MSP430 系列

单片机接口技术及系统设计实例

魏小龙 编著

北京航空航天大学出版社
http://www.buaapress.com.cn

TI 公司 MSP430 系列单片机丛书

- MSP430 系列超低功耗 16 位单片机原理与应用　胡大可 主编
- MSP430 系列 FLASH 型超低功耗 16 位单片机　胡大可 编著
- MSP430 系列单片机接口技术及系统设计实例　魏小龙 编著
- MSP430 系列单片机 C 语言程序设计与开发　胡大可 编著

ISBN 7-81077-231-7

9 787810 772310 >

ISBN 7-81077-231-7
定价：45.00元

内容简介

本书较为详细地介绍了TI公司的MSP430系列单片机，全书共分5章。首先讲述MSP430全系列所涉及的片内外围模块的功能、原理、应用及各个系列的模块构成情况；其次讲述MSP430的开发环境及如何使用汇编语言、C语言进行程序设计；最后例举大量的以MSP430为核心的系统设计应用实例，大部分实例同时给出汇编语言和C语言的源程序，且全部在作者设计的实验工具上测试通过，并实现了预期的功能。

本书配光盘一张，包含书中所用到的源程序及TI公司的网址与笔者的网址连接，还有MSP430软件的下载地址。

本书可作为高等院校计算机、自动化及电子技术类专业的教学参考书，也可作为大学生电子设计以及毕业设计的参考书，更适用于从事单片机开发的科技人员。

图书在版编目(CIP)数据

MSP430系列单片机接口技术及系统设计实例/魏小龙编著.
—北京：北京航空航天大学出版社，2002.11
ISBN 7-81077-231-7

Ⅰ.M… Ⅱ.魏… Ⅲ.①单片微型计算机，MSP430-接口
②单片微型计算机，MSP430-系统设计 Ⅳ.TP368.1

中国版本图书馆CIP数据核字(2002)第082109号

MSP430系列单片机接口技术及系统设计实例

魏小龙 编著

责任编辑 王 实

*

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:pressell@publica.bj.cninfo.net

河北省涿州市新华印刷厂印装 各地书店经销

*

开本：787×1 092 1/16 印张：27 字数：691千字

2002年11月第1版 2002年11月第1次印刷 印数：5 000册

ISBN 7-81077-231-7 定价：45.00元

MSP430 系列
单片机接口技术及系统设计实例

魏小龙　编著

北京航空航天大学出版社
http://www.buaapress.com.cn

序

1999 年夏天，TI(香港)有限公司的 HPA 的市场部经理陈维哲先生找到利尔达单片机技术有限公司，向我们介绍了 MSP430 单片机的情况。当时的 MSP430 还没有 FLASH，只有 C11、X31、X32、X33 系列。陈先生还向我们介绍了 X31、X32 系列 1996 年在欧美市场的销售情况和应用领域。虽然当时 MSP430 的主要特点是低功耗，但我们认为它是一个很不错的产品，因此我们还是愿意与 TI 公司合作，成为其在中国第一个专门针对 MSP430 这一产品的增值经销商。在 1999 年秋季单片机与嵌入式系统研讨会上，TI 公司和我们一起在杭州主办了学术研讨会。很多的客户对 X32 系列非常感兴趣。清华同方、深开发及青岛海信等公司开始选择使用 MSP430X32 设计产品。

TI 公司分别于 2000 年初推出了 MSP430F11X，13X，14X；2001 年推出了 F41X；2002 年推出了 MSP430F43X，F44X；计划在 2003 年推出 F45X，F46X，F5XX 系列。2001 年 MSP430 在中国的销售额从 1999 年的 50 万美元增加到 150 万美元，预计 2002 年将会达到 300 万美元。应该说，MSP430 在中国市场的销路基本打开了。

众所周知，MSP430 系列包容了 MCU 在国际上的先进技术 JIAG 和 FLASH 在线编程技术。2001 年 TI 公布了 BOOTSTRAP 技术，利用它可在烧断熔丝以后只要几根线就可更改并运行内部的程序。这为系统软件的升级提供了又一种方便的手段。BOOTSTRAP 具有很高的保密性，口令可达到 32 个字节的长度。2002 年初推出的 MSP430F43X，F44X 在 13X，14X 的基础上可驱动 160 段 LCD，电压范围为 1.8～3.6 V。

任何一个外资公司，当它的产品进入中国市场时，要使所有中国的电子工程师更深入地了解其产品，首先要有中文资料。利尔达的应用工程师和浙江大学等高校的教授合作，翻译和编写了部分中文资料和数据手册等。其中有广大读者熟悉的《MSP430 系列超低功耗 16 位单片机原理和应用》和《MSP430 系列 FLASH 型超低功耗 16 位单片机》，还有一些中文资料已放到网上。利尔达还找了南京航空航天大学的魏小龙老师，请他编写了这本《MSP430 系列单片机接口技术及系统设计实例》。魏老师可以说是 MSP430 的一个资深发烧友，自己建立了网站，自己掏钱购买工具研制了国内最早的 JIAG 仿真器，组织了 MSP430 单片机爱好者沙龙，并举办了讲座和培训班。该书稿写完以后，魏老师将初稿发给我，我和我的同事季燕飞、平红光、梁源等认真地阅读了此书稿。这本书详细讲解了 MSP430 全系列的接口原理、设计开发方法以及大量的系统设计实例。这些设计案例都在作者设计的实验板或面包板调试通过，并实现了预期的功能。

杭州利尔达单片机技术有限公司愿意为广大的读者提供 MSP430 方面的技术帮助和支持。如果读者对 MSP430 感兴趣请与我们联系。

杭州利尔达单片机技术有限公司副总经理

段焕春

2002 年 9 月 12 日

利尔达单片机技术有限公司联系方式

杭州公司

地址：杭州市教工路 18 号电子市场 291 室

电话：0571—88800000　88886195

传真：0571—88805970　88053601

邮编：310012

E－mail：lierda@mail.hz.zj.cn

北京办事处

地址：北京市海淀区知春路 132 号中发电子大厦 808 室

电话：010—82622345　62576660

传真：010—82622627

邮编：100086

E－mail：beijing@lierda.com

上海办事处

地址：上海市漕溪路 165 号华谊大楼 1503 室

电话：021—54591878

传真：021—54240654

邮编：200233

E－mail：shanghai@lierda.com

深圳办事处

地址：深圳上步南路国企大厦永辉楼 10F 室

电话：0755—25880248

传真：0755—82129206

邮编：518031

E－mail：shenzhen@lierda.com

青岛办事处

地址：青岛市南区燕儿岛路 18 号鹏程花园 C 座 1807 室

电话：0532—5785216

传真：0532—5785216

邮编：266071

E－mail：qingdao@lierda.com

前　言

单片机(或微控制器)技术已经渗透到人类生活的方方面面,在家用电器、通信产品等日用电子设备中都可见到单片机.估计全世界人均有几片单片机(或微控制器),此技术正在积极地影响着人类的生活。

TI 公司 MSP430 系列是一个超低功耗类型的单片机,特别适合于电池应用的场合或手持设备。同时,该系列将大量的外围模块整合到片内,也特别适合于设计片上系统;有丰富的不同型号的器件可供选择,给设计者带来很大的灵活性。它是一个 16 位的精简指令构架,有大量的工作寄存器和数据存储器(目前最大的 RAM 为 2 KB),其 RAM 单元也可以实现运算。MSP430 系列是众多单片机系列中的一颗耀眼的新星。

在超低功耗方面,MSP430 系列单片机能够实现在 1.8～3.6 V 电压、1 MHz 的时钟条件下运行,耗电电流(在 0.1～400 μA 之间)因不同的工作模式而不同;同时能够在实现液晶显示的情况下,只耗电 0.8 μA。典型情况:在 4 kHz, 2.2 V 条件下工作消耗电流 2.5 μA;在 1 MHz, 2.2 V 条件下工作消耗电流 280 μA;在只有 RAM 数据保持的低功耗模式下耗电 0.1 μA(见 MSP430X13X、14X 数据手册)。

在运算速度方面,MSP430 系列单片机能在 8 MHz 晶体的驱动下,实现 125 ns 的指令周期。16 位的数据宽度、125 ns 的指令周期以及多功能的硬件乘法器(能实现乘加)相配合,能实现数字信号处理的某些算法(如 FFT 等)。

在整合方面,MSP430 系列单片机将大量的 CPU 外围模块集成在了片内,有如下一些模块:

看门狗(WDT)	定时器 A (Timer_A)	定时器 B (Timer_B)
模拟比较器	串口 0、1(USART0、1)	硬件乘法器
液晶驱动器	10 位/12 位 ADC	14 位 ADC(ADC14)
端口 0(P0)	端口 1～6(P1～P6)	基本定时器(Basic Timer)

其中:定时器 A、B 均带有多个捕获/比较寄存器,同时可实现多路 PWM 输出;模拟比较器与定时器配合,可方便地实现 ADC;液晶驱动多达 160 笔段;硬件 ADC 模块能在小于 10 μs 的速率下实现 10～14 位的高速、高精度转换,同时提供采样/保持与参考电压;端口 0,1,2(P0,P1,P2)能够接收外部上升沿或下降沿的中断输入。

我在 1998 年底开始接触 MSP430 单片机,那时只有网络上的大量英文资料,通过阅读发现其优点,便开始细读。1999 年,我申请到 MSP430F1121 样片,进行了一些实验,更加证实了 MSP430 的诸多优点。最初我使用 MSP430 实现了热敏电阻测量温度,这是 TI 网站提供的典型案例,但是 TI 文档中的程序格式在 TI 网站提供的 IAR 软件中却不能编译,我非常熟悉 51 却也对其中许多硬件的操作不是很理解。然而 MSP430 的确有很多优点:可以方便地调整器件的运行耗电量;硬件功能齐备,如定时器 A(当时的 F1121 就有)带有 3 个捕获/比较器,可以实现 3 路时序,也可实现 PWM 输出等。鉴于此,为了让更多想使用 MSP430 的朋友少走弯路,我便使用网络(笔者的个人主页 http://ppowxl.top263.net,现改为 http://www.mcu-china.com)发布了本人的使用心得和自己写的一些源程序。我的个人主页在 263 网站停止其

使用之前的访问量达 60 000 多 IP 地址，这个统计数据是 263 提供的，非常可靠。通过网络我认识了很多使用 MSP430 的朋友，相互切磋，受益颇多，我的很多心得体会在书中都有体现。后来笔者陆续使用 MSP430 做了一些开发，比如电子水表、热表、黑匣子记录仪、智能传感器、电站使用的操作票掌上机、一些电池供电的医用仪表等。这些开发工作的进行使得我对 MSP430 更加熟悉。由于很多网友对 MSP430 一些问题的询问，以及我对 MSP430 的使用已经积累了不少心得体会，在 2001 年我萌生了编写 MSP430 接口与应用方面的书的念头，后与北京航空航天大学出版社联系，得到很大支持。近一年半的时间，我在大量的实验基础上，总结了与 MSP430 有关的多次开发经历，于 2002 年暑假完成了书稿。近来 TI 发布的所有 FLASH 型芯片，我都使用其做过实验，并体现于本书中。

本书讲解了 MSP430 全系列的接口原理、设计开发方法以及大量的系统设计实例。本书共分为 5 章：第 1 章讲述了 MSP430 的大致情况，对各个系列(11X，12X，13X，14X，31X，32X，33X，41X，43X，44X)分门别类地进行了介绍，可为设计者选型提供参考；第 2 章详细地讲述了指令系统及汇编语言和 C 语言的设计方法；第 3 章讲述了 MSP430 全系列所涉及到的所有片内外围模块(目前器件)的接口原理和使用方法，从最早的 3XX 系列到最新的 4XX 系列所涉及到的 MSP430 单元模块，其中大部分都结合接口原理的讲解给出了实际的应用；第 4 章介绍了开发环境，详细地讲述了开发软件的使用方法；第 5 章给出了大量的应用实例，由简单程序设计和硬件应用到较为完整的系统设计，每个例子都有详细的设计原理、汇编和(或)C 语言源程序，读者可直接借鉴；最后为附录，由于 MSP430 单片机的片内外围设备都拥有大量的寄存器，因此在附录中将这些寄存器集中起来以方便读者查阅。

在这里，要着重强调的是：本书中的源程序以及第 5 章的设计实例都是笔者亲自编写并调试通过的，尤其是第 5 章的实例都在笔者设计的实验板或面包板上调试通过，并实现了预期的功能，读者可放心借鉴。相信对有无单片机设计经历的读者都会有帮助。

本书配光盘一张，包含书中所用到的源程序及 TI 公司的网址与笔者的网址连接，还有 MSP430 软件的下载地址。

本书在成书过程中得到了 TI 公司 MSP430 中国代理利尔达单片机技术有限公司的段焕春副总经理和平红光经理的大力支持，在此表示衷心的感谢。

限于笔者水平，书中错误与不妥在所难免，恳请读者批评指正。同时欢迎访问我的个人主页(www. mcu－china. com)，以相互交流。

作　者

2002 年 6 月于南京航空航天大学

目　录

门狗可以使程序失控时迅速复位；比较器进行模拟电压的比较，配合定时器可以设计为 A/D 转换器；定时器具有捕获/比较功能、可用于事件计数、时序发生、PWM 等；有的器件更具有两个串口，可方便地实现多机通信等应用；具有较多的并行端口，最多达 6×8 条 I/O 口线，而且 I/O 口线具有中断能力；12/14 位硬件 A/D 转换器有较高的转换速率，最高可达 200 kbps，能满足大多数数据采集应用；能直接驱动液晶多达 120 段。MSP430 系列单片机的这些片内外设为系统的单片解决方案提供了极大的方便。

● **方便高效的开发环境**

目前 MSP430 系列有 4 种类型器件：OTP 型、FLASH 型、EPROM 型和 ROM 型。这些器件的开发手段不同。对于 OTP 型和 ROM 型的器件是用相对应的 EPROM 型器件作为开发片，或使用仿真器开发成功之后再烧写或掩膜芯片；而对于 FLASH 型则有十分方便的开发调试环境，因为器件片内有 JTAG 调试接口，还有可电擦写的 FLASH 存储器，因此采用先下载程序到 FLASH 内，再在器件内通过软件控制程序的运行，由 JTAG 接口读取片内信息供设计者调试使用的方法进行开发。这种方式只需要一台 PC 机和一个 JTAG 调试器，而不需要仿真器和编程器。开发语言有汇编语言和 C 语言。

● **工业级的产品**

MSP430 系列器件均为工业级的，运行环境温度为 - 40～+85 ℃。

下面简要介绍 MSP430 家族不同系列的情况，而目前的所有器件都列举在表 1.1 中。

表 1.1 MSP430 器件性能表

器件名称	程序存储器容量/KB	数据存储器容量/B	A/D 转换器类型	液晶驱动能力/段	捕获/比较功能的有无	串口类型	硬件乘法器的有无	定时器的数量	I/O 端口的数量
MSP430C1111	2	128	slope		Yes	软件方式	No	2	8+6 I/O
MSP430C1121	4	256	slope		Yes	软件方式	No	2	8+6 I/O
MSP430C1331	8	256	slope		Yes	硬件方式	No	3	6×8 I/O
MSP430C1351	16	512	slope		Yes	硬件方式	No	3	6×8 I/O
MSP430C311	2	128	slope	64	No	软件方式	No	6	1×8 I/O+30 O
MSP430C312	4	256	slope	92	No	软件方式	No	6	1×8 I/O+30 O
MSP430C313	8	256	slope	92	No	软件方式	No	6	1×8 I/O+30 O
MSP430C314	12	512	slope	92	No	软件方式	No	6	1×8 I/O+30 O
MSP430C315	16	512	slope	92	No	软件方式	No	6	1×8I/O+30 O
MSP430C323	8	256	14bit	84	No	软件方式	No	6	6 三态+8 I/O+24 O
MSP430C325	16	512	14bit	84	No	软件方式	No	6	6 三态+8 I/O+24 O
MSP430C336	24	1 024	slope	120	Yes	硬件方式	Yes	7	6 三态+5×8 I/O+34 O
MSP430C337	32	1 024	slope	120	Yes	硬件方式	Yes	7	6 三态+5×8 I/O+34 O
MSP430C412	4	256	slope	96	Yes	软件方式	No	3	6×8 I/O
MSP430C413	8	256	slope	96	Yes	软件方式	No	3	6×8 I/O
MSP430F110	1	128	slope		Yes	软件方式	No	1	8+6 I/O
MSP430F1101	1	128	slope		Yes	软件方式	No	1	8+6 I/O

续表 1.1

器件名称	程序存储器容量/KB	数据存储器容量/B	A/D 转换器类型	液晶驱动能力/段	捕获/比较功能的有无	串口类型	硬件乘法器的有无	定时器的数量	I/O 端口的数量
MSP430F1101A	1	128	slope		Yes	软件方式	No	2	8+6 I/O
MSP430F1111A	2	128	slope		Yes	软件方式	No	2	8+6 I/O
MSP430F112	4	256	slope		Yes	软件方式	No	1	8+6 I/O
MSP430F1121	4	256	slope		Yes	软件方式	No	1	8+6 I/O
MSP430F1121A	1	256	slope		Yes	软件方式	No	2	8+6 I/O
MSP430F122	4	256	ADC10		Yes	硬件方式	No	2	6+8+8 I/O
MSP430F123	8	256	ADC10		Yes	硬件方式	No	2	6+8+8 I/O
MSP430F133	8	256	12 bit		Yes	硬件方式	No	3	6×8 I/O
MSP430F135	16	512	12 bit		Yes	硬件方式	No	3	6×8 I/O
MSP430F147	32	1 024	12 bit		Yes	硬件方式	Yes	3	6×8 I/O
MSP430F148	48	2 048	12 bit		Yes	硬件方式	Yes	3	6×8 I/O
MSP430F149	60	2 048	12 bit		Yes	硬件方式	Yes	3	6×8 I/O
MSP430F412	4	256	slope	96	Yes	软件方式	No	3	6× I/O
MSP430F413	8	256	slope	96	Yes	软件方式	No	3	6×8 I/O
MSP430F435	16	512	ADC12	160	Yes	硬件方式	No	5	6×8 I/O + 44 O
MSP430F436	24	1 024	ADC12	160	Yes	硬件方式	No	5	6×8 I/O + 44 O
MSP430F437	32	1 024	ADC12	160	Yes	硬件方式	No	5	6×8 I/O + 44 O
MSP430F447	32	1 024	ADC12	160	Yes	硬件方式	Yes	5	6×8 I/O + 44 O
MSP430F448	48	2 048	ADC12	160	Yes	硬件方式	Yes	5	6×8 I/O + 44 O
MSP430F449	60	2 048	ADC12	160	Yes	硬件方式	Yes	5	6×8 I/O + 44 O
MSP430P112	4	256	slope		Yes	软件方式	No	2	6+8 I/O
MSP430P315	16	512	slope	92	No	软件方式	No	6	1×8 I/O+30 O
MSP430P325	16	512	14bit	84	No	软件方式	No	6	6 三态+8 I/O+24 O
MSP430P325A	16	512	14bit	81	No	软件方式	No	6	6 三态+8 I/O+24 O
MSP430P337	32	1 024	slope	120	Yes	硬件方式	Yes	7	6 三态+5×8 I/O+34 O
MSP430P337A	32	1 024	slope	120	Yes	硬件方式	Yes	7	6 三态+5×8 I/O+34 O

注：

1. slope —斜边 ADC。
2. 软件方式——用软件的方法实现串口通信。
3. 硬件方式—— 用硬件的方法实现串口通信。

以上这些器件可以大致分为这样一些系列(后面还要细化)：

MSP430X1XX 系列；

MSP430X3XX 系列；

MSP430X4XX 系列。

这些系列有相同的命名规则，以 MSP430P325IPM 为例，其命名规则如图 1.1 所示。

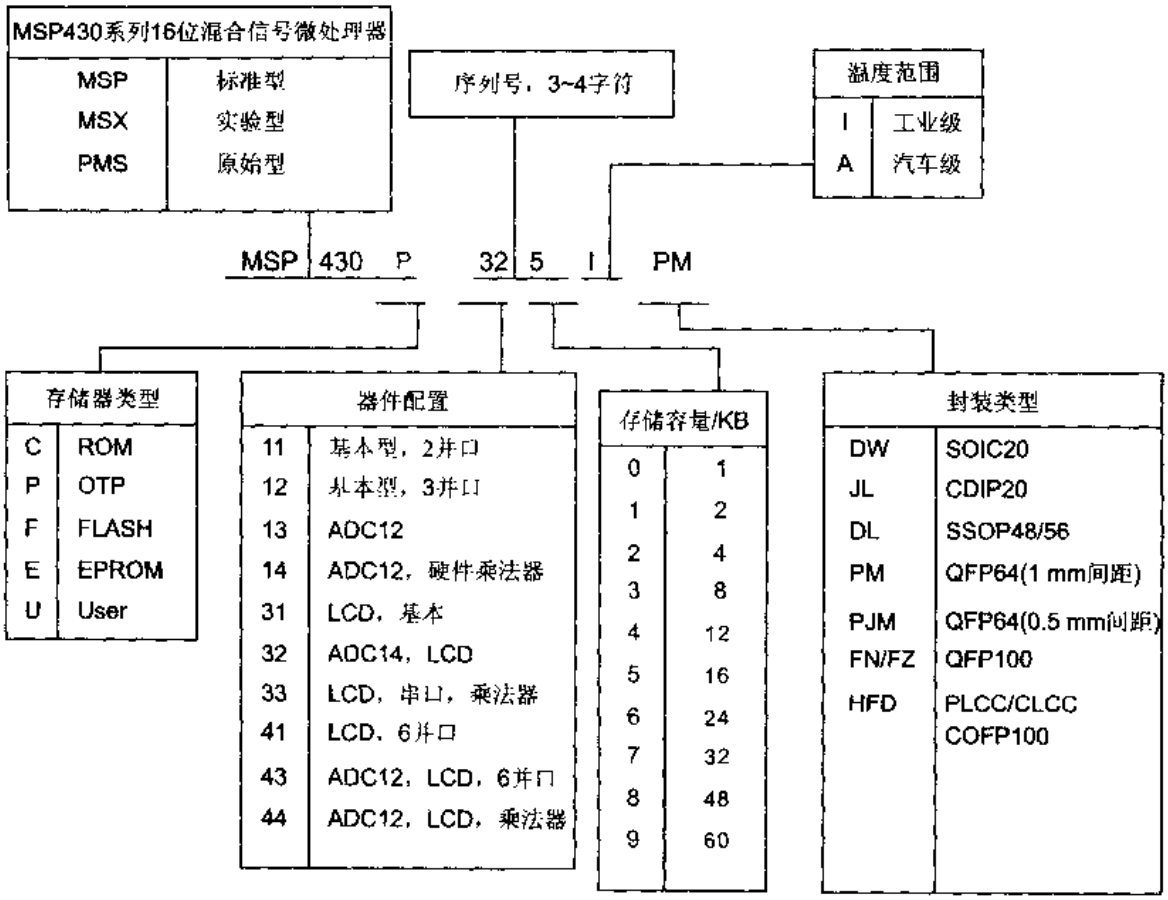


图 1.1 MSP430 的命名规则

1.2 MSP430X1XX 系列

MSP430X1XX 系列为目前品种最多的一个系列，有以下一些器件：

MSP430C1111 MSP430C1121
MSP430P112
MSP430F110 MSP430F1101 MSP430F1101A MSP430F1111A
MSP430F112 MSP430F1121 MSP430F1121A
MSP430F122 MSP430F123
MSP430F133 MSP430F135
MSP430F147 MSP430F148 MSP430F149

这些器件又可以细分为：MSP430X11X 系列；MSP430X12X 系列；MSP430X13X 系列；MSP430X14X 系列。其中，MSP430CXXXX 表示该器件为 ROM 型；MSP430PXXXX 表示该器件为 OTP 型（一次性可编程器件）；MSP430FXXXX 表示该器件为 FLASH 型（可以反复电擦除/编程的 FLASH 型）；MSP430FXXX2 表示该器件为 MTP 型（可以多次编程）。而 MSP430X11X 系列、MSP430X12X 系列、MSP430X13X 系列、MSP430X14X 系列内部构件各不一样，从它们的内部结构框图可以看出这一点。

1.2.1　MSP430X11X 系列

1. 特　点

- 低电源电压范围：1.8～3.6 V。
- 超低功耗：

  6 μA　　@ 4 kHz，2.2 V；

  250 μA　@ 1 MHz，2.2 V。
- 5 种节电模式：

  等待方式　0.8 μA；

  RAM 保持的节电方式　0.1 μA。
- 从等待方式唤醒时间：6 μs。
- 16 位 RISC 结构，150 ns 指令周期。
- 基本时钟模块配置：

  多个内部电阻，一个外部电阻；

  32 kHz 晶体；

  高频率晶体；

  谐振器；

  外部时钟源。
- 配合外部器件可构成单斜边 A/D 转换器。
- MSP430F1132 内有 10 位 200 kbps 的 A/D 转换器，自带采样保持。
- 具有 3 个捕获/比较寄存器的 16 位定时器，
- 安全熔丝的程序代码保护。
- 该家族品种繁多：

  ROM 型的 MSP430C1111，MSP430C1121；

  可一次编程的 MSP430P112；

  FLASH 型的 MSP430F110，MSP430F1101，MSP430F1101A，MSP430F1111A，MSP430F112，MSP430F1121，MSP430F1121A，MSP430F1132；

  EPROM 型的 MSP430E1121。

2. 结构框图

本章的结构框图摘自 TI 的数据手册

该系列结构框图如图 1.2 所示，其中 MSP430F1132 的结构框图如图 1.3 所示。

3. 引脚图及说明

MSP430X11X 系列引脚图如图 1.4 所示。

MSP430X11X 系列单片机引脚说明如表 1.2 所列。

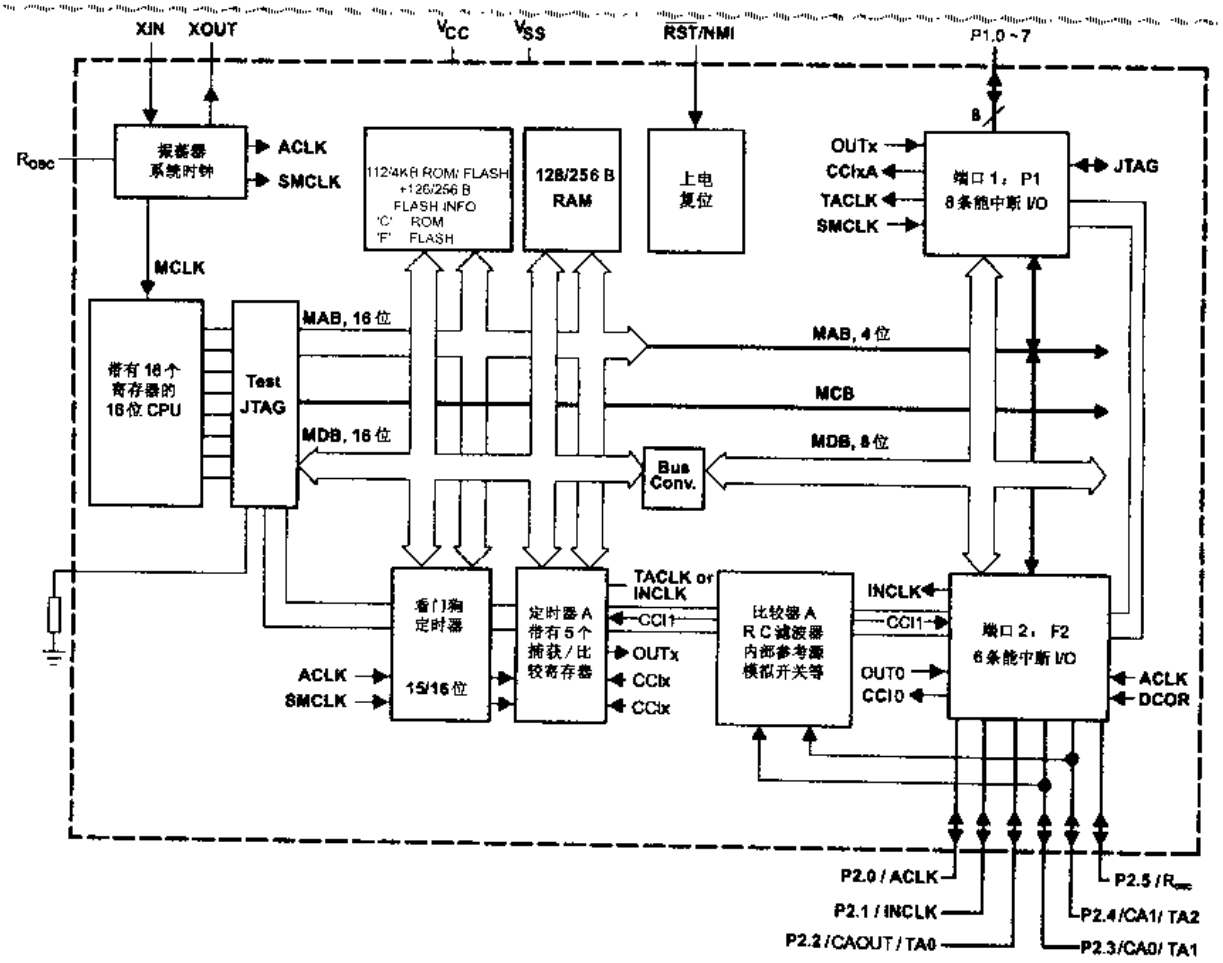


图 1.2 MSP430X11X 系列结构框图

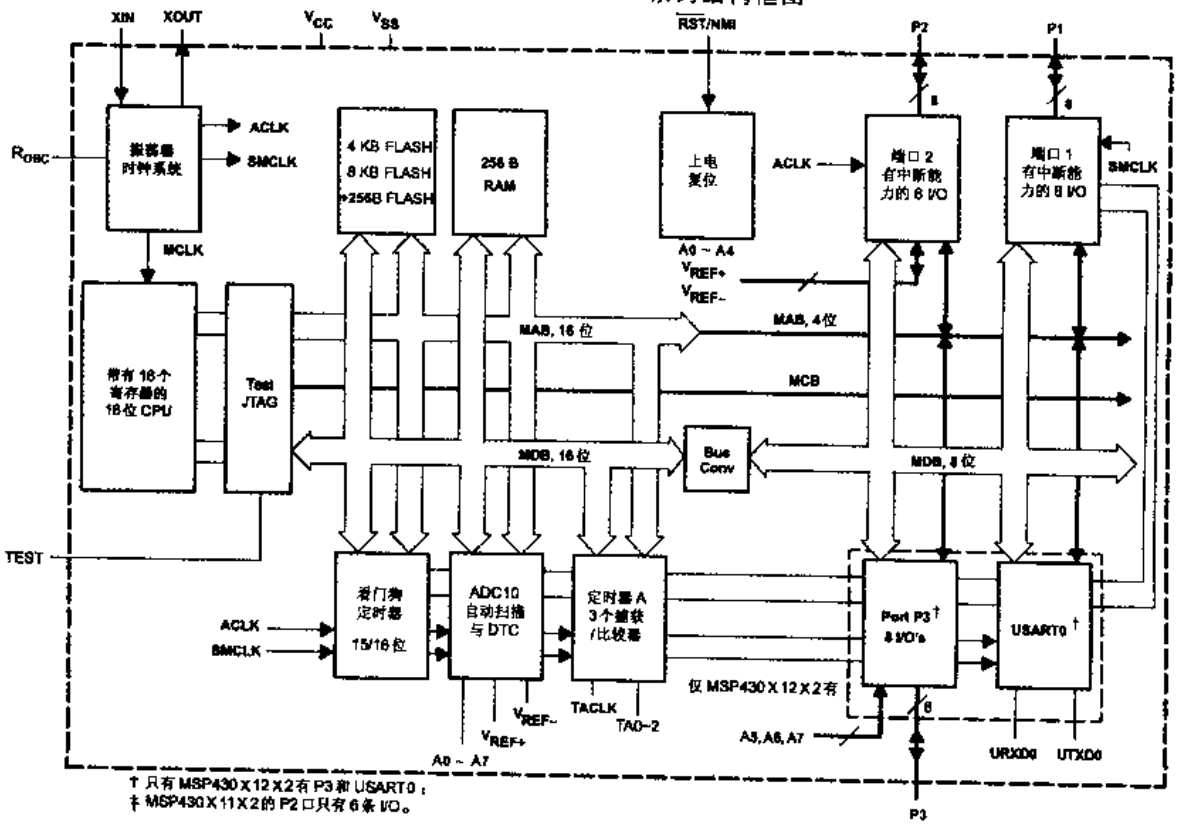


图 1.3 MSP430F1132 结构框图

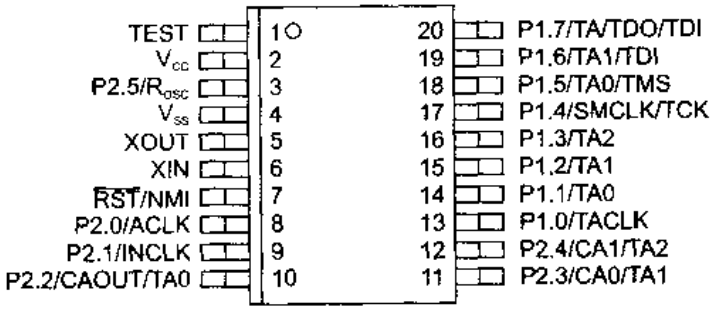


图 1.4　MSP430X11X 系列引脚图

表 1.2　MSP430X11X 系列单片机引脚说明

引脚名称	引脚序号	I/O	说明
P1.0/TACLK	13	I/O	通用数字 I/O 引脚/Timer_A，TACLK 时钟信号输入
P1.1/TA0	14	I/O	通用数字 I/O 引脚/Timer_A，捕获：CCI0A 输入，比较：OUT0 输出
P1.2/TA1	15	I/O	通用数字 I/O 引脚/Timer_A，捕获：CCI1A 输入，比较：OUT1 输出
P1.3/TA2	16	I/O	通用数字 I/O 引脚/Timer_A，捕获：CCI2A 输入，比较：OUT2 输出
P1.4/SMCLK/TCK	17	I/O	通用数字 I/O 引脚，SMCLK 信号输出/测试时钟，用于器件编程和测试的时钟输入端
P1.5/TA0/TMS	18	I/O	通用数字 I/O 引脚/Timer_A，比较：OUT0 输出/测试方式选择，器件编程和测试输入端
P1.6/TA1/TDI	19	I/O	通用数字 I/O 引脚/Timer_A，比较：OUT1 输出/测试数据输入端
P1.7/TA2/TDO/TDI	20	I/O	通用数字 I/O 引脚/Timer_A，比较：OUT2 输出/测试数据输出端/编程时数据输入端
P2.0/ACLK	8	I/O	通用数字 I/O 引脚，ACLK 输出端
P2.1/INCLK	9	I/O	通用数字 I/O 引脚，Timer_A，INCLK 时钟信号
P2.2/CAOUT/TA0	10	I/O	通用数字 I/O 引脚/Timer_A，捕获：CCI0B 输入，比较：OUT0 输出
P2.3/CA0/TA1	11	I/O	通用数字 I/O 引脚/Timer_A，捕获：CCI1B 输入，比较：OUT1 输出
P2.4/CA1/TA2	12	I/O	通用数字 I/O 引脚/Timer_A，比较：OUT2 输出
P2.5/R_{osc}	3	I/O	通用数字 I/O 引脚/外接一电阻用以确定 DCO 的工作频率
RST/NMI	7	I	复位信号输入/不可屏蔽中断输入
TEST	1	I	用于端口 1 的 JTAG 引脚的测试方式选择
V_{CC}	2		电源引入端
V_{SS}	4		电源地
XIN	6	I	晶体振荡器连接端
XOUT/TCLK	5	I/O	晶体振荡器输出端/测试时钟输入端

1.2.2　MSP430X12X 系列

MSP430X12X 系列器件主要有 MSP430F122，MSP430F123。

1. 特　点

- 低电源电压范围：1.8～3.6 V。
- 超低功耗：

　　6 μA　　@ 4 kHz，　2.2 V；

200 μA @ 1 MHz, 2.2 V。

- 5 种节电模式：

  等待方式 0.7 μA；

  RAM 保持的节电方式 0.1 μA。

- 从等待方式唤醒时间：6 μs。
- 16 位 RISC 结构，150 ns 指令周期。
- 基本时钟模块配置：

  多个内部电阻，一个外部电阻；

  32 kHz 晶体；

  高频率晶体；

  谐振器；

  外部时钟源。

- 配合外部器件可构成单斜边 A/D 转换器。
- 具有 3 个捕获/比较寄存器的 16 位定时器。
- 串行通信接口可用于异步或同步(UART / SPI)。
- 与 MSP430X11X 相比，有 3 个并行端口。
- 串行在系统编程。
- 安全熔丝的程序代码保护。

2. 结构框图

该系列结构框图如图 1.5 所示。

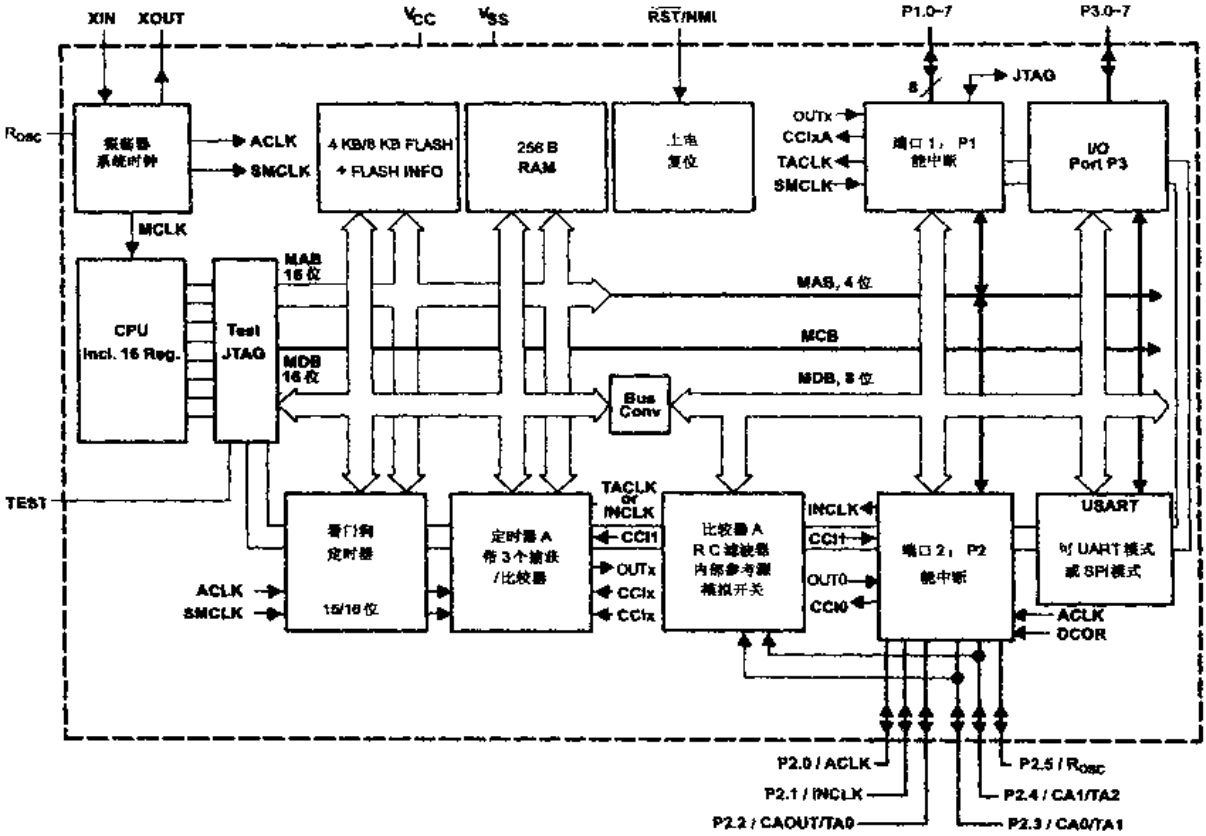


图 1.5 MSP430X12X 系列结构框图

续表 1.3

引脚名称	引脚序号	I/O	说明
P3.7	18	I/O	通用数字 I/O 引脚
RST/NMI	7	I	复位信号输入 不可屏蔽中断输入
TEST	1	I	用于端口 1 的 JTAG 引脚的测试方式选择
V_{CC}	2		电源引入端
V_{SS}	4		电源地
XIN	6	I	晶体振荡器连接端
XOUT TCLK	5	I/O	晶体振荡器输出端 / 测试时钟输入端

1.2.3 MSP430X13X 系列

MSP430X13X 系列器件主要有 MSP430F133、MSP430F135。

1. 特 点

- 低电源电压范围:1.8～3.6 V。
- 超低功耗:
  2.5 μA @ 4 kHz、2.2 V;
  160 μA @ 1 MHz、2.2 V。
- 5 种节电模式:
  等待方式 0.7 μA;
  RAM 保持的节电方式 0.1 μA。
- 从等待方式唤醒时间:6 μs。
- 16 位 RISC 结构,150 ns 指令周期。
- 基本时钟模块配置:
  高速晶体(最高 8 MHz);
  低速晶体(32 768 Hz);
  DCO。
- 配合外部器件可构成单斜边 A/D 转换器。
- 12 位 200 kbps 的 A/D 转换器,自带采样保持,多种转换方式。
- 内部温度传感器。
- 具有 3 个捕获/比较寄存器的 16 位定时器 Timer_A,Timer_B。
- 串行通信接口可用于异步或同步(软件选择 UART / SPI 模式)。
- 6 个 8 位并行端口,且 2 个 8 位端口有中断能力。
- 片内 FLASH 存储器,方便开发与调试。
- JTAG 引脚单独引出,不与 I/O 口线复用。
- 串行在系统编程。
- 安全熔丝的程序代码保护。

2. 结构框图

该系列结构框图如图 1.7 所示。

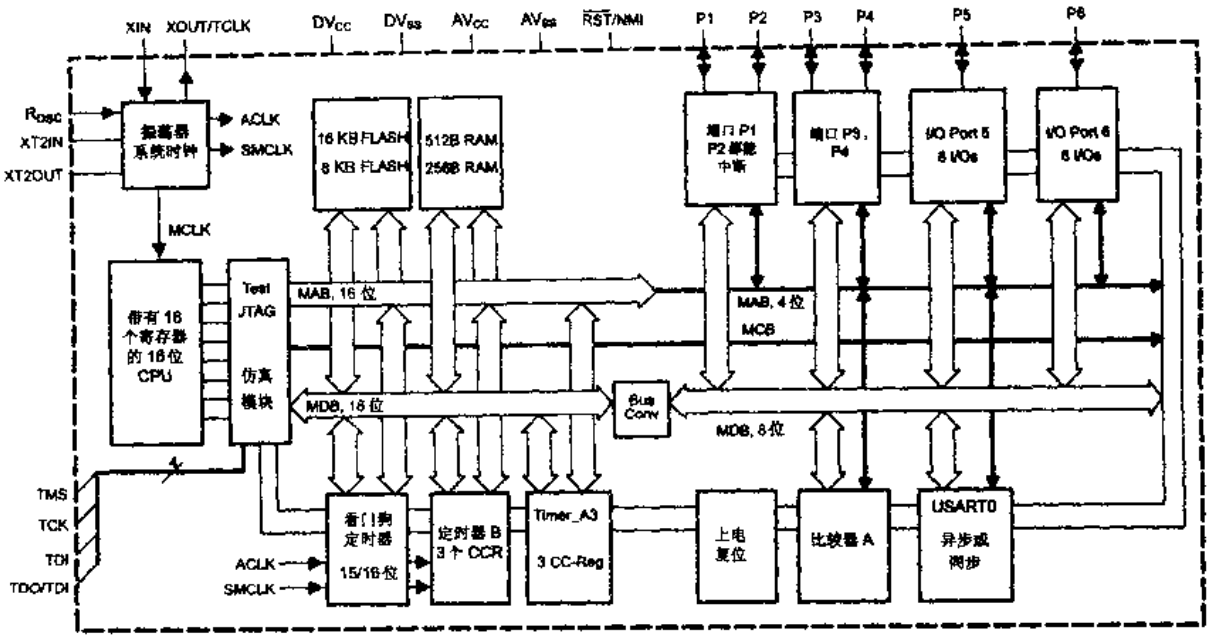


图 1.7　MSP430X13X 系列结构框图

3. 引脚图及说明

MSP430X13X 系列引脚图如图 1.8 所示。

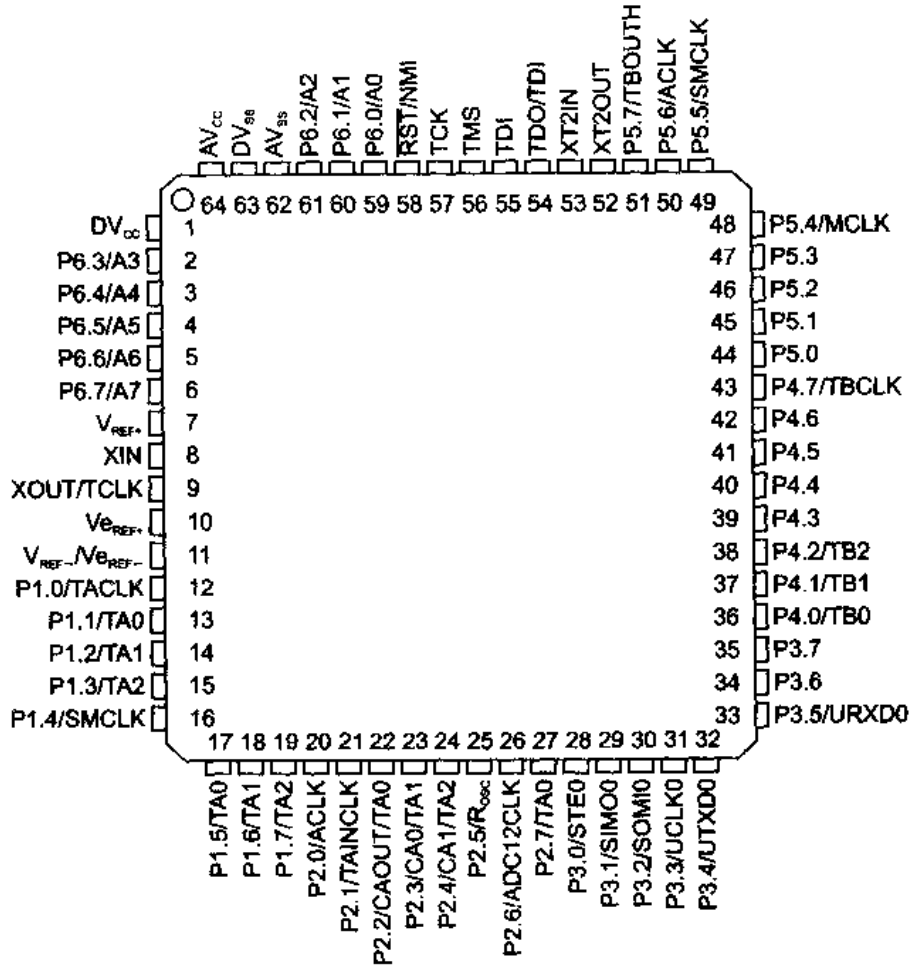


图 1.8　MSP430X13X 系列引脚图

续表 1.4

引脚名称	引脚序号	I/O	说明
P5.3	17	I/O	通用数字 I/O 引脚
P5.4/MCLK	18	I/O	通用数字 I/O 引脚 / 主系统时钟 MCLK 输出
P5.5/SMCLK	19	I/O	通用数字 I/O 引脚 / 子系统时钟 SMCLK 输出
P5.6/ACLK	50	I/O	通用数字 I/O 引脚 / 辅助时钟 ACLK 输出
P5.7/TBOUTH	51	I/O	通用数字 I/O 引脚 / 切换所有的 PWM 数字输出口为高阻抗 — 定时器 B_3 TB0～TB2
P6.0/A0	59	I/O	通用数字 I/O 引脚 / 12 位 A/D 转换器模拟输入通道 0
P6.1/A1	60	I/O	通用数字 I/O 引脚 / 12 位 A/D 转换器模拟输入通道 1
P6.2/A2	61	I/O	通用数字 I/O 引脚 / 12 位 A/D 转换器模拟输入通道 2
P6.3/A3	2	I/O	通用数字 I/O 引脚 / 12 位 A/D 转换器模拟输入通道 3
P6.4/A4	3	I/O	通用数字 I/O 引脚 / 12 位 A/D 转换器模拟输入通道 4
P6.5/A5	4	I/O	通用数字 I/O 引脚 / 12 位 A/D 转换器模拟输入通道 5
P6.6/A6	5	I/O	通用数字 I/O 引脚 / 12 位 A/D 转换器模拟输入通道 6
P6.7/A7	6	I/O	通用数字 I/O 引脚 / 12 位 A/D 转换器模拟输入通道 7
$\overline{RST}$/NMI	58	I	复位输入/不可屏蔽中断输入口，或自动加载程序启动(FLASH 版本器件有此功能)
TCK	57	I	测试时钟，TCK 是用于器件测试与自动加载程序启动的时钟输入接口(FLASH 版本器件有此功能)
TMS	56	I	测试方式选择，器件编程与测试的输入口
TDI	55	I	测试数据输入口，器件的保护熔丝被连接到 TDI
TDO/TDI	54	I/O	测试数据输出口 编程数据输入口
Ve_{REF-}	10	I/O	送到模数转换器 ADC12 的外部基准电压
V_{REF+}	7	O	模数转换器 ADC12 内部基准电压的正输出端
V_{REF-} /Ve_{REF-}	11	O	模数转换器 ADC12 内部基准电压或外部加的基准电压负端
XIN	8	I	晶体振荡器 XT1 的输入口
XOUT/TCLK	9	I/O	晶体振荡器 XT1 的输出口或测试时钟的输入口
XT2IN	53	I	晶体振荡器 XT2 的输入口，只能接标准晶体
XT2OUT	52	O	晶体振荡器 XT2 的输出口
AV_{CC}	64		模拟电源的正输入端，送到模数转换器 ADC12 的模拟部分
AV_{SS}	62		模拟电源的负输入端，送到模数转换器 ADC12 的模拟部分
DV_{CC}	1		数字电源的正输入端
DV_{SS}	63		数字电源的负输入端

1.2.4　MSP430X14X 系列

MSP430X14X 系列器件主要有 MSP430F143、MSP430F148、MSP430F149。

1. 特　点

- 低电源电压范围：1.8～3.6 V。

- 超低功耗：

  2.5 μA @ 4 kHz、2.2 V；

  160 μA @ 1 MHz、2.2 V。

- 5 种节电模式：

  等待方式 0.7 μA；

  RAM 保持的节电方式 0.1 μA。

- 从等待方式唤醒时间：6 μs。
- 16 位 RISC 结构，150 ns 指令周期。
- 基本时钟模块配置：

  高速晶体(最高 8 MHz)；

  低速晶体(32 768 Hz)；

  DCO。

- 配合外部器件可构成单斜边 A/D 转换器。
- 12 位 200 kbps 的 A/D 转换器，自带采样保持。
- 内部温度传感器。
- 具有 3 个捕获/比较寄存器的 16 位定时器 Timer_A，Timer_B。
- 两通道串行通信接口可用于异步或同步(软件选择 UART / SPI 模式)。
- 硬件乘法器。
- 多达 60 KB FLASH ROM 和 2 KB RAM。
- 串行在系统编程。
- 安全熔丝的程序代码保护。

2. 结构框图

该系列结构框图如图 1.9 所示。

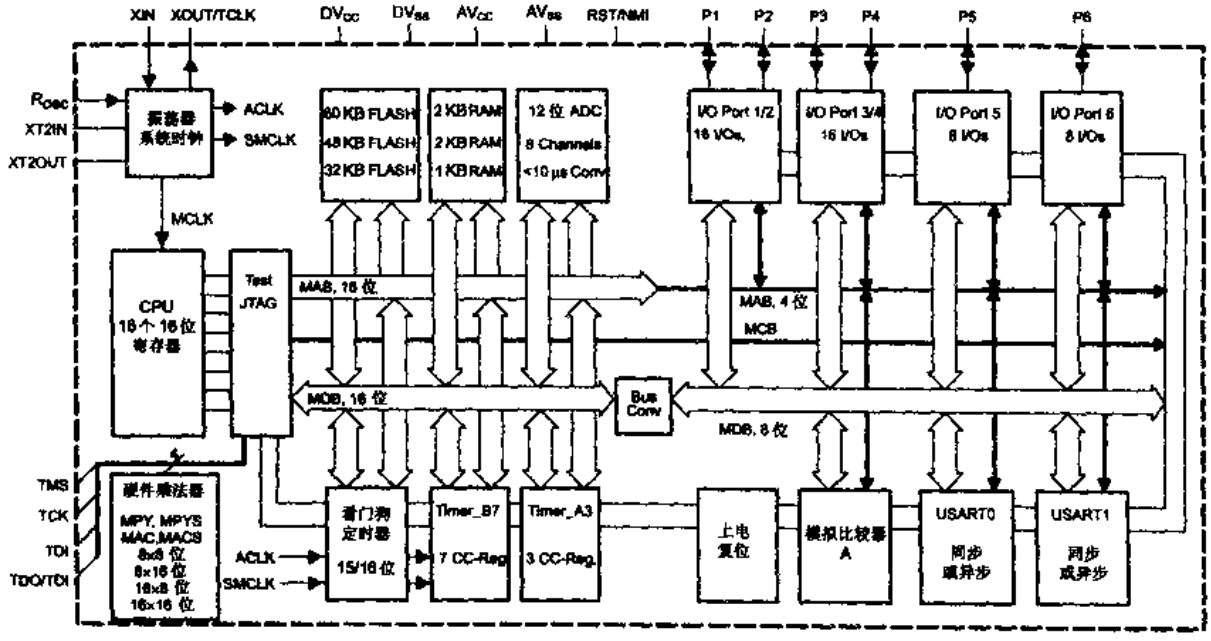


图 1.9 MSP430X14X 系列结构框图

3. 引脚图及说明

MSP430X14X 系列引脚图如图 1.10 所示。

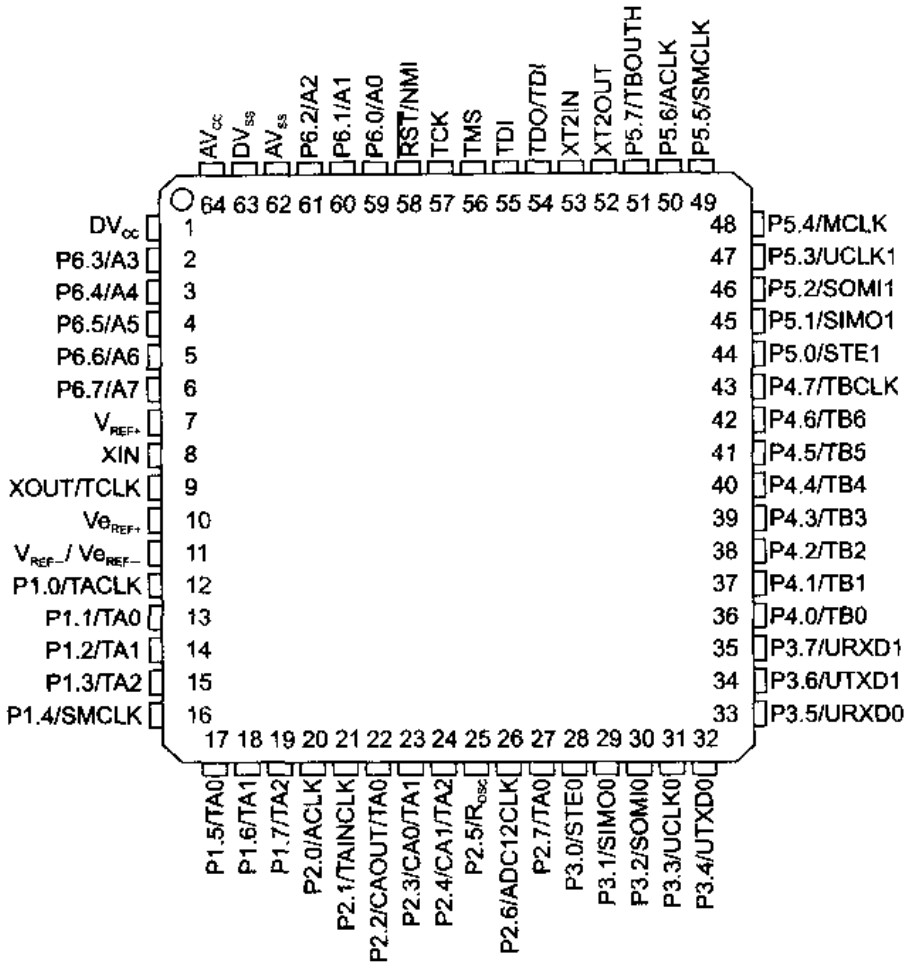


图 1.10　MSP430X14X 系列引脚图

MSP430X14X 系列单片机引脚说明如表 1.5 所列。

表 1.5　MSP430X14X 系列单片机引脚说明

引脚		I/O	说明
名称	序号		
P1.0/TACLK	12	I/O	通用数字 I/O 引脚 / Timer_A，TACLK 时钟信号输入
P1.1/TA0	13	I/O	通用数字 I/O 引脚 / Timer_A，捕获：CCI0A 输入，比较：OUT0 输出
P1.2/TA1	14	I/O	通用数字 I/O 引脚 / Timer_A，捕获：CCI1A 输入，比较：OUT1 输出
P1.3/TA2	15	I/O	通用数字 I/O 引脚 / Timer_A，捕获：CCI2A 输入，比较：OUT2 输出
P1.4/SMCLK	16	I/O	通用数字 I/O 引脚，SMCLK 信号输出
P1.5/TA0	17	I/O	通用数字 I/O 引脚 / Timer_A，比较：OUT0 输出
P1.6/TA1	18	I/O	通用数字 I/O 引脚 / Timer_A，比较：OUT1 输出
P1.7/TA2	19	I/O	通用数字 I/O 引脚 / Timer_A，比较：OUT2 输出
P2.0/ACLK	20	I/O	通用数字 I/O 引脚 / ACLK 输出端
P2.1/TAINCLK	21	I/O	通用数字 I/O 引脚 / Timer_A，INCLK 时钟信号

续表 1.5

引脚名称	引脚序号	I/O	说明
P2.2/CAOUT/TA0	22	I/O	通用数字 I/O 引脚，Timer_A，捕获：CCI0B 输入，比较：OUT0 输出
P2.3/CA0/TA1	23	I/O	通用数字 I/O 引脚，Timer_A，捕获：CCI1B 输入，比较：OUT1 输出
P2.4/CA1/TA2	24	I/O	通用数字 I/O 引脚，Timer_A，比较：OUT2 输出
P2.5/R_{OSC}	25	I/O	通用数字 I/O 引脚，外接一电阻用以决定 DCO 频率
P2.6/ADC12CLK	26	I/O	通用数字 I/O 引脚，12 位 A/D 转换器的转换时钟
P2.7/TA0	27	I/O	通用数字 I/O 引脚，Timer_A，比较：OUT0 输出
P3.0/STE0	28	I/O	通用数字 I/O 引脚，从机传输使能—USART0，SPI 模式
P3.1/SIMO0	29	I/O	通用数字 I/O 引脚，USART0，SPI 模式的从输入或主输出
P3.2/SOMI0	30	I/O	通用数字 I/O 引脚，USART0，SPI 模式的从输出或主输入
P3.3/UCLK0	31	I/O	通用数字 I/O 引脚，外部时钟输入—USART0，UART 或 SPI 模式，时钟输出——USART0，SPI 模式
P3.4/UTXD0	32	I/O	通用数字 I/O 引脚，发送数据输出——USART0，UART 模式
P3.5/URXD0	33	I/O	通用数字 I/O 引脚，接收数据输入—USART0，UART 模式
P3.6/UTXD1	34	I/O	通用数字 I/O 引脚，发送数据输出——USART1，UART 模式
P3.7/URXD1	35	I/O	通用数字 I/O 引脚，接收数据输入—USART1，UART 模式
P4.0/TB0	36	I/O	通用数字 I/O 引脚，捕获 I/P 或 PWM 输出口——定时器 B_7 CCR0
P4.1/TB1	37	I/O	通用数字 I/O 引脚，捕获 I/P 或 PWM 输出口——定时器 B_7 CCR1
P4.2/TB2	38	I/O	通用数字 I/O 引脚，捕获 I/P 或 PWM 输出口——定时器 B_7 CCR2
P4.3/TB3	39	I/O	通用数字 I/O 引脚，捕获 I/P 或 PWM 输出口——定时器 B_7 CCR3
P4.4/TB4	40	I/O	通用数字 I/O 引脚，捕获 I/P 或 PWM 输出口——定时器 B_7 CCR4
P4.5/TB5	41	I/O	通用数字 I/O 引脚，捕获 I/P 或 PWM 输出口—定时器 B_7 CCR5
P4.6/TB6	42	I/O	通用数字 I/O 引脚，捕获 I/P 或 PWM 输出口——定时器 B_7 CCR6
P4.7/TBCLK	43	I/O	通用数字 I/O 引脚，定时器 B_3 的输入时钟 TBCLK
P5.0/STE1	44	I/O	通用数字 I/O 引脚，从机发送使能——USART1，SPI 模式
P5.1/SIMO1	45	I/O	通用数字 I/O 引脚，USART1 的从输入、主输出或 SPI 方式
P5.2/SOMI1	46	I/O	通用数字 I/O 引脚，USART1 的从输出、主输入或 SPI 方式
P5.3/UCLK1	47	I/O	通用数字 I/O 引脚，外部时钟输入——USART1，UART 或 SPI 模式，时钟输出—USART1，SPI 模式
P5.4/MCLK	48	I/O	通用数字 I/O 引脚，主系统时钟 MCLK 输出
P5.5/SMCLK	49	I/O	通用数字 I/O 引脚，子系统时钟 SMCLK 输出
P5.6/ACLK	50	I/O	通用数字 I/O 引脚，辅助时钟 ACLK 输出
P5.7/TBOUTH	51	I/O	通用数字 I/O 引脚，切换所有的 PWM 数字输出口为高阻抗—定时器 B_3 TB0--TB2
P6.0/A0	59	I/O	通用数字 I/O 引脚，12 位 A/D 转换器模拟输入通道 0
P6.1/A1	60	I/O	通用数字 I/O 引脚，12 位 A/D 转换器模拟输入通道 1
P6.2/A2	61	I/O	通用数字 I/O 引脚，12 位 A/D 转换器模拟输入通道 2
P6.3/A3	2	I/O	通用数字 I/O 引脚，12 位 A/D 转换器模拟输入通道 3
P6.4/A4	3	I/O	通用数字 I/O 引脚，12 位 A/D 转换器模拟输入通道 4

续表 1.5

引脚名称	序号	I/O	说明
P6.5/A5	4	I/O	通用数字 I/O 引脚　12 位 A/D 转换器模拟输入通道 5
P6.6/A6	5	I/O	通用数字 I/O 引脚　12 位 A/D 转换器模拟输入通道 6
P6.7/A7	6	I/O	通用数字 I/O 引脚　12 位 A/D 转换器模拟输入通道 7
$\overline{RST}$/NMI	58	I	复位输入/不可屏蔽中断输入口，或自动加载程序启动(FLASH 版本器件有此功能)
TCK	57	I	测试时钟，TCK 是用于器件测试与自动加载程序启动的时钟输入接口(FLASH 版本器件有此功能)
TMS	56	I	测试方式选择，器件编程与测试的输入口
TDI	55	I	测试数据输入口，器件的保护熔丝被连接到 TDI
TDO/TDI	54	I/O	测试数据输出口，编程数据输入口
V_{eREF+}	10	I/P	送到模数转换器 ADC12 的外部基准电压
V_{REF+}	7	O	模数转换器 ADC12 内部基准电压的正输出端
V_{REF-}/V_{eREF-}	11	O	模数转换器 ADC12 内部基准电压或外部加的基准电压负端
XIN	8	I	晶体振荡器 XT1 的输入口
XOUT/TCLK	9	I/O	晶体振荡器 XT1 的输出口或测试时钟的输入口
XT2IN	53	I	晶体振荡器 XT2 的输入口，只能接标准晶体
XT2OUT	52	O	晶体振荡器 XT2 的输出口
AV_{CC}	64		模拟电源的正输入端，送到模数转换器 ADC12 的模拟部分
AV_{SS}	62		模拟电源的负输入端，送到模数转换器 ADC12 的模拟部分
DV_{CC}	1		数字电源的正输入端
DV_{SS}	63		数字电源的负输入端

1.3　MSP430X3XX 系列

该系列目前有以下一些器件：

MSP430C311　MSP430C312　MSP430C313　MSP430C314　MSP430C315
MSP430P315　MSP430C311S　MSP430C315S
MSP430C323　MSP430C325　MSP430P325　MSP430P325A
MSP430C336　MSP430C337　MSP430P337　MSP430P337A

这些器件又可分为：MSP430X31X 系列；MSP430X32X 系列；MSP430X33X 系列。

目前这些器件只有 ROM 型和 OTP 型，它们的开发需要借助仿真器或相应的开发工具，或使用 EPROM(窗口)型的开发片。它们都内含液晶驱动电路，能直接与液晶屏或液晶模块相连接。MSP430X32X 系列更有 14 位模数转换器。所有 MSP430X3XX 系列器件都有三态输出端口，这是与其他系列器件的不同之处。

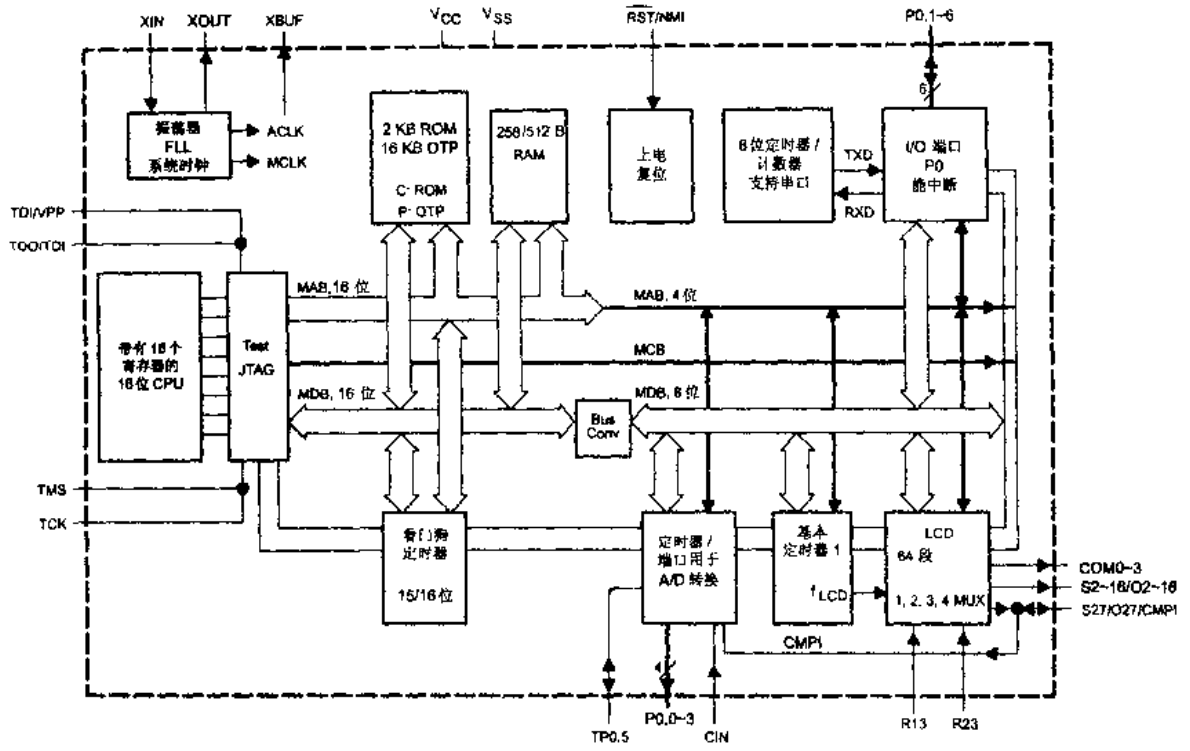


图 1.12　48 引脚的 MSP430X31X 系列结构框图

3. 引脚图及说明

48 脚和 56 脚封装形式的引脚图分别如图 1.13 和图 1.14 所示。

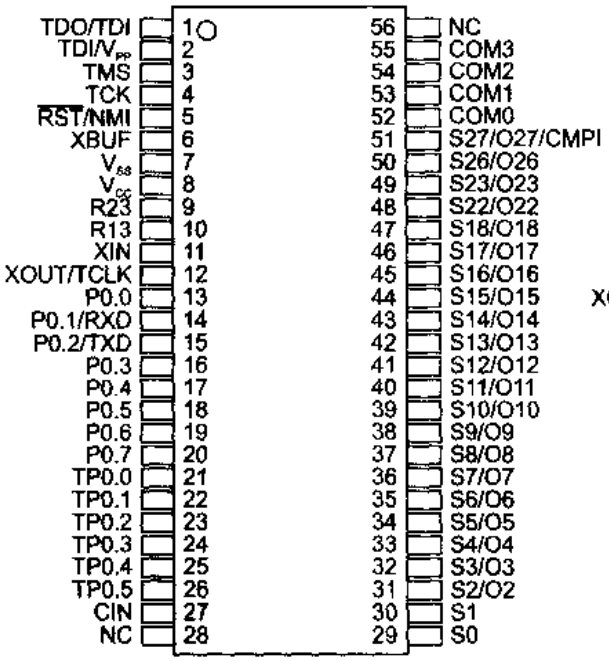


图 1.13　56 引脚的 MSP430X31X 系列引脚图

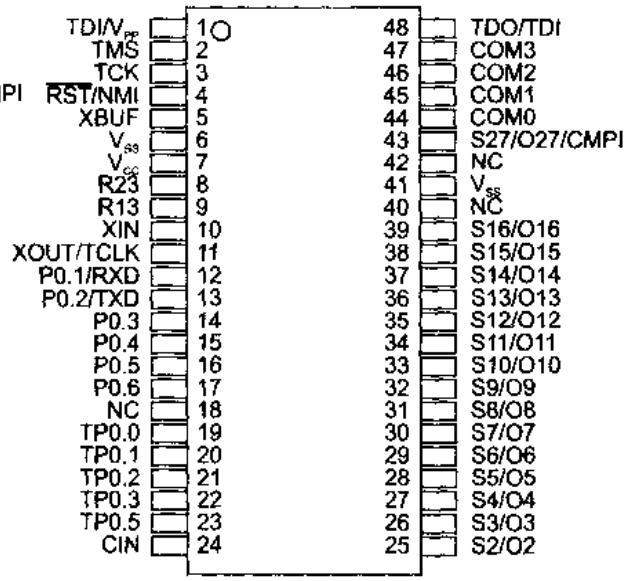


图 1.14　48 引脚的 MSP430X31X 系列引脚图

引脚说明分别如表 1.6 和表 1.7 所列。

表 1.6 56 引脚 MSP430X31X 器件引脚说明

引脚 名称	引脚 序号	I/O	说明
CIN	27	I	计数器使能
COM0～COM3	52～55	O	COM0～COM3 为液晶使用的公共输出引脚
P0.0	13	I/O	通用数字 I/O 引脚
P0.1/RXD	14	I/O	通用数字 I/O 引脚，数据接收端口
P0.2/TXD	15	I/O	通用数字 I/O 引脚，数据发送端口
P0.3～P0.7	16～20	I/O	通用数字 I/O 引脚
R23	9	I	LCD 模拟电压第二正输入(V2)
R13	10	I	LCD 模拟电压第一正输入(V3,V4)
$\overline{RST}$/NMI	5	I	复位输入端或不可屏蔽中断输入端
S0	29	O	LCD 段输出 S0
S1	30	O	LCD 段输出 S1
S2,O2～S5/O5	31～34	O	通用数字输出引脚，LCD 段输出
S6,O6～S9/O9	35～38	O	通用数字输出引脚，LCD 段输出
S10/O10～S13/O13	39～42	O	通用数字输出引脚，LCD 段输出
S14/O14～S17/O17	43～46	O	通用数字输出引脚，LCD 段输出
S18/O18	47	O	通用数字输出引脚，LCD 段输出
S22/O22～S23/O23	48,49	O	通用数字输出引脚，LCD 段输出
S26/O26	50	O	通用数字输出引脚，LCD 段输出
S27/O27/CMPI	51	I/O	通用数字输出引脚，LCD 段输出/比较器的输入端
TCK	4	I	测试时钟，TCK 是用于器件测试与编程的时钟输入接口
TDI/V_{PP}	2	I	测试数据输入口，编程电压也由此送入
TDO/TDI	1	I/O	测试数据输出口，编程数据输入口
TMS	3	I	测试方式选择，器件编程与测试的输入口
TP0.0	21	O	通用数字三态输出引脚 0(定时器/端口)
TP0.1	22	O	通用数字三态输出引脚 1(定时器/端口)
TP0.2	23	O	通用数字三态输出引脚 2(定时器/端口)
TP0.3	24	O	通用数字三态输出引脚 3(定时器/端口)
TP0.4	25	O	通用数字三态输出引脚 4(定时器/端口)
TP0.5	26	I/O	通用数字三态输出引脚 5(定时器,端口)
V_{CC}	8		电源电压正输入端
V_{SS}	7		电源电压负输入端
XBUF	6	O	系统时钟(MCLK)或晶体时钟(ACLK)的输出
XIN	11	I	晶体连接端
XOUT/TCLK	12	I/O	晶体振荡器信号输出或测试时钟输入

表 1.7　48 引脚 MSP430X31X 器件引脚说明

引脚名称	引脚序号	I/O	说明
CIN	24	I	计数器使能
COM0～COM3	44～47	O	COM0～COM3 为液晶使用的公共输出引脚
P0.1/RXD	12	I/O	通用数字 I/O 引脚 / 数据接收端口
P0.2/TXD	13	I/O	通用数字 I/O 引脚 / 数据发送端口
P0.3～P0.6	14～17	I/O	通用数字 I/O 引脚
R23	8	I	LCD 模拟电压第二正输入(V2)
R13	9	I	LCD 模拟电压第二正输入(V3,V4)
$\overline{RST}$/NMI	4	I	复位输入端/不可屏蔽中断输入端
S2/O2～S5/O5	25～28	O	通用数字输出引脚 / LCD 段输出
S6/O6～S9/O9	29～32	O	通用数字输出引脚 / LCD 段输出
S10/O10～S13/O13	33～36	O	通用数字输出引脚 / LCD 段输出
S14/O14～S16/O16	37～39	O	通用数字输出引脚 / LCD 段输出
S27/O27/CMPI	45	I/O	通用数字输出引脚 / LCD 段输出,也能被用作比较器的输入端
TCK	3	I	测试时钟,TCK 是用于器件测试与编程的时钟输入接口
TDI/V_{PP}	1	I	测试数据输入口/编程电压也由此送入
TDO/TDI	48	I/O	测试数据输出口/编程数据输入口
TMS	2	I	测试方式选择,器件编程与测试的输入口
TP0.0	19	O	通用数字三态输出引脚 0(定时器/端口)
TP0.1	20	O	通用数字三态输出引脚 1(定时器/端口)
TP0.2	21	O	通用数字三态输出引脚 2(定时器/端口)
TP0.3	22	O	通用数字三态输出引脚 3(定时器/端口)
TP0.5	23	I/O	通用数字三态输出引脚 5(定时器/端口)
V_{CC}	7		电源电压正输入端
V_{SS}	6,41		电源电压负输入端
XBUF	5	O	系统时钟(MCLK)或晶体时钟(ACLK)的输出
XIN	10	I	晶体连接端
XOUT/TCLK	11	I/O	晶体振荡器信号输出或测试时钟输入

1.3.2　MSP430X32X 系列

1. 特　点

- 低电压:2.5～5.5 V 工作电压。
- 低功耗:3 V,1 MHz 时耗电 400 μA;

备用时耗电可降到 0.1 μA。

- 有 5 种节电模式。
- 单一 32 kHz 外部晶体，内部可运行在 3.3 MHz 频率下。
- 从备用唤醒只要 6 μs。
- 16 位 RISC 结构，指令周期 300 ns。
- 内嵌液晶驱动电路，驱动液晶多达 84 段。
- 三态输出端口。
- 内部整合 12+2 位模数转换器。
- 有 EPROM 版本器件：PMS430E325A。
- 串行在系统编程。
- 通过加密熔丝保护设计者代码。

2. 结构框图

该系列结构框图如图 1.15 所示。

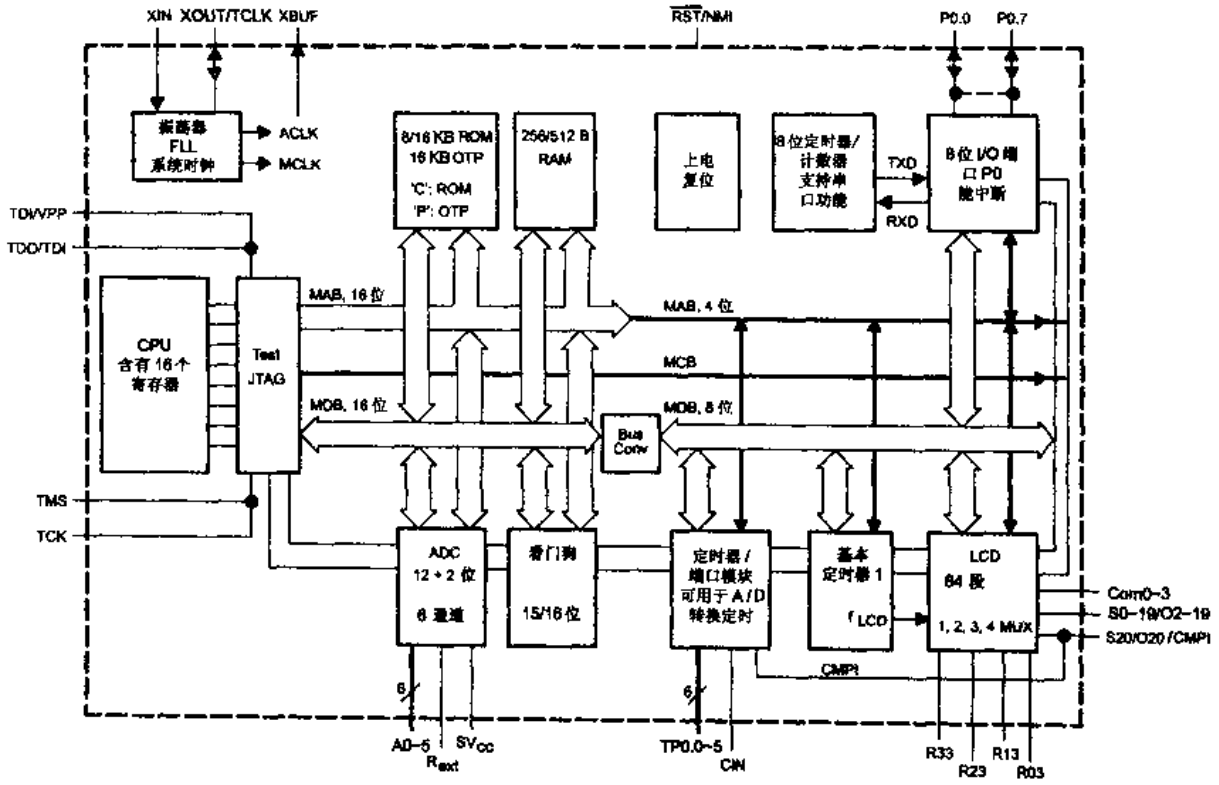


图 1.15 MSP430X32X 系列结构框图

3. 引脚图及说明

该系列脚图如图 1.16 所示。

该系列引脚说明如表 1.8 所列。

续表 1.8

引脚 名称	引脚 序号	I/O	说明
S1	31	O	LCD 段输出 S1
S2/O2～S5/O5	32～35	O	通用数字输出引脚，LCD 段输出
S6/O6～S9/O9	36～39	O	通用数字输出引脚，LCD 段输出
S10/O10～S13, O13	40～43	O	通用数字输出引脚，LCD 段输出
S14/O14～S17/O17	44～47	O	通用数字输出引脚，LCD 段输出
S18/O18～S19, O19	48～49	O	通用数字输出引脚，LCD 段输出
S20/O20/CMPI	50	I/O	通用数字输出引脚，LCD 段输出，比较器输入端口
TCK	58	I	测试时钟，TCK 是用于器件测试与编程的时钟输入接口
TDI/V_{PP}	56	I	测试数据输入口，编程电压也由此送入
TDO/TDI	55	I/O	测试数据输出口，或编程数据输入口
TMS	57	I	测试方式选择，器件编程与测试的输入口
TP0.0	12	O	通用数字三态输出引脚 0(定时器/端口)
TP0.1	13	O	通用数字三态输出引脚 1(定时器/端口)
TP0.2	14	O	通用数字三态输出引脚 2(定时器/端口)
TP0.3	15	O	通用数字三态输出引脚 3(定时器/端口)
TP0.4	16	O	通用数字三态输出引脚 4(定时器/端口)
TP0.5	17	O	通用数字三态输出引脚 5(定时器/端口)
XBUF	60	O	系统时钟(MCLK)或晶体时钟(ACLK)的输出
XIN	9	I	晶体连接端
XOUT/TCLK	10	I/O	晶体振荡器信号输出或测试时钟输入

1.3.3 MSP430X33X 系列

1. 特 点

- 低电压：2.5～5.5 V 工作电压。
- 低功耗：3 V，1 MHz 时耗电 400 μA；备用时耗电可降到 0.1 μA。
- 有 5 种节电模式。
- 单一 32 kHz 外部晶体，内部可运行在 3.8 MHz 频率下。
- 从备用唤醒只要 6 μs。
- 16 位 RISC 结构，指令周期 300 ns。
- 内嵌液晶驱动电路，驱动液晶多达 120 段。
- 三态输出端口。
- 配合外部器件可实现单斜边 A/D 转换。
- 16 位定时器带 5 个捕获/比较寄存器。
- 内部硬件乘法器可执行 16×16 位的操作。
- 串行通信可软件选择 UART/SPI 模式。
- 有 EPROM 版本器件：PMS430E337A。
- 串行在系统编程。
- 通过加密熔丝保护设计者代码。

2. 结构框图

该系列结构框图如图 1.17 所示。

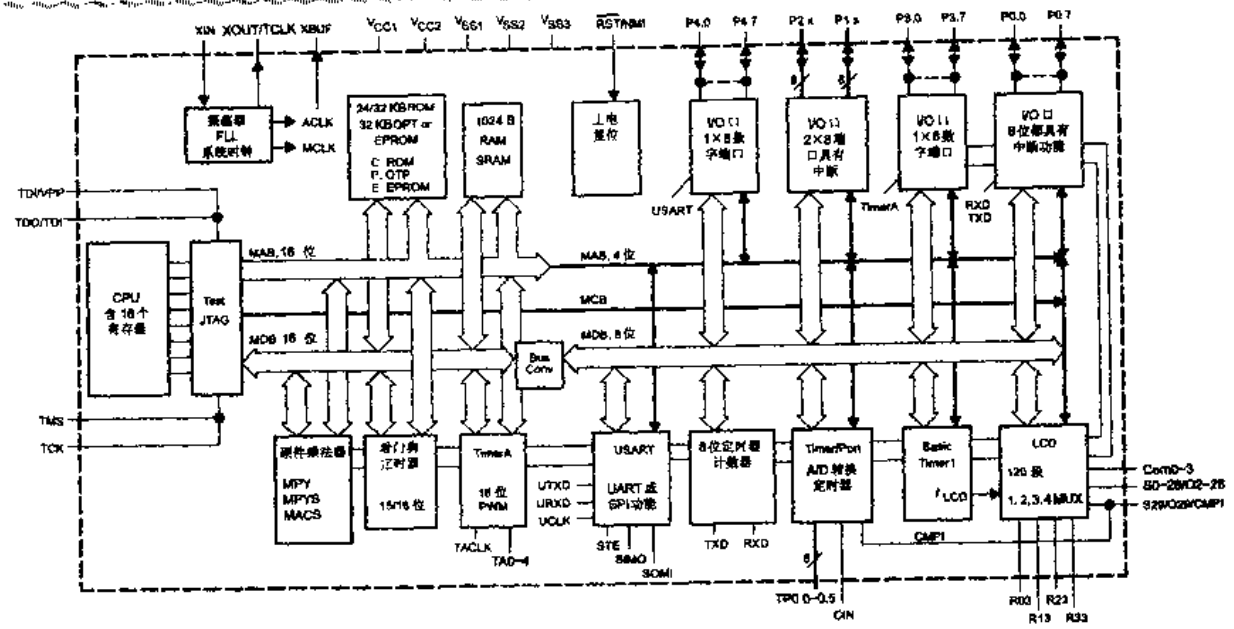


图 1.17　MSP430X33X 系列结构框图

3. 引脚图及引脚说明

该系列引脚图如图 1.18 所示。

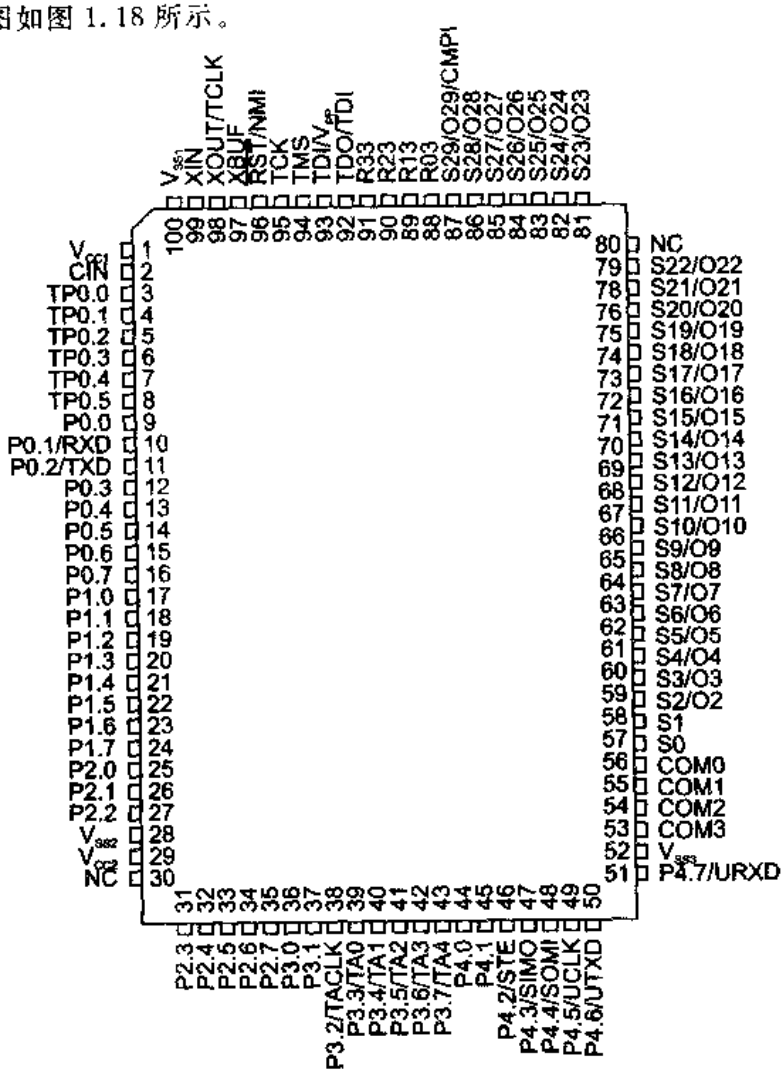


图 1.18　MSP430X33X 引脚图

该系列引脚说明如表 1.9 所列。

表 1.9 MSP430X33X 引脚说明

引脚名称	引脚序号	I/O	说明
CIN	2	I	计数器使能
COM0~COM3	56~53	O	COM0~COM3 为液晶使用的公共输出引脚
NC	30,80		空脚
P0.0	9	I/O	通用数字 I/O 引脚
P0.1,RXD	10	I/O	通用数字 I/O 引脚，数据接收端口
P0.2,TXD	11	I/O	通用数字 I/O 引脚，数据发送端口
P0.3~P0.7	12~16	I/O	通用数字 I/O 引脚
P1.0~P1.7	17~24	I/O	通用数字 I/O 引脚
P2.0~P2.2	25~27	I/O	通用数字 I/O 引脚
P2.3~P2.7	31~35	I/O	通用数字 I/O 引脚
P3.0,P3.1	36,37	I/O	通用数字 I/O 引脚
P3.2/TACLK	38	I/O	通用数字 I/O 引脚，Timer_A 的时钟输入
P3.3/TA0	39	I/O	通用数字 I/O 引脚，捕获端口或 PWM 输出端口，Timer_A CCR0
P3.4/TA1	40	I/O	通用数字 I/O 引脚，捕获端口或 PWM 输出端口，Timer_A CCR1
P3.5/TA2	41	I/O	通用数字 I/O 引脚，捕获端口或 PWM 输出端口，Timer_A CCR2
P3.6/TA3	42	I/O	通用数字 I/O 引脚，捕获端口或 PWM 输出端口，Timer_A CCR3
P3.7/TA4	43	I/O	通用数字 I/O 引脚，捕获端口或 PWM 输出端口，Timer_A CCR4
P4.0	44	I/O	通用数字 I/O 引脚
P4.1	45	I/O	通用数字 I/O 引脚
P4.2/STE	46	I/O	通用数字 I/O 引脚，从属传送使能，USART,SPI 模式
P4.3/SIMO	47	I/O	通用数字 I/O 引脚，从输入、主输出，USART/SPI 模式
P4.4/SOMI	48	I/O	通用数字 I/O 引脚，主输入、从输出，USART,SPI 模式
P4.5/UCLK	49	I/O	通用数字 I/O 引脚，外部时钟输入，USART 模式
P4.6/UTXD	50	I/O	通用数字 I/O 引脚，发送数据输出，USART/UART 模式
P4.7/URXD	51	I/O	通用数字 I/O 引脚，接收数据输入，USART/UART 模式
R03	88		液晶电压 V4
R13	89		液晶电压 V3
R23	90		液晶电压 V2
R33	91		液晶电压 V1
$\overline{RST}$/NMI	96	I	复位输入/非屏蔽中断输入端口
S0	57	O	LCD 段输出 S0
S1	58	O	LCD 段输出 S1
S2/O2~S5/O5	59~62	O	通用数字输出引脚，LCD 段输出
S6/O6~S9/O9	63~66	O	通用数字输出引脚，LCD 段输出
S10,O10~S13/O13	67~70	O	通用数字输出引脚，LCD 段输出

续表 1.9

引脚名称	引脚序号	I/O	说明
S14/O14~S17/O17	71~74	O	通用数字输出引脚，LCD 段输出
S18/O18~S21/O21	75~78	O	通用数字输出引脚，LCD 段输出
S22/O22	79		通用数字输出引脚，LCD 段输出
S23/O23~S25/O25	81~83	O	通用数字输出引脚，LCD 段输出
S26/O26~S28/O28	84~86	O	通用数字输出引脚，LCD 段输出
S29/O29/CMPI	87	I/O	通用数字输出引脚，LCD 段输出，比较器输入端口
TCK	95	I	测试时钟，TCK 是用于器件测试与编程的时钟输入接口
TDI, V_{PP}	93	I	测试数据输入口/编程电压也由此送入
TDO/TDI	92	I/O	测试数据输出口/编程数据输入口
TMS	91	I	测试方式选择，器件编程与测试的输入口
TP0.0	3	O	通用数字三态输出引脚 0（定时器/端口）
TP0.1	4	O	通用数字三态输出引脚 1（定时器/端口）
TP0.2	5	O	通用数字三态输出引脚 2（定时器/端口）
TP0.3	6	O	通用数字三态输出引脚 3（定时器/端口）
TP0.4	7	O	通用数字三态输出引脚 4（定时器/端口）
TP0.5	8	I/O	通用数字三态输出引脚 5（定时器/端口）
V_{CC1}	1		电源电压正端
V_{CC2}	29		电源电压正端
V_{SS1}	100		电源电压负端
V_{SS2}	28		电源电压负端
V_{SS3}	52		电源电压负端
XBUF	97	O	系统时钟(MCLK)或晶体时钟(ACLK)的输出
XIN	99	I	晶体连接端
XOUT/TCLK	98	I/O	晶体振荡器信号输出/测试时钟输入

1.4　MSP430X4XX 系列

该系列包含以下器件：

MSP430C412　MSP430C413

MSP430F412　MSP430F413

MSP430F435　MSP430F436　MSP430F437

MSP430F447　MSP430F448　MSP430F449

除 MSP430C412 MSP430C413 之外这里将它们细分为：MSP430F41X 系列；MSP430F43X 系列；MSP430F44X 系列。这些器件都是 FLASH ROM 型的，开发起来非常方便，都能直接驱动液晶。

1.4.1　MSP430X41X 系列

该系列目前有 4 个器件：

MSP430C412　　MSP430C413

MSP430F412　　MSP430F413

1. 特点

- 低工作电压：1.8 V～3.6V。
- 超低功耗：

  活动模式　225 μA　@1 MHz，2.2 V；

  待机模式　0.8 μA；

  掉电模式(RAM 数据保持)　0.1 μA。
- 5 种节电模式。
- 从待机到唤醒的响应时间不超过 6 μs。
- 片内频率锁相环 FLL＋，可使系统工作在稳定的频率上。
- 16 位精简指令结构(RISC)，150 ns 指令周期。
- 带有 3 个捕获/比较寄存器的 16 位定时器。
- 集成 96 段液晶驱动器。
- 片内比较器配合其他器件可构成单斜边 A/D 转换器。
- 可在线串行编程。
- 可编程的保险熔丝保护设计者代码。
- FLASH 存储器。

2. 结构框图

该系列器件的结构框图如图 1.19 所示。

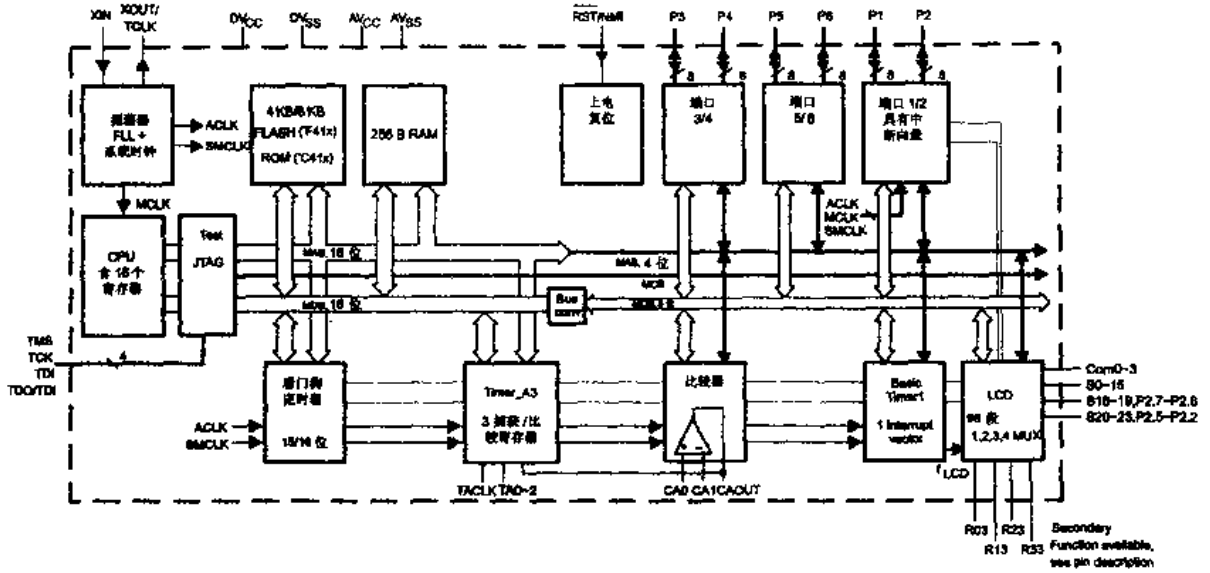


图 1.19　MSP430X41X 结构框图

3. 引脚图及说明

MSP430X41X 系列器件的引脚图如图 1.20 所示。

该系列器件的引脚说明如表 1.10 所列。

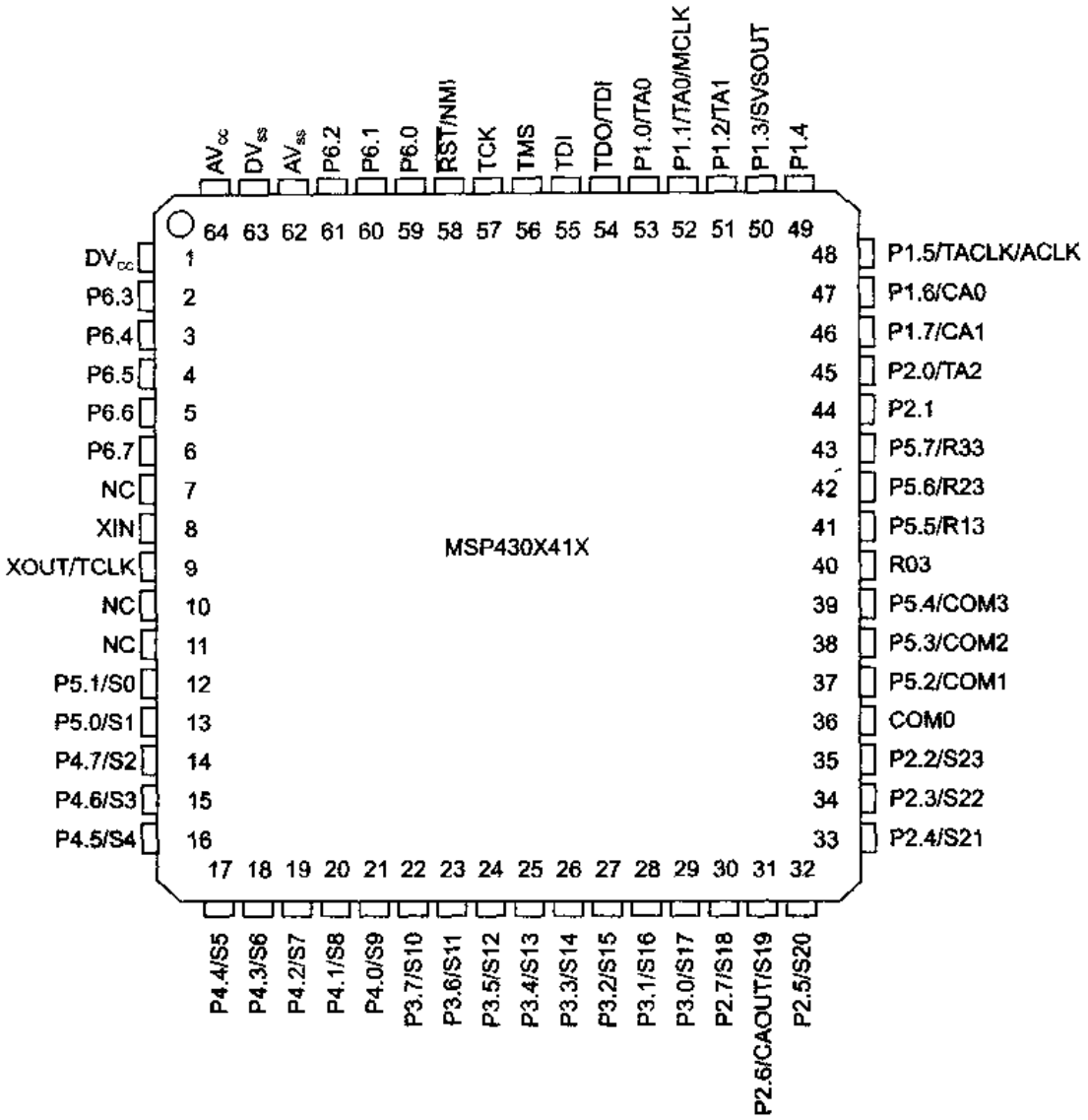


图 1.20　MSP430X41X 系列器件的引脚图

表 1.10　MSP430X41X 引脚说明

引脚		I/O	说　明
名　称	序　号		
AV_{CC}	64		模拟电源正端，向 SVS、节电、振荡器、FLL+、比较器 A 端口 1、LCD 等电路供电
AV_{SS}	62		模拟电源的参考点，内部连接于 DV_{SS}
DV_{CC}	1		数字电源正端
DV_{SS}	63		数字电源参考点
NC	7,10,11		空脚
P1.0/TA0	53	I/O	通用数字 I/O 引脚 / Timer_A，捕获：CCI0A 输入，比较：OUT0 输出
P1.1/TA0/MCLK	52	I/O	通用数字 I/O 引脚 / Timer_A，捕获：CCI0B 输入/MCLK 输出
P1.2/TA1	51	I/O	通用数字 I/O 引脚　Timer_A，捕获：CCI1A 输入，比较：OUT1 输出
P1.3/SVSOUT	50	I/O	通用数字 I/O 引脚 / SVS 比较器的输出端

续表 1.10

引脚 名称	引脚 序号	I/O	说明
P1.4	49	I/O	通用数字 I/O 引脚
P1.5/TACLK/ACLK	48	I/O	通用数字 I/O 引脚 Timer_A 输入时钟/ACLK 输出
P1.6/CA0	47	I/O	通用数字 I/O 引脚，比较器 A 输入端
P1.7/CA1	46	I/O	通用数字 I/O 引脚 比较器 A 输入端
P2.0/TA2	45	I/O	通用数字 I/O 引脚，Timer_A，捕获：CCI2A 输入，比较：OUT2 输出
P2.1	44	I/O	通用数字 I/O 引脚
P2.2/S23	35	I/O	通用数字 I/O 引脚，LCD 段输出
P2.3/S22	34	I/O	通用数字 I/O 引脚，LCD 段输出
P2.4/S21	33	I/O	通用数字 I/O 引脚，LCD 段输出
P2.5/S20	32	I/O	通用数字 I/O 引脚，LCD 段输出
P2.6/CAOUT/S19	31	I/O	通用数字 I/O 引脚，LCD 段输出/比较器 A 输出
P2.7/S18	30	I/O	通用数字 I/O 引脚/LCD 段输出
P3.0/S17	29	I/O	通用数字 I/O 引脚，LCD 段输出
P3.1/S16	28	I/O	通用数字 I/O 引脚/LCD 段输出
P3.2/S15	27	I/O	通用数字 I/O 引脚/LCD 段输出
P3.3/S14	26	I/O	通用数字 I/O 引脚/LCD 段输出
P3.4/S13	25	I/O	通用数字 I/O 引脚/LCD 段输出
P3.5/S12	24	I/O	通用数字 I/O 引脚/LCD 段输出
P3.6/S11	23	I/O	通用数字 I/O 引脚/LCD 段输出
P3.7/S10	22	I/O	通用数字 I/O 引脚/LCD 段输出
P4.0/S9	21	I/O	通用数字 I/O 引脚，LCD 段输出
P4.1/S8	20	I/O	通用数字 I/O 引脚/LCD 段输出
P4.2/S7	19	I/O	通用数字 I/O 引脚/LCD 段输出
P4.3/S6	18	I/O	通用数字 I/O 引脚/LCD 段输出
P4.4/S5	17	I/O	通用数字 I/O 引脚，LCD 段输出
P4.5/S4	16	I/O	通用数字 I/O 引脚/LCD 段输出
P4.6/S3	15	I/O	通用数字 I/O 引脚/LCD 段输出
P4.7/S2	14	I/O	通用数字 I/O 引脚/LCD 段输出
P5.0/S1	13	I/O	通用数字 I/O 引脚/LCD 段输出
P5.1/S0	12	I/O	通用数字 I/O 引脚/LCD 段输出
COM0	36	O	LCD 公共输出端 0
P5.2/COM1	37	I/O	通用数字 I/O 引脚/LCD 公共输出端 1
P5.3/COM2	38	I/O	通用数字 I/O 引脚/LCD 公共输出端 2
P5.4/COM3	39	I/O	通用数字 I/O 引脚/LCD 公共输出端 3
R03	40		液晶电压 V4

续表 1.10

引脚名称	引脚序号	I/O	说明
P5.5 R13	41	I/O	通用数字 I/O 引脚 /液晶电压 V3
P5.6 R23	42	I/O	通用数字 I/O 引脚 /液晶电压 V2
P5.7/R33	43	I/O	通用数字 I/O 引脚 /液晶电压 V1
P6.0	59	I/O	通用数字 I/O 引脚
P6.1	60	I/O	通用数字 I/O 引脚
P6.2	61	I/O	通用数字 I/O 引脚
P6.3	2	I/O	通用数字 I/O 引脚
P6.4	3	I/O	通用数字 I/O 引脚
P6.5	4	I/O	通用数字 I/O 引脚
P6.6	5	I/O	通用数字 I/O 引脚
P6.7	6	I/O	通用数字 I/O 引脚
$\overline{\text{RST}}$ NMI	58	I	复位输入 不可屏蔽中断输入口
TCK	57	I	测试时钟.TCK 是用于器件测试与自动加载程序启动的时钟输入接口（FLASH 版本器件有此功能）
TMS	56	I	测试方式选择.器件编程与测试的输入口
TDI	55	I	测试数据输入口.器件的保护熔丝被连接到 TDI
TDO/TDI	54	I/O	测试数据输出口或编程数据输入口
XIN	8	I	晶体振荡器 XT1 的输入口
XOUT/TCLK	9	I/O	晶体振荡器 XT1 的输出口或测试时钟的输入口

1.4.2　MSP430F43X 系列

该系列目前有 3 个器件：

MSP430F435

MSP430F436

MSP430F437

1. 特　点

- 低工作电压：1.8～3.6 V。
- 超低功耗：

  活动模式　280 μA　@1 MHz,2.2 V；

  待机模式　1.1 μA；

  掉电模式(RAM 数据保持)　0.1 μA。
- 5 种节电模式。

- 从待机到唤醒的响应时间不超过 6 μs。
- 12 位 A/D 转换器带有内部参考源、采样保持、自动扫描特性。
- 16 位精简指令结构(RISC),150 ns 指令周期。
- 带有 3 个捕获/比较寄存器的 16 位定时器有:定时器 A 和定时器 B。
- 串行通信可软件选择 UART/SPI 模式。
- 片内比较器配合其他器件可构成单斜边 A/D 转换器。
- 可编程电压监测器。
- 可在线串行编程,不需要外部编程电压。
- 驱动液晶能力可达 160 段。
- 可编程的保险熔丝保护设计者代码。
- FLASH 存储器。

2. 结构框图

该系列器件的结构框图如图 1.21 所示。

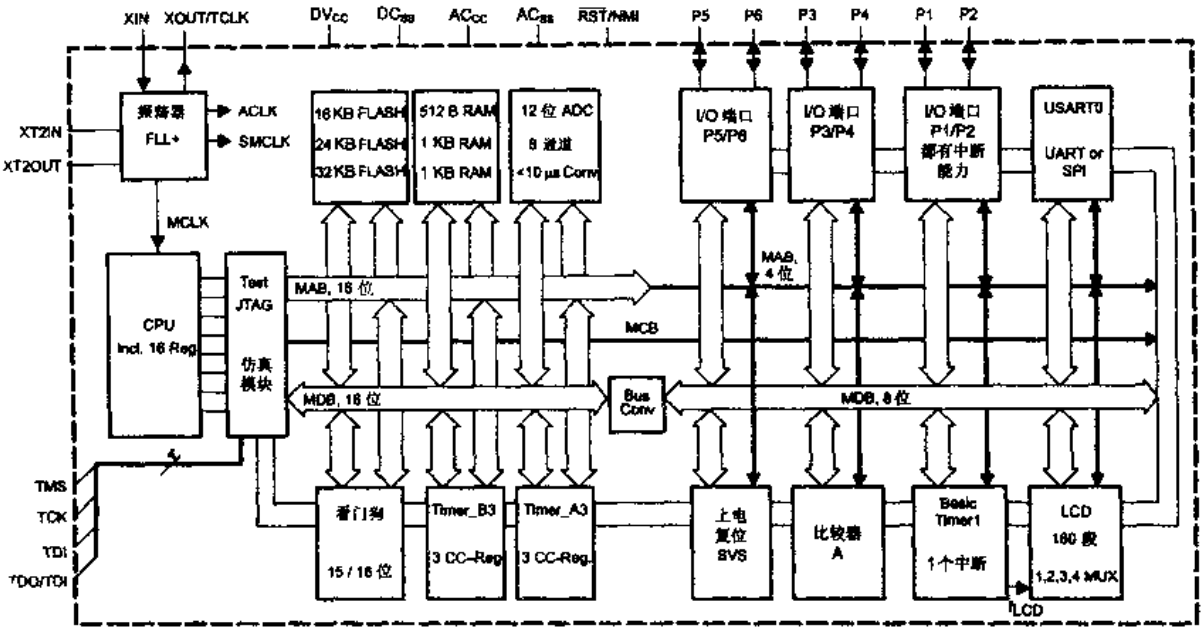


图 1.21 MSP430F43X 结构框图

3. 引脚图及说明

MSP430F43X 系列器件有两种封装形式:80 引脚的 PLASTIC 80 - PIN QFP 和 100 引脚的 PLASTIC 100 - PIN QFP。这里只介绍 80 引脚的引脚图及引脚说明,100 引脚的可参照后面的 MSP430F44X 器件。

80 引脚的 MSP430F43X 系列器件的引脚图如图 1.22 所示。

该系列的引脚说明如表 1.11 所列。

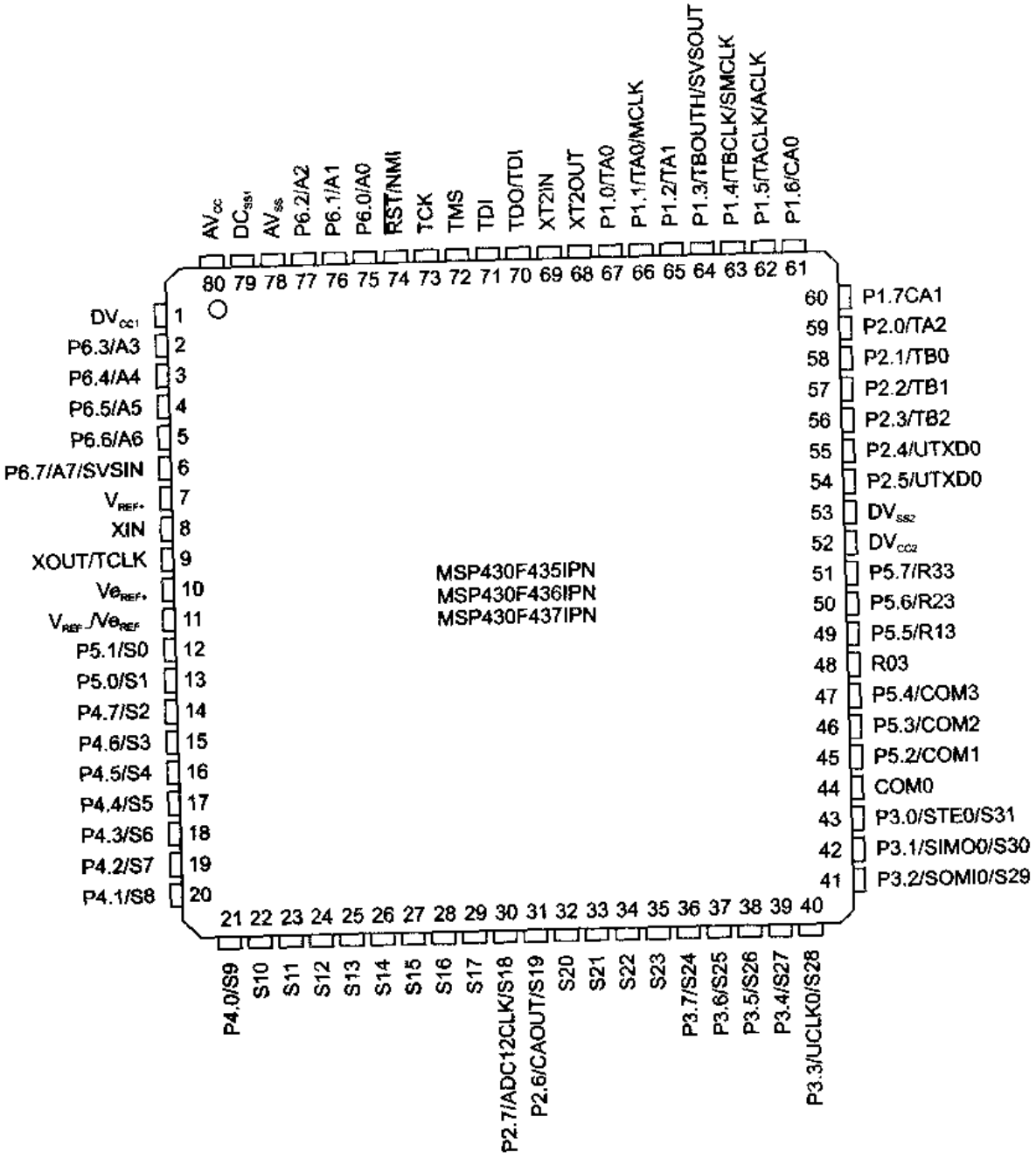


图 1.22　MSP430F43X 系列器件的引脚图

表 1.11　MSP430F43X 引脚说明

引脚		I/O	说明
名称	序号		
DVcc1	1		数字电源电压正端
P6.0/A0～P6.2/A2	75～77	I/O	通用数字 I/O 端口/A/D 转换器模拟输入端
P6.3/A3～P6.6/A6	2～5	I/O	通用数字 I/O 端口/A/D 转换器模拟输入端
P6.7/A7/SVSIN	6	I/O	通用数字 I/O 端口/A/D 转换器模拟输入端/SVS 比较器输入端

续表 1.11

引脚 名称	引脚 序号	I/O	说明
P5.6/R23	50	I/O	通用数字 I/O 端口 / 液晶电压输入端 V2
P5.7/R33	51	I/O	通用数字 I/O 端口 / 液晶电压输入端 V1
DV_{CC2}	52		数字电源电压 / 端
DV_{SS2}	53		数字电源参考点
P2.5/URXD0	54	I/O	通用数字 I/O 端口 / 接收数据输入端—USART0/UART 模式
P2.4/UTXD0	55	I/O	通用数字 I/O 端口 / 发送数据输出端　USART0/UART 模式
P2.3/TB2	56	I/O	通用数字 I/O 端口 / Timer_B3 CCR2，捕获：CCI2A/CCI2B 输入，比较：OUT2 输出
P2.2/TB1	57	I/O	通用数字 I/O 端口 / Timer_B3 CCR1，捕获：CCI1A/CCI1B 输入，比较：OUT1 输出
P2.1/TB0	58	I/O	通用数字 I/O 端口 / Timer_B3 CCR0，捕获：CCI0A/CCI0B 输入，比较：OUT0 输出
P2.0/TA2	59	I/O	通用数字 I/O 端口 / Timer_A，捕获：CCI2A 输入，比较：OUT2 输出
P1.7/CA1	60	I/O	通用数字 I/O 端口 / 比较器_A 输入端
P1.6/CA0	61	I/O	通用数字 I/O 端口 / 比较器_A 输入端
P1.5/TACLK/ACLK	62	I/O	通用数字 I/O 端口 / Timer_A 时钟信号 TACLK 输入端 / ACLK 输出端
P1.4/TBCLK/SMCLK	63	I/O	通用数字 I/O 端口 / Timer_B 时钟信号 TBCLK 输入端 / SMCLK 输出端
P1.3/TBOUTH/SVSOUT	64	I/O	通用数字 I/O 端口 / 设置所有的 PWM 端口为高阻 / SVS 输出
P1.2/TA1	65	I/O	通用数字 I/O 端口 / Timer_A，捕获：CCI1A 输入，比较：OUT1 输出
P1.1/TA0/MCLK	66	I/O	通用数字 I/O 端口 / Timer_A，捕获：CCI0B / MCLK 输出
P1.0/TA0	67	I/O	通用数字 I/O 端口 / Timer_A，捕获：CCI0A，比较：OUT0 输出
XT2OUT	68	O	标准晶体振荡器 2 输出端
XT2IN	69	I	标准晶体振荡器 2 输入端
TDO/TDI	70	I/O	测试数据输出口 / 编程数据输入口
TDI	71	I	测试数据输入口，器件的保护熔丝被连接到 TDI
TMS	72	I	测试方式选择，器件编程与测试的输入口
TCK	73	I	测试时钟，TCK 是用于器件测试与自动加载程序启动的时钟输入接口（FLASH 版本器件有此功能）
$\overline{RST}$/NMI	74	I	复位输入/不可屏蔽中断输入口
AV_{SS}	78		模拟电源负端，向 SVS、节电、振荡器、FLL+、比较器_A、端口 1、LCD 等电路供电
DV_{SS1}	79		数字电源参考点
AV_{CC}	80		模拟电源正端，向 SVS、节电、振荡器、FLL+、比较器_A、端口 1、LCD 等电路供电

1.4.3 MSP430F44X 系列

该系列目前有 3 个器件：

MSP430F447　　MSP430F448　　MSP430F449

1. 特　点

- 低工作电压：1.8～3.6 V。
- 超低功耗：

  活动模式　280 μA　@1 MHz、2.2 V；

  待机模式　1.1 μA；

  掉电模式(RAM 数据保持)　0.1 μA。
- 5 种节电模式。
- 从待机到唤醒的响应时间不超过 6 μs。
- 12 位 A/D 转换器带有内部参考源、采样保持、自动扫描特性。
- 16 位精简指令结构(RISC)，150 ns 指令周期。
- 带有 3 个捕获/比较寄存器的 16 位定时器有：定时器 A 与定时器 B。
- 串行通信可软件选择 UART/SPI 模式。
- 片内比较器配合其他器件可构成单斜边 A/D 转换器。
- 可编程电压监测器。
- 可在线串行编程，不需要外部编程电压。
- 驱动液晶能力可达 160 段。
- 可编程的保险熔丝保护设计者代码。
- FLASH 存储器多达 60 KB，RAM 多达 2 KB。

2. 结构框图

该系列器件的结构框图如图 1.23 所示。

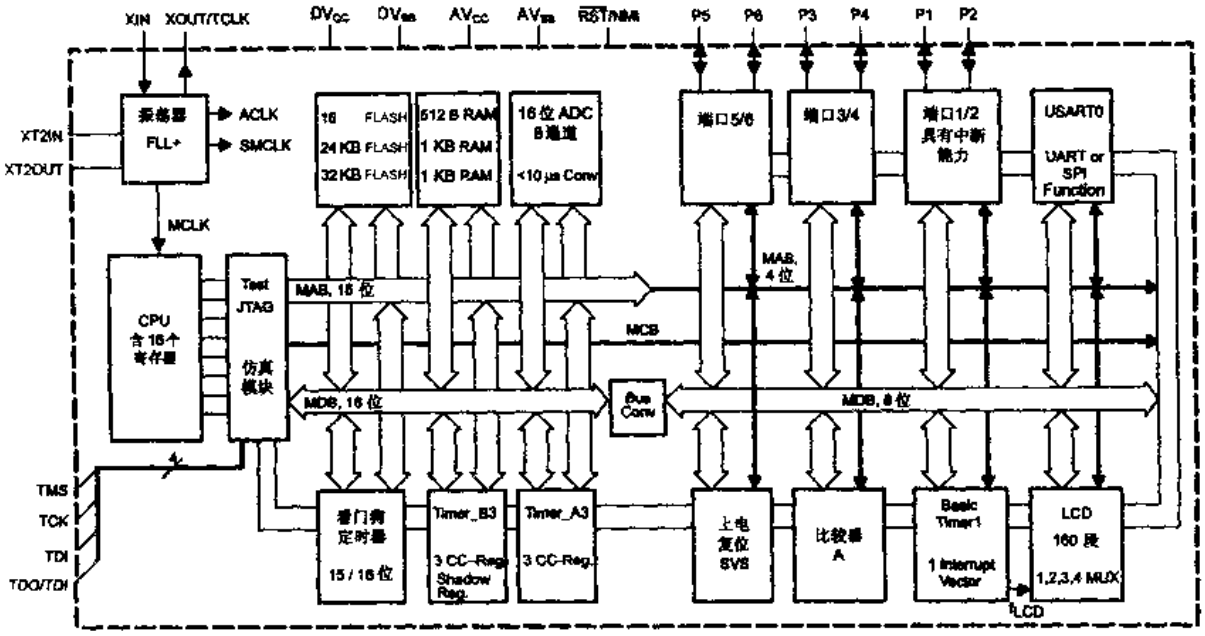


图 1.23　MSP430F44X 结构框图

续表 1.12

引脚 名称	引脚 序号	I/O	说明
P6.6/A6	5	I/O	通用数字 I/O 端口 / A/D 转换器模拟输入端
P6.7/A7/SVSIN	6	I/O	通用数字 I/O 端口 / A/D 转换器模拟输入端/SVS 输入
V_{REF+}	7	O	A/D 转换器内部参考源的输出正端
XIN	8	I	标准晶体的输入端
XOUT/TCLK	9	I/O	标准晶体输出端测试时钟输入端
Ve_{REF+}	10	I	外部参考源的正输入端
V_{REF-} Ve_{REF-}	11	I	A/D 转换器参考源负端 内部或外部
P5.1/S0	12	I/O	通用数字 I/O 端口 / 液晶驱动段输出
P5.0/S1	13	I/O	通用数字 I/O 端口 / 液晶驱动段输出
S2～S33	14～45	O	液晶驱动段输出
P4.7/S34	46	I/O	通用数字 I/O 端口 / 液晶驱动段输出
P4.6/S35	47	I/O	通用数字 I/O 端口 / 液晶驱动段输出
P4.5/UCLK1/S36	48	I/O	通用数字 I/O 端口 / 串口外部时钟输入 / 液晶驱动段输出
P4.4/SOMI1/S37	49	I/O	通用数字 I/O 端口 / 串口模式:从出主入 / 液晶驱动段输出
P4.3/SIMO1/S38	50	I/O	通用数字 I/O 端口 / 串口模式:从入主出 / 液晶驱动段输出
P4.2/STE1/S39	51	I/O	通用数字 I/O 端口 / 串口从传送使能 / 液晶驱动段输出
COM0	52	O	液晶公共输出端
P5.2/COM1	53	I/O	通用数字 I/O 端口 / 液晶公共输出端
P5.3/COM2	54	I/O	通用数字 I/O 端口 / 液晶公共输出端
P5.4/COM3	55	I/O	通用数字 I/O 端口 / 液晶公共输出端
R03	56	I	液晶电压输入端 V5
P5.5/R13	57	I/O	通用数字 I/O 端口 / 液晶电压输入端 V4/V3
P5.6/R23	58	I/O	通用数字 I/O 端口 / 液晶电压输入端 V2
P5.7/R33	59	I/O	通用数字 I/O 端口 / 液晶电压输入端 V1
DV_{CC2}	60		数字电源正端
DV_{SS2}	61		数字电源参考点
P4.1/URXD1	62	I/O	通用数字 I/O 端口 / 接收数据输入端——USART1/UART 模式
P4.0/UTXD1	63	I/O	通用数字 I/O 端口 / 发送数据输出端——USART1/UART 模式
P3.7/TB6	64	I/O	通用数字 I/O 引脚 / 定时器 B7 CCR6,捕获:CCI6A/CCI6B 输入,比较:OUT6 输出
P3.6/TB5	65	I/O	通用数字 I/O 引脚 / 定时器 B7 CCR5,捕获:CCI5A/CCI5B 输入,比较:OUT5 输出
P3.5/TB4	66	I/O	通用数字 I/O 引脚 / 定时器 B7 CCR4,捕获:CCI4A/CCI4B 输入,比较:OUT4 输出
P3.4/TB3	67	I/O	通用数字 I/O 引脚 / 定时器 B7 CCR3,捕获:CCI3A/CCI3B 输入,比较:OUT3 输出
P3.3/UCLK0	68	I/O	通用数字 I/O 引脚 / 外部时钟输入——USART0/UART 或 SPI 模式,时钟输出——USART0/SPI 模式

第 2 章　MSP430 指令系统与程序设计

MSP430 的内核 CPU 结构是按照精简指令集和高透明的宗旨来设计的，使用的指令有硬件执行的内核指令和基于现有硬件结构的高效率的仿真指令。仿真指令使用内核指令及芯片内额外配置的常数发生器 CG1 和 CG2。本章描述内核指令（硬件执行的指令）和仿真指令，并举例说明仿真指令的使用方法。在讲解指令系统之前，首先分析 MSP430 与指令系统相关的 CPU 结构和存储器系统。

2.1　MSP430 的 16 位 CPU

MSP430 系列采用的是“冯·诺依曼”结构，ROM 和 RAM 在同一地址空间，使用一组地址数据总线。中央处理单元 CPU 采用了精简的、高透明的、高效率的正交设计，它包括：一个 16 位的 ALU（算术逻辑运算单元）、16 个寄存器和一个指令控制单元。16 个寄存器中有 4 个为特殊用途，它们分别是：程序计数器、堆栈指针、状态寄存器和常数发生器。程序流程通过程序计数器控制，而程序执行的现场状态体现在程序状态字中。在表 2.1 中对 16 个寄存器作了简要说明。

表 2.1　MSP430 的 CPU 中的 16 个寄存器

简　写	功　能
R0	程序计数器 PC，指示下一条将要执行的指令的地址
R1	堆栈指针 SP，指向堆栈的栈顶
R2	状态寄存器 SR/常数发生器 CG1
R3	常数发生器 CG2
R4	通用工作寄存器 R4
…	……
R15	通用工作寄存器 R15

1. 程序计数器 PC

MSP430 的指令根据其操作数的多少，其指令长度分别为 1、2 或 3 字长。程序计数器指示出下一条即将执行的指令的地址。程序计数器 PC 的内容总是偶数，指向偶字节地址。其内容在调试程序时，可以通过寄存器窗口查看（参见第 4 章）。

2. 堆栈指针 SP

系统堆栈在系统调用子程序或进入中断服务程序时，保护程序计数器 PC。堆栈指针 SP 总是指向堆栈的顶部。系统在将数据压入堆栈时，总是先将堆栈指针 SP 的值减 2，然后再将数据送到 SP 所指的 RAM 单元。将数据从堆栈中弹出正好相反：先将数据从 SP 所指示的内存单元取出，再将 SP 的值加 2。堆栈的操作有两种情况：隐式与显式。系统对堆栈的操作为隐式，主要为自动保存 PC 的数值。在用户程序中也可对 SP 操作。下面举例说明，见图 2.1。图中，(a)表示进行堆栈操作之前的 RAM 情况；(b)表示执行 PUSH #8H 操作之后的情况；

(c)表示执行 POP R15 之后的情况。

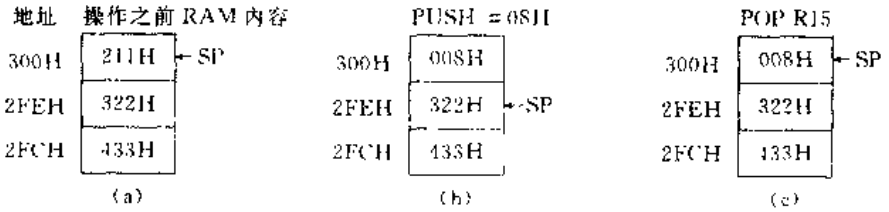


图 2.1　对堆栈的操作

3. 状态寄存器 SR

状态寄存器记录程序执行过程中的现场情况，在程序设计中有相当重要的地位。MSP430 的状态寄存器为 16 位，目前只用到前 9 位，其结构如下：

15～9	8	7	6	5	4	3	2	1	0
保留	V	SCG1	SCG0	OscOff	CPUOff	GIE	N	Z	C

位 0　C　进位标志　当运算结果产生进位时置位，否则复位。

位 1　Z　零标志　当运算结果为 0 时置位，否则复位。

位 2　N　负标志　当运算结果为负时置位，否则复位。

位 3　GIE　中断控制位　置位允许中断，复位禁止所有的中断。该位由中断复位，RETI 指令置位，也可以用指令改变。

位 4　CPUOff　CPU 控制位　置位使 CPU 进入关闭模式，此时除了 RAM 内容、端口、寄存器保持外，CPU 处于停止状态，可用所有允许的中断将 CPU 从此状态唤醒。

位 5　OscOff　晶振控制位　置位使晶体振荡器处于停止状态，CPU 从此状态唤醒；只有在 GIE 置位的情况下，由外部中断或 NMI 唤醒。要设置 OscOff=1，必须同时设置 CPUOff=1。

位 6　SCG0　此位与位 7 一起控制系统时钟发生器的 4 种活动状态。

位 7　SCG1　此位与位 6 一起控制系统时钟发生器的 4 种活动状态。

系统时钟发生器的 4 种活动状态如下所示：

SCG1	SCG0	时钟发生器的状态
0	0	SMCLK、ACLK
0	1	SMCLK、ACLK
1	0	ACLK
1	1	ACLK

位 8　V　当算术运算结果超出有符号数范围时置位。

4. 常数发生器 CG1 和 CG2

在 16 个寄存器中 R2 和 R3 为常数发生器，利用 CPU 的 27 条内核指令配合常数发生器可以生成一些简洁高效的模拟指令。表 2.2 列出了 CG1 和 CG2 可以产生的常数。

通过下面的例子，可以了解模拟指令是如何利用常数发生器的。

```
CLR   DST          ;将 DST 单元清零
```

这不是内核指令，是一条模拟指令，汇编器将 As=00，R3=0，用

```
MOV   R3,DST
```

来模拟。

表 2.2 CG1 和 CG2 可以产生的常数

寄存器	As	常 数	说 明
R2	00	-	寄存器模式
R2	01	(0)	绝对寻址模式
R2	10	00004H	+4,位处理
R2	11	00008H	+8,位处理
R3	00	0000H	0,字处理
R3	01	00001H	+1
R3	10	0002H	+2,位处理
R3	11	0FFFFH	-1,字处理

5. 通用工作寄存器

R4～R15 为通用工作寄存器。MSP430 的通用寄存器是 430 活动的大部分场所,可以执行算术逻辑运算,也可以作为临时的暂存单元;可以字操作,也可以字节操作。比如:

```
MOV     #1234H,R15      执行后 R15 内容为 1234H
MOV.B   #23H,  R15      执行后 R15 内容为 0023H
ADD.B   #34H,  R15      执行后 R15 内容为 0057H
```

MSP430 指令的寻址方式包括立即寻址、索引寻址、符号寻址和绝对寻址。这 4 种寻址方式均可用于源操作数,而索引、符号和绝对寻址方式只可用于目的操作数。源操作数和目的操作数的指令集需占用代码存储器中的 1～3 个字。

2.2 MSP430 的存储器组织

MSP430 系列的存储空间采用"冯-诺依曼"结构,ROM 和 RAM 在同一地址空间,使用一组地址数据总线。存储空间的组织又分大模式和小模式。在小模式时,总的寻址空间为 64 KB;大模式时,总的寻址空间为 1 MB。小模式时采用线性寻址空间;大模式时代码可访问 16 个 64 KB 的代码段,数据可访问的地址空间为 16 个 64 KB 的页,即为分段分页方式。图 2.2 为总的存储空间示意图。

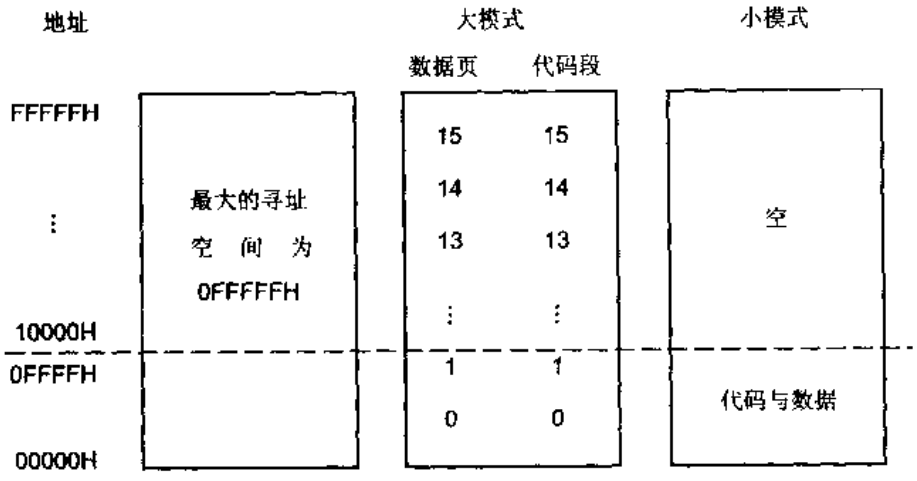


图 2.2 MSP430 总的存储器组织模式

当存储器组织为 64 KB 或更少时采用小模式，地址空间为最低的 64 KB，而目前的器件都设计成小模式，最大的程序存储空间 60 KB。在小模式中，所有的程序存储器、数据存储器、I/O 口、其他外围模块的控制器等都安排在 64 KB 空间中。现在只讨论 64 KB 存储空间的使用情况。

由于采用"冯-诺依曼"结构，ROM 和 RAM 在同一地址空间，从 00000H～0FFFFH 这一段范围内从低到高分别是：特殊功能寄存器、外围模块、数据存储器、程序存储器和中断向量表。根据不同型号其存储器的具体组织不一样。表 2.3 列举了几个常用的 MSP430 器件的存储器组织结构。

表 2.3　常用 MSP430 器件的存储器组织结构

结　构	MSP430F1121	MSP430F149	MSP430F123	MSP430C325
存储器大小 KB	4	60	8	16
中断向量地址	0FFFFH～0FFE0H	0FFFFH～0FFE0H	0FFFFH～0FFE0H	0FFFFH～0FFE0H
代码存储器地址	0F000H～0FFFFH	01100H～0FFFFH	0E000H～0FFFFH	0C000H～0FFFFH
信息存储器大小/B	256	256	256	
信息存储器地址	0EF00H～0EFFFH	01000H～010FFH	01000H～010FFH	
引导存储器大小/KB	1	1	1	
引导存储器地址	0800H～0BFFH	0C00H～0FFFH	0C00H～1000H	
数据存储器大小/B	256	2 048	256	512
RAM 地址	0200H～02FFH	0200H～09FFH	0200H～02FFH	0200H～03FFH
16 位外围模块地址	0100H～01FFH	0100H～01FFH	0100H～01FFH	0100H～01FFH
8 位外围模块地址	0010H～00FFH	0010H～00FFH	0010H～00FFH	0010H～00FFH
特殊功能寄存器地址	0000H～000FH	0000H～000FH	0000H～000FH	0000H～000FH

从表 2.3 可以看出它们的存储器结构有相同之处，也有不同之处。

相同之处在于：

- 所有器件的中断向量放在相同的地方　0FFE0H～0FFFFH；
- 所有器件的 8 位、16 位外围模块占用相同范围的存储器地址；
- 所有器件的特殊功能寄存器占用相同范围的存储器地址；
- 数据存储器开始于相同的地址，即从 0200H 处开始；
- 代码存储器的最高地址都是 0FFFFH。

不同之处在于：

- 不同型号器件的代码存储器容量不一样，从它型号（参见第 1 章）的命名规则可以看出；
- 代码存储器的起始地址不一样，每一种器件的代码存储器的起始地址为

起始地址 = 10000H － 该器件的代码容量；

- 仅 FLASH 型有信息存储器，而且不同的器件地址也不一样，但容量都是 256 B；
- 仅 FLASH 型有引导存储器，而且不同的器件地址也不一样，但容量都是 1 KB；
- 各器件数据存储器的末地址也不一样，其末地址为

末地址 = 该器件数据 RAM 容量 + 0200H；

- 中断向量的具体内容因器件不同而不同；
- 所有器件的 8 位、16 位外围模块地址范围内的具体内容因器件不同而不同。

从表 2.3 还可看出，MSP430 系列器件在存储空间的全部范围安排了：ROM、RAM 以及外围模块的寻址地址等。这些模块通过内部总线与 CPU 相连接，而且有的可以字/字节访问，有的只能字访问，有的只能字节访问。图 2.3 是片内的各种模块与数据总线的连接示意图。下面分别予以讨论。

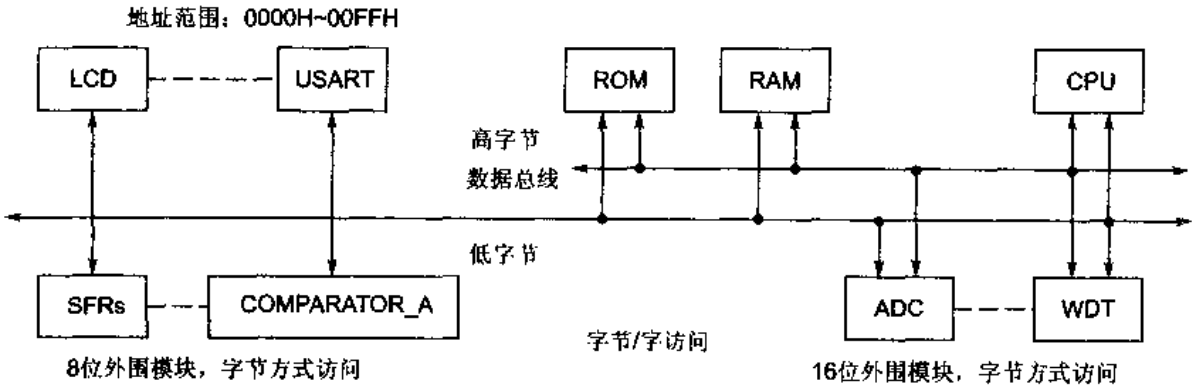


图 2.3 各种模块与总线的连接示意图

2.2.1 数据存储器 RAM

MSP430 的数据存储器位于存储器地址空间的 0200H 以上，这些存储器一般用做数据的保存与堆栈，同时也是数据运算的场所，在特殊场合还可以用做程序存储器。可以字操作，也可以字节操作，通过指令后缀加以区别。但用做程序存储器时只能字操作。字与字节操作情况如图 2.4 所示。

图 2.4 存储器中的位、字节、字

在字节操作时，每 8 位为一个操作单位；在字操作时，每两个字节为一个操作单位，而且对准偶地址操作。如：

```
MOV.B   #20H,&221H      执行后地址 221H 的内容为 20H
MOV.B   #324H,&221H     执行后地址 221H 的内容为 24H
MOV.W   #3234H,&222H    执行后地址 222H 的内容为 34H，地址 223H 的内容为 32H
MOV.W   #324H,&221H     执行后地址 221H 的内容为 03H，地址 220H 的内容为 24H
```

RAM 空间还可以进行运算，如：

```
MOV.B   #33H,&220H      执行后地址 220H 的内容为 33H
ADD.B   #22H,&220H      执行后地址 220H 的内容为 55H
MOV.B   #11H,&221H      执行后地址 221H 的内容为 11H
```

ADD	#1234H,&220H	执行后地址 220H 的内容为 89H,执行后地址 221H 的内容为 23H
RLA	&220H	执行后地址 220H 的内容为 12H，执行后地址 221H 的内容为 47H

FLASH 型的器件还有信息存储区，也可以当作数据 RAM 使用，同时它是 FLASH 型，掉电后数据不丢失，可以保存重要参数。

2.2.2　程序存储器 ROM

程序 ROM 区为 0FFFFH 以下一定数量存储空间，可存放指令代码和数据表格。程序代码必须偶地址寻址。程序代码可分 3 种情况：中断向量区、用户程序代码及系统引导程序（个别器件有，如 FLASH 型）。

中断向量区用来说明相应中断的中断服务程序首地址。某应用程序的中断向量区如图 2.5 所示。

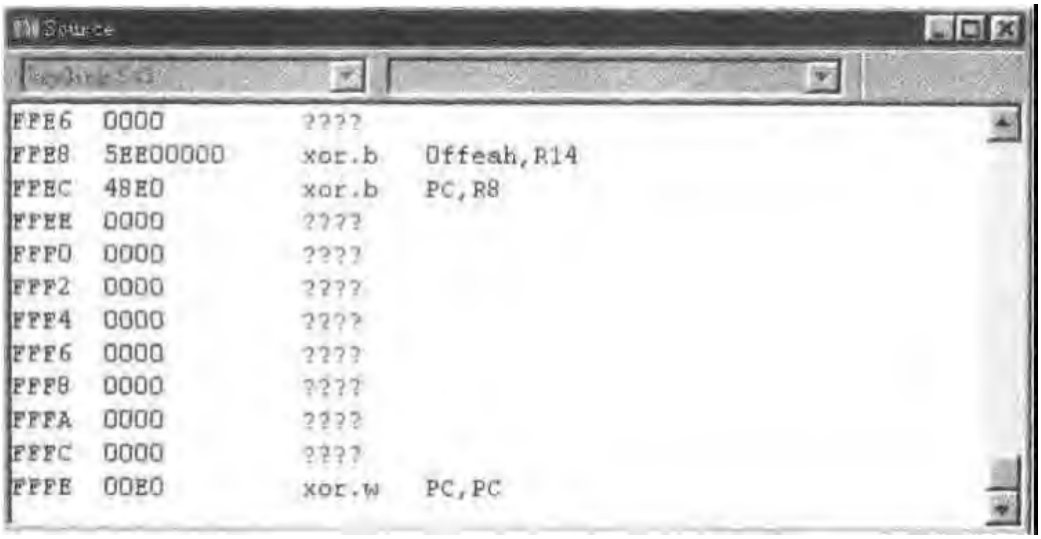


图 2.5　中断向量区

图 2.5 所示是调试环境 CSPY 中的一个窗口，可参考第 4 章。其中最左边一列 FFFE，FFFC，FFFA 等表示地址，中间一列 00E0，0000，48E0，5EE00000 等表示对应于左边地址的相应内容，最右边 xor.w　pc,pc 等表示汇编指令或相应地址中数据的反汇编指令。这里要着重理解的是第二栏数据：00E0，0000，48E0，5EE00000 等，这些数据就是中断向量，它只表示相应中断的中断服务程序首地址。还要注意这些数据的格式：00E0 实质上是 4 位十六进制数 0E000H，48E0 实质上是 0E048H，因为在地址栏（最左边）是从偶地址开始的，因此 00 是 0FFFEH 地址的内容，E0H 是 0FFFFH 地址的内容；而 5EE00000 则是因为汇编器在反汇编时将这些数据反汇编成一条指令而写在一起的，实质上它们只是 0E05EH 和 0000H 两个 4 位十六进制数据。

图 2.5 同时也说明在这个应用程序中，用户使用 3 个中断，写了 3 个中断服务程序，它们的中断服务程序的首地址分别是：E000H——系统复位或非屏蔽中断的中断服务程序的开始地址；E048H——定时器 A 中断；E05EH——P1 口中断。

不同的器件，中断向量含义不同。表 2.4～2.7 为不同器件的中断向量表。

表 2.4 MSP430X3XX,MSP430X11XX 中断向量表

中断源	中断标志	系统中断	地 址	优先级
上电、外部复位、看门狗	WDTIFG	复位	0FFFEH	15,最高
NMI、振荡器故障	NMIFG OFIFG	非屏蔽 可屏蔽	0FFFCH	14
I/O 专用	P0IFG.0	可屏蔽	0FFFAH	13
I/O 专用	P0IFG.1	可屏蔽	0FFF8H	12
比较器 A		可屏蔽	0FFF6H	11
看门狗定时器	WDTIFG	可屏蔽	0FFF4H	10
定时器 A	CCIFG0	可屏蔽	0FFF2H	9
定时器 A	TAIFG	可屏蔽	0FFF0H	8
串口接收	URXIFG	可屏蔽	0FFEEH	7
串口发送	UTXIFG	可屏蔽	0FFECH	6
ADC	ADCIFG	可屏蔽	0FFEAH	5
定时器.端口		可屏蔽	0FFE8H	4
P2	P2IFG.0~7	可屏蔽	0FFE6H	3
P1	P1IFG.0~7	可屏蔽	0FFE4H	2
基本定时器	BTIFG	可屏蔽	0FFE2H	1
P0	P0IFG.2~7	可屏蔽	0FFE0H	0,最低

表 2.5 MSP430X13X,MSP430X14X 中断向量表

中断源	中断标志	系统中断	地 址	优先级
上电、外部复位、看门狗、FLASH 存储器	WDTIFG	复位	0FFFEH	15,最高
NMI、振荡器故障、FLASH 访问出错	NMIFG、OFIFG、ACCVIFG	非屏蔽 可屏蔽	0FFFCH	14
定时器 B	BCCIFG0	可屏蔽	0FFFAH	13
定时器 B	BCCIFG1~6、TBIFG	可屏蔽	0FFF8H	12
比较器 A	CMPAIFG	可屏蔽	0FFF6H	11
看门狗定时器	WDTIFG	可屏蔽	0FFF4H	10
串口 0 接收	URXIFG0	可屏蔽	0FFF2H	9
串口 0 发送	UTXIFG0	可屏蔽	0FFF0H	8
ADC	ADCIFG	可屏蔽	0FFEEH	7
定时器 A	CCIFG0	可屏蔽	0FFECH	6
定时器 A	CCIFG1~2、TAIFG	可屏蔽	0FFEAH	5
P1	P1IFG.0~7	可屏蔽	0FFE8H	4
串口 1 接收	URXIFG1	可屏蔽	0FFE6H	3
串口 1 发送	UTXIFG1	可屏蔽	0FFE4H	2
P2	P2IFG.0~7	可屏蔽	0FFE2H	1
			0FFE0H	0,最低

表 2.6　MSP430F41X 中断向量表

中断源	中断标志	系统中断	地　址	优先级
上电、外部复位、看门狗、FLASH	WDTIFG	复位	0FFFEH	15，最高
NMI、振荡器故障、FLASH 访问出错	NMIFG，OFIFG，ACCVIFG	非屏蔽　可屏蔽	0FFFCH	14
			0FFFAH	13
			0FFF8H	12
比较器 A	CMPAIFG	可屏蔽	0FFF6H	11
看门狗定时器	WDTIFG	可屏蔽	0FFF4H	10
				9
				8
				7
定时器 A	CCIFG0	可屏蔽	0FFF2H	6
定时器 A	CCIFG1～2，TAIFG	可屏蔽	0FFFAH	5
P1	P1IFG.0～7	可屏蔽	0FFE4H	4
			0FFE6H	3
			0FFE4H	2
P2	P2IFG.0～7	可屏蔽	0FFE2H	1
基本定时器	BTIFG	可屏蔽	0FFE2H	0

表 2.7　MSP430X43X，MSP430X44X 中断向量表

中断源	中断标志	系统中断	地　址	优先级
上电、外部复位、看门狗、FLASH	WDTIFG	复位	0FFFEH	15，最高
NMI、振荡器故障、FLASH 访问出错	NMIFG，OFIFG，ACCVIFG	非屏蔽　可屏蔽	0FFFCH	14
定时器 B	BCCIFG0	可屏蔽	0FFFAH	13
定时器 B	BCCIFG1～6，TBIFG	可屏蔽	0FFF8H	12
比较器 A	CMPAIFG	可屏蔽	0FFF6H	11
看门狗定时器	WDTIFG	可屏蔽	0FFF4H	10
串口 0 接收	URXIFG0	可屏蔽	0FFF2H	9
串口 0 发送	UTXIFG0	可屏蔽	0FFF0H	8
ADC	ADCIFG	可屏蔽	0FFEEH	7
定时器 A	CCIFG0	可屏蔽	0FFF2H	6
定时器 A	CCIFG1～2，TAIFG	可屏蔽	0FFFAH	5
P1	P1IFG.0～7	可屏蔽	0FFE4H	4
串口 1 接收	URXIFG1	可屏蔽	0FFE6H	3
串口 1 发送	UTXIFG1	可屏蔽	0FFE4H	2
P2	P2IFG.0～7	可屏蔽	0FFE2H	1
基本定时器	BTIFG	可屏蔽	0FFE2H	0

程序 ROM 除了中断向量表外的其他空间都可随意用作用户程序区。

对于 FLASH 型的器件还有 1 KB 的引导 ROM(自动加载程序),这是一段出厂时已经固化的程序,为闪速存储器的读、写、擦除等操作提供环境。

2.2.3 外围模块寄存器地址

从表 2.3 中可以看出 MSP430 的外围模块的寻址被安排在 0010H～01FFH 这一区域,同时还分字寻址和字节寻址。

字模块是经全部 16 位总线相连的模块,位于存储空间 100H～1FFH。这部分空间又被分割成 16 个帧,每一帧 8 个字,一般每个字模块占用 1～3 个帧的地址空间。表 2.8 为 MSP430F149 字模块存储空间的使用情况。

表 2.8 MSP430F449 字模块的空间分割

地 址	说 明	地 址	说 明
1F0H～1FFH	保留	170H～17FH	定时器 A
1E0H～1EFH	保留	160H～16FH	定时器 A
1D0H～1DFH	保留	150H～15FH	ADC12 转换
1C0H～1CFH	保留	140H～14FH	ADC12 转换
1B0H～1BFH	保留	130H～13FH	硬件乘法器
1A0H～1AFH	ADC12 控制和中断	120H～12FH	看门狗、FLASH 控制
190H～19FH	定时器 B	110H～11FH	保留
180H～18FH	定时器 B	100H～10FH	保留

字节模块是经总线的低 8 位相连的模块,只能以字节方式来访问;而用字方式对字节模块进行操作,读时高字节将产生非预期的结果,写时高字节被忽略。字节模块占用存储空间 0000H～00FFH,一般 8 字分为一组,共 16 组。表 2.9 为 MSP430F449 字节模块的地址分配。

表 2.9 MSP430F449 字节模块的地址分配

地 址	说 明	地 址	说 明
0F0H～0FFH	保留	070H～7FH	串口 1 串口 0
0E0H～0EFH	保留	60H～6FH	保留
0D0H～0DFH	保留	50H～5FH	比较器 A、系统时钟
0C0H～0CFH	保留	40H～4FH	基本定时器
0B0H～0BFH	保留	30H～3FH	端口 6,端口 5
0A0H～0AFH	液晶模块	20H～2FH	端口 2,端口 1
090H～09FH	液晶模块	10H～1FH	端口 3,端口 4
080H～08FH	ADC12 存储控制	00H～0FH	SFR

特殊功能寄存器 SFR 处于存储空间的最低位置,位于 0000H～000FH,16 个字节,只能字节方式访问。目前只用了最前面的 6 个字节:

0000H IE1 中断允许 1

0001H IE2 中断允许 2

```
TAB    DW     13F2H,2213H,3ED4H
```

解释 此语句的目的在于将符号 TAB 所表示的数据作为地址，再将这个地址中的数据送达 R5。指令执行前后分别见图 2.8 和图 2.9。

分析 执行之前，R5=43F2H。TAB 为一个地址标号，指示在 JMP TTAW 语句之后的地址处，而 TAB 标号所指示的地址处的数据为字 13F2H。执行该语句就是将数 13F2H 送到 R5，结果如图 2.9 所示。

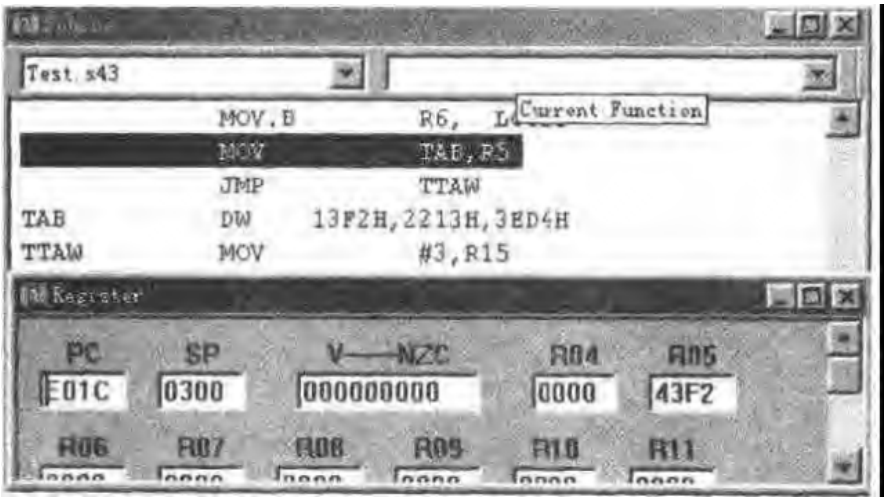


图 2.8 执行 MOV TAB，R5 指令之前

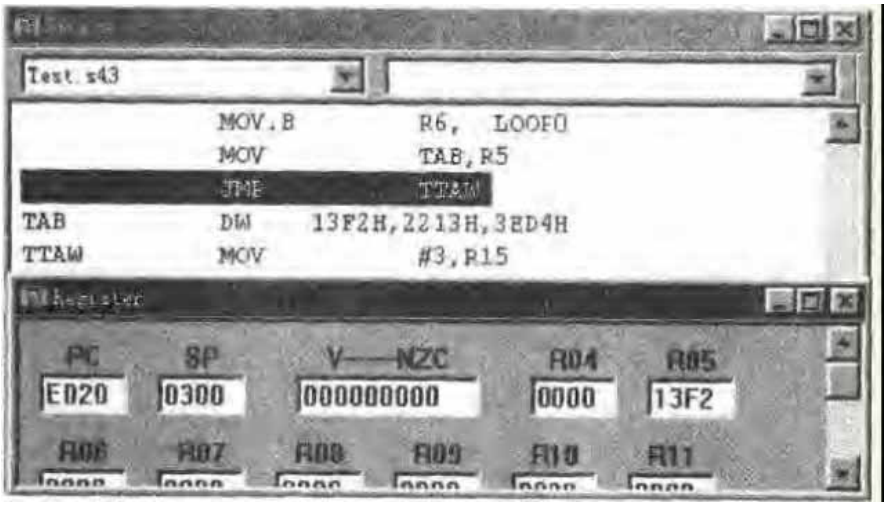


图 2.9 执行 MOV TAB，R5 指令之后

2.3.4 绝对寻址模式

汇编源程序 MOV &EDE, &TONI

这种寻址模式中的 EDE 和 TONI 即为操作数的地址。在指令代码中，紧跟操作码的一个或两个字就是操作数的地址。这种寻址模式可用于源操作数或目的操作数，也可以既是源操

作数又是目的操作数。下面的语句都属于绝对寻址模式：

```
MOV     #2345H, &RESETT      ;目的操作数绝对寻址
MOV     &RES, R15            ;源操作数绝对寻址
MOV     &220H, R15           ;源操作数绝对寻址
MOV.B   R5, &200H            ;目的操作数绝对寻址
  ……
```

［举例］

```
Reset       MOV   #2345H, R6
AAA         MOV   R6, R7
            SUB   &AAA , &Reset
```

解释　最后一句将地址 Reset 中的数据减去地址 AAA 中的数据，再将结果送达地址 Reset，指令 SUB　&AAA , &Reset 执行前后分别见图 2.10 和图 2.11。

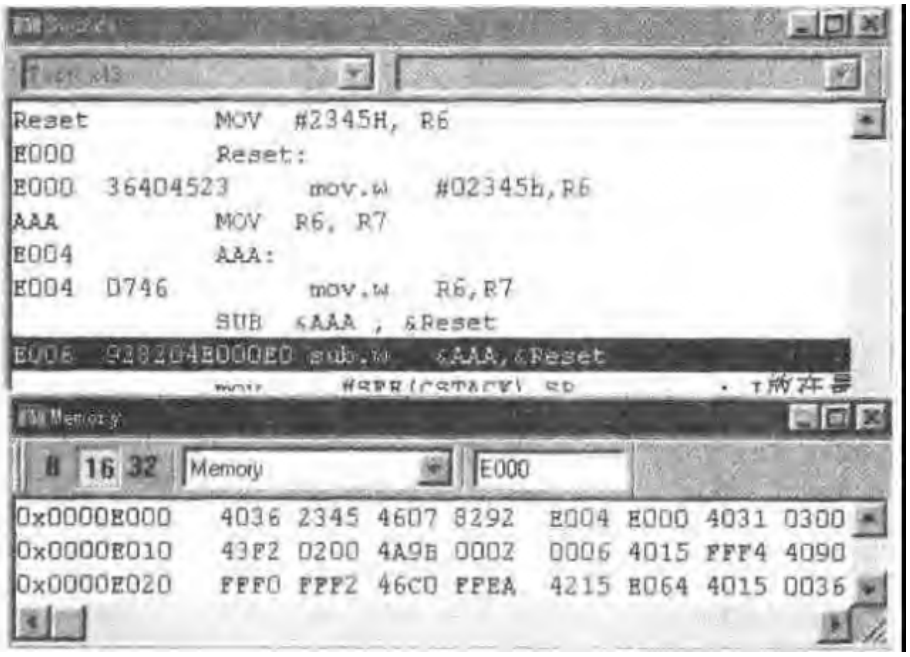


图 2.10　指令 SUB　&AAA , &Reset 执行前

分析　指令执行之前，从图 2.10 可以看出：

AAA＝0E004H；

Reset＝0E000H；

而地址 0E000H 中的数据为 4036H；

地址 0E004H 中的数据为 4607H。

所以指令执行的实质就是

$$4036H-4607H=FA2FH(C=1)$$

再将结果 FA2FH 送达 &Reset：地址 0E000H。

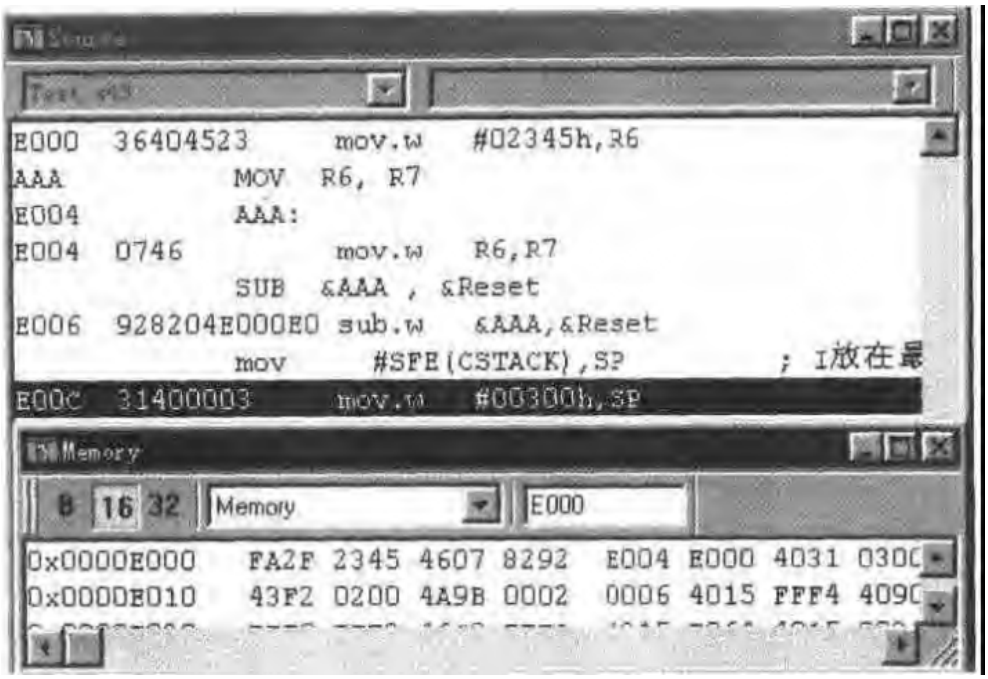


图 2.11 指令 SUB &AAA , &Reset 执行后

2.3.5 间接寻址模式

汇编源程序 MOV @ R10 , 2 (R11)

这种寻址模式将地址放在寄存器中，即寄存器中数据为所寻址数据的地址，而寄存器中的数据不改变。这种模式只对源操作数有效，对于目的操作数只能用变址寻址模式 0(Rd)替代。以下指令都是间接寻址模式：

```
MOV     @R5 ,R6
MOV     @R5, 2(R6)
ADD     @R5 ,&220H
MOV     @R5, &AAA
SUB.B   @R6 ,4 (R7)
    ……
```

[举例] SUB.B @R4, 4 (R5)

解释 以 R5 中数据加 4 为地址的数据减以 R4 为地址的数据，再将结果送达以 R5 中数据加 4 的地址，指令执行前后分别见图 2.12 和图 2.13。

分析 执行指令之前，从图 2.12 可以看出：

R4 = 220H；

R5 = 230H；

&220H = 0E34FH；

目的操作数地址 R5+4 = 234H；

目的操作数内容 &234H = 0。

指令的执行：字节减法，0－4FH＝0B1H(C＝1，有借位)。指令执行结果：&234H＝0B1H，其余(寄存器与其余 RAM 单元)不变。

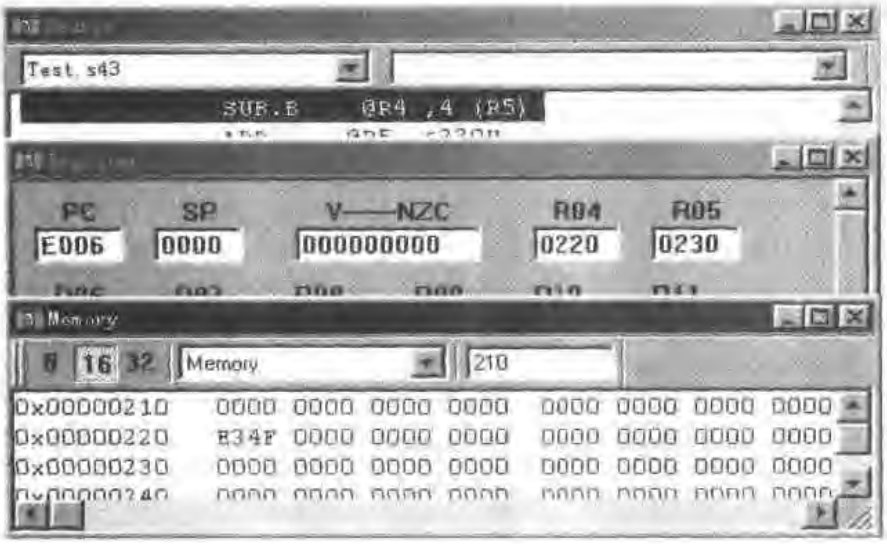


图 2.12　指令 SUB.B　@R4, 4 (R5)执行前

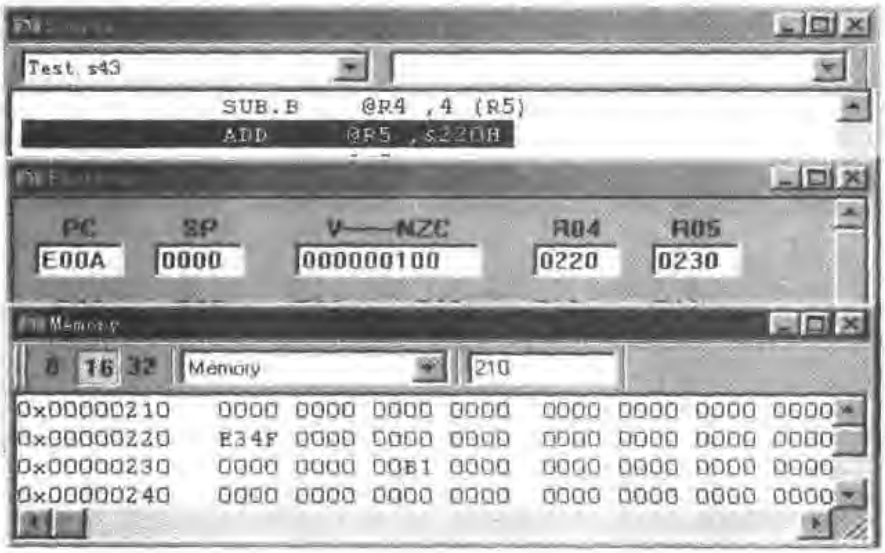


图 2.13　指令 SUB.B　@R4, 4 (R5)执行后

2.3.6　间接增量寻址模式

汇编源程序　　MOV　　@ R10 + , 0 (R11)

这种寻址模式将地址放在寄存器中，即寄存器中数据为所寻址数据的地址，而源寄存器中的数据增加 1(字节操作)或 2(字操作)。这种模式只对源操作数有效，对于目的操作数只能用变址寻址模式 0(Rd)，同时 INC / INCD Rd(手动改变目的操作数指针)替代。以下指令都是间接寻址模式：

MOV　　@R5 +,R6

```
MOV     @R5+, 2(R6)
ADD     @R5 +,&220H
MOV     @R5+, &AAA
SUB.B   @R6 +,4 (R7)
    ……
```

[举例] SUB @R4+,4 (R5)

解释 以 R5 中数据加 4 为地址的数据减以 R4 为地址的数据，再将结果送达以 R5 中数据加 4 的地址，同时 R4 自动指向下一数据（R4＋2→R4），指令执行前后分别见图 2.14 和图 2.15。

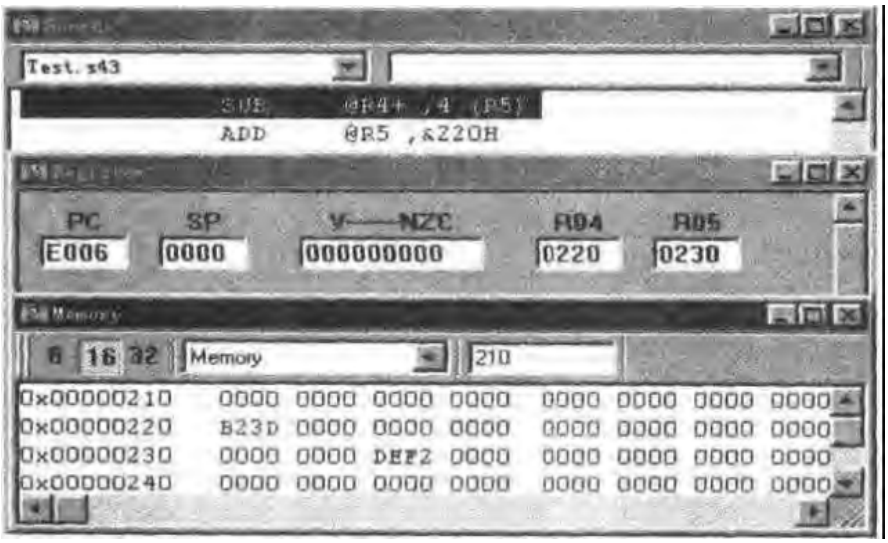


图 2.14 指令 SUB @R4+ ,4 (R5)执行前

分析 执行指令之前，从图 2.14 可以看出：

R4 = 220H；

R5 = 230H；

&220H = 0B23DH；

目的操作数地址 R5+4 = 234H；

目的操作数内容 &234H = DEF2H。

指令的执行：字减法，0DEF2H－0B23DH ＝2CB5H（C＝0，没有借位），R4 加 2。指令执行结果：&234H＝2CB5H ，R4＝222H。

间接增量寻址模式常用在需大量数据搬动或查表等情况。

在传送了第一个数后，数据指针 Rn 自动增加，指向下一个数，可以为程序设计带来很多方便。

指针 Rn 的增加是加 1 还是加 2，应根据执行的是字操作还是字节操作而定。

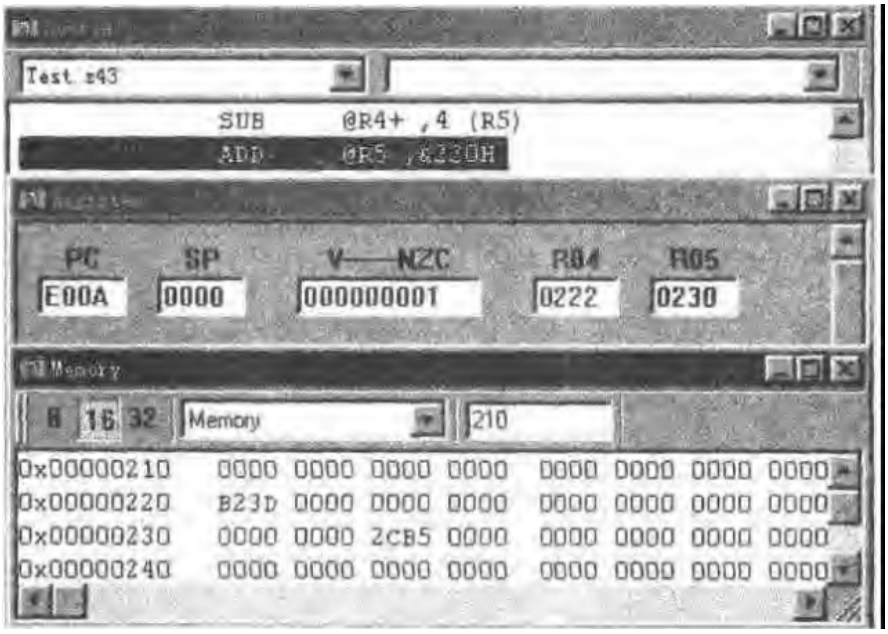


图 2.15　指令 SUB　@R4+ ,4 (R5)执行后

2.3.7　立即寻址模式

汇编源程序　MOV　#2345H，TONI

这种寻址模式的操作数能够立即得到，在指令代码中紧跟操作码后。下面列举的指令都使用了立即寻址模式：

```
MOV     #1234H，R6
MOV     #1234H，2(R6)
ADD     #1234H，&220H
MOV     #1234H，AAA
SUB.B   #12H,   &220H
```

[举例]　MOV　#1234H，R6

解释　将立即数 1234H 送达 R6，执行结果：R6 = 1234H。

2.4　指令格式

2.4.1　指令书写格式

MSP430 汇编源文件是由汇编程序指令组成的 ASCII 字符文件。它包括若干条指令语句，每一条语句使用如下格式：

[标号]　(伪)指令助记符　[操作数 1],[操作数 2]　[;注释]

- 标号汇编器将标号翻译成该行语句的物理地址，书写时最左边对齐，不必用冒号。
- 指令助记符指明该语句将执行的操作。

● 操作数 1、操作数 2 指明该语句的操作数，如果有两个操作数，则操作数 1 为源操作数，操作数 2 为目的操作数，两操作数之间用逗号间隔；如果只有一个操作数，则为目的操作数或既为目的操作数又为源操作数。

● 注释是设计者对该语句写的解释，之前用分号间隔。如：

```
START   MOV   #2345H,R15    ;R15 设置初始值
LOOP    DEC   R15           ;R15 减 1
        JNZ   LOOP          ;R15 没有减到 0 就继续减 1，直到减完
```

指令书写中的常用符号见表 2.11。

表 2.11 指令书写常用符号

符 号	功 能	符 号	功 能
Rn	R0～R15 16 个寄存器一般指 R4～R15	PC/R0	程序计数器
#	后面的数为立即数	SP/R1	堆栈指针
&	后面的数据为具体的地址	TOS	堆栈顶
@	后面数据中的内容为最终寻址地址	C	进位位
+	内容增加	N	负位
−	内容减少	V	溢出位
.W/.B	字操作/字节操作	Z	零位
dst	目的操作数	MSB	最高有效位
src	源操作数	LSB	最低有效位

从以上 MSP430 汇编指令的书写格式可以看出，它与其他汇编指令的书写格式有些不一样。

下面介绍 MSP430 的指令代码格式，它因操作数多少而不同。

2.4.2 双操作数指令(内核指令)

该指令格式使用双操作数，由 4 个域组成，共有 16 位代码：

● 操作码域——4 位　　[操作码]；

● 源域　　6 位　　[源寄存器+As]；

● 字节操作识别符——1 位　　[B/W]；

● 目的域——5 位　　[目的寄存器+Ad]。

源域由 2 个寻址位和 4 位寄存器(R0～R15)组成。

目的域由 1 个寻址位和 4 位寄存器数(R0～R15)组成。

字节操作识别符 B/W 表明指令是以一个字节(B/W=1)还是以一个字(B/W=0)的形式执行。

15～12	11～8	7	6	5	4	3～0
操作码	源寄存码	Ad	B/W	As		目的寄存码
操作码域						

表 2.12 所列为双操作数指令。

表 2.12　双操作数指令

助记符	操作数	说　明	状态位 V	N	Z	C
ADD[.W]/ADD.B	src,dst	src+dst→dst	*	*	*	*
ADDC[.W]/ADDC.B	src,dst	src+dst+C→dst	*	*	*	*
AND[.W]/AND.B	src,dst	src.AND.dst→dst	0	*	*	*
BIC[.W]/BIC.B	src,dst	.NOT.src.AND.dst→dst	-	-	-	-
BIS[.W]/BIS.B	src,dst	src.OR.dst→dst	-	-	-	-
BIT[.W]/BIT.B	src,dst	src.AND.dst	0	*	*	*
CMP[.W]/CMP.B	src,dst	dst-src	*	*	*	*
DADD[.W]/DADD.B	src,dst	src+dst+C→dst(十进制)	*	*	*	*
MOV[.W]/MOV.B	src,dst	src→dst	-	-	-	-
SUB[.W]/SUB.B	src,dst	dst+.NOT.src+1→dst	*	*	*	*
SUBC[.W]/SUBC.B	src,dst	dst+.NOT.src+C→dst	*	*	*	*
XOR[.W]/XOR.B	src,dst	src.XOR.dst→dst	*	*	*	*

注意：将状态寄存器 SR 用于目的操作数的操作会用其结果覆盖 SR 的内容；但是在以下的操作中状态位不受影响。

例如：　ADD　　#3,SR　　；操作：　(SR)+3→SR

2.4.3　单操作数指令(内核指令)

该指令格式使用单操作数，由 3 个域组成，共 16 位：

- 操作码域——9 位且 4 个 MSB 为"1H"；
- 字节操作识别符——1 位　　[B/W]；
- 目的域——6 位　　[目的寄存器+Ad]。

目的域由 2 个寻址位和 4 位寄存器数(R0～R15)组成。目的域位的位置与 2 个操作数指令的位置相同。

字节操作识别符 B/W 表明指令是以一个字节(B/W=1)还是以一个字(B/W=0)的形式执行。

15 ～ 12	11	10	9	8	7	6	5	4	3	2	1	0
0 0 0 1	×	×	×	×	×	B/W	Ad		目的寄存器			
操作码域							目的域					

表 2.13 所列为单操作数指令。

表 2.13　单操作数指令

助记符	操作数	说　明	状态位 V	N	Z	C
RRA[.W]/RRA.B	dst	MSB→MSB→…→LSB→C	0	*	*	*
RRC[.W]/RRC.B	dst	C→MSB→…→LSB→C	*	*	*	*
PUSH[.W]/PUSH.B	src	SP-2→SP,src→@SP	-	-	-	-
SWAP	dst	交换字节	-	-	-	-

续表 2.13

助记符	操作数	说 明	状态位 V N Z C
CALL	dst	PC+2→@SP,dst→SP	– – – –
TETI		从中断返回	* * * *
		TOS→SR,SP−2→SP	
		TOS→PC,SP+2→SZP	
SXT	dst	位 7→位 8→…→位 15	0 * * *

2.4.4 条件和无条件转移指令(内核指令)

(无)条件转移的指令格式包括 2 个域,共 16 位:

- 操作码域——6 位;
- 转移偏移域——10 位。

操作码由 OP-Code(3 位)和下列条件决定的 3 位组成。

15 14 13	12 11 10	9 8 7 6 5 4 3 2 1 0
0 0 1	× × ×	× × × × × × × × × ×
操作码	转移码	符号 偏 移
操作码域		转移,偏移域

条件转移使指令可转移到相对于当前地址范围在−511~+512 字之间的地址。汇编器计算出有符号的偏移,并将它们插入操作码。

表 2.14 所列为条件和无条件转移指令。

表 2.14 条件和无条件转移指令

助记符	操作数	说 明
JC/JHS	标号	进位位被置时转移到标号语句
JEQ/JZ	标号	零位被置时转移到标号语句
JGE	标号	(N. XOR. V)=0 时转移到标号语句
JL	标号	(N. XOR. V)=1 时转移到标号语句
JMP	标号	无条件转移到标号语句
JN	标号	负位被置时转移到标号语句
JNC/JLO	标号	进位位复位时转移到标号语句
JNE/JNZ	标号	零位复位时转移到标号语句

注:

1. 条件转移和无条件转移不影响状态位。
2. 已经发生的转移会用偏移改变 PC 值:PCnew=PCold+2+2×偏移;
   还未发生的转移则在上次指令处继续执行程序。

2.4.5 无需 ROM 补偿的仿真指令

表 2.15 所列指令可以无需额外的 ROM 而被精简指令集仿真。汇编器接收仿真指令的助记符,并插入适当的内核指令操作码。

表 2.15　无需 ROM 补偿的仿真指令

助记符	操作数	说　明	状态位 V	N	Z	C	仿　真
ADC[.W]	dst	进位位加至目的操作数	*	*	*	*	ADDC #0,dst
ADC.B	dst	进位位加至目的操作数	*	*	*	*	ADDC.B #0,dst
DADC[.W]	dst	十进制进位位加至目的操作数	*	*	*	*	DADD #0,dst
DADC.B	dst	十进制进位位加至目的操作数	*	*	*	*	DADD.B #0,dst
DEC[.W]	dst	目的操作数减 1	*	*	*	*	SUB #1,dst
DEC.B	dst	目的操作数减 1	*	*	*	*	SUB.B #1,dst
DECD[.W]	dst	目的操作数减 2	*	*	*	*	SUB #2,dst
DECD.B	dst	目的操作数减 2	*	*	*	*	SUB.B #2,dst
INC[.W]	dst	目的操作数增 1	*	*	*	*	ADD #1,dst
INC.B	dst	目的操作数增 1	*	*	*	*	ADD.B #1,dst
INCD[.W]	dst	目的操作数增 2	*	*	*	*	ADD #2,dst
INCD.B	dst	目的操作数增 2	*	*	*	*	ADD.B #2,dst
SBC[.W]	dst	从目的操作数中减进位位	*	*	*	*	SUBC #0,dst
SBC.B	dst	从目的操作数中减进位位	*	*	*	*	SUBC.B #0,dst
逻辑指令							
INV[.W]	dst	目的操作数求反	*	*	*	*	XOR #0FFFFH,dst
INV.B	dst	目的操作数求反	*	*	*	*	XOR.B #0FFFFH,dst
RLA[.W]	dst	算术左移	*	*	*	*	ADD dst,dst
RLA.B	dst	算术左移	*	*	*	*	ADD.B dst,dst
RLC[.W]	dst	通过进位左移	*	*	*	*	ADDC dst,dst
RLC.B	dst	通过进位左移	*	*	*	*	ADDC.B dst,dst
数据指令(共用)							
CLR[.W]		清除目的操作数	-	-	-	-	MOV #0,dst
CLR.B		清除目的操作数	-	-	-	-	MOV.B #0,dst
CLRC		清除进位位	-	-		0	BIC #1,SR
CLRN		清除负位	-	0	-	-	BIC #4,SR
CLRZ		清除零位	-	-	0	-	BIC #2,SR
POP	dst	项目从堆栈弹出	-	-	-	-	MOV @SP+,dst
SETC		置进位位	-	-	-	1	BIS #1,SR
SETN		置负位	-	1	-	-	BIS #4,SR
SETZ		置零位	-	-	1	-	BIS #2,SR
TST[.W]	dst	测试目的操作数	0	*	*	1	CMP #0,dst
TST.B	dst	测试目的操作数	0	*	*	1	CMP.B #0,dst
程序流指令							
BR	dst	转移到……	-	-	-	-	MOV dst,PC
DINT		禁止中断	-	-	-	-	BIC #8,SR
EINT		使能中断	-	-	-	-	BIS #8,SR
NOP		空操作	-	-	-	-	MOV #0H,#0H
RET		从子程序返回			-	-	MOV @SP+,PC

注：

1. 以上指令的仿真可以由 R2 和 R3 的内容仿真。
2. 寄存器 R2(CG1)包含立即数 2 和 4；寄存器 R3(CG2)包含－1 或 0FFFFH、0、＋1 和＋2，这取决于寻址位 As。
3. 汇编器根据所用的立即数设置寻址位。

续表 2.16

助记符	操作数	说　明	V	N	Z	C
* SBC[.W]/ SBC.B	dst	从目的操作数中减去进位	*	*	*	*
* SETC		置进位位	—	—	—	1
* SETN		置负位	—	1	—	—
* SETZ		置零位	—	—	1	—
SUB[.W]/ SUB.B	src,dst	dst+.NOT.src+1 →dst	*	*	*	*
SUBC[.W]/ SUBC.B	src,dst	dst+.NOT.src+C →dst	*	*	*	*
SWAP	dst	交换字节	—	—	—	—
SXT	dst	位 7→位 8→位 9→……→位 15	0	*	*	*
* TST[.W]/ TST.B	dst	测试目的操作数	0	*	*	1
XOR[.W]/ XOR.B	src,dst	src.XOR.dst→dst	*	*	*	*

注：

1. 前面带有标志(*)的指令是仿真指令。仿真指令的使用结合了 CPU 的结构和执行方法的内核指令，使得代码效率更高和速度更快。
2. ".AND."，".OR."，".NOT."和".XOR."分别表示逻辑"与"、"或"、"非"和"异或"操作。
3. "→"表示"写内容到"。
4. "src"和"dst"分别表示源操作数和目的操作数。
5. 状态位中"*"表示影响，"—"表示不影响，"0"和"1"表示清零和置位。

2.4.7　MSP430 指令的时钟周期与指令长度

MSP430 的指令执行速度(指令所用的时钟周期数，这里时钟周期指 MCLK 的周期)和指令长度(所占用存储器空间)与指令的格式和寻址模式密切相关。在不同的寻址模式下，CPU 寻找操作数的路径不一样，当然要占用不同的时间与不同的存储空间。表 2.17 和表 2.18 列出了 MSP430 指令的时钟周期数(单位为"1 个 MCLK 周期")和指令长度(单位为"字")。

表 2.17　双操作数指令的时钟周期数与指令长度表

寻址模式		周期数	指令长度/字	实　例
As	Ad			
00，Rn	0，Rm	1	1	MOV　R5,R15
	0，PC	2	1	BR　R5
00，Rn	1,x(Rm)	4	2	ADD　R5,3(R15)
	1，EDE		2	XOR　R5,EDE
	1，&EDE		2	MOV　R5,&EDE
01，x(Rn)	0，Rm	3	2	MOV　2(R5),R15
01，EDE			2	AND　EDE,R15
01，&EDE				MOV　&EDE,R15
01，x(Rn)	1，x(Rm)	6	3	ADD　3(R5),3(R15)
01，EDE	01,TONI		3	CMP　EDE,TONI
01，&EDE	1,&TONI		3	ADD　EDE,&TONI
10，@Rn	0，Rm	2	1	AND　@R5,R15
10，@Rn	1,x(Rm)		2	XOR　@R5,3(R15)
	1，EDE		2	MOV　@R5,EDE
	1,&EDE		2	XOR　@R5,&EDE
11,@Rn+	0，Rm	2	1	ADD　@R5+,R15
	0，PC	3	1	BR　@R5+

续表 2.17

寻址模式 As	寻址模式 Ad	周期数	指令长度/字	实 例	
11，#N	0，Rm	2	2	MOV	#2380H,R5
	0，PC	3	2	BR	#2AEH
11，@Rn+	1，x(Rm)	5	2	MOV	@R5+,2(R15)
11，#N	1，EDE		3	ADD	#3245H,EDE
11，@Rn+	1，@EDE		2	MOV	@R5+,&EDE
11，#N			3	ADD	#2345H,&EDE

表 2.18 单操作数指令的时钟周期数与指令长度表

寻址模式 A(s/d)	周期数 RRA RRC SWPB SXT	周期数 PUSH/CALL	指令长度/字	实 例	
00,Rn	1	3/1	1	SWPB	R5
01,x(Rn)	4	5	2	CALL	2(R5)
01,EDE	4	5	2	PUSH	EDE
01,&EDE				SXT	&EDE
10,@Rn	3	4	1	RRC	@R5
11,@Rn+	3	4/5	1	SWPB	@R5+
11,#N			2	CALL	#2344H

对于跳转类指令它们的时钟周期数与指令长度是固定的；Jxx（任意条件跳转指令）无论跳转与否都要占用 2 个时钟周期，指令长度都是 1 个字长；中断返回指令要占用 5 个时钟周期，指令长度都是 1 个字长。

除了 CPU 的指令外，CPU 还有一些操作也将占用时间，但不占用存储空间（因为不是程序指令）。它们是以下一些操作：

中断响应　　占用 6 个时钟周期；

WDT 复位　　占用 4 个时钟周期；

系统复位　　占用 4 个时钟周期。

2.5 指令集说明

指令集按功能分类，并对所有的内核指令和仿真指令（仿真指令前用星号“ * ”标注）进行说明，且附有一些简单例子；指令助记符中的后缀[. W]或无后缀表示其为字操作指令，在字操作指令中，存储器地址一定要对准偶数地址；指令助记符中的后缀[. B]表示其为字节操作符。

2.5.1 数据传送指令

MSP430 有下面一些指令可完成数据移动操作（6 条）：

- MOV[. W] /MOV . B　　src，dst

- CLR[.W]/CLR.B　　dst
- PUSH[.W]/PUSH.B　　src，dst
- POP[.W]/POP.B　　dst
- SWPB　　dst
- SXT　　dst

1. MOV

MOV　源操作数移至目的操作数。

语　法　MOV.B　src,dst　/　MOV[.W]　src,dst

操作数　src→dst

说　明　源操作数被移至目的操作数，目的操作数以前的数据丢失，源操作数不受影响。

状态位　N,Z,C,V 均不受影响。

方式位　OscOff,CPUOff 和 GIE 不受影响。

举　例　地址表 EDE(字数据)的内容被复制到表 TOM，地址表的长度应该为 020H。

```
        MOV     #EDE,R10                ;准备指针
        MOV     #020H,R9                ;准备计数器
Loop    MOV.B   @R10+,TOM-EDE-1(R10)    ;将 R10 中的指针用于两个表
        DEC     R9                      ;计数器减 1
        JNZ     Loop                    ;计数器≠0,继续复制
        ……                              ;完成复制
        ……
```

图 2.16 所示的数据传送方向是可能的。如：

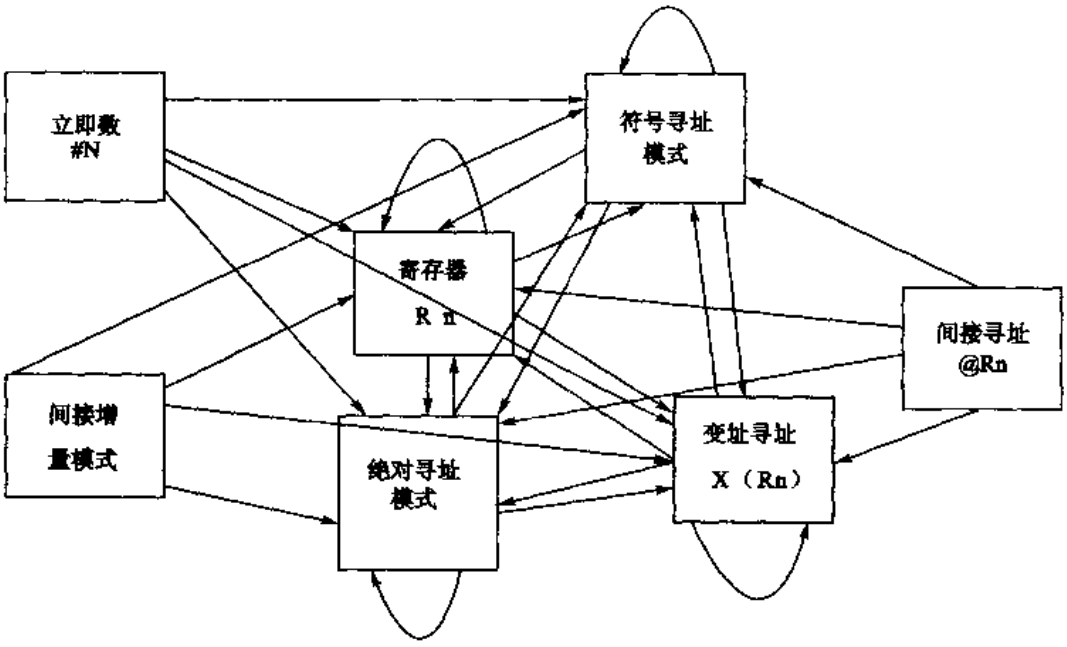


图 2.16　数据传送的可能方向

```
MOV     #234H,  R5
MOV.B   &234H,  R5
MOV     &234H,  &200H
```

src→@SP

说　明　堆栈指针减 2，然后源操作数移至由此指针(TOS)寻址的 RAM 字。

状态位　N,Z,C,V 均不受影响。

方式位　OscOff,CPUOff 和 GIE 不受影响。

举　例　外设 TCDAT 的内容保存在堆栈。

```
PUSH.B    &TCDAT    ;保存 8 位外设模块的数据
                    ;寻址 TCDAT,进栈
```

注　意　对于 PUSH 指令：系统堆栈指针通常减 2，并与字节后缀无关。这是强制性的，因为系统堆栈指针不只用于 PUSH 指令，还用于中断服务程序。

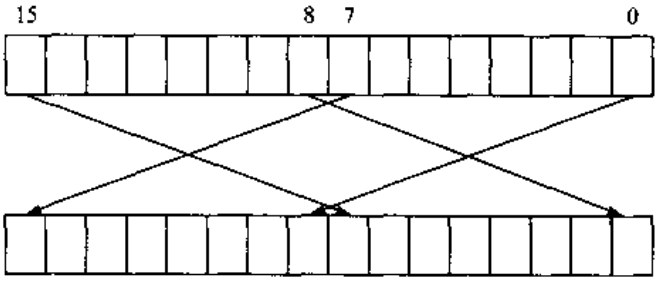
5. SWPB

SWPB　交换字节。

语　法　SWPB　dst

操作数　位 15～8→位 7～0

说　明　目的操作数的高位字节和低位字节互换(字操作)：



状态位　N,C,Z,V 均不受影响。

方式位　OscOff,CPUOff 和 GIE 不受影响。

举　例

```
MOV    #2345H,R5
SWPB   R5
```

指令执行后 R5 的内容为 4523H。

6. SXT

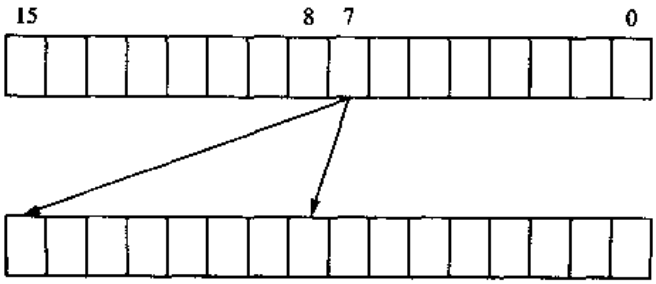
SXT　扩展符号。

语　法　SXT　dst

操作数　位 7→位 8→……→位 15

说　明　低位字节的符号扩展到高位字节(字操作指令)：

位 7 的数据送到位 8～位 15。



状态位 N:结果为负时置位,为正时复位;
Z:结果为 0 时置位,不为 0 时复位;
C:结果为非零时置位,其他情况时复位;
V:复位。

方式位 OscOff,CPUOff 和 GIE 不受影响。

举 例

① MOV #2345H,R5
SXT R5
指令执行后 R5 的内容为 0045H,因为低字节符号位为 0。

② MOV #2388FH,R5
SXT R5
指令执行后 R5 的内容为 0FF8FH,因为低字节符号位为 1。

所有 4 种可用的目的寻址方式的操作数都可用 PUSH,POP,SWPB,SXT 4 种方式进行操作。须注意 POP,PUSH 必须成对出现。

2.5.2 数据运算类指令

◆加法指令

MSP430 有下面一些指令可完成加法操作 (7 条):

- ADC[.W]/ADC.B dst
- ADD[.W]/ADD.B src,dst
- ADDC[.W]/ADDC.B src,dst
- DADC[.W]/DADC.B dst
- DADD[.W]/DADD.B dst
- INC[.W]/INC.B dst
- INCD[.W]/INCD.B dst

1. *ADC

*ADC 将进位位加至目的操作数。

语 法 ADC.B dst / ADC dst 或 ADC[.W] dst

操作数 dst + C→dst

仿 真 ADDC.B #0,dst

说 明 进位 C 加至目的操作数,操作数以前的数据丢失。

状态位 N:结果为负时置位,为正时复位;
Z:结果为 0 时置位,其他情况时复位;
C:结果产生进位时置位,其他情况时复位;
V:发生算术溢出时置位,其他情况时复位。

方式位 OscOf,CPUOff 和 GIE 不受影响。

举 例 R13 指向的 8 位数值加至 R12 指向的 16 位数值。

```
ADD.B   @R13,0(R12)   ; 加 LSDs
ADC.B   1(R12)        ; 将进位加至 MSD
```

两个 32 位数相加：加数分别放在 200H、210H 开始的地址；结果要求放在 220H 开始的地址（都是低位在前）。

分析：先加低位，再用 ADC 指令加进位到高位。

程序段：

```
ADD    &200H,&210H    ;先加低位
MOV    &210H,&220H    ;保存低位
MOV    #0,&222H
ADC    &222H          ;加低位的进位
ADD    &202H,&212H
ADD    &212H,&222H
```

2. ADD

ADD　源操作数加至目的操作数。

语　法　ADD　　src,dst　/　ADD[.W]　src,dst

操作数　src+dst→dst

说　明　源操作数加至目的操作数，源操作数不受影响，目的操作数以前的数据丢失。

状态位　N：结果位负时置位，为正时复位；

Z：结果为 0 时置位，其他情况时复位；

C：结果产生进位时置位，其他情况时复位；

V：发生算术溢出时置位，其他情况时复位。

方式位　OscOff、CPUOff 和 GIE 不受影响。

举　例　R5 加上 10 由一个进位使程序“转移”到 TONI。

```
ADD    #10,R5    ;将 10 加至 R5 的低位字节
JC     TONI      ;若(R5)≥246[0AH+0F6H],则产生进位
……               ;不产生进位
```

3. ADDC

ADDC　源操作数和进位加至目的操作数。

语　法　ADDC.B　src,dst　/　ADDC[.W]　src,dst

操作数　src+dst+C→dst

说　明　源操作数和进位 C 加至目的操作数，源操作数不受影响，目的操作数以前的数据丢失。

状态位　N：结果为负时置位，为正时复位；

Z：结果为 0 时置位，其他情况时复位；

C：结果的 MSB 产生进位时置位，其他情况时复位；

V：发生算溢出时置位，其他情况时复位。

方式位　OscOff、CPUOff 和 GIE 不受影响。

举　例　R13 指向的 32 位数值加至 R13 中的指针之上 11 个字（20/2+2/2）的 32 位计数器。

```
ADD    @R13+,20(R13)    ;无进位加 LSDs
ADDC   @R13+,20(R13)    ;带进位加 MSDs
……                      ;由 LSDs 引起
```

两个 32 位数相加：加数分别放在 200H，210H 开始的地址；结果要求放在 220H 开始的地址（都是低位在前）。

分析：先加低位，再用 ADC 指令加进位到高位。

程序段：

```
ADD    &200H,&210H        ;先加低位
MOV    &210H,&220H        ;保存低位
ADDC   &202H,&212H        ;加低位的进位
MOV    &212H,&222H
```

4. * DADC

* DADC　加上十进制的进位位。

语　法　DADC.B　dst　/　DADC[.W]　src,dst

操作数　dst+C→dst(十进制)

仿　真　DADD　#0,dst

说　明　进位位 C 作为十进制加至目的操作数。

状态位　N：MSB 为 1 时置位；

Z：dst 为 0 时置位，其他情况时复位；

C：目的操作数从 9999 增至 0000 时置位；

V：不确定。

方式位　OscOff，CPUOff 和 GIE 不受影响。

举　例　R5 中的 4 位十进制数加至 R8 指向的 8 位十进制数。

```
CLRC                      ;复位进位位
                          ;定义下一个指令的开始条件
DADD   R5,0(R8)           ;加 LSDs+C
DADC   2(R8)              ;将进位位加至 MSD
```

5. DADD

DADD　将十进制的进位位和源操作数加至目的操作数。

语　法　DADD.B　src,dst　/　DADD[.W]　src,dst

操作数　src+dst+C→dst(十进制)

说　明　源操作数和目的操作数被当作 4 个带有正符号的二～十进制(BCD)数。十进制的源操作数和进位 C 被加至目的操作数。源操作数不受影响，目的操作数以前的数据丢失。该操作对于二～十进制数是不确定的。

状态位　N：MSB 为 1 时置位，其他情况时复位；

Z：结果为 0 时置位，其他情况时复位；

C：结果大于 9999 时置位；

V：不确定。

方式位　OscOff，CPUOff 和 GIE 不受影响。

举　例　R5 和 R6 中的 8 位二～十进制数加至 R3 和 R4 中的 8 位二～十进制数(R4 和 R3 含有 MSBs)。

```
CLRC                      ;清除进位位
DADD   R5,R3              ;加 LSDs
```

② 堆栈顶部的字节增2。

```
INCD.B   0(SP)          ;TOS处的字节增2
```

◆ **减法指令**

MSP430有下面一些指令可完成减法操作（7条）：

- SUB[.W]/SUB.B src, dst
- SUBC[.W]/SUBC.B src, dst
- DEC[.W]/DEC.B src, dst
- DECD[.W]/DECD.B src, dst
- SBC[.W]/SBC.B src, dst
- CMP[.W]/CMP.B src, dst
- TST[.W]/ TST.B dst

1. SUB

SUB　从目的操作数中减去源操作数。

语　法　SUB.B　src,dst　/　SUB[.W]　src,dst

操作数　dst+.NOT.src+1→dst　或　(dst−src→dst)

说　明　从目的操作数中减去源操作数，方法是将源操作数求反再加上1。源操作数不受影响，目的操作数以前的数据丢失。

状态位　N：结果为负时置位，为正时复位。

Z：结果为0时置位，不为0时复位。

C：[]结果的MSB产生进位时置位，其他情况时复位；无借位时置为1，有借位时复位。

V：发生算术溢出时置位，其他情况时复位。

方式位　OscOff，CPUOff和GIE不受影响。

注　意　借位可视为一种非进位：

借位	进位位
是	0
否	1

举　例　①

```
MOV    #2345H, R5
SUB    #4563H, R5
```

执行结果：R5=0DDE2H，C=0（有进位）。

```
MOV    #3456H, R5
SUB    #45H, R5
```

执行结果：R5=03411H，C=1（无进位）。

②

```
MOV.B   #45H, R5
SUB.B   #63H, R5
```

执行结果：R5=E2H，C=0（有进位）。

```
MOV.B   #56H, R5
SUB.B   #45H, R5
```

执行结果：R5=011H，C=1（无进位）。

2. SUBC

SUBC　从目的操作数中减去源操作数和借位.非进位。

② 地址 LEO 处的存储器字节减 1。

```
DEC.B   LEO; MEM(LEO)减 1
;将从 EDE 开始的 255 字节存储区块移到从 TONI 开始的存储区
;地址表不能重叠:目的地址 TONI 的起始点不能位于 EDE~EDE+0FEH 的范围内
        MOV     #EDE,R6
        MOV.B   #255,LEO
L$1     MOV.B   @R6+,TONI-1-EDE-1(R6)
        DEC.B   LEO
        JNZ     L$1
```

4. * DECD

* DECD　目的操作数减 2。

语　法　DECD.B　dst　/　DECD[.W]　dst

操作数　dst−2→dst

仿　真　SUB　#2,dst

说　明　目的操作数减 2,以前的数据丢失。

状态位　N:结果为负时置位,为正时复位。

Z:dst 包含 2 时置位,其他情况时复位。

C:dst 包含 0 或 1 时复位,其他情况时置位。

V:产生算术溢出时置位,其他情况时复位;目的操作数的初始值为 08001H 或 08000H 时置位,其他情况时复位。

方式位　OscOff,CPUOff 和 GIE 不受影响。

举　例　① R10 减 2。

```
DECD    R10
;将从 EDE 开始的 255 字节存储区块移到从 TONI 开始的存储区
;地址表不能重叠:目的地址 TONI 的起点不能位于 EDE~EDE+0FEH 的范围内
        MOV     #EDE,R6
        MOV     #255,R10
L$1     MOV.B   @R6+,TONI-EDE-2(R6)
        DEC     R10
        JNZ     L$1
```

② 地址 LEO 处的存储器减 2。

```
DECD.B  LEO   ;MEM(LEO)减 2
```

状态字节 STATUS 减 2。

```
DECD.B  STATUS
```

5. * SBC

* SBC　从目的操作数减去借位。

语　法　SBC.B　dst　/　SBC[.W]　dst

操作数　dst+0FFFFH+C→dst

仿　真　SUBC　#0,dst

说　明　进位 C 加到减 1 后的目的操作数，目的操作数原来的数据丢失。

状态位　N：结果为负时置位，为正时复位；

Z：结果为 0 时置位，不为 0 时复位；

C：dst 从 0000 减至 0FFFFH 时复位，其他情况时置位；

V：初始 C=0 且 dst=08000H 时置位，其他情况时复位。

方式位　OscOff，CPUOff 和 GIE 不受影响。

注　意　借位可视为非进位 1：

借位	进位位
是	0
否	1

6. * CMP

* CMP　比较源操作数和目的操作数。

语　法　CMP . B　src，dst　/　CMP[. W]　src，dst

操作数　dst + . NOT. src + 1　或　(dst − src)

说　明　从目的操作数中减去源操作数，方法是将源操作数求反再加上 1。源操作数和目的操作数不受影响，不保存结果，只影响状态位。

状态位　N：结果为负时置位，为正时复位(src >= dst)；

Z：结果为 0 时置位，其他情况时复位(src = dst)；

V：结果的 MSB 产生进位时置位，其他情况时复位；

C：发生算术溢出时置位，其他情况时复位。

方式位　OscOff，CPUOff 和 GIE 不受影响。

举　例　① 比较 R5 和 R6，如果它们相等，程序从标号 EQUAL 处运行。

```
CMP   R5,R6      ;R5 - ? R6
JEQ   EQUAL      ;是→程序跳转
```

② 比较两个 RAM 块，如果不等，程序跳转到标号 ERROR。

```
      MOV    #NUM,R5              ;被比较的数字
L$1   CMP    &BLOCK1,&BLOCK2      ;字 1 - ? 字 2
      JNZ    ERROR                ;否→跳转到 ERROR
      DEC    R5                   ;要比较所有字吗?
      JNZ    L$1                  ;否→另一次比较
```

③ 比较被 EDE 和 TONI 寻址的 RAM 字节，如果它们相等，程序继续从标号 EQUAL 处运行。

```
CMP.B   EDE,TONI    ;MEM(EDE) = ? MEM(TONI)
JEQ     EQUAL       ;
```

④ 检查端口引脚 P0 和 P1 的两个键：如果按下键 1，程序跳转到标号 MENU1；如果按下键 2，程序跳转到 MENU2。

```
P0IN     .EQU    010H
KEY1     .EQU    01H
KEY2     .EQU    02H
CMP .B           #KEY1,&P0IN
```

```
JEQ     MENU1
CMP.B       #KEY2,&P0IN
JEQ     MENU2
```

7. * TST

* TST　测试目的操作数。

语　法　TST.B　dst　/　TST[.W]　dst

操作数　dst+0FFFFH+1

仿　真　CMP　#0,dst

说　明　比较目的操作数和零，根据结果设置状态位，目的操作数不受影响。

状态位　N：目的操作数为负时置位，为正时复位；

Z：目的操作数包含 0 时置位，其他情况时复位；

C：置位；

V：复位。

方式位　OscOff，CPUOff 和 GIE 不受影响。

2.5.3　逻辑运算与位操作类指令

◆ 逻辑运算

MSP430 有下面一些指令可完成逻辑运算(10 条)：

- AND[.W]/AND.B　src，dst
- BIC[.W]/BIC.B　src，dst
- BIS[.W]/BIS.B　src，dst
- BIT[.W]/BIT.B　src，dst
- XOR[.W]/XOR.B　src，dst
- RLA[.W]/RLA.B　dst
- RLC[.W]/RLC.B　dst
- RRA[.W]/RRA.B　dst
- RRC[.W]/RRC.B　dst
- INV[.W]/INV.B　dst

1. AND

AND　源操作数和目的操作数与。

语　法　AND.B　src,dst　/　AND[.W]　src,dst

操作数　src.AND.dst→dst

说　明　源操作数和目的操作数作逻辑与，其结果放入目的操作数中。

状态位　N：结果的 MSB 为 1 时置位，为 0 时复位；

Z：结果为 0 时置位，其他情况时复位；

C：结果不为 0 时置位，其他情况不受影响(=.NOT.Zero)；

V：复位。

方式位　OscOff，CPUOff 和 GIE 不受影响。

举　例　R5 中被置的位用做 TOM 寻址字的一个标志(#0AA55H)。如果结果为 0，程序转移到 TONI 处。

```
MOV     #0AA55H,R5          ;将标志装入 R5 寄存器
AND     R5,TOM              ; TOM 用 R5 寻址标志字
JZ        TONI              ;
……                          ;结果为非零
;
AND     #0AA55H,TOM
JZ        TONI
```

标志位#05AH 和低位字节的 TOM 逻辑与，如果结果为 0，程序转移到标号 TONI 处。

```
AND.B   #0A5H,TOM           ;用 R5 标记低位字节 TOM
JZ        TONI              ;
……                          ;结果为非零
```

2. BIC

BIC　清零目的操作数的各位。

语　法　BIC.B　src,dst　/　BIC[.W]　src,dst

操作数　.NOT.src.AND.dst→dst

说　明　求反后的源操作数和目的操作数作逻辑与，结果放入目的操作数，源操作数不受影响。

状态位　N,Z,C,V 均不受影响。

方式位　OscOff,CPUOff 和 GIE 不受影响。

举　例　① 地址 LEO 处的存储器字节减 1。

```
DEC.B   LEO; MEM(LEO)减 1。
;将从 EDE 开始的 255 字节存储区块移到从 TONI 开始的存储区
;地址表不能重叠:目的地址 TONI 的起始点不能位于 EDE~EDE+0FEH 的范围内
;
        MOV       #EDE,R6
        MOV.B     #255,LEO
L$1     MOV.B   @R6+,TONI-EDE-1(R6)
        DEC.B     LEO
        JNZ       L$1
```

RAM 字 LEO 的 6 个 MSBs 被清零。

```
BIC     #0FC00H,LEO             ;清零 MEM(LEO) 中的 6 个 MSBs
```

② 清零端口引脚 P0 和 P1。

```
P0OUT   .equ   011H             ;定义端口地址
P0      .equ   01H
P1      .equ   02H
BIC.B   #P0+P1,&P0OUT           ;置 P0 和 P1 为低电平
```

RAM 字 LEO 的 5 个 MSBs 被清零。

```
BIC.B   #0F8H,LEO               ;清零 RAM(LEO)中的 5 个 MSBs
```

3. BIS

BIS　设置目的操作数的各位。

语　法　BIS.B　src,dst　/　BIS[.W]　src,dst

操作数　src.OR.dst→dst

说　明　源操作数和目的操作数作逻辑或,结果放入目的操作数,源操作数不受影响。

状态位　N,Z,C,V 均不受影响。

方式位　OscOff,CPUOff 和 GIE 不受影响。

举　例　① RAM 字 TOM 的 6 个 LSBs 被置位。

```
BIS    #003H,TOM    ;设置 RAM 区域 TOM 中的 6 个 LSBs
```

② 启动一个 A/D 转换器。

```
ASOC    .equ    1          ;开始转换位
ACTL    .equ    114H       ;ADC 控制寄存器
BIS     #ASOC,&ACTL        ;启动 A-D 转换器
```

③ RAM 字 TOM 的 3 个 MSBs 被置位。

```
BIS.B   #0E0H,TOM    ;设置 RAM 区域 TOM 中的 3 个 MSBs
```

④ 设置端口引脚 P0 和 P1 为高电平。

```
P0OUT   .equ   011H        ;定义端口地址
P0      .equ   01H
P1      .equ   02H
BIS.B   #P0+P1,&P0OUT ;置 P0 和 P1 为高电平
```

4. BIT

BIT　测试目的操作数的各位。

语　法　BIT.B　src,dst　/　BIT[.W]　src,dst

操作码　src.AND.dst

说　明　源操作数和目的操作数作逻辑与,其结果仅影响状态位,源操作数和目的操作数不受影响。

状态位　N:结果的 MSB 为 1 时置位,为 0 时复位;

Z:结果为 0 时置位,其他情况时复位;

C:结果不为 0 时置位,其他情况时复位(=.NOT.Zero);

V:复位。

方式位　OscOff,CPUOff 和 GIE 不受影响。

举　例　① 如果 R8 中的位 9 被置位,程序转移到标号语句 TOM 处。

```
BIT     #0200H,R8      ;R8 的位 9 是否被置位
JNZ     TOM            ;是→转移到标号 TOM 处
……                     ;否→程序继续
```

② 确定哪一个 A/D 通道由 MUX 配置。

```
ACTL    .equ    114H ;ADC 控制寄存器
BIT     #4,&ACTL     ;通道 0 是否被选择
JNZ     FOD          ;是→程序转移到 FOD 处
……                   ;否→程序继续
```

③ 如果 R8 中的位 3 被置位，程序转移到 TOM 处。

```
BIT.B     #8,R8
JC        TOM
```

④

```
;串行通信的接收位 RCV 被测试，用 BIT 指令测试单个位时相当于被测试位的状态，所
;以进位还会用于后续指令；读取的信息移入 RECBUF 寄存器
;
;串行通信，LSB 在前
;××××  ××××  ××××  ××××
BIT.B     #RCV,RCCTL  ;位信息成为进位
RRC       RECBUF      ;进位→RECBUF 的 MSB
                      ;c×××  ××××
……                    ;重复前 2 个指令 8 次
                      ;cccc  cccc
                      ;↑     ↑
                      ;MSB   LSB
;串行通信 MSB 在前
BIT.B     #RCV,RCCTL  ;位信息成为进位
RRC       RECBUF      ;进位→RECBUF 的 LSB
                      ;××××  ×××c
……                    ;重复前 2 个指令 8 次
                      ;cccc  cccc
                      ;↓     ↓
                      ;LSB   MSB
```

5. XOR

XOR　源操作数和目的操作数异或。

语　法　XOR.B　src,dst　/　XOR[.W]　src,dst

操作码　src.XOR.dst→dst

说　明　源操作数和目的操作数异或，结果放入目的操作数，源操作数不受影响。

状态位　N：结果的 MSB 为 1 时置位，为 0 时复位；

Z：结果为 0 时置位，其他情况时复位；

C：结果不为 0 时置位，其他情况时复位(=.NOT.Zero)；

V：两个操作数均为负时置位。

方式位　OscOff，CPUOff 和 GIE 不受影响。

6. * INV

* INV　目的操作数求反。

语　法　INV.B　dst　/　INV[.W]　dst

操作数　.NOT.dst→dst

仿　真　XOR　#0FFFFH,dst

说　明　目的操作数取反，以前的数据丢失。

状态位　N：结果为负时置位，为正时复位；

Z:dst包含0FFFFH时置位,其他情况时复位;

C:结果不为0时置位,为0时复位(=.NOT.Zero);

V:初始目的操作数为负时置位,其他情况时复位。

方式位 OscOff,CPUOff和GIE不受影响。

举 例 ① R5的内容被取消(2的补码)。

```
MOV    #00AEH,R5      ;R5=00AEH
INV    R5             ;R5求反,R5=0FF51H
INC    R5             ;取消R5,R5=0FF52H
```

② 存储字节LEO的内容被取消。

```
MOV.B  #0AEH,LEO      ;MEM(LEO)=0AEH
INV.B  LEO            ;MEM(LEO)=051H
INC.B  LEO            ;取消MEM(LEO),MEM(LEO)=053H
```

7. * RLA

* RLA 算术左移。

语 法 RLA.B dst / RLA[.W] dst

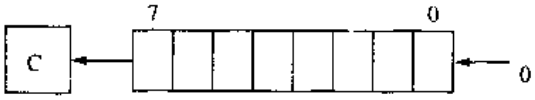
操作数 C←MSB←MSB-1←…←LSB+1←LSB←0

仿 真 ADD dst,dst

说 明 目的操作数左移一位,MSB成为进位位C,LSB填0,RLA指令可当作符号乘2。在执行该操作前,如果04000H≤dst<0C000H,则产生溢出,结果会改变符号。

字操作模式如下:

字节操作模式如下:



状态位 N:结果为负时置位,为正时复位;

Z:结果为0时置位,不为0时复位;

C:从MSB获取;

V:如产生算术溢出,即初始值为04000H≤dst<0C000H时置位,其他情况时复位。

方式位 OscOff,CPUOff和GIE不受影响。

注 意 汇编语言不识别RLA @R5+,必须将之替换成:ADD @R5+,-2(R5)。

8. * RLC

* RLC 通过进位位左移。

语 法 RLC.B dst RLC[.W] dst

操作数 C←MSB←MSB-1←…←LSB+1←LSB←C

仿 真 ADDC dst,dst

说　明　目的操作数左移一位，进位位 C 移入 LSB，MSB 移入进位位 C。

字操作模式如下：

字节操作模式如下：

状态位　N：结果为负时置位，为正时复位；

Z：结果为 0 时置位，不为 0 时复位；

C：从 MSB 获取；

V：如产生算术溢出，即初始值为 03FFFH≤dst<0C000H 时置位，其他情况时复位。

方式位　OscOff，CPUOff 和 GIE 不受影响。

注　意　汇编语言不识别 RLC　@R5+，必须将之替换成：ADDC　@R5+，-2(R5)。

9. RRA

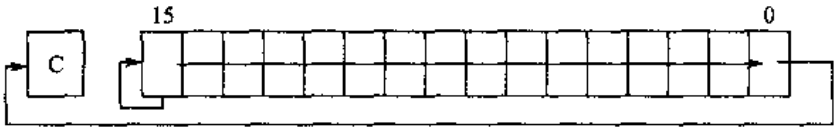
RRA　算术右移。

语　法　RRA.B　dst　　/ RRA[.W]　dst

操作数　MSB→MSB，MSB→MSB-1，MSB-1→MSB-2，…，LSB+1→LSB，LSB→C

说　明　目的操作数右移一位，MSB 移入 MSB，MSB 移入 MSB-1，LSB+1 移入 LSB。

字操作模式如下：



字节操作模式如下：

状态位　N：结果为负时置位，为正时复位；

Z：结果为 0 时置位，不为 0 时复位；

C：从 LSB 获取；

V：复位。

方式位　OscOff，CPUOff 和 GIE 不受影响。

10. RRC

RRC　通过进位位右移。

语　法　RRC.B　dst　或　RRC[.W]　dst

操作数　C→MSB→MSB-1→…→LSB+1→LSB→C

说　明　目的操作数右移一位，进位位 C 移入 MSB，LSB 移入进位位 C。

字操作模式如下：

字节操作模式如下：

状态位　N：结果为负时置位，为正时复位；

Z：结果为 0 时置位，不为 0 时复位；

C：从 LSB 获取；

V：初始目的操作数为正且初始进位位被置位时置位，其他情况时复位。

方式位　OscOff，CPUOff 和 GIE 不受影响。

◆ 位操作指令

MSP430 有下面一些指令可完成位操作(8 条)：

- CLRC
- CLRN
- CLRZ
- DINT
- EINT
- SETC
- SETZ
- SETN

1. * CLRC

* CLRC　清除进位位。

语　法　CLRC

操作数　0→C

仿　真　BIC　#1，SR

说　明　进位位被清零。该指令是一个字指令。

状态位　N，Z，V 不受影响；C 清零。

方式位　OscOff，CPUOff 和 GIE 不受影响。

举　例　R13 指向的 16 位十进制数值加至 R12 指向的 32 位数值。

```
CLRC                        ;C=0;确定开始
DADD        @R13,0(R12)     ;16 位数值加至 32 位数值的低位字
DADC        2(R12)          ;进位加至 32 位数值的高位字
```

2. * CLRN

* CLRN　清除负位。

语　法　CLRN

操作数　0→N　或　(.NOT. src. AND. dst→dst)

仿　真　BIC　　#4,SR

说　明　常数 04H 求反后(0FFFBH)和目的操作数作逻辑与,结果放入目的操作数。该指令是一个字指令。

状态位　N 复位至 0;Z、C、V 不受影响。

方式位　OscOff、CPUOff 和 GIE 不受影响。

举　例　状态寄存器中的负位被清零,这样可避免用负数调用子程序时的特殊处理。

```
CLRN
CALL        SUBR
……
……
SUBR   JN    SUBRET    ;如输入为负,无操作并返回
……
……
SUBRET   RET
```

3. * CLRZ

* CLRZ　清除零位。

语　法　CLRZ

操作数　0→Z　或　(.NOT. src. AND. dst→dst)

仿　真　BIC　　#2,SR

说　明　常数 02H 求反后(0FFFDH)和目的操作数作逻辑与,结果放入目的操作数。该指令是一个字指令。

状态位　N 不受影响;Z 复位至 0;C、V 不受影响。

方式位　OscOff,CPUOff 和 GIE 不受影响。

举　例　清除状态寄存器中的零位。

```
CLRZ
```

4. * SETC

* SETC　设置进位位。

语　法　SETC

操作数　1→C

仿　真　BIS　　#1,SR

说　明　进位位 C 被设置(常用操作)。

状态位　N,Z,V 不受影响;C 置位。

方式位　OscOff,CPUOff 和 GIE 不受影响。

5. * SETN

* SETN　设置负位。

语　法　SETN

操作数　1→N
仿　真　BIS　#4,SR
说　明　负位 N 被置位。
状态位　N 置位;Z,C,V 不受影响。
方式位　OscOff,CPUOff 和 GIE 不受影响。

6. * SETZ

* SETZ　设置零位。
语　法　SETZ
操作数　1→Z
仿　真　BIS　#2,SR
说　明　零位 Z 被置位。
状态位　Z 置位;N,C,V 不受影响。
方式位　OscOff,CPUOff 和 GIE 不受影响。

7. DINT

* DINT　禁止(一般)中断。
语　法　DINT
操作数　0→GIE　或(0FFF7H. AND. SR→SR/ . NOT. src. AND. dst→dst)
仿　真　BIC　#8,SR
说　明　禁止所有中断;常数 08H 求反并与状态寄存器 SR 作逻辑与,结果放入 SR。
状态位　N,Z,C,V 均不受影响。
方式位　OscOff 和 CPUOff 不受影响;GIE 复位。
举　例　状态寄存器中的一般中断使能位 GIE 清零,可允许非中断移动一个 32 位数值,这样可以保证在任何中断移动期间,该数值不会被修改。

```
DINT                    ;所有使用 GIE 位的中断事件被禁止
MOV  COUNTHI,R5         ;复制数值
MOV  COUNTLO,R6
EINT                    ;所有使用 GIE 位的中断事件被使能
```

注　意　在 DINT 执行期间,当中断请求生效时才能执行禁止中断指令 DINT 后面的指令。如果任何代码顺序需要保护自身不被中断,则至少应该在此代码的前一个指令执行 DINT 指令。

8. * EINT

* EINT　使能(一般)中断。
语　法　EINT
操作数　1→GIE　或(0008H. OR. SR→SR/ . NOT. src. OR. dst→dst)
仿　真　BIS　#8,SR
说　明　使能所有中断。(常数 08H 与状态寄存器 SR 作逻辑或,结果放入 SR)。
状态位　N,Z,C,V 均不受影响。
方式位　OscOff 和 CPUOff 不受影响;GIE 置位。
举　例　状态寄存器中的一般中断使能位 GIE 被置位。

```
;口 P0.2 ~P0.7 中断程序
;此系统的中断级别最低
;P0IN 是可读取所有口位的寄存器地址,P0IFG 是可锁存所有中断事件的寄存器地址
        PUSH.B      &P0IN
        BIC.B       @SP,&P0IFG      ;仅复位接收到的标记
        EINT                        ;预置保存在堆栈中的口 0 中断标记
                                    ;允许其他中断
        BIT         #Mask,@SP
        JEQ         MaskOK          ;现有的标记和标志相同:程序跳转
        ……
MaskOK  BIC         #Mask,@SP
        ……                          ;整理:在中断子程序开始时倒回到
                                    PUSH 指令
        INCD        SP              ;改正堆栈的指针
        RETI
```

注　意　任何情况下都可以执行使能中断指令 EINT 后面的指令,甚至在中断服务请求悬而未决时。

2.5.4　跳转与程序流程的控制类指令

MSP430 有下面一些指令可执行跳转与程序流程的控制（13 条）:

- BR
- CALL
- JC / JHS
- JEQ / JZ
- JGE
- JL
- JMP
- JN
- JNC / JLO
- JNE / JNZ
- RET
- RETI
- NOP

1. * BR

* BR(BRANCH)　转移至目的操作数。

语　法　BR　dst

操作数　dst→PC

仿　真　MOV　dst,PC

说　明　无条件转移到 64 K 地址空间的任一地址处,可使用所有的源寻址方式。转移指令是一个字指令。

状态位　不影响状态位。

举　例　给出所有寻址方式的示例。

```
                                    ;
BR    #EXEC                         ;转移到标号 EXEC 或直接转移(例:#0A4H)
                                    ;内核指令 MOV @PC+,PC
                                    ;
BR    EXEC                          ;转移到 EXEC 中包含的地址
                                    ;内核指令 MOV X(PC),PC
                                    ;间接寻址
                                    ;
BR    &EXEC                         ;转移到绝对地址 EXEC 中包含的地址
                                    ;内核指令 MOV X(0),PC
                                    ;间接寻址
                                    ;
BR    R5                            ;转移到 R5 中包含的地址
                                    ;内核指令 MOV R5,PC
                                    ;用 R5 间接寻址
                                    ;
BR    @R5                           ;转移到 R5 指向的字中包含的地址
                                    ;内核指令 MOV @R5,PC
                                    ;用 R5 间接寻址
                                    ;
BR    @R5+                          ;转移到 R5 指向的字中包含的地址,然后 R5 中的指针增 1。
                                    ;下一次 SW 流使用 R5 指针 - 它可通过访问 R5 指向的
                                    ;地址表中的下一个地址改变程序的执行
                                    ;内核指令 MOV @R5+,PC
                                    ;用 R5 间接寻址,然后 R5 自动增 1
                                    ;
BR    X(R5)                         ;转移到 R5+X 指向的地址中包含的地址(例:从 X 开始的
                                    ;地址表),X 可以是一个地址也可以是一个标号
                                    ;内核指令 MOV X(R5),PC
                                    ;用 R5+X 间接寻址
```

2. CALL

CALL　调用子程序。

语　法　CALL　dst

操作数　dst→tmp　　dst 被评估和存储

SP−2→SP

PC→@SP　　将 PC 更新至 TOS

tmp→PC　　将 dst 保存至 PC

说　明　调用 64 K 地址空间中任意一地址处的子程序,可使用所有的寻址方式。返回地址(后续指令的地址)存储在堆栈中,调用指令是一个字指令。

状态位　不影响状态位。

举　例　给出所有寻址方式的示例。

```
CALL    #EXEC       ;调用标号 EXEC 或直接调用(例:#0A4H)
                    ;SP-2→SP,PC+2→@SP,@PC+→PC
CALL    EXEC        ;调用 EXEC 中包含的地址
                    ;SP  2→SP,PC-2→@SP,X(PC) →PC
                    ;间接寻址
CALL    &EXEC       ;调用绝对地址 EXEC 中包含的地址
                    ;SP -2→SP,PC- 2 →@SP,X(PC)→PC
                    ;间接寻址
CALL    R5          ;调用 R5 中包含的地址
                    ;SP-2→SP,PC-2 →@SP,R5 →PC
                    ;用 R5 间接寻址
CALL    @R5         ;调用 R5 指向的字中包含的地址
                    ;SP-2→SP,PC-2 →@SP, @R5→PC
                    ;用 R5 间接寻址
CALL    @R5+        ;调用 R5 指向的字中包含的地址,然后 R5 中的指针自动
                    ;增 1。下一次 S/W 流使用 R5 指针 — 它可通过访问 R5 指
                    ;向的地址表中的下一个地址改变程序的执行
                    ;SP-2→SP,PC-2→@SP, @R5+→PC
                    ;用 R5 间接寻址,然后 R5 自动增 1
CALL    X(R5)       ;调用 R+X 指向的地址中包含的地址(例如从 X 开始的地址
                    ;表)。X 可以是一个地址或是一个标号
                    ;SP-2→SP,PC+2→@SP,X(R5)→PC
                    ;用 R5+X 间接寻址
```

下面将 CALL 与 BR 指令一起详细解释。

所有 7 种寻址方式均可适用于转移与调用指令。转移指令被内核指令

MOV　源操作数,PC

仿真。转移指令和调用指令在一个段内使用,都不使用代码段信息。下面给出转移与调用指令在每种寻址方式下的 PC(程序计数器/程序指针)内容寻址。

(1) 间接转移、间接调用

● 间接转移

```
BR      R5
MOV     R5,PC       ;内核指令
```

指令执行之前:

地　址	内　容		寄存器	
0FF16H	0××××H	PC	0FF14H	
0FF14H	0 4 5 0 0 H	R5	0FA32H	
0FF12H	0××××H			
0FA34H	0××××H			
0FA32H	0××××H			
0FA30H	0××××H			

指令执行之后:

地　址	内　容		寄存器	
0FF16H	0××××H	PC	0FA32H	
0FF14H	0 4 5 0 0 H	R5	0FA32H	
0FF12H	0××××H			
0FA34H	0××××H			
0FA32H	0××××H			
0FA30H	0××××H	PC		

● 间接调用

CALL R5

指令执行之前：

地址	内容	
0FF16H	0××××H	
0FF14H	01285H	PC
0FF12H	0××××H	
0FA34H	0××××H	
0FA32H	0××××H	
0FA30H	0××××H	
0FFCH	0××××H	SP
0FFAH	0××××H	
0FF8H	0××××H	

寄存器	
PC	0FF14H
SP	0FACH
R5	0FA32H

指令执行 CALL 后：

地址	内容	
0FF16H	0××××H	
0FF14H	01285H	
0FF12H	0××××H	
0FA34H	0××××H	
0FA32H	0××××H	PC
0FA30H	0××××H	
0FFCH	0××××H	
0FFAH	0FF16H	SP
0FF8H	0××××H	

寄存器	
PC	0FA32H
SP	0FFAH
R5	0FA32H

指令执行 RET

地址	内容	
0FF16H	0××××H	PC
0FF14H	01285H	
0FF12H	0××××H	PC
0FA34H	0××××H	
0FA32H	0××××H	
0FA30H	0××××H	
0FFCH	0××××H	SP
0FFAH	0FF16H	
0FF8H	0××××H	

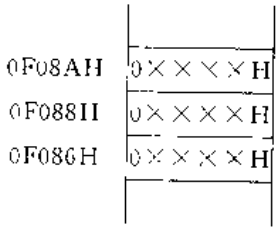
RET

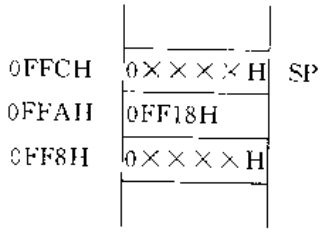
寄存器	
PC	0FF16H
SP	0FFCH
R5	0FF32H

(2) 间接索引转移、间接索引调用

● 间接索引转移

```
BR      2(R5)
MOV     2(R5),PC    ;内核指令
```



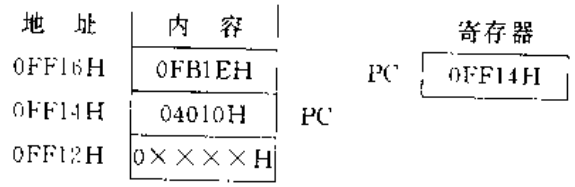


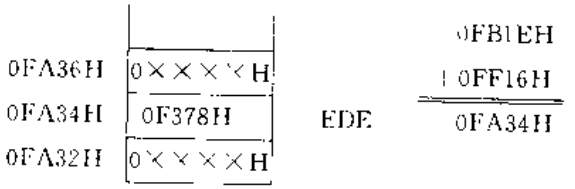
(3) 间接符号转移、间接符号调用

● 间接符号转移

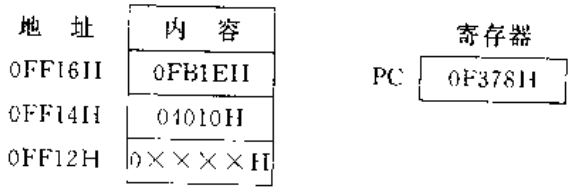
```
BR      EDE
MOV     EDE,PC      ;内核指令
```

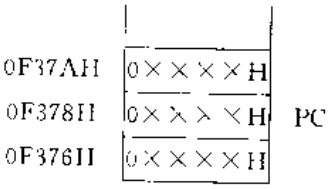
指令执行之前：





指令执行之后：

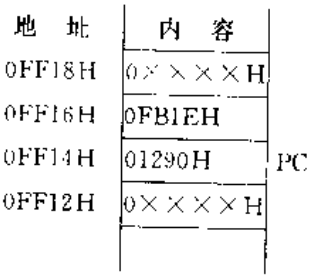


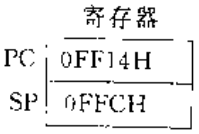


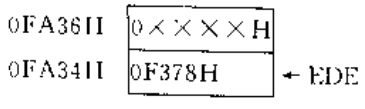
● 间接符号调用

```
CALL    EDE
```

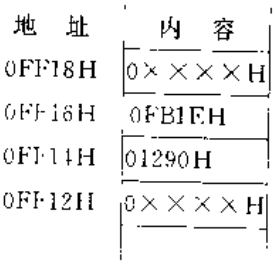
指令执行之前：

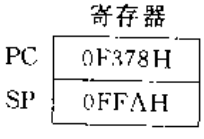


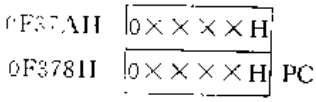




指令执行 CALL







指令执行之前：

地 址	内 容	
0FF18H	0×××××H	
0FF16H	0F378H	
0FF14H	01292H	PC
0FF12H	0×××××H	

寄存器

PC	0FF14H
SP	0FFCH

0F37AH	0×××××H	
0F378H	01234H	
0F376H	0×××××H	

0F378H
+00000H
0F378H

0FFCH	0×××××H	SP
0FFAH	0×××××H	
0FF8H	0×××××H	

指令执行 CALL

地 址	内 容	
0FF18H	0×××××H	
0FF16H	0F378H	
0FF14H	01292H	
0FF12H	0×××××H	

寄存器

PC	01234H
SP	0FFAH

01236H	0×××××H	
01234H	0×××××H	PC
01232H	0×××××H	

0FFCH	0×××××H	
0FFAH	0FF18H	SP
0FF8H	0×××××H	

指令执行 RET

地 址	内 容	
0FF18H	0×××××H	
0FF16H	0F378H	PC
0FF14H	01292H	
0FF12H	0×××××H	

寄存器

0FF18H
0FFCH

0F37AH	0×××××H	
0F378H	01234H	
0F376H	0×××××H	

0FFCH	0×××××H	SP
0FFAH	0FF18H	
0FF8H	0×××××H	

(5) 间接间接转移、间接间接调用

● 间接间接转移

```
BR      @R9
MOV     @R9.PC    ;内核指令
```

指令执行之前：

地　址	内　容
0FF16H	0××××H
0FF14H	04920H
0FF12H	0××××H

PC

寄存器	
PC	0FF14H
R9	0FA32H

地　址	内　容
0FA34H	0××××H
0FA32H	0F124H
0FA30H	0××××H

指令执行之后：

地　址	内　容
0FF16H	0××××H
0FF14H	04920H
0FF12H	0××××H

寄存器	
PC	0F124H
	0FA34H

地　址	内　容	
0F126H	0××××H	
0F124H	0××××H	PC
0F122H	0××××H	

● 间接间接调用(如 CALL @R5)与间接间接转移差不多，此处略。

(6) 可自动增 1 的间接间接转移和间接间接调用

● 可自动增 1 的间接间接转移

```
BR      @R5+
MOV     @R5+,PC ;内核指令
MOV     @R5+,PC ;内核指令
```

指令执行之前：

地　址	内　容
0FF16H	0××××H
0FF14H	04530H
0FF12H	0××××H

PC

寄存器	
PC	0FF14H
R5	0FA32H

地　址	内　容
0FA34H	0××××H
0FA32H	0F124H
0FA30H	0××××H

指令执行之后：

地　址	内　容
0FF16H	0××××H
0FF14H	04210H
0FF12H	0××××H

寄存器	
PC	0F124H
R5	0FA34H

地　址	内　容	
0F126H	0××××H	
0F124H	0××××H	PC
0F122H	0××××H	

● 可自动增 1 的间接间接调用

```
CALL    @R5+
```

指令执行之前：

地　址	内　容
0FF16H	0××××H
0FF14H	012B5H
0FF12H	0××××H

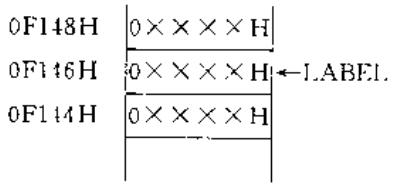
PC

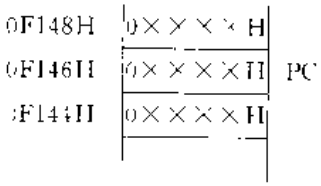
寄存器
0FF14H
0FFCH
0FA32H

指令执行之后：

地　址	内　容
0FF16H	0××××H
0FF14H	012B5H
0FF12H	0××××H

寄存器
0F124H
0FFAH
0FA34H





● 直接调用

CALL　#0F146H　或　CALL　#LABEL

指令执行之前：

地　址	内　容	
0FF18H	0××××H	
0FF16H	0F146H	
0FF14H	012B0H	PC
0FF12H	0××××H	

寄存器

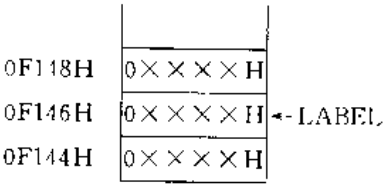
PC	0FF14H
SP	0FFCH

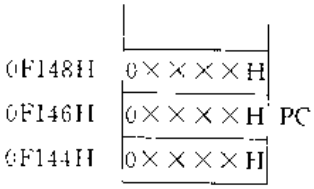
指令执行之后：

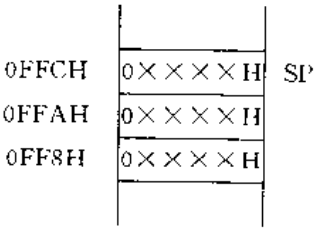
地　址	内　容	
0FF18H	0××××H	
0FF46H	0F146H	
0FF14H	012B0H	
0FF12H	0××××H	

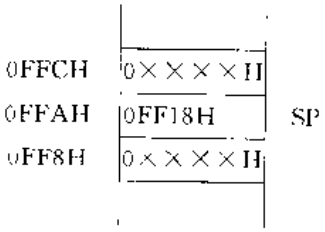
寄存器

0F146H
0FFAH







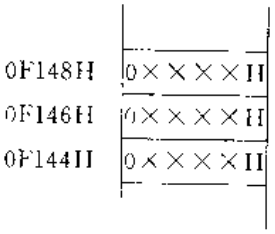


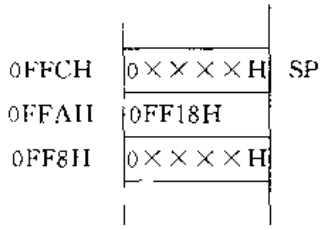
指令执行　RET

地　址	内　容	
0FF18H	0××××H	
0FF16H	0F146H	PC
0FF14H	012B0H	
0FF12H	0××××H	

寄存器

0FF18H
0FFCH





3. JC/JHS

JC　　设置进位位时程序跳转。

JHS　　大于和等于时程序跳转。

语　法　JC　　标号

　　JHS　　标号

操作数　若 C=1，PC+2×偏移→PC；

　　若 C=0，执行下一条指令。

说　明　测试状态寄存器的进位位。如果 C 被置位，则指令的 LSB 中包含的 10 位符号偏移加至程序计数器；如果 C 被复位，则执行 JUMP 后面的一条指令。JC 用于比较无符号数(0～65 536)(有进位/大于等于时跳转)。

状态位　N，Z，C，V 均不受影响。

举　例　① 比较 R5 和 15，如果大于或等于，程序转移到标号。

```
CMP    #15,R5
JHS    LABEL          ;若 R5 大于等于 15，程序跳转
……                    ;若 R5 小于 15，程序继续
```

② 输入 P0IN.1 的信号用于定义或控制程序流。

```
BIT    #10H,&P0IN     ;信号状态→进位
JC     PROGA          ;若 C=1，执行程序 A
……                    ;若 C=0，程序继续进行
```

4. JEQ/JZ

JEQ　　等于时程序跳转。

JZ　　为零时程序跳转。

语　法　JEQ　　标号

　　JZ　　标号

操作数　若 C=1，PC+2×偏移→PC；

　　若 C=0，执行下一条指令。

说　明　测试状态寄存器的零位 Z。如果 Z 被置位，则指令的 LSB 中包含的 10 位符号偏移加至程序计数器；如果 Z 被复位，则执行 JUMP 后面的一条指令。

状态位　N，Z，C，V 均不受影响。

举　例　① 若 R7 包含 0，程序跳转到地址 TOIN。

```
TST    R7             ;测试 R7
JZ     TON1           ;若为 0，程序跳转
```

② 如果 R6 等于地址表的内容，程序跳转至地址 LEO。

```
CMP    R6,Table(R5)   ;比较 R6 和 MEM 的内容(地址表 +R5)
```

```
JEQ    LEO                ;若两数相等,程序跳转
……                        ;若两数不等,程序继续执行
```

③ 若 R5 为 0,程序转移至标号。

```
TST   R5
JZ    LABEL
……
```

5. JGE

JGE　大于等于时程序跳转。

语　法　JGE　标号

操作数　若(N. XOR. V)=0,程序跳转至标号:PC+2×偏移→PC;

若(N. XOR. V)=1,执行下一条指令。

说　明　测试状态寄存器的负位 N 和溢出位 V。如果 N 和 V 均被置位或复位,则指令的 LSB 中包含的 10 位符号偏移加至程序计数器;如果 N 和 V 其中之一被复位,则执行 JUMP 后面的一条指令。

该指令允许比较符号整数。

状态位　N,Z,C,V 均不受影响。

举　例　如果 R6 的内容大于或等于 R7 指向的存储器的内容,则程序在标号 EDE 处继续。

```
CMP    @R7,R6             ; R6≥? (R7),比较符号数
JGE    EDE                ;是, R6≥R7
……                        ;否,程序继续执行
……
……
```

6. JL

JL　小于时程序跳转。

语　法　JL　标号

操作数　若(N. XOR. V)=0,程序跳转至标号:PC+2×偏移→PC;

若(N. XOR. V)=1,执行下一条指令。

说　明　测试状态寄存器的负位 N 和溢出位 V。如果 N 和 V 其中之一被置位,则指令的 LSB 中包含的 10 位符号偏移加至程序计数器;如果 N 和 V 均被置位或复位,则执行 JUMP 后面的一条指令。

该指令允许比较符号整数。

状态位　N,Z,C,V 均不受影响。

举　例　如果 R6 的内容小于 R7 指向的存储器的内容,则程序在标号 EDE 继续。

```
CMP    @R7,R6             ;R6<? (R7),比较符号数
JL     EDE                ;是,R6<(R7)
……                        ;否,继续
……
```

7. JMP

JMP　程序无条件跳转。

语　法　JMP　　标号

操作数　PC+2×偏移→PC

说　明　指令的LSB中包含的10位符号偏移加至程序计数器。

状态位　N,Z,C,V均不受影响。

举　例　该字指令在相对于现有程序计数器的-511～+512的范围内可取代BR指令。

8. JN

JN　　为负时程序跳转

语　法　JN　　标号

操作数　若N=1,PC+2×偏移→PC;

若N=0,执行下一条指令

说　明　测试状态寄存器的负位N。如果N被置位,则指令的LSB中包含的10位符号偏移加至程序计数器;如果N被复位,则执行JUMP后面的一条指令。

状态位　N,Z,C,V均不受影响。

举　例　从COUNT中减去R5中的计算结果,如果结果为负,则COUNT将被清除,且程序在另一个路径继续执行。

```
SUB    R5,COUNT          ;COUNT-R5→COUNT
JN     L$1               ;若为负,程序继续
……                       ;程序继续,COUNT≥0
……
……
……
L$1    CLR    COUNT
       ……
       ……
```

9. JNC/JLO

JNC　进位未设置时程序跳转。

JLO　小于时程序跳转。

语　法　JNC　　标号

JLO　　标号

操作数　若C=0,PC+2×偏移→PC;

若C=1,执行下一条指令。

说　明　测试状态寄存器的进位位C。如果C被复位,则指令的LSB中包含的10位符号偏移加至程序计数器;如果C被置位,则执行JUMP后面的一条指令。JNC用于比较无符号数(0～65 536)(无进位/小于时跳转)。

状态位　N,Z,C,V均不受影响。

举　例　① R6中的结果加至BUFFER,如果产生溢出,将会使用地址ERROR处的错误处理程序。

```
       ADD    R6    BUFFER      ;BUFFER+R6→BUFFER
       JNC    CONT              ;无进位,程序跳转至CONT
ERROR  ……                      ;错误处理程序开始
```

```
        ……
        ……
CONT    ……                          ;继续正常的程序流
        ……
        ……
```

② 若字节 STATUS 包含 1 或 0，程序转移至 STL2。

```
CMP.B    #2,STATUS
JLO      STL                        ;STATUS<2
……                                  ;STATUS>=2，程序继续执行
```

10. JNE/JNZ

JNE　不等时程序跳转。

JNZ　不为零时程序跳转。

语　法　JNE　标号

　　JNZ　标号

操作数　若 Z=0，PC+2×偏移→PC；

　　若 Z=1，执行下一条指令。

说　明　测试状态寄存器的零位 Z。如果 Z 被复位，则指令的 LSB 中包含的 10 位符号偏移加至程序计数器；如果 Z 被置位，则执行 JUMP 后面的一条指令。

状态位　N，Z，C，V 均不受影响。

举　例　若 R7 和 R8 不等，则程序跳转至 TONI。

```
CMP.B   R7,R8                      ;比较 R7 和 R8
JNE     TONI                       ;若不等，程序跳转
……                                 ;相等，程序继续执行
```

11. * RET

* RET　从子程序返回。

语　法　RET

操作数　@SP→PC

　　SP+2→SP

仿　真　MOV　@SP+，PC

说　明　由 CALL 指令压进栈的返回地址移至程序计数器。

　　程序在子程序调用后的代码地址处继续执行

状态位　N，Z，C，V 均不受影响。

12. RETI

RETI　从中断返回。

语　法　RETI

操作数　TOS→SR

　　SP+2→SP

　　TOS→PC

　　SP+2→SP

说　明　状态寄存器恢复到中断服务程序开始时的值，用 TOS 存储器中的值替换 SR 中

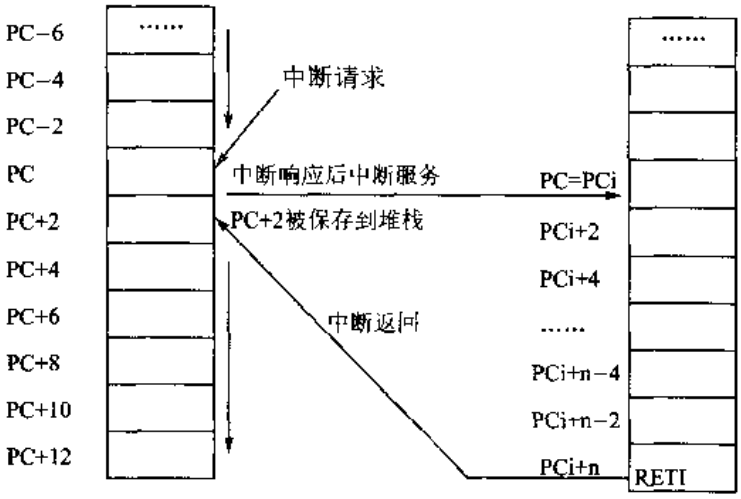
的当前值可做到这一点。堆栈指针 SP 增 2。

程序计数器恢复到中断服务程序开始的值。这是中断程序流的后续步骤。用 TOS 存储器中的值替换 PC 的当前值可实现这种恢复。堆栈指针 SP 增 2。

状态位　N,Z,C,V 均从系统堆栈恢复。

方式位　OscOff,CPUOff 和 GIE 从系统堆栈恢复。

举　例　主程序被中断。



13. * NOP

* NOP　空操作。

语　法　NOP

操作数　无

仿　真　MOV　#0,#0

说　明　不执行操作。该指令可用于在检查软件期间仿真指令或用于已确定的等待时间。

NOP 指令主要用做两个目的：

① 保持一个、两个或三个存储字；

② 调整软件时序。

注：其他指令可用于仿真空指令，例如

```
MOV     0(R4),0(R4)          ;六个周期,三个字
MOV     @R4,0(R4)            ;五个周期,两个字
BIC     #0,EDE(R4)           ;四个周期,两个字
JMP     $+2                  ;两个周期,一个字
BIC     #0,R5                ;一个周期,一个字
```

状态位　N,Z,C,V 均不受影响。

2.5.5　用多个指令仿真的宏指令

下面列出了一些被精简指令集仿真时需要更多字的指令。这些指令很少使用，但有时使用相当方便。如十进制的减法计算，需要循环移位等情况。立即数 -1,0,+1,2,4 和 8 与其他仿真指令一样，由常数产生寄存器 R2/CG1 和 R3/CG2 提供。它们是如表 2.19 所列的一些

指令。

表 2.19　用多个指令仿真的宏指令

仿真指令	指令流		说　明
ABS　dst		TST　dst	;目的操作数的绝对值
		JN　L\$0	;目的操作数为负
	L\$1	…	;目的操作数为正
		…	
		…	
	L\$0	INV　dst	;将目的操作数转换为
		INC　dst	;正目的操作数
		JMP　L\$1	
DSUB　src,dst		ADD　#6660H,src	;十进制减法
		INV　src	;源操作数被损坏
		SETC	
		DADD　src,dst	;dst←src(dec)
NEG　dst		INV　dst	;目的操作数求负
		INC　dst	
RL　dst		ADD　dst,dst	;循环左移
		ADDC　#0,dst	
RR　dst		CLRC	;循环右移
		RRC　dst	
		JNC　L\$1	
		BIS　#8000H,dst	
	L\$1	…	

2.5.6　堆栈指针寻址

在寄存器空间内安置堆栈指针可以实现许多在寄存器空间外安置指针所不可能具有的功能。

```
MOV     Rn,sp          ; 将 Rn 的内容装入 SP
MOV     @Rn,SP         ; 将 Rn 指向的字装入 SP
MOV     @Rn+,SP        ; 将 Rn 指向的字装入 SP,然后,Rn 自动增 1
MOV     X(Rn),SP       ; 将 Rn 指向的地址表内容装入 SP
                       ; X 定义相对于地址表起始处的偏移
MOV     #n,SP          ; 将常数 n 装入 SP(例如,初始化)
MOV     ADDR,SP        ; 将字 ADDR 的内容装入 SP
MOV     &ADDR,SP       ; 将字绝对地址 ADDR 的内容装入 SP
MOV     SP,Rn          ; 将 SP 复制到 Rn(例如,用于以后的恢复)
MOV     @SP,Rn         ; 将堆栈顶端(TOS)移至 Rn
MOV     @SP+,Rn        ; 将堆栈中的项弹到 Rn
MOV     X(SP),Rn       ; 将相对于 SP 的堆栈项移到 Rn
MOV     Rn,0(SP)       ; 用 Rn 的内容替换 TOS
MOV     Rn,X(SP)       ; 替换堆栈中的项。X 定义相对于 SP(TOS)的偏移
INCD    SP             ; 移动 TOS 项
```

堆栈指针可用几种方式传递变量。下例显示了一个带有变量的调用指令及在子程序内的处理。

```
            CALL    #SUBROUT
            .BYTE   MODE,CODE       ; 控制字节
            .WORD   ERRADD          ; 产生错误时的错误地址
            .WORD   ARG1            ; 参数#1
            .WORD   ARG2            ; 参数#2
……                                  ;RETURN 后继续
;
SUBROUT     ……                      ; 准备寄存器
            MOV     @SP,Rn          ; TOS 指向控制字节
            ADD     #8,0(SP)        ; 调整返回地址
            MOV     @Rn+,Rm         ; 控制字节→Rm
            MOV     @Rn+,Rx         ; 错误地址→Rx
            MOV     @Rn,Ry          ; 参数#1 →Ry
            MOV     @Rn,Rz          ; 参数#2 →Rz
            ……
            RETN                    ; 正常返回
ERROR       MOV     Rx,PC           ; 产生错误,返回地址进入 PC
```

可以用不同的方式调用同一子程序。调用指令后的变量可被子程序读取,并适当处理其信息。

2.6 汇编语言程序设计

汇编语言程序设计基本上是汇编指令的堆砌,但这并不是简单的堆砌。高级语言是模块化的设计语言,同样在使用汇编语言进行程序设计时,也要讲究使用模块化的结构。在这一部分,将讲述与汇编语言程序设计相关的问题。

2.6.1 汇编伪指令

在 2.5 小节讲述了 MSP430 的指令系统,在进行汇编程序设计时,它们是程序的主体。但是还有一些伪指令,它们提供程序数据并控制汇编过程,也是必不可少的。一般汇编器伪指令能帮助用户完成以下的事情:

- 将代码和数据汇编到规定的段中;
- 在存储器中用未初始化的变量保留空间;
- 控制汇编后列表文件的格式;
- 初始化存储器;
- 汇编条件块;
- 定义全局变量;
- 规定汇编器可以从中获得宏的库;
- 产生符号化的调试信息。

常用的汇编伪指令有以下一些:

- 模块控制伪指令有 NAME,MODULE,ENDMOD,END 等;

- 段控制伪指令有 ASEG,RSEG,STACK,COMMON,ORG,ALIGN,EVEN 等;
- 数值分配伪指令有 SET,EQU(=),DEFINE,sfrb,sfrw 等;
- 数据定义与分配伪指令有 DB,DW,DL,DF,DS 等。

下面分别予以介绍。

1. 模块控制伪指令

模块控制伪指令标志一个源程序模块的开始和结束,并给模块命名和指示其类型。它使用的助记符为:NAME,MODULE,ENDMOD,END 等。其中:

- NAME (PROGRAM)标志一个程序模块的开始;
- MODULE (LIBRARY)表示一个库模块的开始;
- ENDMOD 表示当前汇编模块的结束;
- END 表示一个汇编文件的最后模块的结束。

使用语法如下:

```
NAME symbol [(expr)]
MODULE symbol [(expr)]
ENDMOD [label]
END [label]
```

在下面的例子里定义了 3 个模块:

```
MODULE
    Module #1
    ENDMOD
    MODULE
     Module #2
    ENDMOD
    MODULE
     Last module
END
```

2. 段控制伪指令

段控制伪指令说明了代码和数据是怎样生成的。它使用如下一些助记符:

- ASEG　　一个绝对段的开始;
- RSEG　　一个可重定位段(相对段)的开始;
- STACK　　定义堆栈段;
- COMMON　　定义公共段;
- ORG　　设置特定的定位指针;
- ALIGN　　通过插入一些填充字节用以校准程序计数器;
- EVEN　　通过插入一些填充字节使程序计数器对准偶地址。

使用语法如下:

```
ASEG [start [(align)]]
RSEG segment [:type] [(align)]
STACK segment [:type] [(align)]
COMMON segment [:type] [(align)]
```

ORG expr

ALIGN align [,code]

EVEN

举例说明：

● 下面的程序段将定位子程序 subr 开始在 123 以后的地址空间。

```
 7     00007B                subr    ASEG    123(8)
 8     000100 34400A00               MOV     #10,R4
 9     000104 0485                   SUB     R5,R4
10     000106 3041                   RET
11     000108
12     000108                        END     main
```

● 下面的程序段将定义 100 字节的可重定位段作为堆栈使用。数据先使用高地址空间，后使用低地址空间。

```
        STACK   rpnstack
parms   DS      100
opers   DS      100
        END
```

● 下面定义了两个公共段用于存放变量。

```
        NAME    common1
        COMMON  data
count   DS      4
        ENDMOD

        NAME    common2
        COMMON  data
up      DS      1
        ORG     $+2
down    DS      1

        END
```

● 下面的程序段使用 ALIGN 伪指令能确保子程序在正确的地址开始。

```
 1     000000                        NAME    align
 2     000000
 3     000000 34400700       main    MOV     #7,R4
 4     000004 9012FA00               CALL    subr
 5     000008 0445                   MOV     R5,R4
 6     00000A
 7     00000A 000000000000           ALIGN   8
 8     000100 35400A00       subr    MOV     #10,R5
 9     000104 0554                   ADD     R4,R5
10     000106 3041                   RET
11     000108
12     000108                        END     main
```

```
        ADD     #value,R5

        END
```

4. 数据定义与分配伪指令

指令格式如下：

- DB　定义一个 8 位常数；
- DW　定义一个 16 位常数；
- DL　定义一个 32 位常数；
- DF　定义一个 32 位浮点常数；
- DS　分配 n 个字节单元。

使用语法如下：

```
DB      expr[,expr]
DW      expr[,expr]
DL      expr[,expr]
DF      expr[,expr]
DS      expr
```

举例说明：

- 下面一条语句在地址 table 处预留了 10 个字节空间。

```
table   DS      0xA
```

- 下面的语句段在地址为 F_TAB 处定义了一些 16 位常数。

```
F_TAB           DW    0EEFH, 0D4EH, 0BDAH
                DW    0B30H, 09F8H, 08E1H, 07E9H
```

- 经常使用的数码管显示程序中有一个显示段码表，实质上也是使用 DB 伪指令定义的一些常数表。

```
TABLED:  DB   3FH, 06H, 5BH, 4FH        ;0   1   2   3
         DB   66H, 6DH, 7DH, 07H        ;4   5   6   7
         DB   7FH, 6FH, 77H, 7CH        ;8   9   A   B
         DB   39H, 5EH, 79H, 71H        ;C   D   E   F
```

2.6.2 常用汇编程序设计方法

前面讲述的机器指令和汇编伪指令分别是机器能执行的指令和汇编器能使用的指令。程序设计实质上就是按照一定的原则、一定的思路和方法将这些指令组织起来，让 CPU 按设计者的思想执行指令，实现一定的功能，最终解决所要解决的问题。可以使用模块化的程序结构组织指令和编写程序。

程序最终要完成设计者的任务。首先，将这些任务划分为一些子任务：任务 1、任务 2、……；然后相应地使用程序模块 1、模块 2、……来完成任务 1、任务 2、……。

接下来的事情就是编写相应的模块，以完成相应的子任务，也就是具体的程序编写。用汇编语言编写程序大体上可以分为 3 个步骤：

① 确定算法，画出流程图；

② 确定数据，包括工作单元的数量、分配存放单元等；

```
MAINCALL  KEY
          BR     JMPTAB(R9)
          ……
JMPTAB    DW     JMP0
          DW     JMP1
          DW     JMP2
          DW     JMP3
          ……
JMP0      实现按键 0 功能的程序段
          ……
          JMP    MAIN
JMP1      实现按键 1 功能的程序段
          ……
          JMP    MAIN
JMP2      实现按键 2 功能的程序段
          ……
          JMP    MAIN
JMP3      实现按键 3 功能的程序段
          ……
          JMP    MAIN
          ……
```

其中每一个功能程序段都要用 JMP　MAIN 作为结束，用以构成一个主循环。

3. 循环结构

循环结构在程序设计中也同样占相当重要的地位。循环结构的使用可以使程序量缩小，节省代码空间。常用的循环结构有 WHILE 和 DO WHILE 两种形式，其流程图分别如图 2.18 和图 2.19 所示。

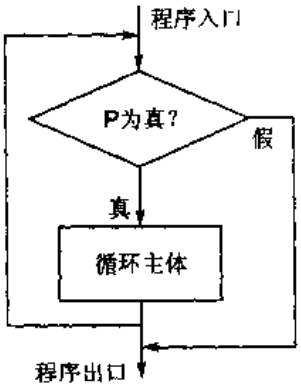


图 2.18　"WHILE"循环结构图

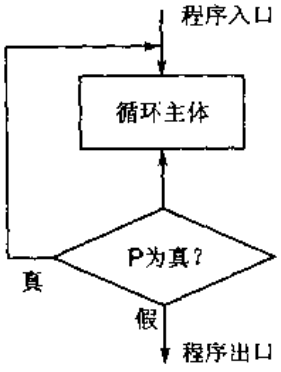


图 2.19　"DO WHILE"循环结构图

图 2.18 和图 2.19 所示的两种循环结构的差别在于前者是先判断再执行，后者是先执行再判断。在使用的时候一定要注意使用的是哪种结构，因为这涉及到与一个循环的几个要素相关的参数。

循环结构的要素：循环变量初值；循环条件是否满足；循环主体；循环变量的改变。

下面要将 200H 开始的 20 字数据移到 260H 为开始的地址空间，使用循环结构。使用 R5 为循环变量，初值为 0；循环主体为数据移动；循环条件为 R5≠20；循环变量的改变为 R5 增 1。程序如下：

```
        MOV    R5,#0                ;变量初值
LOOPMOV 200H(R5),260H(R5)           ;循环主体
        INC    R5                   ;循环变量的改变
        CMP    #20,R5               ;循环条件的判断
        JNZ    LOOP                 ;满足条件则循环
        ……                          ;不满足条件则退出
```

常用的软件延时程序也是一个典型的循环结构。下面的程序将延时 60 000 次，占用时间为 60 000×3 个时钟周期，因为减 1 和判断跳转为 3 个时钟周期。

```
        MOV     #60000,R15
LOOPDEC DEC     R15
        JNZ     LOOP
```

4. 选择结构

选择结构首先对一个条件语句进行测试，当条件为真时，执行一个方向上的流程；当条件为假时执行另一个方向上的流程。如图 2.20 所示。

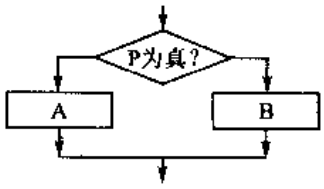


图 2.20　选择结构流程图

这种结构在程序设计时同样也是很重要的。在汇编语言中，常使用 JXX　XXXX 指令来实现。下面举例说明。

在温控系统中，当表示温度的数据大于或等于 50 时，由 P1.0 引脚输出 0，关闭加热设备；当表示温度的数据小于 50 时，由 P1.0 输出 1，控制加热设备加热。当然这只是一个简单的“开关算法”，不会有多少精度，只是说明选择结构的使用。表示温度的数据存放在 R15 中，那么 R15 是否小于 50 是判断条件。

```
        BIS.B   #1,P1DIR
        CLRC
        SUB.B   #50,R15
        JC      KAI
        BIC.B   #1,&P1OUT
        JMP     COMM
KAI     BIS.B   #1,&P1OUT
COMM    ……
        ……
```

2.7　C 语言程序设计基础

MSP430 程序设计除了可使用汇编语言外，使用 C 语言更能提高设计效率，加快开发进度。本书简要介绍如何使用 C 语言进行 MSP430 的程序设计。

2.7.1 MSP430 C 语言的数据类型

MSP430 C 编译器支持的数据类型见表 2.20。

表 2.20 MSP430 的 C 语言数据类型

数据类型	所占字节数	数据表示范围	注 释
sfrb, sfrw	1		声明字节或字长度的 I/O 类型
char (默认类型)	1	0～255	等价于 unsigned char
char (使用 -c 选项)	1	-128～127	等价于 signed char
signed char	1	-128～127	
unsigned char	1	0～255	
short, int	2	-32 768～32 767	
unsigned short unsigned int	2	0～65 535	
long	4	12 147 483 648～ 2 147 483 647	
unsigned long	4	0～4 294 967 295	
pointer	2		指针类型
float	4	18E-38～39E+38	浮点类型
double, long double	4	18E-38～39E+38	

现举例说明如下。

● sfrw 用于定义地址范围为 0x100～0x1FF 的片内外围模块的功能寄存器。看门狗控制寄存器的地址为 120H，则

```
sfrw WDTCTL = 0x120;
void func(void)
{
  WDTCTL = 0x5A08;
}
```

● sfrb 用于定义地址范围为 0x00～0xFF 的片内外围模块的功能寄存器以及特殊功能寄存器。P1 口的输出寄存器的地址为 11H，输入寄存器为 10H，则

```
sfrb P0OUT = 0x11;
sfrb P0IN = 0x10;
void func()
{
  P0OUT = 4;      /* P0.2=1  */
  P0OUT |= 4;
  P0OUT &= ~8
  if (P0IN & 2) printf("ON");
}
```

MSP430 的 C 语言指针类型分为代码指针与数据指针，都占 2 个字节长度，都能指向 0000H～0FFFFH 范围的内存单元。

浮点数为标准的 IEEE 浮点数格式，占 4 字节：

31	30 ... 23	22 ... 0
符号位 S	指数部分 EXPONENT	尾数部分 MANTISSA

所表示的数值为

$$(-1)^S \times 2^{(EXPONENT-127)} \times 1.MANTISSA$$

2.7.2　表达式语句(结构)

C 语言是一种结构化的程序设计语言。程序的最基本元素是表达式语句。所有的语句由";"隔开，或者说每一条语句后面都有一个";"。如下面的一些语句：

```
X=a+b;
Z=(x+y)-1;    I++;
……
```

除了一些运算类的表达式语句外，C 语言还提供了十分丰富的程序控制语句。程序控制语句对于实现特定的算法显得相当重要。下面讲述常用的程序控制语句。

1. 条件语句

条件语句又叫做分支语句，使用关键词 if 构成，表达条件选择的含义：如果怎样……，就怎样……，否则就怎样……。语句表达形式有 3 种：

- if(条件表达式) 语句
- if(条件表达式) 语句 1
  else 语句 2
- if(条件表达式) 语句 1
  else if(条件表达式) 语句 2
  else if(条件表达式) 语句 3
  ……

下面的程序段将在 x 和 y 中选出较大的数。

```
Char  max(char x,char y)
  {
  if(x<y)
  {return(y);}
  else {return(x);}
  }
```

2. 开关语句

开关语句是一种实现多方向条件分支的语句。虽然可用条件语句嵌套实现，但如果用开关语句则可使得程序条理分明，可读性强。开关语句由关键词 switch 构成，一般形式如下：

```
switch(表达式)
    {
    case  常量表达式 1:语句 1
                        break;
```

```
case    常量表达式 2:语句 2
                    break;
case    常量表达式 3:语句 3
                    break;
        ……
default;            语句 d
}
```

开关语句的执行过程是:将 switch 后面的表达式的值与 case 后面的常量表达式逐个比较,当遇到相等时则执行 case 后面的语句;break 语句的功能是终止当前语句的执行,使得程序能跳出 switch 语句。如果没有相等的情况,则执行语句 d。

在键盘程序中常使用开关语句。

```
Switch(key())
    case  0:key0();
          break ;
    case  1:key1();
          break ;
    case  2:key2();
          break ;
    case  3:key3();
          break ;
    ……
```

先调用键盘子程序得到按键值,然后对按键值进行比较,最后执行相应的按键程序。

3. 循环语句

循环语句给需要进行反复多次的操作提供了方便。在 C 语言中提供了 4 种循环控制语句,其构成形式如下。

● while(条件表达式) 语句

当条件满足时,就反复执行后面的语句,一直执行到条件不满足时为止。以软件延时程序为例说明该语句是如何执行的。

```
void delay(long v)
     {
     while(v! =0)v--;
     }
```

该程序段使用 while 语句,先判断 v 的值是否为 0,当不为 0 时执行其后的语句;当 v=0 时,退出循环。在循环体中同样也要条件,以使得能够退出循环。

● do 语句　while(条件表达式)

先执行一次循环体的语句,再判断条件是否满足,以决定是否再执行循环体。下面的程序将数组 BUFF[20]中的全部数据相加。

```
Int  x=0;
Char I=0;
Do{
```

```
        x = BUFF[I] + x;
        I=I+1;
        }
    while(I<20);
    ……
```

● for([初值设定表达式];[循环条件表达式];[条件更新表达式])语句

for 语句常用于需固定循环次数的循环。下面的程序段同样实现将数组 BUFF[20]中的全部数据相加的功能。

```
Int   x   0;
Char I=0;
for(I=0;I<20;I++)
    x = BUFF[I] + x;
……
```

● goto　　语句标号

goto 语句常用于跳转到一个固定的地址标号。其中固定的地址标号是一个带":"的标志符。如：

```
        ……
MM: ……
        ……
        goto   MMM
        ……
```

4. 返回语句

return(表达式);

该语句主要用于函数的返回参数。"表达式"为返回值。

2.7.3　函数的定义与调用

在 C 语言中函数是基本模块，一个 C 语言程序是由若干个函数(至少一个，是主函数)构成的。但只有一个主函数 main()，同时 C 程序都是由主函数 main()开始，它是程序的起点。使用函数可大大提高编程效率。函数有两种：编译系统提供的标准库函数和用户自定义函数。标准库函数可直接调用，而用户自定义函数须自己编写或定义之后才能调用。函数定义的一般形式为：

```
函数类型　函数名(形式参数表)
    形式参数说明
    {
    局部变量定义
    函数体语句
    }
```

其中：函数类型说明了自定义函数返回值的类型；

函数名是自定义函数的名字；

形式参数表中列出了在主调用函数与被调用函数之间传递数据的形式参数，形式参数的

类型必须加以说明，主调用函数可以是主函数，也可以是其他函数；

局部变量定义将定义在函数内部使用的局部变量；

函数体语句是为了完成该函数功能而写的各种语句的总和。

上述是一般函数的定义，在 MSP430 系统中还经常使用中断函数，中断函数的定义在形式上有些不同，下面是中断函数定义的格式：

[存储变量类型] interrupt [中断矢量变量] 函数类型 函数名（形式参数表）

形式参数说明

{

局部变量定义

函数体语句

}

格式中：interrupt 说明了该函数是中断服务函数；

[中断矢量变量]说明了该中断服务函数在中断向量表中的中断地址；

其他与一般函数的定义相同。

● 下面的函数是经常使用的延时函数。

```
void delay(long v)
{
while(v! =0)v--;
}
```

其中：void 定义该函数没有返回参数；v 是由调用函数传递进来的形式参数。

● 下面的函数计算了一个整数的正整数次幂。

```
int  mizhi(char  x,  char n)
    {
    int i,p;
    p=1;
    for(i=1; i<=n; ++i)
    p=p*x;
    return(p);
    }
```

其中：第一个 int 定义了整个函数将返回一个整数类型的值，这个值将传递给调用函数；x 与 n 为调用函数传递过来的形式参数。

中断服务函数的定义要注意的其他问题：主函数的设置要能使满足中断条件时响应中断，否则中断函数的编写毫无意义。下面是一个利用定时器中断实现在 P1.0 端口输出方波的完整程序。

```
#include <msp430x11x1.h>
void main(void)
{
  TACTL = TASSEL1 + TACLR;              // 设置定时器 A
  CCTL0 = CCIE;                         // CCR0 中断使能
  CCR0 = 20000;
  P1DIR |= 0x01;                        // P1.0 为输出口
```

```
  TACTL |= MC0;                          // 以增计数模式开始 Timer_A
  _EINT();                               // 总的中断使能
   for (;;)
  {
    _BIS_SR(CPUOFF);                     // 关 CPU
    _NOP();                              //
  }
}
interrupt[TIMERA0_VECTOR] void Timer_A (void)// 定义定时器 A 中断函数
{
  P1OUT ^= 0x01;                         // P1.0 求反
}
```

在这个程序中，主函数就是设置了能够使定时器 A 进入中断的一些参数，然后休眠。interrupt 表明是一个中断服务函数；TIMERA0_VECTOR 声明了该函数的入口地址，在 MSP430X11X1.h 文件中可以找到对 TIMERA0_VECTOR 的说明：

```
#define TIMERA0_VECTOR(9 * 2) /* 0xFFF2 Timer_A CC0 */;
```

也就是说中断入口地址为 0xFFE0＋18。

在 C 语言中函数须先声明或定义再调用。为了保险起见，建议读者最好在程序的开始先对将要用到的函数进行声明。如果调用了一个没有声明或定义的函数，将会导致编译报错；同样如果先调用，再定义函数也会导致编译报错。

2.7.4　MSP430 C 语言标准库函数

MSP430 C 语言编译环境提供了大量的标准库函数。要使用这些标准库函数，非常简单，只要在程序的开始声明要使用的库函数所在的头文件，之后在程序中就可以直接调用了。头文件的声明使用　#include　"****.h" 语法即可。

常用的头文件有以下一些，这里对其中的函数只作简要介绍，详细使用情况可以查看软件中的帮助文件。

1. ctype.h　字符处理类

isalnum int isalnum(int c)	字母还是数字
isalpha int isalpha(int c)	是否字母
iscntrl int iscntrl(int c)	是否控制码
isdigit int isdigit(int c)	是否数字
isgraph int isgraph(int c)	是否为可打印的非空字符
islower int islower(int c)	是否小写字母
isprint int isprint(int c)	是否为可打印字符
ispunct int ispunct(int c)	是否为表示标点符号的字符
isspace int isspace(int c)	是否为空白字符
isupper int isupper(int c)	是否为大写字符
isxdigit int isxdigit(int c)	是否为十六进制数
tolower int tolower(int c)	转换为小写字符
toupper int toupper(int c)	转换为大写字符

```
        * key, const void * base,
        size_t nmemb, size_t size,
        int (* compare) (const void
        * _key, const void * _base));
```

calloc void * calloc(size_t nelem, size_t elsize)	为目标数组分配存储器单元
div div_t div(int numer, int denom)	除法运算函数
exit void exit(int status)	结束程序
free void free(void * ptr)	释放存储器单元
labs long int labs(long int j)	整形数取绝对值
ldiv ldiv_t ldiv(long int numer, long int denom)	长整形除法
malloc void * malloc(size_t size)	分配存储器
qsort void qsort(const void * base, size_t nmemb, size_t size, int (* compare) (const void * _key, const void * _base));	数组排序
rand int rand(void)	随机数生成函数
realloc void * realloc(void * ptr, size_t size)	重新分配存储器单元函数
srand void srand(unsigned int seed)	设置随机数(的种子)
strtod double strtod(const char * nptr, char ** endptr)	将字符串转换为双精度数
strtol long int strtol(const char * nptr, char ** endptr, intase)	将字符串转换为长整形数
strtoul unsigned long int strtoul const char * nptr, char ** endptr, base int)	将字符串转换为无符号整形数

6. string.h　字符串处理类

memchr void * memchr(const void * s,int c, size_t n)	在存储器中搜索字符
memcmp int memcmp(const void * s1,const void * s2, size_t n)	比较存储器内容
memcpy void * memcpy(void * s1,const void * s2, size_t n)	拷贝存储器内容
memmove void * memmove(void * s1,const void * s2, size_t n)	移动存储器内容
memset void * memset(void * s,int c, size_t n)	置存储器
strcat char * strcat(char * s1,const char * s2)	逻辑字符串
strchr char * strchr(const char * s, int c)	在字符串中找某一个字符
strcmp int strcmp(const char * s1,const char * s2)	比较两个字符串
strcoll int strcoll(const char * s1,const char * s2)	比较字符串
strcpy char * strcpy(char * s1, constchar * s2)	拷贝字符串
strcspn size_t strcspn(const char * s1, const char * s2)	在字符串中跨过被排除的字符
strerror char * strerror(int errnum)	给出一个错误信息字符串
strlen size_t strlen(const char * s)	计算字符串长度函数
strncat char * strncat(char * s1,const char * s2, size_t n)	将指定数量的字符与字符串连接起来
strncmp int strncmp(const char * s1,const char * s2, size_t n)	将指定数量的字符与字符串相比较
strncpy har * strncpy(char * s1,const char * s2, size_t n)	在字符串中复制指定的字符
strpbrk char * strpbrk(const char * s1, const char * s2)	在字符串中寻找任何指定的字符

strrchr char * strrchr(const char * s, int c)	从字符串的右端开始寻找字符
strspn size_t strspn(const char * s1, const char * s2)	在字符串中统计和分析字符
strstr char * strstr(const char * s1, const char * s2)	在字符串中搜索子字符串
strtok char * strtok(char * s1, constchar * s2)	将标志前的字符剪掉
strxfrm size_t strxfrm(char * s1,const char * s2, size_t n)	转换字符串并返回其长度

2.7.5 C语言编程实例

下面的程序将在P1.0输出方波。程序中使用了两个函数:一个是主函数,一个是中断函数。主函数主要是对看门狗定时器和端口进行设置。在主函数中有一个循环(for语句),这是整个程序的主循环。

看门狗定时器的中断服务程序由它的中断向量标志[WDT_VECTOR]决定,在进入中断之后,CPU到地址0FFE0H中WDT_VECTOR中去找PC的内容。

```
#include  <msp430x11x1.h>
void main(void)
{
  WDTCTL = WDT_ADLY_250;                    // 设置看门狗定时时间 250 ms
  IE1 |= WDTIE;                             // 使能 WDT 中断
  P1DIR |= 0x01;                            // 设置 P1.0 为输出方向
  _EINT();                                  // 使能总的中断
    for (;;)                                // 主循环
  {
    _BIS_SR(LPM3_bits);                     // CPU 和 DCO 都不需要了
    _NOP();
  }
}
interrupt[WDT_VECTOR] void watchdog_timer(void)
{
  P1OUT ^= 0x01;                            // P1.0 求反以输出方波
}
```

第 3 章　MSP430 单片机片内外设原理与使用方法

MSP430 系列单片机将大量的外围模块集成到片内，称之为片内外设。在第 1 章已经介绍了 430 家族每一系列的情况，从它们的结构框图可以看出，不同型号器件的片内外设不同。本章将详细讲述 MSP430 家族器件中的各种片内外设的工作原理与使用方法。最基本的部件 16 位的 CPU 和存储器系统在第 2 章已经讲述，这里将着重讲解 MSP430 的时钟模块、端口、定时器、通信模块、液晶驱动部件、模数转换器、硬件乘法器、模拟比较器和 FLASH 存储器等。

3.1　基础时钟模块与低功耗

MSP 所有器件都有时钟模块，都能实现超低功耗应用，但不同器件又不完全相同。总的来说，MSP430 的时钟由高速晶体、低速晶体、数字控制振荡器 DCO、锁频环 FLL 以及锁频环增强版本 FLL＋等部件构成。各系列的时钟模块见表 3.1。

表 3.1　各系列的时钟模块

部　件	430X3XX	430X13X, 14X	43/ F4XX	430X11XX	430F12X
高速晶体		✓	✓		
低速晶体	✓	✓	✓	✓	✓
DCO	✓	✓	✓	✓	✓
FLL	✓				
FLL＋			✓		

而各个系列的不同基础时钟模块产生相同的结果：输出 3 种不同频率时钟 ACLK（辅助时钟）、MCLK（主系统时钟）和 SMCLK（子系统时钟），送给各种不同需求的模块。图 3.1 所示为基本时钟模块的输入输出关系。

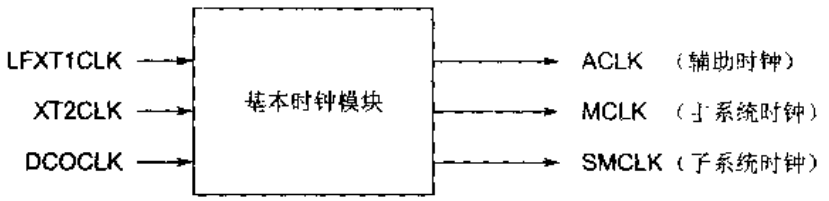


图 3.1　MSP430 的时钟

正是由于有 3 种不同频率的时钟输出给不同的模块，才使的整个系统超低功耗成为可能。系统的功耗与系统工作的频率成正比。MSP430 从基本时钟模块输出 3 种用户可调整的不同时钟以满足不同的需要，如：用户可用高速晶体产生频率较高的 MCLK 供给 CPU 以满足高速的数据运算的需要；也可以在不需要 CPU 工作的时候关闭 MCLK；而对于实时时钟可用较准确的 ACLK 供给；为减少电流消耗，EMI（电磁干扰）等可使用低频率。

32 768 Hz的时钟晶体，同时将所需两个小电容也集成在内部。这样可降低系统成本，同时降低系统功耗。低速晶体振荡器在 MSP430 的技术文献中一般用 LFXT1 表示（本书以后也用 LFXT1 表示），LFXT1 振荡器在发生有效的 PUC 信号后开始工作，一次有效的 PUC 信号可以将 SR 寄存器（状态寄存器）中的 OscOff 位复位，即允许 LFTX1 工作。如果 LFTX1CLK 信号没有用作 SMCLK 或 MCLK 信号，则可以用软件将 OscOff 位置位以禁止 LFTX1 工作。

LFTX1 振荡器的结构图如图 3.3 所示。当使用 32 768 Hz 的晶体时只须连到相应的引脚，两个小电容已经集成在芯片内部。应注意：这时 LFTX1 工作在低频模式，XTS 位应复位。

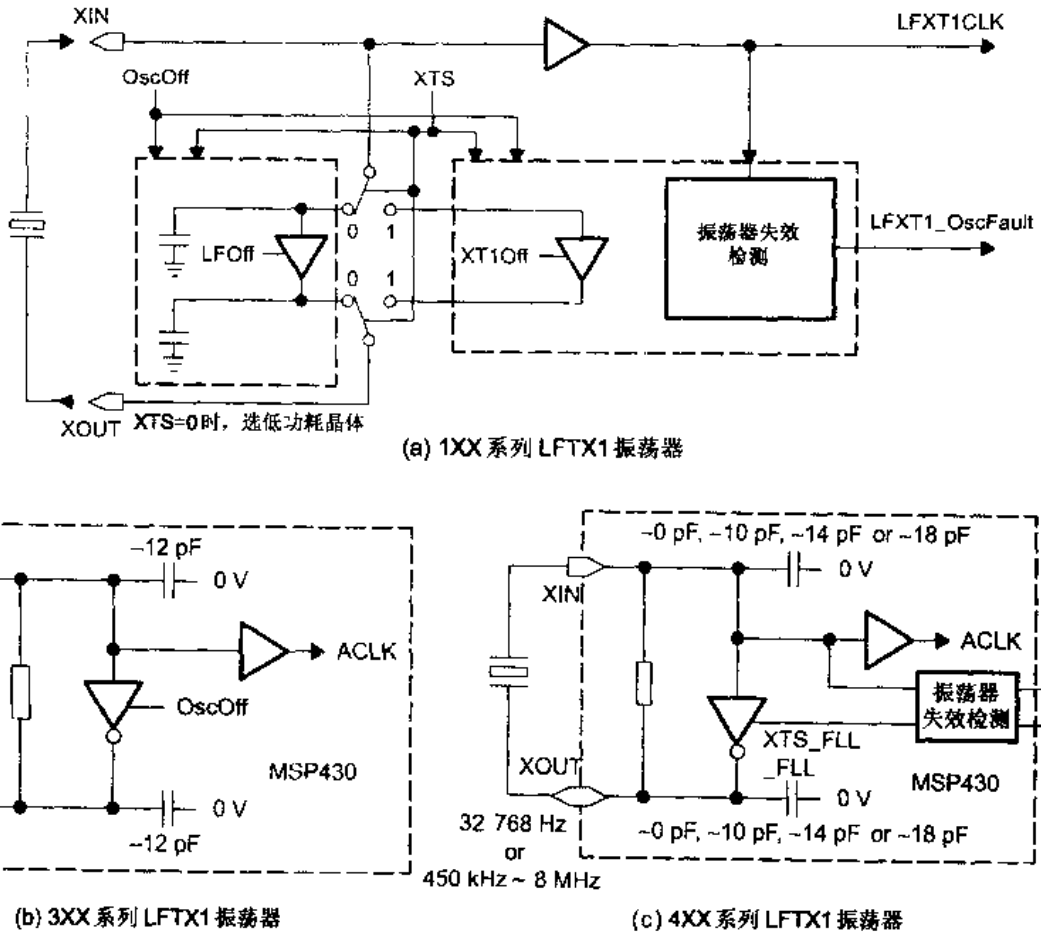


图 3.3　MSP430 系列 LFTX1 振荡器

同时 LFTX1 振荡器也可以外接频率较高的高速晶体振荡器或陶瓷谐振器，以工作在高速模式；这时 XTS 位必须置位，同时两个引脚还要外接电容，电容大小根据晶体或谐振器的特性来选择。

3.1.2　高速晶体振荡器

高速振荡器主要存在于 F13X、F14X、F4XX 等器件，一般称之为第二振荡器 XT2，它产生时钟信号 XT2CLK，它的工作特性与 LFTX1 振荡器工作在高频模式时类似。图 3.4 是 XT2 振荡器的控制逻辑。如果 XT2CLK 信号没有用做 MCLK、SMCLK 时钟信号，则可用 XT2Off 控制位关闭 TX2；如果 CPUOff=0，SELM=2，则 XT2CLK 用做 MCLK 时钟；如果 SCG1=0

且 SELS=1，则 XT2CLK 用做 SMCLK 时钟。

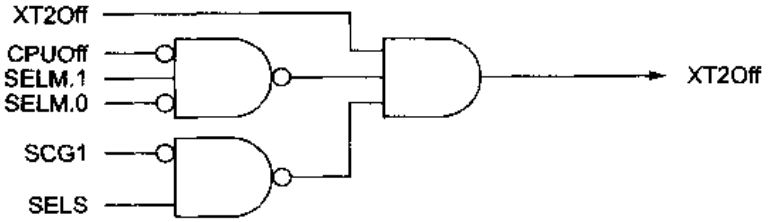


图 3.4 XT2 振荡器的控制逻辑

对于 MSP430 器件，一般都有外部振荡器 LFXT1 或 XT2，系统由外部振荡器提供较为准确的时钟。但振荡器也有失效的时候，这时 MSP430 有硬件电路检测振荡器是否失效。一般当振荡器丢失大约 100 个振荡周期时，设置振荡器失效标志 OscFault。OscFault 标志设置振荡器失效中断标志 OFIFG，如果允许振荡器失效中断，则产生非屏蔽中断请求。

3.1.3 DCO 振荡器

MSP430 的两个外部振荡器产生的时钟信号都可以经 1，2，4，8 分频后用作系统主时钟 MCLK。当振荡器失效时，DCO 振荡器会自动被选作 MCLK 的时钟源。由于 DCO 振荡器被自动用于 MCLK，因此由 XT 振荡器失效引起的 NMI 中断请求可以得到响应，甚至在 CPU 关闭的情况下也能得到处理。MSP430 可让任意被允许的中断请求在低功耗模式下得到服务，甚至在 LPM4 模式下。MCLK 在中断服务时自动有效。

DCO 振荡器实质上是一个可数字控制的 RC 振荡器。因为 RC 振荡器的频率会随温度和工作电压的变化而变化，同一型号芯片产生的频率有所不同。但同时 DCO 频率可通过 DCO，MOD，Rsel 等控制位用软件调节，这又增加了振荡频率的稳定性。

1. DCO 振荡器原理

图 3.5 是 MSP430F1XX 的 DCO 控制原理图，它将一个内部或外部电阻接到直流发生器，它的阻值决定了 DCOCLK 的基础频率。最终的 DCO 输出频率 DCOCLK 由以下功能建立：

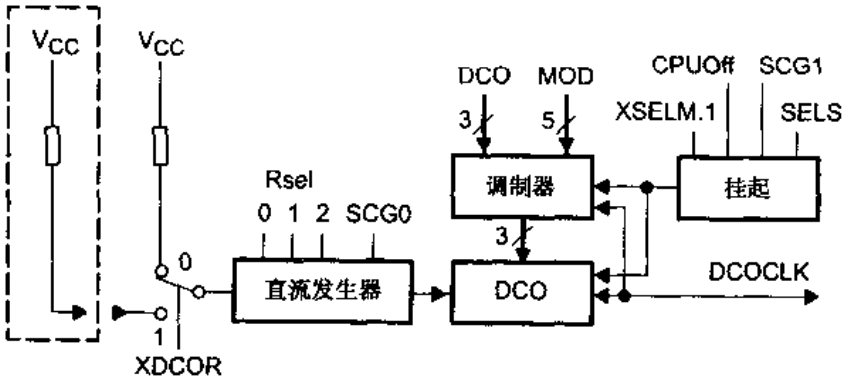


图 3.5 DCO 的控制原理图

① DCO 基础频率由内部或外部电阻向 DC 发生器注入的电流决定，由 DCOR 控制位选择片内或片外电阻。

② 基础频率由控制位 Rsel2，Rsel1 和 Rsel0 分频为 8 个标称频率范围，而这些频率范围因不同型号而异。

③ 控制位DCO0,DCO1和DCO2可分段调节DCOCLK频率。

④ 由5个调整位MOD0～MOD4控制切换DCO和DCO+1选择的两种频率。

2. DCO控制寄存器

由DCO控制位选择的DCOCLK频率大约可以在时钟周期的10%范围内调整。这里以MSP430F1XX为例详细介绍如何控制MSP430的DCOCLK频率。在MSP430F1XX系列中DCO的控制由3个寄存器来完成:DCOCTL,BCSCTL1及BCSCTL2。下面介绍这3个寄存器。

● **DCOCTL** 地址为056H,PUC后的值为060H,每一位定义如下:

7	6	5	4	3	2	1	0
DCO.2	DCO.1	DCO.0	MOD.4	MOD.3	MOD.2	MOD.1	MOD.0

DCO.0～DCO.2 定义了8种频率之一,而频率由注入直流发生器的电流定义。

MOD.0～MOD.4 定义了在32个DCO周期中插入的f_{DCO+1}周期,而在余下的DCO周期中为f_{DCO}周期。如果DCO常数为7,由于已经选择了最高频率将不进行调整。

● **BCSCTL1** 地址为057H,PUC后的值为084H,每一位定义如下:

7	6	5	4	3	2	1	0
XT2OFF	XTS	DIVA.1	DIVA.0	XT5V	Rsel.2	Rsel.1	Rsel.0

XT2OFF 控制XT2振荡器的开启与关闭。

XT2OFF=0,XT2振荡器开启;

XT2OFF=1,XT2振荡器关闭。

XTS 选择LFXT1工作在低频晶振模式还是高频晶振模式,选择须同实际晶体。

XTS=0,LFTX1工作在低频模式;

XTS=1,LFTX1工作在高频模式。

DIVA.0,DIVA.1 两位选择ACLK的分频系数。

DIVA=0,ACLK的分频系数是1;

DIVA=1,ACLK的分频系数是2;

DIVA=2,ACLK的分频系数是4;

DIVA=3,ACLK的分频系数是8。

XT5V 此位须为0。

Rsel.0,Rsel.1,Rsel.2 三位选择某个内部电阻以决定标称频率。

Rsel=0,选择最低的标称频率;

……

Rsel=7,选择最高的标称频率。

● **BCSCTL2** 地址为058H,PUC后的值为00H,每一位定义如下:

7	6	5	4	3	2	1	0
SELM.1	SELM.0	DIVM.1	DIVM.0	SELS	DIVS.1	DIVS.0	DCOR

SELM.0,SELM.1 两位选择MCLK的时钟源。

SELM=0,MCLK的时钟源为DCOCLK;

SELM=1,MCLK的时钟源为DCOCLK;

SELM=2,MCLK 的时钟源为 LFTX1CLK(对于 MSP430F11XX),
MCLK 的时钟源为 TX2CLK(对于 MSP430F13X、14X);
SELM=3,MCLK 的时钟源为 LFTX1CLK。

DIVM.0,DIVM.1 两位选择 MCLK 的分频因子。
DIVM=0,分频因子为 1;
DIVM=1,分频因子为 2;
DIVM=2,分频因子为 4;
DIVM=3,分频因子为 8。

SELS 选择 SMCLK 的时钟源。
SELS=0,SMCLK 的时钟源为 DCOCLK;
SELS=1,SMCLK 的时钟源为 LFTX1CLK(对于 MSP430F11XX),
SMCLK 的时钟源为 TX2CLK(对于 MSP430F13X、14X)。

DIVS.0,DIVS.1 两位选择 SMCLK 的分频因子。
DIVS=0,分频因子为 1;
DIVS=1,分频因子为 2;
DIVS=2,分频因子为 4;
DIVS=3,分频因子为 8。

DCOR 选择内部电阻还是外部电阻。
DCOR=0,选择内部电阻;
DCOR=1,选择外部电阻。

3. 实验说明

以上介绍的是 MSP430F1XX 系列器件中的 DCO 控制寄存器的详细情况。

下面将用实验进一步说明。图 3.6 为实验对象,通过改变 MSP430F133(实验的实际器件)的寄存器的值,得到对应的 MCLK 的实际测量值,MCLK 由 P5.4 输出送到频率计,得到数据如图 3.6 所示。

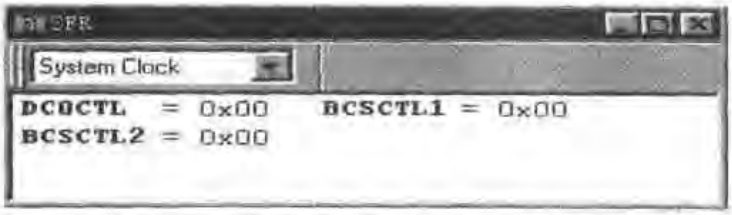


图 3.6 F13X、F14X 系统时钟模块的寄存器

首先,标称频率由 8 个内部电阻或外部电阻决定,这里使用内部电阻,DCO 控制器的其他参数不变(使用复位时的值),只改变决定电阻的参数:Rsel.0,Rsel.1,Rsel.2。实验数据如下:

BCSCTL1 的数值	80H	81H	82H	83H	84H	85H	86H	87H
MCLK 输出频率/kHz	129	189	278	468	738	1 334	2 065	3 200

可以看出,通过改变内部电阻的阻值可以很大程度改变系统的运行时钟范围。

其次,DCO 频率又有 8 种选择,DCO 控制器的其他参数不变(使用复位时的值),由此三

个控制位 DCO.0,DCO.1,DCO.2 选择 8 种频率之一。实验数据如下：

DCOCTL 的数值	00H	20H	40H	60H	80H	A0H	C0H	E0H
MCLK 输出频率/kHz				747	825	895	990	1124

这些数据反应出通过控制 DCO.0,DCO.1,DCO.2 参数可以较为精确地控制系统的运行时钟。

最后，选择了 DCO 的电阻，也选择了 DCO 在标称频率之上的某个频率，还可以更进一步将这个频率准确化：运用 MOD 参数进行调制，调制的具体含义为在 32 个 DCO 周期中插入频率为 DCO+1 的时钟信号，而插入的个数由 MOD 参数决定，其余的为 DCO 信号。下列实验数据说明了这个问题。在实验中只改变 MOD 参数，DCO 控制器的其他参数不变(使用复位时的值)。实验数据如下：

DCOCTL 的数值	60H	65H	6AH	73H	7AH	7FH	80H
MCLK 输出频率/kHz	748	759	770	800	814	824	825

从上面所列的实验数据可以看出，DCO 时钟调制的范围为 32 档，(这里只做了 7 组数据)可以将 DCO 频率定义得很精确。但通过 MOD 参数的调制范围为两个 DCO 频率之间，即在 DCO 与 DCO+1 频率之间，也就是说 DCO 频率调制是分段进行的，如图 3.7 所示。

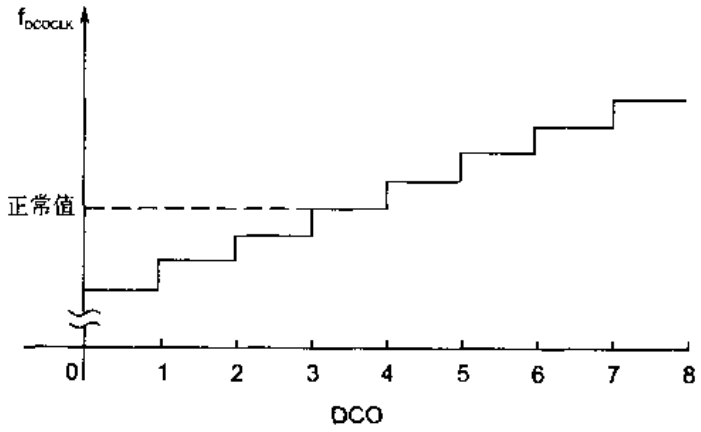


图 3.7　DCO 频率分段控制

对于 MSP430X3XX 和 MSP430F4XX 来说，DCO 控制寄存器是不一样的。MSP430X3XX 使用 SCFI0，SCFI1，SCFQCTL，CBCTL；而 MSP430F4XX 使用 SCFI0，SCFI1，SCFQCTL，FLL+CTL0，FLL+CTL1。但基本原理大同小异，请参考相应技术手册。

3.1.4　锁频环 FLL/FLL+

在某些 MSP430 器件上使用了锁频环技术 FLL 或增强型锁频环技术 FLL+，如在 MSP430X3XX 中使用 FLL 技术，在 MSP430F4XX 中使用 FLL+技术等。在 MSP430F4XX 系列的时钟模块中，同样有 DCO、LFXT1 振荡器及 LFXT2 振荡器(41 系列没有 LFXT2 振荡器)。但在时钟模块中增加了 FLL+技术，可以在低频振荡器的驱动下，得到较高的稳定频率。下面讲述 FLL+，其余相同部分不予赘述。图 3.8 是 4XX 的时钟模块。

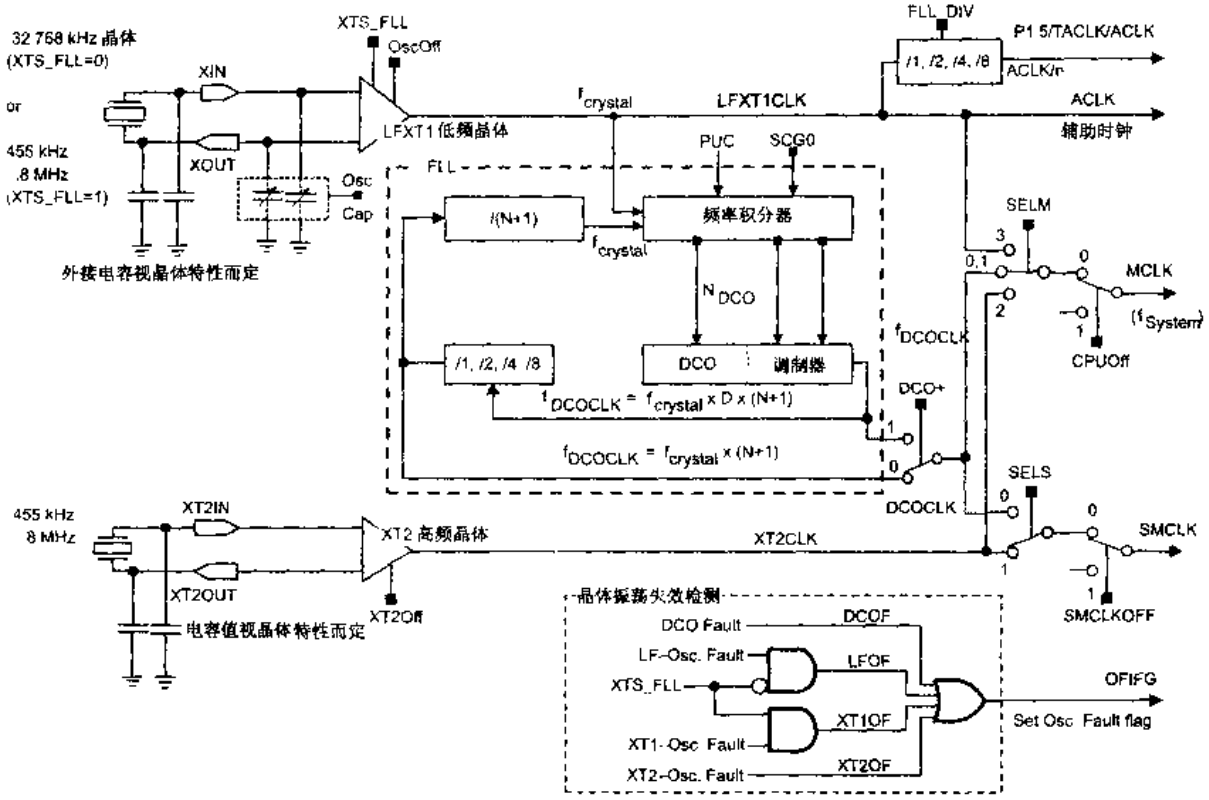


图 3.8 MSP430F4XX 的时钟模块

由图可以看出，DCO 的最终输出频率与前面所讲的有些不同：其一，ACLK 可以分频之后再由引脚输出；其二，这里的 DCO 输出为低频振荡器输出频率的 N+1 倍。因为使用晶体的低频振荡器输出的频率比较稳定，所以 DCO 的输出频率也较稳定。其原因是使用了增强型锁频环技术(FLL+)，消除了由 RC 型振荡器因温度、电压等不稳定因素带来的不稳定性。DCO 是在 MSP430F4XX 的 FLL+时钟模块中的一个 RC 振荡器。DCO 产生一个时钟信号 f_{DCOCLK}。这个信号通常用作 MSP430F4XX 的 CPU 时钟信号和外围设备的时钟信号。MCLK 是 ACLK 的倍频。倍频因子 N 在控制寄存器 SCFQCTL(SCFQCTL.6～SCFQCTL.0)的低 7 位内。起初 N 在 PUC 信号后为 31，D 为 2，DCO+被复位，使 ACLK 信号被 32 倍频和产生 1.048 576 MHz 的 MCLK 和 SMCLK，见图 3.9。

由倍频因子(N+1)和 D 来改变 MCLK 和 SMCLK 的频率：

DCO+=0，$f_{DCOCLK}=f_{ACLK}\times(N+1)$

DCO+=1，$f_{DCOCLK}=f_{ACLK}\times D\times(N+1)$

锁频环技术使 MCLK/SMCLK 的频率非常稳定。当和 DCO 组合使用时有以下两个优点：

● 快速启动。MSP430F4XX 的 DCO 振荡器的响应时间小于 6 μs，可以用来支持长睡眠周期和突发事件的执行。

● 数控信号。DCO 振荡器启动的时候有相同的设置，这就使正常运行的时候不需要一个长的锁定周期。

用户软件可以在任何时候通过改变分频因子 N 和 D 加上 DCO+位来改变 DCO 的频率，

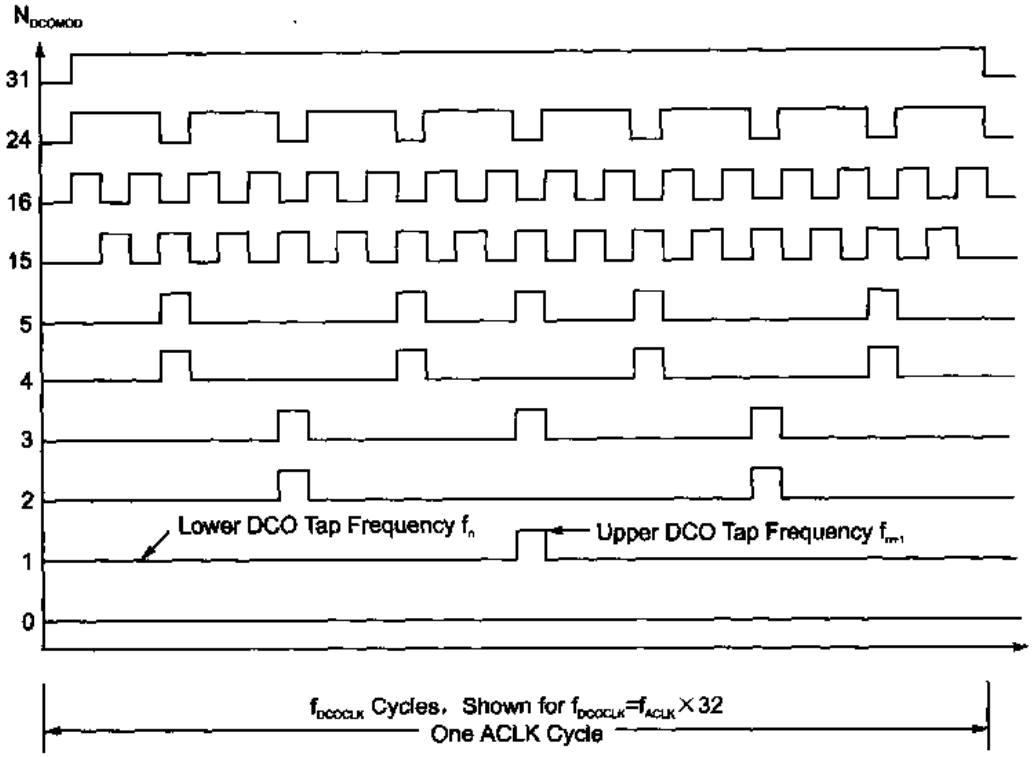


图 3.11 调制器的调节

表 3.2 DCO 频率范围

FN_8	FN_4	FN_3	FN_2	f_{DCOCLK} /MHz*
0	0	0	0	0.7～6
0	0	0	1	1.4～12
0	0	1	×	2.2～17
0	1	×	×	3.2～25
1	×	×	×	5～40

* 最终的 MCLK 不能超出器件手册允许的频率范围。

4. FLL＋模块的控制寄存器

FLL＋模块是使用控制寄存器 SCFQCTL，SCFI0，SCFI1，FLL＋CTL0，FLL＋CTL1（CBCTL 为 3XX 系列的）和 CPU 状态寄存器的 SCG1，SCG0，OscOff 及 CPUOff 4 个控制位来实现控制的，如图 3.12 所示。用户可以利用软件在任何时候修改这些状态寄存器。这些控制寄存器都是字节形式的，必须以字节指令来访问。

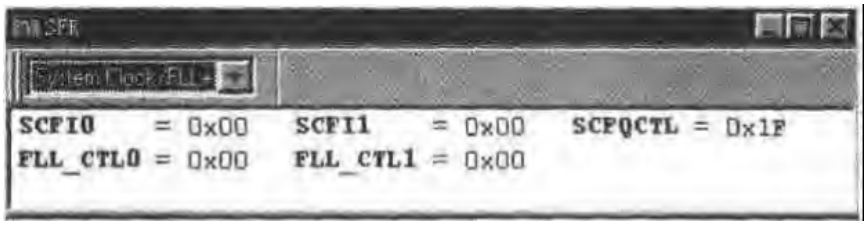


图 3.12 FLL＋的寄存器

(1) SCFQCTL 寄存器(MCLK/SMCLK 频率控制)

寄存器 SCFQCTL 内的值和 SCFI0 的 D 位控制着 DCOCLK 的频率,来作为 MCLK 和 SMCLK 信号。寄存器 SCFQCTL 的位与含义如下:

7	6	5	4	3	2	1	0
M	2^6	2^5	2^4	2^3	2^2	2^1	2^0

其中 7 位指明范围为 1+1~127+1,任何小于 1 的值都将导致不确定的运行。用户必须确定所选择的值不要超过系统允许的最大值。MCLK 和 SMCLK 的值在用户的使用手册中标明。

$f_{System}=(x\cdot 2^6+x\cdot 2^5+x\cdot 2^4+x\cdot 2^3+x\cdot 2^2+x\cdot 2^1+x\cdot 2^0+1)\cdot f_{crystal}$[DCO+=0]

$f_{System}=D\times(x\cdot 2^6+x\cdot 2^5+x\cdot 2^4+x\cdot 2^3+x\cdot 2^2+x\cdot 2^1+x\cdot 2^0+1)\cdot f_{crystal}$[DCO+=1]

在 PUC 信号后,SCFQCTL 的值默认为 31 时,将导致 32×(DCO+=0)的结果。频率积分器的输出控制着 DCO。

如果调制位 M 置位,只有 DCO 的频率决定系统频率,邻近的 DCO 周期不被混合。然而,如果 FLL+在运行(SCG0=0)时,会不断的调整 DCO 周期。如果有请求需要系统频率保持一段时间的,那么调制和 FLL+都得停止(M=1,SCG0=1)。

(2) SCFI0 和 SCFI1 寄存器

该两寄存器包含 3 方面的信息:系统频率输出的分频系数;系统频率的调整范围;10 位调整参数。

图 3.13 所示为 SCFI0 和 SCFI1 寄存器。

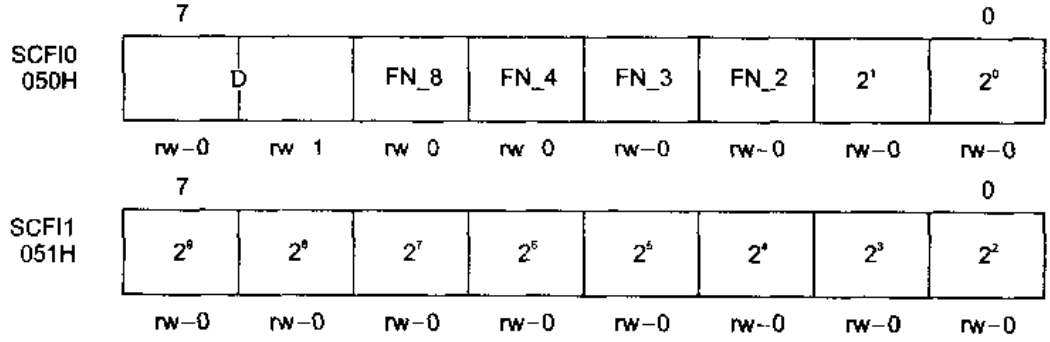


图 3.13　SCFI0 和 SCFI1 寄存器

图中 D 为分频系数参数。D 的数值 0,1,2,3 分别对应分频系数 1,2,4,8。FN_2~FN_8 为频率的可调整范围(如表 3.3 所列),当然要在器件允许的范围内才有效。

表 3.3　频率的可调整范围

FN_8	FN_4	FN_3	FN_2	f_{DCOCLK} ,MHz+
0	0	0	0	0.7~6
0	0	0	1	1.1~12
0	0	1	×	2.2~17
0	1	×	×	3.2~25
1	×	×	×	5~40

+ 最终的 MCLK 不能超出器件手册允许的频率范围。

(3) FLL_CTL0 和 FLL_CTL1 寄存器

图 3.14 所示为 FLL _CTL0 和 FLL_CTL1 寄存器。

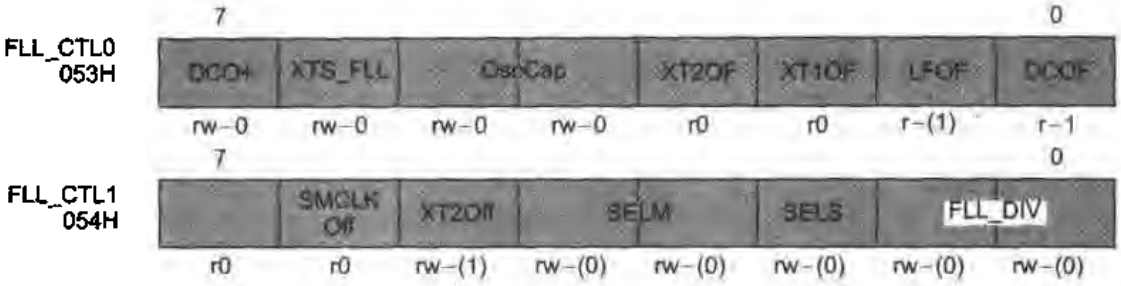


图 3.14 FLL_CTL0 和 FLL_CTL1 寄存器

该两控制寄存器各位的含义如下：

B0～3 DCOF,LFOF,XT1OF 和 XT2OF 位是用来表明时钟失效的。DCOF 对应的是 DCO 振荡失效;LFOF 指的是低频模式(LF)下的 LFXT1 振荡失效;XT1OF 对应高频模式(HF)下的 LFXT1 振荡失效;XT2OF 对应 XT2 振荡失效。如果这些信号一个或多个置位时,振荡器失效标志 OFIFG 置位。这些位是只读的,只有在发生错误情况时才会置位或复位。

0:无错误;

1:有错误。

注意:XT2OF 位在 MSP430X41X 系列中读出值永远为 0,因为在此系列的器件内无 XT2 振荡器。

OscCap 在振荡器引脚存有一个小范围变化的电容,容量在 0～18 pF,当它与引脚及电路板间的典型电容 2 pF 混合时,其有效晶振负载电容约在 1～10 pF 之间。

0:有效晶振负载电容大约为 1 pF;

1:有效晶振负载电容大约为 6 pF;

2:有效晶振负载电容大约为 8 pF;

3:有效晶振负载电容大约为 10 pF。

注意:OscCap 的默认值为 0,提供一个 1 pF 有效晶振负载电容。OscCap 值的选择或外部电容提供恰当的负载电容,可以达到可靠的晶振运行。

XTS_FLL LFXT1 振荡器是一个低频晶体(典型频率 32 768 Hz,LF 模式)或一个高频晶体或谐振器(HF 模式)。

0:LF 模式;

1:HF 模式。

DCO+ 当 DCO 输出需要预分频时要选择 DCO+位来实现。分频率用 FLL_DIV 位来实现。

0:DCO 不分频;

1:DCO 在 MCLK 或 SMCLK 前分频。

FLL_DIV 此位是用来作为 LFXT1 频率的分频因子,信号 ACLK/n 可以被选用,在引脚 P1.5/TACLK/ACLK。

0:n=1,ACLK/n 信号为 ACLK;

1:n=2,ACLK/n 信号为 ACLK/2;

2:n=4,ACLK/n 信号为 ACLK/4;

3:n=8,ACLK/n 信号为 ACLK/8。

SELS　选择外围模块的时钟信号源。

0:SMCLK=DCOCLK;

1:SMCLK=XT2CLK。

SELM　选择 CPU 使用的 MCLK 的信号的时钟源。

0,1:MCLK=DCOCLK;

2: MCLK=XT2CLK;

3: MCLK=ACLK(LFXT1 振荡器)。

XT2Off　关闭 XT2 振荡器。

0:如果 XT2 没有被用做 MCLK(SELM≠2 或 CPUOFF=1)或 SMCLK(SELS=0 或 SMCLKOFF=1),则关闭 XT2;

1:打开 XT2。

SMCLKOff　关闭时钟信号 SMCLK。

0:打开 SMCLK;

1:关闭 SMCLK。

5. 特殊功能寄存器中与 FLL+有关的位

FLL+模块会影响特殊功能寄存器的 OFIFG 和 OFIE 两个功能位。振荡失效中断允许位(OFIE)位于中断允许寄存器 IE1 的 1 位;振荡失效中断标志位(OFIFG)位于中断标志寄存器 IFG1 的 1 位,如图 3.15 所示。

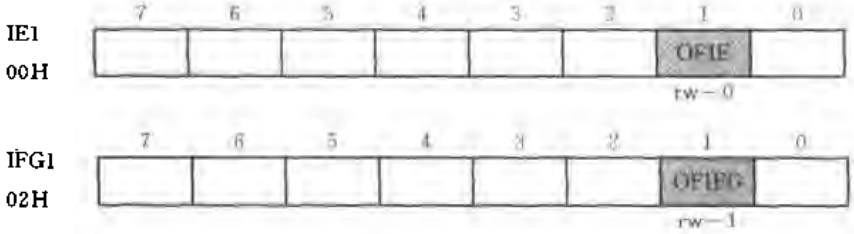


图 3.15　特殊功能寄存器的两个功能位

只要振荡器失效,振荡失效信号就会使 OFIFG 一直置位。如果 OFIE 置位时,振荡失效中断将请求一个非屏蔽中断,如果非屏蔽中断被接受,振荡失效允许位自动复位。OFIE 的初始状态是复位的,即使有错误情况发生,也不会有振荡失效请求中断。

3.1.5　基础时钟模块与低功耗

通过控制 MSP430 的时钟系统,能方便地使其构成超低功耗应用系统。MSP430 有多种工作模式,而这些工作模式的实现依靠对时钟的控制。图 3.16 所示为 MSP430 的工作模式状态。任何一个中断事件可以将系统从各种低功耗模式中唤醒,而 RETI(中断返回)指令又可将系统返回到中断前的状态。使用图 3.16 所示的程序框图可进行超低功耗的系统设计。

在主程序中完成系统初始化,如中断的设置、端口的分配和时钟的调度等,CPU 也就没用了;甚至进入低功耗模式后主系统时钟也可以停止,这时系统功耗只在 μA 数量级的范围。一

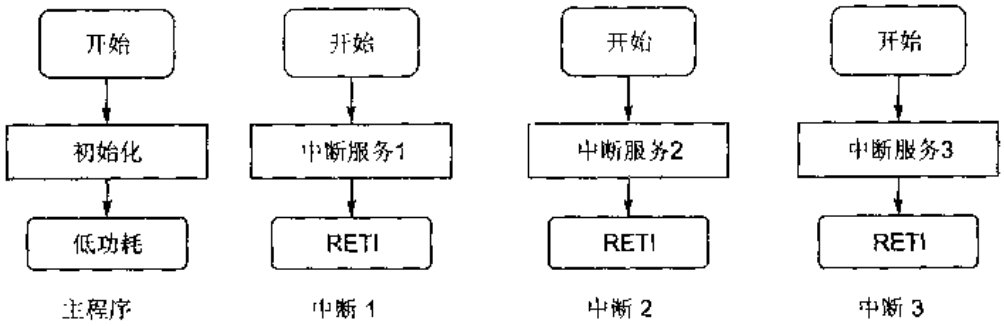


图 3.16 典型的程序框架

11有允许的中断请求，CPU 将在 6 μs 的时间内被唤醒，进入活动模式，执行中断服务程序。执行完毕，在 RETI 指令之后，系统返回到中断前的状态，继续低功耗模式。也就是说，系统功耗的情况取决于中断服务程序的执行时间。

从图 3.17 可看出，MSP430 有 1 种活动模式和 5 种低功耗模式。通过相应的设置可从活动模式进入相应的低功耗模式；而各种低功耗模式都可通过中断的方式进入活动模式。

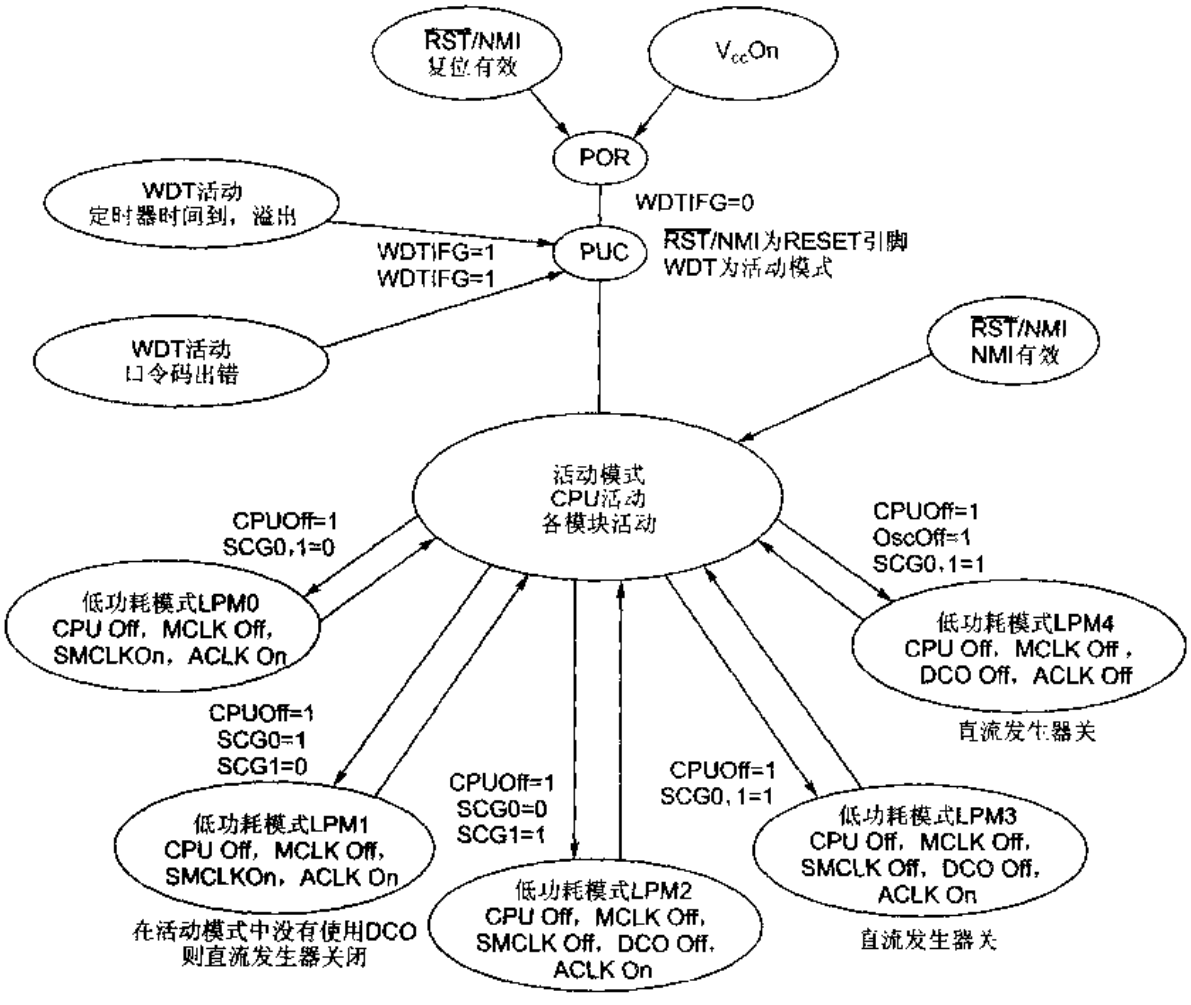


图 3.17 MSP430 工作模式状态图

续表 3.4

工作模式	控制位	CPU 状态、振荡器及时钟
低功耗模式 4 (LPM4)	SCG1=X SCG0=X OscOff=1 CPUOff=1	CPU 处于禁止状态 DCO 被禁止，DCO 的直流发生器被禁止 所有振荡器停止工作 MCLK 被禁止 SMCLK 被禁止，ACLK 被禁止

3.1.6 时钟系统的应用举例

有这样一个应用系统，要求包含实时时钟、测量、运算和输出，并且要求电池供电和长期工作。前面的要求很容易实现，要解决后面的要求就必须运用时钟系统进行恰当地时钟分配。

对于实时时钟模块，使用低频振荡器产生辅助时钟 ACLK，送到硬件定时器，如定时器 A（见定时器部分）；再选择分频系数，使进入计数器的频率降低。这样，可使执行实时时钟中断程序的频率降低（执行程序的次数少），可降低功耗

对于测量部分，需要相对准确的时钟作为模数转换硬件模块的时钟源，这里也选用低频振荡器产生的 ACLK。测量由中断完成。

对于运算部分，由于程序的执行要求时间尽可能短，而对频率的准确度没有要求，所以用 DCO 产生 MCLK 作为系统主时钟。同样该功能在中断中实现，一般紧接在测量之后。

系统程序的框架使用图 3.16 所示的模式。在主程序中完成一系列的初始化，包括时钟的分配，主要为进入中断做准备；而所有实实在在的事情在各自的中断服务程序中实现。

通过下面的程序，可将系统 3 个时钟由端口引出（以 MSP430F119 为例）。

```
#include    "msp430x14x.h"
            ORG     0F000h                          ;程序开始处
RESET       mov.w   #300h,SP                        ;初始化堆栈指针
StopWDT     mov.w   #WDTPW+WDTHOLD,&WDTCTL          ;停止看门狗
            bis.b   #010h,&P1DIR                    ; P1.4 输出
            bis.b   #010h,&P1SEL                    ; P1.4   SMCLK
            bis.b   #001h,&P2DIR                    ; P2.0 输出
            bis.b   #001h,&P2SEL                    ; P2.0 = ACLK
            bis.b   #010h,&P5DIR                    ; P5.4 输出
            bis.b   #010h,&P5SEL                    ; P5.4=  ACLK
            bis.w   #CPUOFF,SR                      ; CPU 关闭

            ORG     0FFFEh                          ; MSP430 复位地址
            DW      RESET                           ;
            END
```

3.2 MSP430 各种端口

MSP430 有丰富的端口可供用户使用。在目前产品中有 P0，P1，P2，P3，P4，P5，P6，TP0，

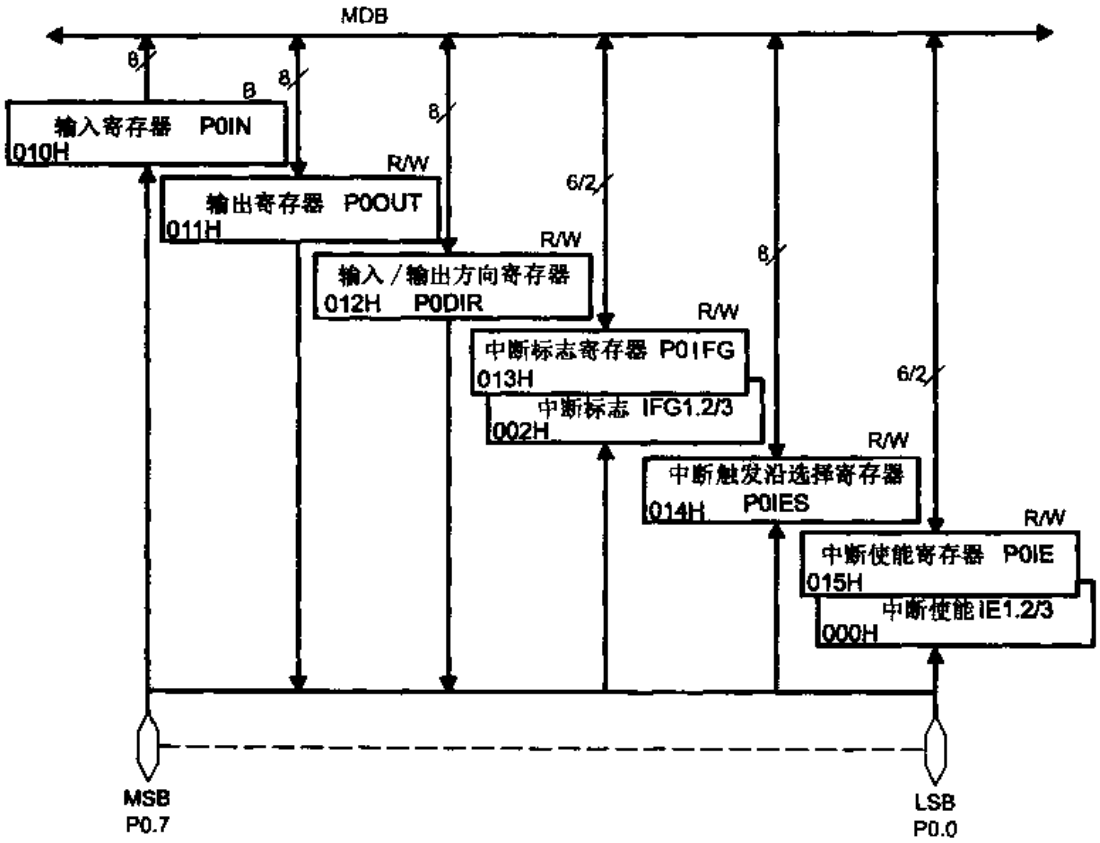


图 3.20 P0 口的结构

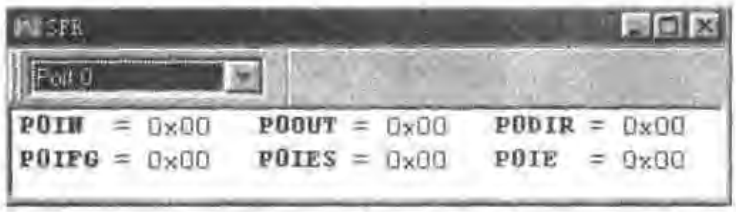


图 3.21 P0 口调试界面

7	6	5	4	3	2	1	0
P0IE.7	P0IE.6	P0IE.5	P0IE.4	P0IE.3	P0IE.2		

0：禁止该位中断；

1：允许该位中断。

(5) P0IES P0 口中断触发沿选择寄存器

如果允许 P0 口的某个引脚中断，还须定义该引脚的中断触发沿。该寄存器的 8 位分别定义了 P0 口的 8 个引脚的中断触发沿。

0：对应引脚由低到高的电平跳变(上升沿)使相应标志置位；

1：对应引脚由高到低的电平跳变(下降沿)使相应标志置位。

(6) P0IFG　P0 口中断标志寄存器

该寄存器有 6 个标志位，它们含有相应引脚是否有待处理中断的信息，或相应引脚是否有中断请求。如果 P0 口的某个引脚允许中断，同时选择上升沿，则当该引脚发生电平由低向高跳变时，P0IFG 的相应位就会置位，表明在该引脚上有中断事件发生。6 个标志位分别对应高 6 位，低 2 位标志在 SFR 中，如下所示：

7	6	5	4	3	2	1	0
P0IFG.7	P0IFG.6	P0IFG.5	P0IFG.4	P0IFG.3	P0IFG.2		

0：没有中断；

1：有中断请求。

关于 P0 口的中断，在图 3.20 中，还另有两个寄存器中使用了两位：IFG1.2、3 和 IE1.2、3，这两个寄存器是 SFR 寄存器，位于地址 0002H 和 0000H。

2. 端口 P1 和 P2

在 MSP320X3XX，MSP430F1XX 及 MSP430F4XX 中都有 P1 和 P2。与 P0 口一样，也有一系列寄存器对其控制，P1 和 P2 口的结构如图 3.22 所示。调试界面如图 3.19 所示。可以看出有 7 个控制寄存器对每个端口操作，图中的 n=1、2，两个端口共 14 个寄存器，对它们的访问须用字节指令以绝对模式进行访问。

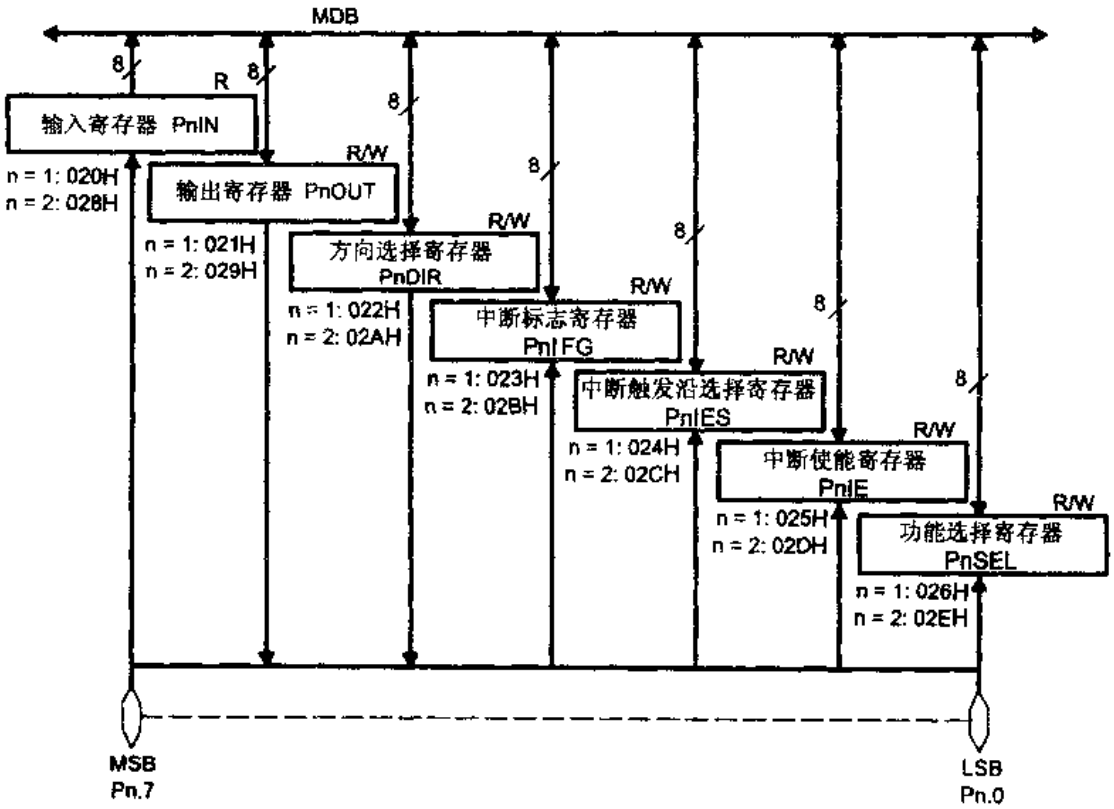


图 3.22　P1 和 P2 口的结构

(1) P1DIR，P2DIR　P1 和 P2 端口方向选择寄存器

相互独立的 8 位分别定义了 8 个引脚的输入/输出方向。8 位在 PUC 后都被复位。一般在使用端口时，都要先定义该寄存器，使引脚的输入/输出满足设计者的要求。

3.2.2　端口 P3,P4,P5 和 P6

这些端口没有中断能力,其余功能与 P1 和 P2 一样,能实现输入/输出功能和外围模块功能。每个端口有 4 个寄存器供用户使用。用户可通过这 4 个寄存器对它们进行访问和控制。图 3.23 为 P3,P4,P5 和 P6 的结构。

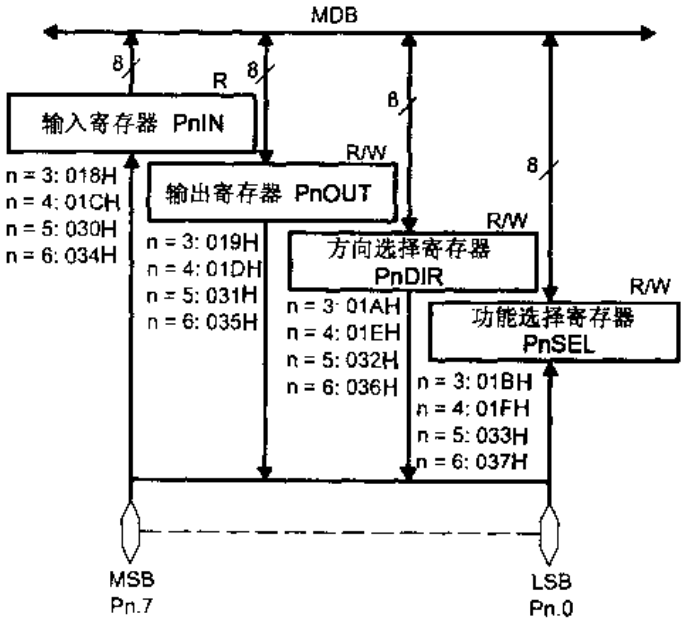


图 3.23　P3,P4,P5 和 P6 的结构

在 MSP430F1XXX 系列器件中都有 P5 和 P6 两端口;在 MSP430F11XX,MSP430F12XX 及 MSP430X3XX 等系列器件中有 P3 端口;在 MSP430F13X/14X 和 MSP430X3XX 等系列器件中有 P4 端口。

(1) PnDIR　P3,P4,P5 和 P6 端口方向选择寄存器

相互独立的 8 位分别定义了 8 个引脚的输入/输出方向。8 位在 PUC 后都被复位。一般在使用端口时,都要先定义该寄存器,使引脚的输入/输出满足设计者的要求。

0:I/O 引脚被切换成输入模式;

1:I/O 引脚被切换成输出模式。

(2) PnIN　P3,P4,P5 和 P6 端口输入寄存器

输入寄存器是 CPU 扫描 I/O 引脚信号的只读寄存器。用户不能对它写入,只能通过读取该寄存器中内容知道 I/O 端口的输入信号。此时引脚的方向必须选定为输入。

(3) PnOUT　P3,P4,P5 和 P6 端口输出寄存器

该寄存器为 I/O 端口的输出缓冲寄存器,可用所有包含目的操作数的指令修改,以达到改变 I/O 口状态的目的。在读取时输出缓存的内容与引脚方向定义无关。改变方向寄存器的内容,输出缓存的内容不受影响。

(4) PnSEL　P3,P4,P5 和 P6 端口功能选择寄存器

P3,P4,P5 和 P6 端口还有其他片内外设功能,考虑减少引脚,将这些功能与芯片外的联系通过复用 P1 和 P2 引脚的方式来实现。P1SEL 和 P2SEL 用来选择引脚的 I/O(输入/输

出)端口功能和外围模块功能。

0：选择引脚为 I/O 端口；

1：选择引脚为外围模块功能。

3.2.3 端口 TP0

在 MSP430X3XX 系列器件中，有三态输出端口 TP0，它是一个 6 位的端口，由以下两个寄存器控制，分别使用低 6 位。

(1) TPD 端口数据寄存器

TPD.0～TPD.5 的内容为引脚 TP0.0～TP0.5 的输出值。

7	6	5	4	3	2	1	0
		TPD.5	TPD.4	TPD.3	TPD.2	TPD.1	TPD.0

(2) TPE 端口允许寄存器

TPE.0～TPE.5 分别控制 TP0.0～TP0.5 的三态特性。

7	6	5	4	3	2	1	0
		TPE.5	TPE.4	TPE.3	TPE.2	TPE.1	TPE.0

3.2.4 COM 和 S 端口

这些端口实现与液晶片的直接接口。COM 端口为液晶片的公共端，S 端口为液晶片的段码端。液晶片输出端也可经软件配置为数字输出端口。

3.2.5 端口应用举例

P0 口可实现 3 种功能：输入、输出及外部中断。

使用输入和输出功能时，必须先定义端口的方向。作为输入时，只能读；作为输出时，可读可写。

P1 口在 MSP430 的很多型号器件中都有，这里举一个使用 P1 口的中断能力实现键盘输入的例子。在很多应用(如 BP 机、电子表等)中使用 3 个按键实现数据的录入，而一般使用中断的方式。这里在 P1.0，P1.1 及 P1.2 三根 I/O 口线连接了 3 个按键，下面的程序将能感知按键，并识别按键。

```
        MOV.B   #07H,&P1DIR
        MOV.B   #00H,&P1OUT
        MOV.B   #0F8H,&P1DIR        ;P1 低 3 位为输入
        MOV.B   #07H ,&P1IE         ;P1 低 3 位允许中断
        MOV.B   #0, &P1IES          ;P1 低 3 位相应引脚上升沿触发中断
        EINT                        ;开总中断
P1INT   TST.B   #01H, &P1IN         ;P1.0 有按键按下吗?
        JZ      KEY1TEST            ;如果有，跳到相应程序段，处理
        TST.B   #02H,&P1IN          ;如果没有，继续测试下一位
        JZ      KEY2TEST            ;如果有，跳到相应程序段，处理
```

```
            TST.B   #08H,&PIN           ;如果没有,继续测试下一位
            JZ      KEY3TEST            ;如果有,跳到相应程序段,处理
            RETI                        ;5键都没有按下,为干扰,不处理,退出
KEY1TEST
            CALL    #KEY1               ;具体处理程序
            RETI
KEY2TEST
            CALL    #KEY2               ;具体处理程序
            RETI
KEY3TEST
            CALL    #KEY3               ;具体处理程序
            RETI
```

3.3 定时器

MSP430系列有多种定时器模块:看门狗定时器(WDT)、基本定时器(Basic Timer1)、8位定时器/计数器(8-bit Timer Counter)、定时器A(Timer_A)和定时器B(Timer_B)等。这些模块都能实现定时功能,但并不是所有器件都有。如所有器件都有看门狗定时器,而Basic Timer1只有MSP430X3XX系列有。下面分别介绍这些具有定时功能的模块。

3.3.1 看门狗定时器

看门狗定时器(WDT)实质上是一个定时器,其主要功能是:当程序发生故障时能使受控系统重新启动。如果WDT超过WDT所定时的时间,即发生系统复位。如果系统不需要看门狗功能,也可将它当定时器使用,当到达WDT所定时的时间时能产生中断。图3.24是MSP430F1XX和MSP430F4XX系列中的看门狗定时器原理图。MSP430X3XX系列中的看门狗定时器与前述两系列的惟一差别在于两个时钟源为ACLK和MCLK。

WDT有如下特性:

- 其主体是一个16位计数器;
- 需要口令才能对其操作;
- 有看门狗和定时器两种模式;
- 有8种可选的定时时间。

1. WDT寄存器

WDT有一个专门的控制寄存器,有一个计数单元,中断允许和中断标志在SFR中。WDT的计数器WDTCNT是16位增计数器,不能直接用软件访问,需要经WDTCTL(地址为0120H)对WDTCNT进行访问。从图3.24可以看出WDTCTL被分成两部分:高8位被用做口令,低8位才是对WDT操作的控制命令。而要写入操作WDT的控制命令,必须先正确写入高字节看门狗口令。

在读WDTCTL时,不需要口令,可直接读取120H中的内容,读出数据低字节为WDTCTL的值,高字节始终为69H。

而要写入WDTCTL时,必须写入正确的口令才能实现写操作。高字节为口令,口令为

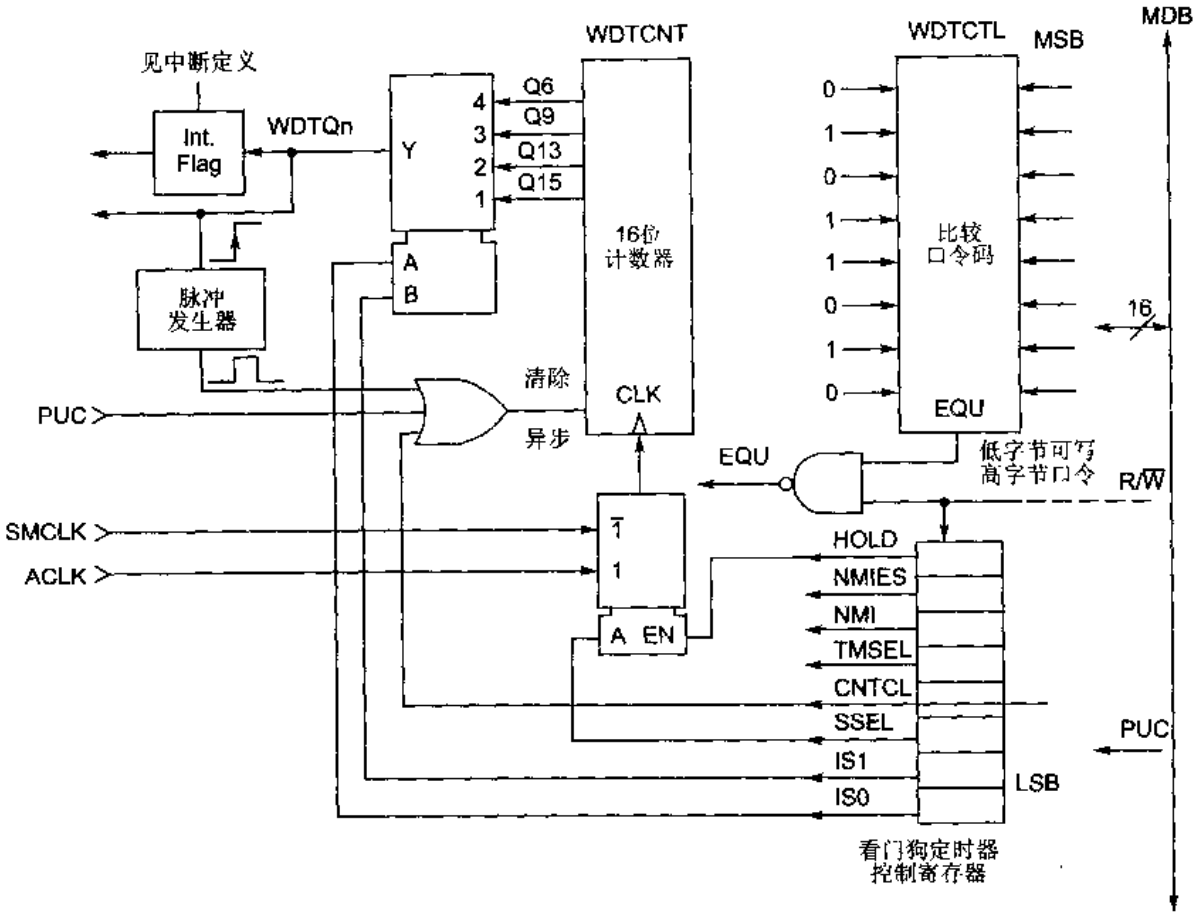


图 3.24 MSP430F1XX 和 MSP430F4XX 系列的看门狗定时器原理图

5AH，如果口令写错将导致系统复位。

下面是 WDTCTL 寄存器各位的定义：

15～8	7	6	5	4	3	2	1	0
口令	HOLD	NMIES	NMI	TMSEL	CNTCL	SSEL	IS1	IS0

IS0，IS1　选择 WDTCNT 的 4 个输出之一。

这两位为图 3.24 中数据选择器的两个地址选择端，看门狗定时器的定时输出由这两位选择，而选择的 4 个频率分别是 WDTCNT 的 4 个输出：Q6，Q9，Q13，Q15。由此可知，4 个输出频率为 $T\times 2^{6}$，$T\times 2^{9}$，$T\times 2^{13}$ 和 $T\times 2^{15}$，其中 T 为 WDTCNT 的输入时钟频率。

SSEL　选择 WDTCNT 的时钟源。

0　选择 SMCLK 作为 WDTCNT 的时钟源；

1　选择 ACLK 作为 WDTCNT 的时钟源。

由 IS0，IS1 及 SSEL 三位便可确定 WDT 的定时时间，表 3.5 列出了在晶振为 32 768 Hz，SMCLK＝1 MHz 条件下，WDT 可选的定时时间。

表 3.5　晶振为 32 768 Hz,SMCLK=1 MHz 条件下,WDT 的定时时间

SSEL	IS1	IS0	定时时间/ms	
0	1	1	0.064	$t_{SMCLK}\times 2^{5}$
0	1	0	0.5	$t_{SMCLK}\times 2^{9}$
1	1	1	1.9	$t_{ACLK}\times 2^{6}$
0	0	1	8	$t_{SMCLK}\times 2^{13}$
1	1	0	16.0	$t_{ACLK}\times 2^{9}$
0	0	0	32	$t_{SMCLK}\times 2^{15}$ + PUC 复位后的值
1	0	1	250	$t_{ACLK}\times 2^{13}$
1	0	0	1000	$t_{ACLK}\times 2^{15}$

CNTCL　清除 WDTCNT。当该位为 1 时,对于 WDT 的两种模式,WDTCNT 都将从 0 开始计数。

TMSEL　工作模式选择。

0　工作在看门狗模式;

1　工作在定时器模式。

NMI　选择RST/NMI 引脚功能,在 PUC 后被复位。

0　$\overline{\text{RST}}$/NMI 引脚为复位端;

1　$\overline{\text{RST}}$/NMI 引脚为边沿触发的非屏蔽中断输入。

NMIES　在选择$\overline{\text{RST}}$/NMI 引脚为非屏蔽中断输入时,该位选择引脚的电平跳变沿。

0　由低向高的上升沿触发 NMI 中断;

1　由高向低的下降沿触发 NMI 中断。

HOLD　停止看门狗定时器工作。

0　WDT 功能激活;

1　时钟禁止输入,计数停止。

而 WDT 的中断允许位 WDTIE 位于 IE1.0,中断标志位 WDTIFG 位于 IFG1.0。

2. WDT 的工作模式

WDT 可使用户通过 WDTCTL 寄存器中的 TMSEL 控制位设置工作在看门狗模式和定时器模式,同时还可将 WDT 关闭。

(1) 看门狗模式

当 TMSEL=0 时,WDT 工作在看门狗模式。在这种模式下,一旦 WDT 定时时间到或写入错误的口令都会触发 PUC 信号,同时自动清除系统寄存器中的各位,WDT 被再次设置为看门狗(TMSEL=0),$\overline{\text{RST}}$/NMI 引脚为复位模式。

由于在上电复位或系统复位时,WDT 自动进入看门狗模式,WDTCNT 和 WDTCTL 两寄存器内容被全部清除,而 WDT 的时钟来源 ACLK 和 SMCLK 都有信号,这些情况将导致 WDT 的运行。所以,用户软件一般都需要进行 WDT 的初始化设置,以保证 WDT 的正确使用。

看门狗的目的在于发现程序跑飞,其原理在于:看门狗定时器设置一定时时间,如 250 ms,这个时间是所有用户程序一定能在此时间内执行完该程序的一个时间;设置好这个定时时间之后,所有用户程序就必须在这个设定的时间内将看门狗计数器的值清零,使计数器

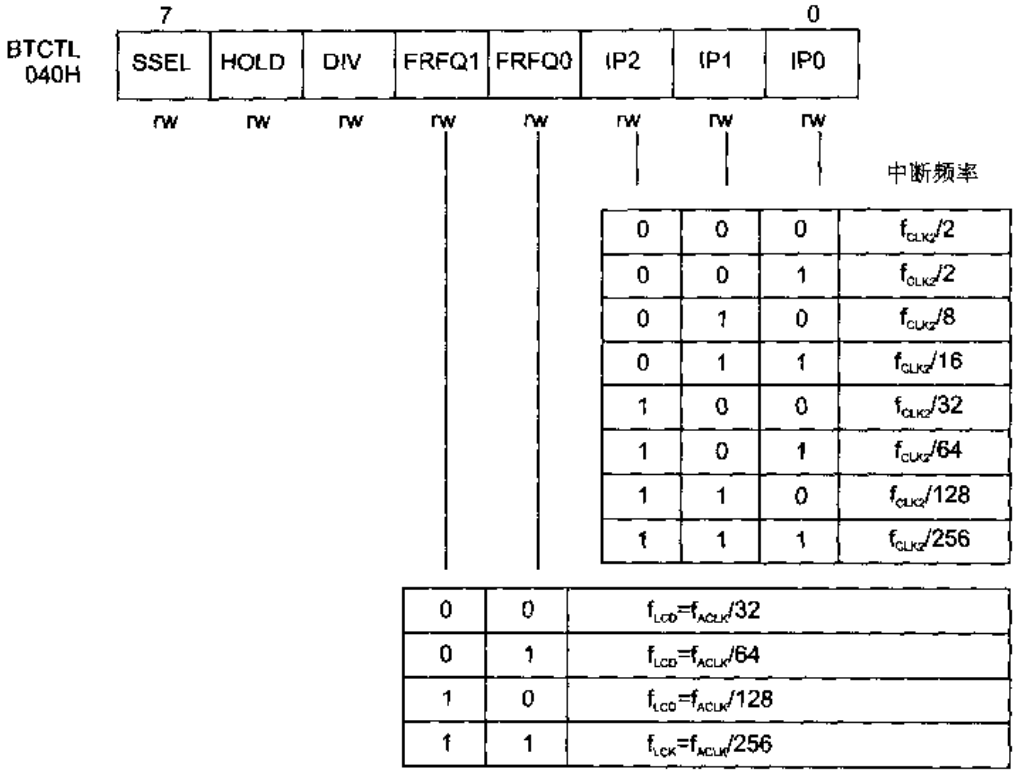


图 3.26 BTCTL 各位功能

源有 3 个：ACLK，MCLK 及 ACLK/256，由 SSEL 和 DIV 两位选择，如表 3.6 所列。BTCNT2 的输出是为了产生中断，使中断标志置位。从表 3.6 和图 3.25 还可看出：Basic Timer1 可以工作在 16 位定时器/计数器模式，因为 CLK2 可以来源于 BTCNT1 的最高位输出。当作 16 位计数器时，时钟源只能是 ACLK。

表 3.6 CLK2 的选择

SSEL	DIV	CLK2
0	0	ACLK
0	1	ACLK/256
1	0	MCLK
1	1	ACLK/256

2. Basic Timer1 的中断

Basic Timer1 中断允许位 BTIE 位于 IE2.7。

Basic Timer1 中断标志位 BTIFG 位于 IFG2.7，该标志自动复位。

3. Basic Timer1 应用举例

在这个例程中，从 P5.1 口线输出一方波信号。利用 Basic Timer1 定时产生中断信号，在中断服务程序中，求反 P5.1 输出，这样就得到了要求的信号。注意，定时器的定时时间应该是方波周期的一半。源程序如下：

```
#include   "msp430x44x.h"
           ORG     01100H                          ; MSP430F449 的程序开始处
RESET      mov.w   #0A00H,SP                       ; MSP430F449 堆栈指针初始化
Init_Sys   mov.w   #WDTPW+WDTHOLD,&WDTCTL          ; 停止看门狗
```

```
SetupBT   mov.b   #BTSSEL-BTIP2-BTIP1-BTIP0,&BTCTL  ; ~214 μs Int.
          bis.b   #BTIE,&IE2                        ;使能 Basic Timer 中断
SetupP5   bis.b   #002H,&P5DIR                      ; P5.1 为输出
          eint                                      ;使能总中断
Mainloop  bis.w   #CPUOFF,SR                        ; CPU 不用了
          nop                                       ;
BT_ISR    xor.b   #002H,&P5OUT                      ; P5.1 求反输出
          reti                                      ;
          ORG     0FFFEH                            ; MSP430 RESET 地址
          DW      RESET                             ;
          ORG     0FFE0H                            ; BT 中断向量地址
          DW      BT_ISR                            ;
          END
```

3.3.3　8 位定时器/计数器

8 位定时器/计数器(8 - bit Timer/Counter)有的资料称之为 8 位间隔/定时器(The 8 - bit interval timer)。

在 MSP430X3XX 系列器件中都有 8 位定时器/计数器，其主要应用为：支持串行通信或数据交换，脉冲计数或脉冲累加以及定时器使用。

在 MSP430 的早期产品中，支持硬件串口的器件较为昂贵，故在没有硬件串口的情况下，可以利用 8 位定时器/计数器的特性软件实现串口功能。

1. 8 位定时器/计数器的结构

图 3.27 为 8 位定时器/计数器的原理图。从图中可以看出，它主要由以下一些 8 位模块组成：

- 8b　Preload Reg.　　8 位预置数寄存器；
- 8b　Counter　　8 位计数器；
- Control Reg.　　控制寄存器；
- Input clock selector　　输入时钟选择器；
- Edge　detection　　触发沿检测器；
- I/O Data latch　　输入输出数据锁存器。

(1) TCCTL　控制寄存器

控制寄存器的信息将决定定时器/计数器的操作。它是一个 8 位寄存器，其中各位定义如下：

7	6	5	4	3	2	1	0
SSEL1	SSEL0	ISCTL	TXE	ENCNT	RXACT	TXD	RXD

SSEL0,SSEL1　选择输入时钟源。在图 3.27 中进入 8 位计数器的时钟信号 CLK 是由 SSEL0 和 SSEL1 两位控制的 4 选 1 的数据选择器，被选择的时钟源如表 3.7 所列。

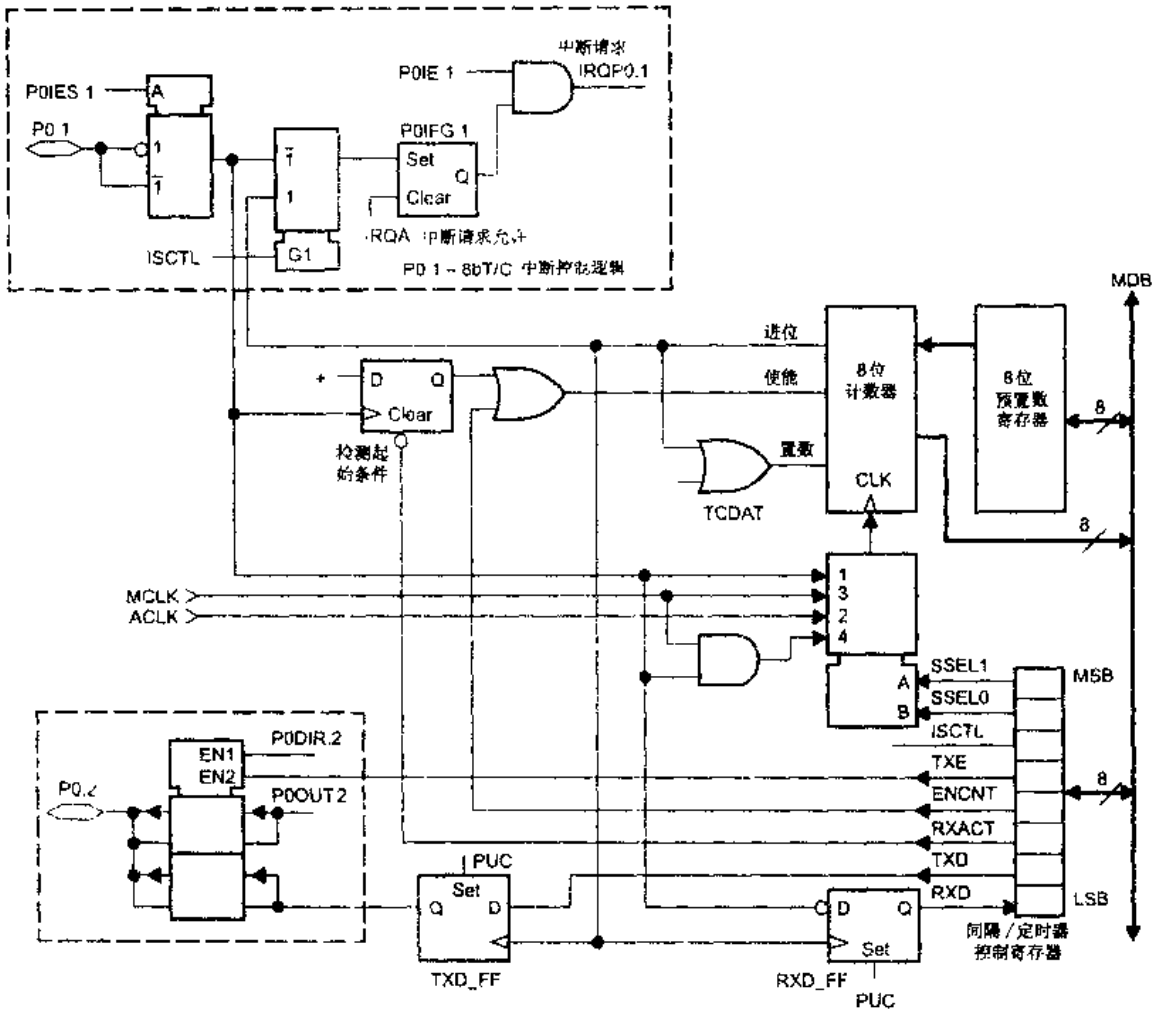


图 3.27 8 位定时器/计数器的原理图

表 3.7 CLK 的时钟源选择

SSEL1	SSEL0	CLK 的时钟源
0	0	由 P0IES.1 决定的 P0.1 引脚信号
1	0	MCLK
0	1	ACLK
1	1	由 P0IES.1 决定的 P0.1 引脚信号与 MCLK 的与

ISCTL 选择 P0.1 还是 8 位计数器的进位信号作为中断源。

0 引脚 P0.1 为 P0IFG.1 的中断源；

1 8 位计数器的进位信号为 P0IFG.1 的中断源。

TXE 控制 TXD 的三态输出缓存器。

0 三态；

1 激活输出缓存。

ENCNT 计数允许信号。8 位计数器在每个时钟上升沿加 1 计数。该位与 RXACT 一起提供启/停控制。

RXACT 控制边沿检测逻辑。边沿检测要用 ENCNT 位复位来允许计数操作。

0 清除边沿检测触发器，使它不能成为允许计数操作的信号；

1 允许边沿检测触发器，由 P0IES.1 选定的 P0.1 引脚的上升沿或下降沿使触发器置位，且计数器准备好，一旦触发器置位，它将保持到 RXACT=0。

TXD 该位由 8 位计数器的进位来定时控制从 P0.2 输出信号的缓存。8 位定时器/计数器主要根据其特性实现串口功能，串口要发送的数据将写入 TXD 位。

RXD 该位只读，由 P0.1 来的外部信号在 8 位定时器进位时锁存。外部信号由固定的时序扫描。8 位定时器/计数器主要根据其特性实现串口功能，串口接收到的数据锁存到 RXD 位。

(2) TCDAT、TCPLD 8 位带预置数寄存器的增计数器

TCDAT 和 TCPLD 分别是图 3.27 中的 8 位预置数寄存器和 8 位计数器。计数器的输入时钟由 SEL1 和 SEL0 选择。计数器由两个控制端——使能端和置数端(Enable、Load)对其控制，见图 3.27。

使能端(Enable)表示是否允许计数器进行增计数操作。当该位置位时，计数器在每个输入时钟上升沿加 1 计数，否则不计数。

置数端(Load)将预置数寄存器中的内容装入计数器。对计数器的写操作自动完成置数操作，注意这时计数器中的数据是预置数寄存器中的数，而不是指令中的数据。而所有指令均可对预置数寄存器读写操作。预置数寄存器相当于一个缓存器，并且可在计数器完成置数后立即写操作。

图 3.28 为 8 位定时器/计数器的使用举例。这里使该计数器从 37H 开始计数到 0FFH。在 8 位预置数寄存器中送数 37H，在计数器的增计数过程中，每当计数器计满，进位位为 1，将预置数寄存器中数 37H 装入计数器，计数器从 37H 开始增计数。也就是说，定时器的定时时间是(100H－37H)×T_{CLK}。

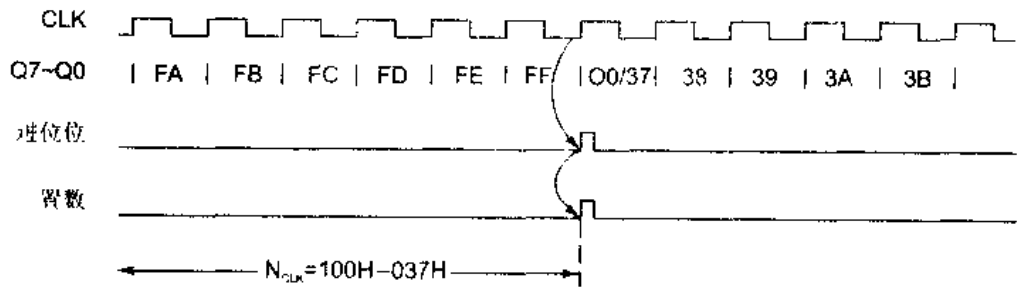


图 3.28 8 位定时器/计数器使用举例

2. 8 位定时器/计数器的中断

8 位定时器/计数器没有独立的中断标志位，它与 P0 端口共享中断标志位。寄存器 TCCTL 中的 ISCTL 位决定中断源。

3.3.4 通用定时器/端口

在 MSP430X3XX 系列器件中有通用定时器/端口(Universal Timer/Port)模块，该模块支持多种系统功能。如：

- 多达 6 个独立的三态输出；
- 2 个 8 位计数器，可级联成 16 位计数器；
- 用于斜坡转换原理的 A/D 转换的精密比较器。

1. 通用定时器/端口的结构

图 3.29 是通用定时器/端口的原理图。

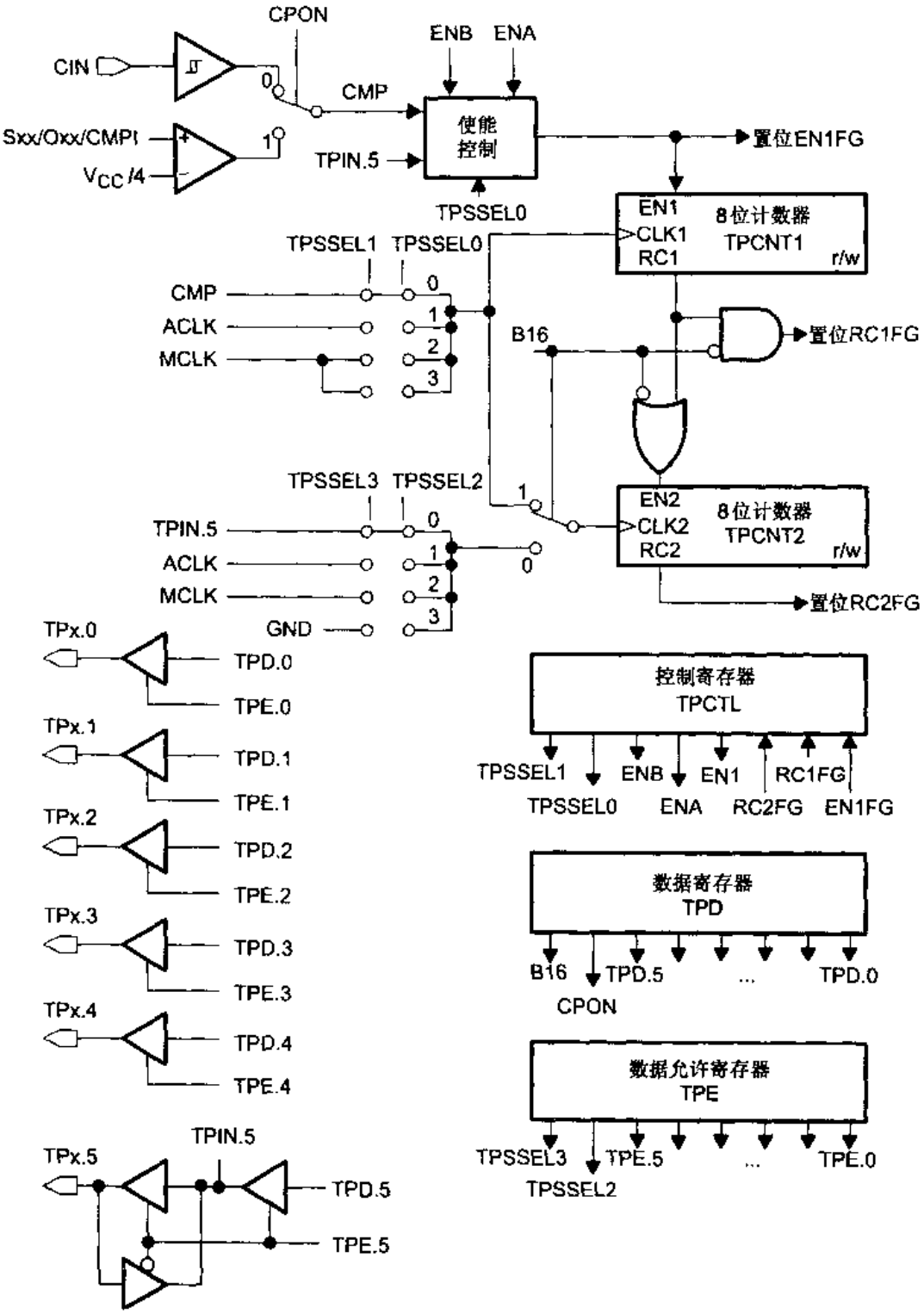


图 3.29 通用定时器/端口原理图

在该电路中，除了一些数据选择器和其他电路外，有 5 个寄存器起着重要作用，它们是：两个 8 位计数器 TPCNT1 和 TPCNT2、控制寄存器 TPCTL、数据寄存器 TPD 以及数据允许寄存器 TPE。下面分别介绍这些寄存器。

(1) TPCTL　控制寄存器

控制寄存器决定通用定时器 B 端口模块的操作，寄存器的各位定义如下：

7	6	5	4	3	2	1	0
TPSSEL1	TPSSEL0	ENB	ENA	EN1	RC2FG	RC1FG	EN1FG

TPSSEL0、TPSSEL1　两位选择 TPCNT1 的 3 个时钟源，具体如表 3.8 所列。

表 3.8　CLK1 的选择

TPSSEL1	TPSSEL0	CLK1
0	0	CMP
0	1	ACLK
1	X	MCLK

EN1　计数器 TPCNT1 的使能信号。该位为只读信号，它的值由 ENA，ENB 及 TPSSEL0 三位共同决定，其关系如表 3.9 所列。

表 3.9　EN1 的控制

ENB	ENA	TPSSEL0	EN1
0	0	X	0
0	1	X	1
1	0	0	$\overline{\text{TPIN.5}}$
1	0	1	TPIN.5
1	1	0	$\overline{\text{CMP}}$
1	1	1	CMP

RC2FG　该位指示 TPCNT2 计数器溢出，这一事件使 RC2FG 位置位。同时必须由用户软件复位，否则保持为 1。该位可用于中断服务程序中判断中断事件来源。

RC1FG　该位指示 TPCNT1 计数器溢出，这一事件使 RC1FG 位置位。同时必须由用户软件复位，否则保持为 1。该位可用于中断服务程序中判断中断事件来源。

EN1FG　使能标志。如果使能信号来自 CMP 和 TPIN.5，则计数器 TPCNT1 的使能信号 EN1 的下降沿允许该位置位。该位须用户软件复位，否则保持为 1。该位用在中断服务程序中以判断是由使能位还是由进位标志引起的中断。

(2) TPD　数据寄存器

数据寄存器有 6 位输出数值(用于输出到 TP0.0～TP0.5)和 2 位比较器控制位。各位定义如下：

7	6	5	4	3	2	1	0
B16	CPON	TPD.5	TPD.4	TPD.3	TPD.2	TPD.1	TPD.0

B16　16 位模式 / 8 位模式选择位。由于两个计数器为 8 位，所以经级联为 16 位模式，

也必须用字节指令对它们进行操作。

0 两个独立8位计数器模式；

1 级联为16位计数器模式。

CPON 比较器使能位。该位用于比较器的供电电源。

TPD.0～TPD.5 该6位数值位输出到引脚TP0.0～TP0.5的输出值。当TPE.0～TPE.5允许这些引脚三态输出时，数据加到相应引脚上。它们在PUC后复位。其中TP.5已在模块内利用，并可通过TPCTL的使能位EN1读取。

(3) TPE 允许寄存器

该寄存器有6个控制位(用于控制TP0.0～TP0.5)和2个用于计数器TPCNT2的时钟源选择位。各位定义如下：

7	6	5	4	3	2	1	0
TPSSEL3	TPSSEL2	TPE.5	TPE.4	TPE.3	TPE.2	TPE.1	TPE.0

TPSSEL2，PSSEL3 两位控制计数器TPCNT2的时钟源选择。这时B16必须复位，否则该2位不起作用，而TPCNT2的时钟源同TPCNT1。TPCNT2的时钟源选择具体情况如表3.10所列。

表3.10 TPCNT2的时钟源选择

B16	TPSSEL3	TPSSEL2	CLK2
0	0	0	TPIN.5
0	0	1	ACLK
0	1	0	MCLK
0	1	1	MCLK
1	X	X	=CLK1

TPE.0～TPE.5 此6位控制引脚TP0.0～TP0.5的三态。在PUC后复位且输出高阻态。

(4) TPCNT1，TPCNT2 8位计数器1和计数器2

两个计数器都是8位，必须通过字节指令访问，可读写。

在图3.30中，每个计数器的使能、时钟及输出都不一样。TPCNT1的使能与时钟的控制见表3.8和表3.9。有3个时钟源：CMP，ACLK及MCLK，由TPSSEL0和TPSSEL1两位选择；表3.10列出了TPCNT2的时钟源选择；而使能信号EN1由ENA，ENB及TPSSEL0三位共同决定。

两计数器的值都可写可读。但要注意：计数器一旦使能，计数器就在输入时钟上升沿时增计数，所以刚写进去的数与读出来的数可能不一样，后者很可能大一些。

两计数器可作为两个8位的独立计数器，也可以级联在一起作16位计数器用。这时必须将B16位置位，以实现这一操作。

2. 通用定时器/端口的中断

定时器/端口模块有一个中断向量、多个中断标志和一个中断允许位，如图3.30所示。

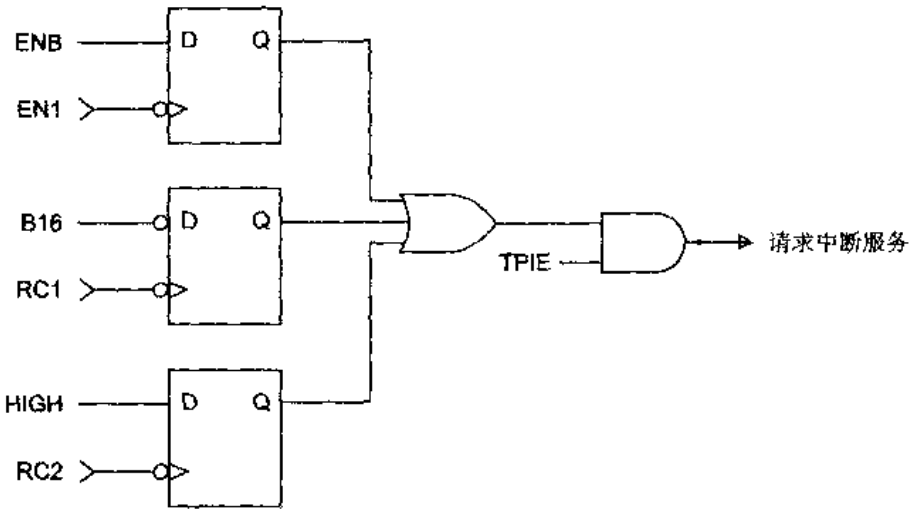


图 3.30　定时器/端口模块中断原理图

从图 3.30 中可以看出，中断能否得到响应，TPIE(中断允许位)为重要的一关，只有该位为 1，才可能提出中断请求。TPIE 位于寄存器 IE.2，其初始状态为 0。

有 3 个中断源，分别是标志位：RC1FG，RC2FG 及 EN1FG，即图中 3 个输入与门的输入端。它们位于 TPCTL 寄存器中，初始状态为 0。这 3 个标志位在中断服务时不能由硬件自动复位，须用户软件清除。

当选择 8 位模式时，所有 3 个中断标志都可能请求中断服务；而工作在 16 位模式时，只有 RC2FG 和 EN1FG 两个中断源可能请求中断服务。因此在中断服务程序中必须判断是哪个中断源申请的中断，以执行相应的服务。

3.3.5　16 位定时器 A

16 位定时器 A(Timer_A)是 MSP430 所有系列器件都有的模块，是一个用途非常广泛的通用 16 位定时器/计数器。它有以下一些特点：

- 16 位计数器，4 种工作模式；
- 多种可选的计数器时钟源；
- 具有多个可配置输入端的捕获/比较寄存器；
- 有 8 种输出模式的多个可配置的输出单元。

Timer_A 可支持同时进行的多种时序控制、多个捕获/比较功能及多种输出波形(PWM)，也可以是几种功能的组合。每个捕获/比较寄存器可以以硬件方式支持实现串行通信。

Timer_A 具有中断能力。中断可由计数器溢出引起，也可来自具有捕获或比较功能的捕获/比较寄存器。每个捕获/比较模块可独立编程，由捕获或比较外部信号以产生中断。外部信号可以是上升沿，也可是下降沿，也可二者都有。

在不同的 MSP430 器件中，Timer_A 模块中的捕获/比较器的数量不一样，如在 MSP430F133 中 Timer_A 模块含有 3 个捕获/比较器(简称 CCR)，因此常称为 Timer_A3，表示该模块含有 3 个 CCR。图 3.31 所示为 Timer_A 的结构原理图(各个系列稍有不同，此图为 MSP430F1XX 系列)。

图中，可以将 Timer_A 分解成几个部分：计数器部分、捕获/比较寄存器及输出单元。其

中，计数器部分完成时钟源的选择与分频、模式控制及计数等功能；捕获/比较寄存器用于捕获事件发生的时间或产生时间间隔；输出单元用于产生用户需要的输出信号。

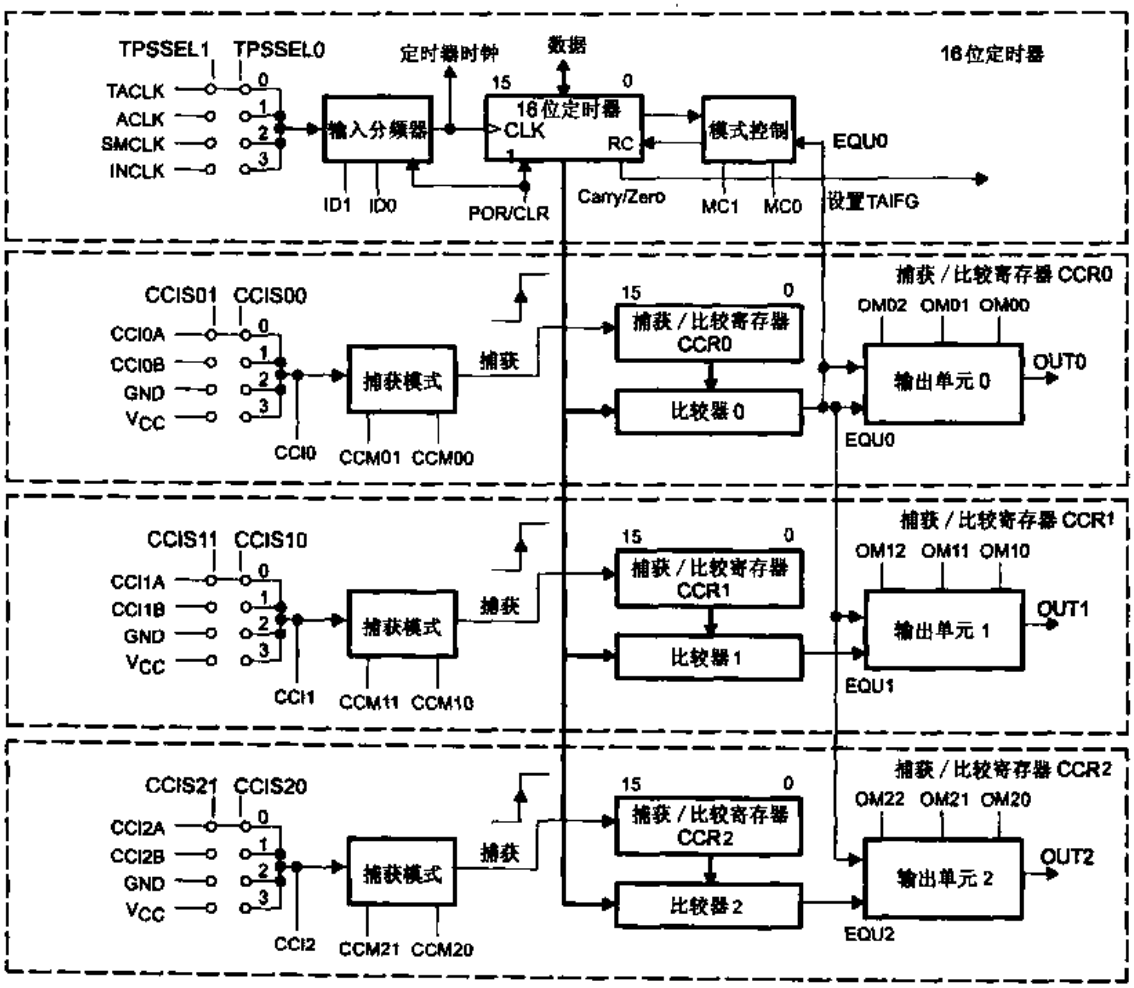


图 3.31 Timer_A 的结构原理图

1. Timer_A 的寄存器

用户对 Timer_A 的所有操作都是通过操作该模块的寄存器完成的，这里首先介绍它的寄存器。Timer_A 有下列寄存器(器件为 MSP430F133)如图 3.32 所示。

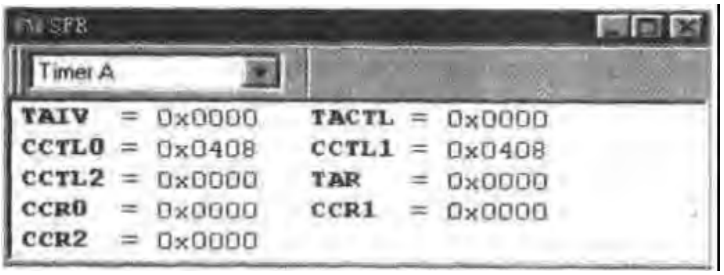


图 3.32 Timer_A 的寄存器

(1) TACTL 控制寄存器

TACTL 寄存器中含有全部的 Timer_A 控制位，为 16 位寄存器，必须使用字指令对其访

增计数模式　当定时器由 CCR0 计数到 0 时，TAIFG 置位。

连续计数模式　当定时器由 0FFFFH 计数到 0 时，TAIFG 置位。

增/减计数模式　当定时器由 1 减计数到 0 时，TAIFG 置位。

计数器输入时钟源的选择与分频控制如图 3.34 所示。

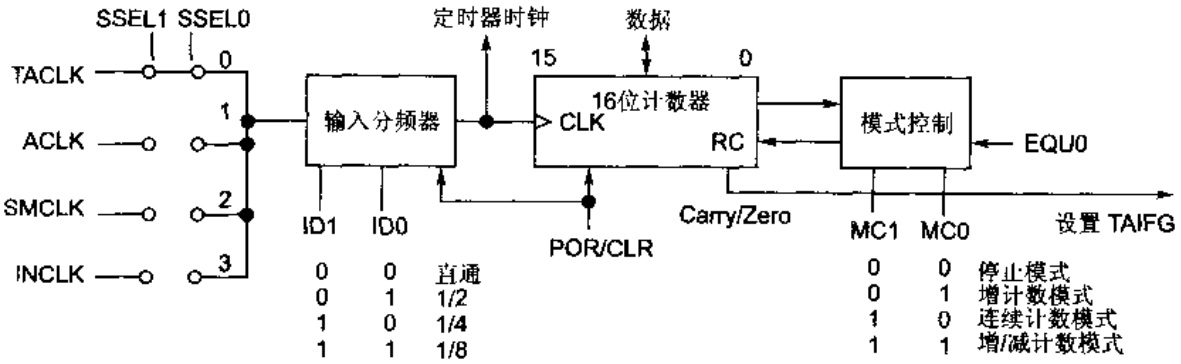


图 3.34　计数器输入时钟源的选择与分频控制

由 SSEL0 和 SSEL1 两位选择时钟源，然后再由 ID0 和 ID1 选择分频系数将输入信号分频，分频后的信号才用于计数器计数。在 MSP430F4XX 系列器件中，INCLK 信号经过反向驱动之后再送入，这一点与其他器件有点差别。

由此可见，TACTL 寄存器几乎控制了 Timer_A 的第一部分 —— 计数器部分。

对于在该控制寄存器中决定的定时器工作模式在后面详细介绍。

(2) TAR　16 位计数器内容

该单元就是执行计数的单元，是计数器的主体。其内容可读可写，但要注意：当计数时钟不是 MCLK 时，写入应该在计数器停止计数时写，因为它与 CPU 时钟不同步。

(3) CCTLx　捕获/比较控制寄存器

由于 Timer_A 有多个捕获/比较模块，每个模块都有自己的控制字 CCTLx。这里 x 为捕获/比较模块序号，在寄存器中各位名称中的 x 也一样。该寄存器在 POR 信号后全部复位，但在 PUC 信号后不受影响。该寄存器中各位的定义如下(空格表示未用)：

15　14	13　12	11	10	9	8	7　6　5	4	3	2	1	0
CAPTMOD1～0	CCIS1～0	SCS	SCCI		CAP	OUTMODx	CCIEx	CCIx	OUT	COV	CCIFGx

CAPTMOD1～0　该两位选择捕获模式(Capture Mode)。

00　禁止捕获模式；

01　上升沿捕获；

10　下降沿捕获；

11　上升沿和下降沿都捕获。

CCIS1～0　输入选择。在捕获模式中选择捕获事件的输入信号源，在比较模式不用。

00　选择 CCIxA 为捕获事件的输入信号源；

01　选择 CCIxB 为捕获事件的输入信号源；

10　选择 GND 为捕获事件的输入信号源；

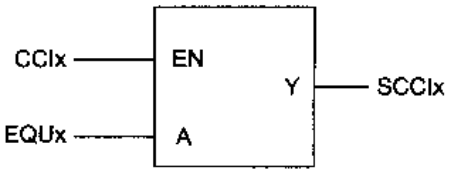
11　选择 V_{cc} 为捕获事件的输入信号源。

SCS　用于捕获输入信号与定时器时钟信号同步。

0　异步捕获；

1　同步捕获。

SCCIx　右图说明了 SCCIx 的含义：比较器相等信号将选定的输入信号 CCIx（CCIxA，CCIxB，V_{cc} 和 GND）锁存在锁存器中，由 SCCIx 输出。



CAP　模式选择位。选择捕获模式还是比较模式。

0　该模块工作在比较模式；

1　该模块工作在捕获模式。

OUTMODx　该 3 位选择输出模式。输出模式在后面输出单元部分将详细介绍。

000　输出；

001　置位；

010　PWM 翻转/复位；

011　PWM 置位/复位；

100　翻转；

101　复位；

110　PWM 翻转/置位；

111　PWM 复位/置位。

CCIEx　中断允许位。该位决定相应的捕获/比较模块能否提出中断请求。

0　禁止；

1　允许。

CCIx　捕获/比较模块的输入信号。由 CCIS0 和 CCIS1 选择的输入信号可通过该位读出。

OUT　输出信号。如果 OUTMODx 选择输出模式 0（输出），则该位决定输出到 OUTx 的具体信号。

COV　捕获溢出标志。在比较模式下（CAP＝0），捕获信号复位，捕获事件不会使 COV 置位。在捕获模式下（CAP＝1），如果捕获寄存器的值被读出前再次发生捕获事件，则 COV 置位。COV 在读捕获值时不会复位，须用户软件复位。

CCIFGx　各模块的捕获比较中断标志。具体含义为：捕获模式时，CCIFGx＝1 表示在寄存器 CCRx 中捕获了定时器 TAR 中的值；比较模式时，CCIFGx＝1 表示定时器 TAR 中的值等于寄存器 CCRx 中的值。

而在 3 个中断标志中，CCIFG0 在被中断服务时能自动复位；CCIFG1 和 CCIFG2 两位在读中断向量字 TAIV 后，自动复位。如果不访问 TAIV 寄存器，则不能自动复位，须用户软件清零；如果相应的中断允许位复位（不允许中断），则将不会产生中断请求，但中断标志位仍存在（CCIFGx＝1），这时须用户软件清除。

(4) CCRx　捕获/比较寄存器

在捕获/比较模块中，可读可写。

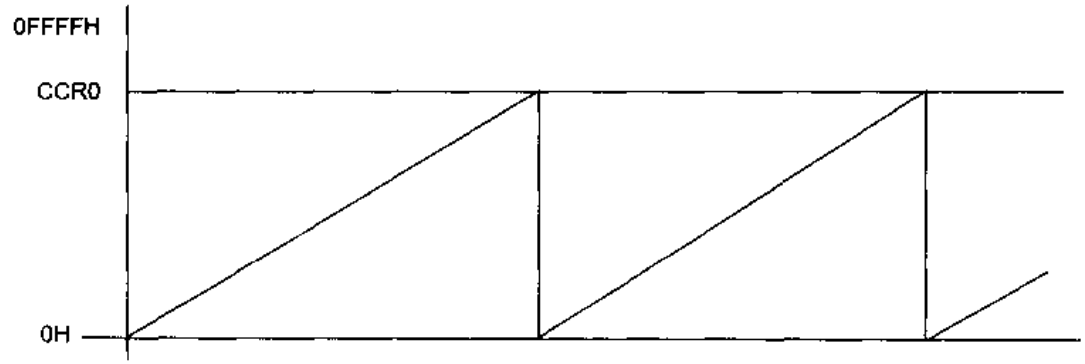


图 3.35　增计数模式的计数器 TAR

而标志位的设置过程如图 3.36 所示。当定时器的值等于 CCR0 的值时，设置标志位 CCIFG0 为 1，而当定时器从 CCR0 计数到 0 时，设置标志位 TAIFG 为 1。

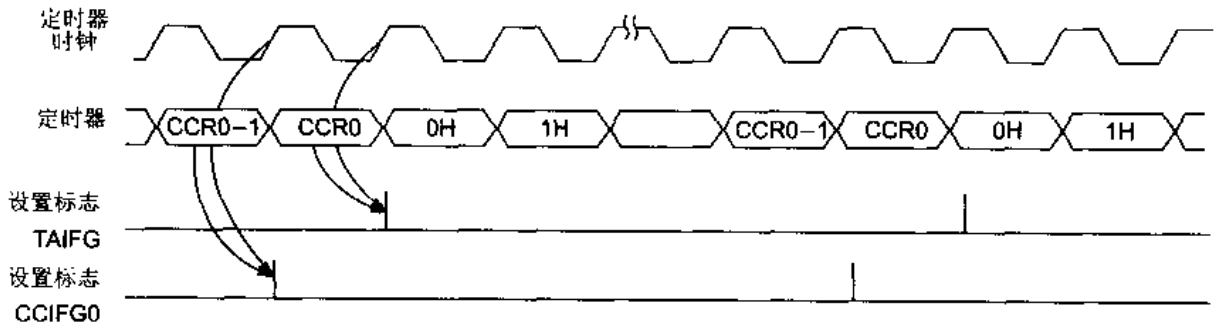


图 3.36　增计数模式的标志位设置

(3) 连续计数模式

在需要 65 536 个时钟周期的定时应用场合常用这种模式。其典型的应用是产生多个独立的时序信号。在这种计数模式中，CCR0 的工作方式与其他比较寄存器的工作方式相同。利用捕获/比较寄存器和各输出单元的输出模式，可以捕获各种外部事件发生的定时器数据(事件发生的时间)或者产生不同类型的输出信号。

连续计数模式的计数器活动规则：定时器从它的当前值开始计数，当计数到 0FFFFH 后，又从 0 开始重新计数，如图 3.37 所示。

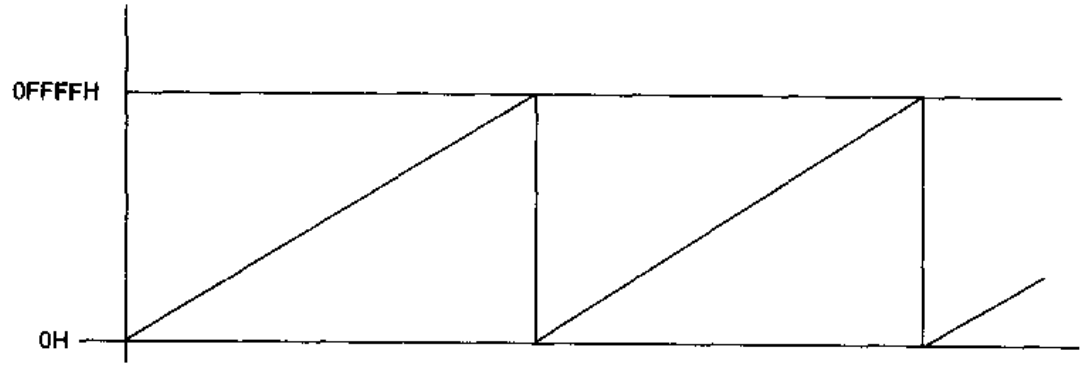


图 3.37　连续计数模式计数器 TAR

在这种模式下标志位的设置过程如图 3.38。当定时器从 0FFFFH 计数到 0 时，设置标志位 TAIFG。

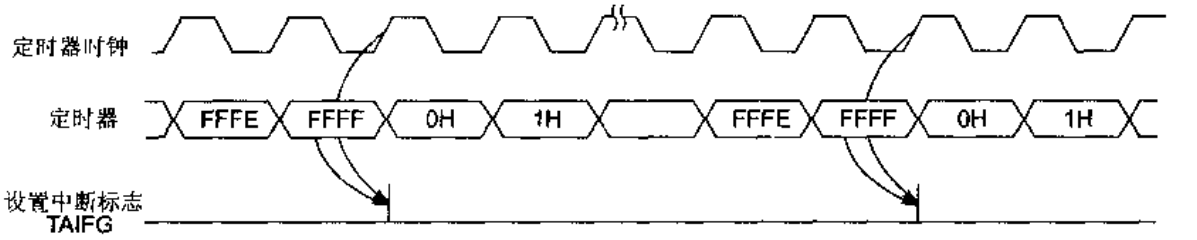


图 3.38 连续计数模式的标志位设置

如果相应的中断允许，则每当一个定时间隔到都会产生中断请求。那么在连续模式下，须将下一事件发生的时间在当前的中断程序中加到 CCRx 中。由图 3.39 可以看出，每隔 Δt 产生中断，须在定时器等于 CCR0a 时产生的中断服务程序中，将 CCR0b 加到 CCR0 寄存器中。

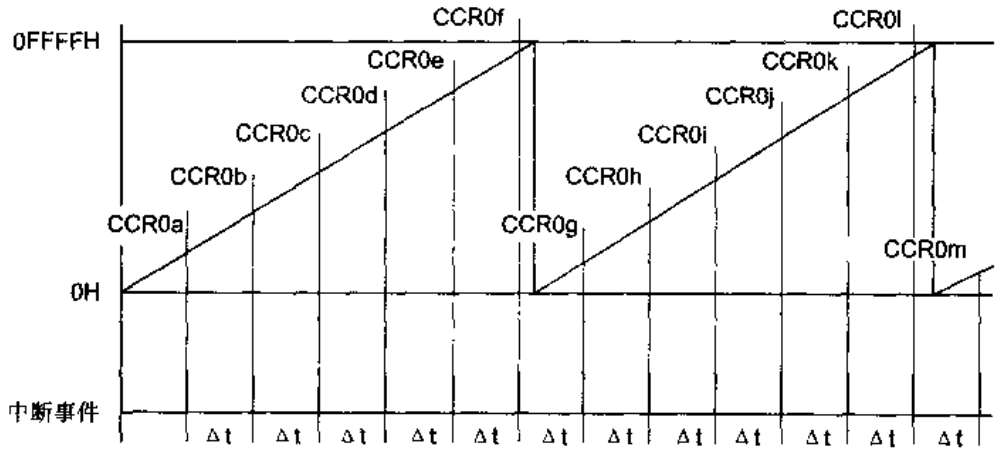


图 3.39 在连续计数模式时中断与 CCRx 的关系

(4) 增/减计数模式

在增/减计数模式下，计数器 TAR 的值先增后减：当增计数到 CCR0 的值时，计数器停止增计数，变为减计数；当减计数到 0 时，设置标志位 TAIFG。由此可见，这种模式的计数周期为 CCR0 值的 2 倍。所以常用于须得到对称波形的场合。这种模式时计数器中数值的变化情况如图 3.40 所示。

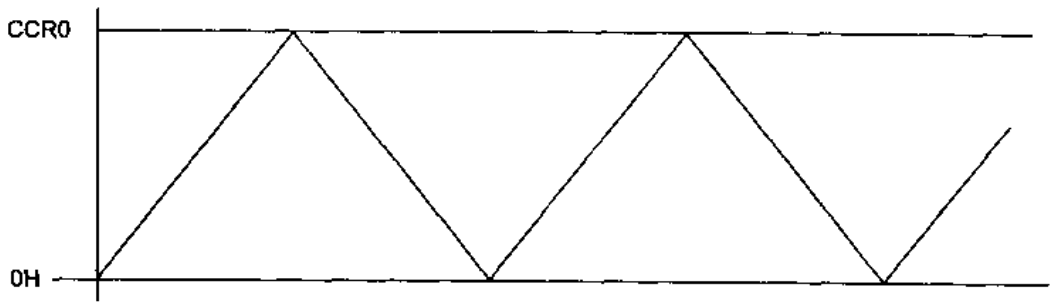


图 3.40 增/减计数模式时的计数器 TAR

增/减计数模式时，中断标志 CCIFG0 或 TAIFG 会在相等的时间间隔置位。在一个完整的周期中，每个标志位只置位一次，分别在半周期时发生。当定时器 TAR 的值从 CCR0－1 增计数到 CCR0 时，中断标志 CCIFG0 置位；当定时器从 1 减计数到 0 时，中断标志 TAIFG 置位。图 3.41 说明了标志位何时设置。从图中可以看出，TAR 的值是先增加再减少。

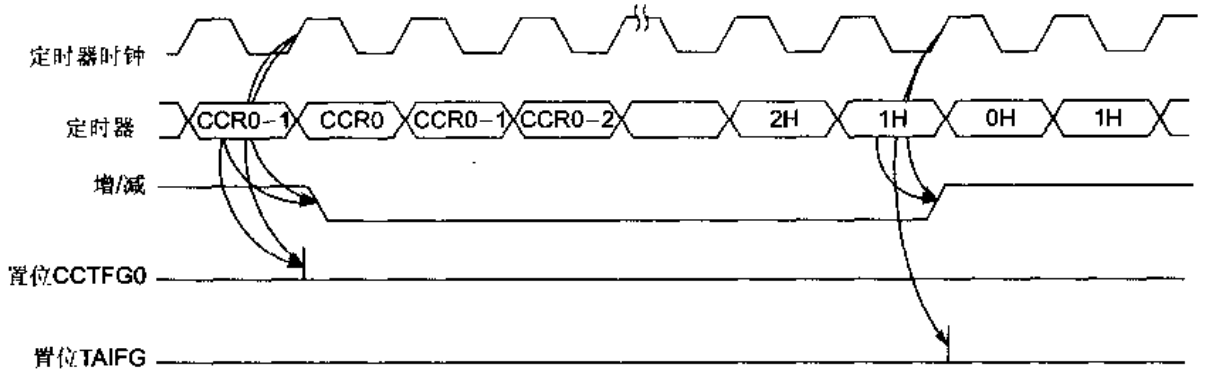


图 3.41　在增/减计数模式下标志位的设置

(5) 在各种模式下改变定时周期的情况

当定时器工作时，如果改变 CCR0 的值，会使情况发生变化，这变化在各种模式下是不一样的。

- 在停止模式下，定时器没有计数，其值不会发生改变。
- 在增计数模式下，有以下两种情况：

当新周期大于旧周期时，定时器会在等于旧周期之前增计数到新周期。

当新周期小于旧周期时，更新 CCR0 时的定时器时钟相位将影响定时器响应新周期：在定时器时钟为高时写入 CCR0 新的小周期，则定时器在下一个时钟上升沿返回到 0；如果在定时器时钟为低时写入 CCR0 新的小周期，则在定时器接受新周期并返回到 0 之前，继续增加一个时钟周期，见图 3.42。

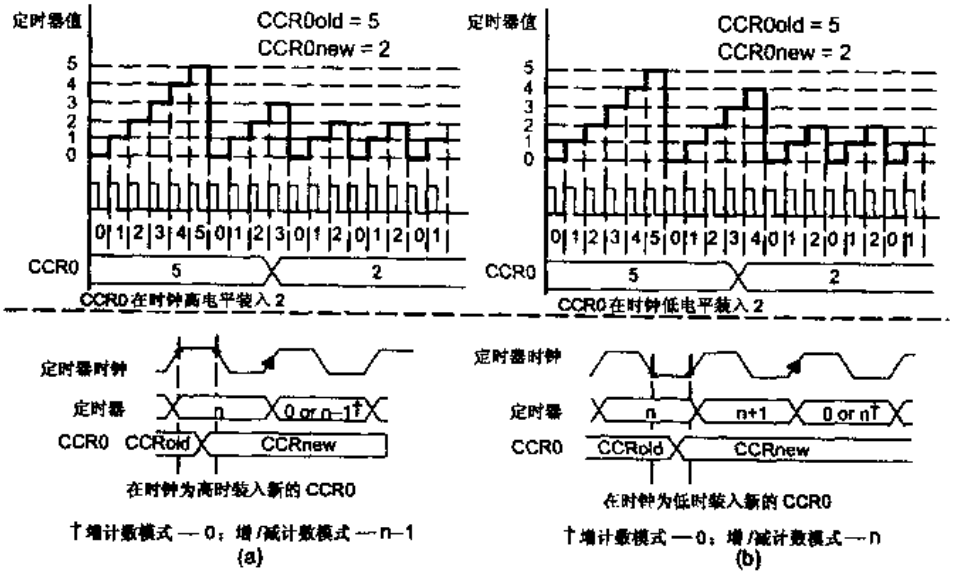


图 3.42　新周期小于旧周期定时器的值变化的情况

- 在连续计数模式下，一定要改变 CCR0 的值，因为这种模式为定时器连续增计数，要产生一定的时间间隔，因此要不断的增加 CCR0 的值，以使定时器每到一定的时间间隔都能与 CCR0 的内容相等，以便能引起标志位设置。
- 在增/减计数模式下，修改 CCR0 的值。在增计数时与增计数模式完全相同；在减计数时，新周期在减计数完成(减计数到 0)后才起作用。见图 3.43。

当捕获完成后，中断标志位 CCIFGx 将被置位。如果总的中断允许位 GIE 允许，相应的中断允许位 CCIEx 也允许，则将产生中断请求。

(2) 比较模式

当 CCTLx 中的 CAPx=0，该模块工作在比较模式。这时与捕获有关的硬件停止工作，在计数器 TAR 中计数值等于比较器中的值时设置标志位，产生中断请求；也可结合输出单元产生所需要的信号。

3 个捕获/比较器在比较模式时设置 EQUx 信号有差别：当 TAR 的值大于或等于 CCR0 中的数值时，EQU0 = 1；当 TAR 的值等于相应的 CCR1 或 CCR2 的值时，EQU1 = 1 或EQU2 = 1。

4. 输出单元

每个捕获/比较模块都包含一个输出单元，用于产生输出信号。每个输出单元有 8 种工作模式，可产生基于 EQUx 的多种信号。输出单元的结构及时序如图 3.45 所示。

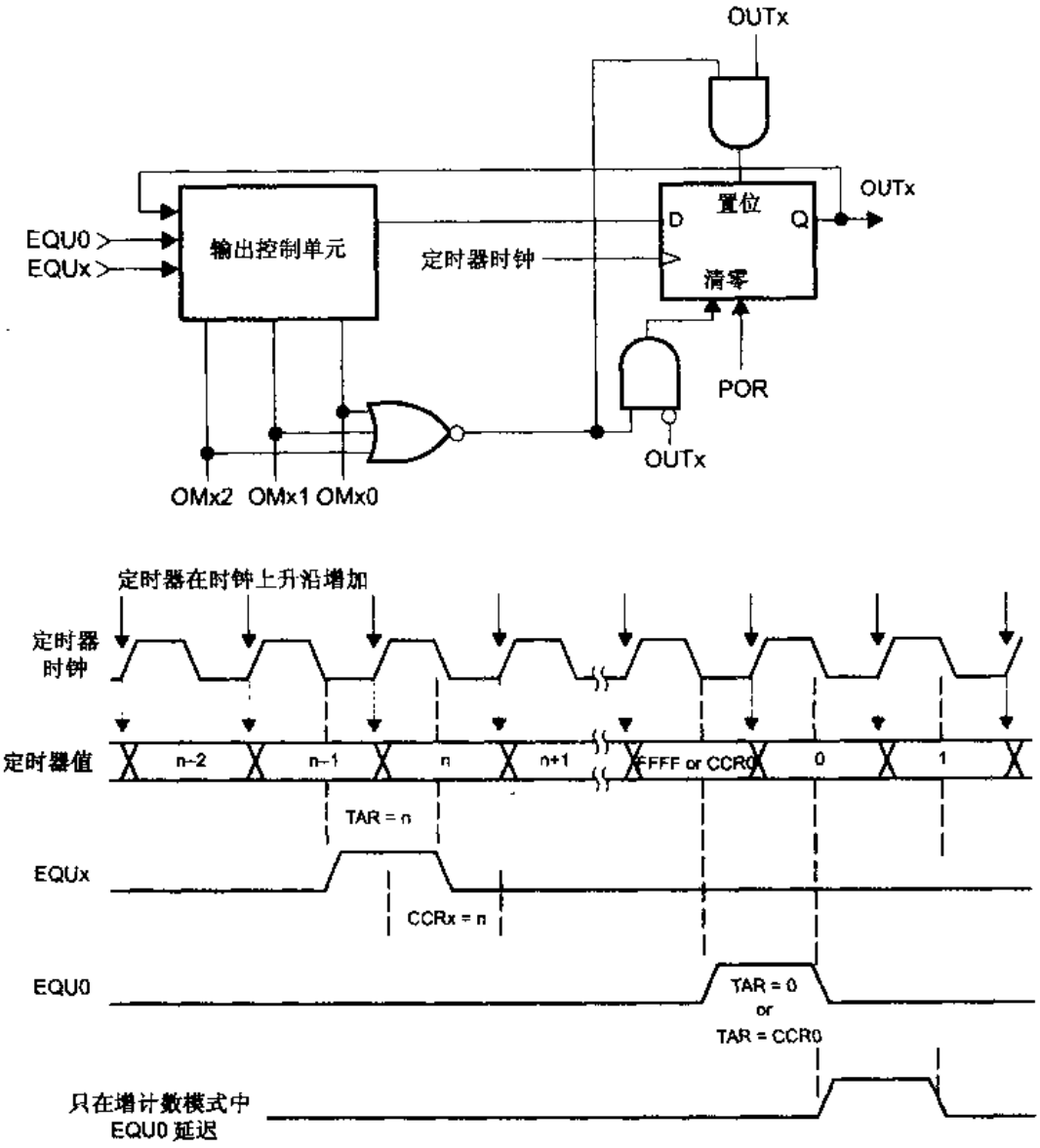


图 3.45　输出单元的结构及时序

从图中可以看出，最终的输出信号源于一个 D 触发器。该触发器的数据输入源于输出控制模块。输出控制模块又将 3 个输入信号(EQU0、EQU1/2 和 OUTx)经模式控制位 OMx0、OMx1 和 OMx2 运算后再输出到 D 触发器。D 触发器的置位端和复位端也都将影响到最终的输出。还可以看出：D 触发器的时钟信号为定时器时钟。在时钟为低电平时采样 EQU0 和 EQU1/2，在时钟的下一个上升沿锁存入 D 触发器中。

输出模式由模式控制位 OMx0、OMx1 及 OMx2 决定，共有 8 种输出模式，分别对应 OMx2、OMx1 及 OMx0 的值。比如 OMx2，OMx1，OMx0＝011 时为模式 3。除模式 0 外，其他的输出都在定时器时钟上升沿时发生变化。输出模式 2，3，6，7 不适合输出单元 0。所有的输出模式定义如下：

● 输出模式 0　输出模式。输出信号 OUTx 由每个捕获/比较模块的控制寄存器 CCTLx 中的 OUTx 位定义，并在写入该寄存器后立即更新。最终为 OUTx 直通。

● 输出模式 1　置位模式。输出信号在 TAR 等于 CCRx 时置位，并保持置位到定时器复位或选择另一种输出模式为止。

● 输出模式 2　PWM 翻转/复位模式。输出在 TAR 的值等于 CCRx 时翻转，当 TAR 的值等于 CCR0 时复位。

● 输出模式 3　PWM 置位/复位模式。输出在 TAR 的值等于 CCRx 时置位，当 TAR 的值等于 CCR0 时复位。

● 输出模式 4　翻转模式。输出电平在 TAR 的值等于 CCRx 时翻转，输出周期是定时器周期的 2 倍。

● 输出模式 5　复位模式。输出在 TAR 的值等于 CCRx 时复位，并保持低电平直到选择另一种输出模式。

● 输出模式 6　PWM 翻转/置位模式。输出电平在 TAR 的值等于 CCRx 时翻转，当 TAR 的值等于 CCR0 时置位。

● 输出模式 7　PWM 复位/置位模式。输出电平在 TAR 的值等于 CCRx 时复位，当 TAR 的值等于 CCR0 时置位。

输出单元在输出控制位的控制下，有 8 种输出模式输出信号。这些模式都与 TAR 的值、CCRx 的值及 CCR0 的值有关。

● 在增计数模式下，当 TAR 增加到 CCRx 或从 CCR0 计数到 0 时，OUTx 信号按选择的输出模式发生变化。实例如图 3.46 所示。

图中定时器工作在增计数模式下，CCR0 和 CCR1 的值为两根横线，斜线为 TAR 的值。图的下面为各种模式下的输出波形。

第 1 个波形为模式 1 的输出：开始时为低，当 TAR＝CCR1 时，输出高电平。

第 2 个波形为模式 2 的输出：当 TAR＝CCR1 时输出发生翻转；当 TAR＞CCR1 时输出高电平；当 TAR＜CCR1 时输出低电平。高低电平的时间由 CCR1 和 CCR0 两个寄存器的内容决定。TAR 的值在 0 与 CCR0 的值之间增加，输出波形的高低由 CCR1 和 CCR0 的内容决定；因此通过改变 CCR1 和 CCR0 的值就可改变输出波形的占空比，并输出脉冲宽度调制 PWM 波。

第 3 个波形为模式 3 的输出：可以看出，这种输出模式输出的波形与模式 2 的输出一样。输出在 TAR 的值等于 CCR1 时为高电平，直到 TAR 增加到 CCR0。

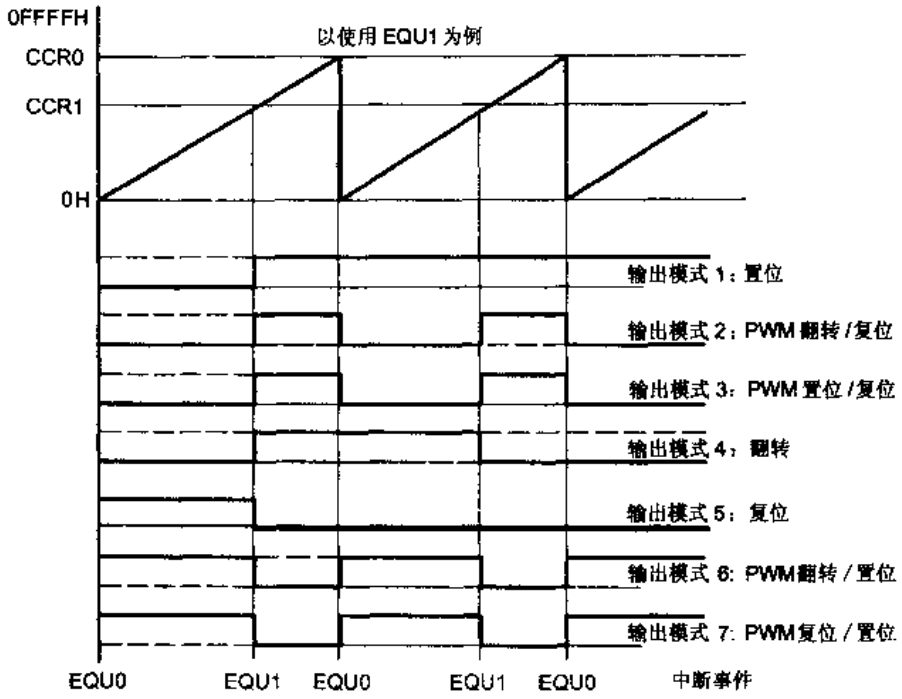


图 3.46　增计数模式时的输出实例

第 4 个波形为模式 4 的输出：当增计数到 CCR1 时，输出电平发生改变。当 TAR 增计数到 CCR1 时，输出为整个波形的一部分，整个波形的周期为定时器定时周期的 2 倍。

第 5 个波形为模式 5 的输出：这个输出波形与模式 1 的输出正好相反。当增计数到 CCR1 时输出复位，并一直保持。

第 6 个波形为模式 6 的输出：与模式 2 的输出波形正好相反。输出在计数值增到 CCR1 时为高电平，而在计数值从 CCR1 到 CCR0 的一段时间内为低电平。

第 7 个波形为模式 7 的输出：与模式 6 的输出波形一样。

● 在连续计数模式下的输出波形如图 3.47 所示。可以看出，其波形与增计数模式一样，只是计数器在增计数到 CCR0 后还要继续增计数到 0FFFFH，这样就延长了计数器计数到 CCR1 的数值后的时间。这是与增计数模式不同的地方。

● 在增/减计数模式下的输出实例如图 3.48。可以看出，这时的各种输出波形与定时器增计数模式或连续计数模式不同。当定时器在任意计数方向上等于 CCRx 时，OUTx 信号都按选择的输出模式发生改变。下面分别说明，这里以 CCR2 为例。

第 1 个波形为模式 1 的输出：与定时器的增计数模式或连续计数模式的输出波形相同。

第 2 个波形为模式 2 的输出：当 TAR=CCR2 时输出发生翻转，从 0 增计数到 CCR2 时及从 CCR0 减计数到 CCR2 时都翻转。当 CCR0>TAR>CCR2 时输出高电平；当 0<TAR<CCR2 时输出低电平。高低电平的时间由 CCR2 和 CCR0 两个寄存器的内容决定。定时器 TAR 的值在 0 与 CCR0 的值之间增加或在 CCR0 与 0 之间减小，输出波形的高低由 CCR2 和 CCR0 内容决定；因此通过改变 CCR2 和 CCR0 的值就可改变输出波形的占空比，并输出脉冲宽度调制 PWM 波。

第 5 个波形为模式 5 的输出：这个输出波形与模式 1 的输出正好相反。当增计数到 CCR2 时输出复位，并一直保持。

第 6 个波形为模式 6 的输出：与模式 2 的输出波形正好相反。输出在计数值增到 CCR0 时为高电平，并在计数值从 CCR2 增到 CCR0 及从 CCR0 减到 CCR2 的一段时间内都为高电平。

第 7 个波形为模式 7 的输出：当 TAR 的值等于 CCR2 的值时输出为 0，当 TAR 的值等于 CCR0 的值时输出为 1。

总结上述各种输出模式的输出与定时器的关系（见图 3.45）可以看出，整个输出单元包括：输入信号 EQU1/2 和 EQU0；3 个控制信号 OMx2、OMx1 及 OMx0；一个输出信号 OUTx。可将它简化为图 3.49。在时钟上升沿时，输出由这 5 个信号共同决定。表 3.13 为定时器时钟上升沿时 OUTx 在各种模式下的状态。

图 3.49　输出单元简图

表 3.13　定时器时钟上升沿时 OUTx 在各模式下的状态

输出模式	EQU0	EQUx	OUTx 状态（或触发器输入端 D）
0	X	X	X（OUTx 位）
1	X	0	OUTx（不变）
	X	1	1（置位）
2	0	0	OUTx（不变）
	0	1	OUTx（与以前相反）
	1	0	0
	1	1	1（置位）
3	0	0	OUTx（不变）
	0	1	1（置位）
	1	0	0
	1	1	1（置位）
4	X	0	OUTx（不变）
	X	1	OUTx（与以前相反）
5	X	0	OUTx（不变）
	X	1	0
6	0	0	OUTx（不变）
	0	1	OUTx（与以前相反）
	1	0	1
	1	1	0
7	0	0	OUTx（不变）
	0	1	0
	1	0	1
	1	1	0

5. 应用举例

定时器的编程应用一般要注意这样一些问题：

- 定时器的计数信号源，由 TPSSEL1 和 TPSSEL0 选择 TACLK，ACLK，SMCLK 或INCLK；
- 分频系数，由 ID1 和 ID0 选择 1，2，4 或 8 分频；
- 计算捕获/比较寄存器的值，由应用要求决定；
- 定时器、捕获/比较寄存器及输出单元模块等的工作模式，由应用要求决定。

[例 1] 利用定时器 A 输出周期为 2 ms 的方波，并由端口 P1.0 送出。

使用在 P1.0 端口每隔 1 ms 将输出求反的方式实现题目的要求。也就是说，定时器要产生的定时时间为 1 ms。在默认条件下，DCO 产生的频率为 800 kHz。使用 800 kHz 的 SMCLK 作为定时器的时钟源，则定时 1 ms 的计数值为

$$1\times10^{-3}\div(1\div(800\times1000))=800$$

这里分别使用增计数模式和连续计数模式来实现，虽然这两种模式有差别，但其结果是一样的：在示波器上，两个程序于 P1.0 上输出的波形完全相同。下面是使用增计数模式的程序。

```
#include    "msp430x11x1.h"
            ORG     0F000h                  ;程序开始处
RESET       mov.w   #WDTPW+WDTHOLD,&WDTCTL  ;停看门狗
SetupTA     mov.w   #TASSEL1+TACLR,&TACTL   ;使用 SMCLK 信号，清除 TAR
SetupC0     mov.w   #CCIE,&CCTL0            ;使能 CCR0 中断
            mov.w   #800,&CCR0              ;设置比较值
SetupP1     bis.b   #001h,&P1DIR            ; P1.0 设为输出
            bis.w   #MC0,&TACTL             ;以增计数模式开始定时器 A
            eint                            ;使能中断
Mainloop    bis.w   #CPUOFF,SR              ;此时 CPU 已经不需要了，关掉
            nop                             ;
TA0_ISR     xor.b   #001h,&P1OUT            ; P1.0 输出反向
            reti                            ;中断返回
            ORG     0FFFEh                  ; MSP430 RESET 向量
            DW      RESET                   ;
            ORG     0FFF2h                  ; Timer_A0 向量
            DW      TA0_ISR                 ;
            END
```

可以发现：在进入中断时，TAR 的值都是 0。因为在增计数模式下，只有当 TAR 的值由 CCR0 计数到 0 时，才设置标志位，才能引起中断。

下面的程序用连续计数模式实现同样的目的。

```
            ORG     0F000h                  ;程序开始处
RESET       mov.w   #300h,SP                ;初始化堆栈指针
StopWDT     mov.w   #WDTPW+WDTHOLD,&WDTCTL  ;停看门狗
SetupTA     mov.w   #TASSEL1+TACLR,&TACTL   ;使用 SMCLK 信号，清除 TAR
SetupC0     mov.w   #CCIE,&CCTL0            ;使能 CCR0 中断
```

```
            mov. w   #800,&CCR0                ;设置比较值,也是第一次中断的时间
SetupP1     bis. b   #001h,&P1DIR              ; P1.0 设为输出
            bis. w   #MC1,&TACTL               ;以连续模式开始定时器 A
            eint                               ;使能中断
Mainloop    bis. w   #CPUOFF,SR                ;此时 CPU 已经不需要了,关掉
TA0_ISR     xor. b   #001h,&P1OUT              ;定时器中断服务:P1.0 输出反向
            add. w   #800,& CCR0               ;在连续模式中要加上定时值
            reti                               ;
            ORG      0FFFEh                    ;
            DW       RESET                     ;
            ORG      0FFF2h                    ; Timer_A0 中断向量
            DW       TA0_ISR                   ;
            END
```

这段程序与第一段程序差别在于:定时器 TAR 的值在满足中断条件时是跳跃式改变,而不是第一个程序中的每一次中断到时 TAR 都是 0。

[例 2]　利用定时器 A 输出周期为 2 ms 的占空比为 25%和 75%的矩形波。

占空比为 25%和 75%的矩形波实质就是 PWM 波形,这里充分利用定时器 A 的输出模块的功能,输出 PWM 波。器件还是使用 MSP430F1101,其 P1.2 和 P1.3 为定时器 A 的输出模块 OUT1 和 OUT2 的输出引脚(请参见器件手册)。注意,该两引脚必须定义为模块方式。

```
#include   "msp430x11x1.h"
           ORG      0F000h                    ;程序开始处
RESET      mov. w   #300h,SP                  ;堆栈指针初始化
StopWDT    mov. w   #WDTPW+WDTHOLD,& WDTCTL   ;关掉看门狗
SetupTA    mov. w   #TASSEL1+TACLR,&TACTL     ;使用 SMCLK 时钟,清除 TAR 值
SetupC0    mov. w   #512-1,&CCR0              ;确定 PWM 周期
SetupC1    mov. w   #OUTMOD_7,&CCTL1          ;捕获/比较模块 1 的输出为模式 7
           mov. w   #384,&CCR1                ;确定 CCR1 PWM 高电平时间(占空比)
SetupC2    mov. w   #OUTMOD_7,&CCTL2          ;捕获/比较模块 2 的输出也为模式 7
           mov. w   #128,&CCR2                ;确定 CCR2 PWM 高电平时间(占空比)
SetupP1    bis. b   #00Ch,&P1DIR              ; P1.2 和 P1.3 定义为输出
           bis. b   #00Ch,&P1SEL              ; P1.2 和 P1.3 为定时器 A 的输出引脚
           bis. w   #MC0,&TACTL               ; Timer_A 工作在增计数模式
Mainloop   bis. w   #CPUOFF,SR                ; CPU 不需要了,信号自动由引脚输出
           ORG      0FFFEh                    ;
           DW       RESET                     ;
           END
```

3.3.6　16 位定时器 B

16 位定时器 B(Timer_B)在 MSP430F13X/14X 和 MSP430F43X/44X 等器件中出现,其他器件里没有。而在不同器件中,Timer_B 所带的捕获/比较模块也不一样,例如:MSP430F13X 有 3 个捕获/比较寄存器,因此常称 MSP430F13X 中的 Timer_B 为 Timer_B3;MSP430F14X 有 7 个捕获/比较寄存器,因此常称 MSP430F44X 中的 Timer_B 为 Timer_B7。

Timer_B 与 Timer_A 的功能基本相同，但在 Timer_B 中没有 SCCI 位。Timer_B 有如下一些特点：

- 有 4 种工作模式，有 4 种可选计数长度分别为 8、10、12 或 16 位；
- 具有可选、可配置的计数器输入时钟源；
- 有 7 个独立可配置的带可配置输入与双缓冲比较寄存器的捕获/比较寄存器；
- 有 7 个具有 8 种输出模式的可配置输出单元。

图 3.50 为 Timer_B 的结构原理图

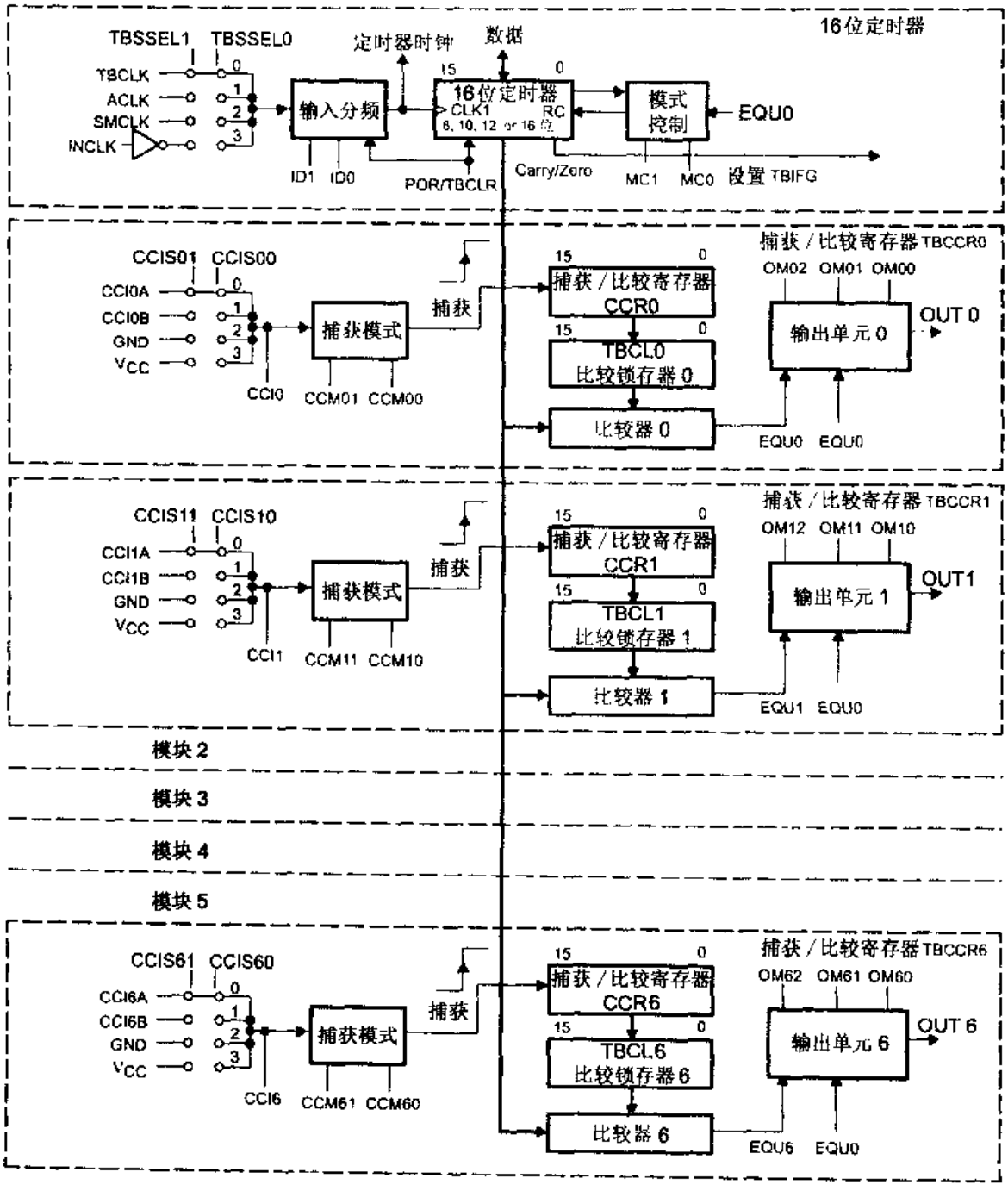


图 3.50 Timer_B 的原理结构图

Timer_B 与 Timer_A 在结构上基本相同，但比 Timer_A 稍微复杂一点，因此功能有所增强，主要有以下差异：

- Timer_B 的计数长度为 8、10、12 和 16 位可编程，而 Timer_A 固定为 16 位。
- Timer_B 没有实现 Timer_A 中的 SCCI 位的功能。
- Timer_B 在比较模式下的捕获/比较寄存器功能与 Timer_A 不同，增加了比较锁存器。
- 有的型号芯片的 Timer_B 输出实现了高阻态。
- 比较模式的原理也稍有不同：在 Timer_A 中，CCRx 寄存器中保存与定时器 TAR 相比较的数据；而在 Timer_B 中，CCRx 寄存器中保存的是要比较的数据，但定时器 TBR 并不与它相比较，而是将 CCRx 送到相应的锁存器之后，由锁存器与定时器 TBR 比较。
- 增加的比较数据锁存器能够让设计者设计出较精确的时间控制器件。可让多个比较器成组工作，同时发生比较器的更新。
- Timer_B 具有中断能力（这是定时器的共性）。中断源可以是定时器的溢出，也可以来自具有捕获/比较功能的捕获/比较寄存器。7 个捕获/比较模块的每一个都可独立编程，且输入信号可源自内部或外部，外部信号可上升沿、可下降沿及任意沿。

由于 Timer_B 的以上特性，Timer_B 的使用比 Timer_A 更灵活。可支持多个同时进行的时序控制、多捕获/比较及多输出波形或这些功能的组合。此外，由于比较数据的双缓冲能力，因此多个 PWM 波形可同步地产生。

下面介绍 Timer_B 的寄存器，Timer_B 的所有寄存器在调试软件中如图 3.51 所示。

图 3.51　Timer_B 的所有寄存器

1. Timer_B 的寄存器

从图 3.51 中可以看出，Timer_B7 共有 17 个寄存器，而 Timer_B3 的寄存器没有这么多，只有 9 个。其实它们当中的很多是一种类型或一个原理，因此这 17 个寄存器只相当于 5 个：TBCTL（Timer_B 控制寄存器），TBR（Timer_B 计数器），TBCCTLx（Timer_B 捕获/比较控制寄存器），TBCCRx（Timer_B 捕获/比较寄存器）和 TBIV（Timer_B 中断向量寄存器）。此外，还有 7 个比较锁存器 TBCLx 用户不可见。这些寄存器都是 16 位的，因此必须用字访问方式访问。

（1）TBCTL　控制寄存器

Timer_B 的全部控制都集中在这个寄存器中，在 POR 信号后该寄存器的所有位全部自动复位，但在 PUC 后不受影响。该寄存器各位的定义如下（空格表示未用）：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	TBCLGRP1～0		CNTL1～0			SSEL1～0		ID1～0		MC1～0			TBCLR	TBIE	TBIFG

TBCLGRP1～0 该两位决定单独还是成组装载比较锁存器，而装载信号被相应的捕获/比较控制寄存器中的 CLLDx 位控制。

00 单独装载

比较锁存器由各自相应的 TBCCTLx 寄存器的 CLLDx 位控制。

01 分为 3 组装载

TBCTL1＋TBCTL2：TBCCTL1 中的 CLLD 位定义操作模式；

TBCTL3＋TBCTL4：TBCCTL3 中的 CLLD 位定义操作模式；

TBCTL5＋TBCTL6：TBCCTL5 中的 CLLD 位定义操作模式。

10 分 2 组装载

TBCTL1＋TBCTL2＋TBCTL3：TBCCTL1 中的 CLLD 位定义操作模式；

TBCTL4＋TBCTL5＋TBCTL6：TBCCTL4 中的 CLLD 位定义操作模式。

11 选择 1 组装载

TBCTL1～TBCTL6 为一组：TBCCTL1 中的 CLLD 位定义操作模式。

CNTL1～0 该两位配置定时器的 4 种定时长度。

00 16 位，最大计数值为 0FFFFH；

01 12 位，最大计数值为 0FFFH；

10 10 位，最大计数值为 03FFH；

11 8 位，最大计数值为 0FFH。

SSEL1～0 选择进入输入分频器的输入时钟源。

00 选择 TBCLK(外部引脚信号)作为输入源；

01 选择 ACLK(辅助时钟)作为输入源；

10 选择 MCLK(主时钟)作为输入源；

11 选择 INCLK(外部时钟)作为输入源。

ID1～0 输入信号分频系数选择位。

00 选择直通；

01 选择 2 分频；

10 选择 4 分频；

11 选择 8 分频。

MC1～0 计数模式控制位。

00 停止模式；

01 增计数模式；

10 连续计数模式；

11 增/减计数模式。

TBCLR 清除位。定时器和输入分频器在POR信号或TBCLR=1时被复位。

0 定时器保持原状；

1 定时器和输入分频器被复位。

TBIE 定时器溢出中断允许位。

0 禁止中断；

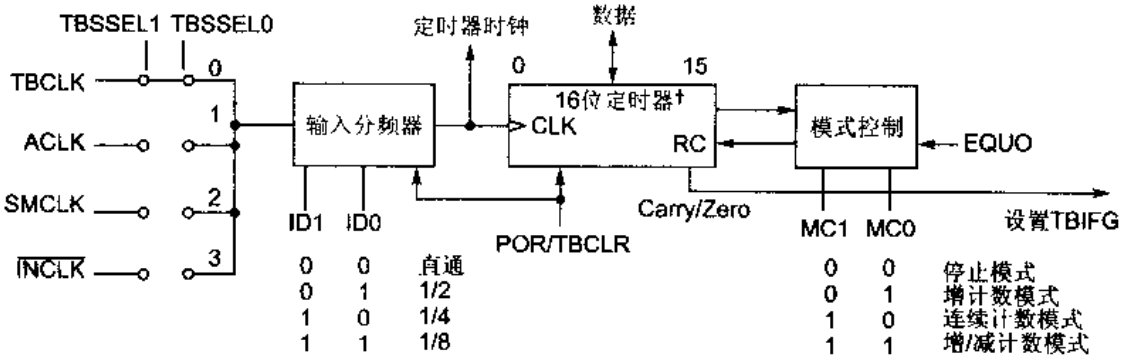
1 允许中断。

TBIFG 定时器溢出标志。

0 定时器没有溢出；

1 定时器溢出。

从寄存器TBCTL可以看出，Timer_B的大量控制位都集中在这里：选择定时器的位数、定时器的工作模式、输入信号的选择、输入信号的分频系数和中断的控制等。图3.52是定时器的详细结构。



†可选8，10，12和16位计数长度

图3.52 定时器的结构

(2) TBR 计数器

Timer_B定时器的计数值就在TBR寄存器中。它是一个16位计数器，但当计数长度被设置为11位、10位及8位时，就只能计数到相应的计数长度。定时器在各种模式下工作，还有定时器的其他模块，包括捕获/比较模块等，都要与TBR寄存器打交道。比较是与TBR比较，捕获是将TBR的内容捕获保存。

TBR是一个16位寄存器，须用字访问方式进行读写，但要写入TBR数据应该先停止定时器，这样写入数据才可靠，否则容易使TBR的值不可预测。

(3) TBCCTLx 捕获/比较控制寄存器

Timer_B有7个捕获/比较模块，每个模块都有自己的控制字TBCCTLx。该寄存器在POR信号后复位，而不受PUC信号影响。该寄存器为16位寄存器，须字访问方式对其进行读写。其中各位的定义如下：

15 14	13 12	11	10 9	8	7 6 5	4	3	2	1	0
CAPTMOD1～0	CCIS1～0	SCS	CLLD1～0	CAP	OUTMODx	CCIEx	CCIx	OUT	COV	CCIFG

CAPTMOD1～0 该两位选择捕获模式(Capture Mode)。

00 禁止捕获模式；

01 上升沿捕获；
10 下降沿捕获；
11 上升沿和下降沿都捕获。

CCIS1～0 输入选择。在捕获模式中选择捕获事件的输入信号源，在比较模式不用。
00 选择 CCIxA 为捕获事件的输入信号源；
01 选择 CCIxB 为捕获事件的输入信号源；
10 选择 GND 为捕获事件的输入信号源；
11 选择 V_{cc} 为捕获事件的输入信号源。

SCS 用于捕获输入信号与定时器时钟信号同步。
0 异步捕获；
1 同步捕获。

CLLD1～0 定义比较锁存器 TBCLx 的装载方式。
00 立即装载。
01 当定时器 TBR 计数到 0 时 TBCCRx 的数据装载到 TBCLx。
10 在增/减计数模式下，当 TBR 计数到 TBCL0 或 0 时，TBCCRx 的数据装载到 TBCLx；在连续计数模式下，当 TBR 计数到 0 时，TBCCRx 的数据装载到 TBCLx。两种模式稍有差别。
11 当 TBR 计数到 TBCL0 时，TBCCRx 的数据装载到 TBCLx。

CAP 模式选择位。选择捕获模式还是比较模式。
0 该模块工作在比较模式；
1 该模块工作在捕获模式。

OUTMODx 该 3 位选择输出模式。输出模式在后面将详细介绍。
000 输出；
001 置位；
010 PWM 翻转/复位；
011 PWM 置位/复位；
100 翻转；
101 复位；
110 PWM 翻转/置位；
111 PWM 复位/置位。

CCIEx 中断允许位。该位决定相应的捕获/比较模块能否提出中断请求。
0 中断禁止；
1 中断允许。

CCIx 捕获/比较模块的输入信号。由 CCIS0 和 CCIS1 选择的输入信号(CCIxA，CCIxB，V_{cc} 或 GND)可通过该位读出。

OUT 输出信号。如果 OUTMODx 选择输出模式 0(输出)，则该位决定输出到 OUTx 的具体信号。

COV 捕获溢出标志。在比较模式下(CAP=0)，捕获信号复位，捕获事件不会使 COV 置位；在捕获模式下(CAP=1)，如果捕获寄存器的值被读出前再次发生捕获事

(6) TBCLx 比较锁存器

Timer_B的比较锁存器锁存相应的TBCCRx的值。

2. 定时器模式

Timer_B与Timer_A一样有4种工作模式。由控制寄存器TBCTL中MC0和MC1两位决定(见TBCTL寄存器)。

(1) 停止模式

当MC1=MC0=0时,定时器暂停。暂停时定时器的值(TBR的内容)不受影响。当定时器在暂停后重新计数时,计数器将从暂停时的值开始以暂停前的计数方向计数。如果不能这样,则可通过TBCTL中CLR控制位来清除定时器的方向记忆特性。

(2) 增计数模式

当MC0=1,MC1=0时,定时器工作在增计数模式。该模式用于定时周期小于定时器B的最大计数值(最大计数值因定义的计数位数而异)的连续计数方式。捕获/比较寄存器TBCCR0的数据定义定时器的计数周期。

增计数模式的计数器活动规则:当计数器TBR增计数到TBCL0的值或当计数值与TBCL0的值相等(或定时器值大于TBCL0的值)时,定时器复位,并从0开始重新计数。图3.53说明了增计数模式的计数过程。注意,这里是与比较锁存器TBCL0的值相比较。以后的其他计数模式也一样。

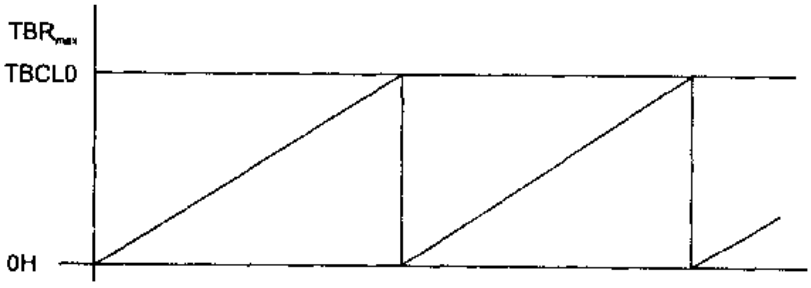


图 3.53 增计数模式的计数器TBR

标志位的设置过程如图3.54所示。当定时器的值等于TBCL0的值时,设置标志位CCIFG0为1,而当定时器从TBCL0计数到0时,设置标志位TBIFG为1。

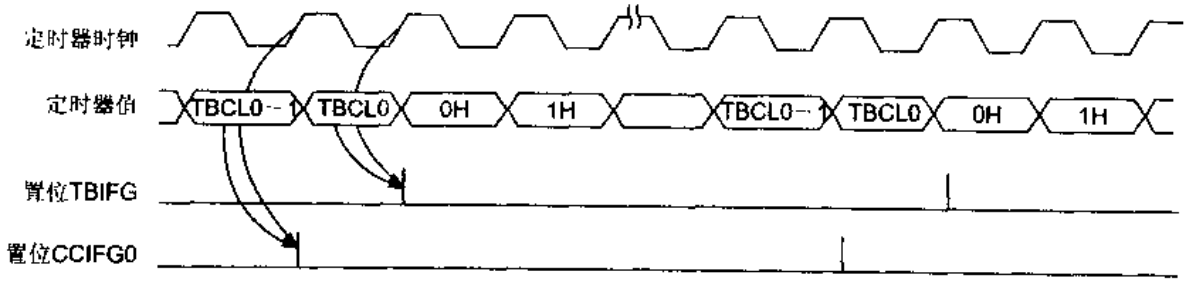


图 3.54 增计数模式的标志位设置

(3) 连续计数模式

在需要65 536个时钟周期的定时应用场合常用这种模式。其典型的应用是产生多个独立的时序信号。在这种计数模式下,TBCCR0的工作方式与其他比较寄存器的工作方式相同。利用捕获/比较寄存器与各输出单元的输出模式,可以捕获各种外部事件发生的定时器数

止增计数，变为减计数；当减计数到 0 时，设置标志位 TBIFG。由此可见这种模式的计数周期为 TBCL0 锁存器值的 2 倍。所以常用于须得到对称波形的场合。这种模式时计数器中数值变化情况如图 3.58 所示。

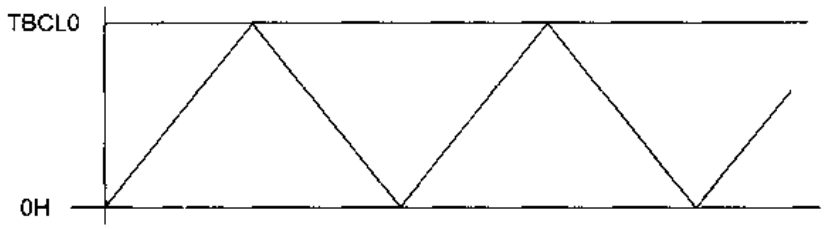


图 3.58 增/减计数模式时的计数器 TBR

增/减计数模式时，中断标志 CCIFG0 和 TBIFG 会在相等的时间间隔置位。在一个完整的周期中，每个标志位只置位一次，分别在半周期时发生。当定时器 TBR 的值从 TBCL0－1 增计数到 TBCL0 时，中断标志 CCIFG0 置位；当定时器从 1 减计数到 0 时，中断标志 TBIFG 置位。图 3.59 说明了标志位何时设置。从图中也可看出，计数器 TBR 的值是先增加再减少。

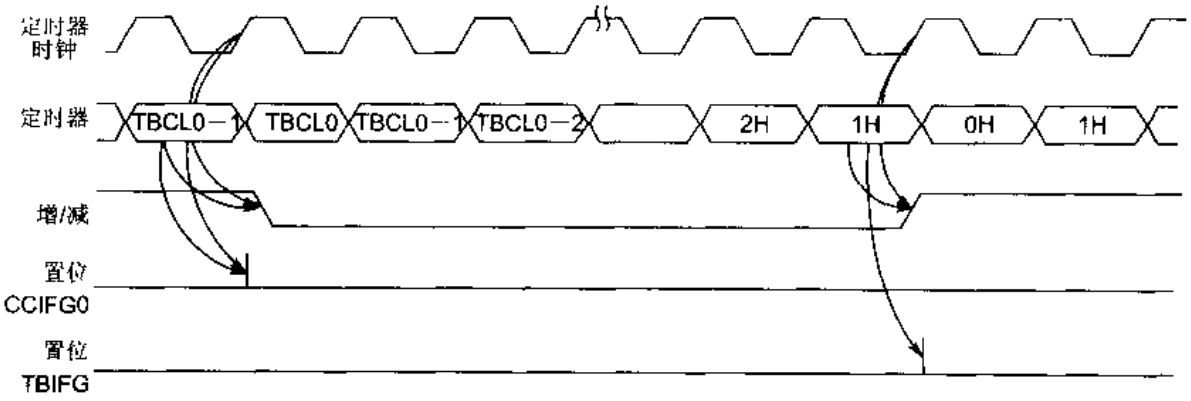


图 3.59 标志位在增/减模式下的设置

(5) 在各种模式下改变定时周期的情况

如果当定时器工作时，改变 TBCL0 的值，会使情况发生变化，这变化在各种模式下是不一样的。

- 在停止模式下，定时器没有计数，其值不会发生改变。
- 在增计数模式下，有以下两种情况：

当新周期大于旧周期时，定时器会在等于旧周期之前增计数到新周期。

当新周期小于旧周期时，更新 TBCL0 时的定时器时钟相位将影响定时器响应新周期：在定时器时钟为高时写入 TBCCR0 新的小周期，从 TBCCR0 传给 TBCL0，则定时器在下一个时钟上升沿返回到 0；如果在定时器时钟为低时写入 TBCCR0 新的小周期，并从 TBCCR0 传给 TBCL0，则在定时器接受新周期并返回到 0 之前，继续增加一个时钟周期，见图 3.60。

- 在连续计数模式下，一定要改变 TBCCR0 的值，因为这种模式为定时器连续增计数，要产生一定的时间间隔，因此要不断的增加 TBCCR0 的值，以使定时器每到一定的时间间隔都能与 TBCCR0 的内容相等，以便能引起标志位设置。

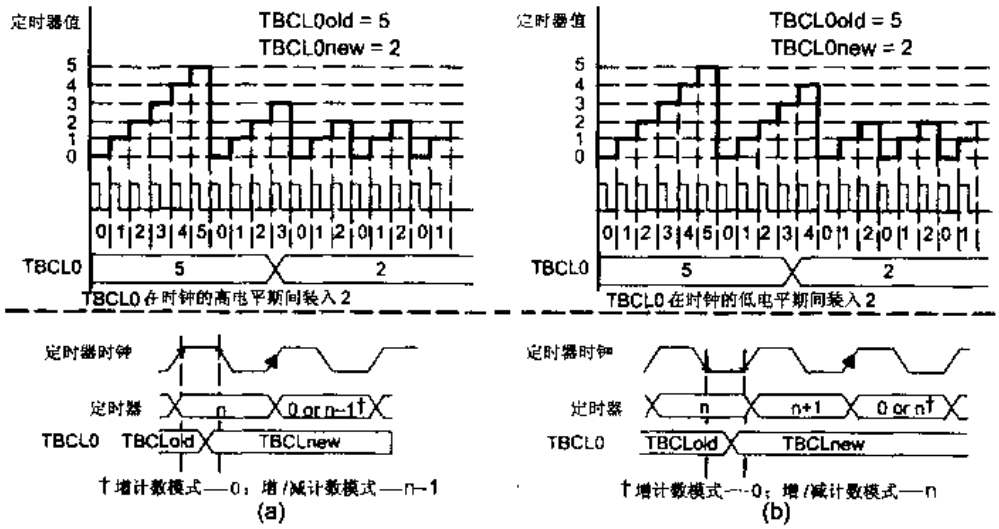


图 3.60　新周期小于旧周期定时器的值变化的情况

● 在增/减计数模式下，修改 TBCCR0 的值并写入 TBCL0 中。

在增计数时与增计数模式完全相同；在减计数时，新周期在减计数完成(减计数到 0)后才起作用，如图 3.61 所示。

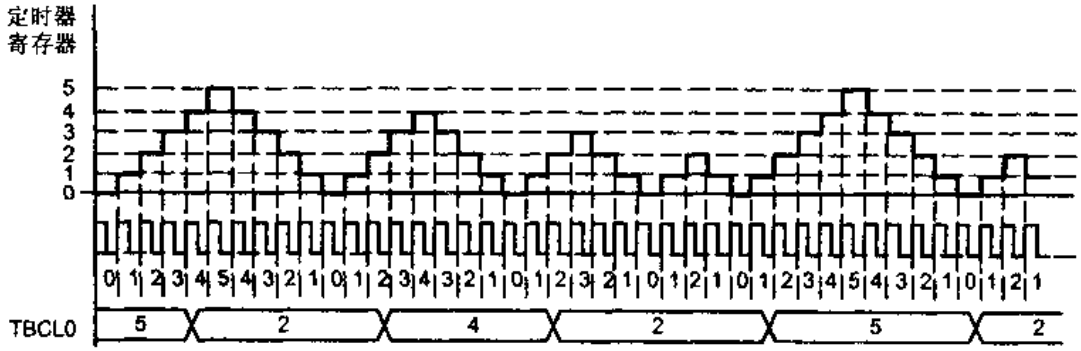


图 3.61　在增/减计数模式下修改定时周期

(6) Timer_B 计数模式总结

Timer_B 有 4 种计数模式，有 4 种可选的计数长度。4 种计数模式的含义与 Timer_A 基本相同。但在各种模式下，计数器 TBR 的内容与 TBCL0 相比较，而 TBCL0 的数据来源于 TBCCR0，由 TBCCR0 装载到 TBCL0，而在什么情况下装载由相应的控制位(CLLD1～0)决定。Timer_B 的 4 种计数长度，也能使应用更加灵活。这是与 Timer_A 不同的地方，同时也是 Timer_B 的优势所在。

3. 捕获/比较模块

该定时器有 7 个相同的捕获/比较模块，每个模块都可用于捕获事件发生的时间或产生一定的时间间隔，它为实时处理提供了灵活的手段。当发生捕获事件或定时时间到都将引起中断。该模块可用于捕获模式，也可用于比较模式。用 TBCCTLx 中的 CAPx 选择模式，用 CC-

Mx1 和 CCMx0 选择捕获条件(见本小节面寄存器部分)。捕获/比较模块的结构如图 3.62 所示。

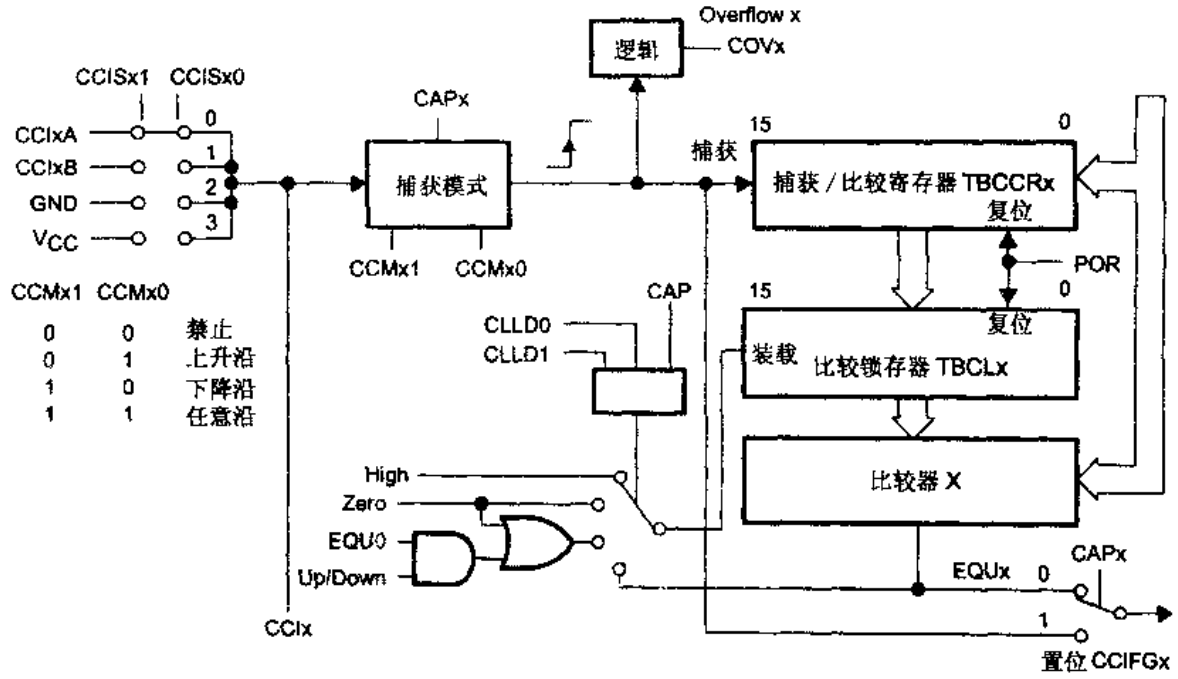


图 3.62 捕获/比较模块的逻辑结构

图中输入信号由 CCISx1 和 CCISx0 选择。输入信号可以来自外部引脚,也可来自内部信号,还可由 CCIx 信号输出。有一个比较器、一个比较锁存器及一个捕获/比较寄存器与定时器总线相连;可在满足捕获条件时,将 TBR 的值写入捕获寄存器;可在 TBR 的值与比较锁存器的值相等时,设置标志位。模式控制位 CAPx 不但决定该模块的工作模式,也决定标志位由什么信号设置,见图 3.62 的右下方部分。

(1) 捕获模式

当 TBCCTLx 中的 CAPx=1,该模块工作在捕获模式。这时如果在选定的引脚上发生选定的脉冲触发沿(上升沿、下降沿或任意跳变),则 TBR 中的值将写入到 TBCCRx 中。因此,常将此模式用于确定事件发生的时间,如速度的计算和时间的测量等。

捕获的时序如图 3.63 所示。这里要注意的是捕获的同步问题。

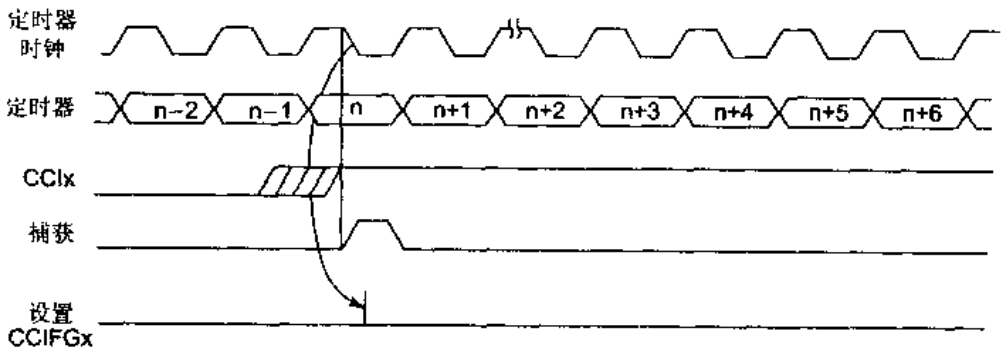


图 3.63 捕获功能时序

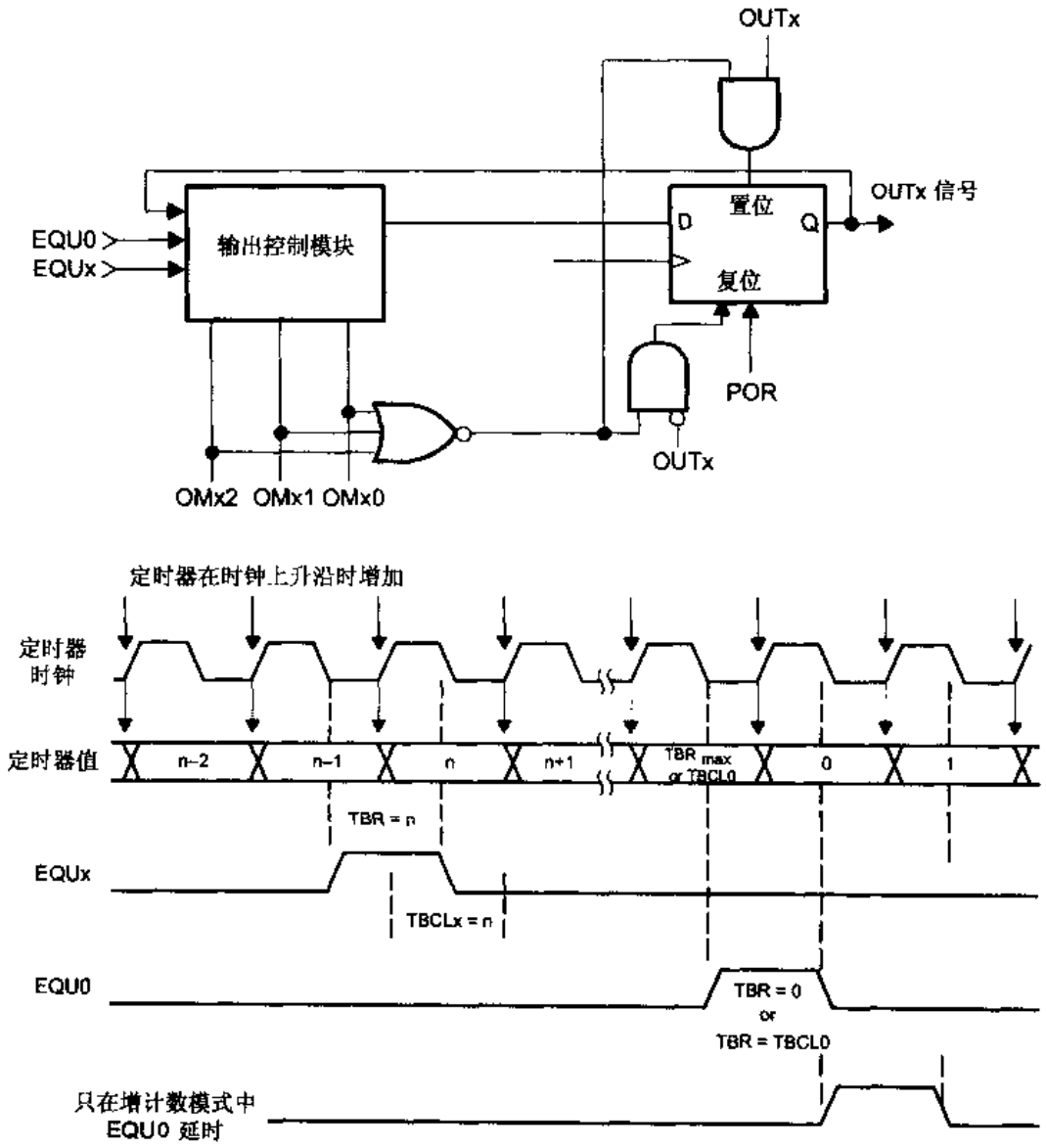


图 3.64 输出单元的结构以及时序

OMx1 及 OMx2 运算后输出到 D 触发器。D 触发器的置位端和复位端也都将影响到最终的输出。还可以看出：D 触发器的时钟信号为定时器时钟，在时钟为低电平时采样 EQU0 和 EQU1～6，在时钟的下一个上升沿锁存入 D 触发器中。

输出模式由模式控制位 OMx0，OMx1 及 OMx2 决定，共有 8 种输出模式，分别对应 OMx2，OMx1 及 OMx0 的值。如 OMx2，OMx1，OMx0＝011 时为模式 3。除模式 0 外，其他的输出都在定时器时钟上升沿时发生变化。输出模式 2，3，6，7 不适合输出单元 0。所有的输出模式定义如下：

● 输出模式 0　输出模式。输出信号 OUTx 由每个捕获/比较模块的控制寄存器 TBCCTLx 中的 OUTx 位定义，并在写入该寄存器后立即更新。最终为 OUTx 直通。

● 输出模式 1　置位模式。输出信号在 TBR 等于 TBCLx 时置位，且保持置位到定时器

复位或选择另一种输出模式为止。

● 输出模式 2　PWM 翻转/复位模式。输出在 TBR 的值等于 TBCLx 时翻转，当 TBR 的值等于 TBCL0 时复位。

● 输出模式 3　PWM 置位/复位模式。输出在 TBR 的值等于 TBCLx 时置位，当 TBR 的值等于 TBCL0 时复位。

● 输出模式 4　翻转模式。输出电平在 TBR 的值等于 TBCLx 时翻转，输出周期是 TBR 的周期的 2 倍。

● 输出模式 5　复位模式。输出在 TBR 的值等于 TBCLx 时复位，并保持低电平直到选择另一种输出模式。

● 输出模式 6　PWM 翻转/置位模式。输出电平在 TBR 的值等于 TBCLx 时翻转，当 TBR 的值等于 TBCL0 时置位。

● 输出模式 7　PWM 复位/置位模式。输出电平在 TBR 的值等于 TBCLx 时复位，当 TBR 的值等于 TBCL0 时置位。

输出单元在输出控制位的控制下，有 8 种输出模式输出信号。这些模式都与 TBR 的值、TBCLx 的值及 TBCL0 的值有关。

● 在增计数模式下，当 TBR 增加到 TBCLx 或从 TBCL0 计数到 0 时，OUTx 信号按选择的输出模式发生变化。实例见图 3.65。

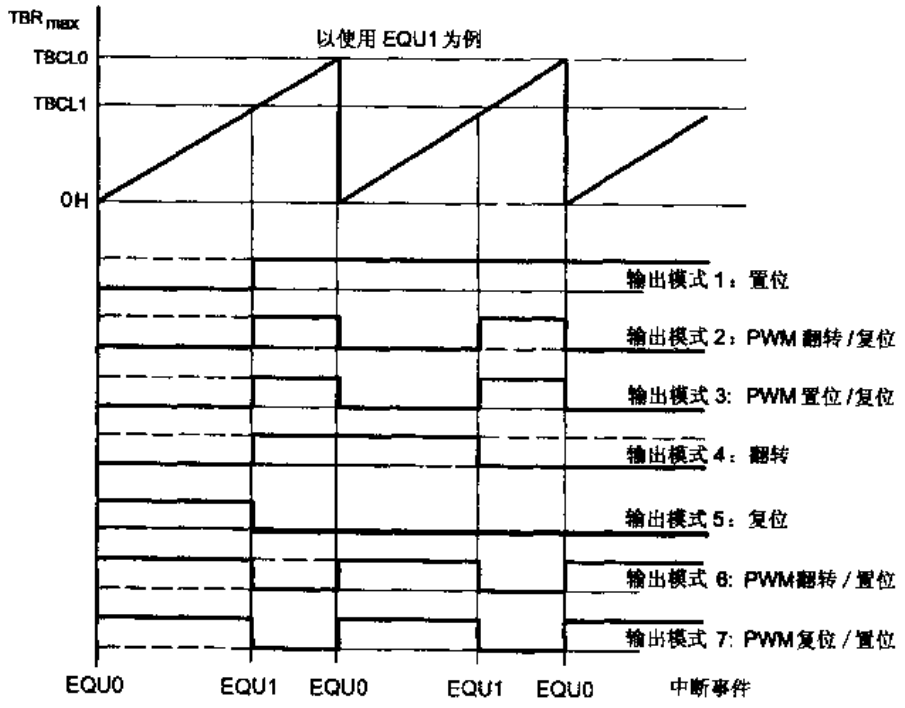


图 3.65　增计数模式时的输出实例

图中，定时器工作在增计数模式下，TBCL0 和 TBCL1 的值为两根横线，斜线为 TBR 的值。图的下面为各种模式下的输出波形。

第 1 个波形为模式 1 的输出：开始时为低，当 TBR＝TBCL1 时，输出高电平。

第 2 个波形为模式 2 的输出：当 TBR＝TBCL1 时输出发生翻转，当 TBR＞TBCL1 时输

出高电平，当 TBR<=TBCL1 时输出低电平。高低电平的时间由 TBCL1 和 TBCL0 两个寄存器的内容决定。TBR 的值在 0 与 TBCL0 的值之间增加，输出波形的高低也由 TBCL1 和 TBCL0 的内容决定；因此通过改变 TBCCR1 和 TBCCR0 的值（TBCCR1 和 TBCCR0 的值将传给 TBCL1 和 TBCL0）就可改变输出波形的占空比，并输出脉冲宽度调制 PWM 波。

第 3 个波形为模式 3 的输出：这种输出模式输出的波形与模式 2 的输出一样。输出在 TBR 的值等于 TBCL1 时为高电平，直到 TBR 的值增加到 TBCL0。

第 4 个波形为模式 4 的输出：当增计数到 TBCL1 时，输出电平发生改变。当定时器 TBR 增计数到 TBCL1 时，输出为整个波形的一部分，整个波形的周期为定时器定时周期的 2 倍。

第 5 个波形为模式 5 的输出：这个输出波形与模式 1 的输出正好相反。当增计数到 TBCL1 时输出复位，并一直保持。

第 6 个波形为模式 6 的输出：这个输出波形与模式 2 的输出正好相反。输出在计数值增到 TBCL1 时为高电平，而在计数值从 TBCL1 到 TBCL0 的一段时间内为低电平。

第 7 个波形为模式 7 的输出：与模式 6 的输出波形一样。

● 在连续计数模式下的输出波形如图 3.66 所示，可以看出其波形与增计数模式相同，只是计数器在增计数到 TBCL0 后还要继续增计数到 TBR_{max}，TBR_{max} 因设置的计数长度而异。这样就延长了计数器计数到 TBCL1 的数值后的时间。这是与增计数模式不同的地方。

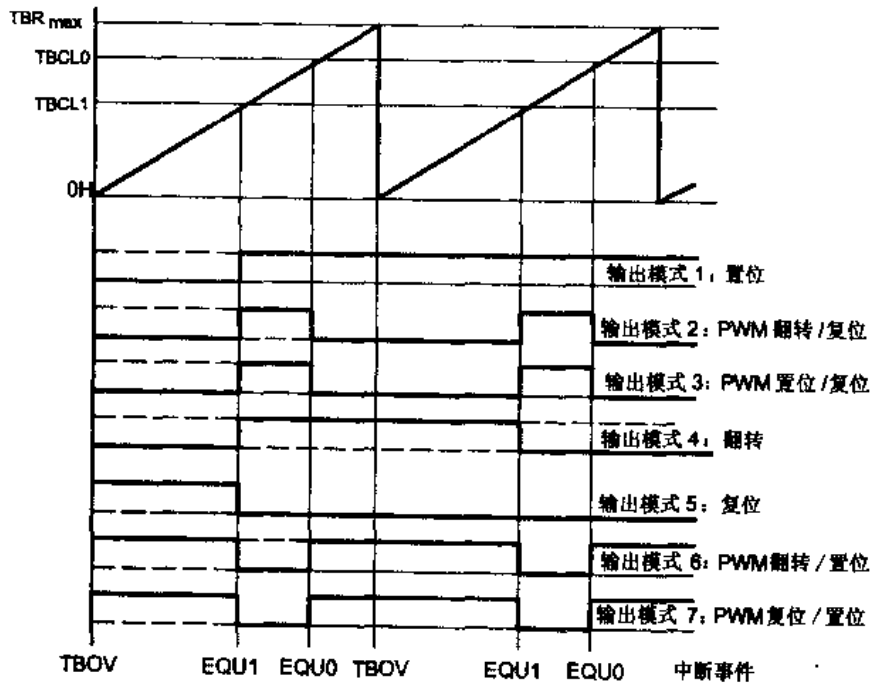


图 3.66 连续计数模式时的输出实例

● 在增/减计数模式下的输出实例如图 3.67 所示。可以看出，这时的各种输出波形与定时器增计数模式或连续计数模式不同。当定时器在任意计数方向上等于 TBCLx 时，OUTx 信号都按选择的输出模式发生改变。下面分别说明，这里以 TBCL3 为例。

第 1 个波形为模式 1 的输出：与定时器的增计数模式或连续计数模式的输出波形相同。

第 2 个波形为模式 2 的输出：当 TBR = TBCL3 时输出发生翻转，从 0 增计数到 TBCL3

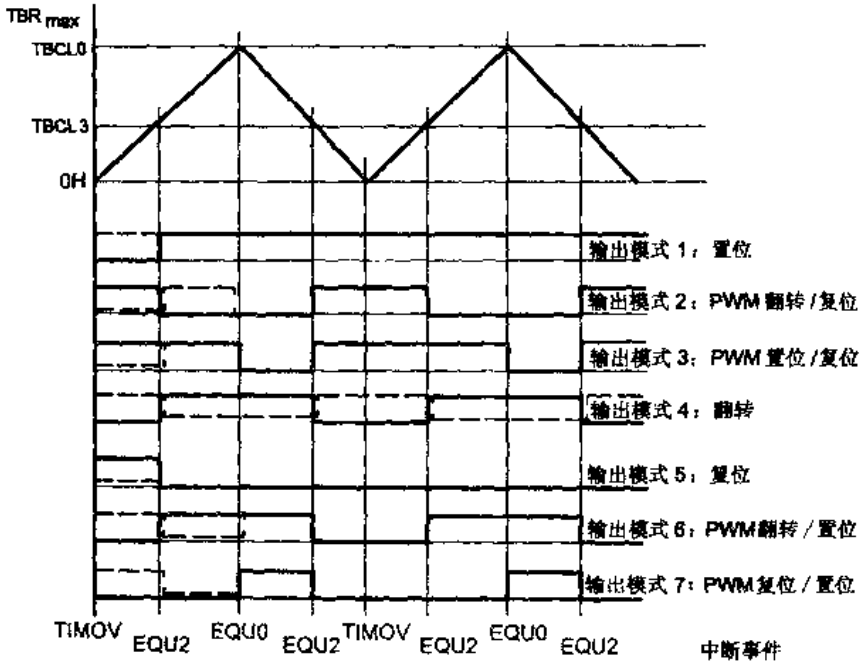


图 3.67　增/减计数模式时的输出实例

时及从 TBCL0 减计数到 TBCL3 时都翻转。TBCL0＞TBR＞TBCL3 时输出高电平，当 0＜TBR＜TBCL3 时输出低电平。高低电平的时间由 TBCL3 和 TBCL0 两个寄存器的内容决定。TBR 的值在 0 与 TBCCR0 的值之间增加或在 TBCCR0 与 0 之间减小，输出波形的高低由 TBCCR3 和 TBCCR0 的内容决定(TBCCR3 和 TBCCR0 的值将传给相应的 TBCL3 和 TBCL0)，通过改变 TBCCR3 和 TBCCR0 的值就可改变输出波形的占空比，输出脉冲宽度调制 PWM 波。

第 3 个波形为模式 3 的输出：输出在 TBR 的值等于 TBCL3 时为高电平，直到定时器增加到 TBCL0 时变为低电平。

第 4 个波形为模式 4 的输出：当增计数到 TBCL3 时(无论增计数还是减计数)，输出电平发生翻转(与以前电平相反)。

第 5 个波形为模式 5 的输出：这个输出波形与模式 1 的输出正好相反。当增计数到 TBCL3 时输出复位，并一直保持。

第 6 个波形为模式 6 的输出：这个输出波形与模式 2 的输出正好相反。输出在计数值增到 TBCL0 时为高电平，而在计数值从 TBCL3 增到 TBCL0 及从 TBCL0 减到 TBCL3 的一段时间内都为高电平。

第 7 个波形为模式 7 的输出：当 TBR 的值等于 TBCL3 的值时输出为 0；当 TBR 的值等于 TBCL0 的值时输出为 1。

5. 应用举例

定时器的编程应用一般要注意以下一些问题：

- 定时器的计数信号源，由 TPSSEL1 和 TPSSEL0 选择 TACLK，ACLK，SMCLK 或 INCLK；
- 分频系数，由 ID1 和 ID0 选择 1，2，4 或 8 分频；
- 计算捕获/比较寄存器的值，由应用要求决定；

● 定时器、捕获/比较寄存器及输出模块等的工作模式，由应用要求决定。

[例 1] 利用定时器 B 输出周期为 2 ms 的方波，并由 P1.0 输出。

使用在 P1.0 端口每隔 1 ms 将输出求反的方式实现题目的要求。也就是说，定时器要产生的定时时间为 1 ms。在默认条件下 DCO 产生的频率为 800 kHz。使用 800 kHz 的 SMCLK 作为定时器的时钟源，则定时 1 ms 的计数值为

$$1\times10^{-3}/(1/(800\times1000))=800$$

```
#include    "msp430x14x.h"
            ORG     01100h
RESET       mov.w   #0A00h,SP           ;初始化堆栈指针
SetupTB     mov.w   #TBSSEL1+TBCLR,&TBCTL   ;选择 SMCLK 时钟
SetupC0     mov.w   #CCIE,&TBCCTL0      ; TBCCR0 中断开放
            mov.w   #800,&TBCCR0        ;
SetupP1     bis.b   #001h,&P1DIR        ; P1.0 定义为输出
            bis.w   #MC1,&TBCTL         ; Timer_B 工作在连续模式
            eint                        ;开总中断
Mainloop    bis.w   #CPUOFF,SR          ; CPU 关掉
TB0_ISR     xor.b   #001h,&P1OUT        ; P1.0 输出反向
            add.w   #800,&TBCCR0        ; Add Offset to TBCCR0
            reti                        ;
            ORG     0FFFEh              ; MSP430 RESET 向量
            DW      RESET               ;
            ORG     0FFFAh              ; Timer_B0 向量
            DW      TB0_ISR             ;
```

[例 2] 利用定时器 B 输出周期为 2 ms 的占空比为 25%的矩形波。

基本与定时器 A 相同，这里不再重复。

3.4 硬件乘法器

MSP430 系列中，有些器件有硬件乘法器。它们是 MSP430X3XX，MSP430F14X 及 MSP430F44X。在这些器件中，硬件乘法器作为一个 16 位的片内外设，它的运行独立于 CPU，并通过内部总线与 CPU 相连，如图 3.68 所示。

硬件乘法器能够实现 16×16 位、8×16 位、16×8 位或 8×8 位运算；支持无符号乘法(MPY)、有符号乘法(MPYS)、无符号乘加(MAC)和有符号乘加(MACS)。硬件乘法器的结构如图 3.69 所示。

图 3.69 的上部是两个操作数寄存器(OP1 和 OP2)，第一个操作数可来源于 4 个寄存器：MPY，MPYS，MAC 及 MACS。使用时只能用其中之一作为第一个操作数。这 4 个寄存器不仅为第一操作数的暂存，也同时确定了乘法操作的类型。当第二个操作数写入后，相应的乘法操作立即执行，一般需 4 个周期数。为保险起见，一般在读结果之前，须额外写一些指令，如 NOP 等，以确信执行完毕，得到正确的结果。必须注意，一定要先写第一个操作数，再写第二个操作数，否则乘法操作不会被执行。

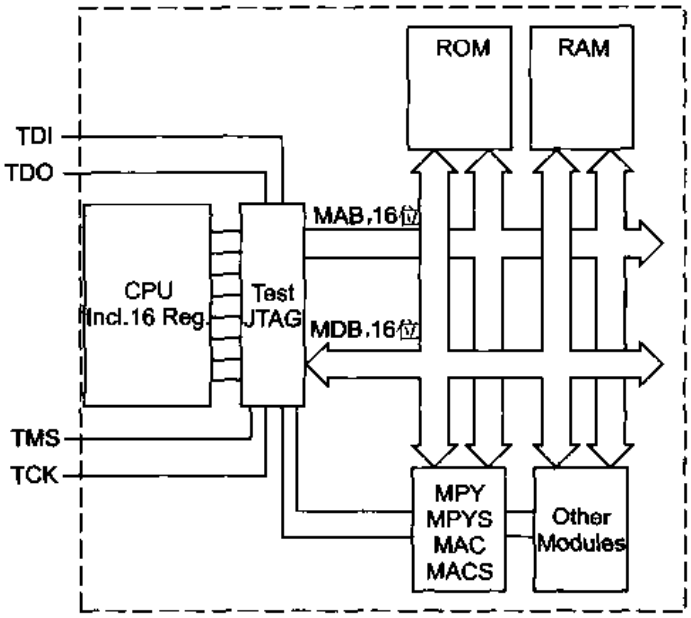


图 3.68　硬件乘法器通过内部总线与 CPU 相连

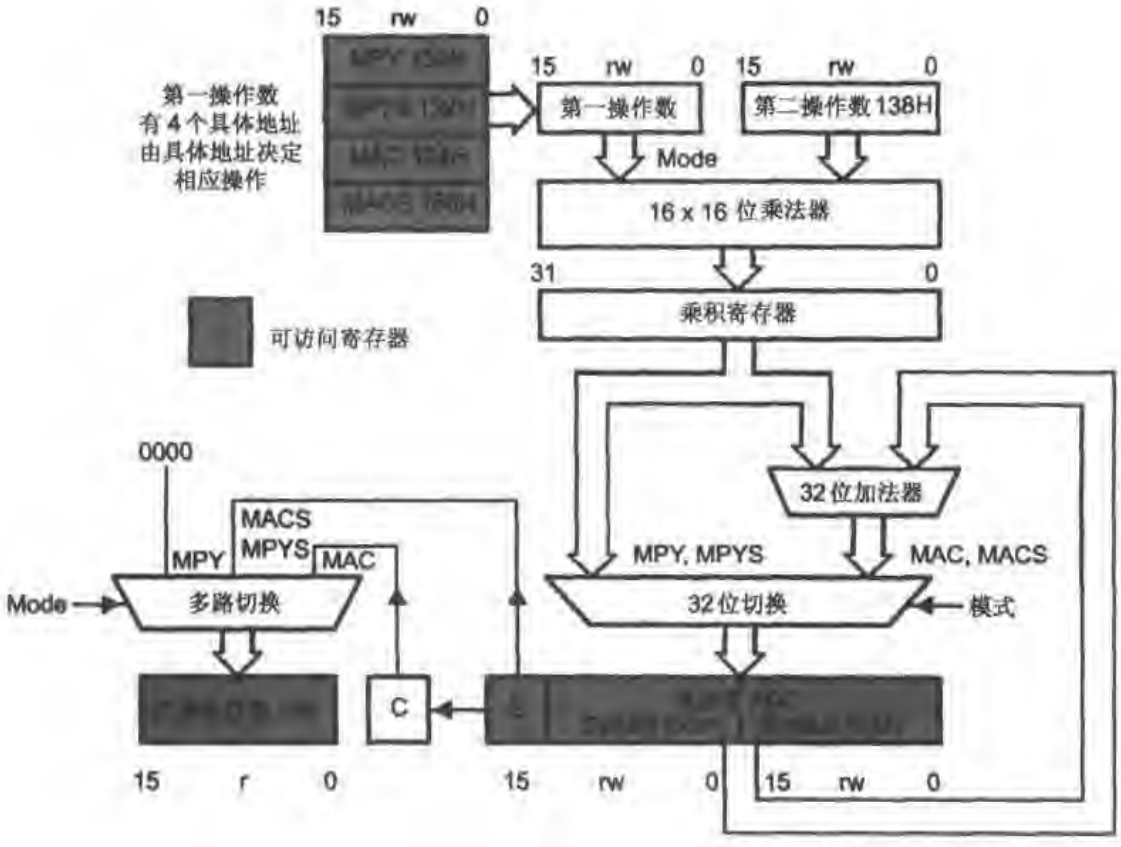


图 3.69　16×16 位硬件乘法器的结构图

如果相乘的两个数据已经有了，执行乘法之后，结果就保存在紧接着的乘积寄存器中。这是一个 32 位寄存器，在一般情况下，有两个乘数和一个乘积，这个乘积只是暂存；乘加运算将用到累加器 ACC，在不同的模式下，它经过相应的转换与运算，最终保存到图 3.69 最下面的 3 个16 位寄存器中，分别是：结果高字寄存器(RESHI)、结果低字寄存器(RESLO)及结果扩展寄存器(SUMEXT)。寄存器 RESHI 和 RESLO 的内容为两 16 位数相乘的 32 位乘积结论，而寄存器 SUMEXT 的内容由执行的乘法模式及乘积的结果决定。表 3.15 说明了它们的关系。

表 3.15 SUMEXT 的内容

寄存器	MPY	MPYS		MAC		MACS	
OP1	×	+ −	+ +	两数积+ACC ≤ 0FFFFFFFFH	两数积+ACC > 0FFFFFFFFH	两数积 +ACC> 07FFFFFFH	两数积 +ACC≤ 07FFFFFFH
OP2	×	+ −	− −				
SUMEXT	0	0000H	0FFFFH	0000H	0001H	0FFFFH	0000H

(1) 硬件乘法器寄存器

从图 3.70 可以看出，硬件乘法器有如下一些寄存器(硬件乘法器调试界面)：

图 3.70 16×16 位硬件乘法器的寄存器

- MPY 操作数 1，同时指示操作为无符号数相乘。
- MPYS 操作数 1，同时指示操作为有符号数相乘。
- MAC 操作数 1，同时指示操作为无符号数累加。
- MACS 操作数 1，同时指示操作为有符号数累加。
- OP_2 操作数 2 。
- RESLO 结果低字寄存器。
- RESHI 结果高字寄存器。
- SUMEXT 结果扩展寄存器。

以上所有寄存器都是 16 位的字寄存器，在使用时，用户可字操作或字节操作，这样就形成了不同位数乘法的 4 种运算。

(2) 硬件乘法器使用时的注意事项

使用硬件乘法器完成一个乘法运算只需 3 个步骤：

① 写入第一个操作数 OP1 到 MPY，MPYS，MAC 及 MACS 4 个寄存器之一，同时也确定了乘法运算的类型；

② 写第二个操作数 OP2，写入完毕，乘法运算立即进行；

③ 在 RESHI，RESLO 及 SUMEXT 寄存器中取出乘法结果。

在这个过程中须注意以下事项。

首先，第二个操作数写入完毕，乘法运算就开始。它的执行靠硬件，虽然相当快，但也要花时间，要确保其运算完成，才能取出结果。一般在取出结果之前插入 1～2 条指令，以保证运算时间的需要。

其次，在一个器件中只有一个硬件乘法器，如果遇到多处使用的情况，必须在每一次使用完成后再进行下一次使用。举一个简单例子：在主程序中，一个乘法运算在刚写入第一个操作数 OP1 时就进入了中断。而中断服务程序中也有一个使用硬件乘法器的地方，它也要写入第一、二操作数，执行乘法运算，等到中断返回，主程序应该写入第二个操作数了，写入完毕，执行乘法运算。而这时的乘法运算是中断服务程序中的 OP1 与主程序中的 OP2 两数相乘，所以结果肯定就错了！

再次，在程序设计时，所使用的器件中一定要有硬件乘法器，才能使用。如果没有，则寻址将找不到它的寄存器，也就无法使用了。无论是汇编语言还是 C 语言，都将写上器件描述的包含语句，如使用的是 14X 器件：#include <msp430x14x.h>。

最后要注意，结果扩展寄存器(SUMEXT)的内容，与运算类型及运算结果都有关系，如表 3.15 所列。

(3) 硬件乘法器使用举例

硬件乘法器的使用相当简单，可在图 3.70 界面下直接输入数据，回车之后，在结果寄存器中马上就出现乘积结果。在程序里也一样，先写第一操作数，再写第二操作数，然后取出结果。

[例 1] 16×16 位无符号乘法，两乘数在 R6 和 R7 中，积放在 R8 和 R9 中，程序如下：

```
#include  <msp430x14x.h  msp430x14x.h / msp430x3xx.h >;这 3 类器件都有硬件乘法器
        MOV     R6,&MPY
        MOV     R7,&OP_2
        NOP
        NOP
        MOV     &RESLO,R8
        MOV     &RESHI,R9
        ……
```

[例 2] 16×16 位有符号乘法，两乘数在 R6 和 R7 中，积放在 R8 和 R9 中，程序如下：

```
#include  <msp430x14x.h >
        MOV     R6, &MPYS      ;这里与例 1 不同
        MOV     R7,&OP_2
        NOP
        NOP
        MOV     &RESLO,R8
        MOV     &RESHI,R9
        ……
```

在这个程序中，如果 R6 和 R7 为两个同号数相乘，则结果扩展寄存器(SUMEXT)的内容为 0000H；如果为异号数相乘，则结果扩展寄存器(SUMEXT)内容为 0FFFFH。

[例 3] 16×16 位无符号乘加，两乘数在 R6 和 R7 中，乘加次数放在 R5 中，积放在 R8 和 R9 中，程序如下：

```
#include  <msp430x14x.h>
          MOV     #0,&RESLO
          MOV     #0,&RESHI
LOOP      MOV     R6, &MAC
          MOV     R7,&OP_2
          NOP
          DEC     R5
          JNZ     LOOP
          MOV     &RESLO,R8
          MOV     &RESHI,R9
          ……
```

这个例子可用在需要乘加的场合，运算之前应该清除结果寄存器的值，再进行乘法运算，然后计算的值与 ACC 相加，直到所有的乘加运算完毕。

3.5 比较器 A

比较器 A（Comparator_A）在大部分器件中都有，如 MSP430F11X1，MS430F12XX，MS430F13X/14X，MS430X3XX 和 MS430F4XX。在该模块中参与比较的是两个模拟量，结合其他模块可实现模数转换的功能。其功能简述如下：

- 比较器 A 模块可软件打开/关闭，不用时关闭以节省能耗；
- 无回差输入；
- 能提供内部模拟参考电平，同时可对外提供；
- 比较器输入端可任意切换；
- 比较器的输出有 RC 滤波电路，而且软件选择；
- 能中断，有中断向量。

模拟比较器的结构原理图如图 3.71 所示。

从图中可以看出，它有两个模拟量输入端 CA0 和 CA1、一个模拟比较器、参考电压发生器和输出滤波器，还有一些控制单元。下面简单介绍其结构。

- 模拟比较器有正、负两个输入端，这两个输入端可通过控制信号选择 6 种信号（CA0、CA1、0.5V_{CC}、0.25V_{CC}、三极管阀值电压和外部参考源），并多种组合进行比较。这些信号哪些放在正输入端，哪些放在负输入端，与什么信号比较，都将直接影响到比较器的输出结果。这些在控制寄存器中将会详细介绍。

- 参考电压发生器，从图中可看出，它的实质是一个电阻分压器，可产生 4 种参考电压：0.5V_{CC}、0.25V_{CC}、三极管阀值电压和外部参考源。这对电源的稳定性有较高的要求。另外还有一个控制位（CAON）控制参考电压发生器的电源，该位同时控制着整个比较器的能量供给，不用时可通过它关闭比较器。

- 比较器（这里指的是图的上面中间部分，不是整个模块）的输出经过一个与门整形之后，由控制位 CAEX 选择正向或反向输出，然后送到由 RC 组成的低通滤波器。最终的信号可有 3 个去处：送给内部的其他模块，作为其他模块的一个输入信号；由外部引脚引出；设置中断标志位，以引起中断请求。

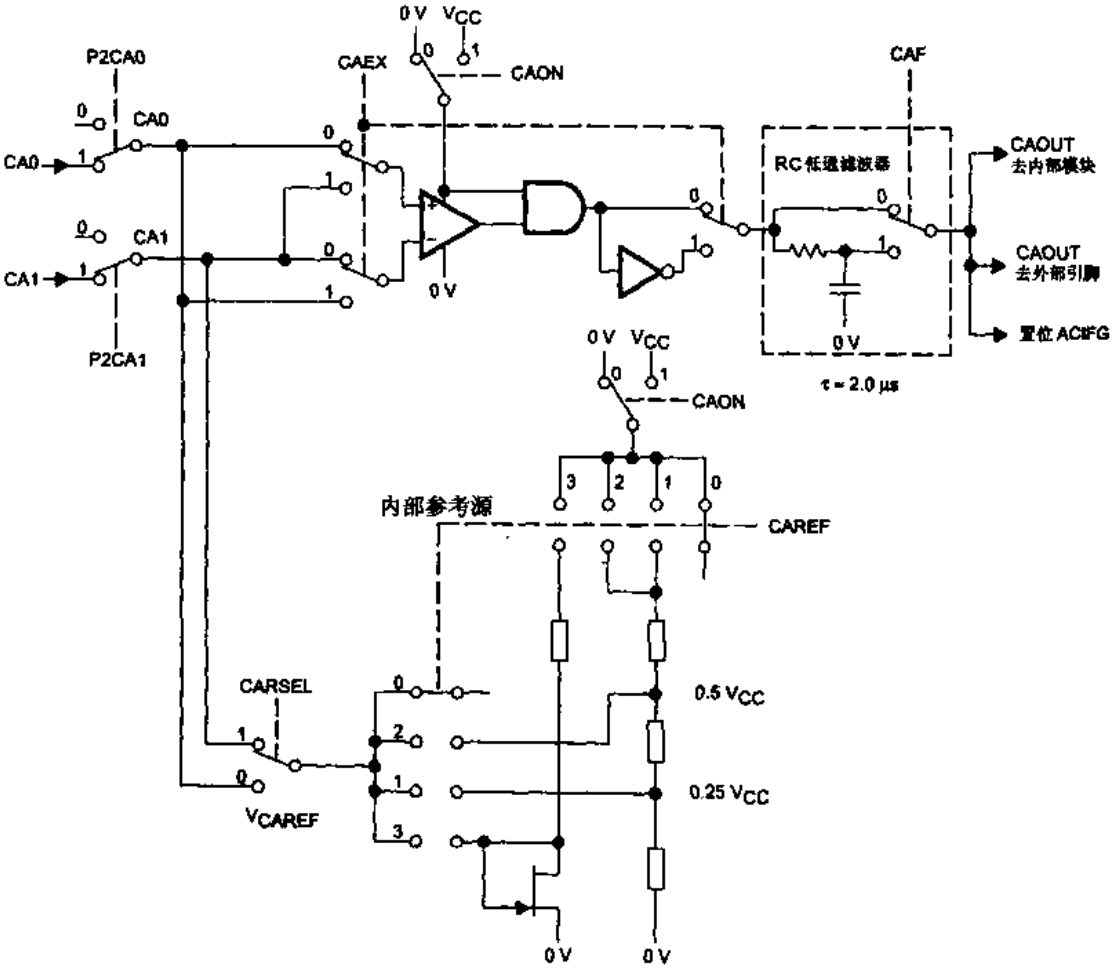


图 3.71　模拟比较器的结构原理

1. 比较器 A 的寄存器

模拟比较器的寄存器在调试软件中如图 3.72 所示。这些寄存器被安排在字节空间，因此必须使用字节方式指令予以访问。

图 3.72　模拟比较器的寄存器

(1) CACTL1　控制寄存器 1

该寄存器包含着模拟比较器的大部分控制位，是 8 位寄存器，各位的定义如下：

7	6	5	4	3	2	1	0
CAEX	CARSEL	CAREF1	CAREF0	CAON	CAIES	CAIE	CAIFG

CAOUT　比较器 A 的输出。

(3) CAPD　端口禁止寄存器

单片机的引脚有限，因此通常一根引脚由多模块共用。这里也不例外，比较器模块的输入输出也与 I/O 口共用引脚。而 MSP430 的 I/O 端口有输入和输出缓冲器，当 I/O 口与比较器模块复用时，可关掉输入缓冲，这一功能由 CAPD 寄存器控制。该寄存器为字节类型寄存器，每位对应某端口的相应位的控制。

0　对应端口的相应位允许输入缓冲；

1　对应端口的相应位关闭输入缓冲。

MSP430F1121 的资料表明：CAPD 控制 P2 口的相应位，如果 CAPD.3＝1，则将读不到 P2.3 引脚上的输入信号(数字量)。

2. 比较器 A 的中断

比较器 A 具有中断能力，有一个中断电路和中断向量。图 3.73 为其中断电路图。整个电路的输入信号源自比较器 A 的比较结果，比较器输出的上升沿或下降沿都可使中断标志 CAIFG 置位。由 CAIES 选择上升、下降边沿，中断允许位 CAIE 和 GIE 控制能否产生中断请求。当发生了中断服务，硬件会自动清除中断标志位。

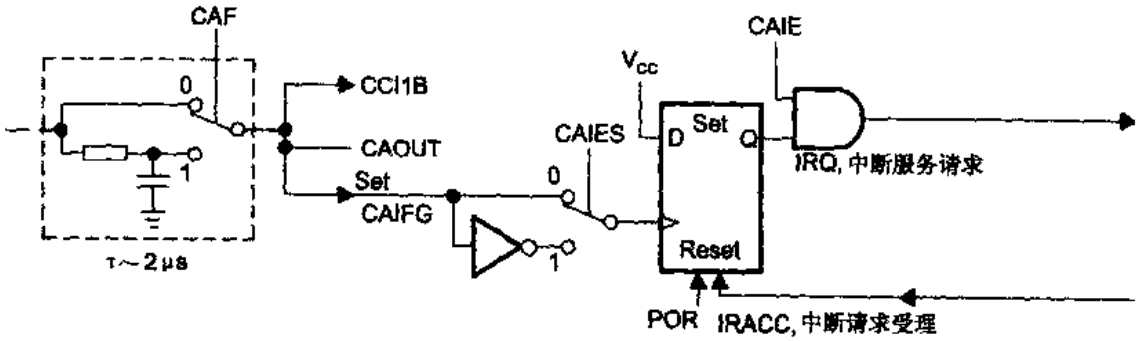


图 3.73　比较器 A 中断电路图

3. 比较器 A 的应用

比较器常用在模拟量的比较和测量等场合。下面的应用举例中通过前两个例程就可以较为清楚地认识模拟比较器。

[例 1]　输入模拟比较器的参考电压源。

模拟比较器有片内参考源，分别是电源的 1/2 和 1/4。同时可以通过引脚将它们输出，对外提供。下面的程序将实现这个目的，用的器件是 MSP430F1121，模拟参考源由 P2.3 输出。其他型号请查阅相关手册，选择模拟输入端作为输出。程序如下(C 语言)：

```
#include  "msp430x11x1.h"
void delay(void);                          //定义延时子程序
void main (void)
{
  WDTCTL = WDTPW + WDTHOLD;                //停止看门狗
  CACTL2 = P2CA0;                          //定义 P2.3 作为模拟比较器的正输入端 +comp
  while (1)                                //主循环
  {
```

```
    CACTL1 = 0x00;                    //设置内部参考电压不发生
    delay();
    CACTL1 = CAREF0 + CAON;           //设置产生在 P2.3 引脚输出 0.25Vcc
    delay();
    CACTL1 = CAREF1 + CAON;           //设置产生在 P2.3 引脚输出 0.5Vcc
    delay();
  }
}
void delay(void)                      //延时程序:等待用电压表测量 P2.3 上的电压
{
  unsigned long i;
    for (i = 0x7FFFF; i > 0; i--);
}
```

[例 2] 模拟比较器的比较实例。

这个例程将片内 0.25V_{CC}接在比较器的负端,在比较器的正端 P2.3 外接一个模拟电压,改变此外接模拟电压。当外接电压大于 0.25V_{CC}时 P1.3 输出低电平;当外接电压小于 0.25V_{CC}时 P1.3 输出高电平。程序如下(C 语言):

```
#include  "msp430x11x1.h"
void main (void)
{
  WDTCTL = WDTPW + WDTHOLD;           //停止看门狗
  P1DIR |= BIT3;                      //P1.3 输出
  CACTL1 = CARSEL + CAREF0 + CAON;    //定义模拟比较器负输入端为内部参考 0.25Vcc
  CACTL2 = P2CA0;                     //定义 P2.3 作为模拟比较器的正输入端+comp
  while (1)                           //测试 comparator_A 的输出
  {
    if ((CAOUT & CACTL2) == CAOUT) P1OUT |= BIT3;   //如果 CAOUT=1,则 P1.3=1
    else P1OUT &= ~BIT3;              //否则 P1.3=0
  }
}
```

[例 3] 电阻型传感器阻值的测量。

电阻型传感器如温度传感器等,要测量温度,使用热敏电阻感知温度,然后反映在热敏电阻阻值的变化上,使温度测量转化为电阻的测量。这里使用 MSP430F1101,它是 MSP430 系列中很廉价的一种型号,片内含模拟比较器和定时器等模块。电路连接如图 3.74 所示,被测电阻 R_{SENS}放在 P1 的某根口线和模拟比较器的一个输入端(+);参考电阻 R_{REF}放在 P1 口的另一口线和模拟比较器的输入端;同时在比较器的输入端接一电容到地。这时可通过控制 P1 口的P1.x和 P1.y 来控制通过 R_{SENS}或 R_{REF}对电容充放电。而模拟比较器的输入端(+)电压就会因电容充放电而变化。如果先被测电阻充放电,再参考电阻充放电,则由于两电阻阻值不一样,而引起充放电的时间也不一样,如图 3.75 所示。

选择模拟比较器的另一个模拟输入端(-)连接到内部参考源 0.25V_{CC}。先通过电阻对电容充电一段时间,然后再放电。那么每当模拟比较器的正端电压下降到 0.25V_{CC}时会输出比

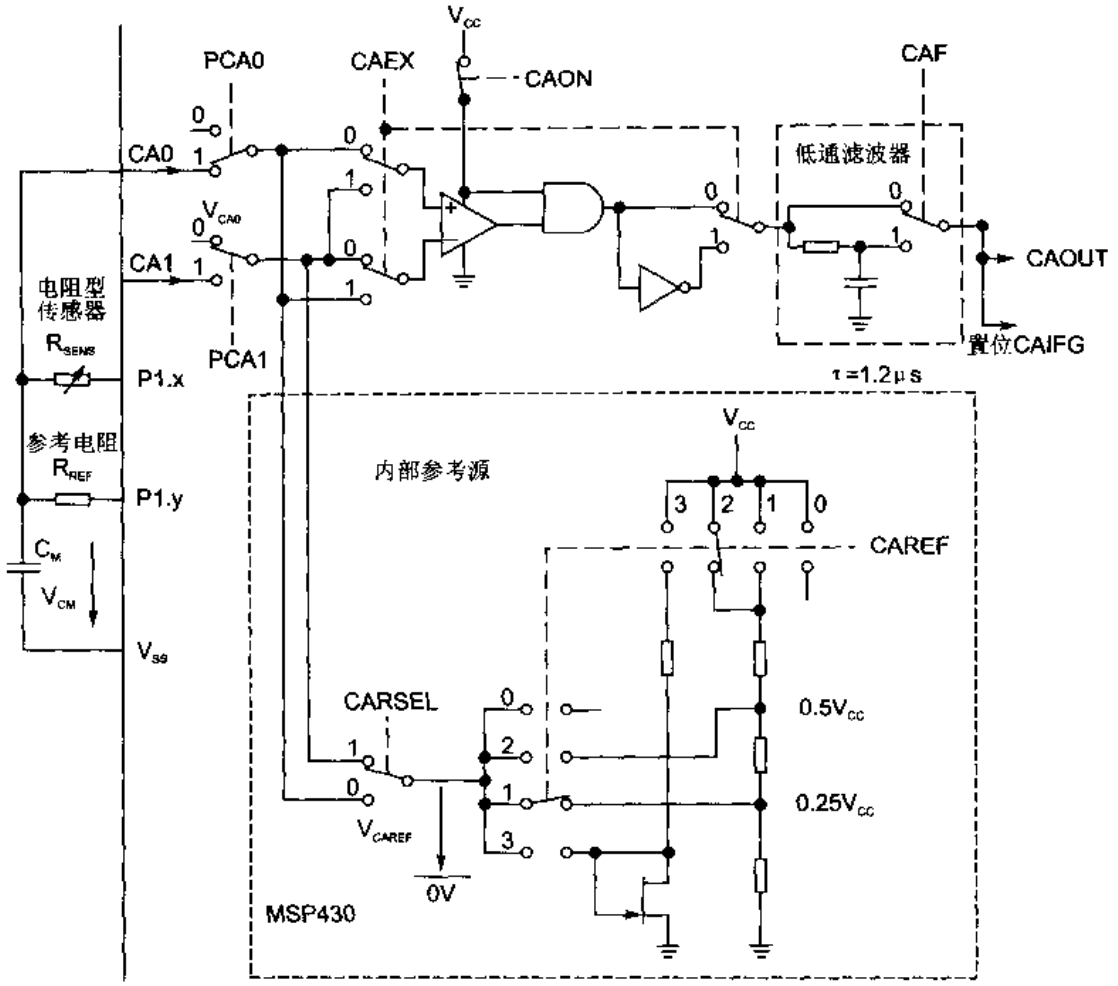


图 3.74　电阻型传感器阻值的测量原理电路图

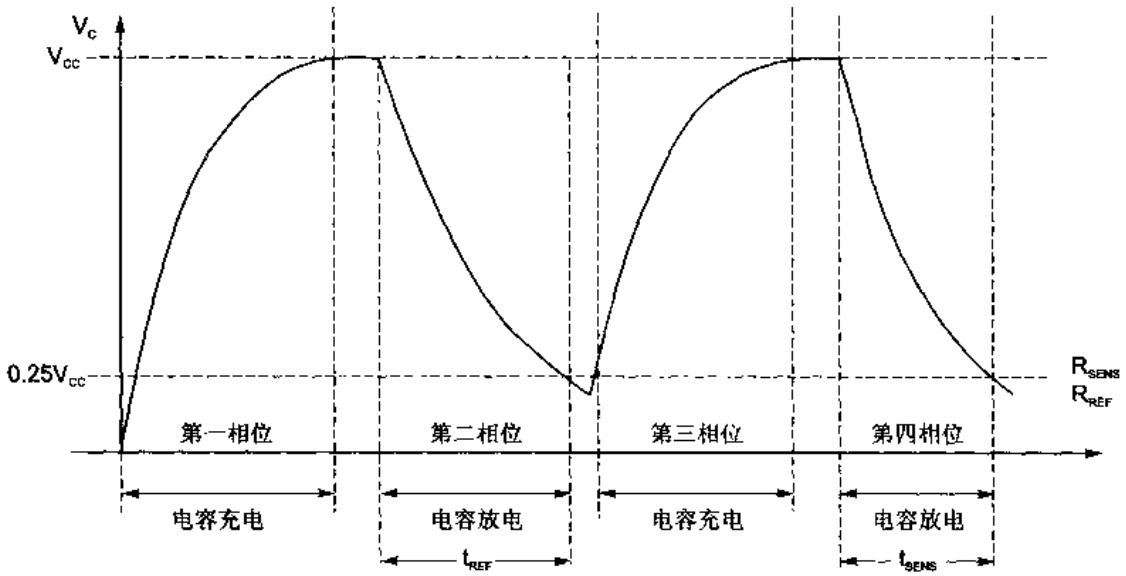


图 3.75　模拟比较器正端电压

较器信号 CAOUT。在放电开始时记录下定时器的值 TAR，当模拟比较器输出信号 CAOUT 时，让定时器发生捕获，再次读入定时器值，则两次读出的定时器值之差就是接某电阻时的放电时间。如果被测电阻与参考电阻的放电时间(t_{SENS}和 t_{REF})都知道，因放电曲线近似直线，所以可近似为

$$R_{SENS}/R_{REF}=t_{SENS}/t_{REF}$$

因此

$$R_{SENS}=(t_{SENS}/t_{REF})\times R_{REF}$$

由此得出程序如下：

```
Ref         equ     001H                        ; P2.0 连接参考电阻
Sensor      equ     002H                        ; P2.1 连接传感器
Mainloop    mov     #Sensor,R14
            call    #Measure                    ;测量传感器的放电时间
            mov     R14,R11
            mov     #Ref,R14
            call    #Measure                    ;测量参考电阻的放电时间
            push    R14                         ;
Calculate   mov     #10000,R12                  ;根据两个放电时间以及已知的参考电阻值
                                                ;计算被测传感器的电阻值
            call    #MPYU                       ;已知参考电阻值为 10 kΩ，乘以传感器的放
                                                ;电时间，再除以参考电阻的放电时间
            pop     R11
            mov     R14,R12                     ;
            mov     R15,R13                     ;
            call    #DIVIDE                     ; R14 = Sensor * 10 000/Ref
            mov     R14,R15
Measure;    输入参数：R14 = Sensor or Ref P2.x 位
;           输出参数：R14 = Sensor or Ref 放电时间
Charge ;    Capacitor is charged via Ref        ;这段程序为充电
            bis.b   #CAON,&CACTL1
            bis.b   #Ref,&P2OUT
            bis.b   #Ref,&P2DIR
            mov     &TAR,&CCR1
            add     #5000,&CCR1
            mov     #CCIE,&CCTL1
            bis     #LPM0,SR
            bic.b   #Ref,&P2DIR
            bic.b   #Ref,&P2OUT
Discharge;  Measure Discharge Time
            mov     #CM1+CCIS0+CAP+CCIE,&CCTL1  ;放电
            push    &TAR                        ;测量放电时间
            bis.b   R14,&P2DIR
            bis     #LPM0,SR
            mov     &CCR1,R14
```

```
            sub     @SP+,R14
            bic.b   #Sensor+Ref,&P2DIR
            clr     &CCTL1
            bic.b   #CAON,&CACTL1
            ret                                     ;
TAX_ISR                                             ;定时器 A 的中断服务程序
            add     &TAIV,PC
            reti                                    ; CCR0
            jmp     CCR1_ISR                        ; CCR1
            reti                                    ; CCR2
            reti                                    ; CCR3
            reti                                    ; CCR4
TA_over     reti                                    ; Timer_A 溢出
                                                    ;
CCR1_ISR    bic     #LPM0,0(SP)                     ;退出 LPM0 模式,继续工作
            reti                                    ;
WDT_ISR;
            bic     #LPM3,0(SP)                     ;退出 LPM3
            reti
```

模拟比较器不但可结合定时器作电阻型传感器的电阻值测量，还可测量电压和电流等电参数，是一种较为经济的测量手段。不过在这种测量方式下要注意的问题是：测量精度与很多因素有关，放电时间的长短、放电曲线的线性度等都将影响测量结果的可信度。

3.6　FLASH 存储器模块

FLASH 存储器模块是 FLASH 型器件中都有的一个模块，但不同型号器件中的 FLASH 的容量不同，所在的地址空间也不一样；然而它们有一点是相同的：都是由 n 段主存储器与 2 段信息存储器组成。信息存储器为每段 128 字节，取名为信息存储器 A 和 B，主存储器每段为 512 字节。所有型号器件的信息存储器的地址完全相同，从 1000H～10FFH。主存储器的地址范围不一样，但起始地址一样，都是第 0 段源于地址 0FFFFH，不同型号的器件最后一段的结束地址不同。如图 3.76 所示。

图 3.76 只示出了部分容量的 FLASH 段与地址的对应关系，16 KB 以下还有 8 KB，4 KB，2 KB 和 1 KB 等 FLASH 容量，其 FLASH 段与地址的对应关系也是一样的。这里给出一个满足所有 FLASH 型号器件的每一段的起始和结束地址。

信息存储器 A 段的起始地址在 1080H，结束地址在 10FFH。

信息存储器 B 段的起始地址在 1000H，结束地址在 107FH。

主存储器第n(n∈ [0，119])段的起始地址为

$$FE00H-n\times 512$$

主存储器第n(n∈ [0，119])段的结束地址为

$$FFFFH-n\times 512$$

FLASH 型号器件中的 FLASH 存储器主要用作程序代码、数据表格以及用户信息等的

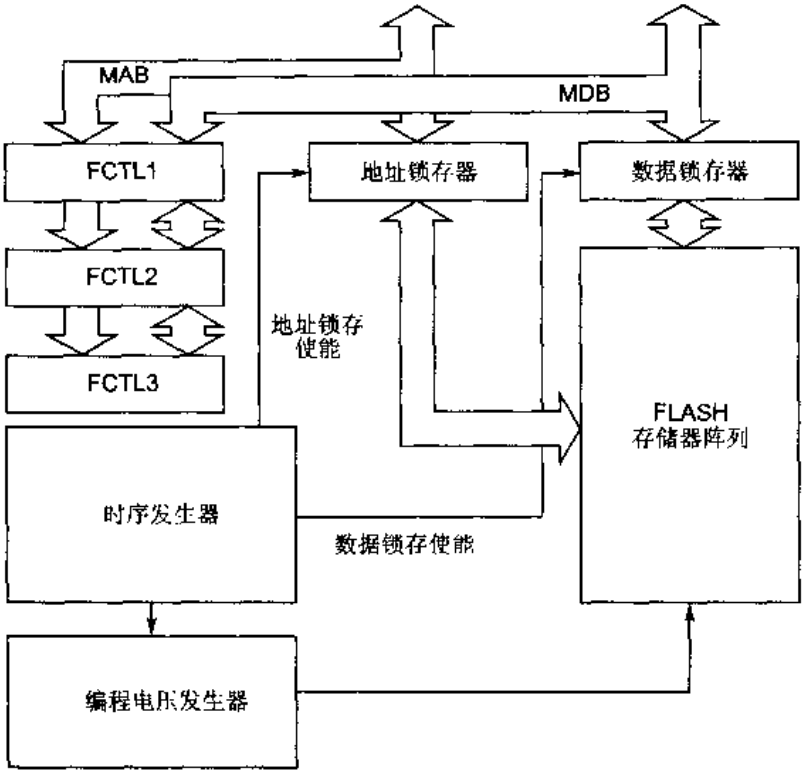


图 3.77　FLASH 存储器的结构框图

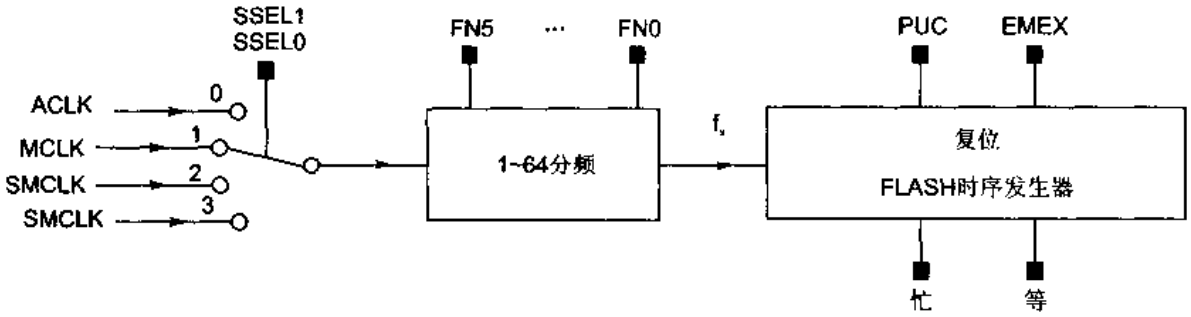


图 3.78　时序发生器的电路框图

时序发生器的原始时钟源自系统的 3 个时钟：ACLK，MCLK 和 SMCLK。它们由两个控制位 SSEL1 和 SSEL0 选择。这时所选的时钟还没有进入时序发生器，这个信号再经过分频器得到适当快慢的时钟后，得到 f_x 信号，这是送入时序发生器的真正原始时钟。时序发生器产生的所有需要的时钟都源于 f_x 信号。它产生的信号有：FLASH 存储器阵列地址锁存器的锁存信号、数据锁存器的锁存信号及编程电压发生器需要的信号等。这些时序信号对用户不可见。而用户能用的信号有：BUSY，WAIT 及 EMEX 等。BUSY＝1 时存储器正在被操作，这时不能对其访问。WAIT＝1 表明对 FLASH 的操作已经正确完成。如果在对 FLASH 操作过程中出错，可用 EMEX 位紧急退出。

编程电压发生器由控制位控制产生需要的 FLASH 编程电压。该电压在编程时供给 FLASH 存储器阵列。

FN5～FN0　分频系数选择位。分频系数为 FN5～FN0 值加 1。

0　直通；

1　2 分频；

2　3 分频；

……

63　64 分频。

(3) FCTL3　控制寄存器 3

FCTL3 寄存器主要是一些标志位。高字节同样是安全键值。各位的定义如下：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
安全键值：096H(读)，0A5H(写)										EMEX	Lock	WAIT	ACCVIFG	KEYV	BUSY

EMEX　紧急退出位。在对 FLASH 的操作失控时使用该位作紧急处理。

0　无作用；

1　立即停止对 FLASH 的操作。

Lock　锁定位，给已经编程好的 FLASH 存储器加锁。该位可由用户程序写入，也可由硬件自动设置。可在编程、段擦除、主存擦除期间置位，置位后当前操作能正常结束。在段编程模式中，如果 BLKWRT＝1，同时 WAIT＝1 时将 Lock 位置位，则 BLKWRT 和 WAIT 都将复位，段编程模式结束。如果在段编程模式中发生非法访问，则 ACCVIFG 和 Lock 位将置位。

0　不加锁，不加锁的 FLASH 存储器可读、可擦除、可写入；

1　加锁，加锁的 FLASH 存储器可读，但不可擦除和写入。

WAIT　等待指示信号位，该位只读。在段编程模式中指示 FLASH 存储器可以接受编程数据。如果 BLKWRT 位复位或 Lock 位置位，则 WAIT 自动复位，段编程结束使得 WAIT 位置位。在 BLKWRT＝1 时，每一次成功的写入之后，BUSY 位被复位，以指示其他的字或字节单元可以被编程操作。

0　段编程操作已经开始，编程操作进行中；

1　段编程操作有效，当前数据已经正确地写入了 FLASH 存储器，后续编程数据被列入计划。

ACCVIFG　非法访问中断标志。当对 FLASH 阵列进行编程或擦除操作时不能访问 FLASH，否则将使得该位置位。如果非法访问中断允许同时总的中断也允许，则将执行 NMI 中断程序。

0　没有对 FLASH 存储器的非法访问；

1　有对 FLASH 存储器的非法访问。

KEYV　安全键值(口令码)出错标志位。

0　对 3 个控制寄存器的访问，写入时高字节是 0A5H。

1　对 3 个控制寄存器的访问，写入时高字节不是 0A5H，同时引发 PUC 信号。KEYV 不会自动复位，须用软件复位。

BUSY　忙标志位。该位表示 FLASH 模块现在的状态，是否正在对其操作，现在忙与否。该位只读，为用户提供一个判断标志位。每次编程或擦除之前都应该

检查 BUSY 位。当编程或擦除启动以后，时序发生器将自动设置该位为 1，操作完成后，BUSY 位自动复位。

0 FLASH 存储器不忙；

1 FLASH 存储器忙，在段编程的 WAIT 状态时也处于忙状态。

3. 对 FLASH 存储器的操作

由于 FLASH 存储器由很多相对独立的段组成，因此可在一个段中运行程序，而对另一个段进行擦除或写入数据等操作。如果程序的运行和擦除或编程的段为同一段，则设置标志位 BUSY=1，而使得 CPU 挂起，直至编程周期结束，标志位 BUSY=0 时为止。这时才能继续 CPU 的运行，执行下一条指令。正在执行编程或擦除等操作的 FLASH 段是不能被访问的，因为这时该段是与片内地址数据总线暂时断开的。

由此可见，对 FLASH 模块的操作可分 3 类：擦除、写入及读出。而擦除又可分为单段擦除和整个擦除；写入可分为字写入、字节写入、字连续写入和字节连续写入，同时也可分为经过 JTAG 接口的访问与用户程序的访问。

(1) 擦除操作

对 FLASH 要写入数据，必须先擦除相应的段；要对某段中的某位编程，必须全部擦除该位所在的段。经过一次成功擦除后，该段的所有位全为 1。如果将某一位变成 0，则只有擦除操作才能将它恢复为 1。擦除可以一段一段地进行，也可以多段一起擦除，还可以整个 FLASH 模块全部擦除。

擦除操作的顺序如下：

① 选择适当的时钟源和分频因子，为时序发生器提供正确的时钟输入。

② 如果 Lock=1，则将它复位。

③ 监视 BUSY 标志位，只有当 BUSY=0 时才可以执行下一步，否则一直监视 BUSY。

④ 如果擦除一段，则设置 ERASE=1。

⑤ 如果擦除多段，则设置 MERAS=1。

⑥ 如果整个 FLASH 全擦除，则设置 ERASE=1 同时 MERAS=1。

⑦ 对擦除的地址范围内任意位置作一次空写入，用以启动擦除操作。如果空写的地址在不能执行擦除操作的段地址范围内，则写入操作不起作用。

擦除操作在满足下列条件时才能正确完成：

① 在擦除周期中，选择的时钟源始终有效。

② 在擦除周期中，不修改分频因子。如果时钟源改变或分频因子改变容易引起 FLASH 擦除时序的失控。

③ 在 BUSY=1 期间不再访问所操作的段，这包括：不从中读出，不对它写入，或不对这一段再次作擦除操作。如果发生这些操作，则会使 KEYV 置位，并产生 NMI 中断。在中断服务程序中作相应的处理。

④ 电源电压应符合芯片的相应要求，只允许有较小的容差。电压的跌落容易使电压超出正常的范围，而不能完成操作。

对 FLASH 的擦除要做 4 件事：① 对 FLASH 控制寄存器写入适当的控制位；② 监视 BUSY 位；③空写一次；④等待。每当擦除开始时，FLASH 模块要做的事情是：产生适当的时序信号；产生正确的 FLASH 操作电压。擦除周期如图 3.79 所示。在编程电压产生之后，

就由时序发生器控制整个擦除操作过程。擦除完毕，编程电压消失。整个过程段擦除需要 4 817 个时钟周期(注意：这个时钟是用户选择的时钟源经过分频之后，送到 FLASH 时序发生器的时钟信号)，而整个 FLASH 擦除需要 5 296 个时钟周期。

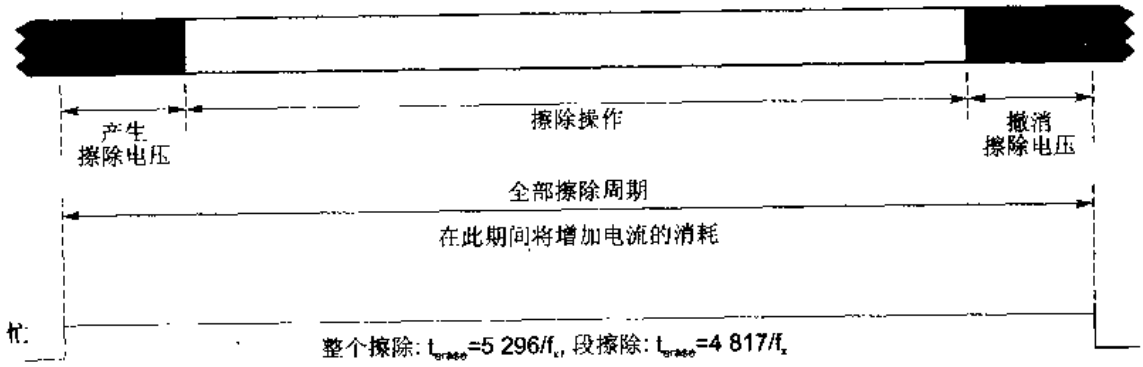


图 3.79　擦除周期

(2) FLASH 编程操作

FLASH 存储器主要用于保存用户程序或重要的数据、信息等一些掉电后不丢失的数据。只有通过对 FLASH 的编程操作，才能将这些数据写入 FLASH 存储器。有两种方式可以对 FLASH 编程：单个字或字节写入，多个字或字节顺序写入或块写入。

对 FLASH 编程按如下顺序进行：

① 选择适当的时钟源以及合适的分频因子；

② 如果 Lock=1，将它复位；

③ 监视 BUSY 位，直到 BUSY=0 时才可进入下一步；

④ 如果写入单字或单字节，则将设置 WRT=1；

⑤ 如果块写或多字、多字节顺序写入，则将设置 WRT=1，BLKWRT=1；

⑥ 将数据写入选定地址时启动时序发生器，在时序发生器的控制下完成整个过程。

块写入可用于在 FLASH 段中的一个连续的存储区域写入一系列数据。一个块为 64 字节长度。开始在 0XX00H、0XX40H、0XX80H 和 0XXC0H 等地址，结束在 0XX3FH、0XX3FH、0XXBFH 和 0XXFFH 等地址。下面是一些能写入连续数据的存储器块：

0F000H～0F03FH，0F040H～0F07FH，0F080H～0F0BFH，0F0C0H～0F0FFH 和 0F100H～0F13FH 等。

块写操作在 64 字节的分界处需要特殊的软件支持。它们是以下一些操作：

- 等待 WAIT 位，直到 WAIT=1，这表明最后一个字或字节写操作结束；
- 将控制位 BLKWRT 复位；
- 保持 BUSY 位为 1，直到编程电压撤离 FLASH 模块；
- 在新块被编程之前，等待 t_{rcv}(编程电压恢复时间)时间。

在写周期中，必须保证满足以下条件：

- 被选择的时钟源在写过程中一直保持有效；
- 分频因子不发生改变；
- 在 BUSY=1 期间，不访问 FLASH 存储器模块。

对 FLASH 的写入要做 4 件事：①对 FLASH 控制寄存器写入适当的控制位；②监视

BUSY 位；③写一个数据；④继续写，直到写完。单字或单字节写入与块写入的控制时序是不一样的。图 3.80 和图 3.81 分别为单写周期和块写周期。

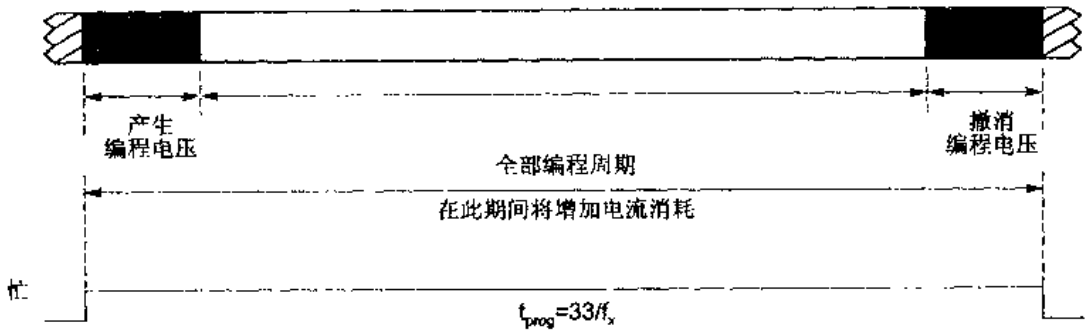


图 3.80 FLASH 单字或单字节写入周期

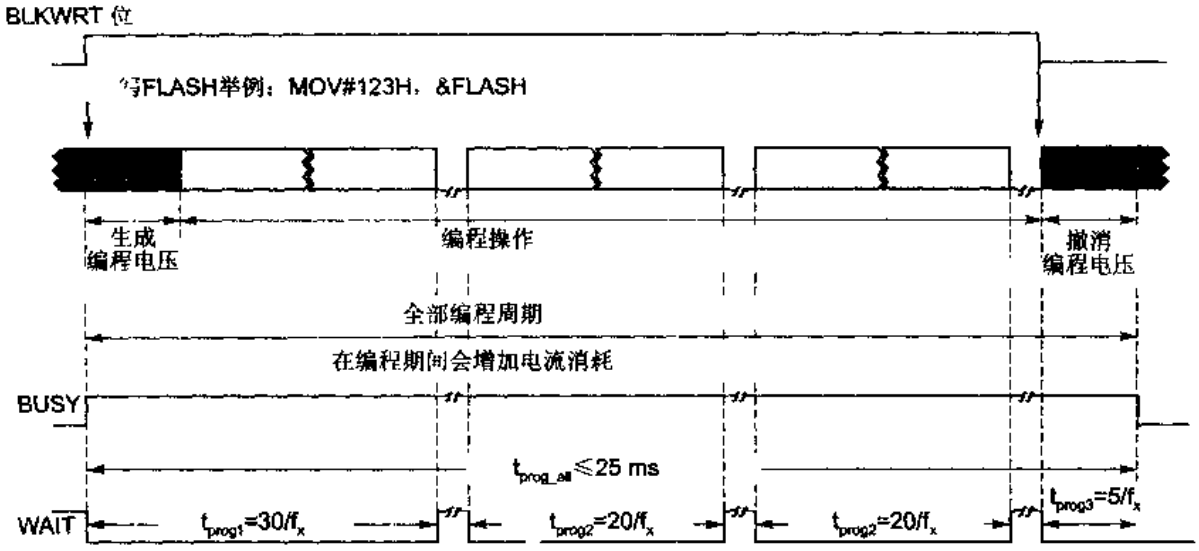


图 3.81 FLASH 块写周期

(3) FLASH 错误操作的处理

在写入 FLASH 控制寄存器控制参数时，如果写入高字节口令码错误，则引发 PUC 信号，这是错误之一；在对 FLASH 操作期间读 FLASH 内容，则会引发 ACCVIFG 状态位的设置，这是错误之二；因为在对 FLASH 操作期间，需要较长的时间，如果这时看门狗定时器内的数据将近尾声，则看门狗定时器溢出，这是错误之三；所有的 FLASH 类型的 MSP430 器件的 0 段都包含有中断向量等重要的程序代码，如果对其进行擦除操作，将会引起严重的后果，这是错误之四。

针对以上的错误操作有如下对策：

对于前两种错误一般小心就可避免；对于第三种错误，建议用户程序在进行 FLASH 操作之前先停掉看门狗定时器，等操作结束之后再打开看门狗；对于第四种错误，建议用户程序在进行 FLASH 操作之前，先将该段中的重要数据(或程序代码)保存到 RAM 中或写入到其他暂时未用的段中，等该段操作完毕再还原那些数据(或程序代码)；同时一定不要使正在执行的程序处在正要被擦除的段中；也不要在 FLASH 操作期间允许中断的发生，因为这时 CPU 是

没有办法执行程序的。

(4) FLASH 操作小结

对 FLASH 模块有 3 种操作：读、擦除及写。其中读很简单，可使用各种寻址方式，借助指令就可轻松完成。而擦除与写入就不一样了，需要按其固有的操作过程，通过控制 FLASH 模块的 3 个控制字中的相应位来完成，只有控制位的惟一组合才能实现相应的功能。表 3.16 为正确的控制位组合。

表 3.16　编程与擦除控制位

功　能	BLKWRT	WRT	Meras	Erase	BUSY	WAIT	Lock
字或字节写入	0	1	0	0	0	0	0
块写入	1	1	0	0	0	1	0
段擦除并写入	0	0	0	1	0	0	0
擦除 A 和 B 以外段	0	0	1	0	0	0	0
全部擦除并写入	0	0	1	1	0		0

在表 3.16 以外的控制位组合下操作 FLASH 模块会引起非法访问，将标志位 ACCVIFG 置位。在表中所列各功能被执行期间(BUSY=1)，对 FLASH 进行读操作，都是不对的，这时出现在数据总线的数总是 3FFFH。

FLASH 模块在 POR 信号之后，处于缺省的读模式，不须对控制位作任何操作，就可读出其中数据。

4. FLASH 模块操作举例

FLASH 存储模块除了保存用户程序外，还经常用来记录用户数据。下面的例子是在使用 FLASH 期间经常操作的范例：写入某地址一个数据；擦除某块；写入一些数据等(提醒读者：在进行 FLASH 操作期间一定要关掉看门狗定时器和所有中断)。

[例 1]　判断标志位后写入数据。

FLASH 在自身的时序控制下操作期间，会给出忙标志位(BUSY)，在忙期间用户不能对其操作，所以可通过对 BUSY 位的判断，来确定是否能进入下一步操作。

这里使用的器件为 MSP430F1121，在其 FLASH 空间的地址 0FF20H 处写入刚刚采集到的数据值 ADCOUT(ADCOUT 为片内 RAM 单元用来保存模数转换的结果)。相应的程序框图(见图 3.82)与程序清单如下：

```
TEST_BUSY1    BIT    # BUSY ,&FCTL3
              JNZ    TEST_BUSY1                ;等待不忙
              MOV    #FWKEY , &FCTL3           ;清除 Lock 位
              MOV    # ( FWKEY+WRT ) , &FCTL1  ;设置允许编程位
              MOV    &ADCOUT , &0FF20H         ;保存转换结果
TEST_BUSY2    BIT    # BUSY , &FCTL3
              JNZ    TEST_BUSY2                ;等待写完
              MOV    #FWKEY , & FCTL1          ;复位允许编程位
              XOR    # ( FXKEY+LOCK ) , &FCTL3  ;锁定，保护数据
```

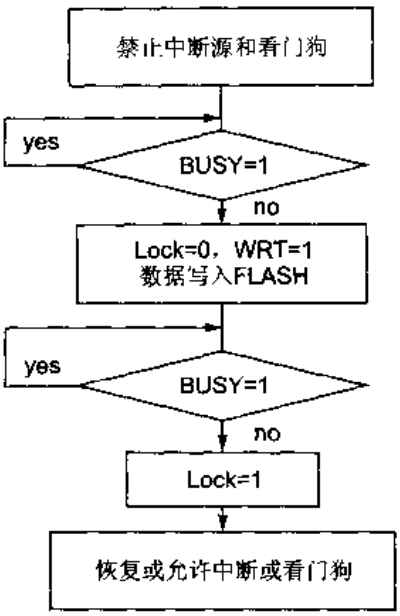


图 3.82 判断标志位写的程序框图

[例 2] 不用判断标志位直接写入数据。

在一般情况下，BUSY 位的测试是没有必要的。按照下面的程序流图写程序就可以了。图 3.83 为这种方式下的框图。下面是源代码。

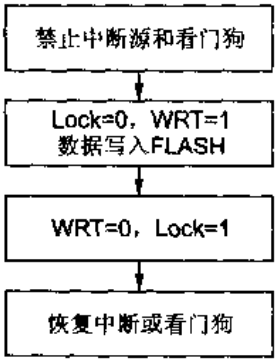


图 3.83 不判断标志位的程序框图

```
MOV   #FWKEY,&FCTL3          ;清除 Lock 位
MOV   #FWKEY+WRT,&FCTL1      ;允许写入数据
MOV   &ADCOUT,&01082H        ;保存数据到信息区 1082H 单元
MOV   #FWKEY，& FCTL1        ;复位允许编程位
XOR   #（FXKEY+LOCK），&FCTL3 ;锁定，保护数据
```

[例 3] 不用判断标志位直接写入数据序列。

在例 2 中只写了一个数据，稍加改进就可以写入很多数据。在本例中，假设片内 RAM 单元 240H～260H 为 4 次 8 路外部模拟信号的采样值，另外还有很多次模拟转换要进行，有大量的数据要保存，片内 RAM 是不够的，因为目前 MSP430 系列片内 RAM 最多只有 2 KB。那么将这些数据分批保存到 FLASH 中，目前器件 FLASH 容量最大有 60 KB。图 3.84 为将

这 32 字节数据写入信息存储区 1080 H 以后的 32 字节的程序框图。下面是完整的程序清单。

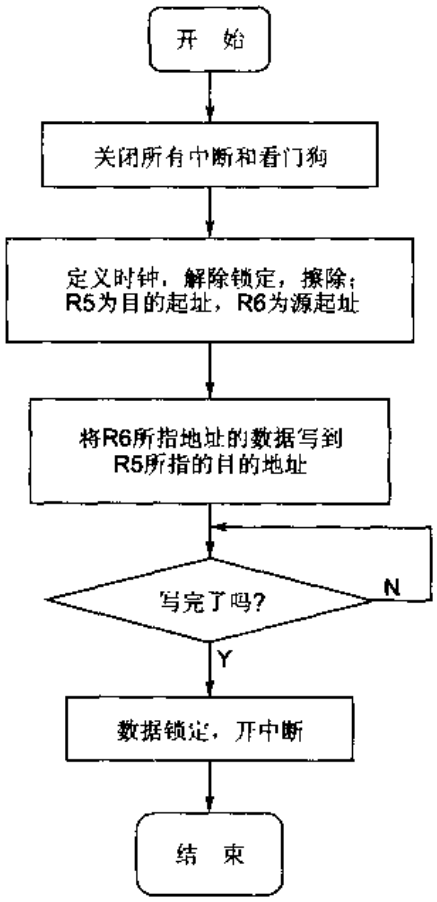


图 3.84　将片内 RAM 的若干数据写到 FLASH 的框图

```
 #include    "msp430x11x.h"
             ORG     0F000h                          ;程序开始地址
RESET        mov.w   #300h,SP                        ;初始化堆栈指针
StopWDT      mov.w   #WDTPW+WDTHOLD,&WDTCTL          ;停止看门狗
Mainloop     call    #Write_SegA                     ;调用子程序:将内存地址 240H 后
                                                     ;32 字节写入段 A
             ……
Write_SegA                                           ;子程序:将内存地址 240H 后的 32 字
                                                     ;节写入段 A
Timing       mov.w   #FWKEY+FSSEL0+FN0,&FCTL2        ;定义原始时钟
Erase_SegA   mov.w   #FWKEY,&FCTL3                   ; Lock = 0
             mov.w   #FWKEY+ERASE,&FCTL1             ;先擦除
             mov.w   #0,&01080h                      ;空写启动擦除
Prog_SegA    mov.w   #FWKEY+WRT,&FCTL1               ;允许写 FLASH
             mov     #240h , r6                      ;将要写的源数据的开始地址
             mov.w   #01080h,R5                      ;将要写入的目的开始地址
Prog_L1      mov.b   0 (r6) , 0 (R5)                 ;将源数据写入到目的地址
```

```
inc     R5                          ;增加目的地址
inc     r6                          ;增加源地址
cmp.w   #(01080h+32),R5             ;写完了吗?
jne     Prog_L1                     ;没有完则继续
mov.w   #FWKEY+LOCK,&FCTL3          ; Lock = 1 锁定刚写入的数据
Ret
ORG     0FFFEh                      ; MSP430 复位中断向量
DW      RESET                       ;
END
```

[例 4] 段擦除举例。

其实段擦除在上一个例程中已经用到——将信息段擦除，程序复制如下：首先定义时钟源；其次解锁；再次写擦除控制位；最后空写要擦除的段中任意字节或字。这是擦除一段，而其他擦除方式也很简单，只要改变第三句即可。

```
Timing      mov.w   #FWKEY+FSSEL0+FN0,&FCTL2    ;定义原始时钟
Erase_SegA  mov.w   #FWKEY,&FCTL3               ; Lock = 0
            mov.w   #FWKEY+ERASE,&FCTL1         ;允许擦除
            mov.w   #0,&01080h                  ;空写启动擦除
```

擦除信息段 A 和 B 以外的其他全部段的程序如下：

```
Timing      mov.w   #FWKEY+FSSEL0+FN0,&FCTL2    ;定义原始时钟
Erase_SegA  mov.w   #FWKEY,&FCTL3               ; Lock = 0
            mov.w   #FWKEY+MERAS,&FCTL1         ;允许擦除
            mov.w   #0,&01080h                  ;空写启动擦除
```

擦除所有全部段的程序如下：

```
Timing      mov.w   #FWKEY+FSSEL0+FN0,&FCTL2    ;定义原始时钟
Erase_SegA  mov.w   #FWKEY,&FCTL3               ; Lock = 0
            mov.w   #FWKEY+MERAS+ERASE,&FCTL1   ;允许擦除
            mov.w   #0,&01080h                  ;空写启动擦除
```

注意：正在运行的程序如果在将要擦除的段中，则一定要先将这些程序代码转移到片内 RAM 中，再执行擦除操作，否则，擦除操作完成之后，后续的程序代码也被擦除了。一般用一个搬移，将用户代码转移到片内 RAM。

[例 5] FLASH 操作 C 语言举例。

这个范例使用 C 语言实现，先擦除信息段 A，再将段 A 全写 0，然后擦除段 B，再将段 A 中的所有数据传到段 B 中。源程序如下：

```
#include  <msp430x11x1.h>
char  value;                        // 将写入信息段 A 中的 8 位数值
void  write_SegA (char value);      //函数声明
void  copy_A2B (void);              //函数声明

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;         //停止看门狗
  FCTL2 = FWKEY + FSSEL0 + FN0;     //定义 FLASH 时序发生器输入时钟
```

1. 波特率的产生

在进行异步通信时，波特率的产生是必须的。波特率部分由时钟输入选择和分频、波特率发生器、调整器和波特率寄存器等组成。串行通信时，数据接收和发送的速率就由这些构件控制。图 3.86 为其较为详细的结构。

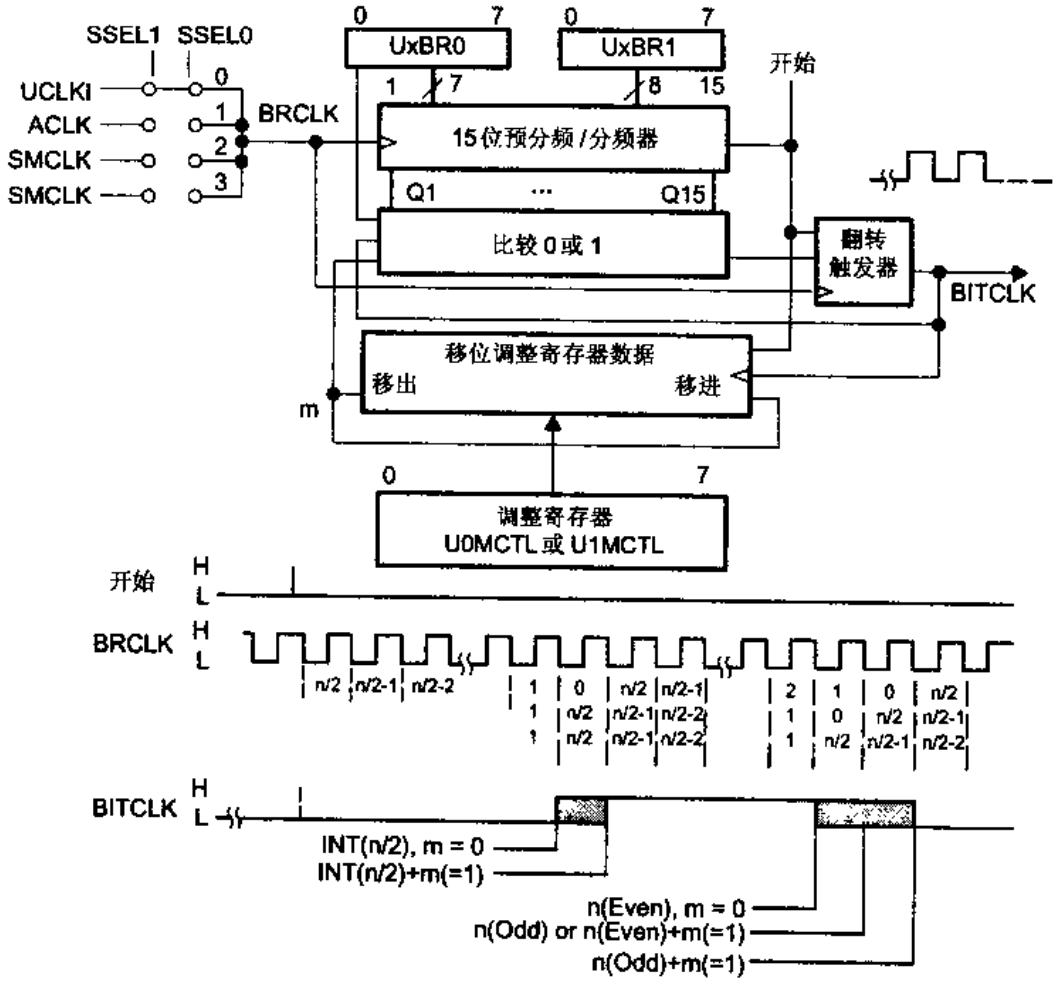


图 3.86　波特率发生器框图及举例

整个模块的时钟源来自内部 3 个时钟或外部输入时钟，由 SSEL1 和 SSEL0 选择，以决定最终进入模块的时钟信号 BRCLK 的频率。时钟信号 BRCLK 送入一个 15 位的分频器，通过一系列的硬件控制，最终输出移出与移进两移位寄存器使用的移位位时钟 BITCLK 信号。这个信号(BITCLK)的产生，由图 3.86 的下半部分可以看出，是分频器在起作用。当计数器减计数到 0 时，输出触发器翻转，送给 BITCLK 信号，所以 BITCLK 信号周期的一半就是定时器(分频计数器)的定时时间。

2. 波特率的设置与计算

MSP430 的波特率发生器使用一个分频计数器和一个调整器，分频因子 N 由送到分频计数器的时钟(BRCLK)频率和所需的波特率来决定

$$N = BRCLK / 波特率$$

如果使用常用的波特率与常用晶体产生的 BRCLK，则一般得不到整数的 N，还有小数部

分。分频计数器实现分频因子的整数部分，调整器使得小数部分尽可能准确。分频因子定义如下：

$$N = UBR + (M7 + M6 + \cdots + M0)/8$$

其中：N 为目标分频因子；

UBR 为 UxBR1 和 UxBR0 中的 16 位数据值；

Mx 为调整器寄存器(UxMCTL)中的各数据位。

那么波特率可由下式计算：

$$\text{波特率} = BRCLK / N = BRCLK /(UBR + (M7 + M6 + \cdots + M0) /8$$

可以看出，MSP430 的波特率产生方法与其他类型的 MCU 的产生方法有些不同，它不但有一个分频计数器，还有一个调整器。

分频计数器的工作原理：在大多数 MCU 中都使用预分频器与分频器的方法产生合适的波特率，就是计数器减计数到 0 或加计数到满时使输出信号翻转。

关于调整器的工作原理，下面通过一个实际波特率的例子(产生 2 400 波特率)来说明。

波特率发生器可以简化为如图 3.87 所示，由分频器和调整器组成。分频器完成分频功能，调整器的数据按每一位计算，将对应位的数据(0 或 1)加到每一次分频计数器的分频值上。

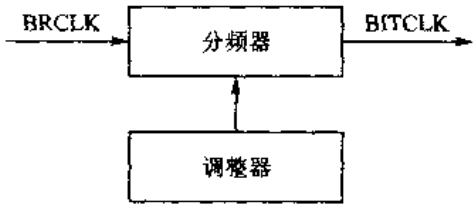


图 3.87 波特率发生器简图

在这个例子中，BRCLK＝32 768 Hz，最终要产生 BITCLK＝2 400 Hz，那么分频器的分频系数为

$$32\,768 / 2\,400 = 13.65$$

也就是说，分频器的分频系数应该是 13.65，在 MSP430 系列中它由分频器和调整器共同产生。设置分频器的计数值为 13(取整数部分)，即 UBR0＝13，UBR1＝0。用调整器的值来设置小数部分的 0.65。调整器为一个 8 位寄存器，其中每一位分别对应 8 次分频情况，如果对应位为 0，则分频器按设定的分频系数分频计数；如果对应位为 1，则分频器按设定的分频系数加 1 进行分频计数。按照这个原则，0.65×8＝5，也就是说 8 次分频计数过程中应该有 5 次加 1 分频计数，3 次不加 1 分频计数，就可以总体上满足 13.65 的分频系数。调整器的数据应该是由 5 个 1 和 3 个 0 组成，调整器内的数据是每 8 次周而复始循环使用，最低位最先调整，那么如果设置调整器的数据为“6BH”(即 01101011，也可以设为其他值，但必须是 5 个 1，而且要相对分散点)，即 UMOD＝6FH。实际上每 8 次分频时，分频器都按如下顺序进行：

$$13,14,14,13,14,13,14,14$$

每 8 次完毕再重复。实际效果的分频系数是

$$(13+14+14+13+14+13+14+14)/ 8 = 13.625$$

这个值与预定值有点误差：$\frac{13.625-13.65}{13.65}\times 100\% = -0.2\%$。这个误差在标准范围之内。当然，每一位的误差可能或大或小，但同样都在标准范围之内。每位误差如下：

$$\text{Start bit Error} = \left(\frac{\text{baud rate}}{BRCLK}\times((0+1)\times UxBR+1)-1\right)\times 100\% = 2.54\%$$

$$\text{Data bit D0 Error} = \left(\frac{\text{baud rate}}{BRCLK}\times((1+1)\times UxBR+2)-2\right)\times 100\% = 5.08\%$$

$$\text{Data bit D1 Error} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((2+1) \times \text{UxBR} + 2) - 3\right) \times 100\% = 0.29\%$$

$$\text{Data bit D2 Error} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((3+1) \times \text{UxBR} + 3) - 4\right) \times 100\% = 2.83\%$$

$$\text{Data bit D3 Error} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((4+1) \times \text{UxBR} + 3) - 5\right) \times 100\% = -1.95\%$$

$$\text{Data bit D4 Error} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((5+1) \times \text{UxBR} + 4) - 6\right) \times 100\% = 0.59\%$$

$$\text{Data bit D5 Error} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((6+1) \times \text{UxBR} + 5) - 7\right) \times 100\% = 3.13\%$$

$$\text{Data bit D6 Error} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((7+1) \times \text{UxBR} + 5) - 8\right) \times 100\% = -1.66\%$$

$$\text{Data bit D7 Error} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((8+1) \times \text{UxBR} + 6) - 9\right) \times 100\% = 0.88\%$$

$$\text{Parity bit Error} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((9+1) \times \text{UxBR} + 7) - 10\right) \times 100\% = 3.42\%$$

$$\text{Stop bit 1 Error} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((10+1) \times \text{UxBR} + 7) - 11\right) \times 100\% = -1.37\%$$

$$\text{Stop bit 2 Error} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((11+1) \times \text{UxBR} + 8) - 12\right) \times 100\% = 1.17\%$$

依此类推，则可得出其他波特率在其他输入频率下的设置参数(UBR1，UBR0，UMOD)及相应的误差。如表 3.17 所列。

表 3.17　常用波特率及其对应设置参数与对应误差表

baud rate	Divide by		ACLK (32 768 Hz)						MCLK (1 048 576 Hz)				
	ACLK	MCLK	UxBR1	UxBR0	UxMCTL	Max. TX Error/%	Max. RX Error/%	Synchr. RX Error/%	UxBR1	UxBR0	UxMCTL	Max. TX Error/%	Max. RX Error/%
75	436.91	13 981	1	B4	FF	−0.1/0.3	−0.1/0.3	±2	36	9D	FF	0/0.1	±2
110	297.89	9 532.51	1	29	FF	0/0.5	0/0.5	±3	25	3C	FF	0/0.1	±3
150	218.45	6 990.5	0	DA	55	0/0.4	0/0.4	±2	1B	4E	FF	0/0.1	±2
300	109.23	3 495.25	0	6D	22	−0.3/0.7	−0.3/0.7	±2	0D	A7	00	−0.1/0	±2
600	54.61	1 747.63	0	36	D5	−1/1	−1/1	±2	06	D3	FF	0/0.3	±2
1 200	27.31	873.81	0	1B	03	−4/3	−4/3	±2	03	69	FF	0/0.3	±2
2 400	13.65	436.91	0	0D	6B	6/3	−6/3	±4	01	B4	FF	0/0.3	±2
4 800	6.83	218.45	0	06	6F	−9/11	−9/11	±7	0	DA	55	0/0.4	±2
9 600	3.41	109.23	0	03	4A	−21/12	−21/12	±15	0	6D	03	−0.4/1	±2
19 200		54.61							0	36	6B	−0.2/2	±2
38 400		27.31							0	1B	03	−4/3	±2
76 800		13.65							0	0D	6B	−6/3	±4
115 200		9.10							0	09	08	−5/7	±7

4. 数据流的接收与发送

数据流的传送或接收主要是一个移位寄存器在起作用。在图 3.85 中，最上面部分为接收，最下面部分为发送，都是移位寄存器加缓存的结构。在接收时，当移位寄存器将接收来的数据位流组合满一个字节，就保存到接收缓存 URXBUF；在发送时，是将发送缓存 UTXBUF 内的数据一位一位地送到发送端口。发送和接收两个移位寄存器的移位时钟都是波特率发生器产生的时钟信号 BITCLK。

MSP430 的接收和发送分别用两个移位寄存器，是全双工的。

3.7.2 USART 模块的寄存器

MSP430 的 USART 模块也用到了很多寄存器进行通信控制、波特率设置、数据缓存等。这里分别对它们进行介绍。MSP430 器件中有的型号有两个通信硬件模块 USART0 和 USART1，因此它们有两套寄存器，这里用 x 表示 0 和 1。MSP430 的 USART 模块的各寄存器都在字节地址范围内，必须使用字节访问方式予以操作。MSP430F149 的串行口寄存器如图 3.88所示。

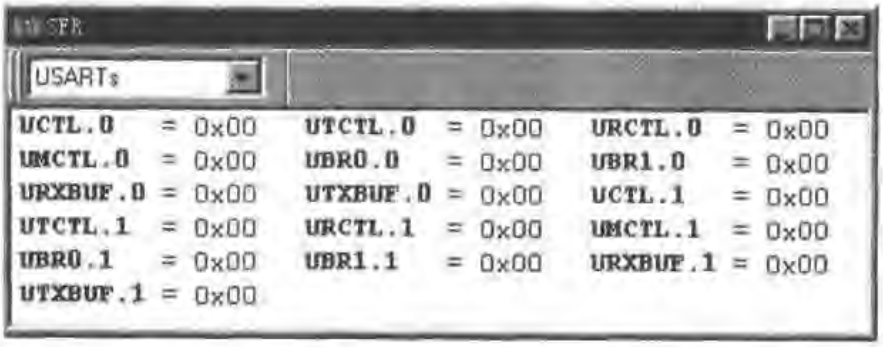


图 3.88 MSP430F149 的串行口寄存器

(1) UxCLT 控制寄存器

USART 模块的基本操作由此寄存器的控制位决定。如通信协议的选择、通信模式及校验位等。其中 BIT5，BIT6 及 BIT7 三位在 SPI 模式下没有用到，在 UART 模式下全都用了。该寄存器的各位定义如下：

7	6	5	4	3	2	1	0
PENA	PEV	SP	CHAR	Listen	SYNC	MM	SWRST

PENA 在异步通信时，校验允许位。如果禁止校验，则发送时不产生校验位，接收时也不期望收到这一位。因为校验位不是数据位之一，接收到的校验位不送入接收缓存器(URXBUF)，在地址位多机模式中，地址位包括在校验计算。

0 校验禁止；

1 校验允许。

PEV 奇偶校验位。如果允许校验(PENA=1)，则 PEV 位按发送或接收字符、地址位

和校验位中“1”的数量定义奇校验或偶校验。

0　奇校验；

1　偶校验。

SP　停止位。在异步方式下，决定发送时停止位数，但接收时接收器只检测 1 位停止位。

0　1 位停止位；

1　2 位停止位。

CHAR　字符长度。选择字符以 7 位或 8 位发送。7 位时不用发送或接收缓存的最高位，补 0。

0　7 位；

1　8 位。

Listen　选择是否将发送数据由内部反馈给接收器。

0　无反馈；

1　有反馈，发送信号由内部反馈给接收器，自己发送的数据同时被自己接收，通常被称为自环模式。

SYNC　USART 模块的模式和功能选择。

0　UART 模式(异步)；

1　SPI 模式(同步)。

MM　多机模式选择位(异步模式)，主机模式或从机模式选择位(同步模式)。MSP430 的 USART 模块支持两种多机协议：线路空闲和地址位。多机模式的选择将影响自动地址解码功能的实现方式。而在同步方式下数据通信时又分主机模式或从机模式。

0　线路空闲多机协议(异步模式)，选择从机模式(在同步方式时)；

1　地址位多机协议(异步模式)，选择主机模式(在同步方式时)。

SWRST　控制位。该位的状态影响着其他一些控制位和状态位的状态。在串行口的使用过程中，这一位是比较重要的控制位。一次正确的 USART 模块初始化应该是这样的顺序：先在 SWRST＝1 的情况下设置串口；然后设置 SWRST＝0；最后如果需要中断，则设置相应的中断使能。

1　如果该位置位，则 USART 状态机和操作运行标志位都被初始化成复位状态(URXIFG＝URXIE＝UTXIE＝0，UTXIFG＝1)；同时所有受影响的逻辑保持在复位状态，直到 SWRST 位被复位。这就意味着，当系统复位之后，只有对 SWRST 位复位，USART 的功能才能被重新允许；但接收和发送允许标志 URXE 和 UTXE 不受 SWRST 控制位的影响。

该位会使这些位复位：URXIE，UTXIE，URXIFG，RXWAKE，TXWAKE，RXERR，BRK，PE，OE 和 FE。同时会使 UTXIFG 和 TXEPT 位置位。

0　USART 模块被允许。

(2) UxTCTL　发送控制寄存器

UxTCTL 寄存器控制与数据发送操作相关的 USART 模块硬件。其中 BIT1 和 BIT7 两位在 UART 模式下没有用到，而在 SPI 模式下 BIT2 和 BIT3 未用。该寄存器的各位定义

如下：

7	6	5	4	3	2	1	0
CKPH	CKPL	SSEL1	SSEL0	URXSE	TXWAKE	STC	TXEPT

CKPH　时钟相位控制位(SPI 模式)。该位控制 SPICLK 信号的相位。

0　使用正常的 UCLK 时钟；

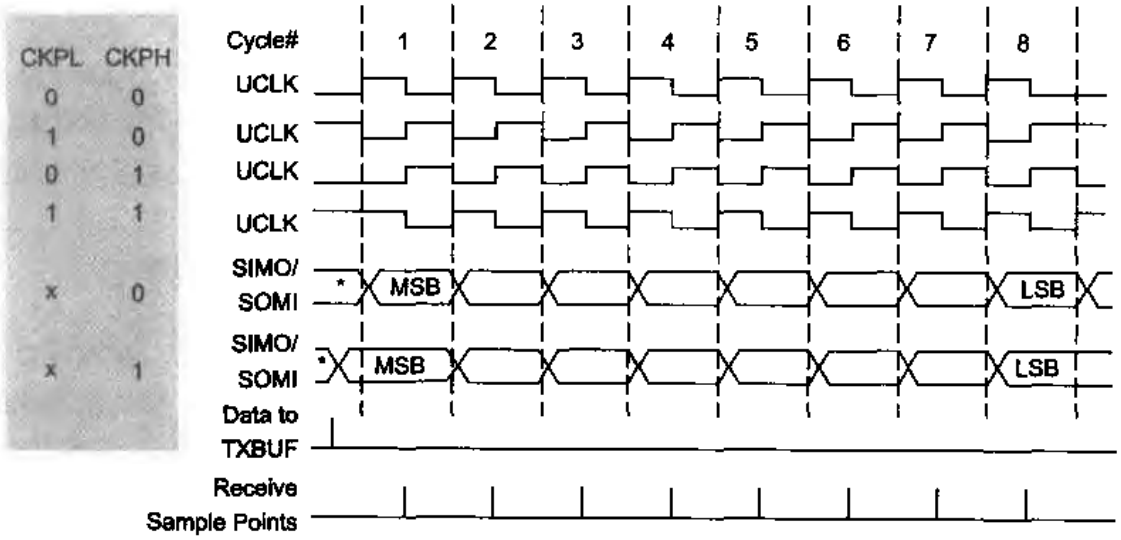
1　UCLK 时钟信号被延迟半个周期。

CKPL　时钟极性控制位。在 UART 模式下，该位控制 UCLKI 信号的极性；在 SPI 模下，控制 SPICLK 信号的极性。

0　在异步模式，UCLKI 信号与 UCLK 信号极性相同；在同步模式，表示时钟信号的低电平为无效电平，数据在 UCLK 的上升沿输出，输入数据在 UCLK 的上升沿被锁存。

1　在异步模式，UCLKI 信号与 UCLK 信号极性相反；而在同步模式，表示时钟信号的高电平为无效电平，数据在 UCLK 的下降沿输出，输入数据在 SPI-CLK 的上升沿被锁存。

关于在同步模式下，数据输入或输出与时钟信号的相位和极性之间的关系如图 3.89 所示。



* 以前的数据位

图 3.89　数据输入或输出与时钟信号的相位和极性之间的关系

当运行在 CKPH=1 时，USART 模块的同步模式发送移位寄存器装入数据，并在 UCLK 的第一个沿前准备好数据的第一位。数据位在 UCLK 的第一个沿锁存，在第二个沿发送。

SSEL1，SSEL0　时钟源选择位。该两位确定波特率发生器的时钟源。

0　选择外部时钟 UCLK；

1　选择辅助时钟 ACLK；

2　选择子系统主时钟 SMCLK；

3　选择子系统主时钟 SMCLK。

在同步方式时，只有选择了主机模式才需要由这两位定义用于波特率发生器的时钟源；而在从机模式下，时钟信号来源于主机，所以这时该两位无意义。

URXSE　接收触发沿控制位。该位在同步方式时没有使用。

0　没有接收到数据。

1　接收到数据，请求接收中断服务。注意：为了能正确地得到中断服务，必须设置好相应的使能位 URXIE 和 GIE；同时如果所选择的时钟源一直有效，则尽管 CPU 处于低功耗模式，也能进行接收操作。

TXWAKE　多处理器通信传送控制位（该位在同步方式时不用）。通过装入 UTXBUF 开始一次发送操作，使用该位的状态来初始化地址鉴别特性。硬件能自动清除，SWRST 也能清除它。

STC　STE 引脚选择位。（异步模式不用）在同步模式下，从机发送控制位 STC 选择 STE 引脚信号在主机和从机中使用。

0　选择 SPI 的 4 线模式，STE 信号用于主机以避免总线冲突，或用于从机模式的控制发送或接收允许。

1　选择 SPI 的 3 线模式，则此时 STE 引脚信号在主机、从机模式中不起作用。

TXEPT　发送器空标志。在异步模式与同步模式时不一样。

0　在异步模式时表示发送缓冲器（UTXBUF）有数据。在同步的主机模式也一样：在数据写入 UTXBUF 时为 0。

1　在异步模式时，表示发送移位寄存器和 UTXBUF 空。在同步的主机模式时同样表示发送移位寄存器和 UTXBUF 空；但在同步的从机模式下，当发送移位寄存器和 UTXBUF 空时，TXEPT 并不发生置位操作。

（3）URCTL　接收控制寄存器

URCTL 控制与接收操作相关的串行口硬件，并保存由最新写入接收缓存 URXBUF 的字符引起的出错状况和唤醒条件。一只有 PE，FE，OE，BRK，RXERR 和 RXWAKE 等位的任何一位被置位，都不能通过接收到下一个字符来复位。它们的复位要通过访问接收缓存或串行口的软件复位，或系统复位，或直接用指令修改。URCTL 寄存器的各位定义如下（在同步 SPI 模式下，只用到了 2 位：FE 和 OE）：

7	6	5	4	3	2	1	0
FE	PE	OE	BRK	URXEIE	URXWIE	RXWAKE	RXERR

FE　帧错标志。

0　没有帧错。

1　帧错。在异步模式时，当一个接收字符的停止位为 0 并被装入接收缓存，很明显，接收缓存中的数据不是对方发送过来的数据，为一个错误的帧，那么帧错标志被设置为 1，即使在多停止位模式时也只检测第一个停止位。同样，丢失停止位意味着从起始位开始的同步特性被丧失，也是一个错误

帧。在同步的4线模式时，因总线冲突使有效主机停止，并在STE引脚信号出现下降沿时使FE位设置为1。

PE 校验错标志位(同步模式不用)。

0 校验正确。

1 校验错。当接收字符中1的个数与它的校验位不相符，并被装入接收缓存时，发生校验错，设置该位为1。

OE 溢出标志位。

0 无溢出。

1 有溢出。当一个字符写入接收缓存URXBUF时，而一个字符还没有被读出，这时前一个字符因被覆盖而丢失，发生溢出(同步与异步情况相同)。

BRK 打断检测位。

0 没有被打断。

1 被打断。当发生一次打断同时URXEIE置位时，该位被设置为1，表示接收过程被打断过。RXD线路从丢失的第一个停止位开始连续出现至少10位低电平被识别为打断。

URXEIE 接收出错中断允许位。

0 不允许。接收到的出错字符不改变URXIFG标志位。

1 允许中断。根据URXWIE位的设置，所有字符都能使标志位URXIFG置位。

URXWIE 接收唤醒中断允许位。

0 接收到的每一个字符都将使标志位URXIFG置位。

1 只有作为地址的字符才能使标志位URXIFG置位。

表3.18说明了在各种条件下URXEIE和URXWIE对URXIFG的影响。

表3.18 在各种条件下URXEIE和URXWIE对URXIFG的影响

URXEIE	URXWIE	字符出错	地址字符	接收字符后的标志位URXIFG
0	X	1	X	不变
0	0	0	X	置位
0	1	0	0	不变
0	1	0	1	置位
1	0	X	X	置位(接收所有字符)
1	1	X	0	不变
1	1	X	1	置位

RXWAKE 接收唤醒检测位。

0 没有被唤醒。

1 唤醒。当接收的字符是一地址字符同时被送入接收缓存时，该位被设置为1。有两种多机模式。在地址位多机模式时，接收字符地址位置位时，该机被唤醒，RXWAKE=1；在线路空闲多机模式时，在接收到字符前检测到URXD线路空闲时，该机被唤醒，RXWAKE=1。

RXERR　接收错误标志位。

0　没有接收错误。

1　有接收错误。当 RXERR＝1 时，表明有一个或多个出错标志(FE，PE，OE 和 BRK 等)被置位。该位不能自动复位，需用户指令清除。

(4) UxBR0，UxBR1　波特率选择寄存器 0 和 1

该两寄存器用于选择波特率发生器的分频器分频因子的整数部分。其中 UxBR0 为低字节，UxBR1 为高字节。两字节合起来为一个 16 位字，称为 UBR。在异步通信时，UBR 的允许值不小于 3，即 3≤ UBR＜ 0FFFFH。如果 UBR＜ 3，则接收和发送会发生不可预测的情况。在同步通信时，最小的分频因子为 2。

(5) UxMCTL　波特率调整控制寄存器

如果波特率发生器的输入频率 BRCLK 为所需波特率的整数倍，则这个倍率就是分频因子，将它写入 UBR 寄存器即可。但如果波特率发生器的输入频率 BRCLK 不是所需波特率的整数倍，带有一小数，则整数部分写入 UBR 寄存器，小数部分由调整控制寄存器 UxMCTL 的内容反映。波特率由以下公式计算：

$$\text{波特率} = \text{BRCLK} /(\text{UBR}+(\text{M7}+\text{M6}+\cdots+\text{M0})/ 8)$$

其中 M0，M1，…，M6 及 M7 为调整控制寄存器 UxMCTL 中的各位。调整寄存器中的 8 位分别对应 8 次分频。如果 MI ＝1，则相应次的分频增加一个时钟周期；如果 MI ＝0，则分频计数值不变。

在同步通信时不需要调整寄存器，使用时最好全部写 0。

(6) URXBUF　接收数据缓存

当接收移位寄存器接收的数据满时，将接收的数据转移到接收数据缓存 URXBUF。读取其中数据，将使这样一些位复位：接收出错位 RXERR、接收唤醒检测位 RXWAKE 及中断标志位 URXIFG。如果通信模式使用 7 位方式，则 URXBUF 的最高位总为 0。

当接收和控制条件为真时，接收缓存装入当前接收到的字符，如表 3.19 所列。

表 3.19　当接收和控制条件为真时接收数据缓存结果

条　件		结　果			
URXEIE	URXWIE	装入 URXBUF	PE	FE	BRK
0	1	无差错地址字符	0	0	0
1	1	所有地址字符	X	X	X
0	0	无差错字符	0	0	0
1	0	所有字符	X	X	X

(7) UTXBUF　发送数据缓存

当前要发送的数据保存在发送数据缓存中，发送移位寄存器中的数据源自 UTXBUF。将数据写入 UTXBUF 将初始化发送功能。如果发送移位寄存器为空或就要为空，则数据的发送立即开始。注意：只有当 UTXBUF 为空时，写入缓存的数据才算有效。

在同步方式时的主机模式下，将数据写入 UTXBUF 将初始化发送功能。如果发送移位寄存器为空或就要为空，则数据的发送立即开始。在选择 7 位方式时，发送缓存中的数据须左对齐，因为最高有效位最先发送。

3.7.3 异步模式

在异步模式下，接收器自身实现帧的同步，但外部通信设备并不使用这一时钟源。波特率的发生是在本地完成的。也就是说，要正确地进行异步通信，通信的双方必须使用相同的波特率。

1. 异步通信的帧格式

异步帧格式由 1 位起始位、7 位或 8 位数据位、校验位（可奇/可偶/可无）、1 位地址位（地址位模式时）和 1 位或 2 位停止位组成。而每位数据位的时值由波特率发生器决定。异步帧格式如图 3.90 所示。

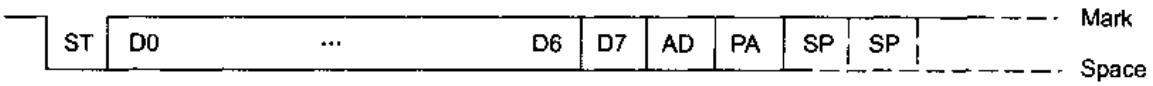


图 3.90 异步帧格式

接收操作以收到有效起始位开始。起始位由检测 URXD 端口的下降沿开始，然后以 3 次采样多数表决的方法取值。

2. 线路空闲多机模式

在异步模式下，支持两种多机模式：线路空闲多机模式和地址位多机模式。由控制寄存器中的 MM 位确定使用哪种多机模式。这两种模式都使用激活 TXTWAKE 和 RXWAKE 位来唤醒串口。

当 MM=0 时，为线路空闲多机模式。在这种模式下，数据块被一段空闲的时间分隔。在字符的第一个停止位之后，收到 10 个以上的 1，则表示检测到接收线路空闲，如图 3.91 所示。

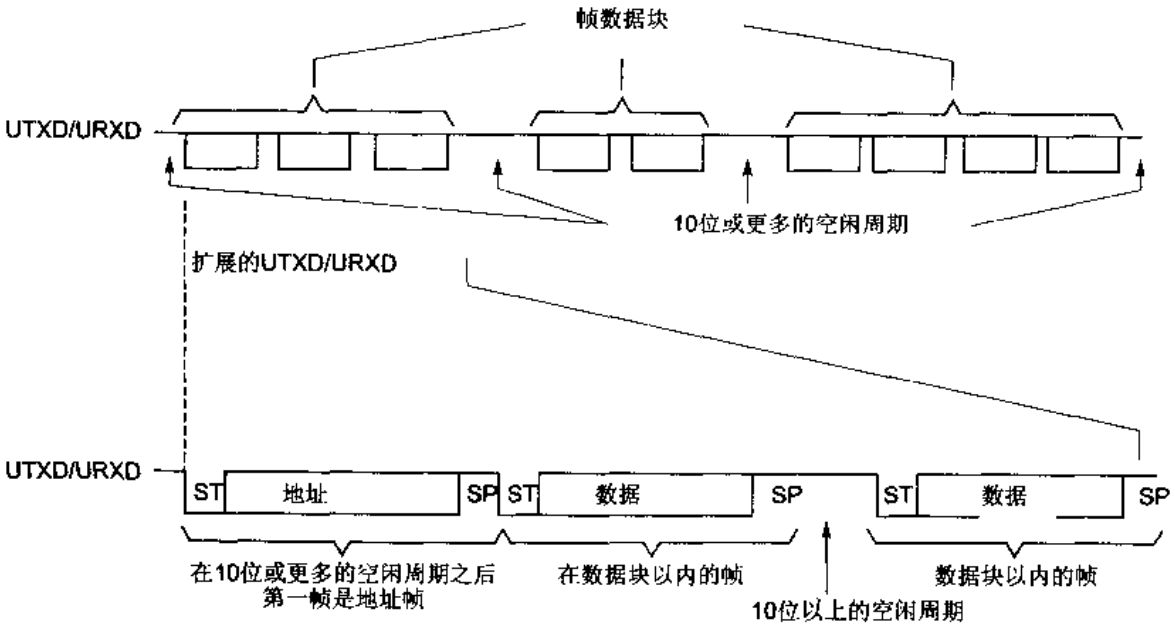


图 3.91 线路空闲多机模式

如果采用两位停止位，则第二个停止位被认为空闲周期的第一个传号。空闲周期的第一个字符是地址字符。RXWAKE 位可用于地址字符的标志。当接收到的字符是地址字符时，

表明该字符是一个地址。当接收字符是地址时，RXWAKE 置位，并且将接收的字符送入接收缓存 URXBUF(当接收被允许时)。

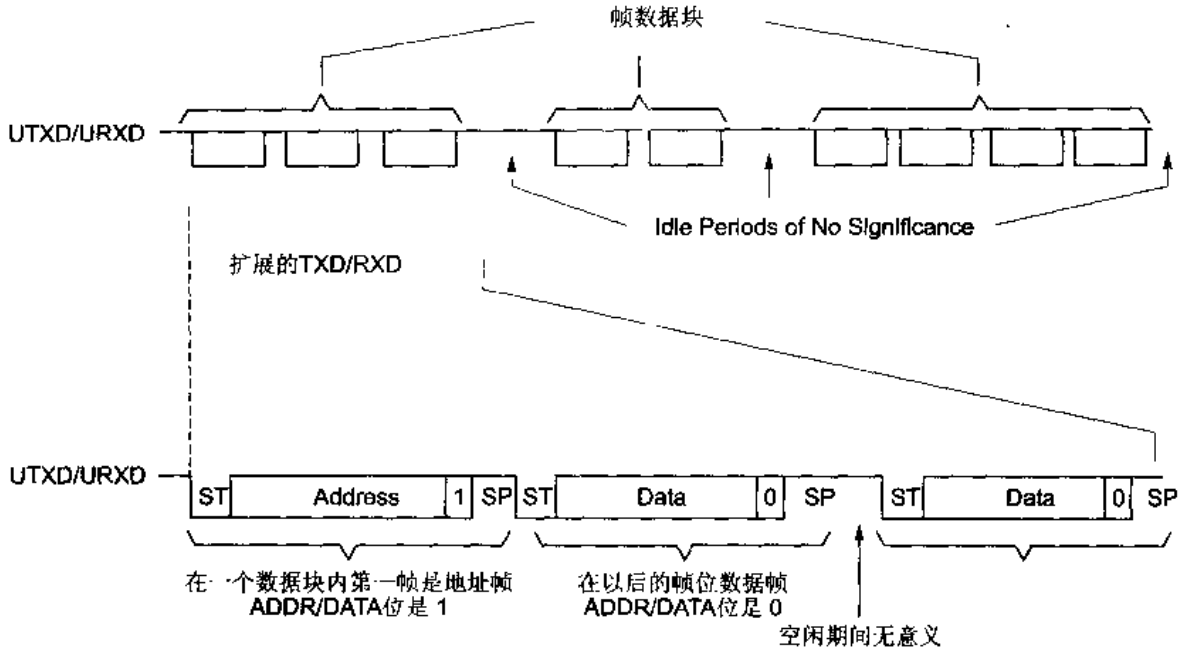


图 3.94 地址位多机模式

在 USART 的 URXWIE=1 时，数据字符在通常方式的接收器内拼装成字节，但它们不会被送入接收缓存，也不产生中断。只有当接收到一个地址位为 1 的字符时，接收器才被激活，接收到的字符被送往 URXBUF，同时 URXIFG 被置位。如果有错误，则相应的错误标志被设置。应用软件在判断后作出相应的处理。在地址位多机模式下，通过写 TXWAKE 位控制字符的地址位。每当字符由 UTXBUF 传送到发送器时，TXWAKE 位装入字符的地址位，再由 USART 将 TXWAKE 位清除。

4. 异步方式的中断

USART 模块有接收和发送两个独立的中断源。使用两个独立的中断向量，一个用于接收中断事件，一个用于发送中断事件。USART 模块的中断控制位位于 SFR 中。其中，有如下一些位：

- URXIFG 接收中断标志；
- URXIE 接收中断使能；
- URXE 接收允许；
- UTXIFG 发送中断标志；
- UTXIE 发送中断允许；
- UTXE 发送允许。

MSP430 的异步收发器是完全独立操作的，但使用同一个波特率发生器，则接收器和发送器使用相同的波特率。

接收允许位 URXE 的置位或复位，将设置允许或禁止接收器从 URXD 数据线路接收数据。当禁止接收时，如果已经开始一次接收操作，则会在完成后停止下一次接收操作；如果无

接收操作，则在进行中立即停止接收操作，连起始位的检测也被禁止。

发送允许位 UTXE 的置位或复位，将设置允许或禁止发送器将串行数据数发送到线路。当 UTXE=0 时，已被激活的发送并不停止操作，而将在完成发送已经写入发送缓存内的全部数据后被禁止。在 UTXE=0 之前写入发送缓存的数据有效。也就是说，当 UTXE 复位时，发送缓存可照样写入数据，但数据不发送到串行线路。而一旦 UTXE 被置位，缓存内的数据立即发送到线路。

每当接收到字符，且装入接收缓存时，接收中断标志 URXIFG 被置位。此时可将数据由接收缓存取出。

3.7.4　同步模式

MSP430 的串行通信模块不但可实现异步通信，还可实现同步通信(SPI)。在同步模式下，允许 7 位或 8 位数据流以内部或外部确定的速率移入或移出 MSP430。当 USART 模块控制寄存器 UCTL 的 SYNC 位置位时，串行模块工作在 SPI 模式。MSP430 的同步通信有如下特点：

- 由 SOMI，SIMO，UCLK 及 STE 等引脚连接，支持 3 线或 4 线 SPI 操作；
- 可选主模式与从模式；
- 接收和发送有各自的寄存器，且接收和发送为双缓存；
- 移位时钟的极性和相位可编程；
- 主模式的时钟频率可控；
- 7 位或 8 位字符长度。

1. 同步操作

在同步模式时，主机提供时钟与数据，从机利用这一时钟接收数据，或在这一时钟下送出数据。有 4 线制和 3 线制两种连接方法，4 线 SPI 模式用附加的控制线来允许从机数据的接收和发送，它由主机控制。

(1) SIMO　从进主出

方向由 SIMODIR 定义，当 SIMODIR=0 时定义为输入。

SIMODIR=[SYNC. AND. MM. AND. (STC. OR. STE)]

选择 SPI+主模式，也就是选择输出模式。当 STC=0，选择 4 线模式，由 STE 上加低电平来强制进入输入方向。

(2) SOMI　从出主进

方向由 SIMODIR 定义，当 SIMODIR=0 时定义为输入。

SOMIDIR=[SYNC. AND. . NOT. (MM)]. OR. [STC. OR. . NOT. (STE)]

选择 SPI+主模式，也就是选择输出模式。当 STC=0，选择 4 线模式，由 STE 上加低电平来强制进入输入方向。

(3) UCLK　USART 时钟信号

该信号由主机驱动，从机用它发送和接收数据。方向由 UCLKDIR 定义，UCLKDIR=0 时为输入。

UCLKDIR=[SYNC. AND. MM. AND. (STC. OR. STE)]

当 STC=0，选择 4 线模式，由 STE 上加低电平来强制进入输入方向。

(4) STE 从机发送允许信号

用于 4 线模式中控制多主从系统中的多个从机。

图 3.95 和图 3.96 是 MSP430 的 USART 模块使用同步模式时,与另一设备的连接。

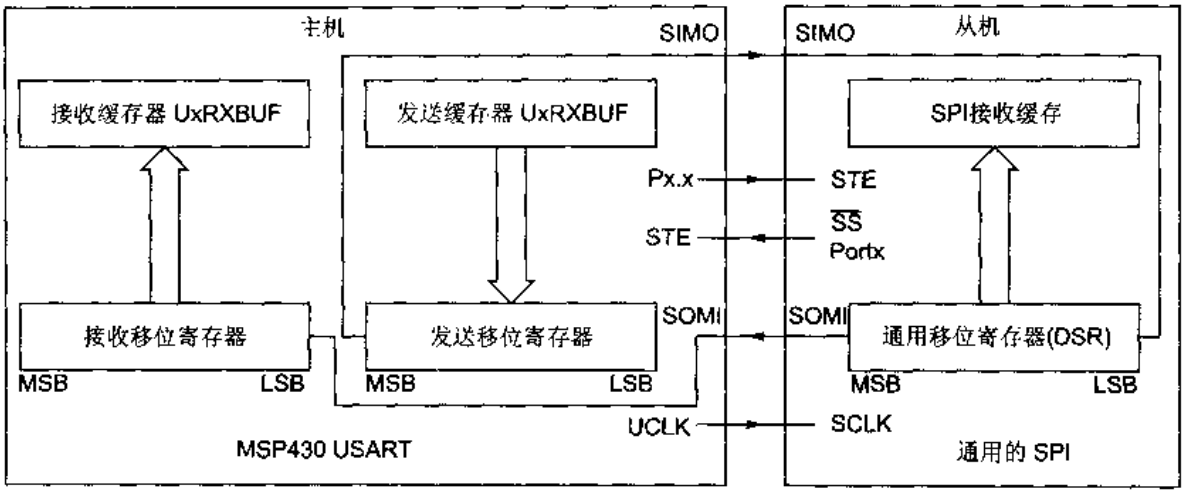


图 3.95 MSP430 的 USART 模块为主机在同步模式下与其他从设备相连

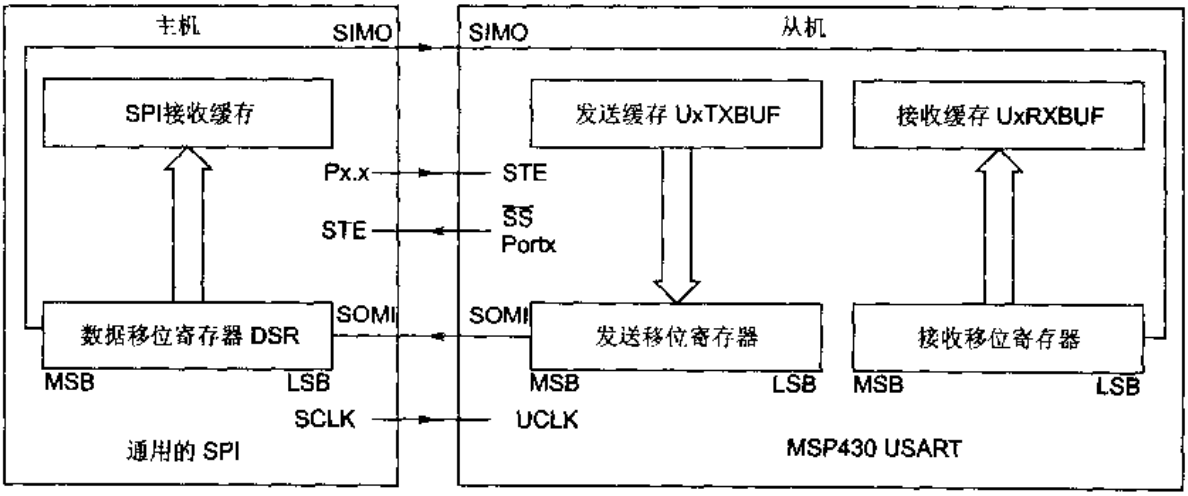


图 3.96 MSP430 的 USART 模块为从机在同步模式下与其他从设备相连

主机通过发送 UCLK 信号用于初始化发送。主机在一个时钟跳变沿将数据移出发送移位寄存器,在另一个反向跳变沿移入接收寄存器。从机数据的移位操作与此相同,但用公共的移位寄存器接收和发送数据。主机与从机数据的发送和接收是同时进行的,因为使用的是不同的线路。

图 3.97 是 7 位字长串行同步数据发送的例子,接收移位寄存器的初始内容为 00。

那么会发生如下系列事件:

A:从机写 98H 到 DSR,并等待主机将数据移出。

B:主机写 0B0H 到 UTXBUF,它将立刻送往发送移位寄存器,且开始发送。

C:完成第一个字符,中断标志置位。

D:从机由它的接收缓存读出 58H(右对齐)。

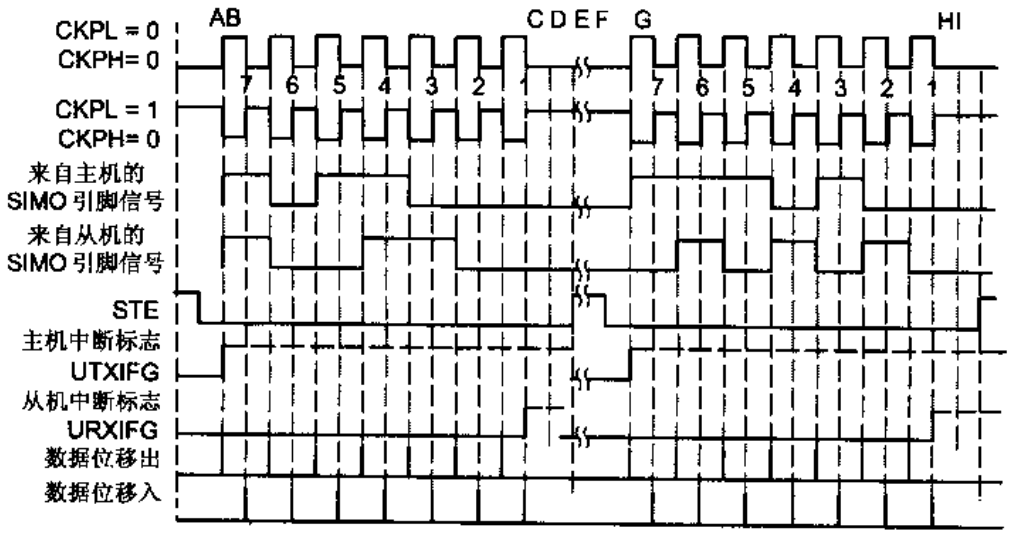


图 3.97　同步串行数据通信

E：从机写 54H 到 DSR，并等待主机将数据移出。

F：主机从它的接收缓存 URXBUF 读出 4CH(右对齐)。

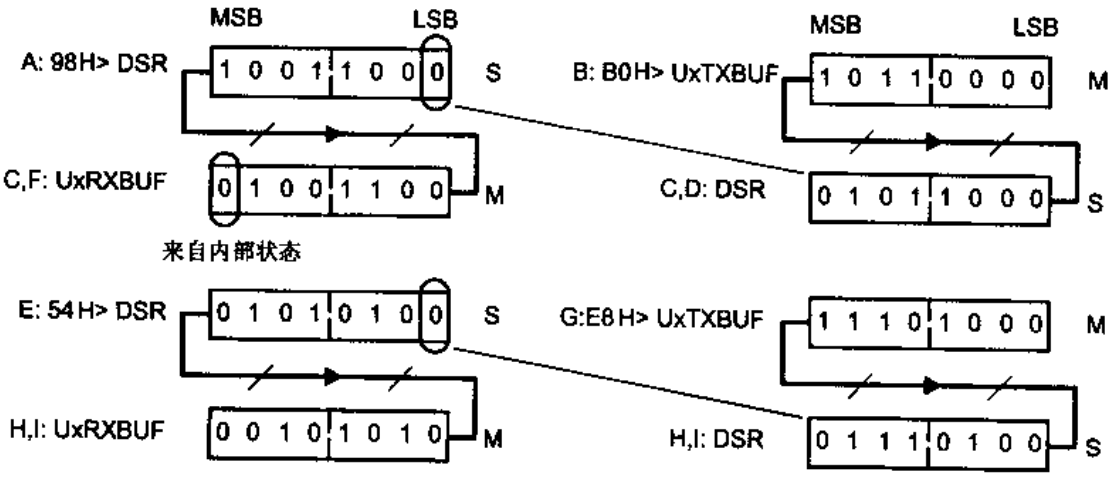
G：主机写 0E8H 到发送缓存 UTXBUF，并开始发送。

如果 USART 为从机模式，则 D 事件后至 G 事件前不需要 UCLK。如果为主机模式，则内部需要两个时钟(内部产生，非 UCLK)来终止发送和接收第一个字符，并准备下一个字符的发送和接收。

H：完成第二个字符，中断标志置位。

I：主机收到 2AH，从机收到 74H(右对齐)。

这些过程如图 3.98 所示(当为 7 位模式时，RXBUF 的 MSB 读出为 0)。



S　从；M—主

图 3.98　数据传输循环

2. SPI 模式中的主机模式

当控制寄存器中的 MM=1 时，工作在主机模式。USART 模块通过在 UCLK 引脚上的

```
unsigned int i;
WDTCTL = WDTPW + WDTHOLD;                 停止看门狗
BCSCTL1 |= XTS;                           ACLK 使用高频晶振
do

IFG1 &= ~OFIFG;                           清除 OSCFault 标志
for (i = 0xFF; i > 0; i--);

while ((IFG1 & OFIFG) == OFIFG);          如果 OSCFault = 1
BCSCTL2 |= SELM1+SELM0;                   MCLK = LFXT1 (safe)
UCTL0 = CHAR;                             8 位字符
UTCTL0 = SSEL0;                           UCLK = ACLK
UBR00 = 0x45;                             在 8 MHz 下进行 115 200 波特率通信
UBR10 = 0x00;                             在 8 MHz 下进行 115 200 波特率通信
UMCTL0 = 0x00;                            调整寄存器
ME2 |= UTXE0 + URXE0;                     使能 USART0 TXD/RXD
IE2 |= URXIE0;                            使能 USART0 RX 中断
P3SEL |= 0x30;                            P3.4,5 = USART0 TXD/RXD
P3DIR |= 0x10;                            P3.4 为输出
_EINT();                                  开总中断
while(1)

  BIS_SR(CPUOFF);                         低功耗，如果接收到数据，则接收中断处理
  NOP();

}

#pragma vector=UART0RX_VECTOR
__interrupt void usart0_rx (void)
{
  while ((IFG2 & UTXIFG0) == 0);          USART0 TX 接收缓存有数据吗?
  TXBUF0 = RXBUF0;                        RXBUF0 to TXBUF0 将接收到的数据
```

3.8　MSP430 模数转换模块

MSP430 系列单片机大部分都内嵌模数转换器模块，且转换精度在 10 位、12 位及 14 位不等。而其他没有硬件模数转换器模块的型号也可以利用内嵌的模拟比较器使用软件的办法实现模数转换(请参考模拟比较器部分)。所以 MSP430 系列在数据采集等需求的应用中使用非常方便。这里只介绍 MSP430 的 3 个不同精度模数转换器硬件模块。

3.8.1　ADC10 模数转换模块

ADC10 是在 MSP430X1XX2 内的高速 10 位模数转换器。

1. 特征与结构概述

ADC10 的主要特征归纳如下：

- 最大转换速率 200 ksps；
- 10 位的转换，其非线性微分误差是 1LSB，其非线性积误差是 1LSB；
- 采样和保持电路建立；
- 内置 RC 振荡器，可用于产生采样时序；
- 温度测量传感器内置；
- 10 路模拟输入（分成两组外部参考输入），对于 MSP430x11x2 有 4 路外部输入，对于 MSP430x12x2 有 8 路外部模拟输入；
- 4 个内部转换通道：温度、AV_{cc} 及外部参考（包括正、负两路）；
- 芯片内部产生参考电压：1.5 V 或 2.5 V，由软件选择；
- 每个通道可以单独选择参考电压：正的和负的，内部的和外部的；
- 多种可选转换时钟源 ADC10OSC（RC oscillator in ADC10），ACLK，MCLK 和 SMCLK；
- 通用的转换模式包括单通道单次、单通道多次、序列通道单次及序列通道多次 4 种模式；
- 每次转换之后，将转换的结果放入对应寄存器（缓冲器）；
- ADC 的核心和参考电压发生器可以关闭以节省能耗。

ADC10 模数转换器的结构框图如图 3.99 所示。

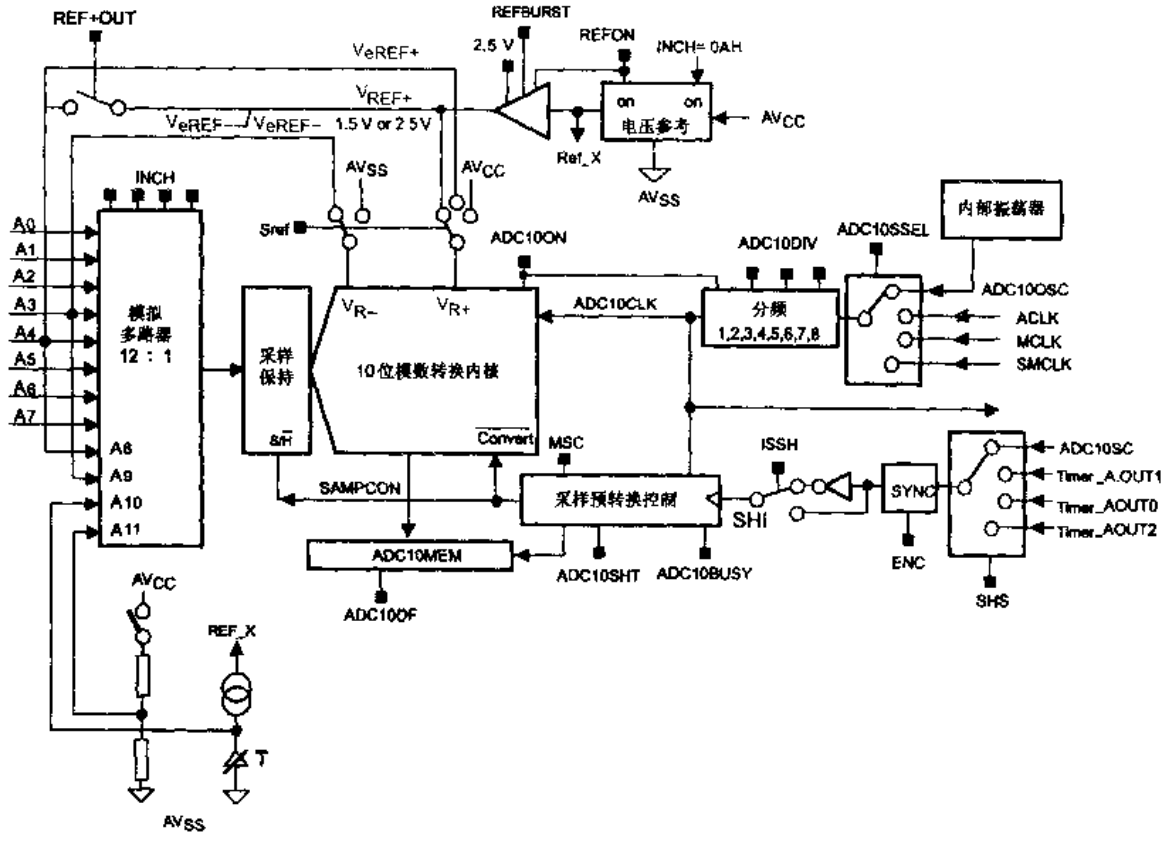


图 3.99 ADC10 结构框图

ADC10 模数转换器有以下 4 个主要的功能模块：

- 带有采样和保持的 ADC 内核；
- 参考电压发生器；
- 转换时钟源的选择和控制电路；
- 采样与转换时序控制电路。

ADC10 的核心是带采样和保持的 10 位模数转换器内核，其输入为 12 选 1 的模拟多路器，转换所需的参考电压可由内部发生也可外接，转换所需的时序可有非常丰富的选择。

ADC10 能够转换 8 个输入或者 4 个内部电压。这 4 个内部通道用于测量温度(经过芯片上的温度二极管)、V_{CC}(经过 $V_{CC}/2$)和应用正的或负的参考电压 Ve_{REF+} 和 V_{REF-}/Ve_{REF-}。

ADC 能使用内部或外部或两者联合的参考电压(在寄存器部分介绍)。

ADC 有通用的采样和保持电路。一个采样占用 4 个 ADC10CLKs 周期，它能由软件(ADC10SC)或者信号 ADC10I1，ADC10I2 或 ADC10I3 触发。典型的情况是来自于 MSP430 定时器，例如 Timer_A 的内部定时信号。

因为采样定时器，用户有几个 ADC10 转化器时钟可以选择。ADC10 转化时钟可以是 ACLK，MCLK 或 SMCLK，或者将 ADC10 外围的振荡器分频并进行选择。被选择的时钟源可以 1～8 分频。

ADC10 的内核为 10 位模数转换器，所以只能有 10 位的量化结果。转换之后，结果存放在 ADC10MEM 的相应寄存器中。它的核心使用两个可编程/可选择的参考电平(V_{R+} 和 V_{R-})来定义转换范围的高和低的极限以及定义满范围。当输入的信号等于或大于 V_{R-} 时，数字输出满范围；当输入的信号等于或小于 V_{R-} 时，数字输出为 0。转换控制寄存器定义了输入通道和参考电压水平(V_{R+} 和 V_{R-})，所以转换的公式是

$$N_{ADC} = 1\ 023 \times \frac{Vin - V_{R-}}{V_{R+} - V_{R-}}$$

2. ADC10 的寄存器

在 ADC10 模块中使用了很多寄存器，其中有转换控制寄存器，也有存放转换结果的寄存器，如图 3.100 所示(MSP430F1132)。

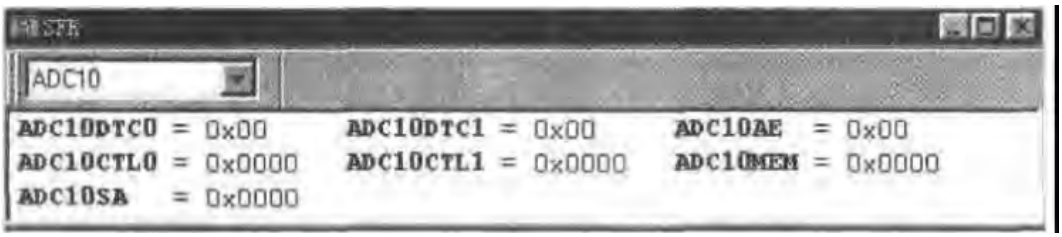


图 3.100　MSP430F1132 中 ADC10 模块的寄存器

(1) ADC10CTL0　控制寄存器 0

ADC10 的所有操作都由 ADC10CTL0 和 ADC10CTL1 两个寄存器的相应位控制，而且大多数位只有在 ENC＝0 时可被修改(下面的阴影部分)，而其他位可在任何时候被修改。ADC10CTL0 中各位及其定义如下：

15～13	12	11	10	9	8	7	6	5	4	3	2	1	0
Sref	ADC10 SHT		ADC10 SR	REF+ OUT	REF BURSH	MSC	2.5V	REF ON	ADC10 ON	ADC10 IE	ADC10 IFG	ENC	ADC10 SC

Sref　参考电源选择位。该 3 位共有 8 种组合，选择参考电源的 6 种可能情况，如下所示：

0　$V_{R+}=AV_{CC}$，　$V_{R-}=AV_{SS}$；

1　$V_{R+}=V_{REF+}$，　$V_{R-}=AV_{SS}$；

2,3　$V_{R+}=Ve_{REF+}$，　$V_{R-}=AV_{SS}$；

4　$V_{R+}=AV_{CC}$，　$V_{R-}=V_{REF-}/Ve_{REF-}$；

5　$V_{R+}=V_{REF+}$，　$V_{R-}=V_{REF-}/Ve_{REF-}$；

6,7　$V_{R+}=Ve_{REF+}$，　$V_{R-}=V_{REF-}/Ve_{REF-}$。

ADC10SHT　对于采样时间，选择 ADC10CLK 的个数。

0　4×ADC10CLKs；

1　8×ADC10CLKs；

2　16×ADC10CLKs；

3　64×ADC10CLKs。

ADC10SR　最大采样速率选择位。

0　最大采样速率可高达 200 ksps；

1　最大采样速率可高达 50 ksps。

REF+OUT　内部参考电压输出与否控制位。

0　不输出；

1　输出。

REFBURSH　内部参考电压在采样转换期间的输出控制位。

0　在采样与转换期间，不输出可节省系统能耗；

1　在采样与转换期间，如果同时 REF+OUT=1，则内部参考电压对外输出。

MSC　多次采样/转换控制位。只有当 ADC10 选择在单通道多次、序列通道及序列通道多次等模式时才有效。

0　每次采样与转换时，由 SHI 的上升沿触发采样定时器；

1　由 SHI 的第一个上升沿触发采样定时器，后面的采样与转换由前一次转换完成后立即执行，而不需要 SHI 的上升沿信号。

2.5 V　内部参考电压值选择位。

0　当参考电压发生器打开时，选择内部 1.5 V 参考电压；

1　当参考电压发生器打开时，选择内部 2.5 V 参考电压。

REFON　内部参考电压发生器控制位。

0　关闭内部参考电压发生器，可以节省能耗；

1　打开内部参考电压发生器。

ADC10ON　ADC10 内核电源控制位。

0　关闭 ADC10 内核的电源，则不能转换；

1　给 ADC10 内核提供能量，可以转换。

ADC10IE　ADC10 中断使能信号。

1　可以中断；

0　不可以中断。

当然有个前提：总中断被设置(GIE=1)，有中断请求(ADC10IFG=1)。

ADC10IFG　ADC10 转换中断标志。如果 ADC10MEM 已经放满转换结果，则该位被置位。当中断服务已经开始，则该位可自动复位，也可软件清零。

ENC　转换使能位。只有在 ENC=1 时才可以通过软件或外部信号启动转换，而且在 ADC10CTL0 和 ADC10CTL1 中，大多数控制位也只有在 ENC=0 时才可以改变。

0　初始状态，没有开始转换。

1　在 SHI 的第一个上升沿开始第一次采样和转换，只要 ENC=1，所选操作将开始。CONSEQ=0，ADC10BUSY=1，ENC=1→0：在这种模式下，如果 ENC 复位，则当前的转换立即停止，转换结果不可信。

CONSEQ≠0，ADC10BUSY=x，ENC=1→0：

在这种模式下，如果 ENC 复位，则当前的转换将完成，转换结果是正确的。在当前转换完成后停止。

ADC10SC　采样/转换控制位。该位被用于转换控制，当 ENC=1 时，可被修改，最好 ISSH 也为低电平。ISSH=0 或 ISSH=1，则该位将启动采样/转换操作。当 ADC10 完成转换(BUSY=0)时该位将被自动复位。

(2) ADC10CTL1　控制寄存器 1

ADC10CTL1 和 ADC10CTL0 一起控制 ADC10 的相关操作，同时大多数位只有在 ENC=0时可被修改(下面的阴影部分)，而其他位可在任何时候被修改。ADC10CTL0 中各位及其定义如下：

15～12	11～10	9	8	7,6,5	4,3	2,1	0
INCH	SHS	ADC10DF	ISSH	ADC10DIV	ADC10SSEL	CONSEQ	ADC10BUSY

INCH　模拟通道选择位。模拟信号通道号为该 4 位二进制数的值。

SHS　采样输入信号源选择控制位。

0　选择 ADC10SC 位；

1　ADC10I1；

2　ADC10I2；

3　ADC10I3。

ADC10DF　在 ADC10MEM 中数据格式为二进制格式或二的补码格式。

0　二进制格式。在此格式下，数据 0 表示输入电压等于参考电压负，数据 03FFH 表示输入电压等于参考电压正。

1　二的补码格式。在此格式下，数据 200H 表示输入电压等于参考电压负，数据 01FFH 表示输入电压等于参考电压正。

ISSH 采样输入信号反向控制位。

0 采样输入信号不反向；

1 采样输入信号反向。

ADC10DIV 进入 ADC10 的时钟信号的分频因子选择位。分频因子就是该 3 位二进制数的值加 1。由分频器分频后，最终输出信号 ADC10CLK。对于一次转换需要 11 个 ADC10CLK 信号。

ADC10SSEL 转换内核时钟源选择控制位。该两位用于选择 ADC10 的采样与转换时钟。

0 选择 ADC10 内部振荡器提供的信号——ADC10OSC；

1 ACLK；

2 MCLK；

3 SMCLK。

CONSEQ 转换模式选择位。共有 4 种转换模式可供选择。

0 单通道单次模式；

1 序列通道单次模式；

2 单通道多次模式；

3 序列通道多次模式。

ADC10BUSY 忙标志位。该标志位指示一次正在进行的采样或转换操作，它只用在单通道单次转换模式。因为在此模式下，ENC 位复位，则转换立即停止，转换结果无效，所以该位的测试应在 ENC=0 之前进行。

0 标志没有正在进行的转换；

1 标志有一个正在进行的转换。

(3) ADC10MEM 转换结果寄存器

每当转换结束时，转换结果转存到该寄存器。它为 16 位寄存器，但只用 10 位，有两种数据格式：二进制格式和二的补码格式。二进制格式如下：高 6 位为 0，低 10 位为有效转换结果。

15～10	9	8	7	6	5	4	3	2	1	0
0	MSB									LSB

二的补码数据格式如下：高 10 位为有效数据，低 6 位为 0。

15	14	13	12	11	10	9	8	7	6	5～0
MSB									LSM	0

(4) ADC10AE 模拟信号输入使能控制寄存器

该寄存器共有 8 位，这些控制位将是下面两个操作的原因：

- 一个模拟信号被使用，则停止任何输入缓存的操作；
- 防止任何来自通用 I/O 口线的反馈到模拟信号源。

当使用 ADC10 做精确的模拟转换时，相应的 ADC10AE.x 位须设置为高电平。该寄存

器的各位如下：

7	6	5	4	3	2	1	0
ADC10 AE.7	ADC10 AE.6	ADC10 AE.5	ADC10 AE.4	ADC10 AE.3	ADC10 AE.2	ADC10 AE.1	ADC10 AE.0

0 相应模拟输入无效；

1 相应模拟输入使能。

(5) ADC10SA 数据传递的开始地址寄存器

该寄存器的内容为ADC10的DTC功能开始地址。因为ADC10的转换结果是字数据，所以这个地址必须是偶地址，且最低位必须为0：

15～1	0
数据传递开始地址	0

(6) ADC10DTC0 数据传递控制寄存器0

该寄存器包含了传递模式等控制位，其各位定义如下：

7～4	3	2	1	0
保 留	ADC10TB	ADC10CT	ADC10B1	ADC10Fetch

ADC10TB DTC写n字到一或二缓存块。

0 一块传递模式。地址变化范围是：SA至SA+2n－2。

1 两块传递模式。地址变化范围是：

第一块：SA至SA+2n－2；

第二块：SA+2n至SA+4n－2。

ADC10CT DTC连续传递使能控制位。

0 在一块或两块传递模式下，一块或两块都传递之后，数据传递将结束。

1 数据传递将继续。只有当ADC10CT=0，或有任何数据写到开始地址寄存器ADC10SA时，DTC将停止。注意，如果在一块传递过程中，ADC10CT=0，则该过程将成功地完成。

ADC10B1 转存块满标志。当存储块1与存储块2被转换数据填满时，该标志位被置位。初始状态为0。该位只有当ADC10IFG被设置之后才有效。写转存开始地址寄存器将复位ADC10IFG标志。

ADC10Fetch 该位仅为了调试目的。

0 该功能无效；

1 将延迟数据的传递，直到当前的CPU指令被完成。

(7) ADC10DTC1 数据传递控制寄存器1

该寄存器的内容为一个8位二进制数n，该数据定义了每一个块的传递转换数据数量。

0 没有数据传递被使能。DTC状态机保持在初始状态，回到初始状态或当一个活动的传递操作完成后停止操作。

1 ～ 255　数据传递使能，同时该寄存器的内容所表示的二进制数定义为每一块传递的数量。

3. ADC10 内核、多路输入及采样保持电路

ADC10 内核、多路输入及采样保持电路如图 3.101 所示。

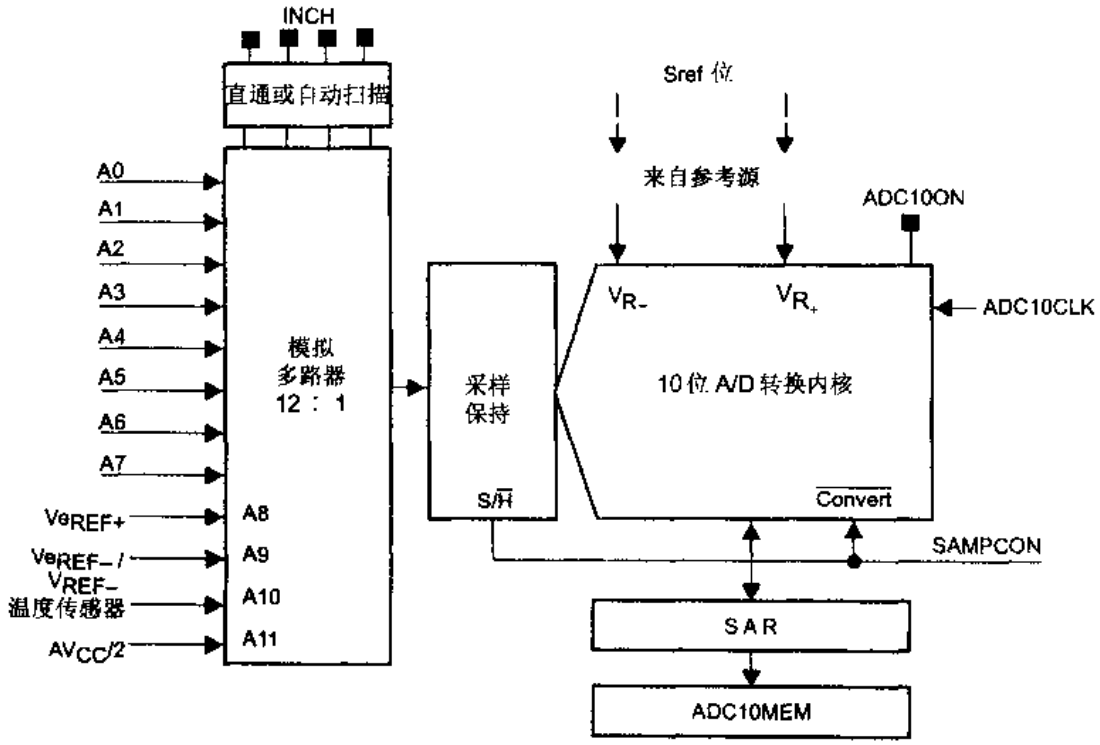


图 3.101　ADC10 模块的模拟输入、内核及采样保持等电路图

输入端实质是一个 12 选 1 的模拟多路器。A0 ～ A7 由外部引脚可输入外部模拟信号；A8～ A11 为内部输入，可测片内参考电压的正、负、电源的 1/2 及片内温度传感器的输出。这12 路模拟输入由 INCH 位选择。

模拟多路器实质是一个 T 型的先关后开型开关（见图 3.102），它可减少因通道切换而引入的噪声。在使用中，为了减少通道间的耦合，须将未选中的通道与 A/D 内核隔离。因为开关两端与开关之间存在寄生耦合电容，这将引起通道间的串扰，可使用屏蔽或 PCB 的布线技术来减少或消除。

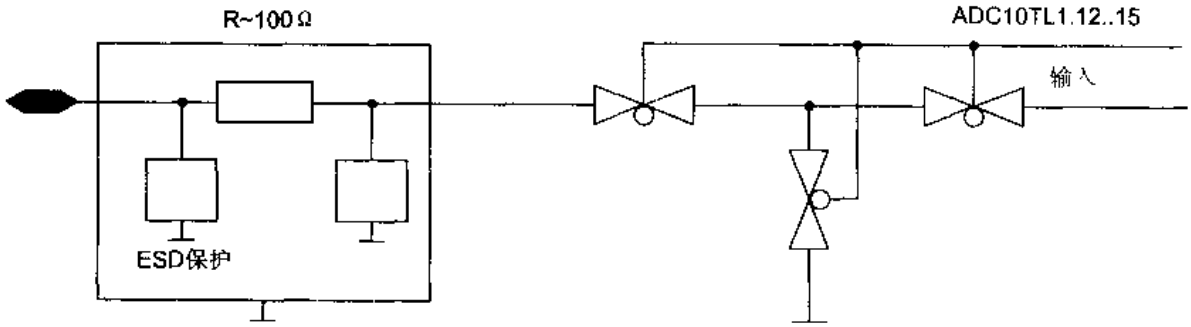


图 3.102　ADC10 模拟开关结构

在采样过程中，模拟输入信号将连接到 A/D 内核的电容阵列，所以电容网络的充电将由模拟信号源提供。电容网络的充电将在采样期间完成。那么模拟信号源的内阻、动态阻抗以及电容网络的容量应该与采样周期相匹配，以保证能达到 10 位转换精度。

另外，信号源的内阻也影响转换的精度。由于漏电流或因为转换开关引起的平均直流电流会使采样信号在引脚出现电压跌落。对于 10 位 A/D，因漏电流引起的误差可按下式计算：

$$\text{误差(LSB)} = 1\,024 \times \text{漏电流}(\mu\text{A}) \times \text{信号源内阻}(\text{k}\Omega)/(V_{R+} - V_{R-})$$

例如，一个 10 kΩ 内阻的信号源工作产生 50 nA 的漏电流，参考电压是 1.5 V，将产生 1/3LSB的电压。

对于信号源内阻引起的误差的讨论，同样也适用于加在 Ve_{REF+} 引脚的外部电压发生器的输出阻抗。输出阻抗应足够低，以保证电压的建立时间小于 0.2/ADC10CLK 和由于漏电流引起的误差远小于 1LSB。采样的定时和采样请参考采样部分。

采样将由 SAMPCON 信号控制。

ADC10 转换内核是一个 10 位精度的模数转换器，同时带有一个转换结果寄存器。它使用两个可编程的参考电压(V_{R+} 和 V_{R-})来定义转换的最大值和最小值。输入模拟电压的最终转换结果是

$$N_{ADC} = 1\,023 \times \frac{Vin - V_{R-}}{V_{R+} - V_{R-}}$$

为了使用芯片的温度传感器，用户只能选择模拟输入的通道 10。选择二极管通道自动打开内部参考电压发生器；但并没有允许 V_{REF-} 的输出，也不确定转换时参考电压的配置；因此，参考电压的操作仍然和其他通道相同。温度传感器的性能请参看芯片手册。

4. 参考电压及其设置

ADC10 的核心包括了一个参考电压发生器，这个参考电压可以选择(1.5 V 和 2.5 V)。它们中的每一个参考电压都可应用于 ADC10 内核，也可以应用于外部的管脚 V_{R+}(检查器件的数据单以确定 V_{REF+} 脚有效)。另外，一个外部的参考电压也可以通过 Ve_{REF+} 提供给 V_{R+}(检查数据单以确定 Ve_{REF+} 脚有效)。

参考电压 V_R 能被选作为 AV_{SS} 或者通过 V_{REF-}/Ve_{REF-} 外部提供(检查器件的数据单以确定 V_{REF-}/Ve_{REF-} 脚有效)。如果 V_{REF-}/Ve_{REF-} 无效，那么 V_{R-} 被连在 AV_{SS} 上。参考电压电路如图 3.103 所示。

参考电压的配置由 ADC10CTL0 寄存器的 Sref 位(13,14 及 15 位)完成。可以有 6 种正负参考电压如表 3.20 所列。

表 3.20　6 种正负参考电压的组合

Sref	V_{R+} 电压	V_{R-} 电压
0	AV_{CC}	AV_{SS}
1	V_{REF+}(内部)	AV_{SS}
2,3	Ve_{REF+}(外部)	AV_{SS}
4	AV_{CC}	V_{REF-}/Ve_{REF-}(内部或外部)
5	V_{REF+}(内部)	V_{REF-}/Ve_{REF-}(内部或外部)
6,7	Ve_{REF+}(外部)	V_{REF-}/Ve_{REF-}(内部或外部)

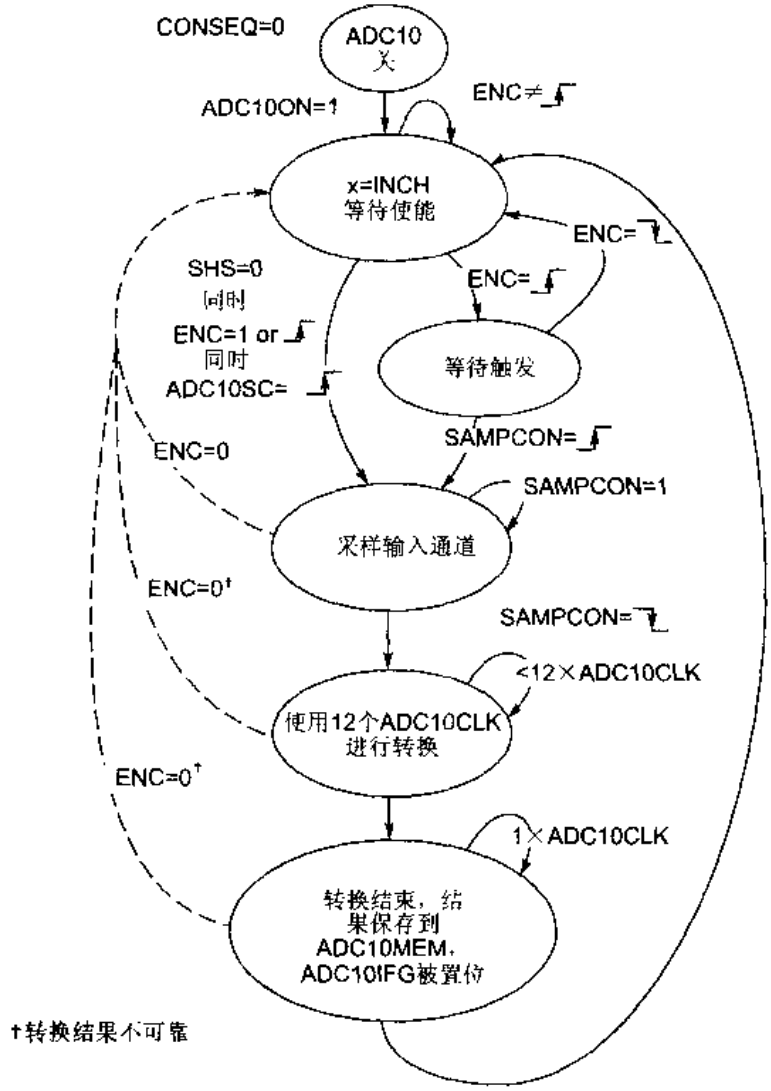


图 3.107 单通道单次转换模式状态图

在转换开始但没有完成转换之前，也可以改变转换模式。新的转换模式将在当前转换完成之后生效。

(2) 序列通道单次转换模式

序列通道单次转换模式对通道从 A(由 INCH 选择的通道)直到 A0 序列做一次转换。当 A0 的转换结果存放在 ADC10MEM 之后转换停止。每次转换完成后，转换结果存放在 ADC10MEM。每次寄存器 ADC10MEM 被装入数据，中断标志 ADC10IFG 位被置位。另外，如果中断允许位 ADC10IE 置位，将产生一次中断请求。

当软件正在用 ADC10SC 位来启动转换时，可简单地通过设置 ADC10SC 位(ENC 位能保持设置或可以与 ADC10SC 同时设置)来启动下一个序列。然而，当任何其他触发源(ADC10I1、ADC10I2 或 ADC10I3)用来启动转换时，在 NEC 复位并再次置位前的输入采样信号将被忽略。

转换模式可以在转换开始后和结束前改变，新模式会在当前序列转换完成后起作用。转换模式部分的切换也是如此。

如果转换模式可以在转换开始但未结束前，且 NEC 位为高时改变转换模式，转换能正常

结束。新的模式在原序列转换完成后生效，但是新模式是单通道单转换的除外。这时，如果原模式未进行的采样及转换或者正在进行采样及转换已完成，则原模式停止。原采样的序列可能未完成，但是已经完成的转换结果是有效的。

如果在序列转换已经开始但未结束前，且 NEC 位已经翻转时改变转换模式，则原序列转换能正常结束。新的模式在原序列转换完后有效，但是新模式是单通道单转换的除外。如果新模式是单通道单次转换，则当没有活跃的采样及转换或活跃的采样及转换完成后，当前的序列通道转换将停止。这时，如果原模式未进行采样及转换或者正在进行的采样及转换已完成，或者 NEC 位复位，则原模式停止。新模式在 NEC 再次置位时有效。

将 CONSEQ.1 复位选择单通道单转换，并且将 NEC 位复位，能使当前序列转换模式立即停止。这时，转换存储寄存器 ADC10MEMx 中的数据不可靠，中断标志 ADC10IFG.x 可能置位也可能不置位。因此这种处理方法应该避免，但在紧急的情况下仍可能用到。

在这种模式下，ADC10 的状态转换图如图 3.108 所示。

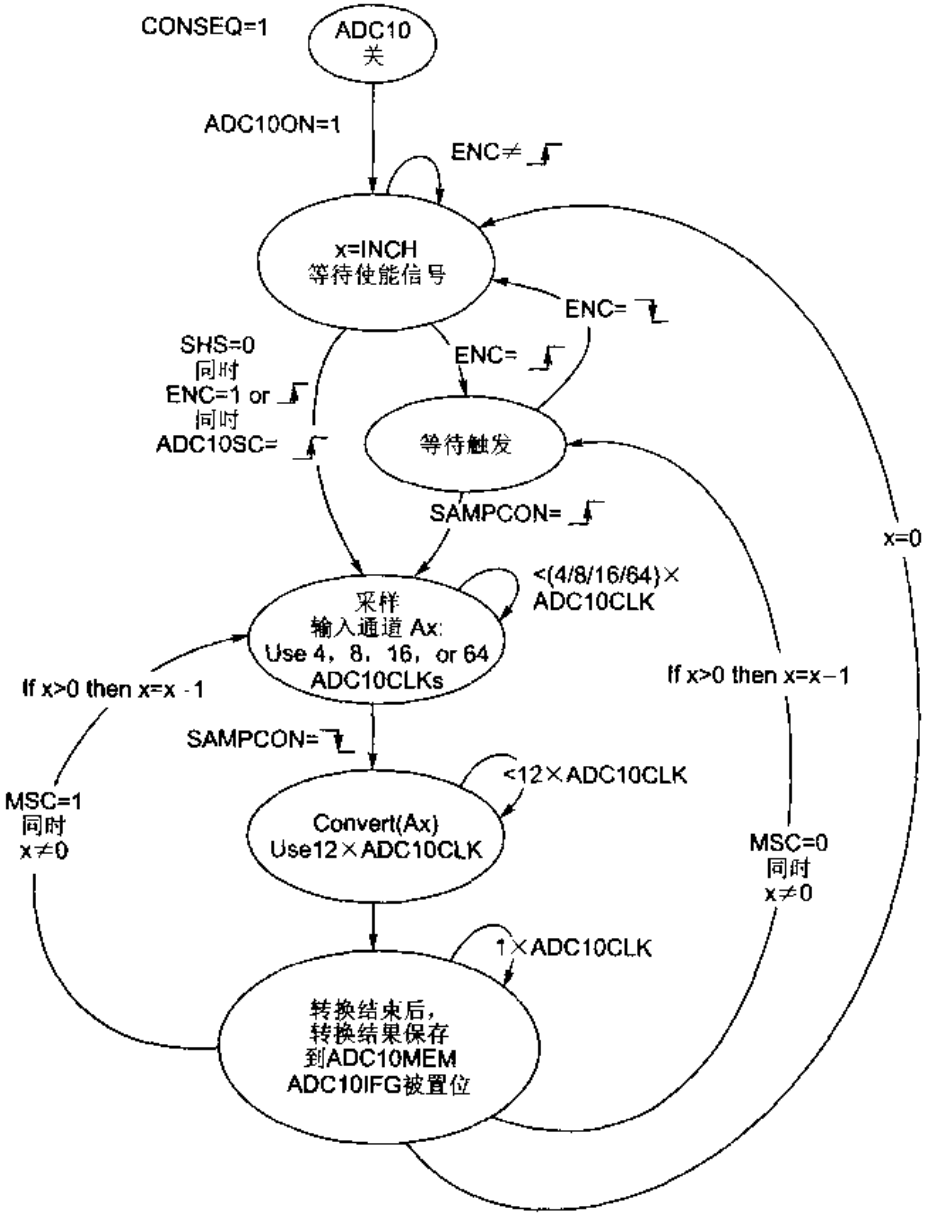


图 3.108　序列单通道转换状态图

(3) 单通道多次转换模式

单通道多次转换模式的状态图如图 3.109 所示。

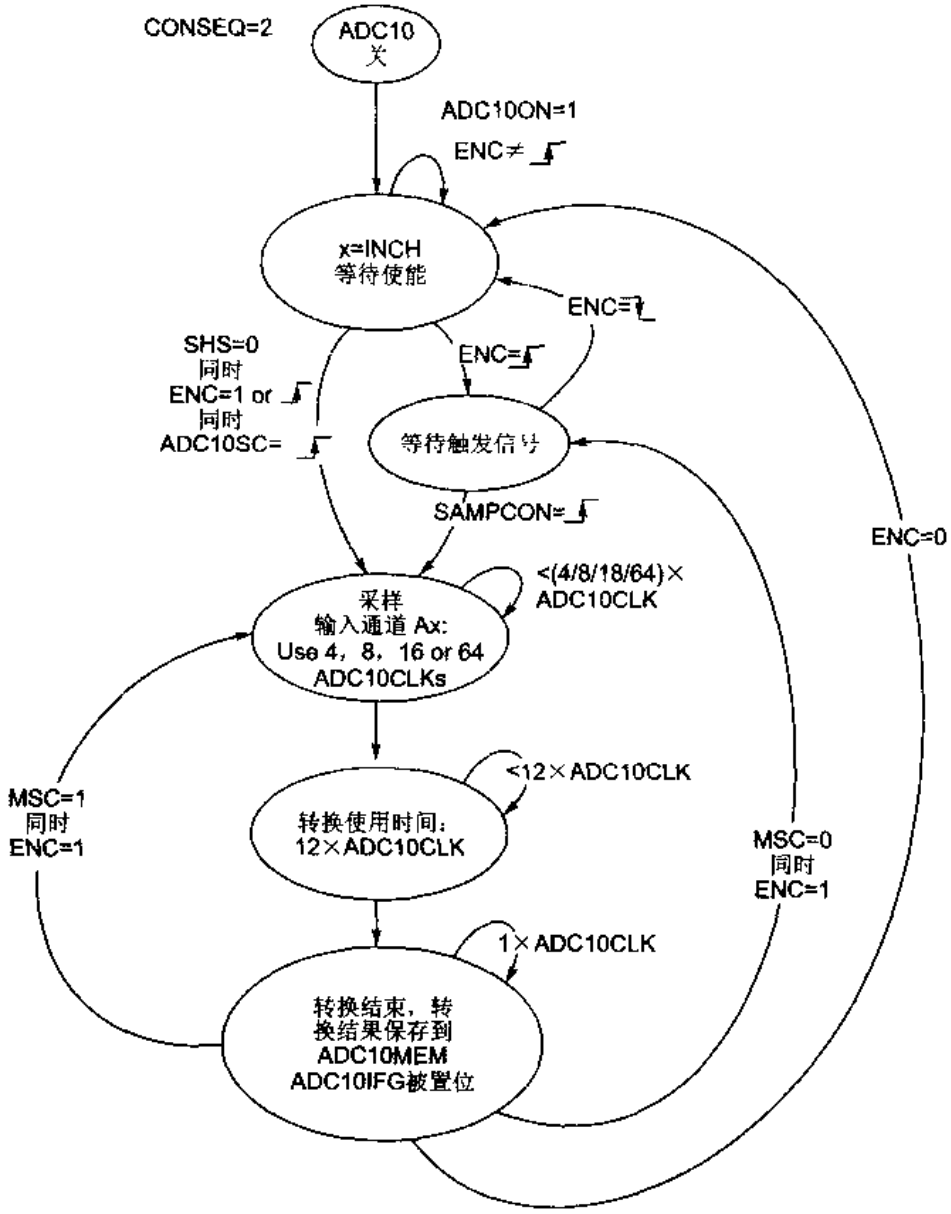


图 3.109 单通道多次转换模式状态图

单通道多次转换模式与单通道单次转换模式类似，只是转换在选定的通道上重复进行，直到用软件将它停止。每次转换结束，转换的结果存在 ADC10MEM，每次 ADC10MEM 被装入数据都将设置中断标志 ADC10IFG。如果这时允许中断，将产生中断请求。

改变转换模式不必先停止转换。一旦模式改变，它必须在当前序列完成后有效。

有 3 种方法来停止单通道多次转换，即：

● 用 CONSEQ 位来选择单通道单次转换模式来代替单通道多次转换模式。当前序列转换能正常结束，转换的结果存入 ADC10MEM，中断标志 ADC10IFG 将置位。

● 复位 NEC 位（ADC10CTL0. 1）则当前序列完成后停止。转换的结果存入

ADC10MEM，中断标志 ADC10IFG 将置位。

● 选择单通道单次转换模式来代替单通道多次转换模式，并且将转换使能位 NEC 位复位，可使当前转换模式立即停止。这时，转换存储寄存器 ADC10MEM 中的数据不可靠，中断标志 ADC10IFG 可能置位也可能不置位。因此这种处理方法应该避免。

(4) 序列通道多次转换模式

序列通道多次转换模式的状态图如图 3.110 所示。

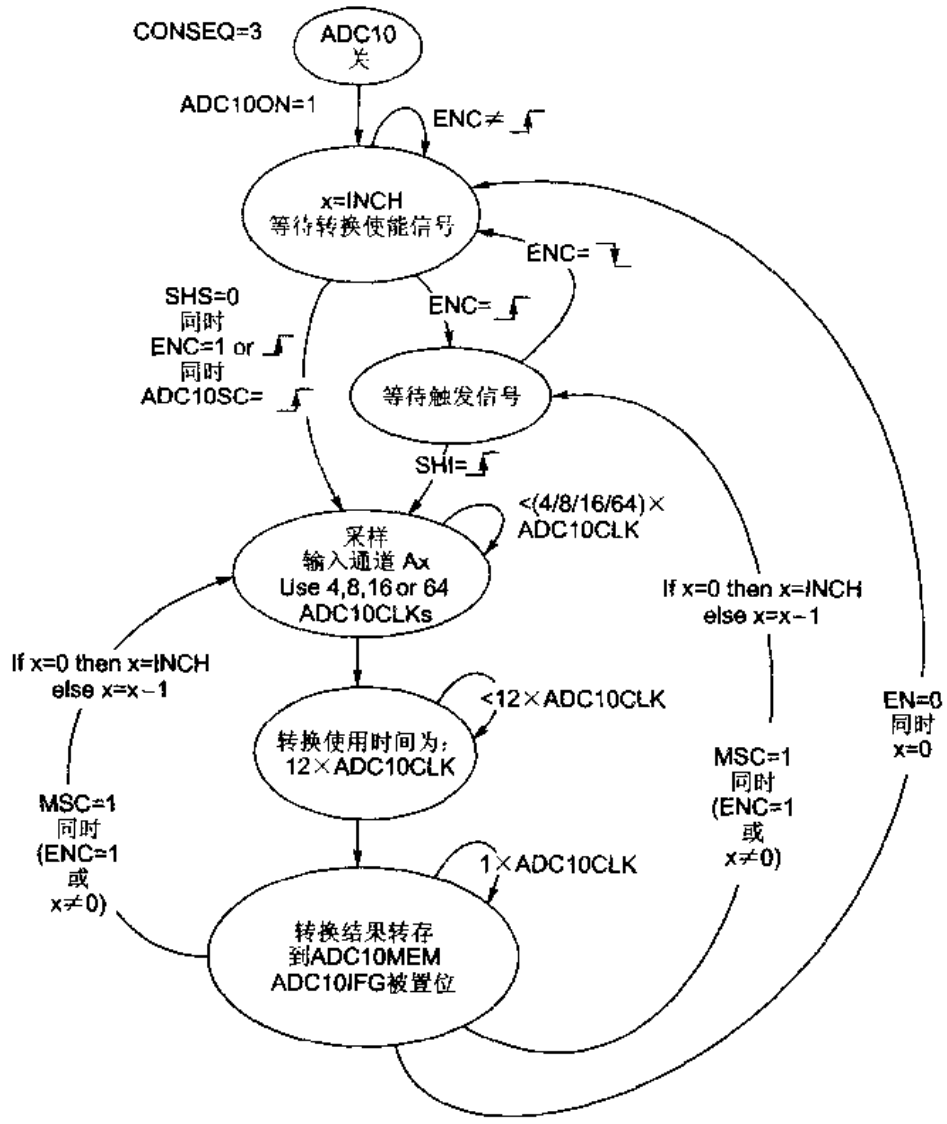


图 3.110　多通道多次转换状态图

序列通道多次转换模式与序列通道单次转换模式类似，只是序列转换重复进行，直到用软件将它停止。每次转换结束，转换的结果存入 ADC10MEM，中断标志 ADC10IFG 将置位指示转换结束。如果这时允许中断，将产生中断请求。

改变转换模式不必先停止转换。一旦模式改变，它必须在当前序列完成后有效。除非新的模式是单通道多次模式，这时，不必等到序列完成，新模式立即成为有效。

有4种方法来停止序列通道多次转换，即：

● 用CONSEQ=1来选择单通道多次转换模式来代替序列通道多次转换模式(CONSEQ=3)。当前序列转换正常结束后不再转换，转换的结果存入ADC10MEM，中断标志ADC10IFG置位。

● 用NEC复位使当前序列完成后停止。转换的结果存入ADC10MEM，中断标志ADC10IFG置位。

● 用CONSEQ=2来选择单通道多次转换模式来代替序列通道多次转换模式；然后选择单通道多次转换模式，则当前序列转换正常结束，转换的结果存入ADC10MEM，中断标志ADC10IFG置位。

● 用CONSEQ=0选择单通道单次转换模式来代替序列通道多次转换模式，并且将NEC位复位，能使当前系列转换模式立即停止。这时，转换存储寄存器ADC10MEM中的数据不可靠，中断标志ADC10IFG可能置位也可能不置位。因此这种处理方法应该避免。

7. 高速转换支持——数据传递控制逻辑以及对它的操作

ADC10模块内含一个数据传递控制逻辑(DTC)。DTC通常被用作自动传递ADC10转换结果到存储器单元(典型的是RAM)。通常，在微控制器应用中，一个转换结束标志或一个中断标志被设置，或一个用于模数转换的中断服务被请求，那么由于有ADC10的DTC硬件，转换结果能自动被传递到被选择的目的单元保存起来。只有当所选的存储单元满时，才会请求软件的服务，进行数据处理。

ADC10的DTC硬件在需要多路转换的数字信号处理应用中特别有用，DTC硬件原理如图3.111所示。

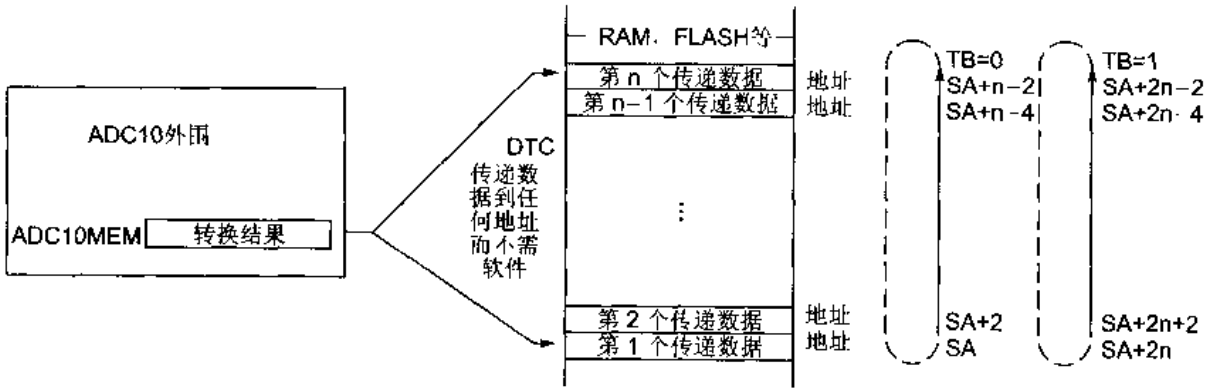


图3.111 DTC原理图

对于具体的应用有的需要DTC，有的不需要DTC，这完全根据需要来确定。

(1) 一块传递模式

对基于DTC的数据传递，由每一次转换结果装入ADC10MEM缓存引起(如果DTC使能并已经初始化)。当DTC被使用时，每一次ADC10MEM缓存数据更新将不引起ADC10IFG标志位的置位，而是在转换结果缓存块被填满时ADC10IFG置位。同时DTC支持一个或两个块的传递操作。

当DTC被使能，n≠0，ADC10TB=0，是一个块传递，则传递模式如图3.112所示。在这种模式下，数据传递的开始地址(目的)是ADC10SA寄存器中的值，而结束地址为

SA+2n-2，其中，SA为ADC10SA中的值，n是ADC10DTC1寄存器中的值，表示一个转换缓存数据块的长度。

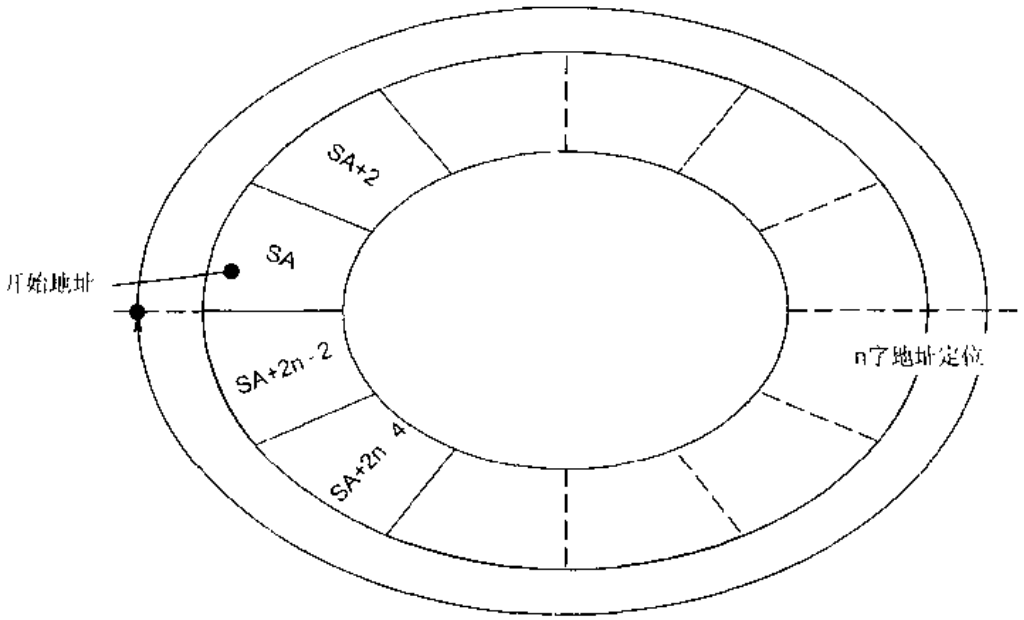


图3.112 一块传递模式

ADC10MEM第一次被装入数据启动转换数据传递操作。DTC传递ADC10MEM中的数据到由ADC10SA这个指针所指的字地址中。下一个数据将放在指针加2的地址中。而DTC内部有一指针，初始为n，每传递一次数据减1。当该指针减为0时，操作完成，最后一个数据也被传递完，ADC10IFG被置位。如果中断条件满足，将发生中断服务。

(2) 两块传递模式

当DTC被使能，如果n≠0，ADC10TB=1，则是两块传递模式。在这种模式下，地址变化范围是：第一块为SA～SA+2n-2；第二块为SA+2n～SA+4n-2。其中SA是ADC10SA寄存器中的值，n是ADC10DTC1寄存器中的值。数据传递模式可以用图3.113来说明。

与一块传递模式一样，整个操作由ADC10MEM的第一次装载(第一次ADC10转换完成，将转换结果存放在ADC10MEM中)启动。DTC传递数据的地址为SA指针所指的地址，地址指针每次增2；而内部指针初始化为n，每次减1。重复操作(将ADC10MEM中的数据传递到块)直到内部指针为0。这是第一块被转换数据填满，ADC10IFG将被置位，如果条件满足，可以引起中断；同时ADC10B1位被置位，指示第一块已满。紧接着，内部传递指针再次被使用，与第一块相同，进行第二块数据的传递。当两块都满时，ADC10IFG被置位，同时ADC10B1位被清除。ADC10B1=0指示第二块也填满了数据。

8. ADC10应用举例

下面的示例为将MSP430F1232片内温度传感器所表示的温度值换算为摄氏或华氏表示法。先将片内温度传感器所表示温度的模拟量转换为数字表示。

对于模数转换部分：选择A10为模拟输入通道；选择片内参考电压1.5 V。

对于转换结果部分：摄氏温度值放在200H字单元，BCD表示；华氏温度值放在202H字单元，BCD表示。

出口参数：R12　0000 -- 091h

```
Trans2TempC   mov.w   &ADC10MEM,R12          ;
              mov.w   #423,R11               ; C
              call    #MPYU                  ;
              bic.w   #00FFh,R14             ; 1024
              add.w   R15,R14                ;
              swpb    R14                    ;
              rra.w   R14                    ;
              rra.w   R14                    ;
              mov.w   R14,R12                ;
              sub.w   #278,R12               ; C
              ret                            ;
                                             ;
```

转换为华氏温度的方法如下所示：

R12 = ADC10MEM/1024 * 761 − 468

oF = ((x/1024) * 1500mV) - 923mV) * 1/1.97mV = x * 761/1021 - 468

入口参数：ADC10MEM　0000 — 0FFFh

出口参数：R12　0000 - 0124h

```
Trans2TempF   mov.w   &ADC10MEM,R12          ;
              mov.w   #761,R11               ; F
              call    #MPYU                  ;
              bic.w   #00FFh,R14             ; 1024
              add.w   R15,R14                ;
              swpb    R14                    ;
              rra.w   R14                    ;
              rra.w   R14                    ;
              mov.w   R14,R12                ;
              sub.w   #468,R12               ; F
              ret                            ;
```

将两位二进制数转换为 BCD 码程序如下：

```
BIN2BCD4      入口参数：R12　0000 — 0FFFh
;             出口参数：R13　0000 -- 4095
              mov.w   #16,R15                ;循环计数器
              clr.w   R13                    ;结果先为 0
BIN1          rla.w   R12                    ;移位
              dadd.w  R13,R13                ;
              dec.w   R15                    ;
              jnz     BIN1                   ;
              ret                            ;
```

在运算过程中将用到乘法，可使用下面的子程序(也可参照第 5 章)：

```
MPYU    ;   功能：R11 x R12 = R15 R14
        ;   入口参数：R11，R12
        ;   出口参数：R15，R14
            clr.w   R14                     ;
            clr.w   R15                     ;结果寄存器清 0
MACU        clr.w   R13                     ;工作寄存器清 0
            mov.w   #1,R10                  ;
MPY2        bit.w   R10,R11                 ;测试位
            jz      MPY1                    ;如果 0：继续测试
            add.w   R12,R14                 ;如果 1：增加结果值
            addc.w  R13,R15                 ;
MPY1        rla.w   R12                     ;
            rlc.w   R13                     ;
            rla.w   R10                     ;准备测试下一位
            jnc     MPY2                    ;如果进位位为 1，则退出
            ret                             ;
ADC10_ISR;  mov.w   #GIE,0(SP)              ;中断可以退出任何低功耗状态
            reti                            ;
            ORG     0FFFEh                  ; MSP430 复位
            DW      RESET                   ;
            ORG     0FFEAh                  ; ADC10 中断
            DW      ADC10_ISR               ;
            END
```

3.8.2 ADC12 模数转换模块

ADC12 模块能够实现 12 位精度的模数转换，具有高速和通用的特点。该模块存在于 MSP430F13X，MSP430F14X，MSP430F43X 和 MSP430F44X 等系列器件中。目前其他系列没有 ADC12 模块。ADC12 的结构原理如图 3.114 所示。

1. ADC12 结构及特点概述

由图 3.114 可以看出，ADC12 由 5 大功能模块构成，而且都可通过用户软件独立配置。

- ADC12 的内核是一个带有采样与保持功能的 12 位转换器；
- 内部参考电压发生器，同时有两种参考电压值可供选择；
- 采样与转换过程中所需要的时钟信号源可以选择；
- 采样及转换所需的时序控制电路；
- 转换结果有专门的桶型缓存。

ADC12 的主要特点归纳为如下：

- 采样速度快，最高可达 200 ksps；
- 12 位转换精度，1 位非线性微分误差，1 位非线性积分误差；
- 内置采样与保持电路；
- 有多种时钟源可提供给 ADC12 模块，而且模块本身内置时钟发生器；
- 内置温度传感器；

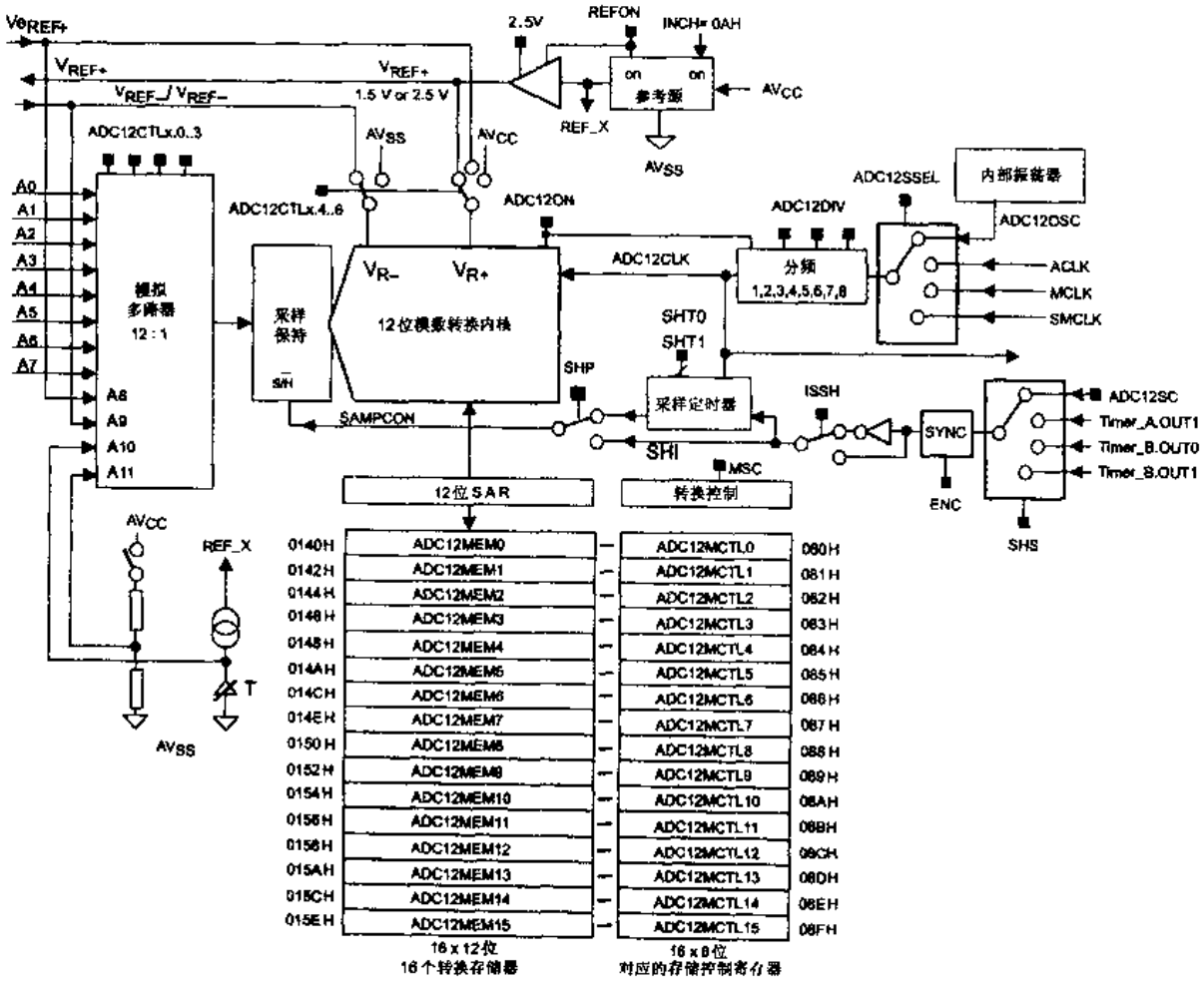


图 3.114　ADC12 电路图

- 配置有 8 路外部通道与 4 路内部通道；
- 内置参考电源，而且参考电压有 6 种可编程的组合；
- 模数转换有 4 种模式，可灵活地运用以节省软件量及时间；
- ADC12 内核可关断以节省系统能耗。

2. ADC12 的寄存器

可以看出，ADC12 使用起来相当灵活方便，对它的操作使用相关的控制寄存器实现。该模块的寄存器很多，大致可分为 4 类：转换控制类、中断控制类、存储控制类及存储器类。图 3.115所示为 ADC12 模块相关的所有寄存器。

其中，转换控制类为：ADC12CTL0，DC12CTL1；

中断控制类为：ADC12IFG、ADC12IE、ADC12IV；

存储控制类为：ADC12MCTL0～ADC12MCTL15；

存储器类为：ADC12MEM0～ADC12MEM15。

(1) ADC12CTL0　控制寄存器 0

该寄存器中的位与 ADC12CTL1 中的位一起控制了 ADC12 的大部分操作，而且其中的大多数位(下面的阴影部分)只有在 ENC=0 时才可被修改。该寄存器的各位定义如下：

0　选择内部 1.5 V 参考电压；

1　选择内部 2.5 V 参考电压。

REFON　参考电压控制位。

0　内部参考电压发生器关闭，这样也可节省能耗；

1　内部参考电压发生器打开，将消耗能量。

ADC12ON　ADC12 内核控制位。

0　关闭 ADC12 内核，则不消耗能量，也不能进行转换；

1　打开 ADC12，可以进行转换。

ADC12OVIE　溢出中断允许位。当 ADC12MEMx 中原有数据还没有读出，而又有新的转换结果数据要写入时，则发生溢出。如果相应的中断允许，则会发生中断请求。

0　没有发生溢出；

1　发生溢出。

ADC12TVIE　转换时间溢出中断允许位。在当前转换还没有完成时，又发生一次采样请求，则会发生时间溢出。如果这时允许中断，则会发生中断请求。

0　没有发生转换时间溢出；

1　发生转换时间溢出。

ENC　转换允许位。只有在该位为高电平时，才能用软件或外部信号启动转换。而且控制寄存器 ADC12CTL1 和 ADC12CTL0 中有很多位只在该位为低电平时才可修改。

0　ADC12 模块处于初始状态，不能启动 A/D 转换；

1　首次转换由 SAMPCON 的上升沿启动，在 ENC 为高电平期间操作有效。

当 CONSEQ=0 且 ADC12BUSY=1 时，ENC 由高电平变为低电平，则当前操作立即停止，转换结果不可靠；

当 CONSEQ≠0 时，ENC 由高电平变为低电平，则当前转换正常结束，且转换结果有效，在当前转换结束时停止操作。

ADC12SC　采样/转换控制位。当 ENC=1 时，可用软件修改作为转换控制。建议将 ISSH 位设置为低电平。

如果采样信号 SAMPCON 由采样定时器产生(SHP=1)，将 ADC12SC 由 0 改为 1 将启动转换操作。当 A/D 转换完成后 ADC12SC 将自动复位。

如果采样直接由 ADC12SC 控制(SHP=0)，则 ADC12SC 保持为高电平时采样。当 ADC12SC 复位时启动一次转换。

在使用软件控制 ADC12SC 时，必须满足应用中的时序要求。

用软件启动一次转换，可使用一条指令来完成 ADC12SC 与 ENC 的置位。

(2) ADC12CTL1　控制寄存器 1

该寄存器中的位与 ADC12CTL0 中的位一起控制了 ADC12 的大部分操作，而且其中的大多数位(下面的阴影部分)只有在 ENC=0 时才可被修改。该寄存器的各位定义如下：

15～12	11～10	9	8	7～5	4,3	2,1	0
CSStartAdd	SHS	SHP	ISSH	ADC12 DIV	ADC12 SSEL	CONSEQ	ADC12 BUSY

CSStartAdd　转换存储器地址定义位。该4位定义了在ADC12MEMx中作为单次转换地址或序列转换的首地址。该4位所表示的二进制数0～15分别对应ADC12MEM0～15。由于每一个转换存储器有一个对应的转换存储器控制寄存器，所以也同时定义了ADC12MCTLx。

SHS　采样输入信号源选择控制位。

0　ADC12SC；

1　Timer_A. OUT1；

2　Timer_B. OUT0；

3　Timer_B. OUT1。

SHP　采样信号(SAMPCON)选择控制位。

0　SAMPCON直接源自采样输入信号；

1　SAMPCON直接源自采样定时器，由采样输入信号的上升沿触发采样定时器。

ISSH　采样输入信号反向与否控制位。

0　采样输入信号为同向输入；

1　采样输入信号为反向输入。

ADC12DIV　ADC12时钟源分频因子选择位。该3位选择分频因子，则分频因子为该3位二进制数加1。

ADC12SSEL　ADC12内核时钟源选择。

0　ADC12内部时钟源：ADC12OSC；

1　ACLK；

2　MCLK；

3　SMCLK。

CONSEQ　转换模式选择位。ADC12模块提供4种转换模式，将在转换模式部分详细讲述。

0　单通道单次转换模式；

1　序列通道单次转换模式；

2　单通道多次转换模式；

3　序列通道多次转换模式。

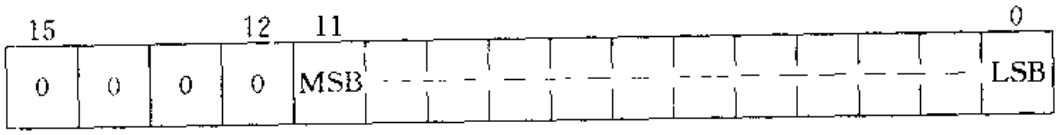
ADC12BUSY　ADC12忙标志位。它专用于单通道单次转换模式，在此模式下，如果ENC复位，则转换将立即停止，因而转换结果无效，所以需要在使ENC=0之前，测试ADC12BUSY位以确定是否为0。而在其他模式下，该位无效，因为复位ENC不会立即生效。

0　表示没有活动的操作；

1　表示ADC12正处于采样期间、转换期间或序列转换期间。

(3) ADC12MEM0～ADC12MEM15　转换存储寄存器

ADC12 共有 12 个转换通道，这里设置了 16 个转换存储器用于暂存转换的结果。当设置好之后，ADC12 硬件会自动将转换结果存放到相应的 ADC12MEM 寄存器中。这 16 个寄存器均为 16 位寄存器，但只用其中低 12 位，高 4 位在读出时为 0，如下所示：



(4) ADC12MCTLx　转换存储器控制寄存器

对应于 16 个转换存储器有 16 个转换存储器控制寄存器 ADC12MCTL0～ADC12MCTL15，每个转换存储器 ADC12MEMx 都有自己的控制寄存器 ADC12MCTLx。控制寄存器控制各个转换存储器须选择基本的转换条件，如：模拟信号通道、参考电压源及指示采样序列的结束等。

该寄存器为 8 位寄存器，位于字节地址中。其中的各位同样只有在 ENC 为低电平时可修改。在 POR 时，各位被复位。该寄存器的各位定义如下：

7	6,5,4	3,2,1,0
EOS	Sref　参考电压源	INCH　输入通道

EOS　序列结束控制位。

0　序列没有结束；

1　表示为该序列中最后一次转换。

Sref　参考电压源选择位。共有 6 种情况可供选择，分别为 V_{R+} 与 V_{R-} 的组合。

0　$V_{R-}=AV_{CC}$，　$V_{R-}=AV_{SS}$；

1　$V_{R+}=V_{REF+}$，　$V_{R-}=AV_{SS}$；

2,3　$V_{R+}=Ve_{REF+}$，　$V_{R-}=AV_{SS}$；

4　$V_{R+}=AV_{CC}$，　$V_{R}=V_{REF}/Ve_{REF-}$；

5　$V_{R+}=V_{REF+}$，　$V_{R}=V_{REF-}/Ve_{REF-}$；

6,7　$V_{R+}=Ve_{REF+}$，　$V_{R-}=V_{REF-}/Ve_{REF-}$。

INCH　选择模拟输入通道。该 4 位所表示的二进制数为所选的模拟输入通道。

0～7　A0～A7；

8　Ve_{REF+}；

9　V_{REF-}/Ve_{REF-}；

10　片内温度传感器的输出；

11～15　$(AV_{CC}-AV_{SS})/2$。

(5) ADC12IFG　中断标志寄存器

有 16 个中断标志位 ADC12IFG.x，对应于 16 个转换存储寄存器 ADC12MEMx。它是一个 16 位字结构，位于字地址，如下所示：

15	14	…	1	0
ADC12 IFG.15	ADC12 IFG.14	…	ADC12 IFG.1	ADC12 IFG.0

中断标志位 ADC12IFG. x 在转换结束后，转换结果装入转换存储寄存器 ADC12MEMx 时置位；在 ADC12MEMx 被访问时复位；而在访问中断向量字 ADC12IV 时不复位，用以保证能处理发生溢出的情况。因为如果在 ADC12IFG. x 未复位时（数据没有被读走）有转换数据写入 ADC12MEMx 时，会发生溢出。

（6）ADC12IE　中断允许寄存器

与 ADC12IFG 寄存器一样对应于 16 个转换存储寄存器 ADC12MEMx。该寄存器的各位将允许 1 或禁止 0 相应的中断标志位 ADC12IFG. x 在置位时发生的中断请求服务。它也是一个 16 位寄存器，位于字地址空间。

（7）ADC12IV　中断向量寄存器

ADC12 有 18 个中断标志（ADC12IFG. x 与 ADC12TOV，ADC12OV），但只有一个中断向量。18 个中断标志共用一个中断向量，并按照优先级来安排中断标志的响应，即 18 个标志有一个优先级顺序，高优先级的请求可以中断正在服务的低优先级。表3. 21是它们的优先级顺序与对应的中断向量值。

表 3. 21　ADC12 各中断标志对应的 ADC12IV 值

ADC12 中断标志 ADC12IFG															ADC12TOV	ADC12OV	AIX12IV
15	14	13	12	11	10	9	8	7	6	4	3	2	1	0			
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	1	2
×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	1	0	4
×	×	×	×	×	×	×	×	×	×	×	×	×	×	1	0	0	6
×	×	×	×	×	×	×	×	×	×	×	×	×	×	0	0	0	8
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
×	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	34
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	36

优先级最高的是溢出标志 ADC12OVIFG，然后是时间溢出中断标志 ADC12TOVIFG，最后才是对应于转换存储器的标志 ADC12IFG. x。而在 16 个对应于转换存储器的标志中 ADC12IFG. 0 最高，ADC12IFG. 15 最低。各中断标志将会产生一个 0～36 的偶数，0 标志没有中断或没有中断标志发生置位；其他数字（2～36）对应于各中断标志位。位于 ADC12IV 中的数字将加在 PC（程序计数器）上，用于实现自动进入相应中断服务程序。只有相应的中断允许位以及总的中断允许位设置之后，才可能响应中断请求，而发生中断服务。

对于中断标志，ADC12OVIFG 和 ADC12TOVIFG 会在访问 ADC12IV 后自动复零。但在响应了 ADC12IFG. x 标志对应的中断服务之后，相应的标志不自动复零，需在用户软件中复零，或通过访问对应转换存储器 ADC12MEMx 的标志位自动复零。

同时，MSP430 没有提供 ADC12OVIFG 和 ADC12TOVIFG 两标志位的软件访问。要想知道两标志的情况，可通过访问 ADC12IV 来间接实现。

3. 转换模式

ADC12 提供 4 种转换模式，以方便设计者使用：

- 单通道单次转换；
- 序列通道单次转换；
- 单通道多次转换；
- 序列通道多次转换。

表3.22对4种转换模式作了简要说明。

表3.22 ADC12的4中转换模式

转换模式	CONSEQ	操作说明
单通道单次	00	对选定的通道进行单次转换；x=CSStartAdd，指向转换开始地址；转换结果存放在ADC12MEMx，对应的中断标志ADC12IFG.x。在ADC12MCTLx寄存器中定义了通道和参考电压
序列通道单次	01	对序列通道进行单次转换；x=CSStartAdd，指示转换开始通道，而序列中最后通道（y）使用EOS(ADC12MCTLx.7)=1标志，非最后通道的EOS(ADC12MCTLx，ADC12MCTL(x+1)，…，ADC12MCTL(y-1))都是0，表示序列没有结束；结果存放在ADC12MEMx，…，ADC12MEMy；中断标志为ADC12IFG.x，…，ADC12IFG.y。在ADC12MCTLx寄存器中定义了通道和参考电压
单通道多次	10	对选定的通道作多次转换，直到关闭该功能或ENC=0；x=CSStartAdd，指示转换开始通道；结果存放在ADC12MEMx。在ADC12MCTLx寄存器中定义了通道和参考电压
序列通道多次	11	对序列通道作多次转换，直到关闭该功能或ENC=0；x=CSStartAdd，指示转换开始通道，而序列中最后通道（y）使用EOS(ADC12MCTLx.7)=1标志，非最后通道的EOS(ADC12MCTLx，ADC12MCTL(x+1)，…，ADC12MCTL(y-1))都是0，表示序列没有结束。在ADC12MCTLx寄存器中定义了通道和参考电压

（1）单通道单次转换模式

该模式实现对单一通道的一次采样与转换。在转换存储控制寄存器ADC12MCTLx中定义了采样转换通道以及转换范围，同时指定了用于保存转换结果的转换存储寄存器。如果ENC=0，则可使转换立即停止。这时的转换结果不可靠或转换并没有执行。图3.116可说明这一过程。

图3.116表明了在各种情况下，使用ENC=0来停止转换时，ADC12模块的采样和转换情况。而真正的有效采样和转换是在ENC上升沿之后，保持为1的情况下完成的。关于ADC12模块在各种条件下，自身所处的状态，可以参见图3.117。

当转换正常结束时，转换结果写入选定的存储寄存器ADC12MEMx，相应的中断标志位ADC12IFG.x置位。如果这时允许中断，则可产生中断服务请求。在存储寄存器ADC12MEMx中的值被访问之后，中断标志位ADC12IFG.x复位。

转换完成时，必须使ENC再次复位并置位（上升沿），以准备下一次转换。在ENC复位并再次置位之前的输入信号将被忽略。转换模式可以在转换开始但结束之前切换，新模式会在当前转换完成之后起作用。

当用户软件使用ADC12SC位启动转换时，下一次转换可以通过简单地设置ADC12SC位（ENC保持为高，或能在设置ADC12SC位的同时置位ENC）来启动。然而，当有其他任何触

发源用于开始转换，ENC 位必须在每次转换之间固定(be toggled)。其他的采样输入信号将在 ENC 复位并置位之前被忽略。

在此模式下，对于转换存储器的设置见图 3.118 的示例。其含义为：

- 对通道 4(A4)作单次转换(ADC12MCTL1 的 INCH=4)；
- 转换范围为：V_{R+} 使用 AV_{CC}，同时 V_{R-} 使用 AV_{SS}(Sref=0)；
- 转换结果存储在 ADC12MEM1 中(请设置 ADC12CTL1 的 CSStartAdd=1)。

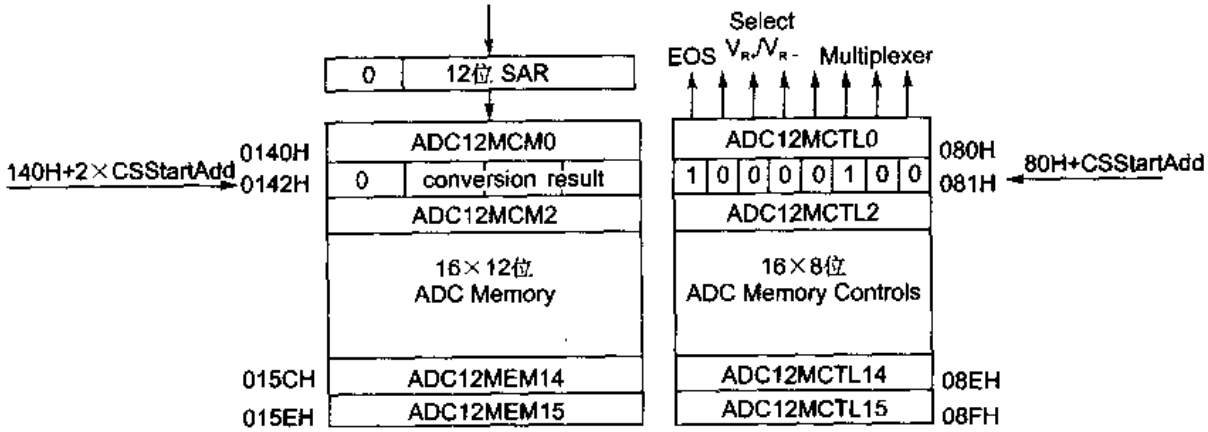


图 3.118　转换存储设置

(2) 序列通道单次转换模式

该模式对一个序列通道(有顺序的多个通道)作一次转换。ADC12CTL1 中的 CSStartAdd 位指向第一个转换存储寄存器。其后的转换结果将顺序地在转换存储寄存器中存放。如要转换的序列为 3，而 CSStartAdd 指向转换存储器 4，则第一个转换结果存放在 ADC12MEM4，第二个转换结果存放在 ADC12MEM5，第三个转换结果存放在 ADC12MEM6。

序列通道转换与单通道转换不同，单通道由于只有一个通道，所以转换所需参数(通道号与参考电压等)在一个转换存储控制寄存器中。而序列通道有多个通道，每一个通道的转换参数由相应的存储控制寄存器设置。如上面的例子，3 个通道分别使用 ADC12MEM4，ADC12MEM5 及 ADC12MEM6，则相应通道的转换参数在 ADC12MCTL4，ADC12MCTL5 及 ADC12MCTL6 中设置。同时 ADC12MCTLx 寄存器还将设置什么时候该序列转换结束，如上面的例子，如果在转换结果存放在 ADC12MEM6 时结束，则 ADC12MCTL6 的 EOS 位为 1，而其余的 EOS 为 0。更详细的情况如图 3.119 和图 3.120 所示。

图 3.119 很清楚地说明了序列通道单次转换的参数设置流程。要转换的序列通道是 A0，A5，A7，A0，A0 及 A3；而第一个转换结果存放在 ADC12MEM6 中，则其后 5 个转换结果分别存放在 ADC12MEM7，8，9，10 及 11 中(见图 3.120)。而本序列的各个转换参数也在相应的转换存储控制寄存器中，可归纳为表 3.23。

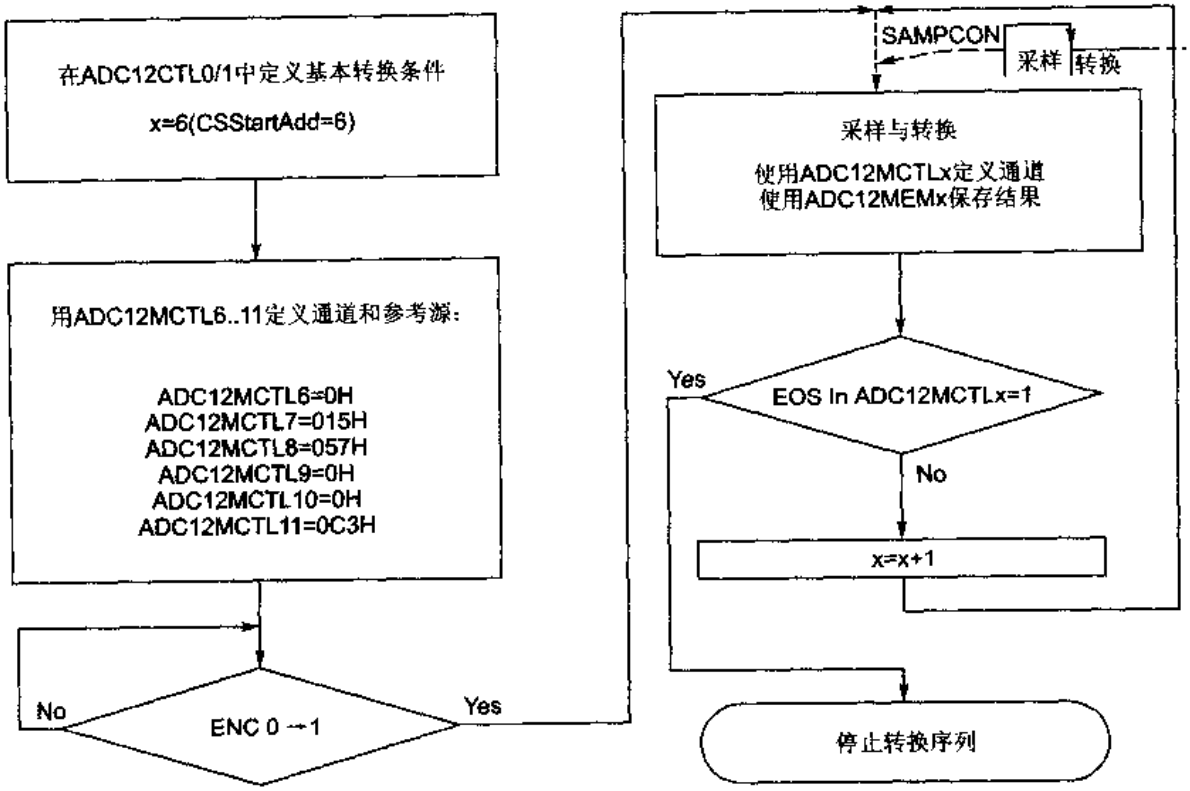


图 3.119 序列通道单次转换流程图

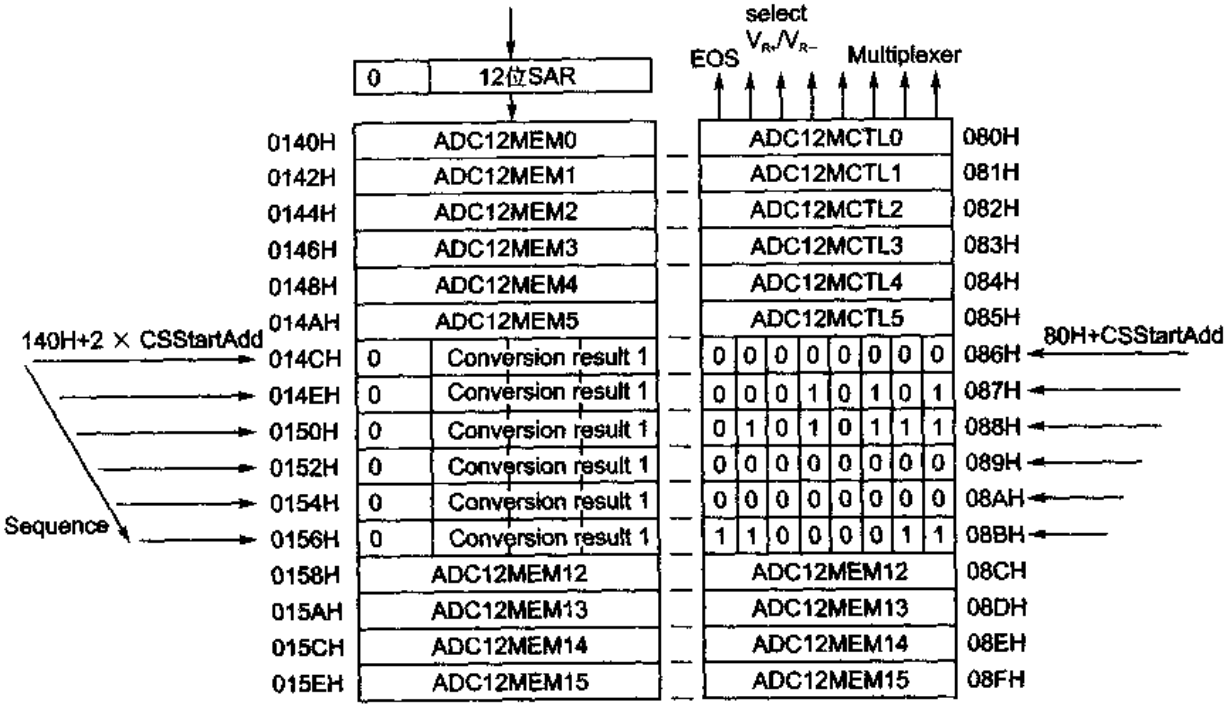


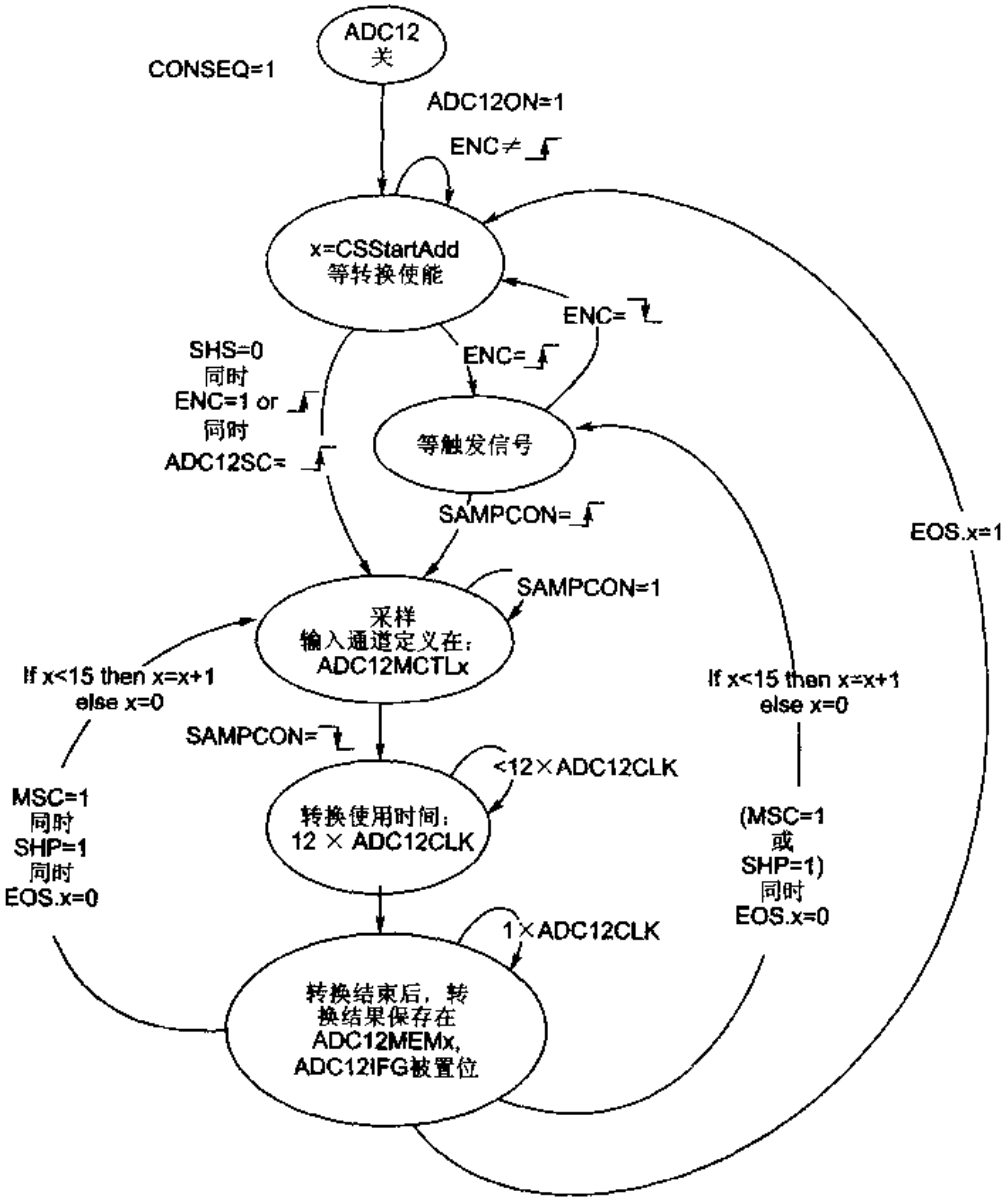
图 3.120 序列通道单次转换存储示例

表 3.23　序列通道转换举例

转换通道	参考电压		ADC12MEMx	ADC12MCTLx
	V_{R+}	V_R		
A0	AV_{CC}	AV_{SS}	ADC12MEM6	ADC12MCTL6=0
A5	V_{REF+}	V_{REF-}	ADC12MEM7	ADC12MCTL7=15H
A7	V_{REF+}	V_{REF-}	ADC12MEM8	ADC12MCTL8=57H
A0	AV_{CC}	AV_{SS}	ADC12MEM9	ADC12MCTL9=0
A0	AV_{CC}	AV_{SS}	ADC12MEM10	ADC12MCTL10=0
A3	V_{REF+}	V_{REF-}	ADC12MEM11	ADC12MCTL11=0C3H

在表中的 ADC12MCTL11=0C3H，则该寄存器的最高位(EOS)为 1，说明在本序列中，转换到这次完成时，本次序列转换完全结束。

序列通道单次转换同样遵循它的转换状态图(见图 3.121)。



注：x—指转换存储寄存器(ADC12MEM0～15)
及转换存储控制寄存器(ADC12MCTL0～15)

图 3.121　序列通道单次转换状态图

为了再次执行同一序列或一个新的序列，ENC 必须先复位然后再次置位。在 ENC 再次置位前的输入信号将被忽略。但是序列转换一旦开始，ENC 位即可复位，而序列转换会正常完成，见图 3.122。

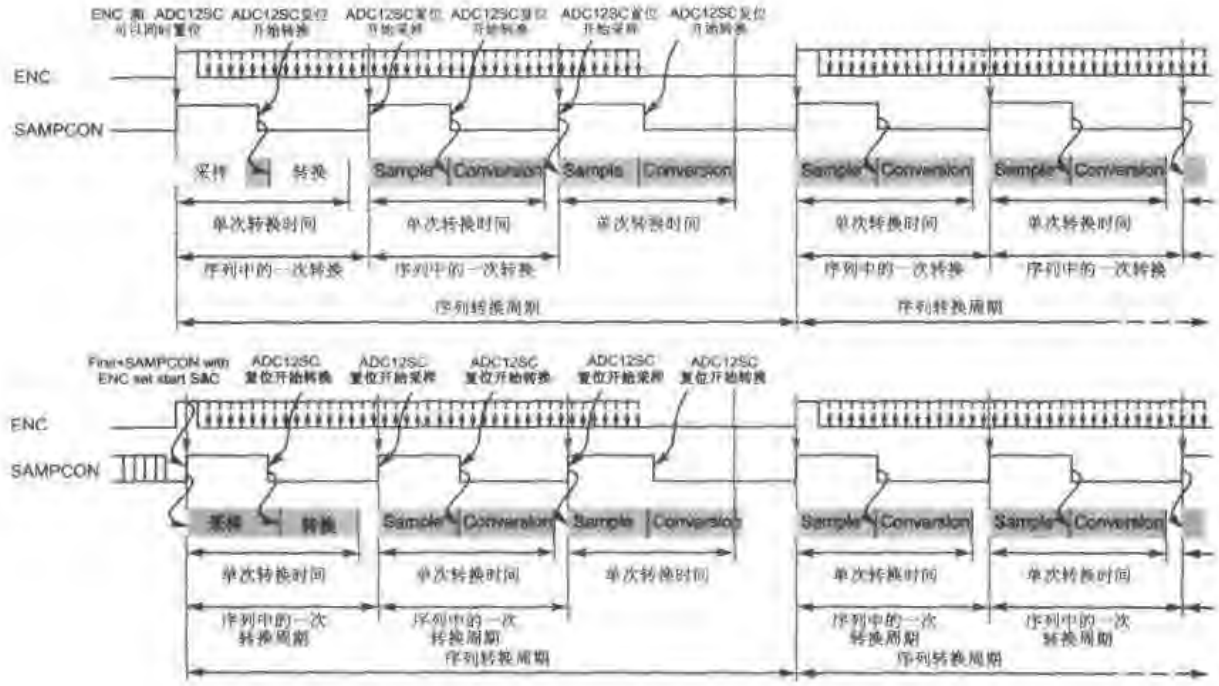


图 3.122　ENC 位对序列单次转换的影响

如果在序列转换已经开始但没有结束、且 ENC 保持为高时改变转换模式，则原序列仍正常完成。新的转换模式（单通道单次模式除外）在原序列完成后生效。这时，如果原模式未进行采样或正在进行的采样及转换已经完成，则原来模式停止。如果原序列没有完成（如上面的举例，一个序列共 6 次采样转换，如果已经转换了前面 3 次，这时改变转换模式，则该序列没有完成，但已经有 3 次转换正常完成），则已完成的转换结果是有效的。

如果在序列转换已经开始面又没有结束、且 ENC 已经翻转时改变转换模式，则原序列仍正常完成。新的转换模式（单通道单次模式除外）在原序列完成后生效。

将 CONSEQ.1 位复位选择单通道单次转换，并且将 ENC 复位，能使当前正在进行的序列转换模式立即停止。这时转换存储寄存器内的数据不可靠，因为转换立即停止，没有真正完成。中断标志也不一定置位。

（3）单通道多次转换模式

单通道多次转换与单通道单次转换类似。在选定的通道上多次转换，直到用户软件将其停止为止。每次转换完成，转换结果存放在相应的 ADC12MEMx 中，由相应的中断标志位 ADC12IFG.x 置位来标志转换结束。因为已经有了中断标志，如果允许中断，将产生中断服务请求。

在这种模式下，改变转换模式，不必先停止转换，在当前正在进行的转换结束后，可改变转换模式。该模式的停止可有如下几种办法：

- 使用 CONSEQ=0 的办法，改变为单通道单次模式；
- 使用 ENC=0 直接使当前转换完成后停止；

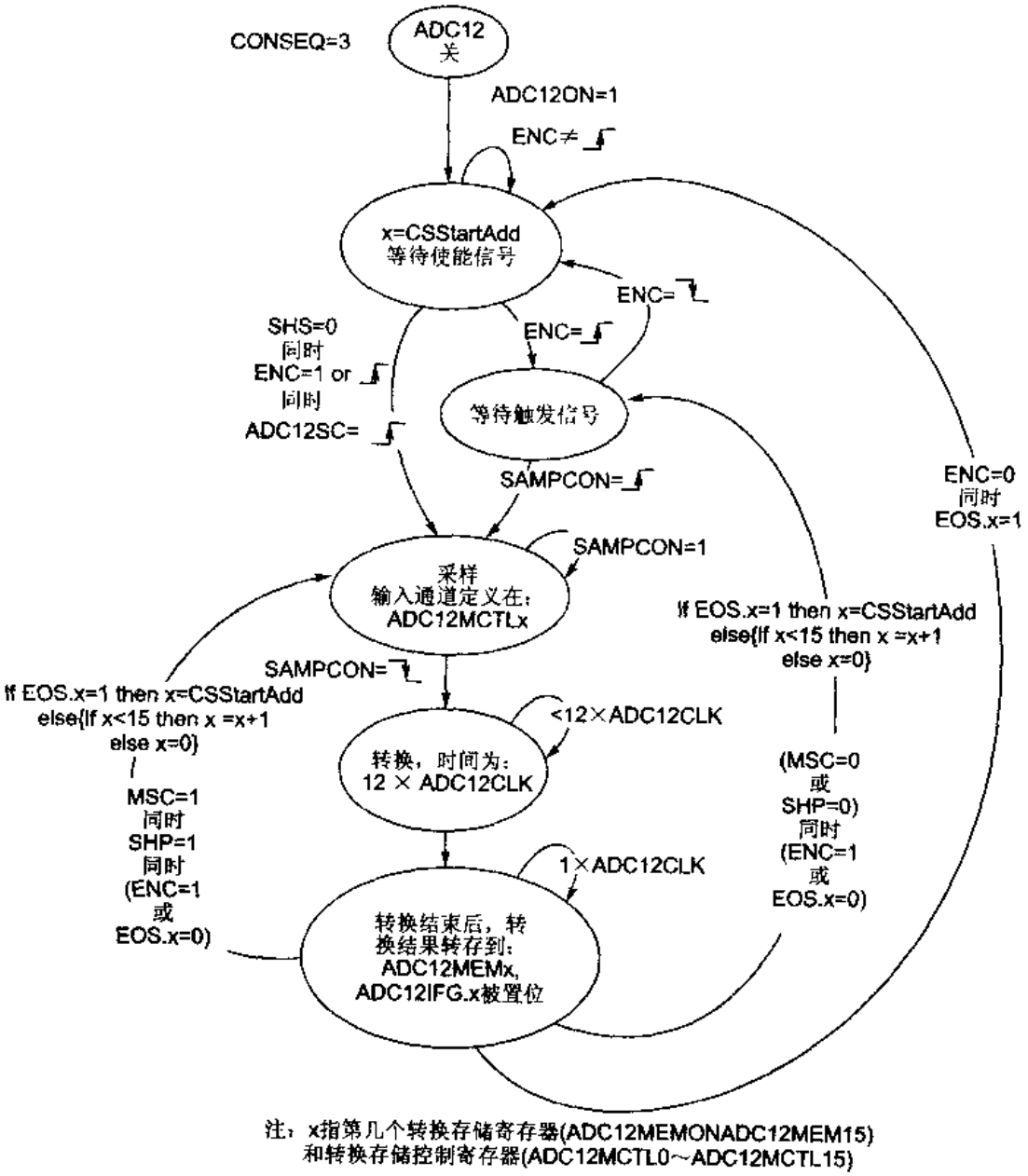


图 3.124 序列通道多次转换状态图

4. 在各种模式下的转换举例

(1) 单通道单次转换

```
#include "MSP430x13x.h"
#define ADCMEM ((int *) 0x0140)          // ADC12MEMx 定义
void Init(void);                         // 系统初始化
void main(void)
{
    unsigned int i;
    unsigned int j;
    unsigned int Result[2];              //用以保存转换结果
```

```
    Init();                                    对 AIC 进行初始化
    while (1)
    {
       for (i=0;i<2;i++)
       {
            ADC12CTL0 |= 0x01;                 开始转换
            ADC12CTL0 &= ~0x01;
          for (j=0;j<=1000;j++);               延时,准备触发序列中的下一次转换
       }
       while ((ADC12CTL1&0x01)==1);            等待整个序列的转换完成
       for(j = 0 ; j <2; j++)
       {
            Result[j] = ADCMEM[j];
       }
       for (i=0;i<=10000;i++);                 两个序列之间的一个延时
    }
    void Init(void)
    {
       WDTCTL = WDTPW + WDTHOLD;               停止 Watchdog
    /* ADC12 Settings: */
       P6SEL = 0xFF;                           所有 P6 口线均为 ADC 模块使用
       ADC12CTL0 &= ~0x02;                     在进行设置时首先复位 ADC 的转换使能
                                               选择参考电压和输入管脚
       ADC12MCTL0   = 0x00;                    REF = AVss, AVcc; Input = A0
       ADC12MCTL1   = 0x8a;                    REF = AVss, AVcc; Input = A10
                                               ;ADC12MCTL1 是最后一个转换通道
       ADC12CTL1 = 0x0202;                     ADC12SC 位触发采样和保持
                                               采样脉冲由采样定时器产生
                                               时钟源:内部振荡器
                                               时钟分频: 1
                                               转换模式:多通道、单次转换
       ADC12CTL0 = 0x0010+SHT0_8;
       ADC12CTL0 |= 0x02;                      使能 ADC 转换
    }
```

(2) 序列通道单次转换

```
#include "MSP430x14x.h"
#define ADCMEM ((int*) 0x0140)                 // ADC12MEMx 定义
//-------------------------------------------------------------
void Init(void);                               // 系统/控制寄存器初始化
void main(void)
{
    unsigned int i;
```

```
    unsigned int j;
    unsigned int Result[2];                    // 用以保存转换结果
    Init();                                    // 对 ADC 进行初始化
    while (1)
    {
        for (i=0;i<2;i++)
        {
            ADC12CTL0 |= 0x01;                 // 开始转换
            ADC12CTL0 &= ~0x01;
            for (j=0;j<=1000;j++);             // 延时,准备触发序列中的下一次转换
        }
        while ((ADC12CTL1&0x01)==1);           // 等待整个序列的转换完成
        for(j = 0 ; j <2; j++)
        {
            Result[j] = ADCMEM[j];
        }
        for (i=0;i<=40000;i--);                // 两个序列之间的一个延时
    }
}
void Init(void)
{
    WDTCTL = WDTPW + WDTHOLD;                  // 停止 Watchdog
/* ADC12 Settings: */
    P6SEL = 0xFF;                              // 所有 P6 口线均为 ADC 模块使用
    ADC12CTL0 &= ~0x02;                        // 在进行设置时首先复位 ADC 的转换使能
                                               // 选择参考电压和输入管脚
    ADC12MCTL0  = 0x00;                        // REF = AVss, AVcc; Input = A0
    ADC12MCTL1  = 0x8a;                        // REF = AVss, AVcc; Input = A10
                                               // ADC12MCTL1 是最后一个转换通道
    ADC12CTL1 = 0x0202;                        // ADC12SC 位触发采样和保持
                                               // 采样脉冲由采样定时器产生
                                               // 时钟源:内部振荡器
                                               // 时钟分频: 1
                                               // 转换模式:多通道、单次转换
    ADC12CTL0 = 0x0010+SHT0_8;
    ADC12CTL0 |= 0x02;                         // 使能 ADC 转换
}
```

(3) 单通道多次转换

```
#include "MSP430x14x.h"
#define ADCMEM ((int*) 0x0140)                 // ADC12MEMx 定义
void Init(void);                               // 初始化系统及 ADC 寄存器
interrupt [ADC_VECTOR] void ADC12(void);
unsigned int Result;
```

```
void main(void)                                    //使用中断方式
{
    Init();
    ADC12CTL0 |= 0x01;                             //开始转换
    LPM0;                                          //进入低功耗状态,等待中断
    _NOP();                                        //
}
void Init(void)
{
    WDTCTL = WDTPW + WDTHOLD;                      //停止 Watchdog
  * ADC12 Settings: */
    P6SEL = 0xFF;                                  //所有 P6.11线均为 ADC 模块使用
    ADC12CTL0 &= ~0x02;                            //在进行设置时首先复位 ADC 的转换使能
    ADC12CTL0 = SHT0_8+MSC+ADC12ON;                //内部振荡器,置位 MSC 位,因此转换能自动进行
    ADC12CTL1 = 0x0204;                            // ADC12SC 位触发采样和保持
                                                   //采样脉冲由采样定时器产生
                                                   //时钟源:内部振荡器
                                                   //时钟分频: 1
                                                   //转换模式:单通道、多次转换
                                                   //选择参考电压和输入管脚
    ADC12MCTL0    = 0x0a;                          // REF = AVss, AVcc; Input = A10
                                                   // source = Temperature diode
    ADC12IE = 0x001;                               //使能通道 10 转换完成后中断
    _EINT();
    ADC12CTL0 |= 0x02;                             //使能 ADC 转换
}
interrupt [ADC_VECTOR] void ADC12(void)            // ADC 中断处理程序
{
    Result = ADCMEM[0];
}
```

(4) 序列通道多次转换

```
#include <msp430x13x.h>
#define ADCMEM ((int*) 0x0140)                     // ADC12MEMx 定义
void Init(void);
unsigned int Result[2];
void main(void)                                    //使用中断方式取结果
{
    Init();
    ADC12CTL0 |= 0x01;                             //开始转换
    LPM0;                                          //进入低功耗状态,等待中断
    _NOP();
}
void Init(void)
```

```
{
    WDTCTL = WDTPW + WDTHOLD;
    P6SEL = 0xFF;                          // 所有 P6.1 线均为 ADC 模块使用
    ADC12CTL0 &= ~0x02;                    // 在进行设置时首先复位 ADC 的转换使能
    ADC12MCTL0= 0x00;                      // REF = AVss, AVcc ;Input = A0
    ADC12MCTL1= 0x8a;                      // REF = AVss, AVcc ;Input = A10
    ADC12CTL1 = 0x0206 ;                   // 第一个转换结果被放在 ADC12MEM0
                                           // 第二个转换结果被放在 ADC12MEM1
                                           // 采样脉冲由采样定时器产生
                                           // 时钟源:内部振荡器
                                           // 时钟分频: 1
                                           // 转换模式:多通道、多次转换
    ADC12CTL0 = SHT0_8+ MSC+ ADC12ON ;

    ADC12IE = 0x002;                       // 使能转换中断
    ADC12CTL0 |= 0x02;
    _EINT();                               // 使能全局中断
}
interrupt [ADC_VECTOR] void ADC12(void)    // ADC 中断处理程序
{
    unsigned int  i;
    for(i = 0 ; i < 2 ; i++ )
    {
        Result[i] = ADCMEM[i] ;            // 读取结果
    }
}
```

5. ADC12 的采样

ADC12 的采样与保持电路可通过程序设置 ADC12CTL0 和 ADC12CTL1 中的控制位来实现。图3.125为ADC12的采样/保持电路图。采样时钟可有4种来源：ADC12OSC，

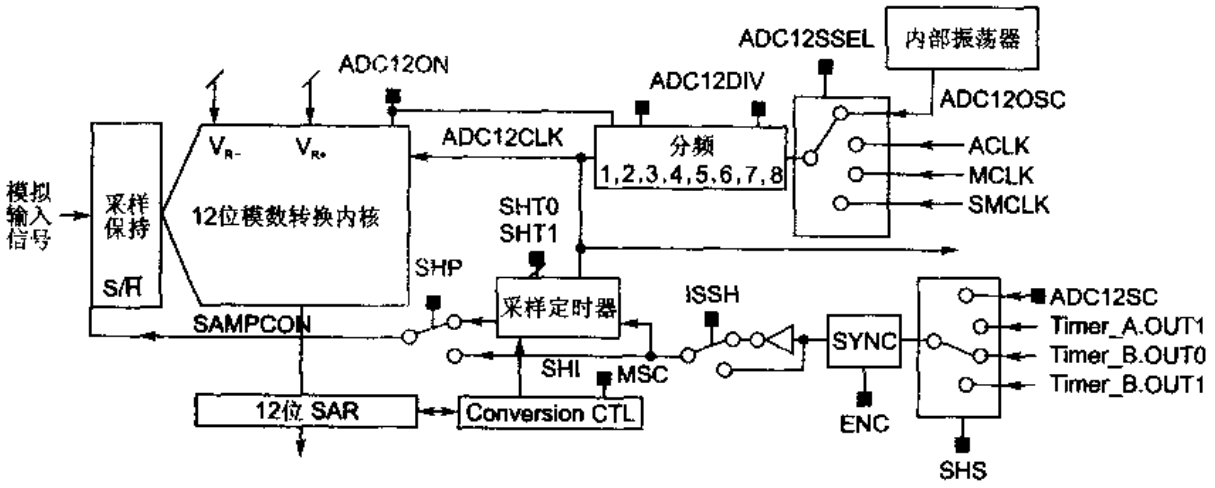


图 3.125 ADC12 的采样/保持电路图

ACLK、MCLK 及 SMCLK 等，由分频器分频，形成 ADC12CLK 送入 ADC12 内核。采样触发也有 4 种来源：ADC12SC、Timer_A. OUT1、Timer_B. OUT0 及 Timer_B. OUT1。

转换速度由最终的 ADC12CLK 周期决定。

当采样信号 SAMPCON 为高时，采样/保持电路对模拟输入信号采样。在 SAMPCON 的下降沿开始转换；当 SAMPCON 为低时，模拟输入信号由采样/保持电路保持住。整个转换需要 13 个时钟周期，如图 3.126 所示。

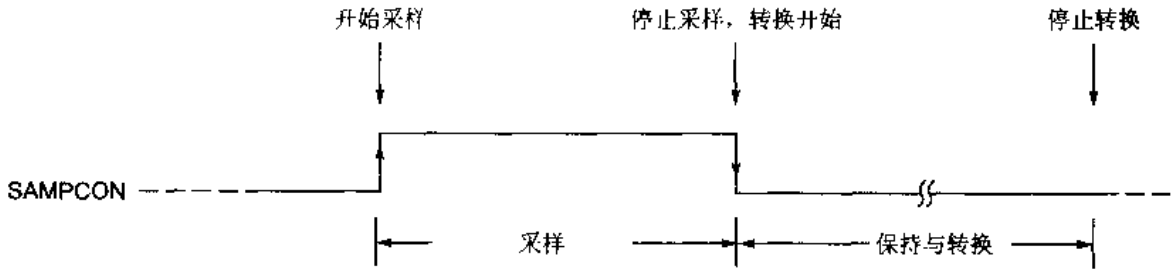


图 3.126　整个转换过程示意图

在采样期间，输入模拟信号必须保持稳定，这样可得到较为精确的结果。同时应将相邻通道的数字活动取消，可避免电源毛刺、地电平波动以及交互串扰等影响转换结果。

从图 3.125 可以看出，采样的控制及转换的开始 SAMPCON 有很多来源，可以来自采样信号输入，也可来自采样定时器。采样信号输入在发生输入选择时有效。当采样定时器作为 SAMPCON 的来源时，采样信号输入可用于触发采样定时器。

3.8.3　ADC14 模数转换模块

ADC14 模块为 12+2 位精度模数转换器。它是 MSP430X32X 系列中的一个片内外围模块。相对于 ADC10 和 ADC12 模块而言，操作起来简单多了。该模块大致有如下特点：

- 有 8 路模拟信号输入，在不用于模数转换时，可作为数字端口；
- 有 4 路模拟输入端用于可编程电流源；
- 可用做比例测量(用于概测)或绝对测量(用于精确测量)；
- 内建采样与保持电路；
- 在转换结束时提供中断信号，同时有转换结果暂存器用于暂存结果，直到下次转换开始；
- 低功耗，可将模块的供电开启或关闭；
- 整个转换过程由模块独立完成，不需要 CPU 的额外开销；
- 可选 12 位或 14 位分辨率，且有较快的转换速度。

1. ADC14 的基本原理

图 3.127 是 ADC14 的结构框图，它表明该模块有 8 个模拟输入通道，其中 A0、A1、A2 及 A3 还可作为 4 个电流源输出，其值由外接电阻 R_{ext} 设定。

8 个模拟输入端也可以用于数字输入，对 AEN 寄存器的相应位编程可实现模拟端口的数字输入。通道上的数据(数字量)可由 AIN 寄存器读取。但请注意，与 ADC10 和 ADC12 一样，当正在进行采样或转换时，不要让这些端口有数字量活动，因为这样会产生噪声，对转换有影响，得到不正确的转换结果。

2. ADC14 的控制寄存器

与 ADC10 或 ADC12 相比较，ADC14 的寄存器相对较少，只有 4 个，见图 3.128。但同样设计者只有与它们打交道才能对 ADC14 进行操作。4 个寄存器都须使用字方式操作，位于字地址空间。

图 3.128　ADC14 的寄存器

(1) ACTL　转换控制寄存器

该寄存器控制着转换参数的选择，为 16 位寄存器，各位定义如下：

15	14,13	12	11,10,9	8,7,6	5,4,3,2	1	0
0	ADCLK	Pd	Range	Current	INCH	V_{REF}	SOC

ADCLK　ADC 时钟选择位，如表 3.24 所列。

表 3.24　ADC 时钟选择

ACTL.14	ACTL.13	ADCLK
0	0	MCLK
0	1	MCLK/2
1	0	MCLK/3
1	1	MCLK/4

Pd　省电模式控制位。

0　接通电源；

1　省电模式，短开电源。

Range　测量范围选择位。ADC14 可以有 4 个测量范围供选择，选定之后将在指定的测量范围内产生 12 位精度的测量结果。测量范围的定义要在测量之前确定好，如表 3.25 所列。如果 ACTL.11＝1，则转换器在采样期间通过对输入信号的两次采样自动选择测量范围。第一次采样用于选择测量范围，第二次采样用于 12 位转换。

表 3.25　测量范围

ACTL.11	ACTL.10	ACTL.9	范　围	模拟输入范围
0	0	0	A	$0.00 \cdot V_{REF} \delta V_{IN} < 0.25 \cdot V_{REF}$
0	0	1	B	$0.25 \cdot V_{REF} \delta V_{IN} < 0.50 \cdot V_{REF}$
0	1	0	C	$0.5 \cdot V_{REF} \delta V_{IN} < 0.75 \cdot V_{REF}$
0	1	1	D	$0.75 \cdot V_{REF} \delta V_{IN} < 1.00 \cdot V_{REF}$
1	X	X	自动	

其中 V_{REF} 为 SV_{CC} 引脚的电压，它或由外部提供，或直接接在 AV_{CC} 上。在选定适当的测量范围后，所选的输入端口上的模拟信号将被连接到转换器的输入端准备转换。转换的结果为

$$N_{adc}=INT\left|\frac{V_{IN}\times 2^{11}}{V_{REF}}-2^{13}\times ACTL.10-2^{12}\times ACTL.9\right|$$

Current　电流源输出控制位。

当 ACTL.8=0 时，ACTL.7、6 两位所表示的二进制数为所选的作为电流源输出的通道号；

当 ACTL.8=1 时，没有电流源输出。

ADC14 的电流源原理图如图 3.129 所示。有 4 个模拟端口可用做电流源输出。电流源输出的电流（I_{source}）由外接电阻（R_{ext}）设定，由 A0～A3 输出。输出电流大小为

$$I_{source}=(0.25\times SV_{CC})/R_{ext}$$

其中 V_{REF} 为 SV_{CC} 端的电压；R_{ext} 为 SV_{CC} 与 R_{ext} 之间的电阻值。

当 ADC14 在做精密的外接传感器测量时，有时需要精密恒流源。这时可由 ADC14 自身对外提供。

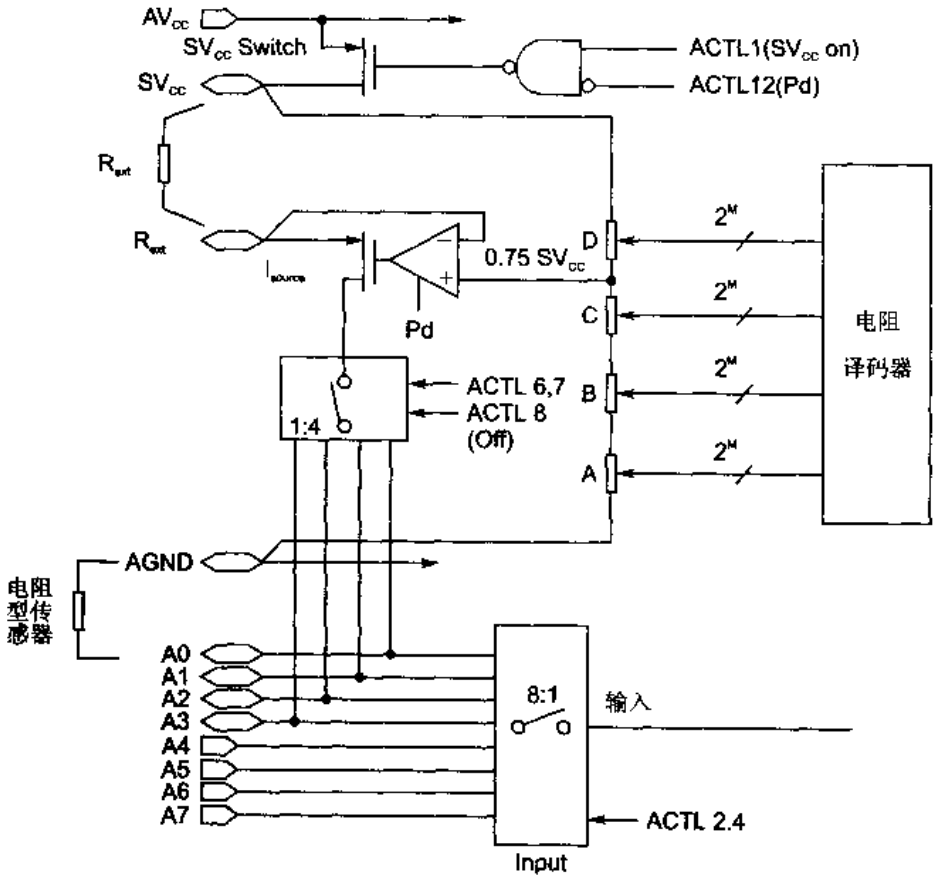


图 3.129　ADC14 的电流源原理图

INCH　模拟输入通道选择位。

当 ACTL.5=0 时，ACTL.4、3、2 表示的二进制数就是选择的模拟输入通道；

当 ACTL.5=1 时，没有模拟输入信号。

V_{REF}　　V_{REF}来源。

0　SV_{CC}开关关闭，ADC 的参考源由外部提供；

1　SV_{CC}开关打开，ADC 的参考源由内部提供。

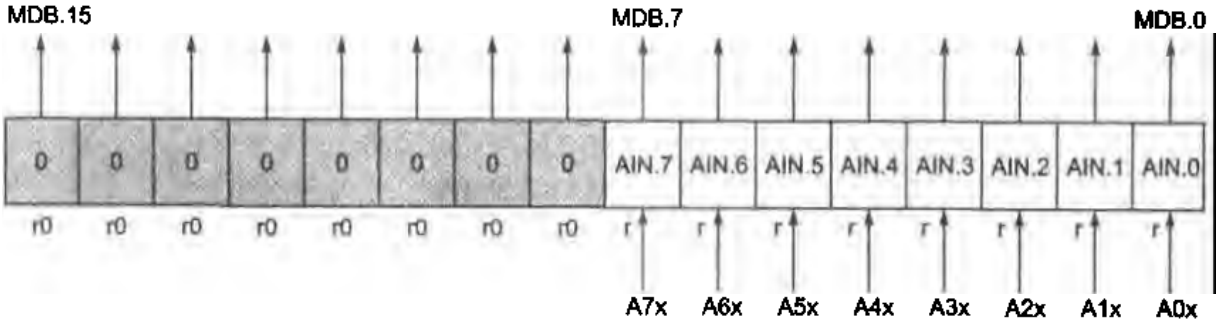
SOC　　转换启动控制位。

0　不启动；

1　启动转换。

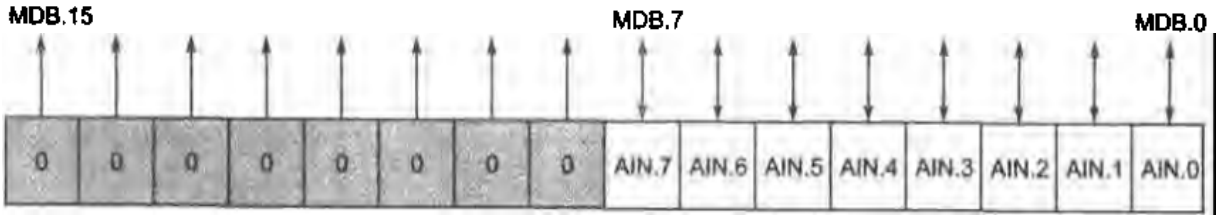
(2) AIN　输入寄存器

前面说过，A0～A7 端口上的信号既可作为模拟量，也可作为数字量输入。数字量可以通过访问输入寄存器读出。AIN 作为只读寄存器连接于 16 位 MDB，高字节始终为 0。该寄存器各位的定义如下：



(3) AEN　输入允许寄存器

该寄存器也是 16 位寄存器，同样只用了低 8 位，如下所示：



当为 0 时，表明对应端口为模拟量输入，如果要读 AIN，则为 0；

当为 1 时，表明对应端口为数字量输入，引脚上将为逻辑电平。

(4) ADAT　数据寄存器

该寄存器是 A/D 转换的结果值。转换数据在转换结束时进入该寄存器，并保持到由 SOC 位置位引起的下一次转换的启动。该寄存器为 12 位或 14 位有效值，最低位为转换结果的最低位，高 2 位或 4 位为 0。

3. 采样与转换

与所有的模数转换一样，在转换之前必须选对输入信号进行采样。同时必须保证模拟输入信号在采样期间的有效性和稳定性。在整个采样与转换过程中也不允许有相邻通道上的数字信号活动，以保证没有电源毛刺、接地电压变化及交互干扰等对输入信号的影响。ADC14 对输入信号的采样时序如图 3.130 所示。在 SOC 信号的作用下启动采样。同时要注意 ADC14 模块的上电过程，只有当模块供电稳定之后，才对其操作。ADC14 模块由省电模式到激活模式需要 6 μs 时间。

ADC14 的转换过程在采样之后。转换的时钟频率为 ADCLK 频率的 1/12。ADCLK 频

在 12+2 位模式下，对输入的模拟信号进行两次采样，第一次确定输入信号的范围，第二次采样之后将进行 12 位转换，如图 3.132 所示，共需要的时钟周期为 132 个 ADCLK。

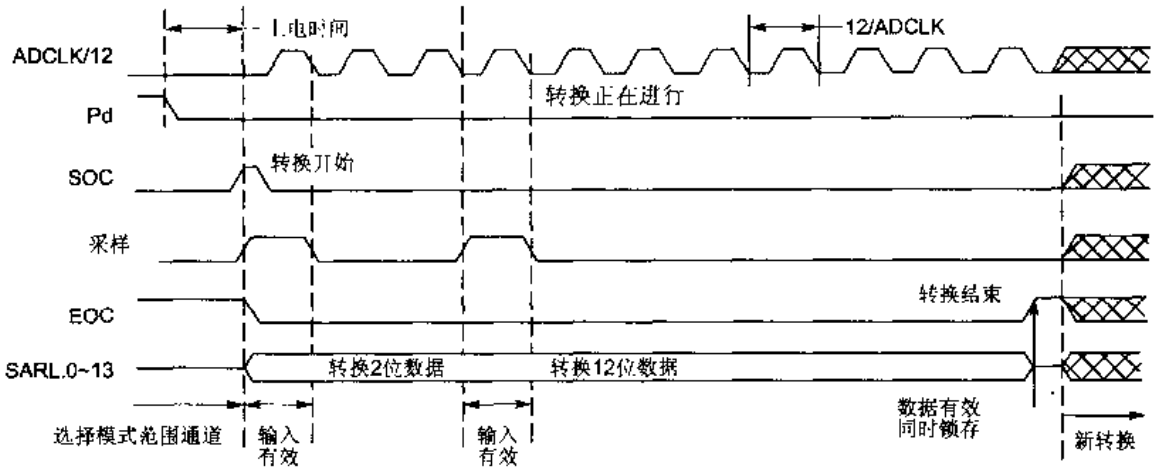


图 3.132　ADC14 的 12+2 位转换时序

图 3.133 为 ADC14 的电路原理图。

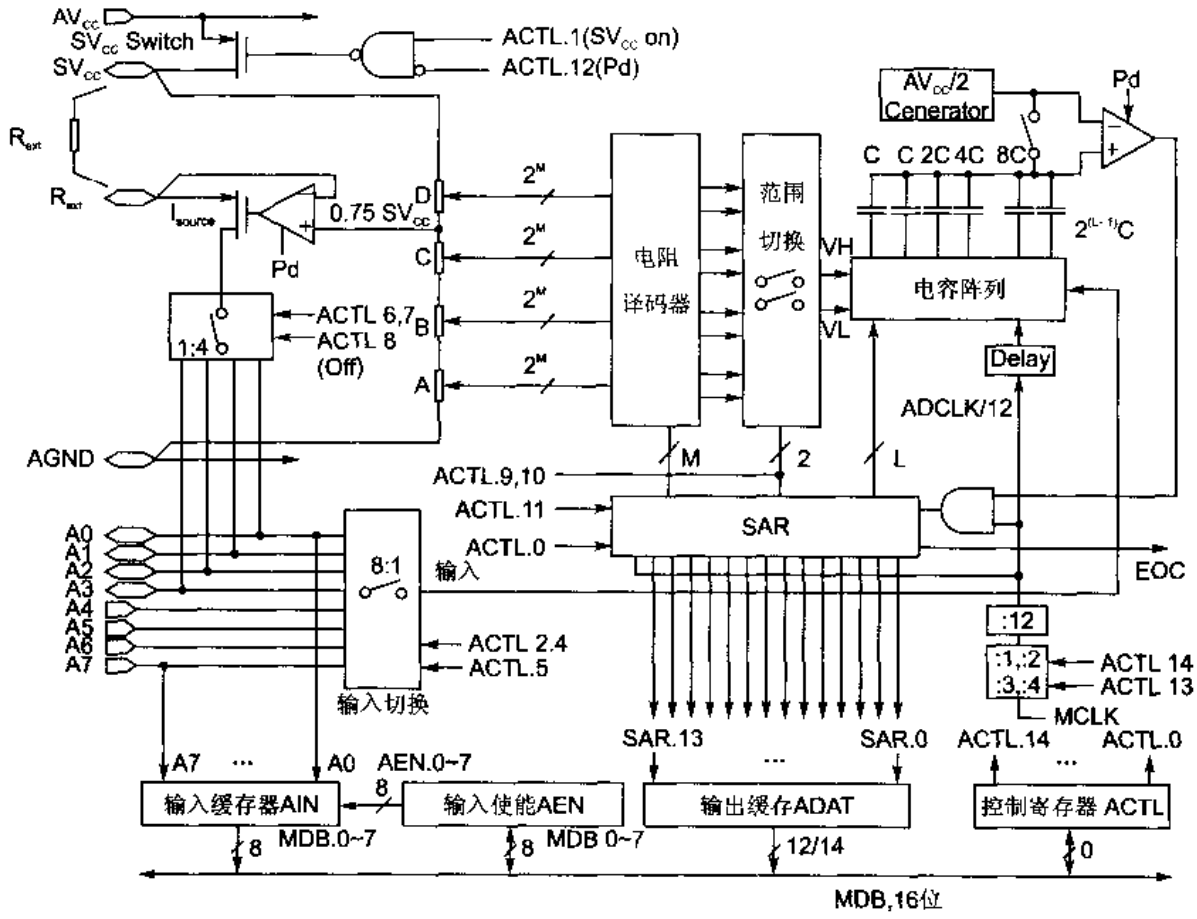


图 3.133　ADC14 的电路原理图

下面是 ADC14 的具体转换过程。整个 ADC14 模块通过 MDB 与系统交换数据，设计者通过寄存器对 ADC14 操作。ACTL 为操作 ADC14 的控制寄存器，其中的位决定着对 ADC14

如何操作；转换之后的数据暂存在输出缓存寄存器 ADAT 中，转换结束，可直接在 ADAT 中读取转换结果。转换内核为电容阵列加上电阻阵列，基于逐次逼近技术。

设置 SOC 位将激活 ADC 时钟，开始一个新的转换（包括采样）。当选择了要转换的模拟通道之后，采样过程就开始了。模拟输入电压将采样到电容阵列的正电极，采样完成后，模拟开关与 A/D 断开，这时外部模拟输入信号无效。所以，ADCLK 的频率太小，转换将不准确的原因就在这里。采样以后模拟输入信号已经与 ADC 断开，只有采样电容上电压信号有效。

在电阻阵列与电容阵列上通过逐次逼近得到模拟输入信号的量化结果为

$$N_{typ}=INT\left|\frac{V_{IN}\times 2^{11}}{V_{REF}}-2^{11}\times ACTL.10-2^{12}\times ACTL.9\right|$$

4. ADC14 应用举例

下面的应用程序将对模拟通道 A1 进行采样转换，使用中断方式读取转换结果。同时利用 MSP430X32X 系列器件的液晶驱动能力，将转换结果显示在液晶上，这里对液晶只有初始化语句，没有具体的显示程序（见 3.9 节中的液晶应用举例）。

```
#include  "msp430x32x.h"
#define   ADData   R4                          ;定义转换结果的存放位置
#define   ADReady  R7
          RSEG     CSTACK                      ;系统堆栈
          DS       0
          RSEG     CODE                        ;程序代码段
Reset     mov      #SFE(CSTACK),SP             ;初始化堆栈指针
          mov      #(WDTHOLD+WDTPW),&WDTCTL    ;停掉看门狗
          clr.b    &IFG1                       ;清所有中断标志
          clr.b    &IFG2
          mov.b    #ADIFG,&IE2                 ;使能 ADC 中断
          mov.b    #0FFh,&LCDCTL               ;设置液晶打开电源,4MUX 模式
          call     #CLEARLCD
          eint                                 ;使能总中断
          clr      ADData                      ;清除 ADC 暂存寄存器内容
          mov      #ADIN_A1+ADRNG_AUTO+ADSOC,&ACTL
                                               ;设置 AC 为 SVCC,
                                               ;输入通道为 A1,自动量程,开始转换
          clr      ADReady
Loop      cmp      #0h,ADReady                 ;如果转换已经准备了
          jz       Loop                        ;则转换,否则等待
          mov      &ADAT,ADData                ;取出转换结果
          clr      ADReady                     ;清除准备好单元
          bis      #ADSOC,&ACTL                ;开始下一次转换
          call     #SETHEX                     ;写到液晶上显示
          jmp      Loop                        ;主循环
ADInt     mov      #01h,ADReady                ; ADC14 的中断服务程序,设置转换好单元
          reti
COMMON    INTVEC                               ;中断向量表
```

```
ORG     RESET_VECTOR
DW      Reset
ORG     ADC_VECTOR
DW      ADInt
```

3.9　MSP430 液晶驱动模块

在 MSP430 系列单片机中，液晶驱动作为一个片内外围模块存在于 MSP430F4XX 和 MSPX3XX 等系列型号的器件中，其他系列的器件目前没有液晶驱动模块。MSP430 器件上的液晶显示器的控制/驱动将简化液晶显示器的显示。不同的型号中的液晶驱动能力不同，情况如表 3.26 所列。

表 3.26　型号及驱动段数

型号 MSP430X	31X	32X	33X	11X	13X	41X
驱动段数	64/92	84	120	96	128/160	160

对于液晶驱动端口，在不用于液晶驱动时，有的型号既可作输入又可作输出，而有的型号则只能作输出(请参考 3XX 系列器件手册)。液晶的驱动有 4 种方法：

- 静态；
- 2MUX　或 1/2 占空比、1/2 偏压；
- 3MUX　或 1/3 占空比、1/3 偏压；
- 4MUX　或 1/4 占空比、1/4 偏压。

对于不同系列、不同型号的液晶驱动原理，控制方法都是一样的，不同点在于驱动液晶的段数不一样，或可显示信息的多少不一样。

1. 液晶驱动的基本原理

液晶本身不发光，其显示是通过反射环境光线实现的，因此液晶本身的功耗是很低的。由于液晶的特性，液晶的驱动需要交流信号，而直流驱动将损坏液晶。在驱动电路中，液晶可以等效为电容。两个电极板分别为公共极与段极。公共极由 COMn 信号驱动，段极由 SEGn 信号驱动(常用的表示法)。

对于液晶的驱动，有 4 种方法。

(1) 静态驱动

静态驱动将只使用一个引脚作为液晶公共端 COM0，而每一段需要另一个引脚驱动，则总的液晶引脚数为

$$\text{引脚数} = 1 + \text{段数}$$

对于每一笔段都需要一引脚作为段驱动，见图 3.134(a)。

(2) 2MUX 驱动

2MUX 驱动方式将使用两个引脚作为液晶公共端 COM0，COM1，而每两段需要另一个引脚驱动，则总的液晶引脚数为

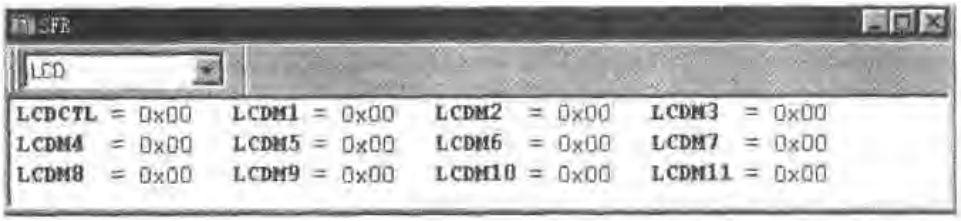
$$\text{引脚数} = 2 + \text{段数} / 2$$

对于每两笔段都需要一引脚作为段驱动，见图 3.134(b)。

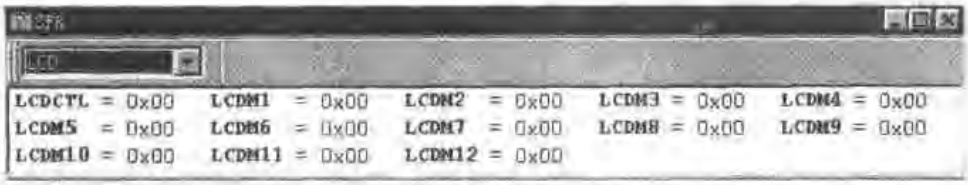
● 4MUX　4 + 80 / 4 = 24。

2. MSP430 的液晶控制与驱动

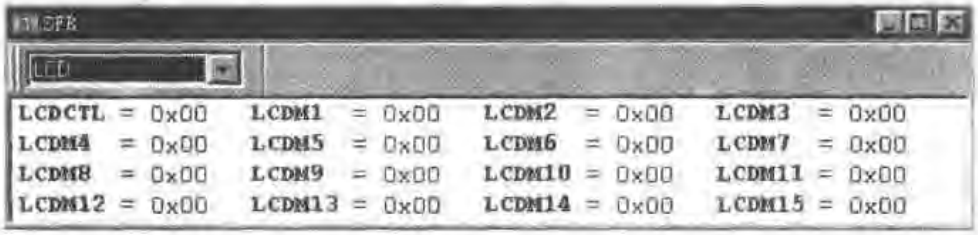
图 3.135 为 MSP430 所有系列的液晶模块的寄存器。不同系列的器件对于液晶的驱动能力是不同的，其区别主要体现在所驱动液晶的段数上。从图 3.135 可以看出：所有 430 系列的液晶模块的寄存器都有 LCDCTL 和 LCDMx，不同在于 LCDMx 寄存器的数量不一样。LCDMx寄存器的数量体现了液晶的驱动能力。LCDMx 就是液晶的显示缓存器，缓存越大，显示的内容就越多。



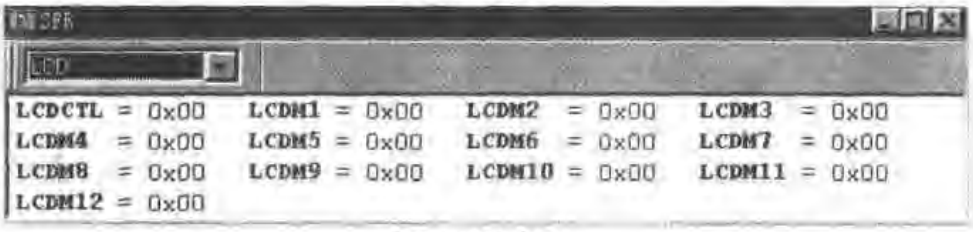
(a) MSP430X325



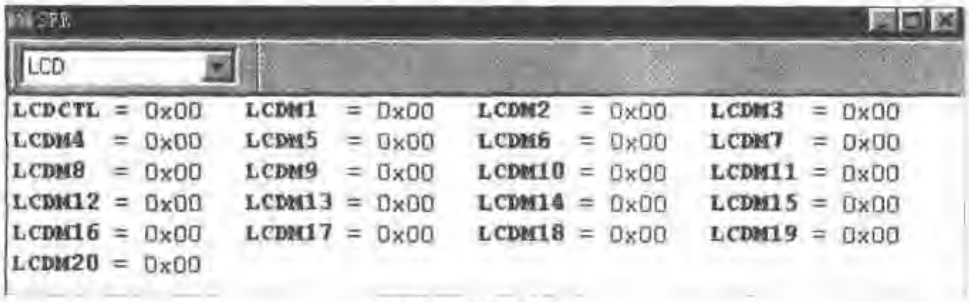
(b) MSP430X315



(c) MSP430X337



(d) MSP430F412



(e) MSP430F449

图 3.135　MSP430 液晶模块寄存器

(1) LCDCTL 控制寄存器

该寄存器定义了对液晶的各种操作。它为字节结构,位于字节地址空间,须使用字节指令予以访问。该寄存器各位的定义如下:

7	6	5	4	3	2	1	0
LCDM7	LCDM6	LCDM5	LCDM4	LCDM3	LCDM2	LCDM1	LCDM0

LCDM7,6,5 选择输出段或端口信息的组合,如表 3.27 所列。

表 3.27 输出段或端口信息组合

LCDM7	LCDM6	LCDM5	段功能	Group0	Group1	Group2	Group3	Group4	Group5	Group6	Group7
0	0	0	仅用于端口	0	0	0	0	0	0	0	0
0	0	1	S0~S15	1	0	0	0	0	0	0	0
0	1	0	S0~S19	1	1	0	0	0	0	0	0
0	1	1	S0~S23	1	1	1	0	0	0	0	0
1	0	0	S0~S27	1	1	1	1	0	0	0	0
1	0	1	S0~S31	1	1	1	1	1	0	0	0
1	1	0	S0~S35	1	1	1	1	1	1	0	0
1	1	1	S0~S39	1	1	1	1	1	1	1	0

LCDM4,3,2 选择显示模式。

LCDM2 选择禁止或允许段输出;

LCDM3,4 选择 4 种输出模式,如表 3.28 所列。

表 3.28 4 种输出模式

LCDM4	LCDM3	LCDM2	显示模式	LCD+偏压	LCD+偏压
×	×	0	显示关闭,端口输出保持稳定		
0	0	1	静态	1,1	R33,R03
0	1	1	2MUX	1,2	R33,R13,R03
1	0	1	3MUX	1,3	R33,R23,R13,R03
1	1	1	4MUX	1,4	R33,R23,R13,R03

LCDM1 通过选择模拟电压发生器的内阻来选择 LCD 驱动电压的幅度。这只对 LCD+模块有效。

0 模拟发生器的内阻为高阻抗;

1 模拟发生器的内阻为低阻抗。

LCDM0 定时发生器开关。

0 定时发生器关闭。COM 线(公共端)与段驱动端为低电平;液晶复用的端口被选为输出端口的输出不受影响;LCD+模块将切断电阻网络的电源。

1 定时发生器打开。COM 线(公共端)与段驱动端将按照液晶显存的数据输出对应的信号;液晶复用的端口被选为输出端口的输出不受影响;LCD+模块将切断电阻网络的电源。

图 3.136 为 MSP430F449 中的液晶控制器/驱动器的功能框图。其中包括:液晶控制寄存器、液晶显示缓存器、段输出控制、公共端输出控制、液晶模拟电压多路器及时序发生器等。

段输出和公共端输出将按照显示缓存器的内容送出相应信号到液晶玻璃片；模拟电压多路器将产生适合的驱动波形；控制寄存器定义对液晶的各种操作。

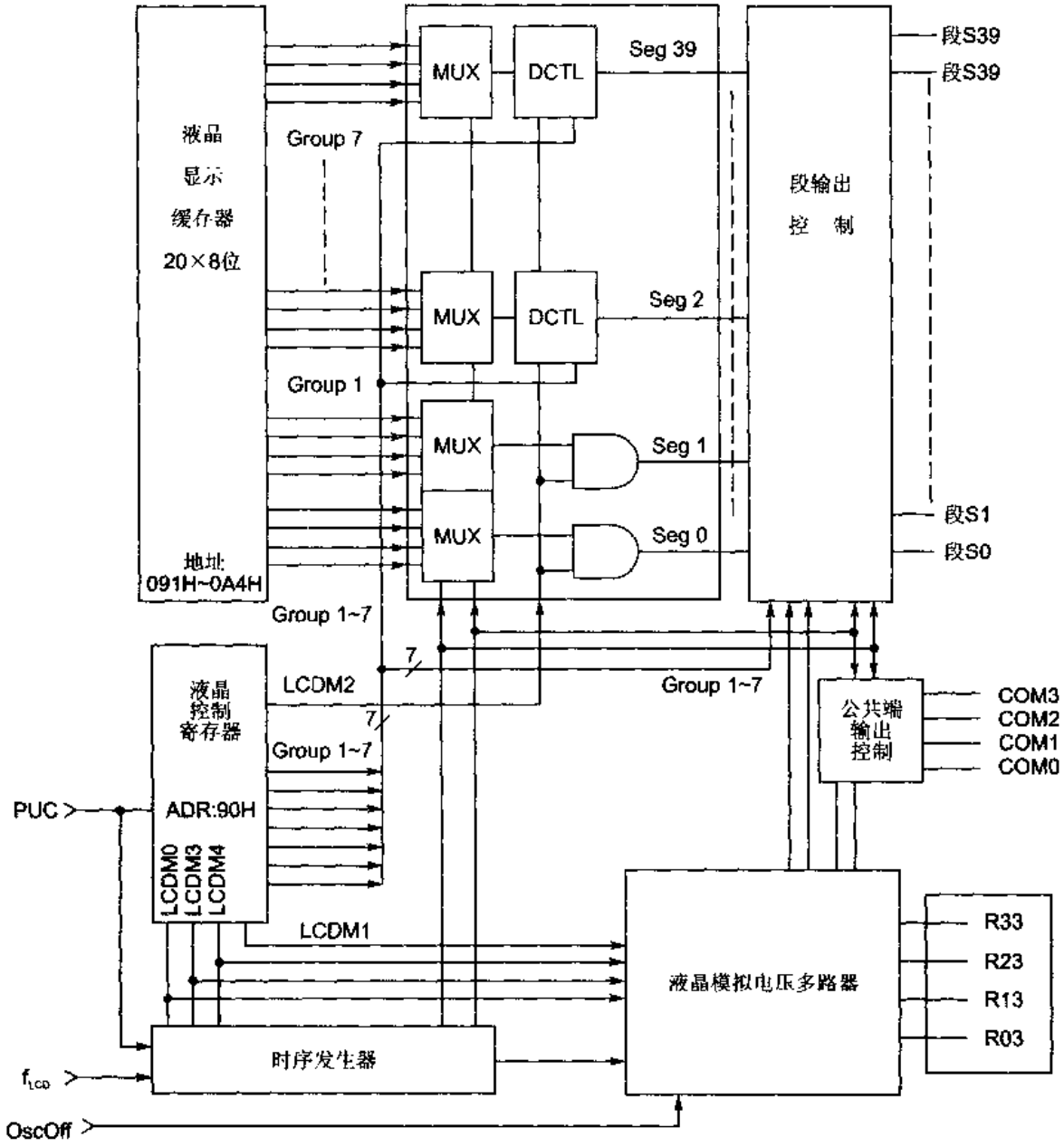


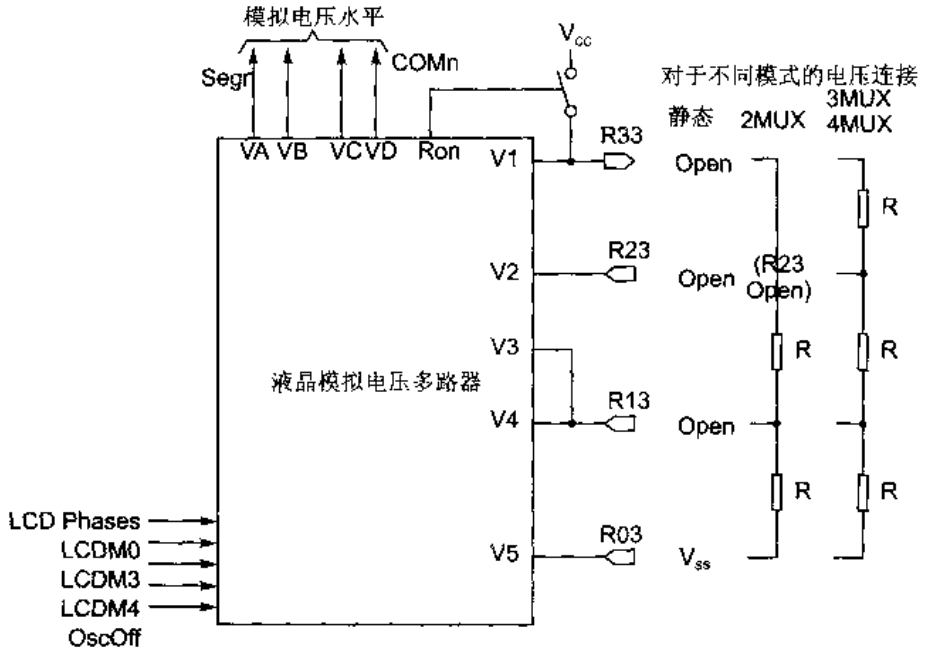
图 3.136　MSP430F449 中的液晶控制器/驱动器的功能框图

(2) 液晶模拟电压多路器

液晶所需的模拟信号由外部提供，加在 R33、R23、R13 及 R03 引脚上。一般是通过外接的等值电阻产生。图 3.137 为 MSP430 液晶模拟电压多路器的示意图，其中的 R33 和 R03 并非在所有的 MSP430 中都有，可参考具体的器件手册。

当没有 R33 和 R03 时，V1 就是 V_{CC}，V5 就是 V_{SS}，它们在内部已经连接好了。当有 R33 和 R03 时，则将提供给设计者两个方便：其一，R33 被转换到 V_{CC} 输出，由于 V_{CC} 串接了电阻，将降低电流消耗；其二，R03 在内部没有直接接在 V_{SS} 端，则允许设计者控制液晶的偏置电压，

那么可以在温度漂移时进行温度补偿，使得液晶对比度在正常状态。



OscOff	LCDM4	LCDM3	LCDM0	VA	VB	VC	VD	R33
X	X	X	0	0	0	0	0	OFF
1	X	X	X	0	0	0	0	OFF
0	0	0	1	V5/V1	V1/V5	V5/V1	V1/V5	ON
0	0	1	1	V5/V1	V1/V5	V3/V3	V1/V5	ON
0	1	X	1	V5/V1	V2/V4	V4/V2	V1/V5	ON

图 3.137 液晶模拟电压多路器示意图

(3) 液晶显示缓存器

液晶显示缓存器在 MSP430F449 中为 20 字节，而在其他系列中没有这么多，这只是驱动液晶的段数不一样而已，使用时没有任何区别。显存的各个位与液晶的段一一对应。图 3.138～3.141为显示缓存器中位与液晶段的对应关系示意图。

图 3.138 为静态方式驱动的显示缓存器与驱动的对应示意图。由于只用 1 条 COM 线(COM0)，所以可用显存的位 4 和位 0 存储段信息，其余没有用到的位可用于一般存储使用。MSP430F449 的 20 个显存可显示 40 段。

图 3.139 为 2 MUX 方式驱动的显示缓存器与驱动对应的示意图。由于用 2 条 COM 线(COM0,COM1)，所以可用显存的位 4,5 和位 0,1 存储段信息。其余没有用到的位可用于一般存储使用。MSP430F449 的 20 个显存可显示 80 段。

图 3.140 为 3MUX 方式驱动的显示缓存器与驱动的对应示意图。由于用 3 条 COM 线(COM0,COM1,COM2)，所以可用显存的位 4,5,6 和位 0,1,2 存储段信息，其余没有用到的位可用于一般存储使用。MSP430F449 的 20 个显存可显示 20×6=120 段。

图 3.141 为 4MUX 方式驱动的显示缓存器与驱动的对应示意图。由于用 4 条 COM 线(COM0,COM1,COM2,COM3)，所以可用显存的位 4,5,6,7 和位 0,1,2,3 存储段信息。所有的显示缓存器位都用于段驱动，这时能达到最多的显示。MSP430F449 的 20 个显存可显示 20×8=160 段。

输出引脚与显示元件的连接

'430 Pins		LCD Pinout PIN	COM0
S0	⟷	1	1a
S1	⟷	2	1b
S2	⟷	3	1c
S3	⟷	4	1d
S4	⟷	5	1e
S5	⟷	6	1f
S6	⟷	7	1g
S7	⟷	8	1h
S8	⟷	9	2a
S9	⟷	10	2b
S10	⟷	11	2c
S11	⟷	12	2d
S12	⟷	13	2e
S13	⟷	14	2f
S14	⟷	15	2g
S15	⟷	16	2h
S16	⟷	17	3a
S17	⟷	18	3b
S18	⟷	19	3c
S19	⟷	20	3d
S20	⟷	21	3e
S21	⟷	22	3f
S22	⟷	23	3g
S23	⟷	24	3h
S24	⟷	25	4a
S25	⟷	26	4b
S26	⟷	27	4c
S27	⟷	28	4d
S28	⟷	29	4e
S29	⟷	30	4f
S30	⟷	31	4g
S31	⟷	32	4h
COM0	⟷	33	COM0
COM1	NC		
COM2	NC		
COM3	NC		

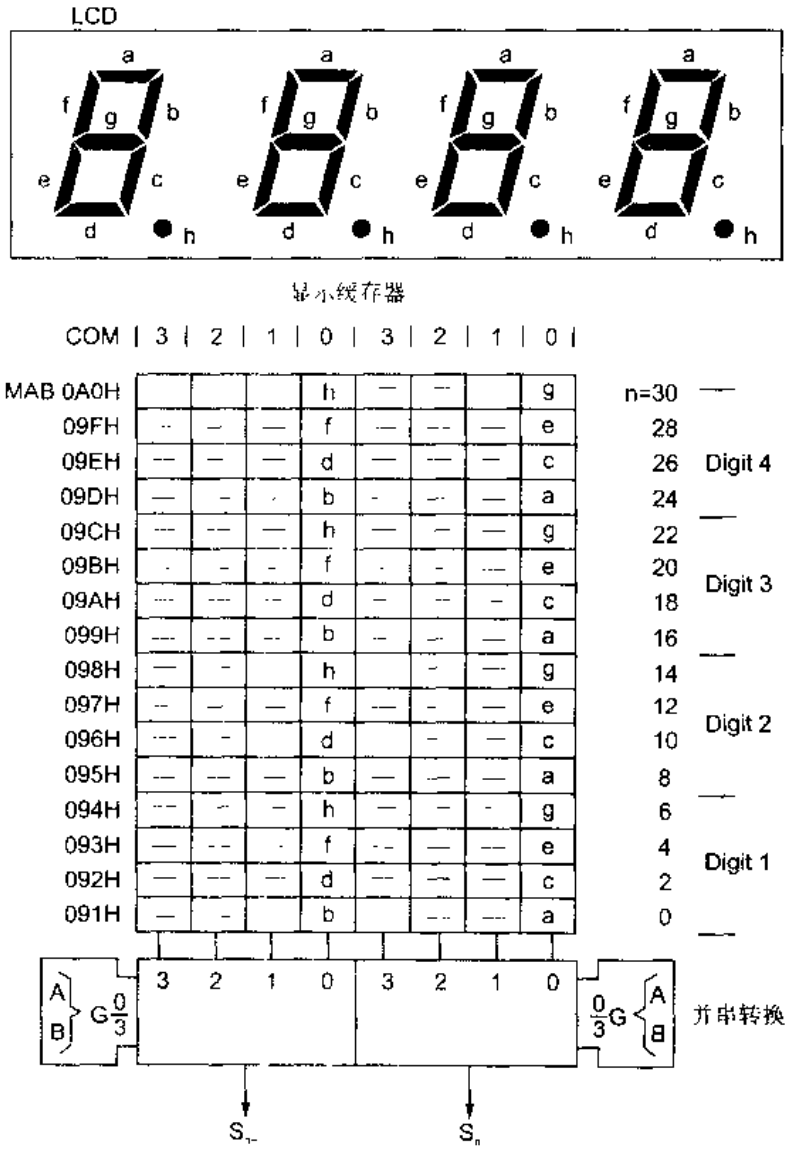


图 3.138 静态方式显示缓存器中位与液晶段的对应关系示意图

3. 液晶应用举例

下面的程序在 MSP430F44X 器件上连接段码液晶显示器，显示字符 0～6。在设置好液晶控制器与基本定时器之后，直接将要显示的数据(注意，这里的显示数据为显示段码，不是要显示的数字)写入显示缓存器即可。

程序清单如下：

```
#include  "msp430x44x.h"
char digit[10] = {
0xB7, /* "0" LCD段码 a+b+c+d+e+f */
0x12, /* "1" */
0x8F, /* "2" */
0x1F, /* "3" */
0x3A, /* "4" */
```

输出引脚与显示元件的连接

'430 Pins		LCD Pinout		
		PIN	COM0	COM1
S0	⟷	1	1f	1a
S1	⟷	2	1h	1b
S2	⟷	3	1d	1c
S3	⟷	4	1e	1g
S4	⟷	5	2f	2a
S5	⟷	6	2h	2b
S6	⟷	7	2d	2c
S7	⟷	8	de	2g
S8	⟷	9	3f	3a
S9	⟷	10	2h	3b
S10	⟷	11	3d	3c
S11	⟷	12	3e	3g
S12	⟷	13	4f	4a
S13	⟷	14	4h	4b
S14	⟷	15	4d	4c
S15	⟷	16	4e	4g
S16	⟷	17	5f	5a
S17	⟷	18	5h	5b
S18	⟷	19	5d	5c
S19	⟷	20	5e	5g
S20	⟷	21	6f	6a
S21	⟷	22	6h	6b
S22	⟷	23	6d	6c
S23	⟷	24	6e	6g
S24	⟷	25	7f	7a
S25	⟷	26	7h	7b
S26	⟷	27	7d	7c
S27	⟷	28	7e	7d
S28	⟷	29	8f	8a
S29	⟷	30	8h	8b
S30	⟷	31	8d	8c
S31	⟷	32	8e	8g
COM0	⟷	33	COM0	
COM1	⟷	34		COM1
COM2	NC			
COM3	NC			

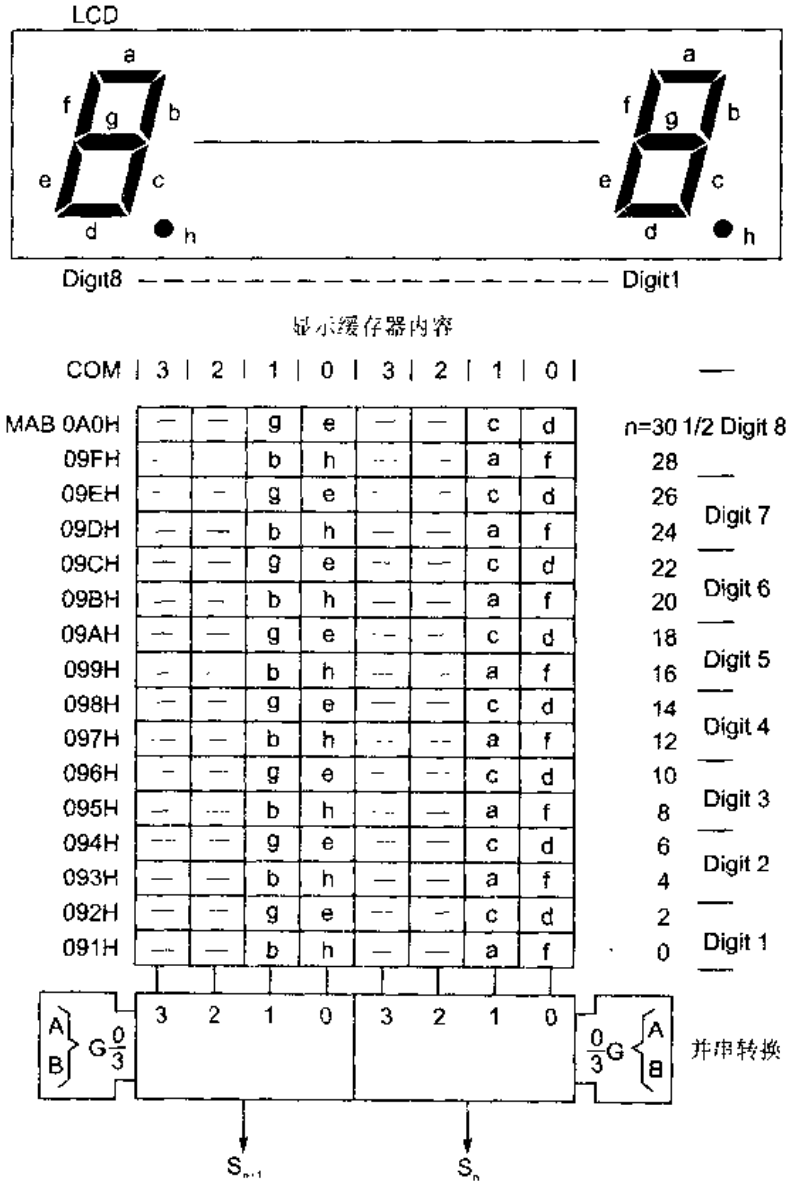
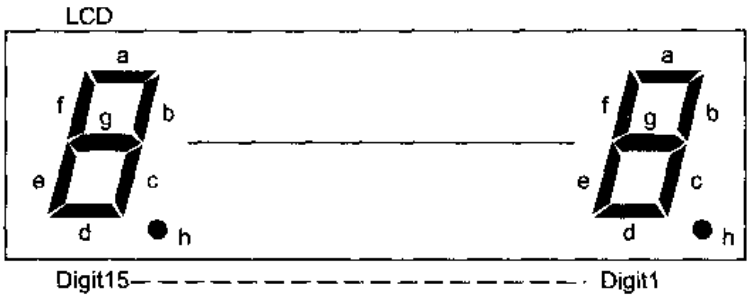


图 3.139 2MUX 方式显示缓存器中位与液晶段的对应关系示意图

```
0x3D, /* "5" */
0xBD, /* "6" */
0x13, /* "7" */
0xBF, /* "8" */
0x3F /* "9" */,
};
void main(void)
{
  int i;
  WDTCTL = WDTPW + WDTHOLD;                 //停掉看门狗
  FLL_CTL0 |= XCAP14PF;                     //操作 FLL+
  LCDCTL = LCDON + LCD4MUX + LCDP2;         //液晶使用 4MUX 模式,使用 S0~S17 段
```



输出引脚与显示元件的连接

430 Pins		LCD Pinout				
		PIN	COM0	COM1	COM2	COM3
S0	⟷	1	1d	1e	1g	1f
S1	⟷	2	1h	1c	1b	1a
S2	⟷	3	2d	2e	2g	2f
S3	⟷	4	2h	2c	2b	2a
S4	⟷	5	3d	3e	3g	3f
S5	⟷	6	3h	3c	3b	3a
S6	⟷	7	4d	4e	4g	4f
S7	⟷	8	4h	4c	4b	4a
S8	⟷	9	5d	5e	5g	5f
S9	⟷	10	5h	5c	5b	5a
S10	⟷	11	6d	6e	6g	6f
S11	⟷	12	6h	6c	6b	6a
S12	⟷	13	7d	7e	7g	7f
S13	⟷	14	7h	7c	7b	7a
S14	⟷	15	8d	8e	8g	8f
S15	⟷	16	8h	8c	8b	8a
S16	⟷	17	9d	9e	9g	9f
S17	⟷	18	9h	9c	9b	9a
S18	⟷	19	10d	10e	10g	10f
S19	⟷	20	10h	10c	10b	10a
S20	⟷	21	11d	11e	11g	11f
S21	⟷	22	11h	11c	11b	11a
S22	⟷	23	12d	12e	12g	12f
S23	⟷	24	12h	12c	12b	12a
S24	⟷	25	13d	13e	13g	13f
S25	⟷	26	13h	13c	13b	13a
S26	⟷	27	14d	14e	14g	14f
S27	⟷	28	14h	14c	14b	14a
S28	⟷	29	15d	15e	15g	15f
S29	⟷	30	15h	15c	15b	15a
COM0	⟷	31	COM0			
COM1	⟷	32		COM1		
COM2	⟷	33			COM2	
COM3	⟷	34				COM3

显示缓存器内容

COM	3	2	1	0	3	2	1	0		
MAB 09FH	a	b	c	h	f	g	e	d	n=30	Digit 16
09EH	a	b	c	h	f	g	e	d	28	Digit 15
09DH	a	b	c	h	f	g	e	d	26	Digit 14
09CH	a	b	c	h	f	g	e	d	24	Digit 13
09BH	a	b	c	h	f	g	e	d	22	Digit 12
09AH	a	b	c	h	f	g	e	d	20	Digit 11
099H	a	b	c	h	f	g	e	d	18	Digit 10
098H	a	b	c	h	f	g	e	d	16	Digit 9
097H	a	b	c	h	f	g	e	d	14	Digit 8
096H	a	b	c	h	f	g	e	d	12	Digit 7
095H	a	b	c	h	f	g	e	d	10	Digit 6
094H	a	b	c	h	f	g	e	d	8	Digit 5
093H	a	b	c	h	f	g	e	d	6	Digit 4
092H	a	b	c	h	f	g	e	d	4	Digit 3
091H	a	b	c	h	f	g	e	d	2	Digit 2
									0	Digit 1

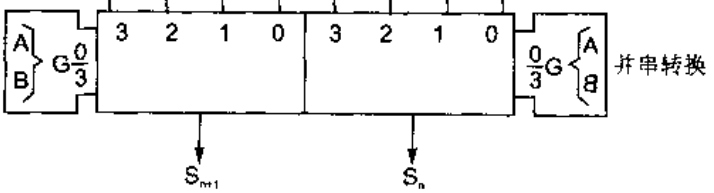


图 3.141 4MUX 方式显示缓存器中位与液晶段的对应关系示意图

第 4 章　MSP430 开发环境简介

MSP430 的开发软件较多，但常用的是 IAR 公司的集成开发环境：IAR Embedded Workbench 嵌入式工作台以及调试器 C-SPY。

4.1　Embedded Workbench（嵌入式工作台）

IAR Embedded Workbench 为开发不同的目标处理器的项目提供强有力的开发环境，并为每一种目标处理器提供工具。Embedded Workbench 使用项目模式来组织应用程序。它有如下一些特点。

（1）通用性

- 可以在 Windows 环境下运行；
- 分层的项目表示；
- 直观的用户界面；
- 工具与编辑器全集成；
- 全面的超文本帮助。

（2）编辑器

- 可以同时编辑汇编和 C 语言源文件；
- 汇编程序和 C 语言程序的句法用文本格式和颜色区别显示；
- 强有力的搜索和置换命令，而且可以多个文件搜索；
- 从出错列表直接跳转到出错的相关文件的相关语句；
- 可以设置在出错语句前标志；
- 圆括号匹配；
- 自动缩进，可以设置自动缩进的空格；
- 每个窗口的多级取消与恢复。

下面将介绍 Embedded Workbench 的安装和使用。

4.1.1　Embedded Workbench 安装

在 Windows 环境下，双击 FET_R202.EXE 或 FET_301.EXE，目前最新版本为 3.04，即 FET_304.EXE 安装过程中使用默认值。对机器没有特殊的要求，目前的计算机都能满足对内存、硬盘、机器速度的要求。安装完成以后应出现如图 4.1 所示的界面。

双击 IAR Embedded Workbench 图标，进入嵌入式工作台软件环境，可以进行程序的编辑、项目的管理、编译及连接等工作。双击 IAR CSPY Debugger 图标，进入嵌入式调试环境，可以方便地进行程序的模拟调试，还可以下载程序到具体的器件（FLASH 型）进行联机在线调试。

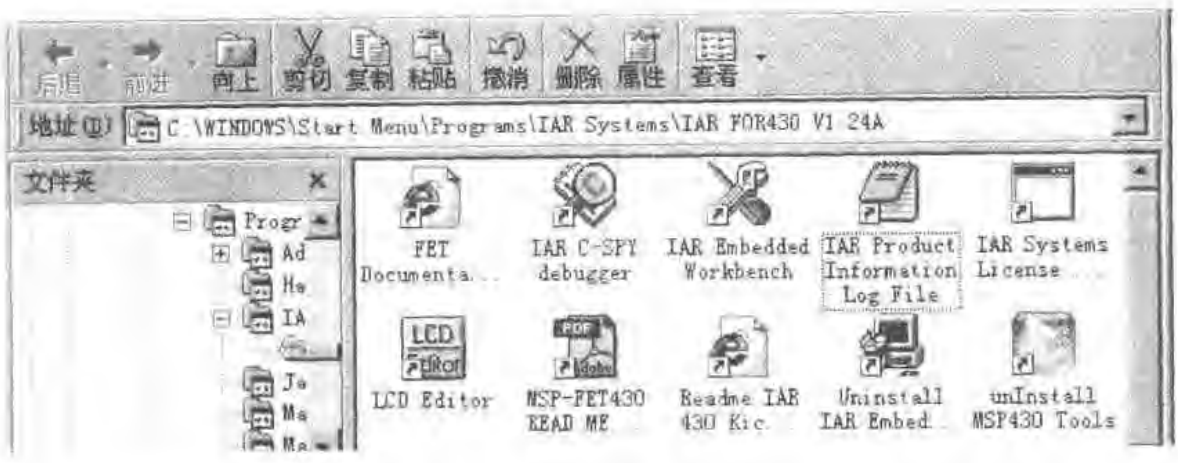


图 4.1 Embedded Workbench 安装成功图例

4.1.2 Embedded Workbench 概述

IAR Embedded Workbench 为开发各种不同的目标处理器的项目提供了强有力的开发环境。本节介绍在这个环境中使用项目模式进行典型的用户应用程序的开发。IAR Embedded Workbench 被专门设计成能适合常用的软件开发项目的组织方式。若设计者需要开发适合于不同版本目标硬件的应用程序的相应版本，而这些相应版本的应用程序又有相同的部分源文件，那么使用项目管理方式就很方便。设计者只需要维护惟一的副本，就可以对应用程序的每一个版本进行改进。Embedded Workbench 适合维护用于建造应用程序的所有版本的源文件，允许设计者以树状体系结构组织项目，并能一目了然地显示文件之间的依赖关系。

项目的树状结构如图 4.2 所示。

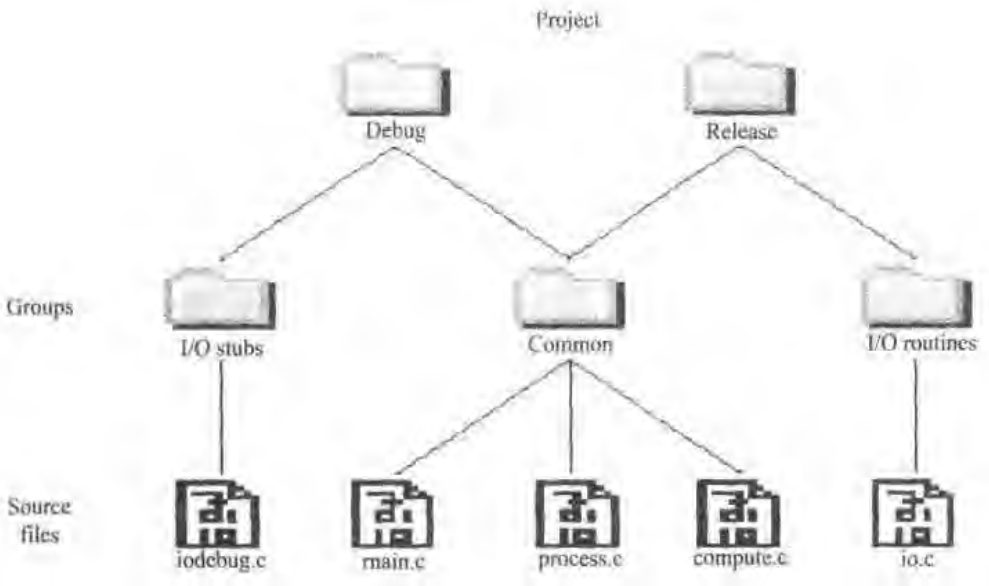


图 4.2 项目的树状组织结构

在树状结构中，目标位于最高层，它规定了设计者想要建立的应用程序的不同目标版本。对于一般的简单应用，只需要两个目标：Debug(调试)和 Release(发布)。较为复杂的项目还

会有其他的目标。每一个组将一个或多个相关的源文件组合起来，并包含于一个或多个目标之中。每一个源文件则会被包含在一个或多个组中。当设计者使用项目(Project)工作时，总有一个选定的当前目标(CURRENT TARGET)在项目窗口中，只有作为该目标成员(MEMBER)的组以及它们所包含的源文件才能被真正建立，并由此产生目标代码。

4.1.3　Embedded Workbench 使用指南

下面举一个具体的例子来说明如何使用 IAR Embedded Workbench。这个例子很简单，就是工具套件中的在 P1.0 上连接发光二极管闪烁的例子。

首先打开 IAR Embedded Workbench。在 Windows 环境下依次单击："开始"、"程序"、IAR Systems、IAR Embedded Workbench For MSP430 Kickstart、IAR Embedded Workbench，进入如图 4.3 所示的 IAR Embedded Workbench 集成环境。

图 4.3　第一次进入 Embedded Workbench

然后在该环境下建立一个项目。单击 File，New 后出现如图 4.4 所示界面。选择 Project，单击"确定"，进入如图 4.5 所示界面。选择目标 CPU、输入项目名称，单击"确定"保存到所选择的路径。

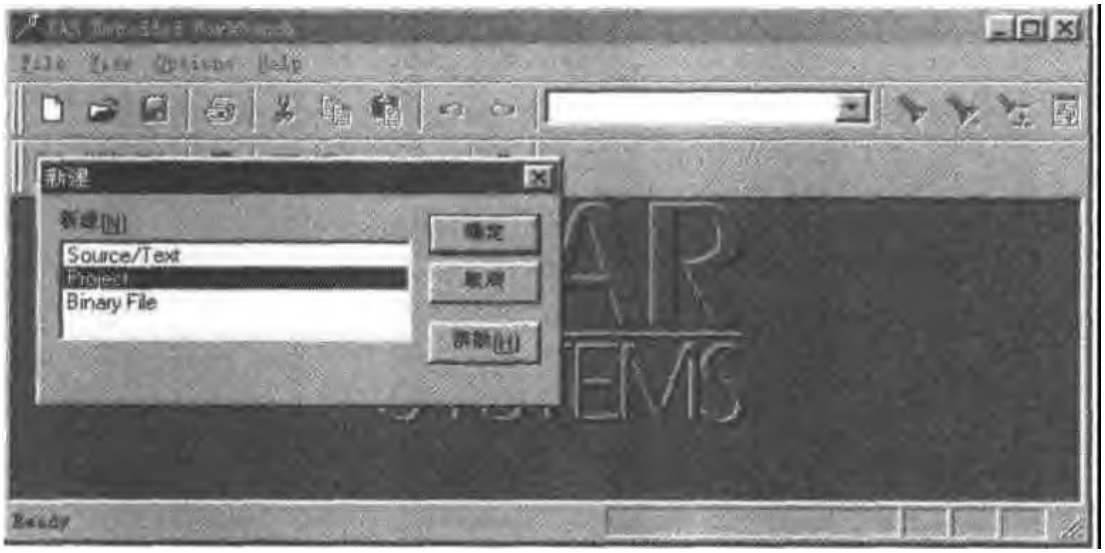


图 4.4　新建一个项目的界面

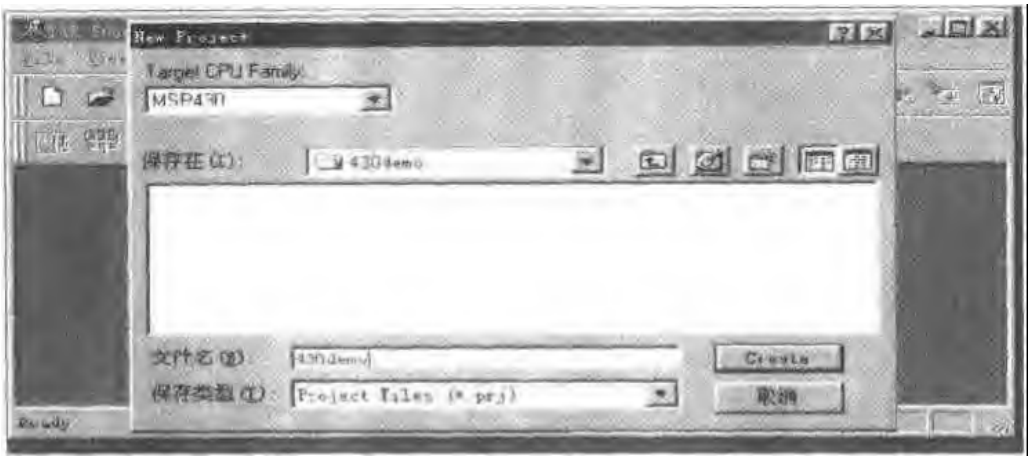


图 4.5 选择目标 CPU 并输入项目名称

此时可以看到，在 430demo 项目下有一个 Debug 目标，但没有源文件，如图 4.6 所示。单击 File.New 进入如图 4.7 所示界面。选择 Source/Text 选项，单击“确定”进入源程序编辑界

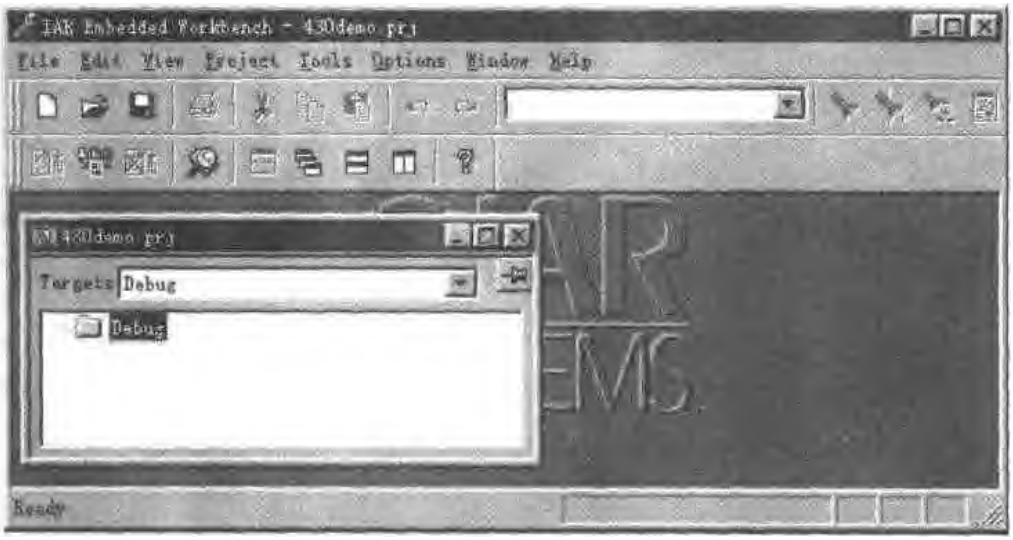


图 4.6 430demo 项目对话框

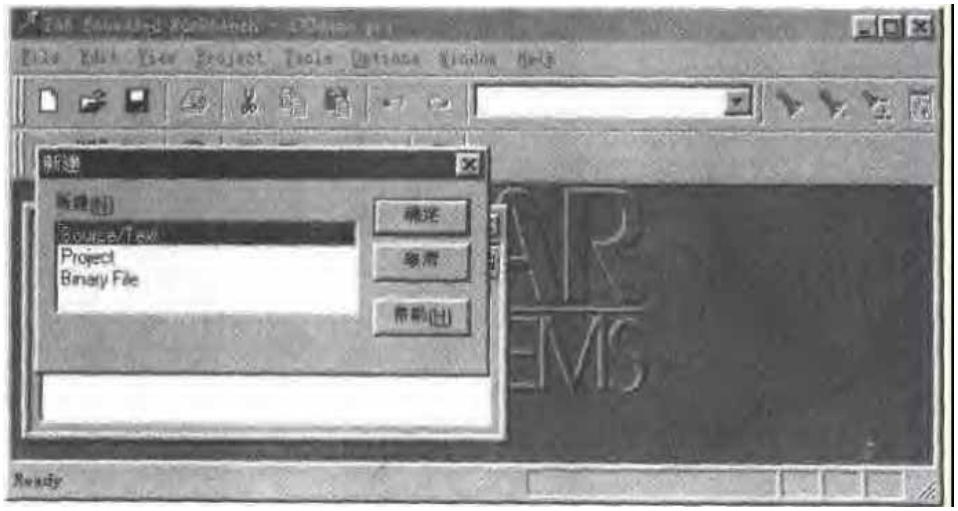


图 4.7 进入编写源程序

面，如图 4.8 所示。在这里进行源程序的编辑。源程序编辑好之后，保存为 430led.s43。然而此时的项目 430demo 与源程序 430led.s43 是孤立的，没有联系。

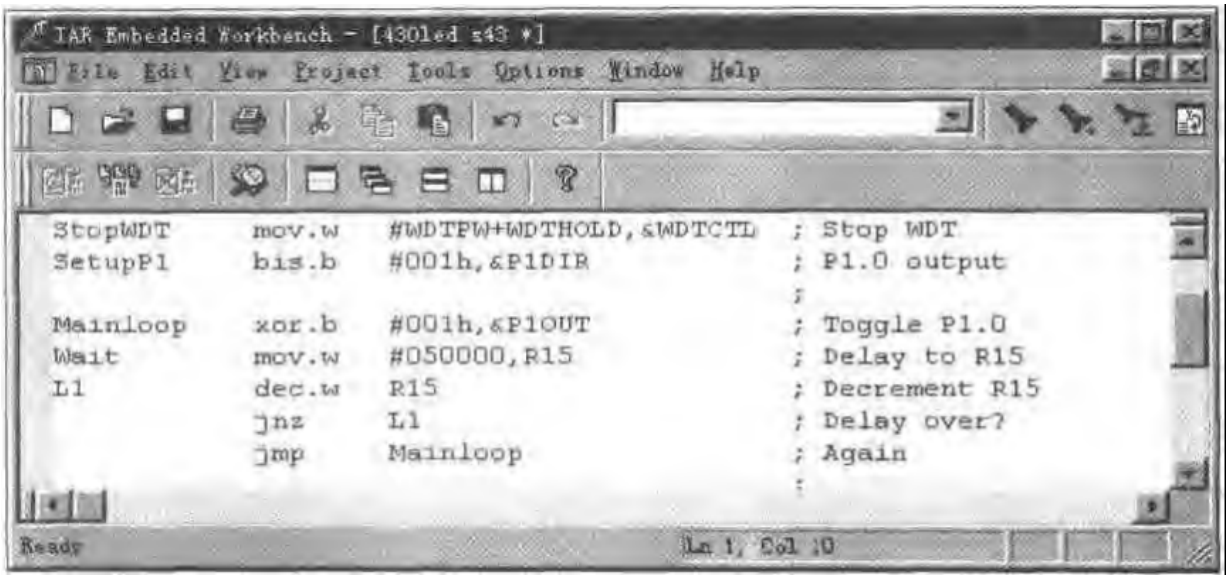


图 4.8　源文件的编辑

单击 Project，Files 出现如图 4.9 所示界面。选择所编辑的汇编源程序，单击 Add 按钮将源文件加入组中，在组里的文件框就可以看到所选的文件出现在里面，如图 4.10 所示。

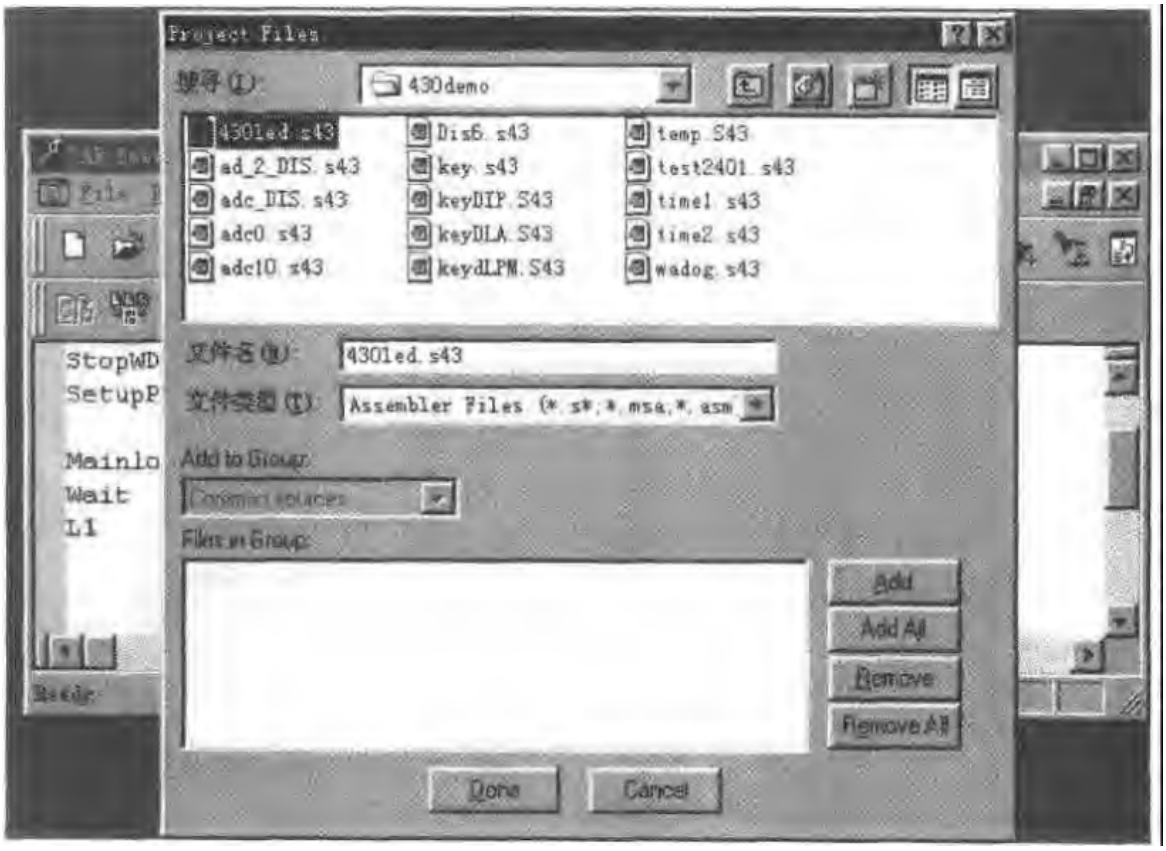


图 4.9　将源文件加入组(Group)

图 4.10 单击 Add 按钮加入源文件

这样 430demo 项目里就有了 430led.s43 程序，编译、连接之后产生的目标代码就是 430led.s43 汇编后的代码。单击 Done 按钮后，可以看到项目管理窗口（如图 4.11）与图 4.6 明显不同：图 4.6 中 Debug 目标里没有组，也没有源文件；而在图 4.11 中的 Debug 目标里有组 Common Sources，也有源文件 430led.s43。

图 4.11 源文件已经加入项目中

程序清单如下：

```
            ORG     0F000h                          ;
RESET       mov.w   #300h,SP                        ;初始化堆栈指针
StopWDT     mov.w   #WDTPW+WDTHOLD,&WDTCTL          ;停止看门狗
SetupP1     bis.b   #001h,&P1DIR                    ;P1.0 为输出
Mainloop    xor.b   #001h,&P1OUT                    ;P1.0 求反
Wait        mov.w   #050000,R15                     ;延时初值
L1          dec.w   R15                             ;
            jnz     L1                              ;
            jmp     Mainloop                        ;反复执行
            ORG     0FFFEh                          ;430 的第一条指令位置
            DW      RESET                           ;
            END
```

单击 Project.compile 或按 Ctrl+F9 键，或单击按钮进行文件编译或汇编，出现如图 4.12 所示的错误提示。

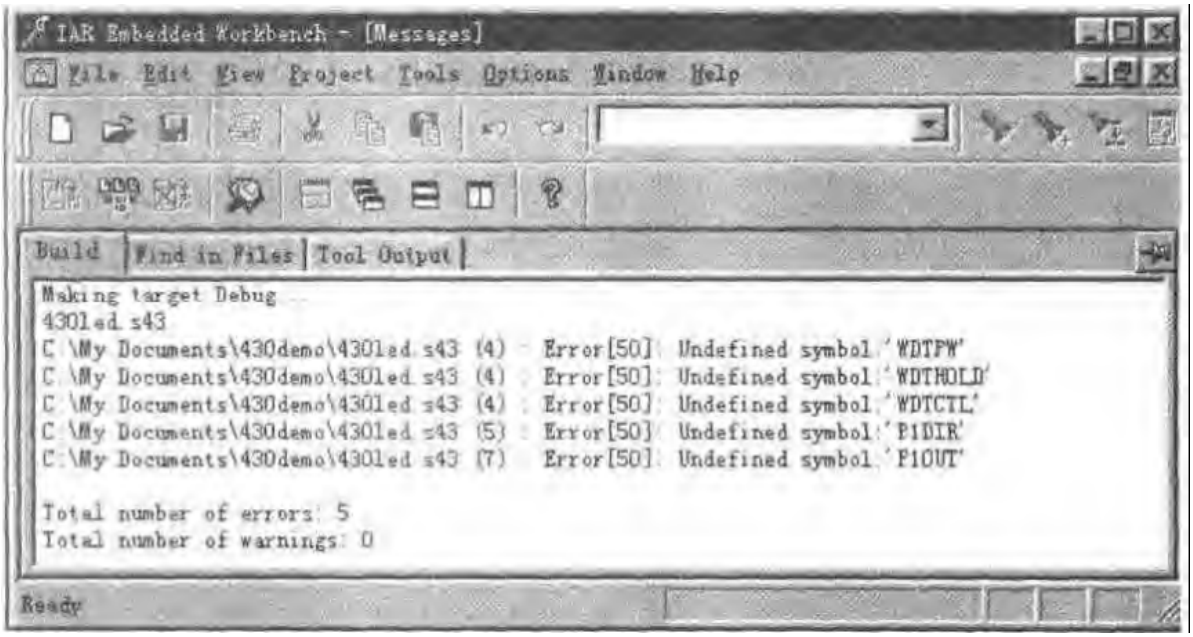


图 4.12　汇编之后产生的错误

任意单击某一个错误提示，系统会自动给出有错误的语句行，如图 4.13 所示。

仔细阅读错误提示，会发现是一些特殊功能寄存器没有定义。在源文件的开始加一句

```
#include  "msp430x11x.h"
```

如图 4.14 所示，再编译则没有错误了，因为在"msp430x11x.h"等的包含文件中已经有关于器件中的特殊功能寄存器的地址说明。

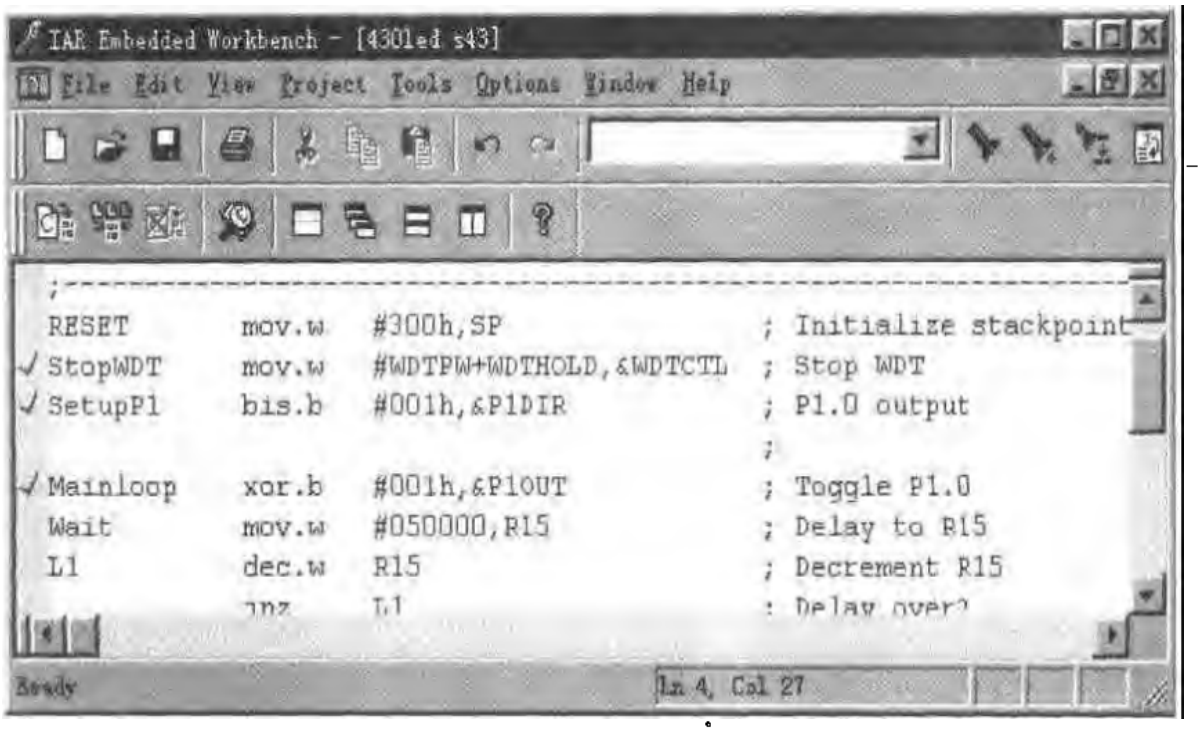


图 4.13 系统自动指出有错的语句行

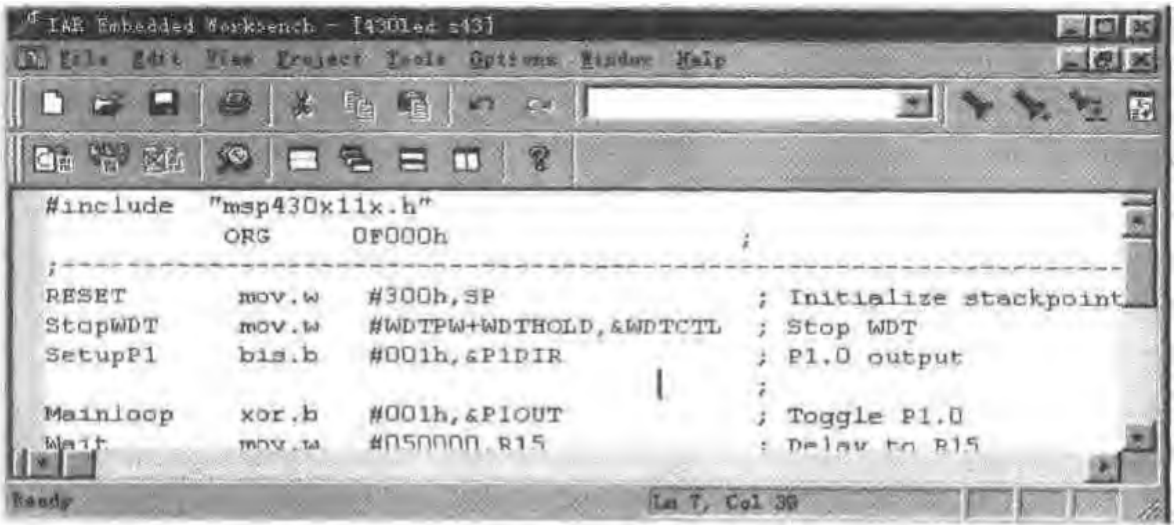


图 4.14 修改后的源文件

源文件编译通过之后，将生成目标代码。这时可以单击 Projec,Make，或按 F9 键，或单击 按钮，进行连接以生成目标代码，出现如图 4.15 所示的错误提示。这是因为环境的设置不对，单击 Project,Options 进入设置界面后，单击 XLINK 进入如图 4.16 所示界面。

在这些设置中，最关心的是 Include 选项。其中 Include 路径可以用默认的，不用改动；但

XCL file name 则需要改动。

首先选中 Override default，然后根据设计者所使用的目标处理器(MPU/CPU)选择相关的 XCL 文件名。在这个设计中用的是 MSP430F1121，而且是汇编语言，所以这里选择 msp430F1121A. xcl，如图 4.17 所示；但如果用的是 C 语言，则选择 msp430F1121C. xcl。设置 XLINK 后单击 OK 按钮以保存设置，如图 4.18 所示。这时再单击 Project，Make 或按 F9 键进行编译连接，则完全通过。

最后对所编程序进行调试，以验证设计的正确。

图 4.15　连接后出现的错误

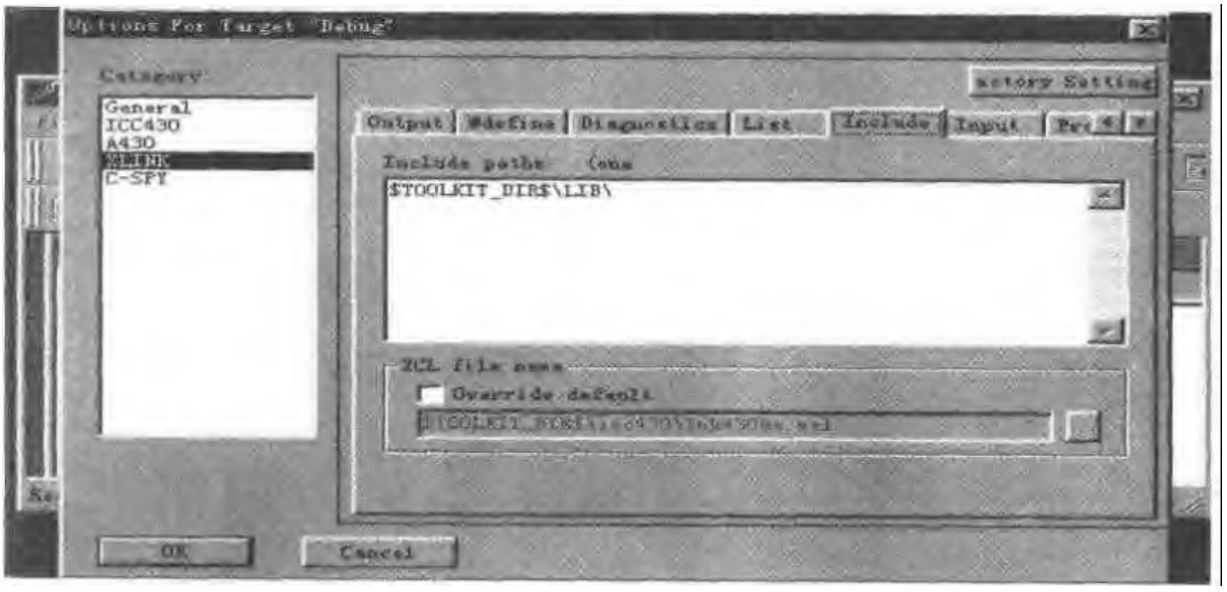


图 4.16　环境设置界面

单击 Project.Options 进入设置后，单击 CSPY 进入如图 4.19 所示界面。在驱动选项中有 3 个选项，Simulator/Flash Emulation Tool/ROM-Monitor。它们分别是软件模拟/Flash 型仿真工具/ROM 监控方式。一般情况下，如果使用的是 Flash 工具套件，如德州仪器的 MSP-FETP430P140，MSP-FETP430P110，MSP-FETP430P410 或笔者自己设计的系列 430Flash 工具套件，须选择 Flash Emulation Tool 选项作为驱动；如果使用软件模拟则选择 Simulator。

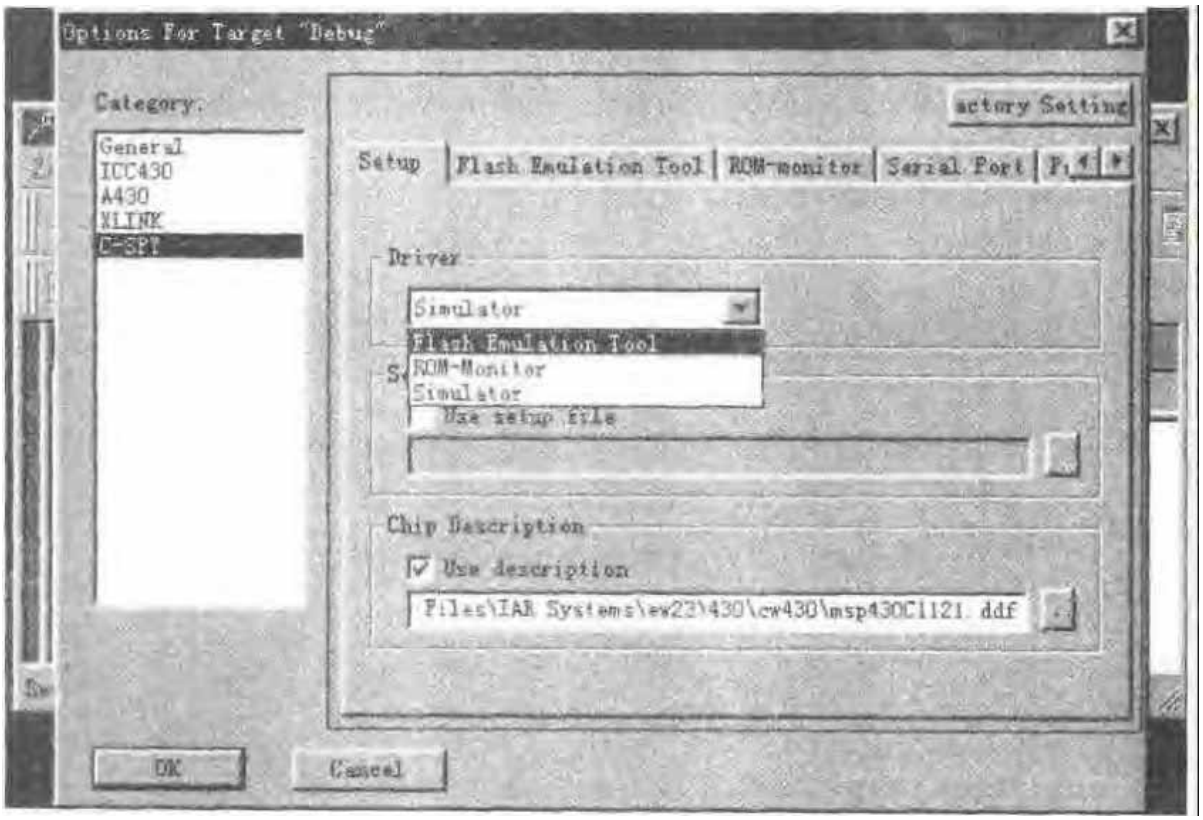


图 4.19　设置 CSPY 参数

在晶片描述（Chip Description）一栏正确选择所使用的芯片，如本例使用的是 MSP430F1121，那么就选择 MSP430F1121.DDF 文件，注意文件的路径。在进入调试环境之后，芯片的情况就按这个文件进行描述，主要是器件的 0000H～01FFH 之间的功能寄存器的名称描述。

这些设置好之后单击 OK 按钮保存。

4.1.4　Embedded Workbench 综述

File(文件)菜单　提供打开项目和源文件、保存、打印以及退出 Embedded Workbench 的命令。

Edit(编辑)菜单　提供在编辑窗口中编辑和搜索的命令。

View(视图)菜单　此菜单提供的命令允许用户改变显示在 Embedded Workbench 窗口中的信息。

Project(项目)菜单　提供将文件添加到项目、创建组及在当前项目上运行 IAR 工具的

命令。

Tools(工具)菜单　此菜单是用户可配置的菜单,用户可将与 Embedded Workbench 一起使用的工具添加到此菜单中。

Options(选项)菜单　此菜单允许用户定制 Embedded Workbench,对各种环境参数进行满足用户要求的配置。

Windows(窗口)菜单　此菜单上的命令允许用户管理 Embedded Workbench 的窗口,并改变它们的屏幕排列:用户可以打开需要的工作窗口,并随意在屏幕上排列打开的窗口。

Help(帮助)菜单　提供此工作环境的帮助。

IAR Embedded Workbench 快捷按钮如图 4.20 所示,左边为文件的"建立"、"打开"、"保存"、"打印"及文件内的"剪切"、"拷贝"、"粘贴"等操作,右边为"查找"、"替换"等与整个环境的界面控制操作等以及快捷的帮助按钮。通过这些快捷按钮可以方便地进行操作,主要是文件的编辑和项目的组织。

图 4.20　IAR Embedded Workbench 快捷按钮

4.2　CSPY 使用指南

在 IAR Embedded Workbench 中可以方便地进入 CSPY 调试环境,但必须在设计的程序通过了 Make("编译"、"汇编"、"连接"),生成目标代码之后。有 3 种方式可以进入 CSPY 调试环境,分别是:在 Embedded Workbench 中单击 Project,Debugger;在快捷按钮中单击 按钮;在 Windows 环境下依次单击"开始"、"程序"、IAR Systems、IAR Embedded Workbench For MSP430 Kickstart、IAR CSPY debugger。使用上述 3 种方式之一便可进入如图 4.21 所示的集成环境。

该环境的最顶上为系统主菜单(下拉式),紧接着为快捷按钮(见图 4.22 和图 4.23),最主要部分为若干打开的窗口:源程序窗口、寄存器窗口、存储器窗口、观察窗口等。在这里可以非常方便地调试用户所设计的程序。

在 CSPY 调试环境中,经常用到的是与程序的执行和与窗口有关的命令,图 4.22 为常用的与窗口有关的命令,图 4.23 为常用的调试工具。

在 CSPY 环境中,可以打开调试程序所需的若干窗口:源程序窗口、寄存器窗口、观察窗口、存储器窗口、特殊功能寄存器窗口等。

这些窗口在集成环境中可以多种排列方式显示,如图 4.24 中的 Window 菜单所示。图 4.21 为常用的一种显示方式。

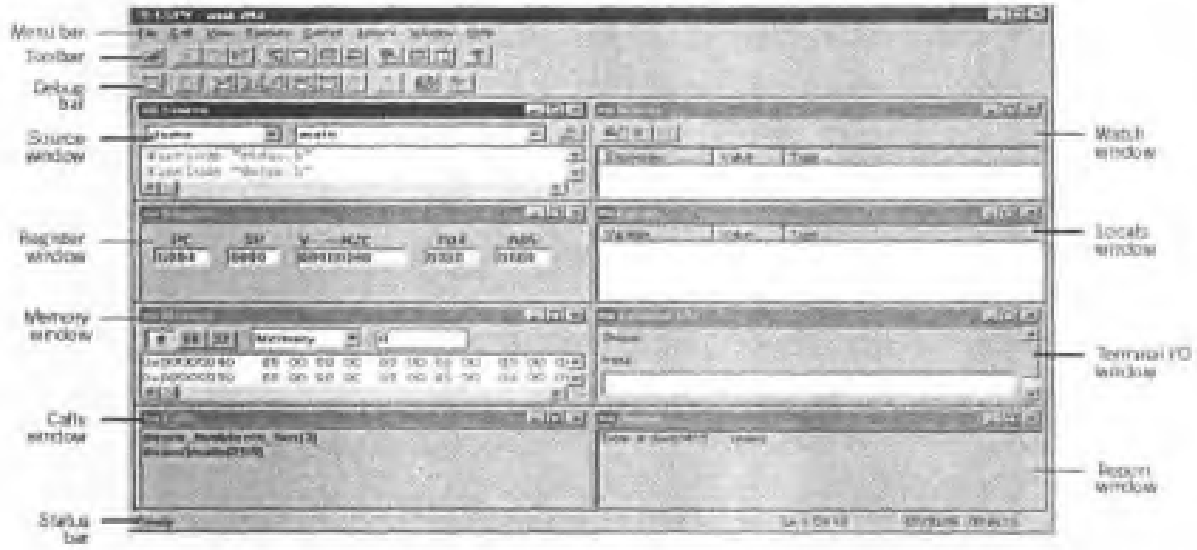


图 4.21 CSPY 环境简介

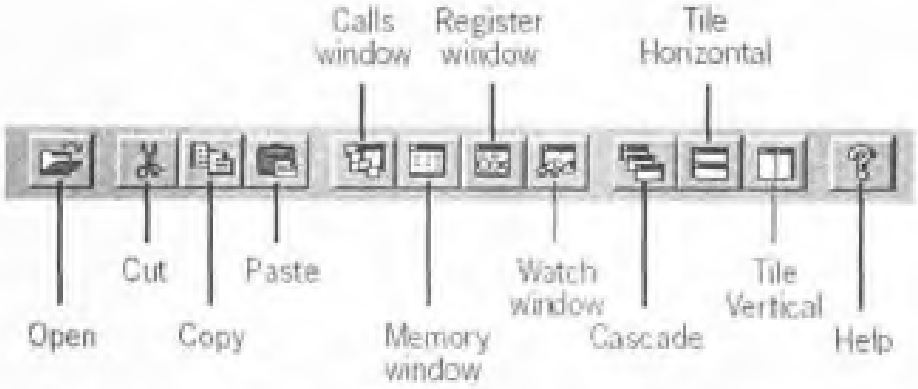


图 4.22 CSPY 快捷按钮之一

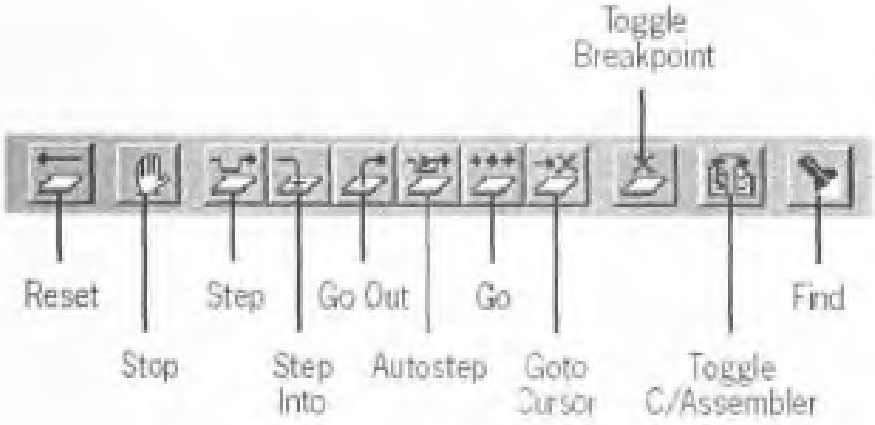


图 4.23 CSPY 快捷按钮之二

码一指令。而且可以看出每一条指令都开始于偶地址处。

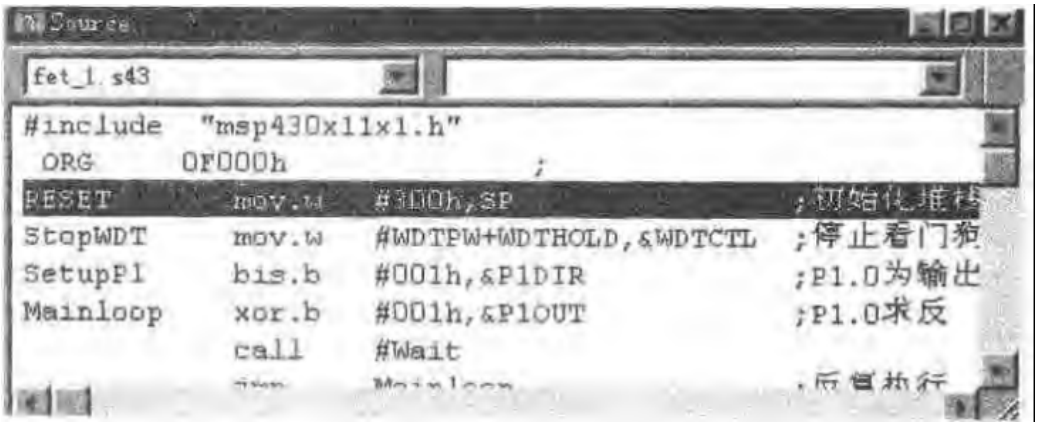


图 4.26　源程序窗口

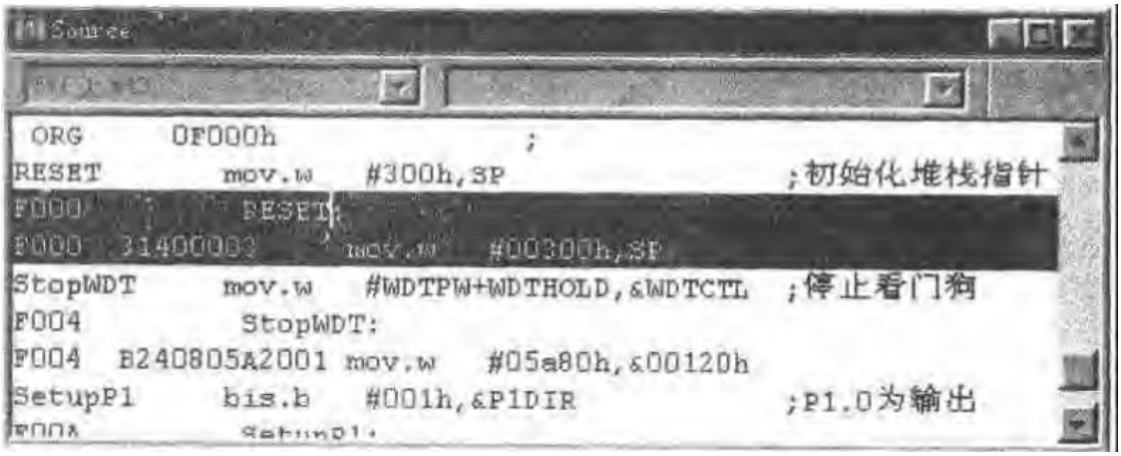


图 4.27　源程序/代码窗口

3. 寄存器窗口

寄存器窗口是调试程序另一个常用的窗口，如图 4.28 所示。其打开方式为：单击图 4.24 中的 Register，或单击图 4.22 中的 Register window。其中：R0～R2 分别是前 3 个 PC(程序计数器)、SP(堆栈指针)、SR(状态寄存器，图中看到的是状态寄存器的各位名称与对应位的值)；R3 为常数发生器，模拟指令使用，这里用户看不到；R4～R15 为用户使用，可以查看与修改；CYCLES 为程序执行所用的机器周期数，通过它可以方便地知道执行代码的时间。

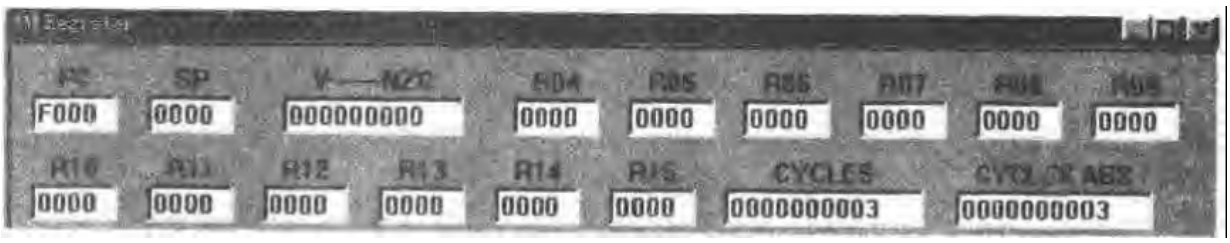


图 4.28　寄存器窗口

4. 特殊功能寄存器窗口

特殊功能寄存器窗口也是调试程序一个常用的窗口，如图 4.29 所示。其打开方式为：单

击图 4.24 中的 SFR。这个窗口有一些选项，图中选的是 Ports，所以打开的是与端口有关的一些特殊功能寄存器，同样可以打开其他一些相关的特殊功能寄存器。

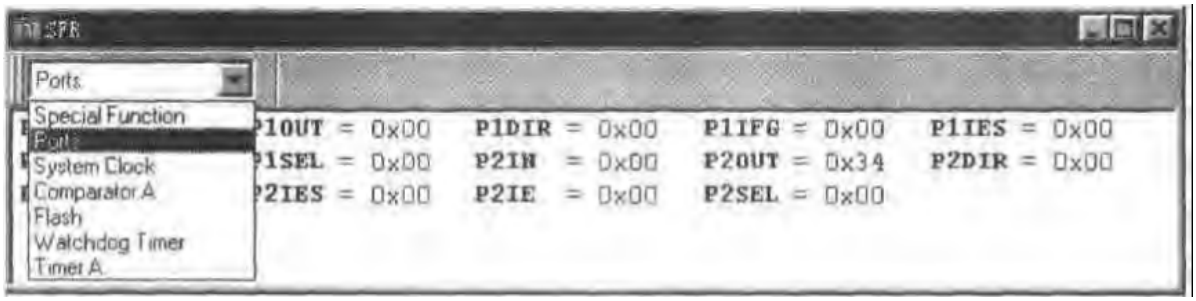


图 4.29 特殊功能寄存器窗口

5. 观察窗口

单击图 4.24 中的 Watch，可以打开如图 4.30 所示的观察窗口。这里可查看用户想知道的变量的值，左边是变量名称，右边是变量的数值。已经有 3 个变量，要添加其他变量，可在下面的虚线框内右击，再单击 Add，然后输入变量名称；或直接在源程序中要观察的变量处右击，再击 Quick Watch。观察窗口中不想查看的变量名称也可方便地删除，在不想查看的变量处右击，再单击 Remove。

图 4.30 观察窗口

4.3 汇编程序调试举例

以前面在 Embedded Workbench 环境中写的程序为例，说明如何在 CSPY 环境中调试。源程序如下：

```
ORG     0F000h                                  ;
RESET       mov.w   #300h,SP                    ;初始化堆栈指针
StopWDT     mov.w   #WDTPW+WDTHOLD,&WDTCTL      ;停止看门狗
SetupP1     bis.b   #001h,&P1DIR                ;P1.0 为输出
Mainloop    xor.b   #001h,&P1OUT                ;P1.0 求反
Wait        mov.w   #050000,R15                 ;延时初值
L1          dec.w   R15                         ;
            jnz     L1;
            jmp     Mainloop                    ;反复执行
            ORG     0FFFEh                      ;430 的第一条指令位置
            DW      RESET                       ;
            END
```

MSP430 的数据与程序存储器为线性统一编址。

程序的第一句

```
ORG     0F000h
```

表明程序代码应放在 0F000H 开始的空间，如图 4.31 所示。

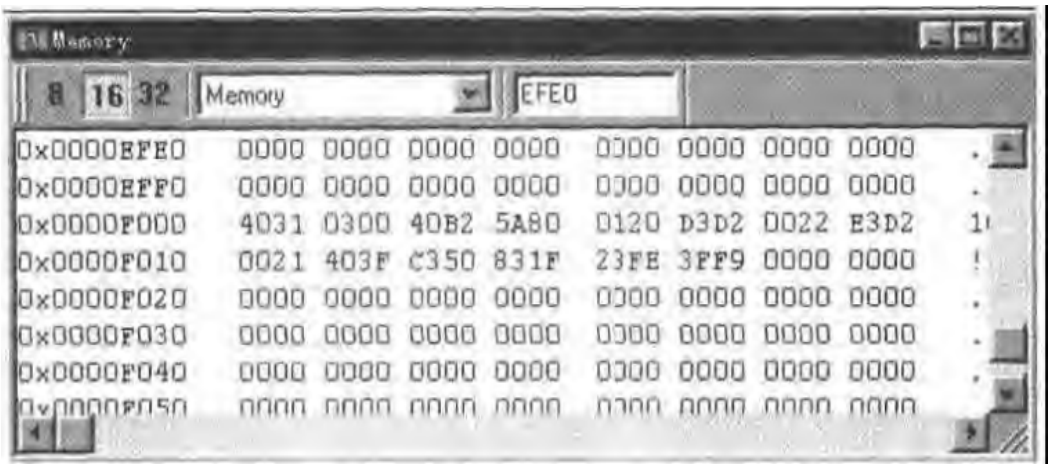


图 4.31　程序代码

程序的后两句

```
ORG     0FFFEh
DW      RESET
```

指示程序复位之后应该执行的第一条指令的位置，RESET 是一个地址标号 0F000H，这个数据在地址 0FFFEH 中，如图 4.32 所示。在这个例子里用的是 430F1121 芯片，它的程序存储器的地址为 0F000H～0FFFFH，其中 0FFE0H～0FFFFH 为中断向量，而地址 0FFFEH 处的中断向量为器件复位之后执行的第一条语句的物理地址，将它放在 0F000H 处，故在 0FFFEH 处的数据为 0F000H。

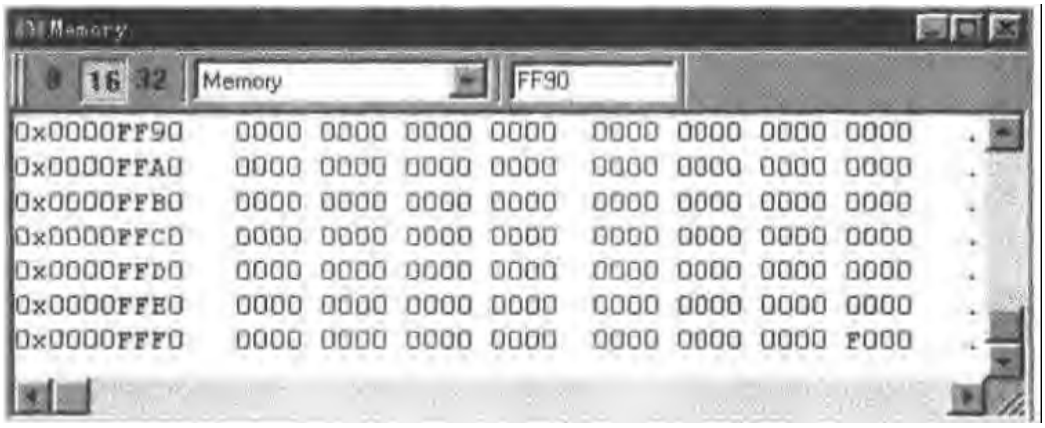


图 4.32　MSP430F1121 复位中断向量的内容 0F000H

在这里有几种方式调试程序：单步、断点、连续、运行到光标等。下面分别阐述。

(1) 单步方式

先单步执行第一句

```
mov. w    #300h,SP
```

单步运行程序一定要使之处于非实时状态，单击 CONTROL，再取消 REALTIME，这样就能单步运行程序了。单步执行可单击[redacted]按钮，或依次单击 EXECUTE，STEP 或按 F2 键，执行之后 SP 的内容变为 300H。为了查看，可以单击[redacted]按钮或依次单击 Window，Register，打开如图 4.33 所示寄存器窗口。图中，SP 的内容发生了变化，而且为红色，表示刚刚改变；而源程序指示到下一句，表示将要执行的语句；在 PC 的内容上也发生了变化，指向了 0F004H。

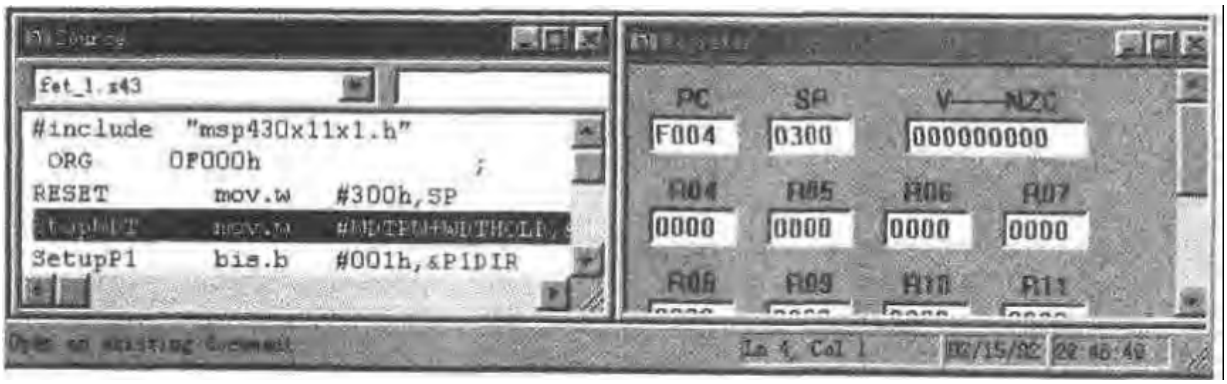


图 4.33 第一句执行后的情况

继续使用单步执行第二句。第二句为对看门狗的寄存器操作，可以依次单击 Window，SFR，打开寄存器窗口，然后找到看门狗定时器。这时就可以观察 WDTCTL 的内容，执行之后如图 4.34 所示，同时 PC 指向下一句。

图 4.34 第二句执行后的情况

继续使用单步执行第三、四两句。这两句为对端口操作，同样依次单击 Window，SFR，打开寄存器窗口，然后找到 Ports。这时就可以观察端口 1、端口 2 的全部内容，执行之后如图 4.35所示，红色表示刚刚改变，同时 PC 指向下一句。

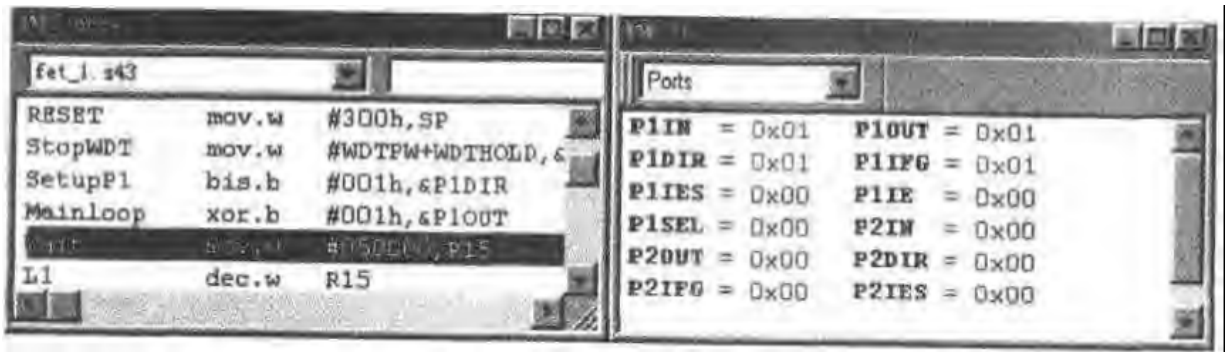


图 4.35 第四句执行后的情况

(4) 运行到光标

运行到光标处时先要取消断点:与设置断点一样按 F5 键。取消断点之后,将光标移到想要运行到的语句条,再单击按钮或依次单击 EXECUTE,GO TO CURSOR。

在程序中的延时用的是软件的方法:执行若干条需要一定时间的语句。这里用的是循环,R15 设置一初始值,再减 1 判断是否减完。为了比较,先记录下复位时的情况,如图 4.38 所示;再运行

```
Wait    mov.w     #050000,R15
```

这一句执行之后看现场的情况,如图 4.39 所示。R15 已经是程序所赋予的数据 5 0000,即十六进制的 0C350H。这一句执行的是数据传送操作,状态标志不受影响,Z=1 是以前的结果。紧接着是一句减 1 操作,执行之后如图 4.40 所示。图中,状态标志有 3 位发生了变化:N,Z,C;程序将根据 Z 的取值情况来决定程序的取向,0C350H 减 1 后为 0C34FH,而 Z=0,执行下一句之后将会跳转,继续减 1 判断。直到 R15 中的数据为 0 时,状态寄存器中的标志位 Z=1,jmp L1 语句执行的结果为不跳转到 L1 处,而将执行其下一句,如图 4.41 所示。21 个机器周期是循环之前用掉的,实际上循环体用掉了 150 000 个机器周期。150 000 T 就是发光二极管闪烁的延时时间(亮与熄之间的时间间隔)。

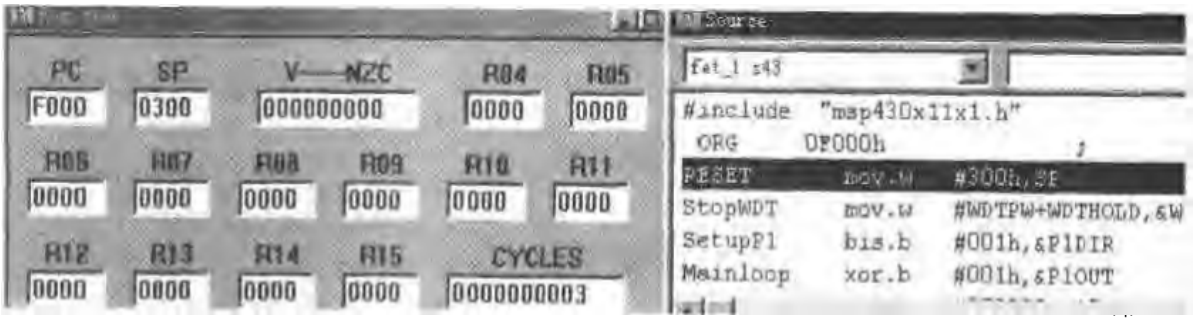


图 4.38 复位时的情况

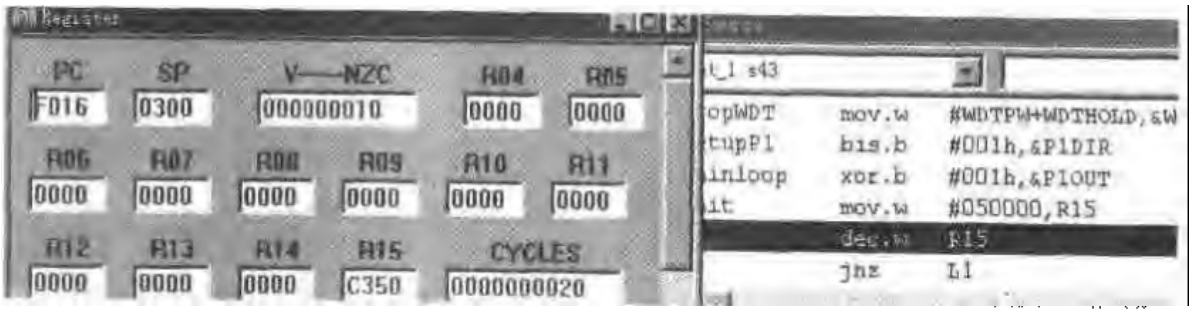


图 4.39 执行 Wait 语句行之后

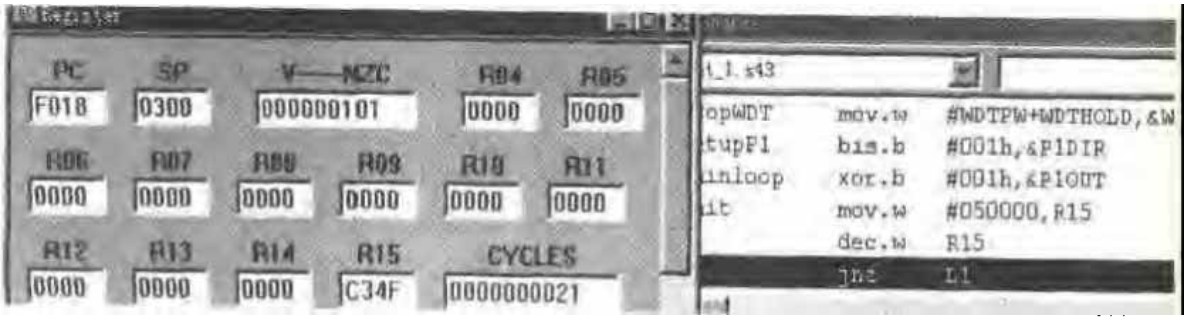


图 4.40 执行减 1 操作之后

```
#define BIT2        0x0004
#define BIT3        0x0008
#define BIT4        0x0010
#define BIT5        0x0020
#define BIT6        0x0040
#define BIT7        0x0080
#define BIT8        0x0100
#define BIT9        0x0200
#define BITA        0x0400
#define BITB        0x0800
#define BITC        0x1000
#define BITD        0x2000
#define BITE        0x4000
#define BITF        0x8000
```

源程序编辑好之后，再将它加入项目、组，进行编译、连接。连接未通过，出错信息提示如图 4.42 所示。这是因为连接命令的参数设置不对，设置为如图 4.43 所示的环境参数即可。这时编译、连接全通过。CSPY 中用的器件设置与 4.2 节一样。单击按钮之后进入调试环境，运用单步、断点、连续等手段调试，查看执行情况也与汇编程序一样，这里不再赘述。

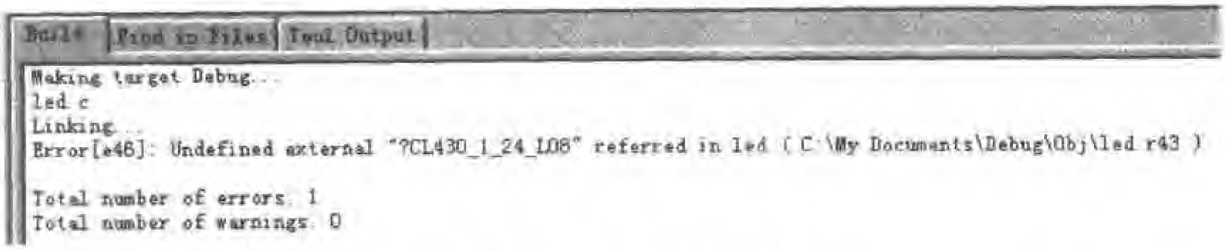


图 4.42　连接后的出错信息

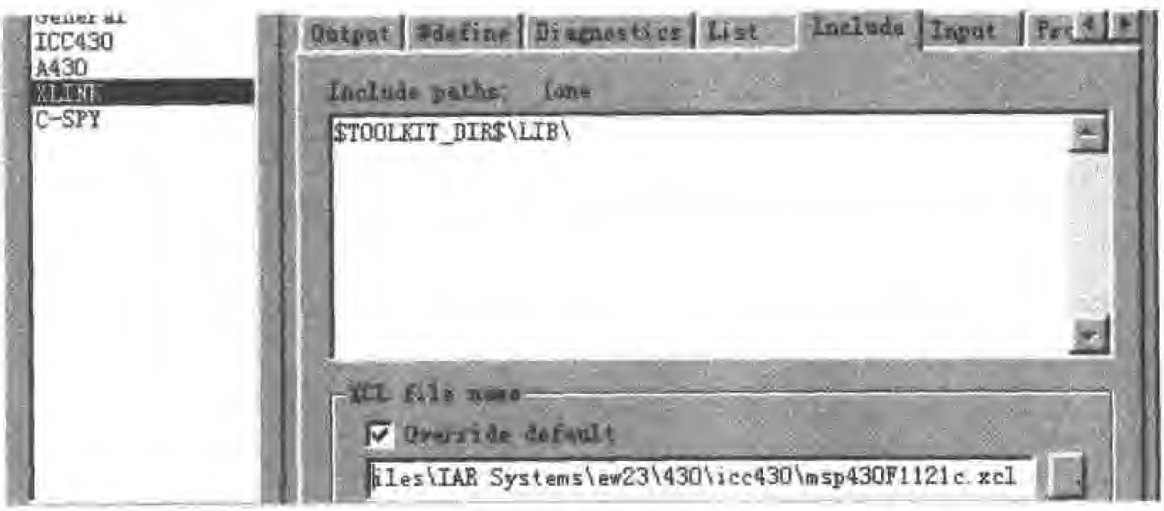


图 4.43　连接命令的参数设置

对于 C 语言的调试，还提供了一个非常方便的手段：除了使用 Watch（观察窗口）外，Locals（局部变量观察窗口）的使用也能给调试带来方便。下面还以实现在 P1.0 上的发光二

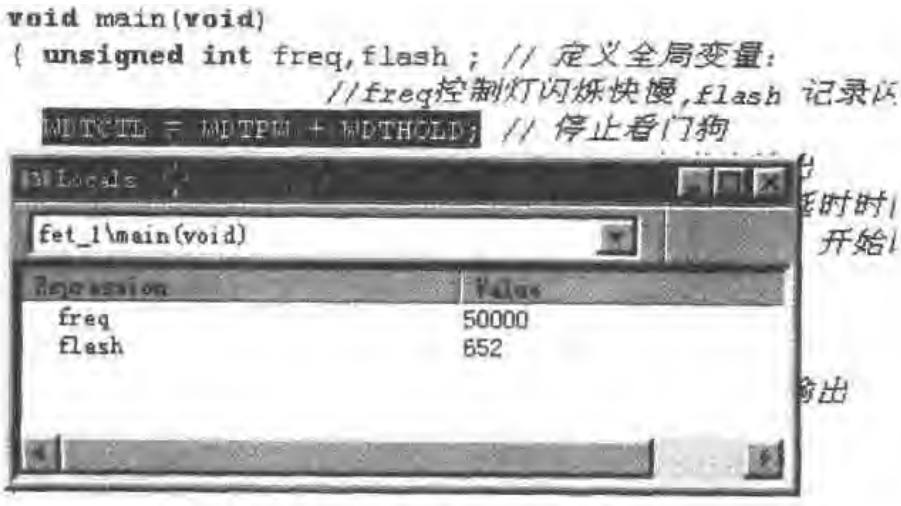


图 4.44　复位时的 Locals 窗口

而当程序执行到子程序时，该窗口的内容会发生改变，变为正在执行的子程序中的变量及其值，如图 4.45 所示。

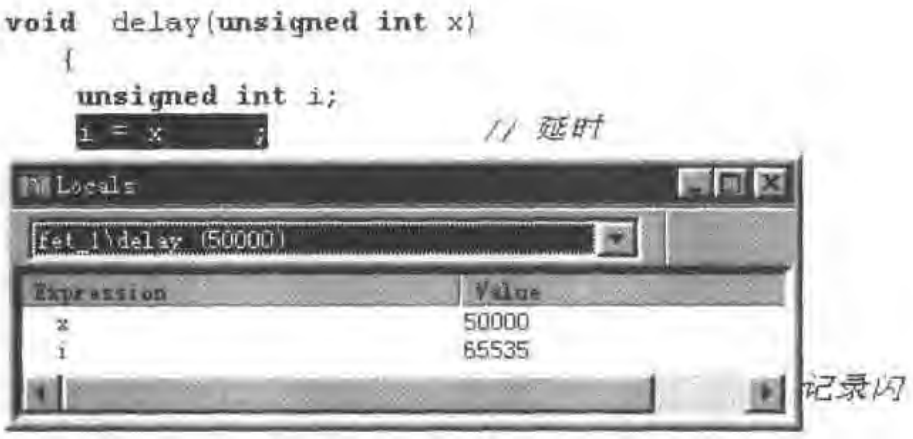


图 4.45 执行到 delay 子程序时的 Locals 窗口

因此，使用 Locals 窗口对 C 语言程序进行调试是相当方便的。

第 5 章　MSP430 单片机的应用

本章将通过大量丰富详实的具体实例来说明 MSP430 单片机的应用。这些实例都是在笔者设计的 MSP430 实验开发板上调试通过的，读者可直接借鉴、修改或移植。相关的程序将会放在笔者的个人主页(http://www.mcu-china.com)上。

本章分为 3 部分：基础应用部分，介绍一些较为单元性的软硬件设计，其中大部分的片内硬件应用在第 3 章已经有较多的具体实例，这里将着重讲解软件方面以及片外硬件方面的应用设计；综合应用部分，一些较为综合的应用，往往一个应用包含若干单元；系统应用部分，将举几个详细完整的实例。

5.1　基础应用部分

5.1.1　MSP430 头文件

MSP430 系列的每一型号都有大量的片内外围模块，而这些模块都通过大量的控制寄存器来控制，每一个寄存器有它的物理地址，对它们的使用实质是直接操纵(读写)这些地址。但大量的寄存器很难记住它们对应的物理地址，MSP430 的 IAR 调试环境提供了各种系列的标准头文件，这些文件已经定义好各寄存器的相应物理地址，只要在程序中使用寄存器名以及寄存器中的控制位名即可。如在端口 P1.1 输出高电平，用下列源程序即可实现：

(1) 汇编程序

```
MOV.B    #02H , &22H
MOV.B    #02H ,&21H
```

也可以使用下列语句段：

```
#include   "msp430x11x1.h"
MOV.B    #BIT1 , &P1DIR
MOV.B    #BIT1 , &P1OUT
```

(2) C 语言程序

```
#include   "msp430x11x1.h"
P1DIR =0x02;
P1OUT=0x02;
```

以上 3 段程序都能实现在端口 P1.1 输出高电平。第 1 段 P1DIR 的地址是 22H，P1OUT 的地址是 21H，则两句执行之后得到结果。第 2 段开始包含了头文件 msp430x11x1.h，在后面两句中就直接使用寄存器名，还有 BIT1 在头文件中也有定义，# define BIT1 (0x0002)，所以这里的两条语句与第 1 段的语句实质上是完全相同的。第 3 段为 C 语言程序，寄存器的定义同样在头文件 msp430x11x1.h 中。表 5.1 列出 msp430x11x1.h 文件的内容，以供读者方便查阅，也可在软件的 INC 子目录下查找。在本书中对寄存器的使用都直接使用头文件中的定义。

表 5.1 msp430x11xt.h 头文件内容

头文件内容	头文件内容
#ifndef __msp430x11x #define __msp430x11x #define BIT0 (0x0001) #define BIT1 (0x0002) #define BIT2 (0x0004) #define BIT3 (0x0008) #define BIT4 (0x0010) #define BIT5 (0x0020) #define BIT6 (0x0040) #define BIT7 (0x0080) #define BIT8 (0x0100) #define BIT9 (0x0200) #define BITA (0x0400) #define BITB (0x0800) #define BITC (0x1000) #define BITD (0x2000) #define BITE (0x4000) #define BITF (0x8000) #define C (0x0001) #define Z (0x0002) #define N (0x0004) #define V (0x0100) #define GIE (0x0008) #define CPUOF (0x0010) #define OSCOF (0x0020) #define SCG0 (0x0040) #define SCG1 (0x0080) #ifndef __IAR_SYSTEMS_ICC #define LPM0 (CPUOFF) #define LPM1 (SCG0+CPUOFF) #define LPM2 (SCG1+CPUOFF) #define LPM3 (SCG1+SCG0+CPUOFF) #define LPM4 (SCG1+SCG0+OSCOFF+CPUOFF) #define LPM0_bits (CPUOFF) #define LPM1_bits (SCG0+CPUOFF) #define LPM2_bits (SCG1+CPUOFF) #define LPM3_bits (SCG1+SCG0+CPUOFF) #define LPM4_bits (SCG1+SCG0+OSCOFF+CPUOFF) #include <In430.h>	#define LPM0 _BIS_SR(LPM0_bits) #define LPM0_EXIT _BIC_SR_IRQ(LPM0_bits) #define LPM1 _BIS_SR(LPM1_bits) #define LPM1_EXIT _BIC_SR_IRQ(LPM1_bits) #define LPM2 _BIS_SR(LPM2_bits) #define LPM2_EXIT _BIC_SR_IRQ(LPM2_bits) #define LPM3 _BIS_SR(LPM3_bits) #define LPM3_EXIT _BIC_SR_IRQ(LPM3_bits) #define LPM4 _BIS_SR(LPM4_bits) #define LPM4_EXIT _BIC_SR_IRQ(LPM4_bits) #endif #define IE1_ (0x0000) sfrb IE1 = IE1_; #define WDTIE (0x01) #define OFIE (0x02) #define NMIIE (0x10) #define ACCVIE (0x20) #define IFG1_ (0x0002) sfrb IFG1 = IFG1_; #define WDTIFG (0x01) #define OFIFG (0x02) #define NMIIFG (0x10) #define WDTCTL_ (0x0120) sfrw WDTCTL = WDTCTL_; #define WDTIS0 (0x0001) #define WDTIS1 (0x0002) #define WDTSSEL (0x0004) #define WDTCNTCL (0x0008) #define WDTTMSEL (0x0010) #define WDTNMI (0x0020) #define WDTNMIES (0x0040) #define WDTHOLD (0x0080) #define WDTPW (0x5A00) #define WDT_MDLY_32 (WDTPW+WDTTMSEL+WDTCNTCL) #define WDT_MDLY_0_5

续表 5.1

头文件内容	头文件内容
(WDTPW + WDTTMSEL + WDTCNTCL + WDTIS1)	#define P1IFG_ (0x0023)
#define WDT_MDLY_0_064	sfrb P1IFG = P1IFG_;
(WDTPW + WDTTMSEL + WDTCNTCL + WDTIS1 + WDTIS0)	#define P1IES_ (0x0024)
#define WDT_ADLY_1000	sfrb P1IES = P1IES_;
(WDTPW + WDTTMSEL + WDTCNTCL + WDTSSEL)	#define P1IE_ (0x0025)
#define WDT_ADLY_250	sfrb P1IE = P1IE_;
(WDTPW + WDTTMSEL + WDTCNTCL + WDTSSEL + WDTIS0)	#define P1SEL_ (0x0026)
#define WDT_ADLY_16	sfrb P1SEL = P1SEL_;
(WDTPW + WDTTMSEL + WDTCNTCL + WDTSSEL + WDTIS1)	#define P2IN_ (0x0028)
#define WDT_ADLY_1_9	const sfrb P2IN = P2IN_;
(WDTPW + WDTTMSEL + WDTCNTCL + WDTSSEL + WDTIS1 + WDTIS0)	#define P2OUT_ (0x0029)
#define WDT_MRST_32	sfrb P2OUT = P2OUT_;
(WDTPW + WDTCNTCL)	#define P2DIR_ (0x002A)
#define WDT_MRST_8	sfrb P2DIR = P2DIR_;
(WDTPW + WDTCNTCL + WDTIS0)	#define P2IFG_ (0x002B)
#define WDT_MRST_0_5	sfrb P2IFG = P2IFG_;
(WDTPW + WDTCNTCL + WDTIS1)	#define P2IES_ (0x002C)
#define WDT_MRST_0_064	sfrb P2IES = P2IES_;
(WDTPW + WDTCNTCL + WDTIS1 + WDTIS0)	#define P2IE_ (0x002D)
#define WDT_ARST_1000	sfrb P2IE = P2IE_;
(WDTPW + WDTCNTCL + WDTSSEL)	#define P2SEL_ (0x002E)
#define WDT_ARST_250	sfrb P2SEL = P2SEL_;
(WDTPW + WDTCNTCL + WDTSSEL + WDTIS0)	#define TAIV_ (0x012E)
#define WDT_ARST_16	const sfrw TAIV = TAIV_;
(WDTPW + WDTCNTCL + WDTSSEL + WDTIS1)	#define TACTL_ (0x0160)
#define WDT_ARST_1_9	sfrw TACTL = TACTL_;
(WDTPW + WDTCNTCL + WDTSSEL + WDTIS1 + WDTIS0)	#define TACCTL0_ (0x0162)
#define P1IN_ (0x0020)	sfrw TACCTL0 = TACCTL0_;
const sfrb P1IN = P1IN_;	#define TACCTL1_ (0x0164)
#define P1OUT_ (0x0021)	sfrw TACCTL1 = TACCTL1_;
sfrb P1OUT = P1OUT_;	#define TACCTL2_ (0x0166)
#define P1DIR_ (0x0022)	sfrw TACCTL2 = TACCTL2_;
sfrb P1DIR = P1DIR_;	#define TAR_ (0x0170)
	sfrw TAR = TAR_;
	#define TACCR0_ (0x0172)
	sfrw TACCR0 = TACCR0_;
	#define TACCR1_ (0x0174)
	sfrw TACCR1 = TACCR1_;
	#define TACCR2_ (0x0176)
	sfrw TACCR2 = TACCR2_;
	#define CCTL0_ (0x0162)
	sfrw CCTL0 = CCTL0_;
	#define CCTL1_ (0x0164)

续表 5.1

头文件内容		头文件内容	
sfrw CCTL1	= CCTL1_;	# define CCI	(0x0008)
# define CCTL2_	(0x0166)	# define OUT	(0x0004)
sfrw CCTL2	= CCTL2_;	# define COV	(0x0002)
# define CCR0_	(0x0172)	# define CCIFG	(0x0001)
sfrw CCR0	= CCR0_;	# define OUTMOD_0	(0 * 0x20)
# define CCR1_	(0x0174)	# define OUTMOD_1	(1 * 0x20)
sfrw CCR1	= CCR1_;	# define OUTMOD_2	(2 * 0x20)
# define CCR2_	(0x0176)	# define OUTMOD_3	(3 * 0x20)
sfrw CCR2	= CCR2_;	# define OUTMOD_4	(4 * 0x20)
# define TASSEL2	(0x0400)	# define OUTMOD_5	(5 * 0x20)
# define TASSEL1	(0x0200)	# define OUTMOD_6	(6 * 0x20)
# define TASSEL0	(0x0100)	# define OUTMOD_7	(7 * 0x20)
# define ID1	(0x0080)	# define CCIS_0	(0 * 0x1000)
# define ID0	(0x0040)	# define CCIS_1	(1 * 0x1000)
# define MC1	(0x0020)	# define CCIS_2	(2 * 0x1000)
# define MC0	(0x0010)	# define CCIS_3	(3 * 0x1000)
# define TACLR	(0x0004)	# define CM_0	(0 * 0x4000)
# define TAIE	(0x0002)	# define CM_1	(1 * 0x4000)
# define TAIFG	(0x0001)	# define CM_2	(2 * 0x4000)
		# define CM_3	(3 * 0x4000)
# define MC_0	(0 * 0x10)	# define DCOCTL_	(0x0056)
# define MC_1	(1 * 0x10)	sfrb DCOCTL	= DCOCTL_;
# define MC_2	(2 * 0x10)	# define BCSCTL1_	(0x0057)
# define MC_3	(3 * 0x10)	sfrb BCSCTL1	= BCSCTL1_;
# define ID_0	(0 * 0x40)	# define BCSCTL2_	(0x0058)
# define ID_1	(1 * 0x40)	sfrb BCSCTL2	= BCSCTL2_;
# define ID_2	(2 * 0x40)	# define MOD0	(0x01)
# define ID_3	(3 * 0x40)	# define MOD1	(0x02)
# define TASSEL_0	(0 * 0x100)	# define MOD2	(0x04)
# define TASSEL_1	(1 * 0x100)	# define MOD3	(0x08)
# define TASSEL_2	(2 * 0x100)	# define MOD4	(0x10)
# define TASSEL_3	(3 * 0x100)	# define DCO0	(0x20)
# define CM1	(0x8000)	# define DCO1	(0x40)
# define CM0	(0x4000)	# define DCO2	(0x80)
# define CCIS1	(0x2000)	# define RSEL0	(0x01)
# define CCIS0	(0x1000)	# define RSEL1	(0x02)
# define SCS	(0x0800)	# define RSEL2	(0x04)
# define SCCI	(0x0400)	# define XT5V	(0x08)
# define CAP	(0x0100)	# define DIVA0	(0x10)
# define OUTMOD2	(0x0080)	# define DIVA1	(0x20)
# define OUTMOD1	(0x0040)	# define XTS	(0x40)
# define OUTMOD0	(0x0020)	# define XT2OFF	(0x80)
# define CCIE	(0x0010)	# define DIVA_0	(0x00)

周期。使用下面的语句：

```
        MOV     #DELAYDATA ,R15
LOOP    DEC     R15
        JNZ     LOOP
```

程序中延时次数将决定延时时间，循环用到的两条指令 DEC 和 JNZ 共需 3 个机器周期，所以程序中的 DELAYDATA=80 000/3=26 666。这个数没有超出一个字所表示的范围，但如果需要的延时时间较长，一个字不够，则嵌套一层循环即可。下面的程序延时 1 s。

```
        MOV     #10 ,R14
LOOP    MOV     #26666,R15
LOOP1   DEC     R15
        JNZ     LOOP1
        DEC     R14
        JNZ     LOOP
```

使用 C 语言设计软件延时就更轻松了。可以固定延时时间，只修改循环变量即可；也可以设一参数，由该参数决定延时时间。

固定时间延时程序如下(改变 00x7FFF 就改变延时时间)：

```
void delay(void)
{
  unsigned long i;
    for (i = 0x7FFFF; i > 0; i--);
}
```

带参数延时程序如下(延时时间由参数 DELAY 决定)：

```
void delay(unsigned long  DELAY;)
{
 unsigned long i;
    for (i = DELAY; i > 0; i--);
}
```

2. Timer_A 硬件延时

延时程序也可使用硬件定时器来完成，用 Timer_A，定时 5 ms，则定时器的计数值为 0.005/(1/800 000) =4 000，相应程序如下：

```
SetupTA  mov.w   #TASSEL1+TACLR,&TACTL ;以 SMCLK 为定时器输入时钟并清除 TAR
         mov.w   #4000,&CCR0           ;确定定时时间
         bis.w   #MC1,&TACTL           ;以增计数开始 Timer_A
LOOP     BIT     #1,&TACTL             ;等定时时间到(标志位将被置位)
         JZ      LOOP                  ;不到，继续等
         ......                        ;时间到
```

3. 看门狗硬件延时

经常在没有使用看门狗的情况下，可用看门狗定时器实现上面的程序功能。这里从略。

5.1.3 常用数学程序的设计

下面的程序是常用的数学程序，其中很少用到硬件乘法器，因为大多数型号里没有硬件乘

法器模块。

1. 32 位数加法

```
ADD32 ;功能:实现 32 位数相加
      ;入口参数:R4,R5 为第 1 个 32 位加数,R6,R7 位第 2 个加数
      ;出口参数:R4,R5 为结果的两字,C 为最高位
      CLRC
      ADD     R7,R5
      ADDC    R6,R4
      RET
```

2. 32 位数减法

```
ADD32 ;功能:实现 32 位数相减
      ;入口参数:R4,R5 为 32 位减数,R6,R7 位被减数
      ;出口参数:R4,R5 为差
      CLRC
      SUB       R7,R5
      SUBC      R6,R4
      RET
```

以上两程序相当简单,因为 MSP430 本身就是 16 位机。要扩展为更多位数的加减法运算也是很简单的,这里不赘述。

3. 16×16 位无符号数乘法子程序

如果使用硬件乘法器则相当简单,同时速度也相当快捷,只要将数据放到相应的寄存器单元即可,但是所用器件必须带有硬件乘法器。下面是对应的程序:

```
MPY_      ;功能:  R11 x R12 = R15 | R14
          ;入口参数:R11, R12 为两个无符号乘数
          ;出口参数:R15, R14 为乘积,其中 R15 为高字,R14 为低字
          MOV   R11,&MPY
          MOV   R12,&OP2
          NOP
          NOP
          MOV  &SUMHI, R15
          MOV  &SUMLO,R14
          RET
```

如果所用器件片内没有硬件乘法器,那么使用下面的程序:

```
MPYU   ;功能:R11 x R12 = R15 | R14
       ;入口参数:R11, R12 为两个无符号乘数
       ;出口参数:R15, R14 为乘积,其中 R15 为高字,R14 为低字
;-------------------------------------------------------------
            CLR     R14                ;起始状态清除结果低字内容
            CLR     R15                ;起始状态清除结果高字内容
MACU        CLR     R13
            MOV     #1,R10
MPY2        BIT     R10,R11
```

```
            x  -= 2;
            } else
        y -= x;
        x--+;
        y <<= 1;                    // <y, h> <<= 2
        if (h & Minus) y++;
        h <<= 1;
        y <<= 1;
        if (h & Minus) y++;
        h <<= 1;
        }
        return x;
}
```

根据该算法编成下面的汇编子程序，直接调用即可。

```
Sqrt        ;入口参数:x_MSB 或 R4 为被开方数的高字
            ;         x_LSB 或 R5 为被开方数的低字
            ;出口参数:x_MSB 或 R4 为开方运算结果的整数部分
            ;         x_LSB 或 R5 为开方运算结果的小数部分

#define     x_MSB   R4
#define     x_LSB   R5
#define     y_MSB   R6
#define     y_LSB   R7
#define     h_MSB   R8
#define     h_LSB   R9
#define     i       R10
            Mov     x_MSB, h_MSB
            Mov     x_LSB, h_LSB
            Clr     x_MSB
            Clr     x_LSB
            Clr     y_MSB
            Clr     y_LSB
            Mov     #32,    i
Sqrt10      SetC                        ; x <<= 1; x++;
            Rlc     x_LSB
            Rlc     x_MSB
            Sub     x_LSB, y_LSB        ; y.l -= x.l;
            Subc    x_MSB, y_MSB
            Jhs     Sqrt12              ; if (y.l & Minus)
            Add     x_LSB,y_LSB         ; {
            Addc    x_MSB, y_MSB        ; y.l += x.l;
            Sub #2, x_LSB               ; x.l -= 2; }
Sqrt12      Inc     x_LSB               ; x.l++;
```

```
                                    ; <y.l, HilfsReg> <<= 2
        Rla     h_LSB               ; <y.l, HilfsReg> <<= 1
        Rlc     h_MSB
        Rlc     y_LSB
        Rlc     y_MSB
        Rla     h_LSB               ; <y.l, HilfsReg> <<= 1
        Rlc     h_MSB
        Rlc     y_LSB
        Rlc     y_MSB
        Dec     i
        Jne     Sqrt10
        Ret
```

5.1.4 码制转换程序设计

1. 二进制数转换到 BCD 码

```
BINTOBCD ;入口参数:R15 为待转换的二进制数
         ;出口参数:转换的结果为以 210H 为首地址的以后 5 字节,210H 内容为万位
         mov    r15,r12
         mov    #0,r13
         mov    #10000,r11
         call   #DIVIDE          ;原数除以 10 000
         mov.b  r14,&210h
         mov    r13,r12
         mov    #0,r13
         mov    #1000,r11
         call   #DIVIDE          ;余数除以 1 000
         mov.b  r14,&211h
         mov    r13,r12
         mov    #0,r13
         mov    #100,r11
         call   #DIVIDE          ;余数除以 100
         mov.b  r14,&212h
         mov    r13,r12
         mov    #0,r13
         mov    #10,r11
         call   #DIVIDE          ;余数除以 10
         mov.b  r14,&213h
         mov.b  r13,&214h
         ret
```

以上的程序是一个很笨拙的程序,调用的是除法子程序。而除法子程序本身就是很耗时间的程序,这里还多次调用。但这个程序在功能上是没有问题的。

MSP430 指令系统提供了十进制加法指令 DADD, src,dst ,利用这条指令再配合移位指令就可以编写出相当简洁的二进制数到十进制数转换的程序。

单字的二进制数转换为十进制数的子程序源代码如下：

```
BIN_BCD16  ;入口参数:R4 为待转换的二进制数
           ;出口参数:R6,R5 为压缩 BCD 码,R6 的最高半字节为十进制数的最高位
           MOV     #16,R7
           CLR     R6
           CLR     R5
LOOP       RLA     R4          ;移位
           DADD    R5,R5
           DADD    R6,R6
           DEC     R7
           JNZ     LOOP
           RET
```

对上面程序稍加改进，就可设计出 32 位二进制数转换为十进制数的程序，代码如下：

```
BIN_BCD32  ;入口参数:R5,R4 为待转换的二进制数,R5 为高 16 位
           ;出口参数:R7,R6 为压缩 BCD 码,R7 的最高半字节为十进制数的最高位
           ;此程序转换出的最大十进制数为 9999 9999,所以输入的最大二进制数应
           ;该不大于 05F5E0FFH,否则将超出范围
           MOV     #32,R15
           CLR     R7
           CLR     R6
LOOP       RLA     R5          ;移位
           RLC     R4
           DADD    R6,R6
           DADD    R7,R7
           DEC     R15
           JNZ     LOOP
           RET
```

2. BCD 码转换到二进制数

将 4 位十进制数转换为 16 位二进制数，假设 4 位十进制数的每位数字是 a1,a2,a3,a4，则这个十进制数的数值为

$$x = a1\times1000+a2\times100+a3\times10+a4=a4+10\times(a3+10\times(a2+10\times a1))。$$

下面的程序使用的就是这种算法。该程序经常用于将由键盘输入的数据(常为十进制数)进行的二进制数转换。

```
BCDBIN     ;入口参数:R4 为将要转换的十进制数
           ;出口参数:R5 转换结果(二进制数)
           ;由于十进制数为 16 位表示,所以最大也就是 9999→270FH
           MOV     #4,R8       ; R8 为循环计数器,循环 4 次
           CLR     R5
           CLR     R6
SHFT4      RLA     R4          ; 算术左循环
           RLC     R6          ; 经过 C 位左循环
           RLA     R4
```

```
        RLC     R6
        RLA     R4
        RLC     R6
        RLA     R4
        RLC     R6
        ADD     R6,R5       ; ai+10*ai-1
        CLR     R6
        DEC     R8          ; 改变循环变量
        JZ      _END        ; 完了结束
MPY10   RLA     R5          ; 没有完则继续
        MOV     R5,R7
        RLA     R5
        RLA     R5
        ADD     R7,R5
        JMP     SHFT4
_END    RET
```

下面的程序段由硬件乘法器来实现将 BCD 码转换到二进制数的功能，可以看出程序要简洁的多，同时执行时间也短得多：

```
;----4 位 BCD 转化 16 位 BIN 程序----------
;使用无符号数乘加计算
;其中利用 430F14X 的乘法器完成
;4 位 BCD 码存放在以 R6 作为存放数据首地址的以后 4 个单元内
;-------------------------------------
BCD4_BIN16
        MOV.B   2(R6),      R10
        MOV     R10,        &RESLO      ;将百位放入结果低字节寄存器 RESLO
        MOV.B   3(R6),      R10         ;[(千位*10+百位)*10+十位]*10+个位
        MOV     R10,        &MAC        ;千位装入操作数 1,定义无符号乘加
        MOV     #0AH,       &OP2        ;10 装入操作数 2,开始乘加运算
        MOV     &RESLO,     &TEM        ;TEMP=千位*10+百位
        MOV.B   1(R6),      R10
        MOV     R10,        &RESLO      ;将十位放入结果低字节寄存器 RESLO
        MOV     &TEM,       &MAC        ;百位装入操作数 1,定义无符号乘加
        MOV     #0AH,       &OP2        ;10 装入操作数 2,开始乘加运算
        MOV     &RESLO,     &TEM        ;TEMP=(千位*10+百位)*10+十位
        MOV.B   0(R6),      R10
        MOV     R10,        &RESLO      ;将个位放入结果低字节寄存器 RESLO
        MOV     &TEM,       &MAC        ;十位装入操作数 1,定义无符号乘加
        MOV     #0AH,       &OP2        ;10 装入操作数 2,开始乘加运算
        MOV     &RESLO,     &TEMP_BIN   ;TEMP=[(千位*10+百位)*10+十位]*
                                            10+个位
        MOV     &RESHI,     &TEMP_BIN+2
        RET
```

3. 二进制数转换为 ASCII 码

将 1 位十六进制数转换为 ASCII 码。1 位十六进制数由“0～9”这 10 个数字与“A～F”这 6 个字母组成。它们与 ASCII 码的对应关系分别为“加 30H”与“加 37H”则得到对应的 ASCII 码。程序如下：

```
DEC_ASCII           ;入口参数 R9:待转换的十六进制数
                    ;出口参数 R9:转换后的 ASCII 码
        CLRC
        PUSH    R9
        SUB     #10,R9
        JNC     DEC_AS1     ;判断是否小于 10
        POP     R9
        ADD     #30H,R9
        RET
DEC_AS1 POP     R9
        ADD     #37H,R9
        RET
```

4. ASCII 码转换为二进制数

将 ASCII 码转换为 1 位十六进制数。同样的道理：1 位十六进制数由 0～9 这 10 个数字与 A～F 这 6 个字母组成。它们与 ASCII 码的对应关系分别为“加 30H”与“加 37H”则得到对应的 ASCII 码。这里做个减法即可。程序如下：

```
DEC_ASCII           ;入口参数 R9:待转换的 ASCII 码
                    ;出口参数 R9:转换后的十六进制数
        CLRC
        PUSH    R9
        SUB     #10,R9
        JNC     DEC_AS1     ;判断是否小于 10
        POP     R9          ;因为在判断是否小于 10 时,做了减法,R9 的值改变了
        SUB     #30H,R9     ;所以先保存到堆栈,在运算时再恢复
        RET
DEC_AS1 POP     R9
        SUB     #37H,R9
        RET
```

5.1.5　发光二极管类显示器件接口设计

在单片机系统中显示部分较为简单、廉价，一般是发光二极管、数码管及液晶显示器(不是接计算机的，而是只能显示几个数字、字符或少量点阵)等。

1. 与发光二极管的接口

发光二极管是一个二极管，有正负两极，在正极上接上相对于负极一个大于其管压降的电压，有正向的电流流过，则发光。这里要注意的是，MSP430 输出能力不是很强(请查阅相关的器件手册)，则在驱动发光二极管时一定要限流，加个电阻即可。

发光二极管有两种连接方式。

① 发光二极管的阴极接地，阳极接 MSP430 的输出端口（比如 P1.0）。点亮发光二极管的程序（汇编）为

```
BIS.B    #1,&P1DIR
BIS.B    #1,&P1OUT
```

或（C 语言）

```
P1DIR |= BIT0
P1OUT |= BIT0
```

让发光二极管熄灭的程序为

```
BIC.B    #1,  &P1OUT
```

② 发光二极管的阳极接电源正端，阴极接 MSP430 的输出端口（比如 P1.0）。点亮发光二极管的程序（汇编）为

```
BIS.B    #1,&P1DIR
BC.B     #1,&P1OUT
```

或（C 语言）

```
P1DIR |= BIT0
P1OUT &= ~BIT0
```

让发光二极管熄灭的程序为

```
BIS.B    #1,  &P1OUT
```

2. 与数码管的接口

与发光二极管的连接主要用于一些状态的显示、开关量的显示等方面，但它只能显示少量信息。那么要显示数字，用数码管即可。数码管将 8 只发光二极管按照一定的规则排列，如图 5.1 所示。

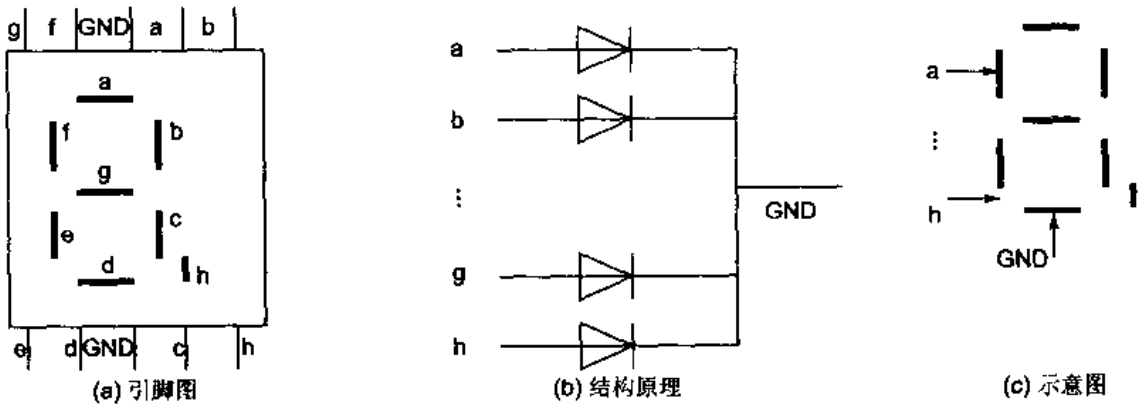


图 5.1 共阴数码管

如果它们的阴极接地，在 a～h 输入不同的高低电平，则显示相应的图形。如给 g 和 h 输入低电平，而其他都输入高电平，就显示“0”的图形；如果 b 和 c 等于 1，而其他都是 0，则显示“1”。一般情况下，将 a,b,c,d,e,f,g,h 对应与单片机的某个端口相连接，如 MSP430 的 P1 口，就是 a—p1.0,b—p1.1,c—p1.2,…,h—P1.7。那么，要显示数字“0”，就给 P1 口输出数据 03FH；要显示数字“1”，就给 P1 口输出数据 06H。数据 03FH 和 06H 称为显示段码。下面的表 5.2 列出了常用的显示段码。

表 5.2　常用显示字符的显示段码表

显示字符	数码管的各段("1"表示高电平,否则低电平,一般 a 为最低位)								显示段码
	h	g	f	e	d	c	b	a	
0			1	1	1	1	1	1	3FH
1						1	1		06H
2		1		1	1		1	1	5BH
3		1			1	1	1	1	4FH
4		1	1			1	1		66H
5		1	1		1	1		1	6DH
6		1	1	1	1	1		1	7DH
7						1	1	1	07H
8		1	1	1	1	1	1	1	7FH
9		1	1		1	1	1	1	6FH
A		1	1	1		1	1	1	77H
B		1	1	1	1	1			7CH
C		1		1	1	1			39H
D		1		1	1	1	1		5EH
E		1	1	1	1			1	79H
F		1	1	1				1	71H
0.	1		1	1	1	1	1	1	0BFH

数码管与单片机的连接方式很多,主要有以下几种方式。

(1) 与数码管直接连接显示

直接通过 MSP430 的 I/O 口线与数码管连接,电路如图 5.2 所示。MSP430 系列中 FLASH 的 13X,14X,43X 及 44X 都有 6 个 8 位 I/O 口,如果端口富余,则完全可使用这种接口方法:P1.0～P1.7 通过 500 Ω 的电阻对应连接第一只数码管的引脚 a～h,P2.0～P2.7 通过 500 Ω 的电阻对应连接第二只数码管的引脚 a～h,P3.0～P3.7 通过 500 Ω 的电阻对应连接第三只数码管的引脚 a～h,所有数码管的 GND 引脚都接地。

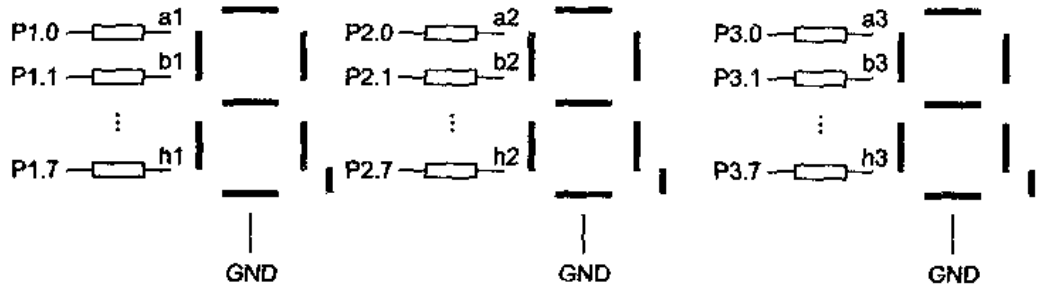


图 5.2　3 只数码管与 P1,P2 及 P3 相连接

要在 3 只数码管上分别显示 1,2,3,可使用下面的程序段:

```
MOV.B    #0FFH,&P1DIR
MOV.B    #0FFH,&P2DIR
MOV.B    #0FFH,&P3DIR      ;定义三端口为输出方向
```

```
MOV.B   #06H,&P1OUT      ;给出对应的显示码
MOV.B   #5BH,&P2OUT
MOV.B   #4FH,&P3OUT
```

一般显示程序并不直接给对应端口以显示段码，而是写一个公用的显示子程序。入口参数就是要显示的数字，常放在称为显示缓存的一片存储区域，这里使用200H，201H及202H 3个存储单元分别对应3只数码管。用查表的方法将要显示的数字转换为对应的显示段码。汇编语言程序如下：

```
DISP3       PUSH   R5
            MOV.B  #0FFH,&P1DIR
            MOV.B  #0FFH,&P2DIR
            MOV.B  #0FFH,&P3DIR          ;定义三端口为输出方向
            MOV.B  &200H,R5
            MOV.B  DISTAB(R5),&P1OUT     ;在显示码表中查得显示码，并送给端口P1
            MOV.B  &201H,R5
            MOV.B  DISTAB(R5),&P2OUT     ;在显示码表中查得显示码，并送给端口P2
            MOV.B  &202H,R5
            MOV.B  DISTAB(R5),&P3OUT     ;在显示码表中查得显示码，并送给端口P3
            POP    R5
            RET
DISTAB      DB  3FH，06H，  5BH    ,4FH ;0   1   2   3
            DB  66H，6DH，  7DH，  07H  ;4   5   6   7
            DB  7FH，6FH，  77H，  7CH  ;8   9   A   B
            DB  39H，5EH，  79H    ,71H ;C   D   E   F
```

C语言程序如下：

```
unsigned char seg[]={0x3f,0x06,0x5b,0x4f,
    0x66,0x6d,0x7d,0x07,
    0x7f,0x6f,0x77,0x7c,
    0x39,0x5e,0x79,0x71
    }
void    disp3(unsigned char a, unsigned char b, unsigned char c)
    {
        P1DIR=0XFF;
        P2DIR=0XFF;
        P3DIR=0XFF;          //先定义端口为输出方向
        P1OUT=seg[a];        //在显示段码表中查得相应段码，并送到对应端口
        P2OUT=seg[b];
        P3OUT=seg[c];
  }
```

要在3只数码管上分别显示1,2,3，则可以改为使用下面的汇编程序段：

```
    ……
    MOV.B   #1,  &200H
    MOV.B   #2,  &201H
```

```
MOV.B    #3,  &202H
CALL     # DISP3
……
```

或更简洁的 C 语言程序：

```
……
disp3(1,2,3) ;     /* 如此一句话即可
……
```

使用这种方式与数码管连接有它的优点：为静态显示，在软件上较为省事，只要调用一次显示子程序，将显示码送到相应的端口即可。但它的缺点也很明显：占用口线太多。在 I/O 口线富余的情况下，还是很值得使用的。

(2) 扫描方式显示

直接显示的明显缺点是占用 I/O 口线太多。如果有 6 位数码要显示，则 MSP430 的 P1～P6 都被占用了。使用扫描显示的办法即可解决这个问题：使用较少的 I/O 口线实现较多的数码显示。

这种方式使用的口线少是因为所有数码显示器的段驱动端都对应地连在一起，第一只的 a 与第二只的 a 与第三只的 a 与……都连在一起，同样所有数码管的 b 也连在一起。这样，一只数码管要 8 条 I/O 口线进行段驱动，10 只数码管也只要 8 条 I/O 口线进行段驱动。但所有的数码管的接地端就不能都接地了，也要用 I/O 口线进行驱动。只有当接地端为 0(共阴极)的数码管才能显示，接地端为 1 的数码管不能显示。

所有的数码管的段驱动端都连在一起，其显示原理是对每一个数码管的接地端进行控制，因为只有接地端为 0 的数码管才能显示。例如有 8 只数码管要分别显示 1,2,3,4,5,6,7,8。可以这样做：给显示器的段驱动口送“1”的显示段码 06H，并使第一只数码管的接地端为低电平，其他数码管的接地端为高电平，再延时 2 ms；之后，给显示器的段驱动口送“2”的显示段码 5BH，并使第二只数码管的接地端为低电平，其他数码管的接地端为高电平，再延时 2 ms；……；之后，给显示器的段驱动口送“8”的显示段码 7FH，并使第八只数码管的接地端为低电平，其他数码管的接地端为高电平，再延时 2 ms。这就是所说的扫描方式。这样反复操作，则人眼睛看的效果就是所有的 8 只数码管分别显示了 1,2,3,…,8。因为人的眼睛看事物有一个 0.4 s 的视觉暂停，对每只数码管的 14 ms 不显示时间是分辨不出来的(16 ms 中，实质上只有 2 ms 在显示，其余 14 ms 没有显示)，因此看上去就好象全都在显示，而没有停留。图 5.3 是扫描显示的原理图。

图中有 8 只数码管显示器，它们的段驱动对应连在一起，再通过 P5 口的 8 条口线经驱动器驱动。8 只数码管的接地端通过 P1 口的 8 条口线分别经过驱动器连接。

在程序设计时，同样设置一显示缓存，用于保存将要显示的数据。在显示程序中，将显示数据取出，查得相应显示段码，送到段驱动端口，再使对应的数码管的接地端输出低电平(其余为高电平)，延时一段时间；再将其余数据用同样的方法使相应的数码管显示。显示流程图如图 5.4 所示。

扫描显示的汇编程序如下：

```
DIP      MOV.B    #0FFH,&P1DIR      ;P1 口输出
         MOV.B    #0FFH,&P5DIR      ;P5 口输出
         MOV      #200H,R9          ;初始化显示指针
```

```
        delay(500);                          //延时一会儿
      }
    }
```

子程序的使用：先在显示缓存写上要显示的数据，再调用显示子程序。（举例略）

（3）通过移位寄存器与数码管接口

上面已经介绍了两种与数码管接口的方法，它们各有千秋。这里再介绍一种通过移位寄存器与数码管的接口方法。这种方法更加节省 MCU 的口线，只要两条 I/O 线。首先移位寄存器有很多种，这里使用 74HC164。74HC164 是 CMOS 型串行输入、8 位并行输出的移位寄存器，可以与 MSP430 共用电源。图 5.5 是 74HC164 的功能表和引脚图。

Inputs				Outputs			
$\overline{CLR}$	CLK	A	B	Q_A	Q_B	…	Q_H
L	X	X	X	L	L		L
H	L	X	X	Q_{A0}	Q_{B0}		Q_{H0}
H	↑	H	H	H	Q_{An}		Q_{Gn}
H	↑	L	X	L	Q_{An}		Q_{Gn}
H	↑	X	L	L	Q_{An}		Q_{Gn}

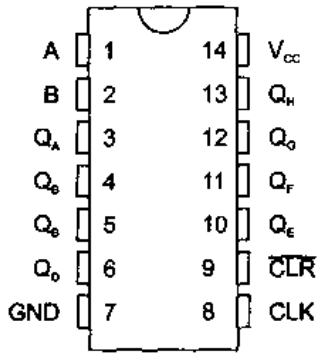


图 5.5　74HC164 的功能表和引脚图

数码管通过 74HC164 与 MSP430 单片机的连接图如图 5.6 所示。8 片 74HC164 的 CLK 端连在一起再与 P1.0 相连；所有 74HC164 的 B 和 $\overline{CLR}$ 都接高电平；第一片 74HC164 的 A 连接到单片机的 P1.1，第二片 74HC164 的 A 连接到第一片 74HC164 的 Q_H 端……；每一片 74HC164 输出到 LED。

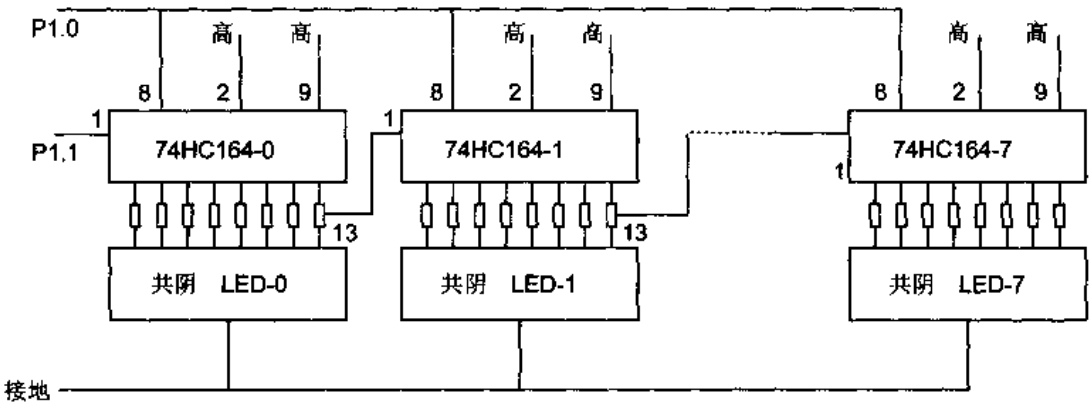


图 5.6　数码管通过 74HC164 与 MSP430 连接

当第一片 74HC164 的数据端 A 输入一位数据后，再在 CLK 端给一个上升沿，则数据端 A 的数据被移到第一片 74HC164 的 Q_A 端；如果再在数据端 A 送一个数据，之后给 CLK 端一个上升沿，则刚才的数据移到了 Q_B 端，现在送入的数据被移到 Q_A 端；……。图 5.7 所示为 74HC164 的时序图。

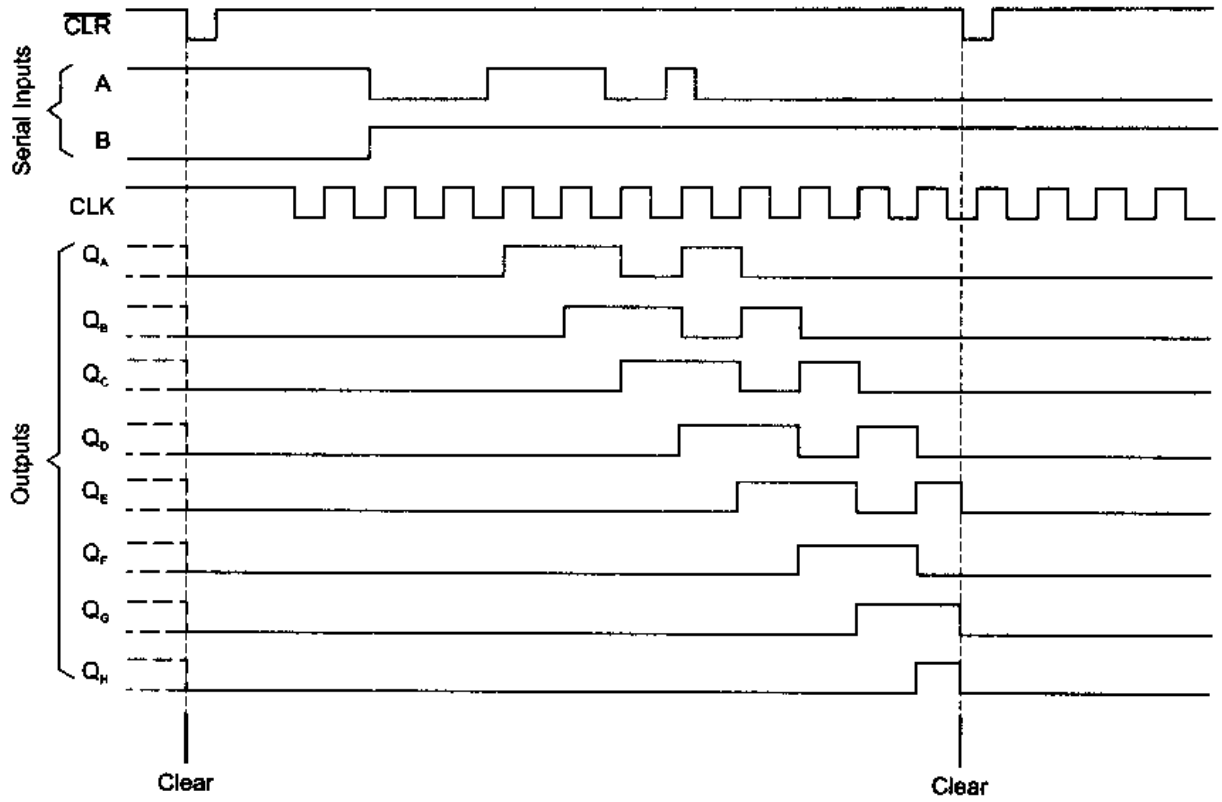


图 5.7　74HC164 的时序图

如果要在第一个数码管上显示数字“3”，则只须将“3”的显示码 4FH 的 8 个数据位依次送给 74HC164。这样依次将 8 个要显示数据的显示码的各位送给 74HC164 就达到了 8 位数码显示效果。这里要特别注意的是：最先送出的显示数据在最后一个位置上显示，而最后送出的在最前面位置上显示；同时在移位过程中，由于所有的显示位都会在数码管上出现，所以尽管是静态显示也会有闪烁的情况。解决闪烁问题的办法是控制所有数码管的接地端，当在数据移动时，让所有数码管的接地端都为 1(对共阴型)，使得它全熄灭，当数据移完之后，再打开显示即可。

针对图 5.6 硬件的相应软件如下：

```
;显示缓存为 200H～207H
DIS_12      PUSH     R4              ; 显示 8 个数据到显示器
            PUSH     R5
            MOV.B    #8,R5
DIS_121     MOV.B    1FFH(R5),R4     ; 取出 8 个要显示的数据之一
            CALL     #DIS_1          ; 调一位数据显示子程序
            DEC.B    R5
            JNZ      DIS_121         ; 8 位显示完了吗?
            POP      R5
            POP      R4
            RET
DIS_1       PUSH     R5
            PUSH     R4
```

```
        MOV     #8,R5                   ;显示一个数字
        MOV.B   DIS_TAB(R4),R4          ;查得显示码
LOOP:   RLC.B   R4                      ;移出将要送到74HC164的数据位
        JC      LOOP1                   ;C=1 JMP P2.0=1
        BIC.B   #1,&P2OUT               ;送出数据"0"
        JMP     LOOP2
LOOP1   BIS.B   #1,&P2OUT               ;送出数据"1"
LOOP2   CALL    #CLK164                 ;给出时钟上升沿
        DEC     R5
        JNZ     LOOP                    ;8位数据没有移完,则继续
        POP     R4
        POP     R5
        RET
CLK164  BIS.B #2,&P2OUT                 ;时钟子程序
        BIC.B #2,&P2OUT                 ;
        RET
        ;以下为显示段码表
TABLED: DB  3FH,06H,  5BH,  4FH     ;0  1  2  3
        DB  66H,6DH,  7DH,  07H     ;4  5  6  7
        DB  7FH,6FH,  77H,  7CH     ;8  9  A  B
        DB  39H,5EH,  79H,  71H     ;C  D  E  F
```

对该程序的使用是:在显示缓存写要显示的数据,再调用此子程序即可。

(4) 通过显示译码器与数码管连接

显示译码器是将4位二进制数翻译成显示码的器件。当然也可以使用它将要显示的数据翻译到数码管。常用的显示译码器有CD4511或74HC48等,这里以CD4511为例说明如何与MSP430连接。CD4511的内部结构和引脚图如图5.8所示。

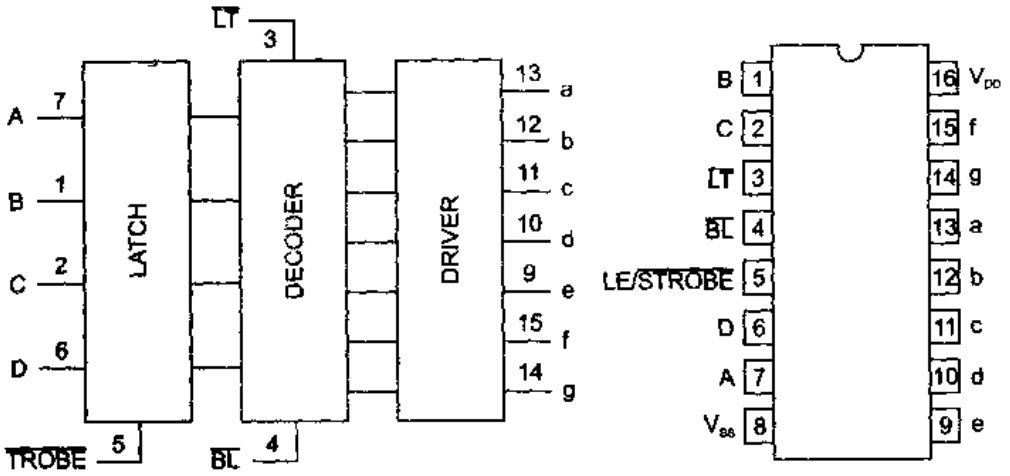


图5.8 CD4511的内部结构和引脚图

CD4511由3部分组成:输入锁存、译码及输出驱动。有4条输入信号为要译码的数据输入;有7条输出可直接驱动数码管显示;有3个控制信号在译码时控制引脚的正确连接为

LE=0,BI=1,LT=1。那么 CD4511 与 MSP430 单片机及数码管的连接如下(所有的 LE=0,BI=1,LT=1):

第一片 CD4511 的A　与单片机的 P1.0 相连;
B　与单片机的 P1.1 相连;
C　与单片机的 P1.2 相连;
D　与单片机的 P1.3 相连;
a　与数码管 1 的 a 相连;
b　与数码管 1 的 b 相连;
……
g　与数码管 1 的 g 相连。
第二片 CD4511 的A　与单片机的 P1.4 相连;
B　与单片机的 P1.5 相连;
C　与单片机的 P1.6 相连;
D　与单片机的 P1.7 相连;
a　与数码管 2 的 a 相连;
b　与数码管 2 的 b 相连;
……
g　与数码管 2 的 g 相连。

整个电路有一片 MSP430 单片机、两片 CD4511 及两只数码管显示器,则相应的显示子程序 DIS4511 如下:

```
;入口参数 R5 为压缩的 BCD 码,低半字节在第一数码管显示,高半字节在第二数码管显示
DIS4511    MOV.B #0FFH,&P1DIR
           MOV.B  R5,  &P1OUT      ;字节送出要显示的两个 4 位二进制数,由 CD4511 译码
           RET
```

以上几种与数码管的连接方式比较,通过译码器连接的软件的编写最简单。

3. 与点阵 LED 的接口

点阵 LED 显示器实质上与数码管差不多,也是很多的发光二极管排列成 N×M 点阵,这样可显示更为丰富的内容。常用的点阵 LED 有用于 ASCII 显示的 5×7 点阵 ,灵活的 8×8点阵等,常用 4 块 8×8 点阵拼起来用于 16 点阵汉字的显示。

点阵 LED 显示器几乎没有使用静态方式显示的,而全部都是扫描方式显示的动态结构。图 5.9 是 5×7 点阵的结构原理图。其他类型或其他规模的点阵 LED 的结构原理和显示方法是一样的。

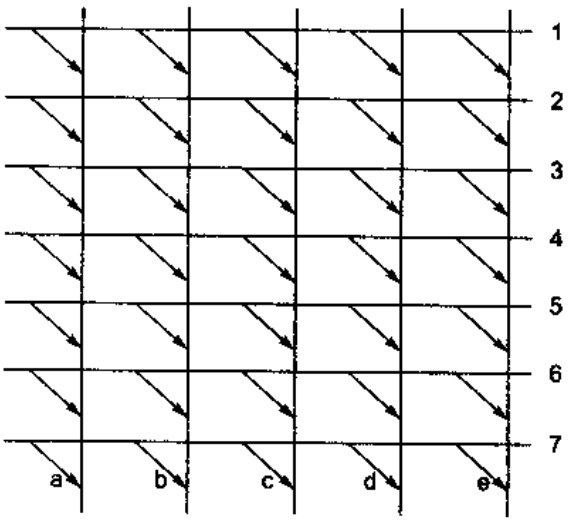


图 5.9　5×7 点阵的结构原理图

该结构分为行驱动 1～7 与列驱动 a,b,c,d,e。在行与列的交接处有一个发光

```
        delay(500);                    // 延时一会儿，延时子程序与前面的相同，这里不重复
      }
    }
```

5.1.6 键盘接口设计

在单片机应用中，键盘是人机对话的输入设备，借助键盘可向系统设置参数，发出控制指令等。但在单片机应用中的键盘不同于通用的计算机键盘，它须设计者自行设计。在这部分将详细讲述 MSP430 单片机应用中键盘的软件设计。

1. 按键的工作原理

在单片机设计中常用轻触按键作为输入设备——键盘的单元电路。它的一般结构是由两个电极和一个弹簧金属片构成的，如图 5.12 所示。当金属弹簧片上的按键 K 按下时，两个电极 A 和 B 被连通。但实际使用并非如此简单，因为单片机的运行速度相对于操作者按下按键的速度来是非常快的，所以应考虑更多的细节问题。

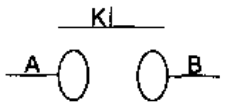


图 5.12 单元电路结构图

这个细节主要是按下按键的前后都有抖动。如果按键的 A 端接地，B 端接上拉电阻，则平时按键的 B 端为高电平，当按键按下时为低电平，松开后又是高电平，这是理论值；而实际 B 端的情况如图 5.13 所示。

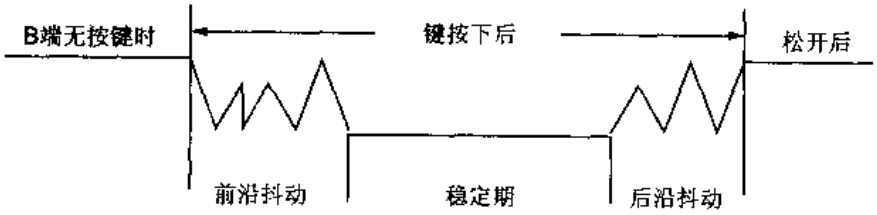


图 5.13 按键的抖动

如果将按键的 B 端信号送到 MCU，系统会认为有几个低电平，就按了几次按键。因为 MCU 认为低电平是按键按下，而抖动过程有很多低电平，同时还有不短的时间，一般在 10 ms 左右。常用的清除抖动的方法有 3 种：

① 使用 R－S 触发器构成消抖动开关；

② 使用电阻和电容构成积分器；

③ 使用软件延时。

前两种方法是使用硬件，第三种方法是使用软件，即：当 MCU 得知按键的 B 端出现低电平时，就知道可能有按键按下，于是等待 10 ms，10 ms 之后再检测按键的 B 端，如果还是低电平，则就一般的机械按键而言，已经是处于稳定期了，按键的抖动被消除了；如果 10 ms 之后按键的 B 端没有低电平了，则说明是干扰信号，而非按键按下。

2. 键盘程序的一般编写方法

键盘是由若干上述的独立按键按一定的规则组合而成的。也就是说，键盘的基本元素是按键，那么消除按键的抖动是必须的。同时，键盘是由若干按键构成的，那么判断是哪一个按下则需要通过判键得到键值。得到键值之后，还有一件事情就是等待按下的按键松开（注意：如果系统中使用了看门狗，则在这里要不断地清空看门狗，因为假如使用者长时间按着键，则

看门狗超时、系统复位)。综合起来,一般的键盘程序有如下 3 个步骤:

- 消除按键抖动(如果使用硬件,则可略);
- 判断是哪个按键按下,识别键码;
- 等待按键松开。

3. 独立按键式键盘

在系统中需要少量按键时,可使用按键与单片机的 I/O 口线直接连接的方法构成。平常使用的电子表、BP 机及计步器等,它们的体积都很小巧,不可能使用很多按键构成输入部分,都只有 3 个左右的按键。在这种情况下,就使用独立按键式键盘。如图 5.14 所示。

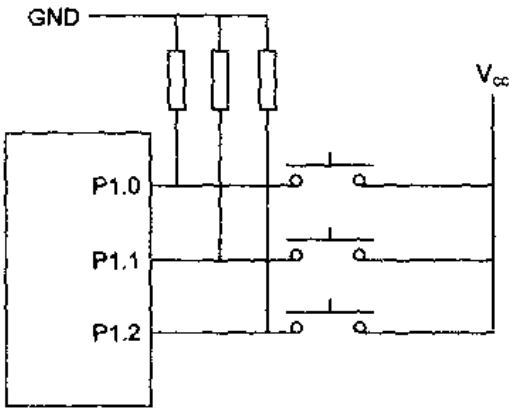


图 5.14　3 只独立按键直接与 3 条口线相连

在软件的编写上,可采用查询方式,也可采用中断方式。因为 MSP430 的 P0,P1,P2 等 3 个 8 位端口都有中断能力,建议读者使用中断方式。键盘程序主要是按上述的 3 个步骤进行编写。在主程序中必须设置 P1 口使之能进入中断。

汇编主程序:

```
……
BIS.B     # 07H ,    &P1DIR
BIC.B     # 07H ,    &P1OUT        ; 先输出低电平
BIC.B     # 07H,     &P1DIR        ; P1.0,P1.1,P1.2 为输入模式
BIS.B     #07H,     &P1IE         ; P1.0,P1.1,P1.2 中断使能
BIC.B     # 07H,     &P1IES        ; P1.0,P1.1,P1.2 上升沿触发中断
EINT                                ; 总中断使能
……
```

中断服务程序:

```
                                    ; 出口参数:按键键值在 R5 中
P1KEY3    CALL1     #KEYJ3        ; 判断是否有按键
          JNC       KEYEND        ; 没有则退出
          CALL      #DELAY10MS    ; 如有,则延时 10 ms   } 消抖动
          CALL      #KEYJ3        ; 再判键
          JNC       KEYEND        ; 如没有按下则退出
          CALL      #KEYCODE3     ; 如有,则调认键程序得到键值
          PUSH      R5            ; 保护键值
KEYLOOP   CALL      #KEYJ3        ; 等待按键松开
          JC        KEYLOOP       ; 没有松开,则继续等待
          POP       R5            ; 按键松开之后,恢复键值
KEYEND    RETI

KEYJ3     BIT.B     #07H,    &P1IN  ; 判断有无按键按下,如果有,则 C=1
```

```
                RET                                ；如果没有按键按下，则 C=0
KEYCODE         BIT.B     #1,     &P1IN    ；判断 3 个按键中是哪一个被按下
                JNC       K2
                MOV       #0,     R5       ；如果是接到 P1.0 的按键，则输出 R5=0
                RET
K2              BIT.B     #2,     &P1IN
                JNC       K3
                MOV       #1,     R5       ；如果是接到 P1.1 的按键，则输出 R5=1
                RET
K3              BIT.B     #4,             &P1IN
                JNC       K4
                MOV       #2,     R5       ；如果是接到 P1.2 的按键，则输出 R5=2
K4              RET
```

对应的 C 语言程序(在主程序中的 P1 口设置语句略)中，连接 P1 口的键盘中断服务程序如下：

```
unsigned char  keybuf;                    // 键值缓存器
unsigned char p1keyj(void)                // 判键子程序
    {unsigned char x;
     x=(P1IN&0X07);                       // P10～P12 接有按键
    return(x);                            // 有按键返回，非 0
    }

unsigned char  keycode()                  // 找哪个按键被按下，查键值子程序
    {
    unsigned char  x;
    if((P1IN&0x07)= = 1)                  // 是否第一个按键
        then x=0;
        else
        if((P1IN&0x07)= = 2)              // 是否第二个按键
            then x=1;
            else
            if((P1IN&0x07)= = 4)          // 是否第三个按键
            x=2;
    return(x);
    }

interrupt[PORT1_VECTOR] void port1key(void)
    {                                     // 端口 1 的中断服务程序
    while(p1keyj()! =0)
        {
        delay(500);                       // 延时消除抖动
        while(p1keyj()! =0)
```

```
            {
            keybuf   =   keycode();          // 确信有按键按下，找按键得键值，送到全局变量 keybuf
            while(p1keyj()== =0)             // 等待按键松开
                    ;
            }
        }
    }
```

以上两程序都使用中断方式，使用查询方式的程序类似，这里略，请读者自己编写。

4. 行列扫描式键盘

如果应用系统需要较多的按键，则采用独立式键盘的结构就不能满足需要了。如需 16 个按键，则独立式键盘将花费系统资源为 16 条 I/O 口线，显然是不科学的。这里使用行列扫描的方法实现键盘接口（也被称作矩阵键盘），则可使用少量的 I/O 口线连接较多的按键。图 5.15 为通过 MSP430 的 P1 口接的 4×4＝16 个按键（编号为 0～15）构成的行列扫描式键盘。下面分析如何在行列扫描式键盘上实现键盘的 3 个步骤：判键消抖动；键码识别；等待按下按键的松开。

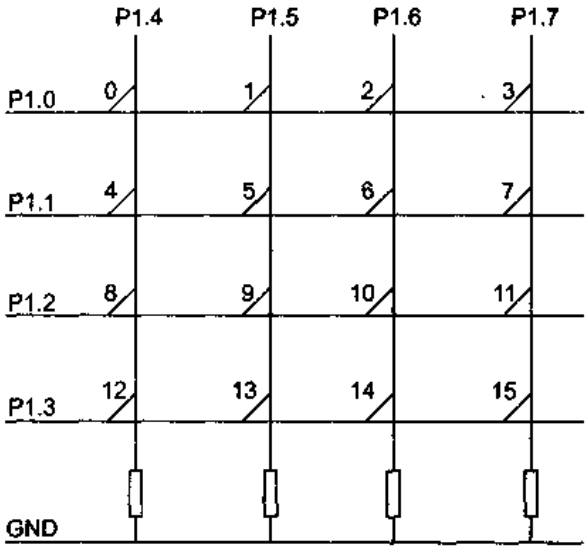


图 5.15　扫描键盘接口

（1）判断有无按键按下的判键

在图 5.15 中，P1 口的 8 条 I/O 口线被分为 4 条行线 P1.0～P1.3，4 条列线 P1.4～P1.7，其中列线分别由电阻下拉到地。在行线与列线的每一个交界处有个按键，按键的两端 A 和 B 分别接在行线和列线上。如果有按键按下，则与之相接的行线和列线被连通。在检测是否有按键按下时，先使 4 条行线 P1.0～P1.3 输出高电平，读列线 P1.4～P1.7；如果有按键按下，则列线读进来的数据为非 0；如果没有按键按下，则因所有的列线被下拉，读入 MCU 的数据为 0。由此即可判断是否有按键被按下，相应程序如下。

汇编程序：

```
KEYJUDGE  MOV.B     #0FFH,&P1DIR
          MOV.B     #0,&P1OUT
          MOV.B     #0FH,&P1DIR
          MOV.B     #0FH,&P1OUT      ; 输出行线高电平
          BIT.B     #0F0H,&P1IN      ; 测试 4 条列线是否有高，是否有按键
          RET                        ; 若有，C=1
```

C 语言程序：

```
unsigned char keyj(void)
    {unsigned char x;
    P1DIR=0XFF;
```

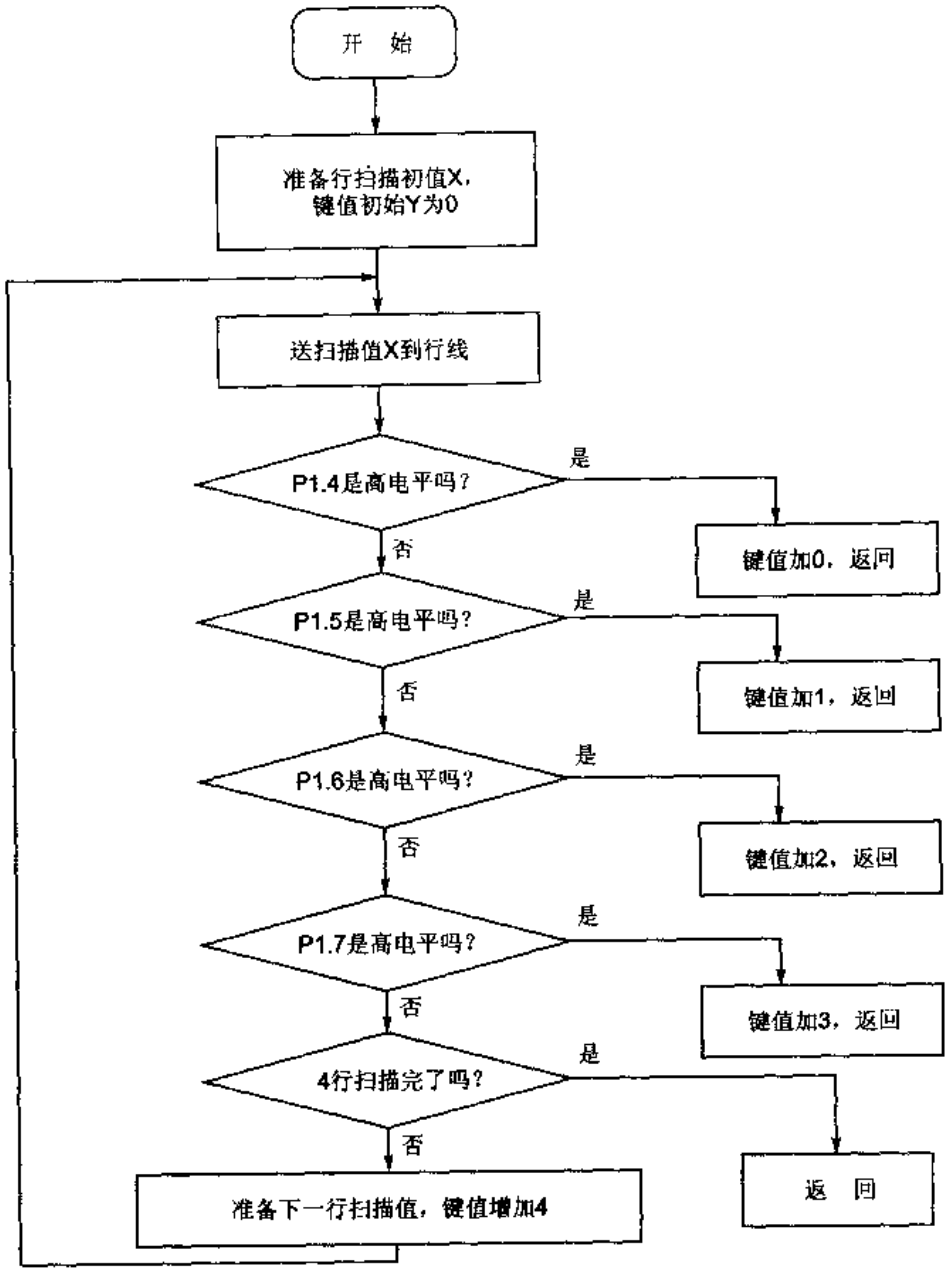


图 5.16　扫描式键盘的认键流程图

汇编程序如下：

```
KEYCODE        MOV.B    #0,&P5DIR          ；关闭显示
               MOV.B    #0FH,&P1DIR        ；低 4 位作为扫描线行输出，高 4 位作为列线读入
               MOV      #0,R9
               MOV      #1,R8
KEYCODELOOP    MOV.B    R8,&P1OUT          ；R8 为扫描信号的输出
               BIT.B    #10H,&P1IN
               JC       KEYCODE1           ；测试 P1.4
               BIT.B    #20H,&P1IN
```

```
                JC          KEYCODE2          ; 测试 P1.5
                BIT.B       #40H,&P1IN
                JC          KEYCODE3          ; 测试 P1.6
                BIT.B       #80H,&P1IN
                JC          KEYCODE4          ; 测试 P1.7
                RLA.B       R8                ; 改变扫描码
                ADD.B       #4,R9             ; 键值的行间改变
                CMP.B       #12,R9            ; 4 根行线扫描完了吗
                JNZ         KEYCODELOOP
                RET
KEYCODE1        ADD         #0,R9             ; 键值的行内改变
                RET
KEYCODE2        ADD         #1,R9
                RET
KEYCODE3        ADD         #2,R9
                RET
KEYCODE4        ADD         #3,R9
                RET
```

C 语言程序如下：

```
unsigned char key(void)                    // 此程序键盘为 4 行×3 列,12 个按键
  {
    unsigned char x=0xff;
    P1DIR=0X0F;
    P1OUT=0X01;                            // 扫描第一行
    if((P1IN&0X70)==0X10)                  // 如果第一行、第一根列线上有键按下,则键值为 0
      x=0;
      else
        if((P1IN&0X70)==0X20)              // 如果第一行、第二根列线上有键按下,则键值为 1
          x=1;
              else
                if((P1IN&0X70)==0x40)      // 如果第一行、第三根列线上有键按下,则键值为 2
                  x=2;
      else
       {
        P1OUT=0X2;                         // 扫描第二行
        if((P1IN&0X70)==0X10)              // 如果第二行、第一根列线上有键按下,则键值为 3
          x=3;
          else
            if((P1IN&0X70)==0X20)          // 如果第二行、第二根列线上有键按下,则键值为 4
              x=4;
                else
                  if((P1IN&0X70)==0x40) // 如果第二行、第三根列线上有键按下,则键值为 5
```

```
                x=5;
            else
            {
             P1OUT=0X4;                    // 扫描第三行,以下与上相同
             if((P1IN&0X70)==0X10)
             x=6;
               else
                 if((P1IN&0X70)==0X20)
                 x=7;
                   else
                   if((P1IN&0X70)==0x40)
                     x=8;
              else
              {P1OUT=8;                    // 扫描第四行
              if((P1IN&0X70)==0X10)
        x=9;
        else
           if((P1IN&0X70)==0X20)
             x=10;
                 else
                   if((P1IN&0X70)==0x40)
                     x=11;
                     }
              }
          }
    return(x);
    }
```

(3) 等待按键松开

这与独立式键盘一样，反复调用判键子程序，直到判断结果为没有按键按下为止。程序略，与前相同。

(4) 完整的扫描式键盘程序

上面所讲述的只是键盘程序的一些细节和实用键盘子程序，完整的键盘程序则有很多种方式。图 5.17 所示的键盘流程图是工作在查询方式的键盘程序，可以看出，采用此方式时，MCU 一直在查询有没有按键被按下，MCU 不能做其他的事情，只有等按下按键之后，

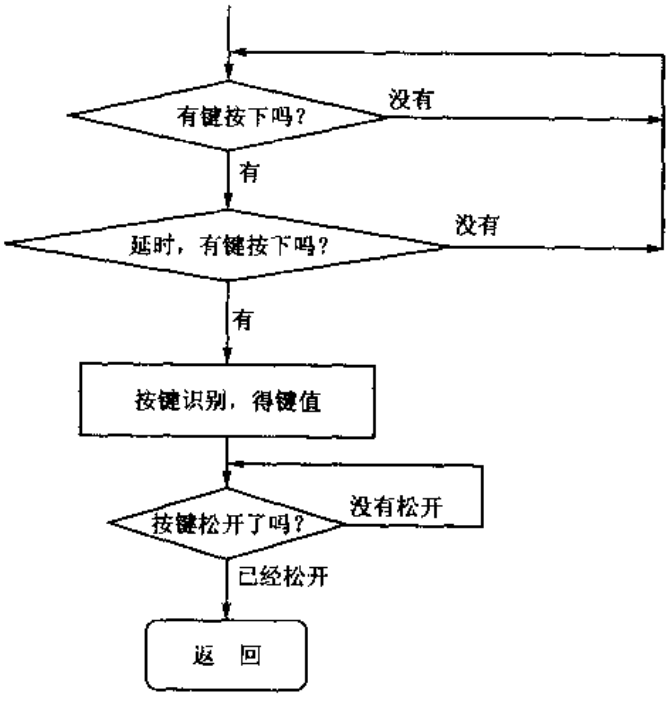


图 5.17　查询式键盘程序

才会有时间处理相应的事务。MCU 采用此方式效率很低。在实际应用中，键盘只是输入部分。它的功能是将用户的输入送给 MCU。为了提高 MCU 的效率，常使用中断的方式实现键盘输入。

中断方式的键盘实现又有两种方式：直接 I/O 口中断和定时器中断。它们的流程图分别见图 5.18 和图 5.19。

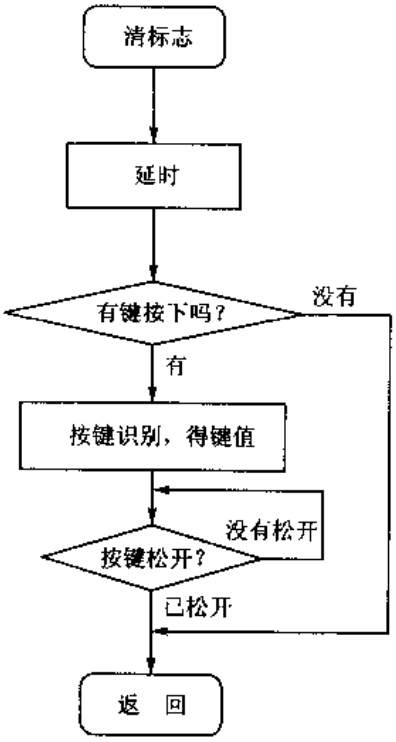


图 5.18 I/O 口中断方式键盘流程图

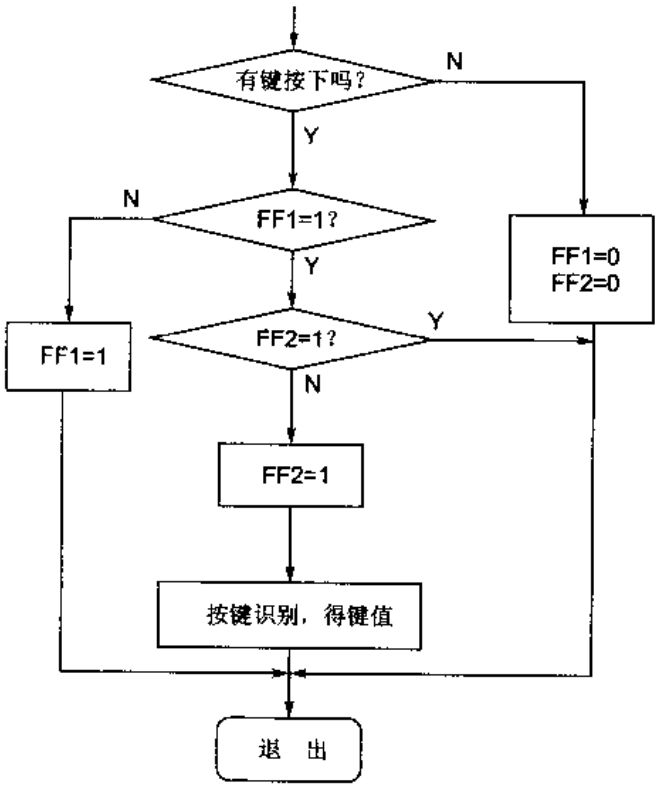


图 5.19 定时器中断方式键盘流程图

两流程框图的差别在于：使用 I/O 口线中断方式有延时消除抖动的过程；而使用定时器中断查键盘则没有延时去抖动细节，因为定时器的定时时间就是消抖动的时间，如 10 ms。设置两个标志位 FF1 和 FF2，其中 FF1 用于消抖动标志，FF2 用于按键识别完成标志。初始值都是 0，表示没有消抖动和完成按键的识别。当进入中断时，如果 FF1＝0，则没有消抖动，那么须判断是否有键按下；如有，设置 FF1＝1，并退出程序，等待下次中断（等 10 ms）；如果再次中断，则继续判断是否有键按下；如有，说明是真正的按键被按下，而且已是键盘的稳定期；调键码识别子程序得到键值，设置 FF2＝1，表示完成了按键的识别；如果没有按键被按下，则设置 FF1＝0，退出。如果这时再次中断，显然都是一次按键，FF2＝1，同时表明没有松开按键，也就退出了。等到按键被松开时，设置 FF1＝0 和 FF2＝0，为再次按键作准备。

使用 I/O 口中断方式键盘程序如下（使用定时器中断方式键盘程序请读者自己编写）：

```
P1KEY_INT   CALL    # KEYJUDGE          ；判断是否有按键
            JNC     KEYEND             ；没有则退出
            CALL    #DELAY10MS         ；如有，则延时 10 ms，消抖动
            CALL    # KEYJUDGE         ；再判键
            JNC     KEYEND             ；如没有按下则退出
```

```
                CALL    #KEYCODE        ; 如有,则调认键程序得到键值
                PUSH    R5              ; 保护键值
KEYLOOP         CALL    #KEYJUDGE       ; 等待按键松开
                JC      KEYLOOP         ; 没有松开,则继续等待
                POP     R5              ; 按键松开之后,恢复键值
KEYEND          RETI
```

5. N 线 N*(N+1)按键键盘的设计

有的应用场合,单片机的 I/O 口线相当紧缺,而又需要有较多按键的键盘,这时可使用在 N 条口线上连接 N*(N+1)个按键的方法予以解决,如图 5.20 所示。

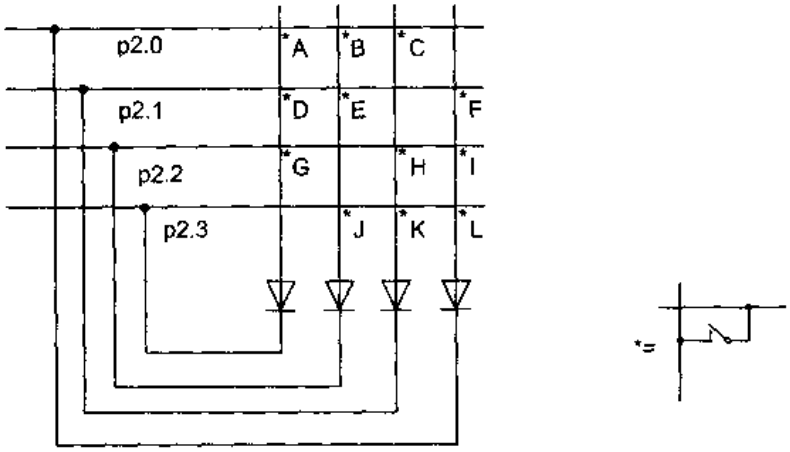


图 5.20　使用 4 条 I/O 口线连接 12 个按键

图中使用了 4 条 I/O 口线连接 12 个按键,这里只讨论如何区分这 12 个按键。当只有 P2.0 为高电平时,则只可能识别 A,B,C 三键,其他同理。在扫描时,各口线依次输出高电平,再转换为输入状态测试输入数据。下面为识别键码程序:

```
KEYCODE         MOV.B   #4,R5          ; 循环计数器
                MOV.B   #1,R6          ; 扫描初值
KEYCODELOOP     BIS.B   #0FH,&P2DIR    ;
                BIC.B   #0FH,&P2OUT    ; 先给 4 条口线输出低电平
                MOV.B   R6,&P2DIR      ;
                MOV.B   R6,&P2OUT      ; 依次扫描
                CMP.B   R6,&P2IN       ; 比较该口线上有没有按键按下
                JNZ     KEYCODEEND     ; 有,则计算是哪个按键
                RLA.B   R6             ; 没有,则继续扫描下一条
                DEC.B   R5             ; 循环计数器减 1
                JNZ     KEYCODELOOP    ; 4 条口线没有扫完,继续
KEYCODEEND      CALL    #KEYCOUN       ; 计算键值
                RET                    ; 返回
```

完整的 C 语言测试程序如下:(显示按键键值)

```
#include <msp430x13x.h>
unsigned char keybuff=0;
```

```
  return(0x00);                          // 没有按键返回 0
  }

unsigned char key4_12code(void)          // 扫描键码值
   {
   unsigned char hang=1,i,temp=0;
   for(i=0;i<4;i++)
     {
     P2DIR|=0x0f;                        // P2 低 4 位输出方式
     P2OUT&=~0x0f;                       // P2 低 4 位输出低电平
     P2OUT=hang;                         // 扫描信号
     P2DIR&=~(0x0f-hang);
     if((P2IN&0x0f)!=hang)               // 下面为逐行扫描,并找到按键的相应位置
          {
          if(((P2IN&0xf)-hang)==1)
            return(3+temp);
            else
              if(((P2IN&0xf)-hang)==2)
              return(2+temp);
              else
                  if(((P2IN&0xf)-hang)==4)
                  return(1+temp);
                  else
                      if(((P2IN&0xf)-hang)==8)
                      return(0+temp);
          }
     hang=hang<<1;
     temp=temp+4;
     }
   }

void key4_12(void)
  {
  unsigned char keytab[]={1,2,3,15,4,5,15,6,7,15,8,9,15,10,0,11};
  if(key4_12judge()!=0)
       {
       disp();
       disp();
       if(key4_12judge()!=0)
            {
            keybuff=key4_12code();
            keybuff=keytab[keybuff];       // 通过位置查表得键值
keywait;     if(key4_12judge()!=0)         // 等待按键松开
```

```
                goto keywait;
            }
        }
    }

void main(void)
{
  unsigned char x=5;
  WDTCTL = WDTPW + WDTHOLD;          /* Stop WDT */
  P5DIR=0XFF;
  P1DIR=0xff;                        /* P1.0 output */
  while(1)

  key1_12();
  disbuffer[5]=keybuff;
  disp();
  }
}
```

5.1.7 与存储器的接口设计

MSP430目前的所有型号总线对外不开放，只能通过I/O口与存储器的三总线相连接，使用通用I/O口线模拟存储器的读写时序完成与存储器接口。对于MSP430来说，程序存储器应该够用了，最大有60 KB的ROM空间，在以后的型号中，使用分页方式，还会增加很多。扩展存储器主要是增加数据存储器，因为MSP430的RAM最大只有2 KB。对于FLASH型号来说，片内FLASH存储空间一样可用做RAM，但速度较慢。这里主要讲述与数据存储器的接口。常用的数据存储器有：静态存储器(SRAM)、动态存储器(DRAM)及掉电后数据不丢失的非易失类存储器(E^2PROM)和FLASH等。

1. MSP430与静态存储器(SRAM)的接口

这里以静态存储器6264为例。6264是采用CMOS工艺的8 K×8位的28引脚静态读写存储器。其读写访问时间因具体型号而异，一般在20～200 ns之间。数据输入与输出引脚I/O0～I/O7为共用，三态输出；A0～A11为地址输入引脚；控制信号为CE1和CE2两片选，WE写允许，OE输出允许，所有控制引脚除CE2外都是低电平有效。其工作方式如表5.3所列。

表5.3 6264的工作方式

WE	CE1	CE2	OE	I/O0～I/O7	工作方式
×	H	×	×	高阻	未选通
×	×	L	×	高阻	未选通
H	L	H	H	高阻	输出禁止
H	L	H	L	数据输出	读操作
L	L	H	H	数据输入	写操作
L	L	H	L	数据输入	写操作

MSP430 与 6264 按图 5.21 所示的方式连接。

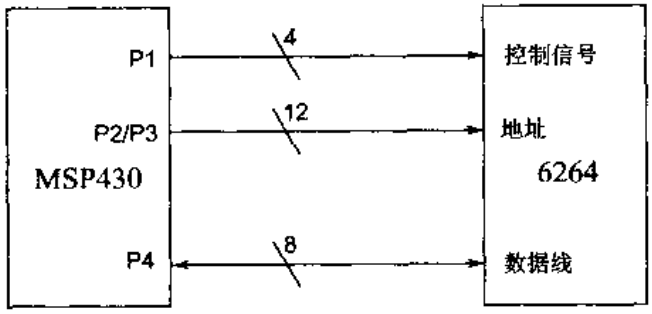


图 5.21　MSP430 与 6264 的连接

用 MSP430 的 P1.0,P1.1,P1.2 及 P1.3 分别接 6264 的 WE,CE1,CE2 及 OE;MSP430 的 P2 和 P3 两个 8 位端口接 6264 的地址线,其中 P2 接地址线的低 8 位;MSP430 的 P4 口与 6264 的数据线相连接。对 6264 的读写操作,则完全按照 6264 所提供的功能进行。下面是相应的程序:

```
;读 6264 子程序
;程序入口参数:R5 为要读 6264 的地址
;      出口参数:R5 为从指定地址中读出的 6264 中的数据
READ_6264    BIS.B    #0FH,    &P1DIR
             MOV.B    #00H,    &P4DIR      ;P4 为数据读入端口
             MOV.B    #0FFH,   &P2DIR
             MOV.B    #0FFH,   &P3DIR      ;指定与 6264 相连的端口方向
             MOV.B    R5,      &P2OUT      ;写入低 8 位地址
             SWPB     R5
             MOV.B    R5,      &P3OUT      ;写入高 8 位地址
             BIC.B    #0AH,    &P1OUT
             BIS.B    #5H,     &P1OUT      ;指定 6264 的工作方式为读
             MOV.B    &P4IN,   R5          ;读出数据
             RET

;写 6264 子程序
;程序入口参数:R5 为要写 6264 的地址
;             R6 为要写入 6264 指定地址的数据
WRIT_6264    BIS.B    #0FH,    &P1DIR
             MOV.B    #0FFH,   &P4DIR
             MOV.B    #0FFH,   &P2DIR
             MOV.B    #0FFH,   &P3DIR      ;指定与 6264 相连的端口方向
             MOV.B    R5,      &P2OUT      ;写入低 8 位地址
             SWPB     R5
             MOV.B    R5,      &P3OUT      ;写入高 8 位地址
             BIC.B    #03H,    &P1OUT
             BIS.B    #0CH,    &P1OUT      ;指定 6264 的工作方式为写
             MOV.B    R6,      &P4IN       ;将数据写入
             RET
```

2. MSP430 与动态存储器(DRAM)的接口

静态存储器与单片机的连接将使用太多的 I/O 口线,以上与 6264 的接口就很明显,用了 MSP430 三个半的 8 位端口。在大多数情况下这是不允许的,除非系统需要高速的、大量的数据存取。下面讲述与动态存储器的连接。

动态存储器由于在地址线上分行线和列线,所以地址线减少了一半,同时每一个存储单元为单管结构,存储容量也可能很大。静态的 6264 为 8 K×8 位结构,用掉了 28 根口线。下面介绍的动态存储器 TMS44400/P 为 1 M×4 位结构,存储容量为 6264 的 64 倍,只要 18 条I/O 口线。

TMS44400/P 为 3.3 V 工作的 CMOS 动态存储器,能寻址 1 M 地址空间,数据总线为 4 位宽度,典型的存取时间为 70 ns,有较低的功耗。地址线为行列 A0～A9 复用。数据线为 DQ0～DQ3,三态输出。控制线有:G 输出允许,低电平有效;W 为读/写允许线,高电平选择读方式,低电平选择写方式;RAS 行地址选通线,低电平有效;CAS 列地址选通线,低电平有效。内部有 32 个 128 K×8 位的存储阵列、行列地址缓存、行列地址译码、输出单元和定时等部件。内部结构如图 5.22 所示。

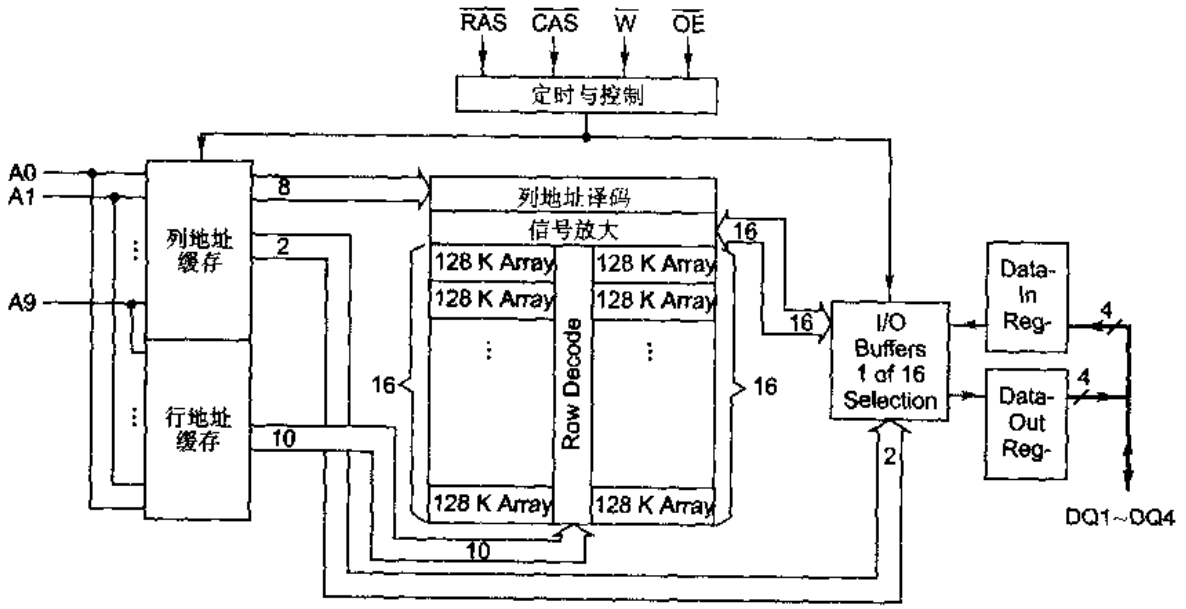


图 5.22 TMS44400/P 的内部结构原理图

对 TMS44400/P 的操作可以分解为几个过程:芯片的初始化;读操作;写操作;刷新操作。芯片初始化是指芯片上电后,为了确保内部基片能很好地建立偏置电压,需延迟 200 μs;接着使行地址选通信号至少保持 8 个工作周期有效,以此对所有的 RAM 单元初始化。而读、写及刷新操作都需要对芯片内部 RAM 单元寻址。下面先讲如何寻址内部单元。

芯片对外总共有 10 条地址线,但为行列复用,整个 10 位行地址与 10 位列地址构成了 20 位地址,可寻址 1 M 单元地址。对所有的读写操作总是先 RAS 有效,再 CAS 有效。如要寻址第 0 单元的 4 位数据,则寻址方法为:先给 10 位地址线的数据为 0,然后 RAS 有效(低电平);再给 10 位地址线的数据为 0,然后 CAS 有效(低电平)。

图 5.23 是 TMS44400 与 MSP430 的接口电路图。图中使用的是 MSP430X3XX 系列器件。

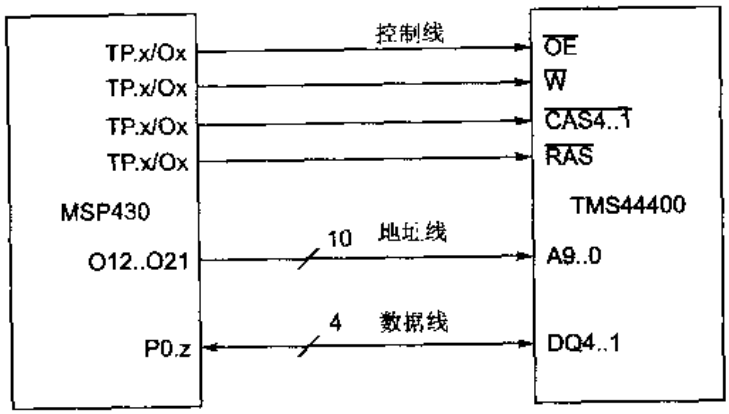


图 5.23　TMS44400 与 MSP430 的接口电路

对存储器的操作(读、写及刷新)可按图 5.24～5.27(摘自 TMS44400 DATABOOK)所示的时序进行。

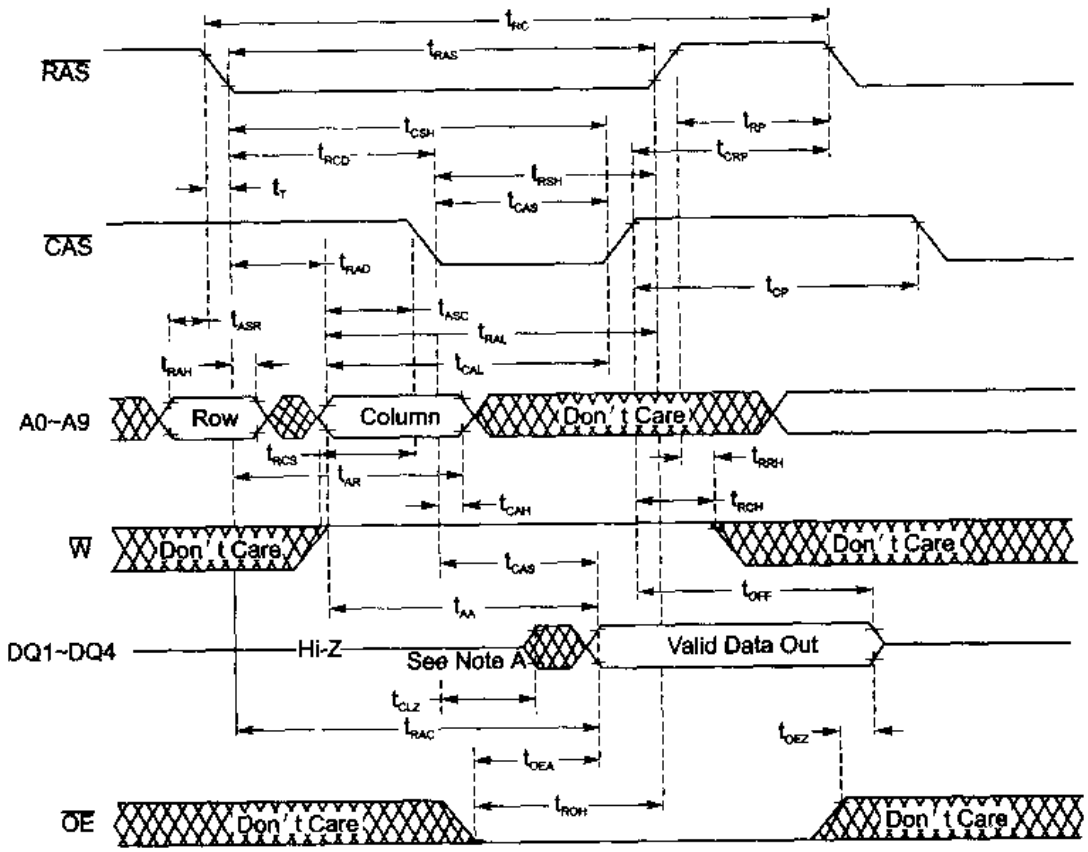


图 5.24　TMS44400 读操作时序

读/写操作举例：

① 输出 10 位行地址；

② 使能行信号为低；

③ 输出 10 位列地址；

④ 使能列地址；

⑤ 设置 OE=0，W=1(读)；

⑥ 读取数据。

或

⑤ 设置 OE=1，W=0(写)；

⑥ 将要写入的数据送到数据端口。

程序略。

5.1.8　MSP430 与模数转换器的接口

大多数 MSP430 器件都有 10～14 位模数转换器，它们的使用在第 3 章有具体的介绍。有少量的 MSP430 器件没有片内模数转换器模块，要使用模数转换器则只有外接其他器件了。这里讲述模数转换器 TLV0831/2 与 MSP430 的接口。

TLV0831/2 是 8 位逐次逼近型模数转换器，有单路或两个可多路选择的模拟输入通道，与微处理器串行接口。其多路器可软件配置为单端或差分输入。差分的模拟电压输入可共模抑制和使模拟输入电压偏移值为 0。另外输入基准电压可调整大小，在 8 位分辨率下允许任意小的模拟电压编码间隔。其特性简述如下：

- 8 位分辨率；
- 电源电压 2.7～3.6 V，能直接与 MSP430 接口；
- 满比例转换或以 V_{cc} 为基准，以 V_{cc} 为基准时，输入范围为 0 V～V_{cc}；
- 单通道或双通道输入，可单端输入或差分输入；
- 在时钟为 250 kHz 时，转换时间为 32 μs；
- 输入和输出与 TTL 或 CMOS 兼容。

TLV0831 与处理器接口的只有 3 条线：片选线 CS、数据输出线 DO 及时钟输入线 CLK。当设置 CS=0 时，使能所有逻辑电路启动一次转换开始，注意在整个转换过程中都必须为低；紧接着由处理器提供一个时钟，TLV0831 转换出一位数据由 DO 端输出，先转换出最高位；经过 8 个时钟后，转换完成，所有的 8 位转换值都输出了。当片选端 CS 为高电平时，TLV0831 内部的所有寄存器被清零复位，此时输出电路变为高阻态。如果要继续转换，则重复此过程：CS 变低，给时钟，读数据……。TLV0831 的操作时序如图 5.28 所示。

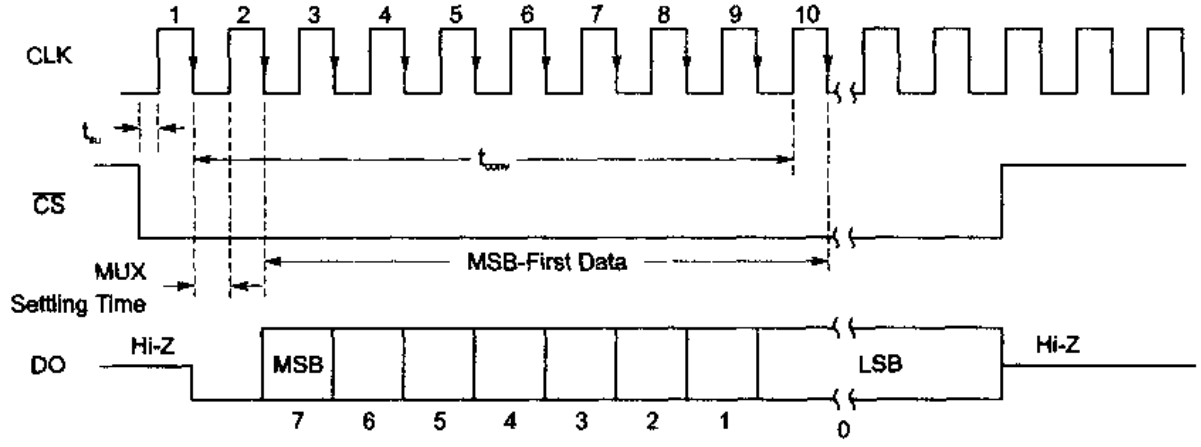


图 5.28　TLV0831 的操作时序

使用 MSP430 的 P1 口低位与 TLV0831 相连，如图 5.29 所示。

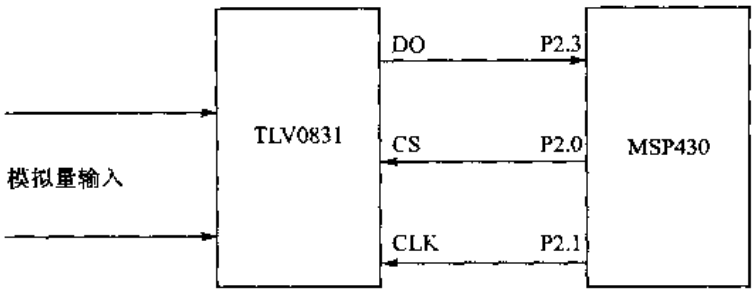


图 5.29 TLV0831 与 MSP430 接口图

相应的程序如下(使用 MSP430F1121 的完整汇编程序)：

```
#include   "msp430x11x1.h"
CS          equ     001h            ; P2.0 被定义为 CS 端
CLK         equ     002h            ; P2.1 被定义为 CLK 端
DO          equ     008h            ; P2.3 被定义为数据输出端

            ORG     0F000h          ; MSP430F1121 程序开始处
RESET       mov     #0300h,SP       ; 初始化堆栈指针
            bis.b   #CS,&P2OUT      ; CS=1
            bis.b   #CS+CLK,&P2DIR  ; CS 和 CLK 定义为输出方向
Mainloop    call    #Meas_0831      ; 循环调用 0831 子程序，这里只为示例
            jmp     Mainloop
            ;0831 测量子程序，转换的数据存放在内部 RAM  200 H 单元
Meas_0831
            bic.b   #CS,&P2OUT      ; 片选 CS=0，开始转换
            push.b  #09             ; 循环计数器，一位开始，八位数据
ADC_Loop    bis.b   #CLK,&P2OUT     ; CLK=1
            bic.b   #CLK,&P2OUT     ; CLK=0
            bit.b   #DO,&P2IN       ; DO → C (carry)，读转换结果
            rlc.b   &200h           ; C →200H 依次存放到 200H 单元
            dec.b   0(SP)           ; 9 次循环完了吗？
            jnz     ADC_Loop        ; 没有完，则继续
            incd.w  SP              ; 还原堆栈指针
            bis.b   #CS,&P2OUT      ; CS=1
            ret
            ORG     0FFFEh          ; RESET 向量地址
            DW      RESET           ; RESET 向量数据
            END
```

如果分辨率不够高，可选 12 位模数转换器 MAX1240 等其他器件，其用法与 TLV0831 类似。MAX1240 的电源电压为 2.7～3.6 V，能直接与 MSP430 共电源，提供内部 2.5 V 参考电源，也可接外部参考源，并提供低功耗模式，通过引脚设置。图 5.30 为 MAX1240 操作时序，软件与 TLV0831 类似，只需少量修改：一个完整的转换周期需 16 个时钟，读出数据为 12 位。

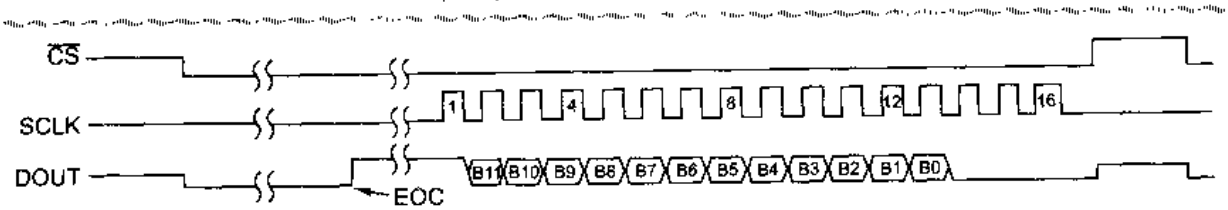


图 5.30　MAX1240 操作时序

5.1.9　MSP430 乐音的输出

利用单片机输出乐音是很有用的。如在数据采集中，当缓冲器存满时，输出一段乐音提示操作人员操作；在控制系统中，当被控制量超出警戒值，输出一段乐音用于报警；在玩具中就更有用了。那么乐音是怎样产生的呢？

先分析扬声器的发声原理。扬声器发声是因为扬声器里处于磁场中的音频线圈（音圈）在电流的驱动下，音圈产生力，发生运动。由于音圈的运动，带动音圈上的纸盆运动。音圈的运动频率就是纸盆的运动（振动）频率。纸盆在一定频率下振动，就会发出一定频率的声音。一般情况下，频率指的是送给音圈的正弦电流频率。当送给音圈一定频率的正弦电流，则扬声器发出一定频率的声音，这样是很单调的，不是乐音，而是刺耳的噪音。为了解决这个问题，常改为送给音圈一个以一定频率为基波，并带有该频率的丰富谐波的电流信号，则扬声器发出的声音就很悦耳了。利用单片机发生方波是很简单的事情，方波是个由一个基波与非常多的谐波构成的多谐波。用这个波形驱动扬声器则会发出非常悦耳的乐音，即用单片机输出基波频率的方波给扬声器即可。

MSP430 产生方波在定时器部分已经有了详细讲解，这里以中音 DO 为例。中音 DO 的频率为 261.6 Hz，定时器工作在连续模式，在定时器中断服务程序中：将输出口线求反；增加重复定时时间间隔的数据到 CCR0；选用主时钟 MCLK 为定时器输入时钟，在默认条件下，MCLK 的频率大致为 1 MHz，同时分频系数选取 1，即不分频。下面计算应该增加到 CCR0 中的是数据 N。

中音 DO 的周期：　　$1/261.6 = 3\ 823\ \mu s$

在定时器中使用输出线求反的方法，实质上定时器的定时时间间隔应该是输出周期的一半，即

$$3\ 823\ \mu s\ /\ 2\ =\ 1\ 912\ \mu s$$

定时器的最终输入时钟 Timer 时钟为 1 MHz，周期为 1 μs，那么定时器 CCR0 应该增加的数据 N 为

$$1\ 912\ \mu s\ /\ 1\ \mu s = 1\ 912\ (778H)$$

同理可以计算出其他频率的音阶相对应的数据 N，如表 5.4 所列。

表 5.4　C 调各音阶与定时器增到 CCR0 中数据对照表

低　音	对应 N	中　音	对应 N	高　音	对应 N
1	0EEFH	1	0778H	1	03BCH
2	0D4EH	2	06A7H	2	0354H
3	0BDAH	3	05EDH	3	02F7H
4	0B30H	4	0598H	4	02CCH
5	09F8H	5	04FCH	5	027EH

续表 5.4

低 音	对应 N	中 音	对应 N	高 音	对应 N
6	08E1H	6	0471H	6	0239H
7	07F9H	7	03F5H	7	01F8H

在下面的例程为乐曲《两只老虎》(儿歌)的完整放音程序。程序较为笨拙,但很容易理解。相信读者在理解程序的基础上很快就能在 MSP430 的应用中输出所喜欢的音乐声了。

```
;两只老虎的音乐程序
#include    "msp430x11x1.h"
            RSEG    CSTACK                  ; 系统堆栈
            DS      0
            RSEG    CODE                    ; 程序代码的开始
Reset       mov     #SFE(CSTACK),SP         ; 初始化堆栈指针
SetupWDT    mov     #WDTPW+WDTHOLD,&WDTCTL  ; 停看门狗定时器
SetupTA     mov     #TASSEL1+TACLR,&TACTL   ; 初始化 Timer A
SetupC0     mov     #CCIE,&CCTL0            ; 使能 CCR0 中断
            mov     #1000,&CCR0             ; CCR0 给一个初始值
SetupP1     bis     #MC1,&TACTL             ; 定时器工作在连续模式
            MOV.B   #8,&P4DIR               ; 乐音最终由 P4.3 输出
            EINT                            ; 开中断
h;          MOV     #0778H,&200H            ; 1 的频率数据暂存于 200H 单元
            CALL    #dely                   ; 1 的节拍,4 分音符
            MOV     #06A7H,&200H            ; 2 的频率数据暂存于 200H 单元
            CALL    #dely
            MOV     #05EDH,&200H            ; 3
            CALL    #dely
            MOV     #0778H,&200H            ; 1
            CALL    #dely
            DINT                            ; 关中断,也就关了输出,起到休止符
                                            ; 号的作用
            CALL    #delyyy
            EINT                            ; 休止符号结束,继续允许发出音乐
            MOV     #0778H,&200H            ; 1
            CALL    #dely
            MOV     #06A7H,&200H            ; 2
            CALL    #dely
            MOV     #05EDH,&200H            ; 3 的频率数据暂存于 200H 单元
            CALL    #dely
            MOV     #0778H,&200H            ; 1—
            CALL    #dely                   ; 1 的节拍,2 分音符,该音的时值长,
                                            ; 则两次延时
            CALL    #dely                   ; 如果更长,则更多次调用
```

```
                MOV     #0778H,&200H            ;1
                CALL    #dely
                MOV     #09F8H,&200H            ;5
                CALL    #dely
                MOV     #0778H,&200H            ;1—
                CALL    #dely
                CALL    #dely
                DINT
                CALL    #dely
                EINT
                JMP     h                       ;循环放音,当然也可在此设置按键,
                                                ;确定是否再放音
delyy:          MOV     #40000,     R5          ;时间稍微短一点的基础节拍
DELY11          DEC     R5                      ;在主程序中直接调用,调用多次则
                                                ;产生较长的节拍
                JNZ     DELY11
                RET
delyyy:         MOV     #10000,     R5          ;时间最短的节拍,用在休止符上
DELY111         DEC     R5
                JNZ     DELY111
                RET
dely:           MOV     #60000,R5               ;主要基础节拍
DELY1           DEC     R5
                JNZ     DELY1
                RET
TA0_isr         XOR.B   #08H,&P4OUT             ;简洁的定时器中断服务程序:输出
                                                ;求反
                ADD     &200H,&CCR0             ;产生音频:增加到比较器 CCR0 以
                                                ;稳定数据
                RETI

                COMMON  INTVEC                  ;中断向量表
                ORG     TIMERA0_VECTOR
                DW      TA0_isr
                ORG     RESET_VECTOR
                DW      Reset
                END
```

5.2 综合应用设计

在前面讲述了 MSP430 的一些单元应用设计,相对较为简单。这里将讲述一些较为复杂的器件与 MSP430 的接口,以及如何将一些相对简单的应用组合成较为复杂的应用。

5.2.1　MSP430 与 I²C 总线方式的 E²PROM 接口

在单片机应用系统中，使用 I²C 总线方式的 E²PROM 存储器可以在节省系统资源的情况下增加存储容量。它与单片机只需两根口线连接，而存储容量可大到 1 M 位(128 K×8 位)；但与单片机通信的速度不高，适用于数据记录与常数保存。这里以 24CXXX 为例讲解如何与 MSP430 接口。

24CXXX 系列 E²PROM 存储器目前的型号有 24C01，24C02，…，24C256，24C512 及 24C1024，存储容量为从 256 位到 1 M 位。它们与单片机的硬件接口完全一样，只是软件上因器件内部寻址范围不同而不同。下面以存储容量较大的 24C256 为例讲解与 MSP430 的接口方法。

1. 24C256 特性

- 与 1 MHz I²C 总线兼容；
- 低功耗 CMOS 技术，电源电压范围为 1.8～6 V；
- 32 K×8 位存储空间，64 字节页写缓冲器；
- 硬件写保护功能，当 WP=1 时为写保护状态；
- 可进行 100 000 次编程与擦写，数据可长期保存不丢失，适用于数据记录和保存；
- 最多可 4 片级联，容量可扩展为 128 KB。

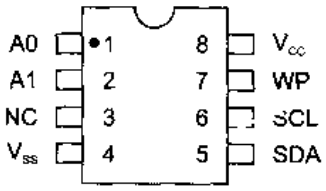


图 5.31　引脚图

24C256 一共 8 个引脚，如图 5.31 所示。其中：

SCL　串行时钟输入引脚，用于产生器件收发数据所需的时钟信号；

SDA　双向数据引脚，用于输入/输出数据，开漏输出，需上拉；

WP　写保护引脚，当它为高电平时，内部数据被硬件写保护了，不能再写进；

A0，A1　器件地址输入引脚，可同时在总线上连接同样的 4 个器件。

24C256 的内部结构如图 5.32 所示。它的使用必须遵循 I²C 总线协议，才能对其正确操作。I²C 协议规定如下：

① 只有总线空闲时，才允许启动数据传送。

② 在数据传送过程中，当时钟线为高电平时，数据位必须在数据线上保持稳定状态，不允许有跳变。时钟线为高电平时，数据线的任何电平变化将被看作是总线的启动信号或停止信号。

2. 24C256 的操作

(1) 总线的启动与停止

在时钟线保持为高电平期间，当数据线有下降沿时，I²C 总线被启动，数据可以传送；在时钟线保持为高电平期间，当数据线有上升沿时，I²C 总线被停止，数据不可以传送。如图 5.33 所示。

(2) 器件寻址

24C256 在应用系统中作为从器件被主器件寻址。主器件通过发送启动信号启动发送过程，然后发送它所要寻址的从器件的地址。对于 24C256 而言，8 位从器件地址的高 5 位被固定为 10100，接下来的 2 位(A1，A0)为器件引脚的连接信息，在软件中的地址信息必须与器件的物理连接相符。从器件地址的最低位为读写控制位。1 表示对从器件的读操作，0 表示对从

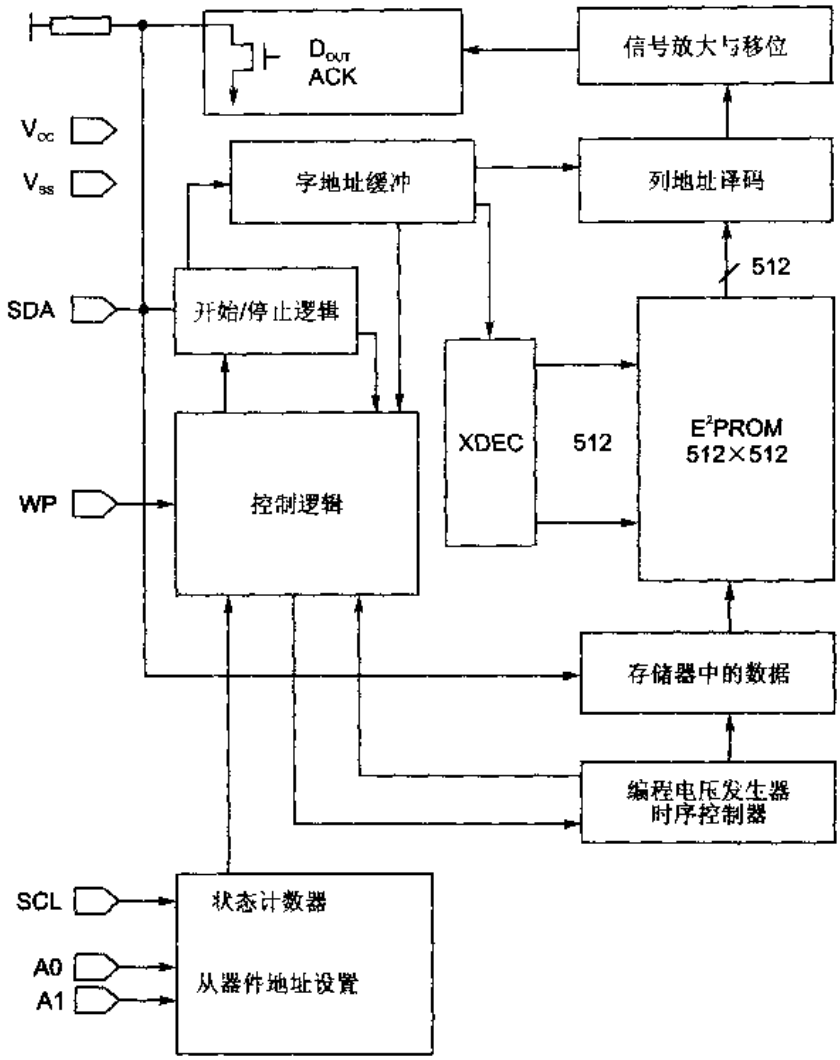


图 5.32 24C256 的内部结构

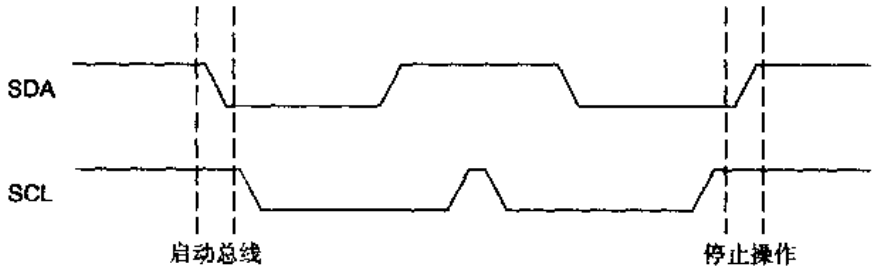


图 5.33 24C256 的启动与停止

器件的写操作，即

1	0	1	0	0	A1	A0	读/写

当主器件发送了总线启动信号和读或写的从器件地址后，24C256 监视总线，并在当其自身地址与发送来的地址相符时，通过 SDA 响应应答信号，再根据读写位的状态进行相应的读/写操作。

(3)应答信号

I²C 总线每成功地传送一个字节数据后，接收器都必须产生一个应答信号。应答的器件在第 9 个时钟周期时将 SDA 线拉低，表示已经收到 8 位数据。注意，无论对主器件还是从器件都是一样的。在软件上，如果主器件发送 8 位数据后，则从器件发出应答信号，主器件将注意接收应答信号；如果主器件接收了 8 位数据，则主动发出应答信号，只有 24C256 接收到应答信号后，才继续发送数据。图 5.34 为应答信号时序。

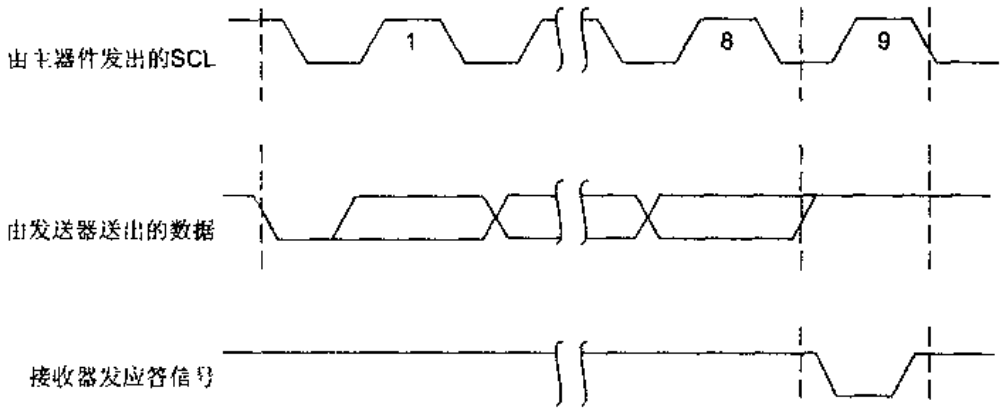


图 5.34　应答信号时序

(4) 写操作

对 24C256 的写操作有两种方式：字节写方式和页写方式。

在字节写方式下，主器件发送启动信号和从器件地址信息(以及写)后，从器件发出应答信号；主器件收到应答信号后，确定已经与从器件通信联系上了，于是开始发送要写的 24C256 片内具体地址，这个地址为 15 位，分 2 次，先发送高字节地址，等到应答信号后，再发送低字节地址信号；再次等待应答信号；收到应答信号之后，主器件送出要写的 8 位数据；同样再等待应答信号；收到应答信号之后，主器件发送停止信号，整个过程结束。在主器件发出停止信号之后，24C256 进入写周期，等待写完之后，才可能响应新的请求。字节写操作时序如图 5.35 所示。

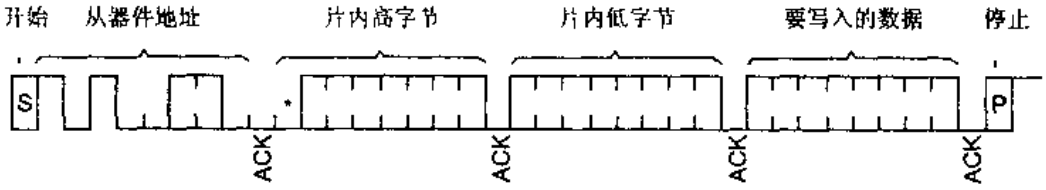


图 5.35　字节写时序

在页写方式下，可以在一个写周期内最多写入 64 个字节。页写操作的启动与字节写操作一样，不同之处在于当主器件送出一个字节数据后，还可继续送出 63 个字节，以写入 24C256 的后续地址(地址自动增加)。要注意的有两点：主器件每发送一个字节后，都将等待应答信号；每次最多 64 字节，否则将覆盖 64 字节前面的数据。页写时序图如图 5.36 所示。

(5) 读操作

当器件寻址字节中的最低位为 1 时，表示主器件对 24C256 的读操作。读操作有 3 种方

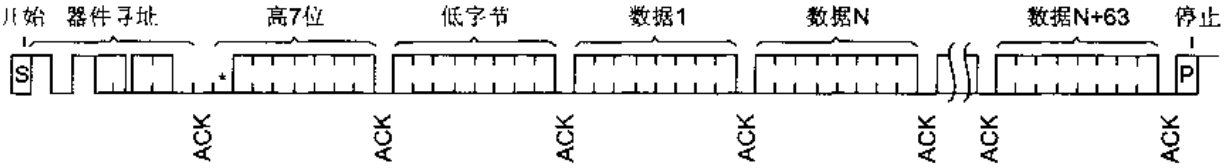


图 5.36 页写方式时序

式：立即/当前地址读、随机/选择读及连续读。

立即/当前地址读方式不用操作器件内部地址，要操作的内部地址为默认地址，即上次读写操作字节的地址加 1 开始。或者说，如果上次操作的字节地址为 N，则立即/当前地址读将从 N+1 地址开始操作。如果 N=32 767，则将从地址 0 开始读操作。当从器件将 8 位数据输出完毕，主器件可发送停止信号，时序如图 5.37 所示。

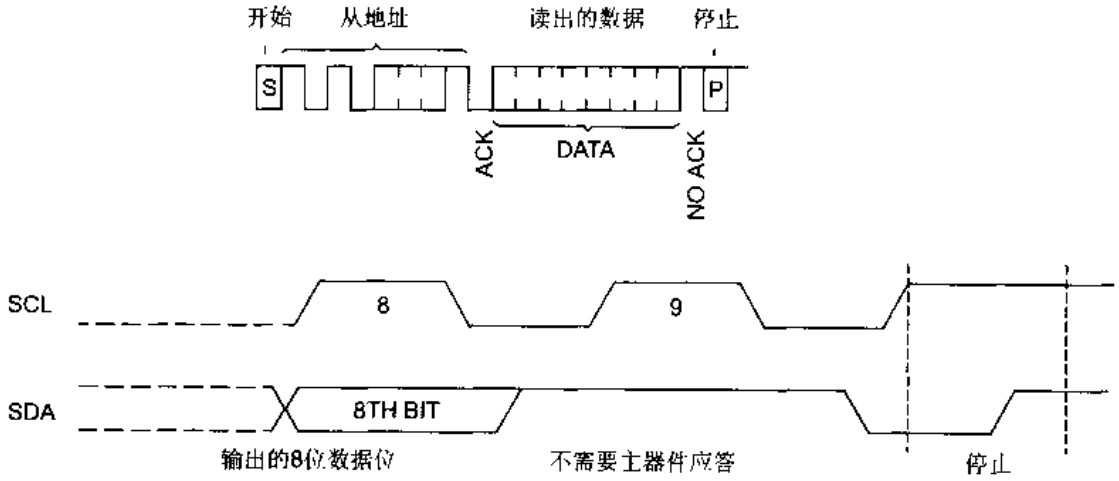


图 5.37 立即/当前地址读时序

随机/选择读方式允许主器件由从器件的任意地址读出数据。具体的操作过程为：主器件发出启动信号；寻址从器件地址(此时为写)；送出要寻址的从器件内部具体地址；然后再发出启动信号；再重新寻址从器件地址(此时为读)；接收从器件送出的 8 位数据；主器件发出停止信号。整个过程完毕。具体时序如图 5.38 所示。

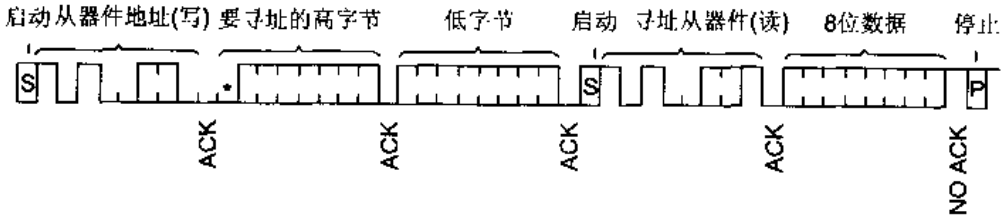


图 5.38 随机/选择读时序

连续读方式可由立即读或选择读启动操作。在 24C256 发送完成 8 位字节数据后，主器件产生应答信号(注意：其他两种读方式主器件不产生应答信号)，告知 24C256 主器件需求更多数据，请求从器件输出。对于每一个主器件发出的应答信号，从器件都将送出一个 8 位字节数据，当主器件不再送出应答信号，而是发出停止信号时，操作结束。当从器件 24C256 每输出一个字节数据后，内部地址自动增 1，当增加到 32 767 时，由地址 0 开始继续输出数据。连

(4) 向总线发送一字节数据

```
I2C_TX      mov. b    #08,BITI2C          ;向总线发送一个字节的数据
I2C_Send    rla. b    RXTXI2C             ;
            jc        I2C_Send1           ;
I2C_Send0   bis. b    #SDA,&P4DIR         ;发送 1
            jmp       I2C_Sx              ;
I2C_Send1   bic. b    #SDA,&P4DIR         ;发送 0
I2C_Sx      bic. b    #SCL,&P4DIR         ;
            CALL      #I2CDLY             ;
            bis. b    #SCL,&P4DIR         ;
            dec. b    BITI2C              ;
            jnz       I2C_Send            ;没有发送完 8 位,则继续
            bic. b    #SDA,&P4DIR         ;
I2C_Ackn    bic. b    #SCL,&P4DIR         ;应答信号
            CALL      #I2CDLY             ;延时
            bis. b    #SCL,&P4DIR         ;
            ret
```

(5) 接收 I^2C 总线上的 8 位数据

```
Read_I2C    mov. b    #Code,RXTXI2C       ;在具体的 I²C 器件中读某一个地址的数据
            call      #I2C_Start          ;启动子程序包含芯片寻址(写操作)
            mov. b    ADDRI2C1,RXTXI2C    ;送要读数据所在的高字节地址
            call      #I2C_TX             ;
            mov. b    ADDRI2C0,RXTXI2C    ;送要读数据所在的低字节地址
            call      #I2C_TX
            mov. b    #Code,RXTXI2C       ;
            add. b    #1,RXTXI2C          ;指示读操作,最低位为 1
            bis. b    #01h,RXTXI2C        ;
            call      #I2C_Start          ;再次以读方式启动总线
            call      #I2C_Read           ;读出 24C256 输出的数据
            call      #I2C_Stop           ;停止操作
            ret
```

(6) 24C256 与 MSP430 接口总的完整测试程序

测试程序的使用方法:R6 和 R7 为要写入 24C256 片内的地址,R8 为要写入 24C256 片内地址的数据,R9 为从 24C256 内部指定(由 R6 和 R7 指定)地址的数据。源程序如下:

```
;I²C 器件的测试程序
;使用器件为 24C256
;硬件连接为 P4.2,P4.7
#include   "msp430x13x.h"
RXTXI2C     equ       202h              ;要发送的数据缓存
ADDRI2C1    equ       204h              ;片内被寻址高地址字节
ADDRI2C0    EQU       205H              ;片内被寻址低地址字节
```

```
DATAI2C     equ     206h                ; 数据
BITI2C      equ     207h                ; 数据位计数器
SDA         equ     004h                ; 数据线连在 P4.2 上
SCL         equ     080h                ; 时钟线连在 P4.7 上
Code        equ     0A0h                ; 定义 24C256 器件的寻址常数(器件地址)
R_I2C       EQU     215H                ; 接收数据缓存器

            ORG     0F000h              ;

RESET       mov.w   #300h,SP
MAIN        MOV.B   R7,ADDRI2C0         ; 要写入的器件内部低地址
            MOV.B   R6,ADDRI2C1         ; 要写入的器件内部高地址
            MOV.B   R8,DATAI2C          ; 要写入的具体数据
            CALL    #Write_I2C
            MOV     #01FFFH,R15
DLY0        DEC     R15
            JNZ     DLY0                ; 延时,等待写周期的结束
            MOV.B   R7,ADDRI2C0         ; 要读的器件内部低地址
            MOV.B   R6,ADDRI2C1         ; 要读的器件内部高地址
            CALL    #Read_I2C           ; 读出刚写入的数据
            mov.b   R_I2C,R9            ; 刚读出的数据转存到 R9
            JMP     MAIN                ; 主循环,这只是一个测试程序

I2CDLY      NOP                         ; 延时,根据时钟频率,NOP 可增可减
            NOP
            RET

Read_I2C    mov.b   #Code,RXTXI2C       ; 在具体的 I²C 器件中读某一个地址的数据
            call    #I2C_Start          ; 启动子程序包含芯片寻址(写操作)
            mov.b   ADDRI2C1,RXTXI2C    ; 送要读数据所在的高字节地址
            call    #I2C_TX             ;
            mov.b   ADDRI2C0,RXTXI2C    ; 送要读数据所在的低字节地址
            call    #I2C_TX
            mov.b   #Code,RXTXI2C       ;
            add.b   #1,RXTXI2C          ; 指示读操作,最低位为 1
            bis.b   #01h,RXTXI2C        ;
            call    #I2C_Start          ; 再次以读方式启动总线
            call    #I2C_Read           ; 读出 24C256 输出的数据
            call    #I2C_Stop           ; 停止操作
            ret

I2C_Start   bic.b   #SCL+SDA,&P4DIR     ; 启动总线
            bis.b   #SDA,&P4DIR         ;
```

```
                bis. b      #SCL,&P4DIR          ;
                CALL        #I2CDLY              ;
                CALL        #I2C_TX              ; 同时写一字节数据到总线
                RET

Write_I2C       mov. b      #Code,RXTXI2C        ; 向 I²C 器件的某一地址写入具体的数据
                call        #I2C_Start           ; 启动总线同时寻址芯片
                mov. b      ADDRI2C1,RXTXI2C     ;
                call        #I2C_TX              ; 先写片内高地址字节
                mov. b      ADDRI2C0,RXTXI2C     ;
                call        #I2C_TX              ; 再写片内低地址字节
                mov. b      DATAI2C,RXTXI2C      ;
                call        #I2C_TX              ; 要写的数据写入 24C256
                call        #I2C_Stop            ; 停止总线
                ret
                ; 其他子程序在前面有介绍,这里略
                ORG         0FFFEh
                DW          RESET
                END
```

5.2.2 将键盘输入的按键值送到显示器显示

这个例程很简单,主要说明子程序的输入和输出问题。对于键盘子程序的使用,当按键被按下时,键盘子程序应该输出键值。对于显示子程序的使用,将要显示的数据按照显示子程序的规定送入对应显示缓存即可。下面只有主程序,子程序已在 5.1 节列出。硬件电路与前面相同:P1 为键盘与显示位选,P5 为显示段驱动。

```
#include    "msp430x13x. h"
                RSEG        CSTACK               ; 系统堆栈
                DS          0
                RSEG        CODE                 ; 程序的开始
Reset           MOV         #300H,SP
MAIN            CALL        #DIP
                CALL        #KEYJUDGE            ; 如果有按键
                JNC         MAIN
                CALL        #KEY                 ; 得到键值

                MOV. B      R9,&200H             ; 送到第一位显示
                MOV. B      R9,&201H             ; 送到第一位显示
                MOV. B      R9,&202H             ; 送到第一位显示
                MOV. B      R9,&203H             ; 送到第一位显示
                MOV. B      R9,&204H             ; 送到第一位显示
                MOV. B      R9,&205H             ; 送到第一位显示
                JMP         MAIN
```

```
COMMON   INTVEC                ; 中断向量
ORG      RESET_VECTOR
DW       Reset
END
```

5.2.3　键盘、显示与低功耗应用

在 5.2.2 节已经讲述了如何在显示器中显示按键值，这里要实现同样的功能，但整个系统要低功耗工作。对于低功耗，可能读者马上就联系到液晶显示，液晶显示器当然功耗低，但不够亮，夜间无法看清。这里还是使用 LED 数码管。采用 LED 数码管作为低功耗显示器可以这样考虑：首先 CPU 应用在低功耗模式；其次，显示并非使用者每时每刻都要观察，利用这一点可以在大多数情况下关闭显示器。需要看时，操作键盘，就可打开显示器，使用者看后，一会儿就自动关闭。这样就能实现 LED 显示器系统的低功耗应用。

然而，MCU 处于低功耗模式，键盘程序就不大好用了。MCU 不会主动去扫描键盘，在主程序中查询键盘不可用；而在硬件上键盘与显示器的位选端公用 P1 口，如果使用 P1 口中断方式得到键盘键值，不可取。因为必须准备 P1 口的状态，以等待有按键按下时能够进入中断，这就影响了显示程序的运行。扫描显示时，只有 P1 口上的一条口线为低电平，而 P1 口键盘中断的准备，需要行线全为低电平。这就是矛盾！

同时没有了键盘扫描程序的运行，显示程序也没有办法运行。因为在键盘的消抖动过程中，调用显示子程序达到一箭双雕的效果：其一，延时消抖动；其二，不断地刷新显示，使得显示流畅，不闪烁。但现在不扫描键盘，显示程序不能运行，显示器也不亮了。

解决的办法是使用定时器中断，定时刷新显示器；定时查看键盘有无按键被按下。同时还可设置一个计数器记录键盘被按动后的时间。当这个时间超出某值时，关掉显示器，节省能耗，同时计数器清零。当再有按键按下时，打开显示器，计数器重新开始计时，超时时再关掉显示器。

主程序如下（硬件电路与前面相同：P1 为键盘与显示位选，P5 为显示段驱动）：

```
#include   "msp430x13x.h"
           RSEG     CSTACK              ; 堆栈初始化
           DS       0
           RSEG     CODE                ; 程序的开始

Reset      mov      #SFE(CSTACK),SP
SetupTA    mov      #TASSEL1+TACLR,&TACTL   ; SMCLK, Clear TAR
SetupC0    mov      #CCIE,&CCTL0        ; 使能 CCR0 中断
           mov      #500,&CCR0
SetupP1    bis      #MC1,&TACTL         ; 定时器工作在连续模式
           eint                         ; 开中断
           MOV      #0FFDFH,&206H
           MOV      #200H,&208H
           MOV      #5000,&20AH
           MOV.B    #22H,&202H
```

```
KEY2        POP.B    &P1OUT
            POP.B    &P1DIR
            RET

KEY_TAB     MOV.B    R9,&200H          ;送到第一位显示
            MOV.B    R9,&201H          ;送到第二位显示
            MOV.B    R9,&202H          ;送到第三位显示
            MOV.B    R9,&203H          ;送到第四位显示
            MOV.B    R9,&204H          ;送到第五位显示
            MOV.B    R9,&205H          ;送到第六位显示
            ;具体的键盘处理程序根据读者需要请自行编写
            NOP                        ;这只是一个框架
            RET
```

关于键盘子程序中用到的判键与键值识别与前面的程序相同，这里略。在主程序中用到的显示子程序也与前面的程序相同，略。

```
            END
```

5.2.4　简易电子琴的设计

在 5.1.9 小节讲了如何使用 MSP430 产生乐音输出，在此基础之上，加上前面所讲的键盘就可以很容易构成一个电子琴。而前面所讲的键盘只有 16 个按键，只能构成简单 16 音阶电子琴，由读者自行扩展键盘就可构成能产生很多音阶的电子琴。程序如下：

```
#include    "msp430x13x.h"
            RDWSEG   CSTACK            ;系统堆栈
            DS       0
            RSEG     CODE              ;Program code 的开始
Reset       mov      #SFE(CSTACK),SP   ;
SetupTA     mov      #TASSEL1+TACLR,&TACTL  ;SMCLK, Clear TAR
SetupC0     mov      #CCIE,&CCTL0      ;使能 CCR0 中断
            mov      #100,&CCR0        ;
SetupP1     bis      #MC1,&TACTL       ;定时器工作在连续模式
            MOV.B    #0FH,&P1DIR
            MOV.B    #0FH, &P1OUT
            MOV.B    #0,&P1IES
            MOV.B    #70H,P1IE
            MOV.B    #8,&P4DIR
            MOV.B    #0FH,&230H
            eint                       ;开中断
Mainloop    bis      #LPM1,SR          ;低功耗模式
TA0_isr     XOR.B    #08H,&P4OUT       ;求反输出音乐
            add      &220H,&CCR0       ;将对应频率的数据加到 CCR0
            RETI
```

设定须用键盘，可使用前面讲述的键盘电路。当实时温度超出设定的报警值时，输出报警，由 P4.3 输出通过三极管驱动扬声器或蜂鸣器。电路前面已经讲过，此处略。

2. 软件描述

要求工作在低功耗模式，所以所有的事情须在中断服务程序中完成。在主程序中要完成初始化，主要是定时器、端口及需要用到的 RAM 单元。程序如下：

```
#include    "msp430x13x.h"
            RSEG    CSTACK                  ; 堆栈初始化
            RSEG    CODE                    ; 程序的开始
Reset       mov     #SFE(CSTACK),SP
SetupTA     mov     #TASSEL1+TACLR,&TACTL   ; 定时器初始化
SetupC0     mov     #CCIE,&CCTL0            ; 使能 CCR0 中断
            mov     0#500,&CCR0
SetupP1     bis     #MC1,&TACTL             ; 定时器工作在连续模式
            eint                            ; 开中断
            MOV     #0FFDFH,&206H           ; 显示时扫描位初始值
            MOV     #200H,&208H             ; 显示缓存起址 200H
            BIS.B   #08H,&P4DIR             ; P43 将报警，为输出方向
            MOV.B   #22H,&202H              ; 第四位显示器不显示
            MOV     #220H,&210H             ; ADC 缓存的起始位置
Mainloop    bis     #LPM1,SR                ; 低功耗模式
```

所有的处理都在定时器中断中进行。首先要定时采样温度值（调用 ADC10 子程序）。然后判断测量缓冲器是否满。在测量子程序中，每次的测量数据保存在由 220H 开始，结束于 240H 的 16 字队列中，当队列满时，对所有数据处理。数据的处理为：去掉最大的和最小的数据，剩下的求平均。而无论对数据处理与否，每次定时器中断都得调用键盘，如果有键按下，则设置报警值，否则往下运行。最后刷新显示，并调用报警子程序，看实时温度值是否超出设定警戒温度值。此程序大量使用子程序，所以较为简洁。程序如下：

```
TA0_isr     add     #500,&CCR0          ; 增加 CCR0 以偏移量
            call    #adc10              ; 温度采样
            CMP     #240h,&210H         ; 当测量缓存满时，处理测量值
            JNZ     TA0_0
            CALL    #ADCHULI
TA0_0       CALL    #KEY                ; 否则不予处理
            CALL    #DISPLAY_LED        ; 显示
            CALL    #ALARM              ; 判断是否报警
            reti
```

下面是报警程序。报警值为两位十进制数，所以只需要比较实时温度值的十位与个位，对于小数点不予考虑。在程序中，先比较十位，如果十位不同，则退出；如果十位相同，则继续比较个位，如果个位不同，则退出；否则输出报警（设置 P4.3=1）。

注意：报警的输出不但要 P4.3 驱动蜂鸣器，而且显示器的最低位小数点亮，在实际使用中，可以单独引出，以便引起现场人员的注意。同时实时温度值的显示中，个位有小数点，在显

示的处理上，有小数点与没有小数点在显示缓存中的数据相差10H(可查看显示码表)。所以在比较个位时，要分别将两数据的高半字节屏蔽掉。

```
ALARM       PUSH      R5                ；报警子程序
            PUSH      R6
            MOV.B     &203H,R5
            MOV.B     &201H,R6
            CLRC
            SUB.B     R6,R5             ；比较十位
            JNC       ALARMEND

            MOV.B     &204H,R5
            AND.B     #0FH,R5
            MOV.B     &200H,R6
            AND.B     #0FH,R6
            CLRC
            SUB.B     R6,R5             ；比较个位时屏蔽掉高半字节
            JNC       ALARMEND
            BIS.B     #08H,&P1OUT       ；超出警戒温度，声音报警
            AND.B     #0FH,&200H
            ADD.B     #10H,&200H        ；显示器最后一位的小数点被点亮
            POP       R6
            POP       R5
            RET

ALARMEND    BIC.B     #08H,&P1OUT       ；没有超出警戒温度，不报警
            AND.B     #0FH,&200H
            POP       R6
            POP       R5
            RET
```

键盘子程序与前面的有差别。主要表现在将键值的处理具体化了，这里键盘被用于温度报警值的设置。设置操作直接放在按键被释放之后。使用了一个标志单元，用于判断刚才的键值是报警温度值的十位还是个位。如果是个位，则放在最后一个显示缓存，否则放在前一个显示缓存单元。

```
KEY         PUSH.B    &P1DIR
            PUSH.B    &P1OUT
            CALL      #KEYJUDGE         ；没有按键就退出，有就得到键值并处理
            JNC       KEY2              ；
            MOV       #2000,R15
KEY0        DEC       R15
            JNZ       KEY0
            CALL      #KEYJUDGE
```

```
        INC     KEY2            ; 去抖动
        CALL    #KEYCODE        ; 得到键值
        PUSH    R9
KEY1    CALL    #KEYJUDGE       ; 等待按键松开
        JC      KEY1
        POP     R9
        Tst.b   &211h           ; 214H 为一个二进制计数器，当为 0 时键值被
                                ; 送到 201H，否则送到 200H
        jz      key11
        mov.b   r9,&200h
        mov.b   #0,&211h
        jmp     key2
key11   mov.b   r9,&201h
        mov.b   #1,&211h
KEY2    POP.B   &P1OUT
        POP.B   &P1DIR
        RET
```

;键盘需要的其他子程序(判键与键码识别)与前面的相同，这里略

下面是温度采样程序。采样得到的数据保存在 220H～240H 缓存中。在这个队列中，当前采样值究竟放在哪个单元，设置一指针(210H)以指示由 210H 寻址保存单元。由于模数转换为 12 位，所以要占用 2 字节，指针的改变应该增 2。

```
adclr   bic     #ENC,&ADC12CTL0             ; ENC =0
        mov     #REFON+ADC12ON,&ADC12CTL0   ; VER=1.5 V(片内)
        mov     #SHP+ADC12SSEL_2,&ADC12CTL1 ; 上升沿采样，主时钟
        bis     #1000h,&ADC12CTL1           ; 选择 MEM1  0001  0000  0000  0000
        mov.b   #1Ah,&ADC12MCTL1            ; 选择 ADC 通道 0A  ：0001 1010
                                            ; 参考电压：VER ～ VSS
        bis     #ENC,&ADC12CTL0             ; 使 ENC=1
        bis     #ADC12SC,&ADC12CTL0         ; 开始转换
testEOC1 bit    #BIT1,&ADC12IFG             ; 是否转换完毕 (ADC12IFG.0=1?)
        jz      testEOC1                    ; 如果没有完成则等待
        MOV     &210H,R15                   ; 210H 为缓存指针
        mov     &ADC12MEM1,0(R15)           ; 转存到 220H ～ 240H
        INC     &210H
        INC     &210H                       ; 改变指针
        MOV.B#0FFH,&P5DIR
        RET
```

当记录单元满时，须及时处理，主要是计算，将测量数据转换为温度值。首先求出 16 个测量数据的平均值。求平均值为所有数据的和除以 16，它不需要使用除法程序，因为右移一次相当于除以 2，所以右移 4 次，就等于除以 16。然后将这个平均值转换为温度值。通过实验得出测量数据与温度值的关系为

温度值 = 测量值 / 16 − 159

或 温度值 = 测量值 / 32。

在数据处理上，分整数部分与小数部分。整数由公式计算。十六进制与压缩 BCD 码的转换使用查表的方法。这里数据范围不大，查表法要方便些。对于小数部分，将平均值的最低 5 位除以 32 即得小数部分，要用于显示，再乘以 10。

```
ADCHULI  MOV     #220H,&0210H      ；初始化记录指针
         DEC.B   &212h
         CMP.B   #30,&212h
         JNZ     adc104
         MOV.B   #0,&212h
         CLRC
         MOV     #0,R5
         MOV     #0220H,R15        ；从 220H 开始加
ADC102   ADD     @R15,R5
         INC     R15
         INC     R15
         CMP     #240H,R15         ；加到 240H 单元
         JNZ     ADC102            ；求和
         RRA     R5
         RRA     R5
         RRA     R5
         RRA     R5
         BIC     #0F000H,R5        ；除以 16 得平均值，
         MOV     R5,R6             ；后面用 R6 处理小数部分
         RRA     R5
         RRA     R5
         RRA     R5                ；当前操作与除以 2 效果一样
         RRA     R5
         BIC     #0F000H,R5        ；除以 16，再减 159 得温度值的整数部分
         SUB     #159,R5
         MOV.B   TABWENDU(R5),R5   ；从表中查得十进制数

         MOV.B   R5,&204H          ；显示格式：两位整数，一位小数，从最左边开始
         BIC.B   #0F0H,&204H
         ADD.B   #10H,&204H        ；个位
         MOV.B   R5,&205H
         RRA.B   &205H
         RRA.B   &205H
         RRA.B   &205H
         RRA.B   &205H
         BIC.B   #0F0H,&205H       ；十位
                                   ；203H 小数部分的计算，平均值后 5 位除以 3.2
```

心都是语音的录入与输出(录音与放音)。本示例演示了在 MSP430 系统中如何录音和放音。

在单片机系统中使用较多的语音设备是 ISD 系列的语音芯片。其中 ISD4004 更有 16 min的语音记录时间。本例的固体录音机将选用此芯片为核心设计。

1. ISD4004 简介

ISD4004 系列芯片的工作电压为 3 V,单片录放时间为 8~16 min,音质好,适用于移动电话及其他便携式电子产品中。芯片采用 CMOS 技术,内含振荡器、防混淆滤波器、平滑滤波器、音频放大器、自动静噪及高密度多电平闪烁存储阵列。芯片设计是基于所有操作必须由微控制器控制,操作命令可通过串行通信接口(SPI 或 Microwire)送入。芯片采用多电平直接模拟量存储技术,每个采样值直接存储在片内闪烁存储器中,因此能够非常真实、自然地再现语音、音乐、音调和效果声,避免了一般固体录音电路因量化和压缩造成的量化噪声和"金属声"。采样频率可为 4.0、5.3、6.4 和 8.0,单位 kHz,频率越低,录放时间越长,而音质则有所下降。片内信息存于闪烁存储器中,可在断电情况下长期保存(典型值),反复录音 10 万次。

2. 引脚描述

● 电源(VCCA,VCCD) 为使噪声最小,芯片的模拟和数字电路使用不同的电源总线,并且分别引到外封装的不同管脚上,模拟和数字电源端最好分别走线,并尽可能在靠近供电端处相连,而去耦电容应尽量靠近器件。

● 地线(VSSA,VSSD) 芯片内部的模拟和数字电路也使用不同的地线。

● 同相模拟输入(ANA IN+) 这是录音信号的同相输入端。输入放大器可用单端或差分驱动。单端输入时,信号由耦合电容输入,最大幅度为峰峰值 32 mV,耦合电容和本端的 3 kΩ电阻输入阻抗决定了芯片频带的低端截止频率。差分驱动时,信号最大幅度为峰峰值 16 mV。

● 反相模拟输入(ANA IN-) 差分驱动时,这是录音信号的反相输入端。信号通过耦合电容输入,最大幅度为峰峰值 16 mV。

● 音频输出(AUD OUT) 提供音频输出,可驱动 5 kΩ 的负载。

● 片选(SS) 此端为低,即向该 ISD4004 芯片发送指令,两条指令之间为高电平。

● 串行输入(MOSI) 此端为串行输入端,主控制器应在串行时钟上升沿之前半个周期将数据放到本端,供 ISD 输入。

● 串行输出(MISO) ISD 的串行输出端。ISD 未选中时,本端呈高阻态。

● 串行时钟(SCLK) ISD 的时钟输入端,由主控制器产生,用于同步 MOSI 和 MISO 的数据传输。数据在 SCLK 上升沿锁存到 ISD,在下降沿移出 ISD。

● 中断(/INT) 本端为漏极开路输出。ISD 在任何操作(包括快进)中检测到 EOM 或 OVF 时,本端变低并保持。中断状态在下一个 SPI 周期开始时清除。中断状态也可用 RINT 指令读取。OVF 标志——指示 ISD 的录、放操作已到达存储器的末尾。EOM 标志——只在放音中检测到内部的 EOM 标志时,此状态位才置 1。

● 行地址时钟(RAC) 漏极开路输出。每个 RAC 周期表示 ISD 存储器的操作进行了一行(ISD4003 系列中的存储器共 1 200 行,ISD4004 系列中的存储器共 2 400 行)。该信号保持高电平为 175 ms,低电平为 25 ms,如图 5.41 所示。快进模式下,RAC 的 218.75 μs 是高电平,31.25 μs 为低电平。该端可用于存储管理技术。

● 外部时钟(XCLK) 本端内部有下拉元件。芯片内部的采样时钟在出厂前已调校,误

差低于 ±1%。商业级芯片在整个温度和电压范围内，频率变化低于±2.25%。工业级芯片在整个温度和电压范围内，频率变化在 −6% ～ +4% 内，此时建议使用稳压电源。若要求更高精度，可从本端输入外部时钟。由于内部的防混淆及平滑滤波器已设定，故上述推荐的时钟频率不应改变。输入时钟的占空比无关紧要，因内部已进行了分频。在不外接地时钟时，此端必须接地。

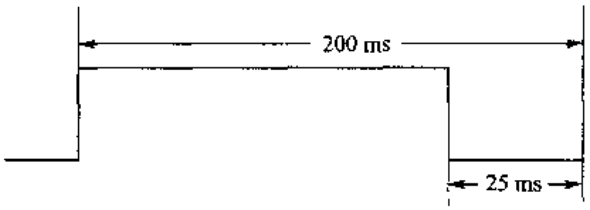


图 5.41　电平保持时间

● 自动静噪(AMCAP)　当录音信号电平下降到内部设定的某一阈值以下时，自动静噪功能使信号衰弱，这样有助于消除无信号(静音)时的噪声。通常本端对地接 1 μF 的电容，构成内部信号电平峰值检测电路的一部分。检出的峰值电平与内部设定的阈值作比较，决定自动静噪功能的翻转点。大信号时，自动静噪电路不衰减，静音时衰减 6 dB。1 μF 的电容也影响自动静噪电路对信号幅度的响应速度。本端接 VCCA 则禁止自动静噪。

3. SPI(串行外设接口)

ISD4004 工作于 SPI 串行接口。SPI 协议是一个同步串行数据传输协议，协议假定微控制器的 SPI 移位寄存器在 SCLK 的下降沿动作，因此对 ISD4004 而言，在时钟上升沿锁存 MOSI 引脚的数据，在下降沿将数据送至 MISO 引脚。协议的具体内容如下：

● 所有串行数据传输开始于 SS 下降沿。

● SS 在传输期间必须保持为低电平，在两条指令之间则保持为高电平。

● 数据在时钟上升沿移入，在下降沿移出。

● SS 变低，输入指令和地址后，ISD 才能开始录放操作。

● 指令格式是 5 位控制码加 11 位地址码。

● ISD 的任何操作(含快进)如果遇到 EOM 或 OVF，则产生一个中断，该中断状态在下一个 SPI 周期开始时被清除。

● 使用读指令使中断状态位移出 ISD 的 MISO 引脚时，控制及地址数据也应同步从 MOSI 端移入。因此要注意移入的数据是否与器件当前进行的操作兼容。当然，也允许在一个 SPI 周期里，同时执行读状态和开始新的操作(即新移入的数据与器件当前的操作可以不兼容)。

● 所有操作在运行位(RUN)置 1 时开始，置 0 时结束。

● 所有指令都在 SS 端上升沿开始执行。

4. ISD4004 指令表

表 5.5 列出了 ISD4004 的指令。

表 5.5　ISD4004 指令表

指　令	5 位控制码(11 位地址)	操作摘要
POWERUP	00100(XXXXXXXXXXX)	上电，等待 TPUD 后器件可以工作
SET PLAY	11100(A10～A0)	从指定地址开始放音，必须后跟 PLAY 指令使放音继续
PLAY	11110(XXXXXXXXXXX)	从当前地址开始放音(直至 EOM 或 OVF)

续表 5.5

指　令	5 位控制码(11 位地址)	操作摘要
SET REC	10100(A10 ~ A0)	从指定地址开始录音,必须后跟 REC 指令使录音继续
REC	10110(XXXXXXXXXXX)	从当前地址开始录音(直至 OVF 或停止)
SET MC	11101(A10 ~ A0)	从指定地址开始快进,必须后跟 MC 指令使快进继续
MC	11111(XXXXXXXXXXX)	执行快进,直到 EOM;若再无信息,则进入 OVF 状态
STOP	0X110(XXXXXXXXXXX)	停止当前操作
STOP WRDN	0X01X(XXXXXXXXXXX)	停止当前操作并掉电
RINT	0X110(XXXXXXXXXXX)	读状态:OVF 和 EOM

5. ISD4004 操作时序

ISD4004 的操作时序如图 5.42 所示。

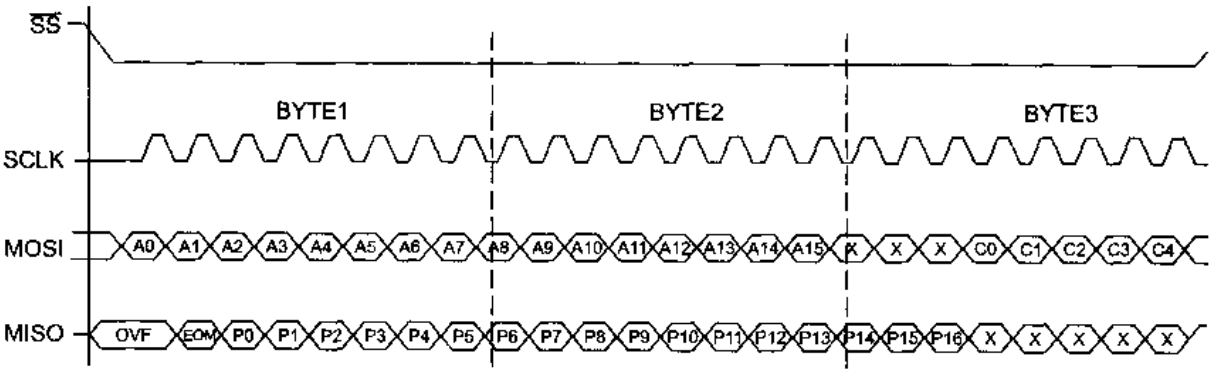


图 5.42　ISD4004 操作时序

6. 固体数码录音机的硬件电路

话筒放大器对语音输入信号放大,以达到 ISD4004 输入信号要求。使用功率放大器驱动扬声器发出语音。由于 ISD4004 与 MCU 的接口为 SPI 串行方式,只需要少量 I/O 口线,这里使用 MSP430F1121 为控制芯片。设置两个按键:录音键和放音键。每个按键第一次按下为启动操作,再次按下为结束操作,即停止当前操作。其电路如图 5.43 所示。其中的所有器件均为 3 V 工作,所以使用电池供电。

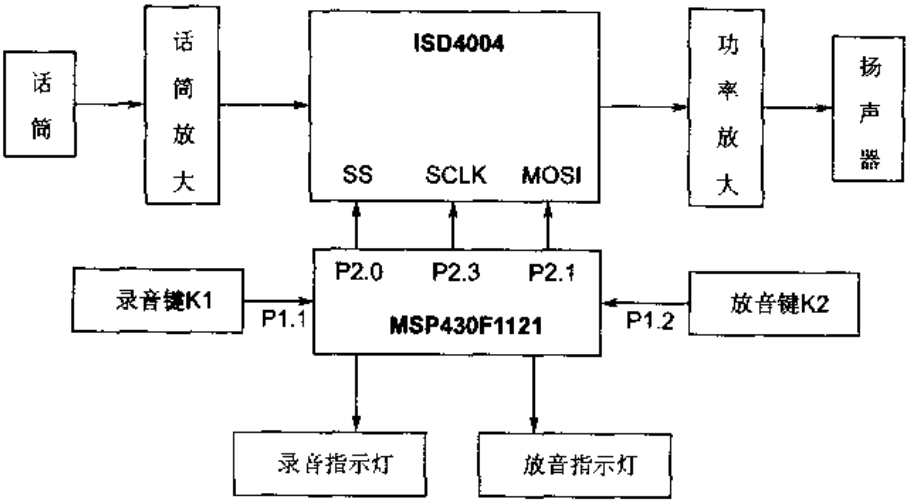


图 5.43　固体录音机电路框图

7. 固体数码录音机的软件设计

先看操作界面，对整个录音机的操作只有两个按键，K1 和 K2 对应于录音和放音。系统的运行听从两按键的指挥，因此主程序很简单，查询键盘。如果 K1 按下，则录音，如果 K1 再次按下，则停止录音；如果 K2 按下，则放音，如果 K2 再次按下，则停止放音。程序如下：

```
;约定:200H 字单元,保存录音与放音的起始地址
;       203H 字节单元,为录音键的计数器,用于判断录音键按下的奇数/偶数,用于决定是录音操作
;       还是停止录音操作,同时对应指示灯指示
;       204H 字节单元,为放音键的计数器,用于判断放音键按下的奇数/偶数,用于决定是放音操作
;       还是停止放音操作,同时对应指示灯指示

#include    "msp430x11x1.h"
            ORG     0F000H
Reset       mov     #  300H ,SP        ;堆栈指针放在最顶端,F133 为 300H
            MOV.B   #0FFH,&P2DIR
            MOV.B   #018H,&P1DIR
            MOV.B   #0FFH,&P2OUT
            MOV.B   #1,&203H            ;初始化为奇数
            MOV.B   #1,&204H            ;初始化为奇数
            BIS.B   #018H,&P1OUT        ;录音指示灯与放音指示灯都熄
            CALL    #DSTOP              ;
            MOV     #0FFFH,R15
MAINLOOP0   DEC     R15                 ;初始化 ISD4004 芯片
            JNZ     MAINLOOP0
            CALL    #POWERUP            ;上电
            MOV     #0FFFFH,R15
MAINLOOP1   DEC     R15
            JNZ     MAINLOOP1
MAIN        BIT.B   #02H,&P1IN          ;判断按键,是否录音键
            JNZ     MAIN1
            BIT.B   #4H,&P1IN           ;判断按键,是否放音键
            JNZ     MAIN2
            JMP     MAIN
MAIN1       CALL    #DELAY5
            BIT.B   #02H,&P1IN          ;延时消除抖动
            JZ      MAIN
MAIN1_0     BIT.B   #2,&P1IN            ;判断按键,是否录音键
            JNZ     MAIN1_0
            CMP.B   #1,&203H            ;是录音键偶数次被按下
            JNZ     MAIN11              ;
            MOV.B   #2 , &203H
            MOV     #1200,&200H         ;从某地址处录音
            BIC.B   #8H,&P1OUT          ;录音指示灯亮
```

```
                CALL    #REC_IN          ; 是录音键,录音
                JMP     MAIN
MAIN11          CALL    #STOPP
                BIS.B   #8H,&P1OUT       ; 录音指示灯熄
                MOV.B   #1,&203H         ; 203H 为奇数,下次按键则被认为录音
                JMP     MAIN
MAIN2           CALL    #DELAY5
                BIT.B   #4H,&P1IN        ; 延时消除抖动
                JZ      MAIN
MAIN2_0         BIT.B   #4,&P1IN         ; 判断按键,是否放音键
                JNZ     MAIN2_0
                CMP.B   #1,&204H         ; 如果奇数次按键,则放音
                JNZ     MAIN22
                MOV.B   #2, &204H
                CALL    #STOPP
                MOV     #1200,&200H      ; 从某地址处放音
                CALL    #PLAY_OUT
                JMP     MAIN
MAIN22          CMP.B   #2,&204H         ; 如果偶数次按键,则停止放音
                JNZ     MAIN23
                MOV.B   #1,&204H
                CALL    #STOPP
                JMP     MAIN
```

下面是录音子程序。按照 ISD4004 的时序串行输入录音指令。

```
;参数:200H 字单元为录音后存放在 ISD4004 内的具体起始地址
;      202H 字节单元为 ISD4004 指令
REC_IN          BIC.B   #1,&P2OUT        ; SS=0
                CALL    #SEND_16
                MOV.B   #10100000B,&202H ; SET REC
                CALL    #SEND_8
                BIS.B   #1,&P2OUT        ; SS=1
                MOV     #03FFH,R15
REC_INLOOP      DEC     R15
                JNZ     REC_INLOOP
                BIC.B   #1,&P2OUT        ; SS=0
                MOV.B   #10110000B,&202H
                CALL    #SEND_8
                BIS.B   #1,&P2OUT        ; SS=1
                RET
```

下面是放音子程序。按照 ISD4004 的时序串行输入放音指令。

```
;参数:200H 字单元为 ISD4004 内的具体起始放音地址
```

```
;       202H 字节单元为 ISD4004 指令
PLAY_OUT  BIC. B    #1,&P2OUT          ; SS=0
          CALL      #SEND_16
          MOV. B    #11100000B,&202H   ; SET PLAY
          CALL      #SEND_8
          BIS. B    #1,&P2OUT          ; S=1
          NOP
          NOP
          BIC. B    #1,&P2OUT          ;SS=0
          MOV. B    #11110000B,&202H
          CALL      #SEND_8
          BIS. B    #1,&P2OUT          ; SS=1
          RET
```

MSP430 与 ISD4004 的串行接口程序，串行送 16 位地址程序如下：

```
;参数:200H 字单元为 ISD4004 内的具体起始录音/放音地址
SEND_16       MOV       &200H,R13
              MOV       #16,R14
SENDDATALOOP  BIC. B    #8H,&P2OUT       ; SCLK=0    P2.3
              BIT       #001H,R13
              JNZ       SEND_11          ; 如果为 1,送 1,否则送 0
              BIC. B    #2,&P2OUT        ; 如果为 0,MOSI=0     P2.1
              JMP       SONGE1
SEND_11       BIS. B    #2,&P2OUT        ; 如果为 1,MOSI=1
SONGE1        BIS. B    #8H,&P2OUT       ; SLCK=1
              RRA       R13
              BIC. B    #8H,&P2OUT       ; SCLK=0
              DEC       R14
              JNZ       SENDDATALOOP
              RET
```

MSP430 与 ISD4004 的串行接口程序，串行送 8 位指令程序如下：

```
;参数: 202H 字节单元为 ISD4004 指令
SEND_8        MOV. B    &202H,R13
              MOV       #8,R14
SENDILOOP     BIC. B    #8H,&P2OUT       ; SCLK=0
              BIT       #001H,R13
              JNZ       SEND_1           ; 如果为 1,送 1,否则送 0
              BIC. B    #2,&P2OUT        ; 如果为 0,MOSI=0
              JMP       SONGE
SEND_1        BIS. B    #2,&P2OUT        ; 如果为 1,MOSI=1
SONGE         BIS. B    #8H,&P2OUT       ; SLCK=1
              RRA       R13
```

```
NOP
NOP
BIC.B    #8H,&P2OUT      ;SCLK=0
DEC      R14
JNZ      SENDILOOP
RET
COMMON   INTVEC          ;中断向量
ORG      RESET_VECTOR
DW       Reset
END
```

5.3 系统应用设计

本节将以相对更为复杂的实例进一步说明如何进行 MSP430 的系统设计。

5.3.1 时间控制器的设计

在实际生活中，时间控制器的使用是非常广泛的。大家很熟悉的打铃系统及酿造厂的发酵罐等。下面以打铃系统为例，讲述时间控制系统的设计问题(程序以 C 语言为主)。

1. 系统设计

整个系统需要一个系统时钟，作为控制器的时间标准。控制输出的时间设定、系统时钟的校准及控制输出的准许与否等，都需要输入设备，这里以前面讲的 12 按键键盘为输入设备，输出使用三极管驱动蜂鸣器。在所有的操作过程中，使用前面讲的 6 位数码管作为显示、显示操作标志以及输入的数据(硬件电路略，可参照前面相应部分)。

(1) 键盘约定

- 0 号键为时间设置键与修改功能键。
- 1 号键为打铃时间设定功能键。
- 11 号键为打铃输出与否修改功能键。

(2) 显示界面约定

① 走时　系统在没有操作时为走时状态。显示界面样式为：13.0723.第二位的小数点为秒信号闪烁。最后一位的小数点表示系统的控制是否输出，小数点亮表示每当到设定时间，则输出打铃；小数点熄表示即使到设定时间，也不打铃。为了符合人们习惯，如果小时的十位为 0，则不予显示。

② 整系统时间　功能键为 0 号按键。显示界面样式为：--0723。最左边两位显示中间杠，后面四位紧接着显示输入的设定时间，只有小时和分钟，没有秒。0723 表示输入为现在时间上午 7 点 23 分。

③ 控制输出时间的设定　功能键为 1 号按键。这个程序指示一个示例，一共设有 6 个打铃输出时间。当按下设定键后，显示界面显示样式为：1-0745。最左边的"1-"表示第一个时间设定点；0745 为原先设定的打铃时间。如果需要修改，则按 0 号键，之后显示界面变为 1-　(后面 4 位不显示)，表示第一个打铃设定时间点将被修改，接着由键盘输入数据；如果不

需要修改，则按任何非 0 号按键。当设置完毕，按任何键则进入下一个打铃时间设置点，界面样式为：1－0745。其余操作与上述相同。当 6 个打铃点设置完毕，返回走时状态。

(3) 全局变量定义

首先走时系统需使用 5 个变量(都是字节变量)：

s01　0.1 秒计数单元；

y3　秒计数单元；

y2　分计数单元；

y1　小时计数单元；

dot　时钟界面中秒闪烁信号指示单元。

对于打铃时间，使用了一个结构数组(下面已经有了初始值)：

```
struct
  {uchar hour,mte;}time[6]={{6,0},{6,5},{6,10},{7,40},{13,40},{17,20}};
```

还需要 6 个显示缓存单元：

```
x[6]={1,2,0,0,0,0}
```

以及打铃输出控制单元(初始值为 1，表示要输出打铃)：

```
bellcontr=1
```

系统主程序如下：

```
#include <msp430x13x.h>
#define uchar unsigned char
#define unit unsigned int
uchar dot=0, bellcontr=1;
uchar x[6]={1,2,0,0,0,0},y1=20,y2=00,y3=00,s01; /* 显缓,计时单元 */
struct                                          // 6 个打铃时间
  {uchar hour,mte;}time[6]={{6,0},{6,5},{6,10},{7,40},{13,40},{17,20}};
/* time[5]为 6 个时间控制点 */
const uchar seg[]={0x3f,0x06,0x5b,0x4f,         // 显示码表
    0x66,0x6d,0x7d,0x07,
    0x7f,0x6f,0x77,0x7c,
    0x39,0x5e,0x79,0x71,
    0xbf,0x86,0xdb,0xcf,
    0xe6,0xed,0xfd,0x87,
    0xff,0xef,0xf7,0xfc,
    0xb9,0xde,0xf9,0xf1,
    0x80,0x40,0x00,0x73,0xc0};
void main(void)                                 // 系统主程序
{
    WDTCTL = WDTPW + WDTHOLD;                   // 停止看门狗
           TACTL = TASSEL1 + TACLR;             // 设置定时器 A
           CCTL0 = CCIE;                        // 使能 CCR0 中断
           CCR0 = 50000;                        // CCR0 初始值
           P4DIR |= 0x08;                       // 端口 P4.3 用于驱动电铃
```

```
        TACTL |= MC1;                          // 开始定时器
        _EINT();                               // 开主中断

while(1)                                       // 主循环
      {
      uchar j;
           x[0]=y1/10;                         // 拆分时间值到对应的显示缓存
           if(x[0]==0)                         // 如果时间的小时值的十位为0
           x[0]=0x22;                          // 则不予显示(符合习惯)
           if(dot==1)                          // 与电子表对应的秒闪烁
           x[1]=y1%10+0x10;
           else
           x[1]=y1%10;                         // 拆分时间值到对应的显示缓存
           x[2]=y2/10;
           x[3]=y2%10;
           x[4]=y3/10;
           if(bellcontr==1)                    // 输出控制位在最后单元的小数点显示
           x[5]=y3%10+0x10;                    // 如果控制信号输出,则最后小数点亮
           else
           x[5]=y3%10;

           dsp();                              // 调用显示
          if(kbjust()!=0)                      // 有否按下的按键
              {                                // 如果有,则:
                switch(kbscan())               // 判断什么键值
                   {
                    case 0:                    // 0:调时间
                       {
                        key0();
                        break;
                         }
                     case 1:                   // 1:设定时间控制点
                        {key1();
                        break;
                        }
                    case 11:                   // 11:设定到时间控制点后,是否输出
                        {key3();
                         break;
                         }
                   }
             if(bellcontr==1)                  // 如果输出畅通
              {
```

```
        }
    }
```

3. 走时时钟的调整

在有时钟的系统中,时钟的调整是必不可少的。根据最初的约定,按照操作过程与相应的显示格式编写程序。先初始化显示界面,再显示按键号码,最后将输入数据计算为时间值。具体说明请参考程序中的解释。程序如下:

```
void key0(void)                    // 设定实时时间
    {
    x[0]=0x21;
    x[1]=0x21;                     // 最前面两显示器显示中间杠--,提示为时间设置
    x[2]=0x22;
    x[3]=0x22;
    x[4]=0x22;
    x[5]=0x22;                     // 接着后面 4 位不显示,等待键盘的输入
    x[2]=kbscan();
    x[3]=kbscan();
    x[4]=kbscan();
    x[5]=kbscan();                 // 键盘输入值送入后面 4 位显示器
    y1=x[2];
    y1=y1*10;
    y1=y1+x[3];
    y1=x[2]*10+x[3];               // 前两个输入数据作为时间的小时值被保存
    y2=x[4]*10+x[5];               // 最后两个输入作为时间的分钟值被保存
    y3=0;                          // 秒被清 0
    }
```

4. 打铃控制时间的设定

在本系统中可以设定 6 个打铃时间,(当然可以设置很多)每当走时时间与设置的打铃控制时间相同时,同时允许打铃,则输出到电铃。本子程序完成 6 个打铃时间的输入与修改。程序按照前面的约定编写。请参考程序中的注释。程序如下:

```
void key1(void)                            // 设定 6 个控制时间点
 {
    uchar tmp;
    uchar i=0;
    for(i=0;i<6;i++)                       // 循环 6 次
     {
    x[0]=i+1;                              // 第一位显示控制时间点的序号
    x[1]=timjust[i];                       // 实现设定序号的自动增加
      x[2]=(time[i].hour)/10;              // 在显示器的后面 4 位显示原先的控制时间点与序号
      x[3]=(time[i].hour)%10;              // 先调出原来的设定值
      x[4]=(time[i].mte)/10;
      x[5]=(time[i].mte)%10;
```

```
        tmp=kbscan();                          //按键为何值
        if(tmp==0)                             //如果按键为 0 则修改,否则不修改,保持不变
            {
            x[2] =0x22;                        //说明需要改变相应的控制时间点
            x[3] =0x22;                        //先将显示清除,序号不变
            x[4]=0x22;
            x[5] =0x22;
            x[2]=kbscan();                     //输入 4 位数为改变后的控制时间点
            x[3]=kbscan();
            x[4]=kbscan();
            x[5]=kbscan();
            time[i].hour=x[2]*10+x[3];         //记录到相应的缓存
            time[i].mte=x[4]*10+x[5];
            kbscan();                          //任意按键表示设定完成
            }
          else
            if(tmp==0x0b)
            x[1]=0x21;
            else x[1]=0x22;
            timjust[i]=x[1];
            kbscan();
        }
    }
```

5. 打铃输出控制

通过 11 号按键设定打铃输出还是不输出,这个程序最简单。只是开关量的改变,控制变量 bellcontr 被初始化为 1,如最低位求反,则为 0,再求反,又为 1。程序如下:

```
void key3(void)                                //控制输出与否位求反
    {
    bellcontr=1;
    }
```

键盘和显示程序等都与前面的类似,不赘述。

5.3.2　用 MSP430 设计的复杂多相位交通灯

交通灯,相信读者并不陌生,这里讲解的是复杂的多相位交通灯的设计。其实现功能为:直行、左转及右转 3 个方向的交通灯控制;交通灯的时间显示;绿灯将结束时的闪烁控制;当有违章车辆时输出信号通知电子警察,拍下其车牌号等待违章处理。可以看出其功能是较为复杂的。

1. 硬件电路

在硬件上,为了简洁,作成模块化。十字路口的交通灯,在前后左右 4 个方向上每个方向上有一个模块,显示本方向上对车辆通行的允许状态。每个模块上有两个数码管显示当前允

许或禁止车辆通行的时间(单位 s);有 9 盏交通灯,分别控制本方向的 3 个细分方向,如本方向为朝东,则 3 个细分方向为向东方向(直行)、向东北方向(右转弯方向)及向东南方向(左转弯方向),每个细分方向有 3 盏灯:红、绿、黄,所以每个模块有 9 盏灯。4 个方向上的 4 个模块是完全相同的。

MSP430 有输出口线相当丰富的系列器件,如 MSP430F13/14 系列有 6×8=48 条 I/O 口线;而 MSP430F43/44 系列除了有 6×8=48 条 I/O 口线外,还有 160 段液晶模块的所有输出端在不驱动液晶时,也可用于普通输出。这样完全有条件直接使用 MSP430 的口线与所有的显示器连接(当然需要驱动)。但这样,不便于模块化,因为每个模块将有很多连线与处理器 MSP430 连接,此方案不可行。应改为使用 74HC164 串行输出与所有显示器连接,则电路相当简洁,所有模块只需 3 条 I/O 口线与处理器 MSP430 连接即可。整个交通灯如图 5.44 所示。每个交通灯模块的电路如图 5.45 所示。

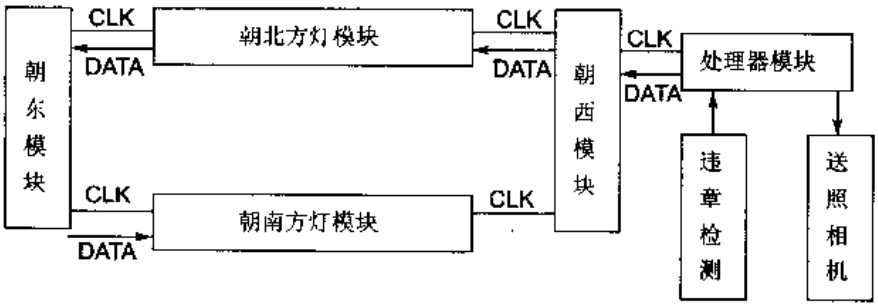


图 5.44 交通灯硬件结构框图

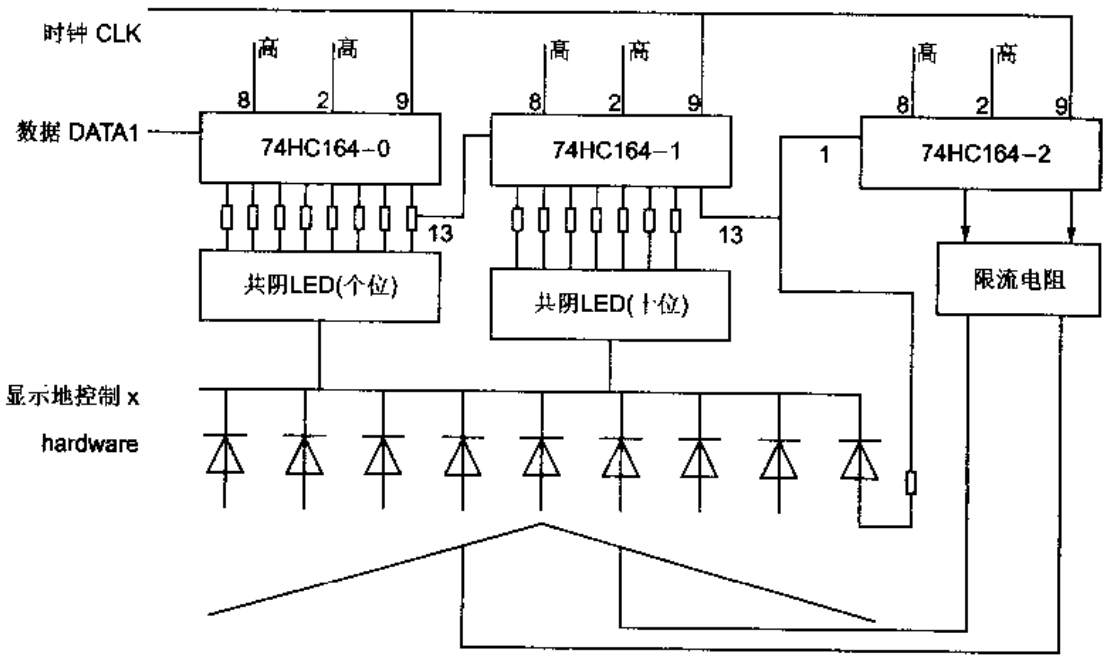


图 5.45 交通灯模块电路图

对于每个交通灯模块,电路见图 5.45。所有 74HC164 的时钟端连在一起,使用 MSP430 的一条 I/O 口线驱动。用两片 74HC164 驱动两位数码管,显示时间。而交通灯有 9 盏,剩下

如下：

```
INIT_P1   MOV.B   #02,&P1DIR        ; P1.0 输出,P1.2 输入
          MOV.B   #02,&P1OUT        ; 平常不闪光,为高电平
          BIC.B   #04H,&P1IES       ; P1.2 上升沿中断
          BIS.B   #04H,&P1IE        ; P1.2 允许中断
          RET
```

程序中将使用大 RAM 单元，在程序开始时需要为一定的具体数据。RAM 初始化如下：

```
INIT_RAM  MOV.B   #0,&220H          ; 秒计数单元
          MOV.B   #0,&221H          ; 0.1 秒计数单元
          MOV.B   #0,&227H
          MOV.B   #0,&228H          ; 0.3 s
          MOV.B   #0,&226H          ; 交通灯工作(运行)状态标志单元
          MOV.B   #20H,&200H        ; 第一模块 8 灯显示值
          MOV.B   #47H,&222H        ;        时间计数值(秒)
          MOV.B   #0,&20CH          ;        第 9 灯显示值
          ;*************************
          MOV.B   #23H,&203H        ; 第二模块 8 灯显示值
          MOV.B   #67H,&223H        ;        时间计数值(秒)
          MOV.B   #0,&20DH          ;        第 9 灯显示值
          ;*************************
          MOV.B   #20H,&206H        ; 第三模块 8 灯显示值
          MOV.B   #47H,&224H        ;        时间计数值(秒)
          MOV.B   #0,&20EH          ;        第 9 灯显示值
          ;*************************
          MOV.B   #23H,&209H        ; 第四模块 8 灯显示值
          MOV.B   #67H,&225H        ;        时间计数值(秒)
          MOV.B   #0,&20FH          ;        第 9 灯显示值
          RET
```

4. 定时器 A 中断程序

当主程序初始化之后，处理器将进入低功耗状态。那么，除了有 P1 口检测违章车辆外，其余的功能都由定时器 A 来完成。定时器 A 应该实现一个走时时钟，作为交通灯的时间基准，交通灯在此时间基准的基础上运行。定时器的时间间隔为 0.1 s，则走时时钟以秒为单位，每 10 次中断增加 1。整个过程为 106 s(在交通灯的运行部分会详细讲述)。在每一秒钟都要完成：交通灯的运行；交通灯计数器的减 1 计数；显示缓存的刷新。而在每次进入中断(0.1 s)之后，都要完成：送显示缓存的数据到显示器；判断是否绿灯闪烁(将在后面绿灯的闪烁部分详细讲解，当绿灯接近尾声时，绿灯闪烁，提醒驾驶员减速行驶)。定时器中断服务程序如下：

```
TA0_ISR   CALL    #DIS_12
          INC.B   &221H
          CMP.B   #10,&221H         ; 0.1 s  INC  10
```

```
        JNZ     TA0END
        MOV.B   #0,&221H        ; 1 s
        CALL    #TR_CONT
        CALL    #TR_RUN
        CALL    #DATA_TO_DIS
        INC.B   &220H           ; 加 1,220H 的内容作为交通灯运行的时间参考
        CMP.B   #106,&220H      ; 整个过程为 106 s
        JNZ     TA0END
        MOV.B   #0,&220H
TA0END  CALL    #FLASH_TO
        add.w   #60000,&CCR0    ; 加偏置量到 CCR0
        reti
```

5. 显示子程序的设计

显示子程序中要将所有的显示数据送到显示器，同时，在送数之前先关显示器，数据送完后，再打开显示器。下面的程序为送 12 个数据到显示器。

```
DIS_12  PUSH    R4              ; 送 12 个数据到显示器
        PUSH    R5
        BIC.B   #04H,&P2OUT     ; 先不让灯亮，以免不该亮的灯被 74HC164 扫亮
        MOV.B   #12,R5          ; 高电平有效，用三极管反向驱动
DIS_121 MOV.B   1FFH(R5),R4
        CALL    #DIS_1
        DEC.B   R5
        JNZ     DIS_121
        BIS.B   #04H,&P2OUT     ; 移位完毕让灯亮，低电平有效
        POP     R5
        POP     R4
        RET
```

下面的程序为将一字节送到 74HC164，被前面的程序调用。

```
DIS_1   PUSH    R5
        PUSH    R4
        MOV     #8,R5           ; 显示一个数字，8 位
        MOV.B   DIS_TAB(R4),R4
LOOP:   RLC.B   R4
        JC      LOOP1           ; C=1 JMP  P2.0=1
        BIC.B   #1,&P2OUT       ; 输出 0
        JMP     LOOP2
LOOP1   BIS.B   #1,&P2OUT       ; 输出 1
LOOP2   CALL    #CLK164         ; 给时钟信号
        DEC     R5
        JNZ     LOOP
        POP     R4
```

```
        POP     R5
        RET
```

下面是按照显示器与 74HC164 的连接方式编写的显示码表：

```
DIS_TAB  DB 0EEH,082H,0DCH,0D6H,0B2H   ; 0~4
         DB 076H,07EH,0C2H,0FEH,0F6H   ; 5~9
         DB 0EFH,083H,0DDH,0D7H,0B3H   ; 0.~4.
         DB 077H,07FH,0C3H,0FFH,0F7H   ; 5.~9.(0...~11H)
         DB 0,0,0,0,0,0,0,0,0,0,0      ; (15H~1FH)
         DB 34H,24H,48H,70H,60H        ; (20H~21H)  0,1,1,0,1
         ;交通灯显示码:红绿红,红绿绿,绿红绿,红红红,红红绿
         DB 21H,0C2H,0E0H,31H          ; (25H~28H)  1,0,0,0
         ;交通灯显示码:红黄绿,黄红黄,红红黄,红黄红
         DB 20H,40H,60H,30H            ; (29H~2CH)  1,0,0,0
         ;以上 4 个灯码在闪烁时使用,为绿灯不亮的灯码
```

6. 交通灯的运行

运行中的交通灯因实际情况的不同而千差万别。如有的道路为单行道，有的方向上允许通行的时间较少，有的通行时间多，有的路口为“丁”字路等，导致实际的交通灯运行很不一样，但多半是运行参数的不同。本示例的交通灯为控制较为全面的南京中山东路和解放路十字路口交通灯的真实写照。因为交通灯前后两模块的运行显示是完全相同的(有的例外)，左右两模块的运行显示也是完全相同的。所以下面交通灯的实际运行状况只描述前面的模块与左边的模块，如图 5.46 所示。

从图中可看出整个交通灯的运行时间是 104 s 一个循环。在程序的编写上，直接按照图 5.46 将交通灯与倒计时数据等送达各个模块即可。

在图 5.46 中，定义了交通灯在各时间段的运行状态。在各个状态里，灯有固定的显示，计数器有固定的计数范围。在理解图 5.46 的基础之上，在一个循环时间内，各运行状态的时间分配如图 5.47 所示。

为了程序编写方便，下面先编写交通灯的显示码。其中前 8 位为一个字节，最后一位单独表示，放在某个特定单元。在后面要讲到的有关闪烁问题，也需要编写显示码。闪烁主要是绿灯闪烁，闪烁的原理是：每隔 0.1 s 进入一次中断，即可间隔 0.1 s 让绿灯亮，再间隔 0.1 s 让绿灯灭。这里将绿灯熄灭的相应显示码也编好，后面可直接使用。显示码表如表 5.6 所列。

表 5.6 交通灯显示码表

灯	8	7	6	5	4	3	2	1	0	前 8 位段码	编 码	第 9 灯
左直右	右绿	右黄	直红	左红	右红	左绿	直绿	左黄	直黄			
红绿红				1	1		1			34H	20H	
红绿绿	1			1			1			24H	21H	1
绿红绿	1		1			1				48H	22H	1
红红红			1	1	1					70H	23H	
红红绿	1		1	1						60H	24H	1

续表 5.6

灯 左直右	8 右 绿	7 右 黄	6 直 红	5 左 红	4 右 红	3 左 绿	2 直 绿	1 左 黄	0 直 黄	前 8 位段码	编　码	第 9 灯
红黄绿	1			1					1	21H	25H	1
黄红黄		1	1					1		0C2H	26H	
红红黄		1	1	1						0E0H	27H	
红黄红		1			1				1	31H	28H	
红熄绿	1			1						20H	29H	1
熄红熄			1							10H	2AH	
红红熄			1	1						60H	2BH	
红熄红				1	1					30H	2CH	

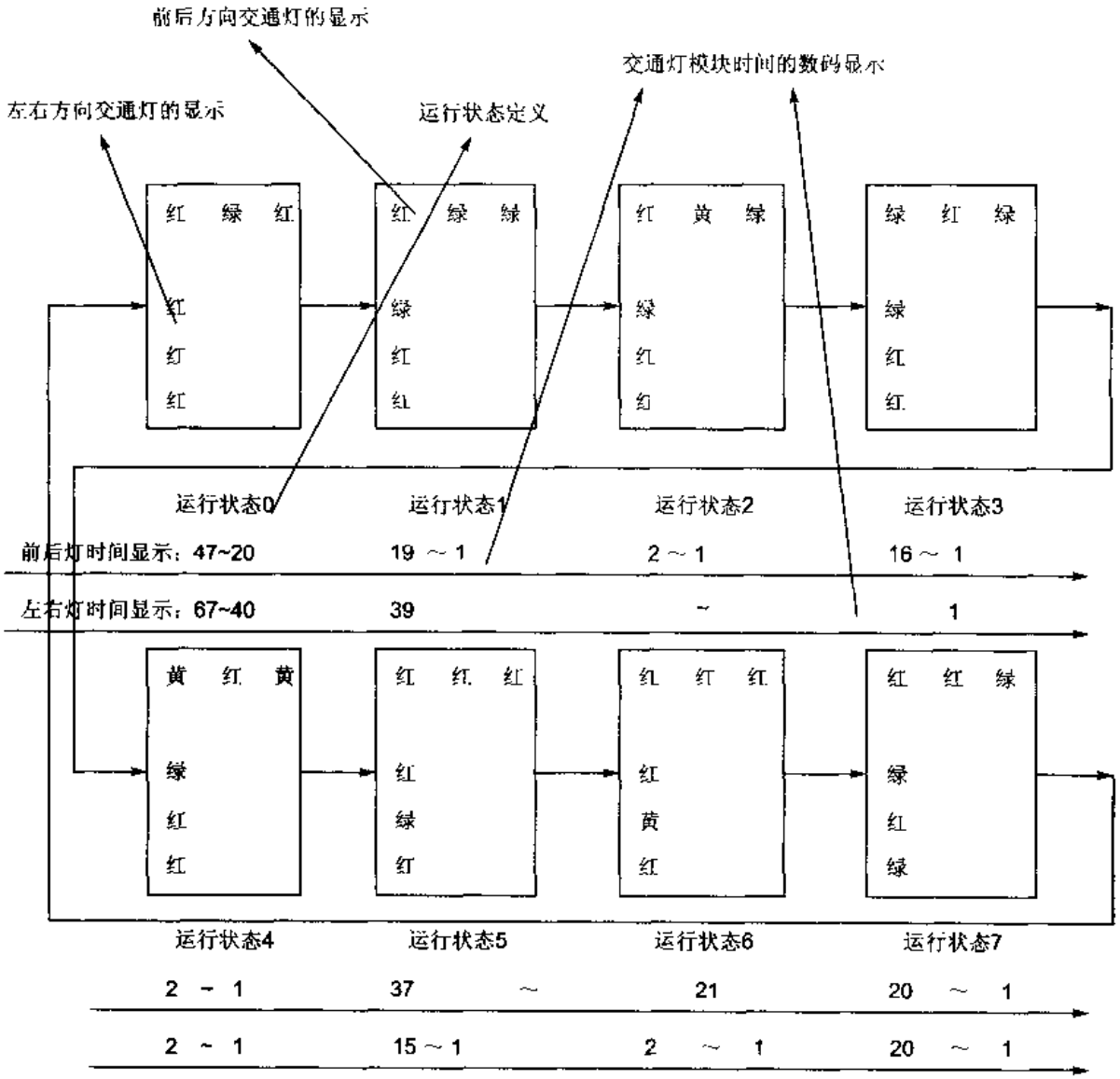


图 5.46　实际交通灯的运行

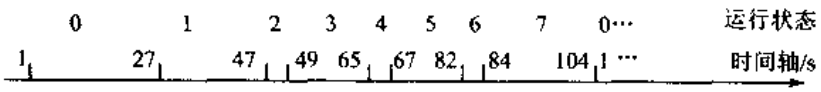


图 5.47　各运行状态的运行时间段示意图

当基准时钟运行到图 5.47 所指示的时刻时，赋予交通灯以各个状态与相应的倒计时起点。对于交通灯的状态，可以将各相应的显示编码直接写入对应的显示缓存。

RAM 单元的使用：

200H 为第一模块 8 灯显示编码，位于显存；　20CH 为第一模块第 9 灯显示值；
203H 为第二模块 8 灯显示编码，位于显存；　20DH 为第二模块第 9 灯显示值；
206H 为第三模块 8 灯显示编码，位于显存；　20EH 为第三模块第 9 灯显示值；
209H 为第四模块 8 灯显示编码，位于显存；　20FH 为第四模块第 9 灯显示值；
222H 为第一模块时间计数器；　223H 为第二模块时间计数器；
224H 为第三模块时间计数器；　225H 为第四模块时间计数器。

```
;交通灯的运行程序
TR_RUN    CMP.B   #0,&220H          ;开始，进入状态 0
          JNZ     TR_RUN1
          MOV.B   #20H,&200H        ;第一模块 8 灯显示值：红绿红
          MOV.B   #47H,&222H        ;       时间计数值
          MOV.B   #0,&20CH          ;       第 9 灯显示值
          MOV.B   #23H,&203H        ;第二模块：红红红
          MOV.B   #67H,&223H
          MOV.B   #0,&20DH
          MOV.B   #20H,&206H        ;红绿红
          MOV.B   #17H,&224H
          MOV.B   #0,&20EH
          MOV.B   #23H,&209H        ;红红红
          MOV.B   #67H,&225H
          MOV.B   #0,&20FH
          MOV.B   #0,&226H
TR_RUN1   CMP.B   #27,&220H         ;进行了 27 s 之后，进入状态 1
          JNZ     TR_RUN2
          MOV.B   #21H,&200H        ;第一模块 8 灯显示值：红绿绿
          MOV.B   #1,&20CH          ;       第 9 灯显示值
          MOV.B   #24H,&203H        ;第二模块：红绿绿
          MOV.B   #1,&20DH
          MOV.B   #21H,&206H        ;第三模块：红红绿
          MOV.B   #1,&20EH
          MOV.B   #24H,&209H        ;第四模块：红红绿
          MOV.B   #01,&20FH
          MOV.B   #1,&226H

TR_RUN2   CMP.B   #47,&220H         ;进行了 47 s 之后，进入状态 2
          JNZ     TR_RUN3
          MOV.B   #25H,&200H        ;第一模块 8 灯显示值：红黄绿
          MOV.B   #2H,&222H         ;时间计数值为黄灯显示时间
          MOV.B   #1,&20CH          ;       第 9 灯显示值
```

```
        MOV.B   #24H,&203H      ;第二模块：绿红红
        MOV.B   #1,&20DH
        MOV.B   #25H,&206H      ;第三模块:红黄绿
        MOV.B   #2H,&224H
        MOV.B   #1,&20EH
        MOV.B   #24H,&209H      ;第四模块
        MOV.B   #01,&20FH
        MOV.B   #2,&226H
TR_RUN3 CMP.B   #49,&220H       ;进行了 49 s 之后,进入状态 3
        JNZ     TR_RUN4
        MOV.B   #22H,&200H      ;第一模块 8 灯显示值:绿红绿
        MOV.B   #16H,&222H      ;时间计数值为黄灯显示时间
        MOV.B   #1,&20CH        ;          第 9 灯显示值
        MOV.B   #24H,&203H      ;第二模块：绿红红
        MOV.B   #1,&20DH
        MOV.B   #22H,&206H      ;第三模块:绿红绿
        MOV.B   #16H,&224H
        MOV.B   #1,&20EH
        MOV.B   #24H,&209H      ;第四模块
        MOV.B   #01,&20FH
        MOV.B   #3,&226H
TR_RUN4 CMP.B   #65,&220H       ;进行了 65 s 之后,进入状态 4
        JNZ     TR_RUN5
        MOV.B   #26H,&200H      ;第一模块 8 灯显示值:黄红黄
        MOV.B   #2H,&222H       ;时间计数值为黄灯显示时间
        MOV.B   #0,&20CH        ;          第 9 灯显示值
        MOV.B   #27H,&203H      ;第二模块：绿红红
        MOV.B   #1,&20EH
        MOV.B   #26H,&206H      ;第三模块
        MOV.B   #2H,&224H
        MOV.B   #0,&20EH
        MOV.B   #27H,&209H      ;第四模块:红红黄
        MOV.B   #01,&20FH
        MOV.B   #4,&226H
TR_RUN5 CMP.B   #67,&220H       ;进行了 67 s 之后,进入状态 5
        JNZ     TR_RUN6
        MOV.B   #23H,&200H      ;第一模块 8 灯显示值:红红红
        MOV.B   #37H,&222H      ;时间计数值
        MOV.B   #0,&20CH        ;          第 9 灯显示值
        MOV.B   #20H,&203H      ;第二模块：红绿红
        MOV.B   #15H,&223H
        MOV.B   #0,&20DH
        MOV.B   #23H,&206H      ;第三模块
```

```
                MOV.B   #37H,&224H
                MOV.B   #0,&20EH
                MOV.B   #20H,&209H      ;第四模块
                MOV.B   #15H,&225H
                MOV.B   #0,&20FH
                MOV.B   #5,&226H
TR_RUN6         CMP.B   #82,&220H       ;进行了 82 s 之后,进入状态 6
                JNZ     TR_RUN7
                MOV.B   #23H,&200H      ;第一模块 8 灯显示值:红红红
                MOV.B   #0,&20CH        ;          第 9 灯显示值(亮与熄)
                MOV.B   #28H,&203H      ;第二模块:红黄红
                MOV.B   #2H,&223H
                MOV.B   #0,&20EH
                MOV.B   #23H,&206H      ;第三模块
                MOV.B   #0,&20EH
                MOV.B   #28H,&209H      ;第四模块
                MOV.B   #2H,&225H
                MOV.B   #0,&20FH
                MOV.B   #6,&226H
TR_RUN7         CMP.B   #84,&220H       ;进行了 84 s 之后,进入状态 7
                JNZ     TR_RUN9
                MOV.B   #24H,&200H      ;第一模块 8 灯显示值:红红绿
                MOV.B   #1,&20CH        ;          第 9 灯显示值
                MOV.B   #22H,&203H      ;第二模块:绿红绿
                MOV.B   #20H,&223H
                MOV.B   #1,&20EH
                MOV.B   #24H,&206H      ;第三模块
                MOV.B   #1,&20EH
                MOV.B   #22H,&209H      ;第四模块
                MOV.B   #20H,&225H
                MOV.B   #1,&20FH
                MOV.B   #7,&226H
TR_RUN8         CMP.B   #104,&220H      ;进行了 104 s 之后,返回状态 0,循环结束
                JNZ     TR_RUNEND
                MOV.B   #0FFH,&220H     ;220H 复位
                MOV.B   #0,&226H
TR_RUNEND       RET
```

7. 计数器减 1 计数

如果只有前面的交通灯运行程序,则交通灯的显示能完全符合需要,但数码管的显示只有在各个状态转换的时刻所发生的改变,而没有倒计时显示。倒计时实质为十进制的减 1 计数。在 MSP430 的指令系统中,提供了二进制加法、十进制加法及二进制减法,而没有十进制减法。其实办法很简单,用十进制数加上 99,则十进制加法的结果相当于十进制的减 1。每当 1

s 的到来运行下面的程序，实现 4 个灯模块的减 1 计数。

```
TR_CONT   CLRC                          ；所有模块上的计数器减 1 计数
          DADD. B   #99H,&222H
          CLRC
          DADD. B   #99H,&223H
          CLRC
          DADD. B   #99H,&224H
          CLRC
          DADD. B   #99H,&225H
          RET
```

8. 绿灯的闪烁

实际的交通灯运行中，当绿灯结束后，显示黄灯，黄灯之后，为红灯。在黄灯期间，过线的车辆继续前进，线内的车辆则不能行驶。所以一般都在绿灯要结束时闪烁显示，提醒驾驶员要减速，准备停车等红灯。绿灯的闪烁其实并不麻烦。首先要判断在什么时间内可以闪烁，然后在可以闪烁的时间内每到中断时，奇数次两次中断绿灯显示，偶数次两次中断绿灯不显示，就达到了绿灯闪烁的效果。闪烁 2～3 s，闪烁频率为每秒 2.5 次。

首先要确定闪烁的时间段。这里以 1～104 s 的时间基准为依据。由图 5.46 和图 5.47 可看出：45～47 s 前后路直行绿灯闪烁；61～64 s 前后路左右转弯绿灯闪烁，同时左右路右转弯绿灯也闪烁；78～81 s 左右路直行绿灯闪烁。使用减法操作确定是否在所需的时间段内；设置一中断计数器用于确定是否满足闪烁频率的要求。程序如下：

```
FLASH_TO   PUSH. B   &220H
           CLRC
           SUB. B    #44,&220H          ；45～47 s 为前后路直行绿灯闪烁时间
           JNC       FLASH_TO1
           POP. B    &220H
           PUSH. B   &220H
           CLRC
           SUB. B    #46,&220H
           JNC       FLASH_TO1          ；是否小于 47
           CMP. B    #1,&226H           ；是否为运行状态 1
           JNZ       FLASH_TO1
           BIT. B    #2,&221H           ；是否为 0.4 s
           JNZ       FLASH_END1         ；如果都不是 则跳转
           MOV. B    #29H,&206H         ；如果条件满足，则闪烁
           MOV. B    #29H,&200H         ；先写入绿灯不亮的显示码
           JMP       FLASH_TO1
FLASH_END1 MOV. B    #21H,&200H
           MOV. B    #21H,&206H         ；再写入绿灯亮的显示码
           JMP       FLASH_TO1
FLASH_TO1  POP. B    &220H
           PUSH. B   &220H
```

```
                CLRC
                SUB. B    #61,&220H          ;61～64 s为前后路左右转弯绿灯闪烁时间
                JNC       FLASH_TO2          ;同时也是左右路右转弯绿灯闪烁时间
                POP. B    &220H
                PUSH. B   &220H
                CLRC
                SUB. B    #64,&220H
                JNC       FLASH_TO2          ;是否小于64
                CMP. B    #4,&226H           ;是否为运行状态4
                JNZ       FLASH_TO2
                BIT. B    #2,&221H           ;是否为0.4 s
                JNZ       FLASH_END2         ;如果都不是,则跳转
                MOV. B    #0,&20DH           ;如果条件满足,则闪烁
                MOV. B    #0,&20FH           ;先写入绿灯不亮的显示码
                MOV. B    #2AH,&206H
                MOV. B    #2AH,&200H
                MOV. B    #0,&20CH
                MOV. B    #0,&20EH
                JMP       FLASH_TO2
FLASH_END2      MOV. B    #22H,&200H         ;再写入绿灯亮的显示码
                MOV. B    #22H,&206H
                MOV. B    #1,&20CH
                MOV. B    #1,&20EH
                MOV. B    #1,&20DH
                MOV. B    #1,&20FH
                JMP       FLASH_TO2
FLASH_TO2       POP. B    &220H
                PUSH. B   &220H
                CLRC
                SUB. B    #78,&220H          ;78～81 s为左右路直行绿灯闪烁时间
                JNC       FLASH_TO3
                POP. B    &220H
                PUSH. B   &220H
                CLRC
                SUB. B    #81,&220H
                JNC       FLASH_TO3          ;是否小于81
                CMP. B    #6,&226H           ;否为运行状态6
                JNZ       FLASH_TO3
                BIT. B    #2,&221H           ;是否为0.4 s
                JNZ       FLASH_END3         ;如果都不是,则跳转
                MOV. B    #2CH,&203H         ;先写入绿灯不亮的显示码
                MOV. B    #2CH,&209H
                JMP       FLASH_TO3
```

```
FLASH_END3  MOV. B    #20H,&203H
            MOV. B    #20H,&209H        ;再写入绿灯亮的显示码
            JMP       FLASH_TO3
FLASH_TO3   POP. B    &220H
            RET
```

9. 违章车辆的检测与照相机的控制

在现代化的交通设施中电子警察扮演着重要角色。在上面所设计的交通灯中稍加改进，就可实现电子警察的功能。首先要检测通过的车辆，然后要能判断该通过的车辆是否违章。如果违章，则通知照相机将违章车辆的牌号拍照下来，送交通部门处理。

使用红外发射与接收实现车辆通过的检测，如图 5.48 所示。

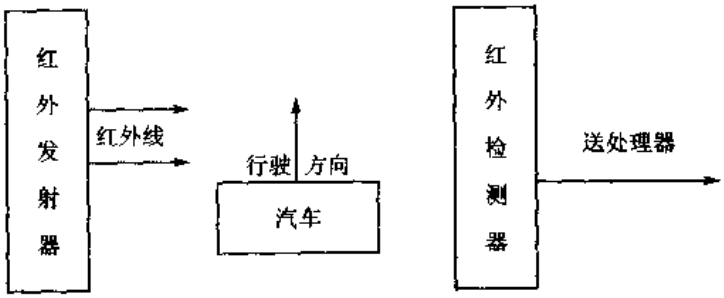


图 5.48　车辆通过检测原理

红外线发射器一直发射着红外线，红外检测器将接收到的信号送处理器，与 MSP430F1121 端口 P1.2 连接。当没有车辆通过时，检测器接收到红外线，经三极管输出低电平，P1.2=0；当有车辆通过时，检测器输出高电平，P1.2=1，这时 P1.2 发生上升沿，将引起中断。在中断处理程序中判断该通过的车辆是否被允许通过。如果允许，则退出；如果不允许，则通知照相机拍照。下面的程序只针对直行线路。

```
;P1 口中断服务程序,在非绿灯时间段内,如果有车通过,则开闪光灯对汽车牌号照相
P1_INT      BIC. B    #04H,&P1IFG
            PUSH. B   &226H
            DINT
            PUSH      R15
            MOV       #200,R15
P1_INTLOOP1 DEC       R15               ;延时消除抖动(注意时间不能长)
            JNZ       P1_INTLOOP1
            POP       R15
            BIT. B    #04H,&P1IN
            JZ        P1_INTEND
            CLRC                        ;判断是否为禁止通行时段
            SUB. B    #2,&226H          ;运行状态标志小于2时允许通行,其他时间不能通行
            JNC       P1_INTEND
            BIC. B    #01H,&P1OUT
            BIC. B    #02H,&P1OUT       ;输出信号给照相机
            PUSH      R15
```

3 个按键。因为大多数的事情是掌上机自动处理的，需键盘操作较少。汉字库使用 24C1024 可存放系统需要的全部汉字点阵。记录器用于操作票文件的存放与操作记录文件的存放。在允许操作以及操作错误的情况下，使用语音提示。语音电路使用 ISD 系列语音芯片，根据需要语音时间的长短选择具体的型号。

使用模块化进行功能程序设计，这里将与 12864 液晶模块相关的汉字显示程序说明如下，其余略。

所有电路使用两节普通干电池，可连续工作半年(不开液晶背光)。

硬件连接如下：

```
//先片选:CS1,CS2   连接 P35 与 P26
//D/I=0            连接 P3.2
//R-W=0            连接 P3.3
//数据             连接 P2 口
//E:0-1-0          连接 P3.4

void wcode(uchar c,uchar cs1,uchar cs2)
  {                                          // 写指令子程序
    //先片选:CS1,CS2
    //D/I=0
    //R-W=0
    //数据
    //E:0-1-0
    if(cs1==1)
      P3OUT|=BIT5;
      else  P3OUT&=~BIT5;
    if(cs2==1)
      P3OUT|=BIT6;
      else  P3OUT&=~BIT6;                    // 操作左半屏,还是操作右半屏
    P3OUT&=~BIT2;                            // D/I=0,表示写入数据为命令
    P3OUT&=~BIT3;                            // W/R=0,写操作
    P2OUT=c;                                 // 具体参数由调用程序设置
    P3OUT&=~BIT4;                            // E 信号为低—高—低
    delay(3);                                // 以满足时序要求
    P3OUT|=BIT4;
    delay(3);
    P3OUT&=~BIT4;
  }

void wdata(uchar c,uchar cs1,uchar cs2)
  {                                          // 写数据子程序
    //先片选:CS1,CS2    :P35,P36
    //D/I=1             :P32
    //R-W=0             :P33
```

```
        //数据              :P2
        //E:0—1—0           :P34

        if(cs1==1)
            P3OUT|=BIT5;
            else  P3OUT&=~BIT5;
        if(cs2==1)
            P3OUT|=BIT6;
            else  P3OUT&=~BIT6;
        P3OUT|=BIT2;
        P3OUT&=~BIT3;
        P2OUT=c;                                  //此处参数为将写入显示存储器的显示数据
        P3OUT&=~BIT4;
        delay(3);
        P3OUT|=BIT4;
        delay(3);
        P3OUT&=~BIT4;
    }

void set_startline(uchar i)
{
        i=0xc0+i;                                 //设置开始行子程序
        wcode(i,1,1);
}
void set_adr(uchar x,uchar y)
{
        x=x+0xb8;y=y+0x40;                        //设置地址子程序
        wcode(x,1,1);
        wcode(y,1,1);
}

void dison_off(uchar o)
{                                                 //显示开关子程序
        o=o+0x3e;
        wcode(o,1,1);
}

void reset(void)
{                                                 //液晶显示屏复位子程序
        //RST=0,1           :P37
        P3OUT&=~BIT7;
        delay(50);
        P3OUT|=BIT7;
```

```
{
uchar i,y,css;
uchar ss=0x0;
    y=x*12;
    if(ss==1)
       css=0;
       else css=1;
         set_adr(2*lline,y);
        set_startline(0);
        for(i=0;i<12;i++)                    // 先写上 12 列点阵
            {
            wdata(*(p+i),css,ss);
            }
        y=x*12+64;
        set_adr(2*lline+1,y);
        set_startline(0);
        for(i=12;i<24;i++)                   // 再写下 12 列点阵数据
            {
            wdata(*(p+i),css,ss);
            }
   }
```

附录表 1.2 MSP430F4XX 系列

名 称	地 址	位							
		7	6	5	4	3	2	1	0
模块使能 2 ME2	0005H			UTXE1 rw-0	URXE1 USPIE1 rw 0				
模块使能 1 ME1	0004H	UTXE0 rw-0	URXE0 USPIE0 rw-0						
中断标志 2 IFG2	0003H			UTXIFG1 rw 1	URXIFG1 rw-0				
中断标志 1 IFG1	0002H	UXIFG0 rw-1	URXIFG0 rw-0		NMIFG rw-0			OFIFG rw-1	WDTIFG rw-0
中断使能 2 IE2	0001H			UTXIE1 rw-0	URXIE1 rw-0				
中断使能 1 IE1	0000H	UTXIE0 rw-0	URXIE0 rw-0	ACCVIE rw-0	NMIIE rw-0			OFIE rw-0	WDTIE rw-0

附录表 1.3 MSP430X3XX 系列

名 称	地 址	位							
		7	6	5	4	3	2	1	0
	000FH								
模块使能 2 ME2	0005H							UTXE rw-0	URXE USPIIE rw-0
模块使能 1 ME1	0004H								
中断标志 2 IFG2	0003H	BTIFG rw					ADIFG rw-0	UTXIFG rw-0	URXIFG rw-0
中断标志 1 IFG1	0002H				NMIIFG rw-0	P0IFG.1 rw-0	P0IFG.0 rw-0	OFIFG rw-1	WDTIFG rw-0
中断使能 2 IE2	0001H	BTIE rw-0				TPIE‡ rw-0	ADIE† TPIE† rw-0	UTXIE rw-0	URXIE rw-0
中断使能 1 IE1	0000H					P0IE.1 rw-0	P0IE.0 rw-0	OFIE rw-0	WDTIE rw-0

† ADIE:ADC12+2 中断使能(32x devices)。
TPIE:定时器/端口中断使能(31x devices)。
‡ TPIE:定时器/端口中断使能(32x,33x devices)。

附录 2 I/O 端口

附录表 2 MSP430F1XX 系列、MSP430F4XX 系列、MSP430X3XX 系列

名 称	地 址	位							
		7	6	5	4	3	2	1	0
功能选择 P4SEL	001FH	P4SEL.7 rw-0	P4SEL.6 rw-0	P4SEL.5 rw-0	P1SEL.4 rw-0	P1SEL.3 rw-0	P4SEL.2 rw-0	P4SEL.1 rw-0	P4SEL.0 rw-0
方向选择 P4DIR	001EH	P4DIR.7 rw-0	P4DIR.6 rw-0	P4DIR.5 rw-0	P1DIR.4 rw-0	P4DIR.3 rw-0	P1DIR.2 rw-0	P4DIR.1 rw-0	P4DIR.0 rw-0
输出寄存 P1OUT	001DH	P4OUT.7 rw	P4OUT.6 rw	P4OUT.5 rw	P1OUT.4 rw	P4OUT.3 rw	P4OUT.2 rw	P1OUT.1 rw	P4OUT.0 rw
输入寄存 P4IN	001CH	P4IN.7 r	P4IN.6 r	P4IN.5 r	P1IN.4 r	P1IN.3 r	P1IN.2 r	P4IN.1 r	P4IN.0 r
功能选择 P3SEL	001BH	P3SEL.7 rw-0	P3SEL.6 rw-0	P3SEL.5 rw-0	P3SEL.1 rw-0	P3SEL.3 rw-0	P3SEL.2 rw-0	P3SEL.1 rw-0	P3SEL.0 rw-0
方向选择 P3DIR	001AH	P3DIR.7 rw-0	P3DIR.6 rw-0	P3DIR.5 rw-0	P3DIR.4 rw-0	P3DIR.3 rw-0	P3DIR.2 rw-0	P3DIR.1 rw-0	P3DIR.0 rw-0
输出寄存 P3OUT	0019H	P3OUT.7 rw	P3OUT.6 rw	P3OUT.5 rw	P3OUT.4 rw	P3OUT.3 rw	P3OUT.2 rw	P3OUT.1 rw	P3OUT.0 rw
输入寄存 P3IN	0018H	P3IN.7 r	P3IN.6 r	P3IN.5 r	P3IN.4 r	P3IN.3 r	P3IN.2 r	P3IN.1 r	P3IN.0 r
	0017H								
	0016H								
中断使能 P0IE	0015H	P0IE.7 rw-0	P0IE.6 rw-0	P0IE.5 rw-	P0IE.4 rw-	P0IE.3 rw-	P0IE.2 rw-	† r0	† r0
功能选择 P0IES	0014H	P0IES.7 rw	P0IES.6 rw	P0IES.5 rw	P0IES.1 rw	P0IES.3 rw	P0IES.2 rw	P0IES.1 rw	P0IES.0 rw
中断标志 P0IFG	0013H	P0IFG.7 rw-0	P0IFG.6 rw-0	P0IFG.5 rw-0	P0IFG.4 rw-0	P0IFG.3 rw-0	P0IFG.2 rw-0	† rw-0	† rw-0
方向选择 P0DIR	0012H	P0DIR.7 rw-0	P0DIR.6 rw-0	P0DIR.5 rw-0	P0DIR.4 rw-0	P0DIR.3 rw-0	P0DIR.2 rw-0	P0DIR.1 rw-0	P0DIR.0 rw-0
输出寄存 P0OUT	0011H	P0OUT.7 rw	P0OUT.6 rw	P0OUT.5 rw	P0OUT.4 rw	P0OUT.3 rw	P0OUT.2 rw	P0OUT.1 rw	P0OUT.0 rw
输入寄存 P0IN	0010H	P0IN.7 r	P0IN.6 r	P0IN.5 r	P0IN.4 r	P0IN.3 r	P0IN.2 r	P0IN.1 r	P0IN.0 r
功能选择 P6SEL	0037H	P6SEL.7 rw-0	P6SEL.6 rw-0	P6SEL.5 rw-0	P6SEL.4 rw-0	P6SEL.3 rw-0	P6SEL.2 rw-0	P6SEL.1 rw-0	P6SEL.0 rw-0
方向选择 P6DIR	0036H	P6DIR.7 rw-0	P6DIR.6 rw-0	P6DIR.5 rw-0	P6DIR.4 rw-0	P6DIR.3 rw-0	P6DIR.2 rw-0	P6DIR.1 rw-0	P6DIR.0 rw-0
输出寄存 P6OUT	0035H	P6OUT.7 rw	P6OUT.6 rw	P6OUT.5 rw	P6OUT.4 rw	P6OUT.3 rw	P6OUT.2 rw	P6OUT.1 rw	P6OUT.0 rw
输入寄存 P6IN	0034H	P6IN.7 r	P6IN.6 r	P6IN.5 r	P6IN.4 r	P6IN.3 r	P6IN.2 r	P6IN.1 r	P6IN.0 r
功能选择 P5SEL	0033H	P5SEL.7 rw-0	P5SEL.6 rw-0	P5SEL.5 rw-0	P5SEL.4 rw-0	P5SEL.3 rw-0	P5SEL.2 rw-0	P5SEL.1 rw-0	P5SEL.0 rw-0
方向选择 P5DIR	0032H	P5DIR.7 rw-0	P5DIR.6 rw-0	P5DIR.5 rw-0	P5DIR.4 rw-0	P5DIR.3 rw-0	P5DIR.2 rw-0	P5DIR.1 rw-0	P5DIR.0 rw-0

续附录表 2

名 称	地 址	位							
		7	6	5	4	3	2	1	0
输出寄存 P5OUT	0031H	P5OUT.7 rw	P5OUT.6 rw	P5OUT.5 rw	P5OUT.4 rw	P5OUT.3 rw	P5OUT.2 rw	P5OUT.1 rw	P5OUT.0 rw
输入寄存 P5IN	0030H	P5IN.7 r	P5IN.6 r	P5IN.5 r	P5IN.4 r	P5IN.3 r	P5IN.2 r	P5IN.1 r	P5IN.0 r
	002FH								
功能选择 P2SEL	002EH	P2SEL.7 rw-0	P2SEL.6 rw-0	P2SEL.5 rw-0	P2SEL.4 rw-0	P2SEL.3 rw-0	P2SEL.2 rw-0	P2SEL.1 rw-0	P2SEL.0 rw-0
中断使能 P2IE	002DH	P2IE.7 rw-0	P2IE.6 rw-0	P2IE.5 rw-0	P2IE.4 rw-0	P2IE.3 rw-0	P2IE.2 rw-0	P2IE.1 rw-0	P2IE.0 rw-0
中断沿选择 P2IES	002CH	P2IES.7 rw	P2IES.6 rw	P2IES.5 rw	P2IES.4 rw	P2IES.3 rw	P2IES.2 rw	P2IES.1 rw	P2IES.0 rw
中断标志 P2IFG	002BH	P2IFG.7 rw-0	P2IFG.6 rw-0	P2IFG.5 rw-0	P2IFG.4 rw-0	P2IFG.3 rw-0	P2IFG.2 rw-0	P2IFG.1 rw-0	P2IFG.0 rw-0
方向选择 P2DIR	002AH	P2DIR.7 rw-0	P2DIR.6 rw-0	P2DIR.5 rw-0	P2DIR.4 rw-0	P2DIR.3 rw-0	P2DIR.2 rw-0	P2DIR.1 rw-0	P2DIR.0 rw-0
输出寄存 P2OUT	0029H	P2OUT.7 rw	P2OUT.6 rw	P2OUT.5 rw	P2OUT.4 rw	P2OUT.3 rw	P2OUT.2 rw	P2OUT.1 rw	P2OUT.0 rw
输入寄存 P2IN	0028H	P2IN.7 r	P2IN.6 r	P2IN.5 r	P2IN.4 r	P2IN.3 r	P2IN.2 r	P2IN.1 r	P2IN.0 r
	0027H								
功能选择 P1SEL	0026H	P1SEL.7 rw-0	P1SEL.6 rw-0	P1SEL.5 rw-0	P1SEL.4 rw-0	P1SEL.3 rw-0	P1SEL.2 rw-0	P1SEL.1 rw-0	P1SEL.0 rw-0
中断使能 P1IE	0025H	P1IE.7 rw-0	P1IE.6 rw-0	P1IE.5 rw-0	P1IE.4 rw-0	P1IE.3 rw-0	P1IE.2 rw-0	P1IE.1 rw-0	P1IE.0 rw-0
中断沿选择 P1IES	0024H	P1IES.7 rw	P1IES.6 rw	P1IES.5 rw	P1IES.4 rw	P1IES.3 rw	P1IES.2 rw	P1IES.1 rw	P1IES.0 rw
中断标志 P1IFG	0023H	P1IFG.7 rw-0	P1IFG.6 rw-0	P1IFG.5 rw-0	P1IFG.4 rw-0	P1IFG.3 rw-0	P1IFG.2 rw-0	P1IFG.1 rw-0	P1IFG.0 rw-0
方向选择 P1DIR	0022H	P1DIR.7 rw-0	P1DIR.6 rw-0	P1DIR.5 rw-0	P1DIR.4 rw-0	P1DIR.3 rw-0	P1DIR.2 rw-0	P1DIR.1 rw-0	P1DIR.0 rw-0
输出寄存 P1OUT	0021H	P1OUT.7 rw	P1OUT.6 rw	P1OUT.5 rw	P1OUT.4 rw	P1OUT.3 rw	P1OUT.2 rw	P1OUT.1 rw	P1OUT.0 rw
输入寄存 P1IN	0020H	P1IN.7 r	P1IN.6 r	P1IN.5 r	P1IN.4 r	P1IN.3 r	P1IN.2 r	P1IN.1 r	P1IN.0 r

附录 3 MSP430F4XX 系列基本定时器(Basic Timer1)

附录表 3 MSP430F4XX 系列

名 称	地 址	位							
		7	6	5	4	3	2	1	0
8 位计数器 BTCNT2	0047H	2^7 rw	2^6 rw	2^5 rw	2^4 rw	2^3 rw	2^2 rw	2^1 rw	2^0 rw
8 位计数器 BTCNT1	0046H	2^7 rw	2^6 rw	2^5 rw	2^4 rw	2^3 rw	2^2 rw	2^1 rw	2^0 rw
BTCTL	0040H	SSEL rw	Hold rw	DIV rw	FRFQ1 rw	FRFQ0 rw	IP2 rw	IP1 rw	IP0 rw

附录 4　MSP430X3XX 系列、MSP430X4XX 系列定时器/端口

附录表 4　MSP430X3XX 系列、MSP430X4XX 系列

名　称	地　址	位							
		7	6	5	4	3	2	1	0
使能寄存器 Timer/Port TPE	01FH	TPSSEL3 rw－0	TPSSEL2 rw－0	TPE.5 rw－0	TPE.4 rw－0	TPE.3 rw－0	TPE.2 rw－0	TPE.1 rw－0	TPE.0 rw－0
数据寄存器 Timer/Port TPD	01EH	B16 rw－0	CPON rw－0	TPD.5 rw－0	TPD.4 rw－0	TPD.3 rw－0	TPD.2 rw－0	TPD.1 rw－0	TPD.0 rw－0
计数器 2 Timer_Port TPCNT2	01DH	2^7 rw	2^6 rw	2^5 rw	2^4 rw	2^3 rw	2^2 rw	2^1 rw	2^0 rw
计数器 1 Timer/Port TPCNT1	01CH	2^7 rw	2^6 rw	2^5 rw	2^4 rw	2^3 rw	2^2 rw	2^1 rw	2^0 rw
控制寄存器 Timer_Port TPCTL	01BH	TPSSEL1 rw－0	TPSSEL0 rw－0	ENB rw－0	ENA rw－0	EN1 rw－0	RC2FG rw－0	RC1FG rw－0	EN1FG rw－0
8 位计数器 Basic Timer BTCNT2	0047H	2^7 rw	2^6 rw	2^5 rw	2^4 rw	2^3 rw	2^2 rw	2^1 rw	2^0 rw
8 位计数器 Basic Timer BTCNT1	0046H	2^7 rw	2^6 rw	2^5 rw	2^4 rw	2^3 rw	2^2 rw	2^1 rw	2^0 rw
	0045H								
8 位计数器 Timer/Counter TCDAT	0044H	TCDAT.7 rw	TCDAT.6 rw	TCDAT.5 rw	TCDAT.4 rw	TCDAT.3 rw	TCDAT.2 rw	TCDAT.1 rw	TCDAT.0 rw
预置寄存器 Timer/Counter TCPLD	0043H	TCPLD.7 rw	TCPLD.6 rw	TCPLD.5 rw	TCPLD.4 rw	TCPLD.3 rw	TCPLD.2 rw	TCPLD.1 rw	TCPLD.0 rw
控制寄存器 Timer/Counter TCCTL	0042H	SSEL1 rw－0	SSEL0 rw－0	ISCTL rw－0	TXEN rw－0	ENCNT rw－0	RXACT rw－0	TXD rw－0	RXD r(－1)
	0041H								
Basic Timer BTCTL	0040H	SSEL rw	Hold rw	DIV rw	FRFQ1 rw	FRFQ0 rw	IP2 rw	IP1 rw	IP0 rw

附录 5　MSP430F1XX 系列基本时钟

附录表 5　MSP430F1XX 系列

名　称	地　址	位							
		7	6	5	4	3	2	1	0
BCSCTL2	0058H	SELM.1 rw－0	SELM.0 rw－0	DIVM.1 rw－0	DIVM.0 rw－0	SELS rw－0	DIVS.1 rw－0	DIVS.0 rw－0	DCOR rw－0

续附录表 5

名 称	地 址	位							
		7	6	5	4	3	2	1	0
BCSCTL1	0057H	XT2OFF rw-(1)	XTS rw-(0)	DIVA.1 rw-(0)	DIVA.0 rw (0)	XT5V rw-0	Rsel.2 rw-1	Rsel.1 rw-0	Rsel.0 rw-0
DCOCTL	0056H	DCO.2 rw-0	DCO.1 rw 1	DCO.0 rw-1	MOD.4 rw-0	MOD.3 rw-0	MOD.2 rw 0	MOD.1 rw 0	MOD.0 rw-0

附录 6 MSP430F4XX 系列 FLL+模块

附录表 6 MSP430F4XX 系列

名 称	地 址	位							
		7	6	5	4	3	2	1	0
FLL+控制寄存器 0 FLL+CTL0	0053H	M rw-0	2^5 rw 0	2^5 rw 0	2^4 rw 0	2^3 rw-0	2^2 rw-0	2^1 rw 0	2^0 rw-0
FLL+控制寄存器 1 FLL+CTL1	0054H	M rw-0	2^6 rw 0	2^5 rw 0	2^4 rw 0	2^3 rw-0	2^2 rw-0	2^1 rw 1	2^0 rw-1
频率控制寄存器 0 SCFQCTL	0052H	M rw-0	2^7 rw-0	2^6 rw-0	2^5 rw-1	2^4 rw-1	2^3 rw-1	2^2 rw-1	2^1 rw 1
频率合成器 1 SCFI1	0051H	2^9 rw-0	2^8 rw-0	2^7 rw-0	2^6 rw-0	2^5 rw-0	2^4 rw-0	2^3 rw-0	2^2 rw-0
频率合成器 0 SCFI0	0050H	D rw-0	D rw-1	FN_8 rw-0	FN_1 rw-0	FN_3 rw-0	FN_2 rw-0	2^1 rw-0	2^0 rw-0

附录 7 MSP430X3XX 系列 FLL 模块

附录表 7 MSP430X3XX 系列

名 称	地 址	位							
		7	6	5	4	3	2	1	0
频率控制器 SCFQCTL	0052H	M rw-0	2^6 rw-0	2^5 rw-0	2^4 rw-1	2^3 rw-1	2^2 rw-1	2^1 rw-1	2^0 rw 1
频率合成器 SCFI1	0051H	2^9 rw-0	2^8 rw 0	2^7 rw-0	2^6 rw-0	2^5 rw-0	2^4 rw-0	2^3 rw 0	2^2 rw-0
频率合成器 SCFI0	0050H	0 r	0 r	0 r	FN_4 rw-0	FN_3 rw-0	FN_2 rw-0	2^1 rw-0	2^0 rw-0

附录 8 模拟比较器

附录表 8 MSP430 全系列

名 称	地 址	位							
		7	6	5	4	3	2	1	0
比较器 A 端口使能 CAPD	005BH	CAPD.7 rw-(0)	CAPD.6 rw-(0)	CAPD.5 rw-(0)	CAPD.4 rw-(0)	CAPD.3 rw (0)	CAPD.2 rw-(0)	CAPD.1 rw-(0)	CAPD.0 rw-(0)
比较器 A 控制寄存器 2 CACTL2	005AH	CACTL2.7 rw-(0)	CACTL2.6 rw-(0)	CACTL2.5 rw-(0)	CACTL2.4 rw-(0)	CA1 rw-(0)	CA0 rw (0)	CAF rw-(0)	CAOUT r-(0)
比较器 A 控制寄存器 1 CACTL1	0059H	CAEX rw-(0)	CARSEL rw (0)	CAREF1 rw-(0)	CAREF0 rw-(0)	CAON rw-(0)	CAIES rw-(0)	CAIE rw-(0)	CAIFG rw-(0)

附录 11 MSP430F4XX 系列 SVS 模块

附录表 11 MSP430F4XX 系列

名 称	地 址	位							
		7	6	5	4	3	2	1	0
SVSCTL	0055H	VLD. 3 rw—(0)	VLD. 2 rw—(0)	VLD. 1 rw—(0)	VLD. 0 rw—(0)	PORON rw—(0)	SVSON r	SVSOP r	SVSFG rw—(0)

附录 12 UART 模式下的两个串口

附录表 12 MSP430 系列

名 称		地 址	位							
			7	6	5	4	3	2	1	0
USART1	发送缓存器 U1TXBUF	07FH	2^7 rw	2^6 rw	2^5 rw	2^4 rw	2^3 rw	2^2 rw	2^1 rw	2^0 rw
	接收缓存器 U1RXBUF	07EH	2^7 r	2^6 r	2^5 r	2^4 r	2^3 r	2^2 r	2^1 r	2^0 r
	波特率 U1BR1	07DH	2^{15} rw	2^{14} rw	2^{13} rw	2^{12} rw	2^{11} rw	2^{10} rw	2^9 rw	2^8 rw
	波特率 U1BR0	07CH	2^7 rw	2^6 rw	2^5 rw	2^4 rw	2^3 rw	2^2 rw	2^1 rw	2^0 rw
	调整控制寄存器 U1MCTL	07BH	m7 rw	m6 rw	m5 rw	m4 rw	m3 rw	m2 rw	m1 rw	m0 rw
	接收控制器 U1RCTL	07AH	FE rw—0	PE rw—0	OE rw—0	BRK rw 0	URXEIE rw—0	URXWIE rw—0	RXWAKE rw—0	RXERR rw—0
	发送控制器 U1TCTL	079H	Unused rw—0	CKPL rw—0	SSEL1 rw—0	SSEL0 rw—0	URXSE rw—0	TXWAKE rw—0	Unused rw—0	TXEPT rw—1
	控制寄存器 U1CTL	078H	PENA rw—0	PEV rw—0	SP rw—0	CHAR rw- 0	Listen rw—0	SYNC rw—0	MM rw—0	SWRST rw—1
USART0	发送缓存器 U0TXBUF	077H	2^7 rw	2^6 rw	2^5 rw	2^4 rw	2^3 rw	2^2 rw	2^1 rw	2^0 rw
	接收缓存器 U0RXBUF	076H	2^7 r	2^6 r	2^5 r	2^4 r	2^3 r	2^2 r	2^1 r	2^0 r
	波特率寄存器 U0BR1	075H	2^{15} rw	2^{14} rw	2^{13} rw	2^{12} rw	2^{11} rw	2^{10} rw	2^9 rw	2^8 rw

续附录表 12

名　称		地　址	位							
			7	6	5	4	3	2	1	0
USART1	波特率寄存器 U0BR0	074H	2^7 rw	2^6 rw	2^5 rw	2^4 rw	2^3 rw	2^2 rw	2^1 rw	2^0 rw
	调整寄存器 U0MCTL	073H	m7 rw	m6 rw	m5 rw	m4 rw	m3 rw	m2 rw	m1 rw	m0 rw
	接收控制器 U0RCTL	072H	FE rw-0	PE rw-0	OE rw-0	BRK rw-0	URXEIE rw-0	URXWIE rw-0	RXWAKE rw-0	RXERR rw-0
	发送控制器 U0TCTL	071H	Unused rw-0	CKPL rw-0	SSEL1 rw-0	SSEL0 rw-0	URXSE rw-0	TXWAKE rw-0	Unused rw-0	TXEPT rw-1
	串口控制器 U0CTL	070H	PENA rw-0	PEV rw-0	SP rw-0	CHAR rw-0	Listen rw-0	SYNC rw-0	MM rw-0	SWRST rw-1

注：在 MSP430 系列的器件中，有的有一个，有的一个也没有。

附录 13　SPI 模式下的两个串口

附录表 13　MSP430 系列

名　称		地　址	位							
			7	6	5	4	3	2	1	0
USART1	发送缓存器 U1TXBUF	07FH	2^7 rw	2^6 rw	2^5 rw	2^4 rw	2^3 rw	2^2 rw	2^1 rw	2^0 rw
	接收缓存器 U1RXBUF	07EH	2^7 r	2^6 r	2^5 r	2^4 r	2^3 r	2^2 r	2^1 r	2^0 r
	波特率寄存器 U1BR1	07DH	2^{15} rw	2^{14} rw	2^{13} rw	2^{12} rw	2^{11} rw	2^{10} rw	2^9 rw	2^8 rw
	波特率寄存器 U1BR0	07CH	2^7 rw	2^6 rw	2^5 rw	2^4 rw	2^3 rw	2^2 rw	2^1 rw	2^0 rw
	波特率调整器 U1MCTL	07BH	m7 rw	m6 rw	m5 rw	m4 rw	m3 rw	m2 rw	m1 rw	m0 rw
	接收控制器 U1RCTL	07AH	FE rw-0	Undef rw-0	OE rw-0	Undef rw-0	Unused rw-0	Unused rw-0	Undef rw-0	Undef rw-0
	发送控制器 U1TCTL	079H	CKPH rw-0	CKPL rw-0	SSEL1 rw-0	SSEL0 rw-0	Unused rw-0	Unused rw-0	STC rw-0	TXEPT rw-1
	串口控制器 U1CTL	078H	Unused rw-0	Unused rw-0	Unused rw-0	CHAR rw-0	Listen rw-0	SYNC rw-0	MM rw-0	SWRST rw-1

续附录表 13

名称		地址	位							
			7	6	5	4	3	2	1	0
USART0	发送缓存器 U0TXBUF	077H	2^7 rw	2^6 rw	2^5 rw	2^4 rw	2^3 rw	2^2 rw	2^1 rw	2^0 rw
	接收缓存器 U0RXBUF	076H	2^7 r	2^6 r	2^5 r	2^4 r	2^3 r	2^2 r	2^1 r	2^0 r
	波特率 U0BR1	075H	2^{15} rw	2^{14} rw	2^{13} rw	2^{12} rw	2^{11} rw	2^{10} rw	2^9 rw	2^8 rw
	波特率 U0BR0	074H	2^7 rw	2^6 rw	2^5 rw	2^4 rw	2^3 rw	2^2 rw	2^1 rw	2^0 rw
	调整控制器 U0MCTL	073H	m7 rw	m6 rw	m5 rw	m4 rw	m3 rw	m2 rw	m1 rw	m0 rw
	接收控制 U0RCTL	072H	FE rw—0	Undef rw—0	OE rw—0	Undef rw—0	Unused rw—0	Unused rw—0	Undef rw—0	Undef rw—0
	发送控制 U0TCTL	071H	CKPH rw—0	CKPL rw—0	SSEL1 rw—0	SSEL0 rw—0	Unused rw—0	Unused rw—0	STC rw—0	TXEPT rw—1
	串口控制 U0CTL	070H	Unused rw—0	Unused rw—0	Unused rw—0	CHAR rw—0	Listen rw—0	SYNC rw—0	MM rw—0	SWRST rw—1

注：在 MSP430 系列的器件中，有的有一个，有的一个也没有。

附录 14 FLASH 系列 ADC12 模块(1XX、4XX)

附录表 14.1 ADC12MCTLx

名称	地址	位							
		7	6	5	1	3	2	1	0
ADC12 MCTL15†	008FH	EOS rw—(0)	Sref.2 rw—(0)	Sref.1 rw—(0)	Sref.0 rw—(0)	INCH.3 rw—(0)	INCH.2 rw—(0)	INCH.1 rw—(0)	INCH.0 rw—(0)
ADC12 MCTL14†	008EH	EOS rw—(0)	Sref.2 rw—(0)	Sref.1 rw—(0)	Sref.0 rw—(0)	INCH.3 rw—(0)	INCH.2 rw—(0)	INCH.1 rw—(0)	INCH.0 rw—(0)
ADC12 MCTL13†	008DH	EOS rw—(0)	Sref.2 rw—(0)	Sref.1 rw—(0)	Sref.0 rw—(0)	INCH.3 rw—(0)	INCH.2 rw—(0)	INCH.1 rw—(0)	INCH.0 rw—(0)
ADC12 MCTL12†	008CH	EOS rw—(0)	Sref.2 rw—(0)	Sref.1 rw—(0)	Sref.0 rw—(0)	INCH.3 rw—(0)	INCH.2 rw—(0)	INCH.1 rw—(0)	INCH.0 rw—(0)
ADC12 MCTL11†	008BH	EOS rw—(0)	Sref.2 rw—(0)	Sref.1 rw—(0)	Sref.0 rw—(0)	INCH.3 rw—(0)	INCH.2 rw—(0)	INCH.1 rw—(0)	INCH.0 rw—(0)
ADC12 MCTL10†	008AH	EOS rw—(0)	Sref.2 rw—(0)	Sref.1 rw—(0)	Sref.0 rw—(0)	INCH.3 rw—(0)	INCH.2 rw—(0)	INCH.1 rw—(0)	INCH.0 rw—(0)
ADC12 MCTL9†	0089H	EOS rw—(0)	Sref.2 rw—(0)	Sref.1 rw—(0)	Sref.0 rw—(0)	INCH.3 rw—(0)	INCH.2 rw—(0)	INCH.1 rw—(0)	INCH.0 rw—(0)
ADC12 MCTL8†	0088H	EOS rw—(0)	Sref.2 rw—(0)	Sref.1 rw—(0)	Sref.0 rw—(0)	INCH.3 rw—(0)	INCH.2 rw—(0)	INCH.1 rw—(0)	INCH.0 rw—(0)
ADC12 MCTL7†	0087H	EOS rw—(0)	Sref.2 rw—(0)	Sref.1 rw—(0)	Sref.0 rw—(0)	INCH.3 rw—(0)	INCH.2 rw—(0)	INCH.1 rw—(0)	INCH.0 rw—(0)
ADC12 MCTL6†	0086H	EOS rw—(0)	Sref.2 rw—(0)	Sref.1 rw—(0)	Sref.0 rw—(0)	INCH.3 rw—(0)	INCH.2 rw—(0)	INCH.1 rw—(0)	INCH.0 rw—(0)

续附录表 14.1

名 称	地 址	位							
		7	6	5	4	3	2	1	0
ADC12 MCTL5†	0085H	EOS rw (0)	Sref. 2 rw (0)	Sref. 1 rw (0)	Sref. 0 rw (0)	INCH. 3 rw-(0)	INCH. 2 rw-(0)	INCH. 1 rw-(0)	INCH. 0 rw-(0)
ADC12 MCTL4†	0084H	EOS rw-(0)	Sref. 2 rw-(0)	Sref. 1 rw-(0)	Sref. 0 rw-(0)	INCH. 3 rw-(0)	INCH. 2 rw-(0)	INCH. 1 rw-(0)	INCH. 0 rw-(0)
ADC12 MCTL3†	0083H	EOS rw-(0)	Sref. 2 rw-(0)	Sref. 1 rw-(0)	Sref. 0 rw-(0)	INCH. 3 rw-(0)	INCH. 2 rw-(0)	INCH. 1 rw-(0)	INCH. 0 rw-(0)
ADC12 MCTL2†	0082H	EOS rw-(0)	Sref. 2 rw-(0)	Sref. 1 rw-(0)	Sref. 0 rw-(0)	INCH. 3 rw-(0)	INCH. 2 rw-(0)	INCH. 1 rw-(0)	INCH. 0 rw-(0)
ADC12 MCTL1†	0081H	EOS rw-(0)	Sref. 2 rw (0)	Sref. 1 rw-(0)	Sref. 0 rw-(0)	INCH. 3 rw-(0)	INCH. 2 rw-(0)	INCH. 1 rw-(0)	INCH. 0 rw-(0)
ADC12 MCTL0†	0080H	EOS rw-(0)	Sref. 2 rw (0)	Sref. 1 rw-(0)	Sref. 0 rw-(0)	INCH. 3 rw-(0)	INCH. 2 rw-(0)	INCH. 1 rw-(0)	INCH. 0 rw-(0)

† 只有当 ENC=0 时修改。

附录表 14.2 ADC12MEMx

名 称	地 址	位					
		15	14	13	12	11	0
ADC12 MEM15	015EH	Unused r0	Unused r0	Unused r0	Unused r0	MSB ← Conversion Result → LSB All conversion-result bits of type rw	
ADC12 MEM14	015CH	Unused r0	Unused r0	Unused r0	Unused r0	MSB ← Conversion Result → LSB All conversion-result bits of type rw	
ADC12 MEM13	015AH	Unused r0	Unused r0	Unused r0	Unused r0	MSB ← Conversion Result → LSB All conversion-result bits of type rw	
ADC12 MEM12	0158H	Unused r0	Unused r0	Unused r0	Unused r0	MSB ← Conversion Result → LSB All conversion-result bits of type rw	
ADC12 MEM11	0156H	Unused r0	Unused r0	Unused r0	Unused r0	MSB ← Conversion Result → LSB All conversion-result bits of type rw	
ADC12 MEM10	0154H	Unused r0	Unused r0	Unused r0	Unused r0	MSB ← Conversion Result → LSB All conversion-result bits of type rw	
ADC12 MEM9	0152H	Unused r0	Unused r0	Unused r0	Unused r0	MSB ← Conversion Result → LSB All conversion-result bits of type rw	
ADC12 MEM8	0150H	Unused r0	Unused r0	Unused r0	Unused r0	MSB ← Conversion Result → LSB All conversion-result bits of type rw	
ADC12 MEM7	014EH	Unused r0	Unused r0	Unused r0	Unused r0	MSB ← Conversion Result → LSB All conversion-result bits of type rw	
ADC12 MEM6	014CH	Unused r0	Unused r0	Unused r0	Unused r0	MSB ← Conversion Result → LSB All conversion-result bits of type rw	
ADC12 MEM5	014AH	Unused r0	Unused r0	Unused r0	Unused r0	MSB ← Conversion Result → LSB All conversion-result bits of type rw	
ADC12 MEM4	0148H	Unused r0	Unused r0	Unused r0	Unused r0	MSB ← Conversion Result → LSB All conversion-result bits of type rw	
ADC12 MEM3	0146H	Unused r0	Unused r0	Unused r0	Unused r0	MSB ← Conversion Result → LSB All conversion-result bits of type rw	
ADC12 MEM2	0144H	Unused r0	Unused r0	Unused r0	Unused r0	MSB ← Conversion Result → LSB All conversion-result bits of type rw	
ADC12 MEM1	0142H	Unused r0	Unused r0	Unused r0	Unused r0	MSB ← Conversion Result → LSB All conversion-result bits of type rw	
ADC12 MEM0	0140H	Unused r0	Unused r0	Unused r0	Unused r0	MSB ← Conversion Result → LSB All conversion-result bits of type rw	

† 只有当 ENC=0 时修改。

附录表 14.3 中断使能、中断标志及转换控制寄存器

名 称	地 址	位															
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADC12IE	01A6H	ADC12 IE. 15 rw—(0)	ADC12 IE. 14 rw—(0)	ADC12 IE. 13 rw—(0)	ADC12 IE. 12 rw—(0)	ADC12 IE. 11 rw—(0)	ADC12 IE. 10 rw—(0)	ADC12 IE. 9 rw—(0)	ADC12 IE. 8 rw—(0)	ADC12 IE. 7 rw—(0)	ADC12 IE. 6 rw—(0)	ADC12 IE. 5 rw—(0)	ADC12 IE. 4 rw—(0)	ADC12 IE. 3 rw—(0)	ADC12 IE. 2 rw—(0)	ADC12 IE. 1 rw—(0)	ADC12 IE. 0 rw—(0)
ADC12IFG	01A4H	ADC12 IFG. 15 rw—(0)	ADC12 IFG. 14 rw—(0)	ADC12 IFG. 13 rw—(0)	ADC12 IFG. 12 rw—(0)	ADC12 IFG. 11 rw—(0)	ADC12 IFG. 10 rw—(0)	ADC12 IFG. 9 rw—(0)	ADC12 IFG. 8 rw—(0)	ADC12 IFG. 7 rw—(0)	ADC12 IFG. 6 rw—(0)	ADC12 IFG. 5 rw—(0)	ADC12 IFG. 4 rw—(0)	ADC12 IFG. 3 rw—(0)	ADC12 IFG. 2 rw—(0)	ADC12 IFG. 1 rw—(0)	ADC12 IFG. 0 rw—(0)
ADC12CTL1	01A2H	CSStart Add. 3† rw—(0)	CSStart Add. 2† rw—(0)	CSStart Add. 1† rw—(0)	CSStart Add. 0† rw—(0)	SHS. 1† rw—(0)	SHS. 0† rw—(0)	SHP† rw—(0)	ISSH† rw—(0)	ADC12 DIV. 2† rw—(0)	ADC12 DIV. 1† rw—(0)	ADC12 DIV. 0† rw—(0)	ADC12 SSEL. 1† rw—(0)	ADC12 SSEL. 0† rw—(0)	CONSEQ. 1 rw—(0)	CONSEQ. 0 rw—(0)	ADC12 BUSY r—(0)
ADC12CTL0	01A0H	SHT1. 3† rw—(0)	SHT1. 2† rw—(0)	SHT1. 1† rw—(0)	SHT1. 0† rw—(0)	SHT0. 3† rw—(0)	SHT0. 2† rw—(0)	SHT0. 1† rw—(0)	SHT0. 0† rw—(0)	MSC† rw—(0)	2. 5 V† rw—(0)	REFON† rw—(0)	ADC12 ON† rw—(0)	ADC12 OVIE† rw—(0)	ADC12 TVIE† rw—(0)	ENC† rw—(0)	ADC12 SC† rw—(0)

† 只有当 ENC=0 时可修改。

附录 15 MSP430F1XX 系列 ADC10 模块

附录表 15.1 ADC10CTLx

名 称	地 址	位															
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADC10CTL1	01B2H	INCH. 3† rw—(0)	INCH. 2† rw—(0)	INCH. 1† rw—(0)	INCH. 0† rw—(0)	SHS. 1† rw—(0)	SHS. 0† rw—(0)	ADC10 DF† rw—(0)	ISSH† rw—(0)	ADC10 DIV. 2† rw—(0)	ADC10 DIV. 1† rw—(0)	ADC10 DIV. 0† rw—(0)	ADC10 SSEL. 1† rw—(0)	ADC10 SSEL. 0† rw—(0)	CONS EQ. 1 rw—(0)	CONS EQ. 0 rw—(0)	ADC10 BUSY r—(0)
ADC10CTL0	01B0H	ADC10 Sref. 2† rw—(0)	ADC10 Sref. 1† rw—(0)	ADC10 Sref. 0† rw—(0)	ADC10 SHT. 1† rw—(0)	ADC10 SHT. 0† rw—(0)	ADC10 SR† rw—(0)	REF+ OUT† rw—(0)	REF BURST† rw—(0)	MSC† rw—(0)	2. 5V† rw—(0)	REFON† rw—(0)	ADC10 ON† rw—(0)	ADC10 IE† rw—(0)	ADC10 IFG† rw—(0)	ENC† rw—(0)	ADC10 SC† rw—(0)

† 只有当 ENC=0 时可修改

附录 16　MSP430X3XX 系列 ADC14 模块

附录表 16　ADC14 的寄存器

名　称	地　址	位															
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	11FH																
数据寄存器 ADAT	118H	r0	r0	R1† r0	R0† r0	2^{11} r	2^{10} r	2^{9} r	2^{8} r	2^{7} r	2^{6} r	2^{5} r	2^{4} r	2^{3} r	2^{2} r	2^{1} r	2^{0} r
Reserved	116H																
控制寄存器 ACTL	114H	ACTL. 15 r0	ACTL. 14 rw 0	ACTL. 13 rw−0	ACTL. 12 rw- 1	ACTL. 11 rw−0	ACTL. 10 rw−0	ACTL. 9 rw−0	ACTL. 8 rw−0	ACTL. 7 rw−0	ACTL. 6 rw−0	ACTL. 5 rw−0	ACTL. 4 rw−0	ACTL. 3 rw−0	ACTL. 2 rw−0	ACTL. 1 rw−0	ACTL. 0 rw- 0
输入使能寄存器 AEN	112H	r0	r0	r0	r0	r0	r0	r0	r0	AEN. 7 rw−0	AEN. 6 rw−0	AEN. 5 rw- 0	AEN. 4 rw−0	AEN. 3 rw−0	AEN. 2 rw−0	AEN. 1 rw−0	AEN. 0 rw- 0
输入寄存器 AIN	110H	r0	r0	r0	r0	r0	r0	r0	r0	AIN. 7 r	AIN. 6 r	AIN. 5 r	AIN. 4 r	AIN. 3 r	AIN. 2 r	AIN. 1 r	AIN. 0 r

附录 17　硬件乘法器模块

附录表 17　硬件乘法器寄存器

名　称	地　址	位															
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
扩展寄存器 SUMEXT	013EH	† r	† r	† r	† r	† r	† r	† r	† r	† r	† r	† r	† r	† r	† r	† r	† r
结果高字 RESHI	013CH	2^{15} rw	2^{14} rw	2^{13} rw	2^{12} rw	2^{11} rw	2^{10} rw	2^{9} rw	2^{8} rw	2^{7} rw	2^{6} rw	2^{5} rw	2^{4} rw	2^{3} rw	2^{2} rw	2^{1} rw	2^{0} rw

续附录表 17

名 称	地 址	位															
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
结果低字 RESLO	013AH	2^{15} rw	2^{14} rw	2^{13} rw	2^{12} rw	2^{11} rw	2^{10} rw	2^{9} rw	2^{8} rw	2^{7} rw	2^{6} rw	2^{5} rw	2^{4} rw	2^{3} rw	2^{2} rw	2^{1} rw	2^{0} rw
第二操作数 OP_2	0138H	2^{15} rw	2^{14} rw	2^{13} rw	2^{12} rw	2^{11} rw	2^{10} rw	2^{9} rw	2^{8} rw	2^{7} rw	2^{6} rw	2^{5} rw	2^{4} rw	2^{3} rw	2^{2} rw	2^{1} rw	2^{0} rw
有符号加第一操作数 MACS	0136H	2^{15} rw	2^{14} rw	2^{13} rw	2^{12} rw	2^{11} rw	2^{10} rw	2^{9} rw	2^{8} rw	2^{7} rw	2^{6} rw	2^{5} rw	2^{4} rw	2^{3} rw	2^{2} rw	2^{1} rw	2^{0} rw
无符号加第一操作数 MAC	0134H	2^{15} rw	2^{14} rw	2^{13} rw	2^{12} rw	2^{11} rw	2^{10} rw	2^{9} rw	2^{8} rw	2^{7} rw	2^{6} rw	2^{5} rw	2^{4} rw	2^{3} rw	2^{2} rw	2^{1} rw	2^{0} rw
有符号乘法第一操作数 MPYS	0132H	2^{15} rw	2^{14} rw	2^{13} rw	2^{12} rw	2^{11} rw	2^{10} rw	2^{9} rw	2^{8} rw	2^{7} rw	2^{6} rw	2^{5} rw	2^{4} rw	2^{3} rw	2^{2} rw	2^{1} rw	2^{0} rw
无符号乘法第一操作数 MPY	0130H	2^{15} rw	2^{14} rw	2^{13} rw	2^{12} rw	2^{11} rw	2^{10} rw	2^{9} rw	2^{8} rw	2^{7} rw	2^{6} rw	2^{5} rw	2^{4} rw	2^{3} rw	2^{2} rw	2^{1} rw	2^{0} rw

+结果扩展寄存器保存 16×16 位有符号乘法结果的符号或无符号乘法的溢出或有符号乘加的符号，有符号的上溢或下溢必须用软件来处理。

附录 18　定时器 A 模块

附录表 18.1　定时器 A 的中断向量

名 称	地 址	位															
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TAIV	12EH	0 r0	0 r0	0 r0	0 r0	0 r0	0 r0	0 r0	0 r0	0 r0	0 r0	0 r0	0 r0	TAIV r—(0)	TAIV r—(0)	TAIV r—(0)	0 r0

附录表 18.2　定时器 A 的寄存器

名　称	地　址	位															
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	017EH																
	017CH																
捕获/比较寄存器 CCR4†	017AH	2^{15} rw—(0)	2^{14} rw—(0)	2^{13} rw—(0)	2^{12} rw—(0)	2^{11} rw—(0)	2^{10} rw—(0)	2^{9} rw—(0)	2^{8} rw—(0)	2^{7} rw—(0)	2^{6} rw—(0)	2^{5} rw—(0)	2^{4} rw—(0)	2^{3} rw—(0)	2^{2} rw—(0)	2^{1} rw—(0)	2^{0} rw—(0)
捕获/比较寄存器 CCR3†	0178H	2^{15} rw—(0)	2^{14} rw—(0)	2^{13} rw—(0)	2^{12} rw—(0)	2^{11} rw—(0)	2^{10} rw—(0)	2^{9} rw—(0)	2^{8} rw—(0)	2^{7} rw—(0)	2^{6} rw—(0)	2^{5} rw—(0)	2^{4} rw—(0)	2^{3} rw—(0)	2^{2} rw—(0)	2^{1} rw—(0)	2^{0} rw—(0)
捕获/比较寄存器 CCR2	0176H	2^{15} rw—(0)	2^{14} rw—(0)	2^{13} rw—(0)	2^{12} rw—(0)	2^{11} rw—(0)	2^{10} rw—(0)	2^{9} rw—(0)	2^{8} rw—(0)	2^{7} rw—(0)	2^{6} rw—(0)	2^{5} rw—(0)	2^{4} rw—(0)	2^{3} rw—(0)	2^{2} rw—(0)	2^{1} rw—(0)	2^{0} rw—(0)
捕获/比较寄存器 CCR1	0174H	2^{15} rw—(0)	2^{14} rw—(0)	2^{13} rw—(0)	2^{12} rw—(0)	2^{11} rw—(0)	2^{10} rw—(0)	2^{9} rw—(0)	2^{8} rw—(0)	2^{7} rw—(0)	2^{6} rw—(0)	2^{5} rw—(0)	2^{4} rw—(0)	2^{3} rw—(0)	2^{2} rw—(0)	2^{1} rw—(0)	2^{0} rw—(0)
捕获/比较寄存器 CCR0	0172H	2^{15} rw—(0)	2^{14} rw—(0)	2^{13} rw—(0)	2^{12} rw—(0)	2^{11} rw—(0)	2^{10} rw—(0)	2^{9} rw—(0)	2^{8} rw—(0)	2^{7} rw—(0)	2^{6} rw—(0)	2^{5} rw—(0)	2^{4} rw—(0)	2^{3} rw—(0)	2^{2} rw—(0)	2^{1} rw—(0)	2^{0} rw—(0)
定时器 A 计数器 TAR	0170H	2^{15} rw—(0)	2^{14} rw—(0)	2^{13} rw—(0)	2^{12} rw—(0)	2^{11} rw—(0)	2^{10} rw—(0)	2^{9} rw—(0)	2^{8} rw—(0)	2^{7} rw—(0)	2^{6} rw—(0)	2^{5} rw—(0)	2^{4} rw—(0)	2^{3} rw—(0)	2^{2} rw—(0)	2^{1} rw—(0)	2^{0} rw—(0)
	016EH																
	016CH																
捕获/比较控制寄存器 CCTL4†	016AH	CM41 rw—(0)	CM40 rw—(0)	CCIS41 rw—(0)	CCIS40 rw—(0)	SCS4 rw—(0)	SCCI4 rw—(0)	Unused rw—(0)	CAP4 rw—(0)	OUT MOD42 rw—(0)	OUT MOD41 rw—(0)	OUT MOD40 rw—(0)	CCIE4 r	CCI4 rw—(0)	OUT4 rw—(0)	COV4 rw—(0)	CCIFG4 rw—(0)
捕获/比较控制寄存器 CCTL3†	0168H	CM31 rw—(0)	CM30 rw—(0)	CCIS31 rw—(0)	CCIS30 rw—(0)	SCS3 rw—(0)	SCCI3 rw—(0)	Unused rw—(0)	CAP3 rw—(0)	OUT MOD32 rw—(0)	OUT MOD31 rw—(0)	OUT MOD30 rw—(0)	CCIE3 r	CCI4 rw—(0)	OUT3 rw—(0)	COV3 rw—(0)	CCIFG3 rw—(0)

续附录表 18.2

名 称	地 址	位															
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
捕获/比较控制寄存器 CCTL2	0166H	CM21 rw-(0)	CM20 rw-(0)	CCIS21 rw-(0)	CCIS20 rw-(0)	SCS2 rw-(0)	SCCI2 rw-(0)	Unused rw-(0)	CAP2 rw-(0)	OUTMOD22 rw-(0)	OUTMOD21 rw-(0)	OUTMOD20 rw-(0)	CCIE2 r	CCI2 rw-(0)	OUT2 rw-(0)	COV2 rw-(0)	CCIFG2 rw-(0)
捕获/比较控制寄存器 CCTL1	0164H	CM11 rw-(0)	CM10 rw-(0)	CCIS11 rw-(0)	CCIS10 rw-(0)	SCS1 rw-(0)	SCCI1 rw-(0)	Unused rw-(0)	CAP1 rw-(0)	OUTMOD12 rw-(0)	OUTMOD11 rw-(0)	OUTMOD10 rw-(0)	CCIE1 r	CCI1 rw-(0)	OUT1 rw-(0)	COV1 rw-(0)	CCIFG1 rw-(0)
捕获/比较控制寄存器 CCTL0	0162H	CM20 rw-(0)	CM00 rw-(0)	CCIS01 rw-(0)	CCIS00 rw-(0)	SCS0 rw-(0)	SCCI0 rw-(0)	Unused rw-(0)	CAP0 rw-(0)	OUTMOD02 rw-(0)	OUTMOD01 rw-(0)	OUTMOD00 rw-(0)	CCIE0 r	CCI0 rw-(0)	OUT0 rw-(0)	COV0 rw-(0)	CCIFG0 rw-(0)
定时器 A 控制寄存器 TACTL	0160H	Unused rw-(0)	Unused rw-(0)	Unused rw-(0)	Unused rw-(0)	Unused rw-(0)	Unused rw-(0)	SSEL1 rw-(0)	SSEL0 rw-(0)	ID1 rw-(0)	ID0 rw-(0)	MC1 rw-(0)	MC0 rw-(0)	Unused rw-(0)	CLR rw-(0)	TAIE rw-(0)	TAIFG rw-(0)

† 保留的寄存器。

附录 19 定时器 B 模块

附录表 19.1 定时器 B 的寄存器

名 称	地 址	位															
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
捕获/比较寄存器 CCR6†	019EH	2^{15} rw-(0)	2^{14} rw-(0)	2^{13} rw-(0)	2^{12} rw-(0)	2^{11} rw-(0)	2^{10} rw-(0)	2^{9} rw-(0)	2^{8} rw-(0)	2^{7} rw-(0)	2^{6} rw-(0)	2^{5} rw-(0)	2^{4} rw-(0)	2^{3} rw-(0)	2^{2} rw-(0)	2^{1} rw-(0)	2^{0} rw-(0)
捕获/比较寄存器 CCR5†	019CH	2^{15} rw-(0)	2^{14} rw-(0)	2^{13} rw-(0)	2^{12} rw-(0)	2^{11} rw-(0)	2^{10} rw-(0)	2^{9} rw-(0)	2^{8} rw-(0)	2^{7} rw-(0)	2^{6} rw-(0)	2^{5} rw-(0)	2^{4} rw-(0)	2^{3} rw-(0)	2^{2} rw-(0)	2^{1} rw-(0)	2^{0} rw-(0)
捕获/比较寄存器 CCR4†	019AH	2^{15} rw-(0)	2^{14} rw-(0)	2^{13} rw-(0)	2^{12} rw-(0)	2^{11} rw-(0)	2^{10} rw-(0)	2^{9} rw-(0)	2^{8} rw-(0)	2^{7} rw-(0)	2^{6} rw-(0)	2^{5} rw-(0)	2^{4} rw-(0)	2^{3} rw-(0)	2^{2} rw-(0)	2^{1} rw-(0)	2^{0} rw-(0)

附录 21 MSP430F4XX 系列液晶驱动模块

附录表 21 液晶驱动模块寄存器

名 称	地 址	位							
		7	6	5	1	3	2	1	0
LCDM 19	0A4F	S39C3 rw	S39C2 rw	S39C1 rw	S39C0 rw	S38C3 rw	S38C2 rw	S38C1 rw	S38C0 rw
LCDM 18	0A3H	S37C3 rw	S37C2 rw	S37C1 rw	S37C0 rw	S36C3 rw	S36C2 rw	S36C1 rw	S36C0 rw
LCDM 18	0A2H	S35C3 rw	S35C2 rw	S35C1 rw	S35C0 rw	S34C3 rw	S34C2 rw	S34C1 rw	S31C0 rw
LCDM 17	0A1H	S33C3 rw	S33C2 rw	S33C1 rw	S33C0 rw	S32C3 rw	S32C2 rw	S32C1 rw	S32C0 rw
LCDM 16	0A0H	S31C3 rw	S31C2 rw	S31C1 rw	S31C0 rw	S30C3 rw	S30C2 rw	S30C1 rw	S30C0 rw
LCDM 15	09FH	S29C3 rw	S29C2 rw	S29C1 rw	S29C0 rw	S28C3 rw	S28C2 rw	S28C1 rw	S28C0 rw
LCDM 14	009EH	S27C3 rw	S27C2 rw	S27C1 rw	S27C0 rw	S26C3 rw	S26C2 rw	S26C1 rw	S26C0 rw
LCDM 13	09DH	S25C3 rw	S25C2 rw	S25C1 rw	S25C0 rw	S24C3 rw	S24C2 rw	S24C1 rw	S24C0 rw
LCDM 12	09CH	S23C3 rw	S23C2 rw	S23C1 rw	S23C0 rw	S22C3 rw	S22C2 rw	S22C1 rw	S22C0 rw
LCDM 11	09BH	S21C3 rw	S21C2 rw	S21C1 rw	S21C0 rw	S20C3 rw	S20C2 rw	S20C1 rw	S20C0 rw
LCDM 10	09AH	S19C3 rw	S19C2 rw	S19C1 lrw	S19C0 rw	S18C3 rw	S18C2 rw	S18C1 rw	S18C0 rw
LCDM 9	099H	S17C3 rw	S17C2 rw	S17C1 rw	S17C0 rw	S16C3 rw	S16C2 rw	S16C1 rw	S16C0 rw
LCDM 8	098H	S15C3 rw	S15C2 rw	S15C1 rw	S15C0 rw	S14C3 rw	S14C2 rw	S14C1 rw	S14C0 rw
LCDM 7	097H	S13C3 rw	S13C2 rw	S13C1 rw	S13C0 rw	S12C3 rw	S12C2 rw	S12C1 rw	S12C0 rw
LCDM 6	096H	S11C3 rw	S11C2 rw	S11C1 rw	S11C0 rw	S10C3 rw	S10C2 rw	S10C1 rw	S10C0 rw
LCDM 5	095H	S9C3 rw	S9C2 rw	S9C1 rw	S9C0 rw	S8C3 rw	S8C2 rw	S8C1 rw	S8C0 rw
LCDM 4	094H	S7C3 rw	S7C2 rw	S7C1 rw	S7C0 rw	S6C3 rw	S6C2 rw	S6C1 rw	S6C0 rw
LCDM 3	093H	S5C3 rw	S5C2 rw	S5C1 rw	S5C0 rw	S4C3 rw	S4C2 rw	S4C1 rw	S4C0 rw
LCDM 2	092H	S3C3 rw	S3C2 rw	S3C1 rw	S3C0 rw	S2C3 rw	S2C2 rw	S2C1 rw	S2C0 rw
LCDM 1	091H	S1C3 rw	S1C2 rw	S1C1 rw	S1C0 rw	S0C3 rw	S0C2 rw	S0C1 rw	S0C0 rw
LCD CTL	0090H	LCDM7 rw－0	LCDM6 rw－0	LCDM5 rw－0	LCDM4 rw－0	LCDM3 rw－0	LCDM2 rw－0	LCDM1 rw－0	LCDM0 rw－0

Powered by xiaoguo's publishing studio
QQ:8204136