DSP 控制器原理及应用

宁改娣　杨拴科　编著

科 学 出 版 社

北　京

内 容 简 介

DSP的应用范围已日益广泛，许多用MCU的领域也积极地更新换代为DSP。DSP控制器是一种适合于控制系统的DSP芯片。本教材以TMS320C24x为模型介绍其结构、指令系统及应用；书中介绍了DSP的开发流程和开发环境，提供实验内容，带领读者进入一个完整的学习DSP的环境，同时给出了一些通过实验调试成功的应用实例。此外还简单介绍了MATLAB软件，并给出用MATLAB实现DSP算法仿真的实例。

本书简单明了、易学易用，应用举例全部通过调试和验证。本书可作为学习DSP控制器的本科生和研究生教材，也适用于高等院校相关专业师生和科技人员阅读参考。

图书在版编目(CIP)数据

DSP控制器原理及应用/宁改娟，杨拴科编著. 北京：科学出版社，2002
ISBN 7-03-010909-0

Ⅰ.数… Ⅱ.①宁…②杨… Ⅲ.数字信号-信号处理-微处理器
Ⅳ.TN911.72

中国版本图书馆CIP数据核字(2002)第081451号

责任编辑：赵卫江/责任校对：包志虹
责任印制：吕春珉/封面设计：飞天创意

出版
北京东黄城根北街16号
邮政编码：100717
http://www.sciencep.com
印刷
科学出版社发行 各地新华书店经销
*
2002年11月第 一 版 开本：787×1092 1/16
2006年8月第三次印刷 印张：17 3/4
印数：6 001—7 000 字数：400 000

定价：26.00元

(如有印装质量问题，我社负责调换〈环伟〉)

前　言

DSP 一方面是指数字信号处理技术，另一方面是指数字信号处理器。数字信号处理器即 DSP 芯片的诞生，使得数字信号处理技术的理论研究成果广泛应用到实际系统中，成为数字信号处理技术和应用之间的桥梁，并进一步促进数字信号处理技术的深入发展和应用领域的拓宽。本教材主要介绍数字信号处理器。

在微电子技术发展的带动下，DSP 芯片功能日益强大，性能价格比不断提高，开发环境日臻完善，应用领域不断扩大。在步入数字化时代的进程中扮演着举足轻重的角色。

虽然早在 20 世纪 80 年代中期，DSP 芯片在国内已有许多应用，但直到最近几年，各大专院校才有相应的课程设置，而且许多院校开设的课程都是以 TI 的 TMS320C5000 系列芯片为模型机进行介绍，教材较少。介绍 TMS320C2000 系列及其应用的教材更是凤毛麟角。本教材就是为了方便教学和应用 DSP 控制器的科技人员参考为目的而编写的。

全书共分 8 章，主要包括 3 个部分。

第一部分为基础部分，包括第 1～3 章和第 5～6 章。主要介绍 DSP 芯片的发展、特点，TI 公司 TMS320 DSP 各系列特点以及数字基础，以 TMS320C24X DSP 控制器为模型介绍了其指令系统和硬件结构，在对硬件深入介绍的同时还给出部分模块的扩展方法。本部分还详细介绍了 DSP 指令系统的寻址方式，对各种指令只给出分类列表，而对每条指令的理解和使用，我们给大家提供了一个学习的途径和方法。

第二部分为应用开发部分，包括第 4 章和第 8 章。主要介绍 DSP 开发流程、开发工具的使用及实验。在第一部分基础上，介绍 DSP 的开发环境（软件 CCS 和 PCI 仿真卡），使得开发者能够很快熟悉开发环境，缩短开发周期。根据以往的教学经验和学生的反馈意见，对于学习微处理器芯片及其应用类课程，实验是非常重要的一个环节。因此，在第 8 章中提供了大量的实验内容。部分实验给出了通过调试的源程序。

第三部分（第 7 章）为算法仿真，DSP 的主要特点是适用于数字信号处理。在具体实现 DSP 算法以前，一般首先要对其进行模拟仿真以判断其正确性、复杂性和可靠性。以前 DSP 算法一般用 C 语言或其他高级语言来模拟实现，但模拟过程繁琐、调试不便。目前这种模拟可以用 MATLAB 语言快速方便地实现，仿真结果可以转化为 C 语言。另外 MATLAB 还可以与 TI 的 DSP 仿真环境通讯，但这部分有待大家一起探索。由此可见，借助 MATLAB 将会非常有利于数字信号处理的实现。因此，本书对 MATLAB 仿真软件做简单介绍，并给出通过调试的算法仿真程序。

本书由宁改娣和杨拴科编写，第一、三章由杨拴科编写；其余部分由宁改娣编写。在编写过程中，参阅了不少国内外参考书及资料，学习和吸取了不少经验。在此向这些作者致以谢意！在书稿的录入过程中，刘涛、杜倩宁、何永威、雍亮、廉海涛等同志参与了少部分工作，在此表示感谢！另外，书中一些实例是由选修 DSP 课程的部分研究生提供，陶坤宇工程师也对本书提出了许多建议，在此一并表示感谢！本教材能得以出版还要特别感谢樊捷和李洁的支持。本书由申忠如教授主审，并提出了许多宝贵意见，在此表示衷心的

感谢。

我校 DSP 课程的开设以及本书的出版也得到电气工程学院各位领导及电工电子教学中心领导和同事们的大力支持和关心，在此表示诚挚的谢意。

由于时间仓促，加之作者的水平和掌握的资料有限，书中错误和不当在所难免，恳请读者批评指正。

作　者

2002 年 6 月于西安交通大学

目　　录

第三章　寻址方式与指令系统

本章首先介绍′C2××的寻址方式，然后讨论其指令集、汇编指令，并简要介绍宏指令。

3.1　寻址方式

获得操作数的方式称为寻址方式。′C2××有三种寻址方式：立即寻址、直接寻址和间接寻址。后两种为存储器寻址方式，即指令的操作数是在数据存储器中。

在立即寻址中，操作数在指令代码中直接以常数给出。

当用户访问数据存储器时，可采用直接寻址或间接寻址。直接寻址将指令字代码中的7位偏移地址与数据存储器页指针(DP)的9位组合起来形成16位数据存储器地址。间接寻址通过8个16位辅助寄存器访问数据存储器。以下分别介绍三种寻址方式。

3.1.1　立即寻址方式

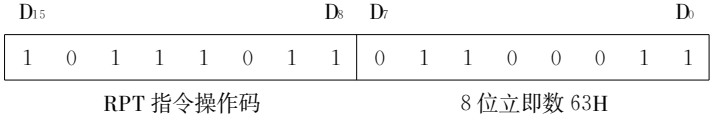
在立即寻址方式下，指令字里包含了一个供该指令安排的常数。与其他微处理器一样，在指令中该常数可以用十进制、二进制(B)和十六进制(H)来表示，十六进制最高位大于9时要在数字前面加0与程序的标号相区分。另外，立即数前面必须加“#”。′C2××支持以下两种立即寻址方法。

1. 短立即数寻址

使用短立即数寻址的指令时，指令代码中用一个8bit、9bit或13bit的常数作为操作数。短立即数指令是一个单指令字，常数包含在该指令字中。

例如，采用短立即寻址的RPT指令：

```
RPT     #63H        ;将紧跟其后的指令重复执行100次
```



2. 长立即数寻址

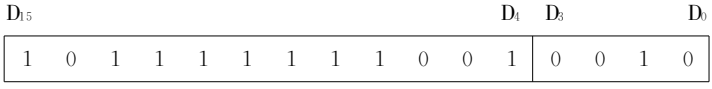
使用长立即数寻址的指令用一个16bit的常数作为操作数且是双指令字。该常数是指令代码的第二个字。该16bit的值可以是一个绝对值常数，或是一个二进制补码值。

例如，ADD指令

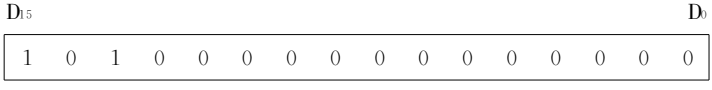
```
ADD     #0A000H,2    ;将立即数A000H左移2位后与累加器内容相加结果存累
```

;加器中

第一个字



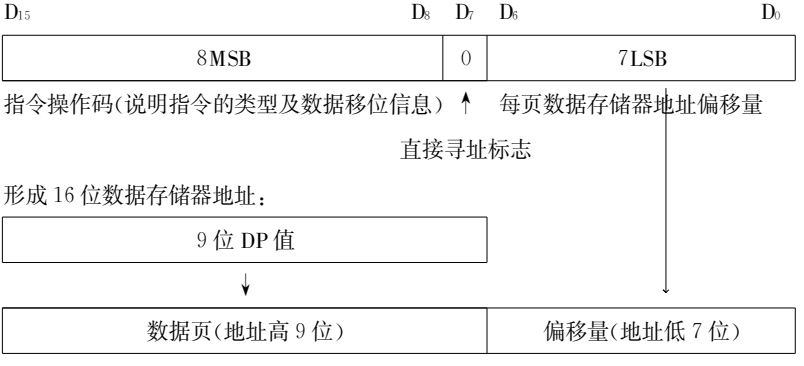
第二个字



3.1.2 直接寻址方式

直接寻址方式是较常用的一种存储器寻址方式,可以访问 64K 字数据存储器。将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,标注为 0～511。每页 128 个字空间。当前数据页是由状态寄存器 ST0 中的一个 9bit 的数据页指针(DP)来决定的。例如,如果 DP 的值是 000000000B,则当前数据页指向第 0 页;如果 DP 的值是 000000010B,则当前数据页指向第 2 页。确定当前数据页后,该数据页的 128 个字的具体位置(即页内偏移地址)由直接寻址方式指令给出,指令代码的低 7 位就指定了每页内寻址字的偏移量。直接寻址方式的指令代码和 16 位数据存储器地址形成过程如下:

直接寻址指令代码:



因此,在使用直接寻址方式时,首先为页指针寄存器 DP 装入一个适当的数(从 0 到 511)。例如,下述汇编语句使用页指针装入指令 LDP,将数据页设置为 32(即地址为 1000h～107Fh):

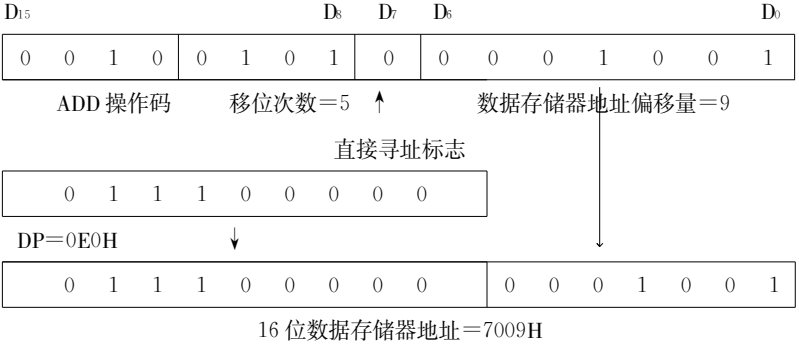
```
LDP     #32
```

然后,9 位的 DP 内容与指令代码中的 7 位偏移值拼在一起就确定了惟一的数据存储器字地址。例如,

```
LDP     #0E0H   ;DP 指向 E0H 数据页,地址为 7000H～707FH
ADD     9H,5    ;该指令将数据存储器 7009H 单元的内容左移 5 位后与 ACC 相
```

;加,结果存 ACC,指令代码及地址形成过程如下:

直接寻址指令代码:



16 位数据存储器地址=7009H

在使用直接寻址方式时,对当前数据页指针 DP 要特别注意以下几点:

• DSP 复位时并没有初始化 DP 内容,虽然 DSP 仿真开发环境可以复位 DP 为 0,但程序中要访问数据存储器前还应该用 LDP 指令确定当前数据页。

• 如果程序需要访问不同的数据页,则每当访问新的数据页前,必须先修改 DP 值,以确保访问正确的数据页。当然,程序中如果只访问同一个数据页,只需要在程序前段装载 DP 一次即可。

3.1.3　间接寻址方式

DSP 内部的 8 个辅助寄存器(AR0～AR7)和辅助寄存器算术运算单元(ARAU)提供了功能强大而灵活的间接寻址操作方式。用 16 位辅助寄存器内容作为间接的地址,可以访问 64K 数据存储空间中的任一单元,不受当前数据页的限制。

8 个辅助寄存器都可以作为间接寻址寄存器,但每次寻址只能使用其中一个。为了选择一个特定的辅助寄存器(AR0～AR7),将一个 0～7 的值装入状态寄存器 ST0 中的辅助寄存器指针(ARP)。ARP 所指向的辅助寄存器称为当前 AR,而当前 AR 的内容,就是要访问的数据存储器的地址。辅助寄存器算术运算单元(ARAU)根据指令的要求可以对辅助寄存器的内容进行运算,并将运算结果地址送到数据读地址总线(DRAB)或数据写地址总线(DWAB)。然后,再根据指令的要求,通过 ARAU 来修改当前辅助寄存器指针 ARP 的值。

'C2×× 提供 4 种间接寻址方式:

*——不增量不减量。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,当前辅助寄存器的值不改变。

+或−——加 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值加 1 或减 1。

*0+或*0−——增加或减去一个指定的量(或称为变址量)。辅助寄存器 AR0 存指定的量。指令将当前辅助寄存器的值作为数据存储器地址,指令执行之后,将当前辅助寄存器的值增加或减去 AR0 的内容,即指定量。

*BR0+或*BR0−——使用逆向进位增加或减去一个指定的量。辅助寄存器 AR0

存指定的量。当前辅助寄存器的值作为数据存储器的地址使用之后，当前辅助寄存器的值增加或减去 AR0 保存的变址量。但该加法或减法是逆向进位(用于 FFT)，即加减法是从高位开始运算，向低位进位或借位。

上述 4 种运算均由辅助寄存器算术运算单元(ARAU)在流水线指令译码的同一个周期内完成，因此，DSP 的间接寻址速度非常快。这 4 种间接寻址类型实际上提供了如表 3.1 所列的 7 种间接寻址。

表 3.1 间接寻址及其操作数

说明	操作数符号	举 例
不增量不减量	*	LT *；将当前 AR 所指定的数据存储器的值装入暂时寄存器(TREG)
加 1	*+	LT *+；将当前 AR 所指定的数据存储器的值装入暂时寄存器(TREG)，当前 AR 的值加 1
减 1	*-	LT *-；将当前 AR 所指定的数据存储器的值装入暂时寄存器(TREG)，当前 AR 的值减 1
加指定量	*0+	LT *0+；将当前 AR 所指定的数据存储器的值装入暂时寄存器(TREG)，将 AR0 的值加给当前 AR
减指定量	*0-	LT *0-；将当前 AR 所指定的数据存储器的值装入暂时寄存器(TREG)，将当前 AR 的值减去 AR0 的值
逆向进位加指定量	*BR0+	LT *BR0+；将当前 AR 所指定的数据存储器的值装入暂时寄存器(TREG)，将 AR0 的值逆向进位加给当前 AR
逆向进位减指定量	*BR0-	LT *BR0-；将当前 AR 所指定的数据存储器的值装入暂时寄存器(TREG)，将当前 AR 的值逆向进位减去 AR0 的值

间接寻址举例：

例 1：MAR *，AR2 ；选择 AR2 作为当前辅助寄存器，即 ARP=2

例 2：LAR AR2，#2000H ；将立即数 2000H→AR2

例 3：ADD *+，8 ；当前辅助寄存器的内容左移 8 位后加到 ACC，并将当前辅助寄存器内容加 1

例 4：ADD *BR0+，AR5 ；当前辅助寄存器所指存储单元的内容加到 ACC，并将当前辅助寄存器内容与 AR0 ；的内容按反向进位加，且指定该指令结束后 AR5 为当前辅助寄存器。

3.2 'C2××的指令集

'C2×× 具有强大的指令集，支持信号处理运算和通用目的的应用。'C2×× 的指令集和'C2×的指令集兼容；为'C2×写的代码可以在'C2×× 上重新编译和运行。而'C5×的指令集则是'C2×× 的超集，因此为'C2×× 写的代码可以在'C5×上运行。

'C2×× 的指令按功能可分为以下 6 种类型：

(1) 累加器，算术与逻辑指令

(2) 辅助寄存器和数据页指针指令

(3) 暂时寄存器(TREG)、乘积寄存器(PREG)和乘法指令

(4) 转移指令

(5) 控制指令

(6) I/O 和存储器操作

限于篇幅,这里只能列出指令集中的指令及其简要描述。表 3.2 至表 3.7 分别列出了这几组指令。

表 3.2　累加器,算术与逻辑指令

助记符	指　令　描　述	指令字	周期
ABS	ACC 的绝对值	1	1
ADD	加给 ACC,带 0～15bit 移位,直接或间接	1	1
	加给 ACC,带 0～15bit 移位,长立即数	2	2
	加给 ACC,带 16bit 移位,直接或间接	1	1
	加给 ACC,短立即数	1	1
ADDC	加给 ACC,带进位,直接或间接	1	1
ADDS	加给 ACC 低段,带符号展开抑制,直接或间接	1	1
ADDT	加给 ACC,由 TREG 决定移位,直接或间接	1	1
AND	ACC 与数据值,直接或间接	1	1
	ACC 与长立即数,带 0～15bit 移位	2	2
	ACC 与长立即数,带 16bit 移位	2	2
CMPL	ACC 取补	1	1
LACC	带移位 0～15bit 装入 ACC,直接或间接	1	1
	带移位 0～15bit 装入 ACC,长立即数	2	2
	带移位 16bit 装入 ACC,直接或间接	1	1
LACL	装入 ACC 的低段,直接或间接	1	1
	装入 ACC 的低段,短立即数	1	1
LACT	装入 ACC,由 TREG 低 4 位(0～3bit)决定移位(0～15bit),直接或间接	1	1
NEG	ACC 取负	1	1
NORM	ACC 内容归一化,间接	1	1
OR	ACC 或数据值,直接或间接	1	1
	ACC 或长立即数,带 0～15bit 移位	2	2
	ACC 或长立即数,带 16bit 移位	2	2
ROL	ACC 循环左移	1	1
ROR	ACC 循环右移	1	1
SACH	存高段 ACC,带移位 0～7bit,直接或间接	1	1
SACL	存低段 ACC,带移位 0～7bit,直接或间接	1	1
SFL	ACC 左移	1	1
SFR	ACC 右移	1	1
SUB	从 ACC 减,带移位 0～15bit,直接或间接	1	1
	从 ACC 减,带移位 0～15bit,长立即数	2	2
	从 ACC 减,带移位 16bit,直接或间接	1	1
	从 ACC 减,短立即数	1	1
SUBB	从 ACC 带借位减,直接或间接	1	1

续表

助记符	指 令 描 述	指令字	周期
SUBC	条件减,直接或间接	1	1
SUBS	从 ACC 减,抑制符号展开,直接或间接	1	1
SUBT	从 ACC 减,由 TREG 决定的移位(0～15bit),直接或间接	1	1
XOR	ACC 异或数据值,直接或间接	1	1
	ACC 异或长立即数,带 0～15bit 移位	2	2
	ACC 异或长立即数,带 16bit 移位	2	2
ZALR	ACC 低段置 0,舍入后装入 ACC 高段,直接或间接	1	1

表 3.3 辅助寄存器指令

助记符	指 令 描 述	指令字	周期
ADRK	常数加给 AR,短立即数	1	1
BANZ	当前 AR 非 0 转移,间接	2	2
CMPR	当前 AR 与 AR0 比较	1	1
LAR	从指定的数据位置装入指定的 AR,直接或间接	1	2
LAR	常数装入指定的 AR,短立即数	1	2
	常数装入指定的 AR,长立即数	2	2
MAR	修改当前 AR 和/或 ARP,间接(直接时无操作)	1	1
SAR	存指定的 AR 至指定位置,直接或间接	1	1
SBRK	从当前 AR 减去常数,短立即数	1	1

表 3.4 暂时寄存器(TREG)、乘积寄存器(PREG)和乘法指令

助记符	指 令 描 述	指令字	周期
APAC	PREG 加到 ACC	1	1
LPH	装入 PREG 高位	1	1
LT	装入 TREG,直接或间接	1	1
LTA	装入 TREG,累加前次乘积,直接或间接	1	1
LTD	装入 TREG,累加前次乘积,搬移数据,直接或间接	1	1
LTP	装入 TREG,存 PREG 入 ACC,直接或间接	1	1
LTS	装入 TREG,减去前次乘积,直接或间接	1	1
MAC	乘且累加,直接或间接	2	3
MACD	乘且累加,数据转移,直接或间接	2	3
MPY	TREG 乘数据值,直接或间接	1	1
	TREG 乘 13bit 常数,短立即数	1	1
MPYA	乘且累加前次乘积,直接或间接	1	1
MPYS	乘且减去前次乘积,直接或间接	1	1
MPYU	乘无符号数,直接或间接	1	1

续表

助记符	指　令　描　述	指令字	周期
PAC	PREG 装入 ACC	1	1
SPAC	从 ACC 减去 PREG	1	1
SPH	存高段 PREG,直接或间接	1	1
SPL	存低段 PREG,直接或间接	1	1
SPM	设置乘积移位方式	1	1
SQRA	平方且累加前次乘积,直接或间接	1	1
SQRS	平方且减去前次乘积,直接或间接	1	1

表 3.5　转移指令

助记符	指　令　描　述	指令字	周期
B	无条件转移,间接	2	4
BACC	转移至 ACC 指定的地址	1	4
BANZ	当前 AR 非 0 时转移,间接;判断(AR),(AR－1)→(AR)	2	4
BCND	条件转移	2	4
CALA	调用 ACC 指定位置的子程序,间接	2	4
CALL	调用子程序,间接	2	4
CC	条件调用	2	4
INTR	软中断	1	4
NMI	不可禁止的中断	1	4
RET	从子程序返回	1	4
RETC	条件返回	1	4
TRAP	软件中断	1	4

表 3.6　控制指令

助记符	指　令　描　述	指令字	周期
BIT	位测试,直接或间接	1	1
BITT	由 TREG 指定的位测试,直接或间接	1	1
CLRC	清除 C 位	1	1
	清除 INTM 位	1	1
	清除 OVM 位	1	1
	清除 SXM 位	1	1
	清除 TC 位	1	1
	清除 XF 位	1	1
IDLE	停止执行,直至中断	1	1
LDP	装入数据页指针,直接或间接	1	2
	装入数据页指针,短立即数	1	2

续表

助记符	指 令 描 述	指令字	周期
LST	装入状态寄存器 ST0,直接或间接	1	2
	装入状态寄存器 ST1,直接或间接	1	2
NOP	无操作	1	1
POP	将堆栈顶弹出至 ACC 低段	1	1
POPD	将堆栈顶弹出至数据存储器,直接或间接	1	1
PSHD	将数据存储器的值压入堆栈,直接或间接	1	1
PUSH	将 ACC 低段压入堆栈	1	1
RPT	重复执行下一条指令,直接或间接	1	1
	重复执行下一条指令,短立即数	1	1
SETC	设置 C 位	1	1
	设置 CNF 位	1	1
	设置 INTM 位	1	1
	设置 OVM 位	1	1
SETC	设置 SXM 位	1	1
	设置 TC 位	1	1
	设置 XF 位	1	1
SPM	设置乘积移位模式	1	1
SST	存状态寄存器 ST0,直接或间接	1	1
	存状态寄存器 ST1,直接或间接	1	1

表 3.7 I/O和存储器指令

助记符	指 令 描 述	指令字	周期
BLDD	数据块从数据存储器搬移至数据存储器,直接或间接,长立即数作源存储器地址	2	3
	数据块从数据存储器搬移至数据存储器,直接或间接,长立即数作目标存储器地址	2	3
BLPD	数据块从程序存储器搬移至数据存储器,直接或间接,长立即数作源存储器地址	2	3
DMOV	数据在数据存储器中搬移,直接或间接	1	1
IN	从 I/O 输入数据,直接或间接	2	2
OUT	输出数据至口,直接或间接	2	3
SPLK	存长立即数至数据存储器,直接或间接	2	2
TBLR	读数据表,直接或间接	1	3
TBLW	写数据表,直接或间接	1	3

如果要查每条指令的详细内容,可以在 TI 网站寻找相应 DSP 的指令系统资料。'C24×的指令系统对应文献为 SPRU160C。另外,开发软件 CCS 的帮助菜单也提供了每条指令的详细内容。这些资料对每条指令都给出了详细说明。下面以 ADDC 指令为例说明:

1. 指令格式(即语法)

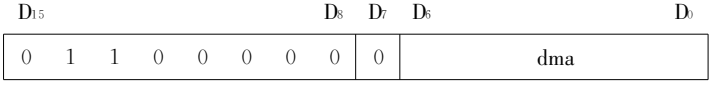
ADDC *dma*　　;直接寻址
ADDC *ind*[,AR*n*]　　;间接寻址
对语法要作以下说明:
语法的黑体符号如 **ADDC** 和 **AR** 是不能修改的,必须原样录入;
语法的斜体符号如 *dma*、*n* 和 *ind* 是可以由用户需要改变的;
语法中方扩号[ ]内的内容为可选项,而[]外的不管是斜体,还是黑体都是指令不可缺少的。

2. 操作数

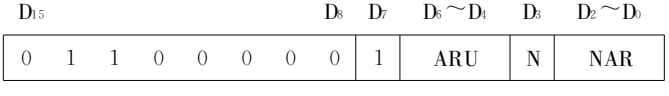
dma　　;直接寻址时,数据存储器地址低 7 位
n　　;指示下一个辅助寄存器 *n* =0～7
ind　　;选择下列 7 种间接寻址方式中的一种:
　　*　　*+　　*−　　*0+　　*0−　　*BR0+　　*BR0−
汇编指令一般有 0 到 2 个操作数,操作数可以是常数、表达式、I/O 端口、寄存器地址和各类其他常数。

3. 操作码

ADDC dma



ADDC ind[,ARn]



其中,ARU 为辅助寄存器更新代码。确定当前辅助寄存器是否变化以及如何增减,见表 3.8。

表 3.8

ARU(D_6～D_4)	当前辅助寄存器 AR 上执行的算术运算
0 0 0	当前 AR 不变
0 0 1	当前 AR−1→当前 AR
0 1 0	当前 AR+1→当前 AR
0 1 1	保留
1 0 0	当前 AR−AR0→当前 AR(反向进位减)
1 0 1	当前 AR−AR0→当前 AR
1 1 0	当前 AR+AR0→当前 AR
1 1 1	当前 AR+AR0→当前 AR(反向进位加)

N 为下一个辅助寄存器指示符。本位声明该指令是否改变 ARP 的值。

N=0 ARP 内容保持不变；

N=1 NAR 内容被装入 ARP,且旧的 ARP 值被装入状态寄存器 ST1 的辅助寄存器缓冲器 ARB 中。

NAR 为下一个辅助寄存器的代码值。AR0～AR7 的对应代码分别为 0～7。

4. 执行

Increment PC, then ...

(ACC)+(data－memory address)+(C)→ACC

执行部分提供了一条指令执行所发生的指令操作时序。若执行动作取决于寻址方式,则执行部分将规定相应于某一寻址方式下的动作。下面是用于执行部分的一些常用注释符号：

(r) 表示寄存器或存储单元 r 的内容,例如,(ACC)表示累加器中的数值。

x→y 数值 x 被赋于寄存器或存储单元 y。

r(n:m) 寄存器或存储单元 r 的位 n 到位 m。例如 ACC(7:0)表示累加器的位 7 到位 0。

(r(n:m)) 寄存器或存储单元 r 的位 n 到位 m 的内容。

5. 状态位

影响本指令的标志位：OVM　　受本指令影响的标志位：C 和 OV

该指令不受 SXM 位影响。

CPU 状态寄存器 ST0 和 ST1 中的位,影响指令的操作,也受一些指令的影响。每条指令的状态位部分指明哪些位影响指令执行,哪些位受指令影响。

6. 说明

这部分解释指令执行过程中发生的动作以及对处理器其余部分或存储器的影响。是对执行部分的补充说明。ADDC 指令的说明部分如下：

The contents of the addressed data-memory location and the value of the carry bit are added to the accumulator with sign extension suppressed. The carry bit is then affected in the normal manner: the carry bit is set (C = 1) if the result of the addition generates a carry and is cleared (C = 0) if it does not generate a carry.

The ADDC instruction can be used in performing multiple－precision arithmetic.

7. 指令代码字数

该部分说明了本指令的机器代码数量,单位为字。

8. 周期

每条指令周期说明中分别给出单次执行和重复执行(与 RPT 指令连用)时,在给定存

储器配置情况下，指令执行所需的处理器机器周期数。单次执行 ADDC 指令的周期数见表 3.9。

表 3.9

指令操作数所在位置	程序所在位置			
	ROM	DARAM	SARAM	外部程序存储器
DARAM	1	1	1	1+p
SARAM	1	1	1,2+	1+p
外部数据存储器	1+d	1+d	1+d	2+d+p

注：+：如果操作和操作码在同 SARAM 模块；p 为程序存储器等待状态数；d 为数据存储器等待状态数。

重复(RPT)执行 ADDC 指令 n 次的周期数见表 3.10。

表 3.10

指令操作数所在位置	程序所在位置			
	ROM	DARAM	SARAM	外部程序存储器
DARAM	n	n	n	n+p
SARAM	n	n	d,n+1+	n+p
外部数据存储器	n+nd	n+nd	n+nd	n+1+nd+p

注：+ 如果操作和操作码在同 SARAM 模块。

9. 举例

```
ADDC    DAT300   ;(DP = 6: addresses 0300h-037Fh;
                 ;设 DAT300 为已赋值 300H 的常数)
```

指令执行前		指令执行后	
数据存储器 300H	0004H	数据存储器 300H	0004H
ACC 1	0013H	ACC 0	0018H
C		C	

3.3 ′C2××的伪指令

汇编语言包括指令性语句和伪指令语句。指令性语句就是上面介绍的用各种助记符表示的机器指令，每条指令都有其机器代码或指令代码；伪指令语句(汇编指令)是指示性语句。简称伪指令。恰当地使用伪指令，将能极大地方便我们应用汇编语言进行编程。伪指令一般不产生指令代码。这类指令在汇编过程中与汇编程序“通信”，说明源程序的起止、分段情况、安排各类信息的存储结构以及有关的变量说明等。具体实现以下任务：

(1) 将数据和代码汇编进特定的段
(2) 为初始化的变量保留存储器空间
(3) 展开列表的形式
(4) 汇编条件块
(5) 定义全局变量
(6) 指定汇编器可以获得宏的定义库
(7) 检查符号调试信息

以下分别介绍分段定义、常数初始化、段程序计数器排列、输出列表格式、引用其他文件、条件汇编、汇编时的符号以及其他伪指令：

1. 段定义指令

.asect——asect 初始化的段具有绝对地址。一个用.asect 来定义的段可包括代码或数据。asect 指令对于将片外程序代码存储器的内容引导到片内快速存储器非常有用(在一个绝对段中,可以使用.label 指令来定义浮动标志)。

.bss——为未初始化的变量保留空间。一般存放于数据存储器区域。

.data——定义已知数据段,它通常包含初始化了的数据。由命令文件(参见第 4.4 节)可以将定义的数据存放于程序或数据存储器中。

.sect——定义初始化后命名的段,并设置序列代码或数据与之相联系。常用于定义中断向量表。

.text——确认.text 段中的代码部分,它通常包含可执行的代码。一般存放与程序存储器区域。

.usect——在一个未命名的段中保留空间。该指令与.bss 相类似,但它允许保留与.bss段不同的空间。

由命令文件分配到同一存储器区域(如,B0、B1、B2、外部数据存储器及程序存储器等)的不同段,在存储器中的分配顺序与其在命令文件中的先后顺序有关,在命令文件中先定义的段,其内容放在存储器前面部分,其余依次存放。

2. 常数初始化指令

.bes 和.space——在当前段内保留一个特定的 bit 数。汇编器将这些保留的 bit 填 0。当使用带有.bes 的标志时,它指向保留 bit 的第一个字;而使用带有.space 的标志时,它指向保留 bit 的最后一个字。

.byte——将一个或多个 8bit 的值置入当前段中的连续的字中。该指令类似于.word,不同之处在于每个值的宽度限于 8bit。

.field——将一个值置入当前字的特定数量的 bit。

.float 和.bfloat——计算单精度 32bit IEEE 浮点数的值,并将其存入当前段的两个连续的字中。.bfloat 指令保证目标不会越过数据页的边界。

.int 和.word——将一个或多个 16bit 值置入当前段的连续字中。

.long 和.blong——将 32bit 值置入当前段的连续字中。.blong 指令保证目标不会越过数据页的边界。

.sring——从一个或多个字符串中将 8bit 字符置入当前段。该指令类似于.byte,不同之处在于它将两个字符打包置入每个字。

段定义指令和常数定义的部分伪指令是较常用的伪指令,在第 4.4 节还会有进一步介绍。下面举例说明其使用及汇编器对段定义伪指令的处理。

以下是使用伪指令编写工程 testd 的源程序(testd.asm)后,汇编器产生的对应列表文件。

```
C:\TDS-EMU2XX\C2000\CGTOOLS\BIN\DSPA.EXE testd.asm testd.obj -ls -v2xx
TMS320C1×/C2×/C2××/C5× COFF Assembler Version 7.00 Tue Jun 4 09:16:38 2002
Copyright (c) 1987-1999 Texas Instruments Incorporated
testd.asm                                                              PAGE    1
1                ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
2                ;* * * assemble an initialized table into .data * *
3                ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
4  0000                    .data
5  0000  0011   coeff .word 011h, 022h, 033h
   0001  0022
   0002  0033
6                ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
7                ;* * * reserve space in .bss for two variables * *
8                ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
9  0000            .bss var1,1
10 0001            .bss buffer,10
11               ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
12               ;* * * still in data * *
13               ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
14 0003  0123   ptr .word 0123h
15               ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
16               ;* * * assemble code into .text section * *
17               ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
18 0000          .text
19 0000  100f    add:LAC 0Fh
20 0001  bfa0    aloop: SBLK 1
   0002  0001
21 0003  e3cc                BLEZ aloop
   0004  0001'
22 0005  9000                SACL var1,0
23               ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
24               ;* * * assemble another initialized table into * *
25               ;* * * the .data section * *
26               ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
27 0004                  .data
```

```
1    2     3                                                    4
28   0004  00aa  ivals .word 0AAh, 0BBh
     0005  00bb
29               ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
30               ;* * * define the named section "newvars" * *
31               ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
32   0000        var2 .usect "newvars",1
33   0001        inbuf .usect "newvars",7
34               ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
35               ;* * * assemble more code into .text * *
36               ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
37   0006                .text
38   0006  b8ff        ADD #0FFh
```

第2列域 第3列域

```
39               END
```

第1列域 第4列域

第1列域内容为对应源程序的行号；

第2列域内容为各段的字数计数器；

第3列域内容为目标代码，此时目标代码地址是浮动的，存储器中所存内容为链接后的代码。

第4列域内容为源程序描述。

以上程序共产生了4个段：

.text 包含7个字的目标代码。分别为100F、BFA0、0001、E3CC、0001、9000、B8FF。

.data 包含6个字的目标代码。分别为0011、0022、0033、0123、00AA、00BB。

.bss 在存储区为变量保留了11个字单元。

newvars 由 usect 指令命名的段，它在存储器中为变量保留了8个字单元。

在没有命令文件时，链接器(参见第四章)对各段有一个缺省分配模式，当使用缺省分配模式时，链接器首先读目标文件(*.obj)，确定目标DSP处理器类型，然后根据DSP类型分配各段。如果为TMS320C2XX系列DSP，则上述的testd工程的存储器分配模式如图3.1所示。链接器从外部程序存储区域1000H开始，先分配.text段存储空间，占空间大小由程序目标代码多少确定。然后为.data段分配空间。并从片内的存储器模块B1(地址为0300H～03FFH)及外部数据存储器区域开始对.bss段分配存储区域。任何未初始化的命名段依次(在链接器输入文件中的次序)存放在.bss段后。如usect定义newvars段紧跟.bss后。链接器分配好各段存储器空间之后，由开发软件CCS的载入器将对应段内容存放到存储器。

3. 段程序计数器排列指令

.align——在一个128字的范围排列SPC(在汇编器中为各段提供了一个段程序计数器，section program counters，简称SPC)。它保证该指令后面的代码从一个数据页范围内开始。即以页为边界分配段。

.even——以偶数字为边界分配段。

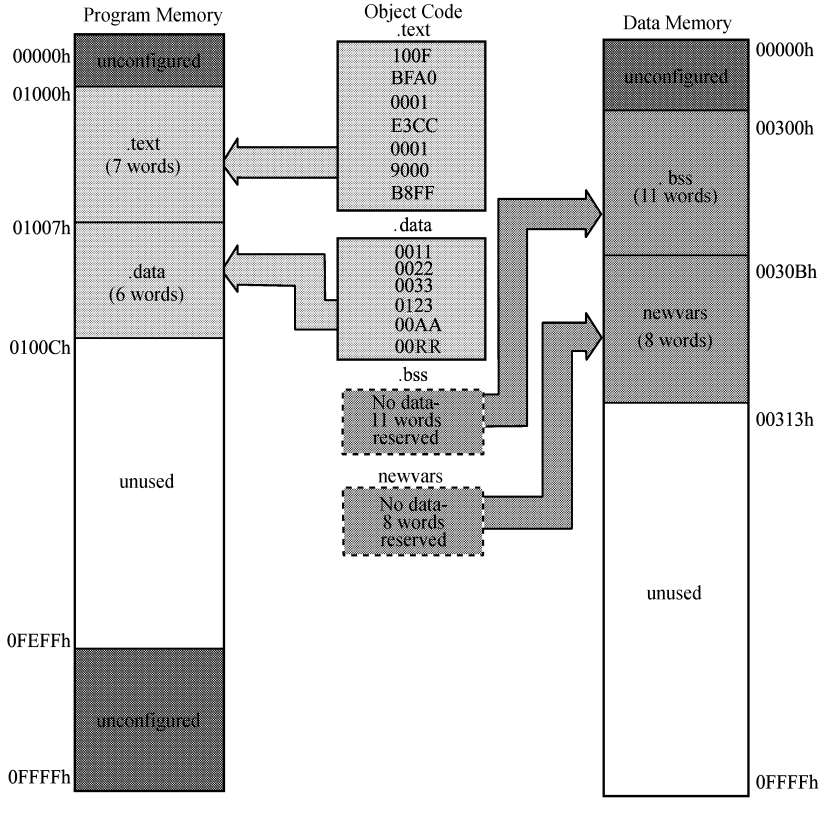


图 3.1　没有使用命令文件时各段在存储器中的默认分配(图中以 testd 工程为例)

4. 输出列表格式指令

.drlist 和 .drnolist——将指令行加入/不加入列表文件。
.fclist 和 .fcnolist——列表/不列表假条件块(false condition block)。
.length——控制列表文件的页长度。
.list 和 .nolist——作/不作输出列表。
.mlist 和 .mnolist——允许/禁止宏展开和循环块列表。
.option——控制列表文件的一些特性。该指令包括下述操作数:

- B 将 .byte 指令的列表限制于一行。
- D 将 .int 和 .word 指令的列表限制于一行。
- F 复位 B ,D ,L , M 和 T 指令。
- L 将 .long 指令的列表限制于一行。
- M 在列表中关闭宏展开。
- T 将 .string 指令的列表限制于一行。
- X 产生一个符号的交叉引用列表。

.page——在输出列表中分页。

.sslist/.ssnolist——允许/禁止替换符号展开列表。

.tab——定义 tab 的大小。

.title——为汇编提供一个打印在每页顶部的标题。

.width——展开列表文件的页宽度。

5. 引用其他文件的指令

.copy 和.include——告诉汇编器开始从其他文件读入源语句。从被 copy 文件读入的源语句将在列表文件中列出,从被 include 文件读入的源语句不会在列表文件中列出。

.def——确认在当前模块中定义而又可以由其他模块使用的符号。汇编器将该符号列入符号表。

.global——将一个符号说明为外部的,以便在连接时可以为其他模块使用。

.mlib——为汇编提供一个包括有宏定义的归档库的名字。

.ref——确认在当前模块中使用但是在其他模块中定义的符号。汇编器将该符号标志为未定义的外部符号,放入目标符号表,以便连接器可以解决其定义。

6. 条件汇编指令

.if/.elseif/.else/.endif——告诉汇编器按照表达式的值来有条件地汇编代码块。

.loop/.break/.endloop——告诉汇编器按照表达式的值来重复地汇编代码块。

7. 汇编时的符号指令

.asg——将一个字符串赋给一个替换符号。其值存在替换符号表中。当汇编器处理一个替换符号时,就将该符号用其字符串的值来替换。

.set/.equ——为一个符号设置一个常数值。该符号存在符号表中,不能再定义。

.struct/endstruct——设置类似于 C 的结构定义。

.tag——将结构特性赋予一个标志。

.eval——计算一个表达式,将其值转化为一个字符,并将该字符串赋予一个替换符号。

.newblock——复位局部标志。局部标志是形式为 $ n(n 是一个十进制数)的符号。它们是在标志域内定义的。局部标志是暂时性的标志,可以用作跳转指令的操作码。

8. 其他方面的指令

.end——终止汇编。它是一个程序的最后一条源语句。

.label——定义一个特定的符号代表当前段中的装入地址。

.mmregs——为存储映射的寄存器定义符号名字。

.port——接通汇编器的口开关。

.sblock——为分块指定段。分块是一种类似于分页的地址排列机制,但要弱一些。一个分块的段如果小于一页,则保证不跨越一页的范围。分块的段如果大于一页,则从一页的范围内开始。

.version——告诉汇编器该代码是属于哪一代处理器。

.emsg——将出错信息送给标准输出器件。

.mmsg——将汇编时的信息送给标准输出器件。

.wmsg——将警告信息送给标准输出器件。

3.4　宏　指　令

宏汇编属于高级汇编语言技术，允许在汇编程序中引进一系列具有“宏”功能的伪指令。在这种情况下，就可以将一段具有独立功能而且重复调用的源程序预定义为一个宏，用一个名字代表这段程序。而将定义的宏在程序中就像汇编指令一样在需要时调用，从而可以简化和缩短源程序。

TMS320 系列 DSP 的汇编器支持宏指令，使用户能够建立自己的“指令”。在需要将一个特定的任务执行若干次时，宏指令尤其有用。其功能包括：

- 定义自己的宏，重新定义已有的宏；
- 简化长的或复杂的汇编代码；
- 访问由归档器建立的宏库；
- 在一个宏内定义条件的和可循环的块；
- 在一个宏内作串操作；
- 控制展开列表。

使用宏的过程如下所述。

1. 定义宏

在程序中使用宏之前，必须首先定义，可以用以下的两种方法来定义宏：

(1) 宏可以在源文件起始处，或者在.include/.copy 文件中定义。

(2) 宏也可以在宏库中定义。

宏定义格式为：

```
宏名字          .macro（形式参数）
                …（宏定义体，即一段汇编程序）
                .endm
```

例如，编写一个对数据存储单元某位置 1 的宏指令如下：

```
SBIT1     .macro  DMA,MASK  ;置位宏
          MAR   *,AR1
          LAR AR1,#DMA
          LACC  *
          OR    #MASK
          SACL  *
          .endm
```

在程序中定义了宏之后，就可以在源程序中，将宏的名字作为指令来调用这个宏。例如，在程序中如果希望将 0200H 单元的 D_4 位置 1，则可以在程序中调用宏：

```
SBIT1 0200H,0010H
```

第八章　DSP 应用(实验)

8.1　实验一:熟悉实验软、硬件环境

1. 实验目的

熟悉开发软件 CC/CCS(Code Composer)的使用;并通过第 4 章介绍的 LEDS.ASM 源程序了解 EVM 板上资源及程序的完整调试过程,掌握 DSP 的汇编语言编程及调试方法。

2. 实验要求

按照第 4 章介绍的方法建立一个新工程,并调试程序。熟悉 CCS 提供的调试工具。

3. 实验步骤及内容

根据第 4 章的介绍安装好开发环境,并了解了第 4 章的软件开发流程后,按第 4 章第 5 节内容新建工程并汇编链接和调试程序。本实验要求读者通过简单的实例熟悉使用开发环境的全过程。

8.2　实验二:中断编程方法(以捕获单元为例)

1. 实验目的

进一步熟悉 CC 环境和 DSP 汇编编程,了解 DSP 中断过程,学习 DSP 中断编程方法。了解捕获单元中断及用捕获单元实现测量信号频率或两个上升沿之间的时间。

2. 实验要求

利用捕获单元的中断方式测试某一信号的频率。

3. 实验原理

利用系统已知频率的信号,比如将 CLKOUT/IOPC1 编程为时钟输出,使 DSP 的 CLKOUT 引脚输出一已知频率信号(当输入频率为 10MHz 时,WDCLK 一般为 15625Hz),并将捕获单元 1 的输入引脚 CAP1/QEP1 设置为捕获输入,将 EVM 板上 P4 控制信号接口的 CLKOUT/IOPC1 与 P1 的 CAP1/QEP1/IOPC4 相连。让通用定时器 GP1 定时 1 秒,CPUCLK 确定后,设定 GP1 的周期寄存器,允许周期匹配中断(中断优先级高于捕获单元中断);允许捕获单元中断。捕获单元测试输入引脚 CAP1/QEP1/IOPC4 上输入信号的上升沿,每来一个上升沿中断一次并使计数变量 COUNTER 加 1;定时 1 秒到,定时中断服务程序读 COUNTER 值得到频率并存储。但为了避免捕获单元第一次捕

获同步误差较大,第一次得到的 COUNTER 数据要丢弃,再重复几次测试并取其平均值,得到比较准确的结果。(以下实验例程中,捕获中断不必处理 CAPFIFO 的内容。)

4. 实验例程[①] (已通过调试)

```
;Measuring the Frequency of an Input Square Wave Using the TMS320F240 EVM
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; File Name: cap1.asm
; Originator: Digital control systems Apps group — Houston
; Target System: 'C24x Evaluation Board
; Description: Capture Input of the Event Manager Module is set up
; to count the number of rising edges that an input square wave has in a 1
;second period To view the results in the debugger environment, The value
;in FREQ is the number of rising edges that occur within a 1 second period.
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  .include f240regs.h
;————————————————————————————————————————————
; Vector address declarations
;————————————————————————————————————————————
  .sect ".vectors"
RSVECT      B START    ; Reset Vector
INT1        B PHANTOM  ; Interrupt Level 1
INT2        B PERIOD_ISR ; Interrupt Level 2
INT3        B PHANTOM ; Interrupt Level 3
INT4        B CAP_ISR ; Interrupt Level 4
INT5        B PHANTOM ; Interrupt Level 5
INT6        B PHANTOM ; Interrupt Level 6
RESERVED B PHANTOM ; Reserved
SW_INT8 B PHANTOM ; User S/W Interrupt
SW_INT9 B PHANTOM ; User S/W Interrupt
SW_INT10 B PHANTOM ; User S/W Interrupt
SW_INT11 B PHANTOM ; User S/W Interrupt
SW_INT12 B PHANTOM ; User S/W Interrupt
SW_INT13 B PHANTOM ; User S/W Interrupt
SW_INT14 B PHANTOM ; User S/W Interrupt
SW_INT15 B PHANTOM ; User S/W Interrupt
SW_INT16 B PHANTOM ; User S/W Interrupt
TRAP B PHANTOM ; Trap vector
NMINT B PHANTOM ; Non—maskable Interrupt
EMU_TRAP B PHANTOM ; Emulator Trap
```


① 详细内容参考 TI 提供的文件 SPRA416。

```
SW_INT20 B PHANTOM ; User S/W Interrupt
SW_INT21 B PHANTOM ; User S/W Interrupt
SW_INT22 B PHANTOM ; User S/W Interrupt
SW_INT23 B PHANTOM ; User S/W Interrupt
;=========================================
; MAINCODE - starts here
;=========================================
   .text
   NOP
START:
   LDP   #0
   SETC  INTM              ;Disable interrupts
   SPLK  #000Ah,IMR        ;Mask all core interrupts except INT4 and INT2
   LACC  IFR               ;Read Interrupt flags for clear it
   SACL  IFR               ;Clear all interrupt flags
   CLRC  SXM               ;Clear Sign Extension Mode
   CLRC  OVM               ;Reset Overflow Mode
   CLRC  CNF               ;Config Block B0 to Data mem
;-----------------------------------------
; Set up PLL Module
;-----------------------------------------
   LDP   #00E0h
   SPLK #0000000001000001b,CKCR0    ;CLKMD=PLL; Disable,SYSCLK=CPUCLK/2
   SPLK #0000000010111000b,CKCR1    ;CLKIN(OSC)=10MHz,CPUCLK=5MHz
   SPLK #0000000011000001b,CKCR0    ;CLKMD=PLL Enable
   SPLK #0100000001000000b,SYSCR    ;CLKOUT=WDCLK
   SPLK #006Fh,WDCR                 ;Disable WD if VCCP=5V (JP5 in pos. 2-3)
   KICK_DOG                         ;Reset Watchdog
;-----------------------------------------
; Set up Digital I/O Port
;-----------------------------------------
   LDP     #225            ;DP=225(0E1H), Data Page to Configure OCRx
   SPLK #0011100000000000b,OCRA  ;OCRA - Output Control Register A
        ; Bit 15 (0) CRA.15 - IOPB7   ; Bit 14 (0) CRA.14 - IOPB6
        ; Bit 13 (1) CRA.13 - T3PWM/T3CMP   ; Bit 12 (1) CRA.12 - T2PWM/T2CMP
        ; Bit 11 (1) CRA.11 - T1PWM/T1CMP   ; Bit 10 (0) CRA.10 - IOPB2
        ; Bit 9 (0) CRA.9 - IOPB1   ; Bit 8 (0) CRA.8 - IOPB0
        ; Bits 7-4 (0000)Reserved   ; Bit 3 (0) CRA.3 - IOPA3
        ; Bit 2 (0) CRA.2 - IOPA2   ; Bit 1 (0) CRA.1 - IOPA1
        ; Bit 0 (0) CRA.0 - IOPA0
   SPLK #11110000b,OCRB   ;OCRB - Output Control Register B
        ; Bit 7 (1) CRB.7 - CAP4   ; Bit 6 (1) CRB.6 - CAP3
```

```
        ; Bit 5 (1) CRB.5 — CAP2/QEP2   ; Bit 4 (1) CRB.4 — CAP1/QEP1
        ; Bit 3 (0) CRB.3 — BIO   ; Bit 2 (0) CRB.2 — XF
        ; Bit 1 (0) CRB.1 — Reserved   ; Bit 0 (0) CRB.0 — IOPC0
;*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—;—
;Event Manager Module Reset
;This section resets all of the EV Registers. This is ;necessary for
; silicon revision 1.1, but not for silicon revisions 2.0 and later.
;*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*
  LDP   #232              ;DP=232 Data Page for the Event Manager
  SPLK #0000h,GPTCON          ;Clear General Purpose Timer Control
  SPLK #0000h,T1CON           ;Clear GP Timer 1 Control
  SPLK #0000h,T2CON           ;Clear GP Timer 2 Control
  SPLK #0000h,T3CON           ;Clear GP Timer 3 Control
  SPLK #0000h,COMCON          ;Clear Compare Control
  SPLK #0000h,ACTR            ;Clear Full Compare Action Control Register
  SPLK #0000h,SACTR           ;Clear Simple Compare Action Control Register
  SPLK #0000h,DBTCON          ;Clear Dead—Band Timer Control Register
  SPLK #0000h,CAPCON          ;Clear Capture Control
  SPLK #0FFFFh,EVIFRA         ;Clear Interrupt Flag Register A
  SPLK #0FFFFh,EVIFRB         ;Clear Interrupt Flag Register B
  SPLK #0FFFFh,EVIFRC         ;Clear Interrupt Flag Register C
  SPLK #0000h,EVIMRA          ;Clear Event Manager Mask Register A
  SPLK #0000h,EVIMRB          ;Clear Event Manager Mask Register B
  SPLK #0000h,EVIMRC          ;Clear Event Manager Mask Register C
;*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—;—
;End of RESET section for silicon revision 1.1 *
;*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—*—;
; Set up Event Manager Module
;————————————————————————————
T1PERIOD .set 9895h ; T1Period Initialized to 1Hz value
;时间常数与 CPUCLK 频率 fCPUCLK、定标系数 k、工作方式、定时时间 t 有关
;对于连续增计数模式周期 T=(n+1)TCLK= (n+1)TCPLCLK * 定标系数=(n+1)1/(5 * 10^6) * 128=1,
;所以计数值 n=9895H
  .text
  LDP   #232          ;DP=232, Data Page for Event Manager Addresses
  SPLK   #0000001010101b,GPTCON ;GPTCON — GP Timer Control Register
        ; Bit 15 (0) T3STAT — GP Timer 3 Status. READ ONLY
        ; Bit 14 (0) T2STAT — GP Timer 2 Status. READ ONLY
        ; Bit 13 (0) T1STAT — GP Timer 1 Status. READ ONLY
        ; Bits 12—11 (00) T3TOADC — ADC start by event of GP Timer 3
        ; No event starts ADC
        ; Bits 10—9 (00) T2TOADC — ADC start by event of GP Timer 2
```

```
        ; No event starts ADC
        ; Bits 8—7 (00) T1TOADC — ADC start by event of GP Timer 1
        ; No event starts ADC
        ; Bit 6 (1) TCOMPOE — Compare output enable
        ; Enable all three GP timer compare outputs
        ; Bits 5—4 (01) T3PIN — Polarity of GP Timer 3 compare output
        ; Active Low
        ; Bits 3—2 (01) T2PIN — Polarity of GP Timer 2 compare output
        ; Active Low
        ; Bits 1—0 (01) T1PIN — Polarity of GP Timer 1 compare output
        ; Active Low
    SPLK #T1PERIOD,T1PR ; T1PR = 9895h
    SPLK #0000h,T1CNT      ; Initialize Timer 1
    SPLK #0000h,T2CNT      ; Initialize Timer 2
    SPLK #0000h,T3CNT      ; Initialize Timer 3
    SPLK #0001011100000110b,T1CON ;T1CON — GP Timer 1 Control Register
        ; Bits 15—14 (00) FREE,SOFT — Emulation Control Bits
        ; Stop immediately on emulation suspend
        ; Bits 13—11 (010) TMODE2 — TMODE0 — Count Mode Selection
        ; Continuous—Up Count Mode
        ; Bits 10—8 (111) TPS2 — TPS0 — Input Clock Prescaler
        ; Divide by 128,计数频率 = CPUCLK/128
        ; Bit 7 (0) Reserved
        ; Bit 6 (0) TENABLE — Timer Enable
        ; Disable timer operations
        ; Bits 5—4 (00) TCLKS1,TCLKS0 — Clock Source Select Internal Clock Source
        ; Bits 3—2 (01) TCLD1,TCLD0 — Timer Compare Register Reload Condition
        ; When counter is 0 or equals period register value
        ; Bit 1 (1) TECMPR — Timer compare enable
        ; Enable timer compare operation
        ; Bit 0 (0) Reserved
    SPLK #0000000000000000b,T2CON      ;GP Timer 2—Not Used
        ;T2CON — GP Timer 2 Control Register
        ; Bits 15—14 (00) FREE,SOFT — Emulation Control Bits
        ; Stop immediately on emulation suspend
        ; Bits 13—11 (000) TMODE2—TMODE0 — Count Mode Selection Stop/Hold
        ; Bits 10—8 (000) TPS2—TPS0 — Input Clock Prescaler Divide by 1
        ; Bit 7 (0) TSWT1 — GP Timer 1 timer enable bit Use own TENABLE bit
        ; Bit 6 (0) TENABLE — Timer Enable Disable timer operations
        ; Bits 5—4 (00) TCLKS1,TCLKS0 — Clock Source Select
        ; Internal Clock Source
        ; Bits 3—2 (00) TCLD1,TCLD0 — Timer Compare Register Reload Condition
        ; When counter is 0
```

```
    ; Bit 1 (0) TECMPR — Timer compare enable
    ; Disable timer compare operation
    ; Bit 0 (0) SELT1PR — Period Register select
    ; Use own period register
SPLK #0000000000000000b,T3CON      ;GP Timer 3—Not Used
    ;T3CON — GP Timer 3 Control Register
    ; Bits 15—14 (00) FREE,SOFT — Emulation Control Bits
    ; Stop immediately on emulation suspend
    ; Bits 13—11 (000) TMODE2—TMODE0 — Count Mode Selection
    ; Stop/Hold
    ; Bits 10—8 (000) TPS2—TPS0 — Input Clock Prescaler
    ; Divide by 1
    ; Bit 7 (0) TSWT1 — GP Timer 1 timer enable bit
    ; Use own TENABLE bit
    ; Bit 6 (0) TENABLE — Timer Enable
    ; Disable timer operations
    ; Bits 5—4 (00) TCLKS1,TCLKS0 — Clock Source Select
    ; Internal
    ; Bits 3—2 (00) TCLD1,TCLD0 — Timer Compare Register Reload Condition
    ; When counter is 0
    ; Bit 1 (0) TECMPR — Timer compare enable
    ; Disable timer compare operation
    ; Bit 0 (0) SELT1PR — Period Register select
    ; Use own period register
SPLK #1010010001010101b,CAPCON     ;CAPCON — Capture Control Register
    ; Bit 15 (1) CAPRES — Capture Reset
    ; No action
    ; Bits 14—13 (01) CAPQEPN — Capture Units 1 & 2 and QEP Circuit Control
    ; Enable Capture Units 1 & 2. Disable QEP Circuit
    ; Bit 12 (0) CAP3EN — Capture Unit 3 Control
    ; Disable Capture Unit 3
    ; Bit 11 (0) CAP4EN — Capture Unit 4 Control
    ; Disable Capture Unit 4
    ; Bit 10 (1) CAP34TSEL — GP Timer Selection for Capture Units 3 & 4
    ; Select GP Timer 3
    ; Bit 9 (0) CAP12TSEL — GP Timer Selection for Capture Units 1 & 2
    ; Select GP Timer 2
    ; Bit 8 (0) CAP4TOADC — Capture Unit 4 starts ADC
    ; No Action
    ; Bits 7—6 (01) CAP1EDGE — Edge Detection for Capture Unit 1
    ; Detect Rising Edge
    ; Bits 5—4 (01) CAP2EDGE — Edge Detection for Capture Unit 2
    ; Detect Rising Edge
```

```
        ; Bits 3—2 (01) CAP3EDGE — Edge Detection for Capture Unit 3
        ; Detect Rising Edge
        ; Bits 0—1 (01) CAP4EDGE — Edge Detection for Capture Unit 4
        ; Detect Rising Edge
    SPLK #01111111b,CAPFIFO      ;CAPFIFO — Capture FIFO Status Register
        ; Bits 15—14 CAP4FIFO Status — READ ONLY
        ; Bits 13—12 CAP3FIFO Status — READ ONLY
        ; Bits 11—10 CAP2FIFO Status — READ ONLY
        ; Bits 9—8 CAP1FIFO Status — READ ONLY
        ; Bit 7 (1) CAPFIFO15 — CAP4FIFO bit 15 Clear
        ; Clear Bit 15 of CAPFIFO
        ; Bit 6 (1) CAPFIFO14 — CAP4FIFO bit 14 Clear
        ; Clear Bit 14 of CAPFIFO
        ; Bit 5 (1) CAPFIFO13 — CAP3FIFO bit 13 Clear
        ; Clear Bit 13 of CAPFIFO
        ; Bit 4 (1) CAPFIFO12 — CAP3FIFO bit 12 Clear
        ; Clear Bit 12 of CAPFIFO
        ; Bit 3 (1) CAPFIFO11 — CAP2FIFO bit 11 Clear
        ; Clear Bit 11 of CAPFIFO
        ; Bit 2 (1) CAPFIFO10 — CAP2FIFO bit 10 Clear
        ; Clear Bit 10 of CAPFIFO
        ; Bit 1 (1) CAPFIFO9 — CAP1FIFO bit 9 Clear
        ; Clear Bit 9 of CAPFIFO
        ; Bit 0 (1) CAPFIFO8 — CAP1FIFO bit 8 Clear
        ; Clear Bit 8 of CAPFIFO
    SPLK #00010000000b,EVIMRA    ;EVIMRA — EV Interrupt Mask Register A
        ; Bits 15—11 Reserved
        ; Bit 10 (0) T1OFINT ENABLE — Timer 1 Overflow Interrupt Enable
        ; Disabled
        ; Bit 9 (0) T1UFINT ENABLE — Timer 1 Underflow Interrupt Enable
        ; Disabled
        ; Bit 8 (0) T1CINT ENABLE — Timer 1 Compare Interrupt Enable
        ; Disabled
        ; Bit 7 (1) T1PINT ENABLE — Timer 1 Period Interrupt Enable
        ; Enabled
        ; Bit 6 (0) SCMP3INT ENABLE — Simple Compare Unit 3 Comp IntEnable
        ; Disabled
        ; Bit 5 (0) SCMP2INT ENABLE — Simple Compare Unit 2 Comp InEnable
        ; Disabled
        ; Bit 4 (0) SCMP1INT ENABLE — Simple Compare Unit 1 Comp IntEnable
        ; Disabled
        ; Bit 3 (0) CMP3INT ENABLE — Full Compare Unit 3 Comp Int Enable
        ; Disabled
```

```
        ; Bit 2 (0) CMP2INT ENABLE — Full Compare Unit 2 Comp IntEnable
        ; Disabled
        ; Bit 1 (0) CMP1INT ENABLE — Full Compare unit 1 Comp IntEnable
        ; Disabled
        ; Bit 0 (0) PDPINT ENABLE — Power Drive Protection InterruptEnable
        ; Disabled
    SPLK #0001b,EVIMRC      ;EVIMRC — EV Interrupt Mask Register C
        ; Bits 15—4 Reserved
        ; Bit 3 (0) CAP4INT Enable
        ; Disable
        ; Bit 2 (0) CAP3INT Enable
        ; Disable
        ; Bit 1 (0) CAP2INT Enable
        ; Disable
        ; Bit 0 (1) CAP1INT Enable
        ; Enable
;————————————————————————————————————————
; VARIABLES FOR CAP- ISR
;————————————————————————————————————————
    .bss VALUE1,1 ;1st timer value for 1st period ; interrupt
    .bss VALUE2,1           ;2nd timer value for 2nd period ; interrupt
    .bss VALUE3,1           ;3rd timer value for 3rd period ; interrupt
    .bss VALUE4,1           ;4th timer value for 4th period ; interrupt
    .bss VALUE5,1           ;5th timer value for 5th period ; interrupt
    .bss COUNTER,1          ;Counter to acquire 5 values
    .bss FREQ,1             ;Frequency to count the number of rising edges
    .bss COUNTS,1           ;Counts the number of rising edges in 1 second
    .text
    LAR AR1,#VALUE1         ;AR1 = address of VALUE1
    LAR AR2,#COUNTER        ;AR2 = address of COUNTER
    LDP #0
    SPLK #0000h,VALUE1      ;Initialize VALUE1
    SPLK #0000h,VALUE2      ;Initialize VALUE2
    SPLK #0000h,VALUE3      ;Initialize VALUE3
    SPLK #0000h,VALUE4      ;Initialize VALUE4
    SPLK #0000h,VALUE5      ;Initialize VALUE5
    SPLK #0005h,COUNTER     ;Counter set to acquire 5 values
    SPLK #0000h,FREQ        ;Initialize FREQ
    SPLK #0000h,COUNTS      ;Initialize COUNTS
    LDP #232
    LACC EVIFRC             ;ACC = Interrupt Flags of EVIFRC
    SACL EVIFRC             ;EVIFRC = ACC => clears all flags
    LDP #0
```

```
    LACC IFR                ;ACC = Interrupt Flags of IFR
    SACL IFR                ;IFR = ACC => clears all flags
    LDP #232
    SBIT1 T1CON,B6-MSK      ;Sets Bit 6 of T1CON
          ;TxCON — GP Timer x Control Register
          ; Bit 6 (1) TENABLE — Timer Enable
          ; Enable Timer Operations
    MAR *,AR1 ;ARP = AR1
    CLRC INTM ;Enable Interrupts
WAIT B WAIT ;Wait for an interrupt
; PERIOD INTERRUPT SERVICE ROUTINE
PERIOD-ISR:
    KICK-DOG                ;Clear Watchdog Counter
    LDP#0                   ;DP = 0 for addresses 0000h — 007Fh
    LACC COUNTS             ;ACC = Counts
    SACL *+,0,AR2           ;Save COUNTS to VALUEx, ARP = AR2
    LACC #0                 ;ACC = 0
    SACL COUNTS             ;Clear the Counts
    LACC *                  ;ACC = COUNTER
    SUB #1                  ;Decrement Counter
    SACL *,0,AR1            ;Store new value of Counter; ARP = AR1
    BCND LEAVE,EQ           ;If captured all values stop else restart
    LDP #232                ;DP = 232 for address 7400h — 747Fh
    LACC EVIVRA             ;Reading Vector Register Clears Interrupt Flags
    CLRC INTM               ;Enable Interrupts
    RET                     ;Return to program
LEAVE LDP #0                ;DP = 0, Data Page for the acquired values
    LACC VALUE2             ;ACC = VALUE2
    ADD VALUE3              ;ACC = VALUE2 + VALUE3
    ADD VALUE4              ;ACC = VALUE2 + VALUE3 + VALUE4
    ADD VALUE5              ;ACC = VALUE2 + VALUE3 + VALUE4 +VALUE5
    SFR                     ;Shift ACC right = Divide by 2
    SFR                     ;Shift ACC right = Divide by 2
    SACL FREQ               ;Store value into FREQ
STOP KICK-DOG
        B STOP              ;End the Program
CAP-ISR:                    ; CAPTURE INTERRUPT SERVICE ROUTINE
    LDP #0                  ;DP = 0 for addresses 0000h — 007Fh
    LACC COUNTS             ;ACC = COUNTS
    ADD #1                  ;ACC = COUNTS + 1; Increment Counts
    SACL COUNTS             ;COUNTS = ACC; Store new value
    LDP #232                ;DP = 232 for addresses 7400h — 747Fh
    LACC EVIVRC             ;Reading Vector Register clears Interrupt Flags
```

```
    CLRC INTM                ;Enable Interrupts
    RET                      ;Return from interrupt
;=====================================
; I S R - PHANTOM
; Description: Dummy ISR, used to trap spurious interrupts.
; Modifies: Nothing
;=====================================
PHANTOM KICK_DOG ;Resets WD counter
        B PHANTOM
```

程序调试时一定注意:如果 EVM 板上的 JP5 在 1－2 位置(即 HP0＝0),则 Watchdog 始终使能,即使程序中使 WDCR.6(即 WDDIS 位)＝1,也不能禁止 Watchdog 的工作,也就是说 HP0＝0 时,软件是无法关闭 Watchdog 的。一旦 Watchdog 被使能,惟一正确的做法就是在 Watchdog 复位之前在程序适当的地方加上清除 Watchdog 计数器的宏命令 KICK_DOG。

特别注意:即使在如下的结束程序等待指令中也一定要加 KICK_DOG,否则 Watchdog 会不断地复位系统,影响执行结果。

```
STOP    KICK_DOG
        B STOP          ;End the Program
```


8.3　实验三:数字振荡器的实现

1. 实验目的

熟悉 CCS 环境及编程方法,进一步了解 DSP 中断过程,学习 DSP 中断编程方法。了解数字振荡器的设计原理,加深对 DSP 指令系统的了解。

2. 实验要求

要求利用原理中任一方法,设计一正弦波发生器,并从示波器观察结果。估算所设计的正弦波信号的频率范围。

3. 实验原理

在通信、仪器和控制等领域的信号处理系统中,可能会用到正弦波发生器。实现数字振荡器有多种方法,以下列出几种设计原理供参考。

(1)实现正弦波发生器的方法

• 方法一 ——查表法

利用 TMS320F/C2xx 来实现一个数字振荡器,可以使用查表法,即将正弦信号 0～2π 各弧度的值等分计算其函数值并建表,以等时间间隔(时间不同则输出信号频率不同)将该值送到 DAC 芯片。这种方法无须计算,但需要以大量程序存储器空间作为代价。

• 方法二 ——傅里叶级数展开法

使用傅里叶级数展开法。这也是一种有效的计算正弦量的方法。与查表法相比,它

需要的存储单元少，而且精度高。

一个角度为 θ 的正弦和余弦函数，都可以展开成傅里叶级数，取其前五项进行近似：

$$\sin\theta = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} = x\left\{1 - \frac{x^2}{2*3}\left\{1 - \frac{x^2}{4*5}\left\{1 - \frac{x^2}{6*7}\left\{1 - \frac{x^2}{8*9}\right\}\right\}\right\}\right\}$$

$$\cos\theta = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} = 1 - \frac{x^2}{2}\left\{1 - \frac{x^2}{3*4}\left\{1 - \frac{x^2}{5*6}\left\{1 - \frac{x^2}{7*8}\right\}\right\}\right\}$$

上式中的 x 为 θ 的弧度值。

由傅里叶级数展开法的原理可知对于正弦和余弦函数其收敛域为(−1,1)，在应用傅氏级数时要注意到 $x\in(-1,1)$。由于 TMS320C24x 的 DSP 微处理器是采用定点运算来处理小数的，因此我们采用 Q15 来表示(−1,1)区间的 x 值。

十进制数与定点数的转化关系可以表示为：

十进制真值(x)转化为定点数(x_q)：$x_q=(\text{int})\,x*2^Q$

定点数(x_q)转化为十进制真值(x)：$x_q=(\text{float})\;x_q*2^{-Q}$

例如，十进制数 $x=0.5$，定标 Q=15，则定点数 $x_q=0.5*32678=16384=4000\text{H}$。

反之，一个用 Q15 表示的定点数 4000H，其对应的十进制数为 $16384\times2^{-15}=0.5$。

在 DSP 的汇编语言程序中，不能直接写入十进制小数，如果要定义一个小数 0.5，可以写成：

```
.word    32678*707/1000    ;32678表明是Q15格式
```


由于本实验主要处理的是纯小数，以下举例介绍纯小数乘纯小数的定点乘法。即用 Q15*Q15=Q30 的方法。

例如，0.5*0.5=0.25

```
    0.1000000000000000    ; Q15
×   0.1000000000000000    ; Q15
-------------------------------
00.0100000000000000000000000000000=0.25    ;Q30
```

两个 Q15 的小数相乘后得到的双精度数不必全部保留，而只需保留 16 位的单精度数.由于相乘后得到的高 16 位不满足 15 位的小数精度，为了达到 15 位精度，可将乘积左移一位，下面是上述乘法的 TMS320C2XX 程序片断：

```
LT      OP1             ;OP1=4000H(0.5/Q15)
MPY     OP2             ;OP2=4000H(0.5/Q15)
PAC
SACH    ANS,1           ;ANS=2000H(0.25/Q15)
```

计算时将系数 c1=1/(8*9)、c2=1/(6*7)、c3=1/(4*5)、c4=1/(2*3)分别用 Q15 表示法为 01c7H、030bH、0666H 和 1556H。

• 方法三——查表和线性插值法

使用查表和线性插值的方法实现正弦波信号发生器。设 sine look-up table 表中的 index 索引值从 0 ～ 256 分别对应于 angle 角度 0 ～ 360 度，其对应关系是：每个 angle 是对应 index 的 360/256≈1.41 倍；sin(angle)正弦值的范围从−1 至+1 变化，sinval 幅度从−32768至+32768 变化，幅度 sinval 与 sin(angle)对应关系是：sin(angle)*8000h = sin-

val 。

插值算法原理介绍如下:图 8.1 给出了线性插值的图形和具体算式。

$$Y = y_i + \underbrace{\frac{X - x_i}{x_{i+1} - x_i}}_{\text{r=ratio}} (y_{i+1} - y_i)$$

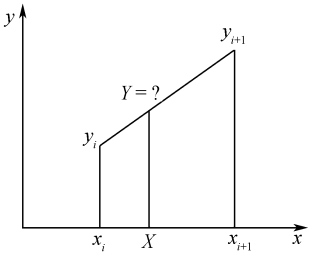


图 8.1　线性插值公式和线性插值法的图形

程序设计思路说明:用三个变量 MODREG、FREQSTEP、MAG 分别表示初相角、步长和峰峰值,分别用于设定初相角,调节频率的大小和获得峰峰值。输出的正弦信号频率除了与 FREQSTEP 的值有关外,还与送出各点的间隔时间有关。

MODREG 的范围是:0000h～ffffh,分别对应于 angle 的 0～360 度,它们的关系是这样:由给定的 MODREG 的值,得到一个索引值 index,由这个索引值 index 去查表,就得到相应的 angle 值,其中计算索引值的公式为:index = MODREG/256。

FREQSTEP 的范围是:0000h～ffffh,分别对应于 angle 的 0～360 度,它们的关系用数字表示如下:若 FREQSTEP 是 1000,则 angle 是 4.22 度,它们的关系也是通过查索引值得到的,其中计算该索引值的公式为:INT[1000/256]=3。

MAG 的范围是:0000h～ffffh,若 MAG 为 7fffh,则幅值 DACVAL 的范围是 0000h～4fffh,在将之压缩到 12 位,则最终幅值 DACVAL 的范围是 0000h～fffh。

在程序设计中,FREQSTEP 赋值:1000,MODREG 赋值:c000h,MAG 赋值:7fffh。程序流程如图 8.2 所示。

• 方法四——递推法

利用 DSP 强大运算功能,计算各弧度正弦值,送 DAC 转换。为了将 DSP 输出的数字正弦波转化为平滑连续的正弦波,还要经过 DAC 和滤波。下面首先简单介绍数字振荡器的原理,然后给出 DSP 实现方法。希望程序能通过改变几个参数就可以得到不同的频率的正弦波。

设需要产生的正弦信号为 $y=\sin(t)$。对其进行 Z 变换,就可以得到用于代替连续正弦信号的离散数字序列递推方程,这样一来,数字系统可以非常容易的计算出这个正弦数字序列。

令 $t=k\omega T$,得到离散正弦信号 $y[k]=\sin(k\omega T)$,对其进行 Z 变换:

$$y(z) = \frac{z\sin\omega T}{z^2 - 2z\cos\omega T + 1} = \frac{Az}{z^2 - Bz + 1}$$

其中 $A=\sin\omega T$, $B=2\cos\omega T$。

设系统的传递函数为

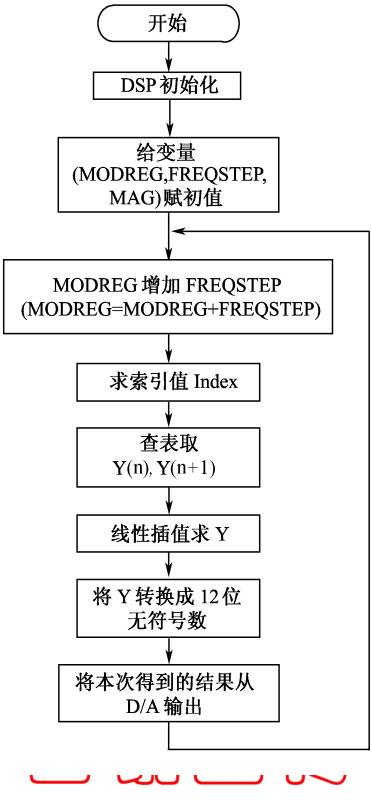


图 8.2 方法三程序流程图

$$H(z)=\frac{Y(z)}{X(z)}$$

当 $x(t)$为单位冲击响应时，上式就是正弦序列 $y[k]=\sin(k\omega T)$的 Z 变换，则

$$H(z)=\frac{Az}{z^2-Bz+1}$$

变换形式为

$$z^2Y(z)-zBY(z)+Y(z)=AzX(z)$$

利用 Z 反变换，并假设初始条件为零，由上式可得差分方程：

$$y[k+2]=By[k+1]-y[k]+Ax[k+1]$$

$x(k)$为单位冲击响应，利用单位冲击响应 $x[k+1]$的性质，即仅当 $k=-1$ 时，$x[k+1]=1$，代入上式得

$$y[0]=0;y[1]=A$$

$$y[k+2]=By[k+1]-y[k]\quad(k\geqslant 0)$$

可见在 $k\geqslant 0$ 的情况下，$y[k+2]$可以用 $y[k+1]$和 $y[k]$计算得到，这是一个递推的差分方程。

以上所得到的公式是理论上的公式。实际上，DSP 采用的是数据位有限的定点运算，所以我们不可能将小数数据准确的表示出来。因此，用定点 DSP 实现时采用以下方法。

如果在一个正弦周期中,采样 360 个数据点(可以采样任意 N 个点,当然采样点数越多,输出越接近正弦波),则正弦数据序列值是以一度为步长的(步长 $\omega T=\frac{360}{N}$度),即

$$y[0]=\sin 0^\circ \quad y[1]=A=\sin \omega T=\sin 1^\circ \quad y[2]=\sin 2^\circ \cdots y[n]=\sin n^\circ \cdots$$

定点运算器不能计算小数,所以可以给公式的两端同时乘以 2^{14},(这里采用的是 14 位定点计算),并对小数部分四舍五入。这样前述公式变为:

$$y[0]=0;\ y[1]=286;\quad B=2\cos \omega T\times 2^{14}=2\cos 1^\circ \times 2^{14}=32763=7\text{FFBH}$$

$$y[k+2]=32763\times y[k+1]/2^{14}-y[k] \quad (k\geqslant 0)$$

因为在 DSP 芯片中,许多指令带有附加的左移功能,所以对公式进行一些调整,两边同时乘以 2^{14},然后进行数据计算,最后将 ACC 再左移两位,结果就存储在 ACC 的高位字中(因为以上数据处理对实际数据共计左移 16 位,所以 ACC 高位字即为运算结果),即

$$y[k+2]=[(7\text{FFBH}\times y[k+1]-y[k]\times 2^{14})\times 2^{2}]/2^{16} \quad (k\geqslant 0) \qquad (8.0)$$

这样一来,将递推的每个点由定时器定时送到 DAC,则可以得到一个正弦信号,该数字振荡器的输出信号频率 f_s 是可以调整的,其频率由定时器定时时间 T 和采样点数 N 确定,$f_s=\frac{1}{NT}$。

(2)正弦输出

模数接口电路是 DSP 处理系统中一个重要的组成部分。一般的 A/D 或 D/A 芯片均采用并行的数字接口,这些芯片与 TMS320C240 接口时须设计一定的译码电路,将转换芯片映射到 DSP 芯片的 I/O 口地址,TMS320C240 用 I/O 指令(IN 和 OUT)与模数接口芯片交换数据。

图 8.3 是一种 TMS320C240 评估板与 DAC7624 构成的正弦波发生器的硬件电路图。其中 DAC7624 是一个 12 位的四路电压输出的数模转换器,它接收 12 位并行的输入数据,有双缓冲的 DAC 输入逻辑,并提供一个内部输入寄存器的反馈模式。DAC7624 能在单极电源+5V 或双极电源+5V 至-5V 下工作。每个 DAC 的低功耗、小体积使 DAC7624 特别适合于自动测试装置、数据访问系统及闭环伺服控制系统。其中 MC1403

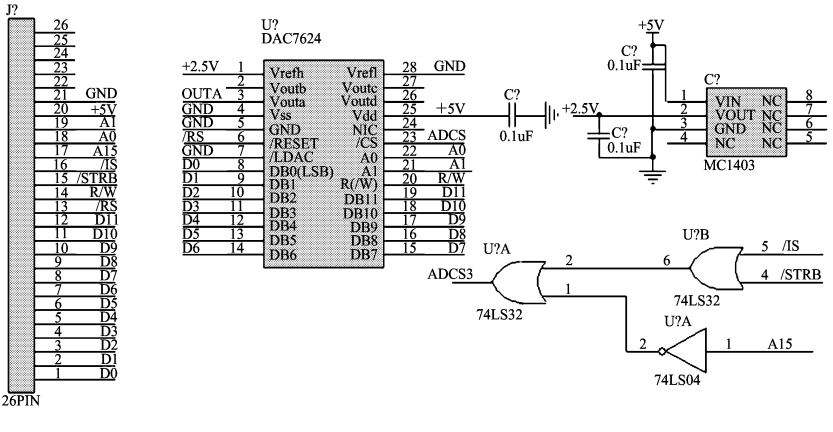


图 8.3　TMS320C240 评估板与 DAC7624 构成的正弦波发生器的硬件电路图

是一个基准源,它提供了一个+2.5的基准电压作为供给DAC7624的参考电压。

在图8.3中,TMS320C240的数据线D0～D11分别与DAC7624的DB0～DB11相连,作为12位并行输入。此电路采用单极电源方式,所以DAC7624的Vrefh接+2.5V,Vrefl接地,Vdd接+5V,Vss接地;DAC7624的片选信号如图由/IS、/STRB、/A15相或得到,即地址为8XXXH时才能访问DAC7624。为了得到平滑的输出信号,在输出端加低通滤波。

4. 实验例程

下面是以方法四实现的正弦波发生器的设计流程及程序,经过调试。

利用通用定时器来产生周期中断,在中断服务程序中计算并输出一个采样点的数据值。整体的软件流程如图8.4和图8.5。

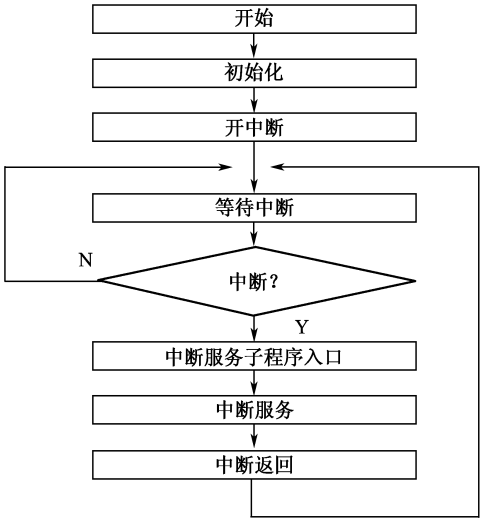


图8.4 主流程

中断服务程序中使用两个变量XY2,XY1,其流程如图8.5。

在此例程中正弦数字信号的输出采用如下简单接法:

(1) 8位数字数据经过DSP芯片的端口B输出至DAC0832的8位数字输入端;

(2) DAC的使能信号直接接通为有效状态,使用即刻转换模式;

(3)因为D/A转换器是电流输出,所以需要一个运算放大器来组成转换电路,将电流输入转换成电压输出。在电压输出端若加一简单的RC低通滤波电路,将得到更为理想的正弦信号。

下面将中断服务程序的指令逐条解释。在实际的程序中,还使用了相关指令的特殊功能。

```
.bss.Temp,1
.bss XY2,1
```

图 8.5　中断服务流程


```
    .bss XY1,1          ;定义两个变量,其中 XY2 的地址比 XY2 的小 1
    .text               ;程序开始,初始化等工作
XPERIOD:   KICK_DOG     ;中断服务开始,Watch Dog 计数器清零
    LDP   #0            ;设置数据页指针 DP
    LACC   XY1,14       ;XY1 左移 14 位装入 ACC
    NEG                 ;ACC 求补码,相当于 ACC=2^32-XY1
    LTD   XY2           ;XY2 装入 TREG;XY2=>XY1;此时 PREG=0,所以 ACC 不变
    SPLK  #7FFBH,Temp
    MPY   Temp          ;PREG=B*XY2
    MPY   Temp
    APAC
    MPY   #0            ;PREG 清零;并将 PREG 累加至 ACC,ACC=(B*XY2-XY1)*2^14
    SACH   XY2,2        ;ACC 的值左移 2 位存入 XY2;ACC 的值保持不变
    ADD   #8000H        ;四舍五入
    RPT   #4
    SFR                 ; ACC 左移 5 位,因为输出只需要高 8 位有效数据
    OR   #FF000000H     ;PBDATDIR 的高位字节为 1,表示端口 B 为输出状态
    LDP   #225
    SACH   PBDATDIR     ;ACC 高位字输出到端口 B
    LDP   #232
    LACC   EVIVRA
    CLRC   INTM
    RET                 ;中断返回
```

该数字振荡器的输出信号频率 f_s 是可以调整的。前面提到,一个周期中采样 360 个数据点,所以 f_s 是周期中断频率的 1/360。根据周期中断频率公式可得:

$$f_s = \frac{F_c / k}{360\,N}$$

其中：F_c 是 CPU 时钟的频率；

k 是 TPS 规定的分频系数，取值 $2^0 \sim 2^7$；

N 是周期寄存器规定的脉冲数，取值 0～65535。

F_c 是由下面的公式计算的：

$$F_c = F_x \times \mathrm{PLLFB}/2^{\mathrm{PLLDIV}}$$

其中：F_x 是外部时钟频率，在本系统中始终为 10MHz；

PLLFB 是 PLL 的倍频率，取值为 1、2、3、4、5 或 9；

PLLDIV 规定了 PLL 是否要经过 2 分频，取值为 2^0 或 2^1；

将上面的两个公式合并，可以得到输出信号的频率公式：

$$f_s = \frac{F_x \times \mathrm{PLLFB}}{360 \times N \times k \times 2^{\mathrm{PLLDIV}}}$$

在根据上面所列的各参数的取值范围，不难得到 f_s 的理论取值范围：

$$0.16557 \times 10^{-2} \leqslant f_s \leqslant 0.25000 \times 10^{6}\,\mathrm{Hz}$$

实际上，这个理论范围是不可能实现的。首先，系统中断服务是需要时间的，不可能在上一个中断没有处理结束以前，就响应下一个中断请求，所以理论的频率上限将受到限制；另外，输出 DAC 的频带等电路也会对上下限有所限制。

程序源代码：

```
        .include f240regs.h
;————————————————————————————————————
;Vector address declarations
;————————————————————————————————————
     .sect ".vectors"
REVECT      B START          ;reset vector
INT1        B PHANTOM        ;interrupt level 1
INT2        B XPERIOD        ;interrupt level 2
INT3        B PHANTOM        ;interrupt level 3
INT4        B PHANTOM        ;interrupt level 4
INT5        B PHANTOM        ;interrupt level 5
INT6        B PHANTOM        ;interrupt level 6
RESERVED    B PHANTOM        ;Reserved
SW-INT8     B PHANTOM        ;User S/W Interrupt
SW-INT9     B PHANTOM        ;User S/W Interrupt
SW-INT10    B PHANTOM        ;User S/W Interrupt
SW-INT11    B PHANTOM        ;User S/W Interrupt
SW-INT12    B PHANTOM        ;User S/W Interrupt
SW-INT13    B PHANTOM        ;User S/W Interrupt
SW-INT14    B PHANTOM        ;User S/W Interrupt
SW-INT15    B PHANTOM        ;User S/W Interrupt
SW-INT16    B PHANTOM        ;User S/W Interrupt
TRAP        B PHANTOM        ;Trap vector
```

```
NMINT       B PHANTOM           ;Non-maskable Interrupt
EMU_TRAP    B PHANTOM           ;Emulator Trap
SW_INT20    B PHANTOM           ;User S/W Interrupt
SW_INT21    B PHANTOM           ;User S/W Interrupt
SW_INT22    B PHANTOM           ;User S/W Interrupt
SW_INT23    B PHANTOM           ;User S/W Interrupt
;-------------------------------------------------
; MAIN CODE-starts here
;-------------------------------------------------
  .text
  NOP
START LDP #0 SETC INTM          ;disable interrupts
  SPLK #0002h,IMR               ;mask all core interrupts except INT2
  LACC IFR                      ;read interrupt flags
  SACL IFR                      ;clear all interrupt flags
  CLRC SXM                      ;Clear Sign Extension Mode
  CLRC OVM                      ;Reset Overflow Mode
  CLRC CNF                      ;Config Block B0 to Data mem
;-------------------------------------------------
;Set up PLL Module
;-------------------------------------------------
  LDP     #00E0h
  SPLK    #00BBh,CKCR1          ;CLKIN=10MHz,CPUCLK=20MHz
  SPLK    #00C3h,CKCR0          ;SYSCLK=CPUCLK/2
  SPLK    #40C0h,SYSCR          ;CLKOUT=WDCLK
  SPLK    #006Fh,WDCR           ;Disable WD if VCCP=5V
  KICK_DOG                      ;Reset Watchdog
;-------------------------------------------------
;Set up Digital I/O Port
;-------------------------------------------------
  LDP     #225                  ;Data page to Configure OCRx
  SPLK    #000Fh,OCRA           ;0000 0000 0000 1111 portB ok
  SPLK    #00FFh,OCRB
  SPLK    #0FFFFh,PBDATDIR      ;B is out port
;-------------------------------------------------
;Event Manager Module Reset
;-------------------------------------------------
  LDP     #232
  SPLK    #0000h,GPTCON
  SPLK    #0000h,T1CON
  SPLK    #0000h,T2CON
  SPLK    #0000h,T3CON
  SPLK    #0000h,COMCON
  SPLK    #0000h,ACTR
  SPLK    #0000h,SACTR
```

```
    SPLK    #0000h,DBTCON
    SPLK    #0000h,CAPCON
    SPLK    #0FFFh,EVIFRA
    SPLK    #0FFFh,EVIFRB
    SPLK    #0FFFh,EVIFRC
    SPLK    #0000h,EVIMRA
    SPLK    #0000h,EVIMRB
    SPLK    #0000h,EVIMRC
    KICK_DOG                    ;Reset Watchdog
;-------------------------------------------------
;End of RESET section for silicon revision 1.1
;-------------------------------------------------
;Set up Event Manager Module
;-------------------------------------------------
    .text
    LDP     #232
    SPLK    #3,T1PR
    SPLK    #0000h,GPTCON
    SPLK    #0000h,T1CNT
    SPLK    #1404h,T1CON
    SPLK    #0000h,T2CON
    SPLK    #0000h,T3CON
    SPLK    #0001000000000b,EVIMRA;Enable period TLPINT
    KICK_DOG
;-------------------------------------------------
;VARIABLES
;-------------------------------------------------
    .bss    Temp                ;定义变量并分配在 B2 模块
    .bss    XY2,1
    .bss    XY1,1
    .text
    SETC    SXM
    LDP     #232
    LACC    EVIFRA
    SACL    EVIFRA
    LDP     #0
   SPLK     #7FFBH,Temp
   SPLK     #286,XY2           ;XY2,XY1 赋初值
   SPLK     #0,XY1
   MPY      #0                 ;清除 PREG 寄存器
   MAR      *,AR2              ;指定 AR2 为当前辅助寄存器
   LAR      AR2,#8000H         ;AR2 指向 8000H,存放递推数据以便用 CCS 的图表观察结果
   LACC     IFR
   SACL     IFR
   MYPA     #0
```

```
    KICK_DOG
    LDP     #232
    SBIT1   T1CON,B6_MSK
    CLRC    INTM
STOP    B STOP
XPERIOD KICK_DOG            ;中断服务程序
    LDP     #0
    LACC    XY1,14
    NEG
    LTD     XY2
    MPY     Temp
    APAC                    ;将乘积结果累加到 ACC
    MPY     #0              ;清除 PREG
    MPYA    #0H
    SACH    XY2,2
    ADD     #10000000H
    RPT     #4              ;Repeat 5 times
    SFR
    SACH    *+              ;结果放在 AR2 所指数据存储区
    OR      #FF000000H
    LDP     #225
    SACH    PBDATDIR
    LDP     #232
    LACC    EVTIVRA
    CLRC    INTM
    RET
;-------------------------------------
;PHANTOM
;-------------------------------------
PHANTOM KICK_DOG
        B PHANTOM
```


8.4　实验四:伪随机序列发生器

1. 实验目的

理解伪随机序列的产生原理,并掌握产生伪随机数的算法;熟悉 DSP 的指令系统,学会使用 TMS320C2××实现算法。

2. 实验要求

利用 DSP 实现一个伪随机序列发生器,并分析其随机性。

3. 实验原理

随着通信理论的发展,噪声信号的应用越发广泛。在某些情况下,为了实现有效的通

信，应采用具有白噪声的统计特性的信号。另外，为了实现高可靠的保密通信，也希望利用随机噪声。然而，利用随机噪声的最大困难是它难以重复产生和处理。直到 60 年代，伪随机噪声的出现才使这一困难得到解决。

伪随机噪声具有类似于随机噪声的一些统计特性，同时又便于重复产生和处理。由于它具有随机噪声的优点，又避免了它的缺点，因此获得了日益广泛的实际应用。目前广泛应用的伪随机噪声都是由数字电路产生的周期序列(经滤波器处理后)得到的。我们常把这种周期序列称为伪随机序列。

通常我们都用反馈移位寄存器电路来产生伪随机序列。它又可分为线性反馈移位寄存器和非线性反馈移位寄存器两类。此实验就是利用线性反馈移位寄存器来产生出周期最长的二进制数字序列，也就是最大长度线性反馈移位寄存器序列，通常简称为 M 序列。

M 序列是由带线性反馈的移位寄存器产生的周期最长的一种序列，在时钟触发下，每次移位后，各级寄存器的状态发生变化。观察其中一级寄存器的输出，随着移位时钟节拍的推移，形成一个序列，这就是移位寄存器序列。可以看出，移位寄存器序列是一种周期序列，但当移位寄存器的级数较多时，其周期将变得很长。此外序列的周期还与反馈逻辑以及移位寄存器初始状态(称为种子)有关。

首先设定带线性反馈逻辑的移位寄存器中各级寄存器的初始状态，在时钟触发下，每次移位后，各级寄存器的状态发生变化。观察其中一级寄存器的输出，随着移位时钟节拍的推移，形成一个序列，这就是移位寄存器序列。可以看出，移位寄存器序列是一种周期序列，但当移位寄存器的级数较多时，其周期将变得很长。此外序列的周期还与线性反馈逻辑以及移位寄存器的初始状态(一般称为种子)有关。

下面以四位移位寄存器构成的序列发生器为例来简单介绍以上概念。

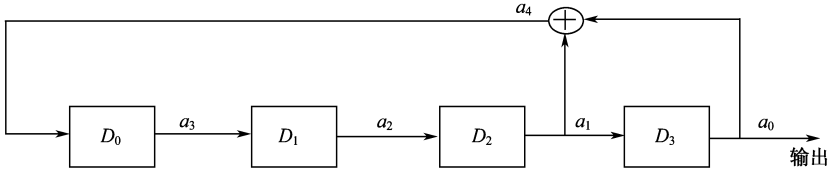


图 8.6　序列发生原理

图 8.6 中线性反馈 $a_4 = a_1 \oplus a_0$，设初始值 $D_0\ D_1\ D_2\ D_3 = 0001$，随着移动时钟节拍，$D_0\ D_1\ D_2\ D_3$ 的变化为：0001→1000→0100→0010→1001→1100→0110→1011→0101→1010→1101→1110→1111→0111→0011→0001，移动 15 个节拍后 $D_0\ D_1\ D_2\ D_3$ 回到初始值，即其周期长度为 15。不难看出，若初始状态全为“0”，即“0，0，0，0”，则移位后得到的仍为全“0”状态，也就是说一旦出现全 0 状态，则以后的序列将恒为 0。这就意味着在这种反馈移位寄存器中应避免出现全零状态，不然移位寄存器的状态将不会改变。因为 4 级移位寄存器共有 $2^4 = 16$ 种可能的不同状态，除全“0”的状态外，只剩 15 种状态可用，即由任何 n 级反馈移位寄存器产生的序列的周期最长为 $2^2 - 1$。如果从末端(或第四级) D_3 (a_0)输出，便可得到如下线性反馈移位寄存序列：

a_0=1000100110101111000100110101111

周期 =15

同样,从任何一级寄存器所得到的序列都是周期为 15 的序列,末级以外都是末级输出序列向前或向后移若干节拍的结果。这些序列都是线性反馈移位寄存器序列,而且是最长线性反馈移位寄存器序列。

如果将线性反馈逻辑改为 $a_4 = a_2 \oplus a_0$,假设初始值仍然为 $D_0\ D_1\ D_2\ D_3 = 0001$,则第四级输出 100010 的重复序列。如果线性反馈逻辑仍为 $a_4 = a_2 \oplus a_0$,而将初始值分别改为 1011 或 1111,则输出序列分别 110 和 111100,周期分别为 3 和 6。

以上说明线性反馈移位寄存器不仅与线性反馈逻辑有关,而且还与初始值有关。然而,在产生最长线性反馈移位寄存器序列时,初始状态并不影响序列的周期长度(即只要初始状态为非全 0 的任何一个),关键在于设计合适的线性反馈逻辑。

下面我们就来讨论反馈电路如何连接才能使移位寄存器产生的序列最长。一般地,n 级线性反馈移位寄存器如图 8.7 所示。图中,C_i($i=0,1,\cdots,n$)表示反馈线的连接状态,$C_i=1$ 表示接通,将第 $n-i$ 级输出加入反馈;$C_i=0$ 表示断开,第 $n-i$ 级输出不参加反馈。因此,一般形式的线性反馈逻辑表达式为:

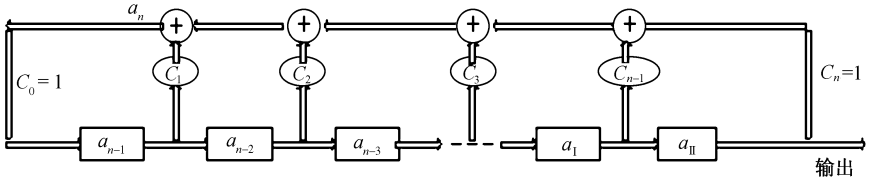


图 8.7　n 级线性反馈移位寄存器

$$a_n = C_1\,a_{n-1} \oplus C_2\,a_{n-2} \oplus C_3\,a_{n-3} \oplus \cdots \oplus C_n a_0 = \sum_{i=1}^{n} C_i a_{n-i}\ (\text{异或})$$

上面曾经指出:C_i 的取值决定了移位寄存器的反馈连接和序列的结构,故 C_i 是一个很重要的参量。现在将它用下列方程表示:

$$f(x) = C_0 + C_1 x + C_2 x^2 + \cdots + C_n x^n = \sum_{i=0}^{n} C_i x^i$$

这一方程称为特征方程(或特征多项式)。式中 x^i 仅指明 C_i 的系数(1 或 0),x 本身的取值并无实际意义,也不需要去计算 x 的值。例如,若特征方程为:

$$f(x) = 1 + x + x^4$$

则它仅表示,x^0,x^1 和 x^4 的系数 $C_0 = C_1 = C_4 = 1$,其余的 C_i 为零($C_2 = C_3 = 0$)。图 8.6 中所示的就是按这一特征方程构成的反馈移位寄存器。

通过复杂的证明可以推得:一反馈移位寄存器能产生 M 序列的充要条件是:反馈移位寄存器的特征多项式为本原多项式。所谓本原多项式是指一个 n 次多项式 $f(x)$ 满足下列条件:

(1) $f(x)$ 为既约的(所谓既约多项式是指不能分解因子的多项式)。

(2) $f(x)$ 可整除 (x^m+1), $m=2^n-1$。

(3) $f(x)$ 除不尽 (x^q+1), $q<m$ 。

例如要求用一个 4 级反馈移位寄存器产生 M 序列时,就要求它的特征多项式。具体求解过程如下:

因为给定 $n=4$,故此移位寄存器产生的 M 序列的长度为 $M=2^n-1=15$,由于其特征多项式 $f(x)$应可整除 $(x^m+1)=(x^{15}+1)$,或者说,$f(x)$应是 $(x^{15}+1)$的一个因子,故将 $(x^{15}+1)$分解因子,从其因子中找 $f(x)$:

$$(x^{15}+1)=(x^4+x+1)(x^4+x^3+1)(x^4+x^3+x^2+x+1)\times(x^2+x+1)(x+1)$$

$f(x)$不仅应为 $(x^{15}+1)$的一因子,而且还应该是一个 4 次本原多项式。上式表明,$(x^{15}+1)$可以分解为 5 个既约因子,其中 3 个是 4 次多项式。可以证明,这 4 个多项式中,前两个是本原多项式,第 3 个不是。因为:

$$(x^4+x^3+x^2+x+1)(x+1)=(x^5+1)$$

这就是说,$(x^4+x^3+x^2+x+1)$不仅可整除 $(x^{15}+1)$,而且还可以整除 (x^5+1),故它不是本原的。因此,我们找到了两个 4 次本原多项式:(x^4+x+1)和 (x^4+x^3+1)。由其中任何一个都可产生 M 序列,用 (x^4+x+1)作为特征多项式构成的 4 级反馈移位寄存器就是图 8.6 中给出的。

由上所述,只要找到了本原多项式,就能由它构成 M 序列产生器。另外,本原多项式的逆多项式也是本原多项式。例如 (x^4+x+1)与 (x^4+x^3+1)互为逆多项式,即 10011 与 11001 互为逆码。所以,每一本原多项式可以组成两种 M 序列产生器。

此外,M 序列还具有均衡性、游程分布、自相关特性和功率谱等性质。这些性质与随机序列的基本性质很相似,所以通常认为 M 序列属于伪噪声序列或伪随机序列。

4. 32bit 伪随机序列的 DSP 实现

利用 TMS320C240 的定时器中断来产生一个 32bit 宽的伪随机序列。具体的过程是:首先设定 DSP 定时器和中断定时器寄存器的初始状态,以便产生定时器中断,再设定待线性反馈逻辑的移位寄存器中各级寄存器的初始状态,然后等待中断,定时器满,产生中断,进入中断服务程序,完成对第 31 位,30 位,28 位,17 位的异或运算,然后将结果反馈至第 0 位。每次移位后,寄存器的状态发生变化,随着移位时钟节拍的推移,形成一个

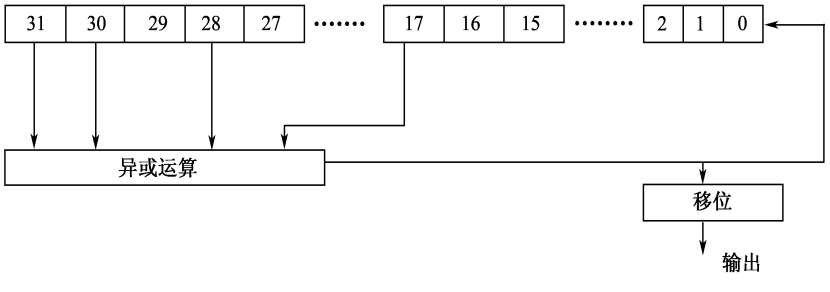


图 8.8 32bit 伪随机序列发生器

移位寄存器序列。最后,通过不断的执行所编程序,可以观察到分别存放高 16 位伪随机数和低 16 位伪随机数的寄存器中的值不断变化。另外,此伪随机序列也可以从移位寄存器输出到端口 1(Port 1)上,然后经过一个 D/A 转换器,就可以用示波器来观察此伪随机序列的波形。从图 8.8 的一般结构形式很容易得到方框图 8.9,以及相应的反馈逻辑表达式如下:

$$\text{Bit0}=\text{Bit31}\oplus\text{Bit30}\oplus\text{Bit28}\oplus\text{Bit27}$$

从前面的讨论可以看出,要得到周期或较长周期的伪随机序列,就必须采用合理的反馈逻辑,足够的寄存器级数,以及合适的初值。我们要编写的程序,就是利用 TMS320C2××的 32bit 累加器以及初始状态>7E521603,产生了一个最长周期的伪随机数。

设 DSP 的时钟为 $f_{CLK}=10\text{MHz}$,采样频率 $f_s=16\text{kHz}$。因而定时器的初值为:

$$N=\frac{f_{\text{CLK}}}{f_s}-1=\frac{10\text{MHz}}{16\text{kHz}}-1=625=271\ (\text{h})$$

5. 程序流程图

程序流程如图 8.9 所示。

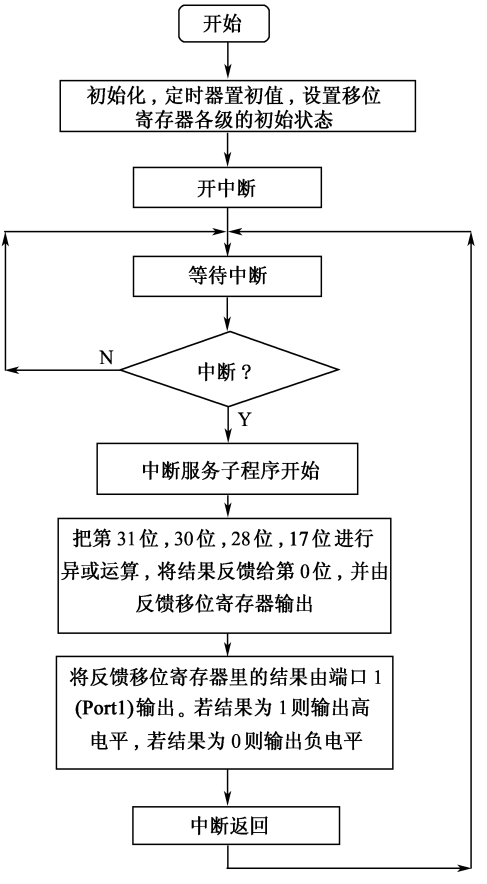


图 8.9　实验流程

8.5 实验五:FFT算法的实现

1. 实验目的

熟悉FFT的算法规则,了解按时间抽取和按频率抽取的算法编程,了解专用于FFT运算的DSP位倒序操作的间接寻址方式。进一步熟悉DSP的指令系统及FFT实现方法。

2. 实验要求

按照实验原理,编写FFT程序。要求能输入一个模拟信号,经过ADC及FFT之后,从示波器观察谱图。

3. 实验原理

(1)DFT概述

DFT(Discrete Fourier Transform缩写为DFT)是信号分析与处理中的一种重要数学变换。其实质是有限长序列傅里叶变换的有限点离散采样,即信号序列的DFT本身就是信号频谱的采样集,它使得数字信号处理可以在频域采用数字运算的方法进行,开辟了频域离散化的道路,大大增加了数字信号处理的灵活性。虽然DFT如此重要,但是很长一段时间里,由于DFT运算的冗长和繁琐(若采样个数为 N,则需要约 N^2 次加法和 N^2 次乘法运算,当 N 较大时,计算量太大),它并没有得到真正的运用。直到快速傅里叶变换(Fast Fourier Transform,缩写为FFT)出现以后,使得运算大大简化,运算时间缩短了一两个数量级之多,从此DFT这种数学变换才真正在实际中得到广泛应用。FFT是DFT的一种高效快速运算方法,为数字信号处理技术应用于各种信号的实时处理创造了条件,同时也推动了数字信号处理技术的发展。1984年,法国的P.Dohamel和H.Hollmamn提出的分裂基快速算法,使运算效率进一步提高。

下面简要介绍各算法及其变换。

(2)DFT算法及特点

设 $x(n)$ 是一个长度为 M 的有限长序列,则 $x(n)$ 的 N 点DFT及DFT逆变换IDFT为

$$X(k) = DFT[x(n)] = \sum_{n=0}^{N-1} x(n) W_N^{kn}, 0 \leqslant k \leqslant N-1$$

$$x(n) = IDFT[x(K)] = \frac{1}{N}\sum_{n=0}^{N-1} X(k) W_N^{-kn}, 0 \leqslant n \leqslant N-1 \tag{8.1}$$

由此可见,要完成一次DFT运算需要 N^2 次复数相乘和 $N(N-1)$ 次复数相加。当 N 较大时,运算量相当可观,对于实时处理来说,必将对计算速度提出难以实现的要求。但是这种运算是可以改进的,式中 W_N(有时简写为 W)代表 $e^{-j(2\pi/N)}$,不难看出 W^{nk} 具有以下性质。

① 系数 $W_N^{nk} = e^{-j(2\pi/N)nk}$ 是一个周期函数,而且周期为 N,即

$$W_N^{(n+mN)(k+lN)} = W_N^{nk} \quad m,l = 0,\pm 1,\pm 2,\cdots \tag{8.2}$$

② 系数 $W_N^{nk} = e^{-j(2\pi/N)nk}$ 具有对称性

$$W_N^{n(N-k)} = W_N^{k(N-n)} = W_N^{-nk} \tag{8.3}$$

又因 $W_N^{N/2} = -1$,因此

$$W_N^{(k+N/2)} = -W_N^k \tag{8.4}$$

利用这些周期性与对称性,使 DFT 运算中有些项可以合并,来改进运算。使 DFT 运算尽量分解为更少点数的运算。

快速傅里叶变换算法正是基于这样的基本思想而发展起来的。它的算法形式有很多种,但基本上可以分为两大类:即时间抽取法和频率抽取法。

(3)频率域采样

时域采样定理告诉我们,在一定条件下,可以由采样信号无失真地恢复原来连续信号。

那么要由频域采样信号(即 DFT 结果 $X(k)$)恢复原来的函数,也必须满足以下条件:

如果 $x(n)$的长度为 M,则只有当频率域采样点数 $N \geqslant M$ 时,才有

$$x_N(n) = IDFT[X(k)] = x(n)$$

即由频域采样 X(k)恢复原来序列 $x(n)$,否则会产生时域混叠现象。这就是频域采样定理。

(4)用 DFT 对信号进行谱分析

所谓信号的谱分析,就是计算信号的傅里叶变换。工程实际中,经常遇到的连续信号 $x(t)$,其频谱函数 $X(j\Omega)$也是连续函数。为了利用 DFT 及借助数字信号处理器这一有力的工具进行谱分析,首先对 $x(t)$进行时域采样,得到 $x(n) = x(nT)$,再对 $x(n)$进行 DFT,得到的 $X(k)$则是 $x(n)$的傅里叶变换 $X(e^{j\omega})$在$[0,2\pi]$上的 N 点等间隔采样,或是 $x(n)$的 Z 变换在单位圆上的 N 点等间隔采样。这里 $X(k)$和 $x(n)$均为有限长序列。然而,由傅里叶变换理论可知,若信号持续时间有限长,则其频谱应无限长;反之,若信号频谱有限宽,则信号持续时间无限长。所以,严格讲持续时间有限的带限信号是不存在的。实际处理中,对频谱很宽的信号,为了防止采样后产生频谱混叠失真,在采样前先滤去副度较小的高频成分,使连续的带宽小于折叠频率。对于持续时间很长的信号,采样点数太多,会受存储器空间和计算量的限制,只好截取有限点进行 DFT。从工程角度看,滤除幅度较小的高频成分和截去幅度较小的部分时间域信号是允许的。由以上分析可见,用 DFT 对连续信号进行谱分析必然是近似的,其近似程度与信号的带宽、采样频率和截取长度有关。

在对连续信号进行谱分析时,主要关心谱分析范围和频率分辨率两个问题。谱分析范围受采样频率 f_s 的限制,为了防止频率混叠失真,$f_s \geqslant 2f_{max}$,f_{max}为信号的最高频率分量,通常取采样频率为 f_{max}的 3～5 倍更好。频率分辨率用频率采样间隔 F 来描述,$F = f_s/N$,F 越小,频率分辨率越高。

如果保持采样点数 N 不变,要提高谱分辨率,必须降低采样频率,从而减小了谱分析范围。如果维持采样频率 f_s 不变,为了提高频率分辨率可以增加采样点数 N,即增加对

信号的观察时间 T_p，$NT=T_p$，$T=1/f_s$。因此，为了提高谱分辨率（$F=f_s/N=1/NT=1/T_p$）和谱分析范围，必须增长对信号的记录时间及提高采样频率。

用DFT对周期为 N 的周期序列进行谱分析，频率也是以 N 为周期的离散谱，即每个周期有 N 条谱线，第 k 条谱线位于 $\omega=(2\pi/N)k$ 处，代表 $x(n)$ 的 k 次谐波分量。而且谱的相对大小与 $X(k)$ 成正比，$k=0$ 时谱线对应的了信号直流分量。如果截取的了整数 m 个周期的序列信号进行DFT谱分析，则得到的谱线幅度扩大了 m 倍。

如果预先不知道序列的周期，可以先截取 M 个点进行DFT，然后再将截取长度扩大1倍（即取2 M 个点）进行DFT，如果两者的主谱差别满足分析误差要求，则取任何一个的DFT（$X(k)$）近似表示序列的频谱。否则，继续将截取长度加倍，直到前后两次分析所得主谱频率差别满足误差要求。设最后截取长度为 rM，则 $X_{rM}(k)$ 表示 $\omega=(2\pi/rM)k$ 的谱线强度。

DFT进行谱分析时，必然有误差存在，主要有以下三种产生误差的情况：

① 混叠，为了防止在折叠频率（$f_s/2$ 或 $\omega=\pi$）附近发生频谱混叠现象，一般在采样前先滤除信号中高于折叠频率 $f_s/2$ 的频率成分，并取采样频率 f_s 为信号最高频率的（3～5）倍。

②栅栏效应，N 点的DFT是在[0,2π]上对信号的频谱进行 N 点等间隔采样，而采样点之间的频谱函数是未知的，这就好像从 $N-1$ 个栅栏缝隙中观看信号的频谱，有可能漏掉大的频谱分量，为了检测出这些分量，可以采用在原序列尾部补零的方法，改变序列长度N，从而增加频域采样点数和采样点位置，使原来漏掉的某些频谱分量被检测出来。对连续信号的谱分析，只要采样频率足够高，且采样点数满足频率分辨率要求，就可以认为DFT后所得离散谱的包络近似代表原信号的频谱。

③截取效应，实际中遇到的序列可能是无限长的，用DFT进行谱分析时必须截取成有限序列，得到的离散谱线向附近展宽，称这种现象为泄漏。显然泄漏使频谱变模糊，谱分辨率降低。同时截取后得到的主谱线两边形成很多旁瓣，引起不同频率分量间的干扰（简称谱间干扰），影响频谱分辨率。减小截断效应最好的方法是用近代谱估计的方法。但谱估计只适合于不需要相位信息的谱分析场合。

（5）FFT

FFT实际上就是将 N 点DFT分解为若干个短序列的DFT，是DFT的快速运算形式。N 点的DFT先分解为2个 $N/2$ 点的DFT，每个 $N/2$ 点的DFT再分解为 $N/4$ 点的DFT，等等。最小变换的点数即所谓的“基数”。因此，基2的FFT算法的最小变换（或称为碟形）是2点的DFT。一般地，对 N 点FFT，对应 N 个输入采样值，有 N 频域样值与之对应。并利用旋转因子 W_N^m 的周期性和对称性来减少运算工作量。最常用的是基2FFT（即 $N=2^M$，将 $x(n)$ 序列经过 $M-1$ 次分解为 $N/2=2^{M-1}$ 个2点的DFT序列）。

FFT的基本算法分为两大类：时间抽取法FFT（Decimation－In－Time FFT，DIT－FFT）——对 $x(n)$ 按 n 的奇偶将 $x(n)$ 分解；频率抽取法FFT（Decimation－In－Frequency FFT，DIF－FFT）——将 $x(n)$ 前后对半分开分解，这种算法是对频域 $X(k)$ 奇偶抽取分解的结果，所以称之为频域抽取法FFT。两种算法的运算次数相同。

下面主要介绍基2按时间抽取FFT算法。这种算法是将输入序列分成奇数项和偶数项。

假设 $N=2^M$,其中 M 为正整数,这样可以将序列 $x(n)$分为两组,偶数项为一组,奇数项为一组。

$$x(2r)=x_1(r)$$
$$x(2r+1)=x_2(r) \tag{8.5}$$

将 DFT 运算也相应分为两组:

$$\begin{aligned}X(k)&=\mathrm{DFT}[x(n)]=\sum_{n=0}^{N-1}x(n)W_N^{nk}\\&=\sum_{\substack{n=0\\n\text{为偶数}}}^{N-1}x(n)W_N^{kn}+\sum_{\substack{n=0\\n\text{为奇数}}}^{N-1}x(n)W_N^{nk}\\&=\sum_{r=0}^{N/2-1}x(2r)W_N^{2rk}+\sum_{r=0}^{N/2-1}x(2r+1)W_N^{(2r+1)k}\\&=\sum_{r=0}^{N/2-1}x_1(r)W_N^{2rk}+W_N^k\sum_{r=0}^{N/2-1}x_2(r)W_N^{2rk}\end{aligned} \tag{8.6}$$

同时我们注意到 W_N^{2n} 可以转化为 $W_{N/2}^n$:

$$W_N^{2n}=\mathrm{e}^{-j\frac{2\pi}{N}2n}=\mathrm{e}^{-j\frac{2\pi}{N/2}n}=W_{N/2}^n \tag{8.7}$$

因此

$$X(k)=\sum_{r=0}^{N/2-1}x_1(r)W_{N/2}^{rk}+W_N^k\sum_{r=0}^{N/2-1}x_2(r)W_{N/2}^{rk} \tag{8.8}$$

其中 $X_1(k)$及 $X_2(k)$分别是 $x_1(r)$及 $x_2(r)$的 $N/2$ 点的 DFT,

$$\begin{aligned}X_1(k)&=\sum_{r=0}^{N/2-1}x_1(r)W_{N/2}^{rk}=\sum_{r=0}^{N/2-1}x(2r)W_{N/2}^{rk}\\X_2(k)&=\sum_{r=0}^{N/2-1}x_2(r)W_{N/2}^{rk}=\sum_{r=0}^{N/2-1}x(2r+1)W_{N/2}^{rk}\end{aligned} \tag{8.9}$$

这样我们就看到,一个 N 点的 DFT 被分解成两个 $N/2$ 点的 DFT 了,这两个 $N/2$ 点的 DFT 再按式(8.8)合成一个 N 点的 DFT。这里应该看到 $X_1(k)$, $X_2(k)$只有 $N/2$ 个点,要用 $X_1(k)$, $X_2(k)$表达全部 $X(k)$值还必须应用 W 系数的周期特性,即

$$W_{N/2}^{r(N/2+k)}=W_{N/2}^{rk}$$

因此

$$X_1(N/2+k)=\sum_{r=0}^{N/2-1}x_1(r)W_{N/2}^{r(N/2+k)}=\sum_{r=0}^{N/2-1}x_1(r)W_{N/2}^{rk} \tag{8.10}$$

即

$$X_1(N/2+k)=X_1(k) \tag{8.11}$$

同样

$$X_2(N/2+k)=X_2(k) \tag{8.12}$$

同时考虑到 W_N^k 的对称性:

$$W_N^{\left(\frac{N}{2}+k\right)}=W_N^{\frac{N}{2}}\cdot W_N^k=-W_N^k \tag{8.13}$$

将式(8.11～8.13)代入式(8.8)就可以将 $X(k)$表达为前后两部分：

$$X(k)= X_1(k)+ W_N^k X_2(k) \tag{8.14}$$

$$k = 0,1,\cdots,\frac{N}{2}-1$$

$$X\left[k+\frac{N}{2}\right] = X_1(N/2+k)+ W_N^{(N/2+k)} X_2(N/2+k) \tag{8.15}$$

$$= X_1(k)- W_N^k X_2(k)$$

$$k = 0,1,\cdots,\frac{N}{2}-1$$

式(8.14)表示了前半部分 $k=0$ 到 $N/2-1$ 的 $X(k)$组成方式，式(8.15)则表示了后半部分由 $N/2$ 到 $N-1$ 的 $X(k)$组成方式。这两式由图 8.10 所示的流图符号来表示，根据其形状称之为蝶形运算符号，由图可见完成一次蝶形运算需要一次乘法和两次加法运算。

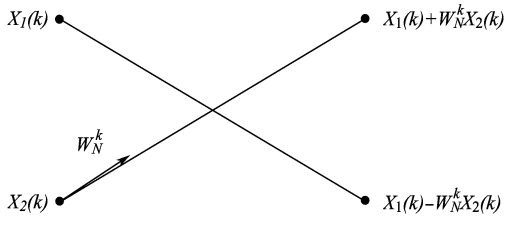


图 8.10　蝶形运算符号

通过这样的分解，每一个 $N/2$ 点的 DFT 只需要 $(N/2)^2 = N^2/4$ 次复数相乘运算。两个 $N/2$ 点的 DFT 需要 $N^2/2$ 次复数乘，再加上将两个 $N/2$ 点 DFT 合成为 N 点 DFT 时，在蝶形结前的 $N/2$ 次复数乘，一共需要 $N^2/2+N/2=N(N+1)/2\approx N^2/2$ 次复乘，工作量比原来差不多少了一倍。以此类推，继续分解下去，经过 $M-1$ 次分解最后剩下的是 2 点的 DFT。这种方法由于每一步分解都是按输入序列在时域上的次序是属于偶数还是奇数来抽取得，所以称为“按时间抽取法”。

图 8.11 为一个完整的 8 点 DIT－FFT 运算流图，图中输入序列不是顺序排列的，但它们的排列有一定规律，即二进制位倒序。图中输出 $X(k)$是自然顺序排列。

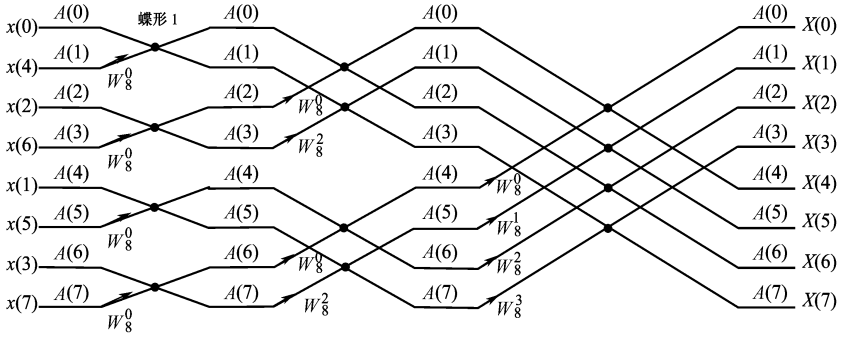


图 8.11　8 点 DIT－FFT 运算流图

基 2 按频率抽取 FFT 算法：

这种算法是将输入序列分为前 $N/2$ 和后 $N/2$ 点。

$$X(k)=\sum_{n=0}^{\frac{N}{2}-1}x[n]W_N^{nk}+\sum_{n=N/2}^{N-1}x[n]W_N^{nk}$$

$$=\sum_{n=0}^{\frac{N}{2}-1}(x_1[n]+e^{-j\pi k}x_2[n]W_N^{nk}$$

其中 $x_1[n]=x[n]$，$x_2[n]=x[n+N/2]$，$n=0,1,\cdots,\frac{N}{2}-1$。将 $X(k)$分为奇数组和偶数组，所以称之为频域抽取法 FFT。它与 DIT－FFT 算法类似，运算次数也相同。不同的是 DIF－FFT 输入序列为自然顺序，而输出序列为位倒序。因此，运算蝶形运算结束后，要对输出顺序进行倒序列，得到自然顺序的 $X(k)$。

4.FFT 编程简介

(1) 原位计算

由图 8.11 可见，DIT－FFT 的运算有一定的规律，$N=2^M$ 点的 FFT 共有 M 级蝶形运算，每级有 $N/2$ 个蝶形运算。同一级中每个蝶形的两个输入数据只对计算本蝶形有用，而且计算完一个蝶形后，数据可以输入原来数据所用存储单元，比如，图中蝶形 1 的两个输入分别存储在 $A(0)$和 $A(1)$单元，则其计算结果可以对应放入 $A(0)$和 $A(1)$中。那么，经过 M 级运算后，存放原始输入序列数据的 $A(0)\sim A(7)$存储单元中则依此存放 $X(k)$的 N 个值，即输出占据输入数据的位置，称为原位计算，当 N 较大时，可以节省一定的存储空间。

(2) 旋转因子变化规律

由图 8.11 可见，每个蝶形都要乘以旋转因子 W_N^p，p 称为旋转因子指数，为了便于编程，必须了解 W_N^p 与 FFT 运算级数的关系。假设，用 L 表示从左到右的运算级数($L=1,2,\cdots,M$)，则第 L 级共有 2^{L-1}个不同的旋转因子。将 $N=8$ 时各级的旋转因子表示如下

$L=1$ 时，$W_N^p=W_{N/4}^J=W_2^{JL}$，$J=0$

$L=2$ 时，$W_N^p=W_{N/2}^J=W_2^{JL}$，$J=0,1$

$L=3$ 时，$W_N^p=W_N^J=W_2^{JL}$，$J=0,1,2,3$

对一般情况则有：$W_N^p=W_2^{JL}=W_N^{J\cdot 2^{M-L}}$，$p=0,1,\cdots,2^{L-1}-1$

所以 $p=J\times 2^{M-L}$；运算时 L 作为循环变量。

由此可见，基 2 的 N 点 FFT 运算的旋转因子数共有：2^{M-1}即 $N/2$ 个。在用汇编编程时，常将 $N/2$ 个旋转因子的实部和虚部制表，用查表法减少运算量。

(3) 序列倒序

从图 8.11 可以看出，算法的输出序列是顺序排列的，而输入序列的排列似乎是混乱的，但仔细分析会发现输入序列排列有一定的规律，即码位倒置或叫比特反转。因此在编程时，要考虑如何完成从混序到自然序列以及如何从自然序列到混序列的操作。当 N 为 2 的幂时，从混序到顺序只需按位倒序存储即可。位倒序的规则如下：如果 $N=2^M$，则序列的自然顺序标号可用 M 位二进制表示，将二进制表达式的各高低位依此倒换，即第 M

位与第1位交换,第 $M-1$ 位与第2位交换……。则最终得到的二进制表达式之值就是在那一位置上混序的标号。表8.1以 $N=8$ 为例给出了倒序表。

表8.1 $N=8$ 时的倒序表

自然顺序标号	二进制表示	位倒序的二进制表示	位倒序标号
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

同理,如果输入是顺序的,输出顺序则符合比特反转规律。

TMS320序列中从C25开始及以后的产品均为FFT运算的位倒序操作提供了一种特殊间接寻址的方式—位倒序(或位反转)的间接寻址方式。不具备这种寻址方式的DSP,必须用软件实现。

(4)归一化

由于TMS320C2xx是定点DSP,在完成FFT运算编程时归一化就是一个很重要的问题,因为数据不仅在输出级有可能溢出,每一级中间结果也有可能溢出。为了说明溢出问题将图8.10再搬过来。

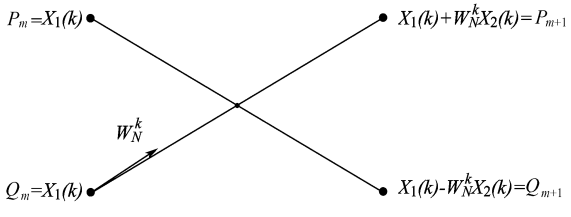


图8.12 蝶形运算符号

实验5.1由图可见,将一次蝶形的输出为:$P_{m+1}=P_m+W_N^iQ_m$

$$Q_{m+1}=P_m-W_N^iQ_m$$

设 PR、PI 和 QR、QI 分别为 Pm 和 Qm 的实部和虚部。令 $x=(2\pi/N)*i$;

则有 $W_N^i=e^{-j(2\pi/N)i}=\mathrm{con}(x)-j\mathrm{sin}(x)$

$$P_{m+1}=PR+jPI+[\mathrm{con}(x)-j\mathrm{sin}(x)](QR+jQI)$$

$$=PR+QR\mathrm{con}(x)+QI\mathrm{sin}(x)+j[PI+QI\mathrm{con}(x)-QR\mathrm{sin}(x)]$$

$$Q_{m+1}=PR+jPI-[\mathrm{con}(x)-j\mathrm{sin}(x)](QR+jQI)$$

$$=PR-QR\mathrm{con}(x)-QI\mathrm{sin}(x)+j[PI-QI\mathrm{con}(x)+QR\mathrm{sin}(x)]$$

如果蝶形的输入用 $Q15$ 表示,幅度不超过 1,则 P_{m+1}和 Q_{m+1}的实部和虚部的最大幅度为:

$$1+\text{con}(45°)+\sin(45°)=2.4142$$

为了避免溢出,每级 FFT 蝶形要用 2.4142 因子进行归一化,会增加运算量。实数 FFT 运算时,考虑 $QI=PI=0$,则 P_{m+1}和 Q_{m+1}的实部和虚部的最大幅度为 2,这样利用 DSP 的移位特性很容易归一化,最后运算的结果已经是$(1/2^M)$倍了,M 为 FFT 蝶式运算的级数。

由 TMS320C2××实现 FFT 时,位反转寻址方式为 FFT 寻址提供很大方便,值得注意的是:在 DSP 中,存放数据的基地址(即当前辅助寄存器的内容)应该为数据阵列的倍数,否则,存储结果将是错误的,达不到 FFT 要求的预期目的。

另外,从示波器观察输出结果时,因为输出为一些不连续的谱线,因而如果直接在示波器上输出,很难看到令人满意的结果,所以,建议使示波器工作在外触发的方式,由 XF 端提供触发信号。

5. 实验例程

下面是利用 C 语言实现的 FFT 运算,程序中采用基 2 时间抽取的 FFT 算法。要求输入 FFT 蝶形运算的级数 M,则可以得到程序中给定输入的频域序列。程序在 Microsoft Visual C++6.0 环境调试通过。


```
//////////////////////// FFT运算 ////////////////////////////////////
    #include "math.h"
    #include "stdio.h"
    #define pi 3.1415926535
    void main()
    {
        struct complex          //定义一个复数结构
        {
            double real;
            double imag;
        };

        double n,ii,nv2,nml,k,pw[1024];
        int m,i,j,L,ip,le,le1;
        struct complex a[512],t,u,w;
        printf("Welcome to use this FFT function! \n");
        printf("Please input the order:M\n");
        scanf("%d",&m);
        n=pow(2,m);
        printf("n=%f\n",n);

//---------- 输入样本信号并产生输入序列 ----------------
        for(i=0;i<n;i++)
```

```
        {
            a[i].real=1+2*sin(2*pi*i/n)+sin(16*pi*i/n)+sin(8*pi*i/n);
            a[i].imag=0;
        }

//  ---------- 将输入序列混序排列 ----------
    nv2=n/2;
    nm1=n-1;
    j=0;
    for(i=0;i<=nm1;i++)
    {
        if(i<j)
        {
            t.real=a[j].real;
            t.imag=a[j].imag;
            a[j].real=a[i].real;
            a[j].imag=a[i].imag;
            a[i].real=t.real;
            a[i].imag=t.imag;
        }
        k=nv2;
        while(k<=j)
        {
            j=j-k;
            k=k/2;
        }
        j=j+k;
    }
//----------进行M级蝶形DIT-FFT运算----------
    for(L=1;L<=m;L++)
    {
        le=pow(2,L);
        le1=le/2;
        u.real=1;       //u=con()-jsin()表示旋转因子,起始con0=1 sin0=0
        u.imag=0;
        w.real=cos(pi/le1);
        w.imag=-1*sin(pi/le1);
        for(j=0;j<le1;j++)
        {
            for(i=j;i<n;i=i+le)
            {
                ip=i+le1;          //按实验5.1进行蝶形运算
                t.real=a[ip].real*u.real-a[ip].imag*u.imag; //实现QW
```

```
                t.imag=a[ip].real*u.imag+a[ip].imag*u.real;
                a[ip].real=a[i].real-t.real;          //P-QW
                a[ip].imag=a[i].imag-t.imag;
                a[i].real=a[i].real+t.real;           //P+QW
                a[i].imag=a[i].imag+t.imag;
            }
            t.real=u.real*w.real-u.imag*w.imag;
            t.imag=u.real*w.imag+u.imag*w.real;
            u.real=t.real;
            u.imag=t.imag;
        }
    }

//-------- FFT运算结果输出1行4个频域数据--------
        printf("输出横坐标为2*pi的倍数\n");
        for(i=0;i<n;i++)
        {
            b[i]=pow(a[i].real,2)+pow(a[i].imag,2);
        }
        i=0;
        while(i<n)
        {
            for(j=0;j<4;j++)
            {
                printf("pw[%2d]=%9.4f,",i+j,pw[i+j]);
            }
            printf("\n");
            i=i+4;
        }
    }
//-----------FFT结束-----------------
```


运行 FFT.exe 并输入 5 得到如下结果：

```
Welcome to use this FFT function!
Please input the order:M
n=32.000000
横坐标为2*pi的倍数
pw[ 0]=1024.0000, pw[ 1]=1024.0000, pw[ 2]= 0.0000, pw[ 3]= 0.0000,
pw[ 4]= 256.0000, pw[ 5]= 0.0000, pw[ 6]= 0.0000, pw[ 7]= 0.0000,
pw[ 8]= 256.0000, pw[ 9]= 0.0000, pw[10]= 0.0000, pw[11]= 0.0000,
pw[12]= 0.0000, pw[13]= 0.0000, pw[14]= 0.0000, pw[15]= 0.0000,
pw[16]= 0.0000, pw[17]= 0.0000, pw[18]= 0.0000, pw[19]= 0.0000,
pw[20]= 0.0000, pw[21]= 0.0000, pw[22]= 0.0000, pw[23]= 0.0000,
```

pw[24]= 256.0000, pw[25]= 0.0000, pw[26]= 0.0000, pw[27]= 0.0000,
pw[28]= 256.0000, pw[29]= 0.0000, pw[30]= 0.0000, pw[31]=1024.0000,

利用 C 程序理解了 FFT 算法后,用 DSP 汇编编程可以更充分地利用 DSP 的结构特点,得到运算更优化的目标代码。

在实验中应考虑用 DSP 内部的 ADC 采样模拟信号,然后 DIT-FFT 运算,并将信号的谱线 D/A 变换后输出,用示波器的外触发方式对实验进行观察。

考虑到本实验所用到的系统资源较少,因此,将外接的 DAC0832 设计工作在直通状态,DAC 的数据由 DSP 的 PB 口提供,这样以来,使得这个实验简单、可靠。

DAC0832 是电流型的,有电流输出 Iout1 和电流输出 Iout2 两个输出端,在 DAC0832 的输出端接一运算放大器,将电流信号转换为电压信号。本实验的硬件接线如图 8.13 所示。

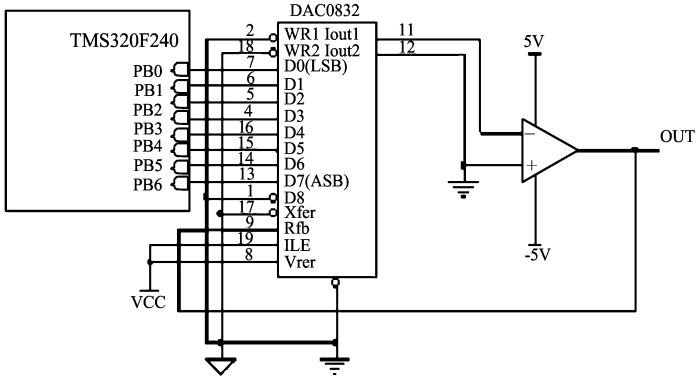


图 8.13 DAC0832 与 DSP 的连接

8.6 实验六:双音多频电话拨号音频解调/发生器

1. 实验目的

本实验的目的,是设计一个系统,它可以解调与产生标准的双音多频(DTMF)按键拨号信号。

2. 实验要求

完成用 DSP 实现 DTMF 发生及检测的具体实现方法,并利用 MATLAB 对设计进行仿真。

3. 实验原理

DTMF(Double Tone Multi-Frequency)主要用于在电话频段内传送简单的操作信号,如在具有 DTMF 功能的电话机上按数字键,电话机将该数字键信号转换为 DTMF 信号,并通过电话线传到交换机。交换机侧再将信号还原为原来的数字,并进行相应操作。

在 DTMF 中共有 8 个频率,分为 4 个高频音和 4 个低频音或称为 4 个列频和 4 个行频。用一个列频率(高频)和一个行频率(低频)的组合来表示一个信号,这样,一个双音多

频电话机可以发出 16 个不同的频率模式,每个模式代表一个键。表 8.2 为 DTMF 系统中的频率组合和分配情况。

表 8.2　DTMF 频率组合与分配

行 \ 列	1 (1209Hz)	2 (1336Hz)	3 (1477Hz)	4 (16330Hz)
1(697Hz)	1	2	3	A
2(770Hz)	4	5	6	B
3(852Hz)	7	8	9	C
4(941Hz)	*	0	#	D

例如,按下键“*”,则所选的频率为行频 941Hz 和列频为 1209Hz。反之,如果这两个频率被检测出来,就说明收到键“*”信号。AT&T 对 DTMF 信号所规定的指标是:传送/接收率为每秒 10 个数字,或每个数字 100ms。该信号必须存在至少 45ms,且不得多于 55ms。而 100ms 内的其余时间是无声的。

DTMF 可以用模拟或数字方法实现。在此仅介绍数字方法。

DTMF 数字信号的产生可以用查表的方法(同实验二)。

DTMF 信号的检测(解调)可以用 FFT 算法实现,也可以用滤波器。为了确定滤波器的指标,将各拨号键码的行列频率及差值列表,如表 8.3。从表中可见,“*”键的两个频率差值最小,这个频率是确定滤波器频带的依据。

表 8.3　拨号键码的频率差

键码	行频(Hz)	列频(Hz)	差频(Hz)
0	941	1336	395
1	697	1209	512
2	697	1336	639
3	697	1477	780
4	770	1209	439
5	770	1336	566
6	770	1477	707
7	852	1209	357
8	852	1336	484
9	852	1477	625
A	697	1633	936
B	770	1633	863
C	852	1633	781
D	941	1633	692
*	941	1209	268
#	941	1477	536

8.7 实验七:PWM 波发生器

1. 实验要求

采用 DSP 的 PWM 输出,实现有关专业利用 PWM 技术的设计。

2.PWM 技术原理

PWM(脉宽调制)技术是近年来新兴的数字化控制技术,其基本原理来源于采样控制理论中的结论:冲量相等而形状不同的窄脉冲加在惯性的环节上时,其效果基本相同。这里冲量即窄脉冲的面积,效果基本相同是指环节的输出响应基本相同。基于这个结论,我们可以用一系列等幅不等宽的脉冲来代替一个正弦半波。如图 8.14(a)所示,如果把正弦半波 N 等份,就可以把正弦半波看成 N 个彼此相连的脉冲序列所组成的波形。这些脉冲宽度相等,都等于 π/N,但幅值不等,且脉冲顶部不是水平直线,而是曲线,各脉冲的幅值按正弦规律变化。如果把上述序列利用相同数量的等幅而不等宽的矩形脉冲代替,使矩形脉冲的中点和相应正弦波部分的中点重合,且使脉冲和相应的正弦波部分面积(冲量)相等,由以上面积相等即等效原理,此 PWM 波形和正弦半波是等效的,而负半周也同理可得,如图 8.14(b)。当需要改变等效正弦波的幅值时,只要按照同一比例系数改变各脉冲的宽度即可。这里的脉冲宽度按正弦规律变化而和正弦信号等效的 PWM 波形,也称 SPWM(Sinusodial PWM)波形。

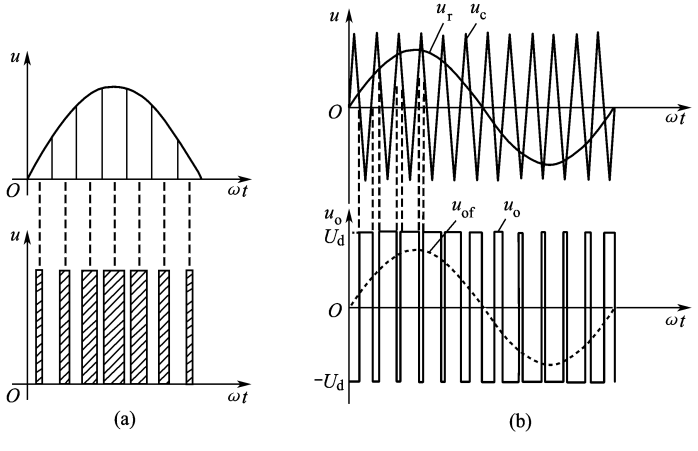


图 8.14 用 PWM 波代替正弦半波

3.SPWM 的产生

如果给出了所要得到的正弦波输出频率、幅值等,SPWM 的产生可以通过两种方法产生,一种为计算法,另一种为调制法。计算法是根据正弦波频率、幅值和半周期脉冲数,准确计算 PWM 波各脉冲宽度和间隔,就可得到所需 PWM 波形。这种方法不需要输入调制波,可以通过 DSP 内部的运算产生所需的调制信息,产生预知正弦波的 PWM 波。

调制法是将待输出波形作为调制信号,进行调制得到期望的 PWM 波,这种方法可以实时的跟踪所要调制的信号。

4. 产生带死区的 PWM

PWM 技术应用非常广泛。以在交流调速系统为例,目前用来控制功率管开关的 PWM 信号由于微处理器的速度和功能限制,多采用三个相差 120°的正弦调制信号与一个三角载波信号相比较的正弦调制法(SPWM)。随着 TI 公司于 1998 年中正式推出专门面向电机控制的新一代 DSP——TMS320F240 和 TMS320C240,由于其 DSP 内核 20MIPS 的高速处理能力和面向电机控制的专用外围设备,利用微处理器进行 PWM 波形的产生,从而使 PWM 波形的产生更加容易。本实验中,主要是应用 TMS320F240 自身提供的 PWM 输出功能。TMS320F240 可同时提供 12 路 PWM 波形输出,其中 9 路为独立。有三个具有可编程死区控制的全比较单元产生独立的三个 PWM 输出,死区时间最小一个 CPU 时钟周期。PWM 脉冲宽度及增减变化量最小为一个 CPU 时钟周期,最大 16 位的 PWM 分辨率,可编程产生非对称、对称和空间矢量 PWM 输出及比较和周期寄存器的自动重载功能使得用该 DSP 产生 PWM 非常方便。

5. 全比较单元 PWM 简介

全比较单元组成在第6章已做了详细介绍,三个全比较单中的每一个和事件管理器模块的通用计数器 1,死区单元,以及输出逻辑一起可以被用来产生一对 PWM 输出并且可编程为带有死区输出。图 8.15 为一个全比较单元 PWM 输出。

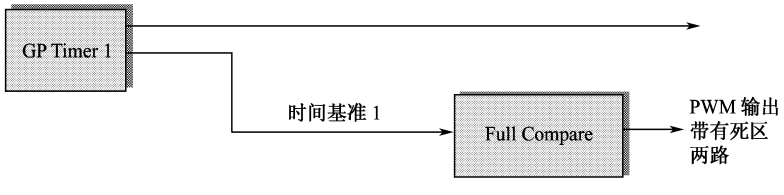


图 8.15

两路 PWM 输出都是基于时间基准 1 的,但是这两相 PWM 输出是彼此反相的,并且是带有死区的,这有利于防止电机控制和电力电子等系统中两个功率器件的同时开启而导致换相失败或击穿。输出波形将如图 8.16 所示。

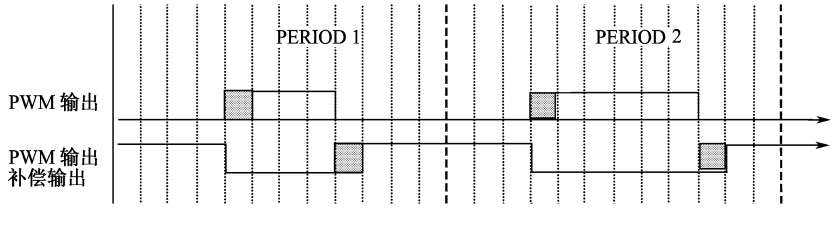


图 8.16　带有死区的互相补偿的 PWM 输出波形

6. 实验例程

产生 PWM 程序编制的主要内容有以下方面：

(1)比较单元程序编制主要有：初始化 ACTR 控制 6 个比较输出引脚、比较寄存器 1CMPR1、死去单元定时器控制寄存器 DBTCON 和比较控制寄存器 COMCON 等。

(2)用通用定时计数器 1 实现时间基准，编程主要包括：初始化定时计数器 1 周期寄存器 T1PR、定时器 1 计数器寄存器 T1CNT 和控制寄存器 T1CON(第六位为 0，定时器不工作)，编程选择计数模式、时钟预定标、时钟源、比较器重装数条件和使能比较操作；写 T1CON(第六位为 1，启动定时器)。

程序运行后，生成的波形如图 8.17 所示，包括一个输出，一个补偿输出，带有死区。

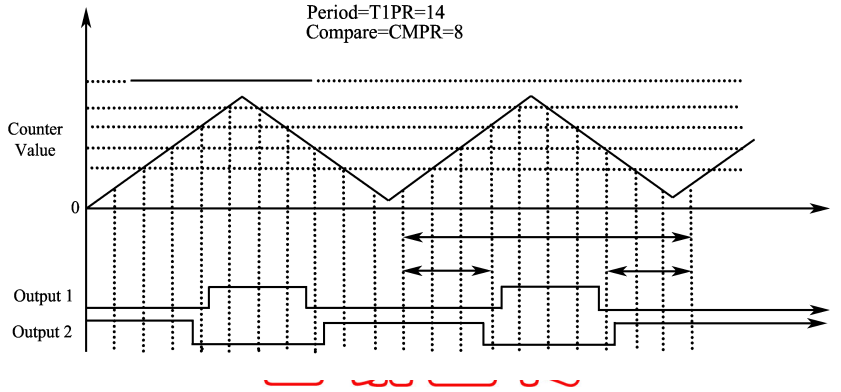


图 8.17 全比较输出波形

实验源程序：

```
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    .include        f240regs.h
    .bss        Vreflo,1      ; reference voltage——low
    .bss        Result,1;
    .bss        Vrefhi,1      ; reference voltage——high
    .bss        DRE,1;
    ; Vector address declarations
    ;————————————————————————————————————————————————
    .sect ".vectors"
RSVECT      B   START       ; Reset Vector
INT1        B   PHANTOM     ; Interrupt Level 1
INT2        B   PHANTOM     ; Interrupt Level 2
INT3        B   PHANTOM     ; Interrupt Level 3
INT4        B   PHANTOM     ; Interrupt Level 4
INT5        B   PHANTOM     ; Interrupt Level 5
INT6        B   PHANTOM     ; Interrupt Level 6
RESERVED    B   PHANTOM     ; Reserved
```

```
    SW_INT8     B   PHANTOM      ; User S/W Interrupt
    SW_INT9     B   PHANTOM      ; User S/W Interrupt
    SW_INT10    B   PHANTOM      ; User S/W Interrupt
    SW_INT11    B   PHANTOM      ; User S/W Interrupt
    SW_INT12    B   PHANTOM      ; User S/W Interrupt
    SW_INT13    B   PHANTOM      ; User S/W Interrupt
    SW_INT14    B   PHANTOM      ; User S/W Interrupt
    SW_INT15    B   PHANTOM      ; User S/W Interrupt
    SW_INT16    B   PHANTOM      ; User S/W Interrupt
    TRAP        B   PHANTOM      ; Trap vector
    NMINT       B   PHANTOM      ; Non-maskable Interrupt
    EMU_TRAP    B   PHANTOM      ; Emulator Trap
    SW_INT20    B   PHANTOM      ; User S/W Interrupt
    SW_INT21    B   PHANTOM      ; User S/W Interrupt
    SW_INT22    B   PHANTOM      ; User S/W Interrupt
    SW_INT23    B   PHANTOM      ; User S/W Interrupt
;=================================================
    ;MAIN  CODE- starts here
;=================================================
    .text
        NOP
    START: SETC INTM                ;初始化中断、时钟
        SPLK  #1000,IMR
        LACC  IFR
        SACL  IFR
        CLRC  SXM
        CLRC  OVM
        CLRC  CNF
        LDP   #00E0H
        SPLK  #00BBH,CKCR1      ;CPUCLK=20MHz
        SPLK  #00C3H,CKCR0
        SPLK  #40C0H,SYSCR      ;CLKOUT=CPUCLK
        SPLK  #006FH,WDCR
        KICK_DOG
        SPLK  #05H,Vrefhi
        SPLK  #00H,Vreflo
        SPLK  #00H,Result

        LDP   #232              ;初始化全比较单元和定时器 1
        SPLK  #666H,ACTR
        SPLK  #05E0H,DBTCON     ;死区定时器周期,并使能 3 个全比较单元死区定时器
        SPLK  #4B57H,COMCON
        SPLK  #0CB57H,COMCON
```

```
        SPLK   #8H,CMPR1
        SPLK   #14H,T1PR
        SPLK   #0H,T1CNT
        SPLK   #0A80AH,T1CON
        SPLK   #0A84AH,T1CON
        SPLK   #03FH,GPTCON
;INITIALIZE ADC MODULE
        LDP    #224
        SPLK   #8C04H,ADCTRL1
        SPLK   #05H,ADCTRL2;
        SPLK   #8C05H,ADCTRL1
        SETC   TC
;=============================================
PHANTOM KICK-DOG      ;Resets WD counter
        B      PHANTOM
```

8.8 实验八:自选设计实验

(1) EV 模块实验(参考文件:SPRA367、SPRA368、SPRA369、SPRA490、SPRA363、SPRA413、SPRA410、SPRA411、SPRA415、SPRA416、SPRA370)

(2) 数字滤波器设计

(3) 用 DSP 实现变频器

(4) 串口实验(参考文件:SPRA451、SPRA418)

(5) 显示、键盘及其接口

(6) D/A 接口、JTAG 接口,存储器、I/O 扩展

(7) 电机拖动应用,请访问 www.ti.com/sc/dmcsoftware

(8) DTMF/MFC 模块 (CPLD+DSP)

(9) 电力系统保护

(10)电力监控应用

(11)信息安全

(12)自选题目(DSP 在自己专业领域的应用)