

DSP 原理及电机控制应用

——基于 TMS320LF240x 系列

刘和平
邓 力 江 渝 郑群英 编著

北京航空航天大学出版社

内 容 简 介

介绍了TMS320LF240x DSP的硬件概况,内部资源、寻址方式、指令系统、程序设计和调试环境等。以TMS320LF240x 模块的原理和应用为主线,介绍了一个功能模块(或外设)的基本原理,并列举出相应的应用实例,给出应用电路原理接线图以及汇编程序和C 程序清单。使用C 语言或使用C语言与汇编语言混合编程开发DSP 控制应用程序,可以达到事半功倍的效果,在满足控制应用程序运行速度的基础上,可以更好地维护和移植程序。书中提供的所有程序均在作者设计的实验开发板上调试通过

本书可作为大学本科生和研究生的DSP 原理及应用课程的教材,也可作为 DSP 应用开发人员的参考用书。

图书在版编目(CIP) 数据

DSP 原理及电机控制应用: 基于TMS320LF240x 系列/
刘和平等编著. —北京: 北京航空航天大学出版社,
2006.10

ISBN 7-81077-819-6

I. D… II. 刘… III. ①数字信号—微处理器②数字信号—信号处理—应用—电机—控制 IV.①TP332
②TM301.2

中国版本图书馆CIP 数据核字(2002)第121455号

2006, 北京航空航天大学出版社, 版权所有。
未经本书出版者书面许可, 任何单位和个人不得以任何形式或手段复制或传播本书及其所附光盘内容。侵权必究。

DSP 原理及电机控制应用——基于 TMS320LF240x 系列
刘和平

编 著

邓 力 江 渝 郑群英
责任编辑 李 键

*

北京航空航天大学出版社出版发行

北京市海淀区学院路37号(100083) 发行部电话: 010-82317024 传真: 010-82328026

<http://www.buaapress.com.cn> E-mail: blpress@263.net

北京市松源印刷有限公司印装 各地书店经销

开本: 787×960 1/16 印张: 29 字数: 650千字

2006年10月第1版 2006年10月第1次印刷 印数: 5.000册

ISBN7-81077-819-6

定价: 42.00元(含光盘1张)

前言

TMS320LF240x 是定点的DSP 芯片，其功能强大的结构设计提供了低成本、低功耗和高性能的处理能力。它集成了对电机的数字化运动控制非常有用的先进外设，提供真正的单芯片数字信号控制器解决方案。与24x DSP 控制器芯片代码兼容的同时，240x 芯片具有处理性能更好(40 MIPS)，外设集成度更高，程序存储器更大以及 A/D 转换速度更快等特点。240x 芯片提供了多种类型，包含不同容量存储器和不同外设的芯片以满足各种应用要求。它拥有高达32K 字的FLASH 存储器系列，也包括ROM 存储器系列。其引脚是完全兼容的。

所有240x 芯片都至少有一个事件管理器模块，用于电机数字化控制应用。该模块包括中间和/或边缘对齐的 PWM 发生器；具有可编程的死区控制性能，以防止桥式驱动主电路的上下桥臂短路；还可实现同步 A/D 转换功能。如果带有双事件管理器(TMS320LF2407A)，那么就能用一个240x DSP 控制芯片对多个电机的和/或逆变器进行控制。

其高性能10位A/D 转换器(ADC) 的转换时间为500ns，并提供多达16路的模拟输入。它具有自动排序功能，使得最大为16路的转换在同一个转换期间进行而不会增加 CPU 的开销。

该系列所有的控制器都集成有串行通信接口(SCD)，使之能够与系统中的其他控制器进行异步通信。对于要求额外通信接口的系统，F2407 和 F2406 控制器提供了一个16位的同步串行外围接口(SPI)，它们还提供了符合 CAN2.0B 规范要求的CAN 通信接口。

本书在已出版的《TMS320LF240xDSP 结构、原理及应用》和《TMS320LF240xDSP 结构、原理及其C 语言应用》的基础上进行了修改：引入了最新的 CCS3.1 集成开发环境，增加了电机控制的内容、液晶显示器和数码管的接口，汇集了DSP 工程应用经验。多次反复修改设计的实验开发板已应用在工程项目中。

本书介绍了TMS320LF240x DSP的硬件概~~况~~、内部资源、寻址方式、指令系统、程序编写和开发环境等。编写体系以TMS320LF240x 模块的原理和应用为主线，介绍了一个功能模块(或外设)的基本原理，并列举出相应的应用实例，给出应用的电路原理接线图和汇编程序清单、C 语言程序清单。使用C 语言或使用C 语言与汇编语言混合编程开发 DSP 控制应用程序，可达到事半功倍的效果，在满足控制应用程序的运行速度基础上可更好地维护程序和移植程序。书中提供的所有程序均在重庆大学—美国德州仪器公司数字信号处理器解决方案实验室设计的实验开发板上调试通过。

本书在成书过程中得到了重庆大学电气工程学院电力电子与电力传动系的许多老师的大力帮助和支持，在此表示感谢。本书还得到了刘平、周有为、杨利辉、严利平、张学锋、卓清锋、洗成瑜和杨立勇等同学的帮助，他们为此做了大量的工作，在此一并表示感谢。

在这里还要感谢美国德州仪器公司大学计划项目所提供的大力支持。

限于编者的水平，书中难免存在错误和不当之处，恳请读者批评指正。

作 者
重庆大学
2006年7月

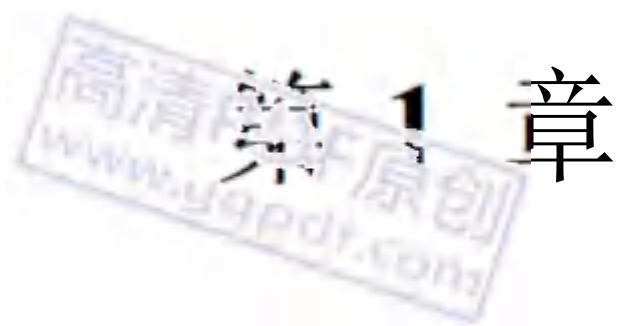
作者联系方式：

重庆市重庆大学电气工程学院 刘和平

邮编：400044

电话：023-65102438

Email:enginner@cqu.edu.cn



TMS320LF240x 概述

1.1 TMS320 系列DSP 概况

TMS320 系列包括：定点、浮点、多处理器 DSP 和定点 DSP 控制器。TMS320 系列 DSP 的体系结构专为实时信号处理而设计，该系列DSP 控制器将实时处理能力和控制器外设功能集于一身，为控制系统应用提供了一个理想的解决方案。下列特性使得 TMS320 系列成为很多信号处理及控制应用的理想选择：

- ＞灵活的指令集；
- ＞灵活的内部操作；
- ＞高速的运算能力；
- ＞改进的并行结构；
- ＞有效的成本。

目前 TI 公司主推的 DSP 有：定点系列 TMS320C2000、TMS320C5000，浮点系列 TMS320C6000。其中 TMS320C6000 系列中也有部分为定点 DSP。

TMS320 同一产品系列中的器件具有相同的 CPU 结构，但片内存储器和外设的配置不同。派生的器件集成了新的片内存储器和外设，以满足世界范围内电子市场的不同需求。通过将存储器和外设集成到控制器内部，TMS320 器件降低了系统成本，节省了电路板空间，提高了系统的可靠性。

1.2 TMS320LF240x 系列芯片概述

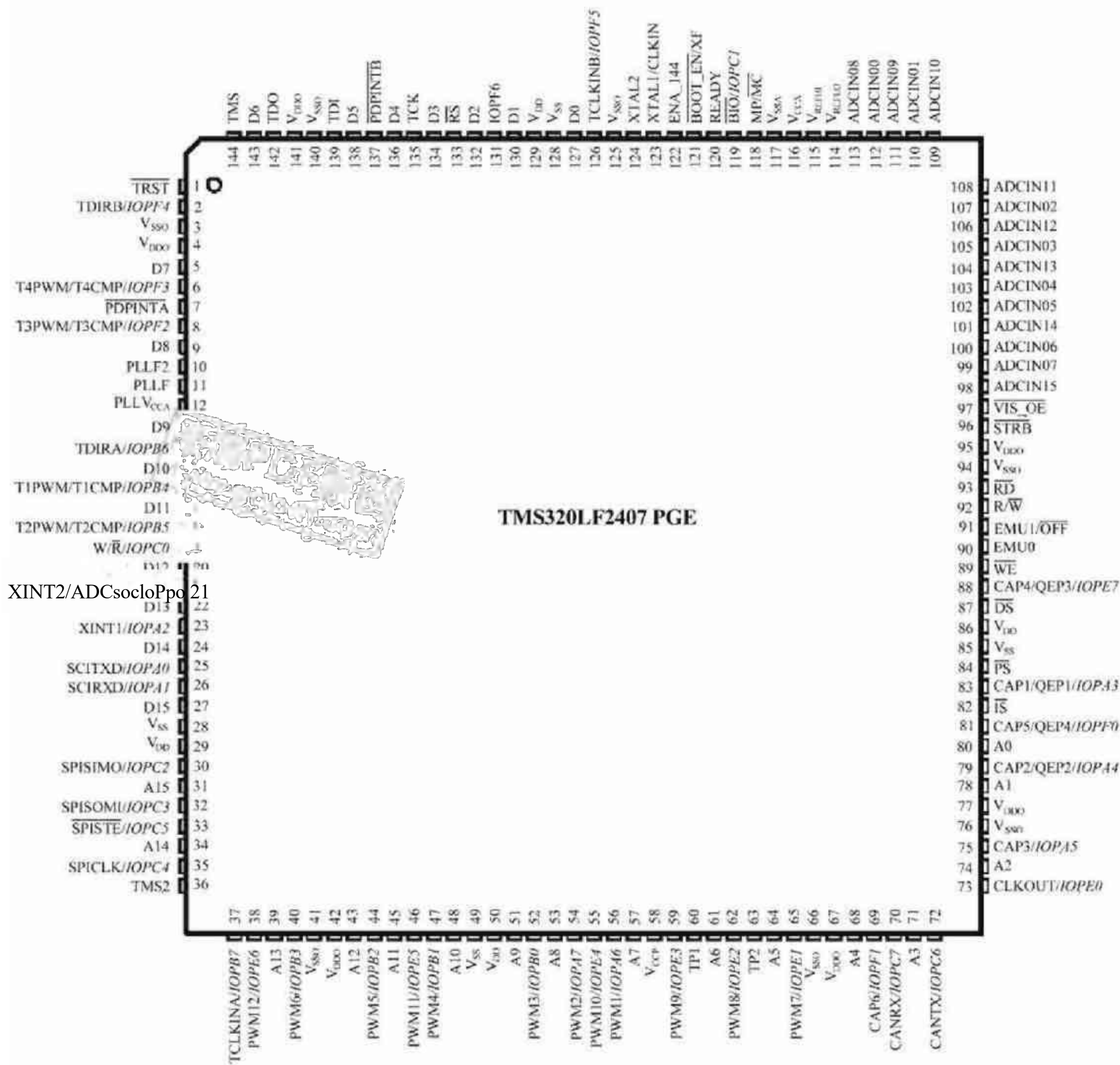
在 TMS320 系列 DSP 的基础上，TMS320LF240x（本书以后简称 F240x）系列 DSP 有以下特点：

- ＞采用高性能静态 CMOS 技术，使得供电电压降为 3.3 V，减小了控制器的功耗；30 MIPS 的执行速度使得指令周期缩短到 33 ns (30 MHz)，从而提高了控制器的实时控制能力。

- ＞ 基于 TMS320C2xx DSP 的 CPU 核，保证了 F240x 系列 DSP 代码与 TMS320 系列 DSP 代码兼容。
- ＞ 片内有高达 32K 字的 FLASH 程序存储器、高达 1.5K 字的数据/程序 RAM、544 字双口 RAM(DARAM) 和 2K 字的单口 RAM(SARAM)。
- ＞ 两个事件管理器模块 EVA 和 EVB，每个包括：两个 16 位通用定时器；8 个 16 位脉宽调制(PWM) 通道。它们能够实现：三相反相器控制，PWM 的对称和非对称波形，快速的 PWM 通道关闭(当外部引脚 PDPINTx 出现低电平时)，可编程的 PWM 死区控制(以防正上下桥臂同时输出触发脉冲)，3 个捕获单元，片内光电编码器接口电路，16 通道 A/D 转换器。事件管理器模块适用于控制交流感应电机、无刷直流电机、开关磁阻电机、步进电机、多级电机和逆变器。
- ＞ 可扩展的外部存储器(LF2407) 总共 192K 字空间：64K 字程序存储器空间、64K 字数据存储器空间和 64K 字 I/O 寻址空间。
- ＞ 看门狗定时器模块(WDT)。
- ＞ 10 位 A/D 转换器最小转换时间为 500 ns，可选择由两个事件管理器来触发的两个 8 通道输入 A/D 转换器或一个 16 通道输入 A/D 转换器。
- ＞ 控制器局域网络(CAN)2.0B 模块。
- 串行通信接口(SCI) 模块。
- 16 位串行外设(SPI) 接口模块。
- 基于锁相环的时钟发生器。
- ＞ 高达 40 个可单独编程或复用的通用 I/O 引脚(GPIO)。
- ＞ 5 个外部中断(两个电机驱动保护、复位和两个可屏蔽中断)。
- 电源管理包括 3 种低功耗模式，能独立地将外设器件转入低功耗工作模式。

1.3 TMS320LF240x 系列 DSP CPU 控制器的功能结构图

图 1.1~1.4 列出 F2407/2406/2402 的 PGE 封装图及 F2407 DSP CPU 控制器的功能结构图。



+ talicized pin names indicate pin function after reset,

图 1 . 1 F2407 的 PGE 封装图



t ualirete ames idiae pit hinatinufet peseL

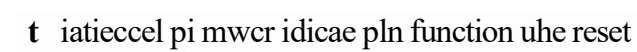
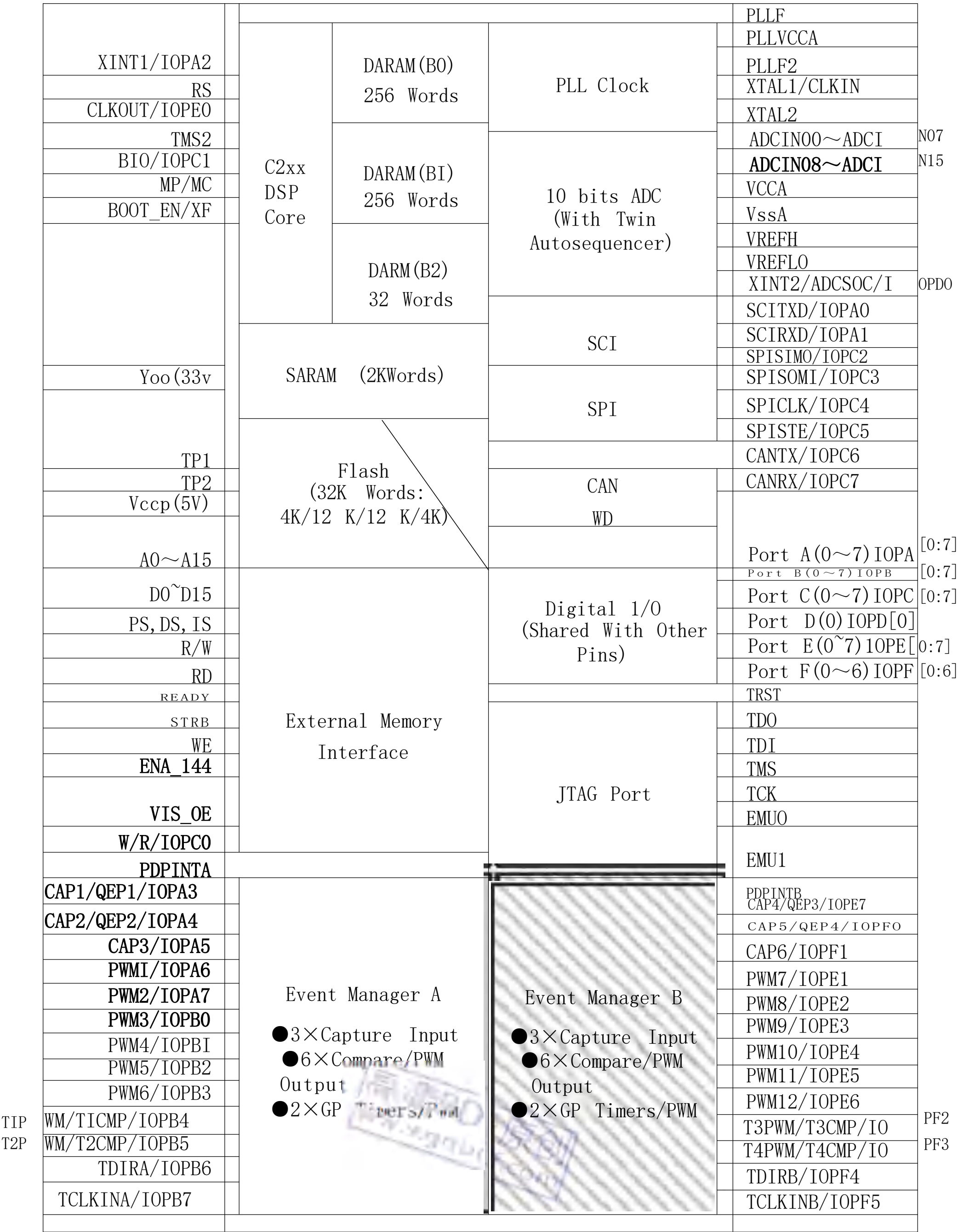


图1.2 F2406的PGE封装图



表示可选模式。
芯片型号不同，其存储器大小和外设模块会有变化。

图1.4 F2407 DSP控制器功能结构图

1.4 TMS320LF240x 系列DSP 引脚功能介绍

F2407 控制器是所有F240x 系列中功能最全的一种超集控制器，F2407 控制器上有该系列所有的信号。表1. 1列出了F240x 控制器上可用的所有信号及其功能描述。

表1.1 F240x控制器引脚列表

引脚名称	2407	2406	2402	功能描述
事件管理器A (EVA)				
CAP1/QEPI/IOPA3	83	57	4	捕捉输入#1/ 正交编码脉冲输入#1 (EVA) 或通用I/O(个)
CAP2/QEP2/IOPA4	79	55	3	捕捉输入#2/ 正交编码脉冲输入#2CEVA) 或通用I/O(个)
CAP3/IOPA5	75	52	2	捕捉输入#3 (EVA) 或通用I/O(个)
PWM1/IOPA6	56	39	59	比较/PWM输出引脚#1 (EVA) 或通用I/O(个)
PWM2/IOPA7	54	37	58	比较/PWM输出引脚#2 (EVA) 或通用I/O(个)
PWM3/IOPE	52	36	57	比较/PWM输出引脚#3 (EVA) 或通用I/O(个)
PWM4/IOPBI	47	33	54	比较/PWM输出引脚#4 (EVA) 或通用I/O(个)
PWM5/OPB2	44	31	53	比较PWM输出引脚#5 (EVA) 或通用I/O(个)
PWM6/IOPB3	40	28	50	比较/PWM输出引脚#6 (EVA) 或通用I/O(个)
T1PWM/TICMP. IOPB4	16	12	40	TMR1比较输出 (EVA) 或通用I/O(个)
T2PWM/T2CMP/IOPB5	18	13	41	TMR2比较输出 (EVA) 或通用I/O(个)
TDIRA/IOPB6	14	11		通用定时器计数方向选择 (EVA) 或通用I/O(个)。如果TDIRA=1. 则选择加计数；否则选择减计数(个)
TCLKINA/IOPB7	37	26	49	通用定时器 (EVA) 的外部时钟输入或通用I/O, 注意该定时器也可用内部时钟(个)
事件管理器B (EVB)				
CAP4/QEP3/IOPE7	88	60		捕捉输入#4/ 正交编码脉冲输入#3CEVB) 或通用I/O(个)
CAP5/QEP4/IOPF0	81	56		捕捉输入#5/ 正交编码脉冲输入#4 (EVB) 或通用I/O(个)
CAP6/IOPFI	69	48		捕捉输入#6 (EVB) 或通用I/O(个)
PWM7/IOPEI	65	45		比较/PWM输出引脚#7 (EVB) 或通用I/O(个)
PWM8/IOPF2	62	43		比较/PWM输出引脚#8 (EVB) 或通用I/O(个)
PWM9/IOPE3	59	41		比较/PWM输出引脚#9 (EVB) 或通用I/O(个)
PWM10/IOPE4	55	38		比较/PWM输出引脚#10 (EVB) 或通用I/O(个)
PWM ¹ 1/IOPE5	46	32		比较/PWM输出引脚#11 (EVB) 或通用I/O(个)
PWM12/IOPE6	38	27		比较/PWM输出引脚#12 (EVB) 或通用I/O(个)

续表1 . 1

引脚名称		2407	2406	2402	功能描述
T3PWM/T3CMP/IOPF2		8	7		TMR3比较输出 (EVB) 或通用I/O (个)
T4PWM/T4CMP/IOPF3		6	5		TMR4比较输出 (EVB) 或通用I/O (个)
TDIRB/IOPF4		2	2		通用定时器计数方向选择 (EVB) 或通用I/O。如果TDIRB=1. 则选择加计数； 否则选择减计数 (个)
TCLKINB/IOPF5		126	89		通用定时器 (EVB) 的外部时钟输入或通用I/O。注意该定时器也可用内部时钟 (个)
A/D转换器ADC					
ADCIN00		112	79	18	ADC的模拟输入#0
ADCIN01		110	77	17	ADC的模拟输入#1
ADCIN02		107	74	16	ADC的模拟输入#2
ADCIN03		105	72	15	ADC的模拟输入#3
ADCIN04		103	70	14	ADC的模拟输入#4
ADCIN05		102	69	13	ADC的模拟输入#5
ADCIN06		100	67	12	ADC的模拟输入#6
ADCIN07		99	66	11	ADC的模拟输入#7
ADCIN08		113	80		ADC的模拟输入#8
ADCIN09		111	78		ADC的模拟输入#9
ADCIN10		109	76		ADC的模拟输入#10
ADCIN11		108	75		ADC的模拟输入#11
ADCIN12		106	73		ADC的模拟输入#12
ADCIN13		104	71		ADC的模拟输入#13
ADCIN14		101	68		ADC的模拟输入#14
ADCIN15		98	65		ADC的模拟输入#15
VREFHI		115	82	20	ADC模拟输入高电平参考电压输入端
VREFLO		114	81	19	ADC模拟输入低电平参考电压输入端
VccA		116	83	21	ADC模拟供电电压 (3. 3V)
VssA		117	84	22	ADC模拟地
CAN SCI SPI					
CANRX/IOPC7	CANRX	70	49	—	CAN接收数据引脚或通用I/O (个)
	IOPC7	70	49	63	

续表1. 1

引脚名称		2407	2406	2402	功能描述
CANTXIOPC6	CANTX	72	50		CAN发送数据引脚或通用I/O(个)
	IOPC6	72	50	64	
SCITXD/IOPA0		25	17	43	SCI异步串行口发送数据引脚或通用I/O(个)
SCIRXD/IOPAI		26	18	44	SCI异步串行口接收数据引脚或通用I/O(个)
SPICLKIOPC	SPICLK	35	24		SPI时钟引脚或通用I/O(个)
	IOPC4	35	24	47	
SPISIMO/IOPC2	SPISIMO	30	21		SPI从动输入、主控输出引脚或通用I/O(个)
	IOPC2	30	21	45	
SPISOMI/IOPC3	SPISOM1	32	22		SPI从动输出、主控输入引脚或通用I/O()
	IOPC3	32	22	46	
SPISTE/IOPCS	SPISTE	33	23		SPI从动发送使能(可选)引脚或通用I/O()
	IOPC5	33	23		
外部中断					
RS		133	93	28	控制器复位。当RS使能时，控制器终止执行并使PC=0。当RS拉为高电平时，从程序存储器的0位置开始执行。RS影响相关的寄存器和状态位。当WDT定时时间溢出时，在RS引脚产生一个系统复位脉冲(个)
PDPINTA		7	6	36	功率驱动保护中断输入引脚。当电机驱动/电源逆变器不正常时，比如出现过电压、过电流等。该中断有效。将PWM输出引脚(EVA)置为高阻态。PDPINTA是一个下降沿有效的中断(个)
XINT1/IOPA2		23	16		外部用户中断1或通用I/O。XINT1、2都是边沿信号有效。边沿极性可编程(个)
XINT2/ADCSOC/IOPD		21	15	42	外部用户中断2可作A/D转换开始输入引脚或通用IO。XINT1、2都是边沿有效，边沿极性可编程(个)
CLKOUT/IOPE0		73	51	1	时钟输出或通用I/O。输出时钟为CPU时钟或监视定时器时钟，这由系统控制状态寄存器中的CLKSRC(bit 14)决定。当不用于时钟输出时，可用作通用I/O(个)
PDPINTB		137	95		功率驱动保护中断输入引脚。当电机驱动/电源逆变器不正常时，比如出现过电压，过电流等，该中断有效。将PWM输出引脚(EVB)置为高阻态。PDPINTB是一个下降沿有效的中断(个)
振荡器、PLL、FLASH、引导程序及其他					
XTAL1/CLKIN		123	87	24	PLL振荡器输入引脚，晶振或时钟源输入到PLL。该引脚接到参考晶振的一端

续表1 . 1

引脚名称	2407	2406	2402	功能描述
XTAL2	124	88	25	晶振、PLL振荡器输出引脚。该引脚接到参考晶振的一端。当EMU1/OFF引脚为低电平时，该引脚呈高阻态
PLLVccA	12	10	39	PLL电压 (3. 3V)
BOOTEN/XF	121	86	23	引导ROM使能，通用I/O, XF引脚。该引脚在复位期间被输入采样以更新SCSR1. 3 (BOOT_EN位), 然后驱动XF作为输出信号。复位之后，XF被置为高电平。BOOT_EN只能接无源回路 (个)
IOPF6	131	92		通用I/O引脚 (个)
PLL1F	11	9	38	锁相环外接滤波器输入1
PLL1F2	10	8	37	锁相环外接滤波器输入2
Vccp (5V)	58	40	60	FLASH编程电压输入引脚。在硬件仿真时，该引脚电平必须为5V, 在程序下载时，该引脚电平可为5V或0V; 在程序下载进DSP之后运行时，该引脚必须接地。在该引脚上不要使用任何限流电阻
TP1 (FLASH)	60	42	61	FLASH阵列测试引脚，悬空
TP2 (FLASH)	63	44	62	FLASH阵列测试引脚，悬空
BIO/IOPCI	119	85		分支控制输入引脚。由BCND pma, BIO指令查询该引脚电平，若为低，则执行分支程序。如果不用该引脚，则必须将其拉为高电平。所有控制器复位时，将该位配置为分支控制输入。当不用此功能时，该引脚可作通用I/O (个)
仿真和测试				
EMU0	90	61	7	带内部上拉仿真器I/O引脚#0。当TRST引脚拉高时，该引脚用作来自或到仿真器系统的中断，通过JTAG扫描可定义为I/O引脚 (个)
EMU1/OFF	91	62	8	仿真器引脚#1。该引脚可禁止所有输出。当TRST引脚拉高时，该引脚用作来自或到仿真器系统的中断，通过JTAG扫描可定义为I/O引脚。当TRST拉低时，该引脚设定为OFF引脚；当低电平有效时，所有输出引脚驱动为高阻态。注意. OFF只用于测试和仿真目的 (而不适用于多处理器应用), 因此，对于OFF状态有以下情况：TRST=0. EMU0=0, EMU1/OFF=0 (年)
TCK	135	94	29	带内部上拉JTAG测试时钟 (个)
TDI	139	96	30	带内部上拉JTAG测试数据输入。在TCK的上升沿从TDI输入的数据被锁存到选定的寄存器 (指令或数据) (个)
TDO)	142			JTAG扫描输出，测试数据输出。在TCK的下降沿，选定寄存器中的内容 (指令或数据) 被移出到TDO引脚 (个)

续表1. 1

引脚名称		2407	2406	2402	功能描述
TMS		144	100	32	带内部上拉的JTAG测试方式选择。该串行控制输入在TCK的上升沿锁存到TAP控制器中
TMS2					带内部上拉的JTAG测试方式选择2。该串行控制输入在TCK的上升沿锁存到TAP控制器中。仅用于测试和仿真。在用户应用中，该引脚可不接(个)
TRST		1	1	33	带内部下拉的JTAG测试复位。当TRST拉高时，扫描系统控制器的运行。若该信号引脚未接或为低电平，控制器运行在功能方式，并且测试复位信号无效()
地址、数据和存储器控制信号					
DS		87			数据空间选通引脚。IS、DS和PS总保持高电平，除非要用低电平请求访问相关的外部存储器或I/O空间。在复位、掉电和EMUI低电平有效期间，这些引脚为高阻态
Is		82			I/O空间选通引脚。TS、DS和PS总保持为高电平，除非要用低电平请求访问相关的外部存储器或I/O空间。在复位、掉电和EMUI低电平有效期间，这些引脚为高阻态
PS		84			程序空间选通引脚。IS、DS和PS总保持为高电平，除非要用低电平请求访问相关的外部存储器或I/O空间。在复位、掉电和EMUI低电平有效期间，这些引脚为高阻态
R/W		92			读/写选定信号。它指明了与外部装置通信期间信号的传送方向，通常情况下为读方式(高电平)，除非低电平请求执行写操作。当EMU1/OFF低电平有效和掉电期间，该引脚被置为高阻态
W/R/IOPCo	W/R	19			写/读选定或通用I/O。它是一个对“0等待状态”存储器接口很有用的反向传输读写信号。通常情况下为低电平，除非执行存储器写操作(个)
	IOPCa	19	14		
RD		93			读使能引脚。读使能表示一个有效的外部读周期，它对所有外部程序、数据和I/O读有效。当EMU1/OFF低电平有效时，该引脚被置为高阻态
WE		89			写使能引脚。该信号下降沿表示该控制器驱动外部数据线(D15-D0)，它对所有外部程序、数据和I/O写有效。当EMU1/OFE低电平有效时，该引脚被置为高阻态
STRB		96			外部存储器访问选通。该引脚总为高电平，除非插入一个低电平来表示一个外部总线周期，在访问片外空间时该信号有效。当EMU1/OFF低电平有效时和掉电期间，该引脚被置为高阻态

续表1 . 1

引脚名称	2407	2406	2402	功能描述
READY	120			访问外部设备时, READY被拉低来增加等待状态。它表示一个外部器件为将要完成的总线处理做好准备。若该外设未准备好, 则将READY拉为低电平(此时, 处理器将等待一个周期, 并且再次检测READY)。注意, 若要处理器执行READY检测, 程序至少要设定一个软件等待状态。为了满足外部READY时序要求, 等待状态发生控制寄存器(WSGR)至少要设定一个等待状态(年)
MP/MC	118			微处理器/微控制器方式选择引脚。复位期间该引脚若为低电平, 则工作在微控制器方式下, 并从内部程序存储器(FLASH EEPROM)的0000h开始执行程序; 若在复位期间为高电平, 则工作在微处理器方式下, 并从外部程序存储器的0000h开始执行程序。同时, 将MP/MC位(SCSR2寄存器的第2位)置位(↓)
ENA_144	122			高电平有效时, 使能外部接口信号。若为低电平, 则产生一个无效地址。该引脚内部下拉
VIS_OE	97			透视度(VIS)输出使能引脚(当数据总线输出时有效)。在透视度方式下, 在外部数据总线驱动为输出的任何时候, 该引脚有效(为低电平)。当运行在透视度方式下, 该引脚可用作外部编码逻辑, 以防止数据总线冲突
A0	80			16位地址总线的bit 0
A1	78			16位地址总线的bit 1
A2	74			16位地址总线的bit 2
A3	71			16位地址总线的bit 3
A4	68			16位地址总线的bit 4
A5	64			16位地址总线的bit 5
A6	61			16位地址总线的bit 6
A7	57			16位地址总线的bit 7
A8	53			16位地址总线的bit 8
A9	51			16位地址总线的bit 9
A10	48			16位地址总线的bit 10
A11	45			16位地址总线的bit 11
A12	43			16位地址总线的bit 12
A13	39			16位地址总线的bit 13

续表1．1

引脚名称	2407	2406	2402	功能描述
A14	34			16位地址总线的bit 14
A15	31			16位地址总线的bit 15
D0	127			16位数据总线的bit0(↑)
D1	130			16位数据总线的bit 1(个)
D2	132			16位数据总线的bit 2(个)
D3	134			16位数据总线的bit 3(个)
D4	134			16位数据总线的bit 4(个)
D5	138			16位数据总线的bit 5(个)
D6	143			16位数据总线的bit 6(个)
D7	5			16位数据总线的bit7(个)
D8	9			16位数据总线的bit 8(个)
D9	13			16位数据总线的bit 9(个)
D10	15			16位数据总线的bit 10(个)
D11	17			16位数据总线的bit 11()
D12	20			16位数据总线的bit 12()
D13	22			16位数据总线的bit 13(个)
DI4	24			16位数据总线的bit 14(个)
D15	27			16位数据总线的bit 15(个)

供电电源

VED	29, 50, 86. 129	20. 35, 59. 91	6, 27. 56	内核电源电压+3. 3 V; 数字逻辑电源电压
VDxo	4. 42. 67. 77. 95. 141	4. 30. 47. 54. 64. 98	I0. 35. 52	I/O) 缓冲器电源电压+3. 3V; 数字逻辑和缓冲器电源电压
Vss	28. 49. 85. 128	19. 34. 58. 90	5. 26. 55	内核地; 数字参考地
Vsso	3. 41. 66. 76. 95. 125. 140	3. 29. 46. 53. 63. 97	9, 34. 51	I/O) 缓冲器接地; 数字逻辑和缓冲器参考地

注：1. 斜体引脚名称表明复位后的引脚功能
2. 复位后. 所有的通用I/O 引脚为输入状态。
3.Vcex 与数字供电电压分开供电(VssA与数字地分开), 以达到ADC 的精确度并提高抗干扰能力。
4. 为了使控制器能够正常运行，所有电源引脚(Vpm.Vopo.Vss,Vsso) 必须正确连接，任一电源引脚都不能悬空。
5. “个”为内部上拉，“↓”为内部下拉(典型的上拉/下拉有效值为士16 μ A)。

1.5 TMS320LF240x 系列 DSP 存储器映射图

图1. 5~1. 7为 F240x 系列 DSP 存储器映射图。

Hex	Program	Hex	Data	Hex	I/O
0000	<u>FLASH SECTOR0 (4K)</u> Interrupt Vectors (0000~003Fh) Reserved° (0040~0043h) User Code Begins at 0044h	0000	Memory-Mapped Registers/Reserved Addresses	0000	External
0FFF 1000	FLASH SECTOR 1 (12K)	005F			
		0060			
		007F	On-Chip DARAMB2		
		0080	Illegal		
		00FF			
		0100	Reserved		
		01FF			
		0200	On-Chip DARAM (B0) (CNF=0) Reserved (CNF=1)		
		02FF			
3FFF 4000	FLASH SECTOR 2 (12K)	0300	On-Chip DARAM (B1) ④		
		03FF			
		0400	Reserved		
		04FF			
		0500	Illegal		
		07FF			
		0800	SARAM (2K) Internal (DON=1) Reserved (DON=0)		
		0FFF			
		1000	Illegal		
6FFF 7000	<u>FLASHSECTOR3 (4K)</u>	6FFF			
		7000	Peripheral Memory-Mapped Registers (System, WD, ADC, SCI, SPI, CAN, IO, Interupts)		
		7FFF			
		8000			
87FF 8800	SARAM (2K) Internal (PON=1) External (PON=0)				
FDFE FE00	External				
FDFE FE00	Reserved² (CNF=1) External (CNF=0)				
FEFF FF00	On-Chip DARAM (B0)² (CNF=1) External (CNF=0)				
FFFF		FFFF		FFFF	

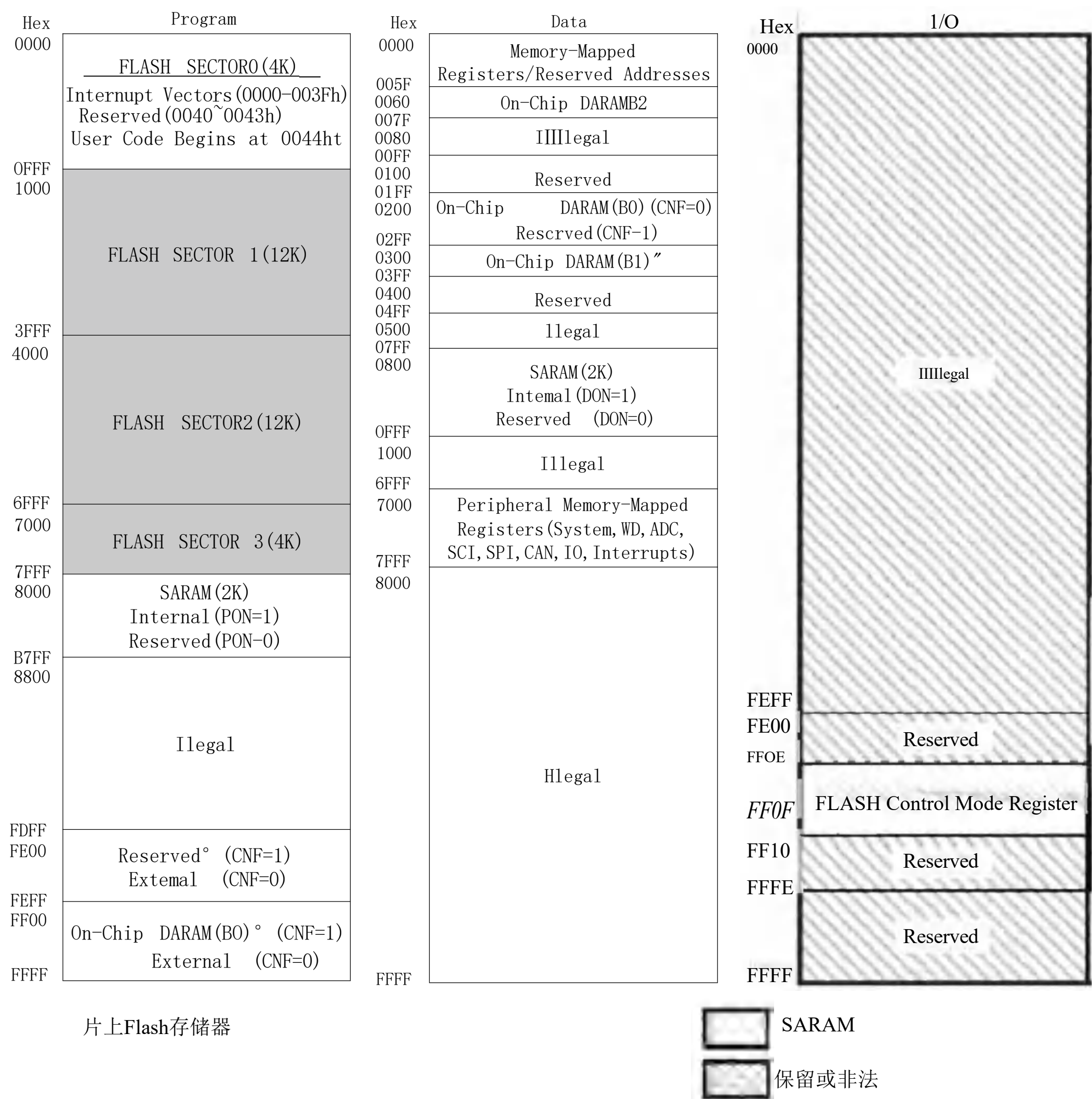
如果MP/MC=0, 则映射到片上Flash存储器
如果MP/MC=1, 则映射到片外程序存储器



注：如果使能引导ROM, 则程序存储器空间的0000h~00FFh 将被引导ROM 占用。

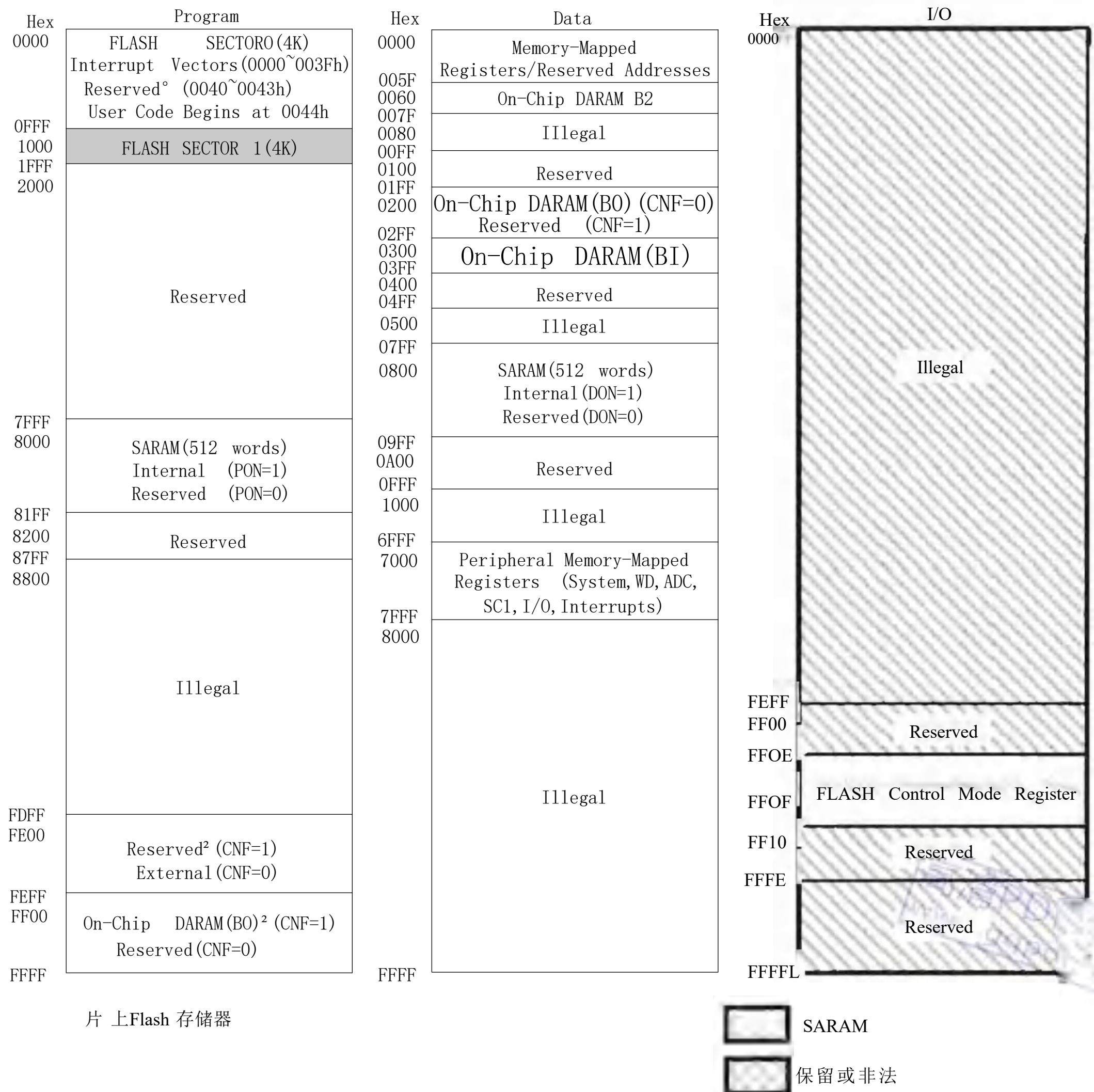
- ① 程序存储器空间的0040h~0043h 为保留空间。
- ② 当 CNF=1 时, 地址FE00h~FEFFh 和 FFO0h~FFFFh 被映射到程序存储器空间的同一物理区(B0)。例如：向FE00h写数和向FF00h写数具有同样的效果。为了简化起见, 将地址段FE00h~FEFFh 看做保留区间。
- ③ 当 CNF=0 时, 地址0100h~01FFh 和0200h~02FFh 被映射到数据存储器空间的同一物理区(B0)。例如：向0100h写数和向0200h写数具有同样的效果。为了简化起见, 将地址段0100h~01FFh 看做保留区间。
- ④ 地址0300h~03FFh 和0400h~04FFh 被映射到数据存储器空间的同一物理区(B1)。例如：向0300h 写数和向0400h 写数具有同样的效果。为了简化起见, 将地址段0400h~04FFh 看做保留区间。

图1. 5 F2407 的存储器映射图



注：此图的图注同图1.5

图1.6 F2406 的存储器映射图

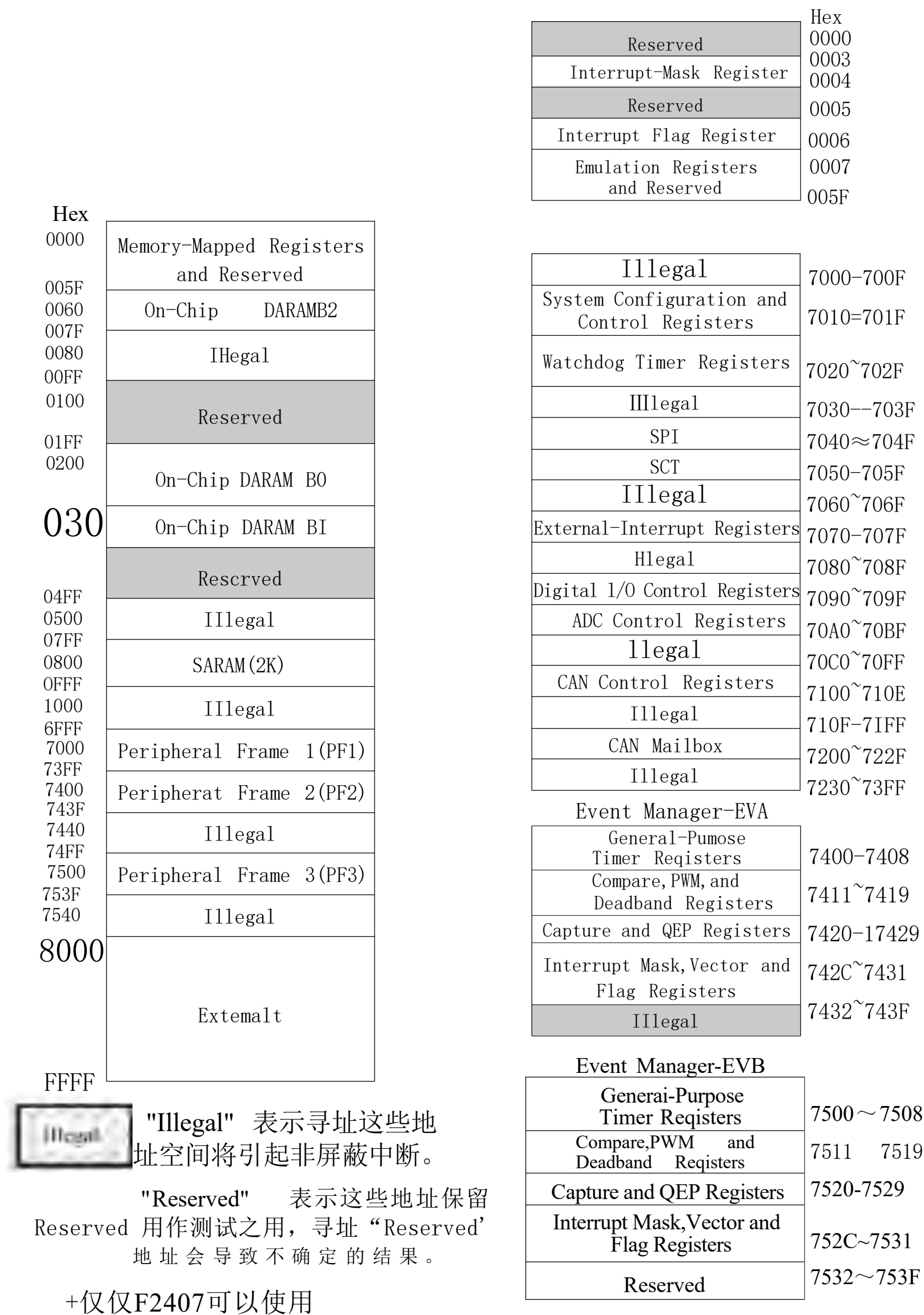


注：此图的图注同图1.5。

图1.7 F2402 的存储器映射图

1.6 TMS320LF240x 系列DSP 外设存储器映射图

F240x DSP 外设存储器映射图如图1.8所示。



Illegal

"Illegal" 表示寻址这些地址空间将引起非屏蔽中断。

Reserved

"Reserved" 表示这些地址保留

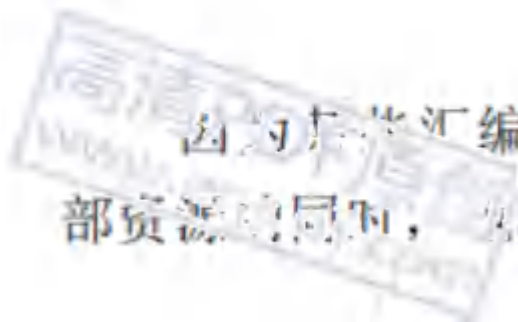
Reserved 用作测试之用，寻址“Reserved”地址会导致不确定的结果。

+仅仅F2407可以使用

图1.8 F240x 外设存储器映射图

第 2 章 TMS320LF240x 系 列 DSP

内部资源介绍



因为汇编语言指令与DSP 的内部硬件结构紧密相关，所以本章在着重介绍DSP 内部资源的同时，也将介绍部分汇编语言指令。

2.1 TMS320LF240x 系 列DSP 的 CPU内部功能模块介绍

图2. 1为 F240x 的 CPU 内部硬件功能模块图。表2. 1为图2. 1的符号说明。

表2. 1 F240x DSP 的 CPU 内部硬件功能模块图符号说明

符 号	名 称	说 明
ACC	累加器	一个32位寄存器，用来保存CALU计算结果，并为下一次CALU操作提供输入，它具有移位和循环操作功能
ARAU	辅助寄存器算术单元	一个无符号、16位算术单元间接寻址时，用辅助寄存器算术单元来计算辅助寄存器地址
AUX REGS	辅助寄存器0~7	这些16位寄存器可用作指针，指向数据存储空间范围内的任何地址。它们面向ARAU单元操作，由辅助寄存器指针(ARP)选定。ARO可用来作为更新ARx(x为1~7)参考值，也可作为ARx的比较值
C	进位	寄存器进位位由CALU输出，C被反馈到CALU单元，用于扩展运算操作。C位位于状态寄存器1(ST1), 其状态可通过条件指令测试。C位也可用于累加器移位和循环
CALU	中央算术逻辑单元	TMS320C2xx核的32位主要算术逻辑单元。CALU在一个单机器周期内执行32位操作。它对来自ISCALE或PSCALE的数据和来自ACC的数据进行运算，并将运算后的状态结果存于PCTRL单元
DARAM	双 口 RAM	如果片内RAM配置控制位(CNF)被设置为0, 那么可配置的双口RAM(DARAM)块B0被映射到数据存储空间；否则，B0被映射到程序存储空间。块B1和B2分别映射到地址分别为0300h~03FFh和0060h~007Fh的数据存储器空间。块0和1的容量为256字，而块2的容量为32字
DP	数据存储器页面指针	9位DP寄存器与一个指令字的低7位(LSBs)一起形成一个16位的直接寻址地址。DP值可由LST和LDP指令改变

续表2. 1

符 号	名 称	说 明
GREG	全局存储器配置寄存器	GREG指定全局数据存储器的空间大小。由于F240x器件没有使用全局存储器空间, 这个寄存器被保留下来
IMR	中断屏蔽寄存器	IMR寄存器的各位分别屏蔽或使能对应的7个中断
IFR	中断标志寄存器	IFR的7位分别指示TMS320C2xx已进入7个可屏蔽的中断中的任意一个中断
INT#	中断陷阱	总共有32个可通过硬件和/或软件产生的中断
1SCALE	输入数据定标移位器	一个16~32位的桶式左向移位器。ISCALE能将输入的16位数据的0 16位在本周期内, 向左移位以得到32位输出, 因此输入定标移位操作不需要额外的周期
MPY	乘法器	经过16×16位乘法运算得到一个32位的乘积, MPY可在一个单周期内完成乘法运算。MPY进行有符号或者无符号的二进制补码乘法运算
著2M STACK	微堆栈	当程序地址生成逻辑电路用来在数据存储空间内生成连续的地址时, MSTACK单元为下一条指令的地址提供暂存空间
MUX	多路选择器	从多个总线输入中选择一个输入
NPAR	下一条程序地址寄存器	NPAR单元保存卜一指令周期出现在程序地址总线上的程序地址
OSCALE	输出数据定标移位器	一个16~32位桶式左向移位寄存器。OSCALE将32位累加器输出左移 0~7位, 以实现数据的归一化管理, 并把移位后32位数据的高16位或低 16位输出到数据写总线 (DWEB)
PAR	程序地址寄存器	PAR保存当前程序地址总线上出现的程序地址, 保存多个指令周期, 直到 CPU完成当前总线周期所排定的所有存储器操作
PC	程序计数器	PC计数器每次将NPAR的值增1, 以便为下一次取指操作和数据传送操作提供地址
PCTRL	程序控制器	PCTRL, 对指令进行译码、管理流水线、储存状态位, 并对条件操作指令进行译码
PREG	乘积寄存器	一个保存16×16乘法结果的32位寄存器
PSCALE	乘积定标移位器	将乘积结果左移0位、1位、4位或右移6位。左移可得到由二进制补码乘法运算产生的附加标志位; 右移可用来将数字量按比例地减少, 以防止 CALU内的乘积累加溢出; PSCALE输入连接到乘法器, 输出连接到 CALU或者数据写总线 (DWEB)。移位操作不需要额外的指令周期
STACK	堆栈	STACK是一块存储器单元, 用来存储子程序和中断服务程序的返回地址, 或存储数据。C2xx堆栈为16位宽, 8级深
TREG	临时寄存器	这是一个16位寄存器, 保存乘法运算操作数中的一个。TREG保存 LACT, ADDT和SUBT指令的移位个数; 还保存BITT指令测试位的位置

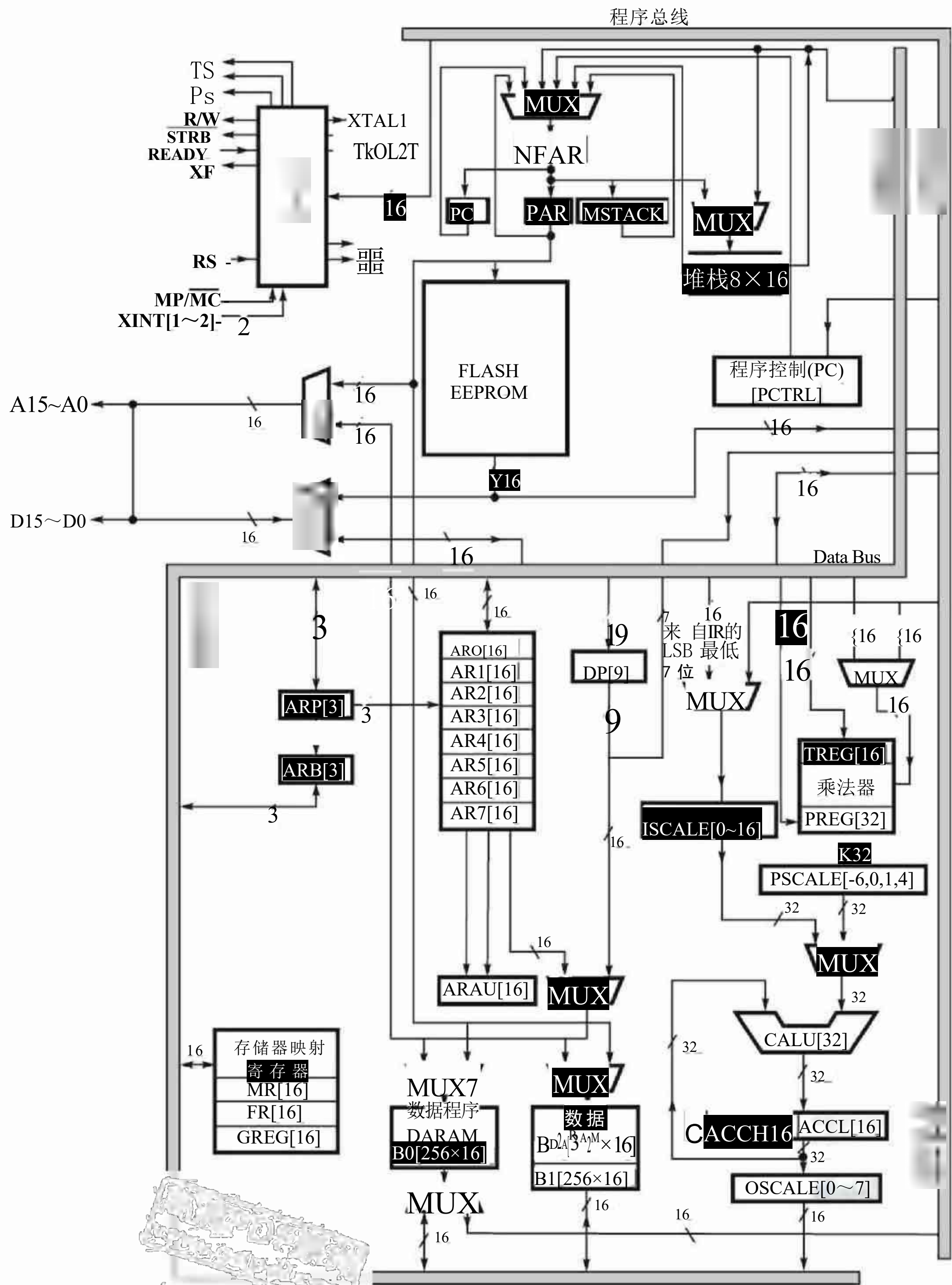


图 2.1 F240x 的 CPU 功能模块图

CPU 的输入定标器、中央算术逻辑单元和乘法器单元的功能框图，见图2.2。

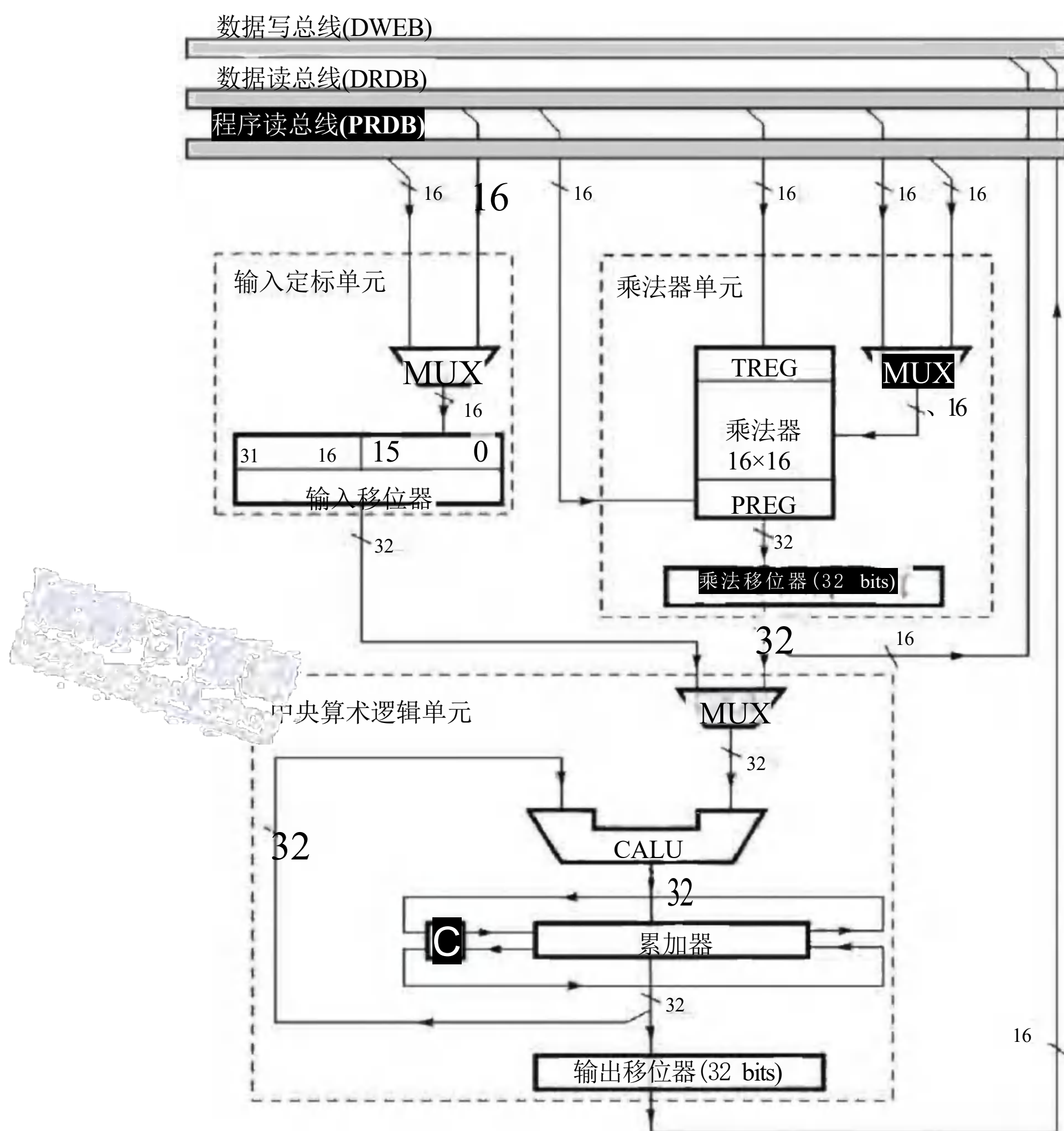


图2.2 CPU 的输入定标器、中央算术逻辑单元和乘法单元的功能框图

2.1.1 输入定标移位器

F240x 器件提供了一个输入定标移位器，该移位器将来自程序存储器或数据存储器的16位数据调整为32位数据送到中央算术逻辑单元(CALU)。因此其16位输入与数据总线相连，32位输出与CALU 单元相连。由于输入定标移位器作为从程序/数据存储空间到CALU 间数据传输路径的一部分，因此不会占用时钟开销。该移位器在算术定标，以及在逻辑操作对屏蔽定位设置中非常有用。

输入定标移位器对输入数据进行0~15位左移。左移时，输出的最低有效位LSB 段填0；

最高有效位 MSB 根据状态寄存器 ST1 的 SXM(符号扩展方式)位的值来决定是否需要进行符号扩展。当SXM 位为1时，高位进行符号扩展；当 SXM 位为0时，高位填0。移位量由包含在指令字中的常量或由临时寄存器 TREG 指定。由指令字指定的移位量允许用户用特定的数据定标或调整运算来得到特定的代码。基于临时寄存器 TREG 的移位允许动态调整数据定标因子，从而可以适应不同的系统要求。

2.1.2 乘法器

F240x 系列DSP 采用一个 16×16 位的硬件乘法器，可以在单个机器周期内产生一个32位乘积结果的有符号或无符号数。除了MPYU 指令(无符号乘法指令)，所有的乘法指令都执行有符号的乘法操作。即被相乘的两个数都作为2的补码数，而运算结果为一个32位的2的补码数。乘法器接收两个输入：一个来自16位临时寄存器(TREG)；另一个通过数据读总线(DRDB) 来自数据存储器，或通过程序读总线(PRDB) 来自程序存储器。两个输入值相乘后，32位乘积结果存放在32位乘积寄存器(PREG) 中。PREG 的输出连接到乘积定标移位器(PSCALE)，通过乘积定标移位器，将乘积结果从 PREG 送到CALU 或数据存储器。

PSCALE 对乘积结果采用4种乘积移位方式，见表2.2。移位方式由状态寄存器 ST1 的乘积移位方式位(PM) 指定。这些移位方式对于执行乘法/累加操作，进行小数运算，或者进行小数乘积的调整都是很有用的。

表2.2 PSCALE 乘积移位方式

PM	SHIFT	说 明
00	没有移位	乘积结果没有移位地送到CALU单元或数据总线
01	左移1位	移去在一次2的补码乘法运算中产生的1位附加符号位，以得到一个Q31的乘积
10	左移4位	当与一个13位的常数相乘时，移去在 16×13 位的2的补码乘法运算中产生的4位附加符号位，以生成一个Q31的乘积
11	右移6位	对乘积结果进行定标，以使得运行128次的乘积累加而累加器不会溢出

注：Q31 格式是一种二进制小数格式，该格式在二进制小数点的后面有31个数字。

乘积结果可以进行1位左移，以补偿在两个16位2的补码数相乘(MPY 指令)时产生的附加符号。在MPY 指令中有1个短立即数(13位或者更少)时，左移4位操作可以消除1个16位数与1个13位立即数，相乘产生的4个附加的符号位。右移方式将乘积右移6位，使连续执行128次乘法/累加运算而不会溢出。注意，PREG 的内容一直保持不变，其值被复制到乘积定标移位器中，并在乘积定标移位器中进行移位。

通常LT (装载 TREG) 指令将数据从数据总线载入 TREG，以提供1个操作数；而MPY (乘法)指令提供第2个操作数，该操作数也从数据总线上得到。当使用 MPY 指令时，乘法运算也可对一个13位的立即数进行操作。然后，每两个指令周期得到1个乘积结果。当代码执

行多路乘法运算和乘积求和运算时，CPU 支持 TREG 流水线装载操作通过 CALU 得到的前次乘积结果。与装载TREG 操作并行运行的流水线操作包括：把PREG 的值装载入 ACC(用 LTP 指令)；把PREG 的值加载到 ACC (用LTA 指令)；把PREG 的值加到 ACC，移位 TREG 输入数据(用DMOV 指令)到数据存储器的下一地址(用 LTD 指令)；从 ACC 中减去 PREG 的值(用LTS 指令)。

乘且累加并带动数据移动指令 MAC 和 MACD 充分利用了乘法器的计算宽度，允许两个操作数被同时处理。在每个指令周期，这些操作的数据都可通过程序和数据总线传送到乘法器中。然后采用重复指令 RPT，使单周期乘法/累加操作变得更容易。在这些指令中，数据表地址由程序地址产生逻辑(PAGEN) 产生，而数据地址由数据地址产生逻辑(DAGEN) 产生。从而允许重复指令从数据表中连续取值，以及在任一间接寻址方式下获取数据。

当重复执行时，MACD 指令支持滤波器设计(加权平均)，所以随着乘积连加的执行，采样数据被移入存储器，为下一次采样数据腾出空间，同时丢掉最前面一次的采样值。

无符号乘法运算 MPYU 指令大大方便了用于扩展精度的算术运算。TREG 中的无符号数与指定地址的数据存储器中无符号数相乘，相乘后的结果置于PREG 中。这个操作允许将位数超过16位的操作数分解成一些16位的字分别处理，以产生位数超过32位的乘积结果。平方并累加前次乘积指令 SQRA 和平方并减去前次乘积指令 SQRS. 传送相同的数到乘法器的两个输入端，以对一个数据存储器单元的值进行平方运算。

在32位乘积结果装入32位乘积结果寄存器PREG 后，通过执行储存乘积的高字节指令 SPH 和储存乘积的低字节指令 SPL,PREG 寄存器中的乘积可传送到CALU 单元或数据存储器。注意，不管把PREG 寄存器中的乘积传送至CALU 单元还是数据总线，都要通过乘积定标移位器(PSACLE)。

由于在数据传送过程中，PSACLE 的值受PM 位段所规定的乘积移位方式的影响，所以在执行中断服务程序前，必须对 PREG 的值进行保存。可通过执行“MPY #0”指令将 PREG 寄存器清0。当把被保存的低半字装入 TREG 寄存器并执行一条“MPY#1” 指令后，乘积寄存器中的值可恢复。再通过LPH 指令，高半字也可装入 TREG 寄存器。

2.1.3 中央算术逻辑部分

中央算术逻辑部分的主要构成有：

- 中央算术逻辑单元(CALU)——实现大动态范围内的算术和逻辑运算；
- >32位累加器(ACC)——接收来自CALU 的输出，并且可根据进位位(C) 的值实现移位；
- >输出定标移位器——将累加器的高位字和低位字在送入数据存储器之前进行移位。

1. 中央算术逻辑单元

中央算术逻辑单元实现许多算术和逻辑运算功能，且大多数的功能都只需1个时钟周期。

这些运算功能包括：16位加、16位减、布尔运算、位测试以及移位和旋转功能。

由于CALU 可以执行布尔运算，因此使得控制器具有位操作功能。CALU 的位移动和旋转在累加器中完成。CALU 之所以被称为中央算术逻辑单元，是因为它是一个独立的算术单元，与后面要介绍的辅助寄存器算术单元(ARAU) 在程序执行时是完全不相关的两个模块。

CALU 有两个输入：一个由累加器提供，另一个由乘积定标移位器或输入数据定标移位器提供。当CALU 执行完一次操作后，将结果送至32位累加器，由累加器对其结果进行移位。累加器的输出连至32位输出数据定标移位器。经过输出数据定标移位器，累加器的高、低16位字可分别被移位或存入数据寄存器。

对绝大多数指令，状态寄存器 ST1 的第10位符号扩展位(SXM) 决定在 CALU 计算时，是否使用符号扩展：若SXM 为0，符号扩展不使能；若SXM 为1，则符号扩展使能。

2. 累加器

一旦 CALU 中的运算完成，其结果就被送至累加器，并在累加器中执行单个移位或循环操作。累加器的高位字和低位字中的任意一个可以被送至输出定标移位器。在此定标移位后，又被存至数据存储器。以下为一些与累加器有关的状态位和转移指令。

与累加器有关的状态位有4个，它们是：

(1) 进位位(C)—— 状态寄存器 ST1 的第9位。下述情况之一影响进位位：

① 加至累加器或从累加器减：

>C=0 当减结果产生借位时；当加结果未产生进位时。

特例：当ADD 指令和移位16位一同使用，但无进位产生时，该指令对 C 位无影响。

>C=1 当加结果产生进位时；当减结果未产生借位时。

特例：当SUB 指令和移位16位一同使用，但无借位产生时，该指令对C 位无影响。

② 将累加器数值移位1位或循环移位1位。在左移或循环左移的过程中，累加器的最高有效位被送至C 位；在右移或循环右移过程中，累加器的最低有效位被送至C 位。

(2) 溢出方式位(OVM)—— 状态寄存器 STO 的第11位。OVM 决定累加器如何反映算术运算的溢出。当累加器处于溢出方式(OVM=1) 且有溢出发生时，累加器被填充下列两个特定值之一：

>若为正溢出，则累加器被填充最大正数7FFF FFFFh；

>若为负溢出，则累加器被填充最大负数80000000h。

当OVM=0 时，累加器中的结果正常溢出。

(3) 溢出标志位(OV)—— 状态寄存器 STO 的第12位。当未检测到累加器输出时，OV 未被锁存，其值为0；当溢出发生时，OV 置1且被锁存。

(4) 测试/控制标志位(TC)—— 状态寄存器 ST1 的第11位。根据被测试位的值，该位被置0或1。

与累加器有关的转移指令大都取决于位C、OV、TC的状态和累加器的数值。详细介绍见

第3章。

3. 输出定标移位器

输出定标移位器存储指令中，指定的位数将累加器输出的内容左移0~7位，然后用SA-CH 或 SACL 指令将移位器的高位字或低位字存至数据存储器中。在此过程中，累加器的内容保持不变。

2.1.4 辅助寄存器算术单元

CPU 中还包括辅助寄存器算术单元 (ARAU)，该算术单元完全独立于中央算术逻辑单元。ARAU 的主要功能是在 CALU 操作的同时，执行对辅助寄存器 (AR7~AR0) 上的算术运算。这8个辅助寄存器提供了强大而灵活的间接寻址能力，利用包含在辅助寄存器中的16位地址，可访问64K 数据空间中的任意单元。关于间接寻址的细节，请参见第3章。图2.3给出了 ARAU 和相关逻辑。

为选择一特定的辅助寄存器，须向状态寄存器 STO 中的3位辅助寄存器指针 (ARP) 中装入0~7 的数值。可通过 MAR 指令或 LST 指令把装载 ARP 作为主要操作来执行，也可通过任何支持间接寻址的指令把装载 ARP 作为辅操作来执行。其中 MAR 指令仅用于修改辅助寄存器和 ARP，而 LST 指令可通过数据读总线 DRDB 把一个数据存储器的值装入 STO。

由 ARP 所指定的寄存器被当作当前辅助寄存器或当前 AR。在一条指令的执行过程中，当前辅助寄存器存放被访问的数据存储器的地址。若指令需要从数据存储器读数，则 ARAU 将该地址送至数据读地址总线 DRAB；若指令需要向数据存储器写数，则 ARAU 将该地址送至数据写地址总线 DWAB。当指令使用完该数据值以后，当前辅助寄存器的内容可以被 ARAU 增加或减小。ARAU 可以实现无符号的16位算术运算。

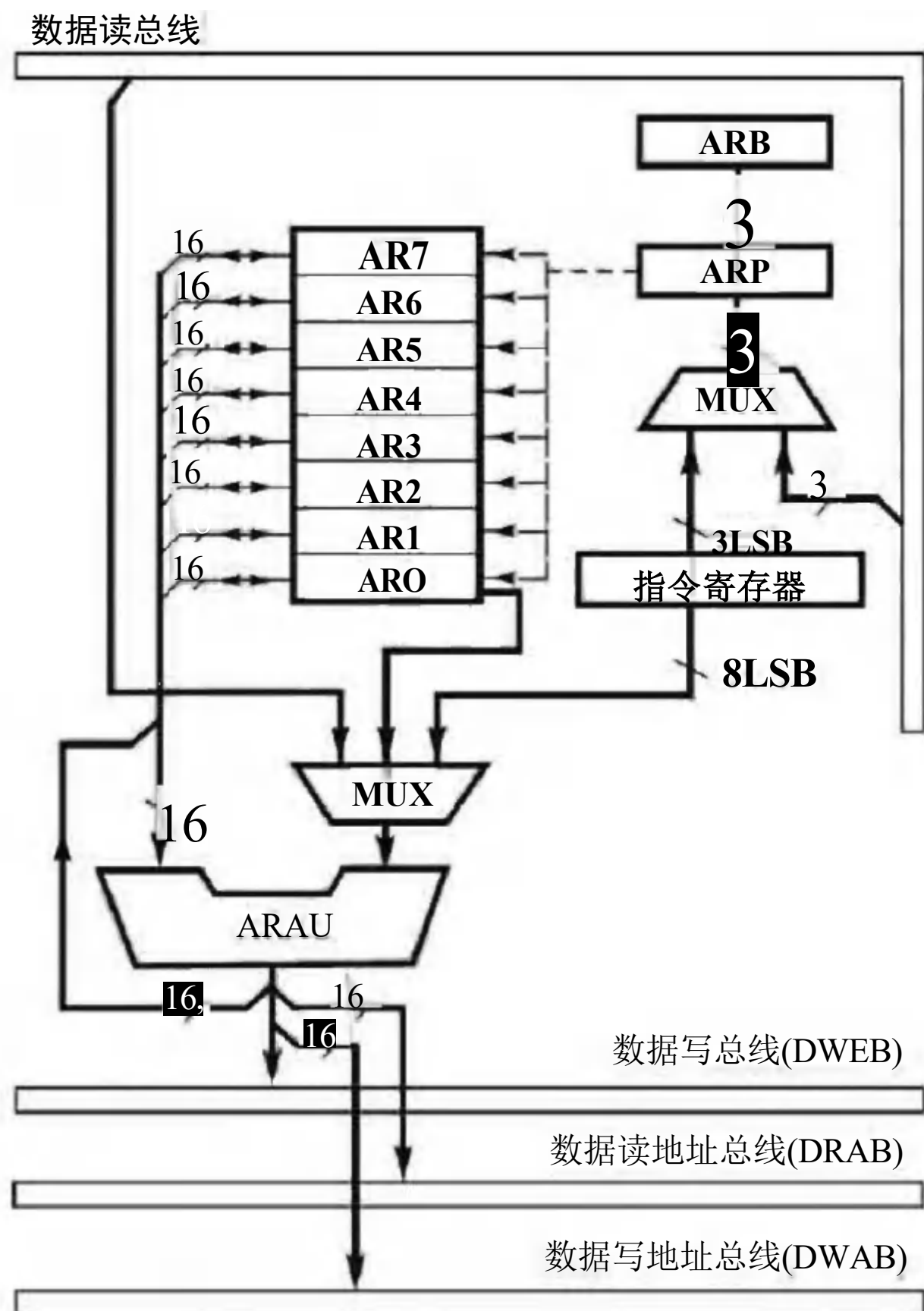


图2.3 ARAU 和相关逻辑

以下介绍 ARAU 和辅助寄存器功能。 ARAU 可执行下述操作：

- 通过执行任何一条支持变址的指令，将辅助寄存器值增加或减少1，也可以增加或减少1个变址值。
- 将1个常数加至辅助寄存器值(ADRK 指令)，或从辅助寄存器值中减去1个常数(SBRK 指令)。该常数取自指令字的8位最低有效位。
- 比较ARO 和当前 AR 的内容，然后将比较结果放至状态寄存器 ST1 中的测试/控制位 TC(CMPR 指令)。结果经由数据写总线 DWEB 传送至 TC。

通常，在流水线的译码阶段(即当指令所指明的操作正被译码时)，ARAU 执行其算术运算。从而使下一条指令译码之前，能够产生本条指令的地址。但有一种情况例外：在处理 NORM 指令时，是在流水线的执行过程中完成对辅助寄存器的修改的。

辅助寄存器除了被用作数据存储器地址的存储器外，还可用作其他用途。

- 通过CMPR 指令，利用辅助寄存器支持条件转移、调用和返回。 CMPR 指令将 ARO 的内容与当前AR 的内容进行比较，并将比较结果存至状态寄存器 ST1 中的测试/控制位 TC。
- 将辅助寄存器作为暂存单元。例如，使用 LAR 指令向辅助寄存器装入数值，以及使用 SAR 指令将辅助寄存器值存至数据存储器。
- 将辅助寄存器用作软件计数器，根据需要将其增加1或减1。

2.1.5 状态寄存器 STO 和 ST1

F240x 系列DSP 有两个状态寄存器 STO 和 ST1，含有各种状态和控制位，是应用中特别重要的两个寄存器。其内容可被保存到数据存储器，或从数据存储器读出加载到 STO 和 ST1，从而在子程序调用或进入中断时，实现 CPU 各种状态的保存。

加载状态寄存器指令 LST 将数据写入 STO 和 ST1，而保存状态寄存器指令 SST 则将 STO 和 ST1 的内容读出并保存起来。 INTM 位不受LST 指令的影响。当采用SETC 指令和 CLRC 指令时，这些寄存器的每一位都可被单独置1或清0。

图2.4给出了状态寄存器 STO 和 ST1 的结构，指出了每个寄存器包含的所有状态位。这些状态寄存器中的一些位保留未用，读出时为逻辑1。表2.3列出了状态寄存器每位字符的意义。

15

13

12

11

10

9

8

0

STO

ARP

QV

OVM

1

INTM

DP

15

13

12

11

10

9

8

7

6

5

4

3

2

1

0

ST1	ARB	CNF	TC	SXM	C	1	1	1	1	XF	1	1	PM

图2.4 状态寄存器 STO 和 ST1

表2.3 状态寄存器ST0 和 ST1 的字段定义

符 号	功 能
ARB	辅助寄存器指针缓冲器：当ARP被装载入ST0时，除了在LST指令外，原有的ARP值将被复制到ARB中；当通过“LST #1”指令装载ARB时，也把相同的ARB值复制到ARP
ARP	辅助寄存器指针：ARP选择间接寻址时，当前的辅助寄存器AR。当ARP被装载时，原有的ARP值被复制到ARB寄存器中。在间接寻址时，ARP可由存储器相关指令改变，也可由LARP、MAR和LST指令改变。当执行“LST#I”指令时，ARP也可载入与ARB相同的值
C	进位位：此位在加法结果产生进位时被置为1，或在减法结果产生借位时被清0；否则，除了执行带有I6位移位的ADD或SUB指令外，C在加法后被清除或在减法后被设置。在ADD或SUB指令时，ADD仅可对进位位进行置位，而SUB仅可对进位位进行清除，而不会对进位位产生其他影响。移位工位和循环指令也可影响进位位C，并且SETC、CLRC和LST指令也可影响C。条件转移、调用和返回指令可根据C的状态进行执行，复位时C被置1
CNF	片内DARAM配置位：若CNF=0，可配置的双口RAM区被映射到数据存储空间；若CNF=1，可配置的双口RAM区被映射到程序存储空间。CNF位可通过“SETC CNF”、“CLRC CNF”和LST指令修改。RS复位时，CNF置为0
DP	数据存储器页指针：9位DP寄存器与一个指令字的低7位一起形成一个16位直接寻址地址。可通过LST指令和LDP指令对其修改
INTM	中断模式位：当INTM被置0时，所有未屏蔽的中断使能；当它被置1，所有可屏蔽的中断禁止。可通过“SETC INTM”指令和“CLRC INTM”指令将INTM位置1或清0；RS中断也可对INTM进行设置；INTM位对不可屏蔽中断RS和溢出中断没有影响，注意INTM位不受LST指令的影响，复位时该位置1；在处理可屏蔽中断时，该位被置为1
OV	溢出标志位：该位保存一个被锁存的值，用以指示CALU中是否有溢出发生；一旦发生溢出，OV位保持为1，直到下列条件中的一个发生时才能被清除—复位、溢出时条件转移、无溢出时条件转移指令或LST指令
OVM	溢出方式位：当位OVM=0时，累加器中结果正常溢出；当OVM=1时，根据溢出的情况，累加器被设置为它的最大正值或负值。SETC指令和CLRC指令分别对该位进行置位和复位，也可用LST指令对OVM进行修改
PM	乘积移位方式：若PM=00，乘法器的32位乘积结果不移位，直接装入CALU；若PM=01，PREG输出左移一位后载入CALU，最低位1SB以0填充；若PM=10，PREG输出左移4位后载入CALU，最低位段LSB以0填充；若PM=11，PREG输出进行符号扩展右移6位。注意，PREG中的内容是一直保持不变的。当把PREG中的内容传送到CALU单元中时，发生移位操作。PM可由SPM指令和“LST#1”指令加载。复位时，PM位清0

续表2.3

符 号	功 能
SXM	符号扩展方式位：当SXM=1时，数据通过定标移位器传送到累加器时，将产生符号扩展；当SXM=0时，将抑制符号扩展。SXM位对某些指令没有影响。例如，ADDS指令将抑制符号扩展，而不管SXM位的状态。SXM可通过“SETC SXM”指令或“CLRC SXM”指令对其置位或复位，并且“LST#1”指令将对SXM位进行加载。复位时，SXM置1
TC	测试/控制标志位：在下述情况之一，TC位被置1,即由BIT或BITT指令测试的位为1。当利用NORM指令测试时，累加器的两个最高有效位“异或”功能为“真”。条件转移、调用和返回指令可根据TC位的条件来执行。BIT、BITT、CMPR、LST和NORM指令影响TC位
XF	XF引脚状态位：该位决定XF引脚的状态。“SETCXF”指令可对位XF进行置位，而“CLRC XF”指令可对其进行清0。复位时，XF置1

2.2 存储器和 I/O 空间

F240x 系列**DSP** 的设计基于增强的哈佛结构。它可以通过3组并行总线访问存储空间。它们分别是：程序地址总线(PAB)、数据读地址总线(DRAB) 和数据写地址总线(DWAB)。其中的任意一组可访问不同的程序空间，以实现不同的器件操作。由于总线工作是独立的，所以可同时访问程序和数据空间。在一个给定的机器周期内，CALU 可以执行多达3次的并行存储器操作。

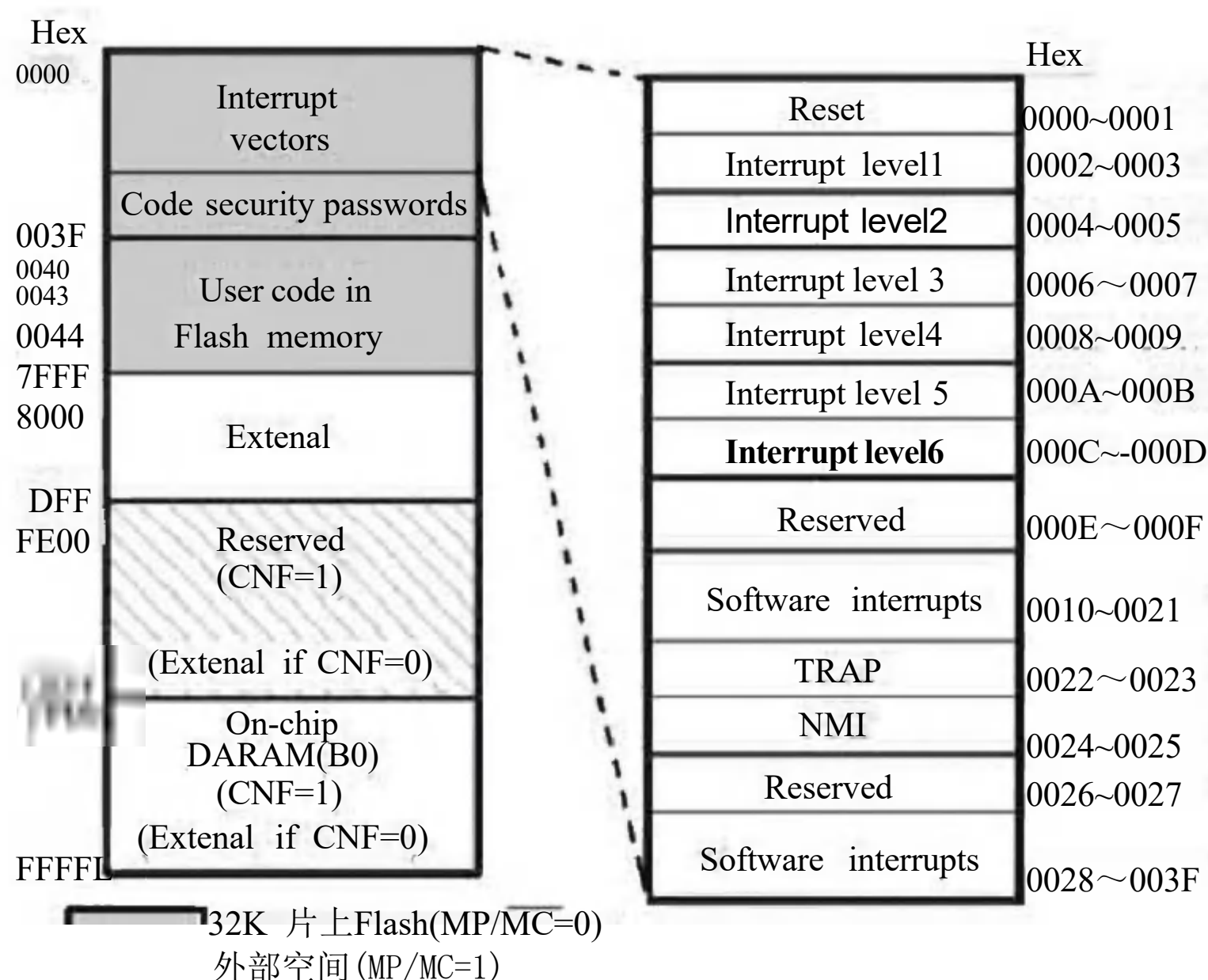
F240x 系列**DSP** 的地址映象被组织为3个可独立选择的空间：程序存储器(64K 字)、数据存储器(64K 字)及输入/输出(I/O) 空间(64K 字)。这些空间提供了共192K 字的地址范围。

2.2.1 程序存储器

程序存储器用于保存程序代码以及数据表信息和常量。程序存储器的寻址空间为64K 字，这包括片内 DARAM 和片内 FLASH EEPROM/ROM。当访问片外程序地址空间时，**DSP** 自动产生一个访问外部程序地址空间的信号 PS。图 2.5 为 F2407 的存储器映射图。有两个因素决定程序存储器的配置。

- >CNF 位 CNF 为 0 时，B0 块被映射为片外程序空间；CNF 为 1 时，B0 块被映射为片内程序空间。
- >MP/MC 引脚 MP/MC 引脚为0时，器件被配置为微控制器方式，可访问片内ROM 或 FLASH，器件从片内程序存储器中读取复位向量；MP/MC 引脚为1时，器件被配置为微处理器方式，禁止使用片内 Flash 器件从外部程序存储器中读取复位向量。无论

MP/MC 为何值，F240x 系列 DSP 都从程序存储器的0000h 单元读取复位向量。只有带外部程序存储器接口的器件才有 MP/MC 引脚。



注：图中Reserved 为保留区间

图2.5 F2407 的程序存储器映射图

2.2.2 数据存储器

数据存储器的寻址范围高达64K 字。每个器件都有3个片内DARAM 块：B0、B1 和 B2 块。B0 块既可配置为数据存储器，也可配置为程序存储器；B1 和 B2 块只能配置为数据存储器。图2.6为 F2407 的存储器映射图。

存储器可以采用两种寻址方式：直接寻址和间接寻址。当使用直接寻址时，按128字(称作数据页)的数据块对数据存储器进行寻址。图2.7显示了这些块如何被寻址。全部64K 的数据存储器包含512个数据页，其标号范围为0~511。当前页由状态寄存器 STO 中的 9 位数据页指针(DP) 的值来确定。因此，当使用直接寻址指令时，用户必须事先指定数据页，并在访问数据存储器的指令中指定偏移量。

1. 第0页数据页地址映射

数据存储器中包括存储器映射寄存器，它们位于数据存储器的第0页(地址0000h~007Fh)，见表2.4。用户必须注意以下几点。

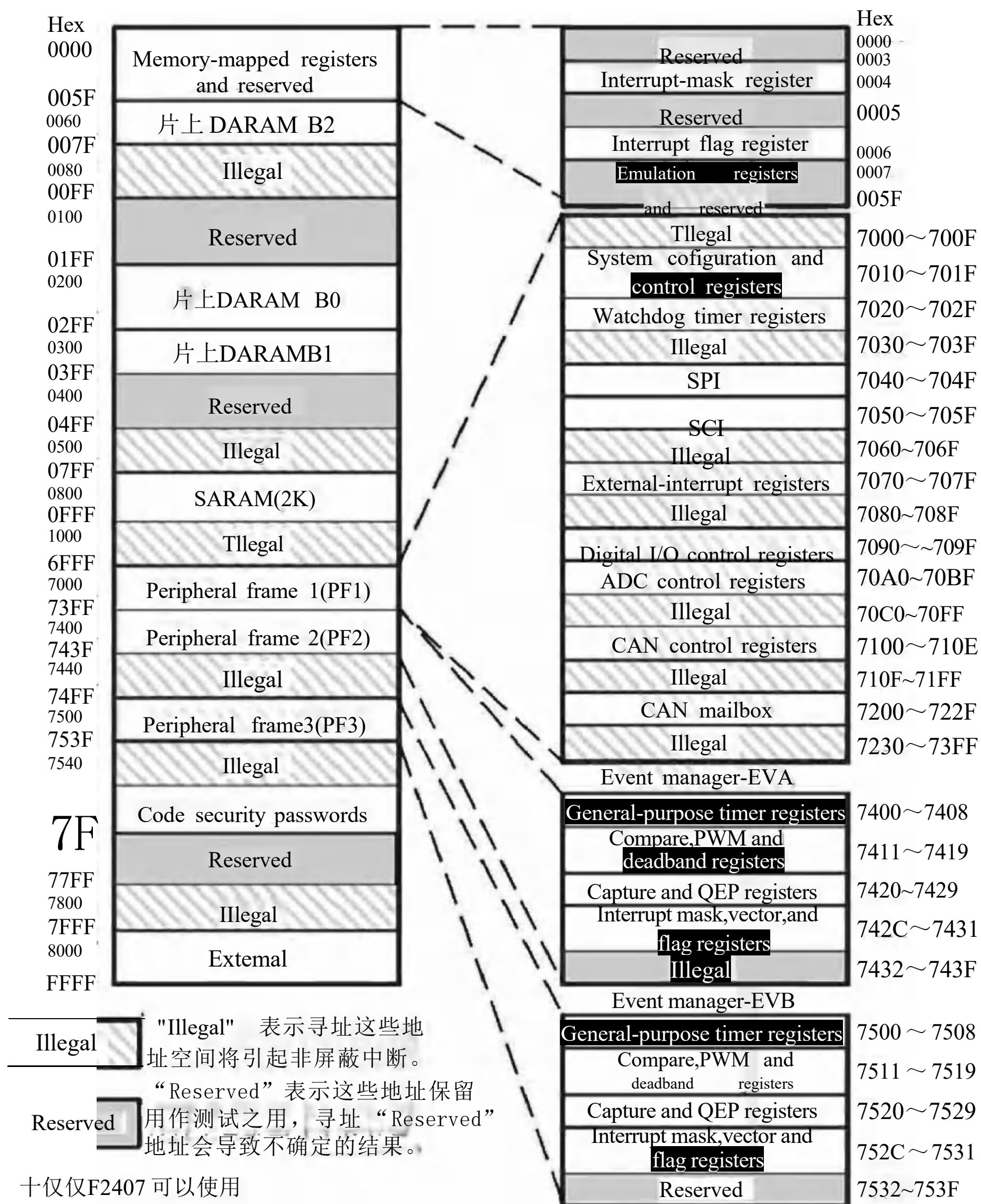


图2.6 LF2407 的数据存储器映射图

(1) 可以零等待状态访问两个映射寄存器：中断屏蔽寄存器(IMR) 和中断标志寄存器(IFR)。

(2) 测试/仿真保留区被测试和仿真系统用于特定信息的发送，因此不能对测试/仿真地址进行写操作；否则可能使器件改变其工作方式，从而影响正常工作。

(3) 32个字的B2 块用于变量的存储，同时又不会弄碎较大的内部和外部RAM 块。此处RAM 支持双口访问操作，且可用任何数据存储器寻址方式寻址。

表2 . 4 第0页数据页地址映射

地 址	名 称	描 述	地 址	名 称	描 述
0000h--0003h		保留	0023h-0027h		保留
0004h	IMR	中断屏蔽寄存器	002Bh~002Fh		保留用作测试和仿真
0005h		保留	0060h--007Fh	B2	双口RAM (DARAM B2)
0006h	IFR	中断标志寄存器			

2. 数据存储器配置

CNF 位决定数据存储器的配置：CNF 为 0 时，BO 块被映射为片外程序空间；CNF 为 1 时，BO 块被映射为片内程序空间。复位时BO 块被配置为数据存储空间。

2.2.3 I/O 空间

I/O 空间存储器共可寻址64K 字，图2. 7给出了F2407 的1/O 空间地址映射。



图2 . 7 F2407I/O 空间地址映射图

2.3 系统配置和中断

本节主要介绍系统的配置寄存器和与 F240x 系列 DSP 有关的中断。主要内容有系统配置寄存器、中断优先级和中断向量表、外设中断扩展控制器(PIE)、中断向量、中断响应的流程、中断响应的延时、CPU 中断寄存器、外设中断寄存器、复位、无效地址检测以及外部中断控制寄存器。

2.3.1 系统配置寄存器

1. 系统控制和状态寄存器 1 SCSR1)—— 地址7018 h

15	14	13	12	11	10	9	8
保留位	CLKSRC	LPM1	LPM0	CLKPS2	CLKPS1	CLKPS0	保留位
R_0	RW_0	RW_0	RW_0	RW_1	RW_1	RW_1	R_0
7	6	5	4	3	2	1	0
ADC CLKEN	SCI CLKEN	SPI CLKEN	CAN CLKEN	EVB CLKEN	EVA CLKEN	保留位	ILLADR
RW_0	RW_0	RW_0	RW_0	RW_0	RW_0	R_0	RC_0

注: R= 可读, W= 可写, C=清除, 0=复位值。

位 15 保留位。

位14 CLKSRC。CLKOUT 引脚源选择位。

0 CLKOUT 引脚输出 CPU 时钟;

1 CLKOUT 引脚输出 WDCLK 时钟。

位13~12 LPM(1:0)。 低功耗模式选择位。这两位声明了CPU 在执行IDLE 指令时进入哪一种低功耗方式。

00 CPU 进入 IDLE1(LPM0)模式;

01 CPU 进入 IDLE2(LPM1) 模式;

1x CPU 进入 HALT(LPM2) 模式。

位11~9 PLL 时钟预定标选择位。这3位对输入时钟选择 PLL 倍频系数, 见下表。

CLKPS2	CLKPS1	CLKPS0	系统时钟频率	CLKPS2	CLKPS1	CLKPS0	系统时钟频率
0	0	0	$4 \times f_m$	1	0	0	$0.8 \times f_m$
0	0	1	$2 \times f_m$	1	0	1	$0.66 \times f_m$
0	1	0	$1.33 \times f_m$	1	1	0	$0.57 \times f_m$
0	1	1	$1 \times f_m$	1	1	1	$0.5 \times f_m$

注: f。为输入时钟频率。

- 位 8 保留位。
- 位 7 ADC CLKEN。ADC 模块时钟使能控制位。
0 禁止 ADC 模块的时钟(即：关断 ADC 模块，以节约能量)；
1 使能 ADC 模块的时钟，且正常运行。
- 位 6 SCI CLKEN。SCI 模块时钟使能控制位。
0 禁止SCI 模块的时钟(即：关断 SCI 模块，以节约能量)；
1 使能SCI 模块的时钟，且正常运行。
- 位 5 SPI CLKEN。SPI模块时钟使能控制位。
0 禁止 SPI 模块的时钟(即：关断 SPI 模块，以节约能量)；
1 使能SPI 模块的时钟，且正常运行。
- 位 4 CAN CLKEN。CAN 模块时钟使能控制位。
0 禁止CAN 模块的时钟(即：关断CAN 模块，以节约能量)；
1 使能CAN 模块的时钟，且正常运行。
- 位 3 EVB CLKEN。EVB模块时钟使能控制位。
0 禁 止EVB 模块的时钟(即：关断EVB 模块，以节约能量)；
1 使 能EVB 模块的时钟，且正常运行。
- 位 2 EVA CLKEN。EVA 模块时钟使能控制位。
0 禁止EVA 模块的时钟(即：关断 EVA 模块，以节约能量)；
1 使能EVA 模块的时钟，且正常运行。

注意：要修改或读取任何外设寄存器的内容时，必须通过向相应的位写1来使能到外设的时钟。

- 位 1 保留位。
- 位 0 ILLADR。 无效地址检测位。

2. 系统控制和状态寄存器 2 (SCSR2) ——地 址 7 0 1 9h

15~8							
保留位							
RW_0							
7	6	5	4	3	2	1	0
保留位	I/P QUAI.	WD () VERRIE	XMIF HIZ	BOOT EN	MP / MC	DON	PON
RW_o		RC1	RW_	RW-BO(T EN引脚RW-MP/MC引脚		RW_1	RW_1

- 位15~7 保留位。这些位的读出值不确定，写入值无影响。
- 位 6 I/P QUAL。输入时钟限定器。输入时钟限定器限定输入到 F240x 器件的 CAP1~6、XINT1~2、ADCSOC 以 及PDPINTA/B 引脚上的信号。如果这些引脚作I/O 功能使用，则不会使用输入时钟限定电路。当这些引脚使用特殊

功能时，该位规定了输入到这些引脚上的信号，需要被锁存的最少脉冲宽度。脉冲只有达到此宽度之后，内部的输入状态才改变。

0 输入限定器锁存脉冲至少5个时钟周期；

1 输入限定器锁存脉冲至少11个时钟周期。

注意：虽然CLKOUT 与内部时钟的频率一样，但两者并不完全相同。

位 5 **WD OVERRIDE**。WD 保护位。复位后的缺省值为1, 从而使用户可以通过用软件将 WDCR 寄存器中的 WDDIS 位置1, 来禁止WD 工作。该位是一个只能清0位，通过向该位写入1来对其清0。

0 使用户不能通过软件来禁止 WD。 该位是一个只能清0位，不能通过软件来置1；

1 复位时的缺省值，使用户能够通过软件来禁止 WD 工作。

位 4 **XMIF HI-Z**。XMIF 高阻控制位。该位控制外部存储器接口信号(XMIF)。

0 所有XMIF 信号处于正常驱动模式(即非高阻态)；

1 所有XMIF 信号处于高阻态。

位 3 **BOOT EN**。BOOT 使能脚。该位反映了BOOT EN/XF 引脚在复位时的状态，该位可被软件改变。

0 使能引导 ROM， 地址空间0000h~00FFh 被片内引导 ROM 块占用。这种方式下，禁止使用 FLASH 存储器；

注意：在 ROM 器件中没有片内引导 ROM。

1 禁止引导 ROM， 对 F2407A 和 F2406A 片内 FLASH 程序存储器映射地址范围为0000h~7FFFh, 对 F2402A 地址范围为0000h~1FFFh。

位 2 **MP/MC**。微处理器/微控制器选择位。该位反映了器件复位时 MP/MC 引脚上的状态。复位之后，可通过软件来改变该位，以动态映射存储器到片内或片外。

0 器件设置为微控制器方式，程序地址范围0000h~7FFFh 被映射到片内；

1 器件设置为微处理器方式，程序地址范围0000h~7FFFh 被映射到片外(即用户必须自己提供外部存储器器件)。

位1~0 **SARAM 程序/数据空间选择位。**

DON PON SARAM 状态；

0 0 地址空间不被映射，该空间被分配到外部存储器；

0 1 SARAM 被映射到片内程序空间；

1 0 SARAM 被映射到片内数据空间；

1 1 SARAM 即被映射到片内程序空间，又被映射到片内数据空间。

2.3.2 中断优先级和中断向量表

F240x 系列利用CPU 支持的6个可屏蔽中断，采用集中化的中断扩展设计来满足大量的外设中断需求。表2.5为 F240xA 器件的中断源优先级和中断入口地址向量表。

表2.5 F240xA 中断源优先级和中断向量表

中断名称	优先级	CPU中断和 向量地址	在PIRQRx和 PIACKRx中的 数位位置	外围中断向量 (PIV)	能否被 屏蔽	外围中断 源模块	描 述
Reset	1	RSN 0000h		N/A	N	RS引脚 看门狗	来自引脚的复位信号. 看门 狗溢出
保留位	2	— 0026h	三	N/A	N	CPU	用于仿真
NMI	3	NMI 0024h		N/A	N	不可屏 蔽中断	不可屏蔽中断， 只能是软件中断
PDPINTA	4	INT1 0002h	0.0	0020h	Y	EVA	功率驱动保护引脚中断
PDPINTB	5		2.0	0019h	Y	EVB	
ADCINT	6		0.1	0004h	Y	ADC	高优先级模式的ADC中断
XINT1	7		0.2	0001h	Y	外部中 断逻辑	高优先级模式的 外部引脚中断
XINT2	8		0.3	0011h	Y	外部中 断逻辑	
SPIINT	9		0.4	0005h	Y	SPI	高优先级模式的SPI中断
RXINT	10		0.5	0006h	Y	SCI	高优先级模式的 SCI接收中断
TXINT	11		0.6	0007h	Y	SCI	高优先级模式的 SCI发送中断
CANMBINT	12	INT2 0004h	0.7	0040h	Y	CAN	高优先级模式的 CAN邮箱中断
CANERINT	13		0.8	0041h	Y	CAN	高优先级模式的 CAN错误中断
CMPIINT	14		0.9	0021h	Y	EVA	Compare1中断
CMP2INT	15		0.10	0022h	Y	EVA	Compure2中断

续表2. 5

中断名称	优先级	CPU中断和 向量地址	在PIRQRx和 PIACKRx中的 数位位置	外围中断向量 (PIV)	能否被 屏蔽?	外围中断 源模块	描 述
CMP3INT	16	INT2 0004h	0. 11	0023h	Y	EVA	Compare3中断
T1PINT	17		0. 12	0027h	Y	EVA	Timer1周期中断
TICINT	18		0. 13	0028h	Y	EVA	Timer1比较中断
T1UFINT	19		0. 14	0029h	Y	EVA	Timer1下溢中断
T1OFINT	20		0. 15	002Ah	Y	EVA	Timer1上溢中断
CMP4INT	21		2. 1	0024h	Y	EVB	Compare4中断
CMP5INT	22		2. 2	0025h	Y	EVB	Compare5中断
CMP6INT	23		2. 3	0026h	Y	EVB	Compare6中断
T3PINT	24		2. 4	002Fh	Y	EVB	Timer3周期中断
T3CINT	25		2. 5	0030h	Y	EVB	Timer3比较中断
T3UFINT	26		2. 6	0031h	Y	EVB	Timer3下溢中断
T3OFINT	27		2. 7	0032h	Y	EVB	Timer3上溢中断
T2PINT	28	INT3 0006h	1. 0	002Bh	Y	EVA	Timer 2周期中断
T2CINT	29		1. 1	002Ch	Y	EVA	Timer 2比较中断
T2UFINT	30		1. 2	002Dh	Y	EVA	Timer 2下溢中断
T2OFINT	31		1. 3	002Eh	Y	EVA	Timer 2上溢中断
T4PINT	32		2. 8	0039h	Y	EVB	Timer 4周期中断
T4CINT	33		2. 9	003Ah	Y	EVB	Timer 4比较中断
T4UFINT	34	INT4 0008h	2. 10	003Bh	Y	EVB	Timer4下溢中断
T4OFINT	35		2. 11	003Ch	Y	EVB	Timer4上溢中断
CAP1INT	36		1. 4	0033h	Y	EVA	Capture 1中断
CAP2INT	37		1. 5	0034h	Y	EVA	Capture 2中断
CAP3INT	38		1. 6	0035h	Y	EVA	Capture 3中断
CAP4INT	39		2. 12	0036h	Y	EVB	Capture 4中断
CAP5INT	40		2. 13	0037h	Y	EVB	Capture 5中断
CAP6INT	41		2. 14	0038h	Y	EVB	Capture 6中断

续表2. 4

中断名称	优先级	CPU中断和 向量地址	在PIRQRx和 P1ACKRx中的 数位位置	外围中断向量 (PIV)	能否被 屏蔽	外围中断 源模块	描 述
SPIINT	42	INT5 000Ah	1. 7	0005h	Y	SPI	低优先级模式的SPI中断
RXINT	43		1. 8	0006h	Y	SCI	低优先级模式的 SCI接收中断
TXINT	44		1. 9	0007h	Y	SCI	低优先级模式的 SCI发送中断
CANMBINT	45		1. 10	0040h	Y	CAN	低优先级模式的 CAN邮箱中断
CANERINT	46		1. 11	0041h	Y	CAN	低优先级模式的 CAN错误中断
ADCINT	47	INT6 000Ch	1. 12	0004h	Y	ADC	低优先级模式的ADC中断
XINTI	48		1. 13	0001h	Y	外部中 断逻辑	低优先级模式的 外部引脚中断
XINT2	49		1. 14	0011h	Y	外部中 断逻辑	
保留位		000Eh		N/A	Y	CPU	分析中断
TRAP	N/A	0022h	二	N/A	N/A	CPU	TRAP命令
假中断向量	N/A	N/A		0000h	N/A	CPU	假中断向量
INT8~ INT16	N/A	0010h~ 0020h		N/A	N/A	CPL	软件中断向量
INT20~ INT31	N/A	00028h 0003Fh		N/A	N/A	CPU	

注：1. 由于某些特殊的器件缺乏某些外围模块，故这些器件不具有相应的中断。
2. 表中□部分表示F2407/F2406 新的外围中断和向量。

2.3.3 外设中断扩展控制器

F240x 内核提供一个不可屏蔽的中断 NMI 和6个按优先级获得服务的可屏蔽中断 INT1~INT6。而这6个中断级的每一个都可被很多外设中断请求共享，图2.8为外设中断扩展控制器(PIE) 模块图。

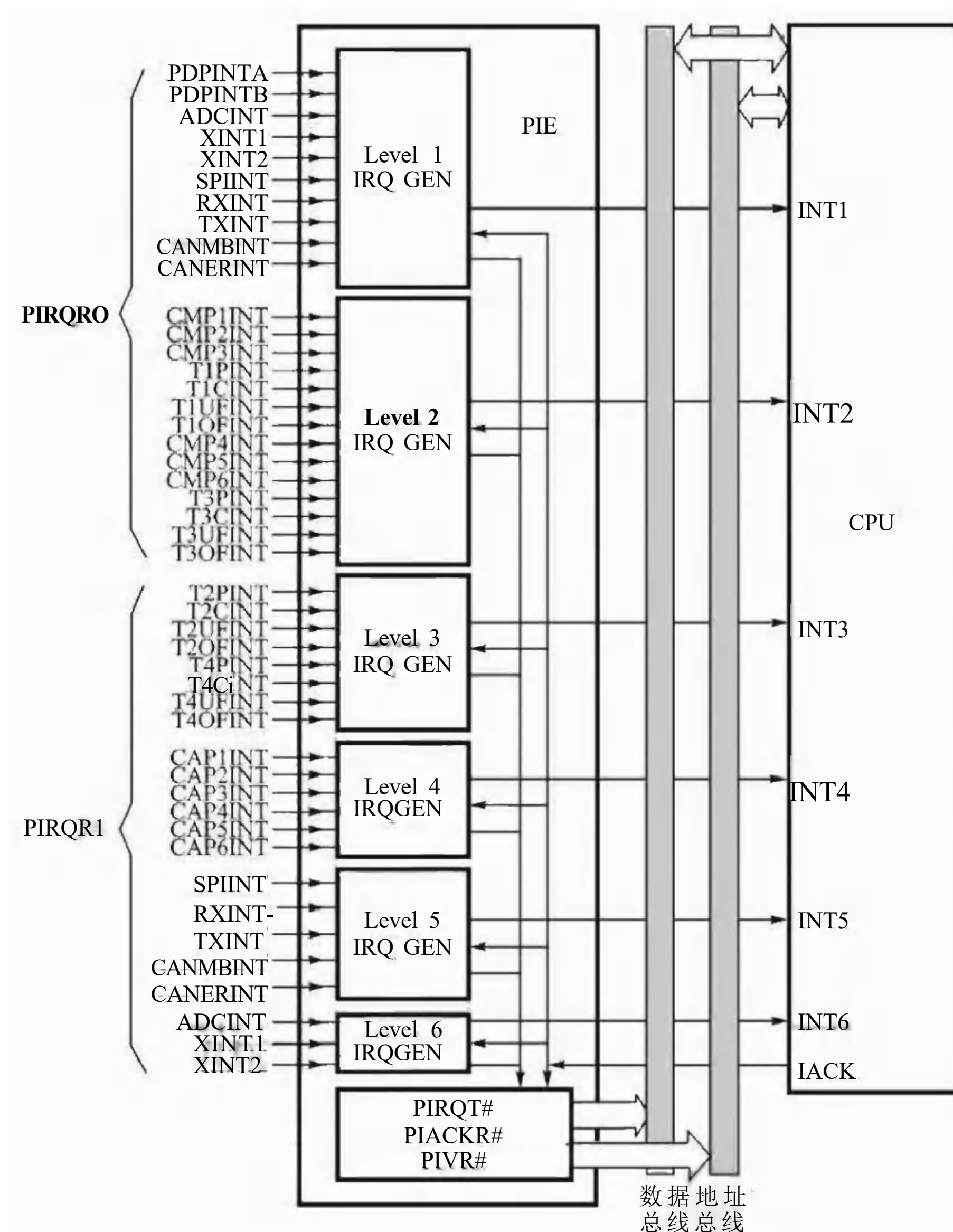


图 2.8 外设中断扩展控制器模块图

1. 中断层次

F240x 系列DSP 通过中断请求系统中的一个两级中断来扩展系统可响应的中断个数。因此，DSP 的中断请求/应答硬件逻辑和中断服务程序软件都是一个两级的层次。

2. 中断请求的结构

在底层中断，从几个外设来的外设中断请求(PIRQ) 在中断控制器处产生一个到CPU 的中断请求(INTn)。这就是内核级的中断请求。在外设配置寄存器中，为每一个产生外设中断请求的事件都有中断使能位和中断标志位。如果一个引起中断的外设事件发生且相应的中断使能位置1，则会产生一个从外设到中断控制器的中断请求。此中断请求反映了外设中断标志位的状态和中断使能位的状态。当中断标志位清0时，中断请求也清0。对某些要设置中断优先级的外设事件来说，当这类事件发生时，其中断优先级的值也被送到中断控制器，而中断请求也是保持到中断应答或者软件将其清0。

在高层中断，被“或”的多个外设中断请求产生一个到CPU 的中断(INTn) 请求，到F240x 的中断请求信号是2个CPU 时钟脉冲宽的低电平脉冲。当任何一个控制INTn 的外设中断请求PIRQ 有效时，都会产生一个到CPU 的中断请求脉冲INTn。如果一个外设中断请求PIRQ 在CPU 对INTn 应答后的一个周期内仍然有效，则另一个中断请求脉冲INTn 也会产生。CPU 总是响应优先级高的外设中断请求。在CPU 内核，这些中断标志在CPU 响应中断时自动清0(注意：不是在外设中断级将中断标志清0)。

2.3.4 中断向量

当CPU 接受中断请求时，它并不知道是哪一个外设事件引起的中断请求。因此，为了让CPU 能够区别这些引起中断的外设事件，在每个外设中断请求有效时都会产生一个唯一的外设中断向量。此外设中断向量被装载外设中断向量寄存器(PIVR) 里面。CPU 应答外设中断时，从PIVR 寄存器中读取相应中断的向量，并产生一个转到该中断服务程序入口的向量。

实际上有两个向量表：CPU 的向量表用来得到响应CPU 中断请求的一级通用中断服务子程序(GISR)；外设向量表用来得到响应某一特定外设事件的特定中断服务子程序(SISR)。GISR 中的程序代码应该读出PIVR 中的值，在保存必要的上下文之后，用PIVR 中的值来产生一个转移到SISR 的向量。

图2.9为XINT1 (在高优先级模式下)产生一个中断的过程。XINT1 在高优先级模式下，值0001h 被装载入PIVR 寄存器。CPU 获得被装载入PIVR 中的值之后，用这个值来判断是哪个外设引起的中断，接着转移到相应的SISR。该转移应当是一个条件转移，即只有在PIVR 寄存器被装载入一个特定的值之后转移才执行。将PIVR 中的值装载入累加器时须先左移，再加上一个固定的偏移量，接着程序转到累加器指定的地址入口。这个地址将指向SISR。

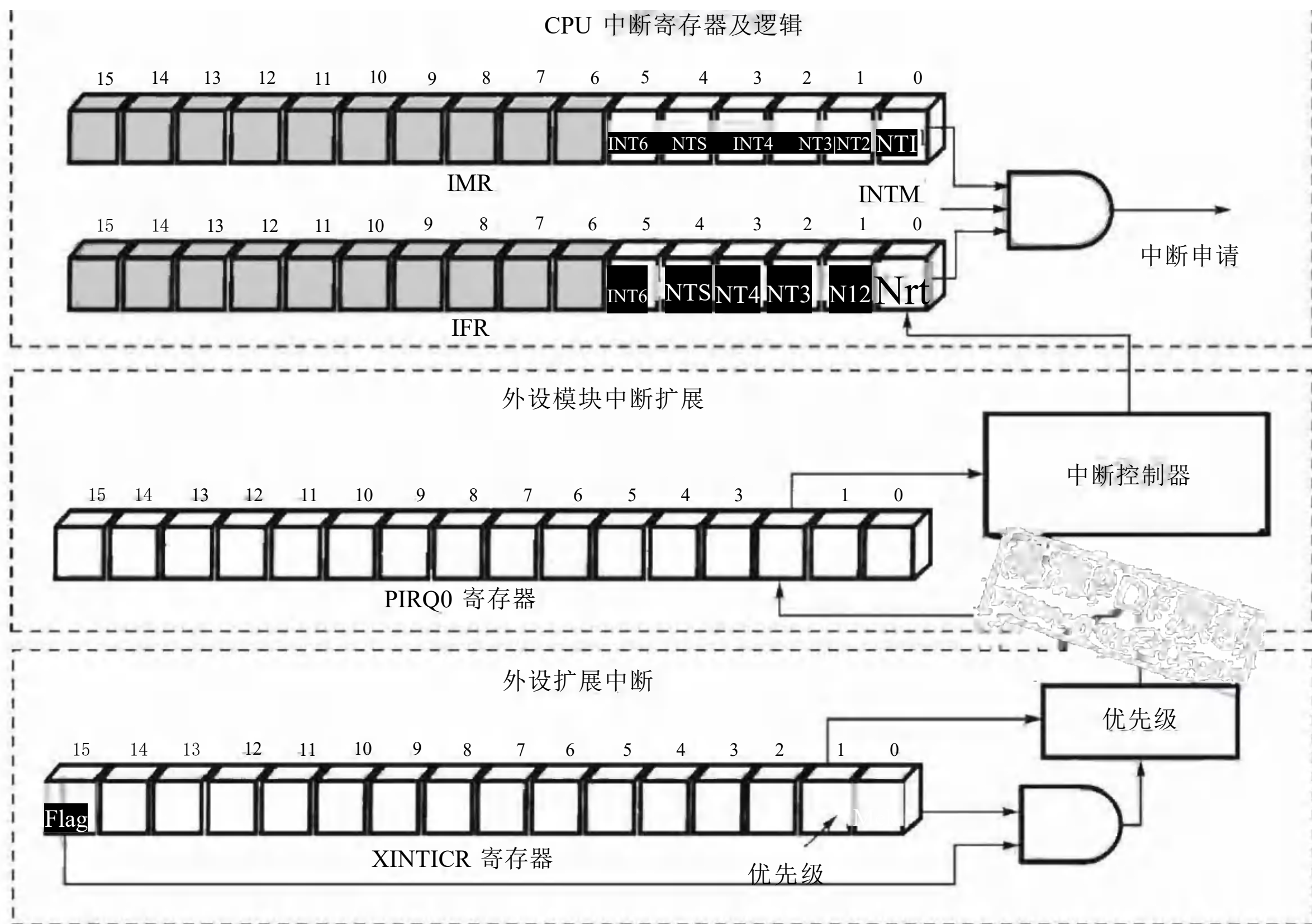


图2.9 可屏蔽的中断结构示例

1. 假中断向量

假(phantom) 中断向量是保持中断系统完整性的一个特性。当一个中断已被响应，但无外设将中断向量地址偏移量装入外设中断向量寄存器 PIVR 中时，假中断向量0000h 被装入 PIVR。这种缺省保证了系统按照可控的方式进行处理。产生假中断的原因有：

- >CPU 执行一个软件中断指令 INTR，使用参数1~6，用于请求服务6个可屏蔽中断级 (INT1~INT6) 之一；
- >中断请求线发生故障，外设发出中断请求，而其 INTn 标志位却在CPU 应答请求之前已经被清0，因此中断响应时，没有外设向 PIVR 装入中断向量地址偏移量，此时向 PIVR 中装入假中断向量。

2. 软件结构

中断服务子程序有两级：通用中断服务子程序GISR 和特定中断服务子程序 SISR。在 GISR 中保存必要的上下文，从外设中断向量寄存器 PIVR 中读取外设中断向量，这个向量用

来产生转移到SISR 的地址入口。对每一个从外设来自中断控制器的中断都有一个特定的SISR, 在 SISR 中执行对该外设事件的响应。

程序一旦进入中断服务程序后, 所有的可屏蔽中断都被屏蔽。GISR 必须在中断被重新使能之前, 读取PIVR 中的值; 否则在另一个中断请求发生之后, PIVR 中将装入另一个中断请求的偏移量, 这将导致原外设中断向量参数的永久丢失。

外设中断扩展PIE 不包括像复位和NMI 这样的不可屏蔽中断。

3. 不可屏蔽中断

F240x 器件没有不可屏蔽中断(NMI) 引脚。在访问无效的地址时, 不可屏蔽中断就会发出请求, 程序转到不可屏蔽中断向量入口地址0024h 处。没有与NMI 相对应的控制寄存器。

2.3.5 中断响应的流程

外设事件要引起CPU 中断, 必须保证: 外设事件的中断使能位被使能, CPU 内核级的6个可屏蔽中断中, 能控制该外设事件高级中断至少有一个被使能。在外设事件发生时, 首先将其在外设中断控制器中的标志位置1, 从而引起CPU 内核的INT1~INT6 中的一个产生中断。图2.10为响应外设中断的流程图。

2.3.6 中断响应的延时

有3种因素导致中断响应的延时: 外设同步接口时间、CPU 响应时间和ISR 转移时间。

>外设同步接口时间是指从外设接口识别出从外设发来的中断请求, 经判优、转换后将请求发送至CPU 的时间。

>CPU 响应时间是指CPU 识别出已经被使能的中断、响应中断、清除流水线, 到从CPU 的中断向量表中获得第一条指令的时间。

>ISR 转移时间是指为了转移ISR 中特定部分而必须执行一些转移所需要的时间。该时间长度根据用户所实现的ISR 的不同而有所变化。

2.3.7 CPU 中断寄存器

CPU 中断寄存器包括: 中断标志寄存器(IFR) 和中断控制寄存器(IMR)。

1.CPU 中断标志寄存器

16位的中断标志寄存器位于数据存储器空间中的0006h 处, 用于识别和清除悬挂的中断。IFR 包含用于所有可屏蔽的中断的标志位。

当一个可屏蔽中断被请求时, 中断控制寄存器的相应标志位被置1。如果同一控制寄存器中的中断使能位也为1, 则该中断请求被送到CPU, 并设置IFR 的相应位。这表示该中断正被悬挂, 等待响应。

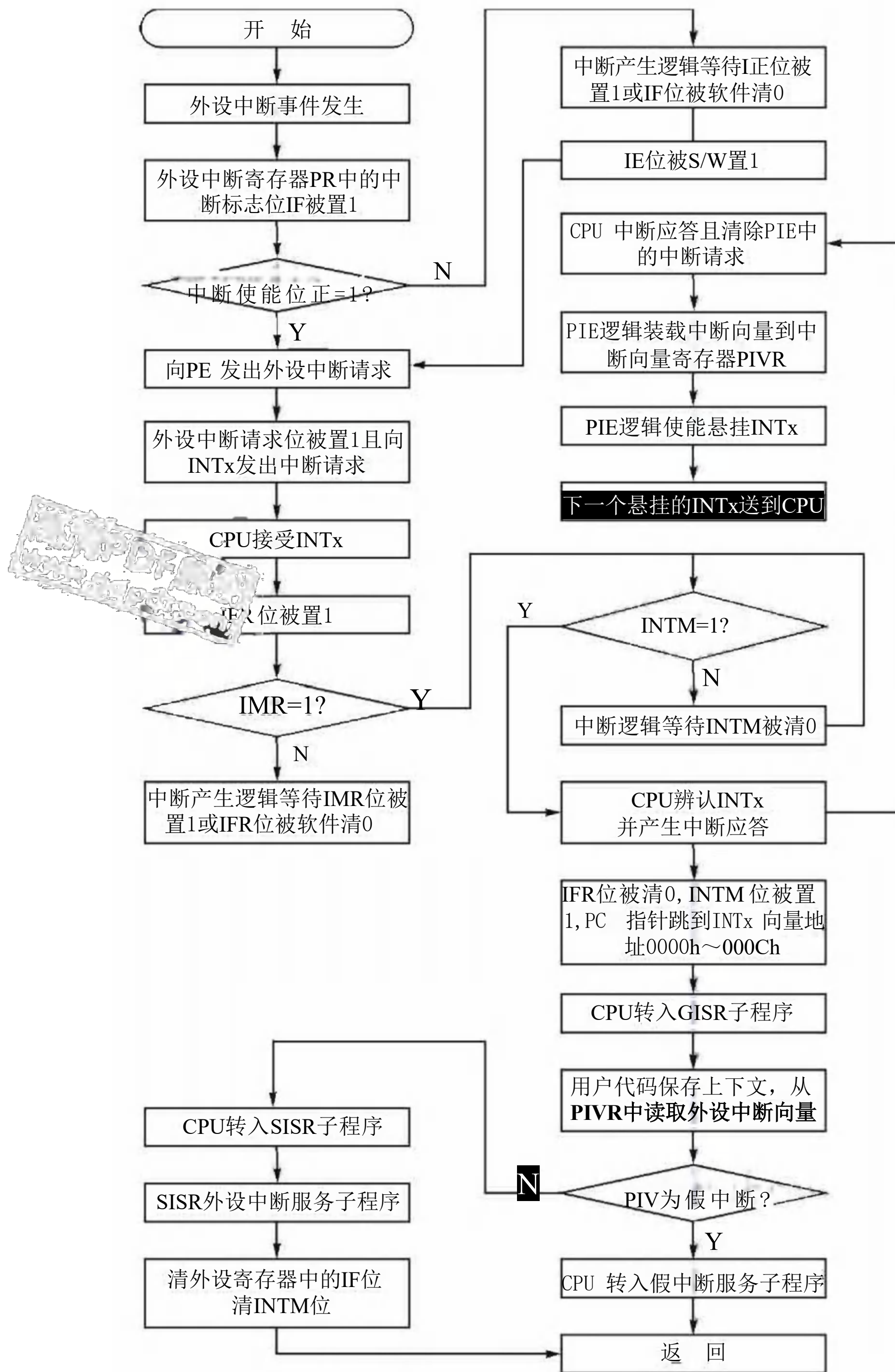


图2.10 CPU 响应外设中断的流程图

读取IFR 可以识别悬挂的中断，而写IFR 则将清除已挂起的中断。为了清除一个中断请求，向相应的 IFR 位写1。把IFR 中当前的内容写回IFR，则可清除所有挂起的中断。器件复位时，将清除所有的IFR 位。

CPU 响应中断或者器件复位都能将 IFR 标志清除。

- 注意：**
- 为清除一个IFR 位，必须向其写1. 而不是0。
 - 当一个可屏蔽中断被响应时，只有IFR 位被清除。相应的外设控制寄存器中的标志位不会被清除。如果应用需要清除这些外设控制寄存器中的标志位，应由用户软件来清除。
 - 当通过INTR 指令来请求中断且相应的IFR 位 被 置 1 时，CPU 不会自动清除该位，该位必须由软件来清除。
- IMR 和 IFR 寄存器控制内核级的中断，所有的外设在它们各自的配置/控制寄存器中都有相应的中断屏蔽和标志位。

中断标志寄存器(IFR)—— 地 址**0006 h**

15~6	5	4	3	2	1	0
保留位	INT6 flag	INT5 flag	INT4 flag	INT3 flag	INT2 flag	INT1 flag
0	RWIC0	RWIC0	RWiC_0	RWIC0	RWIC0	RWIC_0

注：0—读出为0;R= 可读；WIC= 写1清除该位；0=复位值。

- 位15~6 保留位。这些位读出时始终为0。
- 位5 INT6 flag 中断6标志位。该位用作连至第6级中断 INT6 的所有中断标志。
- 0 无 INT6 的中断悬挂；
- 1 至少一个 INT6 的中断悬挂 • 向该位写1可将该位清除为0, 即清除中断请求
- 位 4 INT5 flag中断5标志位。该位用作连至第5级中断 INT5 的所有中断标志。
- 0 无 INT5 的中断悬挂；
- 1 至少一个 INT5 的中断悬挂，向该位写1可将该位清除为0, 即清除中断请求 。
- 位 3 INT4 flag中断4标志位。该位用作连至第4级中断 INT4 的所有中断标志。
- 0 无 INT4 的中断悬挂；
- 1 至少一个 INT4 级的中断悬挂，向该位写1可将该位清除为0. 即清除中断请求 。
- 位 2 INT3 flag中断3标志位。该位用作连至第3级中断 INT3 的所有中断标志。
- 0 无 INT3 的中断悬挂；
- 1 至少一个 INT3 级的中断悬挂，向该位写1可将该位清除为0, 即清除

中断请求。

- 位 1

INT2 flag 中断2标志位。该位用作连至第2级中断 INT2 的所有中断标志。

0 无 INT2 的中断悬挂；

1 至少一个 INT2 级的中断悬挂，向该位写1可将该位清除为0, 即清除中断请求。
- 位 0

INT1 flag中断1标志位。该位用作连至第1级中断 INT1 的所有中断标志。

0 无 INT1 的中断悬挂；

1 至少一个 INT1 级的中断悬挂，向该位写1可将该位清除为0, 即清除中断请求。

2. CPU 中断屏蔽寄存器

中断屏蔽寄存器(IMR) 是一个映射至数据存储器空间0004h 处的16位寄存器。 IMR 中包含所有中断(INT1~INT6) 的屏蔽位。IMR 中不包含NMI 和 RS。因此 IMR 对这两个引脚无影响。

读 IMR 寄存器可以识别出已屏蔽或使能的中断，而向 IMR 中写，则可屏蔽中断级或将使能所有中断。为了使能中断，应设置相应的 IMR 位为1, 而屏蔽中断时，只需将相应的 IMR 位设为0。

中断屏蔽寄存器(IMR)—— 地址 **0004 h**

15~6	5	4	3			0
保留位	INT6 mask	INT5 mask	INT4 mask	INT3 mask	INT2 mask	INT1 mask
0	RW	RW	RW	RW	RW	RW

注：0=读出为0;R= 可读；W= 可写；该位的值不受器件复位的影响。

- 位15~6

保留位。
- 位 5

INT6 mask 中断6的屏蔽位。

0 屏蔽中断INT6；

1 使能中断INT6。
- 位 4

INT5 mask 中断5的屏蔽位。

0 屏蔽中断 INT5；

1 使能中断INT5。
- 位 3

INT4 mask 中断4的屏蔽位。

0 屏蔽中断INT4；

1 使能中断INT4。

- 位 2

INT3 mask 中断3的屏蔽位。

0 屏蔽中断INT3;

1 使能中断INT3。
- 位 1

INT2 mask 中断2的屏蔽位。

0 屏蔽中断INT2;

1 使能中断INT2。
- 位 0

INT1 mask 中断1的屏蔽位。

0 屏蔽中断INT1;

1 使能中断 INT1。

注意：器件复位时. IMR 位不受影响

2.3.8 外设中断寄存器

外设中断寄存器包括：

►外设中断向量寄存器(PIVR);

- 外设中断请求寄存器0/1/2(PIRQR0/1/2), 其各位定义见表2.6~2.8;
- >外设中断应答寄存器0/1/2(PIACKRo/1/2), 其各位定义见表2.9~2.11。

外设中断请求寄存器0/1/2和外设中断应答寄存器0/1/2都属于外设中断扩展模块。用来向CPU 产生INTI~INT6 中断请求的内部寄存器。这些寄存器用于测试目的. 而非用户应用目的, 因此在编程时可忽略。

1. 外设中断向量寄存器(PIVR)—— 地址701Eh

15	14	13	12	11	10	9	8
V15	V14	V13	V12	V11	V10	V9	V8
R_0	R_0	R_0	R_0	R_0	R_0	R_0	R_0
7	6	5	4	3	2	1	0
V7	V6	V5	V4	V3	V2	V1	Vo
R_0	R_0	R_0	R_0	R_0	R_0	R_0	R_0

注：R= 可读；L0= 复位值。

位15~0 V15~V0。 中断向量V15~VO 位。该寄存器存有最近一次应答的外设中断的地址向量。

2. 外设中断请求寄存器**0 (PIRQRO)**—— 地址**7010 h**

15	14	13	12	11	10	9	8
IRQ0. 15	IRQ0. 14	IRQ0. 13	IRQ0. 12	IRQ0. 11	IRQ0. 10	IRQ0. 9	IRQ0. 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
IRQ0. 7	IRQ0. 6	IRQ0. 5	IRQ0. 4	IRQ0. 3	IRQ0. 2	IRQ0. 1	IRQ0. 0
RW_0	RW_o	RW_0	RW_0	RW_o	RW_o	RW_o	RW_0

注： • 写入1会发出一个中断请求到DSP内核，写入0没有影响。
• 有中断请求而未被响应，称为悬挂。

位15～0 **IRQ0.15～IRQ0.0** 位。

- 0 相应的中断请求未被悬挂；
- 1 悬挂中断请求。

表2.6 外设中断请求寄存器 PIRQRO 各位的定义

位的位置	中 断	中断描述	中断优先级
IRQ0. 0	PDPINTA	功率驱动保护引脚中断	INT1
IRQ0. 1	ADCINT	高优先级模式的ADC中断	INT1
IRQ0. 2	XINT1	高优先级模式的外部引脚1中断	INT1
IRQ0. 3	XINT2	高优先级模式的外部引脚2中断	INT1
IRQ0. 4	SPIINT	高优先级模式的SPI中断	INT1
IRQ0. 5	RXINT	高优先级模式的SCI接收中断	INT1
IRQ0. 6	TXINT	高优先级模式的SCI发送中断	INT1
IRQ0. 7	CANMBINT	高优先级模式的CAN邮箱中断	INT1
IRQ0. 8	CANERINT	高优先级模式的CAN错误中断	INT1
IRQ0. 9	CMP1INT	Compare1中断	INT2
IRQ0. 10	CMP2INT	Compare2中断	INT2
IRQ0. 11	CMP3INT	Compare3中断	INT2
IRQ0. 12	TIPINT	Timer1周期中断	INT2
IRQ0. 13	T1CINTT	Timer1比较中断	INT2
IRQ0. 14	T1UFINT	Timer1下溢中断	INT2
IRQ0. 15	T1OFINT	Timer1上溢中断	INT2

3. 外设中断请求寄存器1(PIRQR1)——地 址7011h

15	14	13	12	11	10	9	8
保留位	IRQ1. 14	IRQ1. 14	IRQ1. 14	IRQ1. 14	IRQ1. 14	IRQ1. 14	IRQ1. 14
R_o	RW_0	RW_0	RW0	RW_0	RW_0	RW_0	RW0
7	6	5	4	3	2	1	0
IRQ1. 14	IRQ1. 14	IRQ1. 14	IRQ1. 14	IRQ1. 14	IRQ1. 14	IRQ1. 14	IRQ1. 14
RW 0	RW_0	RW0	RW_0	RW0	RW 0	RW 0	RW0

注：写入1会发出一个中断请求到DSP内核. 写入0没有影响。

位15 保留位。读出为0, 写入没影响。

位14~0 IRQ1.14~IRQ1.0 位。

0 相应的中断请求未悬挂；

1 悬挂中断请求。

表2.7 外设中断请求寄存器 PIRQRI 各位的定义

位的位置	中 断	中断描述	中断优先级
IRQ1. 0	T2PINT	Timer2周期中断	INT3
IRQ1. 1	T2CINT	Timer2比较中断	INT3
IRQ1. 2	T2UFINT	Timer2下溢中断	INT3
IRQ1. 3	T2OFINT	Timer2上溢中断	INT3
IRQL. 4	CAP1INT	Capture1中断	INT4
IRQ1. 5	CAP2INT	Capture2中断	INT4
IRQL. 6	CAP3INT	Capture3中断	INT4
IRQ1. 7	SPIINT	低优先级模式的SPI中断	INT5
IRQ1. 8	RXINT	低优先级模式的SCI接收中断	INT5
IRQ1. 9	TXINT	低优先级模式的SCI发送中断	INT5
IRQ1. 10	CANMBINT	低优先级模式的CAN邮箱中断	INT5
IRQ1. 11	CANERINT	低优先级模式的CAN错误中断	INT5
IRQ1. 12	ADCINT	低优先级模式的ADC中断	INT6
IRQ1. 13	XINT1	低优先级模式的外部引脚1中断	INT6
IRQ1. 14	XINT2	低优先级模式的外部引脚2中断	INT6

4. 外设中断请求寄存器2(PIRQR2)—— 地址 7012 h

15	14	13	12	11	10	9	8
IRQ2. 15	IRQ2. 14	IRQ2. 13	IRQ2. 12	IRQ2. 11	IRQ2. 10	IRQ2. 9	IRQ2. 8
R_0	RW0	RW_0	RW_0	RW_0	RW Q	RW_0	RWQ
7	6	5	4	3	2	1	0
IRQ2. 7	IRQ2. 6	IRQ2. 5	IRQ2. 4	IRQ2. 3	IRQ2. 2	IRQ2. 1	IRQ2. 0
RW_o	RW_0	RW_0	RW_0	RW_0	RW_0	RW_0	RW_o

注：写入1会发出一个中断请求到DSP 内核，写入0没有影响。

位15~0 IRQ2.15~IRQ2.0 位。

- 0 相应的中断请求未悬挂；
- 1 悬挂中断请求。

表2 . 8 外设中断请求寄存器 PIRQR2 各位的定义

位的位置	中 断	中断描述	中断优先级
IRQ2. 0	PDPINTB	功率驱动保护中断引脚	INT1
IRQ2. 1	CMP4INT	Compare4中断	INT2
IRQ2. 2	CMP5INT	Compare5中断	INT2
IRQ2. 3	CMP6INT	Compare6中断	INT2
IRQ2. 4	T3PINT	Timer3周期中断	INT2
IRQ2. 5	T3CINT	Timer3比较中断	INT2
IRQ2. 6	T3UFINT	Timer3下溢中断	INT2
IRQ2. 7	T3OFINT	Timer3上溢中断	INT2
IRQ2. 8	T4PINT	Timer4周期中断	INT3
IRQ2. 9	T4CINT	Timer4比较中断	INT3
IRQ2. 10	T4UFINT	Timer4下溢中断	INT3
IRQ2. 11	T4OFINT	Timer4上溢中断	INT3
IRQ2. 12	CAP4INT	Capture4中断	INT4
IRQ2. 13	CAP5INT	Capture5中断	INT4
IRQ2. 14	CAP6INT	Capture6中断	INT4

5. 外设中断应答寄存器0 (PIACKR O)—— 地 址7014h

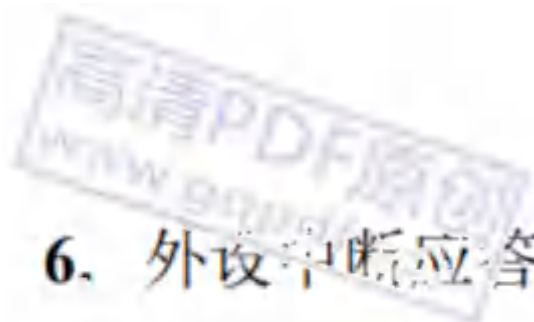
15	14	13	12	11	10	9	8
IAK0. 15	IAK0. 14	LAK0. 13	IAK0. 12	IAK0. 11	IAK0. 10	IAK0. 9	IAK0. 8
RW_0	RW_0	RW_0	RW_0	RW_0	RW_0	RW_0	RW_0
7	6	5	4	3	2	1	0
IAK0. 7	IAK0. 6	IAK0. 5	IAK0. 4	IAK0. 3	IAK0. 2	IAK0. 1	IAK0. 0
RW_0	RW_0	RW_0	RW_0	RW_0	RW_0	RW_0	RW_0

位15~0 ICK0.15~ICK0.0 位。外设中断应答位。写入1引起相应的外设中断应答被插入，从而将相应的外设中断请求位清0。

注意：通过向该寄存器写入1. 来插入的中断应答并不更新PIVR 寄存器的内容. 读这个寄存器得到的结果通常是0。

表2.9 外设中断应答寄存器PIACKR0 各位的定义

位的位置	中 断	中断描述	中断优先级
IAK 0. 0	PDPINT	功率驱动保护中断	INT1
IAK 0. 1	ADCINT	高优先级模式的ADC中断	INT1
IAK 0. 2	XINT1	高优先级模式的外部引脚1中断	INT1
IAK 0. 3	XINT2	高优先级模式的外部引脚2中断	INT1
IAK0. 4	SPIINT	高优先级模式的SPI中断	INT1
IAK0. 5	RXINT	高优先级模式的SCI接收中断	INT1
IAK0. 6	TXINT	高优先级模式的SCI发送中断	INT1
IAK0. 7	CANMBINT	高优先级模式的CAN邮箱中断	INT1
IAK0. 8	CANERINT	高优先级模式的CAN错误中断	INT1
IAK 0. 9	CMPIINT	Compare1中断	INT2
IAK 0. 10	CMP2INT	Compare2中断	INT2
IAK0. 11	CMP3INT	Compare3中断	INT2
IAK 0. 12	TIPINT	Timer1周期中断	INT2
IAK 0. 13	TICINT	Timer1比较中断	INT2
IAK0. 14	TIUFINT	Timer1下溢中断	
IAK 0. 15	T1OFINT	Timer1上溢中断	



6. 外设中断应答寄存器1 (PIACKR1)—— 地 址**7015h**

15	14	13	12	11	10	9	8
保留位	IAK1. 14	IAK1. 13	IAK1. 12	IAK1. 11	IAK1. 10	IAK1. 9	IAK1. 8
R_0	RW_0	RW_0	RW_0	RW_0	RW_0	RW_0	RW_0
7	6	5	4	3	2	1	0
IAK1. 7	IAK1. 6	IAK1. 5	IAK1. 4	IAK1. 3	IAK1. 2	IAK1. 1	IAK1. 0
RW_0	RW_0	RW_0	RW_0	RW_0	RW_0	RW_0	RW_0

位15 保留位。读出为0, 写入没影响。

位14~0 IAK1.14~IAK1.0 位。作用与 PIACKRO 寄存器一样。

表2. 10 外设中断应答寄存器 PIACKR1 各位的定义

位的位置	中 断	中断描述	中断优先级
IAK1. 0	T2PINT	Timer2周期中断	INT3
IAK1. 1	T2CINT	Timer2比较中断	INT3
IAK1. 2	T2UFINT	Timer2下溢中断	INT3
IAK1. 3	T2OFINT	Timer2上溢中断	INT3
IAK1. 4	CAPINT1	Capture1中断	INT4
IAK1. 5	CAPINT2	Capture2中断	INT4
IAK1. 6	CAPINT3	Capture3中断	INT4
IAK1. 7	SPIINT	低优先级模式的SPI中断	INT5
IAK1. 8	RXINT	低优先级模式的SCI接收中断	INT5
IAK1. 9	TXINT	低优先级模式的SCI发送中断	INT5
IAK1. 10	CANMBINT	低优先级模式的CAN邮箱中断	INT5
IAK1. 11	CANERINT	低优先级模式的CAN错误中断	INT5
IAK1. 12	ADCINT	低优先级模式的ADC中断	INT6
IAK1. 13	XINT1	低优先级模式的外部引脚1中断	INT6
IAK1. 14	XINT2	低优先级模式的外部引脚2中断	INT6

7. 外设中断应答寄存器2 (PIACKR2)——地 址7016h

15	14	13	12	11	10	9	8
IAK2. 15	IAK2. 14	IAK2. 13	IAK2. 12	IAK2. 11	IAK2. 10	IAK2. 9	IAK2. 8
R_0	R/W_0	R/W_0	R/W_0	R/W0	R/W_0	R/W_D	R/W_0
7	6	5	4	3	2	1	0
IAK2. 7	IAK2. 6	IAK2. 5	IAK2. 4	IAK2. 3	IAK2. 2	IAK2. 1	IAK2. 0
R/W_0	R/W_o	R/W_6	R/W_0	R/W_0	R/W_0	R/W_D	R/W_0

位15～0 IAK2.15~IAK2.0 位。作用与 PIACKRO 寄存器一样。

表2. 11 外设中断应答寄存器 PIACKR2 各位的定义

位的位置	中 断	中断描述	中断优先级
IAK2. 0	PDPINTB	功率驱动保护中断	INT1
IAK2. 1	CMP4INT	Compare4 中断	INT2
IAK2. 2	CMP5INT	Compare5中断	INT2
IAK2. 3	CMP6INT	Compare6中断	INT2
IAK2. 4	T3PINT	Timer3周期中断	INT2
IAK2. 5	T3CINT	Timer3比较中断	INT2
IAK2. 6	T3UFINT	Timer3下溢中断	INT2
IAK2. 7	T3OFINT	Timer3上溢中断	INT2
IAK2. 8	T4PINT	Timer4周期中断	INT3
IAK2. 9	T4CINT	Timer4比较中断	INT3
IAK2. 10	T4UFINT	Timer4下溢中断	INT3
IAK2. 11	T4OFINT	Timer4上溢中断	INT3
IAK2. 12	CAP41NT	Capture4中断	INT4
IAK2. 13	CAP51NT	Capture5中断	INT4
IAK2. 14	CAP6INT	Capture6中断	INT4

2. 3. 9 复 位

F240x 器件有两个复位源： 一个外部复位引脚复位和一个程序监视定时器复位。

复位引脚为一个I/O 脚，如果有内部复位事件(程序监视定时器复位)发生，则该引脚被设置为输出方式，并且被驱动为低，向外部电路表明F240x 器件正在自己复位。

外部复位引脚和程序监视定时器复位相“或”后一起驱动 CPU 的复位信号。

2.3.10 无效地址检测

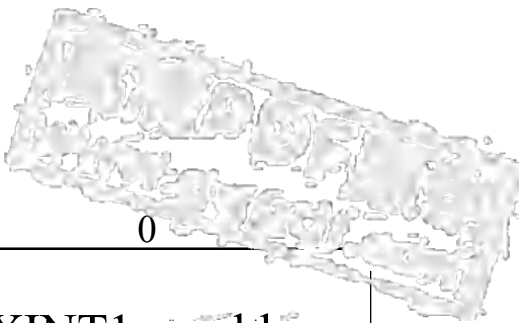
系统和外设模块控制寄存器地址映射中包含不可实现单元，译码逻辑能够检测任何对这些无效地址的访问。一旦检测到对无效地址的访问，就将系统控制和状态寄存器1(SCSR1)中的无效地址标志位(ILLADR) 置1, 从而产生一个不可屏蔽中断(NMI)。 无论何时检测到访问无效地址，都会产生插入一个无效地址条件。无效地址标志位在无效地址条件发生之后置1, 并保持直到软件将其清除。

2.3.11 外部中断控制寄存器

XINT1CR 和 XINT2CR 为两个控制和监视 XINT1 和 XINT2 引脚状态的外部中断控制寄存器。在F240x 器件中，XINT1 和 XINT2 引脚必须至少被拉低6个(或12个) CLKOUT 周期，才能被 CPU 内核认可。

1. 外部中断1控制寄存器(XINT1CR) ——地址7070 h

15	14~3	2	1	0
XINT1 flag	保留位	XINT1 polarity	XINT1 priority	XINT1 enable
RC0	R_0	RW_0	RW_0	RW_0



- 位15

XINT1 标志位。该位指示在该 XINT1 引脚上是否检测到一个所选的跳变。无论外部中断1是否被使能，该位都可置位。当相应的中断被应答时，该位被自动清0。通过软件向该位写1(写入0无效)或者器件复位时，该位也被清0。
0 未检测到跳变；
1 检测到跳变。
- 位14~3

保留位。读出数据为0, 写入未影响。
- 位 2

XINT1 极性。该读/写位决定中断是在 XINT1 引脚信号的上升沿还是下降沿产生。
0 在下降沿(由高到低跳变)产生中断；
1 在上升沿(由低到高跳变)产生中断。
- 位 1

XINT1 优先级。该读/写位决定哪一个中断优先级被请求。 CPU 的优先级层次和相应的高低优先级已经被编码到外设中断扩展寄存器中，可参见表2. 5中断源优先级和向量表。
0 高优先级；
1 低优先级。

- 位 0 XINT1 使能位。该读/写位使能或屏蔽外部中断 XINT1。
- 0 屏蔽中断；
- 1 使能中断。

2. 外部中断2控制寄存器 (XINT2CR)——地 址7071h

15	14~3	2	1	0
XINT2 flag	保留位	XINT2 polarity	XINT2 priority	XINT2 enable
RC 0	R o	RW 0	RW 0	RW 0

- 位15 XINT2 标志位。该位指示在该XINT2 引脚上是否检测到一个所选的跳变。无论外部中断1是否被使能，该位都可置位。当相应的中断被应答时，该位被自动清0。通过软件向该位写1（写0无效）或者器件复位时，该位也被清0。
- 0 未检测到跳变；
- 1 检测到跳变。
- 位 I4~3 保留位。读出数据为0, 写入未影响。
- 位 2 XINT2 极性。该读/写位决定了中断是在 XINT2 引脚信号的上升沿还是下降沿产生。
- 0 在下降沿（由高到低跳变）产生中断；
- 1 在上升沿（由低到高跳变）产生中断。
- 位 1 XINT2 优先级。该读/写位决定哪一个中断优先级被请求。CPU 的优先级层次和相应的高低优先级已经被编码到外设中断扩展寄存器中，可参见表2.5中断源优先级和向量表。
- 0 高优先级；
- 1 低优先级。
- 位 0 XINT2 使能位。该读/写位使能或禁止外部中断 XINT2。
- 0 使能中断；
- 1 接着中断。

2.4 程序控制

程序控制设计一个或多个指令块的次序调度。通常程序是顺序执行的，器件执行这些连续程序存储器地址处的指令。但有时候，程序必须转移到其他地址，并在新地址处开始顺序执行指令，因此F240x 器件支持调用、返回和中断。

2.4.1 程序地址的产生

程序流要求处理器在执行当前指令的同时产生下一个程序地址(顺序或非顺序)。程序地址产生方式如图2.11所示,程序地址产生小结见表2.12。

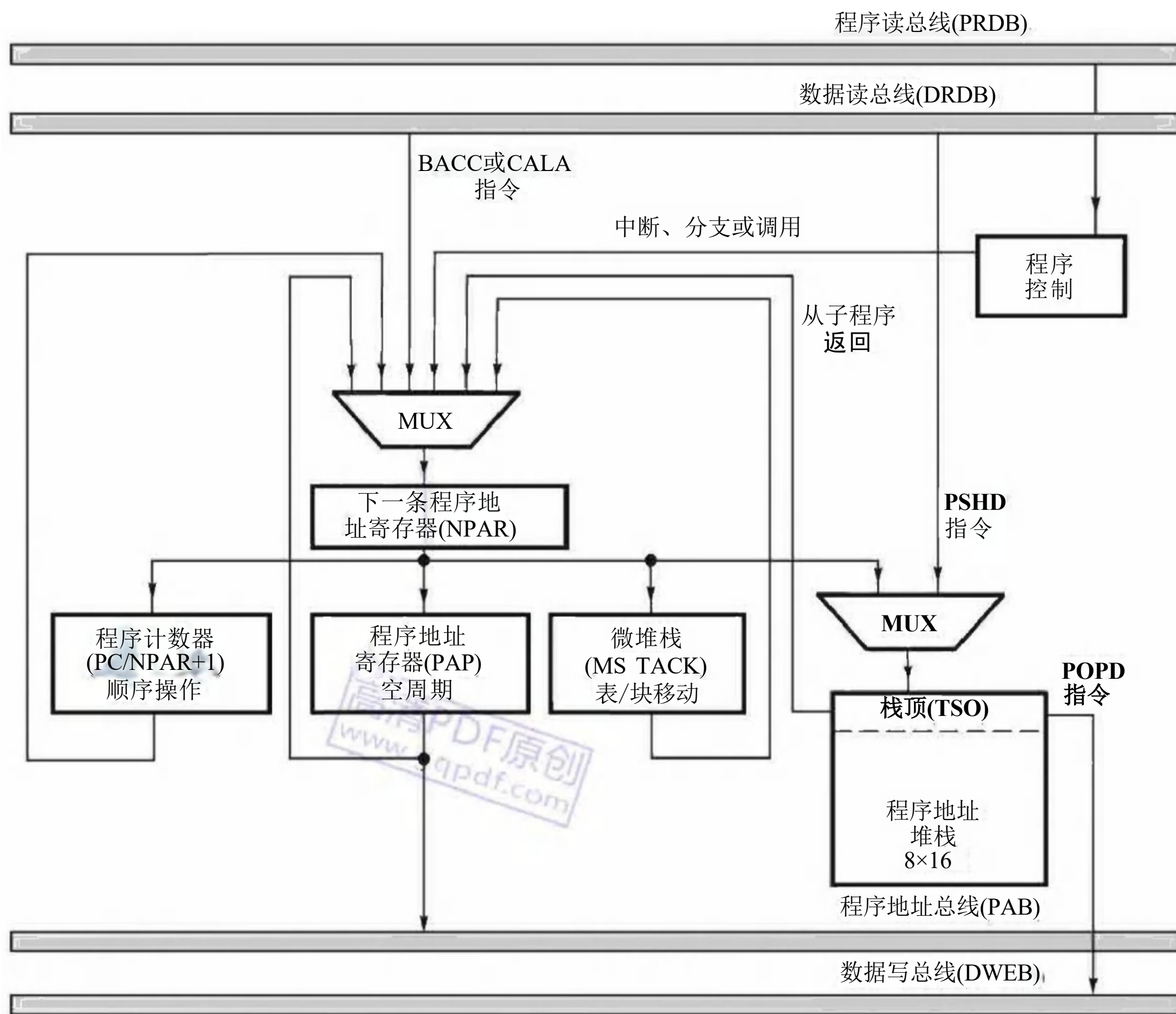


图 2.11 程序地址产生方框图

F240x 器件程序地址产生逻辑使用下列硬件。

➤程序计数器(PC) F240x 具有16位程序计数器(PC)，在取址时，它对内部或外部程序存储器进行寻址。

程序地址寄存器(PAR) PAR 驱动程序地址总线(PAB)。PAR 是16位总线，它同时为读/写程序提供地址。

程序堆栈产生逻辑包含用于存储最多8个返回堆栈的16位宽度、8级硬件逻辑

堆栈。此外，用户还可将堆栈用于临时存储器。

- 微堆栈(MSTACK) 有时，程序地址产生逻辑使用16位宽度、1级微堆栈MSTACK来存储一个返回地址。
- 重复计数器(RPTC) 16位RPTC与重复(RPT)指令一起使用，以决定RPT之后的指令重复多少次。

表2.12 程序地址产生小结

操 作	程序源
顺序操作	PC(包含程序地址+I)
空周期	PAR(包含程序地址)
从子程序返回	栈顶(TOS)
从表移动或块移动返回	栈底(MSTACK)
转移或调用至指令规定的地址	使用程序读总线(PROB)的转移或调用指令
转移或调用至累加器低位字规定的地址	使用数据读总线(DRDB)的累加器低位字
转移至中断服务子程序	使用程序读总线(PRDB)的中断向量存储单元

1. 程序计数器

程序地址产生逻辑使用16位程序计数器(PC)对内部或外部程序存储器寻址。PC保存将被执行的下一条指令的地址。通过程序地址总线(PAB)，从程序存储器中该地址处取回指令并装入指令寄存器。当装载入指令寄存器时，PC保存下一地址。

F240x可以采用多种方法装载PC，从而适应顺序和非顺序的程序流。表2.13列出的是依据所完成的代码操作，装入PC的地址。

表2.13 装入PC的地址

代码操作	装入PC的地址
顺序执行	如果当前指令只有一个字，那么PC装入PC+1;如果当前指令具有两个字.那么PC将装入PC+2
转移	PC装入直接跟随在转移指令之后的长立即数的值
子程序调用和返回	对于调用，下一指令的地址从PC中压入堆栈，然后直接跟随在调用指令之后的长立即数被装入PC、返回指令把返回地址弹回到PC内，从而返回到调用处的代码
软件和硬件中断	PC装入适当的中断向量单元地址。在此单元中存放一条转移指令，该指令将把相应的中断服务子程序的地址装入PC
计算转移	累加器低16位内容装入PC。利用BACC(转移到累加器中的地址)或CALAC调用累加器所规定的地址单元中的子程序)指令可实现计算转移操作

2. 堆 栈

F240x 具有16位宽度、8级深度的硬件堆栈。当子程序调用或中断发生时，程序地址产生逻辑把堆栈用于存储返回地址。当指令强迫 CPU 进入子程序或中断强迫 CPU 进入中断服务程序时，返回地址被自动压入堆栈顶部，该事件不需要附加的周期。当子程序或中断服务程序完成时，返回指令把返回地址从堆栈顶部弹出到程序计数器。

当8级堆栈不用于保存地址时，在子程序或中断服务子程序内，堆栈可用于保存上下文数据或其他存储。

用户可以使用两组指令访问堆栈。

>PUSH (压入)和 POP (弹出)。PUSH 指令把累加器的低位字复制到堆栈顶部；POP 指令把堆栈顶部的数值复制到累加器的低位字。

>PSHD 和 POPD。这两条指令允许用户在数据存储器中为超过8级的子程序或中断嵌套建立软件堆栈。PSHD 指令把数据存储器的值压入软件堆栈的顶部；POPD 指令把数值从软件堆栈顶部弹出至数据存储器。

无论何时，利用指令或利用地址产生逻辑。把数值压入堆栈顶部时，每一级的内容将向下推一级，而堆栈底部(第8个)单元的内容将丢失。因此如果在弹出堆栈之前发生了多于8次的连续压栈，那么数据将会丢失(堆栈发生溢出)。图2.12为压栈操作的示意图。

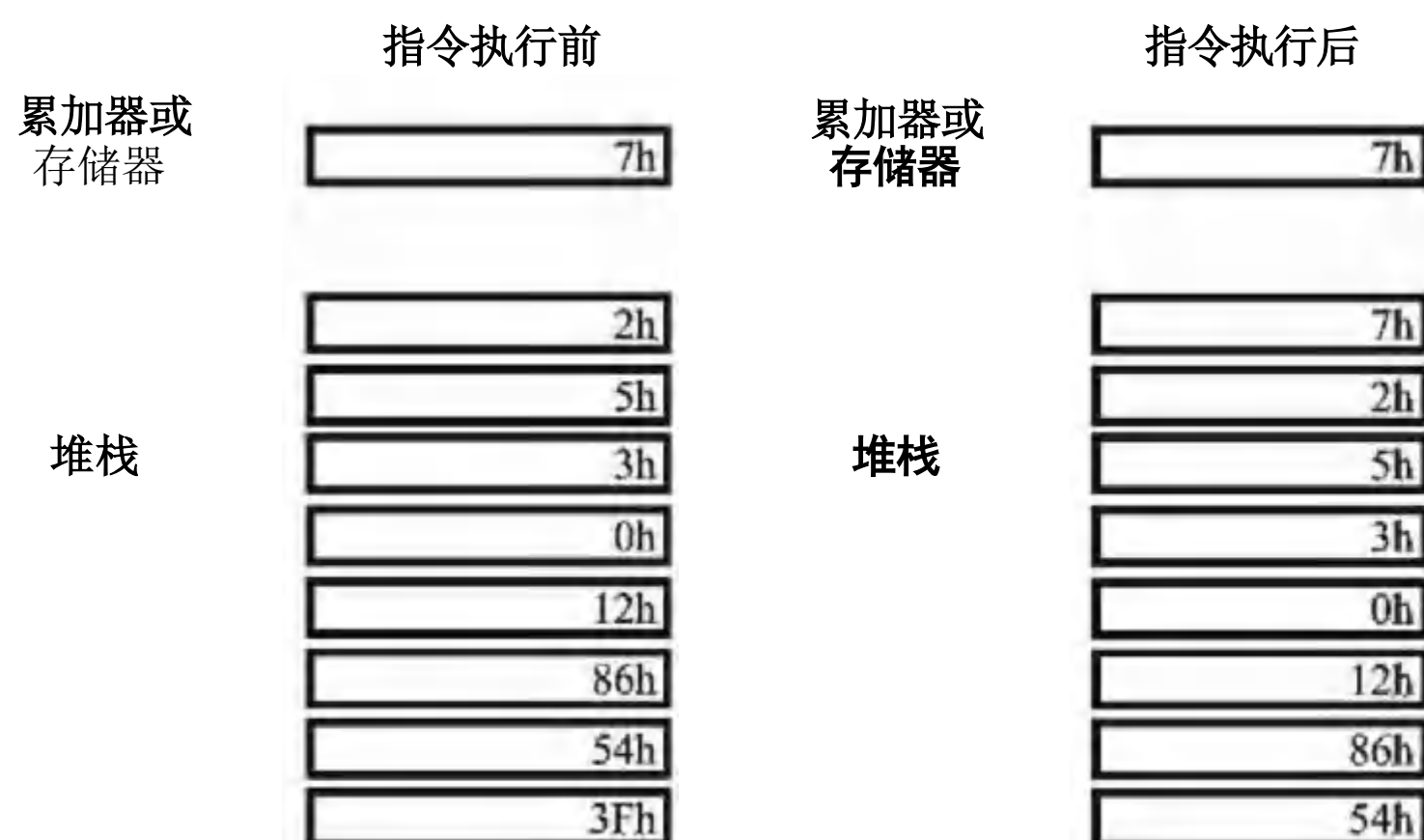


图2.12 压栈操作

弹栈操作与压栈操作相反。弹栈操作把每一级的数值复制到相邻的较高级。在连续7次的弹栈操作之后，因为此时底层的数值已被复制到堆栈的所有层，所以此后的任何弹栈操作都将与堆栈底部的值相同。图2.13为弹栈操作的示意图。

3. 微堆栈

在执行某些指令之前，程序地址产生逻辑使用16位宽度、1级深度的微堆栈(MSTACK)

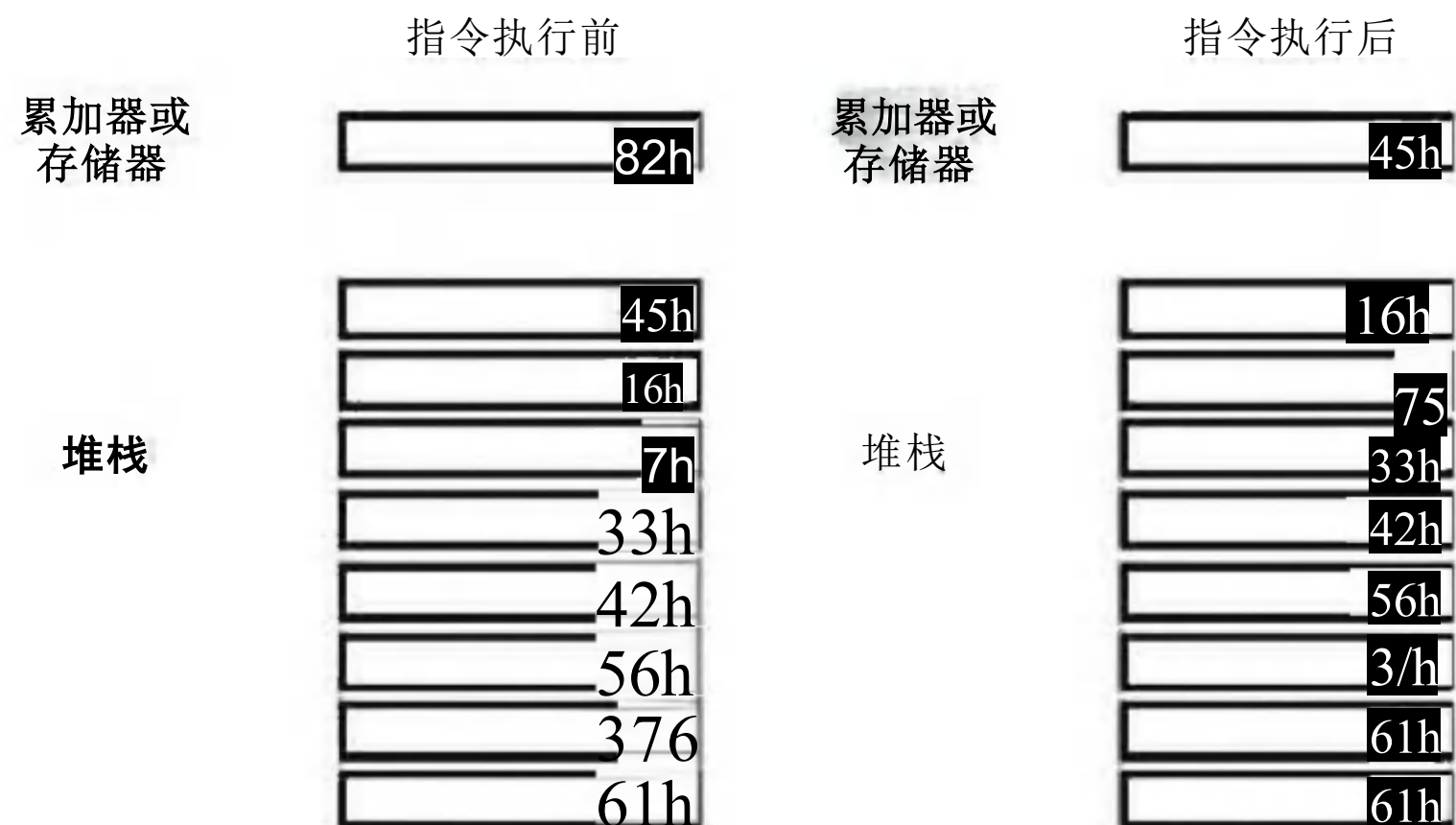


图2.13 弹栈操作

存储返回地址。这些指令使用程序地址产生逻辑提供双操作数指令的第二地址。这些指令是：BLDD、BLPD、MAC、MACD、TBLR及TBLW。在重复执行时，这些指令利用PC 使第一个操作数地址增1, 并使用辅助寄存器算术单元 (ARAU) 产生第二个操作数地址。在使用这些指令时，返回地址(将被取出的下一指令的地址)被压入 MSTACK。重复指令执行完后，MSTACK 的值被弹出并送至程序地址产生逻辑。MSTACK 操作对用户是不可见的。与堆栈不同，MSATCK 只能被用于程序地址产生逻辑，不允许用户把MSATCK 用于存储指令。

2.4.2 流水线操作

指令流水线由发生于指令执行期间内的总操作序列组成。F240x 流水线具有4个独立的阶段：取指令、指令译码、取操作数以及指令执行。由于4个阶段是独立的，所以这些操作可以重叠。在任何给定的操作之内，1~4条指令可以被激活，每条指令处于不同的阶段。图2.14 表示适用于单字、单周期指令且无等待状态执行的4级流水线操作。

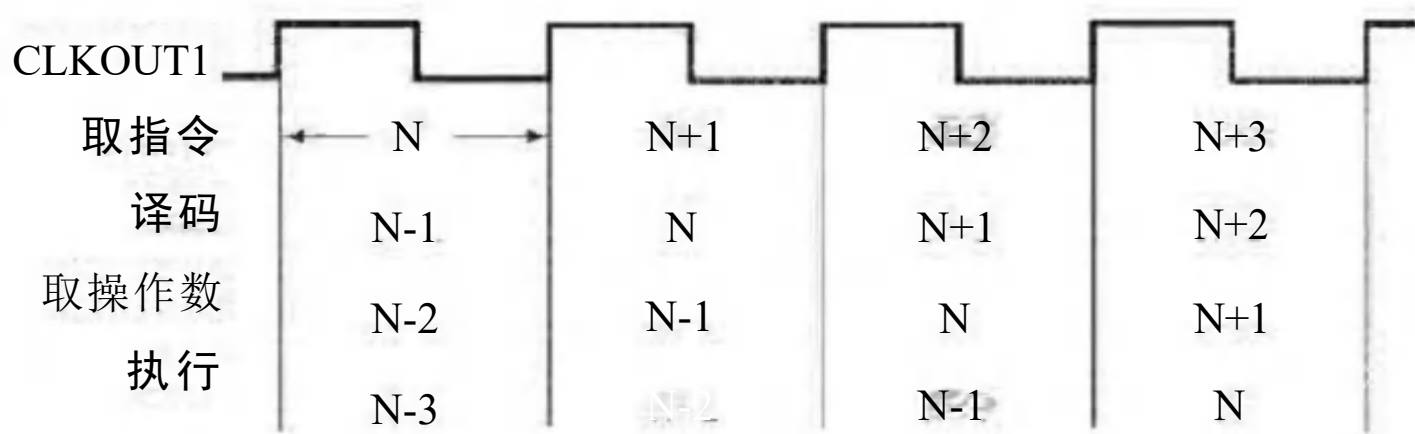


图2.14 4级流水线操作

2.4.3 转移、调用和返回

转移、调用和返回通过把控制传送到程序存储器的另一单元，而中断顺序执行程序流。转

移仅把控制传送到新的地址。调用还把返回地址(跟随在调用之后的指令地址)保存到硬件堆栈的顶部。每一被调用的子程序或中断服务程序以返回指令结束,该指令把返回地址弹出堆栈并送入PC。

F240x 器件有两种类型的转移、调用和返回:无条件和有条件。

2.4.3.1 无条件转移、调用和返回

1. 无条件转移

当遇到无条件转移时,它总是被执行。在执行期间,PC 装入指定的程序地址且程序从该地址处开始执行。装入 PC 的地址来自调用指令的第2个字或累加器的低16位。

在转移指令到达流水线的执行节拍时,下两条指令已被取回。这两条指令将从流水线中清除而不被执行,然后从转移至的地址处继续执行。无条件转移指令是 B(转移)和 BACC(转移到累加器指定的存储器单元)。

2. 无条件调用

当遇到无条件调用时,它总是被执行。在执行期间,PC 装入指定的程序地址且程序从该地址处开始执行。装入PC 的地址来自调用指令的第2个字或累加器的低16位。在PC 被装载之前,返回地址已被保存在堆栈内。在子程序或函数执行完后,返回指令把来自堆栈的返回地址装入 PC,并从跟在调用之后的那条指令处恢复执行。

在无条件调用指令到达流水线的执行节拍时,下两条指令已被取回。这两条指令字将从流水线中清除从而不被执行,返回地址则被压入堆栈,然后从被调用函数的开始处继续执行。无条件调用指令是 CALL 和 CALA(调用累加器指定的单元地址处的子程序)。

3. 无条件返回

当遇到无条件返回(RET) 指令时,它总是被执行。在无条件返回被执行时,PC 装入堆栈栈顶的值,且从该地址处恢复执行。

在无条件返回指令到达流水线的执行节拍时,下两条指令已被取回。这两条指令字将从流水线中清除从而不被执行,返回地址从堆栈中取出,然后从调用函数之后继续执行。

2.4.3.2 条件转移、调用和返回

F240x 提供仅在满足一个或多个条件时才执行的转移、调用和返回指令。用户把条件规定为条件指令的操作数。表2.14列出了用户可以与这些条件指令一起使用的条件以及它们对应的操作数符号。

1. 使用多个条件

可以把多个条件列为条件指令的操作数。如果列出为多个条件,那么为了执行这些指令,所有条件都必须满足。注意,只有某些条件组合才有意义。对于每一种组合,必须从组1和组2中选择条件,见表2.15。



表2.14 用于条件调用和返回的条件

操作数符号	条 件	说 明
EQ	ACC=0	累加器值等于零
NEQ	ACC≠0	累加器值不等于零
LT	ACC<0	累加器值小于零
LEQ	ACC≤0	累加器值小于或等于零
GT	ACC>0	累加器值大于零
GEQ	ACC≥0	累加器值大于或等于零
C	C=1	进位位被设置为1
NC	C=0	进位位被清除为0
OV	OV=1	检测到累加器溢出
NOV	OV=0	未检测到累加器溢出
BIO	BIO为低	BIO引脚为低电平
TC	TC=1	测试/控制标志位被设置为1
NTC	TC=0	测试/控制标志位被清除为0

表2.15 条件分组

组 1		组 2		
A 类	B 类	A 类	B 类	C 类
EQ	OV	TC	C	BIO
NEQ	NOV	NTC	NC	
LT				
LEQ				
GT				
GEQ				

- 组1, 用户可以选择两个条件。这些条件的每一个必须来自不同的类(A 或 B), 用户不能使用来自同一类的两个条件。例如, 用户可同时测试 EQ 和 OV, 但不能同时测试 GT 和 NEQ。
- 组2, 用户可以选择3个条件。这些条件的每一个必须来自不同的类(A、B或C 类), 用户不能使用来自同一类的两个以上的条件。例如, 用户可以同时测试TC、C 和 BIO, 但是不能同时测试 C 和 NC。

2. 条件状态

条件指令必须能测试状态位的最近值。因此, 直到流水线的第4阶段, 即前一条指令已被执行之后的一个周期, 才认为条件是稳定的。在条件稳定之前, 流水线控制器停止条件指令之后的任何指令的译码。

3. 条件转移

转移指令把程序控制传送到程序存储器中的任何地址。条件转移指令仅当满足用户一个或多个规定的条件(参见表2.14)时才被执行。如果满足所有的条件, 那么PC 将装入转移指令的第2个字, 它包含了将要转移值的地址, 且从该处继续执行。

在条件已被测试时, 条件转移指令后面的两条指令字也被取到流水线中。如果满足所有

条件，那么这两个字将从流水线中清除而不被执行，然后从转移至的地址处继续执行。如果条件不满足，那么转移指令后的两条指令将被执行。因为条件转移使用的条件由前面两条指令的执行而定，所以比非条件转移多用一个周期。

条件转移指令是 BCND(条件转移)和 BANC(若当前辅助寄存器的内容不为0, 则转移)。BANC 指令对实现循环很有用。

4. 条件调用

条件调用(CC) 指令仅在满足规定的一个或多个条件满足时才被执行。这允许用户程序根据被处理的结果在多个子程序之间做出选择。如果满足所有的条件，那么调用指令的第2个字将被装入 PC, 它包含了子程序的起始地址。在转移到子程序之前，处理器把调用指令后的那条指令的地址(即返回地址)存入堆栈。函数必须以返回指令作为结束，它将使返回地址弹出堆栈并使处理器恢复到调用前程序断点处执行。

在条件调用指令的条件已被测试时，调用指令后面的两条指令字也被取到流水线中。如果满足所有条件，那么这两个字将从流水线中清除而不被执行，然后从被调函数开始处继续执行；如果条件不满足，那么调用指令后的两条指令将被执行。由于需要一个等待周期条件才稳定，所以比无条件调用多用一个周期。

5. 条件返回

返回和调用与中断一起使用。调用或中断把返回地址存储到堆栈，然后把程序控制转移到程序存储器中新的单元地址。被调用的子程序或中断服务程序用返回指令结束，该指令把返回地址从堆栈顶弹出并送入 PC。

条件返回指令(RETC) 仅在一个或多个条件(参见表2. 14)被满足时才执行。通过使用 RETC 指令，用户可向子程序或中断服务程序提供多于一个的可能返回路径。被选择的路径取决于所处理的数据。

此外，为了避免在子程序或中断服务程序结束处有条件地转移或者绕过返回指令，也可以使用条件返回指令。

如果满足执行 RETC 的所有条件，那么处理器将把返回地址从堆栈弹出到 PC 并恢复调用或被中断的程序处继续执行。

RETC 和 RET 都是单字指令，但是，由于潜在的 PC 的不连续性，其有效执行时间与条件转移(BCND) 和条件调用(CC) 相同。当条件返回的指令已被测试时，返回指令后面的两条指令字已被取到流水线中。

如果满足所有条件，那么这两个字将从流水线中清除而不执行，然后执行调用程序。

如果条件不满足，那么将执行调用指令后的两条指令。由于需要一个等待周期条件才稳定，所以比无条件调用多用一个周期。

2.4.4 重复单条指令

F240x 的重复单条(RPT) 指令使单条指令执行 $N+1$ 次, 其中 N 为被指定 RPT 指令的操作数。当执行RPT 指令时, 重复计数器 PC 装入 N 。然后重复指令每执行一次, RPTC 便减1, 直到RPTC 等于0为止。当计数器值取自时间存储器单元时, RPTC 可以用作16位计数器。如果计数值规定为常量操作数, 那么它是8位计数器。

重复特性可以与NORM(规格化累加器的内容)、MACD(乘加并带数据移动)以及 SUBC(条件减)这样的指令一起使用。当指令被重复时, 程序存储器的地址和数据总线被释放, 而与数据存储器的地址和数据总线并行取回第2个操作数。这样使得 MACD 和 BLPD 指令在重复时, 能在单个周期内有效的执行。

第 3 章 TMS320LF240x 寻址方式和指令系统

由于F240x 系列DSP 芯片采用与TMS320C240 相同的核，所以两者的指令系统是相同的，读者可以参见TMS320C24x 的指令系统。

3.1 寻址方式

F240x 指令集采用3种基本的存储器寻址方式：立即寻址方式、直接寻址方式和间接寻址方式。

在立即寻址方式中，指令中所需要的常数作为指令的操作数直接给出。立即寻址方式包括短立即寻址和长立即寻址。在短立即寻址中，指令字包含一个8位、9位或13位的操作数，而长立即寻址采用16位的操作数。

当需要访问数据存储器时，用户可采用直接或间接寻址方式。直接寻址方式将指令字的7位与数据存储器页指针(DP) 的9位连接起来，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。

3.1.1 立即寻址方式

在立即寻址方式中，指令字中包含指令所需的一个常数。对于短立即寻址，采用短立即寻址的指令将一个8位、9位或13位的常数作为操作数。短立即寻址指令为一个单指令字，并且有一个常数嵌在该指令中。长立即寻址的指令将一个16位常数作为操作数，从而需要两条指令字。该常数作为第2条指令字被发送，该16位值可以是绝对常数或二进制补码。

【例3.1】采用短立即寻址的RPT 指令

RPT #49 ;将紧跟RPT 指令后的那条指令执行50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	0	1	1	0	0	0	1
采用立即寻址的RPT 代码								8位常数=49							

在例3.1中，立即操作数作为RPT 指令字的一部分。对于该RPT 指令，将用例3.1中的值装载指令寄存器。立即操作数以符号“#”为前缀。

【例3.2】 采用长立即寻址的 ADD 指令，顺次装入的指令寄存器的两个字。

ADD 井65534. 2 ;将数据65534左移两位后. 再将结果加至累加器

第1条指令字:

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

采用立即寻址的ADD 代码 Shift=2

第2条指令字:

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

16位常数=65534=FFFEh

在例3.2中，立即操作数包含在第2条指令字中，该指令寄存器顺次接收如例3.2中的两个指令字的16位值。

3.1.2 直接寻址方式

在直接寻址方式中，数据存储器地址以128为单位被分成若干块，这些块被称为数据页。64K的数据存储器总共包含512个数据页，标号为0~511, 如表3.1所列。当前数据页的状态寄存器 STO 中的9位数据页指针(DP) 值决定。

表3.1 数据存储器页

D P 值	偏移量	数据存储器
000000000	0000000	第0页: 0000h~007Fh
	:	
	1111111	
000000001	0000000	第1页: 0080h~00FFh
	:	
	1111111	
000000010	0000000	第2页: 0100h~017Fh
	:	
	1111111	
:	:	
111111111	0000000	第511页: FF80h~FFFFh
	:	
	1111111	

除数据页之外，处理器还必须知道该页上被访问的特定单元，这取决于7位偏移量，见表3.1。偏移量由指令寄存器的7位最低有效位提供，指令寄存器中包含将被执行的下一条指令。在直接寻址方式中，指令寄存器的内容为：

位15~8 用于指示指令类型(例如 ADD)和指令所访问的数据值的移位信息。

位 7 直接/间接指示符。0表示直接寻址，1表示间接寻址。

位6~0 指示该指令所访问的数据存储器地址的偏移量。

为了得到一个16位地址，处理器将页指针 DP 值和指令寄存器中的7位最低有效位连接起来，即页指针 DP 提供地址的9位高有效位，即页码数；指令寄存器的7位最低有效位提供地址的7位最低有效位，即偏移量。例如，为访问数据地址0050h, 用户须指定DP=000000000和偏移量为1010000, 这样将得到16位地址0000000001010000即0050h。因此，用户必须注意，在程序中初始化 DP。

1. 使用直接寻址方式

使用直接寻址方式时，处理器用页指针(DP) 来寻找数据页，用指令寄存器的低7位有效位来寻址该页上的特定地址。通常做法为：

(1) 设置数据页

将合适的值(0~511)装入 DP。DP 可由LDP 指令装载，也可由向STO 装载的任何指令来装载。LDP 指令直接装载 DP, 并且不影响STO 的其他位。例如：

LDP #32 ;设置当前数据页为32(1000h~107Fh)

(2) 指明偏移量

7位偏移量由指令的操作数提供。例如，如果希望ADD 指令取当前数据页的第5个地址处的数据，则应使用如下指令：

ADD 5H ;将当前数据页中偏移量为5 h 处的数据加至累加器

采用直接寻址时，不必每条指令前都要设置数据页。如果一段代码的所有直接寻址指令都访问同一个数据页，则只需在该段代码的最前面装载一次 DP 值。总之，必须保证在访问新数据页之前改变DP。

2. 直接寻址示例

【例3.3】采用直接寻址的 ADD 指令(移位0~15位)

LDP #4 ;数据页设为4(0200h~027Fh)

ADD 9H,5 ;将数据地址0209h 处的内容左移5位后加至累加器

【例3.4】采用直接寻址的 ADDC 指令

LDP #500 ;数据页设为500(FA00h~FA7Fh)

ADDC 8H ;数据地址FA08h 处内容和进位值(C) 被加至累加器

3.1.3 间接寻址方式

8个辅助寄存器(AR0~AR7) 为处理器提供了强大而灵活的寻址能力，提供间接寻址的16位地址，可以访问64K 数据存储器空间的任意单元。

1. 当前辅助寄存器

通过向状态寄存器 STO 中的3位辅助寄存器指针(ARP) 装入0~7, 可选择特定的辅助寄存器。ARP 可由MAR 指令、LST 指令或任何支持间接寻址的指令来装载。辅助寄存器指针APR 所指的寄存器作为当前辅助寄存器(或简称为当前 AR)。 执行指令过程中，当前 AR 的内容用做被访问的数据存储器地址。

2. 间接寻址选项

TMS320C24x 提供4种间接寻址选项：不增不减，增1或减1, 增加或减去索引量，增加或减去索引量且反向进位。表3. 2给出了与每种选项相应的指令操作数及其示例。

表3.2 间接寻址选项及操作数示例

选 项	操作数	示 例
不增不减	为	LACC :用当前AR所指的数据存储器地址中内容装载累加器
增1	十	LACCw十 ;用当前AR所指的数据存储器地址中内容装载累加器. ;然后当前AR内容加1
减1	二	1ACC — ;用当前AR所指的数据存储器地址中内容装载累加器. ;然后当前AR内容减1
加上索引量	*0+	1ACC0+ ;用当前AR所指的数据存储器地址中内容装载累加器. ;然后当前AR内容加上AR0的内容
减去索引量	0-	1ACC0— ;用当前AR所指的数据存储器地址中内容装载累加器. ;然后当前AR的内容减去ARo的内容
加上索引量，反向进位	BRO十	LACC BRO+;用当前AR所指的数据存储器地址中内容装载累加器. ;然后当前AR内容加上AR0的内容. 该加法采用反向进位传送
减去索引量. 反向进位	*BRO-	LACC BRO+;用当前AR所指的数据存储器地址中内容装载累器， ;然后当前AR内容减去AR0的内容，该减法采用反向进位传送

3. 辅助寄存器

除了更新当前辅助寄存器的内容以外，某些指令还可以指明下一个辅助寄存器。当本条指令执行完之后，下一个辅助寄存器便成为当前辅助寄存器。

【例3.5】 选择一个当前的辅助寄存器

MAR *,AR1 ;将当前辅助寄存器设为 AR1
 LACL * + , AR2 ;用辅助寄存器 AR1 所指向的地址中内容装载累加器的低16位,
 ;AR1 内容加1. 使得 AR2 为当前辅助寄存器
 SACL * + ;将累加器的低16位存于 AR2 所指向的地址单元, AR2 内容加1

4. 间接寻址操作码格式

在间接寻址中, 指令寄存器的内容为:

位15~8 用于指示指令类型(例如 ADD)和指令所访问的数据值的移位信息
 位 7 直接/间接指示符。0表示直接寻址, 1表示间接寻址。
 位6~4 辅助寄存器更新代码 ARU。该3位决定当前辅助寄存器是否以及如何进
 增或减。
 位 3 下一辅助寄存器指示符 N。该位说明该指令是否改变辅助寄存器指针 ARP
 值。N=0, 辅助寄存器指针 ARP 内容保持不变; N=1, 下一个辅助寄存器
 NAR 的内容被装入辅助寄存器指针 ARP。
 位2~0 下一辅助寄存器的值。该3位包括下一辅助寄存器的值。若 N=1, 则 NAR
 被装入 ARP。

5. 间接寻址示例

【例3.6】 不增不减

ADD *.8 ;把当前辅助寄存器指向的数据存储器地址的内容左移8位之后
 ;加至累加器

【例3.7】 减 1

ADD *-,8 ;除了例3.6操作之外. 将当前辅助寄存器内容减1

6. 修改辅助寄存器内容

用于修改辅助寄存器内容的特定指令有: LAR、ADRK、SBRK 和 MAR。LAR 指令装载 AR;ADRK 和 SUBRK 分别用来从 AR 中加上或减去一个立即数; MAR 指令可使 AR 值增加/减少1或增加/减少一个索引量。另外任何支持间接寻址操作数的指令都可修改辅助寄存器。详细介绍参见指令集部分。

3.2 指令集

本节根据指令的功能, 分为6张表来说明指令集概况: 累加器、算数和逻辑指令见表3.3;辅助寄存器和数据页指针指令见表3.4;TREG、PREG 和乘法指令见表3.5;转移指令见表3.6;控制指令见表3.7;I/O 和存储器指令见表3.8。

表3.3 累加器、算数和逻辑指令

助记符	指令功能	周期	寻址方式	指令说明
ABS	$(ACC) \rightarrow ACC.0 \rightarrow C$	1		ACC内容取绝对值
ADD	$(ACC) + (\text{数据存储器地址}) \times 2 \text{shif} \rightarrow ACC$	-	直接/间接	移位时低位填0。若SXM=1.高位用符号扩展；若SXM=0.高位填0.结果存在ACC中.该指令使C=0。寻址短立即数时.加操作不受SXM的影响，且不能重复执行
	$(ACC) + (\text{数据存储器地址}) \times 26 \rightarrow ACC$	-	直接/间接	
	$(ACC) + k \rightarrow ACC$	-	短立即数k	
	$(ACC) - 1k \times 2 \text{sh} \rightarrow ACC$	2	长立即数1k	
ADDC	$(ACC) + (\text{数据存储器地址}) + (C) \rightarrow ACC$	1	直接/间接	该指令抑制符号扩展
ADDS	$(ACC) + (\text{数据存储器地址}) \rightarrow ACC$	1	直接/间接	该指令抑制符号扩展。无论SXM为何值.数据均作为符号16位数看待。当SXM=0和移位次数等于0时. ADDS与ADD指令结果相同
ADDT	$(ACC) + (\text{数据存储器地址})$ 左移(TREG(3:0))位 $\rightarrow (ACC)$	1	直接/间接	被寻址的数据寄存器单元的内容左移并加到ACC.移位次数由TREG的低4位确定 SXM位的值控制移位时是否作符号扩展
AND	$(ACC(15:0)) \text{ AND } (\text{数据存储器地址})$ $ACC(15:0)$ $0 \rightarrow ACC(31:16)$	1	直接/间接	使用直接/间接寻址。ACC的低位字与数据存储器单元的值作“与”运算，结果存在ACC的低位、ACC的高位清0;使用立即数寻址，长立即数可以左移.移位时，32位中未被长立即操作数填充的位均清0
	$(ACC31:0) \text{ AND } 1k \times 2 \text{hd} \rightarrow ACC$	2	长立即数	
	$(ACC31:03) \text{ AND } 1k \times 2)^6 \rightarrow ACC$	2	长立即数	
CMPL	(ACC)的逻辑反 $\rightarrow ACC$	1		累加器中的内容用其逻辑反取代
L. AC ^o C	$(\text{数据存储器地址}) \times 2 \text{hif} \rightarrow ACC$	-	直接/间接	指定的数据存储器单元的内容或16位常数左移并加载到ACC.低位填0。若SXM=1.高位用符号位扩展；若SXM=0.则填0
	$(\text{数据存储器地址}) \times 21 \rightarrow ACC$	1	直接/间接	
	$1k \times 2 \text{shf} \rightarrow ACC$	2	长立即数	
LACL	$(\text{数据存储器地址}) \rightarrow ACC15:0)$	1	直接/间接	指定的数据存储器单元的内容或用0扩展的8位常数左移并加载到ACC的低16位。ACC的高位填0。数据作为16位无符号数处理。无论SXM为何值.本指令都不进行符号扩展
	$0 \rightarrow ACC(31:16)$			
	$k \rightarrow ACC(7:0)$	1	短立即数	
	$0 \rightarrow ACC(31:8)$			
LACT	(数据存储器地址) 左移 TREG(3:0)位 $\rightarrow (ACC)$	1	直接/间接	将数据存储器单元的内容左移后加载到ACC.左移位数由TREG的低4位确定。 SXM的值决定ACC高位是否符号扩展
NEG	$(ACC) \times (-1) \rightarrow ACC$	1		将ACC的内容换成其相反数。当对8000 0000h作NEG操作时，OV置1 若(VM=1,则ACC=7FFFFFFFh:若OVM=0,则ACC=8000.0000h。若ACC≠0.则C=0:若ACC=0.则C=1

续表3. 3

助记符	指令功能	周期	寻址方式	指令说明
NORM	若 $(ACC) = 0$, 则 $TC \rightarrow 1$ 否则, 若 $(ACC(31)) \text{ XOR } (ACC(30)) = 0$, 则 $TC \rightarrow 0$, $(ACC) \times 2 \rightarrow ACC$ 按指定的方式修改AR. 否则 $TC \rightarrow 1$	1	间接	NORM指令将ACC中的有符号数进行规格化。对定点数进行规格化即将其分成指数和尾数, 为此对位31和位30作“异或”运算即可确定位30是数值还是符号。对当前AR的缺省修改方式是加1;该指令对正、负二进制补码均适用。由于该指令在流水线的第4阶段才完成对AR的操作, 而其他指令在流水线的第2阶段完成对AR的操作, 因此紧跟在NORM指令后的两条指令不能对AR值或者ARP值进行修改; 否则NORM指令将不能操作正确的数据存储器单元
OR	$(ACC(15:0)) \text{ OR } (\text{数据存储器地址}) \rightarrow ACC(15:0)$ $(ACC(31:16)) \rightarrow ACC(31:16)$ $(ACC) \text{ OR } 1k \times 2 \text{ shif} \rightarrow ACC$ $(ACC) \text{ OR } 1k \times 216 \rightarrow ACC$	1 2 2	直接/间接 长立即数 长立即数	ACC的内容和被寻址的数据地址单元的值或左移后的立即数作或操作。SXM位对OR指令没有影响。数据操作数没有被占用的位总是填0;因此在直接或间接寻址或不移位的长立即数的情况下, ACC的高位是不变的。当操作数是立即数且移位次数不等于()时, 操作数左移低位补0
ROL	$(ACC(31)) \rightarrow C$ $(ACC(30:0)) \rightarrow ACC(31:1)$ $C \rightarrow ACC(0)$	1		将ACC内容左移一位, 进位位被移入最低位, 最高位被移入进位位
ROR	$C \rightarrow ACC(31)$ $(ACC(31:1)) \rightarrow ACC((30:0))$ $(ACC(0)) \rightarrow C$	1		将ACC内容右移一位, 进位位被移入最高位, 最低位被移入进位位
SACH	$((ACC) \times 2 \text{ shift})$ 的高16位→ 数据存储器地址	1	直接/间接	SACH将整个ACC的32位数左移0~7位, 将移位后数值的高16位保存到数据存储器
SACL	$((ACC) \times 2 \text{ shilt})$ 的低16位→ 数据存储器地址	1	直接/间接	SACL将整个ACC的32位数左移0~7位, 将移位后数值的低16位保存到数据存储器
SFL	$(ACC(31)) \rightarrow C$ $(ACC(30:0)) \rightarrow ACC(31:1)$ $0 \rightarrow ACC(0)$	1		SFL指令使ACC的内容左移一位, ACC的最低位填0. 最高位移入进位位
SFR	若 $SXM = 0$, 则 $0 \rightarrow ACC(31)$ 若 $SXM = 1$, 则 $ACC(31) \rightarrow ACC(31)$ $(ACC(31:1)) \rightarrow ACC(30:0)$ $(ACC(0)) \rightarrow C$	1		SFR指令将ACC的内容右移一位。若 $SXM = 1$, 则产生算数右移, ACC的符号位不变, 位31复制到位30, 位0移入进位位; 若 $SXM = 0$, 则产生逻辑右移, ACC的所有内容右移一位, 位0移入进位位, 位31填0

续表3. 3

助记符	指令功能	周期	寻址方式	指令说明
SUB	$(ACC) - (数据存储器地址) \times 2 \xrightarrow{if} ACC$ $(ACC) - (数据存储器地址) \times 2 \rightarrow ACC$ $(ACC) - k \rightarrow ACC$ $(ACC) - 1k \times 2 \xrightarrow{shl} ACC$	2	直接/间接 直接/间接 短立即数 长立即数	在直接/间接和长立即数寻址方式里被寻址的数据存储单元的内容或16位常数左移. 然后累加器减去移位后的值。移位时低位填0: 高位扩展与否由SXM决定。当使用短立即数时. 累加器减去8位正常数; 此时不可以指定移位值. 减法不受SXM的影响, 并且是不可重复的。通常情况下, 若减法结果有借位, 则进位位置0: 若减法结果没有借位, 则进位位置1。但若移位次数为16, 则当减法结果有借位时, 进位位置0, 减法不产生借位时, 进位位置不变
SUBB	$(ACC) - (数据寄存器地址) - (C的逻辑反) \rightarrow ACC$	1	直接/间接	ACC的值减去被寻址的数据寄存单元的内容及进位位的逻辑反值, 抑制符号扩展。若相减结果产生借位. 则进位位置0; 若没有借位, 则进位位置1
SUBC	若 $(ACC) \geq 0$ 且 $(数据寄存器地址) \geq 0$. 则 $(ACC) - (数据寄存器地址) \rightarrow$ ALU输出 若ALU输出 ≥ 0 . 则 $(ALU输出) \times 2 + 1 \rightarrow$ ACC 否则 $(ACC) \times 2 \rightarrow ACC$	1	直接/间接	ACC实现条件减. 可用于除法: 把16位的正被除数放在ACC的低16位. ACC的高位清0。16位的正除数放在数据存储单元中; 执行SUBC指令16次。最后一次SUBC指令完成后ACC的低16位是除数的商, ACC的高16为是余数。若ACC和存储单元的内容为负. 则不能用SUBC实现除法; 若16位被除数的有效数字少于16位可将它放在ACC中并左移, 左移的位数为前面非有效的0的个数. SUBC指令执行的次数为16减去移位次数。执行该指令不会因正溢或负溢而饱和
SUBS	$(ACC) - (数据寄存器地址) \rightarrow ACC$	1	直接/间接	ACC减去数据存储单元的内容. 抑制符号扩展。不管SXM为何值. 数据存储单元的内容都作为无符号处理。ACC还是作为有符号数 若相减结果有借位则清0. 若没有借位则置1
SUBT	$(ACC) - (数据寄存器地址) \times$ $2 \xrightarrow{fTREG} ACC$ 若SXM=1. 则(数据存储单元)用符号扩展 若SXM=0. 则(数据存储单元) 不用符号扩展	1	直接/间接	ACC减去左移后的数据存储单元的内容 移位次数由TREG的低4位确定。SXM控制是否对数据存储单元的内容作符号扩展

续表3. 3

助记符	指令功能	周期	寻址方式	指令说明
XOR	ACC(15:0))XOR(数据寄存器地址)→ACC(15:0) (ACCC31:16))→ACC(31:16)	1	直接/间接	直接和间接寻址时ACC的低16位和被寻址的数据存储单元的内容作“异或”运算，结果存在ACC的低16位，ACC的高16位不变；使用长立即数寻址时，长立即数左移，数据操作数没有占用的位填0. 然后与ACC中的32位数作“异或”运算
	或(ACC(31:0))XOR 1k×2shift→ACC(31:0) 或(ACC(31:0))XOR 1k×216→ACC(31:0)	2	长立即数	
ZALR	(数据存储单元地址)→ACC(31:16) 8000h→ACC(15:0)	1	直接/间接	数据存储单元的内容送到ACC的高16位，ACC的低15位清0, 第15位置1

表3. 4 辅助寄存器指令

助记符	指令功能	周期	寻址方式	指令说明
ADRK	(当前AR)+k→当前AR	1		8位立即数按右对齐方式与当前辅助寄存器值相加
BANZ	若(当前AR)≠0, 则操作数→PC; 否则(PC)+2→PC 按指令要求的方式修改当前AR和ARP	4	间接	若当前辅助寄存器的内容不为0. 程序控制转到指定的程序存储器地址；否则执行下一条指令。对当前AR缺省的修改方式是减1; 在进入循环前将辅助寄存器(作循环计数器)初始化为N-1, 可执行N次循环
		2		
CMPR	(当前AR)与(ARO)比较, 并把结果放在状态寄存器STI的TC位	1	直接/间接	该指令完成由操作数指定的比较: 若CM==00, 则测试是否当前AR=ARO; 若CM==01, 则测试是否当前AR<ARO; 若CM==10. 则测试是否当前AR>ARO; 若CM==11. 则测试是否当前AR=ARO。 若条件为“真”, TC位置1; 否则TC位清0
LAR	(数据存储单元地址)→ARx k→ARx 1k→ARx	2	直接/间接 短立即数 长立即数	指定数据寄存器单元中的内容。8位常数或16位常数加载到指定的辅助寄存器。不论SXM为何值, 所指定的常数均为无符号整数
MAR	按指令指定的方式修改(当前AR)和(ARP)	1	间接	直接寻址方式下MAR指令的作用同NOP指令。间接寻址方式下可以修改辅助寄存器的值和ARP的值, 对寄存器的引用不起作用
SAR	(ARx)→数据寄存器地址	1	直接/间接	将指定的辅助寄存器(ARx)的内容复制到指定的数据存储单元
SBRK	(当前AR)-K→当前AR	1		当前辅助寄存器(由ARP指定的那个)的内容减去8位立即数, 结果存在当前辅助寄存器中

表3.5 TREG、PREG 和乘法指令

助记符	指令功能	周期	寻址方式	指令说明
APAC	ACC) 十移位后的 (PREG) →ACC	1		按ST1寄存器中PM状态位指定的方式将PREG的内容进行移位，把移位后的值与ACC的内容相加。状态寄存器中的SXM位对APAC指令无影响
LPH	(数据寄存器地址) →PREG (31:16)	1	直接/间接	被寻址的数据寄存单元的内容加载到PREG的高16位：PREG的低16位不变
LT	(数据寄存器地址) →TREG	1	直接/间接	被寻址的数据寄存单元的内容加载到TREG
LTA	(数据寄存器地址) →TREG (数据寄存器地址) 十移位后的 (PREG) →ACC	1	直接/间接	指令的数据寄存单元的内容加载到TREG；按PM状态位指定的方式对乘积寄存器的内容进行移位，并把移位后的值与ACC相加，结果放在ACC中。若相加结果有进位则C=1，否则C=0
1. TD	(数据寄存器地址) →TREG (数据寄存器地址) →数据寄存器地址+1 (ACC) 十移位后的 (PREG) →ACC	1	直接/间接	指令将数据寄存单元的内容加载到TREG。按PM状态位指定的方式对乘积寄存器的内容进行移位，并把移位后的值与ACC相加. 结果放在ACC中。指定的数据存储单元的内容复制到地址加工的数据存储单元。该指令移动数据的功能不能用于外部数据寄存器或存储器映射的寄存器
L. TP	(数据寄存器地址) →TREG 移位后的 (PREG) →ACC	1	直接/间接	指令的数据寄存单元的内容加载到TREG按PM状态位指定的方式对乘积寄存器的内容进行移位，并把移位后的值送到ACC中
LTS	(数据寄存器地址) →TREG ACC一移位后的 (PREG) →ACC	1	直接/间接	指令的数据寄存单元的内容加载到TREG按PM状态位指定的方式对乘积寄存器的内容进行移位，并用ACC的内容减去 (PREG) 移位后的值送到ACC中。若相减的结果产生了借位，则进位位清0; 否则进位位置1
MAC	PC+1 即 PC=PC+1 PC) →MSTACK 16位程序存储器地址→PC (ACC) 十移位后的 (PREG) →ACC (数据存储器地址) →TREG (数据存储器地址) × (16位程序存储器地址) →PREG 间接寻址时. 按指定方式修改 (当前AR) 和 (ARP) 当 (重复计数器) ≠0时. (重复计数器)-1→重复计数器 (MSTACK) →PC	3	直接/间接	MAC指令可以 (1) 按PM状态位指定的方式把先前的乘积移位，再与ACC的内容相加。 (2) 把指定的数据存储单元的内容加载到TREG (3) 将数据存储单元的内容乘以指定的程序存储器地址中的内容：若程序存储器是片内RAM块D0), 则CNF位必须置1； 当重复MAC指令时. 每重复一次包含在PC中的程序存储器地址加1 若使用间接寻址指定数据存储器地址. 则每次重复时就可以访问新的数据存储器地址；若使用直接寻址方式指定的数据存储器地址是常数. 重复时不会对其进行修改

续表3. 5

助记符	指令功能	周期	寻址方式	指令说明
MACD	操作与MAC相同，增加了(数据存储器地址)→数据存储器地址+1	3	直接/间接	该指令与MAC的不同之处，在于片内增加了数据的移动
MPY	$(TREG) \times (数据存储器地址) \rightarrow PREG$ $(TREG) \times k \rightarrow PREG$	1	直接/间接	TREG的内容乘以被寻址的数据存储单元的内容。使用短立即数时TREG的内容乘以有符号的13位常数；短立即数值右对齐，相乘前. 无论SXM如何都将该常数用符号扩展
MPYA	$(ACC) + 移位后的(PREG) \rightarrow ACC$ $(TREG) \times (数据存储器地址) \rightarrow PREG$	1	直接/间接	TREG的内容乘以被寻址的数据存储单元的内容。按PM状态位指定的方式对先前的乘积进行移位，并将移位后的值加到ACC中
	$(ACC) - 移位后的(PREG) \rightarrow ACC$ $(TREG) \times (数据存储器地址) \rightarrow PREG$	1	直接/间接	TREG的内容乘以被寻址的数据存储单元的内容。按PM状态位指定的方式对先前的乘积进行移位，并从ACC中减去移位后的值，结果放在ACC中
MPYU	无符号数 $(TREG) \times$ 无符号数(数据存储器地址)→PREG	1	直接/间接	TREG的无符号数乘以被寻址的数据存储单元的无符号数。作无符号相乘，不能使用PM=3这种移位模式，因为此时该移位器总要将PREG的值进行符号扩展
PAC	移位后的(PREG)→ACC	1		按指定的方式将PREG的内容移位，并把移位后的结果加载到累加器
SPAC	$(ACC) - 移位后的(PREG) \rightarrow ACC$	1		SPAC不受SXM影响，PREG的值总是要作符号扩展
SPH	(PREG) 移位后的高16位→数据存储器地址	1	直接/间接	按PM指定的方式把SPEG的内容移位，移位后的高16位数值存到数据存储单元。若右移6位，则高位用符号扩展，低位丢失；若左移，则高位丢失而低位填0。PREG和ACC中的值都保持不变
SPL	(PREG) 移位后的低16位→数据存储器地址	1	直接/间接	按PM指定的方式把SPEG的内容移位，移位后的低16位数值存到数据存储单元。若右移6位，则高位用符号扩展，低位丢失；若左移，则高位丢失而低位填0。PREG和ACC中的值都保持不变
SPM	常数→乘积移位模式(PM) 位	1		指令字中的最低2位复制到状态寄存器ST1的乘积移位模式PM位。PM控制PREG输出移位器的移位模式

续表3. 5

助记符	指令功能	周期	寻址方式	指令说明
SQRA	(ACC) 千 (PREG) 移位后的值→ACC (数据存储器地址)→TREG (TREG) × (数据存储器地址)→PREG	1	直接/间接	按PM位指定的方式把PREG的内容移位, 并将其加到累加器。被寻址的数据存储单元的值加载到TREG, 计算其平方
SQRS	(ACC)= (PREG) 移位后的值→ACC (数据存储器地址)→TREG (TREG) × (数据存储器地址)→PREG	1	直接/间接	按PM位指定的方式把PREG的内容移位, 累加器减去移位后的值。被寻址的数据存储单元的值加载到TREG. 计算其平方

表3. 6 转移指令

助记符	指令功能	周期	指令说明
B	16位程序存储器地址→PC 间接寻址	4	按指令要求的方式修改当前辅助寄存器和ARP的内容。把控制转到指令指定的程序存储器地址
BACC	ACCC15:0) →PC	4	控制转换到ACC的低位字所指定的地址
BANZ	若 (当前AR) ≠0. 则16位程序存储器地址→PC 否则 (PC) +2→PC	4 2	如果当前辅助寄存器的内容不为0, 则程序控制转到指定的程序存储器地址; 否则执行下一条指令。对当前AR缺省的修改方式是减1
BCND	若条件都满足. 则16位程序存储器地址→PC 否则PC增量	4 1	如果指定的条件都满足. 则分支到指定的程序存储器地址 注意不是所有的条件组合都有意义 条件EQ表示ACC=0: NEQ表示ACC≠0 条件LT表示ACC<0; LEQ表示ACC≤0 条件GT表示ACC>0: GEQ表示ACC≥0 条件NC表示C=0; C表示C=1 条件NOV表示OV=0; OV表示OV=1 条件BIO引脚为低表示C=0; C表示C=1 条件NTC表示TC=0: TC表示TC=1 UNC标志无条件
CALA	PC+1→TOS ACC (15:0) →PC	4	当前程序计数器 (PC) 加1后压入栈顶 (TOS); 再将累加器的低位字加载到PC. 程序从该地址继续运行
CALL	PC+2→TOS 16位程序存储器地址→PC 按指令要求的方式修改 (AR) 和 (ARP)	4	当前程序计数器 (PC) 加2后压入栈顶 (TOS); 再将程序存储器地址的内容加载到PC 该指令为间接寻址方式
CC	若条件都满足则PC+2→TOS 且16位程序存储器地址→PC 否则PC增量	4 2	若指定的条件都满足, 则控制转到指定的程序存储器地址。同样应注意, 不是所有的条件组合都有意义

续表3.6

助记符	指令功能	周期	指令说明
INTR	(PC)+1→堆栈 相应的中断向量地址→PC	4	处理器有32个中断向量，每个K(0~31)值代表一个中断向量。该指令是软件中断，它使程序控制转换到与K值对应的地址. 有该地址引导到相应的中断服务程序。INTM位和中断屏蔽位都不影响INTR指令
NMI	(PC)+1→堆栈 24h→PC 1→INTM	4	该指令将程序计数器强置为不可屏蔽中断向量地址24H。它与硬件不可屏蔽中断NMI的效果相同
RET	(TOS)→PC 堆栈上弹一级	4	子程序和中断服务程序以RET指令结束，使程序控制返回到调用程序或被中断的程序
RET	如果条件都满足，则 (TOS)→PC 同时堆栈上弹一级，否则继续	4 2	如果条件都满足，则执行标准的返回。但不是所有的条件组合都有意义
TRAP	(PC)+1→堆栈 22h→PC	4	该指令是软件中断使控制转换到程序存储单元22h, PC加1后推入硬件堆栈。在程序空间22h可以存放分支命令使控制转到TRAP例程。TRAP指令是不可屏蔽的

注：对于表中的条件指令，如果条件成立，则所需的指令周期为4;如果条件不成立，则所需的周期为2。

表3.7 控制指令

助记符	指令功能	周期	寻址方式	指令说明
BIT	(数据位(15-位码))→TC	1	直接/间接	该指令将数据存储单元中被指定位的值复制到状态寄存器ST1中的TC位
BITT	(数据位(15-TREG(3:0)))→TC	1	直接/间接	该指令将数据存储单元中被指定位的值复制到状态寄存器ST1中的TC位
CLRC	0→控制位	1		把指定的控制位清0;也可用LST指令加载ST0及STI。控制位有：C、CNF、INTM、OVM、SXM、TC和XF
IDLE	PC增量，等待未被屏蔽的或不可屏蔽的硬件中断	2		该指令使程序停止运行，直至CPU接收到没被屏蔽的硬件中断(内部或外部的),NMI或复位。该指令使C2xx进入低功耗模式。C2xx在进入低功耗模式前将PC加1。在空闲状态中PC不变。即使INTM为1未被屏蔽的中断也可使C2xx退出空闲状态。当未被屏蔽的中断使C2xx退出空闲状态后.CPU的工作取决于INTM:若INTM为0,程序分支到它所响应的中断服务程序；若INTM为1,程序从IDLE后面的指令继续执行。因NMI或复位退出空闲状态时，不管INTM为何值都执行相应的中断服务程序

续表3. 7

助记符	指令功能	周期	寻址方式	指令说明
LDP	(数据寄存器地址) 的低9位→DP 或k→DP	2 2	直接/间接 短立即数	被寻址的数据寄存单元的低9位或9位的立即数值加载到状态寄存器ST0中的DP. 也可以用LST指令加载DP
LST	(数据存储器地址) → 状态寄存器STm	2	直接/间接	用被寻址的数据寄存器的值加载状态寄存器但应注意：(1) “LST≠0” 操作不影响STI寄存器中的ARB字段；(2) “LST并1”操作加载到ARB的值也加载到ARP；(3) 状态寄存器中的保留位总是读1. 写入对其无影响，(4) 间接寻址时若操作数指定下一AR值，则用所寻址的数据存储单元中的；高3位加载ARP；子程序调用或中断后可用LST指令恢复状态寄存器
NOP	PC+1 即 PC=PC+1	1		该指令只影响PC. 不作其他任何操作。它可用于建立流水线和延时
POP	(TOS)→ACC(15:0) 0→ACC(31:16) 堆栈上弹一级	1	直接/间接	硬件堆栈是后进先出的8个单元。若弹出的次数多于7次，则堆栈中的所有值将都相同；没有检查堆栈是否下溢的措施
POPD	(TOS)→数据存储器地址 堆栈上弹一级	1	直接/间接	栈顶的值弹出并传送到指令指定的数据存储单元。其他操作同POP. 没有检查堆栈是否下溢的措施
PUSH	(数据存储器地址) → 堆栈下压一级 堆栈下压一级 ACC(15:0)→TOS	1 1		将指令指定的数据存储单元的内容传送到栈顶。堆栈中低7个单元的值都下移一级。堆栈中最低一级单元的内容丢失
RPT	(数据存储器地址) RPTC k→RPTC	1 1	直接/间接 短立即数	直接或间接寻址时，被寻址的数据存储单元的内容加载到重复计数器(RPTC), 使用短立即数寻址时. 8位立即数加载到RPTC。RPT后面的那条指令重复n次. n为RPTC的初始值加1, 该指令是不可中断的，器件复位时RPTC清0。重复指令本身不能重复
SETC	1→控制位	1		把指定的控制位置1, 也可用LST指令加载ST0和ST1
SPM	常数→乘积移位模式PM 位常数为0—3	1	直接/间接	指令字中的最低2位复制到状态寄存器ST1的乘积移位模式PM位 (ST1的位1和位0)
SST	(状态寄存器TSm) → 数据寄存器地址	1	直接/间接	直接寻址时，不论ST0中的数据页面指针DP为何值，指定的状态寄存器总是被保存到0页。不必修改DP, 执行该指令时. 处理器自动访问0页。使用间接寻址时. 由所选用的辅助寄存器指定存储地址；因此状态寄存器的值可以保存到数据寄存器的任何页面内

表3.8 I/O 和存储器指令

助记符	指令功能	周期	指令说明
BLDD	(PC) → MSTACK. 1 k → PC. (源) → 目的 间接寻址时按指令指定的方式修改(当前AR)和(ARP) (PC)+1 → PC 当(重复计数器) ≠ 0时 (源) → 目的 间接寻址时按指令指定的方式修改(当前AR)和(ARP) (PC)+1 → PC (重复计数器)-1 → 重复计数器 (MSTACK) → PC	3	把“源”指定的数据存储器中的字复制到“目的”指定的数据存储单元。源和目的地址可由长立即数地址或数据存储器地址指定。但必须是：如果源地址为长立即数，则目的地址只能为直接或间接；如果源地址为直接或间接，则目的地址只能为长立即数。该指令不能用于存储器映射的寄存器。使用RPT指令重复BLDD操作期间，中断被禁止
BLPD	(PC) → MSTACK. 16位程序存储器地址 → PC, (源) → 目的 间接寻址时按指令指定的方式修改(当前AR)和(ARP) (PC)+1 → PC. 当(重复计数器) ≠ 0时 (源) → 目的 间接寻址时按指令指定的方式修改(当前AR)和(ARP) (PC)+1 → PC (重复计数器)-1 → 重复计数器 (MSTACK) → PC	3	把“源”指定的程序存储器中的字复制到“目的”指定的数据存储单元。源空间的第一个字由长立即数指定。目的地址可由辅助寄存器或数据存储器地址指定。使用RPT指令重复BLDD操作期间，中断被禁止
DMOV	(数据存储器地址) → 数据存储器地址+1	1	指定的数据存储单元的内容复制到地址加1的单元，复制时原来单元中的内容保持不变。该指令只能用于片内数据RAM块。可以工作于任何配置的RAM块，但要求这些RAM块配置为数据存储器。数据的移动可以跨越块的边界进行。此功能不能用于外部数据存储器。若指令指定了外部存储器地址，该指令读指定的存储单元但不进行其他操作。数据移动对于实现DSP总的Z(-1)延时很有用处
IN	PA → 地址总线A15~A0 数据总线D15~D0 → 数据存储器地址 (PA) → 数据存储器地址	2	该指令从I/O单元读入16位值，将其送到指定的数据存储单元。IS线变低表示进行I/O访问。直接或间接寻址
OUT	PA → 地址总线A15~A0 (数据存储器地址) → 数据总线D15~D0 (数据存储器地址) → PA	3	该指令将数据存储单元中的16位值写到指定的I/O端口。IS线变低表示进行I/O访问。直接或间接寻址
SPLK	1k → 数据存储器地址	2	将16位常数写到任意的数据存储单元

续表3. 8

助记符	指令功能	周期	指令说明
TBLR	(PC) →MSTACK (ACC(15:0)) →PC (16位程序存储器地址) →数据存储器地址 间接寻址时按指令指定的方式修改(当前AR)和(ARP) (PC)+1→PC 当(重复计数器) ≠0时 (16位程序存储器地址) →数据存储器地址 间接寻址时按指令指定的方式修改(当前AR)和(ARP) (PC)+1→PC (重复计数器)-1→重复计数器 (MSTACK) →PC	3	该指令将程序存储单元中的一个字传送到指令指定的数据存储器单元，程序存储单元中的地址由累加器的低16位指定。该指令先从程序存储单元读出，然后写入指定的数据存储器单元，直接或间接寻址
TBLW	(PC+1) →MSTACK (ACC(15:0)) →PC+1 (数据存储器地址) → 16位程序存储器地址 间接寻址时按指令指定的方式修改(AR)和(ARP) (PC)+1→PC 当(重复计数器) ≠0时 (重复计数器)-1→重复计数器 (MSTACK) →PC+1	3	该指令将数据存储器单元中的一个字传送到指令指定的程序存储单元，数据存储器单元的地址由指令指定. 程序存储器地址由累加器的低16位确定。该指令先从数据存储器单元读出. 然后写入指定的程序存储单元。直接或间接寻址

在每张表中，指令按字母顺序排列。执行每条指令所需要的周期数在表中给出，所有指令都假设从内部程序存储器和内部数据存储器中执行，指令的周期数适用于单指令执行，不适用于重复方式。编程时，由于用户必须对每条指令的寻址方式清楚了解，因此在表中给出了每条指令的寻址方式。由于指令的操作码对用户编程没有多大指导意义，因此在这里就未列出。

为了方便起见，先给出表3. 3～表3. 8这6张表的符号的意义。

- ACC
- 累加器。
- AR
- 辅助寄存器。
- ARX
- 用于LAR 和 SAR 指令的3位数据值. 指定被操作的辅助寄存器。
- BITX
- 4 位数值，用于指定数据存储器中的哪一位将被 BIT 指令所测试。
- CM
- 2位数值，CMPR 指令执行 CM 值所声明的比较：
若 CM=00， 则测试是否(当前 AR)=(AR0);
若CM=01 则测试是否(当前AR)<(AR0);
若CM=10 则测试是否(当前 AR)>(AR0):
若CM=11, 则测试是否(当前AR)≠(AR0)。

SHIFT 4 位移位值。

TP 用于条件执行指令的2位数值，代表如下4种条件：若 BIO 引脚为低，则 TP=00；若 TC 位=1，则 TP=01；若 TC 位=0，则 TP=10；若无条件，则 TP=11。

3.3 典型指令说明

本节将在指令集的基础上，以几个常用的指令为例，说明在编程过程中如何使用指令集给出的指令。举例时，假定直接寻址一律认为 DP 指针已经指向要寻址的数据区，不用再重新装载 DP，而间接寻址则认为辅助寄存器指针 ARP 已经指到当前 AR，也不用再声明当前 AR。

1. 对累加器的加操作指令 ADD

ADD 指令执行的操作是将数据存储器单元的数或立即数左移后加至累加器。移位时，低位填0，高位在 SXM=1 时为符号扩展，在 SXM=0 时填0，结果存在累加器中。寻址短立即数时，加操作不受 SXM 的影响，且不能重复执行。

ADD	5.2	;(DP=4:0200h~027h) 将数据存储器单元0205h 的内容左移2位 ;之后与 ACC 相加，结果存在 ACC中
ADD	*十， 2, ARO	;(ARP=4, AR4=282) 将数据存储器单元0282h 的内容左移2位 ;之后加至 ACC, 结果存在 ACC 中，指令执行后 AR4=283, ARP=0
ADD	#2	;短立即数2与 ACC 相加，结果存在 ACC中
ADD	#1111h.2	;长立即数1111h 左移2位后与 ACC 相加 • 结果存在 ACC中

2. 和累加器逻辑“与”操作指令 AND

AND 指令用来实现被寻址单元的内容和累加器的逻辑“与”操作，以及长立即数经过移位之后和连接器进行逻辑“与”操作。逻辑“与”操作之后的结果保存在累加器中。

AND	16	;(DP=4:0200~027Fh) 将数据存储器单元0210h 的内容与 ;ACC 的内容进行逻辑“与”操作. 结果保留在ACC 中
AND	*	;(ARP=0, AR0=0301h) 将数据存储器单元0301h 的内容与ACC 的 ;内容进行逻辑“与”操作，结果保留在ACC 中
AND	#00FFh.4	;将立即数00FFh 左移4位之后和 ACC 逻辑“与”，结果保留在 ACC 中

3. 辅助寄存器不等于零转移指令 BANZ

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；否则控制转移到下一条指令。当前 AR 的缺省修改为减1。该指令可用来实现程序的循环执行。

MAR	*, ARO	;ARP 指向 ARO
LAR	AR1, #3	;AR1 中装入3

	LAR	ARo.#60h	;ARO 中装入0060h
PI	ADD	* + .AR1	;将 ARO所指的数加到 ACC.并将 ARO的值加1. ARP 指向 AR1
	BANZ	P1.ARO	;若 AR1≠0 则循环

4. 条件转移指令BCND

当所规定的条件符合时，控制转移到指定的程序存储器地址。

BCND PILEQ :若 ACC的内容小于等于0. 则程序转到PI 处开始执行

5. 位测试指令BIT

该指令将数据存储器中的指定位的值复制到状态寄存器 ST1 的 TC 位。将该指令和 BCND 指令结合可判断指定位的状态，并根据该位的状态来控制程序的转移。

BIT	Oh.15	;(DP=6) 测试0300h 处的最低有效位
BCND	P1.TC	;若该位为1. 则程序转到P1 处执行

6. 数据存储器至数据存储器间的块传送BLDD

把指定的数据存储器源地址中的字复制到指定的数据存储器单元目的地址中。源地址和目的地址可由长立即数地址或数据存储器地址指定。但是，如果源地址为长立即数，则目的地址只能为直接或间接；如果源地址为直接或间接，则目的地址只能为长立即数。该指令不能用于存储器映射的寄存器。使用RPT 指令重复BLDD 操作期间中断被禁止。当BLDD 指令重复使用时，由长立即数指定的源(目的)地址被保存在 PC 中，每次重复过程中PC 加1, 从而可以访问一串源(目的)地址。若使用间接寻址方式来指定目的(源)地址，则在每次重复过程中，一个新目的(源)地址可以被访问。若使用直接寻址方式，所指定的源(目的)地址是个常数，在重复过程中不被修改。

BLDD	#300h.20h	;(DP=6:0300h~037Fh) 将数据存储器单元0300h的内容复制到数据存储器单元0320h
BLDD	干, #321h,AR3	
		;执行前: ARP=2,(AR2)=0301h,(0301h)=0001h,(0321h)=0001h
		;执行后: ARP=3,(AR2)=0302h,(0301h)=0001h,(0321h)=0001h

7. 清除控制位指令CLRC

CLRC 指令将指定的控制位清除为0。指定的控制位为下述控制位之一：

- C 状态寄存器 ST1 的进位位；
- CNF 状态寄存器 ST1 的 RAM 配置控制位；
- INTM 状态寄存器 STO 的中断方式位；
- OVM 状态寄存器 STO 的溢出方式位；

SXM 状态寄存器 ST1 的符号扩展方式位；
 TC 状态寄存器 ST1 的测试/控制标志位；
 XF 状态寄存器 ST1 的 XF 引脚状态位。

CLRC TC ;将 ST1 的 TC 位清0

注：用LST指令也可装入STO和ST1寄存器。

8. 从端口输入数据指令IN

IN 指令从一个I/O 单元读一个16位值到指定的数据存储器单元。 IS 引脚变为低电平，用以指示访问I/O 口。STRB、RD 和 READY 时序与读外部存储器一样。

IN #7.1000h ;(DP=6) 从口地址为1000h 的外设读数据，并将数据存于数据存储器单元0307h 中

IN *,5h ;从口地址为0005h 的外设读数据，并将数据存至当前辅助寄存器所指定的数据存储器单元中

9. 装载累加器的LACC 指令

LACC 指令执行的操作是将指定的数据存储器单元的内容或一个16位常量左移后送入累加器。移位时，低位填0, 高位在SXM=1 时为符号扩展，在SXM=0 时填0。

LACC 5.4 ;(DP=8:0400~047Fh) 将数据存储器单元0405h 的内容左移4位之后送到ACC

LACC *.4 ;(ARP=2,AR2=0305h) 将数据存储器单元0305h 的内容左移4位之后送到ACC

LACC 井1234h.2 ;将长立即数1234h 左移2位之后送到 ACC

10. 装载累加器低位并清零累加器高位指令 LACL

LACL 指令将被寻址数据存储器单元的内容或者被零扩展的8位常量装入累加器的低16位，累加器的高半部分填0。数据被作为无符号的16位数来处理，而非二进制补码。无论SXM 为何状态，该指令的操作数都抑制符号扩展。

LACL #10h ;将0010h 装载入ACC

LACL 1 ;(DP=0~7,AR0~7=0300~0307Fh) 将数据存储器单元030h 的内容装载入ACC

LACL *一, AR4 ;(AR0~7,AR0=0300~0307Fh)=2) 将数据存储器单元0301h 的内容装载入ACC 指令执行完后ARO=0300h,ARP=4

11. 修改辅助寄存器指令MAR 和装载辅助寄存器指令 LAR

MAR 指令用来修改辅助寄存器 ARP 的值，该指令在直接寻址方式下相当于 NOP 指令。LAR 指令用来将数据存储器器的值装载入辅助寄存器。 LAR 和 SAR 指令可在子程序调用或

中断处理时装载和存储辅助寄存器，从而实现在中断或子程序调用时上下文的保存。

MAR	.AR1	;指定当前辅助寄存器为AR1
MAR	+ .AR5	;将当前辅助寄存器(AR1) 增1. 并向ARP 装入5
LAR	AR1.5H	; (DP=4:0200h~027Fh) 将数据存储器地址0205h 的内容装入AR1 寄 ;存器
LAR	AR1. 井 5 0H	;将短立即数0050h 装入AR1寄存器
LAR	AR1.#1234H	;将长立即数1234h 装入ARI 寄存器

12. 装载数据页指针指令LDP

该指令将被寻址数据存储器单元的9位最低有效位或9位立即数转入状态寄存器 ST0 的数据页指针 DP。DP 也可由 LST 指令装入。

LDP	5	;DP=5: 地址0280h~02FFh

13. 装载状态寄存器指令LST

LST 指令将被寻址数据存储器单元中的值装入指定的状态寄存器(ST0 或 ST1)。 注 意 以下几点:

- >“LST #0” 操作向 ARP 装入新值，但并不影响 ST1 寄存器中的 ARB 字段;
 - >"LST #1" 操作中. 送入 ARB 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，与之替代的是将被寻址数据存储器单元所含的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，写这些位不起作用;
- LST 指令用于子程序调用和中断后恢复状态寄存器。

MAR	.ARO	
LAT	井0.*, AR1	;将辅助寄存器 ARO 所寻址的数据存储器单元内容送入状态寄存 ;器 ST0.但不包括INTM 位。尽管指定了下- ARP 值. 但该值被 ;忽略，指定的ARP 也不送入ARB
LST	井1. 0h	; (DP=6:0300h~037Fh) 将数据存储器单元0300h 的内容装入 ST1

14. 装 载TREG 寄存器指令 LTD

LTD 指令将数据寄存单元的内容加载到 TREG。 按 PM 状态位指定的方式，对乘积寄存器的内容进行移位，并把移位后的值与 ACC 相加，结果放在 ACC 中。指定的数据存储器单元的内容复制到地址加1的数据存储单元。数据传送功能可通过连续存储块的边界，但该指令移动数据的功能不能用于外部数据寄存器或存储器映射的寄存器。若 LTD 被用于外部数据寄存器，则功能与LTA 相同。

LTD 123 ;DP=5:0280h~02FFh.PM=0: 乘积不移位

执行前: (02FBh)=0022h,(02FCh)=0000h,(TREG)=0003h,(PREG)=000Fh,(ACC)=0005h

执行后: (02FBh)=0022h,(02FCh)=0022h,(TREG)=0022h,(PREG)=000Fh,(ACC)=0014h

LTD *.AR3 ;(PM=0)

执行前: ARP=1,(AR1)=02FBh,(02FBh)=0022h,(02FCh)=0000h,(TREG)=0003h,(PREG)=000Fh,(ACC)=0005h

执行后: ARP=3,(AR1)=02FBh,(02FBh)=0022h,(02FCh)=0022h,(TREG)=0022h,(PREG)=000Fh,(ACC)=0014h

15. 乘且累加并带数据移动指令 MACD

MACD 指令可以完成以下功能:

►按 PM 状态位指定的方式把先前的乘积移位, 再与ACC 的内容相加;
把指定的数据存储器单元的内容加载到 TREG;

>将存放在 TREG 寄存器中的数据存储器单元值乘以指定的程序存储器地址中的内容;

>将指定的数据存储器地址中的内容复制到下一个数据存储器。

当重复执行MACD 指令时, 每重复一次包含在 PC 中的程序存储器地址加1。若使用间接寻址指定数据存储器地址, 则每次重复时就可以访问新的数据存储器地址; 若使用直接寻址方式, 指定的数据存储器地址是常数, 重复时不会对其进行修改。若 MACD 寻址存储器映像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, 数据移动不会发生。

MACD FFO0h.0008h ;DP=6:0300h~037Fh;PM=0;CNF=1:BO 配置为程序存储器

执行前: 数据存储器 (0308h)=0023h,(0309h)=0018h, 程序存储器(FF00h)=0004h,
(TREG)=0045h,(PREG)=00458972h,(ACC)=0723EC41h

执行后: 数据存储器 (0308h)=0023h,(0309h)=0023h, 程序存储器(FF00h)=0004h,
(TREG)=0023h,(PREG)=008Ch,(ACC)=076975B3h

16. 乘指令 M PY

T 寄存器内容和被寻址数据存储器单元的内容相乘, 其结果转入P 寄存器中。若使用立即数寻址, 则T 寄存器和带符号的13位常数相乘。无论 SXM 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。

MPY 5 ;(DP=4:0200h~027Fh) 将数据存储器单元0205h 的内容和 TREG
;寄存器中的内容相乘, 结果保存在PREG 中

MPY AR2 ;(ARP=1.AR1=040Dh) 将数据存储器单元040Dh 的内容和TREG
 ;中的内容相乘. 结果保留在 PREG 中. 指令执行完后ARP=2
MPY #031h ;立即数0031h 和 TREG 中的内容相乘, 结果保存在PREG 中

17. 重复执行下一条指令 RPT

若使用直接或间接寻址, 则被寻址的数据存储器单元中的值送入重复计数器(RPTC); 若使用短立即数寻址, 则8位立即数送入 RPTC。紧接 RPT 后的那条指令被执行 n 次, n 为 RPTC 初值加1。由于在上下文切换时不能保存RPTC 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 清 0。

RPT #20 ;执行NOP 指令21次
NOP

18. 移位并存储累加器高位指令 SACH

SACH 指令将整个累加器复制到输出移位寄存器中, 然后全部32位数左移0~7位, 再将移位后数值的高16位复制到数据存储器。在移位时, 低位补0, 高位丢失, 累加器内容不变。

SACH 10h.1 ;(DP=4:0200h~027Fh) 将 ACC 的左移一位. 高16位存至数据
 ;存储器单元020Ah 中
SACH * + .AR2 ;(ARP=1) 将 ACC 的高16位存至AR1 指向的数据存储器单元,
 ;操作完成之后ARP=2

19. 移位并存储累加器低位指令 SACL

SACL 指令将整个累加器复制到输出移位寄存器中, 然后全部32位数左移0~7位, 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0, 高位丢失, 累加器内容不变。

SACL 1Dh.1 ;(DP=4:0200h~027Fh) 将 ACC的左移一位 • 低16位存至数据
 ;存储器单元020Ah 中
SACH * + .AR2 ;(ARP=1) 将 ACC 的高16位存至 ARI 指向的数据存储器单元.
 ;操作完成之后ARP=2

20. 存储辅助寄存器指令 SAR

SAR 指令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, 当SAR 指令同时也要对当前辅助寄存器内容进行修改时, SAR 将在增、减辅助寄存器内容前将辅助寄存器值存至数据存储器。

SAR AR0,30h ;(DP=6:0300h~037Fh) 将 ARO的值存至数据存储器单元0330h 中
SAR ARO, 关 干 ;将 ARO的值存入辅助寄存器AR0 指向的数据存储器单元. 同时 ARO
 ;的值增 I

执行前: ARP=0,ARo=0400h,(0400h)=0000h

执行后: ARP=0,ARo=0401h,(0400h)=0400h

21. 从当前辅助寄存器中减去立即数指令 SBRK

该 SBRK 指令从指定的辅助寄存器中减去8位立即数值, 其结果替换原有的辅助寄存器中的内容。减法在辅助寄存器算术单元(ARAU) 中进行, 立即数值被作为8位正数处理。所有辅助寄存器的算术运算都是无符号的。

SBRK #20h

执行前: ARP=5,AR5=0050h; 执行后: ARP=5,AR5=0030h

22. 设置控制位指令 SETC

SETC 指令设置指定的控制位为1。 LST 指令也可用于装载 STO 和 ST1 寄存器。指定的控制位为下述控制位之一:

C 状态寄存器 ST1 的进位位;
CNF 状态寄存器 ST1 的 RAM 配置控制位;
INTM 状态寄存器 STO 的中断方式位;
OVM 状态寄存器 STO 的溢出方式位;
SXM 状态寄存器 ST1 的符号扩展方式位;
TC 状态寄存器 ST1 的测试/控制标志位;
XF 状态寄存器 ST1 的 XF 引脚状态位。

SETC TC ;将ST1的 TC 位置1

23. 存储长立即数至数据存储器指令 SPLK

SPLK 指令将一个16位值写入任何一个数据存储器单元。在直接寻址方式下使用该指令对数据存储器单元赋值时, 通常需要将数据页指针 DP 指向该数据存储器单元所在的数据页。

SPLK #30h,5 ;(DP=4) 将数据0030h 存至数据存储器单元0205h 处

SPLK #1122h,* 十, AR4

执行前: ARP=0,AR0=0400h,(0400h)=0000h

执行后: ARP=4,ARO=0401h,(0400h)=1122h

24. 存储状态寄存器指令 SST

SST 指令将指定的状态寄存器 ST0 和 ST1 存入数据存储器。在直接寻址方式下, 不管 STO 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。虽然处理器将自动访问第0页, 但 DP 指针不会被修改。在存储 STO 和 ST1 时, DP 不被修改也存储。第0页内的特定存储单元在指令中给出, 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意页内的地址中。

SST #0.60h ;自动访问数据页0. 将STO 的值存入数据存储器单元0060h 中

25. 表读指令TBLR

TBLR 指令将程序存储单元中的一个字传送到指令指定的数据存储器单元中。程序存储单元中的地址由ACC 的低16位指定。该指令先从程序存储单元读出，然后写入指定的数据存储器单元。当和重复指令(RPT) 一同重复使用时，TBLR 成为单周期指令，并且用ACC(15:0) 装载后的程序计数器每个周期增1。

TBLR 6h ;(DP=4)

执行前: (ACC)=0023h, 程序存储器 (0023h)=0306h, 数据存储器 (0206h)=0075h
执行后: (ACC)=0023h, 程序存储器 (0023h)=0306h, 数据存储器 (0206h)=0306h

TBLR ,AR7

执行前: ARP=0.(AR0)=0300h,(ACC)=0024h. 程序存储器 (0024h)=0307h, 数据存储器 (0300h)=0075h
执行后: ARP=7,(ARO)=0300h,(ACC)=0024h, 程序存储器 (0024h)=0307h, 数据存储器 (0300h)=0307h

由于篇幅有限，本节仅以几个比较典型的指令为例，介绍了编程过程中指令的使用方法。需要注意的是，直接寻址时 DP 指针一定要定位到要访问的数据存储区，间接寻址时当前 ARP 必须设置正确. 否则不能正确访问单元。关于 TMS320C24X 指令集每条指令的详细介绍请参见“TMS320C24x DSP Controllers Reference Guide CPU and Instruction Set”。



第 4 章 CCS3.1 集成调试环境安装

4.1 CCS3.1 软件安装

将安装光盘插入 CD-ROM 驱动器中，执行安装程序。下面将详细叙述在 Windows XP 操作系统之下，安装 CCS3.1 版本的具体步骤。

(1) 找到名为 CCS3.1 SETUP.EXE 应用程序的图标。双击该图标(若不声明，单击或双击均指操作鼠标左键)，出现如图 4.1 所示对话框。



图 4.1 CCS3.1 安装对话框

(2) 单击 Next 按钮，出现如图 4.2 所示对话框。

(3) 再单击 Next 按钮，出现如图 4.3 所示对话框。

(4) 选择 I accept the License Agreement 后，再单击 Next 按钮，出现如图 4.4 所示对话框。默认典型安装，选择 Typical install; 用户自定义安装，选择 Custom install。

(5) 此处选择默认典型安装，即选择 Typical install 图标，然后单击 Next 按钮出现如

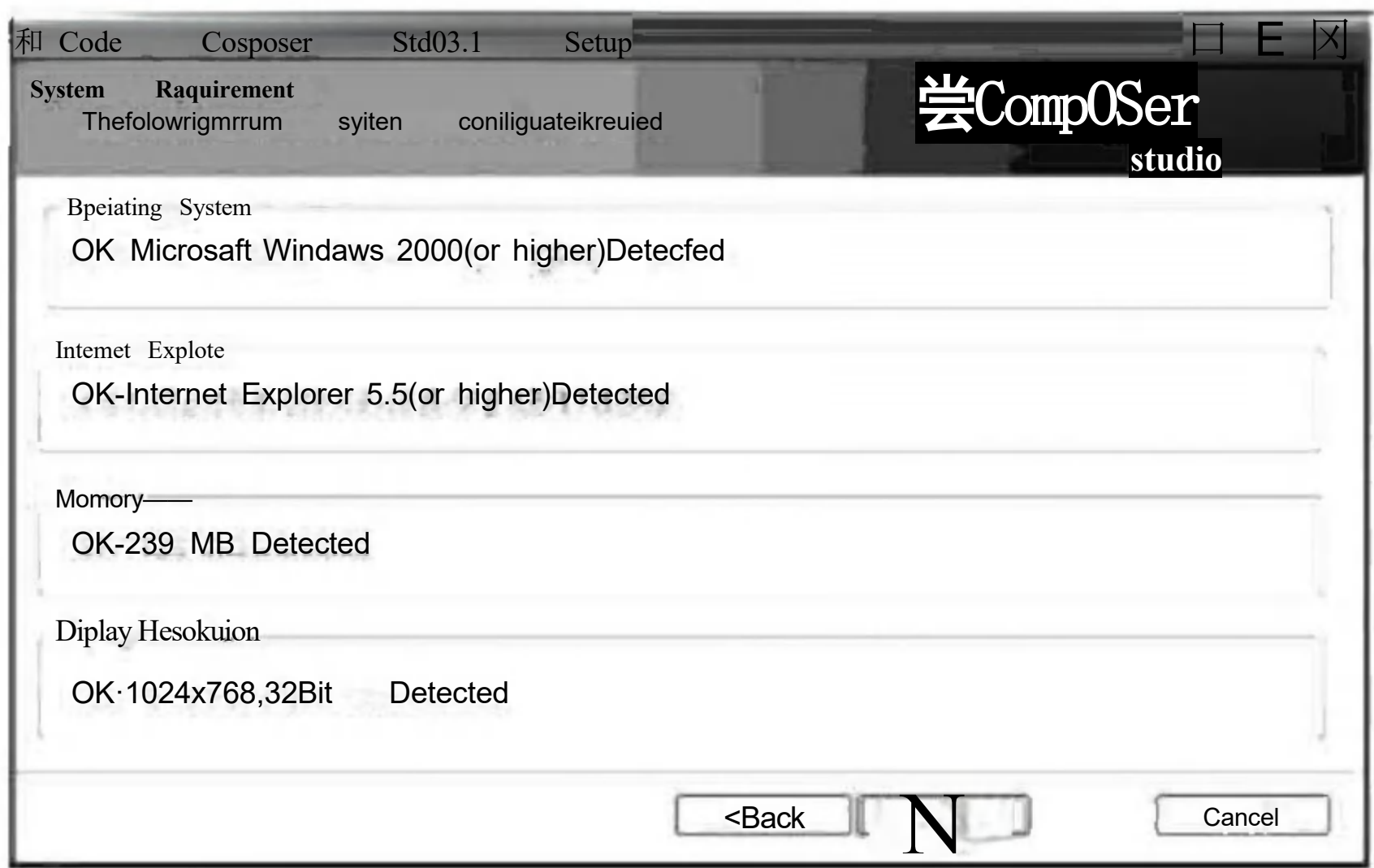


图4.2 CCS3.1 安装系统需求对话框

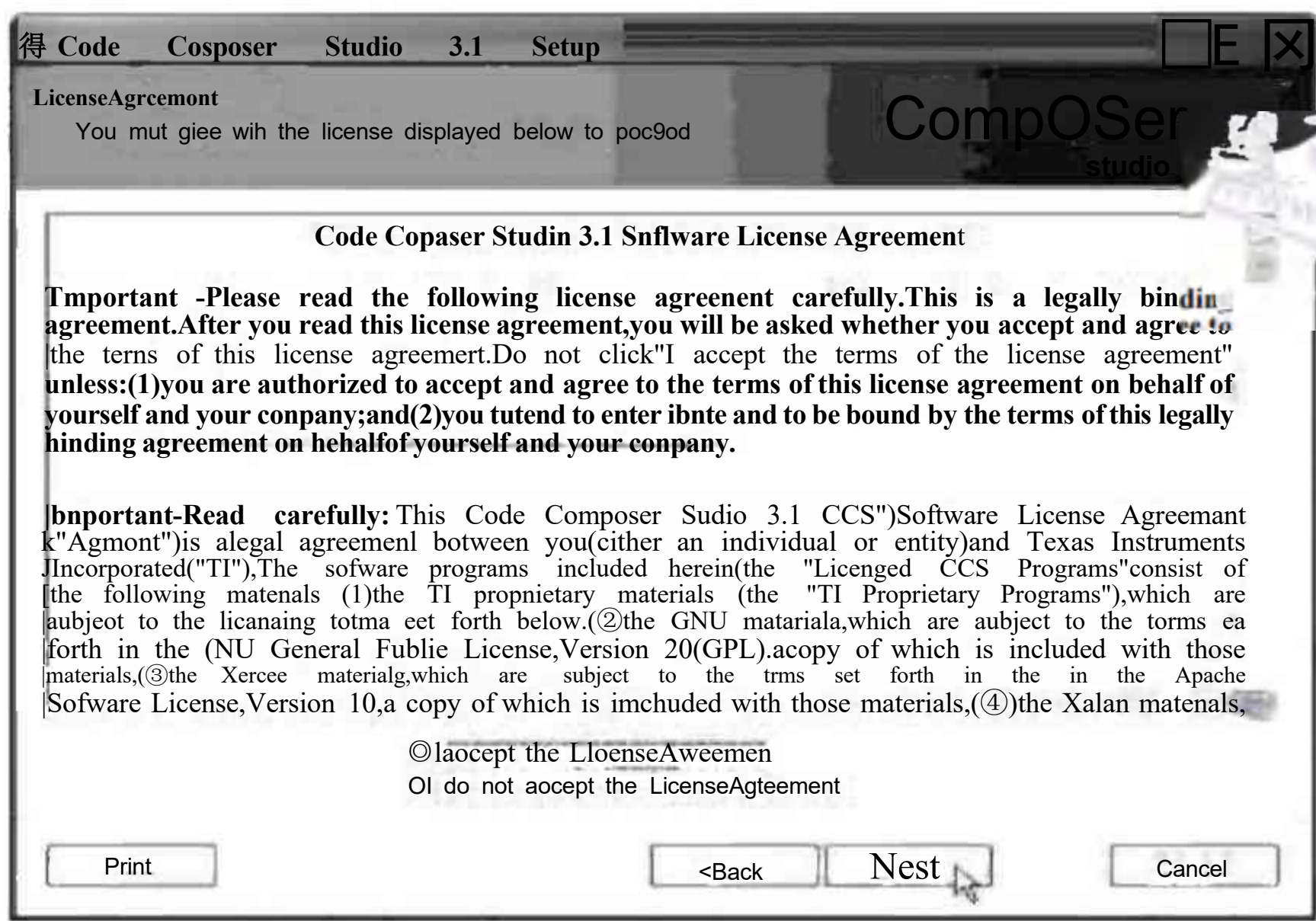


图4.3 CCS3.1 许可协议选择对话框

图4.5所示对话框。其中C:\CCStudio_y3.1 为默认安装路径。若要改变软件安装路径，则可单击Browse 按钮，在此确定软件安装路径。这里选择默认安装路径。

- (6)单击 Next 按钮，出现如图4.6所示对话框。
- (7)单击Install Now 按钮，出现如图4.7所示对话框。
- (8)安装完成，出现如图4.8所示对话框，单击 Finish 按钮。

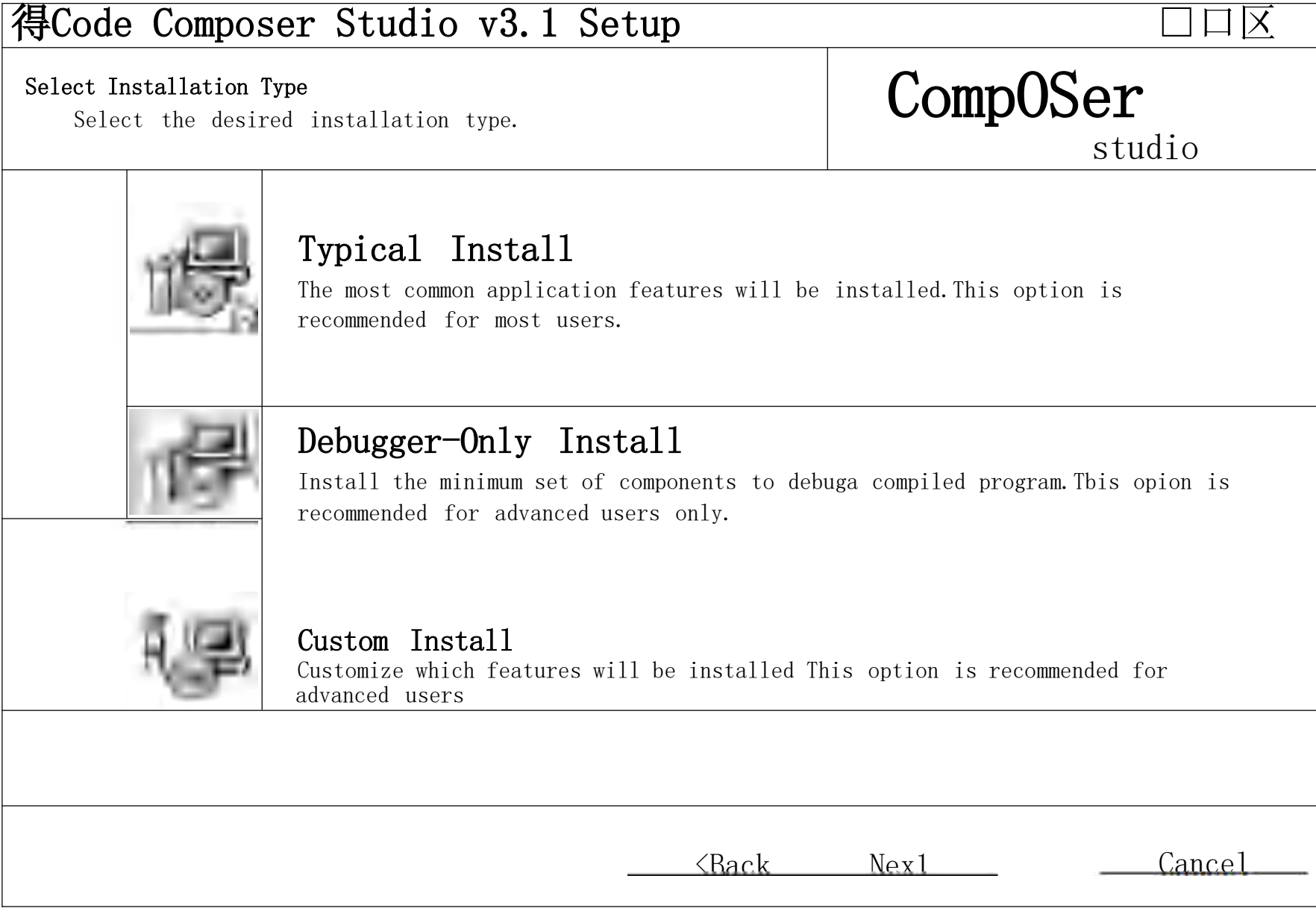


图4.4 CCS3.1 安装类型选择对话框

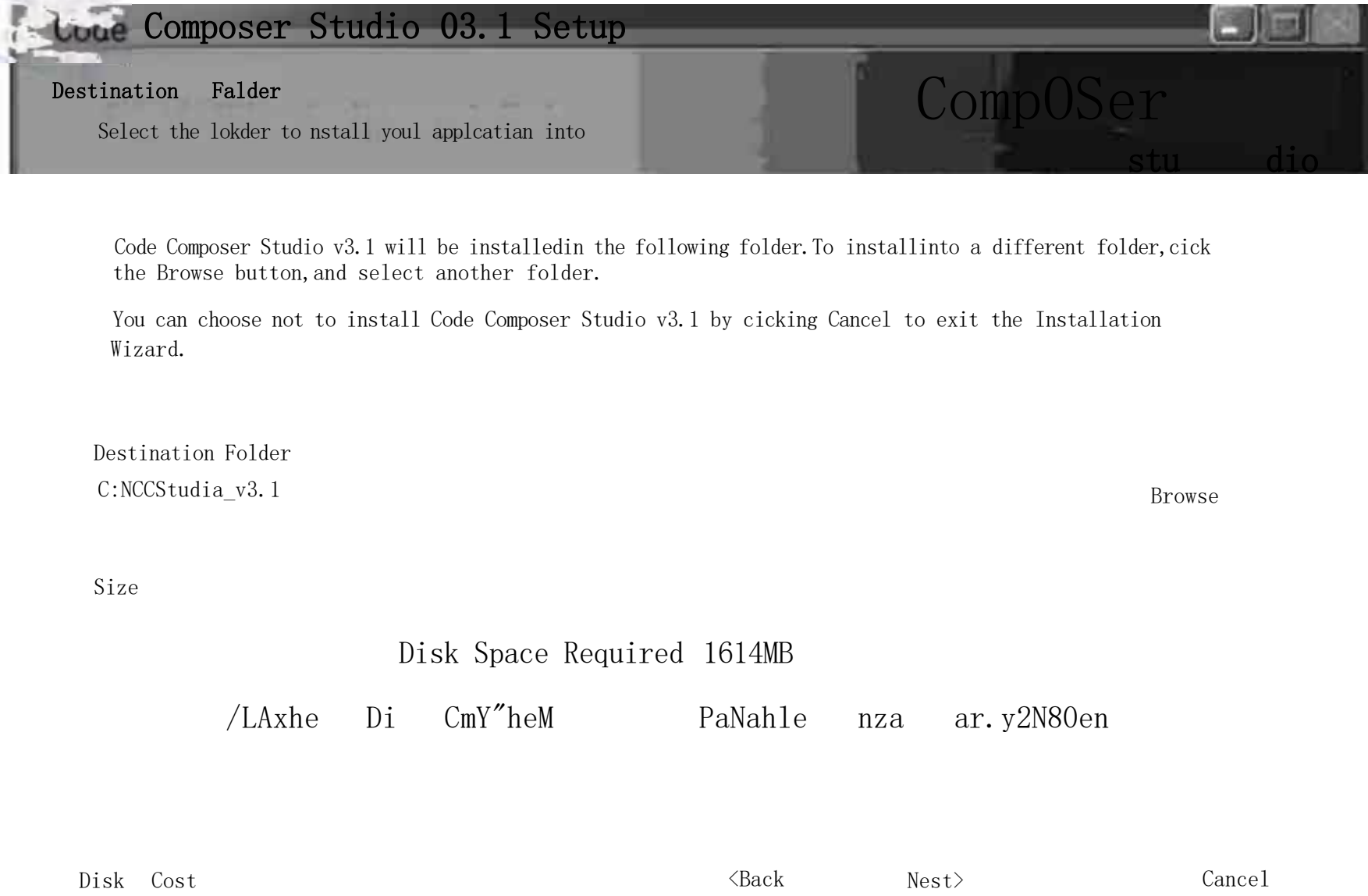


图4.5 CCS3.1 安装路径选择对话框

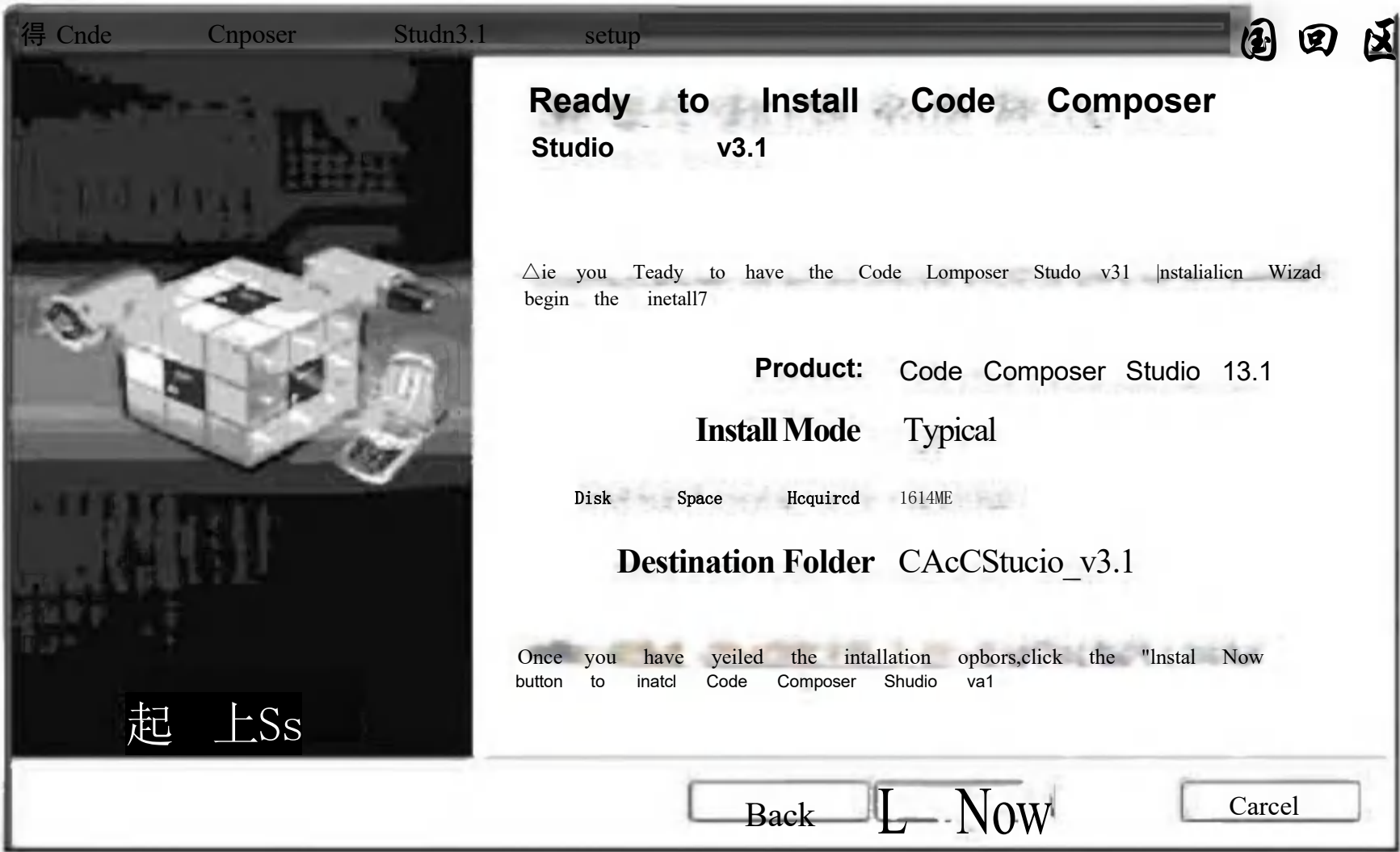


图4.6 准备安装CCS3.1 对话框

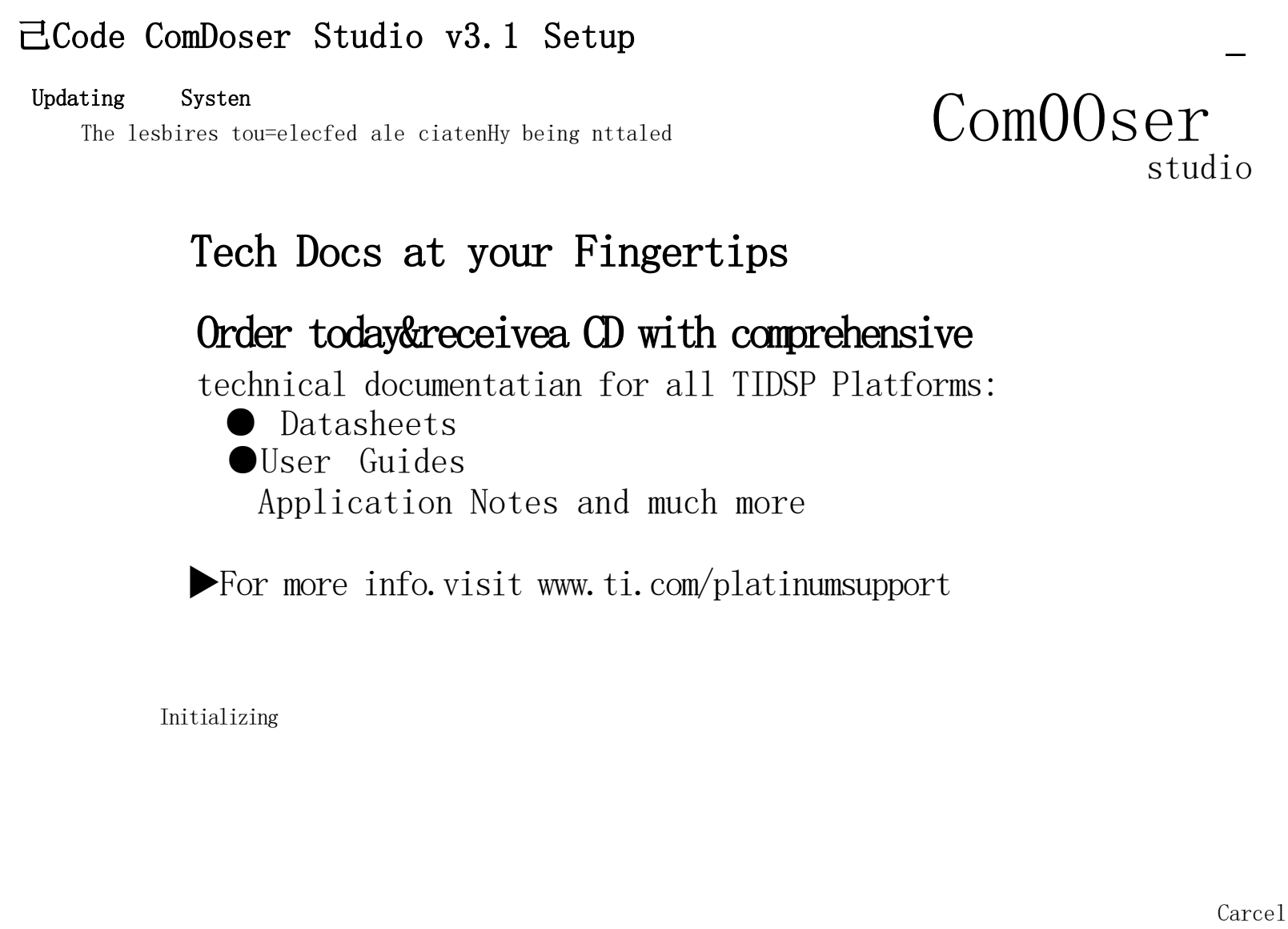


图4.7 CCS3.1 正在安装图

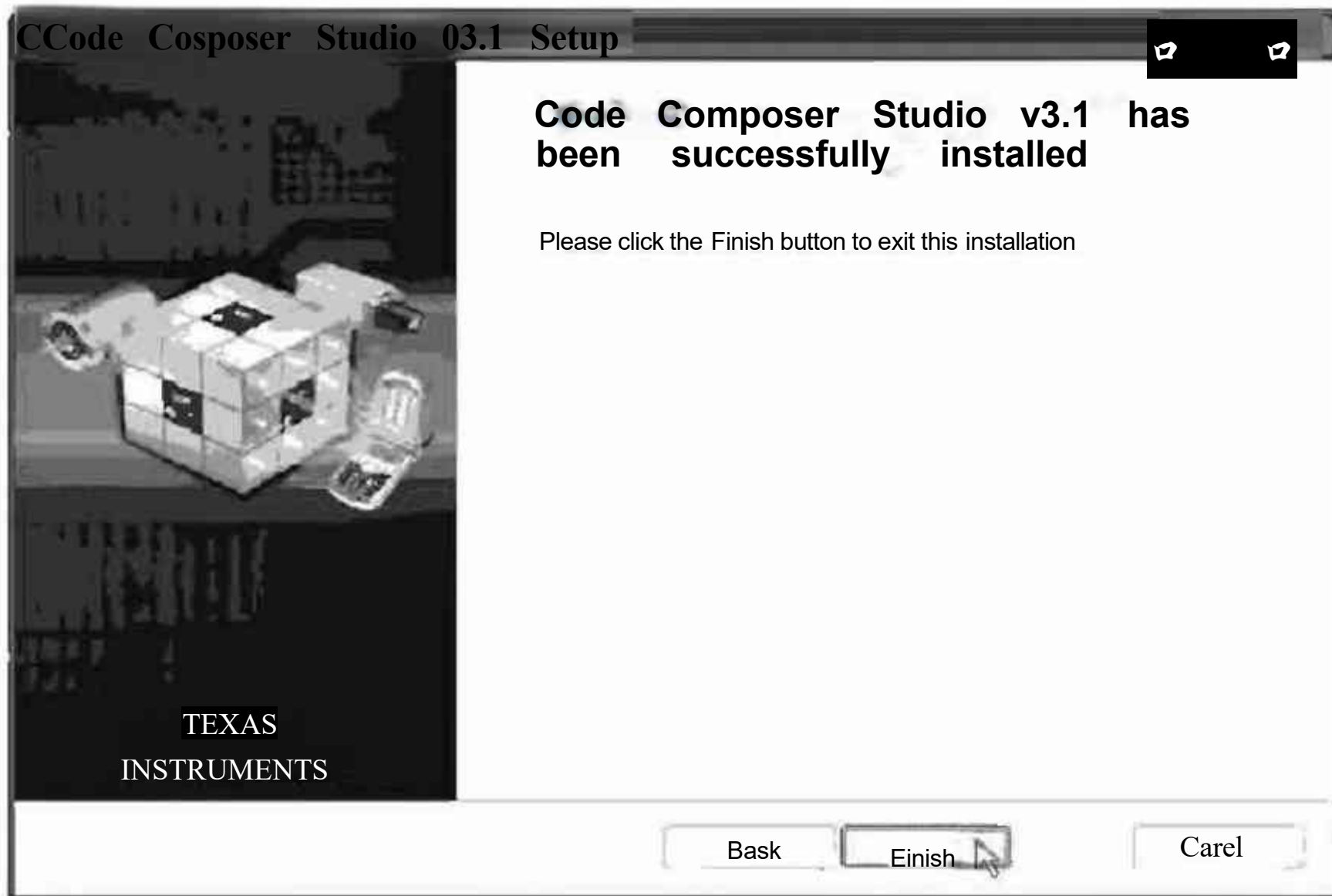


图4.8 CCS3.1 安装完成对话框

4.2 USB 接口仿真器驱动程序安装

在安装完 CCS 集成调试环境之后，按以下步骤安装USB 接口仿真器驱动程序。

(1) 找到USB 接口仿真器驱动程序SEEDUSB2.0，双击如图4.9所示图标，出现图4.10所示对话框。



图4.9 选择 USB 驱动程序图标

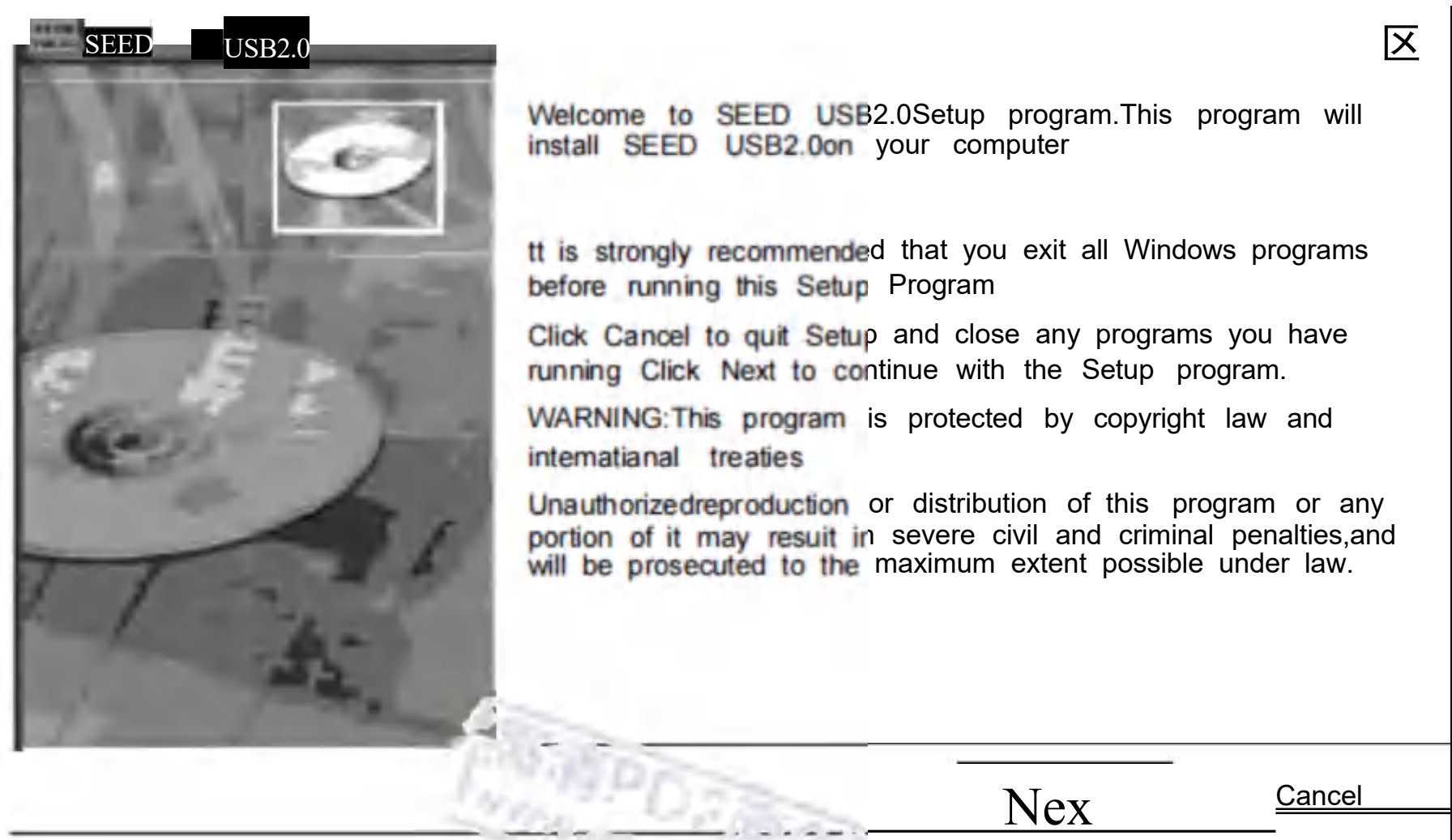


图4. 10 USB 驱动程序安装对话框

(2) 单击Next 按钮，出现如图4. 11所示对话框。

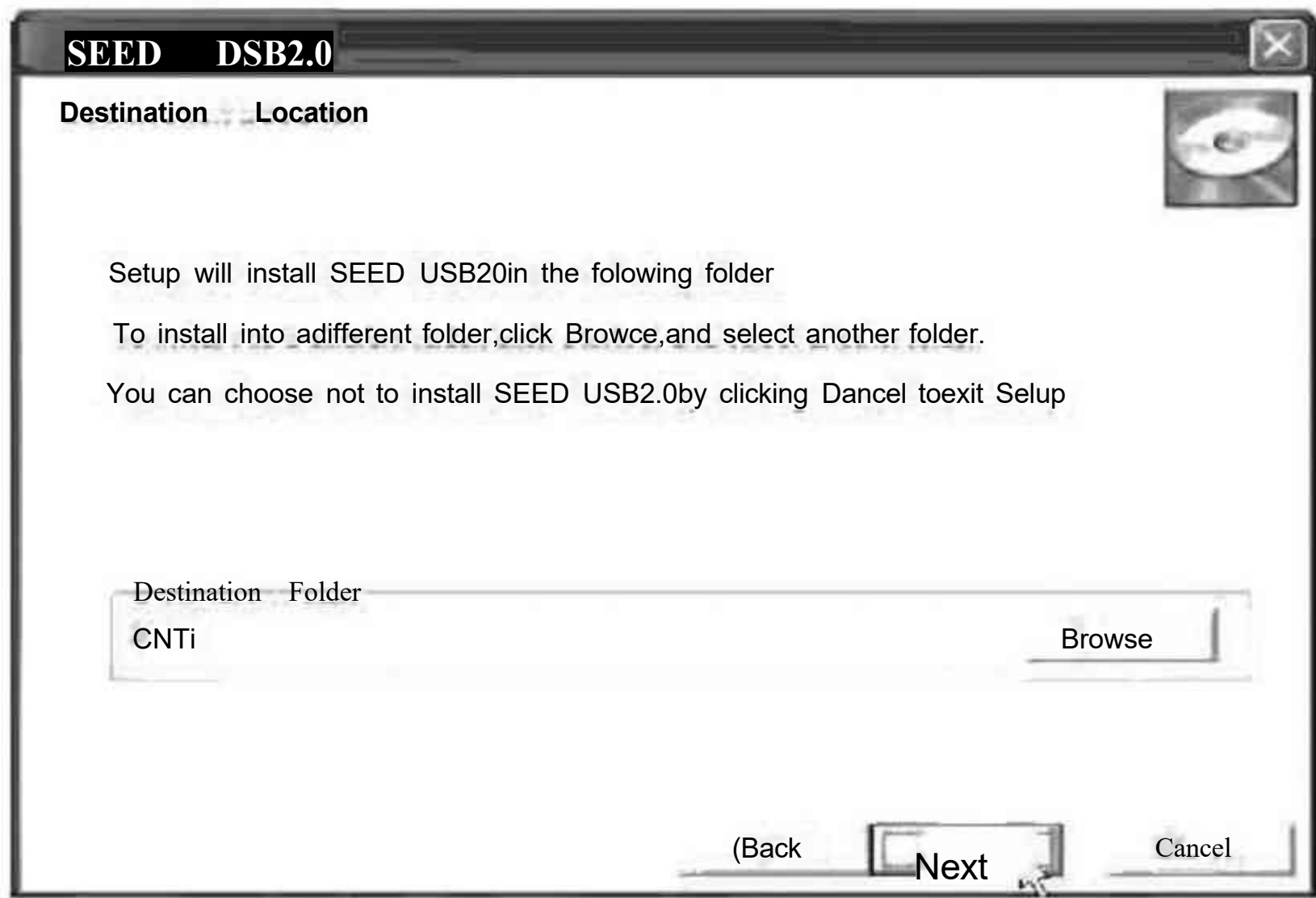


图4. 11 安装路径选择对话框

(3) 单击Browse 按钮，可以修改安装路径，出现如图4. 12所示对话框。

(4) 选择 CCStudi_y3.1, 单击OK 按钮，弹出一个提示框，如图4. 13所示。

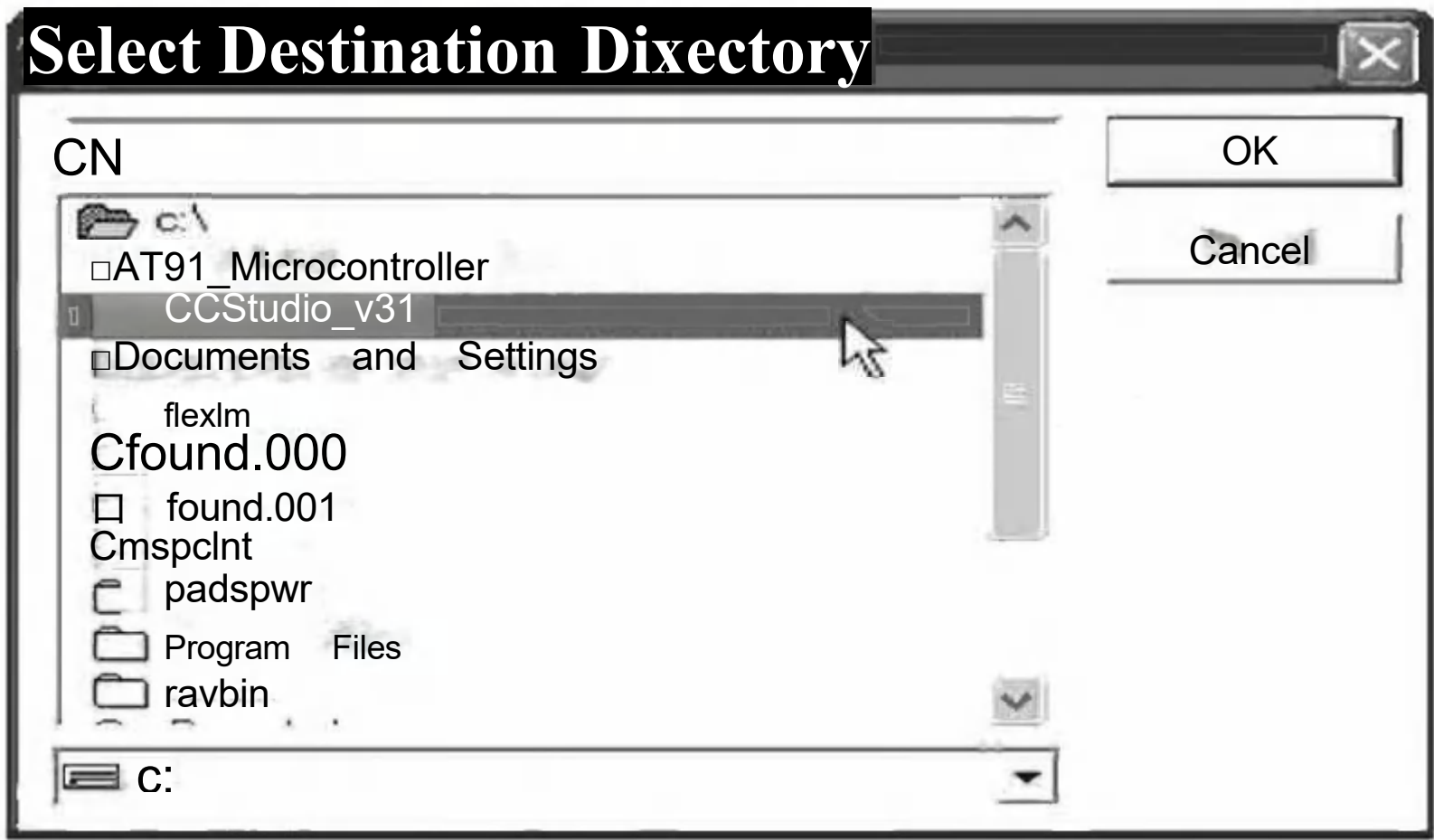


图4.12 选择安装路径

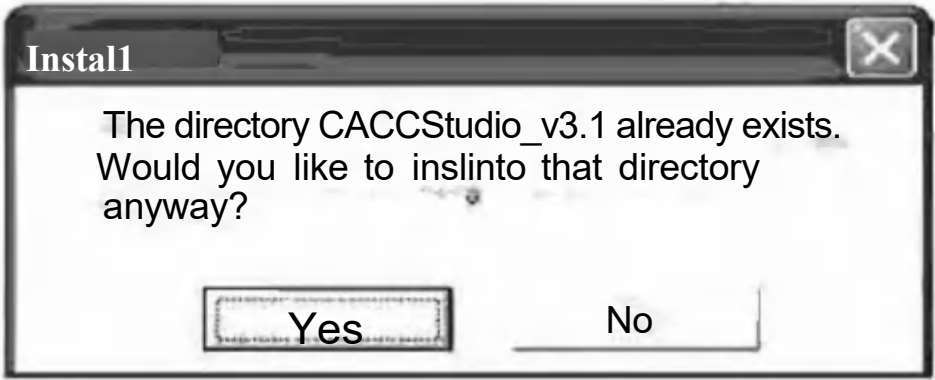


图4.13 确认安装路径

(5) 单击Yes 按钮，出现如图4.14所示对话框。

SEED USB2.0

Destination Location



Setup will install SEED USB2.0 in the following folder.
To install into a different folder,click Browse,and select another folder.
You can choose not to install SEED USB2.0 by clicking Cancel to exit Setup.

Destination Folder	
C:\CCStudio_v3.1	Browse...

<Back Next> Cancel

图4.14 安装路径提示

(6) 单击 Next 按钮，出现如图4. 15所示对话框。

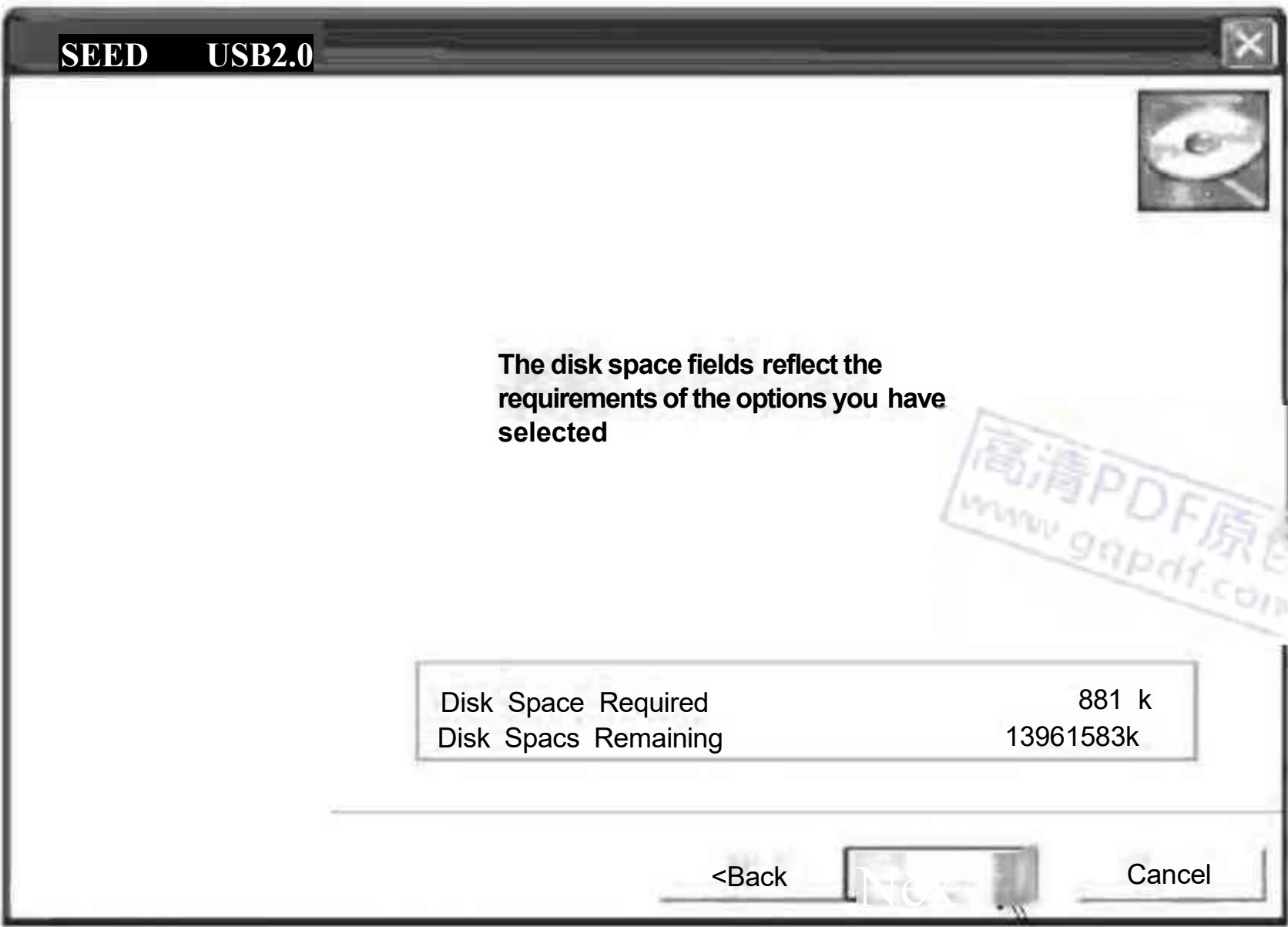


图4 . 15 安装所需空间提示

(7) 单击 Next 按钮，出现如图4. 16所示对话框。

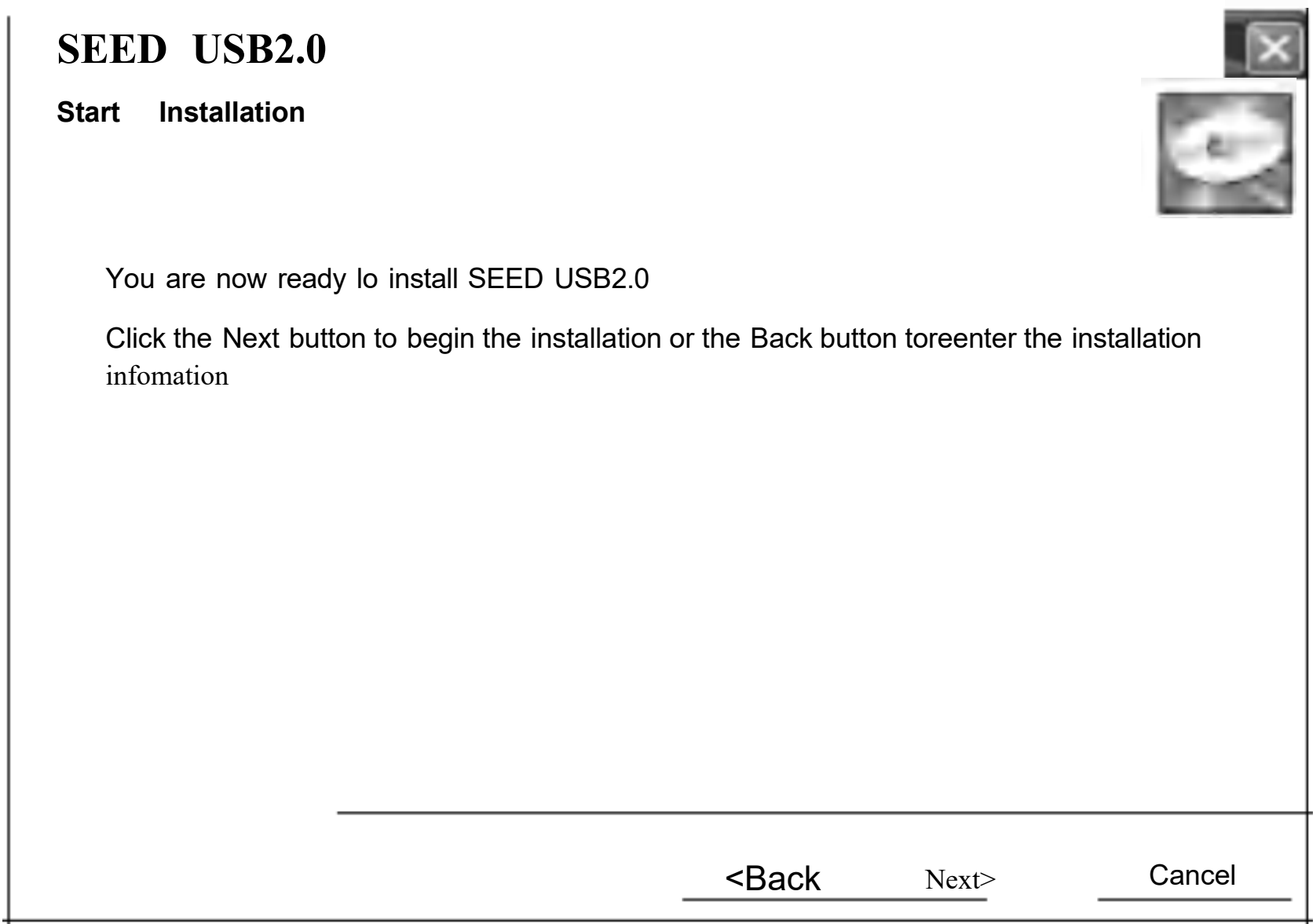


图4 . 16 准备安装对话框

(8) 单击Next 按钮，出现如图4.17所示对话框。



图4.17 安装完成对话框

(9) 单击Finish 按钮，完成安装。

(10) 安装硬件仿真器与目标系统。上电正常，仿真器上的指示灯亮(注：也可不上电，同样能识别USB 设备)。硬件连接完毕后，出现如图4.18所示对话框。选择第3个选项后，单击“下一步”按钮，出现如图4.19所示对话框。

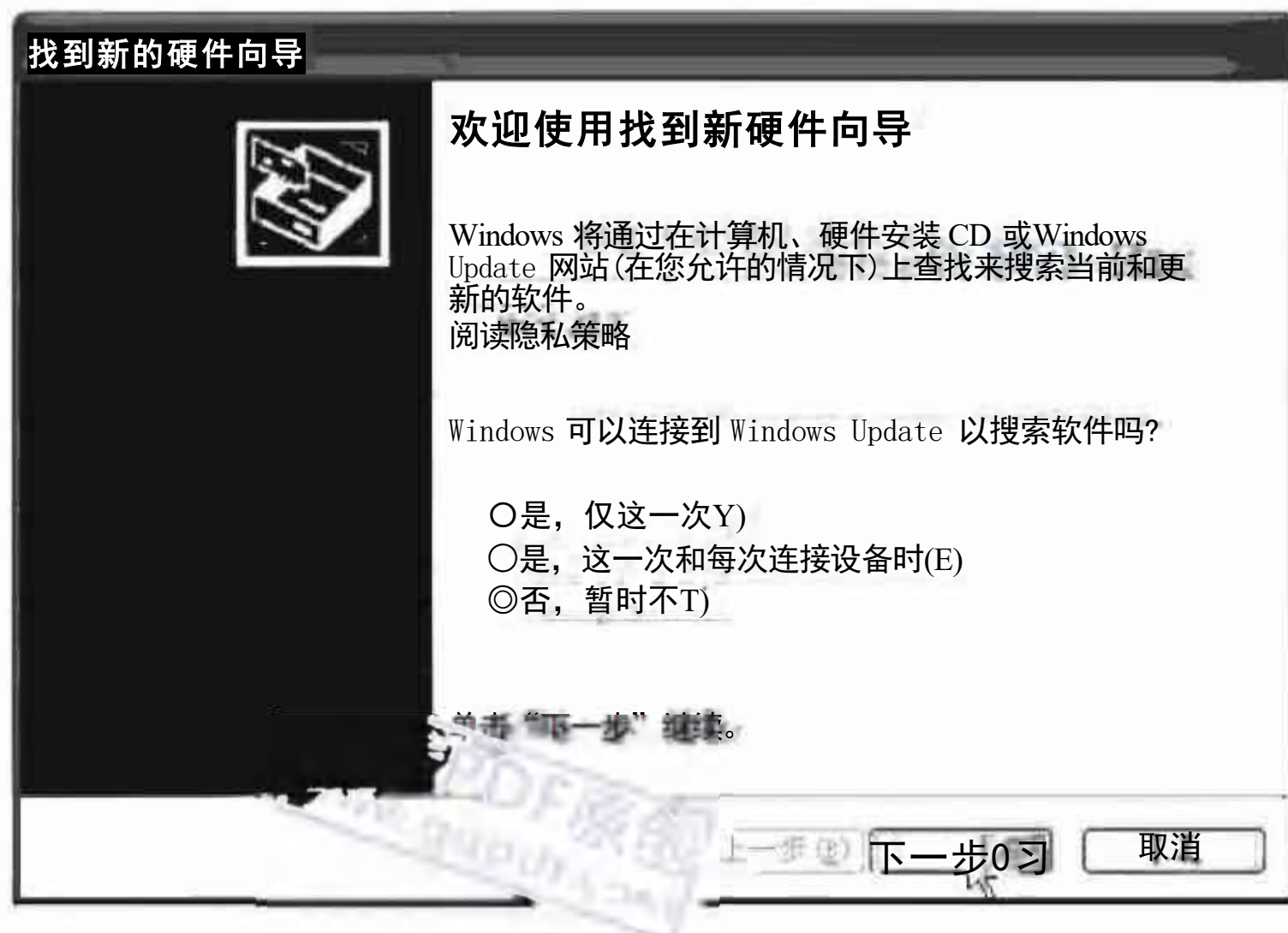


图4.18 “找到新的硬件向导”对话框



图4.19 选择安装方式对话框

(11) 选择第2项后，单击“下一步”按钮，弹出如图4.20所示对话框。

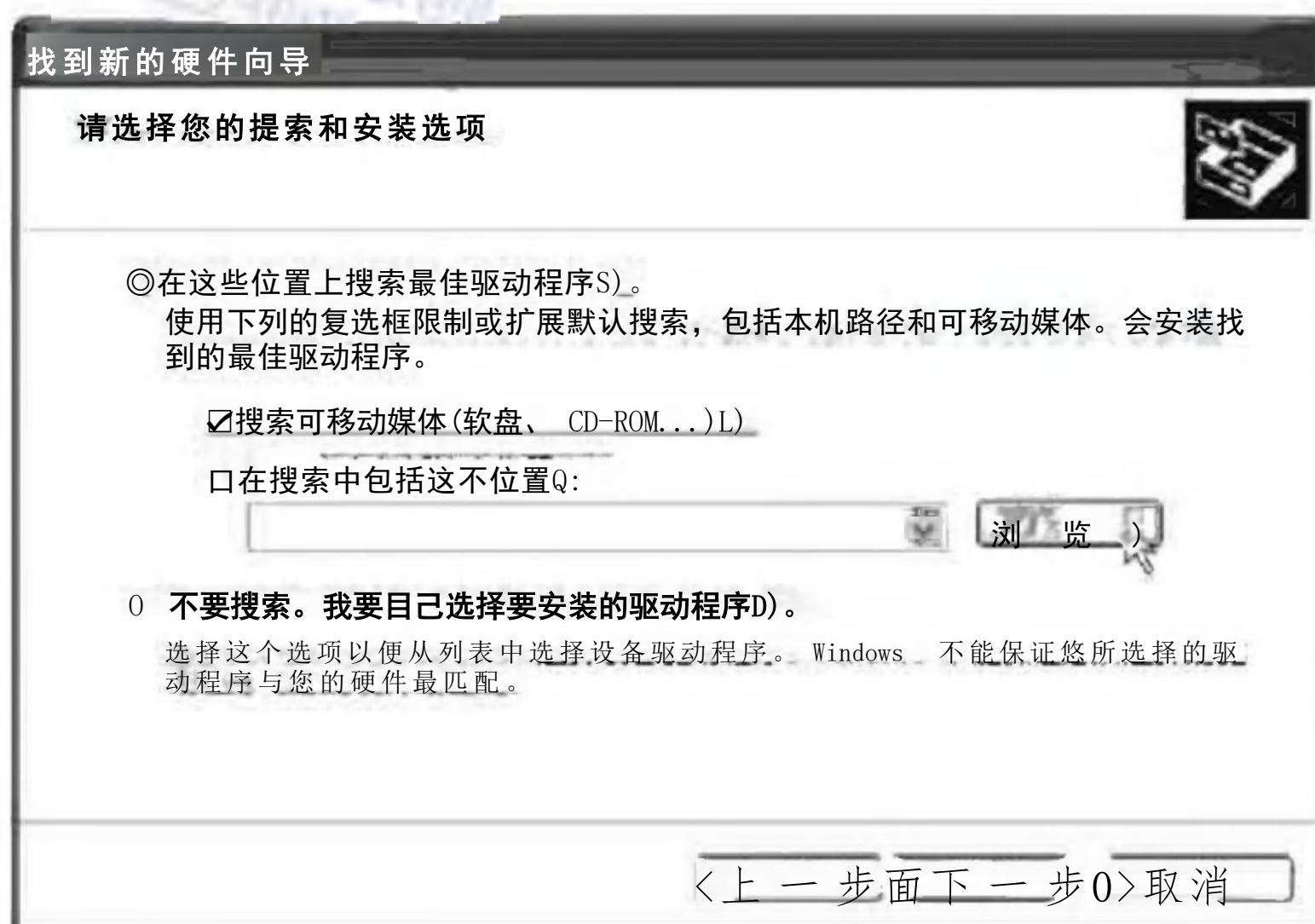


图4.20 选择搜索和安装选项对话框

(12) 选择“在搜索中包括这个位置”选项，单击“浏览”按钮，选择路径。然后单击“下一步”按钮，弹出如图4.21所示对话框。

(13) 单击“仍然继续”按钮，出现如图4.22所示对话框，开始安装软件。

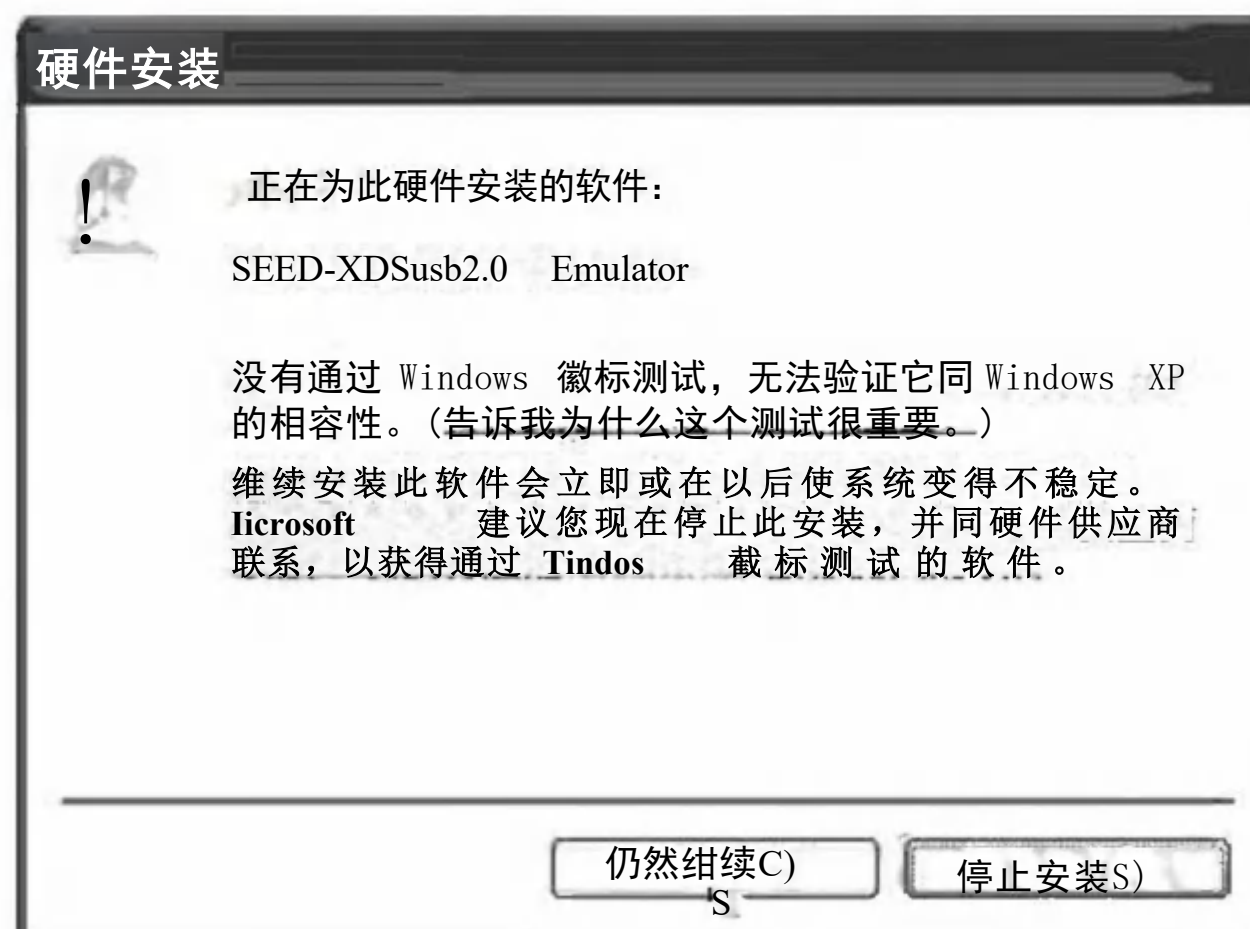


图4.21 继续安装确认对话框



图4.22 正在安装驱动程序

(14) 安装完成后，出现如图4.23所示对话框。单击“完成”按钮，重新启动计算机。

(15) 安装完毕后，打开控制面板查看系统中的设备管理器，出现如图4.24所示对话框。当出现SEED-XDSusb2.0 Emulator时，说明硬件连接已经成功；否则须检查硬件是否连接正确以及USB驱动程序安装是否正确。

(16) 找到\SEED-XDSUSB2.0\Windows XP\USBdriver\子目录下的usb20emurst.exe文件，并执行。如果出现图4.25中的下面一个提示，那么说明仿真器与目标系统连接正常。



图4.23 硬件驱动程序安装完成对话框



图4.24 设备管理器对话框

(此步只是检查 USB 接口的仿真器与目标系统连接是否正常，并未安装CCS 集成开发环境与仿真器之间的驱动程序。)

至此，CCS3.1 调试环境和 USB 接口仿真器驱动程序全部安装完成。

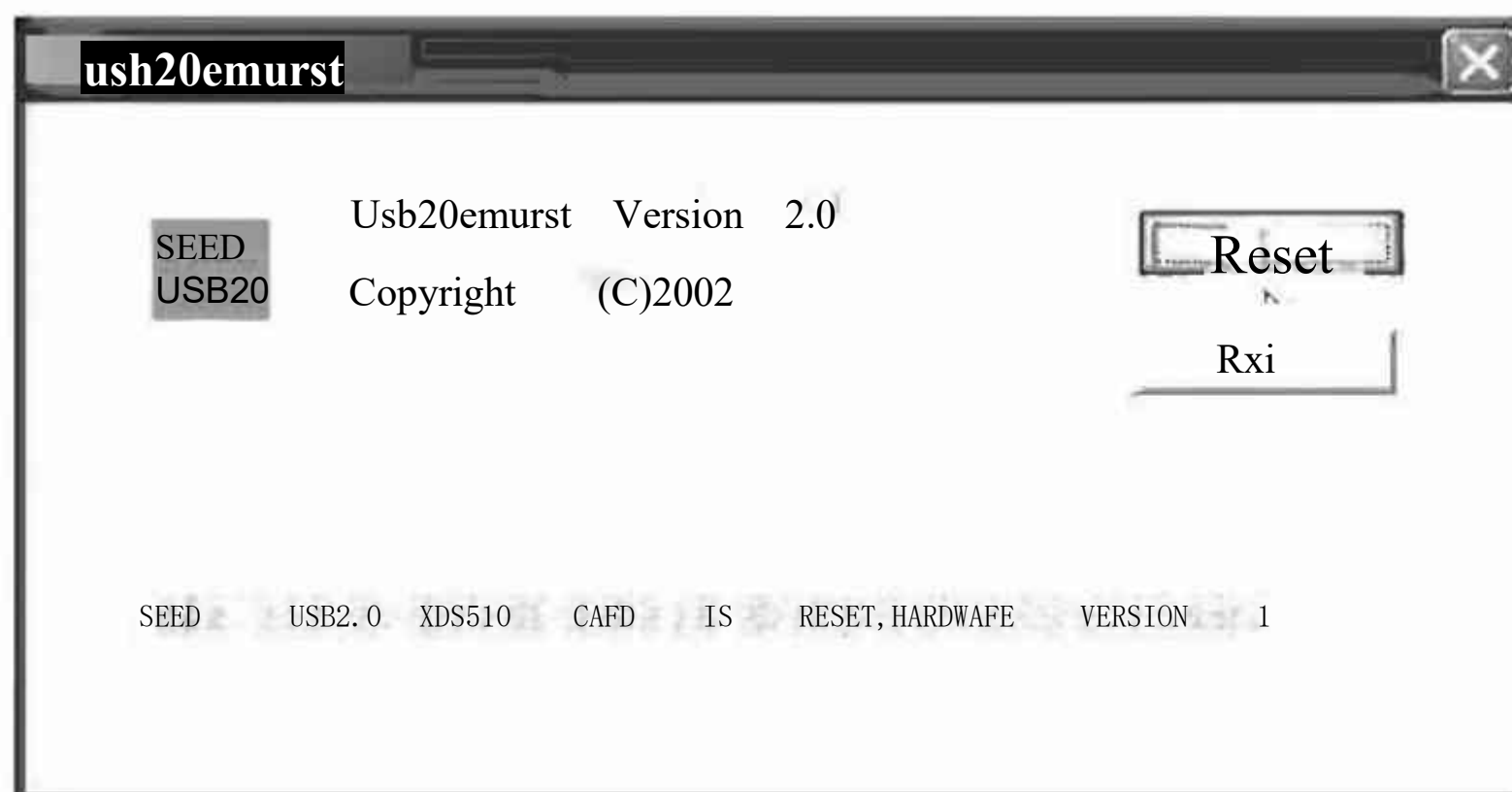


图4.25 USB 仿真器复位对话框

4.3 配置 CCS3.1 的运行环境

在PC 机系统上安装好编译器软件后，在计算机桌面上将出现两个快捷方式图标：一个是 Setup CCStudio v3.1,另一个是 CCStudio 3.1。Setup CCStudio v3.1用来对该编译器的运行环境进行配置；CCStudio 3.1为程序仿真调试集成环境软件。

编译器设置定义了使用编译器的目标板或仿真器，称为“系统配置”。它决定了编译器选用什么工具，因此在编译一个应用程序之前必须进行系统配置。编译器分为系统配置、已安装的仿真器类型及设置命令3大部分。

配置仿真器的在线帮助可通过 Code Composer Setup的帮助菜单获得。

如果想修改系统配置，那么须在保证仿真器与实验开发板之间连线正确后，给实验开发板供电，然后按以下步骤操作。

(1) 双击Setup CCStudio v3.1图标，打开设置界面，如图4.26所示。在Available Factory Boards 栏中显示选择的系统配置。

(2) 在 Available Factory Boards 栏中选择合适的系统配置(此处选择 F2407 XDS510 Emulator)，然后双击鼠标左键，出现如图4.27所示对话框。

(3) 在 System Configuration 栏中选中 F2407 XDS510 Emulator,右击鼠标出现如图4.28所示对话框。选择 Properties 选项，出现如图4.29所示对话框。

(4) 打开图4.29中部下拉列表框，选择Auto-generate board data file with extra configuration，如图4.30所示。

(5) 单击 Browse 按钮，选中\CCStudio_v3.1 目录 drivers 文件夹中的 Seedusb2.cfg 文件，如图4.31所示。单击“打开”按钮，出现如图4.32所示对话框。

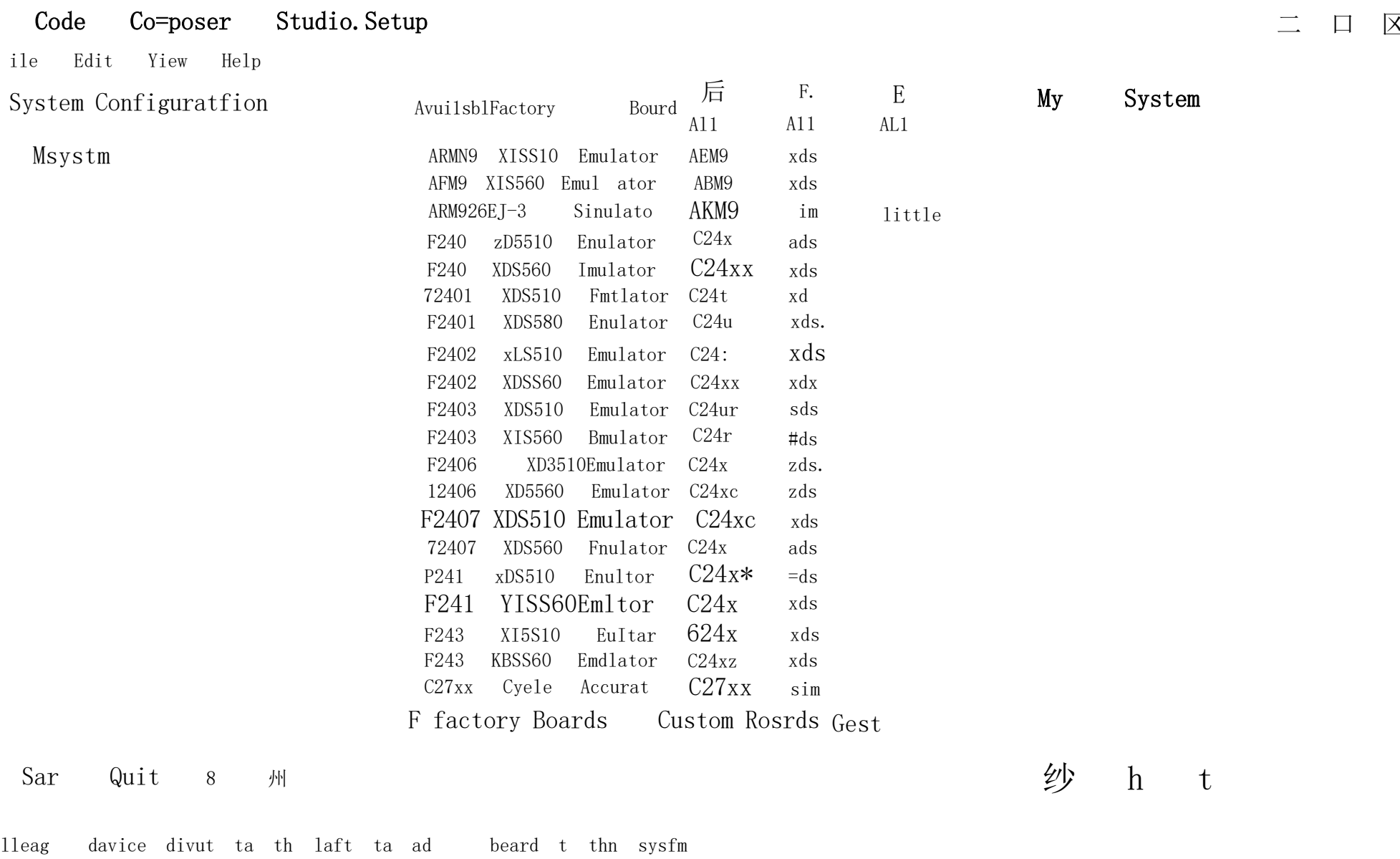


图4. 26 Setup CCStudio y3.1 界面

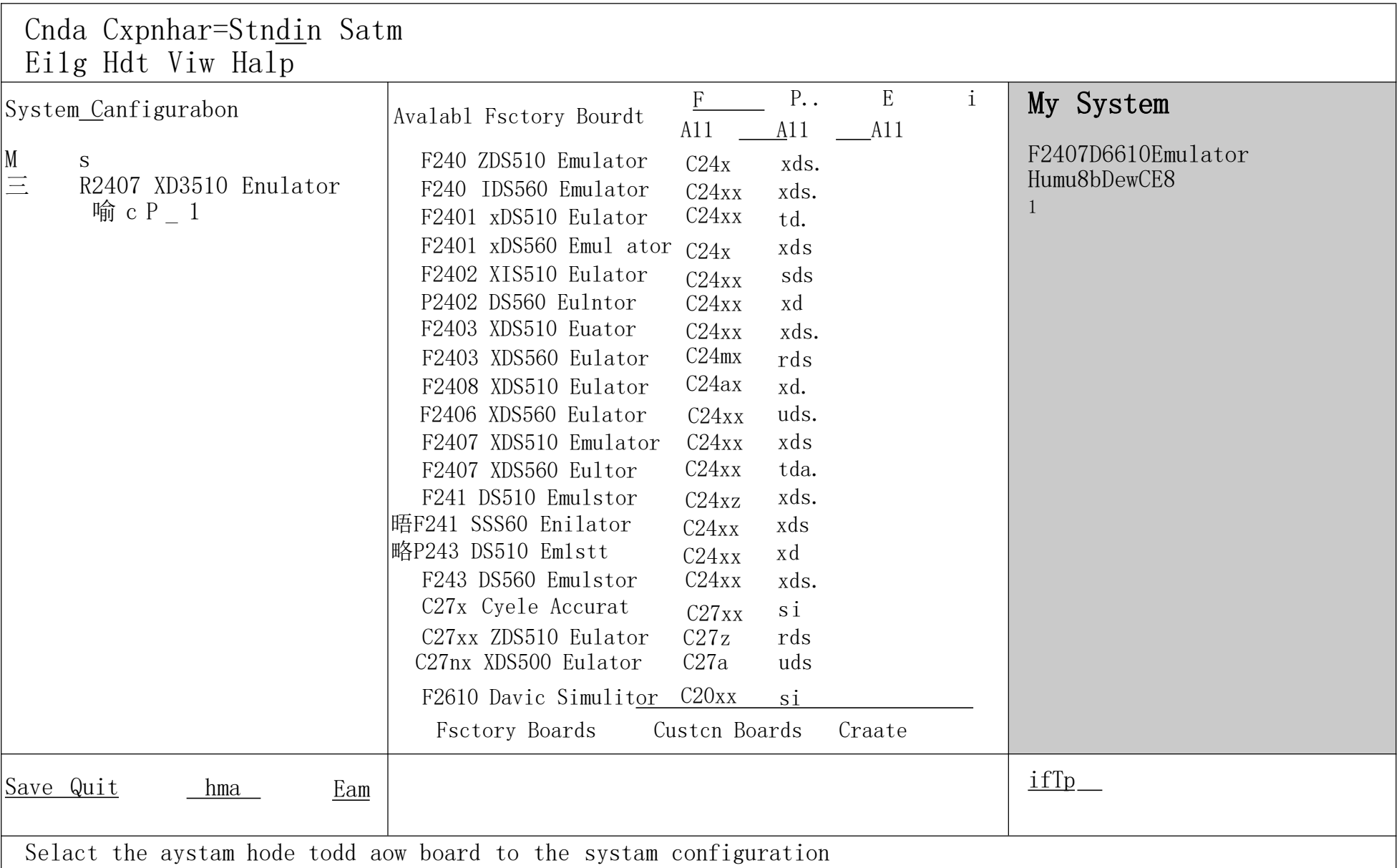


图4. 27 选中目标板

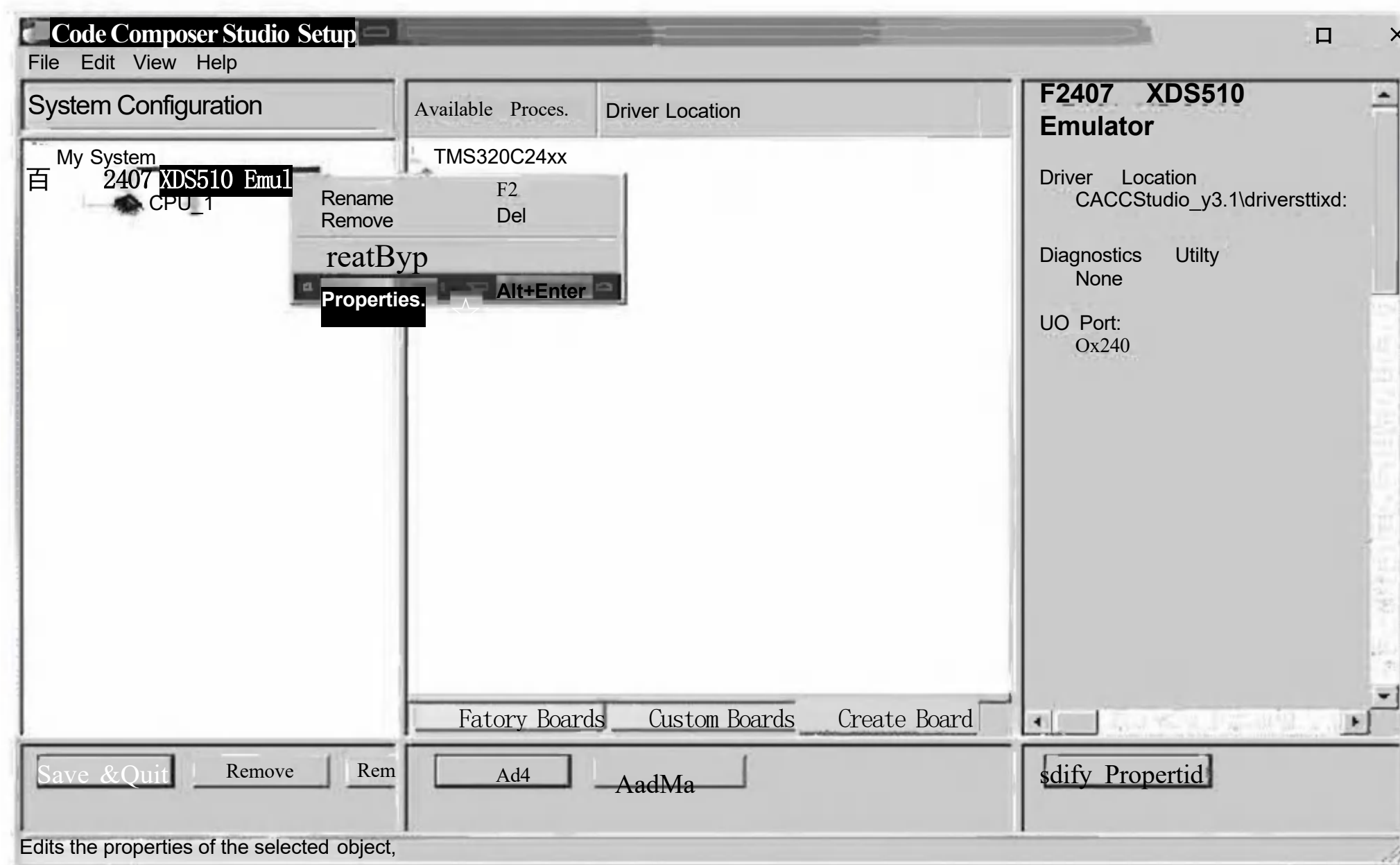


图4.28 选择目标板属性

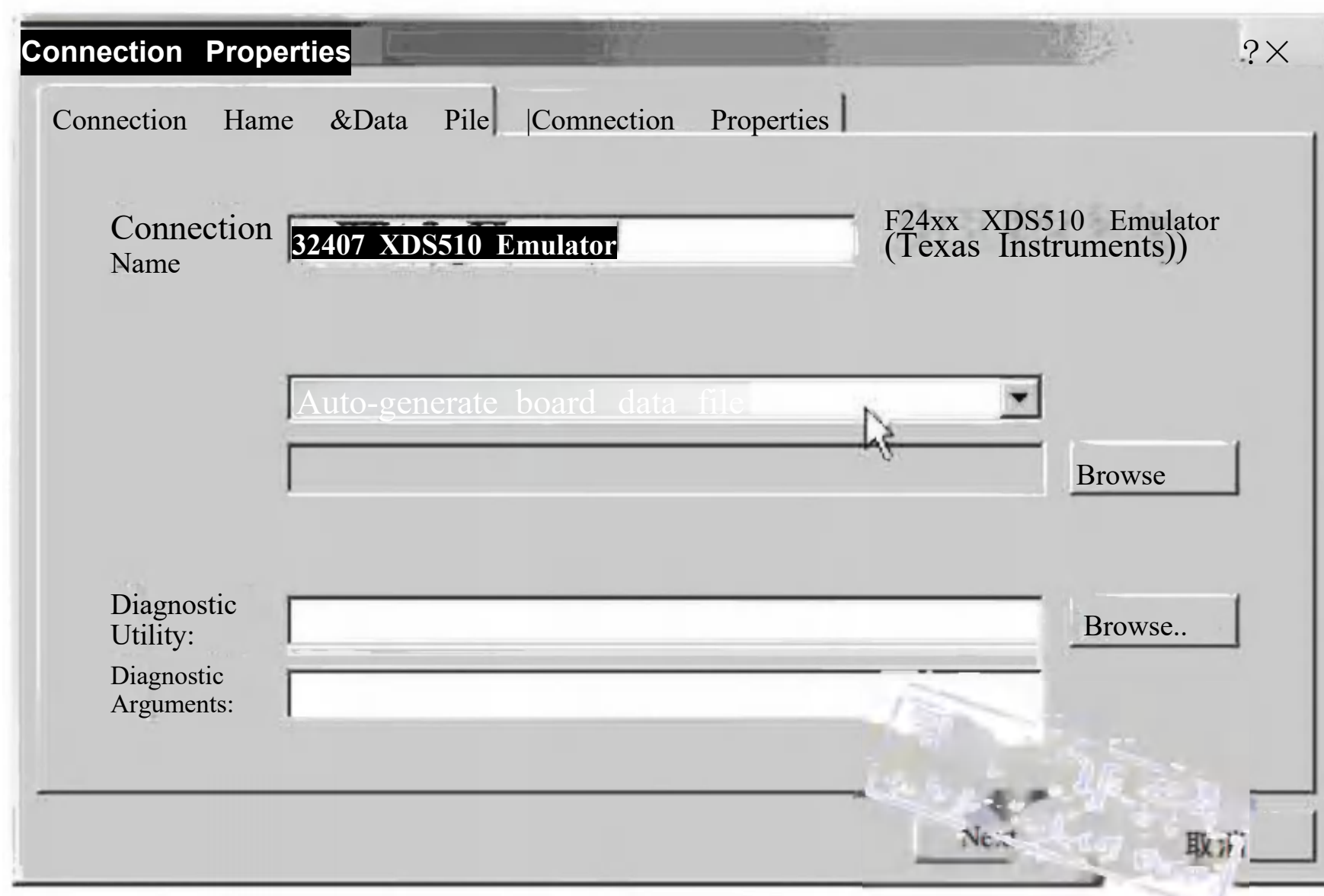


图4.29 属性对话框

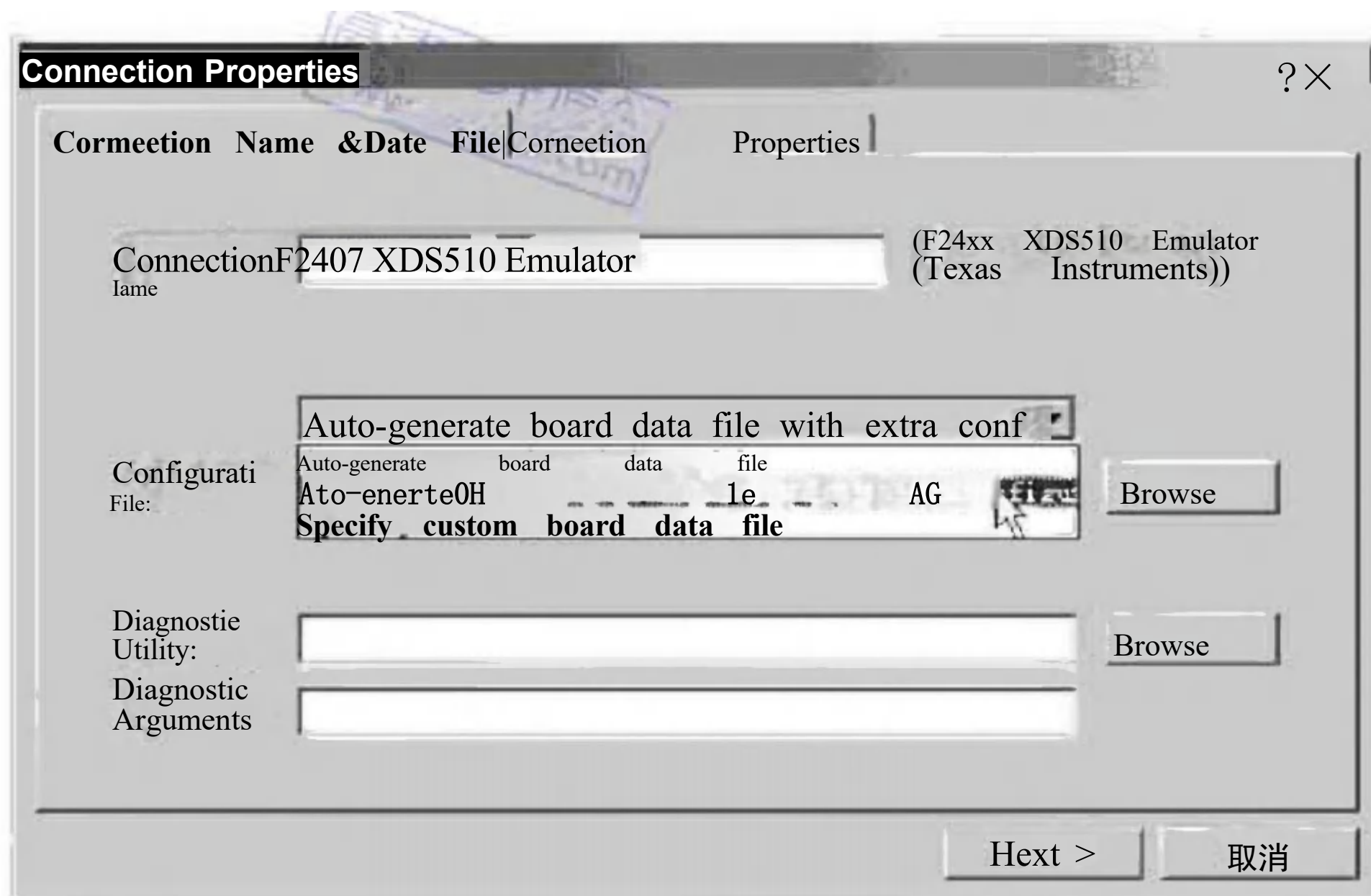


图4. 30 属性配置文件选择对话框

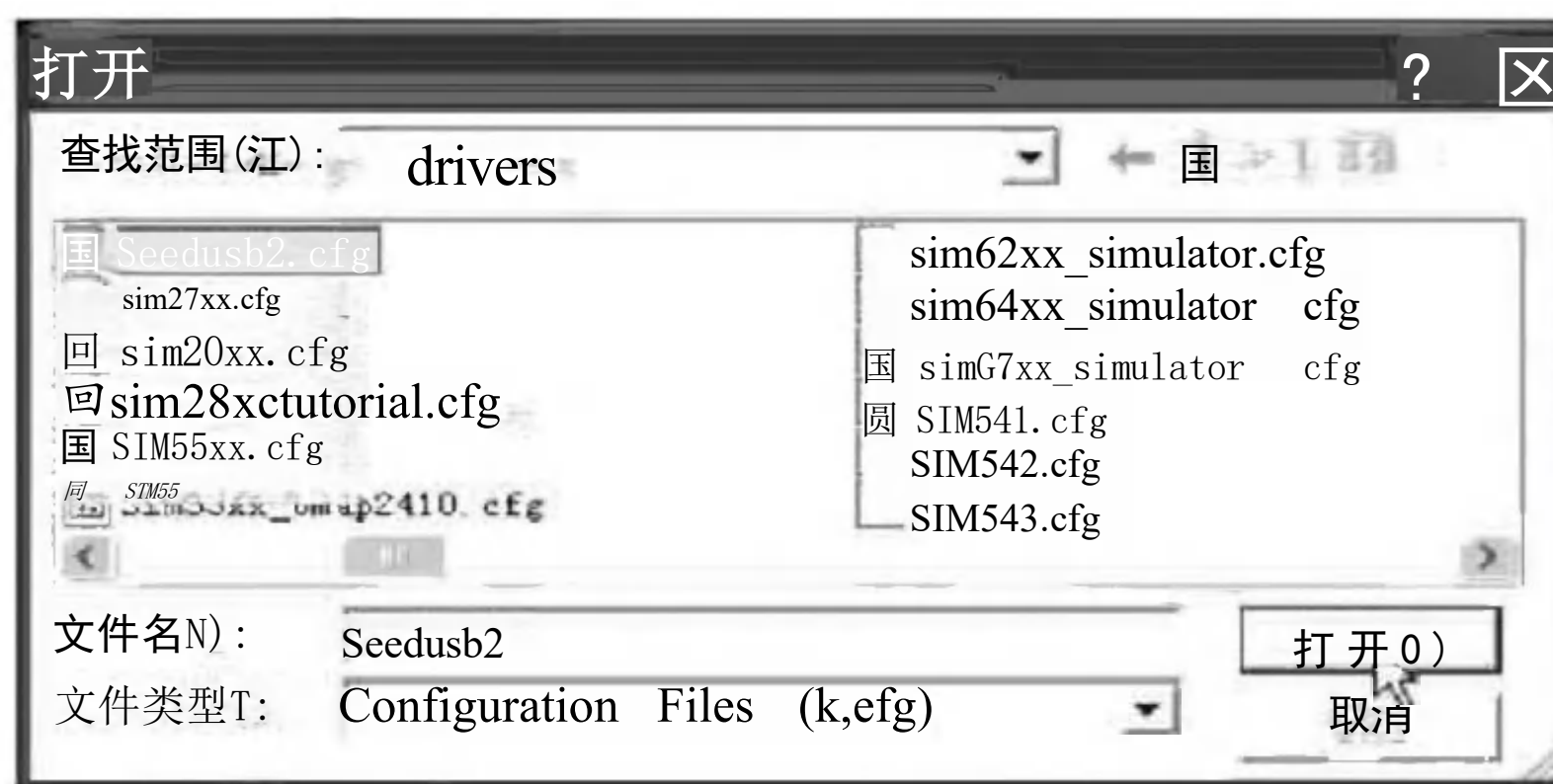


图4. 31 配置文件选择对话框

- (6) 单击Next 按钮，出现如图4. 33所示对话框。
- (7)I/O Port 为 0x240(也可以设为0x340)。单击 Finish 按钮，回到图4. 27所示对话框。
- (8) 单击图4 . 27中的 Save & Quit 按钮，出现如图4 . 34所示对话框。
- (9) 单击“否”按钮，关掉CCS Setup,表明不启动CCS 集成开发环境；选择“是”按钮，则启动CCS 集成调试环境。

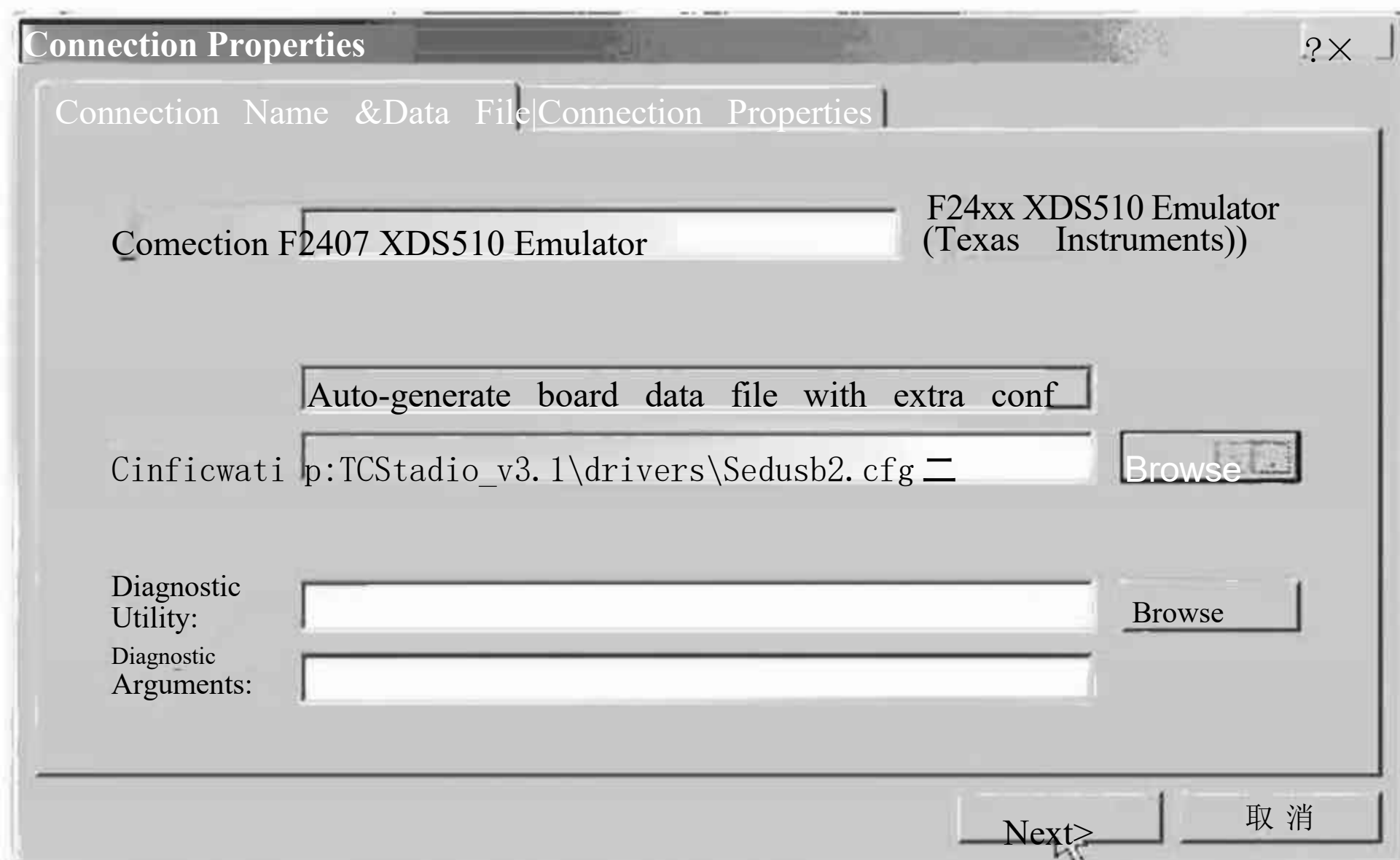


图4.32 配置属性选择结果

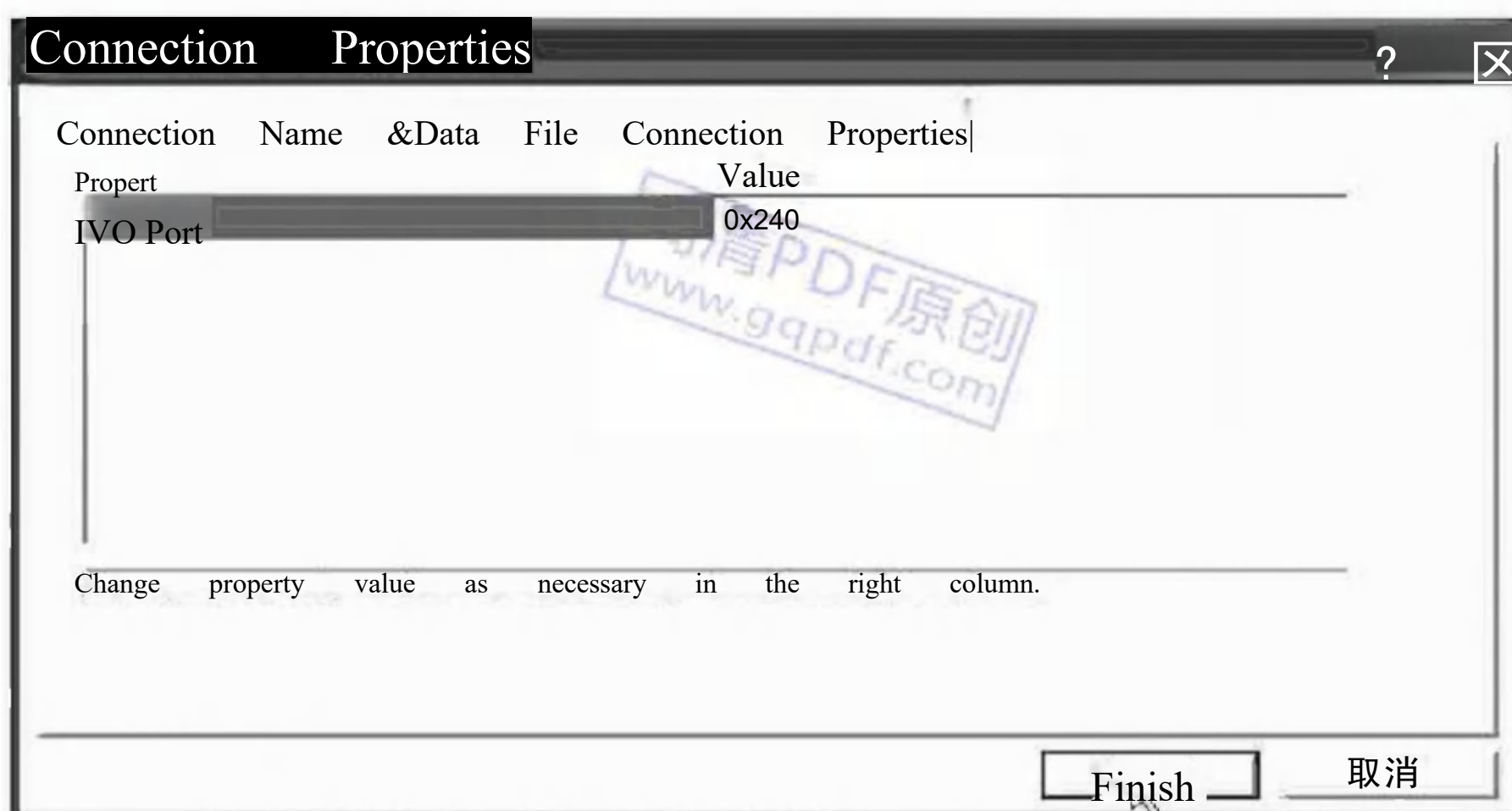


图4.33 端口配置对话框

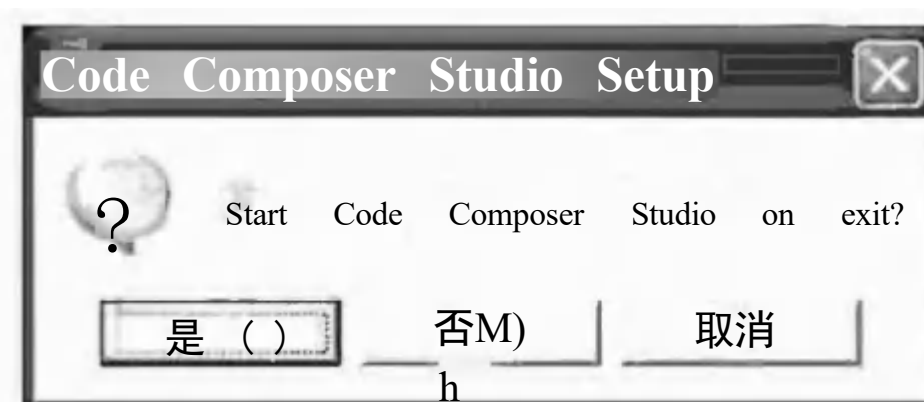


图4.34 是否启动 CCS 集成开发环境对话框

第 5 章 CCS3.1 集成调试环境简介

完成对系统的配置之后，给系统上电，双击快捷方式CCStudio 3.1 文件图标，即可进入图5.1所示集成开发环境。

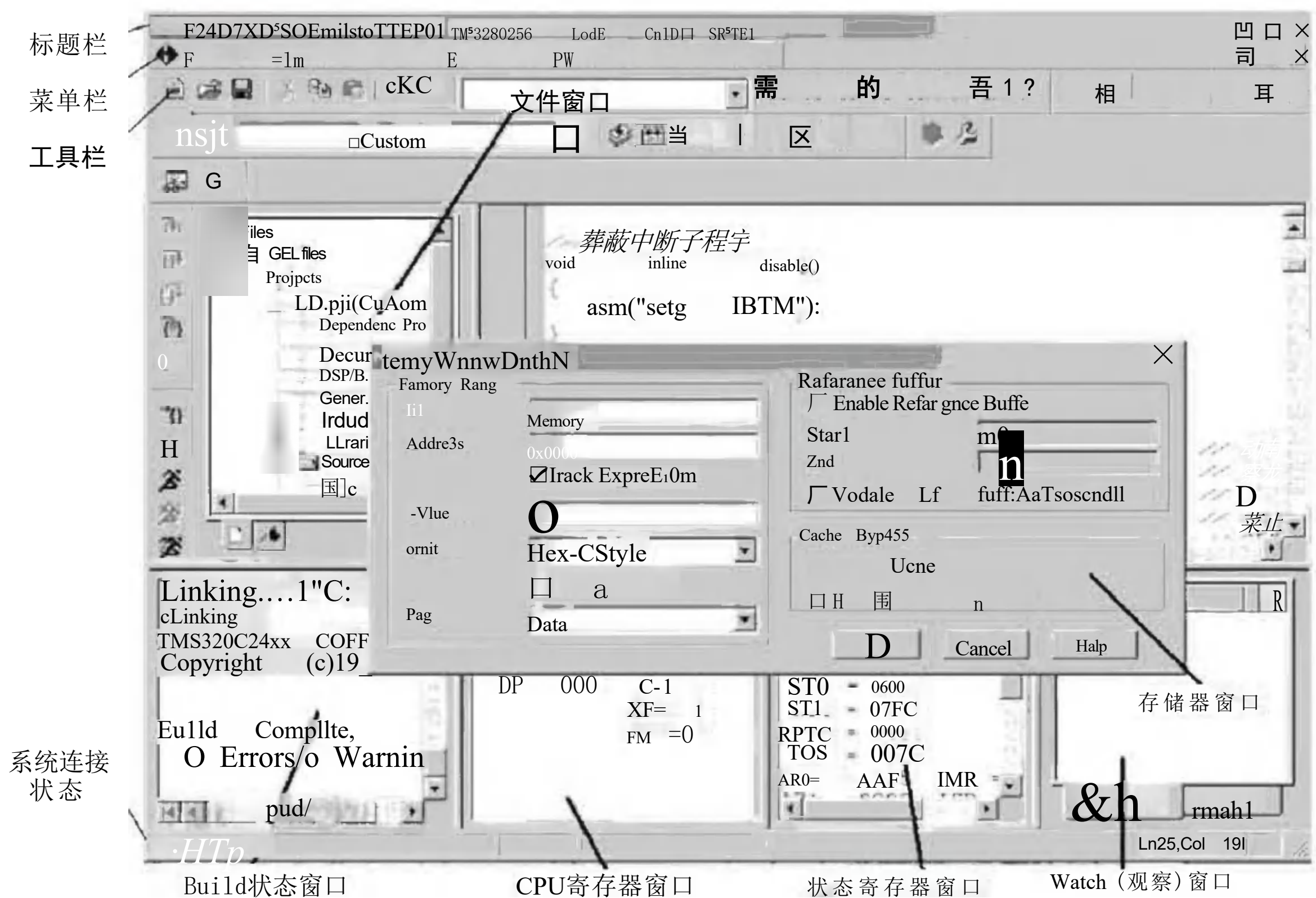


图5.1 CCS3.1 集成调试环境窗口

5.1 CCS3.1 集成调试环境主要菜单及功能

5.1.1 Project(项目)菜单

CCS 集成开发环境不能直接将汇编源代码或C 语言源代码文件 Build 生成 DSP 可执行代码。必须使用项目(Project) 来管理整个设计和调试过程。项目文件保存时为*. pjt 文件。Project 菜单中的主要命令如下:

- >Project→New 新建一个项目, 将该项目保存至新建项目文件夹中。
- >Project→Open 打开一个已有的项目。
- >Project→Add Files to Project 将文件加载到该项目中。项目中可以加入的文件的扩展名如下:
 - *.C C 源程序文件。项目管理将对这一类文件进行编译和链接。
 - *.ASM 汇编源程序文件。项目管理将对这一类文件进行汇编和链接。
 - *.OBJ 目标文件。项目管理将对这一类文件进行链接。
 - *.LIB 库文件。项目管理将对这一类文件进行链接。
 - *.CMD 链接命令文件。项目管理在链接各个文件时, 根据此文件分配系统程序空间及数据空间。

对于头文件和在程序中用include 引用的文件, 项目管理会自动加入。项目管理不允许用户添入其他类型的文件。

- >Project→Save 将一个已打开的项目保存。
- >Project→Close 将一个已打开的项目关闭。
- >Project→Add Files to Project 添加该项目中用到的所有文件和所需的库文件。
- >Project→Compile Files 对项目中的C 语言和汇编语言源代码文件进行编译。
- >Project→Build 对项目进行编译、汇编和链接, 生成可执行文件为*. OUT。对于没有修改的源程序文件不重新编译。
- >Project→Rebuild All 对项目重新进行一次编译、汇编和链接, 生成可执行文件为*. OUT。
- >Project→Stop Build 停止对项目的编译、汇编和链接。
- >Project→Show Project Dependencies 显示该项目中用到的所有文件和所需的库文件。
- >Project→Scan All File Dependencies 详细查看该项目中用到的所有文件和所需的库文件。
- >Project→Build Options 对编译器、汇编器和链接器的参数进行配置。如图5.2所示, 可以设置寄存器优先调试、特定函数优先调试或某个子程序优先调试等选项。这为某

些程序的调试和修改带来方便。通常在Build 时采用默认设置。

>Project→Recent Project Files 显示最近打开过的项目。

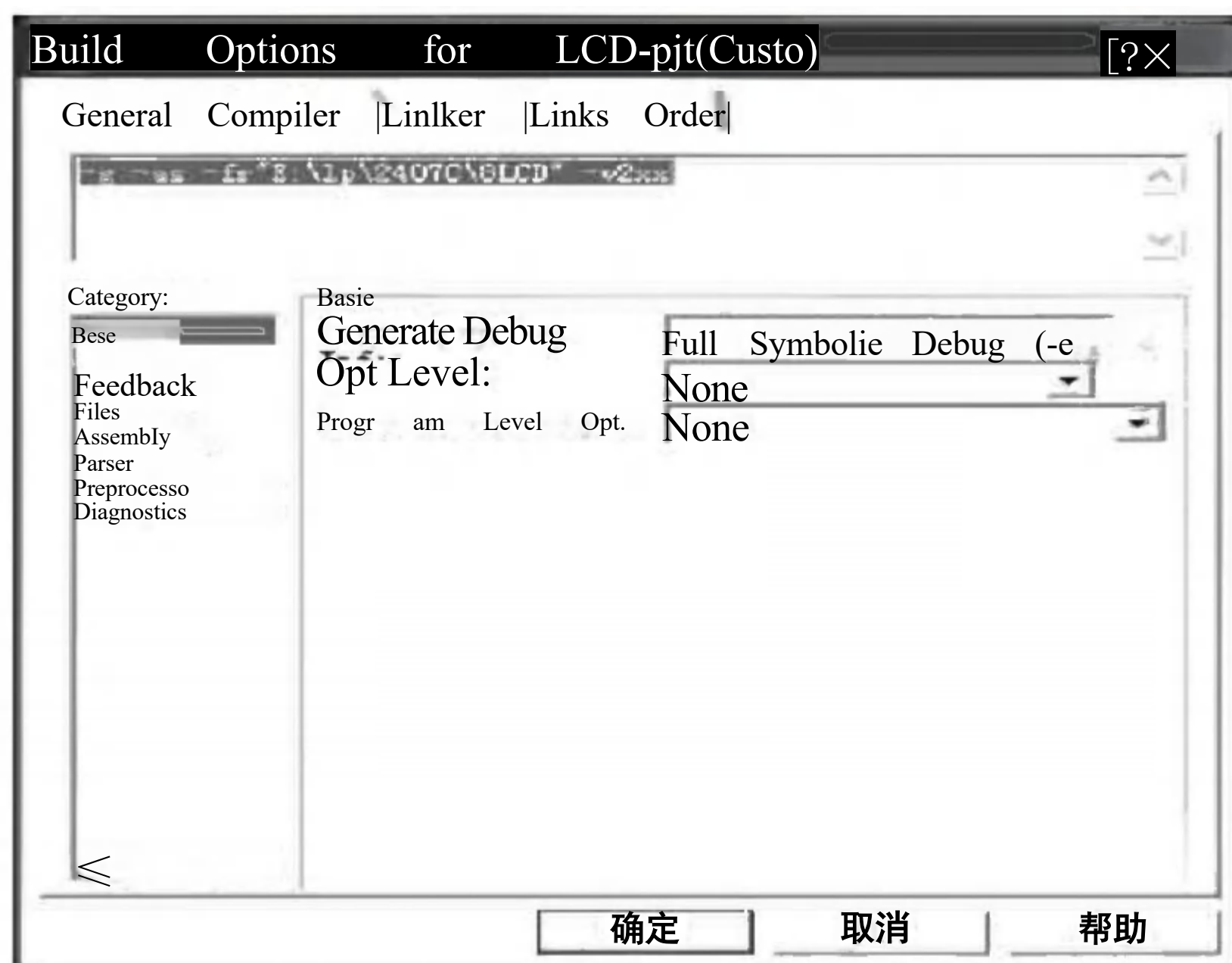


图5.2 编译器设置对话框

5.1.2 View(观察)菜单

在 View 菜单中，可以选择是否显示各种工具栏、窗口和对话框等。其中比较重要的命令如下：

>View→Disassembly 观察反汇编程序窗口。当将 DSP 可执行程序*.OUT 文件载入

目标系统后，CCS 集成开发环境将自动打开一个反汇编窗口。反汇编窗口根据存储器的内容显示反汇编指令和符号信息。

>View→Memory 观察存储器窗口。单击此选项，屏幕上出现如图5.3所示对话框。

在Address 文本框中输入需要观察的存储器的起始地址。若 Q-Value 文本框填0, 则后面出现的存储器窗口中的显示值就是实际值(也可以填其他数, 但后面出现的存储器窗口中的显示值不是实际值)。在Format 下拉列表框中可以选择存储器的显示格式, 如十六进制和二进制等。如果显示时须采用IEEE 浮点格式, 则选择 Use IEEE Float 选项。从Page 下拉列表框中, 可以选择显示存储器的类型: 程序存储器(Program)、数据存储器(Data) 和 I/O 空间。Enable Reference Buffer 复选框以及其他选项较少用到。

>View→Registers 包含 CPU 和 Status 两个选项, 单击相应的选项可以观察调试过程

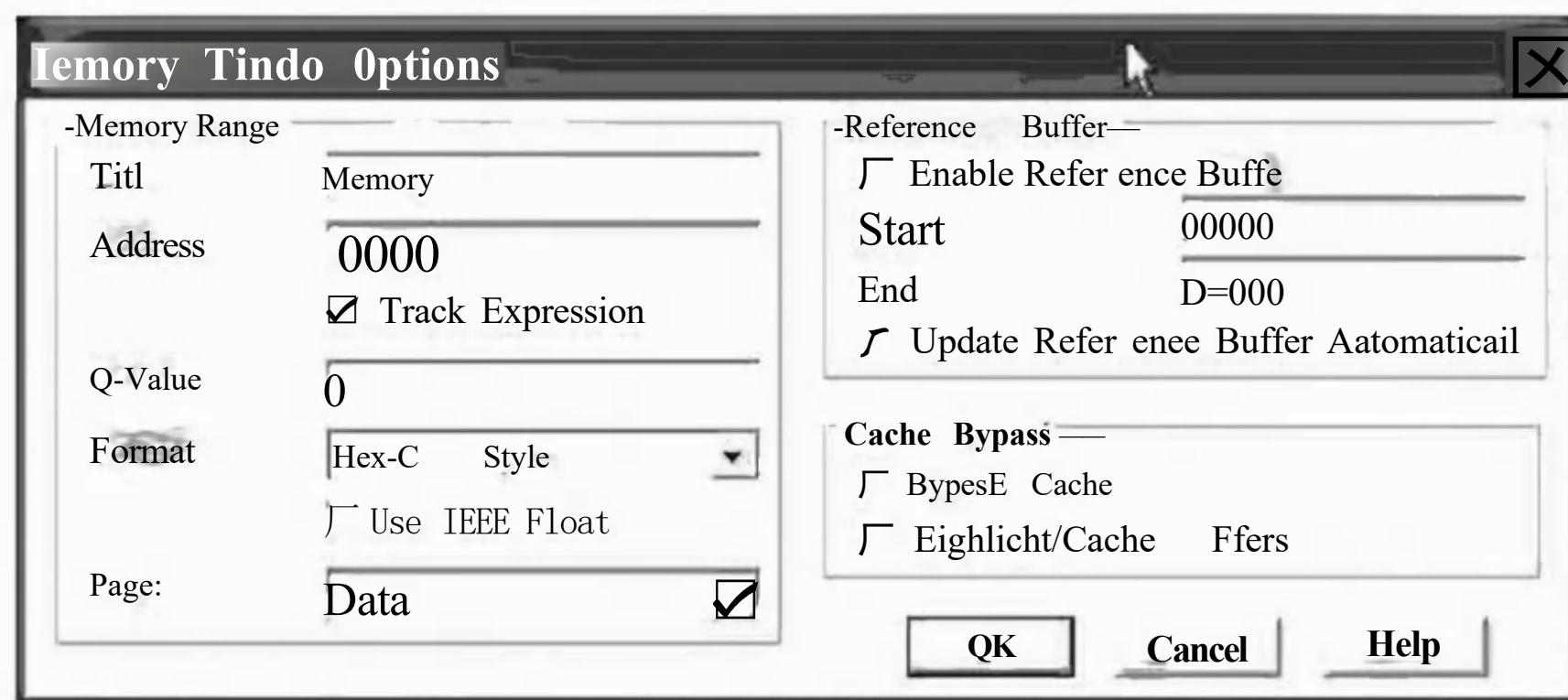


图5.3 观察存储器对话框

中 CPU 寄存器和各个状态位的变化。

>View→Peripherals 观察调试过程中外设寄存器的变化。

>View→Watch Window 观察调试过程中的变量、C 语言表达式、地址和寄存器的值。选择此选项后，将有一个空白窗口出现在CCS 集成调试环境窗口的下部。

通常用如下方法在 Watch Window 中加入一个新的表达式：在 Watch Window 中的 Name 文本框中输入要观察的变量名，或者在程序窗口中选中要观察的变量名，然后单击右键，再在打开的快捷菜单中选择 Add to Watch Window 即可，如图5.4所示。

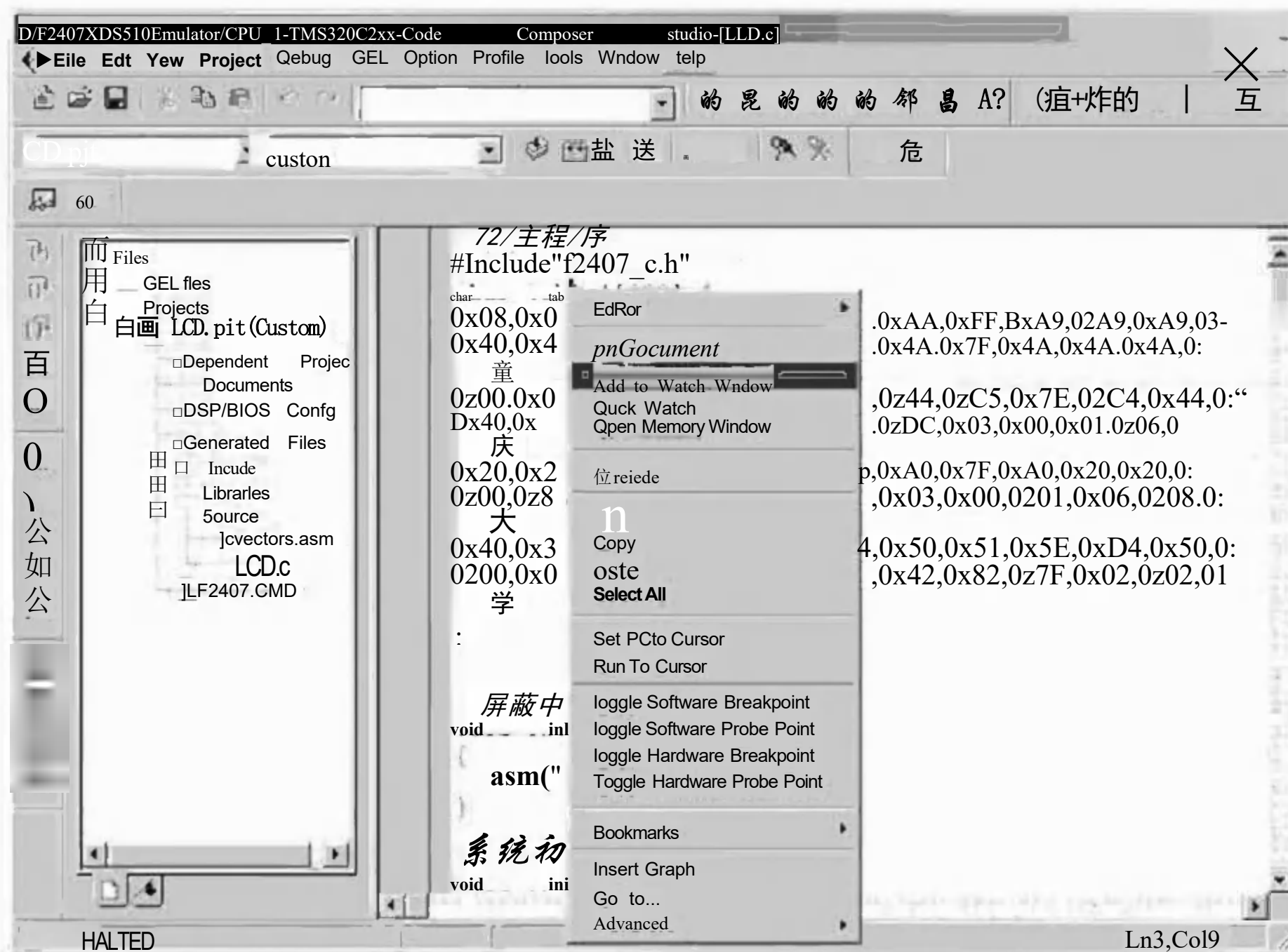


图5.4 添加观察窗口

若在 Watch Window 中删除一个表达式，则选择待删除的表达式，再右击鼠标，在弹出的快捷菜单中选择Delete Selected Item。

如果添加变量时在“添加变量对话框”文本框中只输入变量名，则在 Watch Window 中只显示该变量的地址。如果需要显示该变量的值，则需要变量名前加“※”号。

如果变量标志在一个 C 源程序文件中被定义，并且特别指定了调试信息标志(-g)，则变量标志代表的是相应地址的值。

变量的地址或值的显示默认格式为十进制。如果想改变显示格式，则在输入变量时在其后面紧跟一个逗号和格式指示字母。常用的格式指示字母与其代表的格式的对应关系如表5. 1所列。

表5. 1 变量显示常用格式列表

字母符号	代表的格式	字母符号	代表的格式
D	十进制	(	八进制
E	指数浮点	U	无符号整型
F	十进制浮点	C	ASCII字符(字节)
X	十六进制		

>View→Mixed Source 选择此选项则能同时显示 C 语言代码及与之关联的反汇编代码(反汇编代码位于C 语言代码下方)。在进行C 语言编程时该选项特别有用。若需要取消此功能，只需再次单击此选项即可。

5.1.3 Debug(调试) 菜单

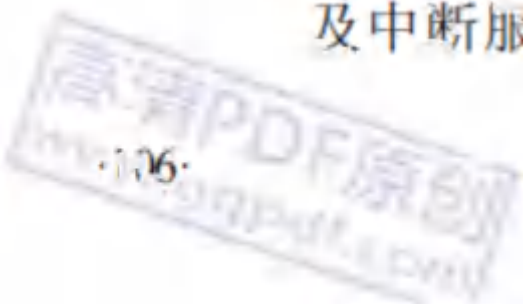
Debug 菜单中包含所有可用的调试命令. 其中比较常用的命令如下：

>Debug→Breakpoints 断点设置。将光标放在程序中需要设置断点处，选择Debug→Breakpoints 并进行相应的设置，可在程序中设置断点。但这种方法设置断点较为复杂，最简单的方法是移动鼠标至需要设置断点的程序语句上，右击鼠标，在弹出的快捷菜单中选择Toggle breakpoint 即可。若要取消断点，则选中相应的断点处语句，再右击鼠标，在弹出的快捷菜单中再次选择Toggle breakpoint 即可。

>Debug→Probe Points 探测点设置。在设置探测点的地方，将主机文件的数据读到 DSP 目标系统的存储器中，或将 DSP 目标系统存储器的数据写入主机的文件中。

>Debug→Step Into 单步执行。如果运行到调用函数处，则跳入函数子程序中继续单步执行。

>Debug→Step Over 单步执行。与Step Into不同的是，它不进入调用的程序(子程序及中断服务程序)。即该命令一步执行完被调用的子程序。



>Debug→Step Out 跳出函数子程序。当使用 Step Into 或 Step Over 单步执行指令时，如果程序运行到一个子程序中，那么执行该命令将使程序执行完函数或子程序后，回到调用的地方。

>Debug→Run 执行程序到断点、探测点或用户中断(ESC)。

>Debug→Halt 当执行Run 时，用Halt 来停止程序的执行。

>Debug→Reset CPU 对处理器进行复位。复位 DSP 目标系统，终止程序的执行，初始化所有的寄存器。

>Debug→Restart 将程序指针指向程序的起始地址。

>Debug→Reset Emulator 对仿真器进行复位。

以上只是简单地介绍了 CCS 集成开发环境中的一些常用菜单，在该集成开发环境中还有其他一些菜单，这里不再详细介绍。一些常用的菜单在 CCS 集成开发环境中都有快捷键，用户可从其帮助文件中得到详细信息。

5.2 工作窗口区

在CCS 集成开发环境中，常用的工作窗口有Files（文件）窗口、Disassembly（反汇编程序）窗口、CPU 寄存器窗口、Memory（存储器）窗口以及 Watch(观察)窗口等，如图5.1所示。根据需要，用户可对这些窗口分别打开、关闭、调整大小、移动和浏览窗口的内容等。

1.Files（文件）窗口

该窗口包括当前系统用到的GEL 文件和项目。通常在GEL 文件中包括缺省的f2407.gel 文件。在项目中包括用户用到的头文件、用include 定义的引用文件和用户建立或添加的C 源程序文件*.C、汇编源程序文件*.ASM、目标文件*.OBJ、库文件*.LIB 和链接命令文件 *.CMD。把鼠标移到相应的*.pjt 文件夹处，单击右键，可以从弹出的快捷菜单中选择 Add Files to Project 选项，选择添加适当的文件。如果须删除文件，则把鼠标移到相应文件处，单击右键，从弹出的快捷菜单中选择 Remove from Project 选项即可。

2.Disassembly（反汇编程序）窗口

该窗口用来显示程序的反汇编代码和程序存储器的内容。在这个窗口中，用一反显高亮条来表示当前程序指针。同时可用鼠标单击汇编语句的方法来设置断点，再单击一次，则取消设置的断点。

3.CPU 寄存器窗口

该窗口显示在程序调试过程中的 CPU 寄存器(程序指针 PC、累加器 ACC、状态寄存器 STO 和 ST1、重复计数器 RTC、辅助寄存器 XAR0~XAR7、中断使能寄存器 IER 和中断标志寄存器 IFR 等)的状态。用户可通过CPU 寄存器窗口来监视程序的执行过程。若要改变寄

寄存器内容，则可单击需要改变的寄存器，输入值后回车即可。

4. 状态寄存器窗口

该窗口显示状态寄存器中的各个状态位：SXM、CVM、T、C、Z、N、V、PM、OVC、INTM、DBGM、PAGE0、VMAP、SPA、LOOP、EALLOW、IDLES7A7、AMODE、OBJMODE、CNT、M0M1MAP、XF 及 ARP。通过该窗口，用户可以很清楚地看到程序执行过程中各状态位的变化。若要改变状态位内容，则可单击需要改变的状态位，输入值后回车即可。

5.Memory（存储器）窗口

该窗口显示存储器的内容。存储器可为程序存储器、数据存储器 and I/O 空间，其详细配置见5.1.2小节中的 View→Memory 内容。若要改变存储器内容，则单击需改变的存储器地址，输入值后回车即可。注意，某些存储器内容不能改变。

6.Watch（观察）窗口

该窗口用来显示所选择的变量、寄存器和存储器的内容，其详细配置见5.1.2小节中的 View→Watch Window 内容。

7.Build（建立）状态窗口

对项目进行编译、汇编和链接时，该窗口显示一些状态信息。如果在编译、汇编和链接过程中发现错误，则该窗口显示错误类型和错误的位置。用户可根据此信息对发生错误的地方进行修改。单击错误信息处，光标就跳到程序中相应的位置，调试使用起来十分方便。

5.3 4个基本文件和1个库函数文件简介

用DSP 的汇编语言和C 语言编写程序时，使用CCS3.1 集成开发环境的情况与代码编译器CC4.10 编译环境不一样：在代码编译器 CC4.10 编译环境中，须使用Windows 操作系统中的记事本编辑程序，或者使用UltraEdit-32 文本编辑器编写程序；而在CCS3.1 集成开发环境中，可以直接编写汇编语言程序和C 语言程序，编写程序完成后，直接运行编译链接等调试，不用像CC4.10 编译器那样，进行从编译环境到文本编辑器之间麻烦的转换。CCS3.1 集成调试环境功能更强，调试更方便。

若要编写一些模块化的程序块，则在进行调试之前，一般需要1个库函数文件和4种格式的文件：C 语言文件、汇编语言文件、头文件和命令文件。编写这4类文件时，通常要用到一些汇编伪指令。

F240x 系列芯片的伪指令极为丰富，限于本书篇幅，现仅列举几个伪指令如下。其他F240x 伪指令的信息介绍，请参见“TMS320C1X/C2X/C2XX/C5X Assembly Language Tools User's Guide”。

几个常用汇编伪指令介绍如下：

1. 定义段的伪指令

定义段的伪指令把汇编语句程序的各部分与适当的段联系起来。

(1).asect 创建具有绝对地址的初始化命名段。用 .asect 定义的段可以包含代码和数据。在绝对段内，用户可以使用 .label 伪指令来定义可重定位标号。

(2).bss 为未初始化的变量在 .bss 段内保留空间。

(3).sect 定义初始化命名段并把后续代码或数据与该段相联系。用 .sect 定义的段可以包含代码或数据。

(4).text 标示 .text 段内代码部分。text 段通常包含可执行代码。

(5).usect 在未初始化命名段内保留空间。 .usect 伪指令和 .bss 伪指令相类似，但是它允许用户与段 .bss 分开保留空间。

2. 初始化常数的伪指令

(1).bes 和 .space 在当前段中保留特点的位。汇编器用0填充这些保留位。用户可以通过把位数乘16来保留规定的字数。当使用 .space 的标号时，它指向保留位的第一个字。当使用 .bes 标号时，它指向保留位的最后一个字。

(2).byte 将一个或多个8位的数值置入当前段连续的字中。除了每个数值的宽度被限制为8位以外，此伪指令与 .word 相类似。

(3).field 将单个数值置入当前字规定的位数中。用户可以用 .field 伪指令将多个域(field) 组装到单个字中。在字被填满之前，汇编器将不使 SPC 增量。

(4).float 和 .bfloat 计算单精度32位 IEEE 格式浮点数，并会存入当前段两个连续的字中：先存储低字，后存储高字。 .bfloat 伪指令保证定义的浮点数不跨越数据页边界，这是 float 前加 b 的含义。

(5).int 和 .word 将一个或多个16位值置入当前段的连续字中。

(6).long 和 .blong 将32位数值放入当前段的连续两个字块中。低位字优先存放。 .blong 伪指令保证目标会不跨越数据页边界。

(7).string 将8位字符从一个或多个字符串置入当前段。除了把两个字符组装入每一个字外，这个伪指令类似于 .byte。需要时，字符串中最后一个字用空字符(0) 填充。

3. 调准段程序计数器的伪指令

(1).align 把 SPC 调准在128个字的边界。这确保了跟随在 .align 伪指令之后的代码从数据页边界处开始。如果 SPC 已调至数据页边界，那么它不再增量。

(2).even 调准SPC，使其指向下一个完整字。在使用 .field 伪指令之后，用户应当使用 .even。如果 .field 伪指令未填满一个字，那么 .even 伪指令将使汇编器填满全字，并把未用到的位填0。

4. 引用其他文件的伪指令

(1).copy 和 .include 告诉汇编器开始从其他文件中读入源语句。当汇编器完成从copy(复制)或者include(包含)文件内读源语句时,它恢复从当前文件读源语句。

(2).def 识别在当前模块中定义且可以被其他模块使用的符号。

(3).global 声明外部符号以便在链接时可将其用至其他模块。global 伪指令对于已定义符号起.def的作用.对于未定义符号起.ref的作用。

(4).ref 识别在当前模块中使用,但在其他模块中定义的符号。

5. 条件汇编伪指令

条件汇编伪指令使用户能指示汇编器根据表达式求值结果的“真”或“假”来汇编代码的某些段。共有两组伪指令允许用户汇编代码的条件块。

(1).if/.elseif/.else/.endif 告诉汇编器根据表达式的值有条件地汇编代码块。

if expression 标志条件块的开始。如果.if条件为“真”,那么汇编后面的代码块。

.elseif expression 如果.if条件为“假”且.elseif为“真”,那么汇编后面的代码块。

.else 如果.if为“假”,那么汇编后面的代码块。

.endif 标志条件块的末尾并结束条件块。

(2).loop/.break/.endloop 告诉汇编器根据表达式的值来重复地汇编代码块。

.loop expression 标志可重复代码块的开始。

.break expression 告诉汇编器如果.break表达式为“假”,那么继续重复汇编;如果表达式为“真”,那么转移到紧接在.endloop之后的代码。

.endloop 标志可重复块的末尾。

6. 汇编时(Assembly-Time) 符号伪指令

(1).asg 把字符串赋给替代符号。

(2).set 和 .equ 把常量值赋予符号。

(3).struct/.endstruct 建立类C 语言的结构定义。tag 伪指令把类C 的结构特性赋给标号。

类C 的结构定义使用户能把类似的元素组合在一起,然后把元素偏移量(offset)的计算留给汇编器。struct/.endstruct 伪指令不分配存储器,它们简单地创建可重复使用的符号模板。

通过把结构特性赋给标号,.tag 伪指令简化了符号表示,并提供了定义结构的能力。该结构可包含其他结构。.tag 伪指令不分配存储器,结构标记必须在使用之前定义。

(4).eval 对表达式求值,把结果转化为字符,并把字符赋给替代符号。

7. 其他伪指令

(1).end 终止汇编。它是一个程序的最后一条源语句。此伪指令与文件结束符具有同

样的效果。

(2).label 定义特定一个的标号。它表示当前段的装载地址。当段在一个地址处装载，但在另一个地址处运行时，该指令特别有用。

(3).version 告诉汇编器该代码是属于哪一种处理器。 .version伪指令必须出现在指令之前，否则将出错。

5.3.1 C 语言程序文件

C 语言程序文件名的后缀为 .c，在 C 语言中实现 DSP 要完成的功能，由开发人员编写。

5.3.2 寄存器头文件 F2407 C.h

F2407_C.H 头文件中定义DSP 系统用到的一些寄存器映射地址，用户用到的常量和用户自定义的寄存器都在该文件中定义。该文件名的后缀为.H。寄存器头文件可以从其他文件或相关资料中复制，或复制后根据需要进行修改。以下为头文件 F2407_C.H 示例。

头文件 F2407 C.H-- 在该头文件中，寄存器都以指针方式进行寻址。

```

;File      Name:  F2407_C.h
;Originator:Texas      Instrument-Chongqing      University      DSPs      Lab.
;Description:LF2407      Peripheral      Registers      +other      useful      definitions
;*****/
/*片内外围模块寄存器定义~~~~~*/
/*C2xx 内核寄存器~~~~~*/
volatile unsigned int*IMR=(volatile unsigned int*)0x0004;
/*中断屏蔽寄存器*/
volatile unsigned int*GREG=(volatile unsigned int*)0x0005;
/*全局变量定位寄存器*/
volatile unsigned int*IFR=(volatile unsigned int*)0x0006;
/*中断标志寄存器*/
volatile unsigned int*ABRPT =(volatile unsigned int*)0x01f;
/*分析断点*/
ioport      unsigned      portFFFF;
#define WSGR      portFFFF
/*系统配置寄存器*/
volatile unsigned int      *PIRQRO      =(volatile unsigned int *)0x7010;
/*外围中断请求寄存器0*/
volatile unsigned int      *PIRQR1      =(volatile unsigned int *)0x7011;
/*外围中断请求寄存器1*/
volatile unsigned int      *PIRQR2      =(volatile unsigned int *)0x7012;

```

```

*外围中断请求寄存器2*
volatile unsigned int    PIAKRO      (volatile unsigned int *)0x7014;
/*外围中断应答寄存器0*7
volatile unsigned int    PIAKR1      =(volatile unsigned int*)0x7015;
/*外围中断应答寄存器1*/
volatile unsigned int    PIAKR2      =(volatile unsigned int*)0x7016;
*外围中断应答寄存器2*/
volatile unsigned int    SCSRI      三 (volatile unsigned int )0x7018;
/*系统控制和状态寄存器1*
volatile unsigned int    SCSR2      =(volatile unsigned int )0x7019;
/*系统控制和状态寄存器2
volatile unsigned int    DIN        =(volatile unsigned int )0x701C;
/*模块识别寄存器
volatile unsigned int    PVIR      =(volatile unsigned int*)0x701E;
/*外围中断向量寄存器*
*看门狗/实时中断(RTI)/ 锁相环(PIL) 寄存器 */
volatile unsigned int    WDCNTR      =(volatile unsigned int )0x7023;
/*看门狗计数寄存器*
volatile unsigned int    WDKEY      =(volatile unsigned int #)0x7025;
/*看门狗 Key 寄存器 *
volatile unsigned int    WDCR      =(volatile unsigned int *)0x7029;
/长看门狗控制寄存器
/关外围串行接口(SPI) 寄存器~~~~~*
volatile unsigned int    SPICCR      =(volatile unsigned int *)0x7040;
*SPI 配置控制寄存器1*
volatile unsigned int    SPICTL      (volatile unsigned int *)0x7041;
/*SPI 运行控制寄存器2*/
volatile unsigned int    SPISTS      =(volatile unsigned int*)0x7042;
*SPI 状态寄存器
volatile unsigned int    SPIBRR      =(volatile unsigned int*)0x7044;
/*SPI 波特率控制寄存器*
volatile unsigned int    SPIEMU      =(volatile unsigned int )0x7046;
/若 SPI仿真缓冲寄存器
volatile unsigned int    SPIRXBUF     =(volatile unsigned int *)0x7047;
人* SPI 串行输入缓冲寄存器 /
volatile unsigned int    SPITXBUF     (volatile unsigned int )0x7048;
*SPI 串行输出缓冲寄存器*
volatile unsigned int    SPIDAT      =(volatile unsigned int )0x7049;
/*SPI 串行数据寄存器

```

```

volatile unsigned int    SPIPC1
/*SPI  端口控制寄存器1*/
volatile unsigned int    *SPIPC2
/*SPI  端口控制寄存器2*/
volatile unsigned int    *SPIPRI
/*SPI  优先级控制寄存器*/
/*串行通信接口寄存器~~~~~*/
volatile unsigned int    SCICCR
/*SCI  通信控制寄存器*/
volatile unsigned int    *SCICTL1
/*SCI  控制寄存器1*/
volatile unsigned int    *SCIHBAUD
/*SCI  波特率控制寄存器高位*/
volatile unsigned int    SCILBAUD
/*SCI  波特率控制寄存器低位*/
volatile unsigned int    *SCICTL2
/*SCI  控制寄存器 2*/
volatile unsigned int    *SCIRXST
/*SCI  接收状态寄存器*/
volatile unsigned int    *SCIRXEMU
/*SCI EMU 数据缓冲寄存器*/
volatile unsigned int    *SCIRXBUF
/*SCI  接收数据缓冲寄存器*/
volatile unsigned int    *SCITXBUF
/*SCI  发送数据缓冲寄存器*/
volatile unsigned int    *SCIPRI
/*SCI  优先级控制寄存器*/
/*外部中断配置寄存器~~~~~*/
volatile unsigned int    *XINT1CR
/*Int1  配置寄存器*/
volatile unsigned int    *XINT2CR
/*Int2  配置寄存器*/
/*数字量I/O 控制寄存器~~~~~*/
volatile unsigned int    MCRA
/*I/O  复用控制寄存器A*/
volatile unsigned int    *MCRB
/*I/O  复用控制寄存器 B*/
volatile unsigned int    *MCRC
/*I/O  复用控制寄存器 C*/

```

```

=(volatile unsigned int*)0x704D;

=(volatile unsigned int *)0x704E;

=(volatile unsigned int*)0x7023;

~~~~~*/
=(volatile unsigned int*)0x7050;

=(volatile unsigned int*)0x7051;

=(volatile unsigned int*)0x7052;

=(volatile unsigned int*)0x7053;

=(volatile unsigned int*)0x7054;

=(volatile unsigned int*)0x7055;

=(volatile unsigned int*)0x7056;

=(volatile unsigned int *)0x7057;

=(volatile unsigned int*)0x7059;

=(volatile unsigned int*)0x705F;

~~~~~*/
=(volatile unsigned int *)0x7070;

=(volatile unsigned int*)0x7071;

~~~~~*/
=(volatile unsigned int*)0x7090;

=(volatile unsigned int *)0x7092;

=(volatile unsigned int *)0x7094;

```



```

volatile unsigned int *IPSRA      =(volatile unsigned int *)0x7094;
/关输入状态寄存器A*/
volatile unsigned int  IPSRB      =(volatile unsigned int  )0x7096;
    输入状态寄存器 B
volatile unsigned int  PADATDIR    =(volatile unsigned int*)0x7098;
/*I/O  端口A 数据和方向控制寄存器*
volatile unsigned int  *PBDATDIR  =(volatile unsigned int*)0x709A;
/关I/O  端口B 数据和方向控制寄存器关/
volatile unsigned int  *PCDATDIR   =(volatile unsigned int*)0x709C;
/  I/O  端口 C数据和方向控制寄存器*/
volatile unsigned int  PDDATDIR    =(volatile unsigned int  )0x709E;
    *I/O  端口D 数据和方向控制寄存器×
volatile unsigned int  PEDATDIR    =(volatile unsigned int  )0x7095;
    *I/O  端口E 数据和方向控制寄存器*
volatile unsigned int  PFDATDIR    =(volatile unsigned int*)0x7096;
    I/O  端口F 数据和方向控制寄存器*
    *ADC 寄存器定义 ~~~~~~*
volatile unsigned int  ADCTRL1     (volatile unsigned int*)0x70A0;
/长ADC控制寄存器1*/
volatile unsigned int  ADCTRL2     =(volatile unsigned int*)0x70A1;
    ×ADC控制寄存器2*/
volatile unsigned int  MAXCONV      =(volatile unsigned int  )0x70A2;
    关最大转换通道寄存器*
volatile unsigned int  CHSELSEQ1    ≡ (volatile unsigned int*)0x70A3;
    *通道选择域：结果3. 2, 1. 0*
volatile unsigned int  CHSELSEQ2    =(volatile unsigned int*)0x70A4;
    *通道选择域：结果7. 6. 5. 4*/
volatile unsigned int  CHSELSEQ3    =(volatile unsigned int  )0x70A5;
    *通道选择域：结果11, 10.9.8*
volatile unsigned int  CHSELSEQ4    =(volatile unsigned int*)0x70A6;
/*通道选择域：结果15, 14, 13. 12*
volatile unsigned int  AUTO_SEO_SR  =(volatile unsigned int*)0x70A7;
/  自动排序状态寄存器  /
volatile unsigned int  RESULT0      =(volatile unsigned int*)0x70A8;
人  转换结果寄存器0*
volatile unsigned int  RESULT1      =(volatile unsigned int  )0x70A9;
    转换结果寄存器1*
volatile unsigned int  RESULT2      =(volatile unsigned int  )0x70AA;
/关转换结果寄存器2*

```

```

volatile unsigned int    RESULT3
/*转换结果寄存器3*/
volatile unsigned int    *RESULT4
/*转换结果寄存器4*/
volatile unsigned int    *RESULT5
/*转换结果寄存器 5*/
volatile unsigned int*RESULT6
/*转换结果寄存器6*/
volatile unsigned int*RESULT7
/*转换结果寄存器7*/
volatile unsigned int *RESULT8
/*转换结果寄存器8*/
volatile unsigned int*RESULT9
/*转换结果寄存器9*/
volatile unsigned int*RESULT10
/*转换结果寄存器 10*/
volatile unsigned int *RESULT11
/*转换结果寄存器11*/
volatile unsigned int*RESULT12
/*转换结果寄存器12*/
volatile unsigned int*RESULT13
/*转换结果寄存器13*/
volatile unsigned int*RESULT14
/*转换结果寄存器14*/
volatile unsigned int*RESULT15
/*转换结果寄存器15*/
volatile unsigned int *CALIBRATION
/*校准寄存器*/
/*局部控制网络(CAN)寄存器~~~~~*/
volatile unsigned int *CANMDER
/*邮包方向/使能寄存器*/
volatile unsigned int *CANTCR
/*发送控制寄存器 */
volatile unsigned int *CANRCR
/*接收控制寄存器*/
volatile unsigned int *CANMCR
/*主动控制寄存器*/
volatile unsigned int * CANBCR2
/*位配置寄存器 2*/

```

```

=(volatile unsigned int*)0x70AB;

=(volatile unsigned int *)0x70AC;

=(volatile unsigned int*)0x70AD;

=(volatile unsigned int*)0x70AE;

=(volatile unsigned int*)0x70AE;

=(volatile unsigned int*)0x70B0;

=(volatile unsigned int *)0x70B1;

=(volatile unsigned int *)0x70B2;

=(volatile unsigned int *)0x70B3;

=(volatile unsigned int*)0x70B4;

=(volatile unsigned int*)0x70B5;

=(volatile unsigned int*)0x70B6;

=(volatile unsigned int *)0x70B7;

=(volatile unsigned int*)0x70B8;

=(volatile unsigned int *)0x7100;

=(volatile unsigned int*)0x7101;

=(volatile unsigned int*)0x7102;

=(volatile unsigned int*)0x7103;

=(volatile unsigned int*)0x7104;

```

```

volatile unsigned int    CANBCR1    =(volatile unsigned int*)0x7105;
/* 位配置寄存器1*
volatile unsigned int    CANESR     =(volatile unsigned int )0x7106;
    错误状态寄存器
volatile unsigned int    CANGSR     =(volatile unsigned int )0x7107;
/*全局状态寄存器
volatile unsigned int    *CANCEC    =(volatile unsigned int*)0x7108;
/*关CAN 错误计数寄存器*/
volatile unsigned int    *CANIFR    =(volatile unsigned int*)0x7109;
/*CAN 中断标志寄存器/
volatile unsigned int    CANIMR     =(volatile unsigned int )0x710A;
    CAN 中断屏蔽寄存器*
volatile unsigned int    CANLAMOH   =(volatile unsigned int )0x710B;
/*关局部接收屏蔽寄存器0 高位 *
volatile unsigned int    CANLAMOL   =(volatile unsigned int *)0x710C;
    *局部接收屏蔽寄存器0 低位*/
volatile unsigned int    CANLAM1H   =(volatile unsigned int*)0x710D;
/*关局部接收屏蔽寄存器1 高位 */
volatile unsigned int    CANLAMIL   =(volatile unsigned int*)0x710E;
    *局部接收屏蔽寄存器1 低位 *
volatile unsigned int    CANMSGIDOL (volatile unsigned int *)0x7200;
/*对于邮包0的信息标志符低位*
volatile unsigned int    CANMSGIDOH =(volatile unsigned int*)0x7201;
/*对于邮包0的信息标志符高位*/
volatile unsigned int    CANMSGCTRL0 =(volatile unsigned int*)0x7202;
    关对于邮包0的信息控制域*
volatile unsigned int    CANMBXOA    三 (volatile unsigned int )0x7204;
/*邮包0 A
volatile unsigned int    CANMBXOB    =(volatile unsigned int )0x7205;
/*邮包0 B
volatile unsigned int    CANMBXOC    =(volatile unsigned int )0x7206;
/*邮包0 C*
volatile unsigned int    CANMBXOD    =(volatile unsigned int )0x7207;
    *邮包0 D*
volatile unsigned int    CANMSGID1L  =(volatile unsigned int )0x7208;
/*对于邮包1的信息标志符低位*/
volatile unsigned int    CANMSGID1H  =(volatile unsigned int )0x7209;
/*对于邮包1的信息标志符高位 /
volatile unsigned int    *CANMSGCTRL1 =(volatile unsigned int*)0x720A;

```



```

/*对于邮包1的信息控制域*/
volatile unsigned int *CANMBX1A      =(volatile unsigned int*)0x720C;
/* 邮包 1 A*/
volatile unsigned int *CANMBX1B      =(volatile unsigned int*)0x720D;
/*邮包1 B*/
volatile unsigned int *CANMBX1C      =(volatile unsigned int*)0x720E;
/* 邮包 1C*/
volatile unsigned int *CANMBX1D      =(volatile unsigned int*)0x720F;
/*邮包1 D*/
volatile unsigned int *CANMSGID2L     =(volatile unsigned int*)0x7210;
/*对于邮包2的信息标志符低位*/
volatile unsigned int *CANMSGID2H     =(volatile unsigned int*)0x7211;
/*对于邮包2的信息标志符高位*/
volatile unsigned int *CANMSGCTRL2    =(volatile unsigned int*)0x7212;
/*对于邮包2的信息控制域*/
volatile unsigned int *CANMBX2A      =(volatile unsigned int *)0x7214;
/* 邮包 2 A*/
volatile unsigned int CANMBX2B       =(volatile unsigned int*)0x7215;
/* 邮包 2 B*/
volatile unsigned int *CANMBX2C      =(volatile unsigned int*)0x7216;
/* 邮包 2C*/
volatile unsigned int *CANMBX2D      =(volatile unsigned int *)0x7217;
/* 邮包 2D*/
volatile unsigned int *CANMSGID3L     =(volatile unsigned int*)0x7218;
/*对于邮包3的信息标志符低位*/
volatile unsigned int *CANMSGID3H     =(volatile unsigned int*)0x7219;
/*对于邮包3的信息标志符高位*/
volatile unsigned int *CANMSGCTRL3    =(volatile unsigned int*)0x721A;
/*对于邮包3的信息控制域*/
volatile unsigned int *CANMBX3A      =(volatile unsigned int*)0x721C;
/* 邮包 3 A*/
volatile unsigned int *CANMBX3B      =(volatile unsigned int*)0x721D;
/* 邮包 3 B*/
volatile unsigned int *CANMBX3C      =(volatile unsigned int*)0x721E;
/* 邮包 3 C*/
volatile unsigned int *CANMBX3D      =(volatile unsigned int*)0x721F;
/* 邮包 3 D*/
volatile unsigned int *CANMSGID4L     =(volatile unsigned int *)0x7220;
/*对于邮包4的信息标志符低位*/

```

```

volatile unsigned int CANMSGID4H =(volatile unsigned int*)0x7221;
/关对于邮包4的信息标志符高位*/
volatile unsigned int CANMSGCTRL4 =(volatile unsigned int )0x7222;
    对于邮包4的信息控制域*
volatile unsigned int CANHBX4A =(volatile unsigned int 女 ) 0x7224;
/* 邮包 4 A*7
volatile unsigned int CANMBX4B =(volatile unsigned int*)0x7225;
/* 邮包 4 B*/
volatile unsigned int CANMBX4C =(volatile unsigned int*)0x7226;
/ 邮包 4 C+/
volatile unsigned int CANMBX4D =(volatile unsigned int )0x7227;
    * 邮包 4 D*
volatile unsigned int CANMSGID5L =(volatile unsigned int )0x7228;
关对于邮包5的信息标志符低位*
volatile unsigned int *CANMSGID5H =(volatile unsigned int *)0x7229;
    对于邮包5的信息标志符高位*
volatile unsigned int CANMSGCTRL5 =(volatile unsigned int )0x722A;
/关对于邮包5的信息控制域*/
volatile unsigned int CANMBX5A =(volatile unsigned int )0x722C;
/* 邮包 5 A*
volatile unsigned int CANNBX5B (volatile unsigned int *)0x722D;
/各邮包 5B*
volatile unsigned int CANMBX5C =(volatile unsigned int*)0x722E;
    邮包 5C/
volatile unsigned int CANMBX5D =(volatile unsigned int )0x722F;
    关邮包5 Dn
/*事件管理器(EV)/ 事件管理器A(EVA)寄存器
volatile unsigned int GPTCONA =(volatile unsigned int*)0x7400;
    *通用定时器控制寄存器
volatile unsigned int T1CNT =(volatile unsigned int*)0x7401;
/*T1 计数器 *
volatile unsigned int T1CHP =(volatile unsigned int )0x7402;
/ T1 比较值寄存器*
volatile unsigned int T1PER =(volatile unsigned int*)0x7403;
人* T1 周期寄存器
volatile unsigned int T1CON =(volatile unsigned int )0x7404;
    T1 控制寄存器*
volatile unsigned int T2CNT =(volatile unsigned int )0x7405;
/关T2 计数器*

```



```
volatile unsigned int *T2CMP
/*T2 比较值寄存器*/
volatile unsigned int *T2PER
/*T2 周期寄存器*/
volatile unsigned int *T2CON
/*T2 控制寄存器*/
volatile unsigned int *COMCONA
/*比较控制寄存器*/
volatile unsigned int *ACTRA
/*比较输出动作控制寄存器*/
volatile unsigned int *DBTCONA
/*死区控制寄存器*/
volatile unsigned int *CMPR1
/*比较值1*/
volatile unsigned int *CMPR2
/*比较值2*/
volatile unsigned int *CMPR3
/*比较值3*/
volatile unsigned int *CAPCONA
/*捕捉控制寄存器*/
volatile unsigned int *CAPFIFOA
/*捕捉FIFOA 状态寄存器*/
volatile unsigned int *CAP1FIFO
/*捕捉1 FIFO 顶部*/
volatile unsigned int *CAP2FIFO
/*捕捉2 FIFO顶部*/
volatile unsigned int *CAP3FIFO
/*捕捉3 FIFO 顶部*/
volatile unsigned int *CAP1FBOT
/*捕捉1 FIFO 底部*/
volatile unsigned int *CAP2FBOT
/*捕捉2 FIFO 底部*/
volatile unsigned int *CAP3FBOT
/*捕捉2 FIFO 底部*/
volatile unsigned int *EVAIMRA
/*EVA 中断屏蔽寄存器 A*/
volatile unsigned int *EVAIMRB
/*EVA 中断屏蔽寄存器 B*/
volatile unsigned int *EVAIMRC
```

```
=(volatile unsigned int*)0x7406;
=(volatile unsigned int *)0x7407;
=(volatile unsigned int *)0x7408;
=(volatile unsigned int*)0x7411;
=(volatile unsigned int *)0x7413;
=(volatile unsigned int *)0x7415;
=(volatile unsigned int*)0x7417;
=(volatile unsigned int *)0x7418;
=(volatile unsigned int*)0x7419;
=(volatile unsigned int*)0x7420;
=(volatile unsigned int*)0x7422;
=(volatile unsigned int*)0x7423;
=(volatile unsigned int *)0x7424;
=(volatile unsigned int*)0x7425;
=(volatile unsigned int*)0x7427;
=(volatile unsigned int*)0x7428;
=(volatile unsigned int*)0x7429;
=(volatile unsigned int*)0x742C;
=(volatile unsigned int *)0x742D;
=(volatile unsigned int*)0x742E;
```



```

/*EVA中断屏蔽寄存器 C*
volatile unsigned int      EVAIFRA      =(volatile unsigned int*)0x742F;
/*EVA 中断标志寄存器 A*
volatile unsigned int      EVAIFRB      =(volatile unsigned int*)0x7430;
/*EVA 中断标志寄存器 B
volatile unsigned int      EVAIFRC      =(volatile unsigned int*)0x7431;
/*EVA中断标志寄存器 C*
/*事件管理器B(EVB)寄存器 ~~~~~
volatile unsigned int      *GPTCONB      =(volatile unsigned int*)0x7500;
/* 通用定时器控制寄存器*/
volatile unsigned int      T3CNT         =(volatile unsigned int )0x7501;
/*T3 计 数 器
volatile unsigned int      T3CNP         =(volatile unsigned int )0x7502;
/*T3比较值寄存器
volatile unsigned int      T3PER         =(volatile unsigned int*)0x7503;
/*T3 周 期 寄 存 器 *
volatile unsigned int      T3CON         =(volatile unsigned int*)0x7504;
/*×T3 控制寄存器*
volatile unsigned int      首 T4CNT      =(volatile unsigned int 首 ) 0x7505;
/*T4 计 数 器 *
volatile unsigned int      T4CMP         (volatile unsigned int *)0x7506;
/*KT 4 比较值寄存器*/
volatile unsigned int      关 T4PER      =(volatile unsigned int*)0x7507;
/* T4 周期寄存器*
volatile unsigned int      T4CON         =(volatile unsigned int )0x7508;
/*T4 控制寄存器*
volatile unsigned int      COMCONB       三 (volatile unsigned int )0x7511;
/*比较控制寄存器*7
volatile unsigned int      ACTRB         =(volatile unsigned int *)0x7513;
/* 比较输出动作控制寄存器*
volatile unsigned int      关 DBTCNB     =(volatile unsigned int*)0x7515;
/*死区控制寄存器
volatile unsigned int      苦 CMPR4      =(volatile unsigned int )0x7517;
/* 比较值 4 *
volatile unsigned int      CMPR5         =(volatile unsigned int )0x7518;
/* 比较值5*/
volatile unsigned int      *CMPR6        =(volatile unsigned int )0x7519;
/*比较值6*/
volatile unsigned int      CAPCONB       =(volatile unsigned int*)0x7520;

```

```

/*捕捉控制寄存器*/
volatile unsigned int *CAPFIFOB =(volatile unsigned int*)0x7522;
/*捕捉FIFO A 状态寄存器*/
volatile unsigned int *CAP4FIFO =(volatile unsigned int*)0x7523;
/*捕捉4 FIFO 顶部*/
volatile unsigned int *CAP5FIFO =(volatile unsigned int*)0x7524;
/*捕捉5 FIFO 顶部*/
volatile unsigned int *CAP6FIFO =(volatile unsigned int*)0x7525;
/*捕捉6 FIFO 顶部*/
volatile unsigned int CAP4FBOT =(volatile unsigned int*)0x7527;
/*捕捉4 FIFO底部*/
volatile unsigned int *CAP5FBOT =(volatile unsigned int*)0x7528;
/*捕捉5 FIFO底部*/
volatile unsigned int *CAP6FBOT =(volatile unsigned int*)0x7529;
/*捕捉6 FIFO底部*/
volatile unsigned int *EVBIMRA =(volatile unsigned int*)0x752C;
/*EVB 中断屏蔽寄存器 A*/
volatile unsigned int *EVBIMRB =(volatile unsigned int*)0x752D;
/*EVB中断屏蔽寄存器 B*/
volatile unsigned int *EVBIMRC =(volatile unsigned int*)0x752E;
/*EVB 中断屏蔽寄存器 C*/
volatile unsigned int *EVBIFRA =(volatile unsigned int*)0x752F;
/*EVB 中断标志寄存器 A*/
volatile unsigned int *EVBIFRB =(volatile unsigned int*)0x7530;
/*EVB中断标志寄存器 B*/
volatile unsigned int *EVBIFRC =(volatile unsigned int*)0x7531;
/*EVB 中断标志寄存器 C*/
/*常量的定义 数据空间~~~~~*/
#define B2_SADDR 0x0060 /*块B2的开始地址*/
#define B2EADDR 0x007F /*块B2的结束地址*/
#define B0_SADDR 0x0200 /*块B0的开始地址*/
#define B0_EADDR 0x02FF /*块B0的结束地址*/
#define B1_SADDR 0x0300 /*块B1的开始地址*/
#define B1_EADDR 0x03FF /*块B1的结束地址*/
#define SARAM 0x0800 /*SARAM 的开始地址(0800h~0FFFh)*/
#define EXTDATA 0x8000 /*外部的数据空间的开始地址*/
/*经常用到的数据页~~~~~*/
#define DP_B2 0 /*页0 数据空间*/
#define DP_B01 4 /*页4 B0(200h/80h)*/

```

```
#define DP_B02 5 ×页5 B0(280h/80h)
#define DP B11 6 /*页6 B1(300h/80h)
#define DP B12 7 / 页7 B1(380h/80h) /
#define DP_SARAM1 16 页16 SARAM(800h/80h)
#define DP_PF1 224 页1外设帧文件(7000h/80h)(0xE0)*/
#define DP_PF2 225 /*页2外设帧文件(7080h/80h)(0xE1)*
#define DP_PF3 226 *页3外设帧文件(7100h/80h)(0xE2)*
#define DP_PF4 227 / 页4外设帧文件(7180h/80h)(0xE3)
#define DP_PF5 228 /*页5外设帧文件(7200h/80h)(0xE4)*
#define DP_EVA 232 /EVA 寄存器文件(7400h/80h)(0xE8) /
#define DP_EVB 234 /*EVB 寄存器文件(7500h/80h)(0xEA)*
#define DP_EXT1 256 外部存储器的第一块(8000h/80h)*/
/关位测试指令的位代码(BIT)
#define BIT15 0 /位代码15*
#define BIT14 1 *位代码14*
#define BIT13 2 位代码13*
#define BIT12 3 位代码12*
#define BIT11 4 /位代码11*
#define BIT10 5 /位代码10*
#define BIT9 6 位代码9*
#define BIT8 7 位代码8*
#define BIT7 8 位代码7*/
#define BIT6 9 *位代码6*
#define BIT5 10 位代码5*/
#define BIT4 11 位代码4*
#define BIT3 12 位代码3*
#define BIT2 13 位代码2*/
#define BIT1 14 /位代码1*
#define BIT0 15 位代码0*
/*用SBITO和SBIT1宏屏蔽位
#define B15_MSK 0x8000 *位屏蔽15*
#define B14_MSK 0x4000 位屏蔽14#
#define B13_MSK 0x2000 位屏蔽13*
#define B12_MSK 0x1000 *位屏蔽12*/
#define B11_MSK 0x0800 /位屏蔽11*左
#define B10_MSK 0x0400 ×位屏蔽10*/
#define B9_MSK 0x0200 位屏蔽9*
#define B8_MSK 0x0100 位屏蔽8*/
#define B7_MSK 0x0080 /位屏蔽7*
```



```
#define      B6_MSK      0x0040      /*位屏蔽6*/
#define      B5_MSK      0x0020      /*位屏蔽5*/
#define      B4_MSK      0x0010      /*位屏蔽4*/
#define      B3_MSK      0x0008      /*位屏蔽3*/
#define      B2_MSK      0x0004      /*位屏蔽2*/
#define      B1_MSK      0x0002      /*位屏蔽1*/
#define      B0_MSK      0x0001      /*位屏蔽0*/

/*看门狗定时器复位字符串      关
#define      wd_rst_1      0x0055
#define      wd_rst_2      0x00AA
/×以下为用户自行定义的变量~~~~~
```

5.3.3 命令文件.CMD

在给出命令文件示例之前，首先介绍该文件中常用到的几个伪指令。

- .cinit 程序编译、链接时，该标号标示部分存放已明确初始化的全局变量和静态变量；
- .const 程序编译、链接时，该标号标示部分存放已明确初始化的字符串常量、全局常量和静态常量；
- .switch 程序编译、链接时，该标号标示部分存放大型的 switch 语句的跳转表；
- .text 程序编译、链接时，该标号标示部分存放可执行代码和浮点数常量；
- .bss 该标号标示部分用于存放没有初始化的全局变量和静态变量；
- .stack 该标号定义软件堆栈。

命令文件名的后缀为.CMD。该文件实现对程序存储器空间和数据存储器空间的分配。其常用的伪指令有 MEMORY 和 SECTIONS。

当然用户也可用C 语言编程实现所需的功能。

MEMORY 伪指令用来标示实际存在目标系统中且可被使用的存储器范围。每个存储器范围具有名字、起始地址和长度。MEMORY 伪指令的一般语法为：

```
MEMORY
{
    PAGE0:      name1[(attr)];      origin=constant,length=constant;
    PAGEn:      namen[(attr)];      origin=constant,length=constant;
}
```

PAGE 标示存储器空间。用户可以规定多达255数据页。通常PAGE0 规定程序存储器，PAGE1 规定数据存储器。

name 命名存储器范围。存储器可以是1~8个字符。在不同页上的存储器范围可以具有相同的名字，但是在一页之内，所有的存储器范围必须具有唯一的名字

且必须不重叠。

attr 规定与已命名范围有关的1~4个属性。未规定属性的存储器具有所有4个属性。有效的4个属性包括：

R 表示存储器可以读出；

W 表示存储器可以写入；

X 表示存储器含有可执行代码；

I 表示存储器可以初始化。

origin 表示存储器范围的起始地址。

length 表示存储器范围的长度。

SECTIONS 伪指令的作用是：描述输入段怎样被组合到输出段内；在可执行程序内定义输出段；规定在存储器内何处放置输出段；允许重命名输出段。SECTIONS 伪指令的一般语法是：

SECTIONS

name :Lproperty,property,property]

name :[property ,property ,property .]

name :[property .property .property .,]

K

每一个以 name (名字)开始的段的规格说明定义了一个输出段。在段名之后是特性列表，定义段的内容以及它们是怎样被分配的。特性可以用逗号来分开。段可能具有的特性如下：

(1) 装载位置，规定段将被装载在存储器内何处；

(2) 运行位置，定义段将在存储器内何处运行；

(3) 输入段，定义组成输出段的输入段；

(4) 段类型，定义特定段类型的标志；

(5) 填充值，定义用于填充未初始化空位的数值。

命令文件. CMD 示例如下：

- stack 40

苦 */

命令文件——存储空间 F2407 */

* — */

MEMORY

{

PAGE 0: VFCS : origin =0 h , length =40 h * 程 序 复 位 *

VE: argi: = 40h , length =70 h *外围模块中断向量x/

```

        PROG      :origin= 0b0h,length=      7F50h    /* 在片 FLASH*/
PAGE 1: MMRS      :origin=      0h,length      =      05Fh    /*MMRS*/
        B2        :origin          =0060h,length=      020h    /*DARAM B2 块 * /
        B0        :origin          =0200h,length=      100h    /*DARAM  B0块 * /
        B1        :origin=0300h,length=          100h    /*DARAM B1块 * /
        SARAM:origin=0800h,length=          0800h    *SARAM  块 * /
        EXT      :origin          =8000h,length=      8000h    /* 外部存储器*/
    }
/*-----*/
/*SECTIONS ALLOCATION                                */
/*-----*/
SECTIONS
{
    .reset      : <}>VECS      PAGE 0    /*复位中断向量表*/
    .vectors    :  }>VECS      PAGE 0    /*中断向量表 */
    .pvecs      : { }>PVECS    PAGE  0    /*外围模块中断向量表*/
    .text       :  }>PROG      PAGE 0    /* 代 码 */
    .cinit      : { }>PROG      PAGE 0
    .bss        :  }>SARAM      PAGE  1    / * 块 B2*/
    .const      :  }>SARAM      PAGE  1    / * 块 B2*/
    .stack      :  }>B1        PAGE  1    /*堆栈 — 40个单元*/
}

```

该命令文件只是一个实际程序的示例，并不一定都适用于本书的部分应用程序。读者在调试应用程序时，应根据各个程序自身的特点，对本命令文件进行适当的修改，使之成为能够适合该应用程序的命令文件。

例如，如果用户需要从数据空间的地址单元0300h 开始开辟出20h 个存储单元，并命名为 USERBLK，则用户可以在上面的 .CMD 文件中删掉“B1 :origin=0300h,length=100h”一句，加上“USERBLK:origin=0300h,length=20h”一句；如果用户想把该存储单元块用作未初始化变量的空间，则可以在上面 SECTIONS 后面的 {} 中加上 “ .bss : { }>USERBLK PAGE 1”一句。对 .CMD 文件的配置是 DSP 应用程序开发中比较重要的一个步骤，并且 .CMD 文件必须与 .SAM 文件对应。如果不对应，则即使语法完全正确，ASM 程序代码也不会编译通过。在配置 .CMD 文件时，用户必须对 DSP 的数据存储空间、程序存储空间和汇编伪指令十分熟悉。所有的空间分配都必须建立在 DSP 物理结构的基础上，用户不能通过 .CMD 文件虚构出 DSP 在物理结构上不存在或不合法的空间。数据存储空间、程序存储空间的详细资料请参见第2章。汇编伪指令的使用参考 TI 公司的“SPRU018.PDF”文献。

5.3.4 中断向量文件

中断向量文件 `vectors.asm` 定义了系统各种中断服务程序。

有了上述4种基本文件之后，就可以在调试环境中将汇编源程序编译、链接，最后生成可执行文件进行仿真调试。这4种文件的编程示例见本书后面章节的程序例程。

5.3.5 库文件 `rts2xx.lib`

`rts2xx.lib` 库文件可在 `C:\CCStudio_v3.1\C2400\cgtools\lib` 下找到，不得修改，完全复制到需要的地方。

`rts2xx.lib` 包含如下内容：

>ANSIC 标准库；

系统启动程序 `c__int0`；

允许C 访问特殊指令的函数和宏。

5.4 利用 CCS 编译器调试程序的简单步骤

在 CCS 集成调试环境下新建一个项目，对该项目进行调试，或者要使用 TMS320C2x C2xx/C5xC 编译器开发一个 DSP 的 C 语言应用程序，须按以下简单步骤进行。

5.4.1 4种类型的文件

开发一个 DSP 的 C 语言应用程序，需要以下4种类型的文件：C 语言文件、汇编语言文件、头文件和命令文件。C 语言文件是必须的。汇编语言文件则根据实际情况而定。在C 语言程序中如何嵌入汇编见第6章。就一般情况而言，编写复位初始化程序和中断向量表时，需要用到汇编语言，因此C 语言和汇编语言混合编程是很有必要的。头文件定义 DSP 内部寄存器地址分配，编写一次后可被其他程序反复使用。命令文件主要定义堆栈、程序空间分配和数据空间分配等。这些文件可以用 Windows 操作系统中的“记事本”编写，也可以用其他一些常用的文本编辑器编写，只须存储成相应的格式：C 语言文件为“.C”格式，汇编语言文件为“.ASM”格式，头文件为“.H”格式，命令文件为“.CMD”格式。注意，这4种类型的文件和1个库文件必须存储在同一个文件夹中。

5.4.2 建立一个项目

一个项目的信息存储在一个项目文件中 (*.pjt)。当项目中建立多个文件时，每个文件名都必须是唯一的。安装好 CCS3.1 调试仿真软件后，在安装目录下会出现一个文件夹 `C:\CCStudio_y3.1\MyProjects`，用户调试的项目可以放在该文件夹中。当然用户也可以将项目

放在其他子目录下。本例将新建的项目放在 My Projects 文件夹下。

选择菜单命令 **Project→New**，在弹出的文件选择对话框中选择需要保存项目文件夹子目录，输入项目文件名，再单击“保存”按钮退出。注意：由于不同项目使用的源程序文件和头文件不完全相同，所以最好每个项目选择一个文件夹来保存自己的源程序文件和头文件，以便把不同的项目区分开来。在项目管理窗口中，可以察看建立的项目以及项目所包含的源程序文件和目标文件等。

DSP 的 C 语言程序开发也采用项目管理的方法。建立一个项目的步骤如下：

(1) 单击桌面上的 CCSstudio 3.1 图标，出现如图 5.5 所示窗口。

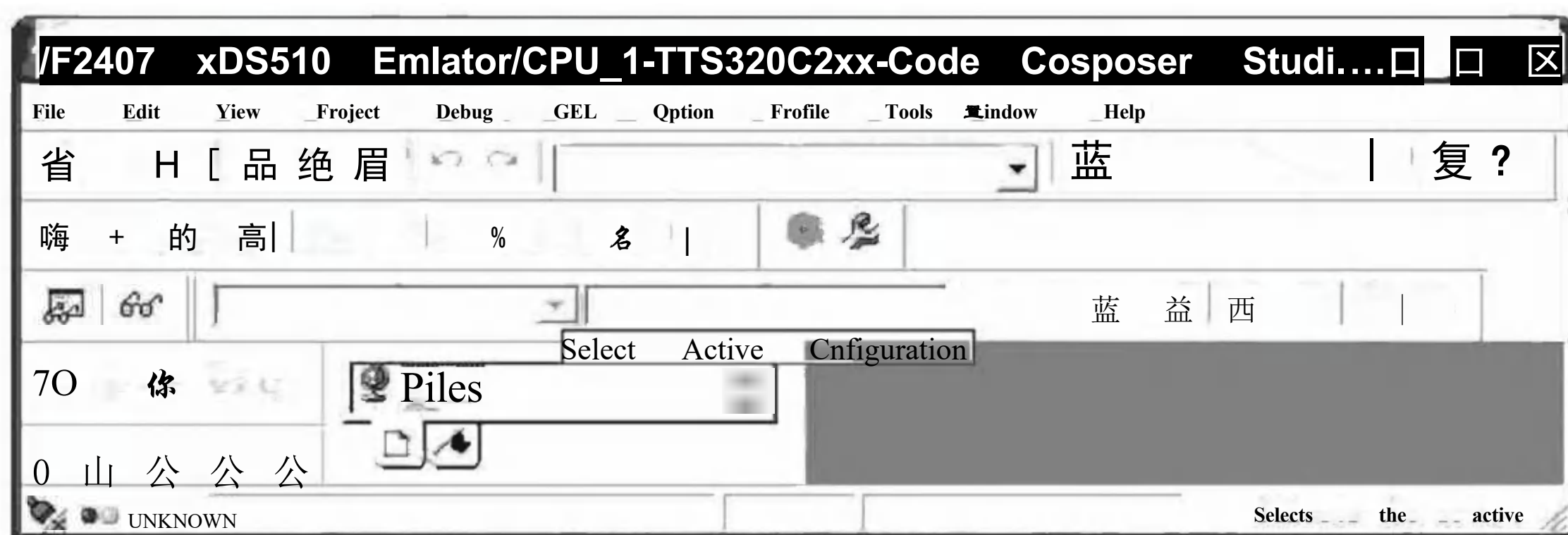


图 5.5 CCSstudio 3.1 应用程序窗口

(2) 选择 **Project→New** 选项，出现如图 5.6 所示对话框。给该项目取一个合适的名称，并存于相应的子目录中，即该子目录必须包含前面所述的 4 种类型的文件。

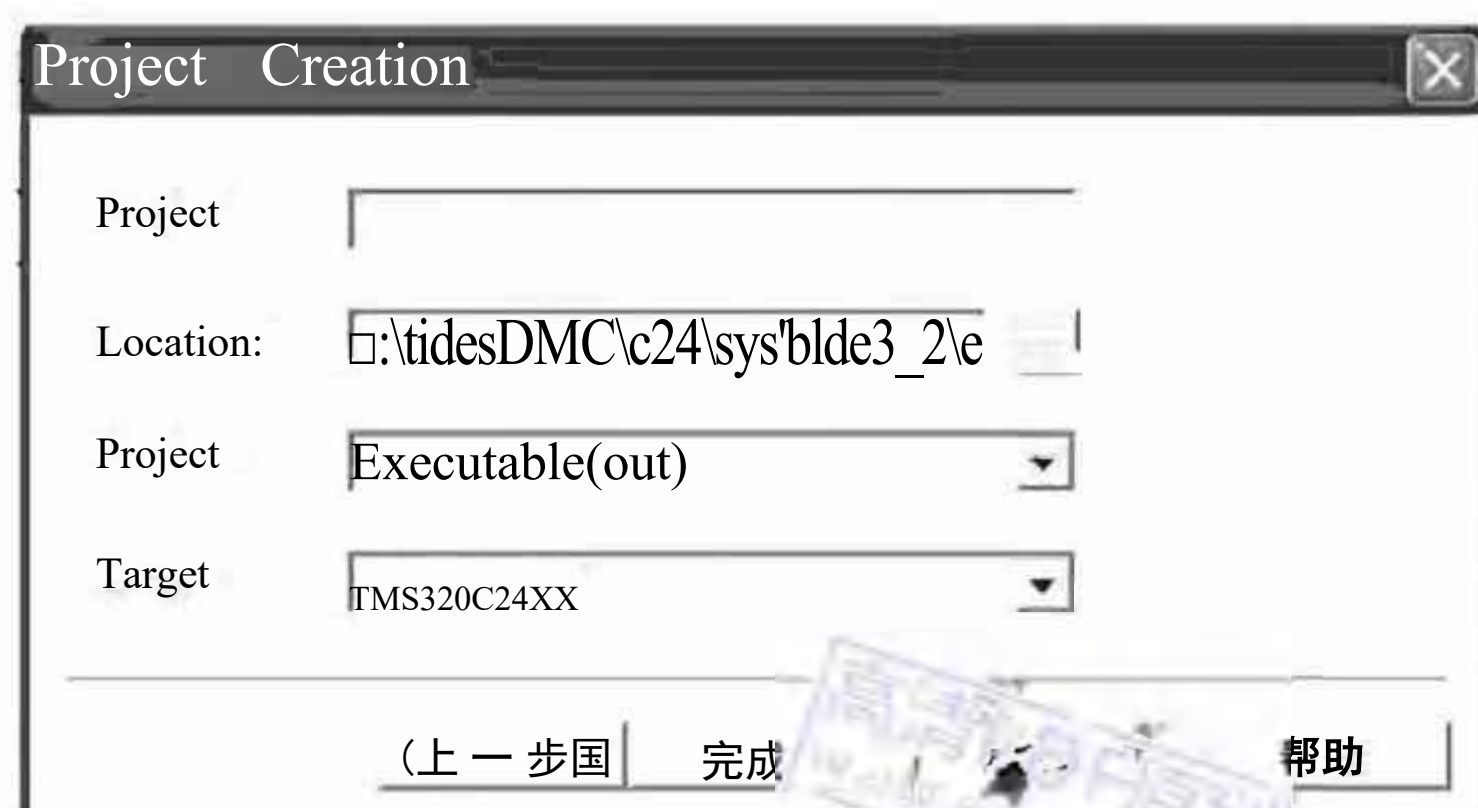


图 5.6 建立新项目对话框

(3) 在图 5.5 中选择 **Project→Add Files to Project** 选项，然后找到前面所述的子目录，将该项目中用到的 *.asm、*.s*、*.cmd、*.o* 和 *.lib 等文件添加到该项目中。如果程序

中调用了一些库函数，比如`_atan`、`_sqrt`、`cos` 和 `_sin` 等，则须将这些函数所在的库文件添加到该项目中。把C 语言文件、汇编语言文件、命令文件和 `rts2xx.lib` 文件添加到当前项目后，头文件不需要用户自己添加，只要头文件和项目在同一个目录下，链接时调试工具将自动识别。

到此为止，就建立了一个项目。

5.4.3 编译环境的简单配置

在图5.5中选择Option→Customize 选项，出现如图5.7所示对话框，选择 Load Program After Build 选项。这样在每次 Build（重建）源程序文件后，如果没有语法错误，则生成的可执行代码将自动下载到DSP 芯片中。

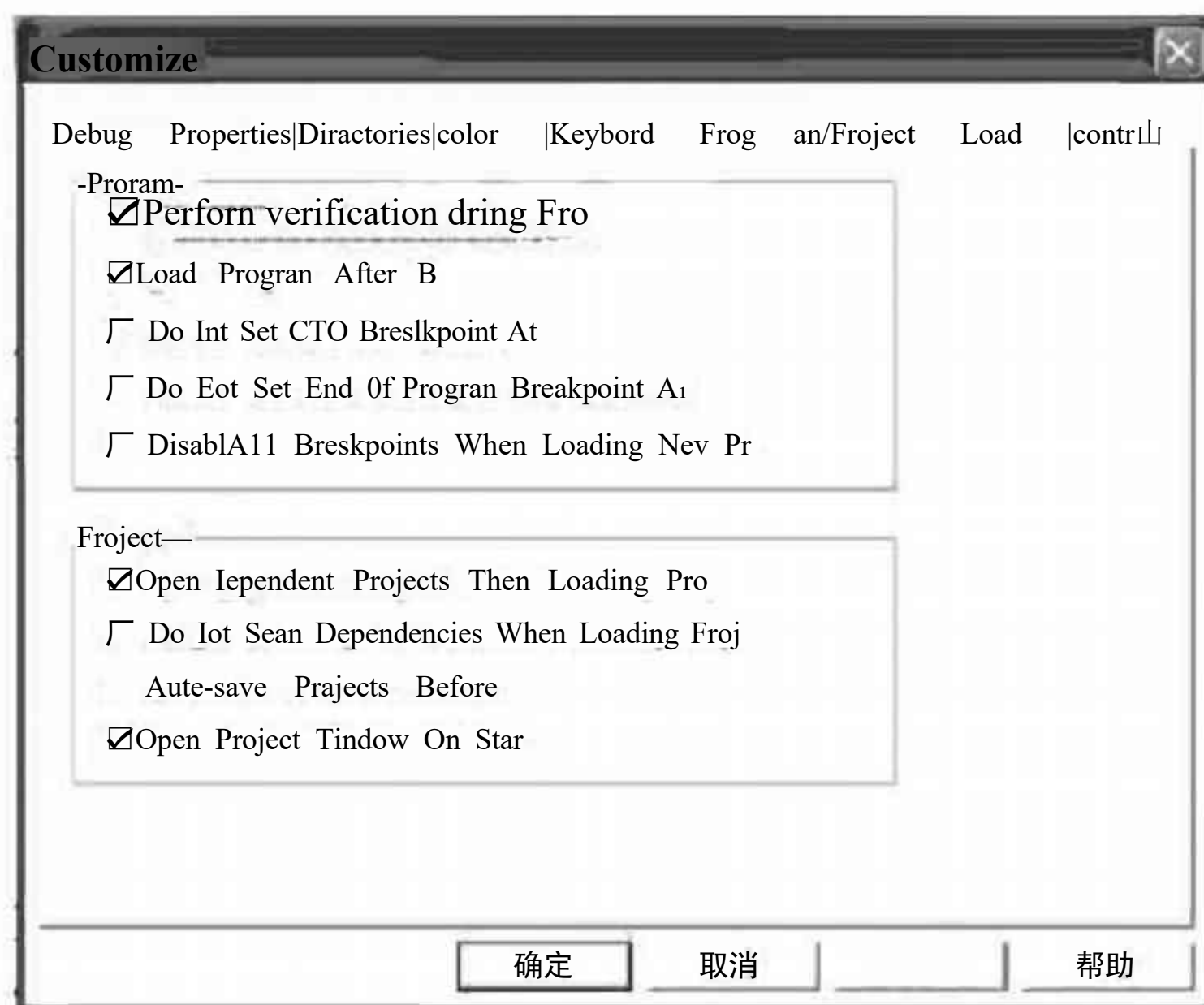


图5.7 编译环境的简单配置对话框

5.4.4 编译源程序文件

在图5.5中选择 Project→Build 选项，就可实现 C 源程序文件的编译。如果没有语法错误，则可直接生成可执行代码并直接下载到DSP 芯片中。如果有语法错误或其他类型的错误，则编译器会给出相应的提示。根据提示改掉错误后，再进行同样的操作，直到没有任何错误为止。

编译和运行程序按以下步骤进行：

(1) 选择 Project→Rebuild All 选项，或者单击工具栏中的按钮。CCS 就会重编译、汇编并且链接项目中所有的文件。如果在汇编、链接过程中没有错误，则生成一个名字与项目名

相同且可下载执行的输出文件.out;如果在编译、链接过程中出现错误,则系统会提示该错误的类型和位置,用户可根据此信息返回源程序中修改。

(2)默认的.out 文件放在项目的调试目录中。要想改变这个目录,可以在CCS 工具栏中选择不同的目录即可实现。



(3)选择File→Load Program选项(下载程序),则调试工具将.out 文件下载到实验开发板上。也可以在CCS 中直接设置“编译完后直接下载”功能,这样每次编译后可以自动完成程序下载。当.out 文件下载之后,即可对源程序进行仿真调试。CCS 的仿真调试环境功能很强大,在 Debug 调试时,用户可根据需要,选择任意一种或多种调试方式,见 Debug 菜单。

(4)选择 View→Mixed Source选项,可直接观察C 源程序代码以及汇编的结果代码。

(5)选择Debug→Go Main 选项可直接开始主程序的执行。执行就暂停在主程序入口并用 标示。

(6)选择 Debug→Run 选项,或者单击运行工具栏中的公按钮可以运行程序。

当用户执行程序很大,又想观察程序执行到某处相应寄存器的值时,可以将 Run 和 Break Points 一起使用。程序运行到断点处自动停止,重新选择 Run 之后又可继续运行。例如,在作 A/D 采样时,用定时器中断启动 A/D 每隔200 ms 采样一次,主程序中等待定时器时间到,循环执行A/D 采样。用户执行程序时只是想知道每次采样结果,而不想进入每一个子程序,也不想对其中的等待过程进行仿真。因此,用户可以在主程序中调 A/D 子程序的下一行设置断点,执行 Run 命令,则每次程序在断点处停止时,都能通过 A/D 采样的结果寄存器看到最近一次A/D 采样的结果。

在调试过程中,对于数据存储器空间中的某些单元可以进行修改,以满足调试的需要。

(7)选择 Debug→StepInto 选项可以进行单步调试。在调试过程中,菜单栏Debug 调试中给出了一些调试选项,用户可以选择单步执行并进入子程序StepInto 来观察程序执行每一步时相应寄存器单元的变化,或者选择单步执行但不进入子程序StepOver 来观察执行子程序的结果,或者采用全速运行 Run 来运行程序等。

(8)选择Debug→Halt 选项,可以暂停正在运行的程序。

(9)可以在 GEL 菜单下查看程序执行过程中寄存器的变化。

CCS 集成开发环境支持硬件在线调试。在调试过程中,通常希望看到某些寄存器和状态位的变化,CCS 集成开发环境支持这种功能。在菜单栏中选择 View→Register 选项,可将 CPU 寄存器和状态寄存器窗口打开。在CPU 寄存器中,用户可观察在仿真过程中累加器、乘

积寄存器、状态寄存器、PC 指针、数据页指针 DP、全局中断屏蔽寄存器、全局中断标志寄存器以及8个辅助寄存器的单元值变化状况。

当用户需要对某一外设进行操作时，常常希望能观察到与该外设有关的寄存器随仿真运行而变化的状况，这时可选择View→Memory 选项。输入该外设所处的地址范围，例如在作A/D 采样时，希望知道采样的结果寄存器中的值，则输入地址为70A0h，选择 page(页面)为 data(数据)页，其他选项采用默认值，则系统将打开一个从70A0h 开始的数据存储器空间，用户就可以很方便地观察 A/D 采样的结果。

当然，有时用户只想观察一个变量或一个表达式的变化情况，则用户可以选择 View→Watch window选项，输入要观察的变量或表达式，调试工具就打开一个 Watch 窗口，显示变量或表达式的值。

调试过程中反复使用(1)～(8)步，直到调试程序达到工程应用的要求为止。

在CCS 集成开发环境中还有许多十分有用的功能，比如可以计算整个程序的实际运行时间等，用户可以通过帮助文件知道这些功能的用法。在此只是简单给出了将汇编文件转换成可执行文件的方法，以及一些基本的调试功能。用户可参见 CCS3.1 的帮助文件，得到比较详细的信息。

第 6 章 C 语言和汇编语言程序混合编程方法及中断处理方法



6.1 C 语言和汇编语言程序混合编程方法

C 语言和汇编语言程序的混合编程有以下2种方法。

6.1.1 在 C 语言中嵌入汇编语言程序

在用C 语言开发DSP 应用程序时，可能会遇到一些对实时性要求较高或是需要对 DSP 的底层资源进行操作的场合。这时用C 语言编写相应的代码就会给开发带来一定的难度，而且对于某些操作，C 语言根本就无法实现(如对 OVM 和 CNF 等位的操作)。这时，就需要在 C 语言中嵌入汇编语言程序。有两种方法可以实现此功能。

1. 在 C 语言中嵌入单条汇编语句

在某些场合，只需要在 C 语言中嵌入一句或几句汇编语句，例如在程序的初始化阶段对 SXM、OVM 及 CNF 等位的配置时。用asm（需要嵌入的汇编语句）的方式就可以实现单条汇编语句的嵌入。下面是一个应用示例。

```
//系统初始化子程序
void initial()
{
    asm("setc    SXM");        //SXM 位置1,抑制符号位扩展
    asm("clrc    OVM");        //OVM 位清0,累加器中结果正常溢出
    asm("clrc    CNE");        //CNF 位清0,B0被配置为数据存储空间
}
```

2. 在 C 语言中嵌入实现某一完整功能的多条汇编语句

在 C 语言中嵌入单条汇编语句时，虽然操作简便，但由于这种方式破坏了C 语言的完整性，因此只提倡在程序开始的系统初始化部分少量采用。而在 C 语 言 中 嵌 入 实 现 某 一 完 整 功 能的多条汇编语句时，就不提倡采用这种方法。

在C 语言调用C 语言子程序时，其调用规范是完全一致的。即对应的汇编语句，在子程序的入口处需要保护一些特定的寄存器值，即保护C 语言在调用子程序前的运行现场；在子程序的出口处，需要恢复前面保护的寄存器值，以恢复现场。

如果把需要嵌入的多条汇编语句编制成一个子程序，并在其入口和出口处遵循C 语言的调用规范，进行一些必要的操作，则可以方便地在C 语言中调用汇编语言程序。

在C 语言中调用汇编语言程序编写的子程序，主要应弄清楚堆栈的使用情况，特别注意这里使用的堆栈不是DSP 的硬件堆栈，而是.CMD 文件定义的软堆栈。图6. 1详细地表明了C 语言调用子程序时堆栈的使用情况。其表示的是一种较复杂的调用情况：主函数需要向被调函数传递若干个参数，被调函数使用了若干个局部变量。图6. 1还描述了被调函数局部数据结构的分配情况(局部数据结构用于存储局部变量)。如果主函数不需要向被调函数传递参数且被调函数没有局部变量，则不需要分配局部数据结构。

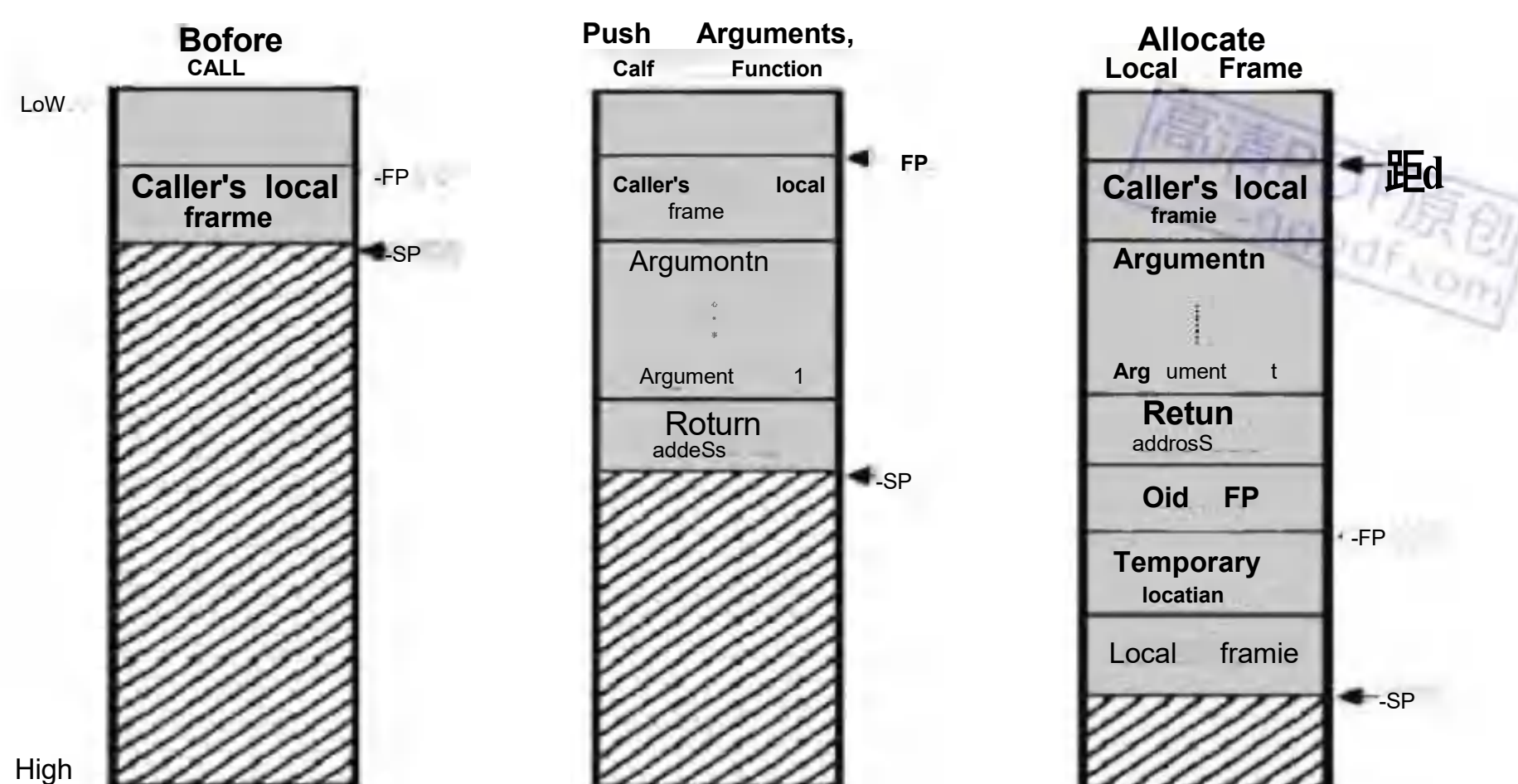


图6.1 在进行函数调用时堆栈的使用情况

C 语言调用子程序时的规范如下：

1) 主程序需要进行的操作

主程序调用另外一个子程序时，需要进行如下操作。特别须注意 ARP 已经由编译器自动设置为 AR1。

① 主程序把需要向子程序传递的参数按反序压入堆栈(最右边的参数最先被压入堆栈，最左边的参数最后被压入堆栈)；

② 主程序调用子程序；

③ 在子程序返回前，ARP 已经被设置为 AR1；

④ 当子程序调用完毕后，主程序要弹出先前压入堆栈的传递参数，此操作通过语句

“SBRK n”(n是主程序向子程序传递的参数的个数)实现。

②) 子程序需要进行的操作

一个子程序被调用时，需要进行如下操作。在子程序的入口处，假设ARP 已经被设置为AR1(这是由C 编译器自动完成的)。

- ① 从硬件堆栈中弹出返回地址，然后把它压入软件堆栈；
- ② 把主程序的数据结构指针FP 压入堆栈；
- ③ 如果子程序改变了AR6 或者 AR7, 那么也需要把它们压入堆栈；
- ④ 分配局部数据结构；
- ⑤ 执行子程序的实际任务代码；
- ⑥ 如果子程序有返回值，则把此返回值放入累加器中；
- ⑦ 设置 ARP 为 AR₁；
- ⑧ 解除对局部数据结构；
- ⑨ 如果 AR6 和 AR7 曾经被保存过，则从软件堆栈恢复它们的值；
- ⑩ 从软件堆栈恢复 FP；
- ⑪ 把软件堆栈中存储的返回地址压入硬件堆栈；
- ⑫ 返回。

如果是C 语言主程序调用其子程序，则上述的调用规范是C 编译器在生成汇编语言程序代码时自动完成的；如果是C 语言主程序调用汇编语言编写的子程序，则子程序的编写必须遵循前述“子程序需要进行的操作”规范，而“主函数需要进行的操作”则由C 编译器自动完成。

6.1.2 分开编写C 语言和汇编语言程序

1. 分离的C 语言程序和汇编语言程序接口

这是一种常用的C 语言程序和汇编语言程序接口。采用此方法需要注意的是：在编写汇编语言程序时，必须遵守有关的调用规则和寄存器规则，这样不会影响C 语言程序的运行环境。C 语言程序既可调用汇编语言程序，也可访问汇编语言程序中定义的变量。同样，汇编语言程序也可调用C 语言函数或访问C 语言程序中定义的变量。

2. 在C 语言程序中访问汇编语言程序的变量

该方式需要根据变量或常数定义的方式用以下3种方法来实现这种操作。

(1) 汇编变量在.bss 段中定义。访问在.bss 中定义的变量按以下步骤就可进行正常访问：

- ① 采用.bss 命令定义变量；
- ② 采用.global命令定义变量；

- ③ 在变量名前加一下划线“_”;
- ④ 在 C 语言程序中将变量说明为外部变量。

(2) 变量不在 .bss 段中定义。访问不在 .bss 中定义的变量按以下步骤就可进行正常访问:

- ① 在汇编语言程序中定义一个常数,并定义一个指向该表的指针;
- ② 在 C 语言程序访问该表时,另外说明一个指向该表的指针。

(3) 用 .set 和 .global 命令定义全局常数。从 C 语言程序中访问在汇编语言程序中用 .set 和 .global 命令定义的常数,按以下步骤就可以进行正常访问:

- ① 在汇编语言程序中将常数(符号)定义为全局常数;
- ② 在 C 语言程序中访问汇编语言程序中的常数时,应在常数名之前加一个地址操作符“&”。

3. 在汇编语言程序中访问 C 语言程序变量

从汇编语言程序中访问 C 语言程序变量按以下步骤就可进行正常访问:

- ① 在 C 语言程序中定义变量(如 x);
- ② 在汇编语言程序中使用时前面加下划线“_”(如x)。

C 语言与汇编语言混合编程的实例可参见第17章三相感应电动机恒压频比控制系统所附光盘中的程序,其中有许多 C 语言程序和汇编程序,均是调试通过的应用程序,完好地解决了混合编程问题,可以将程序修改后,应用到自己的控制程序中,也可以直接使用这些程序。

6.2 用 C 语言处理 DSP 中断

本节将详述F240x DSP的可屏蔽中断的特点及其响应过程,介绍可屏蔽中断程序用C 语言实现的两种方法,并给出实现可屏蔽中断的实例。

F240x 内核提供了一个不可屏蔽的中断 NMI 和6个按优先级获得服务的可屏蔽中断 INT1~INT6。而这6个中断的每一个都可被很多外设中断请求共享,见图2.9外设中断扩展模块图。

F240x 系列DSP 通过中断系统中的一个两级中断来扩展系统可响应的中断个数,因此 DSP 的中断请求/应答硬件逻辑和中断服务程序软件都是一个两级的层次结构。

在底层中断,从几个外设来的外设中断请求(PIRQ) 在中断控制器处相“或”产生一个到 CPU 的中断请求(INTn)。这就是内核级的中断请求。在外设配置寄存器中对每一个产生外设中断请求的事件都有一个中断使能位和中断标志位。如果一个引起中断的外设事件发生且相应的中断使能位置1,则会产生一个从外设到中断控制器的中断请求。此中断请求反映了外设中断标志位的状态和中断使能位的状态。当中断标志位清0时,中断请求也清0。

对某些要设置中断优先级的外设事件，当这类事件发生时，其中断优先级的值被送到中断控制器，而中断请求也是保持到中断应答或者软件将其清0为止。

在高层中断中，被相“或”的多个外设中断请求产生一个到 CPU 的中断请求。到 F240x 的中断请求信号是一个为两个 CPU 时钟脉冲宽的低电平脉冲。当任何外设中断请求 PIRQ 有效时，都会产生一个到 CPU 的中断请求脉冲 INTn。并且如果一个外设中断请求 PIRQ 在 CPU 对 INTn 应答后的一个周期内仍然有效，则另一个中断请求脉冲 INTn 也会产生。CPU 总是响应优先级高的外设中断请求。在 CPU 内核，这些中断标志在 CPU 响应中断时自动清0(注意：不是在外设中断级将中断标志清0)。

当 CPU 接受中断请求时，它并不知道是哪一个外设事件引起的中断请求。因此，为了让 CPU 能够区别这些引起中断的外设事件，在每个外设中断请求有效时，都会产生一个唯一的外设中断向量，其被装载外设中断向量寄存器(PIVR)中。CPU 应答外设中断时，从 PIVR 寄存器中读取相应中断的向量，并产生一个转到该中断服务子程序入口的向量。

正如前述，用 C 语言对 DSP 系统进行开发时，可以有两种方法实现可屏蔽的中断。下面将结合实例分别对这两种方法进行说明。

1. 通过软件识别中断标志的方法实现可屏蔽的中断

例如，如果来实现捕捉单元4的中断(CAP4INT)，则先建立一个如下所示的复位和中断向量文件 vectors.asm。

```
.Li title "vectors.asm"
.ref c_int0,nothing,_capint ;定义各个需要用到的符号
.sect ".vectors"
reset: b c_int0 ;复位向量
INT1: b nothing
INT2: b nothing
INT3: b nothing
INT4: b capint ;当发生 INT4中断时，程序跳转至capint 所标志的中断
;服务程序
INT5: b nothing
INT6: b nothing ;当发生其他级别的中断时，直接跳转至nothing 所标
;志的中断服务程序
```

因为CAP4INT 是 INT4 的一个外围中断源，故发生 CAP4INT 中断时，程序的 PC 指针将跳向INT4 处；而 INT4 处的执行语句为“b capint”，故程序随即跳向程序中 capint 标号处的中断服务程序中。但是当 INT4 的其他外围中断源（如 CAP1INT、CAP2INT 及 CAP3INT 等）发生中断时，程序也会跳转至程序中 capint 标号处的中断服务程序，这些中断却不是要求的中断。通过前面的叙述可知，每一个中断发生时，都会建立相应的中断标志，故

可以在capint 标号的中断服务程序中，通过相应的中断标志来辨别该中断是否为需要的合法中断。在本例中，当发生CAP4INT 时，其相应的中断标志位——EVBIFRC 的最低位置1。因此，在capint 标号的中断服务程序中，就可以对中断源进行如下判断：若 EVBIFRC 的最低位为1，则执行后面的中断服务程序；若 EVBIFRC 的最低位为0，表明没有发生CAP4INT 中断，则中断直接返回（或跳到与该中断标志相对应的中断服务程序）。在C 语言中可方便地实现前面所述的中断源辨别，其程序代码如下：

```
//CAPINT4中断服务程序
void interrupt capint()

    int flag;
    flag=EVBIFRC&0x01;          //判断是否是CAP4 中 断
    if(flag!=0x01)

        asm("clre INTN");      //返回前开中断
        return;                //如果不是CAP4中断，则直接返回
    }

    ...;                        1/如果是CAP4 中断，则执行响应的中断操作
    EVBIFRC=EVBIFRC|0x01;      // 写 1 清 除CAP4 中断标志
    asm("clrc INTM");           返回前开中断
    return;                     //中断返回
```

下面给出一个完整的示例，其汇编代码如下：

(1) 复位和中断向量文件 `vc101.asm`

```
title "vectors.asm"
.ref c_int0,nothing,GISR4          // 定义各个需要用到的符号
.sect ".vectors"
reset: b c_int0                    /复位向量
int1:   b nothing
int2:   b nothing
int3:   b _nothing
int4:   b GISR4
int5:   b nothing
int6:   b nothing
```

；当发生其他级别的中断时，直接跳转至符号_nothing 所标志的中断服务程序

2) 主程序

```

//该程序用于测试 DSP的 CAPTURE 模块，由TIMER1的比较模块输出一个 PWM 波形
//输入CAP4 引脚，待 CAP4 捕捉该PWM 的10次上升沿后，停止捕捉，并将10次捕捉值全部
//存于数组 result[10] 中
#include      "register.h"
//初始化子程序
int      result[10];          //定义一个存储结果的数组
int      k=0;
//系统初始化子程序
void      initial()

    asm("setc  INTM");          //禁止所有中断
    asm("setc  SXM");          //抑制符号位扩展
    asm("clrc  OVM");          //累加器中结果正常溢出
    asm("clrc  CNF");          //BO 被配置为数据存储空间
    SCSR1=0x81FE;              //CLKIN=6    MHz,CLKOUT=4×CLKIN=24    MHz
    WDCR=0x0E8;                //不使能看门狗，因为 SCSR2 中的 WDOVERRIDE
                                //即 WD 保护位复位后的缺省值为1, 故可以用软件禁止看门狗
    IMR=0x0000;                //禁止所有中断
    IFR=0x0FFFF;               //清除全部中断标志，写1清0
}
//捕获单元4初始化子程序
void      CAP4 INT()

    T3PER=0x0FFFF;             //通用定时器3的周期寄存器为0xFFFF
    T3CON=0x1400;               //通用定时器3为连续增计数模式
    T3CNT=0x00;                 //计数器清0
    WSGR=0x0000;                //禁止所有等待状态
    CAPCONB=0X0A440;            //设置捕获单元4为检测上升沿，且选择 TIMER3 为时钟
    asm("clrc  INTM");          //开总中断
    IMR=0X08;                   //允许中断优先级4的中断
    EVBIMRC=EVBIMRC|0x0001;     //允许 CAPTURE4 中 断
    EVBIFRC=EVBIFRC|0x0FFFF;

//定时器1初始化子程序，使其比较单元输出一个 PWM 波形
void      timerlint()

    MCRA=MCRA|0x1000;           //配置 IOPB4口为定时器1的比较输出
    MCRC=MCRC|0x0080;           //配置 IOPE7口为捕捉功能

```




```
GPTCONA=GPTCONA|0x0042;
T1PR=0x1FE;           //给定时器1的周期寄存器赋值
T1CON=0x1442;          //允许TIMER1比较输出,且TIMER1为连续增计数模式,立即启动
T1CNT=0x00;            //定时器1的计数器清0
T1CMP=0x0FF;           //给定时器1的比较寄存器赋值
}

/主程序
main()

    initial();          //系统初始化
    timerlint();         //定时器1初始化.使其输出一个PWM波形供捕捉
    CAP4INT();           //捕捉单元4初始化
    T3CON=T3CON|0x0040;  //启动定时器3
    while(1)
    {
        if(k==10)       break;

        asm("setc    INTN"); //捕捉10次后,禁止再中断

//若是由于干扰引起其他中断,则执行此子程序
void interrupt nothing()

    asm("clrc    INTM"); //返回前开中断
    return;
}

/CAPINT4 中断服务程序
void interrupt capint()

    int    flag;
    flag=EVBFRC&0X01;    //判断是否是CAP4中 断
    if(flag!=0x01)

        asm("clre    INTM"); // 返回前开中断
        return;           //如果不是CAP4中断,则直接返回
    }

    //如果是CAP4中断,则装载捕捉值
    EVBFRC=EVBFRC|0x01;   //写1清除CAP4中断标志
    asm("setc    INTN");   //返回前开中断
    return;               //中断返回
```

```

}

//装载捕捉值子程序
void    load()
{
    result[k]=CAP4FIFO;
    k++;
}

```

读取寄存器值，存于相应的数组

用这种方法实现中断具有程序代码少、易于理解等优点。

2. 通过外围中断向量寄存器 PIVR 的值识别中断的方法实现可屏蔽的中断

由 CPU 响应外设中断的流程可知，当某一个外设中断发生时，CPU 就会把相应的中断向量装载到外设中断向量寄存器 PIVR 中，并且每一个中断向量和各个外设中断是一一对应的。因此可以用 C 语言中的 SWITCH，在中断服务程序的入口处通过 PIVR 的值来判断发生的中断是否为需要的中断。详细程序代码如下(假设也是实现 CAP4INT，其 PIVR=0x0036):

```

//中断服务子程序
void    interrupt    GISR4()
{
    switch(*PIVR)
    {
        case 0x0036:
            break;                //如果是CAP4中断，则执行响应的中断服务程序代码
        default:break;           //如果不是CAP4中断，则直接返回
    }

    EVBIFRC=EVBIFRC|0x01;        //写1清除 CAP4 中断标志
    asm("clrc    INTM");         //返回前开中断
    return;                      //中断返回
}

```

下面给出一个利用此种方法实现中断的完整应用实例。此示例实现的功能与前述方法完全相同。源程序代码如下：

(1) 复位和中断向量文件 **vectors.asm**

```

.title            "vectors.asm"
.ref              _int0,nothing,GISR4        //定义各个需要用到的符号
.sect             ".vectors"
reset:            b            cint0        //复位向量

```

```
int1:      b      nothing
int2:      b      nothing
int3:      b      nothing
int4:      b      GISR4
int5:      b      nothing
int6:      b      nothing
```

/当发生其他级别的中断时，直接跳转至符号_nothing 所标志的中断服务程序

2) 程 序

//该程序用于测试 DSP的 CAPTURE 模块，由TIMER1 的比较模块输出一个 PWM 波形，此波形输入CAP4 引脚. 待CAP4 捕捉该 PWM 的10次上升沿后，停止捕捉，并将10次捕捉值存于数组 result[10] 中

```
#include "register.h"
```

//初始化子程序

```
int result[10]; //定义一个存储结果的数组
```

```
int k=0;
```

/系统初始化子程序

```
initial()
```

```
{
```

```
asm("setc INTM"); //禁止所有中断
```

```
asm("setc SXM"); //抑制符号位扩展
```

```
asm("clrc OVM"); //累加器中结果正常溢出
```

```
asm("clrc CNF"); //B0 被配置为数据存储空间
```

```
SCSR1=0x81FE; CLKIN=6 MHz,CLKOUT=4×CLKIN=24 MHz
```

```
WDCR=0x0E8; //不使能看门狗，因为 SCSR2 中的 WDOVERRIDE.即 WD 保
```

//护位复位后的缺省值为1. 故可以用软件禁止看门狗

```
IMR=0x0000; //禁止所有中断
```

```
IFR=0x0FFFF; //清除全部中断标志，写1清0
```

//捕获单元4初始化子程序

```
void CAP4INT()
```

```
T3PER=0XOFFFE; //通用定时器3的周期寄存器为0xFFFF
```

```
T3CON=0X1400; //通用定时器3为连续增计数模式
```

```
T3CNT=0X00; V/ 计数器清0
```

```
WSGR=0x0000; //禁止所有等待状态
```

```
CAPCONB=0X0A440; //设置捕获单元4为检测上升沿，且选择TIMER3为时钟
```

```
asm("clrc INTM"); //开总中断
```

```
IMR=0X08; //允许中断优先级4产生中断
```



```

    EVBIMRC=EVBIMRC|0X0001;           // 允许 CAPTURE4 中断
    EVBIFRC=EVBIFRC|0X0FFFF;
}

//定时器1初始化子程序，使其比较单元输出一个 PWM 波形
void timer1int()

    MCRA=MCRA|0X1000;                 //配置 IOPB4口为定时器1的比较输出
    MCRC=MCRC|0X0080;                 //配置 IOPE7 口为捕捉功能
    GPTCONA=GPTCONA|0X0042;
    T1PR=0X1FE;                       //给定时器1的周期寄存器赋值
    T1CON=0X1442;                     //允许TIMER1 比较输出，且 TIMER1 为连续增计数模式，
                                     //立即启动
    T1CNT=0X00;                       //定时器1的计数器清0
    T1CMP=0XOFF;                      //给定时器1的比较寄存器赋值
}

//主程序
main()

    initialO);                       //系统初始化
    timer1int();                      //定时器1初始化，使其输出一个PWM 波形供捕捉
    CAP4INT();                        //捕获单元4初始化
    T3CON=T3CON|0X0040;              //启动定时器3
    while(1)
    {
        if(k==10)break;
    }
    asm("setc INTM");                 //捕捉10次后，禁止再发生中断
}

//若是由于干扰引起其他中断，则执行此子程序
void interrupt nothing()

    asm("clrc INTM");                 //返回前开中断
    return;
}

//中断服务子程序
void interrupt GISR4()

    switch(*PIVR)

```

```
    case 0x0036:load() ;break;    / 如果是 CAP4中断 • 则装载相应的捕捉值
default:break;
```

```
    EVBIFRC=EVBIFRC|0x01;        //写1清除CAP4中断标志
asm("clrc INTM");                返回前开中断
return;                          //中断返回
```

K

//装载捕捉值子程序，相当于实际的中断服务程序

```
void    load()
```

```
{
```

```
    result[k]=CAP4FIFO;          //读取捕捉值，存于相应的数组
    k++;
```

```
}
```

由上述两个示例可以很清楚地了解，C 语言中断服务程序的编程方法，这些程序可以稍加修改，用于其他的中断服务程序，完成DSP 的各个中断的编程任务。



第 7 章 TMS320LF2407 实验开发系统



7.1 TMS320LF2407 实验开发板介绍

1. 结构介绍

F2407 实验开发系统由具有 JTAG 接口的仿真器和实验开发板构成。仿真器所配置的软件 CCS3.1 可开发全系列的 TI DSP 芯片。实验开发板配置如下：

- > F2407 芯片 1 片；
- > 64K×16 位 SRAM1 片；
- > 手动复位电路；
- > 8 个按键输入；
- > 8 个发光二极管输出；
- > 光电隔离的 8 路开关量输入/输出；
- > 128×64 点阵式液晶 LCD 显示电路；
- > 片内 16 个通道、10 位 A/D 转换器输入；
- > 6 路 PWM 输出、3 路比较/捕获 CAP 输入；
- > CAN 总线接口电路；
- 异步串行通信接口 SCI 接口电路；
- > SPI 总线接口的 4 路 D/A 转换输出；
- SPI 总线接口的 4 个 7 段数码管输出；
- > I²C 总线接口的非易失性存储器 EEPROM(24LC256)。

2. 实验开发板的主要功能及结构特点

实验开发系统兼具教学实验和产品开发两种功能。它功能齐全，考虑了工程应用中的信号隔离问题，特别适合电机控制的实验和产品开发，也可用于其他检测装置的开发；可根据实际要求配置为相应产品的开发装置；软件采用 C 语言源代码和汇编语言源代码调试，可以观察 CPU、输入/输出波形和通信工作等的状态；提供丰富的 DSP 外围电路，硬件采用模块化设计，各模块之间可灵活组合构成，操作方便、直观。

3. 可在目标板上进行的实验

CPU 指令实验(熟悉 DSP 指令特征);

CAN 总线的自发/自收及通过CAN 总线与上位机进行通信实验;

RS-232 异步串行通信接口与PC 机通信实验;

键盘与LED、LCD、7 段数码管显示实验;

►I/O 端口输入/输出实验;

>A/D 转换实验;

D/A 转换实验;

> 中断编程实验;

定时器实验;

>PWM 与 SVPWM 输出实验;

>捕获单元CAP 实验;

>EEPROM 非易失数据存储编程实验;

>SPI 总线与IC 总线接口编程实验。



7.2 TMS320LF2407 实验开发板模块功能介绍

1. 实验开发板概述

对 DSP 工程应用的开发者来说,要想缩短开发周期,降低开发难度,就必须有一套完整的软硬件开发工具,构成其高效的开发平台。实验开发板是一个比较好的选择,其结构框图如图7.1所示。由于F2407 要求3.3 V 供电,实验开发板上采用TI 公司的DC-DC 转换芯片TPS77633 把5V 电压转换成3.3V。可用20 MHz 外部四脚有源晶振或10 MHz 经锁相环4 倍频至24 MHz。

在线调试与仿真通过JTAG(Joint Test Action Group)标准测试接口及相应的控制器,从而不但能控制和观察系统中处理器的运行,测试每一块芯片,还可以用此接口下载程序。在TMS320 系列中,与JTAG 测试端口同时工作的还有一个分析模块,它支持断点的设置和程序存储器、数据存储器及DMA 的访问;程序的单步运行和跟踪以及程序的分支和外部中断的计数等。通过结合TI 公司的集成开发环境(CCS 3.1)给我们学习和应用DSP 提供了很大的方便。

F2407 芯片内含10位单极性A/D 转换模块。A/D 采样共有16个通道,最小转换时间为500 ns,既可实现对电机的三相电压采样而无须进行相位补偿,又可以实现与PWM 波形的同步采样。但由于A/D 转换模块是单极性的,所以对交流采样时,须加直流电平提升电路,使交流信号电压调整到0.3~3 V 之间。

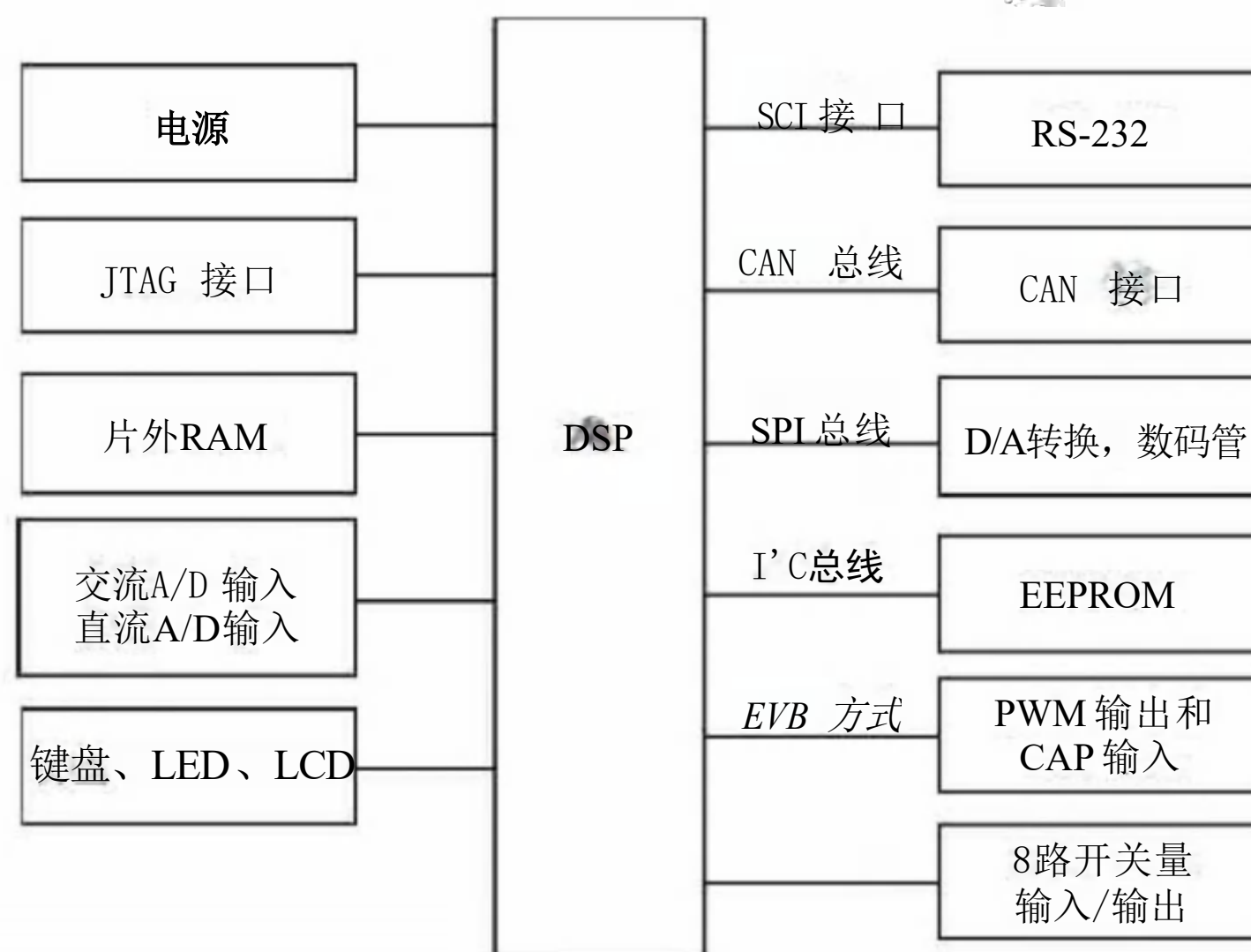


图 7.1 实验开发板结构框图

6路脉宽调制PWM 输出可用于变频调速系统中对电机的控制；3路捕获单元CAP 可用于记录脉冲的宽度，从而可为FFT 算法提供时间基准；光电编码器接口可用于测量电机转速以及将转速用于矢量控制等算法中。

2. 人机交换接口

键盘和显示器是人机交互信息的主要途径。实验开发板上的8个按键、8个发光二极管、4个7段数码管和液晶显示器(LCD)，可实现人机接口，输入和显示必要的简单信息。使用通用输入/输出IOPB 端口将输出信号送入74HC273，并锁存驱动发光二极管。可通过编制一些简单的端口程序来控制 I/O 输出，实现如点亮发光二极管及延时循环点亮发光二极管等功能。因其简单直观，对初学 DSP 汇编或用C 编程的读者来说，上手容易，对应用开发 DSP 有很大的帮助。LCD 可显示16×16点阵的汉字、图形以及其他符号，能够显示较大的信息量。8个按键与8个LED、7段数码管或 LCD 液晶显示器配合使用，可实现多种初学者易于掌握的功能。

3. 串行通信接口(SCI)

F2407 的 SCI 通信接口通过RS-232 转换芯片驱动与PC 机连接，可实现与 PC 机的异步通信。端部接口连线满足 RS-232 规范，可直接用市场上出售的 RS-232 转接线将 DSP 与 PC 机连接在一起，构成上位机与下位机的通信系统：将下位机(DSP) 中的中间变量和控制变量送到上位机上观察保存。

4. 控制器区域网络(CAN)

CAN 总线是德国Bosch 公司从20世纪80年代初开始，为解决现代汽车中众多的控制与

测试仪器之间的数据交换而开发的一种串行数据通信协议。它是一种多主机总线结构，通信介质可以是双绞线、同轴电缆或光导纤维。通信速率可达1 Mbps。

F2407 芯片自带CAN 控制器，在无外部 CAN 总线设备连接情况下，可以通过设置内部寄存器的自测试位，来实现CAN 控制器的自发/自收功能，为调试 CAN 通信提供了方便。实验开发板上设计了CAN 驱动器，采用 PCA82C250T 或者 MCP2551 将 DSP 引脚上的信号转换成CAN 总线差分驱动信号与其他CAN 节点或上位机相连，实现CAN 总线的通信。

5. 串行外设接口(SPI

实验开发板将 SPI 总线与4个8段数码管的驱动芯片74HC595 相连，通过SPI 总线将数据送到7段数码管上，也将带电压参考的D/A 芯片TLV5638 或 MAX5741 与 F2407 的 SPI 总线连接。由于D/A 芯片只能作为 SPI 总线的接收方(即从机)，所以作为 SPI 总线的主机 DSP 只需编程为发送数据形式。D/A 芯片输出的电压可以是 DSP 的各种参数和变量，例如，A/D 转换的结果值。

6. 用 FC 总线实现的非易失数据存储器件EEPROM 接口电路

由于F2407 没有设计IC 总线接口模块，所以只能使用通用I/O 引脚来模拟 I²C 总线的数据线 SDA 和时钟线 SCL，借助软件来模拟IC 总线的时序。实验开发板上的非易失数据存储器件 EEPROM 芯片为带写保护的24LC256。

由于 IC 总线的EEPROM 写操作至少需要5ms 的写周期，所以在调试和编程时，不能在写入数据后立即去读取刚写入的数据。经过写/读操作后，数据虽然被正确地写入到相应的地址，但读出的数据与写入的数据却不相符。

7. 光电隔离的8路开关量输入和8路开关量输出

在 DSP 应用中经常须采集开关量信号，而且也常常需要用开关量作为控制信号的输入。可适当地选择连接光电耦合阴极的电阻值，使光电耦合可靠导通，来实现不同电压等级下的电平匹配与电气隔离，从而模拟实际控制过程的开关量输入。

7.3 实验开发板的硬件设置

此实验开发板为板结构，由外部5 V 电源供电，板上设计的DC-DC 转换器将5 V 电压转换为3.3V，提供给 F2407 芯片和一些用3.3V 供电的器件。板上也有用5V 供电的芯片，例如7段数码管。

7.3.1 外部接口功能

外部接口功能如表7.1 所列。

表7.1 外部接口功能表

符 号	实际含义	功能介绍
P1	电源	电源输入，输入电压范围为9V~76 V
P2	电源	电源输入，输入电压5V
P3	JTAG	仿真器接口
P4	IOIN_IOOUT	8路开关量输入，8路开关量锁存输出
P5	DRIVE_PORT	6路脉宽调制(PWM)输出，4路故障保护输入，牛角插座
P6	DRIVE_PORT	6路脉宽调制(PWM)输出，4路故障保护输入. 接线端子
P7	CAN	CAN总线通信接口
P8	CAP_RS232	3路捕获单元(CAP)输入，RS-232串行通信接口
P9	ADIN1	直流电流输入
P10	ADIN2	直流电压输入
P11	ADIN3	交流电流输入
P12	ADIN4. DA	交流电压输入，D/A输出

7.3.2 实验开发板跳线设置

F2407 实验开发板上有5个跳线，这些跳线决定了实验开发板的功能。 F2407 实验开发板的大部分跳线为1×3。这些跳线只能选择1-2或2-3, 引脚2为中针，引脚1处有一方形焊盘，可在印刷电路板上焊接的一侧见到。此引脚在电路板丝网上常用“1”来做标记。默认设置都为连1与2连接。

警告：除非特别注明，否则所有跳线必须安装在1-2或2-3位置上。

实验开发板跳线设置跳线引脚定义如图7.2所示。跳动线说明如表7.2所列。实验开发板的跳线与端子位置如图7.3所示。

表7.2 实验开发板跳线说明

跳线号	尺 寸	功 能
JP1	1×2	CAN总线1200电阻选择
JP2	1×3	VCCP闪存烧写电压选择
JP3	1×3	PS/DS选择
JP4	1×3	MP/MC选择
JP5	1×3	外部扩展总线接口使能选择
JP6	1×3	PWM输出脚上电复位电平选择

同三

图7.2 跳线引脚定义

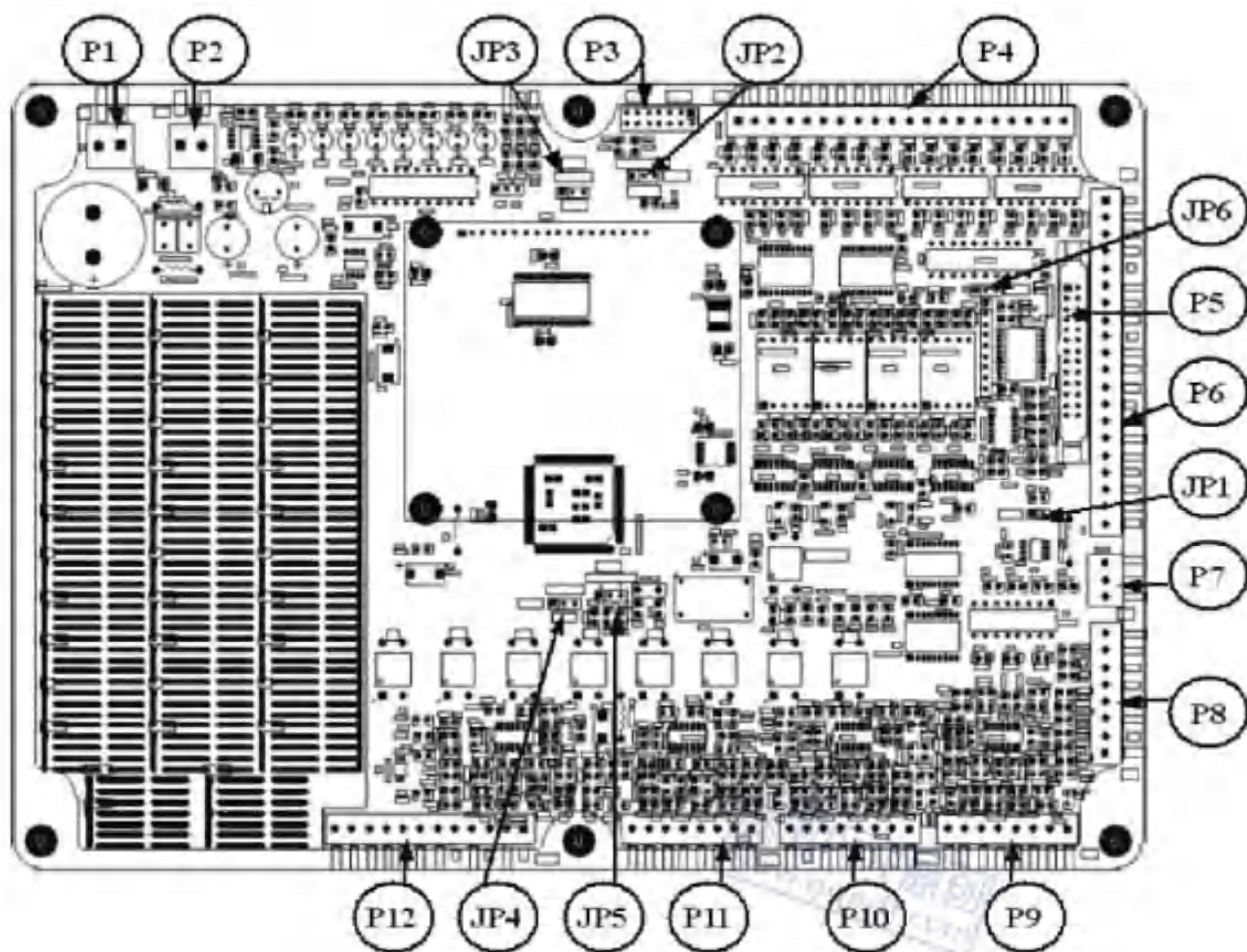


图7.3 实验开发板跳线与接口位置图

7.3.3 部分插座引脚详细说明

实验开发板端部插座引脚详细说明如表7.3所列。

DSP 实验开发板实物图如图7.4所示。

注：AVCC 为模拟电源3.3 V,AGND 为模拟地，VCC 为数字电路电源3.3 V.GND 为数字电路地，VIN 为隔离的外部输入电源12 V.GNDIN 为隔离的外部输入电源的地。

表7.3 实验开发板端部插座引脚详细说明

功能介绍		部分主要引脚定义			
P1	电源输入端	1:9 V~76 V 2:GND(地)	1:F0 2:PWM1 3:VCC 4:GND 5:F1 6:PWM2 7:VCC 8:GND 9:F2 10:PWM3 11:VCC 12:GND 13:F3 14:PWM4 15:PWM5 16:PWM6 17:VCC 18:GND 19:PWM1 20:VCC		
P2	电源输入端	1:5V 2:GND(地)			
P3	JTAG仿真器接口	按JTAG接口标准			
P4	8路开关量输入， 一路开关量锁存输出 接线端子	1:VIN 2:IN1 3:IN2 4:IN3 5:IN4 6:IN5 7:IN6 8:IN7 9:IN8 10:GNDIN 11:VIN 12:OUT1 13:OUT2 14:OUT3 15:OUT 4 16:OUT5 17:OUT6 18:OUT7 19:OUT 8 20:GNDIN	P6	6路脉宽调制 (PWM) 输出，4路故障保护输 入接线端子	
		P7	CAN总线通信接线端	1:CANH 2:CANL 3:GNDL (屏蔽线接地)	
		P8	3路捕获单元 (CAP) 输入，RS-232串行通 信接线端子	1:5V 2:HC 3:B_QEP 4:A_QEP 5:GND 6:GND 7:RS-TX 8:RS-RX	
P5	6路脉宽调制 (PWM) 输出，4路故障保护输 入排线插座	1:PWM1 2:PWM2 3:PWM3 4:PWM4 5:PWM5 6:PWM6 7:F0 8:F1 9:F2 10:F3 11:PWM7 12:VCC 13:VCC 25~26:VCC 14~24:GND	P9	模拟量输入 (直流电 流)	1:IADC 2:AGND 3:IBDC 4:AGND 5:ICDC 6:AGND 7:DC=V 8:AVCC

续表7.3

功能介绍	部分主要引脚定义	功能介绍	部分主要引脚定义
P10模拟量输入(直流电压)	1:UADC 2:AGND 3:UBDC 4:AGND 5:UCDC 6:AGND 7:DC 8:AVCC	P12 4路模拟量输出, 交流电压输入接线端子	1:UA 2:AGND 3:UB 4:AGND 5:UC 6:AGND 7:OUTI 8:OUTC 9:AGND 10:OUTB 11:OUTA 12:AVCC
P11模拟量输入(交流电流)	1:CURA 2:AGND 3:CURB 4:AGND 5:CURC 6:AGND 7:DC-C 8:AVCC		

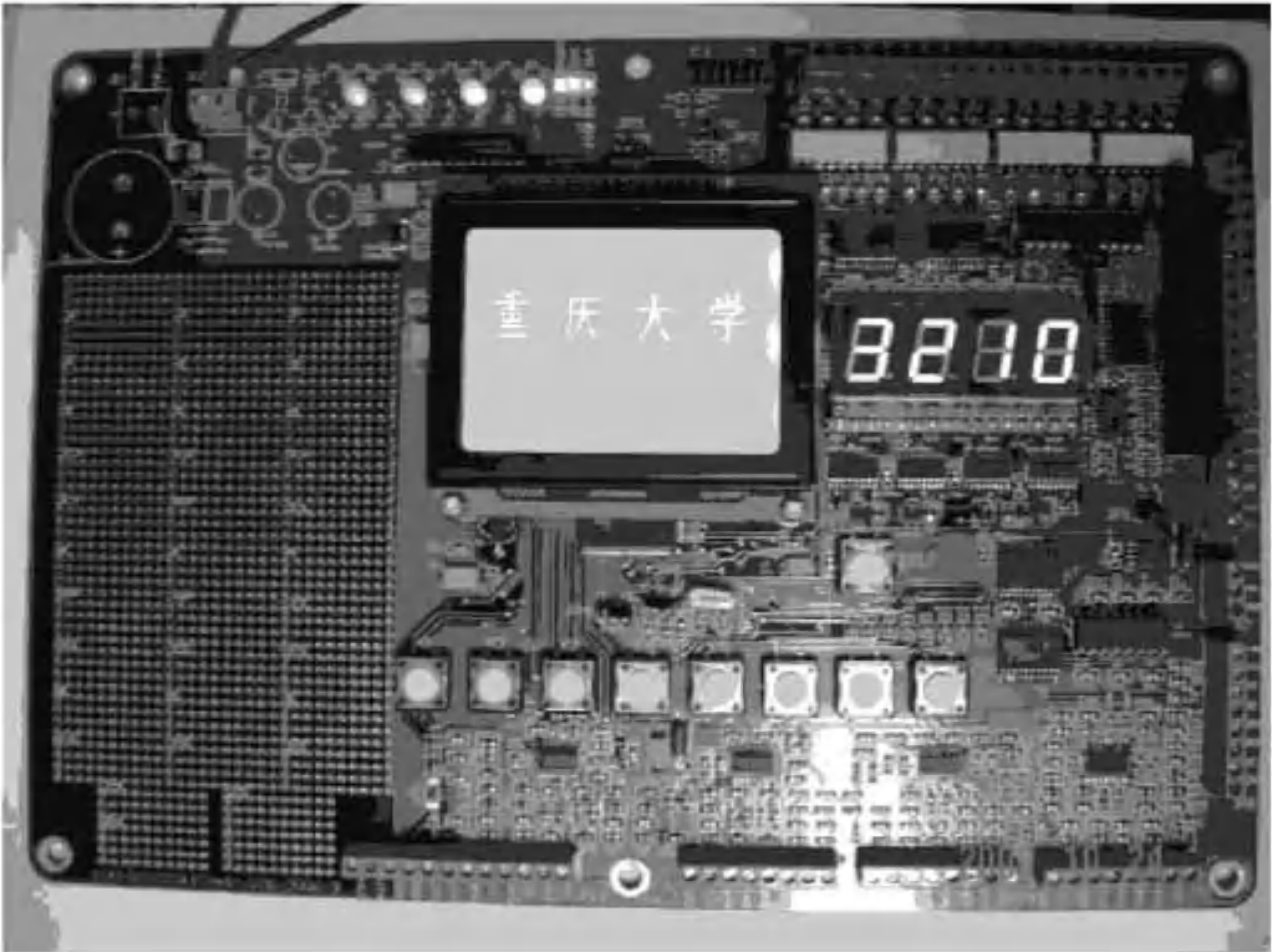


图7.4 实验开发板实物图

第 8 章 数字量I/O



8.1 数字量I/O 端口概述

F2407 系列有多达41个通用、双向的数字量I/O(GPIO) 引脚，其中大多数都是基本功能和一般I/O 复用引脚。F2407 系列的大多数 I/O 引脚都可用来实现其他功能。数字量 I/O 端口模块采用了一种灵活的方法，可实现控制专用I/O 和复用 I/O 引脚的功能。所有专用 I/O 和复用I/O 引脚的功能都可通过9个16位控制寄存器来设置。这些寄存器可分为两类：

- ＞I/O 端口复用控制寄存器(MCRx) 用来控制选择 I/O 端口作为基本功能或一般I/O 引脚功能。
- ＞数据和方向控制寄存器(PxDATDIR) 当 I/O 端口用作一般I/O 引脚功能时，用数据和方向控制寄存器可控制数据和双向 I/O 引脚的数据方向。这些寄存器直接与双向 I/O 引脚相连。

8.2 数字量I/O 端口寄存器

图8. 1给出了 F2407 系列I/O 端口复用引脚配置简图。从图上可以看出一些寄存器单元的配置与实际I/O 引脚的内部结构之间的联系。

表8. 1列出了数字量I/O 模块可用的寄存器单元。与其他 F240x 外设一样，这些寄存器被存储器映射到数据空间，地址为7090h~709Fh。 寄存器单元中保留的位是无效的，读时为0, 写对其无影响。

表8. 1 数字量I/O 控制寄存器地址

地 址	存储寄存器	说 明	地 址	存储寄存器	说 明
7090h	MCRA	I/O端口复用控制寄存器A	709Ch	PCDATDIR	I/O端口C数据和方向寄存器
7092h	MCRB	I/O) 端口复用控制寄存器B	709Eh	PDDATDIR	I/O端口D数据和方向寄存器
7094h	MCRC	I/O端口复用控制寄存器C	7095h	PEDATDIR	I/O) 端口E数据和方向寄存器
7098h	PADATDIR	I/O) 端口A数据和方向寄存器	7096h	PFDATDIR	I/O端口F数据和方向寄存器
709Ah	PBDATDIR	I/O端口B数据和方向寄存器			

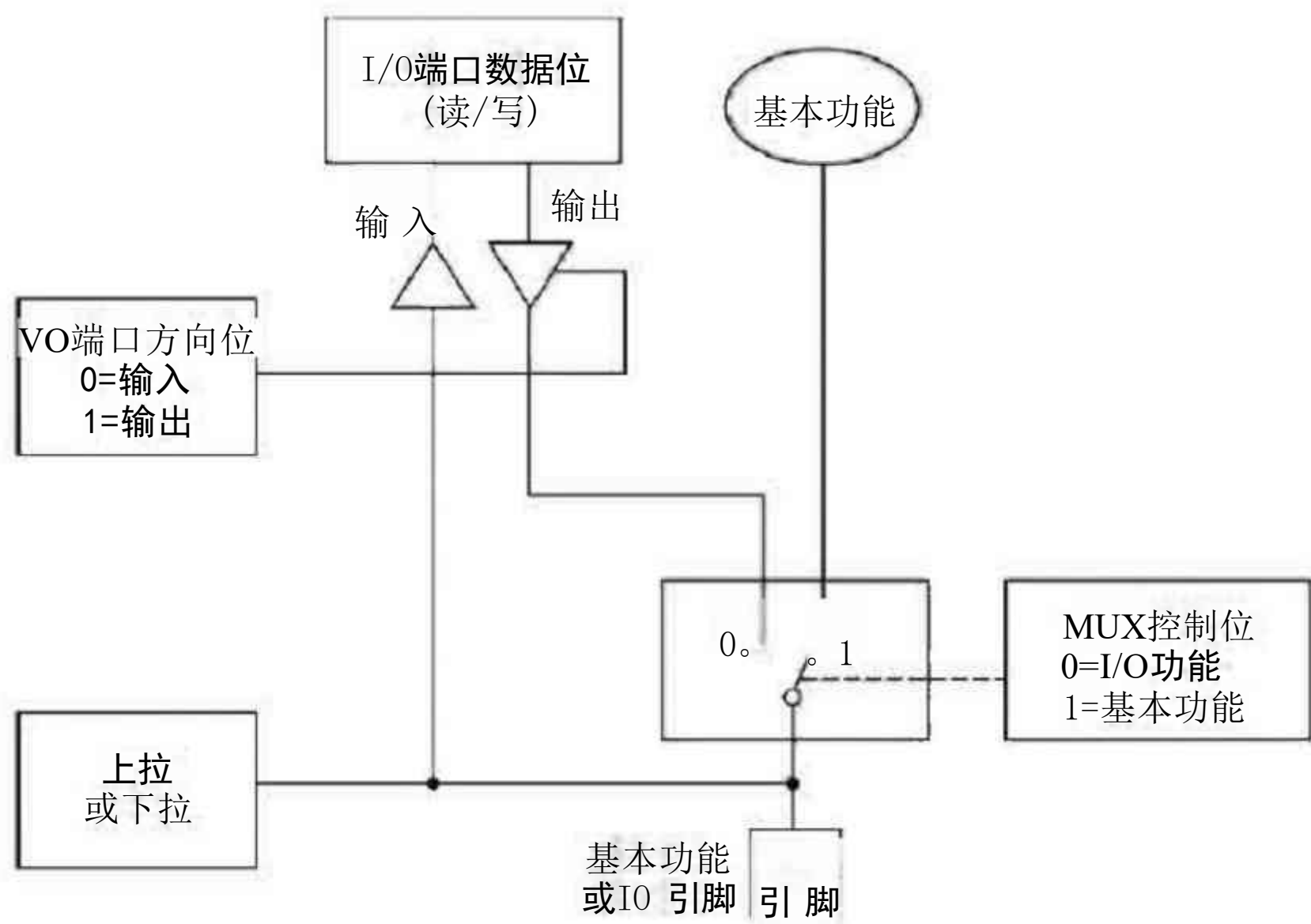


图 8.1 复用引脚配置简图

8.2.1 I/O 端口复用输出控制寄存器

1. I/O 端口复用控制寄存器 A (MCRA) —— 地 址 7 h

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

注：R= 可读；W= 可写；0=复位后的值。

I/O 端口复用控制寄存器 A(MCRA) 配置如表 8.2 所列。

表8.2 I/O 端口复用控制寄存器 A (MCRA) 配 置

位	位的名称	引脚功能选择	
		基本功能 (MCRA. n=1)	一般I/O端口 (MCRA. n=0)
0	MCRA. 0	SCITXI	IOPA0
1	MCRA. 1	SCIRXD	IOPA1

续表8. 2

位	位的名称	引脚功能选择	
		基本功能 (MCRA. n=1)	一般I/O端口 (MCRA. n=0)
2	MCRA. 2	XINT1	IOPA2
3	MCRA. 3	CAP1/QEP1	IOPA3
4	MCRA. 4	CAP2/QEP2	IOPA4
5	MCRA. 5	CAP3	IOPA5
6	MCRA. 6	PWM1	IOPA6
7	MCRA. 7	PWM2	IOPA7
8	MCRA. 8	PWM3	IOPB0
9	MCRA. 9	PWM4	IOPB1
10	MCRA. 10	PWM5	IOPB2
11	MCRA. 11	PWM6	IOPB3
12	MCRA. 12	T1PWM/TICMP	IOPB4
13	MCRA. 13	T2PWM/T2CMP	IOPB5
14	MCRA. 14	TDIRA	IOPB6
15	MCRA. 15	TCLKINA	IOPB7

2. I/O 口复用控制寄存器B (MCRB) —— 地址7092h

15	14	13	12	11	10	9	8
MCRB. 15	MCRB. 14	MCRB. 13	MCRB. 12	MCRB. 11	MCRB. 10	MCRB. 9	MCRB. 8
RW_1	RW_1	RW_1	RW_1	RW_1	RW_1	RW_1	RW_o
7	6	5	4	3	2	1	0
MCRB. 7	MCRB. 6	MCRB. 5	MCRB. 4	MCRB. 3	MCRB. 2	MCRB. 1	MCRB. 0
RW_0	RW_0	RW_0	RW_0	RW_0	RW_0	RW_1	RW_1

注：R=可 读；W=可写；_x= 复位后的值，x=1,x=0。

I/O 端口复用控制寄存器 B(MCRB) 配置如表8. 3所列。

表 8 . 3 1/ O 端口复用控制寄存器 B(MCRB) 配置

位	位的名称	引脚功能选择	
		基本功能 (MCRB. n=1)	一般I/O端口 (MCRB. n=0)
0	MCRB. 0	W/R	IOPC0
1	MCRB. 1	BIO	IOPCI
2	MCRB. 2	SPISIMO	IOPC2
3	MCRB. 3	SPISOMI	IOPC3
4	MCRB. 4	SPICLK	IOPC4
5	MCRB. 5	SPISTE	IOPC5
6	MCRB. 6	CANTX	IOPC6
7	MCRB. 7	CANRX	IOPC7
8	MCRB. 8	XINT2/ADCSOC	1OPD0
9	MCRB. 9	EMUO	保留位
10	MCRB. 10	EMUI	保留位
11	MCRB. 11	TCK	保留位
12	MCRB. 12	TDI	保留位
13	MCRB. 13	TDO	保留位
14	MCRB. 14	TMS	保留位
15	MCRB. 15	TMS2	保留位

注意：表8 . 3中MCRB.9~MCRB.15 即用斜体表示的位必须配置成1。这些引脚是系统的专用引脚，与硬件仿真器接口(JTAG) 相连，实现 DSP 的在线仿真功能。对以上的任一位写0, 将会引起不可预测的结果。

3.I/O 口复用控制寄存器 C(MCRC)—— 地址**7094 h**

15	14	13	12	11	10	9	8
保留	MCRC. 14	MCRC. 13	MCRC. 12	MCRC. 11	MCRC. 10	MCRC. 9	MCRC. 8
	RWa	RW_0	RW_0	RW_0	RW_0	RW_0	RW_0
7	6	5	4	3	2	1	0
MCRC. 7	MCRC. 6	MCRC. 5	MCRC. 4	MCRC. 3	MCRC. 2	MCRC. 1	MCRC. 0
RW_0	RW_0	RW_0	RW_0	RW_0	RW_0	RW_0	RW_]

I/O 端 口 复 用 控 制 寄 存 器 C(MCRC) 配 置 如 表 8 . 4 所 列 。

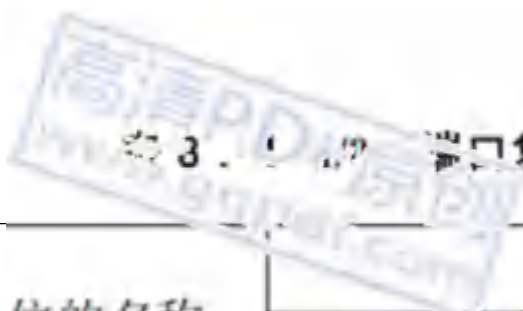


表 3.9.12 端口复用控制寄存器 C(MCRC) 配置

位	位的名称	引脚功能选择	
		基本功能 (MCRC. n=1)	一般I/O端口 (MCRC. n=0)
0	MCRC. 0	CLKOUT	IOPE0
1	MCRC. 1	PWM7	IOPE1
2	MCRC. 2	PWM8	IOPE2
3	MCRC. 3	PWM9	IOPE3
4	MCRC. 4	PWM10	IOPE4
5	MCRC. 5	PWM11	IOPE5
6	MCRC. 6	PWM12	IOPE6
7	MCRC. 7	CAP4/QEP3	IOPE7
8	MCRC. 8	CAP5/QEP4	IOPF0
9	MCRC. 9	CAP6	IOPF1
10	MCRC. 10	T3PWM/T3CMP	IOPF2
11	MCRC. 11	T4PWM/T4CMP	IOPF3
12	MCRC. 12	TDIRB	IOPF4
13	MCRC. 13	TCLKINB	IOPF5
14	MCRC. 14	IOPF6	IOPF6
15	MCRC. 15	保留位	保留位

8.2.2 I/O 端口数据和方向寄存器

F2407 系列共有6个数据和方向控制寄存器(PxDATDIR)。当 I/O 端口用作通用I/O 功能时，用数据和方向控制寄存器可控制数据和到双向 I/O 引脚的数据方向。这些寄存器直接与 I/O 引脚相连。当I/O 用作基本功能时，这些寄存器的设置对相应的引脚无影响。

1. 端 口A 数据和方向控制寄存器(PADATDIR) —— 地 址7098 h

15	14	13	12	11	10	9	8
A7DIR	A6DIR	A5DIR	A4DIR	A3DIR	A2DIR	A1DIR	A0DIR
RW_0	RW_0	RW_0	RW0	RW_0	RW_0	RW_0	RW_0
7	6	5	4	3	2	1	0
IOPA7	IOPA6	IOPA5	IOPA4	IOPA3	IOPA2	IOPA1	IOPA0
RW_十	RW_十	RW_十	RW_十	RW_十	RW_十	RW_十	RW_十

注：_十=复位后的值与相应引脚的状态有关；R= 可读；W= 可写；0=复位后的值。

位15~8 AnDIR

- 0 配置相应的引脚为输入方式；
- 1 配置相应的引脚为输出方式。

位7~0 IOPAn

当 AnDIR=0, 即引脚为输入方式时：

- 0 读相应引脚的值为低电平；
- 1 读相应引脚的值为高电平。

当 AnDIR-1, 即引脚为输出方式时：

- 0 设置相应的引脚，使其输出为低电平；
- 1 设置相应的引脚，使其输出为高电平。

如果 I/O 端口用作一般I/O，则在端口初始化时.就必须对数据和方向控制寄存器进行设置，规定其为输入还是输出。

由于其他端口的数据和方向控制寄存器的设置方法与端口A 相同，在此不作重复介绍，只给出其寄存器单元。

2. 端 口B 数据和方向控制寄存器(PBDATDIR) ——地址709 Ah

15	14	13	12	11	10	9	8
B7DIR	B6DIR	B5DIR	B4DIR	B3DIR	B2DIR	BIDIR	BODIR
RW_0	RW_o	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	
RW_十	RW_十	RW_击	RW_十	RW_十	RW_十	RW_十	

3. 端 口 C 数据和方向控制寄存器(PCD ATDIR)—— 地 址709 Ch

15	14	13	12	11	10	9	8
C7DIR	C6DIR	CSDIR	C4DIR	C3DIR	C2DIR	CIDIR	CODIR
RW0	RW0	RW0	RW0	RW9	RW9	RW_0	RW_0
7	6	5	4	3	2	1	0
IOPC7	IOPC6	IOPC5	IOPC4	IOPC3	IOPC2	IOPCI	IOPCO
RW_+	RW+	RW_+	RW_十	RW_十	RW_十	RW_+	RW_x

注：十=复位后的值与相应引脚的状态有关；R=可 读 ； W=可写； 0=复位后的值； x= 未定义。

4. 端 口D 数据和方向控制寄存器(PDDATDIR)—— 地 址709 Eh

15~9	8
保留位	DODIR
	RW_0
7~1	0
保留位	IOPA0
	RW_+

5. 端 口 E 数据和方向控制寄存器(PEDATDIR)—— 地 址 7095 h

15	14	13	12	11	10	9	8
E7DIR	E6DIR	E5DIR	E4DIR	E3DIR	E2DIR	E1DIR	EODIR
RW_0	RW_0	RW_0	RW_0	RW_0	RW_0	RW_0	RW_0
7	6	5	4	3	2	1	0
IOPE7	IOPE6	IOPE5	IOPE4	IOPE3	IOPE2	IOPE1	IOPE0
RW_+	RW_+	RW_+	RW_+	RW_+	RW_+	RW_+	RW_x

6. 端 口 F 数据和方向控制寄存器(PFDATDIR)—— 地 址7096 h

15	14	13	12	11	10	9	8
保留	F6DIR	F5DIR	F4DIR	F3DIR	F2DIR	F1DIR	FODIR
RW_0	RW_0	RW_0	RW_0	RW_0	RW_0	RW_0	RW_0
7	6	5	4	3	2	1	0
保留	IOPF6	IOPF5	IOPF4	IOPF3	IOPF2	IOPF1	IOPF0
RW_0	RW_+	RW_+	RW_+	RW_+	RW_+	RW_+	RW_+

8.3 I/O 端口应用

8.3.1 I/O 端口作为输出

下面的设计是一个F2407 与发光二极管的接口电路及应用。用发光二极管作指示灯是控制系统中一种常用的方法。其接口电路简单，编程方便且直观，易于初学者上手。设计电路如图8.2所示。74HC273 用于驱动8个发光二极管，CLR 接 F2407 的复位引脚，使在复位时74HC273 输出为低电平，不点亮发光二极管。 F2407 的 IOPCA3、IOPCA4 及 IOPCA5 分别接

74LVC138 的 A、B 及 C 输入，通过其 Y7 选通 CLK 作为 74HC273 的控制信号。当 $\overline{Y7}$ 输出为高时，74HC273 的输出有效(即输出信号反映输入的内容);当 Y7 输出为低时，输出无效(即输入变化时输出不变)。用 IOPB 端口来控制发光二极管的亮灭(某位为高则相应的 LED 发光，某位为低则相应的 LED 熄灭)。编制程序时，通常先向 IOPB 端口发送需要显示的数据，再通过 Y7 发送一个脉冲，循环点亮 8 个发光二极管。

1. 汇编程序设计

下面给出的程序用来循环点亮 8 个发光二极管。在此用软件延时的方法来调节发光二极管的延时间隔。通过点亮发光二极管，使编程者对 DSP 指令有一定的了解，进而对 F2407 的内部结构和 I/O 端口寄存器的配置有进一步的认识。

以下是源程序代码：

```
TOSFT REG    usect    ".data0",1           ;要显示的数据寄存器
              .include  "F2407REGS.H"       ;引用头部文件
              .def      _c_int0
              .text
c      int0:                                     :相当于主程序的入口
              CALL      SYSINIT              ;调系统初始化程序
              LDP        #DP_PF2              ;指向7080h~7100h 区
              LACL       NCRA
              AND        #000FFH              ;IOPB 端口配置为一般I/O 功能
              SACL       MCRA
              LACL       MCRA
              AND        #OFF00H              ;TOPA 端口配置为一般I/O 功能
              SACL       MCRA
              LACL       PBDATDIR
              OR         #OFF00H              ;IOPB 端口设置为输出方式
              SACL       PBDATDIR
              LACL       PADATDIR
              OR         #OFF00H              ;ICP 端口设置为输出方式
              SACL       PADATDIR              ;指向0280h~0300h 区
              LDP        #5H
              SPLK       #80H,IOSFT_REG       ;给显示的数据赋初值
LOOP:        LDP        #DP_PF2
              LACL       PADATDIR
              AND        #OFFC7H              ;138选择ch0,ledclk=1
              SACL       PADATDIR              ;开74HC273 片选信号
              LDP        #5H
              LACL       IOSFT_REG
```



```
LDP      #DP_PF2
OR       #0FF00H
SACL     PBDATDIR      ;送要显示的数据到 IOPB 端口
LACL     PADATDIR
OR       #0038H      ;138选择 ch7.1edclk=0
SACL     PADATDIR      ;关 7 4HC273 片选信号
CALL     DELAY      ;调延时程序
LDP      #5H
LACL     IOSFT_REG
BIT      IOSFT_REG,BITO ;判断是否循环完一次，即已点亮第8个发光二极管
ROR
SACL     IOSFT_REG
BCND     LOOP1,TC
B        WAIT
LOOP1:   LDP      #5H
        SPLK     #80H,IOSFT_REG ;如循环完一次，则显示数据赋初值
WAIT:    NOP
        B        LOOP
;系统初始化程序
SYSINIT:
        SETC     INTM
        CLRC     SXM
        CLRC     OVM
        CLRC     CSI      ;B0 被配置为数据存储空间
        LDP      #0EC
        SPLK     #01H,CSR1 ;CLKIN=6 MHz,CLKOUT=24 MHz
        SPLK     #0E8H,WDCR ;禁止 WDT
        LDP      #0
        SPLK     #0000H,IMR ;禁止
        SPLK     #0FFFFH,IFR ;清全部中断标志
        RET
;软件延时程序
DELAY:
        MAR      *,AR4
        LAR      AR4,#OFFFEH
        LAR      ARO,#00H
DELAY1:  SBRK     #1
        NOP
        CMPR     00
        BCND     DELAY1,NTC
        RET
```

2.C 语言程序设计

下面给出的程序用来循环点亮8个发光二极管。在此用软件延时的方法来调节发光二极管的延时间隔。通过点亮发光二极管，使编程者对 DSP 的 C 语言编程开发有一定了解，进而对 F2407 的内部结构和 I/O 端口寄存器的配置有进一步的认识。

关于该程序的几点说明如下：

(1) 该程序中所有与DSP 硬件有关的寄存器都用指针方式寻址，这与文件 register.h 中 DSP 寄存器的定义相对应。

(2) SXM、OVM、CNF 和 INTM 等位于STO 或 ST1 寄存器中，而这两个寄存器在数据存储器中无直接映射地址，所以对这些位的操作应该嵌入汇编语句。

(3) 寄存器PBDATDIR 的高8位是IOPB 端口数据流动方向的设置，设置完成后一般不再改变；低8位是数据内容，在该程序中若改变其内容，则可实现LED 的循环显示。若直接通过 PBDATDIR 移位的方式实现其数据内容的变化，则很不方便(因为移位时高8位也要移动)，因此在程序中设置了另外一个寄存器 led，其高8位为0，低8位存储需要显示的数据。用 C 语言指令“led=led>>1” 实现数据右移一位。每次须改变 LED 的显示内容时，用指令“PBDATDIR=PBDATDIR&0x0FF00” 屏蔽掉其数据位，再用指令“PBDATDIR=PBDATDIR[led]” 把需要显示的数据赋给IOPB 端口即可。

以下是源程序代码：

7/ 该程序用于实验演示模板上的8个 LED的循环显示

```
#include "register.h"
main() //初始化子程序
{
    int led;
    int i,k;
    asm("setc SXM"); //抑制符号位扩展
    asm("clre OVM"); //累加器中结果正常溢出
    asm("clrc CNF"); //B0 被配置为数据存储空间
    asm("setc INTM"); //禁止所有中断
    SCSR1=0x81FE; //CLKIN=10 MHz,CLKOUT=4×CLKIN=40 MHz
    WDCR=0x0E8; //禁止看门狗，因为 SCSR2 中的WDOVERRIDE
    //可以用软件禁止看门狗
    IMR=0x0000; //禁止所有中断
    IFR=0x0FFFF; //清除全部中断标志，写1清0
    MCRA=MCRA&0x00c7; //IOPA3、IOPA4、TOPA5及 IOPB端口配置为一般的I/O 功能
    //F240x 的端口均为8位，MCRA 为16位因此控制了IOPA
    //和 IOPB 的设置
}
```

```

PBDATDIR=0x0FF00;
PADATDIR=PADATDIR&0x0FF00;
PADATDIR=PADATDIR|0x0FF38;

while(1)
{
    for(led=0x0080,i=0;i<8;led=led
    <
        PBDATDIR=PBDATDIR&0x0FF00;
        PBDATDIR=PBDATDIR|led;
        PADATDIR=PADATDIR&0x0FF00;
        PADATDIR=PADATDIR|0x0FF38;
        for(k=0;k<0x0fffe;k++)
            k=k;
    }
}

//直接返回中断服务程序
void interrupt nothing()
{
    return;
}

```

//IOPB端口设置为输出方式，熄灭全部的LED

//IOPA3,IOPA4,IOPA5设置为输出方式

//且 LEDCLK=1

>>1,i++)

//首先屏蔽B 端口的各位

//把需要显示的值给 IOPB端口

//LEDCLK=1

8.3.2 键盘与发光二极管配合使用程序

下面的设计是一个 F2407 与键盘和发光二极管的接口电路及应用。有8个键和8个发光二极管，键盘的值先用 IOPB 端口直接读取，即相应的 I/O 用作输入方式；把 IOPB 端口作为输出方式，去点亮8个发光二极管。其电路如图8.2所示。键盘 K1~K8 分别对应于I/O 端口 IOPB0~IOPB7。如果按下8个键盘中的某个按键，则相应的发光二极管点亮。

在此用查询方式来读取键值。当S1 键按下时，就点亮发光二极管LED0。S1 键对应的输入 I/O 引脚为IOPB0。在 IOPB0 作为一般I/O 输入的情况下，若SI 键按下，则端口B 数据和方向控制寄存器(PBDATDIR) 的第0位为0。经过延时消抖动后，若 PBDATDIR.0 仍为0，则表示S1 键按下。

下面给出具体程序。

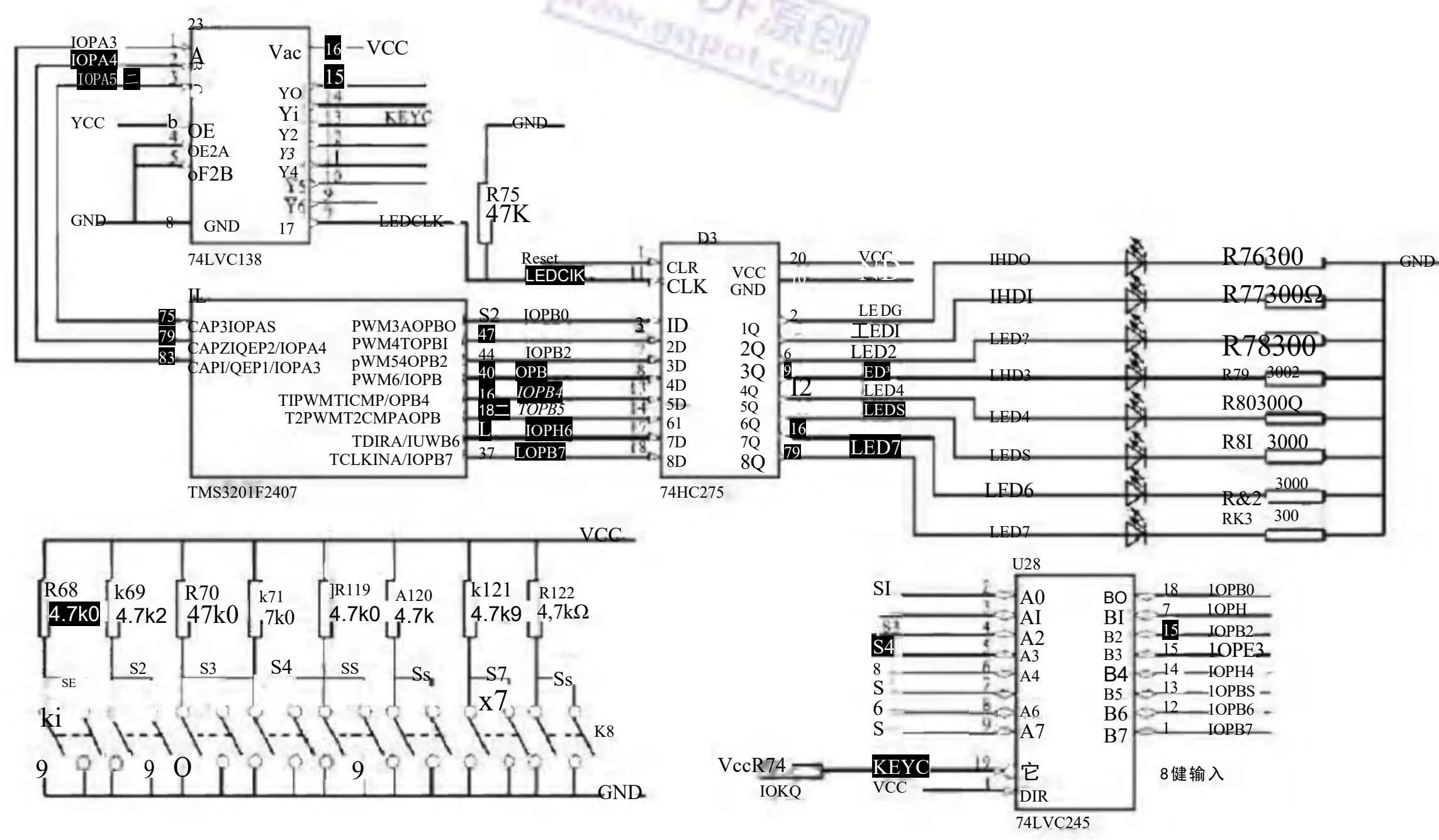


图 8.2 F2407 与 键盘、LED 接口 电路

1. 汇编程序

```
KEYXS      .usect      ".data0",1      ;LED 显示的数据
KEYDATA    .usect      ".data0",1      :前次LED 显示的数据
DP_USER    .set        5
            .include    "F2407REGS.H"   ;引用头部文件
            def         c_int0

;主程序
            .text

c_int0
            CALL        SYSINIT          :调系统初始化程序
            LDP          #DP_USER         ;指向0280h~0300h 区
            SPLK         #0000H,KEYXS     ;给要显示的数据赋初值
            B           LED              ;使LED全部熄灭

LOOP:      CALL        KEY               ;调键盘程序，即扫描键盘
LED:       LDP          #DP_PF2          ;选数据页
            LACL        PADATDIR         ;将PADATDIR 装载到ACC 的低16位
            OR          #00038H          ;LEDCLK=0
            SACL        PADATDIR         ;开74HC273片选信号
```

```
LDP      #DP_USER
LACL     KEYXS
OR       #OFF00H      ;IOPB 端口为输出方式
LDP      #DP_PF2
SACL     PBDATDIR      ;送要显示的数据到 IOPB 端口
LACL     PADATDIR
OR       #Off10H      ;LEDCLK=1
AND      #Off10H
SACL     PADATDIR      ;关74HC273 片选信号
CALL     KEYDELAY
WAIT:    RPT          #7      ;延时
NOP
B        LOOP          ;跳转，循环查键
```

;系统初始化程序

SYSINIT:

```
SETC     INTM
CLRC     SXM
SETC     OVM
CLRC     CNF      ;B0 被配置为数据存储空间
LDP      #0EOH
SPLK     #81FEH,SCSR1      ;CLKIN=6MHz,CLKOUT=24      MHz
SPLK     #0E8H,WDCR      ;禁止看门狗(WDT)
LDP      #0
SPLK     #0000H,IMR      ;禁止中断
SPLK     #OFFFFH,IFR      ;清全部中断标志
RET
```

;键盘初始化程序

KEYINIT:

```
LDP      #DP_PF2
LACL     MCRA
AND      #000c7H      ;IOPA3、IOPA4、TOPA5和 IOPB端口配置为一般的 I/O 功能
SACL     MCRA
LACL     PBDATDIR
AND      #0000H      ;IOPB 端口设置为输入方式
SACL     PBDATDIR
LACL     PADATDIR
OR       #OFF11H      ;IOPA3、IOPA4 和 IOPA5 设置为输出方式，且KEYC=1
```

```

        AND      #0FF10H
        SACL     PADATDIR
        RET

;
; 键盘程序
KEY:
        CALL     READKEY           :调读键程序
        LACL     KEYXS
        BCND     KEY,EQ           ;没有键按下，跳转到 KEY,循环查键
        CALL     KEYDELAY         ;延时消抖动
        CALL     READKEY         ;再一次读键值
        LACL     KEYXS
        BCND     KEY,EQ           ;没有键按下，跳转到KEY,循环查键
        B        LED             :有键按下，跳转到 LED,进行显示
```

;读键子程序

```

READKEY:
        CALL     KEYINIT          ;调键盘初始化程序
        LDP      #DP_PF2
        LACL     PBDATDIR         ;取出键值PBDATDIR.0~PBDATDIR.3
        OR       #OFF00H         ;屏蔽高8位
        CMPL
        LDP      #DP_USER
        SACL     KEYXS            ;存放键值
        RET                    ;返回
```

;

;用软件延时30 ms 消抖动

```

KEYDELAY:
        LACC     #6000
KEYDI:  SUB      #1
        RPT      #80
        NOP
        BCND     KEYDI,NEQ
        RET
```

;假中断程序
PHANTOM


```
KICK_DOG          ;复位看门狗
RET
END
```

2.C 语言程序

```
//主程序
//该程序用于实验演示模板上的8个按键和8个LED的显示
#include    "register.h"

int j,k;
main()      //初始化子程序
{
    asm("setc  SXM");      //抑制符号位扩展
    asm("clrc  OVM");      //累加器中结果正常溢出
    asm("clrc  CNF");      //B0 被配置为数据存储空间
    asm("setc  INTM");     //禁止所有中断
    SCSR1=0x81FE;          //CLKIN=10      MHz,CLKOUT=4×CLKIN=40      MHz
    WDCR=0x0E8;           //禁止看门狗，可以用软件禁止看门狗
    IMR=0x0000;           //禁止所有中断
    IFR=0x0FFFF;         //清除全部中断标志，写1清0
    MCRA=MCRA&0x00c7;     //IOPA3、IOPA4、IOPA5 和 IOPB端口配置为一般的 I/O 功能，
                        //F240x 的端口均为8位，MCRA 为16位，因此控制了IOPA
                        //和 IOPB的设置
    PBDATDIR=0x0FF00;     //IOPB 端口设置为输出方式
    PADATDIR=PADATDIR|0x0FF38;
    PADATDIR=PADATDIR&0x0FF00;
    PADATDIR=PADATDIR|0x0FF10;
    while(1)
    {
        PBDATDIR=PBDATDIR&0x00ff;
        k=PBDATDIR&0x00FF;
        if(k==0x00FF)    k=0;
        else    k=1;
        if(k==1)
        {
            for(j=0;j<3000;j++);
            k=PBDATDIR&0x00FF;
            if(k==0x00FF)    k=0;
            else    k=1;
        }
        if(k==1)

```

```

k=PBDATDIR&0x00FF;
switch(k)
{
case 0x00FE:PBDATDIR=(PBDATDIR&0x0FF00) |0x0FF01;break;    / 按 下 S1 点 亮LED0
case 0x00FD:PBDATDIR=(PBDATDIR&0x0FF00)|0x0FF02;break;      /按下S2 点 亮LED1
case 0x00FB:PBDATDIR=(PBDATDIR&0x0FF00)|0x0FF04;break;      //按下S3 点 亮LED2
case 0x00F7:PBDATDIR=(PBDATDIR&0x0FF00)|0x0FF08;break;      //按下S4 点 亮LED3
case 0x00EF:PBDATDIR=(PBDATDIR&0x0FE00) |0x0FF10;break;      // 按 下 S5 点 亮LED4
case 0x00DF:PBDATDIR=(PBDATDIR&0x0FF00)|0x0FF20;break;      //按下S6 点 亮LED5
case 0x00BF:PBDATDIR=(PBDATDIR&0x0FF00)|0x0FF40;break;      //按下S7 点 亮LED6
case 0x007E:PBDATDIR=(PBDATDIR&0x0FF00) |0x0FE80;break;      // 按 下 S8 点 亮LED7
default:PBDATDIR=PBDATDIR;
}

PADATDIR=PADATDIR|0x0FF38;
PADATDIR=PADATDIR&0x0FF00;          //IOPA3,IOPA4 和 IOPA5 设置为输出方式且 LEDCLK=1
PADATDIR=PADATDIR|0x0FF10;

/直接返回中断服务程序
void interrupt nothing()

return;
}

```

8.3.3 光电隔离的8路开关量输入与8路开关量输出电路

下面的设计,利用DSP 的 I/O 端 口 IOPB0~7 与 74LVC245 构成 8 路 开 关 量 输 入 通 道。同时 DSP 的 I/O 端 口 IOPB0~7 与 74HC273 锁 存 器 构 成 8 路 开 关 量 输 出 通 道。其中 DSP 的 I/O 端 口 IOPA3~5, 这 3 个 信 号 为 74LVC138 的 输 入 信 号, 经 138 译 码 后 输 出 一 个 信 号 来 使 能 74LVC245, 从 而 控 制 端 口 IOPB0~7 输 入 8 路 开 关 量 信 号, 或 者 经 138 译 码 后 输 出 一 个 信 号 来 控 制 74HC273 锁 存 器, 从 而 使 I/O 端 口 IOPB0~7 输 出 8 路 开 关 量 信 号 被 锁 存 到 74HC273 的 寄 存 器 中。调 试 时, 可 以 适 当 地 选 择 连 接 光 电 耦 合 二 极 管 串 联 的 电 阻 值, 使 光 电 耦 合 器 件 可 靠 导 通, 实 现 不 同 电 压 等 级 下 的 电 平 匹 配 与 电 气 隔 离。从 而 模 拟 实 际 控 制 过 程 中 的 光 电 隔 离 型 开 关 量 输 入。此 时, 光 电 耦 合 器 件 的 VIN 接 5 V 电 平, 其 IN0~7 为 8 路 开 关 量 输 入。OUT0~7 为 8 路 开 关 量 输 出。开 关 量 输 入 为 低 电 平 时, 输 出 也 为 低 电 平 信 号, 如 图 8.3 所 示。

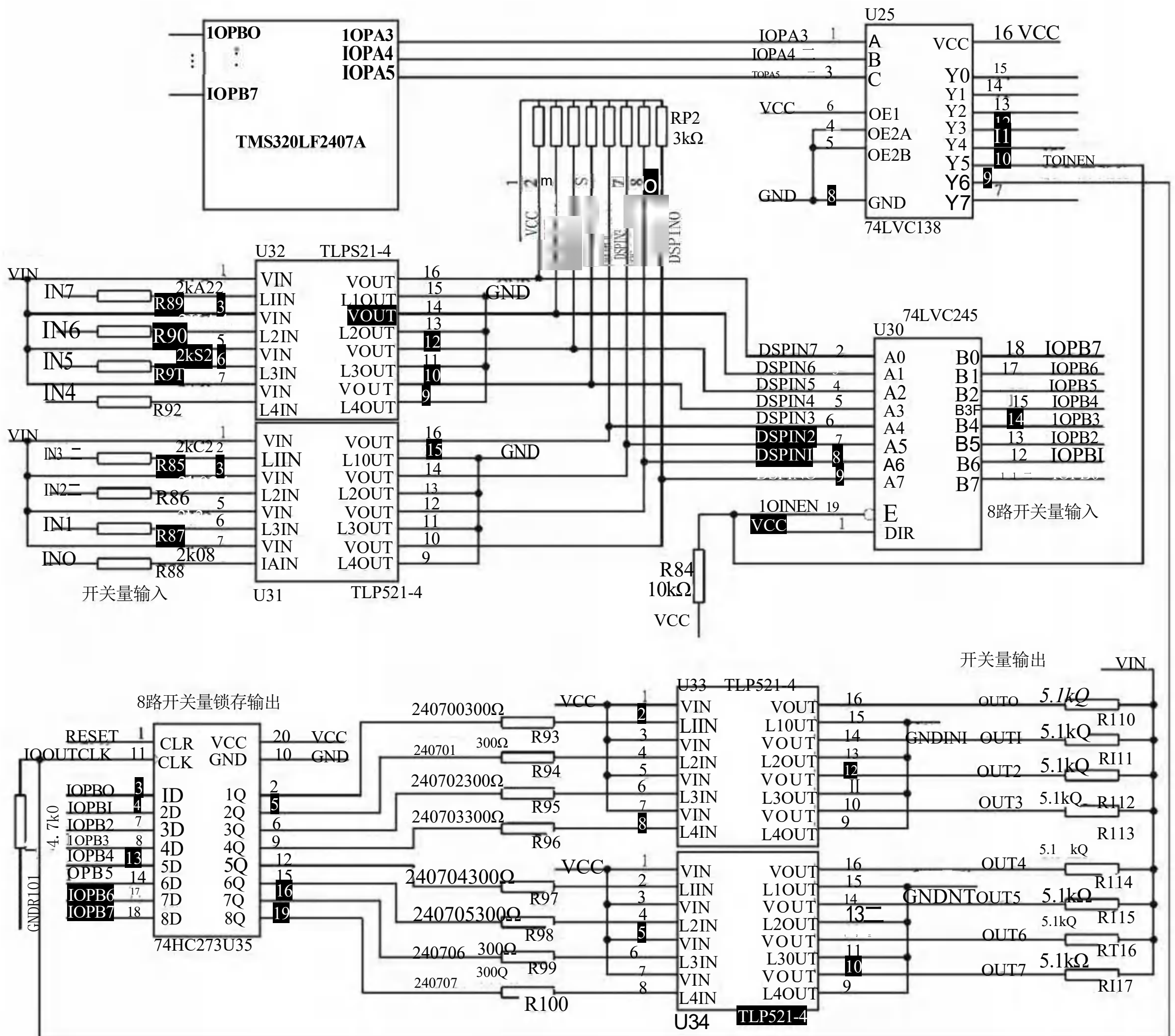


图 8.3 F2407 与光电隔离的8路开关量输入/输出接口电路

C 语言源程序如下：

```
// 该程序用于8路开关量输入和8路开关量输出
#include "register.h"

int j,k;

main()

asm("setc SXM"); //抑制符号位扩展
asm("clrc OVM"); //累加器中结果正常溢出
```



```

asm("clrc  CNF");           //B0 被配置为数据存储空间
asm("setc  INTM");          //禁止所有中断
SCSRI=0x81FE;               /CLKIN=10  MHz.CLKOUT=4×CLKIN=40  MHz
WDCR=0x0E8;                 /禁止看门狗，可以用软件禁止看门狗
IMR=0x0000;                 /禁止所有中断
IFR=0x0FFFF;               //清除全部中断标志，写1清0
MCRA=MCRA&0x00c7;          IOPA3、IOPA4、IOPA5和 IOPB端口配置为一般的I/O 功能
                             //F240x 的端口均为8位，MCRA 为16位，因此控制了IOPA
                             //和 IOPB的设置
PBDATDIR=0x00ff;           //IOPB 端口设置为输入方式
PADATDIR=(PADATDIR&0x00ff)|0x00f28;

                             IOPA3、IOPA4和 IOPA5 设置为输出方式，且IOINEN=1,
                             //选通8路开关量输入

while(1)

    PBDATDIR=PBDATDIR&0x00ff;
    k=PBDATDIR&0x00FF;
    if (k==0x00FF)          k=0;
    else  k=1;
        if(k==1)
            (
                for(j=0;j<3000;j++);
                k=PBDATDIR&0x00FF;
                if(k==0x00FF)  k=0;
                else          k=1;

            )

        if(k==1)

            PADATDIR=(PADATDIR&0x0FF00)|0x0FF30;
            PADATDIR=PADATDIR&0x0FF00;
            X/IOPA3、IOPA4和 IOPA5设置为输出方式，且 IOOUTCLK=1.选通8路开关量输出
            PADATDIR=PADATDIR|0x0FF28;
            //IOPA3、IOPA4和 IOPA5设置为输出方式，且IOINEN=1. 选通8路开关量输入


//中断服务程序直接返回
void  interrupt  nothing()

    return;

```

第 9 章 事件管理器模块

9.1 事件管理器模块概述

每个F240x 器件都包括两个事件管理模块EVA 和 EVB。事件管理器模块包括通用定时器(GP)、比较单元、捕获单元以及正交编码脉冲电路。 EVA 和 EVB 的定时器，比较单元以及捕获单元的功能都相同，只是名称不同，表9.1列出了两个模块的特征、功能以及信号的名称。

事件管理模块 EVA 和 EVB 有相同的外设寄存器，EVA 的起始地址是7400h,EVB 的起始地址是7500h。本节用EVA 描述通用定时器、比较单元、捕获单元以及正交编码脉冲电路的功能。这些功能描述对 EVB 模块也适用，只是模块和信号的名称要随之改变。

表9.1 EVA 和 EVB 模块及其信号名称

事件管理模块	EVA模块	信 号	EVB模块	信 号
GP定时器	Timer 1	T1PWM/T1CMP	Timer 3	T3PWM/T3CMP
	Timer 2	T2PWM/T2CMP	Timer 4	T4PWM/T4CMP
比较单元	Compare 1	PWM1/2	Compare 4	PWM7/8
	Compare 2	PWM3/4	Compare 5	PWM9/10
	Compare 3	PWM5/6	Compare 6	PWM11/12
捕获单元	Capture 1	CAP1	Capture 4	CAP4
	Capture 2	CAP2	Capture 5	CAP5
	Capture 3	CAP3	Capture 6	CAP6
正交编码脉冲电路QEP	QEP1	QEPI	QEP3	QEP3
	QEP2	QEP2	QEP4	QEP4
外部输入	计数方向 外部时钟	TDIRA TCLKINA	计数方向 外部时钟	TDIRB TCLKINB

9.1.1 事件管理器结构框图

事件管理器 EVA 和 EVB 的结构框图如图 9.1 和图 9.2 所示。从图 9.1 中可以看出 EVA 模块使用CAP1/QEP1、CAP2/QEP2 和 CAP3 这3个引脚作为捕获或正交编码器脉冲

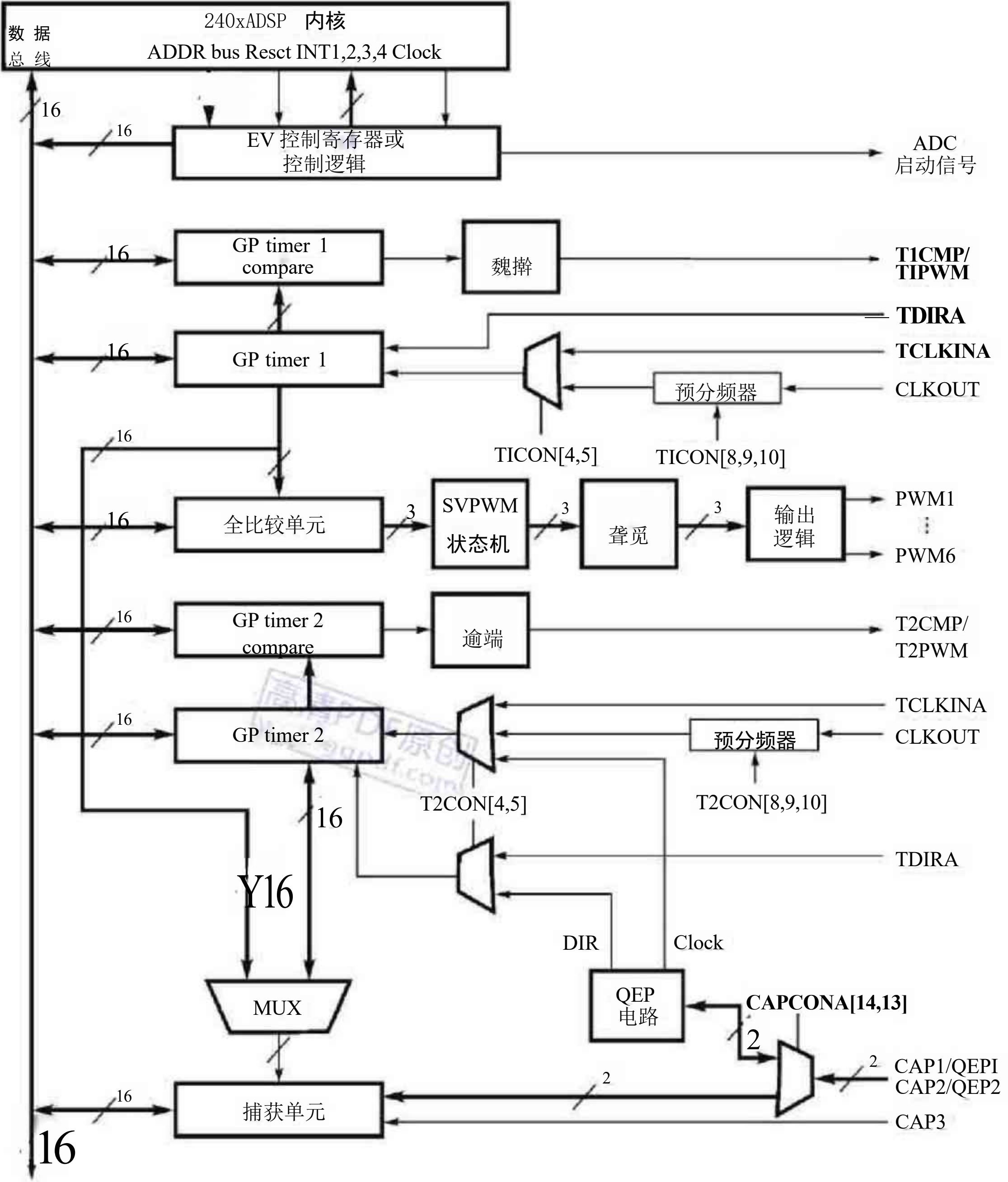


图9 . 1 EVA 结构框图

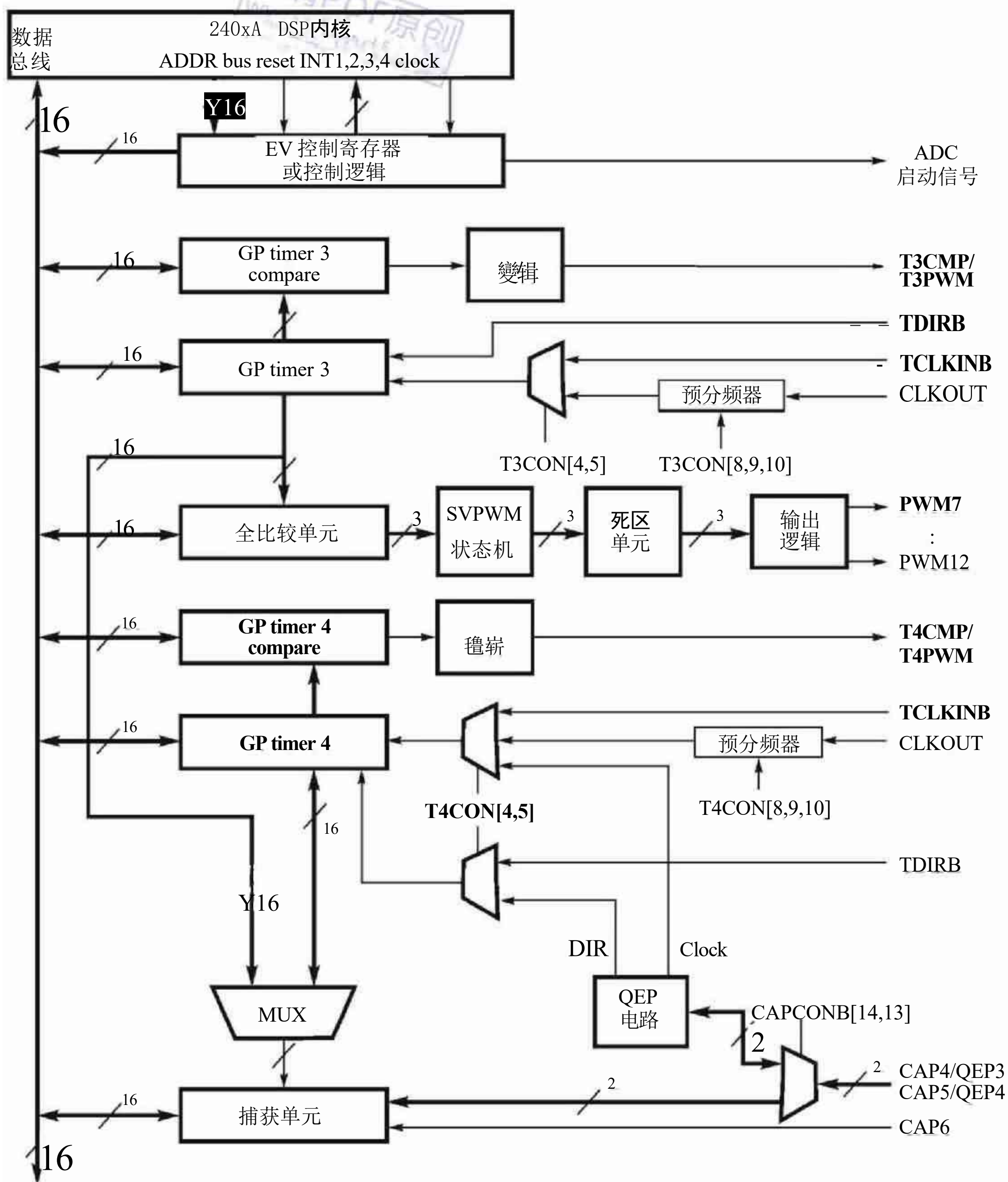


图9.2 EVB 结构框图

的输入脚。

事件管理器模块中的通用定时器可以编程为在外部或内部 CPU 时钟的基础上运行。引脚TCLKINA 提供了外部时钟输入，引脚 TDIRA 用于当通用定时器处于定向增/减计数方式时，规定计数方向。

事件管理器模块中的所有输入都由内部 CPU 协调同步，一次跳变脉冲宽度必须保持到两个 CPU 时钟的上升沿后，才被事件管理器模块所识别。也就是说，如果CPU 时钟已被选作CLKOUT 输出的信号源，则跳变必须保持 CLKOUT 输出的两个下降沿。因此，建议任何跳变脉冲宽度必须保持至少两个 CPU 时钟周期。

9.1.2 事件管理器寄存器地址列表

表9.2~9.9是事件管理寄存器地址的分类列表。EVA 的起始地址是7400h,EVB 的起始地址是7500h。

表9.2 EVA 时间寄存器地址

地 址	寄存器	名 称	
7400h	GPTC()NA	定时器控制寄存器	定时器1
7401h	TICNT	定时器I的计数寄存器	
7402h	TICMPR	定时器1的比较寄存器	
7403h	TIPR	定时器1的周期寄存器	
7404h	TICON	定时器1的控制寄存器	
7405h	T2CNT	定时器2的计数寄存器	定时器2
7406h	T2CMPR	定时器2的比较寄存器	
7407h	T2PR	定时器2的周期寄存器	
7408h	T2CON	定时器2的控制寄存器	

表9.3 EVB 时间寄存器地址

地 址	寄存器	名 称	
7500h	GPTCONB	定时器控制寄存器	定时器3
7501h	T3CNT	定时器3的计数寄存器	
7502h	T3CMPR	定时器3的比较寄存器	
7503h	T3PR	定时器3的周期寄存器	
7504h	T3CON	定时器3的控制寄存器	
7505h	T4CNT	定时器4的计数寄存器	定时器4
7506h	T4CMPR	定时器4的比较寄存器	
7507h	T4PR	定时器4的周期寄存器	
7508h	T4CON	定时器4的控制寄存器	

表9.4 EVA 比较控制寄存器地址

地 址	寄存器	名 称	地 址	寄存器	名 称
7411h	COMCONA	比较控制寄存器	7417h	CMPR1	比较寄存器1
7413h	ACTRA	比较方式控制寄存器	7418h	CMPR2	比较寄存器2
7415h	DBTCONA	死区时间控制寄存器	7419h	CMPR3	比较寄存器3

表9.5 EVB 比较控制寄存器地址

地 址	寄存器	名 称	地 址	寄存器	名 称
7511h	COMCONB	比较控制寄存器	7517h	CMPR4	比较寄存器4
7513h	ACTRB	比较方式控制寄存器	7518h	CMPR5	比较寄存器5
7515h	DBTCONB	死区时间控制寄存器	7519h	CMPR6	比较寄存器6

表9.6 EVA 捕获控制寄存器地址

地 址	寄存器	名 称
7420h	CAPCONA	捕获控制寄存器
7422h	CAPFIFOA	捕获FIFO状态寄存器A
7423h	CAP1FIFO	两级深度的捕获FIFO)堆栈1
7424h	CAP2FIFO	两级深度的捕获FIFO堆栈2
7425h	CAP3FIFO	两级深度的捕获FIFO堆栈3
7427h	CAP1FBOT	FIFO堆栈的栈底寄存器，允许读最近的CAPTURE值
7428h	CAP2FBOT	
7429h	CAP3FBOT	

表9.7 EVB 捕获控制寄存器地址

地 址	寄存器	名 称
7520h	CAPCONB	捕捉控制寄存器
7522h	CAPFIFOB	捕获FIFO状态寄存器B
7523h	CAP4FIFO	两级深度的捕获FIFO堆栈4
7524h	CAP5FIFO)	两级深度的捕获FIFO堆栈5
7525h	CAP6FIF)	两级深度的捕获FIFO堆栈6
7527h	CAP4FBOT	FIFO堆栈的栈底寄存器，允许读最近的CAPTURE值
7528h	CAP5FBOT	
7529h	CAP6FBOT	

表9.8 EVA 中断寄存器地址

地 址	寄存器	名 称	地 址	寄存器	名 称
742Ch	EVAIMRA	中断屏蔽寄存器A	742Fh	EVAIFRA	中断标志寄存器A
742Dh	EVAIMRB	中断屏蔽寄存器B	7430h	EVAIFRB	中断标志寄存器B
742Eh	EVAIMRC	中断屏蔽寄存器C	7431h	EVAIFRC	中断标志寄存器C

表9.9 EVB中断寄存器地址

地 址	寄存器	名 称	地 址	寄存器	名 称
752Ch	EV BIMRA	中断屏蔽寄存器A	752Fh	EV BIFRA	中断标志寄存器A
752Dh	EV BIMRB	中断屏蔽寄存器B	7530h	EV BIFRB	中断标志寄存器B
752Eh	EV BIMRC	中断屏蔽寄存器C	7531h	EV BIFRC	中断标志寄存器C

9.1.3 事件管理器中断

1. 中 断 组

事件管理器中断事件分为3组：事件管理器中断组A、B和 C。每组都有各自不同的中断标志、中断使能寄存器和一些外设事件中断请求。表9. 10和表9. 11分别列出了 EVA 和 EVB 模块相关的中断、中断优先级和中断组。每个 EV 中断组都有一个中断标志寄存器和相应的中断屏蔽寄存器，如表9. 12所列。如果 EVAIMRx(x=A、B 和 C) 相应位是 0，则 EVAIFRx 中的标志位被屏蔽(即不产生中断请求信号)

表9.10 EVA 中断

中断组	中 所	优先级	中断向量 (ID)	描述/中断源	INT
	PDPINTA	I (最高)	0020h	功率驱动保护中断A	1
A	CMPIINT	2	0021h	比较单元1比较中断	2
	CMP2INT	3	0022h	比较单元2比较中断	
	CMP3INT	4	0023h	比较单元3比较中断	
	TIPINT	5	0027h	通用定时器1周期中断	
	T1CINT	6	0028h	通用定时器1比较中断	
	TIUFINT	7	0029h	通用定时器1下溢中断	
	T1 (FINT	8 (最低)	002Ah	通用定时器I上溢中断	

续表9. 10

中断组	中 断	优先级	中断向量 (ID)	描述/中断源	INT
B	T2PINT	1 (最高)	002Bh	通用定时器2周期中断	3
	T2CINT	2	002Ch	通用定时器2比较中断	
	T2UFINT	3	002Dh	通用定时器2下溢中断	
	T2OFINT	4	002Eh	通用定时器2上溢中断	
C	CAP1INT	1 (最高)	0033h	比较单元1中断	4
	CAP2INT	2	0034h	比较单元2中断	
	CAP3INT	3	0035h	比较单元3中断	

表9. 11 EVB 中断

中断组	中 断	优先级	中断向量 (ID)	描述/中断源	INT
	PDPINTB	1 (最高)	0019h	功率驱动保护中断B	1
A	CMP4INT	2	0024h	比较单元4比较中断	2
	CMP5INT	3	0025h	比较单元5比较中断	
	CMP6INT	4	0026h	比较单元6比较中断	
	T3PINT	5	002Fh	通用定时器3周期中断	
	T3CINT	6	0030h	通用定时器3比较中断	
	T3UFINT	7	0031h	通用定时器3下溢中断	
	T3OFINT	8 (最低)	0032h	通用定时器3上溢中断	
B	T4PINT	1 (最高)	0039h	通用定时器4周期中断	3
	T4CINT	2	003Ah	通用定时器4比较中断	
	T4UFINT	3	003Bh	通用定时器4下溢中断	
	T4OFINT	4	003Ch	通用定时器4上溢中断	
C	CAP4INT	1 (最高)	0036h	比较单元4中断	4
	CAP5INT	2	0037h	比较单元5中断	
	CAP6INT	3	0038h	比较单元6中断	

当外设中断请求信号被CPU 接受时，PIE 控制器将相应的外设中断向量装入到外设中断向量寄存器(PIVR) 中。

外设中断向量寄存器(PIVR) 中的值可以区分，该组哪一个悬挂的中断具有最高优先级。外设中断向量寄存器中的值可从中断服务子程序 (ISR) 中读出。

表9.12 EV 中断标志寄存器及相应的中断屏蔽寄存器

标志寄存器	屏蔽寄存器	EV模块	标志寄存器	屏蔽寄存器	EV模块
EVAIFRA	EVAIMRA	EVA	EVBI FRA	EVBI MRA	EVB
EVAIFRB	EVAIMRB		EVBI FRB	EVBI MRB	
EVAIFRC	EVAIMRC		EVBI FRC	EVBI MRC	

2. 中 断 产 生

若事件管理器模块中产生一个中断事件，则其中一个事件管理器中断标志寄存器的相应标志位就置1。如果标志位局部未被屏蔽(EVAIMRx 中的相应位置1)，那么外设中断扩展控制器(PIE) 就产生一个外设中断请求。

3. 中 断 向 量

当中断请求被 CPU 接受时，已置位的中断标志中，具有最高优先级的中断标志相应的中断向量被装载到累加器中。

在中断服务程序(ISR) 中读取中断向量，中断标志位必须在中断服务程序中，用软件直接向中断标志寄存器中的相应位置1来清除。如果中断标志位未被清除，则以后该中断就不再产生中断请求。

4. 中 断 处 理

当事件管理器中断请求被接受后，必须将外设中断向量寄存器(PIVR) 读入累加器，并左移一位或几位，然后将偏移地址(中断子向量入口表的开始地址)加至累加器。再使用 BACC 指令来跳转到相应的中断地址，另有一条指令从表中转移到相应的中断源的中断服务子程序。此处理过程将引起一个典型20个CPU 周期的延迟。该延迟是指从中断产生到相应的中断服务程序中的第一条指令被执行之间的时间。

如果需要进行最小的保护现场，则该延迟为25个 CPU 周期。

如果一个事件管理器中断组只允许一个中断，则这个延迟可以减少到最少8个 CPU 周期。

如果不考虑存储器空间，则可将该延迟减少到16个CPU 周期，而无须要求每个事件管理器中断组只允许一个中断。

5. 事件管理器中断标志寄存器

事件管理器中断寄存器的地址列表如表9. 8和表9. 9所列。这些寄存器都是16位存储器映射寄存器。当软件读这些寄存器时，未使用的位读出值为0。向未使用的位写入则无影响。因为事件管理器中断标志寄存器(EVxIFRx) 是可读寄存器，所以当中断被屏蔽时，可通过软件查询事件管理器中断标志寄存器中相应的位，来监测中断事件的发生。

1)EVA 中断标志寄存器 A(EVAIFRA) —— 地 址742 Fh

15~11		10		9	8
保留位		T1OFINT FLAG	T1UFINT FLAG	T1CINT FLAG	
R_0		RW1C_0	RW1C0	RW1C_0	
7	6~4	3	2	1	0
T1PINT FLAG	保留位	CMP3INT FLAG	CMP2INT FLAG	CMP1INT FLAG	PDPINTA FLAG
RW1C_0	RW1C_0	RW1C_0	RW1C_0	RW1C0	RW1C_0

注：R= 可读；W1C= 写1清除；0-多位值。

位15~11	保留位。			
位 10	T1OFINT	FLAG。通用定时器1上溢中断标志位。		
	读：0	标志被复位；	写：0	无效；
	1	标志被置位。	1	复位标志位。
位 9	T1UFINT	FLAG。通用定时器1下溢中断标志位。		
	读：0	标志被复位；	写：0	无效；
	1	标志被置位。	1	复位标志位。
位 8	T1CINT	FLAG。通用定时器1比较中断标志位。		
	读：0	标志被复位；	写：0	无效；
	1	标志被置位。	1	复位标志位。
位 7	T1PINT	FLAG。通用定时器1周期中断标志位。		
	读：0	标志被复位；	写：0	无效；
	1	标志被置位。	1	复位标志位。
位6~4	保留位。			
位 3	CMP3INT。	比较单元3中断标志位。		
	读：0	标志被复位；	写：0	无效；
	1	标志被置位。	1	复位标志位。
位 2	CMP2INT。	比较单元2中断标志位。		
	读：0	标志被复位；	写：0	无效；
	1	标志被置位。	1	复位标志位。
位 1	CMP1INT。	比较单元1中断标志位。		
	读：0	标志被复位；	写：0	无效；
	1	标志被置位。	1	复位标志位。

位 0 PDPINTA。 功率驱动保护中断标志位。
读： 0 标志被复位； 写： 0 无 效；
 1 标志被置位。 1 复位标志位。

(2) EVA 中断标志寄存器 BEVAIFRB) —— 地址7430 h

15~4	3	2	1	0
保留位	T2OFINT FLAG	T2UFINT FLAG	T2CINT FLAG	T2PINT FLAG
R_0	RW1C_0	RWIC0	RW1C_0	RWIC_0

位15~4 保留位。
位 3 T2OFINT FLAG。通用定时器2上溢中断标志位。
读： 0 标志被复位； 写： 0 无效；
 1 标志被置位。 1 复位标志位。
位 2 T2UFINT FLAG。通用定时器2下溢中断标志位。
读： 0 标志被复位； 写： 0 无效；
 1 标志被置位。 1 复位标志位。
位 1 T2CINT FLAG。通用定时器2比较中断标志位。
读： 0 标志被复位； 写： 0 无效；
 1 标志被置位。 复位标志位。
位 0 T2PINT FLAG。通用定时器2周期中断标志位。
读： 0 标志被复位； 写： 0 无效；
 1 标志被置位。 1 复位标志位。

(3)EVA 中断标志寄存器 C(EVAIFRC)—— 地 址7431h

15~3	2	1	0
保留位	CAP3INT FLAG	CAP2INT FALG	CAP1INT FLAG
R_0	RWIC0	RWIC 0	RWIC_0

位15~3 保留位。
位 2 CAP3INT FLAG。捕获单元3中断标志位。
读： 0 标志被复位； 写： 0 无效；
 1 标志被置位。 1 复位标志位。
位 1 CAP2INT FLAG。捕获单元2中断标志位。

读：0 标志被复位； 写：0 无效；
1 标志被置位。 1 复位标志位。

位 0 CAPIINT FLAG。捕获单元1中断标志位。

读：0 标志被复位； 写：0 无效；
1 标志被置位。 1 复位标志位。

(4)EVA 中断屏蔽寄存器 A(EVAIMRA)——地址742 Ch

15~11		10		9	8
保留位		T1(FINT ENABLE		T1UFINT ENABLE	T1CINT ENABLE
R_Q		RW_0		RW_Q	RW_0
7	6~4	3	2	1	0
T1PINT ENABLE	保留位	CMP3INT ENABLE	CMP2INT ENABLE	CMPIINT ENABLE	PDPINTA ENABLE
RW_0	R0	RW_0	RW_0	RW_0	RW_0

位15~11 保留位。

位10 T1OFINT ENABLE。通用定时器1上溢中断使能位。
0 禁止； 1 使能。

位 9 T1UFINT ENABLE。通用定时器1下溢中断使能位。
0 禁止； 1 使能。

位 8 T1CINT ENABLE。通用定时器1比较中断使能位。
0 禁止； 1 使能。

位 7 T1PINT ENABLE。通用定时器1周期中断使能位。
0 禁止； 1 使能。

位6~4 保留位。

位 3 CMP3INT ENABLE。比较单元3中断使能位。
0 禁止； 1 使能。

位 2 CMP2INT ENABLE。比较单元2中断使能位。
0 禁止； 1 使能。

位 1 CMP1INT ENABLE。比较单元1中断使能位。
0 禁止； 1 使能。

位 0 PDPINTA ENABLE。功率驱动保护中断使能位。
0 禁止； 1 使能。

(5) EVA 中断屏蔽寄存器 **BEVAIMRB**) —— 地 址**742Dh**

15~4		3	2	1	0
保留位		T2OFINT ENABLE	T2UFINT ENABLE	T2CINT ENABLE	T2PINT ENABLE
R_0		RW1C0	RWIC0	RW1C0	RWIC_0
位15~4	保留位。				
位 3	T2OFINT ENABLE。通用定时器2上溢中断使能位。 0 禁止； 1 使能。				
位 2	T2UFINT ENABLE。通用定时器2下溢中断使能位。 0 禁止； 1 使能。				
位 1	T2CINT ENABLE。通用定时器2比较中断使能位。 0 禁止； 1 使能。				
位 0	T2PINT ENABLE。通用定时器2周期中断使能位。 0 禁止； 1 使能。				

6)EVA 中断屏蔽寄存器**C(EVAIMRC)**— 地址**742Eh**

15~3		2	1	0
保留位		CAP3INT ENABLE	CAP2INT ENABLE	CAP1INT ENABLE
R0		RW_o	RW_0	RW_0
位15~3	保留位。			
位 2	CAP3INT ENABLE。捕获单元3中断使能位。 0 禁止； 1 使能。			
位 1	CAP2INT ENABLE。捕获单元2中断使能位。 0 禁止； 1 使能。			
位 0	CAP1INT ENABLE。捕获单元1中断使能位。 0 禁止； 1 使能。			

由于事件管理器EVB 模块的功能与EVA 模块一样，只是把定时器1、2改为定时器3、4，比较单元1、2、3改为比较单元4、5、6，捕获单元1、2、3改为捕获单元4、5、6，因此在此只给出EVB 模块中断寄存器单元地址。

(7)EVB 中断标志寄存器 A(EVBIFRA)— 地址752 Fh

15~11		10		9	8
保留位		T3OFINT FLAG	T3UFINT FLAG	T3CINT FLAG	
R_0		RW1C0	RW1C0	RW1C_0	
7	6~4	3	2	1	0
T3PINT FLAG	保留位	CMP6INT FLAG	CMP5INT FLAG	CMP5INT FLAG	PDPINTB FLAG
RW1C_0	R_0	RW1C_0	RW1C0	RW1C_0	RW1C_0

(8)EVB 中断标志寄存器 B(E VBIFRB)—— 地址7530 h

15~4		3	2	1	0
保留位		T4OFINT FLAG	T4UFINT FLAG	T4CINT FLAG	T4PINT FLAG
R_0		RW1C_0	RW1C_0	RW1C_0	RW1C_0

9)EVB 中断标志寄存器 C(EVBIFRC)— 地址7531h

15~3		2	1	0
保留位		CAP3INT ENABLE	CAP2INT ENABLE	CAP1INT ENABLE
R_0		RW_0	RW_0	REW_0

(10)EVB 中断屏蔽寄存器 A(EVBIMRA) ——地址 752 Ch

15~11		10		9	8
保留位		T3OFINT ENABLE	T3UFINT ENABLE	T3CINT ENABLE	
R_0		RW_0	RW_0	RW_0	
7	6~4	3	2	1	0
T3PINT ENABLE	保留位	CMP6INT ENABLE	CMP5INT ENABLE	CMP4INT ENABLE	PDPINTB ENABLE
RW_0	R_0	RW_0	RW_0	RW_0	RW_0

(11)EVB 中断屏蔽寄存器B(EVBIMRB) —— 地 址 75 2Dh

15~4	3	2	1	0
保留位	T4OFINT ENABLE	T4UFINT ENABLE	T4CINT ENABLE	T4PINT ENABLE
R_0	RW0	RW 0	RW 0	RW 0

(12)EVB 中断屏蔽寄存器 C(E VBIMRC) — 地 址752Eh

15-3	2	i	0
保留位	CAP6INT ENABLE	CAP5INT ENABLE	CAP4INT ENABLE
R_0	RW_0	RW_0	RW_0

9.2 通用定时器

9.2.1 通用定时器概述

事件管理模块有两个通用的可编程定时器。定时器 x (对 于EVA. x=1. 2; 对 于EVB. x =3, 4) 包括：

- 一个16位的定时器增/减计数的计数器 TxCNT， 可读/写；
- 一个16位的定时器比较寄存器(双缓冲，带影子寄存器) TxCMPR， 可读/写；
- 一个16位的定时器周期寄存器(双缓冲，带影子寄存器) TxPR， 可读/写；
- 一个16位的定时器控制寄存器 TxCON， 可读/写；
- 可选择的内部或外部输入时钟；
- 用于内部或外部时钟输入的可编程的预定标器(prescaler)；

控制和中断逻辑用于4个可屏蔽的中断——下溢、上溢、定时器比较和周期中断；

- 可选择的输入引脚TDIRx （当用双向计数方式时用来选择向上或向下计数）。

通用定时器的方框图如图9 . 3所示。

9.2.2 通用定时器功能模块

各个GP 定时器之间可以彼此独立工作或相互同步工作。与每个 GP 定时器有关的比较寄存器可用作比较功能或 PWM 波形发生。对每个 GP 定时器在增/减计数方式中，有3种连续的工作方式。每个 GP 定时器的内部或外部的输入时钟都可进行可编程的预定标。 GP 定时器还向事件管理器的子模块提供时基。GP 定时器1为所有的比较和PWM 电路提供时基，

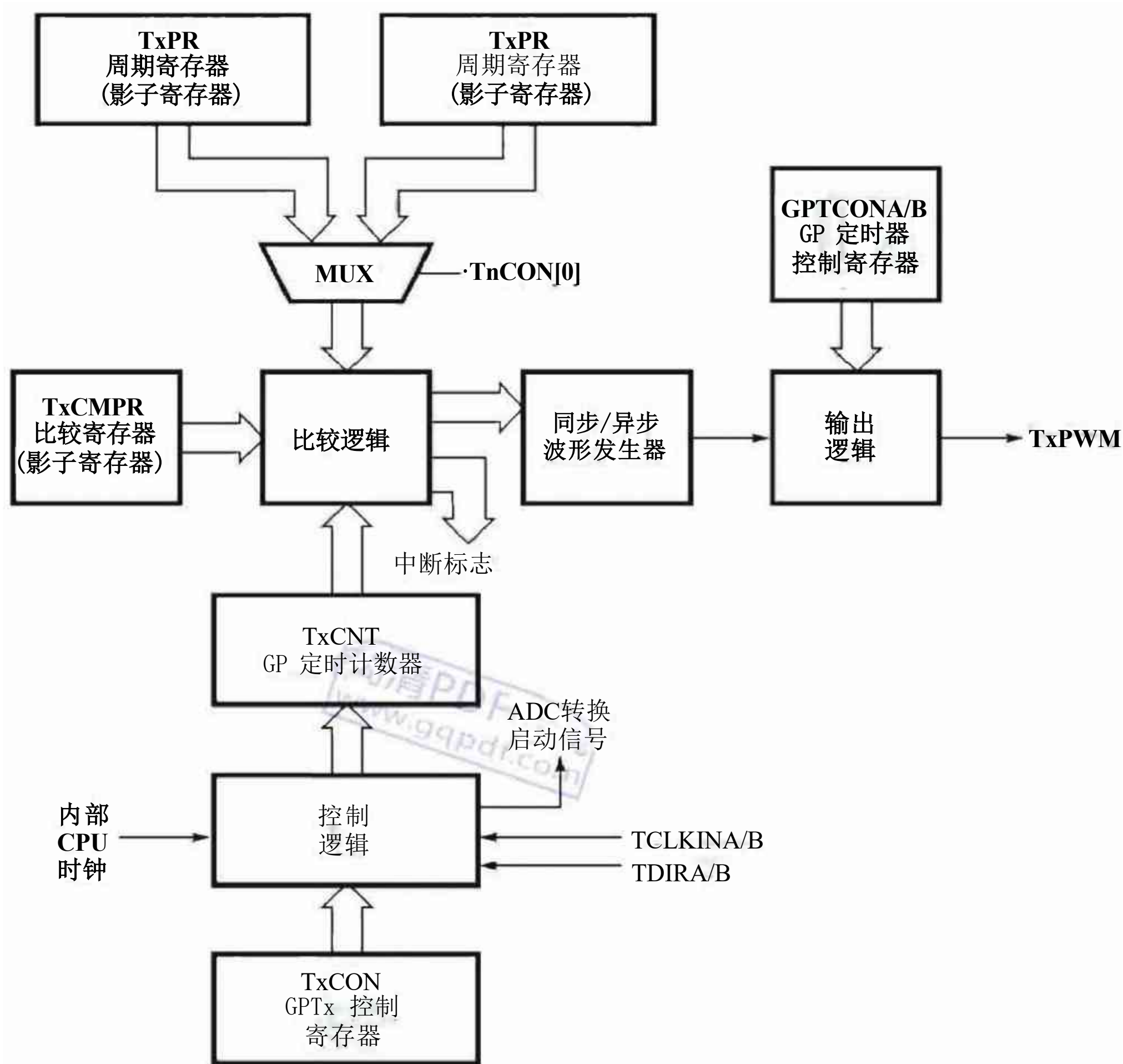


图9.3 通用定时器方框图 (x=2 或 4)

(当 x=2 时, y=1 且 n=2; 当 x=4 时 y=3 且 n=4)

GP 定时器2或1为捕获单元和正交脉冲计数操作提供时基。周期寄存器和比较寄存器的双缓冲，允许根据需要编程改变 PWM 周期和 PWM 脉冲宽度。

控制寄存器 GPTCONA/B 规定了通用定时器针对不同的定时器事件所采取的操作，并且指明了所有4个通用定时器的计数方向。虽然写状态位的信息不起作用，但是 GPTCONA/B 是可读/写的。

1. 通用定时器输入

通用定时器的输入如下：

- > 内部 CPU 时钟；
- > 外部时钟 TCLKINA/B，最大频率是 CPU 时钟频率的1/4；

- 方向输入 TDIRA/B. 被通用定时器用于指示增/减计数模式;
- 复位信号 RESET。

另外, 当一个通用定时器与正交编码器脉冲电路一起使用时, 正交编码器脉冲电路同时产生定时器的时钟和计数方向。

2. 通用定时器输出

通用定时器的输出如下:

- 通用定时器比较输出TxCMP,x=1.2.3,4;
- 至ADC 模块的A/D 转换启动信号;
- 比较逻辑和比较单元的下溢、上溢、比较匹配和周期匹配信号;
- 计数方向指示位。

3. 单个通用定时器控制寄存器

通用定时器的操作模式由其控制寄存器 TxCON 决定。寄存器 TxCON 中的各位意义如下:

- 通用定时器处于4种计数模式中的哪一种;
- 通用定时器使用外部时钟还是内部CPU 时钟;
- 输入时钟使用8种预定标因子(范围为1/128~1)中的哪一种;
- 何种条件下重装载定时器比较寄存器;
- 通用定时器是否使能;
- 通用定时器的比较操作是否使能;
- 通用定时器2使用它自身的还是通用定时器1的周期寄存器(EVA);
- 通用定时器4使用它自身的还是通用定时器3的周期寄存器(EVB)。

定时器x 控制寄存器 (TxCON:x=1,2,3,4) —— 地址7404h(T1CON),7408h(T2CON), 7504h(T3CON),7508h(T4CON)

15	14	13	12	11	10	9	8
Free	Soft	保留位	TMODE1	TMODE0	TPS2	TPS1	TPSo
RW_0	RW_0	RW_0	RW_0	RW_0	RW_0	RW_0	RW_0
7	6	5	4	3	2	1	0
T2SWT1/ T4SWTI+	TENABLE	TCLKS1	TCLKS0	TCLD1	TCLD0	TECMPR	SELTIPR/ SELT3PR#
RW_0	RW 0	RW 0	RW0	RW 0	RW 0	RW0	RW0

注: R= 可读; W= 可写; 0=复位值。

位15~14	Free/Soft。 仿真控制位。
	00 一旦仿真悬挂，立即停止；
	01 一旦仿真悬挂，在当前定时器周期结束后停止；
	10 操作不受仿真悬挂的影响；
	11 操作不受仿真悬挂的影响。
位13	保留位，读为0，写无影响。
位12~11	TMODE1/TMODE0。 计数模式选择。
	00 停止/保持；
	01 连续增/减计数模式；
	10 连续增计数模式；
	11 定向增/减计数模式。
位10~8	TPS2~TPS0。 输入时钟预定标系数。
	000 X/1; 001 X/2;
	010 X/4; 011 X/8;
	100 X/16; 101 X/32;
	110 X/64; 111 X/128。
	X=CPU 时钟频率。
位 7	T2SWT1/T4SWT1。 定时器2、4周期寄存器选择位。
	0 定时器2、4使用自身的周期寄存器；
	1 使用T1CON、T3CON 中的定时器使能位来使能或禁止相应操作，从而忽略自身的定时器使能位。
位 6	TENABLE。 定时器使能位。
	0 禁止定时器操作，即定时器被置于保持状态且预定标器被复位；
	1 使能定时器操作。
位5~4	TCLKS1/TCLKS0。 时钟源选择。
	00 内部CPU时钟；
	01 外部时钟；
	10 保留；
	11 正交编码脉冲电路——只适用于定时器2和定时器4，在定时器1、3中保留，这种操作只在SELT1PR=0 时有效。
位3~2	TCLD1/TCLD0。 定时器比较寄存器(如果有效)的重装载条件。
	00 当计数值是0时重装载；
	01 当计数值是0或等于周期寄存器值时重装载；
	10 立即重装载；

- 11

保留。
- 位 1

TECMP。

定时器比较使能位。

0

禁止定时器比较操作；

1

使能定时器比较操作。
- 位 0

SELT1PR/SELT3PR。

周期寄存器选择，在定时器2.4中有效，定时器1、3中保留。

0

使用自身的周期寄存器；

1

使用T1PR（在EVA 模块）或T3PR（在EVB 模块）作为周期寄存器，而忽略自身的周期寄存器。

4. 全局通用定时器控制寄存器

全局通用定时器控制寄存器(GPTCONA/B) 规定了通用定时器针对不同定时器事件所采取的操作，并指明了它们的计数方向。

(1)全局通用定时器控制寄存器A(GPTCONA)—— 地址7400 h

15	14	13	12~11	10~9	8~7
保留位	T2STAT	T1STAT	保留位	T2TOADC	T1TOADC
RW_0	R_1	R_1	RW_0	RW_0	RW_0
6			5~4	3~2	1~0
TCOMPOE		保留位		T2PIN	T1PIN
RW_D		RW_0		RW_0	RW_0

- 位15

保留位。
- 位14

T2STAT。

通用定时器2的状态，只读。

0

减计数；

1

增计数。
- 位13

T1STAT。

通用定时器1的状态，只读。

0

减计数；

1

增计数。
- 位12~11

保留位。
- 位10~9

T2TOADC。

通用定时器2启动A/D 转换事件。

00

无事件启动 A/D 转换；

01

设置由下溢中断标志来启动 A/D 转换；

10

设置由周期中断标志来启动 A/D 转换；

11

设置由比较中断标志来启动A/D 转换。

- 位8~7 T1TOADC。通用定时器1启动 A/D 转换事件。
- 00 无事件启动A/D 转换；
- 01 设置由下溢中断标志来启动 A/D 转换；
- 10 设置由周期中断标志来启动 A/D 转换；
- 11 设置由比较中断标志来启动 A/D 转换。
- 位 6 TCOMP0E。比较输出允许。如果 PDPINT_x 有效，则该位设置为0。
- 0 禁止所有通用定时器比较输出(所有比较输出都置成高阻态)；
- 1 使能所有通用定时器比较输出。
- 位5~4 保留位。
- 位3~2 T2PIN。通用定时器2比较输出极性。
- 00 强制低；
- 01 低有效；
- 10 高有效；
- 11 强制高。
- 位1~0 T1PIN。通用定时器1比较输出极性。
- 00 强制低；
- 01 低有效；
- 10 高有效；
- 11 强制高。

由于全局通用定时器控制寄存器 B(GTPCONB) 的内容与GTPCONA 一样，只是把定时器1、2改为定时器3、4。在此只给出了其寄存器单元地址。

(2) 全局通用定时器控制寄存器 **B(GPTCONB)** —— 地 址**7500 h**

15	14	13	12~11	10~9	8~7
保留位	T4STAT	T3STAT	保留位	T4TOADC	T3TOADC
RW_0	R_1	R_1	RW_0	RW_0	RW_0
6	5~4	3~2	1~0		
TCOMP0E	保留位	T4PIN	T3PIN		
RW_0	RW0	RW_0	RW_0		

5. 通用定时器的比较寄存器

与通用定时器相关的比较寄存器存储持续与通用定时器的计数器进行比较的值。当发生匹配时，将产生以下事件：

- ＞ 根据GPTCONA/B 位的设置不同，相关的比较输出发生跳变，或启动ADC；

＞相应的中断标志将被置位；

若中断未屏蔽，则产生外设中断请求。

通过设置 TxCON 寄存器的相关位，可以使能或禁止比较操作。比较操作和输出适用于任何一种定时模式.当然也包括QEP 模式。

通用定时器比较寄存器的地址为：7402h(T1CMPR)、7406h(T2CMPR)、7502h(T3CMPR)和7506h(T4CMPR)。

6. 通用定时器的周期寄存器

通用定时器周期寄存器的值决定了定时器的周期。当周期寄存器的值与定时器计数器的值产生匹配时，通用定时器的操作就停止并保持其当前值.并根据计数器所处的计数方式执行复位为零或开始递减计数。

通用定时器的周期寄存器的地址为：7403h(T1PR)、7407h(T2PR)、7503h(T3PR) 和 7507h(T4PR)。

7. 通用定时器的比较和周期寄存器的两级缓存

通用定时器的比较寄存器TxCMPR 和周期寄存器TxPR 是带有影子寄存器的。在一个周期中的任一时刻一个新值，可以写到这两个寄存器的任一个中。注意，新值是被写到相应的影子寄存器中的。对比较寄存器来说，仅当由 TxCON 寄存器所规定的某一个特定定时事件发生时，影子寄存器的内容才加载到工作的比较寄存器中；对周期寄存器来说，仅当计数寄存器TxCNT 为0时，工作的周期寄存器才重新加载其影子寄存器的值。比较寄存器的加载的情况可能是下列情况之一：

＞在写信息到影子寄存器后立即加载；

下溢时，即通用定时器计数器的值为0时；

＞下溢或周期匹配时，即当计数器的值为0或计数器的值与周期寄存器的值相等时。

周期寄存器和比较寄存器的双缓存特点，允许应用代码在一个周期中的任何时候，都可以更新周期寄存器和比较寄存器，这将改变下一个周期的定时器周期和 PWM 的脉冲宽度。对于 PWM 发生器来说，定时器周期值的高速变化，就意味着载波频率的高速变化。

注意：

＞初始化周期寄存器。通用定时器的周期寄存器应该在计数器被初始化为一个非0值之前进行初始化；否则.周期寄存器的值将保持不变，直到下一次下溢发生。

＞当相应的比较操作被禁止时，比较寄存器是透明的，即新装入值直接进入工作的比较寄存器。这适用于事件管理器的所有比较寄存器。

8. 通用定时器的比较输出

通用定时器的比较输出可规定为高有效、低有效、强制高或强制低，这取决于 GPTCON 寄存器的相关位是如何配置的。当它为高有效/低有效时，在第一次比较匹配发生时比较输出由

(2)EVB 模块

- 置 T3CON 寄存器中的 TENABLE 位为 1，且置 T4CON 寄存器中的 T4SWT3 位为 1。此时，将同时启动两个定时器的计数器。

在启动同步操作前，将通用定时器 3 和 4 的定时计数器初始化成不同的值。

- 置 T4CON 寄存器中的 SELT3PR 位为 1，使通用定时器 4 将通用定时器 3 的周期寄存器作为它自己的周期寄存器(忽略它自身的周期寄存器)使用。

这就允许通用定时器事件之间的有效同步。因为每一个通用定时器从其计数寄存器中的当前值开始计数操作，所以一个通用定时器被编程为其他通用定时器启动之后，会延时一段已知的时间再启动。

13. 通用定时器事件启动 A/D 转换

GPTCONA/B 寄存器的位可以规定 A/D 转换器的启动信号，由通用定时器的哪些事件来产生，如下溢、比较匹配或周期匹配。这种特点允许在没有 CPU 干涉的情况下，为通用定时器事件和 A/D 转换之间的启动提供同步。

14. 仿真悬挂时的通用定时器

通用定时器的控制寄存器 TxCON 还定义了仿真悬挂期间的通用定时器操作。当仿真中断发生时，这些位被设置为允许通用定时器操作持续运行，也就使得在线仿真成为可能。当仿真中断出现时，这些位也可被设置为规定的通用定时器立即停止操作，或是在当前计数周期完成后停止操作。

当内部 CPU 时钟被仿真器终止时，例如，当仿真时遇到一个断点时，仿真悬挂就发生了。

15. 通用定时器的中断

通用定时器在 EVAIFRA、EVAIFRB、EVBIFRA 和 EVBIFRB 中有 12 个中断标志。每个通用定时器可根据以下事件产生 4 个中断：

- 上溢 — TxOFINF(x=1、2、3 或 4)；
- 下溢 — TxUFINF(x=1、2、3 或 4)；
- 比较匹配 — TxCINT(x=1、2、3 或 4)；
- 周期匹配 — TxPINT(x=1、2、3 或 4)。

当通用定时计数器的值与比较寄存器的值相同时，就产生定时器比较事件(匹配)。如果比较操作被使能，则相应的比较中断标志，在匹配后的两个 CPU 时钟周期被置位。

定时计数器的值达到 FFFFh 时，就产生上溢事件；定时计数器的值达到 0000h 时，就产生下溢事件。类似地，当定时计数器的值与周期寄存器的值相同时，就产生一个周期事件。在每个事件发生后的两个 CPU 周期后，定时器的上溢、下溢和周期中断标志位将被置位。

9.2.3 通用定时器的计数操作

每个通用定时器有4种可选择的操作模式：

- > 停止/保持模式；
- > 连续增计数模式；
- ▶ 定向增/减计数模式；
- > 连续增/减计数模式。

相应的定时器控制寄存器TxCON 中位的形式决定通用定时器的计数模式。定时器使能位(TxCON[6], 即 TENABLE) 使能或禁止定时器的计数操作。当定时器被禁止时, 定时器的计数操作停止, 并且定时器的预定标器被复位为 x/1; 当定时器使能时, 定时器按照寄存器TxCON 中的其他位(TxCON[12~11]) 确定其计数模式, 并开始计数。

1. 停止/保持模式

在这种模式下, 通用定时器的操作停止并保持当前状态, 定时器的计数器、比较输出和预定标计数器都保持不变。

2. 连续增计数模式

在这种模式下, 通用定时器将按照定标的输入时钟计数, 直到其计数器的值与周期寄存器的值相等为止。匹配之后的下一个输入时钟的上升沿到来时, 通用定时器复位为0, 并开始另一个计数周期。

在定时计数器与周期寄存器匹配后的两个 CPU 时钟周期时, 周期中断标志被置位。如果外设中断未被屏蔽, 那么将产生外设中断请求。如果该周期中断已通过GPTCONA/B 寄存器中的相应位设置去启动 A/D 转换器, 那么在中断标志被设置的同时, A/D 转换启动信号就被送到 A/D 转换模块。

通用定时器变成0的两个CPU 时钟周期之后, 定时器的下溢中断标志位被置位。如果该周期中断已通过 GPTCONA/B 寄存器中的相应位设置去启动A/D 转换器, 那么在中断标志被设置的同时, A/D 转换启动信号就被送到 A/D 转换模块。

在定时计数器的值达到FFFFh 后, 定时器的上溢中断标志位在两个CPU 时钟周期后被置位。如果外设中断未被屏蔽, 那么将产生外设中断请求。

除了第一个周期外, 定时器周期的时间为 TxPR+1 个定标的时钟输入周期。如果定时器计数器开始计数时为0, 那么第一个周期的时间也与以后的周期时间相同。

通用定时器的初始值可以是0000h~FFFFh 中的任意值。当该初始值大于周期寄存器的值时, 定时器将计数至FFFFh, 复位为0后将继续以上操作, 好像初始值是0一样; 当该初始值等于周期寄存器的值时, 定时器将置位周期中断标志, 计数器复位为0, 设置下溢中断标志并继续以上操作, 好像初始值是0一样; 当定时器的值在0和周期寄存器的值之间时, 定时

器将计数到周期寄存器的值，并且继续完成该计数周期，就好像计数器的初始值与周期寄存器的值相同一样。

在该模式下，GPTCONA/B 寄存器中的定时器计数方向指示位为1。无论是内部 CPU 时钟或是外部时钟输入都可选作定时器的输入时钟。在这种计数模式下，TDIRA/B 引脚输入将被通用定时器忽略。

通用定时器的连续增计数模式特别适用于边沿触发或异步 PWM 波形的产生，也适用于许多电机和运动控制系统的采样周期的产生。

通用定时器连续增计数模式(TxPR=3 或2)如图9.4所示。从计数器达到周期寄存器的值，到它开始另一个计数周期的过程中没有丢失一个时钟周期。

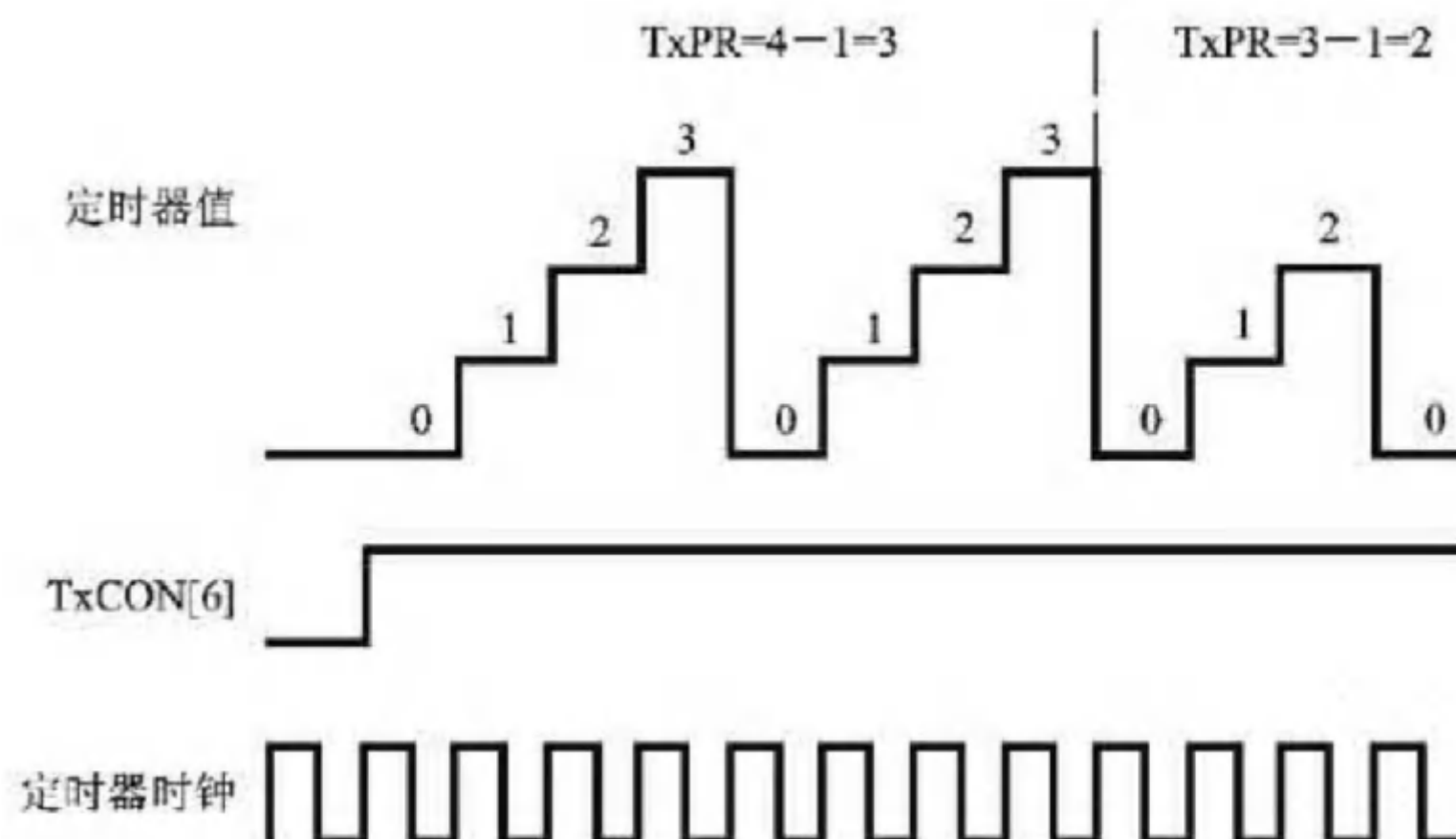


图9.4 通用定时器连续增计数模式

【例9.1】通用定时器1在连续增计数模式下的初始化示例。

```
LDP      #DP EVA           ;指向7400h~7480h 单元
SPLK     #41H,GPTCONA      ;TCOMPOE=1允许定时器比较输出
SPLK     #5H,TIPR          ;设置周期寄存器
SPLK     #3H,T1CMPR        ;设置比较寄存器
SPLK     #0H,T1CNT         ;设置计数寄存器
SPLK     #074EH,T1CON      ;TNODE=10 连续增计数模式，TPS=111 预分频为128
                          ;TENABLE=1 定时器计数使能，TCLKS=00内部时钟
                          ;TECMPR=1 定时器1比较使能，SELT1PR=0
```

3. 定向增/减计数模式

通用定时器在定向增/减计数模式中，将根据定标的时钟和 TDIRA/B 引脚的输入来增或减计数。当引脚 TDIRA/B 保持为高时，通用定时器将增计数直到计数值达到周期寄存器的值。当定时器的值等于周期寄存器的值或FFFFh，并且引脚 TDIRA/B 保持高时，定时器的

计数器复位到0并继续增计数到周期寄存器的值。当引脚 TDIRA/B 保持为低时，通用定时器将减计数直到计数值为0。当定时器的值为0，并且引脚 TDIRA/B 保持为低时，定时器的计数器重新载入周期寄存器的值，开始新的减计数。

定时器的初始值可以为0000h~FFFFh 中的任何值。当定时器的初始值大于周期寄存器的值时，如果引脚 TDIRA/B 保持为高，那么定时器的计数器增计数到 FFFFh，才自复位到0，并继续计数直到周期寄存器的值；如果引脚 TDIRA/B 保持为低，且定时器的初始值大于周期寄存器的值时，那么计数器将减计数到周期寄存器的值后，再减计数直到0，当计数器的值为0后，重新装入周期寄存器的值，开始新的减计数。

周期、下溢和上溢中断标志位、中断和相关的事件都由各自的匹配产生，其产生方式与连续增计数模式下的一样。

从引脚 TDIRA/B 的变化到计数方向的变化之间的延时，是当前计数结束后的两个 CPU 时钟周期，即当前预定标的计数器周期结束之后的两个 CPU 时钟周期。

定时器在这种模式下的计数方向由 GPTCONA/B 寄存器中的方向指示位给出：1表示增计数；0表示减计数。无论从引脚 TCLKINA/B 输入的外部时钟还是内部CPU 时钟都可作为该模式下的定时器输入时钟。

通用定时器定向增/减计数模式如图9.5所示，图中预定标因子为1，TxPR=3。

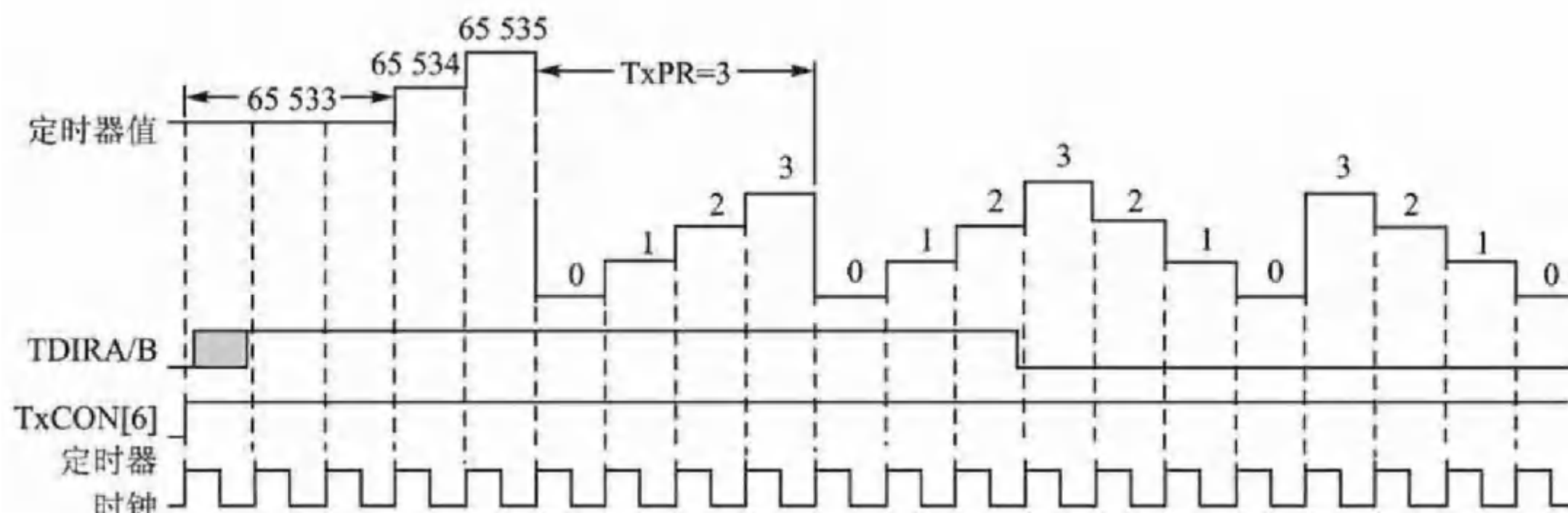


图9.5 通用定时器定向增/减计数模式

【例9.2】通用定时器1在定向增/减计数模式下的初始化示例。

```
LDP      #DP_EVA      ;指向7400h~7480h 单元
SPLK     #41H,GPTCONA ;TCOMPOE=1允许定时器比较输出
SPLK     #5H,T1PR      ;设置周期寄存器
SPLK     #3H,T1CMPR     ;设置比较寄存器
SPLK     #0H,T1CNT      ;设置计数寄存器
SPLK     #01F4EH,T1CON ;TMODE=11 定向增/减计数模式，TPS=111 预分频为128
                        ;TENABLE=1 定时器计数使能，TCLKS=00 内部时钟
                        ;TECMR=1 定时器1比较使能，SELT1PR=0
```


通用定时器的定向增/减计数模式，能够用于事件管理模块中的正交编码脉冲电路。在这种情况下，正交编码脉冲电路为定时器2和4提供计数时钟和方向。此工作方式也可用于运动控制、电机控制和电力电子设备中的外部事件定时。

4. 连续增/减计数模式

此计数模式与定向增/减计数模式基本相同，只是在连续增/减计数模式下，引脚 TDIRA/B 的状态不再影响计数的方向。定时器的计数方向仅在计数器的值达到周期寄存器的值时（或在FFFFh 时，定时器的初始值大于周期寄存器的值），才从增计数变为减计数。定时器的计数方向仅在计数器的值达到0时，才从减计数变为增计数。

除了第一个周期外，定时器周期是 $2 \times (TxPR)$ 个定标输入时钟周期。如果定时器的计数器初始值为0，那么第一个计数周期的时间就与其他周期一样。

通用定时器的计数器初始值可以是0000h~FFFFh 中的任何值。当计数器初始值不同时，其第一个周期的计数方向和周期也不同。当计数器初始值大于周期寄存器的值时，定时器将增计数到FFFFh，然后复位为0，开始正常的连续增/减计数，就好像其初始值为0一样。当计数器初始值与周期寄存器的值相同时，定时器将减计数至0，然后开始正常的连续增/减计数，就好像其初始值为0一样。当计数器初始值在0与周期寄存器的值之间时，定时器将增计数至周期寄存器的值并继续完成该周期，就好像计数器初始值与周期寄存器的值相同一样。

周期、下溢和上溢中断标志位、中断和相关的事件都由各自的匹配产生，其产生方式与连续增计数模式下的一样。

定时器在这种模式下的计数方向由GPTCONA/B 寄存器中的方向指示位给出：1表示增计数；0表示减计数。无论从引脚TCLKINA/B 输入的外部时钟还是内部CPU 时钟都可作为该模式下的定时器输入时钟。只是该模式下定时器忽略了引脚 TDIRA/B 的输入。

连续增/减计数模式如图9.6所示，这种计数模式尤其适用于产生对称的 PWM 波形。该波形广泛应用于运动控制、电机控制和电力电子设备中。

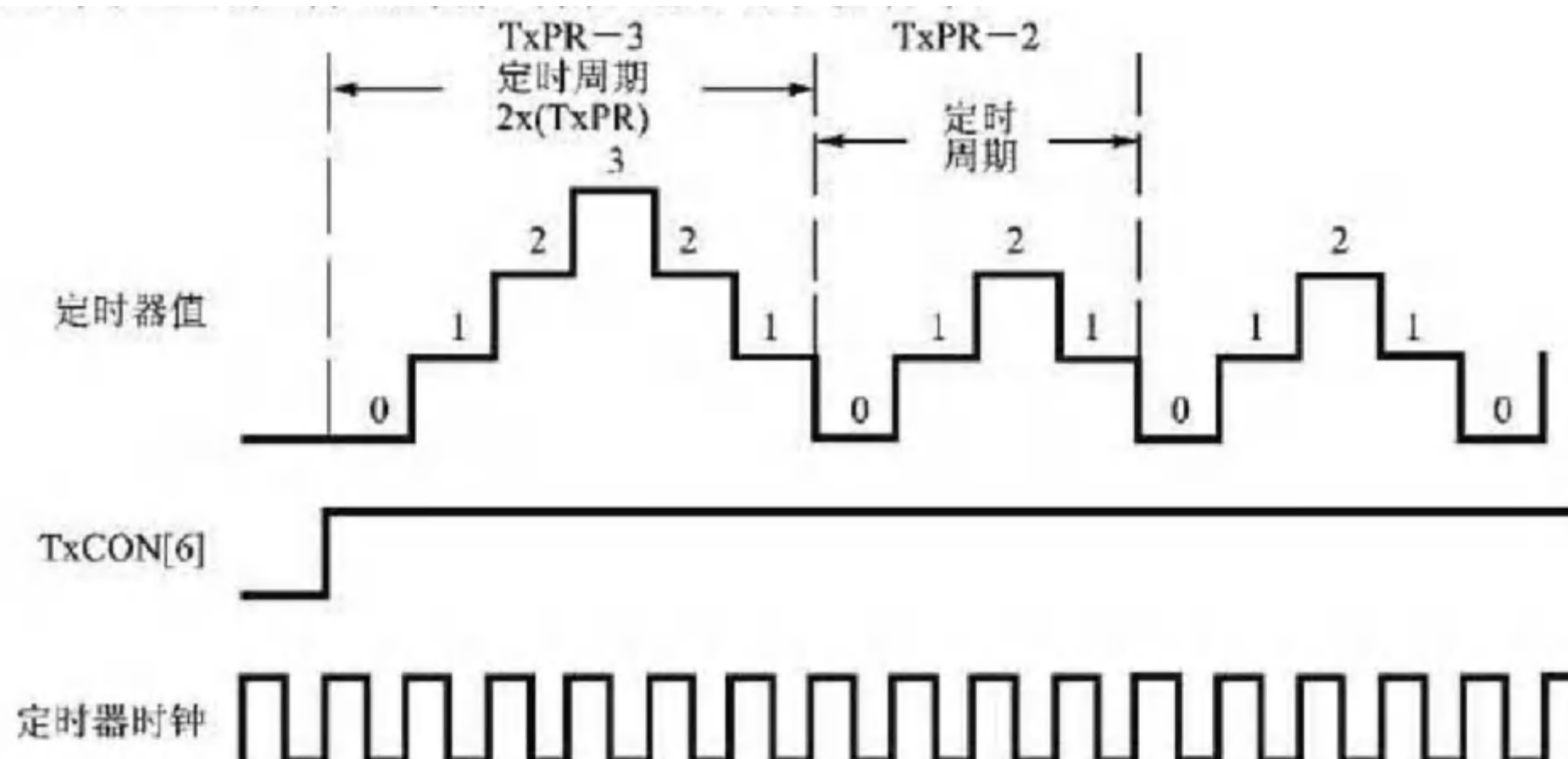


图9.6 通用定时器定向连续/减计数模式

【例9.3】 通用定时器1在连续增/减计数模式下的初始化例程。

```
LDP      #DP_EVA          ;指向7400h~7480h 单元
SPLK     #41H,GPTCONA     ;TCOMPOE=1 允许定时器比较输出
SPLK     #5H,T1PR         ;设置周期寄存器
SPLK     #3H,T1CMPR       ;设置比较寄存器
SPLK     #0H,T1CNT        ;设置计数寄存器
SPLK     #0F4EH,T1CON     ;TMODE=01连续增/减计数模式, TPS=111 预分频为128
                          ;TENABLE=1 定时器计数使能, TCLKS=00 内部时钟
                          ;TECMPR=1 定时器1比较使能, SELT1PR=0
```

9.2.4 通用定时器的比较操作

通用定时器有一个相关的比较寄存器 $TxCMPR$ 和一个 PWM 输出引脚 $TxPWM$ 。其值总是与相关的比较寄存器的值进行比较，当定时计数器的值与比较寄存器的值相等时，就产生比较匹配。可通过置 $TxCON$ 的1位为1来使能比较操作。产生比较匹配时，将发生下列事件：

- 定时器的比较中断寄存器标志位在匹配后的1个 CPU 时钟周期后被置位；
- 在匹配后的1个 CPU 时钟周期后，根据 $GPTCONA/B$ 寄存器相应位的配置情况，相关的 PWM 输出将发生跳变；
- 如果比较中断标志位已通过设置 $GPTCONA/B$ 寄存器中的相应位去启动 A/D 转换器，则当比较中断标志位被置位的同时，也将产生 A/D 转换的启动信号。

如果比较中断未被屏蔽，则将产生一个外设中断请求。

1. PWM 输出跳变

PWM 输出跳变由非对称和对称的波形发生器以及相关的输出逻辑来控制，并且取决于以下条件：

- $GPTCONA/B$ 寄存器中相应位的定义；
- 定时器所处的计数模式；
- 在连续增/减计数模式下的计数方向。

2. 非对称和对称波形发生器

非对称和对称波形发生器在通用定时器所处计数模式的基础上，产生一个非对称和对称的 PWM 波形输出。

3. 非对称波形发生

当通用定时器处于连续增计数模式时产生非对称波形，如图9.7所示。在此模式下，波形发生器的输出在以下情况的基础上有所变化：

- > 计数操作开始前为0；
- 保持不变直到比较匹配发生；
- > 比较匹配时，产生触发；
- 如果在下一周期的新比较值不是0，则发生周期匹配的周期结束后复位为0。

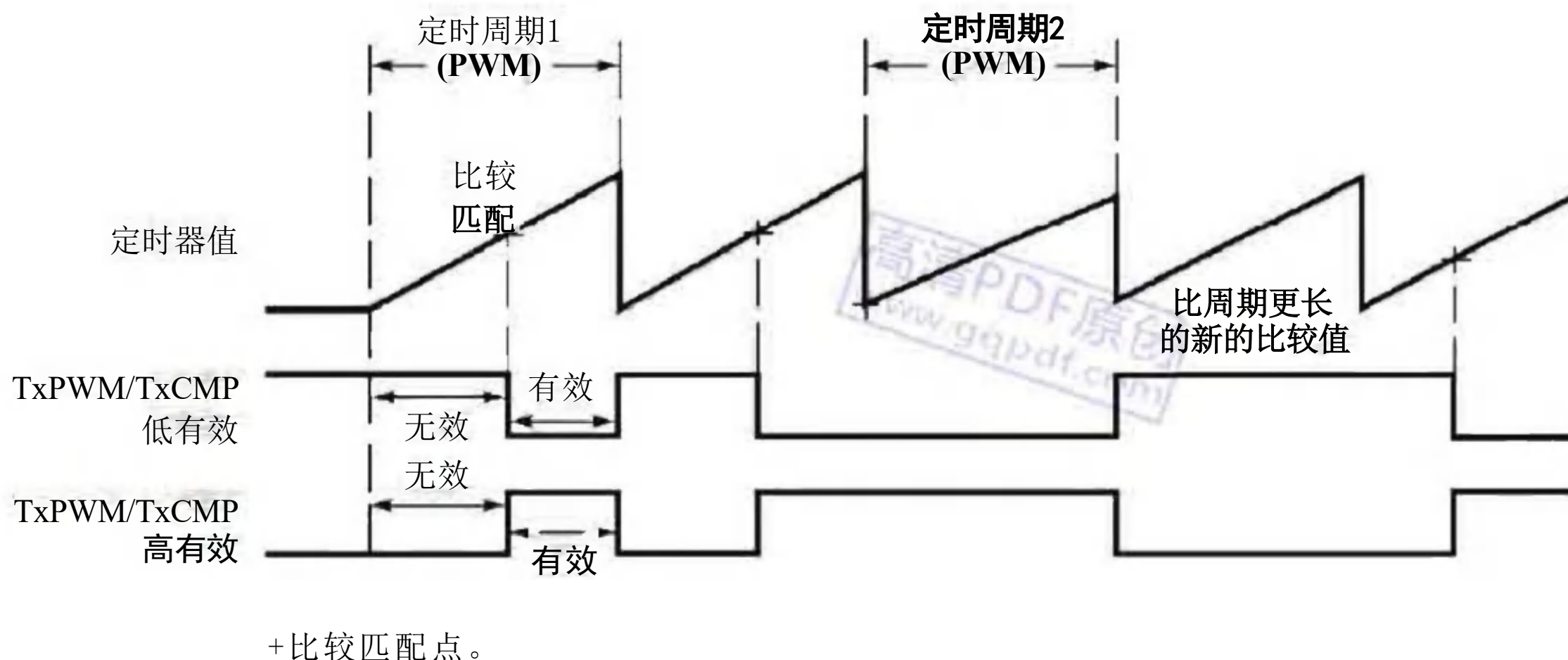


图9.7 连续增计数模式下的通用定时器比较输出

如果一个周期开始时的比较值为0，则在整个周期输出将是1；如果下一周期的新比较值也是0，则输出将不复位为0。这点非常重要，因为它允许产生占空比从0%~100%的无故障的 PWM 脉冲。如果比较值大于周期寄存器中的值，则整个周期输出为0；如果比较值等于周期寄存器的值，则输出为1，将保持一个定标后的时钟输入周期。

比较寄存器值的改变只影响PWM 脉冲的单边，这是非对称PWM 波形的一个特点。

4. 对称波形发生

当通用定时器处于连续增/减计数模式时产生对称波形，如图9.8所示。在这种模式下，波形发生器的输出状态由以下情况决定：

- > 计数操作开始前为0；
- > 保持不变直到第一次比较匹配；
- 第一次比较匹配时，产生触发；
- > 保持不变直到第二次比较匹配；
- 第二次比较匹配时，产生触发；
- > 保持不变直到周期结束；
- ▶ 如果没有第二次匹配并且下一周期的新比较值不为0，则在周期结束后复位为0。

如果比较值在周期开始时为0，则周期开始时将输出为1，并且保持1不变直到第二次比较匹配发生。如果比较值在周期的后半部分是0，则输出将保持为1直到周期结束。在这种

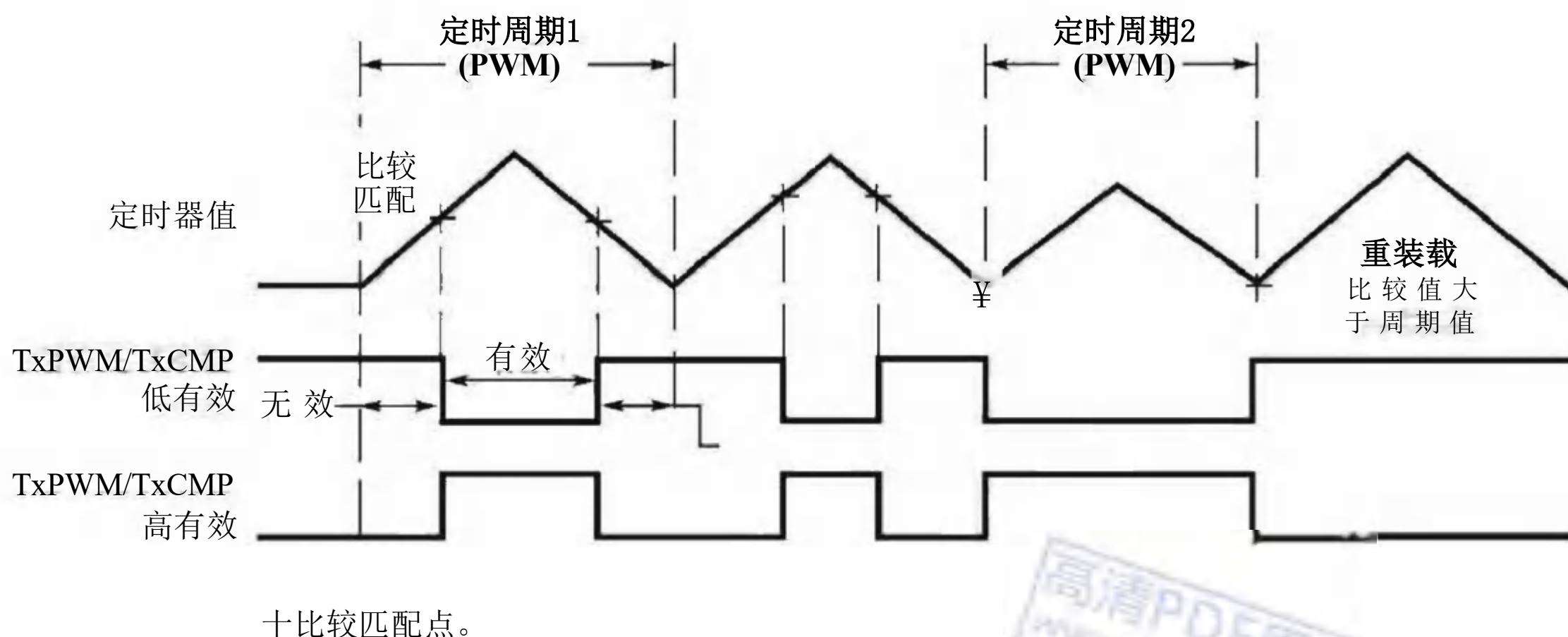


图 9.8 连续增/减计数模式下的通用定时器比较输出

情况下，如果新比较值仍为0，输出将不会复位为0。这会重复出现，以保证占空比为0%~100%的PWM 脉冲产生。如果前半周期中的比较值大于或等于周期寄存器的值，则第一次跳变将不会发生。不过，当后半周期发生比较匹配时，输出仍将跳变。这种输出错误的跳变经常是由应用程序计算不正确引起的，它将在周期结束时被纠正，因为除非下一周期的比较值为0，输出才被复位为0；否则输出将保持为1，这将把波形发生器的输出重新置为正确的状态。

注意：输出逻辑决定了所有输出引脚的有效状态。

5. 输出逻辑

输出逻辑可以调整波形发生器的输出，以生成不同类型功率设备所需的 PWM 波形。PWM 输出可以通过配置GPTCONA/B 寄存器中的相应位来规定高有效、低有效、强制高或强制低。

当 PWM 输出规定为高有效时，它的极性与相关非对称和对称波形发生器的输出极性相同。

当 PWM 输出规定为低有效时，它的极性与相关非对称和对称波形发生器的输出极性相反。

一旦 GPTCONA/B 寄存器中的相应位规定 PWM 输出为强制高/低后，PWM 输出就立即被置成1/0。

总的来说，在一个正常的计数模式下，如果比较已被使能，则通用定时器的PWM 输出发生变化如表9.13(连续增计数模式)和表9.14(连续增/减计数模式)所列。

置位有效是指：当高有效时，则输出高电平；当低有效时，则输出低电平。置位无效则与上述相反。

表9. 13 连续增计数模式下的通用定时器比较输出

在一个周期的时间	比较输出的状态
比较匹配之前	不变
比较匹配时	置位有效
周期匹配时	置位无效

表9. 14 连续增/减计数模式下的通用定时器比较输出

在一个周期的时间	比较输出的状态
第一次比较匹配之前	不变
第一次比较匹配时	置位有效
第二次比较匹配时	置位无效
第二次比较匹配之后	不变

基于定时器计数模式和输出逻辑的非对称和对称波形发生器，同样适用于比较单元。
出现下列任何一种情况时，所有的通用定时器 PWM 输出都置为高阻态：

- >软件将 GPTCONA/B[6] 置成0；
- PDPINx 引脚上的电平被拉低并且未被屏蔽；
- 任何一个复位事件发生；
- >软件将 TxCON[I] 置为0。

6. 有效/无效时间计算

对于连续增计数模式，比较寄存器中的值从计数周期开始到发生第一次比较匹配之间经过的时间，也就是无效相位的长度。这段时间等于定标的输入时钟周期乘以 TxCMPR 寄存器的值。因此，有效相位长度，即输出脉冲宽度等于(TxPR-TxCMPR+1) 个定标的输入时钟周期。

对于连续增/减计数模式，比较寄存器在减计数和增计数模式下可以有不同的值。对于连续增/减计数模式下的有效相位长度，即输出脉冲宽度是由(TxPR-TxCMPR+TxPR-TxCMPRm) 个定标输入时钟周期给定，这里的 TxCMPR 是增计数模式下的比较值，TxCMPR. 是减计数模式下的比较值。

当定时器处于连续增计数模式时，如果 TxCMPR 中的值为0, 则通用定时器的比较输出将在整个周期中有效。对于连续增/减计数模式，如果 TxCMPR 中的值为0, 那么比较输出在周期开始时有效；如果TxCMPR 山也为0, 那么输出将保持有效直到周期结束。

对于连续增计数模式，当TxCMPR 中的值大于TxPR 的值时，有效相位长度，即输出脉冲宽度为0。对于连续增/减计数模式，当TxCMPR 大于或等于TxPR 时，第一次跳变将不会发生；同样当TxCMPR. 大于或等于TxPR 时，第二次跳变也将不会发生；如果 TxCMPRp 和 TxCMPR. 都大于或等于 TxPR 时，通用定时器的比较输出在整个周期中都无效。

9.2.5 通用定时器的 PWM 输出

每个通用定时器都可以独立地用于提供一个PWM 输出通道，因此，通用定时器最多可提供 4 个 PWM 输出。

可选用连续增或连续增/减计数模式来产生 PWM 输出。选用连续增计数模式时可产生边沿触发或非对称 PWM 波形；选用连续增/减计数模式时可产生对称 PWM 波形。可按以下的方法来设置通用定时器的 PWM 输出操作：

- ＞ 根据所需的 PWM（载波）周期设置 TxPR；
- ＞ 设置 TxCON 寄存器以确定计数模式和时钟源并启动 PWM 输出操作；
- ＞ 将相应于 PWM 脉冲的在线计算宽度（占空比）的值加载到 TxCMPR 寄存器中。

当选用连续增计数模式来产生非对称的 PWM 波形时，通过将所需的 PWM 周期除以通用定时器输入时钟的周期，并减1以获得定时器的周期；当选用连续增/减计数模式来产生对称的 PWM 波形时，通过将所需的 PWM 周期除以两倍通用定时器输入时钟的周期，以获得定时器的周期。

当要产生的波形其占空比不同时，只需在线计算出相对于占空比的宽度的值，并加载到比较寄存器 TxCMPR 中即可。

9.2.6 通用定时器复位

当任何复位事件发生时，将产生以下情况：

- GPTCONA/B 寄存器中除计数方向指示位外的所有与通用定时器相关的位都被复位为0，因此所有通用定时器的操作都被禁止，计数方向指示位都置为1；
- ＞ 所有的定时器中断标志位都被复位为0；
- ＞ 所有的定时器中断屏蔽位都被复位为0，因此所有通用定时器的中断都被屏蔽；
- ＞ 所有通用定时器的比较输出都被置成高阻态。

9.2.7 通用定时器的中断实现

下面是用通用定时器1产生1 ms 的延时中断服务的汇编程序和 C 语言子程序。外部输入时钟为6 MHz，经 DSP 内部锁相环4倍频为24 MHz，通用定时器1预分频因子为64，所以计算出延时1 ms 周期寄存器的值为0177h。

1. 汇编程序设计

；INT2的中断向量为0004h,T1PINT的外设中断向量为0027h

.include "F2407REGS.H" ;引用头部文件

.def c_int0

；建立中断向量表

sect ".vectors" ;定义主向量段

RSVECT	B	c_int0	；PM 0	复位向量	1
INT1	B	PHANTOM	；PM 2	中断优先级1	4
INT2	B	GISR2	；PM 4	中断优先级2	5

```
INT3          B    PHANTOM          ;PM 6    中断优先级3          6
INT4          B    PHANTOM          ;PM 8    中断优先级4          7
INT5          B    PHANTOM          ;PM A    中断优先级5          8
INT6          B    PHANTOM          ;PM C    中断优先级6          9
RESERVED      B    PHANTOM          ;PM E    模拟量输入中断(保留) 10
SW_INT8       B    PHANTOM          ;PM 10   用户定义软件中断      —
...
SW_INT31      B    PHANTOM          ;PM 3E   用户定义软件中断
;中断子向量入口定义pvecs
.sect         ",pvecs"              ;定义子向量段
PVECTORS      B    PHANTOM          ;保留向量地址偏移量  0000h
              B    PHANTOM          ;保留向量地址偏移量  0001h
              :
              B    PHANTOM          ;保留向量地址偏移量  0026h
              B    TIPINT_ISR        ;保留向量地址偏移量  0027h  T1PINT中断
              B    PHANTOM          ;保留向量地址偏移量  0028h
              :
              B    PHANTOM          ;保留向量地址偏移量  0041h

:主程序

.text
c_int0
    SETC      INTM
    CLRC      SXM
    CLRC      OVM
    CLRC      CNF          ;BO 区被配置为数据空间
    LDP       #OE0H        ;指向7000h~7080h 区
    SPLK      #81FEH,SCSR1  ;时钟 4 倍频, CLKIN=6  MHz,CLKOUT=24  MHz
    SPLK      #OEBH,WDCR    ;禁止 WDT
    LDP       #0
    SPLK      #02H,IMR      ;使能中断优先级 INT2
    SPLK      #OFFFh,IFR    ;清中断标志
    LDP       #DP_EVA       ;指向7400h~7480h 区
    SPLK      #80H,EVAIMRA  ;使能TIPINT中断
    SPLK      #OFFFh,EVAIFRA ;清 EVA中断标志
    SPLK      #0.GPTCONA
    SPLK      #177H,T1PR    ;使定时器每1ms 产生一次中断
    SPLK      #0.T1CNT
    SPLK      #0164CH.T1CON ;TMODE=10 连续增计数模式, TPS=110 预分频为64
                                ;TENABLE=1 定时器计数使能, TCLKS=00 内部时钟
```



```

;TECMPR=1 定时器1比较使能, SELT1PR=0
;开总中断
CLRC      INTM
;中断程序
GISR2:
    保护现场
    LDP      #0EOH
    LACC     PIVR,1      ;读取外设中断向量寄存器(PIVR),并左移一位
    ADD      #PVECTORS   ;加上外设中断入口地址
    BACC     ;跳到相应的中断服务子程序
T1PINT_ISR:
    通用定时器1中断入口
    LDP      #DP_EVA
    SPLK     #0,T1CNT
GISR2_RET:
    中断返回
    恢复现场
    CLRC     INTM      ;开总中断, 因为一进中断就自动关闭总中断
    RET
;假中断程序
PHANTOM
    KICK_DOG      ;复位看门狗
    RET
    END

```

从以上程序可以看出，当定时器1的T1CNT 计数到0177h 即 1 ms 时，就向CPU 申请中断，如期间没有其他的中断产生或申请，则 CPU 接受中断申请。先在主向量表查到优先级INT2 的中断向量入口GISR2，在 GISR2 中读取外设中断向量寄存器 PIVR 中的中断地址偏移量为0027h，再加上子向量表起始地址，通过查中断子向量段表，跳入通用定时器1中断入口 T1PINT_ISR。这样就实现了一个中断服务子程序。

在此说明假中断 PHANTOM 的作用。假中断向量是保持中断系统完整性的一个特性。当一个中断已被响应，但却无外设将中断向量地址偏移量装入外设中断向量寄存器 PIVR 时，假向量0000h 被装入 PIVR。这种缺省保证了系统可按照可控的方式进行处理。

2.C 语言程序设计

关于程序的几点说明：

(1) 程序中所有与 DSP 硬件有关的寄存器都用指针方式寻址。

(2) 因为该程序利用定时器1的周期中断实现所要求的功能，而定时器1的周期中断是中断优先级2(INT2) 的中断源之一，故在文件 vectors.asm 中 INT2 的中断向量直接跳向定

时器1的周期中断服务程序。

(3) 因为中断优先级2(INT2) 的中断源不只一个(除了定时器1的周期中断外还有比较中断等), 所以为了防正程序执行时, 由于干扰进入中断服务程序引起中断的错误执行, 在定时器1的周期中断服务程序的入口处, 根据定时器1的周期中断标志 TIPINT FLAG(寄存器 EVAIFRA 的第8位)判断, 是否真正发生了定时器1的周期中断。若为错误中断, 则不再执行中断服务程序而直接返回。

(4) 其他优先级的中断虽然在程序中被禁止, 但也可能由于干扰而引起其执行, 故在程序中也都要对其进行处理。

以下是C 源程序代码:

/所需的复位和中断向量定义文件vectors.asm

```
.ref          nothing          ;直接返回的中断服务程序符号
.ref          c_int0           ;复位向量符号
.sect         ".vectors"

RSVECT       B      c_int0     ;PM O   复位向量          1
INT1         B      nothing    ;PM 2   中断优先级 1       4
TNT2         B      T1INT      ;PM 4   中断优先级2       5
INT3         B      nothing    ;PM 6   中断优先级3       6
INT4         B      nothing    ;PMB   中断优先级4       7
TNT5         B      nothing    ;PM A   中断优先级5       8
INT6         B      nothing    ;PM C   中断优先级6       9
```

//主程序

//该程序利用了定时器1的周期中断功能, 每1 ms 产生一次中断

```
#include      "register.h"
```

//屏蔽中断程序

```
void         inline      disable()
```

```
{
    asm(setc INTM);
```

X/开总中断程序

```
void inline enable()
```

```
{
    asm("clrc INTM");
```

//系统初始化子程序

```
initial()
```

```
asm("setc SXM");          //符号位扩展有效
```

```
asm("clre OVM");          //累加器中结果正常溢出
```

```

asm("clrc   CNF");
SCSR1=0x81FE;
WDCR=0x0E8;

IMR=0x0002;
IFR=0x0FFFF;
WSGR=0X00;

//BO 被配置为数据存储空间
//CLKIN=6      MHz,CLKOUT=4×CLKIN=24      MHz
//不使能看门狗, 因为 SCSR2 中 的WDOVERRIDE
//即 WD 保护位复位后的缺省值为1, 故可以用
//软件禁止看门狗
//允许 INT2 中 断
//清除全部中断标志, 写1清0
//禁止所有的等待状态

//定时器1的初始化子程序
int   timerlint()

    EVAIMRA=EVAIMRA|0X0080;
    EVAIFRA=EVAIFRA&0X0080;
    T1CON=0x160C;
    T1PER=0x0177;
    T1 CNT=0x00;

//允许定时器1的周期中断
//清除定时器1周期中断标志
//Timer1 为连续增计数模式, 预分频值为64, 使用内部时钟
//Timer1 的周期寄存器的值根据1 ms 延时和预分频值确定
//Timer1 的计数器清0

//定时器1周期中断服务程序
void  interrupt  T1INT()

    int   flag;
    flag=EVAIFRA&0X0080;
    if(flag!=0x0080)

        enable();
        return;

    }

//如果是定时器1的周期中断, 则执行下面的程序
T1CNT=0x00;
EVAIFRA=EVAIFRA&0X0080;
enable();
return;

//其他优先级的中断服务程序, 虽然程序中已经禁止,
//但可能由于干扰会引起他们的执行, 故该中断服务程序无额外操作, 直接返回
void  interrupt  nothing()

    enable();

```

//允许总中断, 因一进中断服务程序后, 总中断就自动关闭了


```

return;                                /中断返回

/主程序
main()
(
    disable();                          //初始化时禁止中断
    initial();                          //系统初始化
    timerlint();                        //定时器1的初始化
    enable();                           /允许总中断
    TICON=TICON 0X0040:                //启动Timer1
    while(1)

//等待中断发生

```

9.3 比较单元

事件管理器EVA 模块中有3个全比较单元(比较单元1、2和3), EVB 模块同样也有3个全比较单元(比较单元4、5和6)。每个比较单元都有两个相关的PWM 输出。比较单元的时基由通用定时器1(EVA 模块)和通用定时器3(EVB 模块)提供。

每个模块的比较单元包括:

- × 3个16位的比较寄存器(EVA 模块, 为 CMPR1、CMPR2 和 CMPR3;EVB 模块, 为 CMPR4、CMPR5 和 CMPR6), 它们各带一个可读/写的影子寄存器;
- × 一个可读/写的比较控制寄存器(EVA 模块为 COMCONA,EVB 模块为 COMCONB);
- × 一个16位比较方式控制寄存器(EVA 模块为 ACTRA,EVB 模块为 ACTRB), 它们各带一个可读/写的影子寄存器;
- × 6个比较PWM (三态)输出引脚(EVA 模块为 PWM_y,y=1,2,3,4,5,6;EVB 模块为 PWM_z,z=7,8,9,10,11,12);

控制和中断逻辑。

比较单元结构框图如图9.9所示。图中对于 EVA 模块x=1、2、3,y=1、3、5,z=1; 对于 EVB 模块x=4、5、6,y=7、9、11,z=3。

比较单元和相关的PWM 电路的时基由通用定时器1(EVA 模块)和通用定时器3(EVB 模块)提供。不管在哪一种计数模式下, 只要比较使能, 比较输出将会发生跳变。

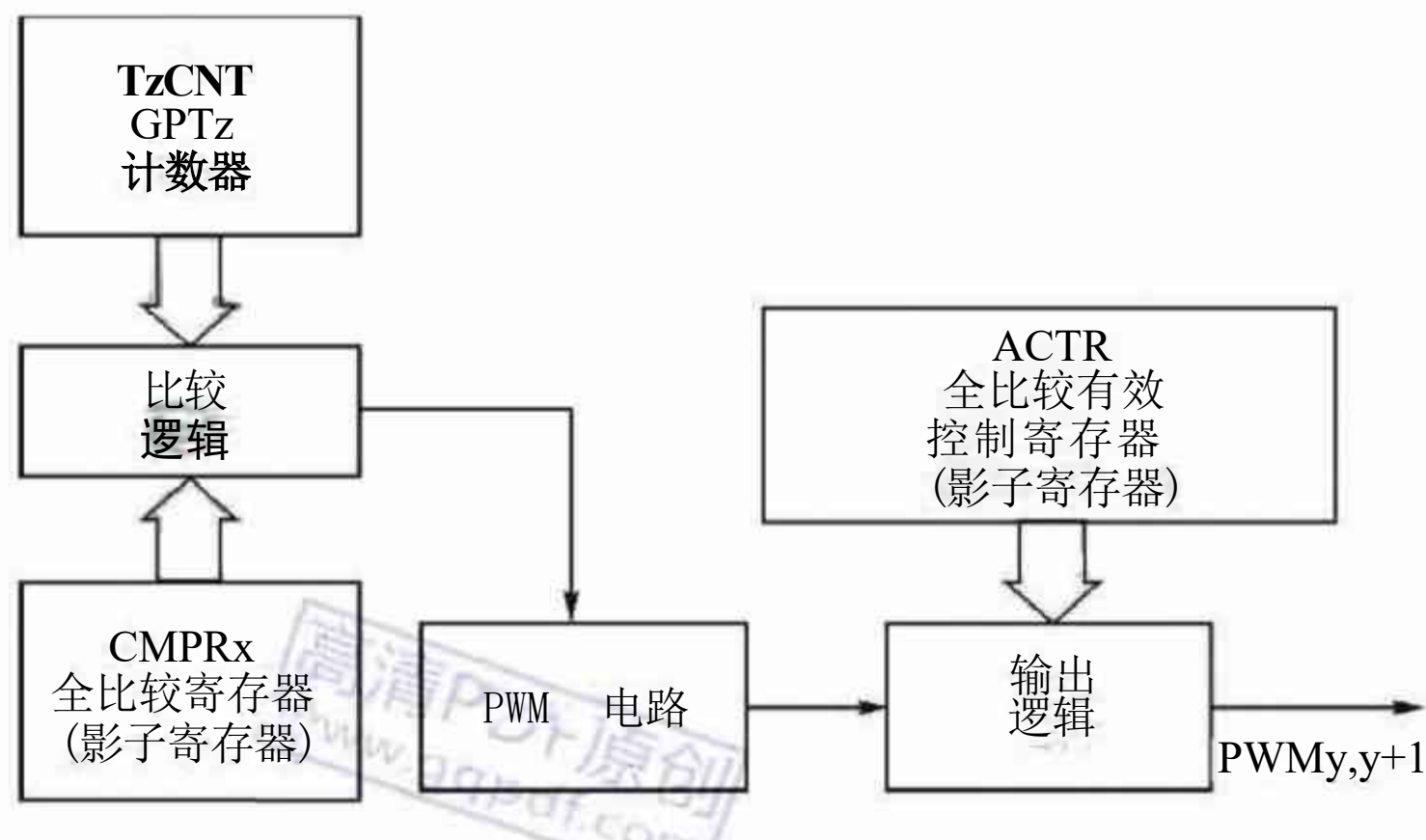


图 9.9 比较单元结构框图

1. 比较输入/输出

比较单元的输入包括：

＞控制寄存器的控制信号；

通用定时器1和3(T1CNT 和 T3CNT) 及其下溢和周期匹配信号；

复位信号。

比较寄存器的输出是一个比较匹配信号。如果比较操作使能，则该匹配信号将置中断标志位，并将使比较单元相关的两个输出引脚上发生跳变。

2. 比较操作模式

比较单元的比较操作模式由比较控制寄存器COMCONx 决定，这些位决定以下情况：

＞比较操作是否被使能；

＞ 比较输出是否被使能；

比较寄存器用其影子寄存器的值进行更新的条件；

空间矢量 PWM 输出模式是否被使能。

下面一段描述了EVA 模块比较单元的操作，对于EVB 模块只是把通用定时器1和 ACTRA 改为通用定时器3和 ACTRB。

通用定时器1的计数器不断与比较寄存器的值进行比较。当发生匹配时，比较单元的两个输出将根据方式控制寄存器 ACTRA 中的位进行跳变。ACTRA 寄存器中的位可以分别确定在比较匹配时每个输出为高有效触发或低有效触发(如果没有强制高或低)。当通用定时器1的计数器和比较单元的比较寄存器之间发生匹配且比较使能时，比较单元的比较中断寄存器将被置位。如果中断不屏蔽，则产生外设中断请求信号。输出跳变的时序、中断标志位的设置和中断请求的产生都与通用定时器的比较操作相同。输出逻辑、死区单元和空间矢量 PWM 单元可改变比较单元在比较模式下的输出。

3. 比较单元操作的寄存器配置

比较单元操作的寄存器配置步骤见表9. 15。

4. 比较单元的中断和复位

在 EVAIFRA 和 EVBIFRA 寄存器中，每个比较单元都有一个可屏蔽的中断标志使能位。如果比较操作被使能 • 比较匹配后的1个CPU 时钟周期后，比较单元的中断标志将置位。如果中断没有被屏蔽，则将产生一个外设中断请求。

当任何复位事件发生时，所有与比较单元相关的寄存器都复位为0, 且所有的比较输出引脚被置成高阻态。

5. 比较单元寄存器

比较单元的操作由比较控制寄存器 COMCONA 和 COMCONB 控制。如果 COMCONx.15 位使能了比较操作，则当比较事件发生时，比较方式控制寄存器 ACTRA 和 ACTRB 便控制 6 个比较输出引脚(ACTRA 为 PWMxx=1~6;ACTRB 为 x=7~12) 的输出方式。ACTRA 和 ACTRB 是双缓冲的，其重装条件由COMCONx 寄存器中相应的位来确定。它们也包含了空间矢量 PWM 操作所需的SVRDIR、D2、D1 和 D0 位。

(1)比较控制寄存器 A(COMCONA) —— 地址7411 h

15	14	13	12	11	10	9	8
CENABL. E	CLDI	CLDO	SVENABLE	ACTRLDI	ACTRL. D0	FCOMPOE	PDPINTA STATUS
RW_0	RW_0	RW_0	RW_0	RW_o	RW 0	RW_0	R_PDPINTA 引脚
7~0							
保留位							
R_0							

- 位15CENABLE。 比较使能位。
0 禁止比较操作，CMPRx 和 ACTRA 寄存器的影子寄存器为透明;
1 使能比较操作。
- 位14~13CLD1/CLDO。 比较寄存器CMPRx 重载条件。
00 当 TICNT=0 时(即下溢);
01 当 TICNT=0 或 TICNT=T1PR 时(即下溢或周期匹配);
10 立即;

表9. 15 比较单元操作的寄存器配置步骤

对于EVA	对于EVB
设置TIPR	设置T3PR
设置ACTRA	设置ACTRB
初始化CMPRx	初始化CMPRx
设置COM CON A	设置COM CON B
设置T1CON	设置T3CON

- 11 保留，结果不可预测。
- 位12 SVENABLE。空间矢量PWM 模式矢量位。
- 0 禁止空间矢量 PWM 模式；
- 1 使能空间矢量 PWM 模式。
- 位11~10 ACTRLD1/ACTRLD0。方式控制寄存器。
- 00 当 T1CNT=0 时(即下溢)；
- 01 当 T1CNT=0 或 T1CNT=T1PR 时(即下溢或周期匹配)；
- 10 立即；
- 11 保留。
- 位 9 FCOMPOE。比较输出使能位，有效的PDPINTA 将清此位为0。
- 0 PWM 输出引脚为高阻态，即禁止；
- 1 PWM 输出引脚未处于高阻态，即使能。
- 位 8 PDPINTA STATUS。此位反映了当前PDPINTA 引脚的状态。(此位只在 F240xA 系列中应用，在 F240x 系列中为保留位)
- 位7~0 保留位。

2) 比较方式控制寄存器 A(ACTRA)——地址7413 h

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

- 位 15 SVRDIR。空间矢量PWM 旋转方向位，仅用于空间矢量 PWM 输出的产生。
- 0 正向(CCW)； 1 反向(CW)。
- 位14~12 D2~D0。基本的空间矢量位，仅用于空间矢量PWM 输出的产生。
- 位11~10 CMP6ACT1/0。引脚 PWM6/IOPB3 上的比较输出方式选择位。
- 00 强制低； 01 低有效；
- 10 高有效； 11 强制高。
- 位9~8 CMP5ACT1/0。引脚 PWM5/IOPB2 上的比较输出方式选择位。
- 00 强制低； 01 低有效；
- 10 高有效； 11 强制高。
- 位7~6 CMP4ACT1/0。引脚PWM4/IOPB1 上的比较输出方式选择位。

	00	强制低；	01	低有效；
	10	高有效；	11	强制高。
位5~4	CMP3ACT1/0。 引 脚 PWM3/IOPB0 上的比较输出方式选择位。			
	00	强制低；	01	低有效；
	10	高有效；	11	强制高。
位 3 ~ 2	CMP2ACT1/0。 引 脚 PWM2/IOPA7 上的比较输出方式选择位。			
	00	强制低；	01	低有效；
	10	高有效；	11	强制高。
位 1 ~ 0	CMP1ACT1/0。 引 脚PWM1/IOPA6 上的比较输出方式选择位。			
	00	强制低；	01	低有效；
	10	高有效；	11	强制高。

由于EVB 模块中的比较控制寄存器COMCONB 和比较方式控制寄存器ACTRB 的 内 容 与 EVA 模 块 中 的 COMCONA 和 ACTRA 一 样 ， 只 是 把 定 时 器 1 改 为 定 时 器 3 ， 把 引 脚 PWM1~PWM6 改 为 PWM7~PWM12。 在此只给出了其寄存器单元地址。

3) 比较控制寄存器 **B(COMCONB)** —— 地 址**7511 h**

	14	13	12	11	10	9	8
	LDI	CLD0	SVENABLE	ACTRLDI	ACTRLD0	FCOMPOE	PDPINTB STATUS
RW_o	RW_o	RW_0	RW_o	RW_o	RW_o	RW_o	R_PDPINTB引脚
							7~0
保留位							
							R_0

(4) 比较方式控制寄存器 **B (ACTRB)** —— 地 址**7513 h**

15	14	13	12	11	10	9	8
SVRDIR	D2	DI	D0	CMP12ACT1	CMP12ACT0	CMP11ACT	CMP11ACT0
RW0	RW_0	RW_o	RW0	RW0	RW_0	RW_0	RW_0
7	6	5	4	3	2	1	0
CMP10ACT	CMP10ACT0	CMP9ACT1	CMP9ACT0	CMP8ACT1	CMP8ACT0	CMP7ACT1	CMP7ACT0
RW_0	RW_o	RW_o	RW_0	RW_0	RW_0	RW_0	RW_o

9.4 脉宽调制电路 PWM

事件管理模块可同时产生多达8路的PWM 波形输出。其由3个带可编程死区控制的比较单元产生独立的3对输出信号(即6个输出),以及由 GP 定时器比较产生的2个独立的 PWM 输出组成。

PWM 的特性如下:

- > 16 位 寄存器;
- > 有从0~16 μs 的可编程死区发生器控制的 PWM 输出;
- > 最小的死区宽度为1个CPU 时钟周期;
- > PWM 频率的变动可根据需要改变 PWM 的载波频率;
- > 在每个 PWM 周期内,可根据需要改变 PWM 脉冲的宽度;
- 可屏蔽的功率驱动保护中断;
- > 脉冲形式发生器电路,用于可编程的对称、非对称以及4个空间矢量 PWM 波形产生;
- > 自动重载的比较和周期寄存器使 CPU 的开销最小。

9.4.1 与比较单元相关的 PWM 电路

对于每个 EV 模块,与比较单元相关的 PWM 电路,使带有可编程死区和输出极性控制的 6 路 PWM 输出的产生成为可能。EVA 模块的 PWM 电路结构框图如图9.10所示。

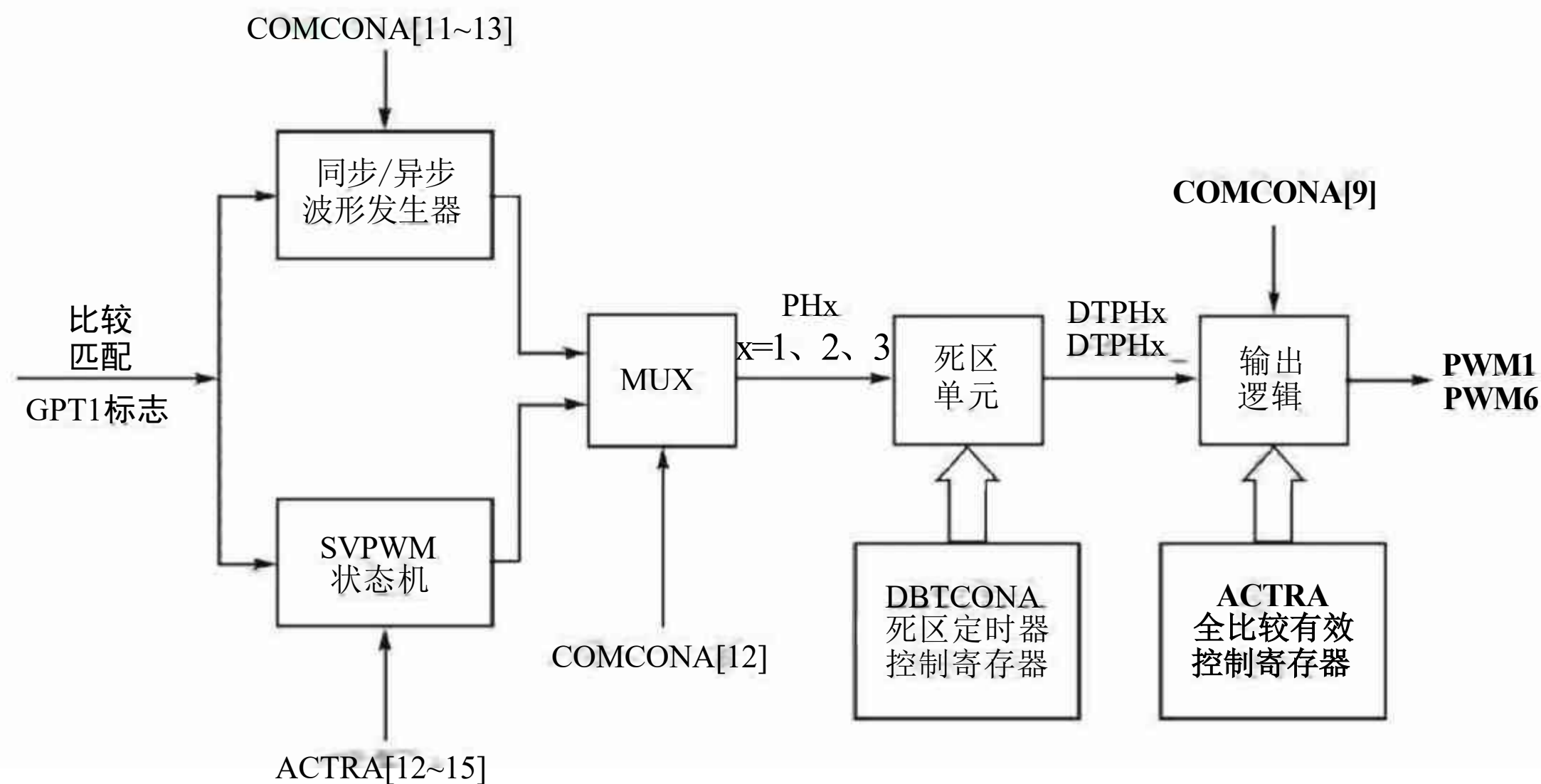


图9.10 PWM 电路结构框图

它包括以下的功能单元：

- >非对称/对称波形发生器；
- 可编程的死区单元(DBU)；
- >输出逻辑；
- > 空间矢量PWM 状态机。

对于EVB 模块的 PWM 电路结构框图与EVA 模块的一样，只是改变相关寄存器的配置，在此不再介绍。

在电机控制的应用中，PWM 电路极大地减少了产生PWM 波形的CPU 开销和用户的工作量。与比较单元相关的 PWM 电路，其波形的产生由以下寄存器控制：对于 EVA 模块为 T1CON、COMCONA、ACTRA 和 DBTCONA； 对 于 EVB 模 块 为 T3CON、COMCONB、AC-TRB 和 DBTCONB。

1. 可编程的死区单元

EVA 和 EVB 模块都有它们各自的可编程的死区单元DBTCONA 和 DBTCONB, 可 编 程 死区单元具有以下特点：

- 一个可读/写的16位死区控制寄存器 DBTCONx;
- 一个输入时钟预分频器：X/1、X/2、X/4、X/8、X/16 及 X/32;
- 内 部 CPU 时钟输入；
- >3个4位减计数定时器；
- 控制逻辑。

死区控制寄存器 A(DBTCONA)—— 地址7415 h

15~12				11	10	9	8
保留位				DBT3	DBT2	DBT1	DBT0
R_0				RW_o	RW_0	RW_0	RW_0
7	6	5	4	3	2	1~0	
EDBT3	EDBT2	EDBT1	DBTPS2	DBTPSI	DBTPS0	保留位	
RW 0	RW0	RW_0	RW0	RW0	RW0	R0	

- 位15~12 保留位。
- 位 1 1 ~ 8 DBT3~DBT0。死区定时器周期，这些位规定了3个4位死区定时器的周期值。
- 位 7 EDBT3。 死区定时器3使能位(对比较单元3的引脚PWM5 和 PWM6 来说)。
0 禁止； 1 使能。
- 位 6 EDBT2。 死区定时器2使能位(对比较单元2的引脚PWM3 和 PWM4 来说)。
0 禁止； 1 使能。

- 位 5

EDBT1。死区定时器1使能位(对比较单元1的引脚 PWM1 和 PWM2 来说)。
0 禁止; 1 使能。
- 位4~2

DBTPS2~DBTPS0。死区定时器的预分频器。
000X/1; 001X/2;

010X/4; 011X/8;
100X/16; 101X/32;
000X/32; 001X/32。
- 位1~0

保留位。

由于EVB 模块的死区控制寄存器 DBTCONB 的内容与DBTCONA 的一样,在此不再介绍,只给出它的单元地址。

2. 死区单元的输入和输出

死区控制寄存器 **DBTCONB**) ——地 址 **751**

15~12				11	10	9	8
保留位				DBT3	DBT2	DBT1	DBT0
R_0				RW_0	RW_0	RW_0	RW_0
7	6	5	4	3	2	1~0	
EDBT3	EDBT2	EDBT1	DBTPS2	DBTPS1	DBTPS0	保留位	
RW_0	RW_0	RW_0	RW_0	RW_0	RW_0	R_0	

死区单元的输入是PH1、PH2 和 PH3, 它们分别是由比较单元1、2和3的非对称/对称波形发生器产生的。

死区单元的输 出是DTPH1、DTPH1_DTPH2、DTPH2_ 和 DTPH3、DTPH3_。 它们 分别对应于 PH1、PH2 和 PH3。

3. 死区的产生

死区产生波形图如图9 . 11所示。对应于每个死区输入信号 PH_x, 产生两个输出信号 DTPH_x 和 DTPH_x。当比较单元和相关输出的死区未被使能时, 这两个输出信号完全相同; 当比较单元的死区单元使能时, 这两个信号的跳变沿被一段称做死区的时间间隔分开。该时间间隔由 DBTCON_x 寄存器中的相应位决定: 假设 DBTCON_x.11~8位中的值为 m, 并且 DBTCON_x.4~2 位中的值对应的预分频因子为 x/p, 则死区值为(p×m) 个 CPU 时钟周期。

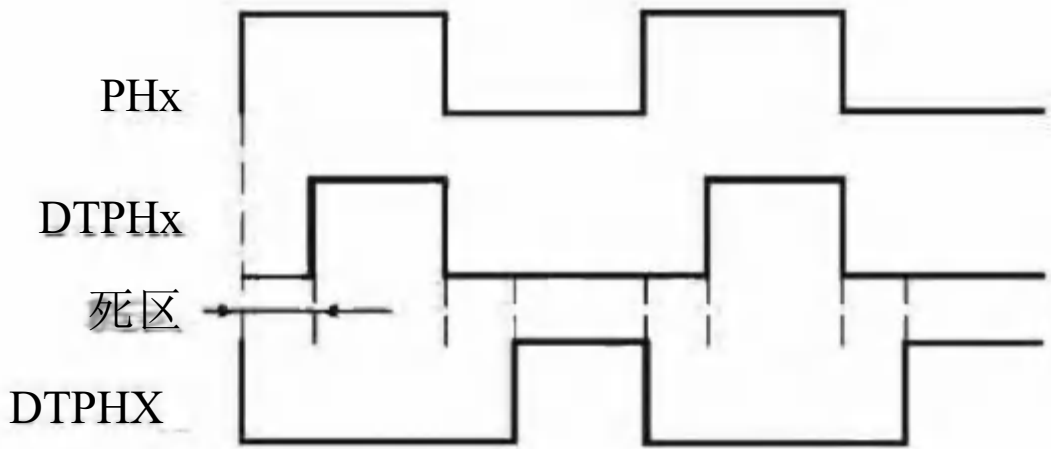


图 9 . 11 死区波形图

4. 死区单元的一些重要特征和作用

死区单元用于保证在任何情况下，每个比较单元相关的2路PWM 输出，控制一对正向导通和负向导通设备时没有重叠，即在一个器件在没有完全关断时，另一个器件不导通。极端的情况包括，用户装载了一个比占空周期更大的死区值或占空比为100%或0%。如果比较单元的死区被使能，则周期结束时与这个比较单元相关的PWM 输出不会复位到一个无效的状态。

在许多电机和电力电子应用中，常将两个功率器件(一个正向导通，另一个负向导通)串联到一个功率转换器的引脚上，并且两个器件一定不能同时导通，这是为了避免发生短路而击穿器件。因此，经常要用一对无重叠的PWM 输出，去正确地开启和关断这两个器件。死区时间经常被插入到一个器件的关断和另一个器件的开启之间。这种延时使得在一个功率器件在开启前，另一个功率器件已完全关断。所需的延时时间，由功率器件的开启和关断特性以及具体应用中的负载特性决定。

5. 输出逻辑

输出逻辑电路决定了比较发生匹配时，输出引脚 PWM_x(x=1~12) 上的输出极性和方式。每个比较单元相关的输出方式可分为低有效、高有效、强制低和强制高。比较单元的PWM 输出极性和方式通过 ACTR_x 寄存器中相应的位来配置。当发生以下任何一种情况时，所有的 PWM 输出引脚都置成高阻态。

软件清COMCON_x[9] 位；

＞ 当PDPINT_x 未被屏蔽时，由硬件将 PDPINT_x 引脚上的电平拉低；

发生任何复位事件；

＞ 来自死区单元的 DTPH₁、DTPH₁、DTPH₂、DTPH₂ 、DTPH₃和 DTPH₃ 及比较匹配信号；

＞ACTR_x 寄存器中的控制位。

比较单元的输出逻辑的输出如下：

PWM_x,x=1~6(EVA 模块)；

PWM_y,y=7~12(EVB 模块)。

PWM 输出逻辑结构框图如图9.12所示，图中x=1、2 或 3 ;EVA 模块y=1~6.EVB 模块y=7~12。

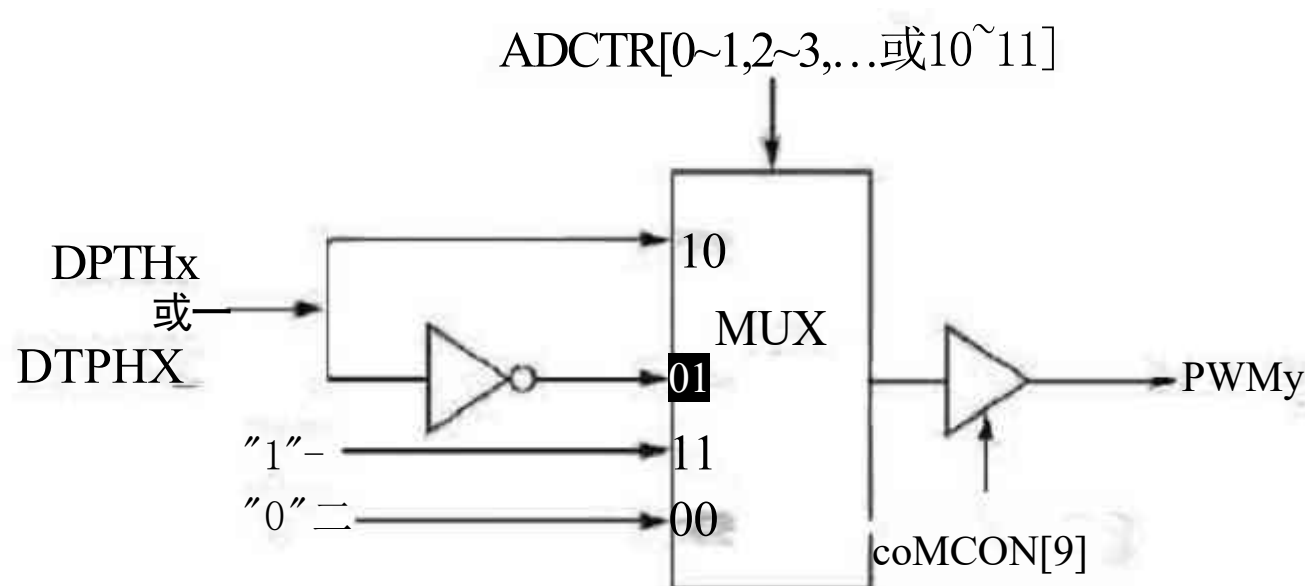


图9.12 PWM 输出逻辑结构框图

9.4.2 比较单元和PWM 电路中的PWM 波形产生

一个 PWM 信号是一串宽度变化的脉冲序列。这些脉冲平均分布在一段定长的周期中，

从而每个周期中有一个脉冲。该定长周期被称为 PWM (载波)周期, 它的倒数被称为 PWM (载波)频率。 PWM 脉冲的宽度由另一个具有所需值的调制信号决定或调制。

1. 事件管理器 PWM 输出的产生

在事件管理器模块中, 每个比较单元和通用定时器1(EVA 模块)或通用定时器3(EVB 模块), 死区单元及输出逻辑可在两个特定的器件引脚上产生一对具有可编程死区以及输出极性的 PWM 输出。在每个 EV 模块中有 6 个这种与比较单元相关的 PWM 输出引脚, 这 6 个特定的 PWM 输出引脚可用于控制三相交流感应电机和直流无刷电机。由比较方式控制寄存器(ACR_x) 所控制的多种输出方式, 能轻易地控制应用广泛的开关磁阻电机和同步磁阻电机。PWM 电路还可用于控制其他类型的电机, 如直流有刷电机和用于控制单轴和多轴的步进电机。如果需要, 则每个通用定时器的比较单元可基于它自身的定时器产生 1 路 PWM 输出。

2. 产生 PWM 的寄存器设置

比较单元和相关电路的所有 3 种 PWM 波形的产生, 需要对相同的事件管理寄存器进行配置, 产生 PWM 输出的设置步骤如下:

- (1) 设置和装载 ACR_x 寄存器;
- (2) 如果使能死区, 则设置和装载 DBTCO_x 寄存器;
- (3) 设置和装载 T1PR 或 T3PR 寄存器, 即规定 PWM 波形的周期;
- (4) 初始化 CMPR_x 寄存器;
- (5) 设置和装载 COMCO_x 寄存器;
- (6) 设置和装载 T1CON 或 T3CON 寄存器, 来启动比较操作;
- (7) 更新 CMPR_x 寄存器的值, 使输出的 PWM 波形的占空比发生变化。

3. 非对称 PWM 波形的产生

边沿触发或非对称 PWM 信号的特性由 PWM 周期中心非对称的调制脉冲规定, 如图 9.13 所示。每个脉冲的宽度只能从脉冲的一边开始变化。

图 9.13 给出了 EVA 模块下的非对称的 PWM 波形, 图中 x=1、3 和 5, 对于 EVB 模块也类似。为了产生非对称的 PWM 信号, 需将通用定时器 1 设置为连续增计数模式。通用定时器 1 的周期寄存器中装入了所需 PWM 载波周期的值。COMCO_A 寄存器中的相应位用来设置比较操作使能, 再将选中的输出引脚置成 PWM 输出并且使能这些输出。如果死区被使能, 则通过软件将所需的死区值写入到 DBTCO_A[11~8] 中, 并将它作为 4 位死区定时器的周期。一个死区值将用于所有的 PWM 输出通道。

用软件对 ACTRA 寄存器进行正确的配置后, 与比较单元相关的一个 PWM 输出引脚上将产生 1 路正常的 PWM 信号, 于此同时, 另一个输出引脚可在 PWM 周期的开始、中间和末尾处保持低电平(关闭)或高电平(开启)。这种可用软件灵活控制的 PWM 输出, 尤其适用于控制开关磁阻电机。

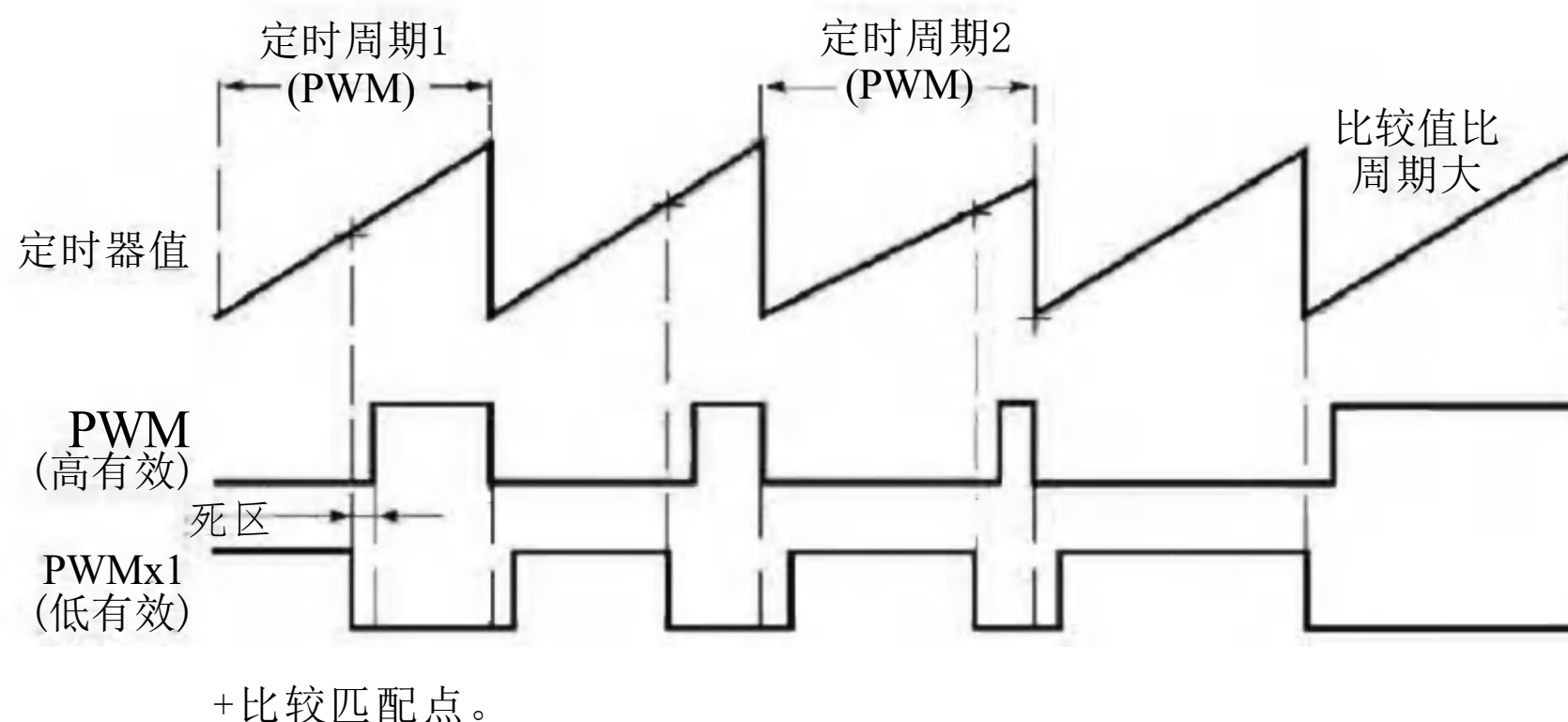


图9.13 比较单元和 PWM 电路产生非对称的 PWM 波形

通用定时器1启动后，比较寄存器在每个PWM 周期中可重新写入新的比较值，以调整用于控制功率器件的导通和关断时间的PWM 输出的宽度(即占空比发生变化)。因为比较寄存器是带影子寄存器的，所以在一个周期中的任何时候都可以将新值写入。同样，在周期的任何时候，可以将新值写入到周期寄存器 T1PR 和比较方式控制寄存器ACTRA 中，以改变 PWM 周期或强制改变PWM 的输出方式。

4. 对称PWM波形的产生

对称 PWM 信号的特性由 PWM 周期中心对称的调制脉冲规定。对称 PWM 信号相比非对称 PWM 信号，其优势在于它在一个周期内有两个无效的区段(每个PWM 周期的开始和结束处)。已经证明当使用正弦波调整时，在交流电机(如感应电机)和直流无刷电机的相电流中，对称的 PWM 信号比非对称的 PWM 信号引起的谐波失真更小。

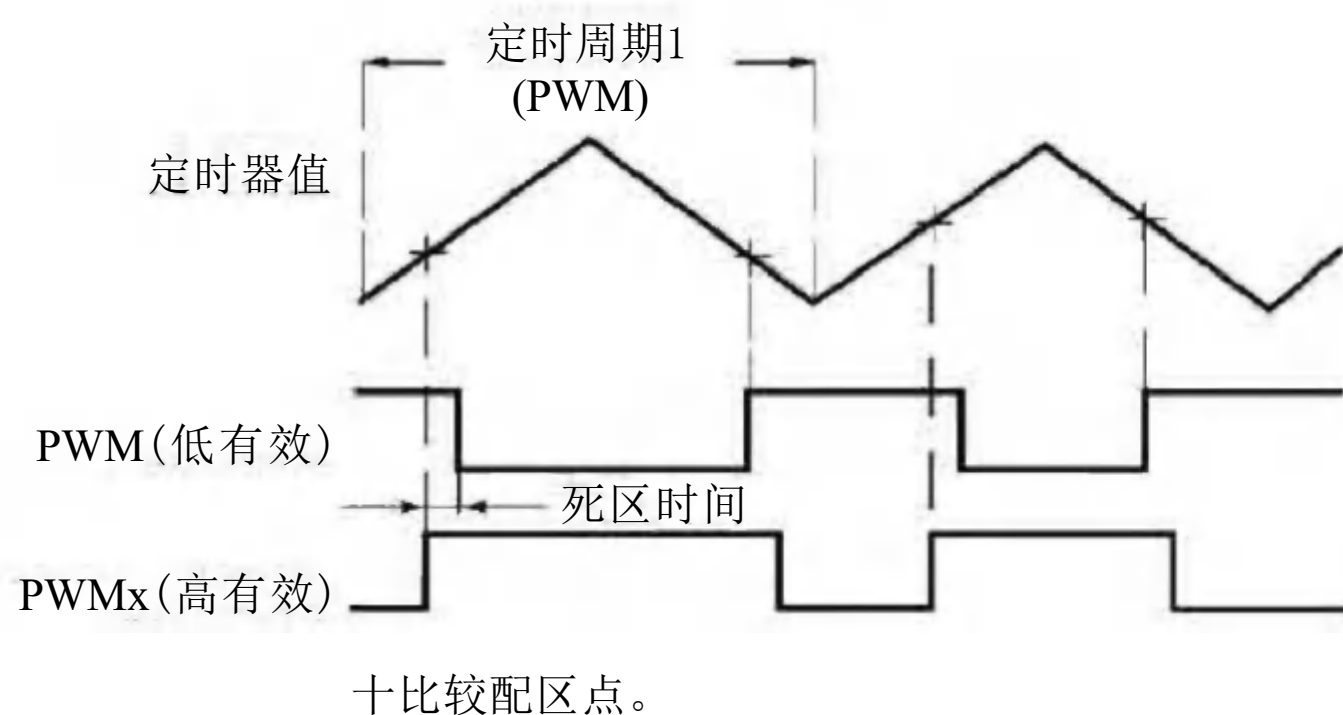


图9.14 比较单元和 PWM 电路产生对称的 PWM 波形

图9.14给出了对称 PWM 波形。

比较单元和PWM 电路的PWM 波形的产生与非对称的情况相似，唯一不同的是，通用定时器的计数模式应设置为连续增/减计数模式。

在对称 PWM 波形发生的一个周期内通常有两次比较匹配：一次在周期匹配前的增计数期间；另一次是在周期匹配后的减计数期间。新的比较值在匹配后更新了比较寄存器中的值，从而可以提前或推迟 PWM 脉冲的第二个边沿的到来。这种修改 PWM 波形的特性，可弥补由交流电机控制中的死区所导致的电流误差。

9.4.3 事件管理器的空间矢量 PWM 波形产生

空间矢量PWM 是指三相功率转换器的6个功率晶体管的一种特殊转换机制。它可以在三相交流电机的绕组中产生的电流谐波失真最小。它比用正弦波调制法，能更有效地利用电源电压。在此对空间矢量PWM 的原理不作介绍。

1. 产生空间矢量PWM 波形的寄存器设置

每个 EV 模块都具有极大简化对称空间矢量PWM 波形产生的内置硬件电路。为了输出空间矢量PWM 波形，用户需要设置以下寄存器：

- ＞ 设置 ACTRx 寄存器用来定义比较输出引脚的输出方式；
- ＞ 设置 COMCONx 寄存器来使能比较操作和空间矢量PWM 模式，并且把 CMPRx 的重装入条件设置为下溢；
- ＞ 将通用定时器1或3设置成连续增/减计数模式，并启动定时器。

然后，用户须确定在二维d-q 坐标系下输入到电机的电压 U_o ，并分解 U_o ，以确定每个PWM 周期的以下参数：

- ＞ 两个相邻向量， U 和 $U + 60^\circ$ 。

参数 T_1 、 T_2 和 T_a 。

- ＞ 将 相应于 U 的开启方式写入到 ACTRx.14~12， 并将 1 写入 ACTRx.15 中，或者将 $U+60^\circ$ 的开启方式写入 ACTRx.14~12， 并将0写入 ACTRx.15。
- ＞ 将 $T_1/2$ 的值写入到CMPR1 寄存器，将 $(T_1+T_2)/2$ 的值写入到 CMPR2 寄存器。

2. 空间矢量 PWM 的硬件

为完成一个空间矢量 PWM 周期，每个 EV 模块的空间矢量 PWM 硬件工作如下：

- ＞ 在每个周期的开始，将 PWM 输出置成由ACTRx.14~12 设置的新方式 U_y 。
- ＞ 在增计数期间，当CMPR1 和通用定时器1或3发生第一次匹配时，如果ACTRx.15 为 0，则PWM 将输出方式 U_y+60° ；如果 ACTRx.15 为1，则将 PWM 输出方式 U ，($U_o-60^\circ=U_3, U_3+60^\circ=U_6$)。
- ＞ 在增计数期间，当 CMPR2 和通用定时器1或3发生第一次匹配时，即计数器达到 $(T_1+T_2)/2$ 时，PWM 将输出开关方式000或111。它们与第二类输出方式之间只有1位的差别。
- ＞ 在减计数期间，当CMPR2 和通用定时器1或3发生第二次匹配时，将PWM 输出置回到第二类输出方式。
- ＞ 在减计数期间，当CMPR1 和通用定时器1或3发生第二次匹配时，将 PWM 输出置回到第一类输出方式。

3. 未用到的比较寄存器

在产生空间矢量PWM 输出中只用到了比较寄存器 CMPR1 和 CMPR2。



寄存器也一直在和通用定时器1进行比较。当发生一次比较匹配时，若相应的比较中断没有被屏蔽，则相应的比较标志位将置位并发出中断请求信号。因此，没有用于空间矢量 PWM 输出的CMPR3 寄存器仍可应用于其他定时事件的发生。同样地，由于状态机引入了附加延时，在空间矢量 PWM 模式中比较输出跳变被延时 1 个 CPU 时钟周期。

4. 空 间 矢 量 PWM 的边界条件

在空间矢量 PWM 模式中，当两个比较寄存器CMPR1 和 CMPR2 装入值都是0时，3个比较输出全都变成无效。因此，在使用空间矢量 PWM 时，应满足如下关系式： $CMPR1 \leq CMPR2 \leq T1PR$ ；否则将导致不可预测的状况。

5. 空间矢量 PWM 波形

生成的空间矢量 PWM 波形是关于每个PWM 周期中心对称的，因此被称为对称空间矢量 PWM 生成法，图9.15给出了空间矢量 PWM 波形的示例。

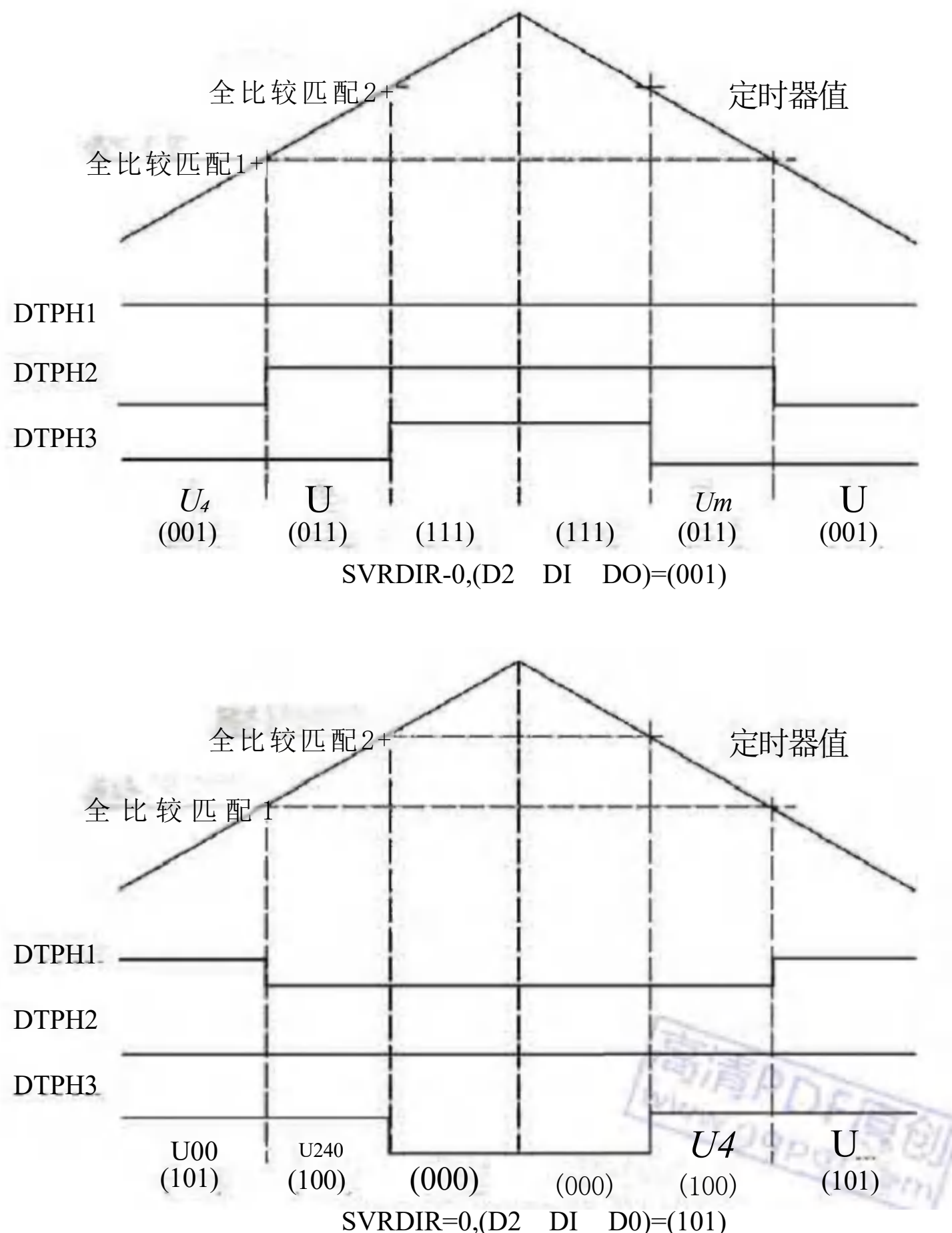


图9.15 对称空间矢量 PWM 波形


```

CALL    SYSINIT                ;系统初始化程序
CALL    PWM_INIT              ;EVB 模块PWM初始化程序

WAIT

NOP

B       WAIT

;系统初始化程序
SYSINIT:
    SETC    INTM
    CLRC    SXM
    CLRC    OVM
    CLRC    CNF                ;B0 区被配置为数据空间
    LDP     #OE0H              ;指向7000h~7080h 区
    SPLK    #81FEH,SCSR1      ;时钟 4 倍频, CLKIN=6 MHz,CLKOUT=24 MHz
    SPLK    #0E8H,WDCR        ;不使能WDT
    LDP     #0
    SPLK    #0002H,IMR        ;使能中断第二级 INT2
    SPLK    #0FFFFH,IFR      ;清全部中断标志
    RET

;EVB 模块的PWM 初始化程序
PWM_INIT:
    LDP     #DP  PF2          ;指向7080h~7100h 区
    LACL    MCRC
    OR      #007EH           ;IOPE[1~6] 被配置为基本功能方式: PWM[7~12]
    SACL    MCRC
    LDP     #DP_EVB          ;指向7500h~7580h 区
    SPLK    #0FFFFH,EVBIFRA  ;清 EVB全部中断标志
    SPLK    #0666H,ACTRB     ;PWM12,10、8 低有效, PWM11、9、7高有效
    SPLK    #00H,DBTCONB     ;不使能死区控制
    SPLK    #10H,CHPR4       ;设置比较初值 PWM7 高电平占50/60, 低电平占10/60
    SPLK    #20H,CHPR5       ;设置PWM9、10 的比较寄存器
    SPLK    #30H,CHPR6       ;设置PWM11、12的比较寄存器
    SPLK    #60H,T3PR        ;设置定时器3周期寄存器,
                                ;即 PWM 周期为 60 个 CPU 时钟周期
    SPLK    #0A600H.COMCONB  ;使能比较操作
    SPLK    #0.T3CNT
    SPLK    #41H,GPTCONB     ;TCOMPOE=1.T3PIN=01
    SPLK    #080H.EVBIMRA    ;通用定时器3使能
    SPLK    #0174EH,T3CON    ;TMODE =10连续增计数模式, TPS=111 预分频为128

```



```

                                ;TENABLE=1 定时器计数使能, TCLKS=00 内部时钟
                                ;TECMR=1定时器3比较使能, SELT3PR=0
                                ;开总中断

                                CLRC    INTM
                                RET

;定时器3中断程序
GISR2:                          ;优先级 INT2中断入口
                                保护现场

                                LDP      #OE0H
                                LACC     PIVR,1                      ;读取外设中断向量寄存器(PIVR), 并左移一位
                                ADD      #PVECTORS                    ;加上外设中断入口地址
                                BACC     ;跳到相应的中断服务子程序

T3GP_ISR:                       ;通用定时器3中断入口

                                LDP      #DPEVB
                                SPLK     #0,T3CNT

GISR2_RET:                      ;中断返回
                                恢复现场

                                CLRC     INTM                      ;开总中断, 因为一进中断, 就自动关闭总中断
                                RET

;假中断程序
PHANTOM

                                KICK_DOG    ;复位看门狗
                                RET

                                END

```

2.C 语言源程序

1) 所需的复位和中断向量定义文件 **vectors.asm**

//该文件利用汇编语言代码定义了复位和中断向量

```

.ref      nothing                ;直接返回的中断服务程序符号
.ref      c_int0                ;复位向量符号
.sect     ".vectors"

RSVECT    B      c_int0          //PM 0   复位向量                1
INT1      B      nothing         //PM 2   中断优先级1            4
INT2      B      nothing         //PM 4   中断优先级2            5
INT3      B      nothing         //PM 6   中断优先级3            6
INT4      B      nothing         //PM 8   中断优先级4            7
INT5      B      nothing         //PM A   中断优先级5            8
INT6      B      nothing         //PM C   中断优先级6            9

```

(2) 主程序

//该程序利用EVB模块的PWM7—PWM12 引脚产生不同占空比的方波

```
#include "register.h"
```

//屏蔽中断程序

```
void inline disable()
```

```
{
```

```
asm("setc INTM");
```

```
X
```

/系统初始化子程序

```
int initial()
```

```
asm("setc SXM");
```

```
asm("clrc OVM");
```

```
asm("clrc CNE");
```

```
*SCSR1=0x81FE;
```

```
WDCR=0x0E8;
```

```
*IMR=0x0000;
```

```
IFR=0x0FFFF;
```

```
WSGR=0x00;
```

```
}
```

Y/EVB 模块的PWM初始化程序

```
int pwminitial()
```

```
MCRC=*MCRC|0x007E;
```

```
ACTRB=0x0666;
```

```
*DBTCNB=0x00;
```

```
CMPR4=0x1000;
```

```
*CMPR5=0x3000;
```

```
*CMPR6=0x5000;
```

```
*T3PR=0x6000;
```

```
COMCONB=0x0200;
```

```
T3CON=0x1000;
```

//符号位扩展有效

//累加器中结果正常溢出

/B0 被配置为数据存储空间

//CLKIN=6 MHz,CLKOUT=4×CLKIN=24 MHz

//不使能看门狗, 因为 SCSR2 中的WDOVERRIDE

//即 WD 保护位复位后的缺省值为1. 故可以用

//软件禁止看门狗

//禁止所有中断

//清除全部中断标志, 写1清0

/禁止所有的等待状态

//IOPEI~6 被配置为基本功能方式, PWM7~12

//PWM12、10、8低有效, PWM11、9、7高有效

//不使能死区控制

设置定时器3的周期寄存器, 并设置 CMPR4~6, 以确定不

//同的输出占空比

//使能比较操作

/定时器3为连续增计数模式

//该中断服务程序主要是为了防止干扰. 不做任何其他操作

```
void interrupt nothing()
```

```
{
```

```
return;
```

//中断直接返回

```
}
```

//主程序

```

main()
{
    disable();           //总中断禁止
    initial();           //系统初始化
    pwminitialO);        //PWM 输出初始化
    *T3CON=*T3CON|0x0040; //启动定时器3
    while(1)
    {
        ;
    }
}

```

9.5 捕获单元

9.5.1 捕获单元概述

捕获单元在F240x 的捕获引脚上出现跳变时被触发，事件管理器总共有6个捕获单元。EVA 模块与它相关的捕获单元引脚有3个，它们分别为 CAP1、CAP2 和 CAP3，其可以选择通用定时器1或2作为它们的时基，然而CAP1 和 CAP2 一定要选择相同的定时器，作为它们的时基；EVB 模块与它相关的捕获单元引脚也有3个，它们分别为 CAP4、CAP5 和 CAP6，其可以选择通用定时器3或4作为它们的时基，然而 CAP4 和 CAP5 一定要选择相同的定时器，作为它们的时基。

当捕获输入引脚 CPAX（对 EVA,x=1、2、3；对 EVB,x=4、5、6）上检测到所选的跳变时，所选的GP 定时器的计数值被捕获并存入到一个2级深的FIFO 栈中。EVA 模块中的捕获单元结构框图如图9.16所示，对于 EVB 模块也类似，只是相应的通用定时器和寄存器的设置不同。

捕获单元包括下列特性：

- ＞1个16位的捕获控制寄存器 CPACONx（可读/写）。
- ＞1个16位的捕获FIFO 状态寄存器CAPFIFOx。
- ＞可选择通用定时器1/2(对 EVA) 或者3/4(对 EVB) 作为时基。
- ＞6个16位2级深的FIFO 栈(CAPxFIFO)，每个捕获单元一个。
- ＞3个施密特触发器输入引脚(对于 EVA,CAP1、2、3；对于 EVB,CAP4、5、6)，每个捕获单元一个输入引脚。(所有的输入和内部CPU 时钟同步，为了使跳变被捕获，输入必须在当前电平保持两个 CPU 时钟周期。输入引脚 CAP1、2 和 CAP4、5 也可用作正交编码脉冲电路的正交编码脉冲输入。)
- ＞用户可定义的跳变检测方式(上升沿、下降沿或者上升下降沿)。

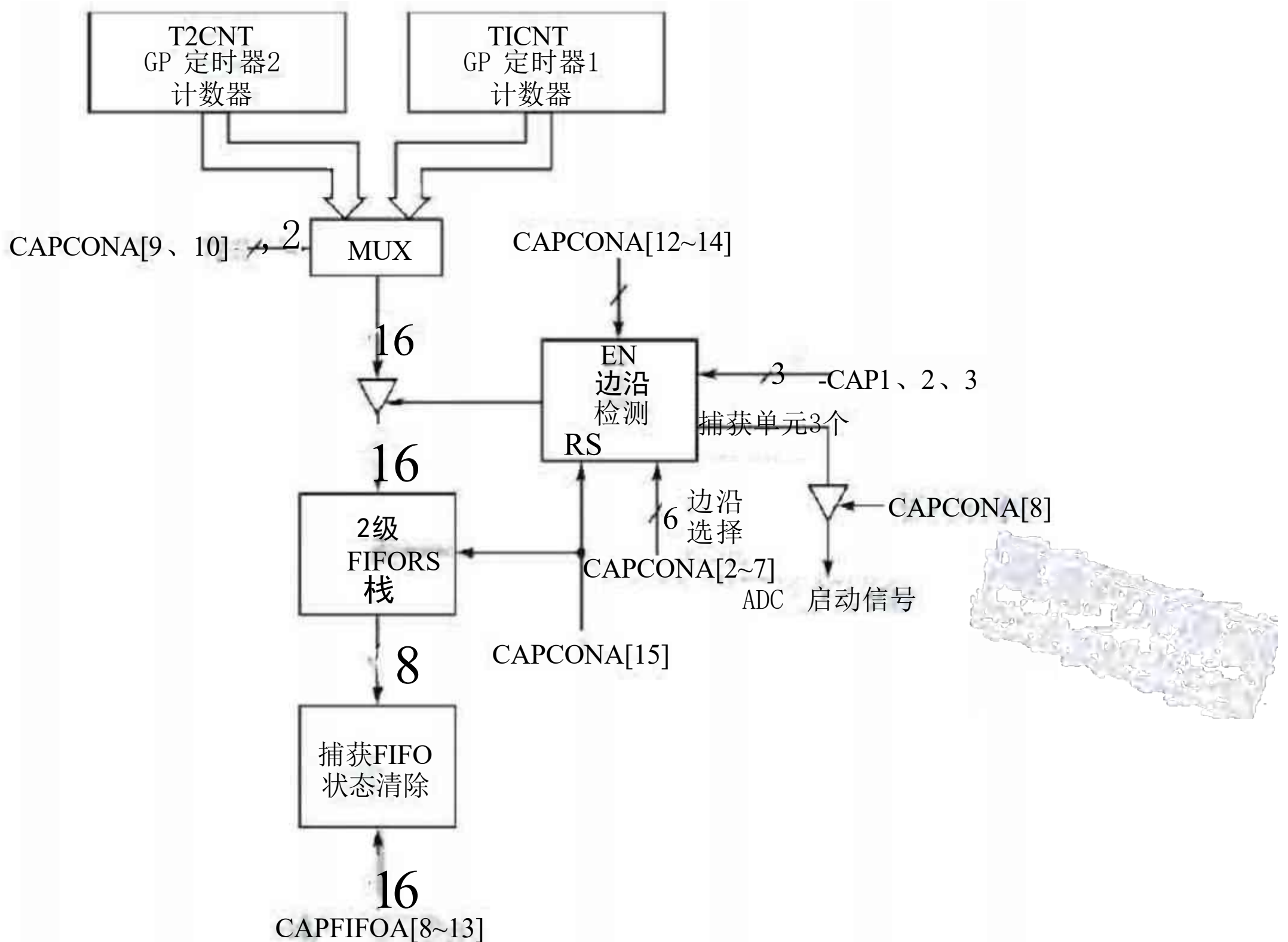


图9. 16 EVA 模块中的捕获单元结构框图

≥6个可屏蔽的中断标志位，每个捕获单元一个。

9.5.2 捕获单元操作

在捕获单元使能后，输入引脚上的指定跳变，将所选通用定时器的计数值装入到相应的 FIFO 栈中。与此同时，相应的中断标志位置位。如果该中断标志未被屏蔽，则外设中断将产生一个中断请求信号。每当将捕获到的新计数值存入到 FIFO 栈时，捕获 FIFO 状态寄存器 (CAPFIFOx) 的相应位就进行调整，以反映 FIFO 栈新的状态。从捕获单元输入引脚处发生跳变，到所选通用定时器的计数值被锁存之间的延时需要 2 个 CPU 时钟周期。

1. 捕获单元时基的选择

对于 EVA 模块，捕获单元CAP3 有自己独立的时基选择位，而CAP1 和 CAP2 没有。这就需要同时使用2个通用定时器： CAP1 和 CAP2 共用一个，而 CAP3 独立使用一个。在 EVB 模块，CAP6 有自己独立的时基选择位。

捕获单元的操作并不影响任何通用定时器或与通用定时器相关的比较/PWM 操作。

2. 捕获单元的设置

- 为使捕获单元能正常工作，需对以下寄存器进行设置：
- 初始化捕获FIFO 状态寄存器(CAPFIFOx)，并将相应的状态位清0；
 - 设置所选通用定时器的一种操作模式；
 - 如有必要，则应设置相关通用定时器的比较寄存器和周期寄存器；
 - 设置相应的捕获控制寄存器(CAPCONx)。

3. 捕获单元的寄存器

捕获单元的操作由4个16位的寄存器 CAPCONA/B 和 CAPFIFOA/B 控制。由于使用了通用定时器1~4为捕获单元提供时基，因此也用到了寄存器 TxCON。寄存器 CAPCONA/B 也可用于正交编码脉冲电路的检测。

(1)捕获控制寄存器 A(CAPCONA)——地址7420 h

15	14~13	12	11	10	9	8
CAPRES	CAPQEPN	CAP3EN	保留位	CAP3TSEL	CAP12TSEL	CAP3TOADC
RW_0	RW_0	RW_0	RW_0	RW_o	RW_0	RW_0

7~6	5~4	3~2	1~0
CAP1EDGE	CAP2EDGE	CAP3EDGE	保留位
RW_0	RW_0	RW_o	RW_0

- 位15 CAPRES。捕获复位，读该位总为0。
- 0 所有捕获单元和正交编码脉冲电路的寄存器清0；
 - 1 无操作。
- 位14~13 CAPQEPN。捕获单元1和2的控制位。
- 00 禁止捕获单元1和2, 其FIFO 栈保持原内容；
 - 01 使能捕获单元1和2；
 - 10 保留；
 - 11 保留。
- 位12 CAP3EN。捕获单元3控制位。
- 0 禁止捕获单元3, 其 FIFO 栈保持原内容；
 - 1 使能捕获单元3。
- 位11 保留位。
- 位10 CAP3TSEL。捕获单元3的通用定时器选择位。
- 0 选择通用定时器2；

- 1

选择通用定时器1。
- 位 9

CAP12TSEL。 捕获单元1和2的通用定时器选择位。
- 0

选择通用定时器2；
- 1

选择通用定时器1。
- 位 8

CAP3TOADC。 捕获单元3事件启动 A/D 转换位。
- 0

无操作；
- 1

当 CAP3INT 标志置位时，启动 A/D 转换。
- 位 7 ~ 6

CAP1EDGE。 捕获单元1的边沿检测控制位。
- 00

无检测；
- 01

检测上升沿；
- 10

检测下降沿；
- 11

检测两个边沿。
- 位 5 ~ 4

CAP2EDGE。 捕获单元2的边沿检测控制位。
- 01

无检测；
- 01

检测土升沿；
- 11

检测下降沿；
- 11

检测两个边沿。
- 位 3 ~ 2

CAP3EDGE。 捕获单元3的边沿检测控制位。
- 02

无检测；
- 01

检测上升沿；
- 12

检测下降沿；
- 11

检测两个边沿。
- 位 1 ~ 0

保留位 。

2) 捕获 **FIFO** 状态寄存器 **A(CAPFIFOA)** — 地 址**7422 h**

15~14	13~12	11~10	9~8
保留位	CAP3FIFO	CAP2FIFO)	CAPIFIFO
R_0	RW_0	RW_0	RW_0
7~0			
保留位			
R_0			

- 位 15 ~ 14

保留位。
- 位 13 ~ 12

CAP3FIFO。 捕获单元3的FIFO 栈状态位。
- 00

空 ；
- 0I

已有一个值压入栈；
- 10

已有两个值压入栈；
- 11

栈中已有两个值且又压入一个值，则最先压入的值出栈，即丢弃。
- 位 11 ~ 10

CAP2FIFO。 捕获单元2的FIFO 栈状态位。
- 00

空 ；



- 01 已有一个值压入栈；
10 已有两个值压入栈；
11 栈中已有两个值且又压入一个值，则最先压入的值出栈，即丢弃。
- 位9~8 CAP1FIFO。捕获单元1的FIFO栈状态位。
00 空；
01 已有一个值压入栈；
10 已有两个值压入栈；
11 栈中已有两个值且又压入一个值，则最先压入的值出栈，即丢弃。
- 位7~0 保留位。

由于EVB模块中的捕获单元寄存器CAPCONB和CAPFIFOB与EVA模块类似，在此只给出了其寄存器单元地址。

3) 捕获控制寄存器B(CAPCONB) ——地址7520 h

15	14~13	12	11	10	9	8
CAPRES	CAPQEPN	CAP6EN	保留位	CAP6TSEL	CAP5TSEL	CAP6TGADC
RW_0	RW_0	RW_0	RW_0	RW_0	RW_0	RW_0
7~6	5~4	3~2				
CAP4EDGE	CAP5EDGE	CAP6EDGE				保留位
RW_0	RW_0	RW_0				RW_0

(4) 捕获FIFO状态寄存器BCAPFIFOB)——地址7522 h

15~14	13~12	11~10	9~8
保留位	CAP6FIFO	CAP5FIFO	CAP4FIFO
R_0	RW_0	RW_0	RW_0
7~0			
保留位			
R_0			

4. 捕获单元FIFO栈

每个捕获单元都具有一个专用的2级FIFO栈，顶层栈包括CAP1FIFO、CAP2FIFO和CAP3FIFO（对于EVA模块），CAP4FIFO、CAP5FIFO和CAP6FIFO（对于EVB模块）；底层栈包括CAP1FBOT、CAP2FBOT和CAP3FBOT（对于EVA模块），CAP4FBOT、CAP5FBOT和CAP6FBOT（对于EVB模块）。顶层FIFO寄存器是只读寄存器，它总是存放相应捕获单元捕获到的旧计数值。因此对捕获单元FIFO栈的读访问总是读取栈中旧的计数值。当

FIFO 栈的顶层寄存器中的旧计数值被读取时，栈底层寄存器中的新计数值(如果有)将被压入顶层寄存器中。

如果读取了底层寄存器的值，那么捕获FIFO 状态寄存器的相应位将发生变化。如果读取前捕获FIFO 状态寄存器的相应位为10或11, 则读取后变成为01, 即栈中只有一个值；如果读取前捕获FIFO 状态寄存器的相应位为01, 则读取后变成为00, 即栈为空。

(1) 第1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所选通用定时器的计数值写入到空栈的顶层寄存器。同时，CAPFIFOx 寄存器中相应的 FIFO 状态位被置成01。如果在下一次捕获前对 FIFO 栈进行了读访问，则FIFO 状态位被复位为00。

(2) 第2次捕获。如果在前次捕获计数值被读取之前产生了另一次捕获，则新捕获到的计数值送至底层寄存器。同时，CAPFIFOx 寄存器中相应的FIFO 状态位被置成10。如果在下一次捕获前对 FIFO 栈进行了读访问，顶层寄存器中的旧计数值被读取，且底层寄存器中的新计数值被压入到顶层寄存器中，则FIFO 状态位被置成为01。

(3) 第3次捕获。如果在捕获发生时 FIFO 栈中已有两个捕获到的计数值，则位于栈顶层寄存器中最老的计数值将被弹出栈并丢弃，而栈底层寄存器中的计数值将压入到顶层寄存器中，新捕获到的计数值被压入到底层寄存器中，并且 FIFO 的状态位置成为11。

5. 捕获单元的中断

当进行捕获时捕获 FIFO 栈中至少有一个捕获到的计数值时，则相应的中断标志被置位。如果该中断没有被屏蔽，则会产生一个外设中断请求信号。如果使用了捕获中断，则可以从中断服务程序中读取捕获到的计数值；如果没有使用中断，则也可以通过查询中断标志位和 FIFO 栈的状态位来确定是否发生了捕获事件，若已发生捕获事件，则可从相应捕获单元的 FIFO 栈中读取捕获到的计数值。

9.5.3 捕获单元应用举例

下面的程序给出了用捕获单元4(CAP4) 对脉冲宽度进行捕获，用事件管理器 B(EVB) 的通用定时器3对脉冲进行计数。捕获单元可捕获上升沿、下降沿和上升下降沿，在此对脉冲的上升沿进行捕获，用中断的方式读取捕获值，使用捕获单元时要注意以下几点：

- 不知道捕获对象宽度时，应尽量使定时器定时最长，如不倍频或128预分频；
- > 如超过DSP 的最大捕获时间，则用定时器溢出的方法再加软件计数；
- F2407 为3.3 V 供电，因此捕获引脚输入电平应不超过3.3 V。

1. 汇编语言源程序代码

```
CAP4TEMP    .usect  ".data0".1           ;CAP4临时寄存器
             .include  "F2407REGS.H"      ;引用头部文件
             .def      c      int0
```

;建立中断向量表

```
                .sect".vectors"
RSVECT         B      c_int0
INT1           B      PHANTOM
INT2           B      PHANTOM
INT3           B      PHANTOM
INT4           B      GISR4
INT5           B      PHANTOM
INT6           B      PHANTOM
RESERVED       B      PHANTOM
SW_INT8        B      PHANTOM
```

:

```
SW_INT31      B      PHANTOM
```

;中断子向量入口定义 pvecs

```
                .sect ".pvecs"
                B      PHANTOM
                B      PHANTOM
                :
                B      PHANTOM
                B      CAP4_ISR
                B      PHANTOM
                :
                B      PHANTOM
```

;主程序

```
                .text
                c_int0
                LDP      #5
                SPLK      #00H,CAP4TEMP
                CALL      SYSINIT
                CALL      CAP_INIT
WAIT
                NOP
                B          LOOP
```

;系统初始化程序

SYSINIT:

;定义主向量段

```
;PM 0   复位向量           1
;PM2   中断优先级 1       4
;PM 4   中断优先级 2       5
;PM 6   中断优先级 3       6
;PM 8   中断优先级 4       7
;PMA   中断优先级 5       8
;PMC   中断优先级 6       9
;PM E   (保留位)          10
;PM 10  用户定义软件中断
;PM 3E  用户定义软件中断  —
```

;定义子向量段

```
;保留向量地址偏移量 - 00h
;保留向量地址偏移量 - 01h
;保留向量地址偏移量 - 35h
;保留向量地址偏移量 - 36h   CAP4 中 断
;保留向量地址偏移量 - 37h
;保留向量地址偏移量 - 41h
```

;调 CAP4 初始化程序


```

    SETC    INTM
    CLRC    SXM
    CLRC    OVM
    CLRC    CNF                ;BO 区被配置为数据空间
    LDP     #OE0H              ;指向7000h~7080h 区
    SPLK    #81FEH,SCSR1      ;时 钟 4 倍 频 ， CLKIN=6MHz,CLKOUT=24    MHz
    SPLK    #0E8H, WDCR       ;不使能 WDT
    LDP     #0
    SPLK    #0008H,IMR        ;使能中断第4级 INT4
    SPLK    #0FFFFH,IFR       ;清全部中断标志
    RET

;捕获单元初始化程序
CAP_INIT:
    LDP     #DP_PF2           ;指向7090h~7100h
    LACL    MCRC
    OR      #0380H            ;IOPE7、 IOPFO 和 IOPF1 被配置为特殊功
    SACL    MCRC              ;能方式： CAP[4~6]
    SETC    INTM
    LDP     #0,EVB
    SPLK    #040H, CAPCONB     ;TCOMPOE=1,T4PIN=10,T3PIN=01
    SPLK    #0,T3CNT
    SPLK    #01742H, T3CON     ;TMODE=10 连续向上计数模式， TPS=111 预分频为128
                                ;TENABLE=1定时器计数使能，  TCLKS=00内部时钟
                                ;TCLD=00,TECMPR=1 定时器3比较使能，  SELT3PR=0
    SPLK    #0FFFFH,T3PR      ;定时器周期比较寄存器设为最大
    SPLK    #00H,EVBIMRA
    SPLK    #00H,EVBIMRB
    SPLK    #22COH, CAPCONB    ;CAP4 捕获允许， 捕获上升沿
    SPLK    #01H,EVBIMRC      ;CAP4中断使能
    SPLK    #0FEFFh,EVBIFRA    ;清 EVB 全部中断标志
    SPLK    #0FFFFh,EVBIFRB
    SPLK    #0FFFFh,EVBIFRC
    CLRC    INTM              ;开总中断
    RET

;中断程序程序
```

```

GISR4:                                ;优先级 INT4 中断入口
    保护现场
    LDP    #0EOH
    LACC    PIVR,1                    ;读取外设中断向量寄存器(PIVR), 并左移一位
    ADD     #PVECTORS                ;加上外设中断入口地址
    BACC                                          ;跳到相应的中断服务子程序
CAP4_ISR:                             ;捕获单元4中断程序入口
    LDP     #DPEVB
    LACL    CAP4FIFO
    LDP     #5
    SACL    CAP4TEMP                ;读出捕获的值
    LDP     #DP_EVB
    SPLK    #0,T3CNT                ;清 T3 计数值, 使其重新计数
GISR2_RET:                            ;中断返回
    恢复现场
    CLRC    INTM                    ;开总中断, 因为一进中断就自动关闭总中断
    RET
;假中断程序
PHANTOM
    KICK_DOG                        ;复位看门狗
    RET
    END

```

2.C 语言源程序代码

(1) 所需的复位和中断向量定义文件 **vectors.asm**

// 该文件利用汇编语言代码定义了复位和中断向量

```

.title    "vectors.asm"
.ref      c__int0,nothing,capint
.sect     ".vectors"
reset:    b      c__int0
int1:     b      nothing
int2:     b      nothing
int3:     b      nothing
int4:     b      capint
int5:     b      nothing
int6:     b      nothing

```

(2) 主程序

//该程序用于测试DSP的CAPTURE模块，由TIMER1的比较模块输出一个PWM波形
1/此波形输入CAP4引脚，待CAP4捕捉该PWM的10次上升沿后，停止捕捉，并把10
1/次捕捉值存于数组result[10]中

```
#include "register.h"
```

```
//初始化子程序
```

```
int result[10]; //定义一个存储结果的数组
```

```
int k=0; //定义的中断次数值
```

```
//系统初始化子程序
```

```
int initial()
```

```
asm("setc INTM");
```

```
//禁止所有中断
```

```
asm("setc SXM");
```

```
//抑制符号位扩展
```

```
asm("clrc OVM");
```

```
//累加器中结果正常溢出
```

```
asm("clrc CNF");
```

```
//BO被配置为数据存储空间
```

```
SCSR1=0x81FE;
```

```
//CLKIN=6 MHz,CLKOUT=4×CLKIN=24 MHz
```

```
WDCR=0x0E8;
```

```
//不使能看门狗，因为SCSR2中的WDOVERRIDE
```

```
//即WD保护位复位后的缺省值为1，故可以用
```

```
//软件禁止看门狗
```

```
INT1=0x0000;
```

```
//禁止所有中断
```

```
IFR=0x0FFFF;
```

```
//清除全部中断标志，写1清0
```

K

//捕获单元4初始化子程序

```
int CAP4INT()
```

```
(
```

```
T3PER=0x0FFFF;
```

```
//通用定时器3的周期寄存器为0xFFFF
```

```
T3CON=0x1400;
```

```
//通用定时器3为连续增计数模式
```

```
T3CNT=0x00;
```

```
//计数器清0
```

```
WSGR=0x0000;
```

```
//禁止所有等待状态
```

```
CAPCONB=0x0A440;
```

```
//设置捕获单元4为检测上升沿，且选择TIMER3为时钟
```

```
asm("clrc INTM");
```

```
//开总中断
```

```
IMR=0X08;
```

```
//允许中断优先级4的中断
```

```
EVBMRC=EVBMRC|0x0001;
```

```
//允许CAPTURE4中断
```

```
EVBTFR=EVBTFR|0X0FFFF;
```

//定时器1初始化子程序，使其比较单元输出一个PWM波形

```
int timerlint()
```

```
MCRA=NCRA|0X1000;
```

```
//配置IOPB4端口为定时器1的比较输出
```

```
NCRC=NCRC|0x0080;
```

```
//配置IOPE7端口为捕捉功能
```

```
GPTCONA=GPTCONA|0X0042;
```



```

T1PR=0x1FE;
T1CON=0x1442;

T1CNT=0x00;
T1CMP=0x0FF;
}

//主程序
main()
{
    initial();                //系统初始化
    timerlint();              //定时器1初始化, 使其输出一个PWM 波形供捕捉
    CAP4INT();                //捕获单元4初始化
    T3CON=T3CON|0x0040;       //启动定时器3
    while(1)
    {
        if(k==10)            break;    //k 保存中断次数值, 是全局变量
    }
    asm("setc INTM");          //捕捉10次后, 禁止再中断
}

//若是由于干扰引起其他中断, 则执行此子程序
void interrupt nothing()

    asm("clrc INTM");          //返回前开中断
    return;
}

//捕捉中断服务程序
void interrupt capint()

    int flag;
    flag=EVBFRC&0x01;         //判断是否是CAP4 中断
    if(flag!=0x01)
    {
        asm("clrc INTM");      //返回前开中断
        return;                //如果不是 CAP4 中断, 则直接返回
    }
    load();                    //如果是CAP4中断, 则装载捕捉值
    EVBFRC=EVBFRC|0x01;        //写1清除 CAP4 中断标志
    asm("clrc INTM");          //返回前开中断
    return;                    //中断返回
}

//装载捕捉值子程序

```

```
int    load()
{
    result[k]=CAP4FIFO;           //读取捕捉值，存于相应的数组
    k++;
}
```

9.6 正交编码脉冲电路

9.6.1 正交编码脉冲电路概述

每个 EV 模块都有一个正交编码脉冲电路。该电路被使能后，可以在编码和计数引脚 CAP1/QEP1 和 CAP2/QEP2（对于 EVA 模块）或 CAP4/QEP3 和 CAP5/QEP4（对于 EVB 模块）上输入正交编码脉冲。正交编码脉冲电路可用于连接光电编码器以获得旋转机械的位置和速率等信息。如果使能了正交编码脉冲电路，则相应引脚上的捕获功能将被禁止。

1. 正交编码脉冲电路的引脚

两个QEP 输入引脚与捕获单元1和2(EVB 模块为3和4)共享，因此须正确设置 CAP-CONx 寄存器中相应的位，来使能正交编码脉冲电路，并禁止捕获功能，从而将相关的输入引脚分配给正交编码脉冲电路。

2. 正交编码脉冲电路的时基

正交编码脉冲电路的时基可由通用定时器2(EVB 模块为通用定时器4)提供。通用定时器必须设置成定向增/减计数模式，并以正交编码脉冲电路作为时钟源。图9.17给出了EVA 模块中正交编码脉冲电路的结构框图，EVB 模块也类似。

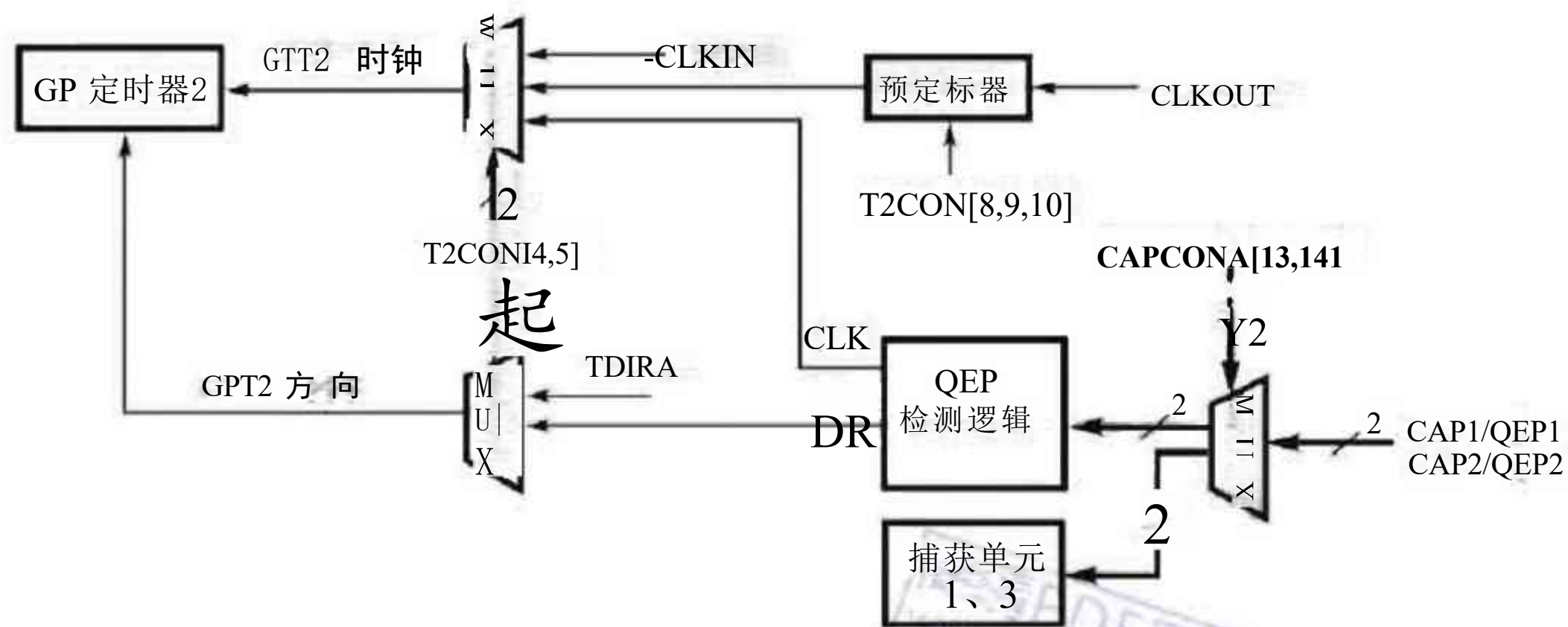


图9.17 EVA 模块中的正交编码脉冲电路结构框图

9.6.2 正交编码脉冲电路的编码操作

正交编码脉冲是两个频率变化且正交(即相位相差 90°)的脉冲。当它由电机轴上的光电编码器产生时,电机的旋转方向可通过检测两个脉冲序列中先到达的列来确定,角位置和转速可由脉冲数和脉冲频率(即齿脉冲和圈脉冲)来决定。

1. 正交编码脉冲电路

每个EV 模块中的正交编码脉冲电路的方向检测逻辑,决定两个序列中的哪一个是先导序列。接着它产生方向信号作为通用定时器2或4的计数方向输入。如果 CAP1/QEP1 (EVB 模块为CAP4/QEP3) 输入是先导序列,则通用定时器进行增计数;如果 CAP2/QEP2 (EVB 模块为 CAP5/QEP4) 输入是先导序列,则通用定时器进行减计数。

两列正交输入脉冲两个边沿都被正交编码脉冲电路计数,因此,产生的时钟频率是每个输入序列的4倍,并把这个时钟作为通用定时器2或4的输入时钟。

图9.18给出了正交编码脉冲、增/减计数方向以及其时钟的波形。

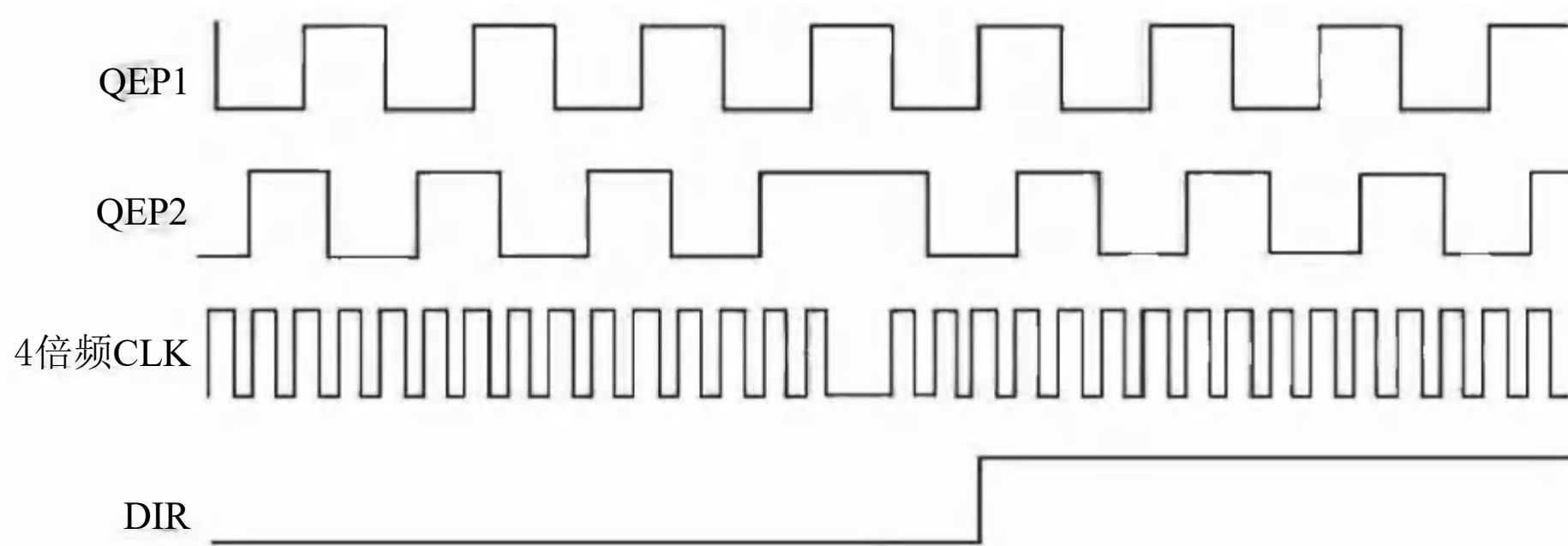


图 9.18 正交编码脉冲和编码定时器时钟及方向

2. 正交编码脉冲电路的计数

通用定时器2或4总是从计数器中的当前值开始计数,因此可以在使能正交编码脉冲电路前,将所需的值装载到所选通用定时器的计数器中。当正交编码脉冲电路的时钟作为通用定时器的时钟源时,选定的通用定时器将忽略输入引脚 TDIRA/B 和 TCLKINA/B (定时器方向和时钟)。

用正交编码脉冲电路作为时钟输入的通用定时器的周期、下溢、上溢和比较中断信号产生于各自的匹配发生时。如果中断没有被屏蔽,则将产生外设中断请求信号。

3. 正交编码脉冲电路寄存器的设置

正交编码脉冲电路启动设置如下:

(1) 对于 EVA 模块

>如果有必要,将所需的值装载到通用定时器2的计数器、周期和比较寄存器中;

- ＞ 设 置 T2CON 寄存器，将通用定时器2设置成定向增/减计数模式，以正交编码脉冲电路作为时钟源并使能通用定时器2；
- ＞ 设 置 CAPCONA 寄存器以使能正交编码脉冲电路。
- (2) 对于EVE 模拟
- ＞如果有必要，将所需的值装载到通用定时器4的计数器、周期和比较寄存器中；
- ＞ 设 置 T4CON 寄存器，将通用定时器4设置成定向增/减计数模式，以正交编码脉冲电路作为时钟源并使能通用定时器4；
- ＞ 设 置 CAPCONB 寄存器以使能正交编码脉冲电路。

9.6.3 正交编码脉冲电路的编程应用

下面的程序为正交编码电路的简单应用。为了调试方便，由 F240X 的 IOPE1 和 IOPE2 引脚模拟输出一个正交编码脉冲，模拟正交编码脉冲的软件流程如图9.19所示。

在图9.19中，每一处的软件延时都相等。不难看出，引脚 IOPE1 和 IOPE2 输出的波形与图9.18中 QEP1 和 QEP2 的波形相同。故可以作为 QEP 脉冲的模拟输入（即调试程序时，使 IOPE1 和 IOPE2 引脚分别与 CAP4/QEP3 和 CAP5/QEP4 相连）。改变软件延时的时间，就可以改变 QEP 脉冲的频率。

C 语言源程序代码如下：

1) 所需复位和中断向量定义文件 **vectors.asm**

// 该文件利用汇编语言代码定义了复位和中断向量

```

ititle          "vectors.asm"
.ref            _c_int0,nothing
.sect           ".vectors"

reset:          b      _c_int0
int1:           b      nothing
int2:           b      nothing
int3:           b      nothing
int4:           b      nothing
int5:           b      nothing
int6:           b      nothing
    
```

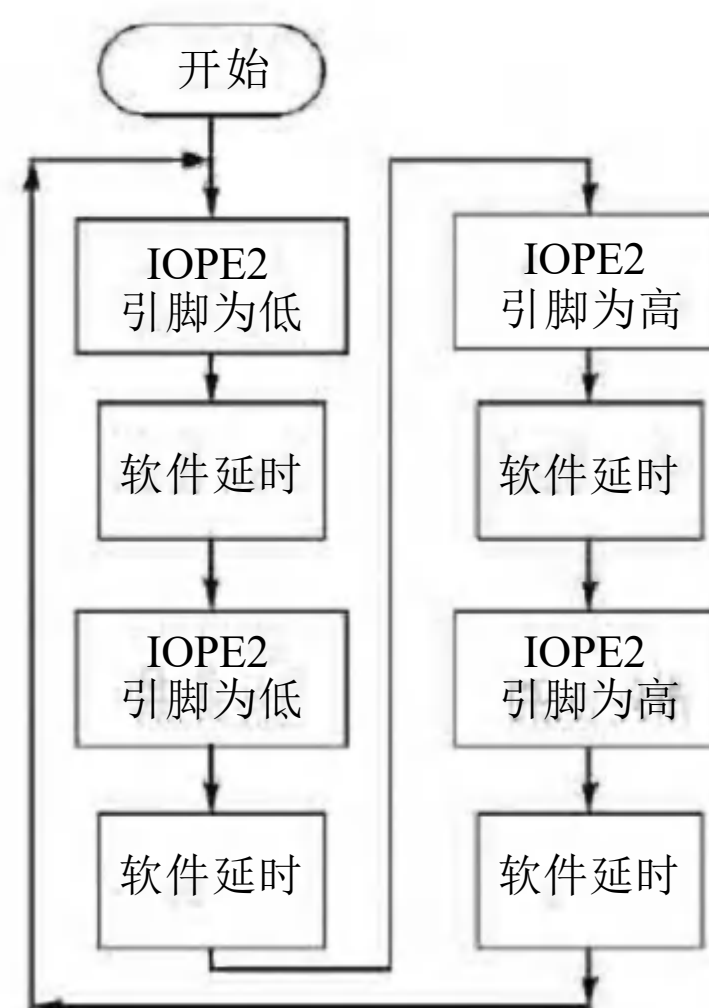


图9.19 模拟 QEP 脉冲软件流程图

(2) 主程序

//该程序用于测试 F240X的 EVB 模块的正交编码脉冲(QEP)电路

```
#include "register.h"
```

//初始化子程序

```
initial()
```

```
{
    asm("setc INTM"), //禁止所有中断
    asm("setc SXM"); //抑制符号位扩展
    asm("clrc OVM"); //累加器中结果正常溢出
    asm("clrc CNF"); //B0 被配置为数据存储空间
    SCSR1=0x81FE; //CLKIN=6 MHz,CLKOUT=4×CLKIN=24 MHz
    WDCR=0x0E8; //不使能看门狗, 因为 SCSR2 中的WDOVERRIDE
                //即WD 保护位复位后的缺省值为1, 故可以用
                //软件禁止看门狗
    IMR=0x0000; //禁止所有中断
    IFR=0x0FFFF; //清除全部中断标志, 写1清0
}
```

//QEP 电路初始化子程序

```
int QEPINT()
```

```
{
    T4PR=0x0FFFF; //通用定时器4的周期寄存器为0xFFFF
    T4CON=0x1870; //通用定时器4为定向增减计数模式, 并且以正交编码脉冲
                //电路作为时钟源
    T4CNT=0x00; //计数器清0
    WSGR=0x0000; //禁止所有的等待状态
    CAPCONB=0x0EO0; //使能EVB的 QEP 电路
}
```

//I/O 端口初始化子程序

```
int IOINT()
```

```
    MCRC=MCRC|0x0180; //设置相应的通用引脚的 I/O 功能
    PEDATDIR=PEDATDIR|0x0606; //设置IOPE1、IOPE2 为输出引脚且都为高电平
}
```

//延时子程序

```
int DELAY()
```

```
{
    int k;
    for(k=0;k<30000;k++)
```

```

//主程序
main()
(
    initial();           //系统初始化
    IOINT();             //IO口初始化
    QEPINT();            //QEP电机初始化
    while(1)
    {
        PEDATDIR=PEDATDIR&0xFFFB;    //使IOPE2 引脚变低
        DELAY();                     //延时子程序
        PEDATDIR=PEDATDIR&0xFFFD;    //使 IOPE1引脚变低
        DELAY();                     //延时子程序
        PEDATDIR=PEDATDIR|0x0004;    //使IOPE2 引脚变高
        DELAY();                     //延时子程序
        PEDATDIR=PEDATDIR|0X0002;    //使 IOPE1 引脚变高
        DELAY();                     //延时子程序
    }
}

```

//若由于干扰引起中断请求，则执行此中断服务程序，此中断程序相当于汇编中的假中断程序，即什么事情也不作，以保证中断系统的完整性. 防止程序跑飞

```
void interrupt nothing()
```

```
return;
```


第 10 章 A/D 转换模块

10.1 A/D 转换模块概述

F2407 的 A/D 转换模块(ADC) 具有以下特性:

- >带内置采样和保持(S/H) 的 10 位 ADC。
- >多达16个的模拟输入通道(ADCIN0~ADCIN15)。
- >自动采样能力。一次可执行最多16个通道的“自动转换”,而每次要转换的通道都可通过编程来选择。
- >两个独立的最多可选择8个模拟转换通道的排序器(SEQ1 和 SEQ2) 可以独立工作在双排序器模式,或者级连之后工作在一个最多可选择16个模拟转换通道的排序器模式。
- >在给定的排序方式下, 4个排序控制器(CHSELSEQn) 决定了模拟通道转换的顺序。
- >可单独访问的16个结果寄存器(RESULT0~RESULT15) 用来存储转换结果。
- > 可有多多个触发源启动 A/D 转换:
 - 软件: 软件立即启动(用SOC SEQn 位);
 - EVA: 事件管理器 A(在 EVA 中有多个事件源可以启动A/D 转换);
 - EVB: 事件管理器B(在 EVB 中有多个事件源可以启动A/D 转换);
 - 外部: ADC SOC引脚。
- >灵活的中断控制, 允许在每一个或每隔一个序列的结束时产生中断请求。
- >排序器可工作在启动/停止模式, 允许多个按时间排序的触发源同步转换。
- >EVA 和 EVB 可各自独立地触发 SEQ1 和 SEQ2(仅用于双排序器模式)。
- >采样和保持获取时间窗口有单独的预定标控制。
- > 内置校验模式。
- 内置自测试模式。

由于F240xA 器件的 ADC 模块与F24x 器件的ADC 模块不兼容, 因此F24x 器件的ADC 程序代码不能被移植到F240xA 器件。

10.2 自动排序器的工作原理

ADC 的排序器包括两个独立的最多可选择8个模拟转换通道的排序器(SEQ1 和 SEQ2)。这两个排序器可被级连成一个最多可选择16个转换模拟通道的排序器(SEQ)。图 10.1为ADC 模块工作在最多可选择16个自动转换模拟通道的排序器(SEQ) 模式下的结构框图,图10.2为 ADC 模块工作在两个最多可选择8个自动转换模拟通道的排序器(SEQ1 和 SEQ2) 模式下的结构框图。

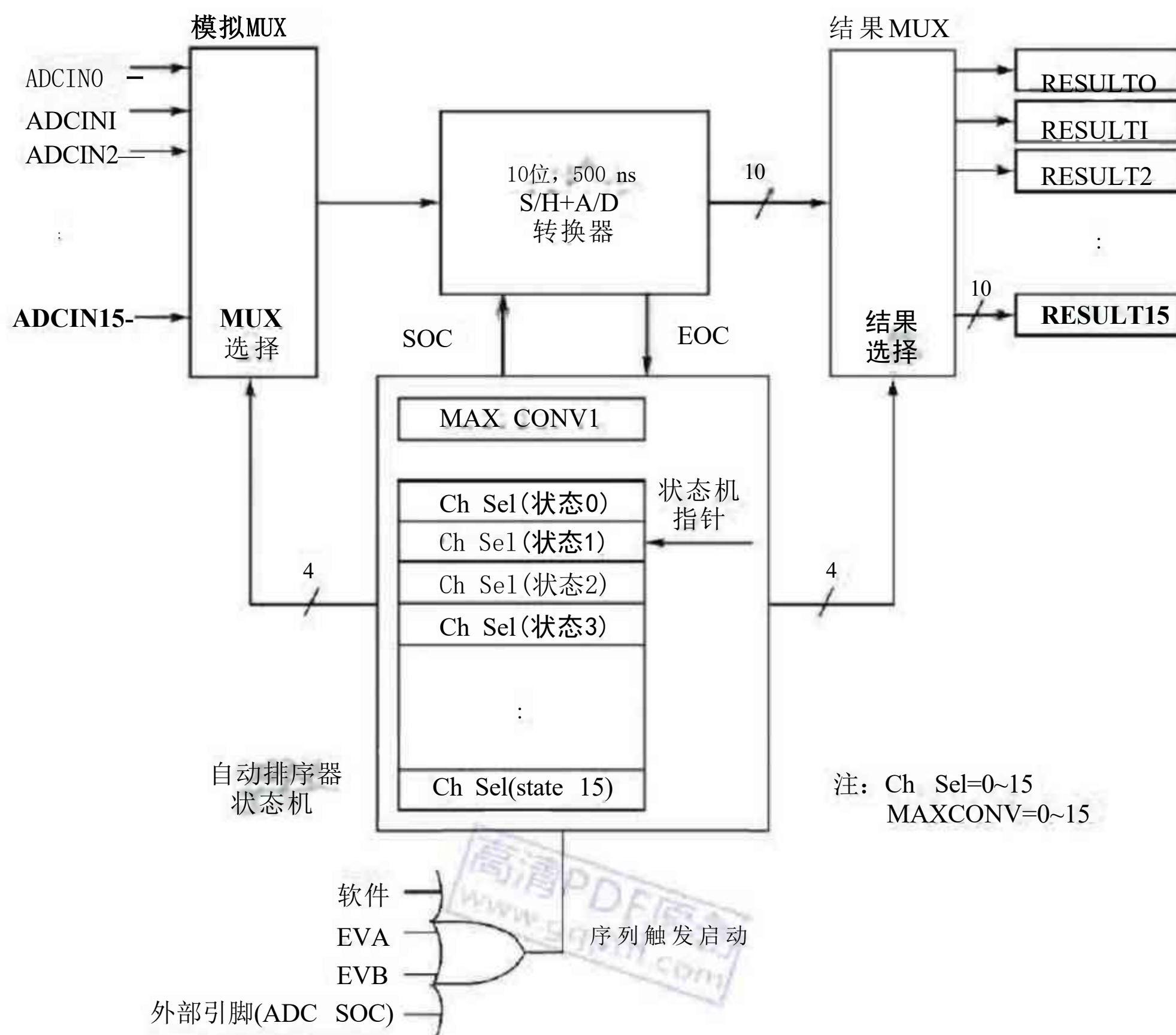
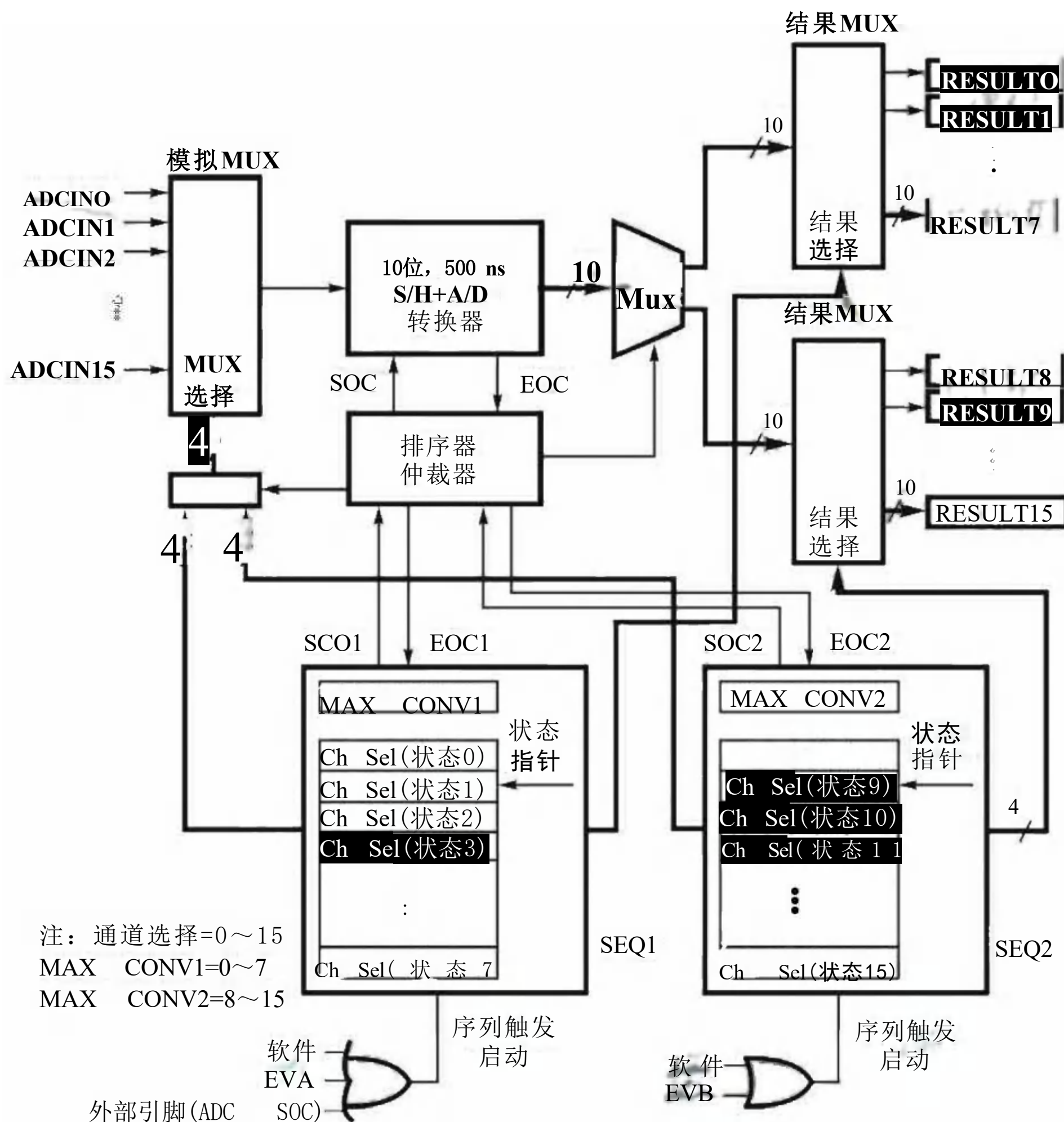


图10.1 级连工作方式下自动排序 ADC 的结构框图

在这两种工作模式下，ADC 模块都能够进行序列转换并自动排序。可通过模拟输入通道的多路选择器来选择要转换的通道。转换结束后，转换后的数值结果保存在该通道相应的结果寄存器(RESULTn) 中。即第0通道的转换结果保存在 RESULT0 中，第1通道的转换结果保存在 RESULT1 中，依此类推。而且，用户可以对同一个通道进行多次采样，即对某一通



注：在 DSP 中只有一个 A/D 变换器，这个变换器由双排序器工作模式下的两个排序器共享。

图10.2 双排序器工作方式下自动排序 ADC 的结构框图

道实行“过采样”，这样得到的采样结果比传统的采样结果分辨率高。

注意，在双排序器模式下，来自“未被激活”的排序器的 A/D 启动请求将在“被激活”的排序器完成采样之后自动开始执行。即，假设 A/D 转换正在忙于处理 SEQ2 的操作，当 SEQ1 启动一个 SOC 信号后，A/D 转换器在完成 SEQ2 的操作之后立即开始响应 SEQ1 的请求。

最多可选择8个自动转换通道双排序器工作方式和最多可选择16个自动转换通道单排序器工作方式的的操作大致相同。表10.1列出它们之间的对比情况。

为了描述方便，以后描述排序器时作以下规定：

排序器 1(SEQ1) 指 CONV00~CONV07;

- > 排序器 2(SEQ2) 指 CONV08~CONV15;
 级连排序器(SEQ) 指 CONV00~CONV15。

表10.1 双排序器和单排序器工作比较

特征参数	单8通道排序器1 (SEQ1)	单8通道排序器2 (SEQ2)	16通道级连排序器 (SEQ)
开始转换触发方式	EVA、软件和外部引脚	EVB和软件	EVA, EVB、软件和外部引脚
最大自动转换通道数(比如：排序器长度)	8	8	16
序列转换完成后自动停止	是	是	是
触发优先权	高	低	不适用
A/D转换结果寄存器	0~7	8~15	0~15
排序控制器位分配 (CHSELSEQn)	CONV00~CONV07	CONV08--CONV15	CONV00~CONV15

10.2.1 连续的自动排序模式

下文仅适用于最多可实行8个通道的自动转换排序器 (SEQ1 或 SEQ2)。该模式下，SEQ1/SEQ2 在一次排序过程中，对多达8个的任意通道进行排序转换，每次转换结果被保存到8个结果寄存器。SEQ1 的结果寄存器为 RESULT0~RESULT7,SEQ2 的结果寄存器为 RESULT8~RESULT15。

在一个排序中的转换个数受MAX CONVn(MAX CONV 寄存器中的一个3位段域或4位段域)控制，该值在自动排序的转换开始时，被装载到自动排序状态寄存器(AUTO_SEQ SR) 的排序计数器段域(SEQ CNTR3~0)。MAX CONVn段域的值在0~7范围变化。当排序器从通道CONVO0 开始有顺序地转换时，SEQ CNTRn段域的值从装载值开始向下计数直到 SEQ CNTRn为0。一次自动排序完成的转换数为 MAX CONVn+1。

【例10.1】 在双排序器模式下用SEQ1 进行 A/D 转换。

假设需要用SEQ1 来完成7个通道的转换(通道2、3、2、3、6、7和12需要排序自动转换)。则 MAX CONV1的值应设为6且CHESELQn 寄存器中应填入如表10.2所列的值。

表10.2 例10.1中CHESELQn 寄存器填入值

地 址	Bit 15~12	Bir 11~8	Bit 7~4	Bit 3~0	寄存器
70A3h	3	2	3	2	CHSELSEQ1
70A4h	x	12	7	6	CHSELSEQ2
70A5h	x	X	X	x	CHSELSEQ3
70A6h	x	X	X	X	CHSELSEQ4

注：表10.2中数值为十进制.x为无须关心的值。

连续的自动排序方式 A/D 转换流程图如图10.3所示。

在 SEQ CNTR 为0之后，如果连续转换位 CONT RUN 为1，则转换自动重新开始（即 SEQ CNTR_n装入最初的MAX CONV₁的值，并且SEQ1 的通道指针指向 CONVO0）。在这种情况下，用户必须保证下一个转换开始之前，读取结果寄存器的值。在 ADC 模块企图向结果寄存器写入数据而用户却企图从结果寄存器中读取数据时，ADC 模块的仲裁逻辑保证在发生这种冲突时不会崩溃。

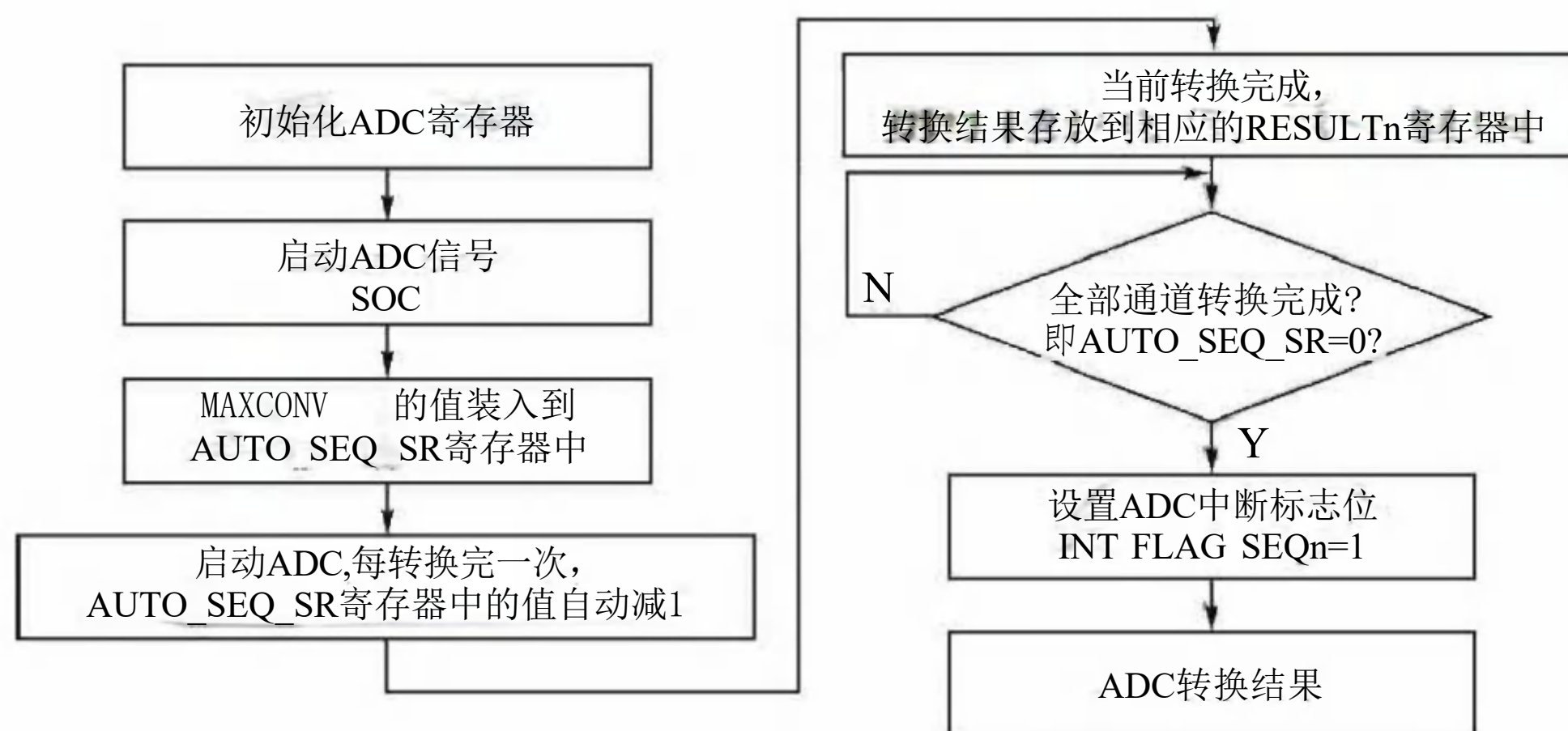


图 10.3 连续的自动排序方式 A/D 转换流程图

在 SEQ CNTR 为0之后，如果连续转换位 CONT RUN 为0，则排序器指针停留在最后状态。如：在例10.1中指向 CONV06，且 SEQ CNTR_n 保持为0。

由于在 SEQ CNTR_n 每次到达0时，中断标志位都被设置为1，所以如果需要，用户可以在中断服务子程序中用 ADCTRL2 寄存器中的 RST SEQ_n 位将排序器手动复位。复位之后，在下一个 A/D 启动信号来到时，SEQ CNTR 装入 MAX CONV_n 中的原始值，且 SEQ1 指针指向 CONVO₀，此特点在启动/停止操作发生中非常有用。上述例子也适用于 SEQ2 和 SEQ。

10.2.2 排序器的启动/停止模式

除了连续的自动排序模式外，任何一个排序器(SEQ1、SEQ2 和 SEQ) 都可工作在启动/停止模式。该模式与例10.1相同，可实现在时间上单独与多个启动信号触发源同步的功能。所不同的是，排序器完成第一个转换序列后，在中断服务程序中不需要被复位，即排序器初始指针不需要指到 CONVO0 就可以被重新触发。因此一个转换序列之后，排序器指针指到当前的通道。在这种方式下，CONT RUN 位必须被设置为0。

【例10.2】 排序器工作在启动/停止方式

要求：触发源1(定时器下溢)启动3个自动转换(I1、I2 和 I3)，触发源2(定时器周期)启

动3个自动转换(V1、V2 和 V3)。触发源1和触发源2在时间上是分开的，即间隔25 μ s，如图10.4所示。只有SEQ1被用到。

注：触发源1和2可以是来自事件管理器 EVA，外部引脚或软件的 SOC 信号。例10.2中需要的两个触发源可以用同一个触发源发生两次来满足。

在这种情况下，MAX CONV1的值被设置为2. 且 ADC 模块的输入通道选择排序控制寄存器应填入如表10.3所列的值。

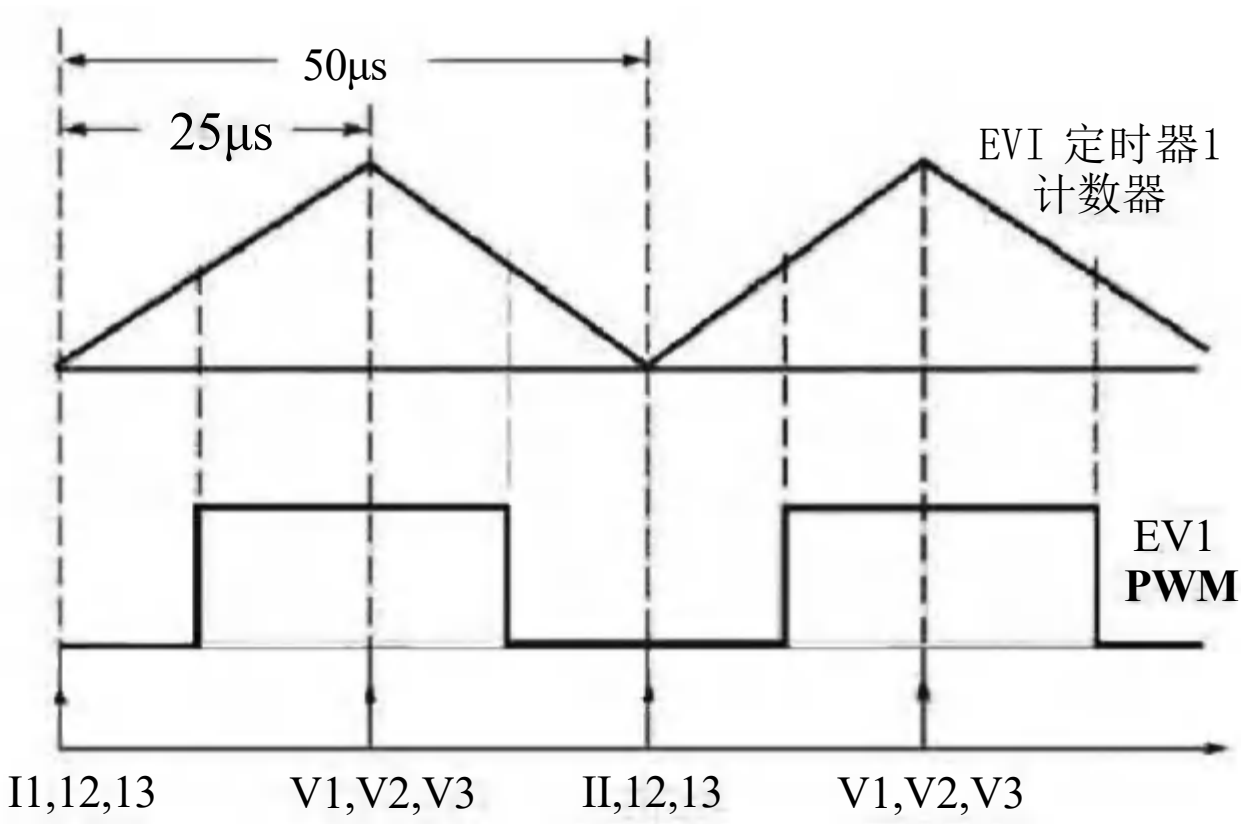


图10.4 事件管理器A(EVA)启动排序器

表10.3 例10.2中ADC 模块的输入通道选择排序控制寄存器填入值

地 址	Bit 15~12	Bit 11~8	Bit 7~4	Bit 3~0	寄存器
70A3h	VI	13	12	11	CHSELSEQ1
70A4h	X	X	V3	V2	CHSELSEQ2
70A5h	X	X	X	X	CHSELSEQ3
70A6h	X	X	X	X	CHSELSEQ4

一旦复位和初始化之后，SEQ1 就开始等待触发源的到来。第一个触发源到来之后，通道选择值为 CONVO0(II)、CONVO1(I2) 和 CONVO2(I3) 的3个转换被执行。转换完成之后，SEQ1 等待另一个触发源的到来。25 μ s 之后另一个触发源到来，ADC 模块开始另外3个通道选择值为 CONVO3(V1)、CONVO4(V2) 和 CONVO5(V3) 的转换。

在两种触发源的情况下，MAX CONV1的值都被自动装入 SEQ CNTR1 中。如果第二个触发源要求转换的个数与第一个不一样，那么用户必须在第二个触发源来到之前，通过软件改变 MAX CONV1的值；否则ADC 模块将重新使用原来的 MAX CONV1中的值。用户可以在第一个触发源引起的转换完成之后的中断复位程序中改变 MAX CONV1的值，为第二个触发源引起的转换个数作准备。

在第二个转换序列完成之后，ADC 模块的结果寄存器中得到如表10.4所列的值。

在第二个触发源转换序列完成后，SEQ1 在当前状态保持等待状态，直到另一个触发源来到为正。用户可通过软件复位SEQ1 将排序器指针指到CONVO0，并重复同样的触发源1和2转换操作。

表10.4 例 10.2 中 ADC 模块的结果寄存器所得值

缓冲寄存器	ADC结果缓冲区	缓冲寄存器	A D C 结 果 缓 冲 区
结果寄存器0 (RESULT0)	11	结果寄存器8 (RESULT8)	X
结果寄存器1 (RESULT1)	12	结果寄存器9 (RESULT9)	X
结果寄存器2 (RESULT2)	13	结果寄存器10 (RESUI. T10)	X
结果寄存器3 (RESULT3)	14	结果寄存器11 (RESUI. T11)	X
结果寄存器4 (RESULT4)	15	结果寄存器12 (RESUI. T12)	X
结果寄存器5 (RESULT5)	16	结果寄存器13 (RESUL. T13)	X
结果寄存器6 (RESULT6)	X	结果寄存器14 (RESULT14)	X
结果寄存器7 (RESULT7)	X	结果寄存器15 (RESULT15)	X

10.2.3 输入触发源

每一个排序器都有一套能被使能或禁止的触发源。 SEQ1、SEQ2 和 SEQ 的有效输入触发源见表10.5。

表 10.5 排序器有效输入触发源

排序器1 (SEQ1)	排序器2 (SEQ2)	级连排序器 (SEQ)
软件触发 事件管理器A (EVA SOC) 触发 外部引脚 (ADC SOC) 触发	软件触发 事件管理器B (EVB SOC) 触发	软件触发 事件管理器A (EVA SOC) 触发 事件管理器B (EVB SOC) 触发 外部引脚 (ADC SOC) 触发

- 注：1. 排序器处于空闲状态是指：在接收到一个触发源之前，排序器指针指到CONVOo 或者排序器已经完成一个转换序列(即 SEQCNTR 为0)。无论何时，只要一个排序器处在空闲状态， 一个启动信号触发就可启动一个自动转换序列。
2. 如果一个新的启动触发信号来到时，当前转换序列正在进行，则将 ADCTRL2 寄存器中的SOC SEQn 位置 1 (该位在前一个转换开始时被清0)。然而如果此时 SOC SEQn 位已经为1, 则该启动触发信号将丢失。
3. 一旦被触发，排序器就不能在转换中途被停止或中断。程序必须等到一个序列的停止信号再继续操作，或者对排序器进行复位。复位使排序器立即返回到空闲的起始状态(SEQ1 和级连的排序器指针指到CONVoo, SEQ2 的排序器指针指到CONVO8)。
4. 当 SEQ1/2 工作在级连方式下，到SEQ2 的触发源被忽略，而到SEQ1 的触发源仍然有效。因此级连方式可以看做是SEQ1 有最多16个转换通道而非8个转换通道。

10.2.4 在排序转换时的中断操作

排序器可以在两种工作模式下产生中断，这两种方式由ADCTRL2 寄存器中的中断模式

控制位决定。

对例10.2稍作变动，就可说明在不同的工作条件下中断方式1和中断方式2的用途。在排序转换时的中断操作如图10.5所示。

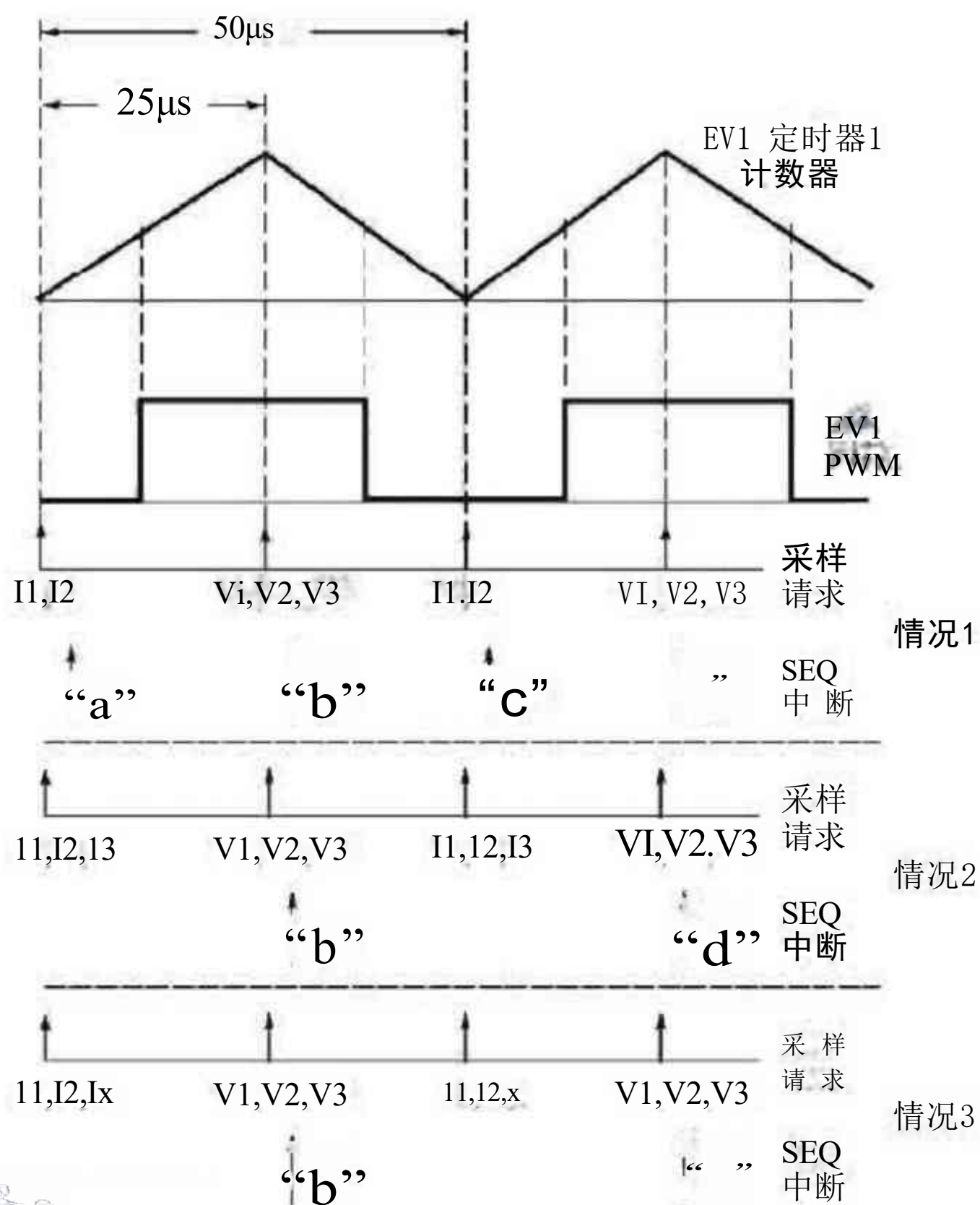


图 10.5 在排序转换时的中断操作图

1. 情况1: 第一个序列和第二个序列中采样的个数不一样

在这种情况下使用中断方式1(即在每次 EOS 到来时，产生中断请求)。

(1) 排序器设置 MAX CONV_n=1以转换I1 和 I2。

(2) 在中断服务子程序a 中，将MAX CONV_n的值改为2, 以转换 V1、V2 和 V3。

(3) 在中断服务子程序b 中完成以下操作：

- 将 MAX CONV_n 的值改为1, 以转换I1 和 I2；
- I1、I2、V1、V2 和 V3 的值从ADC 结果寄存器中读出；
- 排序器被复位。

(4) 重复第(2)步和第(3)步。

注意：中断标志位在每次 SEQ CNTR 到0时都被置1, 且产生两次中断。

2. 情况2:两个序列的采样个数相等

在这种情况下使用中断方式2(即每隔一个 EOS 信号, 产生一次中断请求)。

(1) 排序器设置 MAX CONVn=2 以转换 I1、I2 和 I3(或者 V1、V2 和 V3)。

(2) 在中断服务子程序 b 或 d 完成以下操作:

·I1、I2、I3、V1、V2 和 V3 的值从 ADC 结果寄存器中读出;

●排序器被复位。

(3) 重复第(2)步和第(3)步。

注意：中断标志位在每次 SEQ CNTR 到0时都被置1, 即在 ADC 转换完 I1、I2 和 I3 或者 V1、V2 和 V3 之后中断标志位都被置1。但是只有转换完 V1、V2 和 V3 之后才产生中断。

3. 情况3:两个序列的采样个数相等(带虚读)

在这种情况下使用中断方式2(即每隔一个 EOS 信号, 产生一次中断请求)。

(1) 排序器设置 MAX CONVn=2以转换 I1、I2 和 x。

(2) 在中断服务子程序 b 或 d 完成以下操作:

·I1、I2、x、V1、V2 和 V3 的值从ADC 结果寄存器中读出;

●排序器被复位。

(3) 重复第(2)步和第(3)步。

注意：第 3 个x 采样为一个假采样, 并未要求采样。然而, 为了使中断服务子程序的开销和CPU 的干扰最小, 可利用中断方式2的中断请求特性。

10.3 ADC 时钟预定标

可调整F240xA 器件 ADC 的采样/保持模块, 来适应输入信号阻抗的变化。可选用改变 ADCTR1 寄存器中的 ACQ PS3~ACQ PSO 位段域和 CPS 位来实现。A/D 转换可被分成两个时间段, 见图10.6。

如果 ACQ PS3~ACQ PSO 位段域的值全为0, 即预定标器的值为1, 并且CPS 为 0 时, PS 时钟将同CPU 时钟一样。对于预定标器的任何其他值, PS 都会被放大(即增加采样/保持窗口的时间)。PS 具体的放大倍数见表10.7对 ACQ PS3~ACQ PSO 位段域的描述。如果 CPS 为1, 则S/H 窗口长度为原来的2倍。被扩大为原来的S/H 窗口再加上被预定标器拉长的倍数才是最后的 PS。图10.7表明了各个预定标位的作用。

注意：在CPS 为 0 时, PS 和 Ac.k将和CPU 时钟相等。

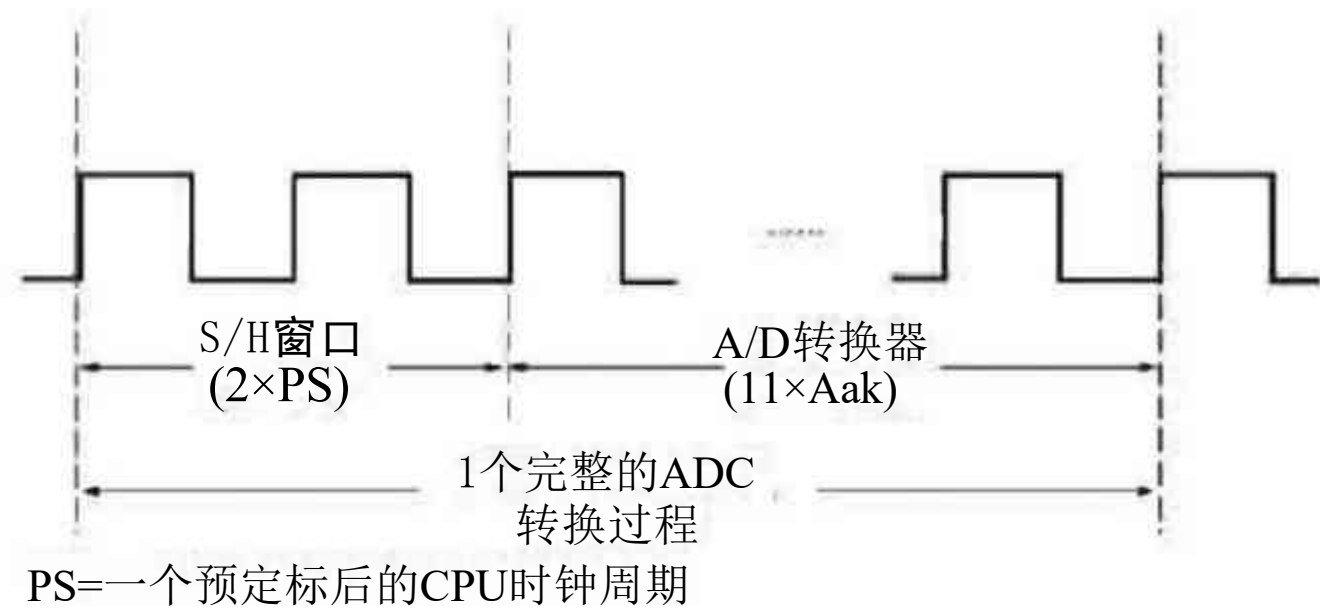


图10.6 A/D 转换时间

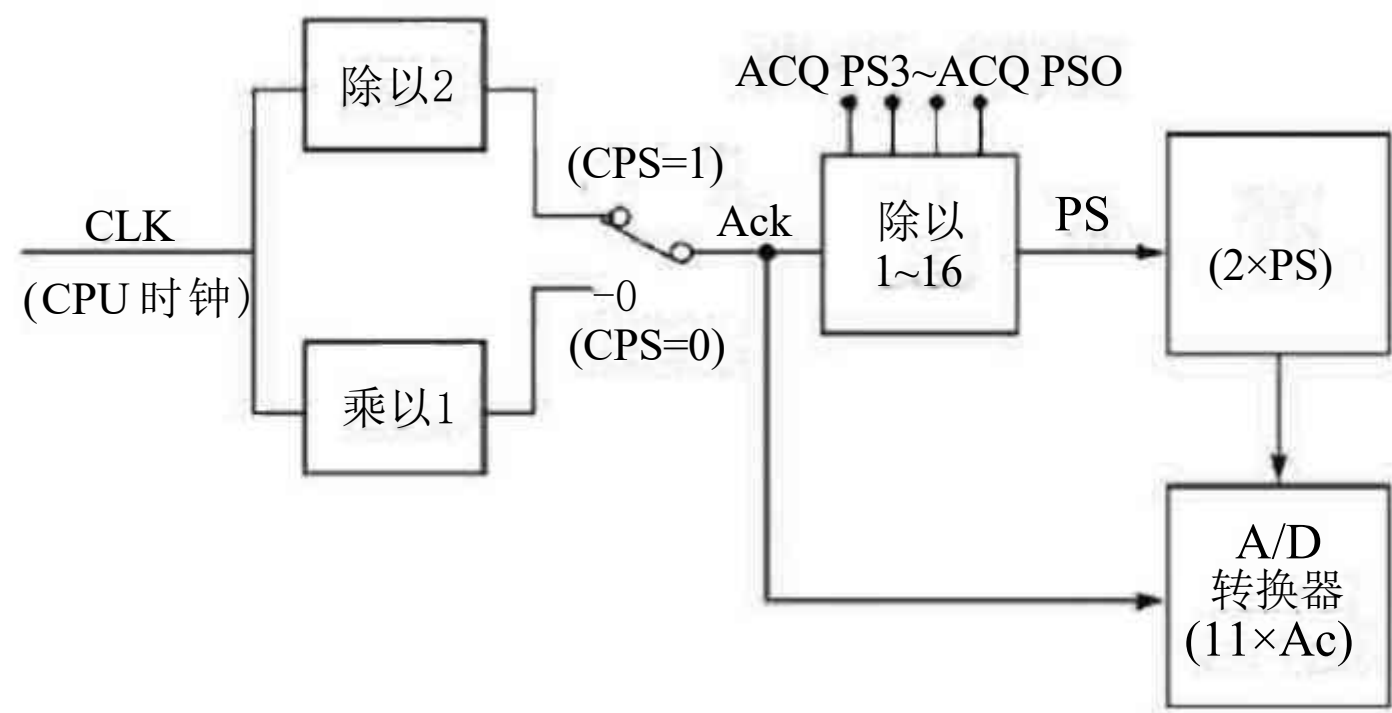


图10.7 F240xA ADC模块的时钟预定标

10.4 校准模式

在校准方式下，ADCIN_n 引脚未接A/D 转换器，且不能对排序器进行操作。接收A/D 转换器输入端的信号为 BRG ENA 位(桥使能)和 HI/LO 位(VREFHI/VREFLO)。 这两位将 VREFHI、VREFLO 或者其中间值送到A/D 转换器的输入端，ADC 模块完成一个转换。校准模式可计算ADC 模块的零点、中点和最大值时的偏移误差。该偏移误差值的二进制补码被保存在校准寄存器(CALIBRATION) 中(二进制补码操作只适用于误差值为负的情况)。在此基础上，ADC 硬件自动地将偏移误差量加到转换值上。

校准寄存器(CALIBRATION)——地 址70B8h

15	14	13	12	11	10	9	8
D9	D8	D7	D6	D5	D4	D3	D2
7	6	5	4	3	2	1	0
DI	D0	0	0	0	0	0	0

总之，CALIBRATION 寄存器中存放校准模式下的最后结果。在正常模式下时，ADC 转换结果被保存到结果寄存器之前，CALIBRATION 寄存器中的值被自动加到 ADC 转换结果的输出。

以下举一个校准实例。

在校准模式下，ADC 将转换结果保存在 CALIBRATION 寄存器中。在校准转换结束之后，INT_FLAG_SEQ1 位置1。如果中断被使能，则产生中断。此时用户软件必须读取 CALIBRATION 寄存器的值，以判断该值对理想值的偏移量有多大，然后在离开这种校准方式之前，将该偏移量写回 CALIBRATION 寄存器中。在后来的转换值被送到结果寄存器之前，此值将被加到转换值上(不是在校准模式)。

为了得到精确的结果，校准操作必须作两次，即在保证已经完成第一个校准操作后，应当立即启动第二个校准操作(这将覆盖第一次校准操作的结果)。

例1: 如果选择 VREFHI 作校准操作，则期望的结果是3FFh。如果从 CALIBRATION 寄存器读出的校准转换结果为3FDh，那么用户应当在离开校准模式之前，将值0002h 写到 CALIBRATION 寄存器中。这个值将对以后的转换值进行两个数的调整。

例2: 如果选择 VREFLO 作校准操作，则期望的结果是0000h。如果从 CALIBRATION 寄存器读出的校准转换结果为0003h，那么用户应当在离开校准模式之前，将值0003h 写到 CALIBRATION 寄存器中。这个值将对以后的转换值进行3个数的调整。

校准模式不能自动完成的原因是，某些用户可能只希望对不同的点进行测量并将测量结果进行平均，而另一些用户则只关心某一点，如中点。这根据用户需要而定。

10.5 自测试模式

自测试模式用来检测ADC 引脚的短路/开路。在这种情况下，采样周期为正常模式的两倍。在采样周期的前半部分，除了用户提供的模拟输入信号外，VREFHI 或者 VREFLO 被接到 ADC 转换器的输入；在采样周期的后半部分，只有用户提供的信号被接到 ADC 转换器的输入。假设使用 VREFHI 作自测试，且 ADC 引脚为开路，则结果寄存器中只包含表示 VREFHI 的数字值。

自测试模式只能用来检测短路或开路，在正常工作模式下不能使用。而且正常模式和自测试模式不能同时使用。

10.6 ADC 模块的寄存器

ADC 模块寄存器如表10.6所列。

表10.6 ADC 模块寄存器地址列表

地 址	寄存器	名 称	地 址	寄存器	名 称
70A0h	ADCTRL. I	ADC控制寄存器1	70ADh	RESULT5	转换结果缓冲寄存器5
70AIh	ADCTRL2	ADC控制寄存器2	70AEh	RESUL. T6	转换结果缓冲寄存器6
70A2h	MAX CONV	最大转换通道寄存器	70AFh	RESULT7	转换结果缓冲寄存器7
70A3h	CHSELSEQ1	通道选择排序控制寄存器1	70Boh	RESUL. T8	转换结果缓冲寄存器8
70A4h	CHSELSEQ2	通道选择排序控制寄存器2	70B1h	RESUL. T9	转换结果缓冲寄存器9
70A5h	CHSELSEQ3	通道选择排序控制寄存器3	70B2h	RESUIT10	转换结果缓冲寄存器10
70A6h	CHSELSEQ4	通道选择排序控制寄存器4	70B3h	RESULT11	转换结果缓冲寄存器11
70A7h	AUTO_SEQ_SR	自动排序状态寄存器	70B4h	RESULT12	转换结果缓冲寄存器12
70A8h	RESULT0	转换结果缓冲寄存器口	70B5h	RESULT13	转换结果缓冲寄存器13
70A9h	RESULT1	转换结果缓冲寄存器1	70B6h	RESULT14	转换结果缓冲寄存器14
70AAh	RESULT2	转换结果缓冲寄存器2	70B7h	RESUI. T15	转换结果缓冲寄存器15
70ABh	RESULT3	转换结果缓冲寄存器3	70B8h	CALIBRATION	校准结果寄存器. 用来校正转换结果
70ACh	RESULT4	转换结果缓冲寄存器4			

1.ADC 控制寄存器1(ADCTRL1)——地 址70A0h

15	14	13	12	11	10	9	8
保留位	RESET	SOFT	FREE	ACQ PS3	ACQ PS2	ACQ PS1	ACQ PS0
	RW_o	RW_0	RW_0	RW0	RW 0	RW0	RW 0
7	6	5	4	3	2	1	0
CPS	CONTRUN	INT PRI	SEQ CASC	CAL ENA	BRG ENA	HI/1 ()	STEST ENA
RW_0	RW_0	RW_0	RW_0	RW_0	RW_0	RW_o	RW_0

注：R= 可读；W= 可写；0=复位值。

位15 保 留 位 。

位14 复位位。ADC 模块软件复位位。此位引起一个对整个 ADC 模块的主动复位。所有的寄存器和排序器指针都复位到器件复位引脚被拉低或者上电复位时的初始状态 。

0 无影响；

1 复位整个 ADC 模块。

注：ADC 模块可在系统复位时被复位，但用户可通过向此位写1来实现，在

任何时候对 ADC 模块的复位。一个 NOP 指令后，就可向 ADCTRL1 寄存器中写入相应的值。

SPLK #01xxxxxxxxxxxxxb,ADCTRL1 ;复位整个 ADC 模块(RESET=1)

NOP ;保证写入 ADCTRL1 寄存器所需的延时

SPLK #00xxxxxxxxxxxxxb,ADCTRL1 ;清除 ADC 的复位位(RESET=0)

在缺省设置足够时，复位 ADC 模块之后，也可不向 ADCTRL1 中写数。

位 13~12 SOFT 位和 FREE 位。这两位决定仿真悬挂时 ADC 模块的工作情况。在自由运行模式下，ADC 模块的运行不受仿真影响，在停止模式下，仿真悬挂时，ADC 模块可立即停止或者完成当前操作之后停止。

SOFT FREE

0 0 仿真悬挂时，ADC 模块立即停止；

1 0 仿真悬挂时，ADC 模块完成当前转换后停止；

x 1 自由运行，继续运行而不管仿真悬挂。

位 11~8 采样时间选择位 ACQ PS3~ACQ PSO。这几位决定了 ADC 时钟的预定标系数，见表 10.7。

位 7 CPS 位。转换时钟预定标位。该位决定了 ADC 转换逻辑时钟的预定标。

0 FCLK=CLK/1；

1 FCLK=CLK/2。

其中 CLK 为 CPU 时钟频率。

位 6 连续转换位 CONT RUN。该位决定排序器工作在连续转换模式或者启动/停止模式。用户可在当前转换序列正被执行时向该位写数，但是只有在当前转换序列完成之后才生效。在连续模式下，用户不用对排序器复位，但在启动/停止模式下，排序器必须被复位，以使排序器指针指到 CONV00。

0 启动/停止模式；

1 连续转换模式。

位 5 ADC 中断请求优先级位 INT PRI。

0 高优先级；

1 低优先级。

位 4 级连排序器工作方式位 SEQ CASC。

0 双排序器工作模式，SEQ1 和 SEQ2 作为两个最多可选择 8 个转换通道的排序器；

1 级连模式，SEQ1 和 SEQ2 级连起来作为一个最多可选择 8 个转换通道的排序器 SEQ。

- 位3

偏差校准使能位CAL ENA。

9禁止校准模式；

1使能校准模式。
- 位 2

桥使能位 BRG ENA,见表10. 8。

0满的参考电压被接到 ADC 输入；

1参考的中点电压被接到 ADC 输入。
- 位 1

VREFHI 和 VREFLO) 选择位 HI/LO, 见表10. 8。

0用 VREFLO 作为 ADC 输入的值；

1用 VREFHI 作为 ADC 输入的值。
- 位 0

自测试使能位STEST ENA。

0禁止自测试模式；

1使能自测试模式。

表10.7 A/D 模块的预定标系数

CPU时钟为30 MHz时ADC模块的预定标系数

井	ACQ PS3	ACQ PS2	ACQ PS1	ACQ PS0	预定标因子 (除以)	采样窗口时间	信号源阻抗Z/Q CPS=0)	信号源阻抗Z/ Ω (CPS=1)
0	0	0	0	0	1	2×TcLK	67	385
1	0	0	0	I	2	4XTcLk	385	1020
2	0	0	1	0	3	6XTcrk	702	1655
3	0		I	I	4	8×TeLk	1020	2290
4	0	1	0	0	5	10×TeLk	1337	2925
5	0	1	0	1	6	12×TcLK	1655	3560
6	0	1	1	0	7	14×TcLK	1972	4194
7	0	1	1	1	8	16×TcLK	2290	4829
8	1	0	0	0	9	18×Te. K	2607	5464
9	1	0	0		10	20×TcLK	2925	6099
A	I	0	I	0	11	22×TcLK	3242	6734
B	1	0	1	1	12	24×TetK	3560	7369
C	1	1	0	0	13	26×TetK	3877	8004
D	1	1	0	1	14	28×TetK	4194	8639
E	1	1	1	0	15	30×TetK	4512	9274
F		1	1	1	16	32×TctK	4829	9909

续表10. 7

CPU时钟为40 MHz时ADC模块的预定标系数								
井	ACQ PS3	ACQ PS2	ACQ PSI	ACQ PS0	预定标因子 (除以)	采样窗口时间	信号源阻抗Z/ Ω (CPS=0)	信号源阻抗Z/ Ω (CPS=1)
0	(	0	0	0	1	2×TcLK	53	291
1	0	0	0	1	2	4×TcLK	291	767
2	0	0	1	0	3	6×TcLK	529	1244
3	0	0	1	1	4	8×TcLK	1767	1720
4	0	1	0	0	5	10×Tc. K	1005	2196
5	0	1	0	1	6	12×Tc. K	1244	2672
6	0	1	1	0	7	14×TcLK	1482	3148
7	)	1	1	1	8	16×TcLK	1720	3625
8	1	0	0	0	9	18×TcLK	1958	4101
9	1	0	0	1	10	20×TcLK	2196	4577
A	1	0	1	0	11	22×TcLK	2434	5053
B	1	0	1	1	12	24×TcLK	2672	5529
C	1	1	0	0	13	26×TcLK	2910	6005
D	1	1	0	1	14	28×TcLK	3148	6482
E	1	1	1	0	15	30×TcLK	3386	6958
F	1	1	1	1	16	32×TcLK	3625	7434

注：① TcLk取决于转换时钟预定标因子位(Bit 7).例如：
CPS=0:TcLK=1/CLK （例：对于CLK=40 MHz,TcLK=25 ns)
CPS=1:TcLK=2×(1/CLK) （例：对于CLK=40 MHz,TcLK=50 ns)
② 信号源阻抗Z 仅为估计值。

表 10.8 参考电压位选择定义

BRG ENA	HI/LO)	CAL ENA=1 参考电压/V	参考电压/V
0	0	VREFLO	VREFLO
0	1	VREFHI	VREFHI
1	(	(VREFHI-VREFLO) /2	VREFLO
1	1	(VREFLO-VREFHi) /2	VREFHI

2.ADC 控制寄存器2(ADCTRL2)—— 地址70A1h

15	14	13	12	11	10	9	8
EVB SOC SEQ	RST SEQ1 STRT CAL	SOC SEQ1	SEQ1 BSY	INT ENA SEQ1 (模式1)	INT ENA SEQ1 (模式0)	INT FLAG SEQ1	EVA SOC SEQ1
RW_0	RS 0	RW_o	R_0	RW_o	RW_o	RC_0	RW0
7	6	5	4	3	2	1	0
EXT SOC SEQ1	RST SEQ2	SOC SEQ2	SEQ2 BSY	INT ENA SEQ2 (模式1)	INT ENA SEQ2 (模式0)	INT FLAG SEQ2	EVBSOC SEQ2
RW_0	RS_0	RW_0	R_0	RW_0	RW_0	RC_0	RW_0

注: R=可读; W= 可写; S= 仅为设置; C= 清除; 0-复位后的值

- 位15 EVB SOC SEQ 位。EVB 的 SOC 信号使能级连排序器位。
- 0 不起作用;
- 1 允许级连的排序器 SEQ 被事件管理器 B 的信号启动。
- 位14 RST SEQ1/STRT CAL 位。复位排序器1/启动校准转换方式位。
- 在校准转换方式禁止的情况下:
- 0 不起作用;
- 1 立即复位排序器, 使排序器指针指到CONVo0。
- 在校准转换方式使能的情况下:
- 0 不起作用;
- 1 启动校准转换方式。
- 位13 SOC SEQ1 位。启动SERQ1 转换位。以下触发源可引起此位置 I:
- S/W 软件向该位写1;
- EVA 事件管理器 A;
- EVB 事件管理器B(仅在级连模式下);
- EXT 外部引脚(即 ADC SOC引脚)。
- 当一个触发源到来时, 有3种情况可能发生:
- SEQ1 处于空闲状态且 SOC 位 清 0 时 , SEQ1 立即启动(在判优仲裁控制下);该位置工后立即清0, 允许后来的触发源被悬挂。
 - SEQ 处于忙状态但SOC 位为0时, 该位置1以表示一个触发源请求正被悬挂; 当SEQ1 完成当前的转换又重新开始时, 该位清0。



• SEQ1 处于忙状态且SOC 位已经置1时，此时来到的触发源被忽略。

0 清除一个悬挂的SOC 请求；

1 软件触发启动 SEQ1。

注：当RST SEQ1 位和 SOC SEQ1位在同一个指令中被设置时，将复位排序器而不会启动排序器。正确的操作顺序应该是先设置 RST SEQ1 位，再在下一条指令中设置SOC SEQ1 位，这就可以保证复位一个排序器之后，再将其重新启动。对RST SEQ2 位和 SOC SEQ2 位的操作也是如此。

位12 SEQ1 BSY 位。SEQ1 忙状态位。

0 SEQ1 处于空闲状态；

1 SEQ1 处于忙状态，一个转换序列正在进行。

位11~10 对 SEQ1 的中断方式使能控制位见表10.9。

表10.9 位 11~10 中断方式

位11	位10	中断方式说明
0	0	中断不使能
0	1	中断模式1 在中断标志位 (INT FLAG SEQ1)置1时，立即申请中断
1		中断模式2 当INT FLAG SEQ1标志已经设置时，产生中断请求。如果清除了+, 则INT FLAG SEQ1置1, 中断请求被屏蔽。(模式2允许其他排序结束产生中断申请)
1	1	保留

注：+ 如果有两个排序通道要求申请中断，则最后完成的排序通道将具有较高的优先级。

位 9 INT FLAG SEQ1 位。ADC 模块SEQ1 的中断标志位。

0 无中断事件发生；

1 有中断事件发生。

位 8 EVA SOC SEQ1位。事件管理器 A 对 SEQ1 产生 SOC 信号的屏蔽位。

0 SEQ1 不能被 EVA 的触发源启动；

1 允许 SEQ1/SEQ 被 EVA 的触发源启动。

位 7 EXT SOC SEQ1 位。外部信号对 SEQ1 的启动转换位。

0 不起作用；

1 允许一个来自ADCSOC 引脚上信号，启动ADC 自动转换序列。

位 6 RST SEQ2 位。复位排序器2。

0 不起作用；

1 立即复位排序器，使排序器指针指到CONV00。

位 5 SOC SEQ2 位。启动SEQ2 转换位(仅适用于双排序器模式)。

以下触发源可引起该位置1：

S/W 软件向该位写1；

EVB 事件管理器B。

当一个触发源到来时有3种情况可能发生：

- SEQ2 处于空闲状态且 SOC 位 清 0 时，SEQ2 立即启动(在判优仲裁控制下)；该位置1后立即清0，允许后来的触发源被悬挂。
- SEQ2 处于忙状态但 SOC 位为0时，该位置1以表示一个触发源请求正被悬挂；当SEQ2 完成当前的转换又重新开始时，该位清0。
- SEQ2 处于忙状态且SOC 位已经置1时，此时来到的触发源被忽略。

0 清除一个悬挂的 SOC 请求；

1 软件触发启动SEQ1。

位 4 SEQ2 BSY 位。SEQ2 忙状态位。

0 SEQ2 处于空闲状态；

1 SEQ2 处于忙状态，一个转换序列正在进行。

位3~2 对 SEQ2 的中断方式使能控制位见表10. 10。

表10. 10 位3~2中断方式

位3	位2	中断方式说明
0	0	中断不使能
0	1	中断模式1 在中断标志位(1NT FLAG SEQ2)置1时，立即申请中断
1	C	中断模式2 仅当INT FLAG SEQ1标志已经设置时，产生中断请求。如果清除了土, 则INT FLAG SEQ1置1. 中断请求被屏蔽。(模式2允许其他排序结束产生中断申请)
1	I	保留

注：丰如果有两个排序通道要求申请中断. 则最后完成的排序通道将具有较高的优先级。

位 1 INT FLAG SEQ2 位。ADC 模块 SEQ2 的中断标志位。

0 无中断事件发生；

1 有中断事件发生。

位 0 EVA SOC SEQ2 位。事件管理器 B 对 SEQ2 产生 SOC 信号的屏蔽位。

0 SEQ2 不能被EVB 的触发源启动；

1 允许 SEQ2 被 EVB 的触发源启动。

3. 最大转换通道寄存器(MAX CONV)——地 址70A2h

15~8							
保留位							
R_x							
7	6	5	4	3	2	1	0
保留位	MAX CONV2_2	MAX CONV2_1	MAX CONV20	MAX CONV1_3	MAX CONV ₁ 2	MAX CONV ₁ 1	MAX CONV ₁ 0
RW_0	RW_0	RW_0	RW_0	RW_0	RW_0	RW_0	RW_0

位15~7 保留位。

位6~0 MAX CONV_n位域。该位域决定了一次自动转换最多转换的通道个数。该位域及其操作随着排序器工作模式的变化而变化。

【例10.3】 对 MAX CONV 寄存器位的编程。

MAX CONV₁的位定义与转换个数之间的关系如表10.11所列。如果需要进行5个转换，则MAX CONV 设置为4。在双排序器模式下使用 SEQ1 或者在级连模式下使用SEQ时，排序器指针依次从CONV00 指到CONV04，并且这5个转换结果依次被存放在转换结果寄存器00~04中。在双排序器模式下使用 SEQ2 时，排序器指针依次从 CONV08 指到CONV12，并且这5个转换结果依次被存放在转换结果寄存器08~12中。

当 SEQ1 工作在双排序器模式下，而写入 MAX CONV₁中不宜超过7时，SEQ CNTR_n超过7之后继续计数，使得排序指针重新指到CONV00 并继续转换。见表10.11。

表10.11 MAX CONV₁ 的位定义和转换个数之间的关系

MAX CONV1.3~0	转换个数	MAX CONV1.3~0	转换个数	MAX CONV1.3~0	转换个数
0000	1	0110	7	1100	13
0001	2	0111	8	1101	14
0010	3	1000	9	1110	15
0011	4	1001	10	1111	16
0100	5	1010	11		
0101	6	1011	12		

对 SEQ1 操作，使用位域 MAX CONV₁2~0;

对SEQ2 操作，使用位域 MAX CONV2_2~0;

对 SEQ 操作，使用位域 MAX CONV₁3~0。

4. 自动排序状态寄存器 (AUT O_{....}SEQ SR)——地 址70A7h

15~12				11	10	9	8
保留位				SEQ CNTR3	SEQ CNTR2	SEQ CNTR1	SEQ CNTR0
R_x				R_0	R_0	R_0	R_0
7	6	5	4	3	2	1	0
保留位	SEQ2- State2	SEQ2- State1	SEQ2- State0	SEQ1- State3	SEQ1- State2	SEQ1- State1	SEQ1- State0
R_x	R_0	R_0	R_0	R_0	R_0	R_0	R_0

位15~12 保留位。

位11~8 SEQ CNTR3~SEQ CNTR0。排序器计数状态位见表10. 12。

位 7 保留位。

位 6~4 SEQ2-State2~SEQ2-State0。这几位反映了SEQ2 排序器指针的状态。如果需要，那么用户可以根据这几位的值，在结束转换(EOS) 信号来到之前，读取中间结果。

位3~0 SEQ1-State3~SEQ1-State0。这几位反映了SEQ1 排序器指针的状态。如果需要，那么用户可以根据这几位的值，在结束转换 EOS 信号来到之前，读取中间结果。

每一个4位域都可为自动转换选择16个模拟输入通道中的任意一个，见表10.13。

表10. 12 SEQ CNTRn 位定义

SEQ CNTRn (只读)	剩余转换个数	SEQ CNTRn (只读)	剩余转换个数
0000	1	1000	9
0001	2	1001	10
0010	3	1010	11
0011	4	1011	12
0100	5	1100	13
0101	6	1101	14
0110	7	1110	15
0111	8	1111	16

表10.13 ADC输入通道选择定义

CON V nn值	ADC输入通道选择	CON V nn值	ADC输入通道选择
0000	通道0	1000	通道8
0001	通道1	1001	通道9
0010	通道2	1010	通道10
0011	通道3	1011	通道11
0100	通道4	1100	通道12
0101	通道5	1101	通道13
0110	通道6	1110	通道14
0111	通道7	1111	通道15

5.ADC 输入通道选择排序控制寄存器(CHSELSEQn)——地址70 A3h~70A6h

15~12	11~8	7~4	3~0	
CONV03	CONV02	CONV01	CONV00	CHSELSEQ1
RW_x	RW_0	RW_0	RW_0	
15~12	11~8	7~4	3~0	
CONV07	CONV06	CONV05	CONV04	CHSELSEQ2
RW_x	RW_0	RW_0	RW_0	
15~12	11~8	7~4	3~0	
CONV11	CONV10	CONV09	CONV08	CHSELSEQ3
RW_x	RW_0	RW_0	RW_0	
15~12	11~8	7~4	3~0	
CONV15	CONV14	CONV13	CONV12	CHSELSEQ4
RW_x	RW_0	RW_0	RW_0	

6.ADC 转换结果缓冲寄存器(RESULTn)——地址70A8h~70B7h

15	14	13	12	11	10	9	8
D9	D8	D7	D6	D5	D4	D3	D2
7	6	5	4	3	2	1	0
D1	D0	0	0	0	0	0	0

注：1. 缓冲区地址为70A8h~70B7h(16个寄存器)；
2. 10位转换结果放在 D9~D0。

10.7 ADC 转换时钟周期

转换时间与一个给定的序列中转换的个数有关。转换周期可被分成5个阶段，见表10. 14。

- >启动时序同步(SOS 同步)，SOS 同步只在一个转换序列的第一个通道转换时需要；
- 采样时间(ACQ)；
- >转换时间(CONV)；
- >结束转换时间(EOC),ACQ、CONV 和 EOC 在每个转换中都需要；
- >序列转换完成后，设置标志位时间 EOS，该时间仅在一个序列的最后一个转换时需要。

表10. 14 ADC 转换各个阶段需要的CLKOUT 时钟周期

转换阶段	CLKOUT时钟 周期(CPS=0)	CLKOUT时钟 周期(CPS=1)	转换阶段	CLKOUT时钟 周期(CPS=0)	CLKOUT时钟 周期(CPS=1)
启动SOS时序同步	2	2或3	结束转换时间(EOC)	1	2
采样时间(ACQ)	21+	4	序列转换完成后。设置标志位时间(EOS)	1	1
转换时间(CONV)	10	20			

注：+ 当 CPS=1 时，启动时序根据软件设置的触发方式使用相应的CLKOUT 时钟周期同步ADC的时钟周期(ADCCLK)

++ 采样时间与ACQ PSn上的位有关。采样时间在ACQ PS=1、2和3的值列于表10. 15. 从表中可推断其他ACQ PSn 值所对应的采样时间(ACQ)

表10. 15 ACQPS=1, 2, 3 时 ACQ 的值

ACQPS	CPS=0	CPS=1	ACQPS	CPS=0	CPS=1
1	ACQ=4	ACQ=8	3	ACQ=8	ACQ=16
2	ACQ=6	ACQ=12			

10.8 ADC 转换应用举例

本节介绍一个 A/D 采样的例程，该程序用事件管理器 B 的定时器4定时时间到来触发 A/D 采样的启动。采样时采用级连模式，一次作16个转换，转换通道分别是0~15。转换完成后，在 A/D 中断服务子程序中将转换结果读出。该程序作一次 A/D 采样。

1. 汇编源程序

```
.include          "F2407REGS.H"                ;引用头部文件
.def              c_int0
```

;建立中断向量表

```
.sect ".vectors"
RSVECT B c_int0
INT1 B GISR1
INT2 B PHANTOM
INT3 B PHANTOM
INT4 B PHANTOM
INT5 B PHANTOM
INT6 B PHANTOM
RESERVED B PHANTOM
SW_INT8 B PHANTOM
:
SW_INT31B PHANTOM
```

;中断子向量入口定义pvecs

```
.sect ".pvecs"
PVECTORS B PHANTOM
B PHANTOM
B PHANTOM
B PHANTOM
ADCINT_ISR
PHANTOM
:
B PHANTOM
```

;主程序：

```
.text
c_int0
SETC INTM
CLRC SXM
CLRC OVM
CLRC CNF
LDP #OE0H
SPLK #81FEH,SCSR1
SPLK #OE8h,WDCR
LDP #0
SPLK #0001h,IMR
SPLK #OFFFh,IFR
CALL ADINIT
```

;定义主向量段

```
;PM 0 复位向量 1
;PM 2 中断优先级1 4
;PM 4 中断优先级2 5
;PM 6 中断优先级3 6
;PM 8 中断优先级4 7
;PMA 中断优先级5 8
;PMC 中断优先级6 9
;PME 模拟量输入中断(保留) 10
;PM 10 用户定义软件中断
:
;PM 3E 用户定义软件中断
```

;定义子向量段

```
;保留向量地址偏移量 0000h
;保留向量地址偏移量 0001h
;保留向量地址偏移量 0002h
;保留向量地址偏移量 0003h
;保留向量地址偏移量 0004h ADC中 断
;保留向量地址偏移量 0005h
:
;保留向量地址偏移量 0041h
```

```
;CLKIN=6 MHz,CLKOUT=24 MHz
;关看门狗
;允许 INT1中 断
;清所有中断标志
;初始化 ADC程序
```

```

        CLRC    INTM                ;开总中断
        CALL    AD                  ;启动A/D 转换程序
WAIT:    NOP
        B       WAIT
;ADC 初始化程序
ADINIT:LDP    #225                  ;设置通用定时器4
        SPLK    #0000H,T4CNT
        SPLK    #170CH, T4CON
        SPT, K   廿075H, T4PFR
        SPLK    #0400H,GPTCONB
        SPLK    #0FFFFH,EVBIFRB
        SPLK    #0000H, EVBIMRB
        LDP     #DP_PF2
        SPLK    #0010H, ADCTRL1      :设置ADC控制寄存器
        SPLK    #8404H,ADCTRL2
        SPLK    #000FH.MAXCONV       ;16通道
        SPLK    #3210H,CHSEISEQ1
        SPLK    #7654H, CHSELSEQ2
        SPLK    #0BA98H. CHSELSEQ3
        SPLK    #0FEDCH.CHSELSEQ4
        LDP     #DP_SARAM2           :指向0A00h~0A80h
        SPLK    #ADRESULT,ADCOUNT
        RET
;启动A/D 转换程序
AD:      LDP     #225
        LACL    T4CON
        OR      #40H                ;启动定时器4
        SACL    T4CON
        RET
;中断程序
GISR1:                                :优先级 INTI 中断子程序入口
        ;保护现场
        LDP     #OE0H
        LACC    PIVR,1              ;读取外设中断向量寄存器(PIVR),并左移一位
        ADD     #PVECTORS           ;加上外设中断入口地址
        BACC                                ;跳到相应的中断服务子程序
ADCINT  ISR:
```




```
CLRC    SXM

LDP      #DP_SARAM2
LAR      AR1,ADCOUNT
LAR      ARO,#15
LAR      AR2,#RESULTO
MAR      *,AR2
ADC1:    LACC    *+,10,AR1
        SACH    *
        ADRK    #32
        MAR      *,ARO
        BANZ    ADC1,*    — , AR2
        LDP      #DP_PF2
        LACL     ADCTRL2
        OR       #4000H           ;复位 SEQ1
        AND      #OFFDFH         ;清 INT FLAG SEQ1
        SACL     ADCTRL2
        LDP      #DP_SARAM2
        LACL     ADCOUNT
        SUB      #ADRESULT+31
        BCND     ADC2,GEQ
        ADD      #ADRESULT+32    ;<32
        SACL     ADCOUNT
        B        GISR1_RET
ADC2:    SPLK     #ADRESULT,ADCOUNT ;=32
        LACL     T4CON
        AND      #0FFBFH         ;停止定时器4,即停止A/D转换
        SACL     T4CON
        B        GISR1_RET
GISR1_RET:           ;中断返回出口
                ;恢复现场
        CLRC     INTM           ;开总中断,因为一进中断就自动关闭了总中断
        RET
;假中断程序
PHANTOM
        KICK_DOG           ;复位看门狗
        RET
        END
```

2.C 语言源程序

```
//所需的复位和中断向量定义文件vectors.asm
//该文件利用汇编语言代码定义了复位和中断向量
.title      "vectors.asm"
.ref        c_int0,nothing,adint
.sect       ".vectors"

reset:      b      c      int0      //复位向量
int1:       b      adint      //A/D 中断向量
int2:       b      nothing
int3:       b      nothing
int4:       b      nothing
int5:       b      nothing
int6:       b      nothing

//主程序
//该程序用于进行A/D转换的演示，A/D转换结果存于数足ADRESULT[16]中
//寄存器 cesi 用于测试每个A/D转换结果
#include     "register.h"

int  ADRESULT[16];      //定义一个数组用于保存 A/D 转换结果
volatile unsigned int  *j;    //定义一个指针变量
int  i=0x00,cesi;
//屏蔽中断子程序
void inline  disable()
《
    asm("setc INTM");
}

//开总中断子程序
void inline  enable()

    asm("clrc INTM");

//系统初始化子程序
void  initial()

    asm("setc  SXM");      V/符号位扩展有效
    asm("clrc  OVM");      /累加器中结果正常溢出
    asm("clrc  CNE");      //B0 被配置为数据存储空间
    SCSRI =0x81FE;         /CLKIN=6   MHz.CLKOUT=4CLKIN=24   MHz
    WDCR=0x0E8;           //不使能看门狗，用软件禁止看门狗
```

```

    IMR=0x0001;           //允许 INT1 中断
    IFR=0x0FFFF;         //清除全部中断标志，写1清0
}
//A/D 初始化子程序
void ADINIT()
{
    T4CNT=0x0000;         //T4 计数器清0
    T4CON =0x170C;        //T4 为连续增计数模式，128分频，且选用内部时钟源
    T4PR=0x75;            //设置 T4 的周期寄存器
    GPTCONB=0x400;        //T4 周期中断标志触发A/D转换
    EVBIFRB=0x0FFFF;     //清除 EVB中断标志，写"1"清0
    ADCTRL1=0x10;         //采样时间窗口预定标位 ACQPSO为0，
                          //转换时间预定标位CPS为0，A/D为启动方式，排
                          //序器为级连工作方式，且禁止特殊的缓冲工作方式
    ADCTRL2=0x8404;       //可以用EVB的一个事件信号触发A/D转换，
                          //且用中断模式1
    MAXCONV=0x0F;         //16通道
    CHSELSEQ1=0x3210;
    CHSELSEQ2=0x7654;
    CHSELSEQ3=0x0BA98;
    CHSELSEQ4=0x0FEDC;    //转换通道是0~15
}
//启动A/D转换子程序(通过启动定时器4的方式间接启动)
void ADSOC()

    T4CON=T4CON|0x40;     //启动定时器4

//若是其他中断，则直接返回子程序
void interrupt nothing()

    return;
}
//A/D 中断服务子程序
void interrupt adint()
{
    asm("clrc          SXM"); //抑制符号位扩展
    j=RESULTO;               //取得 RESULTO 的地址
    for(i=0;i<=15;i++,j++)

```



```

        ADRESULT[]=j>>6;           //把A/D转换的结果左移6位后存入规定的数组
        cesi=ADRESULT[i];          //检验每个 A/D 转换的结果
    }

    ADCTRL2=ADCTRL2|0x4200;         //复位 SEQ1,且清除 INT FLAG SEQ1标志 写 1 清 0
    enable();                       Y/开总中断, 因为一进入中断总中断就自动关闭
}

main()

    disable();                     //禁止总中断
    initial();                     //系统初始化
    ADINIT();                      //A/D 初始化子程序
    enable();                      /开总中断
    ADSOC();                      启动A/D转换
    while(1)

        if(i==0x10)    break;

                                //如果已发生中断, 则停止等待(发生中断后, i=0x10)
                                //等待中断发生
        T4CON=T4CON&0x0FFBF;      //停止定时器4, 即间接停止A/D转换
    while(1)

        ;

    } //死循环. 在实际的工程应用中, 可以将A/D 转换的结果用于一些运算中
}

```

在用本书介绍的实验开发板调试此程序时, 可能出现调试不能通过的情况。调试时只需将 A/D 的中断优先级由高优先级改为低优先级, 即在复位和中断向量定义文件 `vectors.asm` 中将 A/D 的 INT1 中断改为 INT6 中断, 还要在主程序中将 IMR 修改为允许 INT6 中断以及修改 ADCTRL1 寄存器设置 A/D 为低优先级中断。因为在试验时, 实验开发板的功率保护中断引脚可能会一直使中断信号产生 (功率保护电路未接, 见图 17.25), 导致功率保护中断 PDPINTB 一直占用中断, 而不能让其他中断得到响应时间, 它在中断类 INT1 中与高优先级的 A/D 中断发生冲突, 使高优先级的 A/D 中断失效。

11.4

11.1

SPI 模块的特性包括:

- SPISOMI:SPI 从动输出/主动输入引脚；
- SPISIMO:SPI 从动输入/主动输出引脚；
- SPISTE:SPI 从动发送使能引脚；
- SPICLK:SPI 串行时钟引脚。

›两种工作方式：主动或从动工作方式。

＞数据字长：1~16个数据位。

一无延时的下降沿：串行外设接口在SPICLK 信号下降沿发送数据，而在 SPICLK 信号上升沿接收数据。

—有延时的下降沿：串行外设接口在 **SPICLK** 信号下降沿之前的半个周期时发送数据，而在**SPICLK** 信号下降沿接收数据。

一无延时的上升沿：串行外设接口在 **SPICLK** 信号上升沿发送数据，而在 **SPICLK** 信号下降沿接收数据。

一有延时的上升沿：串行外设接口在SPICLK 信号上升沿之前的半个周期时发送数据，而在SPICLK 信号上升沿接收数据。

> 同时接收和发送操作(发送功能可用软件禁止)。

发送和接收操作可通过中断或查询方法来完成。

> 9 个SPI 模块控制寄存器地址位于7040h~704Fh 之间。

注意：在SPI 模块中，有的控制寄存器为8位，但它又与16位的外设总线相连。当访问这些寄存器时，寄存器的数据在低字节(0~7位)，高字节(8~15位)读作0. 对高字节的写操作无效。

串行外设接口结构框图如图11. 1所示，图中给出的是DSP 为从动方式下的结构。

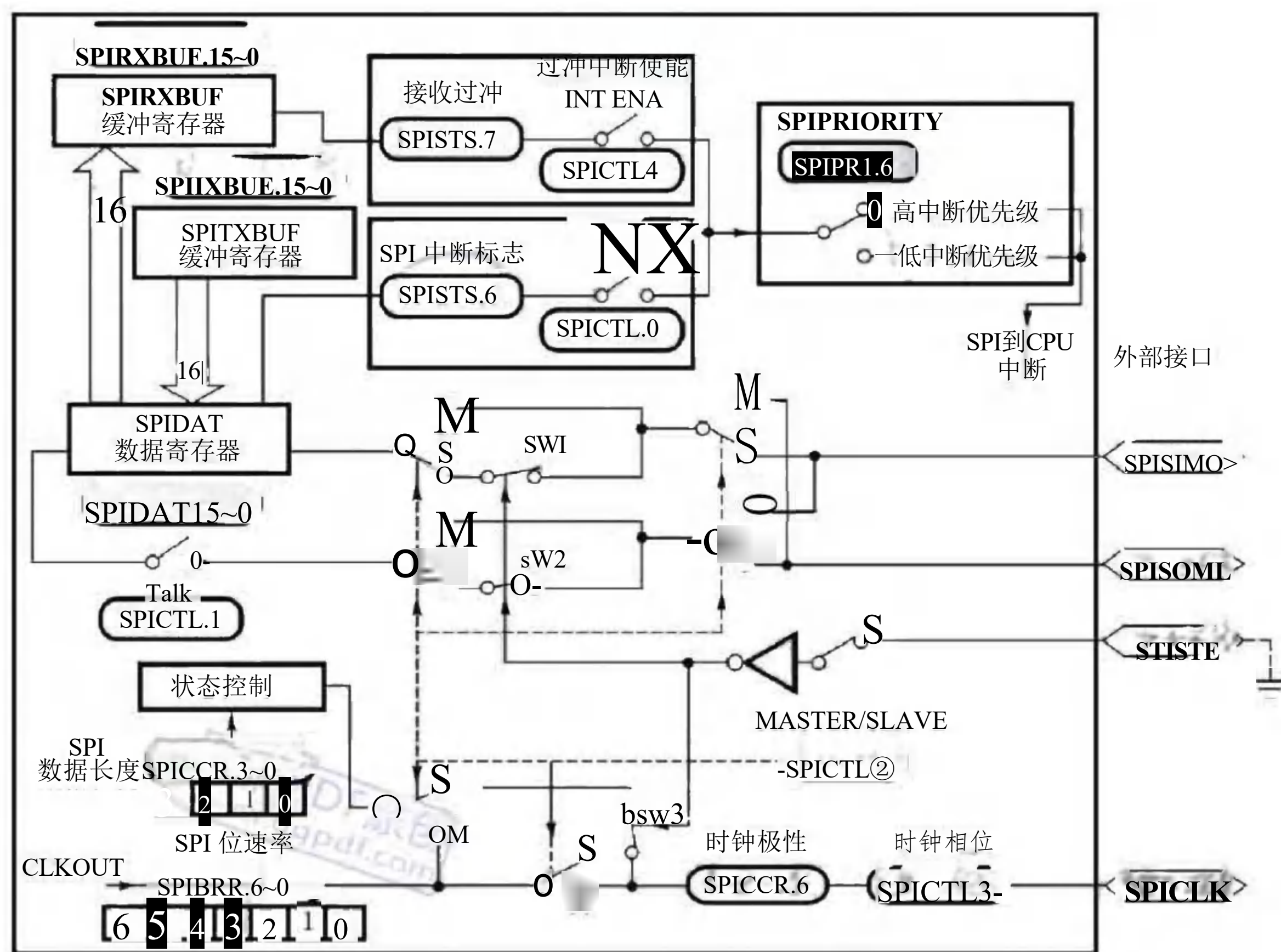


图11. 1 串行外设接口结构框图

11.2 串行外设接口操作

本节将介绍串行外设接口的操作，其中包括：操作方式、中断、数据格式、时钟源和初始化，并给出典型数据传送的时序图。

11.2.1 操作介绍

图11.2表明用于通信的串行外设接口与两个控制器(主控制器和从控制器)之间的连接。

主控制器通过输出SPICLK 信号来启动数据传送。对于主控制器和从控制器，数据都是在SPICLK 的一个边沿移出移位寄存器，并在相对的另一个边沿锁存到移位寄存器。如果位CLOCK PHASE(SPICTL.3)为1,则在SPICLK 跳变之前的半个周期时数据被发送和接收。因此，两个控制器可同时发送和接收数据。应用软件可决定数据的真伪。发送数据的方法有以下3种可能：

- 主控制器发送数据，从控制器发送伪数据；
- > 主控制器发送数据，从控制器发送数据；
- > 主控制器发送伪数据，从控制器发送数据。

主控制器可在任一时刻启动数据发送，因为它控制着 SPICLK 信号。但是软件决定了主控制器如何检测从控制器何时准备发送数据，以启动 SPI 传送数据。

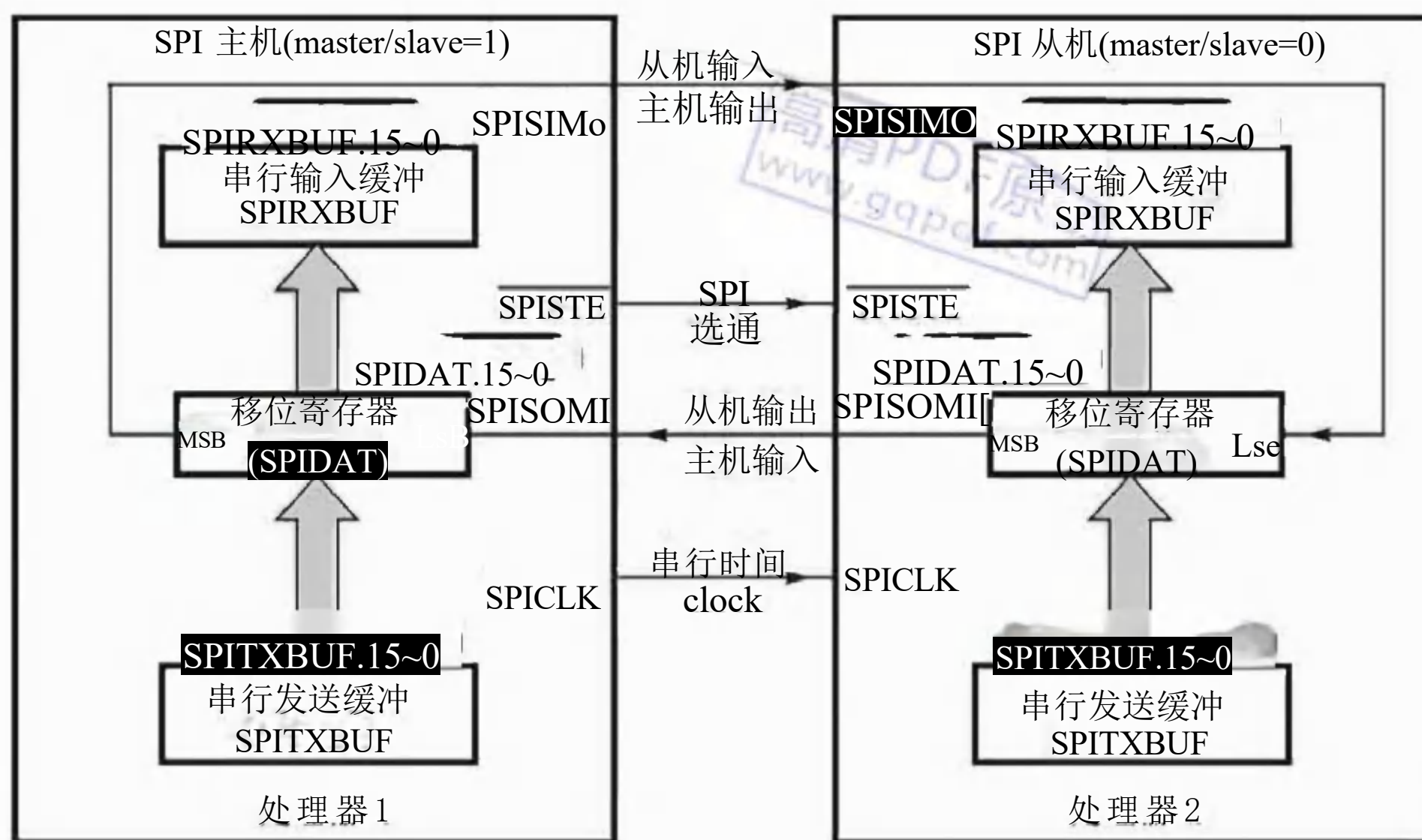


图11.2 串行外设接口主控制器/从控制器的连接

11.2.2 串行外设接口模块的主动和从动方式

串行外设接口可工作于主动方式或从动方式。用MASTER/SLAVE(SPICTL.2) 位来选择工作方式和 SPICLK 信号的来源。

1. 主动方式

在主动方式下(MASTER/SLAVE=1), 串行外设接口在 SPICLK 引脚上提供整个串行通信网络的串行时钟。数据从SPISIMO 引脚输出, 并在SPISOMI 引脚输入。

SPI 波特率设置寄存器(SPIBRR) 决定着网络发送和接收的位传输率, SPI 可选择126种不同的数据传输率。

写入到SPIDAT 或 SPITXBUF 寄存器的数据启动了SPISIMO 引脚上的数据发送. 先发送最高位。同时, 接收到的数据通过 SPISOMI 引脚移入 SPIDAT 的最低位。当选定数量的位发送完时, 则整个数据发送完毕。首先将接收到的数据传送到SPIRXBUF 寄存器, 并进行右对齐供 CPU 读取。

当指定数量的数据位已经通过 SPIDAT 移位后, 则将发生下列事件:

- >SPIDAT 中的内容传送到SPIRXBUF 寄存器中;
- >SPI INT FLAG 位置位;
- > 如果SPITXBUF 寄存器中还有有效数据, 或TXBUF FULL标志位为1, 则此数据将传送到SPIDAT 寄存器中, 否则将已接收到的数据移出SPIDAT 寄存器后, SPICLK 时钟即停止;
- 如果SPI INT ENA 使能, 则将产生中断。

在典型应用中, SPISTE 引脚作为SPI 从控制器的片选信号引脚(在接收主控制器的数据前把SPISTE 引脚置低, 在接收完主控制器的数据后再把 SPISTE 引脚置高)。

2. 从动方式

在从动方式下(MASTER/SLAVE=0), 数据从 SPISOMI 引脚输出并且由 SPISIMO 引脚输入。SPICLK 引脚作为串行移位时钟的输入, 该时钟由SPI 网络主控制器提供。传输率由该时钟决定, SPICLK 的输入频率应不超过器件系统时钟的1/4。

当网络主控制器的SPICLK 信号为合适的边沿时, 写入到 SPIDAT 或 SPITXBUF 寄存器中的数据被传送到网络。当SPIDAT 寄存器中的所有位被移出后, SPITXBUF 中的数据将传送到SPIDAT 寄存器中。如果当前 SPITXBUF 寄存器中的数据没有往网络发送, 则其数据立即传送到SPIDAT 寄存器中。为了接收数据, 串行外设接口等待网络主控制器送出的 SPICLK 信号, 然后再将SPISIMO 引脚上的数据移入到 SPIDAT 寄存器。如果从控制器同时也发送数据, 则必须在SPICLK 信号开始之前把数据写入到SPIRXBUF 或 SPIDAT 寄存器中。

当 TALK(SPICTL.1) 位清0时, 数据传送被禁止, 从控制器输出引脚 SPI SMD 被置成高阻态。在数据发送期间, TALK 位清0, 虽然SPISOMI 引脚被强制置成高阻态, 但是正在发送的数据将完全发送完毕。这就使得同一个 SPI 网络上有多个从器件, 但是在任一时刻只能有一个从器件起作用。

当 SPISTE 引脚用作从控制器片选引脚时, 引脚 SPISTE 上的低有效信号使得从串行外设接口将数据传送到串行数据线。而高无效信号则使得串行外设接口的串行移位寄存器终止, 并且其串行输出引脚被置成高阻态。这使得同一网络上可以有多个从器件, 但是同一时刻只能有一个从器件起作用。

11.2.3 串行外设接口中断

有5个控制位与串行外设接口中断相关:

- >SPI 中断使能位 SPI INT ENA(SPICTL.0);
- >SPI 中断标志位 SPI INT FLAG(SPISTS.6);
- >SPI 过冲中断使能位 OVERRUN INT ENA(SPICTL.4);
- >SPI 过冲中断标志位 RECEIVE OVERRUN INT FLAG(SPISTS.7);
- >SPI 中断优先级选择位 SPI PRIORITY(SPIPRI.6)。

在 SPI 中断使能的情况下, 当数据被移入或移出 SPIDAT 寄存器时, 中断标志位被置位, 并产生中断。中断标志位保持置位, 直到以下情况之一发生时才被清除:

- > 中断被响应;
- >CPU 读取SPIRXBUF 寄存器(注意, 读 SPIRXEMU 并不清除中断标志位);
- > 软件清除 SPI SW RESET 位;
- 系统复位。

当 SPI 中断发生时, 接收到的数据已存放在 SPIRXBUF 寄存器中并等待 CPU 读取。如果 CPU 在下一个字符已经接收完毕时, 还没有读取 SPIRXBUF 寄存器中的数据, 则新数据被写入到SPIRXBUF 寄存器, 覆盖旧数据, 并且将置位 SPI 过冲中断标志位。

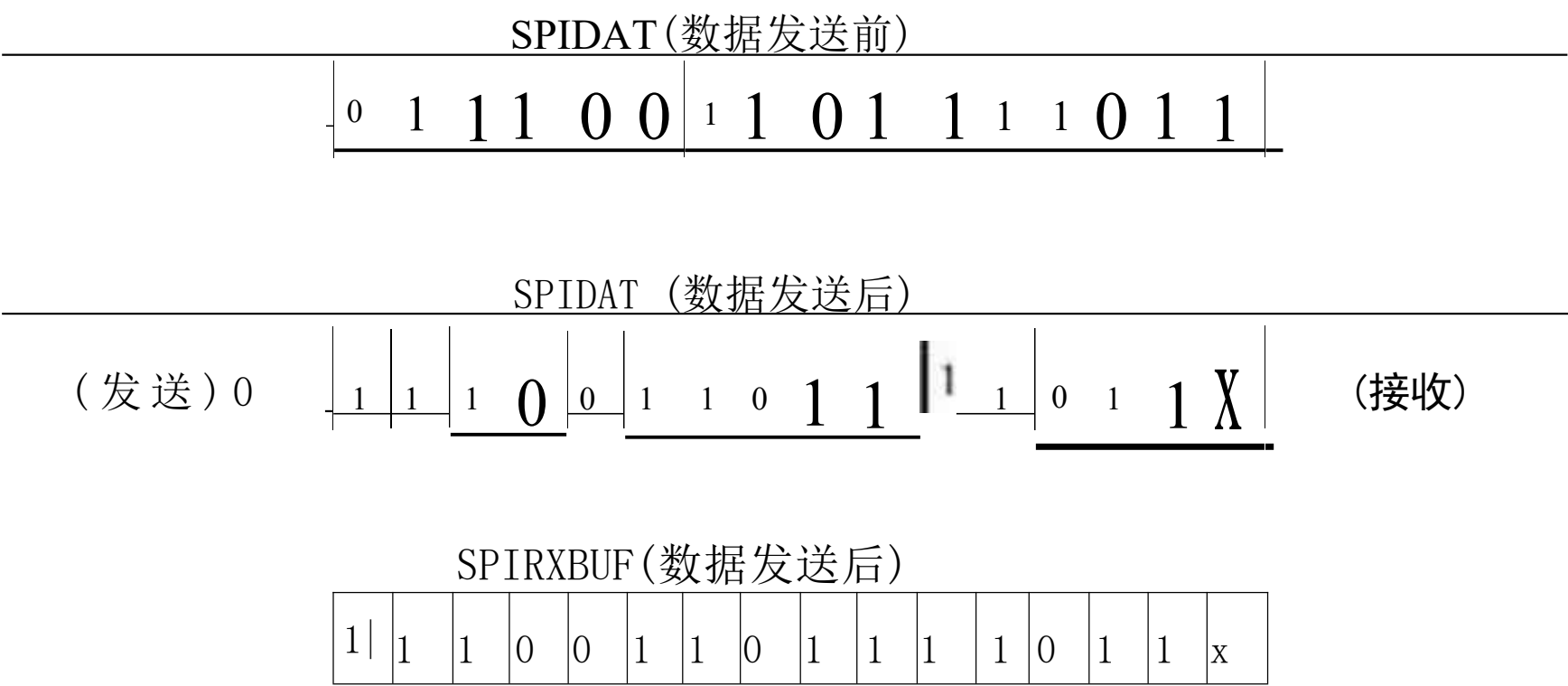
11.2.4 数据格式

SPICCR.3~0 这4位确定了数据的位数(1~16位), 该信息指导状态控制逻辑计算接收和发送的位数, 从而决定何时处理完一个数据, 下列情况适用于少于16位的数据:

- >当数据写入到 SPIDAT 或 SPITXBUF 寄存器时必须左对齐;
- > 数据从 SPIRXBUF 寄存器读取时必须右对齐;
- >SPIRXBUF 中存放最新接收到的数据位(右对齐), 再加上那些已移位到左边的前次留下的位。

如果发送字符的长度为1, 且 SPIDAT 当前值为737Bh, 那么在主动方式下, 将 SPIDAT

和 SPIRXBUF 寄存器在数据发送前和发送后的数据格式表示如下：



注意：如果SPISOMI引脚上的电平为高，则X=1；如果SPISOM1引脚上的电平为低，则X=D.

11.2.5 串行外设接口波特率设置和时钟方式

串行外设接口支持125种不同的波特率和4种不同的时钟方式。根据串行外设接口处于从动工作方式还是主动工作方式. 决定引脚 SPICLK 是接收一个外部的 SPI 时钟信号还是由DSP 提供SPI 时钟信号。

- >在从动工作方式中，DSP串行外设接口时钟由SPI 外部的SPI 时钟信号提供，并且该时钟信号的频率不能大于 CPU 时钟的1/4；
- 在主动工作方式中，串行外设接口时钟由DSP 串行外设接口产生并由 SPICLK 引脚输出。

1. 波特率的计算

>SPIBRR=3~127 时：

SPI 波特率=SYSCLK/(SPIBRR+1)
SPIBRR=(SYSCLK/SPI 波特率)-1

SPIBRR=0,1或2时：

SPI 波特率=SYSCLK/4

其中，SYSCLK 为系统时钟频率；SPIBRR 为主串行外设接口器件中的SPIBRR 寄存器中的内容。

【例11.1】 已知系统时钟频率SYSCLK=24 MHz,现要将SPI 的波特率设置为3 Mb/s, 试确定SPI 波特率寄存器的值即 SPIBRR。

根据上面计算 SPIBRR 的公式可知：

SPIBRR=24×10⁶/(3×10⁶)-1=8-1=7h

所以 SPI 波特率寄存器设置为 SPIBRR=7h。

2. 串行外设接口的时钟方式

时钟极性位 CLOCK POLARITY(SPICCR.6)和时钟相位位 CLOCK PHASE(SPICTL.3)

3) 控制着引脚 SPICLK 上的4种不同的时钟方式。时钟极性位选择时钟有效沿为上升沿还是下降沿；时钟相位位则选择时钟的1/2周期延时。4种不同的时钟方式如下：

- 无延时的下降沿：串行外设接口在 SPICLK 信号下降沿发送数据，而在 SPICLK 信号上升沿接收数据；
- 有延时的下降沿：串行外设接口在 SPICLK 信号下降沿之前的半个周期时发送数据，而在 SPICLK 信号下降沿接收数据；
- 无延时的上升沿：串行外设接口在 SPI-CLK 信号上升沿发送数据，而在 SPICLK 信号下降沿接收数据；
- 有延时的上升沿：串行外设接口在 SPI-CLK 信号上升沿之前的半个周期时发送数据，而在 SPICLK 信号上升沿接收数据。

串行外设接口时钟方式选择如表11.1所列，这4种时钟方式与图11.3中的发送和接收是一一对应的。

表11.1 串行外设接口时钟方式选择

SPICLK方式	时钟极性 (SPICCR.6)	时钟相位 (SPICTL.3)
无延时上升沿	0	0
有延时上升沿	0	1
无延时下降沿	1	0
有延时下降沿	1	1

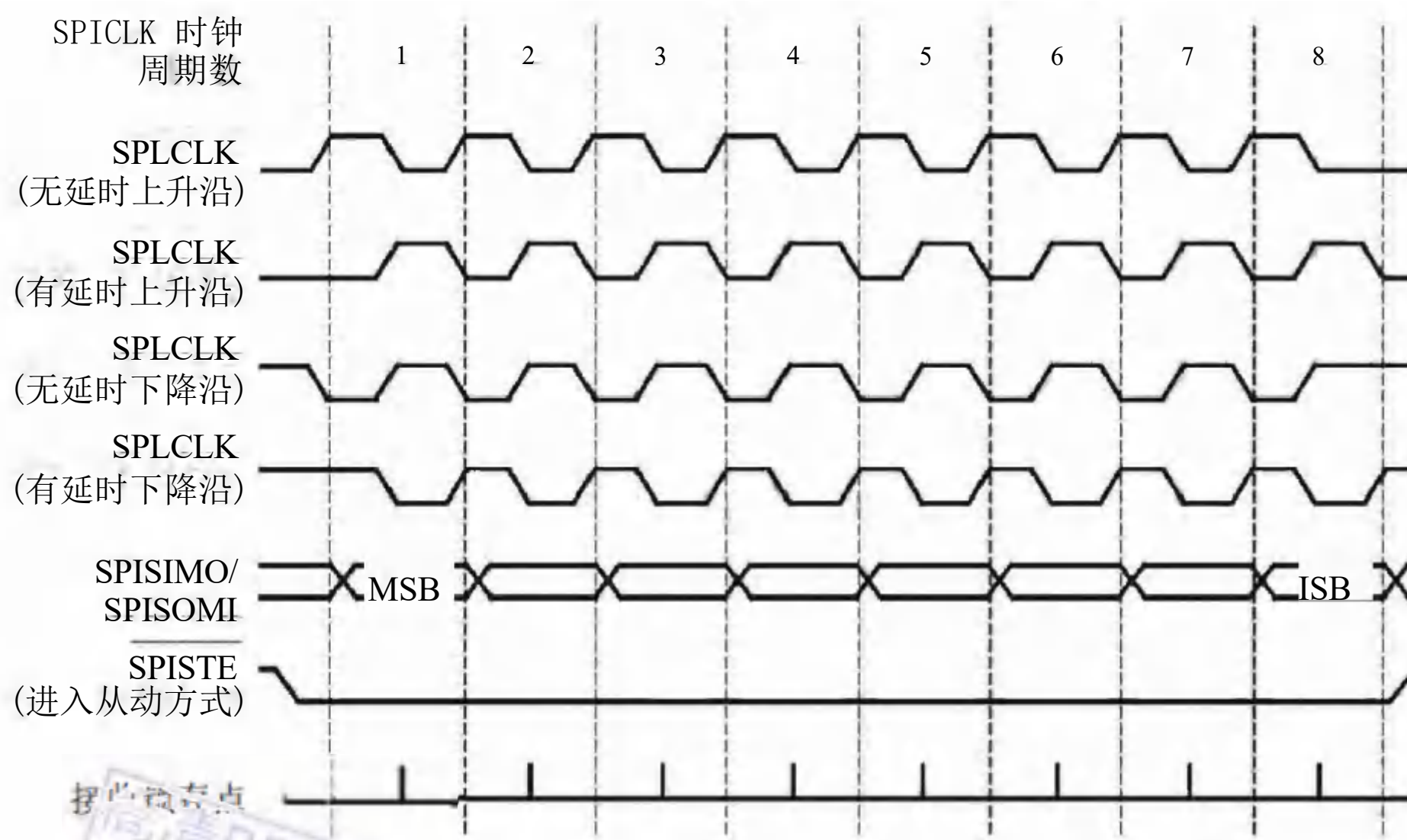


图 11.3 串行外设接口时钟方式时序图

对于串行外设接口，仅当 $(SPIBRR+1)$ 的结果为偶数时，才保持 SPICLK 的对称性。当 $(SPIBRR+1)$ 为奇数并且 SPIBRR 大于 3 时，SPICLK 变成非对称。当CLOCK POLARITY 位置 0 时，SPICLK 的低脉冲比它的高脉冲长一个系统时钟；当CLOCK POLARITY 位置 1 时，SPICLK 的高脉冲比它的低脉冲长一个系统时钟。

图 11.4 为当 $(SPIBRR+1)$ 为奇数， $SPIBRR > 3$ 且 CLOCK POLARITY=1 时的 SPI-CLK 引脚的输出特性。

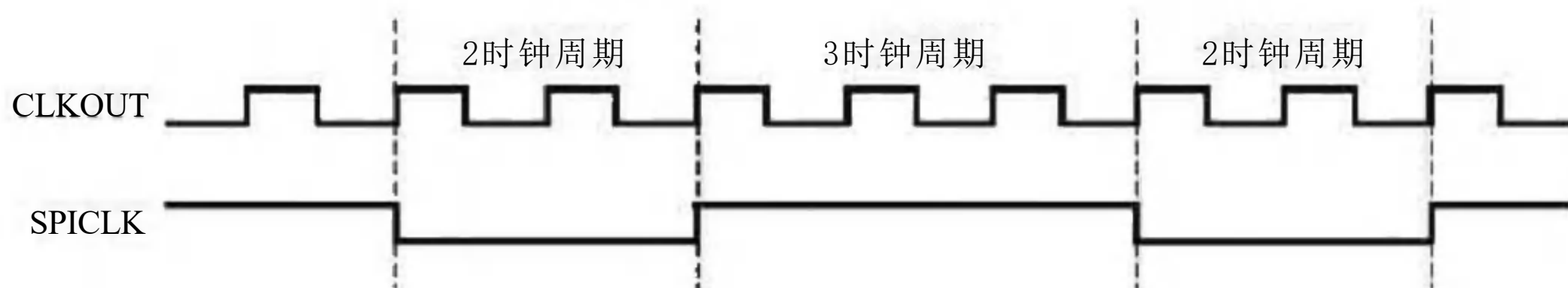


图11.4 串行外设接口SPICLK 引脚输出特性

11.2.6 串行外设接口的初始化

当系统复位时迫使SPI 外设模块进入下列缺省的配置：

- > 该单元被配置成从动模式(MASTER/SLAVE=0);
- 禁止发送功能(TALK=0);
- > 在 SPICLK 信号的下降沿输入数据被锁存;
- 字符长度为1位;
- > 禁止串行外设接口中断;
- > SPIDAT 寄存器中的数据被复位为0000h;
- > SPI 的4个引脚被配置成一般的I/O 功能。

为了改变串行外设接口在系统复位后的配置，应进行如下操作：

- 将 SPI SW RESET 位置1, 强制 SPI 进入复位状态;
- > 初始化串行外设接口的配置、数据格式、波特率和所需引脚的功能;
- > 将SPI SW RESET 位置1, 使 SPI 进入工作状态;
- > 写数据到SPIDAT 或 SPITXBUF 寄存器中(这就启动了主方式下的通信过程);
- > 在数据传送完成后，即SPISTS.6=1 时，读取SPIRXBUF 寄存器中的数据。

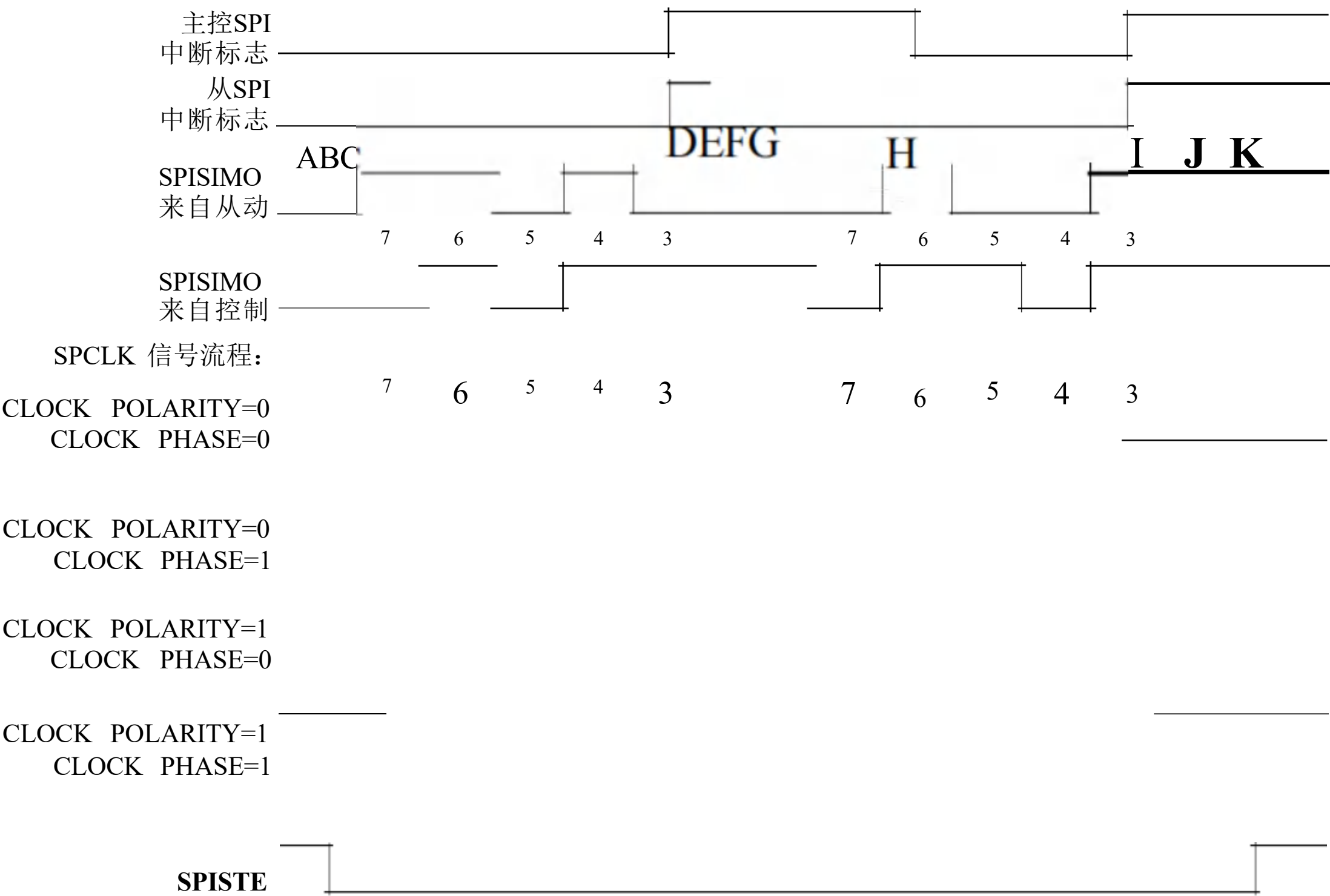
11.2.7 数据传送示例

SPI 数据传送的时序如图11.5所示，图中表明在使用对称的SPICLK 信号时，两个 SPI 器件之间进行5位字符的 SPI 数据传送。

使用非对称的 SPICLK 时序具有与图11.5类似的性质，但有一点除外：在低脉冲期间

(CLOCK POLARITY=0)或高脉冲期间(CLOCK POLARITY=1),采用非对称 SPICLK 的数据传送每一位时,要延长一个系统时钟周期。

图11.5给出的 SPI 数据传送的时序图只适用于串行位流最大为8位的 DSP 器件,对于具有串行位流最大为16位的 F240x 系列不适用,但也可从图中了解到 SPI 数据传送的基本情况。



- 注：A 从控制器将0D0h写入到SPIDAT,并等待主控制器移出数据。
- B 主控制器将从控制器的SPISTE 引脚的电平置低(有效)。
- C 主控制器将058h 写入到SPIDAT 来启动传送过程。
- D 第一个字节传送完成,设置中断标志。
- E 从控制器从它的SPIRXBUF 寄存器中读取数据。
- F 从控制器将04Ch 写入到SPIDAT 中,并等待主控制器移出数据。
- G 主控制器将06Ch 写入到SPIDAT 中来启动传送过程。
- H 主控制器从SPIRXBUF 寄存器中读取01Ah。
- I 第二个字节传送完成,设置中断标志。
- J 主、从控制器分别从相应的SPIRXBUF 寄存器中读取89h 和 8Dh。在用户程序屏蔽掉了未使用的位之后,主、从控制器分别接收到09h 和 ODh。
- K 主控制器将从控制器的 SPISTE 引脚的电平置高(无效)。

图11.5 SPI 数据传送时序图

11.3 串行外设接口控制寄存器

串行外设接口总共有9个控制寄存器地址位于7040h~704Fh. 如表11 . 2所列。

表 11.2 串行外设接口控制寄存器列表

地址	寄存器名称	位								
		15~8	7	6	5	4	3	2	1	0
7040h	SPICCR	三	SPI SW RESET	CLOCK POLARITY	三		SPI CHAR3	SPI CHAR2	SPI CHAR1	SPI CHAR0
7041h	SPICTL	一				OVERRUN INT ENA	CLOCK PHASE	MASTER SLAVE	TALK	SPI INT ENA
7042h	SPISTS		RECEINER OVERRUN FLAG	SPI INT FLAG	TXBUF FUL ₁ FLAG					
7043h		二	—							
7044h	SPIBRR	—	—	SPI BIT RATE6	SPI BIT RATE5	SPI BIT RATE4	SPI BIT RATE3	SPI BIT RATE2	SPI BIT RATE1	SPI BII RATE0
7045h	—		—							
7046h	SPIRXEMU	ERXB 15~8	ERXB7	ERXB6	ERXB5	ERXB4	ERXB3	ERXB2	ERXB1	ERXB0
7047h	SPIRXBUF	RXB 15~8	RXB7	RXB6	RXB5	RXB4	RXB3	RXB2	RXB1	RXB0
7048h	SPITXBUF	TXB 15~8	TXB7	TXB6	TXB5	TXB4	TXB3	TXB2	TXB1	TXB0
7049h	SPIDAT	SDAT 15~8	SDAT7	SDAT6	SDAT5	SDAT4	SDAT3	SDAT2	SDAT1	SDAT0
704Ah	—		—							
704Bh			三							
704Ch										
704Dh		三	二							
704Eh			—							
704Fh	SPIPRI		二	SPI PRIORITY	SPI SUSP SOFT	SPI SUSP FREE				

非法的

1. 串行外设接口配置控制寄存器(SPICCR) —— 地 址 7040 h

7	6	5~4	3	2	1	0
SPI SW RESET	CLOCK POLARITY	保留位	SPI CHAR3	SPI CHAR2	SPI CHAR1	SPI CHAR0
RW_0	RW_0	R_0	RW_0	RW_0	RW_0	RW_0

- 位 7

SPI SW RESET。SPI 软件复位位。用户在改变配置前，应把该位清0, 并在恢复操作前把该位置1。

0 初始化串行外设接口操作标志位至复位条件；

1 串行外设接口准备发送或接收下一个字符。
- 位 6

CLOCK POLARITY。移位时钟极性位，该位控制 SPICLK 信号的极性。具体见表11. 1。

0 在SPICLK 信号的上升沿输出数据，在下降沿输入数据。当无数据发送时，SPICLK 保持低电平；

1 在SPICLK 信号的下降沿输出数据，在上升沿输入数据。当无数据发送时，SPICLK 保持高电平。
- 位5~4

保留位。
- 位3~0

SPI CHAR3~SPI CHAR0。数据长度选择位，具体见表11. 3。

表11.3 数据长度选择对应表

SPI CHAR3	SPI CHAR2	SPI CHAR1	SPI CHAR0	字符长度	SPI CHAR3	SPI CHAR2	SPI CHAR1	SPI CHAR0	字符长度
0	0	0	0	1	1	0	0	0	9
0	0	0	1	2	0	0	1	1	10
0	0	1	0	3	1	0	1	0	11
0	0	1	1	4	1	0	1	1	12
0	1	0	0	5	1	1	0	0	13
0	1	0	1	6	1	1	0	1	14
0	1	1	0	7	1	1	1	0	15
0	1	1	1	8	1	1	1	1	16

2. 串行外设接口操作控制寄存器(SPICTL)——地址7041 h

7~5	4	3	2	1	0
保留位	OVERRUN INT ENA	CLOCK PHASE	MASTER/SLAVE	TALK	SPI INT ENA
R_0	RW_0	RW_0	RW_0	RW_0	RW_0

- 位 7 ~ 5保留位。
- 位4OVERRUN INT ENA。过冲中断使能位。

0禁止过冲中断；

1使能过冲中断。
- 位3CLOCK PHASE。SPI时钟相位选择位，具体见表11. 1。

0正常的SPI 时钟方式；

1延迟半个周期的SPICLK 信号。
- 位 2MASTER/SLAVE。SPI 主从工作方式选择位。

0从工作方式；

1主工作方式。
- 位 1TALK。主/从工作方式下发送允许位。

0禁止发送：

●在从工作方式下. 如果以前没有被配置成一般的 I/O 功能. 则引脚 SPISOMI 将置成高阻态；

• 在主工作方式下，若以前没有被配置成一般的 I/O 功能，则引脚 SPISIMO 将置成高阻态。

1允许发送。
- 位 0SPI INT ENA。SPI中断使能位。

0禁止中断；

1使能中断。

3. 串行外设接口状态寄存器(SPISTS)——地 址7042 h

7	6	5	4~0
RECEIVER OVERRUN FLAG件	SPI INT FLAG性	TX BUF FULL FLAG	保留位
RC0	RC_D	RC_0	R_0

注：R= 可读；C= 清0：0=复位值。

+ RECEIVER OVERRUN FLAG接收过冲位和SPI INT FLAG中断标志位共用同一个中断矢量

王 向5、6和7位写入0不会影响这些位的值。

位 7 **RECEIVER OVERRUN FLAG**。SPI接收过冲标志位，该位为只读只清除标志位。在前一个数据从缓冲器中读出之前又完成了下一个数据的接收或发送操作，则SPI 硬件将设置该位。该位表明最后一个接收到的数据已经被覆盖写入，并因此而丢失。如果OVERRUN INT ENA 位已被置1, 则该位每次置位时，SPI 就发生一次中断请求。该位可由以下3种操作来清除：写1到该位；写0到 SPI SW RESET 位；系统复位。

OVERRUN INT ENA 位被置位，则 SPI 将在第一次 RECEIVER OVERRUN FLAG 置位时产生一次中断请求。如果在该标志位仍置位时又发生了接收过冲事件，则SPI 将不会再次申请中断请求。这就使得在每次发生接收过冲事件后，必须向SPI SW RESET 位写0来清除该标志位，使下一次发生接收过冲事件时，又能产生过冲中断请求。换句话说，如果 RECEIVER OVERRUN FLAG 标志位由中断服务子程序保留设置(未被清除)，则当中断服务子程序退出时，将不会立即产生另一个过冲中断。无论如何，在中断服务子程序期间应清除 RECEIVER OVERRUN FLAG 标志位。因为 RECEIVER OVERRUN FLAG 标志位和 SPI INT FLAG 标志位共用相同的中断向量。在接收下一个数据时这将减少关于中断源的任何可能疑问。

- 0 无中断请求；
- 1 有中断请求。

位 6 **SPI INT FLAG**。SPI 中断标志位。

- 0 无中断请求；
- 1 有中断请求。

位 5 **TX BUF FULL FLAG**。SPI 发送缓冲器满标志位。当向SPITXBUF 寄存器写入数据时，将置位该位。当 SPITXBUF 寄存器中的数据移入到 SPIDAT 寄存器中后，将自动清除该位。

- 0 发送缓冲器空；
- 1 发送缓冲器中有数据。

位4~0 保留位。

4. 串行外设接口波特率设置寄存器 (**SPIBR R**) —— 地 址 **7044 h**

7	5		4	3	2	1	0
保留位	SPI BIT RATE 6	SPI BIT RATE5	SPI BIT RATE 4	SPI BIT RATE 3	SPI BIT RATE 2	SPI BIT RATE1	SPI BIT RATE 0
R_0	RW_0	RW_0	RW_0	RW_0	RW_o	RW_o	RW_0

位 7 保留位。

位6~0 SPI BIT RATE6-SPI BIT RATE0。SPI 波特率设置位，具体计算参见 11.2.5小节。

5. 串行外设接口仿真接收缓冲寄存器(SPIRXEMU)—— 地 址7046 h

15	14	13	12	11	10	9	8
ERXB15	ERXB 14	ERXB 13	ERXB12	ERXB 11	ERXB 10	ERXB 9	ERXB8
R_0	R_0	R_0	R_0	R_0	R_0	R_0	R_0
7	6	5	4	3	2	1	0
ERXB7	ERXB6	ERXB5	ERXB4	ERXB3	ERXB 2	ERXB1	ERXB 0
R_0	R_0	R_0	R0	R_0	R_0	R_o	R_0

位15~0 ERXB 15~ERXB0。仿真缓冲器接收的数据。 SPIRXEMU 寄存器的功能与 SPIRXBUF 基本相同，只是读 SPIRXEMU 时，不会清除 SPI INT FLAG 标志位。一旦 SPIDAT 已经接收到完整的数据，该数据被传送到 SPIRXEMU 和SPIRXBUF 寄存器中，在这两个寄存器中的数据可被读取。与此同时，SPI INT FLAG 标志位置位。创建该镜像寄存器是为了支持仿真，读SPIRXBUF 时，将清除SPI INT FLAG 标志位。在仿真器的正常操作中，通常靠读取控制寄存器来不断地更新显示在屏幕上的寄存器内容，对于 SPI 来说，是采用在仿真器读取 SPIRXEMU 中的值来更新其显示在屏幕上的内容。 SPIRXEMU 寄存器允许仿真器更准确地模拟 SPI 的真实操作，因此建议在正常的仿真器工作方式下应读取 SPIRXEMU 寄存器中的值。

6. 串行外设接口接收缓冲寄存器(SPIRXBUF)——地址7047 h

15	14	13	12	11	10	9	8
RXB15	RXB14	RXB13	RXB 12	RXB 11	RXB10	RXB9	RXB8
R_0	R_0	R_0	R_0	R_0	R0	R°	R0
7	6	5	4	3	2	1	0
RXB7	RXB 6	RXB 5	RXB 4	RXB 3	RXB2	RXB1	RXB0
R_0	R_0	R_0	R_0	R_0	R_0	R_0	R_0

位15~0 RXB15~RXB0。 接收到的数据。 一旦 SPIDAT 已经接收到完整的数据，该数据就被传送到SPIRXBUF 寄存器，供CPU 读取，同时 SPI INT FLAG 标志位被置位。因为数据首先被移位到SPI 的最高有效位中，所以数据在该寄存器中采用右对齐方式存储。

串行外设接口发送缓冲寄存器(SPITXBUF)。 该寄存器存储下一个要发送的数据，当写数据到该寄存器中时，置位 TX BUF FULL 标志位。当正在发送的数据发送完毕时，该寄存器中的数据自动装入到SPIDAT 寄存器中，并清除发送缓冲器满 TX BUF FULL 标志位。如果当前发送没有被激活，则该寄存器中的数据将传送到 SPIDAT 寄存器中，且发送缓冲器满 TX BUF FULL 标志位不置位。

在主动工作模式下，如果当前发送没有被激活，则写入数据到该寄存器将启动发送，同时数据被传送到SPIDAT 寄存器中。

7. 串行外设接口发送缓冲寄存器 (SPITXBUF) —— 地址7048 h

15	14	13	12	11	10	9	8
TXB15	TXB 14	TXB13	TXB12	TXB 11	TXB 10	TXB9	TXB8
RW_0	RW_0	RW_0	RW_0	RW_0	RW_0	RW_0	RW_0
7	6	5	4	3	2	1	0
TXB7	TXB6	TXB5	TXB4	TXB3	TXB 2	TXB1	TXB0
RW_0	RW_0	RW_0	RW_0	RW_0	RW_0	RW_0	RW_0

位15~0 TXB 15~TXB0。发送的数据。

串行外设接口发送/接收寄存器(SPIDAT)， 当 SPIDAT 寄存器中的数据在连续个 SPI-CLK 周期中被移出去时(最高位)。其移出的 SPI 的每一位(最高位)的同时，将有一位移入到该寄存器的最低位。

8. 串行外设接口发送/接收缓冲寄存器 (SPIDAT) —— 地址7049 h

15	14	13	12	11	10	9	8
SDAT 15	SDAT 14	SDAT 13	SDAT 12	SDAT11	SDAT 10	SDAT 9	SDAT8
RW_0	RW_0	RW_0	RW_0	RW_0	RW0	RW_0	RW_0
7	6	5	4	3	2	1	0
SDAT 7	SDAT 6	SDAT 5	SDAT4	SDAT 3	SDAT 2	SDAT1	SDAT0
RW_0	RW_0	RW_0	RW_0	RW_0	RW_0	RW_0	RW_o

位15~0 SDAT 15~SDAT0。串行数据。写入 SPIDAT 的操作可执行两种功能：如果 TALK 位被置位，则该寄存器提供了将被输出到串行输出引脚的数据。当 SPI 处于主动工作模式时，数据开始发送。在主动工作模式下，将伪数据写入到 SPIDAT 用以启动接收器的排序功能，因为硬件不支持少于16位的数据进行对齐处理，所以发送的数据必须先进行左对齐，而接收的数据则用右对齐格式读取。

串行外设中断优先级控制寄存器(SPIPRI)。 该寄存器用于选择 SPI 中断优先级的级别，可选择INT1 级 或INT5 级中断。并确定当仿真器挂起时的 SPI 操 作 。

9. 串行外设接口中断优先级控制寄存器(SPIPRI)——地 址 7 0 4Fh

7	6	5	4	3
保留位	SPI PRIORITY	SPI SUSP SOFT	SPI SUSP FREE	保留位
R_a	RW	RW	RW_o	R_o
位 7	保留位。			
位 6	SPI PRIORITY。SPI中断优先级选择位。 0 高优先级中断请求； 1 低优先级中断请求。			
位5~4	SPI SUSP SOFT,SPI SUSP FREE。SPI中断挂起时的操作控制位。 00 一旦仿真挂起，就立即停止； 10 一旦仿真挂起，就在当前的接收或发送完成后停止； 01 SPI 操作与仿真挂起无关； 11 SPI 操作与仿真挂起无关。			
位3~0	保留位。			

11.4 串行外设接口应用举例1

下面的设计是SPI 串 行D/A 转换芯片MAX5741 与 F2407 的接口及应用。由于SPI 系
统总线只需3~4位数据线和控制线，即可扩展具有 SPI 接口的各种器件，而并行总线扩展方
法需8根数据线、8~16位地址线、1~2位控制线；又由于如今 SPI 器件种类多，有SPI 语音芯
片、LED 驱动芯片、A/D 转换芯片、D/A 转换芯片和 EEPROM 芯片等，给设计带来了更多的
选择。因此在实时性要求不高的系统，使用SPI 总线可简化电路设计，省掉很多常规电路中的
接口器件，提高设计的可靠性。

11.4.1 硬件电路的设计

1.MAX5741 芯片功能介绍

MAX5741 是4路、10位、低功耗、电压缓冲输出的 D/A 转换器(DAC)。 高达 2 0MHz 的
时钟频率，3线串行接口兼容于SPI、QSPI、MICROWIRE 和 DSP，可大大节省电路板空间，降
低光电或变压器隔离应用的复杂程度。宽达+2.7~+5.5 V 的电源电压范围，消耗229 μ A
电源电流，适于低功耗和低电压应用。 DAC 输出使用片上精密输出放大器满摆幅输出。

MAX5741 基准输入电压范围为0~VDD。

MAX5741 片上上电复位(POR) 电路设定DAC 输出为0, 并以对地100 kQ 的电阻加载输出。这为驱动电磁阀或变送器等在上电时提供了额外的安全性。MAX5741 具有软件控制的关断模式, 将电源电流降低至0.1 μA 以下, 提供软件可选的关断输出负载(1 k Ω 、100 kQ 或高阻抗), 进一步减小系统的总功耗。MAX5741 满足汽车级温度范围(-40~+125℃),

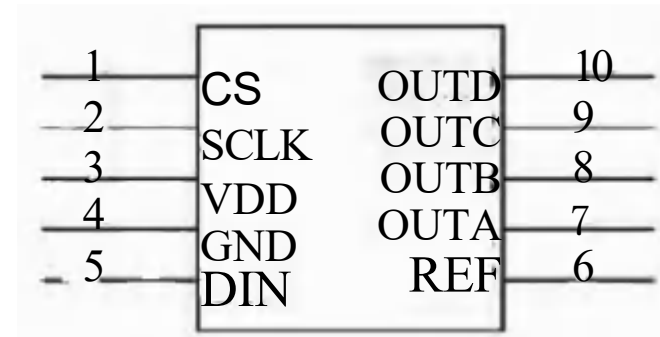


图11.6 MAX5741 引脚示意图

提供10引脚 μMAX 封装。芯片的引脚分布如图11.6所示, 各引脚的名称和功能如表11.4所列。

表11.4 MAX5741 的引脚名称和引脚功能

引脚号	名称	功能
1	CS	芯片选择输入端, 为低时选中芯片(数字量输入)
2	SCLK	串行时钟输入
3	VDD	正向电压输入端。通过一组0.1 μF 和4.7 μF 的并联电容与GND连接
4	GND	接地
5	DIN	串行数据输入端。数据在SCLK的上升沿被锁存
6	REF	外部参考电压输出端
7	OUTA	DAC电压输出。上电复位电路使得输出为0, 并使OUT端带100 kQ电阻接地
8	OUTB	
9	OUTC	
10	OUTD	

2. MAX5741 的 SPI 接口指令

MAX5741 有两种工作模式: 一种是低功耗的睡眠模式(Power-Down Mode); 另一种是正常的3线串行工作模式。它们分别有15条指令, 具体指令如表11.5所列(在此列出了部分指令, 其他的照推)。

3. MAX5741 的 SPI 接口工作时序图

标准的SPI 是8位、16位或32位结构, MAX5741 和 F2407 都是16位结构, 所以两者能很好地配合。DSP 提供了4种不同的时钟设计, 可参见11.2.5小节中的“串行外设接口的时钟方式”部分。

MAX5741 的工作时序如图11.7所示。可以看出: 当CS 为低电平时, 串行数据才能被移入输入寄存器。当CS 为低电平时, 在每一个SCLK 的时钟上升沿, 将 DIN 的一位数据移入16位输入寄存器。当16位数据全被移入串行输入寄存器之后, 输入寄存器的内容被锁入DAC 锁存器。在写下一组数据之前, CS 必须维持高电平至少80 ns。

表11.5 MAX5741 的 SPI 接口指令

控制位				数据位		通道	功 能
C3	C2	C1	C0	D9~D0	S1S0		
0	0	0	0	X	X	A	输入寄存器透明. 数据直接移位到DAC寄存器, OUTA输出更新
0	0	0	1	X	X	B	输入寄存器透明. 数据直接移位到DAC寄存器, OUTB输出更新
0	0	1	0	X	X	C	输入寄存器透明. 数据直接移位到DAC寄存器. OUTC输出更新
0	0	1	1	X	X	D	输入寄存器透明. 数据直接移位到DAC寄存器. OUTC输出更新
0	1	0	0	X	X	A	数据移位到输入寄存器. OUTA输出不变
0	1	0	1	X	X	B	数据移位到输入寄存器. OUTB输出不变
	1	1	0	X	X	C	数据移位到输入寄存器. OUTC输出不变
0	1	1	1	X	X	D	数据移位到输入寄存器. OUTD输出不变
1	0	0	0	X	X	A	数据从输入寄存器移位到DAC寄存器. OUTA输出更新
1	0	0	1	X	X	B	数据从输入寄存器移位到DAC寄存器. OUTB输出更新
1	0	1	0	X	X	C	数据从输入寄存器移位到DAC寄存器. OUTC输出更新
1	0	1	1	X	X	D	数据从输入寄存器移位到DAC寄存器. OUTD输出更新
1	1	0	0	X	X	A-D	输入寄存器透明, 数据直接移位到DAC寄存器, OUTA~OUTD输出更新
1	1	0	1	X	X	A-D	数据移位到输入寄存器. OUTA~OUTD输出不变
1	1	1	0	X	X	A-D	数据从输入寄存器移位到DAC寄存器. OUTA~OUTD输出更新

注：X 为不关心的值。

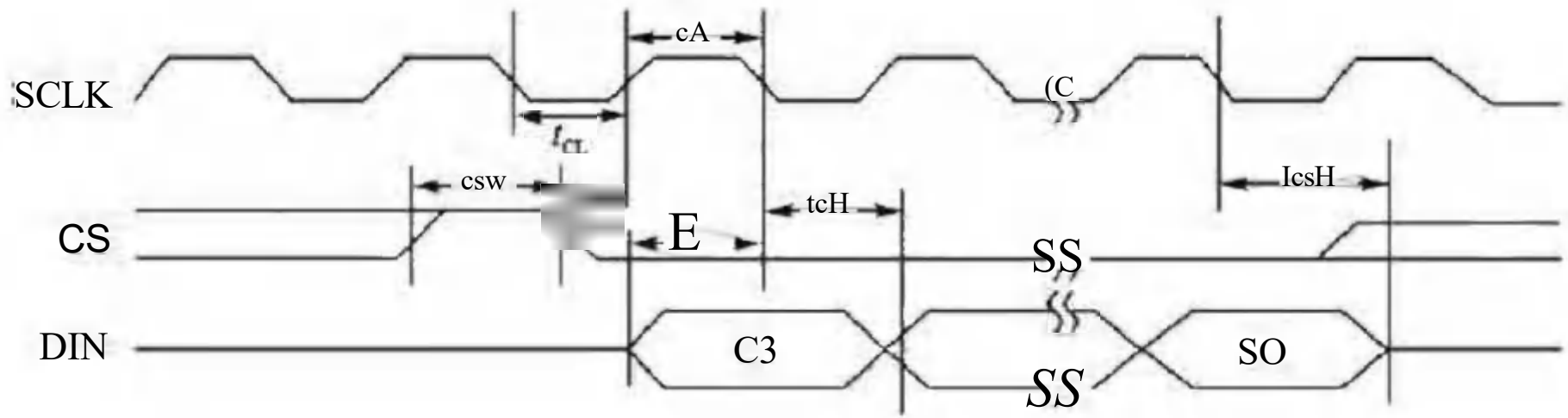


图11.7 MAX5741 的 SPI 时序图

4.MAX5741 与 DSP 的 SPI 接口电路

MAX5741兼容了SPI、QSPI、MICROWIRE 和 DSP 的串行接口。而F2407 DSP器件的SPI 是一个高速、同步串行I/O 口，它允许长度可编程的串行数据流(1~16位)以可编程的位传输速度移入或者移出器件。利用F2407 的 SPI(I/O) 口 与MAX5741构成的DAC 电路，如图11.8所示。

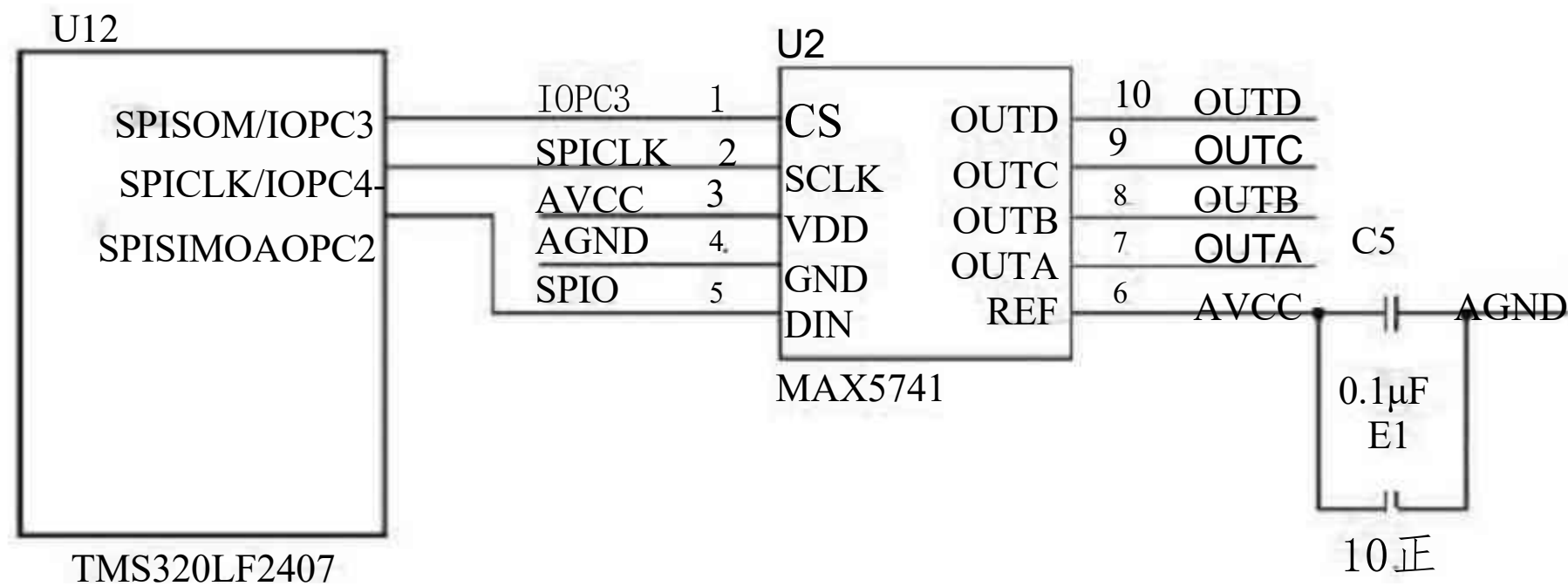


图11.8 F2407 与 MAX5741 接口电路

在图11.8中，F2407 的 SPISIMO 脚接到了 MAX5741 的 DIN。这是 F2407 的 SPI 从动输入/主动输出引脚，MAX5741 作为从机，即 MAX5741 只接收来自主机的数据，然后进行 D/A 转换，从 OUT 引脚输出模拟电压信号。

11.4.2 软件设计

1.SPI 总线波特率的设置

不同 SPI 器件的最高频率不同，为了使 SPI 总线高效工作，应了解各个 SPI 器件最高能接受的频率，再取较低频率的 SPI 器件的最高频率作为通信波特率。在此 DSP 的主频为 24 MHz，它作为 SPI 的主机时，SPI 总线频率可达到 1/4 主频（即 6 MHz）；作为 SPI 从机时，其频率也可达到 1/4 主频（即 6 MHz）。MAX5741 的 SPI 通信频率可达到 20 MHz。由于 DSP 只作为 SPI 的主机，因此本设计中 SPI 总线的最高频率为 6 MHz。

2.MAX5741 与 DSP 的 SPI 软件实现

本实例选用的是 16 位数据模式，编程时最主要应注意时序以及编写一个数据的输入子程序。利用 F2407 的 SPI 模块向 MAX5741 芯片发送数据用于 D/A 转换。以下是在 MAX5741 的 OUTA 端输出一个三角波来说明 SPI 总线的软件编程。由于 MAX5741 为 SCLK 的上升沿时接收 SPI 线上的数据，因此 DSP 用无延时的下降沿（或有延时的下降沿）时钟方式来发送 SPI 数据，这样才能配合 MAX5741 的 SPI 时序。采用软件查询的方式来发送数据。

(1) 汇编源程序

```

;用户寄存器定义
SPI_DATA      .usect      ".data0",1      ;临时数据寄存器
SPI_FLAG      .usect      ".data0",1      ;SPI 标志位
SPI_CON       .usect      ".data0",1      ;MAX5741的控制字
DP_USER       .set        5               ;用户寄存器存放页指针
;MAX5121的控制字宏定义
DACOUT        .set        0000H           ;0
UPINREG       .set        07FEH
UPDACREG      .set        OFFCH           ;1
;①主程序
                .include      "F2407REGS.H"
                .def          c_int0
                .text
c_int0
                CALL          SYSINIT      ;调系统初始化程序
                CALL          SPI_INIT     ;调 SPI 初始化程序
LOOP:          CALL          SPI_SEND     ;调输出三角波程序
    
```



```

NOP
WAIT:  LDP      #DP  USER
        SPLK    #00H,SPI_FLAG      ;清标志
        SPLK    #00H,SPI_DATA      ;重置初值
        B       LOOP

```

;②系统初始化程序

```

SYSINIT:
        SETC     INTM
        CLRC     SXM
        CLRC     OVM
        CLRC     CNF
        LDP      #0EOH
        SPLK     #81FEH,SCSR1      ; 4 倍 频CLKIN=6MHz,CLKOUT=24  MHz
        SPLK     #0E8h,WDCR        ;关看门狗
        LDP      #0
        SPLK     井0001h,IMR        ;使能中断1
        SPLK     #0FFFFh,IFR        ;清中断标志
        RET

```

;③SPI 初始化程序

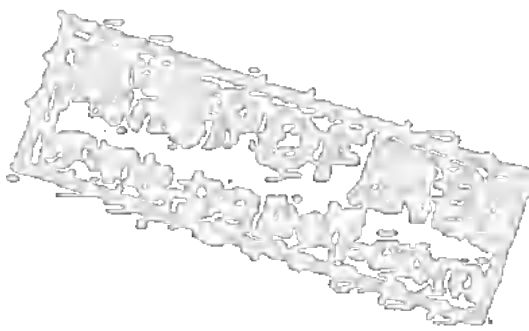
```

SPI_INIT:
        LDP      #DP_PF2
        LACL     MCRB
        OR       #00014H           ;配置 SPISIMO 和 SPICLK引脚为特殊功能方式
        SACL     MCRB
        LACL     MCRB
        AND      #OFFE7H           ;配置 IOPC3 为一般的 I/O 口功能
        SACL     MCRC              ;CS=IOPC3
        LDP      #DP_PF1
        SPLK     #004FH,SPICCR      ;配置SPI 寄存器允许初始化，16位数据输出
        SPLK     #0006H,SPICTL      ;主机方式，时钟方式为无延时的下降沿
        SPLK     #0002H,SPIBRR      ;SPI 波特率为6  MHz
        SPLK     #00CFh,SPICCR      ;初始化结束，并关闭初始化使能位
        LDP      #DP_USER
        SPLK     #00H.SPI_DATA      ;置发送数据初值
        SPLK     #00H.SPI_FLAG      ;SPI_FLAG=00h, 执行三角波上升
                                   ;SPI  FLAG=01h,执行三角波下降
        SPLK     #DACOUT,SPI_CON    ;送 MAX5741的控制字
        RET

```

;④输出三角波程序


```
SPI_SEND:
SPI_TX:  LDP      #DP_PF2
        LACL     PCDATDIR
        OR       #0800H           ;IOPC3 脚为输出方式
        AND      井OFFF7H        ;CS=IOPC3=0
        SACL     PCDATDIR
        NOP
        NOP
        LDP      #DP_USER
        LACC     SPI_DATA
        AND      #01FFEh         ;S0=0
        OR       SPICON
        SACL     SPI_DATA        ;规格化发送的数据
        LDP      #DP_PF1
        SACL     SPITXBUF        ;数据写入到 SPI 发送缓冲区
XMIT_RDY:BIT  SPISTS,BIT6        ;等待数据
        BCND     XMIT_RDY,NTC    ;发送完
        NOP
        NOP
        NOP
        LDP      #DP_PF2
        LACL     PCDATDIR
        OR       #0808H           ;CS=IOPC3=1
        SACL     PCDATDIR        ;锁存数据
        LDP      #DP_USER
        BIT      SPI_FLAG,BITO
        BCND     SPI_FALL,TC
        LDP      #DP_USER        ;三角波上升段程序
        LACC     SPI_DATA
        AND      #01FFEh
        ADD      #04H            ;递增
        SACL     SPI_DATA
        SUB      #01FFEh
        BCND     SPI_FALL, EQ
        B        SPI_TX
SPI_FALL:LDP  #DP_USER           ;三角波下降段程序
        SPLK     #01,SPI_FLAG
        LACC     SPI_DATA
        AND      #01FFEh
```



```
        SUB        #04H                ;递减
        SACL       SPI  DATA
        BCND       SPI_RET, EQ
        B          SPI_TX
SPI_RET:  RET
```

(2)C 语言源程序

//该程序利用F2407 的 SPI 模块向MAX5741 芯片发送数据用于D/A转换
Y/MAX5741的模拟输出端 A 输出一个三角波

```
#include    "register.h"
```

```
int         GPR3;
```

```
int         flag1;
```

```
int         flag;
```

1/系统初始化子程序

```
void        initial()
```

```
    asm("setc    INTM");
```

//屏蔽中断

```
    WSGR=0x00;
```

//禁止所有的等待状态

```
    asm("CLRC    SXM");
```

```
    asm("CLRC    OVM");
```

```
    asm("CLRC    CNF");
```

//B0 被配置为数据空间

```
    SCSR1=0x81FE;
```

//CLKIN=6 MHz,CLKOUT=24 MHz.

//并且使能到SPI 模块的时钟

```
    WDCR=0x0E8;
```

//不使能看门狗

```
    IMR=0x0000;
```

//禁止所有中断

```
    IFR=0x0FFFF;
```

//清除全部中断标志，写1清0

//SPI 模块初始化子程序

```
void        SPIINITIAL()
```

```
    SPICCR=0x0047;
```

//8char bits.在 SPICLK的下降沿发送数据，8位数

```
    SPICTL=0x0006;
```

//使能主动模式，一般的时钟方式，使能TALK

```
    SPIBRR=0x000b;
```

//SPI 的波特率为6 MHz,CLKOUT =24 MHz

```
    MCRB=0x0014;
```

//设置通用引脚为 SPI 功能的引脚，且 IOPC3 为一般

//I/O 口

```
    SPICCR=SPICCR|0x0080;
```

//使SPI 退出复位状态

//SPI 发送数据子程序，需要发送的数据存放在GPR3 寄存器中

```
int         SPITRANS(unsigned int  GPR3)
```

```

PCDATDIR=(PCDATDIR|0x0800)&0x0FFF7; // 设置 IOPC3 为输出, 且输出低电平,
//以选中MAX5741芯片
SPITXBUF=GPR3; //向 SPI 的发送缓冲器写一个发送数据
while(1)
{
    flag =SPISTS&0x40; //flag =SPI 的中断标志
    if(flag ==0x40) break; //如果SPI 的中断标志的中断标志为1,
//则证明SPI 已经发送成功, 则停止等待
}
SPIRXBUF=SPIRXBUF; //虚读 SPIRXBUF 寄存器, 以清除SPI 中断标志
PCDATDIR=PCDATDIR|0x08; //发送完毕, 则 IOPC3 输出高, 使发送的数据输
//出到MAX5741的输入寄存器和DAC寄存器中
}
//主程序
main()
{
    initial(); //系统初始化
    SPIINITIAL(); //SPI 初始化
    GPR3=0x0000; //根据MAX5741的格式写一个数据, 使其输出
//的模拟量为0
    flag1=0x00; //三角波上升沿、下降沿标志, 当flag1=0 时,
//为上升沿, 当flag1=1 时, 为下降沿
    SPITRANS(0x0f000); //唤醒DAC OUTA
    asm("rpt 165");
    asm("nop");
    while(1)
    {
        if(flag1==0x00) GPR3=GPR3+4; //因为数据从倒数第4位开始, 故要使数据加1,
//则需要实际的寄存器中加4, 此操作对应三角波的
//上升沿
        else GPR3=GPR3-4; //因为数据从倒数第4位开始, 故要使数据减1,
//则需要实际的寄存器中减4, 此操作对应三角波的
//下降沿
        if(GPR3==0x0FFc) flag1=0x01; //若需要转换的数据到FFFh, 则开始下降沿
        if(GPR3==0x0000) flag1=0x00; //若需要转换的数据到00h, 则开始上升沿
        SPITRANS(GPR3); //调用SPI 发送数据
    }
    //DAC OUTA 上输出一个模拟量

```



```
//当发生干扰中断时，执行此中断服务程序
void interrupt
{
    nothing()
}

return;
```

实际应用表明，由于F2407 自 带SPI 接口，与其他SPI 器件相连接结构简单、编程方便。

11.5 串行外设接口应用举例2

下面的设计是LED 数码管与F2407 的接口及应用。图11.9为F2407 的 SPI 与 LED 数
码管的接口电路。在通常情况下，需要用LED 数码管来显示系统的工作状态、运行结果和参

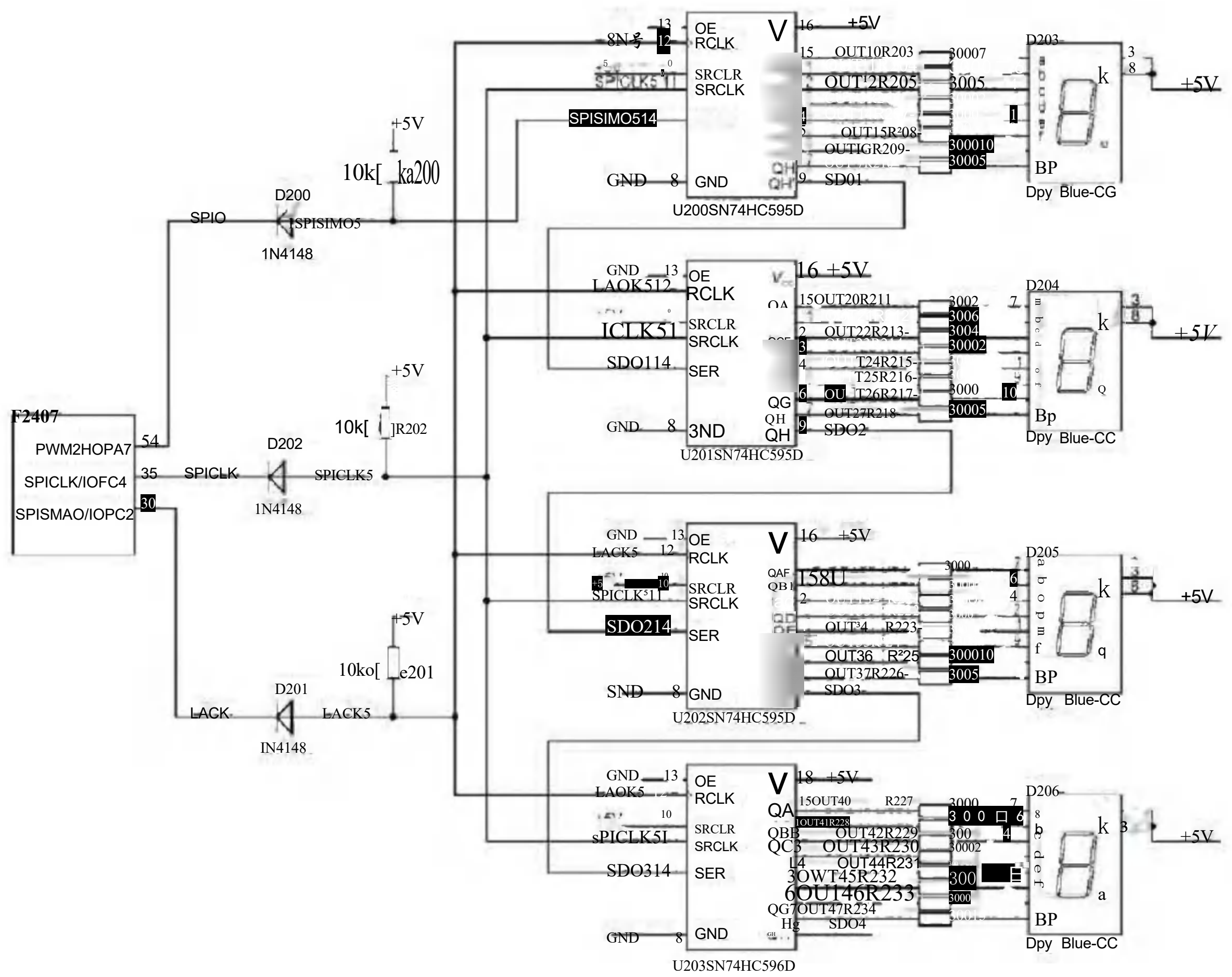


图11.9 F2407 的 SPI 与 LED 数码管的接口

数等信息。采用DSP 的 SPI 接口和移位寄存器芯片74HC595(该芯片把串行数据转换成并行数据输出,并具有一定的输出驱动能力,可直接驱动7段数码显示管)实现LED 的静态显示即通过级连方式把多个74HC595 芯片连在一起形成移位寄存器串,同时每个74HC595 芯片的并行输出连接到一个7段LED 数码管,这样就可通过SPI 的简单编程,实现多个数码管的显示功能。比如要在4个 LED 上分别显示1、2、3和4,可通过 SPI 口连续发送4个数据的显示段码,发送完毕后,通过锁存信号线LACK 发出一个锁存信号,便可以实现静态显示。又因为LED 数码管为+5 V,而 F2407 的输出为3.3 V,本应用设计使用二极管与上拉电阻构成电平匹配电路。

下面为 C 语言源程序:

```
#include    "register.h"
unsigned    table[5]={0x0f900,0x0A400,0xb000,0x9900,0x0c700};
//系统初始化子程序
void    initial()
{
    asm("setc    INTM");                //屏蔽中断
    WSGR=0x00;                          //禁止所有的等待状态
    asm("CLRC    SxM");
    asm("CLRC    OVM");
    asm("CLRC    CNF");                //B0 被配置为数据空间
    SCSR1=0x81FE;                      //CLKIN=6 MHz,CLKOUT=24 MHz,并且使能SPI 模块的时钟
    WDCR=0x0E8;                        //不使能看门狗
    IMR=0x0000;                        //禁止所有中断
    IFR=0x0FFFF;                      //清除全部中断标志, 写1清0
}

//SPI 模块初始化子程序
void    SPIINITIAL()
{
    SPICCR=0x0047;                    //在 SPICLK的下降沿发送数据8位数
    SPICTL=0x0006;                    //使能主动模式,一般的时钟方式,使能 TALK
    SPIBRR=0x000b;                    //SPI 的波特率为6 MHz,CLKOUT=24 MHz
    MCRB=0x0014;                      //设置通用引脚为 SPI 功能的引脚
    SPICCR=SPICCR|0x0080;             //使SPI 退出复位状态
}

//主程序
main()
{
    int    i=0,j;
    initial();                        //系统初始化
```

```
SPIINITIAL();           //SPI 初始化
MCRA=NCRA0x0FE7f;       //设置IOPA7 脚为一般的I/O功能
PADATDIR=0x807f;        //IOPA7 设置为输出方式，
                        //且IOPA7=0. 该端口为74HC595锁存信号

while(1)

    PADATDIR=0x807f;      //IOPA7 设置为输出方式，且IOPA7=0
                        //给LACK 信号一个低电平

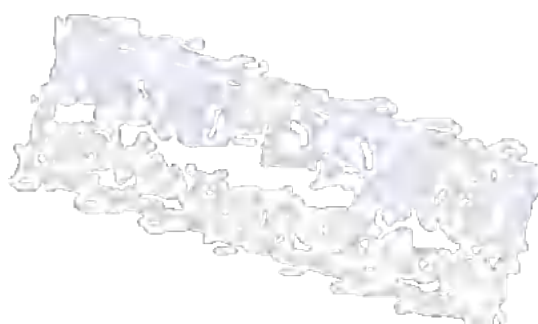
    for(i=0;i<4;i++)

        SPITXBUF  =table[i];    //给数码管送数
        while((SPISTSS0x0040)!=0x040)    台
        SPIRXBUE=SPIRXBUF;

    PADATDIR=0x80ff;      //IOPA7=1, 给 LACK 信号一个高电平
    for(j=0;j<10;j++)

//当发生干扰中断时，执行此中断服务程序
void interrupt nothing()

    return;
```



第 12 章 串行通信接口模块

12.1 串行通信接口概述

F240x 器件包括串行通信接口(SCI) 模块。SCI 模块支持 CPU 与其他使用标准格式的异步外设之间的数字通信。SCI 接收器和发送器是双缓冲的，每一个都有它自己单独的使能和中断标志位。两者都可以独立工作，或者在全双工的方式下同时工作。为了确保数据的完整性，SCI 对接收到的数据进行间断检测、奇偶性校验、过冲和帧出错的检查。通过一个16位波特率选择寄存器，数据传输的速度可被编程为65535多种不同的方式。SCI 模块的结构框图如图12. 1所示。SCI 模块的特性包括：

>两个外部引脚:

—SCITXD:SCI 发送数据引脚;

-SCIRXD:SCI 接收数据引脚。

在不使用SCI 时，这两个引脚可用作通用I/O 口。

通过一个16位波特率选择寄存器，可编程为64K 种不同速率的波特率。

在40 MHz 的 CPU 时钟方式下, 波特率范围从76 bps~1875 Kbps。

数据格式：一个起始位，1~8位的可编程数据字长度，可选择的奇/偶/无校验位，一个或两个停止位。

4种错误检测标志位：奇偶错、过冲、帧出错或间断检测。

> 两种唤醒多处理器方式：空闲线或地址位唤醒。

> 半双工或全双工操作。

＞双缓冲的接收和发送功能。

➤发送和接收的操作可利用状态标志位通过中断驱动或查询算法来完成。

— 发送器: TXRDY 标志(发送缓冲寄存器准备接收另一个字符)和 TX_EMPTY 标志位(发送移位寄存器空);

一 接收器: RXRDY 标志(接收缓冲寄存器准备接收另一个字符)、BRKDT 标志(间断事件发生)和 RX EERR 标志(监视4个中断条件)。

接收器的中断位可独立使能(除 BRKDT 外)。

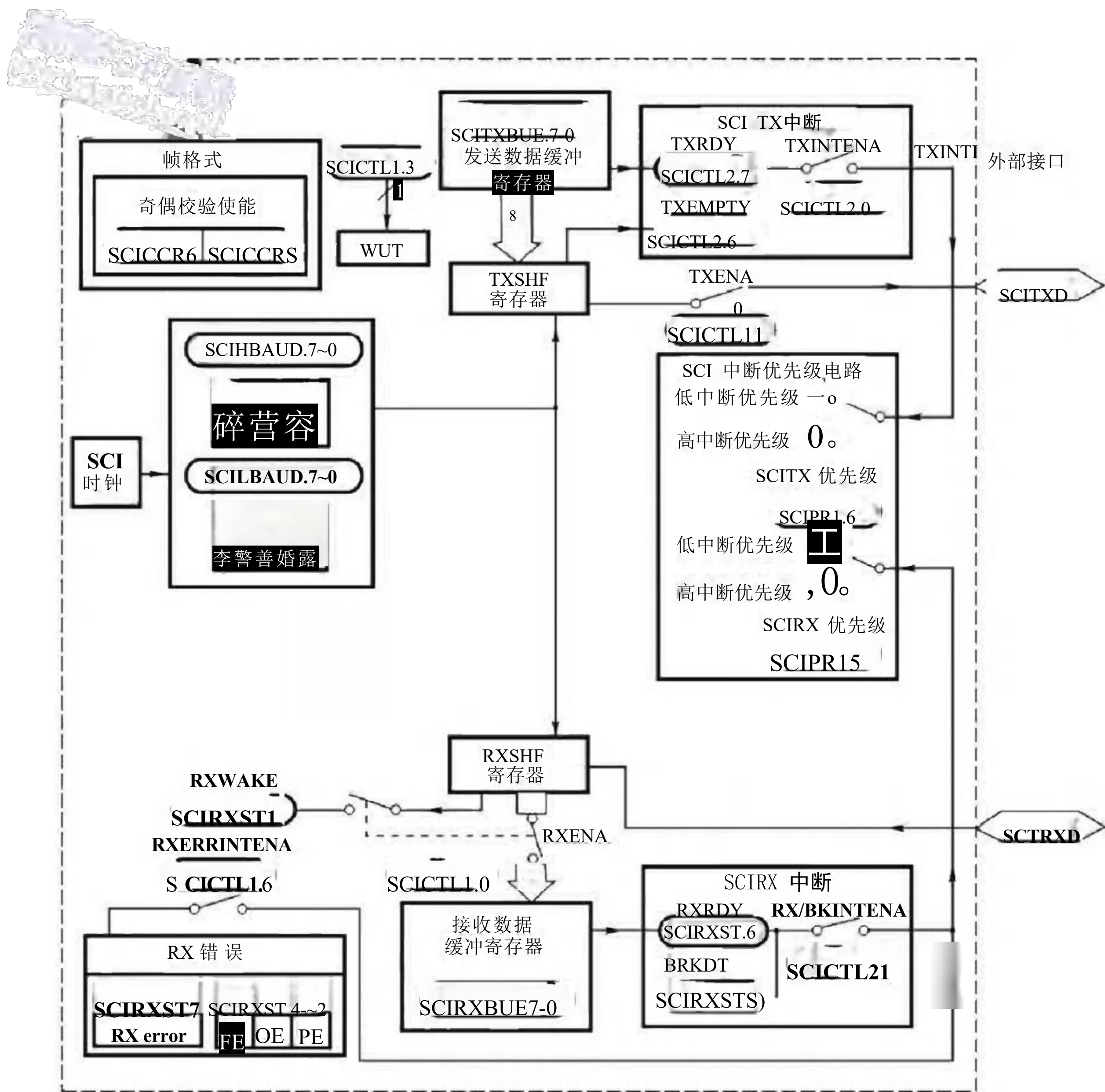


图 12.1 串行通信接口结构框图

▷不返回零(NRZ) 格式。

>SCI 模块的10个控制寄存器地址位于7050h~705Fh。

注意：SCI 模块内的所有寄存器为8位.但是它又与16位外设总线相连。因此当访问这些寄存器时，寄存器的数据在低字节(0~7位),高字节(8~15位)读为0,对高字节的写操作无效。

在双全工模式下 SCI 模块的主要部件如图12.1所示。包括以下部件：

一个发送器 TX 及与它相关的主要寄存器：

—SCITXBUF: 发送数据缓冲寄存器，存放要发送的数据；

—TXSHF: 发送移位寄存器，从发送数据缓冲寄存器(SCITXBUF) 中装入数据，并将数据移位至 SCITXD 引脚，一次移1位。

一个接收器 RX 及与它相关的主要寄存器：

—RXSHF: 接收移位寄存器，从SCIRXD 引脚移入数据，一次移1位；

=SCIRXBUF: 接收数据缓冲寄存器，存有由 CPU 读取的数据。来自串行通信线上的数据装载到接收移位寄存器(RXSHF) 后，又被装入到接收数据缓冲寄存器(SCIRXBUF) 和接收仿真缓冲寄存器(SCIRXEMU)。

> 一个可编程的波特率发生器。

> 数据存储器映射的控制和状态寄存器。

串行通信接口的接收器和发送器可以选用半双工和全双工工作方式。

12.2 多处理器和异步通信模式

串行通信接口模块有两个多处理器通信协议，空闲线多处理器模式和地址位多处理器模式。这些通信协议保证多处理器间进行有效的数据传送。

串行通信接口模块提供了与许多通用异步接收/发送(UART) 通信外设的接口。如异步通信使用3条线(地线、发送线和接收线)连接，诸如采用RS-232 格式的终端和打印机等众多标准器件。数据发送特征如下：

> 1个起始位；

> 1~8个数据位；

> 1个奇/偶校验位或无奇/偶校验位；

> 1~2个停止位。

12.2.1 串行通信接口可编程的数据格式

串行通信接口的数据，无论是接收和发送都采用NRZ(非返回零)格式。 NRZ 数据格式包括：

> 1个起始位；

> 1~8个数据位；

> 1个奇/偶校验位或无奇/偶校验位；

> 1~2个停止位；

> 1个用于区分数据和地址的额外位。

数据的基本单位被称作字符，为1~8位长。每个数据字符的格式为一个起始位，1~2个停止位，可选的奇偶位和地址位。带有格式信息的数据字符叫做一帧，其格式如图12.2所示。可通过设置串行通信接口通信控制寄存器SCICCR 来编程数据格式。



图12.2 典型的串行通信接口数据帧格式

12.2.2 串行通信接口的多处理器通信

多处理器通信模式使得一个处理器能有效地在同条串行连接线上将数据传送到其他的处理器。一条串行线上一次只能进行一次传送，即一条串行线上只能有一个人说话(广播方式)。

地址字节。说话者发送的数据块的第一个字节包括一个地址字节，它被所有的听众读取(侦听方式)，但只有地址相符的听众才能接收跟在地址字节后面的数据，地址不相符的听众不接收地址字节后面的数据，且等待接收下一个地址字节。

SLEEP 位。串行连接的所有处理器将其串行通信接口的SLEEP 位(SCICTL.2) 置为1，使得它们在检测到地址字节时才被中断。当处理器读取到的地址与应用软件设置的本处理器地址相符时，用户程序必须清除SLEEP 位，来确保串行通信接口在收到每个数据字节时产生一个中断。

虽然接收器在 SLEEP 位为1时仍能工作，但除非检测到地址字节并且接收帧中的地址位是1；否则它不能将RXRDY、RXINT 或任何接收错误状态位置为1。串行通信接口不会改变SLEEP 位，必须由用户软件改变。

处理器根据多处理器的模式来识别地址字节，例如：

- ›空闲线模式在地址前留有一个固定空间。该模式没有附加的地址/数据位，它在处理包含多于10字节的数据块方面比地址位模式更有效。空闲线模式应用于典型的非多处理器的SCI 通信。
- ›地址位模式在每个字节中加入一个额外位(地址位)来区分地址和数据。这种模式在处理多个小数据块时更有效。因为它不像空闲线模式在数据块之间那样需要等待。当处于高速发送时，空闲线模式的程序速率不足以避免传送中的10位空闲位。

多处理器的模式可通过 ADDR/IDLE MODE 位(SCICCR.3) 来设置。两种模式都使用发送唤醒标志位 TXWAKE(SCICTL1.3)、接收唤醒标志位 RXWAKE(SCIRXST.1) 和休眠

标志位 SLEEP(SCICTL1.2) 来控制串行通信接口发送器和接收器的工作状态。

在两种多处理器模式中，接收步骤如下：

① 接收地址块时，串行通信接口唤醒并请求中断，所以必须使能RX/BK INTENA 位(SCICTL2.1) 以请求中断。它读取地址块的第一帧数据，其中包括目的处理器的地址。

② 通过中断和校对程序得到的地址进入软件子程序，并且该地址字节与存在内存中的器件地址再次进行校对。

③ 如果校对表明地址块与该DSP 控制器的地址相符，则 CPU 就清除 SLEEP 位并读取块中剩余的数据；否则，退出软件子程序并且 SLEEP 位依旧置位，直到下一个地址块的开始才接收中断。

1. 空闲线多处理器模式

在空闲线多处理器协议中(ADDR/IDLE MODE=0),数据块被块间的时间间隔分开，该时间间隔比块中数据帧之间的时间间隔要长。一帧后的空闲时间(10个或更多个高电平位)表明了一个新块的开始，单个位的时间可直接由波特率的值(位/秒)计算得到。空闲多处理器的通信格式如图12.3所示。

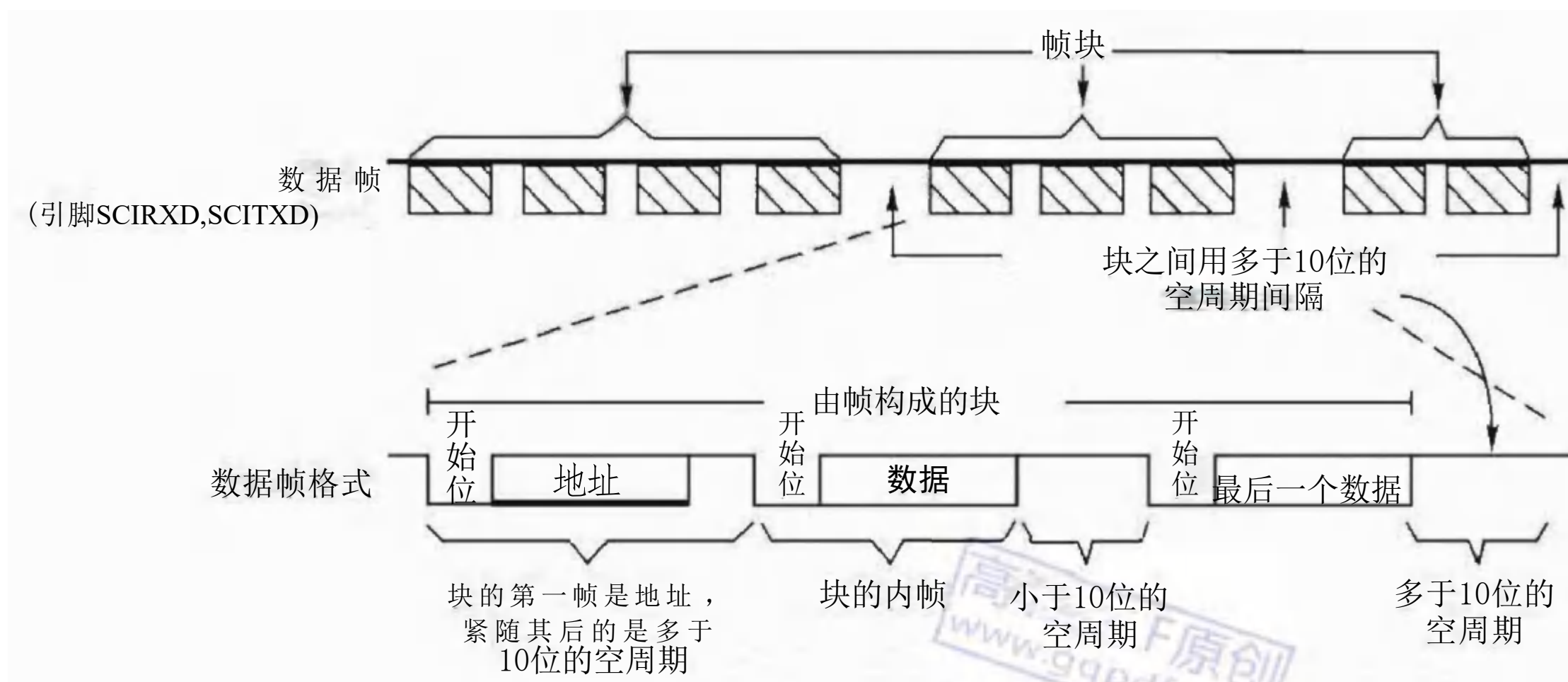


图 12.3 空闲线多处理器的通信格式

空闲线模式采用的步骤如下：

① 接收到块启动信号后串行通信接口唤醒。

② 处理器识别出下一个串行通信接口中断。

③ 中断服务子程序将接收到的地址与自己的地址进行比较。

④ 如果该CPU 的地址与接收到的地址相符，则中断服务子程序就清除 SLEEP 位，并接收块中剩余的数据。

⑤ 如果该CPU 的地址与接收到的地址不相符，则SLEEP 位仍保持置位。这使得在检测

到下一个块的开始前，CPU 继续执行其主程序而不被串行通信接口所中断。

有两种方法可传送块的开始信号：

① 在前一块最后一帧的数据传送与新块的地址帧传送之间特意留出10位或更多位的空闲时间作为延时。

② 在写入到SCITXBUF 寄存器之前，串行通信接口首先将发送唤醒位 TXWAKE 置 1，并传送11位的空闲时间。用这种方法使串行通信线只空闲必要的一些位。

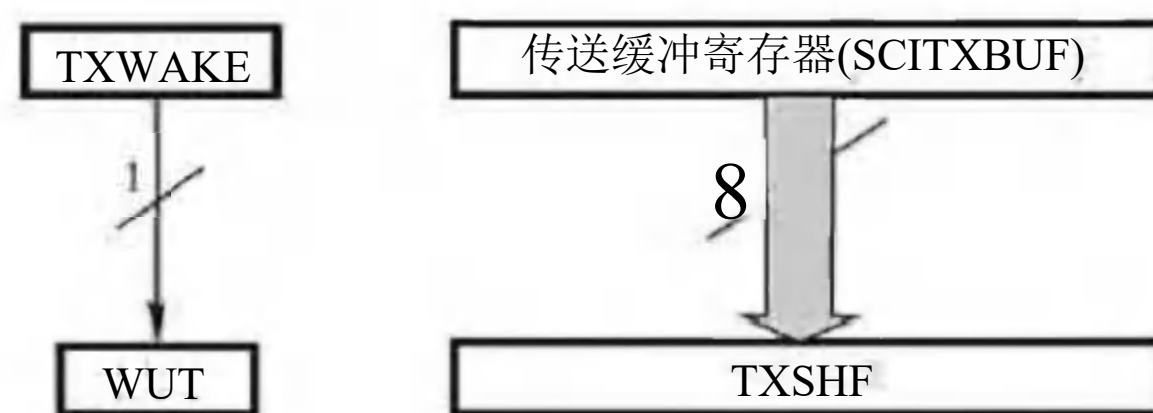
与发送唤醒位 TXWAKE 相关的是唤醒暂时标志位 WUT，这是一个内部标志，与 TXWAKE 构成双缓冲。当发送器的移位寄存器 TXSHF 从串行通信接口发送数据缓冲寄存器 SCITXBUF 装载数据时，WUT 从发送唤醒标志位 TXWAKE 中装入数据，然后 TXWAKE 位清0。这种处理如图12.4所示。

在发送期间，一帧时间的块开始信号执行以下操作：

① 设置发送唤醒位 TXWAKE 为 1。

② 将一个数据字(内容不重要)写到 SCTTXBUF 寄存器作为传送一个块的开始信号。(在块开始信号被发送的同时，第一个

写入的数据字被移位，且在块开始信号发送后被清零。)当 TXSHF 寄存器再次空闲时，SCITXBUF 寄存器的内容被移位到 TXSHF 寄存器中，发送唤醒标志位被移位到 WUT，而后 TXWAKE 位清0。



注：WUT- 唤醒暂存。

图12.4 WUT 和 TXWAKE 的双缓冲

因为 TXWAKE 被置成1，所以起始位、数据位和奇偶位被在上一帧最后停正位后的11位空闲位所取代。

③ 将新地址写入SCITXBUF 寄存器。

这个无关数据字必须首先被写入到SCITXBUF 寄存器，从而TXWAKE 位的值可以移入 WUT 中。在无关数据被移入 TXSHF 中后，因为 TXSHF 和 WUT 都是两级缓冲，因而 SCITXBUF 寄存器(或TXWAKE 位)可重新写入新的值。

接收器的操作与 SLEEP 位无关，但是在检测到一个地址帧之前，接收器并不置位 RXRDY 位，也不置位错误状态位，并且也不请求接收中断。

2. 地址位多处理器模式

在地址位多处理器协议中(ADDR/IDLE MODE=1),帧格式中有一个叫做地址位的额外位，它紧跟在最后一个数据位之后。在数据块第一帧中地址位被置为1，在其他帧中被置为0。空闲周期时序是间隔的，地址位多处理器通信格式如图12.5所示。

TXWAKE 位的值被放置到地址位中。在发送期间，当SCITXBUF 和 TXWAKE 的值分别装载到TXSHF 和 WUT 中时，TXWAKE 复位为0，而 WUT 则变成当前帧的地址位的值。

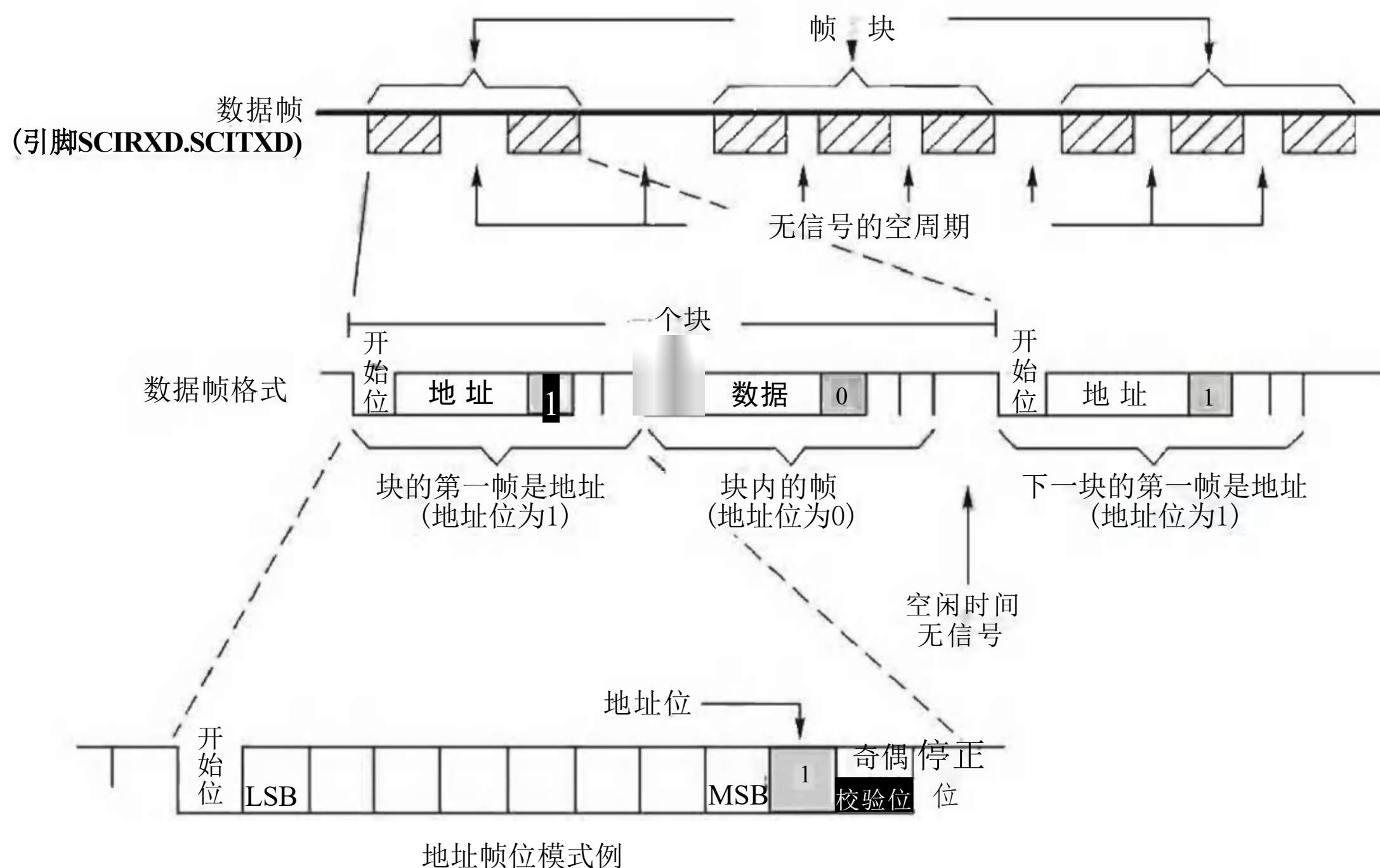


图12.5 地址位多处理器通信格式

因此，发送一个地址的步骤如下：

① 设置 TXWAKE 位为1, 并且将一个合适的地址写入到 SCITXBUF 寄存器中。当该地址值被移入 TXSHF 寄存器中且又被移出时，其地址位被置为1, 这就向串行线上的其他处理器发送标志使其读取地址。

② 因为 TXSHF 和 WUT 都是双级缓冲的，因此 SCITXBUF 和 TXWAKE 可在 TXSHF 和 WUT 装入后，立即写入新值。

③ 在发送块中的无地址帧(即数据帧)时，应将 TXWAKE 位置为0。

12.2.3 串行通信接口通信格式

串行通信接口异步通信格式可使用半双工或全双工通信方式。这种模式下，一帧包括一个起始位，1~8个数据位，一个可选的奇偶校验位，1~2停止位。如图12.6所示，每个数据位占用 8 个 SCICLK 周期。

接收器在收到有效的起始位后开始操作。有效的起始位由如图12.6所示的4个连续内部 SCICLK 周期的零位识别出来。如果任一位不为0, 则处理器的启动结束并开始寻找另一个起始位。

对于起始位后的位，处理器通过对中间位采样3次来决定其位值。这些采样点出现在第

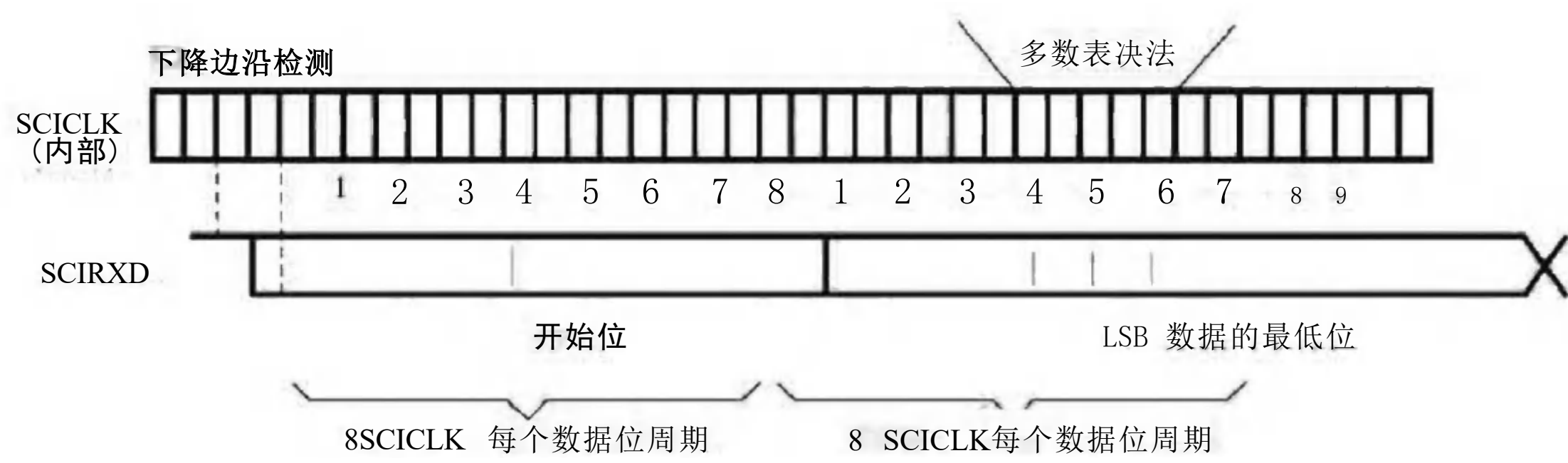


图12 . 6 串行通信接口异步通信格式

4、第 5 和 第 6 个SCICLK 周期，位值取决于多数(多数指2/3以上)。图12 . 6表明了这种异步通信格式，其中的启始位表示如何找到边沿和哪里作出多数决定。

因为接收器本身能够按帧进行同步，所以外部发送和接收器不必使用同步串行时钟，该时钟可局部生成。

1. 通信模式中的接收器信号

图12. 7表明了如下假设条件下接收器信号时序的一个例子。

- 地址位唤醒模式(地址位不出现在空闲线模式中);
- > 每个字符6位。

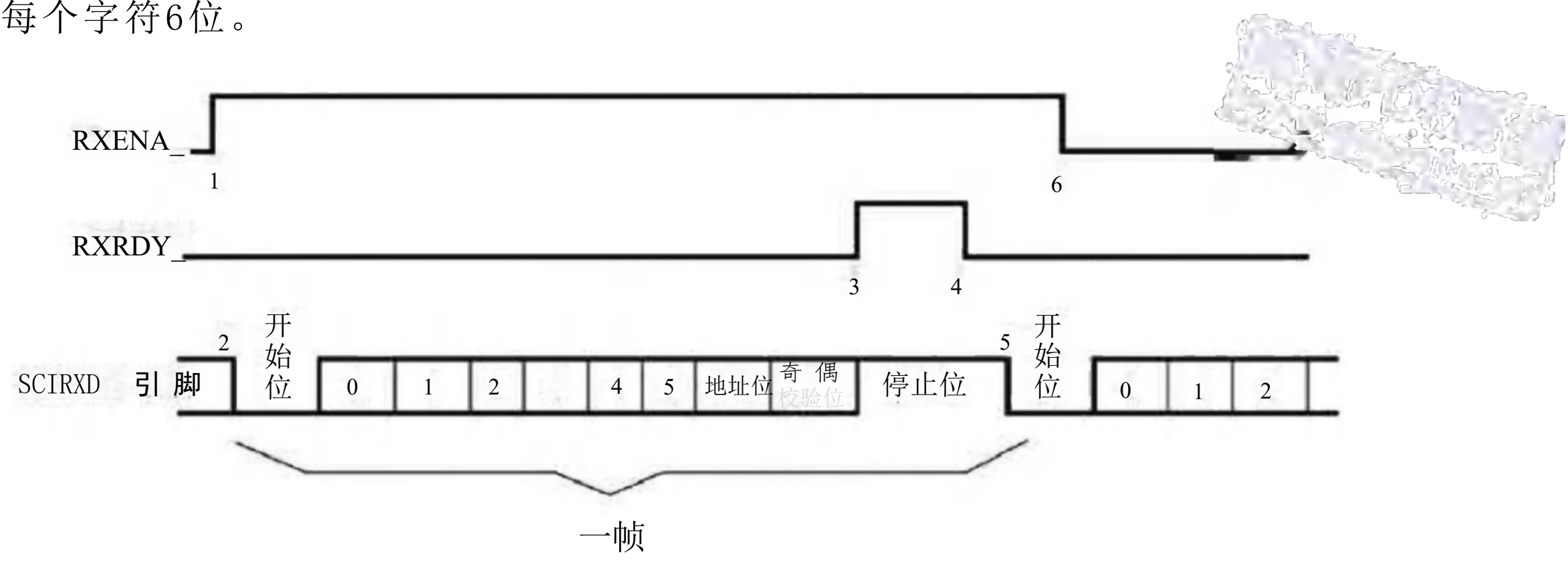


图12 . 7 串行通信接口在通信模式中接收器信号

在使用接收器时应注意以下几点：

- 设置RXENA 位为1, 来使能接收器接收数据。
- 数据到达SCIRXD 引脚后，检测到启始位。
- > 数 据 从 RXSHF 寄存器移到SCIRXBUF 寄存器中，产生中断请求。标志位 RXRDY 变为1 . 表示已收到一个新字符。

› 当读 SCIRXBUF 寄存器时, 标志位 RXRDY 自动被清除。

数据的下一个字节到达 SCIRXD 引脚, 检测到起始位, 然后清除。

› 如果 RXENA 变低, 则接收器被禁止。数据继续保持在 RXSHF 寄存器中, 但没有移入到 SCIRXBUF 寄存器中。

2. 通信模式中的发送器信号

图12.8表明了如下假设条件下发送器信号时序的一个例子。

› 地址位唤醒模式(地址位不出现在空闲线模式中);

› 每个字符3位。

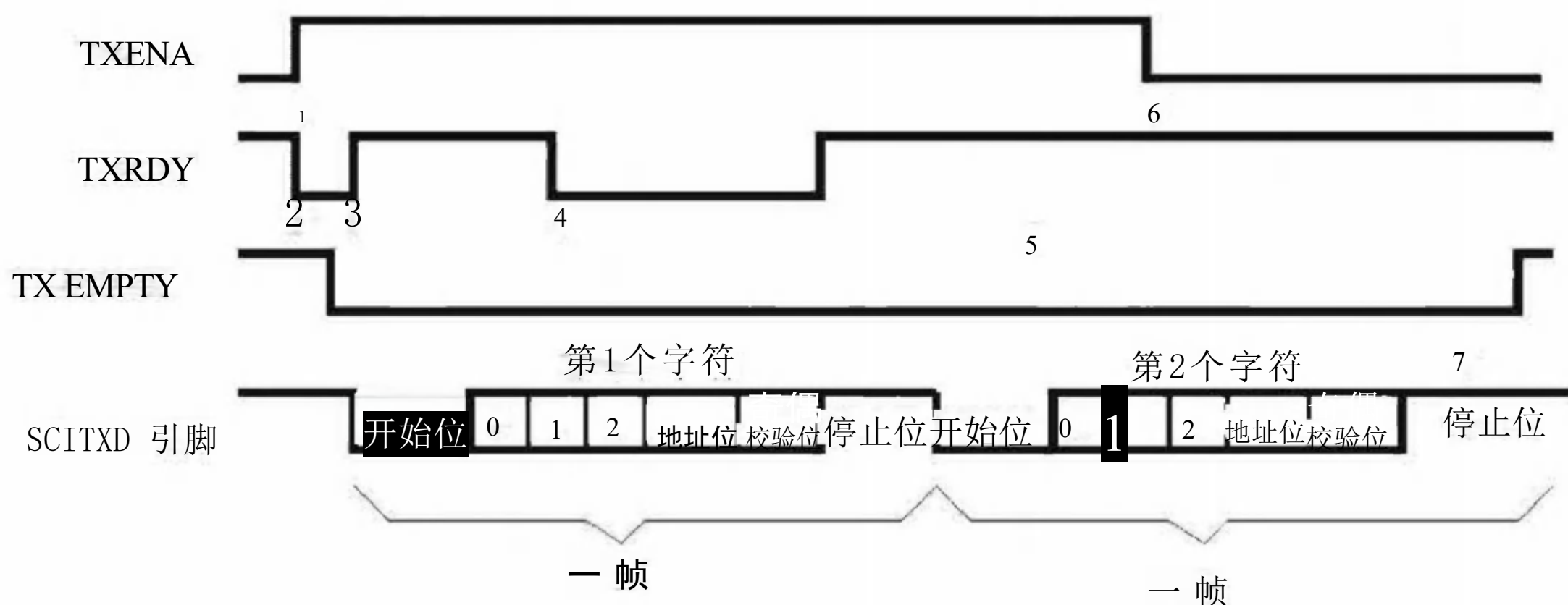


图12.8 串行通信接口在通信模式中发送信号

在使用发送器时应注意以下几点:

设置 TXENA 位为1, 来使能发送器发送数据。

› SCITXBUF 寄存器被写入, 从而发送器不再为空, TXRDY 变低。

› 串行通信接口将数据传送到 TXSHF 寄存器。发送器准备传送第2个字符(TXRDY 变高), 并发出发送中断请求(为使能发送中断, 必须将 TXINT ENA 位置为1)。

› TXRDY 变高后, 程序将第2个字符传送到 SCITXBUF 寄存器, 写入后 TXRDY 又变低。

› 发送完第1个字符后, TX EMPTY 位暂时升高。将第2个字符传送到 TXSHF 寄存器, 使操作开始。

› 如果 TXENA 位变低, 则发送器被禁止。串行通信接口结束发送当前字符。

› 发送完第2个字符后, 发送器变空并准备发送新字符。

12.2.4 串行通信接口中断

串行通信接口的接收器和发送器可由中断来控制, SCICIL 有一个标志位(TXRDY) 用

来指示有效的中断条件，SCIRXST 寄存器中有两个中断标志位(RXRDY 和 RBKDT)。发送器和接收器有各自的中断使能位。当中断屏蔽时，中断不会产生。但条件标志位仍然有效，该位反映了发送和接收的状态，可用于查询方式。

串行通信接口的发送器和接收器有自己独立的外设中断向量。外设中断请求可使用高优先级或低优先级，中断优先级由SCIPRI 寄存器中相应的位来控制。当接收和发送中断都设置为相同的优先级时，接收中断往往具有更高的优先级，这是为了减少接收过冲错误。

如果RX/BK INT ENA 位置位，则当发生以下情况之一就产生一次接收中断。

>串行通信接口接收到一个完整的帧并将 RXSHF 寄存器中的数据传送到 SCIRXBUF 寄存器中。该操作置位 RXRDY 标志位，并初始化中断。

>间断检测条件发生(在一个停止位后 SCIRXD 引脚保持10个周期的低电平)。该操作置位BRRDY 标志位，并初始化中断。

如果TX INT ENA 位置位，那么无论何时，SCITXBUF 寄存器中的数据传送到TXSHF 寄存器中，都将确定一个发送中断，用以指示 CPU 可以把新数据写入到 SCITXBUF 寄存器中。该操作置位TXRDY 标志位，并使中断复位。

12.2.5 串行通信接口波特率计算

内部生成的串行时钟由系统时钟 SYSCLK 频率和波特率选择寄存器决定。串行通信接口使用16位波特率选择寄存器，数据传输的速度可以被编程为65000多种不同的方式。

不同通信模式下的串行通信接口异步波特率由下列方法决定：

>BRR=1~65535 时的串行通信接口异步波特率为：

$$\text{SCI 异步波特率} = \text{SYSCLK} / [(\text{BRR} + 1) \times 8]$$

其中， $\text{BRR} = \text{SYSCLK} / (\text{SCI 异步波特率} \times 8) - 1$

>BRR=0 时的串行通信接口异步波特率为：

$$\text{SCI 异步波特率} = \text{SYSCLK} / 16$$

这里 BRR 等于波特率选择寄存器的16位值。

【例 12.1】 若已知系统时钟频率 SYSCLK=24 MHz, 现要用波特率为9600 b/s 进行 RS-232 通信，试确定波特率选择寄存器的值即BRR。

根据上面计算 BRR 的公式可知：

$$\text{BRR} = 24 \times 10^6 / (9600 \times 8) - 1 = 312.5 - 1 = 311.5 \approx 138h$$

所以，波特率选择寄存器设置为 SCIHBAUD=01h.SCILBAUD=38h。



12.3 串行通信接口控制寄存器

串行通信接口的功能可由软件配置。通过设置相应控制寄存器的位来初始化所需的串行通信接口通信格式，它包括操作模式、协议、波特率、字符长度、奇/偶校验位、停止位的位数、中断优先级和使能控制位。现将其分别介绍如下。

串行通信接口通信控制寄存器(SCICCR) 定义了用于SCI 的字符格式、协议和通信模式。

1. 串行通信接口通信控制寄存器(SCICCR)—— 地 址 h

7	6	5	4	3	2	1	0
STOP BITS	EVEN/ODD PARITY	PARITY ENABLE	LOOPBACK ENA	ADDR/IDLE MODE	SCICCHAR2	SCICCHAR1	SCICCHAR0
RW_0	RW_0	RW_0	RW_0	RW_0	RW_0	RW_0	RW_0

位 7 STOPBITS。SCI 停止位选择位。

0 一个停止位；

1 两个停止位。

位 6 EVEN/ODD PARITY。SCI 奇/偶校验选择位。如果 PARITY ENABLE 被置位，则校验才有效，即判定发送和接收的字符中1的位数为奇数或偶数。

0 奇校验；

1 偶校验。

位 5 PARITY ENABLE。SCI 奇/偶校验使能位。

0 禁止奇/偶校验；

1 使能奇/偶校验。

位 4 LOOP BACK ENA。自测试模式使能位。如果该位使能，则发送引脚与接收引脚在系统内部连接在一起。

0 禁止自测试模式；

1 使能自测试模式。

位 3 ADDR/IDLE MODE。SCI 多处理器模式选择位。

0 选择空闲线多处理器模式；

1 选择地址位多处理器模式。

位2~0 SCI CHAR2~0。SCI 字符长度选择位。

000 1 位 001 2 位

010 3 位 011 4 位

100 5 位 101 6 位

110 7 位 111 8 位

串行通信接口控制寄存器1(SCICTL1) 控制着接收器和发送器使能位、TXWAKE 和 SLEEP 功能、内部时钟使能以及串行通信接口的软件复位。

2. 串行通信接口控制寄存器1(SCICTL1)—— 地址7051h

7	6	5	4	3	2	1	0
保留位	RX ERR INT ENA	SW RESET	保留位	TXWAKE	SLEEP	TXENA	RXENA
R_0	RW_0	RW_0	R_0	RS_	RW_0	RW_0	RW_0

注：R= 可读；W=可写；S= 只可被置位；0=复位值

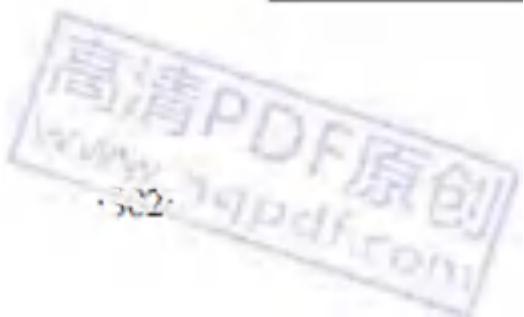
- 位 7 保留位。
- 位 6 RX ERR INT ENA。SCI接收错误中断使能位。如果该位置位，当接收发生错误时，RX ERROR 位将被置位，并且发出接收错误中断。

0 禁止接收错误中断；
1 使能接收错误中断。
- 位 5 SW RESET。SCI软件复位位(低有效)。
 - 将0写入该位来初始化 SCI 状态机和操作标志(SCICTL2 和 SCIRXST 寄存器)至复位条件。 SW RESET 位并不影响其他任何配置位。
 - 所有起作用的逻辑都保持确定的复位态，直至将1写入 SW RESET 位。因此，系统复位后，应将该位置为1来重新使能 SCI。
 - 当接收间断检测(BRKDT 标志位)发生后，将清除 SW RESET 位。
 - SW RESET 影响串行通信接口的操作标志，但不影响配置位，也不恢复复位位。表12. 1列出了受影响的标志位。一旦置位了 SW RESET,标志位就被不再改变直到该位清0。

注意：当SW RESET 位=1时不要改变配置，SCI 的配置只有在SW RESET位清0后才能设置或改变。在置位 SW RESET前，应设置好所有的配置寄存器，否则将会产生不可预测的结果。

表12.1 受 SW RESET位影响的标志位

SCI标志	寄存器的位	SW复位后的值	SCI标志	寄存器的位	SW复位后的值
TXRDY	SCICT1. 2. 7	1	FE	SCIRXST. 4	0
TX EMPTY	SCICT1. 2. 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			



- 位 4

保留位。
- 位 3

TXWAKE。SCI 发送器唤醒方法选择位。

0 没有选定的发送特征；

1 选定的发送特征取决由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 SCITXBUF 寄存器来产生一个11位数据位的空闲周期。

在地址位模式下：写1到TXWAKE，然后将数据写入SCITXBUF 寄存器并设置该帧的地址位为1。
- 位 2

SLEEP。SCI 休眠位。

0 禁止休眠方式；

1 使能休眠方式。
- 位 1

TXENA。SCI 发送使能位。仅当TXENA 置位时，数据才能从 SCITXD 引脚上发送出去。如果复位，则把已写入到 SCITXBUF 寄存器中的数据发送完后才停止发送。

0 禁止发送；

1 使能发送。
- 位 0

RXENA。SCI 接收使能位。从SCIRXD 引脚上接收到的数据送到接收移位寄存器，然后再送到接收缓冲器。该位使能或禁止接收(传送到缓冲器)。

0 禁止将接收到的数据传送到 SCIRXBUF 和 SCIRXEMU 接收缓冲器；

1 发送将接收到的数据传送到SCIRXBUF 和 SCIRXEMU 接收缓冲器。

清除 RXENA 即停止将接收到的数据传送到两个接收缓冲器的操作，并将停止接收中断的产生。但是，接收移位寄存器(RXSHF) 仍可以继续接收 SCIRXD 引脚上的数据。因此，如果在接收字符期间置位RXENA，则已接收到的数据将被传送到接收缓冲器 SCIRXBUF 和 SCIRXEMU 中。

串行通信接口控制寄存器2(SCICTL2) 用来反映发送准备好和发送缓冲器清空，及使能间断检测和 SCITXBUF 中断。

3. 串行通信接口控制寄存器2 (SCICTL2)—— 地址7054 h

7	6	5~2	1	0
TXRDY	TX EMPTY	保留位	RX/BKINT ENA	TX INT ENA
R_1	R_1	R_0	RW_0	RW_o

- 位 7

TXRDY。发送缓冲寄存器准备好标志位。写数据到SCITXBUF 寄存器的操作将自动清除该位。如果 TX INT ENA 位被置位，则当TXRDY 置位时，该标

志位使能发送器中断请求。通过使能 SW RESET 位或系统复位来将该 TXRDY 置位。

- 0 SCITXBUF 满；
- 1 SCITXBUF 空，准备接收下一个数据。

位 6 TX EMPTY。发送器空标志位。

- 0 SCITXBUF 寄存器、TXSHF 寄存器或两者都装入了数据；
- 1 SCITXBUF 寄存器和 TXSHF 寄存器都空。

位5~2 保留位。

位 1 RX/BK INT ENA。接收缓冲器/间断中断使能位。该位控制由 RXRDY 或 RBKDT 标志位置位引起的中断请求。然而，RX/BK INT ENA 并不阻止这些标志位置位。

- 0 禁止RXRDY/BRKDT 中断；
- 1 使能 RXRDY/BRKDT 中断。

位0 TX INT ENA。发送缓冲器(SCITXBUF) 中断使能位。该位控制 TXRDY 标志位引起的中断，但并不阻止 TXRDY 标志位的置位。

- 0 禁止 TXRDY 中断；
- 1 使能 TXRDY 中断。

4. 串行通信接口波特率选择高字节寄存器(SCIHBAUD) —— 地 址 7052 h

15	14	13	12	11	10	9	8
BAUD15 (MSB)	BAUD14	BAUD13	BAUD12	BAUD11	BAUD10-	BAUD9	BAUD8
RW_0	RW_0	RW_0	RW_0	RS=0	RW_g	RW0	RW_0

5. 串行通信接口波特率选择低字节寄存器(SCILBAUD)—— 地址7053h

7	6	5	4	3	2	1	0
BAUD7	BAUD6	BAUD5	BAUD4	BAUD3	BAUD2	BAUDI	BAUDO (LSB)
RW_0	RW_0	RW_0	RW_0	RS-0	RW_0	RW_0	RW_0

位15~0 BAUD15~BAUDO。 串行通信接口16位波特率选择位。 SCIHBAUD（高字节）和 SCILBAUD(低字节)连接在一起形成16位波特率值。波特率的计算可参见12.2.5小节。



串行通信接口状态寄存器(SCIRXST) 包括7个接收状态标志位，其中2个可产生中断请求。每次将一个完整的数据传送到接收缓冲器(SCIRXEMU 和 SCIRXBUF) 时，这些状态标志位都被更新。每次读接收缓冲器时，标志位被清除。

6. 串行通信接口接收状态寄存器(SCIRXST) ——地址7055 h

7	6	5	4	3	2	1	0
RX ERROR	RXRDY	BRKDT	FE	OE	PE	RXWAKE	保留位
R_0	R_0	R_0	R_0	R_0	R_0	R_0	R_0

位 7 RX ERROR。SCI接收器错误标志位。RX ERROR 标志位表明接收状态寄存器中的一个错误标志位被置位。RX ERROR 是间断检测、帧错误、过冲和校验允许标志的逻辑或。该错误标志位不能被直接清除，它由有效的SW RESET或系统复位来清除。

- 0 无接收错误标志被置位；
- 1 有接收错误标志被置位。

位 6 RXRDY。SCI 接收器准备好标志位。当准备将新数据读入 SCIRXBUF 中时，接收器置位该位。如果RX/BK INT ENA位是1, 将产生接收中断。RXRDY 位的清除可通过读SCIRXBUF 寄存器、有效的SW RESET或系统复位来清除。

- 0 SCIRXBUF 中无新数据；
- 1 准备从 SCIRXBUF 中读取新数据。

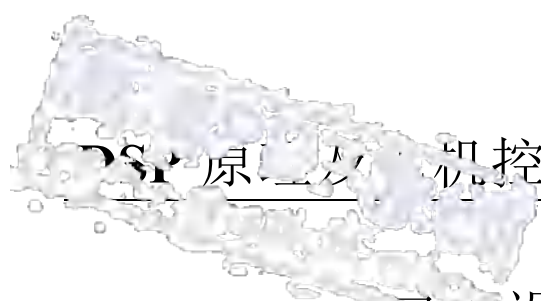
位 5 BRKDT。SCI 间断检测标志位。产生间断检测时，置位该位。当SCI 的接收数据引脚 SCIRXD 在从失去第1个停止位开始，连续保持低电平至少10位时，即满足间断条件。如果 RX/BK INT ENA 位是1, 将产生接收中断。但是这并不会影响接收器缓冲器的装入操作。即使接收器的 SLEEP 位置为1，也将产生 BRKDT 中断。该位可通过有效的SW RESET或系统复位来清除。

- 0 不满足间断条件；
- 1 满足间断条件。

位 4 FE。SCI 帧错误标志位。当没有找到预期的停止位时，串行通信接口将设置该位。只检测到第一个停止位。丢失的停止位表明起始位的同步性已丢失，数据帧格式错误，该位可通过有效的SW RESET或系统复位来清除。

- 0 未检测到帧错误；
- 1 检测到帧错误。

位 3 OE。SCI 过冲错误标志位。当被 CPU 或 DMAC 未完成前一个数据的读取前，而下一个数据又传送到 SCIRXEMU 和 SCIRXBUF 寄存器中，串行通信接



已将设置该位，表明以前的数据被重写并丢失。该位可通过有效的 SW RESET 或系统复位来清除。

0 未检测到过冲错误；

1 检测到过冲错误。

位 2 PE。SCI 奇/偶校验错误标志位。当收到的数据中1的数目与其奇/偶校验不匹配时，该标志位被置位。地址位包括在计算之内。如果奇/偶校验位的产生和检测未被使能时，则 PE 标志位禁止并读出总为0。该位可通过有效的 SW RESET 或系统复位来清除。

0 未检测到奇/偶校验错误；

1 检测到奇/偶校验错误。

位 1 RXWAKE。SCI 接收器唤醒检测标志位。该位为1表明检测到接收器唤醒条件。在地址位多处理器模式中，RXWAKE 反映了保存 SCIRXBUF 寄存器中的地址位的值。在空闲线多处理器模式中，如果检测到 SCIRXD 数据线空闲就置位 RXWAKE。该位可通过下列方式之一清除。

- 在地址字节送至SCIRXBUF 后传送第1字节；
- 读 SCIRXBUF 寄存器；
- 有效的SW RESET；
- 系统复位。

位 0 保留位。

接收数据缓冲寄存器(SCIRXEMU 和 SCIRXBUF) 接收的数据从 RXSHF 传送到接收数据缓冲寄存器中。当传送操作完成时，RXRDY 标志位置位，这表明接收到的数据已经准备好。两个寄存器中存放着相同的数据。它们有分开的地址但在物理上并不是分开的缓冲器。其区别是：SCIRXEMU 寄存器主要是由仿真器(EMU) 使用，读SCIRXEMU 操作并不清除RXRDY 标志位，而读 SCIRXBUF 操作会清除该标志位。

7. 串行通信接口接收数据缓冲寄存器(SCIRXEMU)——地址7056 h

7	6	5	4	3	2	1	0
ERXDT7	ERXDT6	ERXDT5	ERXDT4	ERXDT3	ERXDT2	ERXDT1	ERXDT0
R0	R_0	R_0	R0	R0	R0	R0	R0

8. 串行通信接口接收数据缓冲寄存器(SCIRXBUF)——地址7057 h

7	6	5	4	3	2	1	0
RXDT7	RXDT6	RXDT5	RXDT4	RXDT3	RXDT2	RXDT1	RXDT0
R_0	R_0	R_0	R_0	R_0	R_0	R_0	R_0

发送缓冲寄存器(SCITXBUF) 发送的数据被写入到发送数据缓冲寄存器中。数据从该寄存器传送到发送移位寄存器(TXSHF) 的操作将置位TXRDY 标志位，这表明可向SCITX-BUF 寄存器写入新数据。

9. 串行通信接口发送数据缓冲寄存器(SCITXBUF) ——地 址7059 h

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

10. 串行通信接口优先级控制寄存器(SCIPRI ——地址705 Fh

7	6	5	4	3	2~0		
保留位	SCITX PRIORITY	SCIRX PRIORITY	SCI SOFT	SCI FREE	保 留 位		
R_0	RW_0	RW_0	RW_0	RW_0	R_0		

- 位 7
- 保留位。
- 位 6
- SCITX PRIORITY。SCI 发送中断优先级选择位。

0

高优先级中断请求；

1

低优先级中断请求。
- 位 5
- SCIRX PRIORITY。SCI接收中断优先级选择位。

0

高优先级中断请求；

1

低优先级中断请求。
- 位4~3
- SCI SOFT 和 FREE。SCI 仿真挂起选择位。

00

一旦仿真挂起，立即停止；

10

一旦仿真挂起，在完成当前的接收/发送操作后停止；

01

操作不受仿真挂起影响；

11

操作不受仿真挂起影响。
- 位2~0
- 保留位。

12.4 串行通信接口典型应用举例

下面的设计是用F2407 的串行通信接口与RS-232 串行口进行 DSP 与 PC 机之间的异步通信。RS-232 是美国电子工业协会(EIA) 在20世纪制定的一种串行总线的物理接口标准。此标准规定了串行传输中，主控模板与从属模块之间的物理连接线路的机械、电气、功能和过程特性，两端都必须遵循的共同约定。 RS-232 标准总线为25线，但在实际应用中常用

其简化了的9线接口。

由于上位机PC 都带有 RS-232 接口，所以可利用上位机的串行口与下位机(测控单元)进行 RS-232 通信，以完成上位机与下位机之间的数据交换，实现计算机对生产现场的监测和控制。

1. 串行通信的硬件电路设计

图12.9是 F2407 串行通信接口电路，该电路采用了符合 RS-232 标准的驱动芯片 TCS232，进行串行通信。TCS232 芯片功耗低，集成度高，+5 V 供电，具有两个接收和发送通道。整个接口电路简单，可靠性高。

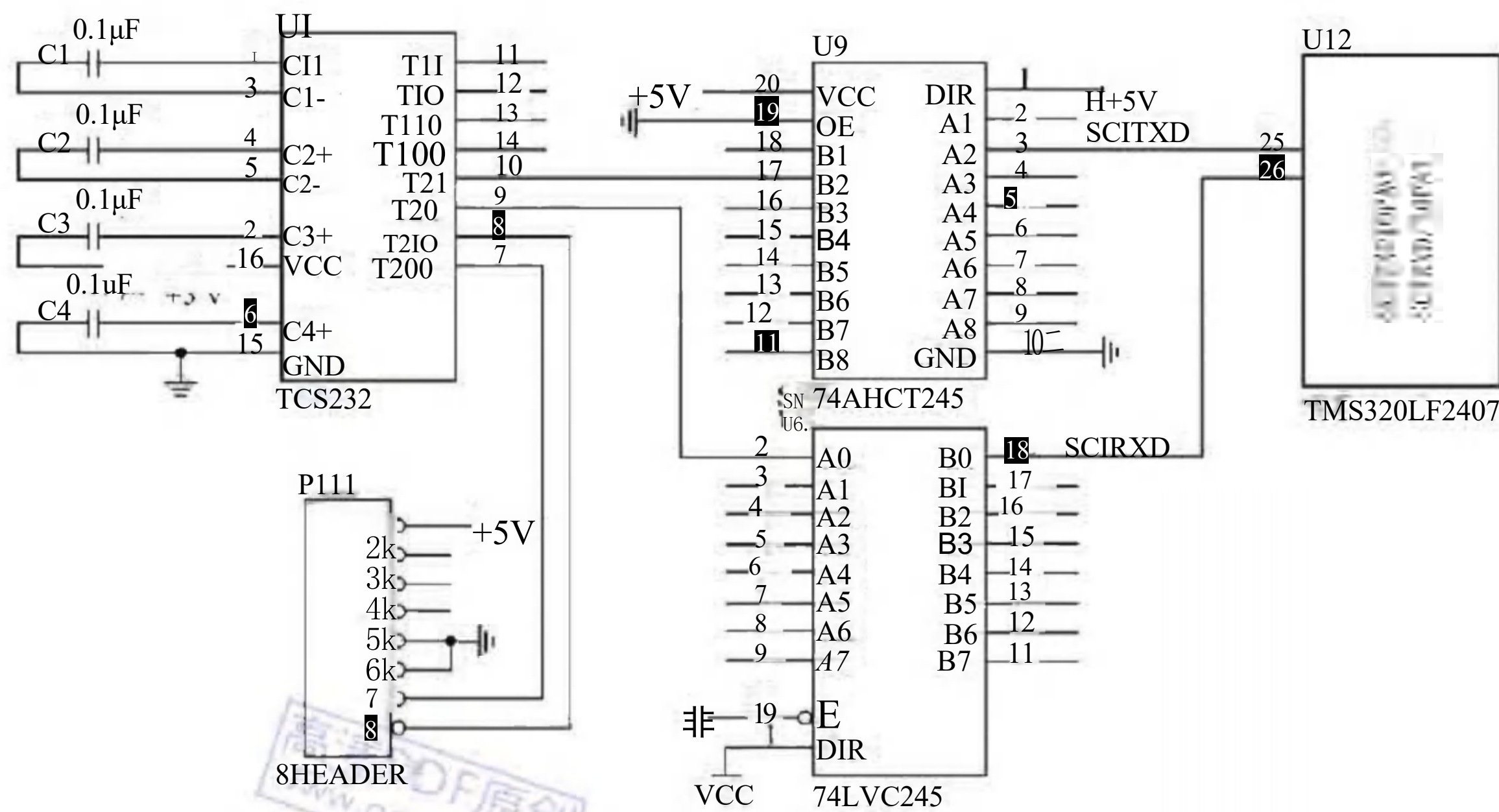


图12.9 F2407 与 TCS232 接口电路

2. 串行通信的软件设计

F2407 串行通信的软件设计可以采用查询和中断两种不同的方式，其中查询方式是在查询相应标志位(如发送寄存器空标志)成立，再执行相应的动作(如发送一个字节)。这种工作方式是在串行口与接口电路之间交换数据、状态和控制3种信息。它使DSP 陷于等待和反复查询之中，其 DSP 的利用率受到严重影响。现采用改进的查询方法，即当要上传数据时，在中断程序或其他的子程序中置发送标志位，在主程序中查询该标志。如成立则进行发送数据；否则跳过发送程序，执行其他的程序。在中断方式下，DSP 启动串行口后，就不再询问其状态，依然执行自己的程序(主程序)，实现 DSP 与串行口的并行工作。当串行口产生中断时，先向 DSP 申请中断，DSP 响应中断后，就暂时中断自己的程序，执行相应的串口中断服务程序，执

行完后又返回主程序，它能使信息得到及时处理。

要正确实现 DSP 与上位机的通信，除一般串行通信设置外，还必须正确设置 SLEEP 位 (SCICTL1.2)，方法如下：上电复位时，将所有参与多机通信的 DSP 的 SLEEP 位都设成1，使它们仅在检测到地址字节时才被中断。在中断服务程序里，将接收到的地址与相应软件设置的地址(本机地址 AB) 进行比较，若相同，则程序清除 SLEEP 位确保串行通信接口在收到每个数据字节时都产生一个中断；否则 SLEEP 位保持1以接收下一个地址。

(1) 汇编语言程序设计

下面给出地址位唤醒模式实现与上位机通信的 DSP 多机通信程序。该程序设置每个 DSP 有两个地址，如本例中 DSP 的地址为AAh 和 ABh，它们分别作为上位机控制该 DSP 发送和接收的命令。即若上位机以地址 AAh 呼叫，则下位机收到地址AAh 后，就发送N 个数据给上位机；若上位机以地址 ABh 呼叫，则下位机收到地址 ABh 后，就置 SLEEP 位为0，等待接收上位机下传的数据。

下面的程序是DSP 与上位机的通信程序，采用中断方式接收数据并置软件发送标志，在主程序中查询标志位，判断上位机是否需要 DSP 上传数据或 DSP 主动上传数据。 DSP 的时钟频率为24 MHz，串行通信的波特率为9600 b/s,8 位数据，1位停正位，不设奇偶校验位。

以下是源程序代码：

```
SCI_FLAG      .usect      "data0",1          ;SCI 标志寄存器
TXDPTR        .usect      ".data0",8          ;发送的数据存放区
RXD_PTR       .usect      ".data0",8          ;接收到的数据存放区
               .include    "F2407REGS.H"       ;引用头部文件
               .def         c_int0
; ①建立中断向量表
               .sect       ".vectors"          ;定义主向量段
RSVECT        B          c_int0               ;PM 0    复位向量      1
INT1           B          GISR1                ;PM 2    中断优先级1    4
INT2           B          PHANTOM              ;PM 4    中断优先级2    5
INT3           B          PHANTOM              ;PM 6    中断优先级3    6
INT4           B          PHANTOM              ;PM 8    中断优先级4    7
INT5           B          PHANTOM              ;PM A    中断优先级5    8
INT6           B          PHANTOM              ;PM C    中断优先级6    9
RESERVED       B          PHANTOM              ;PM E    (保留位)     10
SWINT8         B          PHANTOM              ;PM 10   用户定义软件中断  -
               :
SW_INT31       B          PHANTOM              ;PM 3E   用户定义软件中断
;中断子向量入口定义 pvecs
```

```

                .sect  ".pvecs"                ;定义子向量段
PVECTORS      B      PHANTOM                ;保留向量地址偏移量  00h
                B      PHANTOM                ;保留向量地址偏移量  01h
                *
                *
                B      PHANTOM                ;保留向量地址偏移量  05h
                B      SCI_RX_ISR            ;保留向量地址偏移量  06h,SCI 接收中断
                B      PHANTOM                ;保留向量地址偏移量  07h
                *
                *
                B      PHANTOM                ;保留向量地址偏移量  41h

```

;②主程序

```

                .text
c      int0
                SETC      INTM
                CLRC      SXM
                CLRC      OVM
                CERC      CNF
                LDP        #0EOH
                SPLK       #81FEH,SCSR1        ;CLKIN=6   MHz,CLKOUT=24   MHz
                SPLK       #0E8h,WDCR         ;关看门狗
                CALL       SCI_INIT           ;调串口初始化程序
                LDP        #5H
                SPLK       #00H,SCI_FLAG       ;清发送软件标志位
TXD_LOOP:
                LDP        #5H
                BIT        SCI_FLAG.BITO
                BCND       TXD_DATA.TC        ;SCI_FLAG.0=1 则发送数据
                B          WAIT
TXDDATA:
                ;发送 N 个数据程序
                MAR        *,ARO
                LAR        ARO,#TXD_PTR
                ADRK       #N+1                ;发送 N 个数据
                MAR        兴. AF
                LACC       若 ~
                CMPR       00
                BCND       TXDDATAEND,1C
                MAR        *,AR2
                SACL       ,AR4                ;数据送 SCITXBUF 寄存器

```

```
XMIT_RDY:
    LDP      #DP_PF1
    BIT      SCICTL2,BIT7
    BCND     XMIT_RDY,NTC           ;判断发送器是否空
    B        TXD_DATA
TXDDATA_END:
    MAR      *,AR4
    LAR      AR4,#TXD_PTR           ;恢复发送的数据指针
    LDP      #5H
    SPLK     #00H,SCIFLAG           ;清发送软件标志位
WAIT:      NOP
    B        TXD_LOOP
;③串行通信初始化程序
SCI_INIT:
    LDP      #OE1H
    LACL     MCRA
    OR       #03H
    SACL     MCRA                   ;配置串行口引脚为特殊功能： TXD、RXD
    LDP      #DP_PF1
    SPLK     #000FH, SCICCR         ;地址位唤醒模式，8位数据，1位停止位，无奇偶校验
    SPLK     #0007H, SCICTL1       ;接收，发送，内部时钟使能，SLEEP=1
    SPLK     #0002H, SCICTL2       ;接收中断使能
    SPLK     #0001H,SCIHBAUD
    SPLK     #0038H, SCILBAUD      ;波特率为9600
    SPLK     #0027H,SCICTL1        ;串口初始化完成
    LAR      AR1,#SCIRXBUF         ;接收缓冲寄存器地址
    LAR      AR2,#SCITXBUF         ;发送缓冲寄存器地址
    LAR      AR3,#RXD_PTR          ;接收的数据指针
    LAR      AR4,#TXD_PTR          ;发送的数据指针
    LDP      #0
    SPLK     #0001h,IMR            ;允许 INT1中 断
    SPLK     #OFFFh,IFR            ;清所有中断标志
    CLRC     INTM                  ;开总中断
    RET
;④中断程序
GISR1:
    ;保护现场
    LDP      #OE0H
    LACC     PIVR,1                ;读取外设中断向量寄存器(PIVR)，并左移1位
```



```

ADD    ,PVECTORS           ;加上外设中断入口地址
BACC    ;跳到相应的中断服务子程序
SCI_RX_ISR:                ;接收中断服务程序
LDP     #DP_PTR
MAR     ,AR1
BIT     SCICTL1.BIT2
BCND    RXD_DATA,NTC       ;SLEEP=0, 接收数据
LACC    关                 ;读入地址
SUB     #0AAH
BCND    TXD_FLAG,EQ        ;接收到的地址为AAh,则发送数据
SUB     #0001H
BCND    SCI_RX_END,NEQ     ;地址错误, 退出中断
SPLK    #0033H,SCICTL1    ;地址为ABh,则清 SLEEP=0 表示准备接收数据
B       SCI_RX_END
TXD_FLAG:                  ;置发送标志
LDP     #5H
SPLK    #01H.SCI_FLAG
B       SCIRX_END
RXD_DATA:                  ;接收 N 个数据
LACC    关, AR3            ;读入数据
SACL    *+,ARO             ;保存数据
LAR     ARO,#RXDPTR
ADRK    #N
MAR     *,AR3
CMPR    00
BCND    SCI_RX_END,NTC     ;判断是否接收完
LAR     AR3,#RX_PTR       ;恢复接收的数据指针
SPLK    #0037H, SCICTL1   ;置SLEEP=1 表示准备接收地址
SCI_RX_END:               ;中断返回
    ;恢复现场
CLRC    INTM              ;开总中断, 因为一进入中断就自动关闭总中断
RET
;⑤假中断程序
PHANTOM
KICK_DOG           ;复位看门狗
RET
END

```

通过实际运行表明, RS-232 与 F2407 接口电路简单. 性能可靠. 该下位机的程序只要改变其地址如为: AAh 和 BBh, 就可以用于其他的 DSP 模块中, 实现上位机与DSP 模块之间的

多机通信。 RS-232 串行通信适用于短距离的数据传送。

2)C 语言程序设计

下面给出在两个 DSP 之间实现简单异步通信的通用程序。

//该程序实现F2407 的串口异步通信

```
#include "register.h"
```

//禁止总中断子程序

```
void inline disable()
```

```
{
```

```
    asm("setc INTM");
```

```
    asm("setc SXM");
```

```
}
```

//使能总中断子程序

```
void inline enable()
```

```
{
```

```
    asm("clrc INTM");
```

```
}
```

```
main()
```

```
{
```

```
    disable();
```

//禁止所有中断

```
    IFR=0xFFFF;
```

//清除中断标志

```
    SCSR1=0x81FE;
```

//CLKIN=6 MHz,CLKOUT=24 MHz

```
    WDCR=0xE8;
```

//不使能看门狗

```
    SCICCR=0x7;
```

//1个停止位，不使能奇偶校验，空闲线多处理

//器模式，8位字符

```
    SCICTL1=0x13;
```

//使能接收和发送，SLEEP=0 禁止休眠方式

//禁止接收错误中断，TXWAKE=0 即没有选定的发送特征

```
    SCICTL2=0x3;
```

//使能接收和发送中断

```
    SCIHBAUD=0x2;
```

```
    SCILBAUD=0x70;
```

//波特率=4800 b/s

```
    SCICTL1=0x33;
```

//使SCI 脱离复位状态

```
    SCIPRI=0x60;
```

//SCI 中断(接收和发送中断)为低优先级中断

```
    MCRA=0x3;
```

```
    PADATDIR=0x100;
```

```
    IMR=0x10;
```

//使能UART 中断 INT5

```
    enable();
```

//使能总中断

```
    SCITXBUF=;
```

```
    while(1);
```

```
}
```

```
void UartSent()
```

//发送服务程序

```
const char*var="F2407 UART is fine!";//定义一段需要发送的字符串

static int i=0;
if(i>strlen(var)) return; //如果需要发送的字符都已经发送完，则中断直接返回
    SCITXBUF=var[i++]; //依次发送定义的字符串中的各个字符
    *IFR=0x0010; //清除 IFR 中相应的中断标志
    enable(); //开总中断，因为一进入中断服务程序总中断就自动关闭

void UartRec () //接收服务程序

{
    static int receive[10],j=0;
    receive[j++]=*SCIRXBUF; //依次接收字符
    if(j>9)j=0;
    *IFR=0x0010; //清除 IFR 中相应的中断标志
    enable(); //开总中断，因为一进入中断服务程序总中断就自动关闭
}

void interrupt uarttr() //中断服务程序

{
    switch ( 关 PVIR) //根据中断向量寄存器 PVIR 的值区别是接收还是发送中断

    {
        case 6:UartRec(); //如果PIVR=6，则发生接收中断，执行接收服务程序
        case 7:UartSent(); //如果PIVR=7，则发生发送中断，执行发送服务程序
    }

    //当由于干扰而引起其他中断时.中断进入此程序直接返回主程序
    void interrupt nothing()
    {
        return;
    }
}
```


第 13 章 CAN 控制器模块

13.1 CAN 控制器模块概述

13.1.1 CAN 技术简介

CAN(Controller Area Network)即控制器局域网，主要用于各种设备监测及控制的一种网络。CAN 最初是由德国 Bosch 公司为汽车的监测、控制系统而设计的。CAN 具有独特的设计思想，良好的功能特性和极高的可靠性，现场抗干扰能力强。具体来讲，CAN 具有如下特点：

- >结构简单，只有两根线与外部相连，且内部含有错误探测和管理模块。
- >通信方式灵活。可以多主方式工作，网络上任意一个节点均可以在任意时刻主动地向网络上的其他节点发送信息，而不分主从。
- >可以点对点、点对多点及全局广播方式发送和接收数据。
- >网络上的节点信息可分成不同的优先级，可以满足不同的实时要求。
- >CAN 通信格式采用短帧格式，每帧字节数最多为8个，可满足通常工业控制领域中控制命令、工作状态及测试数据的一般要求。同时，8个字节也不会占用总线时间过长，从而保证了通信的实时性。
- >采用非破坏性总线仲裁技术。当两个节点同时向总线上发送数据时，优先级低的节点主动停止数据发送，而优先级高的节点可不受影响地继续传输数据，这大大地节省了总线仲裁冲突时间，在网络负载很重的情况下也不会出现网络瘫痪。
- >直接通信距离最大可达10 km(速率5 kb/s 以下)，最高通信速率可达1 Mb/s (此时距离最长为40 m)。节点数可达110个，通信介质可以是双绞线、同轴电缆或光导纤维。
- >CAN 总线通信接口中集成了CAN 协议的物理层和数据链路层功能，可完成对通信数据的成帧处理，包括位填充、数据块编码、循环冗余检验、优先级判别等工作。
- >CAN 总线采用CRC 检验并可提供相应的错误处理功能，保证了数据通信的可靠性。

CAN 总线具有以上的一些特点，为工业控制系统中高可靠性的数据传送提供了一种新的解决方案。其在国外工业测控领域已经有了广泛的应用，现国内的许多工业控制领域也开始

使用基于CAN 的现场总线。CAN 总线已成为最有发展前途的现场总线之一。

13.1.2 TMS320LF240x 系列CAN 控制器概述

CAN 控制器模块是一个完全的 CAN 控制器，它是一个16位的外设模块，具有以下特性：

完全支持CAN2.0B 协议：

—标准和扩展标识符；

—数据帧和远程帧。

对象有6个邮箱，其数据长度为0~8字节：

—2个接收邮箱(MBOX0、1),2 个发送邮箱(MBOX4、5)；

—2个可配置为接收或发送邮箱(MBOX2、3)。

对邮箱0、1和2、3有局域接收屏蔽寄存器(LAMn)。

> 可编程的位定时器。

> 中断配置可配置。

可编程的CAN 总线唤醒功能。

> 自动回复远程请求

> 当发送时出现错误或仲裁时丢失数据. CAN 控制器有自动重发送功能。

> 总线错误诊断功能。

> 自测试模式。

CAN 控制器在自测试模式下，接收邮箱接收CAN 自身的发送邮箱发送的信息帧，并产生自应答信号。

CAN 控制器的结构框图如图13.1所示。CAN 控制器必须通过CAN 驱动芯片才能与其他的CAN 控制器进行通信。

CAN 模块是一个16位的外设，对它的访问分成控制/状态寄存器的访问和邮箱的RAM 访问。

邮箱位于一个48×16位的RAM 中，它可被CPU 或 CAN 模块读/写。CAN 读/写访问和CPU 读访问需要一个时钟周期，而CPU 的写访问需要两个时钟周期。在这两个时钟周期里，CAN 完成读——修改——写循环，因此CPU 插入了一个等待状态。

当访问RAM 时，地址总线的第0位用来决定取32位字的低16位字(地址总线的第0位为0)还是高16位字(地址总线的第0位为1)。RAM 的位置由地址总线的第1位到第5位决定。

CAN 控制器模块的内存空间分配如图13.2所示。

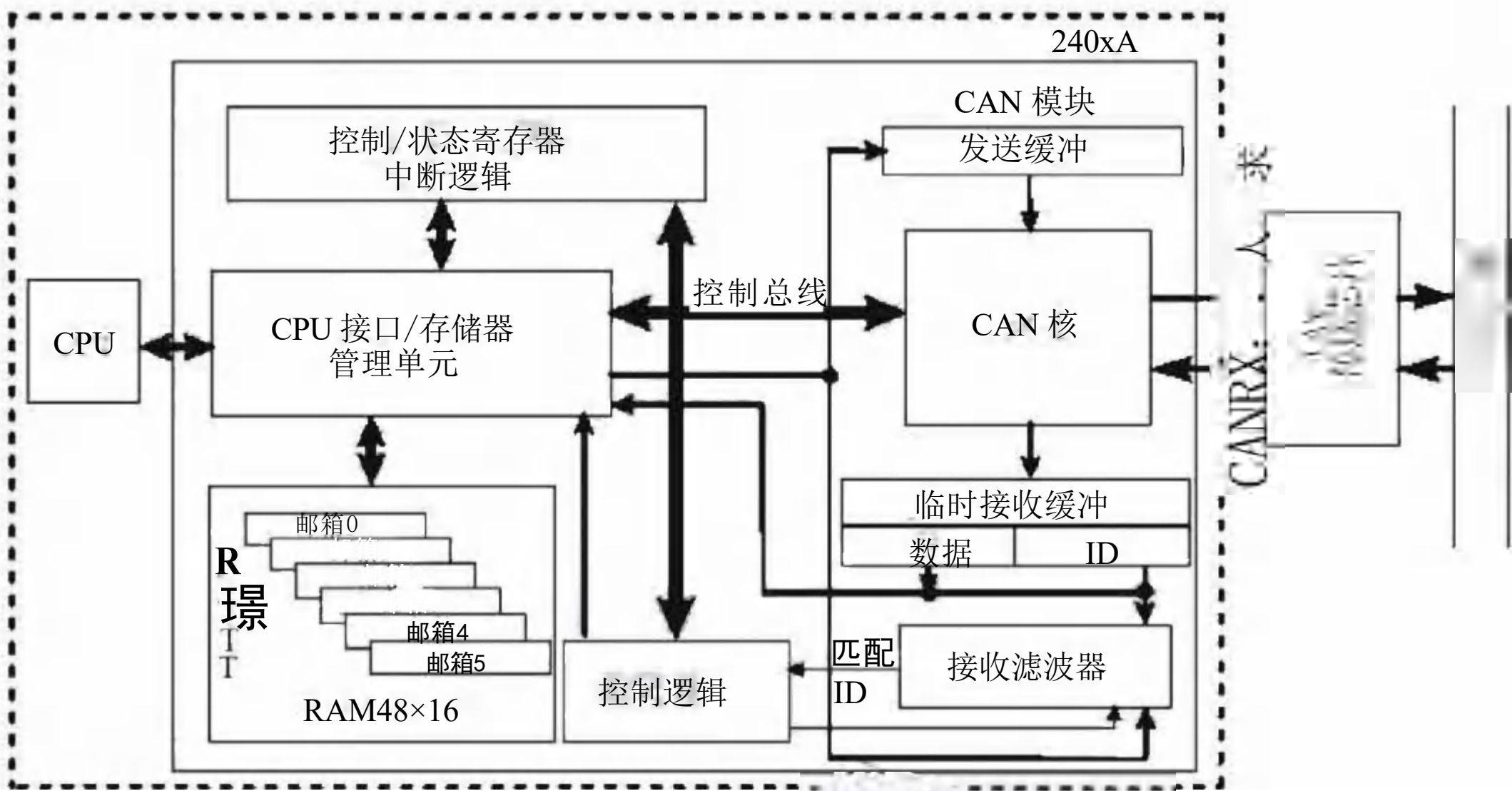


图13.1 CAN 控制

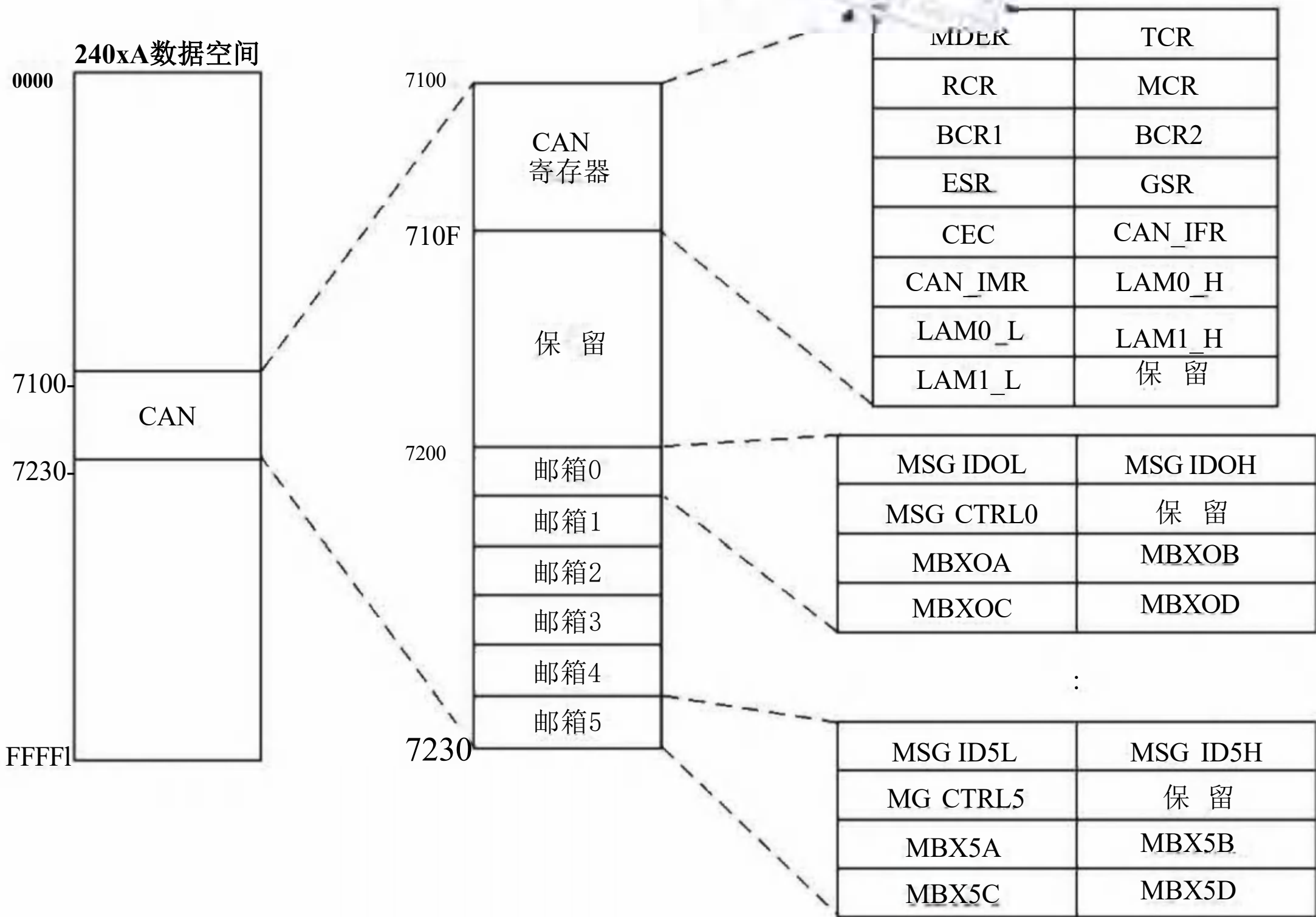


图13.2 CAN 控制器内存空间分配图

13.2 邮 箱

13.2.1 CAN 信息包格式说明

一个有效的CAN 的数据帧由帧起始、仲裁域、控制域、数据域、校验域、应答域和帧结束组成。

F2407 的 CAN 控制器支持两种不同的帧格式，即标准格式和扩展格式。其主要区别在于仲裁域格式不同：标准帧仲裁域由11位标识符和远程发送请求位 RTR 组成；扩展帧仲裁域由29位标识符和替代远程请求 SRR 位、标志位和远程发送请求位RTR 组成。CAN 信息帧如图13.3所示。

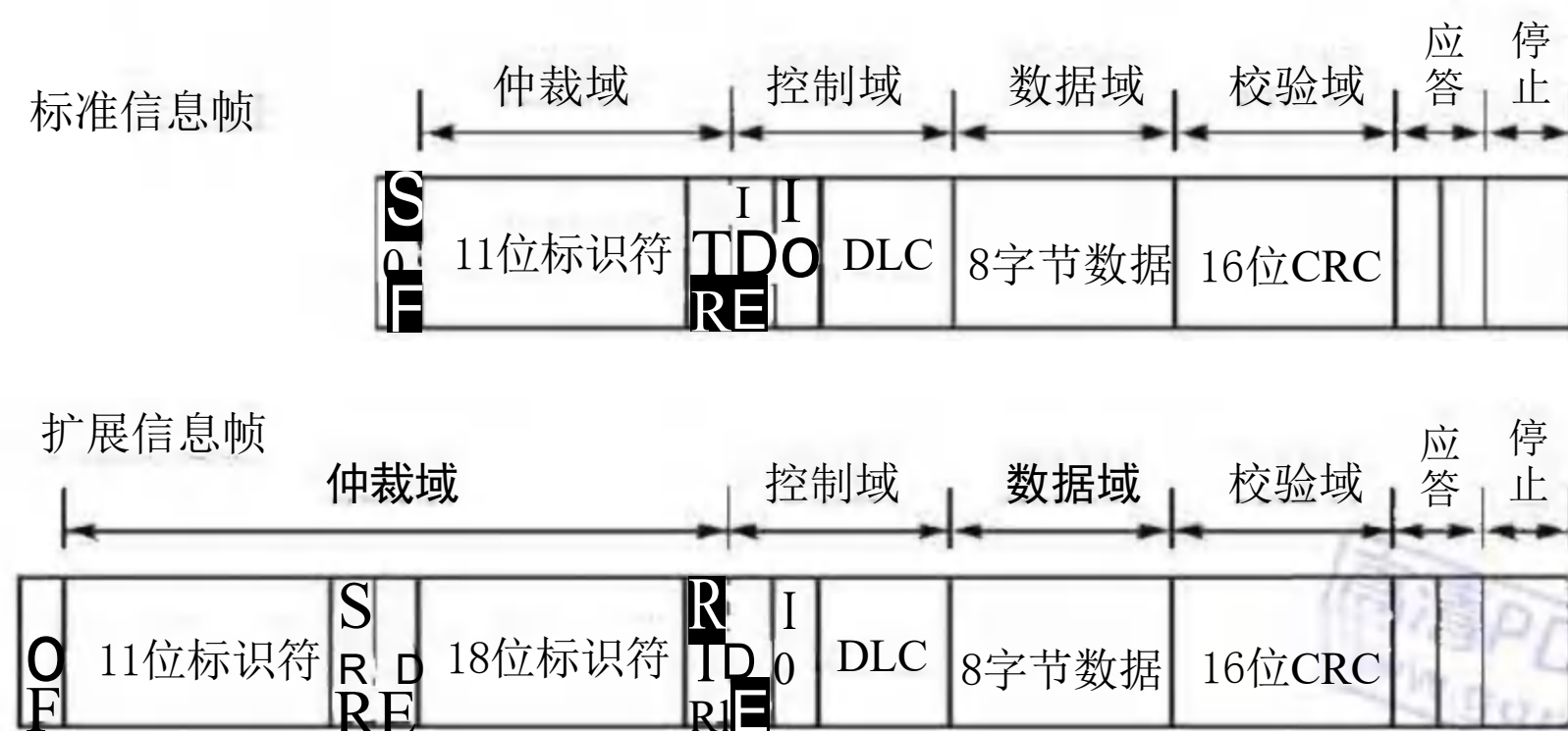


图13.3 CAN 信息帧

标识符是作为报文的名称，在仲裁过程期间，它首先被送到总线。在接收器的验收判断中和仲裁过程确定访问优先权中都要用到。

远程发送请求位 RTR 用来确定是发送远程帧还是数据帧：当RTR 为高电平时，CAN 控制器发送远程帧；为低电平时则发送数据帧。

数据长度码DLC 用来确定每帧要发送几个字节的数据，最多为8字节。

13.2.2 CAN 邮箱寄存器

邮箱寄存器地址如表13.1所列。每个邮箱由邮箱标识寄存器、邮箱控制寄存器及 4×16 位的存储空间组成。

这些空间用于存储发送的数据帧或接收到的数据帧，每个最大可存储8字节数据(MBX-nA、MBXnB、MBXnC和 MBXnD)。当它们不用作邮箱的信息存储空间时，可用作一般的RAM 空间供CPU 使用。

表13.1 邮箱寄存器地址列表

寄存器	邮 箱					
	MBOX0	MBOX1	MBOX2	MBOX3	MBOX4	MBOX5
MSG IDnL	7200	7208	7210	7218	7220	7228
MSG IDnH	7201	7209	7211	7219	7221	7229
MSG CTRLn	7202	720A	7212	721A	7222	722A
保留位						
MBXnA	7204	720C	7214	721C	7224	722C
MBXnB	7205	720D	7215	721D	7225	722D
MBXnC	7206	720E	7216	721E	7226	722E
MBXnD	7207	720F	7217	721F	7227	722F

1. 邮箱标识符

每个邮箱都有各自独立的邮箱标识符，它们存储在两个16位的寄存器(MSGIDnH 和 MSGIDnL) 中。

(1) 邮箱标识符高位寄存器 (MSGIDnHD)——n=0~5

15	14	13	12~0
IDE	AME	AAM	IDH[28:16]
RW	RW	RW	RW

注：R= 可读；W= 可写。

- 位15

IDE。标识符扩展位。
0 用标准标识符；
1 用扩展标识符。
- 位14

AME。接受屏蔽使能位。这一位只与接收邮箱有关，对发送邮箱无影响。
0 禁止相应的标识符屏蔽，即接收邮箱的标识符必须与被接收的邮箱标识符相符才能接收信息；
1 使能相应的标识符屏蔽，即接收邮箱可以接收与它的标识符不符的信息。
- 位13

AAM。自动应答模式位。这一位只与配置为发送邮箱的2和3有关。
0 邮箱不自动应答远程帧请求；
1 如果接收到一个标识符匹配的远程帧请求，则CAN 外设将发送邮箱中的数据。



位12~0 IDH[28~16]。 标识符。相对于扩展信息帧的高13位标识符；IDH[28~16]即 MSGIDnH[12~0] 位相对于标准信息帧的11位标识符。

2) 邮箱标识符低位寄存器 (MSGIDnL)——n=0~5

IDL[15:0]
RW

位15~0 IDL[15~07]。 标识符。相对于扩展信息帧的低16位标识符；与标准信息帧无关。

2. 邮箱控制寄存器

每个邮箱都有一个自己的控制域，即决定信息是远程帧还是数据帧，及每帧的字节数。邮箱控制寄存器 (MSGCTRLn)——n=0~5

15-5	4	3~0
保留位	RTR	DLC[3:0]
	RW	RW

位15~5 保留位。

位 4 RTR。 远程发送请求位。

0 数据帧；

1 远程帧。

位3~0 DLC[3~0]。 数据长度选择位。对于发送邮箱来说，这几位决定发送数据的字节数；对于接收邮箱则无效，即接收数据的字节数由被接收的信息决定。

0001	1 字 节	0010	2 字 节
0011	3 字 节	0100	4 字 节
0101	5 字 节	0110	6 字 节
0111	7 字 节	1000	8 字 节

13.3 CAN 控制寄存器

CAN 控制器总共有15个16位的控制寄存器，如表13. 2所列。这些寄存器控制着CAN的位定时器、邮箱的发送或接收使能、错误状态及CAN 的中断等。

邮箱方向/使能控制寄存器(MDER)， 它决定邮箱的使能位(ME) 和 邮 箱 2 和 邮 箱 3 的 方 向，即是配置为发送邮箱还是接收邮箱。

表13.2 CAN 控制寄存器地址列表

地 址	名 称	描 述
7100h	MDER	邮箱方向/使能控制寄存器(位7~0)
7101h	TCR	发送控制寄存器(位15~0)
7102h	RCR	接收控制寄存器(位15~0)
7103h	MCR	主控制寄存器(位13~6, 1. 0)
7104h	BCR2	位定时器配置寄存器2(位7~0)
7105h	BCR1	位定时器配置寄存器1(位10~0)
7106h	ESR	错误状态寄存器(位8~0)
7107h	GSR	全局状态寄存器(位5~3, 1. 0)
7108h	CEC	CAN错误计数寄存器(位15~0)
7109h	CAN_IFR	中断标志寄存器(位13~8. 6~0)
710Ah	CAN_IMR	中断屏蔽寄存器(位15. 13~0)
710Bh	LAM0_H	对于MBOX0和I的局部接收屏蔽高位寄存器(位31, 28~16)
710Ch	LAM0_I	对于MBOX0和1的局部接收屏蔽低位寄存器(位15~0)
710Dh	LAM1_H	对于MBOX2和3的局部接收屏蔽高位寄存器(位31, 28~16)
710Eh	LAM1_I	对于MBOX2和3的局部接收屏蔽低位寄存器(位15~0)
710Fh	保留位	用以声明来自CAN外设的CAADDRx信号(可以声明非法的地址错误)

1. 邮箱方向/使能控制寄存器(MDER) —— 地 址 7100 h

15~8

保留位

7	6	5	4	3	2	1	0
MD3	MD2	ME5	ME4	ME3	ME2	ME1	ME0

RW_0 RW_o RW_0 RW_o RW_0 RW_0 RW_0 RW_0

位15~8 保留位。

位 7 MD3 。 邮箱3发送/接收配置位。上电时该位复位为0。

- 0 配置成发送邮箱；
- 1 配置成接收邮箱。

位 6 MD2 。 邮箱2发送/接收配置位。上电时该位复位为0。

- 0 配置成发送邮箱；
- 1 配置成接收邮箱。

- 位5~0 ME5~ME0。邮箱使能位。每个邮箱对应一个使能位，在初始化时必须禁止相应邮箱的使能位。
- 0 禁止邮箱；
- 1 使能邮箱。

发送控制寄存器(TCR)，该寄存器控制着邮箱的发送及状态，对发送请求的设置和复位位可以进行独立的写操作。该寄存器对发送邮箱有效，即邮箱4、5及被配置为发送方式的邮箱2和邮箱3。寄存器中的 TA5、AA5、TRS5和 TRR5 为邮箱5的发送控制位，其他的类似。上电后该寄存器的全部位清0。

2. 发送控制寄存器(TCR)——地址7101h

15	14	13	12	11	10	9	8
TA5	TA4	TA3	TA2	AA5	AA4	AA3	AA2
RC_0	RC_0	RC0	RC	RC0	RC0	RC_0	RC_0
7	6	5	4	3	2	1	0
TRS5	TRS4	TRS3	TRS2	TRR5	TRR4	TRR3	TRR2
RS_0	RS0	RS_0	RS_0	RS_D	RS_D	RS_0	RS_0

注：R= 可读；C= 可清除；S= 只可被置位；0=复位值。

- 位15~12 TA5~TA2。发送应答位。如果邮箱 n(n=5~2) 发送信息帧成功，则TAn 置1, 并且置位邮箱中断标志位 MIFn, 在相应中断使能的情况下将产生中断。向该位写1则复位，写入0无影响。
- 位11~8 AA5~AA2。忽略应答位。如果发送的信息帧被忽略，则AAn 被置位，并且置位邮箱忽略应答中断标志位 AAIFn, 在相应中断使能的情况下将产生邮箱错误中断。向该位写1则复位，写入0无影响。
- 位7~4 TRS5~TRS2。邮箱发送请求位。当TRS_n 位被置位1时，CAN 控制器将发送相应发送邮箱的信息帧。
- 当该位被置位1后，则拒绝对邮箱 n 的信息帧进行更新，直到该位被复位。CAN 控制器允许不同的邮箱同时设置该位。
- 当 TRS_n 位被置位1时，对邮箱n 进行写操作无效，并且会产生 WDIF 中断。当发送成功或发送被忽略时，就自动复位该位。
- 位3~0 TRR5~TRR2。发送请求复位位。用户程序将设置该位，中断逻辑将清除该位，用户程序向该位写1 将设置该位，写0无影响。
- 当TRR_n 位被置位1时，对邮箱n 进行写操作无效，并且会产生 WDIF 中断；

当成功发送后，则产生邮箱中断(在相应中断使能的条件下)。

当信息成功发送、发送忽略丢失数据和侦测到CAN 总线错误时，将复位该位。

当 TRSn 处于复位时，该位不起作用，因为当 TRSn 复位，TRRn 也立即复位。

接收控制寄存器(RCR)，该寄存器控制接收到的信息状态和远程帧处理。该寄存器对接收邮箱有效，即邮箱0、1及被配置为接收方式的邮箱2和邮箱3。

3. 接收控制寄存器(RCR)——地址7102h

15	14	13	12	11	10	9	8
RFP3	RFP2	RFP1	RFPO	RML3	RML2	RML1	RML0
RC_0	RC_0	RC_0	RC0	R_0	R_0	R_0	R_0
7	6	5	4	3	2	1	0
RMP3	RMP2	RMP1	RMPO	OPC3	OPC2	OPC1	OPC0
RC_0	RC_0	RC_0	RC_0	RW_0	RW_0	RW_0	RW_0

位15~12 RFP3~RFPO。 远程请求悬挂位。

无论何时 CAN 外设接收到远程帧请求时，将把相应接收邮箱 n 的 RFPn 置位。

如果用户程序要清除该位，同时CAN 外设要设置该位，则该位被清除。

如果 MSGIDnH 寄存器中的 AAM 位没有被置位(即不具有自动发送应答信号功能)，则用户程序必须在远程请求悬挂事件发生后清除该位。

如果信息被成功发送，则CAN 外设清除该位。在发送过程中不能产生远程请求中断。

位11~8 RML3~RMLO。 接收到的信息丢失标志位。

当接收邮箱n 的旧信息被新接收到的信息覆盖时，该位将置位。如果覆盖保护控制位 OPCn 为1, 则不发生信息覆盖，且新的信息将丢失。

该位只能由用户程序复位，并且由中断逻辑置位。当写1到RMPn 位，将清除该位。如果用户程序要清除该位，同时 CAN 外设要置位该位，则该位被置位。

如果1个或多个 RMLn 位被置位，同时将置位CAN 中断标志寄存器(CAN_IFR) 中的RMLIF 标志位。CAN 中断屏蔽寄存器(CAN_IMR) 中的 RMLIM 使能，则将产生中断。

位7~4 RMP3~RMPO。 接收信息悬挂位。

如果被接收的信息帧存储到接收邮箱 n 中，将把 RMPn 置位。

该位只能由用户程序复位，并且由中断逻辑置位。当写1到 RMPn 位，将清除 RMPn 和 RMLn 位。如果用户程序要清除该位，同时 CAN 外设要置位该位，则该位被置位。

如果 OPCn 位为0 • 则新信息将覆盖旧信息 • 并且相应的 RMLn 位被置位。

如果 CAN 中断使能且相应的中断没有被屏蔽，则 RMLn 位被置位的同时将产生中断。

位 3 ~ 0 OPC3~OPCO。 信息覆盖保护使能位。

如果 OPCn 位为1, 则邮箱n 中的旧信息被保护，即不能被新信息覆盖。从而查找下一个标识符匹配的接收邮箱，如不匹配则信息丢失。

如果 OPCn 位为0, 则邮箱n 中的旧信息被覆盖。

4. 主控制寄存器(MCR)—— 地址 7103 h

15~4		13	12	11	10	9	8
保留位		SUSP	CCR	PDR	DBO)	WUBA	CDR
		RW_0	RW_1	RW_0	RW_0	RW_0	RW_0
7	6	5~2				1~0	
ABO	STM	保留位				MBNR[1:0]	
RW_0	RW_0					RW_a	

位15~14 保留位。

位13 SUSP。 仿真挂起操作选择位。这一位对接收无影响。

0 Soft 模式，即一旦仿真挂起，就把当前的信息完全发送完毕才关闭CAN 外设；

1 Free 模式，即CAN 外设继续运行不受仿真挂起影响。

位12 CCR。 改变配置请求位。当CAN 控制器处于脱离 CAN 总线状态或ABO 位为0时，将把该位自动置位。当CAN 控制器恢复总线时，该位被清除。

0 CAN 控制器处于正常工作方式；

1 CAN 控制器处于复位工作方式，即在GSR 寄存器的 CCE 位为1时，允许对位定时器(BCRn) 进行配置，此时 CAN 控制器处于脱离 CAN 总线状态，因此当配置完位定时器后，应将该位清0, 使 CAN 控制器恢复总线。

位 1 1 PDR。 低功耗模式请求位。在进入休眠模式之前，用户程序必须将该位置为1, 当相应的应答位 PDA 被置位后，CAN 控制器就进入休眠模式。

- 0 禁止低功耗模式(CAN 控制器处于正常工作模式);
- 1 使能低功耗模式。
- 位10 DBO。数据字节次序。
 - 0 接收或发送的数据排列成以下的次序: 3-2- 1-0-7-6-5-4;
 - 1 接收或发送的数据排列成以下的次序: 0-1-2-3-4-5-6-7。
- 位 9 WUBA。总线唤醒位。
 - 0 只有在用户程序把 PDR 位清 0 后, CAN 控制器退出低功耗模式;
 - 1 当检测到CAN 总线上任何有效信息时, CAN 控制器就退出低功耗模式。
- 位 8 CDR。数据域改变请求位。当要改变邮箱的数据域时应把该位置为1, CAN 控制器就允许对数据域进行更改, 并且在此期间发送被禁止, 所以更新完数据域后应将该位清0。
 - 0 CAN 控制器处于正常工作模式;
 - 1 数据域改变请求使能。
- 位 7 ABO。自动恢复总线位。
 - 0 在 CCR 位被复位后和在总线上产生连续 128×11 个隐性位后, CAN 控制器恢复总线;
 - 1 在 CAN 控制器脱离总线后的连续 128×11 个隐性位之后, CAN 控制器恢复总线。
- 位 6 STM。自测试模式使能位。
 - 0 CAN 控制器处于正常工作模式;
 - 1 CAN 控制器处于自测试模式, 在这种模式下 CAN 控制器能自己产生应答信号, 因此不需要与 CAN 总线相连, 信息帧没有真正发送出去, 但是信息帧能被相应的邮箱接收, 在自测试模式下, 不能进行远程帧悬挂自动应答, 也不能接收被接收的信息帧的标识符。
- 位5~2 保留位。
- 位1~0 MBNR[1:0]。邮箱2和邮箱3选择位。对邮箱2或3的数据域进行写操作及配置远程帧悬挂。
 - 00 无效;
 - 01 无效;
 - 10 选择邮箱2;
 - 11 选择邮箱3。

位定时器配置寄存器(BCR1 和 BCR2), 这两个寄存器用于配置CAN 节点的时间参数。必须在CAN 控制器处于复位模式下(即CCR=1), 才能对位定时器进行配置。

5. 位配置寄存器2 (BCR2)——地址7104h



位15~8 保留位。

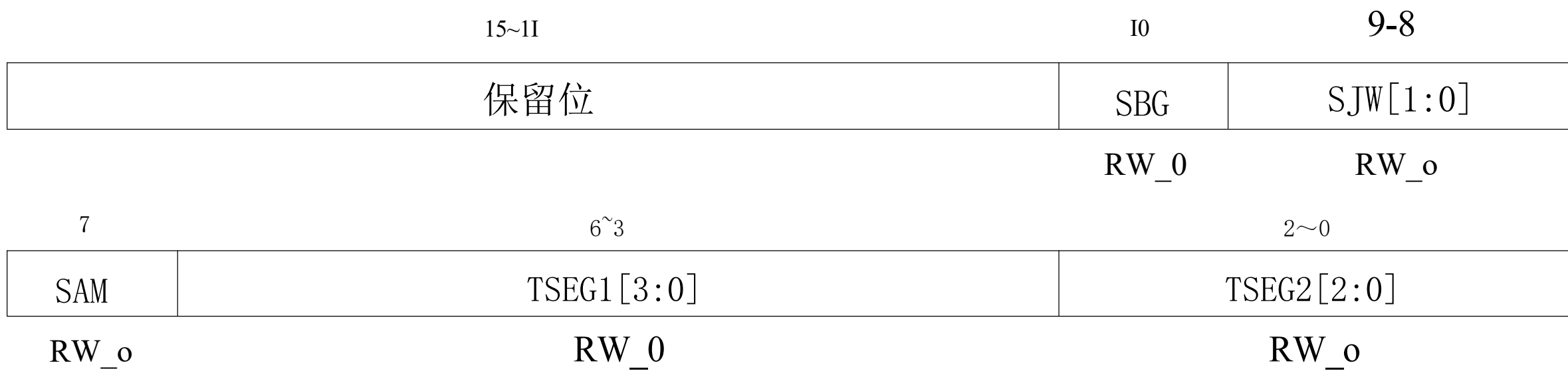
位7~0 BRP[7~0]。 波特率预分频位。

 BRP[7~0] 决定着CAN 控制器的时间片(TQ),TQ 定义为:

$TQ=(BRP+1)/IcLK$

这里 Ick 为 CAN 控制器的系统时钟，也就是DSP 的系统时钟。

6. 位配置寄存器 1 BCR1) ——地 址7105h



位15~11 保留位。

位10 SBG。 边沿重同步方式选择位

 0 CAN 控制器仅在下降沿时发生重新同步；

 1 CAN 控制器在上升沿和下降沿时发生重新同步。

位9~8 SJW[1:0]。 同步跳转宽度选择位。

 为了补偿在不同总线控制器的时钟振荡器之间的相位偏移，任何总线控制器必须在当前传送的相关信号边沿重新同步。同步跳转宽度定义了每一位周期可以被重新同步缩短或延长的时钟周期的最大数目，同步跳转宽度可设置为1~4个TQ 值。

位 7 SAM。 采样次数选择位。

 0 CAN 控制器仅采样1次；

 1 CAN 控制器采样3次，并以多数为准。

位6~3 TSEG1[3:0]。 时间段1。其包含传播延时时间段(PROG SEG)和相位延时

时间段 1(PHASE SEG1)。TSEG1[3~0]决定时间段 1 有多少个 TQ 时间片, 时间段 1 的值可编程为 3~16 个 TQ 时间片, 且时间段 1 必须大于或等于时间段 2。

位 2~0 TSEG2[2:0]。时间段 2。其决定着相位延时时间段 2(PHASE SEG2)的长度。当在下降沿时发生重新同步(即 SBG=0) 时, 时间段 2 的最小值为:

$$TSEG2_{min}=1+SJW$$

因此时间段 2 的值可编程为 2~8 个 TQ 时间片, 且满足以下条件:

$$(SJW+BSG+1)\leq TSEG2\leq 8$$

$$TSEG2\leq TSEG1$$

位定时器的时间段如图 13.4 所示。

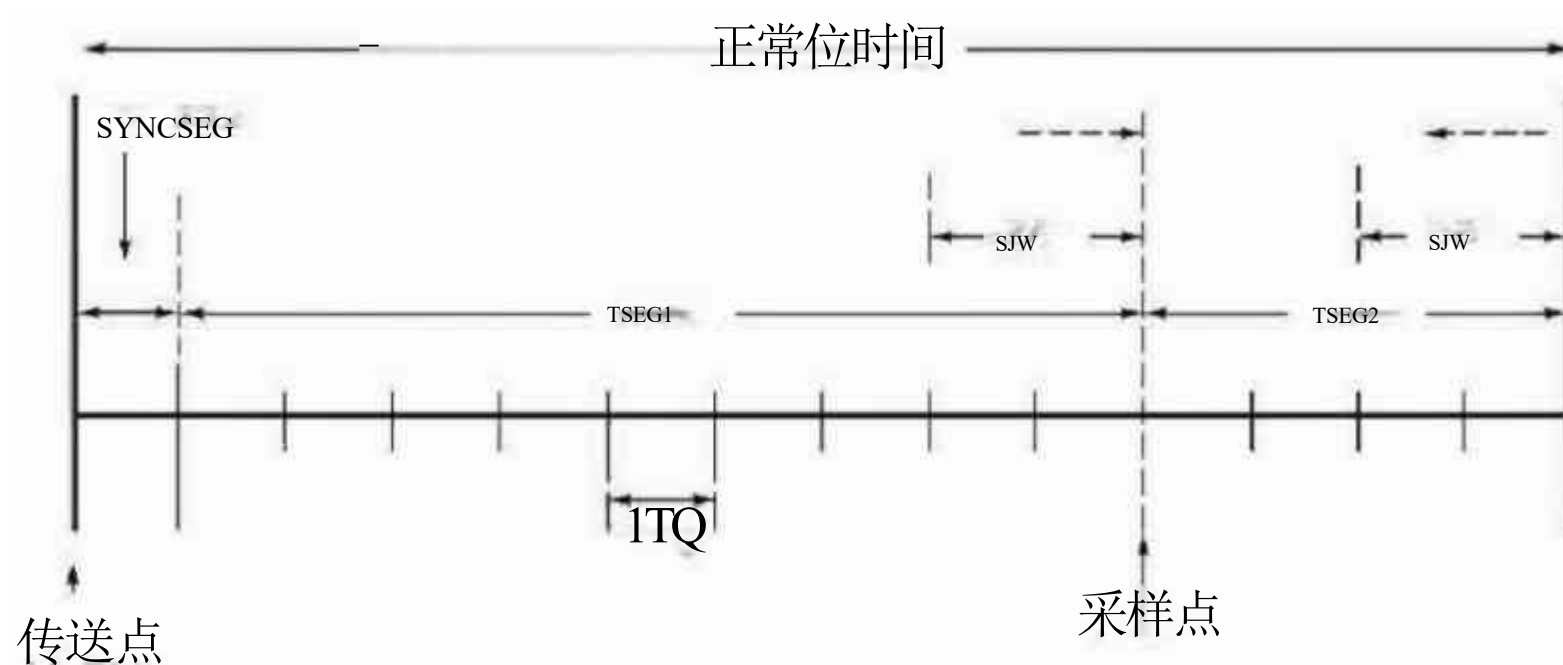


图 13.4 位定时器的时间段

CAN 控制器波特率的计算方法如下:

$$\text{波特率} = f_{clk} / ((BRP+1) + \text{Bit Time})$$

其中: f_{clk} 为 CAN 控制器的时钟频率, 也就是 DSP 的系统频率。

$$\text{位时间}(\text{Bit Time}) = (TSEG1+1) + (TSEG2+1) + 1$$

CAN 控制器的这种波特率设置方法, 使得同一个波特率可以由 BRP、TSEG1 和 TSEG2 不同的组合来实现。

【例 13.1】如系统时钟频率为 24 MHz, CAN 控制器的波特率为 500 Kb/s。试问如何设置位定时器寄存器 BCR2 和 BCR1。

在 SJW=0, SBG=0 和 SAM=0 的条件下:

若 BRP=3, 则满足条件的 TSEG1 和 TSEG2 的值有以下两组:

TSEG1=6, TSEG2=3。即 BCR2=0003h, BCR1=0033h;

>TSEG1=5, TSEG2=4。即 BCR2=0003h, BCR1=001Bh。

若 BRP=1， 则满足条件的 TSEG1 和 TSEG2 的值有以下两组：
>TSEG1=15,TSEG2=6。即BCR2=0001h.BCRI=007Eh;
>TSEG1=14,TSEG2=7 。即BCR2=0001h,BCRI=0077h。

CAN 控制器的状态寄存器有错误状态寄存器(ESR) 和全局状态寄存器(GSR) 组成。
ESR 寄存器反映CAN 控制器的错误状态， GSR 寄存器反映CAN 控制器的一些功能状态。

7. 错误状态寄存器(ESR)——地址7106h

15~9							8
保留位							FER
							RC0
7	6	5	4	3	2	1	
BEF	SA1	CRCE	SER	ACKE	B0)	EP	EW
RC_0	RC_1	RC_0	RC_0	RC0	R_0	R_0	R_0

- 位15~9 保留位。
- 位 8 FER。格式错误。
- 0 CAN 控制器进行正确的发送和接收；
- 1 总线上存在一个格式错误，即固定格式的位场中出现1位或多位非法位时，就产生一个格式错误。
- 位 7 BEF。位错误。
- 0 CAN 控制器进行正确的发送和接收；
- 1 在仲裁域外接收到的位和发送的位不匹配，或在发送仲裁期间一个显性位被发送，但接收到的是一个隐性位。
- 位 6 SA1。显性保留错误。
- 0 CAN 控制器检测到一个隐性位；
- 1 CAN 控制器没有检测到一个隐性位，当硬件复位、软件复位或总线关闭时 SA1 置为1。
- 位 5 CRCE。CRC 错误。
- 0 CAN 控制器没有发生CRC 错误；
- 1 CAN 控制器发生了CRC 错误。
- 位 4 SEG。填充错误。
- 0 填充位错误；
- 1 违反了填充位规则，即发生了填充错误。
- 位 3 ACKE。应答错误。

- 0 CAN 控制器接收到应答信号;
CAN 控制器没有接收到应答信号。
- 位 2 总线关闭状态。
- 0 操作正常, 即在线;
- 1 CAN 总线发生错误, 当发送错误计数器的值达到256时, 就发生总线关闭, 在总线关闭期间 CAN 控制器不能进行发送和接收操作, 清除 CCR 位或设置 ABO 位将恢复总线, 一旦总线恢复, 该位自动清0。
- 位 1 EP。 消极错误状态。
- 0 CAN 控制器不在消极错误模式;
- 1 CAN 控制器处于消极错误模式。
- 位 0 EW。 警告状态。
- 0 接收和发送错误计数器的值都小于96;
- 1 至少有一个错误计数器的值达到96。

8. 全局状态寄存器 (GSR) — 地址7107 h

15~8

保留位

7~6	5	4	3	2	1	0
保留位	SMA	CCE	PDA	保留位	RM	TM
	R_0	R_1	R_0		R_0	R_0

- 位15~6 保留位。
- 位 5 SMA。 悬挂模式应答。
- 0 CAN 控制器不处于悬挂模式;
- 1 CAN 控制器处于悬挂模式。
- 位 4 CCE。 改变配置使能位。
- 0 禁止对配置寄存器进行写操作;
- 1 允许对配置寄存器进行写操作。
- 位 3 PDA。 低功耗模式应答。
- 0 正常工作模式;
- 1 C1AN 控制器已进入低功耗模式。
- 位 2 保留位。
- 位 1 RM。 CAN 控制器处于接收模式。 这一位反映CAN 控制器的实际操作与邮箱的配置无关。

- 0

CAN 控制器没有接收信息；
- 1

CAN 控制器正在接收信息。
- 位 0

TM。CAN 控制器处于发送模式。这一位反映CAN 控制器的实际操作与邮箱的配置无关。
- 0

CAN 控制器没有发送信息；
- 1

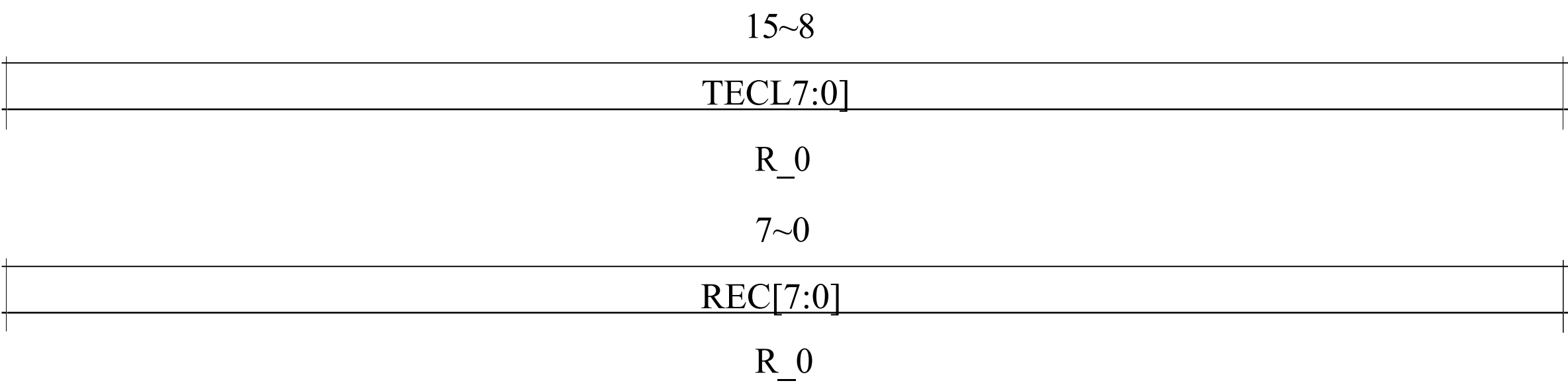
CAN 控制器正在发送信息。

CAN 错误计数寄存器(CE C):CAN 控制器包含两个错误计数器，它们分别是接收错误计数器(REC) 和发送错误计数器(TEC)，其计数值都是可读的。

当接收错误计数器(REC) 的值超过其最大计数值128后，就不再增加。此后当正确接收到一个信息时，计数器的值将被设置在119~127之间。当总线处于关闭状态时，发送错误计数器的值是不确定的，但 REC 将清0。 REC 清0后，其功能将发生改变。当总线上连续出现11个隐性位后，REC 加1。这11位相当于总线上两个报文之间的间隔。当 REC 的值达到128后，如果 MCR 寄存器中的 ABO 位置位，则CAN 控制器自动恢复总线；否则在经过连续128个11位总线空闲信号和 MCR 寄存器中CCR 位被复位后，CAN 控制器恢复总线。在CAN 控制器恢复总线后，CAN 控制器的全部内部标志位被复位，错误计数器清0, 配置寄存器保持原来的值。

在低功耗模式下，错误计数器的值保持不变。当CAN 控制器进入配置模式，错误计数器的值清0。

9. 错误计数寄存器(CEC)—— 地址7108h



中断逻辑：CAN 控制器能产生邮箱中断和错误中断，它们分别可设置为高优先级中断INT1 和低优先级中断 INT5 。 中断使能的情况下，若信息帧被成功地发送或接收，将发生邮箱中断。当发生应答忽略、写拒绝、信息丢失、总线唤醒以及总线关闭等错误时，将发生错误中断。

当产生CAN 中断时，用户程序必须判断 CAN 中断标志寄存器(CAN_IFR) 是否有一个或多个标志位被置位。如果有多于一个标志位被置位，则在相应的中断服务程序中应对每个置位的标志位作相应的处理。因为无论有多少个中断标志位被置位，只响应CAN 中断一次。

CAN 中断标志寄存器(CAN_IF R)：如果 CAN 中断屏蔽寄存器(CAN_IMR) 中相应的中

断屏蔽位被置位，则当CAN 中断标志寄存器(CAN_IFR) 中的相应位被置位时，就产生CAN 中断。中断标志位不能自动清除，用户程序只有写1到相应的中断标志位才能清除。但是MIFx 标志位不能用这种方法来清除，用户程序只有写1到 TCR 寄存器中的 TAn 位(相对于发送邮箱2~5)和写1到RCR 寄存器中的 RMPn 位(相对于接收邮箱0~3),才能清除中断标志位。

10.CAN 中断标志寄存器(CAN_IFR)——地 址7109 h

15~14		13	12	11	10	9	8
保留位		MIF5	MIF4	MIF3	MIF2	MIF1	MIF0
R_0		R_0	R_0	R_0	R_0	R_0	R_0
7	6	5	4	3	2	1	0
保留位	RMLIF	AAIF	WDIF	WUIF	BOIF	EPIF	WLIF
RC_0		RC0	RC_0	RC_0	RC_0	RC_0	RC_0

- 位15~14 保留位。
- 位13~8 MIF5~MIFO。 邮箱5~0中断标志位(接收或发送)

0 没有信息被发送或接收；
1 相应的邮箱成功地发送或接收了信息。
- 位 7 保留位。
- 位 6 RMLIF。 接收丢失中断标志位。

0 没有信息丢失；
1 至少一个接收邮箱发生了接收上溢。
- 位 5 AAIF。 应答忽略中断标志位。

0 发送操作没有忽略；
1 发送操作被忽略。
- 位 4 WDIF。 写拒绝中断标志。

0 成功地对邮箱进行了写操作；
1 写操作遭拒绝。
- 位 3 WUIF。 唤醒中断标志位。

0 CAN 控制器处于休眠模式或正常工作模式；
1 CAN 控制器离开休眠模式。
- 位 2 BOIF。 总线关闭中断标志位。

0 CAN 控制器处于在线状态；
1 CAN 控制器进入总线关闭状态。

- 位 1

EPIF。 消极错误中断标志位。

0 CAN 控制器不处于消极错误模式；

1 CAN 控制器进入消极错误模式。
- 位 0

WLIF。 错误警告中断标志位。

0 错误计数器的值没有达到错误警告值；

1 至少一个错误计数器的值达到错误警告值。

11.CAN 中断屏蔽寄存器 (C AN_{....}IMR)——地址710 Ah

15	14	13	12	11	10	9	8
MIL	保留位	MIM5	MIM4	MIM3	MIM2	MIM1	MIMo
RW_0		RW_0	RW_0	RW_0	RW_o	RW_0	RW_0
7	6	5	4	3	2	1	0
EIL	RMLIM	AAIM	WDIM	WUIM	BOIM	EPIM	W LIM
RW0	RW _o	RW_0	RW_0	RW _g	RW _0	RW _0	RW0

- 位15

MIL。 邮箱中断优先级选择位 (对于邮箱5~0的中断标志位 MIF5~MIF0)。

0 高中断优先级；

1 低中断优先级。
- 位14

保留位。
- 位13~8

MIM5~MIM0。 邮箱5~0中断屏蔽位。

0 禁止中断。

1 使能中断。
- 位 7

EIL。 错误中断优先级选择位。

0 高中断优先级；

1 低中断优先级。
- 位 6

RMLIM。 接收丢失中断屏蔽位。

0 禁止中断；

1 使能中断。
- 位 5

AAIM。 忽略应答中断屏蔽位。

0 禁止中断；

1 使能中断。
- 位 4

WDIM。 写拒绝中断屏蔽位。

0 禁止中断；

1 使能中断。

- 位 3 WUIM。 唤醒中断屏蔽位。
 0 禁止中断；
 1 使能中断。
- 位 2 BOIM。 总线关闭中断屏蔽位。
 0 禁止中断；
 1 使能中断。
- 位 1 EPIM。 消极错误中断屏蔽位。
 0 禁止中断；
 1 使能中断。
- 位 0 WLIM。 错误警告中断屏蔽位。
 0 禁止中断；
 1 使能中断。

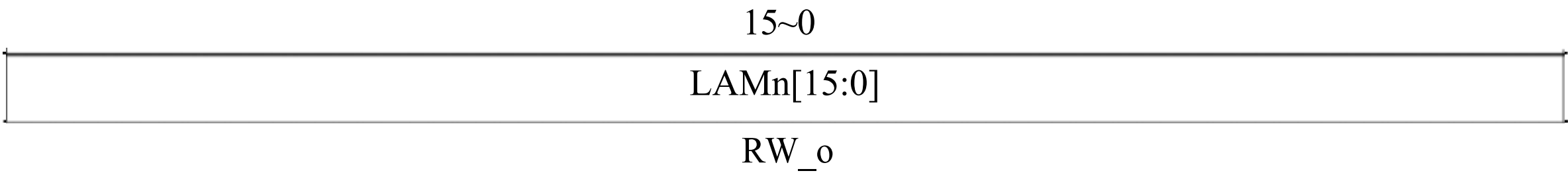
CAN 控制器在接收信息时，先将要接收的信息的标识符与相应接收邮箱的标识符(位于邮箱的 MSGIDnH 和 MSGIDnL 寄存器中)进行比较，只有标识符相同的信息才能被接收。CAN 控制器的接收滤波器，使得接收邮箱可以忽略更多的位来接收信息，即如果只有被屏蔽的那几位标识符不相符，则仍能接收此信息。但当接收屏蔽使能位(AME) 为0时，局部接收屏蔽寄存器将失效。 LAM1 对应于被配置为接收方式下的邮箱2和邮箱3, LAM0 对应于邮箱0和邮箱1。

12. 局部接收屏蔽高位寄存器 nLAM_n H)—— 地址 710 Bh、710Dh

15	14~13	12~0
LAM1	保留位	LAMn[28:16]
RW_0		RW_o

- 位15 LAM1。 局部接收屏蔽高位寄存器扩充位。
 0 接收邮箱是接收标准信息帧还是扩展信息帧由被接收信息的这一位(LAM1) 决定；
 1 标准信息帧和扩展信息帧都可以接收，当接收到扩展信息帧时，接收到29位标识符都可以进行滤波处理，当接收到标准信息帧时，接收到11位标识符只与局部接收屏蔽高位寄存器中的 LAMn_H[12:2] 位进行滤波处理。
- 位14~13 保留位。
- 位12~0 LAMn[28:16]。 高13位局部接收屏蔽位。
 0 相应接收标识符的位值必须相符才能被接收，即接收严格匹配的标识位；
 1 相应接收标识符的位值不相符也能接收，即可接收0或1。

13. 局部接收屏蔽低位寄存器 n (LAMn_ L)—— 地址710 Ch、710Eh



位15~0 LAMn[15:0]。 低16位局部接收屏蔽位。

- 0 相应接收标识符的位值必须相符才能被接收，即接收严格匹配的标识位；
- 1 相应接收标识符的位值不相符也能接收，即可接收0或1

13.4 CAN 控制器的操作

13.4.1 初始化 CAN 控制器

在使用CAN 控制器前必须对它的一些内部寄存器进行设置，如对位定时器进行设置及对邮箱进行初始化。

1. 初始化或重新配置位定时器

主要由 BCR1 和 BCR2 两个寄存器配置位定时器，配置位定时器也就是设置这两个寄存器。BCR1 和 BCR2 寄存器决定CAN 控制器的通信波特率、同步跳转宽度、采样次数和重同步方式。对位定时器的配置步骤如下：

- ＞ 设 置 MCR 寄存器中的改变配置请求位为1，即CCR=1；
 - ＞ 判 断GSR 寄存器中的改变配置使能位是否为1，即 CCE 是否为1，如果CCE=1， 则 进入下一步；
 - ＞ 设 置 BCRI 和 BCR2 寄存器，即配置正确的波特率、同步跳转宽度、采样次数和重同步方式；
 - ＞ 将 MCR 寄存器中的改变配置请求位清0，即CCR=0；
 - ＞ 判 断GSR 寄存器中的改变配置使能位是否为0，即CCE 是否为0，如 CCE=0 则进入下一步；
- 配置完成进入正常工作模式。

图13. 5给出了配置位定时器的流程图。

2. 初始化邮箱

对邮箱初始化主要是设置邮箱的标识符，确定发送的是远程帧还是数据帧，以及对发送的数据区(即对 MBXnA~MBXnD) 赋初值。

初始化邮箱的流程如图13. 6所示。下面为其具体步骤：

- ① 设置 MDER 寄存器中的邮箱使能位为0, 即 $ME_n=0$ ($n=0\sim 5$);
- ② 设置 MCR 寄存器中数据域改变请求位为1, 即 $CDR=1$;
- ③ 设置邮箱的内容, 如标识符寄存器、控制寄存器及数据区;
- ④ 将 MCR 寄存器中数据域改变请求位清0, 即 $CDR=0$ 进入正常工作模式;
- ⑤ 设置 MDER 寄存器中的邮箱使能位为1, 即 $ME_n=1$ 。

通过初始化位定时器和邮箱即完成对 CAN 控制器的初始化, 只要满足一定的条件, 相应的邮箱就能进行正常地发送和接收操作。

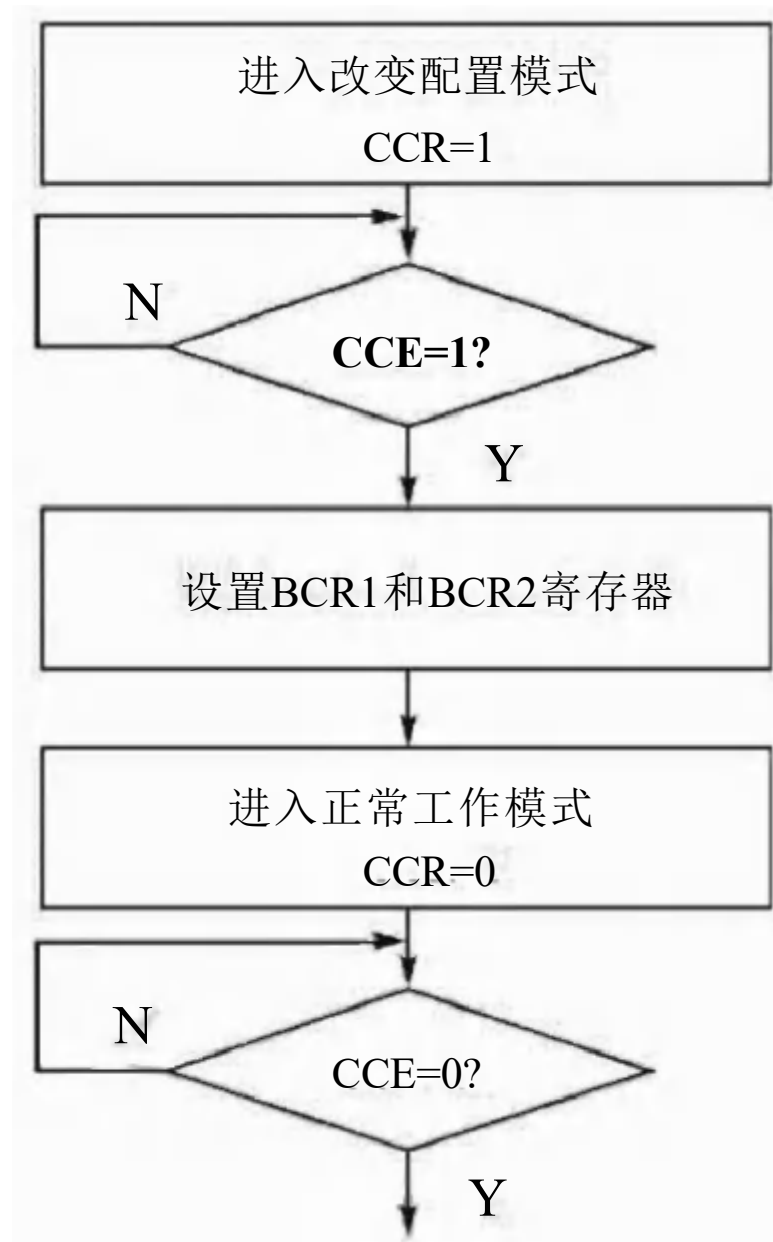


图 13.5 配置位定时器流程图

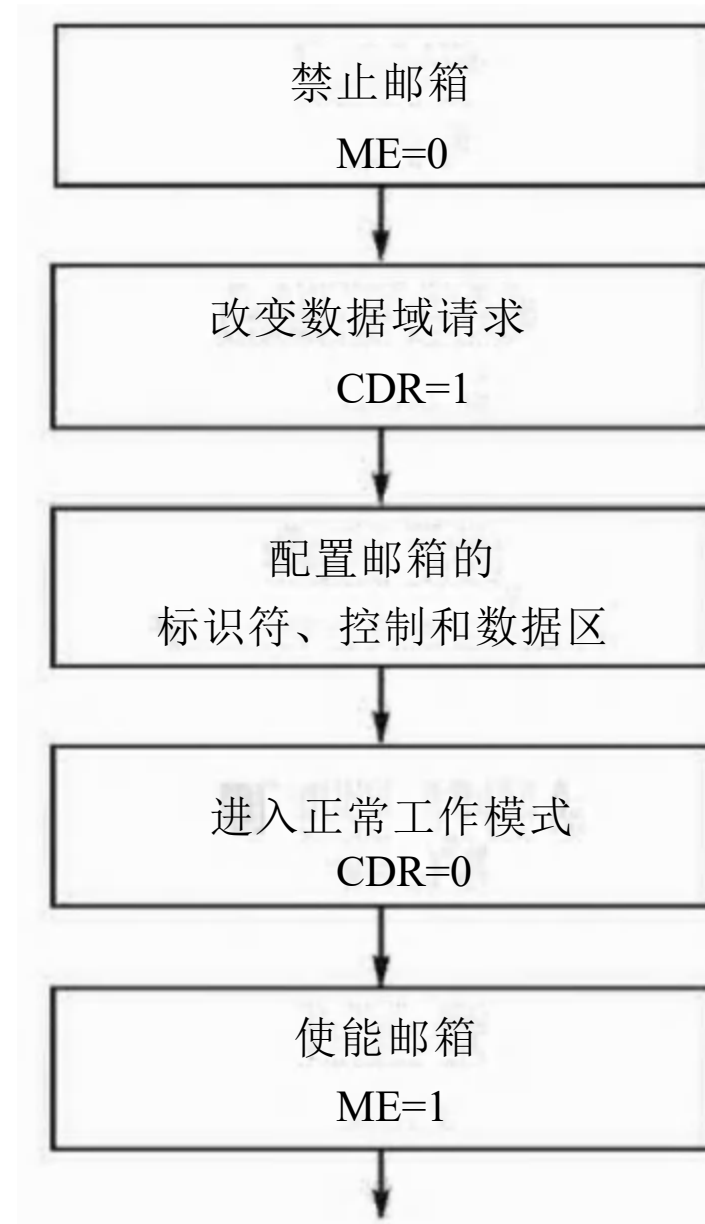


图13.6 初始化邮箱流程图

13.4.2 信息的发送

CAN 控制器的发送邮箱有邮箱4和邮箱5以及被配置为发送方式的邮箱2和邮箱3。在写数据到发送邮箱的数据区后, 如果相应的发送请求位使能, 则信息帧被发送到 CAN 总线上。

信息发送的流程图如图13.7所示。下面是其具体步骤:

- ① 初始化发送邮箱;
- ② 设置 MDER 寄存器中的邮箱使能位为1, 即 $ME_n=1$ ($n=2\sim 5$);
- ③ 设置 TCR 寄存器中的发送请求位为1, 即 $TSR_n=1$;
- ④ 等待发送应答信号 TAn 或发送中断标志位 MIF_n 置位, 如果 $TAn=1$ 或 $MIF_n=1$, 则

发送成功, 进入下一步。

⑤ 向 TCR 寄存器中的发送应答位(TAn) 写1, 清除发送中断标志位和发送应答位。

如果多个发送邮箱的发送请求位置位, 则信息帧将一个接一个地发送出去, 邮箱权限高的先发送。如果发送失败, 则发送邮箱将再次发送。

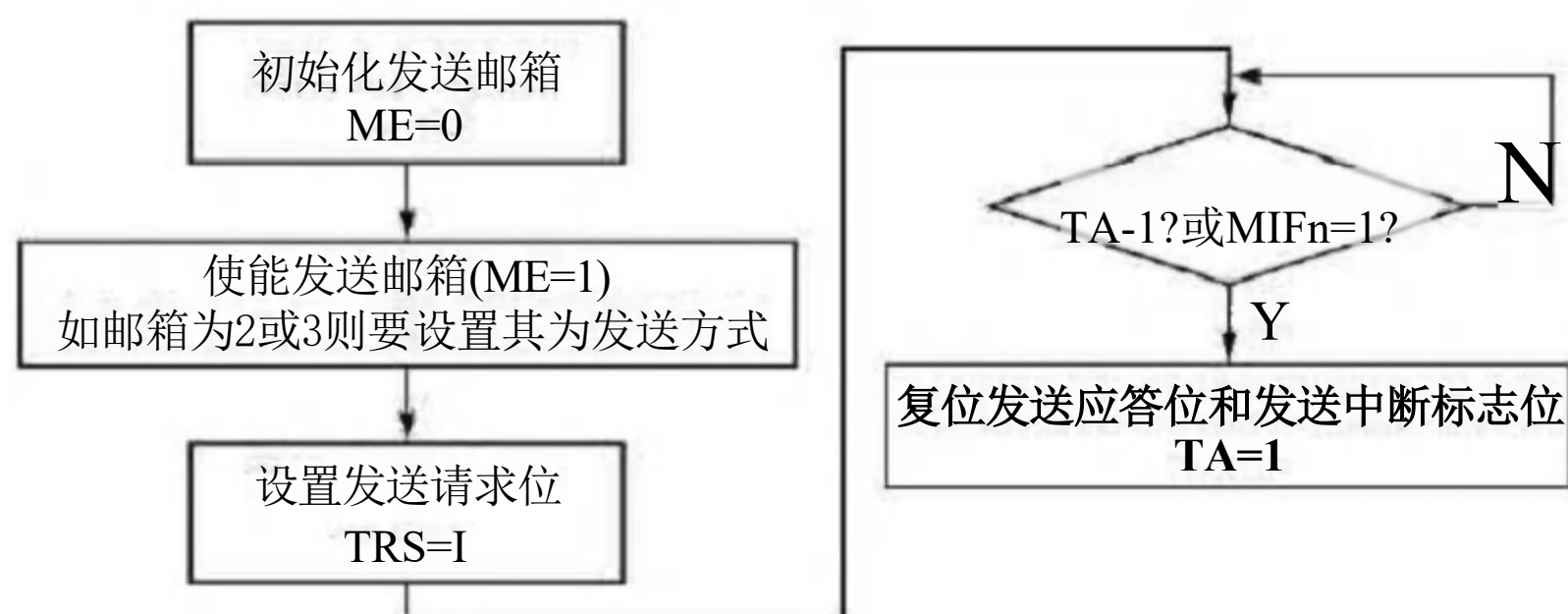


图13.7 信息的发送流程图

13.4.3 信息的接收

CAN 控制器的接收邮箱有邮箱0和邮箱1, 以及被配置为接收方式的邮箱2和邮箱3。接收邮箱初始化时, 要设置其标识符及标识符相关的局部屏蔽寄存器(LAM)。

1. 接收滤波器

CAN 控制器在接收信息时, 先将要接收的信息的标识符与相应接收邮箱的标识符(位于邮箱的 MSGIDnH 和 MSGIDnL 寄存器中)进行比较, 只有标识符相同的信息才能被接收。CAN 控制器的接收滤波器使得接收邮箱可以忽略更多的位来接收信息, 即若只有被屏蔽的那几位标识符不相符, 则接收邮箱仍能接收此信息。但当接收屏蔽使能位(AME) 为0时, 则局部接收屏蔽寄存器将失效。LAM1 对应于被配置为接收方式下的邮箱2和邮箱3. LAM0 对应于邮箱0和邮箱1。

【例13.1】 接收滤波器举例, 其中屏蔽允许, 即 AME=1。

接收信息的标识符=10000000000000000000000011110000

接收邮箱的标识符=10000000000000000000000000000000

当接收屏蔽位=10000000000000000000000011110000时, 接收信息;

当接收屏蔽位=10000000000000000000000000000000时, 拒绝信息。

2. 接收信息的步骤

邮箱接收信息的流程图如图13.8所示。下面是其具体步骤。

- ① 设置局部屏蔽寄存器(LAM);
- ② 设置接收邮箱的标识符和控制寄存器;
- ③ 等待接收信息悬挂位RMPn 或接收中断标志位MIFn 置位, 若RMPn=1 或 MIFn=1

则接收成功，进入下一步；

④ 向 RCR 寄存器中的接收信息悬挂位(RMPn) 写1, 清除接收中断标志位和接收信息悬挂位。

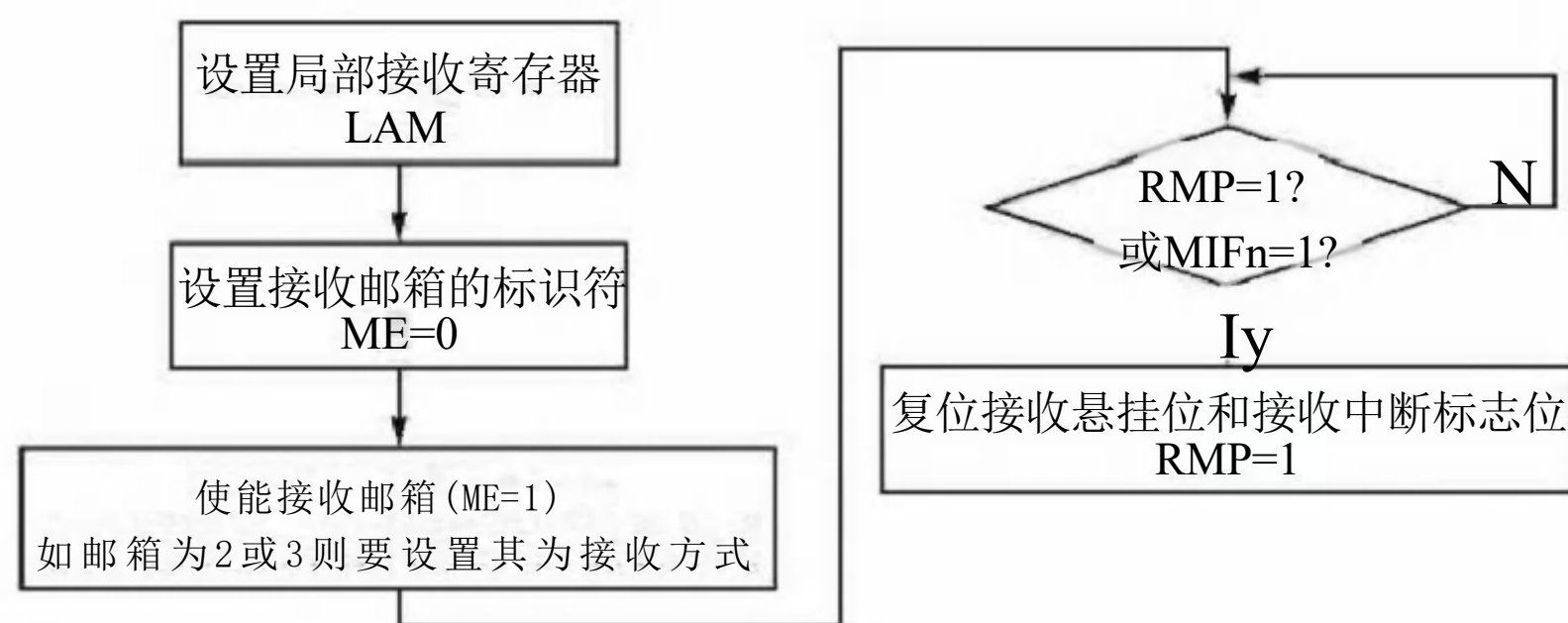


图13.8 信息的接收流程图

13.4.4 远程帧

当邮箱要发送远程帧时，应设置远程发送请求位(RTR) 为1。远程帧与数据帧具有相同的形式，它也有标准信息帧和扩展信息帧两种格式，但远程帧中不含数据区。

远程帧主要用于请求信息。当节点A 向节点B 发送一个远程帧时，如果节点B 中的远程帧信息与节点A 有相同的标识符，则节点B 将作出应答，并发送相应的数据帧到总线上。

1. 发送远程帧

› 邮箱4和邮箱5，以及被配置为发送方式的邮箱2和邮箱3都可作为远程帧的发送邮箱。

› 设置 MSGCTRLn 寄存器的远程发送请求位为1, 即 RTR=1。

设置 TCR 寄存器中的发送请求位为1, 即 TRS=1。

› 远程帧被发送到CAN 总线上。如果远程帧发送邮箱为2或邮箱3, 则当发送成功后，将不置位发送应答位TAn 和发送中断标志位MIFn, 且 TRSn 自动复位，这里n=2,3。

2. 自动应答远程帧

当邮箱接收到远程帧时，将自动发送一个数据帧作为应答。

› 只有被配置为发送方式的邮箱2和邮箱3才有这个功能。

设置 MSGIDn_H 寄存器中的自动应答模式位为1, 即AAM=1。

› 当接收节点的标识符与发送邮箱2或邮箱3的标识符相符时，接收节点将自动发送一个数据帧作为应答。

3. 接收远程帧

› 邮箱0和邮箱1，以及被配置为接收方式的邮箱2和邮箱3都可作为远程帧的接收邮箱。

› 当接收到远程帧时，将接收信息悬挂位和接收中断标志位置位1, 即 RPMn=1,

MIFn=1。这里 n=0~3。

CPU 响应相应的悬挂。

13.5 CAN 控制器的应用举例

下面的设计是带 CAN 控制器的 F2407 芯片与 CAN 卡之间的通信。

13.5.1 硬件电路设计

F2407 与 CAN 驱动芯片的接口电路如图13.9所示，图中 PCA82C250T 是驱动CAN 控制器与物理总线间的接口，提供对总线的差动发送和接收功能。电阻R4 作为CAN 终端的匹配电阻。电阻 R1、R2、R3 及二极管 D1 组成的电路为电平转换电路，因为F2407 用 3.3 V 供电，而 PCA82C250T 用 5 V 供电。

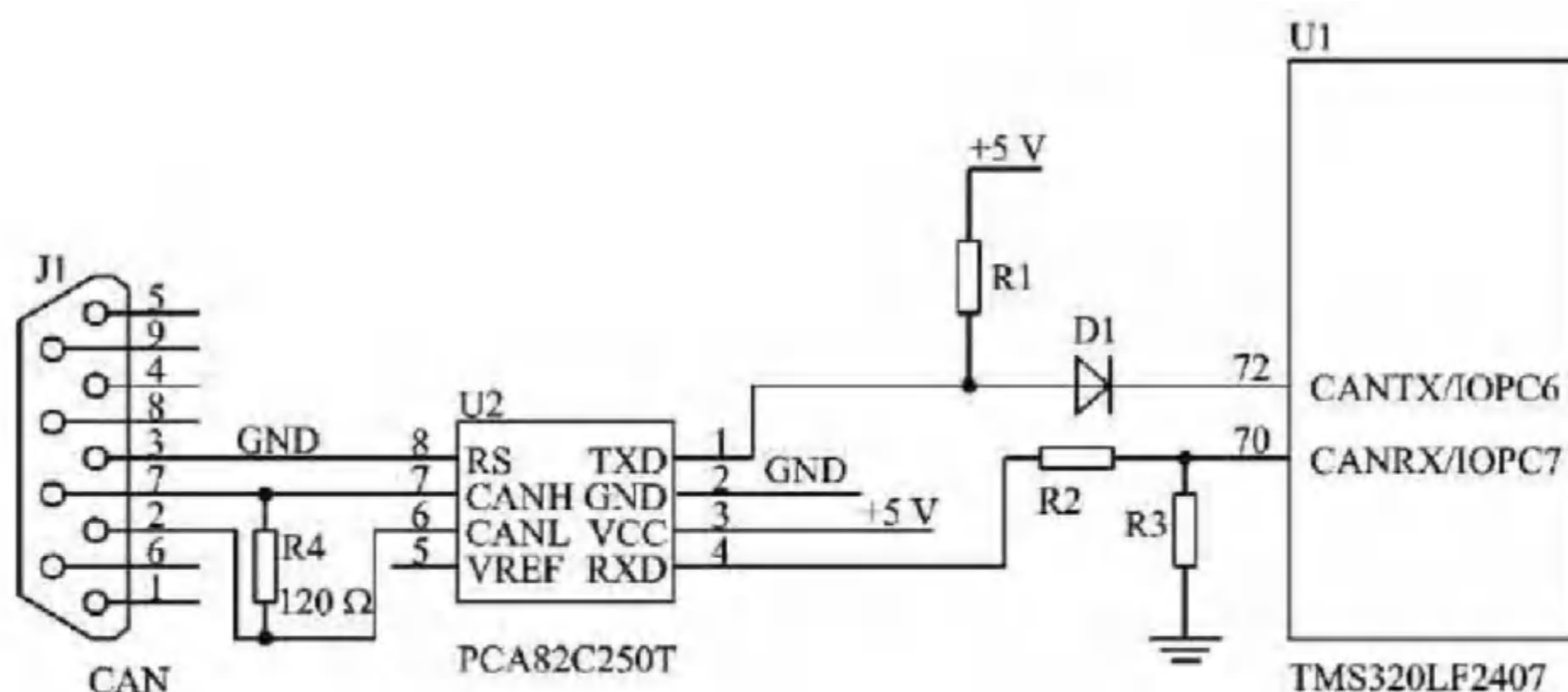


图13.9 F2407与CAN驱动接口电路

13.5.2 软件设计

1. 汇编语言程序设计

下面的程序是DSP 与 CAN 卡之间的通信程序，上位机用台湾研华公司 CAN 卡的调试界面 Monitor。在此只给出了下位机程序。DSP 的 CAN 控制器的邮箱2配置为接收方式，邮箱3配置为发送方式，都采用标准信息帧格式。发送采用查询方式，接收采用中断方式。当接收到数据之后，用接收到的数据更新邮箱3的数据，邮箱2用软件中断优先级GISR5。其程序如下：

```

st0_temp      .usect      ".b20",1
st1_temp      .usect      ".b20".1
context       .usect      ".b20".7                ;62~68
    
```


	.sect	".pvecs"	;定义子向量段
PVECTORS	B	PHANTOM	;保留向量地址偏移量00h
	B	PHANTOM	;保留向量地址偏移量01h
	B	PHANTOM	;保留向量地址偏移量02h
	B	PHANTOM	;保留向量地址偏移量03h
	B	PHANTOM	;保留向量地址偏移量04h
	B	PHANTOM	;保留向量地址偏移量05h
	B	PHANTOM	;保留向量地址偏移量06h
	B	PHANTOM	;保留向量地址偏移量07h
	B	PHANTOM	;Reserved pvector addr offset 08h
	B	PHANTOM	;Reserved pvector addr offset 09h
	B	PHANTOM	;Reserved pvector addr offset 0Ah
	B	PHANTOM	;Reserved pvector addr offset 0Bh
	B	PHANTOM	;Reserved pvector addr offset 0Ch
	B	PHANTOM	;Reserved pvector addr offset 0Dh
	B	PHANTOM	;Reserved pvector addr offset 0Eh
	B	PHANTOM	;Reserved pvector addr offset 0Fh
	B	PHANTOM	;Reserved pvector addr offset 10h
	B	PHANTOM	;Reserved pvector addr offset 11h
	B	PHANTOM	;Reserved pvector addr offset 12h
	B	PHANTOM	;Reserved pvector addr offset 13h
	B	PHANTOM	;Reserved pvector addr offset 14h
	B	PHANTOM	;Reserved pvector addr offset 15h
	B	PHANTOM	;Reserved pvector addr offset 16h
	B	PHANTOM	;Reserved pvector addr offset 17h
	B	PHANTOM	;Reserved pvector addr offset 18h
	B	PHANTOM	;Reserved pvector addr offset 19h
	B	PHANTOM	;Reserved pvector addr offset 1Ah
	B	PHANTOM	;Reserved pvector addr offset 1Bh
	B	PHANTOM	;Reserved pvector addr offset 1Ch
	B	PHANTOM	;Reserved pvector addr offset 1Dh
	B	PHANTOM	;Reserved pvector addr offset 1Eh
	B	PHANTOM	;Reserved pvector addr offset 1Fh
	B	PHANTOM	;Reserved pvector addr offset 20h
	B	PHANTOM	;Reserved pvector addr offset 21h
	B	PHANTOM	;Reserved pvector addr offset 22h
	B	PHANTOM	;Reserved pvector addr offset 23h
	B	PHANTOM	;Reserved pvector addr offset 24h
	B	PHANTOM	;Reserved pvector addr offset 25h

```

B      PHANTOM      ;Reserved pvector addr offset 26h
B      PHANTOM      ;Reserved pvector addr offset 27h
B      PHANTOM      ;Reserved pvector addr offset 28h
B      PHANTOM      ;Reserved pvector addr offset 29h
B      PHANTOM      ;Reserved pvector addr offset 2Ah
B      PHANTOM      ;Reserved pvector addr offset 2Bh
B      PHANTOM      ;Reserved pvector addr offset 2Ch
B      PHANTOM      ;Reserved pvector addr offset 2Dh
B      PHANTOM      ;Reserved pvector addr offset 2Eh
B      PHANTOM      ;Reserved pvector addr offset 2Fh
B      PHANTOM      ;Reserved pvector addr offset 30h
B      PHANTOM      ;Reserved pvector addr offset 31h
B      PHANTOM      ;Reserved pvector addr offset 32h
B      PHANTOM      ;Reserved pvector addr offset 33h
B      PHANTOM      ;Reserved pvector addr offset 34h
B      PHANTOM      ;Reserved pvector addr offset 35h
B      PHANTOM      ;Reserved pvector addr offset 36h
B      PHANTOM      ;Reserved pvector addr offset 37h
B      PHANTOM      ;Reserved pvector addr offset 38h
B      PHANTOM      ;Reserved pvector addr offset 39h
B      PHANTOM      ;Reserved pvector addr offset 3Ah
B      PHANTOM      ;Reserved pvector addr offset 3Bh
B      PHANTOM      ;Reserved pvector addr offset 3Ch
B      PHANTOM      ;Reserved pvector addr offset 3Dh
B      PHANTOM      ;Reserved pvector addr offset 3Eh
B      PHANTOM      ;保留向量地址偏移量3Fh
B      CANMB_X_ISR   ;保留向量地址偏移量40h CAN 接收中断
B      PHANTOM      ;保留向量地址偏移量41h

; ②主程序
        .text
c__int0
        CALL    SYSINIT      ;系统初始化程序
        LDP     #DP_USER
        SPLK    #00H,CAN_FLAG ;清 CAN用户标志, CAN_FLAG=01表示接收到数据
        CALL    CAN_INIT     ;CAN初始化程序
        CLRC    INTM         ;开总中断
LOOP:   LDP     #DP_CAN
        SPLK    #20h,TCR      ;邮箱3发送请求
W_TA3:  LDP     #DP_USER

```



```

        BIT      CAN_FLAG.BITO      ;判断是否接收到新数据
        BCND     LOOP2, TC
        LDP      #DP_CAN
        BIT      TCR. 2
        BCND     W_TA3, NTC          ;等待发送应答
LOOP2:   LDP      #DP_CAN
        SPLK     #2000h, TCR        ;清 TA3 和 MIF3 标志位
LOOP1:   LDP      #DP_USER
        BIT      CAN_FLAG, BITO     ;判是否接收到数据，如接收到则更新邮箱3
        BCND     LOOP1. NTC
        SPLK     #00H,CAN_FLAG      ;清用户接收标志
        LDP      #DP_CAN
        SPLK     #0000H, MDER        ;邮箱不使能
        SPLK     #0140H, MCR        ;CDR=1，数据改变请求
        LDP      #DPCAN2            ;DP→7200h
        LACL     MBX2A              ;更新邮箱3
        SACL     MBX3A              ;把邮箱2的数据移入到邮箱3
        LACL     MBX2B
        SACL     MBX3B
        LACL     MBX2C
        SACL     MBX3C
        LACL     MBX2D
        SACL     MBX3D
WAIT:    NOP
        B        LOOP
```

;③系统初始化程序

SYSINIT:

```

        SETC     INTM
        CLRC     SXM
        CLRC     OVM
        CLRC     CNF
        LDP      #0EOH
        SPLK     #81FEH,SCSRI
        SPLK     #0E8h. WDCR
        LDP      #0
        SPLK     #10h,IMR           ;开中断优先级5
        SPLK     #OFFFh,IFR        ;清中断标志
        RET
```

;④CAN 初始化程序

```
CAN_INIT:
    LDP      #DP_PF2
    LACL     MCRB          ;配置CAN 引脚
    OR       #OCOH        ;IOPC6 和 IOPC7 配置为特殊功能: CANRX 和 CANTX
    SACL     MCRB
; -----CAN 位 定 时 器 配 置 -----
    LDP      #DP_CAN
    ;SPLK    #0040H,MDER    ;MD2=1,MBX2为发送方式
    SPLK     #0FFFFH,CAN_IFR ;清全部 CAN中 断 标 志
    SPLK     #07FFFH,LAM1_H   ;设置邮箱2、3的屏蔽 ID 寄存器
    SPLK     #0FFFFH,LAM1_L   ; 0 则 ID 必须匹配
    SPLK     #1000H, MCR       ;CCR=1 改变配置请求
W_CCE      BIT      GSR,#04h   ;等待改变配置使能
    BCND     W_CCE,NTC         ;当 CCE=1 时即可配置 BCR2 和 BCR1 寄存器
    SPLK     #01H,BCR2        ;波特率预分频寄存器
    SPLK     #0033H,BCR1      ;波特率设置为1 Mb/s
    LACL     MCR
    AND      #0EFFFH
    SACL     MCR              ;CCR=0 改变配置结束请求
W_NCCE     BIT      GSR,#04h   ;等待改变配置不使能
    BCND     W_NCCE,TC        ;只有当 CEE=0 时 , BCR2 和 BCR1寄存器配置成功
    -----写 CAN 邮 箱 内 容 前 寄 存 器 设 置 -----
    LDP      #DP_CAN
    SPLK     #0040H,MDER      ;不使能邮箱, 邮箱2设为接收方式
    SPLK     #0143H,MCR       ;CDR=1, 数据区改变请求
    ----- 写CAN邮 箱 内 容 -----
    LDP      #DP_CAN2
    SPLK     #2447H,MSGID2H    ;设置邮箱2的控制字及 ID
                                ;IDE=0,AME=0,AAM=0
                                ;标准方式为MSGID2H[12:2]
    SPLK     #0FFFFH,MSGID2L
    SPLK     #08H,MSGCTRL2     ;设置控制域
                                ;数据长度 DCL=8,RTR=0 数据帧
    SPLK     #00000H,MBX2A     ;邮箱2信息初始化
    SPLK     #00000H,MBX2B
    SPLK     #00000H, MBX2C
    SPLK     #00000H, MBX2D
    SPLK     #2447H,MSGID3H    ;设置邮箱3的标识符
    SPLK     #0FFFFH,MSGID3L
```

```

        SPLK      #08H,MSGCTRL3           ;RTR=0,DCL=8
        SPLK      #02211h,MBX3A           ;邮箱3信息初始化
        SPLK      #04433h,MBX3B
        SPLK      #06655h,MBX3C
        SPLK      #08877h,MBX3D
; ----- 写CAN 邮箱内容后寄存器设置 -----
        LDP       #DP_CAN
        SPLK      #0F7FFH,CAN_IMR         ;中断 MBX3 不使能, MBX2 使能, 低中断优先级
        SPLK      #OFFFH,CAN_IFR          ;清全部中断标志
        SPLK      #04c0H,MCR               ;DBO=1.CDR=0.ABO=1.STM=0
        SPLK      #04CH,MDER              ;ME2=ME3=1,MBX2接收, MBX3发送
        RET
; ⑤ 中断程序
GISR5:                                     ;优先级 INT5中断入口
        LDP       #0                       ;保存机器上下文
        SST       #0,st0_temp              ;使用自动寻址, DPO
        SST       #1,st1_temp              ;保存状态寄存器到 B2 DARAM
        LDP       #0
        SACL      context                  ;保存ACC 的低16位
        SACH      context+1                ;保存 ACC的高16位
        SAR       AR1,context+2
        SAR       AR2,context+3
        SAR       AR3,context+4
        SAR       AR4,context+5
        SAR       AR5,context+6
        LDP       #0EOH
        LACC      PIVR,1                    ;读取外设中断向量寄存器PIVR,并左移1位
        ADD       #PVECTORS                ;加上外设中断入口地址
        BACC
;跳到相应的中断服务子程序
CANMBXISR:                                ;CAN接收中断程序
        LDP       #DP_CAN
        SPLK      #0040h,RCR               ;复位 RMP2 和 MIF2
        LDP       #DP_USER
        SPLK      #01H,CAN_FLAG            ;置用户接收标志
GISR5 RET:                                ;中断返回
        :恢复现场
        LDP       #DP_EVA
        SPLK      #0FFFH,EVAIFRA
    
```



```

LDP    #0
LAR    AR5,context+6
LAR    AR4,context+5
LAR    AR3,context+4
LAR    AR2,context+3
LAR    AR1,context+2
LACC   context+1.16
ADDS   context
LST    #1,st1_temp
LST    #0,st0_temp
CLRC   INTM                      ;开总中断，因为一进入中断，就自动关闭总中断
RET

;⑥假中断程序
PHANTOM

KICK_DOG                      ;复位看门狗
RET
END

```

2.C 语言程序设计

将主控制器 MCR 中的 STM 位置 1, 可以使 CAN 控制器工作于自测试模式。在这种模式下, CAN 控制器能自己产生应答信号, 因此不需要与 CAN 总线相连, 信息帧没有真正发送出去, 但是信息帧被读回, 并存储在相应的邮箱中。在自测试模式下, 不能进行远程帧悬挂自动应答。也不能保存被接收的信息帧的标识符。

由于 F2407 芯片的 CAN 控制器在自测试模式下, 无须外接其他硬件, 而软件设计与正常模式下区别不大, 因而特别适合于初学者调试学习。自测试模式与正常模式在软件上的区别仅在于需要设置 STM 位为 1。

以下是带 CAN 控制器的 F2407 芯片工作于自测试模式的软件设计。DSP 的 CAN 控制器的邮箱 2 配置为接收方式, 邮箱 3 配置为发送方式, 都采用标准信息帧格式。发送用查询方式, 接收用中断方式。当接收到数据之后, 邮箱 2 中的数据加 1, 用来更新邮箱 3 中的数据, 并进行下一次发送。邮箱 2 用低中断优先级 GISR5, 读者可以通过 C2000 的开发工具 CCS3.1, 来观测邮箱 2 是否接收到邮箱 3 中的数据。程序分为两部分: 第一部分为汇编语言建立的中断向量表; 第二部分为 C 语言编写的源程序(包括系统初始化, CAN 模块初始化以及主程序等)。源程序如下:

(1) 建立中断向量表(汇编)

```

.ref    nothing ,c__int0,GRIS5          ;声明在其他程序中定义
                                              ;并在本程序中需要使用的变量或标号
.sect   ".vectors"                      ;定义主向量段

```

RSVECT	B	c_int0	;复位向量
INT1	B	nothing	;中断向量1
INT2	B	nothing	;中断向量2
INT3	B	nothing	;中断向量3
INT4	B	nothing	;中断向量4
INT5	B	GRIS5	;中断向量5
INT6	B	nothing	;中断向量6
RESERVED	B	nothing	
SW_INT8	B	nothing	

SW_INT31	B	nothing	

(2) 主程序

```
#include "F2407_c.h"
int CAN FLAG;
void system_init();
void CANMBX_ISR();
void CAN_INIT();
void inline disable()
```

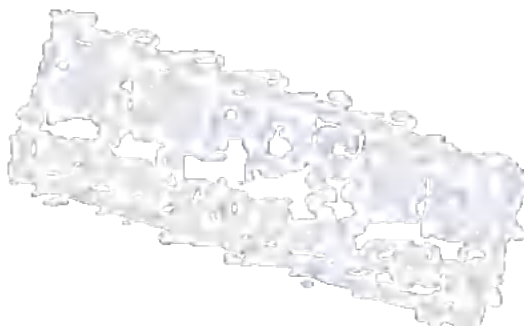
```
asm("setc INTN");

void inline enable()
(
asm("clrc INTM");
}
```

1/①主程序

```
main()
(
system_init();
CAN_FLAG=0x00;
CAN_INIT();
enable():
for(;;)
{
TCR=0x20;
while(TCR&0x2000==0)
continue;
TCR=0x2000;
```

//引用头部文件
//定义标志寄存器
//声明在程序中需要用到是函数



//系统初始化子程序
//清 CAN用户标志，CANFLAG=0 表示收到数据
//CAN 初始化子程序

//MBX3请求发送
//等待发送应答

//清 TA3和 MIF3 标志位

```

while(CAN_FLAG==0)
continue;
CAN_FLAG=0x0000;
MDER=0x0000;
MCR=0x0140;
MBX3A=MBX2A+1;
MBX3B=MBX2B+1;
MBX3C=MBX2C+1;
MBX3D=MBX2D+1;
}
//②系统初始化子程序
void system_init()

    asm("setc SXM");
    asm("clrc OVM");
    asm("clrc CNF");
    asm("setc INTM");
    SCSR1=0x81FE;
    WDCR=0x0E8;

    IMR=0x0010;
    IFR=0x0FFFF;
}
//③CAN 初始化子程序
void CAN_INIT()
{
    MCRB=MCRB|0x0C0;
    CAN_IFR=0x0FFFF;
    LAM1H=0x7FFF;
    LAM1_L=0x0FFFF;
    MCR=0x1040;
    while(GSR&0x0010==0)continue;
    BCR2=0x01;
    BCR1=0x033;
    MCR=MCR&0x0EFFF;
    while(GSR&0x0010!=0)continue;
    MDER=0x040;
    MCR=0x0143;
    MSGID2H=0x2447;

//等待接收数据
//清接收到标志
//邮箱不使能
//CDR=1, 数据改变请求
//邮箱3中数据加1用来更新2中的数据

//抑制符号位扩展
//累加器中结果正常溢出
//B0 被配置为数据存储空间
//禁止所有中断
//CLKIN=6 MHz,CLKOUT=4CLKIN=24 MHz
//不使能看门狗, 因为 SCSR2中的 WDOVERRIDE
//即WD 复位后的缺省值, 故可以用软件禁止看门狗
//开中断优先级5
//清除全部中断标志,

//设置 IOPC6、IOPC7为 CANRX、CANTX
//清所有中断标志
//设置邮箱3、2的屏蔽 ID 寄存器
//0则ID 必须匹配
//CCR=1 改变配置请求
//CCE=1 时即可配置 BCR2、BCR1 寄存器
//波特率预分频寄存器
//波特率设置为500 Kb/s
//CCR=0 改变配置请求
//只有当CCE=0 时, 配置BCR2、BCR1 寄存器成功
//不使能邮箱2, 邮箱2改为接收方式
//CDR=1, 数据区改变请求
//设置邮箱2的控制字及 ID

```



```
MSGID2L=0x0FFFF;
MSGCTRL2=0x08;
MBX2A=0x0000;
MBX2B=0x0000;
MBX2C=0x0000;
MBX2D=0x0000;
MSGID3H=0x2447;
MSGID3L=0x0FFFF;
MSGCTRL3=0x08;
MBX3A=0x2211;
MBX3B=0x4433;
MBX3C=0x6655;
MBX3D=0x8877;
CAN_IMR=0x0F7FF;
CAN_IFR=0x0FFFF;
MCR=0x04c0;
MDER=0x04C;
```

/IDE=0、AME=0、AAM=0
//标准方式为 MSGID2H[12:2]

/设置控制域；数据长度DCL=8.RTR=0 数据帧
// 邮箱2信息初始化

// 设置邮箱3的标识符

/RTR=0,DCL=8
// 邮箱3信息初始化

/DCL=8,RTR=1,STM=1 设置为自测试模式
/ 使能邮箱2和3, 邮箱2为接收方式

K

/④ 中断程序

```
void interrupt GRIS5()
```

```
switch(PIVR)
```

```
{
```

```
case 64:CANMBX ISR();
```

```
break;
```

```
void CANMBX ISR()
```

```
{
```

```
RCR=0x040;
```

```
CAN_FLAG=1;
```

```
}
```

//直接返回中断服务程序

```
void interrupt nothing()
```

```
{
```

```
return;
```

第 14 章 TMS320LF2407 与图形液晶显示模块接口及应用

14.1 概 述

由于液晶显示器(LCD)具有功耗低、体积小、重量轻及超薄等诸多其他显示器无法比拟的优点，因此广泛用于各种智能型仪器和低功耗电子产品中。点阵式(或称为图形显示方式) LCD 液晶显示器不仅可以显示字符及数字，还可以显示各种图形、曲线及汉字，并且可实现屏幕上下左右滚动、动画、闪烁及文本特征显示等功能，用途十分广泛。本章在简介液晶显示器(MCG12864A8-3)的驱动器结构及功能的基础上，将介绍DSP 芯片F2407 与 LCD 硬件接口电路以及软件编程特点。

14.2 液晶显示器与 TMS320LF2407 接口硬件设计

14.2.1 MCG12864A8-3 的结构特点

该液晶显示模块使用两片KS0108B 及其兼容控制驱动器(例如 HD61202)作为列驱动器，一片KS0107B 及其兼容驱动器(例如 HD61203)作为行驱动器，如图14.1所示。由于KS0107B(或HD61203)不与 DSP 发生联系，故只要提供电源，就能产生行驱动信号和各种同步信号。

在 MCG12864A8-3 中，CSA与 KS0108B(1)的 CS1 相连；CSB与 KS0108B(2)的 CS1 相连。因此CSA、CSB选通信号为CSA CSB=01选通(1)，CSA CSB=10选通(2)。

14.2.2 MCG12864A8-3 模块的引脚说明

MCG12864A8-3 模块共有18个引脚，其定义如表14.1所列。

关于MCG12864A8-3 的接口信号和时序等详细资料，可参见液晶显示模块使用手册。

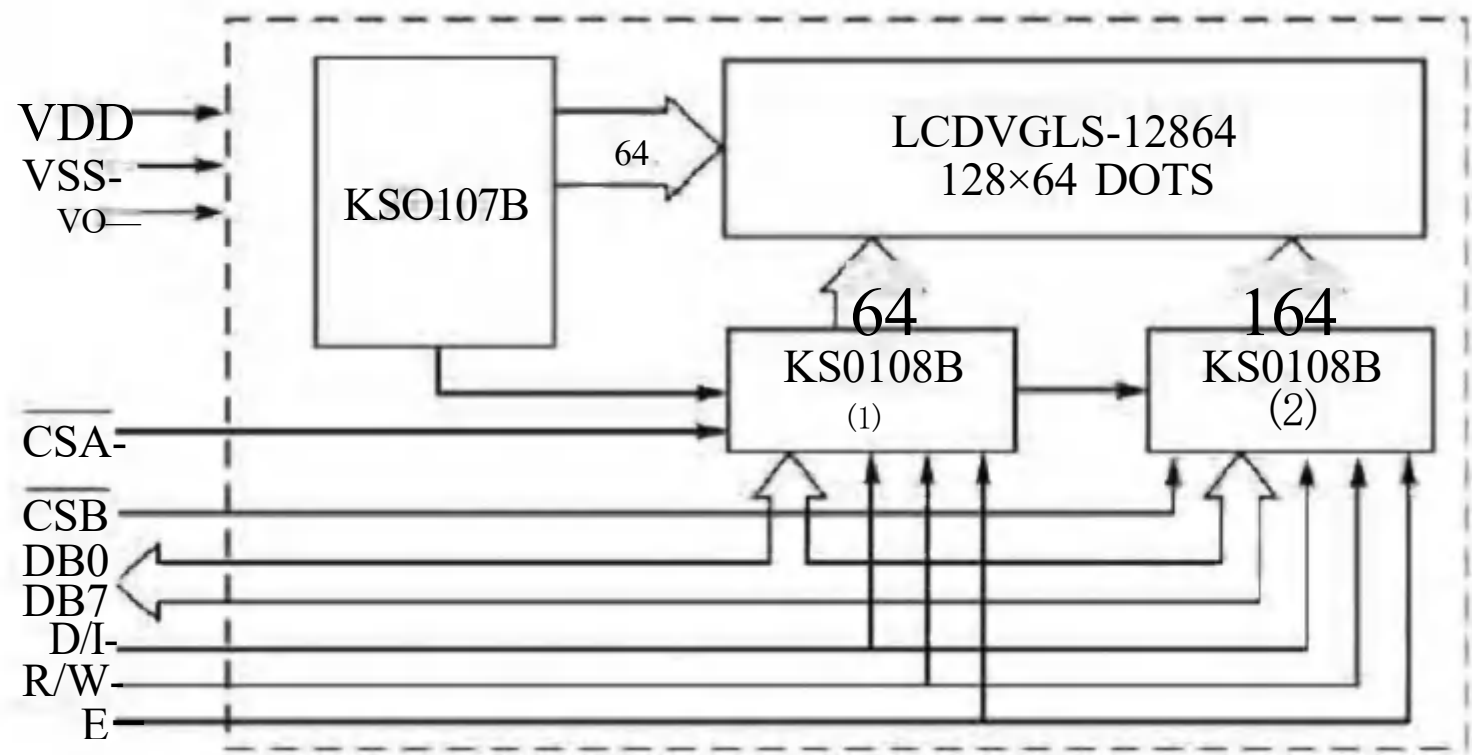


图14. 1 MCG12864A8-3 逻辑电路

表14. 1 MCG12864A8-3 模块的引脚定义

序 号	符 号	状 态	功能说明	序 号	符 号	状 态	功能说明
1	CSA	输入	片选1	7	R/W	输入	读/写选择
2	CSB	输入	片选2	8	E	输入	使能信号. 数据在下降沿时被写入LCM:在高电平时被读出LCM
3	VSS		数字信号地				
4	VDD		逻辑电源+5V	9-16	DB0~DB7	三态	数据线
5	VO (/POFF)	三	对比度调节	17	1. CD+		LCD背光正电源端
6	D/1	输入	指令数据通道	18	LCD—		LCD接地端

14.3 TMS320LF2407 与 MCG12864A8-3 模块的接口电路

现 以TI 公司的F2407 芯片为例，说明 DSP 与 MCG12864A8-3 模块的硬件接口电路。其接口控制时序采用了M6800 操作时序，这样引出的控制信号R/W、A0、E1 和 E2 由 F2407 的 I/O 端口的4个引脚控制。

TMS320LF2407 与 MCG12864A8-3 模块的接口电路如图14. 2所示。

电阻R67 用来驱动液晶的背光灯，其中 DSP 侧的引脚都设为 I/O 端口方式，用软件来模拟 M6800 操作时序。

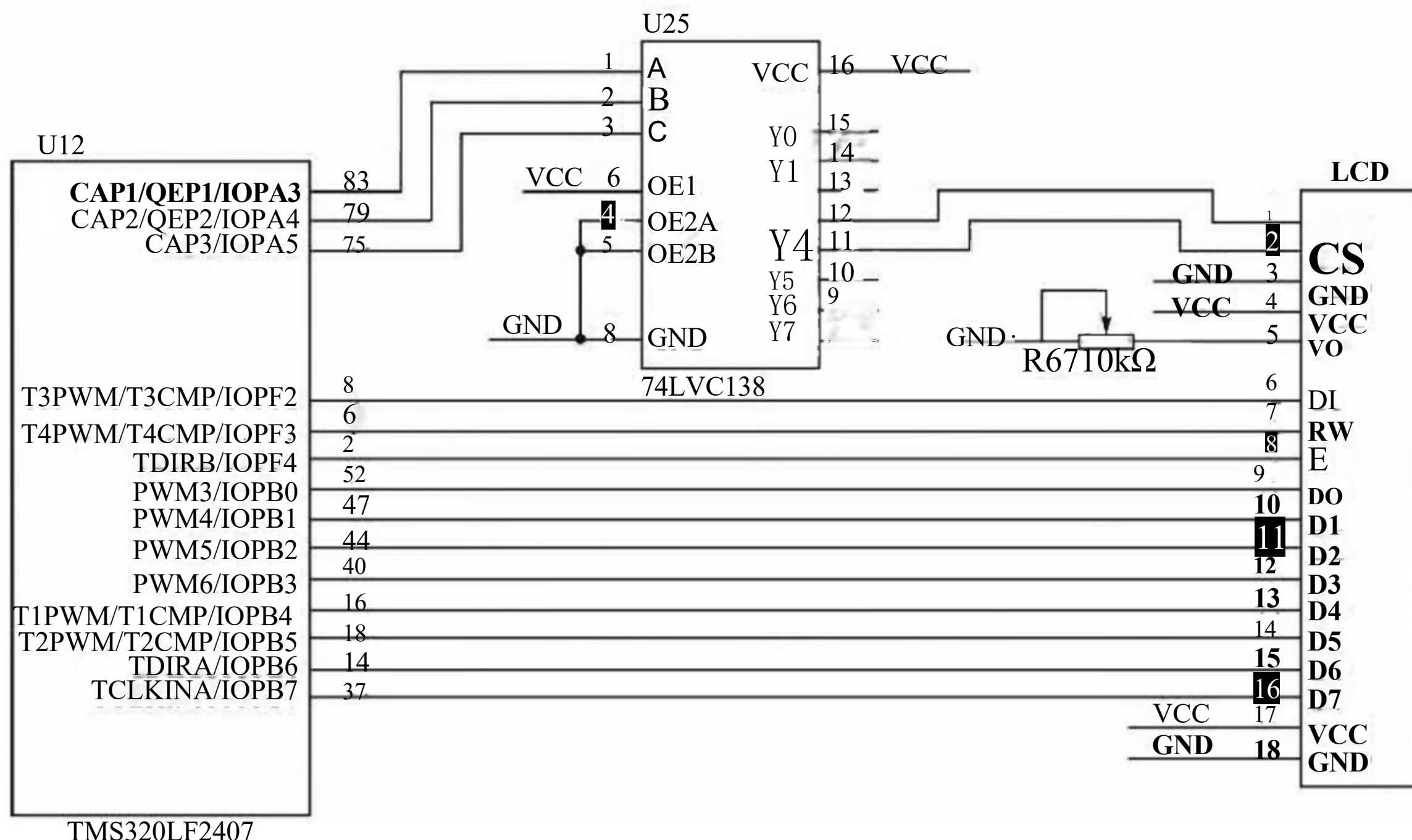


图14.2 F2407 与 MCG12864A8-3 模块的接口电路

14.4 字模软件的使用说明

字模软件可从 <http://robin.51.net/download.htm> 网站下载。网页打开后，单击“字模软件 zimo221.rar”即可下载。

在此对字模软件的使用作简单介绍。字模软件打开后，界面如图14.3所示。左边为操作界面，有6个选项，分别为基本操作、取模方式、修改图像、模拟动画、参数设置及退出程序；右上方为液晶显示区；右下方为文字输入区、点阵生成区和简介。

操作步骤如下：

- ① 单击“参数设置”进行参数设置；单击“文字输入区”进行字体设置，字体的大小可选为小四；单击“取模方式”选择纵向取模，另外选中字节倒序、保留及任何时候都加零。
 - ② 在文字输入区输入需要显示的汉字或数字，结束文字输入，按Enter键。
 - ③ 单击“取模方式”，选择C51格式，在点阵生成区即可获得所需要的点阵。
- 获得点阵后，可将其用于程序编程中。如有需要，可在基本操作中特将点阵保存起来。关于字模软件的使用，还有其他很多方面，在此就不一一介绍。

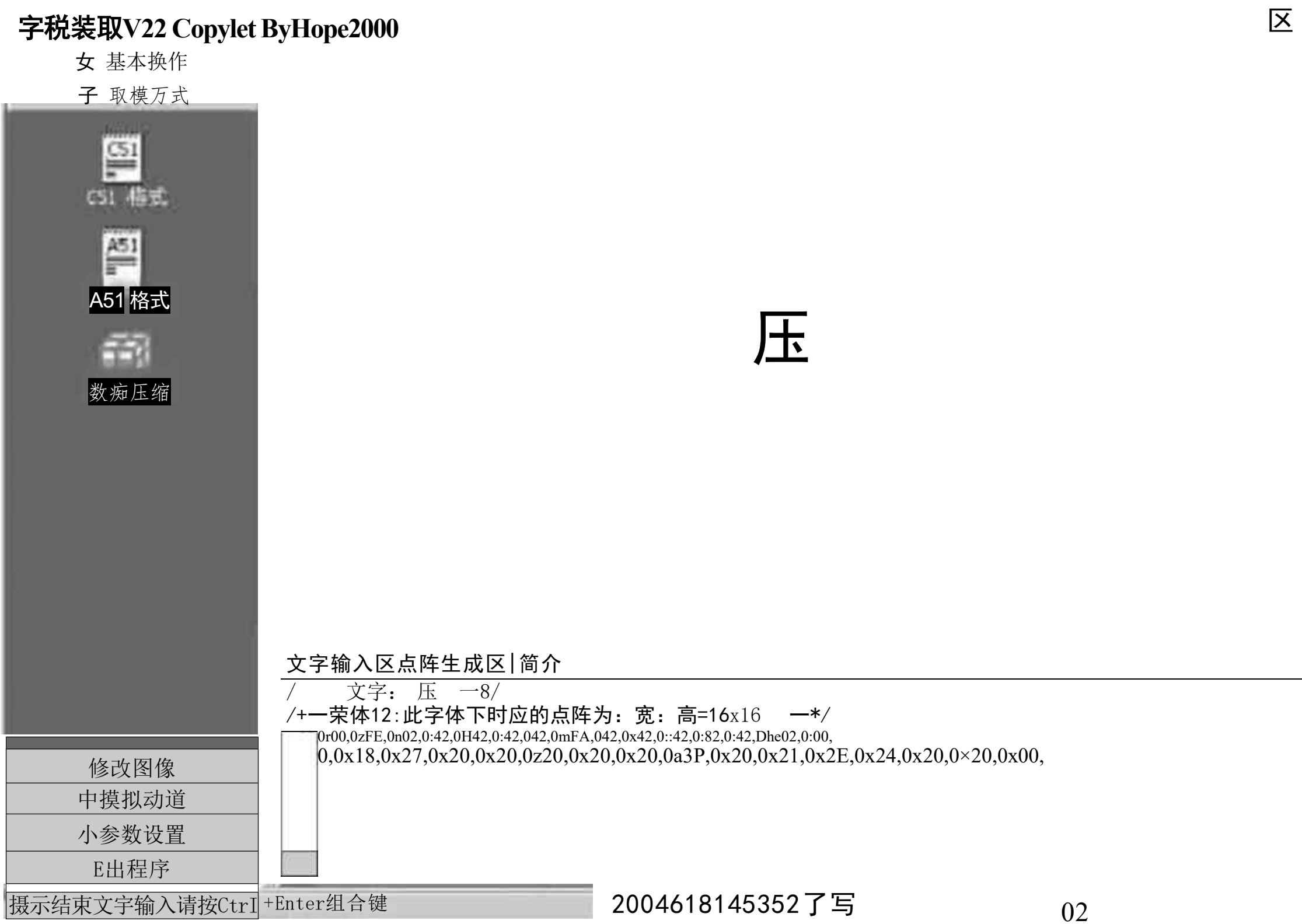


图14 . 3 字模软件的界面图

14.5 液晶显示模块指令系统

KS0108B 及其兼容控制驱动器的指令系统比较简单，共有7种，下面分别介绍，如表14.2~表14.8所列。

表14 . 2显示开/关指令

R/W	D/I	DB7	DB6	DB ⁵	DB4	DB3	DB2	DB1	DB0
0	0	0		1	1	1	1	1	1, /0

当DB0=1 时，LCD 显 示 RAM 中的内容；当 DB0=0 时，关闭显示。

表14. 3 显示起始行(ROW) 设置指令

R/W	D/I	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	0	1	1	显示起始行(0~63)					

该指令设置对应液晶屏最上一行显示 RAM 的行号。有规律地改变显示起始行，可使 LCD 实现显示滚动屏幕的效果。

表14.4 页 (PAGE) 设置指令

R/W	D/I	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	1	0	1	1	1	页号 (0~7)		

显示RAM 共64行，分8页，每页8行。

表14.5 列地址(Y Address)设置指令

R/W	D/I	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	1	显示列地址 (0~63)					

设置页地址和列地址，便唯一确定了显示RAM 中的一个单元。这样 MPU 就可用读、写指令，读出该单元中的内容，或向该单元写进一个字节数据。

表14.6 读状态指令

R/W	D/I	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
1	0	BUSY	0	ON/OFF	REST	1	1	1	1/0

该指令用来查询液晶显示模块内部控制器的状态。各参量含义如下：

BUSY 1：内部在工作； 0:正常状态。

ON/OFF 1：显示关闭； 0:显示打开。

RESET 1：复位状态； 0:正常状态。

在BUSY 和 RESET 状态时，除读状态指令外，其他指令均不对液晶显示模块产生作用。
在对液晶显示模块操作之前，要查询BUSY 状态，以确定是否可以对液晶显示模块进行操作。

表14.7 写数据指令

R/W	D/I	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	1	写数据							

表14.8 读数据指令

R/W	D/I	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
1	1	读显示数据							

读/写数据指令每执行完一次读/写操作列地址就自动增1。必须注意，进行读操作之前，

必须有一次空读操作，紧接着再读，才会读出所需单元中的数据。

说明：只有理解了以上7项指令，才能读懂液晶程序，并对液晶程序作修改，从而得到自己的液晶显示程序。

14.6 液晶显示程序

每一个液晶分为两块，每一块又由8页、64列组成，其中每页为8行。1个汉字为32个字节的数组，1个数字为16字节的数组。每一个字须分两页点亮。

虽然液晶程序看起来很长，但是认真阅读后会发现它很有规律，修改程序比较容易。只要多加一条 for 循环语句，就可多输出一个字，字的点阵可通过字模软件获得。

液晶显示的是采样回来的电压值，包含数字的显示。设置了一个包含0~9所有数字点阵的大数组，每完成一次 A/D 采样后，就将采样数字转换成点阵显示出来，这样保证了液晶显示的实时性。下面给出一个已经在模板上调试通过的液晶显示程序。

下面以图14.1的电路为例，给出LCD 的初始化及显示程序。程序用C 语言编写，注重模块化编程，具有很强的移植性。

源程序如下。

1. 所需的复位和中断向量定义文件 **vectors.asm**

```
.title      "vectors.asm"
.ref       c_int0,nothing
.sect      "vectors"
reset:     b      c_int0
int1:      b      nothing
int2:      b      nothing
int3:      b      nothing
int4:      b      _nothing
int5:      b      nothing
int6:      b      nothing
```

2. 主程序

```
//该程序利用C语言实现液晶的显示，字库以字的形式存储。该程序执行后，实验开发板上的液
/晶片上显示“重庆大学”
#include "F2407_c.h"
char table1 [300]=
0x08.0x08.0x0A.0xEA.0xAA.0xAA.0xAA,0xFF.0xA9.0xA9.0xA9.0xE9,0x08.0x08.0x08.0x00.
0x40.0x40,0x48,0x4B,0x4A.0x4A.0x4A,0x7F.0x4A,0x4A,0x4A,0x4B.0x48.0x40.0x40.0x00,
```

```

//重
0x00,0x00,0xFC,0x44,0x44,0x44,0x44,0xC5,0x7E,0xC4,0x44,0x44,0x44,0x44,0x44,0x00,
0x40,0x30,0x0F,0x40,0x20,0x10,0x0C,0x03,0x00,0x01,0x06,0x18,0x30,0x60,0x20,0x00,
//庆
0x20,0x20,0x20,0x20,0x20,0x20,0xA0,0x7F,0xA0,0x20,0x20,0x20,0x20,0x20,0x20,0x00,
0x00,0x80,0x40,0x20,0x10,0x0C,0x03,0x00,0x01,0x06,0x08,0x30,0x60,0xC0,0x40,0x00,
//大
0x40,0x30,0x10,0x12,0x5C,0x54,0x50,0x51,0x5E,0xD4,0x50,0x18,0x57,0x32,0x10,0x00,
0x00,0x02,0x02,0x02,0x02,0x02,0x42,0x82,0x7F,0x02,0x02,0x02,0x02,0x02,0x02,0x00,
//学
;
//屏蔽中断子程序
void inline disable()
{
    asm("setc    INTM");
}

//系统初始化子程序
void initial()

    asm("setc    SXM");
    asm("clrc    OVM");
    asm("clrc    CNF");
    asm("setc    INTM");
    *SCSR1=0x81FE;
    *WDCR=0x0E8;
    *IMR=0x0000;
    *IFR=0x0FFFF;
    *MCRC=*MCRC&0x0e3FF;
    *PFDATDIR=0x0FFe3;
    *MCRA=*MCRA&0x00c7;
    *PADATDIR=0x0FF00;
    disable();

}

//读液晶显示器的状态子程序，形参 E 代表处理的液晶的边(E1 或 E2 边)
void LCDSTA1(E)
int E;

int E1FLAG,i=0;
PBDATDIR=*PBDATDIR&0x0000;
//抑制符号位扩展
//累加器中结果正常溢出
//B0 被配置为数据存储空间
//禁止所有中断
//CLKIN=6 MHz,CLKOUT=4×CLKIN=24 MHz
//不使能看门狗，用软件禁止看门狗
//禁止所有中断
//清除全部中断标志，写1清0
//IOPF2、F3 和 F4 端口配置为一般的 I/O 功能
//IOPF2、F3 和 F4 端口配置为输出低电平
//配置IOPA[3~5] 和 IOPB[0~7] 为一般 I/O 口功能
//IOPA[3~4] 为输出，且全部为0
//总中断禁止
//IOPB 口为输入

```

```
if(E==1)
    *PADATDIR=0x0FF18;           //E1=1
else    *PADATDIR=0x0FF20;       //E2=1

*PFDATDIR=0x0FF08;              //LCD_RW=1
                                   //LCD_DI=0

while(1)

    *PFDATDIR+=PFDATDIR|0x0FF10; //LCD   E=1
    for(i=0;i<2;i++)

        *PFDATDIR=*PFDATDIR&0x0FFef; //LCDE=0
        E1FLAG=*PBDATDIR&0x0080;     //读液晶状态
        if(E1FLAG==0x0000)break;     //若液晶空闲，则停止等待
```

//对液晶显示器发指令子程序，形参 E 代表处理的液晶的边(E1 或 E2)

void TRANS1(TRANS,E) / 该函数采用形参和实参的方式传递数值，TRANS 为一个形参

```
int  TRANS,E;
(int    i=0;
LCDSTA1(E);           //判断液晶是否为忙
    *PBDATDIR+=PBDATDIR|0x0FF00; //IOPB 口为输出
if(E==1)    PADATDIR=0x0FF18;    //E1=1
else    *PADATDIR=0x0FF20;       //E2=1
    *PFDATDIR=0x0FF10;          //LCD_RW=0
                                   //LCD_DI=0
                                   //LCD_E=1

    *PBDATDIR=( PBDATDIR&0x0FF00)|TRANS; //需要写入的命令字送入数据线
    for(i=0;i<2;i++){
        *PFDATDIR=*PFDATDIR&0x0FFef; //LCD   E=0
```

//对液晶显示器写数据子程序，注意在此子程序中列地址没有加1，需要用户对其进行操作，形参E
//代表处理的液晶的边(E1 或 E2)

void WRITE1(WRITE,Yaddress,E)

//WRITE 代表需要写入的数据，Yaddress 代表需要的列地址

```
int  WRITE.Yaddress,E;
```

```
int    i=0;
```



```

TRANS1(Yaddress,E);           //设置列地址
LCDSTA1(E);                   // 查询液晶是否空闲
*PBDATDIR=*PBDATDIR|0xFF00;   //IOPB 口为输出
if(E==1) *PADATDIR=0xFF18;    //E1=1
else *PADATDIR=0xFF20;        //E1=0,E2=1
*PFDATDIR=0xFF14;            //LCD_RW=0
                                //LCD_DI =1
                                //LCD_E=1

*PBDATDIR=(*PBDATDIR&0xFF00) |WRITE; // 写入数据
for(i=0;i<2;i++){
    *PFDATDIR=*PFDATDIR&0xFFef;    //LCD_E =0
}

//开E1 显示子程序，形参E 代表处理的液晶的边(E1或 E2)
void DISP1(E)
int E;

int E1FLAG,i=0;
while(1)

TRANS1(0x03F,E);             //送出控制字
LCDSTA1(E);                   //判断液晶是否为忙
*PBDATDIR=*PBDATDIR&0x00ff;   //IOPB 口为输入
if(E==1)*PADATDIR=0xFF18;    //E1=1
else*PADATDIR=0xFF20;        //E1=0,E2=1
*PFDATDIR=0xFF18;            //LCD_RW =1
                                //LCD_DI =0
                                //LCD_E =1

for(i=0;i<2;i++){
    *PFDATDIR=*PFDATDIR&0xFFef;    //LCD_E =0;
    E1FLAG=*PBDATDIR&0x0020;        //读液晶状态
    if(E1FLAG==0x0000)break;        //如果液晶被打开，则退出循环
}

//E1 边清屏子程序，形参 E代表处理的液晶的边(E1 或 E2)
void CLEAR1(int E)

int PAGENUM,Yaddress;        //定义所需要的局部变量
for(PAGENUM=0x0B8;PAGENUM<=0x0BF;PAGENUM++)

```

```
TRANS1(PAGENUM,E);           //设置相应的页地址
for(Yaddress=0x40;Yaddress<=0x7F;Yaddress++)

WRITE1(0x00.Yaddress,E);      //向相应的列送0以清屏


//关E1 显示子程序，形参E 代表处理的液晶的边(E1或 E2)
void DISOFF1(int E)
《
    int      E1FLAG,i=0;
    while(1)
    {
        TRANS1(0x0E,E);      //送出控制字
        LCDSTAI(E);           //判断液晶是否为空闲
        PBDATDIR= PBDATDIR&0x00FF; //IOPB 口为输入
        if(E==1)      *PADATDIR=0x0FF18; //E1=1
        else          PADATDIR=0x0FF20;   /E1=0
        PFDATDIR=0x0FF18; //LCD_RW=1
                                //LCD DI=0
                                //LCDE=1

        for(i=0;i<2;i++)
        {
            +PFDATDIR=*PFDATDIR&0x0FFef; //LCD_E=0
            E1FLAG=xPBDATDIR&0x0020;      //读液晶状态
            if(E1FLAG==0x0020)break;       //如液晶关闭，则退出循环
        }
    }
}

```

7/LCD的 E1 初始化子程序，形参E 代表处理的液晶的边(E1 或 E2)

```
void LCDI INIT(int E)
{
    DISOFF1(E);           //关显示E1
    TRANS1(0x0A4,E);      //不静态显示驱动
    TRANS1(0x0A9,E);      //占空比为1/32
    TRANS1(0x0E2,E);      //复位
    TRANS1(0x0A0,E);      //ADC 选择正常输出
    CLEAR1(E);            Y/清 屏
    DISP1(E);             //开显示
}

```

```

//汉字显示子程序，入口为存储汉字库的地址、页地址和起始列地址
//和选中的 E1或 E2 的边数

```

```

DISPLAY1(n,E,PAGADD,Yaddress)

```

```

int  n,PAGADD,Yaddress,E;

```

```

//定义所需要的局部变量

```

```

{

```

```

    int  K;

```

```

    char TEMP;

```

```

    TRANS1(0x0C0,E);

```

```

//从第0行开始显示

```

```

    TRANS1(PAGADD,E);

```

```

//送需要显示的页地址

```

```

    n=n*32;

```

```

    for(K=0;K<16;K++,Yaddress++)

```

```

    {

```

```

        TEMP=table1[K+n];

```

```

        WRITE1(TEMP,Yaddress,E);

```

```

//送需要显示的数据字节

```

```

    }

```

```

    Yaddress=Yaddress-16;

```

```

//恢复列地址

```

```

    PAGADD=PAGADD+1;

```

```

//页地址加1

```

```

    TRANS1(PAGADD,E);

```

```

    for(;K<32;K++,Yaddress++)

```

```

    {

```

```

        TEMP=table1[K+n];

```

```

        WRITE1(TEMP,Yaddress,E);

```

```

//送需要显示的数据字节

```

```

    }

```

```

    TRANS1(0x0c0,E);

```

```

    PAGADD=PAGADD-1;

```

```

    TRANS1(PAGADD,E);

```

```

}

```

```

//主程序

```

```

main()

```

```

    initial();

```

```

//系统初始化

```

```

    LCD1INIT(1);

```

```

//LCD的 E1 边初始化

```

```

    LCD1INIT(2);

```

```

//LCD的 E2 边初始化

```

```

    DISPLAY1(0,1,0xBA,0x47);// 重

```

```

    DISPLAY1(1,1,0xBA,0x67);// 庆

```

```

    DISPLAY1(2,2,0xBA,0x47);// 大

```

```

    DISPLAY1(3,2,0xBA,0x67);// 学

```

```

    for(;;){}

```


//如若因干扰引起中断的误执行，则执行此中断服务程序

```
void interrupt nothing()
{
    return;
}
```

对程序的几点说明：

(1) 对液晶的初始化特别重要，初始化的顺序为：关显示 → 设置正常显示驱动 → 设置占空比 → 复位 → 选择 ADC → 清屏 → 开显示(具体方法见上面的程序)。初始化完成以后，用户即可按照程序中的方法，通过调用模块化的子程序，在液晶显示屏的任意位置显示汉字、字母或其他图形。

(2) 无论是对液晶的初始化还是在液晶的相应位置显示相应的图案，都可能涉及液晶不同的两边。为了节约程序空间和程序的编写量，程序中对两边(E1 及 E2 边)的操作都用同一个程序模块，而通过 C 语言灵活的参数传递的方法区别需要对哪一边进行操作。使用 LCD 底层的子程序库，可以模仿 LCDPO 很简单地实现其他信息的显示。只要设置相应的显示字符字库存储的首地址、需要操作显示的边数(1或2)、显示起始页及显示起始列即可。如果与键盘配合，则可显示很多信息，能很好地满足工业控制中对大量信息显示的要求。

第 15 章 串 行 EEPROM 的接口编程

256 KB(32 KB)串行EEPROM 24LC256 具有结构紧凑以及存储容量大等特点, 其I²C 接口简单, 操作方便, 特别适合控制系统一些重要参数的存储。本章给出了24LC256 与 F2407 DSP 的接口电路及其通用操作的应用程序。

24LC256 是美国 atmel 公司生产的CMOS I²C 接口的串行 EEPROM, 具有连续读、随机读、字节写和高达 8192 字节的页写方式。硬件写保护功能可将24LC256 当作串行ROM 使用。在I²C 总线上串行通信的最高时钟频率可达400 kHz, 最大的写周期时间为5 ms。此外还具有宽电压工作范围(+2.5~+5.5 V), 低功耗设计(在工作电压为5.5 V 时, 静态电流为100 nA, 最大写操作电流为3mA, 最大读操作电流为400 μ A), 以及 I/O 噪声抑制等功能。

15.1 24LC256 概述

1. 引脚说明

24LC256 有4种封装形式, 分别为8脚的PDIP、SOIC、TSSOP 和 MSOP。图15. 1为PDIP 封装的外形图(图中的24XX256 代表一个系列, 该系列的芯片封装都一样), 各引脚的功能说明如下:

- >VCC、VSS 为电源的正负极输入端。
- >SCL 串行时钟输入端, 提供 I²C 信号时钟。
- >SDA 串行数据I/O 复用脚。漏极输出, 在使用时必须加上拉电阻。
- >WP 硬件写保护引脚。当它接到 VSS 时, 可对整个存储空间进行读/写操作; 当它接 VCC 时, 禁止写操作, 但对读操作不影响, 相当于串行 ROM 功能。
- > A0、A1、A2 为多片24LC256 同时使用时用, 接相应的高低电平可以作为片选, 可连接到VSS 或 VCC, 在一条 I²C 总线上最多可以接8片 24LC256。

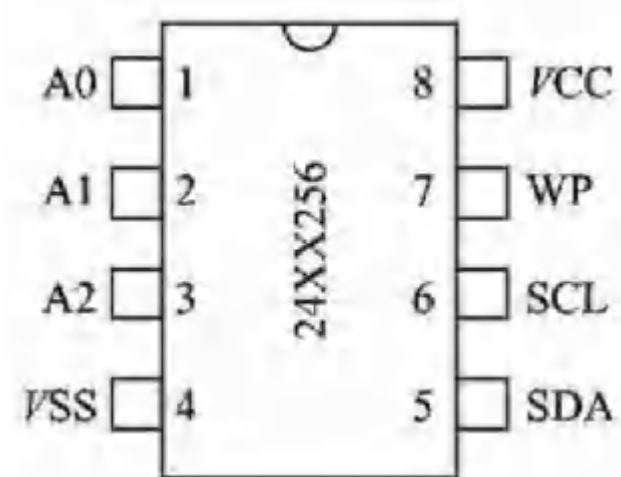


图15.1 24LC256 的封装示意图

2. 地址选择

24LC256 的地址选择分为器件地址选择和器件内地址选择。器件内地址选择是选中一块24LC256 器件内部的某一个地址单元；而器件地址选择是用控制字节中的3位A2A1A0 来完成的。若要选择 IC 总线上的某一个24LC256 器件，则该3位必须与该器件上 A0、A1 及 A2 三个引脚连接的逻辑电平一致。控制字节的结构为1010A2A1A0R/WB。高4位必须为1010, 这是24LC256 器件的控制码；A2A1A0 是器件地址选择位；R/W 为读/写选择位。当该位为1时，表示读操作；为0时，表示写操作。

3. 读写时序

在进行读/写操作之前，必须先设置开始状态。方法是命令 SCL 总线保持高电平，并在 SDA 总线上产生一个由高电平到低电平的跳变。而在每次通信结束时，都须设置结束状态，即将 SCL 总线保持高电平，在 SDA 总线上产生一个由低电平到高电平的跳变。只有设置了结束状态后，器件才退出通信状态。

SDA 总线上数据的改变必须在SCL 总线处于低电平状态下进行。在SCL 总线处于高电平期间，SDA 总线上的数据处于保持状态。

在通信过程中，接收数据方每次有效接收到8位数据后，都应向发送数据方，发送一个 SCL 时钟脉冲宽度的应答信号。具体的操作是：在进行写操作时，当24LC256 每次接收到8位数据(包括8位地址数据)后，要在第9个 SCL 时钟脉冲信号期间，将 SDA 总线拉为低电平作为对 CPU 的应答信号。在进行读操作时，除 CPU 读数据传送块的最后8位数据外，CPU 每次读完24LC256 所发送的8位数据后，都要在第9个SCL 时钟脉冲信号期间，将 SDA 总线拉为低电平作为对24LC256 的应答信号，这样24LC256 就可以继续发送数据。当CPU 读最后8位数据后，不发送应答信号(在第9个 SCL 时钟脉冲期间SDA 总线保持高电平)，这样在第9个 SCL 时钟脉冲后，24LC256 自动将 SDA 总线拉高，等待CPU 发出结束信号。

1) 按字节写时序

先设置开始状态，然后依次发送控制字节(控制字节的 R/W=0 表示写操作)、块内地址高字节、块内地址低字节以及待写的8位数据。24LC256 每接到8位数据，就利用第9个 SCL 时钟脉冲发送应答信号。这样CPU 在接收到4个应答信号后，设置结束状态。最后24LC256 在接收到结束信号后，启动EEPROM 内部写过程，将接收到的数据写入EEPROM 阵列。在进行内部写过程中，24LC256 对外部输入的任何信号都不作应答。

(2) 按页写时序

先设置开始状态，接着发送控制字节、块内地址高字节、块内地址低字节以及待写的第一字节数据。这些操作都与按字节写时序一样，只是在这之后不是设置结束状态，而是继续送出待写的的数据。这样24LC256 在每接收到8位数据后，其内部地址指针的低6位自动加1，而高位保持不变，所以每页包含64字节的数据。当内部地址指针达到本页的最高地址(低4位为

1111b) 后，就自动返回到本页的最低地址(低4位为0000b)，将已写过的数据覆盖。只有发送给24LC256 结束信号后，24LC256 才终止接收数据，开始启动内部写过程。

(3) 读当前地址的数据

如果要读24LC256 内部地址指针所指单元的数据，那么只要发送给24LC256 开始信号和控制字节(R/W 位为1)，24LC256 将产生应答并送出8位数据，而CPU 无须产生应答信号，但一定要发出结束信号。24LC256 发送完数据后，地址指针自动加1。

(4) 随机读取数据

随机读取数据必须设置所要读取数据的地址，这通过写操作来实现。在写操作中，当CPU 送出块内地址并接收到相应的应答信号后，马上设置为结束状态，来结束写操作。但24LC256 内部地址指针已经被设置。接着的操作与前面的“读当前地址的数据”相同。

(5) 连续读取数据

连续读取数据如同随机读取数据一样，首先将要连续读取数据的首地址设置为当前地址；在接收完第一字节数据后，不是发送结束信号，而是发送应答信号；接着24LC256 接收到应答信号后继续发送第二字节数据，依此类推。注意，在CPU 接收到最后一个字节的数据后，不再发送应答信号。在这种工作模式下，每读一个字节数据，24LC256 内部地址指针自动加1，该地址指针允许读取整个存储空间的数据。

15.2 24LC256 与 F2407 的硬件接口

图15.2为24LC256 与 F2407 的硬件接口，用F2407 的端口IOPA2、IOPEO 和 IOPDO 分别控制24LC256 的 SCL、SDA 和 WP。由于F2407 内部对 I/O 口进行了上拉，所以外部可以不再接上拉电阻。

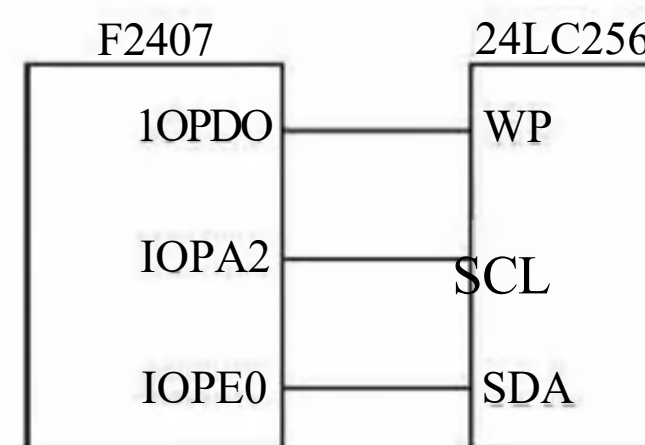


图 15.2 24LC256 与 F2407 的硬件接口

15.3 24 LC256 的应用编程

//该程序实现DSPF2407 与EEPROM 芯片24LC256 的接口编程

```

#include    "register,h"
int        source[4]={4,3,2,1};
int        result[4];
int        n,control;
//系统初始化子程序
void       sysinitial()

```

```
asm("setc INTM");           / 关闭总中断
asm("clrc SXM");             Z/ 抑制符号扩展
asm("clrc OVM");             / 累加器结果正常溢出
asm("clrc CNE");             //B0 被配置为数据空间
                               /CLKIN=6 MHz.CLKOUT=24 MHz
SCSR1=0x81FE;               //不使能看门狗
WDCR=0x0E8;                 //禁止所有中断
IMR=0x00;                   7/不使能所有的等待状态
WSGR=0x0FFEF;
```

//I/O 口初始化子程序

void initial()

//选择意义 IOPDO.IOPA2,IOPEO 为通用I/O 口

MCRA=MCRA&0xfffb;

NCRB=MCRB&0xfeff;

MCRC=MCRC&0xfffe;

PADATDIR=PADATDIR|0x0400;

//IOPA2 为输出

PDDATDIR=PDDATDIR|0x0100;

//IOPDO 为输出，且写允许

YZ启动IC 总线子程序

void I2CStart()

int k;

SCL=0

PADATDIR=PADATDIR&0xfffb;

IOPEO(SDA 为输出)

PEDATDIR=PEDATDIR|0x0100;

SDA=SCL=1

PEDATDIR=PEDATDIR|0x0001;

PADATDIR=PADATDIR 0x0004;

for(k=8;k>=0;k--)(k=k;

软件延时600 ns

//SDA=0

PEDATDIR=PEDATDIR&0xfffe;

for(k=8;k>=0;k--){k=k;

软件延时600 ns

SCL=0

PADATDIR=PADATDIR&0xfffb;

for(k=18;k>=0;k--) k=k;

/软件延时1400 ns

//SDA=1

```

PEDATDIR=PEDATDIR|0x0001;
}
//向I2C 总线写入数据子程序
void I2CSendByte(int data)

int flag,sz,k; //定义局部寄存器
//设置 IOPEO 为输出口
PEDATDIR=PEDATDIR|0x0100;
for(flag=0x0080;flag!=0x00;flag=flag/2)
{
//SCL=0
PADATDIR=PADATDIR&0xfffb;
for(k=0x06;k>=0;k--){k=k;}
sz=data&flag; //屏蔽掉相应的位
if(sz==0)
//如果相应的位为0, 则SDA=0
PEDATDIR=PEDATDIR&0xfffe;
else
//如果相应的位为1, 则SDA=1
PEDATDIR=PEDATDIR|0x0001;
//SCL=1, 在 SCL=1 期间数据线上的状态必须保持不变

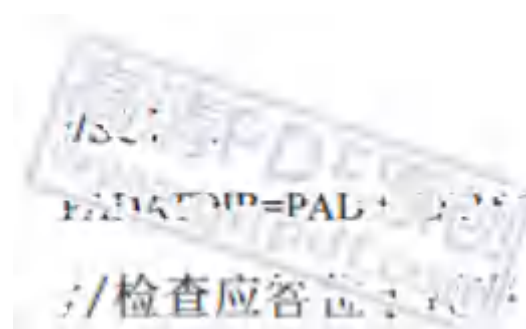
PADATDIR=PADATDIR|0x0004;
for(k=0x016;k>=0;k--){k=k;}

PADATDIR=PADATDIR&0xfffb;
//检查应答位

int I2CRECACK()
{
int k;

//SCL=0
PADATDIR=PADATDIR&0xfffb;
for(k=16;k>=0;k--){k=k;} //软件延时600 ns
//SCL=1, 在 SCL=1 期间数据线上的状态必须保持不变
PADATDIR=PADATDIR|0x0004;
for(k=3;k>=0;k--){k=k;} //软件延时600 ns
//IOPEO(SDA 为输入)
PEDATDIR=PEDATDIR&0xfeff;

```




```
for(k=3;k>=0;k--)          {k=k;          //软件延时600 ns

    SCL=0
    PADATDIR=PADATDIR&0xfffb;
    k=PEDATDIR&0x01;          //k 值存储应答位，若k=0. 则表示操作成功
    //若k=1. 则表示操作失败
    return(k);
```

```
}
```

/ 从I2C 总线读取数据

```
int    I2CRecByte()
```

```
{
```

```
int     k=0;
```

```
int     flag;
```

```
int     sz;
```

```
//IOPCO(SDA 为输入)
```

```
PEDATDIR=PEDATDIR&0xfeff;
```

```
SCL=0
```

```
PADATDIR=PADATDIR&0xfffb;
```

```
for(flag=0x0080;flag!=0x00;flag=flag/2)
```

```
{
```

```
1/SCL=1
```

```
PADATDIR=PADATDIR|0x0004;
```

```
sz=PEDATDIR&flag;
```

```
if(sz==flag)          k=k|flag;
```

```
PADATDIR=PADATDIR&0x0fffb;
```

```
}
```

```
return(k);
```

X

//对 I²C 总线产生应答

```
void    I2CAck()
```

```
{
```

```
int     k;
```

```
PEDATDIR=PEDATDIR    0x0100;
```

```
PEDATDIR=PEDATDIR&0xFFFE;
```

```
for(k=3;k>0;k--)          k=k;
```

```
PADATDIR=PADATDIR|0x0004;
```

```
for(k=7;k>0;k--)          (k=k;
```

```
PADATDIR=PADATDIR&0xFFFFb;
```

```
PEDATDIR=PEDATDIR    0x0001;
```

//取得应该读取的相应位

若读取的相应位为1, 则k 值的相应位置1

//SCL=0

//设置 TOPEO 为输出口

/SDA=0

//软件延时

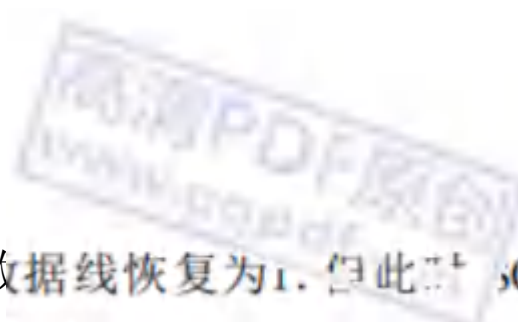
7/SCL=1

/ 软件延时

/SCL=0

//SDA=1. 数据线恢复为1, 但此时 SCL=0.SDA 在

/后来可以改变



```

}

//不对 I2C 总线产生应答
void I2CNoAck()
{
    PEDATDIR=PEDATDIR|0x0100;
    PEDATDIR=PEDATDIR|0x0001;
    PADATDIR=PADATDIR|0x0004;
    PADATDIR=PADATDIR&0x0FFb;
}

//停止 I2C 总线
void I2CStop()
{
    int k;
    PADATDIR=PADATDIR&0x0FFb;
    PEDATDIR=PEDATDIR|0x0100;
    PEDATDIR=PEDATDIR&0x0FFE;
    PADATDIR=PADATDIR|0x0004;
    for(k=7;k>0;k--)
    {
        PEDATDIR=PEDATDIR|0x0001;
        for(k=7;k>0;k--)
        {
            PADATDIR=PADATDIR&0x0FFb;
        }
    }
    //读数据子程序
    int read(array,control,n)
    int *array,control,n;
    {
        int dat;
        I2CStart();
        I2CSendByte(control);
        dat=I2CRECACK();
        if(dat==0){
            for(;n!=1;n--,array++)
            {
                dat=I2CRecByte();
                I2CAck();
                *array=dat;
            }
            dat=I2CRecByte();
            I2CNoAck();
        }
    }
}
//设置 IOPEO 为输出口
//SDA=1
//SCL=1
//SCL=0

//SCL=0
//设置 IOPEO 为输出口
//SDA=0
//SCL=1
//软件延时
//SDA=1
//软件延时
//SCL=0

//设置 I2C 总线的开始状态
//送出控制字节(R/W=1)
//检查应答位

//读取一个字节的的数据
//产生应答信号
//把读取的数据存入数组

//接收最后一个字节的数据
//不产生应答信号

```

```
*array=dat;
```

/把读取的数据存入数组

```
dat=0x00;
```

/dat 值赋0. 表示前面的操作成功

```
}
```

```
return(dat);
```

```
}
```

//写数据子程序

```
int write(array,control,n)
```

```
int array,control,n;
```

```
{
```

```
int dat;
```

```
PDDATDIR=PDDATDIR&0xFFFE;
```

//设置写允许, WP=0

```
for(;n!=0;n--,array++)
```

```
I2C_SendByte(*array);
```

7/写出一个字节数据

```
dat=I2C_RECVACK();
```

```
if(dat==1) break;
```

//如果无应答位. 则写数据失败, 终止写操作

```
}
```

```
PDDATDIR=PDDATDIR | 0x0001;
```

//写操作完成, 设置写保护: WP=1

```
return(dat);
```

```
}
```

//通用读/写24LC256 子程序, 入口参数为输入/输出缓冲区首地址 array

//24LC256 的块内地址address, 控制字节 control, 需要读/写的字节数 n

```
int EEPROM(array,address,control,n)
```

```
int array,address,control,n;
```

```
{
```

```
int dat;
```

```
I2C_Start();
```

//设置 I²C 总线的开始状态

```
dat=control&0xFFFE;
```

//清除控制字节的第0位

```
I2C_SendByte(dat);
```

//送出控制字节(R/W=0)

```
dat=I2C_RECVACK();
```

检查应答位

```
if(dat==0)
```

```
dat =address&0xFF00;
```

```
dat=dat>>8;
```

/dat=24LC256 的块内地址的高字节

```
I2C_SendByte(dat);
```

//送出24LC256的块内地址的高字节

```
dat=I2C_RECVACK();
```

//检查应答位

```
if(dat==0)
```

```
dat=address&0x00FF;
```

```
I2C_SendByte(dat);
```

送出24LC256 的块内地址的低字节

```
dat=I2C_RECVACK();
```

/检查应答位

```
if(dat==0)
```


第16章 实现快速傅里叶变换

16.1 快速傅里叶变换的原理

为了更好地理解在F240x DSP上实现傅里叶变换(FFT) 编程的原理和方法，有必要对FFT 的原理作一个粗略的介绍。

傅里叶变换是一种将时域信号转变为频域信号的变换形式。在频域分析中，频谱分析是信号分析的重要内容，它反映了系统性能的好坏。所谓信号的频谱，就是指信号的频率及对应的幅度值和相位，也可以分别称为幅度谱和相位谱。

傅里叶变换是数字信号处理中对信号进行分析时经常采用的一种方法。但是如果采用常规的傅里叶变换，则该算法的运算量会特别大，不适于需要高速运行的嵌入式控制系统，所以通常方法是采用快速傅里叶变换。下面将讨论FFT 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：

$$X(\omega) = \int_{-\infty}^{\infty} x(t) e^{-j\omega t} dt \quad (16.1)$$

式中计算出来的是信号 $x(t)$ 的连续频谱。但是，在实际的控制系统中能够得到的是连续信号 $x(t)$ 的离散采样值 $x(nT)$ ，其中 T 是采样周期。因此，需要利用离散信号 $x(nT)$ 来计算信号 $x(t)$ 的频谱。

假设经过采样得到了 N 点采样值 $\{x(nT), n=0,1,\dots,n-1\}$ ，那么其频谱采样的谱间距为：

$$\omega_0 = 2\pi/nT$$

可以推导出式(16.1)的离散形式为：

$$X(k\omega_0) = \sum_{n=0}^{N-1} x(nT) e^{-jk\omega_0 nT} = \sum_{n=0}^{N-1} x(nT) e^{-j\frac{2\pi}{N}kn} \quad (16.2)$$

令 $WN=e^{-j\frac{2\pi}{N}}$ ，并省略符号 w_0 和 T ，则上式可写为：

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn} \quad k=0,1,\dots,n-1 \quad (16.3)$$

式(16.3)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(16.3)的计算,可以得到N个频谱条,这就是离散傅里叶变换DFT。可以看出,DFT需要计算大约 N^2 次乘法和 N^2 次加法。当N较大时,这个计算量是非常大的。但是经过细致研究就可以看出蝶形因子 W_N 具有如下特性。

(1) 周期性: $W_N = W_N(N+m) = W_N$

(2) 对称性: $W_N^{(N-k)} = W_N^k W_N^{\frac{N}{2}} = -W_N^k$

利用蝶形因子 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,这样两个 $N/2$ 点DFT总的计算量只是原来的一半,即 $(N/2)^2 + (N/2)^2 = N^2/2$ 。这样的分解可以继续下去,将 $N/2$ 点DFT再分解为 $N/4$ 点DFT等。最小的分解点数称为基数(radix)。

基数为2的FFT就是最小变换为2点的DFT。对于 $N=2^m$ 点的DFT都可以基2的FFT来实现,这样其计算量可以减为 $(N/2)\log_2 N$ 个加乘运算,这就在很大程度上减少了DFT的计算量。需要特别指明的是,FFT不是DFT的近似计算,它们是完全等效的。实现FFT通常有如下两种方法:第一,按时间抽取(DIT)的FFT算法;第二,按频率抽取(DIF)的FFT算法。下面重点讨论DIT。

设 $N=2^m$ (m 为整数), 则:

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

将 $x(n)$ 分解为偶数(even)与奇数(odd)的两序列之和,上式可以写成:

$$X(k) = \sum_{n=\text{even}} [x(n)] W_N^{nk} + \sum_{n=\text{odd}} x(n) W_N^{nk}$$

当 n 为偶数时,令 $n=2r$; 当 n 为奇数时,令 $n=2r+1$ (r 为整数), 则:

$$\begin{aligned} X(k) &= \sum_{r=0}^{\frac{N}{2}-1} x(2r) W_N^{2rk} + \sum_{r=0}^{\frac{N}{2}-1} x(2r+1) W_N^{(2r+1)k} \\ &= \sum_{r=0}^{\frac{N}{2}-1} x(2r) (W_N^2)^{rk} + W_N^k \sum_{r=0}^{\frac{N}{2}-1} x(2r+1) (W_N^2)^{rk} \end{aligned}$$

由于 $W_N^2 = e^{-j2\pi/N} = W_{N/2}$, 将此结果代入上式, 得:

$$X(k) = \sum_{r=0}^{\frac{N}{2}-1} x(2r) W_{N/2}^{rk} + W_N^k \sum_{r=0}^{\frac{N}{2}-1} x(2r+1) W_{N/2}^{rk} \quad (16.4)$$

若记:

$$\left. \begin{aligned} G(k) &= \sum_{r=0}^{\frac{N}{2}-1} x(2r) W_{N/2}^{rk} \\ H(k) &= \sum_{r=0}^{\frac{N}{2}-1} x(2r+1) W_{N/2}^{rk} \end{aligned} \right\} \quad (16.5)$$

则式(16.4)可以改写为:

$$X(k)=G(k)+WH(k) \quad (16.6)$$

例如求 N 点的 DFT 被分解为求两个 N/2 点的 DFT, 即 H(k) 和 G(k)。以 N=8 的 DFT [x(n)] 为例, 利用式(16.6)则有:

$$\left. \begin{aligned} X(0) &= G(0) + W_8^0 H(0) \\ X(1) &= G(1) + W_8^1 H(1) \\ X(2) &= G(2) + W_8^2 H(2) \\ X(3) &= G(3) + W_8^3 H(3) \\ X(4) &= G(4) + W_8^4 H(4) \\ X(5) &= G(5) + W_8^5 H(5) \\ X(6) &= G(6) + W_8^6 H(6) \\ X(7) &= G(7) + W_8^7 H(7) \end{aligned} \right\} \quad (16.7)$$

根据式(16.5), G(k) 和 H(k) 都为 4 点的 DFT, 它们均以 4 为周期, 故有 $G(k+4)=G(k)$, $H(k+4)=H(k)$ 。再考虑 W 的对称性 $W_8^k = -W_8^{k+4}$ ($0 \leq k \leq 3$), 则式(16.7)可写为:

$$\left. \begin{aligned} X(0) &= G(0) + W_8^0 H(0) \\ X(1) &= G(1) + W_8^1 H(1) \\ X(2) &= G(2) + W_8^2 H(2) \\ X(3) &= G(3) + W_8^3 H(3) \\ X(4) &= G(0) - W_8^0 H(0) \\ X(5) &= G(1) - W_8^1 H(1) \\ X(6) &= G(2) - W_8^2 H(2) \\ X(7) &= G(3) - W_8^3 H(3) \end{aligned} \right\} \quad (16.8)$$

式(16.8)可以用信号流图来表示, 如图16.1所示。信号流图的形式如图16.1的右半部分所示。根据输入/输出数据的运算结构, 可把信号流图分为4个基本运算单元, 称为蝶形运算单元。图16.2(a)中由G(1)、H(1)、X(1)和X(5)所构成的结构就是一个蝶形运算单元。由图16.2可以看出, 蝶形运算单元有如下规律: 左方两节点为输入节点, 表示输入数值; 右方两节点为输出节点, 表示输入数值的叠加。蝶形运算是自左向右进行的。线旁标注的加权系数 W_8 和 $-W_8$ 与相应的输入数值作乘法运算, 即:

$$X(1)=G(1)+WH(1)$$

$$X(5)=G(1)-W.H(1)$$

X(1) 和 X(5) 取值都仅与G(1) 和 H(1) 有关, 因此把 X(1) 和 X(5)、H(1) 和 G(1) 称为对偶节点。计算对偶节点的数值, 如 X(1) 和 X(5), 仅须做一次复数乘法: $WH(1)$, 两次复数加法: $G(1)+W!H(1)$ 、 $G(1)-W!H(1)$ 。蝶形单元也可以画成图16.2(b)的形式。

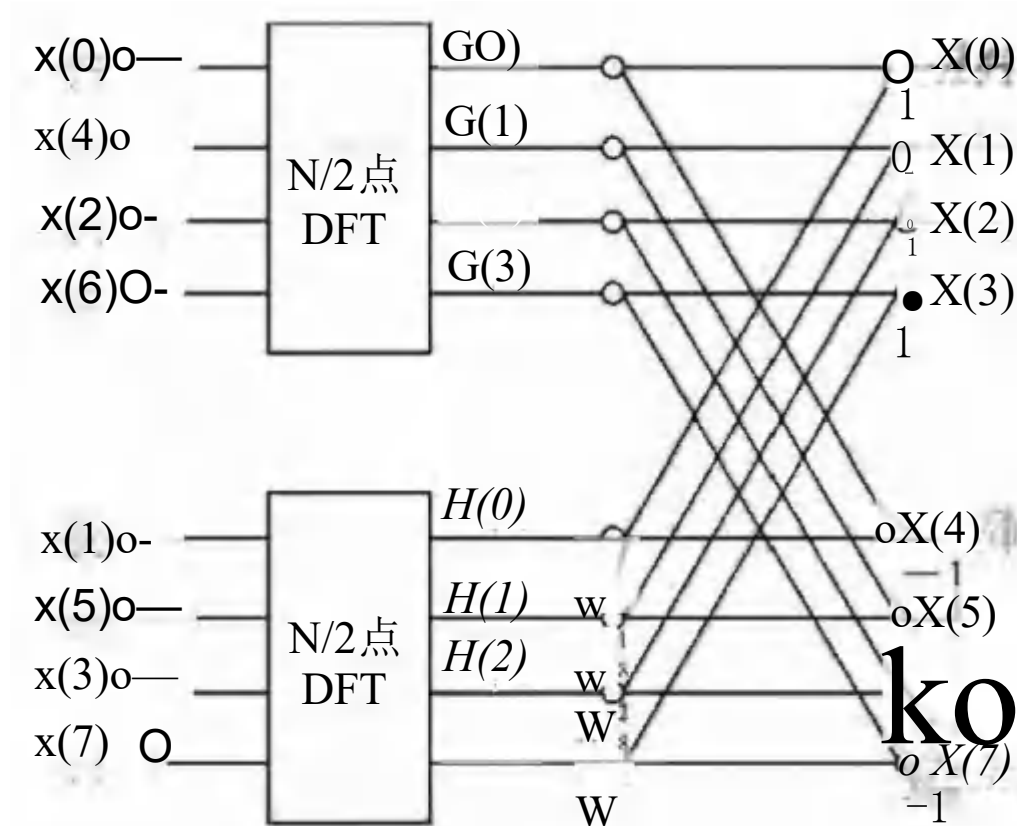


图 16.1 把 8 点 DFT 分解为两个 4 点 DFT 的信号流图

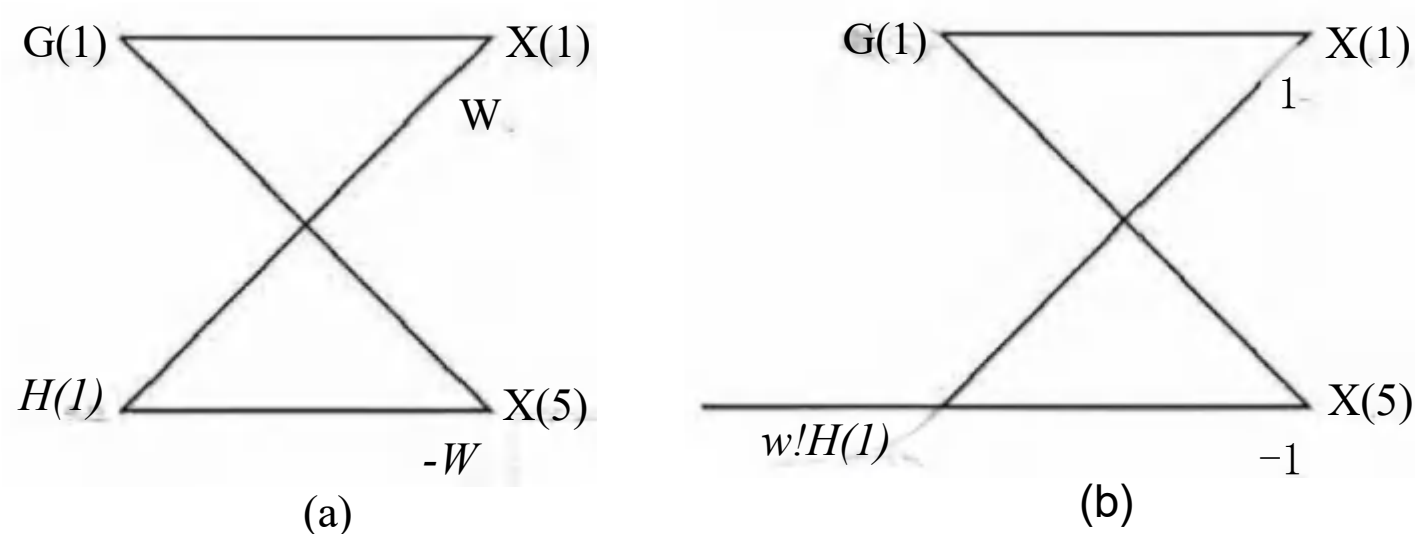


图 16.2 蝶形运算单元信号流图

$G(k)$ 是 4 点的 DFT, $G(k)$ 各点的数值的计算可由式 (16.5) 将 $x(2r)$ 按 r 的奇偶继续分组, 把 4 点 DFT 进一步分解成两个 2 点 DFT, 即:

$$G(k) = \sum_{r=0}^{\frac{N}{2}-1} x(2r)W_{N/2}^{rk} = \sum_{r=\text{even}} x(2r)W_{N/2}^{rk} + \sum_{r=\text{odd}} x(2r)W_{N/2}^{rk}$$

r 为偶数时, 记 $r=2l$; r 为奇数时, 记 $r=2l+1$ (l 均为正整数), 则 $G(k)$ 可以写成:

$$\begin{aligned} G(k) &= \sum_{l=0}^{\frac{N}{4}-1} x(4l)W_{N/2}^{2lk} + \sum_{l=0}^{\frac{N}{4}-1} x(4l+2)W_{N/2}^{(2l+1)k} \\ &= \sum_{l=0}^{\frac{N}{4}-1} x(4l)W_{N/4}^{lk} + W_{N/2}^k \sum_{l=0}^{\frac{N}{4}-1} x(4l+2)W_{N/4}^{lk} \\ &= A(k) + W^2 B(k) \end{aligned}$$

式中,

$$A(k) = \sum_{l=0}^{\frac{N}{4}-1} x(4l)W_{N/2}^{lk}, B(k) = \sum_{l=0}^{\frac{N}{4}-1} x(4l+2)W_{N/2}^{lk}$$

因设 $N=8$, 故 $A(k)$ 和 $B(k)$ 都是 2 点 DFT, 它们与 $G(k)$ 的关系为:

$$G(0)=A(0)+WB(0)$$

$$G(1)=A(1)+W^2B(1)$$

$$G(2)=A(0)-W^0B(0)$$

$$G(3)=A(1)-W^2B(1)$$

由 $A(k)$ 和 $B(k)$ 计算 $G(k)$ 的蝶形信号流图见图16.3(a)。

与 $G(k)$ 的分解类似, 也可以将 $H(k)$ 分解为 2 点 DFT 的组合, 即:

$$H(k) = \sum_{l=0}^{\frac{N}{2}-1} x(4l+1)W_{\frac{N}{2}}^{kl} + W_{\frac{N}{2}}^{k\frac{N}{4}} \sum_{l=0}^{\frac{N}{2}-1} x(4l+3)W_{\frac{N}{2}}^{kl} \\ = C(k) + W^2 D(k)$$

式中, $C(k) = \sum_{l=0}^{\frac{N}{2}-1} x(4l+1)W_{\frac{N}{2}}^{kl}$, $D(k) = \sum_{l=0}^{\frac{N}{2}-1} x(4l+3)W_{\frac{N}{2}}^{kl}$ 也都是 2 点 DFT。由 $C(k)$ 、 $D(k)$ 求 $H(k)$ 的蝶形信号流图见图16.3(b)。

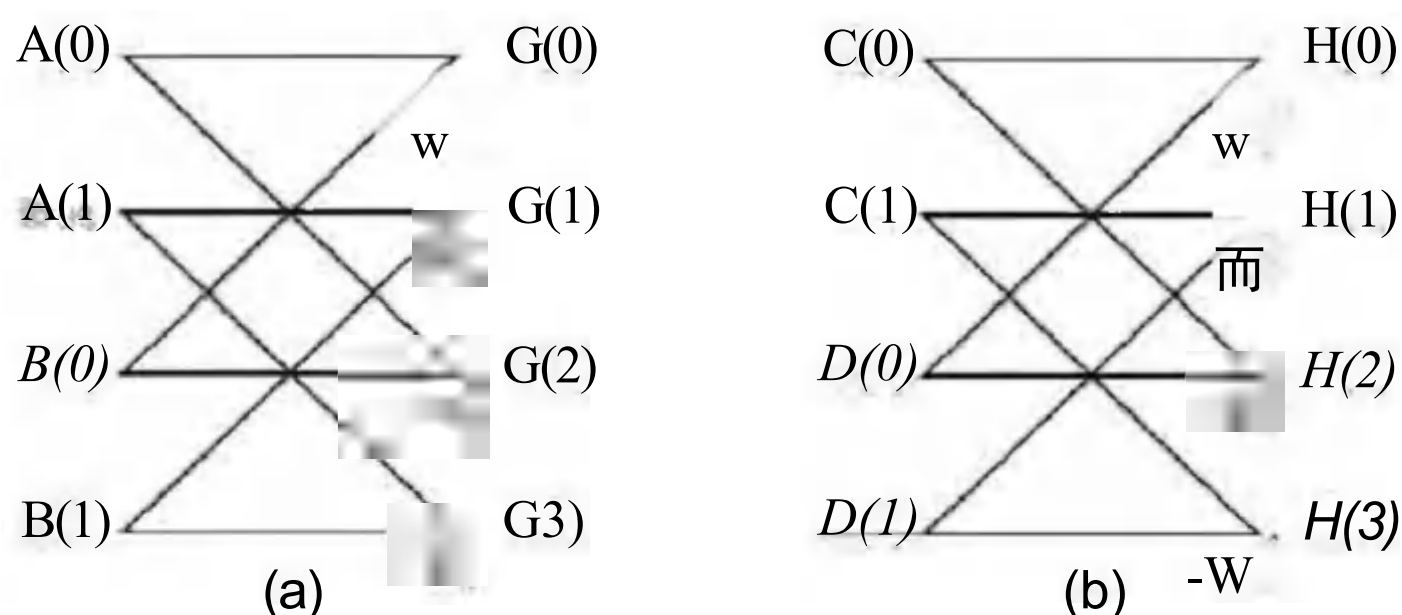


图16.3 4点DFT 分解为2点 DFT的蝶形信号流图

至此, $A(k)$ 、 $B(k)$ 、 $C(k)$ 和 $D(k)$ 都已经是 2 点 DFT。它们均是由原始数据 $x(n)$ 的两点通过一个蝶形运算计算而成。对于 $A(k)$, 有:

$$A(k) = \sum_{l=0}^{\frac{N}{2}-1} x(4l)W_{\frac{N}{2}}^{kl}$$

当 $N=8$ 时, $A(0)$ 和 $A(1)$ 可以写成:

$$A(0)=x(0)+Wgx(4)$$

$$A(1)=x(0)-Wgx(4)$$

这就是最简单的 DFT 运算了。

综上所述, 可得出 8 点 DFT 的蝶形信号流图如图16.4所示。其逐级分解的框图如图16.5所示。由图16.4所示的结构得出的算法即为快速傅里叶变换(FFT)。

由图16.4可见, 蝶形信号流图的输出序列 $X(k)$ 是按 k 由小到大顺序排列的, 而输入序列 $r(n)$ 则是按所谓的码位倒序排列的。

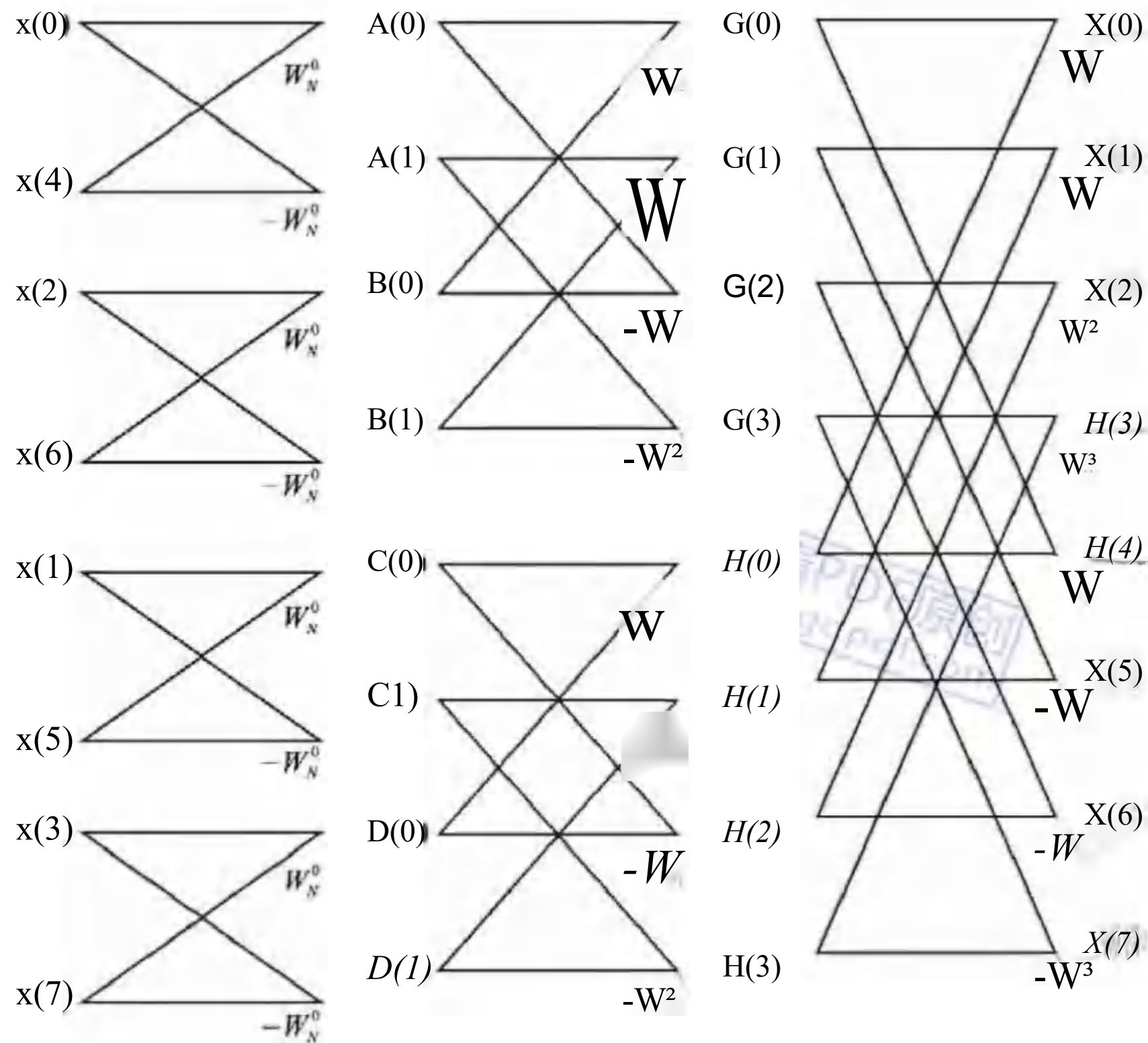


图16.4 8点DFT蝶形信号流图

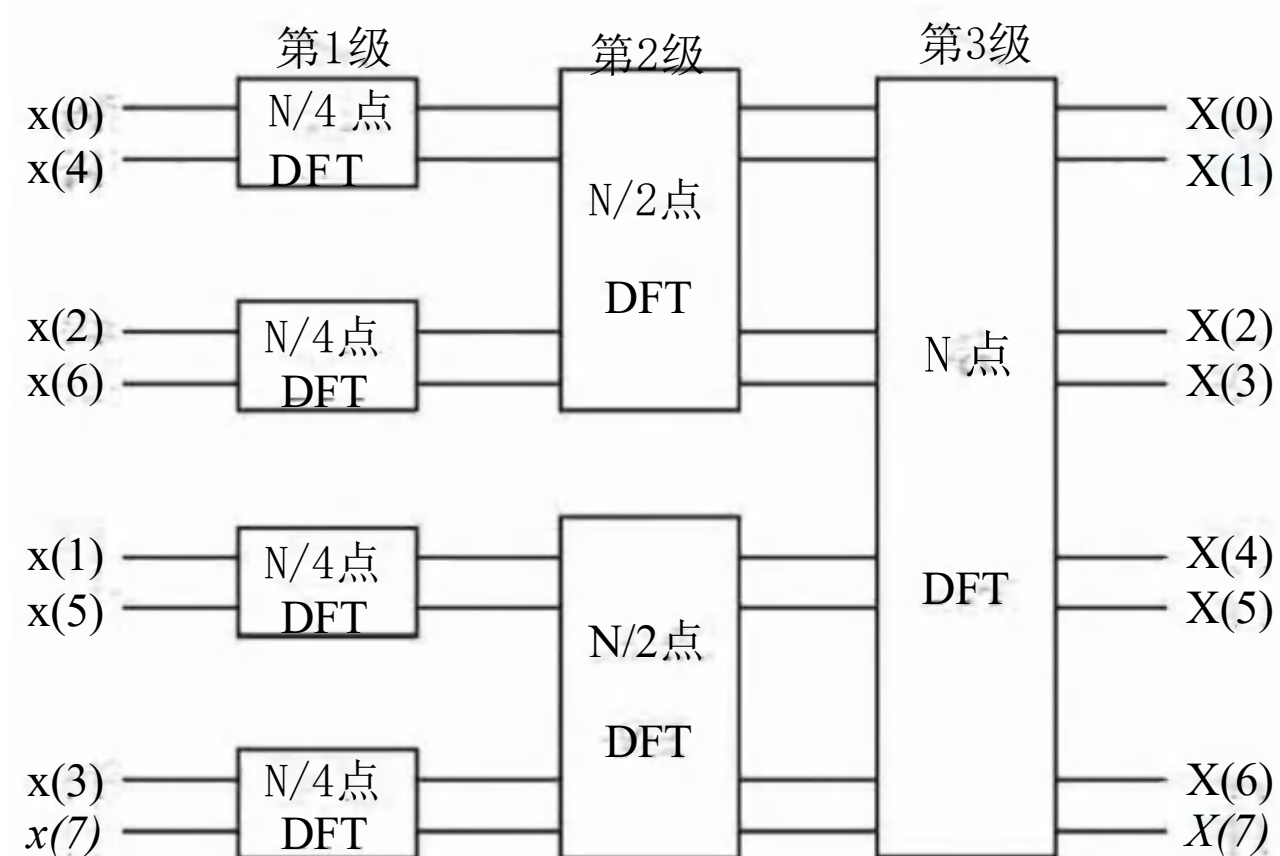


图16.5 8点DFT逐级分解运算框图

如果把十进制数换成二进制数，然后把这些二进制数的首位至末位的顺序颠倒过来再重新换算成十进制数，则该十进制数的排列即为码位倒序排列。N=8 时的自然顺序与码位倒序的比较见表16.1。

表16.1 自然顺序与码位倒序 (N=8)

十进制数	二进制数	二进制数的 码位倒序	码位倒序后 的十进制数	十进制数	二进制数	二进制数的 码位倒序	码位倒序后 的十进制数
0	000	000	0	4	100	001	1
1	001	100	4	5	101	101	5
2	010	010	2	6	110	011	3
3	011	110	6	7	111	111	7

码位倒序的排列规律是由FFT 的算法决定的。如果输入按自然顺序排列，则输出就会变成码位倒序排列；如果输入、输出均要求按自然顺序排列，则蝶形流图的形状会发生扭曲，造成不能即位运算或计算机内存增加等新问题。在控制系统中，一般都令输入为码位倒序，而输出为自然顺序。

上述 FFT 算法是把时间序列 $x(n)$ 按 n 的奇偶分组分解后再来计算的，又称为按时间抽取(DIT) 的 FFT 算法。与此对应，也可以把 $X(k)$ 按 k 的奇偶分组分解后再来计算DFT，称为按频率抽取(DFT) 的 FFT 算法。这两种方法的推导过程相仿，计算工作量是一样的，仅仅是算法的结构有所不同。按频率抽取的 FFT 算法请读者参考相关资料。

以上介绍了8点FFT 的实现过程。对于其他 $N=2^m$ (m 为整数)点FFT 的实现过程与8点是完全类似的，只是在图16.5的基础上增加了运算的级数。

16.2 快速傅里叶变换的DSP 实现

FFT 较之 DFT 的计算量减少了许多，但 FFT 要做到多点(N 较大)和实时运算，对于普通的单片机来说，还是一件比较困难的事情。一方面，FFT 需要对原始自然序列进行码位倒序排列；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 控制器就是针对这些需求而设计的，其有这样的指令和运算能力。 DSP 控制器特有的反序间接寻址，就是专为 FFT 算法而设计的。其他的间接寻址方式还可以实现增/减1或增/减一个变址量，这就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 算法较普通的单片机要容易许多。下面主要讨论它在DSP 控制器F2407 上的具体实现方法。

前面的分析过程重点结合图16.4和图16.5可以得知， $N=2^m$ 点的FFT 运算是非常有规律的，可通过下面3个循环来完成。



For L = I

Fa To2

Step 2

j=i+2-1

$$\left. \begin{aligned} \bar{x}(i) &\leftarrow \bar{x}(i) + W_{2^L}^k \bar{x}(j) \\ \bar{x}(j) &\leftarrow \bar{x}(i) - W_{2^L}^k \bar{x}(j) \end{aligned} \right\} \quad (16.9)$$

Next i

Next k

Next L

上面语句中的 m 为图 16.5 中所示的 FFT 运算的级数； k 为每一级运算的每个子 DFT 运算中蝶形运算的个数； i 为每一级运算过程中子 DFT 运算的个数（子 DFT 运算即为图 16.5 中每个方框内的运算，如在该图第 1 级运算中共有 4 个子 DFT 运算，每个子 DFT 运算中有 1 个蝶形运算，在第 2 级运算中共有两个子 DFT 运算，每个子 DFT 运算中有两个蝶形运算）。这是利用伪代码的形式来说明 FFT 运算的规律。在用实际的编程语言实现 FFT 算法时，要注意式 (16.9) 需要通过中间变量进行缓存，并且 $x(i)$ 、 $x(j)$ 和 W_2 是复数形式。在第 1 次循环时式 (16.9) 中的 $x(i)$ 的值是原始序列 $x(n)$ 经过码位倒序排列后的数据，经过上面的三重循环后最后放在 $x(i)$ 中的数正好对应 $X(k)$ 的值。

考虑到式 (16.9) 是复数运算，令：

$$x(i) = R + jI; \quad x(j) = R + jI, \quad (16.10)$$

$$\left. \begin{aligned} W_{2^L}^k &= \cos \alpha_{Lk} - j \sin \alpha_{Lk} \\ \alpha_{Lk} &= \frac{2\pi k}{2^L} = \frac{2\pi}{N} 2^{m-2} k \quad L = 1, 2, \dots, m; k = 0, 1, \dots, 2^{L-1} - 1 \end{aligned} \right\} \quad (16.11)$$

则式 (16.9) 可以化为：

$$\left. \begin{aligned} XT &= \bar{R}_j \cos \alpha_{Lk} + \bar{I}_j \sin \alpha_{Lk} \\ YT &= -\bar{R}_j \sin \alpha_{Lk} + \bar{I}_j \cos \alpha_{Lk} \\ \bar{R}_i &= \bar{R}_j + XT \\ \bar{I}_i &= \bar{I}_j + YT \\ \bar{R}_i &= \bar{R}_i - XT \\ \bar{I}_i &= \bar{I}_i - YT \end{aligned} \right\} \quad (16.12)$$

上述过程是实现一个蝶形运算的迭代过程。为了用 DSP 控制器来实现上述的过程，还须考虑数据的大小，以免发生溢出。假设原始时间序列 $x(n)$ 已经进行了归一化，即为 Q15 的格式（最高位为符号位，其余是小数位），那么可以推出，式 (16.9) 或式 (16.12) 可能的最大值为：

$$1 + \sin 45^\circ + \cos 45^\circ \approx 1 + 1.4142 = 2.4142$$

这将超出Q15 的格式范围。因为很多情况下是实数的FFT, 此值不超过2, 因此在每一级用因子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 $N=2^m$ 。

式(16.12)的运算涉及到正弦函数和余弦函数, 因此需要预先制作好正弦和余弦表。为了容易实现查表并节省存储容量, 在式(16.11)中令 $L=m$, 并按如下规律存放此表:

$$\begin{aligned} &\text{For } k=0 \quad \text{To } 2^m-2-1 \\ &\quad \cos\left(\frac{2\pi k}{N}\right); \sin\left(\frac{2\pi k}{N}\right) \\ &\text{Next } k \end{aligned} \quad (16.13)$$

可以验证, 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。

分析式(16.9)的循环可知, 第1层循环的次数为 m ; 第2层循环的次数依次为 2^{m-1} ($L=1, 2, \dots, m$); 第3层循环的次数依次为 2^{m-2} ($L=1, 2, \dots, m$)。

对 DSP 的开发可利用汇编语言或C语言。在对FFT 算法进行开发时, 也可采用C语言。但是FFT 的运算量较大, 特别是当点数 N 较大, 且用C语言开发不能直接利用DSP 控制器特有的反序间接寻址等优越的特性时, FFT 运算的实时性就不理想。综合考虑各方面的原因, FFT 算法应采用汇编语言开发。而DSP 的主程序最好用C语言进行开发。因此, 下面给出的FFT 算法程序, 其运算主体是用汇编语言编写的, 但是在各子程序的入口和出口都考虑了与C语言的兼容性, 使这个子程序能够直接被C语言调用。并且简单修改该程序, 可以分别实现32点、64点、128点的FFT 运算。程序中64点和128点的数据也已给出, 供读者使用。

以下是FFT 的程序代码:

(1) 主程序

```
#include "f2407_c.h"
#include "math.h"
#define N 32 //FFT 变换的点数
extern void fft(void);
extern void resave(void);
interrupt void phantom(void);
void sysinit(void);
extern int input[2*N]; // 输入数据的存储数组
int indati[N]=(0);
//
// 128点FFT所需的数据
//采样函数:  $x=1/4+1/4\cos(3 \times 2 \times \pi \times f \times t)+1/4\cos(6 \times 2 \times \pi \times f \times t)+1/4\cos(9 \times 2 \times \pi \times f \times t)$  七);
```



```
/*int      indatr[N]=(16384,14416.10270      ,7762      .9551      ,14651      .18713      ,17034
      8192      -4387      , -14222. -15834, -8304
      11049      ,7843      ,9064      ,13102      ,16149      ,15469.11668      ,8192      ,8452
      12807      .17801      ,18418      ,11818      , -0156      , -1165. , -1165. -1165. , -0156
      11818      .18418      .17801      12807      8452      .8192      ,11668      15469      .16149
      13102      .9064      ,7843      .11049      ,16383      ,18957      ,14810      ,4123      , -8304
      -15834      . -14222, -4387      .8192      ,17034      .18713      ,14651      .9551      ,7762
      10270      .14416      , ;*
```

//128点FFT的 sin 和 cos 值存储表
//--74

```
/*const      int      sintab[N]={0×07fff,0x0,0×07fd9,0x0f9b9,0×07f62,0×0f375,0×07e9d,0x0ed38,
      0×07d8a      .0x0e708      ,0×07c2a, 0x0e0e7      .0×07a7d      .0x0dad8      ,0×07885      ,0x0d4e1      .
      0x07642      ,0x0cf05      .0x073b6      ,0x0c946      ,0x070e3      .0x0c3aa      ,0x06dca      ,0x0be32
      0x06A6C      ,0x0B8E4      0x066CE      .0x0B3C1      ,0x062F1, 0x0AECD      ,0x05ED6      ,0x0AAOC
      0x05A81      .0x0A57F      .0x055F4.0x0A12A      .0x05133      .0x09DOF      .0x04C3F      .0x09932
      0x0471C      ,0x09594      ,0x041CD      ,0x09237      ,0x03C56      ,0x08F1F      ,0x036B9      ,0x08C4B
      0x030FB      ,0x089C0      ,0x02B1E      ,0x0877D      ,0x02527      ,0x08584      ,0x01F19      ,0x083D7
      0x018F8      ,0x08277      ,0x012C7      ,0×08164      ,0x00C8B      ,0x0809E      ,0x00647      ,0x08029      ,
      0x00000      ,0x08001      .0x0F9B9      ,0x08029      0x0F375      ,0x0809F      ,0x0ED39      .0x08164
      0x0E708      ,0x08277      .0x0EOE7      ,0x083D7      ,0×0DAD9      ,0x08584      ,0x0D4E2      ,0x0877D      ,
      0x0CF05      ,0x089CO      ,0x0C947      ,0x08C4B      ,0x0C3AA      ,0×08F1F      ,0x0BE33      ,0x09237
      0x0B8E4      0x09594,0x0B3C1      ,0x09932      ,0x0AECD      ,0x09DOF      ,0x0AAOC      ,0x0A12A
      0x0A57F      ,0x0A57F      .0x0A12A,0x0AAOC      .0x09DOF      ,0x0AECD      ,0x09932      .0x0B3C1
      0x09594      ,0x0B8E4      .0x09237      ,0x0BE33      ,0x08F1F      ,0x0C3AA      ,0x0BC4B      ,0x0C947
      0x089C0      .0x0CF05      ,0x0877D      ,0x0D4E2      ,0x08584      .0x0DAD9      ,0x083D7      ,0x0EOE7
      0x08277      ,0x0E708      0x08164      ,0x0ED39,0x0809F      ,0x0F375      ,0x08029      ,0x0F9B9
      ;
```

// 64 点FFT 的 sin 和 cos 值存储表

```
1      -      -
第 const      int      sintab[N]=(0x7FFE      ,0x0000      ,0x7F61      .0xF375      ,0x7D89      ,0xE708      ,
      0x7A7C      .0xDAD9      .0x7640      .0xCF05      .0x70E1      .0xC3AA      .0x6A6C      ,0xB8E4      ,
      0x62F1      ,0xAECD      ,0x5A81      0xA57F      ,0x5133      ,0x9DOF      ,0x471C      .0x9594      ,
      0x3C56      0x8F1F      0x30FB      ,0x89C0      0x2527      .0x8584      ,0x18F8      .0x8277      ,
      0x0C8B      .0x809F      ,0x0000      ,0x8001      0xF375      .0x809F      .0xE708      .0x8277      ,
      0xDAD9      .0x8584      .0xCF05      ,0x89C0      .0xC3AA      .0x8F1F      ,0xB8E4      .0x9594      ,
      0xAECD      ,0x9DOE      ,0xA57F      ,0xA57F      ,0x9DOF      .0xAECD      ,0x9594      ,0xB8E4
```



```
0xC3AA, 0x89C0, 0xCF05, 0x8584, 0xDAD9, 0x8277, 0xE708,
0x809F, 0xF375, };*/
//--
// 3 2 点 FFT 的 sin 和 cos 值存储表
// - - - - -
const int sintab[N]={
    0x7FFE, 0x0000, 0x7D89, 0xE708, 0x7640, 0xCF05, 0x6A6C, 0xB8E4,
    0x5A81, 0xA57F, 0x471C, 0x9594, 0x30FB, 0x89C0, 0x18F8, 0x8277,
    0x0000, 0x8001, 0xE708, 0x8277, 0xCF05, 0x89C0, 0xB8E4, 0x9594,
    0xA57F, 0xA57F, 0x9594, 0xB8E4, 0x89C0, 0xCF05, 0x8277, 0xE708,
};
extern int table128[];
extern int nom;
main()
{
    int i;
    double x=0,y;
    nom=1;
    sysinit();
    for(i=0;i<=255;i++) input[i]=0;
    resave();
    *PCDATDIR=(*PCDATDIR&0xFF00)|0x01;
    fft();
    *PCDATDIR=*PCDATDIR&0xFF00;
}
void interrupt phantom(void)
{
    return;
}
void sysinit(void)
{
    *SCSR1=0x81FE;
    *WDCR=0x0E8;
    *IFR=0xFF;
    *IMR=0x0;
    WSGR=0;
    *MCRB=0;
    *PCDATDIR=0x100;
}
```

(2) 反序排列子程序

```
.def    _resave
2 / 基 2 时间抽取的128点FET算法需要定义的各量
// Nset      128
基      2 时间抽取的64点 FET 算法需要定义的各量
Nset      64
H 基 2 时 间 抽 取 的 3 2 点FFT算法需要定义的各量
N      ,set
      global    _resave
      ,global    _input
      :global    indatr
      .global    indati

resave:

// 与C语言兼容的代码部分

      POPD      * 于
      SAR      ARO,*+
      SAR      AR1,
      LAR      ARO,+ , AR3

/ 反序排列程序部分
//
      LAR      AR2,#_input
      LAR      AR3.# indatr
      LAR      ARO,#N
      LAR      AR4,#( N-1)
RESAV1
      LACC      若 + , 0 ,AR2
      SACL      *BRO+, AR4
      BANZ      RESAV 1 , - , AR3

/- ..... -
/ 与C语言兼容的程序代码部分

      MAR      关 .ARI
      SBRK      #2
      LAR      ARO,-
      PSHD      关
      RET
```

(3)FFT 应用子程序

```
//--
//函数名: void fft(void)
//功能: 实现32、64或128采样点的快速傅里叶变换
//入口条件:
//sintab          存放 FFT 运算中用到的 sin 和 cos 函数值
//_input          存放 FFT运算中用到的数据, 包括实部和虚部, 按二进制反序排列
//              注意: 由于 “* BRO+” 间接寻址方式, 地址有特殊的要求,
//              所以最好将数组 _input 放置在一个独立的块中, 如 B1 块
//_nom            当 _nom=0 时, 本函数将不对运算结果进行归一化, 反之, 将对每
//              一步运算结果进行归一化处理, 避免溢出, 但是, 它会使运算精度降低
//N              常数, 参与FFT运输的点数, 用户可根据需要选择, 例如, 需要进行128点
//              FFT 时, 请在本函数中作出如下选择:
//              N      ,set      128
//              M      set      7
//              依此类推
//出口条件:
//_input          存放 FFT的运算结果
//本函数可供 C 调用, 请用户在 C 主程序中作以下声明:
//extern          void      fft(void);
//const          int      sintab[N]={...};          N 为128、64或32
//extern          int _input[2×N];
//extern          int nom;
//--
.def _fft
//基2时间抽取的128点FFT算法需要定义的各量
//N          .set      128          //点数
//M          .set      7          //N=2**M
//基2时间抽取的64点FFT算法需要定义的各量
//N          .set      64          //点数
//M          .set      6          //N=2×*M
//基2时间抽取的32点FFT算法需要定义的各量
N          .set      32          //点数
M          .set      5          //N=2**M
_input     .usect      “.data0”,2*N      //输入数据
//          .bss      sintab,N          //sin 和 cos 函数的存储表
          .bss      nom,1          // 当 nom=1 时, FFT需要归一化处理, 为0时则不需要
          .global    fft
          .global    sintab
```



```
global input
.global nom
.text

fft:

// 与 C 语言兼容的代码部分
I/-
    POPD      +          7/ 存储返回地址 ADDRESS
    SAR       AR6,      + // 存储 AR6
    SAR       AR7,*      / 存 储AR7
    SAR       ARO,++     // 存储 ARO
    LAR       AR1,*      器堆栈分布情况: ADDRESS/AR6/AR7/ARO/AR1
    LAR       AR1,*      / ARP=AR1,AR1:AR1
    LAR       AR1,*      /ARP=AR3
    LAR       AR3,*0+,AR3 /AR3:FP,SP=SP+size(size=frame 的长度)
    LAR       AR2,*      ARP=AR3
    LAR       AR2,*      AR2:AR1
    LAR       AR7,#nom   /AR7 指 向 nom

1 -
1/初始化一些寄存器
/ - --
    SPLK#(N-1),*+
    SPLK#(M-1),*      +
    SPLK#1,      + , AR2

//FFT 运算处理部分
//=
    SETCOVM      // 使能溢出模式
    SETCSXM      // 符号扩展使能
    SPM1         / PREG寄存器的输出左移一位
                //自动将两个Q15 相乘后化为 Q15 的 格式

    LACC      * + .AR3
    ADD      #1
    SACL      +. 1, AR2      /IW=2*(N+1)
```

```

//堆栈分布情况: ADDRESS/AR6/AR7
//ARO/N/M/ID/IW/Y
//ARP:AR2,AR2:M,AR3:Y
//AR5=M,ARP:AR2,AR2:ID
//AR3:Y,AR5=M

LAR    AR5,*    +
LOOP3
LAR    AR6,#_input    //AR6:input→Ri,ARP:AR2
//AR2:ID,AR3:Y,AR5=M,AR6:input

LACC    *,1
SACL    *+    //ID=ID*2,ARP:AR2,AR2:IW,
//AR3:Y,AR5=M,AR6:input

LACC    *,15
SACH    *    //IW=IW/2
LACC    * -, 15, AR3
SACH    * +, AR2    //C2=IW/2
//堆栈分布情况: ADDRESS/AR6/AR7/ARO/N/M/ID/IW/C2/Y
//ARP:AR2,AR2:ID,AR3:Y,AR5=M,AR6:input
LAR    ARO,*    //ARO=ID
LOOP2
LAR    AR4,#_sintab    //AR4:sintab
//ARP:AR2,ARO=ID,AR2:ID,AR3:Y,AR4:sintab,AR5=M,AR6:input
LACC    * +, 15, AR3
SACH    * +, AR6    //C1=ID/2=1
//堆栈分布情况: ADDRESS/AR6/AR7/ARO/N/M/ID/IW/C2/C1/Y
//ARP:AR6,ARO=ID,AR2:IW,AR3:Y,AR4:sintab,AR5=M,AR6:input
MAR    *0+,AR4
//ARP:AR4,ARO=ID,AR2:IW,AR3:Y,AR4:sintab,AR5=M,AR6=AR6+ID→Rj
LOOP1
LACC    #0
LT      *+,AR6    //TREG=COSα1k
MPY     *+,AR4    //Rj*COSαk,ARP=AR4,AR4:SINαk,AR6:Ij
LT      *+,AR6
MPYA    * -, AR3    //ACC=ACC+Rj*COSαk,PREG=Ij*SINα1k
//ARP=AR3,AR4:SINαk,AR6:Rj
SPAC
//ACC=ACC-Ij*SINα1k
SACH    *+,AR4    //XT=Rj*COSαk-Ij*SINαik
//堆栈分布情况: ADDRESS/AR6/AR7/ARO/N/M/ID/IW/C2/C1/XT/Y
//ARP:AR4,ARO=ID,AR2:IW,AR3:Y,AR4:SINX,AR5=M,AR6:Q.X
LACC    #0

```



```
LT      -. AR6
MPY     *+.AR4      //Rj*SINak,ARP=AR3,AR4:COSak,AR6:Ij
LT      ,AR6
MPYA    * -. AR3    /ACC=ACC+Rj      SINak.PREG=Ij*COSα1
                        //ARP=AR3,AR4:COSak,AR6:Rj

APAC
SACH    -, AR7      //YT=Rj*SINa+Ij*COSαn

堆栈分布情况: ADDRESS/AR6/AR7/ARO/N/M/ID/IW/C2/C1/XT/YT
//ARP:AR7,ARO=ID,AR2:IW,AR3:XT,AR4:COSX,AR5=M,AR6:Q.X
LACC    ,AR6
BCND    D2, NEQ      7 当nom 不为0时, 需要在运算过程中进
                        / 行归一化操作

//
17不进行归一化操作程序部分
2=--
MAR     *0-          AR6=AR6-ID→Ri
LACC    ,AR3
ADD     *,AR6
SACL    *0+, AR3     Ri=Ri+XT
/  ARP:AR3.ARO=ID.AR2:IW.AR3:XT,AR4:COSαn,AR5=M,AR6:Rj
SUB     *+. 1, AR6
SACL    于          //Rj=Ri+XT-XTW2=Ri-XT,AR6:Ij,AR3:YT
1/ARP:AR6.ARO=ID,AR2:IW,AR3:YT,AR4:COSα,AR5=M,AR6:Ij
MAR     *0-          //AR6:Ii
LACC    ,AR3
ADD     -. AR6
SACL    *0+,AR3     //Ii=Ii+YT,AR6:Ij
//ARP:AR3,AR0=ID,AR2:IW,AR3:XT,AR4:COSαs,AR5=M,AR6:Ij
SUB     .1.AR6
SACL    +. 0, AR2    7/Ij=Ii+YT-YT*2=Ii-YT
//STACK:ADDRESS/AR6/AR7/ARO/N/M/ID/IW/C2/C1/XT/YT
//ARP:AR2,ARO=ID.AR2:IW.AR3:XT.AR4:COSα,AR5=M.AR6:NEXT      Rj
B       D          //AR6 指向下一个 Rj

/--
归一化处理程序部分
//-
D2      MAR     *0-          //AR6→Ri
LAC     *.15.AR3
ADD     , 15, AR6
```



```
SACH      *0+, AR3                      //Ri=(Ri+XT)/2
//ARP:AR3,ARO=ID,AR2:IW,AR3:XT,AR4:COSαik,AR5=M,AR6:Rj
SUB      *+,16,AR6
SACH      *  +                          //Rj=Ri-XT,AR6:Ij,AR3:YT
//ARP:AR6,ARO=ID,AR2:IW,AR3:YT,AR4:COSαk,AR5=M,AR6:Ij
MAR      *0-                          //AR5=I
LACC      *,15,AR3
ADD       *,15,AR6
SACH      *0+, AR3                      //Ii=(Ii+YT)/2,AR6:Ij
//ARP:AR3,ARO=ID,AR2:IW,AR3:YT,AR4:COSαk,AR5=M,AR6:Ij
SUB       -,16,AR6
SACH      *+,0,AR2                      //Ij=Ii-YT
//STACK:ADDRESS/AR6/AR7/ARO/N/M/ID/IW/C2/C1/XT/YT
//ARP:AR2, ARO=ID, AR2:IW, AR3:XT, AR4:COS α k, AR5=M, AR6:          下 一 个Rj
D
LAR      ARO,* 一 , AR4                  //ARO=IW
//ARP=AR4,ARO=IW,AR2:ID,AR3:XT,AR4:COSαk,AR5=M,AR6:          下 一 个 Rj
MAR      *0+,AR2                        //AR4=AR4+IW→ 下一个COS α k
LAR      ARO,*                          /ARO=ID
ADRK      #3                            //AR2:C1
//ARP=AR2, ARO=ID, AR2:C1, AR3:XT, AR4:          下 一 个 COS α k, AR5=M, AR6:Rj
LACC      *
SUB       #1
SACL      *-                            //C1=C1-1
//ARP=AR2,ARO=ID,AR2:C2,AR3:XT,AR4:COSαk,AR5=M,AR6:Rj
BCND LOOP4,LEQ                          / / 跳 转 至LOOP4,IF C1<0
MAR      * 一 , AR4                      //AR2:IW
//ARP=AR4, ARO=ID, AR2:IW, AR3:XT, AR4:          下 一 个COS α ik, AR5=M, AR6:Rj
B        LOOP1
LOOP4    LACC*
SUB       #1
SACL      * 一 , AR3
MAR      * 一 , AR2
//ARP=AR2, ARO=ID, AR2:IW, AR3:C1, AR4:          下 一 个 COS α k, AR5=M, AR6:Ri
MAR      一
BCND     LOOP2,GT
MAR      *,AR3
MAR      * 一 , AR5
//ARP=AR5, ARO=ID, AR2:IW, AR3:C2, AR4          下 一 个 COS α k, AR5=M, AR6:Ri
```

```
        BANZ      LOOP3,      — .AR2
/
Y/ 与 C 语言兼容的代码部分
/  —=
        CLRC      OVM
        SPM      0
        MAR      ,AR1
        SBRK      #09
        LAR      ARO,      关      =
        LAR      AR7,      关      —
        LAR      AR6,      关      =
        PSHD
        RET
```



第17章 三相感应电动机恒压 频比控制系统

随着DSP和功率电子器件的发展，变频调速技术已成为许多新兴科技发展的主要动力，特别是带有实时信号处理器的电机控制专用器件和集成度很高的智能功率模块(IPM)的引入，使得由变频器供电的感应电机控制系统获得越来越广泛的应用。由于功率变换器可以实时地调节电机电压和电流的幅值和频率，所以这种控制系统可获得更高的效率、更优越的性能和更小的噪声。恒压频比控制是感应电机最基本的常用控制方式之一，其控制目标就是保持电机电压的幅值和频率比为某个与电机参数有关的固定比值，保证电机在各个工作条件下都近似保持磁通恒定的工作状态，从而保证电机的转矩输出能力。此外，通过控制电机的瞬态电压和电流，可以提高控制系统的动态性能。

本章在介绍三相感应电动机的数学模型的基础上，详细介绍了恒压频比控制技术以及空间矢量PWM调制技术的基本原理，并在智能功率模块PM300CLA060及F2407A构成的电机控制系统中实现三相感应电动机的恒压频比控制，给出相应的软件流程和源程序。源程序在控制系统中经过验证，因篇幅所限，源程序未在书中详细列出，将其放在本书所附光盘中。

17.1 交流感应电机恒压频比控制和空间矢量方法

17.1.1 交流感应电机的恒压频比控制原理

假设三相交流感应电动机的电压是正弦电压，并且忽略通过定子电阻的电压降。在稳态时，可得到以下向量等式：

$$\mathbf{V} \approx j\omega \mathbf{A} \quad (17.1)$$

由式(17.1)可得：

$$\mathbf{V} \approx \omega \mathbf{A} \quad (17.2)$$

其中， $\mathbf{V}$ 和 $\mathbf{A}$ 分别是定子电压向量和定子磁通向量， V 和 A 分别为它们的幅值。可得：

$$A \approx \frac{V}{\omega} = \frac{1}{2\pi f} V \quad (17.3)$$

由式(17.3)可看出, 无论 f 如何变化, 只要保持 V/f 恒定, 则 A 就保持不变, 电机的转矩与逆变器输出频率无关。在实际应用中, 定子电压幅值和频率的比值由这些变量的额定值来确定。但是, 当电机运行在低频率、低电压条件下时, 就不能忽略定子电阻的电压降, 必须采取相应的补偿措施。当频率高于额定频率时, 为了避免电机绝缘破坏的情况发生, 定子电压不能超过额定电压值, 这样就违反了压频比原理, 如图17.1所示。

在恒压频比控制方式下, 由于定子磁通保持恒定, 所以电机的转差率就决定了电机的转矩, 如图17.2所示。因此, 在恒压频比控制方式下, 通过调节转差, 就可以控制交流电机的转矩和转速。

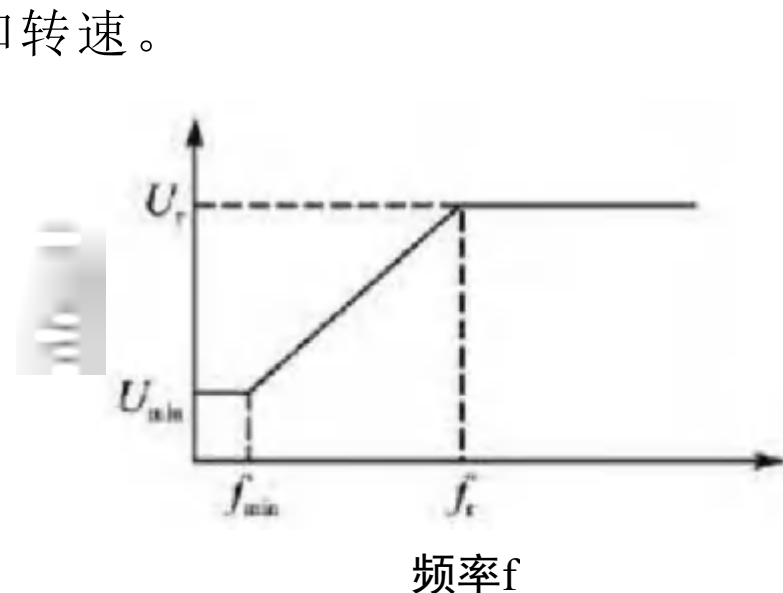


图17.1 恒压频比控制原理

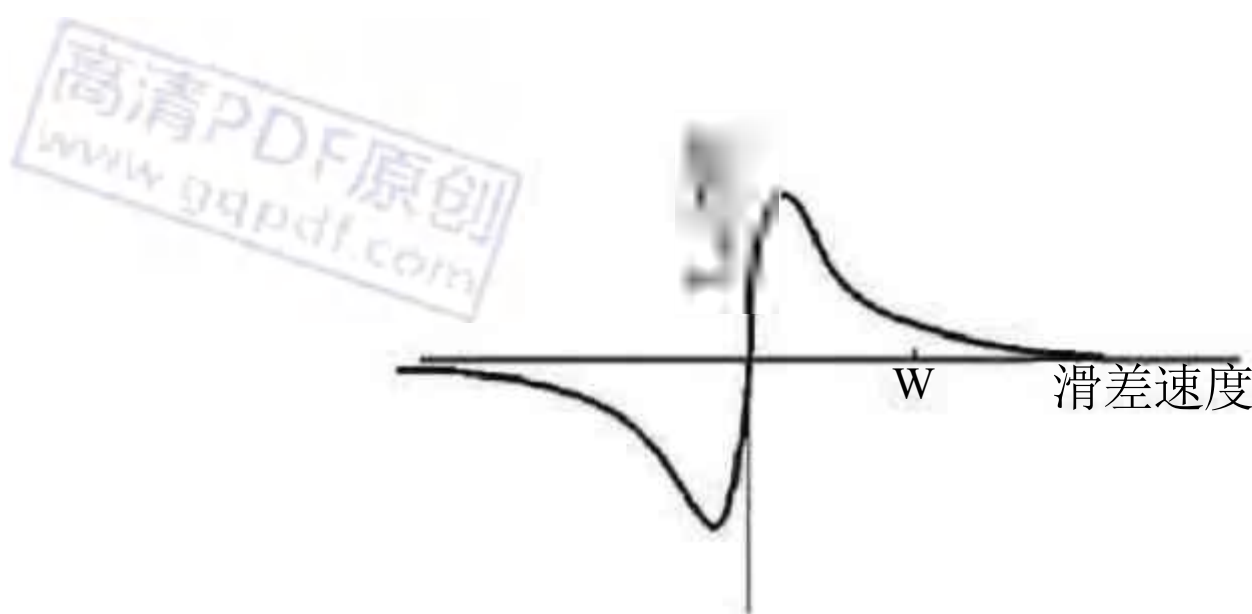


图17.2 在恒定磁通条件下的感应电机转矩和转差曲线

基于恒压频比原理, 交流感应电机既可实现转速开环控制, 也可实现转速闭环控制。转速开环控制主要应用在对转速控制精度要求不高的场合, 例如: 加热、通风、空调、鼓风机和送风机等。在这种情况下, 由转速确定逆变器输出的频率, 同时电机的转速接近其同步转速, 由转差率引起的转速控制误差是可以接受的。

当需要高精度的转速控制时, 可采用带转速闭环的恒压频比控制方案, 通过调节电机的转差, 达到控制转速的目的, 如图17.3所示, 其中, PI 调节器用于调节电机的转差, 以保持电机转速跟踪其设定值。

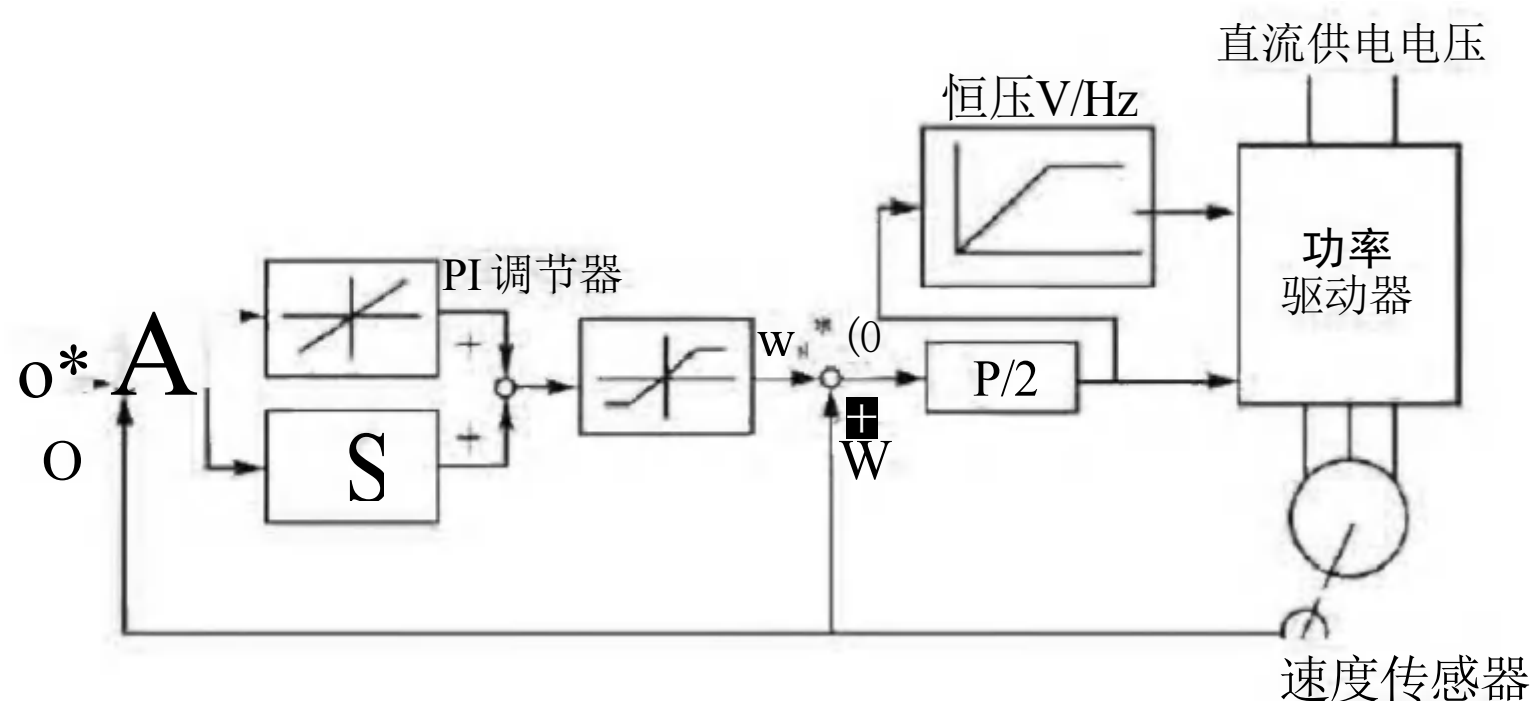


图17.3 基于恒压频比控制方法的三相感应电机闭环 PI 转速控制框图

在恒压频比控制方法中, 有以下两种方式确定电压和频率之间的比例。

(1) 限制变频器输出的最低频率。这种控制方案适用于风扇及送风机等工业应用。因为它们对控制系统的低速响应性能要求不高。当电机工作在额定频率时，电机端的电压为额定电压。所以，在确定电压与频率之间的控制方案时，只需要电机的额定、最小和最大工作频率信息即可。

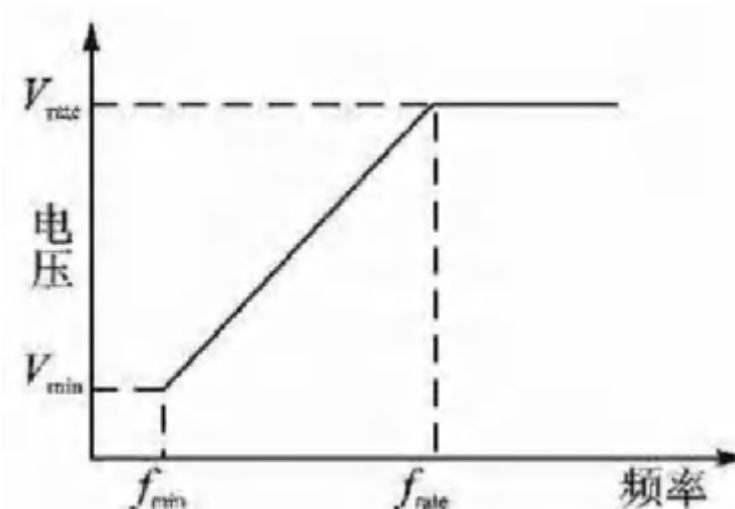


图17.4 恒压频比控制的电压频率曲线

(2) 根据图17.4所示的4个参数(V_{rate} 、 V_{min} 、 f_{rate} 和 f_{min})来确定电压与频率之间的关系。当给定频率低于最小频率时，变频器输出电压保持为 V_{min} ，用于补偿电机定子的电阻压降。

17.1.2 空间矢量PWM 技术

空间矢量脉宽调制SVPWM(Space Vector Pulse Width Modulation)为交流感应电机的一种控制方式，控制三相电压型逆变器的功率器件的开关触发顺序和脉宽。这种开关触发顺序和脉宽的组合将在定子线圈中产生3个相差 120° 电角度且波形失真较小的正弦波电流。通常在恒压频比控制中，采用正弦脉宽调制(SPWM)策略控制功率开关器件的通断，着眼于使逆变器输出电压尽量接近正弦波，其缺点是电压利用率低。磁通正弦PWM（即空间电压矢量SVPWM）技术从电机角度出发，着眼于如何使电机获得幅值恒定的圆形磁场。SVPWM控制用逆变器不同的开关模式产生的实际磁通去逼近基准磁通圆，不但能达到较高的控制性能，而且由于它把逆变器和电机看作一个整体处理，使所得模型简单，便于数字化实现，并具有转矩脉动小、噪声低、电压利用率高等优点。

PWM信号可分成对称PWM和不对称PWM，如图17.5所示。对称PWM信号的脉冲总是以每个PWM周期中点为中心而对称分布。反之，则成为不对称PWM信号。本小节将详细介绍两种PWM信号的应用。

与不对称PWM信号相比，对称PWM信号的输出电压和电流中包含的谐波较少。目前，有许多PWM调制技术适用于感应电机控制，比如正弦PWM、滞环PWM和空间矢量PWM。以下将介绍空间矢量PWM技术。

一种典型的三相电压型逆变器的结构如图17.6所示。

如图17.6所示， V_a 、 V_b 和 V_c 是逆变器的电压输出， $Q_1 \sim Q_6$ 是6个功率管，它们分别被 a 、 a' 、 b 、 b' 、 c 和 c' 这6个控制信号所控制。当逆变桥上半部分的一个功率管开通时（即 a 、 b 或 c 为1时），其下半部分相对的功率管被关闭（即 a' 、 b' 或 c' 为0）。 Q_1 、 Q_3 和 Q_5 这3个功率管的开关状态，即 a 、 b 和 c 为0或为1的状态，将决定 V_a 、 V_b 和 V_c 三相输出电压的波形情况。

逆变桥输出的线电压矢量 $[\underline{V}_L, \underline{V}_L, \underline{V}_L]$ 、相电压矢量 $[\underline{V}_a, \underline{V}_b, \underline{V}_c]$ 和开关变量矢量 $[a, b, c]$ 之间的关系可以用下面两个等式表示：

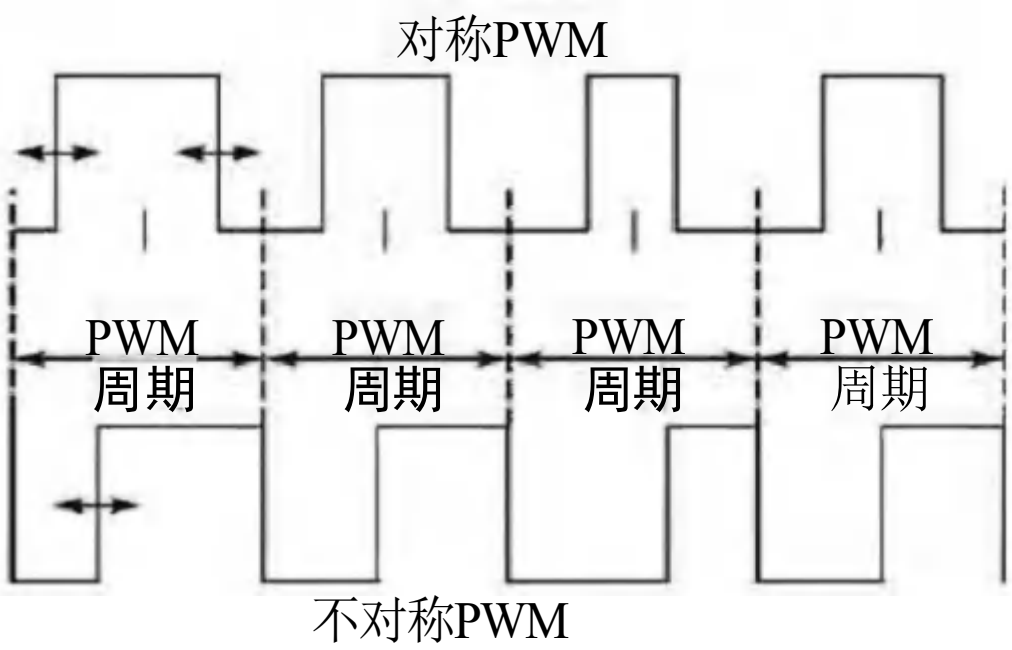


图17.5 PWM 控制脉冲图

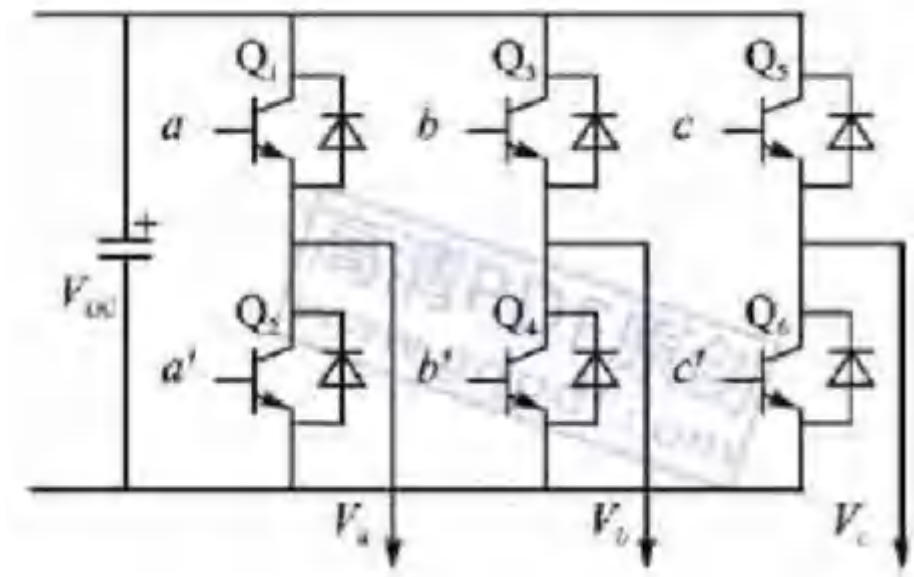


图17.6 三相电压型逆变器结构图

$$\begin{bmatrix} V_a \\ V_b \\ V_c \end{bmatrix} = V_{DC} \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} \quad (17.4)$$

$$\begin{bmatrix} V_a \\ V_b \\ V_c \end{bmatrix} = \frac{1}{3} V_{DC} \begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} \quad (17.5)$$

式中， V_{DC} 是电压型逆变器的直流供电电压，或者称为总线电压。

不难看出，因为开关变量矢量 $[a,b,c]$ 有8个不同的组合值(a 、 b 和 c 只能取0和1)，即逆变桥上半部分3个功率管的开关状态有8种不同的组合，故其输出的相电压和线电压也有8种对应的组合。开关变量矢量 $[a, b, c]$ 与输出的线电压和相电压的对应关系如表17.1所列。

表17.1 功率管的开关状态和与之对应的输出线电压和相电压的关系表

	方	4	V	V	V	V	V	V
0	0	0	0	0	0	0	0	0
0	0	1	$2V_{pe}/3$	$-V_{re}/3$	$-V_{re}/3$	V_{Tx}	0	$-V_{rx}$
0		0	$=V_{rc}/3$	$2V_{pc}/3$	$二V_{Dc}/3$	$-V_{DC}$	V_{re}	0
0			$V_p/3$	$V_{p:}/3$	$-2V_x/3$	0	V_D	$-V_{pe}$
	0.	0.	$-V_{pc}/3$	$-V_{pc}/3$	$2V_x/3$	0	$-V_{pe}$	V_X
	0		$V_{be}/3$	$-2V_x/3$	$V_{pe}/3$	V_p	$-V_{pe}$	0
		0.	$-2V_{px}/3$	$V_{pc}/3$	$-V_{pc}/3$	$-V_{pe}$	0	$V_{px:}$
			0	0		0		0

在该表中 V 、 V_b 和 V_c 表示3个输出的相电压， V 、 V 和 V 表示3个输出的线电压。

在(α,β)坐标系中与输出的三相线电压相对应的分量可以由式(17.6)和(17.7)表示：

$$V=V_a$$

(17.6)

$$V_p=(2V_+V_a)/\sqrt{3}$$

上面的等式也可以用矩阵形式表示如下：

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

(17.7)

由于逆变桥中，功率管开关状态的组合一共只有8个，则对应于开关变量矢量[a,b,c]在(a,β)坐标系中的V、Vg也只有有限种组合。V、Vβ是空间矢量分解得到的子轴分量，它们的对应关系如表17.2所列。

表17.2 开关变量矢量与其对应的空间矢量(α,β)子轴分量的关系表

a	b	a	V	V	矢 量	c	b	a	V	V	矢 量
0	0	0	0	0	0 ₀	1	0	0			U ₂₄₀
0	0	1		0	U ₀	1	0	1			U ₃₀₀
(	1	(			U ₁₂₀	1	1	0		0	U ₁₈₀
0	1	1			U ₆₀	1	1	1	0	0	u

表17.2中V_α、V_β被称为基本空间矢量的(α,β)轴分量，每个基本空间矢量与恰当的功率管的开关命令信号组合(c,b,a)相对应。例如，当(c,b,a)=001时，表示此时的空间矢量为U₀。被功率管的开关组合所决定的8个基本的空间矢量如图17.7所示。

空间矢量PWM技术的目的是，通过与基本的空间矢量对应的开关状态的组合，得到一个给定的定子参考电压矢量U_{out}。参考电压矢量U_{out}用其(α,β)轴分量U_α和U_β表示。图17.8表示参考电压矢量U_{out}与之对应的(α,β)轴分量U_α和U_β以及基本空间矢量U₀和U₆₀的对应关系。

在图17.8中，ΣV_α表示U_α和U₅₀的α轴分量之和，ΣV_β表示U_β和U₆₀的β轴分量之

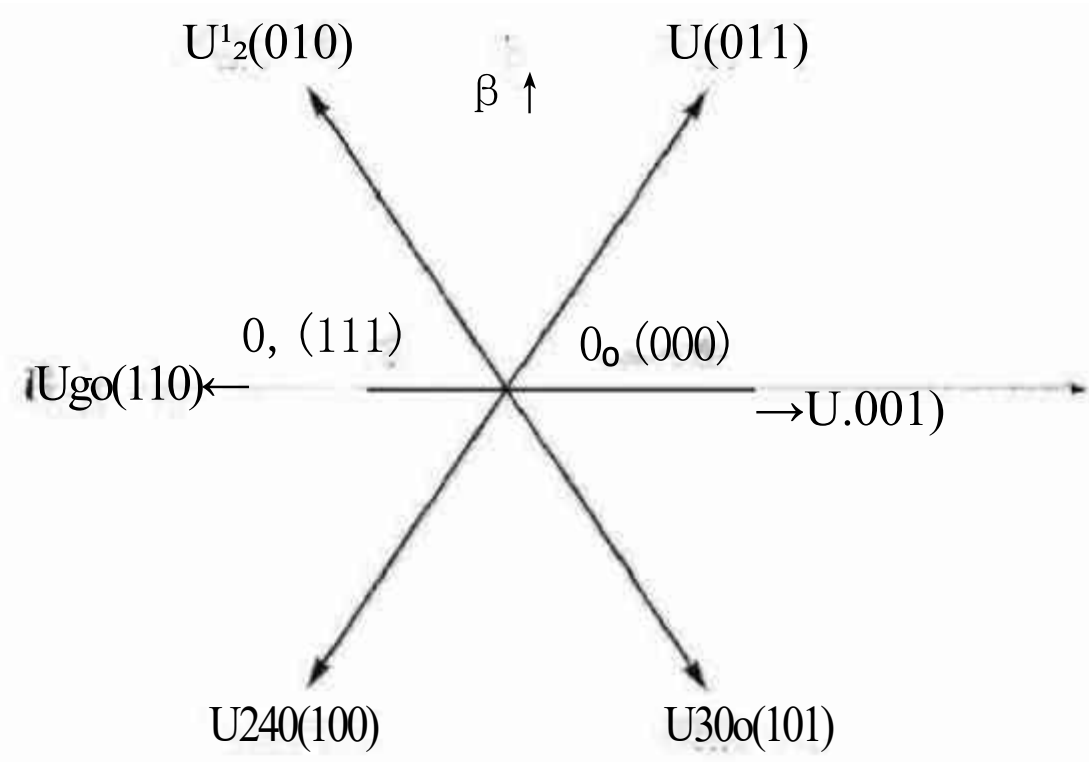


图17.7 基本的空间矢量与对应的(c, b, a)示意图

和。结合表 17.2 可知基本空间矢量的幅值都为 $2V_x/3$, 故有:

$$\begin{cases} \sum V_{\beta} = 0 + \frac{V_{\Gamma C}}{\sqrt{3}} = \frac{V_{\Gamma C}}{\sqrt{3}} \\ \sum V_{\alpha} = \frac{2V_{\Gamma C}}{3} + \frac{V_{\Gamma C}}{3} = V_{\Gamma C} \end{cases} \quad (17.8)$$

在图 17.8 所示情况中，参考电压矢量 U_o 位于被基本空间矢量 U_u 、 U_v 所包围的扇区中，因此 U_o 可以用 U_u 和 U_v 两个矢量来表示。于是，可以有：

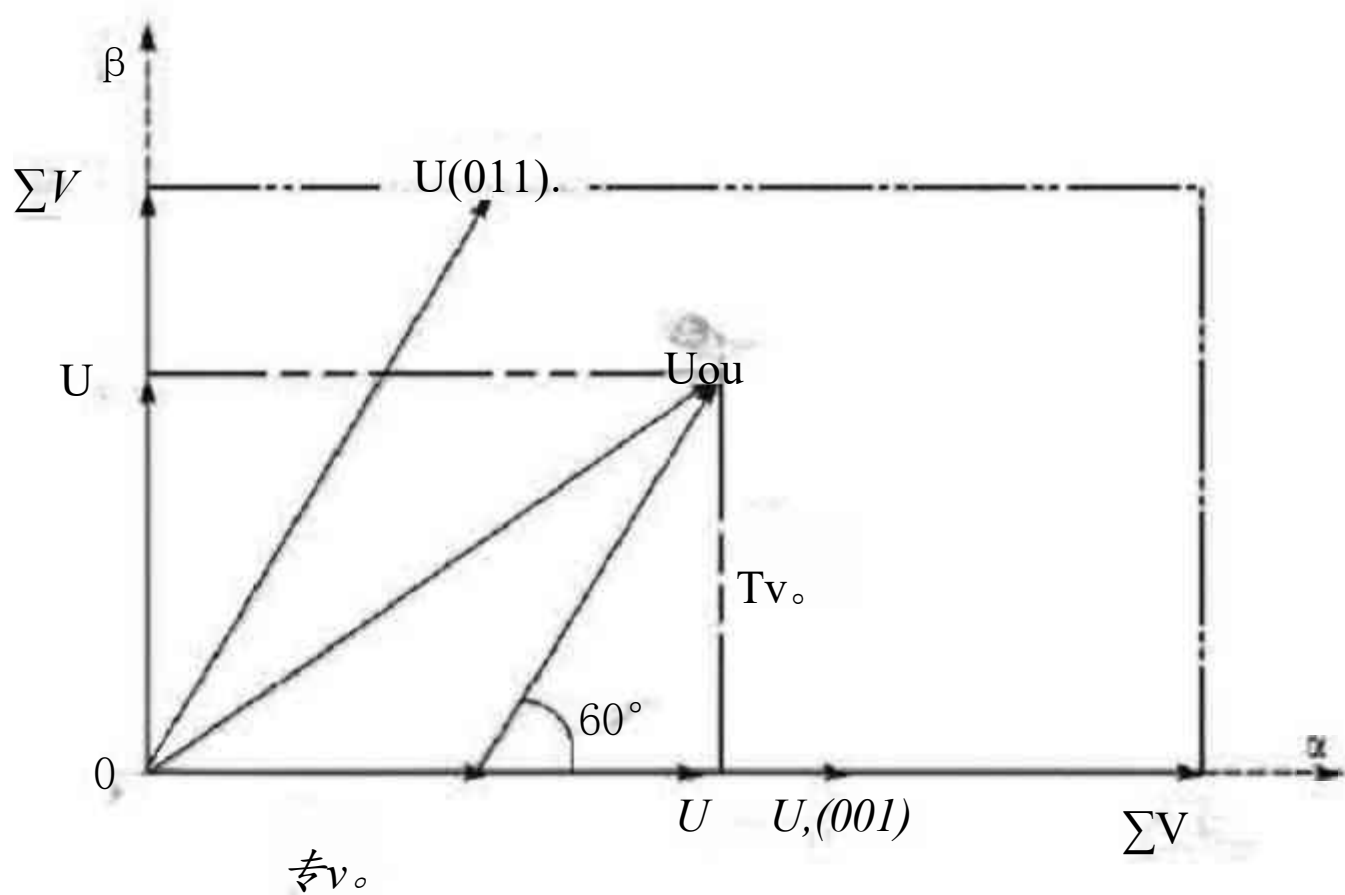


图17.8 U_{au} 和 U_a 、 U , 以及 U_a 、 U 。的对应关系图

$$\begin{cases} T = T_1 + T_2 + T_3 \\ \mathbf{U}_{\text{out}} = \frac{T_1}{T} \mathbf{U}_1 + \frac{T_2}{T} \mathbf{U}_2 + \frac{T_3}{T} \mathbf{U}_3 \end{cases} \quad (17.9)$$

式中， T_1 和 T_2 分别是在周期时间 T 中基本空间矢量 U_1 、 U_2 各自的作用时间； T_0 是 0 矢量的作用时间。 T_1 和 T_2 可以计算如下：

$$\left. \begin{aligned} U_{\beta} &= \frac{T_1}{T} |U_{90}| \sin 60^\circ \\ U_{\alpha} &= \frac{T_1}{T} |U_0| + \frac{T_2}{T} |U_{90}| \cos 60^\circ \end{aligned} \right\} \quad (17.10)$$

从前面的表述不难看出，所有基本空间矢量的幅值都为 $2V_{pc}/3$ 。如果它们取相对于最大相电压 $V_{be}/\sqrt{3}$ （最大线电压为 V_c ，则最大相电压为 $V_{pe}/\sqrt{3}$ ）的标么值，则空间矢量的幅值变成 $2/\sqrt{3}$ ，即经过归一化后空间矢量的幅值为 $|U_0|=|U_6|=2/\sqrt{3}$ 。代入式（17.10）得：

$$\left. \begin{aligned} T_1 &= \frac{T}{2}(\sqrt{3}U_\alpha - U_\beta) \\ T_2 &= TU_\beta \end{aligned} \right\} \quad (17.11)$$

在式(17.10)和式(17.11)中, U_{α} 和 U_{β} 表示矢量 U_a 相对于最大相电压 $V_{pc}/\sqrt{3}$ 归一化后(即取标么值)后的 (α, β) 轴分量; $T_0 = T - T_1 - T_2$ 是 0 矢量的作用时间。取 T_1 、 T_2 与周期 T 的相对值有如下关系:

$$\left. \begin{aligned} t_1 &= \frac{T_1}{T} = \frac{1}{2}(\sqrt{3}U_a - U_\beta) \\ t_2 &= \frac{T_3}{T} = U_\beta \end{aligned} \right\} \quad (17.12)$$

同理，如果矢量 U_{120} 位于被基本空间矢量 U_{60} 、 U_{120} 所包围的扇区中， $|U_{60}|=|U_{120}|=2/\sqrt{3}$ ，矢量作用时间的相对值可表示为：

$$\left. \begin{aligned} t_1 &= \frac{T_2}{T} = \frac{1}{2}(-\sqrt{3}U_a + U_\beta) \\ t_2 &= \frac{T_3}{T} = \frac{1}{2}(\sqrt{3}U_a + U_\beta) \end{aligned} \right\} \quad (17.13)$$

在上面的等式中， T_2 是空间矢量 U_{20} 在周期 T 中的作用时间。如果定义 X 、 Y 和 Z 这 3 个变量如下：

$$\left. \begin{aligned} X &= U_\beta \\ Y &= \frac{1}{2}(\sqrt{3}U_a + U_\beta) \\ Z &= \frac{1}{2}(-\sqrt{3}U_a + U_\beta) \end{aligned} \right\} \quad (17.14)$$

综上所述，矢量 U_{out} 位于被基本空间矢量 U_{60} 、 U_{60} 所包围的扇区（即扇区 0），则可得 $t_1 = -Z, t_2 = X$ ；矢量 U_{ou} 位于被基本空间矢量 U_{60} 、 U_{120} 所包围的扇区（即扇区 1），则 $t_1 = Z, t_2 = Y$ 。同理，当 U_{ou} 位于被其他空间矢量所包含的扇区中时，相应的 t_1 和 t_2 也可以用 X 、 Y 或 Z 表示，其对应关系如表 17.3 所列。

表 17.3 t_1 、 t_2 与 X 、 Y 和 Z 的对应关系表

扇 区	U_{60} 、 U_{60}	U_{60} 、 U_{120}	U_{120} 、 U_{180}	U_{180} 、 U_{240}	U_{240} 、 U_{300}	U_{300} 、 U_{360}
t_1	$-Z$	Z	X	$-X$	$-Y$	Y
t_2	X	Y	$-Y$	Z	$-Z$	$-X$

已知一个矢量 U_{ou} ，如果要利用表 17.3 计算 t_1 和 t_2 ，则必须知道 U_{60} 所在的扇区。下面着重讨论如何用矢量 U_{out} 计算它所在的扇区。

一般而言，可以用矢量 U_{ou} 的 (a, β) 轴分量 U_a 和 U_β 来表示矢量本身，则可以把 3 个参考量 V_{ref1} 、 V_{ref2} 和 V_{ref3} 用 U_a 和 U_β 来表示，其关系式如下：

$$\left. \begin{aligned} V_{ref1} &= U_\beta \\ V_{ref2} &= \frac{-U_\beta + U_a \times \sqrt{3}}{2} \\ V_{ref3} &= \frac{-U_\beta + U_a \times \sqrt{3}}{2} \end{aligned} \right\} \quad (17.15)$$

定义3个变量 a、b和c。如果 $V_{\alpha} > 0$ ，则 $a=1$ ，否则 $a=0$ ；如果 $V_{\beta} > 0$ ，则 $b=1$ ，否则 $b=0$ ；如果 $V_{\gamma} > 0$ ，则 $c=1$ ，否则 $c=0$ 。设 $N=4 \times c + 2 \times b + a$ ，则 N 与扇区数(sector) 的对应关系如表17.4所列。

第0扇区即为基本空间矢量 U_0 、 U_1 包围的扇区，第1扇区即为基本空间矢量 U_1 、 U_2 包围的扇区……第5扇区即为基本空间矢量 U_5 、 U_0 包围的扇区。

表17.4 N 与扇区数sector 的对应关系

N	1	2	3	4	5	6
扇 区	1	5	0	3	2	4

到此为止，如果已知参考电压矢量 U_a ，或其在 (α, β) 坐标系中的 $(\alpha \cdot \beta)$ 轴分量 U_α 和 U_β ，就可以根据上面的推导计算出与 U_{om} 对应的两个基本空间矢量的作用时间相对SVPWM 调制周期 T 的比例 t_1 、 t_2 。如果已知 t_1 、 t_2 ，又已知要求的SVPWM 的调制周期 T，则就可以确定空间矢量分别的作用时间 T_1 、 T_2 ，再加上前面其他一些推导，即可很方便地利用 F2407 实现 SVPWM 算法。

17.1.3 利用F2407 实现 SVPWM 算法

1. 产生空间矢量PWM 波形的寄存器设置

每个 F2407 的事件管理器 EV 模块都具有操作十分简单的对称空间矢量PWM 波形产生的内置硬件电路。为了输出空间矢量PWM 波形，用户只需要设置以下寄存器：

- (1) 设置 ACTRx 寄存器来定义比较输出引脚的输出方式，高电平有效，或者低电平有效。
 - (2) 设置COMCONx 寄存器来使能比较操作和空间矢量PWM 模式，并且把CMPRx 的重装入条件设置为下溢出。
 - (3) 将通用定时器1或2、4或5设置成连续增/减计数模式，并启动定时器。
- 然后，用户须确定在二维 (α, β) 坐标系下输入到电机的电压 U_{ou} ，并分解 U_o 为 U_α 和 U_β ，以确定每个 PWM 周期的以下参数：
- (4) 两个相邻矢量， U_i 和 U_{i+1} （通过确定 U_o 所在的扇区数就可以达到此要求）。
 - (5) 根据SVPWM 的调制周期 T，求出两个基本的空间矢量和0矢量分别的作用时间 T_1 、 T_2 和 T_0 。
 - (6) 将相应于 U_i 的开启方式写入ACTRx.14~12 位中，并将1写入 ACTRx.15 中，或者将 U_{i+1} 的开启方式写入ACTRx.14~12 中，并将0写入ACTRx.15 中。
 - (7) 将 T_1 、 T_2 的值写入 CMPR1 或 CMPR4 寄存器，将 $(T_1+T_2)/2$ 的值写入 CMPR2 或 CMPR5 寄存器。

2. 空间矢量PWM 的硬件工作原理

为完成一个空间矢量 PWM 周期，每个事件管理 EV 模块 的空间矢量 PWM 硬件工作

如下：

(1) 在每个周期的开始，将 PWM 输出置成由 ACTRx.14~12 设置的新方式U₁，此称为第一类输出方式。

(2) 在增计数期间，当CMPR1 或 CMPR4 与通用定时器1或3发生第一次匹配时，如果 ACTRx.15 为 0，则将 PWM 输出开启到方式U_{y+60}；如果 ACTRx.15 为 1，则将 PWM 输出开启到方式U_{y-60}(U_{0-so}=U₃₀₀, U₃₆₀₊₆₀=U₆₀)，此称为第二类输出方式。

(3) 在增计数期间，当CMPR2 或 CMPR5 与通用定时器1或3发生第一次匹配时，即计数器达到(T₁+T₂)/2 时，将PWM 输出开启至方式000或111。它们与第二类输出方式之间只有1位的差别。

(4) 在减计数期间，当CMPR2 或 CMPR5 与通用定时器1或3发生第二次匹配时，将 PWM 输出置回到第二类输出方式。

(5) 在减计数期间，当CMPR1 或 CMPR4 与通用定时器1或3发生第二次匹配时，将 PWM 输出置回到第一类输出方式。

3. 未用到的比较寄存器

对 EVA 来说，在产生空间矢量PWM 输出中只用到比较寄存器 CMPR1 和 CMPR2。然而，CMPR3 寄存器也一直在与通用定时器1进行比较。

当发生一次比较匹配时，若相应的比较中断没有被屏蔽，则相应的比较标志位将置位并发出中断请求信号。因此，没有用于空间矢量PWM 输出的CMPR3 寄存器，仍可用于其他定时事件的发生。

同理，对于EVB 来说，没有用于空间矢量PWM 输出的CMPR6 寄存器，仍可用于其他定时事件的发生。

同样地，由于状态机引入了附加延时，在空间矢量PWM 模式中比较输出跳变被延时1个 CPU 时钟周期。

4. 空间矢量PWM 的边界条件

在空间矢量PWM 模式中，当两个比较寄存器 CMPR1 和 CMPR2 装入的值都是0时，3 个比较输出全都变成无效。因此，在使用空间矢量 PWM 时，应满足如下关系式：

$$\text{CMPR1} \leq \text{CMPR2} \leq \text{T1PR} \quad \text{或} \quad \text{CMPR4} \leq \text{CMPR5} \leq \text{T3PR}$$

否则将导致不可预测的情况发生。

5. 空间矢量PWM 波形

生成的空间矢量PWM 波形是关于每个 PWM 周期中心对称的，因此被称为对称空间矢量 PWM 生成法。

图17.9给出了空间矢量PWM 波形的示例。

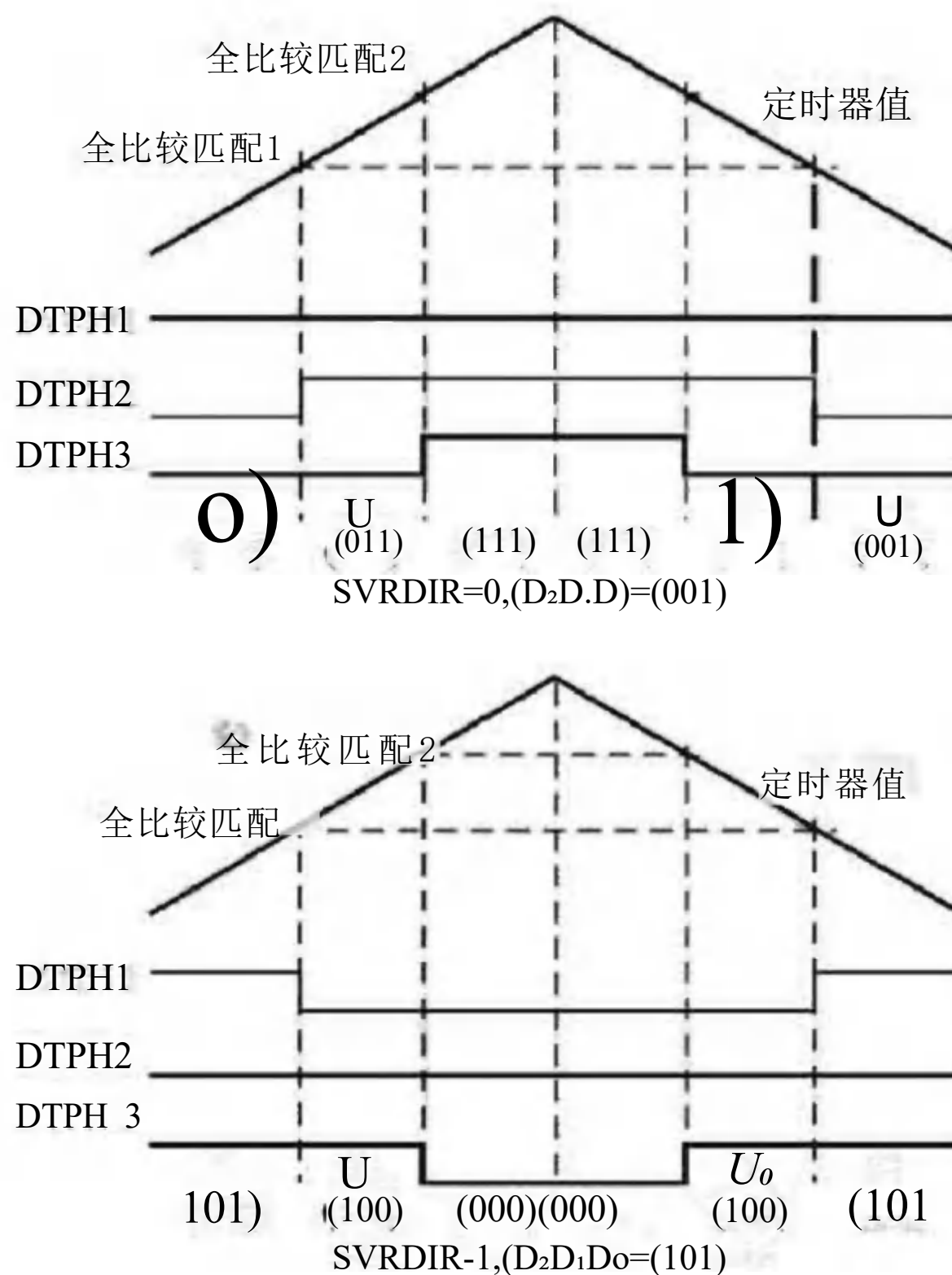


图17.9 对称空间矢量 PWM 波形

17.2 三相感应电动机转速闭环控制系统

本节将详细介绍转速闭环感应电机恒压频比控制方案。转速闭环是由经典的比例积分控制器实现的。以下将这个方案分软件流程、空间矢量 PWM、电机转速测量和闭环控制4个部分分别加以说明。

17.2.1 软件流程

本方案的整个闭环控制算法将在一个中断服务程序(ISR) 中完成, 如图17.10所示。主函数(后台程序)主要负责F2407 外围设备初始化(例如: 锁相环 PLL、程序监视器(看门狗)、中断控制器及事件管理器)和采用转速开环方式启动电机。其余功能则由中断服务程序 PWM_ISR 完成。这个中断服务程序ISR 由事件管理器的定时器1的周期中断标志调用。中断周期为25 μ s, 中断服务程序的执行时间必须小于25 μ s, 留出的时间让主程序工作。

图17.11为PWM_ISR 的软件流程图。除了3个主要的条件程序块之外, 大多数代码都

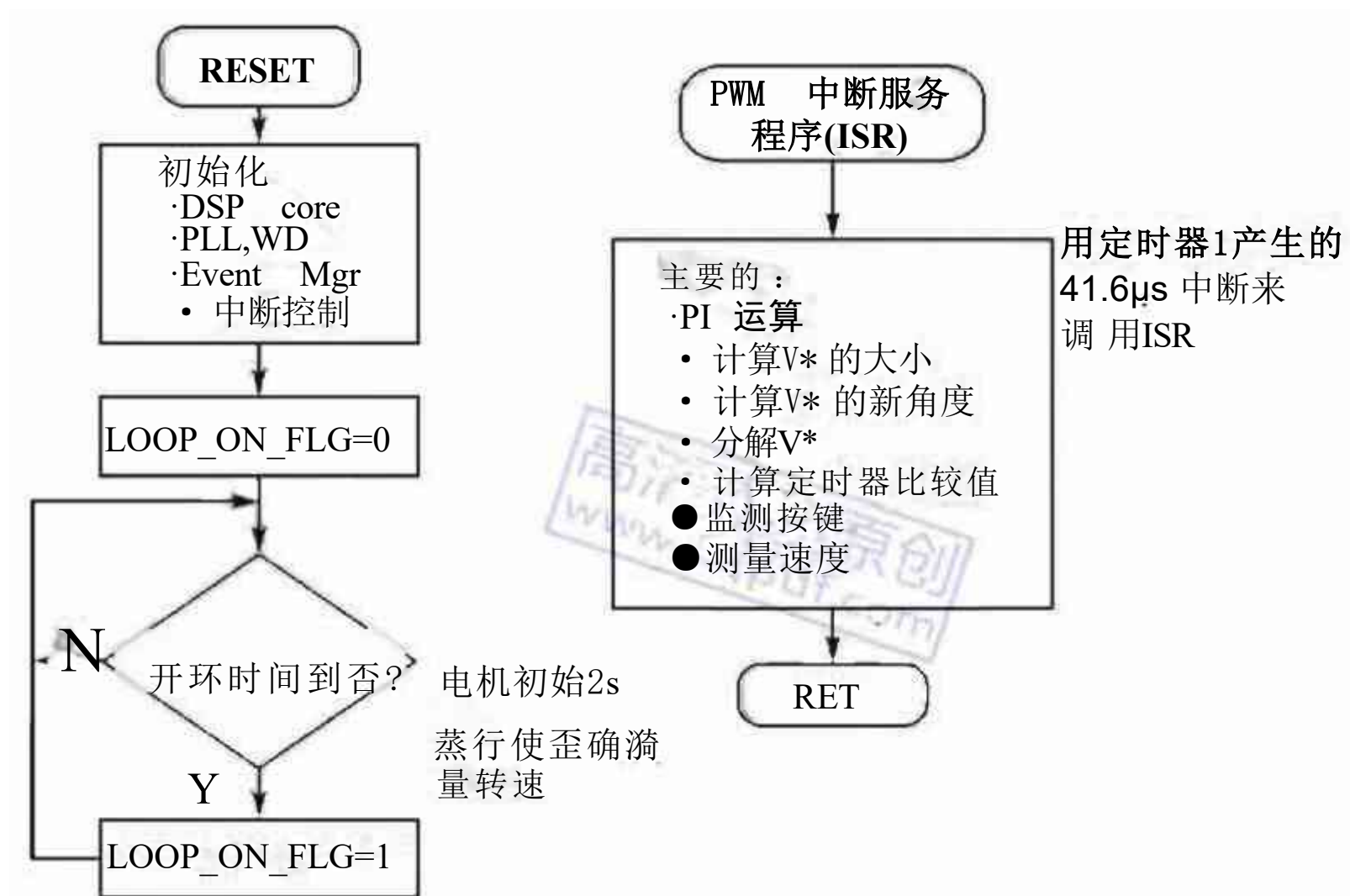


图17.10 软件流程图

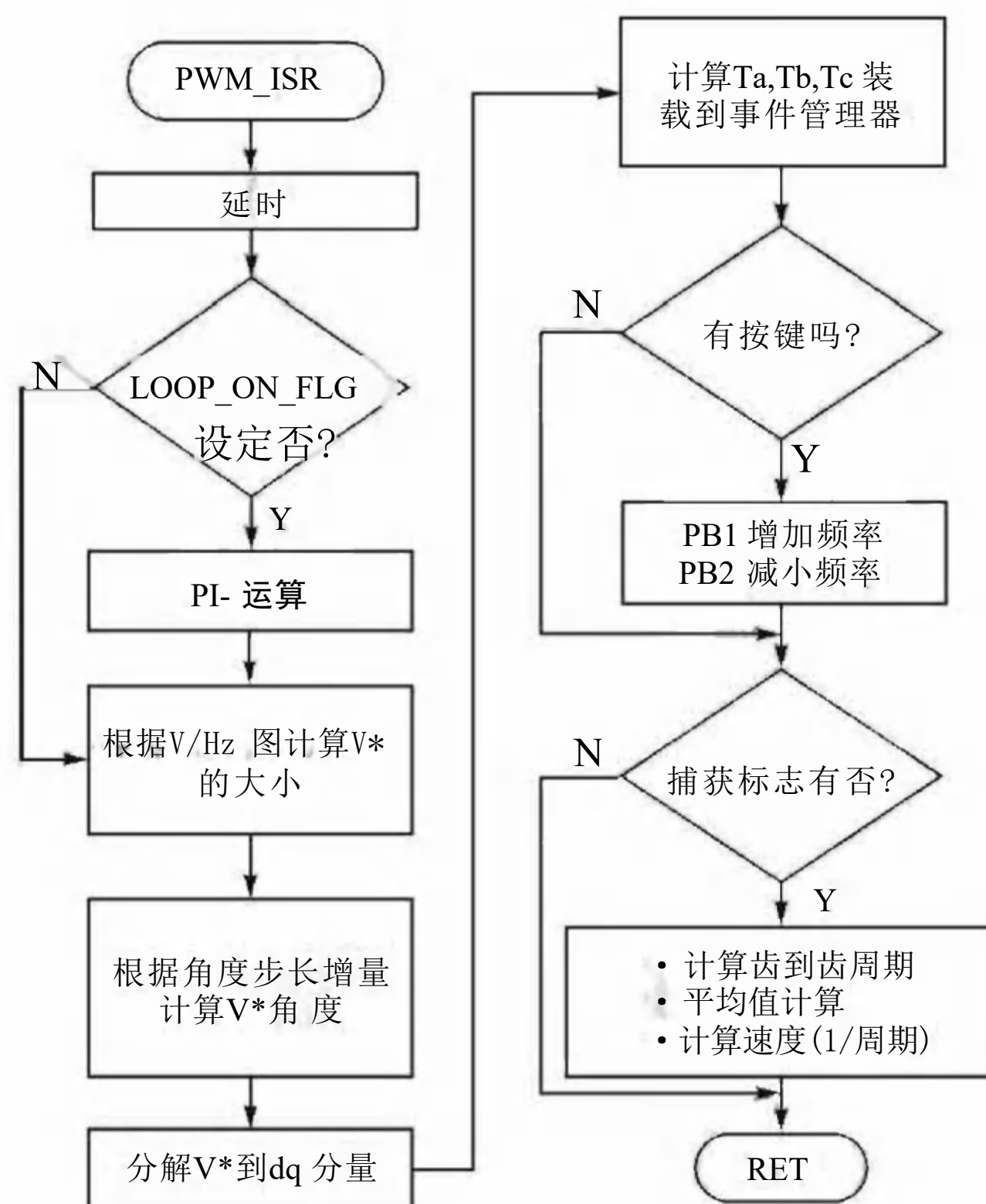


图17.11 PWM_ISR 的软件流程图

是嵌入代码。

一旦启动超时，PI 算法就会被旁路掉，直到LOOP_ON_FIG 标志设置为止。

＞每次中断服务程序 ISR 循环中都会访问控制电机转向设置的 PB1 和 PB2 引脚信号。

但是除非PB1 和 PB2 置低，否则不会执行任何操作(本书光盘所附程序有所改动，以光盘所附程序为准)。

＞仅 当Capture1 标志设置时才计算转速。该标志代表DSP 捕捉到霍尔传感器输出的脉冲上升沿(本书光盘所附程序有所改动，以光盘所附程序为准)。

由于中断服务程序ISR 的调用周期是25 μ s, 这意味着图17. 11所示所有算法的调用频率为40 kHz。 尽管相对于空间矢量PWM 的频率而言，该频率是相匹配的，但是对于PI 转速控制环和电机转速测量和信号去抖动，这个频率太高了。因为F2407 DSP 芯片在40 MIPS 时有足够的带宽，使得它能轻易地在25 μ s 内完成整个 ISR。 此外，由于本方案没采用多中断结构，从而避免了由中断优先级和同步带来的问题，所以程序结构变得很清晰。同时，40 kHz 的高采样频率可以提高系统的性能。

17.2.2 空间矢量PWM 实现

当图17. 6所示的逆变器按照一定的规律输出三相对称正弦电压时，交流感应电机的定子侧就可以建立一个旋转磁动势。产生旋转磁动势需要完成以下步骤：

① 生成参考电压矢量：对图17. 12所示的参考电压矢量进行精确的定位，既要控制其相位、转速，还要控制其幅值。

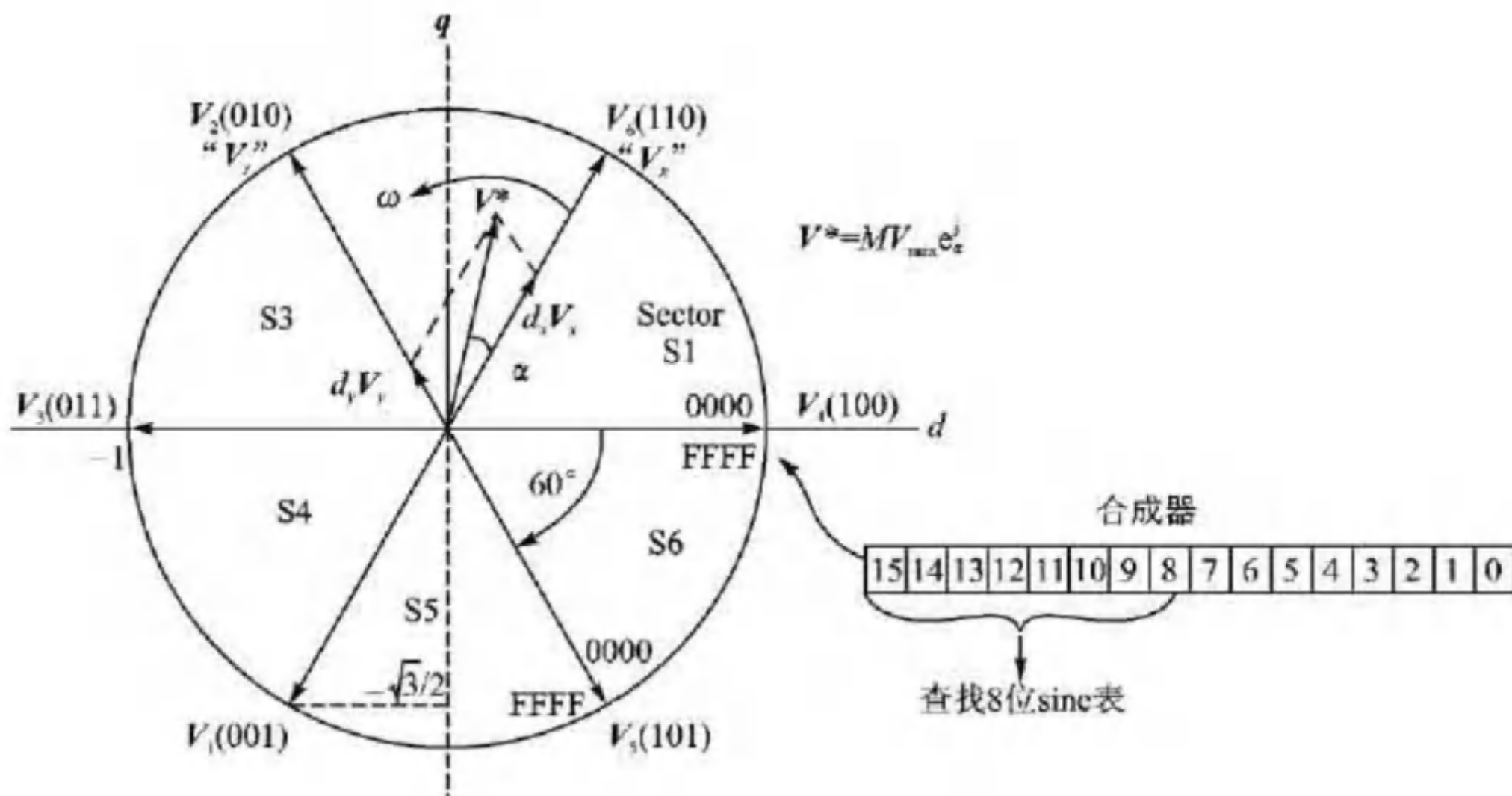


图17. 12 参考电压矢量

② 分解参考电压矢量：将参考电压矢量 V^* 分解为一系列适当的开关变量 a 、 b 、 c 。

③ 通过PWM模块输出相应的开关模式：将这些开关变量转换成相应的比较寄存器值，再通过F2407的PWM模块输出。

要得到参考电压矢量 V^* 在空间和时间上的准确位置，就必须准确控制这个矢量的模 M 、转速和相角 α 。其中，参考矢量的旋转角频率代表感应电机的电气角频率，参考矢量的模 M 则用于控制变频器输出线电压的峰值。

在本应用方案中，参考矢量的角速度 ω 由一种精确频率控制算法生成。这种控制算法巧妙地利用了有限字长寄存器的溢出特性。通常把这个寄存器称为积分器。这个积分器的高8位(F2407的数据存储器地址)用作一个256字正弦表的查表指针。在每个控制周期内，给该寄存器加上一个固定的步长，通过查表就可得到一个恒定角频率的正弦波。实际上，这个累加的过程就是对角速度积分以产生角位移的过程。这样，通过改变步长的大小，可很容易地实现角频率的控制。

为了将参考电压矢量分解成它所在扇区内的两个基本矢量，必须获得参考矢量的角位移 α ，如图17.12所示。因为在6个扇区内电压矢量的分解是完全一样的，所以本方案仅仅需要一个 60° 的正弦表。对于一个已知的积分步长STEP，参考电压矢量的旋转角频率可如下表示：

$$\omega = \frac{\text{STEP} \times f_s}{6 \times 2^m} \quad (17.16)$$

其中， f_s 是采样频率（例如，PWM频率）；STEP是积分步长； m 是积分器的位数。

图17.12给出了6个基本矢量(V_1 、 V_2 、 V_3 、 V_4 、 V_5 和 V_6)，它们分别代表逆变器开关模式与输出相电压之间的关系。在任意一个时刻，逆变器只能输出这些矢量中的一个，或者零矢量 V_0 和 V_7 中的一个。零矢量 V_0 和 V_7 分别对应于开关状态0(000)和状态7(111)。对于任意一个参考电压矢量，它都可以由它所在扇区的两个基本矢量 V_i 和 V_j 和任意一个零矢量 V_0 合成，即：

$$V^* = d_1 V_i + d_2 V_j + d_0 V_0 \quad (17.17)$$

其中， V_0 是零矢量； d_1 、 d_2 和 d_0 分别是开关状态X、Y和Z在一个PWM周期内的占空比。而且，这3个占空比之和必须等于1，即 $d_1 + d_2 + d_0 = 1$ 。

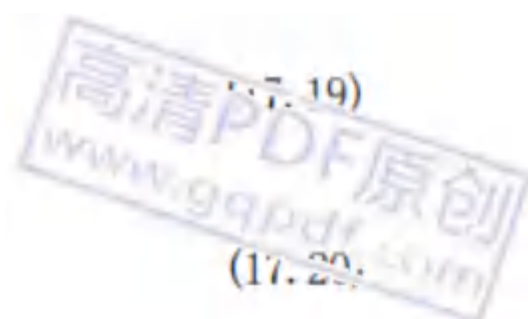
图17.12所示的电压参考矢量 V^* 可以由下式表示：

$$V^* = M V_{\max} e^{j\alpha} = d_1 V_i + d_2 V_j + d_0 V_0 \quad (17.18)$$

式中， M 是调制度。

将 V^* 分解成 d 、 q 分量，公式如下：

$$\begin{aligned} \frac{\sqrt{3}}{2} \times M \cos(\alpha) &= d_1 + \frac{1}{2} d_2 \\ \frac{\sqrt{3}}{2} \times M \sin(\alpha) &= \frac{\sqrt{3}}{2} d_2 \end{aligned}$$



解之可得：

$$d = M \sin(60 - \alpha) \quad (17.21)$$

$$d_s = M \sin(\alpha) \quad (17.22)$$

由于 d - q 坐标平面在空间中的对称性，这使得式(17.21)和式(17.22)适用于任何一个扇区。这也是在本方案中仅需要一个 60° 正弦表的原因。

在实现电压参考矢量的定位和分解程序中，有几点是必须注意的：

计算 d_s 和 d 时均采用 Q15 格式。

> d_s 和 d 的最终结果采用 Q0 格式表示，以获得整数形式的比较寄存器值。

》程序将根据频率给定以及频率—电压函数确定调制度 M 。

> 在本算法中，由于不同扇区之间 d_s 和 d 的计算方法是没有区别的，所以必须在程序内保留一个寄存器，用于指示当前所在的扇区，以获得适当的比较值 (T_s 、 T 和 T_s)。

确定 PWM 占空比 d_s 、 d 和 d 后，就可以据此算出 F2407 事件管理器中的比较寄存器的比较值。本方案将采用不对称 PWM 算法，如图 17.13 所示。在每个 PWM 周期内程序都会重新计算并更新比较寄存器值。

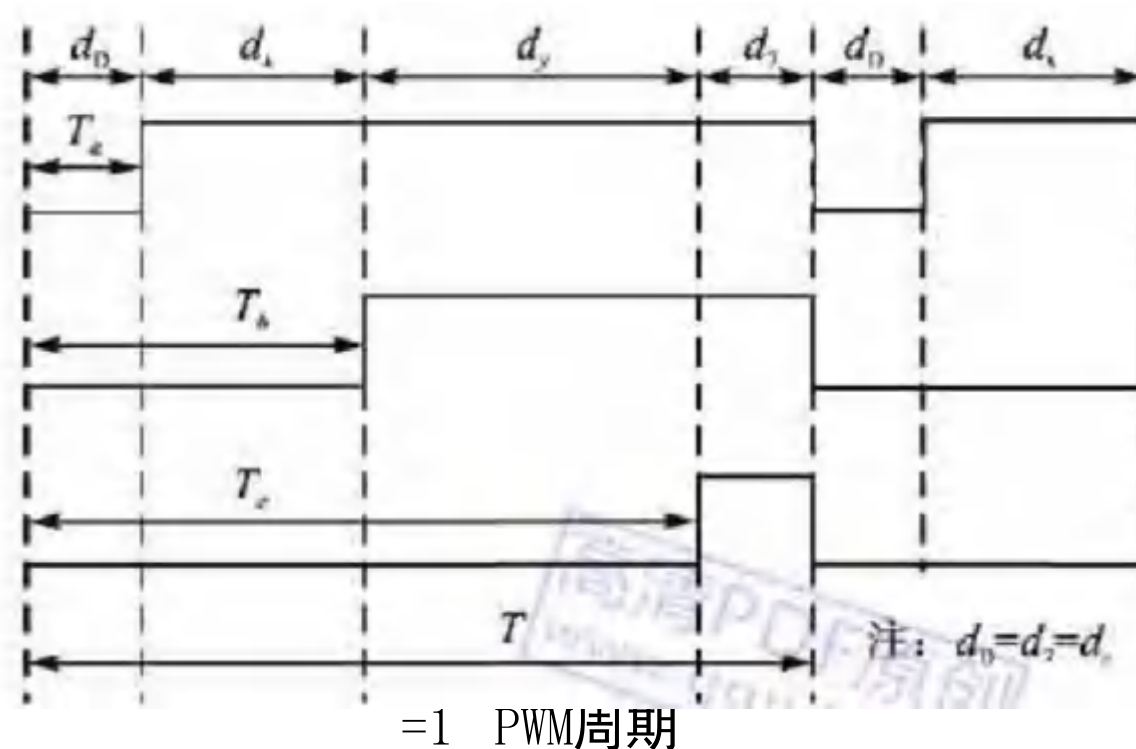


图17.13 转速闭环控制方案的PWM输出开关模式

从图17.13可以看出：

$$T_o = (T - d_s = d) / 2 \quad (17.23)$$

$$T_o = d_s + T \quad (17.24)$$

$$T_s = T - T \quad (17.25)$$

以下程序将根据占空比 d_s 和 d 计算出 PWM 比较值。此外，本程序还包括了相位校正算法。这些程序适用于任意一个扇区，但是，对于任意一个 PWM 周期，输出电压参考矢量只可能在某一个扇区内。

虽然本设计采用的是不对称 PWM，但只需对以上程序进行适当的改动，并把 F2407 的 GP 定时器1设置为连续增/减计数模式，就可以产生对称的 PWM 波形（本书光盘所附程序有

所改动，以光盘所附程序为准)。

一般情况下，对称 PWM 在电机电压和电流中引起的谐波失真比不对称 PWM 要小。但是，在定时器分辨率一定的情况下，对称 PWM 的分辨率却只有不对称 PWM 的一半。在 40 MIPS 下，F2407 输出不对称 PWM 时的分辨率是 25 ns，然而输出对称 PWM 的分辨率却只有 50 ns。

17.2.3 电机转速测量

要实现电机转速的闭环控制，就必须对电机的实际转速进行测量。在本方案中，转速测量由一个低成本的 25 齿的齿轮和一个霍尔传感器完成。

图 17.14 所示是这个齿轮的实际物理结构，及其测量转速的工作原理。每当齿轮的齿靠近霍尔传感器时，传感器都会输出一个方波。所以，电机每转一圈，霍尔传感器就会输出 25 个方波脉冲。F2407 的捕捉单元通过检测每个脉冲的周期就可以算出电机的实际转速。此外，为了减小测量误差，每次测得最新的脉冲周期时，都可以将这个周期与最近测得的 24 个周期值一起取平均值作为当前转速值。

F2407 捕捉单元可以准确测量不同事件之间的时间 (其测量精度由 F2407 的时钟周期和分频器决定)。在本方案中，选择脉冲上升沿作为捕捉的事件，并将定时计数器 1 的预分频值设为 32，将运行模式设置为连续增计数模式，设置周期寄存器为 0x7FFF，同时使能定时计数器。捕捉的任务就是确定两个脉冲上升沿之间的时间间隔，如图 17.15 所示。

在图 17.15 中，纵轴 $f(t)$ 表示定时计数器

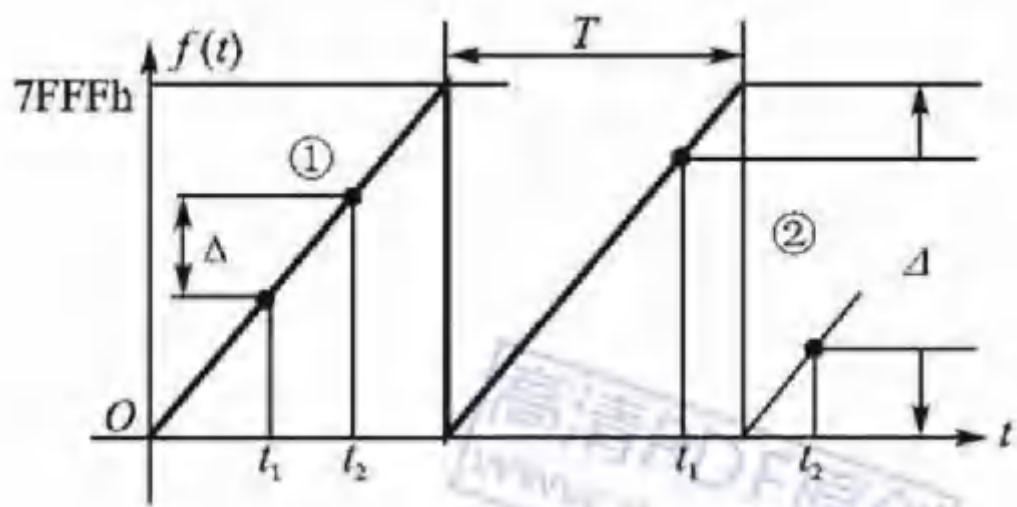


图 17.15 转速的计算

的值。两个脉冲上升沿之间的时间间隔 Δ 可以用 Q15 数据格式表示。在两个捕捉事件之间，定时计数器 1 的状态可能有两种：① 定时计数器 1 没有发生周期中断；② 定时计数器 1 已经发生中断。这两种情况下时间间隔的求取方法不一样。但是，只要保留当前和上一次的捕捉

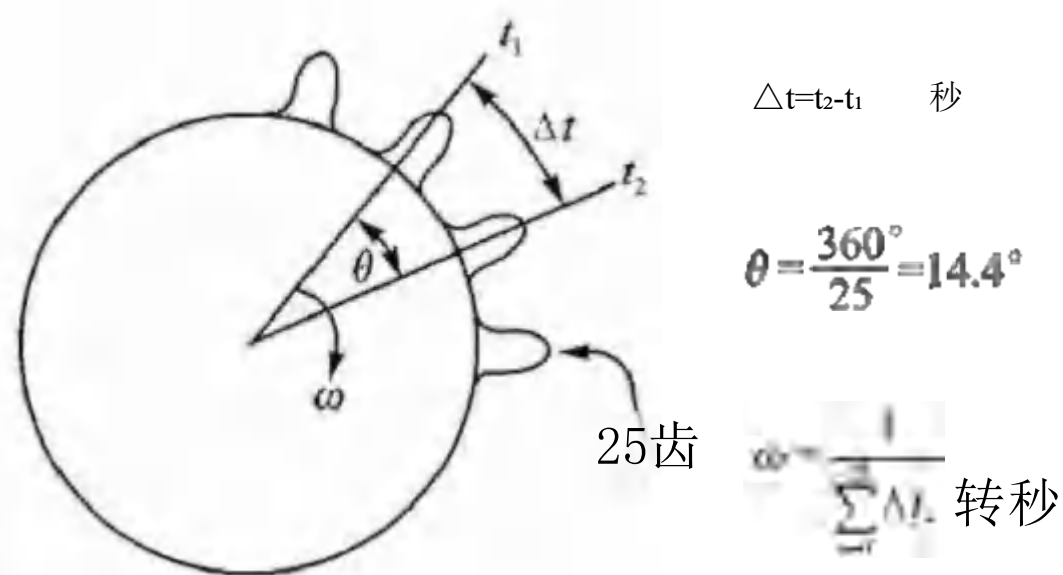


图 17.14 使用齿轮的转速测量原理

- 情况 ① $f(t_2) \geq f(t_1)$
 $\Delta = f(t_2) - f(t_1)$
- 情况 ② $f(t_2) < f(t_1)$
 $\Delta = 1 + f(t_2) - f(t_1)$

注：仅仅 $f(t_2) - f(t_1) < T$ 条件为真。

值，可以将这两种情况区分出来。

当捕捉到当前脉冲的周期值后，就可以计算出电机的角速度（转速）。为了能在整个调速范围内保证足够的转速测量精度，本方案采用图 17.16 所示的一个 32 位 / 16 位除法运算。

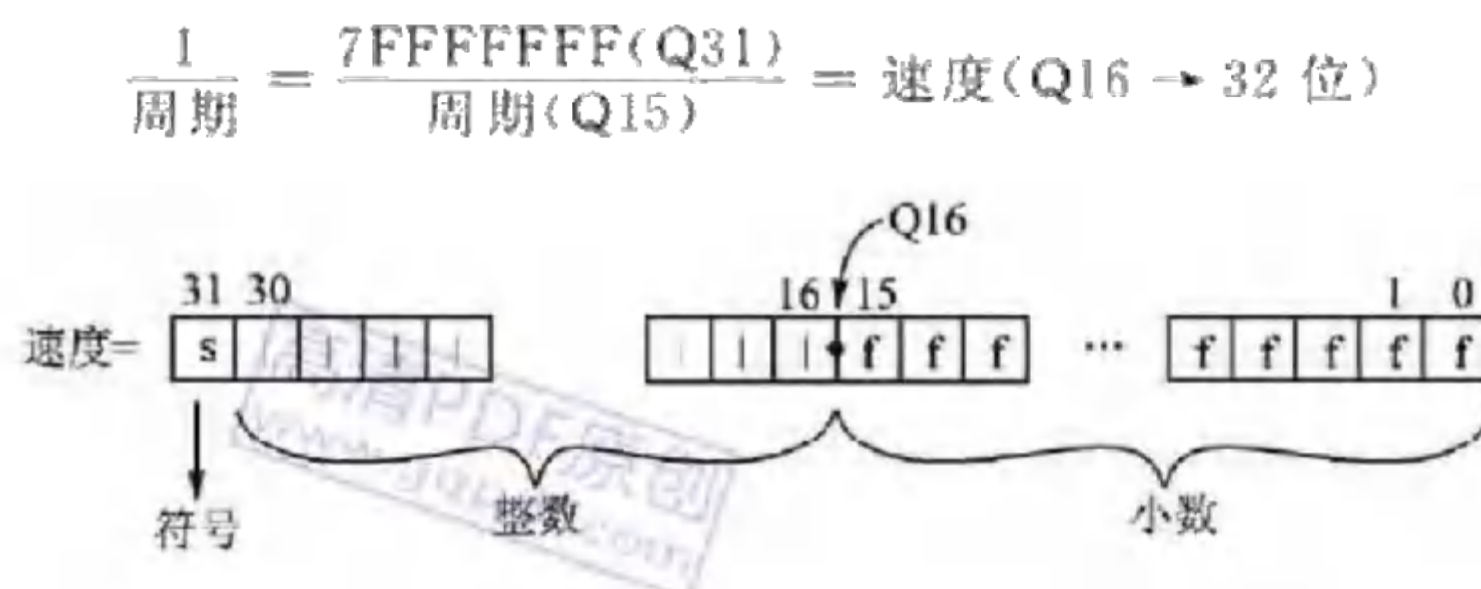


图 17.16 32 位 / 16 位的除法运算

32 位 / 16 位除法程序由 SUBC 指令实现。除法的结果是一个 Q16 格式的 32 位数，包括商和余数。这个值后来被分成一个 16 位的值。再经过规格化就可以得到一个 Q15 格式的商，这个商就是电机的实际转速。

17.2.4 闭环转速控制

本方案采用传统的离散 PI 调节器实现转速闭环控制。PI 调节器的传递函数可表示为：

$$G(s) = K_p + \frac{K_i}{s} \quad (17.26)$$

由式 (17.26) 可见，PI 调节器传递函数中包含有一个积分环节 K/s 。在离散域里，通常采用 3 种方法近似一个函数的积分：梯形法、前向矩形法和后向矩形积分法，如图 17.17 所示。很明显，随着采样频率的增加，这 3 种积分方式的效果会趋于一致，例如当 $\Delta T \rightarrow 0$ 时，上面 3 种方法就会得到相同的效果。

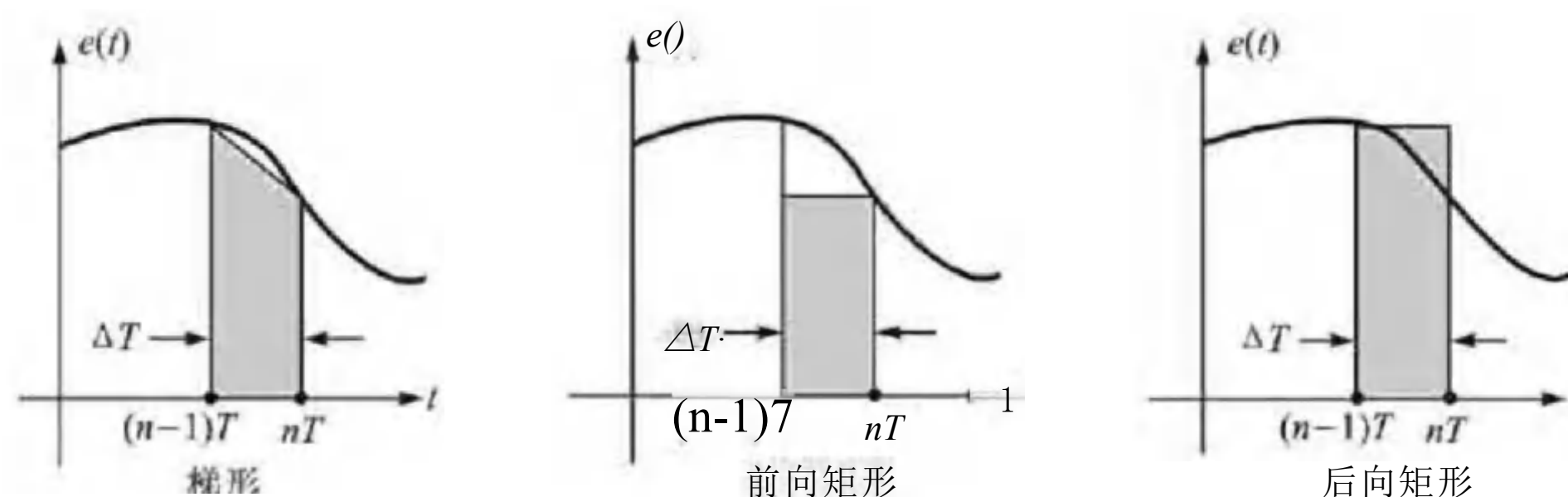


图 17.17 近似积分法

在本方案中，转速环的控制周期等于 PWM 开关周期。对于一个感应电机转速控制来说，此控制频率实在是太高了，但是这样程序的编写方便。因为 F2407 有足够的运算能力在 $25 \mu s$

内完成包括PI 调节器在内的整个控制算法。

在如此高的采样频率下，积分算法的选择不是很重要了，所以本方案采用最简单的前向矩形积分法，其Z变换如下：

$$\frac{K_i}{s} \Leftrightarrow \frac{K_i}{1-z^{-1}} \quad (17.27)$$

PI 调节器的最终差分方程如下：

$$Y_s = Y_{s-1} + K_{i\Delta n} + K_{p\Delta n} \quad (17.28)$$

图17.18为本方案的详细控制框图。此外，在编写PI 调节器的程序时，需要注意以下几点：

>PI 调节器的积分部分 $Y = Y_{s-1} + K_{i\Delta n}$ 采用32位长整型数参与运算，并以Q31 数据格式表示。

采用32位长整型数参与运算，并以Q31 数据

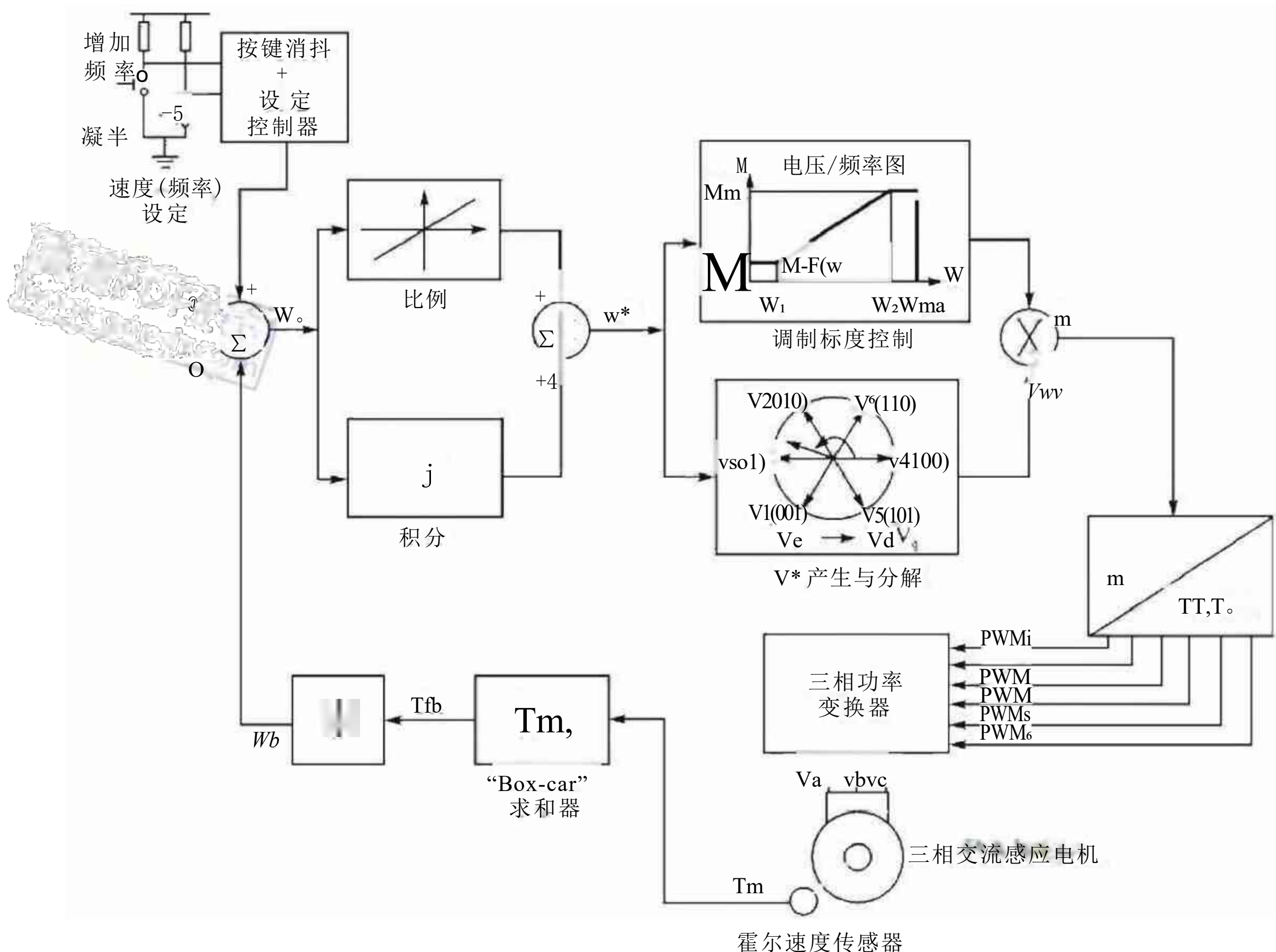


图17.18 转速闭环控制的三相感应电动机详细控制框图

- > **PI** 调节器的输出是频率给定 w^4 ，以 Q0 格式表示。其上、下限幅值分别为0和2048。
这个输出范围恰好对应于电压频率的变化范围：0~125 Hz。
- > 根据V/Hz 曲线和频率给定求出调制度 M。
- > 电机以转速开环的方式启动，2 s 后再投入闭环控制。这样可以保证霍尔传感器输出脉冲的宽度进入 F2407 捕捉单元的可测范围。
- > 在实际调试过程中，PI 调节器的参数 K 和 K 。可以通过试验的方式获取。尽管这种调试方法不是很科学，但是，在无法准确获取电机参数的情况下，这种方法是最方便的。最初可以选择一个比较小的值作为 K 和 K 。的初始值，然后慢慢增加，直到电机出现不稳定为止。记录下此时的PI 调节器参数，将这组参数减去一个较小值就可获得 PI 调节器的一组比较优化的稳定工作值。

17.3 三相感应电动机变频调速系统硬件设计

17.3.1 系统功率电路

图17.19为系统功率电路图。该电路由整流、滤波和逆变电路3大部分组成。整流电路采用单相桥式整流模块，结构简单，可靠性高。逆变电路用 IPM 模块，将驱动电路与IGBT 器件及保护电路集成到一起，提高了系统的稳定性并缩短了产品的开发时间。

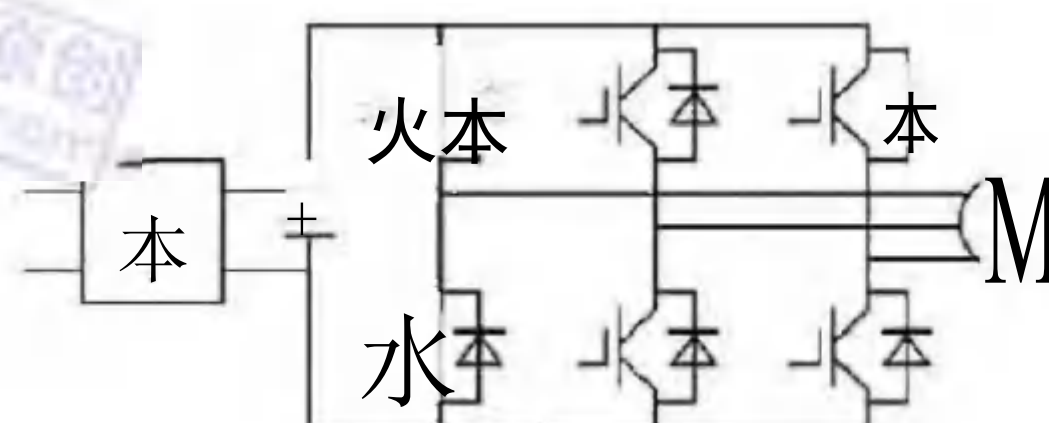


图17.19 系统功率电路图

17.3.2 逆变电路

1. IPM 模块简介

设计和开发智能功率模块 IPM(Intelligent Power Module)产品的主导思想是为应用装置的生产厂家降低在设计、开发和制造上的成本。IPM 一般使用IGBT 作为功率开关元件，不仅把功率开关器件和驱动电路集成在一起，而且还将过电压、过电流和过热等故障检测电路包含在里面，并可将故障保护信号送到CPU 或 DSP 进行处理。它与普通 IGBT 模块相比，在系统性能和可靠性上有进一步的提高，使设计和开发变得简单。由于IPM 通态损耗和开关损耗都比较低，使得散热器体积减小，因而系统尺寸也减小。它由高速低功耗的管芯和优化的门级驱动电路以及快速保护电路构成。如果 IPM 模块发生事故或使用不当而使得其中某种保护电路启动，则IGBT 栅极驱动单元就会关断驱动电流，并输出一个故障信号(FO)，可以使IPM 自身不被损坏。三菱公司的IPM 以其高可靠性和使用方便，赢得越来越大的市场。

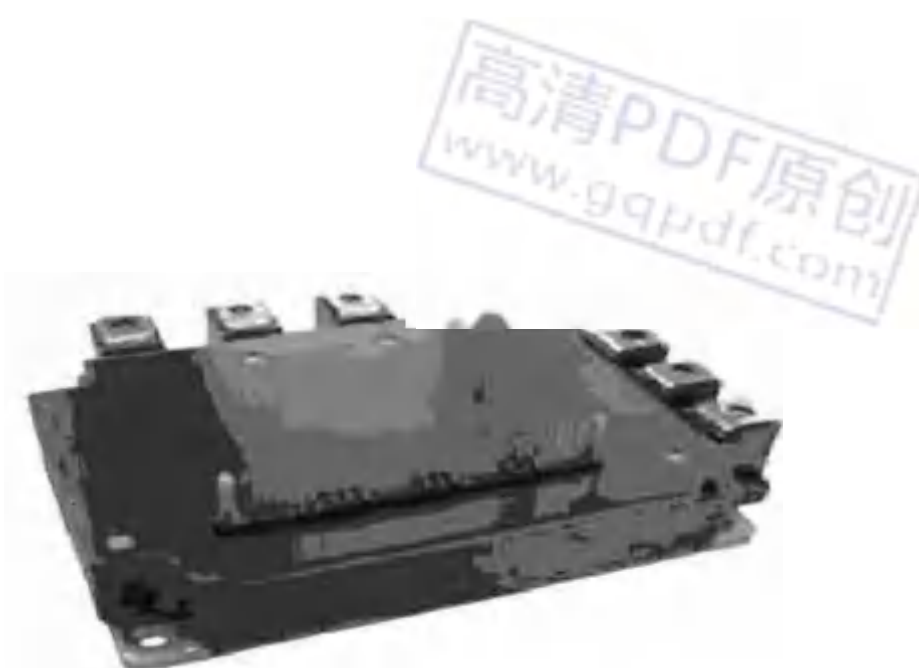
- › 开关速度快。IPM 内的 IGBT 芯片都选用高速型，而且驱动电路紧靠 IGBT 芯片，驱动延时小，所以 IPM 开关速度快，损耗小。
- › 低功耗。IPM 内部的 IGBT 导通压降低，开关速度快，故 IPM 功耗小。
- › 快速的过流保护。IPM 实时检测 IGBT 电流。当发生严重过载或直接短路的情况时，IGBT 将被软关断，同时送出一个故障信号。
- › 过热保护。在靠近 IGBT 的绝缘基板上安装了一个温度传感器。当基板过热时，IPM 内部控制电路将截止栅级驱动，不响应输入控制信号。
- › 桥臂对管互锁。在串联的桥臂上，上下臂的驱动信号互锁。可有效防止上下臂同时导通。
- › 抗干扰能力强。优化的门级驱动与 IGBT 集成，布局合理，无外部驱动线。
- › 驱动电源欠压保护。当电压低于驱动控制电源（一般为 15 V）时，就会造成驱动能力不够，增加导通损坏时损坏的概率。IPM 自动检测驱动电源，当电压低于一定值并且时间超过 10 μ s 时，将截止驱动信号。
- › IPM 内置相关的外围电路。可缩短开发时间，加快产品上市。
- › 无须采取防静电措施。
- › 大大减少了元件数目，体积相应减小。

2. Pm300cla060 智能功率模块简述

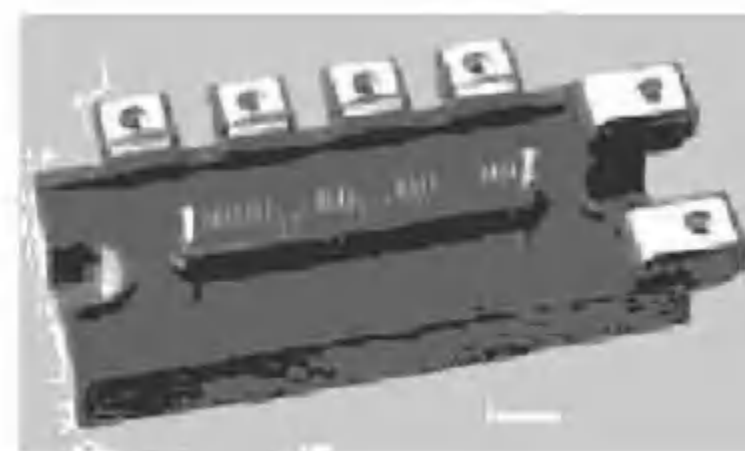
(1) 功能概述

本方案所选用的 IPM 模块是三菱公司推出的第 5 代隔离型智能功率模块 Pm300cla060，实物如图 17.20(a) 所示。图 17.20 (b) 为另一种类型的功率模块 Pm300cla060 封装与引脚定义如图 17.21 所示，它内部包含有一个三相逆变桥的 6 个 IGBT 和 6 个快恢复功率二极管及其相应的驱动电路和保护电路。每个 IGBT 功率单元均为 300 A/600 V。它避免了传统的 IGBT-IPM 模块，当工作温度迅速上升时可能被击穿问题，以及有时由于寄生电容或控制电路产生的寄生电感而引起的噪声干扰，使 IGBT 产生误动作的缺陷，而且其所用元件数量比传统 IPM 少得多，相应可靠性也得到提高。

Pm300cla060 内部功能模块如图 17.22 所示。它作为功率变换的应用，开关频率可以达到 20 kHz，并可提供更好的门极驱动性能和 IGBT 保护功能。由于采用了这种带 IGBT 驱动电路且保护功能完善的 IPM 模块，本系统的体积更小，更可靠，更具备智能化的特点。应用领域包括：功率变换、UPS、电机/伺服控制和供电控制等。



(a) Pm300cla060 实物图



(b) Pm75rla120 实物图

图 17.20 IPM 模块图

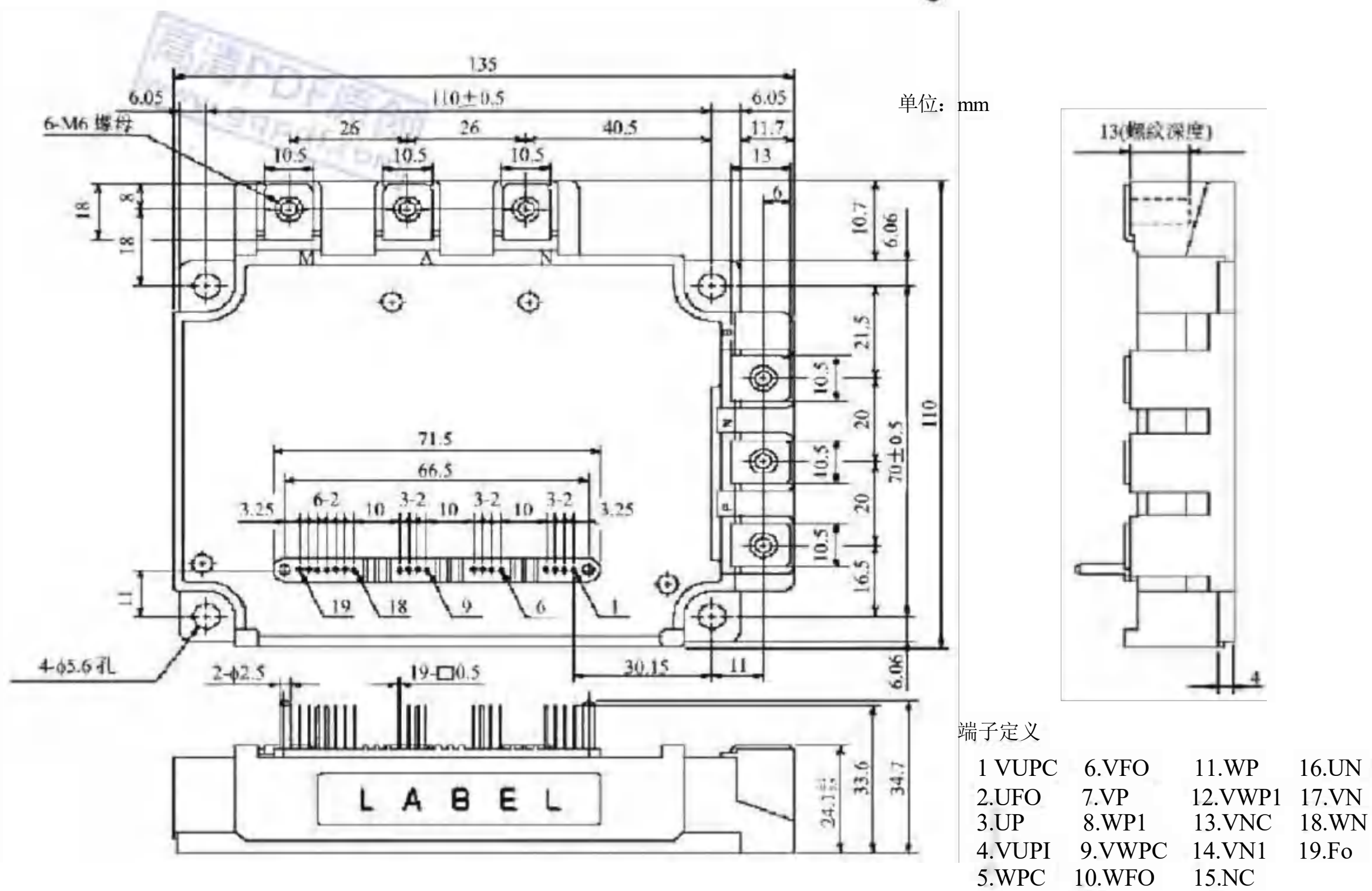


图 17.21 Pm300c1a060 封装与引脚定义

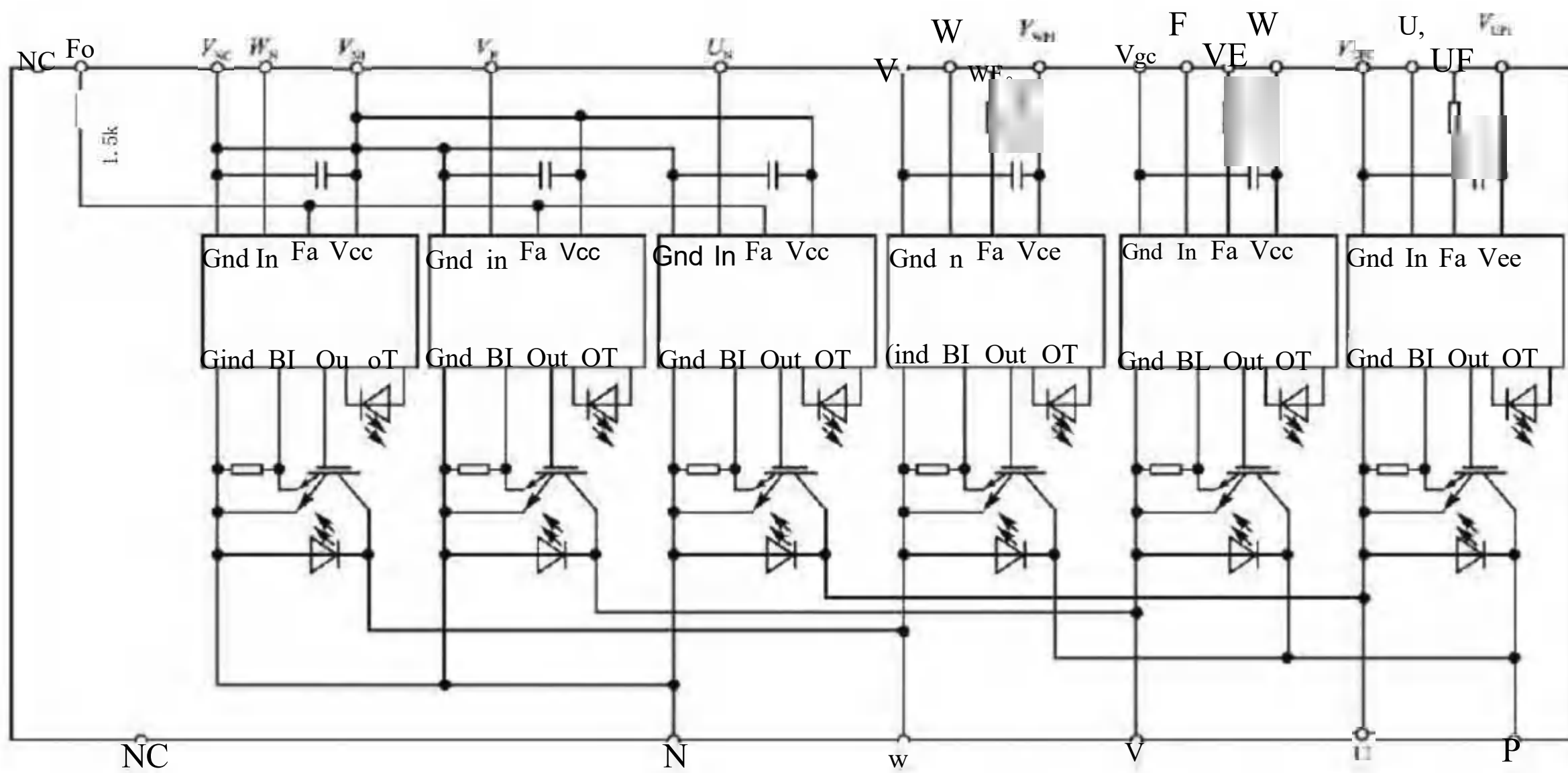


图17.22 Pm300cla060 内部电路图

2) 主要特性

采用第5代IGBT 硅片(CSTBT), 损耗更低。

>6单元紧凑型封装。

>通过采用新的 ASIC 控制di/dt 减少 EMI。

> 在硅片上设置温度传感器, 使过温保护更精确。

>门极驱动。

>保护逻辑: 短路、过热(在线温度传感)和欠压。

3) 输入/输出端子定义

Pm300cla060 输入/输出端子定义如表17. 5所列。

表17. 5 Pm300cla060 输入/输出端子列表

端子编号	端子名称	说 明	端子编号	端子名称	说 明
1	VUPC	U相P侧(高端)驱动电源的地	11	WP	W相P侧(高端)控制信号输入端
2	UFO	U相(高端)故障信号输出端	12	VWP1	W相P侧(高端)驱动电源的正端
3	UP	U相P侧(高端)控制信号输入端	13	VNC	N侧(低端)控制电源的地
4	VUP1	U相P侧(高端)驱动电源的正端	14	VN1	N侧(低端)控制电源的正端
5	VVPC	V相P侧(高端)驱动电源的地	15	NC	空脚
6	VFO	V相(高端)故障信号输出端	16	UN	U相N侧(低端)控制信号输入端
7	VP	V相P侧(高端)控制信号输入端	17	VN	V相N侧(低端)控制信号输入端
8	VVP1	V相P侧(高端)驱动电源的正端	18	WN	W相N侧(低端)控制信号输入端
9	VWPC	W相P侧(高端)驱动电源的地	19	FO	(低端)故障信号输出端
10	WFO)	W相(高端)故障信号输出端			

4) 各种保护功能

正常驱动。高电平导通驱动逻辑。低电平输入信号($V_{cN}<V_{hcof}$) 驱动 IGBT 关断, 高电平输入信号($V_{cN}>V_{h(on)}$) 驱动 IGBT 开通。

控制电源欠压锁定(U V)。内部控制电路由一个+15 V 直流电源供电。如果由于某种原因这一电源电压低于规定的欠压动作数值(UV),IGBT 将被关断并输出一个故障信号。但是电源电压上的小毛刺干扰(低电压时间小于规定的 td_{UV} 时间)时, 欠压保护电路不会动作。

应该注意的是, 在控制电源上电后不稳定的电压, 如果功率电路直流母线电压上升速率大于 20 V/ μ s,可能会损坏功率器件。若控制电源的电压上升的 dV/dt 大于 5 V/ μ s, 有可能引起欠压锁定误动作。

过热保护(OT)。在靠近IGBT 芯片的绝缘基板上安装了一个温度传感器。如果基板温度超出过热动作数值(OT),IPM 内部控制电路将截止栅极驱动信号, 不响应控制输入信号,

直到温度恢复正常为止，从而保护了功率器件。过热保护的動作条件苛刻，只能操作几次，应避免反复出现过热保护动作。

短路保护(S C)。如果负载发生短路或系统控制器故障而导致上下桥臂同时导通，IPM 内置短路保护电路将关断IGBT。当流经IGBT 的电流超出短路动作数值(SC) 时，软关断立即启动并输出一个故障信号。新型IPM 采用了实时电流检测控制技术 RTC，使电流过流响应时间小于100 ns。短路保护的動作都是IGBT 的过负荷运行，应避免其反复动作。

(5) 应用举例

Pm300cla060 应用举例如图17.23所示，其中DSP 与IPM 模块之间的光电隔离驱动电路见图17.25中的驱动板，F2407A 的 PWM 输出经74HC245 驱动后通过排线与驱动板连接。

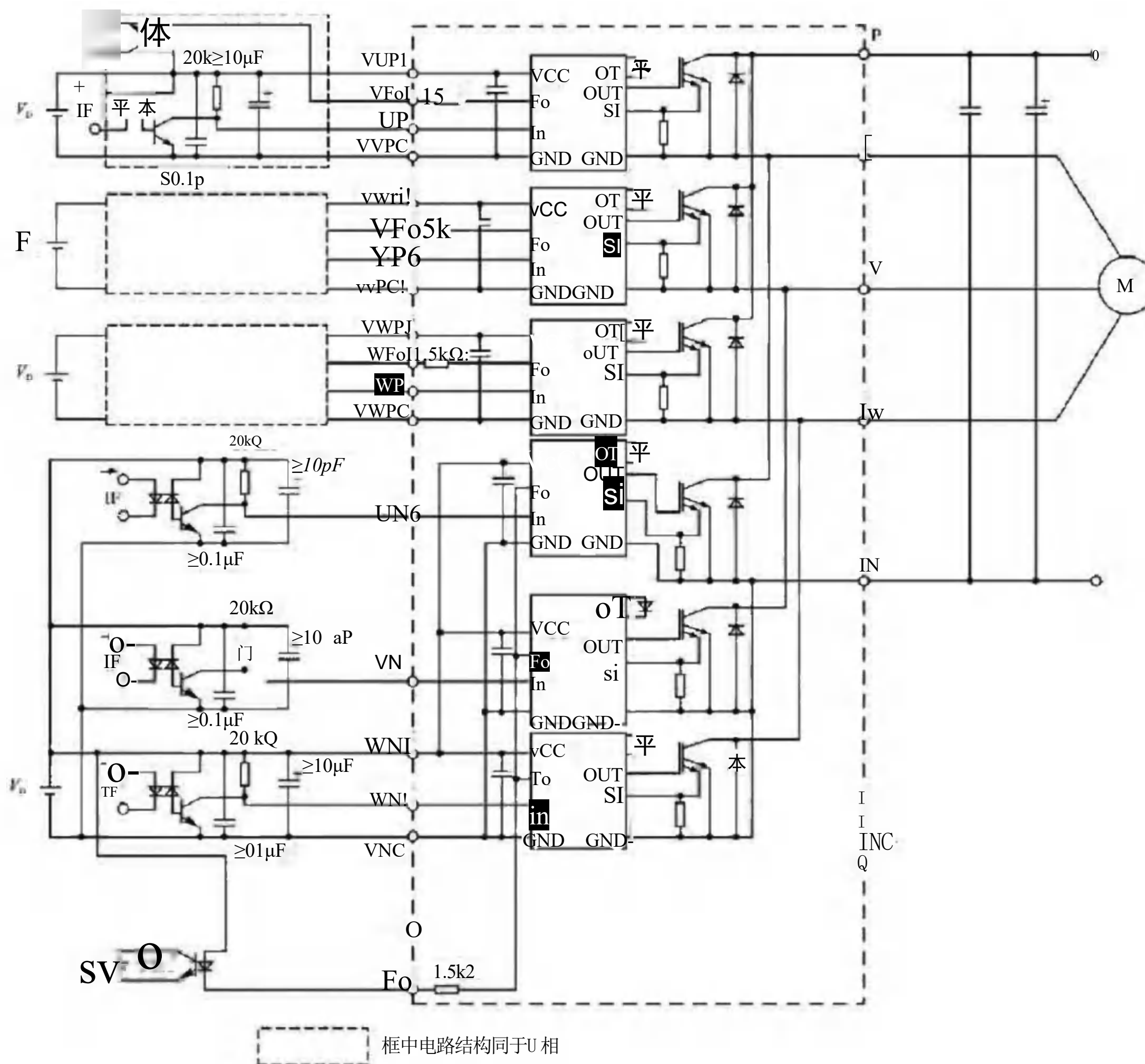


图17.23 Pm300cla060 应用举例

3. 系统硬件总体结构图

图17.24为 F2407A DSP控制的小功率异步电机变频调速系统实验平台的硬件结构框图。功率电路实物图17.25与功率电路配套的DSP 实验开发板实物图见图7.4。变频器系统用 DSP 作为运算控制单元，用 IPM 模块作为功率电路变换单元，用霍尔(Hall) 电流传感器检测电机三相中的两相电流，用光电编码器作为位置(转速)检测单元。三相工频电源经整流桥整流后，再经电容平滑滤波得到直流电压，送到逆变器 IPM 模块 Pm300cla060 的直流输入端 P 和 N 端。F2407A 控制器在对检测出的信号进行相应的运算处理后，由恒压频比算法来产生3个相互差 120° 的3对SVPWM 脉冲信号，经光电隔离后作用于 Pm300cla060 控制输入端(UP、VP、WP、UN、VN和 WN)。在 Pm300cla060 的输出端 U、V 和 W 输出3个相互差 120° 的高压SVPWM 波形来驱动异步电机，通过改变输出频率，实现异步电机变频调速。由于

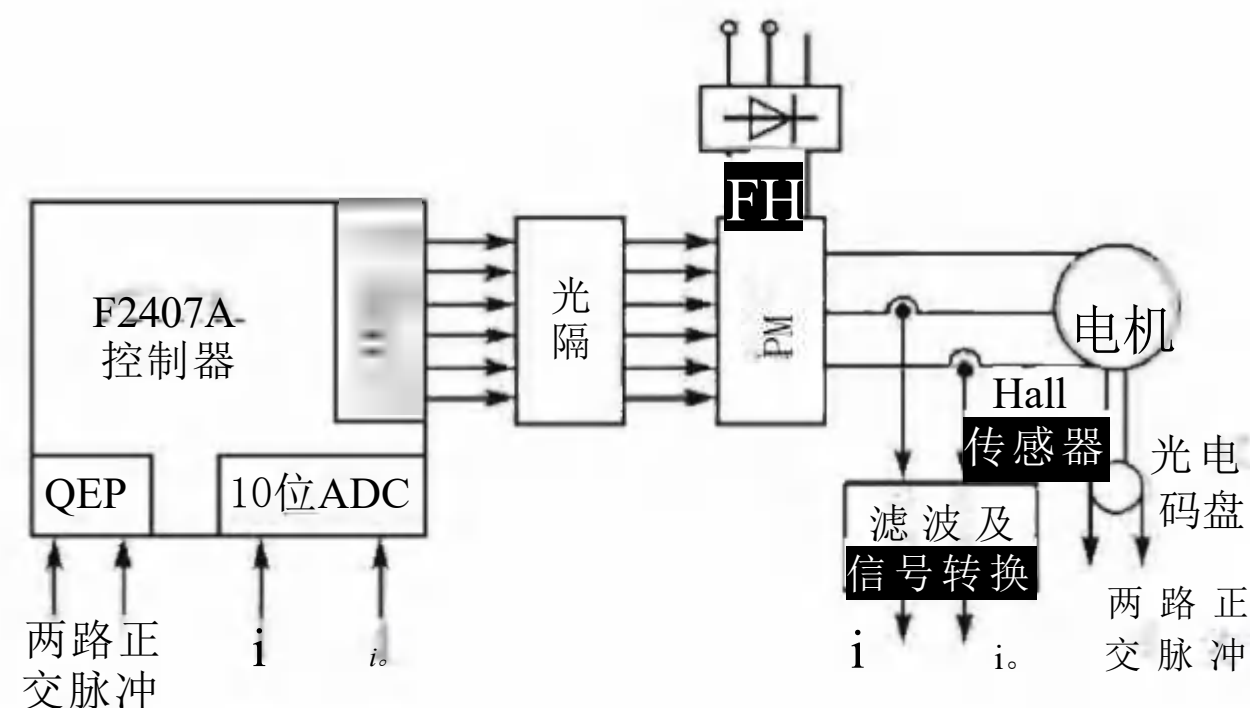


图17.24 控制系统的主要硬件结构框图

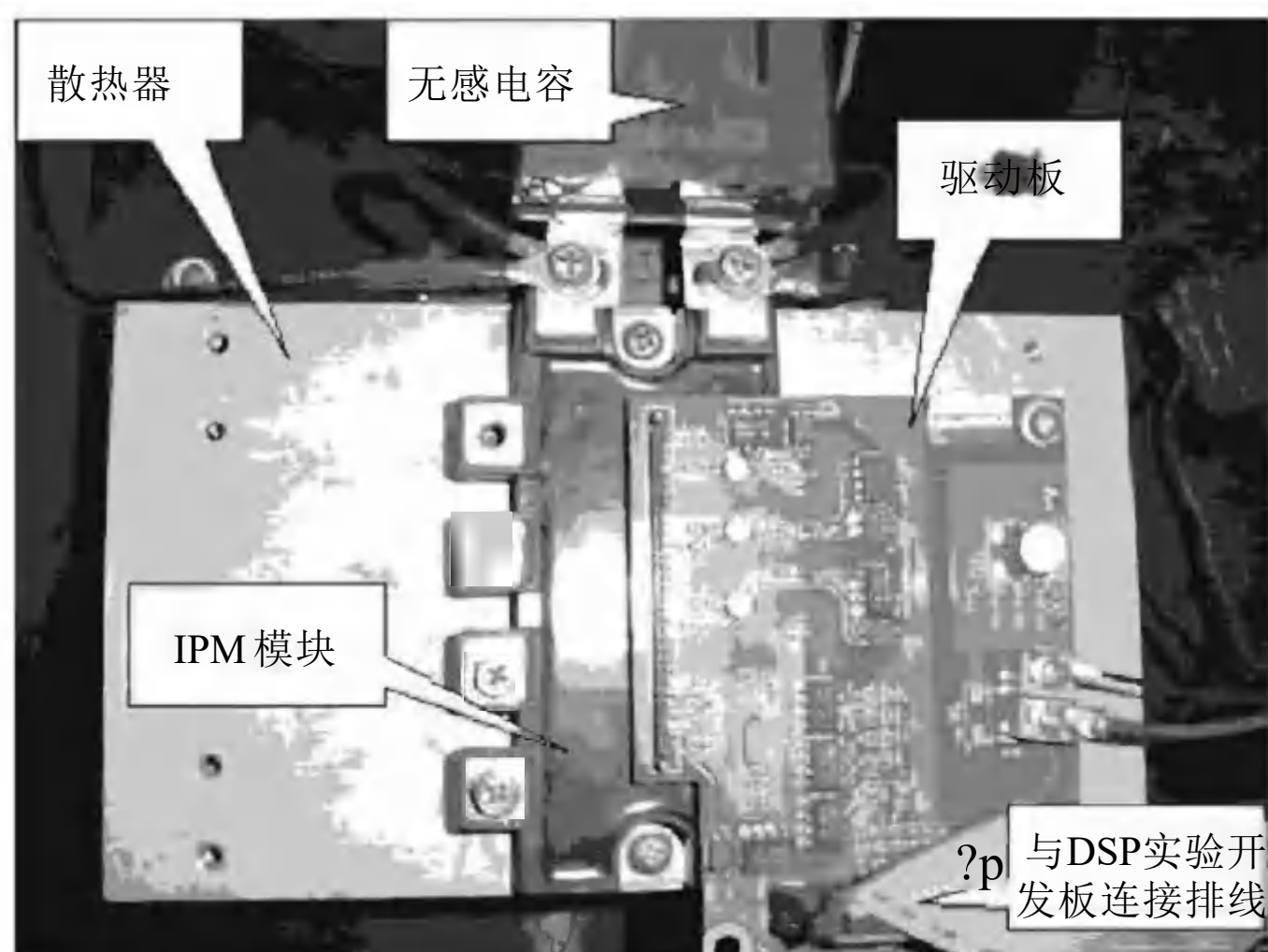


图17.25 变频器功率电路实物图

Pm300cla060 具有短路、欠压、过热探测电路及保护电路，当任何一种故障发生时，它都将封锁内部 6 只 IGBT 功率管，同时送出故障信号到 F2407A 的功率保护中断 PDPINTA 引脚；F2407A 硬件将立即封锁PWM 波形的输出，并向 F2407A 发出功率保护中断请求。变频器的散热器、IPM 模块(采用的是Pm75rla120)、驱动板和无感电容等如图17. 25所示。本书所附光盘中有本章所述的控制方法、电压/频率控制的整套源程序全部调试通过，程序名称和主要作用以及采用的编写语言等见表17. 6所列。

表17.6 变频控制软件程序模块功能

程序模块名称	模块主要功能	备 注
Cursample.asm	采样电机相电流云和i • 并转换为Q12格式(以额定电流为基准), 采用平均值滤波方式	汇编语言
freq_theta.asm	根据已知的参考频率计算△0	汇编语言
polar_rect,asm	实现电机定子电压量由两相旋转坐标系(U. U)向两相静止坐标系(U。 U)变换	汇编语言
qep.asm	转速测量	汇编语言
scopead.asm	将函数参数(Q11格式)转换至D/A芯片MAX5741的量程范围内. 以Q0格式表示	汇编语言
sincos.asm	采用查表、线性插值的方法同时求取函数参数angle(单位: 度)的正弦和余弦函数值	汇编语言
soft_svpwm.asm	根据给定电压量，计算生成空间矢量PWM所需的占空比，最终利用F2407内部的全比较模块输出给定电压	汇编语言
VECTORS.asm	中断矢量程序	汇编语言
volt_freq.asm	压频变换程序	汇编语言
Cursample.h	采样电机相电流 和i定义的变量和结构	C语言头函数
12407_c.h	F2407的C语言头函数	C语言头函数
freq_theta.h	freq_theta子程序的头函数	C语言头函数
polar_rect_c.h	polar_recte子程序的头函数	C语言头函数
Qep.h	Qep子程序的头函数	C语言头函数
Scopead.h	Scopead子程序的头函数	C语言头函数
Sincos.h	Sincos子程序的头函数	C语言头函数
softsvpwm_c.h	soft_svpwm子程序的头函数	C语言头函数
volt_freqc.h	volt_freqc子程序的头函数	C语言头函数
init_VHz_wx,c	系统和外围初始化子程序	C语言程序
int_yhz_wx.c	压频变换中断子程序	C语言程序
var_yhz_wx,c	定义主程序中用到的常量和变量	C语言程序
vhz_wx.c	C语言编写的主程序	C语言程序
RTS2XX.LIB	C语言库函数	C语言库函数文件
VHz_wx.prj	项目配置文件	项目文件
VHzc.cmd	命令文件.cmd	

附录 C C 语言库函数

本附录将对 F240x 的 C 语言库函数及宏定义进行详细说明。在以下说明中，用上标表示指数函数中的幂，例如 x^n ，相当于 C 语言中的 $x**y$ 。

1.Abort 函数

语 法

```
#include<stdlib.h>
void abort(void);
```

说 明

Abort(0) 函数用于中止包含有错误代码的程序。F240x 通过在函数中调用 exit(0) 函数来实现以上功能。

例 程

```
void abort()
{
    exit(0);
}
```

2.abs/labs 函数

语 法

```
#include<stdlib.h>
int abs(int j);
long int labs(long int k);
```

说 明

C 编译器支持以下两个函数，用于返回一个整数的绝对值。

① abs() 函数，返回整型数 j 的绝对值。

② labs() 函数，返回长整型数 k 的绝对值。

因为在 C 语言中，整型(int) 和 长整型(long int)是功能等效的数据类型，所以 abs() 函数和 labs() 函数也是功能等效的。除非在链接时选择 -x0 选项，否则编译器认为 abs() 和 labs

() 函数为内部(inline) 函数。

例 程

```
int    x=-5;
int    y=abs(x);    *  返回 - 5的绝对值*
```

3.acos 函数

语 法

```
#include<math.h>
double acos(double x);
```

说 明

acosO 函数将返回浮点参数x 的反余弦值，其中，x 在 $[-1, 1]$ 区间内。返回值是一个以弧度表示的角度，区间为 $[0, \pi]$ 。

例 程

```
double realval.radians;
realval=0.0;
radians=acos(realval);
return(radians);
```

4.asctime 函数

语 法

```
#include<time.h>
char *asctime(const struct tm *timeptr);
```

说 明

asctime() 函数将时间转化为字符串，返回指向该字符串的指针，为指针型函数。字符串格式如下：

```
Mon Jan 1111:18:361988 \n\0
```

如果需要了解更多关于 asctime() 函数在头文件time.h 中声明及定义等信息，那么请参见time() 函数。

5.asin 函 数

语 法

```
#include<math.h>
double asin(double x);
```

说 明

asin() 函数返回浮点型参数 x 的反正弦值, 参数 x 的区间为 $[-1, 1]$ 。返回值是一个单位为弧度的角度, 区间为 $[-\pi/2, \pi/2]$ 。

例 程

```
double realval.radians;
realval=1.0;
radians=asin(realval);    /*asin 返回  $\pi/2$ */
```

6.assert 宏**语 法**

```
#include<assert.h>
void assert(int expr);
```

说 明

assert 宏用于测试表达式。它可以通过判断表达式的值来决定是发布一个信息、中断当前任务还是继续执行当前任务。此宏在程序的调试时可起很大的作用。

- ① 如果表达式为“假”, 则assert宏将记录相关信息, 并且停止执行当前任务。
- ② 如果表达式为“真”, 则assert宏将不进行任何操作。

定义 assert 宏的头文件还可以应用另外一个宏——NDEBUG。如果源文件中包含有 assert.h 头文件, 并且将 NDEBUG 作为宏名, 则assert 宏将无效。

如果NDEBUG 不被定义成宏名, 并且源文件中包含有 assert.h 头文件, 则当表达式为“假”时, assert 宏将生成一条诊断信息, 里面包括源文件名、被测试的表达式及其行数。

assert 宏与printf() 函数定义在一起, 但是printf() 函数并没有包含在头文件中。在使用 assert 宏时, 有以下两种选择:

- ① 提供自定义的 printf() 函数;
- ② 通过别的方式修改assert 宏的输出信息。

例 程

本例将实现两个整数的除法运算, 即 i/j 。由于在进行除法操作时, 除数不能为0, 所以本例在运算前使用assert 宏测试除数 j 。当 j 为 0 时, assert 宏将输出信息, 并且中止程序运行。

```
int i,j;
assert(j);
q=i/j;
```

7.atan 函 数

语 法

```
井include<math.h>
double atan(double x);
```

说 明

atan() 函数将返回浮点参数x 的反正切值。返回值单位为弧度，区间为[$-\pi/2, \pi/2$]。

例 程

```
double realval,radians;
realval=1.0;
radians=atan(realval);      / 返回值为 0
```

8.atan2 函数

语 法

```
井include<math.h>
double atan2(double y,double x);
```

说 明

atan2() 函数返回y/x 的反正切值，返回值所在的象限由参数的符号决定，单位是弧度，区间为 $[-\pi, \pi]$ 。另外，此函数的两个参数都必须不为0。

例 程

```
atan2(1.0,1.0)      返回值为  $\pi/4^*$ 
atan2(1.0,-1.0)     返回值为  $3\pi/4^*$ 
atan2(-1.0,1.0)     *返回值为  $-\pi/4^*$ 
atan2(-1.0,-1.0)    返回值为  $-3\pi/4$  关
```

9.atexit 函数

语 法

```
井include<stdlib,h
void atexit(void( fun)(void));
```

说 明

atexit() 函数记录指针 fun 所指的函数名，这些函数都是在正常的程序末尾处调用的，并且省略了函数参数。本函数最多可以记录32个函数名。

当程序是通过调用exit() 函数退出，或者程序异常中止，或者程序从 main() 函数中返回

时，程序将按记录的逆序逐个调用被记录的函数，并忽略函数参数。

10.atof/atoi/atol 函数

语 法

```
#include<stdlib.h>
double atof(const char *st);
int atoi(const char *st);
long int atol(const char*st);
```

说 明

atof()、atoi() 和 atol() 这3个函数分别将字符串转换为数字表达式：

- ① atof() 函数将一个字符串转换为浮点数。参数st 是指向字符串的指针；被转换的字符串必须是以下格式：

[空格工符号]数字[. 位数][e|E [符号]整数]

- ② atoi()函数将一个字符串转换为整数。参数st 是指向字符串的指针；被转换的字符串必须是以下格式：

[空格[符号]数字

- ③ atol() 函数将一个字符串转换为长整形数。参数st 是指向字符串的指针；被转换的字符串必须是以下格式：

[空格工[符号]数字

其中，“空格”可以是空格键、TAB 键、回车键、换页符和换行符等，跟在“空格”后的是一个“符号”任选项，“数字”代表数字的整数部分。“数字”后的小数部分用指数表示，并且包括一个“符号”任选项。

字符串的第1个字符不能是数字。

另外，由于在C 语言中，整型数据和长整型数据是相同的，所以 atoi() 函数和 atol() 函数从功能上讲也是相同的。

以上3个函数在转换过程中将忽略所有的溢出。

例 程

```
int i;
double d;
i=atoi("-3291");      *i=-3291*/
d=atof("1.23e-2");      *d=.0123*/
```



11. bsearch 函数

语 法

```
并include<stdlib.h>
```

```
void *bsearch(const void key,const void base,size_t nmemb.  
size_t size,int(compar)(const void*,const void ));
```

说 明

bsearch() 函数从未知矩阵中搜索与关键词相匹配的矩阵元素。 base 指向矩阵的第一个元素； size 规定每个元素的大小(以字节表示)。 size_t 为矩阵参数定义符号。

矩阵的内容必须按升序排列。当出现匹配时，函数将返回指向矩阵中匹配元素的指针；如果矩阵中没用元素与关键词相匹配，则函数返回空指针。

参数compar 指向将关键词与矩阵元素相比较的函数。此比较函数的声明如下：

```
int cmp(const void*ptr1,const void ptr2);
```

cmp() 函数比较指针prt1 和 prt2 指向的目标，然后返回以下值中的一个：

```
<0      ptr1  小于 * ptr2  
0       *ptr1 等于 * ptr2  
>0      ptr1 大于 * ptr2
```

例 程

```
并include<stdlib.h>
```

```
#include<stdio.h>
```

```
int list[]={1,3,4,6,8,9};
```

```
int idiff(const void ,const void 苦 ) ;
```

```
main()
```

```
{
```

```
int key=8;
```

```
int p=bsearch(&.key,list,6,1,idiff); p 指向list[4]地 址 位 置 *
```

```
}
```

```
int idiff(const void *i1,const void i2)
```

```
(
```

```
return*(int)i1 - (int *)i2;
```

```
}
```

12.calloc 函数

语 法

```
#include<stdlib.h>
void*calloc(size_t num,size_t size);
```

说 明

calloc() 函数为对象分配存储器字节长度(整型长度或由size_t 指定), 分配成功返回指向被分配空间的指针, 同时分配的单元的内容初始化为0。如果calloc() 函数没能实现内存分配(即没有足够的内存空间), 则返回一个空指针(0)。

calloc()函数使用的内存单元称为寄存器组或寄存器堆, 它定义在一个名为.sysmem 的未初始化区域中。常数_SYSTEM_SIZE限定堆为1K 字, 也可以通过选择_heap 选项并在选项后直接规定需要堆的大小。

例 程

本例将使用calloc() 函数分配20字节的存储空间。

```
prt=calloc(20,2);          /* 分配20字节*/
```

13.ceil 函数

语 法

```
#include<math.h>
double ceil(double x);
```

说 明

ceil() 函数返回一个用浮点数表示大于或等于x 的最小整数。如果使用-x2 选项, 则ceil() 函数为内部函数。

例 程

```
double answer;
answer=ceil(3.1415);          /*answer=4.0*/
answer=ceil(-3.5);           /*answer=-3.0*/
```

14.clock 函数

语 法

```
#include<time.h>
clock_t clock(void);
```


说 明

clock(O)函数用于测定处理器使用的时间，并把它作为返回值。被测时间的单位由返回值除以宏CLOCKS_PER_SEC 的值后所得的商决定。如果处理器时间无法测得或无法表示，则clock() 函数的返回值为- 1。

注 意

clock0 函数是由目标系统定义的，所以用户必须自己编写clock0)函数。同时，也必须定义宏CLOCKS_PER_SEC 的值，用以确定自定义时钟的单位；否则clock() 函数无法使用返回值与CLOCK_PER_SEC 的商生成时钟单位值。

15.cos 函 数**语 法**

```
#include<math.h>
double cos(double x);
```

说 明

cos() 函数返回浮点数x 的余弦值，其中，x 用弧度表示。此外，当x 很大时，则可能导致返回值很小或者无意义。

例 程

```
double radians,cval;
radians=3.1415927;
cval=cos(radians);    * 返回值=- 1.0元
```

16.cosh 函 数**语 法**

```
#include<math.h>
double cosh(double x);
```

说 明

cosh() 函数返回浮点数x 的双曲余弦值。当函数参数值太大时，函数会出错。

例 程

```
double x,y;
x=0.0;
y=cosh(x);    返回值为1.0*
```

17.ctime 函数

语 法

```
#include<time.h>
char*ctime(const time_t *timer);
```

说 明

ctime() 函数将指针 timer 所指的日历时间转换为本地时间，并以字符串的形式保存起来，其功能等同于asctime(localtime(timer))。 本函数返回由 asctime() 函数返回的指针。

18.difftime 函数

语 法

```
#include<time.h>
double difftime(time_t time1,time_t time0);
```

说 明

difftime() 函数用于计算两个日历时间之差，即 time1 减去 time0，其返回值用 s 表示。

19.div/ldiv 函数

语 法

```
#include<stdlib.h>
div_t div(int numer.denom);
ldiv_t ldiv(long numer.denom);
```

说 明

div() 函数和ldiv() 函数均可实现整数除法(numer/denom)， 返回商并确定余数。

① div() 函数实现整数除法。其函数参数为整数，返回值是div_t 类型的结构，包括商和余数两个部分。结构定义如下：

```
typedef struct
```

```
    int  quot;    /* 商 */
    int  rem;     /* 余数*/
}div_t;
```

② ldiv()函数实现长整型数除法。函数参数为长整型变量，返回值为ldiv 类型的结构，包括商和余数两个部分。结构定义如下：

```
typedef struct
```

```
long int quot;    * 商
long int rem;     /* 余数 */
ldiv_t;
```

如果两个整数不能整除，那么本函数得到的商将是与代数商同号，绝对值为小于代数商绝对值的最大整数，余数符号与商一致。

例 程

```
int    i=-10;
int    j=3:
div_t   result=diy(i,j);    商 = - 3 *
                             余数 = - 1
```

20.exit 函数

语 法

```
井include    stdlib.h
void exit(int status):
```

说 明

exit() 函数将正常中正程序的运行，并且将 atexit() 中登记的函数按逆序调用一次，但不能实现函数返回。还可以用exit() 函数实现专用的关机任务。

在C 语言库函数中，exit() 函数与abort() 函数是等价的。

21.exp 函数

语 法

```
井include    math.h>
double exp(double x);
```

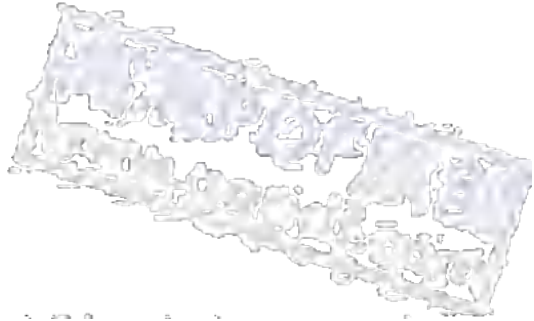
说 明

exp() 函数的返回值为实数x 的幂函数值，即 e^x 。注意，在a 过大的情况下，函数的返回值将出现较大的误差。

例 程

```
double x,y;
x=2.0;
y=exp(x);    y=7.38905. 即 $e^2$ ★/
```





22.fabs 函数

语 法

```
#include<math.h>
double fabs(double x);
```

说 明

fabs() 函数返回浮点数x 的绝对值。如果在编译程序时不使用 -x0 选项，则fabs() 函数为内部函数。

例 程

```
double x,y;
x=-57.5;
y=fabs(x);          * 返回值为+57.5×/
```

23.floor 函数

语 法

```
#include<math.h>
double floor(double x);
```

说 明

floor() 函数的返回值是一个浮点数，用于表示小于或等于 x 的最大整数。如果编译程序时不使用 -x0 选项，则 floor() 函数为内部函数。

例 程

```
double answer;
answer=floor(3.1415);      /*answer=3.0*/
answer=floor(-3.5);        /*answer=-4.0*/
```

24.fmod 函数

语 法

```
#include <math.h>
double fmod(double x,double y);
```

说 明

fmod() 函数返回x/y 的余数，并以精确浮点数表示。如果y=0， 则函数的返回值为0。

例 程

```
double x,y,r;
```

```
x=11.0;
y=5.0;
r=fmod(x,y);    r=1.0#
```

25.free 函 数

语 法

```
井include<stdlib.h>
void free(void*packet);
```

说 明

free() 函数释放由malloc()、calloc() 或 realloc()、call() 等指令预先分配的存储空间(由指针 packed 指定起始地址)。这使存储空间可反复使用。但是, 如果试图释放未经预先分配的空间, 那么本函数将不执行任何操作。

例 程

本例程将分配10字节的存储空间, 然后将它们全部释放。

```
char x;
x=malloc(10);    × 分配10字节*
free(x);         释 放10字节*
```

26.frexp 函 数

语 法

```
井include<math.h>
double frexp(double value,int*exp);
```

说 明

frexp() 函数将一个浮点数拆分为一个规格化小数(f) 和一个2的整数幂(exp)。函数的返回值为 f 和 exp; 其中, $0.5 < |f| < 1.0, value = f \times 2^{\exp}$, frexp() 函数返回的整数幂将保存在整形指针 exp 所指的单元中。如果 value=0, 则两个返回值都为0。

例 程

```
double fraction;
int exp;
fraction=frexp(3.0.8.exp);    素小数0.75, 幂为2#
```

27.gmtime 函数

语 法

```
#include<time.h>
struct tm *gmtime(const time_t*timer);
```

说 明

gmtimeO 函数将 timer 指针所指的日历时间转换成格林威治时间。

28.inport/outputport 函数

语 法

```
#include<ioports.h>
inport(int port,int *ret);
outport(int port,int value);
```

说 明

以下的宏指令用于访问 I/O 端口：

- ① inport 宏将从规定的I/O 空间读取数据，然后通过指针 ret 将数据返回给调用的程序。
- ② outport 宏将对规定的I/O 空间进行写操作；本操作无返回值。

以上程序将以宏的方式执行，而不是函数，所以它们不能用于表达式中。以下的用法是不正确的：

```
call(inport(1,&i));          * 宏定义的不正确用法*/
```

正确用法如下：

```
inport(1,&i);
call(i);                    /* 正确用法 */
```

此外，I/O 通道数必须在1~15之间。使用任意其他的通道数都会导致不确定的操作。

如果只使用一个固定的 I/O 通道数，那么可以将ioport.h 中定义的常数 _PSWITCH 设置成0(缺省)。反之，则需要将该常数设置为1。但是，无论 _PSWITCH 为何值，宏 inport/outport 总会对 I/O 空间进行操作。

29.isxxx 函数

语 法

```
#include<ctype.h>
int isalnum(int c);      int islower(int c);
int isalpha(int c);      int isprint(int c);
```



```
int  isascii(int  c);          int  ispunct(int  c);
int  iscntrl(int  c);          int  isspace(int  c);
int  isdigit(int  c);          int  isupper(int  c);
int  isgraph(int  c);          int  isxdigit(int  c);
```



说 明

本函数将对参数c 进行测试，并判断该参数是不是某一特定的字符——字母、字母数字、数字及 ASCII 码等。如果测试的结果为真，则函数返回一个非0的值；反之，返回值为0。如果在编译程序时选择—x 选项，则以下所有字符类型函数为内部函数。

isulnun() 辨别字母数字ASCII 字符；
 isalpha() 辨别字母 ASCII 字符；
 isacii() 辨 别 ASCII 字 符；
 iscntrl() 辨别控制字符；
 isdigit() 辨别数字字符；
 isgraph(): 辨别任何非空格字符；
 islower() 辨别小写字母 ASCII 字符；
 isprint() 辨别可打印的ASCII 字符，包括空格(ASCII 码32~126)；
 ispunct() 辨 别ASCII 标点符号；
 isspace() 辨别空格、TAB（横或者竖）、回车符、换页符和换行符；
 isupper() 辨别大写 ASCII 字母；
 isxdigit() 辨别十六进制数(包括0~9, a~f,A~F)。

此外，C 编译器还提供一系列宏来实现以上功能。这些宏的名字由相应的函数名前面加下划线构成，例如，宏_isascii 与 isascii() 函数的功能是相同的，而且宏的执行效率要比函数高。

30.Idexp 函 数

语 法

```
井include<math.h>
double Idexp(double x,int exp);
```

说 明

ldexp() 函数实现浮点数x 与 2^{exp} 的乘积，返回值为 $(x*2^{\text{exp}})$ 。指 数exp 可以是任何实数。此外，如果结果太大，则会引起一定的误差。

例 程

```
double result;
result=ldexp(1.5,5);          * 结 果 为 4 8 . 0 共
```

```
result=ldexp(6.0,-3);      /* 结果为0.75*/
```

31.localtime 函数

语 法

```
#include<time.h>
struct tm* localtime(const time_t *timer);
```

说 明

localtime() 函数将指针 timer 所指的日历时间转换为时间段，用当地时间表示出来。函数返回指向被转换时间的指针。

32.log 函数

语 法

```
#include<math.h>
double log(double x);
```

说 明

log() 函数返回实数 x 的自然对数值。此外，当 x 小于或等于 0 时，函数返回值将出现一定的误差。

例 程

```
float x,y;
x=2.718282;
y=log(x);      /*返回值为1.0*/
```

33.log10 函数

语 法

```
#include <math.h>
double log10(double x);
```

说 明

log10 (函数返回实数 x 以 10 为底的对数。当 x 小于或等于 0 时，函数返回值将出现一定的误差。

例 程

```
float x,y;
x=10.0;
y=log(x);      /*返回值为1.0*/
```

34.ltoa 函数

语 法

```
并include<stdlib.h>
int ltoa(long val, char buffer );
```

说 明

ltoa() 函数将长整型数 val 转换成相应的 ASCII 字符串，并写入指针 buffer 所指单元中。如果 val 为负，则输出字符串将以负号开始。ltoa() 函数返回保存在 buffer 中的字符个数。

例 程

```
int i;
char s[10];
i=ltoa(-92993L,s);          *i=6,s="-92993"
```

35.malloc 函数

语 法

```
#include<stdlib.h>
void malloc(size_t size);
```

说 明

malloc() 函数为长度为 size 的对象分配存储空间，并返回指向该空间的指针。如果 malloc() 函数无法实现空间分配，则函数的返回值为0。 malloc() 函数不修改被分配单元中的内容。

malloc() 函数以存储器组的形式使用存储单元。常数__SYSTEM_SIZE的值确定了系统堆的深度为1K 字。同时，通过选择链接器—heap 选项修改系统堆的深度。

例 程

```
struct xy2 p;
p=malloc(sizeof(struct xyz));
```

36.memchr 函数

语 法

```
#include<string.h>

void memchr(const void es, int c, size_t n);
```

说 明

memchr() 函数实现在es 指针指向的n 个字符中查找与 c 相匹配的第1个字符。如果能

够找到，则返回该字符的地址；否则返回空指针(0)。

memchr()函数与strchr() 函数相类似，但是，memchr() 函数查找的目标可以为0或包含0，而 strchr()函数则不行。当选择-x 选项时，memchr() 函数为内部(inline)函数。

37. memcmp 函数

语 法

```
#include<string.h>
int memcmp(const void *cs,const void *ct,size_t n);
```

说 明

memcmp() 函数比较指针 ct 所指单元和指针 cs 所指单元之间n 个字符的内容，返回以下3个值中的任意一个：

```
<0      如果* cs 小于* ct
0       如果*cs 等于* ct
>0      如果* cs 大于* ct
```

memcmp() 函数和 strncmp() 函数相类似，但是，memcmp() 函数所比较的目标可以为0。当选择-x 选项时，memcmp()函数为内部(inline) 函数。

38. memcpy 函数

语 法

```
#include<string.h>
void *memcpy(void *s1,const void*s2,size_t n);
```

说 明

memcpy() 函数将指针 s2 所指的 n 个字符复制到指针 s1 所指的单元中。如果试图复制重叠目标，则本函数的行为是不确定的。本函数的返回值为指针 s1。

memcpy() 函数与strncpy() 函数相类似，但是前者可以复制包含0的目标字符。当使用-x 选项时，memcpy() 函数为内部函数。

39.memmove 函数

语 法

```
#include<string.h>
void*memmove(void*s1,const void *s2,size_t n);
```

说 明

memmove() 函数将指针s2 所指的 n 个字符转移到指针 s1 所指的n 个单元中，函数的返回值为指针 s1。memmove() 函数可以实现两个重叠目标之间的正确复制。

40. memset 函数

语 法

```
#include<string.h>
```

```
void memset(void mem,int ch,size_t length);
```

说 明

memset() 函数将字符 ch 复制到指针 mem 所指长度为 length 字符串的第1个字符中，函数返回mem 的指针。当选择—x 选项时，memset() 函数为内部函数。

41.minit 函 数

语 法

```
#include<stdlib.h>
```

```
void minit(void);
```

说 明

minit() 函数将会清除调用malloc(),calloc() 和 realloc() 等函数分配存储单元留下的空隙空间。

注意，minit() 函数的调用会使堆中的所有存储单元重新使用，预先分配好的任何对象将会丢失。

42.mktime 函 数

语 法

```
#include<time.h>
```

```
time_t mktime(struct tm timeptr);
```

说 明

mktime() 函数将原来的局部时间转换为对应的标准日历时间。指针参数 timeptr 指向的结构包含原来的时间。

函数忽略 tm_wday 和 tm_yday 的原始值，不对结构中的其他值进行限制。完成对时间的转换后，tm_wday 和 tm_yday 被赋予适当的值，其他成员的值在限制范围内。tm_yday 的值只有在确定了 tm_mon 和 tm_year 之后才能被确定。

返回值的类型为 time_t。如果日历时间不能被表示出来，则函数返回值为-1。

例 程

此程序可用来确定2001年7月4日是星期几。

```
#include<time.h>
```

```
static const char * const
```

```
wdays={"Sunday","Monday","Tuesday","Wednesday",
```

```

        "Thursday","Friday","Saturday"};

struct tm time_str;
time_str.tm_year=2001-1900;
time_str.tm_mon=7;
time_str.tm_mday=4;
time_str.tm_hour=0;
time_str.tm_min=0;
time_str.tm_sec=1;
time_str.tm_isdst=1;
mktime(&time_str);    /*通过调用此函数，time_str.tm_wday 中包含了这一天是星期几*/
printf("result is %s\n",wday[time_str.tm_wday]);

```

43.modf 函数

语 法

```

#include<math.h>
double modf(double value,double×ipart);

```

说 明

modf() 函数将一个数分为符号整数和符号小数。这两部分都和输入的参数符号相同。函数返回符号小数，并将整数部分以双精度类型存储在指针 ipart 指向的存储单元里。

例 程

```

double value,ipart,fpart;
value=-3.1415;
fpart=modf(value,&ipart);
/*函数执行后，ipart 的值为-3.0。fpart 的值为-0.1415。*/

```

44.pow 函数

语 法

```

#include<math.h>
double pow(double x,double y);

```

说 明

pow() 函数返回 x 的 y 次方的值。当 $x=0$ 且 $y \leq 0$ ，或者 $x < 0$ 且 y 不为整数时，将出现定义域错误。如果结果太大而无法表达，则出现范围错误。

例 程

```

double x,y,z;

```



```
x=2.0;
y=3.0;
z=pow(x, y);      *   返回值为8.0*
```

45.qsort 函数

语 法

```
include<stdlib.h>
void  qsort(void*base,size_t  nmemb,size_t  size,int(compar)
(const  void  *,const  void*));
```

说 明

qsort() 函数对共有nmemb 个成员的数组进行排序。 base 参数指向未排序数组的第1个成员；参数size 用来规定每个成员的大小。该函数按上升次序对数组进行排序。

Compar 参数指向一个函数，该函数用某一关键数与数组元素进行比较。这个用来比较的函数应该这样声明：

```
int  cmp(*ptr1, ptr2)
void  *ptr1,*ptr2;
```

cmp() 函数比较 ptr1 和 ptr2 指向的单元的大小，返回下面的值：

当 ptr1 小于* ptr2 时，返回值小于零；

当 * ptr1 等于 * ptr2 时，返回值等于零；

当 * ptr1 大于 * ptr2 时，返回值大于零。

例 程

下面的例程中，用qsort() 函数对个数较少的一列整数进行排序。

```
井include<stdlib.h>
int  list[]={3.1.4.1.5.9.2.6};
int  idiff(const  void*,const  void*);
main()
{
    qsort  (list,8,1,idiff);      *   排序后 list[]  中的值为(1. 1. 2. 3. 4, 5. 6. 9) */
}
int  idiff(const  void#i1,const  void  i2)
{
    return  关 (int  *) 江一  (int  )i2;
```

46.rand、srand 函数

语 法

```
#include<stdlib.h>
int    rand(void);
void   srand(unsigned int  seed);
```

说 明

这两个函数一起共同作用产生随机数。

rand() 函数返回一个 0 ~RAND_MAX 之间的整数。对于C 编辑器，RAND_MAX 为 2147483646(231-2)。

Srand() 函数赋给 seed 一个值，从而下一次调用 rand() 函数时产生一个新的随机数。Srand() 函数没有返回值。

在调用srand() 函数之前调用rand() 函数，与先调用参数seed 为 1 的srand() 函数再调用rand() 函数情形一样，都产生同一个随机数顺序。如果调用srand() 函数时的seed 值相同，则rand() 函数产生相同顺序的数字序列。

47.realloc 函 数

语 法

```
#include<stdlib.h>
void *realloc(void *packet,size_t size);
```

说 明

realloc() 函数用来改变已分配的内存空间的大小。这些已分配内存空间以字节为单位，存储的内容不会被改变。

- ① 如果已分配的内存空间为0, 那么 realloc() 函数相当于 malloc() 函数。
- ② 如果指向未分配的存储空间，那么函数将不执行任何操作并返回。
- ③ 如果存储空间不能被重新分配，那么原来的存储空间将不会被改变，realloc() 函数返回0。
- ④ 如果已分配存储空间的大小为0, 并且原来已分配内存空间不为空，那么 realloc() 函数将释放已分配内存空间指向的存储空间。

如果为了分配更多的空间而必须移动整个目标，那么 realloc() 函数将返回一个指针，指向新的存储空间。任何通过这种方法释放的内存都被返还回内存堆中。如果出现错误，则函数将返回一个空指针(0)。

这些被 realloc() 函数使用的内存存在一个特定的存储区或堆中。常量__SYSTEM__ 定义堆的大小为1K 字。用户可以在连接时使用-heap 选项直接改变堆的大小。

48.Setjmp/longjmp 函 数

语 法

```
井include<setjmp.h>
int setjmp(jmp_buf env);
void longjmp(jmpbuf env.int returnval);
```

说 明

头文件setjmp.h 定义了一个数据类型、一个宏，并声明了一个函数，用来跳过正常函数的调用和返回。

- ① jmp_buf 数据类型是一个数组类型，可以用来保存需要恢复的调用环境。
- ② 宏 setjmp 将调用环境保存在数组jmp_buf 中，以方便调用longjmp() 函数。如果从一个直接调用返回，则宏 setjmp 返回0。如果从longjmp() 函数返回，则宏 setjmp 返回一个非零值。
- ③ longjmp() 函数用来恢复保存在数组 jmp_buf 中的现场环境，这些现场环境是由最近激发的宏setjmp 所保存的。如果宏setjmp 没有被调用或在运行时突然被终止，那么 longjmp() 函数的特性将不会被定义。在 longjmp() 函数执行完之后，程序将继续运行。这和由对应的宏setjmp 返回的情形一样。即使 Longjmp() 函数的返回值为0，Longjmp() 函数也不会引起宏setjmp 返回0。如果 Longjmp() 函数的返回值非0，则宏 setjmp 返回1。

例 程

下面的例程是使用以上函数实现从深嵌套函数间接返回的典型例子。

```
井include<setjmp.h>
jmp_bufenv;
main()
{
    int errcode;
    if((errcode=setjmp(env))==0)
        nest1();
    else
        switch(errcode)
        :
    }
    nest42()

    if(input()== ERRCODE42)
```



```

/* 返回到主程序 main() 中调用 setjmp() 处 */
longjmp(env.ERRCODE42);
:
}

```

49.sin 函数

语 法

```

#include <math.h>
double sin(double x);

```

说 明

sin() 函数返回浮点数 x 的正弦值。其中，角度 x 用弧度表示，当 x 很大时，产生的结果可能错误。

例 程

```

double    radian,sval;    /*sval    存放sin 的值*/
radian=3.1415927;
sval=sin(radian);        *sin 返回结果为 -1.0*

```

50.sinh 函数

语 法

```

#include    <math.h>
double sinh(double x);

```

说 明

sinh() 函数返回浮点数 x 的双曲线正弦值。如果参数绝对值太大，那么将产生范围错误。

例 程

```

double    x,y;
x=0.0;
y=sinh(x);    /* 返回值为0.0*/

```

51.sprintf 函数

C 编辑器所提供的实时函数不包括像 sprintf() 函数那样的 I/O 函数。但是，如果 time() 函数调用了 sprintf() 函数，那么 sprintf 将提供最小存储和 printf() 函数功能，这些功能是运行 time() 函数格式所必需的。更多信息参见本附录对 sprintf() 函数的说明。

52.sqrt 函 数

语 法

```
井include    math.h>
double sqrt(double x);
```

说 明

sqrt() 函数返回实数x 的非负平方根。如果参数值为负，那么将产生定义域错误。

例 程

```
double  x,y;
x=100.0;
y=sqrt(x);      * 返回值=10.0
```

53.srand 函 数

srand 函数的应用可参见rand()/srand() 函数。

54.Strcat 函 数

语 法

```
井include<string.h>
char  *strcat(char  *string1,char  string2);
```

说 明

strcat() 函数将字符串2(包括终止字符)复制悬挂到字符串1的末尾。字符串2的第1个字符将覆盖原来的字符串1的结束符。函数返回字符串1的值。当使用了一x 选项时，如果字符串1不够包含整个字符串2，strcat() 函数内部自动将字符串1扩大。

例 程

下面的例程中，指针*a、*b 和 *c 分别指向注释所说明的字符串。注释中，标志\0代表空格字符：

```
char      *a,*b,*c;
:
a-->“The  quick  black  fox\0” /
FMb-->“jumps  over  \0”X7
/  c-->“the  lazy  dog.\0”
strcat(a,b);
a-->“The  quick  black  fox jumps over \0”
strcat(a,c);

/*a-->"The      quick      black      fox      jumps      over      the      lazy      dog.\0" 淋
```

55.strchr 函 数**语 法**

```
#include<string.h>
char*strchr(const char*string,char c);
```

说 明

strchr() 函数用来查找字符c 第一次在字符串string中出现的位置。如果 strchr() 函数找到这样的字符, 那么函数将返回指向这个字符的指针; 否则, 返回空指针(0)。如果使用了—x 选项, 则strchr() 函数为内部函数。

例 程

```
char*a="When zz comes home,the search is on for z's.";
char*b;
char the_z='z';
b=strchr(a,the_z);
```

在执行完这个例程后, b 指向zz 的第一个字符z。

56. strcmp/strcoll函数**语 法**

```
#include <string.h>
int strcoll(const char*string1,const char*string2);
int strcmp(const char string1,const char string2);
```

说 明

这两个函数用来将字符串2与字符串1进行比较。两个函数是等价的; 两者都由ANSI C 标准支持。在strcmp() 函数中, 当使用了一x 选项时, 函数为内部函数。函数的返回值如下:

当字符串1小于字符串2时, 返回值小于0;

当字符串1等于字符串2时, 返回值等于0;

当字符串1大于字符串2时, 返回值大于0。

例 程

```
char *stra="why ask why";
char *strb="just do it";
char *strc="why ask why";
if(strcmp(stra,strb)>0)
{
    /* 将执行此处程序×/
```



```
x
if(strcoll(stra,src)==0)
```

也将执行此处程序/

57.strcpy 函 数

语 法

```
井include<string.h>
char strepy(char string1,const char*string2);
```

说 明

strcpy() 函数将字符串string2（包括结束符）复制到字符串string1。strcpy() 函数不能复制相互重叠的字符串。strcpy() 函数返回指向字符串string1 的指针。当使用了一x选项时，strcpy() 函数为内部函数。

例 程

下面的例程中，被*a 和 *b 指向的字符串是两个独立的不相互重叠的存储区。注释中，标志\0代表空格字符：

```
char a="The quick black fox";
char b="jumps over";
/*a-->"The quick black fox\0"
*b-->"jumps over\0"*/
strcpy(a,b);
a-->"jumps over \0"
/ b-->"jumps over A0"
```

58.strcspn 函数

语 法

```
井include<string.h>
size_t strcspn(const char xstring,const char *chs);
```

说 明

strcspn() 函数返回字符串1开始部分长度，这部分的字符是字符串2中没有的。如果字符串string1 的第1个字符能在字符串string2 中找到，那么函数返回0。

例 程

```
char stra="who is there?";
```

```
char*strb="abcdefghijklmnopqrstuvwxy";
char*strc="abcdefg";
size_t length;
length=strcspn(stra.strb);      /*length=0*/
length=strcspn(stra,strc);      /*length=9*/
```

59.strerror 函数

语 法

```
井include <string.h>
char *strerror(int errno);
```

说 明

strerror() 函数返回字符串中的“字符错误”。提供这个函数以与ANSI 兼容。

60.Strftime 函数

语 法

```
#include<time.h>
size_t*strftime(char *out,size_t maxsize,const char *format,
const struct tm*time);
```

说 明

strftime(0)函数根据字符串格式将一个时间(由time 指针指向的)格式化,函数返回值包含在输出字符串中的格式化结果。这个结果包含的字符受到可被写出最大数量字符串的限制。格式参数以字符串形式给出,它告诉 strftime() 函数如何格式化时间。下面列出有效字符并说明每个字符代表的意义。

```
%a  星期几的缩写(Mon,Tue,...);
%A  星期几的全称;
%b  月份的缩写(Jan,Feb,...);
%B  当地月份的全称;
%c  表示日期和时间;
%d  用十进制表示的一个月的第几天(0~31);
%H  用十进制表示的小时数(24小时制)(00~23);
%I  用十进制表示的小时数(12小时制)(01~12);
%j  用十进制表示的一年中的天数(001~366);
%m  用十进制表示的月份(01~12);
%M  用十进制表示的分钟数(00~59);
```



%p 表示当地时间的上午(A.M) 和 下午(P.M);
 %S 用十进制表示的秒数(00~59);
 %U 用十进制表示的一年中的星期数(星期天是一个星期的第1天)(00~52);
 %x 代表日期;
 %X 代表时间;
 %y 用十进制表示的不包括世纪的年份(00~99);
 %Y 用十进制表示的包括世纪的年份;
 %Z 时间的地区名, 如果没有时间区域则不显示字符。

61.strlen 函 数

语 法

```
#include <string.h>
size_t strlen(const char* string);
```

说 明

strlen() 函数返回字符串的长度。在C语言中, 一个字符串以值为0的字节(即空格符)结束。返回的结果中不包括结束符。当使用了一x选项时, strlen() 函数为内部函数。

例 程

```
char   stra="who   is   there?";
char   strb="abcdefghijklmnopqrstuvwxyz";
char   strc="abcdefg";
size_t  length;
length=strlen(stra);           *length=13*/
length=strlen(strb);           length=26
length=strlen(strc);           length=7
```

62.strncat 函 数

语 法

```
#include <string.h>
char *strncat(char *dest,const char src,size_in);
```

说 明

strncat() 函数将src字符串中的n个字符(包括结束符)连接到dest字符串末尾。Src字符串的第1个字符将dest字符串的结束符覆盖; strncat() 连接一个空格结束符到结果字符串dest的末尾。函数返回字符串dest指针。

例 程

下面的例程中，指针* a、*b 和 *c 分别指向注释所说明的字符串。

```
char      *a,*b,*c;
size_t   size=13;
:
/*a      ->"I do not like them.\0"*/;
/*b----->"Sam      I      am,\0"*/;
/*c----->"I do not like green eggs and ham\0"*/;
strncat(a,b,size);
/*a-->"I do not like them,Sam I am,\0"*/;
/*b----->"Sam      I      am,\0"*/;
/*c----->"I do not like green eggs and ham\0"*/;
strncat(a,c,size);
/*a----->"I do not like them,Sam I am,I do not like\0"*/;
/*b----->"Sam      I      am,\0"*/;
/*c----->"I do not like green eggs and ham\0"*/;
```

63.strncmp 函 数

语 法

```
#include<string.h>
int  strncmp(const  char  *string1,const  char*string2.size_t  n);
```

说 明

strncmpO 函数将字符串 string2 中的n 个字符与字符串 string1 进行比较。函数的返回值如下：

当字符串1小于字符串2时，返回值小于0；

当字符串1等于字符串2时，返回值等于0；

当字符串1大于字符串2时，返回值大于0。

例 程

```
char *stra="y ask why";
char *strb="just do it";
char*strc="why      not?";
size_t size=4;
if(strncmp(stra,strb,size)>0)
{
    /*此处程序被执行*/
```

```
if(strncmp(stra,src,size)==0)
```

此处程序被执行

64.strncpy 函数

语 法

```
#include<string.h>
```

```
char strncpy(const char*dest,const char src,size_tn);
```

说 明

strncpy() 函数将字符串 src 中的 n 个字符复制到 dest 字符串。如果字符串 src 长度为 n 或者更长，空格结束符将不会被复制。对试图进行相互重叠的字符串进行复制的行为，函数没有进行定义。如果 src 字符串长度小于 n，那么 strncpy 将空格结束符复制到 dest 字符串，这时字符串 dest 的长度小于 n。函数返回字符串 dest。

例 程

本例中应注意，strb 字符串以空格开始，因此其长度为5。字符串src 的前面5个字符分别为 I.1 个空格，1个单词 am,1 个空格，所以执行第2个 strncpy() 函数后，字符串 stra 为 I am 再加两个空格。注释中，标志\0代表空格字符。

```
char stra="she's the one mother warned you of";
char*strb="he's";
char src="I am the one father warned you of";
char *strd="oops";
size_t length=5;
strncpy(stra,strb,length);
stra——>"he's the one mother warned you of\0";
strb——>"he's";\0";
/*src-->"I am the one father warned you of\0"著*/;
strd-->"oops\0"*/;
strncpy(stra,src,length);
/*stra——>"I am the one mother warned you of\0" /;
strb-->"he's";\0";
/*stre-->"I am the one father warned you of\0"*/;
—— strd-->"oops\0"*/;
strncpy(stra+strd,length);
```

```
/*stra-->"oops\0"*/;
/*strb-->"he's";\0"*/;
/*strc-->"I am the one father warned you of\0"*/;
/*strd-->"oops\0"*/;
```

65. strpbrk 函数

语 法

```
#include<string.h>
char *strpbrk(const char*string,const char*chs);
```

说 明

strpbrkO 函数查找包含在 chs 字符串中的任何字符第1次出现在字符串 string 中的位置。如果 strpbrk 找到一个相匹配的字符，那么它将返回一个指向那个字符的指针；如果没找到，那么它将返回空指针(0)。

例 程

```
char*stra="it wasn't me";
char *strb="wave";
char *a;
a=strpbrk(stra,strb);
```

执行完这个程序后，a 将指向 wasn't 中的 w。

66.strrchr 函数

语 法

```
#include<string.h>
char *strrchr(const char *string,int c);
```

说 明

strrchr() 函数用来查找字符 c 最后一次在 string 中出现的位置。如果 strrchr() 函数找到对应的字符，它将返回一个指向该字符的指针；否则，返回空指针(0)。当使用 -x 选项时，strrchr() 函数为内部函数。

例 程

```
char *a="When zz comes home,the search is on for z's";
char*b;
char the_z='z';
b=strrchr(a,the_z);
```



执行完这个程序后，*b 指向字符串最后一个 z。

67.strspn 函数

语 法

```
#include<string.h>
```

```
size_f strspn(const char+string,const char chs);
```

说 明

strspn() 函数返回string 字符串中开始部分的长度，这部分全部由字符串chs 中的字符组成。如果string 字符串中的第1个字符不是 chs 字符串中的字符，则 strspn() 函数返回0。

例 程

```
char stra="who is there?";
char strb="abcdefghijklmnopqrstuvwxy";
char strc="abcdefg";
size_t length;
length=strspn(stra,strb);          length=3*
length=strspn(stra,strc);          *length=0
```

68.strstr 函数

语 法

```
#include<string.h>
```

```
char *strstr(const char *string1,const char string2);
```

说 明

strstr() 函数查找字符串 string2 第1次在字符串string1 中出现的位置(不包括结束符)。如果strstr() 函数找到相匹配的字符串，那么它将返回指向这一字符串的指针；如果没有找到，那么它将返回空指针(0)。如果 string2 指向的字符串长度为0, 那么 strstr() 函数返回字符串 string1。

例 程

```
char stra="so what do you want for nothing?";
char strb="what";
char *ptr;
ptr=strstr(stra,strb);
```

现在*ptr 指针指向第1个字符串中的 w。



69. strtod/strtol/strtoul函数

语 法

```
#include<stdlib.h>
double  strtod(const char *test,char **endptr);
long int strtol(const char*test,char**endptr,int base);
unsigned long int strtoul(const char *test,char**endptr,int base);
```

说 明

这3个函数将 ASCII 字符串转换为数字量。每一个函数中的参数 `test` 指向原始字符串。参数 `endptr` 指向一个指针变量；函数使这个指针变量指向转换后字符串的第1个字符。当函数需要将原始字符串转换为整型时，还有第3个参数`base`，它将通知函数将字符串转换为何种进制的整型数。

① `strtod()` 函数将字符串转换为浮点数值。字符串必须有下面的格式：

[空位] [符号]数字[. 数字][e|E [符号]整型]

函数返回被转换后的字符串；如原始字符串为空或者格式不正确，函数返回0。如果转换的字符串将要引起上溢，函数返回正负 `HUGE_VAL`；如果转换的字符串将要引起下溢，函数返回0。

② `strtol()` 函数将一个字符串转换为一个长整型数。字符串必须使用下面的格式：

[空位工[符号]数字[. 数字][e|E [符号]整型]

③ `strtoul()` 函数将一个字符串转换为一个无符号长整型数。字符串必须使用下面的格式：

[空位工符号]数字[. 数字]e|E [符号]整型]

空位可以用以下方法表示：空格、水平线、竖线、回车或另起一行；空位后面是数据整型部分的符号位及数值；再接下来是小数部分，然后是包括符号在内的指数部分。字符串中第1个未被承认的字符代表字符串结束。指针 `Endptr` 指向这一字符。

70.Strtok 函 数

语 法

```
#include<string.h>
char *strtok(char *str1,const char *str2);
```

说 明

连续调用`strtok()` 函数可以将由`str1` 指向的字符串分为几组字符，每组字符之间由 `str2` 指向的字符串中的字符隔开。每次调用时函数返回指向接下来的一组字符的指针。第1次调用时使用字符串`str1`。然后使用空指针(0)作为第1个参数连续调用。在每次调用`strtok()`

函数时 Str2 的值可以不同。特别注意字符串str1 是被 strtok() 函数改变了的。

例 程

在下面的例程中，第工次调用 strtok() 函数后，stra 指针指向字符串excuse\0，因为 strtok() 函数在第1个空格原来的地方插入了一个结束符。在下面的注释中，“\0”代表结束符。

```
char  stra="excuse me while I kiss the sky";
char  *ptr;
pir=strtok(stra,"");    *ptr-->"excuse\0"*

ptr=strtok(0,"");      ptr-->"while\0"
```

71.tan 函数

语 法

```
井include<math.h>
double tan(double x);
```

说 明

tan() 函数返回浮点数x 的正切值，角度x 用弧度表示。如果参数的绝对值太大，那么产生的结果可能很小或者没有意义。

例 程

```
double  x,y;
x=3.1415927/4.0;
y=tan(x);          返回值为1 . 0*
```

72.tanh 函数

语 法

```
井include      math.h>
double tanh(double x);
```

说 明

tanh() 函数返回浮点数 x 的双曲正切值。

例 程

```
double  x,y;
x=0.0;
y=tanh(x);          返回值为0.0
```


73.time 函数

语 法

```
#include<time.h>
time_t time(time_t *timer);
```

说 明

time() 函数用来确定当前的日历时间，它可以表示从1900年1月1日中午12:00开始的时间，其精度为s。如果日历时钟无效，则函数返回-1。如果 timer 不是空指针，则函数也将时间存放到 timer 指针指向的存储单元。

注 意

timeO 函数是一个针对特定目标系统的函数，因此用户必须编写自己的时钟函数。

74.Ti sprintf 函数

语 法

```
#include<stdlib.h>
int ti_sprintf(char*s,const char *format,...);
```

说 明

ti_sprintf() 函数是 sprintf(O) 函数的简要版本，它只支持 time() 函数所必须的转换格式。ti_sprintf() 只支持下面的转换格式：

%[0][digits](s |d)

0 如果有，则表示这个区域将被0填充，而不是空白。

digits 如果有，则表示字符区域的最小宽度。如果转换格式对应的参数小于这一宽度，那么参数右对齐，左端用0或空白填充(根据上一项中是否为0而定)。

s |d 规定参数的类型。s 表示参数类型为 char*;d 表示参数类型为整型。

用户可以从 rst.src 中提取ti__sprintf() 函数，并改变它的程序代码，重新编辑实时函数库。为了更容易地改变ti sprintf()函数，ti__sprintf() 函数注释得十分详细。要从 rst.src 中提取 ti_sprintf(O) 函数，可在命令行输入下面的命令：

```
dspar-x rts.src tsprintf.c
```

75.toascii 函数

语 法

```
#include<ctype.h>
int toascii(int c);
```

说 明

toascii() 函数通过转换整型数c 的低7位确保 c 为一个有效的 ASCII 字符。函数有一个等价的宏调用toascii。

76.Tolower 和 toupper 函数

语 法

并include <ctype.h>

int tolower(int c);

int toupper(int c);

说 明

这两个函数可将字母表中的某个字母转换为大写或小写。

① tolower() 函数将大写字母转换为小写。如果c 已经是小写了，则函数原值返回。

② toupper() 函数将小写字母转换为大写。如果c 已经是大写了，则函数原值返回。

函数具有称为_tolower 和_toupper 的等价宏调用。

例 程

tolower('A' 返回 'a' 若

tolower('十') 返回 '十，

77.va_arg/va_end/va_start函数

语 法

并include<stdarg.h>

typedef char *va_list;

va_arg(ap,type);

void va_end(ap);

void va_start(ap,parmN);

va_list ap

说 明

有些函数调用时使用不同数值不同类型的参数，称为变参数函数。可以使用下面的宏在运行时通过参数表输入参数。参数ap 指向变参数表中的一个参数。

① va_start() 函数将 ap 指针指向变参数表的第1个参数。参数parmN 指向已声明和固定的参数表中最右边的参数。

② va_arg()函数返回参数表中的本次调用的下一个参数值。每次调用va_arg 时，它都修改一次ap，因此连续调用 va_arg 可以实现变参数函数参数的连续变化。参数 type 表示类型名，它是当前表中参数的类型。

③ 在使用了 `va_star` 和 `va_arg` 之后，调用 `va_end()` 函数可以将 `ap` 复位。

例 程

```
int printf(char *fmt,...)
{
    va_list ap;
    va_startCap(fmt);
    :
    /*得到下一个 arg, 一个整型数*/
    i=va_arg(ap,int);
    /* 得到下一个 arg, 一个字符串*/
    s=va_arg(ap,char *);
    /* 得到下一个 arg, 一个长整型数*/
    l=va_arg(ap,long);
    !
    va_end(ap); /* × 复位*/
}
```



参考文献

1. Texas Instruments. TMS320LF/LC240xA DSP Controllers Reference Guide-System and Peripherals SPRU357B. December, 2001
2. Texas Instruments. TMS320LF2407, TMS320LF2406, TMS320LF2402 DSP CONTROLLERS. SPRS094E. August, 2000
3. Texas Instruments. Space-Vector PWM With TMS320C24x/F24x Using Hardware and Software Determined Switching Patterns. SPRA524.
4. Texas Instruments. Code Composer Studio Getting Started Guide. SPRU509C. November, 2001
5. Texas Instruments. TMS320C1x/C2x/C2xx/C5x Assembly Language Tools User's Guide. SPRU018D. March, 1995
6. Texas Instruments. TMS320F/C24x DSP Controllers Reference Guide-CPU and Instruction Set. SPRU160C. June, 1999
7. Texas Instruments. AC Induction Motor Control Using Constant V/Hz Principle and Space Vector PWM Technique with TMS320C240. SPRA284A. April. 1998
8. 三菱公司资料. PM300CLA060 和 PM75RLA120.



北京航空航天大学出版社单片机与嵌入式系统图书推荐

(2005年11月后出版图书)

书 名	作 者	定 价	出 版 日 期
嵌入式系统高端教材			
Nios II 嵌入式软核SOPC设计原理及应用	李兰英	42.0	2006.09
SOPC嵌入式系统基础教程	周立功	29.5	2006.11
SOPC嵌入式系统实验教程(一)	周立功	26.0	2006.11
ARM7嵌入式开发基础实验	刘天时	估30.0	2006.12
ARM7 μClinux开发实验与实践(含光盘)	田 泽	28.0	2006.11
ARM9 嵌 入 式 Linux开 发 实 验 与 实践(含光盘)	田 泽	29.5	2006.11
ARM7嵌入式开发实验与实践(含光盘)	田 泽	29.5	2006.10
ARM9嵌入式开发实验与实践(含光盘)	田 泽	42.0	2006.10
嵌入式系统设计与开发实验——基于XScale平台	石秀民	26.0	2006.10
嵌入式系统基础 μC/OS-II及Linux	任 哲	35.0	2006.08
Windows CE嵌入式系统	何宗键	32.0	2006.08
嵌入式Intel架构微机实验教程(含光盘)		49.0	2006.08
ARM 嵌 入 式 VxWorks实 践 教 程	李忠民	28.0	2006.03
嵌入式系统设计与应用开发	郑灵翔	36.0	2006.02
嵌入式系统设计与实践	张晓林	29.5	2006.01
A R M 嵌 入 式 系 统 实 验 教 程 (三)—扩展实验	周立功	29.5	2006.02
ARM嵌入式系统实验教程(三)	周立功	32.0	2005.11

ARM、SoC 设计、IC 设计及其他嵌入式系统综合类

基于PROTEUS的ARM虚拟开发技术(含光盘)	周润景		2006.12
SRI71x系列ARM微控制器原理与实践	刘建东	42.0	2006.09
C++GUI Qt3编程 (含光盘)	李光峰	49.0	2006.08
嵌入式系统中的模拟设计	李光峰	39.0	2006.07
ARM嵌入式软件开发实例(二)	周立功	55.0	2006.07
ARM9嵌入式Linux系统构建与应用	潘巨龙	29.5	2006.07
ARM SoC设计的软硬件协同验证(含光盘)	周立功	25.0	2006.07
ARM开发工具ADS原理与应用	赵星寒	29.0	2006.06
基于FPGA的可编程SoC设计	董代洁	26.0	2006.06
IAR EWARM嵌入式系统编程与实践 (含光盘)	徐爱钧	49.0	2006.03
英汉双解嵌入式系统词典	马广云等译	59.0	2006.01
嵌入式Ethernet和Internet通信设计技术	骆丽等译	39.0	2006.01
ARM嵌入式Linux系统构建与驱动开发范例	周立功	38.0	2006.02
ARM嵌 入 式 MiniGUI初 步 与 应 用 开 发 范 例	周立功	26.0	2006.01
从51到ARM—32位嵌入式系统入门	赵星寒	38.0	2005.11

书 名	作 者	定 价	出 版 日 期
深入浅出ARM7——LPC213X/214X(下)	周立功	45.0	2006.02

嵌入式操作系统及软件开发

嵌入式可配置实时操作系统eCos软件开发	颜若麟	39.0	2006.06
深入理解Linux虚拟内存管理	白洛等	76.0	2006.06
嵌入式系统接口设计与Linux驱动程序开发	刘 淼	39.0	2006.05
源码开放的嵌入式实时操作系统T-Kernel(含光盘)	周立功等译	45.0	2005.10

DSP

DSP原理及电机控制应用——基于TMS320LF240x系列(含光盘)	刘和平	42.0	2006.11
DSP原理与开发应用	支长义	36.0	2006.08
TMS320X281x DSP原理与应用	徐科军	45.0	2006.08
dsPIC数字信号控制器C程序开发及应用(含光盘)	梁海浪	36.0	2006.06
DSP应用系统设计实践	郑 红	32.0	2006.05

单片机

教材与教辅

高职高专通用教材——凌阳单片机理论与实践	彭传正	估25.0	2006.12
单片机认识与实践	邵贝贝	32.0	2006.08
MC68单片机入门与实践(含光盘)	熊 慧	27.0	2006.08
标准80C51单片机基础教程——原理篇	李学海	29.0	2006.08
高职高专通用教材—单片机原理与应用教程	袁秀英	28.0	2006.08
高职高专通用教材——单片机实训教程	李雅轩	14.0	2006.08
高职高专通用教材——单片机习题与实验教程	李 珍	15.0	2006.08
单片机初级教程——单片机基础(第2版)	张迎新	26.0	2006.09
练中学单片机教程	李 刚	28.0	2006.07
单片机实验与实践教程(一)(第2版)	万光毅	25.0	2006.06
单片机实验与实践教程(二)(第2版)	夏继强	26.0	2006.06
单片机实验与实践教程(三)	周立功	27.0	2006.06
跟我学用单片机(第2版)	肖洪兵	26.0	2006.06
单片机习题与试题解析	高 锋	16.0	2006.03
SoC单片机实验, 实践与应用设计——基于C8051F系列(含光盘)	万光毅	39.5	2006.05
单片机C51程序设计教程与实验	祁 伟	22.0	2006.01

书 名	作 者	定 价	出版 日期
开发实例与实战			
单片机控制实习与专题制作	蔡朝洋	59.0	2006.11
51系列单片机设计实例(第2版)(含光盘)	楼然苗	29.5	2006.02
应用程序设计与开发			
单片机C语言轻松入门(含光盘)	周 坚	29.0	2006.07
单片机智能化产品C语言设计实例详解 (含光盘)	周兴华	28.0	2006.06
单片机C语言和汇编语言混合编程实例详解(含光盘)	杜树青	26.0	2006.06
51系列单片机其他图书			
单片机应用系统电磁干扰与抗干扰技术	王幸之	59.0	2006.01
单片机应用技术选编(11)	何立民	75.0	2006.06
PIC单片机			
PIC单片机初级教程	李荣正	22.0	2006.03
PIC单片机实验教程(含光盘)	李荣正	26.0	2006.02
PIC单片机开发应用与实验工具制作(含光盘)	陈新建	25.0	2006.02
AVR单片机			
AVR单片机GCC程序设计	佟长福	28.0	2006.01
其他公司单片机			
凌阳16位单片机C语言开发(含光盘)	李晓白	35.0	2006.09
16位单片机原理与应用	彭宣戈	25.0	2006.09
凌阳16位单片机开发实例	凌阳科技	19.5	2006.06
凌阳8位通用单片机原理及开发	凌阳科技	28.0	2006.04
凌阳8位单片机——提高篇(含光盘)	李学海	45.0	2006.03
MCU-DSP混合控制器技术与应用——基于凌阳16位单片机	刘海成	26.0	2006.01
凌阳8位单片机——基础篇	李学海	38.0	2005.11
总线技术			
8051单片机USB接口VB程序设计	许永和	估45.0	2006.12
EZ-USB FX2单片机原理、编程及应用	钱 峰	45.0	2006.03

书 名	作 者	定 价	出版 日期
其 它			
无线单片机技术丛书之一——短距离无线数据通信入门与实战(含光盘)	李文仲	25.0	2006.12
无线单片机技术丛书之二——51单片机与短距离无线通信	李文仲	26.0	2006.12
智能技术——系统设计与开发	张洪润	49.0	2006.12
自动控制原理考研试题分析与解答技巧	张苏英	22.0	2006.11
全国大学生电子设计竞赛电路设计	黄智伟	估32.0	2006.11
全国大学生电子设计竞赛系统设计	黄智伟	估31.0	2006.11
计算机硬件类课程设计难点辅导	张 瑜	25.0	2006.08
单片机应用设计200例(上册)	张洪润	60.0	2006.07
单片机应用设计200例(下册)	张洪润	55.0	2006.07
传感器应用电路200例	张洪润	86.0	2006.07
基于Protens的单片机可视化软硬件仿真(含光盘)	林志琦	25.0	2006.09
Verilog FPGA芯片设计	林灶生	35.0	2006.07
Cadence高速PCB设计与仿真分析(含光盘)	黄豪佑	46.0	2006.07
基于PROTEUS的电路与单片机系统设计与仿真(含光盘)	周润景	45.0	2006.06
蓝牙技术原理, 开发与应用	钱志鸿	35.0	2006.03
蓝牙硬件电路	黄智伟	65.0	2005.08
无线通信集成电路	黄智伟	96.0	2005.07
2006年上海市嵌入式系统创新设计竞赛获奖作品论文集	竞赛评审委员会	27.0	2006.10
第四届全国高校嵌入式系统教学研讨会 2006第一届全国嵌入式系统学术交流会		50.0	2006.07
2005年ARM应用技术论文大奖赛论文集	组委会	29.0	2006.05
2005年上海市高校学生嵌入式系统创新设计竞赛获奖作品论文集	竞赛评审委员会	26.0	2005.1
中国西部嵌入式系统与单片机技术论坛2005年学术年会论文集	四川省单片机学会	50.0	2005.12
《单片机与嵌入式系统应用》杂志合订本(2005年1~12月)		65.0	2006.06

注：表中加底纹者为2006年7月后出版的图书。

可直接向出版社邮购以上图书。邮购通信及汇款地址：北京航空航天大学出版社邮购组(邮编100083) 另加3元挂号费。

邮购组电话：010-82316936. 传真：010-82317031 详细图书目录及内容介绍请查阅出版社网站：<http://www.bumapress.com> cn

投稿单片机与嵌入式系统图书请联系：通信：北京航空航天大学出版社第上编辑室(邮缩(100083))

电话：010-8231702282317035 82317044 传真：010-82317022 Email:pressb@nesnet.com.cn