$	写允许引脚。发给外部程序、数据或 I/O，请求允许对它们进行写操作
请求/控制信号	MP/$\overline{MC}$	微处理器/微计算机模式引脚。该引脚为高电平，器件处于微处理器模式，从外部程序存储器取复位向量；该引脚为低电平，器件处于微计算机模式，从片内程序存储器取复位向量
	READY	外围设备就绪引脚。当该引脚被驱动为低时，CPU 处在等待状态，直到该引脚为高

2.5.2 程序存储器

程序存储器的配置如图 2.6 所示。DSP 控制器可以使用片内程序存储器，也可以使用片外程序存储器，由引脚 MP/$\overline{MC}$决定。当 MP/$\overline{MC}$=0 时，使用片内程序存储器；当 MP/$\overline{MC}$=1

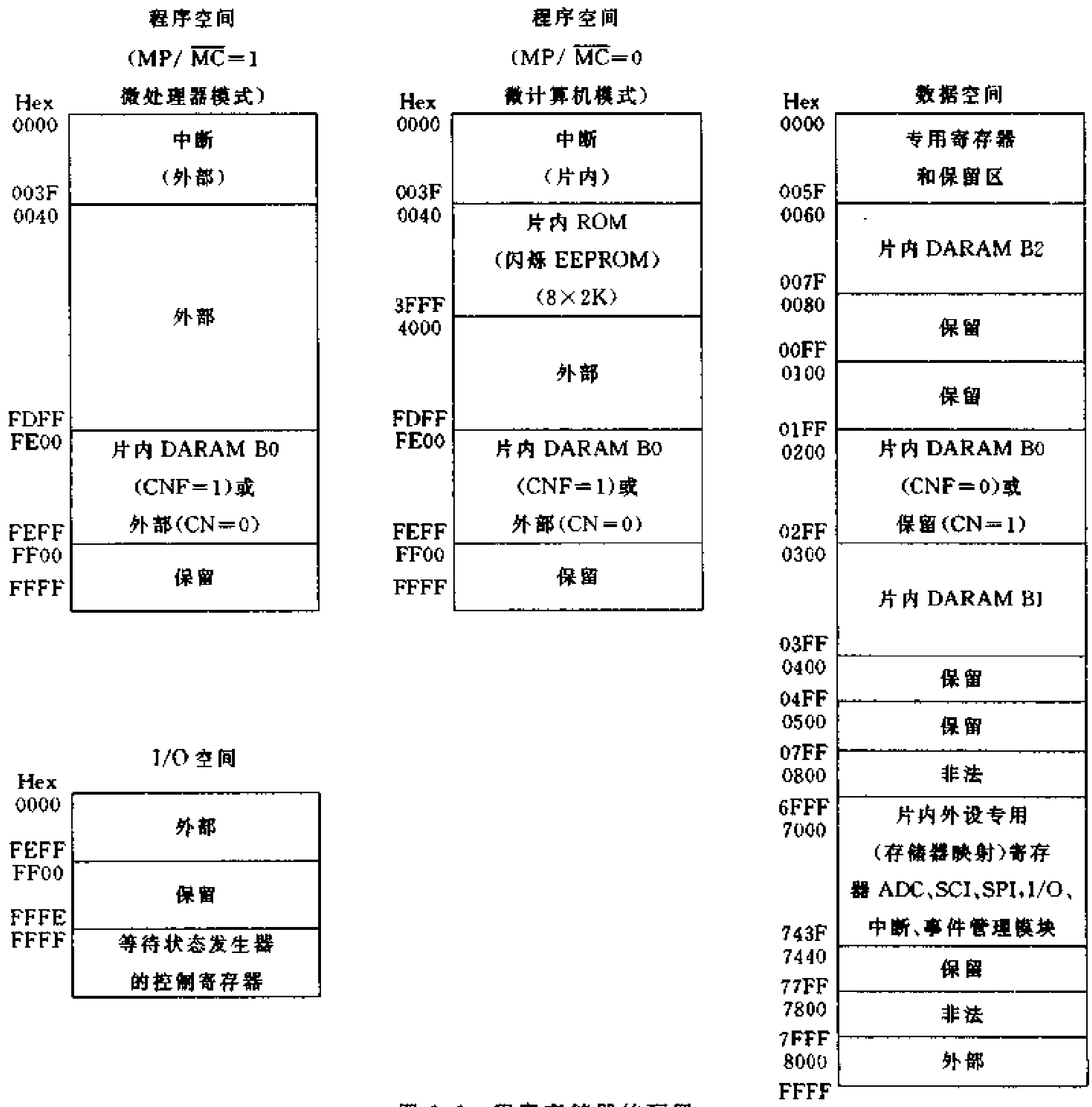


图 2.6 程序存储器的配置

时，使用片外程序存储器。一般情况下，片内程序存储器的访问速度比片外程序存储器速度快，而且比片外程序存储器功耗低。采用片外程序存储器操作的优点是可访问更大的地址空间（64K 字）。

程序存储器的地址分配：

● 0000h～003Fh：用于存储中断入口地址。当有中断请求信号时，CPU 从这个地方取中断服务子程序的入口。0000h 是系统复位向量地址，任何程序都得从此开始运行，所以一般在此安排一条分支跳转指令，让 CPU 转入到用户主程序的入口处。此块地址与中断服务有关，因此这个地方最好不要安排其它的程序指令。

● 0040h～FDFFh：用户程序区。根据不同的型号，可以有 4/8/16/32K 字的片内 FLASH/ROM；0/1/2/4/8/16K 字的单口存储器 SARAM；其余地址空间要使用的话，需要外扩。

● FE00h～FFFFh：这是一个双口存储器 DARAM 区（B0），可以配置给程序存储器，也可以配置给数据存储器，由状态寄存器 ST1 的 CNF 位决定。CNF＝0，配置给数据存储器；CNF＝1，配置给程序存储器。复位时 CNF＝0。

如果片内程序存储器不够用或不能用（片内 ROM 需要厂家写入），就需要外扩程序存储器。图 2.7 是 DSP 控制器与外部程序存储器接口的一个例子。图中利用两片 8K×8 位 EEPROM 构成 8K×16 位的程序存储器。外部程序存储器的数据线、地址线与 DSP 控制器相应的数据线、地址线相连。由于 DSP 控制器访问片内程序存储器时，信号$\overline{PS}$和$\overline{STRB}$处于高阻状态；DSP 控制器访问外部程序存储器时，外部总线被激活，信号$\overline{PS}$和$\overline{STRB}$有效，此时表明外部总线正用于程序存储器。因此，可以用信号$\overline{PS}$作为外部程序存储器的片选信号。如果还需扩展多片外部程序存储器，可以用高位地址线经译码后与信号$\overline{PS}$一起构成外部程序存储器的片选信号。DSP 控制器的处理速度非常快，在外扩程序存储器时必需考虑其响应时间是否与 DSP 控制器匹配。若使用较慢的存储器，则需插入等待状态以匹配二者的速度。插入等待状态有两种方法，一是利用片内等待状态发生器在访问周期中自动插入一个等待状态；二是在外部产生等待逻辑，并将其连接到 DSP 控制器的引脚 READY 上。当信号 READY 为低时，DSP 控制器处于等待状态；当信号 READY 为高时，DSP 控制器处于工作状态。采用第二种方法可以产生一个以上的等待状态，其等待的时间由外部等待逻辑发生器控制。

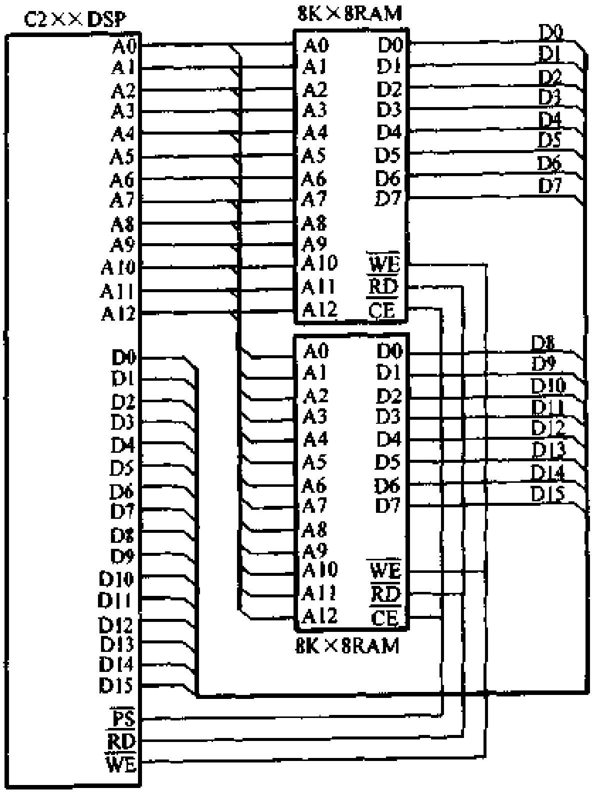


图 2.7　外扩程序存储器

2.5.3　局部数据存储器

局部数据存储器分为片内数据存储器和片外数据存储器，其配置如图 2.8 所示。其中：

● 0000h～005Fh：专用寄存器区。这个区间的地址由 DSP 控制器使用，安排了中断屏蔽寄存器、全局存储器分配寄存器、中断标志寄存器等。

● 0060h～007Fh：32 个字的双口存储器 DARAM (B2)，用户数据区。

● 0080h～00FFh：保留。

● 0100h～02FFh：256/512 个字的双口存储器 DARAM (B0)，只当 CNF=0 才可作为用户数据区。

● 0300h～04FFh：256/512 个字的双口存储器 DARAM (B1)，用户数据区。

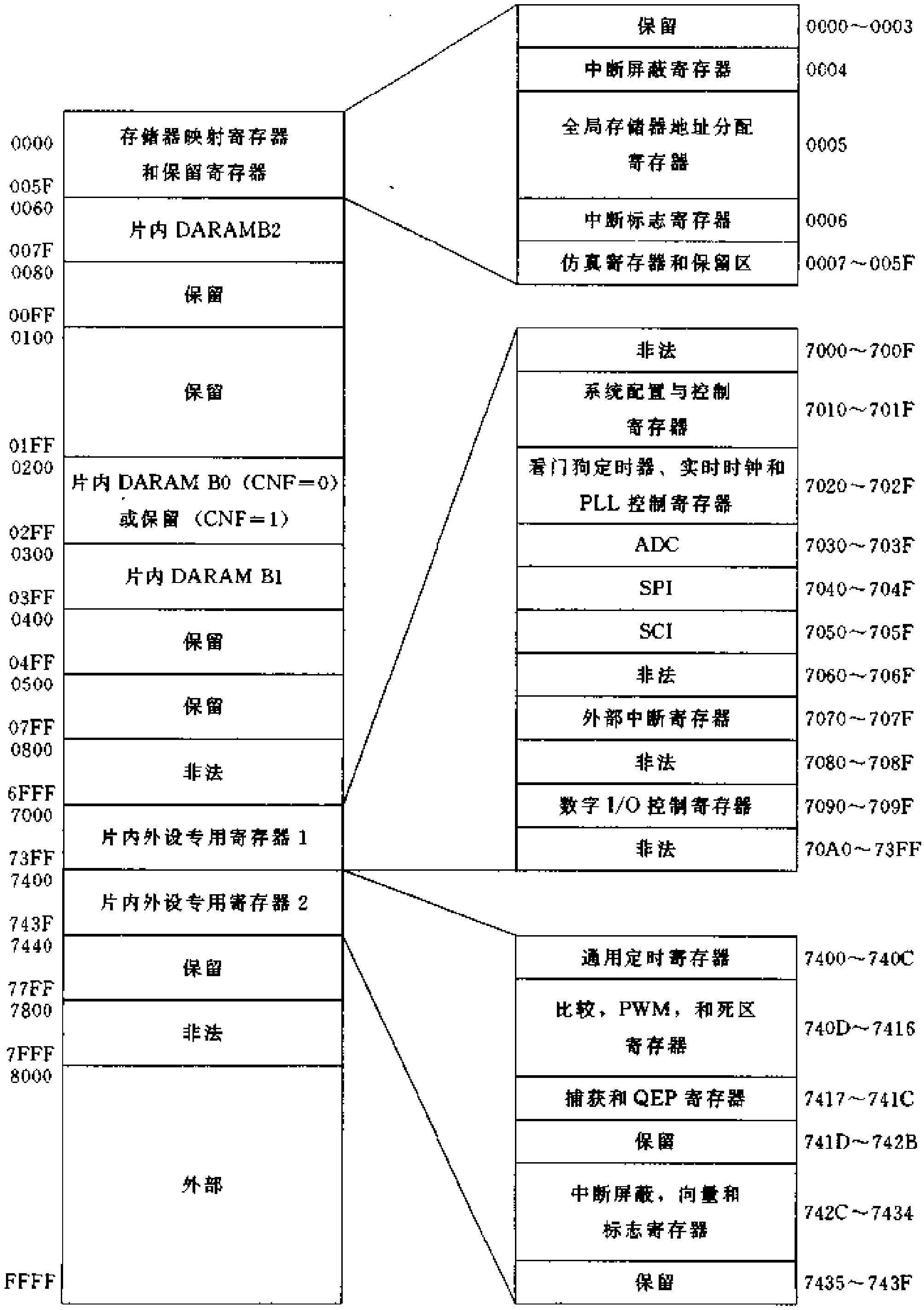


图 2.8　局部数据存储器配置

- 0500h～07FFh：保留。
- 0800h～6FFFh：1/2/4/8/16K 字的单口存储器 SARAM，用户数据区。
- 7000h～743Fh：片内外设控制、数据等寄存器，是给 DSP 控制器用的专用寄存器区，可参见附表。
- 7440h～7FFFh：保留。
- 8000h～FFFFh：给片外局部数据存储器使用。

局部数据存储器的寻址范围是 64K 字。DSP 控制器的指令系统对数据存储器的寻址可以 16 位的物理地址进行访问（例如，间接寻址方式），也可以按页进行访问（例如，直接寻址方式）。

64K 字的局部数据空间分成 512 个数据页（占用 9 位高地址位），每个数据页有 128 个字（占用 7 位低地址位），如表 2.5 所示。状态寄存器 ST0 中 9 位数据页面指针 DP 的值确定当前使用哪个数据页。当前数据页中的每一个字则由 7 位偏移量来指定（含在指令中）。因此，在采用直接寻址方式访问数据存储器时，不但要指定数据页面（确定 DP 的值），还要指定偏移量（由指令确定）。为了加快数据的访问速度，最好把同类的数据放在同一数据页中。

表 2.5　局部数据存储器的地址分配与数据页

DP 值	偏移量	数据存储器
0000 0000 0 ⋮ 0000 0000 0	000 0000 ⋮ 111 1111	页 0：0000h～007Fh
0000 0000 1 ⋮ 0000 0000 1	000 0000 ⋮ 111 1111	页 1：0080h～00FFh
0000 0001 0 ⋮ 0000 0001 0	000 0000 ⋮ 111 1111	页 2：0100h～017Fh
⋮	⋮	⋮
1111 1111 1 ⋮ 1111 1111 1	000 0000 ⋮ 111 1111	页 511：FF80h～FFFFh

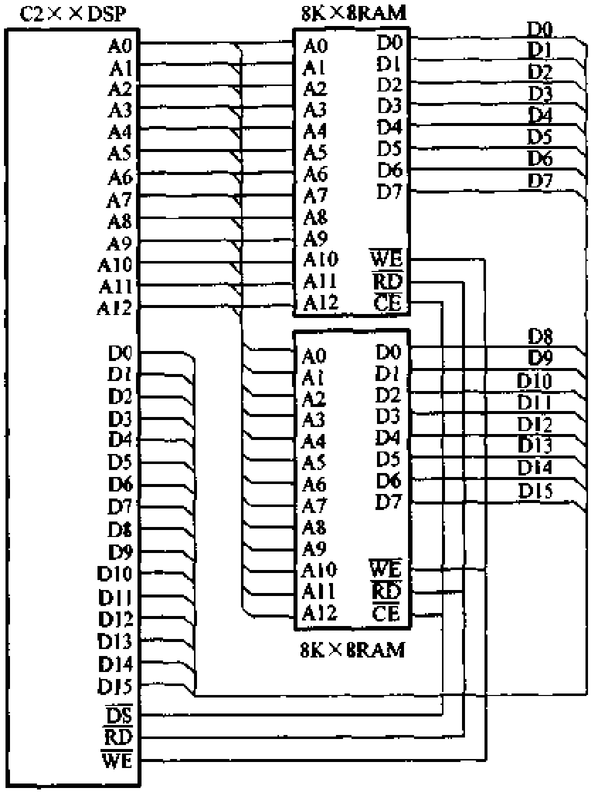


图 2.9　外扩局部数据存储器

如果供用户使用的片内数据存储器 B0、B1 和 B2 不够用，则需外扩数据存储器。DSP 控制器在访问片内数据存储器时，信号$\overline{DS}$和$\overline{STRB}$处于高阻状态。仅当 DSP 控制器访问映射至外部存储器的地址范围（8000h～FFFFh）时，信号$\overline{DS}$和$\overline{STRB}$处于有效状态，表明数据存储器在使用外部总线。因此，可以用信号$\overline{DS}$作为外部局部数据存储器的片选信号。图 2.9 是外扩局部数据存储器的例子，图中用两片 16K×8 位的 RAM 构成 16K×16 位字的 RAM。若还

有多片外部数据存储器要连，可以用高位地址线经译码后与信号$\overline{DS}$一起构成外部局部数据存储器的片选信号。

为加快接口速度，可选用快速外部存储器。若不要求快速访问存储器，可用READY信号或片内等待状态发生器来建立等待状态以便和低速外部存储器接口，这和使用慢速程序存储器时采用的方法类似。

2.5.4 全局数据存储器

DSP控制器除了局部数据存储器外，还有一个全局数据存储器，用于保存与其它处理器共用的数据，或者作为一个附加的数据空间。全局数据存储器的最大寻址空间是32K字(8000h～FFFFh)，它与局部数据存储器可以是同一个物理存储器，也可以是两个不同的物理存储器。

如果是同一个物理存储器，二者通过地址就可以区分，其地址范围由全局存储器分配寄存器（GREG）来指定，如表2.6所示。全局存储器分配寄存器（GREG）确定全局数据存储器空间的大小，范围在256到32K字之间。全局存储器分配寄存器的地址是0005h，只使用低8位。切记只能选择表2.6所列出的GREG值，其它值会使存储器映射分成碎片。

表2.6 全局存储器分配寄存器(GREG)

GREG 的 值	局 部 存 储 器	全 局 存 储 器
×××× ×××× 0000 0000	0000h～FFFFh 65356	… 0
×××× ×××× 1000 0000	0000h～7FFFh 32768	8000h～FFFFh 32768
×××× ×××× 1100 0000	0000h～BFFFh 49152	C000h～FFFFh 16384
×××× ×××× 1110 0000	0000h～DFFFh 57344	E000h～FFFFh 8192
×××× ×××× 1111 0000	0000h～EFFFh 61440	F000h～FFFFh 4096
×××× ×××× 1111 1000	0000h～F7FFh 63488	F800h～FFFFh 2048
×××× ×××× 1111 1100	0000h～FBFFh 64512	FC00h～FFFFh 1024
×××× ×××× 1111 1110	0000h～FDFFh 65024	FE00h～FFFFh 512
×××× ×××× 1111 1111	0000h～FEFFh 65280	FF00h～FFFFh 256

如果不是同一个物理存储器，二者的地址空间将会有重叠的部分，特别是当局部数据存储空间需要64K字，全局数据存储空间需要32K字时，二者的高32K字地址空间（8000h～FFFFh）将完全重叠。这个时候，要区分局部数据空间和全局数据空间的访问，光靠全局存

储器分配寄存器（GREG）来指定地址范围就不行了，需要加上总线请求信号$\overline{\text{BR}}$，如图2.10所示。当DSP控制器访问全局数据空间时，信号$\overline{\text{BR}}$将被置为有效，因此可用它作为全局数据存储器的片选信号。

在局部数据存储器和全局数据存储器是两个不同的物理存储器的情况下进行数据访问时，要根据当前是访问局部数据空间还是全局数据空间，进行修改全局存储器分配寄存器（GREG）的值。当进行全局数据空间访问时，要先将全局存储器分配寄存器（GREG）的值修改到与实际的全局数据存储器的物理地址范围一致；当进行局部数据空间访问时，要先将全局存储器分配寄存器（GREG）的值清零，此时总线请求信号$\overline{\text{BR}}$将失效，而数据存储器选择信号$\overline{\text{DS}}$会有效，从而保证当前访问的是局部数据空间。

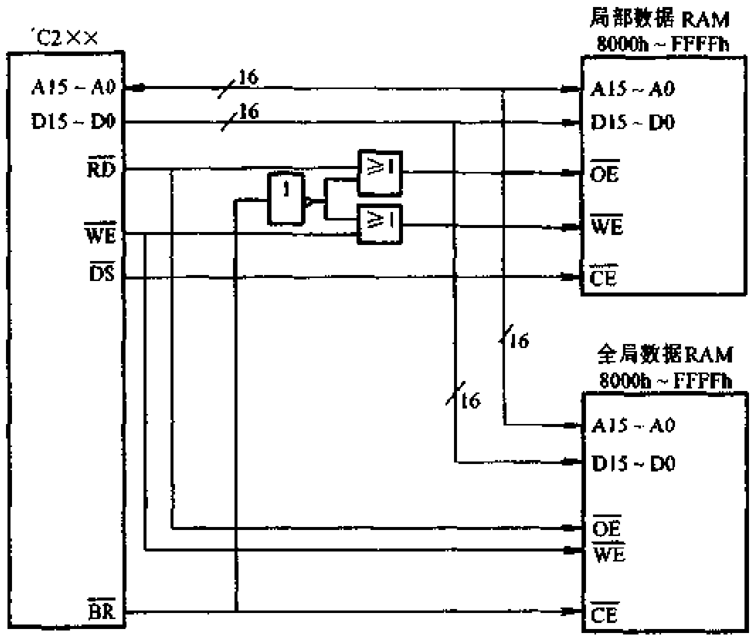


图2.10 外扩局部和全局数据存储器

2.5.5 I/O空间

DSP控制的I/O空间可寻址64K字，由三部分组成：

- 0000h～FEFFh 用于访问片外外设。
- FF00h～FFFEh 保留。
- FFFFh，映射为等待状态发生器的控制寄存器。

所有I/O空间（外部I/O端口和片内I/O寄存器）都可用IN和OUT指令访问。当执行IN或OUT指令时，信号$\overline{\text{IS}}$将变成有效，因此可用信号$\overline{\text{IS}}$作为外围I/O设备的片选信号。访问外部并行I/O端口与访问程序、数据存储器复用相同的地址和数据总线。数据总线宽度为16位，若使用8位的外设，既可使用高8位数据总线也可使用低8位数据总线，以适应特定应用的需要。图2.11是一个16位I/O端口接口电路。需指出，若所用I/O端口很少，那么译码部分可以简化。

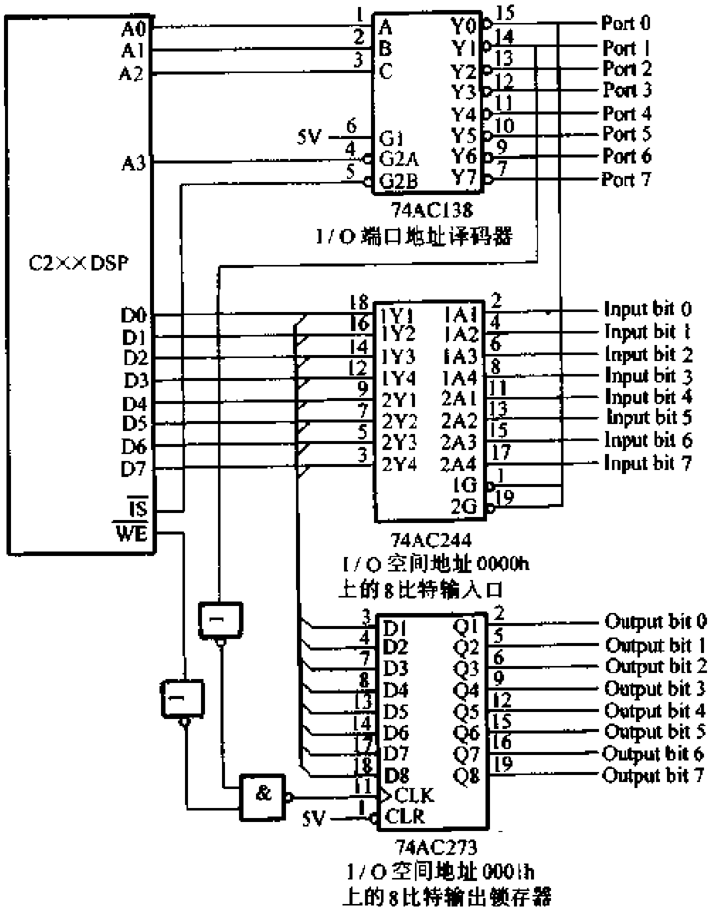


图 2.11　16 位 I/O 端口接口电路

2.6　程序控制

通常程序的流程是顺序的，在程序地址指针的指挥下按节拍进行工作。DSP 控制器的程序地址的产生如图 2.12 所示。

16 位的程序计数器 PC 是程序地址产生的核心部分，PC 也称为程序地址指针。系统复位时由内部硬件逻辑将 PC 置为 0000h（复位中断向量）。PC 的内容经程序地址寄存器（PAR）驱动程序地址总线（PAB），使得中央处理单元 CPU 获得当前的指令。当前指令被装入指令寄存器后，PC 的内容加 1，为下一个地址作准备。PC 的内容决定了 CPU 下次取指的地点。

程序的流程一般是顺序的，但也存在跳变。引起程序跳变有下面一些情况：

- 分支跳转指令。
- 子程序调用。
- 软、硬件中断。
- 块传送或表传送。

如果遇到分支跳转指令，硬件逻辑会把指令中的跳转地址（立即数或累加器的低 16 位）加载到 PC，从而保证分支跳转到指定的地址上。

如果遇到的是子程序调用指令，不但需要考虑 PC 转移到子程序的入口上，而且还要考虑

到调用完后 PC 的返回。因此，首先要将返回（断点）地址，即当前调用的下一条指令地址保护起来，称为现场保护。程序地址产生模块中的 8 级硬件堆栈就是为此而设，这个堆栈采用先进后出、后进先出的策略。当 CPU 执行子程序调用指令时，首先自动地将返回地址压入堆栈；然后再将子程序的入口地址（含在调用指令中的长立即数或累加器的低 16 位）加载到 PC，CPU 转而运行子程序的指令；当碰到返回指令 RET，CPU 自动地将当前栈顶内容（返回地址）弹到 PC，从而恢复原来的断点程序继续运行。由于硬件堆栈有 8 级，而且采用先进后出、后进先出的策略，所以可以实现子程序嵌套，但最多 8 级。

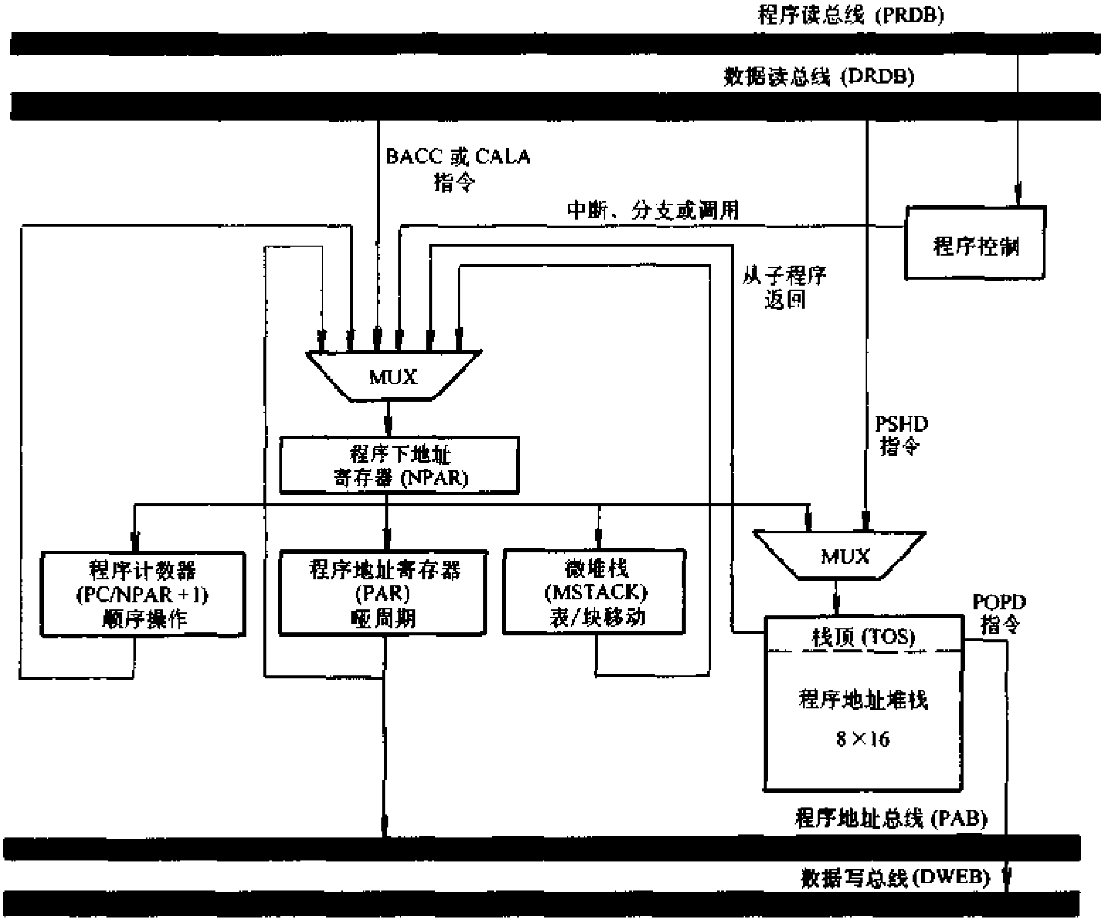


图 2.12 DSP 控制器的程序地址的产生

如果是发生软、硬件中断，与子程序调用完全类似，同样需要考虑 PC 的转移与返回。在中断情况下，需要把当前中断处的指令执行完，并把返回地址（下一条指令的地址）压入堆栈；由内部硬件逻辑把中断服务子程序的入口中断向量（这个中断向量是 DSP 控制器规定的）加载到 PC 中；当碰到中断服务子程序的返回指令 RET 时，CPU 自动地将当前栈顶内容（返回地址）弹到 PC，从而恢复原来的断点程序继续运行。中断和子程序都可以嵌套，但一起不能超过 8 级。

如果进行块传送或表传送，即执行 BLDD、BLPD、MAC、MACD、TBLR 和 TBLW 指令，由于需要利用 PC 使源（目的）操作地址增 1，因此与调用子程序指令一样在执行指令之前要保护返回地址。由于块传送或表传送是单独的指令，不可能含 RET 指令在其中。因此，块传送或表传送的返回地址不能用 8 级硬件堆栈来保护。为了解决这个问题，在程序地址产

生模块中设计了一个 16 位宽、1 级深的微堆栈（MSTACK），用这个微堆栈来保护这个返回地址（块传送或表传送指令的下一条指令的地址）。当块传送或表传送指令执行完后，硬件逻辑自动地将微堆栈的内容弹到 PC 中。与硬件堆栈不同，不能用 PUSH、POP 等指令对微堆栈进行操作，只有程序地址产生逻辑能够使用微堆栈，而且微堆栈的操作是不可见的。

2.7 时钟源模块

DSP 控制器的时钟源模块采用锁相环（PLL）技术，可以对外部振荡频率进行倍频，得到非常稳定的内部时钟。图 2.13 是时钟源模块的结构示意图。

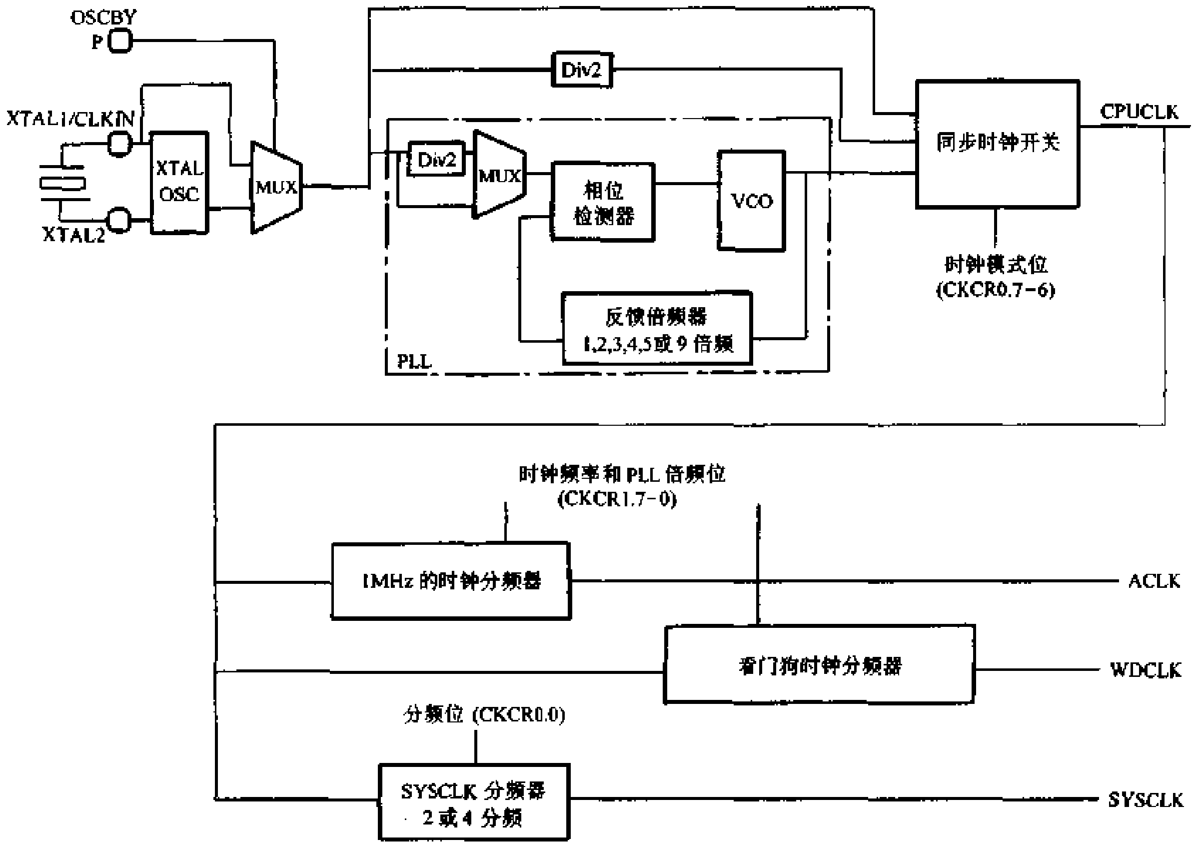


图 2.13　时钟源模块的结构示意图

与时钟源模块相关的外部引脚：

●$\overline{\text{OSCBYP}}$：振荡器的旁路引脚。如果要使用 DSP 控制器的内部振荡器，必须外接晶振，此时该引脚接高电平；如果直接使用外部振荡器，就要将内部振荡器旁路，此时该引脚接低电平。外接晶振和外部振荡器统称为外部时钟。

●XTAL1/CLKIN：当使用内部振荡器时，该引脚接外部晶振；当使用外部振荡器时，该引脚接外部振荡器的输出。外部晶振的频率可以是 2、4、6、8MHz，外部振荡器的输出的频率可在 2～32MHz 之间。

●XTAL2：接外部晶振的另一端。如果采用外部振荡器，该引脚不使用。

在时钟源模块内产生四种时钟：

●CPUCLK：CPU 时钟，供 CPU 及总线使用，也称为机器时钟，它的周期称为 CPU（机器）周期。该时钟可以是外部时钟的 1、1.5、2、2.5、3、4、4.5、5、9 倍。

● SYSCLK：它是 CPU 时钟的 2 分频或 4 分频，为 DSP 控制器的片内外设服务。

● ACLK：这个时钟专为 DSP 控制器中的模拟模块服务，无论外部时钟频率怎样，它的正常频率都是（1+10%）MHz。

● WDCLK：为看门狗和实时时钟模块提供时钟，与 ACLK 一样，无论外部时钟频率怎样，它的正常频率约为 16384Hz，其占空比为 25%。

DSP 控制器的时钟源模块是可编程的，与它相关的两个寄存器是 CKCR0 和 CKCR1，它们各位的组成如表 2.7 所示，每位的定义及说明见表 2.8 和表 2.9。

表 2.7　时钟控制寄存器 CKCR0 和 CKCR1

地址	寄存器	位数							
702Bh	CKCR0	7	6	5	4	3	2	1	0
		CLKMD1	CLKMD0	PLLOCK1	PLLOCK0	PLLPM1	PLLPM0	ACLKENA	PLLPS
		RW-x	RW-x	R-x	R-x	RW-0	RW-0	RW-x	RW-0
702Dh	CKCR1	7	6	5	4	3	2	1	0
		CKINF3	CKINF2	CKINF1	CKINF0	PLLDIV	PLLFB2	PLLFB1	PLLFB0
		RW-x	RW-x	RW-x	RW-x	RW-x	RW-x	RW-x	RW-x

表 2.8　时钟控制寄存器 CKCR0 各位定义与说明

位	定义		说明	
Bits7-6	CLKMD1	CLKMD0	CPU 时钟(CPUCLK)来源方式	
	0	0	CLKIN/2，外部频率的 2 分频	
	0	1	CLKIN，直接使用外部频率	
	1	0	PLL，经过 PLL 电路(可倍频)	
	1	1	PLL，经过 PLL 电路(可倍频)	
Bits5	PLLOCK1	0	只读，使用 PLL 电路	PLL 电路的使用状态
		1	只读，未使用 PLL 电路	
Bits4	PLLOCK0	0	只读，用于系统测试	
		1		
Bits3-2	PLLPM1	PLLPM0	执行空闲指令后，4 种时钟的状况	
	0	0	CPUCLK 停	
	0	1	CPUCLK、SYSCLK、ACLK 停	
	1	0	CPUCLK、SYSCLK、ACLK 停，PLL 停	
	1	1	全停	
Bits1	ACLKENA	0	停止 ACLK	
		1	使能 ACLK	
Bits0	PLLPS	0	SYSCLK 取 CPUCLK 的 4 分频	
		1	SYSCLK 取 CPUCLK 的 2 分频	

表 2.9　时钟控制寄存器 CKCR1 各位定义与说明

位	定　义				说　明
Bits7-4	CKINF3	CKINF2	CKINF1	CKINF0	外部频率的选择 单位：MHz
	0	0	0	0	32
	0	0	0	1	30
	0	0	1	0	28
	0	0	1	1	26
	0	1	0	0	24
	0	1	0	1	22
	0	1	1	0	20
	0	1	1	1	18
	1	0	0	0	16
	1	0	0	1	14
	1	0	1	0	12
	1	0	1	1	10
	1	1	0	0	8
	1	1	0	1	6
	1	1	1	0	4
	1	1	1	1	2
Bits3	PLLDIV		0		PLL 输入不经 2 分频
			1		PLL 输入经过 2 分频
Bits2-0	PLLFB2	PLLFB1	PLLFB0		PLL 倍频率
	0	0	0		1
	0	0	1		2
	0	1	0		3
	0	1	1		4
	1	0	0		5
	1	0	1		9
	1	1	0		1
	1	1	1		1

CPU 时钟是 DSP 控制器最重要的时钟源。它可以直接取自外部时钟或它的 2 分频，也可以由 PLL 电路对外部时钟进行倍频得到，这一切由时钟控制寄存器 CKCR0 的 CLKMD1：0 决定。采用 PLL 电路进行倍频时，其倍频系数由时钟控制寄存器 CKCR1 的 PLLFB2：0 设置。考虑到 PLL 电路的输入可先进行 2 分频（取决时钟控制寄存器 CKCR0 的 PLLDIV 位），因此 CPU 时钟可以是外部时钟的 1、1.5、2、2.5、3、4、4.5、5、9 倍。

设外部晶振或外部时钟的频率为 F_X，CPU 时钟（机器时钟）的频率为 F_C。它们之间的

关系为：

$$F_C=\begin{cases}F_x/2 & 若\ CLKMD1:0=00\\ F_x & 若\ CLKMD1:0=01\\ F_x\times(\text{PLL 倍频率})/2^{PLLDIV} & 若\ CLKMD1:0=10\ 或\ 11\end{cases}\tag{2.1}$$

SYSCLK 时钟是为挂接在片内外设总线上各功能模块服务的，它是 CPU 时钟的 2 分频或 4 分频，由时钟控制寄存器 CKCR0 的 PLLPS 决定。

ACLK 时钟专为片内模拟外设服务。当不需要使用该时钟时，可通过时钟控制寄存器 CKCR0 的 ACLKENV 位将其关闭，以减少能量消耗和电磁辐射（1MHz）。DSP 控制器加电复位时，该时钟处于关闭状态。ACLK 时钟频率在 1MHz 左右，不随外部时钟频率变化，但外部时钟频率的选择必须符合时钟控制寄存器 CKCR1 的 CKINF3:0 的要求。如果CPU 时钟频率的兆赫数是奇数的话，ACLK 时钟频率实际上是 0.5MHz。

WDCLK 时钟为 DSP 控制器的看门狗与实时时钟模块提供时钟源。它的产生办法与 ACLK时钟类似。大约送出 16kHz 左右的时钟信号。如果外部时钟 CLKIN 的频率是 2Hz 的乘方，则 WDCLK 的频率为 16384Hz。如果外部时钟 CLKIN 的频率正好是 4MHz 或其倍数，则 WDCLK 的频率为 15625Hz。如果想增加 WDCLK 的频率，可以采取将时钟控制寄存器 CKCR1 的 CKINF3：0 设为不正确的值来实现，例如外部时钟 CLKIN 的频率为 4.194304MHz，但将 CKINF3：0 设为 1111（2MHz），此时 WDCLK 的频率将为 32768Hz。这不是一个很好的办法，因为它会影响到 ACLK 的频率。

DSP 控制器加电复位时，锁相环电路 PLL 未被使用（CLKMD=00），CPU 时钟为外部时钟的 2 分频，且 PLLF2：0=000，PLLDIV=0。如果要使用锁相环电路 PLL，则要设置 PLLF2：0 和 PLLDIV，并置 CLKMD1=1，经过 100μs 的延迟，锁相环电路 PLL 被加电开始工作。在这段时间若要阻止某些代码执行，可通过读取 PLL 的状态（PLLOCK1）来编程实现。如果在运行过程中修改 PLL 的倍率（PLLF2：0 和 PLLDIV），它不能马上见效。需要先将 CLKMD1 清 0，然后接着置 1，这样新的倍率值就被写入到锁相环电路 PLL 中。

与系统时钟源模块紧密相关的是系统空闲功能的实现。在系统空闲时，可以将某些时钟源停止，以节省能量。当需要时，再将其唤醒。在当今环保要求越来越高的情况下，这一点是非常有意义和重要的。DSP 控制器有四种空闲方式，由时钟控制寄存器 CKCR0 的 PLLPM1：0 来设置，参见表 2.10。当空闲指令（IDLE）被执行时，DSP 控制器将处于 PLLPM1：0 规定的某种空闲方式下。

表 2.10 系统空闲方式

方式	PLLPM1:0	Mem_Domain	Sys_Domain	DCLK	PLL状态	振荡器状态	电源消耗	退出条件
正常	××	On	On	On	On	On	>40mA	
IDEL1	00	Off	On	On	On	On	约 15mA	$\overline{RS}$、NMI、看门狗中断、任何未屏蔽的中断
IDEL2	01	Off	Off	On	On	On	约 4mA	$\overline{RS}$、NMI、看门狗中断、任何未屏蔽的外部中断、未使用 SYSCLK 的片内外设中断

（续）

方式	PLLPM1:0	Mem _ Domain	Sys _ Domain	DCLK	PLL 状态	振荡器状态	电源消耗	退 出 条 件
PLL 掉电（PPD）	10	Off	Off	On	Off	On	约 1mA	$\overline{RS}$、NMI、看门狗中断、任何未屏蔽的外部中断、未使用 SYSCLK 的片内外设中断
振荡器掉电（OPD）	11	Off	Off	Off	Off	Off	<30μA	$\overline{RS}$、NMI、任何未屏蔽的外部中断、未使用 SYSCLK 的片内外设中断

表 2.10 中的存储时钟域（Mem _ Domain）是指用在除了中断寄存器以外的存储器、寄存器的 CPUCLK；系统时钟域（SYS _ Domain）是指 CPUCLK、SYSCLK、ACLK 三种时钟源，它包含了存储时钟域和中断寄存器用的 CPUCLK。唤醒中断包含各种片内、片外中断。

2.8 系统复位

有六种信号可使 DSP 控制器复位：

●电源复位：由引脚$\overline{PORESET}$引起。该引脚产生一个由低到高的电平变化，将产生复位信号。为了可靠复位，其低电平有效时间至少需要 6 个 CPU 时钟周期。

●复位引脚$\overline{RS}$：该引脚是 I/O 类型的，当作为输入引脚时，其作用与引脚$\overline{PORESET}$相同；当作为输出引脚时，可将 DSP 控制器的复位信号送给其它的器件。

●软件复位：将系统控制寄存器 STSCR 的 RESET0 清 0 或 RESET1 置 1，将产生复位信号（见表 2.11～表 2.13）。

●非法地址：如果在程序运行过程中，出现了非法地址，将产生复位信号。

●看门狗定时器溢出：看门狗是为了监控系统运行状况而设。当系统运行出现故障时，通过看门狗定时器溢出产生复位信号，使系统重新开始。

●欠电压复位：这与 DSP 控制器的型号有关。当内嵌了欠电压检测电路时，如果发生欠电压，将产生复位信号。

系统复位见图 2.14。

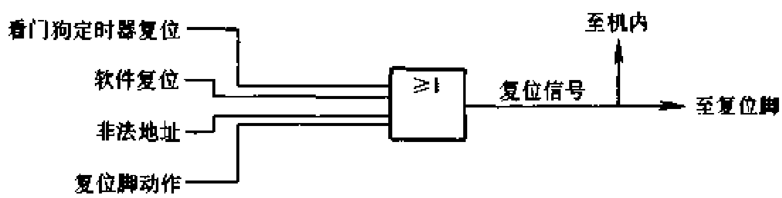


图 2.14 系统复位

复位信号实际上是一个不可屏蔽的中断。当系统收到复位信号后，将复位中断向量 0000H 加载到程序计数器 PC 中。一般情况下，该处设有一条分支指令，以跳转到主程序入口上。系统复位后：

● CNF＝0，双口存储器 DARAM（B0）分配给数据空间。

● INTM＝1，禁止可屏蔽中断。

●系统状态：OV＝0，XF＝1，SXM＝1，PM＝00，C＝1。

●全局存储器分配寄存器 GREG＝×××××××× 00000000。

●重复计数器 RPTC＝0。

●等待状态的周期设为最大。

表 2.11　系统控制寄存器(SYSCR)和系统状态寄存器(SYSSR)的组成

地址	寄存器	位数							
7018h	SYSCR	15	14	13	12	11	10	9	8
		RESET1	RESET0	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
		R/W-0	R/W-1						
		7	6	5	4	3	2	1	0
		CLKSRC1	CLKSRC0	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
		R/W-1	R/W-1						
701Ah	SYSSR	15	14	13	12	11	10	9	8
		PORST	Reserved	Reserved	ILLADR	Reserved	SWRST	WDRST	Reserved
		R/C-x			R/C-x		R/C-x	R/C-x	
		7	6	5	4	3	2	1	0
		Reserved	Reserved	HPO	Reserved	VCCAOR	Reserved	Reserved	VECRD
				R/C-1		R-1			R-0

表 2.12　系统控制寄存器(SYSCR)各位定义与说明

位	定　义		说　明
Bits15-14	RESET1	RESET0	软件复位，两位要同时写
	0	0	系统复位
	0	1	无效
	1	0	系统复位
	1	1	系统复位
Bits13-8	保　留		
Bits7-6	CLKSRC1	CLKSRC0	引脚 CLKOUT 的功能
	0	0	数字 I/O
	0	1	输出 WDCLK
	1	0	输出 SYSCLK
	1	1	输出 CPUCLK
Bits5-0	保　留		

表 2.13　系统状态寄存器(SYSSR)各位定义与说明

位	定　义		说　明
Bits15	PORST	0	没有电源复位
		1	由电源引起了复位
Bits14-13	保　留		

（续）

位	定义		说明
Bits12	ILLADR	0	没有非法地址复位
		1	由非法地址引起了复位
Bits11	保留		
Bits10	SWRST	0	没有软件复位
		1	由软件引起了复位
Bits9	WDRST	0	没有看门狗复位
		1	由看门狗引起了复位
Bits8-6	保留		
Bits5	HPO	0	正常模式
		1	FLASH 存储器编程模式
Bits4	保留		
Bits3	VCCAOR（欠压检测）	0	VCCA 自动调节
		1	VCCA 不调节
Bits2-1	保留		
Bits0	VECRD	0	系统中断向量寄存器 SYSIVR 空
		1	系统中断向量寄存器 SYSIVR 已写入

第3章 片内外设

DSP控制器是一个单片系统，除了有中央处理单元，还有片内程序存储器、数据存储器以及片内外设。片内外设包括事件管理模块（EV）、A/D转换模块（ADC）、串行通信模块（SCI）、串行外设接口模块（SPI）、中断管理系统和系统监视模块。事件管理模块（EV）含有通用定时器、比较器、PWM发生器、捕获器。A/D转换模块（ADC）包含两个8通道10位的A/D转换器，实现模拟量到数字量的转换。串行通信接口模块（SCI）是一个标准的串行异步数字通信接口模块，可以实现半双工或双工的通信，通信速率可达625kbps。串行外设接口模块（SPI）实际上是提供了一个高速同步串行总线，实现与带有SPI接口芯片的连接。目前，设备小型化的要求越来越强烈，为了减少引脚线缩小芯片尺寸，采用SPI接口的芯片越来越多，因此DSP控制器提供SPI接口为工程应用系统带来了便利。中断管理系统负责处理DSP内核中断、片内外设以及外部引脚中断的响应过程。系统监视模块由看门狗和实时中断定时器组成，负责监视DSP控制器的软件、硬件运行状况。一旦系统出现故障，可在一定时间内复位或恢复到定制的状态。

3.1 事件管理模块

在微机控制系统中，两类事件是非常重要的，一类是与时间有关的事件，另一类是外部中断事件。一个控制程序是否优良，与这两类事件使用好坏有着直接的关系。在控制程序中，经常采用定时采样、定时显示、定时轮询等方式，以及要求输出各种各样的控制波形，这些都需要通过与时间有关的事件来完成。此外，通过对时间分片还可以实现多进程的控制方式。中断是微机控制系统另一种非常好的控制方式，因为只是在有中断请求时，CPU才可能去对它服务，这样一来就可以不用软件轮询的方式来访问外设接口，从而节约软件开销，简化程序结构。DSP控制器的事件管理模块主要涉及与时间有关的事件。

图3.1是DSP控制器事件管理模块的结构示意图。它由3个通用定时器、6个全比较单元、3个单比较单元、4个捕获单元、2个正交编码脉冲电路组成。表3.1列出了与事件管理模块相关的引脚说明。

一般情况下，每个单元模块都有多种功能或多种工作方式，它的功能实现是由相关的寄存器和引脚完成。寄存器分为两大类：控制类寄存器和数据类寄存器。当前具体使用哪种功能或哪种工作方式由它的控制类寄存器来规定。因此，在使用某种功能模块之前需要对它的控制类寄存器进行初始化。这种模块也称为可编程模块。从用户编程的角度来看，使用某种功能模块实际上就是对它的控制类寄存器或数据类寄存器进行读写，而对应的引脚变化或其它的物理过程是模块自身完成的。因此，对于寄存器的地址以及每位含义的了解是至关重要的。

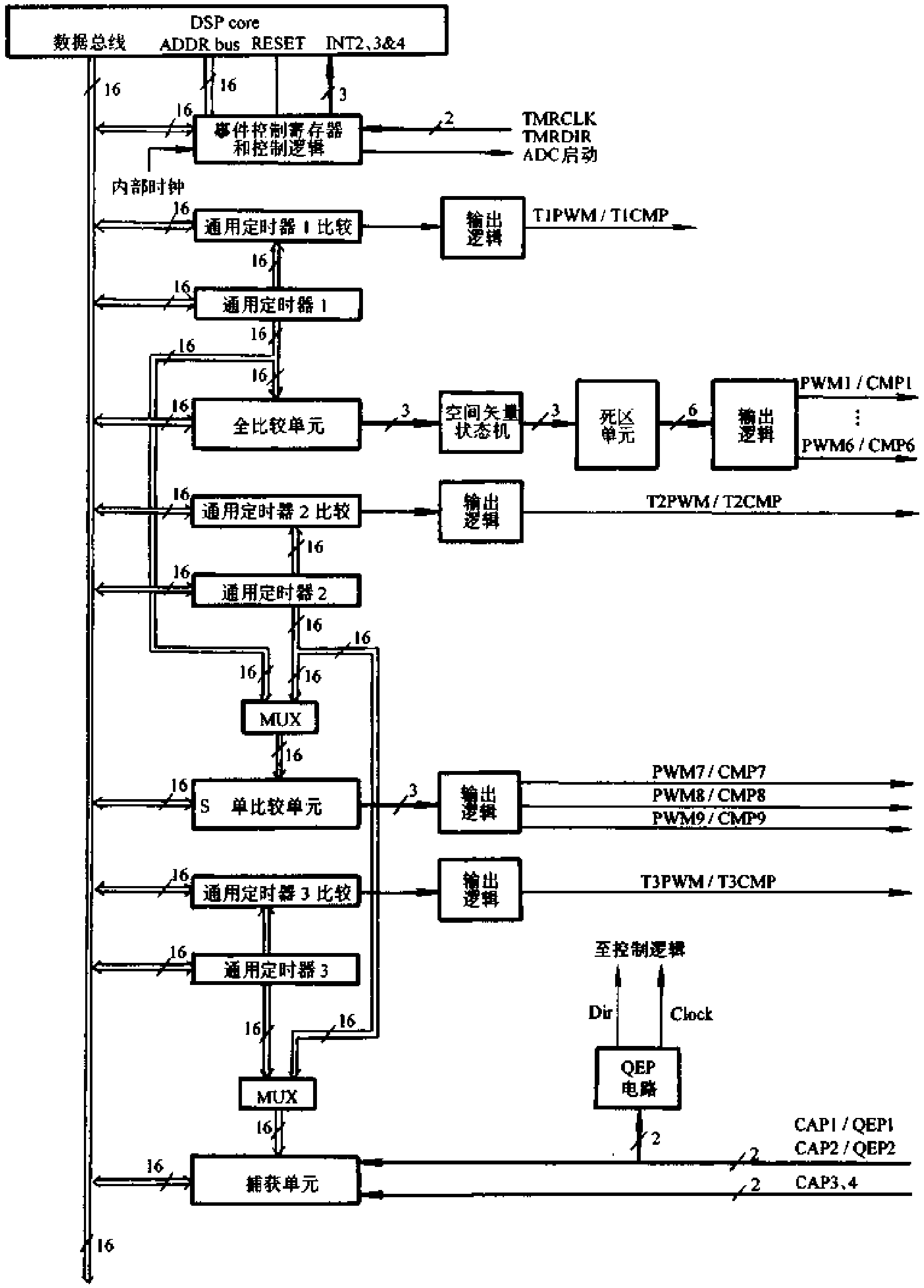


图 3.1 DSP 控制器事件管理模块的结构示意图

表 3.1 事件管理模块的相关引脚的说明

引 脚	说 明	引 脚	说 明
CAP1/QEP1	捕获单元 1 的输入/正交编码脉冲电路 1 的输入	PWM5/CMP5	全比较单元 3 的比较/PWM 输出 1
		PWM6/CMP6	全比较单元 3 的比较/PWM 输出 2
CAP2/QEP2	捕获单元 2 的输入/正交编码脉冲电路 2 的输入	PWM7/CMP7	单比较单元 1 的比较/PWM 输出
		PWM8/CMP8	单比较单元 2 的比较/PWM 输出
CAP3	捕获单元 3 的输入	PWM9/CMP9	单比较单元 3 的比较/PWM 输出
CAP4	捕获单元 4 的输入	T1PWM/T1CMP	通用定时器 1 的比较/PWM 输出
PWM1/CMP1	全比较单元 1 的比较/PWM 输出 1	T2PWM/T2CMP	通用定时器 2 的比较/PWM 输出
PWM2/CMP2	全比较单元 1 的比较/PWM 输出 2	T3PWM/T3CMP	通用定时器 3 的比较/PWM 输出
PWM3/CMP3	全比较单元 2 的比较/PWM 输出 1	TMRCLK	通用定时器外部时钟输入
PWM4/CMP4	全比较单元 2 的比较/PWM 输出 2	TMRDIR	通用定时器外部计数方向输入

3.1.1 通用定时器

定时器是最常用的外围设备，它的核心是计数器。DSP 控制器的三个通用定时器都采用 16 位的计数器，它们的计数范围是 0～65535 个脉冲。计数脉冲可以由内部时钟经分频产生，也可以由外部引脚时钟提供。计数方向可以是增计数也可以是减计数。定时器内设有周期寄存器和比较寄存器。定时器除了产生上溢（增计数时）、下溢（减计数时）事件外，当计数值与周期寄存器的值或比较寄存器的值相等时，还会产生周期匹配或比较匹配两种事件。如果开启了比较输出功能，这些事件还将引起输出引脚的电平变化。所以，DSP 控制器的通用定时器为控制系统的各种应用提供了设计上的便利。图 3.2 是通用定时器的结构示意图。

与通用定时器相关的引脚（x=1，2，3，下同）：

● TMRDIR：用于确定通用定时器计数增/减方式，高电平为增计数，低电平为减计数。

● TMRCLK：外部引脚时钟，最大频率是 CPU 时钟频率的 1/4。

● TxPWM/TxCMP：通用定时器输出引脚。在通用定时器的比较输出操作被开启时，该引脚才起作用，否则处于高阻状态。

● ADC _ Start：A/D 转换的启动信号。当定时时间到了时，自动发出这个启动信号。

与通用定时器相关的寄存器：

● TxCNT（Count）：16 位计数寄存器，可读写。

● TxCMPR（Compare）：16 位定时比较寄存器，存放待比较的值，可读写，双缓冲结构。当计数寄存器的计数值与定时比较寄存器的值相等时将产生比较匹配事件。

● TxPR（Period）：16 位定时周期寄存器，存放周期值，可读写，双缓冲结构。当计数寄存器的计数值与定时周期寄存器的值相等时将产生周期匹配事件。

● TxCON（Control）：16 位定时控制寄存器，可读写，决定操作模式、时钟选择、分频系数预定标因子以及对定时比较寄存器和定时周期寄存器的控制。

● GPTCON（GP Control）：16 位通用定时控制寄存器，可读写，主要规定由哪种定时事件启动 A/D 转换，同时也可决定四种定时事件的优先级。

对可编程功能模块（芯片）的操作，实际上就是对它的寄存器进行读写。上述五个通用

定时器相关寄存器的地址见附表 B。计数寄存器 TxCNT 是通用定时器的核心，它记录输入给它的脉冲数。计数脉冲可由内部引脚时钟提供也可由外部时钟提供，其时钟的选择及分频系数由定时控制寄存器 TxCON 相应位决定。当计数寄存器的值达到 FFFFh 时将产生上溢事件，当计数寄存器的值达到 0000h 时将产生下溢事件，并分别在中断标志寄存器 EVIFRA 和 EVIFRB 中的 TxOFINT 位和 TxUFINT 位上产生置位。

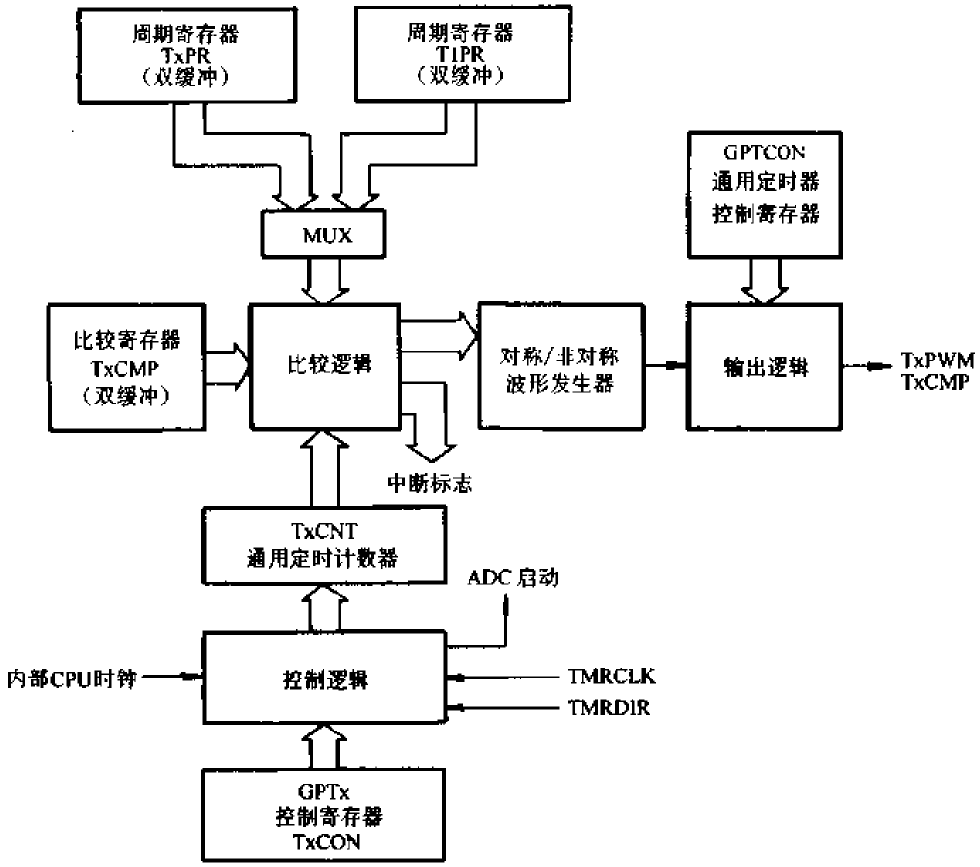


图 3.2　通用定时器的结构示意图

定时比较寄存器 TxCMPR 存放待比较的值，它是双缓冲结构，分为缓冲寄存器和工作寄存器。定时比较缓冲寄存器可在任何时候进行读写。但是，定时比较缓冲寄存器的内容什么时候装载到它的工作寄存器，取决于定时控制寄存器 TxCON 的设置，参见表 3.3。计数寄存器的值不断与定时比较工作寄存器的值进行比较，一旦相等就产生比较匹配事件，在中断标志寄存器 EVIFRA 和 EVIFRB 中的 TxCINT 位上产生置位，同时还会给出相应的 A/D 转换启动信号（取决于通用定时控制寄存器 GPTCON 的 TxADC1：0 的设置），在通用定时器的比较输出操作被开启时（由定时控制寄存器 TxCON 的 TECMPR 和通用定时控制寄存器 GPTCON 的 TCOMPOE 来设置），还将使引脚 TxPWM/TxCMP 产生跳变。

定时周期寄存器 TxCPR 与定时比较寄存器 TxCMPR 相类似，它存放周期值，也是双缓冲结构，分为缓冲寄存器和工作寄存器，可在任何时候对它的缓冲寄存器进行读写。定时周期缓冲寄存器的内容装载到它的工作寄存器，只能在计数寄存器的值为 0 时进行。计数寄存器的值不断与定时周期工作寄存器的值进行比较，一旦相等就产生周期匹配事件，在中断标志寄存器 EVIFRA 和 EVIFRB 中的 TxPINT 位上产生置位。在连续计数模式下，有了定时周

期寄存器就可以产生连续的周期信号，再通过定时比较寄存器控制脉宽，就可以产生任意调制的 PWM 波形。

定时控制寄存器 TxCON 和通用定时控制寄存器 GPTCON 决定了通用定时器的工作方式，在使用通用定时器前必须对它们进行初始化。16 位定时控制寄存器 TxCON 各位组成如表 3.2 所示，各位的定义与说明见表 3.3。16 位通用定时控制寄存器 GPTCON 各位组成如表 3.2 所示，各位的定义与说明见表 3.4。定时控制寄存器 TxCON 主要决定通用定时器的计数模式、分频系数、时钟选择、定时比较寄存器重装载条件、定时器比较输出操作的使能、定时器的开启与关闭等。通用定时控制寄存器 GPTCON 主要决定由哪个定时器的何种事件来启动 A/D 转换以及三个定时器比较输出的极性。

表 3.2 定时控制寄存器和通用定时控制寄存器

地址	寄存器	位数							
7400h	GPTCON	15	14	13	12	11	10	9	8-7
		T3STAT	T2STAT	T1STAT	T3TODAC		T2TODAC		T1TODAC
		R-1	R-1	R-1	RW-0		RW-0		RW-0
		6		5	4	3	2	1	0
		TCOMPOE		T3PIN		T2PIN		T1PIN	
		RW-0		RW-0		RW-0		RW-0	
7404h 7408h 740Ch	TxCON (x=1，2，3)	15	14	13	12	11	10	9	8
		Free	Soft	TMODE2	TMODE1	TMODE0	TPS2	TPS1	TPS0
		RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0
		7	6	5	4	3	2	1	0
		TSWT1	TENABLE	TCLKS1	TCLKS0	TCLD1	TCLD0	TECMPR	SELT1PR
		RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

表 3.3 定时控制器 TxCON 的定义与说明

位	定义			说明
Bit15-14	Free	soft		仿真控制
	0	0		仿真悬挂时立即停止
	0	1		仿真悬挂时完成当前周期后停止
	1	0		不受仿真悬挂的影响
	1	1		不受仿真悬挂的影响
Bit13-11	TMode2	TMode1	TMode0	计数模式
	0	0	0	模式 0：停止/保持
	0	0	1	模式 1：单增计数
	0	1	0	模式 2：连续增计数
	0	1	1	模式 3：定向增/减计数
	1	0	0	模式 4：单增/减计数
	1	0	1	模式 5：连续增/减计数
	1	1	0	保留
	1	1	1	保留

（续）

位	定义			说明
Bit10-8	TPS2	TPS1	TPS0	分频系数（F_c 是 CPU 时钟频率）
	0	0	0	$F_c/1$
	0	0	1	$F_c/2$
	0	1	0	$F_c/4$
	0	1	1	$F_c/8$
	1	0	0	$F_c/16$
	1	0	1	$F_c/32$
	1	1	0	$F_c/64$
	1	1	1	$F_c/128$
Bit7	TSWT1（对 T1CON 无效）		0	使用自己的使能位
			1	以 T1CON 的 TEnable 位作为使能位
Bit6	Tenable		0	停止计数，保持原值
			1	使能（启动）计数
Bit5-4	TCLKS1	TCLKS0		时钟选择
	0	0		内部时钟
	0	1		外部时钟
	1	0		仅用于 T3CON，此时通用定时器 2 和 3 级联为 32 位定时器，定时器 2 作为定时器 3 的时钟源。该位对于 T1CON 和 T2CON 以及 SELT1PR＝1 时无效
	1	1		仅用于 T2CON 和 T3CON，此时使用正交解码脉冲电路作为时钟源。该位对于 T1CON 以及 SELT1PR＝1 时无效
Bit3-2	TCLD1	TCLD0		定时比较寄存器装载条件
	0	0		计数寄存器值为 0
	0	1		计数寄存器值为 0 或等于定时周期寄存器的值
	1	0		立即
	1	1		保留
Bit1	TECMPR（定时比较使能）		0	关闭比较操作
			1	使能比较操作
Bit0	SELT1PR（选择周期寄存器）		0	使用自己的周期寄存器
			1	以 T1PR 作为自己的周期寄存器

表 3.4　通用定时控制器 GPTCON 的定义与说明

位	定义		说明
Bits15	T3STAT（只读）	0	通用定时器 3 采用减计数模式
		1	通用定时器 3 采用增计数模式
Bits14	T2STAT（只读）	0	通用定时器 2 采用减计数模式
		1	通用定时器 2 采用增计数模式

（续）

位	定义		说明
Bits13	T1STAT（只读）	0	通用定时器 1 采用减计数模式
		1	通用定时器 1 采用增计数模式
Bits12-11	T3ADC1	T3ADC0	由通用定时器 3 的事件启动 A/D
	0	0	不启动
	0	1	下溢出
	1	0	周期匹配
	1	1	比较匹配
Bits10-9	T2ADC1	T2ADC0	由通用定时器 2 的事件启动 A/D
	0	0	不启动
	0	1	下溢出
	1	0	周期匹配
	1	1	比较匹配
Bits8-7	T1ADC1	T1ADC0	由通用定时器 1 的事件启动 A/D
	0	0	不启动
	0	1	下溢出
	1	0	周期匹配
	1	1	比较匹配
Bits6	TCOMPOE	0	关闭三个通用定时器的比较输出
		1	使能（开启）三个通用定时器的比较输出
Bits5-4	T3PIN1	T3PIN0	通用定时器 3 比较输出引脚的极性
	0	0	强迫低
	0	1	低有效
	1	0	高有效
	1	1	强迫高
Bits3-2	T2PIN1	T2PIN0	通用定时器 2 比较输出引脚的极性
	0	0	强迫低
	0	1	低有效
	1	0	高有效
	1	1	强迫高
Bits1-0	T1PIN1	T1PIN0	通用定时器 1 比较输出引脚的极性
	0	0	强迫低
	0	1	低有效
	1	0	高有效
	1	1	强迫高

设 F_c 是 CPU（机器）时钟的频率，则定时时间 T 的计算公式为：

$$T=\frac{1}{(F_c/\text{分频系数})\times\text{脉冲数}} \tag{3.1}$$

式中分频系数按 TxCON 中的 TPS2：1：0 取值，脉冲数与定时周期寄存器的值或定时比较寄存器的值有关，F_c 的计算参照 2.7。

每个通用定时器有 6 种计数模式：

●停止/保持模式。

●单增计数模式。

●连续增计数模式。

●定向增/减计数模式。

●单增/减计数模式。

●连续增/减计数模式。

下面先讨论通用定时器 6 种计数模式的基本功能，即“定时”功能；然后讨论通用定时器的比较输出功能与 PWM 调制技术的实现方法。

1. 模式 0：停止/保持　在这种模式下，通用定时器停止操作并保持当前状态，计数寄存器、比较输出和分频系数都保持不变。

2. 模式 1：单增计数模式　在这种模式下，通用定时器的计数寄存器记录输入时钟的脉冲个数，直到计数寄存器的值与周期寄存器的值匹配为止。发生周期匹配后将继续完成下述工作：

●匹配之后的下一个输入时钟的上升沿，计数寄存器复位为零并且定时控制寄存器 TxCON 的使能位 Tenable 被清 0，停止计数操作；

●匹配之后的下一个 CPU 时钟周期，周期匹配中断标志置位，如果通用控制寄存器 GPTCON 规定该定时器的周期匹配事件启动 A/D 转换器，那么在设置周期匹配中断标志的同时，发送 A/D 转换器启动信号；

●匹配之后的下两个 CPU 时钟周期，定时器的下溢中断标志置位，如果通用控制寄存器 GPTCON 规定该定时器的下溢事件启动 A/D 转换器，那么在设置下溢中断标志的同时，发送 A/D 转换器启动信号。

图 3.3 反映了通用定时器单增计数模式下的工作过程。

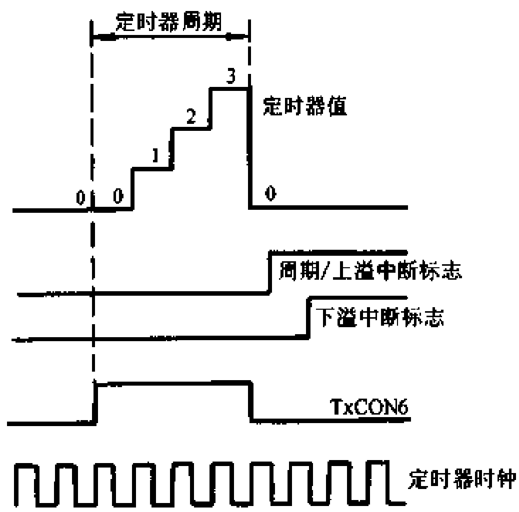


图 3.3　通用定时器单增计数模式下的工作过程

如果计数寄存器的初始值大于周期寄存器的值，计数寄存器首先从初始值计数到 FFFFh，并产生上溢中断标志；然后复位为零，再重新计数至与周期寄存器的值匹配，周期匹配之后的工作与前面所述一致。

如果计数寄存器的初始值等于周期寄存器的值，定时器将立即产生周期匹配，其后的工作同前所述。

在单增计数模式下，计数方向始终是增，引脚

TMRDIR 不起作用，通用控制寄存器 GPTCON 中计数方向指示位 TxSATA 是 1。一个单增计数周期所计的脉冲数为（TxPR+1）。在该模式下，如果要产生连续的定时事件，就要在发生周期匹配后，通过软件使定时控制寄存器 TxCON 的使能位 Tenable 置 1，重新开始下一次的计数操作。

3. 模式 2：连续增计数模式　单增计数模式的连续重复就是连续增计数模式。在这种模式下，计数寄存器不断计数，并与定时周期寄存器进行匹配，一旦匹配就像单增计数模式一样对相应的中断标志进行置位，计数寄存器复位为 0，然后重新开始下一次计数周期。每个计数周期的脉冲数为 TxPR+1。图 3.4 反映了通用定时器连续增计数模式下的工作过程。

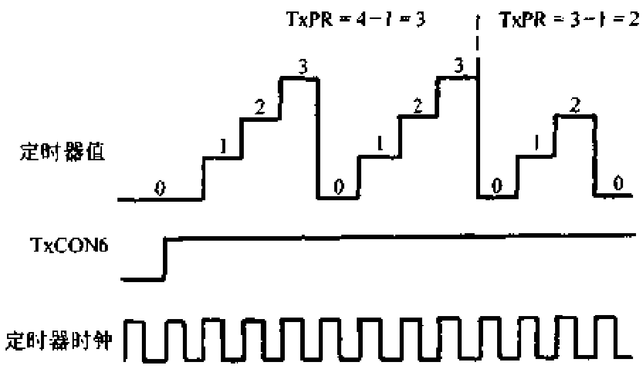


图 3.4　通用定时器连续增计数模式下的工作过程

在连续增计数模式下，可以改变定时周期寄存器的值，以得到不同的周期信号。但要记住，对定时周期寄存器值的修改是对它的缓冲寄存器进行的，要将它装载到工作寄存器需在计数寄存器复位为 0 的时刻，这个装载是自动完成的。

连续增计数模式下的周期匹配、下溢和上溢中断标志的置位，以及对 A/D 转换的启动操作都与单增计数模式下相同。在连续增计数模式下，引脚 TMRDIR 同样不起作用。通用定时器的连续增计数模式特别适用于产生边沿触发或异步 PWM 波形，以及实现定时采样、定时显示、定时轮寻等控制方式。

4. 模式 3：定向增/减计数模式　在定向增/减计数模式下，由引脚 TMRDIR 来规定是增计数还是减计数。当引脚 TMRDIR 为高电平时，计数寄存器进行增计数；当引脚 TMRDIR 为低电平时，计数寄存器进行减计数。定向增/减计数模式特别适用于位置控制，如步进马达的控制、伺服控制等。以增计数代表前进方向，以减计数代表后退方向，以脉冲数表示位移量，这样一来就完全决定了位置的状况。

对于通用定时器 1 和通用定时器 3，其定向增/减计数模式的操作有如下特点：

●只要引脚 TMRDIR 为低电平，计数寄存器的值不断减 1，并与 0000h 进行比较，一旦匹配即产生下溢出事件。随后，计数寄存器保持原值。

●只要引脚 TMRDIR 为高电平而且计数寄存器的初始值小于或等于定时周期寄存器的值，计数寄存器的值不断增 1，并与定时周期寄存器的值进行比较。一旦与定时周期寄存器的值发生匹配，将产生周期匹配事件。随后，计数寄存器保持原值。

●只要引脚 TMRDIR 为高电平而且计数寄存器的初始值大于定时周期寄存器的值，计数寄存器的值不断增 1，并与 FFFFh 进行比较。一旦与 FFFFh 发生匹配，将产生上溢事件。随后，计数寄存器保持原值。

●引脚 TMRDIR 电平改变引起计数方向的改变需要延迟两个 CPU 时钟周期。引脚 TMRDIR 的电平状态会在通用定时控制寄存器 GPTCON 中的方向指示位 TxSATA 得到反映：1 表示增计数；0 表示减计数。

●由于是由引脚 TMRDIR 来规定当前是增计数还是减计数。因此，当引脚 TMRDIR 为高

电平而且计数寄存器的值还未达到定时周期寄存器的值或 FFFFh 时，将引脚 TMRDIR 变为低电平，这时将进行减计数，定时器将不会发生周期匹配或上溢出事件。类似的道理，也有可能不发生下溢出事件。

●周期匹配、下溢和上溢所产生的中断标志，以及相关的操作都与单增计数模式相同。

通用定时器 2 的定向增/减计数模式工作过程与通用定时器 1 或通用定时器 3 的定向增/减计数模式的基本一样。只是在增计数过程中，当与定时周期寄存器的值发生匹配后（产生周期匹配事件），若引脚 TMRDIR 仍为高电平，将还会继续进行增计数，不在定时周期寄存器的值上停留。因此，通用定时器 2 的定向增/减计数模式工作过程是在上溢与下溢之间滚动。这样设计通用定时器 2 的定向增/减计数模式工作过程，主要是为了适应正交编码脉冲电路的工作（参见 3.1.4）。图 3.5 反映了通用定时器定向增减计数模式的工作过程。

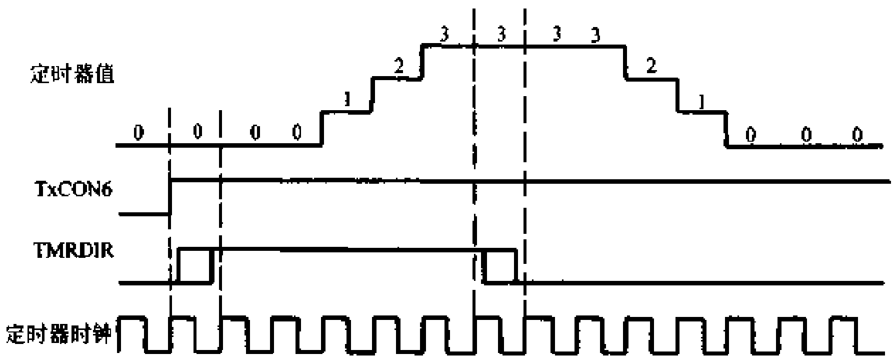


图 3.5　通用定时器定向增减计数模式的工作过程

5. 模式 4：单增/减计数模式　在这种模式下，通用定时器的计数寄存器由初始值增计数至定时周期寄存器的值，然后改变计数方向进行减计数至 0 为止。当计数至 0 时，定时控制寄存器 TxCON 的使能位 Tenable 被清 0，停止计数操作，保持当前状态。

如果计数寄存器的初始值大于定时周期寄存器的值时，计数寄存器先计数至 FFFFh 再复位为 0，然后由 0 增计数至定时周期寄存器的值，然后改变计数方向进行减计数至 0 为止。

周期匹配、下溢和上溢所产生的中断标志，以及相关的操作都与单增计数模式相同，并且引脚 TMRDIR 不起作用。如果计数寄存器的初始值为 0，则其计数周期为 2×（TxPR）个时钟周期。在完成一次单增/减计数过程后，可通过软件使定时控制寄存器 TxCON 的使能位置位 Tenable 置 1，重新开始下一次的单增/减计数过程。图 3.6 反映了通用定时器单增/减计数模式的工作过程。

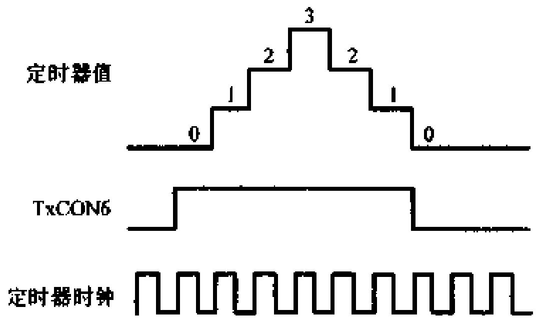


图 3.6　通用定时器单增/减计数模式的工作过程

6. 模式 5：连续增/减计数模式　单增/减计数模式的重复操作就是连续增/减计数模式。在这种操作模式下，一旦计数开始无需软件或硬件干涉，计数寄存器从 0 增计数到定时周期寄存器值，然后减计数至 0，周而复始。在循环计数的过程中，可以改变定时周期寄存器的值，但需到计数寄存器为 0 时才会生效。定时器周期是 2×（TxPR）个输入时钟周期。

如果计数寄存器最开始的值大于定时周期寄存器的值，则先计数至 FFFFh，然后溢出到 0，再周而复始地从 0 到定时周期寄存器值再到 0。如果计数寄存器最开始的值等于定时周期寄存器的值，则先减计数至 0，再周而复始地从 0 到定时周期寄存器值再到 0。

在循环计数的过程中，产生周期匹配、上溢、下溢等中断标志，以及相关的操作与单增计数模式一样。在该模式下，引脚 TMRDIR 不起作用。图 3.7 反映了通用定时器连续增/减计数模式的工作过程。

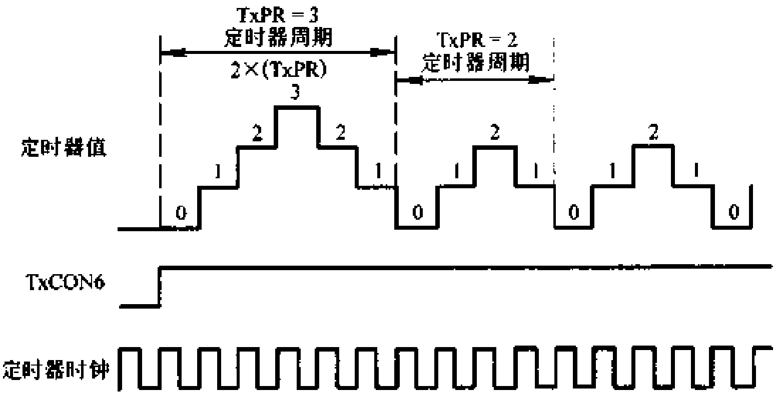


图 3.7　通用定时器连续增/减计数模式的工作过程

连续增/减计数模式适用于产生对称的 PWM 波形，该波形广泛应用于运动控制系统和电力电子等电器设备中。

7. 调制技术的基本实现原理　前面讨论了通用定时器六种计数模式，它的最基本的功能是产生“定时”事件，通过改变定时周期寄存器的值可以得到不同的定时时间，除了周期匹配这种“定时”事件外，还有上溢、下溢事件可利用。这一些都比普通的微处理器的通用定时器的功能要强。但是，对于 DSP 控制器上述功能仅仅是其中的一部分。由于 DSP 控制器的通用定时器都有一个相关的比较寄存器 TxCMPR 和一个输出引脚 TxPWM/TxCMP，因此可以利用它们做成波形发生器。

在数字控制系统中，无论采用什么样的控制方案，最终都需要将数字的控制策略转化成模拟信号以控制外部对象。由于目前大部分功率器件都是开关型器件，因此这种转化过程最常用的一种方法就是采用脉宽调制（PWM）技术，将数字量调制成满足控制策略的各种波形，最后施加到被控对象上。调制技术的核心就是产生周期不变但脉宽可变的信号。周期不变意味着以同样的调制频率工作，脉宽可变意味着可以得到不同的波形。脉宽应该以一种什么规律变化才能得到满足控制策略的波形，这不是本书要讨论的问题（有兴趣的读者可参考有关控制方面的书籍），本书假定已得到脉宽变化的规律，问题是怎样去实现它？

从前面简单的叙述知，调制波形是一系列周期信号，但每个周期中的脉宽是不同的。为了产生这种波形，可以想像需要两种事件：周期匹配与比较匹配。周期匹配保证调制波形的周期不变，比较匹配产生不同的脉宽。因此，根据调制频率来设置定时周期寄存器的值；根据已得到的脉宽变化规律在每个周期内修改定时比较寄存器的值，以得到不同的脉宽。图 3.8 是调制技术的原理图。

前面探讨了调制技术实现的原理，下面具体研究 DSP 控制器的通用定时器是怎样产生调制波形的。要从通用定时器的输出引脚 TxPWM/TxCMP 产生输出信号，必须首先将定时控

制寄存器 TxCON 的 TECMPR 位置 1、以及通用定时控制寄存器 GPTCON 的 TCOMPOE 位置 1，使能（开启）通用定时器的比较输出操作。另外，要对通用定时控制寄存器 GPTCON 的 TxPIN1 和 TxPIN0 进行设置，以规定当发生比较匹配事件时引脚 TxPWM/TxCMP 应处于何种极性（参见表 3.4）。

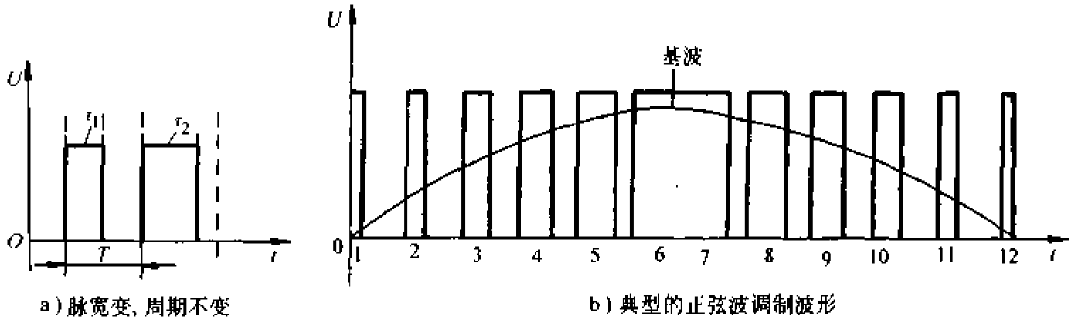


图 3.8 调制技术的原理图

通用定时器的定时比较寄存器 TxCMPR 是一个有特殊意义的寄存器，无论在何种计数模式下，计数寄存器的值总是与定时比较寄存器的值进行比较，当它们相等时，便发生比较匹配事件，并且产生以下相关操作：

●在匹配后的 2 个 CPU 时钟，将中断标志寄存器 EVIFRA 和 EVIFRB 中的比较中断标志位 TxCINT 置位。

●如果通用定时器不处于定向增/减计数模式，并且定时控制寄存器 TxCON 的 TECMPR 位置 1，那么在匹配后的一个 CPU 时钟周期，根据通用定时控制寄存器 GPTCON 的 TxPIN1 和 TxPIN0 设置的情况，相关的引脚 TxPWM/TxCMP 上将发生跳变。如果 TxPIN1 和 TxPIN0 设置为“低有效”，则在比较匹配发生前引脚 TxPWM/TxCMP 为高电平（无效），发生比较匹配后引脚 TxPWM/TxCMP 跳变为低电平（有效）；如果设置为“高有效”，其结果与“低有效”相反；如果设置为“强制低”，则无论比较匹配发生前引脚 TxPWM/TxCMP 是何种电平，发生比较匹配后引脚 TxPWM/TxCMP 都被强制为低电平；如果设置为“强制高”，其结果与“强制低”相反。

●如果通用定时控制寄存器 GPTCON 的 TxADC1 和 TxADC0 位允许比较匹配事件启动 A/D 转换，那么当比较中断标志位置位的同时将产生 A/D 转换的启动信号。

●如果比较中断标志未被屏蔽，且同组中没有其它更高优先级的中断被挂起，则由比较中断标志产生的中断请求被送至 CPU。

●如果通用定时器处于连续计数模式，则引脚 TxPWM/TxCMP 输出连续的信号；如果通用定时器处于单计数模式，则引脚 TxPWM/TxCMP 只发生一次跳变。

出现下列情况时，通用定时器的输出引脚 TxPWM/TxCMP 被置成高阻状态：

●通用定时控制寄存器 GPTCON 的 TCOMPOE 位被置为 0；

●引脚 PDPINT（电源保护中断）置为低电平且该中断未屏蔽；

●系统复位。

8. 非对称波形发生器　如果定时控制寄存器 TxCON 的 TECMPR 位置 1，通用定时控制寄存器 GPTCON 的 TCOMPOE 位置 1，并对通用定时控制寄存器 GPTCON 的 TxPIN1 和 TxPIN0 进行了适当的设置、不失一般性、假定设置为“高有效”；另外，将通用定时器的计

数模式设置为单增或连续增计数模式。这样一来，就可以得到非对称的波形发生器，其原理如图 3.9 所示。

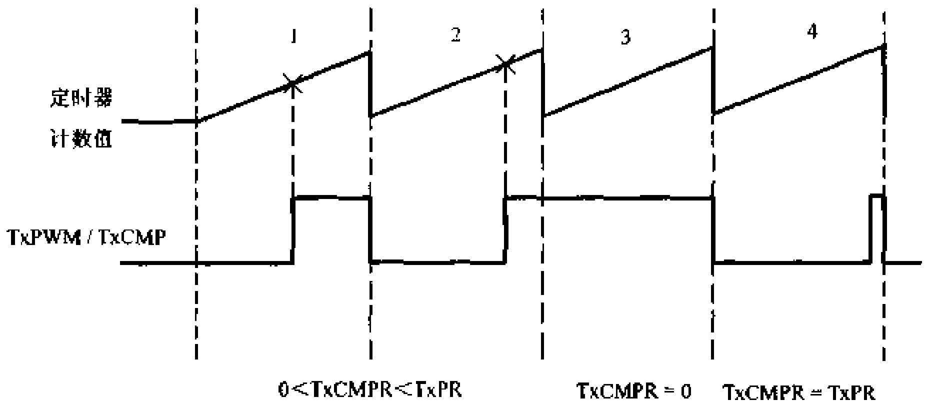


图 3.9 非对称波形发生器的原理

设定时比较寄存器的值为 TxCMPR，定时周期寄存器的值为 TxPR。在正常情况下，即 0<TxCMPR<TxPR，非对称波形发生器的工作过程如下：

●步骤 1：计数寄存器从 0 开始增计数，在未达到定时比较寄存器的值之前，引脚 TxPWM/TxCMP 输出为“无效”的低电平；

●步骤 2：当计数寄存器的值与定时比较寄存器的值相等时，产生比较匹配事件，引脚 TxPWM/TxCMP 输出为“有效”的高电平，此时可以修改定时比较寄存器（缓冲器）的值，为下一周期的脉宽做准备；

●步骤 3：随后，计数寄存器继续增计数至定时周期寄存器的值，此时将产生周期匹配事件，引脚 TxPWM/TxCMP 输出恢复为“无效”的低电平；然后，计数寄存器复位为 0，产生下溢事件，同时将定时比较寄存器进行重装载（如果定时控制寄存器 TxCON 的 TCLD1：0=00）。

●步骤 4：如果下一周期的定时比较寄存器的值还是大于 0 但小于定时周期寄存器的值，则又回到步骤 1 重新一个新的周期。图 3.9 的第 1、第 2 个周期就是这种情况。

由于单增或连续增计数模式的脉冲周期 $T=$（TxPR+1）个计数时钟，而“有效”电平时间 $\tau=$（T−TxCMPR）个计数时钟，故脉冲的占空比为：

$$\delta=\tau/T=(\mathrm{TxPR}+1-\mathrm{TxCMPR})/(\mathrm{TxPR}+1) \tag{3.2}$$

下面有三种特殊情况：

● TxCMPR=0。此时，整个周期都是“有效”电平时间，即 $\delta=100\%$。图 3.9 的第 3 个周期就是这种情况。

● TxCMPR=TxPR。此时，整个周期有 1 个计数时钟是“有效”电平时间。这是由于周期匹配事件发生后需有 1 个计数时钟的延迟来完成相关操作。图 3.9 的第 4 个周期就是这种情况。

● TxCMPR>TxPR。此时，整个周期都是“无效”电平时间，因为不可能发生比较匹配事件，即 $\delta=0\%$。

另外需要说明的是，在每次比较匹配事件发生时，可以为下一个周期的脉宽设置新的定时比较寄存器的值（可在比较匹配中断服务子程序中进行），这个新的定时比较寄存器的值何

时生效，取决于定时控制寄存器 TxCON 的 TCLD1：0 的设置。一般情况下，定时周期寄存器的值是不变化的。但这也不是绝对的，在有的控制系统中需要更改调制频率，这样就需要对定时周期寄存器的值重新修正。一句话，灵活地安排定时比较寄存器的值和定时周期寄存器的值，可以得到各种各样的调制波形。

9. 对称波形发生器　对称波形发生器与非对称波形发生器的工作原理类似。首先将定时控制寄存器 TxCON 的 TECMPR 位置 1，通用定时控制寄存器 GPTCON 的 TCOMPOE 位置 1，并对通用定时控制寄存器 GPTCON 的 TxPIN1 和 TxPIN0 进行了适当的设置，不失一般性，假定设置为"高有效"；另外，将通用定时器的计数模式设置为单增/减或连续增/减计数模式。图 3.10 是对称波形发生器的原理图。

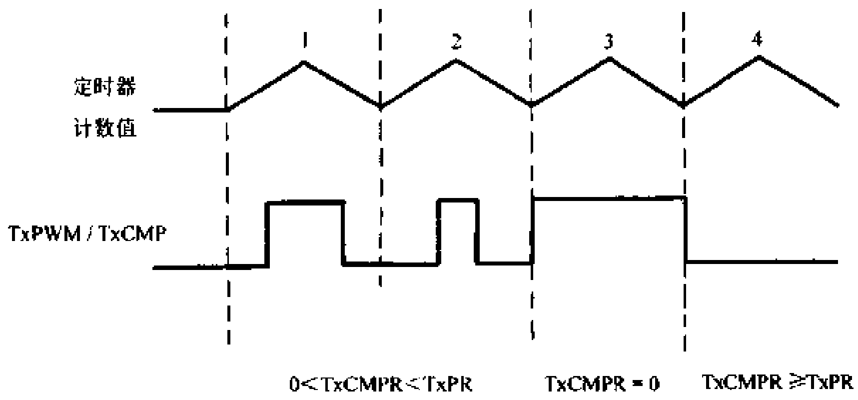


图 3.10　对称波形发生器的原理图

设定时比较寄存器的值为 TxCMPR，定时周期寄存器的值为 TxPR。在正常情况下，即 0<TxCMPR<TxPR，对称波形发生器的工作过程如下：

●步骤 1：计数寄存器从 0 开始增计数，在未达到定时比较寄存器的值之前，引脚 TxPWM/TxCMP 输出为"无效"的低电平。

●步骤 2：当计数寄存器的值与定时比较寄存器的值相等时，产生第 1 次比较匹配事件，引脚 TxPWM/TxCMP 输出为"有效"的高电平。

●步骤 3：随后，计数寄存器继续增计数至定时周期寄存器的值，此时将产生周期匹配事件。由于计数模式是单增/减或连续增/减计数模式，计数寄存器将继续减计数并与定时比较寄存器的值比较，当再次相等时将产生第 2 次比较匹配事件，引脚 TxPWM/TxCMP 输出恢复为"无效"的低电平。然后，计数寄存器继续减计数至 0，产生下溢事件，完成一个周期。

●步骤 4：如果下一周期的定时比较寄存器的值还是大于 0 但小于定时周期寄存器的值，则又回到步骤 1 重新一个新的周期。图 3.10 的第 1、第 2 个周期就是这种情况。

由于计数模式是单增/减或连续增/减计数模式，在一个周期内将发生两次比较匹配事件，如果这两次比较匹配事件的定时比较寄存器的值是一样的，那么得到的波形就是以周期匹配点为中心的对称波形。如果在周期匹配事件上修改定时比较寄存器（缓冲器）的值，并将定时比较寄存器进行重装载的条件设置为下溢事件（定时控制寄存器 TxCON 的 TCLD1：0=00），那么将保证得到对称的调制波形。

从理论上讲，修改定时比较寄存器（缓冲器）的值不一定要在周期匹配事件上，也不一

定采用下溢的重装载条件。如果在一个周期内两次比较匹配事件的定时比较寄存器的值不一样，也可以得到非对称的调制波形。另外，可以根据不同的调制频率设置不同的定时周期寄存器的值。

由于单增/减或连续增/减计数模式的脉冲周期 $T=2\times \text{TxPR}$ 个计数时钟，而“有效”电平时间 $\tau=(\text{TxPR}-\text{TxCMPR}_{\text{up}})+(\text{TxPR}-\text{TxCMPR}_{\text{down}})$ 个计数时钟，$\text{TxCMPR}_{\text{up}}$是第 1 次比较匹配的定时比较寄存器的值，$\text{TxCMPR}_{\text{down}}$第 2 次比较匹配的定时比较寄存器的值，故脉冲的占空比为：

$$\delta=\tau/T=(2\times \text{TxPR}-\text{TxCMPR}_{\text{up}}-\text{TxCMPR}_{\text{down}})/(2\times \text{TxPR}) \quad (3.3)$$

下面说明有两种特殊情况：

● $\text{TxCMPR}_{\text{up}}=\text{TxCMPR}_{\text{down}}=0$。此时，整个周期都是“有效”电平时间，即 $\delta=100\%$。图 3.10 的第 3 个周期就是这种情况。

● $\text{TxCMPR}_{\text{up}}=\text{TxCMPR}_{\text{down}}\geqslant \text{TxPR}$。此时，整个周期都是“无效”电平时间，即 $\delta=0\%$。图 3.10 的第 4 个周期就是这种情况。

10. 组成 32 位定时器　DSP 控制器的三个通用定时器都是 16 位的，在需要计数范围更大时，可将通用定时器 2 和 3 级联成 32 位的定时器。其中，32 位的计数寄存器由通用定时器 2 的计数寄存器（低 16 位）和通用定时器 3 的计数寄存器（高 16 位）组成；32 位的周期寄存器由通用定时器 2 的周期寄存器（低 16 位）和通用定时器 3 的周期寄存器（高 16 位）组成；比较寄存器还是各自独立存在。

此时，通用定时器 3 的输入时钟信号是通用定时器 2 的溢出信号。通用定时器 2 的输入时钟信号可以从内部时钟、外部时钟和正交编码脉冲电路（QEP）中选择。如果以正交编码脉冲电路作为定时器的输入时钟源，其正交编码脉冲输入的频率必须小于等于 CPU 时钟频率的 1/4。这是因为正交解码脉冲电路产生的时钟频率是正交解码脉冲输入通道频率的 4 倍，详情见 3.1.4。

级联的 32 位定时器的计数模式只能采用定向增/减计数模式。它的周期匹配事件是基于 32 位周期寄存器的；它的上溢和下溢事件也是基于 32 位的（FFFFFFFFh 或 00000000h）；但比较匹配事件是按各自的比较寄存器的值与各自的计数寄存器的值进行比较产生的。

11. 通用定时器的同步　在许多应用系统中，常常需要得到一组同步的信号，即它们有相同的周期，但互差一定的相位。DSP 控制器的三个通用定时器可以实现这个同步过程。

●将 T2CON 和 T3CON 的 TSWT1 置 1，让通用定时器 1 同时启动三个通用定时器；

●将 T2CON 和 T3CON 的 SELT1PR 置 1，让三个通用定时器共用通用定时器 1 的 T1PR 作为周期寄存器；

●将三个通用定时器的计数寄存器的初始值按相位差进行适当设置。

12. 通用定时器复位　系统发生复位事件发生时，通用定时器产生以下变化：

● GPTCON 中计数方向指示位都置成 1，其余位都复位为 0，因此所有通用定时器的操作被禁止。

●所有定时器中断标志位复位为 0，所有定时器中断屏蔽位复位为 0，因此所有定时器中断被屏蔽。

●所有通用定时器的比较输出引脚置成高阻状态。

3.1.2 比较单元与 PWM 发生器

前面较详细讨论了 DSP 控制器的三个通用定时器的功能及使用方法。与此相关，DSP 控制器还有三个单比较单元（Simple Compare Units）和三个全比较单元（Full Compare Units）。每个单比较单元有一个关联的输出引脚 CMPy/PWMy（y=7，8，9）；每个全比较单元有一对关联的输出引脚 CMPy/PWMy 和 CMPy+1/PWMy+1（y=1，3，5），这是为桥式电路所设计，一对输出引脚对应一组桥臂，当上桥臂开启时下桥臂应关闭，反之亦然。三个单比较单元和三个全比较单元的功能与通用定时器的比较输出的功能完全类似，可以独立地提供六个 PWM 输出波形。

1. 单比较单元　单比较单元结构框图如图 3.11 所示。

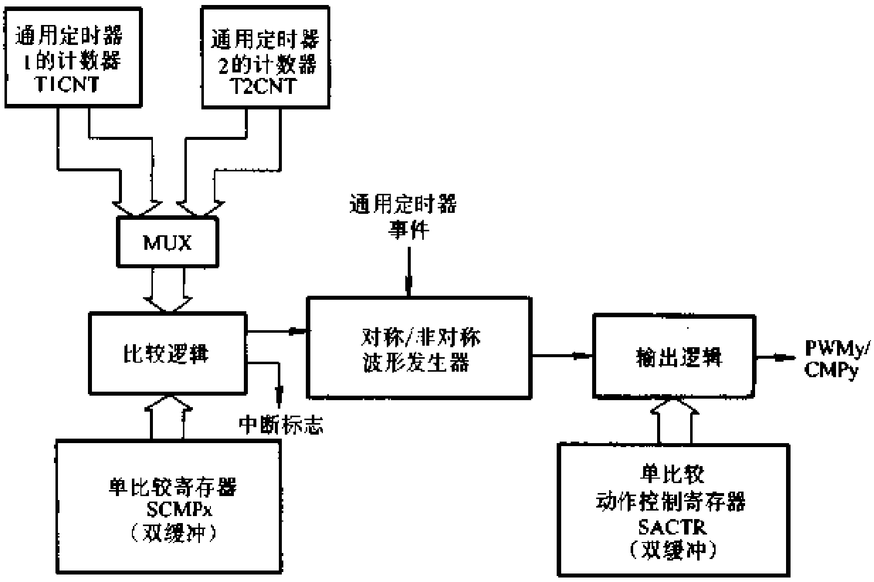


图 3.11　单比较单元结构框图

三个单比较单元包括：

- 三个 16 位比较寄存器（SCMPRx，x=1，2，3），双缓冲结构。
- 一个 16 位比较控制寄存器 COMCON，与全比较单元共享。
- 一个 16 位单比较动作控制寄存器 SACTR，双缓冲结构。
- 三个非对称/对称波形发生器。
- 三个输出引脚（三态）CMPy/PWMy（y=7、8、9）。

单比较动作控制寄存器 SACTR 和比较控制寄存器 COMCON 是单比较单元控制类寄存器，在单比较单元使用前需要对它们进行初始化设置；比较寄存器是存储待比较的值，属于单比较单元数据类寄存器。除了这个数据寄存器外，要进行比较操作还需要一个计数寄存器和一个周期寄存器。因此，单比较单元不能独自工作，需要和通用定时器联合工作。从图 3.11 可看出，配合单比较单元工作的通用定时器可以在通用定时器 1 和通用定时器 2 之间选择。

与比较单元相关的寄存器见表 3.5，包括单比较动作控制寄存器 SACTR，其各位的定义与说明见表 3.6；比较控制寄存器 COMCON，其各位的定义与说明见表 3.7；全比较动作控制寄存器 ACTR，其各位定义与说明见表 3.8；死区控制寄存器 DBTCON，其各位定义与说明见表 3.9。

表 3.5 比较单元的控制寄存器

地址	寄存器	位数							
7411h	COMCON	15	14	13	12	11	10	9	8
		CENABLE	CLD1	CLD0	SVENABLE	ACTRLD1	ACTRLD0	FCOMPOE	SCOMPOE
		RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0
		7	6	5	4	3	2	1	0
		SELTMR	SCLD1	SCLD0	SACTRLD1	SACTRLD0	SELCMP3	SELCMP2	SELCMP1
		RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0
7413h	ACTR	15	14	13	12	11	10	9	8
		SVRDIR	D2	D1	D0	CMP6ACT1	CMP6ACT0	CMP5ACT1	CMP5ACT0
		RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0
		7	6	5	4	3	2	1	0
		CMP4ACT1	CMP4ACT0	CMP3ACT1	CMP3ACT0	CMP2ACT1	CMP2ACT0	CMP1ACT1	CMP1ACT0
		RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0
7414h	SACTR	15	14	13	12	11	10	9	8
		Reserved							
		7	6	5	4	3	2	1	0
		Reserved		SCMP3ACT1	SCMP3ACT1	SCMP2ACT1	SCMP2ACT0	SCMP1ACT1	SCMP1ACT0
				RW-0	RW-0	RW-0	RW-0	RW-0	RW-0
7415h	DBTCON	15	14	13	12	11	10	9	8
		DBT7	DBT6	DBT5	DBT4	DBT3	DBT2	DBT1	DBT0
		RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0
		7	6	5	4	3	2	1	0
		EDBT3	EDBT2	EDBT1	DBTPS1	DBTPS0	Reserved		
		RW-0	RW-0	RW-0	RW-0	RW-0			

表 3.6 单比较动作控制寄存器 SACTR 各位的定义与说明

位	定义		说明
Bits15-6	保留		
Bits5-4	SCMP3ACT1	SCMP3ACT0	引脚 PWM9/CMP9 的极性
	0	0	强制低
	0	1	低有效
	1	0	高有效
	1	1	强制高
Bits3-2	SCMP2ACT1	SCMP2ACT0	引脚 PWM8/CMP8 的极性
	0	0	强制低
	0	1	低有效
	1	0	高有效
	1	1	强制高
Bits1-0	SCMP1ACT1	SCMP1ACT0	引脚 PWM7/CMP7 的极性
	0	0	强制低
	0	1	低有效
	1	0	高有效
	1	1	强制高

表 3.7 比较控制寄存器 COMCON 各位的定义与说明

位	定义		说明
Bits15	CEnable	0	关闭全比较操作
		1	开启全比较操作
Bits14-13	CLD1	CLD0	全比较寄存器重载条件
	0	0	T1CNT=0，通用定时器 1 下溢
	0	1	T1CNT=0 或 T1CNT=T1PR
	1	0	立即
	1	1	保留
Bits12	SVEnable	0	关闭空间矢量 PWM 模式
		1	开启空间矢量 PWM 模式
Bits11-10	ACTRLD1	ACTRLD0	全比较动作控制寄存器重载条件
	0	0	T1CNT=0，通用定时器 1 下溢
	0	1	T1CNT=0 或 T1CNT=T1PR
	1	0	立即
	1	1	保留
Bits9	FCOMPOE	0	全比较输出引脚呈高阻状态
		1	全比较输出引脚呈使能状态
Bits8	SCOMPOE	0	单比较输出引脚呈高阻状态
		1	单比较输出引脚呈使能状态

（续）

位	定义		说明	
Bits7	SELTMR	0	通用定时器 1 为单比较的时基	
		1	通用定时器 2 为单比较的时基	
Bits6-5	SCLD1	SCLD0	单比较寄存器重载条件（y=1，2）	
	0	0	TyCNT=0	
	0	1	TyCNT=0 或 TyCNT=TyPR	
	1	0	立即	
	1	1	保留	
Bits4-3	SACTRLD1	SACTRLD0	单比较动作控制寄存器重载条件	
	0	0	TyCNT=0（y=1，2）	
	0	1	TyCNT=0 或 TyCNT=TyPR（Y=1，2）	
	1	0	立即	
	1	1	保留	
Bits2	SELCMP3	0	比较模式	第 3 个全比较单元
		1	PWM 模式	
Bits1	SELCMP2	0	比较模式	第 2 个全比较单元
		1	PWM 模式	
Bits0	SELCMP1	0	比较模式	第 1 个全比较单元
		1	PWM 模式	

在单比较单元开始工作前，首先要设置比较控制寄存器 COMCON 的 SELTMR 位，以决定是采用通用定时器 1 还是通用定时器 2 作为单比较单元的时基；然后，要使能比较控制寄存器 COMCON 的 SCOMPOE 位，使单比较输出引脚处于工作状态；最后，要设置单比较寄存器重装载条件（COMCON 的 SCLD1：0 位）、单比较动作控制寄存器重装载条件（COMCON 的 SACTRLD1：0 位）以及单比较输出引脚的极性（单比较动作控制寄存器 SACTR）。

单比较单元的工作原理与通用定时器的比较输出原理完全一样，即由作为时基的通用定时器的周期寄存器实现 PWM 的调制频率（周期），由单比较单元的比较寄存器控制脉冲的宽度，从而得到所需要的调制波形。

●首先选择通用定时器 Tz（z=1 或 2）作为单比较单元的时基，并设置它的计数模式，如果要产生连续 PWM 波形，计数模式设置为连续增或连续增/减计数模式，否则设置为单增或单增/减计数模式。

●根据调制频率设置相应的定时周期寄存器 TzPR 的值，初始化计数寄存器 TzCNT 的值。然后，启动定时器。

●按照脉宽的变化规律，设置当前的单比较寄存器 SCMPx（x=1，2，3）值。计数寄存器 TzCNT 按照计数模式进行计数，并与单比较寄存器 SCMPx 的值进行比较，若相等将发生单比较匹配事件，并在延迟 2 个 CPU 时钟后在中断标志寄存器 EVIFRA 的 SCMPxINT 位上置 1，同时使输出引脚 CMPy/PWMy（y=7，8，9）按设定的极性发生变化。

●与上类似，计数寄存器 TzCNT 也同时与定时周期寄存器 TzPR 值进行比较，若相等将发生定时周期匹配事件，从而引发与通用定时器完全一致的相关操作。

●按照在比较控制寄存器 COMCON 设置的单比较寄存器重载条件，为下一周期准备一个新的脉冲宽度。如此循环，得到需要的 PWM 波形。

在这里要说明的是，三个单比较单元共用同一个通用定时器。因此，三个单比较单元输出的 PWM 波形具有同样的调制周期(频率)，但它们可有各自的脉宽变化规律。

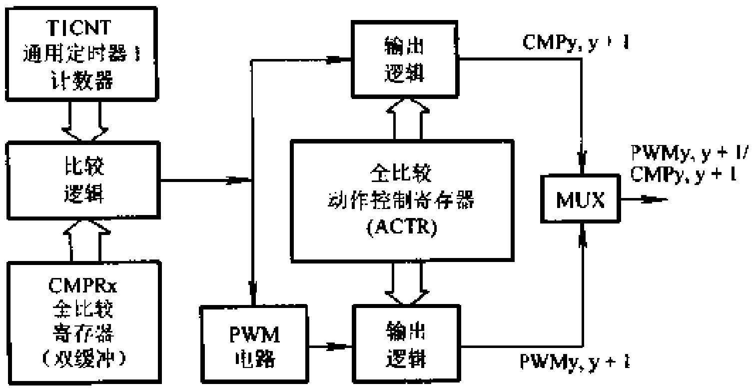


图 3.12　全比较单元结构框图

2. 全比较单元　全比较单元结构框图如图 3.12 所示。

三个全比较单元包括：

●三个 16 位比较寄存器（CMPRx，x=1，2，3），双缓冲结构。

●一个 16 位比较控制寄存器 COMCON，与单比较单元共享。

●一个 16 位动作控制寄存器 ACTR，双缓冲结构。

●三对输出引脚（三态）CMPy/PWMy 和 CMPy+1/PWMy+1（y=1，3，5）。

●内嵌一个 PWM 电路，包含非对称/对称波形发生器和空间矢量状态机。

与单比较单元一样，全比较单元的工作需要由通用定时器来配合，而且它只能由通用定时器 1 提供时基。此时的通用定时器 1 可以处于 6 种计数模式中的任何一种，但当通用定时器 1 处于定向增/减计数模式时，全比较单元的比较输出不发生变化。三个全比较单元共用通用定时器 1 作为时基，它们的 PWM 输出波形具有同样的调制周期（频率），但它们可有各自的脉宽变化规律。

全比较单元的六个输出引脚是成对工作的，它们的输出极性正好是反向，即引脚 CMPy/PWMy（y=1，3，5）为高电平时，引脚 CMPy+1/PWMy+1 一定是低电平（不考虑死区）。这是为桥式电路所设计，当上桥臂开通时，下桥臂一定要关闭，否则将发生短路直通。

全比较单元有两种工作模式：比较模式和 PWM 模式。这两种模式的工作过程与通用定时器的比较输出操作基本一样。有所不同的是：

●比较模式的输出极性分为保持、复位、置位和触发（跳变），由全比较动作控制寄存器设置。

●PWM 模式的输出极性与通用定时器的一样，分为强制低、低有效、高有效和强制高，也是由全比较动作控制寄存器设置。但 PWM 模式的波形发生是经过一个内嵌 PWM 电路产生，在其中有一个死区产生电路和一个空间矢量状态机。死区产生电路用于上、下桥臂状态转换时（即输出发生跳变）增加一个无信号的死区时间，确保不发生短路直通相象。另外，空间矢量是一种新型的脉宽调制方案，有了空间矢量状态机将为这种调制方案的实现提供极大的方便。因此，PWM 模式对于桥式电路的实际应用提供一个更灵活的空间。

与单比较单元一样，全比较单元在开始工作前，首先要设置比较控制寄存器 COMCON 的

相应位，以及全比较动作控制寄存器 ACTR 以确定比较输出引脚的极性等参数。全比较动作控制寄存器 ACTR 各位的组成见表 3.5 所示，各位的定义与说明见表 3.8。全比较单元的工作原理与单比较单元的完全一样。

表 3.8 全比较动作控制寄存器 ACTR 各位的定义与说明

位	定义		说明	
Bits15	SVRDIR	0	空间矢量按顺时针方向（正）	
		1	空间矢量按逆时针方向（负）	
Bits14-12	D2：1：0		空间矢量状态位（8 个状态）	
Bits11-10	CMP6ACT1	CMP6ACT0	比较模式	PWM 模式
	0	0	保持	强制低
	0	1	复位	低有效
	1	0	置位	高有效
	1	1	触发	强制高
Bits9-8	CMP5ACT1	CMP5ACT0	比较模式	PWM 模式
	0	0	保持	强制低
	0	1	复位	低有效
	1	0	置位	高有效
	1	1	触发	强制高
Bits7-6	CMP4ACT1	CMP4ACT0	比较模式	PWM 模式
	0	0	保持	强制低
	0	1	复位	低有效
	1	0	置位	高有效
	1	1	触发	强制高
Bits5-4	CMP3ACT1	CMP3ACT0	比较模式	PWM 模式
	0	0	保持	强制低
	0	1	复位	低有效
	1	0	置位	高有效
	1	1	触发	强制高
Bits3-2	CMP2ACT1	CMP2ACT0	比较模式	PWM 模式
	0	0	保持	强制低
	0	1	复位	低有效
	1	0	置位	高有效
	1	1	触发	强制高
Bits1-0	CMP1ACT1	CMP1ACT0	比较模式	PWM 模式
	0	0	保持	强制低
	0	1	复位	低有效
	1	0	置位	高有效
	1	1	触发	强制高

●首先选择通用定时器 T1 作为全比较单元的时基，并设置它的计数模式，如果要产生连续 PWM 波形，计数模式设置为连续增或连续增/减计数模式，否则设置为单增或单增/减计数模式。

●根据调制频率设置相应的定时周期寄存器 T1PR 的值，初始化计数寄存器 T1CNT 的值，然后启动定时器。

●按照脉宽的变化规律，设置当前的全比较寄存器 CMPRx（x=1，2，3）值。计数寄存器 T1CNT 按照计数模式进行计数，并与全比较寄存器 CMPRx 的值进行比较，若相等将发生全比较匹配事件，并在延迟 2 个 CPU 时钟后在中断标志寄存器 EVIFRA 的 CMPxINT 位上置 1，同时使输出引脚 CMPy/PWMy 和 CMPy+1/PWMy+1（y=1，3，5）按设定的极性发生变化。

●与上类似，计数寄存器 T1CNT 也同时与定时周期寄存器 T1PR 值进行比较，若相等将发生定时周期匹配事件，从而引发与通用定时器完全一致的相关操作。按照在比较控制寄存器 COMCON 设置的全比较寄存器重载条件，为下一周期准备一个新的脉冲宽度。如此循环，得到需要的 PWM 波形。

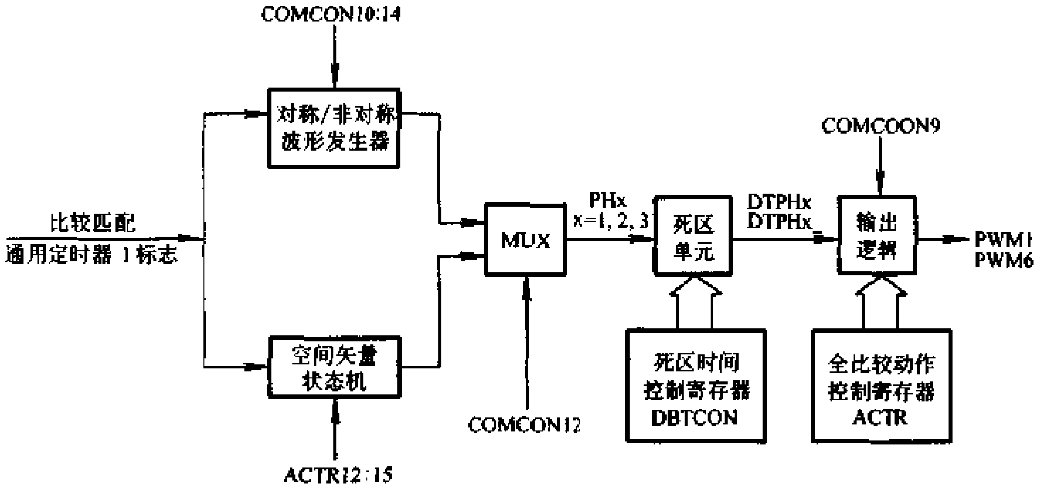


图 3.13　PWM 电路结构示意图

3. PWM 电路　由于全比较单元是为桥式电路而设，它的 PWM 波形在实际应用时有其特点。针对这些特点，在全比较单元内嵌了一个 PWM 电路。下面首先介绍 PWM 电路结构示意图，然后介绍它的两个功能模块：死区产生电路和空间矢量状态机。图 3.13 是 PWM 电路的结构示意图，它包括以下功能单元：

●非对称/对称波形发生器；

●死区产生电路；

●空间矢量状态机；

●输出逻辑。

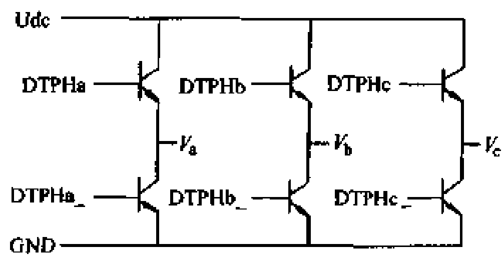


图 3.14　三相桥式电路

全比较单元是为图 3.14 所示的桥式电路而设计的。这种桥式电路是 DC-AC 变换控制系统中最常用的一种电路形式，广泛应用于电机控制、逆变器等大部分电力电子的控制系统中。桥式电路在应用时最关键的是要保证每组桥臂不能发生短路直通现象。这样一来，从输出逻

辑上要保证当上桥臂开通时，下桥臂必须要关断，反之亦然。由于桥式电路由开关器件构成，它们的开通或关断需要一定的时间，因此，在上、下桥臂状态转换时，开关器件会有一个共处放大状态的交叉区间，从而导致短路直通现象的发生。为了避免这一点，就需要在上、下桥臂状态转换时插入一个无信号的死区时间，即确保先完全关断再开通。短路直通与死区如图 3.15 所示。

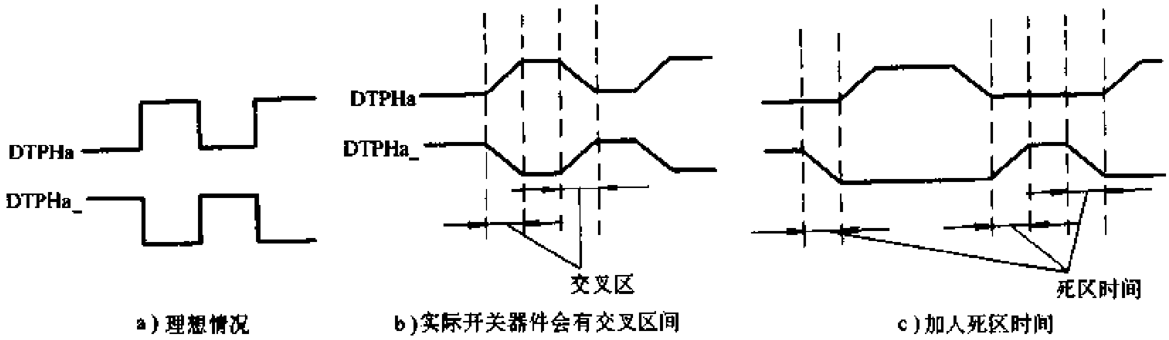


图 3.15 短路直通与死区

4. 死区产生电路 为了自动地在每对全比较输出单元加入无信号的死区，PWM 电路中有一个死区产生电路，如图 3.16 所示。该电路主要是对死区控制寄存器 DBTCON 进行读写。死区控制寄存器见表 3.5，其各位定义及说明见表 3.9。

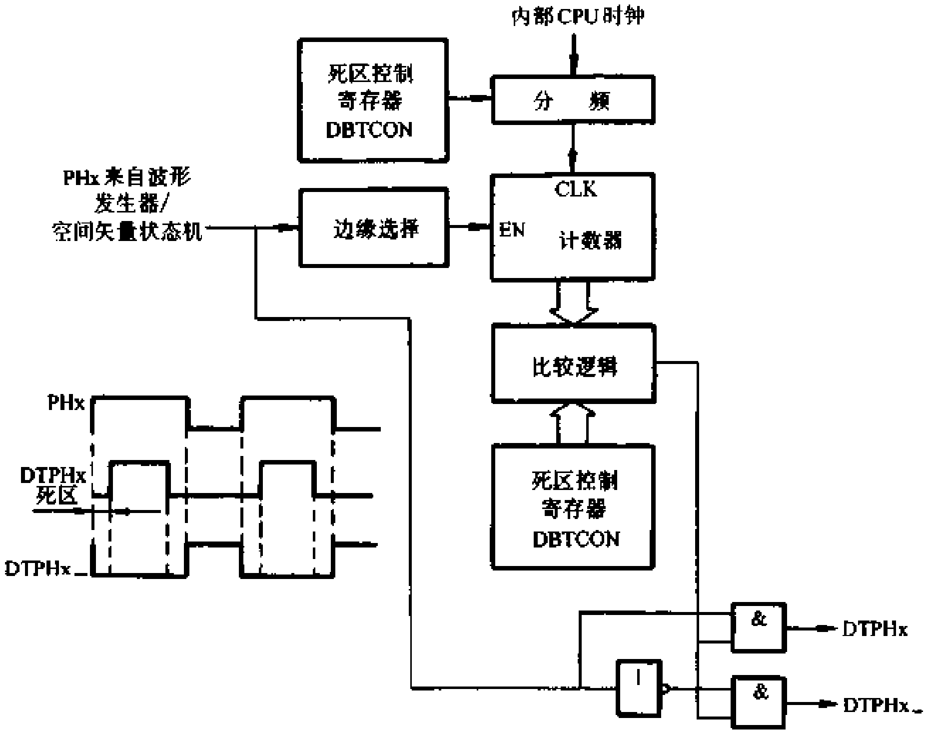


图 3.16 死区产生电路

为了使用死区产生电路，首先要设置死区控制寄存器 DBTCON 的 EDBTx（x=1，2，3）为 1；然后，根据需要的死区时间 δ（多少个 CPU 时钟周期）设置分频系数 k_s（DBTCON 的 DBTPS1：0）和死区定时器的周期值 DBTPR（DBTCON 的 DBT7：0），分频系数乘以死区定时器的周期值再乘以 CPU 时钟周期就是死区时间，即

$$\delta = k_{\delta} \times \mathrm{DBTPR}/F_c \text{（}F_c\text{ 是 CPU 时钟频率）} \tag{3.4}$$

5. 空间矢量状态机　空间矢量是一种针对三相交流电路的新型调制技术，与正弦波调制法相比，可更有效地利用电源电压，减少电流谐波失真。下面介绍它的工作原理。

表 3.9　死区控制寄存器 DBTCON 各位定义及说明

位	定义		说明	
Bits15-8	DBT7：0		死区定时器周期值 DBTPR	
Bits7	EDBT3	0	第三个全比较单元不使用死区电路	PWM6/CMP6
		1	第三个全比较单元使用死区电路	PWM5/CMP5
Bits6	EDBT2	0	第二个全比较单元不使用死区电路	PWM4/CMP4
		1	第二个全比较单元使用死区电路	PWM3/CMP3
Bits5	EDBT1	0	第一个全比较单元不使用死区电路	PWM2/CMP2
		1	第一个全比较单元使用死区电路	PWM1/CMP1
Bits4-3	DBTPS1	DBTPS0	分频系数（F_c 是 CPU 时钟频率）	
	0	0	$F_c/1$	
	0	1	$F_c/2$	
	1	0	$F_c/4$	
	1	1	$F_c/8$	
Bits2-0	保留			

对于图 3.14 的三相桥式电路，假定它的负载是三相平衡负载，U_{dc}是直流侧电压，V_a、V_b 和 V_c 是输出的三相相电压，V_{ab}、V_{bc}和 V_{ca}是输出的三相线电压。不失一般性，可假定它们之间有如下关系：

$$\begin{cases} V_a = V_m \sin\omega t \\ V_b = V_m \sin(\omega t + 120°) \\ V_c = V_m \sin(\omega t - 120°) \end{cases} \qquad \begin{cases} V_{ab} = \sqrt{3} V_m \sin(\omega t + 30°) \\ V_{bc} = \sqrt{3} V_m \sin(\omega t + 120° + 30°) \\ V_{ca} = \sqrt{3} V_m \sin(\omega t - 120° + 30°) \end{cases} \tag{3.5}$$

三相桥式电路的目的是按一定规律控制三对桥臂晶体管的通、断，将直流侧电压 $E=V_{dc}$ 变为三相正弦电压 V_a、V_b 和 V_c 输出。前已说明，桥式电路的上、下桥臂晶体管的通断状态是互为反向的。因此，三相桥式电路各桥臂的通断状态只有 8 种可能，如表 3.10 所示。其中 V_{ab}、V_{bc}、V_{ca}和 V_a、V_b、V_c 与桥臂的通断状态有如下关系：

$$\begin{bmatrix} V_{ab} \\ V_{bc} \\ V_{ca} \end{bmatrix} = E \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} \qquad \begin{bmatrix} V_a \\ V_b \\ V_c \end{bmatrix} = \frac{1}{3} E \begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} \tag{3.6}$$

为了更方便地分析三相交流电路，常常将三相坐标系变换为二相坐标系（详情参见有关电机模型及坐标变换方面的书），即

$$\begin{bmatrix} V_\alpha \\ V_\beta \end{bmatrix} = \sqrt{2/3} \begin{bmatrix} 1 & -1/2 & -1/2 \\ 0 & \sqrt{3}/2 & -\sqrt{3}/2 \end{bmatrix} \begin{bmatrix} V_a \\ V_b \\ V_c \end{bmatrix} \tag{3.7a}$$

$$\begin{bmatrix} V_a \\ V_b \\ V_c \end{bmatrix} = \sqrt{2/3}\begin{bmatrix} 1 & 0 \\ -1/2 & \sqrt{3}/2 \\ -1/2 & -\sqrt{3}/2 \end{bmatrix}\begin{bmatrix} V_\alpha \\ V_\beta \end{bmatrix} \tag{3.7b}$$

表 3.10 三相桥式电路各桥臂的通断状态与输出电压

a	b	c	$V_a(E)$	$V_b(E)$	$V_c(E)$	$V_{ab}(E)$	$V_{bc}(E)$	$V_{ca}(E)$	$V_\alpha(E)$	$V_\beta(E)$	$V_\alpha+V_\beta(E)$
0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	$-1/3$	$-1/3$	$2/3$	0	-1	1	$-1/\sqrt{6}$	$-1/\sqrt{2}$	$\sqrt{2/3}\angle 240°$
0	1	0	$-1/3$	$2/3$	$-1/3$	-1	1	0	$-1/\sqrt{6}$	$1/\sqrt{2}$	$\sqrt{2/3}\angle 120°$
0	1	1	$-2/3$	$1/3$	$1/3$	-1	0	1	$-\sqrt{2/3}$	0	$\sqrt{2/3}\angle 180°$
1	0	0	$2/3$	$-1/3$	$-1/3$	1	0	-1	$\sqrt{2/3}$	0	$\sqrt{2/3}\angle 0°$
1	0	1	$1/3$	$-2/3$	$1/3$	1	-1	0	$1/\sqrt{6}$	$-1/\sqrt{2}$	$\sqrt{2/3}\angle 300°$
1	1	0	$1/3$	$1/3$	$-2/3$	0	1	-1	$1/\sqrt{6}$	$1/\sqrt{2}$	$\sqrt{2/3}\angle 60°$
1	1	1	0	0	0	0	0	0	0	0	0

由于 $V_a+V_b+V_c=0$，所以 V_a、V_b、V_c 与 V_α、V_β 之间可以按上面两个公式进行一一变换。按照这个关系可分别求出表 3.10 中 8 种状态三相电压矢量 V_a、V_b、V_c 对应的 V_α 与 V_β，然后将得到的 V_α 与 V_β 合成为一个电压矢量（称之为空间矢量）在二相坐标系中表示出来，如图 3.17 所示，其中（0，0，0）与（1，1，1）两种状态为无效状态对应于原点，其余 6 种状态为有效状态按 60°均匀分布在一个圆周上。

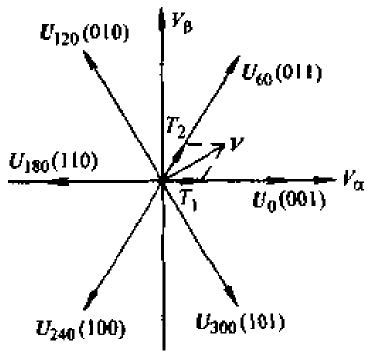


图 3.17 空间矢量图

用二相坐标系代替三相坐标系，为分析三相平衡式电路带来了很大的方便。因为，在二相坐标系中的一个电压矢量 $\boldsymbol{V}$，与某时刻的三相电压 V_a、V_b、V_c 是一一对应的，即将 $\boldsymbol{V}$ 分解为 V_α 与 V_β，然后按照上面的公式（3.7b）就可得到该时刻的三相电压 V_a、V_b、V_c。电压矢量 $\boldsymbol{V}$ 在二相坐标系的旋转频率就是三相电压的频率。结合空间矢量图可以得到如下结论：任意时刻的三相电压 V_a、V_b、V_c（即电压矢量 $\boldsymbol{V}$）可由二个相邻的空间矢量合成而成，当电压矢量 $\boldsymbol{V}$ 沿着逆时针或顺时针方向旋转时，空间矢量由一个有效状态转移到另一个有效状态，从而产生连续的三相电压。这也是称之为空间矢量状态机的原因。电压矢量 $\boldsymbol{V}$ 与二个相邻有效状态的关系可用下述公式表示：

$$V=\frac{T_x}{T_p}\boldsymbol{U}_x+\frac{T_{x\pm60}}{T_p}\boldsymbol{U}_{x\pm60}+\frac{T_0}{T_p}\ (\boldsymbol{O}_{000}\ \text{or}\ \ \boldsymbol{O}_{111}) \tag{3.8}$$

式中，T_p 是调制周期；T_x 是空间矢量 $\boldsymbol{U}_x$ 持续的时间；$T_{x\pm60}$是空间矢量 $\boldsymbol{U}_{x\pm60}$持续的时间；$T_0=T_p-T_x-T_{x\pm60}$是零矢量 $\boldsymbol{O}_{000}$或 $\boldsymbol{O}_{111}$持续的时间；T_x 与 $T_{x\pm60}$可以事先算好做成表存到存储器中。

经过前面的讨论可以得知，以空间矢量的方法产生 PWM 波形，需要一个定时周期寄存器 T1PR 来控制调制周期（T_p），需要二个比较寄存器 CMPR1 和 CMPR2 来控制有效状态 U_x 和 $U_{x\pm60}$ 的持续时间（T_x 和 $T_{x\pm60}$）。因此，它的步骤如下：

●按照输出极性的要求，设置全比较动作控制寄存器 ACTR 的 CMPyACT1：0（y=1，2，3，4，5，6）。不失一般性，将它们均设为“高有效”。

●将比较控制寄存器 COMCON 的 CEnable、SVEnable、FCOMPOE 置为 1；将全比较寄存器 CMPR 和全比较动作控制寄存器 ACTR 的重载条件设为下溢事件。

●置通用定时器 1 的计数模式为连续增/减模式；将定时周期寄存器 T1PR 的值设为 $T_p/2$。

●确定当前空间矢量 U_x 对应的状态编码（表 3.1 中的 cba），并将它写到全比较动作控制寄存器 ACTR 的 D2：1：0 上。

●根据当前空间矢量是逆时针转还是顺时针转，设置全比较动作控制寄存器 ACTR 的 SVRDIR，以此确定与 U_x 组合的空间矢量是 U_{x+60} 还是 U_{x-60}。

●如果是逆时针转，将 $T_x/2$ 写入 CMPR1，将（$T_x/2+T_{x\pm60}/2$）写入 CMPR2 中；如果是顺时针转，将 $T_{x\pm60}/2$ 写入 CMPR1，将（$T_x/2+T_{x\pm60}/2$）写入 CMPR2 中。这样在第一次全比较匹配事件（CMPR1）前，三个全比较单元输出状态 U_x；第一次全比较匹配发生后，三个全比较单元输出变为状态 $U_{x\pm60}$，一直到第二次全比较匹配事件（CMPR2）发生；随后，三个全比较单元输出变为状态 O_{000} 或 O_{111}；由于连续增减模式产生对称 PWM 波形，所以状态 O_{000} 或 O_{111} 一直延续到第三次全比较匹配事件（CMPR2）发生；接着三个全比较单元输出变为状态 $U_{x\pm60}$，该状态继续保持到第四次全比较匹配事件（CMPR1）发生；最后，三个全比较单元输出状态回到 U_x，一直到本次周期结束。图 3.18 说明了这个过程。

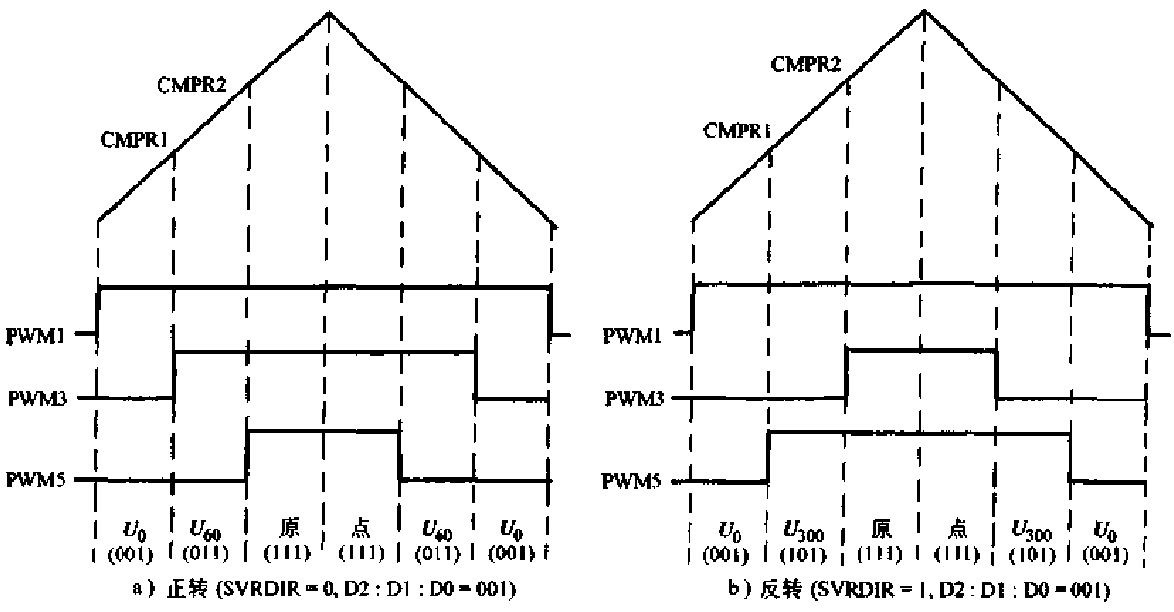


图 3.18　空间矢量 PWM 输出波形

●根据死区时间设置死区控制寄存器 DBTCON。

空间矢量 PWM 输出的生成中只用到了两个全比较寄存器。为了保证空间矢量状态机正常运转，这两个全比较寄存器的值必须满足：

$$\text{CMP1} \leqslant \text{CMP2} \leqslant \text{T1PR} \tag{3.9}$$

如果 CMPR1＝CMPR2＝0，则三组桥臂处于关断状态。此外，要注意空间矢量状态机有附加延时，即在空间矢量 PWM 模式中全比较输出跳变会被延时两个 CPU 时钟周期。

对于没有用于空间矢量 PWM 产生的第三个全比较寄存器，还可以被使用，因为它也一直也在和通用定时器 1 进行比较。如果它发生比较匹配，并且相应的比较中断标志未被屏蔽且同组中没有其它未被屏蔽的更高优先级的中断被挂起，该标志将被置位并且发出中断请求。因此，没有用于空间矢量 PWM 输出形成的那个比较寄存器仍可以用于特定应用中定时事件的发生。

3.1.3 捕获单元

捕获单元是一种输入设备，用于捕获引脚上电平的变化并记录它发生的时刻。普通的微处理器能做到这一点，但需要由 CPU 完成判断和记录工作，占用了 CPU 的资源。另外，对于二次间隔很短的跳变（微秒级）的捕获，普通的微处理器就显得力不从心。DSP 控制器的捕获单元不需要占用 CPU 的资源，与 CPU 并行工作。它有二级 FIFO 堆栈缓冲器，对于二次间隔很短的跳变的捕获得心应手。下面详细介绍它的结构、原理及使用方法。

DSP 控制器共有四个捕获单元，图 3.19 是它的结构示意图。捕获单元包含：

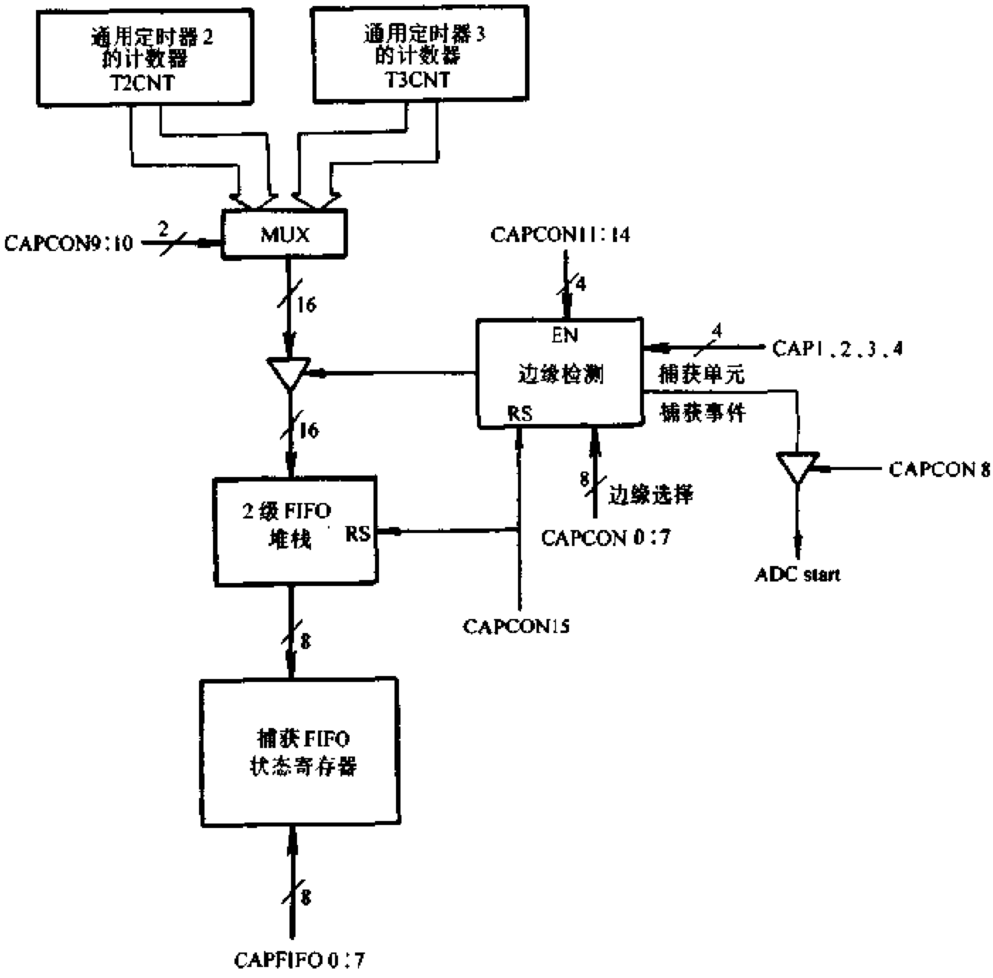


图 3.19 捕获单元结构示意图

●1 个 16 位捕获控制寄存器 CAPCON。

●1 个 16 位捕获 FIFO 状态寄存器 CAPFIFO（其高 8 位只读，而低 8 位只写）。

●可选择通用定时器 2 或 3 作为时基。

●4 个 16 位二级 FIFO 堆栈，对应于每一个捕获单元。

●4 个施密特触发式的捕获输入引脚 CAP1、CAP2、CAP3 和 CAP4，对应于每一个捕获单元。其中引脚 CAP1 与 CAP2 是功能复用的，也是正交编码脉冲电路（QEP）的两个输入引脚。

捕获单元不停地检测捕获输入引脚的跳变，为了可靠地捕获到引脚上跳变信号，该跳变信号至少需保持两个 CPU 时钟的时间。这个跳变可以是上升沿、下降沿或者双沿，由捕获控制寄存器 CAPCON 来规定。捕获单元 1 和 2 共用一个通用定时器作为时基，捕获单元 3 和 4 共用一个通用定时器作为时基。这个通用定时器可以是 2 或 3。通用定时器用作捕获单元的时基时，并不影响它原来的功能，它仍然可以实现通用定时器的“定时”或“比较/PWM 输出”。

捕获控制寄存器 CAPCON 各位的组成见表 3.11，表 3.12 是它各位的定义与说明。捕获 FIFO 状态寄存器 CAPFIFO 各位的组成见表 3.11，表 3.13 是它各位的定义与说明。

表 3.11　捕获控制寄存器 CAPCON 和捕获 FIFO 状态寄存器 CAPFIFO

地址	寄存器	位数							
7420h	CAPCON	15	14	13	12	11	10	9	8
		CAPRES	CAPQEPN		CAP3EN	CAP4EN	CAP34TSEL	CAP12TSEL	CAP4TOADC
		RW-0	RW-0		RW-0	RW-0	RW-0	RW-0	RW-0
		7	6	5	4	3	2	1	0
		CAP1EDGE		CAP2EDGE		CAP3EDGE		CAP4EDGE	
		RW-0		RW-0		RW-0		RW-0	
7422h	CAPFIFO	15	14	13	12	11	10	9	8
		CAP4FIFO		CAP3FIFO		CAP2FIFO		CAP1FIFO	
		R-0		R-0		R-0		R-0	
		7	6	5	4	3	2	1	0
		CAPFIFO15	CAPFIFO14	CAPFIFO13	CAPFIFO12	CAPFIFO11	CAPFIFO10	CAPFIFO9	CAPFIFO8
		W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0

表 3.12　捕获控制寄存器 CAPCON 各位的定义与说明

位	定义		说明
Bits15	CAPRES	0	清所有捕获单元和 QEP 电路的寄存器为 0
		1	无作用
Bits14-13	CAPQEPN1:0		
	0	0	关闭捕获单元 1、2 和 QEP 电路，FIFO 堆栈内容不变
	0	1	开启捕获单元 1、2，但关闭 QEP 电路
	1	0	保留
	1	1	开启 QEP 电路，但关闭捕获单元 1、2

（续）

位	定义		说明
Bits12	CAP3EN	0	关闭捕获单元 3，FIFO 堆栈内容不变
		1	开启捕获单元 3
Bits11	CAP4EN	0	关闭捕获单元 4，FIFO 堆栈内容不变
		1	开启捕获单元 4
Bits10	CAP34TSEL	0	选通用定时器 2 作为捕获单元 3、4 的时基
		1	选通用定时器 3 作为捕获单元 3、4 的时基
Bits9	CAP12TSEL	0	选通用定时器 2 作为捕获单元 1、2 的时基
		1	选通用定时器 3 作为捕获单元 1、2 的时基
Bits8	CAP4TOADC	0	无作用
		1	当 CAP4INT 标志置 1 时，启动 A/D 转换
Bits7-6	CAP1EDGE1：0		捕获单元 1 的边沿检测
	0	0	不检测
	0	1	检测上升沿
	1	0	检测下降沿
	1	1	检测双沿
Bits5-4	CAP2EDGE1：0		捕获单元 2 的边沿检测
	0	0	不检测
	0	1	检测上升沿
	1	0	检测下降沿
	1	1	检测双沿
Bits3-2	CAP3EDGE1：0		捕获单元 3 的边沿检测
	0	0	不检测
	0	1	检测上升沿
	1	0	检测下降沿
	1	1	检测双沿
Bits1-0	CAP4EDGE1：0		捕获单元 4 的边沿检测
	0	0	不检测
	0	1	检测上升沿
	1	0	检测下降沿
	1	1	检测双沿

当捕获输入引脚发生跳变时，捕获单元将该时刻的时基的计数寄存器 TxCNT（x=2 或 3）的值装入相应的 FIFO 堆栈中。捕获单元的 FIFO 堆栈是二级先进先出的堆栈，可以装入两个值，第三个值装入时，会将第一个值挤出堆栈。FIFO 堆栈的状态可以从捕获 FIFO 状态寄存器 CAPFIFO 中得知：

第一次捕获：当捕获单元的输入引脚出现指定的跳变时，捕获单元就将捕获到的时基计

数寄存器的值写入空栈的顶层寄存器（FIFOx，x=1，2，3，4）中。与此同时，FIFO 状态寄存器 CAPFIFO 相应的状态位 CAPxFIFO1：0（x=1，2，3，4）被置成（01）。如果在下一次捕获前对 FIFO 堆栈顶层寄存器进行了读访问，则状态位 CAPxFIFO1：0 被复位为（00），FIFO 堆栈又成为空栈。

表 3.13 捕获 FIFO 状态寄存器 CAPFIFO 各位的定义与说明

位	定义		说明
Bits15-14	CAP4FIFO1：0		CAP4FIFO 的状态，只读
	0	0	空
	0	1	1 个值
	1	0	2 个值
	1	1	2 个值，但前面有值被挤出堆栈
Bits13-12	CAP3FIFO1：0		CAP3FIFO 的状态，只读
	0	0	空
	0	1	1 个值
	1	0	2 个值
	1	1	2 个值，但前面有值被挤出堆栈
Bits11-10	CAP2FIFO1：0		CAP2FIFO 的状态，只读
	0	0	空
	0	1	1 个值
	1	0	2 个值
	1	1	2 个值，但前面有值被挤出堆栈
Bits9-8	CAP1FIFO1：0		CAP1FIFO 的状态，只读
	0	0	空
Bits9-8	0	1	1 个值
	1	0	2 个值
	1	1	2 个值，但前面有值被挤出堆栈
Bits7	CAPFIFO15（只写）	0	无作用
		1	清本寄存器的 Bits15
Bits6	CAPFIFO14（只写）	0	无作用
		1	清本寄存器的 Bits14
Bits5	CAPFIFO13（只写）	0	无作用
		1	清本寄存器的 Bits13
Bits4	CAPFIFO12（只写）	0	无作用
		1	清本寄存器的 Bits12
Bits3	CAPFIFO11（只写）	0	无作用
		1	清本寄存器的 Bits11
Bits2	CAPFIFO10（只写）	0	无作用
		1	清本寄存器的 Bits10
Bits1	CAPFIFO9（只写）	0	无作用
		1	清本寄存器的 Bits9
Bits0	CAPFIFO8（只写）	0	无作用
		1	清本寄存器的 Bits8

第二次捕获：如果在 FIFO 堆栈顶层寄存器被读取之前产生了另一次捕获，则新捕获的时基计数寄存器的值送至底层寄存器。与此同时，相应的状态位 CAPxFIFO1：0 置成（10）。若在再一次捕获发生前，对 FIFO 堆栈顶层寄存器 FIFOx 进行读访问，则顶层寄存器中的旧值被读出，底层寄存器中的新值被弹入顶层寄存器，并且相应的状态位 CAPxFIFO1：0 将置成（01）。若此时再发生捕获，FIFO 堆栈不会溢出。

第三次捕获：如果发生了二次捕获而又未进行顶层寄存器 FIFOx 的读访问，此时若再发生捕获，则位于堆栈顶层寄存器中的旧值将被挤出丢弃，而堆栈底层寄存器中的值将被弹入顶层寄存器，新捕获的时基计数寄存器的值将被写入底层寄存器，并且状态位 CAPxFIFO1：0 置成（11），以表明已经丢弃了一个或多个捕获计数器值。

当捕获单元捕获到输入引脚的跳变时，除了将时基计数寄存器 TxCNT（x=2 或 3）的值装入相应的 FIFO 堆栈外，还将产生捕获事件，在中断标志寄存器 EVIFRC 的 CAPxINT（x

=1，2，3，4）上置 1。若是捕获单元 4 产生了捕获事件，还可以启动 A/D 转换，这为外部事件与 A/D 转换同步提供了途径。

为使捕获单元正常工作，应完成以下寄存器配置：

●初始化 CAPFIFO，将适当的状态位清零。

●设置时基通用定时器的控制寄存器 T2CON 或 T3CON，同时确定它的计数模式。

●如有必要，应设置相关的通用定时器比较寄存器或者通用定时器周期寄存器。

●设置捕获控制寄存器 CAPCON。

3.1.4 正交编码脉冲电路

许多运动控制系统都需要有正反两个方向的运动，为了对位置、速度进行控制，必须要测试出当前的运动方向。图 3.20 是正交编码脉冲（QEP）的原理图，图中的方孔可以透过光电信号，两排方孔在位置上差 90°相位。这样，通过两组脉冲的相位（上升沿的前后顺序）可以判断出运动的方向，通过记录脉冲的个数可以确定具体的位置，通过记录确定周期的脉冲个数可以计算出运动的速度。对于旋转运动，可以采用类似的光电码盘。

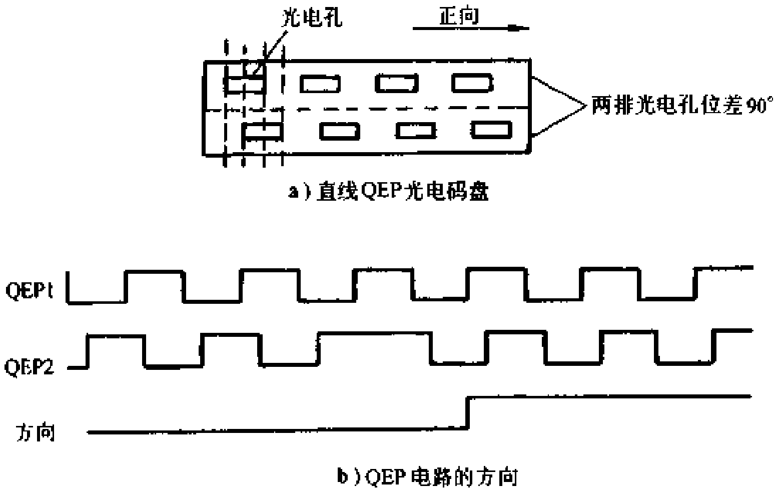


图 3.20 正交编码脉冲的原理图

DSP 控制器内置正交编码脉冲电路，可自动识别由外部引脚上输入的正交编码脉冲的方向，记录脉冲的个数，这为运动控制、伺服控制的实现提供了方便。图 3.21 是正交编码脉冲电路结构图。

正交编码脉冲电路的输入引脚 CAP1/QEP1 和 CAP2/QEP2 与捕获单元 1、2 的输入引脚复用，由捕获控制器 CAPCON 的 CAPQEPN1：0 来设定。正交编码脉冲电路中的译码逻辑模块将自动分辨出正交编码脉冲的方向，并将该方向和编码脉冲组成的时钟送至通用定时器 2 或通用定时器 3。具体使用步骤如下：

●选择时基。通用定时器 2 或通用定时器 3 可以单独作为正交编码脉冲电路的时基，也可以级联为 32 位定时器作为正交编码脉冲电路的时基。作为正交编码脉冲电路的时基时，它们的计数模式（TxCON 的 TMODE2：1：0，x=2 或 3）必须设置为定向增/减计数模式，它们的时钟源（TxCONT 的 TCLK1：0）必须选择正交编码脉冲电路。

●将正交光电码盘的两路脉冲信号接至输入引脚 CAP1/QEP1 和 CAP2/QEP2 上。设定

捕获控制器 CAPCON 的 CAPQEPN1：0＝11。

●这样一来，通用定时控制寄存器 GPTCON 的 TxSTAT 反映了正交光电码盘脉冲信号的方向（此时，外部的方向引脚 TMRDIR 不起作用），计数寄存器 TxCNT 的值反映了正交光电码盘对应的位置与速度。

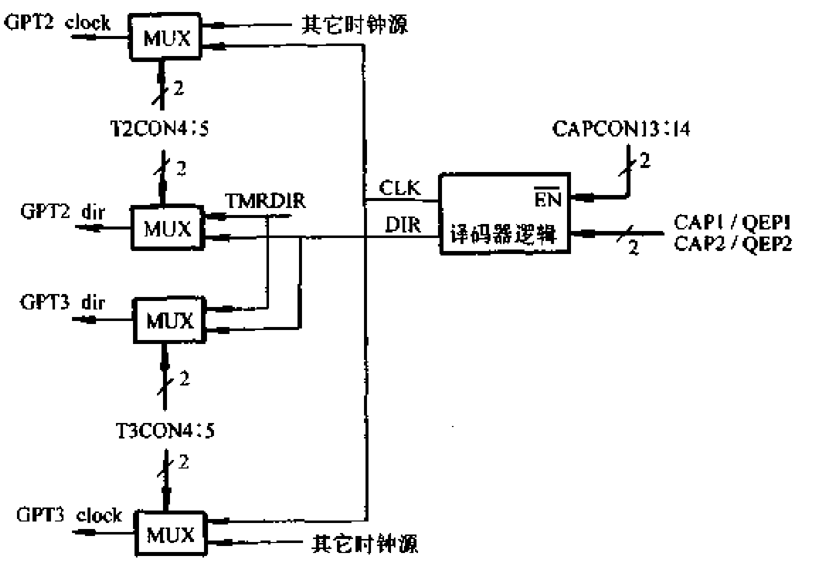


图 3.21　正交编码脉冲电路结构图

当通用定时器 2、3 或级联为 32 位定时器作为正交编码脉冲电路的时基时，其比较匹配、周期匹配、上溢、下溢事件照常产生，中断标志也相应置位，但其比较输出引脚不产生跳变。另外，如果通用定时器 2 或通用定时器 3 同时还是单比较单元的时基，该单比较单元的输出引脚也不产生跳变。

正交解码脉冲电路的计数操作过程，与通用定时器定向增/减计数模式的工作过程完全一样，详情参见 3.1.1。

3.2　模/数转换模块

在自动控制系统中，被控制或被测量的对象，如温度、压力、流量、速度等都是连续变化的物理量，这种连续变化的物理量是指在时间上和数值上都连续变化的量，也就是我们常说的模拟量。这种模拟量的数值和极性可以由传感器进行测量，通常以模拟电压或电流的形式输出。当用单片机参与测量时，必须将它们转变为数字量才能被单片机接受。能够将模拟量转换为数字量的器件称为模/数转换器，简称 ADC 或 A/D。单片机计算结果是数字量，不能直接用于控制执行部件，需要先把它转换为模拟量才能用于控制。这种能将数字量转换为模拟量的器件称为数/模转换器，简称 DAC 或 D/A。早期的单片机只提供数字 I/O 端口，只能直接处理数字信号，需处理模拟信号时必须外接模数转换芯片。随着芯片集成技术的提高，许多单片机已将模数转换的功能集成到片内外设中，使其不需增加其它外围芯片就能直接处理模拟信号，简化单片机系统的设计，拓宽了它的应用领域。普通 MCS-51 系列单片机没有模数转换模块，但某些子系列具有模数转换模块，如 8XC51GX 子系列有 8 个模拟输入通道，8XC51SL 系列有 4 个模拟输入通道。MCS-96 系列单片机带有一个 10 位的 A/D 转换模块和多路模拟开关电路，可采集最多 8 路模拟量数据。许多 MOTOROLA 单片机也具有 A/D 转换

功能。本书介绍的DSP控制器内带两个10位模数转换器共16个模拟输入通道，可并行处理2路模拟输入量，每个模数转换器的最快转换时间是6.6μs。

3.2.1 结构概述

DSP控制器的模数转换模块包括两个独立的模/数转换器，每个模/数转换器带有一个内部采样/保持电路和一个10位双积分型的转换器。每个模/数转换器可接8个模拟输入通道，这8个模拟输入通过多路转换开关（MUX）提供给每个模/数转换器。每个模/数转换器每次只能转换1个模拟输入，但两个独立的模/数转换器可以同时转换2个模拟输入。DSP控制器的模/数转换模块的结构如图3.22所示。

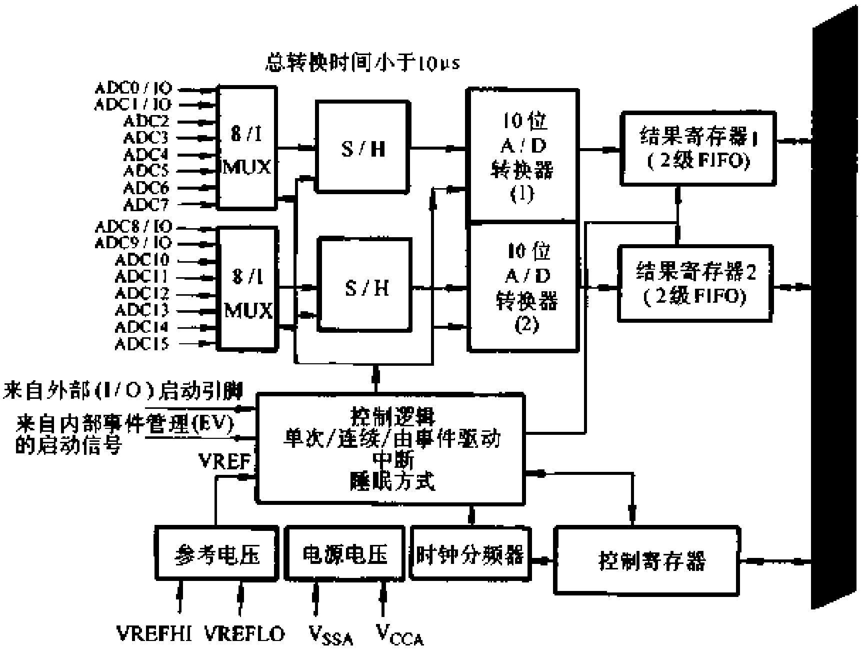


图3.22 模数转换模块方框图

与模/数转换模块有关的引脚：

● ADC0～ADC15：16个模拟输入通道。引脚ADC0～ADC7属于第一个模/数转换器，引脚ADC8-ADC15属于第二个模/数转换器。其中ADC0、ADC1、ADC8、ADC9与数字I/O复用，即可作模拟输入也可作数字I/O，由I/O多路控制寄存器OCRA指定。另外，要注意这四个引脚的转换精度低于其它模拟输入引脚。

● V_{REFHI}和V_{REFLO}：模拟基准电压引脚。模数转换的结果是相对于基准电压而言的，为了得到较高的转换精度，模拟基准电压必须要稳定而且有较小的纹波。模/数转换模块的基准电压由外部提供，通常V_{REFLO}连接到模拟地，V_{REFHI}接到小于或等于5V的基准电压上。

● V_{CCA}和V_{SSA}：模拟电源引脚。V_{CCA}和V_{SSA}分别接到5V直流电源和模拟地上。为了减少干扰的影响，最好将模拟地与数字地隔离。

与模/数转换模块相关的寄存器：

●每个模/数转换器都有一个2级深的FIFO结果寄存器ADCFIFOx（x=1，2），它可以先后存放二次的转换结果，当第三个结果进来时将把第一个结果挤出。由于采用2级深的FIFO的存储方式，有了缓冲的读取时间，因此对于连续快速的模拟数据采集非常有用。

●两个模/数转换控制寄存器。其中，第一个控制寄存器 ADCTRL1 规定采用什么样的信号启动模/数转换，指出模/数转换是否已完成，设置对哪路通道的模拟输入进行转换等；第二个控制寄存器 ADCTRL2 设置模/数转换时钟的分频系数，给出 2 级深 FIFO 结果寄存器的状态等。两个模/数转换控制寄存器及 2 级深 FIFO 结果寄存器说明见表 3.14，两个模/数转换控制寄存器各位定义与说明见表 3.15、表 3.16。

表 3.14 模/数转换控制寄存器及 2 级深 FIFO 结果寄存器说明

地址	寄存器	位数							
7032h	ADCTRL1	15	14	13	12	11	10	9	8
		Suspend	Suspend	ADCIMSTART	ADC2EN	ADC1EN	ADCCONRUN	ADCINTEN	ADCINTFLAG
		RW-0	RW-0	RW-0	SRW-0	SRW-0	SRW-0	SRW-0	RW-0
		7	6	5	4	3	2	1	0
		ADCEOC	ADC2CHSEL			ADC1CHSEL			ADCSOC
		R-0	SRW-0			SRW-0			SRW-0
7034h	ADCTRL2	15	14	13	12	11	10	9	8
		Reserved					ADCEVSOC	ADCECTSOC	Reserved
							SRW-0	SRW-0	
		7	6	5	4	3	2	1	0
		ADCFIFO2		Reserved	ADCFIFO1		ADCPSCALE		
		R-0			R-0		SRW-0		
7036h 7038h	ADCFIFO1 ADCFIFO2	15	14	13	12	11	10	9	8
		D9	D8	D7	D6	D5	D4	D3	D2
		R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
		7	6	5	4	3	2	1	0
		D1	D0	0	0	0	0	0	0
		R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

表 3.15 模数转换控制寄存器 ADCTRL1 各位定义与说明

位	定义		说明
Bit15	Suspend-soft（该位无阴影）	0	当 Suspend-free＝0 时，立即停止
		1	在停止仿真前完成转换
Bit14	Suspend-free（该位无阴影）	0	按照 Suspend-soft 进行运行
		1	在仿真器悬挂期间保持运行
Bit13	ADCIMSTART（该位无阴影）	0	无操作
		1	立即启动转换
Bit12	ADC2EN（模数转换器 2 的使能/禁止位，该位有阴影）	0	关闭模/数转换器 2，不发生采样/保持/转换，数据寄存器 FIFO2 将不发生变化
		1	使能模/数转换器 1

（续）

位	定　义			说　明
Bit11	ADC1EN（模数转换器 1 的使能/禁止位，该位有阴影）		0	关闭模/数转换器 1，不发生采样/保持/转换，数据寄存器 FIFO1 将不发生变化
			1	使能模数转换器 1
Bit10	ADCCONRUN（该位有阴影）		0	无操作
			1	连续转换模式
Bit9	ADCINTEN（中断允许位，该位有阴影）		0	屏蔽模/数转换中断 ADCINTEN
			1	允许模/数转换中断 ADCINTEN
Bit8	ADCINTFLAG（中断标志位，该位无阴影）		0	无中断事件发生
			1	产生了一个中断事件
Bit7	ADCEOC（模数转换状态位，该位无阴影）		0	转换结束
			1	转换正在进行
Bit6-4	ADC2CHSEL（该位有阴影）			选择模/数转换器 2 的通道
	0	0	0	通道 9
	0	0	1	通道 10
	0	1	0	通道 11
	0	1	1	通道 12
	1	0	0	通道 13
	1	0	1	通道 14
	1	1	0	通道 15
	1	1	1	通道 16
Bit3-1	ADC1CHSEL（该位有阴影）			选择模/数转换器 1 的通道
	0	0	0	通道 1
	0	0	1	通道 2
	0	1	0	通道 3
	0	1	1	通道 4
	1	0	0	通道 5
	1	0	1	通道 6
	1	1	0	通道 7
	1	1	1	通道 8
Bit0	ADCSOC（模数转换启动位，该位有阴影）		0	无操作
			1	启动转换

表 3.16　模数转换控制寄存器 2（ADCTRL2）各位定义与说明

位	定　义		说　明
Bit15-11	保　留		读操作不确定，写无效
Bit10	ADCEVSOC（事件管理器 SOC 屏蔽位，该位有阴影）	0	禁止由事件管理器同步启动模/数转换
		1	允许由事件管理器同步启动模/数转换

（续）

位	定义			说明
Bit9	ADCEXTSOC（外部信号屏蔽位，该位有阴影）		0	禁止由外部信号同步启动模/数转换
			1	允许由外部信号同步启动模/数转换
Bit8	保　留			操作不确定，写无效
Bit7-6	ADCFIFO2			数据寄存器 ADCFIFO2 的状态，该位无阴影
	0	0		寄存器 ADCFIFO2 空
	0	1		寄存器 ADCFIFO2 有 1 个结果
	1	0		寄存器 ADCFIFO2 有 2 个结果
	1	1		寄存器 ADCFIFO2 有 2 个结果，并第 1 个结果被丢弃
Bit5	保　留			操作不确定，写无效
Bit3-4	ADCFIFO1			数据寄存器 ADCFIFO1 的状态，该位无阴影
	0	0		寄存器 ADCFIFO1 空
	0	1		寄存器 ADCFIFO1 有 1 个结果
	1	0		寄存器 ADCFIFO1 有 2 个结果
	1	1		寄存器 ADCFIFO1 有 2 个结果，并第 1 个结果被丢弃
Bit2-0	ADCPSCALE			模/数转换输入时钟分频系数（预定标因子值）
	0	0	0	4
	0	0	1	6
	0	1	0	8
	0	1	1	10
	1	0	0	12
	1	0	1	16
	1	1	0	20
	1	1	1	32

3.2.2　模/数转换控制与操作

DSP 控制器模/数转换控制与操作的步骤如下：

●设置模/数转换控制寄存器 ADCTRL1 的 ADC1CHSEL 或 ADC2CHSEL，以确定当前要转换的模拟输入通道。

●如果由软件立即启动转换，则设置模/数转换控制寄存器 ADCTRL1 的 ADCIMSTART 为 1；如果希望由片内事件管理中断同步启动转换，则设置模/数转换控制寄存器 ADCTRL2 的 ADCEVSOC 为 1（参见 3.1）；如果希望由片外引脚 ADCSOC/IOPC0 同步启动转换，则设置模/数转换控制寄存器 ADCTRL2 的 ADCEXTSOC 为 1。

●模/数转换是否完成，可以测试模/数转换控制寄存器 ADCTRL1 的 ADCEOC 是否为 0。如果不为 0，表示还在转换。

●如果模/数转换完成，会在模/数转换控制寄存器 ADCTRL1 的中断标志位 ADCINT-FLAG 置 1。如果模/数转换控制寄存器 ADCTRL1 的中断屏蔽位 ADCINTEN 为 1，则该中断将向 CPU 发出请求信号；否则，该中断不起作用。模/数转换控制寄存器 ADCTRL1 的中断屏蔽位 ADCINTEN 是阴影缓冲位，在转换期间写入时它不会马上见效，需等到下次转换时才起作用。

●当模/数转换完成后，读取结果寄存器前，最好先读取模/数转换控制寄存器 ADCTRL2 的 ADCFIFO1 或 ADCFIFO2，以确定当前结果寄存器的状态，保证读取的结果是正确的。另外，要注意 10 位的转换结果是放在结果寄存器的高 10 位上，该 10 位数据与外部模拟输入电压的关系为：

$$10\text{位数字结果}=1023\times\frac{\text{输入电压}}{\text{基准电压}}$$

●为了确保转换的精度，模/数转换的时间有一定的限制，必须大于 6μs。模/数转换的时间是由时钟源模块的 SYSCLK 经分频器产生。SYSCLK 是 CPU 时钟的 2 分频或 4 分频，由时钟控制寄存器 CKCR0 的 PLLPS 设置（参见 2.7）。分频器的分频系数由模/数转换控制寄存器 ADCTRL2 的 ADCPSCALE 设置。它们之间需满足

$$\text{SYSCLK 的周期}\times\text{分频系数}\times 6\geqslant 6\mu s$$

●如果将模/数转换控制寄存器 ADCTRL1 的 ADCCONRUN 设置为 1，模/数转换将进入连续转换模式，不再需要启动信号。这种方式对于连续快速的数据采集非常有用。该位是阴影位，若在转换期间进行了修改不能马上见效，需等到下次转换时才起作用。

3.3 SCI 串行通信接口模块

计算机与外界交换信息称为通信，通信有两种基本方式：并行通信和串行通信。并行通信就是数据的各位在多根传输线上同时从发送端传送到接收端。并行通信的优点是控制简单、传送速度快；缺点是使用的传输线多，通信成本高，特别是随着通信距离的增加，通信成本和可靠性将成为最突出的问题。因此并行通信适用于近距离、高速数据传输的场合。

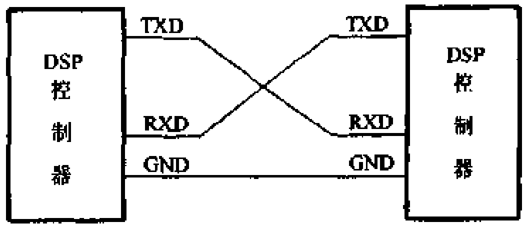
当通信双方距离较远时，一般采用串行通信方式，串行通信就是数据在一根传输线上由低位到高位一位一位地顺序传输。通常计算机之间、计算机与串行外设之间以及实时多处理机分级分布控制系统中，各 CPU 间都采用串行通信方式交换数据。串行通信的特点是通信距离远，通信成本低，但通信速率降低，且要求数据有固定格式，通信过程的控制要比并行通信复杂。

DSP 控制器串行通信接口（SCI）是一个标准的通用异步接收/发送（UART）通信接口。它的接收器和发送器都是双级缓冲的，有自己的使能和中断位，它们可以半双工或全双工工作。为了保证数据的完整性，串行通信接口对接收的数据进行间断检测、奇偶性、超时和帧错误的检查。串行通信接口波特率可高达 64kbps。

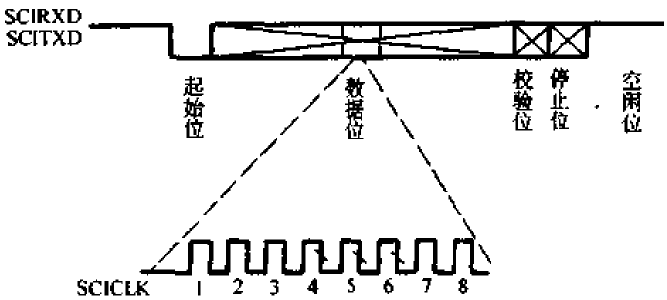
3.3.1 串行通信的工作原理

串行通信的工作原理如图 3.23 所示，这是最简单的点——点方式。为了进行通信，主机 A 的发送端 TXD 要与从机 B 的接收端 RXD 相联，主机 A 的接收端 RXD 要与从机 B 的发送端 TXD 相联，另外主机 A 与从机 B 要共地。

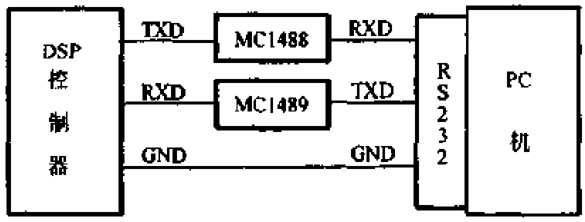
串行通信的发送是在一定的节拍下，一位一位地将数据移到引脚上；串行通信的接收也是在一定的节拍下，一位一位地将数据从引脚上读进来。因此，串行通信必须要有一个时钟来指挥，而且发送端和接收端还必须“同步”，这是正确通信的关键。常用的“同步”方法有两种：通过一根时钟信号线将主从机联接起来，实行强制同步，这种“同步”方式可靠，但需增加一根时钟信号线，后一节介绍的串行外设接口(SPI)就是采用这种“同步”方式；另外一种就是在主从机分别设立时钟(SCICLK)，两个时钟具有相同的周期(波特率)，这样在规定的字符传送格式（起始位＋数据位＋校验位＋停止位）配合下，可以达到主从机的“同步”，如图 3.23 所示。严格讲，前者称为同步串行通信，后者称为异步(准同步)串行通信。采用异步通信方式的主从机通过各自的时钟信号来“同步”，由于二者的时钟相位或时钟的周期不一定完全一致，即各自的时钟有一定的偏差范围，故这个偏差有可能会影响串行通信的正确性，应该特别注意。



a）点一点串行通信



b）信息块的格式



c）电平转换

图 3.23 异步串行通信的原理

串行通信时信息在一根信号线上传送，不仅要传送数据信息，还要传送联络控制信号，为了区分这根传输线上串行传送的信息流中哪个是数据，哪个是联络控制信号，就引出了串行通信中的一系列约定，也称通信规程或通信协议，如数据格式、传输速度、差错检验方式、传输控制步骤等都应作出严格规定。

通用异步接收/发送（UART）通信的信息块格式非常重要，一个完整的信息块格式是：起始位＋数据位＋校验位＋停止位。串行传送开始后，发送端在发送每个数据字符前首先发送一个起始位作为接收该字符的同步信号，然后发送有效数据字符和校验位（也可以没有），在字符结束时再发送 1 位或 2 位的停止位，停止位后是长度不等的空闲位。停止位和空闲位是高电平，以保证在起始位的开始处一定有一个下跳沿作为起始检测标志。在异步通信的信息块格式中增加起始位和停止位，是为了主从机之间来核对收发双方的同步，确保串行通信的可靠性。但由于在每个信息块的首尾都要有增加起始位和停止位，也降低了串行通信的传输效率。

DSP 控制器的串行通信由时钟 SCICLK 指挥，每位占有 8 个 SCICLK 周期，在其第 4、5、6 个脉冲的下降沿采样引脚，采取 3 取 2 的投票机制决定该位的状态。这比普通单片机只在每

位采样1次就决定该位的状态要合理，抗干扰性能也要好。

为了保证异步串行通信主从机之间的同步，除了要有前面的通信数据格式外，还必须要求主从机的发送与接收时钟具有相同的周期，即相同的波特率。

在串行通信中还要解决的一个基本问题是通信的主从机必须按照统一的电气和物理接口标准来连接，如信号电平、信号定义与电缆特性等都必须按统一的标准。如果主从机之间的电气和物理接口标准不一致，一定要进行电气和物理接口的转换。最典型的是DSP控制器与PC机串口连接时，由于前者是TTL（5V）电平，后者是RS-232（15V）电平，必须进行电平转换，如图3.23所示。RS-232是目前最普遍的串行接口标准，它除了对信号电平进行了定义外（按负逻辑定义的，即用－5～15V表示逻辑1，＋5～＋15V表示逻辑0），还对信号电缆的物理性能、传输距离等进行了规定。另外，RS-422/RS-485也是目前流行的串行接口标准，它的传输距离要比RS-232接口标准要远许多。为了解决RS-232的传输距离的问题，也可以采用调制解调器的方法。有兴趣的读者可阅读有关的资料。

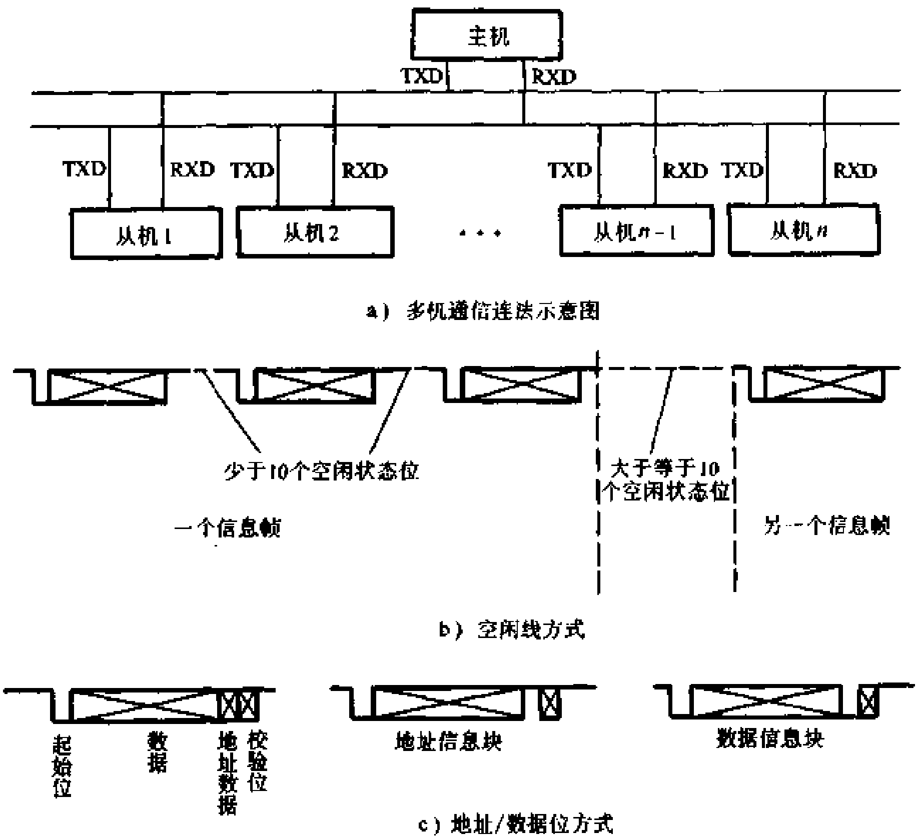


图3.24　多机通信原理

对于点一点的通信方式，在前面的协议基础上可进行正常的通信。对于点一多点，即多机通信时，光有前面的协议还不够，还需要解决一个多机寻址的问题，参见图3.24。在多机通信时，所有设备都挂在串行通信总线（TXD和RXD）上，当主机要与某个从机通信时，怎样去识别这个从机？这个从机怎样确认是与它通信？这是多机通信时必须要解决的问题。为了区分每一个挂接串行通信总线上的设备，自然想到的是给每一个设备分配一个地址。在数据信息传送之前，先在通信总线上广播地址信息，与这个地址匹配的设备将被唤醒，与这个

地址不匹配的设备继续处于睡眠状态，这样一来收发双方就建立起逻辑上的联接，随后就可以进行正常的数据信息传送。对于这种方法，还有一个问题要解决，就是怎样区分接收到的是地址信息还是数据信息？DSP 控制器的 SCI 串行通信接口模块提供了两种方法：

●空闲线方式。多机通信中有地址信息块也有数据信息块，多个信息块组成一个信息帧，一个信息帧包含地址信息块和数据信息块，并且规定地址信息块在前，数据信息块在后。地址信息可以由 1 个或多个地址信息块构成，数据信息也可以由 1 个或多个数据信息块构成。同一个信息帧内的信息块之间的空闲状态位（高电平）小于 10 个；而两个不同的信息帧之间的空闲状态位（高电平）必须要大于或等于 10 个。这样一来，通过空闲状态的长短区分信息帧，在信息帧里前面的信息块是地址信息，后面的是数据信息，地址信息和数据信息各占几块由用户自己定义。

●地址位方式。这种方式是在数据格式上进行一些修改，即增加 1 个地址/数据位，如果该位为 1 表示信息块是地址信息，否则表示信息块是数据信息。采用地址位方式不需要监视空闲状态的长短，编程比较简明。

地址位方式适合于短信息的传送，空闲线方式适合于长信息的传送。主要原因是地址位方式在每个信息块都增加了 1 位地址/数据位，对于长信息的传送增加了许多额外的开销，降低了传送效率。

3.3.2 串行通信接口模块 SCI 的结构

DSP 控制器串行通信接口模块由发送和接收两大部分组成，其结构如图 3.25 所示。与串行通信相关的两个引脚：

- SCIRXD/IO，串行通信数据接收，也可以做普通 I/O。
- SCITXD/IO，串行通信数据发送，也可以做普通 I/O。

与串行通信相关的寄存器说明如表 3.17 所示。其中控制类的寄存器有 7 个，分别用来设置数据格式、中断使能、中断优先级、波特率、引脚复用功能的选择以及反映通信的状态等；数据类寄存器有 3 个，分别是数据发送缓冲寄存器 SCITXBUF、数据接收缓冲寄存器 SCIRXBUF 和仿真数据接收缓冲寄存器 SCIRXEMU。与数据发送缓冲寄存器 SCITXBUF 相连的发送移位寄存器 TXSHF 是由模块本身使用，用户不能直接操作，它的作用就是把数据发送缓冲寄存器 SCITXBUF 中的数据一位一位地移出到引脚 SCITXD/IO 上。与此类似，接收移位寄存器 RXSHF 是将引脚 SCIRXD/IO 上的信息一位一位地移到数据接收缓冲寄存器 SCIRXBUF 和仿真数据接收缓冲寄存器 SCIRXEMU 中，用户不能直接与接收移位寄存器 RXSHF 打交道，它是给模块本身使用。数据接收缓冲寄存器 SCIRXBUF 和仿真数据接收缓冲寄存器 SCIRXEMU 是同一个物理寄存器，只是它们有不同的地址。仿真数据接收缓冲寄存器 SCIRXEMU 主要为仿真器使用，对它的读取不影响标志位的变化。

DSP 控制器的串行通信的功能较普通的单片机的要强，涉及到的控制类的寄存器较多，需要设置的参数较多，这也带来了许多的灵活性。与其它的可编程模块一样，在使用之前需要对控制类的寄存器进行初始化，包括数据格式、中断使能、中断优先级、波特率等参数的设置。在初始化完成后，用户实际上只需跟数据发送缓冲寄存器 SCITXBUF 和数据接收缓冲寄存器 SCIRXBUF 打交道。如果要发送数据，只要把数据写入到 SCITXBUF 即可，加入起始位、停止位、校验位以及在波特率规定的节拍下移位到发送引脚 SCITXD/IO 等工作由串行

通信 SCI 模块本身自动完成。如果要接收数据，只要把 SCIRXBUF 的内容读出即可，因为从引脚 SCIRXD/IO 移位来的信息由串行通信 SCI 模块本身自动地去掉起始位、停止位、校验位，并将数据放到 SCIRXBUF 中。如果串行通信 SCI 模块的发送功能被开启，当数据写入到 SCITXBUF 中时就自动启动了发送过程，发送完成后会在相应的标志位上建立标志。同样的道理，如果串行通信 SCI 模块的接收功能被开启，当引脚 SCIRXD/IO 出现下降沿的跳变时（在没有接收信息时该引脚为高电平），就自动地启动了接收过程，如果属于正常接收，则将数据分离出来放到 SCIRXBUF 中并置接收到的标志；如果属于非正常接收，即出现间断（BRK）、帧错（FE）、溢出（OE）、校验错（PE）等情况，将在相应标志位上置标志。另外，发送完成和接收到都有相应的中断，通过这些中断可以方便地实现各种通信任务。

为了详细了解 DSP 控制器的串行通信功能，必须对它的控制类寄存器进行深入的了解，下面对控制类寄存器逐一进行介绍。

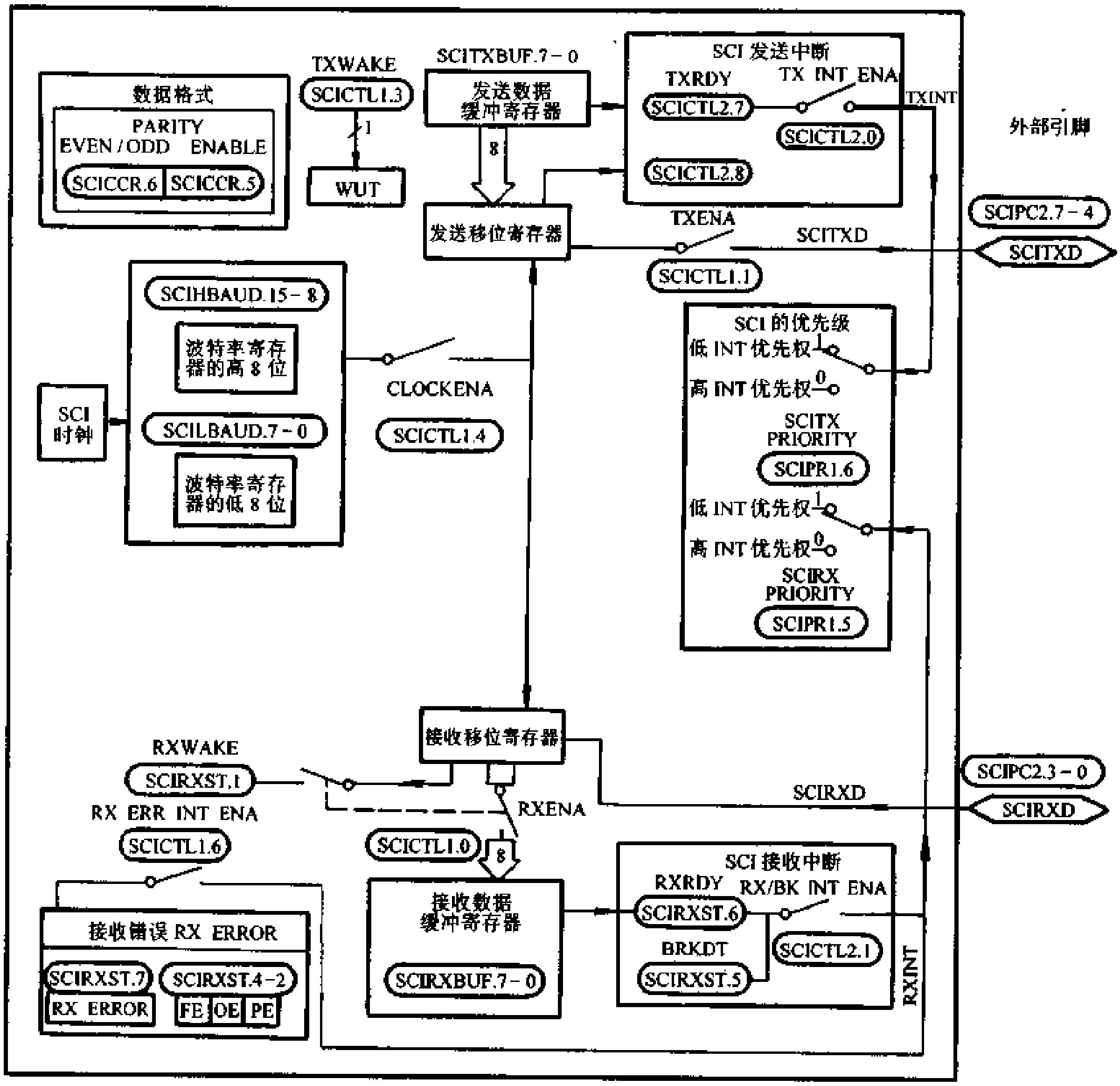


图 3.25 串行通信接口结构图

表 3.17 串行通信接口寄存器说明

地址	寄存器	位数							
7050h	SCICCR	7	6	5	4	3	2	1	0
		STOP BITS	EVEN/ODD	PARITY	SCIENA	ADDR/IDLE	SCI CHAR2	SCI CHAR1	SCI CHAR0
		RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0
7051h	SCICTL1	7	6	5	4	3	2	1	0
		Reserved	RX ERR INT	SW RESET	CLOCK ENA	TXWAKE	SLEEP	TXENA	RXENA
			RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0
7052h	SCIHBAUD	15	14	13	12	11	10	9	8
		BAUD15	BAUD14	BAUD13	BAUD12	BAUD11	BAUD10	BAUD9	BAUD8
		RW-0	RW-0	RW-0	RW-0	RW-0	R-0	RW-0	RW-0
7053h	SCILBAUD	7	6	5	4	3	2	1	0
		BAUD7	BAUD6	BAUD5	BAUD4	BAUD1	BAUD2	BAUD1	BAUD0
		RW-0	RW-0	RW-0	RW-0	RW-0	R-0	RW-0	RW-0
7054h	SCICTL2	7	6	5	4	3	2	1	0
		TXRDY	TX EMPTY	Reserved				RX/BK INT	TX INT ENA
		R-1	R-1					RW-0	RW-0
7055h	SCIRXST	7	6	5	4	3	2	1	0
		RX ERROR	RXRDY	BRKDT	FE	OE	PE	RXWAKE	Reserved
		R-0	R-0	R-0	R-0	R-0	R-0	R-0	
7056h	SCIRXEMU	7	6	5	4	3	2	1	0
		ERXDT7	ERXDT6	ERXDT5	ERXDT4	ERXDT3	ERXDT2	ERXDT1	ERXDT0
		R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x

（续）

地址	寄存器	位数							
7057h	SCIRXBUF	7	6	5	4	3	2	1	0
		RXDT7	RXDT6	RXDT5	RXDT4	RXDT3	RXDT2	RXDT1	RXDT0
		R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x
7058h	—	Reserved							
7059h	SCITXBUF	7	6	5	4	3	2	1	0
		TXDT7	TXDT6	TXDT5	TXDT4	TXDT3	TXDT2	TXDT1	TXDT0
		RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW 0	RW-0
705Ah	—	Reserved							
705Bh	—	Reserved							
705Ch	—	Reserved							
705Dh	—	Reserved							
705Eh	SCIPC2	7	6	5	4	3	2	1	0
		SCITXD DATA IN	SCITXD DATA OUT	SCITXD FUNCTION	SCITXD DATA DIR	SCIRXD DATA IN	SCIRXD DATA OUT	SCIRXD FUNCTION	SCIRXD DATA DIR
		R-x	RW-0	RW-0	RW-0	R-x	RW-0	RW-0	RW-0
705Fh	SCIPRI	7	6	5	4	3	2	1	0
		Reserved	SCITX PRIORITY	SCIRX PRIORITY	SCI ESPEN	Reserved			
			RW-0	RW-0	RW-0				

在进行串行通信之前，首先要确定数据格式或信息块的格式。这一点是由 SCI 通信控制寄存器 SCICCR 进行设置，参见表 3.18。信息块的起始位始终是 1 位，停止位可选择是 1 位或 2 位，由 SCICCR 的 STOP BITS 设置，该位为 0，则选取 1 位停止位，否则选取 2 位停止位；数据位的长度是可编程的，可设置为 1 位到 8 位，这比普通单片机要灵活，由 SCICCR 的 SCICHAR2：0 设置，要注意当数据位长度 L 小于 8 位时，写到发送缓冲寄存器 SCITXBUF 的数据只有低 L 位被发送；校验位可有可无，由 SCICCR 的 PARITY ENABLE 设置，该位为 0，则不需要校验位，否则需要校验位；校验方式有两种，奇校验和偶校验，由 SCICCR 的 PARITY 设置，该位为 1，则选取偶校验，否则选取奇校验；空闲线方式与地址位方式的选择，由 SCICCR的 ADDR/IDLE MODE 设置，该位为 0，则选取空闲线方式，否则选择地址位方式；SCICCR 的 SCIENA 是一个总开关位，该位为 0，则关闭 SCI 模块，否则使能 SCI 模块。

表 3.18　串行通信接口控制寄存器（SCICCR）各位定义与说明

<table>
<tr><th>位</th><th colspan="4">定　　　义</th><th>说　明</th></tr>
<tr><td rowspan="2">Bit7</td><td colspan="3" rowspan="2">STOP BITS（停止位的数目）</td><td>0</td><td>一个停止位</td></tr>
<tr><td>1</td><td>两个停止位</td></tr>
<tr><td rowspan="2">Bit6</td><td colspan="3" rowspan="2">PARITY（奇/偶校验选择，在 PARITY ENABLE 置位时有效）</td><td>0</td><td>奇校验</td></tr>
<tr><td>1</td><td>偶校验</td></tr>
<tr><td rowspan="2">Bit5</td><td colspan="3" rowspan="2">PARITY ENABLE（奇/偶校验使能）</td><td>0</td><td>禁止奇偶校验</td></tr>
<tr><td>1</td><td>使能奇偶校验</td></tr>
<tr><td rowspan="2">Bit4</td><td colspan="3" rowspan="2">SCI ENA（通信使能位）</td><td>0</td><td>禁止通信</td></tr>
<tr><td>1</td><td>使能通信</td></tr>
<tr><td rowspan="2">Bit3</td><td colspan="3" rowspan="2">ADDR/IDLE MODE（多处理机模式控制位）</td><td>0</td><td>选中空闲线模式</td></tr>
<tr><td>1</td><td>选中地址位模式</td></tr>
<tr><td rowspan="9">Bit2-0</td><td colspan="3">SCICHAR2-0</td><td colspan="2">字长控制位</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td colspan="2">1 位字长</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td colspan="2">2 位字长</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td colspan="2">3 位字长</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td colspan="2">4 位字长</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td colspan="2">5 位字长</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td colspan="2">6 位字长</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td colspan="2">7 位字长</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td colspan="2">8 位字长</td></tr>
</table>

在信息块的格式确定之后，就要确定串行通信的波特率。它由 16 位 SCI 波特率寄存器 SCIHBAUD 和 SCILBAUD 设置，前者是波特率寄存器的高 8 位，后者是波特率寄存器的低 8 位，它的数据记为 BRR。串行通信的时钟 SCICLK 是由时钟源模块的 SYSCLK 按照 SCI 波特率寄存器规定的分频系数分频后得到。注意到波特率是按位计算（位/s），而每位需要 8 个 SCICLK 周期，所以串行通信的波特率与 BRR、SYSCLK 的关系为：

$$\text{串行通信的波特率}=\begin{cases}\dfrac{\text{SYSCLK}}{(\text{BRR}+1)\times 8} & 1\leqslant \text{BRR}\leqslant 65535\\[2ex] \dfrac{\text{SYSCLK}}{16} & \text{BRR}=0\end{cases}$$

在设置好串行通信的波特率后，就要确定接收部分或发送部分的控制位，这些是由 SCI 控制寄存器 SCICTL1 和 SCICTL2 来完成，参见表 3.19～表 3.21。

表 3.19　串行通信接口控制寄存器 1（SCICTL1）各位定义与说明

位	定　义		说　明
Bit7	保　留		读操作不确定，写操作无效
Bit6	RX ERR INT ENA（接收错误中断使能）	0	禁止接收错误中断
		1	使能接收错误中断
Bit5	SW RESET（串行通信接口软件复位（低有效））	0	将 0 写入该位来初始化串行通信接口状态机且操作标志至复位条件。该位不影响其他任何配置位，也不改变 CLOCK ENA 位的状态。所有起作用的逻辑都保持确定的复位态直至将 1 写入该位来重新使能串行通信接口。接收间断检测（BRKDT 标志）发生后，清除该位。SW RESET 影响串行通信接口的操作标志，但并不影响配置位。表 3.20 列出了受影响的标志位，一旦确定了 SW RESET，标志位就被冻结直至该位解除
		1	
Bit4	CLOCK ENA（内部时钟使能）	0	禁止内部时钟
		1	使能内部时钟
Bit3	TXWAKE（发送唤醒方法选择）	0	没有选定发送特征
		1	选定的发送特征取决于空闲线模式或地址位模式。在空闲线模式下，写 1 到 TXWAKE，然后将数据写入寄存器 SCITXBUF 来产生一个 11 个数据位的空闲周期；在地址位模式下，写 1 到 TXWAKE，然后将数据写入寄存器SCITXBUF 并设置该帧的地址位为 1
Bit2	SLEEP（休眠位）	0	禁止休眠方式
		1	使能休眠方式
Bit1	TXENA（发送器使能）	0	禁止发送器
		1	使能发送器
Bit0	RXENA（接收器使能）	0	禁止将接收到的字符传送到 SCIRXEMU 和 SCIRXBUF 接收缓冲器
		1	发送接收的字符到 SCIRXEMU 和 SCIRXBUF

表 3.20　受 SW RESET 影响的标志位

SCI 标志位	在寄存器中的位	SW RESET 后的值	SCI 标志位	在寄存器中的位	SW RESET 后的值
TXRDY	SCICTL2.7	1	FE	SCIRXST.4	0
TX EMPTY	SCICTL2.6	1	BRKDT	SCIRXST.5	0
RXWAKE	SCIRXST.1	0	RXRDY	SCIRXST.6	0
PE	SCIRXST.2	0	RX ERROR	SCIRXST.7	0
OE	SCIRXST.3	0			

表 3.21 串行通信接口通信控制寄存器 2（SCICTL2）各位定义与说明

位	定义		说明
Bit7	TXRDY（发送寄存器准备好标志位）	0	SCITXBUF 满
		1	SCITXBUF 准备接收下一个字符
Bit6	TX EMPTY（发送器空标志）	0	发送缓冲器、移位寄存器或两者被装入数据
		1	发送缓冲器和移位寄存器都空
Bit5-2	保留		读操作不确定，写操作无效
Bit1	RX/BK INT ENA（接收缓冲器/间断中断允许）	0	禁止 RXRDY/BRKDT 中断
		1	允许 RXRDY/BRKDT 中断
Bit0	TXINT ENA（SCITXBUF 中断允许位）	0	禁止 TXRDY 中断
		1	允许 TXRDY 中断

如果不希望串行发送功能起作用，可将 SCICTL1 的 TXENA 清 0，否则就要将其置为 1。如果不希望串行接收功能起作用，可将 SCICTL1 的 RXENA 清 0，否则就要将其置为 1。如果要关闭串行通信时钟 SCICLK，可将 SCICTL1 的 CLOCKENA 清 0，否则就要将其置为 1。

当数据写入到 SCITXBUF 时，SCICTL2 的 TXRDY 被清 0；当 SCITXBUF 的数据被全部移出后，即该数据发送完毕，则 SCICTL2 的 TXRDY 被置 1，并且产生发送中断 TXINT。SCICTL2 的 TXEMPTY 是与 TXRDY 类似的标志位，作用相同但不产生中断请求。

如果要屏蔽发送中断 TXINT，可将 SCICTL2 的 TXINTENA 清 0，否则就要将其置为 1。如果要屏蔽接收中断 RXINT，可将 SCICTL2 的 RX/BK INTENA 清 0，否则就要将其置为 1。如果要屏蔽接收错误中断，可将 SCICTL1 的 RXERR INTENA 清 0，否则就要将其置为 1。后面两个中断 RX/BK 和 RXERR 共用 RXINT，它们之间通过 SCI 接收状态寄存器 SCIRXST 的状态标志位来区分（参见表 3.22）。

表 3.22 接收状态寄存器（SCIRXST）各位定义与说明

位	定义		说明
Bit7	RX ERROR（接收器错误标志，该位是间断检测、帧错误、超时和校验允许标志（位 5-2：BRKDT，FE，OE 和 PE）的逻辑或）	0	无错误标志被置位
		1	有错误标志被置位
Bit6	RXRDY（串行通信接口接收器准备好标志）	0	SCIRXBUF 中无新字符
		1	准备从 SCIRXBUF 中读出新字符
Bit5	BRKDT（间断检测标志位）	0	无间断条件
		1	产生间断条件。当串行通信接口的接收数据线从失去的第一个停止位开始后连续保持低电位至少十位时，就产生间断条件
Bit4	FE（帧错误标志位）	0	未检测到帧错误
		1	检测到帧错误。当没有找到预期的停止位时，串行通信接口置位该位

（续）

位	定　　义		说　　明
Bit3	OE（超时错误标志位）	0	未检测到超时错误
		1	检测到超时错误。在前一个字符被 CPU 完全读取前，当字符传送到 SCIRXEMU 和 SCIRXBUF 中时，串行通信接口置位该位
Bit2	PE（奇/偶校验错误标志位）	0	无奇偶校验错误或奇偶校验被禁止
		1	检测到奇偶校验错误
Bit1	RXWAKE（接收器唤醒检测标志位）	0	无接收器唤醒条件
		1	检测到接收器唤醒条件。在地址位多处理机方式中，TXWAKE 反映了保存在 SCIRXBUF 中字符的地址位置；在空闲线多处理机方式中，如果检测到 SCIRXD 数据线空闲就置位 RXWAKE，SCIRXD 线是 RXSHF 的输入
Bit0	保　留		读操作不确定，写操作无效

在进行多机通信时，需要先确认待通信的地址，如果是与自己通信则唤醒接收功能，否则使接收功能处于睡眠状态。这一点由 SCICTL1 的 SLEEP 进行设置，该位为 0 唤醒接收功能，该位为 1 处于睡眠状态。

在空闲线方式下（SCICCR 的 ADDR/IDEL MODE＝0），为了区别两个不同的信息帧需要有 10 个以上的空闲状态位。为了达到这个要求，一种方法是在上一个信息帧发送完后，人为地延迟 10 个以上的空闲状态位后再发下一个信息帧；另一种方法是在上一个信息帧发送完后，将 1 写到 SCICTL1 的 TXWAKE，然后任意写一个数据到发送数据缓冲寄存器 SCITXBUF，SCI 模块将自动地在发送引脚上产生 11 个空闲状态位，这个操作完成后自动地将 SCICTL1 的 TXWAKE 清 0。后一种方法由硬件自动产生空闲状态的时间，比较可靠。

在地址位方式下（SCICCR 的 ADDR/IDEL MODE＝1），如果将 1 写到 SCICTL1 的 TXWAKE，然后向发送数据缓冲寄存器 SCITXBUF 写一个“数”，SCI 模块将把这个“数”当作地址并自动地在地址/数据标志位上置 1；如果将 0 写到 SCICTL1 的 TXWAKE，然后向发送数据缓冲寄存器 SCITXBUF 写一个“数”，SCI 模块将把这个“数”当作数据并自动地在地址/数据标志位上清 0。

在接收的过程中，由于各种干扰的原因可能导致接收失败。出现接收失败时，希望能复位串行通信模块但又不破坏原有的参数配置。这时候可以写一个 0 到 SCICTL1 的 SWRESET，使得 SCI 接收状态寄存器 SCIRXST 中的所有错误标志位被清 0 但又不破坏原有的参数配置。如果在接收过程中探测到“间断”错误，即 SCI 状态寄存器 SCIRXST 的 BRKDT 置了标志，将自动使 SCICTL1 的 SWRESET 清 0。

从引脚 SCIRXD/IO 的电平变化到数据放到接收缓冲寄存器 SCIRXBUF 中，这个过程是自动完成的。这个过程是否正常，出现了哪些错误，也自动地记录在 SCI 接收状态寄存器中。

如果是正常接收且数据已放到接收缓冲寄存器 SCIRXBUF 中，则 SCIRXST 的 RXRDY 被置 1。如果 SCICLTL2 的 RX/BR INTENA 为 1，即接收中断 RXINT 没有屏蔽，还将发出中断请求。要注意，数据接收缓冲寄存器 SCIRXBUF 与仿真数据接收缓冲寄存器 SCIRXE-

MU 是同一个物理寄存器，但它们有不同的地址（前者是 7057H，后者是 7056H），如果以地址 7056H 读取仿真数据接收缓冲寄存器 SCIRXEMU，标志 RXRDY 不发生变化。

如果在接收过程中探测到“间断”错误，即在引脚 SCIRXD/IO 上出现持续 10 个以上低电平状态位，这种情况肯定丢失了停止位，将使 SCIRXST 的 BRKDT 置 1，否则将为 0。如果 SCICLTL2 的 RX/BR INTENA 为 1，即接收中断 RXINT 没有屏蔽，还将发出中断请求。

如果在收到第一个停止位后没再收到期望的停止位，将出现“帧错”。就是说该帧信息出现了不正常的情况。此时，将使 SCIRXST 的 FE 置 1，否则将为 0。

如果在接收缓冲寄存器 SCIRXBUF 的数据还未读出，又有数据要放到 SCIRXBUF 中，将产生“溢出”错误，这将使 SCIRXST 的 OE 置 1，否则将为 0。

如果接收方计算的校验结果与发送过来的不一致，将产生“校验错”的错误，这将使 SCIRXST 的 PE 置 1，否则将为 0。

如果在接收过程中出现了“间断”、“帧错”、“溢出”或“校验错”的任何一种情况，都将使 SCIRXST 的 RXERROR 置 1，否则将为 0。如果 SCICLTL1 的 RX ERR INTENA 为 1，即接收中断 RXINT 没有屏蔽，还将发出中断请求。注意，要使这个状态标志位清 0 必须由 SCICTL1 的 SWRESET 来完成。

SCIRXST 的 RXWAKE 与 SCICTL1 的 TXWAKE 对应。在空闲线方式下，该位为 1 表示当前是空闲状态位；在地址位方式下，该位为 1 表示收到的信息是地址信息。该位是只读位，下述情况将使它复位：读 SCIRXBUF；在地址位方式下，收到数据信息块；使 SCICTL1 的 SWRESET（低）有效；系统复位。

串行发送、接收中断的优先级是可以编程的，由优先级控制寄存器 SCIPRI 来设置（参见表 3.24）。如果 SCIPRI 的 SCITX PRIORITY 为 1，则发送中断 TXINT 是高优先级的中断，否则是低优先级的中断；如果 SCIPRI 的 SCIRX PRIORITY 为 1，则接收中断 RXINT 是高优先级的中断，否则是低优先级的中断。详情可见 3.7。

用于串行通信的二个引脚是功能复用的，通过端口控制寄存器 SCIPC2 来规定，参见表 3.23。如果引脚 SCITXD/IO 要作为串行通信的发送引脚，则要将 SCIPC2 的 SCITXD FUNCTION 置为 1；如果引脚 SCIRXD/IO 要作为串行通信的接收引脚，则要将 SCIPC2 的 SCIRXD FUNCTION 置为 1。当引脚 SCITXD/IO 和 SCIRXD/IO 作为通用 I/O 引脚时，需要规定它们的数据传送方向，即是作输入还是输出，分别由 SCIPC2 的 SCITXD DATA DIR、SCIRXD DATA DIR 来配置。

表 3.23　串行通信接口端口控制寄存器 2（SCIPC2）各位定义与说明

位	定　　义	说　明	
Bit7	SCITXD DATA IN（引脚 SCITXD 的当前值）	0	引脚 SCITXD 的值读出为 0
		1	引脚 SCITXD 的值读出为 1
Bit6	SCITXD DATA OUT（引脚 SCITXD 的输出值，如果 SCITXD 是通用数字 I/O 输出引脚，那么该位就存有由 SCITXD 输出的数据）	0	引脚 SCITXD 上输出 0
		1	引脚 SCITXD 上输出 1
Bit5	SCITXD FUNCTION（定义引脚 SCITXD 的功能）	0	SCITXD 是通用数字 I/O 引脚
		1	SCITXD 是串行通信接口发送引脚

（续）

位	定义		说明
Bit4	SCITXD DATA DIR（定义引脚 SCITXD 的数据方向，如果 SCITXD 是通用数字 I/O 引脚，那么该位确定 SCITXD 的方向）	0	SCITXD 是数字输入引脚
		1	SCITXD 是数字输出引脚
Bit3	SCIRXD DATA IN（引脚 SCIRXD 的当前值）	0	引脚 SCIRXD 的值读出为 0
		1	引脚 SCIRXD 的值读出为 1
Bit2	SCIRXD DATA OUT（引脚 SCIRXD 的输出值，如果 SCIRXD 是通用数字 I/O 输出引脚，那么该位就存有由 SCIRXD 输出的数据）	0	引脚 SCIRXD 上输出 0
		1	引脚 SCIRXD 上输出 1
Bit1	SCIRXD FUNCTION（定义引脚 SCIRXD 的功能）	0	SCIRXD 是通用数字 I/O 引脚
		1	SCIRXD 是串行通信接口发送引脚
Bit0	SCIRXD DATA DIR（定义引脚 SCIRXD 的数据方向，如果 SCIRXD 是通用数字 I/O 引脚，那么该位确定 SCIRXD 的方向）	0	SCIRXD 是数字输入引脚
		1	SCIRXD 是数字输出引脚

表 3.24　串行外设接口优先级控制寄存器（SCIPRI）各位定义与说明

位	定义		说明
Bit7	保　留		读操作不确定，写操作无效
Bit6	SCITX PRIORITY（发送器中断优先级选择）	0	高优先级中断请求
		1	低优先级中断请求
Bit5	SCIRX PRIORITY（接收器中断优先级选择）	0	高优先级中断请求
		1	低优先级中断请求
Bit4	SCI ESPEN（仿真挂起允许）	0	当挂起（suspend）信号变高时，串行通信接口继续操作直到当前的发送和接收功能完成
		1	当挂起（suspend）信号变高时，串行通信接口状态机被冻结
Bit3-0	保　留		读操作不确定，写操作无效

3.3.3　多机通信

多机通信的原理前面已作了叙述。为了区别挂接在串行总线（TXD、RXD）上的设备，对每个设备都要分配一个地址。另外，特别要注意串行总线上每次只能有一个设备处于发送状态，即只有一个“讲者”；但是可以有多个设备处于接收状态，即多个“听者”。因此，为了进行正常的通信，需要对帧的格式进行规定。每帧的开头的信息块应为地址信息块，而后再跟数据信息块。对于每个挂接在串行总线上的设备，平常使自己的接收功能处于睡眠状态。处于睡眠状态时，可以接收但不改变标志 RXRDY 和其它错误标志，也不引起中断 RXINT；但若探测到是地址信息块（RXWAKE=1），则会改变标志 RXRDY，也会引起中断 RXINT（如果没有屏蔽）。当总线上出现帧的开头——地址信息块时，每个“听者”迅速与自己的地址比较，若一致则唤醒接收功能，将后续的数据信息接收进来；否则，继续使自己的接收功能处于睡眠状态。

DSP 控制器的多机通信由 SCICCR 的 ADDR/IDEL MODE、SCICTL1 的 SLEEP、SCI-

CTL1 的 TXWAKE 和 SCIRXST 的 RXWAKE 的逻辑组合来完成。由 ADDR/IDEL MODE 确定多机通信是采用空闲线方式还是地址位方式。由 TXWAKE、RXWAKE 和 SLEEP 控制收发的进程。

1. 空闲线方式

●首先将 SLEEP 置为 1，使接收功能处于睡眠状态。

●如果是发送，在每帧的开始将 TXWAKE 置为 1，并向发送缓冲寄存器 SCITXBUF 任意写一个数，将在发送端的引脚 SCITX/IO 上产生 11 个空闲状态位，以此告知接收方紧跟着的就是地址信息块。随后，将 TXWAKE 清为 0，再将数据信息块依次发出。要注意每块之间的空闲时间要少于 10 个空闲状态位，否则将使接收方误认为是一个新的信息帧，导致接收错误。

●如果是接收，当模块探测到地址信息块到来时（RXWAKE=1），尽管接收功能处于睡眠状态仍会使标志 RXRDY 置 1，也会引起中断 RXINT（如果没有屏蔽）。这样一来，在相应的（子）程序中迅速与自己的地址比较，若一致将 SLEEP 清 0，恢复接收功能，将后续的数据信息块全部接收进来，完成后记得将 SLEEP 重新置为 1，等待下一帧信息的到来；若不一致，保持原状，这个时候尽管自己的引脚 SCIRXD/IO 上继续有数据信息流动，接收缓冲寄存器也接收数据，但是由于标志 RXRDY 不变化，所以有关的（子）程序不会运行，相当于接收功能处于睡眠状态。

2. 地址位方式

●首先将 SLEEP 置为 1，使接收功能处于睡眠状态。

●如果是发送，在每帧的开始将 TXWAKE 置为 1，并将目的地址写入到发送缓冲寄存器 SCITXBUF，此时自动地在数据格式的地址/数据位置 1，告知接收方这是地址信息块。在目的地址发送完后，将 TXWAKE 清为 0，此时自动地在数据格式的地址/数据位清为 0，再将数据信息块依次发出。

●如果是接收，当模块探测到地址信息块到来时（RXWAKE=1），尽管接收功能处于睡眠状态仍会使标志 RXRDY 置 1，也会引起中断 RXINT（如果没有屏蔽）。这样一来，在相应的（子）程序中迅速与自己的地址比较，若一致将 SLEEP 清 0，恢复接收功能，将后续的数据信息块全部接收进来，完成后记得将 SLEEP 重新置为 1，等待下一帧信息的到来；若不一致，保持原状，这个时候尽管自己的引脚 SCIRXD/IO 上继续有数据信息流动，接收缓冲寄存器也接收数据，但是由于标志 RXRDY 不变化，所以有关的（子）程序不会运行，相当于接收功能处于睡眠状态。

3.4 SPI 串行外设接口模块

目前，控制系统微型化的要求越来越高，便携式的控制器、测量仪器等的需求量越来越大。为了使数字处理系统微型化，首先要设法减少芯片的引脚数。这样一来过去常用的并行总线接口方案由于需要较多的引脚线而不得不舍弃，转而采用只需少量引脚线的串行总线接口方案。SPI 就是这样一种串行总线的外设接口，它只需 3 根引脚线就可以与外部设备相接。目前与 SPI 总线兼容的芯片越来越多，因此在 DSP 控制器的片内集成 SPI 接口模块为控制系统的设计带来了很大的方便。

SPI 实际上是一种串行总线标准，它实现二个设备之间的信息交换，与 SCI 串行异步通信

有相似的地方也有不同之处。相同之处在于它们都是串行的信息交换，不同的是 SCI 是一种异步（准同步）方式，两台设备有各自的串行通信时钟，在相同的波特率和数据格式下达到同步，而 SPI 是一种真正的同步方式，两台设备在同一个时钟下工作。因此，SCI 只需两根引脚线（发送与接收）而 SPI 需要 3 根引脚线（发送、接收与时钟）。由于 SPI 是同步方式工作，它的传输速率远远高于 SCI。目前，采用 SPI 接口方式的 A/D、I/O、RAM 等芯片越来越多，传输速率高达几十兆。

3.4.1 串行外设接口结构与工作原理

采用 SPI 总线方式交换数据与 SCI 串行通信类似，有主机、从机的概念。主机的发送与从机的接收相连，主机的接收与从机的发送相连，主机产生的时钟信号要输出至从机的时钟引脚上。图 3.26 是两个 DSP 控制器的 SPI 接口相连，其中一个 DSP 控制器被定义为主机，另一个 DSP 控制器被定义为从机，四个引脚的功能如下：

- SPISIMO：若是主机则为发送（输出）；若是从机则为接收（输入）。
- SPISOMI：若是主机则为接收（输入）；若是从机则为发送（输出）。
- SPICLK：串行外设接口时钟。若是主机则是输出时钟；若是从机则是输入时钟。

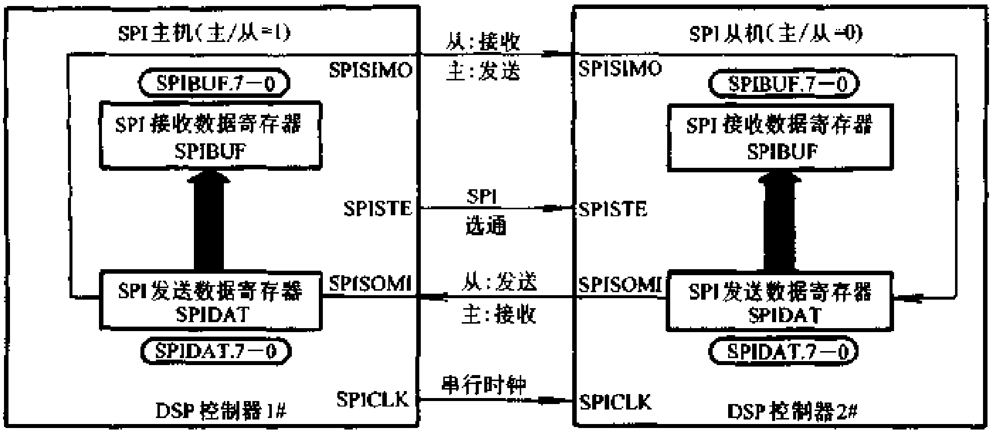


图 3.26 串行外设接口主从控制器的连接

- SPISTE：串行外设接口选通。作为主机，这个引脚的功能就是普通 I/O；作为从机，这个引脚即可以作为 I/O 功能也可以作为选通功能。作为选通功能时，若为高电平将使从机的移位寄存器停止工作且输出引脚呈高阻状态；若为低电平将使能从机的传送功能。因此，可通过这个引脚对从机进行读写控制。

这里要注意的是，除了引脚 SPISTE 是与 I/O 复用的以外，引脚 SPISIMO、SPISOMI 和 SPICLK 都是与 I/O 复用的。引脚功能的选择由 SPI 端口控制寄存器来设置，下面会详细介绍。

由于 SPI 通过一根时钟引线将主机和从机同步，因此它的串行数据交换不需要增加起始位、停止位等用于同步的格式位，直接将要传送的信息（1 位到 8 位的数据）写入到主机的 SPI 发送数据寄存器 SPIDAT；这个写入自动启动了主机的发送过程；即在同步时钟 SPICLK 的节拍下把 SPIDAT 的内容一位一位地移到引脚 SPISIMO；当 SPIDAT 的内容移位完毕，将置一个中断标志 SPIINT FLAG，通知主机这个信息块已发送完毕。

对于从机，同样在同步时钟 SPICLK 的节拍下将出现在引脚 SPISIMO 上的数据一位一

位地移到从机的移位寄存器(与从机的发送数据寄存器SPIDAT是同一个寄存器),当一个完整的信息块接收完成后,将置一个中断标志SPIINT FLAG,通知从机这个信息块已接收完毕,并同时将移位寄存器接收到的内容复制到从机的SPI接收数据寄存器SPIBUF。

如果由从机发送数据、主机接收数据,其过程与上相反。从机的CPU将要发送的数据写入到从机的发送数据寄存器SPIDAT,然后在主机时钟SPICLK的节拍下将数据一位一位地移到引脚SPISOMI上;另一方面,主机在时钟SPICLK的节拍下从引脚SPISOMI上将数据一位一位地移到主机的移位寄存器(即主机的发送数据寄存器SPIDAT),当接收完1个完整的信息块,再将移位寄存器的内容复制到主机的接收数据寄存器SPIBUF中。

前面是SPI的简单工作原理。从这个过程看出,对于用户编程只需关注在发送数据时写SPI发送数据寄存器SPIDAT;在接收数据时读SPI接收数据寄存器SPIBUF。其余的移位、同步、置收发标志等工作都由内置的SPI模块自动完成。下面详细介绍DSP控制器SPI接口模块的结构与使用方法。

图3.27是DSP控制器SPI接口模块的结构示意图。它的结构通过两类寄存器来体现:数据类寄存器和控制类寄存器。数据类寄存器包括SPI发送数据寄存器SPIDAT、SPI接收数据寄存器SPIBUF和SPI仿真接收寄存器SPIEMU;控制类寄存器包括SPI配置控制寄存器SPICCR、SPI操作控制寄存器SPICTL、SPI状态寄存器SPISTS、SPI波特率寄存器SPIBRR、SPI端口控制寄存器SPIPC1与SPIPC2、SPI中断优先级控制寄存器SPIPRI。上述寄存器各位的定义参见表3.26~表3.34。数据类寄存器是CPU与串行外设信息交换的桥

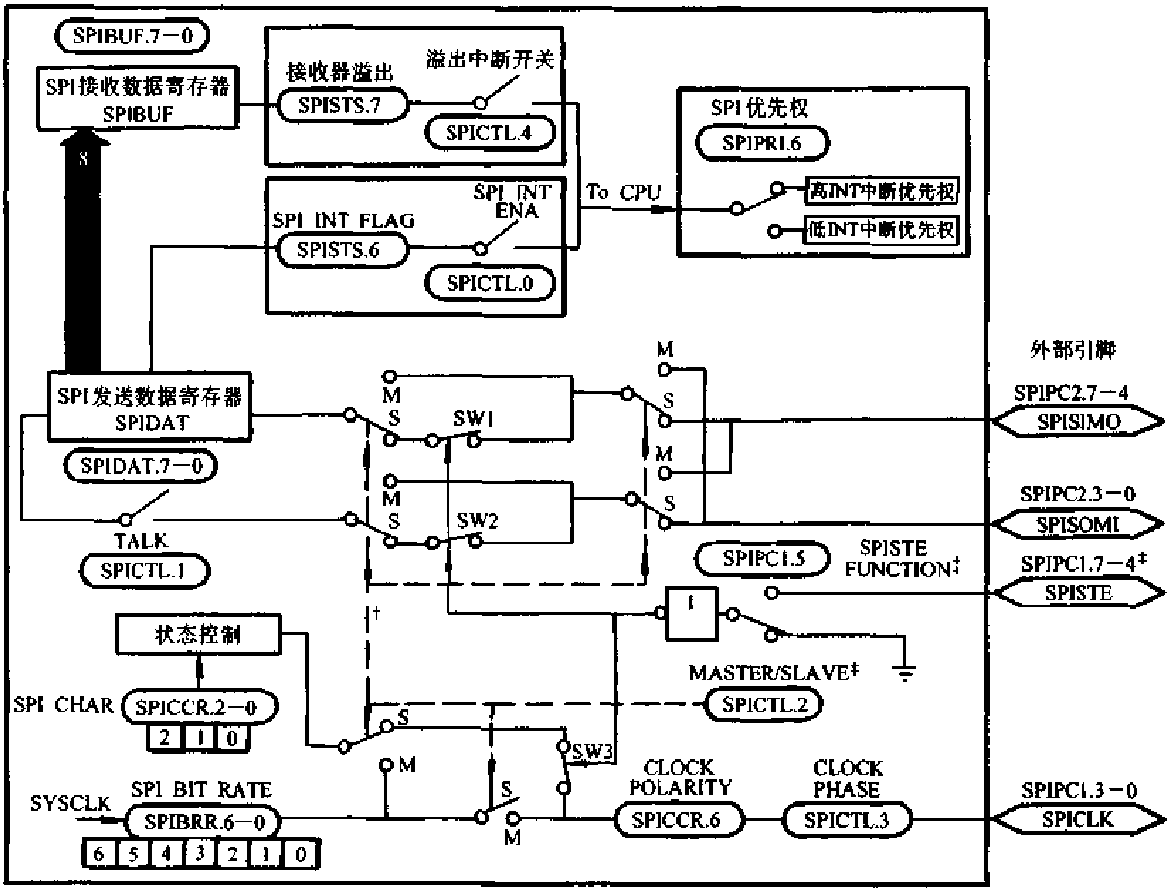


图3.27 DSP控制器SPI接口模块的结构示意图

梁，用户程序只需读写数据类寄存器即可实现对串行外设的访问。在这里要注意的是，SPI仿真接收寄存器SPIEMU与SPI接收数据寄存器SPIBUF的功能基本一样，只是在读SPI仿真接收寄存器SPIEMU时不会影响标志位。控制类寄存器规定模块的工作方式、串行访问的协议参数以及响应标志等，需要在用户程序对其初始化。控制类寄存器与SPI模块的硬件部分联系紧密，因此要了解清楚SPI模块的结构就是要了解清楚它的控制类寄存器的结构和使用方法。

下面先讨论SPI配置控制寄存器SPICCR和SPI操作控制寄存器SPICTL。

●在使用SPI模块之前，首先要确定当前的SPI模块是作为主机还是从机，这由SPICTL的MASTER/SLAVE来决定，该位为1表示是主机，为0表示是从机。

●然后要确定信息块的大小，这由SPICCR的SPICHAR2：0决定，可以在1～8位之间进行选择。

●接着确定SPI同步时钟SPICLK的极性与相位，SPICCR的CLOCK POLARITY为1表示引脚在静止状态时是高电平，否则是低电平；SPICTL的COLCK PHASE为1，SPICLK的相位滞后半个周期，否则不滞后，SPICLK的极性与相位决定了在SPICLK的一个节拍中什么时候将数据移位到引脚上或什么时候从引脚上接收移位来的数据。这2位的四种组合得到表3.25四种SPICLK的时序。总之，发送与接收相差半个周期，即在发送数据稳定半个周期后，接收方才进行采样接收。目前市场上具有SPI接口功能的芯片的收发时序会有不同，因此DSP控制器给出四种SPICLK时序，如图3.28所示，为它与众多的SPI接口芯片可以直接相连提供了方便。

表3.25 四种SPICLK的时序

时序	CLOCK POLARITY	CLOCK PHASE	说　　明
无延时上升沿	0	0	上升沿发送，下降沿接收
有延时上升沿	0	1	上升沿的前半个周期发送，上升沿接收
无延时下降沿	1	0	下降沿发送，上升沿接收
有延时下降沿	1	1	下降沿的前半个周期发送，下降沿接收

●SPI模块的中断有两个：SPINT和OVERRUN INT。前者表示信息块发送完毕或接收完毕；后者表示在上次接收数据还未从SPIBUF中读走又接收到了新的数据。这两个中断共用相同的中断偏移向量，但具有不同的状态标志（SPIINT FLAG和RECEIVER OVERRUN），有关中断偏移向量的内容可参见3.7。如果允许CPU响应这两个中断，必需设置SPICTL的SPIINT ENA和OVERRUN INT ENA为1，否则要将其清0。

●SPI模块的发送功能可以通过编程予以禁止，即SPICTL的TALK清为0将禁止SPI的发送功能，此时发送引脚处于高阻状态，但接收功能不受影响。要注意，在清TALK为0时，如果上次发送还未结束，会继续完成上次的发送，再执行禁止的操作。若要开启发送功能，必需将SPICTL的TALK置为1。

●SCICCR的SPI SW RESET是一个软件复位位。如果该位为1将复位SPI的状态标志，即SPIINT FLAG和RECEIVER OVERRUN被清0，但配置参数保持不变。若是主机，此时引脚SPICLK将返回到CLOCK POLARITY规定的静止状态。在SPI复位期间写入到SPI-

DAT 的数据将不会被发送。复位完成后，要记得将 SPI SW RESET 清 0，以准备进行正常的收发工作。

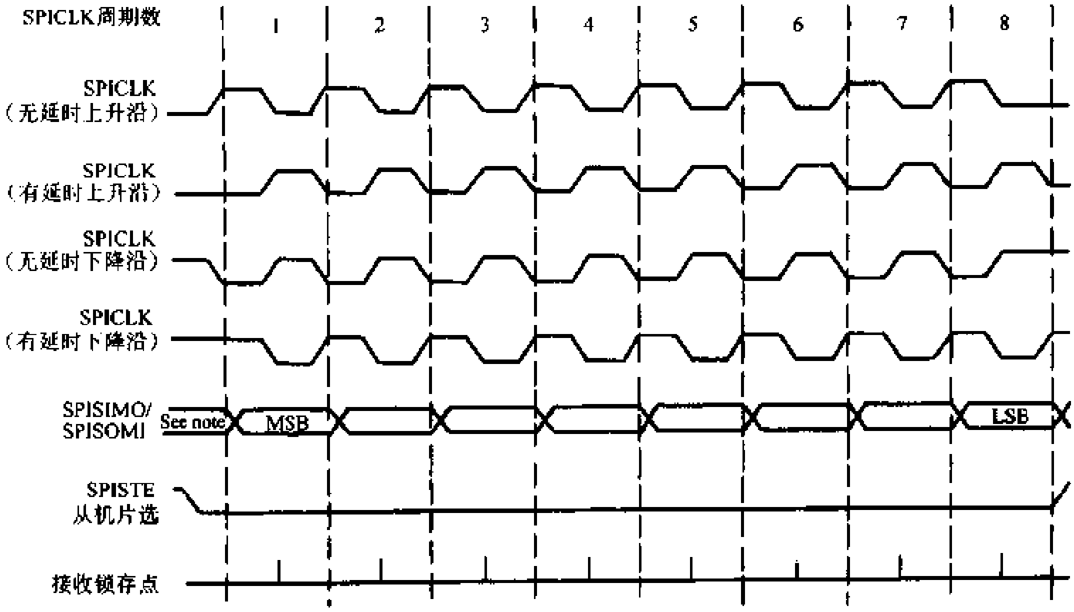


图 3.28　SPICLK 时序图

表 3.26　串行外设模块寄存器

地址	寄存器	位数							
7040h	SPICCR	7	6	5	4	3	2	1	0
		SPISW	CLOCK	Reserved			SPI CHAR2	SPI CHAR1	SPI CHAR0
		RW-0	RW-0				RW-0	RW-0	RW-0
7041h	SPICTL	7	6	5	4	3	2	1	0
		Reserved			OVERRUN INTENA	CLOCK PHASE	MASTER/SLAVE	TALK	SPI INT ENA
					RW-0	RW-0	RW-0	RW-0	RW-0
7042h	SPISTS	7	6	5	4	3	2	1	0
		RECEIVER OVERRUN	SPI INT FLAG	Reserved					
		RC-0	R-0						
7043h	—	Reserved							
7044h	SPIBRR	7	6	5	4	3	2	1	0
		Reserved	SPI BIT RATE6	SPI BIT RATE5	SPI BIT RATE4	SPI BIT RATE3	SPI BIT RATE2	SPI BIT RATE1	SPI BIT RATE0
			RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0
7045h	—	Reserved							
7046h	SPIEMU	7	6	5	4	3	2	1	0
		ERCVD7	ERCVD6	ERCVD5	ERCVD4	ERCVD3	ERCVD2	ERCVD1	ERCVD0
		R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x

（续）

地址	寄存器	位数							
7047h	SPIBUF	7	6	5	4	3	2	1	0
		RCVD7	RCVD6	RCVD5	RCVD4	RCVD3	RCVD2	RCVD1	RCVD0
		R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x
7048h	—	Reserved							
7049h	SPIDAT	7	6	5	4	3	2	1	0
		SDAT7	SDAT6	SDAT5	SDAT4	SDAT3	SDAT2	SDAT1	SDAT0
		RW-x	RW-x	RW-x	RW-x	RW-x	RW-x	RW-x	RW-x
704Ah	—	Reserved							
704Bh	—	Reserved							
704Ch	—	Reserved							
704Dh	SPIPC1	7	6	5	4	3	2	1	0
		SPISTE DATA IN	SPISTE DATA OUT	SPISTE FUNCTION	SPISTE DATA DIR	SPICLK DATA IN	SPICLK DATA OUT	SPICLK FUNCTION	SPICLK DATA DIR
		R-x	RW-0	RW-0	RW-0	R-x	RW-0	RW-0	RW-0
704Eh	SPIPC2	7	6	5	4	3	2	1	0
		SPISIMO DATA IN	SPISIMO DATA OUT	SPISIMO FUNCTION	SPISIMO DATA DIR	SPISOMI DATA IN	SPISOMI DATA OUT	SPISOMI FUNCTION	SPISOMI DATA DIR
		R-x	RW-0	RW-0	RW-0	R-x	RW-0	RW-0	RW-0
704Fh	SPIPRI	7	6	5	4	3	2	1	0
		Reserved	SPI PRIORITY	SPI ESPEN	Reserved				
			RW-0	RW-0					

表 3.27 串行外设接口配置控制寄存器（SPICCR）各位定义与说明

位	定义			说明
Bit7	SPI SW RESET（串行外设接口软件复位）		0	串行外设接口准备发送或接收下一个字符
			1	复位串行外设接口
Bit6	CLOCK POLARITY（SPICLK 的极性）			与 CLOCK PHASE 配合产生 SPICLK 的四种时序，详见表 3.25
Bit5-3	保留			读操作不确定，写操作无效
Bit2-0	SPI CHAR2：0			字符长度控制位，选定的字符长度为：
	0	0	0	1
	0	0	1	2
	0	1	0	3
	0	1	1	4
	1	0	0	5
	1	0	1	6
	1	1	0	7
	1	1	1	8

表 3.28 串行外设接口操作控制寄存器（SPICTL）各位定义与说明

位	定义		说明
Bit7-5	保留		读操作不确定，写操作无效
Bit4	OVERRUN INT ENA（接收溢出中断允许）	0	禁止 RECEIVER OVERRUN 标志位（SPISTS.7）中断
		1	允许 RECEIVER OVERRUN 标志位（SPISTS.7）中断
Bit3	CLOCK PHASE（SPICLK 相位）		与 CLOCK POLARITY 配合产生 SPICLK 的 4 种时序，详见表 3.25
Bit2	MASTER/SLAVE（主/从方式）	0	为从机
		1	为主机
Bit1	TALK（发送允许）	0	禁止传送
		1	允许发送
Bit0	SPI INT ENA（SPIINT）中断允许）	0	SPIINT 禁止中断
		1	SPIINT 允许中断

表 3.29 串行外设接口状态寄存器（SPISTS）各位定义与说明

位	定义		说明
Bit7	RECEIVER OVERRUN（接收溢出标志位）	0	未发生接收溢出
		1	在 SPIBUF 的内容读出之前又接收到新的数据，原数据被丢失。有三种方法之一来清除该位：写 0 到该位；写 1 到 SPI SW RESET（SPICCR.7）；系统复位
Bit6	SPI INT FLAG（SPIINT 中断标志位）	0	未发生 SPIINT 中断
		1	串行外设接口硬件设置该位来表明它已经发送或接收完了最后位，并准备好被服务。该位置位的同时，收到的字符将被放置在接收缓冲器中。有三种方法来清除该位：读取 SPIBUF；将 1 写入 SPI SW RESET（SPICCR.7）；系统复位
Bit5-0	保留		读操作不确定，写操作无效

表 3.30 串行外设接口波特率寄存器（SPIBRR）各位定义与说明

位	定义	说明
Bit7	保留	读操作不确定，写操作无效
Bit6-0	SPI BIT RATE6：0	该 7 位为 SPI 波特率寄存器值

表 3.31 串行外设接口优先级控制寄存器（SPIPRI）各位定义与说明

位	定义		说明
Bit7	保留		读操作不确定，写操作无效
Bit6	SPI PRIORITY（中断优先级选择）	0	设为高优先级中断
		1	设为低优先级中断
Bit5	SPI ESPEN（仿真挂起允许）	0	当仿真挂起时，SPI 继续工作至当前的发送/接收序列完成
		1	当仿真挂起时，SPI 的状态被冻结使得它可在仿真器挂起点接受检查
Bit4-0	保留		读操作不确定，写操作无效

在完成 SPI 配置控制寄存器 SPICCR 和 SPI 操作控制寄存器 SPICTL 的初始化工作后，就要设置 SPI 的波特率，即时钟 SPICLK 的周期。时钟 SPICLK 是时钟源模块产生的 SYSCLK 时钟经 SPI 波特率寄存器 SPIBRR 规定的分频系数分频后得到。因此，SPI 的波特率为：

$$\text{SPI 波特率}=\begin{cases}\dfrac{\text{SYSCLK}}{\text{SPIBRR}+1} & 3\leqslant \text{SPIBRR}\leqslant 127\\[2ex] \dfrac{\text{SYSCLK}}{4} & 0\leqslant \text{SPIBRR}<3\end{cases}$$

SYSCLK 的计算参见 2.7。要注意，SPI 模块工作在从机方式时，允许的最大波特率是 SYSCLK/8。

SPI 状态寄存器 SPISTS 给出两个 SPI 状态标志：SPIINT FLAG 和 RECEIVER OVERRUN。这两个标志反映了 SPI 的工作状况，用户程序需根据这两个标志来维持正常的 SPI 收发工作。

如果当前的信息块已发送完或已收到一个完整的信息块，由硬件将状态标志 SPINT FLAG 置为 1（收发共用同一个标志），此时若中断 SPIINT 没有被屏蔽，还将产生中断请求。该标志是一个只读标志，在下面三种情况下被复位：

●读 SPI 接收数据寄存器 SPIBUF。

●写 1 到 SPI SW RESET，引起 SPI 软件复位。

●系统复位。

如果在 SPI 接收数据寄存器 SPIBUF 的内容未读出之前又接收到了新的数据，将使接收溢出标志 RECEIVER OVERRUN 置 1，此时若中断 OVERRUNINT 没有被屏蔽，还将产生中断请求。该标志可由软件清 0（写 0 到该位上）。当由 SPI SW RESET 引起 SPI 软件复位或系统复位时，该标志将变为 0。要注意，每次溢出标志 RECEIVER OVERRUN 置 1，只产生一次中断请求，尽管溢出标志 RECEIVER OVERRUN 未清 0，也不再产生新的中断请求，这与其它中断有点不同。但是，由于它与中断 SPIINT 共用同一个中断偏移向量，为了不引起混淆，最好在中断服务子程序中由软件将溢出标志 RECEIVER OVERRUN 清 0。

SPI 模块的中断优先级是可以编程的。当 SPI 优先级控制寄存器 SPIPRI 的 SPIPRIORITY 为 0 时，SPI 的中断是高优先级；否则，是低优先级。

3.4.2 SPI 的多机通信

与异步串行通信 SCI 一样，SPI 也是一种串行通信的总线，因此可在 SPI 总线上挂接多台设备，如图 3.29 所示。一般情况下，挂接在串行总线的从机的发送端不能同时处在工作状态，否则总线不能正常工作。因此挂接在串行总线的从机的发送端一般处在高阻状态，当需要进行通信时才激活进入工作状态，而且每次只能激活一台从机。为了使多机通信正常进行，与 SCI 的多机通信一样，需要对每个设备分配一个地址。主机与某个从机进行通信之前，先在总线上广播这个从机的地址信息。每个从机都可以收到这个地址信息，然后与自己的地址进行比较，相同就将自己发送端激活，即将 SPICTL 的 TALK 置 1，否则将它清 0。这样就可以保证在每一个时刻主机只与 SPI 总线上的一个从机进行数据通信。

上面谈到的是 SPI 多机通信的原理，但有一个问题是怎样进行地址广播，或者说怎样区分地址信息与数据信息？在这一点上 SPI 与 SCI 略有不同，因为 SPI 每次传送的信息块不再

含有起始、停止、地址/数据等格式位，因此以地址位的方式来区分有些不便；另外，采用空闲状态时间的长度来区分，对于SPI模块也是不便利的。因此，采用SPI串行通信一般都是基于一种高层协议来进行，例如面向字符的链路控制协议。面向字符的链路控制协议是一种国际标准协议，很多应用场合都采用这种协议。面向字符的链路控制协议是由一些控制字符来构成报文的格式，这些控制字符如表3.32所示。

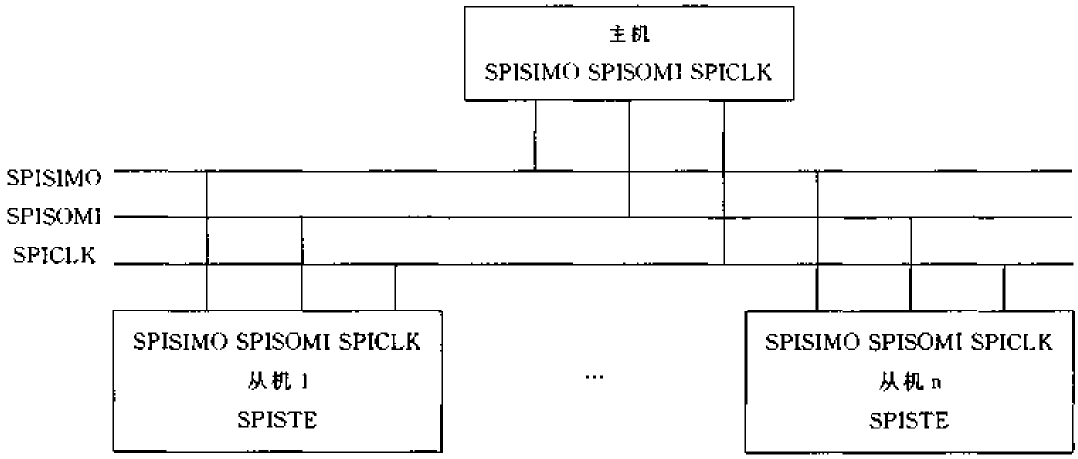


图3.29 SPI的多机通信

表3.32 面向字符的链路控制协议的控制字符

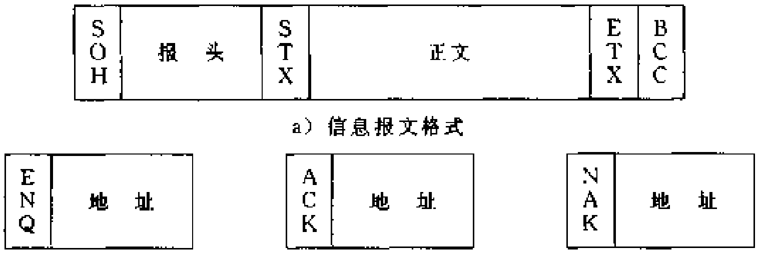
符　号	名　称	ASCII	功　能
SOH	报头始	01h	表示信息报文的报头开始
STX	正文始	02h	表示报头结束，正文开始
ETX	正文终	03h	表示以STX开始的正文结束
EOT	传输结束	04h	通知对方，传输结束
ENQ	询问	05h	询问对方，要求响应
ACK	确认	06h	接收端对发送端的肯定回答
NAK	否认	16h	接收端对发送端的否定回答
DLE	转义	10h	表明其后的字符不是控制字符

面向字符的链路控制协议的信息报文和控制报文格式如图3.30所示，其中BCC是校验块，采用何种校验方式由用户编程决定。这里要说明的是，在实际应用时可以有报头也可以不要报头，只要参与通信的设备遵循同样的规则即可。另外，报头或正文里的数据、字母应以ASCII码表示，以免与控制字符相同引起传输混乱。如果报头或正文里的数据、字母不能以ASCII码表示，当其中出现与控制字符一样的数据时则需在这个数据前加上转义字符DLE，接收方在接收报头或正文时遇上DLE时要将其去掉以还原原来的报文。这种做法称为透明传输。控制报文含有地址信息，当每一个从机收到询问的控制报文时，与自己的地址比较，相同则返回确认的控制报文并激活发送端，从而实现主机与从机传输链路的开通。

在前面介绍的面向字符的链路控制协议的基础上，可以实现SPI多机通信的功能，其中主机与从机的工作流程如图3.31。

从前面的讨论不难看出，面向字符的链路控制协议是一种很通用的链路层通信协议，在

SCI 的通信方式中也可采用。另外，有一些 SPI 接口的芯片有自己特定的通信协议，因此在与这些芯片相连时要首先弄清楚它们的通信协议。



a）信息报文格式

b）控制报文格式

图 3.30　面向字符的链路控制协议的信息报文和控制报文格式

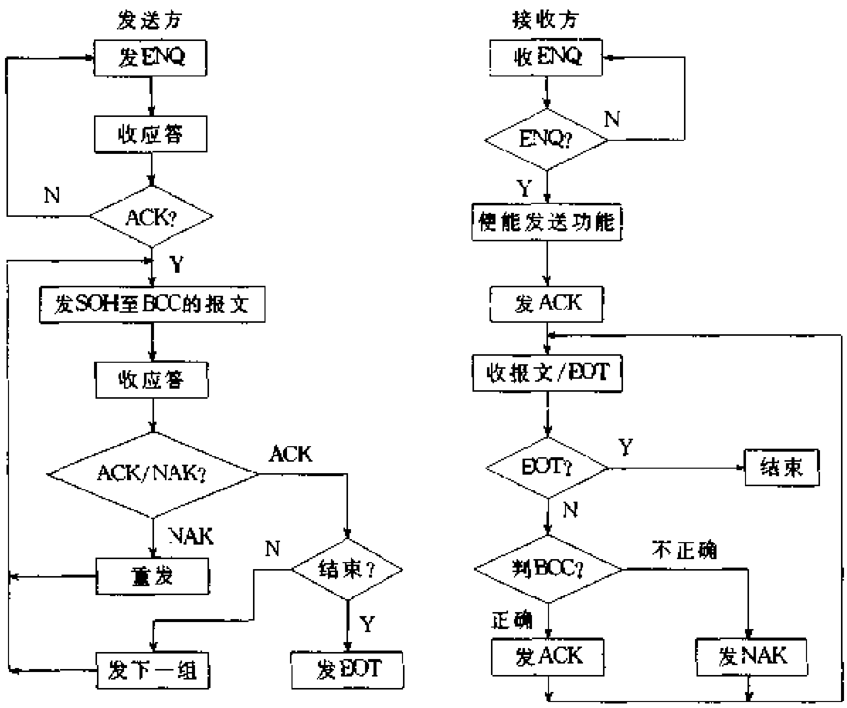


图 3.31　SPI 多机通信的工作流程

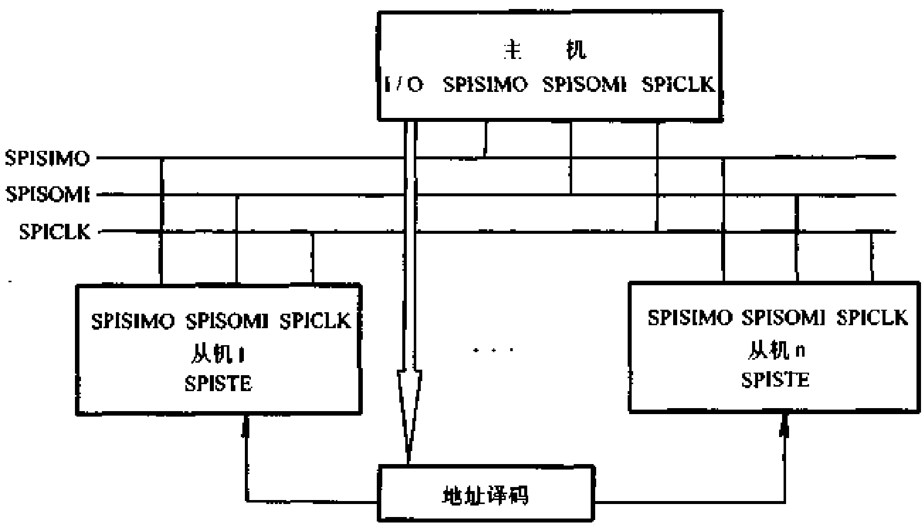


图 3.32　带地址片选线的 SPI 多机通信

SPI多机通信除了采用图3.29的总线方式外，还可以采用图3.32的总线方式。后者比前者增加了一根地址片选线，这样一来使得多机通信的地址确认更为简单，传输的效率得到提高。主机在跟某一个从机进行通信之前，首先向I/O端口写从机的地址，然后经地址译码将对应的从机的选通引脚SPISTE激活(低有效)，这样一来使得该从机的发送功能自动开启，而其余的从机被译码电路封锁(对应的引脚SPISTE为高电平)。在这个工作完成后，剩下的工作就与点——点的通信一样了，当然也可以采用前面的面向字符的链路控制协议进行通信工作。

3.4.3 SPI引脚功能的选择

前面已经提到SPI的四个引脚是功能复用的，即可以作为SPI的功能引脚也可以是普通的I/O引脚。因此，在使用SPI模块之前需要对引脚的功能进行选择。SPI端口控制寄存器SPIPC1和SPIPC2就是为此服务的，它们各位的定义参见表3.26和表3.33、表3.34。

表3.33 SPI端口控制寄存器SPIPC1各位定义与说明

位	定义		说明	
Bit7	SPISTE DATE IN		引脚SPISTE用作I/O功能时的输入数据位，即从该位读该引脚的当前值。向该位写操作无效	
Bit6	SPISTE DATE OUT		引脚SPISTE用作I/O功能时的输出数据位，即写到该位的值将反应到该引脚的电平上	
Bit5	SPISTE FUNCTION	0	引脚SPISTE用作I/O功能	
		1	引脚SPISTE用作SPI的从机片选功能。在主机方式下该位写1无效	
Bit4	SPISTE DATE DIR	0	被配置成输入引脚	引脚SPISTE用作I/O功能时的方向位
		1	被配置成输出引脚	
Bit3	SPICLK DATE IN		引脚SPICLK用作I/O功能时的输入数据位，即从该位读该引脚的当前值。向该位写操作无效	
Bit2	SPICLK DATE OUT		引脚SPICLK用作I/O功能时的输出数据位，即写到该位的值将反应到该引脚的电平上	
Bit1	SPICLK FUNCTION	0	引脚SPICLK用作I/O功能	
		1	引脚SPICLK用作SPI时钟功能	
Bit0	SPICLK DATE DIR	0	被配置成输入引脚	引脚SPICLK用作I/O功能时的方向位
		1	被配置成输出引脚	

表3.34 SPI端口控制寄存器SPIPC2各位定义与说明

位	定义		说明
Bit7	SPISIMO DATE IN		引脚SPISIMO用作I/O功能时的输入数据位，即从该位读该引脚的当前值。向该位写操作无效
Bit6	SPISIMO DATE OUT		引脚SPISIMO用作I/O功能时的输出数据位，即写到该位的值将反应到该引脚的电平上
Bit5	SPISIMO FUNCTION	0	引脚SPISIMO用作I/O功能
		1	引脚SPISIMO用作SPI功能

（续）

位	定　义		说　明	
Bit4	SPISIMO DATE DIR	0	被配置成输入引脚	引脚 SPISIMO 用作 I/O 功能时的方向位
		1	被配置成输出引脚	
Bit3	SPISOMI DATE IN		引脚 SPISOMI 用作 I/O 功能时的输入数据位，即从该位读该引脚的当前值。向该位写操作无效	
Bit2	SPISOMI DATE OUT		引脚 SPISOMI 用作 I/O 功能时的输出数据位，即写到该位的值将反应到该引脚的电平上	
Bit1	SPISOMI FUNCTION	0	引脚 SPISOMI 用作 I/O 功能	
		1	引脚 SPISOMI 用作 SPI 功能	
Bit0	SPISOMI DATE DIR	0	被配置成输入引脚	引脚 SPISOMI 用作 I/O 功能时的方向位
		1	被配置成输出引脚	

SPI 端口控制寄存器 SPIPC1 和 SPIPC2 分别设置 SPISTE、SPICLK、SPISIMO、SPISOMI 四个引脚的功能。其中每个引脚由 4 位来设置，它们分别是：

●功能选择位（FUNCTION）。该位为 0 选择 I/O 功能，该位为 1 选择 SPI 的功能。

●I/O 方向位（DATE DIR）。如果选择 I/O 功能，就要确定是作为输入还是作为输出，如果 I/O 方向位为 0 就配置为输入，否则就配置为输出。

●输入数据位（DATE IN）。当引脚配置为输入的 I/O 引脚时，由输入数据位来读取当前引脚的电平状态；

●输出数据位（DATE OUT）。当引脚配置为输出的 I/O 引脚时，则向输出数据位写 1 将使当前引脚为高电平，写 0 将使当前引脚为低电平。

3.5 数字 I/O 端口

数字 I/O 是单片机与外界联系的接口。DSP 控制器的数字 I/O 引脚都是功能复用的，即可作普通 I/O 用，也可作其它功能。对于普通 I/O，即可作为输入也可作为输出。因此，DSP 控制器的数字 I/O 引脚对应着二类寄存器：控制类寄存器和数据类寄存器。前者指出某个引脚是作普通 I/O 用还是作为特殊功能用；后者指出作为普通 I/O 时的数据方向，是输入还是输出，以及当前引脚对应的电平（数据）。要注意，读引脚的电平或向引脚输出电平，实际上都是对相应的寄存器进行读写。下面详细介绍 DSP 控制器数字 I/O 的结构和对应的寄存器。

3.5.1 数字 I/O 端口概述

DSP 控制器共有 28 个功能复用的双向 I/O 引脚，这些引脚分为 2 组：

●组 1：共 20 个引脚，与单比较、全比较、捕获、A/D 等模块功能复用，分属端口 A、端口 B 和端口 C，见表 3.35。

●组 2：共 8 个引脚，与 SCI、SPI、外部中断和 PLL 时钟源等模块功能复用，见表 3.36。

表 3.35 共享 I/O 引脚组 1

引脚号	MUX 控制寄存器	引脚功能选择		端口数据和方向		
		CRx.n=1	CRx.n=0	寄存器	数据位	方向控制位
72	CRA.0	ADCIN0	IOPA0	PADATDIR	0	8
73	CRA.1	ADCIN1	IOPA1	PADATDIR	1	9
91	CRA.2	ADCIN9	IOPA2	PADATDIR	2	10
90	CRA.3	ADCIN8	IOPA3	PADATDIR	3	11
100	CRA.8	PWM7/CMP7	IOPB0	PBDATDIR	0	8
101	CRA.9	PWM8/CMP8	IOPB1	PBDATDIR	1	9
102	CRA.10	PWM9/CMP9	IOPB2	PBDATDIR	2	10
105	CRA.11	T1PWM/T1CMP	IOPB3	PBDATDIR	3	11
106	CRA.12	T2PWM/T2CMP	IOPB4	PBDATDIR	4	12
107	CRA.13	T3PWM/T3CMP	IOPB5	PBDATDIR	5	13
108	CRA.14	TMRDIR	IOPB6	PBDATDIR	6	14
109	CRA.15	TMRCLK	IOPB7	PBDATDIR	7	15
63	CRB.0	ADCSOC	IOPC0	PCDATDIR	0	8
64	SCR.7-6					
	00	IOPC1		PCDATDIR	1	9
	01	CLKOUT (Watchdog clock)				
	10	CLKOUT (SYSCLK)				
	11	CLKOUT (CPUCLK)				
65	CRB.2	IOPC2	XF	PCDATDIR	2	10
66	CRB.3	IOPC3	$\overline{BIO}$	PCDATDIR	3	11
67	CRB.4	CAP1/QEP1	IOPC4	PCDATDIR	4	12
68	CRB.5	CAP2/QEP2	IOPC5	PCDATDIR	5	13
69	CRB.6	CAP3	IOPC6	PCDATDIR	6	14
70	CRB.7	CAP4	IOPC7	PCDATDIR	7	15

表 3.36 共享引脚组 2

引脚号	主功能	外设模块	引脚号	主功能	外设模块
43	SCIRXD	SCI	49	SPICLK	SPI
44	SCITXD	SCI	51	SPISTE	SPI
45	SPISIMO	SPI	54	XINT2	外部中断引脚
48	SPISOMI	SPI	55	XINT3	外部中断引脚

由于单比较、全比较、捕获、A/D 模块引脚的结构不具备双向 I/O 的性质，因此组 1 的 I/O 引脚功能是专配的，由切换开关进行切换，其结构如图 3.33 所示。每个引脚有 3 位定义它的操作：

●MUX 控制位——此位选择引脚功能，置 1 时为对应模块的功能引脚，置 0 时为 I/O 引脚。

●I/O 方向位——在引脚设置为 I/O 功能时（MUX 控制位置 0），此位确定引脚数据方向，置 0 时为输入引脚，置 1 时为输出引脚。

●I/O 数据位——在引脚设置为 I/O 功能时（MUX 控制位置 0），且引脚方向设置为输入，则数据从此位读入；若引脚方向设置为输出，则数据被写入该位。该位的数据与引脚的电平是一一对应的。

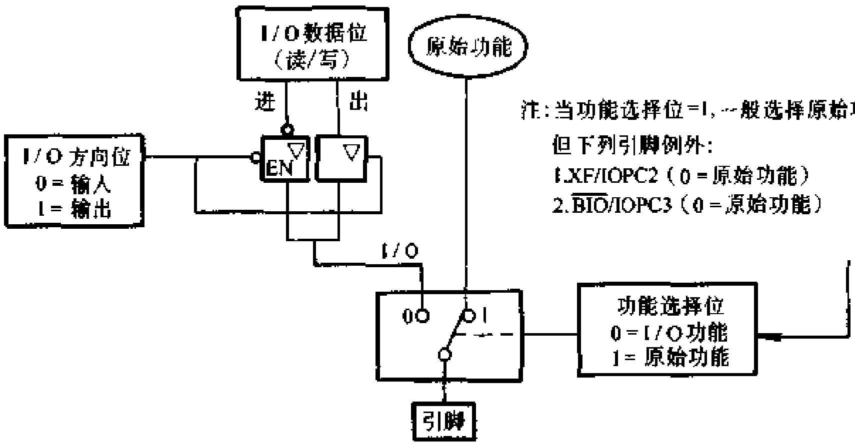


图 3.33　组 1 的复用 I/O 引脚结构示意图

由于 SCI、SPI、外部中断和 PLL 时钟源模块的引脚结构具有双向 I/O 性质，即内置 I/O 功能。因此组 2 的 I/O 引脚不再需要专配 I/O 电路，通过设置相应外设的控制寄存器来设置这组引脚功能即可。

3.5.2　数字 I/O 端口寄存器

对于组 1 的数字 I/O 有二类寄存器：控制寄存器和数据方向寄存器，如表 3.37 所示。其中 I/O 控制寄存器 OCRA 和 OCRB 用来选择端口 A、端口 B 和端口 C 的引脚功能；I/O 数据方向寄存器 PADATDIR、PBDATDIR 和 PCDATDIR 分别用来设置端口 A、端口 B 和端口 C 各引脚的数据方向（输入还是输出），以及读写各引脚对应的电平状态。在对 I/O 控制寄存器、引脚的数据方向初始化完成后，用户程序主要就是通过读写 PADATDIR、PBDATDIR 和 PCDATDIR 的相应位来输入和输出引脚的电平。这些寄存器各位的详细定义见表 3.37，使用方法如下：

●首先确定引脚的功能，即 I/O 控制寄存器 OCRA 和 OCRB 中的 CRx.n（x=A，B；n=0～15）为 1 表示引脚功能是原模块的功能，否则为 I/O 功能。

●如果引脚被配置为 I/O 功能，就需要确定它的方向：输入还是输出。即 I/O 数据方向寄存器 PADATDIR、PBDATDIR 和 PCDATDIR 中的 xnDIR（x=A，B，C；n=0～7）为 1 表示是输出引脚，否则是输入引脚。

●对于 I/O 功能的输入或输出是通过读写 I/O 数据方向寄存器 PADATDIR、PBDATDIR 和 PCDATDIR 中的 IOPxn（x=A，B，C；n=0～7）来实现的。输入引脚对应读操作；输出引脚对应写操作。

表 3.37 数字 I/O 端口寄存器

地址	寄存器	位数							
7090h	OCRA	15	14	13	12	11	10	9	8
		CRA.15	CRA.14	CRA.13	CRA.12	CRA.11	CRA.10	CRA.9	CRA.8
		RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0
		7	6	5	4	3	2	1	0
		Reserved				CRA.3	CRA.2	CRA.1	CRA.0
						RW-0	RW-0	RW-0	RW-0
7092h	OCRB	15	14	13	12	11	10	9	8
		Reserved							
		7	6	5	4	3	2	1	0
		CRB.7	CRB.6	CRB.5	CRB.4	CRB.3	CRB.2	保留	CRB.0
		RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0
7098h	PADATDIR	15	14	13	12	11	10	9	8
		Reserved				A3DIR	A2DIR	A1DIR	A0DIR
						RW-0	RW-0	RW-0	RW-0
		7	6	5	4	3	2	1	0
		Reserved				IOPA3	IOPA2	IOPA1	IOPA0
						RW-0	RW-0	RW-0	RW-0
709Ah	PBDATDIR	15	14	13	12	11	10	9	8
		B7DIR	B6DIR	B5DIR	B4DIR	B3DIR	B2DIR	B1DIR	B0DIR
		RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0
		7	6	5	4	3	2	1	0
		IOPB7	IOPB6	IOPB5	IOPB4	IOPB3	IOPB2	IOPB1	IOPB0
		RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0
709Ch	PCDATAIR	15	14	13	12	11	10	9	8
		C7DIR	C6DIR	C5DIR	C4DIR	C3DIR	C2DIR	C1DIR	C0DIR
		RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0
		7	6	5	4	3	2	1	0
		IOPC7	IOPC6	IOPC5	IOPC4	IOPC3	IOPC2	IOPC1	IOPC0
		RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

3.6 看门狗与实时时钟

单片机控制系统在实际应用时，抗干扰是必须要解决的问题。这个问题比较复杂，一方面要查清干扰的主要来源，另一方面要采取相应措施予以防范。在单片机应用系统中，由于干扰的因素或硬件设备的故障，常常会出现程序“跑飞”或“死机”现象，使系统不能正常工作。为了解决这个问题，在单片机采用看门狗与实时时钟是一个很好的办法。

看门狗实际上就是一个定时器，它独立地运行，一旦定时器溢出就会复位系统。因此，在

应用程序的各个分支上，要按时“喂狗”，即将看门狗的计数器清 0，不让它发生溢出。这样就可以保证，只要应用系统是按照程序设计的路径走，看门狗就不会强行复位系统。但若程序“跑飞”或“死机”，就不可能按时“喂狗”，从而看门狗就会产生溢出强行复位系统，将应用系统重新纳入正轨。

实时时钟也是一个定时器，也可等效地作为看门狗（实际上，只要是定时器都可以这样做），但它更多的是为系统提供一个连续不断的时钟服务。有了实时时钟，可以容易实现日历与实时时间的功能，实现闹钟的功能，实现分时多任务进程控制的功能等等。

图 3.34 是 DSP 控制器看门狗与实时时钟的结构示意图。看门狗包含：

●一个 8 位看门狗计数寄存器（WDCNTR）。溢出时强行复位系统；通过“喂狗”指令清 0。

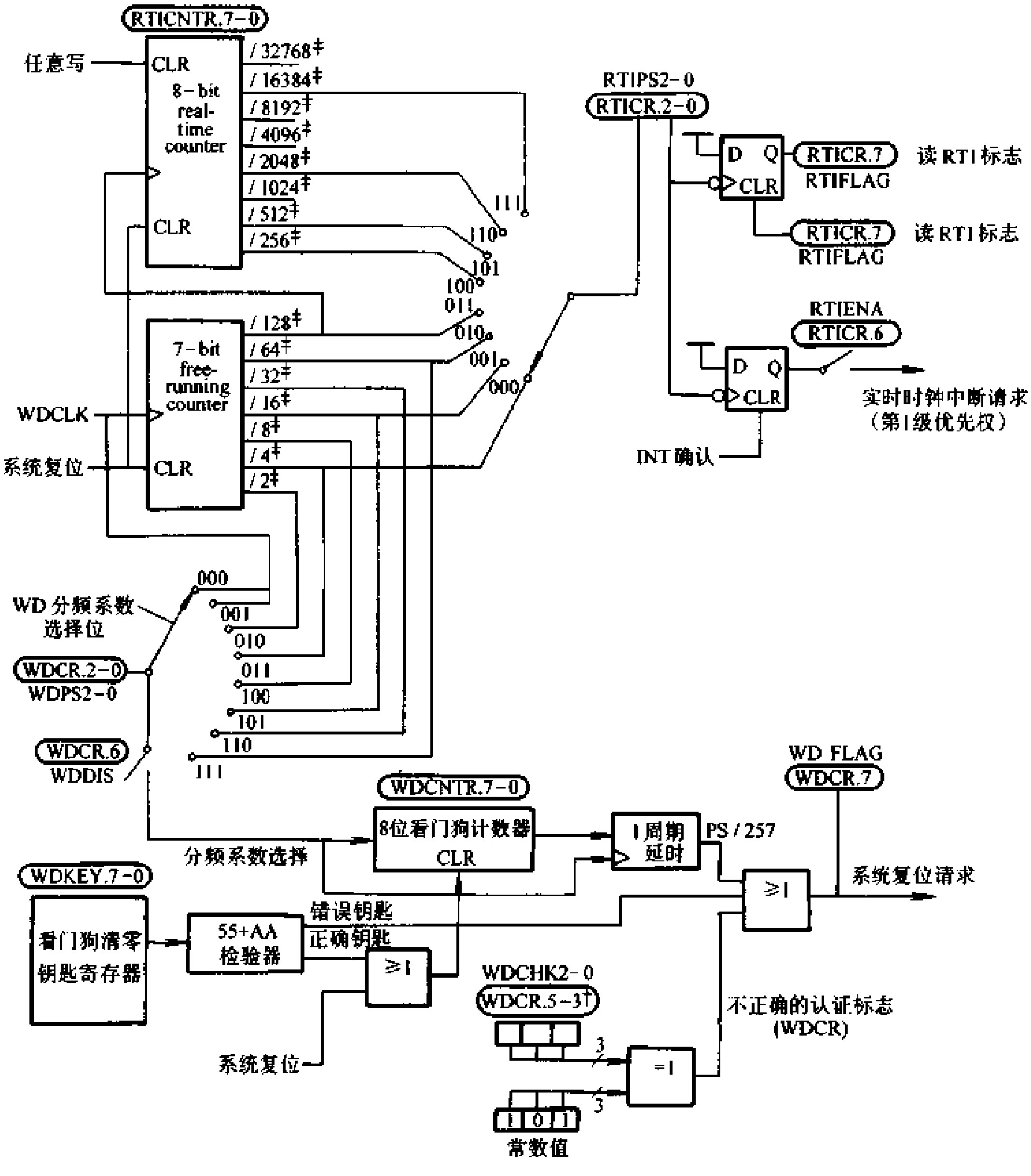


图 3.34 看门狗与实时时钟的结构示意图

●一个7位循环计数器，为看门狗与实时时钟提供分频时钟信号。该计数器自系统复位就循环计数，没有指令可更改它的计数值。

●一个“喂狗”用的看门狗清0钥匙寄存器（WDKEY）。当正确的组合指令写入将使看门狗清0；否则，将强行复位系统。

●一个8位看门狗控制寄存器（WDCR），包含1个看门狗使能位、3个看门狗认证位、3个分频系数位和1个中断标志位。

实时时钟包含：

●一个8位实时时钟计数寄存器（RTICNTR）。它和7位循环计数器一起构成一个15位的计数器，实时时钟何时中断取决这个15位计数器的计数值是否为下列8个值中的一个：4，16，64，128，256，512，2048，16384。前4个值由7位循环计数器提供，后4个值由实时时钟计数寄存器提供。

●一个8位实时时钟控制寄存器（RTICR）。它决定实时时钟何时可以中断，即由定标位从前列8个值中确定一个为基准，当15位的计数器的值与基准相等时便产生中断。另外，还包含1个使能位和1个中断标志位。

表3.38是看门狗与实时时钟的寄存器组成图。表3.39是看门狗控制寄存器（WDCR）各位定义与说明，表3.40是实时时钟控制寄存器（RTICR）各位定义与说明。

表3.38 看门狗与实时时钟的寄存器组成

地址	寄存器	位	数						
7020h	—	Reserved							
7021h	RTICNTR	7	6	5	4	3	2	1	0
		D7	D6	D5	D4	D3	D2	D1	D0
		R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
7022h	—	Reserved							
7023h	WDCNTR	7	6	5	4	3	2	1	0
		D7	D6	D5	D4	D3	D2	D1	D0
		R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
7024h	—	Reserved							
7025h	WDKEY	7	6	5	4	3	2	1	0
		D7	D6	D5	D4	D3	D2	D1	D0
		RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0
7026h	—	Reserved							
7027h	RTICR	7	6	5	4	3	2	1	0
		RTIFLAG	RTIENA	Reserved			RTIPS2	RTIPS1	RTIPS0
		RW-0	RW-0				RW-0	RW-0	RW-0
7028h	—	Reserved							
7029h	WDCR	7	6	5	4	3	2	1	0
		WDFLAG	WDDIS	WDCHK2	WDCHK1	WDCHK0	WDPS2	WDPS1	WDPS0
		RW-x		RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

表 3.39　看门狗控制寄存器（WDCR）各位定义与说明

位	定　　义			说　　明
Bits7	WDFLAG		0	未产生看门狗中断
			1	产生了看门狗中断
Bits6	WDDIS		0	开启看门狗
			1	关闭看门狗
Bits5-3	WDCHK2：0=101			看门狗认证位
Bits2-0	WDPS2：0			
	0	0	0	WDCLK 的分频系数=1
	0	0	1	WDCLK 的分频系数=1
	0	1	0	WDCLK 的分频系数=2
	0	1	1	WDCLK 的分频系数=4
	1	0	0	WDCLK 的分频系数=8
	1	0	1	WDCLK 的分频系数=16
	1	1	0	WDCLK 的分频系数=32
	1	1	1	WDCLK 的分频系数=64

表 3.40　实时时钟控制寄存器（RTICR）各位定义与说明

位	定　　义			说　　明		
Bits7	RTIFLAG		0	未产生实时时钟中断		
			1	产生了实时时钟中断		
Bits6	RTIENA		0	清除目前尚未应答的中断，并关闭以后的中断		
			1	允许中断		
Bits5-3	保　　留					
Bits2-0	RTIPS2：0			基准值（WDCLK 的分频数）	输出频率 Hz（WDCLK=16384Hz）	输出频率 Hz（WDCLK=15625Hz）
	0	0	0	4	4096	3906.25
	0	0	1	16	1024	976.56
	0	1	0	64	256	244.14
	0	1	1	128	128	122.07
	1	0	0	256	64	61.04
	1	0	1	512	32	30.52
	1	1	0	2048	8	7.63
	1	1	1	16384	1	0.95

看门狗与实时时钟由系统时钟源模块的 WDCLK 提供输入时钟。看门狗计数寄存器（WDCNTR）的输入脉冲由 WDCLK 经分频产生，分频系数由看门狗控制寄存器（WDCR）设置。实时时钟计数寄存器（RTICNTR）的输入脉冲由 7 位循环计数器的溢出脉冲提供，因此它们组成一个 15 位的计数器。7 位循环计数器自系统加电复位后，便循环计数，不受其它指

令的影响。看门狗计数寄存器和实时时钟计数寄存器可以清 0，但循环计数器不能，除非系统复位。由于 WDCLK 只在系统空闲方式 3（振荡器掉电）下停止，所以看门狗与实时时钟在正常工作方式以及系统空闲方式 0、1、2 都能作用。

在看门狗计数寄存器（WDCNTR）溢出之前要进行“喂狗”，否则就会强行复位系统。“喂狗”的操作就是对看门狗清 0 钥匙寄存器（WDKEY）写入一个正确的序列：先写 55H，紧接着写 AAH。如果写入序列不正确，也会强行复位系统。看门狗计数寄存器（WDCNTR）溢出或写入序列不正确，都会在看门狗控制寄存器 WDCR 中置标志 WDFLAG。在发生看门狗中断后，需由软件将标志 WDFLAG 清 0，否则将再次进入该中断。

如果不想使用看门狗，一方面要将看门狗控制寄存器 WDCR 中的 WDDIS 置 1；另一方面在系统加电复位期间，引脚 VCCP 必需施加 5V 高电平。只有在这两个条件满足的情况下，才可能关闭看门狗。

看门狗控制寄存器 WDCR 中的 WDCHK2：0 是看门狗的认证位，正常情况下为 101。如果由于某种干扰（尖峰脉冲、电磁干扰等）或人为改写了 WDCHK2：0，不再是 101，将会强行复位系统。

实时时钟的定时时间是可以编程的，与通用定时器不一样，定时时间不能连续设置，只能分 8 级，由实时时钟控制寄存器（RTICR）的 RTIPS2：0 来选择。当实时时钟计数寄存器（RTICNTR）和 7 位循环计数器组成的 15 位计数器的值与选择的基准值相等时便产生实时时钟中断，并将实时时钟控制寄存器（RTICR）的 RTIFLAG 置 1。实时时钟中断标志 RTI-FLAG 由软件清 0。

由于 7 位循环计数器自系统加电后不停地计数，因此实时时钟的第一次定时时间是不确定的，这一点在编程时要注意。

3.7 中断管理系统

中断是计算机一种特殊的运行方式。在正常情况下 CPU 按照程序预定的路线运行；当外围设备（片内或片外）有事件产生需要 CPU 来处理，即发出中断请求信号，CPU 暂停工作，保存好现场；然后转向到该中断请求对应的服务子程序的入口处；待服务子程序运行完毕，CPU 自动恢复现场，从原停顿点继续往下运行。计算机采用中断方式，可以节省 CPU 资源，CPU 可以不花时间去轮寻外围设备是否要服务。每一种计算机都有多个中断源，CPU 对中断的响应也需要按序进行，因此需要一个中断管理系统模块对中断源进行管理控制。

DSP 控制器的中断由 DSP 内核中断、事件管理模块的中断和系统模块中断组成。DSP 内核中断包括：由指令 INTR、NMI 和 TRAP 产生的软件中断和来自复位 $\overline{RS}$、非屏蔽 NMI 和可屏蔽 INTx（x=1，2，3，4，5，6）的硬件中断；事件管理模块的中断包括：通用定时器的周期事件中断、通用定时器的比较事件中断、通用定时器的溢出事件中断、单比较中断、全比较中断、捕获中断和电源驱动保护中断；系统模块中断包括：A/D 转换中断、串行通信 SCI 的接收中断、串行通信 SCI 的发送中断、串行外设接口 SPI 中断、外部引脚 XINTx（x=1，2，3）产生的可屏蔽中断和外部非屏蔽引脚 NMI 中断。它们之间的关系如图 3.35a。

3.7.1 DSP 内核中断

DSP 控制器的 CPU 可以直接响应 DSP 内核中断，但事件管理模块的中断和系统模块中

断要通过 DSP 内核中断与 CPU 挂接。DSP 内核中断分为软件中断和硬件中断：

●软件中断：由指令 INTR、NMI 和 TRAP 产生。

●硬件中断：由来自物理设备的信号产生，包括复位$\overline{RS}$、非屏蔽中断 NMI 和可屏蔽中断 INT1、INT2、INT3、INT4、INT5、INT6。

可屏蔽中断是指可以通过软件将它们禁止（屏蔽）或允许（使能）的中断，这样一来可以通过屏蔽的方法禁止掉那些不想响应的中断。非屏蔽中断是不能通过软件将它们禁止掉的中断，非屏蔽中断包括所有软件中断以及二个外部引脚复位$\overline{RS}$、非屏蔽中断 NMI 产生的中断。

若允许 CPU 响应可屏蔽中断 INTx（x=1，2，3，4，5，6），则需在中断屏蔽寄存器 IMR（参见表 3.41）中的 INTx 位置 1；否则，应将其清 0。在程序初始化时，必需设置中断屏蔽寄存器 IMR，中断屏蔽寄存器 IMR 的复位状态是不确定的。

如果产生了可屏蔽中断 INTx（x=1，2，3，4，5，6），则会在中断标志寄存器 IFR（参见表 3.41）中的 INTx 位由硬件置 1，表明正在等待 CPU 响应。读中断标志寄存器 IFR 中的 INTx 位可以识别该中断是否产生；向中断标志寄存器 IFR 中的 INTx 位写 1 将清除这个中断请求；将中断标志寄存器 IFR 的内容写回 IFR 可清除所有产生的中断请求。

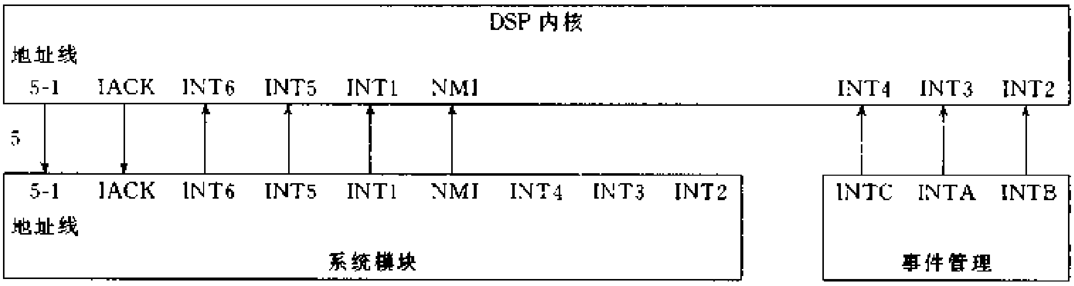
当中断请求信号产生后，最重要的是 CPU 怎样去识别？CPU 怎样去响应？为此，DSP 控制器给每个中断分配了一个特定的入口地址，称为中断向量，见表 3.41。当某个中断发出请求，而且允许它中断，则 CPU 先将当前的 PC 加 1 压入堆栈，即保护返回（断点）地址；然后、CPU 自动地将该请求中断的向量地址送入 PC，CPU 便转入该请求中断的服务子程序运行；当碰到服务子程序的返回指令 RET，CPU 自动将堆栈中的返回地址弹出到 PC 中，恢复中断前的程序继续运行。DSP 中断结构图如图 3.35 所示。

表 3.41 中断屏蔽寄存器 IMR 和中断标志寄存器 IFR 的组成

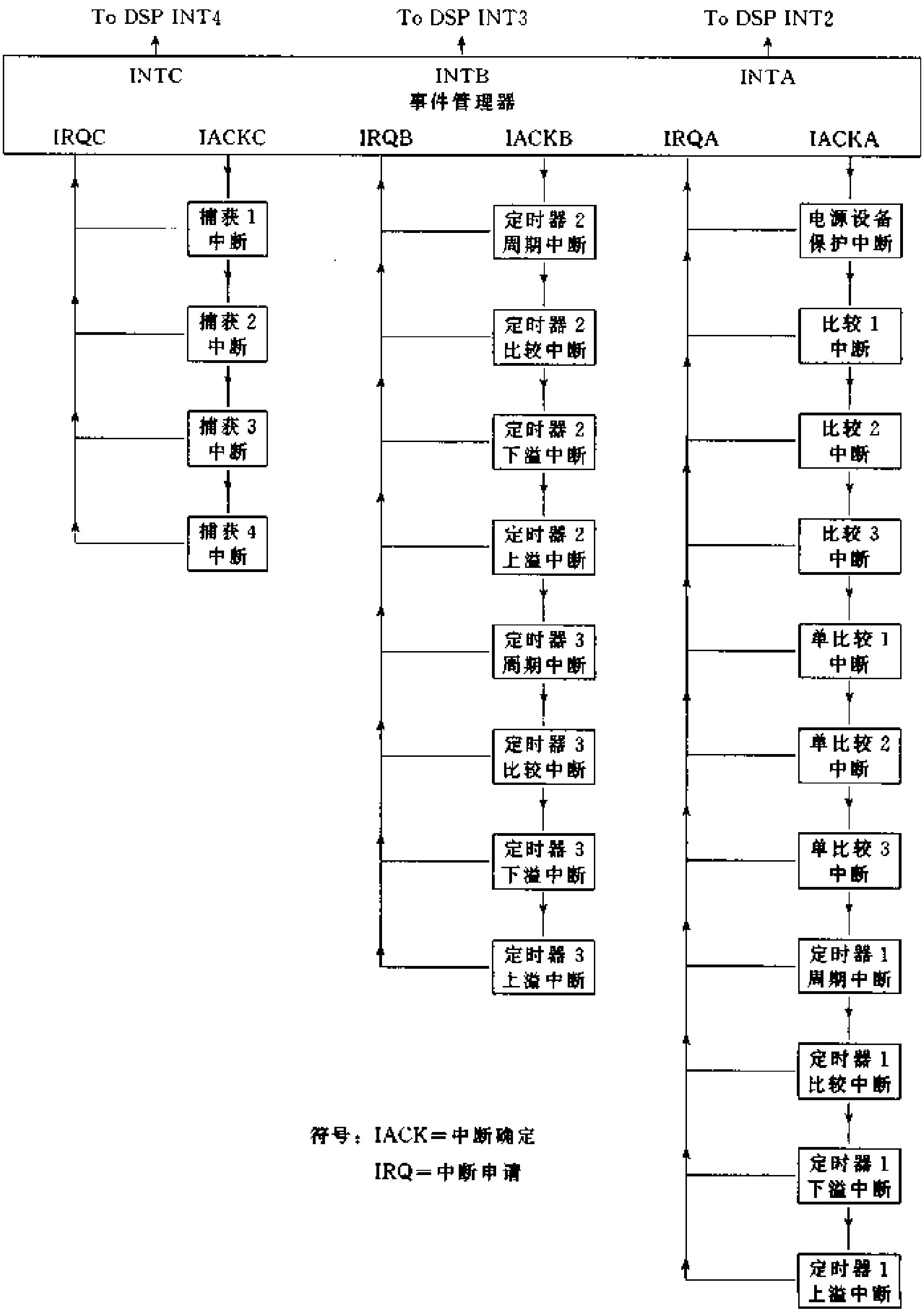
地 址	寄存器	位 数						
0004h	IMR	15～6	5	4	3	2	1	0
		Reserved	INT6	INT5	INT4	INT3	INT2	INT1
			R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
0006h	IFR	15～6	5	4	3	2	1	0
		Reserved	INT6	INT5	INT4	INT3	INT2	INT1
			R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

这里要说明的是，从表 3.42 知两个相邻的中断向量地址相差为 2，要在这个空间中放入中断服务子程序是不够的。因此，在这个空间中一般是放入一条分支指令（例如 B、BACC），这样就可以在较大的程序存储器空间中开辟出一块存放中断服务子程序的空间，通过中断向量处的分支指令转入到真正的中断服务子程序空间的入口上。

每个中断都是独立工作的，它们有可能同时发出中断请求。为了处理这种并发竞争，对每个中断赋有一个优先级，优先级高的先响应。

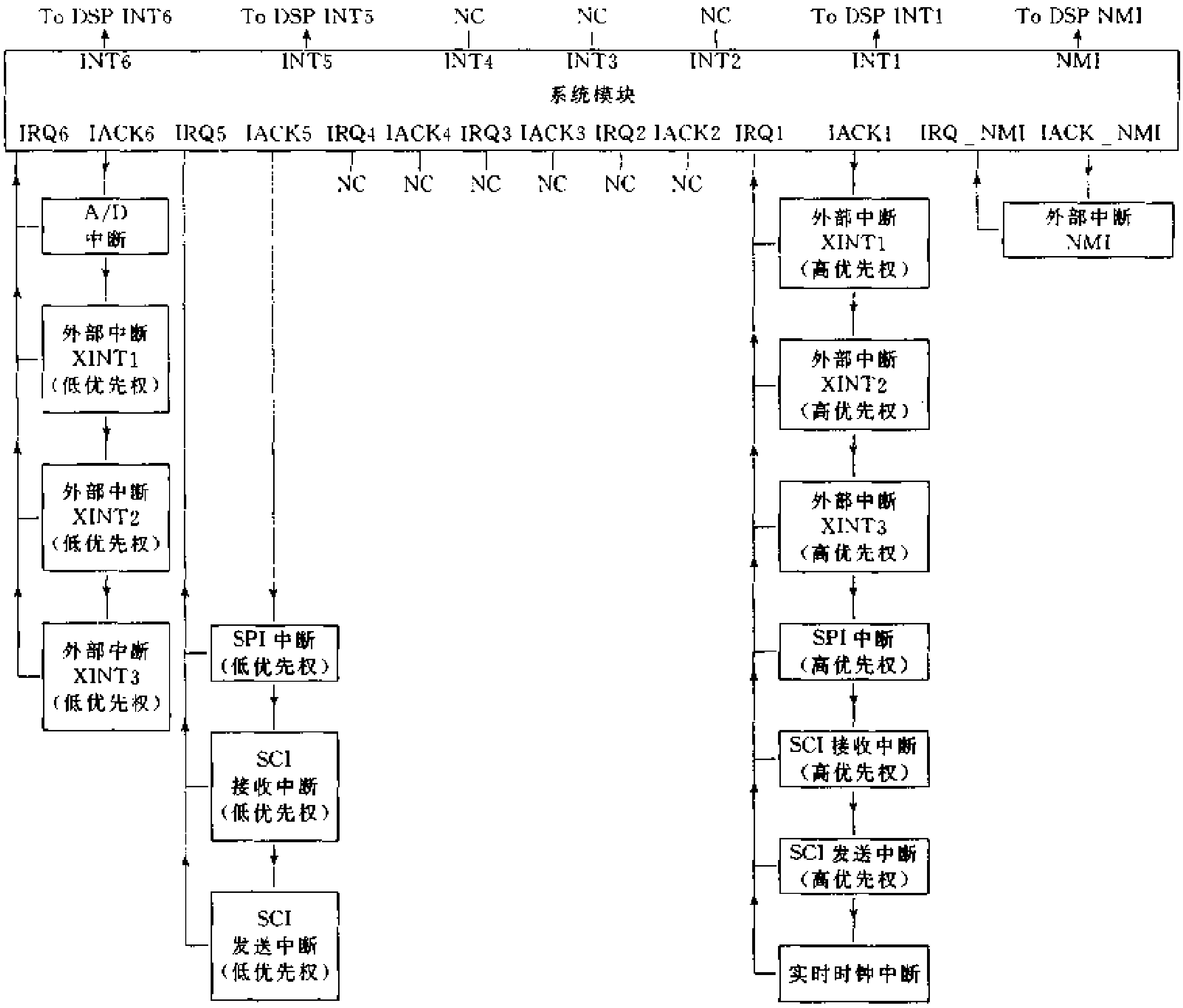


a）内核中断



b）事件管理模块中断

图 3.35　DSP 中断结构图



c）系统模块中断

图 3.35 DSP 中断结构图（续）

DSP 控制器内核的可屏蔽中断 INTx（x=1，2，3，4，5，6）与片内外设和外部可屏蔽中断引脚有着密切关系，下面会详细介绍。对于其它中断，在应用时要注意：

●外部引脚$\overline{RS}$：复位信号实质上是一个中断请求信号，而且是不可屏蔽的。当这个信号有效后，CPU 将终止一切工作，把系统复位到预定的缺省状态上；同时，自动转到复位中断向量 0000H 开始程序的运行，因此在程序存储器的首地址 0000H 必须安排一个分支到主程序入口的指令。

●外部引脚 NMI：引脚 NMI 产生的中断是不可屏蔽的，主要用于处理外部最紧要的事情。当发生引脚 NMI 中断时，状态寄存器 ST0 的中断模式位 INTM 被自动置为 1，所有可屏蔽中断不再响应。另外，引脚 NMI 可以低电平产生中断，也可以高电平产生中断，由外部中断控制寄存器 XINTA-NMICR 来设置。

●指令 INTR k：软件指令引起的中断响应过程与硬件中断一样。当执行到指令 INTR k 时，CPU 将自动转入到 INTR k 对应的中断向量地址处。要注意，指令 INTR k 引起中断时，状态寄存器 ST0 的中断模式位 INTM 被自动置为 1，所有可屏蔽中断不再响应，并且不影响中断标志寄存器 IFR 中的标志。

●指令 NMI：NMI 指令的中断向量与外部引脚 NMI 的中断向量是同一个地址（24H），因此执行 NMI 指令和驱动外部引脚 NMI 为低电平，将执行同一个中断服务子程序。指令 NMI 引起中断时，状态寄存器 ST0 的中断模式位 INTM 被自动置为 1，所有可屏蔽中断不再响应。

●指令 TRAP：该指令引起的中断响应过程与其它中断类似，但不影响可屏蔽中断的响应。

DSP 控制器的硬件堆栈只有 8 级。每进行一次中断或调用子程序，CPU 都自动地将返回地址压入堆栈，所以最多允许中断或子程序嵌套 8 级。如果在中断服务子程序或调用子程序中使用了主程序用到的累加器、乘法器以及内存单元，而且这些数据不允许子程序对其进行修改，那么在子程序的最前面要将这些数据保存，这也是现场保护的一部分，即返回地址由 CPU 自动保护，数据保护由用户编程完成。保存的方法有两种：通过传送指令将要保护的数据放到一个保护区中；或者通过 PUSH、PSHD 指令将要保护的数据压入到硬件堆栈（此时，中断或子程序嵌套将少于 8 级）。当子程序退出前（RET 指令之前），需要将保护的数据按序恢复，以保证主程序的正常运行。为了采用堆栈的形式保护较多的数据，可以通过 PSHD、POPD 指令按一定的逻辑关系将硬件堆栈延伸到数据存储器中。

中断嵌套要特别小心，万一硬件堆栈溢出，将不能正确地返回，从而使程序“跑飞”导致死机。如果不希望可屏蔽中断嵌套，可在中断服务子程序开始，将状态寄存器 ST0 的中断模式位 INTM 置 1；在退出中断服务子程序之前，将状态寄存器 ST0 的中断模式位 INTM 清 0。

发出中断请求到得到服务之间的延迟时间，与很多因素有关。

●软件中断最少要延迟 4 个 CPU 时钟周期。

●外部可屏蔽中断最少要延迟 8 个 CPU 时钟周期。

●若在使用 RPT 重复时发生中断，为了保证指令流水线的完整，需等到该重复循环完才响应中断。但复位中断例外。

●为使 CPU 能完成返回，在 RET 指令后中断被禁止，直至在返回地址上至少执行一条指令。

●读写速度慢的外部存储器需要等待延时，如果中断向量存放在外部存储器，等待状态会影响中断的响应时间。

●如果在 HOLD 操作时发生中断，并且需要从外部存储器取中断向量，那么在 $\overline{\text{HOLDA}}$ 无效前不能取中断向量。

3.7.2 事件管理模块的中断

事件管理模块的中断与 DSP 内核中断的关系如图 3.35b 所示。事件管理模块的中断分为三组，分别对应 DSP 内核中断中的 INT2、INT3、INT4：

● INTA：电源保护中断 PDPINT，全比较中断 COMP1INT、COMP2INT、COMP3INT，单比较中断 SCMP1INT、SCMP2INT、SCMP3INT，通用定时器 1 的周期匹配中断 T1PINT、比较匹配中断 T1CINT、下溢中断 T1UFINT、上溢中断 T1OFINT。

● INTB：通用定时器 2 的周期匹配中断 T2PINT、比较匹配中断 T2CINT、下溢中断 T2UFINT、上溢中断 T2OFINT，通用定时器 3 的周期匹配中断 T3PINT、比较匹配中断

T3CINT、下溢中断 T3UFINT、上溢中断 T3OFINT。

●INTC：捕获中断 CMP1INT、CMP2INT、CMP3INT、CMP4INT。

这三组中断都是可屏蔽的，分别由事件屏蔽寄存器 EVIMRA、EVIMRB、EVIMRC 进行设置（见表 3.43）。事件屏蔽寄存器 EVIMRA、EVIMRB、EVIMRC 中某一位为 0，对应的中断被屏蔽，否则将开启这个中断。当这三组中的任一个中断发出请求信号，都会在相应的事件中断标志寄存器 EVIFRA、EVIFRB、EVIFRC 中的对应的位置上置 1（事件中断的内容可参见 3.1.1）；如果这些中断未屏蔽，将同时触发相应的 DSP 内核中断；如果该 DSP 内核中断未屏蔽，CPU 将响应这个中断。

这里有一个问题，例如由 INTA 组中的某一个中断发出请求信号，如果所有有关的中断都未屏蔽，那么将触发 DSP 内核中断 INT2；按照 DSP 内核中断的响应过程，CPU 先保护返回地址，接着进入到 INT2 的中断向量地址 0004H 处；问题是 CPU 怎样得知是 INTA 组中的哪一个中断发出的请求信号？由于多个中断（INTA）复用同一个中断（INT2）与 CPU 打交道，必然要求有一种区别的方法，才不会混乱。实际上有两种方案可选：

●当进入到 DSP 内核中断（INT2、INT3、INT4）的服务子程序后，通过读事件中断标志寄存器（EVIFRA、EVIFRB、EVIFRC）的标志，以分辨是哪个中断发出了请求信号。这种方案需要较多的 CPU 时钟开销。

●为了更好地处理中断复用情况，DSP 控制器为事件管理模块的每一个中断分配了一个偏移向量地址（见表 3.42），并且当某个事件管理模块中断发出了请求信号，会自动地将该中断的偏移向量地址写入到对应的事件中断向量寄存器（EVIVRA、EVIVRB、EVIVRC 中（见表 3.43）。这样一来，当进入到 DSP 内核中断（INT2、INT3、INT4）的服务子程序后，将事件中断向量寄存器（EVIVRA、EVIVRB、EVIVRC）的内容送到累加器，然后经分支指令便可转入到专为某个事件管理模块中断所写的中断服务子程序的入口上。例如，

```
0004h        B   GISR2                ；INT2 的入口，通过分支指令 B 转入到
                                      ；真正的中断服务子程序入口 GISR2。
--------------------------------------------------------------------------
1000h  GISR2     ……                   ；这里要安排一些现场数据保护的指令。
                 LACC   EVIVRA，8     ；假定 T1PINT 发出请求，则 EVIVRA＝0027H
                 BACC                 ；经过分支指令 BACC 便可转入到真正的
                                      ；T1PINT 的中断服务子程序的入口。
--------------------------------------------------------------------------
2700H                                 ；T1PINT 中断服务子程序的入口。
```

对于中断复用情况采用偏移向量地址是一个很好的办法，可节约 CPU 时钟开销，也可以减少程序存储器的存储开销，因为采用中断标志分辨的方法需要进行许多次条件判断才能把涉及到的中断分辨出来，而采用偏移向量地址的方法，只需二条指令即可完成分辨任务。

表 3.42 DSP 控制器中断向量

优先级	中断名称	DSP 内核中断向量地址	外设向量寄存器地址	外设偏移向量地址	是否可屏蔽	控制器模块	功能
1(最高)	RS	$\overline{RS}$(0000h)	N/A	N/A	不可	内核,SD	外部或系统复位
2	保留	INT7(0026h)	N/A	N/A	不可	内核	仿真中断
3	NMI	NMI(0024h)	N/A	0002h	不可	内核,SD	外部用户中断

（续）

优先级	中断名称	DSP 内核中断向量地址	外设向量寄存器地址	外设偏移向量地址	是否可屏蔽	控制器模块	功能
4	XINT1	INT1(0002h)（系统模块）	SYSIVR 701Eh	0001h	可	SD	高优先级外部用户中断
5	XINT2			0011h	可		
6	XINT3			001Fh	可		
7	SPIINT			0005h	可	SPI	高优先级 SPI 中断
8	RXINT			0006h	可	SCI	SCI 接收中断
9	TXINT			0007h	可	SCI	SCI 发送中断
10	RTINT			0010h	可	WDT	实时中断
11	PDPINT	INT2(0004h)（事件管理器中断组 A）	7432h	0020h	可	外部	功率驱动保护
12	CMP1INT			0021h	可	EV.CMP1	全比较 1 中断
13	CMP2INT			0022h	可	EV.CMP2	全比较 2 中断
14	CMP3INT			0023h	可	EV.CMP3	全比较 3 中断
15	SCMP1INT			0024h	可	EV.CMP4	单比较 1 中断
16	SCMP2INT			0025h	可	EV.CMP5	单比较 2 中断
17	SCMP3INT			0026h	可	EV.CMP6	单比较 3 中断
18	TPINT1			0027h	可	EV.GPT1	定时器 1 周期中断
19	TCINT1			0028h	可	EV.GPT1	定时器 1 较中断
20	TUFINT1			0029h	可	EV.GPT1	定时器 1 下溢中断
21	TOFINT1			002Ah	可	EV.GPT1	定时器 1 上溢中断
22	TPINT2	INT3(0006h)（事件管理器中断组 B）	7433h	002Bh	可	EV.GPT2	定时器 2 周期中断
23	TCINT2			002Ch	可	EV.GPT2	定时器 2 较中断
24	TUFINT2			002Dh	可	EV.GPT2	定时器 2 下溢中断
25	TOFINT2			002Eh	可	EV.GPT2	定时器 2 上溢中断
26	TPINT3			002Fh	可	EV.GPT3	定时器 3 周期中断
27	TCINT3			0030h	可	EV.GPT3	定时器 3 较中断
28	TUFINT3			0031h	可	EV.GPT3	定时器 3 下溢中断
29	TOFINT3			0032h	可	EV.GPT3	定时器 3 上溢中断
30	CAPINT1	INT4(0008h)（事件管理器中断组 C）	7434h	0033h	可	EV.CAP1	捕获 1 中断
31	CAPINT2			0034h	可	EV.CAP2	捕获 2 中断
32	CAPINT3			0035h	可	EV.CAP3	捕获 3 中断
33	CAPINT4			0036h	可	EV.CAP4	捕获 4 中断
34	SPIINT	INT5(000Ah)（系统）	SYSIVR (701Eh)	0005h	可	SPI	低优先级 SPI 中断
35	RXINT			0006h	可	SCI	SCI 接受中断
36	TXINT			0007h	可	SCI	SCI 接收中断

（续）

优先级	中断名称	DSP 内核中断向量地址	外设向量寄存器地址	外设偏移向量地址	是否可屏蔽	控制器模块	功能
37	ADCINT	INT6(00Ch)（系统）	SYSIVR (701Eh)	0004h	可	ADC	模数转换中断
38	XINT1			0001h	可	外部引脚	低优先级外部用户中断
39	XINT2			0011h	可		
40	XINT3			001Fh	可		
41	保留	000Eh	N/A		可	内核	用于分析
N/A	TRAP	0022h	N/A		N/A		TRAP 指令向量

表 3.43 事件管理模块的中断寄存器的组成

地址	寄存器	位数							
742Ch 742Fh	EVIMRA EVIFRA	15	14	13	12	11	10	9	8
		Reserved	Reserved	Reserved	Reserved	Reserved	T1OFINT	T1UFINT	T1CINT
							R/W-0	R/W-0	R/W-0
		7	6	5	4	3	2	1	0
		T1PINT	SCMP3INT	SCMP2INT	SCMP1INT	CMP3INT	CMP2INT	CMP1INT	PDPINT
		R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
742Dh 7430h	EVIMRB EVIFRB	15	14	13	12	11	10	9	8
		Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
		7	6	5	4	3	2	1	0
		T3OFINT	T3UFINT	T3CINT	T3PINT	T2OFINT	T2UFINT	T2CINT	T2PINT
		R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
742Eh 7431h	EVIMRC EVIFRC	15	14	13	12	11	10	9	8
		Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
		7	6	5	4	3	2	1	0
		Reserved	Reserved	Reserved	Reserved	CAP4INT	CAP3INT	CAP2INT	CAP1INT
						R/W-0	R/W-0	R/W-0	R/W-0
7432h 7433h 7433h	EVIVRA EVIVRB EVIVRC	15	14	13	12	11	10	9	8
		0	0	0	0	0	0	0	0
		R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
		7	6	5	4	3	2	1	0
		0	0	D5	D4	D3	D2	D1	D0
		R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

3.7.3 系统模块中断

系统模块的中断与 DSP 内核中断的关系如图 3.35c 所示。系统模块的中断分为四组，分别对应 DSP 内核中断中的 INT1、INT5、INT6 和 NMI：

● INT1：高优先级的外部引脚中断 XINT1、XINT2、XINT3，高优先级的串行外设接口 SPI 中断 SPIINT，高优先级的串行通信 SCI 发送中断 TXINT，高优先级的串行通信 SCI 接收中断 RXINT，实时时钟中断 RTINT。

● INT5：低优先级的串行外设接口 SPI 中断 SPIINT，低优先级的串行通信 SCI 发送中断 TXINT，低优先级的串行通信 SCI 接收中断 RXINT。

● INT6：A/D 转换中断 ADCINT，低优先级的外部引脚中断 XINT1、XINT2、XINT3。

● NMI：外部引脚 NMI 中断。

上述中断除了 NMI 中断外，都是可以屏蔽的。外部引脚中断 XINTx（x=1，2，3）的屏蔽由外部引脚中断控制寄存器 XINTxCR 的使能位 XINTxEN 设置（见表 3.44）；串行外设接口 SPI 中断 SPIINT 的屏蔽由 SPI 工作控制寄存器 SPICTL 的使能位 SPINTEN 设置（见表 3.26）；串行通信 SCI 发送和接收中断 TXINT、RXINT 由 SCI 控制寄存器 SCICLT2 的使能位 TX/BKEN、RX/BKEN 设置（见表 3.17）；A/D 转换中断 ADCINT 的屏蔽由 ADC 控制寄存器 ADCTRL1 的 ADCINTEN 设置（见表 3.14）；实时时钟中断 RTINT 由实时中断控制寄存器 RTICR 的使能位 RTIEN 设置（见表 3.38）。

表 3.44 外部中断控制寄存器的组成

地址	寄存器	位数							
7070h	XINT1CR	15	14	13	12	11	10	9	8
		XINT1 Flag	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
		R/C-0							
		7	6	5	4	3	2	1	0
		Reserved	XINT1 Pin data	XINT1 NWI Enable	Reserved	Reserved	XINT1 Polarity	XINT1 Priority	XINT1 Enable
							R/W-0	R/W-0	R/W-0
7078h 707Ah	XINT2CR XINT3CR (x=1,2,3)	15	14	13	12	11	10	9	8
		XINTx Flag	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
		R/C-0							
		7	6	5	4	3	2	1	0
		Reserved	XINTx Pin data	Reserved	XINTx Data dir	XINTx Data dir	XINTx Polarity	XINTx Priority	XINTx Enable
			R-p				R/W-0	R/W-0	R/W-0
7072h	NMICR	15	14	13	12	11	10	9	8
		NMI Flag	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
		R/C-0							
		7	6	5	4	3	2	1	0
		Reserved	NMI Pin data	NMI Enable	Reserved	Reserved	NMI Polarity	Reserved	Reserved
			R-p	R/W-0			R/W-0		
701Eh	SYSIVR	15	14	13	12	11	10	9	8
		D15	D14	D13	D12	D11	D10	D9	D8
		R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
		7	6	5	4	3	2	1	0
		D7	D6	D5	D4	D3	D2	D1	D0
		R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

外部引脚 NMI 中断、外部引脚中断 XINTx（x=1，2，3）、串行外设接口 SPI 中断、串行通信 SCI 发送中断、串行通信 SCI 接收中断、A/D 转换中断以及实时时钟中断产生后，都会分别在 NMI 控制寄存器 NMICR 的 NMIF 位、XINTx 控制寄存器 XINTxCR 的 XINTxF 位、SPI 状态寄存器 SPISTS 的 SPIINTF 位、SCI 控制寄存器 SCICTL2 的 TXRDY 位、SCI 接收状态寄存器 SCIRXST 的 RXRDY 或 BRKDT 位、ADC 控制寄存器 ADCTRL1 的 ADCINTFLAG 位、实时中断控制寄存器 RTICR 的 RTIF 位上置 1，以表示该中断的发生。

与其它中断不同，外部引脚中断 XINTx、串行外设接口 SPI 中断、串行通信 SCI 发送中断、串行通信 SCI 接收中断的优先级是可编程的。分别由 XINTx 控制寄存器 XINTxCR 的 XINTx Priority 位、SPI 优先级寄存器 SPISTS 的 SPI Priority 位、SCI 优先级控制寄存器 SCIPRI 的 SCITX Priority 位、SCIRX Priority 位进行设置，写 1 为高优先级，写 0 为低优先级。这里的优先级高低是对 DSP 内核中断 INT1、INT5、INT6 而言。

外部引脚中断 XINTx 的触发极性是可编程的，由 XINTx 控制寄存器 XINTxCR 的 XINTx Polarty 位设置。若该位为 0，则以下降沿触发；若该位为 1，则以上升沿触发。

由于系统模块中断与 DSP 内核中断 INT1、INT5、INT6 复用，因此与事件管理模块中断一样，当 CPU 响应 DSP 内核中断 INT1、INT5、INT6 时，存在一个怎样分辨是哪一个系统模块中断引发的问题。为此也有类似的两种方案：

●当进入到 DSP 内核中断（INT1、INT5、INT6）的服务子程序后，通过判断 XINTx 控制寄存器 XINTxCR 的 XINTxF 位、SPI 状态寄存器 SPISTS 的 SPIINTF 位、SCI 控制寄存器 SCICTL2 的 TXRDY 位、SCI 接收状态寄存器 SCIRXST 的 RXRDY 或 BRKDT 位、实时中断控制寄存器 RTICR 的 RTIF 位上是否为 1 来分辨是哪个中断发出了请求信号。这种方案需要较多的 CPU 时钟开销。

●与事件管理模块类似，DSP 控制器也给系统模块的每一个中断（除 NMI 外）分配了一个偏移向量地址（见表 3.41），并且当某个系统模块中断发出了请求信号，会自动地将该中断的偏移向量地址写入到系统模块中断向量寄存器 SYSIVR 中。这样一来，当进入到 DSP 内核中断（INT1、INT5、INT6）的服务子程序后，将系统模块中断向量寄存器 SYSIVR 的内容送到累加器，然后经分支指令便可转入到专为某个系统模块中断所写的中断服务子程序的入口上。

由于 DSP 内核中断 INTk（k=1，2，3，4，5，6），也可以由软件指令产生。因此，当执行软件指令 INTR k（k=1，2，3，4，5，6）或事件管理模块中断、系统模块中断线上出现虚假信号，此时事件中断向量寄存器（EVIVRA、EVIVRB、EVIVRC）或系统模块中断向量寄存器 SYSIVR 的内容应该是不确定的，但是为了使采用偏移向量地址方法能完整地进行，DSP 控制器在这两种情况下，自动地将一个虚构的偏移向量地址（0000h）放入到向量寄存器中。因此，在中断服务子程序中要小心处理这种情况。

第4章 指令系统

前三章详细介绍了DSP控制器的硬件构成，可以看出DSP控制器的硬件有着独特的优点：采用多组总线结构实现并行机制，极大地加快了处理的速度；此外，独立的乘法器、输入与输出移位器、辅助寄存器算术单元以及微堆栈等硬件模块为实现丰富的寻址方式，构成强大的指令系统提供了物质基础。DSP控制器指令系统可以分为四大类：数据传送、算术运算、逻辑运算、分支转移，下面分别给予介绍。

4.1 寻址方式

在介绍具体指令之前，首先讨论DSP控制器的寻址方式。我们知道无论单片机还是DSP控制器，其基本任务都是：从某个地方取数，然后运算，再将结果放到某个地方。因此，在指令中如何表达数据的地址是指令系统的核心。DSP控制器的寻址方式分为三种：立即寻址、直接寻址和间接寻址。立即寻址，即需要寻找的数就在指令里，不需要到存储器中去找，也称为立即数；直接寻址，即指令给出的是需要寻找的数的地址，按此地址直接去访问便可；间接寻址，即指令给出的既不是立即数也不是直接地址，而是将此地址（或寄存器）的内容再作为地址。

4.1.1 立即寻址

在DSP控制器中立即寻址方式分为两种：短立即寻址和长立即寻址。短立即寻址的指令把一个8、9或13位的常数作为操作数。这种寻址方式的指令为单字指令，立即数包含在指令中，而且在立即数前加#以示与直接地址的区别。举例如下：

例4.1 采用短立即寻址的RPT指令，需要重复执行的次数直接跟在指令操作码后（见图4.1）。

RPT #99 ；将紧跟RPT后面的那条指令执行100次。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	0	1	1	0	1	1	0	0	0	1	1

采用立即寻址的RPT操作码　　　　8位常数=99

图4.1 例4.1指令寄存器的内容

长立即寻址的指令把一个16位的常数作为操作数，这时为双字指令，第二个字即为该操作常数。这个16位的数值可以是一个绝对值常数，也可以是一个二进制补码值。举例如下：

例4.2 采用长立即寻址的ADD指令（见图4.2）

ADD #16384,2 ；将数值16384左移2位后与累加器内容相加，结果在累加器中。

第一字

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	1	1	0	0	1	0	0	1	0

长立即寻址的 ADD 指令操作码　　　移位次数=2

第二字

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

16 位常数=16384=4000H

图 4.2　例 4.2 指令寄存器中的两个字

4.1.2　直接寻址

直接寻址是一种常用的寻址方式,可以访问 64K 字数据存储器。在 DSP 控制器中,数据存储器按页进行管理,整个 64K 字数据存储器分为 512 个数据页(详见 2.5.3),每个数据页含有 128 个字。在访问数据存储器时,首先确定当前数据页(通过 LDP 等指令),它由状态寄存器 ST0 中的九位数据页面指针(DP)值确定,例如 DP 值为 000000010b,即当前数据页为 2。确定当前数据页后,该数据页 128 个字中的哪一个字则由指令寄存器的低 7 位(LSB)偏移量指定。DSP 控制器的直接寻址指令中的直接地址就是该低 7 位(LSB)偏移量,如图 4.3 所示。中央处理单元将当前 DP 值与偏移量拼接,就变成 16 位的存储器地址。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
8MSB								0	7LSB						

图 4.3　直接寻址方式下指令代码的内容

8MSB　　位 15 至位 8,指出指令类型(例如 ADD),并包含该指令所访问的数据值的移位信息。

0　　直接/间接寻址指示符。图中位 7 为 0,则定义为直接寻址。

7LSB　　位 6 至位 0,给出该指令所访问的数据存储器地址的偏移量。

使用直接寻址方式时,要特别注意当前数据页的值。一方面,DP 在复位时并没有初始化,所以在程序之前应注意初始化 DP;二方面,若有一程序段中的所有指令都访问同一个数据页,只需在该程序段前装载 DP 一次;三方面,若在程序中需访问不同的数据页,则每当访问新的数据页前,都需修改 DP 值,以确保使用正确的数据页。

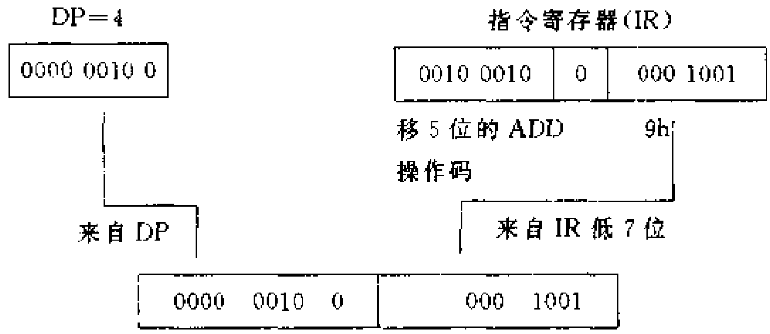
设置数据页可利用 LDP 指令或任何能将数值加载到状态寄存器 ST0 的指令。LDP 指令直接加载 DP,不影响 ST0 的其他位。

设置偏移量,只需在指令的操作数的位置上写上该 7 位偏移量即可(前面不能加#)。

例 4.3　采用直接寻址的 ADD 指令,将数据存储器地址 0209h 中的内容左移 5 位后与累加器的内容相加。数据存储器地址 0209h(000000100　0001001b)的页面地址是 4h,偏移量是 9h,因此采用如下指令可完成任务。

LDP　#4　;将数据页面设置为 4。

ADD　9h,5　;将数据存储器地址 0209H 中的内容左移 5 位后与累加器的内容相加。



16 位数据存储器地址 0209h

4.1.3　间接寻址

间接寻址的能力在很大的程度上反映了指令系统的灵活性和方便性。DSP 控制器内含 8 个辅助寄存器(AR0～AR7)和辅助寄存器算术单元(ARAU),专用于间接寻址的操作,不但提供了灵活而强大的间接寻址能力,而且使得间接寻址的速度非常快。

DSP 控制器用 16 位辅助寄存器的内容作为间接的地址,因此 DSP 控制器的间接寻址可以访问 64K 字数据存储器空间的任一单元,不受当前数据页的限制。

8 个辅助寄存器(AR0～AR7)都可以参与间接寻址,但是每次寻址只能使用其中的一个,它由状态寄存器 ST0 中的 3 位辅助寄存器指针(ARP)来指定。ARP 指定的辅助寄存器称为当前辅助寄存器或当前 AR。

由于在 DSP 控制器中设置了辅助寄存器算术单元(ARAU),因此在进行间接寻址操作的同时可以对辅助寄存器的内容进行运算,甚至修改 ARP 的值,为下次的间接寻址作准备,从而极大地提高间接寻址的速度。

对辅助寄存器的内容进行运算有四种选择:

●不增不减。指令使用当前辅助寄存器的内容作为数据存储器地址,同时不增加也不减少当前辅助寄存器的内容。

●增 1 或减 1。指令使用当前辅助寄存器的内容作为数据存储器地址,指令执行后将当前辅助寄存器的内容增 1 或减 1。

●增或减一个变址量。将 AR0 中的值作为变址量。指令使用当前辅助寄存器的内容作为数据存储器地址,指令执行后将当前辅助寄存器的内容增加或减去这个变址量。

●按反向进位方式增或减一个变址量。将 AR0 中的值作为变址量。指令使用当前辅助寄存器的内容作为数据存储器地址,指令执行后将当前辅助寄存器的内容按反向进位方式增加或减去这个变址量。反向进位方式的加或减是从最高位开始运算,有进位(或借位)给低位。这适用于 FFT 算法。

上述四种运算均由辅助寄存器算术单元(ARAU)在流水线中指令译码的同一周期内完成。表 4.1 列出了七种间接寻址选项,还给出了每种间接寻址选项对应的指令操作数符号及使用每种选项的例子。

表 4.1 间接寻址选项及其操作数

选 项	操作数符号	例 子
不增不减	*	LT* 用当前 AR 所指的数据存储器地址内容装载暂时寄存器(TREG)。
增 1	$*_{+}$	LT $*_{+}$ 用当前 AR 指定的数据存储器地址内容装载暂时寄存器(TREG),然后将当前 AR 内容加 1
减 1	$*_{-}$	LT $*_{-}$ 用当前 AR 指定的数据存储器地址内容装载暂时寄存器(TREG),然后将当前 AR 内容减 1
加变址量	$*0_{+}$	LT $*0_{+}$ 用当前 AR 指定的数据存储器地址内容装载暂时寄存器(TREG),然后将当前 AR 内容加 AR0 的内容
减变址量	$*0_{-}$	LT $*0_{-}$ 用当前 AR 指定的数据存储器地址内容装载暂时寄存器(TREG),然后将当前 AR 内容减 AR0 的内容
反向进位加变址量	$*BR0_{+}$	LT $*BR0_{+}$ 用当前 AR 指定的数据存储器地址内容装载暂时寄存器(TREG),然后按反向进位方式将当前 AR 内容加 AR0 的内容
反向进位减变址量	$*BR0_{-}$	LT $*BR0_{-}$ 用当前 AR 指定的数据存储器地址内容装载暂时寄存器(TREG),然后按反向进位方式将当前 AR 内容减 AR0 的内容

采用间接寻址方式时,加载到指令寄存器指令字格式如图 4.4 所示。

15 14 13 12 11 10 9 8	7	6 5 4	3	2 1 0
8MSB	1	ARU	N	NAR

图 4.4 间接寻址指令寄存器的内容

8MSB　　位 15 至位 8 指示指令类型(如 LT)及与数据移位有关的信息。

1　　直接/间接指示。图中位 7 为 1,定义了间接寻址方式。

ARU　　辅助寄存器更新代码。位 6~4 确定是否对辅助寄存器进行更新以及将其增加还是减少。见表 4.2。

N　　下一辅助寄存器指示。位 3 指明该指令是否改变 ARP 的值。

若 N=0,ARP 的内容保持不变。

若 N=1,NAR 的内容加载到 ARP,原先的 ARP 值加载到状态寄存器 ST1 中的辅助寄存器缓冲器(ARB)。

NAR　　下一辅助寄存器。位 2~0 的值指出下一辅助寄存器。若 N=1,NAR 被加载到 ARP。

表 4.2 给出了各种间接寻址的操作码字段,还说明了对当前辅助寄存器和 ARP 所执行的操作。进行间接寻址时,首先要确定当前辅助寄存器。当前辅助寄存器可由 MAR 指令或任何能将数值加载到状态寄存器 ST0 的指令来指定,也可以在间接寻址指令中指定下一次的当前辅助寄存器。当前辅助寄存器的内容,也就是间接寻址的地址,除了表 4.1 所列几种方式进行修改以外,也可以由 LAR 等指令来修改。下面举几个例说明间接寻址的使用情况。

表 4.2 间接寻址字段及说明

指令操作码的各位					操作数	操 作
15~8	7	654	3	210		
8MSB	1	000	0	NAR	*	对当前 AR 无操作
8MSB	1	000	1	NAR	*,ARn	NAR→ARP
8MSB	1	001	0	NAR	$*_{-}$	当前 AR−1→当前 AR

（续）

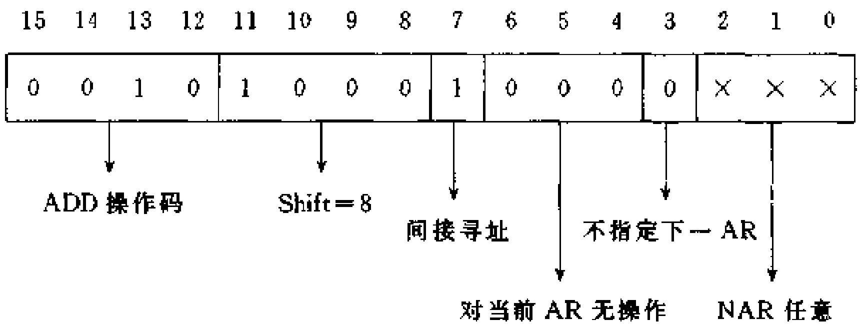
指令操作码的各位					操作数	操　作
15～8	7	654	3	210		
8MSB	1	001	1	NAR	* −,ARn	当前 AR−1→当前 AR,NAR→ARP
8MSB	1	010	0	NAR	* +	当前 AR+1→当前 AR
8MSB	1	010	1	NAR	* +,ARn	当前 AR+1→当前 AR,NAR→ARP
8MSB	1	100	0	NAR	* BR0−	当前 AR−rcAR0→当前 AR+
8MSB	1	100	1	NAR	* BR0−,ARn	当前 AR−rcAR0→当前 AR,NAR→ARP+
8MSB	1	101	0	NAR	* 0−	当前 AR−AR0→当前 AR
8MSB	1	101	1	NAR	* 0−,ARn	当前 AR−AR0→当前 AR,NAR→ARP
8MSB	1	110	0	NAR	* 0+	当前 AR+AR0→当前 AR
8MSB	1	110	1	NAR	* 0+,ARn	当前 AR+AR0→当前 AR,NAR→ARP
8MSB	1	111	0	NAR	* BR0+	当前 AR+rcAR0→当前 AR+
8MSB	1	111	1	NAR	* BR0+,ARn	当前 AR+rcAR0→当前 AR,NAR→ARP+

注：+—倒位序寻址方式；rc—反向进位；NAR—下一 AR；n—0,1,2…7；→加载到。

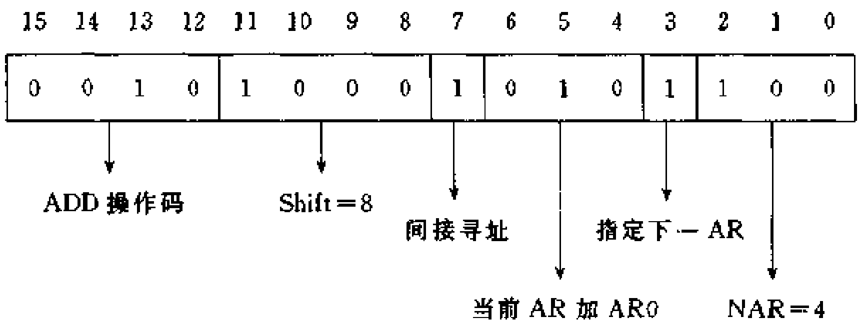
例 4.4　MAR *,AR1　　；选择 AR1 为当前辅助寄存器，即将 1 加载到 ARP 中。

例 4.5　LAR　AR1,#2080h　　；将 2080h 加载到 AR1 中。

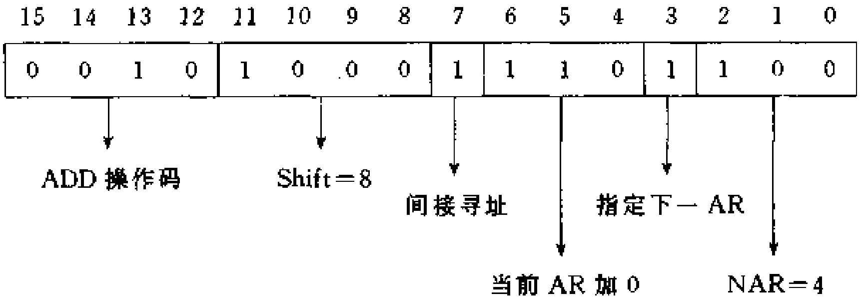
例 4.6　ADD　*,8　　；把当前辅助寄存器指定的数据存储器单元的内容左移 8 次后加至累加器。当前辅助寄存器的内容不增不减。



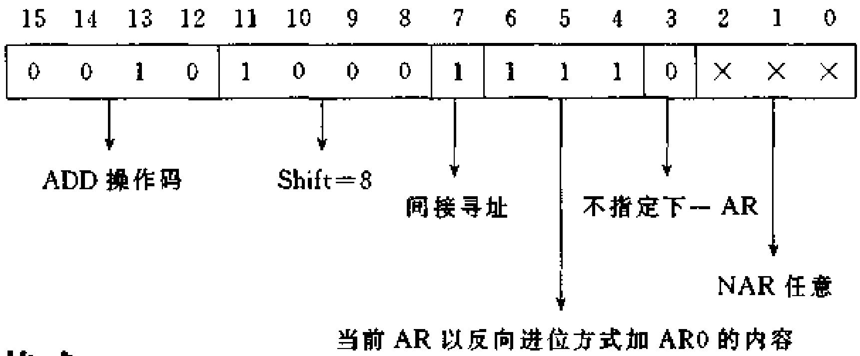
例 4.7　ADD　*+,8,AR4　　；把当前辅助寄存器指定的数据存储器单元的内容左移 8 次后加至累加器。当前辅助寄存器的内容增 1，且指定 AR4 为下一辅助寄存器。



例 4.8 ADD *0+,8,AR4 ; 把当前辅助寄存器指定的数据存储器单元的内容左移 8 次后加至累加器。当前辅助寄存器的内容与 AR0 的内容相加，且指定 AR4 为下一辅助寄存器。



例 4.9 ADD *BR0+,8 ; 把当前辅助寄存器指定的数据存储器单元的内容左移 8 次后加至累加器。当前辅助寄存器的内容与 AR0 的内容按反向进位方式相加。



4.2 句法格式

本节介绍 DSP 汇编语言格式和对指令系统进行较详细的描述。

4.2.1 汇编句法格式

DSP 指令的汇编句法格式为：

操作码 [操作数] ;[注释]

操作码 由可描述指令特征的助记符表示，它规定了指令的操作功能。

操作数 在操作数中定义了该句法表达式中所用的变量，以及寻址方式。

注释 为便于阅读，对指令作的说明。

例如，加法指令的句法有：

ADD dma [, shift]	直接寻址
ADD dma,16	左移 16 位直接寻址
ADD ind [, shift[, ARn]]	间接寻址
ADD ind,16 [, ARn]	左移 16 位间接寻址
ADD #k	短立即寻址
ADD #lk[, shift]	长立即寻址

以上指令中，ADD 表示作加法操作。后面的操作数可以是常数，也可以是引用存储器、

I/O 端口、寄存器地址、指针、移位次数和其它各种常数的汇编表达式。为了后面指令功能叙述方便，现将指令中常用到的有关符号列于表 4.3 中。

表 4.3　指令中常用到的有关符号

符　号	说　　明
dma	数据存储器地址的低 7 位(7LSB)
shift	左移位数 0～15(缺省为 0)
shift2	左移位数 0～7(缺省为 0)
ind	选择以下 7 种间接寻址方式之一： *、*+、*-、*0-、*0-、*BR0+、*BR0-。
[, x]	带方扩号表示 x 为可选项。但在含有可选项的句法中，可选项前的变量必须提供，例如：ADD dval[, mov]，必须提供 dval
[, x1[, x2]]	表示 x1 和 x2 都是可选项。可选项前的变量必须提供，若没有 x1，就不能有 x2。例如指令：ADD　*+,0,AR2，*+必须提供，先有选项 0，才能有选项 AR2
#	立即寻址方式中表示后面跟的是立即数，以避免与直接寻址方式混淆
n	数值 0～7，指定下次的辅助寄存器
k	8 位短立即数值
lk	16 位长立即数值
pma	16 位程序存储器地址
x	用以指示将被装载的辅助寄存器的值：0 至 7
bit code	用以指示被测试位置的值：0 至 15
PA	16 位 I/O 端口或 I/O 映射的寄存器地址
Control bit	选择以下控制位之一：C　CNF　INTM　OVM　SXM　TC　XF
ACC	累加器
AR	辅助寄存器
ARX	用于 LAR 和 SAR 指令的 3 位数字，用以指定哪一个寄存器将被装载(LAR)，或被储存(SAR)
BITX	4 位数字，用以决定指定数据存储器数值中的哪一位将被 BIT 指令所测试
CM	2 位数字。CMPR 指令执行 CM 值所定义的比较： 若 CM=00，测试当前 AR=AR0 否； 若 CM=01，测试当前 AR<AR0 否； 若 CM=10，测试当前 AR>AR0 否； 若 CM=11，测试当前 AR=AR0 否；
IAAA AAAA	(一个 I 后接 7 个 A)左边的 I 代表直接寻址(I=0)还是间接寻址(I=1)。当使用直接寻址时，7 个 A 就是数据存储器地址的 7 位最低有效位(LSB)；对于间接寻址，7 个 A 则是用于控制辅助寄存器使用的位(见间接寻址方式)
IIII　IIII	(8 个 I)用于短立即寻址的 8 位常数
I　IIII IIII	(9 个 I)用于短立即寻址方式下 LDP 指令的 9 位常数
I IIII IIII IIII	(13 个 I)用于短立即寻址方式下 MPY 指令的 13 位常数
NTR#	5 位数值，代表数字 0 至 31。INTR 指令使用该数值将程序控制转移至 32 个中断向量地址中的一个
PM	2 位数值，被 SPM 指令复制到状态寄存器 ST1 中的 PM 位

（续）

符　号	说　　明
SHF	3 位左移量
SHFT	4 位左移量
TP	用于条件执行指令的 2 位数值，代表如下四种条件： BIO 引脚为低　　TP=00 TC 位=1　　TP=01 TC 位=0　　TP=10 无条件　　TP=11
ZLVC ZLVC	两个 4 位字段——每一位代表如下条件： ACC=0　Z ACC<0　L 溢出　V 进位　C 一条条件指令包含两个这种 4 位字段。指令的低 4 位字段是屏蔽字段，相应屏蔽位处的 1 代表该条件要被测试。例如，欲测试 ACC≥0，否则 Z 和 L 置 1，而 V 和 C 不置 1。Z 置 1 用以测试是否 ACC=0。L 置 1 则用以测试条件 ACC>0 否。第二个 4 位字段（位 4～7）指明了被测试条件的状态。这八位的可能条件组合见指令 BCND、CC 和 RETC 的说明
+1 word	双字操作码的第二个字。该字包含一个 16 位常数。根据指令的不同，该常数可以是长立即数值、程序存储器地址、I/O 口和 I/O 映射寄存器地址

4.2.2 指令分类表

本章按功能将 DSP 控制器的指令分为四类，即数据传送类、算术运算类、逻辑运算类和分支转移类，现分别汇总于表 4.4～表 4.7 中。

表 4.4　数据传送一览表

助记符	说　　明	字节	周期	操　作　码
BLDD	数据存储器至数据存储器间的块移动，源地址为长立即数，目的地址为（直接或间接）寻址	2	3	1010　1000　IAAA　AAAA +1 word
	数据存储器至数据存储器间的块移动，目的地址为长立即数，源地址为直接或间接寻址	2	3	1010　1001　IAAA　AAAA +1 word
BLPD	程序存储器至数据存储器间的块移动，源地址为长立即数，目的地址为直接或间接寻址	2	3	1010　0101　IAAA　AAAA +1 word
DMOV	将片内数据存储器内的数据复制到下一单元（直接或间接）	1	1	0111　0111　IAAA　AAAA
IN	从 I/O 单元输入数据（直接或间接）	2	2	1010　1111　IAAA　AAAA +1 word
LACC	左移 0 至 15 位后装入累加器（直接或间接）	1	1	0001 SHFT　IAAA　AAAA
	长立即数左移 0 至 15 位后装入累加器	2	2	1011　1111　1000　SHFT +1 word
	左移 16 位后装入累加器	1	1	0110　1010　IAAA　AAAA
LACL	装载累加器低位字（直接或间接）	1	1	0110　1001　IAAA　AAAA
	用短立即数装载累加器低位字	1	1	1011　1001　IIII　IIII
LACT	按 TREG 规定的左移（0 至 15）后装入累加器（直接或间接）	1	1	0110　1011　IAAA　AAAA
LAR	用指定的数据单元装载指定的 AR（直接或间接）	1	2	0000　0ARX IAAA　AAAA
	用短立即常数装载指定的 AR	1	2	1011　0ARX　IIII　IIII

（续）

助记符	说　　明	字节	周期	操　作　码
	用长立即常数装载指定的 AR	2	2	1011　1111　0000　1ARX +1 word
LDP	装载数据页指针（直接或间接）	1	2	0000　1101　IAAA　AAAA
	用短立即数装载数据页指针	1	2	1011　1101　IIII　IIII
LPH	装载 PREG 高位（直接或间接）	1	1	0111　0101　IAAA　AAAA
LST	装载状态寄存器 ST0	1	2	0000　1110　IAAA　AAAA
	装载状态寄存器 ST1	1	2	0000　1111　IAAA　AAAA
LT	装载 TREG（直接或间接）	1	1	0111　0011　IAAA　AAAA
LTA	装载 TREG 并累加前次乘积（直接或间接）	1	1	0111　0000　IAAA　AAAA
LTD	装载 TREG，累加前次乘积并将被寻址数据移至下一单元（直接或间接）	1	1	0111　0010　IAAA　AAAA
LTP	装载 TREG 并将 PREG 存至累加器（直接或间接）	1	1	0111　0001　IAAA　AAAA
LTS	装载 TREG 并减去前次乘积（直接或间接）	1	1	0111　0100　IAAA　AAAA
MAR	修改当前 AR 和/或 ARP，间接（在直接方式下不执行任何操作）	1	1	1000　1011　IAAA　AAAA
OUT	从 I/O 单元输出数据（直接或间接）	2	3	0000　1100　IAAA　AAAA +1 word
PAC	PREG 装入累加器	1	1	1011　1110　0000　0011
POP	栈顶弹出至累加器低位	1	1	1011　1110　0011　0010
POPD	栈顶弹出至数据存储器（直接或间接）	1	1	1000　1010　IAAA　AAAA
PSHD	数据存储器数值进栈（直接或间接）	1	1	0111　0110　IAAA　AAAA
PUSH	累加器低位进栈	1	1	1011　1110　0011　1100
SAR	将指定的 AR 存储至指定的数据单元（直接或间接）	1	1	1000　0ARX　IAAA　AAAA
SPH	存储 PREG 高位（直接或间接）	1	1	1000　1101　IAAA　AAAA
SPL	存储 PREG 低位（直接或间接）	1	1	1000　1100　IAAA　AAAA
SPLK	存储长立即数至数据存储器单元（直接或间接）	2	2	1010　1110　IAAA　AAAA +1 word
SST	存储状态存储器 ST0（直接或间接）	1	1	1000　1110　IAAA　AAAA
	存储状态存储器 ST1（直接或间接）	1	1	1000　1111　IAAA　AAAA
TBLR	表读（直接或间接）	1	3	1010　0110　IAAA　AAAA
TBLW	表写（直接或间接）	1	3	1010　0111　IAAA　AAAA

表 4.5　算术运算指令一览表

助记符	说　　明	字节	周期	操　作　码
ABS	累加器取绝对值	1	1	1011　1110　0000　0000
ADD	左移 0 至 15 位后加至累加器（直接或间接）	1	1	0010　SHFT　IAAA　AAAA
	左移 0 至 15 位后加至累加器，长立即数	2	2	1011　1111　1001　SHFT +1 word
	左移 16 位后加至累加器（直接或间接）	1	1	0110　0001　IAAA　AAAA
	加至累加器，短立即数	1	1	1011　1000　IIII　IIII
ADDC	带进位位加至累加器（直接或间接）	1	1	0110　0000　IAAA　AAAA
ADDS	抑制符号扩展加至累加器（直接或间接）	1	1	0110　0010　IAAA　AAAA
ADDT	按 TREG 规定进行移位（0 至 15）后加至累加器（直接或间接）	1	1	0110　0011　IAAA　AAAA
ADRK	将短立即常数加至当前 AR	1	1	0111　1000　IIII　IIII
APAC	PREG 加至累加器	1	1	1011　1110　0000　0100
MAC	乘且累加（直接或间接）	2	3	1010　0010　IAAA　AAAA +1 word

（续）

助记符	说　　明	字节	周期	操　作　码
MACD	乘且累加，并将被寻址数据移至下一单元（直接或间接）	2	3	1010　0011　IAAA　AAAA +1 word
MPY	TREG 乘以被寻址数据（直接或间接）	1	1	0101　0100　IAAA　AAAA
	TREG 乘以 13 位短立即常数	1	1	1101　IIII　IIII　IIII
MPYA	累加前次乘积，再将 TREG 与被寻址数据相乘（直接或间接）	1	1	0101　0000　IAAA　AAAA
MPYS	减去前次乘积，再将 TREG 与被寻址数据相乘（直接或间接）	1	1	0101　0001　IAAA　AAAA
MPYU	无符号乘（直接或间接）	1	1	0101　0101　IAAA　AAAA
SBRK	从当前 AR 中减去短立即常数	1	1	0111　1100　IIII　IIII
SPAC	从累加器减去 PREG	1	1	1011　1110　0000　0101
SQRA	平方并累加前次乘积（直接或间接）	1	1	0101　0010　IAAA　AAAA
SQRS	平方并减去前次乘积（直接和间接）	1	1	0101　0011　IAAA　AAAA
SUB	左移 0 至 15 位后从累加器中减去（直接或间接）	1	1	0011　SHFT　IAAA　AAAA
	长立即数左移 0 至 15 位后从累加器中减去	2	2	1011　1111　1010　SHFT +1 word
	左移 16 位后从累加器中减去	1	1	0110　0101　IAAA　AAAA
	从累加器中减去短立即数	1	1	1011　1010　IIII　IIII
SUBB	带借位从累加器减（直接或间接）	1	1	0110　0100　IAAA　AAAA
SUBC	条件减（直接或间接）	1	1	0000　1010　IAAA　AAAA
SUBS	抑制符号扩展从累加器减（直接或间接）	1	1	0110　0110　IAAA　AAAA
SUBT	按 TREG 的规定移位（0～15）后从累加器减（直接或间接）	1	1	0110　0111　IAAA　AAAA

表 4.6　逻辑运算一览表

助记符	说　　明	字节	周期	操　作　码
AND	累加器和数值逻辑"与"（直接或间接）	1	1	0110　1110　IAAA　AAAA
	长立即数左移 0 至 15 位后和累加器逻辑"与"	2	2	1011　1111　1011　SHFT +1 word
	长立即数左移 16 位后和累加器逻辑"与"	2	2	1011　1110　1000　0001 +1 word
CMPL	累加器求补	1	1	1011　1110　0000　0001
NEG	累加器求负		1	1011　1110　00000　0010
NORM	规格化累加器内容，间接	1	1	1010　0000　IAAA　AAAA
OR	被寻址的数据和累加器逻辑"或"（直接或间接）	1	1	0110　1101　IAAA　AAAA
	长立即数左移 0 至 15 位后和累加器逻辑"或"	2	2	1011　1111　1100　SHFT +1 word
	长立即数左移 16 位后和累加器逻辑"或"	2	2	1011　1110　1000　0010 +1 word
ROL	累加器逻辑循环左移	1	1	1011　1110　0000　1100
ROR	累加器逻辑循环右移	1	1	1011　1110　0000　1101
SACH	累加器高位左移后存入数据存储器（直接或间接）	1	1	1001　1SHF　IAAA　AAAA
SACL	累加器低位左移后存入数据存储器（直接或间接）	1	1	1001　0SHF　IAAA　AAAA
SFL	累加器算术左移	1	1	1011　1110　0000　1001
SFR	累加器算术右移	1	1	1011　1110　0000　1010

（续）

助记符	说　　明	字节	周期	操　作　码
XOR	被寻址的数据和累加器逻辑“异或”(直接或间接)	1	1	0110　1100　IAAA　AAAA
	长立即数左移 0 至 15 位后和累加器逻辑“或”	2	2	1011　1111　1101　SHFT +1 word
	长立即数左移 16 位后和累加器逻辑“或”	2	2	1011　1110　1000　0011 +1 word
ZALR	累加器低位置零并带舍入装载累加器高位(直接或间接)	1	1	0110　1000　IAAA　AAAA

表 4.7　分支转移指令一览表

助记符	说　　明	字节	周　期	操　作　码
B	无条件转移，间接	2	4	0111　1001　IAAA　AAAA +1 word
BACC	转移至累加器低 16 位指定的地址	1	4	1011　1110　0010　0000
BANZ	当前 AR 非零则转移	2	4(条件成立) 2(条件不成立)	0111　1011　IAAA　AAAA +1 word
BCND	条件转移	2	4(条件成立) 2(任一条件不成立)	1110　00TP　ZLVC　ZLVC +1 word
CALA	调用累加器低 16 位指定地址处的子程序	1	4	1011　1110　0011　0000
CALL	调用子程序，间接	2	4	0111　1010　IAAA　AAAA +1 word
CC	条件调用	2	4(条件成立) 2(任一条件不成立)	1110　10TP　ZLVC　ZLVC +1 word
INTR	软中断	1	4	1011　1110　011I　NTR#
NMI	不可屏蔽中断	1	4	1011　1110　0101　0010
RET	从子程序返回	1	4	1110　1111　0000　0000
RETC	条件返回	1	4(条件成立) 2(任一条件不成立)	1110　11TP　ZLVC　ZLVC
TRAP	软件陷阱中断	1	4	1011　1110　0101　0001
BIT	位测试(直接或间接)	1	1	0100　BITX　IAAA　AAAA
BITT	测试由 TREG 指定的位(直接或间接)	1	1	0110　1111　IAAA　AAAA
CLRC	清 C 位	1	1	1011　1110　0100　1110
	清 CNF 位	1	1	1011　1110　0100　0100
	清 INTM 位	1	1	1011　1110　0100　0000
	清 OVM 位	1	1	1011　1110　0100　0010
	清 SXM 位	1	1	1011　1110　0100　0110
	清 TC 位	1	1	1011　1110　0100　1010
	清 XF 位	1	1	1011　1110　0100　1100
CMPR	比较当前 AR 和 AR0	1	1	1011　1111　0100　01CM
IDLE	空闲直至中断发生	1	1	1011　1110　0010　0010
NOP	空操作	1	1	1000　1011　0000　0000
RPT	重复执行下一条指令(直接或间接)	1	1	0000　1011　IAAA　AAAA
	重复执行下一条指令，短立即数	1	1	1011　1011　IIII　IIII
SETC	置 C 位	1	1	1011　1110　0100　1111
	置 CNF 位	1	1	1011　1110　0100　0101
	置 INTM 位	1	1	1011　1110　0100　0001
	置 OVM 位	1	1	1011　1110　0100　0011
	置 SXM 位	1	1	1011　1110　0100　0111
	置 TC 位	1	1	1011　1110　0100　1011
	置 XF 位	1	1	1011　1110　0100　1101
SPM	置乘积移位方式	1	1	1011　1111　0000　00PM

4.3 传送指令

数据传送指令一共有30条，既可以将数据传送到累加器，也可将数据传送到寄存器；有数据存储器间的传送指令，也有程序存储器和数据存储器间的传送指令；还有对I/O单元的数据输入输出指令。有的数据传送指令不仅仅传送数据，还有移位及累加功能。数据传送指令是在编程中使用最为频繁的一类指令，下面逐条详细介绍。

□数据存储器至数据存储器间的块传送

1) BLDD #lk, dma	源地址为长立即数，直接寻址
2) BLDD #lk, ind[, ARn]	源地址为长立即数，间接寻址
3) BLDD dma, #lk	目标地址为长立即数，直接寻址
4) BLDD ind, #lk[, ARn]	目标地址为长立即数，间接寻址

●该指令将数据存储器中的一块源数据字连续地复制到指定的目的数据存储单元中。指令中的长立即数#lk是源(目的)的首地址(不是数据!)，块的长度放在重复计数器RPTC中(由RPT指令设置)。

●该指令执行前，会将PC加1并保存到微堆栈，让出PC装源(目的)首地址；随后将源数据复制到目的地址规定的单元中，并且PC自动加1，RPTC自动减1；前面这一步的操作持续到RPTC=0为止；当RPTC=0时，保存在微堆栈的PC值会自动弹回到PC中，使得下一条指令能正常运行。

●对于第1条指令，是将一个(串)数据放到一个固定地址上；对于第3条指令是将一个固定地址上数据放到一个(串)目的地址的单元中。

●对于间接寻址，按规定修改当前AR和ARP。因此，可以实现将一串数据放到一串目的地址的单元中。

●对于专用寄存器(存储器映射的寄存器)之间的数据传送不能使用该指令。

●当进入到重复运行时，即启动了RPT流水线，BLDD就变成了单周期指令。

●该指令不影响任何状态位。

例4.10 BLDD #300h, 20h ;(DP=6)

	执行指令前		执行指令后
数据存储器		数据存储器	
300h	0h	300h	0h
320h	0Fh	320h	0h

例4.11 BLDD *+, #321h, AR3

	执行指令前		执行指令后
ARP	2	ARP	3
AR2	301h	AR2	302h
数据存储器		数据存储器	
301h	01h	301h	01h
321h	0Fh	321h	01h

□程序存储器至数据存储器间的块传送

1) BLPD #pma, dma　　源地址为长立即数，直接寻址

2) BLPD #pma, ind[, ARn]　　源地址为长立即数，间接寻址

●该指令将程序存储器中的一块源数据字连续地复制到指定的目的数据存储单元中。指令中的长立即数#pma是源首地址(不是数据!)，块的长度放在重复计数器RPTC中(由RPT指令设置)。

●该指令执行前，会将PC加1并保存到微堆栈，让出PC装源首地址；随后将源数据复制到目的地址规定的单元中，并且PC自动加1，RPTC自动减1；前面这一步的操作持续到RPTC=0为止；当RPTC=0时，保存在微堆栈PC值会自动弹回到PC中，使得下一条指令能正常运行。

●对于直接寻址，是将一串程序存储器的数据放到一个固定的数据存储器地址上；

●对于间接寻址，按规定修改当前AR和ARP。因此，可以实现将一串程序存储器的数据放到一串数据存储器的目的地址单元中。

●当进入到重复运行时，即启动了RPT流水线，BLDD就变成了单周期指令。

●该指令不影响任何状态位。

例 4.12　BLPD　#800h, 00h　　;(DP=6)

	执行指令前		执行指令后
数据存储器		数据存储器	
800h	0Fh	800h	0Fh
300h	0h	300h	0Fh

例 4.13　BLPD　#800h, *, AP7

	执行指令前		执行指令后
ARP	0	ARP	7
AR0	310h	AR0	310h
程序存储器		程序存储器	
800h	1111h	800h	1111h
数据存储器		数据存储器	
310h	0100h	310h	1111h

□片内数据存储器中的数据传送

1) DMOV dma　　直接寻址

2) DMOV ind[, ARn]　　间接寻址

●该指令把指定的数据存储器内容复制到该地址加1的地址中，原单元的内容保持不变，即数据存储器地址→数据存储器地址+1。

●DMOV只工作于片内数据RAM块。若某块RAM被配置为数据存储器，则DMOV就可在该块内工作。另外，数据传送功能可连续通过块间的边界。

●该指令不能用于外部数据存储器。

●该指令对于处理DSP中的Z^{-1}延迟很有效。DMOV功能也可由LTD和MACD来实现

(详见 LTD 和 MACD 指令)。

●该指令受 CNF 状态位影响。

例 4.14 DMOV DAT8 ;(DP=6)

数据存储器	执行指令前	数据存储器	执行指令后
308h	43h	308h	43h
309h	2h	309h	43h

例 4.15 DMOV *, AR1

	执行指令前		执行指令后
ARP	0	ARP	1
AR0	30Ah	AR0	30Ah
数据存储器		数据存储器	
30Ah	40h	30Ah	40h
30Bh	41h	30Bh	40h

□从端口输入数据

1) IN dma, PA　　直接寻址

2) IN ind, PA[, ARn]　　间接寻址

●IN 指令从 I/O 单元读入 16 位值到指定的数据存储单元。引脚$\overline{IS}$变为低电平,表示访问 I/O 口。STRB、W/$\overline{R}$ 和 READY 时序与读外部数据存储器一样。

●重复指令 RPT 同 IN 指令一起使用时,可以从 I/O 空间连续读入多个字至数据空间。

●该指令不受任何状态位影响。

例 4.16 IN 7, 1000h ;从口地址为 1000h 的外设读数据,并将数存至数据存储器单元 307h(DP=6)。

例 4.17 IN *, 5h ;从口地址为 5h 的外设读数据,并将数存至当前辅助寄存器所指定的数据存储器单元。

□左移后装入累加器

1) LACC dma [, shift]　　直接寻址

2) LACC dma, 16　　直接寻址并左移 16 位

3) LACC ind[, shift[, ARn]]　　间接寻址

4) LACC ind, 16[, ARn]　　间接寻址并左移 16 位

5) LACC #lk[, shift]　　长立即寻址

●第 1 条到第 4 条指令将指定的数据存储器地址的内容左移后(shift 位或 16 位)送入累加器。

●第 5 条指令中的 #lk 是一个 16 位长的数据,不是地址。该条指令将数据左移后送入累加器。

●在移位时,低位填零;高位受状态位 SXM 的影响;若 SXM=1 进行符号扩展;若 SXM=0 不进行符号扩展,填零即可。

例 4.18 LACC 6,4 ;(DP=8;地址 0400h-047Fh,SXM=0)

	执行指令前			执行指令后	
数据存储器			数据存储器		
406h		01h	406h		01h
ACC	X (C)	012345678h	ACC	X (C)	10h

例 4.19 LACC *,4 ;(SXM=0)

	执行指令前			执行指令后	
ARP		2	ARP		2
AR2		0300h	AR2		0300h
数据存储器			数据存储器		
300h		0FFh	300h		0FFh
ACC	X (C)	12345678h	ACC	X (C)	0FF0h

例 4.20 LACC #0F000h,1 ;(SXM=1)

	执行指令前			执行指令后	
ACC	X (C)	12345678h	ACC	X (C)	FFFFE000h

□装载累加器低位并清累加器高位

1) LACL dma 直接寻址

2) LACL ind[,ARn] 间接寻址

3) LACL #k 短立即寻址

●第 1 条和第 2 条指令将被寻址数据存储器单元的内容装入累加器的低 16 位,累加器的高位填零。

●第 3 条指令将 8 位短立即数 #k 扩展为 16 位(高位补 0)并装入累加器的低 16 位,累加器的高位填零。

●无论 SXM 为何状态,该指令都不进行符号扩展。

例 4.21 LACL 1 ;(DP=6;地址 0300h-037Fh)

	执行指令前			执行指令后	
数据存储器			数据存储器		
301h		0h	301h		0h
ACC	X (C)	7FFFFFFFh	ACC	X (C)	0h

例 4.22 LACL *－，AR4

		执行指令前			执行指令后
ARP		0	ARP		4
AR0		401h	AR0		400h
数据存储器			数据存储器		
401h		00FFh	401h		00FFh
ACC	X (C)	7FFFFFFFh	ACC	X (C)	0FFh

例 4.23 LACL #10h

		执行指令前			执行指令后
ACC	X (C)	7FFFFFFFh	ACC	X (C)	010h

□按 TREG 规定的左移后装入累加器低位

1) LACT dma　　　　直接寻址

2) LACT ind[，ARn]　间接寻址

●该指令将数据存储单元中的值左移后装入累加器。左移位数由寄存器 TREG 最低 4 位决定，可左移 0～15 位。

●该指令受状态位 SXM 的影响。移位时，若 SXM＝1，则累加器高位进行符号扩展；若 SXM＝0，则高位填零。

例 4.24 LACT 1 ；(DP＝6；地址 0300h-037Fh，SXM＝0)

		执行指令前			执行指令后
数据存储器			数据存储器		
301h		1376h	301h		1376h
TREG		14h	TREG		14h
ACC	X (C)	98F7EC83h	ACC	X (C)	13760h

例 4.25 LACT *－，AR3 ；(SXM＝1)

		执行指令前			执行指令后
ARP		1	ARP		3
AR1		310h	AR1		30Fh
数据存储器			数据存储器		
310h		0FF00h	310h		0FF00h
TREG		11h	TREG		11h
ACC	X (C)	98F7EC83h	ACC	X (C)	0FFFFFE00h

□装载辅助寄存器

1) LAR　ARx, dma　　　直接寻址

2) LAR　ARx, ind[,ARn]　　间接寻址

3) LAR　ARx, #k　　　短立即数

4) LAR　ARx, #lk　　　长立即数

●第 1 条和第 2 条指令将指定的数据存储器地址中的内容装入指定的辅助寄存器地址(ARx)。

●第 3 条和第 4 条指令将 8/16 位的常量装入指定的辅助寄存器地址(ARx)。无论 SXM 为何值,指定的常量作为无符号整数处理。

● LAR 和 SAR(存储辅助寄存器)指令可在子程序调用或中断处理时用来装载和存储辅助寄存器。若一个辅助寄存器没有用于间接寻址,则 LAR 和 SAR 指令可将该辅助寄存器用作附加的存储寄存器,从而可实现在不影响累加器的前提下交换数据存储器单元中的数。

例 4.26　LAR　AR0,16　　;(DP=6; 地址 0300h-037Fh)

	执行指令前		执行指令后
数据存储器		数据存储器	
310h	18h	310h	18h
AR0	6h	AR0	18h

例 4.27　LAR　AR4,*-,　;指令中指定的 AR 与 ARP 指定的 AR 相同,则不减

	执行指令前		执行指令后
ARP	4	ARP	4
数据存储器		数据存储器	
300h	32h	300h	32h
AR4	300h	AR4	32h

例 4.28　LAR　AR4,#01h

	执行指令前		执行指令后
AR4	0FF09h	AR4	01h

例 4.29　LAR　AR6,#3FFFh

	执行指令前		执行指令后
AR6	0h	AR6	3FFFh

□装载数据页指针

1) LDP dma　　　直接寻址

2) LDP ind[, ARn]　　间接寻址

3）LDP #k　　　　　短立即数

●第 1 条和第 2 条指令将指定的数据存储器单元的低 9 位(LSB)装入状态寄存器 ST0 的数据存储页指针(DP)。

●第 3 条指令将 9 位立即数装入状态寄存器 ST0 的数据存储页指针(DP)。

● DP 也可由 LST 指令装入。

●在直接寻址中，DP 提供 9 位页面地址(位于高端)，指令中指定的 7 位数为偏移地址(位于低端)，以形成可被指令寻址的 16 位数据存储器地址。

例 4.30　LDP　127　　　;(DP＝511：地址 FF80h-FFFFh)

	执行指令前		执行指令后
数据存储器		数据存储器	
FFFFh	FEDCh	FFFFh	FEDCh
DP	1FFh	DP	0DCh

例 4.31　LDP　#0h

	执行指令前		执行指令后
DP	1FFh	DP	0h

例 4.32　LDP　*，AR5

	执行指令前		执行指令后
ARP	4	ARP	5
AR4	300h	AR4	300h
数据存储器		数据存储器	
300h	06h	300h	06h
DP	1FFh	DP	06h

□装载乘积寄存器高位

1）LPH dma　　　　　直接寻址

2）LPH ind[，ARn]　　间接寻址

●这两条指令将指定的数据存储器地址内容装入 P 寄存器的高 16 位。

● P 寄存器的低位不受影响。

● LPH 指令对于中断和子程序调用后恢复 P 寄存器高位内容特别有用。

例 4.33　LPH　DAT0　　　;(DP＝4)

	执行指令前		执行指令后
数据存储器		数据存储器	
200h	0F79Ch	200h	0F79Ch
PREG	30079844h	PREG	0F79C9844h

例 4.34 LPH *，AR6

	执行指令前		执行指令后
ARP	5	ARP	6
AR5	200h	AR0	200h
数据存储器		数据存储器	
200h	0F79Ch	200h	0F79Ch
PREG	30079844h	PREG	0F79C9844h

□装载状态寄存器

1）LST #m，dma　　　直接寻址

2）LST #m，ind[，ARn]　　间接寻址

●LST 指令是向状态寄存器 ST0 或 ST1 装载指定数据存储器的内容。若 m=0，选择 ST0；若 m=1，选择 ST1。

●用 LST 指令装载 ST0 时，不影响 ST0 的 INTM 位。该位是可屏蔽中断的总屏蔽位，通过指令 SETC INTM 和 CLRC INTM 对其操作。

●虽然 LST #0 操作向 ARP 装入新值，但并不影响 ST1 寄存器中的 ARB 字段。

●在 LST #1 操作中，送入 ARB 中的值也被送入 ARP。

●若在间接寻址方式下用一操作数来指定下一 AR 值，则该操作数将被忽略，而将被寻址数据存储器单元所含值的 3 位最高有效位送入 ARP。

●状态寄存器中的保留位读出总为 1。这些位对写入不起作用。

●LST 指令用于子程序调用和中断后恢复状态寄存器。

●该指令影响 ARB，ARP，OV，OVM，DP，CNF，TC，SXM，C，XF 和 PM 状态位，但不影响 INTM。

例 4.35 MAR *，AR0

LST #0，*，AR1　　；将辅助寄存器 AR0 所寻址的数据存储器内容送
　　　　　　　　　；入状态寄存器 ST0，但不包括 INTM 位。尽管
　　　　　　　　　；指定了下一 ARP 值，但该值被忽略。另外，原
　　　　　　　　　；来的 ARP 也不送入 ARB。

例 4.36 LST #0，60h　　；(DP=0)

	执行指令前		执行指令后
数据存储器		数据存储器	
60h	2404h	60h	2404h
ST0	6E00h	ST0	2604h
ST1	05ECh	ST1	05ECh

例 4.37 LST #0,*-,AR1

	执行指令前		执行指令后
ARP	4	ARP	7
AR4	3FFh	AR4	3FEh
数据存储器		数据存储器	
3FFh	EE04h	3FFh	EE04h
ST0	EE00h	ST0	EE04h
ST1	E7ECh	ST1	E7ECh

例 4.38 LST #1,00h ;(DP=6)
;ARB 被装入新的 ARP 值。

	执行指令前		执行指令后
数据存储器		数据存储器	
300h	E1BCh	300h	E1BCh
ST0	0406h	ST0	E406h
ST1	09ECh	ST1	E1FCh

□装载 T 寄存器

1) LT dma 直接寻址

2) LT ind[, ARn] 间接寻址

●这两条指令将指定的数据存储器的内容装入 TREG 寄存器,为乘法作准备。

●该指令不影响任何状态位。

●与乘法相关的指令还可参见 LTA、LTD、LTP、LTS、MPY、MPYA、MPYS 和 MPYU 指令。

例 4.39 LT 24 ;(DP=8;地址 0400h-047Fh)

	执行指令前		执行指令后
数据存储器		数据存储器	
418h	62h	418h	62h
TREG	3h	TREG	62h

例 4.40 LT *，AR3

	执行指令前		执行指令后
ARP	2	ARP	3
AR2	418h	AR2	418h
数据存储器		数据存储器	
418h	62h	418h	62h
TREG	3h	TREG	62h

□装载 T 寄存器并累加前次乘积

1）LTA dma　　　　　直接寻址

2）LTA ind[,ARn]　　间接寻址

●该指令将指定存储器地址中的内容装入 TREG 寄存器。同时，乘积寄存器的内容按 PM 状态位所定义的值移位后与累加器的内容相加，结果保留在累加器中。

●若结果产生进位，则进位位被置位(C=1)；否则，进位位被清零(C=0)。

●LTD 指令的功能包括了 LTA 指令的功能。

●受 PM 和 OVM 的影响，并影响 C 和 OV 状态位。

例 4.41 LTA 36 ；(DP=6；地址 0300h 至 037Fh；PM=0：乘积不移位)

	C	执行指令前		C	执行指令后
数据存储器			数据存储器		
324h		62h	324h		62h
TREG		3h	TREG		62h
PREG		0Fh	PREG		0Fh
ACC	X	5h	ACC	0	14h

例 4.42 LTA *，AR5 ；(PM=0)

	C	执行指令前		C	执行指令后
数据存储器			数据存储器		
ARP		4h	ARP		5h
AR4		324h	AR4		324h
数据存储器			数据存储器		
324h		62h	324h		62h
TREG		3h	TREG		62h
PREG		0Fh	PREG		0Fh
ACC	X	5h	ACC	0	14h

□装载 T 寄存器，累加前次乘积并移动数据

1) LTD dma　　　　直接寻址

2) LTD ind[,ARn]　　间接寻址

●该指令要完成三个动作。首先将指定的存储器地址中的内容装入 TREG 寄存器。然后 PREG 寄存器的内容按 PM 状态位所规定的值移位后加到累加器中。同时，指定的数据存储器地址的内容被复制到地址加 1 的数据存储器中。

●该指令适用于所有被配置为数据存储器的片内 RAM 块。数据传送功能可通过连续存储块的边界，但不能用于外部数据存储器和存储器映象寄存器。

●若 LTD 被用于外部数据存储器，则功能与 LTA 相同，即前一次结果被累加，TREG 寄存器从外部数据存储器装入数，但不会发生数据移动。

●若加的结果产生进位，则进位位被置位(C=1)；否则，进位位被清零(C=0)。

●指令受 PM 和 OVM 的影响，并影响 C 和 OV 状态位。

例 4.43　LTD　126　；(DP=7：地址 0380h 至 03FFh；PM=0：乘积不移位)

	执行指令前		执行指令后
数据存储器		数据存储器	
3FEh	62h	3FEh	62h
数据存储器		数据存储器	
3FEh	0h	3FEh	62h
TREG	3h	TREG	62h
PREG	0Fh	PREG	0Fh
ACC	C: X　5h	ACC	C: 0　14h

例 4.44　LTD　*，AR3　；(PM=0)

	执行指令前		执行指令后
ARP	1	ARP	3
AR1	3FEh	AR1	3FEh
数据存储器		数据存储器	
3FEh	62h	3FEh	62h
3FFh	0h	3FFh	62h
TREG	3h	TREG	62h
PREG	0Fh	PREG	0Fh
ACC	C: X　5h	ACC	C: 0　14h

注：LTD 的数据移动功能仅适用于片内数据存储器 PAM 块。

□装载 T 寄存器并将 P 寄存器存至累加器

1) LTP dma　　　　直接寻址

2) LTP ind[,ARn]　　间接寻址

●该指令先将被寻址数据存储器单元的内容装入 TREG 寄存器。

●再将移位后的 PREG 寄存器的值存入累加器。移位受 PM 状态位的控制。

例 4.45　LTP　36　　;(DP=6:地址 0300h 至 037Fh;PM=0:乘积不移位)

	C	执行指令前		C	执行指令后
数据存储器			数据存储器		
324h		62h	324h		62h
TREG		3h	TREG		62h
PREG		0Fh	PREG		0Fh
ACC	X	5h	ACC	X	0Fh

例 4.46　LTP　*,AR5　　;(PM=0)

	C	执行指令前		C	执行指令后
ARP		2	ARP		5
AR2		324h	AR2		324h
数据存储器			数据存储器		
324h		62h	324h		62h
TREG		3h	TREG		62h
PREG		0Fh	PREG		0Fh
ACC	X	5h	ACC	X	0Fh

□装载 T 寄存器并减去前次乘积

1) LTS dma　　　　直接寻址

2) LTS ind[,ARn]　间接寻址

●该指令先将被寻址数据存储器单元的内容装入 TREG 寄存器。同时乘积寄存器的内容按 PM 状态位所规定的值进行移位,然后从累加器的内容中减去 PREG 移位后的值,结果保存在累加器中。

●若减的结果产生借位,则进位位被清零(C=0);否则,进位位被置位(C=1)。

例 4.47　LTS　DAT36　　;(DP=6:地址 0300h 至 037Fh;PM=0:乘积不移位)

	C	执行指令前		C	执行指令后
数据存储器			数据存储器		
324h		62h	324h		62h
TREG		3h	TREG		62h
PREG		0Fh	PREG		0Fh
ACC	X	05h	ACC	0	0FFFFFFF6h

例 4.48 LTS *, AR2 ; (PM=0)

	执行指令前		执行指令后
ARP	1	ARP	2
AR1	324h	AR1	324h
数据存储器		数据存储器	
324h	62h	324h	62h
TREG	3h	TREG	62h
PREG	0Fh	PREG	0Fh
ACC (C: X)	05h	ACC (C: 0)	0FFFFFFF6h

□修改辅助寄存器

1) MAR dma 直接寻址

2) MAR ind[,ARn] 间接寻址

●在直接寻址方式下,MAR 指令为空操作,不影响状态位。对所指向的存储器不起作用。

●在间接寻址方式下,执行该指令可以修改辅助寄存器和 ARP 的值,当 MAR 修改 ARP 的值时,ARP 原有的值被复制到 ST1 的 ARB 字段。

●任何 MAR 所能实现的间接寻址操作,用其它任何支持间接寻址的指令也能实现。

● ARP 也可由 LST 指令装入。

●对于装入辅助寄存器,见 LAR 指令的说明。对于将辅助寄存器的值存入数据存储器,见 SAR 指令。

例 4.49 MAR *, AR1 ;ARP 装入 1

	执行指令前		执行指令后
ARP	0	ARP	1
ARB	7	ARB	0

例 4.50 MAR *+,AR5 ;将当前辅助寄存器(AR1)增 1,并向 ARP 装入 5

	执行指令前		执行指令后
AR1	34h	AR1	35h
ARP	1	ARP	5
ARB	0	ARB	1

□从端口输出数据

1) OUT dma, PA 直接寻址

2) OUT ind, PA[,ARn] 间接寻址

● OUT 指令将数据存储器单元的 16 位值写到指定 I/O 单元。不影响状态位。

●引脚$\overline{IS}$变低，用以指示访问 I/O 口。

●$\overline{STRB}$、R/$\overline{W}$和 Ready 信号的时序与写外部数据存储器时序相同。

●RPT 指令可以同 OUT 指令一起使用，用来将连续多个字从数据存储器写到 I/O 空间。

例 4.51 OUT DAT0，100h ；(DP＝4)将数据存储器单元 200h 处的数据字写到口地址为 100h 的外设上。

例 4.52 OUT *， 100h ；将当前辅助寄存器所指的数据存储单元中的字写到口地址为 100h 的外设上。

□将乘积寄存器装入累加器

PAC

●该指令将 P 寄存器中的内容按 PM 状态位所规定的值移位后，送入累加器。

例 4.53 PAC ；(PM＝0：乘积不移位)

	C	执行指令前		C	执行指令后
PREG		144h	PREG		144h
ACC	X	23h	ACC	X	144h

□栈顶弹出至累加器低位

POP

●执行该指令后，栈顶(TOS)内容复制到累加器低位中，栈中各值均上弹一级，累加器高位填零。

●硬件堆栈有 8 个后进先出的单元。每发生一次弹出时，每个堆栈中值均被复制到上一级栈单元，栈顶内容则被移出。栈底的最后两个字具有相同的值。如果在任何入栈前有 7 次出栈(使用 POP、POPD、RETC 或 RET 指令)，则栈中各级含有相同的值。

●没有提供对堆栈下溢的检查。该指令不影响状态位。

例 4.54 POP

	C	执行指令前		C	执行指令后
ACC	X	82h	ACC	X	45h
Stack		45h	Stack		16h
		16h			7h
		7h			33h
		33h			42h
		42h			56h
		56h			37h
		37h			61h
		61h			61h

□栈顶弹出至数据存储器

1) POPD dma 直接寻址

2) POPD ind[, ARn]　间接寻址。

●该指令将栈顶的数值传送到指令规定的数据存储器单元。堆栈弹出一级，栈顶以下的七级堆栈单元也依次上弹一级。

●堆栈的其他操作与 POP 指令相同。

例 4.55　POPD DAT10　　;(DP=8)

数据存储器	执行指令前	数据存储器	执行指令后
40Ah	55h	40Ah	92h
Stack	92h	Stack	72h
	72h		8h
	8h		44h
	44h		81h
	81h		75h
	75h		32h
	32h		0AAh
	0AAh		0AAh

□数据存储器值进栈

1) PSHD dma　　直接寻址

2) PSHD ind[, ARn]　间接寻址

●将指令规定的数据存储器单元内容送入栈顶(TOS)。堆栈中低 7 个单元的值都下推一级。最低一级单元的内容将丢失。

●该指令不影响状态位。

例 4.56　PSHD　127　　;(DP=3:地址 0180h～01FFh)

数据存储器	执行指令前	数据存储器	执行指令后
1FFh	65h	1FFh	65h
Stack	2h	Stack	65h
	33h		2h
	78h		33h
	99h		78h
	42h		99h
	50h		42h
	0h		50h
	0h		0h

□累加器低位值进栈

PUSH

●执行该指令后堆栈内容下推一级，累加器低位字复制到硬件堆栈的顶部。

●硬件堆栈是一个 8 单元的后进先出堆栈，如果在一次弹出前，发生 8 次入栈(由 CALA、CALL、CC、PSHD、PUSH、TRAP、INTR 或 NMI 指令引起)，则第一个写入栈中的数值将由于连续入栈而丢失。

例 4.57 PUSH

	C	执行指令前		C	执行指令后
ACC	X	7h	ACC	X	7h
Stack		2h	Stack		7h
		5h			2h
		3h			5h
		0h			3h
		12h			0h
		86h			12h
		54h			86h
		3Fh			54h

□存储辅助寄存器

1) SAR ARx, dma　　　　　直接寻址

2) SAR ARx,ind[, ARn]　　间接寻址

●将指定的辅助寄存器(ARx)内容存入被寻址数据存储器单元。

●在间接寻址方式中，SAR 指令同时也要对当前辅助寄存器内容进行修改，但 SAR 将在增、减辅助寄存器内容前将辅助寄存器值存入数据存储器。

例 4.58 SAR　AR0,DAT30　　　;(DP=6:地址 0300h-037Fh)

	执行指令前		执行指令后
AR0	37h	AR0	37h
数据存储器		数据存储器	
31Eh	18h	31Eh	37h

例 4.59 SAR　AR0, * +

	执行指令前		执行指令后
ARP	0	ARP	0
AR0	401h	AR0	402h
数据存储器		数据存储器	
401h	0h	401h	401h

□存储 PREG 寄存器高位

1) SPH dma

2) SPH ind[, ARn]

●该指令将 PREG 寄存器按 PM 位所规定的方式移位后，数值的高 16 位存储到数据存储器中。

●执行时，先将 32 位 PREG 寄存器复制到乘积移位器中，然后在此按 PM 位所规定的值进行移位。若选择右移 6 位方式，则高位进行符号扩展，低位丢失；若选择左移方式，则高位丢失，低位填零；若 PM=00，移位不发生。移位后数值的高 16 位存入数据存储器。

● PREG 寄存器值和累加器值都不受本指令影响。

例 4.60 SPH DAT3 ;(DP=4：地址 0200h～027Fh；PM=0；不移位)

	执行指令前		执行指令后
PREG	FE079844h	PREG	FE079844h
数据存储器		数据存储器	
203h	4567h	203h	FE07h

例 4.61 SPH * , AR7 ;(PM=2；左移 4 位)

	执行指令前		执行指令后
ARP	6	ARP	7
AR6	203h	AR6	203h
PREG	FE079844h	PREG	FE079844h
数据存储器		数据存储器	
203h	4567h	203h	E079h

□存储 P 寄存器低位

1) SPL dma

2) SPL ind[, ARn]

●该指令将 PREG 寄存器按 PM 位所规定的方式移位后数值的低 16 位存储到数据存储器中。

●执行时，先将 32 位 PREG 寄存器复制到乘积移位器中，然后在此按 PM 位所规定的值进行移位。若选择右移 6 位方式，则高位进行符号扩展，低位丢失；若选择左移方式，则高位丢失，低位填零；若 PM=00，移位不发生。移位后数值的低 16 位存入数据存储器。

● PREG 寄存器值和累加器值都不受本指令影响。

例 4.62 SPL DAT5 ;(DP=4：地址 0200h-027Fh；PM=2；左移 4 位)

	执行指令前		执行指令后
PREG	0FE079844h	PREG	0FE079844h
数据存储器		数据存储器	
205h	4567h	205h	08440h

□存储长立即数值至数据存储器

1) SPLK #lk, dma 直接寻址

2) SPLK #lk, ind[, ARn] 间接寻址

● SPLK 指令允许将一个 16 位的数值写入任何一个数据存储单元。

●该指令不影响状态位。

例 4.63 SPLK #7FFFh, DAT3 ;(DP=6)

	执行指令前		执行指令后
数据存储器		数据存储器	
303h	FE07h	303h	7FFFh

□存储状态寄存器

1) SST #m, dma 直接寻址

2) SST #m, ind[, ARn] 间接寻址

●该指令将指定的状态寄存器ST0或ST1存入数据存储器。若m=0,选择ST0;若m=1,选择ST1。

●在直接寻址方式中,不管ST0中的数据页指针(DP)为何值,指定的状态寄存器总是被存入第0页。虽然处理器自动访问第0页,但DP不会被修改。这就允许在DP不被修改地情况下存储ST0。在0页内的特定存储单元由指令给出。

●在间接寻址方式中,存储地址从被选的辅助寄存器获取。从而,指定的状态寄存器内容可被存储至数据存储器中任一页的地址。

例 4.64 SST #0, 96 ;直接寻址:自动访问数据页0

	执行指令前		执行指令后
ST0	0A408h	ST0	0A408h
数据存储器		数据存储器	
60h	0Ah	60h	0A408h

例 4.65 SST #1, *, AR7 (间接寻址)

	执行指令前		执行指令后
ARP	0	ARP	7
AR0	300h	AR0	300h
ST1	2580h	ST1	2580h
数据存储器		数据存储器	
300h	0h	300h	2580h

□表读

1) TBLR dma 直接寻址

2) TBLR ind[,ARn] 间接寻址

●TBLR指令把程序存储器单元中的一个字,传送到由指令指定的数据存储器单元。

●程序存储器单元地址由累加器的低16位定义。该操作执行完从程序存储器读后,紧接着向数据存储器写。

●当与重复指令RPT一同使用时,TBLR便成为单周期指令。该指令执行前,会将PC加1并保存到微堆栈;然后将由累加器的低16位ACC(15:0)定义的表的首地址装入到程序计数器PC;以后每个周期PC自动加1,RPTC自动减1,该操作持续到RPTC=0为止;当RPTC=0时,保存在微堆栈PC值会自动弹回到PC中,使得下一条指令能正常运行。

●该指令不影响状态位。

例 4.66 TBLR DAT6 ;(DP=4;地址 0200h-027Fh)

	执行指令前		执行指令后
ACC	23h	ACC	23h
程序存储器		程序存储器	
23h	306h	23h	306h
数据存储器		数据存储器	
206h	75h	206h	306h

例 4.67 TBLR * ,AR7

	执行指令前		执行指令后
ARP	0	ARP	7
AR0	300h	AR0	300h
ACC	24h	ACC	24h
程序存储器		程序存储器	
24h	307h	24h	307h
数据存储器		数据存储器	
300h	75h	300h	307h

□表写

1) TBLW dma　　　　　直接寻址

2) TBLW ind[, ARn]　　间接寻址

● TBLW 指令把数据存储器单元中的一个字传送至程序数据存储器。

●数据存储器地址由指令给定,程序存储器地址由累加器的低 16 位指定。该指令首先完成从数据存储器中读,紧接着向程序存储器写。

●当与重复指令 RPT 一同使用时,TBLW 便成为单周期指令。该指令执行前,会将 PC 加 1 并保存到微堆栈;然后将由累加器的低 16 位 ACC(15:0)定义的表的首地址装入到程序计数器 PC;以后每个周期 PC 自动加 1,RPTC 自动减 1,该操作持续到 RPTC=0 为止;当 RPTC=0 时,保存在微堆栈 PC 值会自动弹回到 PC 中,使得下一条指令能正常运行。

●该指令不影响状态位。

例 4.68 TBLW DAT5 ;(DP=32;地址 1000h-107Fh)

	执行指令前		执行指令后
ACC	257h	ACC	257h
数据存储器		数据存储器	
1005h	4339h	1005h	4339h
程序存储器		程序存储器	
257h	306h	257h	4339h

例 4.69 TBLW *

	执行指令前		执行指令后
ARP	6	ARP	6
AR6	1006h	AR6	1006h
ACC	258h	ACC	258h
数据存储器		数据存储器	
1006h	4340h	1006h	4340h
程序存储器		程序存储器	
258h	307h	258h	4340h

4.4 算术操作指令

算术运算类指令共有 22 条，可以完成 4 种基本的算术运算，即加减乘除。其中除法没有直接的操作指令，可以利用条件减指令来完成。

对于不同的算术运算指令，执行时会受到一些状态位的影响，如 PM、OVM 和 SXM，也会影响状态位，如 OV 和 C。下面逐条予以介绍。

□累加器取绝对值

ABS

该指令将累加器的内容取绝对值后放回累加器，指令执行前后的内容均以二进制补码解释。

●执行此指令后，进位位(C)复位至零。

●要注意 8000 0000h 是一特殊情况。当溢出方式 OVM=0，8000 000h 在 ABS 指令执行后仍为 8000 0000h；当溢出方式 OVM=1，8000 0000h 在 ABS 指令执行后为 7FFFFFFFh。以上任一情况，OV 状态位均置 1。

●该条指令不受 SXM 的影响。

例 4.70 ABS

	C	执行指令前		C	执行指令后
ACC	X	1234h	ACC	0	1234h

□加至累加器

1) ADD dma[, shift] 直接寻址

2) ADD ind,[, shift[, ARn]] 间接寻址

3) ADD dma, 16 直接寻址并左移 16 位

4) ADD ind, 16[, ARn] 间接寻址并左移 16 位

5) ADD #k 短立即寻址

6) ADD #lk,[, shift] 长立即寻址

●第 1 条和第 2 条指令将被寻址数据存储器单元的内容左移后加到累加器中。移位时，低位填零，高位在 SXM=1 时进行符号扩展，在 SXM=0 时填零。结果存储在累加器中。

●第 3 条和第 4 条指令将被寻址数据存储器单元的内容左移 16 位后，加到累加器中。

●第 5 条指令将一个短立即常数加到累加器中，不进行移位，也不受 SXM 的影响。

●第 6 条指令将一个长立即数左移后加到累加器中。移位时，低位填零，高位在 SXM=1 时进行符号扩展，在 SXM=0 时填零。结果存储在累加器中。

●若使用间接寻址并更新 ARP，则必须指定一个移位操作数。若并不要求移位发生，则该操作数以 0 代替。例如：ADD *+,0,AR2

●正常情况下，若结果产生进位，则进位位被置 1(C=1)；否则，进位位被清零(C=0)。然而，当进行 16 位移位加法时，若结果产生进位，则进位位被置 1；否则进位位不变。这可使累加器在加一个 32 位数时，产生正确的符号进位。

●该指令都受 OVM 的影响，并且相加的结果会影响状态位 OV。

●在进行二进制补码加法时，状态位 OV 反映了相加结果是否溢出。对于无符号的加法，通过进位位 C 来判断是否溢出。

例 4.71 ADD 1，1 ；(DP=6)

	C	执行指令前		C	执行指令后
数据存储器			数据存储器		
301h		1h	301h		1h
ACC	X	2h	ACC	0	04h

例 4.72 ADD *+,0,AR0

	C	执行指令前		C	执行指令后
ARP		4	ARP		0
AR4		0302h	AP4		0303h
数据存储器			数据存储器		
302h		2h	302h		2h
ACC	X	2h	ACC	0	04h

例 4.73 ADD #1111h， 1 ；加长立即数且移 1 位

	C	执行指令前		C	执行指令后
ACC	X	2h	ACC	0	2224h

□带进位位加至累加器

1）ADDC dma 直接寻址

2）ADDC ind,[, ARn] 间接寻址

●该指令将被寻址数据存储器单元的内容及进位位值加到累加器，抑制符号扩展。

●进位位按正常方式置位：若结果产生进位，则进位位被置 1(C=1)；否则进位位被清零(C=0)。

●可利用 ADDC 实现多精度的运算。

●该指令不受 SXM 的影响，但受 OVM 的影响；相加的结果要影响 C 和 OV。

例 4.74 ADDC DAT300 ;(DP=6:地址 0300h 至 037Fh,DAT300 是 300h 的标号)

		执行指令前			执行指令后
数据存储器			数据存储器		
300h		04h	300h		04h
ACC	C: 1	13h	ACC	C: 0	18h

例 4.75 ADDC *-,AR4 ;(OVM=0)

		执行指令前			执行指令后
ARP		0	ARP		4
AR0		300h	AR0		2FFh
数据存储器			数据存储器		
300h		0h	300h		0h
ACC	C: 1	FFFFFFFFh	ACC	C: 1	0h
	OV: X			OV: 0	

□抑制符号扩展加至累加器

1) ADDS dma 直接寻址

2) ADDS ind,[, ARn] 间接寻址

●该指令将指定的数据存储器单元内容加到累加器中,抑制符号扩展。

●无论 SXM 为何值,数据均被作为 16 位无符号数处理,累加器内容则被视为有符号数。

● ADDS 所产生的结果与执行 SXM=0 且移位为 0 时的 ADD 指令结果一样。

●若结果产生进位,则进位位置 1(C=1);否则,进位位被清零(C=0)。

●该指令不受 SXM 的影响,但受 OVM 的影响;相加的结果要影响 C 和 OV。

例 4.76 ADDS 0 ;(DP=6:地址 0300h~037Fh)

		执行指令前			执行指令后
数据存储器			数据存储器		
300h		0F006h	300h		0F006h
ACC	C: X	00000003h	ACC	C: 0	0000F009h

例 4.77 ADDS *

		执行指令前			执行指令后
ARP		0	ARP		0
AR0		0300h	AR0		0300h
数据存储器			数据存储器		
300h		0FFFFh	300h		0FFFFh
ACC	C: X	7FFF0000h	ACC	C: 0	7FFFFFFFh

□按 T 寄存器内容移位后加至累加器

1) ADDT dma　　　　　　直接寻址

2) ADDT ind,[, ARn]　　　间接寻址

●该指令将指定的数据存储器中的值左移后加到累加器中,其结果取代累加器原来的内容。左移量由 TREG 寄存器低 4 位决定,其值为 0～15 位。

●若结果产生进位,则进位位置 1(C=1);否则,进位位被清零(C=0)。

●该指令受 SXM 和 OVM 的影响;相加的结果要影响 C 和 OV。

例 4.78　ADDT　127　　　;(DP=4:地址 0200h 至 027Fh,SXM=0)

数据存储器	C	执行指令前	数据存储器	C	执行指令后
027Fh		09h	027Fh		09h
TREG		0FF94h	TREG		0FF94h
ACC	X	0F715h	ACC	0	0F7A5h

□短立即数加至辅助寄存器

ADRK　#k　　　　短立即寻址

●该指令将 8 位立即数值靠右对齐加到当前辅助寄存器(由当前 ARP 的值指定),其结果替代该辅助寄存器的内容。

●加操作在 ARAU 中执行,且立即数值被作为 8 位正整数处理。在辅助寄存器上的所有算术运算都是无符号的。

例 4.79　ADRK　#80h

	执行指令前		执行指令后
ARP	5	ARP	5
AR5	4321h	AR5	43A1h

□P 寄存器加至累加器

APAC

●该指令先将 PREG 寄存器中的内容按照 ST1 寄存器中 PM 状态位所规定的值进行移位,然后加到累加器中。

●无论 SXM 为何值,PREG 寄存器始终要符号扩展。

● APAC 受 PM、OVM 的影响。相加结果要影响 C 和 OV。

● APAC 指令具有 LTA、LTD、MAC、MACD、MPTA 和 SQRA 指令的部分功能。

例 4.80　APAC　　　　;(PM=01)

	C	执行指令前		C	执行指令后
PREG		40h	PREG		40h
ACC	X	20h	ACC	0	A0h

□乘且累加

1) MAC pma, dma　　　　　　直接寻址

2) MAC pma, ind, [, ARn]　　　间接寻址

● MAC 指令可一次完成多步操作,并可与重复计数器实现块的操作。基本功能是把程序

存储器中的一串数与数据存储器中的一串(间接寻址)或一个(直接寻址)数分别相乘再累加。

●该指令操作的具体步骤是:将当前 PC 加 1 保存到微堆栈;将块的首地址 pma 装入 PC;将前次乘积的结果按 PM 状态位所规定的值移位,然后加到累加器,若加的结果产生进位,则进位位被置 1(C=1),否则,进位位被清除(C=0);将指定的数据存储器地址中的内容装入 TREG 寄存器;将存放在 TREG 寄存器中的数据存储器值乘以 PC 指定的程序存储器地址中的内容;PC 加 1,重复计数器减 1(第一次不做);如果重复计数器等于零停止,否则继续前面的操作。即

(PC)+1 ⟶PC;(PC)⟶MSTACK

pma ⟶PC

(ACC)+移位后的(PREG)⟶(ACC)

(dma)或(ind, [, ARn])⟶TREG

(TREG)×(PC)⟶PREG

(PC)+1 ⟶PC;

(RPTC)−1 ⟶RPTC(第一次不做)

如果(RPTC)不等于零,继续前面的操作。

(MSTACK)⟶PC

●数据和程序存储器单元可以是任何非保留的片内或片外存储器单元。若程序存储器是片内的 RAM 的 B0 块,则 CNF 位必须置为 1。

●当重复执行 MAC 指令时,包含在 PC 中的程序存储器地址在每次循环中增 1,这样就可以访问程序存储器中的一串操作数。若使用间接寻址来指定数据存储器地址,则在每次循环中可以访问一新的数据存储器地址。若使用直接寻址方式,则指定的数据存储器地址为一常量,在循环中该常量不能被修改。

●MAC 指令在有大量的乘加操作时非常有用,因为一旦 RPT 流水线启动后,在循环中 MAC 指令将变为单周期指令。

●该指令受 PM、OVM 的影响。结果要影响 C 和 OV。

例 4.81 MAC 0FF00h, 02h ;(DP=6,PM=0,CNF=1)

		执行指令前			执行指令后
数据存储器			数据存储器		
302h		23h	302h		23h
程序存储器			程序存储器		
FF00h		4h	FF00h		4h
TREG		45h	TREG		23h
PREG		458972h	PREG		8Ch
ACC	X	723EC41h	ACC	0	76975B3h
	C			C	

□乘且累加,其数据存储器的内容下移一步

1) MACD pma, dma 直接寻址

2) MACD pma, ind,[, ARn] 间接寻址

●该指令的操作与 MAC 基本类似,不同点只是要将当前数据存储器地址中的内容复制到地址加 1 的数据存储器中,即执行一次 DMOV 指令。

●该指令操作的具体步骤是：将当前 PC 加 1 保存到微堆栈；将块的首地址 pma 装入 PC；将前次乘积的结果按 PM 状态位所规定的值移位，然后加到累加器，若加的结果产生进位，则进位位被置 1(C=1)；否则，进位位被清除(C=0)；将指定的数据存储器地址中的内容装入 TREG 寄存器；将存放在 TREG 寄存器中的数据存储器值乘以 PC 指定的程序存储器地址中的内容；将当前数据存储器地址中的内容复制到地址加 1 的数据存储器中；PC 加 1，重复计数器减 1(第一次不做)；如果重复计数器等于零停止，否则继续前面的操作。即

(PC)+1 ⟶PC；(PC)⟶MSTACK

pma ⟶PC

(ACC)+移位后的(PREG)⟶(ACC)

(dma)或(ind，[，ARn])⟶TREG

(TREG)×(PC)⟶PREG

(PC)+1 ⟶PC；

(RPTC)−1 ⟶RPTC(第一次不做)

(数据存储器地址)⟶数据存储器地址+1

如果(RPTC)不等于零，继续前面的操作。

(MSTACK)⟶PC

●数据和程序存储器单元可以是任何非保留的片内或片外存储器单元。若程序存储器是片内的 RAM 的 B0 块，则 CNF 位必须置为 1。若 MACD 寻址存储器映射的寄存器或外部存储器作为数据存储器单元，则 MACD 功能与 MAC 相同，不会有数据移动操作(见 DMOV 指令说明)。

●当 MACD 指令重复执行时，包含在 PC 中的程序存储器地址在每次循环中增 1，这样就可以访问程序存储器中的一串操作数。若使用间接寻址来指定数据存储器地址，则在每次循环中可以访问一新的数据存储器地址。若使用直接寻址方式，则指定的数据存储器地址为一常量，在循环中该常量不能被修改。

●MACD 功能与 MAC 基本相同，所不同的是增加了在片内 RAM 块中移动数据的功能。这一特定功能使 MACD 非常适用于卷积和横向滤波的应用。

●当同 RPT 一同使用时，一旦 RPT 流水线被启动，MACD 指令将变为单周期指令。

例 4.82 MACD 0FF00h，08h ；(DP=6，地址 0300h～037Fh；PM=0；乘积不移位
；CNF=1；RAMB0 块被配置为程序寄存器。)

	执行指令前		执行指令后
数据存储器 308h	23h	数据存储器 308h	23h
数据存储器 309h	18h	数据存储器 309h	23h
程序存储器 FF00h	4h	程序存储器 FF00h	4h
TREG	45h	TREG	23h
PREG	458972h	PREG	8Ch
ACC	C: X　723EC41h	ACC	C: 0　76975B3h

□乘

1) MPY dma　　　　直接寻址

2) MPY ind,[,ARn]　　间接寻址

3) MPY #k　　　　短立即寻址

●操作数 k 为 13 位短立即数。

●第 1 条和第 2 条指令将 TREG 寄存器内容和被寻址数据存储器单元的内容相乘，其结果装入 PREG 寄存器中。

●第 3 条短立即寻址指令将 TREG 寄存器的内容与带符号的 13 位常数相乘。无论 SXM 位为何值，短立即数总是靠右对齐并在相乘之前进行符号扩展。

例 4.83　MPY　DAT13　　;(DP=8)

	执行指令前		执行指令后
数据存储器		数据存储器	
40Dh	7h	40Dh	7h
TREG	6h	TREG	6h
PREG	36h	PREG	2Ah

例 4.84　MPY　#031h

	执行指令前		执行指令后
TREG	2h	TREG	2h
PREG	36h	PREG	62h

□乘且累加前次乘积

1) MPYA dma　　　　直接寻址

2) MPYA ind,[,ARn]　　间接寻址

●该指令将 TREG 寄存器内容和被寻址数据存储器单元的内容相乘，其结果装入 PREG 寄存器中。

●然后将 PREG 寄存器中先前的乘积按 PM 状态位所定义的值进行移位，再加到累加器。

●该指令受 PM 和 OVM 影响，并影响 C 和 OV 状态位。

例 4.85　MPYA　DAT13　　;(DP=6,PM=0)

		执行指令前			执行指令后
数据存储器			数据存储器		
30Dh		7h	30Dh		7h
TREG		6h	TREG		6h
PREG		36h	PREG		2Ah
ACC	C: X	54h	ACC	C: 0	8Ah

□乘且减去前次乘积

1) MPYS dma　　　　直接寻址

2) MPYS ind,[,ARn]　间接寻址

●该指令将 TREG 寄存器内容和被寻址数据存储器单元的内容相乘，其结果装入 PREG 寄存器中。

●然后将 PREG 寄存器中先前乘积按 PM 状态位所定义的值进行移位，再从累加器中减去，最后结果放到累加器中。

●该指令受 PM 和 OVM 影响，并影响 C 和 OV 状态位。

例 4.86　MPYS　DAT13　;(DP=6,PM=0)

执行指令前 数据存储器			执行指令后 数据存储器		
30Dh		7h	30Dh		7h
TREG		6h	TREG		6h
PREG		36h	PREG		2Ah
ACC	C: X	54h	ACC	C: 1	1Eh

□无符号乘

1) MPYU dma　直接寻址

2) MPYU ind,[,ARn]　间接寻址

●该指令将 TREG 寄存器的无符号数与被寻址数据单元中的无符号数相乘，其结果装入 PREG 寄存器中。

●执行该指令时，乘法器的作用如同有符号的 17×17 位的乘法器，且这两个 17 位操作数的最高有效位都被强置为 0。

●当其他指令准备将得到的 PREG 寄存器值送入数据存储器或 CALU 时，PREG 寄存器值将首先被送入输出的乘积移位器，当 PM=3(右移 6 位方式)时，乘积移位器总要将 PREG 寄存器内容进行符号扩展。所以，若希望得到无符号的乘积时则不能使用该移位方式。

● MPYU 指令特别适用于多精度乘积，如将两个 32 位数相乘得到 64 位乘积。

例 4.87　MPYU　16　;(DP=4;地址 0200h～027Fh)

执行指令前 数据存储器		执行指令后 数据存储器	
210h	0FFFFh	210h	0FFFFh
TREG	0FFFFh	TREG	0FFFFh
PREG	1h	TREG	0FFFE0001h

□从当前辅助寄存器中减去立即数

SBRK #k　短立即寻址

●该指令将当前辅助寄存器(ARP 所规定的)减去靠右对齐的 8 位立即数值，其结果替换原有的辅助寄存器内容。

●减法在辅助寄存器算术单元(ARAU)中进行，立即数值被作为 8 位正整数处理。所有辅

助寄存器的算术运算都是无符号的。

例 4.88 SBRK #0FFh

	执行指令前		执行指令后
ARP	7h	ARP	7h
AR7	0h	AR7	FF01h

□从累加器中减去 P 寄存器

SPAC

●该指令将 PREG 寄存器中的内容按 PM 状态位所规定的值进行移位，然后从累加器的内容中减去，其结果存入累加器。

●无论 SXM 为何值，PREG 寄存器值始终要符号扩展。SPAC 指令具有 LTS、MPYS 和 SQRS 指令的部分功能。

●指令受 PM 和 OVM 影响，并影响 C 和 OV 状态位。

例 4.89 SPAC ;(PM=0)

		执行指令前			执行指令后
PREG		10000000h	PREG		10000000h
ACC	C: X	70000000h	ACC	C: 1	60000000h

□累加前次乘积再平方

1) SQRA dma 直接寻址

2) SQRA ind,[, ARn] 间接寻址

●该指令将 PREG 寄存器内容按 PM 状态位所规定的值移位后加到累加器中，然后将被寻址数据存储单元的值送入 TREG 寄存器，再将 TREG 寄存器的值与数据存储单元的值相乘(即求平方)，结果存至 PREG 寄存器。

●该指令受 PM 和 OVM 影响，并影响 C 和 OV 状态位。

例 4.90 SQRA DAT30 ;(DP=6;地址 0300h~037Fh;PM=0;乘积不移位)

		执行指令前			执行指令后
数据存储器			数据存储器		
31Eh		0Fh	31Eh		0Fh
TREG		3h	TREG		0Fh
PREG		12Ch	PREG		0E1h
ACC	C: X	1F4h	ACC	C: 0	320h

□减去前次乘积再平方

1) SQRS dma 直接寻址

2) SQRS ind,[,ARn] 间接寻址

●该指令将 PREG 寄存器内容按 PM 状态位所规定的值移位后从累加器中减去，然后将被寻址数据存储单元的值送入 TREG 寄存器，再将 TREG 寄存器的值与数据存储单元的值相乘(即求平方)，结果存至 PREG 寄存器。

●该指令受 PM 和 OVM 影响，并影响 C 和 OV 状态位。

例 4.91 SQRS DAT9 ;(DP=6：地址 0300h～037Fh；PM=0：乘积不移位)

数据存储器	C	执行指令前	数据存储器	C	执行指令后
309h		08h	309h		08h
TREG		1124h	TREG		08h
PREG		190h	PREG		40h
ACC	X	1450h	ACC	1	12C0h

□从累加器减

1) SUB dma[, shift]　　直接寻址

2) SUB ind,[,shift[, ARn]]　　间接寻址

3) SUB dma,16　　直接寻址并左移 16 位

4) SUB ind,16[, ARn]　　间接寻址并左移 16 位

5) SUB #k　　短立即寻址

6) SUB #lk[, shift]　　长立即寻址

●第 1 条和第 2 条指令将被寻址数据存储器单元的内容左移后，从累加器中减去。移位时，低位填零；高位在 SXM=1 时进行符号扩展，在 SXM=0 时填零。结果存储在累加器中。

●第 3 条和第 4 条指令将被寻址数据存储器单元的内容左移 16 位后，从累加器中减去。

●第 5 条指令为短立即数寻址，从累加器中减去 8 位正常数。这种情况下，不用指定移位值，减法也不受 SXM 的影响，且该指令不可被重复执行。

●第 6 条指令为长立即寻址，执行从累加器中减去 16 位正常数。

●通常，若减法结果产生一借位。则进位位被清 0(C=0)；若未产生借位，则进位位被置 1(C=1)；不过，若移位次数为 16，则产生借位时，进位位清零，减法不产生借位时，进位位不变。

●该指令受还受 OVM 影响，并影响 C 和 OV 状态位。

例 4.92 SUB DAT80 ;(DP=8：地址 0400h～047Fh；SXM=0：抑制符号扩展)

数据存储器	C	执行指令前	数据存储器	C	执行指令后
450h		11h	450h		11h
ACC	X	24h	ACC	1	13h

例 4.93 SUB *－,AR0 ;(左移 1 位,SXM＝0)

	C	执行指令前		C	执行指令后
ARP		7	ARP		0
AR7		301h	AR7		300h
数据存储器 301h		04h	数据存储器 301h		04h
ACC	X	09h	ACC	1	01h

例 4.94 SUB #8h ;(SXM＝1:符号扩展方式)

	C	执行指令前		C	执行指令后
ACC	X	07h	ACC	0	FFFFFFFFh

例 4.95 SUB #0FFFh, 4 ;(左移 4 位,SXM＝0)

	C	执行指令前		C	执行指令后
ACC	X	0FFFFh	ACC	1	0Fh

□带借位从累加器减

1) SUBB dma　　直接寻址

2) SUBB ind,[,ARn]　　间接寻址

●该指令从累加器中减去被寻址数据存储器单元的内容及进位位的逻辑反值,并抑制符号扩展。

●进位位按正常方式变化;若减法结果产生借位,则进位位被清除(C＝0);若未产生借位,则进位位被置 1(C＝1)。

●该指令不受 SXM 的影响。受 OVM 影响,并影响 C 和 OV 状态位。

例 4.96 SUBB DAT5 ;(DP＝8:地址 0400h～047Fh)

	C	执行指令前		C	执行指令后
数据存储器 405h		06h	数据存储器 405h		06h
ACC	0	06h	ACC	0	FFFFFFFFh

例 4.97 SUBB *

	C	执行指令前		C	执行指令后
ARP		6	ARP		6
AR6		301h	AR6		301h
数据存储器 301h		02h	数据存储器 301h		02h
ACC	1	04h	ACC	1	02h

□条件减

1) SUBC dma　　　　　直接寻址

2) SUBC ind,[,ARn]　间接寻址

●该指令只在被减数(ACC)和减数(指定的数据存储器地址的内容)为零或正数时才有效。具体步骤是:

若(ACC)≥0 且(指定的数据存储器地址)≥0,那么

(PC)+1 ⟶PC

(ACC)-[(指定的数据存储器地址)左移 15 次]⟶ALU

如果 ALU≥0,那么(ALU)×2+1 ⟶ACC;

如果 ALU<0,那么(ACC)×2 ⟶ACC。

● SUBC 指令的条件减法,可用于实现除法。用法如下:正的 16 位被除数置于累加器低位,累加器高位清零,正的 16 位除数存于数据存储单元,然后执行 SUBC 指令 16 次。当执行完最后一次 SUBC 指令后,所除得的商在累加器的低 16 位中,余数在累加器的高 16 位中。对于累加器值和/或数据存储单元值为负,则不能将 SUBC 指令用于除法。

●若 16 位被除数所含有效位少于 16 位,则被除数可左移后置于累加器中,左移位数由前面非有效 0 的个数决定。SUBC 指令执行的次数则为 16 位减去移位次数。

●该指令不受 SXM 和 OVM 的影响,运算结果会影响 C 和 OV。所以当执行该指令中有正溢或负溢时,累加器的内容不会以正负最大值填入。在执行该指令期间,进位位按正常方式变化;若减法结果产生借位,则进位位被清零(C=0);若未产生借位,则进位位被置 1(C=1)。

例 4.98　SUBC　DAT2　;(DP=6)

	C	执行指令前		C	执行指令后
数据存储器 302h		01h	数据存储器 302h		01h
ACC	X	04h	ACC	0	08h

例 4.99　RPT　#15

SUBC　*

	C	执行指令前		C	执行指令后
ARP		3	ARP		3
AR3		1000h	AR3		1000h
数据存储器 1000h		07h	数据存储器 1000h		07h
ACC	X	41h	ACC	1	20009h

□抑制符号扩展从累加器中减

1) SUBS dma　　　　　直接寻址

2) SUBS ind, [, ARn]　间接寻址

●该指令从累加器减去抑制符号扩展的指定数据存储单元的内容。无论 SXM 为何值,数据存储器单元内容均被作为 16 位无符号数对待,累加器内容则被视为有符号数。

● SUBS 与 SUB 指令在 SXM=0 且移位为 0 时产生的结果一致。

●该指令不受 SXM 影响,但受 OVM 的影响,运算结果会影响 C 和 OV。若减法产生借位,

进位位被清零(C=0);若未产生借位,则进位位被置 1(C=1)。

例 4.100 SUBS DAT2 ;(DP=6;SXM=1)

	C	执行指令前		C	执行指令后
数据存储器			数据存储器		
802h		0F003h	802h		0F003h
ACC	X	0F105h	ACC	1	102h

例 4.101 SUBS * ;(SXM=1)

	C	执行指令前		C	执行指令后
ARP		0	ARP		0
AR0		310h	AR0		310h
数据存储器			数据存储器		
310h		0F003h	310h		0F003h
ACC	X	0FFF105h	ACC	1	0FFF0102h

□从累加器减去按 T 寄存器规定进行移位的值

1) SUBT dma 直接寻址

2) SUBT ind,[, ARn] 间接寻址

●该指令将数据存储器的值左移后从累加器中减去,结果取代累加器原来的内容。左移位数由 TREG 寄存器 4 位最低有效位决定,可左移 0~15 位。

●数据存储器的值是否进行符号扩展由 SXM 状态位控制。

●该指令受 SXM 影响和 OVM 的影响,运算结果会影响 C 和 OV。若减法结果产生一进位时,则进位位被清零(C=0);若未产生借位,则进位位被置 1(C=1)。

例 4.102 SUBT DAT127 ;(DP=5:地址 0280h 至 02FFh)

	C	执行指令前		C	执行指令后
数据存储器			数据存储器		
02FFh		06h	02FFh		06h
TREG		08h	TREG		08h
ACC	X	0FDA5h	ACC	0	0F7A5h

例 4.103 SUBT *

	C	执行指令前		C	执行指令后
ARP		1	ARP		1
AR1		800h	AR1		800h
数据存储器			数据存储器		
800h		01h	800h		01h
TREG		08h	TREG		08h
ACC	X	0h	ACC	0	FFFFFF00h

4.5 逻辑运算指令

逻辑运算类指令共有 13 条。包括在累加器中进行的与、或、异或、求反、左右移位、清零等逻辑操作。

□累加器的逻辑“与”

1) AND dma　　　　直接寻址

2) AND ind,[,ARn]　　　　间接寻址

3) AND #lk[, shift]　　　　长立即数寻址

4) AND #lk,16　　　　长立即寻址数并左移 16 位

●第 1 条和第 2 条指令将累加器的低位字(0～15 位)和指定的数据存储器单元的内容作“逻辑与”运算,其结果放在累加器的低位字。累加器的高位字(16～31 位)则被清零。

●第 3 条和第 4 条指令为长立即数寻址,可进行 0～15 位或 16 位的左移。移位时,32 位中未被长立即操作数占用的低位和高位皆填零,移位后的值和累加器的内容作“逻辑与”运算,结果存在累加器中。

●该指令不受 SXM 的影响,也不影响其它状态位。

例 4.104　AND　16　　;(DP=4:地址 0200h～027Fh)

	执行指令前		执行指令后
数据存储器		数据存储器	
0210h	00FFh	0210h	00FFh
ACC	12345678h	ACC	00000078h

例 4.105　AND　#00FFh,4

	执行指令前		执行指令后
ACC	12345678h	ACC	00000670h

□累加器求反

CMPL

●该条指令将累加器内容取其“逻辑反”,结果存回累加器。

●该指令不受 SXM 的影响,也不影响其它状态位。

例 4.106　CMPL

		执行指令前			执行指令后
ACC	X (C)	0F7982513h	ACC	X (C)	0867DAECh

□累加器求负

NEG

●该指令将累加器中的内容乘以－1再放回到累加器。累加器前后的内容按二进制补码解释。

●当对8000 0000h进行NEG操作时，OV位被置1，并且若OVM＝1，则累加器内容由7FFF FFFFh取代；若OVM＝0，则结果仍为8000 0000h。

●若累加器内容不等于零，该指令将使进位位C被清为0；若累加器内容等于零，则进位位C被置为1。

●该指令受OVM的影响，结果影响C和OV。

例4.107 NEG ；(OVM＝X)将－3544转换为＋3544

	执行指令前			执行指令后	
ACC	X (C)	0FFFFF228h	ACC	0 (C)	0DD8h
	X (OV)			X (OV)	

□规格化累加器

NORM ind 间接寻址

●该指令对累加器中的定点数进行规格化，也就是将它分为尾数和指数两部分，指数放在当前AR，尾数放在累加器。具体做法是，如果累加器的第31位和第30位相同，即求“异或”为0，则累加器左移一位，并且当前AR按指定方式修改(缺省方式为当前AR的内容加1)。这种做法的理由是，如果累加器的第31位和第30位相同，表明前两位是符号的扩展，而不是有效数据；累加器左移一位相当于乘以2，此时若按缺省方式对当前AR的内容进行修改，即当前AR的内容加1，并且理解AR的指数是负指数，则当前AR与累加器联合表示的数与原来的相等。

●该指令的执行过程是：

(PC)＋1 ⟶PC

若(ACC)＝0，那么TC＝1。

若(ACC.31)XOR(ACC.30)≠0，那么TC＝1。

若(ACC.31)XOR(ACC.30)＝0，那么TC＝0，(ACC)×2 ⟶ACC，按指定方式修改当前AR。

●多次执行NORM指令可将累加器中的32位数完全规格化。当NORM指令和RPT指令一同使用时，若重复次数未到0但TC为1，此时的NORM操作自动转为空操作NOP，直到重复次数为0。

●正、负二进制补码均可用NORM规格化。

●对于NORM指令，辅助寄存器操作在流水线的第四阶段(执行阶段)执行。对于其余指令，辅助寄存器操作在流水线的第二阶段(译码阶段)执行。

●紧跟在NORM后的两条指令不能对辅助寄存器值进行修改。若用于NORM指令的辅助寄存器被紧跟在NORM后的下两条指令字所修改，那么在NORM指令对其进行修改前该辅助寄存器值已经被修改了。

●紧跟NORM后的两条指令不能对辅助寄存器指针(ARP)进行修改。若紧跟NORM后的两条指令中的任一条指明对ARP值进行修改，那么ARP值将在NORM指执行前已被修改。从而当NORM指令执行时，ARP将不能指向正确的辅助寄存器。

例 4.108 NORM *+

		执行指令前			执行指令后
ARP		2	ARP		2
AR2		00h	AR2		01h
ACC	C: X	FFFFF001h	ACC	C: X	FFFFE002h
	TC: X			TC: 0	

例 4.109 31 位规格化

```
        MAR   * ,AR1          ; AR1 用于存放指数
        LAR   AR1,#0h         ; 清指数计数器
LOOP    NORM  *+              ; 规格化一位
        BCND  LOOP,NTC        ; 若 TC=0,则未得到数值
```

例 4.110 15 位规格化

```
        MAR   * , AR1         ; AR1 用于存放指数
        LAR   AR1, #0Fh       ; 初始化指数计数器
        RPT   #14             ; 要求做 15 位规格化(产生 4 位指数和 16 位尾数)
        NORM  *-              ; 当发现第一个有效数值位时,NORM 自动停止
                              ; 移位,在余下的循环中执行空操作。
```

例 4.109 中的方法用来规格化 32 位数并产生 5 位指数值($2^5=32$)。例 4.110 中的方法用来规格化 16 位数并产生 4 位指数值($2^4=16$)。若事先已知道一个数仅需较少次数的规格化,则第二种的方法优于例 4.110 中的方法。这是因为例 4.109 仅运行到规格化完成为止,而例 4.110 一定要执行循环 15 次。特别指出,若所需移位数为 3 或更少,则例 4.109 的方法为好;但若需移动的位数达到或多于 6 位时,则例 4.110 的方法更好。

□累加器的逻辑"或"

1) OR dma　　　　　直接寻址
2) OR ind, [, ARn]　　间接寻址
3) OR #lk[, shift]　　长立即寻址
4) OR #lk, 16　　　　长立即寻址并左移 16 位

●第 1 条和第 2 条指令将累加器的低位字(0~15 位)和指定的数据存储器单元的内容作"逻辑或"运算,其结果放在累加器的低位字。累加器的高位字(16~31 位)不变。

●第 3 条和第 4 条指令为长立即数寻址,可进行 0~15 位或 16 位的左移。移位时,32 位中未被长立即操作数占用的低位和高位皆填零,移位后的值和累加器的内容作"逻辑或"运算。结果存在累加器中。

●该指令不受 SXM 的影响,运算结果不影响其它状态位。

例 4.111 OR DAT8 ;(DP=8)

数据存储器		执行指令前	数据存储器		执行指令后
408h		0F000h	408h		0F000h
ACC	C: X	100002h	ACC	C: X	10F002h

例 4.112 OR #08111h,8

	C	执行指令前		C	执行指令后
ACC	X	0FF0000h	ACC	X	0FF1100h

□累加器逻辑循环左移

ROL

● ROL 指令将累加器内容左移一位。进位位被移入最低有效位，最高有效位被移入进位位。

●该指令不受 SXM 影响。除了进位位 C 外，运算结果不影响其它状态位。

例 4.113 ROL

	C	执行指令前		C	执行指令后
ACC	0	B0001234h	ACC	1	60002468h

□累加器逻辑循环右移

ROR

● ROR 指令将累加器内容右移一位。进位位被移入最高有效位，而累加器的最低有效位被移入进位位。

●该指令不受 SXM 的影响。除了进位位 C 外，运算结果不影响其它状态位。

例 4.114 ROR

	C	执行指令前		C	执行指令后
ACC	0	B0001235h	ACC	1	5800091Ah

□移位并存储累加器高位

1) SACH dma[,shift2]　　　　　　直接寻址

2) SACH ind,[,shift2][,ARn]　　　间接寻址

● SACH 指令将整个累加器复制到输出移位器中，然后全部 32 位左移 0～7 位(shift2)，再将移位后数值的高 16 位复制到指定的数据存储器。在移位时，低位填零，高位丢失，累加器内容不变。

●该指令不受 SXM 的影响。运算结果不影响其它状态位。

例 4.115 SACH DAT10,1;　(DP=4:地址 0200h 至 027Fh;左移 1 位)

	C	执行指令前		C	执行指令后
ACC	X	4208001h	ACC	X	4208001h
数据存储器			数据存储器		
20Ah		0h	20Ah		0841h

例 4.116 SACH *+, 0, AR2   ;不移位

	C	执行指令前		C	执行指令后
ARP		1	ARP		2
AR1		300h	AR1		301h
ACC	X	4208001h	ACC	X	4208001h
数据存储器 300h		0h	数据存储器 300h		0420h

□移位并存储累加器低位

1) SACL dma[, shift2]   直接寻址

2) SACL ind, [, shift2][, ARn]   间接寻址

● SACH 指令将整个累加器复制到输出移位器中，然后全部 32 位左移 0～7 位(shift2)，再将移位后数值的低 16 位复制到数据存储器。在移位时，低位填零，高位丢失，累加器内容不变。

●该指令不受 SXM 的影响。运算结果不影响其它状态位。

例 4.117 SACL DAT11,1   (DP=4;地址 0200h～027Fh;左移 1 位)

	C	执行指令前		C	执行指令后
ACC	X	7C638421h	ACC	X	7C638421h
数据存储器 20Bh		05h	数据存储器 20Bh		0842h

例 4.118 SACL *, 0, AR7   ;不移位

	C	执行指令前		C	执行指令后
ARP		6	ARP		7
AR6		300h	AR6		300h
ACC	X	00FF8421h	ACC	X	00FF8421h
数据存储器 300h		05h	数据存储器 300h		8421h

□累加器算术左移

SFL

● SFL 指令将所有累加器内容左移一位。最低有效位填零，最高有效位填入进位位(C)。

●注意 SFL 指令与 ROL 指令的区别，前者的最低有效位填零，后者最低有效位填进位位 C。

● SFL 不受 SXM 影响。除了进位位 C 外，运算结果不影响其它状态位。

例 4.119 SFL

	C	执行指令前		C	执行指令后
ACC	X	B0001234h	ACC	1	60002468h

□累加器算术右移

SFR

●若 SXM=1，该指令产生算术右移。符号位不改变，并且被复制到第 30 位。第 0 位移入

进位位。

●若 SXM＝0，该指令产生逻辑右移。累加器中所有内容右移一位，最低有效位移入进位位，最高有效位填 0。

●注意 SFR 指令与 ROR 指令的区别，前者的最高有效位填零(SXM＝0)或不变(SXM＝1)，后者最高有效位填进位位 C。

●SFR 受 SXM 影响。除了进位位 C 外，运算结果不影响其它状态位。

例 4.120 SFR ；(SXM＝0：无符号扩展)

	C	执行指令前		C	执行指令后
ACC	X	B0001234h	ACC	0	5800091Ah

例 4.121 SFR ；(SXM＝1：符号扩展)

	C	执行指令前		C	执行指令后
ACC	X	B0001234h	ACC	0	D800091Ah

□累加器的逻辑“异或”

1) XOR dma　　直接寻址

2) XOR ind，[，ARn]　　间接寻址

3) XOR #lk [，shift]　　长立即寻址

4) XOR #lk，16　　长立即寻址并左移 16 位

●第 1 条和第 2 条指令对累加器的低位字和被寻址的数据存储器单元的内容作逻辑“异或”，结果取代累加器的低位字，其累加器的高位字不受影响。

●第 3 条和第 4 条指令为长立即数寻址，长立即数可移位 0～15 位或 16 位。移位时，32 位中未被长立即操作数占用的低位和高位皆填零。移位后的值和累加器的内容作“逻辑或”运算。结果存在累加器中。

●该指令不影响状态位。

例 4.122 XOR DAT127 ；(DP＝511：地址 FF80h～FFFFh)

	C	执行指令前		C	执行指令后
数据存储器			数据存储器		
0FFFFh		0F0F0h	0FFFFh		0F0F0h
ACC	X	12345678h	ACC	X	1234A688h

例 4.123 XOR ＊＋，AR0

	C	执行指令前		C	执行指令后
ARP		7	ARP		0
AR7		300h	AR7		301h
数据存储器			数据存储器		
300h		0FFFFh	300h		0FFFFh
ACC	X	1234F0F0h	ACC	X	12340F0Fh

例 4.124 XOR #0F0F0h, 4 ;(先将数值左移 4 位)

	执行指令前		执行指令后	
	C		C	
ACC	X	11111010h	X	111E1F10h

□清累加器低位且舍入装载累加器高位

1) ZALR dma 直接寻址

2) ZALR ind,[,ARn] 间接寻址

●该指令把指定数据存储单元的内容装入累加器高 16 位。累加器低 15 位(位 14～位 0)清零,累加器的第 15 位置 1,即将 8000h 放入 ACC 的低 16 位。

例 4.125 ZALR DAT3 ;(DP=32;地址 1000h～107Fh)

	执行指令前		执行指令后	
数据存储器				
1003h		3F01h		3F01h
	C		C	
ACC	X	77FFFFh	X	3F018000h

例 4.126 ZALR *-,AR4

	执行指令前		执行指令后	
ARP		7		4
AR7		0FF00h		0FEFFh
数据存储器				
0FF00h		0E0E0h		0E0E0h
	C		C	
ACC	X	107777h	X	0E0E08000h

4.6 分支转移指令

程序控制转移指令共有 21 条,包括有条件与无条件的转移、调用以及各种中断及返回操作的转移指令。也包括了对于一些状态位的测试、清除、置位等控制指令。

□无条件分支转移

B pma[, ind[, ARn]] 间接寻址

●该指令先将指定的程序存储器地址(pma)装入 PC。按规定方式修改当前辅助寄存器和 ARP 的内容,并将控制转移到 PC 所指的程序存储器地址。

● pma 既可以是符号地址,也可以是数字地址。

●该指令不影响任何状态位。

例 4.127 B 191,*+,AR1

191 被装入程序计数器,并且程序从这个单元继续运行。当前辅助寄存器增 1,ARP 被置为指向辅助寄存器 1(AR1)。

□按累加器内容转移

BACC

●执行该指令后，累加器低位字被装入 PC，程序从累加器低位字所指的 16 位地址继续运行。

●该指令提供了按照运算结果进行转移的可能。

●该指令不影响任何状态位。

例 4.128 BACC ;(ACC=191)

191 被装入程序计数器，并且程序从这个单元继续运行。

□辅助寄存器不等于零转移

BANZ pma [, ind[, ARn]] 间接寻址

●若当前辅助寄存器内容不为零，则控制转移至指定的程序存储器地址(pma)；若当前辅助寄存器内容为零，则程序顺序执行下一条指令。

●按规定方式修改当前辅助寄存器和 ARP 的内容，对当前 AR 缺省的修改方式是减 1。

●通过这条指令和一个辅助寄存器可实现循环操作。即在进入循环前，将辅助寄存器的内容初始化为 N－1，从而可执行 N 次循环。

● Pma 既可以是符号地址，也可以是数字地址。

●该指令不影响任何状态位。

例 4.129 BANZ PGM0 ;(PGM0 是程序地址 0 的标号)

	执行指令前		执行指令后
ARP	0	ARP	0
AR0	5h	AR0	4h

由于 AR0 内容不等于零，则程序转移至被装入程序计数器(PC)中的程序地址 0，然后程序从 0 单元继续执行。缺省的辅助寄存器的操作是将当前辅助寄存器内容减 1，从而在执行结束时，AR0 中的值是 4h。

另一种情况：

	执行指令前		执行指令后
ARP	0	ARP	0
AR0	0h	AR0	FFFFh

由于 AR0 内容为 0，从而不执行转移；而是 PC 增 2，程序从紧接着 BANZ 指令后的那条指令继续执行。由于缺省减 1，从而 AR0 被减 1，变为－1。

例 4.130

```
        MAR   *, AR0            ; 设置 ARP 指向 AR0
        LAR   AR1, #3           ; AR1 中装入 3
        ALR   AR0, #60h         ; AR0 中装入 60h
PGM191  ADD   *+, AR1           ; 若 AR1≠0 则循环
        BANZ  PGM191, AR0       ; 将 AR0 所指的数加到累加器，并将
                                ; AR0 的值增 1
```

□条件转移

BCND pma, cond1[,cond2][,…]

●操作数 cond 可使用如下条件

cond	说　明	cond	说　明	cond	说　明
EQ	ACC=0	GEQ	ACC≥0	$\overline{\text{BIO}}$	引脚 BIO 为低
NEQ	ACC≠0	NC	C=0	NTC	TC=0
LT	ACC<0	C	C=1	TC	TC=1
LEQ	ACC≤0	NOV	OV=0	UNC	无条件
GT	ACC>0	OV	OV=1		

●若指定的条件都满足，将指定的程序存储器地址 pma 装入 PC，程序从指定的地址继续执行。若条件不满足，则执行下一条指令。

●设置条件时，要注意有些条件的组合是无意义的。例如同时测试 LT 和 LG 会发生冲突，也不能同时测试 BIO 的非和测试 TC。

●该指令不影响任何状态位。

例 4.131　BCND　PGM191，LEQ，C

若累加器内容小于或等于零且进位位被置 1，则程序地址 0BFh(191)被装入程序计数器，然后程序从该单元继续执行。若条件不成立，则继续执行 PC+2 单元的指令。

□调用累加器低 16 位指定地址处的子程序

CALA

●执行该指令时，当前程序计数器(PC)增 1，并被推入栈顶 TOS，即保护返回地址。然后累加器低 16 位内容送入 PC，接着从该 16 位地址处继续执行。

●该指令提供了按照运算结果分别调用不同子程序的可能。

●该指令不影响任何状态位。

例 4.132　CALA

	执行指令前		执行指令后
PC	25h	PC	83h
ACC	83h	ACC	83h
TOS	100h	TOS	26h

□无条件调用

CALL　pma[,ind[，ARn]]　　　间接寻址

●当前程序计数器 PC 增 2，并被推入栈顶 TOS，即保护返回地址。

●然后，程序存储地址 pma 被送入 PC，接着从该地址处继续执行。

●程序存储地址 pma 既可以是符号地址，也可以是数字地址。

●按照指定要求的方式修改当前辅助寄存器和 ARP 内容。

●该指令不影响任何状态位。

例 4.133　CALL　191，*+，AR0

	执行指令前		执行指令后
ARP	1	ARP	0
AR1	05h	AR1	06h
PC	30h	PC	0BFh
TOS	100h	TOS	32h

程序地址 0BFh(191)被装入程序计数器，然后程序从此单元继续执行。

□条件调用

CC pma,cond1[, cond2][,…]

●操作数 cond 使用的条件与指令 BCND 的一样。

●若指定的条件都满足，则将当前程序计数器 PC 增 2，并被推入栈顶 TOS，即保护返回地址。然后指定的程序存储器地址被装入 PC，程序从指定的地址继续执行。若条件不满足，则执行下一条指令。

●设置条件时，要注意有些条件的组合是无意义的。例如同时测试 LT 和 GT 会发生冲突，也不能同时测试$\overline{BIO}$和 TC。

●当所有条件皆成立时，CC 指令与 CALL 指令操作相同。

●该指令不影响任何状态位。

例 4.134 CC PGM191, LEQ, C

若累加器内容小于或等于零且进位位被置 1，则 0BFh(191)被装入程序计数器，然后程序从该单元继续执行。若条件不成立，则继续执行 CC 指令后面的那条指令。

□中断

INTR k

● INTR 指令是一条软中断指令，操作数 k 取 0 至 31，用以指示将要转移至的中断向量单元，参见 3.7。

●执行时，先将 PC 值+1 后推入栈顶，即保护返回地址。然后将 INTR k 所对应的中断向量装入 PC，从此开始执行中断服务子程序。

●软中断 INTR 是一个非屏蔽中断，它的执行不受 INTM 位的影响。但执行 INTR 指令时将使 INTM 置 1，以屏蔽掉所有可屏蔽中断。除了 INTM 外，该指令的执行不影响其它状态位。

例 4.135 INTR 3 ；PC+1 被推入堆栈，然后控制转移至程序存储器单元
；6h(INTR3 的中断向量)。

□不可屏蔽中断

NMI

● NMI 是非屏蔽软中断，其作用与硬件的引脚非屏蔽中断 NMI 的效果相同。

●执行该指令时，当前 PC 增 1 后被推入堆栈，即保护返回地址；然后将 NMI 对应的中断向量 0024H 装入 PC，从此开始执行 NMI 的中断服务子程序。

●与软中断 INTR 一样，不受 INTM 位的影响。但执行 NMI 指令时将使 INTM 置 1，以屏蔽掉所有可屏蔽中断。除了 INTM 外，该指令的执行不影响其它状态位。

例 4.136 NMI ；PC+1 被推入栈顶，然后控制转移到程序存储器单元
；24h(NMI 的中断向量)。

□从子程序返回

RET

●该指令将栈顶寄存器内容弹出到程序计数器 PC，其余堆栈值依次向上移一级。被中断处的程序位置。

●该指令的执行不影响其它状态位。

例 4.137 RET

	执行指令前		执行指令后
PC	96h	PC	37h
Stack	37h	Stack	45h
	45h		75h
	75h		21h
	21h		3Fh
	3Fh		45h
	45h		6Eh
	6Eh		6Eh
	6Eh		6Eh

□条件返回

RETC cond1[,cond2][,…]

●操作数 cond 使用的条件与指令 BCND 的一样。

●当列出的条件都满足时，将栈顶寄存器内容弹出到程序计数器 PC，其余堆栈值依次向上移一级。否则，不返回。

●但注意有些条件的组合是无意义的。例如同时测试 LT 和 LG 会发生冲突，也不能同时测试$\overline{\text{BIO}}$和测试 TC。

●该指令的执行不影响其它状态位。

例 4.138 RETC GEQ, NOV ；当累加器内容为正或为零，且 OV(溢出)
；位也为零时返回。

□软件陷井中断

TRAP

●TRAP 指令也是一个软中断。执行这条指令时，先将程序计数器 PC 的内容加 1 推入堆栈，即保护返回地址；然后将 TRAP 的中断向量 0022H 装入 PC，开始执行 TRAP 的中断服务子程序。

●与软中断 INTR 一样，不受 INTM 位的影响。执行 TRAP 指令时将不影响 INTM 和其它状态位。

例 4.139 TRAP ；PC+1 被推入堆栈，控制被转移至程序存储器单元 22h(TRAP
；的中断向量)。

□位测试

1) BIT dma , bit code 直接寻址

2) BIT ind, bit code[,ARn] 间接寻址

●BIT 指令将数据存储器数值的指定位复制到状态寄存器 ST1 的 TC 位。

●注意，BITT、CMPR、LST #1 和 NORM 指令也影响状态寄存器 ST1 的 TC 位。

●指令中指定的位码值与数据存储器值的对应位置如下所示。

位码值	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
数值位	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	MSB						数据存储器									LSB

例 4.140 BIT 0h,15 ;(DP=6)测试 300h 处数据的 LSB

	执行指令前		执行指令后
数据存储器		数据存储器	
300h	4DC8h	300h	4DC8h
TC	0	TC	0

例 4.141 BIT * , 0, AR1 ; 测试 310h 处数据的最高有效位,设置 ARP=1

	执行指令前		执行指令后
ARP	0	ARP	1
AR0	310h	AR0	310h
数据存储器		数据存储器	
310h	8000h	310h	8000h
TC	0	TC	1

□测试 T 寄存器规定的位

1) BITT dma 直接寻址

2) BITT ind[,ARn] 间接寻址

● BITT 指令将数据存储器数值的指定位复制到状态寄存器 ST1 的 TC 位。

●注意,BITT、CMPR、LST#1 和 NORM 指令也影响状态寄存器 ST1 的 TC 位。

●指定位的位置由 T 寄存器 4 位最低有效位所包含的位码值给出,如下所示:

位码值	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
数值位	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	MSB						数据存储器									LSB

例 4.142 BITT 00h ;(DP=6)测试 300h 处数据的第 14 位

	执行指令前		执行指令后
数据存储器		数据存储器	
300h	4DC8h	300h	4DC8h
TREG	1h	TREG	1h
TC	0	TC	1

例 4.143　BITT　*　　；测试 310h 处数据的第 1 位

	执行指令前		执行指令后
ARP	1	ARP	1
AR1	310h	AR1	310h
数据存储器		数据存储器	
310h	8000h	310h	8000h
TREG	0Eh	TREG	0Eh
TC	0	TC	0

□清除控制位

CLRC control bit

●操作数 control bit 为下述控制位之一：

控制位	说　明	控制位	说　明
C	状态寄存器 ST1 的进位位	SXM	状态寄存器 ST1 的符号扩展方式位
CNF	状态寄存器 ST1 的 RAM 配置控制位	TC	状态寄存器 ST1 的测试/控制标志位
INTM	状态寄存器 ST0 的中断方式位	XF	状态寄存器 ST1 的 XF 引脚状态位
OVM	状态寄存器 ST0 的溢出方式位		

●该指令把指定的控制位清零。

●注意、LST 指令也可装入 ST0 和 ST1。

●关于各控制位的信息请参阅 2.4。

例 4.144　CLRC　TC　　；TC 是 ST1 的第 11 位

	执行指令前		执行指令后
ST1	x9xxh	ST1	x1xxh

□比较当前辅助寄存器和 AR0

CMPR　CM

●操作数 CM 取值为 0 至 3。

● CMPR 指令按 CM 值实现以下比较：

若 CM＝00，则测试（当前 AR）＝（AR0）否？

若 CM＝01，则测试（当前 AR）＜（AR0）否？

若 CM＝10，则测试（当前 AR）＞（AR0）否？

若 CM＝11，则测试（当前 AR）≠（AR0）否？

若测试条件为真，则 TC 为被置 1；若条件为假，则 TC 位被清零。

●注意，比较时辅助寄存器内容被作为无符号整数处理。

●该指令不受 SXM 的影响，除了 TC 外，也不影响其它状态位。

例 4.145　CMPR　2　　；（当前 AR）＞（AR0）否

	执行指令前		执行指令后
ARP	4	ARP	4
AR0	0FFFFh	AR0	0FFFFh
AR4	7FFFh	AR4	7FFFh
TC	1	TC	0

□空闲直至中断发生

IDLE

●INDE 指令迫使正在执行的程序停止，直至 CPU 接收到一个未被屏蔽的硬件中断请求(外部或内部)、NMI 或复位信号，参见 2.7。

●执行 IDLE 指令使得机器进入省电方式。进入省电方式前，PC 增 1；在等待期间，PC 不增加。片内外设保持激活状态，从而由它们引发的中断可以唤醒处理器。

●INTM 为状态寄存器 ST0 的中断方式位，通常，若 INTM 被置 1，则可屏蔽的中断将被禁止。但是，此时未被屏蔽的中断可以使 CPU 退出等待状态，随后 CPU 的动作取决于 INTM；若 INTM 为 0，则程序转移至相应的中断服务子程序；若 INTM 为 1，则程序从紧接IDLE 后面的那条指令继续执行。

●NMI 和复位是不可屏蔽的。从而，若由 NMI 或复位引起 CPU 退出等待状态，则无论 INTM 为何值，都将执行相应的中断服务子程序。

例 4.146 IDLE ；处理器等待直至有硬件复位、硬件 NMI 或未被屏蔽的中
；断发生

□空操作

NOP

●该指令不执行任何操作。仅对 PC 有影响。

●NOP 指令适用于建立流水线和执行延迟功能。

例 4.147 NOP ；不执行任何操作。

□重复执行下条指令

1) RPT dma 直接寻址

2) RPT ind[, ARn] 间接寻址

3) RPT #k 短立即寻址

●第 1 条和第 2 条指令将被寻址数据存储器单元中的值送人重复计数器(RPTC)。

●第 3 条指令是短立即数寻址，8 位立即数送人 RPTC。

●紧接 RPT 后的那条指令被重复执行 n 次，n 为 RPTC 初始值+1。

●由于转换时不能保存 PRTC 的值，所以重复循环被认为是多周期指令，它不能被中断。器件复位时，RPTC 被清零。

●RPT 很适用于块移动、乘加和规格化。而该指令自身不可重复。

例 4.148 RPT DAT127 ；(DP=31，地址 0F80h 至 0FFFh)
；重复下条指令 13 次

数据存储器	执行指令前	数据存储器	执行指令后
0FFFh	0Ch	0FFFh	0Ch
RPTC	0h	RPTC	0Ch

例 4.149 RPT * ，AR1 ；重复下条指令 4096 次

	执行指令前		执行指令后
ARP	0	ARP	1
AR0	300h	AR0	300h
数据存储器		数据存储器	
300h	0FFFh	300h	0FFFh
RPTC	0h	RPTC	0FFFh

例 4.150　RPT　#1　；重复下条指令 2 次

	执行指令前		执行指令后
RPTC	0h	RPTC	1h

□设置控制位

SETC control bit

●操作数 control bit 使用的控制位与 CLRC 指令一样。

●该指令将指定的控制位置 1。

●注意,LST 指令也可装入 ST0 和 ST1。

●关于各控制位的信息请参阅 2.4。

例 4.151　SETC　TC　;TC 是 ST1 的第 11 位

	执行指令前		执行指令后
ST1	x1xxh	ST1	x9xxh

□设置 P 寄存器输出移位方式

SPM　constant

●操作数 constant 取值为 0 至 3,决定乘积移位的方式。

●指令字的 2 位最低有效位被复制到状态寄存器 ST1 的 PM。

●PM 状态位控制了 PREG 寄存器的输出移位方式。该移位器可将 PREG 寄存器的输出左移 1 位、4 位或右移 6 位,参见 2.2。

●当指令访问 PREG 寄存器时,该寄存器的值先被送往移位器,并在移位器中按指定的方式移位。

●左移可使小数运算的乘积对齐,右移 6 位方式允许连续执行 128 次乘/累加运算而不会发生溢出。

●PM 也可由 LST #1 指令装入。

例 4.152　SPM　3　；选择乘积寄存器移位方式 3(PM=11),使以后所有从
；乘积寄存器(PREG)送出的数都右移 6 位。

第5章 应用实例

DSP控制器是针对工业控制而开发的一款高性能单片机。在工业生产现场中，无论是各种不同类型的机床及生产线，还是其它的生产设备，都离不开电动机作为动力部件，因此电动机的调速控制是工业控制系统中非常重要的一个内容。电动机的调速控制最典型的是交流异步电动机的调速控制。目前，从数百瓦级的家用电器直到数千千瓦级的调速传动装置，大部分都采用交流异步电动机调速传动方式来实现。交流调速传动控制得到如此迅速发展，主要是电力电子器件的制造技术、基于电力电子电路的电力变换技术、PWM技术以及以单片机为基础的全数字化控制技术等关键技术得到突破性进展。DSP控制器由于内嵌PWM电路、A/D转换电路以及其它相关电路，可以很容易实现交流异步电动机的全数字化控制系统。

频谱分析是信号处理的一个重要内容，快速傅里叶变换（FFT）是其有效的分析工具。快速傅里叶变换从算法上比普通的离散傅里叶变换（DFT）要快许多倍，但是要做到实时分析，其运算量普通单片机还是难以承受。DSP控制器能在一个指令周期内完成一次乘法和一次加法，而且提供专用于FFT的反序寻址方式，从而容易地实现实时的FFT分析运算。

5.1 基于空间矢量的通用变频器

变频器分为交—交和交—直—交两种形式。交—交变频器可将工频交流直接变换成频率、电压均可控制的交流；而交—直—交变频器则是先把工频交流通过整流器变成直流，然后再把直流变换成频率、电压均可控制的交流。通用变频器采用交—直—交这种形式，图5.1是一个典型的通用变频器电路，其中整流器、电容和逆变桥为主回路，检测、保护与DSP控制器等构成控制回路。

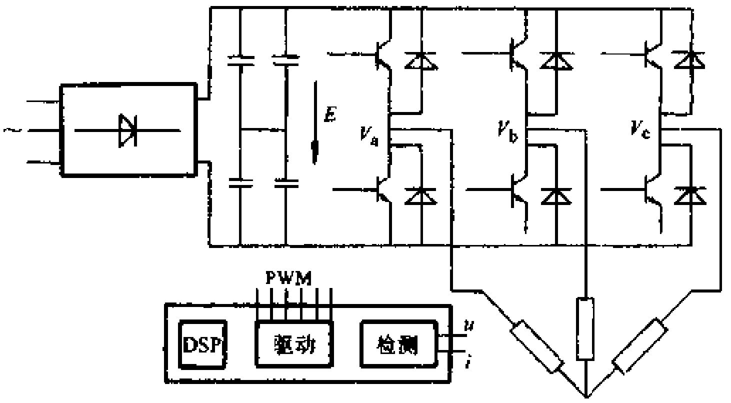


图5.1 一个典型的通用变频器电路

整流器的作用是将三相（或单相）交流电整流成直流电。电容是储能元件，用以缓冲负载的无功功率，由于电容的作用直流侧的电压将比较平稳。逆变桥的作用是通过六个开关器件有规律地通断将直流电变成频率和电压均可调的交流电。电压、电流的检测是为过电压、欠电压、过电流、短路保护服务。

设直流侧的电动势为E，现要求逆变桥的输出电压V_a、V_b、V_c为，

$$
\begin{aligned}
V_a &= V_m\sin\omega t \\
V_b &= V_m\sin(\omega t + 120^\circ) \\
V_c &= V_m\sin(\omega t - 120^\circ)
\end{aligned}
\tag{5.1}
$$

其中V_m、ω可调。问六个开关器件应以什么样的规律来通断，才可使得逆变桥输出电压满足

上式的要求？这个问题是通用变频器的关键问题。解决这个问题有许多种方法，如三角波调制的 PWM 方法、空间矢量方法等。空间矢量方法较之其它 PWM 方法能提高电压的利用率，减少谐波的影响，近几年在通用变频器中得到广泛应用。由于 DSP 控制器内嵌了空间矢量状态机，因而可以很容易地以空间矢量方法来通断六个开关器件，使得逆变桥输出电压满足前面的要求。下面给出详细的推导。

参见 3.1.2 中空间矢量状态机的有关内容，将逆变桥三相输出电压转化为二相坐标系的 V_α 和 V_β，即

$$\begin{aligned}\begin{bmatrix}V_\alpha\\V_\beta\end{bmatrix}&=\sqrt{2/3}\begin{bmatrix}1 & -1/2 & -1/2\\0 & \sqrt{3}/2 & -\sqrt{3}/2\end{bmatrix}\begin{bmatrix}V_a\\V_b\\V_c\end{bmatrix}\\&=\sqrt{2/3}\begin{bmatrix}1 & -1/2 & -1/2\\0 & \sqrt{3}/2 & -\sqrt{3}/2\end{bmatrix}\begin{bmatrix}V_m\sin\omega t\\V_m\sin(\omega t+120^\circ)\\V_m\sin(\omega t-120^\circ)\end{bmatrix}\\&=\sqrt{3/2}\begin{bmatrix}V_m\sin\omega t\\V_m\cos\omega t\end{bmatrix}\end{aligned}\tag{5.2}$$

从图 3.17 和表 3.10 知，每个有效的空间矢量 $\boldsymbol{U}_x$ 在二相坐标系的分量 $\boldsymbol{U}_{x\alpha}$和 $\boldsymbol{U}_{x\beta}$为：

$$\boldsymbol{U}_{x\alpha}=\sqrt{2/3}E\cos x\qquad \boldsymbol{U}_{x\beta}=\sqrt{2/3}E\sin x\tag{5.3}$$

并且，任意时刻的 V_α 和 V_β 可以由 $\boldsymbol{U}_x$ 和 $\boldsymbol{U}_{x\pm60}$来合成，即

$$V_\alpha=k_xU_{x\alpha}+k_{x\pm60}U_{(x\pm60)\alpha}=k_x\sqrt{2/3}E\cos x+k_{x\pm60}\sqrt{2/3}E\cos(x\pm60^\circ)$$

$$V_\beta=k_xU_{x\beta}+k_{x\pm60}U_{(x\pm60)\beta}=k_x\sqrt{2/3}E\sin x+k_{x\pm60}\sqrt{2/3}E\sin(x\pm60^\circ)$$

$$\begin{bmatrix}V_\alpha\\V_\beta\end{bmatrix}=\sqrt{2/3}E\begin{bmatrix}\cos x & \cos(x\pm60^\circ)\\\sin x & \sin(x\pm60^\circ)\end{bmatrix}\begin{bmatrix}k_x\\k_{x\pm60}\end{bmatrix}\tag{5.4}$$

设 $V_m=k_vE$,并考虑式(5.3)则有

$$\begin{aligned}\begin{bmatrix}k_x\\k_{x\pm60}\end{bmatrix}&=\pm\sqrt{2}/E\begin{bmatrix}\sin(x\pm60^\circ) & -\cos(x\pm60^\circ)\\-\sin x & \cos x\end{bmatrix}\begin{bmatrix}V_\alpha\\V_\beta\end{bmatrix}\\&=\pm\sqrt{3}k_v\begin{bmatrix}\sin(x\pm60^\circ) & -\cos(x\pm60^\circ)\\-\sin x & \cos x\end{bmatrix}\begin{bmatrix}\sin\omega t\\\cos\omega t\end{bmatrix}\\&=\pm\sqrt{3}k_v\begin{bmatrix}-\cos(\omega t+x\pm60^\circ)\\\cos(\omega t+x)\end{bmatrix}\end{aligned}\tag{5.5}$$

为了编程更为直观，令 $\omega\tau=90^\circ-\omega t$,则上式化为：

$$\begin{aligned}\begin{bmatrix}k_x\\k_{x\pm60}\end{bmatrix}&=\pm\sqrt{3}k_v\begin{bmatrix}\sin(x\pm60^\circ-\omega\tau)\\\sin(\omega\tau-x)\end{bmatrix}\\&=\begin{cases}\sqrt{3}k_v\begin{bmatrix}\sin(x+60^\circ-\omega\tau)\\\sin(\omega\tau-x)\end{bmatrix} & 0^\circ\leqslant\omega\tau-x\leqslant60^\circ\quad\text{如果是逆时针转}\\\sqrt{3}k_v\begin{bmatrix}\sin(\omega\tau-(x-60^\circ))\\\sin(x-\omega\tau)\end{bmatrix} & 0^\circ\leqslant x-\omega\tau\leqslant60^\circ\quad\text{如果是顺时针转}\end{cases}\end{aligned}\tag{5.6}$$

从式(3.8)知，用 DSP 控制器的空间矢量状态机来实现变频的要求，需要求出在一个调制周期 T_p 内两个有效空间矢量 U_x、$U_{x\pm60}$的持续时间 T_x、$T_{x\pm60}$。根据前面的推导不难得知，这二个时间为：

$$\begin{bmatrix} T_x \\ T_{x\pm60} \end{bmatrix} = T_p \begin{bmatrix} k_x \\ k_{x\pm60} \end{bmatrix} \tag{5.7}$$

式(5.6)和式(5.7)是空间矢量状态机实现变频的关键公式。可以看出，改变 k_v 可以调节电压；改变 ω 可以调节频率；并且只要预先建好正弦表就可容易地编程实现。下面给出空间矢量状态机的初始化和中断服务流程。初始化流程设置比较控制寄存器 COMCON、全比较动作控制寄存器 ACTR、死区控制寄存器 DBTCON、通用定时器 1 的控制寄存器 T1CON，并根据调制周期(频率)T_p 设置通用定时器 1 的定时周期寄存器 T1PR。中断服务流程根据空间矢量的旋转方向(SVDir)、给定的频率(ω)和电压(k_v)，计算出 T_x 和 $T_{x\pm60}$ 的值并设置到全比较寄存器 CMPR1 和 CMPR2，为下个调制周期做准备。图 5.2 和图 5.3 分别是初始化和中断服务流程。

↓

初始化比较控制寄存器 COMCON

1. 设引脚输出为 PWM 模式 SELCMP3 : 2 : 1=111
2. 使引脚输出使能 FCOMPOE=1
3. 设全比较动作控制寄存器重载条件为下溢 ACTRLD1 : 0=00
4. 设全比较寄存器重载条件为下溢 CLD1 : 0=00
5. 开启空间矢量方式 SVEnable=1
6. 开启全比较功能 Cenable=1

↓

初始化全比较动作控制寄存器 ACTR

1. 设置引脚输出极性为"高有效"CMPxACT1 : 0=10
2. 设初始空间矢量为 D2 : 1 : 0=001

↓

初始化死区控制寄存器 DBTCON

1. 启用死区电路 EDBT3 : 2 : 1=111
2. 设置分频系数 DBTPS1 : 0
3. 设置死区时间 DBT7 : 0

↓

初始化通用定时器 1 的控制寄存器 T1CON

1. 设置计数模式为连续增/减 Tmode2 : 1 : 0=101
2. 设置分频系数 TPS2 : 1 : 0
3. 选择内部时钟 TCLKS1 : 0=00

↓

根据调制周期(频率)T_p、CPU 时钟周期，设置通用定时器 1 的定时周期寄存器 T1PR。

↓

启动通用定时器 1，开启下溢中断。

↓

图 5.2　空间矢量的初始化流程

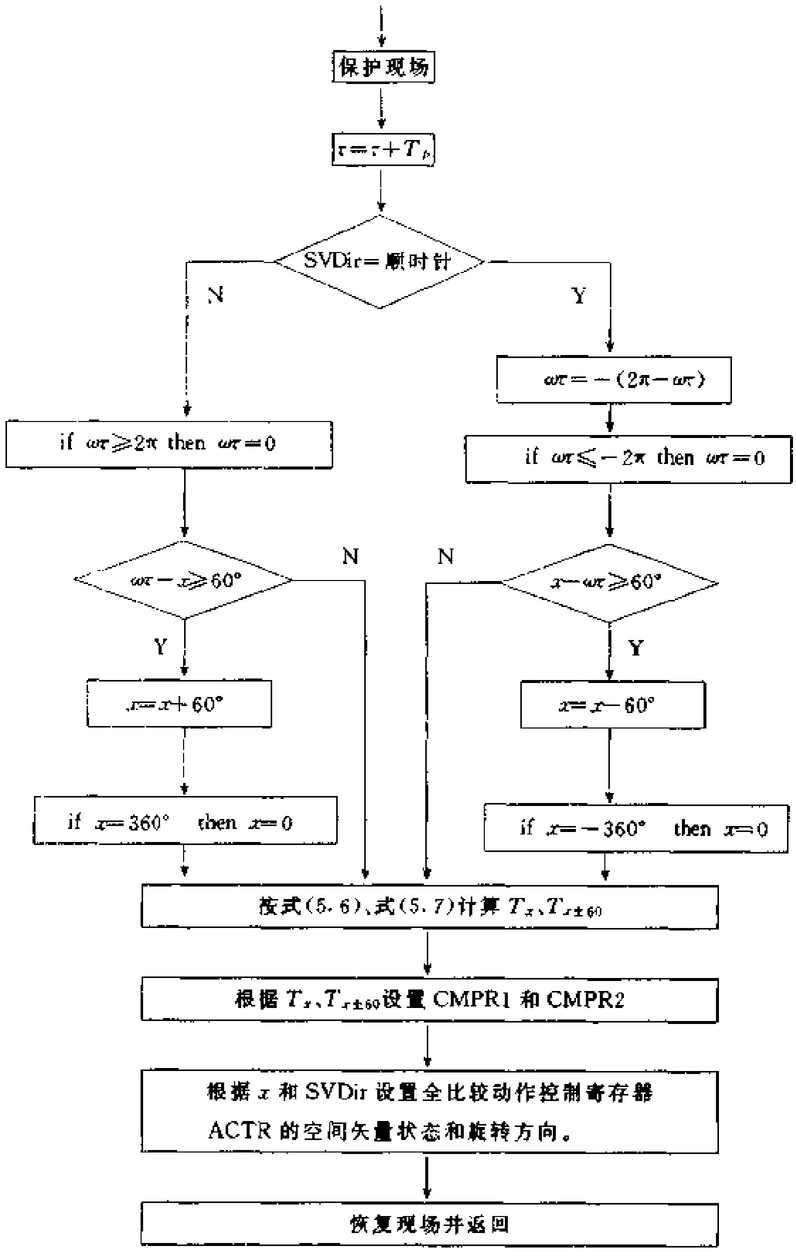


图 5.3 空间矢量中断服务流程

5.2 快速傅里叶变换(FFT)

傅里叶变换是一种将时域信号变换为频域信号的变换形式。在频域分析中，信号的频率及对应的幅值、相位(统称为频谱)是信号分析的重要内容，它反映了系统性能的好坏。下面先简要介绍快速傅里叶变换的基本原理，然后介绍如何在 DSP 控制器中实现快速傅里叶变换。

5.2.1 快速傅里叶变换的基本原理

非周期连续时间信号 $x(t)$的傅里叶变换可以表示为：

$$X(\omega)=\int_{-\infty}^{\infty}x(t)\mathrm{e}^{-\mathrm{j}\omega t}\mathrm{d}t \tag{5.8}$$

以上式计算出来的是信号 $x(t)$的连续频谱。但是，在实际的系统中我们能得到的是信号 $x(t)$

的离散采样值 $x(nT)$（T 是采样周期）。因此以离散信号 $x(nT)$ 来计算其频谱具有实际意义。不失一般性，设经采样得到了 N 点采样值 $\{x(nT), n=0,1,\cdots,N-1\}$，那么其频谱取样的谱间距是

$$\omega_0 = \frac{2\pi}{NT}$$

这样一来可推得式(5.8)的离散形式为：

$$X(k\omega_0) = \sum_{n=0}^{N-1} x(nT)\mathrm{e}^{-\mathrm{j}k\omega_0 nT} = \sum_{n=0}^{N-1} x(nT)\mathrm{e}^{-\mathrm{j}\frac{2\pi}{N}kn} \tag{5.9}$$

令 $W_N=\mathrm{e}^{-\mathrm{j}\frac{2\pi}{N}}$ 并省略符号 ω_0 和 T，则上式可写为：

$$X(k) = \sum_{n=0}^{N-1} x(n)W_N^{kn} \qquad k = 1,2,\cdots,N-1 \tag{5.10}$$

式中　$X(k)$ 是时间序列 $x(n)$ 的频谱，W_N 称为蝶形因子。对于 N 点时域采样值，经过式(5.10)的计算，就可得到 N 个频谱条，这就是离散傅里叶变换(DFT)。可以看出，DFT 需要计算约 N^2 次乘法和 N^2 次加法，在 N 较大时，这个计算量难以承受。

从哪些方面能改进 DFT 的运算以减少运算工作量呢?仔细考察 DFT 的运算就会看到，充分利用蝶形因子 W_N 的下列固有特性，即可改善 DFT 的运算效率。

- W_N 的对称性：$W_N^{k(N-n)}=W_N^{-kn}=(W_N^{kn})^*$
- W_N 的周期性：$W_N^{kn}=W_N^{k(N+n)}=W_N^{(k+N)n}$

利用 W_N 的对称性和周期性，将 N 点的 DFT 分解为两个 $N/2$ 点的 DFT，这样二个 $N/2$ 点的 DFT 总的计算量只是原来的一半($(N/2)^2+(N/2)^2=N^2/2$)。这样的分解可以继续下去，将 $N/2$ 点的 DFT 再分解为 $N/4$ 点的 DFT，等等。最小的分解点数称为基数(radix)，基 2 的 FFT 就是最小变换为 2 点 DFT。对于 $N=2^m$ 点的 DFT 都可以基 2 的 FFT 来实现，这样的计算量可减为 $(N/2)\log_2^N$ 个加乘运算，这比 DFT 的计算量大大地减轻。要注意的是 FFT 不是 DFT 的近似计算，它们是完全等效的。下面给出两种 FFT 的实现方式。

1. 时间抽取(DIT)的 FFT 算法

为了讨论方便，设 $N=2^m$。如果不满足这个条件，可以人为地加上若干零值点来达到。将时间序列 $x(n)$ 分为偶序列和奇序列，从式(5.10)可得

$$\begin{aligned}
X(k) &= \sum_{n=0}^{N-1} x(n)W_N^{kn} \\
&= \sum_{r=0}^{N/2-1} x(2r)W_N^{2rk} + \sum_{r=0}^{N/2-1} x(2r+1)W_N^{(2r+1)k} \\
&= \sum_{r=0}^{N/2-1} x(2r)(W_N^2)^{rk} + W_N^k \sum_{r=0}^{N/2-1} x(2r+1)(W_N^2)^{rk}
\end{aligned} \tag{5.11}$$

由于 $W_N^2=\mathrm{e}^{-\mathrm{j}\frac{2\pi}{N}\times 2}=\mathrm{e}^{-\mathrm{j}\frac{2\pi}{N/2}}=W_{N/2}$，所以

$$X_0(k) = \sum_{r=0}^{N/2-1} x(2r)W_{N/2}^{rk} \qquad k = 0,1,\cdots,N/2-1 \tag{5.12a}$$

$$X_1(k) = \sum_{r=0}^{N/2-1} x(2r+1)W_{N/2}^{rk} \qquad k = 0,1,\cdots,N/2-1 \tag{5.12b}$$

可以看出，$X_0(k)$ 和 $X_1(k)$ 正好是 $x(n)$ 的偶序列和奇序列的 DFT。并且考虑到蝶形因子的周期性，即 $W_{N/2}^{rk}=W_{N/2}^{r(N/2+k)}$，所以又有

$$X_0(N/2+k)=\sum_{r=0}^{N/2-1}x(2r)W_{N/2}^{r(N/2+k)}=\sum_{r=0}^{N/2-1}x(2r)W_{N/2}^{rk}=X_0(k)$$

同理，$X_1(N/2+k)=X_1(k)$。综上可得 $X(k)$的前半部分为：

$$X(k)=X_0(k)+W_N^kX_1(k)\qquad k=0,1,\cdots,N/2-1 \tag{5.13a}$$

后半部分为：

$$\begin{aligned}X(N/2+k)&=X_0(k)+W_N^{N/2+k}X_1(k)\\&=X_0(k)-W_N^kX_1(k)\qquad k=0,1,\cdots,N/2-1\end{aligned} \tag{5.13b}$$

式(5.13)的运算可用流图来表示，如图 5.4 所示，由于形状类似蝴蝶，故称为蝶形运算。

从前面的推导得知，如果先计算出 $N/2$ 个点的 $X_0(k)$和 $X_1(k)$，就能由式(5.13)组合出 $X(k)$。按照这个原理，可继续将 $N/2$ 的子序列再按偶序列和奇序列分解为二个 $N/4$ 点的子序列。持续这样的分解，最后可以得到 2 点的 DFT。也就是一个 $N=2^m$ 点的 DFT，可以进行 m－1 级的分解，到最后一级就是基 2 点的 DFT。计算时，先进行最后一级的基 2 的 DFT 运算；然后逐级后退，以式(5.13)组合出前级的 DFT，最后得到所要求的 $X(k)$。下面以 $N=2^3$ 为例说明整个运算的过程。

先将时间序列 $x(n)$进行分解排序，其下标的排序规律如表 5.1 所示。排序后得到的序列 $\bar{x}(i)$的下标与原始序列 $x(n)$的下标以二进制表示时正好反序。

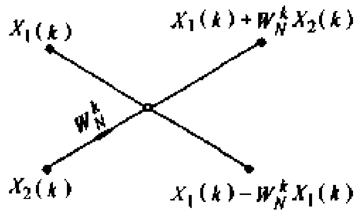


图 5.4 DIT 的蝶形运算

表 5.1 FFT 算法中下标分解排序表

原始 $x(n)$		第一次分解	第二次分解 $\bar{x}(i)$	
000	0	0	0	000
001	1	2	4	100
010	2	4	2	010
011	3	6	6	110
100	4	1	1	001
101	5	3	5	101
110	6	5	3	011
111	7	7	7	111

从式(5.13)可得第一次分解的结果：

$$X(k)\Rightarrow\begin{cases}X_0(k)+W_{2^l}^kX_1(k)\\X_0(k)-W_{2^l}^kX_1(k)\end{cases}\quad l=m=3,k=0,1,\cdots,2^{l-1}-1 \tag{5.14}$$

同样的道理可得第二次分解的结果：

$$X_0(k)\Rightarrow\begin{cases}X_{00}(k)+W_{2^l}^kX_{01}(k)\\X_{00}(k)-W_{2^l}^kX_{01}(k)\end{cases}\quad l=m-1=2,k=0,1,\cdots,2^{l-1}-1 \tag{5.15a}$$

$$X_1(k)\Rightarrow\begin{cases}X_{10}(k)+W_{2^l}^kX_{11}(k)\\X_{10}(k)-W_{2^l}^kX_{11}(k)\end{cases}\quad l=m-1=2,k=0,1,\cdots,2^{l-1}-1 \tag{5.15b}$$

此时，$X_{00}(k)$、$X_{01}(k)$、$X_{10}(k)$和 $X_{11}(k)$是基 2 的 DFT，即

$$X_{00}(k)=\sum_{i=0}^{1}\bar{x}(i+0)W_{2^l}^{ik}\Rightarrow\begin{cases}\bar{x}(0)+W_{2^l}^k\bar{x}(1)\\\bar{x}(0)-W_{2^l}^k\bar{x}(1)\end{cases}\quad l=m-2=1\quad k=0 \tag{5.16a}$$

$$X_{01}(k) = \sum_{i=0}^{1}\bar{x}(i+2)W_2^{ik} \Rightarrow \begin{cases}\bar{x}(2) + W_2^{k}\bar{x}(3) \\ \bar{x}(2) - W_2^{k}\bar{x}(3)\end{cases} \quad l = m - 2 = 1 \quad k = 0 \quad (5.16b)$$

$$X_{10}(k) = \sum_{i=0}^{1}\bar{x}(i+4)W_2^{ik} \Rightarrow \begin{cases}\bar{x}(4) + W_2^{k}\bar{x}(5) \\ \bar{x}(4) - W_2^{k}\bar{x}(5)\end{cases} \quad l = m - 2 = 1 \quad k = 0 \quad (5.16c)$$

$$X_{11}(k) = \sum_{i=0}^{1}\bar{x}(i+6)W_2^{ik} \Rightarrow \begin{cases}\bar{x}(6) + W_2^{k}\bar{x}(7) \\ \bar{x}(6) - W_2^{k}\bar{x}(7)\end{cases} \quad l = m - 2 = 1 \quad k = 0 \quad (5.16d)$$

综上所述，对于按时间抽取的 FFT 的计算过程如下：

- 按照下标反序的规则，将原始时间序列 $x(n)$重新排序为 $\bar{x}(i)$。
- 按照式(5.16)计算最后一级 2 点 DFT。
- 按照式(5.15)及式(5.14)，逐级后退直到组合出 $X(k)$。

这个过程也可用图 5.5 的流图来表示。

2. 按频率抽取(DIF)的 FFT 算法

与按时间抽取的 FFT 算法相对应，还存在另一种称为按频率抽取的算法。这个算法不是将时间序列 $x(n)$分解，而是将代表频域的输出序列 $X(k)$(也有 N 个点)按其顺序是属于偶数还是奇数分解为越来越短的序列。

先将式(5.10)分为前后两半，即

$$\begin{aligned} X(k) &= \sum_{n=0}^{N/2-1} x(n)W_N^{nk} + \sum_{n=N/2}^{N-1} x(n)W_N^{nk} \\ &= \sum_{n=0}^{N/2-1} x(n)W_N^{nk} + \sum_{n=0}^{N/2-1} x(n+N/2)W_N^{(n+N/2)k} \\ &= \sum_{n=0}^{N/2-1} [x(n) + x(n+N/2)W_N^{kN/2}]W_N^{nk} \\ &= \sum_{n=0}^{N/2-1} [x(n) + (-1)^k x(n+N/2)]W_N^{nk} \qquad k = 0,1,\cdots,N-1 \qquad (5.17) \end{aligned}$$

将上式按 k 是偶数还是奇数分解得到：

$$\begin{aligned} X(2r) &= \sum_{n=0}^{N/2-1} [x(n) + x(n+N/2)]W_N^{2rn} \\ &= \sum_{n=0}^{N/2-1} [x(n) + x(n+N/2)]W_{N/2}^{rn} \qquad r = 0,1,\cdots,N/2-1 \qquad (5.18a) \end{aligned}$$

$$\begin{aligned} X(2r+1) &= \sum_{n=0}^{N/2-1} [x(n) - x(n+N/2)]W_N^{(2r+1)n} \\ &= \sum_{n=0}^{N/2-1} [x(n) - x(n+N/2)]W_N^{n}W_{N/2}^{rn} \qquad r = 0,1,\cdots,N/2-1 \qquad (5.18b) \end{aligned}$$

式(5.18)的运算可以用图 5.6 的蝶形运算流图来表示。

与按时间抽取的算法类似，上面按频率奇偶的分解可以持续下去直到求 2 点 DFT 为止。按频率抽取算法的计算量与按时间抽取的是一样的。从流图来看，它们是互为转置的。关于按频率抽取算法的详细计算过程，这里不再详叙，有兴趣的读者可参考有关信号处理方面的资料。

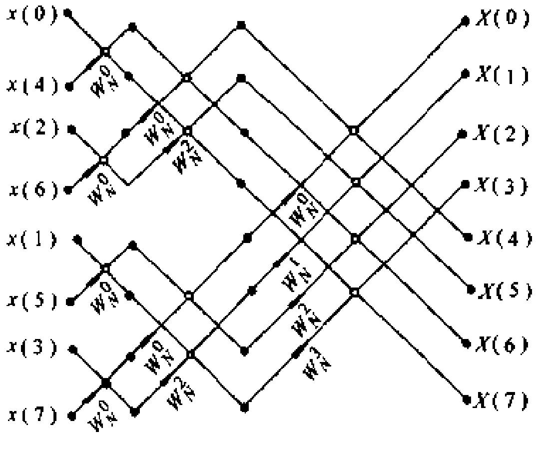


图 5.5 8 点 DIT 的 FFT 蝶形运算

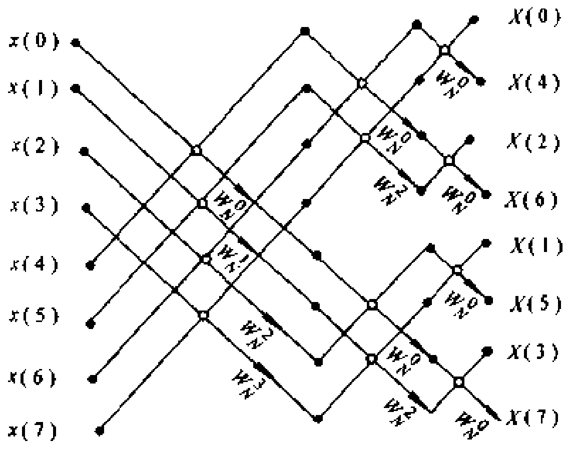


图 5.6 DIF 的蝶形运算

5.2.2 快速傅里叶变换的 DSP 实现

FFT 较之 DFT 的计算量减少了非常多，但 FFT 要做到多点（N 较大）、实时的运算，对于普通单片机来说，还不是一件容易的事。一方面，FFT 需要对原始序列进行反序排列；另一方面，由于 $X(k)$ 是复数，蝶形运算是复数运算，需要多次地查表相乘运算才能实现。因此，为了实现实时的 FFT，需要单片机的指令系统有着丰富的间接寻址方式，并且最好能在一个指令周期内完成乘和累加的工作。DSP 控制器具备这样的条件。从第 4 章知，DSP 控制器特有的反序间接寻址，就是专为 FFT 算法而设计的，其它的间接寻址方式还可以实现增/减 1 或增/减一个变址量，这为各种查表方法的实现提供了方便。另外，DSP 控制器能在一个指令周期内完成乘和累加的工作。因此，以 DSP 控制器来实现 FFT 算法较普通单片机要容易许多。下面讨论它的具体实现方法。

仔细分析式(5.14)、式(5.15)和式(5.16)可以得知，$N=2^m$ 点 FFT 的运算是非常有规律的，可以通过下面三个循环来完成：

For $l=1$ To m

  For $k=0$ To $2^{l-1}-1$

    For $i=k$ To 2^m-1 Step 2^l

      $j=i+2^{l-1}$

      $$\bar{x}(i)\Leftarrow\bar{x}(i)+W_{2^l}^{k}\bar{x}(j) \tag{5.19a}$$

      $$\bar{x}(j)\Leftarrow\bar{x}(i)-W_{2^l}^{k}\bar{x}(j) \tag{5.19b}$$

    Next i

  Next k

Next l

前面是借用 Basic 语言的符号来说明 FFT 运算的规律，在用高级语言实现时，要注意式(5.19)需通过中间变量进行缓存，并且 $\bar{x}(i)$、$\bar{x}(j)$ 和 $W_{2^l}^{k}$ 是复数形式。在第一次循环时式(5.19)中 $\bar{x}(i)$ 的值是原始序列 $x(n)$ 经反序排列后的数据，经过三重循环后最后放在 $\bar{x}(i)$ 中的数正好对应 $X(k)$ 的值。

考虑到式(5.19)是复数运算，令

$$\bar{x}(i) = \bar{R}_i + j\bar{I}_i \qquad \bar{x}(j) = \bar{R}_j + j\bar{I}_j \tag{5.20}$$

$$W_{2^l}^{k} = \cos\alpha_{lk} - j\sin\alpha_{lk} \tag{5.21a}$$

$$\alpha_{lk} = \frac{2\pi}{2^l}k = \frac{2\pi}{N}2^{m-l}k \quad l = 1,2,\cdots,m;k = 0,1,\cdots,2^{l-1} - 1 \tag{5.21b}$$

则式(5.19)可化为：

$$R_j = \bar{R}_j\cos\alpha_{lk} + \bar{I}_j\sin\alpha_{lk} \qquad R_i = \bar{R}_i \tag{5.22a}$$

$$I_j = \bar{R}_j\sin\alpha_{lk} - \bar{I}_j\cos\alpha_{lk} \qquad I_i = \bar{I}_i \tag{5.22b}$$

$$\bar{x}(i) \Leftarrow (R_i + R_j) + j(I_i - I_j) \tag{5.22c}$$

$$\bar{x}(j) \Leftarrow (R_i - R_j) + j(I_i + I_j) \tag{5.22d}$$

为了用DSP控制器来实现上述过程，还需考虑数据的大小以免发生溢出。不失一般性，设原始时间序列$x(n)$已进行归一化，即为Q_{15}的格式（最高位为符号，其余位是小数）。不难推知，式(5.19)或式(5.22)可能的最大值为：

$$1 + \sin45° + \cos45° = 1 + \sqrt{2} = 2.4142$$

这将超出Q_{15}的格式范围。考虑到在大多数情况下是实数FFT，这个最大值不超过2。因此，可在每一级用因子2进行归一化。运用DSP控制器指令系统的移位特性，用2归一化不增加任何运算量。这样，对于$N=2^m$点FFT的运算，若每级用2归一化，则最后的输出相当于除以了$N=2^m$。

值得一提的是，为了避免溢出而对每一级都进行归一化会降低运算的精度。因此，只对可能产生溢出的进行归一化处理是上策。

式(5.22)的运算涉及到正弦函数与余弦函数，因此在编程前需要预制正弦和余弦表。为了容易实现查表并节省存储容量，在式(5.21b)中令$l=m$并按如下规律存放这个表：

$$\text{For } k=0 \text{ To } 2^{m-1}-1$$

$$\cos\left(\frac{2\pi}{N}k\right);\sin\left(\frac{2\pi}{N}k\right) \tag{5.23}$$

Next k

不难验证，当$l<m$时也可以从上面这个表中找到所需要的正弦和余弦值。

另外，仔细分析式(5.19)的循环知道，第一层的循环次数为m；第二层循环的次数依次为$2^{l-1}(l=1,2,\cdots,m)$；第三层循环的次数依次为$2^{m-l}(l=1,2,\cdots,m)$。为了说明DSP控制器实现FFT的优越性，下面以具体的指令程序代码来说明它的实现过程。

```
SETC OVM              ;设置溢出方式为1,溢出时以最大值填入
SETC SXM              ;进行符号扩展
SPM 1                 ;乘积结果左移1位,自动将两个Q15相乘后化为Q15
LAR AR0,N             ;N为点数
LAR AR1,IN_DATA       ;IN-DATA为输入的原始序列的首地址,连续存放
LAR AR2,OUT_DATA      ;OUT-DATA为输出的频域序列的首地址,按实部、虚

LACL N                ;部存放
SUB 1                 ;
SACL N                ;N=N-1
```

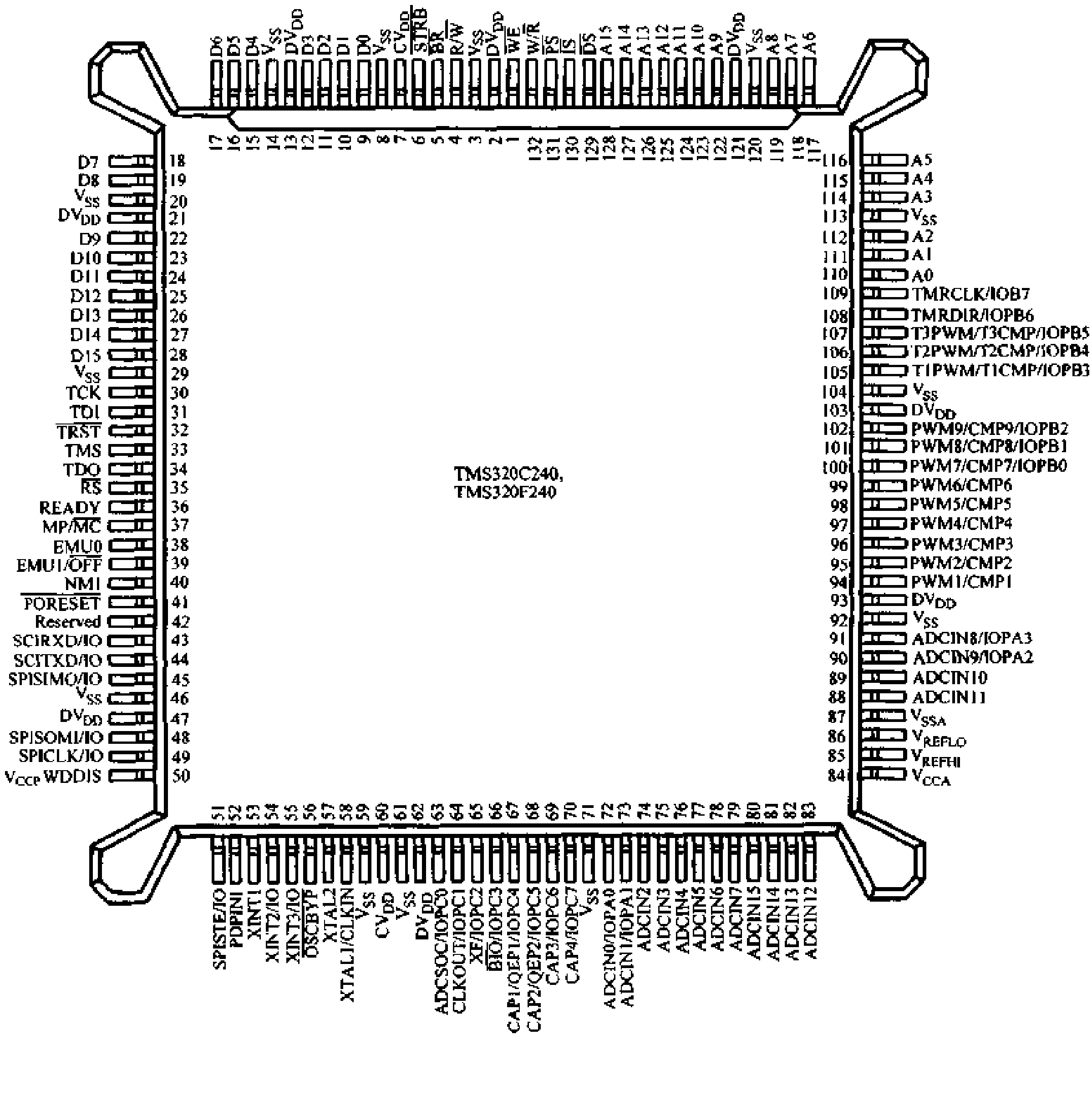
```
        LAR AR3,N              ;
        MAR  *,AR1             ;当前 AR=AR1

                               ;下面将原始序列反序排列到 OUT-DATA 的实部处
LD_Re:  LACL  *+,0,AR2         ;ACC←(AR1),AR1+1,下个 AR 是 AR2
        SACL  *BR0+,0,AR3      ;(AR2)←(ACC),AR2 反序加 AR0,下个 AR 是 AR3
        BANZ LD_Re,*-,AR1;(AR3)<  >0 转 LD-Re,AR3-1,下个 AR 是 AR1

                               ;下面将 0 反序排列到 OUT-DATA 的虚部处
LD_Im:  LACL 0                 ;ACC←0
        MAR  *,AR2             ;当前 AR=AR2
        SACL  *BR0+,0,AR3      ;(AR2)←(ACC),AR2 反序加 AR0,下个 AR 是 AR3
        BANZ LD_Im,*-,AR1;(AR3)<  >0 转 LD-Im,AR3-1。下个 AR 是 AR1

        LACL 1                 ;
        SACL EXPl              ;变量 EXPl 存放 2^(l-1)
        LACL N                 ;
        ADD 1                  ;
        SACL N                 ;由 N-1 还原为 N
        SACL EXPm-l,1          ;变量 EXPm-l 存放 2^(m-l)×2

        LACL m                 ;N=2^m
        SACL lLOOP             ;变量 lLOOP 存放第一层循环的次数。
LOOP_l:LAR AR1,OUT_DATA ;AR1 记 x̄(i)、x̄(j)的下标 i、j
        LAR AR3,SIN_Table      ;AR3 记 2^(m-l)k×2,即正弦表的地址指针(由于按 cos、
        LACL EXPl,1            ;sin 存放,所以乘以 2)。
        SACL EXPl              ;EXPl=2^(l-1)×2
        LACL EXPl,15           ;
        SACH kLOOP             ;变量 kLOOP 存放第二层循环的次数 EXPl/2=2^(l-1)
        LACL EXPm-l,15         ;
        SACH EXPm-l            ;EXPm-l=2^(m-l)
        SACH iLOOP             ;变量 iLOOP 存放第三层循环的次数 2^(m-l)
LOOP_k  LAR AR0,EXPl           ;AR0←2^(l-1)×2(由于按实、虚存放,所以乘 2)

LOOP_i  MAR  *0+,AR3           ;AR1←(AR1)+(AR0),准备坐标 j
        LACL 0                 ;AR1 对应坐标 j,AR3 对应正弦表的地址指针 2^(m-l)k
        LT  *+,AR1             ;T←(AR3)=cosα_lk,AR3+1
        MPY  *+,AR3            ;P←T×(AR1)=R̄_j cosα_lk,AR1+1
        LT  *,AR1              ;T←(AR3)=sinα_lk
```

```
MPYA *-,AR3          ;ACC←(ACC)+(P),P←T×(AR1)=Ījsinαlk,AR1-1
APAC                 ;ACC←(ACC)+(P)=R̄jcosαlk+Ījsinαlk
SACH Rj              ;
LACL 0               ;
LT *-,AR1            ;T←(AR3)=sinαlk,AR3-1
MPY *+,AR3           ;P←T×(AR1)=R̄jsinαlk,AR1+1
LT *,AR1             ;T←(AR3)=cosαlk
MPYA *-              ;ACC←(ACC)+(P),P←T×(AR1)=Ījcosαlk,AR1-1
SPAC                 ;ACC←(ACC)-(P)=R̄jsinαlk-Ījcosαlk
SACH Ij              ;

MAR *0-              ;AR1←(AR1)-(AR0)对应坐标i(实部)
LACC,*,15            ;
ADD Rj,15            ;
SACH *+              ;AR1←(Ri+Rj)/2,实部进行除2的归一化,AR1+1
LACC *,15            ;坐标i(虚部)
ADD Ij,15            ;
SACH *-              ;AR1←(Ii+Ij)/2,虚部进行除2的归一化,AR1-1

LACC *,15            ;回到坐标i(实部)
SUB Rj,15            ;
SACH *+              ;AR1←(Ri-Rj)/2,实部进行除2的归一化,AR1+1
LACC *,15            ;坐标i(虚部)
SUB Ij,15            ;
SACH *-              ;AR1←(Ii-Ij)/2,虚部进行除2的归一化,AR1-1


                     ;
LAR AR0,EXPl         ;
MAR *0+              ;准备下个坐标i
LACL iLOOP           ;
SUB 1                ;
SACL iLOOP           ;
BGZ LOOP_i,*,AR3     ;返回到循环LOOP_i

LAR AR0,EXPm-l       ;
MAR *0+              ;AR3←(AR3)+(AR0),准备下个正弦表的指针2^(m-l)k
LACL kLOOP           ;
SUB 1
SACL KLOOP           ;
```

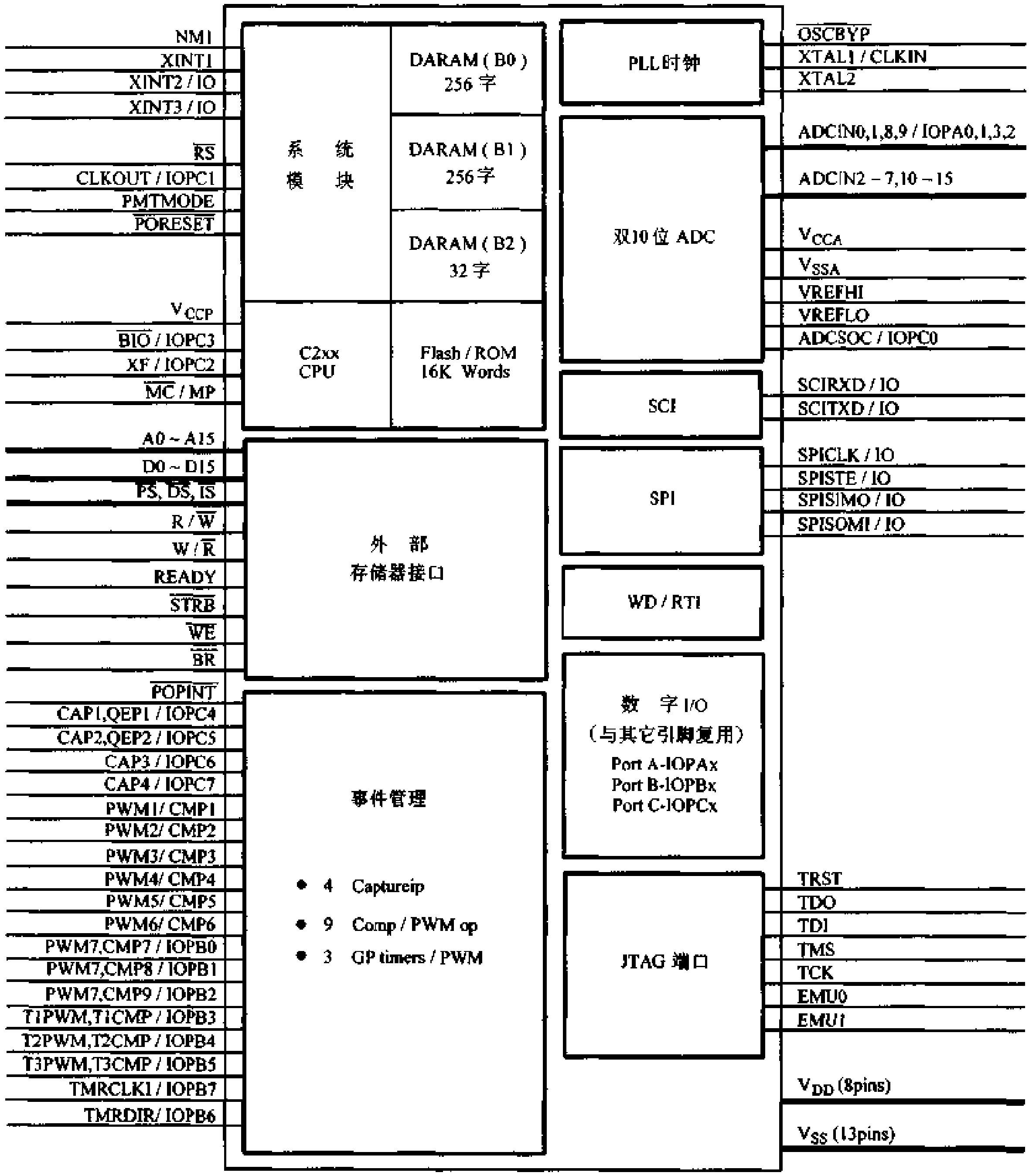
```
BGZ LOOP _k, * ,AR1   ;返回到循环 LOOP _k

LACL lLOOP            ;
SUB 1                 ;
SACL lLOOP            ;
BGZ LOOP _l, * ,AR1   ;返回到循环 LOOP _l
RET                   ;
```

附　　录

附录 A　DSP240 引脚说明



附图 A



附图 A(续)

DSP240 引脚说明

引脚		类 型	说 明
名 称	编 号		
A0(LSB)	110	O/Z	并行地址总线 A0(低位)至 A15(高位)。可寻址外部数据/程序存储器或 I/O 空间。在$\overline{OFF}$低电平有效时量为高阻态。在节电模式中保持原来状态
A1	111		
A2	112		
A3	114		
A4	115		

（续）

引脚		类型	说明
名称	编号		
A5	116	O/Z	并行地址总线 A0(低位)至 A15(高位)。可寻址外部数据/程序存储器或 I/O 空间。在$\overline{OFF}$低电平有效时置为高阻态。在节电模式中保持原来状态
A6	117		
A7	118		
A8	119		
A9	122		
A10	123		
A11	124		
A12	125		
A13	126		
A14	127		
A15(MSB)	128		
D0(LSB)	9	I/O/Z	并行数据总线 D0(低位)至 D15(高位)。可在 DSP 与外部数据/程序存储器及 I/O 空间(片内外设)之间传递数据。在无输出、节电模式、复位申请($\overline{RS}$)或$\overline{OFF}$低电平有效时置为高阻态
D1	10		
D2	11		
D3	12		
D4	15		
D5	16		
D6	17		
D7	18		
D8	19		
D9	22		
D10	23		
D11	24		
D12	25		
D13	26		
D14	27		
D15(MSB)	28		
接口控制信号			
$\overline{DS}$	129	O/Z	数据、程序和 I/O 空间选择信号，总是为高电平，仅在对指定外部空间通信申请时为低电平。在复位、节电和$\overline{OFF}$为有效低电平时置为高阻态
$\overline{PS}$	131		
$\overline{IS}$	130		
READY	36	I	数据就绪引脚。表示一个外围设备经总线传送的数据准备就绪。如果设备未准备好(READY 为低)，处理器等待一个周期后再检测 READY 引脚
R/$\overline{W}$	4	O/Z	读/写信号。表示外围设备通信期间数据的传送方向。通常为高电平读模式，仅在执行写操作时为低电平。在复位、节电和$\overline{OFF}$为有效低电平时置为高阻态
STRB	6	O/Z	选通信号。总是高电平，仅在表示一个外部总线周期时变为低电平。在复位、节电和$\overline{OFF}$为有效低电平时置为高阻态
$\overline{WE}$	1	O/Z	写使能信号。$\overline{WE}$的下降缘表示设备正驱动外部数据总线(D15～D0)。外围设备在$\overline{WE}$的上升缘锁定数据。$\overline{WE}$在所有写外部程序、数据和 I/O 时有效。在复位后和当$\overline{OFF}$为有效低电平时置为高阻态

（续）

引脚		类型	说明
名称	编号		
W/$\overline{\text{R}}$	132	O/Z	读/写信号。该信号为R/$\overline{\text{W}}$的非。可直接连接到外围设备的输出使能端。在复位后和$\overline{\text{OFF}}$为有效低电平时置为高阻态
$\overline{\text{BR}}$	5	O/Z	总线申请信号、$\overline{\text{BR}}$在外部全局数据存储空间的存取期间有效。$\overline{\text{BR}}$可用于数据存储地址空间扩大到32字。$\overline{\text{BR}}$在复位、节电和$\overline{\text{OFF}}$为有效低电平时变为高阻态
V_{ccp}	50	I	闪速编程电源引脚。如果$V_{ccp}=5V$，则WRITE/ERASE可用于所有片内闪速存储块。例如，对闪速存储器编程。如果$V_{ccp}=0V$，则闪速存储器的WRITE/ERASE被禁止。因此可防止所有存储块被重写
ADC输入端(非共享引脚)			
ADCIN2	74	I	至第一块ADC的模拟输入
ADCIN3	75	I	
ADCIN4	76	I	
ADCIN5	77	I	
ADCIN6	78	I	
ADCIN7	79	I	
ADCIN10	89	I	至第一块ADC的模拟输入
ADCIN11	88	I	
ADCIN12	83	I	
ADCIN13	82	I	
ADCIN14	81	I	
ADCIN15	80	I	
位I/O和共享功能引脚			
ADCIN0/IOPA0	72	I/O I	双向数字I/O 到第1片ADC模拟输入
ADCIN1/IOPA1	73	I/O I	双向数字I/O 到第1片ADC模拟输入
ADCIN9/IOPA2	90	I/O I	双向数字I/O 到第2片ADC模拟输入
ADCIN8/IOPA3	91	I/O I	双向数字I/O 到第2片ADC模拟输入
PWM7/CMP7/IO PB0	100	I/O O/Z	双向数字I/O。单比较/PWM 1输出引脚。引脚状态由单比较/PWM和单动作控制寄存器(SACTR)决定。当未屏蔽信号$\overline{\text{PDPINT}}$低电平有效时变为高阻态
PWM8/CMP8/IO PB1	101	I/O O/Z	双向数字I/O。单比较/PWM 2输出引脚，引脚状态由单比较/PWM和单动作控制寄存器(SACTR)决定。当未屏蔽信号$\overline{\text{PDPINT}}$低电平有效时变为高阻态

（续）

引脚		类型	说明
名称	编号		
PWM9/CMP9/IO PB2	102	I/O O/Z	双向数字 I/O。单比较/PWM 3 输出引脚。引脚状态由单比较/PWM 和单动作控制寄存器(SACTR)决定。当未屏蔽信号$\overline{\text{PDPINT}}$低电平有效时变为高阻态
T2PWM/T2CMP /IOPB3	105	I/O O/Z	双向数字 I/O。定时器 1 比较输出。引脚状态当未屏蔽信号$\overline{\text{PDPINT}}$低电平有效时变为高阻态
T3PWM/T3CMP /IOPB4	106	I/O O/Z	双向数字 I/O。定时器 2 比较输出。引脚状态当未屏蔽信号$\overline{\text{PDPINT}}$低电平有效时变为高阻态
T1PWM/T1CMP /IOPB5	107	I/O O/Z	双向数字 I/O。定时器 3 比较输出。引脚状态当未屏蔽信号$\overline{\text{PDPINT}}$低电平有效时变为高阻态
TMRDIR/IOPB6	108	I/O I	双向数字 I/O。定时器的方向信号。低电平时为增计数。高电平时为减计数
TMRCLK/IOPB7	109	I/O I	双向数字 I/O 通用定时器的外部时钟输入
ADCSOC/IOPC0	63	I/O I	双向数字 I/O ADC 转换外部启动输入
CAP1/QEP1/IOPC4	67	I/O I	双向数字 I/O 捕获 1 或 QEP1 输入
CAP2/QEP2/IOPC5	68	I/O I	双向数字 I/O 捕获 2 或 QEP2 输入
CAP3/IOPC6	69	I/O I	双向数字 I/O 捕获 3 输入
CAP4/IOPC7	70	I/O I	双向数字 I/O 捕获 4 输入
XF/IOPC2	65	I/O I	双向数字 I/O。外部标志输出(封锁软件可编程信号)。XF 在多处理结构中用于指示其它处理器，或作为通用输出引脚
$\overline{\text{BIO}}$/IOPC3	66	I/O I	双向数字 I/O。转移控制输入。$\overline{\text{BIO}}$由 BIOZ 指令查询。如果$\overline{\text{BIO}}$为低电平，CPU 执行一个转移操作。如果$\overline{\text{BIO}}$不使用，应将其拉为高电平
CLKOUT/IOPC1	64	I/O I	双向数字 I/O，时钟输出引脚，时钟输出由 SYSCR 寄存器的 CLKSRC 位选择
			串行通信和位 I/O 引脚
SCITXD/IO	44	I/O	SCI 异步串行口传送数据，或通用双向 I/O
SCIRXD/IO	43	I/O	SCI 异步串行口接收数据，或通用双向 I/O
SPISIMO/IO	45	I/O	SPI 从入，主出，或通用双向 I/O
SPISOMI/IO	48	I/O	SPI 从出，主入，或通用双向 I/O
SPICLK/IO	49	I/O	SPI 时钟，或通用双向 I/O

（续）

引脚		类型	说明
名称	编号		
SPISTE/IO	51	I/O	SPI 从传送使能(可选择)，或通用双向 I/O
			比较信号
PWM1/CMP1 PWM2/CMP2 PWM3/CMP3 PWM4/CMP4 PWM5/CMP5 PWM6/CMP6	94 95 96 97 98 99	O/Z	比较单元比较或 PWM 输出。这些引脚的状态由比较/PWM 和 ACTR 确定。CMP1—CMP6 当未屏蔽信号 $\overline{\text{PDPINT}}$ 低电平有效时变为高阻态
			中断和其他信号
$\overline{\text{RS}}$	35	I/O	复位引脚。使 TMS320×240 终止执行并使 PC=0。当 $\overline{\text{RS}}$ 变为高电平时，程序从程序存储器 0h 地址开始执行。$\overline{\text{RS}}$ 影响(或置为零)各寄存器和状态位
MP/$\overline{\text{MC}}$	37	I	MP/$\overline{\text{MC}}$(微处理器/微计算机)选择。若为低电平，选择内部程序存储器。若为高电平，选择外部程序存储器
NMI	40	I	不可屏蔽中断。当该引脚变低时，无论状态寄存器 0 的 INTM 位的状态如何，设备都会产生中断
$\overline{\text{PORESET}}$	41	I	上电复位输入引脚。$\overline{\text{PORESET}}$ 使 TMS320×240 终止执行并使 PC=0。当 $\overline{\text{PORESET}}$ 变为高电平时，程序从程序存储器 0h 地址开始执行。$\overline{\text{RS}}$ 影响(或置为零)各寄存器和状态位(与 $\overline{\text{RS}}$ 相同)。另外，$\overline{\text{PORESET}}$ 初始化 PLL 控制寄存器
XINT1	53	I	外部用户中断 1
XINT2/IO	54	I/O	外部用户中断 2。通用双向 I/O
XINT3/IO	55	I/O	外部用户中断 3。通用双向 I/O
$\overline{\text{PDPINT}}$	52	I	可屏蔽掉电保护中断。若 $\overline{\text{PDPINT}}$ 未屏蔽，则为低电平有效，定时器比较输出立即变为高阻态
			时钟信号
XTAL2	57	O	PLL 振荡器输出引脚。当设备在 PLL 模式(CLKMD[1:0]=1x，CKCR0.7:6)时，XTAL2 受限于参考晶振。在振荡器旁路模式($\overline{\text{OSCBYP}} \leqslant V_{IL}$)。当 EMU1/$\overline{\text{OFF}}$ 低电平有效时，该引脚变为高阻态
XTAL1/CLKIN	58	I/Z	PLL 振荡器输入引脚。当设备在 PLL 模式(CLKMD[1:0]=1x，CKCR0.7:6)或在振荡器旁路模式($\overline{\text{OSCBYP}} \leqslant V_{IL}$)时，XTAL1/CLKIN 受限于参考晶振
OSCBYP	56	I	低电平时旁路振荡器

（续）

引脚名称	引脚编号	类型	说明
电源信号			
V_{SS}	3 14 20 29 46 61 71 92 104 113 120 8 59	I	数字逻辑参考地
V_{SSA}	87	I	模拟参考地
DV_{DD}	2 13 21 47 62 93 103 121	I	数字 I/O 逻辑电源电压
CV_{DD}	7 60	I	数字核逻辑电源电压
V_{CCA}	84	I	模拟电源电压
V_{refHi}	85	I	ADC 模拟参考高电压
V_{refLo}	86	I	ADC 模拟参考低电压
测试信号			
TCK	30	I	IEEE 标准测试时钟。通常是具有 50% 占空比的固定时钟信号。测试存取口(TAP)输入信号(TMS 和 TDI)的变化在 TCK 的上升沿被记录到 TAP 控制器，指令寄存器，或选择的'C2×× 核的测试数据寄存器。TAP 输出信号(TDO)的变化出现在 TCK 的下降沿
TDI	31	I	IEEE 标准测试数据输入。TDI 在 TCK 的上升沿被记录到选择的寄存器(指令或数据)
TDO	34	O/Z	IEEE 标准测试数据输出。被选择的寄存器(指令或数据)内容在 TCK 的下降沿移出 TDO。在 $\overline{OFF}$ 是有效低电平时，TDO 为高阻态
TMS	33	I	IEEE 标准测试模式选择。该串行控制输入在 TCK 上升沿被记录到 TAP 控制器

（续）

引脚 名称	引脚 编号	类型	说明
$\overline{TRST}$	32	I	IEEE 标准测试复位。在有效低电平时，$\overline{TRST}$产生对设备操作的系统控制扫描。如果该信号没有连接或为低电平，设备以自己的功能模式工作，而测试复位信号不起作用
EMU0	38	I/O/Z	仿真器引脚 0。当$\overline{TRST}$为低电平，由于$\overline{OFF}$的触发条件，该引脚应为高电平。当$\overline{TRST}$为高电平，该引脚作为一个到或来自仿真系统的中断并经扫描定义为输入/输出
EMU1/$\overline{OFF}$	39	I/O/Z	仿真引脚 1/禁止所有输出。当$\overline{TRST}$为高电平，该引脚作为一个到或来自仿真系统的中断并经 JTAG 扫描定义为输入/输出。当$\overline{TRST}$为低电平，该引脚成为$\overline{OFF}$。在有效低电平时，EMU1/$\overline{OFF}$将所有输出驱动器置为高阻态。注意$\overline{OFF}$仅用于测试和仿真目的（并不作为多处理应用）。因此，按$\overline{OFF}$条件，下列条件成立：$\overline{TRST}$＝低，EMU0＝高，EMU1/$\overline{OFF}$＝低
PMTMODE	42	I	闪速 EEPROM 并行测试使能引脚。该引脚有一内部下拉电路，可由用户断开

附录 B 可编程寄存器汇总

地址	寄存器	名称	页数
内部	ST0	状态寄存器 0	18
内部	ST1	状态寄存器 1	18
0004h	IMR	中断屏蔽寄存器	103
0006h	IFR	中断标志寄存器	103
7018h	SYSCR	系统控制寄存器	33
701Ah	SYSSR	系统状态寄存器	33
701Eh	SYSIVR	系统中断矢量寄存器	110
7021h	RTICNTR	实时中断计数寄存器	100
7023h	WDCNTR	看门狗计时寄存器	100
7025h	WDKEY	看门狗复位开关寄存器	100
7027h	RTICR	实时中断控制寄存器	101
7029h	WDCR	看门狗定时寄存器	101
702Bh	CKCR0	时钟控制寄存器 0	29
702Dh	CKCR1	时钟控制寄存器 1	30
7032h	ADCTRL1	ADC 控制寄存器 1	68
7034h	ADCTRL2	ADC 控制寄存器 2	69
7040h	SPICCR	SPI 外设接口配置控制寄存器	89

（续）

地　　址	寄 存 器	名　　称	页　　数
7041h	SPICTL	SPI 操作控制寄存器	90
7042h	SPISTS	SPI 状态寄存器	90
7044h	SPIBRR	SPI 波特率寄存器	90
7046h	SPIEMU	SPI 仿真缓冲寄存器	96
7047h	SPIBUF	SPI 串行输入缓冲寄存器	89
7049h	SPIDAT	SPI 串行数据寄存器	89
704Dh	SPIPC1	SPI 端口控制寄存器 1	94
704Eh	SPIPC2	SPI 端口控制寄存器 2	94
704Fh	SPIPRI	SPI 优先权控制寄存器	90
7050h	SCICCR	SCI 通信控制寄存器	78
7051h	SCICTL1	SCI 控制寄存器 1	79
7052h	SCIHBAUD	SCI 波特选择寄存器，高位	76
7053h	SCILBAUD	SCI 波特选择寄存器，低位	76
7054h	SCICTL2	SCI 控制寄存器 2	80
7055h	SCIRXST	SCI 接收状态寄存器	80
7056h	SCIRXEMU	SCI 仿真数据缓冲寄存器	76
7057h	SCIRXBUF	SCI 接收数据缓冲寄存器	77
7059h	SCITXBUF	SCI 转换数据缓冲寄存器	77
705Eh	SCIPC2	SCI 端口控制寄存器 2	82
705Fh	SCIPRI	SCI 优先权控制寄存器	83
7070h	XINT1CR	外部中断控制寄存器	110
7072h	NMICR	外部中断控制寄存器	110
7078h	XINT2CR	外部中断控制寄存器	110
707Ah	XINT3CR	外部中断控制寄存器	110
7090h	OCRA	I/O 多路控制寄存器 A	98
7092h	OCRB	I/O 多路控制寄存器 B	98
7098h	PADATDIR	I/O A 口数据和方向寄存器	98
709Ah	PBDATDIR	I/O B 口数据和方向寄存器	98
709Ch	PCDATDIR	I/O C 口数据和方向寄存器	98
7400h	GPTCON	通用定时器控制寄存器	40
7404h	T1CON	通用定时器 1 控制寄存器	39
7408h	T2CON	通用定时器 2 控制寄存器	39
740Ch	T3CON	通用定时器 3 控制寄存器	39
7411h	COMCON	比较控制寄存器	52
7413h	ACTR	全比较动作控制寄存器	55
7414h	SACTR	单比较动作控制寄存器	52

（续）

地　　址	寄　存　器	名　　称	页　　数
7415h	DBTCON	死区定时控制寄存器	58
7420h	CAPCON	捕捉控制寄存器	62
7422h	CAPFIFO	捕捉 FIFO 状态寄存器	64
742Ch	EVIMRA	EV 中断屏蔽寄存器 A	109
742Dh	EVIMRB	EV 中断屏蔽寄存器 B	109
742Eh	EVIMRC	EV 中断屏蔽寄存器 C	109
742Fh	EVIFRA	EV 中断标志寄存器 A	107
7430h	EVIFRB	EV 中断标志寄存器 B	109
7431h	EVIFRC	EV 中断标志寄存器 C	109
7432h	EVIVRA	EV 中断矢量寄存器 A	109
7433h	EVIVRB	EV 中断矢量寄存器 B	109
7434h	EVIVRC	EV 中断矢量寄存器 C	109
0h	SEG _ CTR	暂存区控制寄存器	
FFFFh	WSGR	等待状态发生器控制寄存器	

附录C　指 令 汇 总

语　　法	说　　明	操　　作　　码	页数
ABS	累加器取绝对值	1011　1110　0000　0000	142

ADD dma[,shift]
ADD ind,[,shift[,ARn]]
ADD dma,16
ADD ind,16[,ARn]
ADD #k
ADD #1k[,shift]
	左移 0～15 位后加至累加器，直接或间接寻址
	左移 16 位后加至累加器，直接或间接寻址
	加至累加器、短立即数
	左移 0～15 位后加至累加器，长立即数
		0010　SHFT　IAAA　AAAA
		0110　0001　IAAA　AAAA
		1011　1000　IIII　IIII
		1011　1111　1001　SHFT+1word
			143

ADDC dma ADDC ind[,ARn]	带进位位加至累加器，直接或间接寻址	0110　0000　IAAA　AAAA	144
ADDS dma ADDS ind[,ARn]	抑制符号扩展加至累加器，直接或间接寻址	0110　0010　IAAA　AAAA	145
ADDT dma ADDT ind[,ARn]	按 TREG 规定进行移位（0～15）后加至累加器，直接或间接寻址	0110　0011　IAAA　AAAA	145
ADRK #k	将短立即常数加至当前 AR	0111　1000　IIII　IIII	146
AND dam AND ind[,ARn] AND #1k[,shift] AND #1k,16	累加器和数值逻辑“与”，直接或间接 长立即数左移 0～15 位后和累加器逻辑“与” 长立即数左移 16 位后和累加器逻辑“与”	0110　1110　IAAA　AAAA 1011　1111　1011　SHFT+1word 1011　1110　1000　0001　+1word	156

（续）

语　　法	说　　明	操　作　码	页数
APAC	PREG 加至累加器	1011 1110 0000 0100	146
B pma[,ind[,ARn]]	间接寻址，无条件转移	0111 1001 1AAA AAAA +1word	162
BACC	转移至累加器低 16 位指定的地址	1011 1110 0010 0000	162
BANZ pma[,ind[,ARn]]	当前 AR 非零则转移	0111 1011 1AAA AAAA+1word	163
BCND pma,cond1[,cond][,…]	条件转移	1110 00TP ZLVC ZLVC +1word	163
BIT dma,bit code BIT ind,bit code[,ARn]	位测试，直接或间接寻址	0100 BITX IAAA AAAA	166
BITT dma BITT ind[,ARn]	测试由 TREG 指定的位，直接或间接寻址	0110 1111 IAAA AAAA	167
BLDD #1k,dma BLDD #1k,ind[,ARn]	数据存储器至数据存储器间的块移动，源地址为长立即数，目的地址为直接或间接寻址	1010 1000 IAAA AAAA +1word	123
BLDD dma,#1k[,ARn] BLDD ind,#1k[,ARn]	数据存储器至数据存储器间的块移动，目的地址为长立即数，源地址为直接或间接寻址	1010 1001 IAAA AAAA +1word	
BLPD #pma,dma BLPD #pma,ind[,ARn]	程序存储器至数据存储器间的块移动，源地址为长立即数，目的地址为直接或间接寻址	1010 0101 IAAA AAAA +1word	124
CALA	调用累加器低 16 位指定地址处的子程序	1011 1110 0011 0000	164
CALL pma[,ind[,ARn]]	调用子程序，间接寻址	0111 1010 1AAA AAAA+1word	164
CC	条件调用	1110 10 TP ZLVC ZLVC	165
CLRC control bit	清 C 位 清除 CNF 位 清除 INTM 位 清除 OVM 位 清除 SXM 位 清除 TC 位 清除 XF 位	1011 1110 0100 1110 1011 1110 0100 0100 1011 1110 0100 0000 1011 1110 0100 0010 1011 1110 0100 0110 1011 1110 0100 1010 1011 1110 0100 1100	168
CMPL	累加器求补	1011 1110 0000 0001	156
CMPR CM	比较当前 AR 和 AR0	1011 1111 0100 01CM	168
DMOV dma DMOV ind[,ARn]	将片内数据存储器内的数据复制到下一单元，直接或间接寻址	0111 0111 IAAA AAAA	124
IDLE	空闲直至中断发生	1011 1110 0010 0010	169
IN dma,PA IN ind,PA[,ARn]	从 I/O 单元输入数据，直接或间接寻址	1010 1111 IAAA AAAA +1word	125
INTR k	软中断	1011 1110 0111 NTR#	165

（续）

语　　法	说　　明	操　作　码	页数
LACC dma[,shift] LACC ind[,shift[,ARn]] LACC dma,16 LACC ind,16[,ARn] LACC #lk[,shift]	左移0～15位后装入累加器，直接或间接寻址 左移16位后装入累加器，直接或间接寻址 长立即数左移0～15位后装入累加器	0001 SHFT IAAA AAAA 0110 1010 IAAA AAAA 1011 1111 1000 SHFT+1word	125
LACL dma LACL ind[,ARn] LACL #k[,shift]	装载累加器低位字，直接或间接寻址 用短立即数装载累加器低位字	0110 1001 IAAA AAAA 1011 1001 IIII IIII	126
LACT dma LACT ind[,ARn]	按TREG规定的左移(0～15)后装入累加器，直接或间接寻址	0110 1011 IAAA AAAA	127
LAR ARx,dma LAR ARx,ind[,ARn] LAR ARx,#k LAR ARx,#lk	用指定的数据单元装载指定的AR，直接或间接寻址 用短立即常数装载指定的AR 用长立即常数装载指定的AR	0000 0ARX IAAA AAAA 1011 0ARX IIII IIII 1011 1111 0000 1ARX+1word	128
LDP dma LDP ind[,ARn] LDP #k	装载数据页指针，直接或间接寻址 用短立即数装载数据页指针	0000 1101 IAAA AAAA 1011 1101 IIII IIII	129
LPH dma LPH ind[,ARn]	装载PREG高位，直接或间接寻址	0111 0101 IAAA AAAA	130
LST #m,dma LST #m,ind[,ARn]	装载状态寄存器ST0 装载状态寄存器ST1	0000 1110 IAAA AAAA 0000 1111 IAAA AAAA	130
LT dma LT ind[,ARn]	装载TREG，直接或间接寻址	0111 0011 IAAA AAAA	131
LTA dma LTA *ind*[,AR*n*]	装载TREG，并累加前次乘积，直接或间接寻址	0111 0000 IAAA AAAA	132
LTD dma LTD ind[,ARn]	装载TREG，累加前次乘积并将被寻址数据移至下一单元，直接或间接寻址	0111 0010 IAAA AAAA	133
LTP dma LTP ind[,ARn]	装载TREG，并将PREG存至累加器，直接或间接寻址	0111 0001 IAAA AAAA	134
LTS dma LTS ind[,ARn]	装载TREG并减去前次乘积，直接或间接寻址	0111 0100 IAAA AAAA	135
MAC pma,dma MAC pma,ind[,ARn]	乘且累加，直接或间接寻址	1010 0010 IAAA AAAA +1word	146
MACD pma,dma MACD pma,ind[,ARn]	乘且累加，并将被寻址数据移至下一单元，直接或间接寻址	1011 0011 IAAA AAAA +1word	147

（续）

语法	说明	操作码	页数
MAR dma MAR ind[,ARn]	修改当前AR和/或ARP，间接（在直接方式下不执行任何操作）	1000 1011 IAAA AAAA	135
MPY dma MPY ind[,ARn] MPY #k	TREG乘以被寻址数据，直接或间接寻址 TREG乘以13位短立即常数	0101 0100 IAAA AAAA 110I IIII IIII IIII	148
MPYA dma MPYA ind[,ARn]	累加前次乘积，再将TREG与被寻址数据相乘，直接或间接寻址	0101 0000 IAAA AAAA	149
MPYS dma MPYS ind[,ARn]	累加前次乘积，再将TREG与被寻址数据相乘，直接或间接寻址	0101 0001 IAAA AAAA	149
MPYU dma MPYU ind[,ARn]	无符号乘，直接或间接寻址	0101 0101 IAAA AAAA	150
NEG	累加器求负	1011 1110 0000 0010	156
NMI	不可屏蔽中断	1011 1110 0101 0010	165
NOP	空操作	1000 1011 0000 0000	169
NORM ind	规格化累加器内容，间接寻址	1010 0000 IAAA AAAA	157
OR dma OR ind[,ARn] OR #lk[,shift] OR #lk,16	被寻址的数据和累加器逻辑“或”，直接或间接寻址 长立即数左移0～15位后和累加器逻辑“或” 长立即数左移16位后和累加器逻辑“或”	0110 1101 IAAA AAAA 1011 1111 1100 SHFT +1word 1011 1110 1000 0010 +1word	158
OUT dma,PA OUT ind,PA[,ARn]	从I/O单元输出数据，直接或间接寻址	0000 1100 IAAA AAAA +1word	136
PAC	PREG装入累加器	1011 1110 0000 0011	136
POP	栈顶弹出至累加器低位	1011 1110 0011 0010	136
POPD dma POPD ind[,ARn]	栈顶弹出至数据存储器，直接或间接寻址	1000 1010 IAAA AAAA	137
PSHD dma PSHD ind[,ARn]	数据存储器数值进栈，直接或间接寻址	0111 0110 IAAA AAAA	138
PUSH	累加器低位进栈	1011 1110 0011 1100	138
RET	从子程序返回	1110 1111 0000 0000	165
RETC con1[,cond2] [,…]	条件返回	1110 11TP ZLVC ZLVC	166
ROL	累加器逻辑循环左移	1011 1110 0000 1100	159
ROR	累加器逻辑循环右移	1011 1110 0000 1101	159

（续）

语　　法	说　　明	操　作　码	页数
RPT dma RPT ind[,ARn] RPT #k	重复下一条指令，直接或间接寻址 重复下一条指令，短立即寻址	0000　1011　IAAA　AAAA 1011　1011　IIII　IIII	169
SACH dma[,shift2] SACH ind[,shift2 [,ARn]]	累加器高位左移后存入数据存储器，直接或间接寻址	1001　1SHF　IAAA　AAAA	159
SACL dma[,shift2] SACL ind[,shift2 [,ARn]]	加器低位左移后存入数据存储器，直接或间接寻址	1001　0SHF　IAAA　AAAA	160
SAR ARx,dma SAR ARx,ind[,ARn]	将指定的AR存储至指定的数据单元，直接或间接寻址	1000　0ARX　IAAA　AAAA	139
SBRK #k	从当前AR中减去短立即常数	0111　1100　IIII　IIII	150
SETC control bit	置C位 置CNF位 置INTM位 置OVM位 置SXM位 置TC位 置XF位	1011　1110　0100　1111 1011　1110　0100　0101 1011　1110　0100　0001 1011　1110　0100　0011 1011　1110　0100　0111 1011　1110　0100　1011 1011　1110　0100　1101	170
SFL	累加器算术左移	1011　1110　0000　1001	160
SFR	累加器算术右移	1011　1110　0000　1010	160
SPAC	从累加器减去PREG	1011　1110　0000　0101	151
SPH dma SPH ind[,ARn]	存储PREG高位，直接或间接寻址	1000　1101　IAAA　AAAA	139
SPL dma SPL ind[,ARn]	存储PREG低位，直接或间接寻址	1000　1100　IAAA　AAAA	140
SPLK #lk, dma SPLK #lk, ind[,ARn]	存储长立即数至数据存储器单元，直接或间接寻址	1010　1110　IAAA　AAAA +1word	140
SPM constant	置乘积移位方式	1011　1111　0000　00PM	170
SQRA dma SQRA ind[,ARn]	平方并累加前次乘积，直接或间接寻址	0101　0010　IAAA　AAAA	151
SQRS dma SQRS ind[,ARn]	平方并减去前次乘积，直接或间接寻址	0101　0011　IAAA　AAAA	151
SST #m,dma SST #m,ind[,ARn]	存储状态存储器ST0，直接或间接寻址 存储状态存储器ST1，直接或间接寻址	1000　1110　IAAA　AAAA 1000　1111　IAAA　AAAA	140

（续）

语　　法	说　　明	操　作　码	页数

SUB dma[,shift]
SUB ind,[,shift[,ARn]]
SUB dma,16
SUB ind,16[,ARn]
SUB #k
SUB #lk[,shift]
	左移 0～15 位后从累加器中减去，直接或间接寻址
	左移 16 位后从累加器中减去，直接或间接寻址
	从累加器中减去短立即数
	长立即数左移 0～15 位后从累加器中减去
		0011　SHFT　IAAA　AAAA
		0110　0101　IAAA　AAAA
		1011　1010　IIII　IIII
		1011　1111　1010　SHFT+1word
			152

SUBB dma SUBB ind[,ARn]	带借位从累加器减，直接或间接寻址	0110　0100　IAAA　AAAA	153
SUBC dma SUBC ind[,ARn]	条件减，直接或间接寻址	0000　1010　IAAA　AAAA	154
SUBS dma SUBS ind[,ARn]	抑制符号扩展从累加器减，直接或间接寻址	0110　0110　IAAA　AAAA	154
SUBT dma SUBT ind[,ARn]	TREG 指定移位数(0～15)，从 ACC 减去直接或间接寻址	0110　0111　IAAA　AAAA	155
TRAP	软件陷阱中断	1011　1110　0101　0001	166
TBLR dma TBLR ind[,ARn]	表读，直接或间接寻址	1010　0110　IAAA　AAAA	141
TBLW dma TBLW ind[,ARn]	表写，直接或间接寻址	1010　0111　IAAA　AAAA	142
XOR dma XOR ind[,ARn] XOR #lk[,shift] XOR #lk,16	被寻址的数据和累加器逻辑"异或"，直接或间接寻址 长立即数左移 0～15 位后和累加器逻辑"或" 长立即数左移 16 位后和累加器逻辑"或"	0110　1100　IAAA　AAAA 1011　1111　1101　SHFT+1word 1011　1110　1000　0011+1word	161
ZALR dma ZALR ind[,ARn]	累加器低位置零并带舍入装载累加器高位，直接或间接寻址	0110　1000　IAAA　AAAA	162

参 考 文 献

1 TMS320F/C24x DSP Controllers CPU and Instruction Set Reference Guide. Texas Instruments, 1999

2 TMS320F/C240 DSP Controllers Peripheral Library And Specific Devices, Texas Instruments, 1999

3 张芳兰等编. TMS320C2XX 用户指南. 北京:电子工业出版社, 1999

4 TMS320C2X User's Guide. Texas Instruments, 1990

5 TMS320C5X User's Guide. Texas Instruments, 1991

6 Digital Signal Processing Applications with the TMS320 Family: Theory Algorithms and Implementations, Volume 2. Texas Instruments, 1990

7 Digital Signal Processing Applications with the TMS320 Family: Theory Algorithms and Implemmentations, Volume 3. Texas Instruments, 1990

8 *TMS320C2X/C5X Optimizing C Compiler User's Guide. Texas Instruments*, 1994

9 TMS320 Floating-point DSP Optimizing C Compiler User's Guide. Texas Instruments, 1994