MCU

MSP430 系列

FLASH 型超低功耗16位单片机

胡大可 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 定价：29.50元

ISBN 7-81077-096-9

9 787810 770965 >

ISBN 7-81077-096-9/TP · 051

定价：30.00元

序　言

1999 年，温州利尔达电子器材公司开始和美国德州仪器公司合作开展 MSP430 单片机在国内的推广和销售。经过我们的努力，今天 MSP430 的独特性能和技术优点已经被国内单片机产品设计开发单位、电子工程技术人员和广大的嵌入式系统爱好者所接受，并在各个领域得到了广泛的应用。

目前国内使用较多的仍然是 4 位和 8 位单片机，而美国德州仪器公司推出的 16 位单片机 MSP430 系列具有更卓越的品质。它具有处理能力强、运行速度快、功耗低等优点，因此性能价格比高。它在欧洲已得到了非常广泛的应用。

MSP430 系列包容了许多先进的技术：

1. JTAG 技术：MSP430 单片机内部预设了 JTAG 模块，它使得每一个单片机芯片都具有完整的在线调试功能，而不必使用复杂的仿真调试工具。

2. FLASH 在线编程技术：现在流行的 FLASH 单片机中，很少能够实现在线程序编程功能，大部分需要外部编程器进行烧写。而 MSP430 系列的 FLASH 型单片机除了可以采用外部编程器进行烧写外，用户可以利用自己的程序修改 FLASH 内容，且不需要外加编程电压。这给系统设计带来以下可能：可以利用片内 FLASH 保存一些运行数据，实现掉电保护；可以利用片内的 FLASH 方便地实现软件升级，可以修改整个程序，也可以只修改局部程序，以达到系统升级的目的。

3. BOOTSTRAP 技术：MSP430 系统的 FALSH 型芯片具有片内的 BOOT ROOM，可以实现程序代码的下载和上载。利用它，只需几根线就可以修改内部的程序，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，口令字达到 32 个字节的长度。

4. MSP430 系列的 FLASH 型芯片集中了许多设计人员十分关注的技术，如：

- 充分运用各种超低耗设计手段，使芯片的电流极小，在 LPM4 时可达 0.1 μA；
- 超低功耗的数控振荡器技术，可以实现频率调节和无晶振运行；
- 采用精简指令集，只有 27 条核心指令，指令周期可达 125 ns；
- 程序代码空间可达 60 KB、数据存储空间可达 2 KB，I/O 引脚可达 48 线；
- 片内集成 12 位 A/D、16 位定时器、模拟比较器、串行接口、硬件乘法器等模块；
- 芯片的工作电压范围宽，工作温度范围符合工业级要求。

本书是继《MSP430 系列超低功耗 16 位单片机原理与应用》一书之后的又一本 MSP430 系列单片机技术参考书。它全面介绍 MSP430 F11x、F13x、F14x 等系列 FLASH 型单片机的基本结构以及各功能模块的原理和应用，内容详细，是在嵌入式系统设计开发过程中必不可少的参考资料。

本书的出版发行得到了北京航空航天大学出版社以及马广云博士的大力支持，在此表示感谢。

温州利尔达电子器材公司

2001 年 9 月

温州利尔达电子器材公司

网址:http://www.lierda.com

杭州公司

地址:杭州市教工路2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

香港公司

地址:香港中环皇后大道中286号金祥大厦九栋B室

电话:00852-25950393

传真:00852-25588160

E-mail:funart@netvigator.com

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:lierda@public3.bta.net.cn

上海办事处

地址:上海市漕溪路251号望族城4号楼1603室

电话:021-64851586

传真:021-64830671

邮编:200233

E-mail:phg@lierda.com

深圳办事处

地址:深圳华发南路中电大院怡静苑801室

电话:0755-3651255

传真:0755-3650657

邮编:518031

E-mail:ygx@lierda.com

西安分销商

地址:西安市劳动南路94号电子市场D档13-15号

电话:029-4230278

传真:029-4230278

邮编:710068

E-mail:hlx@lierda.com

前　言

单片机技术，或者称为微控制器技术，已经在人类生活的方方面面都得到应用。它特别适合用在各种专用的、小型的、省电的、可移动的设备之中，构成所谓的嵌入式应用系统。多年来的快速发展，使得它的家族越来越庞大，系列品种越来越多，而且技术上往往各具特色。

MSP430 系列是一个特别强调超低功耗性能的单片机品种。它适合应用在各种要求极低功率消耗的场合，具有一定的技术特点。在这个系列中有多个型号，它们是由一些基本功能模块按不同的应用目标组合而成。北京航空航天大学出版社出版的《MSP430 系列超低功耗 16 位单片机原理与应用》一书已对 MSP430 系列中的 31x、32x、33x 单片机作了详细的介绍。本书集中介绍这一系列中的 FLASH 型单片机，介绍它们的原理、结构、指令、运行模式、功能模块及应用开发方法。本书的内容适合于 MSP430 系列中的各个 FLASH 型单片机。

MSP430 系列的 FLASH 型芯片又可分为几个分支，如：11x、11x1、13x、14x 等，而且仍处在不断发展的过程中。从应用角度看，它们都具有开发设备简便、可现场编程等特点。它们的许多性能指标比此前的 MSP430 其他各个型号有较大的改进，内部集成了功能更为丰富的各种资源，功能上有了明显的增强。

阅读本书的读者应具备数字电路知识，最好有其他单片机品种的学习或开发经历。

本书成书过程中得到研究生季燕飞、潘卫江等人的协助。

由于作者的学识水平有限，书中难免有错误和不妥之处，恳请读者提出批评指正。

作　者

2001 年 5 月

内 容 简 介

MSP430 系列单片机在超低功耗和功能集成等方面有明显的特点。该系列单片机自问世以来，颇受用户关注。在 2000 年该系列单片机又出现了几个 FLASH 型的成员，它们除了仍然具备适合应用在自动信号采集系统、电池供电便携式装置、超长时间连续工作的设备等领域的特点外，更具有开发方便、可以现场编程等优点。这些技术特点正是应用工程师特别感兴趣的。本书对该系列单片机的 FLASH 型成员的原理、结构、内部各功能模块及开发方法与工具作详细介绍。

图书在版编目(CIP)数据

MSP430 系列 FLASH 型超低功耗 16 位单片机/胡大可编著.
北京：北京航空航天大学出版社，2001.11
ISBN 7-81077-096-9

Ⅰ.M… Ⅱ.胡… Ⅲ.单片微型计算机，MSP430 系列 Ⅳ.TP368.1

中国版本图书馆 CIP 数据核字(2001)第 051901 号

MSP430 系列
FLASH 型超低功耗 16 位单片机
胡大可 编著
责任编辑 刘晓明
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制 各地书店经销
*
开本：787×1092 1/16 印张：20 字数：512 千字
2001 年 11 月第 1 版 2002 年 3 月第 2 次印刷 印数：5 001～10 000 册
ISBN 7-81077-096-9/TP·051 定价：30.00 元

MSP430 系列
FLASH 型超低功耗 16 位单片机

胡大可　编著

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

目　录

第1章 引 论

1.1 MSP430 系列单片机

TI 公司的 MSP430 系列单片机(或称为微控制器),是一种具有超低功耗特性的功能强大的单片机。它有几个子系列,其中的 MSP430x31x、32x、33x 等系列产品已在北京航空航天大学出版社的《MSP430 系列超低功耗 16 位单片机原理与应用》一书中作了介绍。

MSP430 系列在 2000 年又增加了一个 MSP430F1x 子系列。这一系列具有 FLASH 存储器,在系统设计、开发调试及实际应用上都表现出较明显的优点。

MSP430F1x 系列具有以下一些共同的特点:

超低功耗

MSP430F1x 运行在 1 MHz 时钟条件下时,工作电流视工作模式不同为 0.1～400 μA ,工作电压为 1.8～3.6 V。

强大的处理能力

MSP430F1x 具有丰富的寻址方式(源操作数 7 种,目的操作数 4 种),但只需简洁的 27 条指令;片内寄存器数量多,存储器可实现多种运算;有高效的查表处理方法。这些特点保证了可以编制出高效的程序。

MSP430F1x 的中断源较多,并且可以任意嵌套,使用时灵活方便。当系统处于省电的备用状态时,用中断请求将它唤醒只需 6 μs。

丰富的片上外围模块

MSP430F1x 中的各个成员集成了较多的片上外围资源。有几个型号的片上外围模块功能相当丰富,如 MSP430F149 就包含:12 位 A/D,精密模拟比较器,硬件乘法器,2 组频率可达 8 MHz的时钟模块,2 个带有大量捕获/比较寄存器的 16 位定时器,看门狗,2 个可实现异步、同步及多址访问的串行通信接口,数十个可实现方向设置及中断功能的并行输入、输出端口等。

方便高效的开发方式

MSP430F1x 具有 FLASH 存储器,这一特点使得它的开发工具相当简便。利用单片机本身具有的 JTAG 接口或片内 BOOT ROM,可以在一台 PC 及一个结构小巧的 JTAG 控制器的帮助下实现程序的下载,完成程序调试。

多种存储器形式

MSP430F1x 的各个型号大多有性能相同而存储器不同的 ROM 型、OTP 型,以适应产品在设计、开发、生产的各个不同阶段的需要。

适应工业级运行环境

MSP430F1x 的运行环境温度范围为－40～＋85 ℃,所设计的产品适合运行于工业环境下。具有 FLASH 存储器的 MSP430F1x 系列目前有 MSP430F11x、MSP430F11x1、MSP430F13x 和 MSP430F14x 等型号,所集成的存储器容量及外围模块各不相同。在后面的

- 带有 3 个捕捉/比较寄存器的 16 位定时器 Timer_A3。
- 带有 3 个捕捉/比较寄存器的 16 位定时器 Timer_B3。
- 2 个具有中断功能的 8 位并行端口:P1 与 P2。
- 4 个 8 位并行端口:P3、P4、P5 与 P6。
- 模拟比较器 Comparator_A。
- 12 位 A/D 转换器 ADC12。
- 1 个串行通信接口 USART0。

MSP430F13x 系列有以下型号:

- MSP430F133:8 KB +256 字节 FLASH,256 字节 RAM。
- MSP430F135:16 KB +256 字节 FLASH,512 字节 RAM。

1.5 MSP430F14x 系列

MSP430F14x 由以下部分组成:

- 基础时钟模块,包括 1 个数控振荡器(DCO)和 2 个晶体振荡器。
- 看门狗定时器 Watchdog Timer,可用作通用定时器。
- 带有 3 个捕捉/比较寄存器的 16 位定时器 Timer_A3。
- 带有 7 个捕捉/比较寄存器的 16 位定时器 Timer_B7。
- 2 个具有中断功能的 8 位并行端口:P1 与 P2。
- 4 个 8 位并行端口:P3、P4、P5 与 P6。
- 模拟比较器 Comparator_A。
- 12 位 A/D 转换器 ADC12。
- 2 个串行通信接口:USART0 与 USART1。
- 1 个硬件乘法器。

MSP430F14x 系列有以下型号:

- MSP430F147:32 KB +256 字节 FLASH,1 KB RAM。
- MSP430F148:48 KB +256 字节 FLASH,2 KB RAM。
- MSP430F149:60 KB +256 字节 FLASH,2 KB RAM。

表 1.1 是 MSP430 系列的 FLASH 各型号单片机的性能结构一览表。

表 1.1 FLASH 型单片机性能一览表

	F110	F112	F1101	F1121	F133	F135	F147	F148	F149
时钟频率	32 768 Hz～8 MHz								
工作温度	−40～+85 ℃								
程序 FLASH/KB	1	4	1	4	8	16	32	48	60
信息 FLASH/字节	128	256	128	256	256	256	256	256	256
RAM	128 字节	256 字节	128 字节	256 字节	256 字节	512 字节	1 KB	2 KB	2 KB
WDT	有								
P1	有								
P2	有								
P3	无				有				
P4	无				有				
P5	无				有				
P6	无				有				
Timer_A	有，3 个捕捉比较寄存器(3xCCR)								
Timer_B	无				3xCCR		7xCCR		
比较器 A	无		有						
USART0	无				有				
USRAT1	无						有		
ADC12	无				有				
硬件乘法器	无						有		

第 2 章　结构概述

2.1　引　言

MSP430 系列采用存储器—存储器结构，即用一个公共的空间对全部功能模块寻址，同时用精简指令组对全部功能模块进行操作。

图 2.1～图 2.4 是 MSP430F1x 系列不同型号的系统结构。

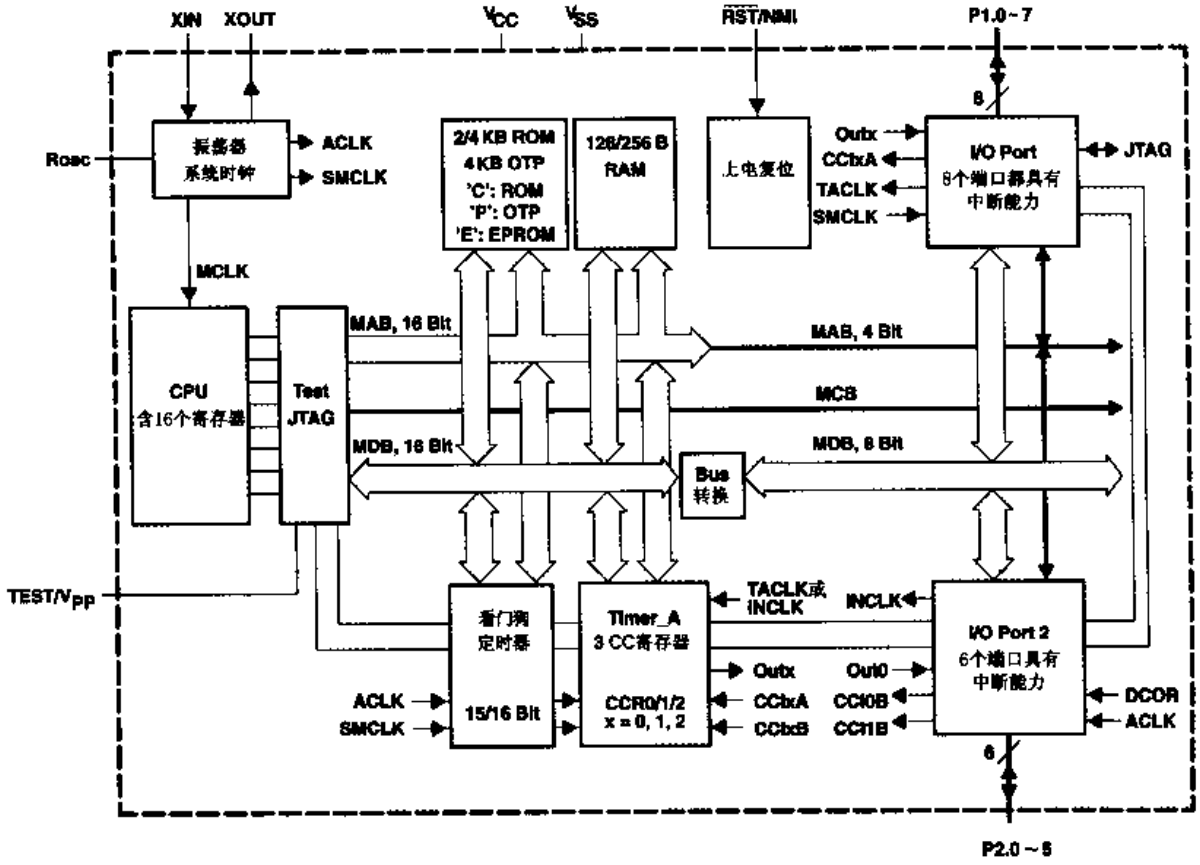
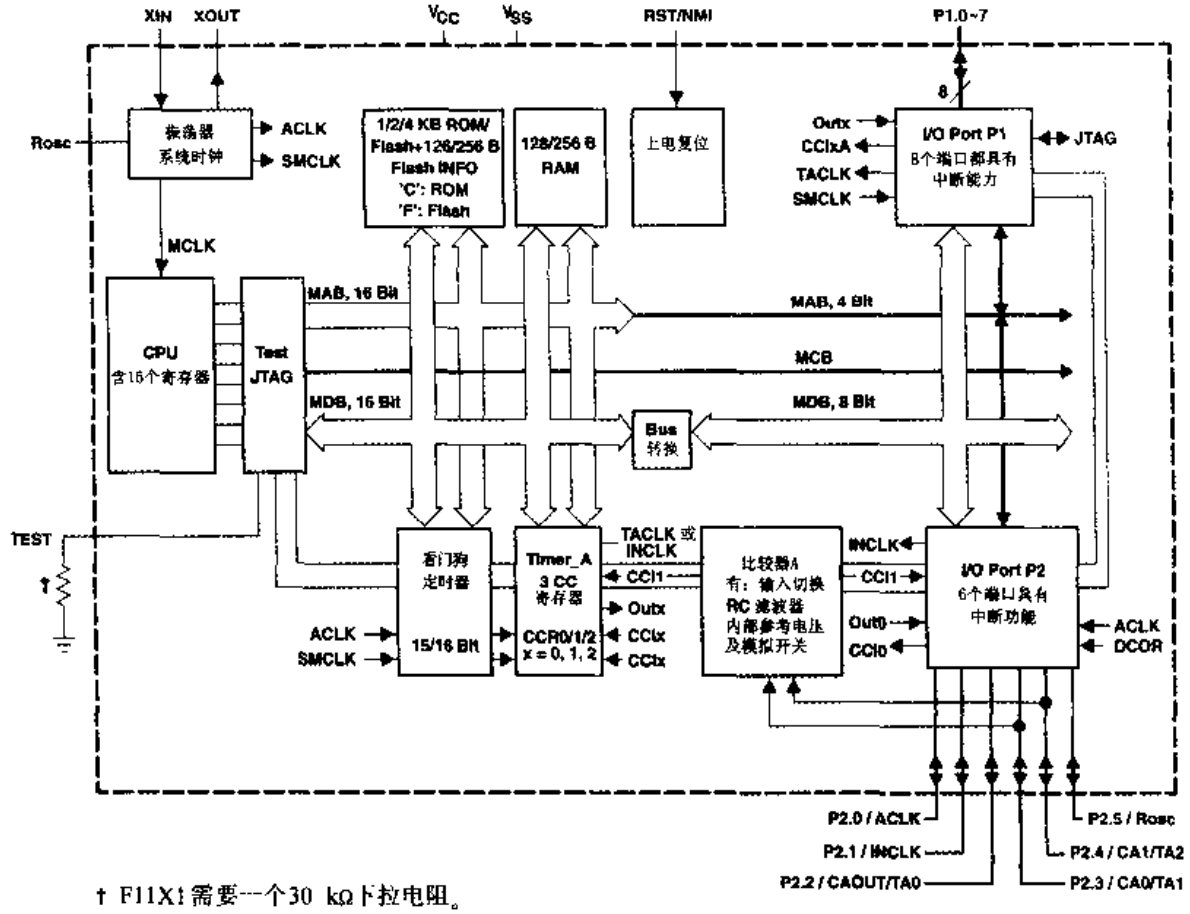


图 2.1　MSP430F11x 系统结构



† F11X1需要一个30 kΩ下拉电阻。

图 2.2 MSP430F11x1 系统结构

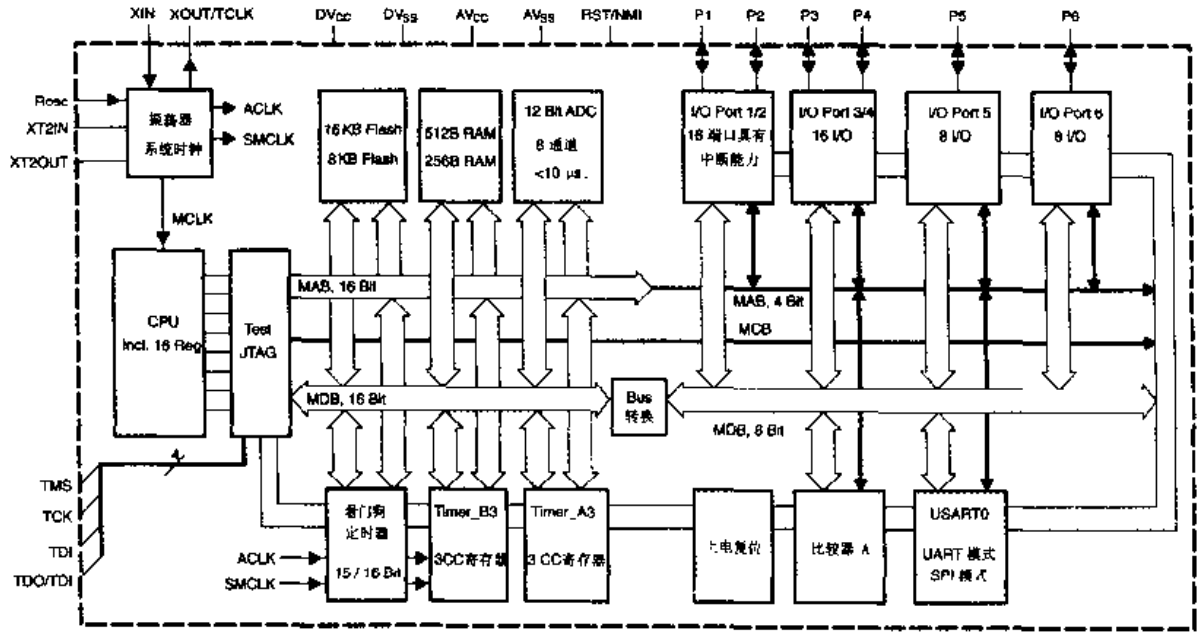


图 2.3 MSP430F13x 系统结构

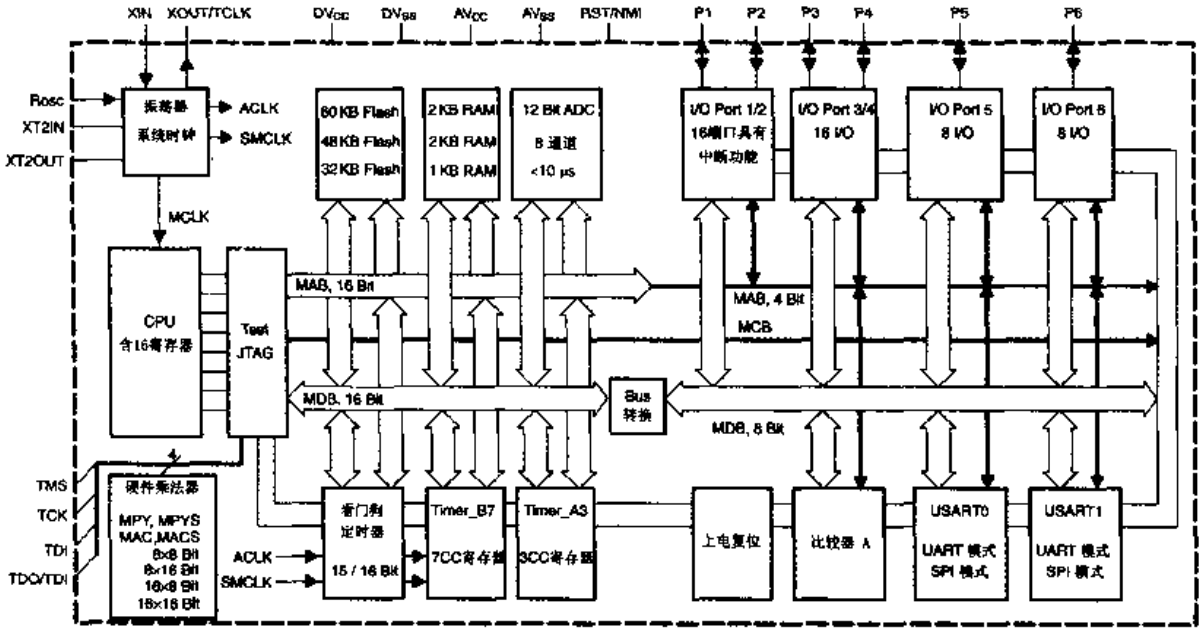


图 2.4　MSP430F14x 系统结构

2.2　CPU

MSP430 的 CPU 运行正交的精简指令集，由 16 位 ALU、指令控制逻辑和 16 个寄存器组成。

寄存器中有 4 个具有特殊用途，即：程序计数器 R0/PC、堆栈指针 R1/SP、状态寄存器和常数发生器 R2/SR/CG1、R3/CG2。除了 CG1 和 CG2，所有寄存器都可作为通用寄存器，用所有指令操作。

常数发生器只用于指令执行时提供常数，但不能存储数据。对 CG1、CG2 访问时的寻址模式可以区分所获得的常数数值。

PC、SP 和 SR 配合精简指令所实现的控制，可以使应用系统的程序设计实现复杂的寻址模式和软件算法。

2.3　程序存储器

对程序存储器的访问，对于程序代码总是以字形式取得，而对于数据可以用字或字节指令进行访问。每次访问需要 16 位数据总线（MDB）和访问当前存储器模块所需的地址总线（MAB）。存储器模块由内部模块允许信号自动选中，这样可以减少总的电流消耗。

对于 MSP430F 系列，程序存储器是 FLASH 的。在程序设计中，可以将数据安排在程序存储器中，它们可以用字或字节指令方式访问，因此可以实现查表处理等应用。

64 KB 空间顶部的 16 字（0FFFFh～0FFE0h）保留用作复位及中断的向量地址。

2.4 数据存储器

数据存储器(RAM)与程序存储器相同,经地址总线(MAB)和数据总线(MDB)与 CPU 相连。RAM 内的数据可以以字或字节宽度访问。

由于 RAM 与程序存储器是经过相同的地址总线和数据总线与 CPU 相连,因此程序代码可以装入 RAM,也可以在 RAM 内运行。这对于程序的调试提供了很大的方便。

所有指令都有字节操作或字操作形式。但是,对堆栈和 PC 的操作是按字宽度进行的,寻址时必须对准偶地址。

2.5 运行控制

MSP430 的运行主要受控于存储在特殊寄存器(SFR)中的信息。SFR 中的各位信息可以允许中断以支持取决于中断标志状态的软件;可以定义外围模块的工作模式,被禁止的外围模块将暂停运行以减少电流的消耗,而所有存储在模块寄存器中的数据仍然保留。

2.6 外围模块

外围模块经 MAB、MDB 和中断服务及请求线与 CPU 相连。对于大多数外围模块,MAB 通常是 5 位,MDB 是 8 位或 16 位。大多数外围模块工作在字节形式。8 位数据总线的模块经总线转换电路连到 16 位的 CPU。这些模块的数据交换毫无例外地必须用字节指令操作,SFR 的处理也全部为字节指令;部分工作在字形式的外围模块必须用字指令操作。

2.7 振荡器与时钟发生器

振荡器 LFXT1 是专门为通用的低功耗 32 768 Hz 时钟晶振设计的,但是也可以用一个高速的晶振工作。当用 32 768 Hz 晶振时,除了晶体外接外,所有的模拟元件都集成在片内。如果采用高速晶振,仍需要外接负载电容。对于 MSP430F13x 和 MSP430F14x,片内还有一个可实现高速晶振的 XT2 振荡器。

除了晶体振荡器外,MSP430 系列 FLASH 型芯片都有一个数控 RC 振荡器(DCO)。DCO 与其他普通的 RC 振荡器不同,它是可以实现数字控制及频率调节的。

外围模块及 CPU 的时钟源选择非常灵活。绝大多数外围模块设计成可以用 32 768 Hz、高速晶振或 DCO 来工作。CPU 可以用 DCO 工作,也可以用晶振时钟信号运行。

注意:软件产生 PUC

如有需要,用软件对看门狗定时器控制寄存器简单写入错误安全键值可产生 PUC。

注意:POR 信号与 PUC 信号

POR 信号有效必然产生一个系统复位中断,PUC 信号的产生并不一定产生系统复位中断。PUC 信号有效时,系统可能发生复位中断,也可能不发生复位中断而产生一个较低优先级的中断,这取决于引起 PUC 的事件。每一型号的技术手册中都详细说明产生中断的原因,这些产生中断的因素是恰当处理全部中断请求时所要考虑的。

若 V_{cc} 的加载有如图 3.2 的快速上升时间,则 POR 信号会提供足够长的延时时间,以保证各部分电路在上电后的正确初始化。当 RST/NMI 设置成复位模式时,在 RST/NMI 引脚上出现低电平信号。如果 V_{cc} 的加载上升时间较慢,如图 3.3 所示,则 POR 检测电路保持 POR 信号有效,直到 V_{cc} 上升超过 POR 电平。这样,同样也保证了正确的初始化。

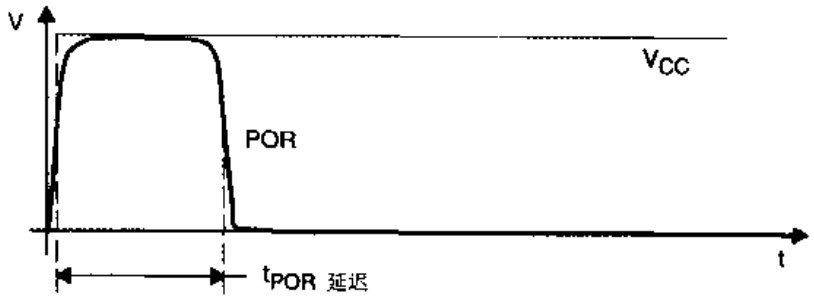


图 3.2 V_{cc} 快速上升时的上电复位

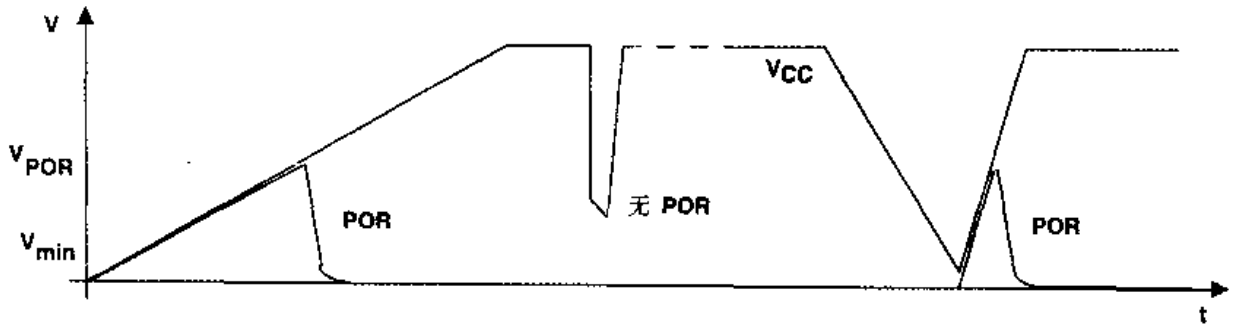


图 3.3 V_{cc} 慢速上升时的上电复位

如果芯片的上电是周期性的,则掉电时 V_{cc} 必须降低到 V_{min},以保证 V_{cc} 再次加载时发生新的 POR 信号。如果在一个周期中 V_{cc} 没有下降到低于 V_{min},或者因为发生干扰,那么 POR 信号就不会发生,这样上电后的初始化状态将是不正确的。

3.1.2 系统复位后的设备初始化

当因为 POR 或 PUC 信号引起设备复位后,系统的初始状态如下:

- I/O 引脚切换成输入模式。
- I/O 标识位清除,见有关 I/O 章节的说明。

- 其他外围模块及寄存器实现初始化，见相应各章的说明。
- 状态寄存器复位。
- PC 装入 0FFFEh 处的地址值，CPU 从这一地址开始执行。

注意：I/O 引脚初始化

I/O 引脚及标志仅在上电时发生初始化。在 MSP430 上电后或运行中，在 RST/NMI 引脚上(在复位模式下)加复位信号对各 I/O 引脚没有影响。

在发生系统复位后，用户程序可以通过各种标志来确定复位源，以作出相应的处理。

在本书中，对各状态位的描述采用统一的符号，如：rw-(0)、rw-0 等。r 表示可读；w 表示可写；加括号的数值表示只受 POR 信号影响的复位状态，它们将不受 PUC 信号的影响；不加括号的数值表示受 PUC 或者受到 POR/PUC 信号组合影响的复位状态。以下是描述的实例：

r:　　只读，无初始状态。
w:　　只写，无初始状态。
rw-(0):　读/写，由 POR 复位。
rw-0:　读/写，由 POR 或 PUC 复位。
r-1:　只读，由 POR 或 PUC 置位。
r0:　读出为“0”。
(w):　无寄存功能，写“1”将产生一个脉冲。读出总是为“0”。
h0:　由硬件复位。

3.2　中断系统结构

MSP430 有 4 类中断：

- 系统复位。
- 可屏蔽中断(maskable)。
- 非屏蔽中断(non-maskable)。
- (非)屏蔽中断((non)-maskable)。

系统复位已在 3.1 节介绍过。

引起可屏蔽中断的来源有：

- 当看门狗定时器选定时器模式时，发生定时器溢出。
- 其他具有中断能力的模块发生中断事件。

非屏蔽中断是无法屏蔽的。既无中断允许控制位，通用中断允许位(GIE)也不会对它产生影响。

(非)屏蔽中断不能用通用中断允许位(GIE)屏蔽，但是可以用各自的中断允许位控制。当一个(非)屏蔽中断请求被接受时，相应的中断允许位就自动复位，因此也禁止了新的中断请求。RETI(从中断服务程序返回)指令不影响相应的(非)屏蔽中断的允许位，因此必须用软件在中断服务程序中的 RETI 指令前将中断允许位置位，以保证在中断服务结束后中断能再次被允许。

(非)屏蔽中断可以由下列条件产生：

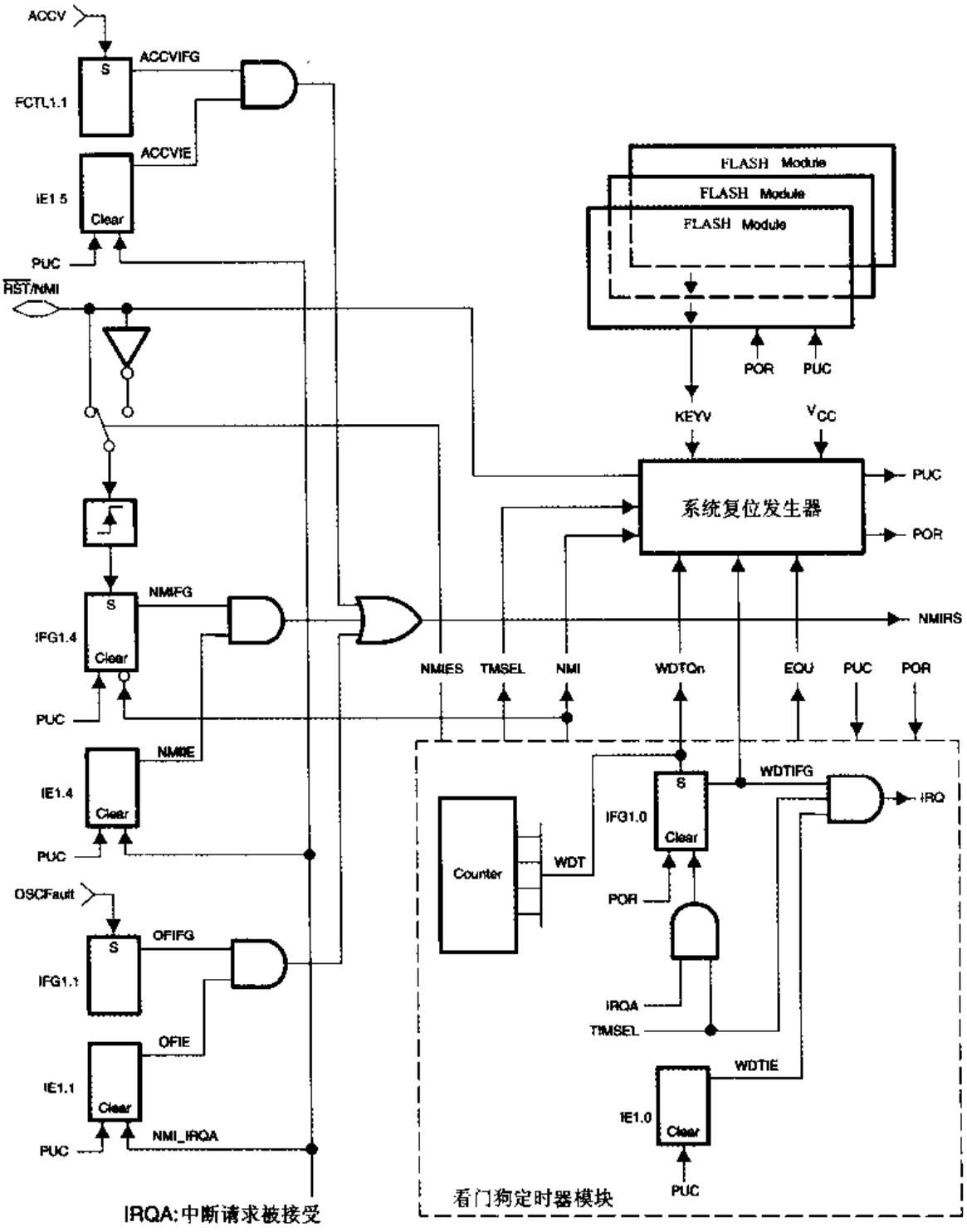


图 3.5　非屏蔽中断源

NMIES=0：上升沿触发非屏蔽中断；

NMIES=1：下降沿触发非屏蔽中断。

b0～4.7：见有关看门狗定时器的章节。

3.3.1 中断操作——复位/NMI

如果选择复位功能，则当 RST/NMI 引脚保持为低时，CPU 一直保持复位状态。当引脚电平变高时，CPU 从 0FFFEh(复位向量)所含的地址开始执行程序。

如果选择 NMI 功能，则一个按 NMIES 位规定的触发沿会产生中断，程序从 0FFFCh 所含的地址重新开始执行，并且 SFR 中的 RST/NMI 标志(IFG1.4)也被置位。

注意：NMI

当选择 NMI 模式时，产生 NMI 事件的信号不可以将 RST/NMI 引脚保持在低状态，除非打算进入复位。当在此引脚上发生 NMI 事件时，PUC 信号成为有效，这样会将 WDTCTL 中的各位复位。这将使 RST/NMI 引脚进入复位模式。如果产生 NMI 的信号仍保持为低，则会使处理机进入复位状态。

当选择 NMI 模式，并改变 NMI 沿选择位时，根据 RST/NMI 引脚上的实际电平可能发生一次 NMI。但是，如果在 NMI 模式选择前改变 NMI 的边沿选择位，就不会发生 NMI。

NMI 中断可以用 NMIIE 位屏蔽。

3.3.2 中断操作——振荡器失效控制

振荡器失效信号对可能由晶振引起的错误发出警告。

当 LFXT1 振荡器设置成高速模式但未起振，振荡器工作停止，或者将振荡器关闭时，都会触发振荡器失效信号。对于有 XT2 振荡器的型号，也会因同样的条件引起振荡器失效。注意 PUC 信号会触发一次振荡器失效，因为 PUC 信号会将 LFXT1 切换到低速模式，即关闭了高速模式。同样，PUC 信号也会将 XT2 振荡器关闭。

用 SFR 中的 OFIE 位允许振荡器失效信号产生 NMI。可以在 NMI 的中断服务程序中通过对 SFR 中的 OFIFG 位的测试来判断 NMI 是否由时钟失效引起。

3.4 中断处理

MSP430 的可编程中断结构可以组成灵活的片上和外部中断体系，以适应实时中断驱动系统的需要。中断可由处理机的运行状态来启动，如看门狗溢出、外围模块或外部发生的事件等。每个中断源可以用各自的中断允许位单独关闭，而状态寄存器中的通用中断允许位 GIE 可以禁止全部可屏蔽中断。

当中断请求发生并且相应的中断允许位和通用中断允许位(GIE)置位时，中断服务程序按以下顺序激活：

(1) CPU 处于活动状态：完成当前执行指令。

(2) CPU 处于停止状态：低功耗模式终止。

(3) 指向下一条指令的 PC 值压入堆栈。

(4) SR 压入堆栈。

(5) 如果在执行上条指令时已发生多个中断请求等待服务，则选择最高优先级者。

(6) 单中断源标志的中断请求位自动复位，多中断源标志仍保持置位以等待软件服务。

(7) 通用中断允许位 GIE 复位；CPUOff 位、OSCOff 位和 SCG1 位复位；状态位 V、N、Z 和 C 复位；SCG0 位保持不变。

(8) 相应的中断向量值装入 PC，程序从此地址继续执行中断处理。

图 3.6 是中断请求接受时的堆栈变化。

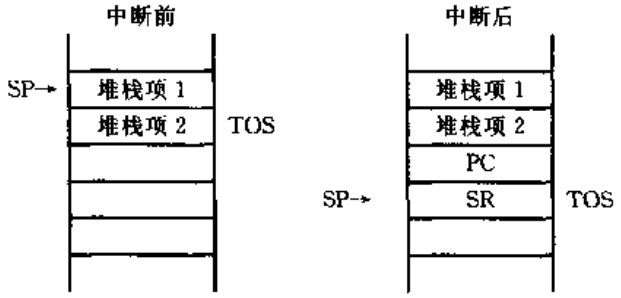


图 3.6　中断响应过程

中断响应从接受中断请求开始到执行相应的中断服务程序的首条指令，持续 6 个时钟。

中断处理程序结束的指令是 RETI。它执行以下内容：

(1) SR 从堆栈中推出。GIE、CPUOff 等标志在中断服务前设置的值变为有效，不受在中断服务期间所设置值的影响。

(2) PC 从堆栈推出，开始从程序被中断的地方继续执行。

图 3.7 是中断返回的堆栈变化。

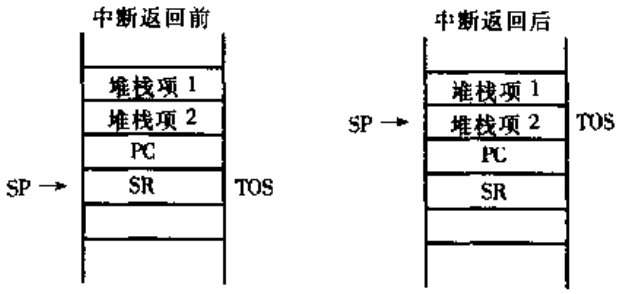


图 3.7　中断结束过程

RETI 指令执行需要 5 个周期。如果在中断处理程序内将 GIE 置位，将允许中断请求嵌套。GIE 位在状态寄存器 SR/R2 中，如下所示：

15 … 9	8	7							0
保留	V	SCG1	SCG0	OSCOff	CPUOff	GIE	N	Z	C
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

除了 GIE 位，中断源可以被控制位单独或成组地允许或禁止。中断允许标志集中于两个地址的 SFR 中。中断请求对程序流的控制可以运用中断允许和屏蔽来方便地调整。硬件只对被允许的中断源中的最高优先级者服务。

3.4.1　SFR 中的中断控制位

大多数中断控制位、中断标志位和中断允许位集中在少数几个 SFR 中。这些 SFR 以字节形式位于低地址区，见表 3.1。SFR 只能以字节指令访问。

表 3.1 SFR 中的中断控制位

地 址	7 ... 0
000Fh	未定义或未实现
000Eh	未定义或未实现
000Dh	未定义或未实现
000Ch	未定义或未实现
000Bh	未定义或未实现
000Ah	未定义或未实现
0009h	未定义或未实现
0008h	未定义或未实现
0007h	未定义或未实现
0006h	未定义或未实现
0005h	模块允许 2(ME2. x)
0004h	模块允许 1(ME1. x)
0003h	中断标志 2(IFG2. x)
0002h	中断标志 1(IFG1. x)
0001h	中断允许 2(IE2. x)
0000h	中断允许 1(IE1. x)

MSP430 的 SFR 影响各模块的功能。各模块的中断源可以分别允许或禁止，见表 3.2、表 3.3及表 3.4。注意各型号之间有差异。

表 3.2 中断允许寄存器 1、2

位	缩 写	PUC 后初始状态	说 明
IE1.0	WDTIE	复位	WDT 允许，选看门狗模式时无效
IE1.1	OFIE	复位	振荡器故障中断允许
IE1.2			未实现
IE1.3			未实现
IE1.4	NMIIE	复位	NMI 中断允许
IE1.5	ACCVIE	复位	FLASH 存储器非法访问中断允许
IE1.6	URXIE0	复位	USART0 接受中断允许(F13x、F14x)
IE1.7	UTXIE0	复位	USART0 发送中断允许(F13x、F14x)
IE2.0			未实现
IE2.1			未实现
IE2.2			未实现
IE2.3			未实现
IE2.4	URXIE1	复位	USART1 接受中断允许(F14x)
IE2.5	UTXIE1	复位	USART1 发送中断允许(F14x)
IE2.6			未实现
IE2.7			未实现

表 3.3　中断标志寄存器 1、2

位	缩　写	初始状态	说　明
IFG1.0	WDTIFG	置位	看门狗模式时溢出或密钥不符时置位
		复位	V_{cc} 上电或 RST/NMI 引脚有复位条件
IFG1.1	OFIFG	置位	振荡器失效时置位
IFG1.2			未实现
IFG1.3			未实现
IFG1.4	NMIIFG	复位	RST/NMI 引脚信号
IFG1.5			未定义
IFG1.6	URXIFG0	复位	USART0 接收标志(F13x、F14x)
IFG1.7	UTXIFG0	置位	USART0 发送标志(F13x、F14x)
IFG2.0			未实现
IFG2.1			未实现
IFG2.2			未实现
IFG2.3			未实现
IFG2.4	URXIFG1	复位	USART1 接收标志(F14x)
IFG2.5	UTXIFG1	置位	USART1 发送标志(F14x)
IFG2.6			未实现
IFG2.7			未实现

表 3.4　模块允许寄存器 1、2

位	缩　写	初始状态	说　明
ME1.0			保留
ME1.1			保留
ME1.2			保留
ME1.3			保留
ME1.4			保留
ME1.5			保留
ME1.6	URXE0 USPIE0	复位 复位	USART0 接收允许(UART 模式) USART0 发送与接收允许(SPI 模式)
ME1.7	UTXE0	复位	USART0 发送允许(UART 模式)
ME2.0			保留
ME2.1			保留
ME2.2			保留
ME2.3			保留
ME2.4	URXE1 USPIE1	复位 复位	USART1 接收允许(UART 模式) USART1 发送与接收允许(SPI 模式)
ME2.5	UTXE1	复位	USART1 发送允许(UART 模式)
ME2.6			保留
ME2.7			保留

3.4.2 中断向量地址

中断向量和上电起始地址位于 ROM 中的 0FFFFh～0FFE0h 地址段。在 MSP430 的各个型号之间，由于开发时间的先后不同，中断向量表的设计有较大差异。当直接针对向量的物理地址编程时必须注意这一点。

表 3.5 是 MSP430F13x 及 MSP430F14x 的中断向量表。

表 3.6 是 MSP430F11x 及 MSP430F11x1 的中断向量表。

向量中包含各中断处理程序的 16 位入口地址。中断向量的优先级按递降排列。

表 3.5 F13x 及 F14x 中断向量

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 KEYV	复位	0FFFEh	15，最高
NMI 振荡器故障 FLASH 存储器非法访问	NMIIFG OFIFG ACCVIFG	(非)屏蔽 (非)屏蔽 (非)屏蔽	0FFFCh	14
Timer_B7	BCCIFG0	可屏蔽	0FFFAh	13
Timer_B7	BCCIFG1 TBIFG	可屏蔽	0FFF8h	12
比较器 A	CMPAIFG	可屏蔽	0FFF6h	11
WDT	WDTIFG	可屏蔽	0FFF4h	10
USART0 接收	URXIFG0	可屏蔽	0FFF2h	9
USART0 发送	UTXIFG0	可屏蔽	0FFF0h	8
AIX	ADCIFG	可屏蔽	0FFEEh	7
Timer_A3	CCIFG0	可屏蔽	0FFECh	6
Timer_A3	CCIFG1 CCIFG2 TAIFG	可屏蔽	0FFEAh	5
P1	P1IFG. 0-7	可屏蔽	0FFE8h	4
USART1 接收	URXIFG1	可屏蔽	0FFE6h	3
USART1 发送	UTXIFG1	可屏蔽	0FFE4h	2
P2	P2IFG. 0-7	可屏蔽	0FFE2h	1
		可屏蔽	0FFE0h	0，最低

表 3.6　F11x 及 F11x1 中断向量

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 KEYV	复位	0FFFEh	15，最高
NMI 振荡器失效 FLASH 存储器非法访问	NMIIFG OFIFG ACCVIFG	(非)屏蔽 (非)屏蔽 (非)屏蔽	0FFFCh	14
			0FFFAh	13
			0FFF8h	12
比较器 A	CMPAIFG	可屏蔽	0FFF6h	11
WDT	WDTIFG	可屏蔽	0FFF4h	10
Timer_A3	CCIFG0	可屏蔽	0FFF2h	9
Timer_A3	CCIFG1 CCIFG2 TAIFG	可屏蔽	0FFF0h	8
			0FFEEh	7
			0FFECh	6
			0FFEAh	5
			0FFE8h	4
P2	P2IFG.0-7	可屏蔽	0FFE6h	3
P1	P1IFG.0-7	可屏蔽	0FFE4h	2
			0FFE2h	1
			0FFE0h	0，最低

3.4.3　外部中断

端口 P1 和 P2 的全部 8 位都可以作为外部事件的中断输入。每一个 I/O 位都可独立编程。可以对端口任意组合成输入、输出和中断，因此能灵活地适应不同的 I/O 结构要求。

3.5　工作模式

MSP430 是为超低功耗应用开发的，它以多种工作模式来实现这一功能。如图 3.8 所示的 MSP430 工作模式状态以强有力的方式支持超低功耗的各种要求，这是通过各模块的智能化运行管理和 CPU 的状态组合而得到的。一个中断事件可将系统从各种工作模式中唤醒，而 RETI 指令又使 MSP430 返回到中断事件发生前的工作模式。

用 CMOS 技术设计的超低功耗系统有三个主要目的：

- 解决运行速度、数据流量与低功耗设计的冲突。
- 将各模块的电流消耗降至最低状态。
- 限制活动状态至最低要求。

SR 中的 4 位用于控制 CPU 和系统时钟发生器的控制位，即 CPUOff、OSCOff、SCG0 和 SCG1。对于控制活动模式 AM 的间歇运行和限制全速工作模式的运行时间，这 4 个控制位

非常有用。工作模式控制位包含在 SR 中的最大优点是在中断服务期间它作为运行状态保护在堆栈中。只要 SR 信息不改变，在 RETI 指令执行后，处理机能继续保持在中断事件发生前的工作模式。通过处理保存在堆栈中的数据或改变 SP，也可以使程序选择另一个流向，可以方便地访问堆栈和 SP 的指令组使得设计者可以分别优化程序结构。

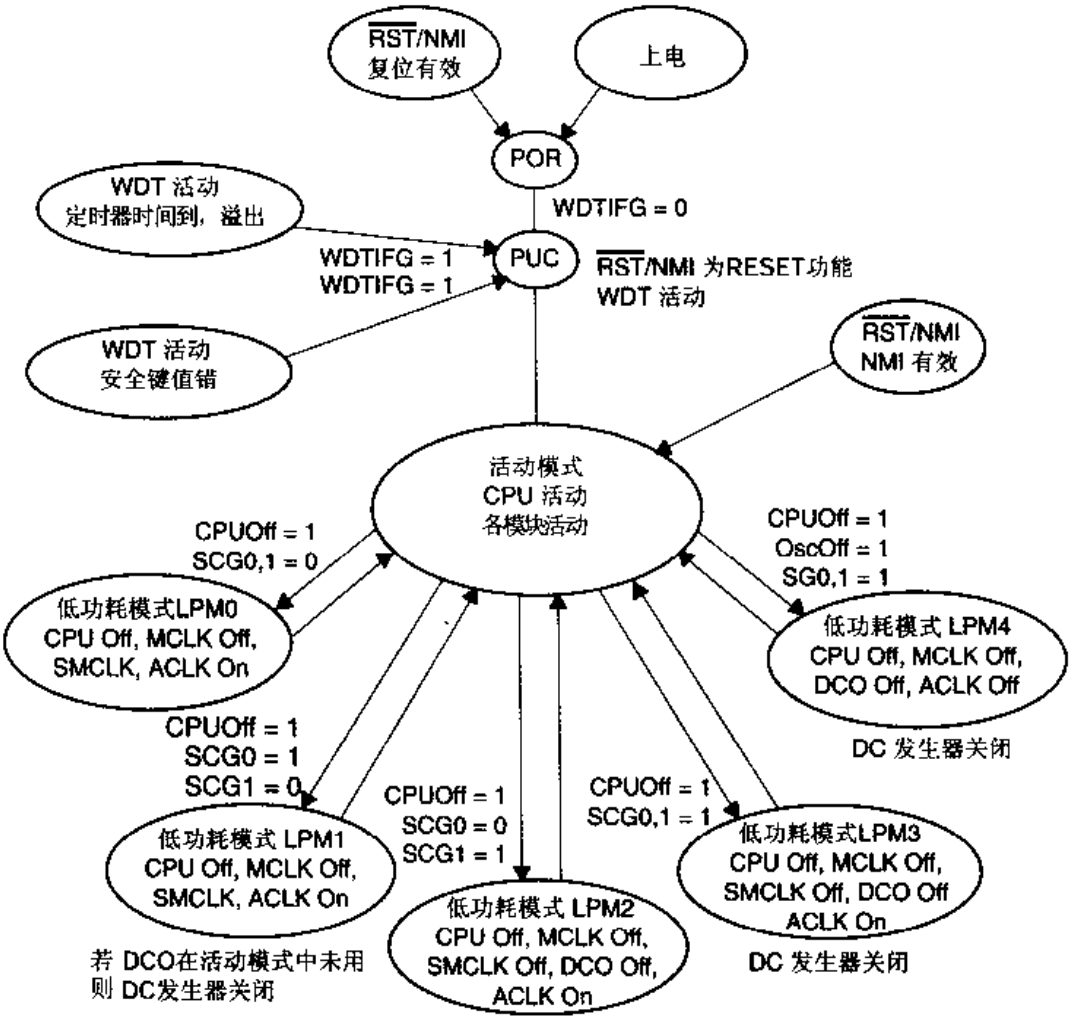


图 3.8 MSP430 工作模式状态图

进入中断程序

如果一个允许的中断唤醒 MSP430，就会进入中断服务程序，开始以下处理：

- SR 和 PC 保存入堆栈，保存了中断事件发生时的现场。
- SR 中的工作模式控制位 OSCOff、SCG1 和 CPUOff 自动被复位。

从中断程序返回

有 2 种从中断服务程序返回的方法：

- 置位低功耗模式位后返回。从中断返回后，PC 指向下一条指令。由于被恢复后的低功耗模式禁止了 CPU 的活动，PC 所指的指令不执行。
- 复位低功耗模式位后返回。从中断返回后，程序从对 SR 中的 OSCOff 或 CPUOff 置位的指令之后的地址继续执行。为了实现这种方式，中断服务程序必须将保存在堆栈中的 OSCOff、CPUOff、SCG0 和 SCG1 位清除。这样，当执行 RETI 指令后，SR 从堆栈推

当工作在低功耗模式 LPM1～4 时，因为禁止 CPU 及时钟工作，达到了降低电流消耗的目的。如果禁止外围模块，则可以进一步降低总电流消耗。各外围模块的控制是由它们各自的控制寄存器实现的。如表 3.4 所列，SFR 中的模块控制位也可以禁止或允许某些外围模块。

3.5.1 低功耗模式 0、1(LPM0 和 LPM1)

对 SR 中的 CPUOff 置位可选择进入低功耗模式"0"或"1"。置位后 CPU 立即停止运行，系统内核的常规操作停止。CPU 的操作暂停直至有任一中断请求或复位发生。所有内部总线停止活动。系统时钟发生器的继续工作和时钟信号 MCLK、SMCLK 及 ACLK 的活动取决于 SR 中的其他 3 位，即 SCG0、SCG1 和 OSCOff。

对外围模块的允许或禁止由各自的控制寄存器和 SFR 中的控制寄存器实现。I/O 端口全部引脚、RAM 及寄存器保持不变。所有被允许的中断事件可以从此状态唤醒程序。

以下举例说明进入及退出 LPM0。这一方法对于其他低功耗模式也是适用的。

下例说明进入 LPM0

```
; === Main program flow with switch to CPUOff Mode ==========
;
    BIS        #18h,  SR     ; 进入 LPM0 并允许中断控制位(CPUOff=1,GIE=1)
                             ; PC 在执行这一指令时增加，指向程序的后续步骤
    ...                      ; 如果中断服务期间 CPUOff 复位，则程序继续执行
                             ; 否则 PC 保持它的值，CPU 保持在 LPM0
```

下例说明清除 LPM0

```
; === Interrupt service routine ==========
    ...                      ; 在作中断处理时，CPU 是活动的
    BIC        #10h,  0(SP)  ; 清除 SR 中的 CPUOff 位，位于栈顶
    RETI                     ; 堆栈中的 SR 已处理，RETI 将 CPU 恢复成活动模式
```

3.5.2 低功耗模式 2、3(LPM2 和 LPM3)

对 SR 中的 CPUOff 和 SCG1 置位可选择进入低功耗模式 2 或 3。置位后 CPU、MCLK 和 SMCLK 立即停止运行。所有内部总线停止活动，直至有任一中断请求或复位发生。

工作时需要 MCLK 或 SMCLK 信号的外围模块因为时钟信号停止而停止活动。工作时需要 ACLK 信号的外围模块根据各自的控制寄存器及 SFR 中的控制位状态可能处于活动状态。I/O 端口全部引脚、RAM 及寄存器保持不变。活动的外围模块或 RST/NMI 的被允许的中断事件可以从此状态唤醒。

3.5.3 低功耗模式 4(LPM4)

全部活动部件停止，只有 RAM、I/O 端口和寄存器的内容仍保持。LPM4 只能由被允许的外部中断事件唤醒。

在启动 LPM4 前，软件要考虑在这一低功耗模式期间系统的工作条件。最重要的两点是环境温度(即 DCO 的温度效应)和定时操作。

环境温度会影响时钟频率合成的值，应该根据需要保持或是校正。当环境温度在系统处

于 LPM4 期间已发生大的改变以致于需要增加或减少时钟频率时，就需要作校正。

3.6　低功耗应用的要点

当电流消耗是系统应用的重要指标时，应该考虑以下的一些基本的设计原则：

- 如果可能，则关闭模拟电路。
- 为系统内核及外围模块选择尽可能低的工作时钟频率。
- 由于程序能快速启动，应充分利用中断驱动软件这一特性。
- 将不用的输入端接一电平，表 3.8 列出了不用的输入端正确连接的电平。

表 3.8　不用输入端的连接

引　脚	电　平	说　明
AV_{CC}	DV_{CC}	
AV_{SS}	DV_{SS}	
X OUT	开路	
XIN	DV_{SS}	
XT2IN	DV_{SS}	F13x、F14x
XT2OUT	开路	F13x、F14x
Px.0～Px.7	开路	不用的端口设置成输出端
RST/NMI	DV_{CC}	100 kΩ 上拉电阻
Test/V_{PP}	DV_{SS}	F11x
Test	DV_{SS}	F11x1
TDO	查阅芯片技术手册中的说明	
TDI		
TMS		
TCK		

4.2　存储器中的数据

字节数据可以定位在偶地址或奇地址；而字数据定位在偶地址；低字节在偶地址，高字节在下一个奇地址。因此，字指令访问时总是只用偶地址。字节与字的定位见图 4.3。

图 4.3　字节结构存储器中的位、字节和字

4.3　片内 ROM 组织

片内 ROM 可能有各种体积，可达 60 KB。公共地址空间由 SFR、外围模块寄存器、数据存储器和程序存储器共享。SFR 和外围模块安排在 0000h～01FFh 的空间中，余下的地址空间 0200h～0FFFFh 由数据存储器和程序存储器分享。

ROM 的开始地址因 ROM 的体积不同而不同。中断向量固定位于 ROM 的最高 16 个字地址区。其中最高优先级的向量位于最高的 ROM 字地址 0FFFEh。图 4.4 是 MSP430 不同型号的 FLASH 型芯片的 ROM 地址分布举例。

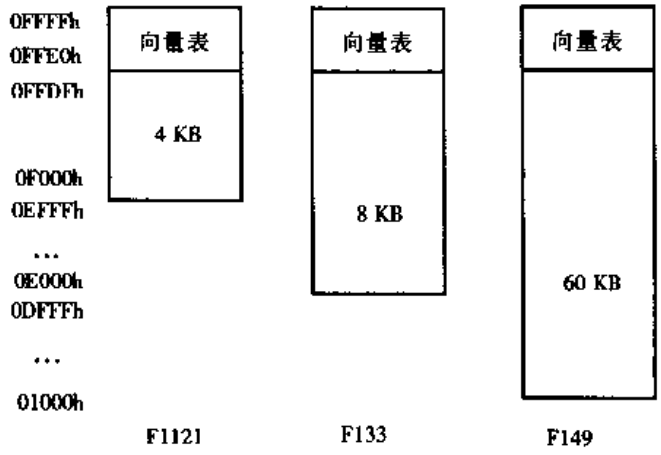


图 4.4　ROM 地址

4.3.1　ROM 表的处理

MSP430 的结构允许 ROM 存放大型数表。这在编制快速而结构清晰的程序时往往是必须的，并且能减少对 ROM 及 RAM 的占用。访问这些表可以用所有的字指令和字节指令，这一点给增加编程的灵活性和节省 ROM 空间带来各种好处，如：

- 用 ROM 存放用于显示的字符点阵。
- 实现计算型数表访问，例如用于显示动态条形图。
- 实现查表方式的程序流向控制。
- 将数据存入表中作查表处理，以实现数据线性化和补偿。

4.3.2　计算分支跳转和子程序调用

可以用标准指令实现计算分支跳转和子程序调用。CALL 和 BR 指令与其他指令采用同

样的寻址模式。

计算分支跳转和子程序调用最理想的寻址模式是间接-间接寻址。充分利用这一编程特点可以得到与常规 8 位、16 位单片机不同的程序结构。利用软件对状态的处理可以方便地实现大量的子程序调用，而不必像通常那样利用"标志"判断来控制程序流向。

计算分支跳转和子程序调用在整个 64 KB 空间内都有效。

4.4 RAM 和外围模块组织

利用适当的指令后缀，整个 RAM 都可按字节或字来访问。外围模块安排在两类地址空间，因此必须用适当的指令来访问，即

- SFR 硬件是面向字节的，安排在地址 0h～0Fh，只能用字节指令。
- 面向字节的外围模块安排在地址 010h～0FFh，只能用字节指令。
- 面向字的外围模块安排在地址 100h～01FFh，只能用字指令。

4.4.1 RAM

RAM 可以用作程序代码存储器，也可以用作数据存储器。对程序代码的访问总是针对偶地址的。

指令助记符的后缀定义了访问的数据是字还是字节。

举例：

```
ADD.B    &TCDATA,TCSUM_L                          ; 字节访问
ADDC.B   TCSUM_H                                  ; 字节访问
ADD      R5,SUM_A      或 ADD.W   R5,SUM_A        ; 字访问
ADDC     SUM_B         或 ADDC.W  SUM_A           ; 字访问
```

一个字由高字节(b15～8)和低字节(b7～0)组成，总是对准偶地址。

图 4.5 是字操作与字节操作的举例说明。

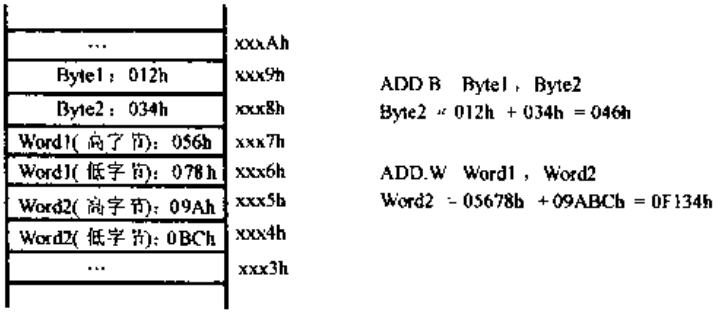


图 4.5 字和字节操作

对堆栈和 PC 的操作总是字操作，必须对准偶地址。

字-字和字节-字节操作同样能得到准确的操作数结果和状态位设置。

举例：字-字操作

R5 = 0F28Eh

字节-字节操作

R5 = 0223h

EDE . EQU 0212h
Mem(0F28Eh) = 0FFFEh
Mem(0212h) = 00112h

ADD　　@R5,&EDE

Mem(0212h) = 00110h
C = 1, Z = 0, N = 0

EDE . EQU 0202h
Mem(0223h) = 05Fh
Mem(0202h) = 043h

ADD. B　　@R5,&EDE

Mem(0202h) = 0A2h
C = 0, Z = 0, N = 1

图 4.6 是寄存器与字节存储单元间的数据传递过程说明。

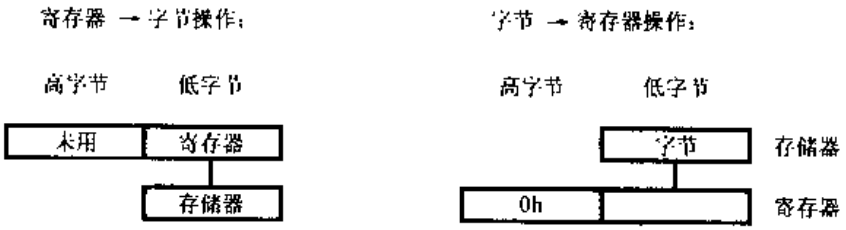


图 4.6　寄存器与字节存储单元间数据传递

举例:寄存器-字节操作

R5 = 0A28Fh
R6 = 0203h
Mem(0203h) = 012h

ADD. B　　R5,0(R6)

```
  08Fh
+ 012h
  0A1h
```

Mem(0203h) = 0A1h
C = 0, Z = 0, N = 1

（寄存器低字节）
+（寻址字节）
→（寻址字节）

字节-寄存器操作

R5 = 01202h
R6 = 0223h
Mem(0223h) = 05Fh

ADD. B　　@R6,R5

```
  05Fh
+ 002h      ; R5 的低字节
  061h      ; 存入 R5,高字节为 0
```

R5 = 061h
C = 0, Z = 0, N = 0

（寻址字节）
+（寄存器低字节）
→（寄存器低字节,高字节填 0）

注意:字-字节操作

在存储器中不支持字-字节操作或者字节-字操作。

每一次寄存器-字节操作和字节-寄存器操作的执行时相当于字节操作。

4.4.2　外围模块——地址定位

地址空间 0000h～00FFh 留作安排字节模块,地址空间 0100h～01FFh 留作安排字模块。安排在字地址空间的外围模块必须用字指令(如 MOV R5,&WDTCTL)访问,安排在字节地址空间的外围模块必须用字节指令(如 MOV. B #1,&P1OUT)访问。

这两种寻址可经过绝对寻址模式实现,也可经过采用变址、间接或间接增量寻址模式的工作寄存器来实现。图 4.7 是外围模块的空间分配及操作。

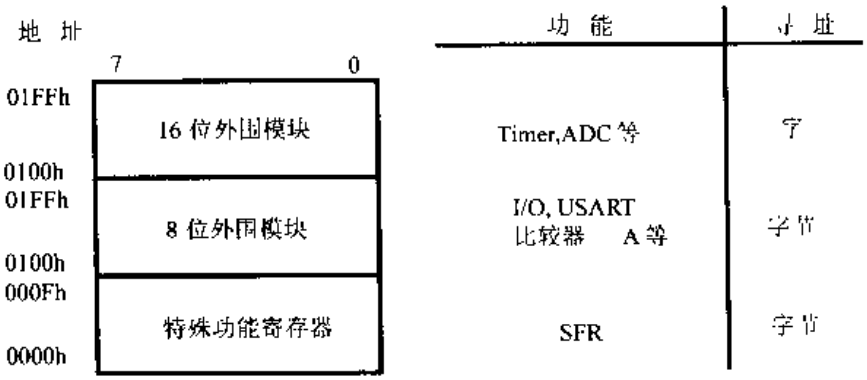


图 4.7 外围模块组织

字模块

字模块是经全部 16 位 MDB 连接的模块。字模块可以用字指令或字节指令访问。但是用字节指令访问时，只有偶地址有效，高字节的访问结果总是“0”。

字模块的寻址空间分割成 16 个帧(frame)，每帧有 8 个字，见表 4.1。

表 4.1 字模块的空间分割

地 址	说 明	地 址	说 明
1F0h～1FFh	保留	170h～17Fh	Timer_A
1E0h～1EFh	保留	160h～16Fh	Timer_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 控制与中断	120h～12Fh	WDT，FLASH 控制
190h～19Fh	Timer_B	110h～11Fh	保留
180h～18Fh	Timer_B	100h～10Fh	保留

字节模块

字节模块是经 MDB 的低 8 位连接的模块。只能以字节指令来访问字节模块。在写操作时字节模块从 MDB 的低字节取数。

字节指令对字节模块的操作无任何限制。用字指令对字节外围模块作读访问会在高字节产生不可预知的结果。写入字节模块的字数据，低字节写入相应的模块寄存器，而高字节可忽略不计。

字节模块的地址空间分割成 16 组，见表 4.2。

表 4.2 外围模块地址分配-字节模块

地 址	说 明	地 址	说 明
00F0h～00FFh	保留	0070h～007Fh	USART0，USART1
00E0h～00EFh	保留	0060h～006Fh	保留
00D0h～00DFh	保留	0050h～005Fh	系统时钟发生器，比较器 A
00C0h～00CFh	保留	0040h～004Fh	保留
00B0h～00BFh	保留	0030h～003Fh	P5，P6
00A0h～00AFh	保留	0020h～002Fh	P1，P2
0090h～009Fh	保留	0010h～001Fh	P3，P4
0080h～008Fh	ADC12 存储器控制	0000h～000Fh	SFR

4.4.3　外围模块——SFR

外围模块的配置和外围模块对 CPU 操作的响应由 SFR 的定义来确定。SFR 位于更低的地址区，以字节方式工作。因此 SFR 只能用字节指令来访问，见表 4.3。

表 4.3　SFR 地址分配

地　址	数据总线	地　址	数据总线
000Fh	未定义或未实现	0007h	未定义或未实现
000Eh	未定义或未实现	0006h	未定义或未实现
000Dh	未定义或未实现	0005h	模块允许 2(ME2. x)
000Ch	未定义或未实现	0004h	模块允许 1(ME1. x)
000Bh	未定义或未实现	0003h	中断标志 2(IFG2. x)
000Ah	未定义或未实现	0002h	中断标志 1(IFG1. x)
0009h	未定义或未实现	0001h	中断允许 2(IE2. x)
0008h	未定义或未实现	0000h	中断允许 1(IE1. x)

系统的功耗受当前允许的模块数和模块功能的影响。禁止一个活动的模块会减少系统功耗。为减少功耗，应将不用的引脚接上适当的电平。

4.5　FLASH 存储器

MSP430 的 FLASH 型芯片由于采用 FLASH 存储器作为程序代码及信息的存储，因此可以实现多次擦除和写入，也可以实现在线写入。FLASH 存储器的写入可以经过 MSP430 系列芯片的 JTAG 接口来完成，也可以由芯片内的驻留软件实现 FLASH 的写入操作，只是要求运行的程序代码存储区与待编程的存储区不在同一模块中。因此，甚至可以实现用 FLASH 存储器中的程序来对 FLASH 存储器进行编程。

4.5.1　FLASH 存储器的组织

FLASH 存储器可以由多个不同体积的模块组成。所谓 FLASH 存储器模块是一个可独立操作的物理存储器单元，见图 4.8。全部模块安排在同一个线性地址空间中。因为互相独立，可以在一个模块中运行程序，而对另一个模块作擦除或编程。

注意：MSP430 芯片的 FLASH 存储器模块

不同型号的 MSP430 芯片的 FLASH 存储器模块数量是不同的，可能一个，也可能多个。

如果运行程序代码和编程目标在同一个 FLASH 存储器模块中，程序会在标志 BUSY 为“1”时暂停，直至编程周期结束，使标志 BUSY 为“0”时为止，然后继续执行下一条指令。可以用软件擦除 FLASH 存储器，只是要当心别擦除运行程序所必须的存储器区域。

正在编程或擦除的 FLASH 存储器模块是不能访问的。这时，这一模块与片内的地址总线及数据总线的连接暂时是断开的。

一个模块又分为多个段(Segment)。图 4.9 是 MSP430F1121 的 FLASH 存储器结构实例。

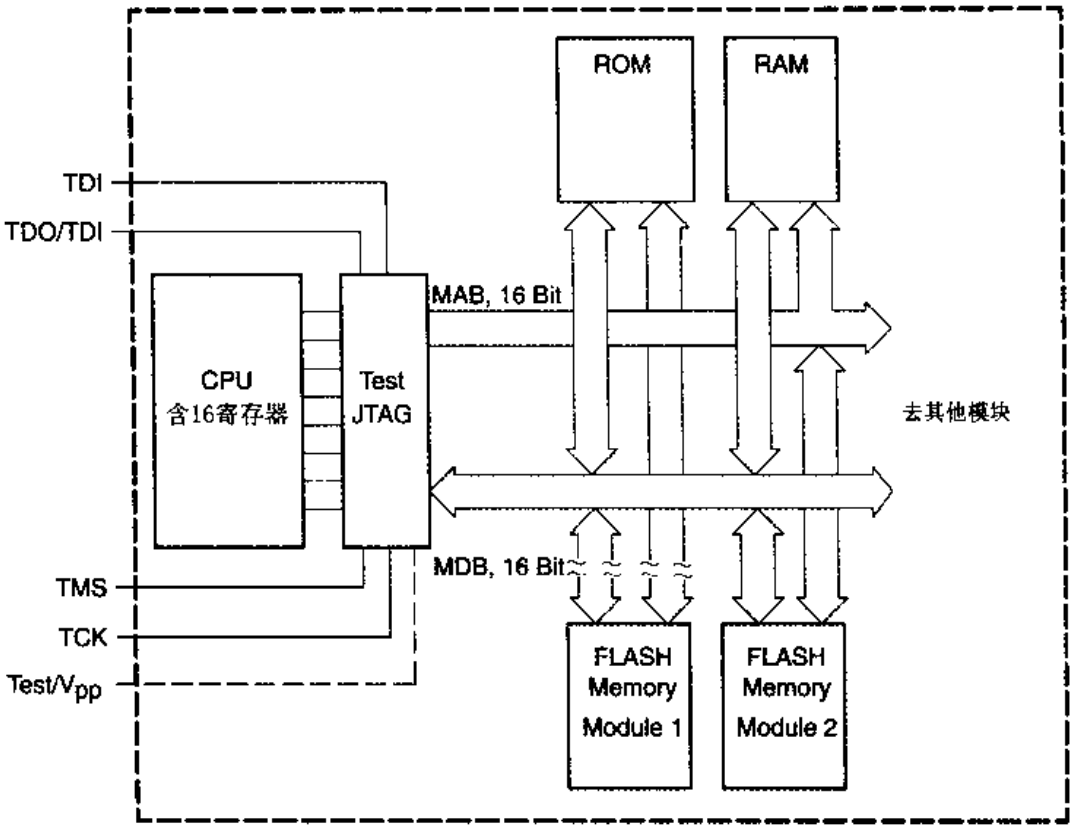


图 4.8 FLASH 存储器模块的连接

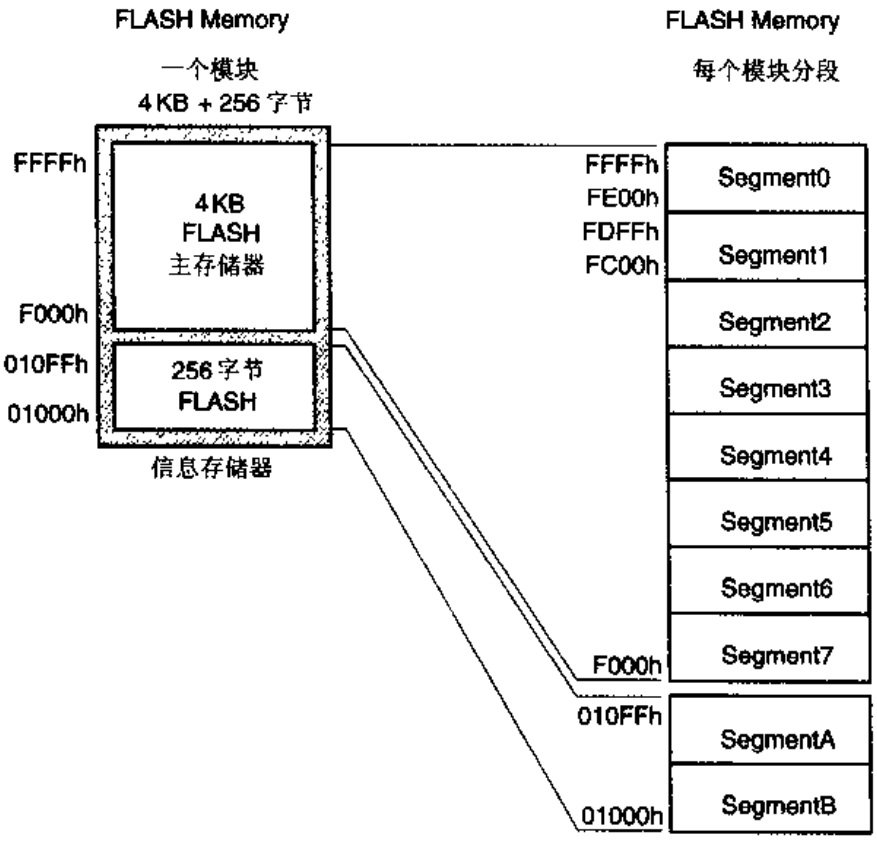


图 4.9 MSP430F1121 的 FLASH 存储器分段

当对 FLASH 存储器的段中的某一位编程时，就必须对整个段作擦除。因为这个原因，FLASH 存储器必须分为较小的段，来实现较方便的擦除和编程。

F1121 的 FLASH 存储器处于一个模块中，信息存储器有 2 个 128 字节的段(Segment A 和 Segment B)。主存储器有 8 个 512 字节的段(Segment 0～Segment 7)。Segment A 和 Segment B 可以单独擦除或成组擦除，Segment 0～Segment 7 也可以分别单独擦除或成组擦除。

4.5.2　FALSH 存储器的数据结构

FLASH 存储器可以按字或字节读写，经过擦除各位为“1”，可以写为“0”。读操作与一般的 ROM、RAM 相同。对写操作有以下限制：

- 擦除后各位为“1”，可以在任意时刻写为“0”。
- 最小的擦除单位是段，无法实现单字或单字节的擦除。
- 只能在不进行写入、编程时访问存储单元，一旦发生访问冲突会用标志指明。

FLASH 存储器基本功能

FLASH 存储器的基本功能有：

- 在程序执行时提供代码和数据。
- 在软件或 JTAG 接口控制下作一段、多段或整个模块的擦除。
- 在软件或 JTAG 接口控制下写入数据，在 x000h～x1FFh 的 512 字节区域内可实现双倍编程速度。

FLASH 存储器结构

图 4.10 是 FLASH 存储器模块的结构框图。一个 FLASH 存储器模块包含 3 个控制寄存器、时序发生器、提供编程及擦除电压的电压发生器和存储器本身。

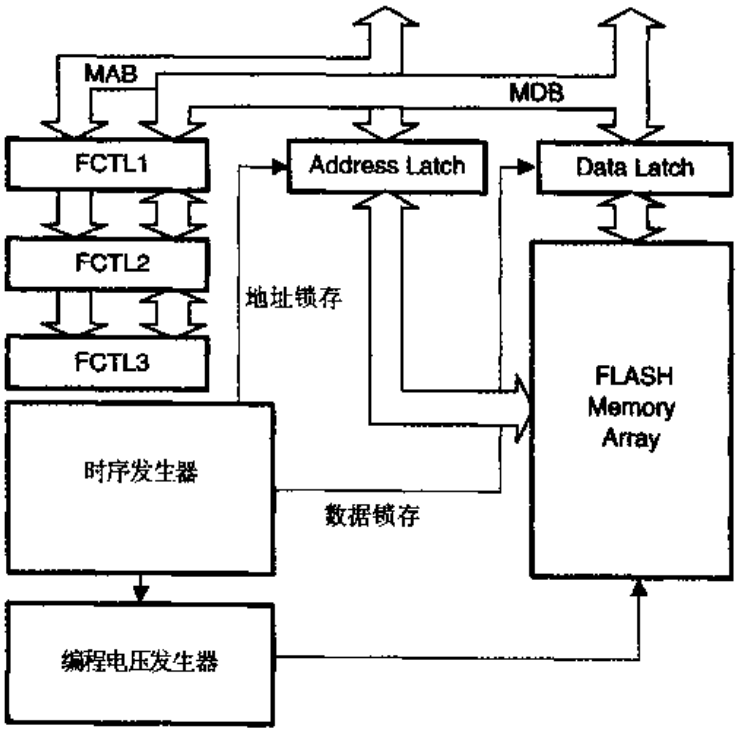


图 4.10　FLASH 存储器框图

表 4.4　编程/擦除控制位

功　能	SEGWRT	WRT	MERAS	ERASE	BUSY	WAIT	LOCK
写入	0	1	0	0	0	0	0
段写入	1	1	0	0	0	1	0
段清除并写入	0	0	0	1	0	0	0
清除 A、B 以外各段	0	0	1	0	0	0	0
全体清除并写入	0	0	1	1	0	0	0

在表中所列组合以外的控制位组合下进行的操作会引起非法访问，ACCVIFG 标志将置位；如果这时 ACCVIE 置位，将产生 NMI 中断请求。

FLASH 存储器在代码执行时的状态

FLASH 存储器提供数据或代码的方式与 ROM 或 RAM 一样。FLASH 存储器模块应处于读模式，也不能激活编程或擦除操作。

当发生 POR 时，FLASH 存储器进入缺省的读模式，不需要定义任何控制位。

FLASH 存储器在擦除时的状态

FLASH 存储器各位的缺省值为“1”。经过一次成功的擦除操作各位置位。一旦将一个位编程为“0”，只有擦除操作才能将它恢复成“1”。擦除可以对一段、几段或整个模块进行。

擦除操作顺序如下：

(1) 选择适当的时钟源及分频因子，为时序发生器提供正确的输入时钟。

(2) 如果 LOCK 位置位，将它复位。

(3) 监视 BUSY 位，直至 BUSY 位复位进入下一步。

(4) 如果擦除 1 段，则将 ERASE 位置位。

(5) 如果擦除多段，则将 MERAS 位置位。

(6) 如果擦除整个模块，则将 MERAS 及 ERASE 位均置位。

(7) 对擦除的地址范围内任意位置作一次空写入(dummy write)。

空写入操作的作用是启动擦除操作，例如 CLR &0F012h。

如果空写入操作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。例如：准备擦除 Segment 1，已经将控制位正确设置，但是空写入是针对信息存储器的，这时擦除操作不会进行，也不会有任何标志指示这一状态。

擦除操作在下列条件满足时能正确完成，即

- 在擦除周期中，选择的时钟源始终有效。
- 在擦除周期中，不修改分频因子。
- 在 BUSY 为“1”期间不再访问这一区域，包括：不从中读出，不对它写入，或不对这一区域再次作擦除操作。如果发生这些操作，则会使 KEYV 置位，并产生 NMI 中断。NMI 服务程序应能作处理。
- 电源电压应符合芯片手册中的说明，允许有较小的容差。

控制位 BUSY 指示一个活动的擦除周期。在用空写入启动时序发生器后，BUSY 立即置位。在擦除操作完成，已经擦除的段或区域可以重新访问之前，BUSY 位一直保持置位。BUSY 不能用软件置位，但是能用软件复位。

当发生紧急情况时，将 EMEX 位置位擦除操作立即停止，BUSY 位也就复位了。需要紧

急停止擦除的例子之一，是发生电源电压大幅度跌落，超出芯片的工作条件。另一个例子是因时钟源丢失引起擦除时序失控。

注意：不完整的擦除

因为硬件原因，在擦除正常完成前会因为时序发生器停止而中止擦除。对不完整的擦除可以经过检验查出，但是 FLASH 存储器的擦除可能处于临界状态。不完整的擦除在电源电压、温度、读出操作及频率等变化时，读出的值是不稳定的。

FLASH 存储器在编程时的状态

经过擦除，FLASH 存储器的各位为“1”。编程只能是从“1”改变为“0”。一旦某一位经过编程，只能用擦除使它返回“1”状态。

可以执行两种稍有差异的写操作：写单个字节或字及写字节或字序列。写字节或字序列可以采用顺序写入方法，也可以采用段写入方法。采用段写入速度大约是顺序写入的 2 倍，见图 4.13 及图 4.14。

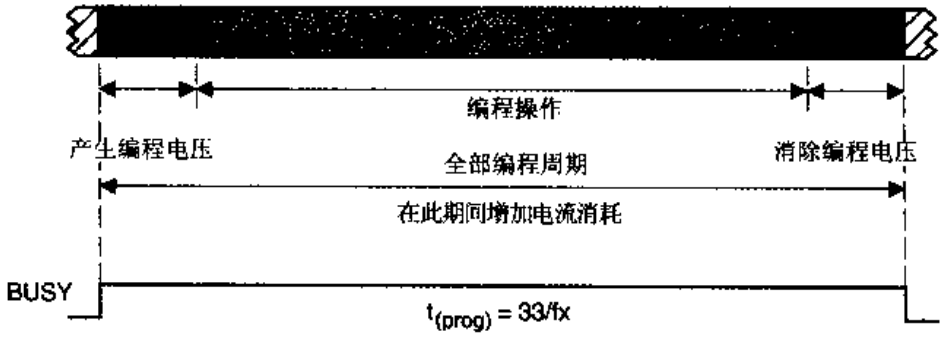


图 4.13 FLASH 存储器单字节或单字写入周期

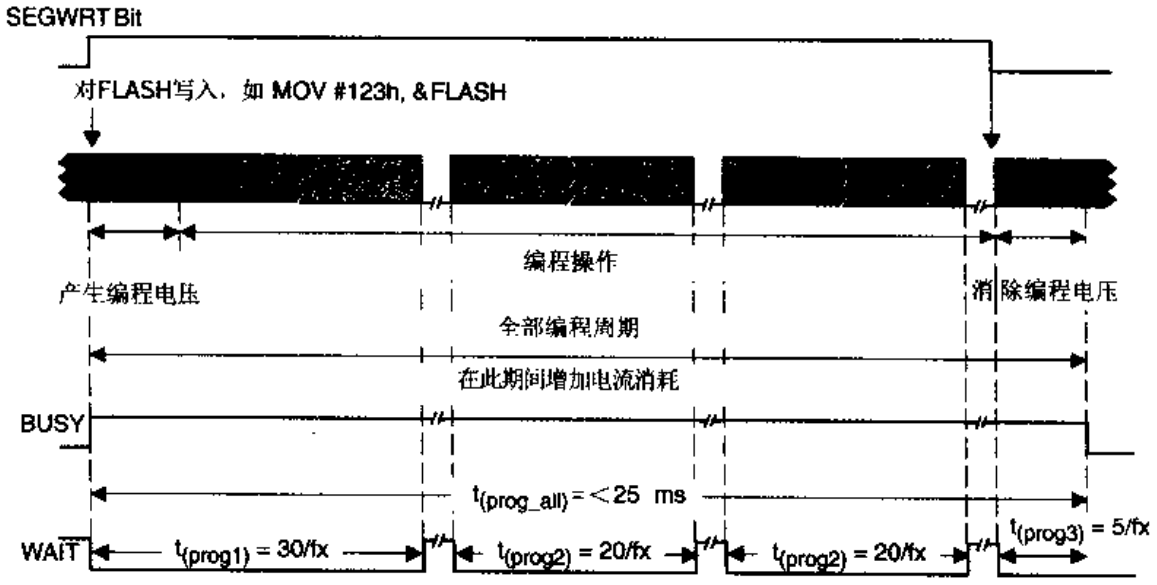


图 4.14 FLASH 存储器段写入周期

写操作按以下顺序进行：

(1) 选择适当的时钟源及分频因子，为时序发生器提供正确的输入时钟。

(2) 如果 LOCK 位置位,将它复位。

(3) 监视 BUSY 位,直至 BUSY 位复位进入下一步。

(4) 如果写入单字节或单字,将写控制位 WRT 置位。

(5) 如果用段写作多字节或多字写入,将 WRT 位及 SEGWRT 位置位。

(6) 将数据写入选定地址启动时序发生器,在时序发生器的控制下进行编程。

注意:不完整的编程

因为硬件原因,在编程正常完成前会因为时序发生器停止而中止编程。对不正确的写入可以经过检验查出,但是 FLASH 存储器的写入可能处于临界状态。这些数据在电源电压、温度、读出操作及频率等变化时,读出值是不稳定的。

可以用段地址进行顺序编程,每段为 512 字节。段内编程需要专门的软件支持,即

- 等待 WAIT 位置位,即最后的字节或字编程已完成。
- 将 SEGWRT 位复位。
- 在 FLASH 存储器的编程电压撤消前,BUSY 位仍保持置位,以免过载。

在开始新段编程前等待一段时间($t_{(rcv)}$)。

在写周期过程中不违反以下规定情况,编程才能正确完成,即

- 在写周期过程中,所选时钟源保持有效。
- 分频因子也不修改。
- 只要 BUSY 位置位,就不访问 FLASH 存储器模块。

表 4.5 是对 FLASH 存储器读出时的各种情况。

表 4.5　读 FLASH 存储器情况一览表

读出时的状态	取指令首字	BUSY	WAIT	MBD 上的数据	作　用
读操作		0	0	所读单元数据	PC=PC+2
编程周期	No	1	0	3FFF	非法访问
	Yes	1	0	3FFF(JMP $)	不影响
页擦除	No	1	0	3FFF	非法访问
	Yes	1	0	3FFF(JMP $)	不影响
主存擦除	No	1	0	3FFF	非法访问
	Yes	1	0	3FFF(JMP $)	不影响
主存与信息擦除	No	1	0	3FFF	非法访问
	Yes	1	0	3FFF(JMP $)	不影响
段编程	N. A.	1	0	3FFF	非法访问并锁定
	No	1	1	3FFF	不影响
	Yes	1	1	3FFF	非法访问并锁定

只有在读操作期间对 FLASH 存储器的读出才有意义。其他情况下对 FLASH 存储器读出都在片内的 MDB 总线上产生数据 3FFFh。如果作为指令操作码读出,则相当于指令 JMP $;否则将产生 FLASH 存储器非法访问状态。对于段编程状态,对 FLASH 存储器的读操作会引起 LOCK 位置位,可保证段编程操作的完成。但是在段编程时,段地址范围内不允许有当前要执行的程序代码,而字节或字编程时允许。

电源电压只允许在芯片手册所允许的范围内作微小的变动。

控制位 BUSY 指示编程或段编程状态在进行中。它由启动时序发生器的写数据入

FLASH 存储器指令置位。并一直保持到编程周期完成且编程电压撤消。在编程模式中，由 BUSY 指示是否可以进行下一次编程。在段编程模式中，由 WAIT 位指示是否可以进行下一次编程，而 BUSY 指示段编程是否完成。在紧急情况下，将 EMEX 位置位会立即停止编程，并关闭编程电压。若电源电压剧烈跌落，超出了芯片的工作条件，或者因丢失时钟信号使时序失控，也会使编程停下来。

4.5.3 FLASH 存储器的控制寄存器

允许编程、擦除或主存擦除要对 3 个控制寄存器的各位定义。这 3 个寄存器只能用字指令访问。并用一个安全键值来防止错误的编程或擦除周期。键值出错将使 KEYV 置位并产生非屏蔽中断请求(NMI)。它的安全键值与看门狗定时器的不同。

当发生 PUC 时，这些寄存器的全部控制位将复位。PUC 在加 V_{CC}、RST/NMI 引脚出现复位条件、看门狗引起复位或对 FLASH 存储器的操作不正常时产生。

FLASH 存储器控制寄存器 FCTL1

在擦除、主存擦除或编程期间对 FCTL1 进行写操作都会造成非法访问，使 ACCVIFG 置位。在段编程模式进行中，如 WAIT=1，对 FCTL1 写入也会如此。只有在 WAIT=0 时才能对 FCTL1 写入。

但是对于读操作没有任何限制。

FCTL1 的各位如下(高 8 位是读、写时的安全键值)：

FCTL1(0128h)

15							8	7							0
								SEGWRT	WRT	保留	保留	保留	MERAS	ERASE	保留
←——096h(读)——→								rw-0	rw-0	r0	r0	r0	rw-0	rw-0	r0
←——0A5h(写)——→															

b1：ERASE，擦除 1 段。

0：不擦除；

1：允许擦除 1 段。段号由空写入指令的地址确定。擦除完成时 ERASE 自动复位。

b2：MERAS，主存擦除，即 Segment 0～Segment n 一起擦除，n 视芯片而定。

0：不擦除；

1：允许擦除所有段。由对主存空写入指令启动。擦除完成时 MERAS 自动复位。

b6：WRT，编程时必须将它置位。

在 WRT 复位时写 FLASH 存储器，会发生非法访问状态，并使 ACCVIFG 置位。

b7：SEGWRT，段编程。如果有较多的顺序数据需要写入，可以用它节省编程时间。

在一段编程完后要对它作复位再置位。在下一条写指令执行前 WAIT 位应为“1”。

0：未选择段编程方式；

1：采用段编程。在两段的操作之间，需要作一次复位和置位。

FLASH 存储器控制寄存器 FCTL2

对于编程、擦除或主存擦除，时序发生器从选定的时钟源产生必要的时序信号。FCTL2 中的 2 个控制位 SSEL0 和 SSEL1 选择三个时钟源之一，6 位分频因子将时钟源频率分频为符合芯片技术手册的频率要求。当发生 PUC，或者用 EMEX 位作紧急退出时，时序发生器

复位。

当 BUSY 置位时，不能对 FCTL2 写入，否则会产生访问失效并使 AVCCIFG 置位。对 FCTL2 的读操作无任何限制。

FCTL2 的各位如下(高 8 位是读、写时的安全键值)：

FCTL2(012Ah)

15							8	7							0
								SSEL1	SSEL0	FN5	FN4	FN3	FN2	FN1	FN0
←——— 096h(读) ———→								rw-0	rw-1	rw-0	rw-0	rw-0	rw-0	rw-1	rw-0
←——— 0A5h(写) ———→															

b0～5：FN0～FN5 定义时钟源信号的分频因子，为 1～64，即 FN 值加 1。

b6～7：选择时钟源。

0：ACLK；

1：MCLK；

2、3：SMCLK。

FLASH 存储器控制寄存器 FCTL3

对 FCTL3 的修改无限制。WAIT 位由 PUC 复位，KEYV 由 POR 复位。

FCTL3 的各位如下(高 8 位是读、写时的安全键值)：

FCTL3(012Ch)

15							8	7							0
								保留	保留	EMEX	LOCK	WAIT	ACCVIFG	KEYV	BUSY
←——— 096h(读) ———→								r0	r0	rw-0	rw-1	r-1	rw-0	rw-(0)r(w)-0	
←——— 0A5h(写) ———→															

b0：BUSY，为“0”时指示 FLASH 存储器可读，否则可能发生非法访问。只读。应该在每个编程或擦除周期之前检查 BUSY 位。当启动编程、段编程、段擦除或主存擦除操作时，由时序发生器硬件将 BUSY 置位，在时序发生器完成它的功能后，由硬件将 BUSY复位。

0：FLASH 存储器不忙，由 POR 或时序发生器复位；

1：FLASH 存储器忙，在段编程的 WAIT 状态时也处于忙状态。

b1：KEYV，安全键值错。

0：安全键值是 0A5h；

1：对寄存器 FCTL1、2、3 写入时安全键值不是 0A5h，引起安全键值错使 KEYV 置位并引起 PUC。KEYV 可用子在 RESET 向量程序中判断引起 PUC 的原因。KEYV 不会自动复位，需要用软件复位。

安全键值的错误总会引起 PUC，而与 KEYV 状态无关。为避免软件进入死循环，在安全键值出错处理程序中不要对 FLASH 存储器控制寄存器进行写操作。可以用软件将 KEYV 置位，同样能产生 PUC。

b2：ACCVIFG，非法访问中断标志。当编程或擦除时访问 FLASH 存储器模块不正确，会使非法访问中断标志置位。如果非法访问中断允许位也置位，则中断被接受，将执行 NMI 中断向量地址的程序。读控制寄存器不会使 ACCVIFG 置位，相应的中断允

许位 ACCVIE 在 IE1 中。软件也可将 ACCVIFG 置位，同样能产生 NMI 中断。

b3：WAIT，在段编程模式中指示 FLASH 存储器可以接受编程数据，只读。

如果 SEGWRT 复位或 LOCK 置位，则 WAIT 自动复位，段编程结束使 WAIT 置位。

0：段编程已开始，编程在进行中。

1：段编程有效，当前数据编程完成，等待后续的编程数据。

b4：LOCK，可以用软件或硬件控制。可在编程、段擦除或主存擦除期间置位。置位后当前的操作能正常结束。在段编程模式中，如果在 SEGWRT 与 WAIT 置位情况下将 LOCK 置位，则 SEGWRT 与 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 与 LOCK 将置位。

0：FLASH 存储器可以作读、编程、擦除及主存擦除等操作；

1：FLASH 存储器可以读出，但是不能作编程、擦除及主存擦除等操作。当前的编程、擦除或主存擦除操作会正常结束。如果在 LOCK 置位情况下访问 FLASH 存储器，则 ACCVIFG 标志将置位。

b5：EMEX，紧急退出。只用于 FLASH 存储器擦除或编程失控情况。

0：无作用；

1：立即停止对 FLASH 存储器的操作，关闭内部控制电路，电流消耗降至活动模式水平，FCTL1 中各位复位。由于 EMEX 由硬件自动复位，读出总是为“0”。

4.5.4 FLASH 存储器的安全键值与中断

有 3 个非屏蔽中断事件共用 1 个的 NMI 向量，即：RST/NMI、振荡器失效(OFIFG)及非法访问 FLASH 存储器(ACCVIFG)。可以用软件检查 NMIFG、OFIFG 和 ACCVIFG 来确定中断源。这些标志置位后一直保持，直至用软件将它们复位。

NMI 中断处理

NMI 中断是多源中断。NMI 中断处理要关注它的全部中断源。进入中断由硬件将中断允许位复位，包括外部 NMI 中断允许 NMIIE，时钟失效中断允许 OFIE 和 FLASH 存储器非法访问中断允许 ACCVIE。由软件分别对各个中断标志复位，并根据应用的需要分别重新允许中断。如果新的 NMI 事件已被接受，则应该在所有中断处理完成后将中断允许位置位。并且应该是中断返回指令 RETI 的前一条指令。违反这一原则，堆栈可能会因为挂起的 NMI 中断而异常增长，造成失控。中断允许位的置位可以用一条位操作指令 BIS 来完成。

图 4.15 是 NMI 中断处理的流程。

保护 FLASH 存储器模块

FLASH 型 MSP430 有一个含有程序代码和中断向量的 FLASH 存储器模块。当对这一模块作编程、擦除或主存擦除操作时执行程序，就会发生非法访问。这将产生中断服务请求。但是这时从 FLASH 存储器读出的数据是 3FFFh，与向量表所定义的向量无关。

为了防止软件进入这种错误状态，应禁止所有中断源。中断允许位在修改前可以先保存在 RAM 中，以备 FLASH 存储器重新可以访问时将原先的允许状态恢复。

要禁止所有的中断服务请求，以下中断允许位应复位：

- GIE=0。
- NMIIE=ACCVIE=OFIE=0。

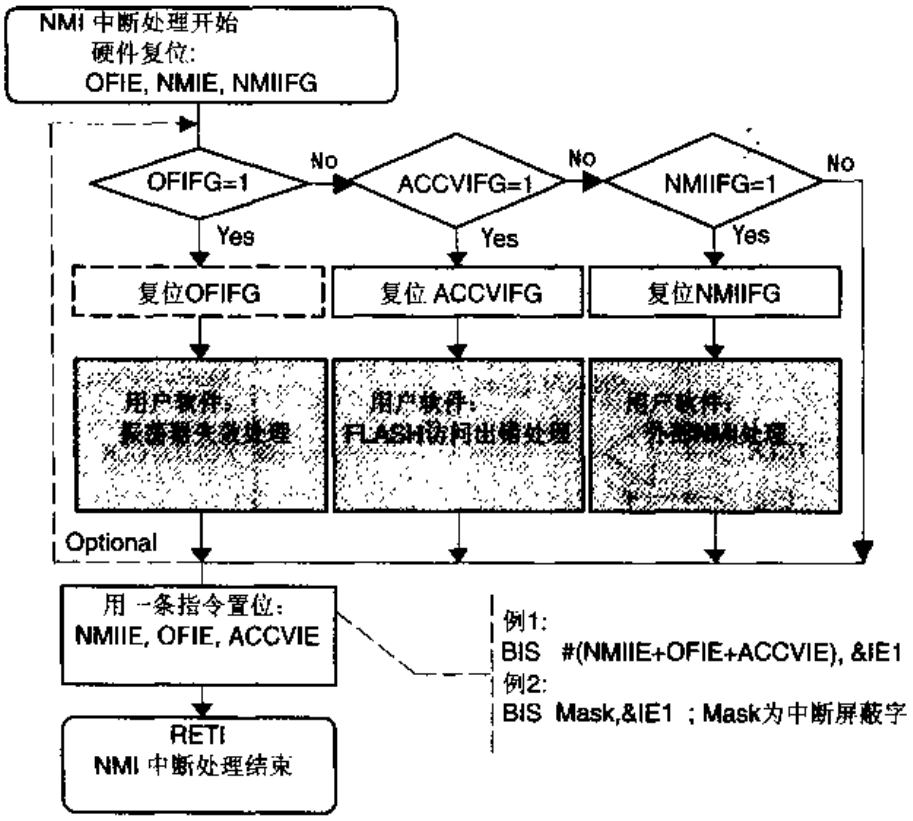


图 4.15　NMI 中断处理

此外，看门狗定时器应暂停，以避免 FLASH 存储器进入忙状态而使定时器时间耗尽，即 WDTHOLD=1。

在 FLASH 存储器可以重新访问后，中断源可以重新设置允许状态或者从原先保存在 RAM 中的状态数据恢复。在允许之前，应检查重要的中断标志，如果有必要，则由软件作中断服务或复位。

4.5.5　经 JTAG 接口访问 FLASH 存储器

FLASH 存储器保护

FLASH 存储器经过用于测试及编程的边界扫描串行接口 JTAG 的访问，在激活加密熔丝后被禁止。通过串行移入 JTAG 的指令来激活加密熔丝。熔丝的激活是不可逆的，一旦激活就不再能访问片内系统，同时 IEEE1149.1 标准所定义的旁路功能也被激活。

用 JTAG 对 FLASH 存储器编程的串行接口

JTAG 的硬件连接经过 4 个引脚，外加 TCLK、TEST、RESET 等引脚来实现。4 个 JTAG 引脚是 TMS、TCK、TDI 和 TDO。图 4.16 是连接逻辑图。

对 FLASH 存储器编程的控制软件

对 FLASH 存储器编程外部不需要专门的硬件。加在 V_{cc} 上的电源应能在编程及擦除模式中提供较大的电流。

控制软件的算法很简单，主要由嵌入在 FLASH 存储器模块内的时序发生器来控制编程及擦除过程。控制软件不能与要编程的地址在同一个 FLASH 存储器模块中，应该运行在 ROM、RAM 或另一个 FLASH 存储器模块中。以下是一些简单的程序段，程序指令将在第 5

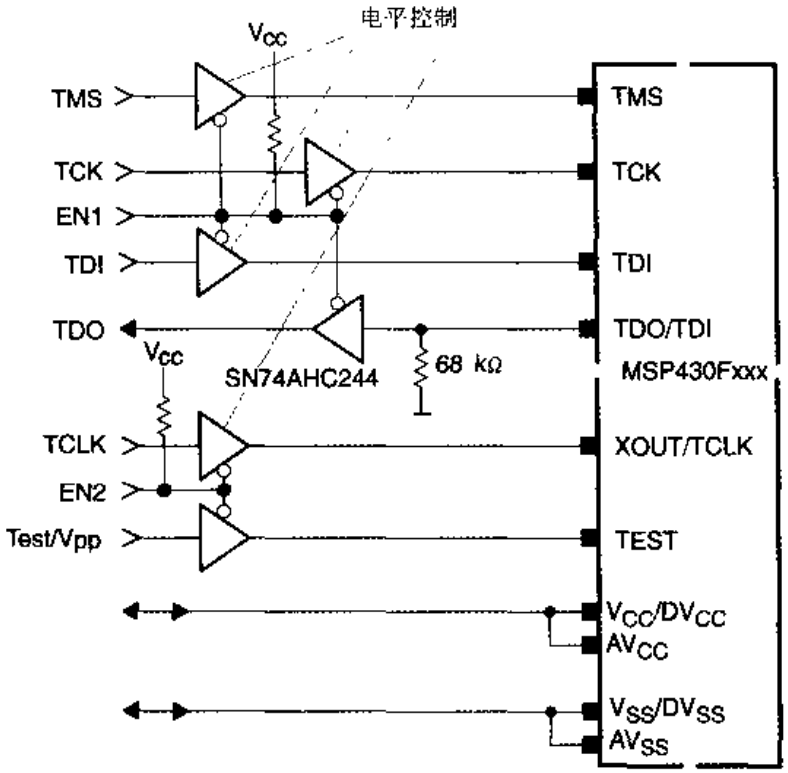


图 4.16 MSP430 的 JTAG 连接逻辑图

章中说明。

程序实例:用模块外软件将一个字编程入 FLASH 存储器

本例的软件与编程的地址在不同的存储器模块上。图 4.17 是程序流程。

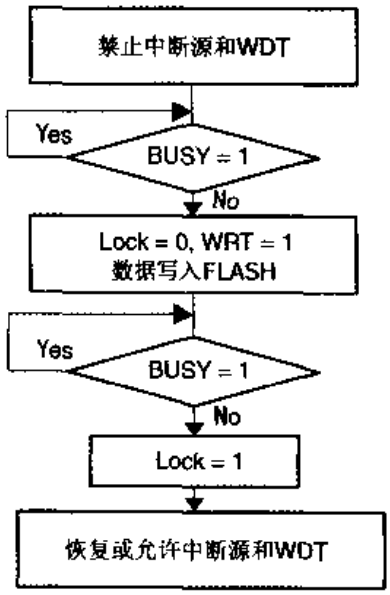


图 4.17 不同模块编程一个字的程序流程

```
Fxkey       .set        03300h
Fwkey       .set        0A500h
; 在 FLASH 存储器编程时不允许中断请求
```

```
Test_Busy1
.           BIT     #BUSY,&FCTL3
            JNZ     Test_Busy1
            MOV     #Fwkey,&FCTL3           ; 清除 LOCK 位
            MOV     #(Fwkey+WRT),&FCTL1     ; 允许编程
            MOV     #123h,&0FF1Eh           ; 编程一个字
Test_Busy2
            BIT     #BUSY,&FCTL3            ; 是否忙
            JNZ     Test_Busy2              ; 是,继续检查
            MOV     #Fwkey,&FCTL1           ; 复位编程
            XOR     #(Fxkey+Lock),&FCTL3    ; 将 LOCK 位改为"1"
; 可以允许中断
```

BUSY 位可在 FLASH 存储器编程完成前或编程开始后作如下测试：

- 如果 FLASH 存储器中只含有数据，最好在编程开始前测试 BUSY 位。由于编程操作由 FLASH 存储器的时序发生器控制，不需要 CPU 的介入，这样做有一些时间上的优点。要记住，在 BUSY 被 FLASH 存储器的硬件复位之前，时钟源要保持有效。负责进入低功耗模式的功率管理及时钟管理要确保 FLASH 存储器所用的时钟源不被关闭。
- 如果 FLASH 存储器中含有程序代码，最好在编程结束后测试 BUSY 位。程序只能在 FLASH 存储器模块重新可以访问时继续前进。执行软件代码时没有特殊的注意事项。每一次对 FLASH 存储器的编程结束时，BUSY 已复位。

在对含有程序代码的 FLASH 存储器编程之前测试 BUSY 位，能确保活动程序不会访问 FLASH 存储器模块。如：执行这一模块中的代码，对这一 FLASH 存储器作读写操作。

程序实例：用同一模块软件将一个字编程入 FLASH 存储器

程序在写 FLASH 存储器指令(MOV #123,&0FE1Eh)后等待 BUSY 再次复位。如果未用其他的编程指令，则 BUSY 位测试就无必要了。图 4.18 是程序流程。

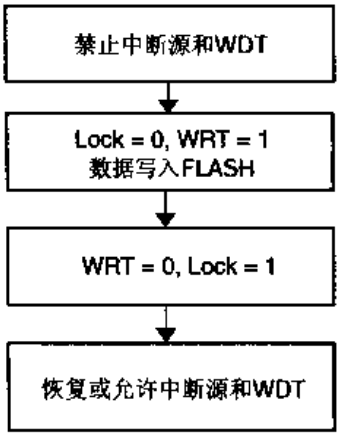


图 4.18　同一模块编程一个字的程序流程

```
Fxkey       .set        03300h
Fwkey       .set        0A500h
; 在 FLASH 存储器编程时不允许中断请求
            MOV         #(Fwkey+WRT),&FCTL1         ; 允许对 FLASH 存储器编程
```

程序实例:用同一模块程序对 FLASH 存储器作段擦除

以下程序用于实现同一模块的擦除。图 4.21 是程序流程。

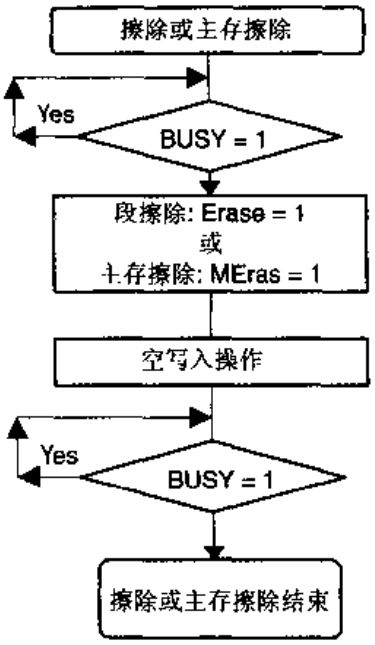


图 4.20 不同模块的段擦除的程序流程

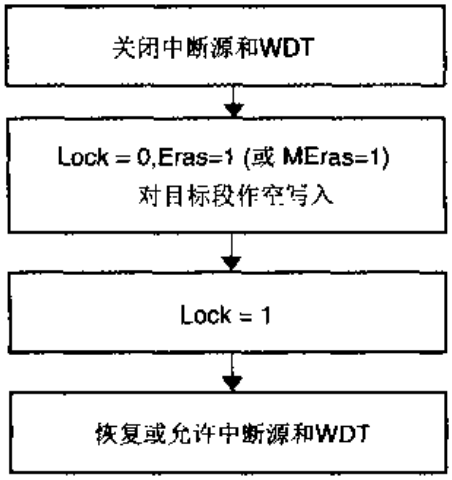


图 4.21 同一模块的擦除程序流程

```
; 禁止中断及 Watchdog
        MOV     #(Fwkey+Eras),&FCTL1        ; 允许擦除
        CLR     &0FA00h                     ; 用空写入开始擦除 Segment 2
; 如果是主存擦除,则程序安排在信息存储器中
        XOR     #(Fwkey+Lock),&FCTL3        ; 将 LOCK 位改为"1"
; 擦除位 ERASE 会自动复位
; 可以允许中断
```

程序实例:编程、擦除及主存擦除程序

以下程序不能运行在要编程或擦除的 FLASH 存储器模块中。运行时可以从 FLASH 存储器中复制到 RAM 中再执行。因此程序的编写应该是与地址无关的,运行时代码可加载到 RAM 中。这种方式可以避免与 FLASH 存储器的编程及擦除的冲突。图 4.22 是程序流程,它将程序代码移动到 RAM 中。

```
; 将程序代码移动到 RAM (load_flash_routine)
Flash_ram          .set    0222h            ; RAM 中程序首地址
Prg_source_start   .set    0xxxxh           ; 程序的开始地址
Prg_source_end     .set    0yyyyh           ; 程序的末地址
Prg_dest_start     .set    Flash_ram
load_flash_routine                          ; 需移动代码的开始
                   PUSH    R9
                   PUSH    R10
                   MOV     #Prg_source_start,R9    ; 装入移动代码源指针
                   MOV     #Prg_dest_start,R10     ; 装入移动代码目的指针
```

```
load_flash_prg
        MOV     @R9,0(R10)              ; 移动一个字
        INCD    R10                     ; 目的指针+2
        INCD    R9                      ; 源指针 + 2
        CMP     # Prg_source_end,R9     ; 检查结束地址
        JNE     load_ flash_prg
        POP     R9
        POP     R10
        RET
```

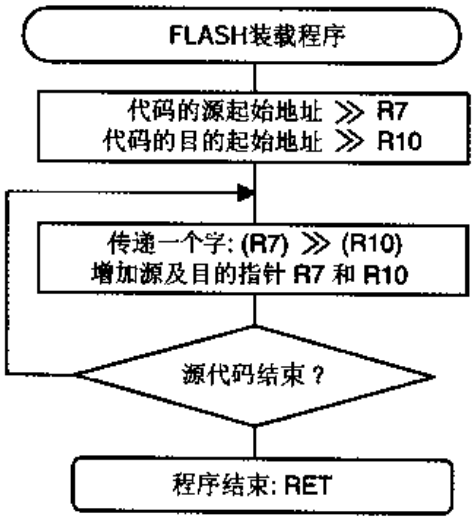


图 4.22　将程序代码移动到 RAM 中执行的程序流程

```
MOV     2(SP),R6        ; 数据项 I2 送 R6
MOV     R7,0(SP)        ; 用 R7 覆盖栈顶内容
MOV     R8,4(SP)        ; 用 R8 改数据项 I1
PUSH    #1              ; 将 1 存入 0xxxh - 6; SP 指向同一地址
POP     R8              ; 从 0xxxh - 6 恢复数据入 R12; SP 指向 0xxxh - 4
MOV     @SP+,R5         ; 数据项 I3 送 R5 (从堆栈推出); 与 POP 指令相同
```

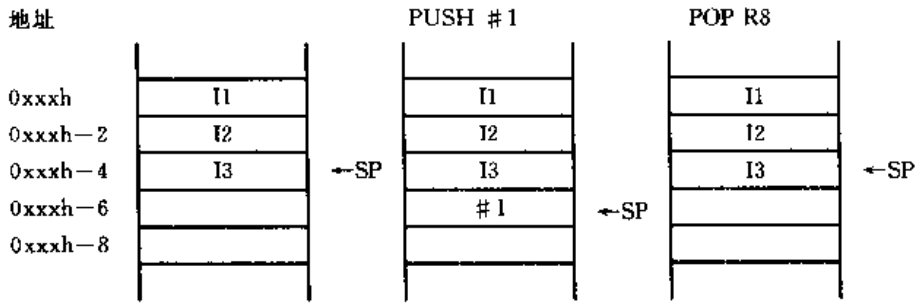


图 5.1　堆栈使用举例

系统堆栈指针 PUSH 和 POP 的特殊情况(参见图 5.2 中的堆栈)

SP 先减少 2,压入栈顶,
PUSH SP 指令后 SP 不变

SP 装入 POP SP 指令前 SP
所指存储器地址的数据

图 5.2　SP 的特殊情况

以下序列:

```
PUSH    SP          ; SP1 是执行 PUSH SP 指令后的堆栈指针
  ⋮
POP     SP          ; SP2 是执行 POP SP 指令后的堆栈指针
```

执行后,堆栈指针比序列执行前低 2 字节。

5.1.3　状态寄存器 SR

状态寄存器 SR 含有 CPU 的各状态位:

- V　　溢出位。
- SCG1　系统时钟发生器控制位 1。
- SCG0　系统时钟发生器控制位 0。
- OSCOff　晶振关闭位。
- CPUOff　CPU 关闭位。
- GIE　　通用中断允许位。

- N　　负数位。
- Z　　零位。
- C　　进位位。

SR 的结构如下：

15 9	8	7							0
保留	V	SCG1	SCG0	OSCOff	CPUOff	GIE	N	Z	C
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

SR 各状态位说明如下：

V：　当算术运算结果超出有符号数范围时置位，对字节和字格式数均有效。

ADD(.B)，ADDC(.B)　当出现以下情况时置位，即
正数 + 正数 = 负数
负数 + 负数 = 正数
其他情况下均复位。

SUB(.B)，SUBC(.B)，CMP(.B)　当出现以下情况时置位，即
正数 − 负数 = 负数
负数 − 正数 = 正数
其他情况下均复位。

SCG1、0：　控制系统时钟发生器的 4 种活动状态，因此将影响处理机运行。

OSCOff：　置位使晶振进入关闭模式：除 RAM 内容、端口和寄存器保持外，全部活动部件停止。只可能在 GIE 置位条件下由外部中断或由 NMI 唤醒。如不同时对 CPUOff 置位，则不能对它置位。

CPUOff：　置位使 CPU 进入关闭模式：除 RAM、端口、寄存器和特别允许的外围模块，如 Timer，UART 等保持活动外，全部活动模块停止。所有允许的中断可以从此状态中唤醒。

GIE：　置位使被允许的中断可以处理，复位禁止所有中断。GIE 位由中断复位，由 RETI 指令恢复。它也可用指令改变。

N：　运算结果为负时置位。
字运算：N 位设置为运算结果的 b15。
字节运算：N 位设置为运算结果的 b7。

Z：　当运算结果为“0”时置位，不为“0”时复位。

C：　当运算结果产生进位时置位，无进位时复位。
字运算：进位是指字运算结果发生进位。
字节运算：进位是指字节运算结果发生进位。
某些指令以 Z 位的非来修改 C 位。

注意：状态位 V、N、Z 和 C

状态位 V、N、Z 和 C 只在某些指令执行时改变。

5.1.4　常数发生寄存器 CG1 和 CG2

经常使用的常数可由常数发生器 R2 和 R3 产生，这样在程序中就不必占用一个 16 位字，所用的常数的数值由寻址位 As 来定义。表 5.2 是 CG1、CG2 可以发生的常数。

表 5.2　常数发生器 CG1、CG2 的值

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	00000h	0，字处理
R3	01	00001h	+1
R3	10	00002h	+2，位处理
R3	11	0FFFFh	−1，字处理

使用这种方法产生常数的优点是：

- 不需要特殊的指令。
- 对 7 种最常用的常数省去了额外的字操作数。
- 不经过 MDB 就能直接访问，因此缩短了指令执行时间。

当 6 种常数之一被用作立即寻址模式的源操作数时，汇编程序会自动转换为利用 R2 或 R3 的寻址方式。状态寄存器 SR/R2(用作源或目的寄存器)只能用于寄存器模式。其他的寻址位 As 组合支持绝对寻址和位处理而不必增加代码。R2 与 R3 用于常数模式时不能作显式寻址，它们只能作为一个寄存器形式的数据源。

常数发生器的方法使得一些指令可以用其他指令来模拟，这使 CPU 的指令系统变得异常简单，整个指令组只需要 27 条指令，例如单操作数指令：

```
CLR      dst
```

可以用双操作数指令以同样长度来模拟，即

```
MOV    R3,dst
```

或等价地有

```
MOV    #0,dst
```

其中，#0 由汇编程序以 As=00 的 R3 访问来取代，这样做使得：

- 指令长度为 1 个字。
- CPU 内不需要额外的硬件和控制操作。
- 源操作数为寄存器寻址，对常数(#0)的读取不需要额外的取指令周期。

5.2　寻址模式

对源操作数的全部 7 种寻址模式和对目的操作数的全部 4 种寻址模式可以访问整个地址空间，由 As 和 Ad 模式位的内容确定，见表 5.3。

表 5.3 寻址模式一览表

As/Ad	寻址模式	语 法	说 明
00/0	寄存器模式	Rn	寄存器内容为操作数
01/1	变址模式	X(Rn)	(Rn+X)指向操作数,X 存于后续字中
01/1	符号模式	ADDR	(PC+X)指向操作数,X 存于后续字中, 使用了变址模式的 X(PC)
01/1	绝对模式	&ADDR	指令后续字含绝对地址
10/—	间接寄存器模式	@Rn	Rn 为指针指向操作数
11/—	间接增量模式	@Rn+	Rn 为指针指向操作数,然后 Rn 增加
11/—	立即模式	#N	指令后续字含立即数 N, 使用了间接增量模式的@PC+

以下用例子详细解说 7 种寻址模式。大多数例子对源和目的操作数用了相同的寻址模式。但是对子一条指令,任何有效的源和目的操作数的寻址模式组合都是可能的。

5.2.1 寄存器模式

汇编源程序	ROM 中内容
MOV R10,R11	MOV R10,R11

长度: 1 或 2 字。

操作: 移动 R10 内容到 R11,R10 不受影响。

备注: 对源和目的操作数均有效。

举例: MOV R10,R11

执行前		执行后	
R10	0A023h	R10	0A023h
R11	0FA15h	R11	0A023h
PC	PC_{old}	PC	$PC_{old}+2$

注意:寄存器中的数据

寄存器中的数据可以用字指令或字节指令访问。如果用了字节指令,则高字节的结果总是"0",状态位将根据字节指令的结果来处理。

5.2.2 变址模式

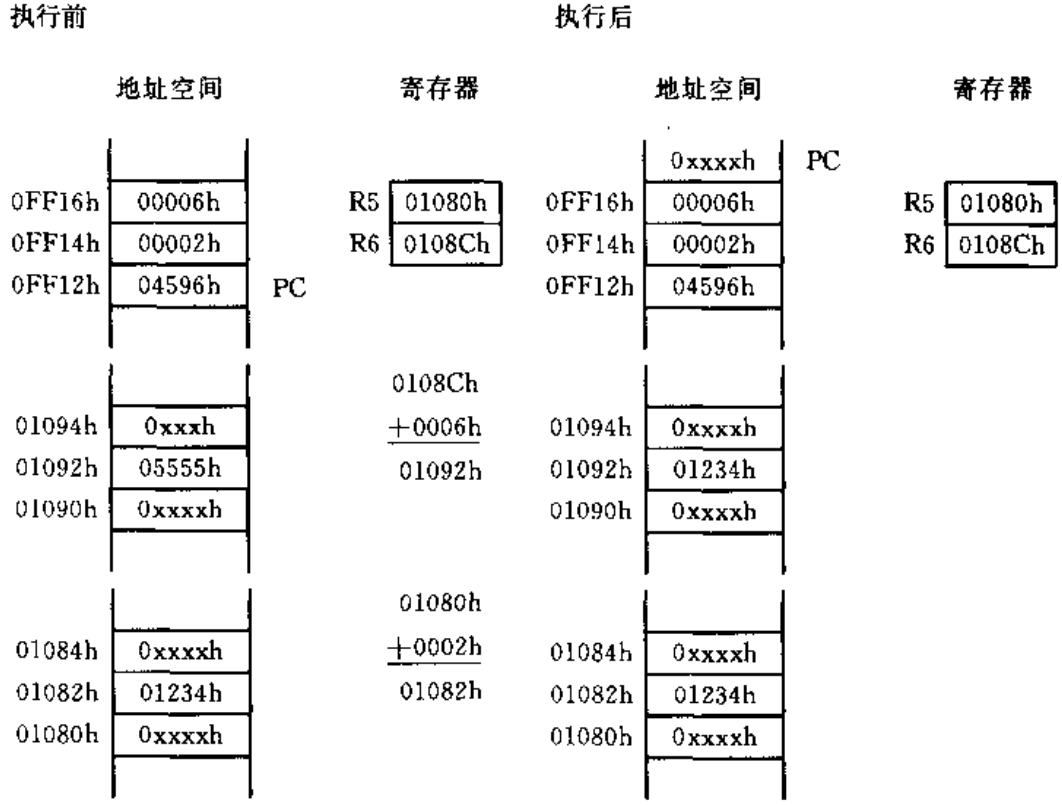
汇编源程序	ROM 中内容
MOV 2(R5),6(R6)	MOV X(R5),Y(R6)
	X = 2
	Y = 6

长度: 2 或 3 字。

操作：　移动源地址内容(R5＋2 的内容)到目的地址(R6＋6 的内容)。源和目的寄存器(R5 和 R6)不受影响。变址模式中 PC 自动增加，因此程序继续执行下条指令。

备注：　对源和目的操作数均有效。

举例：　**MOV　　2(R5),6(R6)**



5.2.3　符号模式

汇编源程序

```
MOV　　EDE,TONI
```

ROM 中内容

MOV　X(PC),Y(PC)
X = EDE －PC
Y = TONI－PC

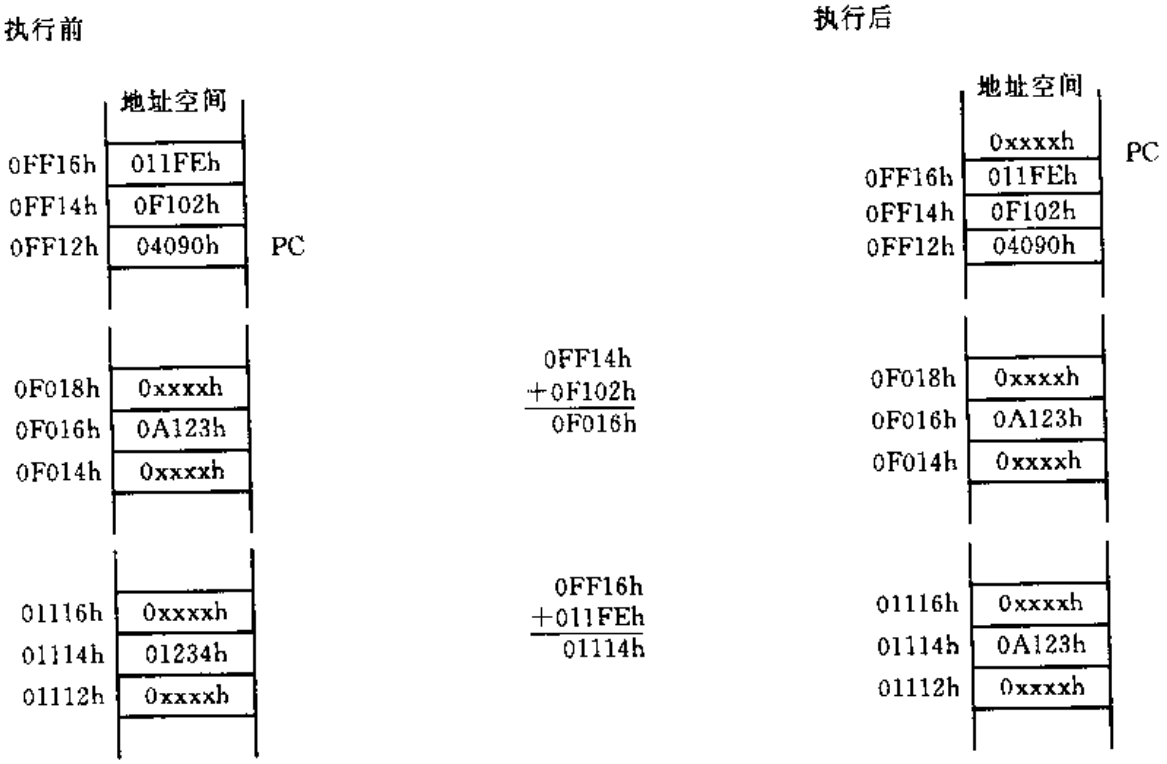
长度：　2 或 3 字。

操作：　移动源地址 EDE 内容(PC＋X 的内容)到目的地址 TONI(PC＋Y 的内容)。指令的后续字含 PC 与源和目的地址的差。汇编程序自动计算并插入偏移量 X 与 Y。符号模式中 PC 自动增加，因此程序继续执行下条指令。

备注：　对源和目的操作数均有效。

举例：

```
MOV　　EDE,TONI　　; 源地址 EDE=0F016h
                    ; 目的地址 TONI=01114h
```



5.2.4 绝对模式

汇编源程序

```
MOV     &EDE,&TONI
```

ROM 中内容

MOV X(0),Y(0)
X = EDE
Y = TONI

长度： 2 或 3 字。

操作： 移动源地址 EDE 内容到目的地址 TONI，指令的后续字含源和目的地址的绝对地址，汇编程序自动计算并插入偏移量 X 与 Y。绝对模式中 PC 自动增加，因此程序继续执行下条指令。

备注： 对源和目的操作数均有效。

举例：

```
MOV     &EDE,&TONI      ; 源地址 EDE=0F016h
                        ; 目的地址 TONI=01114h
```

此寻址模式主要用在定位于绝对的、固定地址的外围模块的访问。对它们用绝对寻址可保证软件的透明度，例如产生代码独立于位置的程序。

5.2.5　间接模式

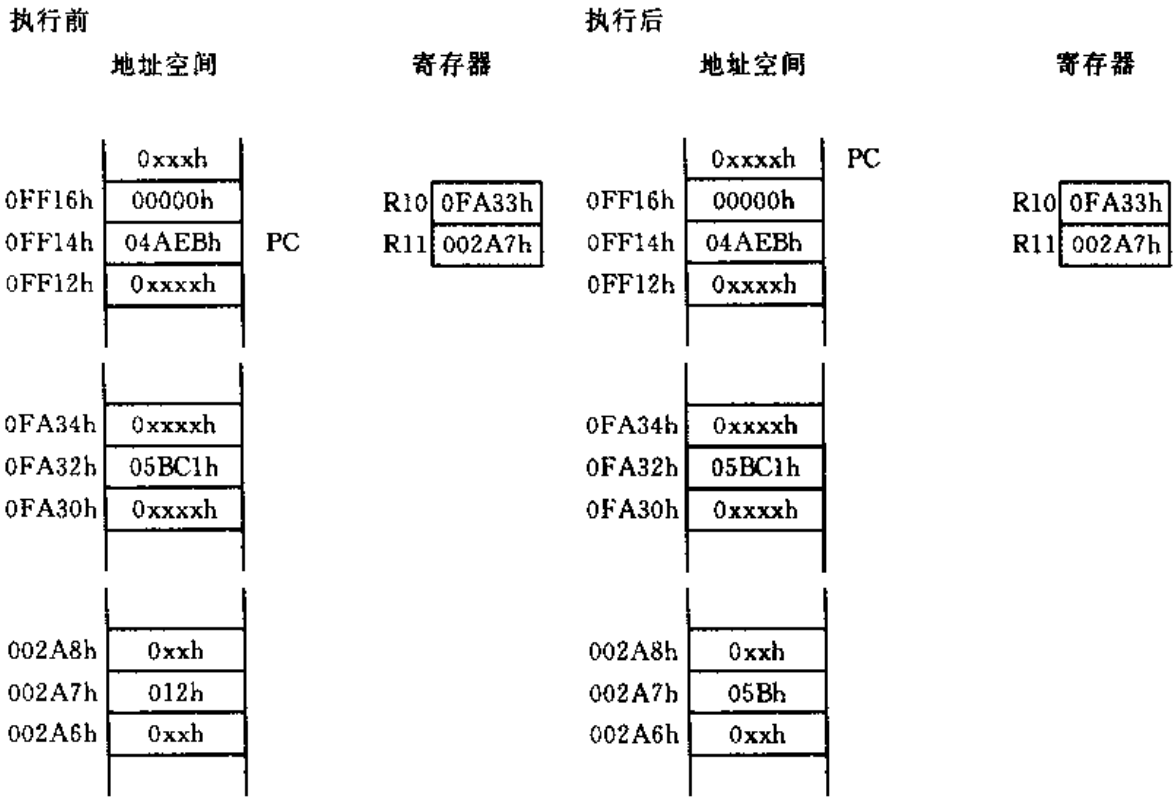
汇编源程序	ROM 中内容
MOV　@R10,0(R11)	MOV　@R10,0(R11)

长度：　1 或 2 字。

操作：　移动源地址内容(R10 内容)到目的地址(R11 内容)，寄存器不受影响。

备注：　仅对源操作数有效，对目的操作数的替代方法是 0(Rd)。

举例：　MOV.B　@R10,0(R11)



5.2.6 间接增量模式

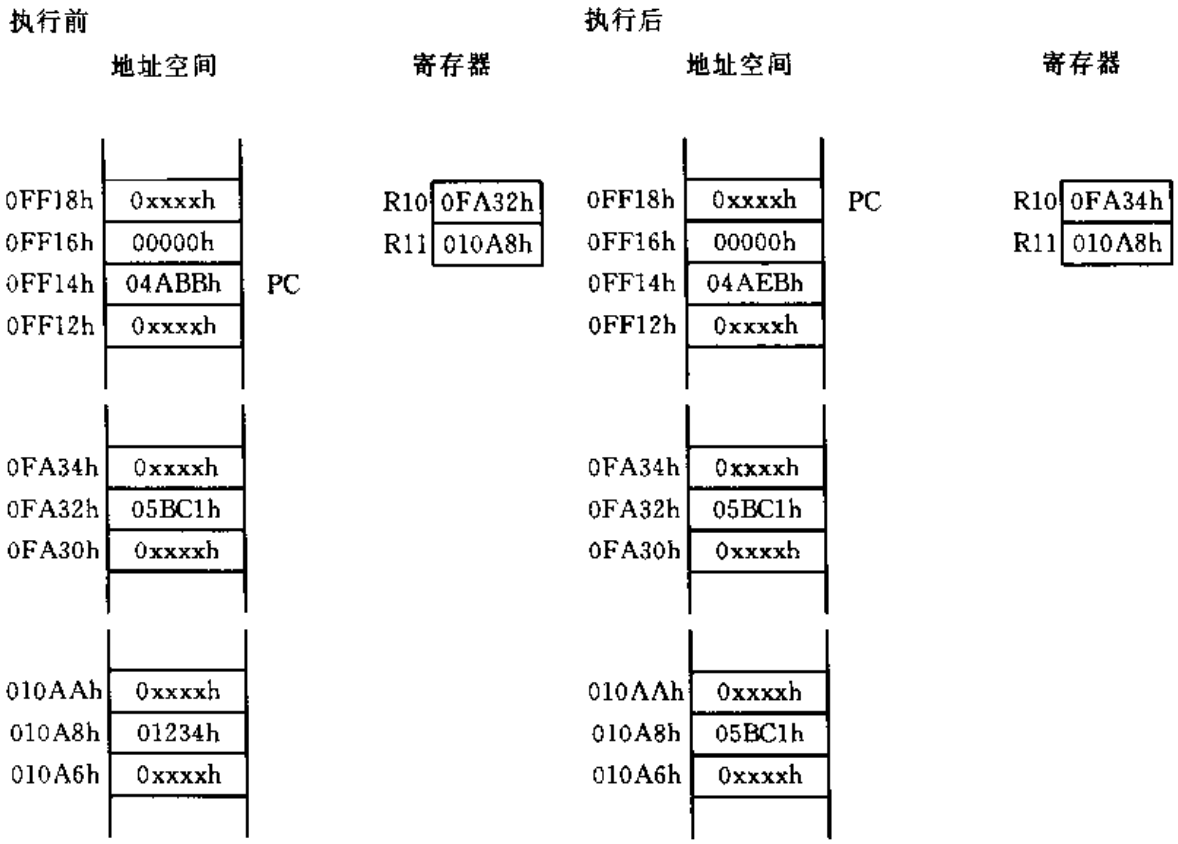
汇编源程序	ROM 中内容
MOV @R10+,0(R11)	MOV @R10+,0(R11)

长度： 1 或 2 字。

操作： 移动源地址内容(R10 的内容)到目的地址(R11 的内容)。R10 取数后增加 1(字节操作)或增加 2(字操作)；它不需要额外开销现已指向下一地址，这对于表处理非常有用。

备注： 仅对源操作数有效。对目的操作数可用 0(Rd)访问并加指令 INCD Rn 来代替。

举例： MOV @R10+,0(R11)



在执行时寄存器的自动增量在取操作数之后，如图 5.3 所示。

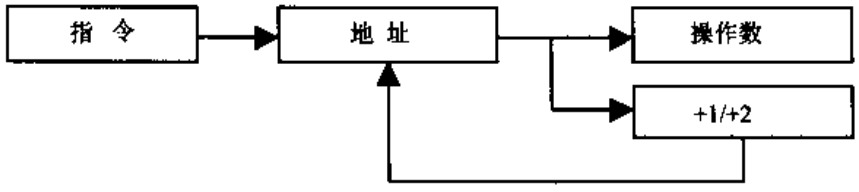


图 5.3 地址增量流程

5.2.7　立即模式

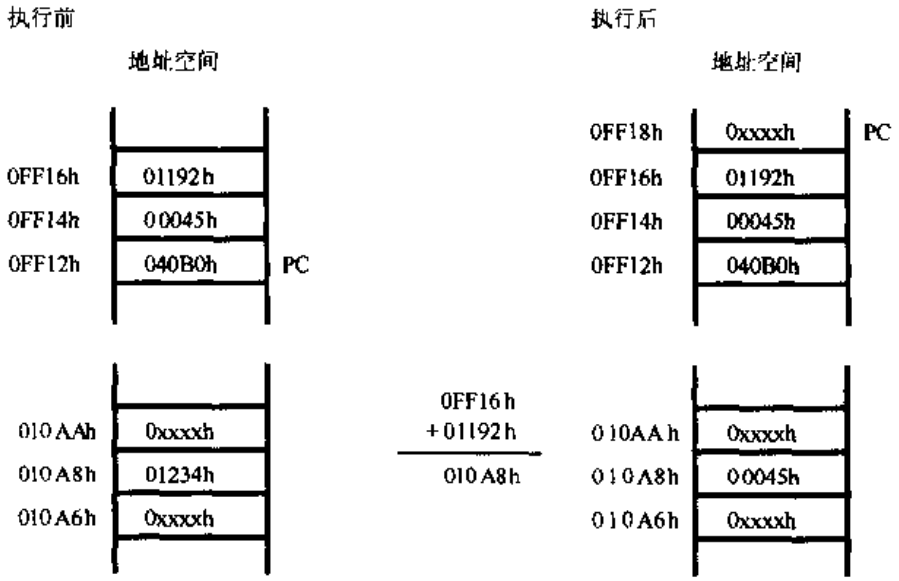
汇编源程序	ROM 中内容
MOV　　#45h,TONI	MOV @PC+,X(PC)
	45
	X = TONI−PC

长度：　2 或 3 字。

操作：　移动在后续字中的立即常数 45 到目的地址 TONI。当取得源操作数时，PC 指向后续字并移动数据到目的地址。

备注：　仅对源操作数有效。

举例：　MOV　　#45h,TONI



5.2.8　指令的时钟周期与长度

CPU 的运行速度取决于指令的格式和寻址模式。表 5.4～表 5.7 是各类指令的时钟周期，周期数是针对 MCLK 信号的。

表 5.4　I 类格式指令

寻址模式		周期数	指令长度	实　例	
As	Ad				
00,Rn	0,Rm	1	1	MOV	R5,R8
	0,PC	2	1	BR	R9
00,Rn	1,x(Rm)	4	2	ADD	R5,3(R6)
	1,EDE		2	XOR	R8,EDE
	1,&EDE		2	MOV	R5,&EDE
01,x(Rn)	0,Rm	3	2	MOV	2(R5),R7
01,EDE			2	AND	EDE,R6
01,&EDE				MOV	&EDE,R8
01,x(Rn)	1,x(Rm)	6	3	ADD	3(R4),6(R9)
01,EDE	1,EDE		3	CMP	EDE,TONI
01,&EDE	1,&EDE		3	MOV	2(R5),&TONI
				ADD	EDE,&TONI
10,@Rn	0,Rm	2	1	AND	@R4,R5

续表 5.4

寻址模式		周期数	指令长度	实 例	
As	Ad				
10,@Rn	1,x(Rm)		2	XOR	@R5,8(R6)
	1,EDE	5	2	MOV	@R5,EDE
	1,&EDE		2	XOR	@R5,&EDE
11,@Rn+	0,Rm	2	1	ADD	@R5+,R6
	0,PC	3	1	BR	@R9+
11,#N	0,Rm	2	2	MOV	#20,R9
	0,PC	3	2	BR	#2AEh
11,@Rn+	1,x(Rm)		2	MOV	@R9+,2(R4)
11,#N	1,EDE	5	3	ADD	#33,EDE
11,@Rn+	1,&EDE		2	MOV	@R9+,&EDE
11,#N			3	ADD	#33,&EDE

表 5.5 Ⅱ类格式指令

寻址模式 A(s/d)	周期数		指令长度	实 例	
	RRA RRC SWPB SXT	PUSH/ CALL			
00,Rn	1	3/4	1	SWPB	R5
01,x(Rn)	4	5	2	CALL	2(R7)
01,EDE	4	5	2	PUSH	EDE
01,&EDE				SXT	&EDE
10,@Rn	3	4	1	RRC	@R9
11,@Rn+ (*)	3	4/5	1	SWPB	@R10+
11,#N			2	CALL	#81h

注意：Ⅱ类格式指令的立即模式(见表 5.5 中 * 号)

指令 RRA、RRC、SWPB 和 SXT 不能对目的操作数用立即寻址模式，这会产生不可预知的程序运行结果。

表 5.6 Ⅲ类格式指令

	时钟周期	指令长度
Jxx	2(无论跳转与否，周期相同)	1 字

表 5.7 其他指令或操作

	时钟周期	指令长度		时钟周期	指令长度
RETI	5	1 字	WDT 复位	4	
响应中断	6		复位(RST/NMI)	4	

5.3　指令组概述

以下对指令组作简单介绍。

指令对 SR 位的影响表示如下：

*	影响状态位。
-	不影响状态位。
0	状态位复位。
1	状态位置位。

指令的源操作数和目的操作数部分由两个字段定义(寻址模式见前述)，即

src	源操作数，由 As 和 S－reg 定义。
dst	目的操作数，由 Ad 和 D－reg 定义。
As	寻址位，表示源操作数的寻址模式。
S－reg	源操作数使用的工作寄存器。
Ad	寻址位，表示目的操作数的寻址模式。
D－reg	目的操作数使用的工作寄存器。
B/W	字节或字操作，0：字操作； 1：字节操作。

注意：目的地址

目的地址可以是 64 KB 范围内任何一处。数据将写入的操作数目的地址必须是能够写入的地址，否则数据将丢失。例如 ROM 可以是一个有效的目的地址，但是却无法修改目的地址的内容，因此指令执行的结果将丢失。

5.3.1　双操作数指令

双操作数指令格式如下，有关指令见表 5.8。

15 14 13 12	11 10 9 8	7	6	5 4	3 2 1 0
操作码	S-Reg	Ad	B/W	As	D-Reg

表 5.8　双操作数指令

指　令		操　作	状态位			
			V	N	Z	C
MOV	src,dst	src→ dst	-	-	-	-
ADD	src,dst	src + dst→ dst	*	*	*	*
ADDC	src,dst	src + dst + C→ dst	*	*	*	*
SUB	src,dst	dst + . not. src + 1→ dst	*	*	*	*
SUBC	src,dst	dst + . not. src + C→ dst	*	*	*	*
CMP	src,dst	dst－src	*	*	*	*
DADD	src,dst	src + dst + C→ dst (dec)	*	*	*	*
AND	src,dst	src . and. dst→ dst	0	*	*	*
BIT	src,dst	src . and. dst	0	*	*	*
BIC	src,dst	. not. src . and. dst→ dst	-	-	-	-
BIS	src,dst	src . or. dst → dst	-	-	-	-
XOR	src,dst	src . xor. dst → dst	*	*	*	*

注意:CMP 和 SUB 指令

除了结果的保存,指令 CMP 与 SUB 相同,这一点对于指令 BIT 与 AND 也是同样的。

5.3.2 单操作数指令

单操作数指令格式如下,有关指令见表 5.9。

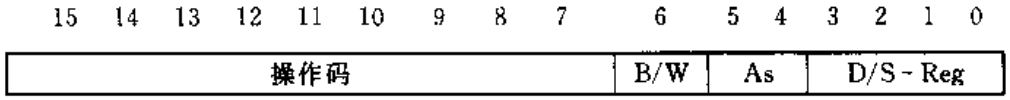


表 5.9 单操作数指令格式

指 令		操 作	状态位			
			V	N	Z	C
RRC	dst	C → MSB→…LSB→C	*	*	*	*
RRA	dst	MSB →MSB→…LSB→C	0	*	*	*
PUSH	src	SP − 2 →SP, src→@SP	-	-	-	-
SWPB	dst	字节交换	-	-	-	-
CALL	dst	SP−2→SP	-	-	-	-
		PC+2→堆栈, dst→PC				
RETI		TOS→SR, SP←SP + 2	*	*	*	*
		TOS→PC, SP←SP + 2				
SXT	dst	Bit7→Bit8→…Bit15	0	*	*	*

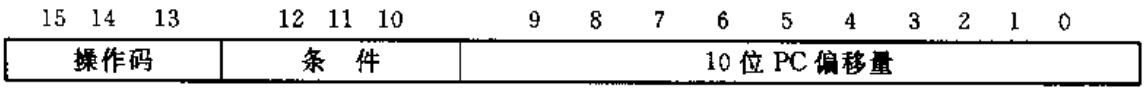
CALL 指令可以用所有寻址模式,如用符号模式(地址)、立即模式(#N)、绝对模式(&EDE)或变址模式(X(Rn)),指令的后续字含地址信息。

5.3.3 条件跳转

条件跳转指令可使程序产生相对于 PC 的分支跳转。可跳转地址是相对于跳转指令执行时 PC 的−511～+512 字范围内。10 位的 PC 偏移量是一个 10 位的有符号数,乘 2 后与 PC 相加。条件跳转不影响状态位。取指令码和 PC 增加按以下公式

$$PCnew = PCold + 2 + PCoffset \times 2$$

条件跳转指令格式如下,有关指令见表 5.10。



CLRC		C 位复位	- - - 0	BIC	#1,SR
CLRN		N 位复位	- 0 - -	BIC	#4,SR
CLRZ		Z 位复位	- - 0 -	BIC	#2,SR
POP	dst	数据出栈	- - - -	MOV	@SP+,dst
SETC		C 位置位	- - - 1	BIS	#1,SR
SETN		N 位置位	- 1 - -	BIS	#4,SR
SETZ		Z 位置位	- - 1 -	BIS	#2,SR
TST[.W]	dst	目的操作数测试	0 * * *	CMP	#0,dst
TST.B	dst	目的操作数测试	0 * * *	CMP.B	#0,dst
程序控制指令					
BR	dst	跳转	- - - -	MOV	dst,PC
DINT		关中断	- - - -	BIC	#8,SR
EINT		开中断	- - - -	BIS	#8,SR
NOP		空操作	- - - -	MOV	#0h,#0h
RET		从子程序返回	- - - -	MOV	@SP+,PC

5.3.5 其他指令

对有些操作数,如 CPUOff 等,没有相应的指令。这些功能的打开和关闭是通过对 SR 及外围模块寄存器中各位的复位和置位来实现的,还有的是用双操作数指令来模拟。

指令举例:

```
BIS     #28h,SR     ; 进入 OSCOff 模式,并允许通用中断 GIE 位
BIS     #18h,SR     ; 进入 CPUOff 模式,并允许通用中断 GIE 位
```

第 6 章　硬件乘法器

硬件乘法器是一个对于高速运算应用非常有用的外围模块。它可以明显提高程序的运行速度，大大减轻 CPU 的程序开销。在 MSP430F14x 系列中，片内实现了一个功能强大的硬件乘法器模块。

6.1　硬件乘法器

硬件乘法器是一个 16 位的外围模块，它并不集成在 CPU 内，因此，它的运行独立于 CPU，也不需要特殊的指令。运行时，只需将操作数装入它的寄存器，在下一条指令即可得到运算结果，对于乘法运算不需要额外的等待周期。硬件乘法器在不改变 MSP430 的基本结构的情况下增强了它的功能。可以实现的乘法运算为

- 16 位×16 位　● 16 位×8 位　● 8 位×16 位　● 8 位×8 位

硬件乘法器支持的乘法类型有：无符号乘(MPY)、有符号乘(MPYS)、无符号乘加(MAC)和有符号乘加(MACS)。图 6.1 是硬件乘法器与片内总线的连接。

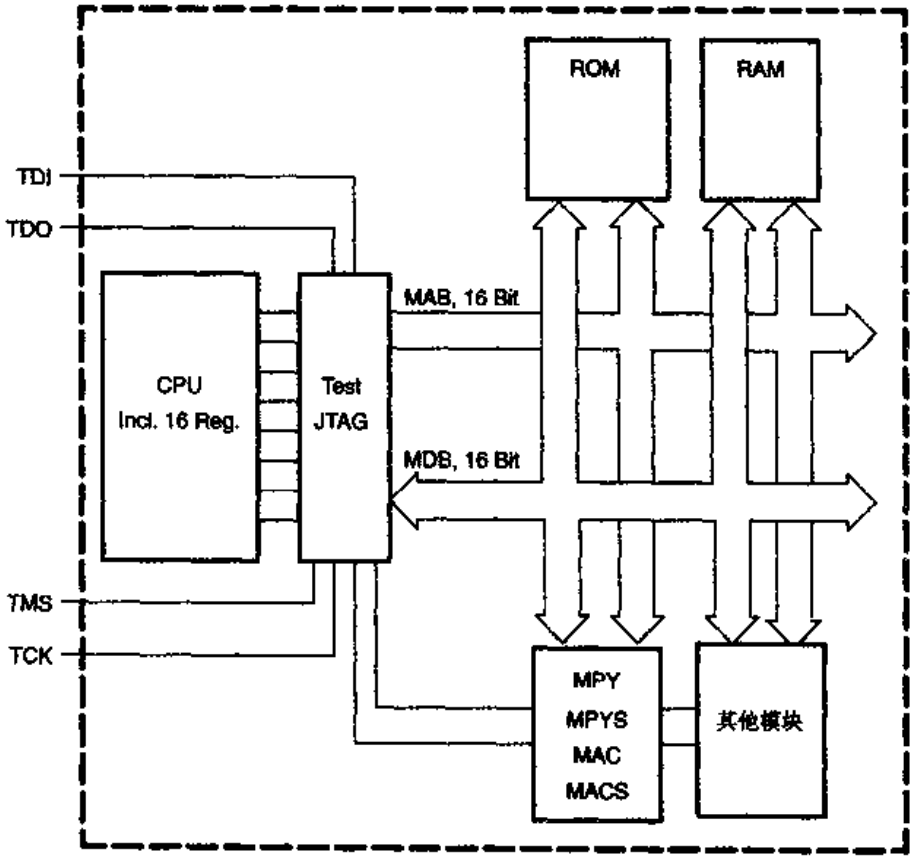


图 6.1　硬件乘法器模块与总线的连接

6.2 硬件乘法器操作

硬件乘法器有 2 个存放操作数的 16 位寄存器(OP1、OP2)和 3 个存放相乘结果的寄存器(RESLO、RESHI、SUMEXT)。只有当用户将第一操作数在第二操作数之前存入操作数寄存器中时,乘法操作才会正确执行。当第一操作数写入相应的寄存器时,乘法的类型即被确定。当用户写入第二个操作数时,乘法操作便开始执行并在用户用变址寻址模式读结果寄存器之前结束。如果用户采用间接寻址或间接增量寻址模式指令读结果寄存器,就必须在写第二个操作数与读结果寄存器的两条指令之间插入一条指令。硬件乘法器中两个操作数的寄存器均可用 5.2 节所述的 7 种寻址模式中的任一种进行访问。乘法运算不需要指令,使得实际操作时不需要增加时钟周期。

图 6.2 是硬件乘法器的硬件结构。

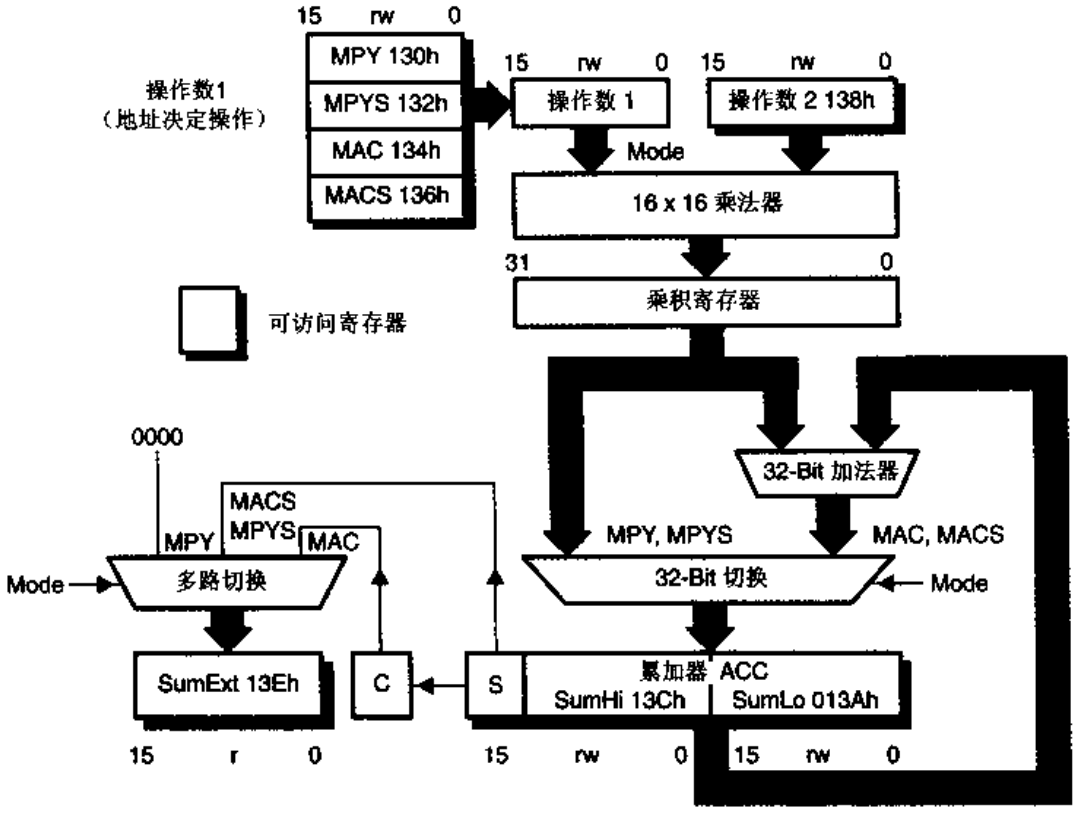


图 6.2 16 位×16 位硬件乘法器

结果扩展寄存器(SUMEXT)的内容因运算结果而异。表 6.1 是 SUMEXT 与运算结果的关系。

有符号乘加运算在以下两种情况下会发生溢出,必须加以避免或用软件处理。

- 有符号乘加运算结果为正且大于 07FFFFFFFh。这时 SUMEXT 为 0FFFFh,结果寄存器出现负数(80000000h—0FFFFFFFFh)。
- 有符号乘加运算结果为负且小于等于 07FFFFFFFh。这时 SUMEXT 为 0000h,结果寄存器出现正数(00000000h—07FFFFFFFh)。

```
.ENDIF
*****************************************************************
*  例:将硬件乘法器结果加入 64 位 RAM 数据                        *
*****************************************************************
ADD     &RESLO,&RAM      ;结果低字加入 RAM
ADDC    &RESHI,&RAM+2    ;结果高字加入 RAM+2
ADC     &RAM+4           ;进位加入扩展字
ADC     &RAM+6           ;进位加入扩展字
```

程序代码 32 字节,执行周期,32 (16 位×16 位)。

6.2.2 有符号数相乘(16 位×16 位、16 位×8 位、8 位×16 位、8 位×8 位)

```
*****************************************************************
*  将 2 个操作数送入硬件乘法器的寄存器                           *
*  如果有 1 个操作数为 8 位,则需要作符号扩展                     *
*  用常数 OPERAND1 和 OPERAND 2 来标明是否为字节数据              *
*****************************************************************
OPERAND1    .EQU    0                   ;0: OPERAND1 为字 (16 Bit)
                                        ;8: OPERAND1 为字节 ( 8 Bit)
OPERAND2    .EQU    0                   ;0: OPERAND2 为字 (16 Bit)
                                        ;8: OPERAND2 为字节 ( 8 Bit)
MPY         .EQU    0130h
MPYS        .EQU    0132h
MAC         .EQU    0134h
OP2         .EQU    0138h
RESLO       .EQU    013Ah
RESHI       .EQU    013Ch
SUMEXT      .EQU    013Eh
            .BSS    OPER1,2,200h
            .BSS    OPER2,2
            .BSS    RAM,8
            .IF OPERAND1=0
            MOV     &OPER1,&MPYS        ;装入操作数 1
                                        ;定义有符号乘法
            .ELSE
            MOV.B   &OPER1,&MPYS        ;装入操作数 1
                                        ;定义有符号乘法
            SXT     &MPYS               ;字节数据经符号扩展为字数据
            .ENDIF
            .IF OPERAND2=0
            MOV     &OPER2,&OP2         ;装入操作数 2,开始有符号乘法运算
            .ELSE
            MOV.B   &OPER2,&OP2         ;装入操作数 2
```

```
*************************************************************
*  例:将硬件乘法结果加入 64 位 RAM 数据                      *
*     乘积保存在 RESLO 和 RESHI 寄存器中。高 2 字保存在 RAM 中  *
*************************************************************
ADDC    &SUMEXT,&RAM+4          ;结果扩展字加入 RAM+4
ADC     &RAM+6                  ;结果扩展字加入 RAM+6
```

程序代码 32 字节,执行周期 32 (16 位×16 位)。

6.2.4 有符号数乘加(16 位×16 位、16 位×8 位、8 位×16 位、8 位×8 位)

```
*************************************************************
*  将 2 个操作数送入硬件乘法器,乘积加入结果寄存器 RESLO 和 RESHI  *
*  用常数 OPERAND 1 和 OPERAND 2 来标明是否为字节数据          *
*************************************************************
OPERAND1    .EQU    0           ;0: OPERAND1 是字 (16 Bit)
                                ;8: OPERAND1 是字节 ( 8 Bit)
OPERAND2    .EQU    0           ;0: OPERAND2 是字 (16 Bit)
                                ;8: OPERAND2 是字节 ( 8 Bit)
MPY         .EQU    0130h
MPYS        .EQU    0132h
MAC         .EQU    0134h
OP2         .EQU    0138h
RESLO       .EQU    013Ah
RESHI       .EQU    013Ch
SUMEXT      .EQU    013Eh
MAXMACS     .EQU    32h         ;MACS运算次数,运算至发生上溢出或下溢出
            .BSS    OPER1,2,200h
            .BSS    OPER2,2
            .BSS    RAM,8
            .BSS    MCOUNT,2
            .IF OPERAND1=8
            MOV.B   &OPER1,&MACS ;装入操作数 1,定义有符号乘加
            STX     &MACS       ;字节数据经符号扩展成字数据
            .ELSE
            MOV     &OPER1,&MACS ;装入操作数 1,定义有符号乘加
            .ENDIF
            .IF OPERAND1=8
            STX     &OPER2      ;OPER2 需要 2 字节存储空间
            MOV     &OPER2,&OP2 ;装入操作数 2,开始乘加运算
            .ELSE
            MOV     &OPER2,&OP2 ;装入操作数 2,开始乘加运算
            .ENDIF
```

```
*************************************************************
*  例:将硬件乘法结果加入 64 位 RAM 数据                      *
*      乘积保存在 RESLO 和 RESHI 寄存器中。高 2 字保存在 RAM 中  *
*************************************************************
INC     MCOUNT                  ; MACS 次数计数
CMP     #MAXMACS,MCOUNT         ; 判断是否要加入 RAM
JNE     NEXTMACS                ;
ADDC    &RESLO,&RAM+0           ; 结果扩展字加入 RAM+0
ADDC    &RESHI,&RAM+2           ; 结果扩展字加入 RAM+2
ADDC    &SUMEXT,&RAM+4          ; 结果扩展字加入 RAM+4
ADC     &RAM+6                  ; 结果扩展字加入 RAM+6
CLR     MCOUNT
NEXTMACS:   …
```

6.3　硬件乘法器寄存器

硬件乘法器的硬件模块虽然是以字为基本单位，但是用户仍可以用处理字或字节的指令对字或字节进行操作。表 6.2 是相关的寄存器。

表 6.2　硬件乘法器的寄存器

寄存器	缩　写	寄存器类型	地　址	初始状态
无符号相乘/操作数 1	MPY	读/写	0130	不改变
有符号相乘/操作数 1	MPYS	读/写	0132	不改变
有符号乘加/操作数 1	MAC	读/写	0134	不改变
有符号乘加/操作数 1	MACS	读/写	0136	不改变
操作数 2	OP2	读/写	0138	不改变
结果低字寄存器	RESLO	读/写	013A	未定义
结果高字寄存器	RESHI	读/写	013C	未定义
结果扩展寄存器	SUMEXT	只读	013E	未定义

两个操作数 OP1 和 OP2 有各自的寄存器，其中 OP1 用 4 种不同寻址方式访问同一个寄存器。不同的寻址信息经解码定义了执行 4 种乘法操作，结果保存于两个字寄存器中，分别为结果高字 RESHI 和结果低字 RESLO。结果扩展寄存器 SUMEXT 保存运算结果的符号或溢出。

所有寄存器的最低有效位为 b0，最高有效位为 b7(字节数据)或 b15(字数据)。

由于硬件乘法器的乘法运算十分快速且不需要中断介入，因此无 SFR 位。

6.4　硬件乘法器的软件限制

使用硬件乘法器时，要注意以下两种情况产生的限制，即

- 用间接或间接增量寻址模式处理结果。

中断发生在第一操作数 OP1 传入硬件乘法器后

第一操作数 OP1 的地址的最低 2 位有效位决定了乘法的类型，而这个信息无法用任何后来的操作恢复。因此在最初两步操作，即传送第一及第二操作数给乘法器之间，不允许接受中断请求。

中断发生在第二操作数 OP2 传入硬件乘法器后

经过最初两步，乘法结果已经被存放在相应的寄存器 RESLO、RESHI 和 SUMEXT 中，可在中断服务程序中加以保存，例如可以暂存在堆栈中(PUSH)并在做完另一次乘法后将其恢复(POP)。但是这样将使中断程序为此增加代码和运行周期。如果在进行乘法操作前关中断(DINT)并在乘法结束后开中断(EINT)，就可以避免上述情况的发生。但是这一方法有个缺点，如果中断事件恰好发生在这段时间内，则紧急中断被抑制的可能大大增加了。

建议采用的设计原则

一般说来，当主程序中已经用了硬件乘法就要避免在中断程序中使用硬件乘法。专门的应用软件、应用软件库或其他所有系统包含的软件都应考虑在内。各种不同方法的讨论都表明，那样应用的结果负面影响多于正面的。遵循原则性建议，使中断程序尽可能简短些是最好的实际方法。

6.4.3 MACS

在 MACS 运算模式中，硬件乘法器不会自动检测是否发生上溢出或下溢出。当有符号乘加运算在累加寄存器中的数据超出了数据的二进制范围时，溢出就会发生。

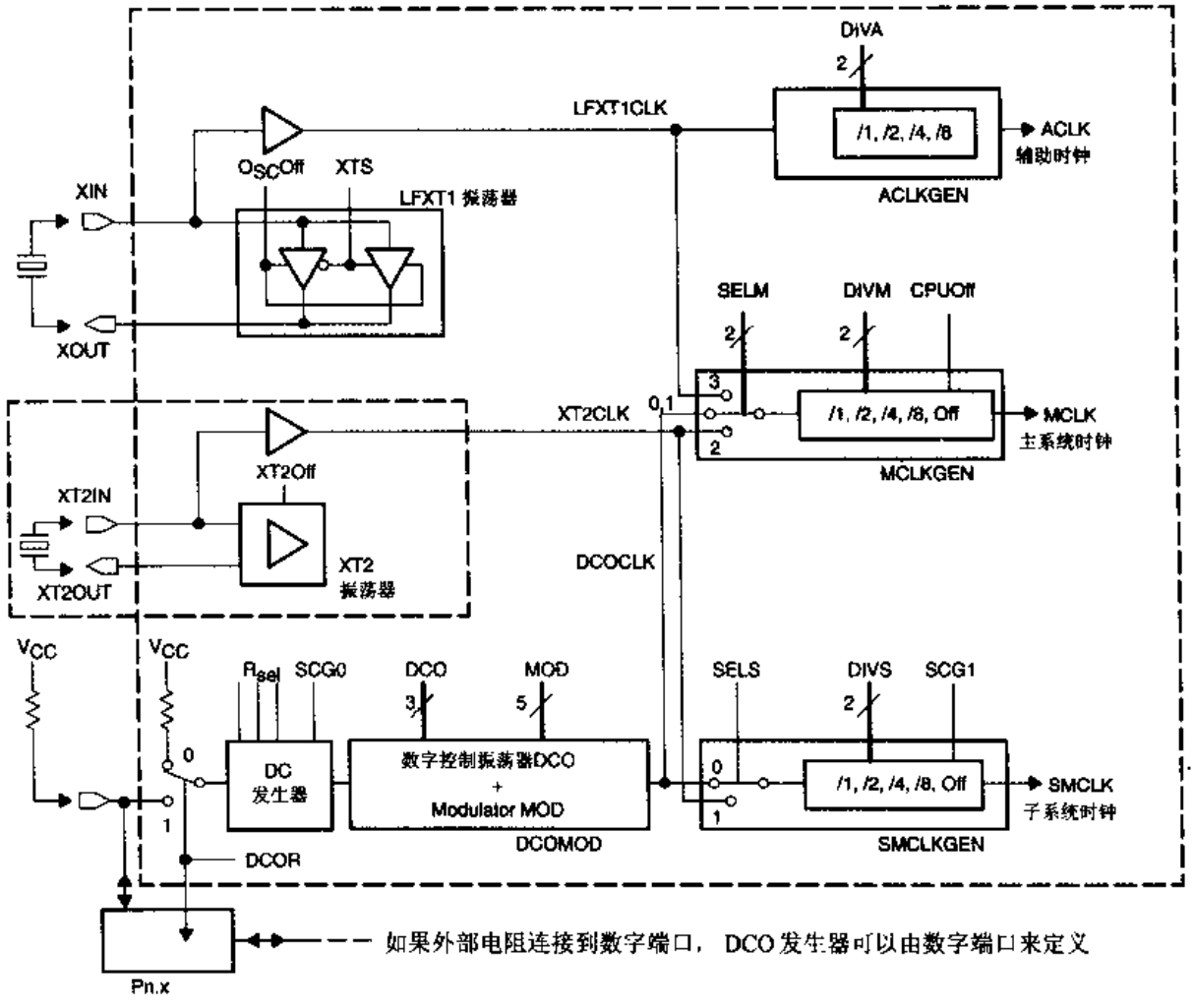
对于正数，累加寄存器的二进制范围是 $0\sim2^{31}-1$(7FFFFFFFh)；对于负数，二进制范围是 $-1\sim-2^{31}$(80000000h)。当两个负数相加产生一个正数范围内的数值时，发生上溢出；当两个正数相加产生一个负数范围内的数值时，发生下溢出。

不至于发生上溢出和下溢出的连续 MACS 操作次数，受到各种不同类型应用问题的限制，可以用发生出错的计算来试探。为找到最大的 MACS 操作次数，应仔细地作运行试验或者加入针对发生溢出的处理方法。

第 7 章　基础时钟模块

7.1　基础时钟模块

基础时钟模块对于达到低廉的系统成本和微弱的系统功耗的设计目标是非常重要的。利用它的 3 个内部时钟信号，开发工程师可以在系统的低功耗和高性能之间得到最佳的平衡。基础时钟模块可以在各种情况下工作：无任何外接元件、无外接电阻、用 1 个或 2 个晶体、用 1 个外接电阻或者是上述各种时钟发生方式的任意组合。基础时钟模块可以作为一个字节模块由 CPU 访问。基础时钟模块的结构见图 7.1。这是针对 MSP430 的 F13x 和 F14x 型的。对于 F11x 和 F11x1 型芯片，基础时钟模块中没有 XT2 振荡器，在模块内的 XT2CLK 信号全部由 LFXT1CLK 信号代替。



注：在 F13、F14 中有 XT2。对于 F11X、F11X、F11X1，是以 LFXT1CLK 代替 XT2CLK 信号。

图 7.1　基础时钟模块

基础时钟模块有 2 个或 3 个时钟源：

- LFXT1CLK 低频/高频时钟源。它可以用低频钟表晶体、标准晶体、陶瓷谐振器或外接时钟源工作。
- XT2CLK 高频时钟源。它可以用标准晶体、陶瓷谐振器或外接 450 kHz～8 MHz 的时钟源工作。
- DCOCLK 时钟源。这是一个可以实现数控(DCO)的 RC 振荡器。

LFXT1 和 XT2 振荡器都需要外接石英晶体，又可以统称为 XT 振荡器。

基础时钟模块可以提供 3 种时钟信号：

- ACLK 辅助时钟。ACLK 是 LFXT1CLK 信号经 1、2、4 或 8 分频后得到的。可用软件选择分频因子。ACLK 可以用软件选作各外围模块的时钟信号。
- MCLK 主时钟。MCLK 可由软件选自 LFXT1CLK、XT2CLK(对于 F13x、F14x)或 DCO 信号，经 1、2、4 或 8 分频后得到。可用软件选择分频因子。MCLK 用于 CPU 和系统。
- SMCLK 子时钟。SMCLK 可由软件选自 LFXT1CLK、XT2CLK(对于 F13x、F14x)或 DCO 信号，经 1、2、4 或 8 分频后得到。可用软件选择分频因子。SMCLK 由软件选择用于各外围模块。

7.2　LFXT1 与 XT2

7.2.1　LFXT1 振荡器

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 中的 OSCOff 位复位，即允许 LFXT1 工作。如果 LFXT1CLK 信号没有用作 SMCLK 或 MCLK 信号，则可以用软件将 OSCOff 位置位来禁止 LFXT1。

LFXT1 振荡器见图 7.2。它被设计成低电流工作并使用 32 768 Hz 的钟表晶振，这时 LFXT1 工作在低频模式，XTS 位应复位。工作时只需将钟表晶振连接到振荡器的两个引脚，不必接其他外部元件。其他的振荡器元件都已集成在芯片内。

LFXT1 振荡器也可以外接高速晶振或陶瓷谐振器工作在高速模式，这时 XTS 位应置位。晶振或陶瓷谐振器连接在振荡引脚，同时每个引脚还要外接电容，容量按晶振或陶瓷谐振器的特性来选择。

图 7.3 是 LFXT1 振荡器的控制逻辑。其中控制信号对 F11x 与 F11x1 系列和 F13x 与 F14x 系列稍有差异。内部控制信号 XT2 对于这两个系列分别是“0”和“1”。对子 F11x 和 F11x1 系列，当 SELM＝3 时，LFXT1CLK 用作 MCLK 时钟；而对于 F13x 和 F14x 系列，当 SELM＝2 或 3 时，LFXT1CLK 用作 MCLK 时钟。如果 SCG1 复位，并且 SELS 置位，则 LFXT1CLK 用作 SMCLK 时钟。

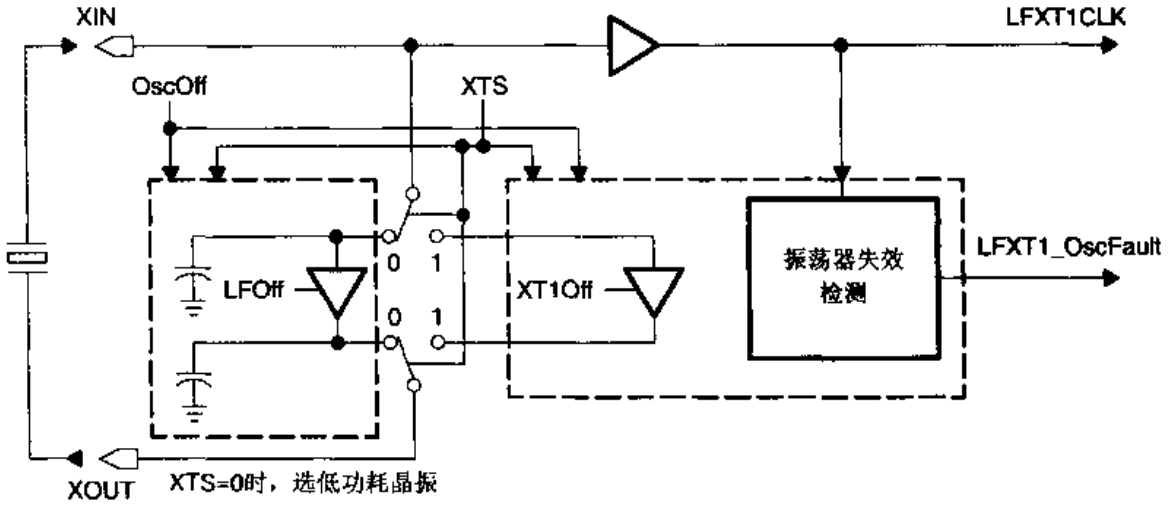


图 7.2 LFXT1 振荡器

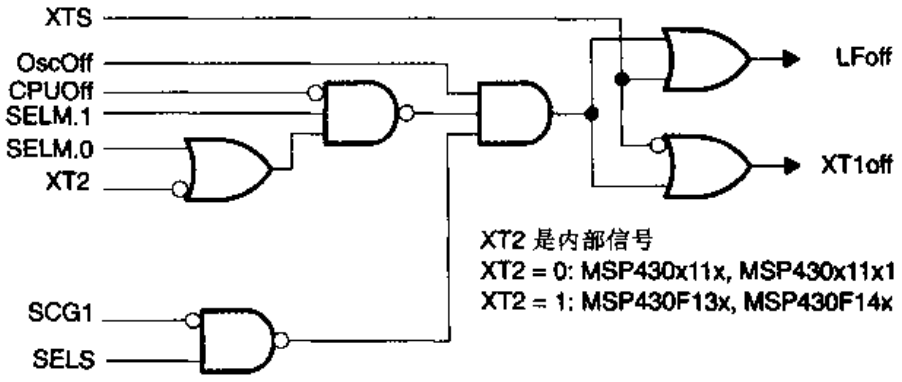


图 7.3 LFXT1 振荡器的控制

7.2.2 XT2 振荡器

对于 F13x 和 F14x，片内有第二个振荡器 XT2。它产生时钟信号 XT2CLK，它的工作特性与 LFXT1 振荡器工作在高频模式时类似。

图 7.4 是 XT2 振荡器的控制逻辑。如果 XT2CLK 信号未用作 MCLK 及 SMCLK 时钟信号，则 XT2Off 控制位可以关闭 XT2；如果 CPUOff 位复位，SELM＝2，则 XT2CLK 用作 MCLK 时钟；如果 SCG1 复位且 SELS 置位，则 XT2CLK 用作 SMCLK 时钟。

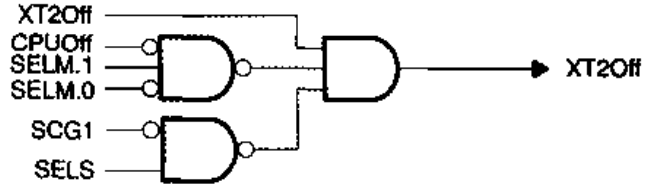


图 7.4 XT2 振荡器的控制

之间的时间延迟关系。当振荡器停振后重新起振时，XT_OscFault 信号仍会保持一段时间的有效，对于 1 MHz 时钟，这个过程大约也需要 50 个振荡周期。

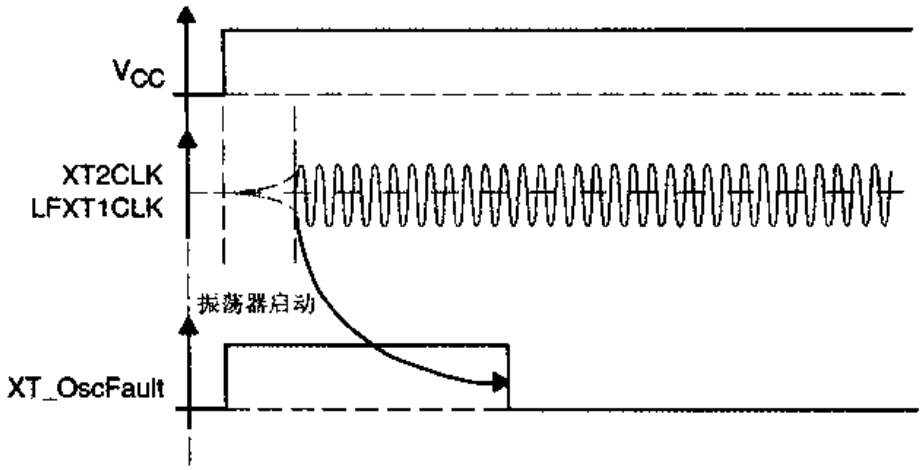


图 7.6 振荡器上电过程

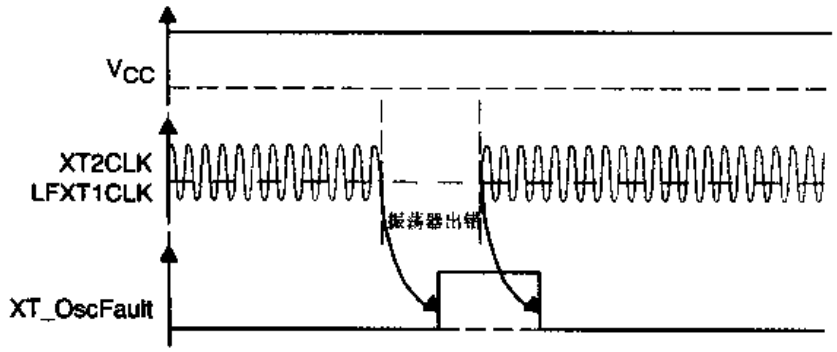


图 7.7 振荡器失效标志与振荡信号的时间延迟

7.2.4 XT 振荡器失效时的 DCO

如果 MCLK 信号来自 LFXT1(只能在高频模式)或 XT2，则当 XT 振荡器失效时 DCO 振荡器会自动被选作产生 MCLK 的源。由于 DCO 振荡器被选为 MCLK，因为 XT 振荡器失效而引起的 NMI 中断请求可以得到响应，这个 NMI 中断的请求甚至在 CPU 被关闭(CPUOff=1)的情况下也能处理。MSP430 可以让任一被允许的中断请求在低功耗模式下得到服务，甚至是在 LPM4 模式下。MCLK 信号在中断服务程序内将自动变为有效。

在编制需要处理时钟失效的 NMI 中断服务程序时，为了正确处理时钟失效情况，必须有几个重要阶段。首先是 NMI 检测到 XT 振荡器失效，选择 DCO 作为 MCLK 的源；然后由用户编制适当的算法来检测 XT 振荡器是否重新工作，并在必要时再次选择 XT 振荡器信号作为系统时钟 MCLK 的源。图 7.8 是时钟失效 NMI 中断服务程序的处理流程。

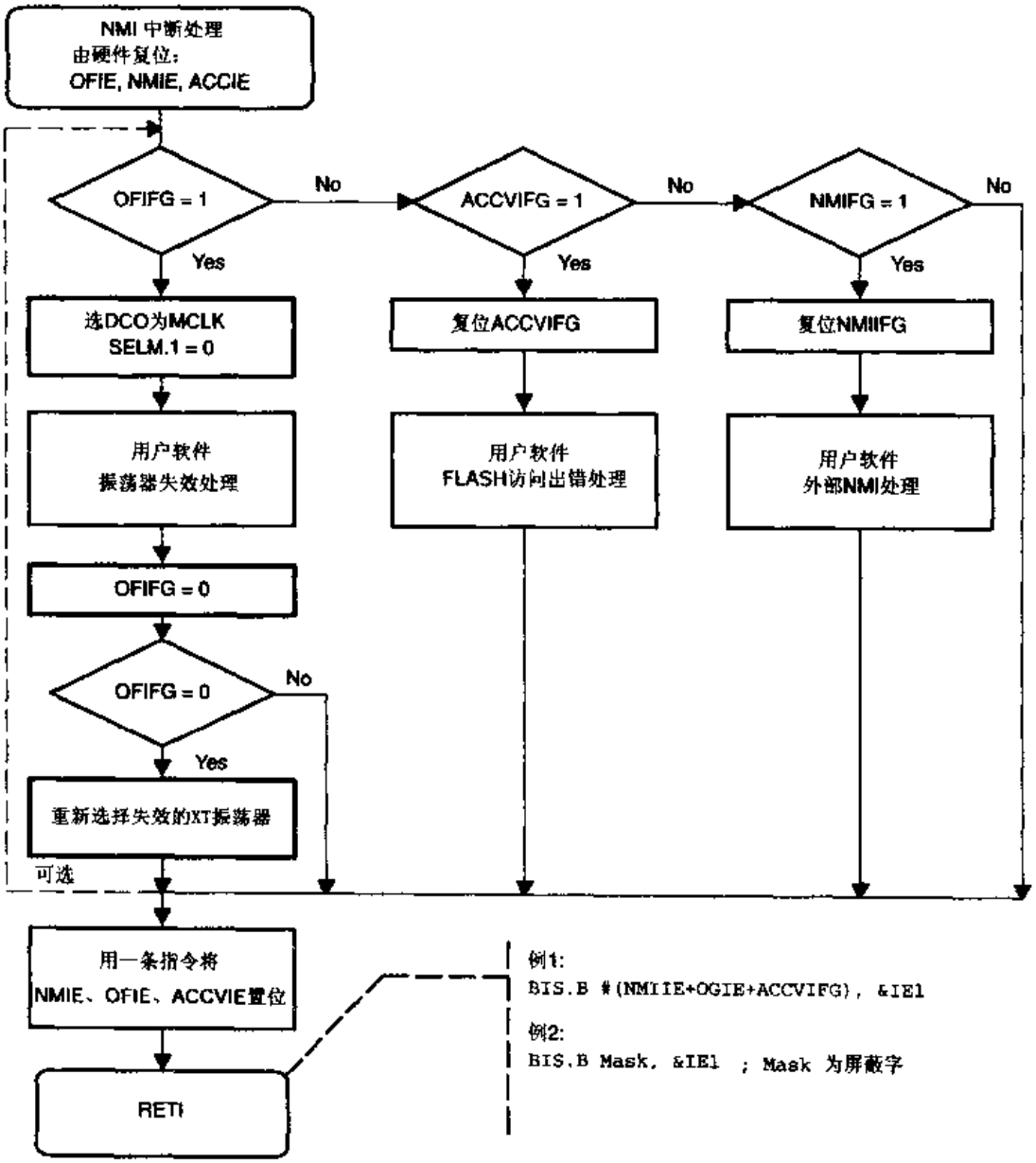


图 7.8　时钟失效 NMI 中断流程

7.3　DCO 振荡器

7.3.1　DCO 振荡器的特性

集成在基础时钟模块中的 DCO 振荡器是一个 RC 振荡器。DCO 频率可以通过 DCO、MOD 和 RSEL 位用软件调节。图 7.9 是 DCO 的控制逻辑。DCO 的调节是单调的。RC 振荡器的频率会因为温度、工作电压的变化而变化，同一型号芯片之间产生的频率也不一样。DCO 的调节功能增加了振荡频率的稳定性，改善了 RC 振荡器的特性。

当关闭 DC 发生器时，由于它的微弱电流特性，需要大约 4 μs 的建立时间。当电流接通时，DC 发生器的电阻需要微安级的注入电流，各种内部和外部的寄生电容会引入微秒级的时间延迟。当工作模式不需要关断 DC 发生器的电流时，就不会引入时间延迟。

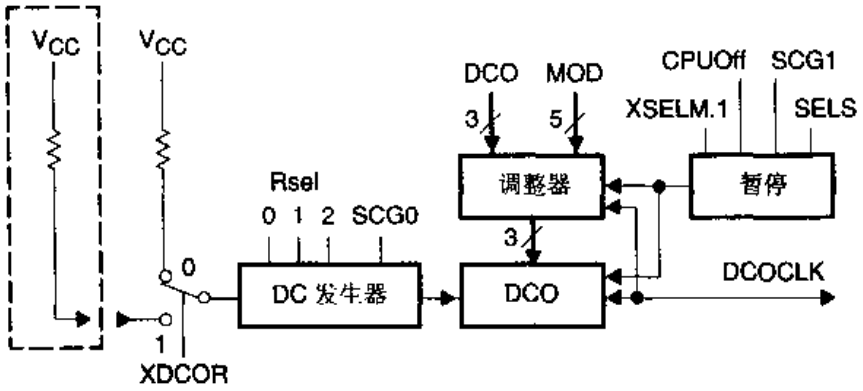


图 7.9 DCO 的控制

可以将一个内部或外部的电阻接到 DC 发生器，它的阻值决定了 DCOCLK 的基础工作频率。

DCLCLK 的频率由下列功能建立：

- 由内部或外部电阻向 DC 发生器(DCGEN)注入电流来决定 DCO 的基础频率，内部电阻或外部电阻的接入由 DCOR 控制位选择。
- 控制位 RSEL2、RSEL1、RSEL0 决定将基础频率分频为 8 个标称频率范围，这些频率范围在各个不同型号芯片的技术手册中说明。
- 由 3 个控制位 DCO0～DCO2 分段调节 DCOCLK 的频率。
- 由 5 个调整位 MOD0～MOD4 控制切换 DCO 和 DCO+1 选择的两种频率。

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。

图 7.10 是 DCO 的分段频率控制关系。

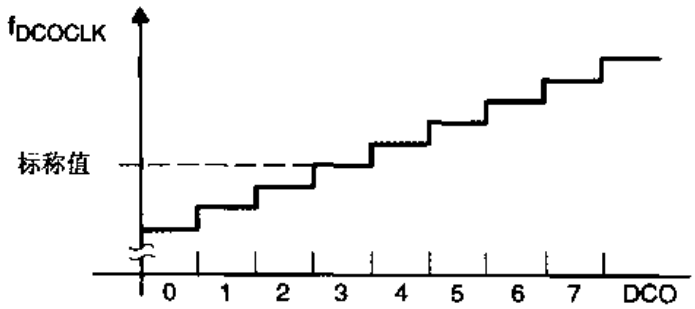


图 7.10 DCO 分段频率控制

7.3.2 DCO 调整器

DCO 调整器的作用是希望通过混合相邻的 DCO 周期来减少长的累积周期变化，通过混合 DCO 周期可以将累积周期变化减至最少。调整器的累积以 32 个 DCOCLK 时钟周期为一周期。图 7.11 是 DCO 的控制逻辑信号。MOD 控制位定义了需要混合的 DCO+1 周期的比例常数，在其后的 32 个时间片内插入 DCO 周期，如果调整常数为“0”，控制寄存器的 DCO 数据定义了振荡器周期。

下列公式定义了累计周期

$$T = (32 - MOD) \times T_{DCO} + MOD \times T_{DCO+1}$$

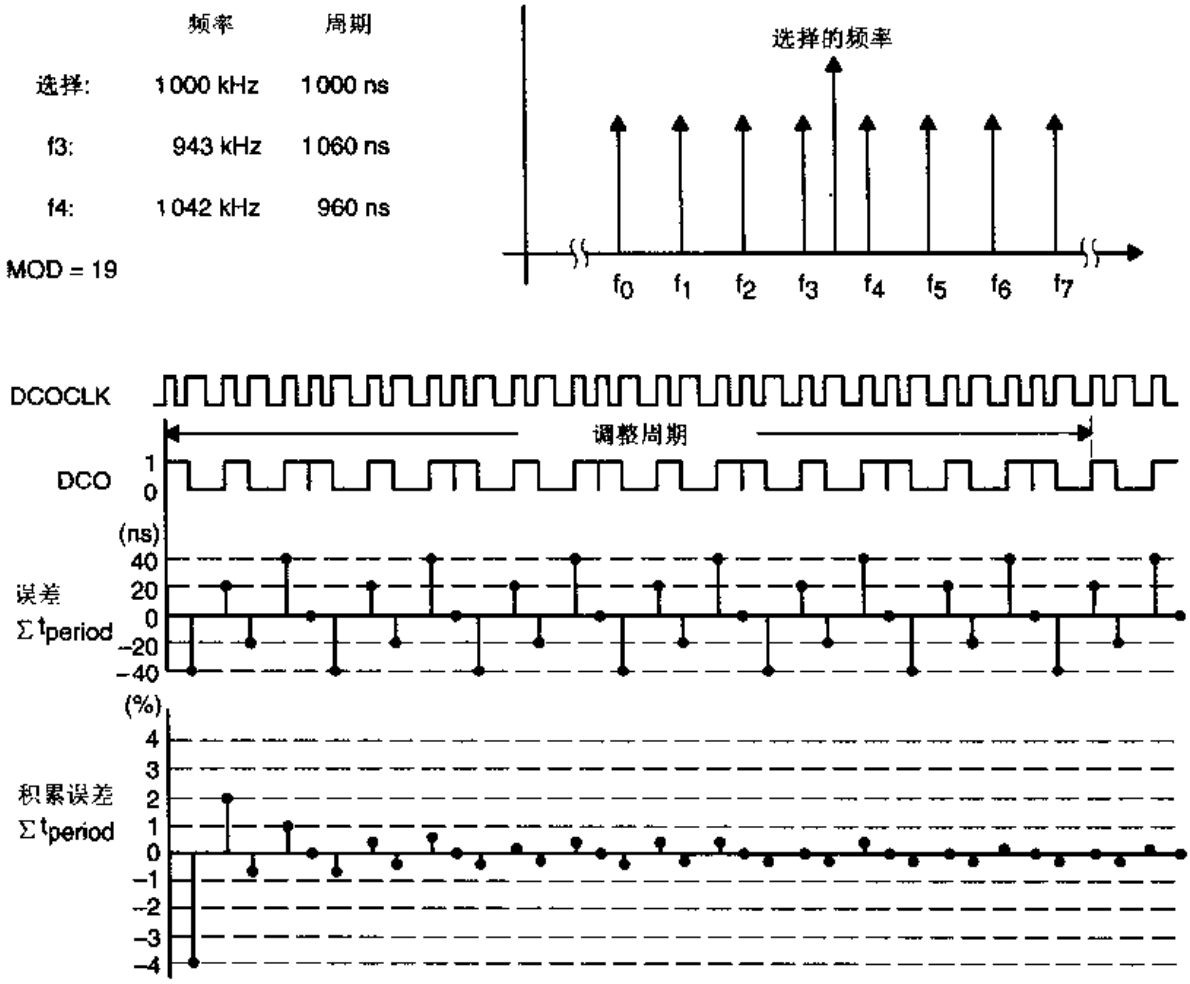


图 7.12 DCO 调整器的工作原理

7.4 时钟与运行模式

当 DCO 振荡器没有用于 MCLK 或 SMCLK 时，它就关闭了。DC 发生器需要单独去关闭，但是一旦 DCOCLK 信号被用于 MCLK 或 SMCLK，它就自动开启。

7.4.1 由 PUC 启动

当发生一次有效的 PUC 时，DC 发生器会选择内部电阻，RSEL＝4，DCO＝3，这样就使 DCO 振荡器工作在一个与外界条件无关的适中的频率上。ACLK 由用钟表晶振工作在低频模式的 LFXT1 振荡器来提供，MCLK 与 SMCLK 来自 DCO。CPU 在执行程序时所需要的 MCLK 来自快速启动工作的 DCO，因此自 PUC 至程序开始非常快捷，典型值少于 6 μs。

7.4.2 基础时钟调整

基础时钟的控制寄存器完全由软件控制。如果系统对时钟的要求不同于 PUC 发生后的缺省值，则可用软件在程序执行的任意时刻对基础时钟进行配置，可以配置多次。

- LFXT1 振荡器外接晶振、陶瓷谐振器工作或者外部时钟信号再经 1、2、4、8 分频产生

ACLK 信号，如果不需要 LFXT1 时钟，则应将 SR 中的 OSCOff 位置位。

- LFXT1 或 XT2(仅 F13x 与 F14x)振荡器外接晶振、陶瓷谐振器工作或者 DCOCLK 时钟信号再经 1、2、4、8 分频产生 SMCLK 信号，SR 中的 SCG1 位实现对 SMCLK 信号的允许或禁止。
- LFXT1 或 XT2(仅 F13x 与 F14x)振荡器外接晶振、陶瓷谐振器工作或者 DCOCLK 时钟信号再经 1、2、4、8 分频产生 SMCLK 信号，SR 中的 CPUOff 位实现对 MCLK 信号的允许或禁止。
- DCOCLK 频率用 RSEL、DCO、MOD 位调节，DCOCLK 一旦不用就停止工作。DC 发生器可以用 SR 中的 SCG0 位允许或禁止。
- XT2 振荡器的 XT2CLK 信号用 XT2Off 位清除。

用户软件可以在任何时刻用 MSP430 各种指令来调整基础时钟以适应系统的需要。以下是一些例子：

```
BIS. B    #007h,&BCSCTL1        ; RSEL = 7
MOV. B    #081h,&BCSCTL1        ; XT2 关闭,RSEL = 1
BIS. B    #070h,&BCSCTL1        ; ACLK = 高速晶振 / 8
BIS. B    #008h,&BCSCTL2        ; SMCLK = LFXT1
INC. B    &DCOCTL               ; DCOCLK 增加
DEC. B    &DCOCTL               ; DCOCLK 减少
```

7.4.3　用于低功耗的基础时钟特性

电池供电的系统常常存在两个矛盾的要求：

- 采用低频时钟以达到节能和延长电池使用时间的要求。
- 采用高频时钟以实现对事件的快速反应和处理突发事务的能力。

基础时钟模块解决上述矛盾的方法是让设计工程师选择 3 个时钟来工作：ACLK、MCLK 和 SMCLK。为优化低功耗特性，ACLK 可以用 32 768 Hz 钟表晶振来产生，以提供稳定的系统时基和低功耗的备用工作状态。MCLK 可以来自片上的 DCO 振荡器，它只是在发生请求服务的事件时才需要。SMCLK 可以来自晶振或 DCO，取决于外围模块的要求。灵活的时钟配置及分频系统可以让设计者适应各种对时钟的要求。基础时钟模块的配置可以完全用软件来控制。

7.4.4　选择晶振产生 MCLK

上电后基础时钟模块用 DCO 时钟产生系统时钟 MCLK。XTS=0，LFXT1 振荡器工作在低频模式。不管应用对时钟系统的要求如何，一旦完成全部初始化，软件运行就由片内的 DCO 来保证。当然，经过软件配置应用软件的执行可以改用晶振时钟。图 7.13 是 MCLK 源自 LFXT1 的实例。

在实例中，MCLK 的源由 DCO 时钟切换到 LFXT1 时钟将按如下步骤完成，即

(1) 启动晶振：

a) OSCOff = 0，XTS 复位(选择低频模式)或置位(选择高频模式)。

```
BIS. B    #XTS,&BCSCTL1         ; XTS 置位选择 LFXT1
BIC       #OSCOff,SR            ; 启动 LFXT1 振荡器
```

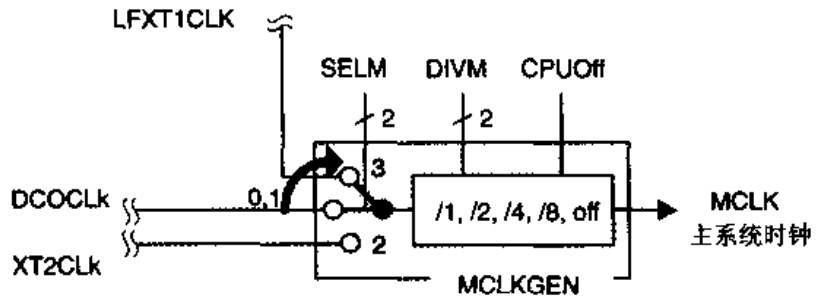


图 7.13 LFXT1 用作 MCLK

b) 对于 F13x 和 F14x,将 XT2Off 复位将选择 XT2 振荡器。

```
        BIC.B   #XT2Off,&BCSCTL1            ; XT2Off 复位,启动 XT2 振荡器
```

(2) 等待 MCLK 所用振荡器完成启动,有 3 种可能:

a) 总是等待足够的时间,使振荡器在任何情况下都能完成设置,由于不同的振荡器工作方式(钟表晶振、标准晶振、陶瓷谐振器)所需时间不同,应用软件要对此作估计。

```
        PUSH    #Wait                       ; 等待时间常数装入堆栈
Loop    DEC     0(SP)                       ; 计算等待时间至"0"
        JNZ     Loop                        ; 等待循环,等待时间=t(MCLK)×(4+2)
        BIC.B   #OFIFG,&IFG1                ; 清除振荡器失效中断标志
        BIS.B   #(SELM1+SELM0),&BCSCTL2     ; 选择 LFXT1
```

b) 用检测振荡器失效(复位 OFTFG,测试 OFIFG=0)来等待振荡器启动。然后等待足够的时间以保证振荡器的幅度足够大。

```
TST_OF  BIC.B   #OFIFG,&IFG1                ; 清除振荡器失效中断标志
Loop    DEC     0(SP)                       ; 计算等待时间至"0"
        JNZ     Loop                        ; 等待循环
        BIT.B   #OFIFG,&IFG1                ; 测试振荡器失效中断标志
        JNZ     TST_OF                      ; 重复测试,直至标志始终复位
        PUSH    #Wait                       ; 等待时间常数装入堆栈
        BIC.B   #OFIFG,&IFG1                ; 清除振荡器失效中断标志
        BIS.B   #(SELM1+SELM0),&BCSCTL2     ; 选择 LFXT1
```

c) 用检测振荡器失效(复位 OFTFG,测试 OFIFG=0)来等待振荡器启动。然后计数时钟脉冲,一般计数 1 024 个脉冲并校验最终的应用状态。

```
TST_OF  BIC.B   #OFIFG,&IFG1                ; 清除振荡器失效中断标志
        BIT_B   #OFIFG,&IFG1                ; 测试振荡器失效中断标志
        JNZ     TST_OF                      ; 重复测试,直至标志始终复位
; 看门狗定时器用 ACLK/2,SSEL=1,IS1=1,IS0=0
; RST/NMI 引脚为复位功能,看门狗定时器为定时器功能,并清除计数
        BIS.B   #DIVA0,&BCSCTL1             ; 设 DIVA.1 位复位
        BIC.B   #WDTIE,&IE1                 ; 看门狗定时器不产生中断
        BIC.B   #WDTIFG,&IFG1               ; 清除 WDTIFG,它表示计数 1 024 脉冲
        MOV     #(WDTPW+WDTTMSEL+WDTCNTCL+WDTSSEL+WDTIS1),
```

```
                &WDTCTL
Retest  BIT.B   #WDTIFG,&IFG1          ;测试是否到 1 024 脉冲
        JNC     Retest
        BIC.B   #OFIFG,&IFG1           ;清除振荡器失效中断标志
        BIS.B   #(SELM1+SELM0),&BCSCTL2 ;选择 LFXT1
```

图 7.14 是用高频模式的 LFXT1 作为 MCLK 的时序实例。

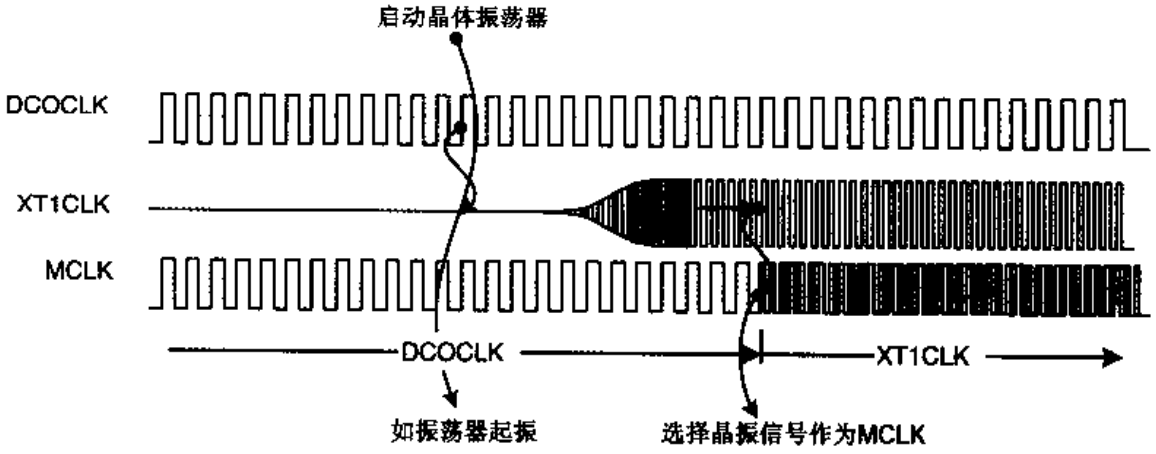


图 7.14　高频模式 LFXT1 用作 MCLK

7.4.5　时钟信号的同步

MCLK 和 SMCLK 时钟可以由不同的时钟源提供。当从一个时钟源切换到另一个时，切换必须同步以避免发生时间竞争。当选择一个新的时钟源时，会发生以下过程：

(1) 继续当前的时钟周期，直至出现上升沿。

(2) 时钟保持为高，直至出现新选择时钟源信号的上升沿。

(3) 新时钟源完成选择，继续完成时钟周期的高半周。

图 7.15 是上述过程的时序实例。

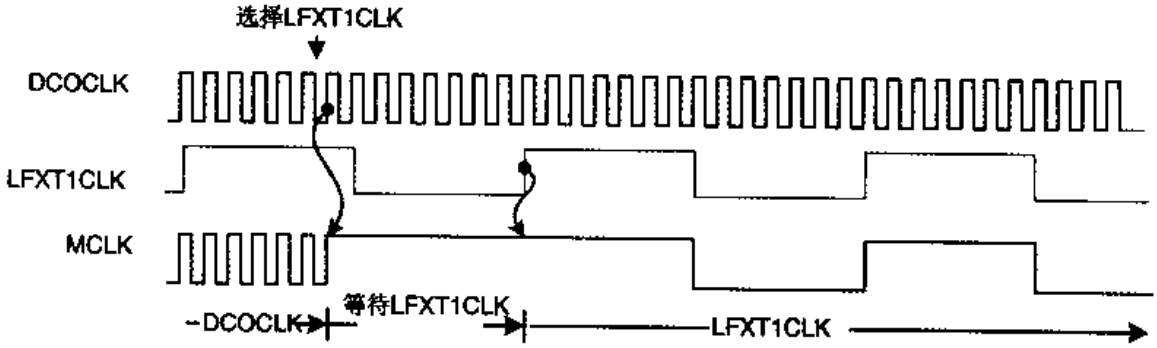


图 7.15　MCLK 由选择 DCO 到选择 LFXT1

7.5　基础时钟模块控制寄存器

基础时钟模块由寄存器 DCOCTL、BCSCTL1、BCSCTL2 和状态寄存器 SR 中的控制位 SCG1、SCG0、OSCOff、CPUOff 来控制。用户软件可以在任何时候修改控制寄存器的缺省值。

基础时钟模块的控制寄存器位于字节模块中，只能用字节指令访问。表 7.1 是基础时钟模块的寄存器一览表。

表 7.1 基础时钟模块控制寄存器

寄存器	简 称	寄存器类型	地 址	初始状态
DCO 控制	DCOCTL	读/写	056h	060h
基础时钟控制 1	BCSCTL1	读/写	057h	084h
基础时钟控制 1	BCSCTL2	读/写	058h	0

7.5.1 DCO 时钟频率控制

DCOCTL 在发生 PUC 后装入 060h。

COCTL(056h)

7							0
DCO. 2	DCO. 1	DCO. 0	MOD. 4	MOD. 3	MOD. 2	MOD. 1	MOD. 0
rw-0	rw-1	rw-1	rw-0	rw-0	rw-0	rw-0	rw-0

b0～4：MOD. 0～MOD. 4，MOD 常数定义了在 32 个 DCO 周期中插入的 f_{DCO+1} 周期。在余下的 DCO 周期中为 f_{DCO} 周期。如 DCO 常数为 7，由于已选择了最高频率无法进行调整。

b5～7：DCO. 0～DCO. 2，DCO 常数定义了选择 8 种频率之一。频率由注入 DC 发生器的电流定义。

7.5.2 振荡器与时钟控制寄存器

BCSCTL1 在发生 PUC 或 POR 后改变。

BCSCTL1(057h)

7							0
XT2Off	XTS	DIVA. 1	DIVA. 0	XT5V	RSEL. 2	RSEL. 1	RSEL. 0
rw-(1)	rw-(0)	rw-(0)	rw-(0)	rw-0	rw-1	rw-0	rw-0

b0～2：RSEL. 0～RSEL. 2，由内部电阻选择 8 个频率之一，阻值定义了标称频率，RSEL=0 时选择最低的标称频率。

b3：XT5V，必须复位。

b4～5：DIVA. 0、DIVA. 1，选择 ACLK 的分频数。

DIVA=0： 1

DIVA=1： 2

DIVA=2： 4

DIVA=3： 8

b6：XTS，选择 LFXT1 工作在低频晶振模式或高频晶振模式，必须与外接晶振符合。

XTS=0： 低频模式；

XTS=1： 高频模式。

第 8 章　输入输出端口

8.1　引　言

MSP430 通用 I/O 端口的设计提供了最大的灵活性。每一端口的所有引脚可以单独配置，大部分具有中断能力。

MSP430 的 FLASH 型芯片有 2 类 I/O 端口。P1 和 P2 是一类，P3、P4、P5 和 P6 是另一类。这两类 I/O 端口都具有控制 I/O 方向、输出、输入的能力。P1、P2 具有中断能力，每一引脚都可以单独选择中断触发沿、单独允许中断。

F11x、F11x1 只有 P1 和 P2。由于封装的原因，P2 只有 6 个引脚，但是内部逻辑是完整的。

F13x、F14x 有全部的 P1～P6 端口。

8.2　端口 P1、P2

图 8.1 是 P1、P2 的结构。

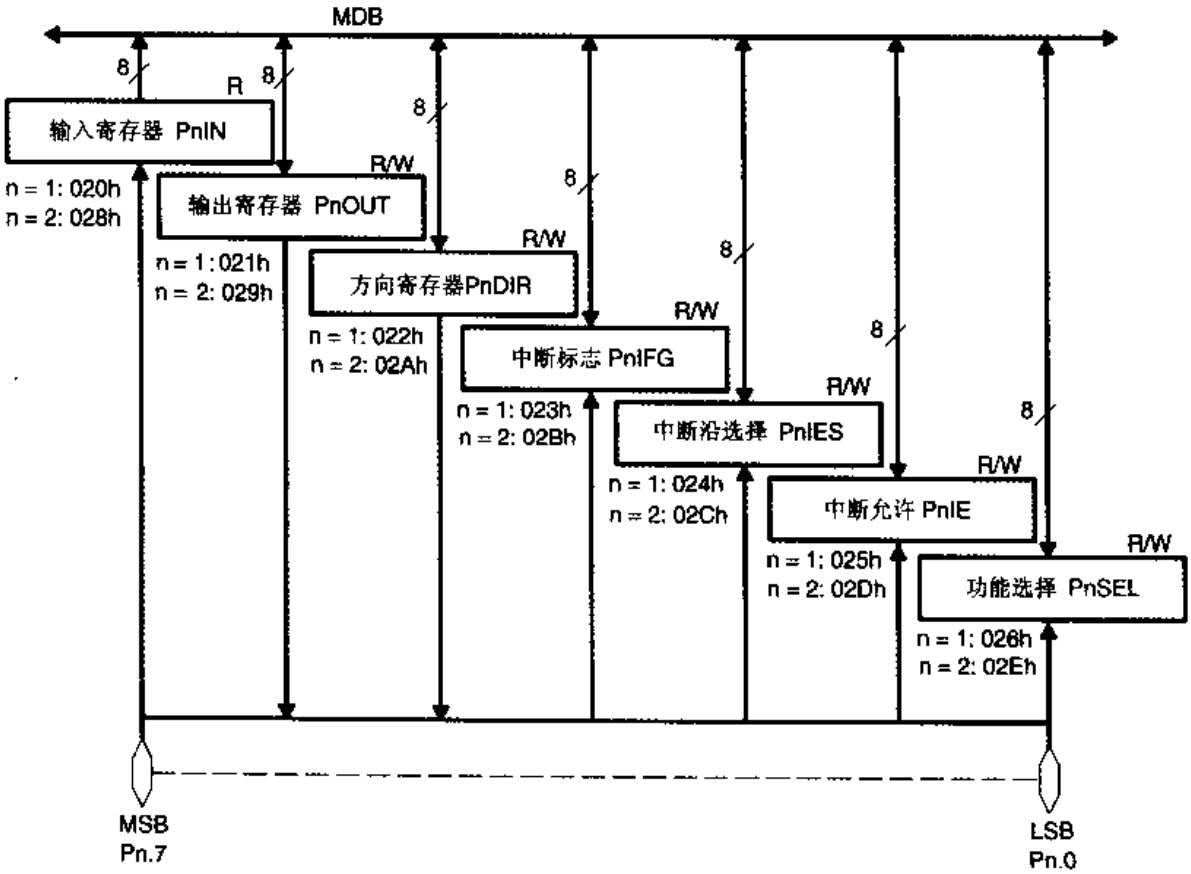


图 8.1　P1、P2 结构

通用端口 P1 和 P2 各有 8 个引脚及各自的控制寄存器，每一引脚可以单独控制，并且每一引脚都可用作中断源。

P1、P2 各有一个中断向量。P1.0～P1.7 产生同一个中断，P2.0～P2.7 也产生同一个中断。

它们各有 7 个寄存器用来控制端口的引脚。

P1、P2 经过 8 位 MDB 和 MAB 与处理内核连接，它们必须用字节指令以绝对寻址模式进行访问。

8.2.1　P1、P2 的控制寄存器

7 个控制寄存器为 I/O 功能的配置提供了最大的灵活性，即

- 所有 I/O 位均可独立编程。
- 可以有各种输入、输出和中断条件的组合。
- P1 和 P2 的 8 位全部可用于对外部事件的中断处理。

P1 和 P2 的 7 个寄存器分别见表 8.1 和表 8.2。

表 8.1　P1 寄存器

寄存器	简　称	寄存器类型	地　址	初始状态
输入寄存器	P1IN	只读	020h	
输出寄存器	P1OUT	读/写	021h	不变
方向寄存器	P1DIR	读/写	022h	复位
中断标志位	P1IFG	读/写	023h	复位
中断触发沿选择	P1IES	读/写	024h	不变
中断允许	P1IE	读/写	025h	复位
功能选择寄存器	P1SEL	读/写	026h	复位

表 8.2　P2 寄存器

寄存器	简　称	寄存器类型	地　址	初始状态
输入寄存器	P2IN	只读	028h	
输出寄存器	P2OUT	读/写	029h	不变
方向寄存器	P2DIR	读/写	02Ah	复位
中断标志位	P2IFG	读/写	02Bh	复位
中断触发沿选择	P2IES	读/写	02Ch	不变
中断允许	P2IE	读/写	02Dh	复位
功能选择寄存器	P2SEL	读/写	02Eh	复位

以上所有寄存器为 8 位，必须用字节指令以绝对寻址模式进行访问。

输入寄存器 P1IN、P2IN

这两个只读寄存器反映 I/O 引脚上的信号。

> **注意：对输入寄存器 P1IN、P2IN 写入**
>
> 对只读寄存器的写入会在写有效期间增加电流消耗。

输出寄存器 P1OUT、P2OUT

如果 I/O 引脚用作输出，则 8 位的输出寄存器提供相应位的输出缓存信息。输出缓存可用所有含目的操作数的指令修改。在读取时，输出缓存的内容和引脚定义方向无关，改变方向

不会改变输出缓存的内容。

方向寄存器 P1DIR 和 P2DIR

每个寄存器含有相互独立的 8 位，用于定义每个 I/O 引脚的方向。各位在 PUC 后复位。

0：I/O 引脚切换成输入模式；

1：I/O 引脚切换成输出模式。

中断标志寄存器 P1IFG 和 P2IFG

每个寄存器有对应于 I/O 引脚的 8 个标志位，表示是否有中断挂起。

0：没有待处理的中断；

1：由于 I/O 引脚电平跳变或软件对这一位的设置会产生待处理的中断。

注意：P1IFG 与 P2IFG

对寄存器 P1OUT 和 P1DIR 及对 P2OUT 和 P2DIR 的操作可能使 P1IFG 或 P2IFG 中相应位置位。

对某一个中断标志写“0”使它复位，写“1”使它置位并产生中断。

每一组中断标志 P1IFG.0～P1IFG.7 和 P2IFG.0～P2IFG.7 只用一个中断向量，它们都是多源中断向量。当中断被接受时这些标志位不会自动复位，要由中断服务程序来确定中断源，并将相应的标志复位。

注意：中断事件宽度

任何外部中断事件必须长于 1.5 倍 MCLK 周期，以保证被接受并使相应中断标志置位。

中断触发沿选择寄存器 P1IES 和 P2IES

中断触发沿选择寄存器中对应于每一个 I/O 引脚都有 1 位选择触发中断标志的电平跳变。意义如下：

0：电平由低到高跳变使标志置位；

1：电平由高到低跳变使标志置位。

注意：P1IES、P2IES 各位的改变

改变 P1IES 或 P2IES 中的位可能会引起相应中断标志位置位，与引脚电平有关。

PnIES.x	PnIN.x	PnIFG.x
0 → 1	0	不变
0 → 1	1	可能置位
1 → 0	0	可能置位
1 → 0	1	不变

中断允许寄存器 P1IE 和 P2IE

中断允许寄存器中对应每一个 I/O 引脚都有一相应的允许位，可允许或禁止中断。意义如下：

0：禁止中断请求；

1：允许中断请求。

注意：P1、P2 的中断触发

只有跳变才能引起中断，静态电平不能。如果在执行 RETI 指令后中断标志仍然置位（例如在中断服务期间又发生跳变），则在 RETI 指令完成后中断将再次发生。这一点保证了每次跳变都能得到软件响应。

功能选择寄存器 P1SEL 和 P2SEL

为了减少 MSP430 的引脚，P1、P2 引脚与外围模块的引脚同用。P1SEL 与 P2SEL 用来选择引脚的 I/O 端口功能或者外围模块功能。每个寄存器都有相互独立的 8 位来定义访问 I/O 引脚的功能为端口功能或者外围模块功能，各位在 PUC 后复位，意义如下：

0：端口功能；

1：外围模块功能。

注意：用 P1SEL、P2SEL 选择功能

如果 PnSEL. x 置位，则中断触发沿选择电路被禁止，输入信号将不会产生中断。

当端口功能选择为外围模块的输入时，输入到外围模块的信号是锁存的。锁存器用 PnSEL. x 位作使能信号，当 PnSEL. x=1 时，外围模块输入信号跟随引脚信号；当PnSEL. x=0 时，作为外围模块输入信号的锁存器输出信号实际上是 PnSEL. x 复位时的引脚信号。

8.2.2　P1、P2 的原理

端口 P1 和 P2 各引脚的逻辑是相同的，每一位都可读写。图 8.2 是 P1、P2 的内部逻辑。

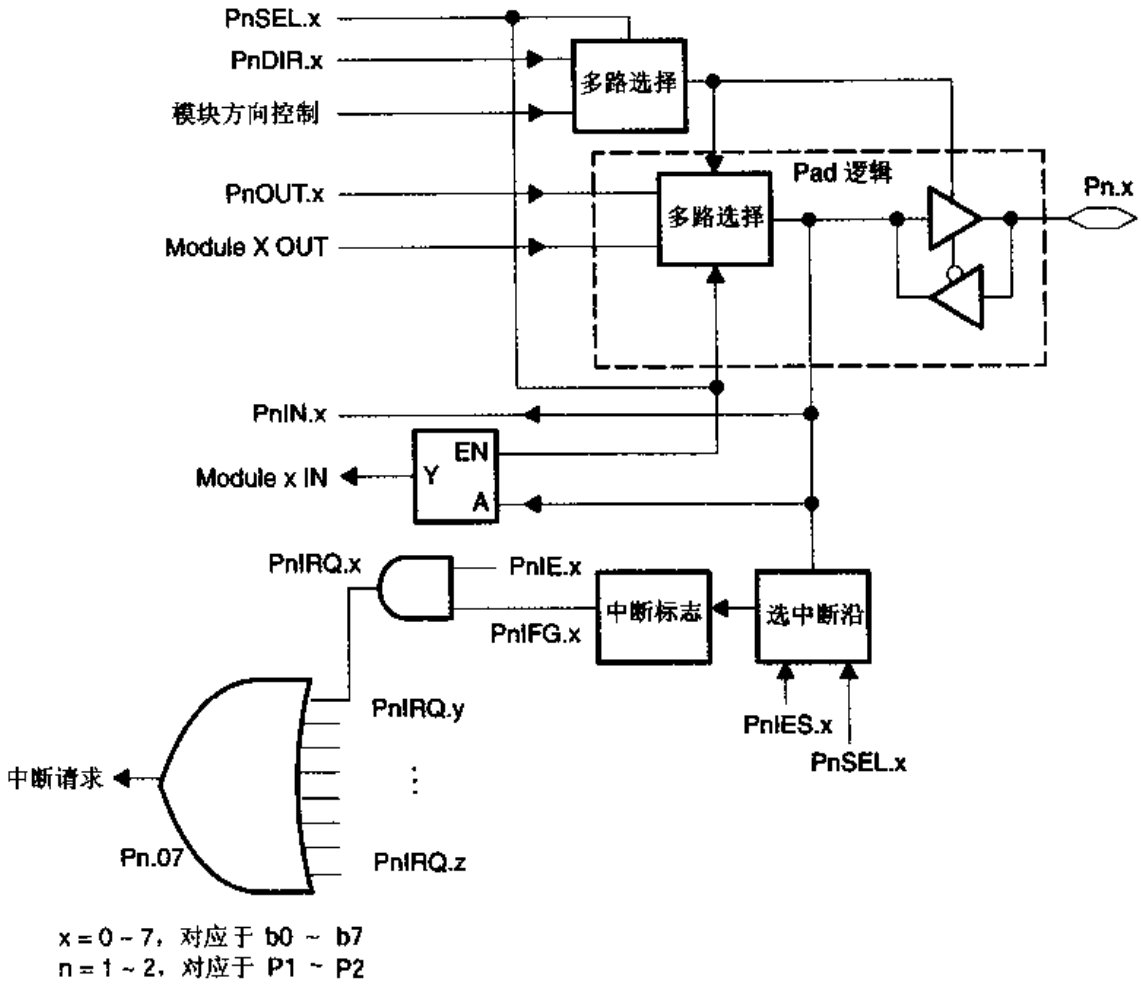


图 8.2　P1、P2 内部逻辑

8.2.3 P1、P2 的中断控制功能

端口 P1 和 P2 有 8 个中断标志位、8 个中断允许位、8 个中断事件触发沿选择位，并各有一个中断向量。

每个中断信号各用 3 位来配置中断，即

- 中断标志位：P1IFG.0～P1IFG.7 和 P2IFG.0～P2IFG.7。
- 中断允许位：P1IE.0～P1IE.7 和 P2IE.0～P2IE.7。
- 中断触发沿选择位：P1IES.0～P1IES.7 和 P2IES.0～P2IES.7。

中断标志位 P1IFG.0～P1IFG.7 产生一个中断，P2IFG.0～P2IFG.7 也产生一个中断。当满足两个条件：PnIE.x 置位且通用中断允许位 GIE 置位时，P1.0～P1.7 或 P2.0～P2.7 中的一个或多个中断事件将产生中断请求。中断标志位不会在中断被接受后自动复位。中断服务程序必须进行中断源的检查和将相应的中断标志位复位的处理。

8.3 端口 P3、P4、P5 和 P6

通用端口 P3、P4、P5 和 P6 的结构见图 8.3。每个引脚可以作为 I/O 端口工作，也能是外围模块的引脚。MSP430 这种引脚功能的复合减少了芯片的引脚。

每个端口各有 4 个控制寄存器。

P3、P4、P5 和 P6 经过 8 位 MDB 及 MAB 与处理机内核连接，它们必须用字节指令以绝对寻址模式来访问。

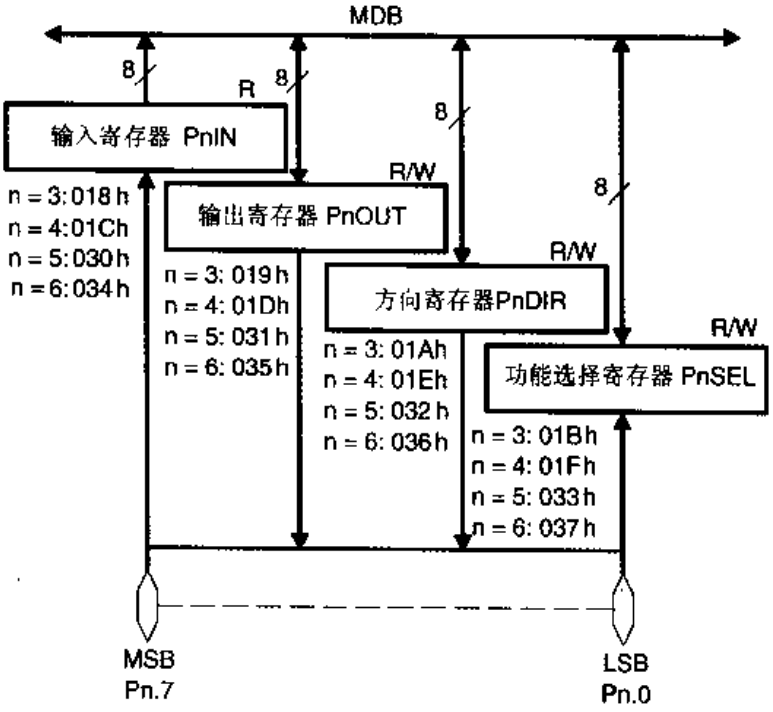


图 8.3 P3、P4、P5 和 P6 的结构

8.3.1　端口 P3、P4、P5 和 P6 的控制寄存器

每个端口有 4 个控制寄存器，为数字 I/O 功能提供了最大的灵活性。

- 所有 I/O 位均可独立编程。
- 可以有各种输入组合。
- 可以任意组合端口功能和模块功能。

每个端口的 4 个寄存器见表 8.3，为字节宽度，必须用字节指令访问。

表 8.3　P3、P4、P5、P6 控制寄存器

寄存器	缩　写	寄存器类型	地　址	初始状态
输入寄存器	P3IN	只读	018h	
	P4IN	只读	01Ch	
	P5IN	只读	030h	
	P6IN	只读	034h	
输出寄存器	P3OUT	读/写	019h	不变
	P4OUT	读/写	01Dh	不变
	P5OUT	读/写	031h	不变
	P6OUT	读/写	035h	不变
方向寄存器	P3DIR	读/写	01Ah	复位
	P4DIR	读/写	01Eh	复位
	P5DIR	读/写	032h	复位
	P6DIR	读/写	036h	复位
端口选择寄存器	P3SEL	读/写	01Bh	复位
	P4SEL	读/写	01Fh	复位
	P5SEL	读/写	033h	复位
	P6SEL	读/写	037h	复位

输入寄存器 PnIN

输入寄存器为只读，反映 I/O 引脚上的信号。

注意：对输入寄存器 PnIN 写入

对只读寄存器的写入会在写有效期间增加电流消耗。

输出寄存器 PnOUT

输出寄存器提供输出缓存的信息。输出缓存可用所有含目的操作数的指令修改。在读取时，输出缓存的内容和引脚定义的方向无关。改变方向定义不会改变输出缓存的内容。

方向寄存器 PnDIR

方向寄存器含有相互独立的 8 位，用于定义 I/O 引脚的方向。各位在 PUC 后复位。

0：I/O 引脚切换成输入模式；

1：I/O 引脚切换成输出模式。

第 9 章　看门狗定时器 WDT

9.1　看门狗定时器

看门狗定时器(WDT)的主要功能是:当程序发生问题时使受控系统重新启动。如果 WDT 超过了选定的定时时间,即发生系统复位。应用中如果不需要此功能,则可以把它当作定时器,当选定定时时间到达后将产生中断。图 9.1 是 WDT 的结构。

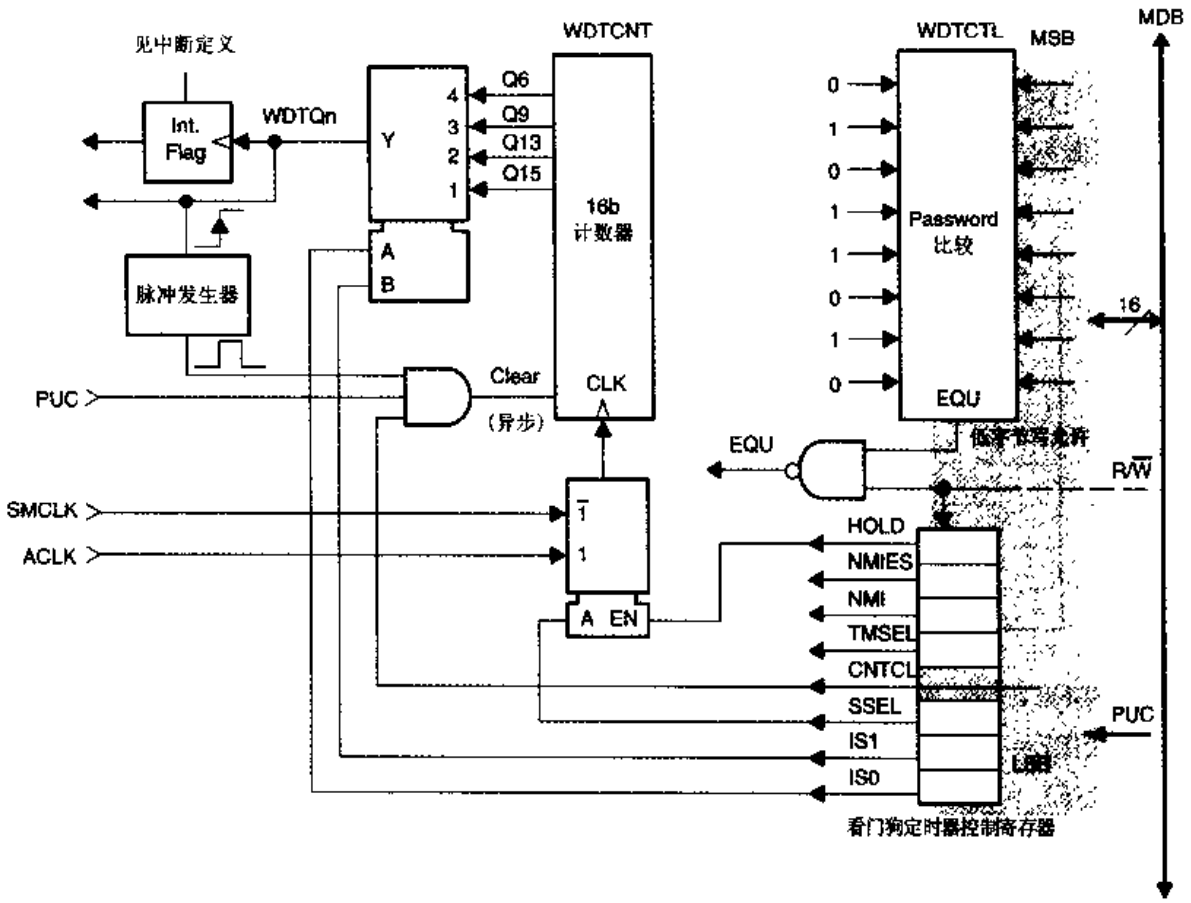


图 9.1　WDT 结构

WDT 特性如下:

- 8 种软件可选的定时时间。
- 2 种工作模式,即:看门狗模式或定时器模式。
- 一旦定时时间到,看门狗模式将产生系统复位,定时器模式将产生中断请求。
- 出于安全原因,对 WDT 控制寄存器的写操作需用有正确的安全键值的指令。
- 用停止模式可以实现超低功耗特性。

9.2 WDT 寄存器

WDT 的计数器 WDTCNT 是 16 位增计数器，它不能直接用软件访问，要经 WDTCTL 对 WDTCNT 进行控制，它定位于字地址 0120h 的 16 位读/写寄存器的低字节。所有读写操作都要用字指令，即不带后缀或用后缀.W。在两种工作模式(看门狗或定时器)中，只有含有正确安全键值(05Ah)的指令才能写入 WDTCTL。

WDTCTL(0120h)

15 ... 8	7							0
	HOLD	NMIES	NMI	TMSEL	CNCTT	SSEL	ISI	IS0
	rw-0	rw-0	rw-0	rw-0	r0(w)	rw-0	rw-0	rw-0
←— 069h —→		WDTCTL 读出						
←— 05Ah —→		WDTCTL 写入						

b0～1：IS0、IS1，选择 WDTCNT 的 4 个输出之一。

设晶振频率＝32 768 Hz，系统频率＝1 MHz，可以选择的定时时间见表 9.1 说明。

表 9.1 WDT 定时时间

SSEL	IS1	IS0	定时/ms	
0	1	1	0.064	$t_{MCLK} \times 2^6$
0	1	0	0.5	$t_{MCLK} \times 2^9$
1	1	1	1.9	$t_{ACLK} \times 2^6$
0	0	1	8	$t_{MCLK} \times 2^{13}$
1	1	0	16	$t_{ACLK} \times 2^9$
0	0	0	32	$t_{MCLK} \times 2^{15}$ (PUC 后的值)
1	0	1	250	$t_{ACLK} \times 2^{13}$
1	0	0	1 000	$t_{ACLK} \times 2^{15}$

b2：SSEL，选择 WDTCNT 的时钟源。

SSEL＝0：WDTCNT 以 SMCLK 为时钟源；

SSEL＝1：WDTCNT 以 ACLK 为时钟源。

b3：CNTCL，清除。在两种模式中，写入“1”都使 WDTCNT 从“0”重启动，读出无定义。

b4：TMSEL，选择工作模式：看门狗模式或定时器模式。

TMSEL＝0：看门狗模式；

TMSEL＝1：定时器模式。

b5：NMI，选择 RST/NMI 引脚功能，在 PUC 后它被复位。

NMI＝0：RST/NMI 引脚为复位端，引脚低电平使内部复位有效(电平触发)；

NMI＝1：RST/NMI 引脚作为边沿触发的非屏蔽中断输入。

b6：NMIES，如果选 NMI 功能，则 NMIES 选择 RST/NMI 输入的触发沿，PUC 后复位。

NMIES＝0：上升沿触发 NMI 中断；

NMIES＝1：下降沿触发 NMI 中断。

注意：改变 NMIES

改变 NMIES 位可能产生 NMI 中断。

b7：HOLD，用于停止 WDT 工作。停止时时钟接入电路被禁止，计数停止。停止时的计数值一直保持到 HOLD 复位，然后计数继续进行。PUC 后 HOLD 复位。

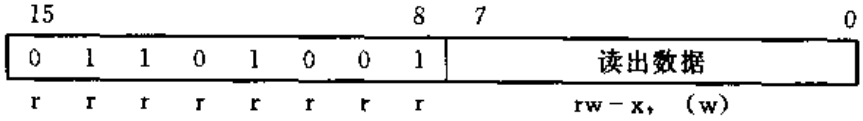
HOLD=0：WDT 功能激活；

HOLD=1：时钟输入禁止，计数停止。

访问 WDT 控制寄存器 WDTCTL

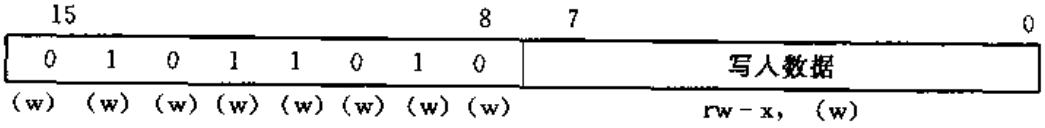
WDTCTL 可以读写，读出时可以不需要安全键值，通过读字地址 0120h 来实现。读出的低字节含有 WDTCTL 的值，高字节读出的值总是 069h。

读 WDTCTL(0120h)



只有在高字节中写入正确的安全键值对 WDTCTL 进行写操作，对 0120h 写操作将改变 WDTCTL 寄存器。低字节含写入 WDTCTL 的数据，高字节应为安全键值 05Ah。如果对字地址 0120h 的高字节中写入除 05Ah 外的其他值将产生系统复位(PUC)。

写 WDTCTL(0120h)



9.3　WDT 中断控制功能

WDT 用 SFR 的两个控制位控制，即

- WDT 中断标志位 WDTIFG(位于 IFG1.0，初始状态为复位)。
- WDT 中断允许位 WDTIE(位于 IE1.0，初始状态为复位)。

当 WDT 被设为看门狗模式时，WDTIFG 标志用于在复位中断服务程序中判断是否是因为看门狗引起的芯片复位。如果 WDTIFG=1，则是 WDT 引起初始化状态(因为定时时间到或写入错误的安全键值)；如果 WDTIFG=0，则是由其他原因引起 PUC。

当 WDT 被设为定时器模式时，WDTIFG 在定时时间到时置位，并产生 WDT 定时器中断请求。定时器模式的中断向量与看门狗模式的不同。在定时器模式中，如果中断请求得到服务，则 WDTIFG 标志在中断服务程序中会自动复位。

WDTIE 位用于对 WDT 在定时器模式时的中断的控制。在定时器模式下，通用中断允许位 GIE 也对 WDT 中断作控制。

9.4 WDT 操作

WDT 模块可置成两种模式:看门狗模式和定时器模式。

看门狗模式

WDT 设置为看门狗模式,WDT 定时时间到或写入错误的安全键值会触发 PUC 信号,这样将自动清除系统寄存器的各位。这也将 WDT 再次设置为看门狗模式,RST/NMI 引脚切换到复位模式。

在发生上电复位或系统复位时,WDT 自动进入看门狗模式,WDTCTL 和 WDTCNT 寄存器全部清除。由于 DCO 振荡器自动设置的频率,WDTCTL 的初始状态导致 WDT 在一个较低的频率上开始运行;WDTCNT 的清除,使用户软件有充足的时间来设置 WDT、关闭 WDT 或调整系统时钟频率。必须根据 MSP430 芯片技术手册及关于时钟系统的章节来仔细确定时钟工作的细节。

选择看门狗模式时,软件必须周期性地在 WDTCTL 的 CNTCL 位中写"1"来使 WDTCNT复位以防止超过设定的定时时间。如果程序发生问题,使定时器因不再复位而超过了定时时间,就会产生系统复位,PUC 信号变成有效。程序从上电时的程序开始地址重新开始执行。可通过检测 SFR 中的 IFG1 的 b0 来判断复位原因,定时时间可以通过 SSEL、IS0 和 IS1 位来选定。

定时器模式

将 WDTCTL 的 TMSEL 置位即选择定时器模式。定时器模式按选定的时间周期产生中断请求。定时时间可用软件对 WDTCTL 寄存器的 CNTCL 置位来进行初始化。

当 WDT 工作在定时器模式时,WDTIFG 在定时时间到时置位,并产生 WDT 定时器中断请求。定时器中断是单源中断,中断请求得到服务时,WDTIFG 标志会自动复位。WDTIE 位和通用中断允许位 GIE 需要置位来允许 WDT 在定时器模式时的中断。

定时器模式的中断向量与看门狗模式的不同。

注意:WDT 改变定时时间

改变定时时间而不同时清除 WDTCNT 将导致不可预料的系统立即复位或中断。定时时间改变应与定时器清除在一条指令中同时完成(例如 MOV #05A0Ah,&WDTCTL)。

在正常工作时改变时钟源可能导致不正确的定时时间。在改变时钟源之前应使定时器暂停。

工作在低功耗模式

MSP430 具有多种低功耗模式。在各种不同类型的低功耗模式中采用了不同的时钟,应用的要求和 MSP430 的时钟系统决定了 WDT 如何配置时钟。例如,准备用到 LPM3,由于在 LPM3 模式中 SMCLK 将停止工作,就不应该将 SMCLK 作为 WDT 的时钟源,否则 WDT 将不能正常工作。

WDT 的保持状态支持低功耗模式,如果需要,则可以结合低功耗模式和 WDT 保持状态来设计系统。

软件实例

```
; 在复位或上电后,WDTCTL 和 WDTCNT 被清除,WDT 处于看门狗模式,定时为 32 ms
;
; 常数定义:
;
WDTCTL    .EQU    0120h                    ; WDT 地址
WDTPW     .EQU    05A00h                   ; 安全键值
T250MS    .EQU    5                        ; SSEL,IS0,IS1 设置 250 ms
T05MS     .EQU    2                        ; SSEL,IS0,IS1 设置 0.5 ms
CNTCL     .EQU    8                        ; WDTCNT 清除位
TMSEL     .EQU    010h                     ; 选模式位
;
; WDT 一旦选择了看门狗模式,需要周期性地清除 WDT
; 例如用以下指令:
;
          ...
          ...
          MOV     #WDTPW+CNTCL,&WDTCTL
          ...
          ...
;
; 用下列指令将 WDT 设置成 250 ms 周期的定时器模式
;
          MOV     #WDTPW+CNTCL+TMSEL+T250MS,&WDTCTL
                                           ; 清除 WDTCNT,选定时器模
                                           ; 式,设置 250 ms
          ...
          ...
; 注意:设定时时间和清除 WDTCNT 应用一条指令完成,以避免不可预期的复位或中断
;
; 切换回看门狗模式并设置 0.5 ms 定时时间用以下指令完成
;
          ...
          ...
          MOV     #WDTPW+CNTCL,&WDTCTL     ; 清除 WDTCNT
;
          MOV     #WDTPW+T05MS,&WDTCTL     ; 选看门狗模式,设置 0.5 ms
          ...
          ...
```

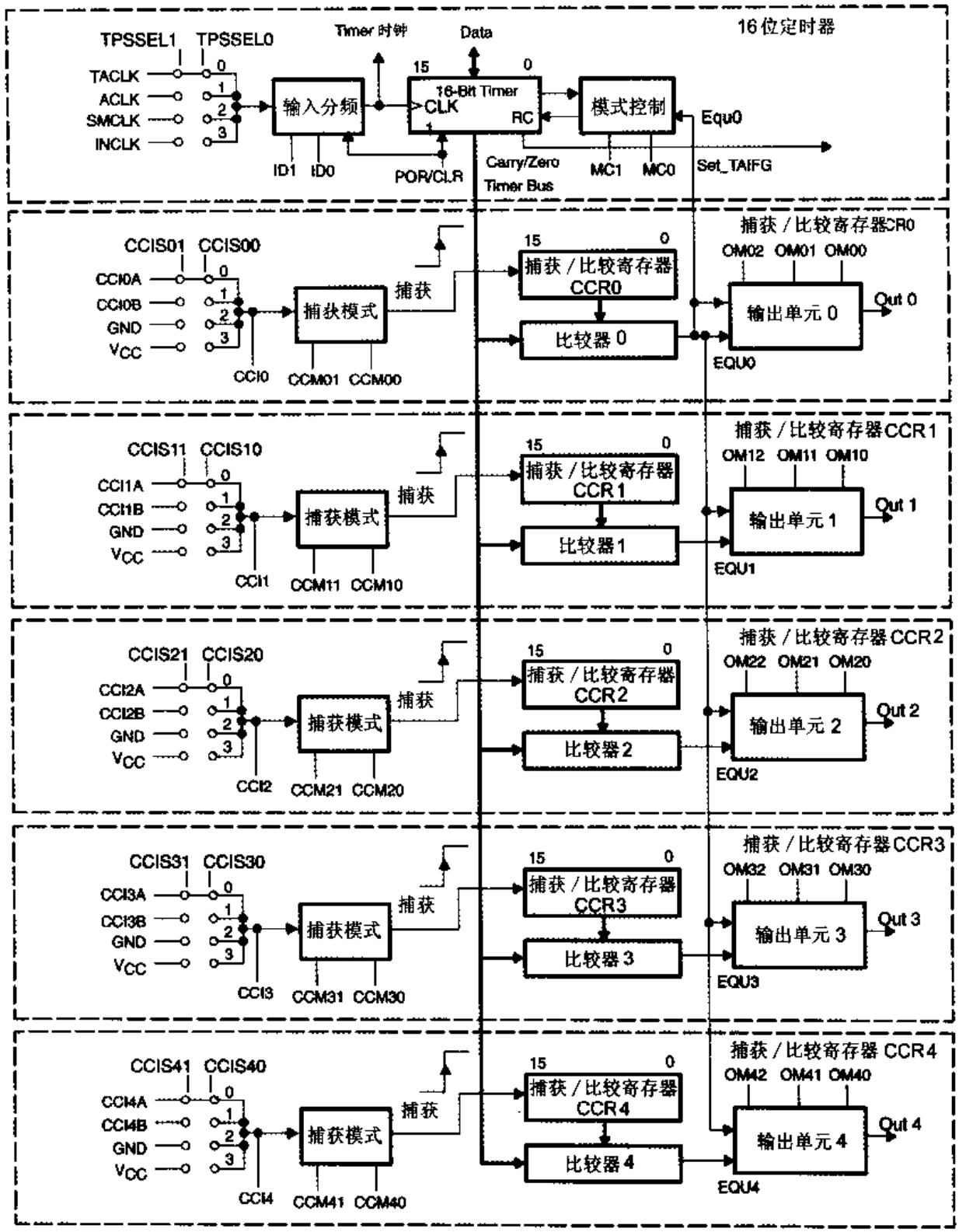


图 10.1　Timer_A 结构

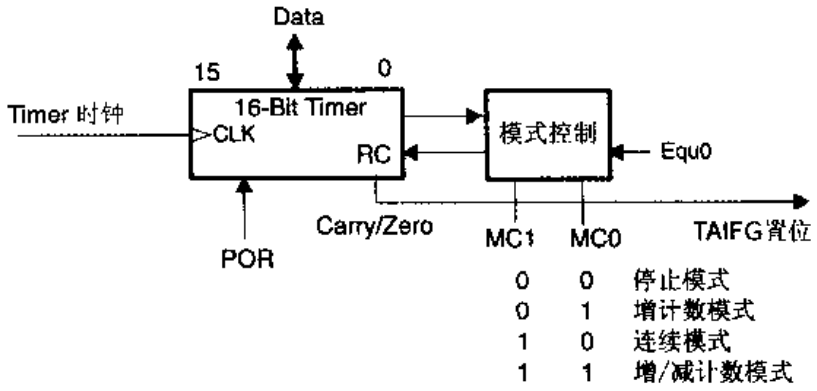


图 10.2 模式控制

10.2.2 时钟源选择和分频

定时器的时钟源可以来自内部时钟(ACLK、SMCLK)或外部时钟(TACLK),见图 10.3。时钟源由 TACTL 中的 SSEL0 和 SSEL1 位来选择。必须注意,改变定时器的时钟源可能发生不确定的时序。因此,选择时钟源之前应停止定时器的工作。

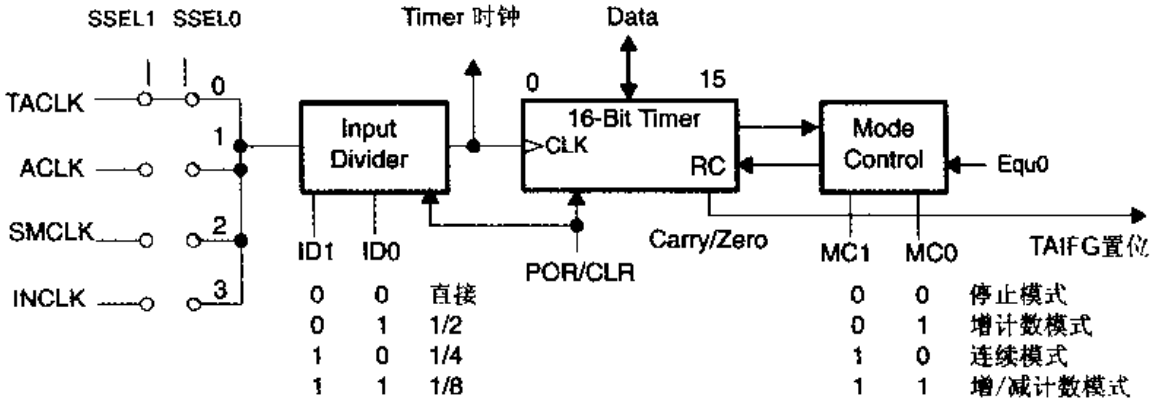


图 10.3 16 位定时器结构

图 10.4 是定时器的时钟源的选择和分频。被选择的时钟信号经过分频器的 1、2、4、8 分

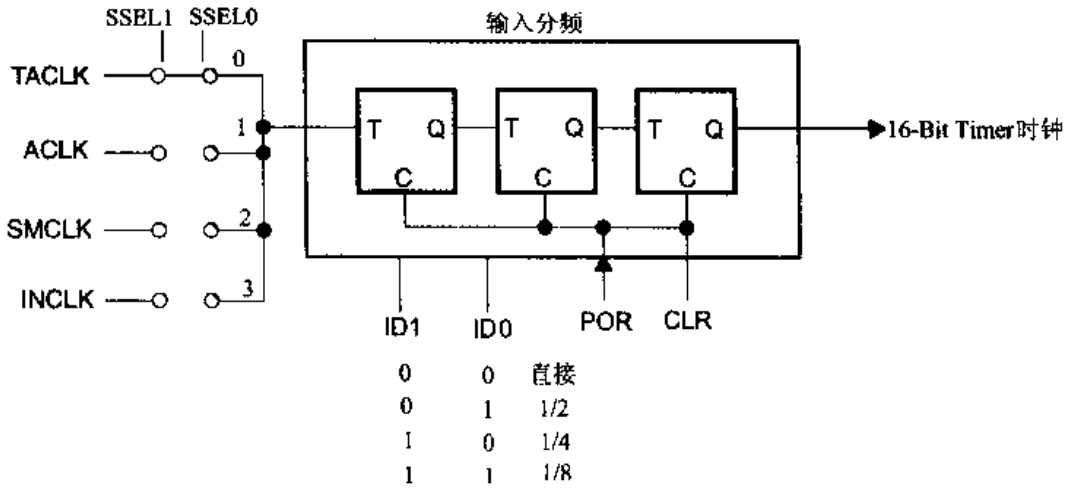
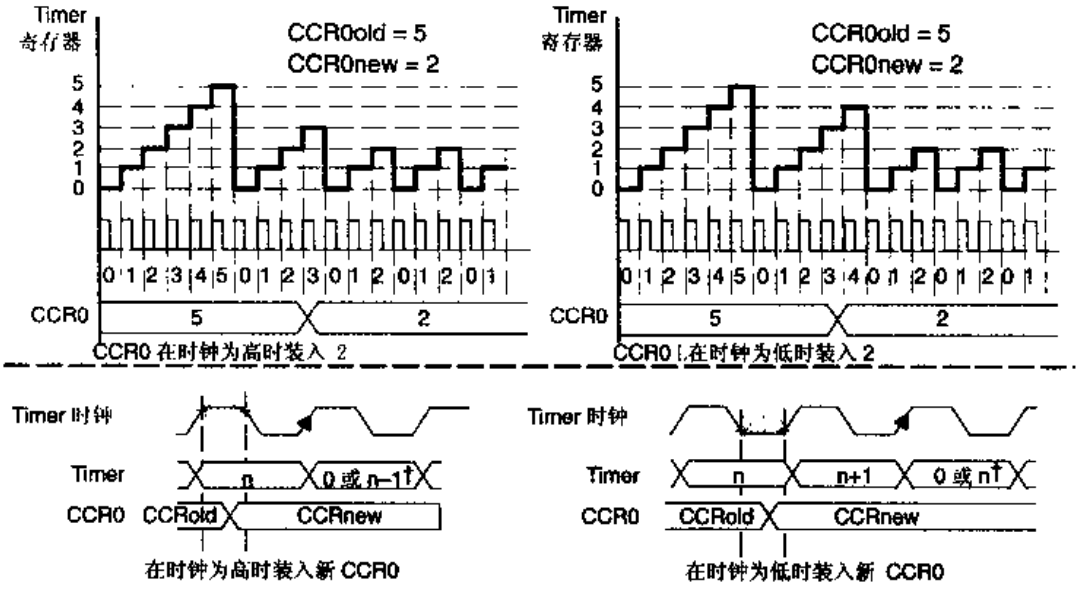


图 10.4 时钟源及分频器



† 增计数模式：0；增/减计数模式：n−1。 † 增计数模式：0；增/减计数模式：n。

图 10.8　新周期小于老周期

10.3.3　连续模式

连续模式用于需要 65 536 时钟周期的应用场合。典型应用是产生多个独立的时序信号。在连续模式中，CCR0 的工作方式与其他比较寄存器相同。

利用捕获/比较寄存器和各输出单元的各种输出模式可以捕获各种外部事件发生的定时器数据或者产生不同类型的输出信号。

在连续模式，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，见图 10.9。

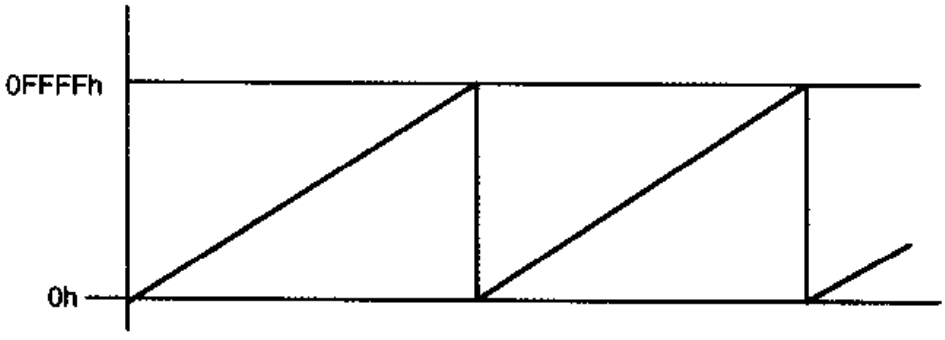


图 10.9　连续模式

当定时器从 0FFFFh 计数到“0”时，TAIFG 标志置位。中断标志会根据各自的中断事件置位，但只有相应的中断允许位及 GIE 位都置位时，中断请求才会发生。图 10.10 是标志设置过程。

连续模式可用于在应用软件中得到定时间隔。每当一个定时间隔到会产生中断请求。在这一事件的中断服务程序中可以将下一个事件发生的时间加到 CCRx 上，见图 10.11。用全部 3 个捕获/比较模块可以产生 3 个独立的时间事件。

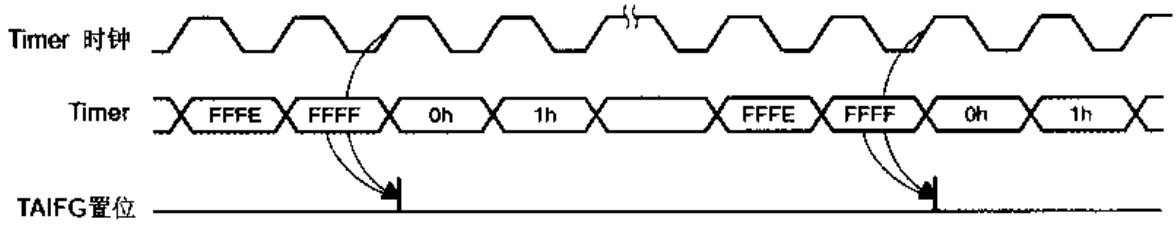


图 10.10 连续模式标志设置

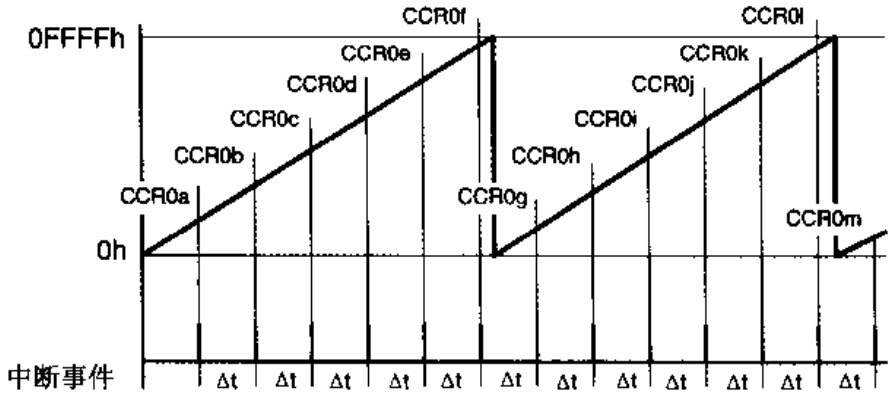


图 10.11 连续模式的定时间隔输出

在 CCR0 用作周期寄存器的其他模式也可以用相同方法产生定时间隔，但是这时的处理会变得很复杂。CCRx 的数据和新的周期相加可能大于 CCR0 的值，当出现这种情况时必须减去 CCR0 的值才能得到正确的定时间隔。

10.3.4 增/减计数模式

增/减计数模式用于必须得到对称波形的场合，定时周期不能为 65 536。在增/减计数模式定时器增计数到与 CCR0 的值相等，然后反向作减计数到“0”，见图 10.12。增/减计数模式的周期是 CCR0 寄存器数值的 2 倍。

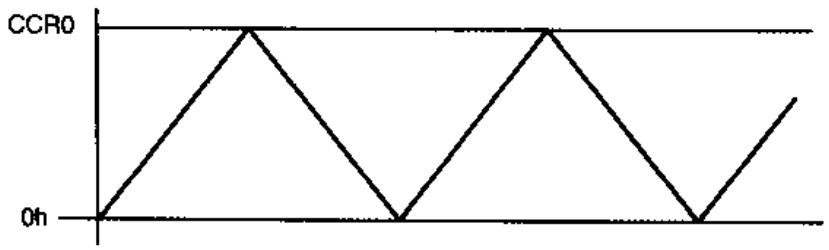


图 10.12 增/减计数模式

增/减计数模式可以应用在要求输出信号之间有死区的应用场合。例如，为了避免过载，驱动 H 桥的 2 个信号不能同时出现高状态。在图 10.13 的例子中，死区时间 t_{dead} 为

$$t_{dead} = T_{Timer} \times (CCR1 - CCR2)$$

其中：t_{dead} 需要 2 个驱动信号都为无效的时间。

T_{Timer} 定时器的时钟周期。

CCRx 捕获/比较寄存器的值。

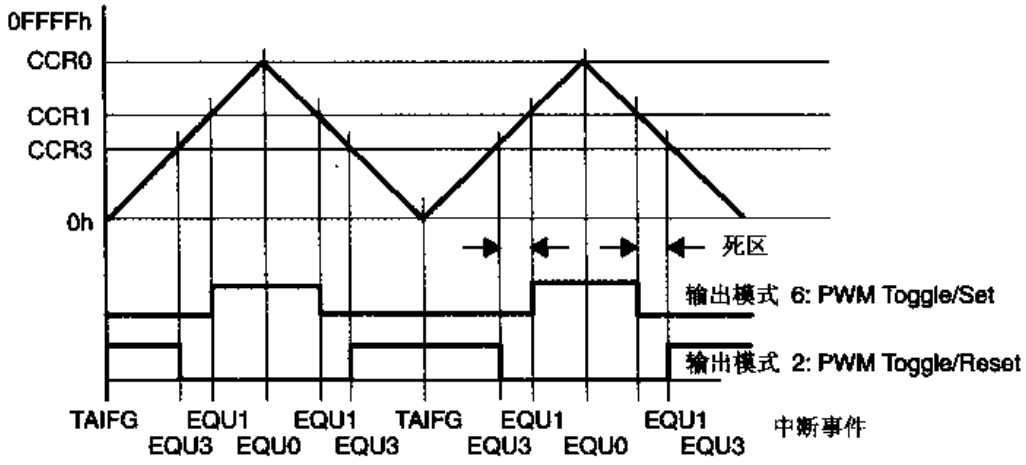


图 10.13　增/减计数模式的输出

计数方向由一个触发器锁存，见图 10.14。记忆计数方向是有用的，当用户将定时器暂停，再重启动时定时器会在停止前的计数方向上继续计数。例如，当定时器因为 MC 位复位停止计数时处于减计数状态，重新启动时，定时器仍然会进行减计数。如果应用中并不要求这一点，则应该用 TACTL 中的 CLR 位来清除计数方向。但是要注意，CLR 位同时也会影响其他已建立的定时器状态。

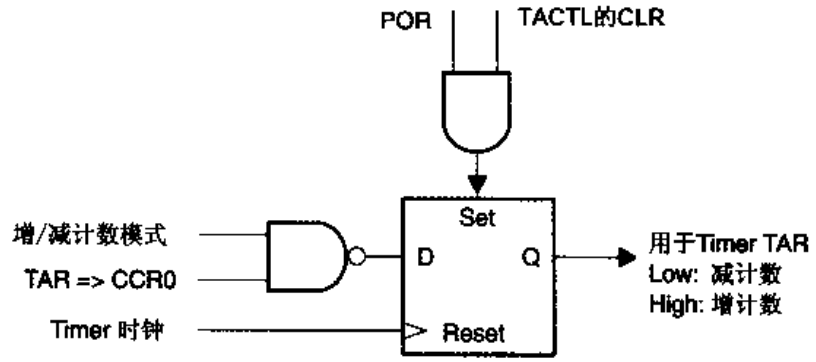


图 10.14　增/减计数方向控制

在增/减计数模式，中断标志 CCIFG0 和 TAIFG 会在相等的定时间隔置位，见图 10.15。在一个完整的周期中，每个标志只置位一次，分别在半周期时发生。当定时器从 CCR1 增计数到 CCR0 时，中断标志 CCIFG0 置位。当定时器从 0001h 减计数到 0000h 时，中断标志 TAIFG 置位。如果中断允许，则每个中断标志都会产生中断请求。

在增/减计数模式中，如果在计数过程中改变周期值，则定时器会比增计数模式更难以捉摸。不但如增计数模式那样，CCR0 改变时的时钟相位会影响定时器的运行，而且，增/减计数的方向也会影响定时器的运行。

如果定时器处于增计数时 CCR0 写入新周期值，则这时定时器的表现与增计数模式中完全一样。但是，如果定时器处于减计数时 CCR0 写入新周期值，这时定时器将继续减计数至“0”。新周期只有在完成减计数后才会起作用，见图 10.16。

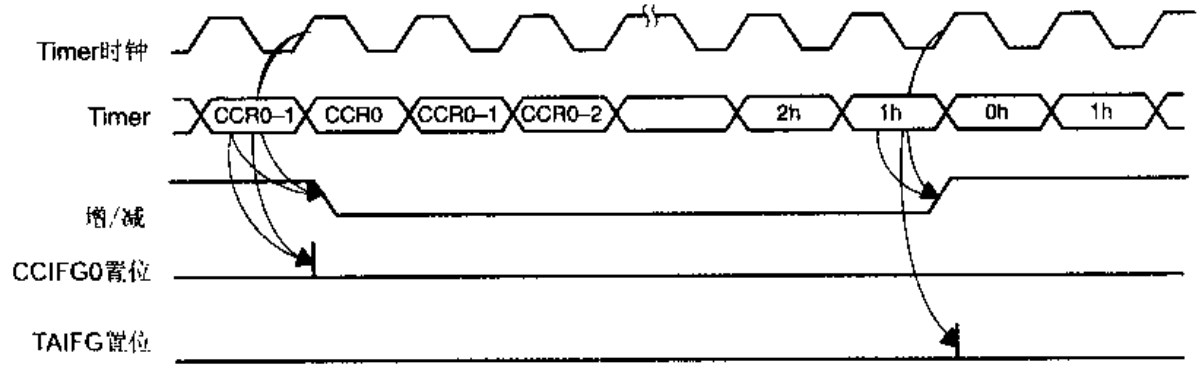


图 10.15 增/减计数模式的标志

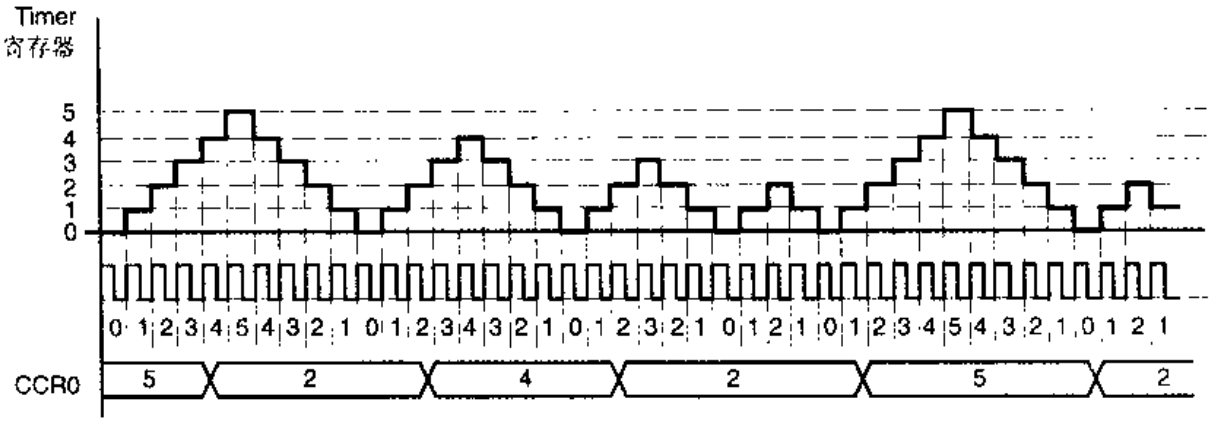


图 10.16 增/减计数模式中修改 CCR0

10.4 捕获/比较模块

定时器有 3 个相同的捕获/比较模块为实时处理提供了灵活的控制手段。每个模块都可用于捕获事件发生时间或产生定时间隔。每发生一次捕获或一次定时间隔，捕获/比较模块寄存器将产生中断。控制字 CCTLX 中的模式位 CAPx 选择比较或捕获模式。控制字 CCTLX 中的 CCMx1 和 CCMx0 定义产生捕获功能的条件。

中断允许位 CCIEx 和中断标志位 CCIFGx 用于捕获和比较模式中。CCIEx 允许相应的中断。当发生捕获或比较事件时，CCIFG 置位。

捕获输入 CCIxA 和 CCIxB 连接外部引脚或内部信号。不同型号的 MSP430 芯片连到 CCIxA 和 CCIxB 的信号也不相同。请查阅芯片的技术手册。

图 10.17 是捕获/比较模块的结构。

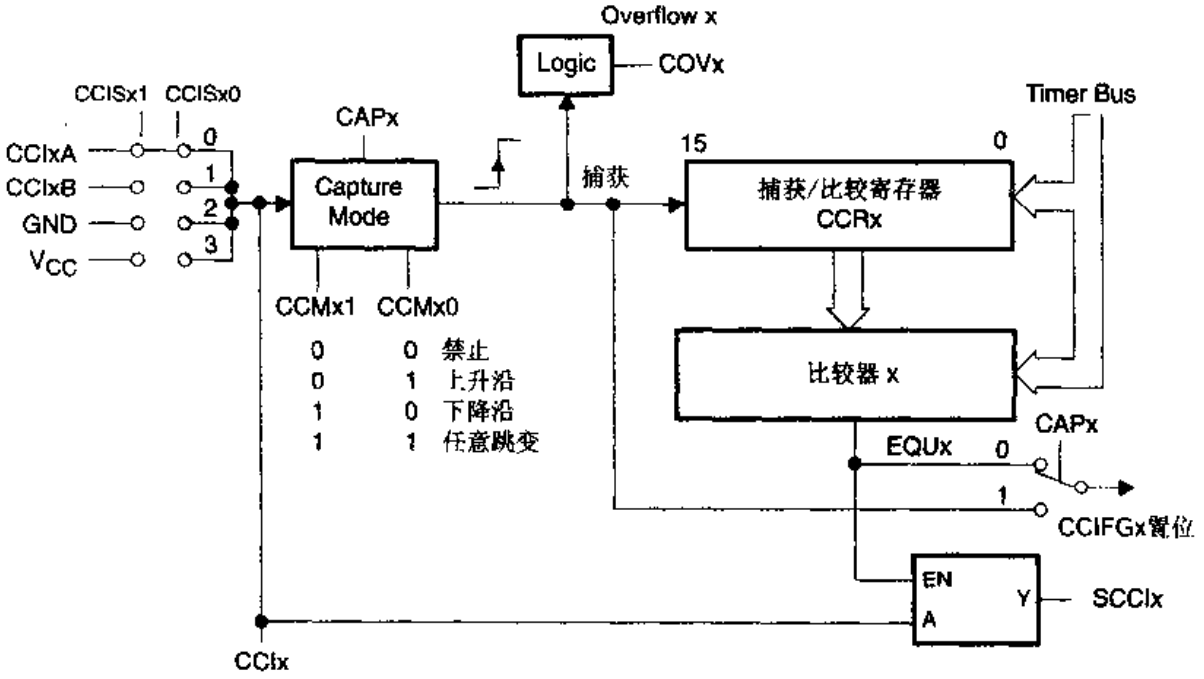


图 10.17　捕获/比较模块结构

10.4.1　捕获模式

控制字 CCTLx 中的模式位 CAPx 置位将选定捕获模式。捕获模式用于确定事件发生的时间，可用于速度计算或时间测量等场合。如果在选定的输入引脚上发生选定的脉冲触发沿（上升沿、下降沿或任意跳变），则定时器计数值将被复制到捕获寄存器 CCRx 中。捕获可以用软件作初始化。

完成捕获后会发生：

- 控制字 CCTLx 中的中断标志 CCIFGx 置位。
- 如果通用中断允许位 GIE 和相应的中断允许位 CCIEx 置位，将产生中断请求。

捕获/比较模块的输入用控制字 CCTLx 中的 CCISx1、CCISx0 位选择，见图 10.18。引脚输入信号可以用软件在任意时刻从 CCIx 位读入，在比较模式中输入信号也可以用比较信号 EQUx 锁存。这一特性的设计是专门针对用 Timer_A 实现串行通信的。

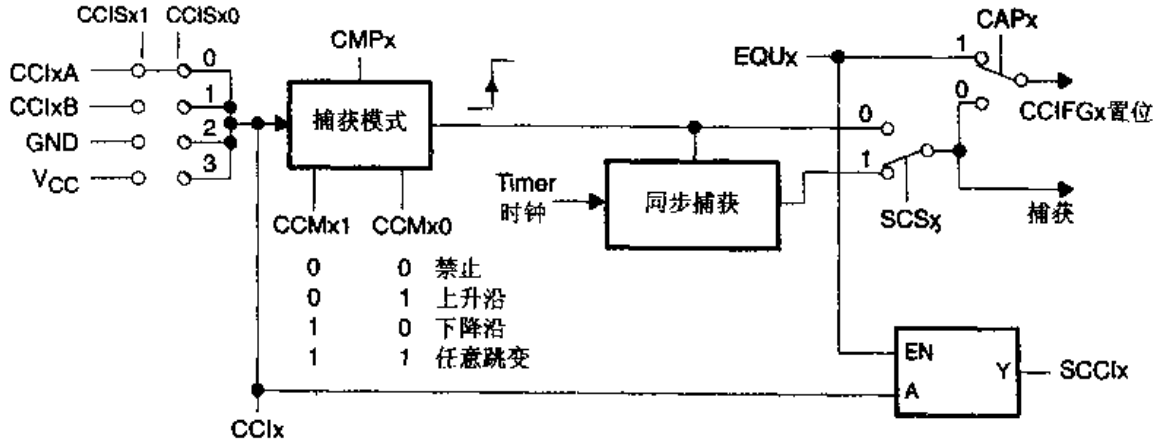


图 10.18　捕获逻辑的输入信号

捕获信号也可以用定时器时钟同步，以避免发生定时器数据与捕获信号的时间竞争，见图 10.19的说明。控制字 CCTLx 中的 SCSx 位选择捕获信号的同步功能。

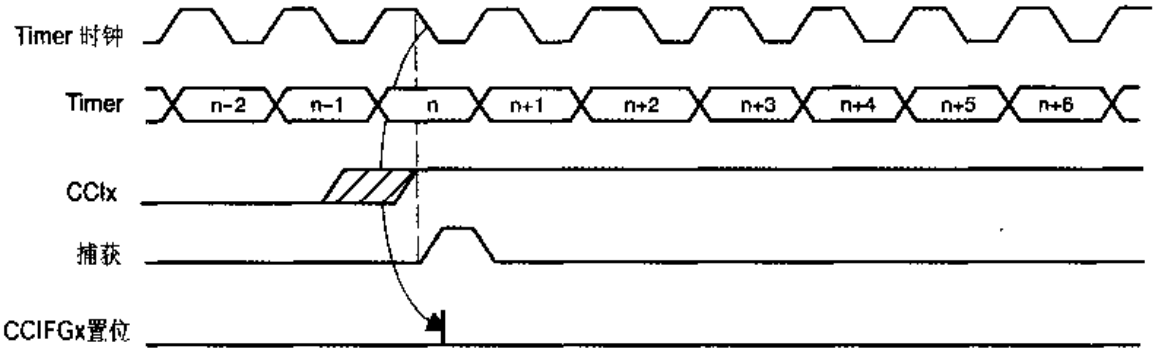


图 10.19 捕获信号

当应用低速定时器时钟时，可以采用异步的信号捕获。下例说明如何用软件验证捕获数据及加以校正。

例：软件处理异步捕获信号

```
;
; 捕获/比较寄存器数据在相应的 CCIFG 中断处理程序中读取
; 定时器时钟大大慢于系统时钟 MCLK
;
CCRx_Int_hand   …                                ; 中断处理开始
                …
                CMP     &CCRx,&TAR               ; 数据检查:CCRx = TAR
                JEQ     Data_Valid
                MOV     &TAR,&CCRx               ; CCRx 中数据错,用定时器数据
Data_Valid      …                                ; CCRx 中数据对
                …
                RETI
```

每个捕获/比较寄存器提供溢出信号，告诉用户在前一次的捕获数据读出之前已开始新的一次捕获。用控制字 CCTLx 中的 COVx 位的置位说明这一状况，见图 10.20。

下例说明溢出位 COVx 用软件复位。

例：捕获数据发生溢出的软件处理

```
;
; 捕获/比较寄存器 CCRx 用软件读取后,下一条指令立即测试溢出位
; 并决定处理的对策
;
CCRx_Int_hand   …                                ; 中断服务程序开始
                MOV     &CCRx,RAM_Buffer
                BIT     #COV,&CCTLx
                JNZ     Overflow_Hand
                …
```

```
                RETI
Overflow_Hand   BIC     #COV,&CCTLx          ; 捕获溢出位复位
                                             ; 恢复丢失的同步
                ...
                RETI
```

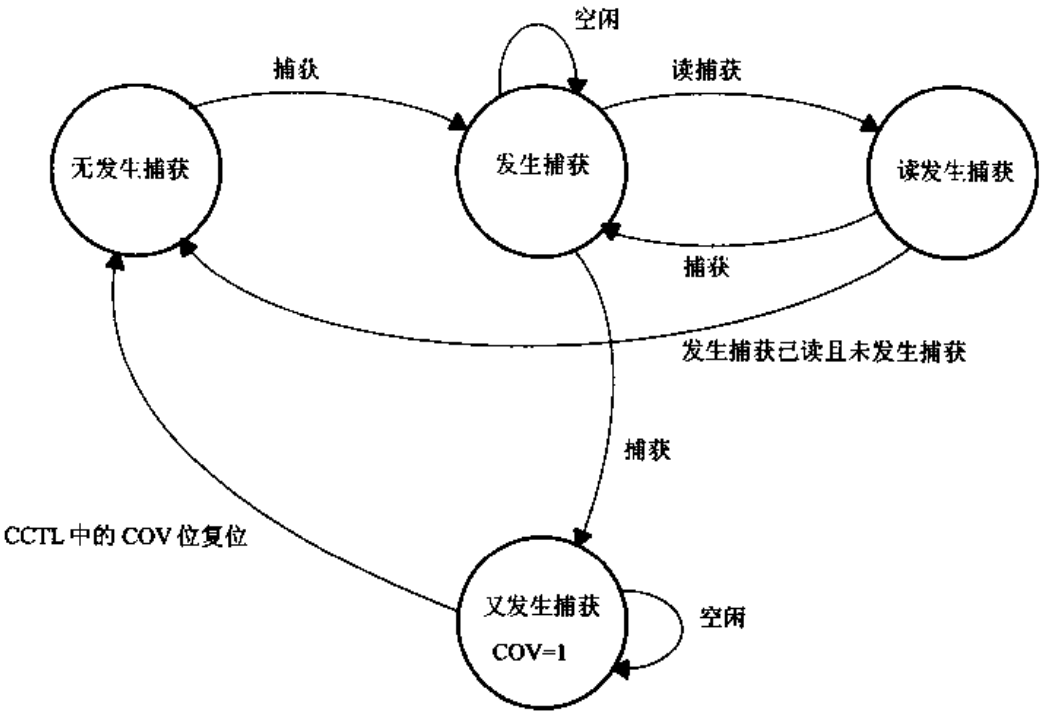


图 10.20　捕获循环状态

注意:捕获与定时器暂停

当定时器暂停时捕获应该停止,顺序应是先停止捕获功能,再停止定时器计数,捕获功能重新开始时,顺序是先开始捕获功能,再开始定时器计数。

捕获可以用软件初始化。这对于以下情况是有用的,如:

- 用软件测量时间。
- 测量 2 个硬件事件之间的时间。
- 测量系统时钟频率。

软件初始化捕获用到 CCISx1、CCISx0 位和捕获模式选择位 CCMx1、CCMx0。最简单的实现例子为:捕获模式选择为任意跳变沿,CCISx1 位置位,在程序中将 CCISx0 在 V_{CC} 和 GND 之间切换,每次切换即发生捕获,见图 10.21。

例: 软件实现捕获

```
; 捕获/比较寄存器 CCRx 由软件读入
; 设 CCMx1、CCMx0 和 CCISx1 置位,CCIS0 选择 CCIx 信号为高或低
;
                ...
                XOR     #CCISx0, &CCTLx
                ...
```

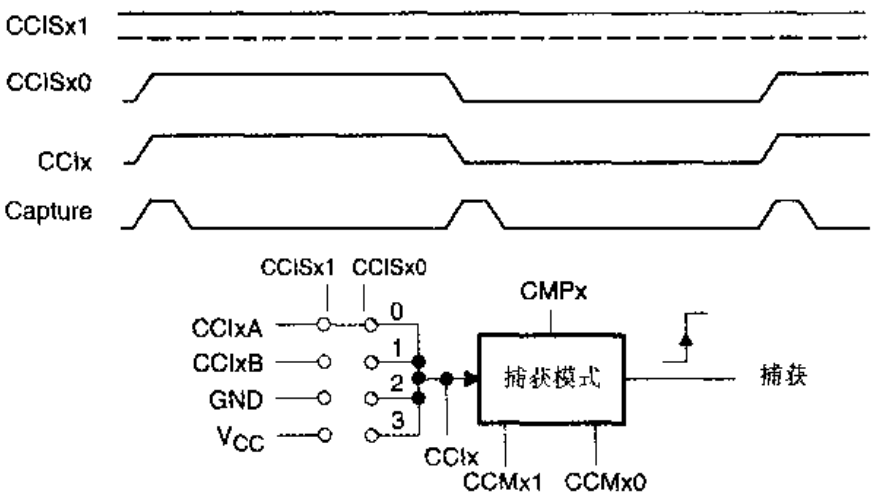


图 10.21 软件捕获举例

10.4.2 比较模式

如果控制字 CCTLx 中的 CAPx 复位，则选择比较模式，这时所有捕获硬件停止工作，捕获模式溢出逻辑也不工作。

比较模式常用于在设定的定时间隔产生中断，或者结合输出单元发生如 PWM 波形这样的信号。如果定时器值等于 CCRx 中的值，那么：

- 位于控制字 CCTLx 中的中断标志 CCIFGx 置位。
- 如果 GIE 和 CCIEx 置位，将产生中断请求。
- EQUx 信号输出到输出单元中。信号根据选定的输出模式影响 OUTx。

当定时器数据大于等于 CCR0 的值时，EQU0 信号为真；当定时器数据等于相应的 CCR1、CCR2 的值时，EQU1、EQU2 信号为真。

10.5 输出单元

每个捕获/比较模块包含一个输出单元，见图 10.22。输出单元用于发生输出信号，如 PWM 波形。每个输出单元有 8 种工作模式，可以产生基于 EQU0 和 EQUx 状态的多种信号。输出模式由 CCTLx 中的控制位 OMx 选择。

10.5.1 输出模式

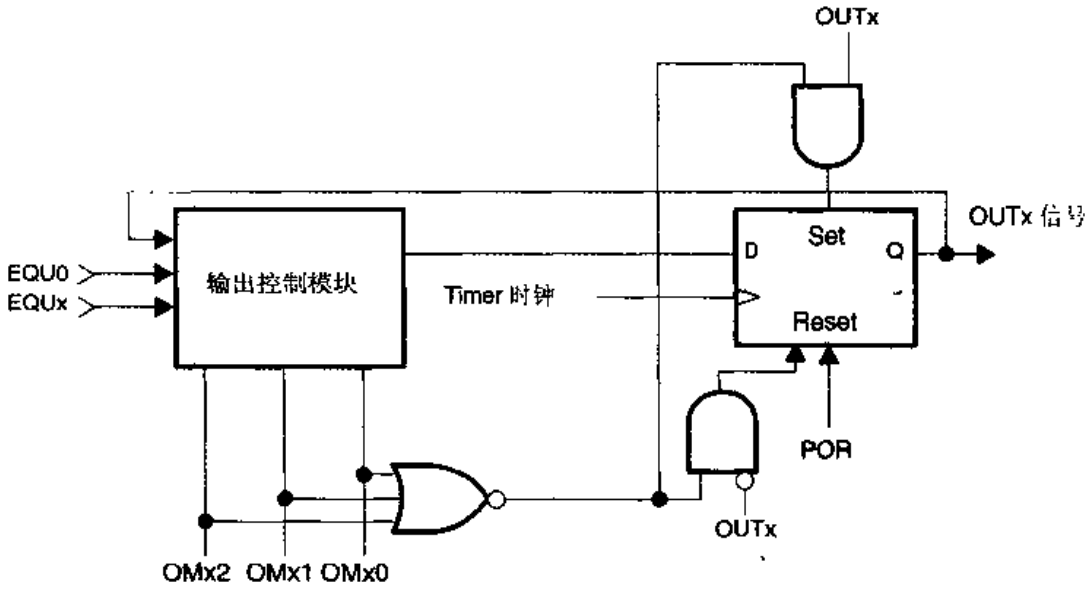
除了输出模式 0，所有的输出都在定时器时钟的上升沿发生变化。输出模式 2、3、6 和 7 不适合输出单元 0。OMx 定义的输出模式如下：

输出模式 0：输出模式。

输出信号 OUTx 由每个捕获/比较模块中控制寄存器 CCTLx 的 OUTx 位定义，输出信号 OUTx 在写入位信息后立即更新。

输出模式 1：设置模式。

输出在定时器值等于 CCRx 时置位。输出保持置位直至定时器复位或选择另一



0	0	0	输出：OUTx信号影响OUTx位，
0	0	1	置位：OUTx信号影响EQUx，
0	1	0	PWM 翻转/复位：EQUx使OUTx翻转，EUQ0使OUTx复位，
0	1	1	PWM 置位/复位：EQUx使OUTx置位，EQU0使OUTx复位，
1	0	0	翻转：EQUx使OUTx翻转，
1	0	1	复位：EQUx使OUTx复位，
1	1	0	PWM翻转/置位：EQUx使OUTx翻转，EQU0使OUTx置位，
1	1	1	PWM 复位/置位：EQUx使OUTx复位和置位 。

注：除模式0外，OUTx 信号均在 Timer 时钟上升沿更新，模式 2,3,6,7 对输出单元 UNIT0 无用。

图 10.22　输出单元

种输出模式。

输出模式 2：PWM 翻转/复位模式。

输出电平在定时器值等于 CCRx 时翻转，当定时器值等于 CCR0 时复位。

输出模式 3：PWM 置位/复位模式。

输出在定时器值等于 CCRx 时置位，当定时器值等于 CCR0 时复位。

输出模式 4：翻转模式。

输出电平在定时器值等于 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。

输出在定时器值等于 CCRx 时复位，输出保持复位直至选择另一种输出模式。

输出模式 6：PWM 翻转/置位模式。

输出在定时器值等于 CCRx 时翻转，当定时器值等于 CCR0 时置位。

输出模式 7：PWM 复位/置位模式。

输出在定时器值等于 CCRx 时复位，当定时器值等于 CCR0 时置位。

10.5.2　输出控制模块

输出控制模块预备好 OUTx 的值，在定时器时钟的下一个上升沿锁存入 OUTx 触发器，见图 10.23 和表 10.2。相等信号 EQUx 和 EQU0 在定时器时钟的低电平采样。

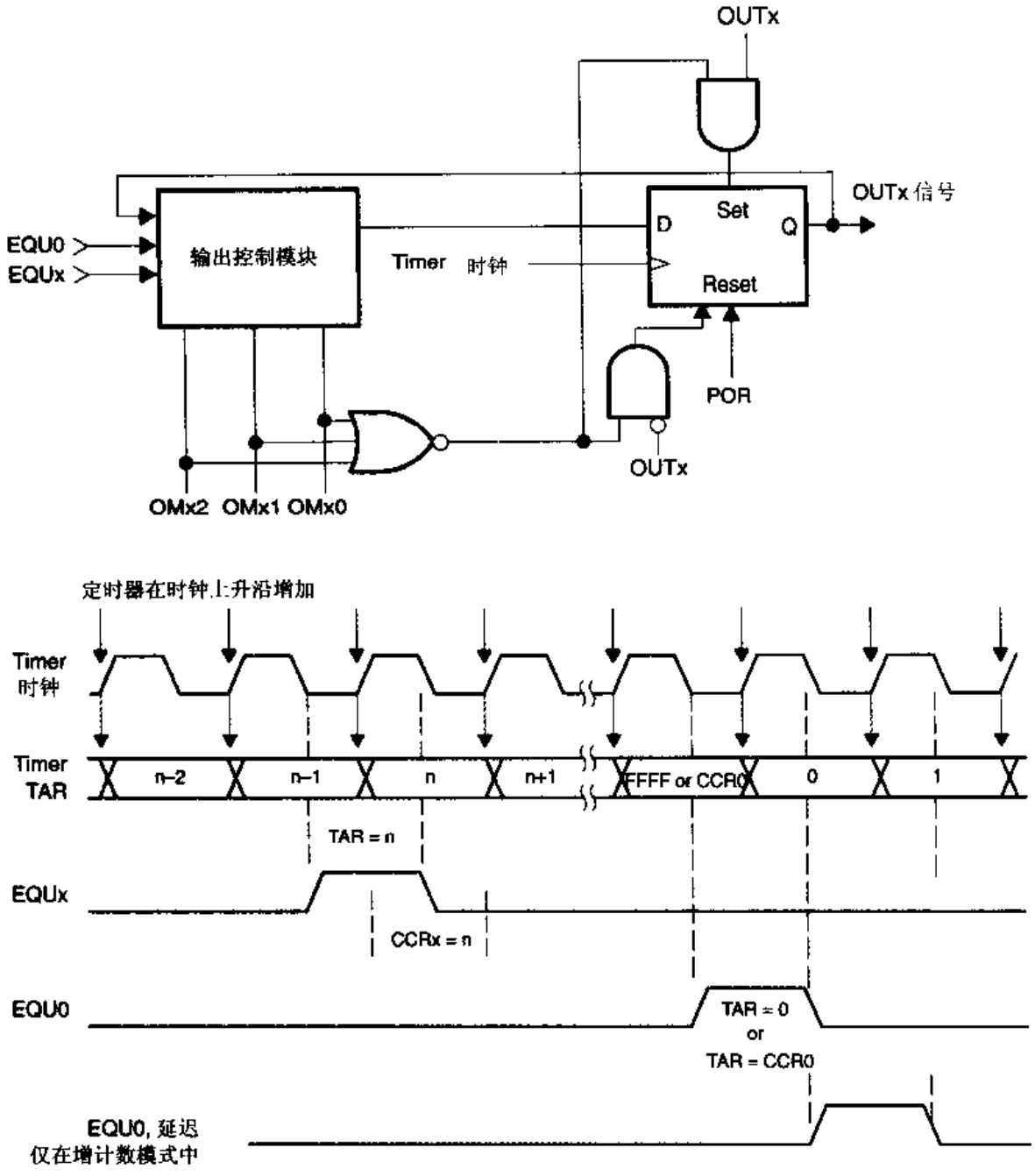


图 10.23 输出控制模块及时序

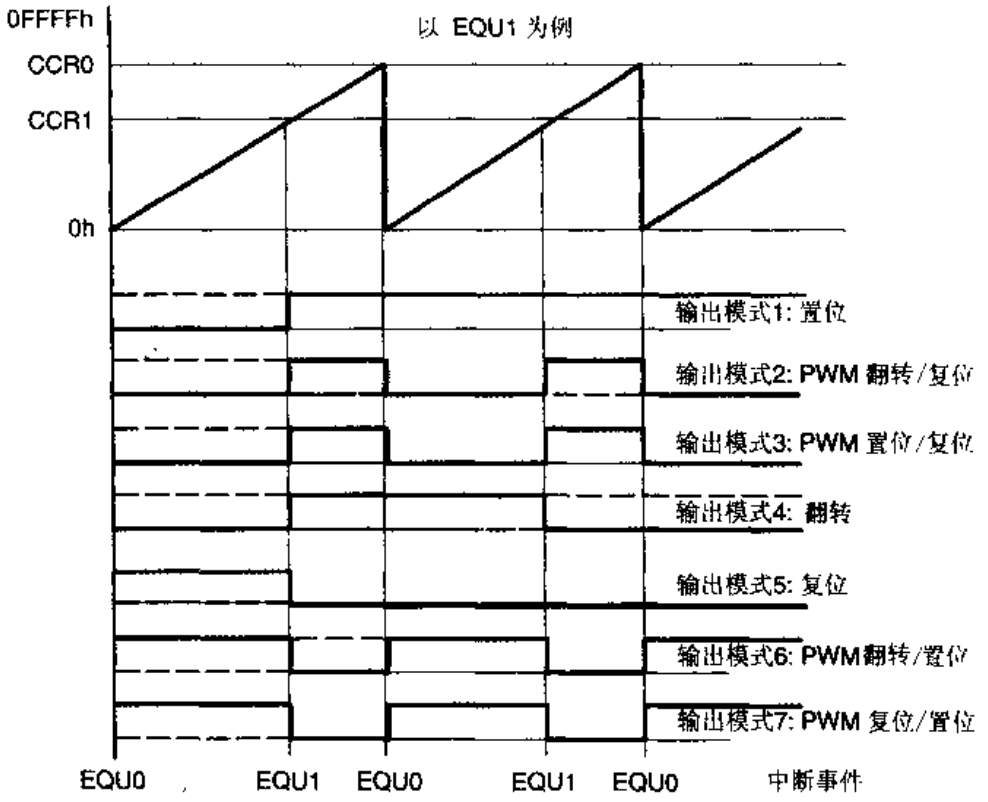


图 10.24 增计数模式的输出实例

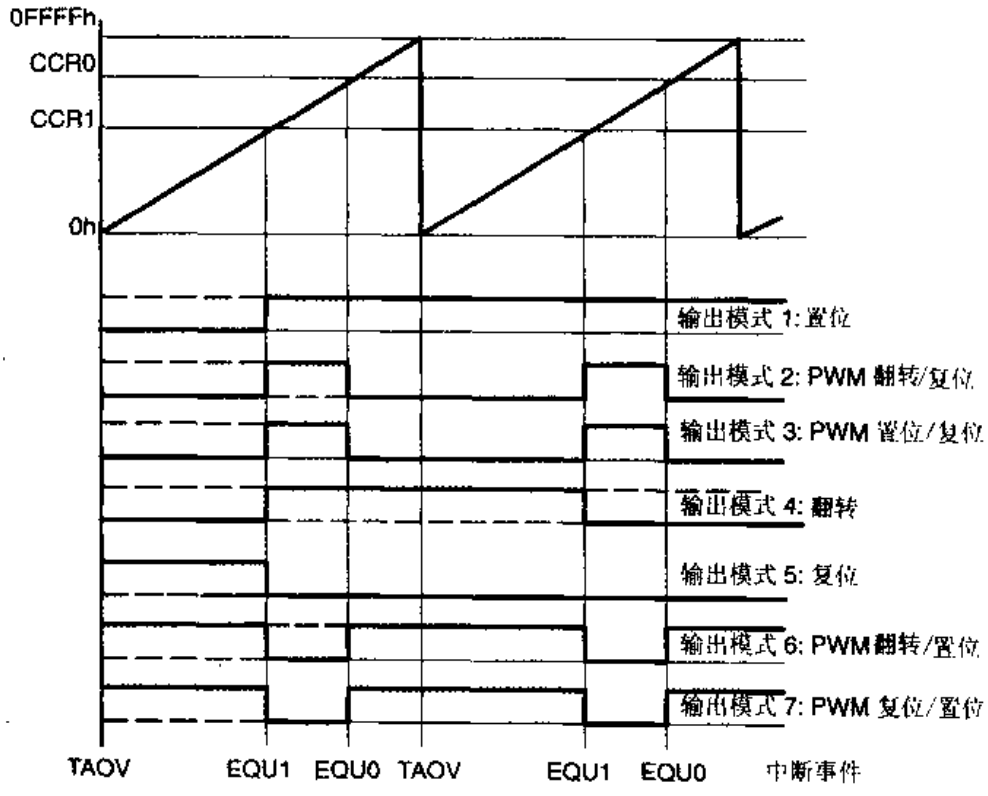


图 10.25 连续模式的输出实例

增/减计数模式实例

当定时器的值在任一计数方向上出现等于 CCRx 和等于 CCR0 时，OUTx 信号按选择的输出模式发生改变，见图 10.26。

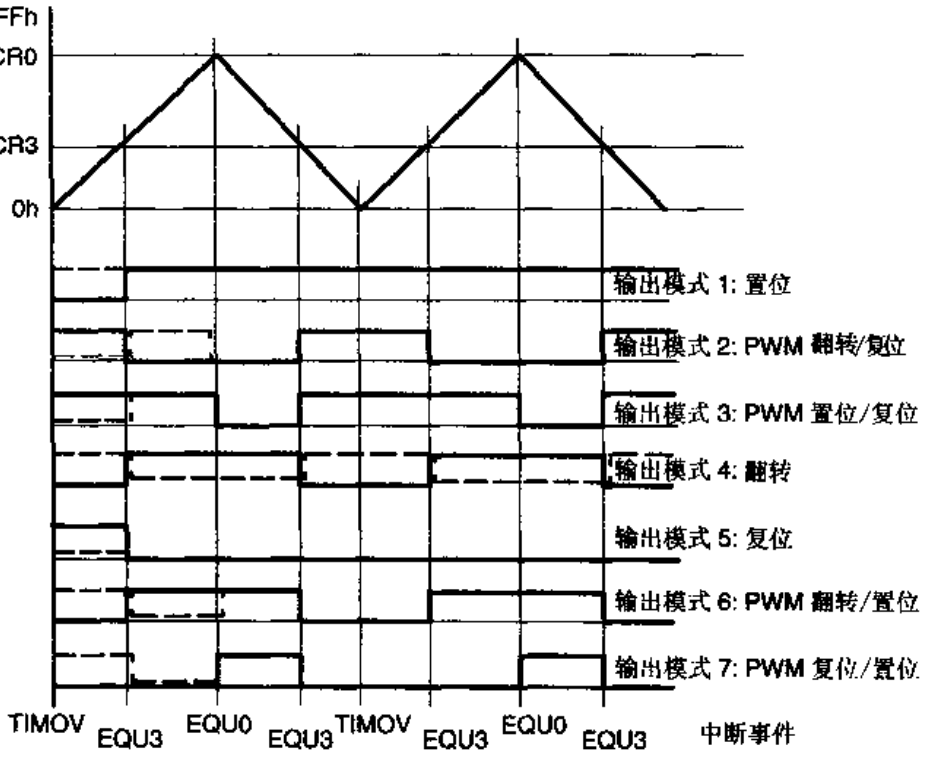


图 10.26　增/减计数模式实例

10.6　Timer_A 的寄存器

Timer_A 的寄存器是字结构的，必须用字指令来访问，见表 10.3。

表 10.3　Timer_A 的寄存器

寄存器	缩　写	寄存器类型	地　址	初始状态
Timer_A 控制寄存器	TACTL	读/写	160h	POR 复位
Timer_A 寄存器	TAR	读/写	170h	POR 复位
捕获/比较控制寄存器 0	CCTL0	读/写	162h	POR 复位
捕获/比较寄存器 0	CCR0	读/写	172h	POR 复位
捕获/比较控制寄存器 1	CCTL1	读/写	164h	POR 复位
捕获/比较寄存器 1	CCR1	读/写	174h	POR 复位
捕获/比较控制寄存器 2	CCTL2	读/写	166h	POR 复位
捕获/比较寄存器 2	CCR2	读/写	176h	POR 复位
中断向量寄存器	TAIV	只读	12Eh	(POR 复位)

10.6.1 Timer_A 控制寄存器 TACTL

全部对定时器及其操作的控制位都位于定时器控制寄存器 TACTL 中。在 POR 信号后各位全部自动复位，但是在 PUC 后不受影响。控制寄存器必须用字指令访问。

TACTL(160h)

15															0
未用								SSEL1 - 0		ID1 - 0		MC1 - 0	未用	CLR	TAIE	TAIFG
rw -	rw -	rw -	rw -	rw -	rw -	rw -	rw -	rw -	rw -	rw -	rw -	rw -	w -		rw -	rw -
(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)		(0)	(0)

b0：TAIFG：指示定时器溢出事件。

增计数模式：　当定时器由 CCR0 计数到“0”时 TAIFG 置位。

连续模式：　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。

b1：定时器溢出中断允许位 TAIE。置位使溢出产生中断请求，复位禁止中断。

b2：定时器清除位 CLR。定时器和分频器在 POR 后或 CLR 置位时复位。CLR 会自动复位，读出总是为“0”。如果不是因清除模式控制位使定时器暂停，则定时器在下一个有效时钟沿开始增计数。

b3：未用。

b4～5：模式控制位。

MC1	MC0	控制模式	说　明
0	0	停止	定时器暂停
0	1	增计数到 CCR0	计数至 CCR0 并从“0”重新开始
1	0	连续增计数	增计数至 0FFFFh 并从“0”重新开始
1	1	增/减计数	增计数至 CCR0，减计数至“0”

b6～7：输入分频控制位。

ID1	ID0	分　频	说　明
0	0	/1	输入时钟直通定时器
0	1	/2	输入时钟 2 分频
1	0	/4	输入时钟 4 分频
1	1	/8	输入时钟 8 分频

b8～9：定时器进入输入分频器的输入时钟源选择。

SSEL1	SSEL0	输入信号	说　明
0	0	TACLK	用外部引脚信号，见芯片手册
0	1	ACLK	用 ACLK
1	0	MCLK	用 SMCLK
1	1	INCLK	见芯片手册说明

b10～15：未用。

注意：修改 Timer_A 控制位

如果用 TACTL 控制寄存器中的控制位改变定时器工作，则在修改时定时器应暂停，尤其是修改输入选择位、输入分频位和定时器清除位。异步时钟、输入时钟和系统时钟的异步可能引起时间竞争，使定时器工作表现不确定。

推荐的修改程序顺序为

(1) 修改控制寄存器和停止定时器。

(2) 启动定时器工作。

```
例：MOV     #01C6h,&TACTL          ；ACLK/8,定时器停止并复位
    BIS     #10h,&TACLT            ；开始增计数模式
```

10.6.2　Timer_A 寄存器 TAR

TAR 寄存器包含定时器的值。

TAR(170h)

15															0
定时器值															
rw-	rw-	rw-	rw-	rw-	rw-	rw-	rw-	rw-	rw-	rw-	rw-	rw-	rw-	rw-	rw-
(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)

注意：修改 Timer_A 寄存器 TAR

如果 ACLK、SMCLK 或外部时钟 TACLK、INCLK 已选择为定时器时钟，则对 TAR 的写入应在定时器停止时进行，否则定时器的值是不可预测的。因为在这种情况下，CPU 时钟 MCLK 和定时器时钟是异步的，可能存在时间竞争。

10.6.3　捕获/比较控制寄存器 CCTLx

每个捕获/比较模块都有它自己的控制字 CCTLx。POR 信号将 CCTLx 复位，PUC 不影响 CCTLx。

CCTLx(162h～166h)

15															0
捕获模式		输入选择		SCS	SCCI	未用	CAP	OUTMODx				CCIE	CCI	OUT	COV	CCIFG
rw-	rw-	rw-	rw-	rw-	rw-	r-	rw-	rw-	rw-	rw-	rw-	rw-	r	rw-	rw-	rw-
(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)		(0)	(0)	(0)

b0：CCIFGx，捕获/比较中断标志。

捕获模式：　置位表示在寄存器 CCRx 中捕获了定时数值。

比较模式：　置位表示定时器值等于比较寄存器 CCRx 的值。

CCIFG0：　当中断请求被接受时，CCIFG0 会自动复位。

CCIFG1～2：　读中断向量字 TAIV 后，确定中断向量的标志自动复位。如果不访问 TAIV 寄存器，则中断标志必须用软件复位。

如果相应的中断允许位复位，则将不会产生中断请求，但标志仍置位，这时标志必须用软件复位。

如果相应的中断允许位置位，则用软件将 CCIFG1～CCIFG4 置位会产生中断请求。

b1：COV，捕获溢出标志。

比较模式(CAP=0)：捕获信号复位，捕获事件不会使 COV 置位。

捕获模式(CAP=1)：如果捕获寄存器值被读出前再次发生捕获事件，则 COV 置位；COV 不会在读捕获值时复位；必须用软件复位。

b2：OUT，如果选输出模式 0，则由 OUTx 位确定 OUTx 信号。

b3：CCIx，捕获/比较输入。可将选定信号(CCIxA、CCIxB、V_{CC}或 GND)读出。

b4：CCIEx，中断允许位。允许(1)或禁止(0)捕获/比较模块 x 中断请求，GIE 位必须同时置位才能产生中断请求。

b5～7：输出模式选择。

b5～7	输出模式	说　明
0	输出	OUTx 位的数字确定 Outx
1	置位	比较信号 EQUx 使 Outx 置位
2	PWM 翻转/复位	比较信号 EQUx 使 Outx 翻转，EQU0 使 Outx 复位
3	PWM 置位/复位	比较信号 EQUx 使 Outx 置位，EQU0 使 Outx 复位
4	翻转	比较信号 EQUx 使 Outx 翻转
5	复位	比较信号 EQUx 使 Outx 复位
6	PWM 翻转/置位	比较信号 EQUx 使 Outx 翻转，EQU0 使 Outx 置位
7	PWM 复位/置位	比较信号 EQUx 使 Outx 复位，EQU0 使 Outx 置位

b8：CAP，选择捕获模式或比较模式。

0：比较模式，1：捕获模式。

b9：只读，总是为“0”。

b10：SCCIx，选定的输入信号(CCIxA、CCIxB、V_{CC}或 GND)用比较器相等信号 EQUx 锁存在透明锁存器中，可用 SCCIx 读出。

b11：SCS，用于捕获输入信号与定时器时钟信号的同步。

0：异步捕获，1：同步捕获。

b12～13：CCIS0 和 CCIS1，输入选择。

在捕获模式中，定义捕获事件的输入信号源；在比较模式中无用。

0：选 CCIxA，1：选 CCIxB，2：GND，3：V_{CC}

b14～15：选择捕获模式。

b14～15	捕获模式	说　明
0	禁止	禁止捕获模式
1	上升沿	上升沿捕获
2	下降沿	下降沿捕获
3	跳变沿	在上升沿与下降沿都捕获

注意：同时捕获和捕获模式选择

如果将工作模式从比较模式切换为捕获模式，则不能同时进行捕获。那样在捕获/比较寄存器中的值是不可预料的。推荐的程序顺序为

(1) 修改控制寄存器，由比较模式切换到捕获模式。

(2) 捕获。

```
例：BIS    #CAP,&CCTL2          ；用 CCR2 选择捕获模式
    XOR    #CCIS1,&CCTL2        ；软件捕获：CCIS0=0，捕获模式=3
```

10.6.4　Timer_A 中断向量寄存器

16 位 Timer_A 有 2 个中断向量：

- CCR0 中断向量，具有高优先级。
- TAIV 中断向量，用于 CCIFG1～CCIFGx 和 TAIFG 标志。

CCR0 中断向量

CCR0 中断向量与 CCR0 相关，如果定时器值等于比较寄存器的值时置位。具有最高的 Timer_A 的中断优先级，见图 10.27。

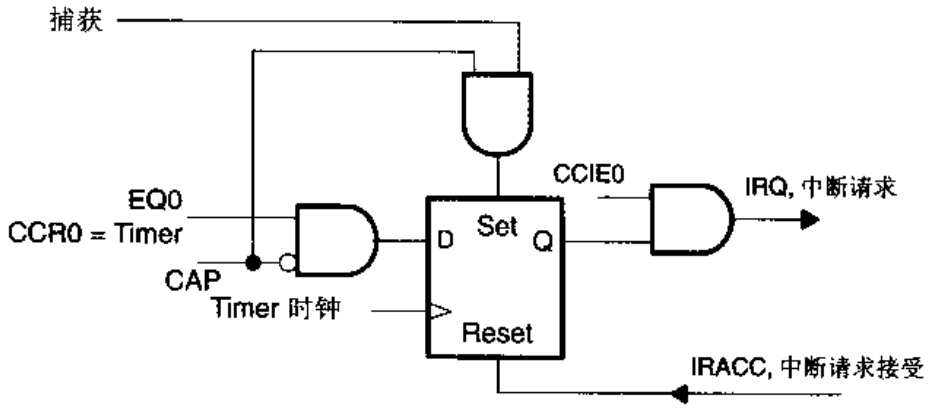


图 10.27　CCR0 中断标志

TAIFG，CCIFG1～CCIFG4 标志的中断向量

CCIFGx(不包括 CCIFG0)与 TAIFG 标志组成有优先级、共用一个中断向量的中断标志，见图 10.28。中断向量寄存器 TAIV 用于确定产生中断请求的中断源。

TAIV(12Eh)

15										5				0
0	0	0	0	0	0	0	0	0	0	0	中断向量			0
r0	r0	r0	r0	r0	r0	r0	r0	r0	r0	r0	r-(0)	r-(0)	r-(0)	r0

具有最高优先级的中断标志产生一个 2～14 的数存入 TAIV。优先级见表 10.4。

表 10.4　TAIV 中断优先级

中断优先级	中断源	缩　写	TAIV 内容
最高	捕获/比较 1	CCIFG1	2
	捕获/比较 2	CCIFG2	4
	保留		6
	保留		8
	定时器溢出	TAIFG	10
	保留		12
最低	保留		14
	没有中断挂起		0

如果 TAIV=0，则无请求服务的中断请求。TAIV 的值与 PC 相加可以自动进入相应的中断服务程序，而不必对 TAIV 作读出、判断的操作，见以下的软件实例。访问 TAIV 能自动

要的时钟周期。此程序是为连续模式写的，与下一中断的时间差已加入相应的比较寄存器。

```
; 例：中断程序                                                    周期
;
; 捕获/比较模块 0 的中断处理，中断标志 CCIFG0 能自动复位
;
TIMMOD0     …                          ; 开始中断处理            6
            RETI                                                 5
;
; 捕获/比较模块 1～2 的处理，中断标志 CCIFGx 和 TAIFG 由硬件复位
; 只有在响应时具有最高优先级的标志复位
;
TIM_HND     $                          ; 潜在的中断响应过程      6
            ADD     &TAIV,PC           ; 加跳转表偏移量          3
            RETI                       ; 向量 0：无中断          5
            JMP     TIMMOD1            ; 向量 2：模块 1          2
            JMP     TIMMOD2            ; 向量 4：模块 2          2
            JMP     TIMMOD3            ; 向量 6：保留
            JMP     TIMMOD4            ; 向量 8：保留
;
; 定时器溢出处理：定时器寄存器扩展入 RAM 中的 TIMEXT(高位)
;
TIMOVH                                 ; 向量 10：定时器溢出处理
            INC     TIMEXT             ;                         4
            RETI                                                 5
;
TIMMOD2                                ; 向量 4：模块 2
            ADD     #NN,&CCR2          ; 时间差累加              5
            …                          ; 处理部分
            RETI                       ; 返回主程序              5
;
TIMMOD1                                ; 向量 2：模块 1
            ADD     #MM,&CCR1          ; 时间差累加              5
            …                          ; 处理部分
            RETI                       ; 返回主程序              5
;
; 如果全部 CCR 寄存器都未置位，则中断向量仍要处理
;
TIMMOD4     RETI                       ; 简单返回主程序          5
;
; 模块 3 处理，检查是否有其他中断挂起
; 需用 5 个周期，但是如有中断挂起，可节省 9 个周期
;
TIMMOD3                                ; 向量 6：模块 3
```

```
                ADD     #PP,&CCR3          ; 时间差累加                    5
                ...                        ; 处理部分
                JMP     TIM_HND            ; 检查挂起中断                  2
;
                .SECT   "VECTORS",0FFF0h   ; 中断向量地址:0FFF0h 或 0FFEAh
;
; 各型号中断向量地址可能不同,F11x、F11x1 为 0FFF0h,F13x、F14x 为 0FFEAh
;
                .WORD   TIM_HND            ; 捕获/比较模块 1~2 及 TAIFG 的向量
                .WORD   TIMMOD0            ; 捕获/比较模块 0 向量
```

如果 CPU 时钟 MCLK 关闭(CPUOff=1),则还需要再增加 2～3 个额外时钟周期用于使 CPU 的启动同步。引起有 1 个时钟周期误差是因为 CPU 时钟 MCLK 的再启动与其他时钟信号是异步的。

对各个中断源、中断程序的基本时间开销包括潜在的中断响应和从中断返回的周期数(不包括处理任务本身)。

- 捕获/比较模块 CCR0　　11 个周期。
- 捕获/比较模块 CCR1～CCR2　16 个周期。
- 定时器溢出 TAIFG　　14 个周期。

定时极限

用 TAIV 寄存器和上述软件,利用一个比较寄存器,两个事件之间的最短重复时间是

$$t_{CKmin} = t_{taskmax} + 16 \times T_{cycle}$$

其中:$t_{taskmax}$　最长时间(最差情况),在中断程序完成任务,如计数器加 1。

T_{cycle}　MCLK 时钟周期。

利用一个捕获寄存器时两个事件之间的最短重复时间是

$$t_{CLmin} = t_{taskmax} + 16 \times T_{cycle}$$

10.7 Timer_A 的 UART 应用

Timer_A 具有独特的实现 UART 的能力,有以下特点:

- 自动实现起始位检测,甚至可以工作在超低功耗模式。
- 硬件实现波特率发生。
- 硬件锁存 RXD 和 TXD 数据。
- 波特率为 75～115 200。
- 全双工。

UART 的实现在各种不同类型的微处理机系统中都不同,可能是用通用的 I/O 端口以软件查询实现位的处理。这样的处理方式需要极大的 CPU 开销,因此增加了功耗,也降低了 CPU 的可用性。

发送特性的实现,用比较功能将数据从输出单元的引脚移出的方法。波特率用比较数据及中断来获得。

接收特性的实现，用捕获/比较功能将引脚数据经 SCCIx 位移入内存。接收开始的时间用捕获输入引脚下降沿的定时器值来识别。然后，同一个捕获/比较模块切换成比较模式，接收的位，用 EQUx 信号自动锁存。由中断服务程序收集各位，供后续的软件处理。

图 10.29 是 UART 的实现原理。

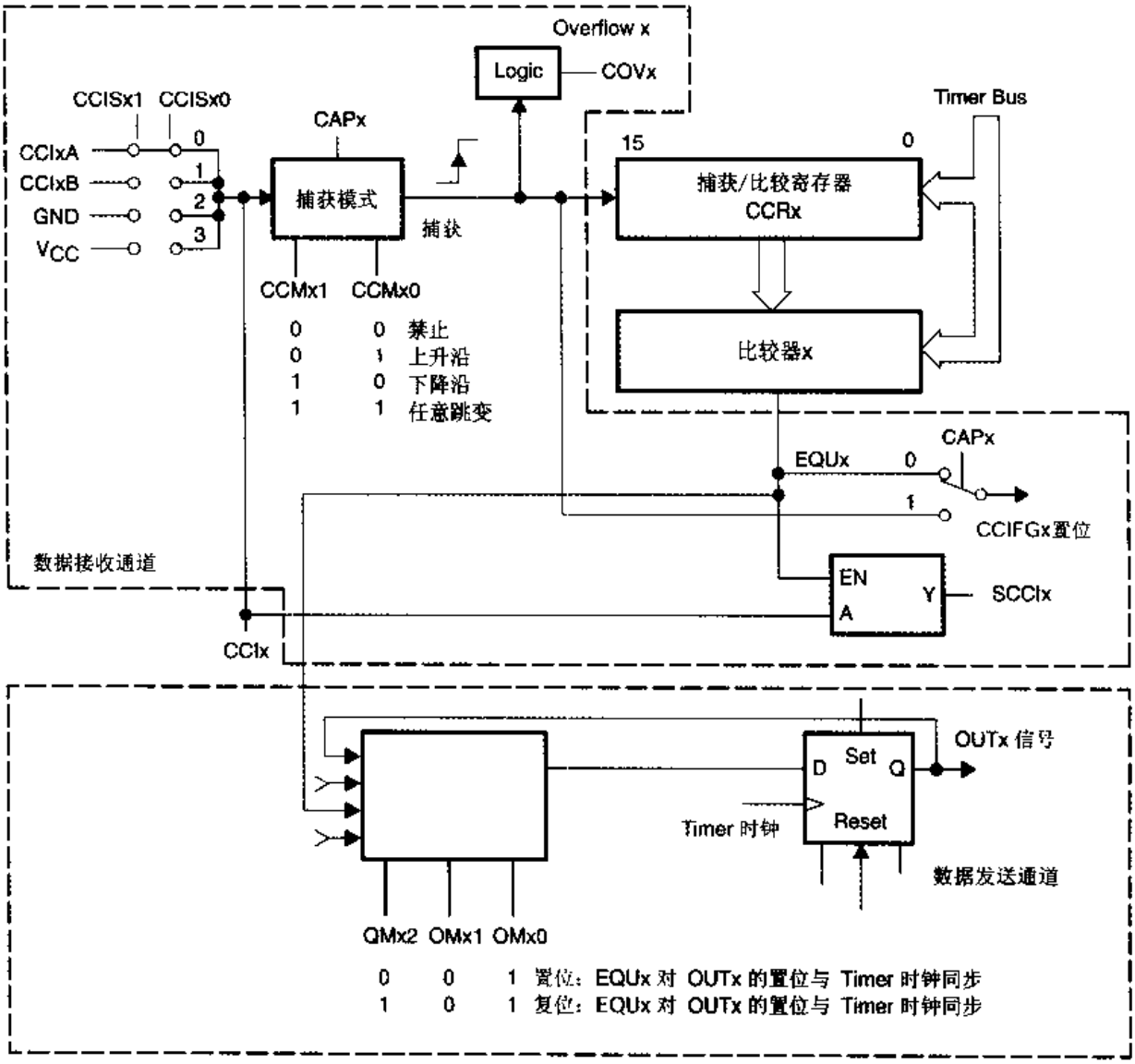


图 10.29　UART 的实现原理

实现半双工只需用 1 个捕获/比较模块，实现全双工则需要 2 个捕获/比较模块。

图10.30 是实现 UART 的时序。

完整的实现方案，包括线路图及完整的软件清单可以从网站中得到。

网址如下：

www. ti. com/sc/MSP430

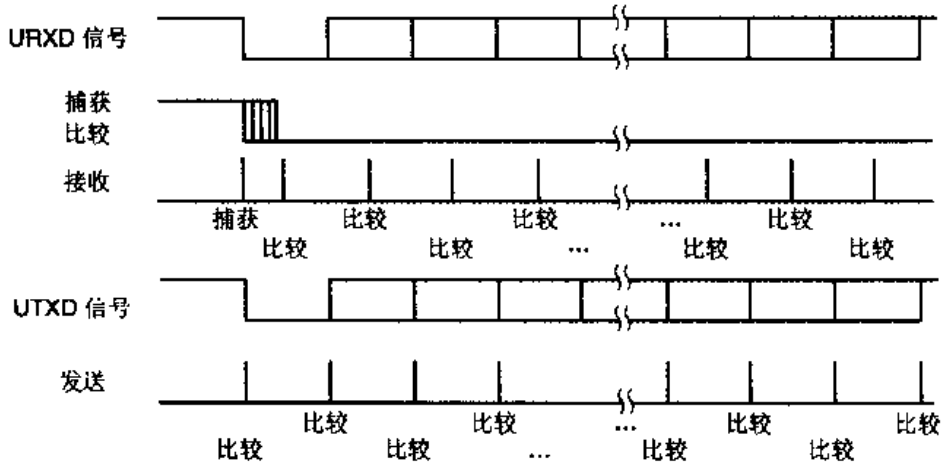


图 10.30 Timer_A 实现 UART 的时序

第 11 章　16 位定时器 Timer_B

外围模块 Timer_B 在 MSP430 的各个不同型号中的实现是不同的。在 F11x、F11x1 中，无 Timer_B。在 F13x 中有一个带有 3 个捕获/比较模块的 Timer_B，在叙述中常用 Timer_B3 来表示。在 F14x 中有一个带有 7 个捕获/比较模块的 Timer_B，在叙述中用 Timer_B7 表示。

Timer_B 与 Timer_A 的功能类似，只是在 Timer_B 中未实现 SCCI 位。

11.1　引　言

Timer_B 是非常有用的定时器，它有以下特点：

- 具有 4 种工作模式的 16 位计数器，可选 8、10、12 或 16 位计数。
- 可以选择的计数器时钟源。
- 3 个或 7 个具有可配置输入端的捕获/比较寄存器。
- 有 8 种输出模式的 3 个或 7 个可配置输出单元。

Timer_B 可支持多个同时进行的时序控制、多个捕获/比较功能、多种输出波形(PWM 波形)，也可以是上述功能的组合。此外，由于具有数据的双缓存能力，多个 PWM 周期可以同步地更新。

Timer_B 具有中断能力。中断可以由计数器溢出引起，也可以来自有捕获或比较功能的捕获/比较寄存器。每个捕获/比较模块可以独立编程，由比较或捕获外部信号来产生中断。外部信号可以是信号的上升沿、下降沿或所有跳变。

图 11.1 是 Timer_B 的结构。

Timer_B 与 Timer_A 几乎相同，但是功能有一些增强，有以下主要差异：

- Timer_B 的计数长度可以编程为 8、10、12 或 16 位，Timer_A 只能是 16 位。
- Timer_B 没有实现 Timer_A 中的 SCCI 位功能。
- Timer_B 在比较模式下的捕获/比较寄存器功能与 Timer_A 稍有不同。
- 有些型号芯片的 Timer_B 输出实现了高阻态，请查阅芯片手册。

在 Timer_A 中，CCRx 寄存器保存与定时器比较的数据。在 Timer_B 中，CCRx 是比较数据的一个锁存缓存，由比较器锁存作比较的数据。在这两种定时器中，比较数据都是写入 CCRx，但是在 Timer_B 中，比较数据要传送到比较器的锁存器才作比较。比较数据从 CCRx 寄存器传送到相应的比较锁存器(TBCLx)的时序是用户可选择的，可以立即进行，也可以在发生定时器事件时进行。

增加的比较数据锁存器给用户提供了对需要精确的比较周期的控制。也可以使多个比较器成组工作，同时发生比较器的更新。例如，对于多个 PWM 信号同步改变占空比的应用是很有用的。

应记住，Timer_B 在缺省状态下，CCRx 的比较数据是立即传送给相应的比较器的。因此，在缺省情况下，Timer_B 的比较模式与 Timer_A 是相同的。

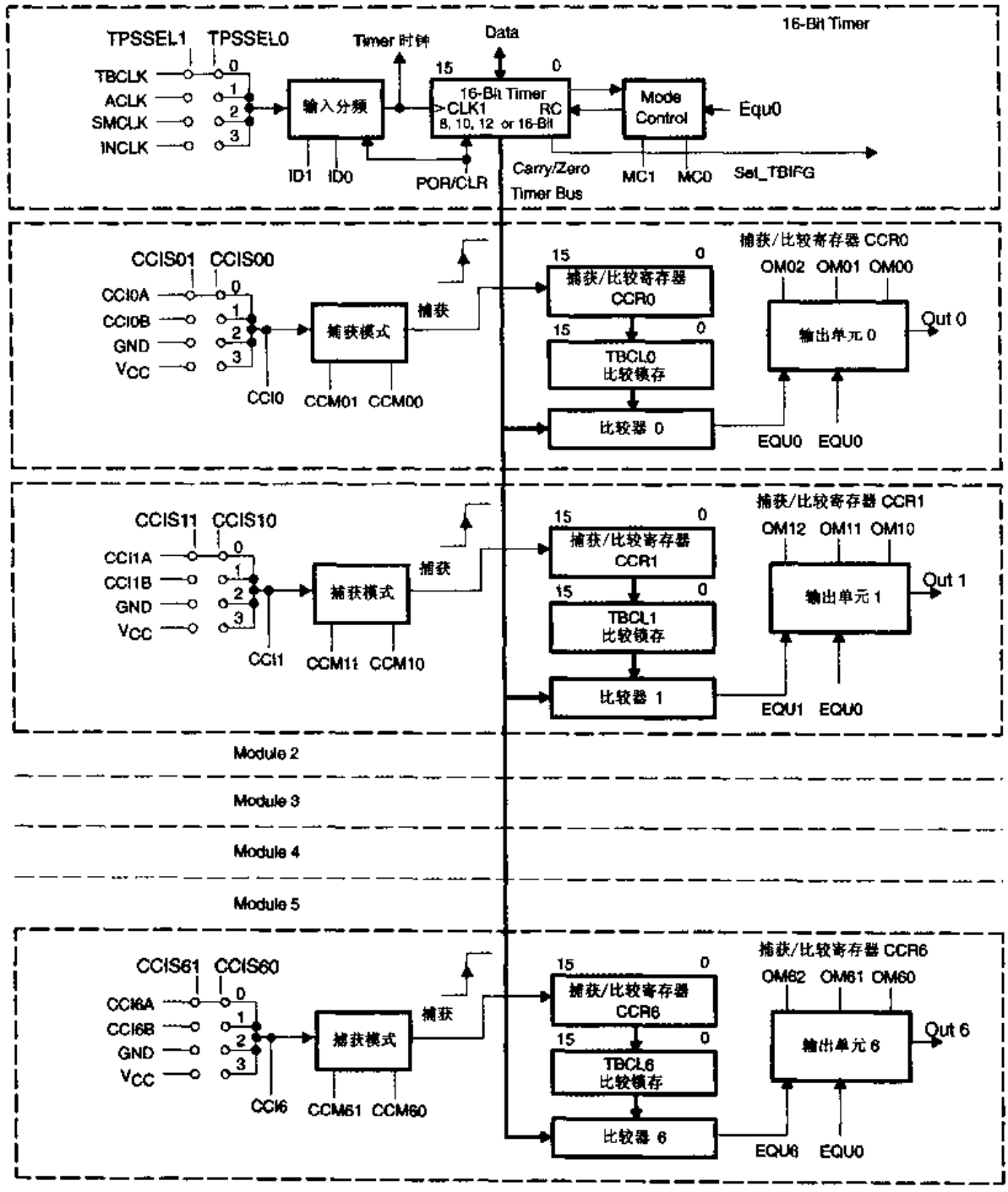


图 11.1 Timer_B 结构

11.2 Timer_B 的操作

通过定义控制寄存器 TBCTL 中的控制位 MC1 和 MC0 可以选择 16 位定时器工作在 4 种模式。定时器在时钟信号的上升沿增加或减少，取决于选择的工作方式。定时器可以用软件读写。当定时器发生溢出时会产生中断。

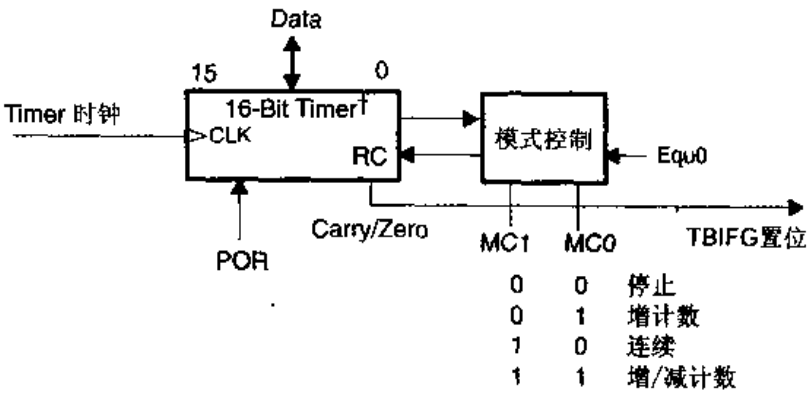
11.2.1　定时器长度

Timer_B 可以用 TBCTL 寄存器配置成真 8 位、10 位、12 位或 16 位工作。在 8、10、12 位情况下，前导各位读出为“0”。数据写入 TBR，前导各位也为“0”。不同模式下最大的计数长度如下：

Timer_B 配置	最大计数长度	Timer_B 配置	最大计数长度
16 位	0FFFFh	10 位	03FFh
12 位	0FFFh	8 位	0FFh

11.2.2　定时器模式控制

图 11.2 和表 11.1 说明了定时器的 4 种工作模式：停止、增计数、连续、增/减计数。工作模式由 TBCTL 中的 MC0 和 MC1 控制位选择。



† 计数长度可以选择 8、10、12 或 16 位。

图 11.2　模式控制

表 11.1　4 种工作模式说明

模式控制		模　式	说　明
MC1	MC0		
0	0	停止	定时器暂停
0	1	增计数	定时器增计数至与比较寄存器 TBCL0 的值相等。 如果 TBCL0 大于最大计数长度，在定时器时钟的下一个上升沿回到“0”
1	0	连续	定时器连续增计数。 最大计数长度为 16 位：0FFFFh　　12 位：0FFFh 10 位：03FFh　　8 位：0FFh
1	1	增/减计数	定时器增计数至 TBCL0，然后减计数至“0”。 如果 TBCL0 大于最大计数长度，则定时器的工作与连续模式相同，不会从最大计数长度开始减计数

11.2.3 时钟源选择和分频

定时器的时钟源可以来自内部时钟(ACLK、MCLK 或 SMCLK)或外部时钟(TBCLK),见图 11.3。时钟源由 TBCTL 中的 SSEL0 和 SSEL1 位来选择。必须注意,改变定时器的时钟源可能发生不确定的时序。因此,选择时钟源之前应停止定时器的工作。

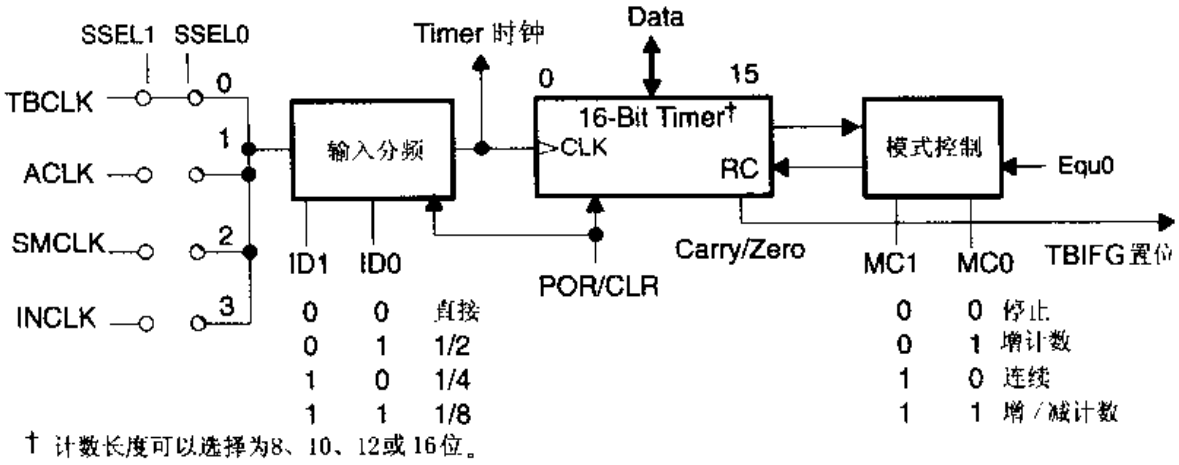


图 11.3 16 位定时器结构

图 11.4 是定时器的时钟源的选择和分频。被选择的时钟信号经过分频器的 1、2、4 或 8 分频送入定时器。TBCTL 中的 ID0、ID1 位选择时钟的分频因子。发生 POR 或者将 TACTL 中的 CLR 位置位使输入分频器复位。除此之外,定时器在变化时,分频器保持不变,并且分频器的状态对于软件是不可见的。

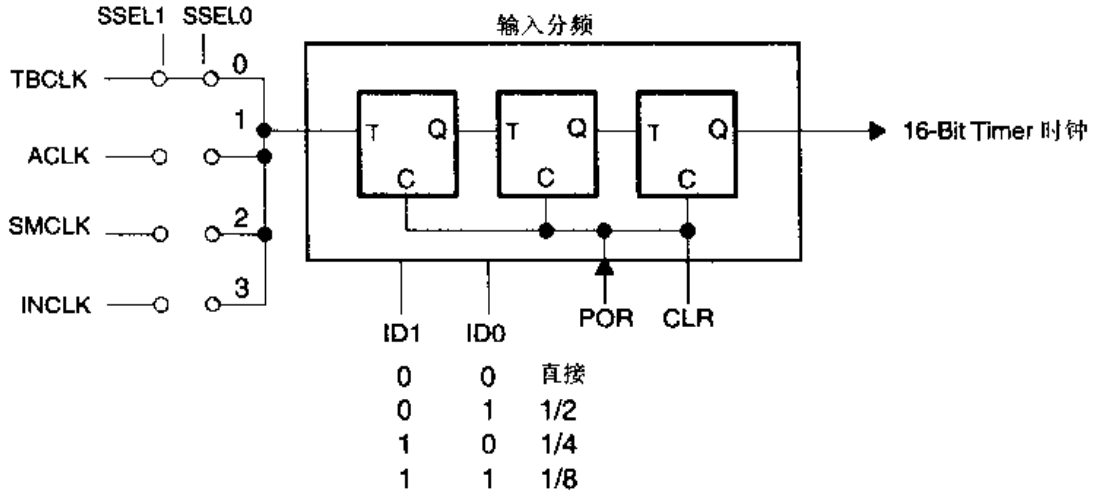


图 11.4 时钟源及分频器

11.2.4 定时器启动

定时器可以用多种方式启动或再启动。

- 暂停状态的释放:用 MC 位选择除停止模式以外的模式。定时器在选定的计数方向上开始计数。

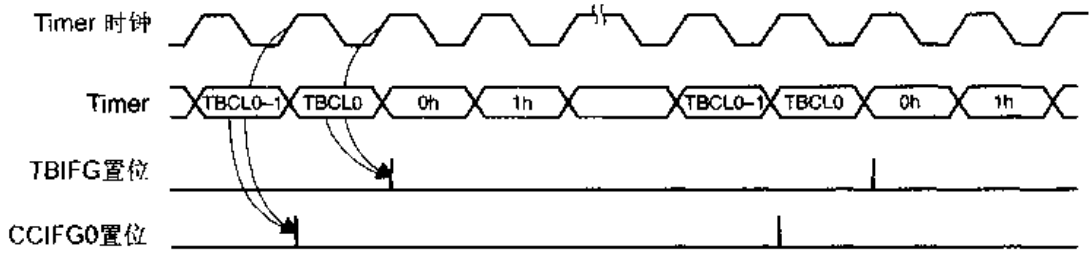


图 11.6 增计数模式标志设置

图 11.7。当新周期小于老周期，更新 TBCL0 时的定时器时钟相位将影响定时器如何响应新周期。

如果在定时器时钟为高时从 CCR0 传送给 TBCL0 新的小周期值，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新的小周期值，则定时器在接受新周期并返回到"0"之前，继续增加一个时钟周期，见图 11.8。

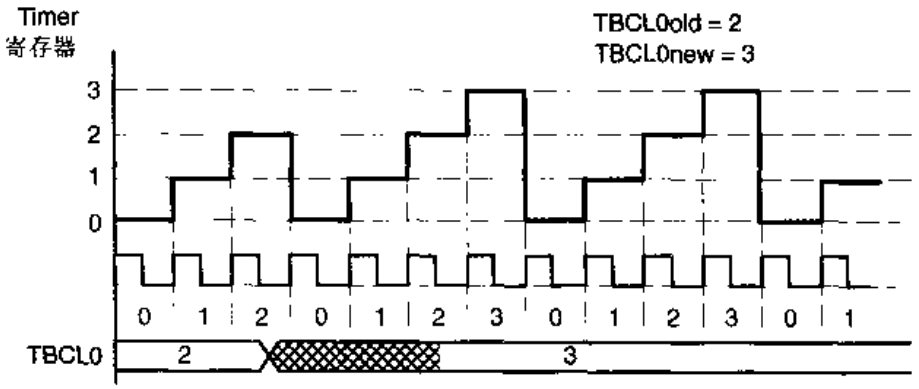
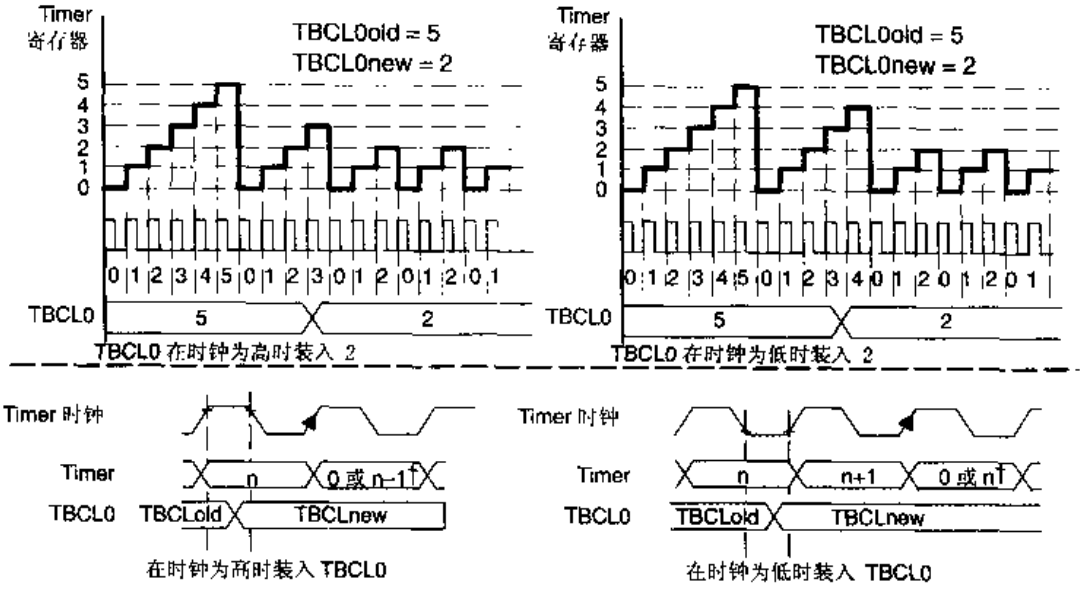


图 11.7 新周期大于老周期



† 增计数模式：0；增/减计数模式：n-1。 † 增计数模式：0；增/减计数模式：n。

图 11.8 新周期小于老周期

11.3.3　连续模式

连续模式用于需要最大计数长度时钟周期的应用场合。典型应用是产生多个独立的时序信号。在连续模式中，CCR0 的工作方式与其他比较寄存器相同。

利用捕获/比较寄存器和各输出单元的各种输出模式可以捕获各种外部事件发生的定时器数据或者产生不同类型的输出信号。

在连续模式，定时器从它的当前值开始计数。当计数到最大计数长度后又从“0”开始重新计数，见图 11.9。连续模式各种配置下的最大计数长度为：0FFFFh(16 位)、0FFFh(12 位)、03FFh(10 位)和 0FFh(8 位)。

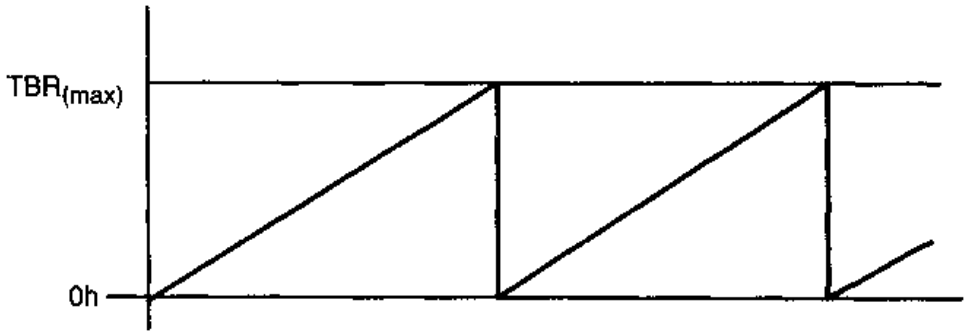


图 11.9　连续模式

当定时器从最大计数长度计数到“0”时，TBIFG 标志置位。中断标志会根据各自的中断事件置位，只有相应的中断允许位及 GIE 位都置位时，中断请求才会发生。图 11.10 是标志设置过程。

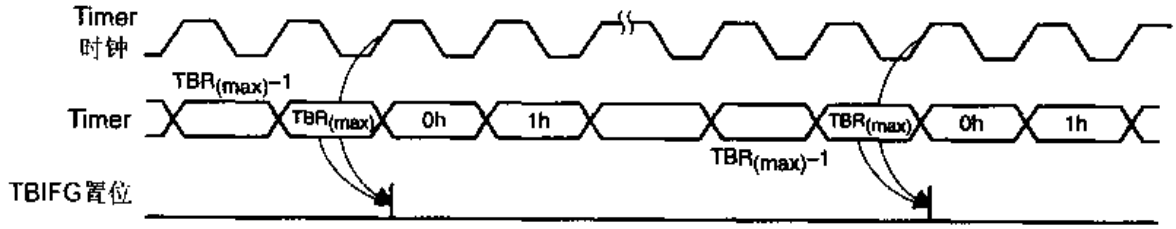


图 11.10　连续模式标志设置

连续模式可用于在应用软件中得到定时间隔。每当一个定时间隔到会产生中断请求。在这一事件的中断服务程序中可以将下一个事件发生的时间加到 CCRx(然后加到 TBCLx)上，见图 11.11。用全部 7 个(或 3 个)捕获/比较模块可以产生 7 个(或 3 个)独立的时间事件。

在 CCR0 用作周期寄存器的其他模式，也可以用相同方法产生定时间隔，但是这时的处理会变得很复杂。CCRx 的数据和新的周期相加可能大于 TBCL0 的值。当出现这种情况时必须减去 CCR0 的值才能得到正确的定时间隔。

求这一点，则应该用 TBCTL 中的 CLR 位来清除计数方向。但是要注意，CLR 位同时也会影响其他已建立的定时器状态。

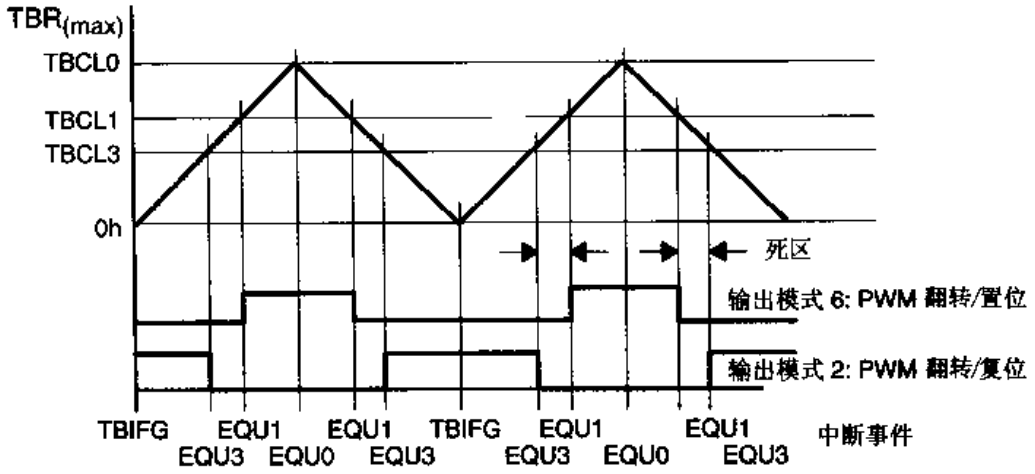


图 11.13　增/减计数模式的输出

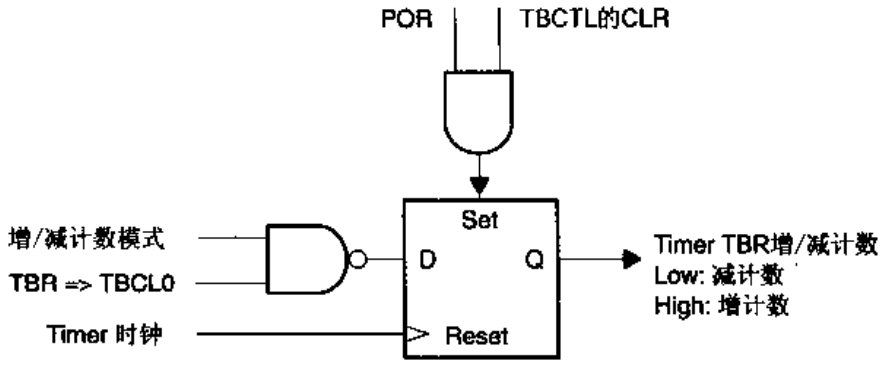


图 11.14　增/减计数方向控制

在增/减计数模式，中断标志 CCIFG0 和 TBIFG 会在相等的定时间隔置位，见图 11.15。在一个完整的周期中，每个标志只置位一次，分别在半周期时发生。当定时器从 TBCL0－1 增计数到 TBCL0 时，中断标志 CCIFG0 置位；当定时器从 0001h 减计数到 0000h 时，中断标志 TBIFG 置位。如果中断允许，每个中断标志都会产生中断请求。

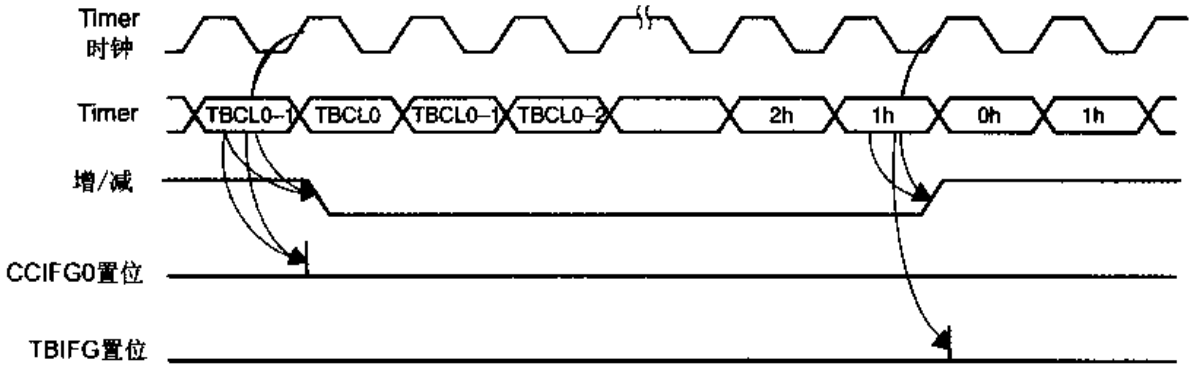


图 11.15　增/减计数模式的标志

在增/减计数模式中，如果在计数过程中改变周期值，那么定时器会比增计数模式更难以

捉摸。不但如增计数模式那样，TBCL0 改变时的时钟相位会影响定时器的运行，而且增/减计数的方向也会影响定时器的运行。

如果定时器处于增计数时由 CCR0 向 TBCL0 写入新周期值，那么这时定时器的表现与增计数模式中完全一样。但是，如果定时器处于减计数时 TBCL0 写入新周期值，则这时定时器将继续减计数至“0”。新周期只有在完成减计数后才会起作用，见图 11.16。

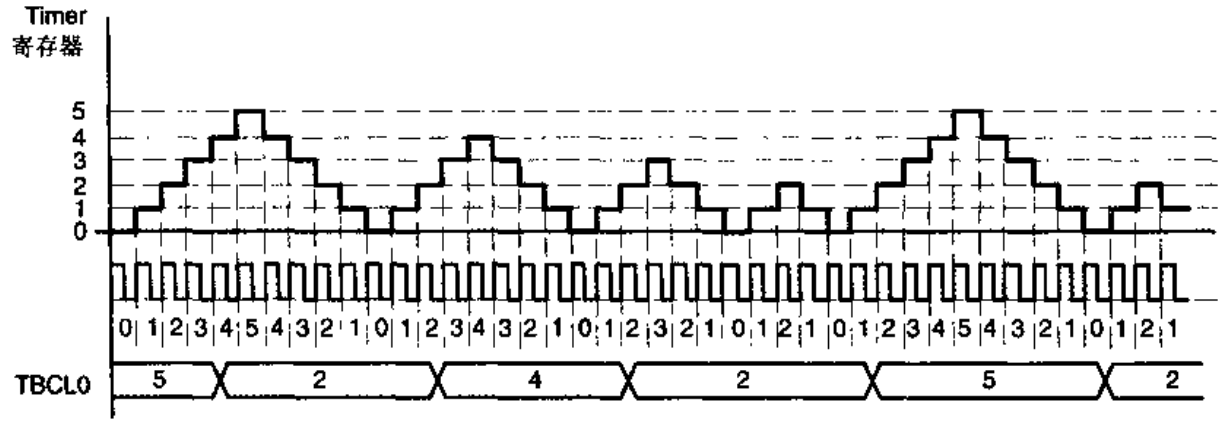


图 11.16 增/减计数模式中修改 TBCL0

11.4 捕获/比较模块

7 个(或 3 个)相同的捕获/比较模块为实时处理提供了灵活的控制手段。每个模块都可用于捕获事件发生时间或产生定时间隔。每发生一次捕获或一次定时间隔，捕获/比较模块寄存器将产生中断。控制字 CCTLX 中的模式位 CAPx 选择比较或捕获模式。控制字 CCTLX 中的 CCMx1 和 CCMx0 定义产生捕获功能的条件。图 11.17 是捕获/比较模块的结构。

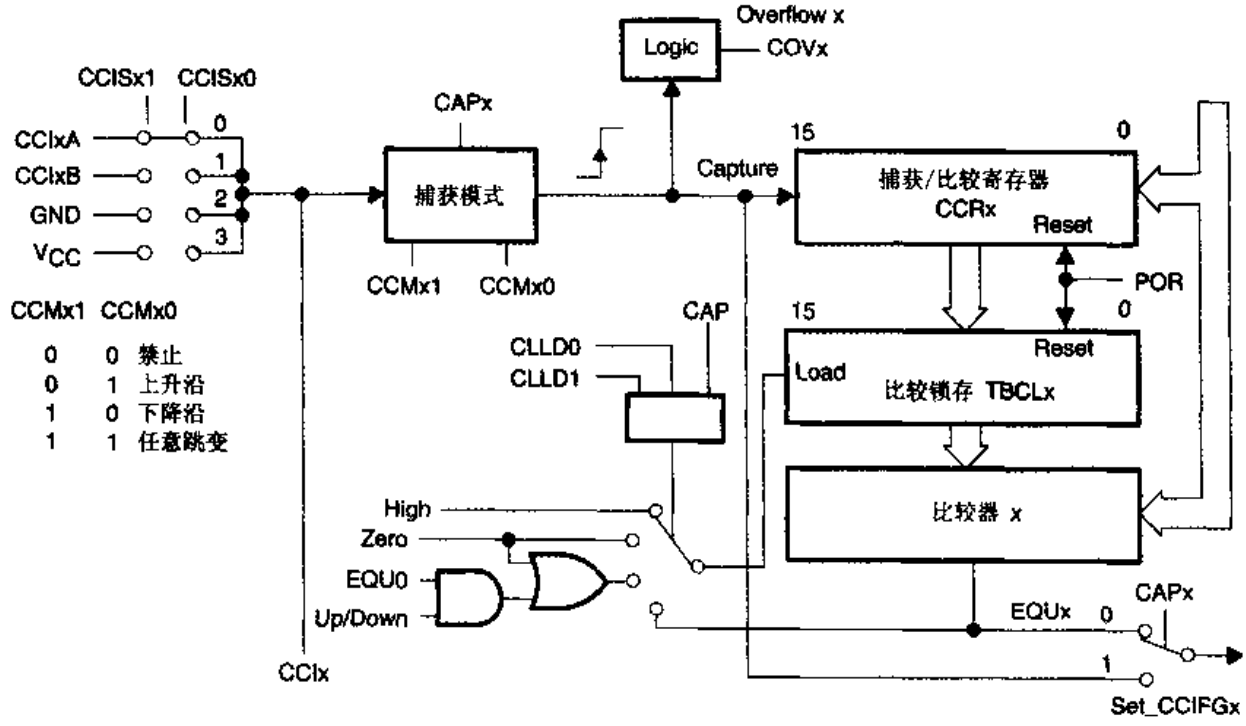


图 11.17 捕获/比较模块结构

当应用低速定时器时钟时，可以采用异步的信号捕获。下例说明如何用软件验证捕获数据及加以校正。

例：软件处理异步捕获信号

```
;
; 捕获/比较寄存器数据在相应的 CCIFG 中断处理程序中读取
; 定时器时钟大大慢于系统时钟 MCLK
;
CCRx_Int_hand   …                          ; 中断处理开始
                …
                CMP     &CCRx,&TBAR        ; 数据检查:CCRX = TBR
                JEQ     Data_Valid
                MOV     &TBR,&CCRx         ; CCRx 中数据错,用定时器数据
Data_Valid      …                          ; CCRx 中数据对
                …
                RETI
```

每个捕获/比较寄存器提供溢出信号，告诉用户在前一次的捕获数据读出之前已开始新的一次捕获。用控制字 CCTLx 中的 COVx 位的置位说明这一状况，见图 11.20。

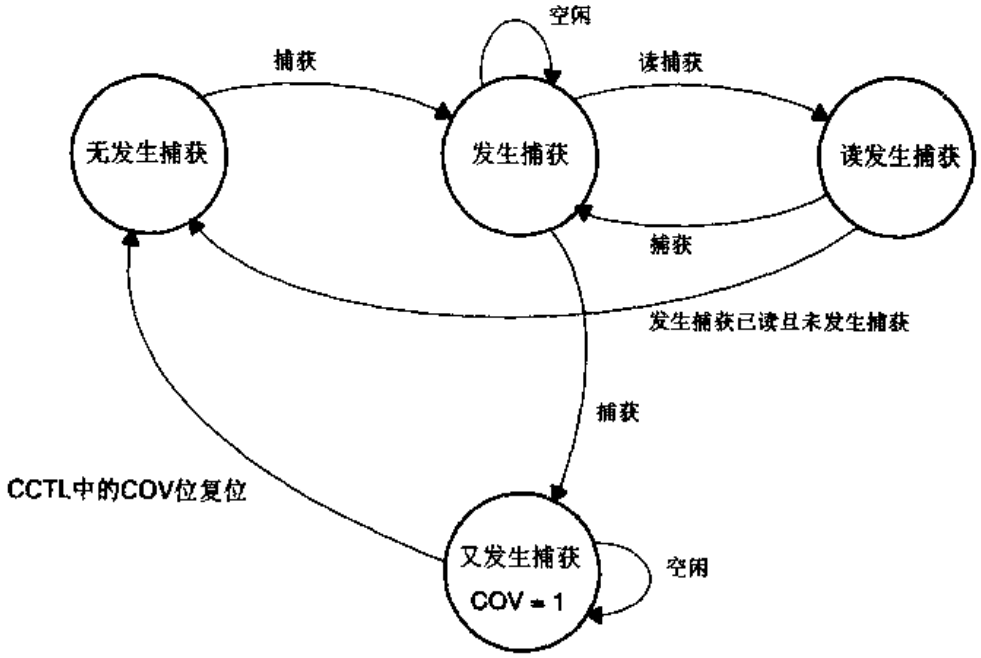


图 11.20 捕获状态循环

下例说明溢出位 COVx 用软件复位。

例：捕获数据发生溢出的软件处理

```
;
; 捕获/比较寄存器 CCRx 用软件读取后,下一条指令立即测试溢出位
; 并决定处理的对策
;
```

```
CCRx_Int_hand   …                                   ; 中断服务程序开始
                MOV     &CCRx,RAM_Buffer
                BIT     #COV,&CCTLx
                JNZ     Overflow_Hand
                …
                RETI
Overflow_Hand   BIC     #COV,&CCTLx                 ; 捕获溢出位复位
                                                    ; 恢复丢失的同步
                …
                RETI
```

注意：捕获与定时器暂停

当定时器暂停时捕获应该停止，顺序应是先停止捕获功能，再停止定时器计数；捕获功能重新开始时，顺序是先开始捕获功能，再开始定时器计数。

捕获可以用软件初始化，这对于以下情况是有用的。

- 用软件测量时间。
- 测量 2 个硬件事件之间的时间。
- 测量系统时钟频率。

软件初始化捕获用到 CCISx1、CCISx0 位和捕获模式选择位 CCMx1、CCMx0。最简单的实现例子为：捕获模式选择为任意跳变沿，CCISx1 位置位，在程序中将 CCISx0 在 V_{CC} 和 GND 之间切换，每次切换即发生捕获，见图 11.21。

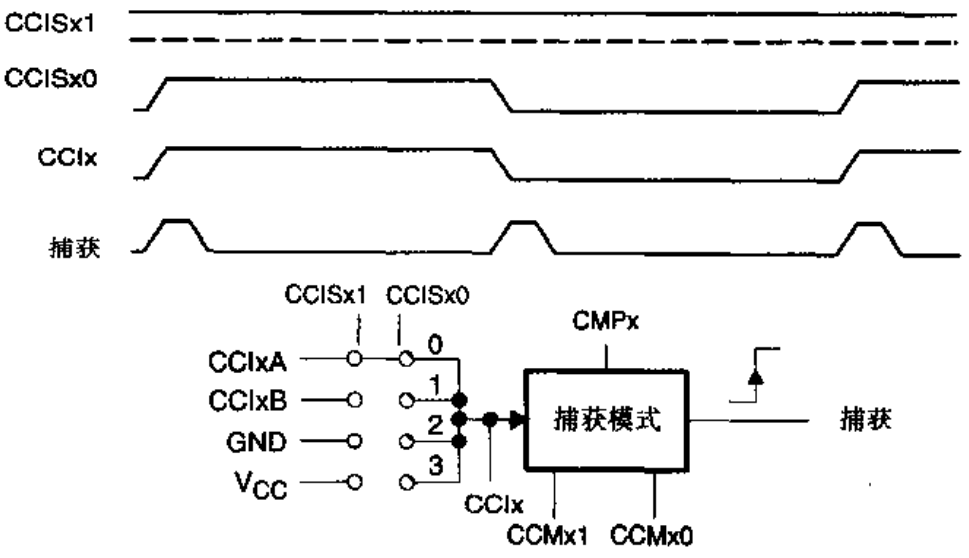


图 11.21　软件捕获举例

例：软件实现捕获

```
; 捕获/比较寄存器 CCRx 由软件读入
; 设 CCMx1、CCMx0 和 CCISx1 置位，CCIS0 选择 CCIx 信号为高或低
;
```

```
...
XOR     #CCISx0, &CCTLx
...
```

11.4.2 比较模式

如果控制字 CCTLx 中的 CAPx 复位则选择比较模式，这时所有捕获硬件停止工作，捕获模式溢出逻辑也不工作。

比较模式常用于在设定的定时间隔产生中断，或者结合输出单元发生如 PWM 波形这样的信号。

比较数据是双重缓存的。软件将比较数据写入捕获比较寄存器，但是数据传送入 TBCLx 作比较由比较逻辑实现。比较数据由 CCRx 传送入比较逻辑可由用户选择，可以立即方式，也可以是由定时器事件触发。这种双重缓存结构，可以实现多比较值的同步更新，这对于多个 PWM 信号要求同步更新周期或占空比是很有用的。

如果定时器值等于 TBCLx 中的值，那么：

- 位于控制字 CCTLx 中的中断标志 CCIFGx 置位。
- 如果 GIE 和 CCIEx 置位，则将产生中断请求。
- EQUx 信号输出到输出单元中。信号根据选定的输出模式影响 OUTx。

当定时器数据大于等于 TBCL0 的值时，EQU0 信号为真。当定时器数据等于相应的 TBCL1～TBCLx 的值时，EQU1～EQUx 信号为真。

比较逻辑是用比较锁存器中数据与定时器值比较的。比较数据先用软件写入捕获/比较寄存器 CCRx，然后以用户选择的加载事件方式自动装入比较锁存器。加载事件由每个 CCTLx 寄存器的 CLLDx 位选择。

比较锁存器可以成组工作，同组的比较锁存器可以在发生加载事件时同步更新。全部比较锁存器可以组成同一个组，也可以分成二三个组。分组由 TBCTLx 寄存器中的 TBCLGRP 位配置。分组后，同组中序号最小的 CCRx 的 CLLDx 位确定全组的加载事件，包括 7 个比较锁存器在同一组的情况(TBCLGUP=3)。例如，用户将比较寄存器分成 3 组。其中，TBCL1、TBCL2 和 TBCL3 为一组，TBCL4、TBCL5 和 TBCL6 为一组。在这种情况下，TBCL1 的 CLLDx 位确定第一组的加载事件，TBCL4 的 CLLDx 位确定第二组的加载事件，CCTL2、CCTL3、CCTL5 和 CCTL6 寄存器的 CLLDx 位不起作用。当全部比较寄存器组成一组(TBCLGRP=3)时，TBCL1 的 CLLDx 位确定加载事件。

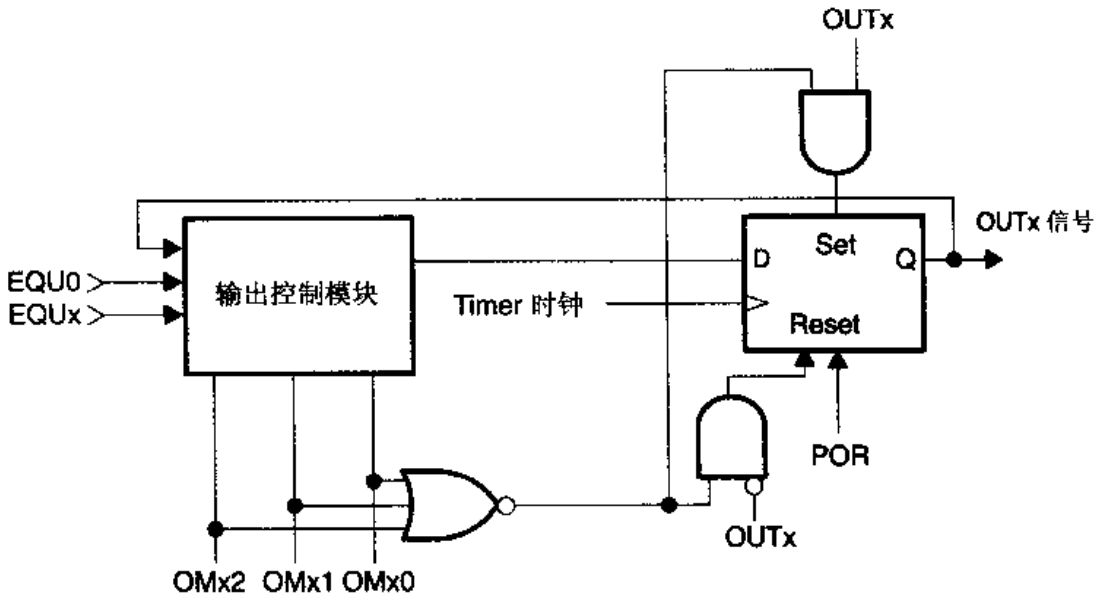
如果采用分组方法，则需要加载的比较寄存器必须满足两个条件。第一，同组的 CCRx 寄存器可以被更新(除非采用立即方式)；第二，加载事件必然发生。这意味着，用户企图在组更新时保留任一 CCRx 寄存器的数据，原数据必须重新写入 CCRx 寄存器，否则比较锁存器将不更新。

起确定作用的 CCTLx 寄存器的 CLLDx 位选择加载事件。有 4 种可选择的加载事件：

- 立即方式。
- TBR 计数到“0”。
- 在连续模式或增计数模式下，TBR 计数到“0”；
  在增/减计数模式下，TBR 计数到 TBCL0 或“0”。
- TBR 计数到 TBCLx。

11.5　输出单元

每个捕获/比较模块包含一个输出单元，见图 11.22。输出单元用于发生输出信号，如 PWM 波形。每个输出单元有 8 种工作模式，可以产生基于 EQU0 和 EQUx 状态的多种信号。输出模式由 CCTLx 中的控制位 OMx 选择。



0	0	0	输出：OUTx信号影响OUTx位；
0	0	1	置位：OUTx信号影响EQUx；
0	1	0	PWM 翻转/复位：EQUx使OUTx翻转，EUQ0使OUTx复位；
0	1	1	PWM 置位/复位：EQUx使OUTx置位，EQU0使OUTx复位；
1	0	0	翻转：EQUx使OUTx翻转；
1	0	1	复位：EQUx使OUTx复位；
1	1	0	PWM 翻转/置位：EQUx使OUTx翻转，EQU0使OUTx置位；
1	1	1	PWM 复位/置位：EQUx使OUTx复位和置位。

注：除模式0外，OUTx 信号均在 Timer 时钟上升沿更新，模式 2、3、6和7 对输出单元 UNIT0 无用。

图 11.22　输出单元

11.5.1　输出模式

除了输出模式 0，所有的输出都在定时器时钟的上升沿发生变化。输出模式 2、3、6 和 7 不适合输出单元 0。OMx 定义的输出模式如下：

输出模式 0：输出模式。

输出信号 OUTx 由每个捕获/比较模块中控制寄存器 CCTLx 的 OUTx 位定义，输出信号 OUTx 在写入位信息后立即更新。

输出模式 1：设置模式。

输出在定时器值等于 TBCLx 时置位。输出保持置位直至定时器复位或选择另一种输出模式。

输出模式 2：PWM 翻转/复位模式。

输出电平在定时器值等于 TBCLx 时翻转，当定时器值等于 TBCL0 时复位。

输出模式 3：PWM 置位/复位模式。

输出在定时器值等于 TBCLx 时置位，当定时器值等于 TBCL0 时复位。

输出模式 4：翻转模式。

输出电平在定时器值等于 TBCLx 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。

输出在定时器值等于 TBCLx 时复位，输出保持复位直至选择另一种输出模式。

输出模式 6：PWM 翻转/置位模式。

输出在定时器值等于 TBCLx 时翻转，当定时器值等于 TBCL0 时置位。

输出模式 7：PWM 复位/置位模式。

输出在定时器值等于 TBCLx 时复位，当定时器值等于 TBCL0 时置位。

11.5.2 输出控制模块

输出控制模块预备好 OUTx 的值，在定时器时钟的下一个上升沿锁存入 OUTx 触发器，见表 11.3 和图 11.23。相等信号 EQUx 和 EQU0 在定时器时钟的低电平采样。

表 11.3 定时器时钟下一个上升沿的 OUTx 状态

Mode	EQU0	EQUx	OUTx 状态
0	x	x	CCTLx 的 OUTx 位
1	x	0	OUTx 不变
	x	1	1
2	0	0	OUTx 不变
	0	1	OUTx 翻转
	1	0	0
	1	1	1
3	0	0	OUTx 不变
	0	1	1
	1	0	0
	1	1	1 置位
4	x	0	OUTx 不变
	x	1	OUTx 翻转
5	x	0	OUTx 不变
	x	1	0
6	0	0	OUTx 不变
	0	1	OUTx 翻转
	1	0	1
	1	1	0
7	0	0	OUTx 不变
	0	1	0
	1	0	1
	1	1	0

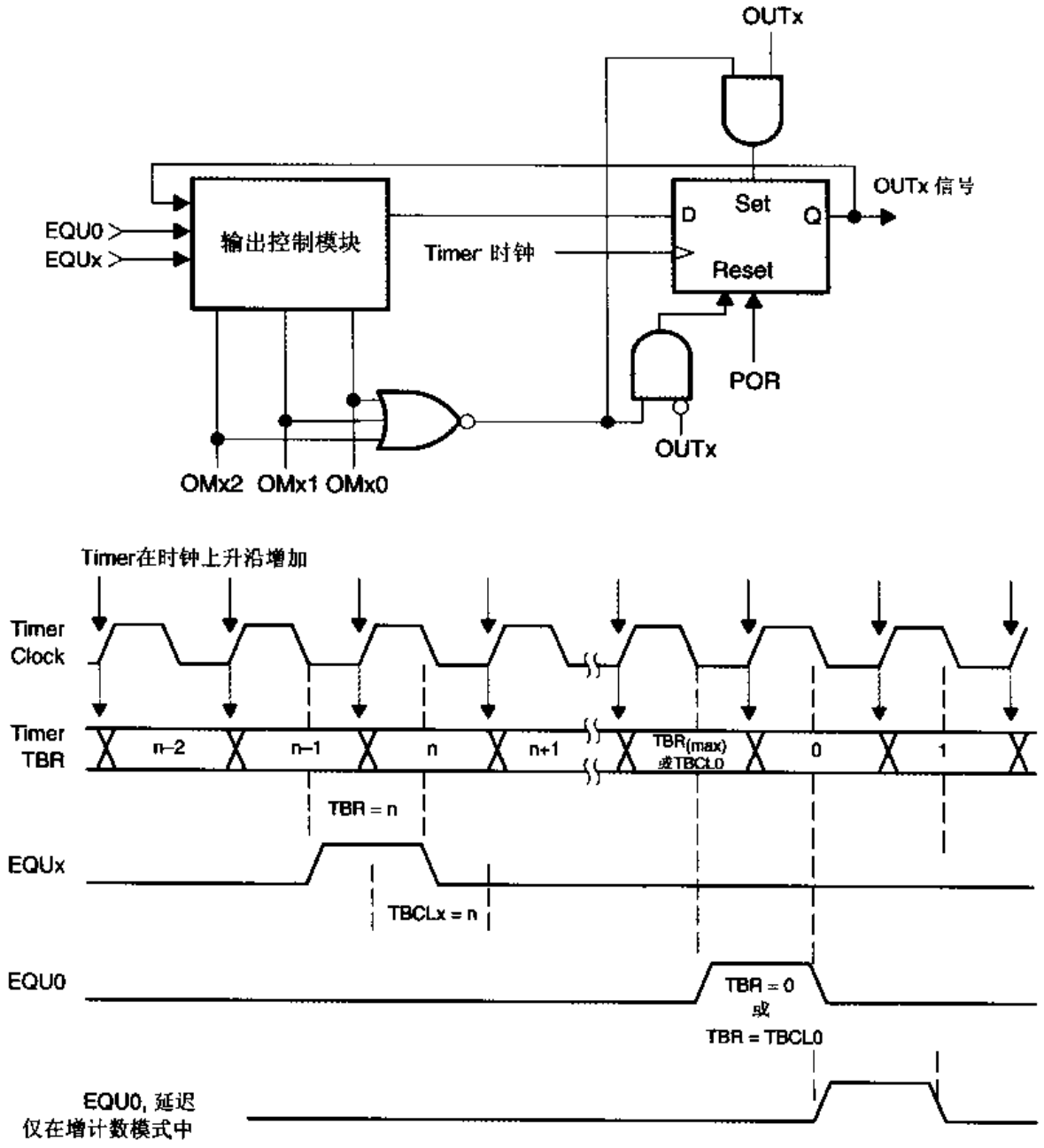


图 11.23　输出控制模块及时序

11.5.3　输出举例

以下是一些利用各种不同类型定时器模式和输出模式的实例。

增计数模式实例

当定时器计数到 TBCLx 或从 TBCL0 计数到“0”时，OUTx 信号按选择的输出模式发生改变，见图 11.24。

连续模式实例

当定时器计数到 TBCLx 和 TBCL0 时，OUTx 信号按选择的输出模式发生改变，见图 11.25。

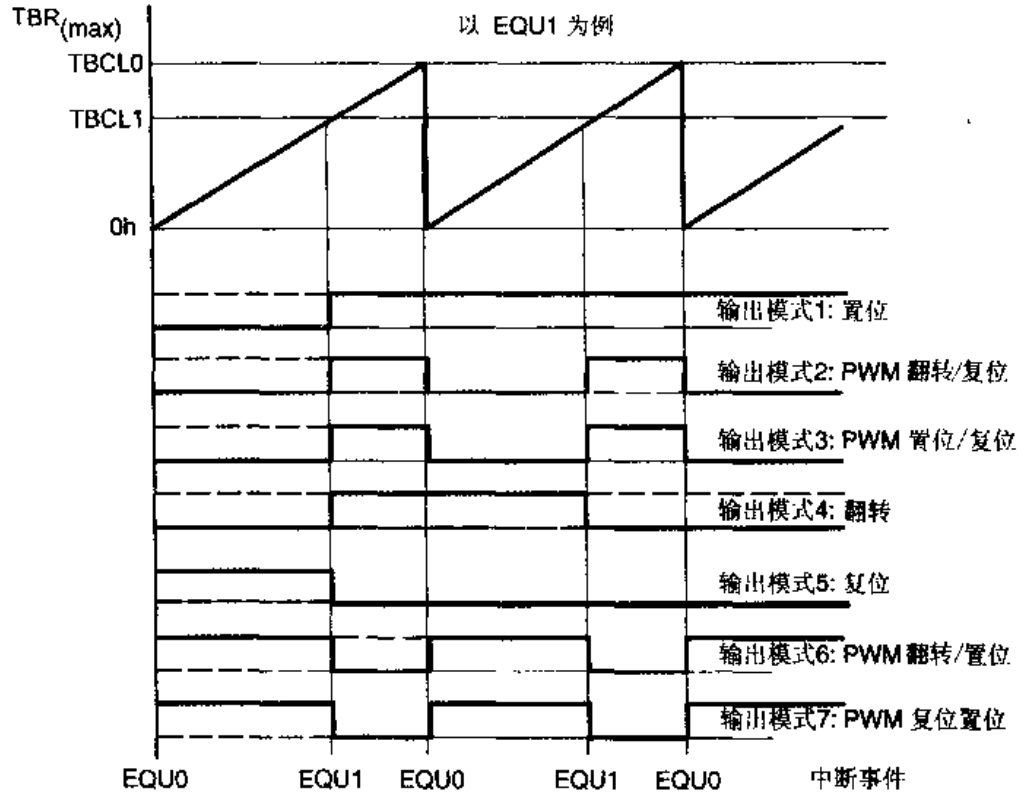


图 11.24 增计数模式的输出实例

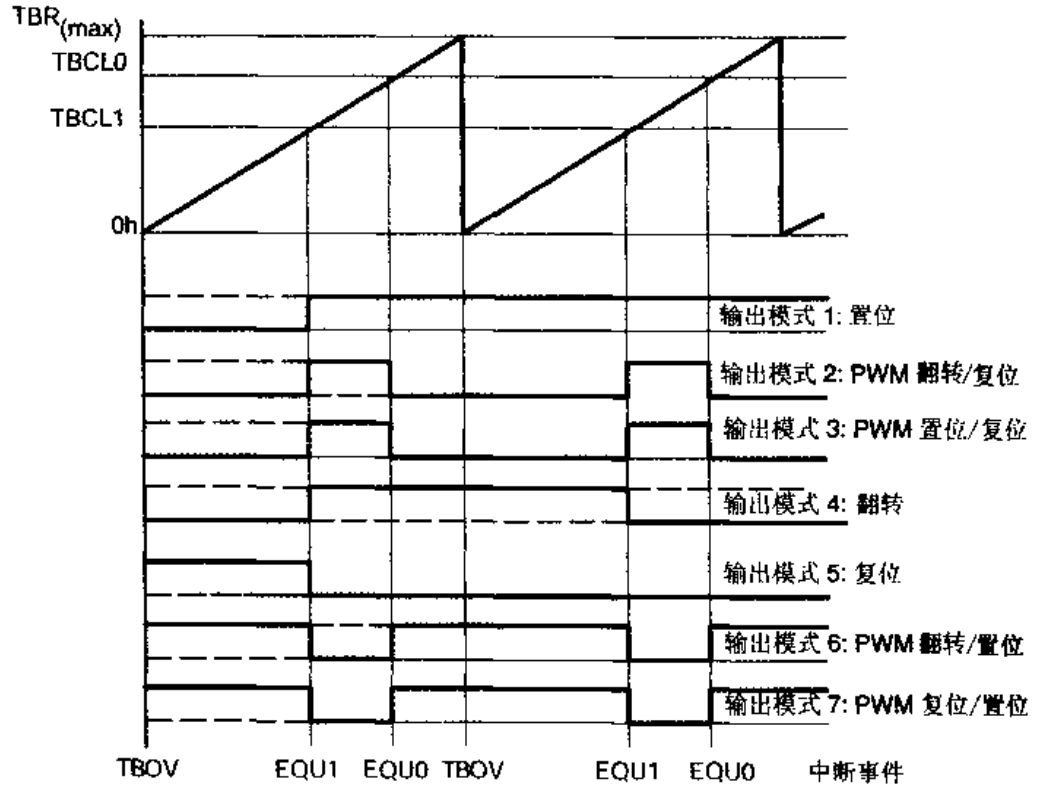


图 11.25 连续模式的输出实例

增/减计数模式实例

当定时器的值在任一计数方向上出现等于 TBCLx 和等于 TBCL0 时，OUTx 信号按选择的输出模式发生改变，见图 11.26。

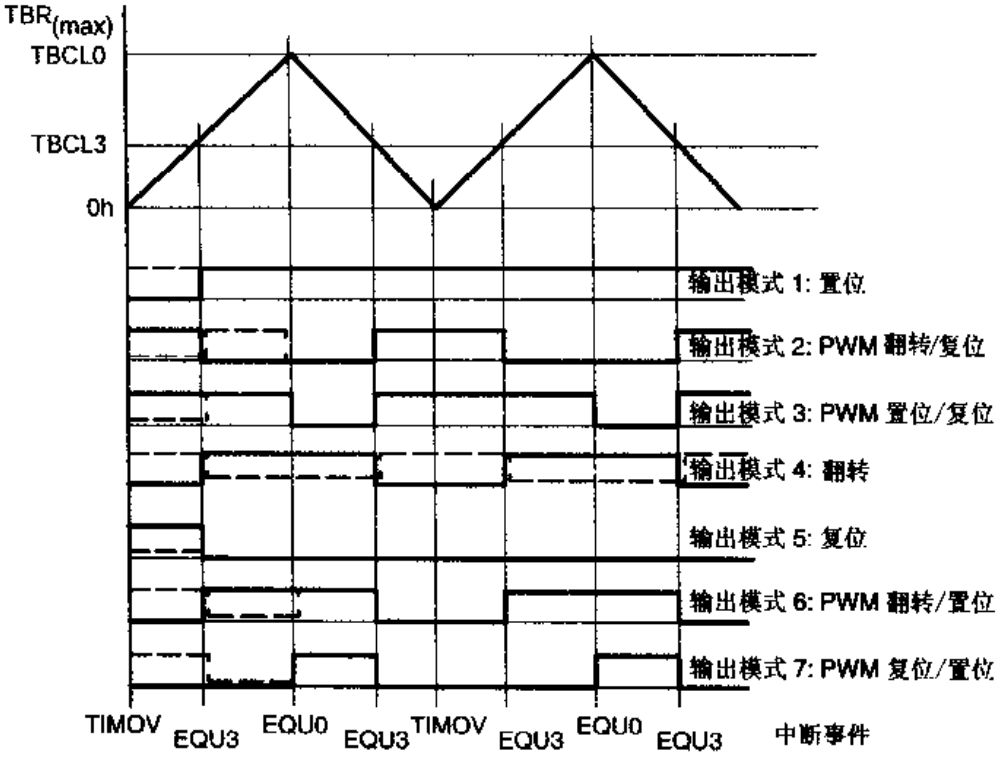


图 11.26　增/减计数模式实例

11.6　Timer_B 的寄存器

Timer_B 的寄存器是字结构的，必须用字指令来访问，见表 11.4。

表 11.4　Timer_B 的寄存器

寄存器	缩　写	寄存器类型	地　址	初始状态
Timer_B 控制寄存器	TBCTL	读/写	180h	POR 复位
Timer_B 寄存器	TBR	读/写	190h	POR 复位
捕获/比较控制寄存器 0	CCTL0	读/写	182h	POR 复位
捕获/比较寄存器 0	CCR0	读/写	192h	POR 复位
捕获/比较控制寄存器 1	CCTL1	读/写	184h	POR 复位
捕获/比较寄存器 1	CCR1	读/写	194h	POR 复位
捕获/比较控制寄存器 2	CCTL2	读/写	186h	POR 复位
捕获/比较寄存器 2	CCR2	读/写	196h	POR 复位
捕获/比较控制寄存器 3	CCTL3	读/写	188h	POR 复位
捕获/比较寄存器 3	CCR3	读/写	198h	POR 复位
捕获/比较控制寄存器 4	CCTL4	读/写	18Ah	POR 复位
捕获/比较寄存器 4	CCR4	读/写	19Ah	POR 复位
捕获/比较控制寄存器 5	CCTL5	读/写	18Ch	POR 复位
捕获/比较寄存器 5	CCR5	读/写	19Ch	POR 复位
捕获/比较控制寄存器 6	CCTL6	读/写	18Eh	POR 复位
捕获/比较寄存器 6	CCR6	读/写	19Eh	POR 复位
中断向量寄存器	TBIV	只读	11Eh	(POR 复位)

11.6.1 Timer_B 控制寄存器 TBCTL

全部对定时器及其操作的控制位都位于定时器控制寄存器 TBCTL 中。在 POR 信号后各位全部自动复位，但是在 PUC 后不受影响。控制寄存器必须用字指令访问。

TBCTL(180h)

15															0
未用	TBCLGRP		CNTL	未用	SSEL1 - 0		ID1 - 0		MC1 - 0		未用	CLR	TBIE	TBIFG	
rw -	rw -	rw -	rw -	rw -	rw -	rw -	rw -	rw -	rw -	rw -	rw -	rw -	w -	rw -	rw -
(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)

b0：TBIFG：指示定时器溢出事件。

增计数模式：　当定时器由 TBCL0 计数到“0”时 TBIFG 置位。

连续模式：　当定时器由最大计数长度计数到“0”时 TBIFG 置位。

增/减计数模式：　当定时器由“1”减计数到“0”时 TBIFG 置位。

b1：定时器溢出中断允许位 TBIE。置位使溢出产生中断请求，复位禁止中断。

b2：定时器清除位 CLR。定时器和分频器在 POR 后或 CLR 置位时复位。CLR 会自动复位，读出总是为“0”。如果不是因清除模式控制位使定时器暂停，则定时器在下一个有效时钟沿开始增计数。

b3：未用。

b4～5：模式控制位。

MC1	MC0	控制模式	说　明
0	0	停止	定时器暂停
0	1	增计数到 CCR0	计数至 TBCL0 并从“0”重新开始。 如果 TBCL0＞最大计数长度，则计数器在定时器时钟下一个上升沿复位为“0”
1	0	连续增计数	增计数至最大计数长度并从“0”重新开始。 最大计数长度：16 位：0FFFFh 12 位：00FFFh 10 位：003FFh 8 位：000FFh
1	1	增/减计数	增计数至 CCR0，减计数至“0”。 如果 CCR0＞最大计数长度，则计数器工作与连续模式相同，而不是减计数至“0”

b6～7：输入分频控制位。

ID1	ID0	分　频	说　明
0	0	/1	输入时钟直通定时器
0	1	/2	输入时钟 2 分频
1	0	/4	输入时钟 4 分频
1	1	/8	输入时钟 8 分频

TBR(190h)

15															0
定时器值															
rw-	rw-	rw-	rw-	rw-	rw-	rw-	rw-	rw-	rw-	rw-	rw-	rw-	rw-	rw-	rw-
(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)

注意:修改 Timer_B 寄存器 TBR

如果 ACLK、SMCLK 或外部时钟 TBCLK、INCLK 已选择为定时器时钟,则对 TBR 的写入应在定时器停止时进行,否则定时器的值是不可预测的。因为在这种情况下,CPU 时钟 MCLK 和定时器时钟是异步的,可能存在时间竞争。

11.6.3 捕获/比较控制寄存器 CCTLx

每个捕获/比较模块都有它自己的控制字 CCTLx。POR 信号将 CCTLx 复位,PUC 不影响 CCTLx。

CCTLx(182h~19Ah)

15															0
捕获模式		输入选择		SCS	CLLD		CAP	OUTMODx			CCIE	CCI	OUT	COV	CCIFG
rw-	rw-	rw-	rw-	rw-	rw-	rw-	rw-	rw-	rw-	rw-	rw-	r	rw-	rw-	rw-
(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)		(0)	(0)	(0)

b0:CCIFGx,捕获/比较中断标志。

捕获模式: 置位表示在寄存器 CCRx 中捕获了定时数值。

比较模式: 置位表示定时器值等于比较寄存器 CCRx 的值。

CCIFG0: 当中断请求被接受,CCIFG0 会自动复位。

CCIFG1~2: 读中断向量字 TAIV 后,确定中断向量的标志自动复位。如果不访问 TAIV 寄存器,则中断标志必须用软件复位。

如果相应的中断允许位复位,那么将不会产生中断请求,但标志仍置位,这时标志必须用软件复位。

如果相应的中断允许位置位,那么用软件将 CCIFG1~CCIFG4 置位会产生中断请求。

b1:COV,捕获溢出标志。

比较模式(CAP=0):捕获信号复位,捕获事件不会使 COV 置位。

捕获模式(CAP=1):如果捕获寄存器值被读出前再次发生捕获事件,则 COV 置位,COV 不会在读捕获值时复位,必须用软件复位。

b2:OUT,如果选输出模式 0,则由 OUTx 位确定 OUTx 信号。

b3:CCIx,捕获/比较输入。可将选定信号(CCIxA、CCIxB、V_{cc}或 GND)读出。

b4:CCIEx,中断允许位。允许(1)或禁止(0)捕获/比较模块 x 中断请求。GIE 位必须同时置位才能产生中断请求。

b5~7:输出模式选择。

b5～7	输出模式	说　明
0	输出	OUTx 位的数字确定 Outx
1	置位	比较信号 EQUx 使 Outx 置位
2	PWM 翻转/复位	比较信号 EQUx 使 Outx 翻转，EQU0 使 Outx 复位
3	PWM 置位/复位	比较信号 EQUx 使 Outx 置位，EQU0 使 Outx 复位
4	翻转	比较信号 EQUx 使 Outx 翻转
5	复位	比较信号 EQUx 使 Outx 复位
6	PWM 翻转/置位	比较信号 EQUx 使 Outx 翻转，EQU0 使 Outx 置位
7	PWM 复位/置位	比较信号 EQUx 使 Outx 复位，EQU0 使 Outx 置位

b8：CAP，选择捕获模式或比较模式。

0：比较模式，1：捕获模式。

b9～10：CLLD，选择比较锁存器 TBCLx 的加载方式。

CLLD＝0：立即。

CLLD＝1：当 TBR 计数到“0”时，CCRx 数据加载到 TBCLx。

CLLD＝2：增/减计数模式：TBR 计数到 TBCL0 或“0”，CCRx 数据加载到 TBCLx。

连续模式：当 TBR 计数到“0”时，CCRx 数据加载到 TBCLx。

CLLD＝3：当 TBR 计数到 TBCLx 时，CCRx 数据加载到 TBCLx。

b11：SCS，用于捕获输入信号与定时器时钟信号的同步。

0：异步捕获，1：同步捕获。

b12～13：CCIS0 和 CCIS1，输入选择。

在捕获模式中，定义捕获事件的输入信号源。在比较模式中无用。

0：选 CCIxA，1：选 CCIxB，2：GND，3：V_{CC}。

b14～15：选择捕获模式。

b14～15	捕获模式	说　明
0	禁止	禁止捕获模式
1	上升沿	上升沿捕获
2	下降沿	下降沿捕获
3	跳变沿	在上升沿与下降沿都捕获

注意：同时捕获和捕获模式选择

如果将工作模式从比较模式切换为捕获模式，则不能同时进行捕获。那样在捕获/比较寄存器中的值是不可预料的。推荐的程序顺序为

(1) 修改控制寄存器，由比较模式切换到捕获模式。

(2) 捕获。

```
例：BIS    #CAP,&CCTL2          ；用 CCR2 选择捕获模式
    XOR    #CCIS1,&CCTL2        ；软件捕获：CCIS0＝0，捕获模式＝3
```

11.6.4　Timer_B 中断向量寄存器

16 位 Timer_B 有 2 个中断向量：

- CCR0 中断向量，具有高优先级。
- TBIV 中断向量，用于 CCIFG1～CCIFGx 和 TBIFG 标志。

CCR0 中断向量

CCR0 中断向量与 CCR0 相关，如果定时器值等于比较寄存器的值时置位。具有最高的 Timer_B 的中断优先级，见图 11.27。

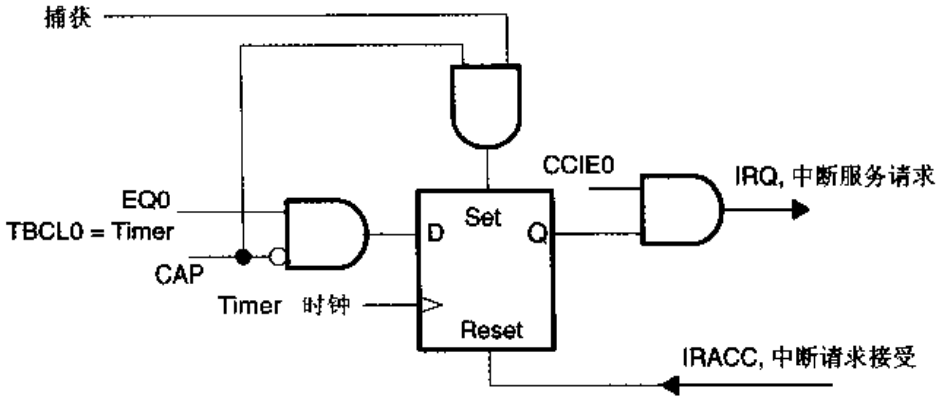


图 11.27 CCR0 中断标志

TBIFG，CCIFG1～CCIFGx 标志的中断向量

CCIFGx(不包括 CCIFG0)与 TBIFG 标志组成有优先级、共用一个中断向量的中断标志，见图 11.28。中断向量寄存器 TBIV 用于确定产生中断请求的中断源。

TBIV(11Eh)

15															0
0	0	0	0	0	0	0	0	0	0	0	中断向量				0
r0	r0	r0	r0	r0	r0	r0	r0	r0	r0	r0	r-(0)	r-(0)	r-(0)	r-(0)	r0

具有最高优先级的中断标志产生一个 2～14 的数存入 TBIV。优先级见表 11.5。

表 11.5 TBIV 中断优先级

中断优先级	中断源	缩 写	TBIV 内容
最高	捕获/比较 1	CCIFG1	2
	捕获/比较 2	CCIFG2	4
	捕获/比较 3	CCIFG3	6
	捕获/比较 4	CCIFG4	8
	捕获/比较 5	CCIFG5	10
	捕获/比较 6	CCIFG6	12
最低	定时器溢出	TBIFG	14
	没有中断挂起		0

访问 TBIV 能自动将最高优先级的中断请求标志复位。如果有另一个挂起的中断标志，则会在对当前中断请求服务后立即产生新的中断请求。例如，CCIFG2 与 CCIFG3 都已置位，当中断服务程序访问 TBIV(读 TAIV 或将 TAIV 加入 PC)时，CCIFG2 自动复位。中断服务程序中的 RETI 指令执行后，CCIFG3 自动产生新的中断请求。

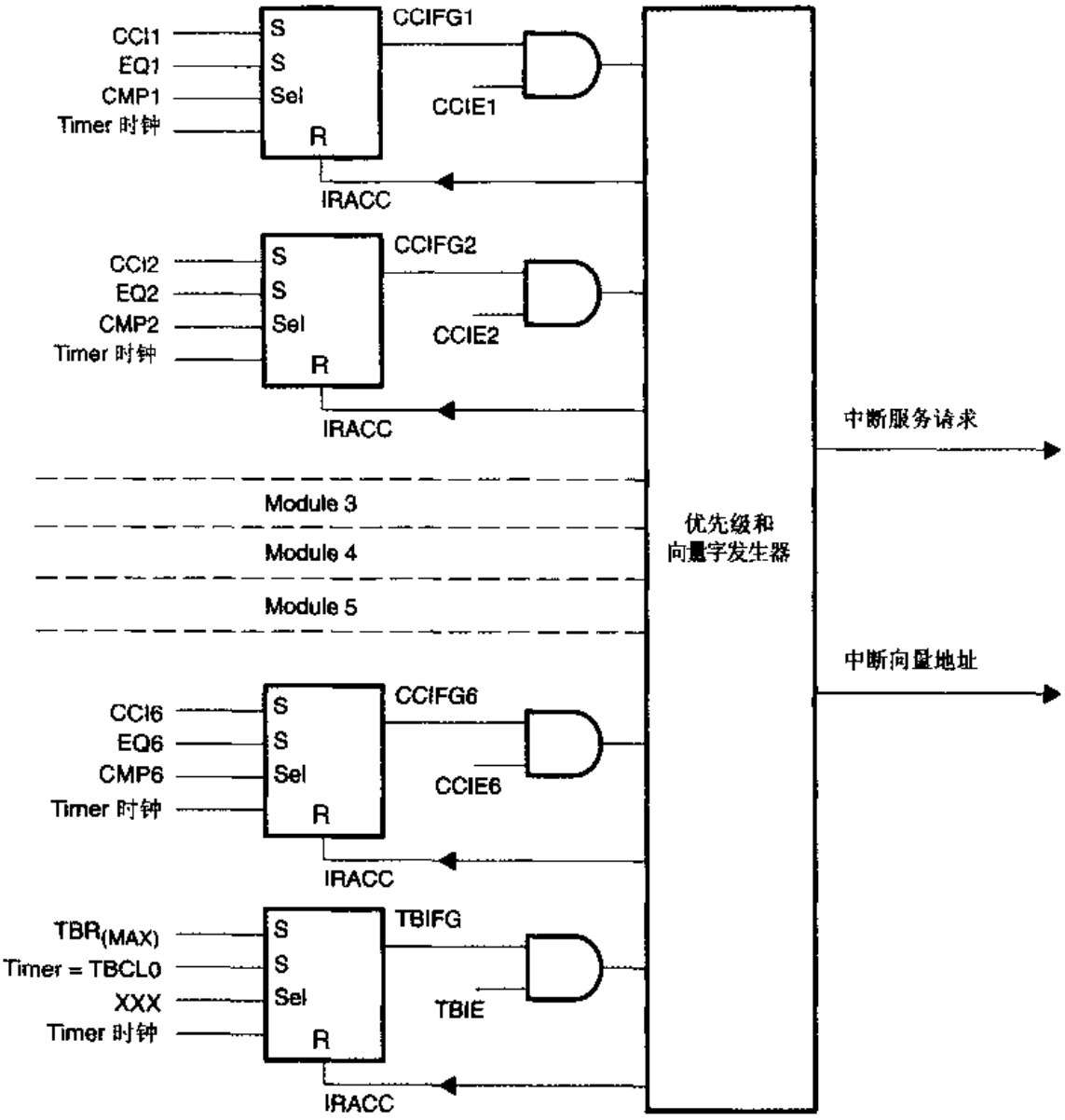


图 11.28　捕获/比较中断向量结构

注意:对只读寄存器 TBIV 写入

不应对 TBAIV 寄存器写入。对 TBIV 作写操作会使最高优先级的当前中断标志复位，其后的软件处理会漏过正在请求服务的中断事件。此外，对只读寄存器写操作将在写信号有效期间增加电流消耗。

例:定时器中断向量寄存器

以下软件实例说明了 Timer_B7 的中断向量 TBIV 的应用和处理开销。右边的数字显示每个指令需要的时钟周期。此程序是为连续模式写的，与下一中断的时间差已加入相应的比较寄存器。

```
; 例:中断程序                                                    周期
;
; 捕获/比较模块 0 的中断处理,中断标志 CCIFG0 能自动复位
```

```
;
TIMMOD0     …                              ; 开始中断处理                    6
            RETI                                                             5
;
; 捕获/比较模块 1~6 的处理,中断标志 CCIFGx 和 TAIFG 由硬件复位
; 只有在响应时具有最高优先级的标志复位
;
TIM_HND     $                              ; 潜在的中断响应过程              6
            ADD     &TAIV,PC               ; 加跳转表偏移量                  3
            RETI                           ; 向量 0:无中断                   5
            JMP     TIMMOD1                ; 向量 2:模块 1                   2
            JMP     TIMMOD2                ; 向量 4:模块 2                   2
            JMP     TIMMOD3                ; 向量 6:模块 3                   2
            JMP     TIMMOD4                ; 向量 8:模块 4                   2
            JMP     TIMMOD5                ; 向量 10:模块 5                  2
            JMP     TIMMOD6                ; 向量 12:模块 6                  2
;
; 定时器溢出处理:定时器寄存器扩展入 RAM 中的 TIMEXT(高位)
;
TIMOVH                                     ; 向量 10:定时器溢出处理
            INC     TIMEXT                 ;                                 4
            RETI                                                             5
;
TIMMOD2                                    ; 向量 4:模块 2
            ADD     #NN,&CCR2              ; 时间差累加                      5
            …                              ; 处理部分
            RETI                           ; 返回主程序                      5
;
TIMMOD1                                    ; 向量 2:模块 1
            ADD     #MM,&CCR1              ; 时间差累加                      5
            …                              ; 处理部分
            RETI                           ; 返回主程序                      5
;
; 模块 3 处理,检查是否有其他中断挂起
; 需用 5 个周期,但是如有中断挂起,可节省 9 个周期
;
TIMMOD3                                    ; 向量 6:模块 3
            ADD     #PP,&CCR3              ; 时间差累加                      5
            …                              ; 处理部分
            JMP     TIM_HND                ; 检查挂起中断                    2
;
            .SECT   "VECTORS",0FFF8h       ; 中断向量地址
            .WORD   TIM_HND                ; 捕获/比较模块 1~6 及 TBIFG 的向量
```

```
            .WORD  TIMMOD0          ; 捕获/比较模块 0 向量
```

以下软件实例说明了 Timer_B3 的中断向量 TBIV 的应用和处理开销。右边的数字显示每个指令需要的时钟周期。此程序是为连续模式写的，与下一中断的时间差已加入相应的比较寄存器。

```
;例:中断程序                                                    周期
;                                                               *
; 捕获/比较模块 0 的中断处理,中断标志 CCIFG0 能自动复位
;
TIMMOD0     …                       ; 开始中断处理              6
            RETI                                                5
;
; 捕获/比较模块 1~6 的处理,中断标志 CCIFGx 和 TAIFG 由硬件复位
; 只有在响应时具有最高优先级的标志复位
;
TIM_HND     $                       ; 潜在的中断响应过程        6
            ADD     &TAIV,PC        ; 加跳转表偏移量            3
            RETI                    ; 向量 0:无中断             5
            JMP     TIMMOD1         ; 向量 2:模块 1             2
            JMP     TIMMOD2         ; 向量 4:模块 2             2
            RETI                    ; 向量 6:
            RETI                    ; 向量 8:
            RETI                    ; 向量 10:
            RETI                    ; 向量 12:
;
; 定时器溢出处理:定时器寄存器扩展入 RAM 中的 TIMEXT(高位)
;
TIMOVH                              ; 向量 14:定时器溢出处理
            INC     TIMEXT          ;                           4
            RETI                                                5
;
TIMMOD2                             ; 向量 4:模块 2
            ADD     #NN,&CCR2       ; 时间差累加                5
            …                       ; 处理部分
            RETI                    ; 返回主程序                5
;
; 模块 1 处理,检查是否有其他中断挂起
; 需用 5 个周期,但是如有中断挂起,可节省 9 个周期
;
TIMMOD1                             ; 向量 6:模块 3
            ADD     #PP,&CCR3       ; 时间差累加                5
            …                       ; 处理部分
            JMP     TIM_HND         ; 检查挂起中断              2
```

```
.SECT   "VECTORS".0FFF8h ; 中断向量地址
.WORD   TIM_HND          ; 捕获/比较模块 1～6 及 TBIFG 的向量
.WORD   TIMMOD0          ; 捕获/比较模块 0 向量
```

如果 CPU 时钟 MCLK 关闭(CPUOff=1),则还需要再增加 2～3 个额外时钟周期用于使 CPU 的启动同步。引起有 1 个时钟周期误差是因为 CPU 时钟 MCLK 的再启动与其他时钟信号是异步的。

对各个中断源,中断程序的基本时间开销包括潜在的中断响应和从中断返回的周期数(不包括处理任务本身)。

- 捕获/比较模块 CCR0　　11 个周期。
- 捕获/比较模块 CCR1～CCR6　16 个周期。
- 定时器溢出 TBIFG　　14 个周期。

定时极限

用 TBIV 寄存器和上述软件,利用一个比较寄存器,两个事件之间的最短重复时间是

$$t_{CRmin} = t_{taskmax} + 16 \times T_{cycle}$$

其中:$t_{taskmax}$　最长时间(最差情况),在中断程序完成任务,如计数器加 1。

T_{cycle}　MCLK 时钟周期。

利用一个捕获寄存器时两个事件之间的最短重复时间是

$$t_{CLmin} = t_{taskmax} + 16 \times T_{cycle}$$

第 12 章　USART 通信模块的 UART 功能

通用串行同步/异步(USART)是一个串行通信接口，它允许 7 或 8 位串行位流以预设的速率或外部时钟确定的速率移入、移出 MSP430。USART 接口支持两种不同的串行协议：通用异步协议(UART 协议)和同步协议(SPI 协议)。

用控制寄存器 UCTL 中的控制位 SYNC 来选择所需的模式。

SYNC=0：　选择异步模式 UART；

SYNC=1：　选择同步模式 SPI。

USART 以字节外围模块与 CPU 相连，通过 3 或 4 个引脚与外部系统相连。

在 MSP430F14x 系列中，片内有 2 个 USART 模块，即 USART0 和 USART1。

在 MSP430F13x 系列中，片内只有 1 个 USART 模块 USART0。

在 MSP430F11x 和 MSP430F11x1 系列中，片内无 USART 模块，串行通信功能需要用定时器 Timer_A 或者完全用软件来实现。

图 12.1 是完整的 USART 功能框图。

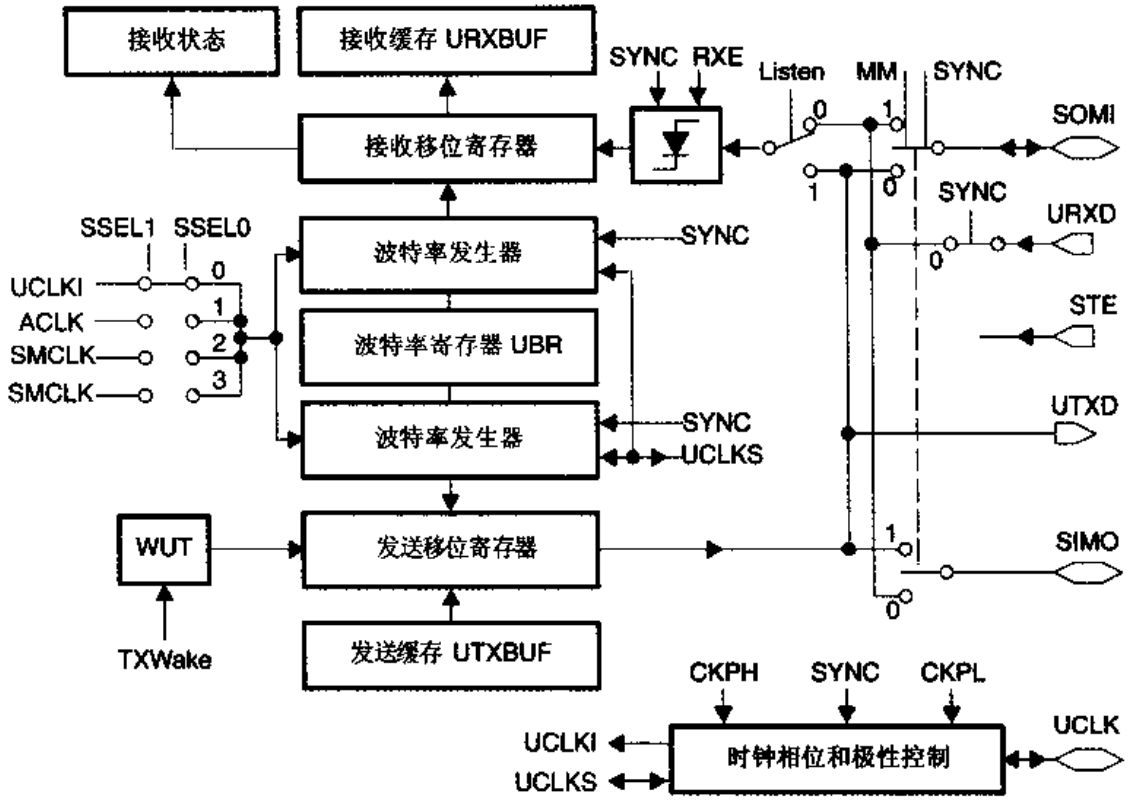


图 12.1　USART 功能框图

USART 控制寄存器 UCTL 的控制位 SYNC 复位时选择异步通信模式。

USART 的串行异步通信特性

- 异步模式，包括线路空闲多机通信协议及地址位多机通信协议。

频因子 N 不能为 18，同样非整数因子也无法使用，如 13.67。

例 1

设 BRCLK 信号频率为 32 768Hz，波特率要求为 4 800 波特，则分频因子为 6.83。但在标准波特率发生器里，最小的因子为 16，因此晶振频率和波特率发生器不能满足要求。

例 2

设 BRCLK 信号的频率为 1.04 MHz(32×32 768 Hz)，波特率要求为 19 200 波特，则分频因子为 54.61；但在标准的波特率发生器里，较接近的分频因子为 48(3×16)和 64(4×16)，因此晶振频率和波特率发生不能满足要求。晶振频率需要特别挑选以满足通信要求。而其他的原则，诸如降低电流消耗、方便的实时时钟功能或降低成本等就无法考虑了。

MSP430 的波特率发生

MSP430 波特率发生器见图 12.6。有一个预分频/分频器和一个调整器的组合，即使晶振频率不是所需波特率的整数倍，这一组合不仅能正常工作而且使通信协议可以工作在最大的波特率。采用这一技术，即使用钟表晶振(32 768 Hz)，波特率也可以达到 4 800 和 9 600 波特。它带来的优点很明显，使得 MSP430 的工作可以选择在低功耗模式。

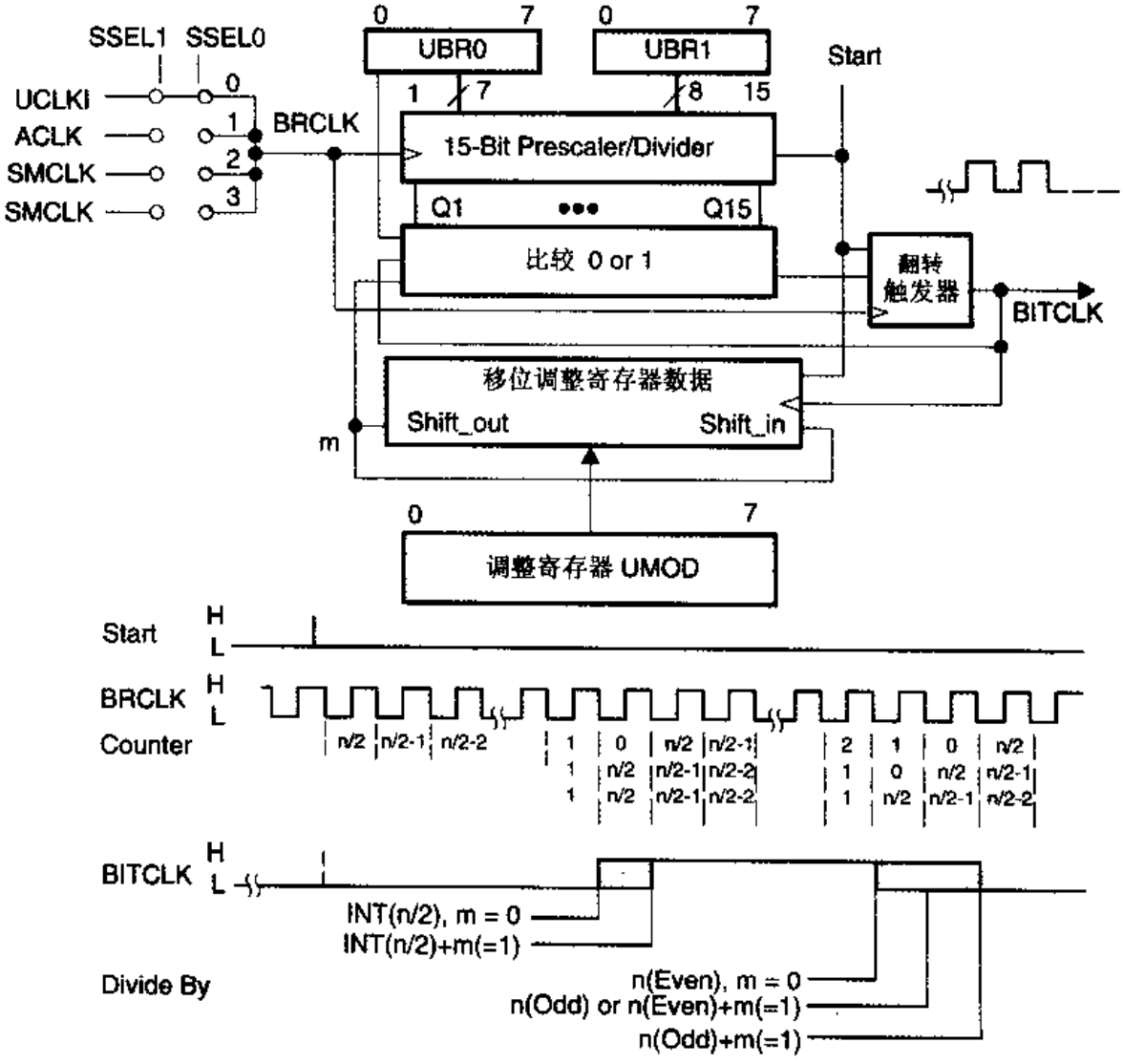


图 12.6 MSP430 波特率发生：n 或 n+1 周期举例

调整寄存器在进行调整时先利用最低位(LSB),它从起始位开始工作。置位的调整位使分频因子增加 1。

例 1：　4 800 波特

假定 BRCLK 的频率为 32 768 Hz,波特率要求为 4 800 波特,则分频因子为 6.83。MSP430 的 USART 采用分频因子 6 加上调整寄存器加载 6Fh(01101111)来产生波特率。即分频器按顺序 7,7,7,7,6,7,7,6,…来分频。在 8 位调整位都使用过后,再重复这一顺序。

例 2：　19 200 波特

假定 BRCLK 信号的频率为 1.04 MHz,波特率要取为 19 200 波特,则分频因子为 54.61。MSP430 的 USART 采用分频因子 54(36h)加上调整寄存器加载 D5h(11010101h)来产生波特率,即分频器按顺序 55,54,55,54,55,54,55,55,…来分频。在 8 位调整位都使用后,再重复这一顺序。

12.1.3　异步通信格式

当选择异步模式时,USART 模块支持 2 种多机通信模式。在同一个串行链路上,多机之间可以用异步通信格式来交换信息。以一个多帧数据块,从一个指定的源送达一个或多个目的位置。USART 可识别数据块的起始,并能抑制接收端处理中断和状态信息,直至数据块起始被识别。在这两种多机模式下,USART 数据交换过程可以用数据查询方法也可用接收中断方式来实现。

两种多机模式,即线路空闲多机模式和地址位多机模式,实现了在多机通信系统间的有效数据传输。它们也大大压缩了系统的激活状态,节省了电流消耗或所需资源。

由控制寄存器的 MM 位确定是地址位多机模式还是线路空闲多机模式。这两种格式都用激活 TXWake 位(地址特性)和 RXWake 位来实现唤醒发送功能。URXWIE 和 URXIE 位控制这些模式的发送和接收。

12.1.4　线路空闲多机模式

在这种模式下,数据块被一段空闲时间分隔。在字符的第一个停止位之后接收到 10 个以上的"1",则表示检测到接收线路空闲,见图 12.7。

如果采用 2 位停止位,则第 2 个停止位看作空闲周期的第 1 个传号。空闲周期后的第一个字符是地址字符。RXWake 位可作为地址字符的标记。在线路空闲多机格式中,当接收字符是地址字符时 RXWake 置位,并送入接收缓存中,见图 12.8。

正常情况下,如果 USART 接收控制寄存器的 URXWIE 置位,则字符在接收器以通常方法合成,但是并不将该字符送入接收缓存 URXBUF,也不会产生中断。只有当接收到地址字符时,接收器将暂时激活,字符送入 URXBUF,同时将中断标志 URXIFG 置位。相应的错误状态标志位也会置位。应用软件可以验证接收到的地址。如果地址匹配,应用软件将处理后续字符并执行适当的操作。如果地址不匹配,那么处理机继续等待下一个地址字符的到来。URXWIE 位不会自动修改,必须由用户根据接收地址字符和非地址字符的需要去修改。

在线路空闲多机模式下,为了产生有效的地址字符识别,可以用精确的空闲周期。与 TXWake 位相关的是临时唤醒标志 WUT,这是一个内部标志,与 TXWake 一起构成双缓存。当发送器从 UTXBUF 装入数据,WUT 也从 TXWake 装入,同时 TXWake 复位,见图 12.9。

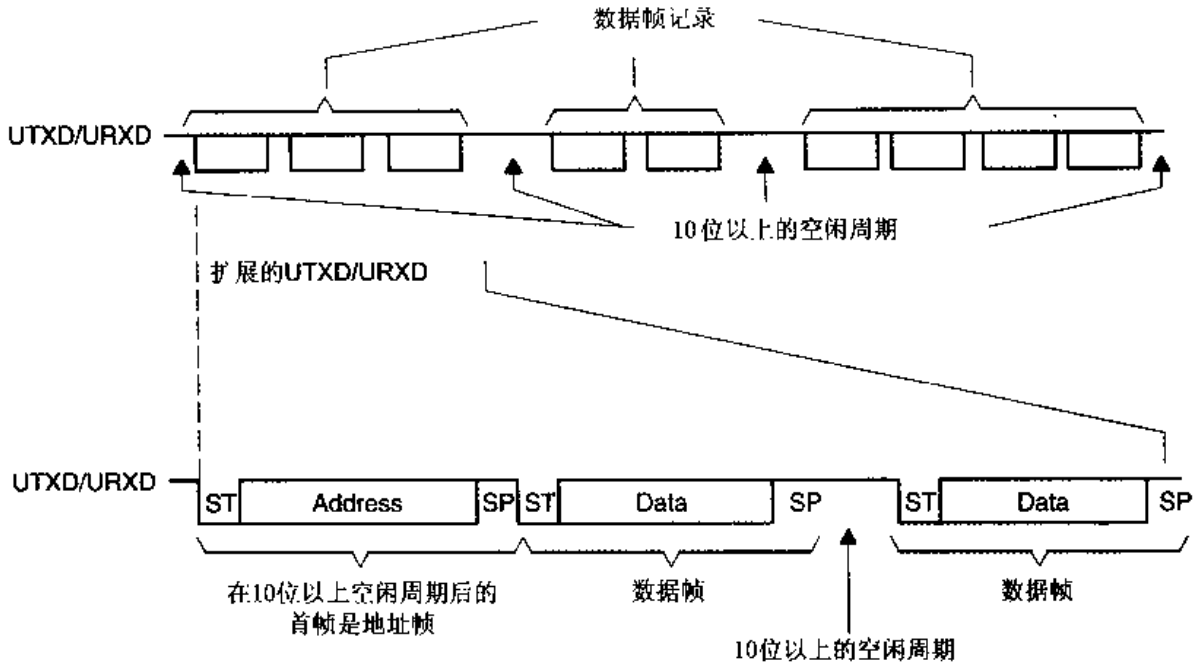


图 12.7 线路空闲多机协议

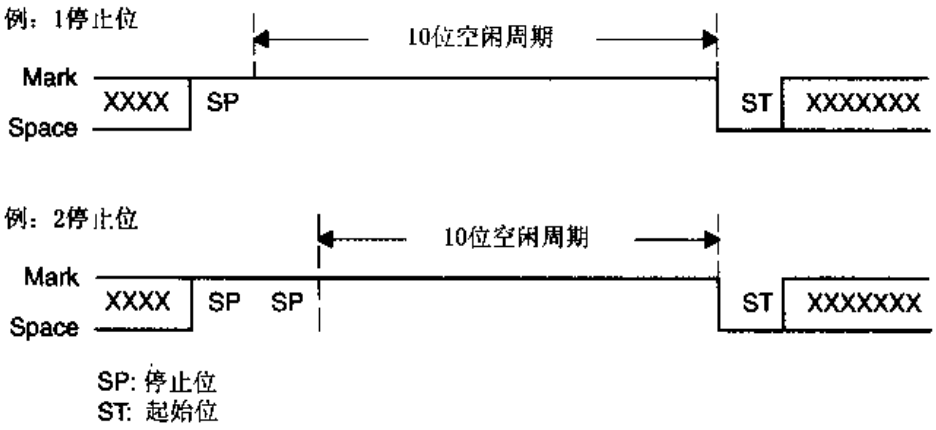


图 12.8 USART 接收空闲检测

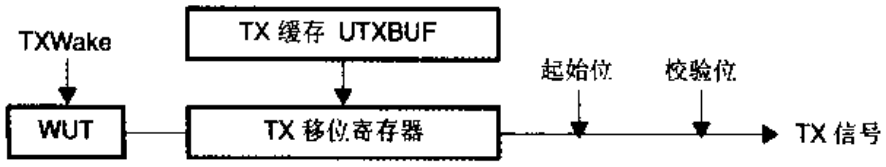


图 12.9 双缓存的 WUT 和 TX 移位寄存器

用发送空闲帧来标识地址字符，按下面步骤完成。

首先 TXWake 位应置位，将任意数据(内容无关)写入 UTXBUF(UTXIFG 应置位)。当发送移位寄存器空时(TXEPT 置位)，UTXBUF 的内容被送入发送移位寄存器，同时 TXWake 的值移入 WUT。

如果此时 WUT 置位，则要发送的起始位、数据位和校验位被抑制，发送一个正好 11 位的

空闲周期，见图 12.10。在地址字符识别空闲周期之后移出串行端口的下一个数据是 TXWake 置位后写入 UTXBUF 中的第二个字符。当地址识别被发送后，写入 UTXBUF 中的第一个字符被抑制并在以后被忽略。将一个任意内容的字符写入 UTXBUF 是必要的，只有这样才能使 TXWake 的值移入 WUT 中。

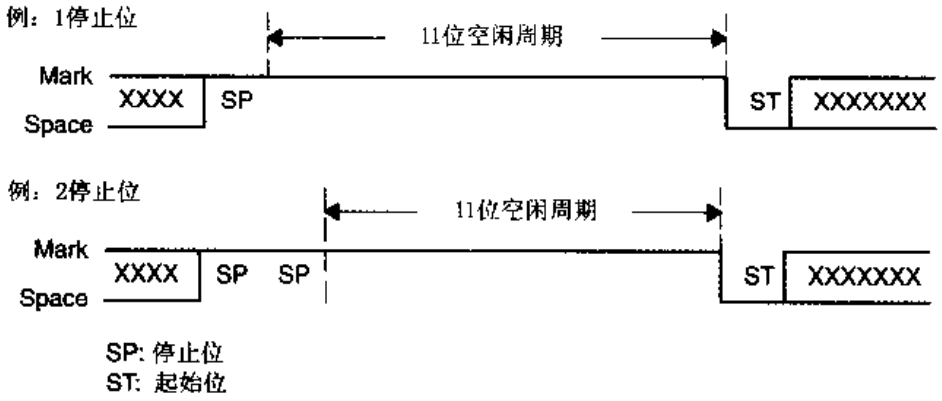


图 12.10　USART 发送空闲周期的产生

12.1.5　地址位多机通信格式

图 12.11 是地址位多机通信模式的格式。在这种模式下，字符包含一个附加位作为地址标识。数据块的第一个字符带有一个置位的地址位，以表明该字符是一地址。当接收字符是地址时 RXWake 置位，并且被送入接收缓存 URXBUF(允许接收时)。

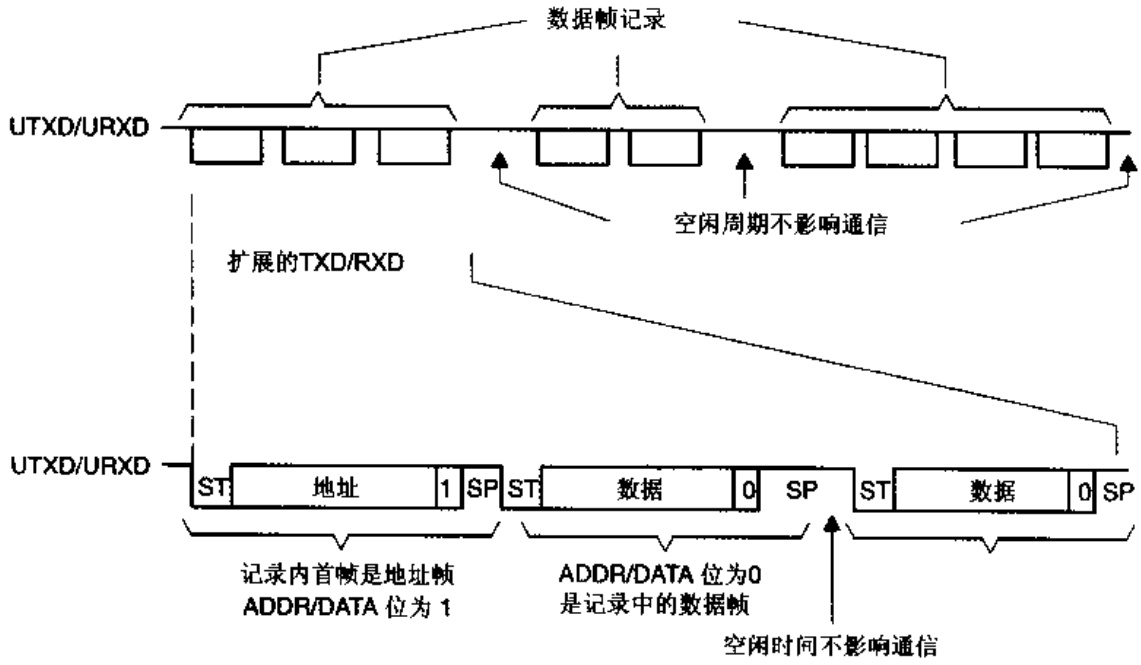


图 12.11　地址位多机协议

在 USART 的 URXWIE 置位情况下，数据字符按通常方式在接收器拼装，但是它们不会送入接收缓存 URXBUF，也不会产生中断。只有当接收到一个地址位置位的字符时，接收器才被激活，字符送入 URXBUF，同时 URXIFG 置位。如果有错误，则相应的错误状态标志将

置位。应用软件可以按有效利用资源、降低功耗的原则进行后续操作。可以检验接收到的地址，如果匹配，则处理机将读取数据块的后续数据；如果地址不匹配，则处理机将等待下一地址字符的到来。

在地址位多机模式下，通过写 TXWake 位控制字符的地址位。每当字符从 UTXBUF 传送至发送器时，TXWake 位装入字符的地址位，然后 TXWake 位将被 USART 清除。

12.2 中断和中断允许

USART 外围模块有发送和接收 2 个中断源。2 个独立的中断向量，一个用于接收中断事件，另一个用于发送中断事件。

USART 中的控制位位于 SFR 地址中。

- 接收中断标志 URXIFG 初始状态复位(用 PUC/SWRST)
- 接收中断允许 URXIE 初始状态复位(用 PUC/SWRST)
- 接收允许 URXE 初始状态复位(用 PUC)
- 发送中断标志 UTXIFG 初始状态置位(用 PUC/SWRST)
- 发送中断允许 UTXIE 初始状态复位(用 PUC/SWRST)
- 发送允许 UTXE 初始状态复位(用 PUC)

接收器和发送器是完全独立操作的，但是使用同一个波特率发生器。发送和接收使用相同的波特率。

12.2.1 USART 接收允许

接收允许位 URXE 的置位或复位，能允许或禁止接收器从 URXD 数据线路接收位流，见图 12.12。禁止 USART 接收器时，如果已开始一次接收操作，则会在完成后停止接收操作；如果无接收操作，则在进行中将立即停止接收操作，起始位检测也同时禁止。

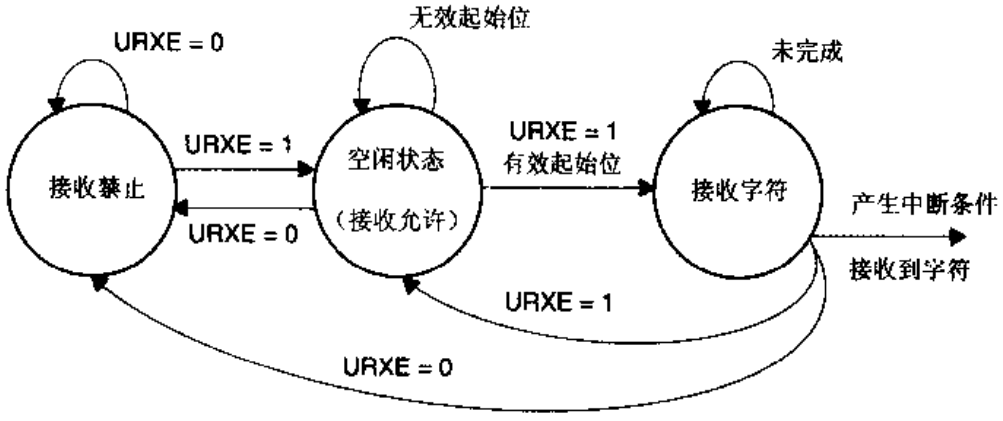


图 12.12 接收允许 URXE 状态图

注意：UART 模式的 URXE 重允许

在接收器完全禁止时，接收器的重允许与通信线路的数据流是异步的。在接收器接收数据前，可以用查找线路空闲状态来实现同步。

12.2.2　USART 发送允许

发送允许位 UTXE 的置位或复位，能允许或禁止串行数据线路上的字符发送，见图 12.13。UTXE 复位时，已激活的发送并不停止，会在完成已写入发送缓存内的全部数据的发送后才被禁止。在 UTXE 复位前写入发送缓存中的数据有效。例如，软件对发送缓存写入一个字节，然后将 UTXE 位复位，这一字节会传送到发送移位寄存器并发送。在这一字节传送完成后，对 UTXBUF 的写操作不会产生发送，但是 UTXBUF 仍然会被新数据更新。

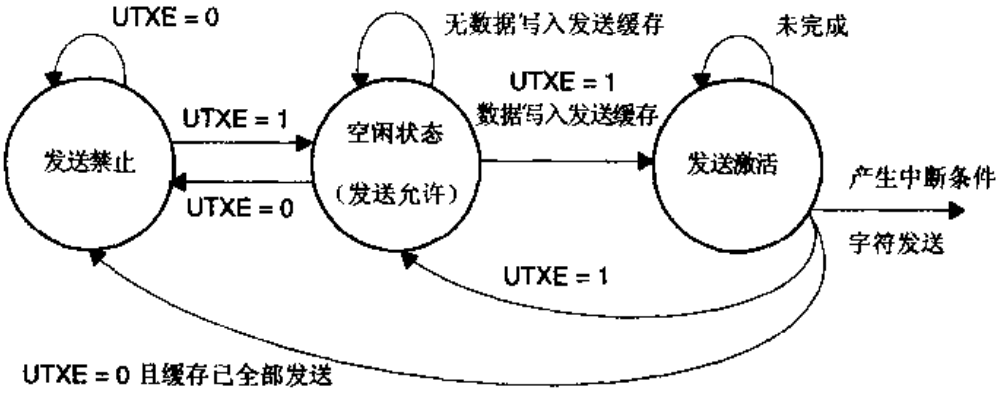


图 12.13　发送允许状态图

当 UTXE 复位时，发送缓存能照常写入，但不会启动发送。一旦 UTXE 置位，缓存内字符的发送立即开始。已写入的字符能正确发送。

注意：写 UTXBUF，UART 模式

在 UTXBUF 未就绪且发送允许（UTXE 置位）时，不要对它写入数据，否则发出的字符可能是随机的。

注意：UART 模式的 UTXBUF 写入及发送复位

禁止发送器应该在发送数据全部移入发送移位寄存器后才进行。如：

```
MOV.B    #…,&UTXBUF
BIC.B    #UTXE,&ME2      ; 如果 BITCLK<MCLK，则在数据移入发送移位寄存器之
                         ; 前，发送器可能已停止工作
```

12.2.3　USART 接收中断操作

每次接收字符并装入接收缓存时，接收中断标志 URXIFG 将置位，见图 12.14。

- 当 URXEIE 复位时，错误字符（校验错、帧错或打断）不会使中断标志 URXIFG 置位，即 URXIFG 不会改变。
- 由 URXWIE 位决定是所有字符（URXWIE=0）或仅地址字符（URXWIE=1）使中断标志 URXIFG 置位。如果 URXEIE 也置位，则错误字符也将使中断标志 URXIFG 置位。

系统复位 PUC 或软件复位 SWRST 后，URXIFG 复位。如果执行中断服务（URXSE=0）或读接收缓存 URXBUF，则 URXIFG 会自动复位。接收中断标志 URXIFG 置位表示有等

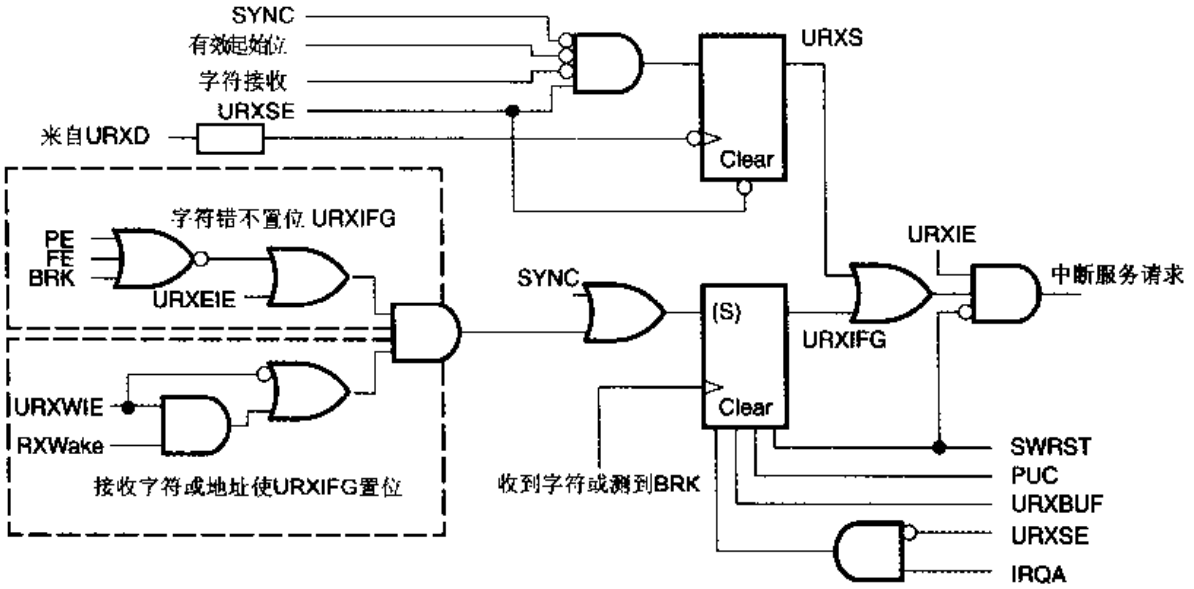


图 12.14 接收中断条件

待服务的中断事件。如果这时接收中断允许位 URXIE 置位，那么就会对等待处理的中断请求服务。接收中断标志位 URXIFG 和接收中断允许位 URXIE 在发生 PUC 和 SWRST 时复位。

URXIFG 可用软件访问，而 URXS 不能。当这两种中断事件，即起始位检测和字符接收有效都被软件允许时，URXIFG 请求的中断服务是字符接收而不是检测到起始位。因为一旦中断处理器接收起始位检测使 URXSE 复位，也就清除了 URXS 位，这样防止 URXS 再次引起中断请求。这样，URXIFG 因为无置位条件发生而复位。

12.2.4 USART 发送中断操作

发送器使发送中断标志 UTXIFG 置位，表示发送缓存 UTXBUF 已准备好处理下一个字符，见图 12.15。如果中断服务软件已启动或对 UTXBUF 写入一个字符，则 URXIFG 被自动复位。这一标志在 UTXIE 和通用中断允许 GIE 置位时会引起发送中断。UTXIFG 在发生 PUC 或 SWRST 信号消除后置位。

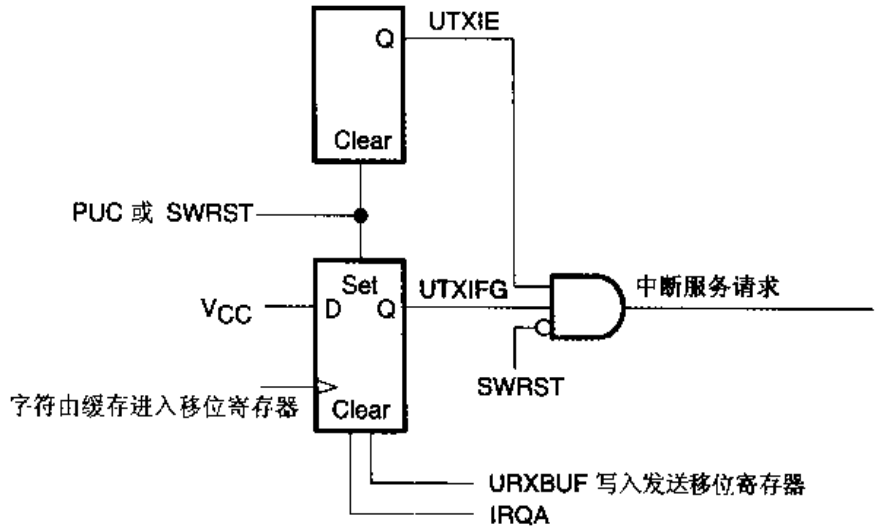


图 12.15 发送中断条件

UCTL0(070h)、UCTL1(078h)

7							0
PENA	PEV	SP	CHAR	Listen	SYNC	MM	SWRST
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1

b0：如果 SWRST 置位，则 USART 状态机和运行标志初始化成复位状态(URXIFG=0，URXIE=0，UTXIE=0，UTXIFG=1)。所有受影响的逻辑保持在复位状态，直至 SWRST 复位。这意味着一次系统复位后，只有对 SWRST 复位，USART 才能重新被允许。接收和发送允许标志 URXE 和 UTXE 不会因 SWRST 而更改。

SWRST 位会使 URXIE、UTXIE、URXIFG、RXWAKE、TXWAKE、RXERR、BRK、PE、OE 及 FE 等复位。

SWRST 位会使 UTXIFG、TXEPT 置位。

b1：多机模式(地址位或线路空闲)。

USART 模块支持两种多机协议：线路空闲和地址位。多机模式的选择影响自动地址解码功能。

MM=0：　线路空闲多机协议；

MM=1：　地址位多机协议。

执行常规的异步协议时需将 MM 复位。

b2：USART 模块的模式和功能选择。

SYNC 位选择 USART 外围接口模块的功能。USART 的一些控制位在 UART 模式和 SPI 模式中有不同的功能。

SYNC=0：　UART 模式；

SYNC=1：　SPI 模式。

b3：Listen 位选择是否将发送数据由内部反馈给接收器。

Listen=0：　无反馈；

Listen=1：　发送信号由内部反馈给接收器。每个 MSP430 的 USART 发送的数据同时被 USART 的接收器接收，通常称为自环模式。

b4：字符长度。

选择字符以 7 或 8 位发送。7 位时不用 URXBUF 和 UTXBUF 的最高位，填“0”。

CHAR=0：　7 位；

CHAR=1：　8 位。

b5：停止位数。

决定发送时停止位数，但是接收器只检测 1 位停止位。

SP=0：　1 位停止位；

SP=1：　2 位停止位。

b6：校验奇偶位。

如果 PENA 置位(校验允许)，则 PEV 位按发送或接收字符、地址位(地址位多机模式)和校验位中的“1”的数量定义奇校验或偶校验。

PEV=0：　奇校验；

PEV=1：　偶校验。

b7：校验允许位。

如果禁止校验，则发送时不会产生校验位，接收时也不期望收到这一位。因为校验位不是数据位之一，接收到的校验位不传送入 URXBUF 中。在地址位多机模式中，地址位包括在校验计算中。

PEN＝0：　校验禁止；

PEN＝1：　校验允许。

注意：传号、空号定义

传号电平与空闲状态信号电平相同。空号电平与传号电平相反，起始位总是空号。

12.3.2　发送控制寄存器 UTCTL

寄存器 UTCTL 控制与发送操作相关的 USART 硬件。

UTCTL0(071h)、UTCTL1(079h)

7							0
未用	CKPL	SSEL1	SSEL0	URXSE	TXWake	未用	TXEPT
rw－0	rw－0	rw－0	rw－0	rw－0	rw－0	rw－0	rw－1

b0：发送器空标志 TXEPT 在发送移位寄存器和 UTXBUF 空时置位，当数据写入 UTXBUF时复位，它也在 SWRST 置位时置位。

b1：未用。

b2：TXWake 位用于控制多处理机通信模式的发送特性。通过装入 UTXBUF 启动一次发送，用 TXWake 位状态初始化地址识别特性。它不必清除，一旦被传送至 WUT（“暂时唤醒”），USART 硬件自动将它清除；SWRST 位也能将它清除。

b3：接收触发沿控制位置位请求接收中断服务。这时相应的允许位和 GIE 应置位。用这一位的优点是能在中断服务程序中启动微控制器时钟系统，包括 MCLK，启动中断服务及通过修改模式控制位来保持运行。

b4～5：时钟源选择 0 和 1。

时钟源选择位确定用于波特率发生器的时钟源。

SSEL1、0	0	选择外部时钟 UCLKI；
	1	选择辅助时钟 ACLK；
	2、3	选择系统主时钟 SMCLK。

b6：时钟极性 CKPL。CKPL 位控制 UCLKI 信号的极性。

CKPL＝0：　UCLKI 信号与 UCLK 极性相同；

CKPL＝1：　UCLKI 信号与 UCLK 极性相反。

b7：未用。

12.3.3　接收控制寄存器 URCTL

URCTL 控制与接收操作相关的 USART 硬件并保存由最新写入 URXBUF 的字符引起的出错状况和唤醒条件。一旦 FE、PE、OE、BRK、RXERR 或 RXWake 的任何一位置位，不能通过接收下一个字符来复位。它们的复位要通过访问接收缓存 URXBUF、USART 的软件复

注意：接收状态控制位

接收状态控制位 FE、PE、OE、BRK 和 RXWake 由硬件根据接收字符条件置位。置位后会一直保持，直到用软件复位或读取接收缓存。未清除的出错位会引起字符解释错或失去中断。

12.3.4　波特率选择和调整控制寄存器

波特率发生器用波特率选择寄存器 UBR0、1 和调整控制寄存器 UMCTL 来产生串行数据位定时。

UBR00(074h)、UBR01(07Ch)

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

UBR10(075h)、UBR11(07Dh)

7							0
2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8
rw	rw	rw	rw	rw	rw	rw	rw

$$\text{波特率} = \text{BRCLK} /(\text{UBR}+(m7+m6+\cdots+m0)/8)$$

式中：UBR=[UBR1，UBR0]，为 UBR1、UBR0 中的 16 位数。

波特率控制范围：3≤ UBR< 0FFFFh。

注意：UBR

当 BUR<3 时，发送和接收会发生不可预测的情况。

即使晶振频率不是所需波特率的整数倍，调整控制寄存器与 UBR0、1 也能确保产生合适的时序时间。

UMCTL0(073h)、UMCTL0(07Bh)

7							0
M7	M6	M5	M4	M3	M2	M1	M0
rw	rw	rw	rw	rw	rw	rw	rw

如果置位，则对应位的时序时间按波特率分频器的输入时钟扩展一个时钟周期。

每接收或发送一位，在调整控制寄存器的下一位被用来决定当前位的定时时间。协议的第一位(即起始位)的定时由 UBR 加上 M0 决定，下一位由 UBR 加上 M1 决定，以此类推。

调整顺序为

$$M0-M1-M2-M3-M4-M5-M6-M7-M0-M1-M2-\cdots$$

12.3.5　USART 接收数据缓存 URXBUF

接收缓存 URXBUF 含有此前从接收移位寄存器来的数据。读取 URXBUF 中数据，将使复位接收出错位、RXWake 位和中断标志 URXIFG 位复位。

URXBUF0(076h)、URXBUF1(07Eh)

7							0
b7	b6	b5	b4	b3	b2	b1	b0
r	r	r	r	r	r	r	r

在 7 位字长模式,URXBUF 的最高位总是"0"。

当接收和控制条件为真,接收缓存装入当前接收的字符。

URXEIE	URXWIE	装入 URXBUF	PE	PE	BRK
0	1	无差错地址字符	0	0	0
1	1	所有的地址字符	x	x	x
0	0	无差错字符	0	0	0
1	0	所有字符	x	x	x

12.3.6 USART 发送数据缓存 UTXBUF

发送数据缓存含有当前要由发送器发送的数据。

UTXBUF0(077h)、UTXBUF1(07Fh)

7							0
b7	b6	b5	b4	b3	b2	b1	b0
rw	rw	rw	rw	rw	rw	rw	rw

UTXIFG 标志表示 UTXBUF 已准备好接收下一个要发送的字符。

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即将为空,那么数据的发送立即开始。

注意:对 UTXBUF 写入

只有当 UTXBUF 为空时,数据才能写入缓存,否则可能发送不可预料的字符。

12.4 UART 模式,低功耗模式应用特性

MSP430 有许多的功能和操作特性支持实现基于 MSP430 结构的超低功耗系统。

- 用对 UART 帧的检测为启动条件,系统可从任意的工作模式开始运行。
- 用最低的输入时钟频率来实现所需波特率。
- 支持多机模式来减少 MSP430 资源的使用。

12.4.1 由 UART 帧启动接收操作

当波特率用 SMCLK 发生时,能最有效地利用接收通道起始检测。在这样的配置下,MSP430 可以在无 SMCLK 的情况下进入低功耗模式。接收起始条件是 URXD 信号的下降沿。每次下降沿触发 URXS 中断标志,如 URXIE 和 GIE 允许会请求中断服务。这样就唤醒了 MSP430,使系统返回活动模式,并支持 USART 传输,见图 12.16。

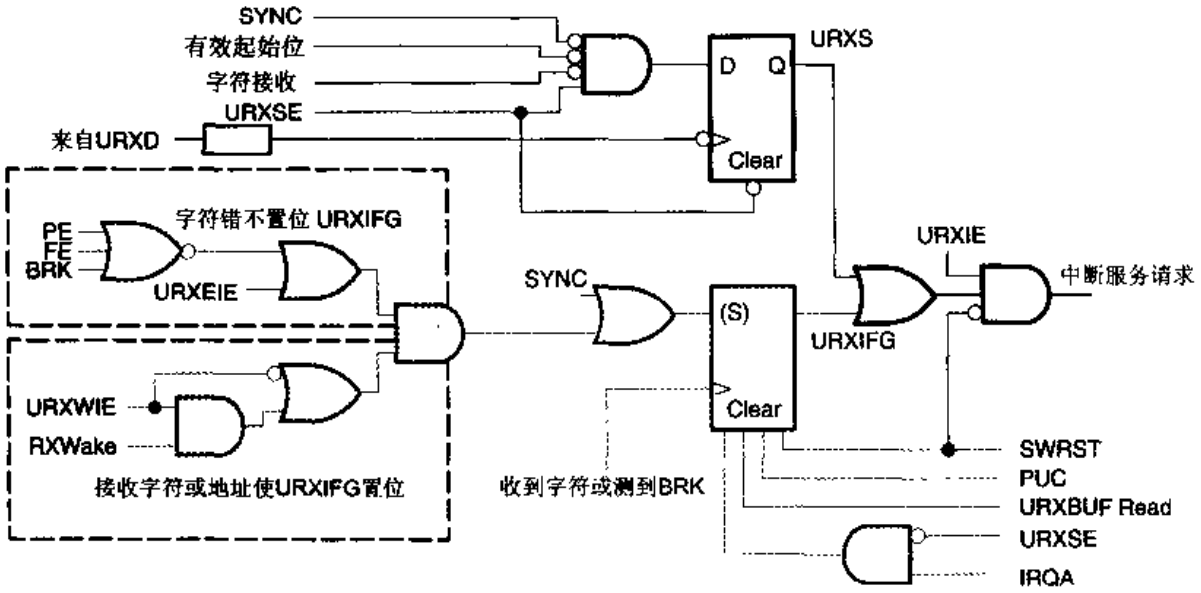


图 12.16　接收起始检测

有三种字符流不能使中断标志 URXIFG 置位，即

- 错误字符(URXEIE=0)。
- 地址字符(URXWIE=1)。
- 检测到无效起始位。

中断服务软件应处理这些情况。

启动条件

进入 USART 模块的 URXD 信号首先进入去毛刺电路，使毛刺不能触发接收起始位标志 URXS。这可防止 URXD 线路上的小毛刺启动通信模块。在噪声环境里，因为毛刺不会启动系统和 USART，电流消耗也会降低，见图 12.17。

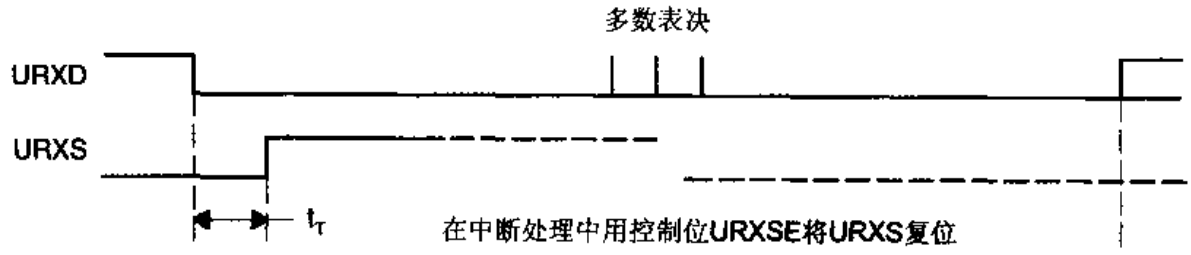


图 12.17　用 URXS 标志接收起始位时序，接受起始位

当 URXD 信号超过去毛刺时间 t_τ，但是起始位的多数表决检测失败时，UART 停止字符的接收，见图 12.18。软件必须处理这种情况，使系统返回到一个适当的低功耗模式，中断标志 URXIFG 不会置位。

MSP430 启动后，URXD 线路上的毛刺被自动压制，不会影响系统，如图 12.19 所示。去毛刺时间 t_τ 的数据在相应的芯片手册中有说明。

在中断服务时需将控制寄存器 UCTL 的 URXSE 复位，以防止 URXS 再次请求中断服务。同时允许接收中断标志 URXIFG 的功能。

附加调整带来极大好处，因为帧的每一位定时都可调整。定时的逐位调整甚至可以不用整数分频。波特率能做到在 32 768 Hz 晶振时得到 4 800 波特，最大误差为 11%。而标准 UART 在同样条件下只能达到的波特率为 75，最大误差已经达到－14.6%。

12.4.3　多处理机模式对节约 MSP430 资源的支持

多字符协议的通信系统可利用线路空闲或者地址位这两种多机模式。第一个字符可能是目标地址、信息标识或其他定义，这一字符由软件去解释。如果对于应用有意义，则采集其后续字符并进一步明确它的定义；毫无意义的首字符将停止处理器的活动。这一特性的应用支持了在接收操作时的中断唤醒特性和用发送来传递唤醒条件，避免无意义的字符激活系统，节约了 MSP430 资源的使用，使系统能维持在最有效的功率消耗状况下。

在多机模式中，错误字符的拒收避免了中断处理这些字符，这点非常有用。处理机将在最有效的功率消耗模式下等待一个需要处理的字符到来。

12.5　波特率计算

MSP430 的波特率发生器使用一个分频器和一个调整器，分频因子 N 由给定的晶振频率和所需的波特率决定。

$$N = BRCLK / 波特率$$

所需的分频因子 N 通常有一个整数部分和一个小数部分。波特率发生器的分频器实现了分频因子 N 的整数部分，调整器尽可能接近地满足小数部分。分频因子 N 定义为

$$N = UBR + (M7+M6+\cdots+M0) / 8$$

其中：N　　目标分频因子；

UBR　UBR1 和 UBR0 中的 16 位数据；

Mx　　调整器的各数据位。

$$波特率 = BRCLK / N = BRCLK / (UBR+(M7+M6+\cdots+M0)/ 8)$$

发送操作的位定时

一帧或一个字符中的每一位的定时是已发送位定时的总和，见图 12.20。波特率发生器相对于所需理想定时值的误差按每一位计算。误差是针对于实际传送位的，而不是对于全体的，见图 12.21。

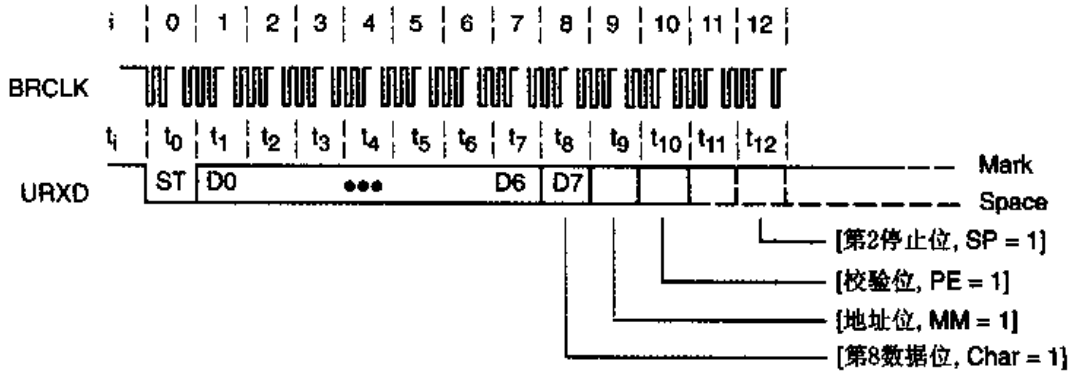


图 12.20　MSP430 发送位定时

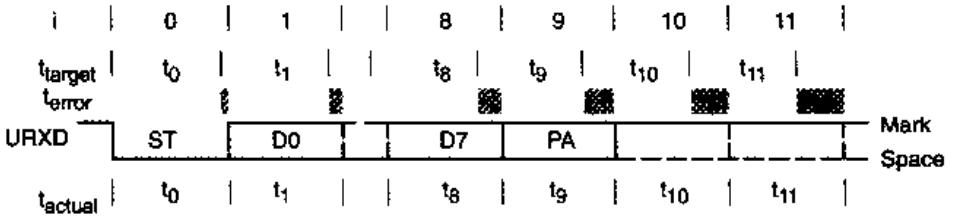


图 12.21 MSP430 发送位定时误差

即使每位误差很小，最终也可能产生大的误差，必须考虑到它们是积累的而不是相对的。每一位的误差可通过下面公式来计算，即

$$\text{Error}\,[\%]=\frac{\sum_{i=0}^{n-1} t_{actual_i}-\sum_{i=0}^{n-1} t_{target_i}}{t_{baud\ rate}}\times 100\%$$

或

$$\text{Error}\,[\%]=\left\{\frac{\text{baud rate}}{\text{BRCLK}}\times[(i+1)\times \text{UBR}+\sum_{i=0}^{n-1} m_i]-(i+1)\right\}\times 100\%$$

其中：baud rate　所需波特率；

BRCLK　输入频率，选自 UCLK、ACLK 或 SMCLK；

i　i=0 针对起始位，i=1 针对数据位 D0，以此类推；

UBR　UBR1 和 UBR0 中的 16 位分频因子。

例 1：2 400 波特

设：　波特率=2 400 波特；

BRCLK=32 768 Hz(ACLK)；

UBR=13，因为理想的分频因子为 13.67；

M=6Bh，即 01101011b，最低位首先使用。

起始　误差[%]=(波特率/BRCLK×((0+1)×UBR+1)−1)×100%=2.54%

D0　误差[%]=(波特率/BRCLK×((1+1)×UBR+2)−2)×100%=5.08%

D1　误差[%]=(波特率/BRCLK×((2+1)×UBR+2)−3)×100%=0.29%

D2　误差[%]=(波特率/BRCLK×((3+1)×UBR+3)−4)×100%=2.83%

D3　误差[%]=(波特率/BRCLK×((4+1)×UBR+3)−5)×100%=−1.95%

D4　误差[%]=(波特率/BRCLK×((5+1)×UBR+4)−6)×100%=0.59%

D5　误差[%]=(波特率/BRCLK×((6+1)×UBR+5)−7)×100%=3.13%

D6　误差[%]=(波特率/BRCLK×((7+1)×UBR+5)−8)×100%=−1.66%

D7　误差[%]=(波特率/BRCLK×((8+1)×UBR+6)−9)×100%=0.88%

校验　误差[%]=(波特率/BRCLK×((9+1)×UBR+7)−10)×100%=3.42%

停止 1　误差[%]=(波特率/BRCLK×((10+1)×UBR+7)−11)×100% =−1.37%

停止 2　误差[%]=(波特率/BRCLK×((11+1)×UBR+8)−12)×100% =1.17%

标准波特率所需要的波特率寄存器和调整寄存器列表如下，针对 32 768 Hz 手表晶振(ACLK)并假定 SMCLK 是 ACLK 的 32 倍条件下。表 12.3 中列出发送和接收的计算误差，在接收时还应考虑同步误差。

表 12.3　常用波特率，波特率数据与误差

波特率	分频信号		ACLK (32 768 Hz)			max. TX 误差	max. RX 误差	同　步 RX 误差	MCLK (1 048 576 Hz)			max. TX 误差	max. RX 误差
	ACLK	MCLK	UBR1	UBR0	UMOD	%	%	%	UBR1	UBR0	UMOD	%	%
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

最大误差是针对接收和发送分别计算的。对于接收，误差是对于每一位扫描的理想正中位置的累积时间；对于发送，误差是对于每一位的理想周期累积时间。

MSP430 的 USART 可以得到尽可能接近时钟频率的波特率。由于利用了对每一位定时的调整，实际误差很小。事实上，标准串行通信允许波特率有高达 20%～30%的误差。

图 12.22 是同步误差示意图。它来源于 URXD 端口信号与内部时钟之间的异步定时，而实际接收信号与 BRSCLK 时钟是同步的。BRSCLK 时钟频率应是位定时的 16～31 倍。

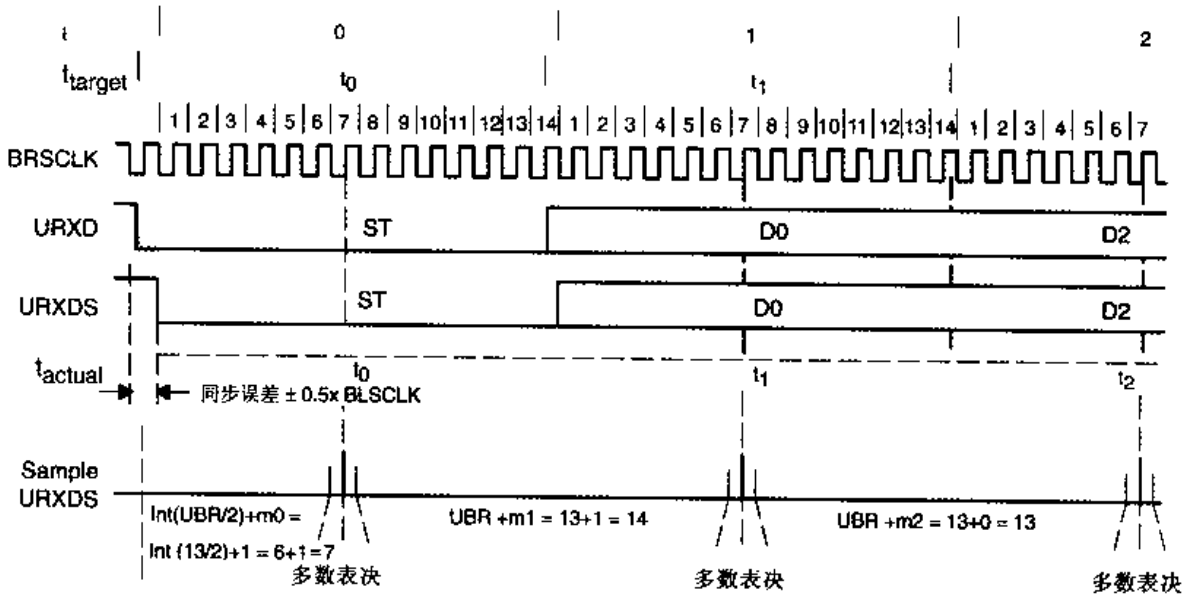


图 12.22　接收定时误差

当：N≤1F　　BRSCLK＝BRCLK；

当：20h≤N≤3Fh　　BRSCLK＝BRCLK/2；

当:40h≤N≤7Fh BRSCLK=BRCLK/4;
当:80h≤N≤FFh BRSCLK=BRCLK/8;
当:100h≤N≤1FFh BRSCLK=BRCLK/16;
当:200h≤N≤3FFh BRSCLK=BRCLK/32;
当:400h≤N≤7FFh BRSCLK=BRCLK/64;
当:800h≤N≤FFFh BRSCLK=BRCLK/128;
当:1000h≤N≤1FFFh BRSCLK=BRCLK/256;
当:2000h≤N≤3FFFh BRSCLK=BRCLK/512;
当:4000h≤N≤7FFFh BRSCLK=BRCLK/1 024;
当:8000h≤N≤FFFFh BRSCLK=BRCLK/2 048。

对于起始位检测的目标波特率定时 t_{target_0} 是波特率定时 $t_{baud\ rate}$ 的一半,因为位测试是在位周期的中间进行的。其他后续位的目标波特率定时 t_{target_i} 就是波特率 $t_{baud\ rate}$。

$$Error[\%]=\frac{t_{actual_0}+t_{taiget_0}}{0.5\times t_{target_0}}+\frac{\sum_{i=1}^{n-1}t_{actual_i}-\sum_{i=1}^{n-1}t_{target_i}}{t_{target_i}}\times 100\ \%$$

或

$$Error[\%]=\left(\frac{baud\ rate}{BRCLK}\times\{2\times[m_0+int(UBR/2)]+(i\times UBR+\sum_{i=1}^{n-1}m_i)\}-1-i\right)\times 100\ \%$$

其中: baud rate 波特率;
BRCLK 输入频率,选自 UCLK、ACLK 或 MCLK;
i i=0 针对起始位,i=1 针对数据位 D0,以此类推;
UBR UBR1 和 UBR0 中的 16 位分频因子。

例 2: 2 400 波特

设: 波特率 = 2 400 波特;
BRCLK = 32 768 Hz(ACLK);
UBR = 13,因为理想的分频因子为 13.67;
M=6Bh,01101011b,最低位首先使用。

起始 误差[%]=(波特率/BRCLK×(2×(1+6)+(0×UBR+0))−0−1)×100%=2.54%
D0 误差[%]=(波特率/BRCLK×(2×(1+6)+(1×UBR+1))−1−1)×100%=5.08%
D1 误差[%]=(波特率/BRCLK×(2×(1+6)+(2×UBR+1))−1−2)×100%=0.29%
D2 误差[%]=(波特率/BRCLK×(2×(1+6)+(3×UBR+2))−1−3)×100%=2.83%
D3 误差[%]=(波特率/BRCLK×(2×(1+6)+(4×UBR+2))−1−4)×100%= −1.95%
D4 误差[%]=(波特率/BRCLK×(2×(1+6)+(5×UBR+3))−1−5)×100%=0.59%
D5 误差[%]=(波特率/BRCLK×(2×(1+6)+(6×UBR+4))−1−6)×100%=3.13%
D6 误差[%]=(波特率/ BRCLK×(2×(1+6)+(7×UBR+4))−1−7)×100%=−1.66%
D7 误差[%]=(波特率/BRCLK×(2×(1+6)+(8×UBR+5))−1−8)×100%=0.88%
校验 误差[%]=(波特率/BRCLK×(2×(1+6)+(9×UBR+6))−1−9)×100%=3.42%
停止1 误差[%]=(波特率/ BRCLK×(2×(1+6)+(10×UBR+6))−1−10)×100% = −1.37%
停止2 误差[%]=(波特率 / BRCLK×(2×(1+6)+(11×UBR+7))−1−11) × 100% = 1.17%

一时钟将串行信息移进或移出。

4 线 SPI 模式用附加的控制线来允许从机数据的发送和接收，它由主机控制。

有 3 或 4 个信号用于数据交换。

(1) SIMO：从进、主出。

方向由 SIMODIR 定义，SIMODIR＝0 为输入。

SIMODIR＝SYNC. and. MM. and. (STC. or. STE)。

当选择 SPI＋主模式时，即选择了输出模式。选择 4 线 SPI 模式(STC＝0)，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 SOMIDIR 定义，SIMODIR＝0 为输入。

SOMIDIR＝SYNC. and…not. (MM). or. (STC. or. STE)。

当选择 SPI＋主模式时，即选择了输出模式。选择 4 线 SPI 模式(STC＝0)，由 STE 上加低电平来强制进入输入方向。

(3) UCLK：USART 时钟，由主机驱动，从机用它发送和接收数据。

方向由 UCLKDIR 定义，UCLKDIR＝0 为输入。

UCLKDIR＝SYNC. and. MM. and. (TSC. or. STE)。

选择 4 线 SPI 模式(STC－0)，由 STE 上加低电平来强制进入输入方向。

(4) STE：从机发送允许，用于 4 线模式中控制多主从系统中的多个从机。

图 13.2 说明 USART 用同步模式与另一设备的串行通道互联时使用一个公共的发送接收移位寄存器，MSP430 可以是主机或从机，操作是相同的。

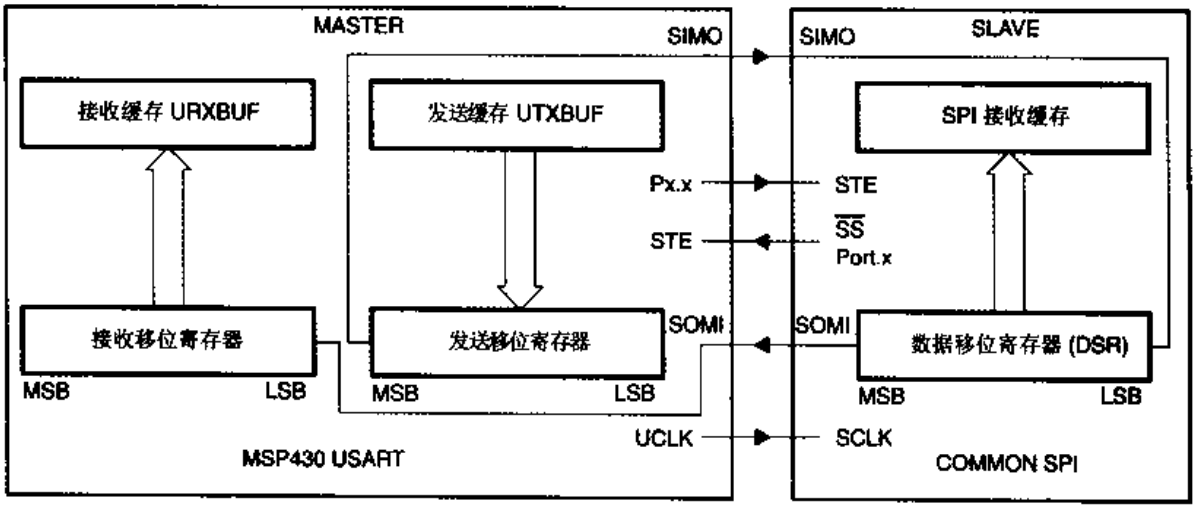


图 13.2 MSP430 的 USART 为主机，其他具有 SPI 模式的设备为从机

主机通过发送 UCLK 信号来初始化发送。主机数据在一个时钟跳变沿移出发送移位寄存器，在另一个反向跳变沿移入接收移位寄存器。从机数据的移位操作相同，用公共的移位寄存器接收和发送数据。主机和从机的发送和接收是同时进行的。

由应用软件来决定数据是否有意义，可能有以下情况：

- 主机发送数据，从机发送伪数据。
- 主机发送数据，从机发送数据。

● 主机发送伪数据，从机发送数据。

图 13.3 及图 13.4 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是 00。会顺序发生以下事件：

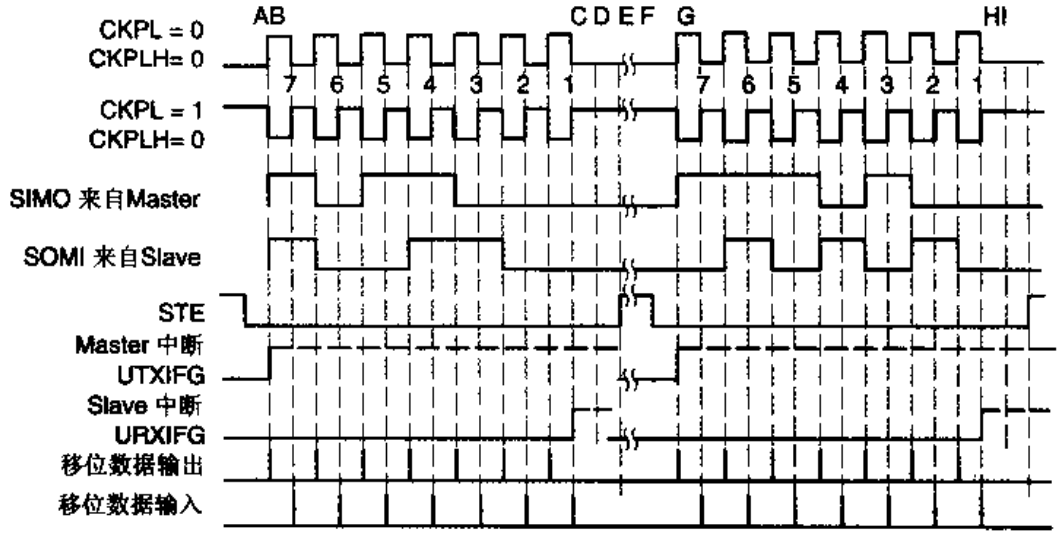
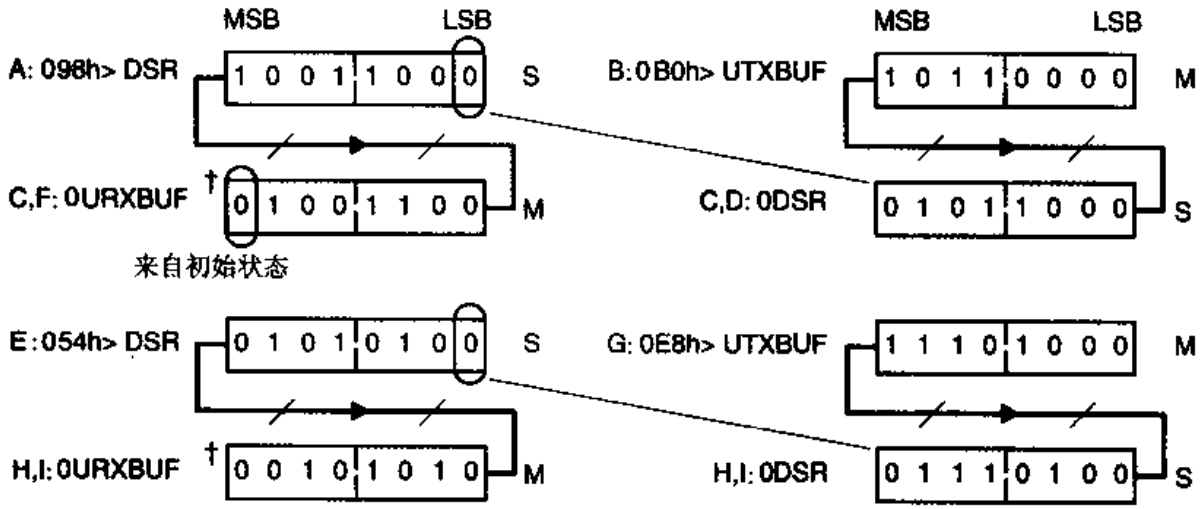


图 13.3　串行同步数据传输



† 对于7位模式，RXBUF的MSB读出总为"0"。
S: Slave ; M: Master。

图 13.4　数据传输循环

A. 从机写 98h 到 DSR 并等待主机将数据移出。

B. 主机写 B0h 到 UTXBUF，它立即送到发送移位寄存器，且开始发送。

C. 完成第一个字符，中断标志置位。

D. 从机从它的接收缓存读出 58 h(右对齐)。

E. 从机写 54 h 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 E8h 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 2 个时钟(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 74 h(右对齐)。

图 13.5 是 MSP430 的 USART 配置成 3 线或 4 线的从机结构。

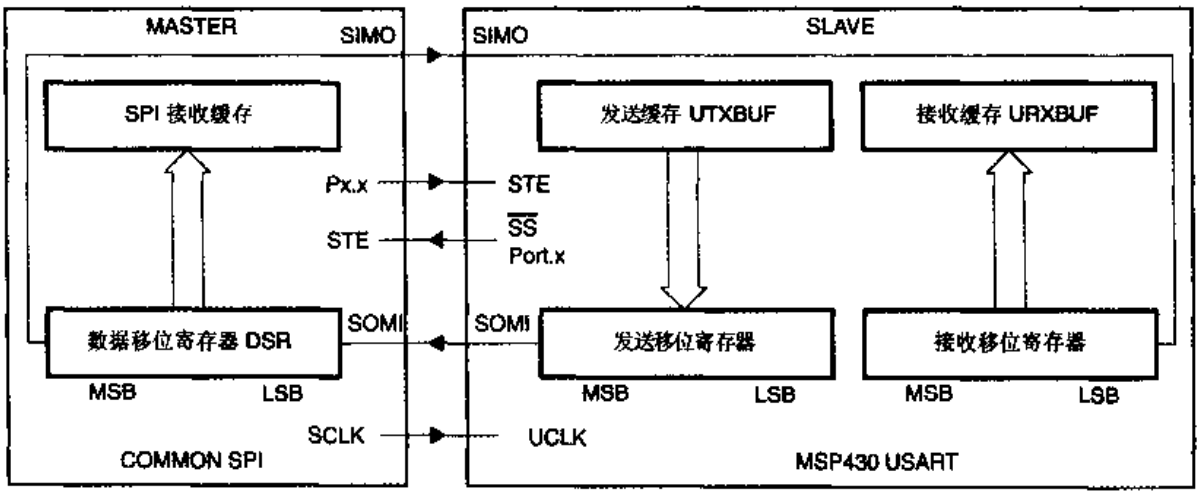


图 13.5 MSP430 的 USART 作为 3 线或 4 线连接的从机

13.1.1 SPI 模式中的主模式

控制寄存器 UCTL 中的主模式位 MM 置位选择主机模式。USART 通过在 UCLK 引脚 UCLK 信号来控制串行通信网。在第一个 UCLK 周期，数据从 SIMO 引脚输出，并在相应的 UCLK 周期的中间，从 SOMI 引脚锁存数据。

一旦移位寄存器为空，已写入发送缓存 UTXBUF 的数据移入移位寄存器，并启动在 SIMO引脚的数据发送，先发送最高有效位。同时，接收数据移入接收移位寄存器，当移完所选定的位数时，接收移位寄存器中的数据移入 URXBUF，并使中断标志 URXIFG 置位。最高有效位首先移入接收移位寄存器，以右对齐的方式存入接收缓存 URXBUF。如果这时前一数据未被读取，则溢出位 OE 置位。

注意：USART 同步主模式，接收初始化

为了接收一个字符，主机必须向发送缓存 UTXBUF 写入数据。当发送移位寄存器为空时接收开始，数据移入这一移位寄存器。接收和发送总是在时钟脉冲的两个反向沿处一起发生的。

用发送中断标志 UTXIFG 或接收中断标志 URXIFG 可以完成协议的控制。数据从移位寄存器发送给从机后，可立即用 UTXIFG 标志将数据从缓存中移入移位寄存器，开始一次发送。从机接收定时应确保能及时获取数据。URXIFG 标志指示数据移出移入完成的时间。主机可利用 URXIFG 确定从机已准备好接收新的数据。

4 线 SPI 主机模式

由激活的主机用 STE 信号防止与别的主机发生总线冲突。如果相应的 PnSEL 位选择模

块功能，则 STE 引脚成为输入。主机在 STE 信号为高时正常操作。当 STE 置为低时，例如另一设备申请成为主机，这时当前的主机作出如下反应：

- 驱动 SPI 总线的 SIMO 和 UCLK 的引脚置为输入。
- 出错位 FE 和 URCTL 中的中断标志 URXIFG 置位。

总线冲突就被消除，即 SIMO 和 UCLK 不再驱动总线线路，同时用出错标志通知软件系统完备性被破坏。当 STE 为低电平时，SIMO、UCLK 引脚被强制变为输入；当 STE 返回高电平时，系统将返回到由相应控制位定义的状态。

在 3 线模式中，STE 输入信号与控制无关。

13.1.2　SPI 模式中的从模式

如果选择为同步模式并且主机模式位 MM 复位，则微控制器进入从机模式。

为接收外部主机提供的串行移位时钟，UCLK 引脚工作在输入状态。数据传输速率由这个时钟信号而不是内部的比特率发生器决定。在开始 UCLK 之前由 UTXBUF 装入到移位寄存器的数据，在主机提供的 UCLK 信号作用下通过 SOMI 引脚发送。同时，在 UCLK 时钟的反向沿 SIMO 引脚上的串行数据移入接收移位寄存器。

接收中断标志 URXIFG 置位，表示数据已被接收并已传送给接收缓存。当新数据写入接收缓存时前一个接收到的数据还未被读取，溢出标志将置位。

4 线 SPI 从机模式

在 4 线 SPI 模式中，STE 信号被从机用作发送、接收允许信号，它由主机提供。当 STE 信号为高时，禁止接收和发送；当 STE 信号为低时，允许接收和发送。无论何时 STE 信号变高，任何已启动的接收操作都将暂停，直到 STE 信号再次变低时继续。STE 信号用于允许一个从机访问数据线路。当 STE 为高时 SOMI 成为输入状态。

13.2　中断与控制功能

USART 外围模块有发送和接收两个主中断源。两个独立的中断矢量，一个对应于接收，一个对应于发送。表 13.1 为 USART 的控制位位于 SFR 地址区。

表 13.1　USART 的控制位位于 SFR 地址区

寄存器	缩　写	初始状态
接收中断标志	URXIFG	复位(用 PUC/SWRST)
接收中断允许	URXIE	复位(用 PUC/SWRST)
接收/发送允许	USPIE	复位(用 PUC)
发送中断标志	UTXIFG	置位(用 PUC/SWRST)
发送中断允许	UTXIE	复位(用 PUC/SWRST)

USART 的发送和接收是并行进行的，并且使用同步主机的同一个波特率发生器。在同步从机中，UCLK 引脚上的外部时钟用于接收和发送。发送器和接收器都用 USPIE 位来允许和禁止。

13.2.1 USART 接收/发送允许位及接收操作

接收允许位 USPIE 允许或禁止接收器从 URXD/SOMI 数据线进行位流的收集。当 USPIE=0时禁止 USART 的接收功能，在完成已开始的接收操作后停止接收。如果无接收操作发生则立即停止接收。在同步模式中，UCLK 不能移动数据到接收移位寄存器中。

USART 接收/发送允许位，MSP430 作为主机

当 MSP430 的 USART 选择为 SPI 主机模式时，它的 3 线和 4 线模式的接收操作是相同的，见图 13.6。

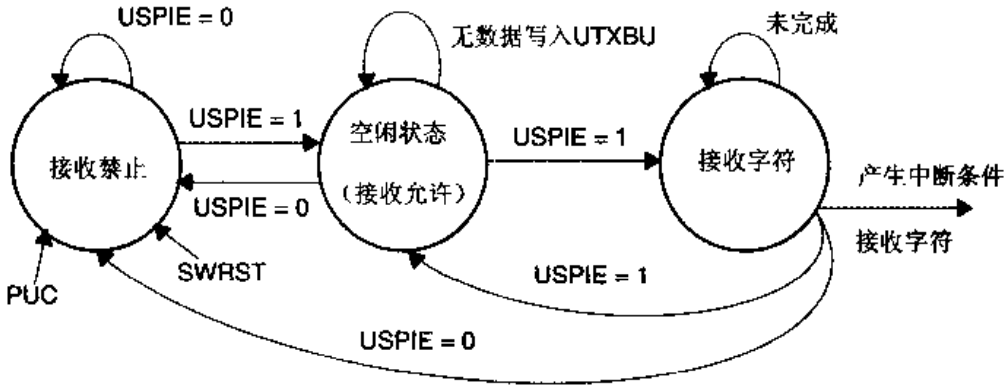


图 13.6 接收允许 UPSIE 的状态图——MSP430 为主机

USART 接收/发送允许位，MSP430 作为从机，3 线模式

当 MSP430 的 USART 选择 SPI 的从机模式时，3 线和 4 线模式的接收操作是不同的。在图 13.7 的 3 线模式中，接收操作一旦启动，没有外部 SPI 接收控制信号能使它停止。只有上电清除 PUC、软件复位 SWRST 或接收允许 USPIE 能停止接收操作，并使 USART 复位。

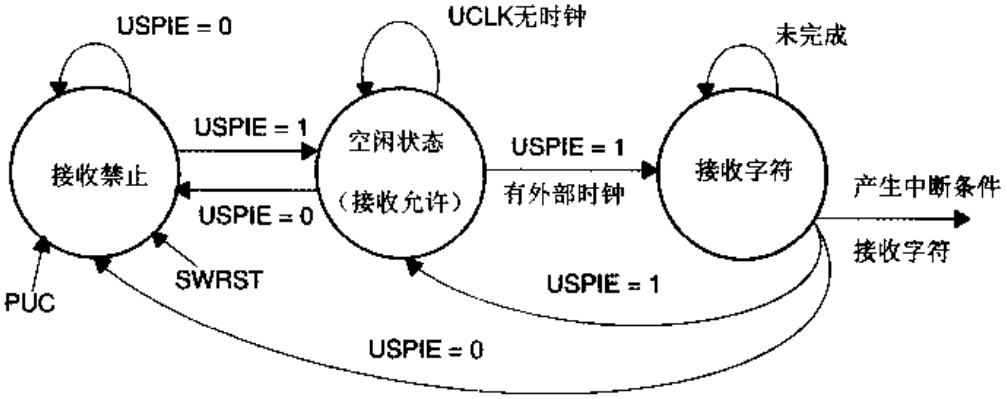


图 13.7 接收允许 USPIE 的状态图——MSP430 为从机，3 线模式

注意：SPI 模式的 USPIE 再允许

由于接收器的完全禁止，接收器的再允许与通信线路的数据流是异步的。要实现与数据流的同步，必须像通常的 3 线 SPI 模式那样采用软件协议来实现。

USART 接收/发送允许位，MSP430 作为从机，4 线模式

在图 13.8 的 4 线模式中，STE 引脚上的外部 SPI 接收控制信号可停止已经启动的接收

操作。上电清除 PUC、软件复位 SWRST 或 USPIE 也能停止接收操作并使操作控制状态机复位。无论何时，当 STE 为高，接收操作就暂停。

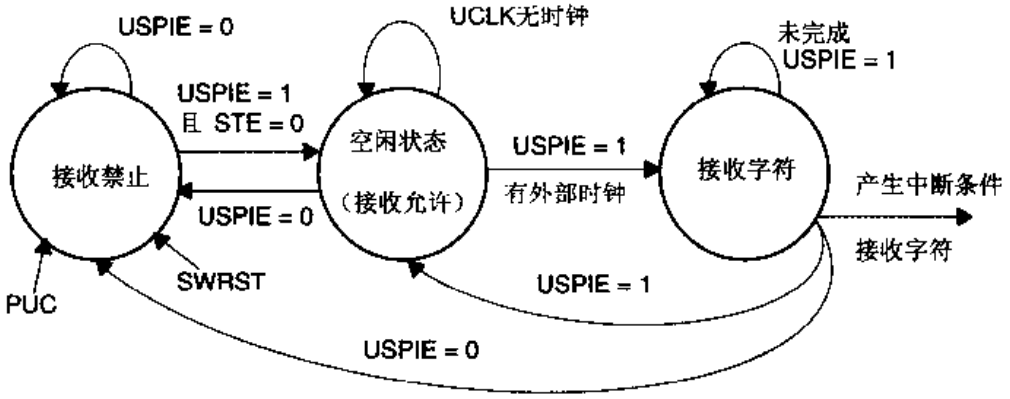


图 13.8　接收允许 USPIE 的状态图——MSP430 为从机，4 线模式

13.2.2　USART 接收/发送允许位及发送操作

发送允许位 USPIE 允许或禁止字符移位到串行数据线。一旦复位将禁止发送器，但是进行中的发送并不暂停，直到已写入发送缓存的数据全部发送完毕。继续向发送缓存写入数据不会产生数据发送。如 UTXBUF 就绪，发送请求将一直保持，一旦 UTXE 置位并且发送器为空，立即启动数据发送。STE 的低电平信号将从总线上取消活动的主机(4 线模式)。STE 为低表明有另一主机请求成为活动的主机。

USART 发送允许，MSP430 是主机

图 13.9 是 MSP430 作为主机时的发送允许状态图。

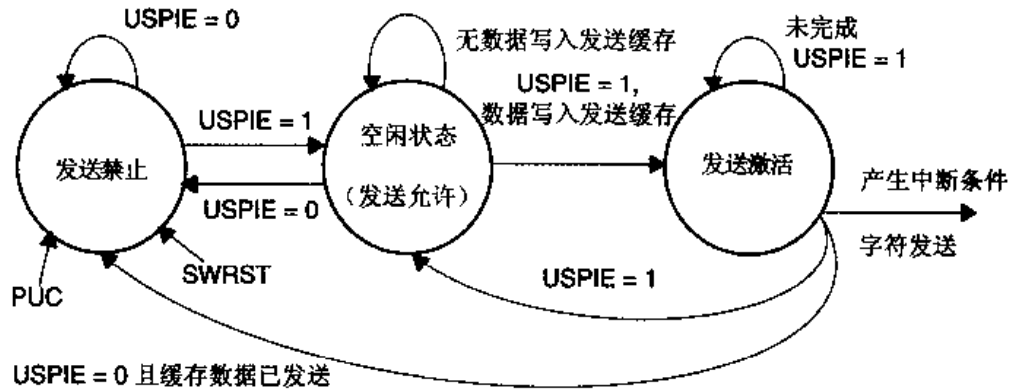


图 13.9　MSP430 为主机的发送允许状态图

USART 发送允许，MSP430 是从机

图 13.10 是 MSP430 为从机时的发送允许状态图。

当 USPIE 复位时，数据能正常写入发送缓存，但是不能启动发送操作。一旦 USPIE 置位，发送缓存中的数据立即装入发送移位寄存器，并启动字符发送。

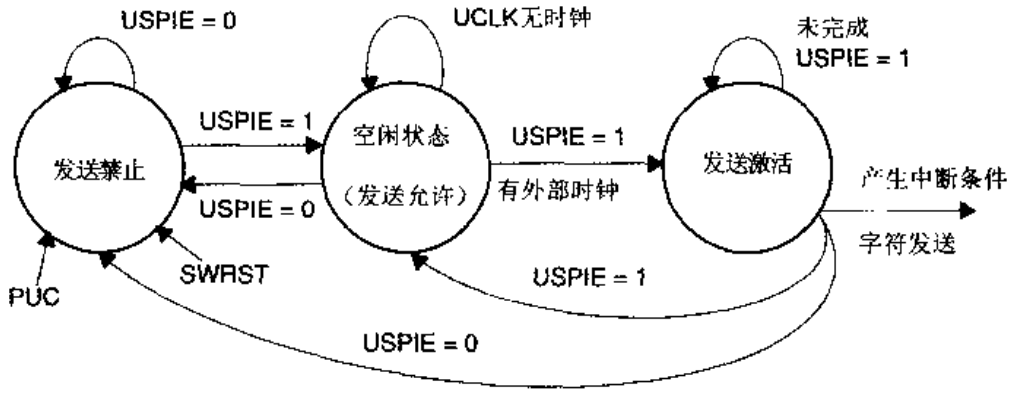


图 13.10 MSP430 为从机的发送允许状态图

注意:写入 UTXBUF,SPI 模式

UTXBUF 未就绪(UTXIFG=0)且发送允许(USPIE=1)时不能写入,否则移出字符是随机的。

注意:写 UTXBUF 与发送器复位,SPI 模式

禁止发送器应在所有数据移入发送移位寄存器后禁止。例如:

```
MOV.B    #…,&UTXBUF
BIC.B    #USPIE,&ME2      ;如果 BITCLK<MCLK,则发送器在缓存数据装入移位
                          ;寄存器之前已经停止工作
```

13.2.3 USART 接收中断操作

接收中断标志 URXIFG 在接收完一个字符并装入接收缓存时置位,见图 13.11。

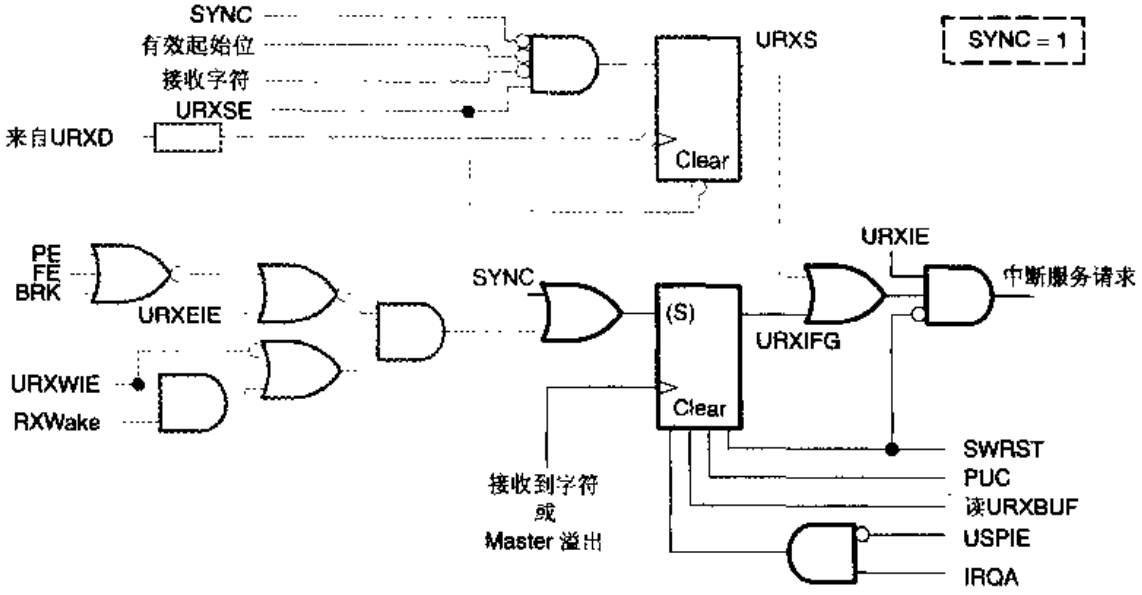


图 13.11 接收中断条件

系统复位 PUC 和软件复位 SWRST 能使 URXIFG 复位。如果执行中断服务或对 URXBUF读操作则将使 URXIFG 自动复位。如果 URXIE 和 GIE 中断允许位置位,则

URXIFG会产生中断。URXIFG 和 URXIE 都因 PUC 和 SWRST 而复位，见图 13.12。

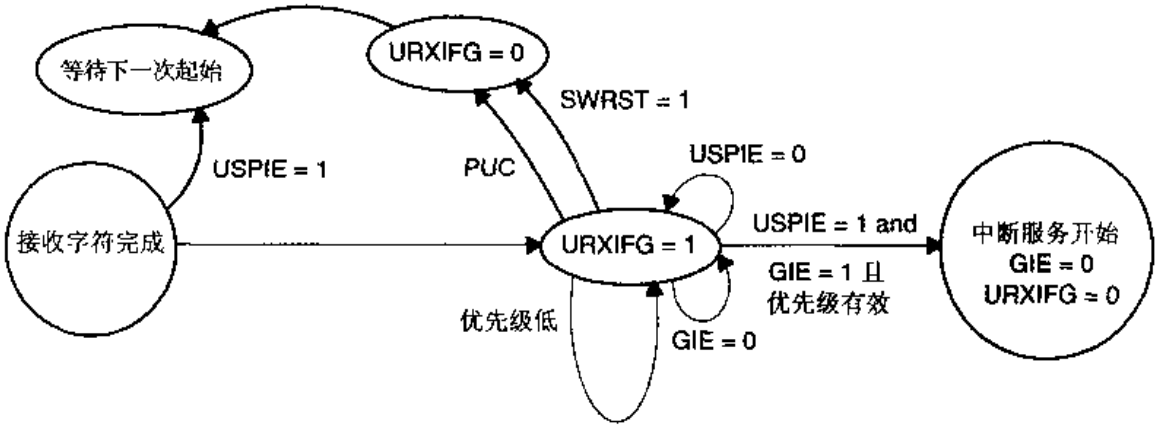


图 13.12　接收中断状态图

13.2.4　USART 发送中断操作

发送器对 UTXIFG 置位，表明发送缓存 UTXBUF 准备好接收下一个字符。如果执行中断请求服务或将字符写入 UTXBUF 中，则 UTXIFG 将自动复位。如果 UTXIE 和 GIE 中断允许位置位，则 UTXIFG 置位时导致一次发送中断请求。UTXIFG 在系统复位 PUC 后或软件复位 SWRST 位后置位，见图 13.13。

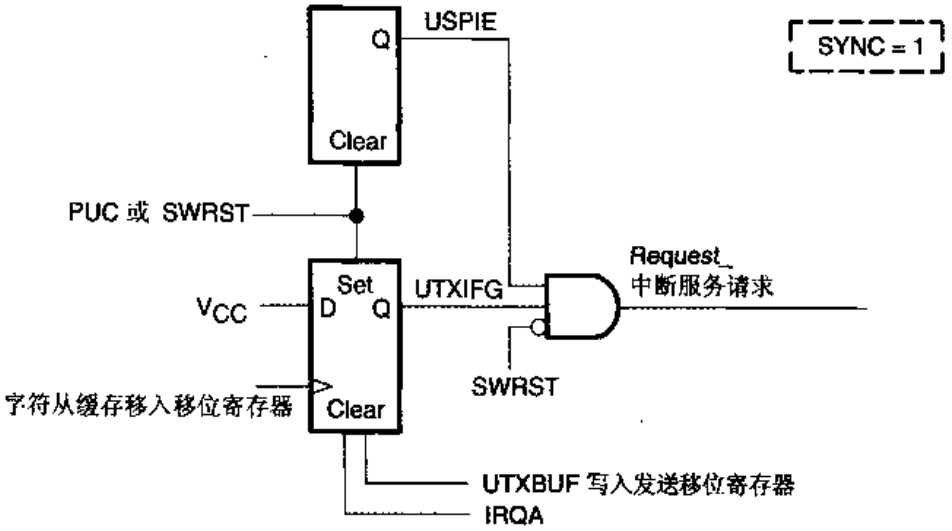


图 13.13　发送中断操作

发送中断允许位 UTXIE 控制 UTXIFG 请求中断的能力，但不影响 UTXIFG 的置位。UTXIE 因 PUC 或 SWRST 位而复位。系统复位 PUC 或软件复位 SWRST，使 UTXIFG 置位，使 UTXIE 复位，这确保了完整的中断控制能力。

13.3　控制与状态寄存器

USART 硬件是字节结构，因此必须以字节指令访问(后缀. B)。表 13.2 及表 13.3 是控制及状态寄存器一览表。

表 13.2 USART0 控制状态寄存器

寄存器	缩 写	寄存器类型	地 址	初始状态
USART 控制寄存器	UCTL0	读/写	070h	见各位说明
发送控制寄存器	UTCTL0	读/写	071 h	
接收控制寄存器	URCTL0	读/写	072 h	
调整控制寄存器	UMCTL0	读/写	073 h	不变
波特率寄存器 0	UBR00	读/写	074 h	不变
波特率寄存器 1	UBR10	读/写	075 h	不变
接收缓存	URXBUF0	读/写	076 h	不变
发送缓存	UTXBUF0	只读	077 h	不变

表 13.3 USART1 控制状态寄存器

寄存器	缩 写	寄存器类型	地 址	初始状态
USART 控制寄存器	UCTL1	读/写	078 h	见各位说明
发送控制寄存器	UTCTL1	读/写	079 h	
接收控制寄存器	URCTL1	读/写	07A h	
调整控制寄存器	UMCTL1	读/写	07B h	不变
波特率寄存器 0	UBR01	读/写	07C h	不变
波特率寄存器 1	UBR11	读/写	07D h	不变
接收缓存	URXBUF1	读/写	07E h	不变
发送缓存	UTXBUF1	只读	07F h	不变

除非在功能描述中说明，PUC 后所有位是随机的。

由 PUC 或 SWRST 位执行 USART 复位。PUC 后如 SWRST 保持置位，USART 将维持在复位状态，直到 SWRST 复位使 USART 的复位状态结束。PUC 后禁止 SPI 模式。

由定义 SYNC 位确定 USART 模块工作在异步模式或同步模式。控制寄存器的各位在两种模式中功能上有差异。下面对各位及其功能的描述都是在同步模式下，即 SYNC＝1；异步模式下的功能已在 USART 串行接口 UART 模式部分中描述。

13.3.1 USART 控制寄存器

控制寄存器 UCTL 的信息决定了 USART 模块的基本操作。寄存器位选择通信模式和字符位数。各位根据选择模式的编程都应在由于 SWRST 复位使 USART 被允许之前进行。

UCTL0(070h)、UCTL1(078h)

7							0
未用	未用	未用	CHAR	Listen	SYNC	MM	SWRST
rw－0	rw－0	rw－0	rw－0	rw－0	rw－0	rw－0	rw－1

从机。

在从机模式，SSEL1、0 的数据无关紧要，这时总是用外部 UCLK 信号作为时钟。

b6～7：控制时钟极性 CKPL 和时钟相位 CKPH，见图 13.14。

CKPL 位控制 SPICLK 信号的极性。

CKPL=0：低电平为无效电平，数据在 UCLK 的上升沿输出，输入数据在下降沿锁存；

CKPL=1：高电平为无效电平，数据在 UCLK 的下降沿输出，输入数据在上升沿锁存。

CKPH 位控制 SPICLK 信号的相位。

CKPH=0：正常的 UCLK 时钟；

CKPH=1：UCLK 被延迟半个周期。

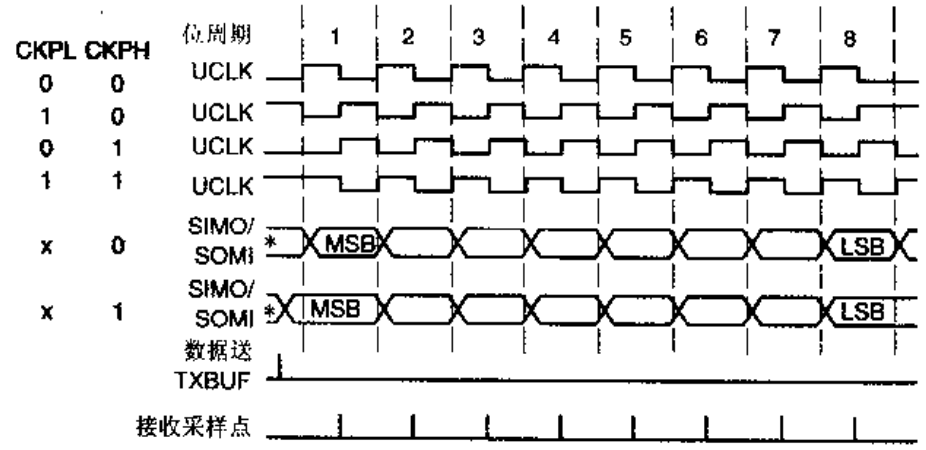


图 13.14 USART 的时钟相位和极性

当工作在 CKPH 置位时，同步模式 USART 在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第一位。数据在 UCLK 的第一个沿锁存，在第二个沿发送。

13.3.3 接收控制寄存器 URCTL

URCTL 控制与接收相关的 USART 硬件，并保存出错信息。

URCTL0(072h)、URCTL1(07Ah)

7							0
FE	未定义	OE	未定义	未用	未用	未定义	未定义
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

b0～1：未定义，由 USART 硬件驱动。

b2～3：未用。

b4：未定义，由 USART 硬件驱动。

b5：当一个字符写入 URXBUF 时前--字符还未被读出时，则溢出标志 OE 置位，这时前--字符被覆盖而丢失。OE 通过 SWRST 位、系统复位、读取 URXBUF 和用指令来复位。

b6：未定义，由 USART 硬件驱动。

b7：帧错。选择 4 线模式时，因总线冲突使有效主机停止并在 STE 引脚信号出现负跳变使 FE 位置位。FE 位通过 SWRST 位、系统复位、读 URXBUF 和用指令来复位。

13.3.4　波特率选择和调制控制寄存器

波特率发生器用波特率选择寄存器的 UBR1 和 UBR2 来产生串行数据流的位定时。最小的分频因子为 2。

UBR00(074h)、UBR00(07Ch)

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

UBR11(075h)、UBR11(07Dh)

7							0
2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8
rw	rw	rw	rw	rw	rw	rw	rw

$$\text{波特率} = \text{BRCLK} /(\text{UBR}+(m7+m6+\cdots+m0)/8)$$

其中：UBR 是 UBR1，UBR0 中的 16 位数据。

可选择的最大发送波特率，主机模式时为波特率发生器输入时钟的一半，从机模式时是在 UCLK 上提供的外部时钟。

在串性同步通信时不需要用调整寄存器，最好将它复位(各位为“0”)。

UMCTL0(073h)、UMCTL1(07Bh)

7							0
M7	M6	M5	M4	M3	M2	M1	M0
rw	rw	rw	rw	rw	rw	rw	rw

13.3.5　USART 接收数据缓存 URXBUF

接收缓存 URXBUF 含有此前从接收移位寄存器来的数据，由 SWRST 位或 PUC 清除。读取 URXBUF 中数据，将使接收出错位和中断标志 URXIFG 复位。

URXBUF0(076h)、URXBUF1(07Eh)

7							0
b7	b6	b5	b4	b3	b2	b1	b0
rw	rw	rw	rw	rw	rw	rw	rw

在 7 位字长模式，URXBUF 的最高位总是为“0”。

13.3.6　USART 发送数据缓存 UTXBUF

发送缓存含有当前需要发送器发送的数据。

第 14 章　比较器 Comparator_A

比较器 A(Comparator_A)是一个实现模拟电压比较的外围模块。可以支持多种 A/D 转换功能的实现。

比较器 A 模块包括以下部分：

- 比较器，可以提供 on/off 信号，无输入回差。
- 内部有模拟参考电平发生器。
- 内部参考电平可向外提供。
- 比较器输入可以切换。
- 比较器的输出有 RC 滤波电路，软件可选。
- 具有中断向量。

14.1　概　述

比较器 A 模块的结构见图 14.1。它的主要应用有：实现精密的斜坡 A/D 转换、电池电压

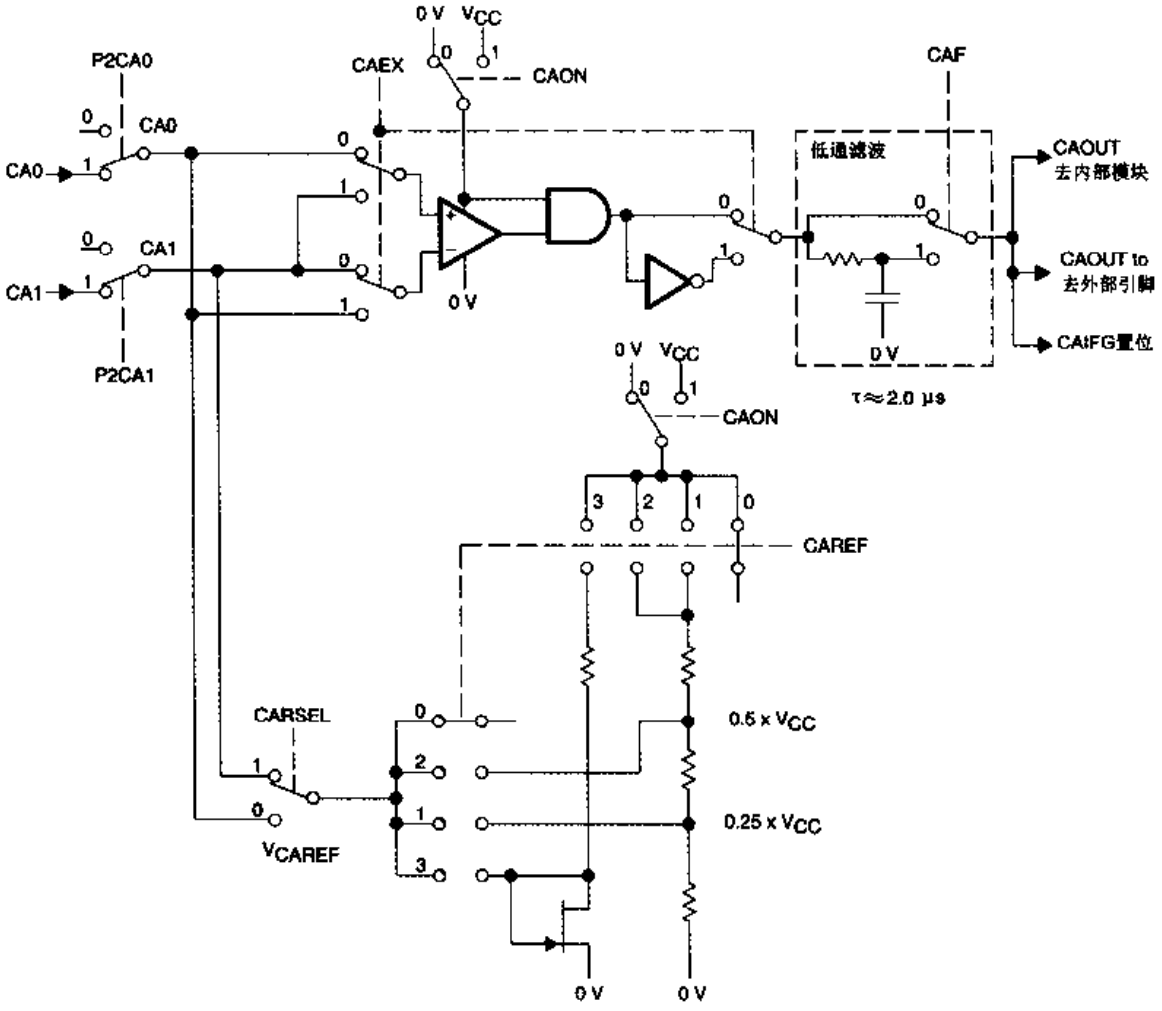


图 14.1　比较器 A 结构

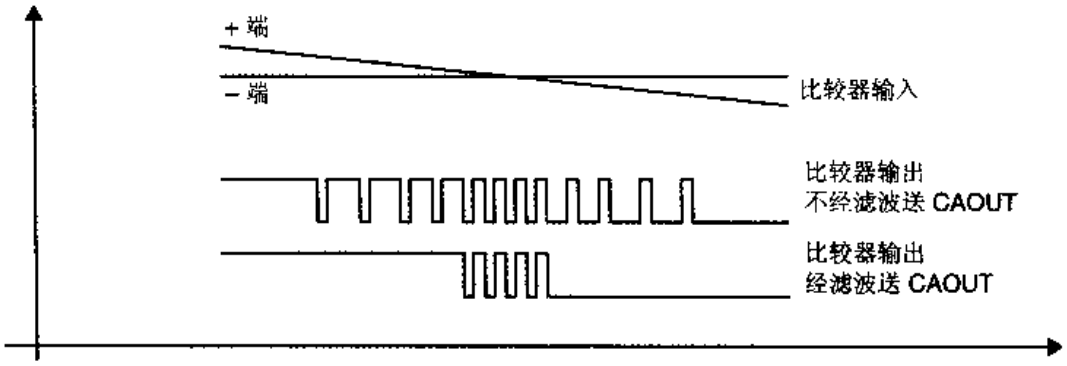


图 14.2　RC 滤波器对比较器输出的响应

14.2.5　参考电平发生器

参考电平发生器产生 V_{CAREF}，V_{CAREF}可以加载在比较器的任一输入端，由控制位 CAREF0 和 CAREF1 控制发生器的输出，控制位 CARSEL 选择 V_{CAREF}加到比较器的输入端。如果比较器的输入端接的都是外部信号，则应该将参考电平发生器关闭，以减少电流消耗。用参考电平发生器的内部分压器，参考电平发生器可以产生为 V_{CC}分压的参考电平，也可以是一个固定的三极管阈值电压。这一阈值电压有一定容差，请查阅芯片手册。

采用比例测量原理，例如，比较来自电阻或电容的未知电压和来自精密电阻或电容的已知电压，可以在不知道电源电压 V_{CC}的情况下得到精确的结果。V_{CC}需要稳定，但是数值不必知道。比例测量的精度取决于已知电阻或电容的数值。

采用绝对测量原理需要 V_{CC}稳定，以确保能产生精密的参考电平。

14.2.6　比较器 A 中断电路

比较器 A 有一个中断电路，有一个中断向量，见图 14.3。比较器输出的上升沿或下降沿都可以使中断标志 CAIFG 置位。由中断触发沿选择位 CAIES 选择使 CAIFG 置位的触发沿。中断允许位 CAIE 和 GIE 位控制 CAIFG 是否能产生中断请求。当 CAIE 和 GIE 都置位时，CAIFG 才能向 CPU 请求中断服务。发生中断服务时 CAIFG 会自动复位。控制位 CAIFG、CAIES 和 CAIE 在比较器 A 的控制寄存器 CACTL1 中。

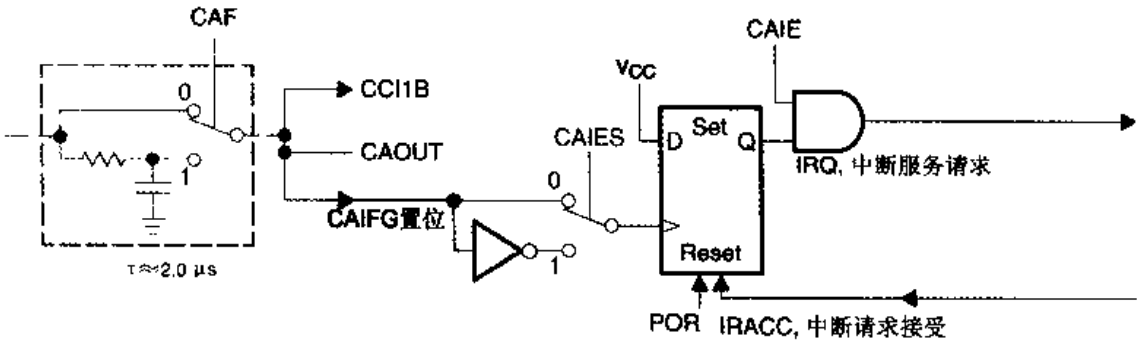


图 14.3　比较器 A 的中断电路

14.3 比较器 A 控制寄存器

比较器 A 模块由 3 个寄存器控制，见表 14.1。寄存器安排在字节模块空间，必须用字节指令访问(后缀 B)。

表 14.1 比较器 A 的控制寄存器

寄存器	缩 写	寄存器类型	地 址	初始状态
比较器控制 1	CACTL1	读/写	059h	复位
比较器控制 2	CACTL2	读/写	05Ah	复位
比较器端口禁止	CAPD	读/写	05Bh	复位

14.3.1 控制寄存器 CACTL1

CACTL1(059h)

7							0
CAEX	CARSEL	CAREF1	CAREF0	CAON	CAIES	CAIE	CAIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

b0：CAIFG。比较器 A 的中断标志。

b1：CAIE。比较器 A 的中断允许。

b2：CAIES。比较器 A 的中断触发沿选择。

0：比较器 A 输出的上升沿使 CAIFG 置位；

1：比较器 A 输出的下降沿使 CAIFG 置位。

b3：CAON。控制比较器 A 的开关。关闭时比较器不消耗电流，但是与参考电平发生器的电流消耗控制是相互独立的。

0：比较器 A 关闭，这时不消耗电流，比较器 A 输出为低；

1：比较器 A 打开，处于工作状态。

b4～5：CAREF。参考电平发生器控制。

0：内部参考电平关闭，这时可以用外部的参考电平；

1：选择 0.25 × V_{CC}为参考电平；

2：选择 0.5 × V_{CC}为参考电平；

3：二极管参考电平，此电压会随芯片、温度、电源而变化，参见芯片手册。

b6：CARSEL。选择内部参考电平加到比较器 A 的(+)端或(−)端。

0：CAEX=0 时，参考电平加(+)端，CAEX=1 时，参考电平加(−)端；

1：CAEX=0 时，参考电平加(−)端，CAEX=1 时，参考电平加(+)端。

b7：CAEX。交换比较器 A 的输入端。用于测量和补偿比较器 A 的偏压。

14.3.2　控制寄存器 CACTL2

CACTL2(05Ah)

7							0
CACTL2.7	CACTL2.6	CACTL2.5	CACTL2.4	P2CA1	P2CA0	CAF	CAOUT
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r-(0)

b0：CAOUT。比较器 A 输出。写入时不会改变。

b1：CAF。选择比较器 A 输出的滤波器。

0：滤波器被旁路；

1：比较器 A 输出经过滤波器。

b2：P2CA0。控制比较器 A 的输入端 CA0。

0：外部引脚信号不连接到比较器 A；

1：外部引脚信号连接到比较器 A。

b3：P2CA1。控制比较器 A 的输入端 CA1。

0：外部引脚信号不连接到比较器 A；

1：外部引脚信号连接到比较器 A。

b4～7：请查阅芯片手册。

14.3.3　端口禁止寄存器 CAPD

比较器 A 的输入及输出常与其他数字 I/O 引脚共用，以减少芯片的引脚。对于斜坡 A/D 为了充放电常要用到多个数字端口来构成多通道 A/D。在这些应用中，数字 I/O 引脚可以关闭输入缓冲的特性很有用。除了需要用作转换的通道外，其他引脚可以关闭缓冲电路成为高阻输入端，来降低电流消耗。

对于 MSP430 的有比较器 A 的型号，数字 I/O 端口就具有这一功能。由 CAPD 控制。

CAPD 的各控制位的初始状态为复位，允许各输入端口的缓冲电路打开。如果相应的 CAPD.x 置位，端口的输入缓存即被关闭。端口与比较器 A 的共用引脚请查阅芯片手册。

CAPD(05Bh)

7							0
CAPD.7	CAPD.6	CAPD.5	CAPD.4	CAPD.3	CAPD.2	CAPD.1	CAPD.0
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

b0～7：CAPD.x，　0：输入缓冲开；

1：输入缓冲关。

14.4　比较器 A 应用

比较器 A 有以下用途：

- 电阻元件测量。
- 外部电压电流检测。

- 外部电压电流测量。
- 测量系统的电池电压。

14.4.1 模拟信号在数字端口的输入

比较器 A 的输入常与数字 I/O 端口共用引脚。当模拟信号加在数字端口的 CMOS 门电路上时，会有由正端(V_{DD}、V_{CC})到负端(V_{SS}、GND)的寄生电流产生。在输入电压接近输入门电路的跃迁电平时发生，见图 14.4。

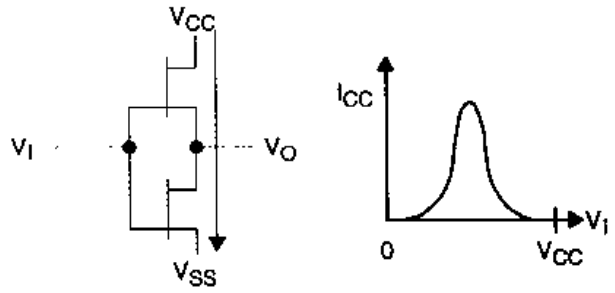


图 14.4 CMOS 非门输入缓冲的传输特性

具有比较器 A 的 MSP430 芯片，在数字 I/O 电路上增加了关闭控制电路，见图 14.5。输入缓冲由 CAPD.x 控制全部 I/O 端口，而不是仅仅比较器 A 的输入端，关闭一个引脚的输入缓冲也关闭了寄生电流，从而降低了总的电流消耗。

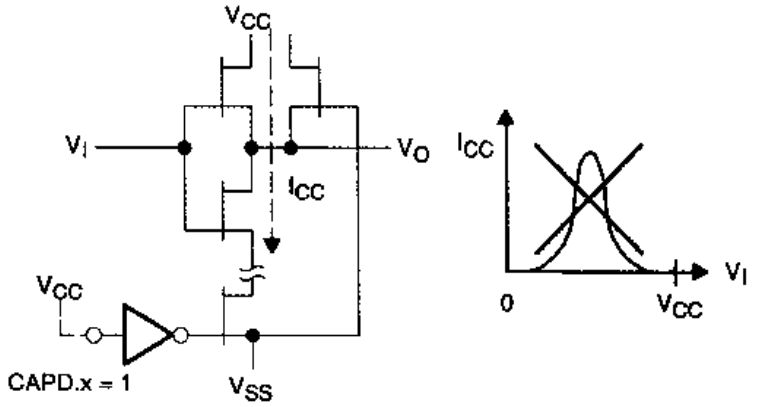


图 14.5 加 CMOS 门控的传输特性和能耗

如果降低功耗是系统设计的重要原则时，则应将那些不需要接受外部驱动的 I/O 端口的输入缓冲，见图 14.6 的实例。

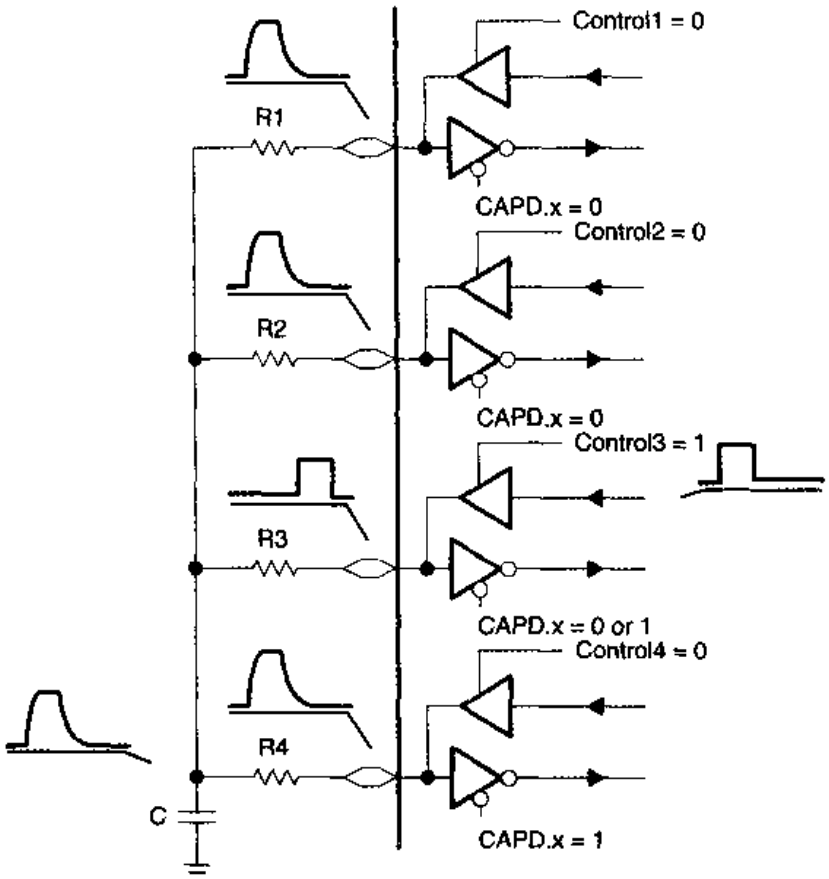


图 14.6　应用实例,CAPD 控制

14.4.2　比较器 A 测量电阻元件

比较器 A 可以用于测量电阻元件。例如,利用热敏电阻,比较热敏电阻与一个标准电阻的放电时间,可以将温度转换成数字,见图 14.7。

电阻元件配合一个电容,通过充放电周期进行比较,见图 14.8。测量基于比例转换原理,即比较两个放电时间。电压和电容的取值是不重要的,因为在比例测量原理中,这些因素在计算中已消除了,只是要求电压(V_{CC})和电容的值在测量过程中保持恒定。

计算原理如下:

因为　$N_{meas} / N_{ref} = (-R_{meas} \times C \times \ln (V_{ref}/V_{CC})) / (-R_{ref} \times C \times \ln (V_{ref}/V_{CC}))$

所以　$N_{meas} / N_{ref} = R_{meas} / R_{ref}$

有　$R_{meas} = R_{ref} \times (N_{meas} / N_{ref})$

MSP430 用于温度测量计算,要用到以下资源:

数字 I/O 端口:

用两次数字输出对电容作充放电。端口置位时,输出 V_{CC} 对电容充电,复位时使电容放电。在不工作时用 CAPD. x 将它关闭。其中,一次让电容经过已知阻值的参考电阻放电,另一次让电容经过测量用的热敏电阻放电。

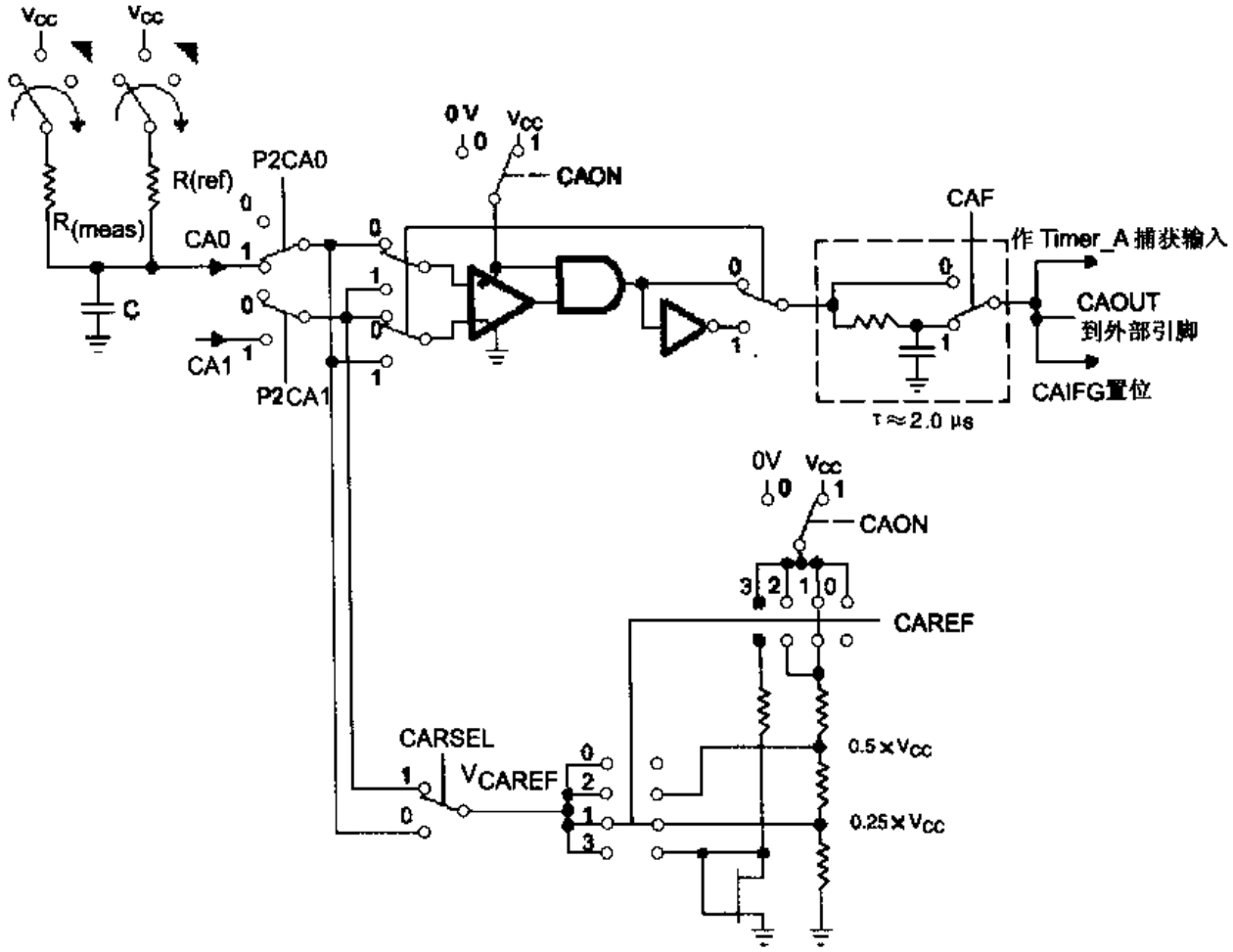


图 14.7 温度测量系统

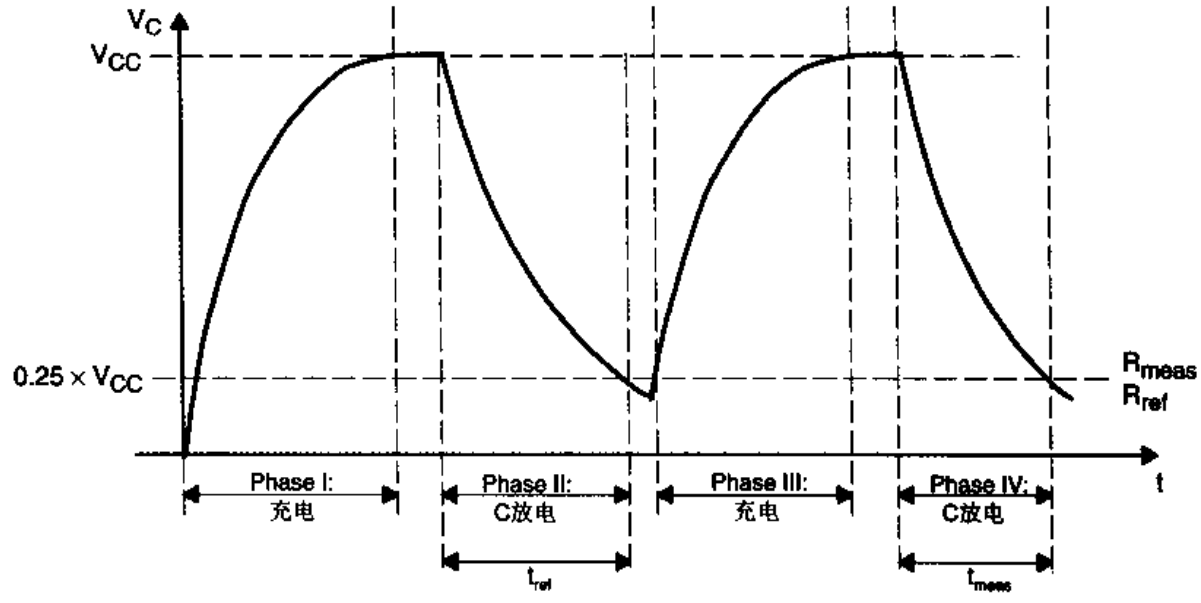


图 14.8 温度测量系统的时序

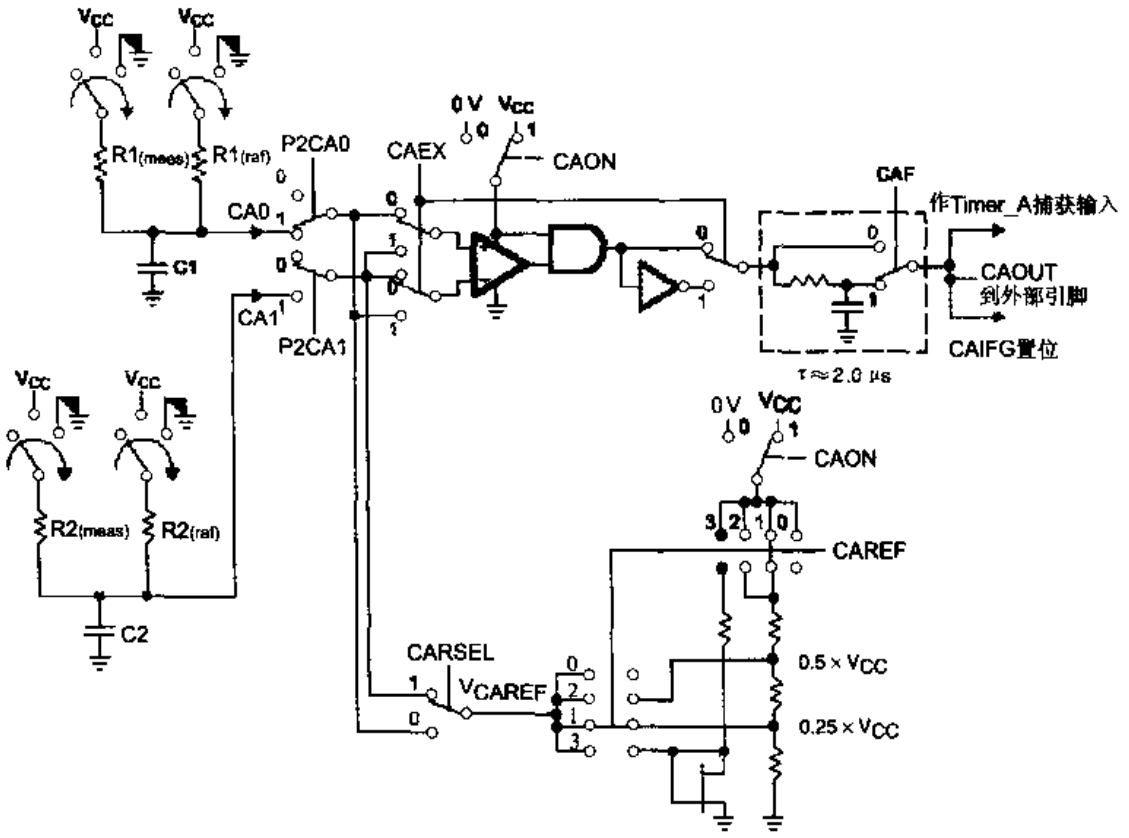


图 14.10 上通道经过 R1(meas)测量温度

14.4.4 比较器 A 检测电流或电压

比较器 A 可以根据电平高于或低于参考电平来检测电流及电压，见图 14.11。参考电平

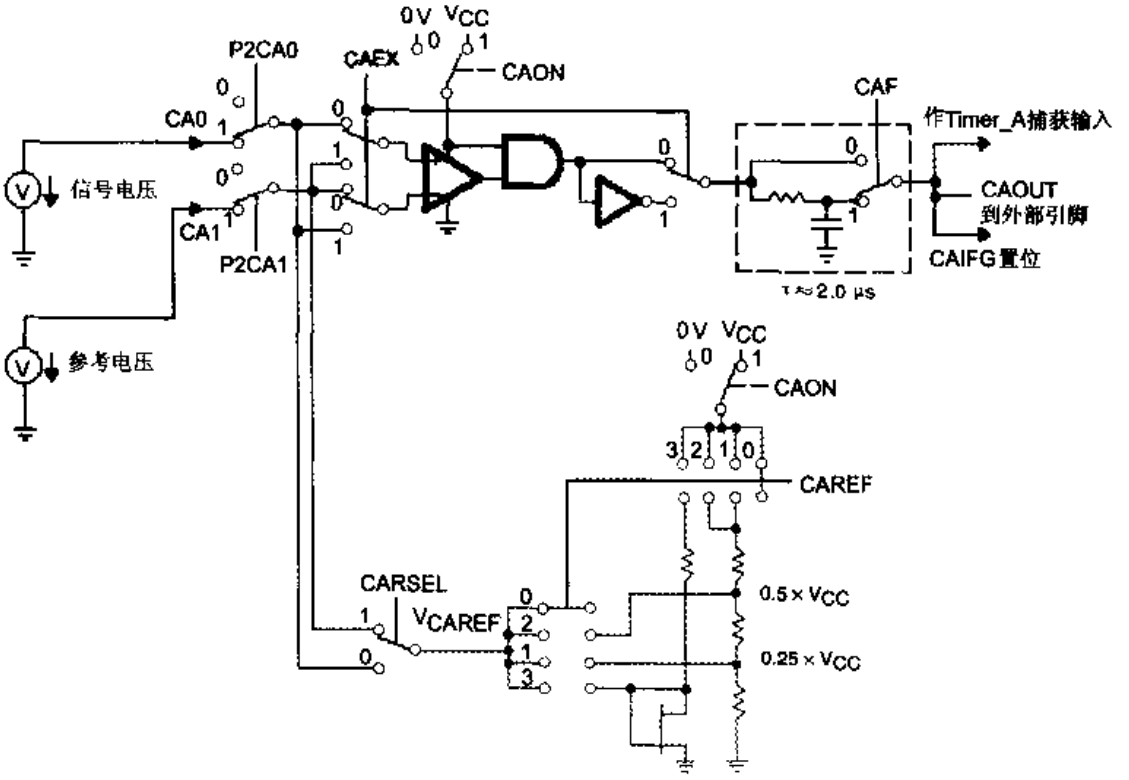


图 14.11 利用外部参考电平的电压检测电路

可以选择内部的参考电平发生器，也可以用外部电压。应用软件可以查询 CAOUT 位来判断比较器 A 的状态。也可以用中断标志 CAIFG 来判断电流及电压源是否跨越了比较器的阈值。

图 14.12 是两个外部电压的比较电路。其中一个是参考电平 V_{ref}。应用软件通过查询 CAOUT 来判断信号电压 V_S。

当 CAOUT=0，$V_S<V_{ref}$；当 CAOUT=1，$V_S>V_{ref}$。

图 14.12 是将电流经电阻转换成电压进行比较检测，电流检测的门限选择 0.25 × V_{CC}。

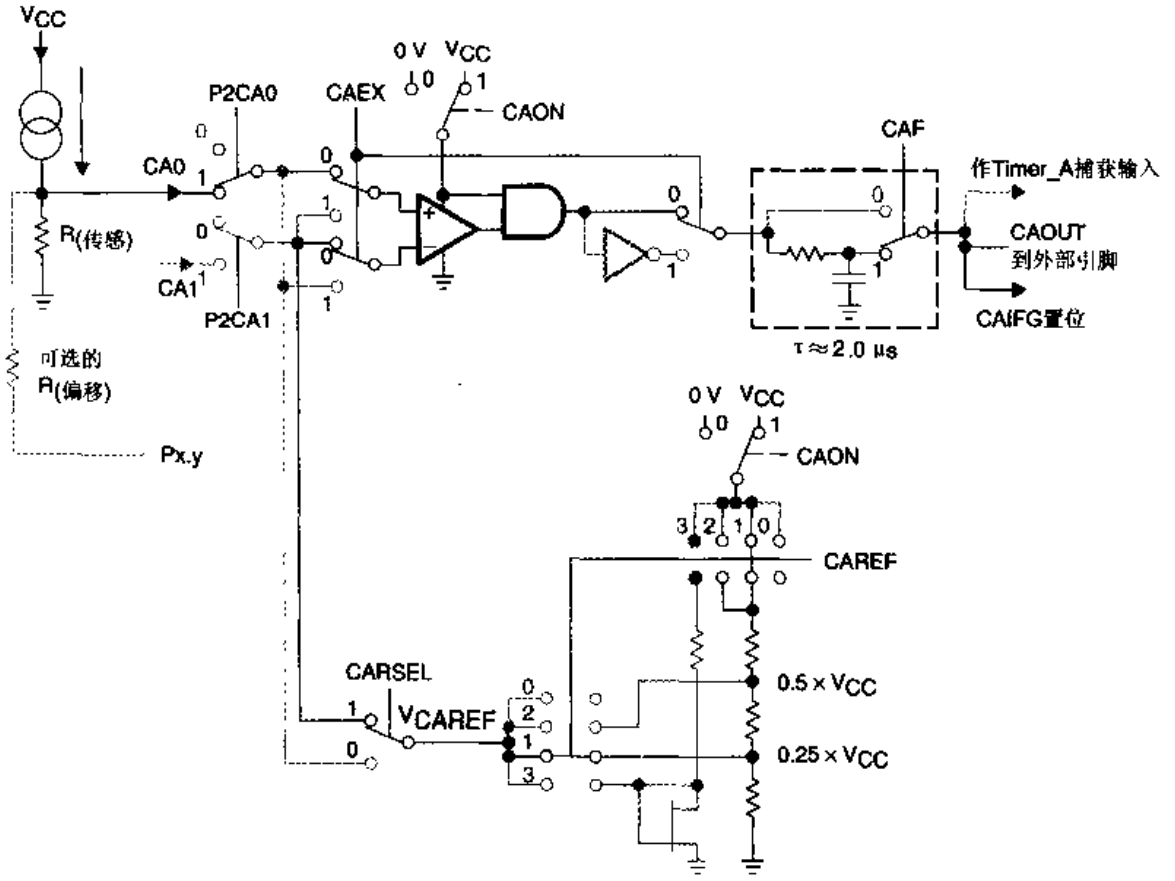


图 14.12　利用内部参考电平的电流检测电路

14.4.5　比较器 A 测量电流或电压

在电平检测的基础上，比较器可以用于电流或电压的测量。为了测量电压，要用一个已知的、稳定的电压源对一个 RC 回路充电至阈值。然后，用待测电压对同一个 RC 回路充电至阈值所需的时间，就可以计算待测电压值。如果系统的 V_{CC} 是稳定而精确的，则 V_{CAREF} 就可以用作这个稳定的已知电压。电路及时序见图 14.13 和 14.14。

电压测量公式为

$$V_{(mess)} = (R1+R2) / R2 \times V_{CC} \times (1-e^{(t_1 \times \ln 0.3)/(t_1+t_2)})$$

电流测量的方法与电压测量非常接近。同样需要一个已知的、稳定的电压源对一个 RC 回路充电至阈值。待测电流通过一个已知电阻转换成电压来测量。电路及时序见图 14.15 和图 14.16。

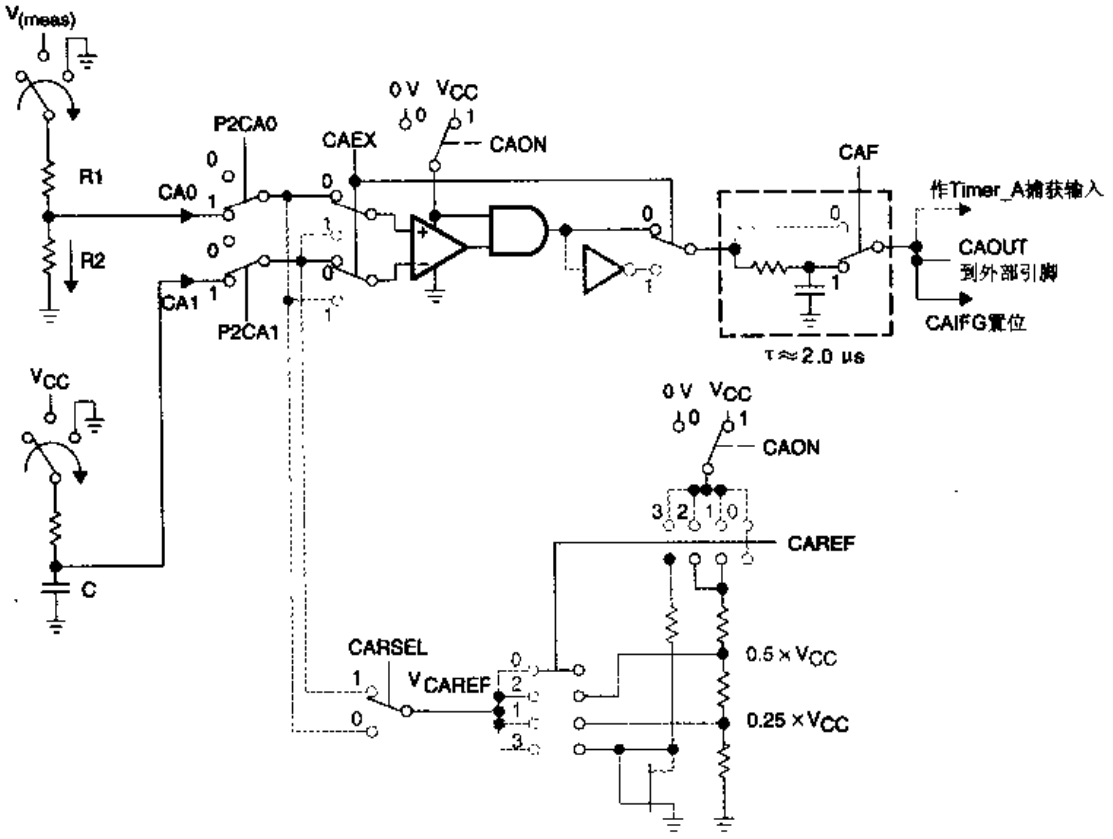
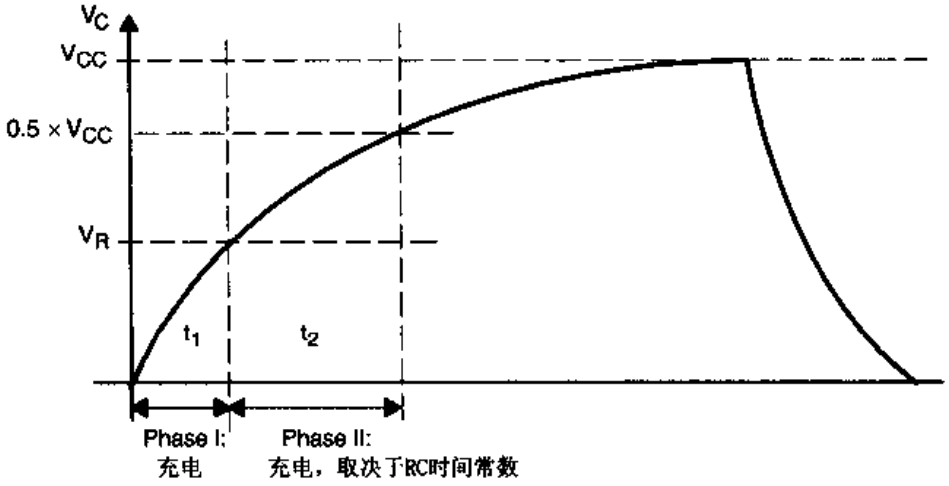


图 14.13 电压测量电路



注：phase I：P2CA0 = 1 和 CAREF = 0。
phase II：P2CA0 = 0，CARSEL = 0 和 CAREF= 1。

图 14.14 电压测量时序

电流测量公式为

$$I_{(meas)} = 1/R_{sense} \times V_{CC} \times (1 - e^{(t_1 \times \ln 0.3)/(t_1 + t_2)})$$

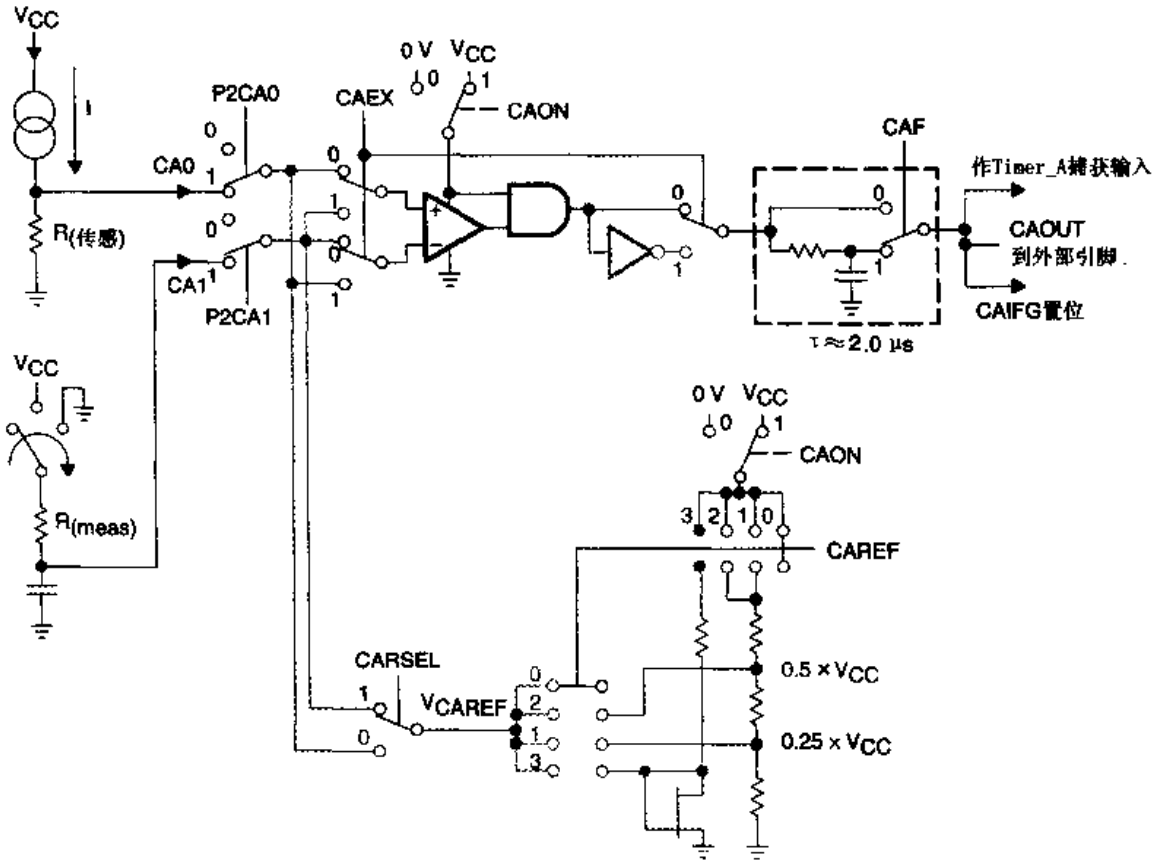


图 14.15　电流测量电路

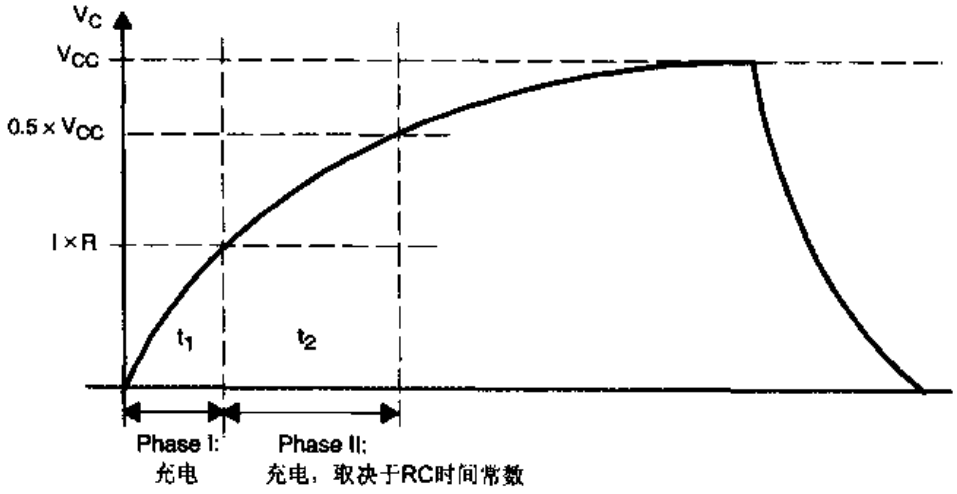


图 14.16　电流测量时序

14.4.6 测量比较器 A 的偏压

比较器的输入偏压对各个芯片都会不同，也会随温度、电源电压、输入电压而变。如果输入电压是稳定的参考电平，那么它对偏压的影响不明显。为了提高测量精度，比较器的偏压可以用以下步骤测量。用图 14.17～图 14.20 说明原理。

首先，用 CAEX＝0 执行一次转换。VCA0 加在比较器 A 的(＋)端。V_{ref}加在比较器 A 的(－)端，见图 14.17。

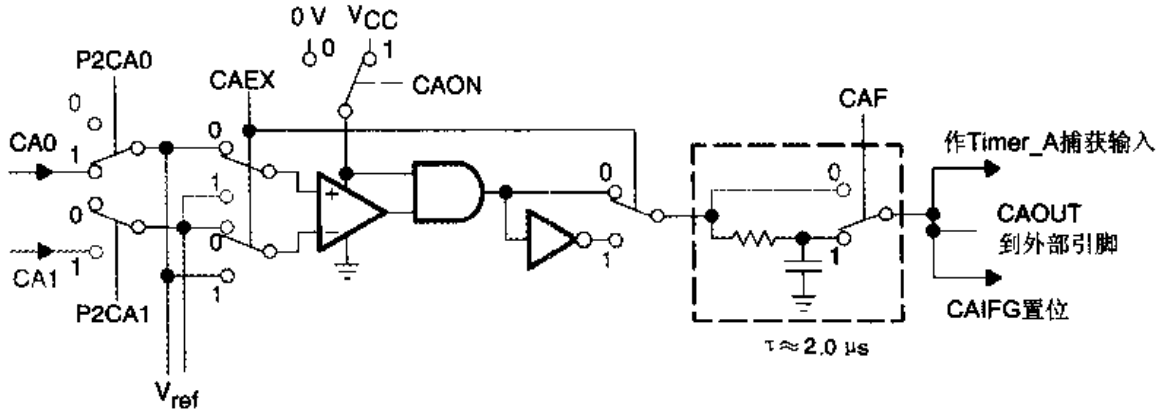


图 14.17 比较器偏压测量，CAEX＝0

在这种情况下，如图 14.18 偏压叠加在 V_{ref}上，有以下关系

$$V_{CA0} = V_{ref} + V_{offset}$$

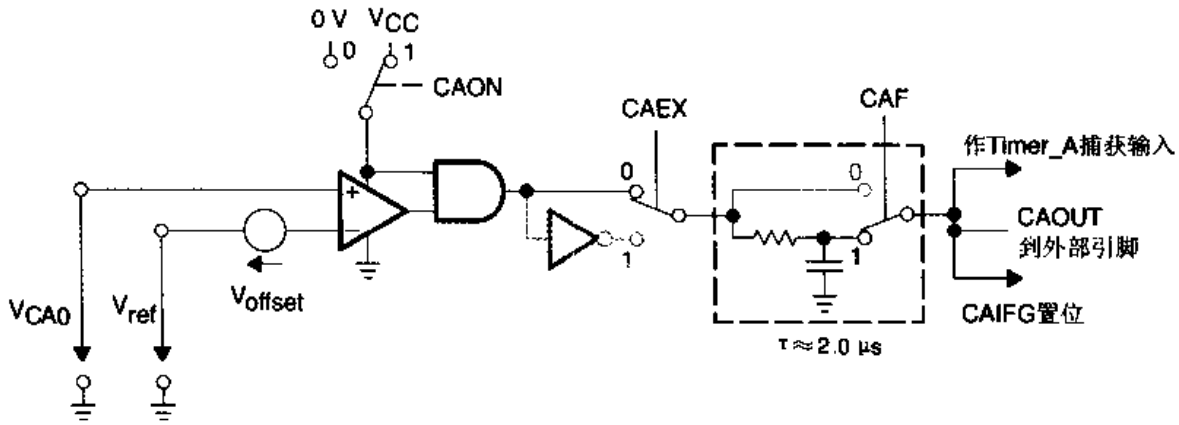


图 14.18 比较器偏压，CAEX＝0

然后，用 CAEX＝1 执行一次转换。V_{CA0}加在比较器 A 的(＋)端，V_{ref}加在比较器 A 的(－)端，见图 14.19。

在这种情况下，如图 14.20 偏压叠加在 V_{CA0}上，有以下关系，即

$$V_{ref} = V_{CA0} + V_{offset}$$

$$V_{CA0} = V_{ref} - V_{offset}$$

于是，V_{ref}可以用以下方程求得

$$N1 = -RV_{CA0} \times C \times \ln((V_{ref} + V_{offset}) / V_{CC}) \times f_{OSC}$$
$$N2 = -RV_{CA0} \times C \times \ln((V_{ref} - V_{offset}) / V_{CC}) \times f_{OSC}$$

解得

$$V_{offset} = V_{CC} \times e^{((N1+N2)/2N1 \times \ln(V_{ref}/V_{CC}))} - V_{ref}$$

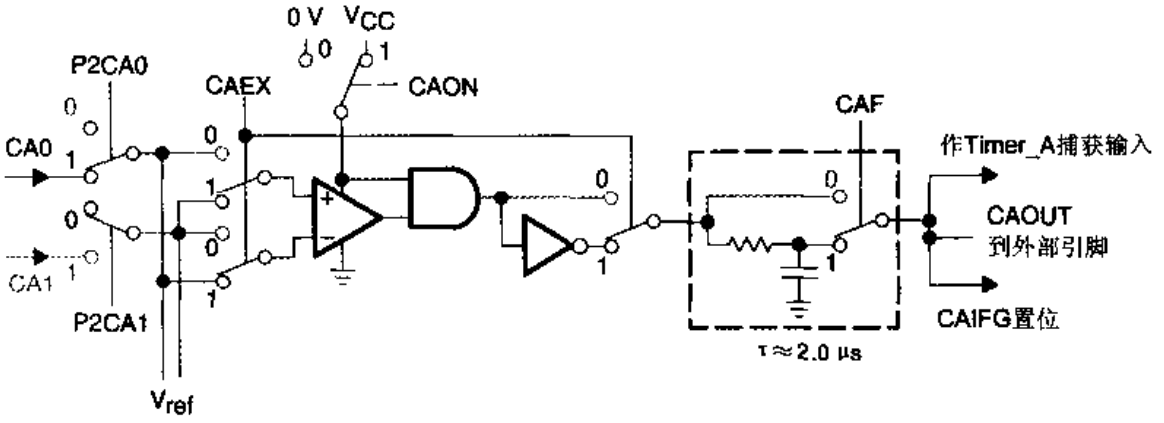


图 14.19　比较器偏压测量，CAEX=1

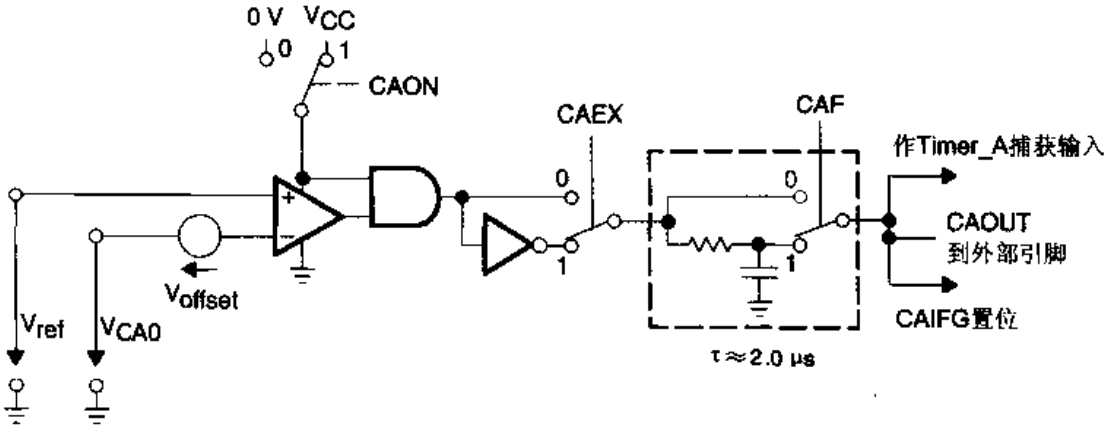


图 14.20　比较器偏压，CAEX=1

14.4.7　比较器 A 的偏压补偿

另一种改进测量精度的方法是补偿输入偏压的影响，而不是将它计算出来。

当 CAEX=0 时，V_{offset} 与 V_{ref} 叠加，有 $V_{CA0} = V_{ref} + V_{offset}$；

当 CAEX=1 时，V_{offset} 与 V_{CA0} 叠加，有 $V_{CA0} = V_{ref} - V_{offset}$。

将两次不同输入配置的转换结果相加再除以 2，就消除了偏压的影响。(N1+N2)/2 即为没有偏压影响的转换值。

14.4.8　增加比较器 A 的回差

当比较器(+)端及(−)端的电平接近时，比较器的输出会发生振荡。会引起两种情况：

- 由于比较器输出要不断驱动信号通道的充放电，增加了电流消耗。
- 产生多余的中断请求，或因查询 CAOUT、CAIFG 标志产生多余的服务请求。

图 14.21 说明了如何对比较器加入回差以避免比较器 A 输出振荡。

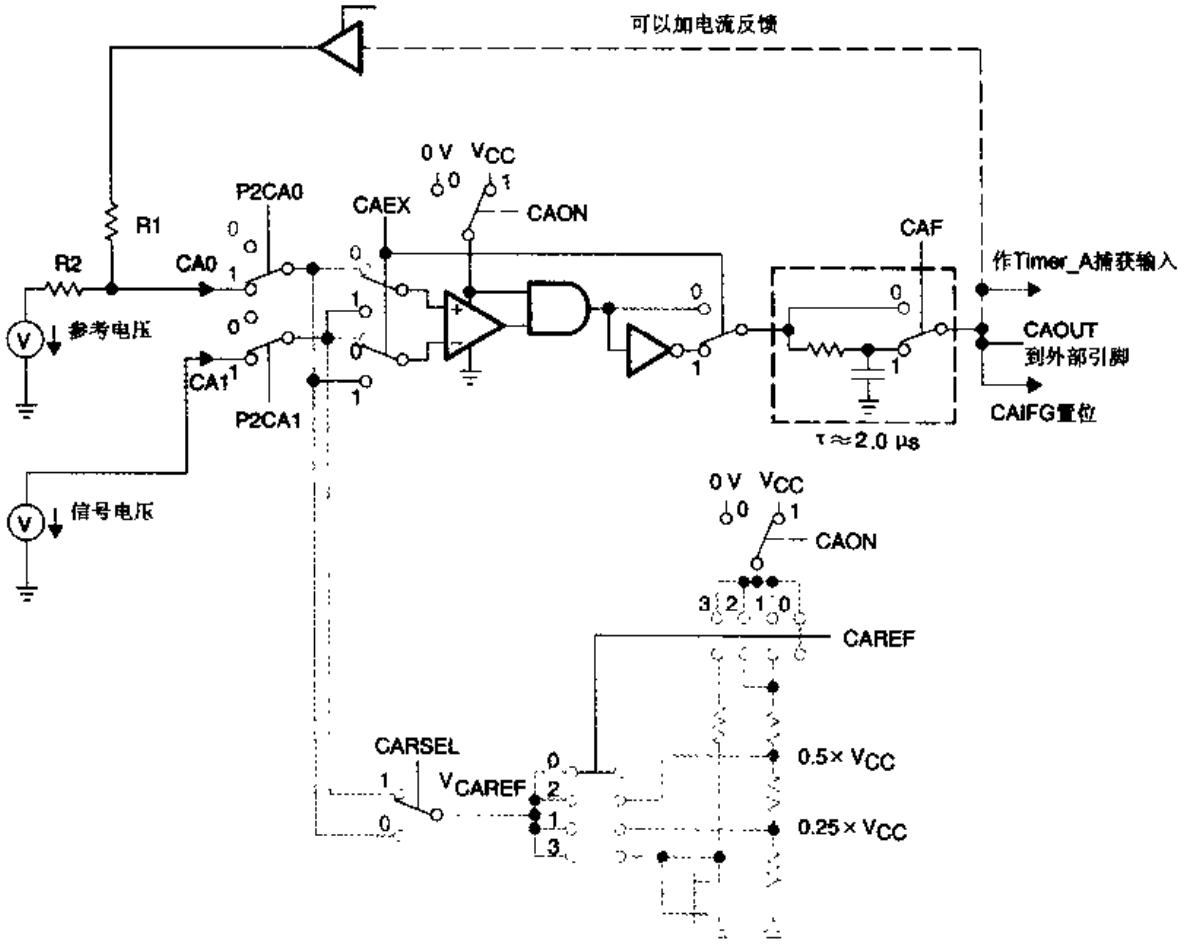


图 14.21 用比较器输出对参考电平加回差电压

加回差只有在 CAOUT 引出到片外时才可实现。请查阅芯片手册。

回差电压 V_h 可按下式计算，即

$$V_h = \pm R2 / (R1+R2) \times V_{CC}$$

第 15 章　模数转换器 ADC12

15.1　概　述

ADC12 是 12 位精度的 A/D 转换模块，具有高速、通用的特点。它在 MSP430F13x、MSP430F14x 芯片中实现。对于 MSP430F11x、MSP430F11x1 系列，A/D 转换的功能要用其他外围模块，例如比较器 A 来实现。

ADC12 的结构见图 15.1。它具有以下 5 大功能模块，都可以独立配置，即

- 带有采样/保持功能的 ADC 内核。
- 可控制的转换存储。
- 可控制的参考电平发生器。
- 可控制和选择的时钟源。
- 可控制的采样及转换时序电路。

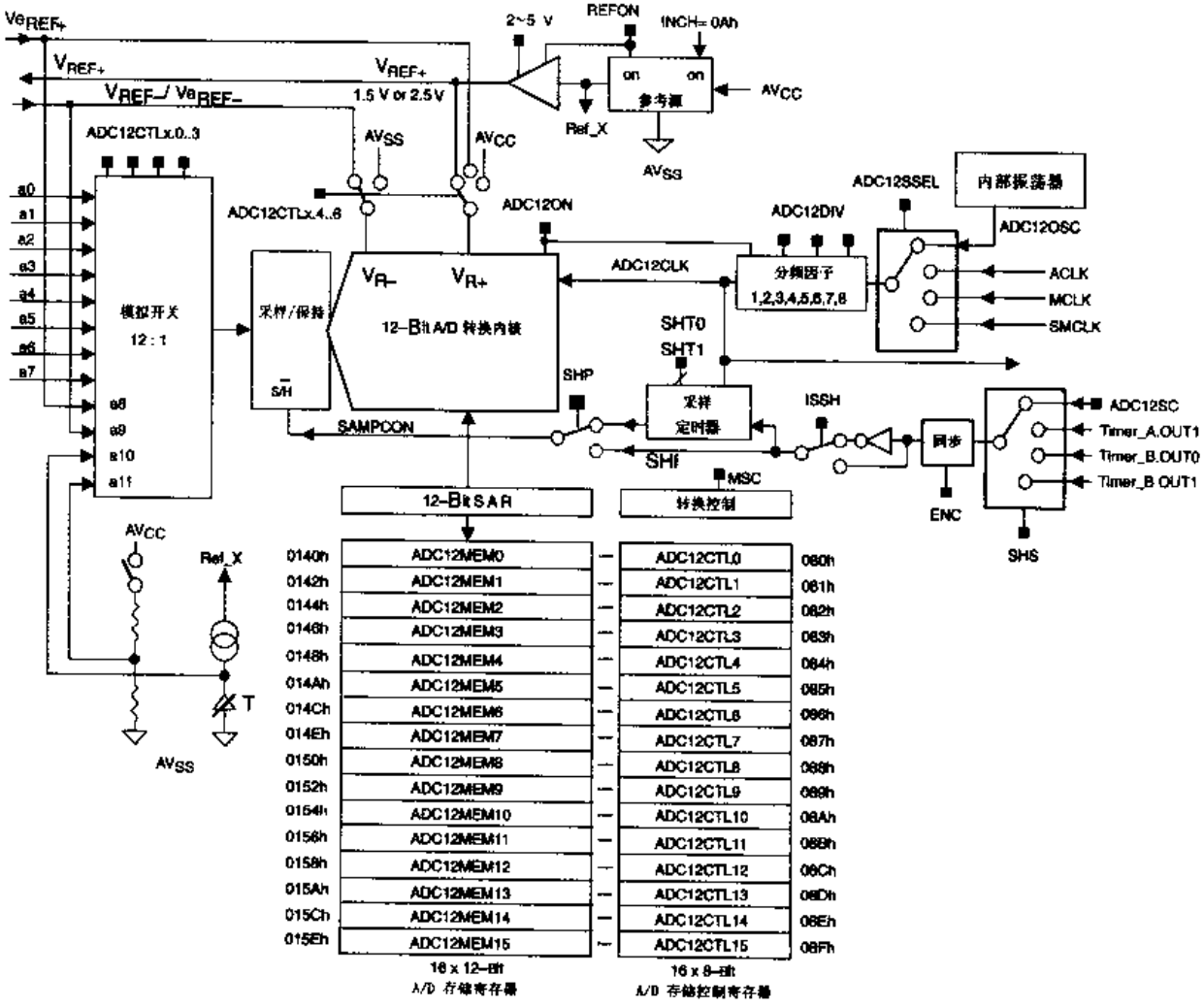


图 15.1　ADC12 结构

ADC12 可以对 8 个外部模拟信号之一或 4 个内部电压之一作转换。4 个内部通道用于温度测量、V_{CC}测量、正参考电平 Ve_{REF+} 测量及负参考电平 V_{REF-}/Ve_{REF-} 测量。

ADC12 工作时可以用内部参考电平，或者外部参考电平，也可以是两者的组合。

ADC12 具有通用的采样/保持电路，给用户提供了对采样时序的各种选择。采样时序可以通过控制位用软件直接控制，也可以用 3 种内部或外部的信号来控制。通常，内部时序信号来自于 MSP430 的定时器，例如 Timer_A。此外，采样时序也可以编程为 ADC12 转换时钟周期的倍数。

用户对 ADC12 的转换时钟，有多种选择来形成采样时序。ADC12 可以选择所有有效的 MSP430 片内时钟，也可以选择 一个外围模块所含的时钟，对于选择的时钟源可以引入一个 1～8的分频因子。

ADC12 有 4 种工作模式。可以在单通道上实现单次转换或多次转换，在序列通道上实现单次转换或重复转换。对于序列通道转换，采样顺序完全由用户来定义。例如，用户可以定义采样通道的顺序为：a1 — a3 — a1 — a6 — a2…，同时每个通道可以独立配置转换所需的参考电平。

转换结果保存在 16 个转换存储寄存器中。每个寄存器有各自相应的配置及控制寄存器，让用户为准备存储的转换结果选择采样通道和转换所需参考电平。

将 ADC12 的主要特性归纳如下：

- 最大采样速率为 200 ksps。
- 转换为 12 位精度，1 位差分非线性(DNL)，1 位积分非线性(INL)。
- 内装采样/保持电路，可选择软件、采样定时器或其他片内定时器来控制采样周期。
- 内装的 RC 振荡器，可以用于产生采样时序。
- 内装的用作温度测量的热敏二极管。
- 8 个可配置的外部信号采样通道。
- 4 个内部通道，用于温度、AV_{CC}及外部参考电平的采样。
- 内部的参考电平为 1.5 V 或 2.5 V，可用软件选择。
- 可以为每个通道的正负参考电平选择内部或外部的电压源。
- 可选择转换时钟源。
- 具有单通道单次、单通道重复、序列通道单次、序列通道重复等转换模式。
- 16 个保存转换结果的 12 位寄存器，可独立用软件访问及配置通道及参考电平。
- ADC 内核与参考电平发生器可分别进入省电模式。

15.2 ADC12 的工作原理及操作

15.2.1 ADC 内核

ADC 内核完成将模拟信号转换成 12 位数据并存入转换存储寄存器，见图 15.2。内核用到 2 个参考电平，即 V_{R+} 和 V_{R-}，作为转换范围的上下限和读数的满量程值和“0”值。转换数值在输入信号大于等于 V_{R+} 时为满量程值，小于等于 V_{R-} 时为“0”。输入通道和参考电平由转换存储控制寄存器定义。转换公式为

如果只用外部的参考电平，那么内部的参考电平发生器可以关闭，以降低功耗。

参考电平 V_{R+} 和 V_{R-} 建立了可以得到读数的模拟输入信号上下限，分别为满量程和“0”值。在实际使用中，参考电平和输入模拟信号不应高于供电电压和低于 V_{SS}，请查阅芯片手册。转换数值在输入信号大于等于 V_{R+} 时为满量程值，小于等于 V_{R-} 时为“0”。

注意：

当内部参考电平发生器用 V_{REFON} 开启后，在开始转换之前必须满足在芯片手册中说明的建立时间。否则，在参考电平建立完成前的转换结果会出错。一旦所有内部和外部的参考电平完成建立过程，此后在为每个通道选择和改变转换范围时就不再需要建立时间。

15.3 模拟输入与多路切换

15.3.1 模拟多路切换

ADC12 的 8 个外部模拟信号通道和 4 个内部信号经过多路模拟开关选择。通道由与每个转换存储寄存器对应的 ADC12MCTLx 寄存器选择。多路模拟开关是先关后开型的，见图 15.3，以减少因通道切换而引入的噪声。它是一个 T 型开关，为了尽量减少通道间的耦合，未选中的通道与 A/D 隔离，中间节点接模拟地(AV_{SS})，这样使得杂散电容接地以减少串扰。

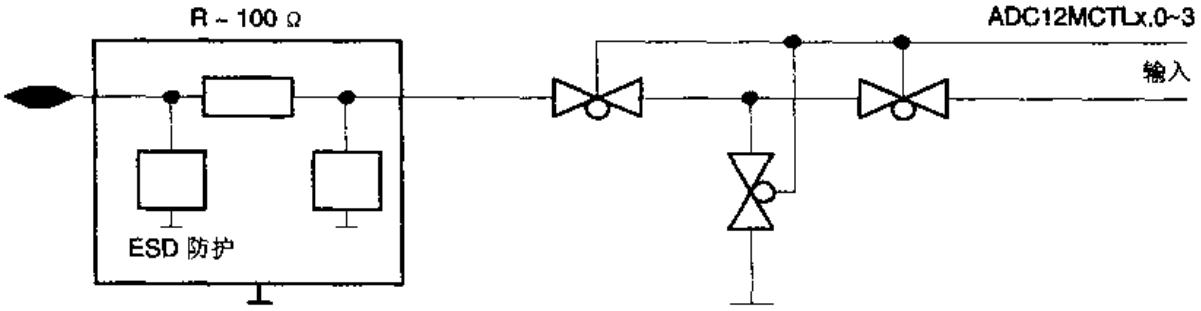


图 15.3 模拟开关结构

因为开关两端和开关之间存在寄生耦合电容，串扰就会发生。它可能有以下形式，如在一个断开的开关的输入和输出之间，或一个关闭通道的输入和相邻通道的输出之间。对于高精度的 A/D，串扰是必须消除的。可以用屏蔽和其他常见的 PCB 布线技术来消除。

15.3.2 输入信号

在采样过程中，输入信号加在 A/D 内核的电容网络上。因此，电容网络的充电是由信号源提供的。电容网络在采样期间必须充分充电，因此，外部信号源的电阻、动态阻抗和电容网络的容量必须与采样周期相匹配，以保证模拟信号的采样达到 12 位精度。

此外，信号源的内阻对于转换精度也有影响。因为漏电流或由输入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 12 位 A/D，因漏电流引入的误差可按下式计算，即

$$\text{误差(LSB)} = 4.096 \times \text{漏电流}(\mu A) \times \text{信号源内阻}(k\Omega)/(V_{R+} + V_{R-})$$

例如，一个 10 kΩ 内阻的信号源工作时有 50 nA 的漏电流，参考电平为 1.5 V，将产生 1.4 LSB的误差。

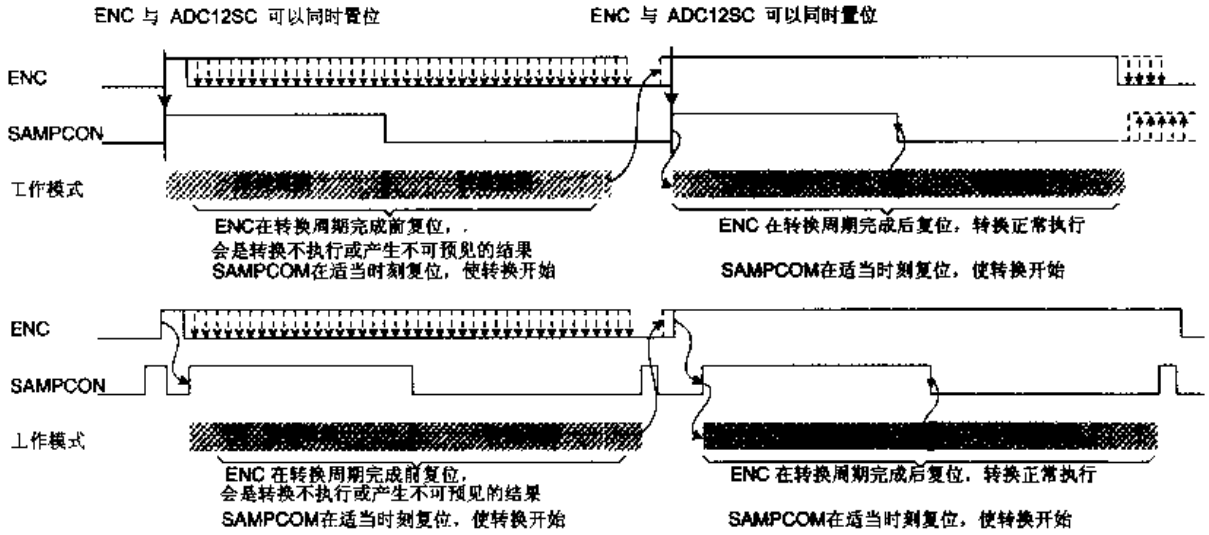
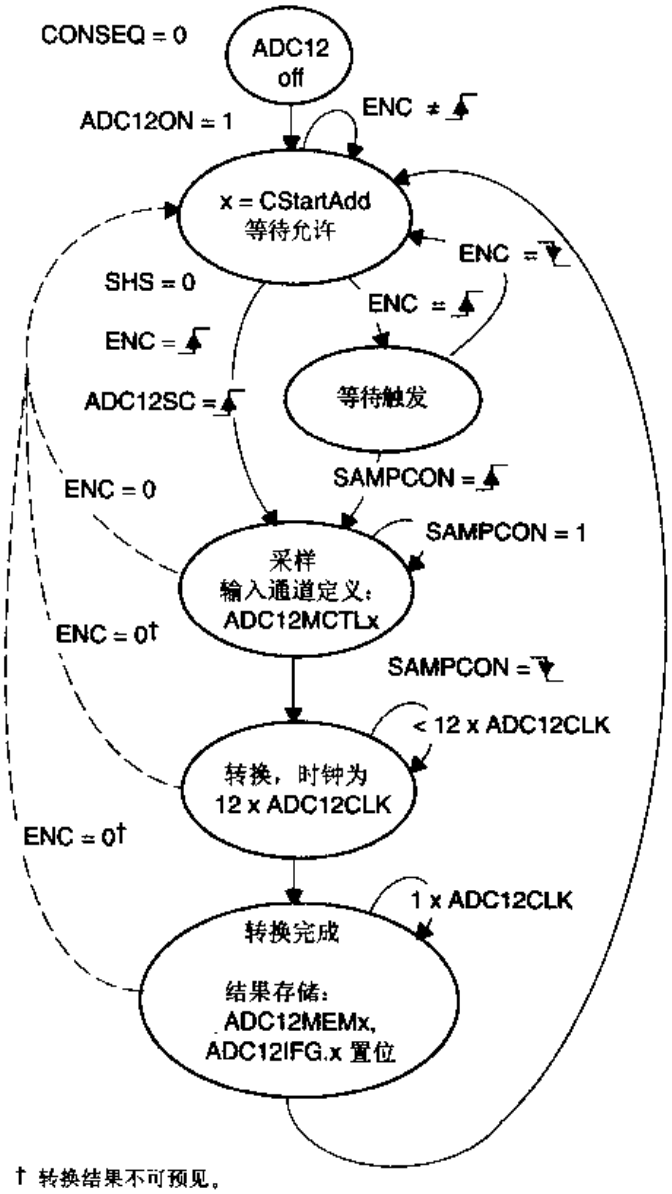


图 15.4　用 ENC 位停止 A/D 转换



† 转换结果不可预见。

图 15.5　单通道单次转换模式状态图

图 15.6 是单通道单次转换的转换存储设置的例子。

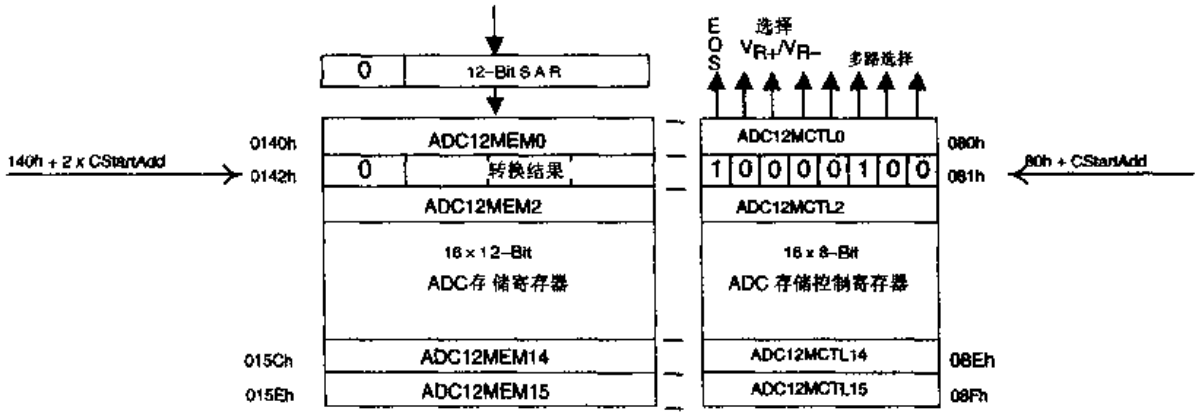


图 15.6 转换存储设置实例

例子应用于以下情况：

- 通道 a4 作单次转换。
- 用内部参考电平，V_{R+} 为 AV_{CC}，V_{R-} 为 AV_{CC}。
- 转换结果保存在转换存储寄存器 ADC12MEM1 中。

因此，ADC12CTL0 的 CStartAdd 位设为“1”，ADC12MCTL1 中的 INCH＝4，SREF＝0。

15.5.2 序列通道单次转换模式

序列通道单次转换模式对一个通道序列作一次转换。ADC12CTL1 中的 CStartAdd 位指向第一个转换存储寄存器。其后的转换将结果存放在顺序的转换存储寄存器中。例如，序列的长度为 3 次转换，CStartAdd 指向转换存储寄存器 4，于是，第一次转换的结果存于 ADC12MEM4，第二次转换的结果存放于 ADC12MEM5，第三次转换的结果存放于 ADC12MEM6。

执行序列转换时，每次转换选择的通道和参考电平由与序列转换所用存储寄存器相配合的转换存储控制寄存器定义。例如，一次序列转换用到 ADC12MEM3～ADC12MEM6，每次转换的通道和参考电平分别定义在 ADC12MCTL3～ADC12MCTL6 之中。

用最后一个转换存储控制寄存器的 EOS 位定义序列转换的结束。每一个转换存储控制寄存器都有 EOS 位，除了最后一个转换存储控制寄存器中的 EOS 位，其余的 EOS 位必须复位。例如，序列转换从 ADC12MEM7 开始，到 ADC12MEM12 结束，就必须将 ADC12MEM7～ADCMEM11的 EOS 复位，ADC12MEM12 的 EOS 置位。当转换到达序列的终点，转换结束。

为了再次执行同一序列转换，或是一个新的序列，ENC 位必须复位然后再置位。在 ENC 再次置位前到来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 15.7。

如果在序列转换已经开始又尚未结束且 ENC 保持为高时改变转换模式，则原序列仍正常完成。新转换模式在原序列完成后成为有效，但是新模式是单通道单次时例外。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，则原模式停止。原采样序列可能未完成，但是已完成的转换结果是有效的。

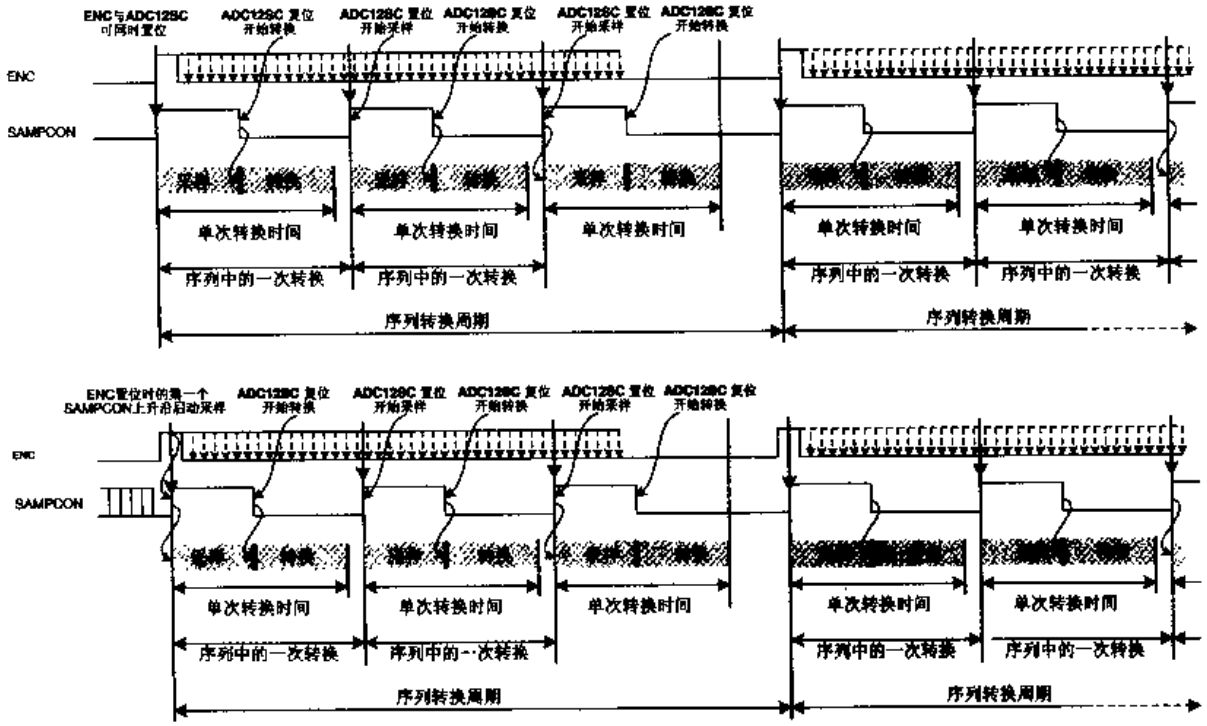


图 15.7　ENC 对序列转换的影响

如果在序列转换已经开始又尚未结束且 ENC 已翻转时改变转换模式，则原序列仍正常完成。新转换模式在原序列完成后成为有效。但是新模式是单通道单次时例外。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 ENC 复位时，则这三个条件中任何一个先发生都能使原模式停止。新模式在 ENC 再次置位时成为有效。

将 CONSEQ.1 位复位选择单通道单次转换，并且将 ENC 位复位，能使当前的序列转换模式立即停止。这时，转换存储寄存器 ADC12MEMx 中的数据不可预料，中断标志 ADC12IFG.x 可能置位也可能未置位。因此，这种处理方法通常是应该避免的，但是在发生紧急情况时仍然可能用到。

每次转换完成，转换结果存入相应的 ADC12MEMx 寄存器，相应的中断标志 ADC12IFG.x 置位来指示转换完成。这时如果相应的中断允许位置位，将发生中断服务请求。图 15.8 是序列通道转换的状态图。

图 15.9 和图 15.10 是序列转换的一个例子。其中，转换序列为 a0、a5、a7、a0、a0、a3，用 ADC12MEM6 存放第一个转换结果。转换序列如下：

- a0，参考电平 V_{R+} 为 AV_{CC}，V_{R-} 为 AV_{SS}。
- a5，参考电平 V_{R+} 为 V_{REF+}，V_{R-} 为 AV_{SS}。
- a7，参考电平 V_{R+} 为 V_{REF+}，V_{R-} 为 Ve_{REF-}/V_{REF-}。
- a0，参考电平 V_{R+} 为 AV_{CC}，V_{R-} 为 AV_{SS}。
- a0，参考电平 V_{R+} 为 AV_{CC}，V_{R-} 为 AV_{SS}。
- a3，参考电平 V_{R+} 为 V_{REF+}，V_{R-} 为 Ve_{REF-}/V_{REF} 。

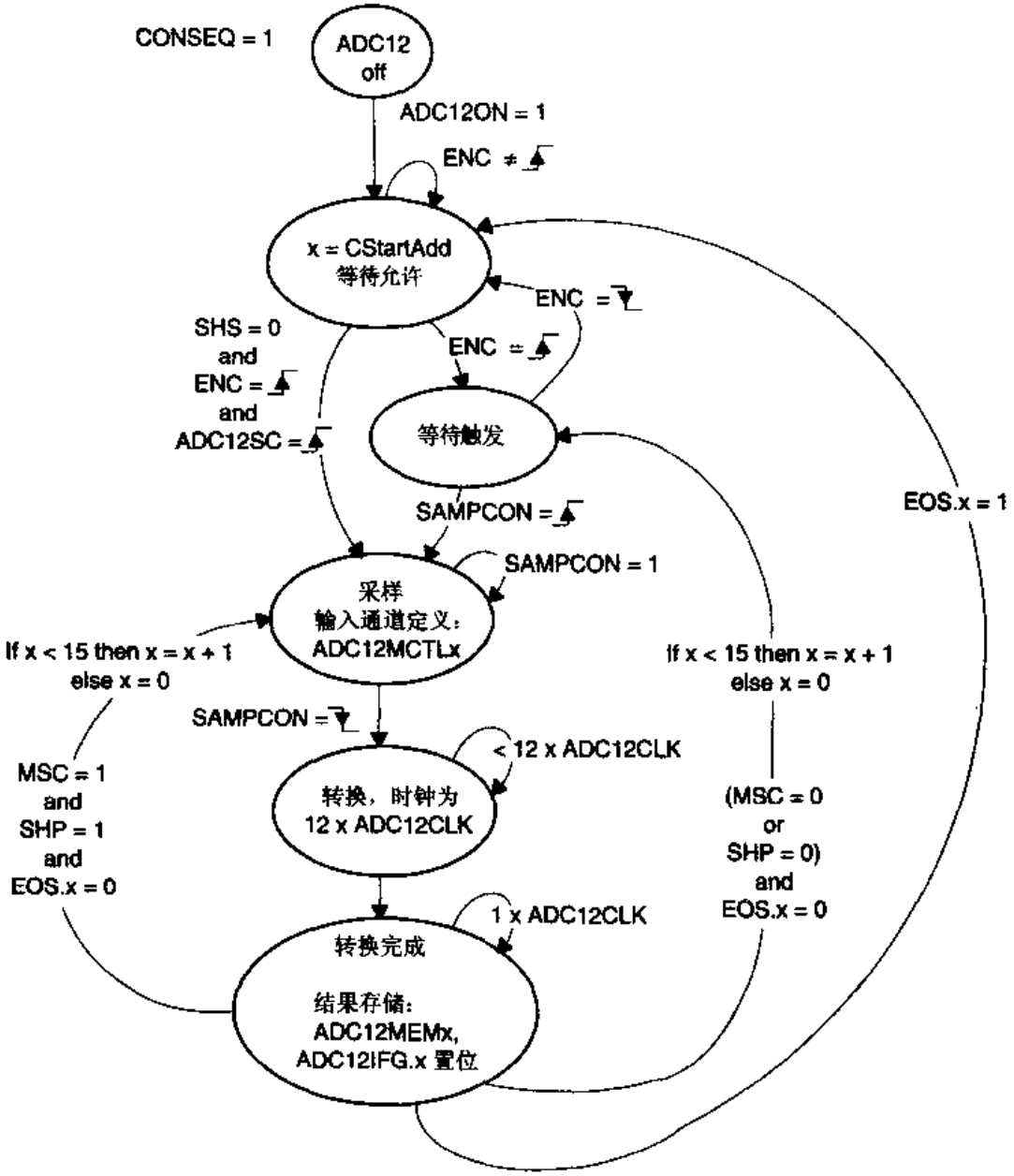


图 15.8 序列通道转换状态图

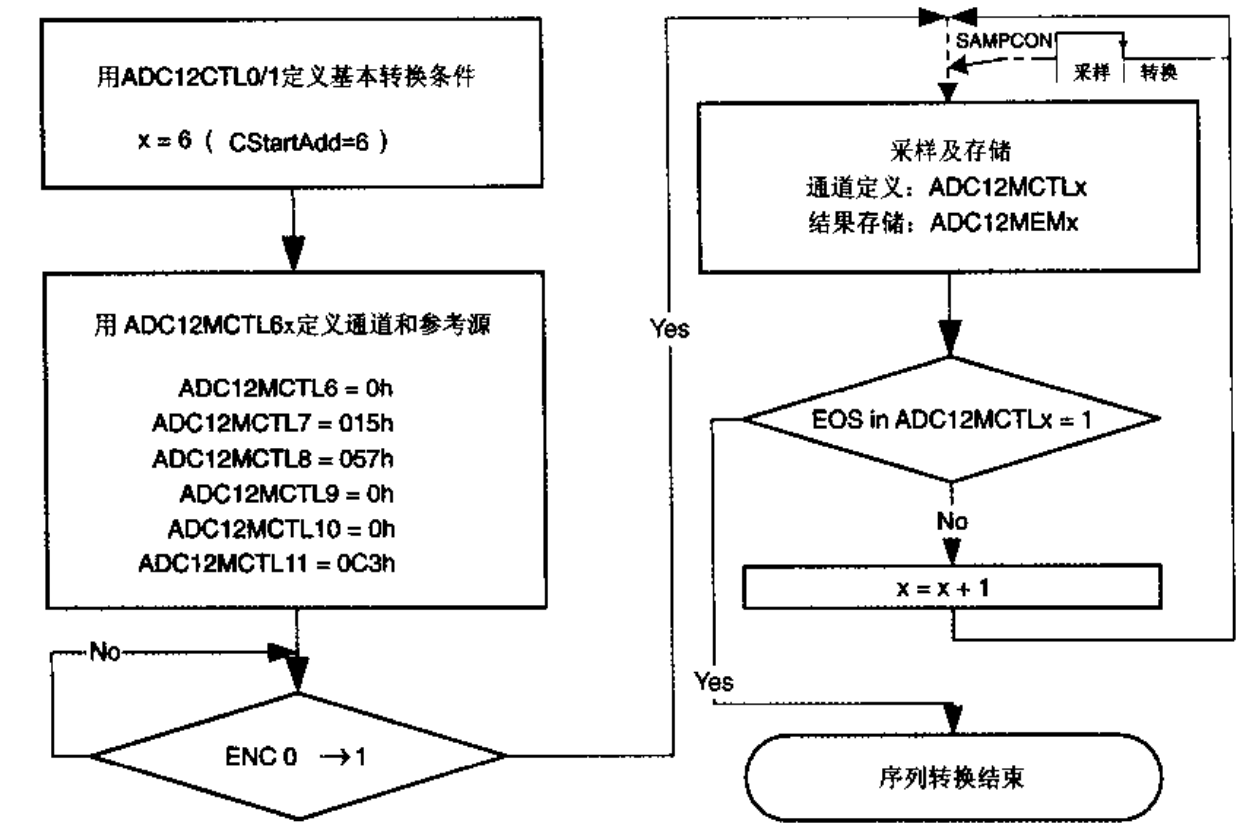


图 15.9　序列转换模式的流程

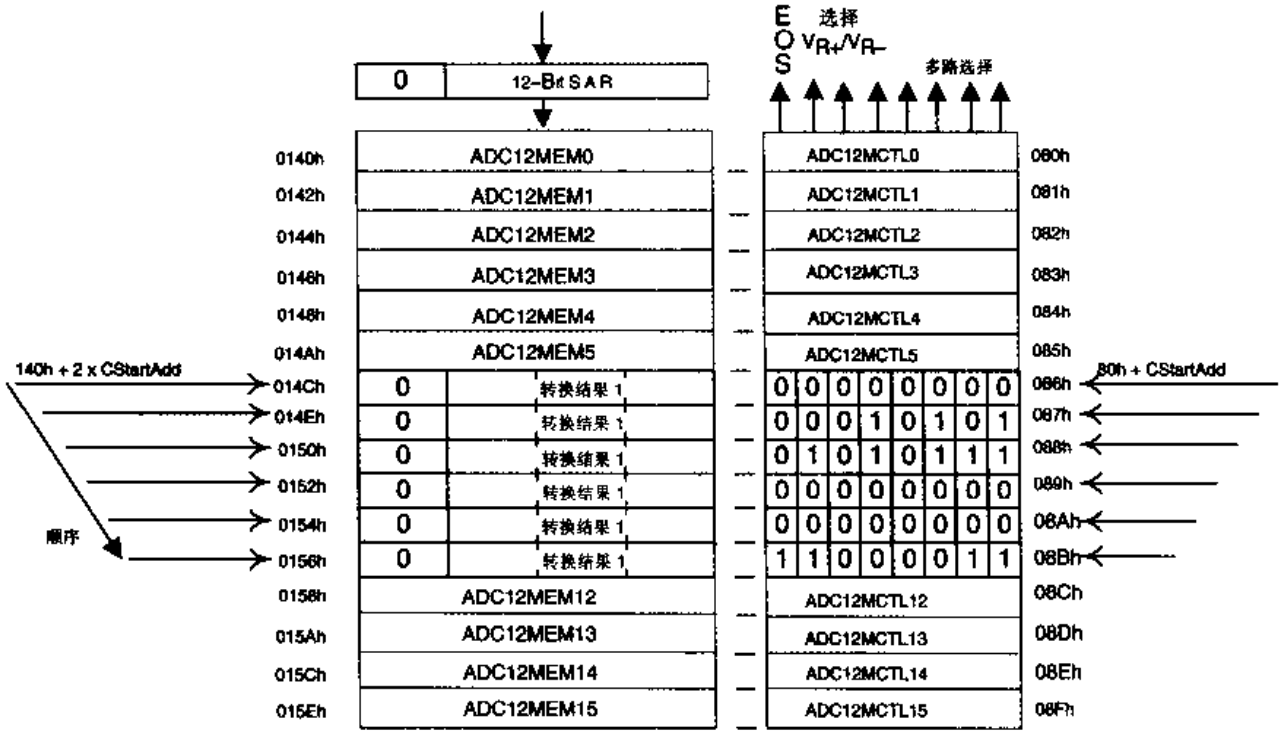


图 15.10　序列转换实例

15.5.3　单通道重复转换模式

单通道重复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发生，直

至用软件将转换停止。每次转换结束，转换结果存入相应的 ADC12MEMx，由相应的中断标志 ADC12IFG. x 置位指示转换的结束。如果这时允许中断，将产生中断服务请求。

改变转换模式不必先停止转换。一旦改变了模式，它会在当前序列完成后成为有效。

有 3 种方法来停止单通道重复转换模式，即

- 用 CONSEQ=0 选择单通道单次转换模式代替单通道重复转换模式。当前序列会正常结束，相应的结果存入 ADC12MEMx，中断标志 ADC12IFG. x 置位。
- 用 ENC 复位使当前序列完成后停止。相应的结果存入 ADC12MEMx，中断标志 ADC12IFG. x 置位。
- 用单通道单次转换模式代替单通道重复转换模式，然后将 ENC 复位。当前序列立即停止，相应的 ADC12MEMx 中的数据是不可预见的，中断标志 ADC12IFG. x 可能置位也可能未置位。一般不建议采用这种方法。

图 15.11 是单通道重复转换模式的状态图。

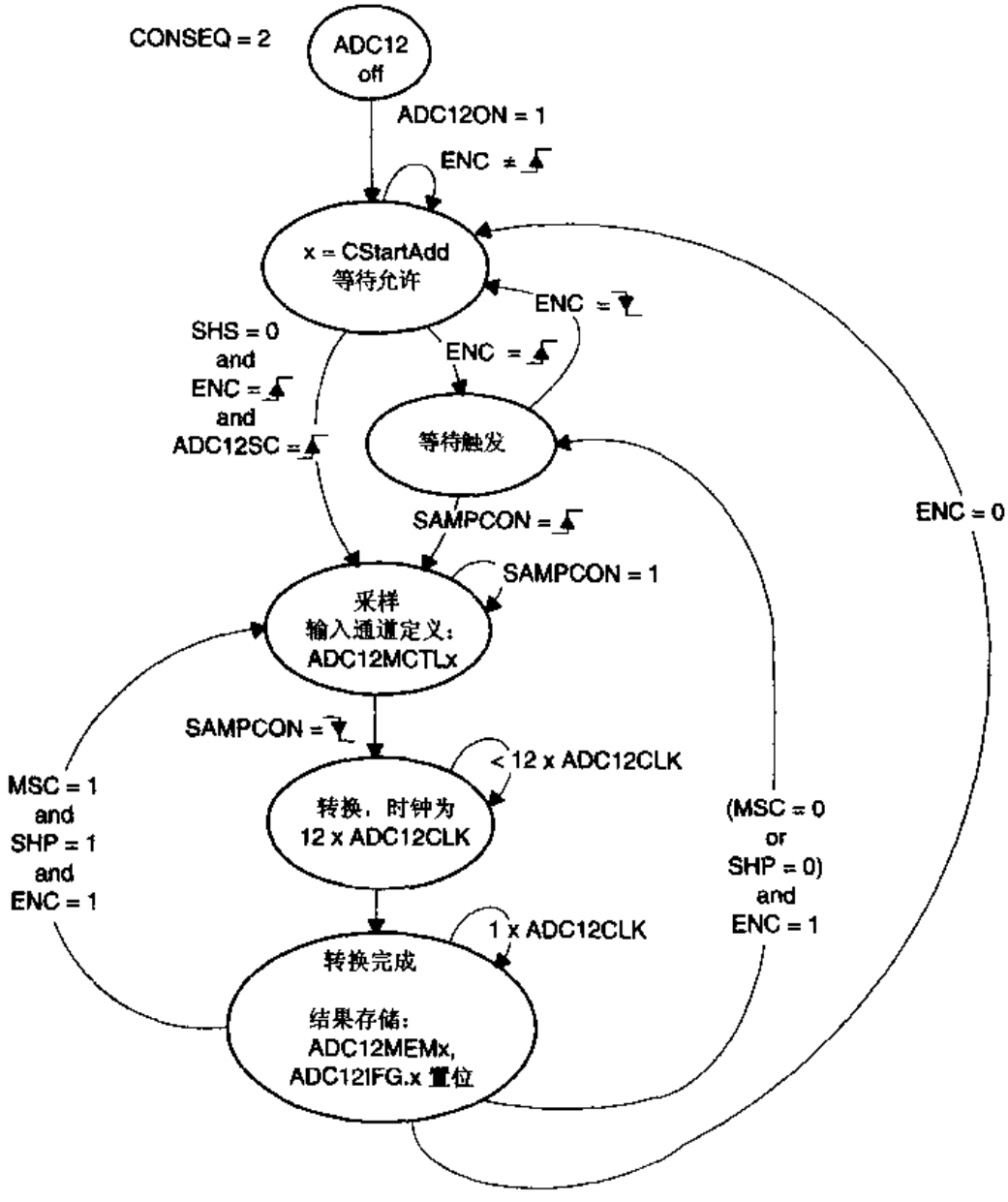


图 15.11 单通道重复转换模式状态图

结束后不再转换，相应结果存入 ADC12MEMx，中断标志 ADC12IFG.x 置位。

- 用 ENC 复位使当前序列完成后停止。相应的结果存入 ADC12MEMx，中断标志 ADC12IFG.x 置位。
- 用 CONSEQ=2 选择序列通道单次转换模式代替序列通道重复转换模式。然后选择单通道重复转换模式。当前序列正常结束，相应的结果存入 ADC12MEMx，中断标志 ADC12IFG.x 置位。x 的值在 CStartAdd 与序列末寄存器号之间。
- 用 CONSEQ=0 选择单通道单次转换模式代替序列通道重复转换模式，然后将 ENC 复位。当前序列立即停止，相应的 ADC12MEMx 中的数据是不可预见的，中断标志 ADC12IFG.x 可能置位也可能未置位。一般不建议采用这种方法。

15.5.5 转换模式之间的切换

当转换并未发生时切换转换模式很简单，只要用 CONSEQ 位定义新模式即可。但是当转换正在进行过程中改变模式，而 CONSEQ 的两位在一条指令中同时改变，可能意外地选中一个过渡模式。因此，以下的模式切换过程应避免：0→3、1→2、2→1 和 3→0。这种过渡模式的选中是因为 CPU 和 ADC12 的时钟异步而引起的。过渡模式的避免可以简单地用在一条指令中只改变一个 CONSEQ 位来避免。例如，在转换正在进行中要将模式 0 切换成模式 3，应该用如下的指令序列来完成，即

```
BIS     #CONSEQ_0,&ADC12CTL1     ; 先将模式 0 切换成模式 1
BIS     #CONSEQ_1,&ADC12CTL1     ; 然后将模式 1 切换成模式 3
```

可以接受的切换过程为：0→1、0→2、1→0、1→3、2→0、2→3、3→1 和 3→2。

15.5.6 低功耗

ADC12 有 2 位与省电模式有关的控制位：ADC12ON 和 REFON。ADC12ON 打开 A/D 内核，REFON 打开参考电平发生器。它们可以用软件分别控制。当这 2 位全复位时，A/D 转换就完全关闭了。ADC12 的寄存器不受这两个控制位的影响，它们可以在任何时间访问及修改，但是 ADC12ON 和 REFON 只能在 ENC 复位时修改。

为了进一步地降低功耗，即使 ADC12 是在运行中，它的功能是自动打开和关闭的。

注意：ADC12 的低功耗控制

转换正在进行时，不能关闭 A/D 内核及参考电平发生器，否则转换结果会出错。

可以在转换开始之前或序列转换结束之后，将 ADC12ON 及 REFON 复位来关闭 A/D 内核及参考电平发生器。例如，ADC12 处于序列转换模式，在序列转换开始后用软件将 ENC 复位，这样就可以在序列转换完成前，用 ADC12ON 和 REFON 复位使 ADC12 进入省电模式。一旦实行，ADC12 立即进入低功耗状态，但是转换数据可能是错的。

注意：ADC12 控制时间

ADC12 用 ADC12ON 打开需要一定的时间，请查阅芯片手册。从打开 ADC12 到开始转换必须满足这一时间要求，否则转换结果可能出错。

内部参考电平用 REFON 打开需要一定的时间，请查阅芯片手册。从建立内部参考电平到开始转换必须满足这一时间要求，否则转换结果可能出错。但是一旦内部及外部的参考电平都已建立，对每个通道在选择和改变量程时不再需要额外的参考电平建立时间。

打开 ADC12 后，在进行第一次转换之前，外部信号应该完成建立，否则转换结果可能出错。

15.6　转换时钟与转换速度

ADC12 的转换时钟(ADC12CLK)可选自多个时钟源，并可作 1～8 分频，见图 15.13。它用于 A/D 转换，并产生采样周期。时钟源可以选自内部振荡器 ADC12OSC 或 ACLK、MCLK、SMCLK。

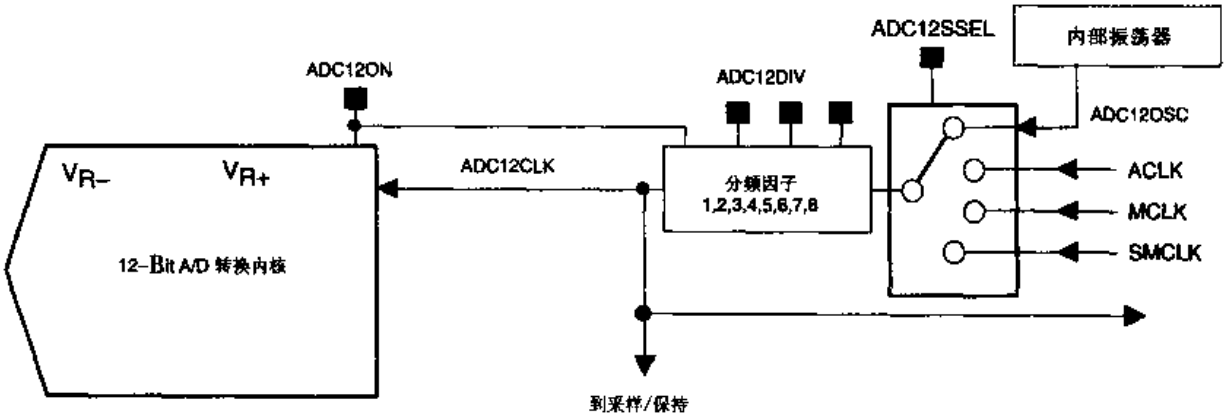


图 15.13　转换时钟 ADC12CLK

内部振荡器产生约 5 MHz 的 ADC12OSC 信号，参数请查阅芯片手册。振荡频率会随芯片、电源电压、温度等因素改变。当需要精确的转换时序时，应选择稳定的时钟源。

A/D 转换从采样信号 SAMPCON 的下降沿开始。整个转换过程需要 13 个时钟周期。转换时间用下式计算，即

$$\text{转换时间} = 13 \times (\text{ADC12DIV} / f_{\text{ADC12CLK}})$$

其中：ADC12DIV 是 1～8 的整数。

ADC12CLK 的频率的上下限必须不超过芯片手册中说明的参数，否则会使转换出错。

注意：转换过程中的 ADC12CLK

在 ADC12 完成操作过程中，ADC12CLK 必须保持有效。在 ADC12 活动时取消这一时钟信号，ADC12 的操作就无法完成，用转换结束来返回主程序也就不可能了。

15.7 采 样

ADC12 的采样/保持电路可由用户通过 ADC12CTL0 和 ADC12CTL1 中的控制位配置来灵活地实现控制，见图 15.14。

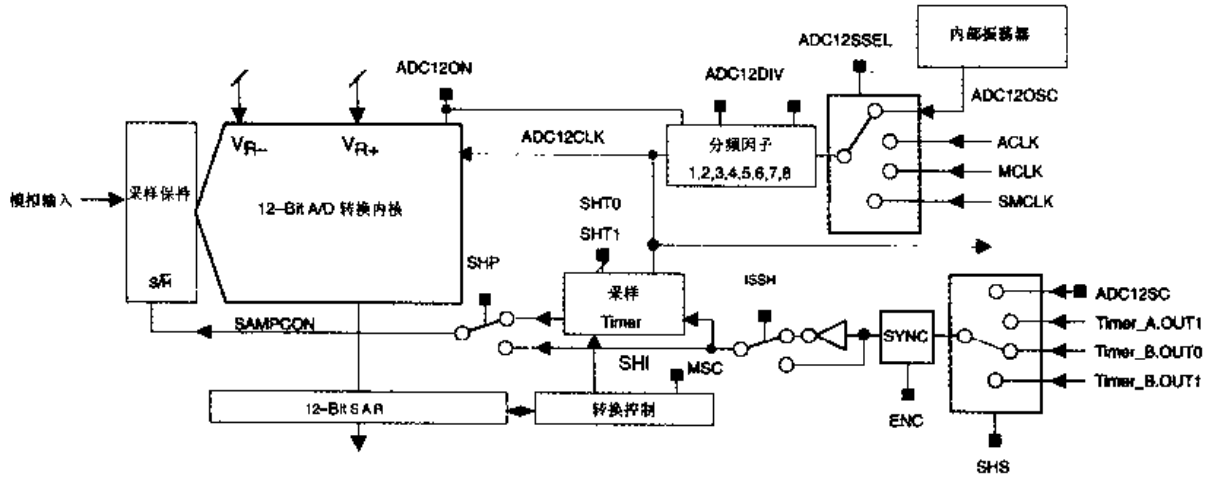


图 15.14 ADC12 中的采样/保持电路

15.7.1 采样操作

当采样信号 SANPCON 为高时，采样/保持电路对模拟信号采样。当 SAMPCON 出现下降沿时，转换立即发生。当 SAMPCON 为低时，采样/保持电路保持信号值。转换需要 13 个时钟周期，见图 15.15。

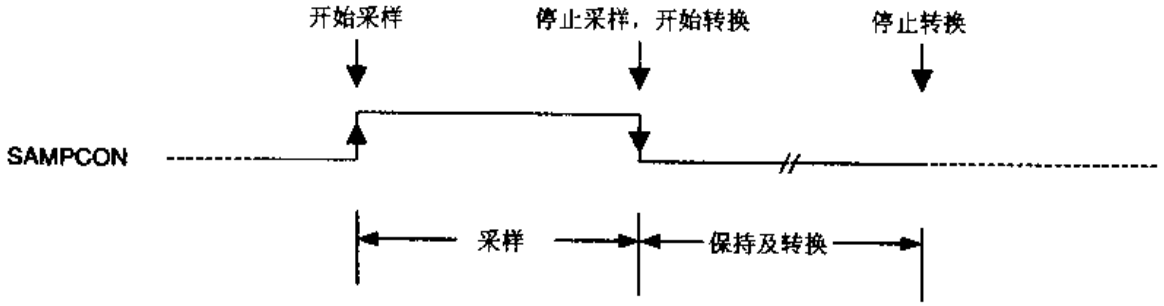


图 15.15 采样及转换时序

为了得到精确的转换，在采样期间，输入模拟信号应保持稳定有效。同时也希望在整个转换过程中，相邻的通道不出现数字活动，以保证电源毛刺、地电平波动、交互串扰等不会影响转换结果。

此外，因内部电荷的增加和损耗也限制了保持时间。用户程序应遵循芯片手册的参数，否则会因为采样信号的涨落而产生错误的转换结果。

15.7.2 采样信号输入选择

控制采样时序及转换开始的 SAMPCON 信号可以来自几种信号。SMAPCON 可来自采样信号输入(sample－signal input)或采样定时器。采样信号输入在发生输入选择时有效，见

图 15.16。当采样定时器作为 SAMPCON 的源时，采样信号输入可用于触发采样定时器。

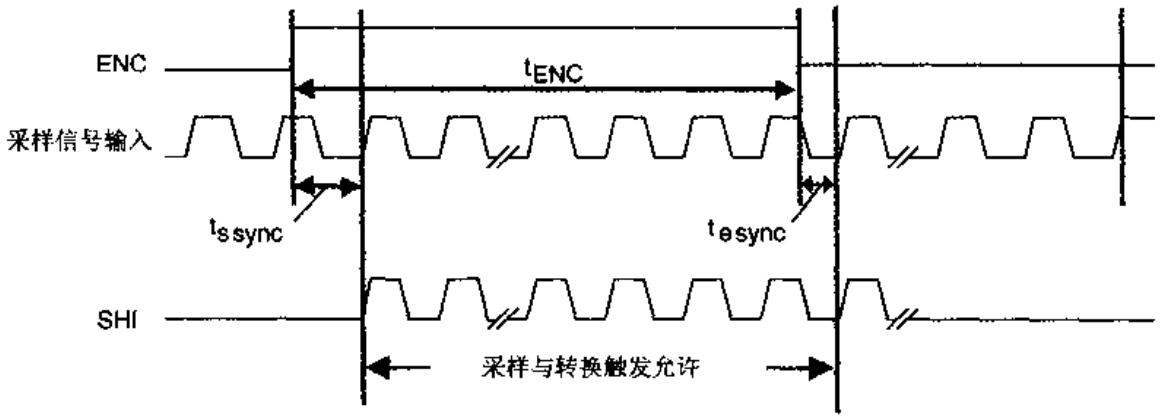


图 15.16　采样及转换信号的同步

采样信号输入用 ADC12CTL1 中的 SHS 位来选择。有 4 种选择，即：ADC12SC、Timer_A.OUT1、Timer_B.OUT0 和 Timer_B.OUT1。采样信号输入的极性用 ISSH 位来选择。采样信号输入也可以在 ENC 位控制下直接送到采样定时器或 SAMPCON。

ADC12SC 是 ADC12CTL0 中的控制位。它的值由软件设置。根据选择的采样模式，它允许软件或者启动一次采样转换周期(SHP=1)，或者完全控制采样周期(SHP=0)。

采样信号输入可能与转换允许信号异步，它可以用 ENC 位来同步及允许。如果不同步，则在 ENC 置位后的首次转换可能出错，取决于 ENC 置位在输入信号周期中发生的时刻。如图 15.18 的例子中，ENC 位在采样信号输入脉冲高电平的中间置位。如果采样信号输入只是简单地直接用作 S/H 信号，则由于采样周期太短首次采样可能出错。

为了避免这一问题。在采样输入开关中加入了同步控制逻辑。它确保首次采样转换周期发生在 ENC 置位后的采样信号的第一个上升沿。并且，末次采样转换周期发生在 ENC 复位后的采样信号的第一个上升沿。

15.7.3　采样模式

采样电路可以工作在 2 种模式，即：脉冲采样模式(pulse - sampling)和扩展采样模式(extended - sampling)。

脉冲采样模式用采样信号输入触发内部的采样定时器，采样定时信号(SAMPCON)由采样定时器产生，周期为 ADC12CLK 周期的整倍数。

扩展采样模式中的采样信号输入不经过采样定时器，直接用作 SAMPCON 信号的源，完全控制采样时序，而与 ADC12CLK 异步。但是转换过程需要 13 个 ADC12CLK 周期仍然不变。

脉冲采样模式

脉冲采样模式中，采样信号输入的上升沿触发采样定时器。然后采样定时器发生采样时序信号。采样时间由 ADC12CTL0 中的 SHT0 或 SHT1 编程。当用转换存储寄存器 ADC12MEM0～ADC12MEM7 保存结果时用 SHT0；用转换存储寄存器 ADC12MEM8～ADC12MEM15 保存结果时用 SHT1。因此当一个采样序列同时用了这两部分的转换存储寄存器，可以有两种不同的采样时间。当外部信号源有不同的内阻，需要不同的采样时序信号时，这一点很有用。

脉冲采样模式的采样时间用下式计算，即

$$\text{采样时间} = 4 \times \text{ADC12CLK 周期} \times \text{SHTx}$$

其中：SHTx 为 SHT0 或 SHT1。

如图 15.17 所示，采样信号 SAMPCON 在同步时间（t_{sync}）和采样时间（t_{sample}）保持为高。转换时间（$t_{convert}$）为 13 个 ADC12CLK 周期。采样/转换周期由采样信号输入触发，在整个周期完成之前，采样信号输入的上升沿不会再起作用。

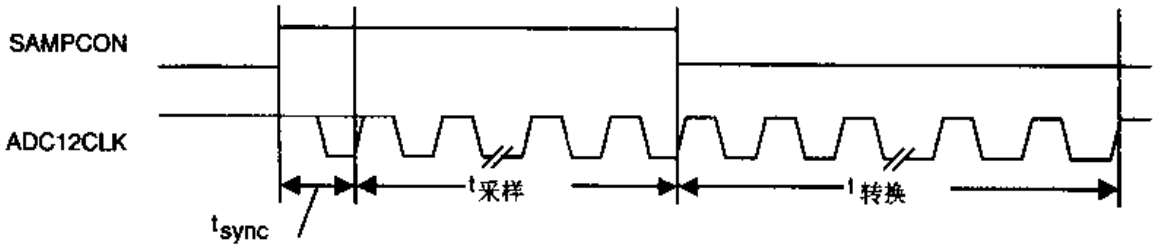


图 15.17 脉冲采样模式时序

图 15.18 是脉冲采样模式的例子，选择的输入信号源是 Timer_B. OUT0。图 15.19 是它的时序。

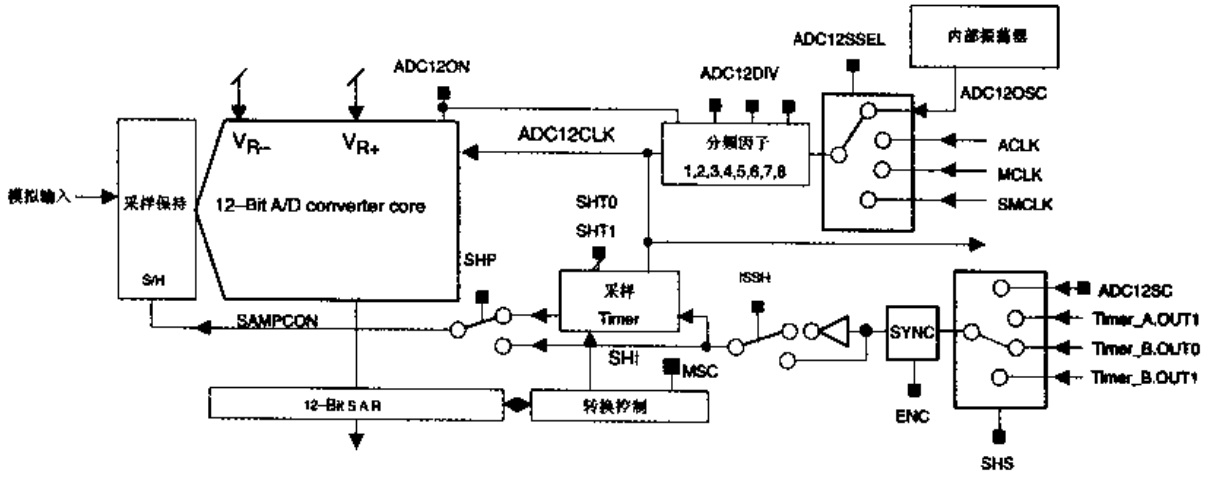


图 15.18 脉冲采样模式结构

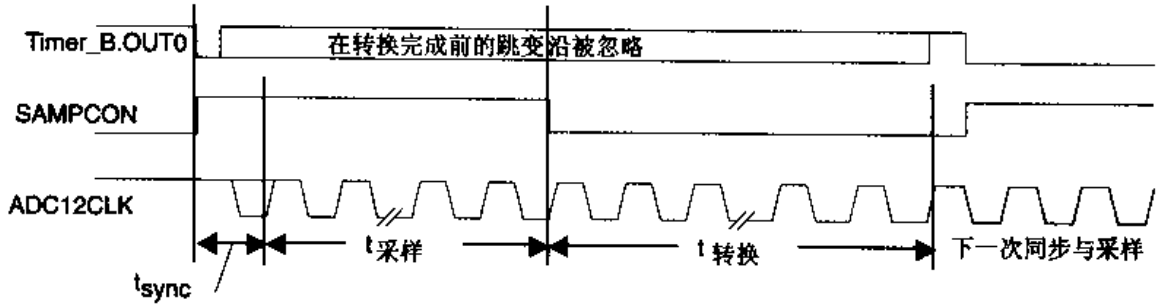


图 15.19 脉冲采样模式时序

扩展采样模式

扩展采样模式用输入信号直接控制采样信号 SAMPCON，不再用内部采样定时器。当

SAMPCON 为高时进行采样，为低时进行保持，见图 15.20。转换在 SAMPCON 信号的下降沿经过同步时间(t_{sync})后开始，转换时间仍然是 13 个 ADC12CLK 周期。

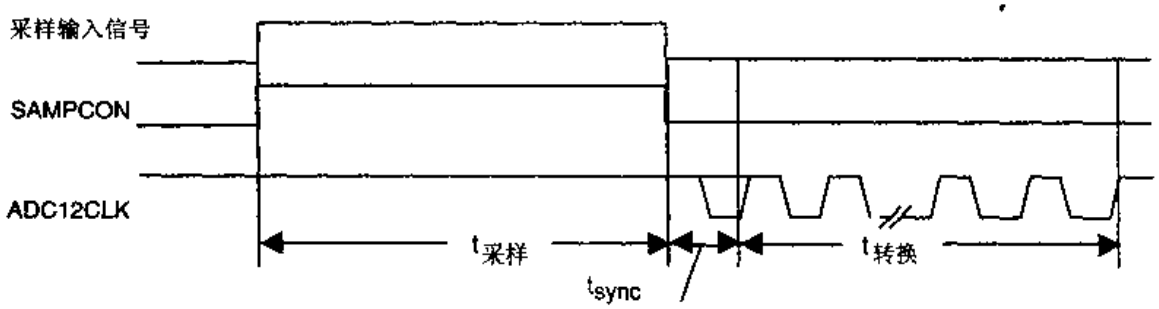


图 15.20　扩展采样模式的时序

扩展采样模式可以完全控制采样周期和转换开始时间。有时对于不同的输入信号源，因为内阻不同需要以延长的采样周期去适应，有时用内部采样定时器产生一个长的采样周期显得效率太低。这时，扩展采样模式很有用。

图 15.21 是扩展采样模式的例子，选择 Timer_B. OUT0 为输入信号，图 15.22 是它的时序。

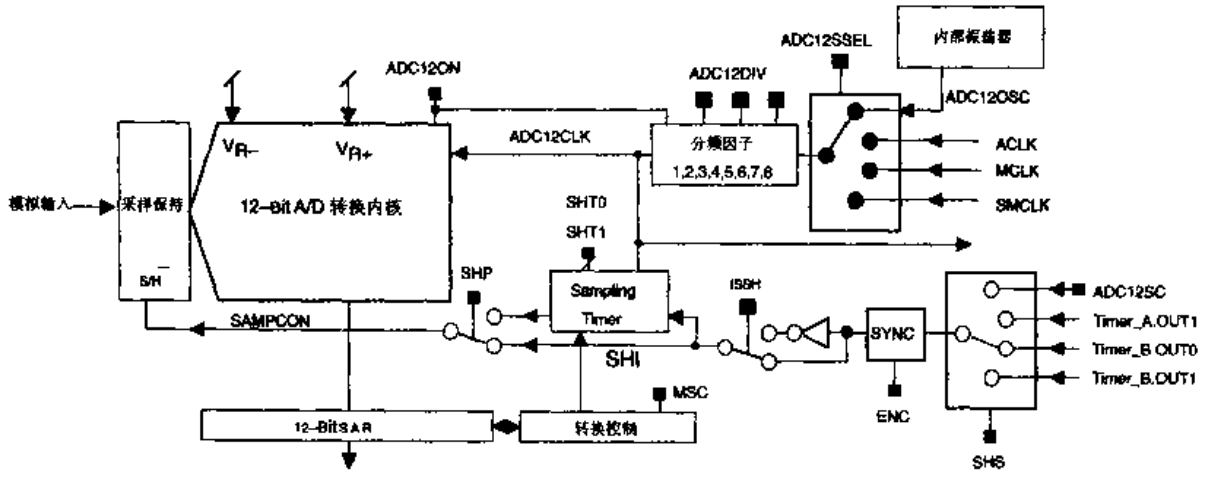


图 15.21　扩展采样模式结构

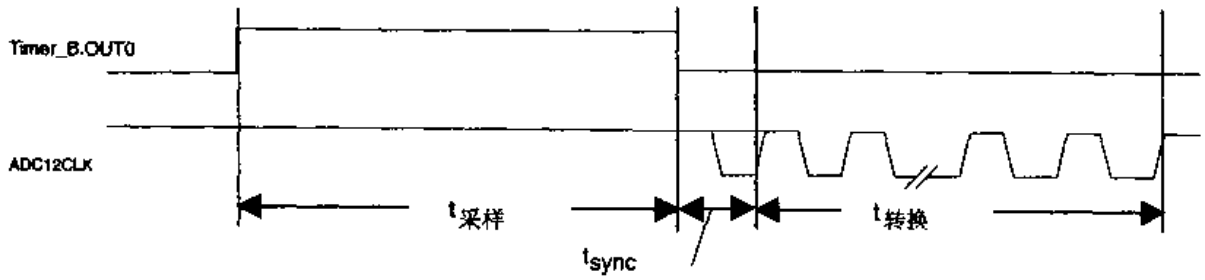


图 15.22　扩展采样模式时序

15.7.4　MSC 位的使用

多次采样/转换控制位(MSC)在不用采样定时器产生 SAMPCON 信号时不起作用。如果用采样定时器产生 SAMPCON 且不处于单通道单次转换模式时(CONSEQ＞0)，则 MSC

位可以控制实现尽可能快的连续转换。

如果 MSC=0,那么不管在哪种模式下,每次转换都需要由一个 SHI 的上升沿来触发。如果 MSC=1,并且 CONSEQ>0,那么首次转换由 SHI 的第一个上升沿触发,但是后续的转换会在每次转换完成后立即自动触发启动。根据不同的工作模式,在序列转换完成成 ENC 位翻转前,SHI 的后续上升沿不起作用。在使用 MSC 位时,ENC 位的作用不变,见图 15.23 和图 15.24。

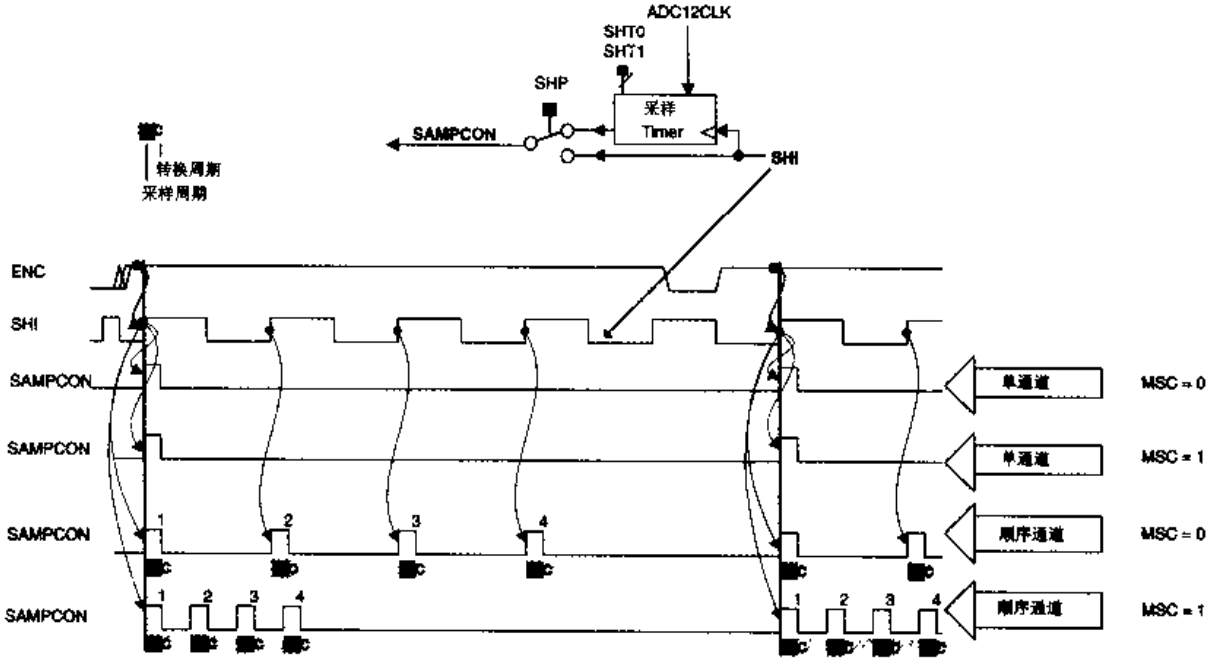


图 15.23 非重复模式下的 MSC 位运用

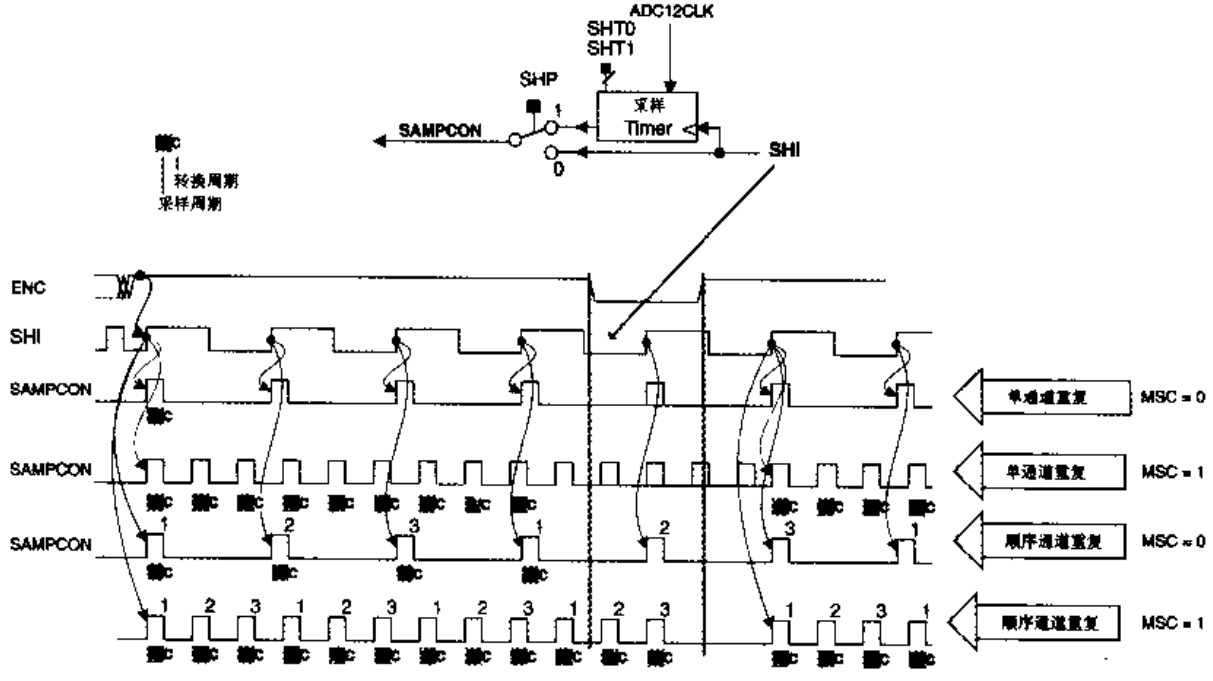


图 15.24 重复模式下的 MSC 位运用

15.7.5　采样时序

由于 A/D 转换的原理是基于电荷再分配，当内部开关切换输入信号进行采样时，因为开关作用会产生流入或流出模拟输入端的转移电流。这些电流主要发生在采样脉冲的前沿和后沿。通常，因为外部的 RC 时间常数总是小于内部的 RC 时间常数，在造成问题之前它们已经过衰减稳定下来。从输入端看，内部的 RC 效应的最大值是 30 pF(电容阵列容量)串接 2 kΩ(开关连通电阻)。但是，如果外部的阻抗很大，在确定的采样时间内，这些瞬变电流就不会稳定，也就无法保证转换的 12 位精度。

为了得到精确的转换，适当的采样时间是必须的。

采样时序分析

图 15.25 是输入端的等效电路。将模拟输入电路的电容从 0 充电到 1/2 LSB 的时间，可以由下列算式导出。

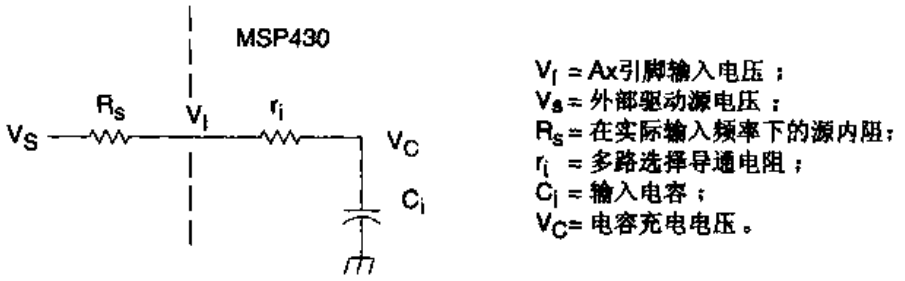


图 15.25　输入等效电路

电容充电电压为

$$V_C = V_S(1 - EXP(-T_C/(R_t \times C_i))) \tag{1}$$

其中：$R_t = R_S + Z_i$，T_C = 采样周期。输入阻抗 Z_i 大约为 1 kΩ(3.0 V)，或 2 kΩ(1.8 V)。1/2 LSB 相当于电压

$$V_C(1/2\ LSB) = V_S - (V_S/8\ 192) \tag{2}$$

由式(1)、式(2)得到

$$V_S - (V_S/8\ 192) = V_S(1 - EXP(-T_C/(R_t \times C_i))) \tag{3}$$

$$T_C(1/2\ LSB) = R_t \times C_i \times \ln(8\ 192)$$

其中：$\ln(8\ 192) = 9.011$。因此，模拟输入信号的稳定时间为

$$T_C(1/2\ LSB) = (R_S + 1\ k\Omega) \times C_i \times 9.011 \tag{4}$$

这一时间必须小于采样时间。如果采用脉冲采样模式，最大的 ADC12CLK 频率为

$$max[f(ADC12CLK)] = SHTx / (T_C(1/2\ LSB)) \tag{5}$$

这一频率不能超越芯片手册中对 ADC12CLK 频率的说明。

15.8　ADC12 控制寄存器

ADC12 用 5 个控制寄存器、16 个转换存储寄存器和 16 个转换存储控制寄存器来配置。所有寄存器只能够用字指令来访问，见表 15.1。

表 15.1 ADC12 控制寄存器一览表

寄存器	缩 写	类 型	地 址	初始状态
ADC 控制寄存器 0	ADC12CTL0	读/写	01A0h	复位(POR)
ADC 控制寄存器 1	ADC12CTL1	读/写	01A2h	复位(POR)
ADC 中断标志寄存器	ADC12IFG	读/写	01A4h	复位(POR)
ADC 中断允许寄存器	ADC12IE	读/写	01A6h	复位(POR)
ADC 中断向量寄存器	ADC12IV	只读	01A8h	复位(POR)
ADC 存储寄存器	ADC12MEM0～ ADC12MEM15	只读	0140h～ 015Eh	不变
ADC 存储控制寄存器	ADC12MCTL0～ ADC12MCTL15	只读	080h～ 08Fh	复位(POR)

15.8.1 控制寄存器 ADC12CTL0 和 ADC12CTL1

ADC12CTLx 的各位在发生 POR 时复位。绝大多数控制位只能在 ENC 复位时修改，以下用灰色背景表示，其余各位的修改无此限制。

ADC12CTL0(01A0h)

15				8	7							0
SHT1				SHT0	MSC	2_5V	REF ON	ADC12 ON	ADC12 OVIE	ADC12 TOVIE	ENC	ADC12 SC
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0) rw-(0) rw-(0) rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

b0：ADC12SC，采样/转换位。

当 ENC 置位时可用软件修改作转换控制。建议将 ISSH 位复位。如果采样信号 SAMPCON由采样定时器产生(SPH＝1)，将 ADC12SC 从“0”改为“1”将启动转换操作。当 A/D 转换完成(BUSY＝0)时 ADC12SC 自动复位。

如果采样直接由 ADC12SC 控制(SHP＝0)，则 ADC12SC 保持为高时采样。当 ADC12SC 复位时开始一次转换。

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，可以用一条指令来完成 ADC12SC 和 ENC 的置位。

b1：ENC，转换允许位。

只有在 ENC 为高时，才能用软件(控制 ADC12SC)或用外部信号启动转换。ADC12CTL0 和 ADC12CTL1 的绝大多数位和 ADCMCTLx 的所有位都只能在 ENC 为低时才能修改。

0：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 为高期间操作有效。

当 CONSEQ＝0 且 ADC12BUSY＝1 时，ENC 由置位变成复位：

当前转换立即停止，转换结果是不确定的；

当 CONSEQ≠0 时，ENC 由置位变复位：

当前转换正常结束，转换结果有效，在当前转换或序列转换结束时转换停止。

b2：ADC12TOVIE，转换时间溢出中断允许位。

当前转换尚未完成时又一次发生采样转换请求，会产生时间溢出。这与由 CONSEQ 选择的转换模式无关。如果时间溢出中断允许标志 ADC12TOVIE 及通用中断允许标志 GIE 置位条件下发生时间中断向量，则会产生中断服务请求，但是没有一个单独的中断标志。详细内容参见 ADC12 中断向量寄存器 ADC12IV 部分。

b3：ADC12OVIE，溢出中断允许位。用于允许溢出中断向量。

当 ADC12MEMx 中原有数据未读出时将新转换结果写入，会发生溢出。如果溢出中断允许标志 ADC12OVIE 及通用中断允许标志 GIE 置位条件下发生时间中断向量，则会产生中断服务请求，但是也没有一个单独的中断标志。详细内容参见 ADC12 中断向量寄存器，ADC12IV 部分。

b4：ADC12ON，打开 ADC 内核位。ADC12 内核的上电时间必须符合要求。

0：ADC 内核不消耗功率，这时无法进行 A/D 转换；

1：ADC 内核消耗功率。如果无 A/D 转换需要，则可以将 ADC12ON 复位以降低功耗。

b5：REFON，参考电压打开位。

0：内部参考电压关闭，参考电压发生器不消耗功率；

1：内部参考电压打开，参考电压发生器消耗功率。参考电压发生器打开时，在第一次采样发生前，必须满足参考电压的建立时间。

b6：2～5 V，参考电平。

0：内部参考电压是 1.5 V，REFON 应为 1；

1：内部参考电压是 2.5 V，REFON 应为 1。

b7：MSC，多次采样转换位。只有在 SHP＝1，并且 A/D 模式选为单通道重复转换、序列通道单次转换或序列通道重复转换模式时有效(CONSEQ≠0)。

0：每次转换，采样定时器需要 SHI 信号的上升沿来触发；

1：首次转换由 SHI 信号的上升沿触发采样定时器，以后的采样及转换会在前一次转换完成后立即进行，不再需要 SHI 上升沿。根据不同的工作模式，在转换序列完成或 ENC 位改变前，SHI 的上升沿不起作用。

b8～11：SHT0，采样保持定时器 0。

这 4 位定义了保存在转换存储寄存器 ADC12MEM0～ADC12MEM7 中的转换结果的采样时序。采样周期是 ADC12CLK 周期×4 的整倍数。

$$T_{sample} = 4 \times \text{ADC12CLK 周期} \times n$$

SHT0	0	1	2	3	4	5	6	7	8	9	10	11	12～15
n	1	2	4	8	16	24	32	48	64	96	128	192	256

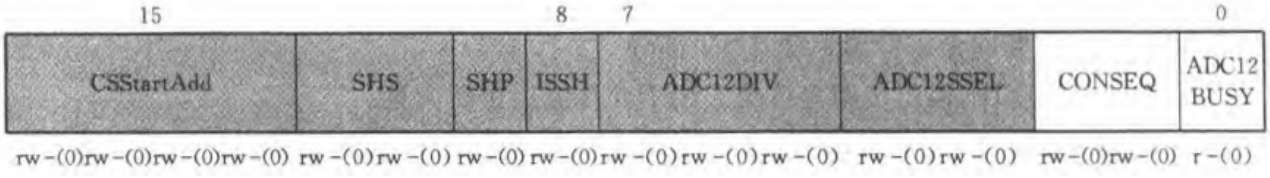
b12～15：SHT1，采样保持定时器 1。

这 4 位定义了保存在转换存储寄存器 ADC12MEM8～ADC12MEM15 中的转换结果的采样时序。采样周期是 ADC12CLK 周期×4 的整倍数。

$$t_{sample} = 4 \times \text{ADC12CLK 周期} \times n$$

SHT1	0	1	2	3	4	5	6	7	8	9	10	11	12～15
n	1	2	4	8	16	24	32	48	64	96	128	192	256

ADC12CTL1(01A2h)



b0：ADC12BUSY，指示一次活动的采样或转换操作。

它专用于单通道单次模式，因为在此模式下如果 ENC 位复位，转换将立即停止，而转换结果是无效的。因此需要在复位 ENC 位之前，测试 ADC12BUSY 位以确定是否为“0”。

在其他模式下此位是无用的，因为复位 ENC 位不会立即发生作用。

0：无活动的操作；

1：处于采样周期，转换或转换序列在进行中。

b1～2：CONSEQ，选择转换模式，CONSEQ.1 置位选择重复模式。

0：单通道单次模式，单一通道转换一次；

1：序列通道单次模式，一个通道序列转换执行一次；

2：单通道重复模式，单一通道的转换重复发生，直至 CONSEQ 置为“0”或“1”。

3：序列通道重复模式，一个通道序列转换重复执行，直至 CONSEQ 置为“0”或“1”。

b3～4：ADC12SSEL，选择转换内核的时钟源。

0：ADC12 内部时钟，ADC12OSC；

1：ACLK；

2：MCLK；

3：SMCLK。

b5～7：选择 ADC12SSEL 位选择的时钟的分频因子。0～7 分别对应分频因子 1～8。对于一次转换需要 13 个时钟节拍。

b8：ISSH，采样输入信号反向。

0：采样输入信号为同向输入。

1：采样输入信号为反向输入。

b9：SHP，选择采样定时器输出或直接选择采样输入信号作为采样信号(SAMPCON)。

0：SAMPCON 直接来自采样输入信号；

1：SAMPCON 来自采样定时器，由采样输入信号的上升沿触发采样定时器。

b10～11：SHS，选择采样输入信号的源。

0：选择控制位 ADC12SC；

1：选择 Timer_A.OUT1；

2：选择 Timer_B.OUT0；

b0～3：INCH，可选择 8 个外部通道或 4 个内部通道之一。

0～7：　a0～a7；

8：　Ve_{REF+}；

9：　V_{REF-}/Ve_{REF-}；

10：　温度二极管；

11～15：　$(AV_{CC}-AV_{SS})/2$。

注意：选择通道 10

选择通道 10 会自动打开片内用于温度二极管的参考电压发生器，但是并不产生 V_{REF+} 输出，也不影响转换选择的参考电压源。

b4～6：Sref，选择 6 种用于转换的参考电压源组合之一。转换范围为 V_{R+}～V_{R-}。

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：　$V_{R+}=Ve_{REF+}$　$V_{R-}=V_{REF-}/Ve_{REF-}$。

b7：EOS，序列结束位，置位时是序列转换中的最后一次转换。

注意：EOS 位

如果 ADC12MCTL15 的 EOS 未置位，则序列会从 ADC12MEM15/ADC12MCTL15 循环进入 ADC12MEM0/ADC12MCTL0。

如果选择了序列模式(CONSEQ 为 1 或 3)，又未将 EOS 位置位，则将 ENC 位复位无法使转换过程停止。要使转换过程停止，必须先选择单次模式(CONSEQ 为 0 或 2)，然后将 ENC 位复位。

15.8.4　中断标志寄存器 ADC12IFG.x 和中断允许寄存器 ADC12IEN.x

中断系统有 16 个中断标志位 ADC12IFG.x，16 个中断允许位 ADC12IE.x，以及 1 个中断向量字。中断标志位和中断允许位对应于 16 个寄存器 ADC12MEMx。

中断标志位与中断允许位在发生 POR 时复位。

ADC12IFG(0184h)

15															0
ADC IFG.15	ADC IFG.14	ADC IFG.13	ADC IFG.12	ADC IFG.11	ADC IFG.10	ADC IFG.9	ADC IFG.8	ADC IFG.7	ADC IFG.6	ADC IFG.5	ADC IFG.4	ADC IFG.3	ADC IFG.2	ADC IFG.1	ADC IFG.0
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

中断标志 ADC12IFG.x 在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，以确保能正确处理发生溢出的情况。如果在 ADC12IFG.x 未复位时有转换数据写入 ADC12MEMx，则将会发生溢出。

ADC12IE(01A6h)

15															0
ADC IE. 15	ADC IE. 14	ADC IE. 13	ADC IE. 12	ADC IE. 11	ADC IE. 10	ADC IE. 9	ADC IE. 8	ADC IE. 7	ADC IE. 6	ADC IE. 5	ADC IE. 4	ADC IE. 3	ADC IE. 2	ADC IE. 1	ADC IE. 0
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

ADC12IE 各位允许或禁止相应的中断标志位 ADC12IFG. x 置位时发生中断请求服务。

15.8.5　中断向量寄存器 ADC12IV

ADC12 的 18 个中断标志共用 1 个中断向量。18 个中断标志按中断优先级组织，中断向量字按当前挂起的最高优先级中断请求发生。表 15.3 是中断优先级与相应的中断向量字。溢出标志 ADC12OVIFG 具有最高的优先级，然后是时间溢出标志 ADC12TOVIFG，接着是对应于转换存储寄存器的标志 ADC12IFG. 0～ADC12IFG. 15。

最高优先级挂起中断标志产生一个 0～36 的数，其中 0 表示无中断挂起。这个数字可以加到 PC 上，来实现自动进入处理相应中断的软件。

如果相应的中断允许位(ADC12OVIE、ADC12TOVIE 或 ADC12IE. x)及通用中断允许位 GIE 置位，一个挂起的中断标志会立即产生中断请求，而使产生中断请求的最高优先级得到服务。

ADC12OVIFG 和 ADC12TOVIFG 会在访问 ADC12IV 寄存器后自动复位。例如，同时发生了这两个中断标志的挂起，ADC12OVIFG 会在第一次访问 ADC12IV 时复位，如果不立即再次发生 ADC12OVIFG 置位，则 ADC12TOVIFG 在第二次访问 ADC12IV 时复位。但是 ADC12IFG. x 必须用软件复位或通过访问相应的转换存储寄存器 ADC12MEMx 来自动复位。

ADC12OVIFG 和 ADC12TOVIFG 不能通过软件访问。惟一的观察方法是通过中断向量字的数值进行。

表 15.3　中断向量字 ADC12IV 的值

ADC12IFG																ADC12 OV	ADC12 OV	ADC12 IV
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0	1	2
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	1	0	4
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	1	0	0	6
x	x	x	x	x	x	x	x	x	x	x	x	x	x	1	0	0	0	8
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
x	1	0	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	0	36

```
                ADD     &ADC12MEM1,R6          ; ADC12IFG1 复位
                ...                            ; A/D 数据处理
                RETI                           ; 返回主程序                    5
;
; 以下程序说明检查是否有挂起中断,需 5 周期,但如有挂起中断可节省 9 周期
;
ADC12MOD0                                      ; Vector 6: ADC12MEM0 加载
                ...                            ; A/D 处理
                JMP     ADC12_HND              ; 再入程序检查挂起中断          2
;
                .SECT   "VECTORS",0FFF4h
                .WORD   ADC12_HND              ; ADC12 中断向量
```

注意:基础时钟系统

如果 CPU 的 MCLK 是关闭的(CPUOff=1),则 CPU 起动的同步需要 2.3 个周期。这一个周期的误差是因为时钟信号与 MCLK 的重启动是异步的。

15.9　ADC12 接地与降噪

作为高精度的转换电路,必须十分注意印制板布线和接地方法,以消除地电流环路、寄生参数效应和噪声。应该遵守工业标准的接地和布线技术来达到这一目的。

地电流环路是因为 A/D 电路与其他模拟或数字电路形成公共回路引起的。如果设计时不仔细,则这一电流会产生一个很小的电压叠加在 A/D 转换器的参考电压或输入信号上。消除地电流环路的方法之一是将 AV_{SS}接成星形,见图 15.26。这种接法使地电流或参考电压源的电流不经过公共的导体,因此消除了引起误差的电压。

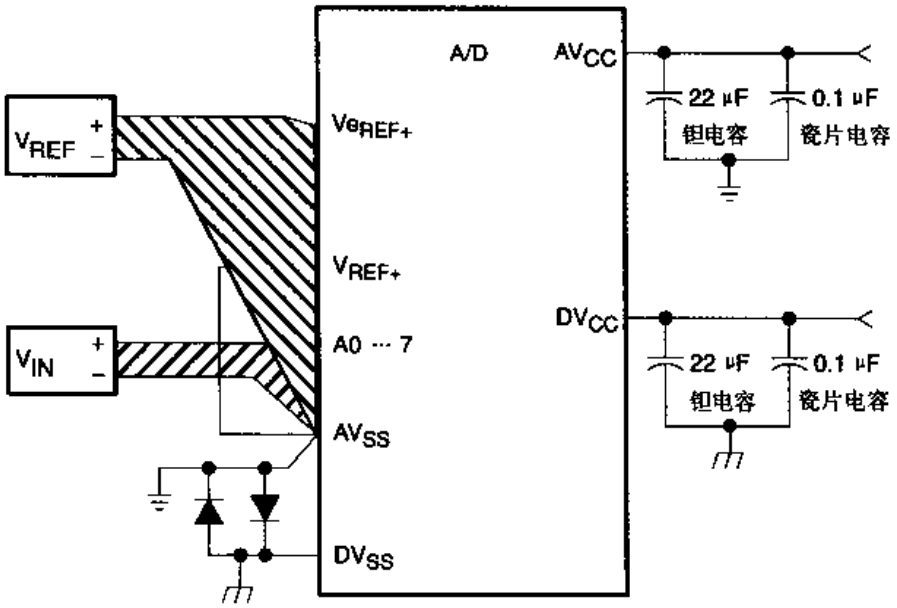


图 15.26　A/D 的接地及降低噪声

数字地 DV_{SS}和模拟地 AV_{SS}也可以采用星形接法。如果数字部分和模拟部分是分开供电的，则可以在这两点之间接入反向并接的二极管对，以消除低于 700 mV 的电压差。

除了接地，因为数字开关及开关电源在电源线路上产生的纹波和噪声脉冲，也会影响转换的结果。如果将参考电压范围(V_{R+}～V_{R-})缩小，那么纹波的影响会变得更明显，从而可能影响到 LSB。因此，电源的清洁无噪声就非常重要。仔细地安排到各自接地点的旁路电容对于减少噪声的影响是很有用的。

第 16 章　FLASH 型芯片的开发

嵌入式系统的开发往往借助于开发系统工具。而各种开发系统一般都比待开发调试的嵌入式系统要复杂得多。

MSP430 的 FLASH 型芯片，由于具有片内的 FLASH 型主存储器和经 JTAG 接口实现的各种调试功能，它的开发系统工具无论是结构还是操作，都要简便得多。

16.1　开发系统概述

16.1.1　开发技术

开发调试功能的实现有多种技术方案，各有不同的应用特点。

CPU 仿真

所谓 CPU 仿真的原理是将待开发调试样机系统的 CPU 用开发系统临时取代。常见的是将 CPU 暂时拔离插座，将开发系统的仿真插头经此插座接入。实际上开发系统与样机的连接主要是所谓的三总线，即：数据线、地址线和控制线。而开发系统内部有一个与样机系统的 CPU 相同的 CPU，控制着三总线。这样一来，样机系统除了 CPU 外，就成了开发系统的一个扩展部分。样机 CPU 在调试完成后重新插回。

根据它的工作原理，开发一种 CPU 的样机，就需要有一种相对应的开发系统。同时，开发系统本身所占据的资源，如：存储空间、I/O 地址、中断源等，必须与样机工作时所需的避开，以免冲突。高性能的开发系统所占用的系统资源可以做到非常少。

为了完成开发系统的调试功能，需要实现程序的代码下载、运行控制和现场观察和修改，其中，实现用户程序的运行断点是最重要的。有两种实现方法，即：软件断点及硬件断点。

软件断点

软件断点是在待调试的用户程序的断点处临时用子程序调用指令取代。使得程序执行到此处时转移到监控程序中，同时将断点地址保护在堆栈顶。这种方法要求待调试的程序在 RAM 中，以便修改指令。实现断点的子程序调用指令应该是单字节（对子 8 位指令系统）或单字（16 位指令系统）。

硬件断点

硬件断点是用一组可以预设断点条件的硬件电路监视程序的运行，当运行中出现与断点条件相符的情况时立即以硬件方法中止用户程序的运行。这种方法，要求对于每一个断点准备一套硬件。一次调试可以设置的断点数只有少数的几个。硬件断点的设置不修改程序代码，因此可以调试 ROM 中的程序。

今天，单片机的结构越来越紧凑，三总线从芯片的引脚引出的越来越少，用 CPU 仿真形式实现开发也越来越不容易。

ROM 仿真

所谓 ROM 仿真的原理，是将待调试的用户程序引出到样机系统外的 RAM 中，由外部的

一台调试主机作代码修改和运行控制。它仍然需要经过三总线来实现开发系统与样机的连接，它是利用外部主机来实现样机 ROM 中程序的调试及修改。

BOOT ROM 引导程序

如果在样机系统中预先驻留一个小程序，那么使得用户样机系统一上电就可以与调试主机实现通信，达到下载用户程序代码的目的。这就是通常称为系统引导程序(BOOT ROM)。这种方法要求系统的硬件基本无故障，起码能保证正常的通信。当系统的程序代码装入 RAM、E^2PROM 或 FLASH 存储器后，BOOT ROM 可以不再工作。

如果样机的 CPU 有一个可靠的引导程序来实现程序调试的第一步，那么将大大提高调试程序的效率。

16.1.2 MSP430 系列的开发

MSP430 系列芯片目前暂未将数据、地址、控制总线从引脚引出。因此它的开发方式暂不采用三总线连接方式。MSP430 不同型号的芯片有不同的开发方式。但是对于 FLASH 型，最方便的开发方式是经过 JTAG 接口实现的联机调试方式和用 BOOT ROM 中程序经串行的通信调试方式。

在 FLASH 型问世前，MSP430 系列单片有以下几种开发方式。

EPROM 方式

MSP430 的 11x、31x、32x、33x 等系列，都有相应的 EPROM 型芯片，如：MSP430E112、MSP430E325、MSP430E337A 等。当程序调试工作量不大时，这是可以采用的方法之一。即用同系列的 EPROM 型代替，通过反复擦写 EPROM 来完成程序调试。但是调试效率很低。

由于相应的 EPROM 型的引脚封装与 OTP 型或 ROM 型不尽相同，不能做到直接经插座来替代。EPROM 型芯片价格昂贵，工作时的环境温度范围也小得多，难以将它用于最终的产品中。

OTP 方式

利用一次性编程芯片来调试程序，如 MSP430P112 等。适用于程序只需少数几次修改即能完成调试的情况。

MSP430 系列开发系统

作为第三方，有多家公司生产了 MSP430 的开发系统，主要是针对 31x、32x、33x 系列的。这些开发系统实现了 CPU 仿真。

为了实现这一点，开发系统内实际上是用多片 FPGA 构成了一个仿真的 MSP430 芯片。这样一来，开发系统的确实现了全部的开发功能，但是代价不小。这些产品都很昂贵。

由于 MSP430 芯片大多采用表贴封装，不能方便地通过插座来实现开发系统与样机系统的连接。

16.1.3 MSP430F 系列的开发

MSP430 的 FLASH 型的出现，也使开发方式有了根本的改变。FLASH 型的开发，只需要一台 PC 和一个称为 FET(FLASH Emulator Tool)的 JTAG 控制器即可实现。它实现的开发功能也是类似于 CPU 仿真。FET 的 JTAG 接口与芯片的连接只有几个引脚。

同时，由于 FLASH 存储器可以在线编程，FLASH 型芯片内有 BOOT ROM，已预先驻留

了引导程序，通过特殊的启动过程即可实现与外部调试主机的通信。

16.2　FLASH 型的 FET 开发方法

MSP430 系列的 FLASH 型，即 MSP430F11x、F11x1、F13x 和 F14x，都具有基本相同的开发方法，即：通过 JTAG 控制器可以实现程序代码的下载、实现运行控制和对现场进行观察与修改。这种方法被称为 FET 调试方法。

16.2.1　MSP430 芯片的 JTAG 接口

JATG 是一种所谓的边界扫描技术标准，即 IEEE1149.1。这是为了在线测试的需要而发展起来的针对芯片及线路板测试的接口技术。它利用串行移位输入及输出的方式对芯片的输入端加载特定的代码序列，并获得输出端给出的响应序列。全部 JTAG 接口只有很少的几根引脚。

MSP430 系列的 FLASH 型芯片在内部都集成了 JTAG 模块，使得用户可以经过 JTAG 接口实现 CPU 仿真调试功能。整个调试过程，外部只需要一台能实现 JTAG 接口控制功能的主机即可。一般用一台 PC 电脑，经过打印机接口就能实现。

各种 FLASH 型芯片与 JTAG 接口相关的引脚见表 16.1。

由于 F11x1 芯片只有 20 个引脚，因此 TDO、TDI、TMS 及 TCK 等引脚采用复用的方式引出，并增加了一个 TEST 引脚来区别。在系统设计时需要注意避免这些引脚上的外部电路对 F11x1 芯片提供的信号与 JTAG 在功能上发生冲突。

表 16.1　JTAG 调试接口引脚

F11x1	引　脚	F13x/F14x	引　脚
TDO　（与 P17 复用）	20	TDO	54
TDI　（与 P16 复用）	19	TDI	55
TMS　（与 P15 复用）	18	TMS	56
TCK　（与 P14 复用）	17	TCK	57
RST	7	RST	58
TCLK/XOUT	5	TCLK/XOUT	9
TEST	10		

16.2.2　FLASH 型仿真工具

FET 有多种产品，如：TI 公司的 FET430X110 和 FET430P140 等。FET 的工作要和调试环境软件配合，如：IAR 系统公司的 Embedded Workbench。

FET 的基本调试功能有：

程序下载

当用户将源程序（汇编语言或 C 语言）经语法检查无误并生成代码时，就可以将程序代码在如图 16.4 的环境中下载到 FLASH 芯片中，而用户的系统可以是在线状态。

设置断点

用户可以通过调试环境软件的人机对话界面，在程序中设置断点。视 FLASH 型的型号

不同，可以同时设置 2 或 3 个断点。这些断点是硬件断点，它是经过 JTAG 接口的传输，由芯片中的几组断点条件寄存器实现的。因此可以调试 FLASH 存储器及 RAM 中的程序，也不改变用户程序的运行条件。

现场观察与修改

用户可以通过调试环境软件的人机对话界面，检查或修改 FLASH 芯片内的各种存储器、寄存器的数据。这些检查与修改也是通过 JTAG 接口的传输完成的。

关于 FET 调试环境的详细内容，请查阅调试环境软件的使用手册。

16.3 FLASH 型的 BOOT ROM

引导加载程序（Bootstrap Loader，BSL）是用于在 MSP430 设计开发及系统更新时对 FLASH 存储器的编程。它可以用经 UART 协议传送的命令来激活，使得用户可以通过一台 PC 控制 MSP430，并实现数据交换。

所用的协议是 MSP430 的具有先导同步字符的标准串行通信协议（SSP）。为了避免 BSL 代码被错误的写操作覆盖，这些代码保存在专用的 BOOT ROM 中。

对于 BSL，任何直接或间接的读命令都需通过保护口令的验证。

进入 BSL，要利用几个引脚经过一个 BSL 进入序列来实现。然后由一个同步字符和后续的命令数据帧来启动所需功能。

16.3.1 标准复位过程和进入 BSL 过程

MSP430F11x1

对于 MSP430F11x1，在 RST/NMI 和 Test 引脚上加一个适当的进入时序，可以使芯片不用地址 0FFFEh 处的 RESET 向量，而改用 BSL RESET 向量来启动程序。可以直接利用 RS232 接口中的 DTR 和 RTS 信号，转换成 TTL 电平后，来驱动这两个引脚。

正常 RESET 的时序见图 16.1。在 RST/NMI 引脚电平上升时，Test 保持为低，这时，地址 0FFFEh 中的 RESET 向量成为程序入口。

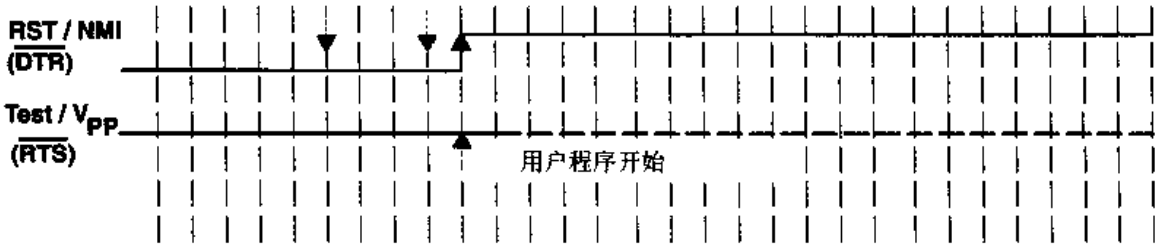


图 16.1 F11x1 的正常复位时序

当 RST/NMI 引脚电平为低时，Test 引脚收到至少 2 个上升沿，并在 RST/NMI 引脚出现上升沿时保持为高，BSL 的 RESET 向量就被用作程序入口，时序见图 16.2。这样一种既有上升沿又有电平检查的触发方式，提高了进入 BSL 的可靠性。

MSP430F11x1 的 Test 引脚的基本功能之一，是在片内用于切换 P1.4～P1.7 引脚的基本功能和 JTAG 功能。如果 Test 引脚发生第二个上升沿时 RST/NMI 引脚为低，则 Test 信

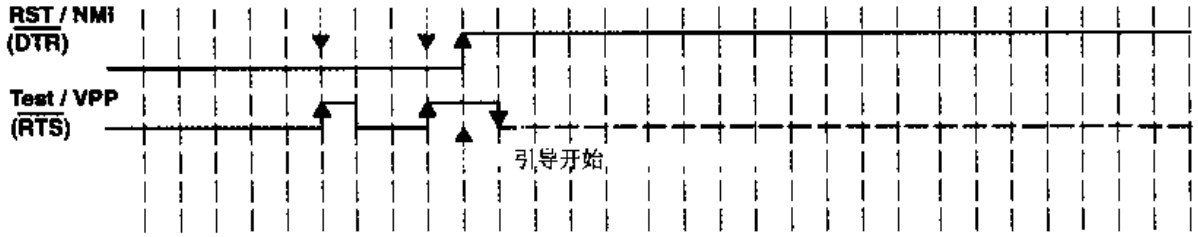


图 16.2　F11x1 的 BSL 复位时序

号就在片内锁存为低。

如果发生以下情况，则 BSL 不会实现经过 BSL RESET 向量的启动。

- 在 RST/NMI 引脚保持为低时，Test 引脚上出现的上升沿少于 2 个。
- 在 RST/NMI 引脚互相上升沿时，Test 引脚为低。
- JTAG 接口已控制了 MSP430 的资源。
- 因 V_{CC}下降引起 POR。

MSP430F13x/14x

对于 MSP430F13x/14x，由于没有 JTAG 与 P1 端口复用的问题，所以不再需要 Test 引脚。它利用 TCK 引脚和 RST/NMI 引脚的时序变化来实现进入 BSL。

正常 RESET 的时序见图 16.3。如果在 RST/NMI 上升时 TCK 一直保持为高，那么这时，地址 0FFFEh 中的 RESET 向量用作程序入口。

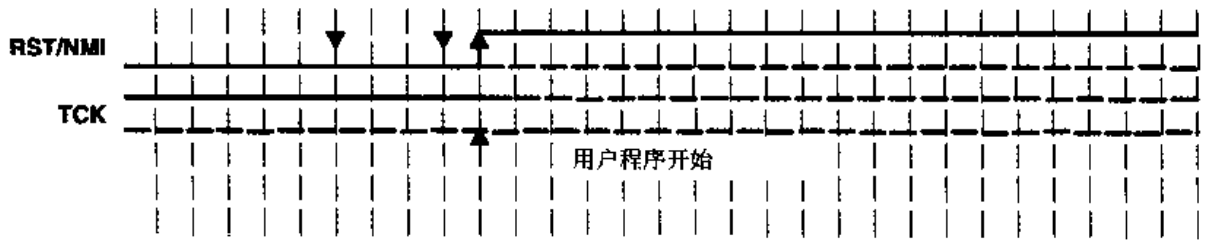


图 16.3　F13x、F14x 的正常复位时序

如果在 RST/NMI 引脚为低时 TCK 引脚收到至少 2 个下降沿，并在 RST/NMI 引脚出现上升沿时保持为低，那么 BSL RESET 向量就被用作程序入口，时序见图 16.4。当 TCK 引脚变化时，TMS 引脚必须保持为高，以确保 JTAG 控制器保持在它的缺省模式。

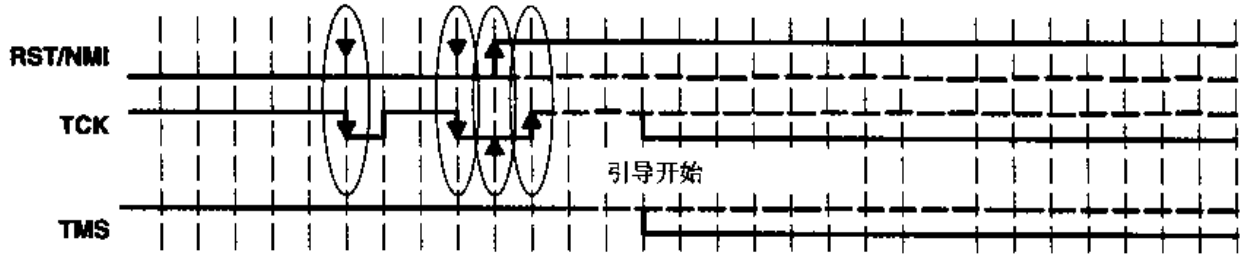


图 16.4　F13x、F14x 的 BSL 复位时序

如果发生以下情况，则 BSL 不会实现经过 BSL RESET 向量的启动。

- 在 RST/NMI 引脚保持为低时，TCK 引脚上出现的下降沿少于 2 个。

- 在 RST/NMI 引脚互相上升沿时，TCK 引脚为高。
- JTAG 控制了 MSP430 的资源。
- 因 V_{cc}下降引起 POR。

BSL 的 RESET 向量在 0C00h 处，早期的一些 MSP430F11x1 芯片在 0800h 处。

BOOT ROM 程序为 1 KB。

16.3.2 BSL 的 UART 协议

BSL 的异步通信协议的基本参数如下：

- 波特率固定为 9 600，半双工，数据格式为 8 位数据、偶校验和 1 个停止位。
- P1.1 为 TX 端(设置为输出高)，P2.2 为 RX 端(设置为输入)。
- 用响应字符实现握手协议。

BSL 在收到任何命令之前，必须收到同步字符 SYNC(80h)。它被用于计算 UART 时序及 FLASH 存储器编程时序，是 BSL 系统的时基参考源。

BSL 用送回响应字符 DATA_ACK(90h)来确认 BSL 接收成功。

BSL 命令分为两类，即：无保护命令和口令保护命令。无保护命令包括接收口令和擦除主存。口令保护命令有接收数据块(对 FLASH 存储器或 RAM 编程)、发送数据块、段擦除和加载 PC 值并启动用户程序等。

如果因为线路故障或数据帧错造成 UART 失控，则恢复 BSL 的方法只能是再次进行 BSL 的启动过程。

对 FLASH 存储器及 RAM 的写入访问是在线的。接收到一个数据字节后立即处理，并在后续字节到来前完成写入。因此，整个写入时间取决于波特率，也不需要内部缓存。

编程数据只能进入 FLASH 存储器、RAM 或字节型外围模块寄存器，因此不会影响到 FLASH 存储器的控制寄存器。

16.3.3 数据格式

BSL 命令数据帧的帧结构见表 16.2 的说明。

表 16.2 BSL 命令数据帧格式

命 令	HDR	CMD	L1	L2	AL	AH	LL	LH	D1	D2…Dn	CKL	CKH	响 应
RX 数据块	80	13	n	n	AL	AH	n−4	0	D1	D2…Dn−4	CKL	CKH	ACK
RX 口令	80	10	24	24	xx	Xx	xx	xx	…	D2…D20	CKL	CKH	ACK
段擦除	80	16	04	04	AL	AH	02	02	—	—	CKL	CKH	ACK
主存擦除	80	18	04	04	xx	xx	xx	xx	—	—	CKL	CKH	ACK
加载 PC	80	1A	04	04	AL	AH	xx	xx	—	—	CKL	CKH	ACK
TX 数据块	80	14	04	04	AL	AH	n	n	—	…	CKL	CKH	—
BSL 回答	80	xx	n	n	D1	D2…Dn					CKL	CKH	—

HDR：　帧头，80h。

CMD：　命令标识。

L1，L2：　从 AL 到 Dn 的字节数，必须满足：L1＝L2，L1＜255，L1 为偶数。

由地址字节 AL、AH 选择适当的段，待擦除的段内的任意偶数地址都有效。如果段擦除成功，则 BSL 将返回 DATA_ACK。

擦除 0 段(Segment 0)也将口令初始化为 32 个 0FFh。

对于 MSP430F1121 和 F1101，段的分割如下：

0FE00h…0FFFFh	Segment 0	主存储区
0FC00h…0FDFFh	Segment 1	
0FA00h…0FBFFh	Segment 2	(仅对 F1121)
0F800h…0F9FFh	Segment 3	(仅对 F1121)
0F600h…0F7FFh	Segment 4	(仅对 F1121)
0F400h…0F5FFh	Segment 5	(仅对 F1121)
0F200h…0F3FFh	Segment 6	(仅对 F1121)
0F000h…0F1FFh	Segment 7	(仅对 F1121)
01080h…010FFh	Segment A	信息存储区
01000h…0107Fh	Segment B	(仅对 F1121)

对于 MSP430F13x 和 F14x，段的分割与 F1121 基本相同，从 Segment 0 开始，因各型号的 FLASH 存储器空间大小不同，段数也有差异。最后一个段(Segment 119)只有 256 字节：

0FE00h…0FFFFh	Segment 0	主存储区
0FC00h…0FDFFh	Segment 1	
…		
01200h…013FFh	Segment 118	
01100h…011FFh	Segment 119	
01080h…010FFh	Segment A	信息存储区
01000h…0107Fh	Segment B	

加载 PC

加载 PC 命令可以将地址范围内的任意地址加载到 PC(R0)。有口令保护。

在收到加载 PC 数据帧后，BSL 返回响应字符 DATA_ACK，并将收到的指定地址送入 PC。于是，程序从这一地址开始继续执行，BSL 的通信终止。

发送数据块

发送数据块命令用于读出 FLASH 存储器、RAM 或从 0000h～00FFh 地址范围内的外围模块寄存器。有口令保护。

由 AL、AH 定义 16 位数据块起始地址。由 LL、LH 定义 16 位数据块长度，但是实际帧内的数据长度不超过 250 字节，因此 LH 总是为“0”。

BSL 在收到发送数据块命令后，将指定的数据块组成帧送回：先发送 HDR，接着是任意数值的 CMD 字节，然后是数据长度 L1、L2 和全部数据字节，最后是校验和字节 CKL、CKH。BSL 发送的数据帧不需要响应字符的确认。

校验和

16 位校验和的计算包括除校验和外的所有字节，以字异或后取反码方法计算，即

$$Checksum = inv[(B1 + 256 \times B2) \text{ xor } (B3 + 256 \times B4) \text{ xor} \cdots \text{xor } (Bn - 1 + 256 \times Bn)]$$

或

CKL=inv[B1 xor B3 xor…xor Bn-1], CKH=inv[B2 xor B4 xor…xor Bn]

16.3.4　退出 BSL

退出 BSL 有两种情况：

- MSP430 在收到加载 PC 命令后，从指定的程序地址继续执行。
- 用标准的 RESET 序列强制使 MSP430 用 0FFFEh 的向量地址开始执行程序。

16.3.5　保护口令

口令保护功能能阻止所有直接或间接读取数据的命令。在进入 BSL 后，只有主存擦除命令和接收口令命令可以不必先接收正确的口令就能执行。

通过接收口令命令接收到正确的口令，将对后续的所有命令解除口令保护。

一旦口令解除，这一状态一直保持到下一次的 BSL 进入过程。发生复位不影响口令保护功能。

口令由从 FFE0h～FFFFh 的 16 个中断向量组成。FFE0h 是口令的第一个字节。经过主存擦除或未编程过的芯片，口令的所有位都为“1”。

用户要十分当心，如果修改了中断向量，并且启动一次新的 BSL 通信则会引起口令的更新。

16.3.6　BSL 的内部设置和资源

启动 BSL 后会影响以下初始状态：

- 看门狗定时器关闭。
- 关中断(GIE=0、NMIIE=0、OFIFG=0、ACCVIFG=0)。
- 堆栈指针不变，但是如果堆栈指针在非法地址就初始化为 021Ah。
- 定义基础时钟模块，使最小时钟频率为 1.5 MHz。

  BCSCTL1=85h　　(RESL=5,XT2Off=1)

  DCOCTL=80h　　(DCO=4,MOD=0)

  由 DCO 产生 MCLK 及 SMCLK。
- 用 Timer_A 建立软件异步通信协议，连续模式，时钟源为 MCLK，DIV=1。

  CCR0 用于比较。

  CCTL0 用于查询 CCIFG0 标志。
- P1.1 为 TX 端(设置为输出高)。
- P2.2 位 RX 端(设置为输入)。
- 启动命令的口令保护功能。

在系统初始化后，BSL 处于就绪状态，等待同步序列及后续的第一个命令的到来。

BSL 占用的存储器及资源：

- BSL 程序代码在 BOOT ROM 中(0C00h… 0FEFh)，0C00 开始的 2 字节为 BSL 的 RESET向量，0FF0h…0FFFh 保有芯片的标识。
- 程序占用 RAM 的 0200…213h 区域。
- 程序占用寄存器 R5…R9，它们的内容未保护。

- 用到的基础时钟模块寄存器有：

  BCSCTL1 (057h)

  DCOCTL (056h)

- 用到的 Timer_A 的寄存器有：

  TACTL (0160h)

  CCTL0 (0162h)

  TAR (0170h)

  CCR0 (0172h)

- BSL 不影响中断服务程序。

附录 A　寻址空间

本附录将外围模块和控制位信息分类列表，以供参考。

每一外围模块的寄存器表示为一行含寄存器控制位或状态位的表格。寄存器符号和外围模块的 16 进制地址在每一行的左侧。

各位的可访问性与/和硬件条件用以下符号标明在每一位的下方。

- rw：　读/写。
- r：　只读。
- r0：　读出为“0”。
- r1：　读出为“1”。
- w：　只写。
- w0：　写入“0”。
- w1：　写入“1”。
- (w)：　无寄存器位，写入“1”将产生一个脉冲，读出总为“0”。
- h0：　由硬件复位。
- h1：　由硬件置位。
- -0,-1：　PUC 信号活动(Reset 或 WDT)后的状态。
- -(0),-(1)：　POR 信号活动(Reset)后的状态。

各寄存器的访问特性要参照书中的相关说明。

表 A.1　SFR——字节访问

000Fh								
模块允许 2,ME2 0005h			UTXE1 rw-0	URXE1 USPIE1 rw-0				
中断标志 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		NMIIFG rw-0			OFIFG rw-1	WDTIFG rw-0
中断允许 2,IE2 0001h			UTXIE1 rw-0	URXIE1 rw-0				
中断允许 1,IE1 0000h	UTXIE0 rw-0	URXIE0 rw-0	ACCVIE rw-0	NMIIE rw-0			OFIE rw-0	WDTIE rw-0

注：有外围模块时才有相应的 SFR 位。

表 A.2a 数字 I/O 组——字节访问

Bit # →	7	6	5	4	3	2	1	0
功能选择寄存器，P4SEL 001Fh	P4SEL.7 rw-0	P4SEL. rw-0	P4SEL.5 rw-0	P4SEL.4 rw-0	P4SEL.3 rw-0	P4SEL.2 rw-0	P4SEL.1 rw-0	P4SEL.0 rw-0
方向寄存器，P4DIR 001Eh	P4DIR.7 rw-0	P4DIR.6 rw-0	P4DIR.5 rw-0	P4DIR.4 rw-0	P4DIR.3 rw-0	P4DIR.2 rw-0	P4DIR.1 rw-0	P4DIR.0 rw-0
输出寄存器，P4OUT 001Dh	P4OUT.7 rw	P4OUT.6 rw	P4OUT.5 rw	P4OUT.4 rw	P4OUT.3 rw	P4OUT.2 rw	P4OUT.1 rw	P4OUT.0 rw
输入寄存器，P4IN 001Ch	P4IN.7 r	P4IN.6 r	P4IN.5 r	P4IN.4 r	P4IN.3 r	P4IN.2 r	P4IN.1 r	P4IN.0 r
功能选择寄存器，P3SEL 001Bh	P3SEL.7 rw-0	P3SEL.6 rw-0	P3SEL.5 rw-0	P3SEL.4 rw-0	P3SEL.3 rw-0	P3SEL.2 rw-0	P3SEL.1 rw-0	P3SEL.0 rw-0
方向寄存器，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
输出寄存器，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
输入寄存器，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								
0010h								

表 A.2b 数字 I/O 组——字节访问

Bit # →	7	6	5	4	3	2	1	0
功能选择寄存器，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
方向寄存器，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
输出寄存器，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
输入寄存器，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
功能选择寄存器，P5SEL 0033h	P5SEL.7 rw-0	P5SEL.6 rw-0	P6SEL.5 rw-0	P5SEL.4 rw-0	P5SEL.3 rw-0	P5SEL.2 rw-0	P5SEL.1 rw-0	P5SEL.0 rw-0
方向寄存器，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
输出寄存器，P5OUT 0031h	P5OUT.7 w	P5OUT.6 rw	P5OUT.5 rw	P5OUT.4 rw	P5OUT.3 rw	P5OUT.2 rw	P5OUT.1 rw	P5OUT.0 rw
输入寄存器，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								
功能选择寄存器，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
中断允许，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

表 A.5a USART 寄存器(UART 模式)——字节访问

Bit # —	7	6	5	4	3	2	1	0
USART1 发送缓存 UTXBUF1 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
USART1 接收缓存 URXBUF1 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
USART1 波特率 UBR11 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
USART1 波特率 UBR01 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
USART1 调整控制 UMCTL1 07Bh	m7 rw	m6 rw	m5 rw	m4 rw	m3 rw	m2 rw	m1 rw	m0 rw
USART1 接收控制 URCTL1 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
USART1 发送控制 UTCTL1 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
USART1 USART 控制 UCTL1 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 UTXBUF0 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
USART0 接收缓存 URXBUF0 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
USART0 波特率 UBR10 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
USART0 波特率 UBR00 074h	2^7 rw	2^6 rw	2^5 rw	2^5 rw	2^3 rw	2^2 rw	2^1 rw	2^0 rw
USART0 调整控制 UMCTL0 073h	m7 rw	m6 rw	m5 rw	m4 rw	m3 rw	m2 rw	m1 rw	m0 rw
USART0 接收控制 URCTL0 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
USART0 发送控制 UTCTL0 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
USART0 USART 控制 UCTL0 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

表 A.5b USART 寄存器(SPI 模式)——字节访问

Bit # →	7	6	5	4	3	2	1	0
USART1 发送缓存 UTXBUF1 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
USART1 接收缓存 URXBUF1 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
USART1 波特率 UBR11 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
USART1 波特率 UBR01 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
USART1 调整控制 UMCTL1 07Bh	m7 rw	m6 rw	m5 rw	m4 rw	m3 rw	m2 rw	m1 rw	m0 rw
USART1 接收控制 URCTL1 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
USART1 发送控制 UTCTL1 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
USART1 USART 控制 UCTL1 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
USART0 发送缓存 UTXBUF0 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
USART0 接收缓存 URXBUF0 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
USART0 波特率 UBR10 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
USART0 波特率 UBR00 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
USART0 调整控制 UMCTL0 073h	m7 rw	m6 rw	m5 rw	m4 rw	m3 rw	m2 rw	m1 rw	m0 rw
USART0 接收控制 URCTL0 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
USART0 发送控制 UTCTL0 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
USART0 USART 控制 UCTL0 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

表 A.6a ADC12 寄存器——字访问或字节访问

Bit #	7	6	5	4	3	2	1	0
ADC12MCTL15† 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)
ADC12MCTL14† 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)
ADC12MCTL13† 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)
ADC12MCTL12† 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)
ADC12MCTL11† 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)
ADC12MCTL10† 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)
ADC12MCTL9† 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)
ADC12MCTL8† 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)
ADC12MCTL7† 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)
ADC12MCTL6† 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)
ADC12MCTL5† 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)
ADC12MCTL4† 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)
ADC12MCTL3† 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)
ADC12MCTL2† 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)
ADC12MCTL1† 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)
ADC12MCTL0† 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)

† ADC12MCTLx 寄存器各位只能在 ENC=0 时修改。

表 A.6b ADC12 寄存器——字访问或字节访问

Bit #	15	14	13	12	11 … 0
ADC12MEM15 015Eh	Unused r0	Unused r0	Unused r0	Unused r0	MSB ← 转换结果 → LSB rw
ADC12MEM14 015Ch	Unused r0	Unused r0	Unused r0	Unused r0	MSB ← 转换结果 → LSB rw
ADC12MEM13 015Ah	Unused r0	Unused r0	Unused r0	Unused r0	MSB ← 转换结果 → LSB rw
ADC12MEM12 0158h	Unused r0	Unused r0	Unused r0	Unused r0	MSB ← 转换结果 → LSB rw
ADC12MEM11 0156h	Unused r0	Unused r0	Unused r0	Unused r0	MSB ← 转换结果 → LSB rw
ADC12MEM10 0154h	Unused r0	Unused r0	Unused r0	Unused r0	MSB ← 转换结果 → LSB rw
ADC12MEM9 0152h	Unused r0	Unused r0	Unused r0	Unused r0	MSB ← 转换结果 → LSB rw
ADC12MEM8 0150h	Unused r0	Unused r0	Unused r0	Unused r0	MSB ← 转换结果 → LSB rw
ADC12MEM7 014Eh	Unused r0	Unused r0	Unused r0	Unused r0	MSB ← 转换结果 → LSB rw
ADC12MEM6 014Ch	Unused r0	Unused r0	Unused r0	Unused r0	MSB ← 转换结果 → LSB rw
ADC12MEM5 014Ah	Unused r0	Unused r0	Unused r0	Unused r0	MSB ← 转换结果 → LSB rw
ADC12MEM4 0148h	Unused r0	Unused r0	Unused r0	Unused r0	MSB ← 转换结果 → LSB rw
ADC12MEM3 0146h	Unused r0	Unused r0	Unused r0	Unused r0	MSB ← 转换结果 → LSB rw
ADC12MEM2 0144h	Unused r0	Unused r0	Unused r0	Unused r0	MSB ← 转换结果 → LSB rw
ADC12MEM1 0142h	Unused r0	Unused r0	Unused r0	Unused r0	MSB ← 转换结果 → LSB rw
ADC12MEM0 140h	Unused r0	Unused r0	Unused r0	Unused r0	MSB ← 转换结果 → LSB rw

表 A.6c ADC12 寄存器——字访问或字节访问

Bit #	15	14	13	12	11	10	9	8
ADC12IE 01A6h	ADC12IE. 15 rw -(0)	ADC12IE. 14 rw -(0)	ADC12IE. 13 rw -(0)	ADC12IE. 12 rw -(0)	ADC12IE. 11 rw -(0)	ADC12IE. 10 rw -(0)	ADC12IE. 9 rw -(0)	ADC12IE. 8 rw -(0)
ADC12IFG 01A4h	ADC12IFG. 15 rw -(0)	ADC12IFG. 14 rw -(0)	ADC12IFG. 13 rw -(0)	ADC12IFG. 12 rw -(0)	ADC12IFG. 11 rw -(0)	ADC12IFG. 10 rw -(0)	ADC12IFG. 9 rw -(0)	ADC12IFG. 8 rw -(0)
ADC12CTL1 01A2h	CStartAdd. 3† rw -(0)	CStartAdd. 2† rw -(0)	CStartadd. 1† rw -(0)	CStartAdd. 0† rw -(0)	SHS. 1† rw -(0)	SHS. 0† rw -(0)	SHP† rw -(0)	ISSH† rw -(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)

†只能在 ENC=0 时修改。

续表 A.6c

Bit # —	7	6	5	4	3	2	1	0
ADC12IE 01A6h	ADC12IE.7 rw-(0)	ADC12IE.6 rw-(0)	ADC12IE.5 rw-(0)	ADC12IE.4 rw-(0)	ADC12IE.3 rw-(0)	ADC12IE.2 rw-(0)	ADC12IE.1 rw-(0)	ADC12IE.0 rw-(0)
ADC12IFG 01A4h	ADC12IFG.7 rw-(0)	ADC12IFG.6 rw-(0)	ADC12IFG.5 rw-(0)	ADC12IFG.4 rw-(0)	ADC12IFG.3 rw-(0)	ADC12IFG.2 rw-(0)	ADC12IFG.1 rw-(0)	ADC12IFG.0 rw-(0)
ADC12CTL1 01A2h	ADC12DIV.2† rw-(0)	ADC12DIV.1† rw-(0)	ADC12DIV.0† rw-(0)	ADC12SSEL.1† rw-(0)	ADC12SSEL.0† rw-(0)	CONSEQ.1 rw-(0)	CONSEQ.0 rw-(0)	ADC12BUSY r-(0)
ADC12CTL0 01A0h	MSC† rw-(0)	2.5V† rw-(0)	REFON† rw-(0)	ADC12ON† rw-(0)	ADC12OVIE rw-(0)	ADC12TOVIE rw-(0)	ENC rw-(0)	ADC12SC rw-(0)

†只能在 ENC=0 时修改。

表 A.7 看门狗定时器寄存器——字访问

Bit # —	15 … 8
看门狗定时器控制寄存器 WDTCTL 120h	读出为 069h 写入为 05Ah

Bit # —	7	6	5	4	3	2	1	0
看门狗定时器控制寄存器 WDTCTL 120h	HOLD rw-0	NMIES rw-0	NMI rw-0	TMSEL rw-0	CNTCL (w),r0	SSEL rw-0	IS1 rw-0	IS0 rw-0

表 A.8 FLASH 存储器控制寄存器——字访问

Bit # —	15 … 8
FCTL3 012Ch	读出为 096h 写入为 0A5h
FCTL2 012Ah	读出为 096h 写入为 0A5h
FCTL1 0128H	读出为 096h 写入为 0A5h

Bit # —	7	6	5	4	3	2	1	0
FCTL3 012Ch	Reserved r0	Reserved r0	EMEX rw-0	Lock rw-1	WAIT r-1	ACCVIFG rw-0	KEYV rw-(0)	Busy r(w)-0
FCTL2 012Ah	SSEL1 rw-0	SSEL0 rw-1	FN5 rw-0	FN4 rw-0	FN3 rw-0	FN2 rw-0	FN1 rw-1	FN0 rw-0
FCTL1 0128H	SEGWRT rw-0	WRT rw-0	Reserved r0	Reserved r0	Reserved r0	MEras rw-0	Erase rw-0	Reserved r0

表 A.10a Timer A 寄存器——字访问

Bit # —	15	14	13	12	11	10	9	8
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)
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)
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)
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)
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)
Timer_A 寄存器 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)
016Eh								
016Ch								
捕获/比较控制 CCTL4† 016Ah	CM41 rw-(0)	CM40 rw-(0)	CCIS41 rw-(0)	CCIS40 rw-(0)	SCS4 rw-(0)	SCCI4 rw-(0)	Unused r0	CAP4 rw-(0)
捕获/比较控制 CCTL3† 0168h	CM31 rw-(0)	CM30 rw-(0)	CCIS31 rw-(0)	CCIS30 rw-(0)	SCS3 rw-(0)	SCCI3 rw-(0)	Unused r0	CAP3 rw-(0)
捕获/比较控制 CCTL2 0166h	CM21 rw-(0)	CM20 rw-(0)	CCIS21 rw-(0)	CCIS20 rw-(0)	SCS2 rw-(0)	SCCI2 rw-(0)	Unused r0	CAP2 rw-(0)
捕获/比较控制 CCTL1 0164h	CM11 rw-(0)	CM10 rw-(0)	CCIS11 rw-(0)	CCIS10 rw-(0)	SCS1 rw-(0)	SCCI1 rw-(0)	Unused r0	CAP1 rw-(0)
捕获/比较控制 CCTL0 0162h	CM01 rw-(0)	CM00 rw-(0)	CCIS01 rw-(0)	CCIS00 rw-(0)	SCS0 rw-(0)	SCCI0 rw-(0)	Unused r0	CAP0 rw-(0)
Timer_A 控制 TACTL 0160h	Unused rw-(0)	Unused rw-(0)	Unused rw-(0)	Unused rw-(0)	Unused rw-(0)	SSEL2 rw-(0)	SSEL1 rw-(0)	SSEL0 rw-(0)

†在 Timer_A3 的芯片中保留的寄存器。

表 A.10b Timer A 寄存器——字访问

Bit #	7	6	5	4	3	2	1	0
017Eh								
017Ch								
捕获/比较寄存器 CCR4† 017Ah	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	2^7 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	2^7 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	2^7 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	2^7 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)
Timer_A 寄存器 TAR 0170h	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 CCTL4† 016Ah	OutMod42 rw-(0)	OutMod41 rw-(0)	OutMod40 rw-(0)	CCIE4 rw-(0)	CCI4 r	OUT4 rw-(0)	COV4 rw-(0)	CCIFG4 rw-(0)
捕获/比较控制 CCTL3† 0168h	OutMod32 rw-(0)	OutMod31 rw-(0)	OutMod30 rw-(0)	CCIE3 rw-(0)	CCI3 r	OUT3 rw-(0)	COV3 rw-(0)	CCIFG3 rw-(0)
捕获/比较控制 CCTL2 0166h	OutMod22 rw-(0)	OutMod21 rw-(0)	OutMod20 rw-(0)	CCIE2 rw-(0)	CCI2 r	OUT2 rw-(0)	COV2 rw-(0)	CCIFG2 rw-(0)
捕获/比较控制 CCTL1 0164h	OutMod12 rw-(0)	OutMod11 rw-(0)	OutMod10 rw-(0)	CCIE1 rw-(0)	CCI1 r	OUT1 rw-(0)	COV1 rw-(0)	CCIFG1 rw-(0)
捕获/比较控制 CCTL0 0162h	OutMod02 rw-(0)	OutMod01 rw-(0)	OutMod00 rw-(0)	CCIE0 rw-(0)	CCI0 r	OUT0 rw-(0)	COV0 rw-(0)	CCIFG0 rw-(0)
Timer_A 寄存器 TACTL 0160h	ID1 rw-(0)	ID0 rw-(0)	MC1 rw-(0)	MC0 rw-(0)	Unused rw-(0)	CLR rw-(0)	TAIE rw-(0)	TAIFG rw-(0)

†在 Timer_A3 的芯片中保留的寄存器。

Bit #	15	14	13	12	11	10	9	8
Timer_A 中断向量 TAIV 12Eh	0 r0	0 r0	0 r0	0 r0	0 r0	0 r0	0 r0	0 r0

Bit #	7	6	5	4	3	2	1	0
Timer_A 中断向量 TAIV 12Eh	0 r0	0 r0	0 r0	0 r0	0 r-(0)	TAIV r-(0)	r-(0)	0 r0

中断向量 TAIV，当芯片内为 Timer_A5(和个捕获/比较器)时如下：

0：无中断请求；

2：CCIFG1 标志置位；

4：CCIFG2 标志置位，CCIFG1－0；

6：CCIFG3 标志置位，CCIFG1＝CCIFG2－0；

8：CCIFG4 标志置位，CCIFG1＝CCIFG2＝CCIFG3＝0；

10：TAIFG 标志置位，CCIFG1－CCIFG2＝CCIFG3＝CCIFG4＝0。

中断向量 TAIV，当芯片内为 Timer_A3(3 个捕获/比较器)时如下：

0：无中断请求；

2：CCIFG1 标志置位；

4：CCIFG2 标志置位，CCIFG1＝0；

6：保留；

8：保留；

10：TAIFG 标志置位，CCIFG1＝CCIFG2＝0。

表 A.11a Timer B 寄存器——字访问

Bit # —	15	14	13	12	11	10	9	8
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)
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)
捕获/比较寄存器 CCR4† 019Ah	2^{15} rwv-(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)
捕获/比较寄存器 CCR3† 0198h	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)
捕获/比较寄存器 CCR2 0196h	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)
捕获/比较寄存器 CCR1 0194h	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)
捕获/比较寄存器 CCR0 0192h	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)
Timer_B 寄存器 TBR 0190h	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)
捕获/比较控制 CCTL6†,018Eh	CM61 rw-(0)	CM60 rw-(0)	CCIS61 rw-(0)	CCIS60 rw-(0)	SCS6 rw-(0)	CLLD6, 1 rw-(0)	CCLD6, 0 rw-(0)	CAP6 rw-(0)
捕获/比较控制 CCTL5†,018Ch	CM51 rw-(0)	CM50 rw-(0)	CCIS51 rw-(0)	CCIS50 rw-(0)	SCS5 rw-(0)	CLLD5, 1 rw-(0)	CCLD5, 0 rw-(0)	CAP5 rw-(0)
捕获/比较控制 CCTL4†,018Ah	CM41 rw-(0)	CM40 rw-(0)	CCIS41 rw-(0)	CCIS40 rw-(0)	SCS4 rw-(0)	CLLD4, 1 rw-(0)	CCLD4, 0 r0	CAP4 rw-(0)
捕获/比较控制 CCTL3†,0188h	CM31 rw-(0)	CM30 rw-(0)	CCIS31 rw-(0)	CCIS30 rw-(0)	SCS3 rw-(0)	CLLD3, 1 rw-(0)	CCLD3, 0 r0	CAP3 rw-(0)
捕获/比较控制 CCTL2,0186h	CM21 rw-(0)	CM20 rw-(0)	CCIS21 rw-(0)	CCIS20 rw-(0)	SCS2 rw-(0)	CCLD2, 1 rw-(0)	CCLD2, 0 r0	CAP2 rw-(0)
捕获/比较控制 CCTL1,0184h	CM11 rw-(0)	CM10 rw-(0)	CCIS11 rw-(0)	CCIS10 rw-(0)	SCS1 rw-(0)	CCLD1, 1 rw-(0)	CCLD1, 0 r0	CAP1 rw-(0)
捕获/比较控制 CCTL0,0182h	CM01 rw-(0)	CM00 rw-(0)	CCIS01 rw-(0)	CCIS00 rw-(0)	SCS0 rw-(0)	CCLD0, 1 rw-(0)	CCLD0, 0 r0	CAP0 rw-(0)
Timer_B 控制 TBCTL,0180h	Unused rw-(0)	TBCLGRP1 rw-(0)	TBCLGRP0 rw-(0)	TBCNTL1 rw-(0)	TBCNTL0 rw-(0)	Unused rw-(0)	TBSSEL1 rw-(0)	TBSSEL0 rw-(0)

† 在 Timer_B3 的芯片中保留的寄存器。

8：CCIFG4 标志置位，CCIFG1＝CCIFG2＝CCIFG3＝0；

10：CCIFG5 标志置位，CCIFG1＝CCIFG2－CCIFG3－CCIFG4－0；

12：CCIFG5 标志置位，CCIFG1＝CCIFG2＝CCIFG3－CCIFG4－CCIFG5＝0；

14：TAIFG 标志置位，CCIFG0－CCIFG2＝CCIFG3＝CCIFG4－CCIFG5＝CCIFG6＝0。

中断向量 TBIV，当芯片内为 Timer_B3(3 个捕获/比较器)时如下：

0：无中断请求；

2：CCIFG1 标志置位；

4：CCIFG2 标志置位，CCIFG1＝0；

6、8、10、12：保留；

14：TAIFG 标志置位，CCIFG1＝CCIFG2＝0。

附录 B　指令说明

MSP430 的 CPU 内核的结构来自于采用具有高度透明格式的 RISC 指令的设计思想。指令分为硬件实现的内核指令和利用这一硬件结构、有更高效率的模拟指令。模拟指令利用了内核指令和内装的常数发生器 CG1 和 CG2。本附录对内核指令和模拟指令作说明。在程序例程时采用模拟指令助记符。

根据指令的寻址方式，它在程序存储器中所占空间为 1～3 字。

每条指令至少用 1 个字。对于变址、符号、绝对和立即寻址方式需要外加占用 1 个字的程序存储器空间。这 4 种寻址方式适用于源操作数。变址、符号和绝对寻址适用于目的操作数。

这样，一条指令加上它的源操作数和目的操作数可能要用 1～3 字的程序存储器。

B.1　指令汇总

指令汇总如表 B.1 所示。

表 B.1　指令汇总

	指　令		参　数	简要说明	状态位(VNZC)
*	ADC[.W];	ADC.B	目的	目的＋C→目的	xxxx
	ADD[.W];	ADD.B	源，目的	源＋目的→目的	xxxx
	ADDC[.W];	ADDC.B	源，目的	源＋目的＋C→目的	xxxx
	AND[.W];	AND.B	源，目的	源 .and. 目的→目的	0xxx
	BIC[.W];	BIC.B	源，目的	.not. 源 .and. 目的→目的	----
	BIS[.W];	BIS.B	源，目的	源 .or. 目的→目的	----
	BIT[.W];	BIT.B	源，目的	源 .and. 目的	0xxx
*	BR		目的	转移到…	----
	CALL		目的	SP－2→SP PC＋2→堆栈，目的→PC	----
*	CLR[.W];	CLR.B	目的	清除目的数	----
*	CLRC			进位位 C 复位	----0
*	CLRN			负数位 N 复位	-0--
*	CLRZ			零位 Z 复位	--0-
	CMP[.W];	CMP.B	源，目的	目的－源	xxxx
*	DADC[.W];	DADC.B	目的	目的＋C→目的(十进制)	xxxx
	DADD[.W];	DADD.B	源，目的	源＋目的＋C→目的(十进制)	xxxx
*	DEC[.W];	DEC.B	目的	目的－1→目的	xxxx
*	DECD[.W];	DECD.B	目的	目的－2→目的	xxxx

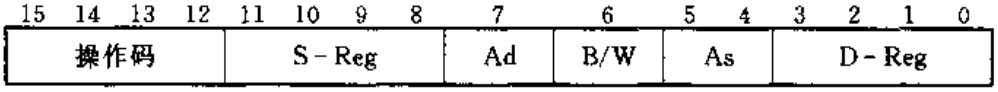
B.2 指令格式

双操作数指令(内核指令)

双操作数指令的 16 位代码分为 4 个字段:

- 操作码,4 Bit [OP - Code]
- 源操作数,6 Bit [S - Reg + As]
- 字节操作标志,1 Bit [B/W]
- 目的操作数,5 Bit [D - Reg + Ad]

源操作数字段由 2 个寻址位和 4 位寄存器号(0…15)组成。目的操作数字段由 1 个寻址位和 4 位寄存器号组成(0…15)。字节标志 B/W 指明指令是作为字节(B/W=1)还是字(B/W=0)来操作。



指 令		参 数	简要说明	状态位(VNZC)
ADD[. W];	ADD. B	源,目的	源+目的→目的	xxxx
ADDC[. W];	ADDC. B	源,目的	源+目的+C →目的	xxxx
AND[. W];	AND. B	源,目的	源 . and. 目的→目的	0xxx
BIC[. W];	BIC. B	源,目的	. not. 源 . and. 目的→目的	---
BIS[. W];	BIS. B	源,目的	源 . or. 目的→目的	---
BIT[. W];	BIT. B	源,目的	源 . and. 目的	0xxx
CMP[. W];	CMP. B	源,目的	目的—源	xxxx
DADD[. W];	DADD. B	源,目的	源+目的+C→目的(十进制)	xxxx
MOV[. W];	MOV. B	源,目的	源 →目的	----
SUB[. W];	SUB. B	源,目的	目的+ . not. 源+1→目的	xxxx
SUBC[. W];	SUBC. B	源,目的	目的+ . not. 源+C→目的	xxxx
XOR[. W];	XOR. B	源,目的	源 . xor. 目的→目的	xxxx

注意:操作时状态寄存器作为目的操作数

所有将状态寄存器作为目的操作数的操作,用运算结果来覆盖状态寄存器的内容,而指令操作对状态位额定影响不会发生。

例: ADD #3,SR ; 操作: (SR) + 3→SR

单操作数指令(内核指令)

单操作数指令的 16 位代码分为 3 个字段,即

- 操作码,9 Bit,高 4 位为"0001"
- 字节操作标志,1 Bit [B/W]
- 目的操作数,6 Bit [D - Reg + Ad]

目的操作数字段由 2 个寻址位和 4 位寄存器号(0…15)组成。目的操作数字段的位置与双操作数相同。字节标志 B/W 指明指令是作为字节(B/W=1)还是字(B/W=0)来操作。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	x	x	x	x	x	B/W	As		D/S-Reg			

指 令	参 数	简要说明	状态位(VNZC)
RRA[.W]; RRA.B	目的	MSB→MSB→…LSB→C	0xxx
RRC[.W]; RRC.B	目的	C→MSB→…LSB→C	xxxx
PUSH[.W]; PUSH.B	源	SP−2→SP,源→@SP	----
SWPB	目的	字节交换	——
CALL	目的	SP−2→SP PC+2→堆栈,目的→PC	----
RETI		栈顶→SR,SP+2→SP 栈顶 →PC,SP+2→SP	xxxx
SXT	目的	b7→b8→…b15	0xxx

条件跳转指令和无条件跳转指令(内核指令)

条件跳转指令和无条件跳转指令的 16 位代码分为 2 个字段,即

- 操作码,6 Bit。
- 跳转偏移量,10 Bit。

操作码字段由 3 位操作码和 3 位跳转条件组成。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	条件			10 位 PC 偏移量									

条件跳转的跳转范围为距当前地址的−512～+512 字范围内。由汇编器计算带符号的偏移量作为代码中的操作数。

指 令	参 数	简要说明	状态位(VNZC)
JC/JHS	Label	C=1 时跳转至 Label	——
JEQ/JZ	Label	Z=1 时跳转至 Label	——
JGE	Label	(N .XOR. V)=0 时跳转至 Label	——
JL	Label	(N .XOR. V)=1 时跳转至 Label	——
JMP	Label	无条件跳转至 Label	——
JN	Label	N=1 时跳转至 Label	——
JNC/JLO	Label	C=0 时跳转至 Label	——
JNE/JNZ	Label	Z=0 时跳转至 Label	——

注意:条件跳转与无条件跳转

条件跳转与无条件跳转指令不影响状态寄存器。

跳转指令发生时，用偏移量(offset)修改 PC

$$PCnew = PCold + 2 + 2 \times offset$$

跳转指令不发生时，程序继续执行后续指令。

B.3 不增加 ROM 开销的模拟指令

下述指令可以用 RISC 指令模拟，而且不增加 ROM 开销。汇编器接受这些助记符，并插入合适的内核指令。

注意：指令的模拟

模拟下述指令可能用到 R2 和 R3 的内容。

R2(CG1)含有立即数 2 和 4；R3(CG2)含有－1、0FFFFh、0、＋1 和＋2。读出数取决于寻址方式，汇编器根据所用的立即数设置适当的寻址位。

模拟指令的助记符

指令助记符	说 明	状态位(VNZC)	替代的模拟指令
算术指令			
ADC[.W] 目的	C 位加到目的操作数	* * * *	ADDC #0, 目的
ADC.B 目的	C 位加到目的操作数	* * * *	ADDC.B #0, 目的
DADC[.W] 目的	C 位十进制加到目的操作数	* * * *	DADD #0, 目的
DADC.B 目的	C 位十进制加到目的操作数	* * * *	DADD.B #0, 目的
DEC[.W] 目的	目的操作数减 1	* * * *	SUB #1, 目的
DEC.B 目的	目的操作数减 1	* * * *	SUB.B #1, 目的
DECD[.W] 目的	目的操作数减 2	* * * *	SUB #2, 目的
DECD.B 目的	目的操作数减 2	* * * *	SUB.B #2, 目的
INC[.W] 目的	目的操作数加 1	* * * *	ADD #1, 目的
INC.B 目的	目的操作数加 1	* * * *	ADD.B #1, 目的
INCD[.W] 目的	目的操作数加 2	* * * *	ADD #2, 目的
INCD.B 目的	目的操作数加 2	* * * *	ADD.B #2, 目的
SBC[.W] 目的	从目的操作数减去 C 位	* * * *	SUBC #0, 目的
SBC.B 目的	从目的操作数减去 C 位	* * * *	SUBC.B #0, 目的
逻辑指令			
INV[.W] 目的	目的操作数取反	* * * *	XOR #0FFFFh, 目的
INV.B 目的	目的操作数取反	* * * *	XOR.B #0FFFFh, 目的
RLA[.W] 目的	算术左循环	* * * *	ADD 目的, 目的

续

指令助记符	说 明	状态位(VNZC)	替代的模拟指令
逻辑指令			
RLA.B 目的	算术左循环	* * * *	ADDB 目的, 目的
RLC[.W] 目的	经 C 位算术左循环	* * * *	ADDC 目的, 目的
RLC.B 目的	经 C 位算术左循环	* * * *	ADDC.B 目的, 目的
数据指令(常用)			
CLR[.W]	清除目的操作数	- - - -	MOV #0, 目的
CLR.B	清除目的操作数	- - - -	MOV.B #0, 目的
CLRC	C 位复位	—0	BIC #1, SR
CLRN	N 位复位	-0--	BIC #4, SR
CLRZ	Z 位复位	—0-	BIC #2, SR
POP 目的	数据出栈	—	MOV @SP+, 目的
SETC	C 位置位	---1	BIS #1, SR
SETN	N 位置位	-1--	BIS #4, SR
SETZ	Z 位置位	—1-	BIS #2, SR
TST[.W] 目的	目的操作数测试	0 * * *	CMP #0, 目的
TST.B 目的	目的操作数测试	0 * * *	CMP.B #0, 目的
程序控制指令			
BR 目的	跳转	- - - -	MOV 目的, PC
DINT	关中断	- - - -	BIC #8, SR
EINT	开中断	- - - -	BIS #8, SR
NOP	空操作	- - - -	MOV #0h, #0h
RET	从子程序返回	- - - -	MOV @SP+, PC

B.4 指令说明(字母顺序)

本节分指令说明全部的内核指令和模拟指令,并作为解释和应用提示给出了一些例子。

带后缀.W 或无后缀的指令执行字操作。

带后缀.B 的指令执行字节操作。

*** ADC[.W] 进位位加入目的操作数**

*** ADC.B 进位位加入目的操作数**

语法 ADC 目的操作数 或 ADC.W 目的操作数

ADC.B 目的操作数

操作 目的操作数 + C→目的操作数

模拟　ADDC　#0，目的操作数
　　ADDC.B　#0，目的操作数

说明　将进位位 C 与目的操作数相加。目的操作数原值消失。

状态位　N：结果为负时置位，否则复位。
　　Z：结果为零时置位，否则复位。
　　C：目的操作数从 0FFFFh 增加至 0000 时置位，否则复位。
　　目的操作数从 0FFh 增加至 00 时置位，否则复位。
　　V：发生算术溢出时置位，否则复位。

模式位　OSCOff、CPUOff 和 GIE 不受影响。

例 R13 所指向的 16 位计数器加入 R12 所指向的 32 位计数器。

```
ADD     @R13,0(R12)          ; 加入低字
ADC     2(R12)               ; 进位位加入高字
```

例 R13 所指向的 8 位计数器加入 R12 所指向的 16 位计数器。

```
ADD.B   @R13,0(R12)          ; 加入低字节
ADC.B   1(R12)               ; 进位位加入高字节
```

ADD[.W]　源操作数加入目的操作数
ADD.B　源操作数加入目的操作数

语法　ADD　源操作数，目的操作数　或　ADD.W　源操作数，目的操作数
　　ADD.B　源操作数，目的操作数

操作　源操作数 ＋目的操作数→目的操作数

说明　源操作数加入目的操作数。源操作数不变。目的操作数原值消失。

状态位　N：结果为负时置位，否则复位。
　　Z：结果为零时置位，否则复位。
　　C：结果产生进位位时置位，否则复位。
　　V：发生算术溢出时置位，否则复位。

模式位　OSCOff、CPUOff 和 GIE 不受影响。

例 R5 加 10,发生进位位时跳转至 TONI。

```
ADD     #10,R5
JC      TONI                    ; 有进位位
…                               ; 无进位位
```

例 R5 加 10,发生进位位时跳转至 TONI。

```
ADD.B   #10,R5                  ; 10 加入 R5 的低字节
JC      TONI                    ; 如(R5)246,产生进位位,[0Ah+0F6h]
…                               ; 无进位位
```

ADDC[.W]　源操作数及进位位加入目的操作数
ADDC.B　源操作数及进位位加入目的操作数

语法　ADDC　源操作数,目的操作数　或　ADDC.W　源操作数,目的操作数
　　ADDC.B　源操作数,目的操作数

操作　源操作数 +目的操作数 + C→目的操作数

说明　源操作数及进位位加入目的操作数。源操作数不变,目的操作数原值消失。

状态位　N:　结果为负时置位,否则复位。
　　Z:　结果为零时置位,否则复位。
　　C:　结果产生进位位时置位,否则复位。
　　V:　发生算术溢出时置位,否则复位。

模式位　OSCOff、CPUOff 和 GIE 不受影响。

例 R13 所指的 32 位计数器加入 R13 加 11 字所指的 32 位计数器。

```
ADD     @R13+,20(R13)           ; 计数器低字相加,无进位位
ADDC    @R13+,20(R13)           ; 计数器高字相加,同时加入低字相加的进位位
…
```

例 R13 所指的 24 位计数器加入 R13 加 11 字所指的 24 位计数器。

```
ADD.B   @R13+,10(R13)           ; 计数器低字节相加,无进位位
ADDC.B  @R13+,10(R13)           ; 计数器中字节相加,同时加入低字节相加的进位位
ADDC.B  @R13+,10(R13)           ; 计数器高字节相加,同时加入中字节相加的进位位
…
```

AND[. W]　源操作数逻辑与目的操作数
AND. B　源操作数逻辑与目的操作数

语法　AND　源操作数,目的操作数　或　AND. W　源操作数,目的操作数
　　AND. B　源操作数,目的操作数

操作　源操作数 . AND. 目的操作数→目的操作数

说明　源操作数与目的操作数逻辑与,结果存入目的操作数。

状态位　N：最高位为 1 时置位,否则复位。
　　Z：结果为零时置位,否则复位。
　　C：结果非零时置位,否则复位。
　　V：复位。

模式位　OSCOff、CPUOff 和 GIE 不受影响。

例　以 R5 各位为屏蔽码(#0AA55h)和 TOM 相与,结果为零跳转至 TONI。

```
MOV     #0AA55h,R5          ; R5 装入屏蔽码
AND     R5,TOM              ; TOM 与屏蔽码相与
JZ      TONI                ;
…                           ; 结果非零
                            ;
                            ;
                            ; 或
                            ;
                            ;
AND     #0AA55h,TOM
JZ      TONI
```

例　屏蔽码(#0A5h)和 TOM 的低字节相与,结果为零跳转至 TONI。

```
AND. B  #0A5h,TOM           ; 用 0A5h 屏蔽 TOM 的低字节
JZ      TONI                ;
…                           ; 结果非零
```

BIC[. W]　目的操作数的位清除
BIC. B　目的操作数的位清除

语法　BIC　源操作数,目的操作数　或　BIC. W　源操作数,目的操作数
　　BIC. B　源操作数,目的操作数

操作　.NOT. 源操作数 .AND. 目的操作数→目的操作数

说明　源操作数反向后和目的操作数逻辑与，结果存入目的操作数。源操作数不变。

状态位　N：不变。
Z：不变。
C：不变。
V：不变。

模式位　OSCOff、CPUOff 和 GIE 不受影响。

例　RAM 中字 LEO 的最高 6 位清除。

```
BIC     #0FC00h,LEO                    ; 清除 LEO 的最高 6 位
```

例　RAM 中字节 LEO 的最高 5 位清除。

```
BIC.B   #0F8h,LEO                      ; 清除 LEO 的最高 5 位
```

例　端口引脚 P0.0 和 P0.1 的清除。

```
P0OUT   .equ    011h                   ; 端口 P0 地址定义
P0_0    .equ    01h
P0_1    .equ    02h
BIC.B   #P0_0+P0_1,&P0OUT              ; P0.0 和 P0.1 清除为零
```

BIS[.W]　目的操作数的位置位
BIS.B　目的操作数的位置位

语法　BIS　源操作数，目的操作数　或　BIS.W　源操作数，目的操作数
BIS.B　源操作数，目的操作数

操作　源操作数 .OR. 目的操作数→目的操作数

说明　源操作数和目的操作数逻辑或，结果存入目的操作数，源操作数不变。

状态位　N：不变。
Z：不变。
C：不变。
V：不变。

模式位　OSCOff、CPUOff 和 GIE 不受影响。

例　RAM 字 TOM 的最低 6 位置位。

```
BIS      #003Fh,TOM                    ; TOM 的最低 6 位置位
```

例　启动 A/D 转换。

```
ASOC     .equ    1                     ; 转换启动位
ACTL     .equ    114h                  ; ADC 控制寄存器
         BIS     #ASOC,&ACTL           ; 启动 A/D 转换
```

例　RAM 字 TOM 的最高 3 位置位。

```
BIS.B    #0E0h,TOM                     ; TOM 的最高 3 位置位
```

例　端口引脚 P0.0 和 P0.1 的置位。

```
P0OUT    .equ    011h
P0       .equ    01h
P1       .equ    02h
         BIS.B   #P0+P1,&P0OUT
```

BIT[.W]　　目的操作数的位测试
BIT.B　　目的操作数的位测试

语法　BIT　源操作数,目的操作数　或　BIT.W　源操作数,目的操作数

操作　源操作数 .AND. 目的操作数

说明　源操作数和目的操作数逻辑与,结果只影响状态位,源操作数、目的操作数不变。

状态位　N：最高位为 1 时置位,否则复位。
　　Z：结果为零时置位,否则复位。
　　C：结果非零时置位,否则复位。
　　V：复位。

模式位　OSCOff、CPUOff 和 GIE 不受影响。

例　如 R8.9=1，程序跳转至 TOM。

```
BIT      #0200h,R8                     ; R8 的 Bit9 是 1 ?
JNZ      TOM                           ; 是,跳转至 TOM
…                                      ; 否,继续执行
```

例 检查选中的 A/D 通道。

```
ACTL    .equ    114h                ; ADC 控制寄存器
BIT     #4,&ACTL                    ; 通道 0 选中?
JNZ     END                         ; 是,跳转至 END
```

例 如 R8.3=1,程序跳转至 TOM。

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

例 测试串行通信接收位 RCV。结果入进位位。后续指令将它移入 RECBUF。

```
; 串行通信,低位先发
BIT.B   #RCV,RCCTL                  ; 测试的位信息进入进位位
RRC     RECBUF                      ; 进位位进入 RECBUF 最高位
…                                   ; 以上两条指令重复 8 次
; 串行通信,高位先发
BIT.B   #RCV,RCCTL                  ; 测试的位信息进入进位位
RLC.B   RECBUF                      ; 进位位进入 RECBUF 最低位
…                                   ; 以上两条指令重复 8 次
```

*** BR　　转移至目的操作数**

语法　　BR　　目的操作数

操作　　目的操作数→PC

模拟　　MOV　目的操作数,PC

说明　　在 64 K 地址空间发生无条件转移。可用所有寻址方式。BR 为字指令。

状态位　状态位不变。

例 以下列举各种寻址方式。

```
BR  #EXEC                           ; 转移至标号 EXEC 或直接地址(如:#0A4h)
                                    ; 内核指令: MOV     @PC+,PC
BR  EXEC                            ; 转移至 EXEC 所指地址
                                    ; 内核指令: MOV     X(PC),PC
                                    ; 间接寻址
BR  &EXEC                           ; 转移至绝对地址 EXEC 所指地址
                                    ; 内核指令: MOV     X(0),PC
                                    ; 间接寻址
```

操作 0→C

模拟 BIC #1,SR

说明 进位位 C 被清除。CLRC 指令是字指令。

状态位 N：不变。
Z：不变。
C：0。
V：不变。

模式位 OSCOff、CPUOff 和 GIE 不受影响。

例 R13 所指 16 位十进制计数器与 R12 所指 32 位计数器相加。

```
CLRC                    ;C=0,初始化
DADD    @R13,0(R12)     ;16 位计数器与 32 位计数器的低字相加
DADC    2(R12)          ;进位位与 32 位计数器高字相加
```

*** CLRN 清除负数位**

语法 CLRN

操作 0 N

模拟 BIC #4,SR

说明 常数 0004h 取反(0FFFBh)和 SR 逻辑与。结果存入 SR。CLRN 是字指令。

状态位 N：0。
Z：不变。
C：不变。
V：不变。

模式位 OSCOff、CPUOff 和 GIE 不受影响。

例 SR 中的 N 位清除，以回避被调用子程序对负数的特殊处理。

```
CLRN
CALL    SUBR
...
```

```
            ...
SUBR        JN      SUBRET      ; 如输入为负,立即返回主程序
            ...
            ...
            ...
SUBRET      RET
```

*** CLRZ　　清除零位**

语法　CLRZ

操作　0　　Z

模拟　BIC　　#2,SR

说明　常数 0002h 取反(0FFFDh)和 SR 逻辑与。结果存入 SR。CLRZ 是字指令。

状态位　N：不变。
Z：　0。
C：不变。
V：不变。

模式位　OSCOff、CPUOff 和 GIE 不受影响。

例　清除 SR 中的零位。

```
CLRZ
```

CMP[.W]　　比较源操作数和目的操作数
CMP.B　　比较源操作数和目的操作数

语法　CMP　　源操作数,目的操作数　　或　　CMP.W　　源操作数,目的操作数
CMP.B　源操作数,目的操作数

操作　目的操作数 + .NOT.源操作数 + 1 或目的操作数－源操作数

说明　源操作数减去目的操作数。以源操作数的补码相加实现。两个操作数均不变,结果不保存,只影响状态位。

状态位　N：结果为负时置位,否则复位(即:源操作数≥目的操作数)。

模拟　DADD　#0,目的操作数
　　DADD.B　#0,目的操作数

说明　进位位 C 以十进制加入目的操作数。

状态位　N：MSB=1 时置位,否则复位。
　　Z：目的操作数=0 时置位,否则复位。
　　C：目的操作数由 9999 增加到 0000 时置位,否则复位。
　　目的操作数由 99 增加到 00 时置位,否则复位。
　　V：不变。

模式位　OSCOff、CPUOff 和 GIE 不受影响。

例　R5 的 4 位十进制数与 R8 的 8 位十进制数相加。

```
CLRC                    ;清除进位位,初始化
DADD    R5,0(R8)        ;低字及进位位相加
DADC    2(R8)           ;进位位加入高字
```

例　R5 的 2 位十进制数与 R8 的 4 位十进制数相加。

```
CLRC                    ;清除进位位,初始化
DADD.B  R5,0(R8)        ;低字节及进位位相加
DADC.B  1(R8)           ;进位位加入高字节
```

DADD[.W]　源操作数及进位位以十进制加入目的操作数
DADD.B　源操作数及进位位以十进制加入目的操作数

语法　DADD　源操作数,目的操作数　或　DADD.W　源操作数,目的操作数
　　DADD.B　源操作数,目的操作数

操作　源操作数 + 目的操作数 + C→目的操作数（十进制）

说明　源操作数和目的操作数以 BCD 码十进制数处理。源操作数和进位位加入目的操作数。源操作数不变,目的操作数原值消失。结果对于非 BCD 码数无定义。

状态位　N：MSB=1 时置位,否则复位。
　　Z：结果为零时置位,否则复位。
　　C：结果大于 9999 时置位。
　　结果大于 99 时置位。
　　V：不变。

例 R10 减 1。

```
DEC             R10                     ; R10 减 1
; 移动 EDE 开始 255 字节数据块到 TONI。TONI 不能在 EDE~EDE+0FEh 范围内以防覆盖
                MOV     #EDE,R6
                MOV     #255,R10
L$1             MOV.B   @R6+,TONI-EDE-1(R6)
                DEC     R10
                JNZ     L$1
```

例 LEO 地址的字节减 1。

```
DEC.B           LEO                     ; MEM(LEO)减 1
; 移动 EDE 开始 255 字节数据块到 TONI。TONI 不能在 EDE~EDE+0FEh 范围内以防覆盖
                MOV     #EDE,R6
                MOV.B   #255,LEO
L$1             MOV.B   @R6+,TONI-EDE-1(R6)
                DEC.B   LEO
                JNZ     L$1
```

*** DECD[.W]　目的操作数减 2**

*** DECD.B　目的操作数减 2**

语法　DECD　目的操作数　或　DECD.W　目的操作数
　　DECD.B　目的操作数

操作　目的操作数 － 2→目的操作数

模拟　SUB　#2,目的操作数
模拟　SUB.B　#2,目的操作数

说明　目的操作数减 2。原值消失。

状态位　N：结果为负时置位，否则复位。
　　Z：目的操作数初值为 2 时置位，否则复位。
　　C：目的操作数初值为 0 或 1 时复位，否则置位。
　　V：发生算术溢出时置位，否则复位。
　　目的操作数初值为 08001h 或 08000h 时置位，否则复位。
　　目的操作数初值为 081 或 080h 时置位，否则复位。

模式位　OSCOff、CPUOff 和 GIE 不受影响。

例 R10 减 2。

```
DECD        R10                    ; R10 减去 2
; 移动 EDE 开始 255 字数据块到 TONI。TONI 不能在 EDE～EDE+1FCh 范围内以防覆盖
            MOV     #EDE,R6
            MOV     #510,R10
L$1         MOV     @R6+,TONI-EDE-2(R6)
            DECD    R10
            JNZ     L$1
```

例 LEO 地址的字节减 2。

```
DECD.B      LEO                    ; MEM(LEO)减去 2
```

状态字节 STATUS 减 2。

```
DECD.B      STATUS
```

*** DINT 关中断**

语法 DINT

操作 0 GIE

模拟 BIC #8,SR

说明 关闭全部中断。
常数 08h 取反和 SR 逻辑与,结果存入 SR。

状态位 N：不变。
Z：不变。
C：不变。
V：不变。

模式位 OSCOff、CPUOff 不受影响,GIE 复位。

例 清除通用中断允许位 GIE,确保 32 位计数器数据传输不打断,以免传输期间因中断使计数器发生改变。

```
DINT                               ; GIE 位控制的所有中断被禁止
MOV         COUNTHI,R5             ; Copy counter
MOV         COUNTLO,R6
EINT                               ; GIE 位控制的所有中断被允许
```

注意:关中断

当执行 DINT 指令时发生中断请求,关中断指令后续的指令将执行。任何程序段要避免中断影响,DINT 指令的执行必须提前程序段至少一条指令。

*** EINT　　开中断**

语法　EINT

操作　1　GIE

模拟　BIS　#8,SR

说明　允许全部中断。

常数 08h 和 SR 逻辑或,结果存入 SR。

状态位　N:不变。

Z:不变。

C:不变。

V:不变。

模式位　OSCOff、CPUOff 不受影响,GIE 置位。

例　SR 中的 GIE 位置位。

```
; P0.2~P0.7 的中断服务程序
; 中断处于系统的最低级
; P0IN 是端口读寄存器地址,P0IFG 是中断事件锁存寄存器地址
;
            PUSH.B  &P0IN
            BIC.B   @SP,&P0IFG          ; 复位已被接受的中断标志
                                        ; 原有 P0 中断标志保存在堆栈
            EINT                        ; 允许其他中断
            BIT     #Mask,@SP
            JEQ     MaskOK              ; 由 MASK 识别标志,跳转
            ...
MaskOK      BIC     #Mask,@SP
            ...
            INCD    SP                  ; 将 SP 调整到中断程序开始位置
            RETI
```

注意:开中断

EINT 指令的后续指令总是要执行的,甚至有中断服务请求挂起。

*** INC[.W] 目的操作数加 1**
*** INC.B 目的操作数加 1**

语法 INC 目的操作数 或 INC.W 目的操作数
INC.B 目的操作数

操作 目的操作数 + 1 →目的操作数

模拟 ADD #1,目的操作数

说明 目的操作数增加 1,原值消失。

状态位 N:结果为负时置位,否则复位。
Z:目的操作数初值为 0FFFFh 时置位,否则复位。
目的操作数初值为 0FFh 时置位,否则复位。
C:目的操作数初值为 0FFFFh 时置位,否则复位。
目的操作数初值为 0FFh 时置位,否则复位。
V:目的操作数初值为 07FFFh 时置位,否则复位。
目的操作数初值为 07Fh 时置位,否则复位。

模式位 OSCOff、CPUOff 和 GIE 不受影响。

例 取消软件堆栈(非系统堆栈)顶的字节数据。

```
SSP     .EQU    R4
                                ;
        INC     SSP             ; 用增加指令取消栈顶项 TOSS (top of SW stack,
                                ; SSP 是字寄存器,不能用 INC.B
```

例 处理过程的状态字节 STATUS 增加。增至 11 时跳转至 OVFL。

```
INC.B   STATUS
CMP.B   #11,STATUS
JEQ     OVFL
```

*** INCD[.W] 目的操作数加 2**
*** INCD.B 目的操作数加 2**

语法 INCD 目的操作数 或 INCD.W 目的操作数
INCD.B 目的操作数

状态位 N：结果为负时置位，否则复位。
Z：目的操作数初值为 0FFFFh 时置位，否则复位。
目的操作数初值为 0FFh 时置位，否则复位。
C：结果非零时置位，否则复位。
V：目的操作数初值为负时置位，否则复位。

模式位 OSCOff、CPUOff 和 GIE 不受影响。

例 R5 取补(2 的补码)。

```
MOV     #00AEh,R5          ; R5 = 000AEh
INV     R5                 ; R5 = 0FF51h
INC     R5                 ; R5 = 0FF52h,原值的补码
```

例 存储器字节 LEO 取补。

```
MOV.B   #0AEh,LEO          ; MEM(LEO) = 0AEh
INV.B   LEO                ; MEM(LEO) = 051h
INC.B   LEO                ; MEM(LEO) = 052h,原值的补码
```

JC　　有进位位时跳转
JHS　　无符号数大于等于时跳转

语法 JC label
JHS label

操作 如果 C = 1：PC + 2 ×偏移量→ PC
如果 C = 0：继续执行后续指令

说明 检查进位位。如果 C=1，指令低 10 位所含带符号偏移量乘 2 后加入 PC。如果 C=0，执行跳转指令的后续指令。JC/JHE 指令用于无符号数(0～65 535)的比较。

状态位 状态位不受影响。

例 P0IN.1 的信号用于定义或控制程序流向。

```
BIT     #10h,&P0IN         ; 信号状态→进位位
JC      PROGA              ; 进位位=1,跳转至 PROGA
…                          ; 进位位=0,继续执行程序
```

例 R5 与 15 比较。当大于等于时跳转至 LABEL。

```
CMP     #15,R5
```

```
JHS     LABEL                    ; R5  15,跳转
...                              ; R5  15,继续执行
```

JEQ 相等时跳转
JZ 为零时跳转

语法 JEQ label
JZ label

操作 如果 Z = 1：PC + 2 ×偏移量→ PC
如果 Z = 0：继续执行后续指令

说明 检查零位。如果 Z=1,指令低 10 位所含带符号偏移量乘 2 后加入 PC。如果 Z=0,执行跳转指令的后续指令。

状态位 状态位不受影响。

例 如 R7=0,跳转至 TONI。

```
TST     R7                       ; 测试 R7
JZ      TONI                     ; Z=1:跳转
```

例 如 R6 与表中数据相等,跳转至 LEO。

```
CMP     R6,Table(R5)             ; R6 与 MEM(Table 地址+ R5)比较
JEQ     LEO                      ; 比较相等,跳转
...                              ; 不相等,继续执行
```

JGE 有符号数大于等于时跳转

语法 JGE label

操作 如果 (N .XOR. V) = 0：PC + 2×偏移量→ PC
如果 (N .XOR. V) = 1：继续执行后续指令

说明 检查负数位 N 和溢出位 V。如果 N、V 同时置位或复位,指令低 10 位所含带符号偏移量乘 2 后加入 PC。如果只有一个置位,执行跳转指令的后续指令。JGE 指令用于有符号整数的比较。

状态位 状态位不受影响。

例 当 R6 所含数据大于等于 R7 所指存储器地址的数据，程序跳转至 EDE。

```
CMP     @R7,R6          ; R6    (R7)?，有符号数比较
JGE     EDE             ; 是，R6    (R7)
…                       ; 否，继续执行
…
…
```

JL　　有符号数小于时跳转

语法　JL　　label

操作　如果（N .XOR. V）= 1：PC + 2 ×偏移量→ PC
　　如果（N .XOR. V）= 0：继续执行后续指令

说明　检查负数位 N 和溢出位 V。如果只有一个置位，指令低 10 位所含带符号偏移量乘 2 后加入 PC。如果 N、V 同时置位或复位，执行跳转指令的后续指令。JL 指令用于有符号整数的比较。

状态位　状态位不受影响。

例 当 R6 所含数据小于 R7 所指存储器地址的数据，程序跳转至 EDE。

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

JMP　　无条件跳转

语法　JMP　　label

操作　PC + 2 ×偏移量→ PC

说明　指令低 10 位所含带符号偏移量乘 2 后加入 PC。

状态位　状态位不受影响。

提示　可以用这一单字指令取代 BR 指令，用于实现相对于 PC 在－511～＋512 字范围内的程序跳转。

JN　　负数时跳转

语法　JN　　label

操作　如果 N = 1：PC + 2 ×偏移量→ PC
　　如果 N = 0：继续执行后续指令

说明　检查负数位。如果 N=1，指令低 10 位所含带符号偏移量乘 2 后加入 PC。如果 N=0，执行跳转指令的后续指令。

状态位　状态位不受影响。

例　COUNT 与 R5 相减，如果结果为负数，程序跳转至 L$1 并清除 COUNT。

```
        SUB     R5,COUNT        ; COUNT - R5→ COUNT
        JN      L$1             ; 如果 COUNT <0,PC=L$1,COUNT 清除
        ...                     ; COUNT≥ 0
        ...
        ...
        ...
L$1     CLR     COUNT
        ...
        ...
        ...
```

JNC　　无进位位时跳转
JLO　　无符号数小于时跳转

语法　JNC　　label
　　JLO　　label

操作　如果 C = 0：PC + 2 ×偏移量→ PC
　　如果 C = 1：继续执行后续指令

说明　检查进位位。如果 C=0，指令低 10 位所含带符号偏移量乘 2 后加入 PC。如果 C=1，执行跳转指令的后续指令。JNC/JLO 指令用于无符号数(0～65 535)的比较。

状态位　状态位不受影响。

例　R6 与 BUFFER 相加。如果发生溢出，跳转至 ERROR 执行差错处理程序。

```
        ADD     R6,BUFFER       ; BUFFER + R6 → BUFFER
        JNC     CONT            ; C=0,跳转至 CONT
ERROR   …                       ; C=1,差错处理程序
        …
        …
        …
CONT    …                       ; 正常程序继续执行
        …
        …
```

例　如状态字节含有 1 或 0,跳转至 STL2。

```
CMP.B   #2,STATUS
JLO     STL2                    ; STATUS < 2
…                               ; STATUS ≥2,继续执行
```

JNE　　不等时跳转
JNZ　　非零时跳转

语法　JNE　label
　　JNZ　label

操作　如果 Z = 0: PC + 2 ×偏移量→ PC
　　如果 Z = 1: 继续执行后续指令

说明　检查零位。如果 Z=0,指令低 10 位所含带符号偏移量乘 2 后加入 PC。如果 Z=1,执行跳转指令的后续指令。

状态位　状态位不受影响。

例　R7 与 R8 不同时跳转至 TONI。

```
CMP     R7,R8                   ; 比较 R7 与 R8
JNE     TONI                    ; 不同,跳转
…                               ; 相同,继续执行
```

MOV[.W]　　移动源操作数到目的操作数
MOV.B　　移动源操作数到目的操作数

语法　MOV　源操作数,目的操作数　或　MOV.W　源操作数,目的操作数
　　MOV.B　源操作数,目的操作数

操作　源操作数 →目的操作数

说明　源操作数移动到目的操作数。源操作数不变，目的操作数的原值消失。

状态位　状态位不受影响。

模式位　OSCOff、CPUOff 和 GIE 不受影响。

例　表 EDE 的字数据复制到表 TOM。表长度为 020h。

```
        MOV     #EDE,R10                ; 准备指针
        MOV     #020h,R9                ; 准备计数器
Loop    MOV     @R10+,TOM-EDE-2(R10)    ; R10 作为两表的指针
        DEC     R9                      ; 计数器减 1
        JNZ     Loop                    ; 计数器不等于 0,继续复制
        ...                             ; 复制完成
        ...
        ...
```

例　表 EDE 所含字节数据复制到表 TOM。表长度为 020h。

```
        MOV     #EDE,R10                ; 准备指针
        MOV     #020h,R9                ; 准备计数器
Loop    MOV.B   @R10+,TOM-EDE-1(R10)    ; R10 作为两表的指针
        DEC     R9                      ; 计数器减 1
        JNZ     Loop                    ; 计数器不等于 0,继续复制
        ...                             ; 复制完成
        ...
        ...
```

*** NOP**　　**空操作**

语法　NOP

操作　无操作

模拟　MOV　　#0,#0

说明　不执行操作。NOP 指令用于软件检查期间或定义等待时间。

状态位　状态位不受影响。

NOP 指令主要用于两个目的：占据 1、2 或是个存储器字；调整软件时序。

注意：其他可以模拟空操作的指令

还有其他一些指令可以用不同的周期和代码字空间来模拟空操作。

```
例：MOV 0(R4),0(R4)      ; 6 cycles, 3 words
    MOV @R4,0(R4)        ; 5 cycles, 2 words
    BIC #0,EDE(R4)       ; 4 cycles, 2 words
    JMP $+2              ; 2 cycles, 1 word
    BIC #0,R5            ; 1 cycles, 1 word
```

*** POP[.W]　　从堆栈推出字到目的操作数**

*** POP.B　　从堆栈推出字节到目的操作数**

语法　　POP　　目的操作数

　　POP.B　　目的操作数

操作　　@SP →目的操作数

　　SP + 2 → SP

模拟　　MOV　　@SP+，目的操作数　　或　　MOV.W　　@SP+，目的操作数

模拟　　MOV.B　　@SP+，目的操作数

说明　　栈顶项(TOS)移动到目的操作数。SP 增加 2。

状态位　　状态位不受影响。

例　R7 和 SR 从堆栈恢复。

```
POP     R7                  ; 恢复 R7
POP     SR                  ; 恢复 SR
```

例　从堆栈恢复 RAM 字节 LEO 的数据。

```
POP.B   LEO                 ; 栈顶项的低字节移动到 LEO
```

例　R7 从堆栈恢复。

```
POP.B   R7                  ; 栈顶项的低字节移动到 R7，R7 额定高字节为 00
```

例　R7 所指存储器及 SR 从堆栈恢复。

```
POP.B    0(R7)                    ; 栈顶项的低字节移动到 R7 所指地址
                                  ; 例:R7 = 203h,Mem(R7) = 栈顶项的低字节
                                  ; 例:R7 = 20Ah,Mem(R7) = 栈顶项的低字节
POP      SR
```

注意:系统堆栈指针

系统堆栈指针增加时总是 2,而与指令的字节后缀无关。这样做的原因是因为 SP 不是仅仅用于 POP 指令,也用于 RETI 指令。

PUSH[.W]　字压入堆栈
PUSH.B　字节压入堆栈

语法　PUSH　源操作数　或　PUSH.W　源操作数
　　PUSH.B　源操作数

操作　SP − 2　SP
　　源操作数　@SP

说明　SP 减少 2,源操作数移动到 SP 所指 RAM 字地址(TOS)。

状态位　状态位不受影响。

模式位　OSCOff、CPUOff 和 GIE 不受影响。

例 SR 及 R8 保存入堆栈。

```
PUSH     SR                       ; 保存 SR
PUSH     R8                       ; 保存 R8
```

例 外围模块寄存器 TCDAT 的内容保存入堆栈。

```
PUSH.B &TCDAT                     ; 保存 8 位外围模块地址 TCDAT 到堆栈
```

注意:系统堆栈指针

系统堆栈指针减少时总是 2,而与指令的字节后缀无关。这样做的原因是因为 SP 不是仅仅用于 PUSH 指令,也用于中断服务程序。

*** RET　　从子程序返回**

语法　RET

操作　@SP　PC
SP + 2　SP

模拟　MOV　@SP+,PC

说明　将由 CALL 指令压入堆栈的返回地址移动到 PC。程序从 CALL 指令的后续地址继续执行。

状态位　状态位不受影响。

RETI　　从中断服务程序返回

语法　RETI

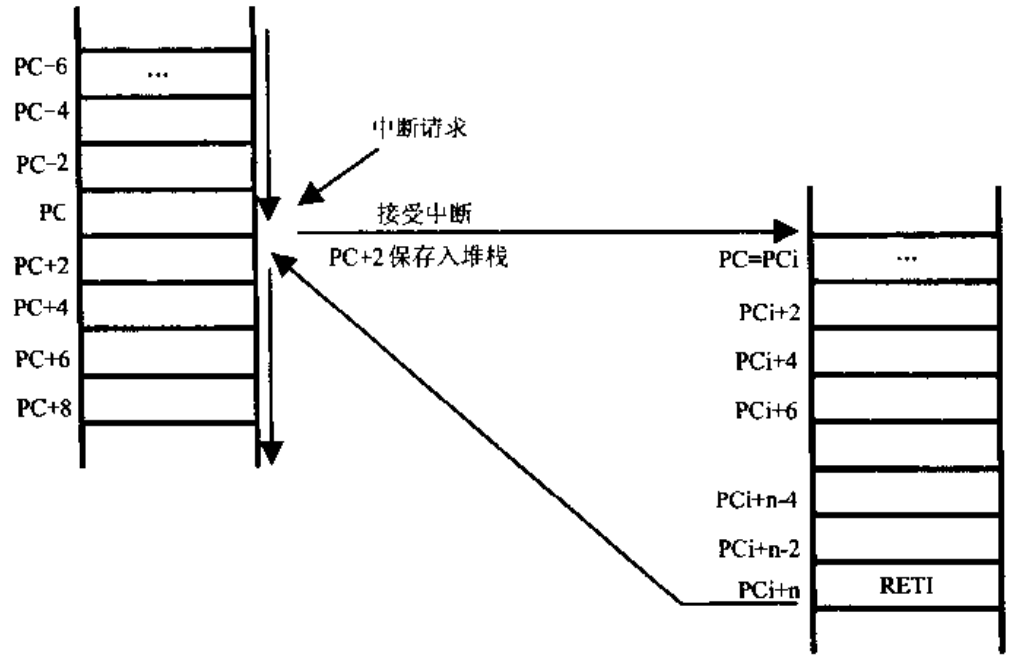
操作　TOS　SR
SP + 2　SP
TOS　PC
SP + 2　SP

说明　1. 用栈顶项取代 SR,使它恢复为中断开始时的数值,SP 增加 2。
2. 用栈顶项取代 PC,使 PC 恢复成中断开始时的地址,SP 增加 2。

状态位　N：从系统堆栈恢复。
Z：从系统堆栈恢复。
C：从系统堆栈恢复。
V：从系统堆栈恢复。

模式位　OSCOff、CPUOff 和 GIE 从系统堆栈恢复。

例　主程序被中断。



*** RLA[.W]　　算术左移**

*** RLA.B　　算术左移**

语法　RLA　目的操作数　或　RLA.W　目的操作数

RLA.B　目的操作数

操作　C ← MSB ← MSB−1 … LSB+1 ← LSB ← 0

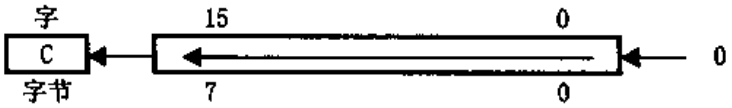
模拟　ADD　目的操作数,目的操作数

ADD.B　目的操作数,目的操作数

说明　目的操作数左移一位。MSB 移入进位位,LSB 填 0。RLA 指令相当于有符号数乘 2。

当目的操作数初值　04000h 且< 0C000h 时发生溢出,因为结果符号发生改变。

当目的操作数初值　040h 且< 0C0h 时发生溢出,因为结果符号发生改变。



状态位　N:结果为负数时置位,否则复位。

Z:结果为零时置位,否则复位。

C:装入 MSB 内容。

V:发生算术溢出时置位,即:

04000h　目的操作数初值 < 0C000h　或 040h　目的操作数初值 < 0C0h

否则复位

模式位 OSCOff、CPUOff 和 GIE 不受影响。

例 R7 乘 4。

```
RLA     R7          ; R7 左移 (× 2),模拟指令 ADD   R7,R7
RLA     R7          ; R7 左移 (× 4),模拟指令 ADD   R7,R7
```

例 R7 低字节乘 4。

```
RLA.B   R7          ; R7 低字节左移 (× 2),模拟指令 ADD.B  R7,
                    ;R7
RLA.B   R7          ; R7 低字节左移 (× 4),模拟指令 ADD.B  R7,
                    ;R7
```

注意:RLA 替代

汇编器不能识别指令:

```
RLA     @R5+        或      RLA.B   @R5+
```

必须用以下指令替代:

```
ADD     @R5+,-2(R5) 或      ADD.B   @R5+,-1(R5)
```

*** RLC[.W] 经进位位左移**

*** RLC.B 经进位位左移**

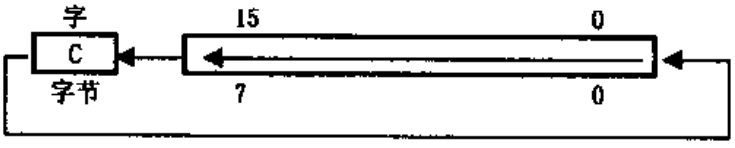
语法 RLC 目的操作数 或 RLC.W 目的操作数

RLC.B 目的操作数

操作 C ← MSB ← MSB−1 … LSB+1 ← LSB ← C

模拟 ADDC 目的操作数,目的操作数

说明 目的操作数左移一位。进位位移入 LSB,MSB 移入进位位。



状态位 N:结果为负数时置位,否则复位。

Z:结果为零时置位,否则复位。

C:装入 MSB 内容。

V:发生算术溢出时置位,即:

04000h 目的操作数初值 < 0C000h 或 040h 目的操作数初值 < 0C0h

否则复位。

```
RRA     R5                  ; R5/2 → R5
; R5 的值× 0.75 (0.5 + 0.25)
PUSH    R5                  ; 暂存 R5 入堆栈
RRA     R5                  ; R5 × 0.5→ R5
ADD     @SP+,R5             ; R5 × 0.5 + R5 = 1.5 × R5 → R5
RRA     R5                  ; (1.5 × R5) × 0.5 = 0.75 × R5 → R5
; 或
RRA     R5                  ; R5 × 0.5 → R5
PUSH    R5                  ; R5 × 0.5 → TOS
RRA     @SP                 ; TOS× 0.5 = 0.5 × R5 × 0.5 = 0.25 × R5 → TOS
ADD     @SP+,R5             ; R5 × 0.5 + R5 × 0.25 = 0.75 × R5 → R5
```

例 R5 低字节右移一位。MSB 保持原值。操作相当于算术除以 2。

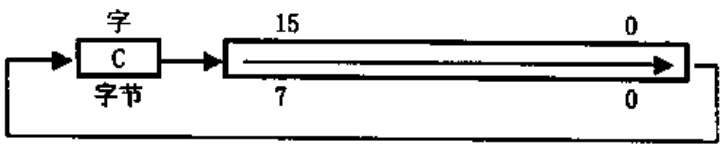
```
RRA.B   R5                  ; R5/2 → R5；只对低字节操作,R5 高字节为"0"
; R5 低字节的值 × 0.75 (0.5 + 0.25)
PUSH.B  R5                  ; 暂存 R5 低字节入堆栈
RRA.B   R5                  ; R5 × 0.5 → R5
ADD.B   @SP+,R5             ; R5 × 0.5 + R5 = 1.5 × R5 → R5
RRA.B   R5                  ; (1.5 × R5) × 0.5 = 0.75 × R5 → R5
; 或
RRA.B   R5                  ; R5 × 0.5 → R5
PUSH.B  R5                  ; R5 × 0.5 → TOS
RRA.B   @SP                 ; TOS × 0.5 = 0.5 × R5 × 0.5 = 0.25× R5→ TOS
ADD.B   @SP+,R5             ; R5 × 0.5 + R5 × 0.25 = 0.75 × R5 → R5
```

RRC[.W]　　经进位位右移
RRC.B　　经进位位右移

语法　RRC　目的操作数　或　RRC.W　目的操作数
　　RRC　目的操作数

操作　C → MSB → MSB−1 … LSB+1→ LSB → C

说明　目的操作数右移一位。进位位移入 MSB,LSB 移入进位位。



状态位　N：结果为负数时置位,否则复位。
　　Z：结果为零时置位,否则复位。
　　C：装入 LSB 内容。

V：目的操作数初值为正且 C=1 时置位，否则复位。

模式位　OSCOff、CPUOff 和 GIE 不受影响。

例　R5 右移一位。MSB 装入 1。

```
SETC                    ; 为 MSB 准备进位位
RRC     R5              ; R5/2 + 8000h → R5
```

例　R5 右移一位。MSB 装入 1。

```
SETC                    ; 为 MSB 准备进位位
RRC.B   R5              ; R5/2 + 80h → R5，只对 R5 的低字节操作
```

*** SBC[.W]　目的操作数减借位**

*** SBC.B　目的操作数减借位**

语法　SBC　目的操作数　或　SBC.W　目的操作数

SBC.B　目的操作数

操作　目的操作数 + 0FFFFh + C →目的操作数

目的操作数 + 0FFh + C →目的操作数

模拟　SUBC　#0，目的操作数

SUBC.B　#0，目的操作数

说明　进位位与目的操作数减 1 相加。目的操作数原值消失。

状态位　N：结果为负数时置位，否则复位。

Z：结果为零时置位，否则复位。

C：目的操作数从 0000 减至 0FFFFh 时置位，否则复位。

目的操作数从 00 减至 0FFh 时置位，否则复位。

V：目的操作数初值=08000h 且 C=0 时置位，否则复位。

目的操作数初值=080h 且 C=0 时置位，否则复位。

模式位　OSCOff、CPUOff 和 GIE 不受影响。

例　R12 所指 32 位计数器减去 R13 所指 16 位计数器。

```
SUB     @R13,0(R12)     ; 低字减
SBC     2(R12)          ; 高字减借位
```

例　R12 所指 16 位计数器减去 R13 所指 8 位计数器。

*** SETN　　负数位置位**

语法　SETN

操作　1 → N

模拟　BIS　　#4,SR

说明　负数位置位。

状态位　N：置位。
Z：不变。
C：不变。
V：不变。

模式位　OSCOff、CPUOff 和 GIE 不受影响。

*** SETZ　　零位置位**

语法　SETZ

操作　1 → Z

模拟　BIS　　#2,SR

说明　零位置位。

状态位　N：不变。
Z：置位。
C：不变。
V：不变。

模式位　OSCOff、CPUOff 和 GIE 不受影响。

SUB[.W]　　目的操作数减源操作数
SUB.B　　目的操作数减源操作数

语法　SUB　　源操作数,目的操作数　或　SUB.W　　源操作数,目的操作数
SUB.B　　源操作数,目的操作数

操作 目的操作数 + .NOT. 源操作数 + 1 →目的操作数
或
目的操作数 − 源操作数 →目的操作数

说明 目的操作数减去源操作数。用加上源操作数取反加 1 来实现。源操作数不变。目的操作数的原值消失。

状态位 N：结果为负数时置位，否则复位。
Z：结果为零时置位，否则复位。
C：当 C=1 时置位，否则复位；或者当不发生借位时置位，否则复位。
V：发生算术溢出时置位，否则复位。

模式位 OSCOff、CPUOff 和 GIE 不受影响。

例 见 SBC 指令的例。

例 见 SBC.B 指令的例。

注意：借位相当于进位位的非

借位相当于进位位取反。

借位	进位位
Yes	0
No	1

SUBC[.W] SBB[.W]　目的操作数减源操作数及借位(进位位的非)
SUBC.B,SBB.B　目的操作数减源操作数及借位(进位位的非)

语法 SUBC 源操作数,目的操作数 或 SUBC.W 源操作数,目的操作数
或
SBB 源操作数,目的操作数 或 SBB.W 源操作数,目的操作数
SUBC.B 源操作数,目的操作数
或
SBB.B 源操作数,目的操作数

操作 目的操作数 + .NOT. 源操作数 + C→目的操作数
或
目的操作数 − 源操作数 − 1 + C →目的操作数

说明 目的操作数减去源操作数。以加上源操作数取反及进位位来实现。源操作数不变。目的操作数原值消失。

状态位 N：结果为负数时置位，否则复位。
Z：结果为零时置位，否则复位。
C：当 C=1 时置位，否则复位；或者当不发生借位时置位，否则复位。
V：发生算术溢出时置位，否则复位。

模式位 OSCOff、CPUOff 和 GIE 不受影响。

例 两个浮点数的 24 位尾数相减。低字在 R13 和 R10 中，高字节在 R12 和 R9 中。

```
SUB.W   R13,R10                 ; 低 16 位减
SUBC.B  R12,R9                  ; 高 8 位减
```

例 R10 与 R11(高字节)的 16 位计数器减去 R13 所指 16 位计数器。

```
SUB.B   @R13+,R10               ; 低字节减
SUBC.B  @R13,R11                ; 高字节减，带进位位
...                             ;
```

注意：借位相当于进位位的非

借位相当于进位位取反。

借位	进位位
Yes	0
No	1

SWPB 字节交换

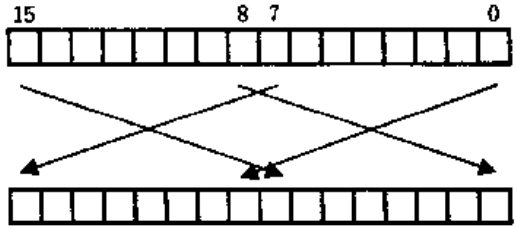
语法 SWPB 目的操作数

操作 b15 ～8 ↔ b7 ～0

说明 目的操作数的高低字节交换。

状态位 N：不变。
Z：不变。
C：不变。
V：不变。

模式位 OSCOff、CPUOff 和 GIE 不受影响。



例

```
MOV     #040BFh,R7              ; R7 = 0100000010111111
SWPB    R7                      ; R7 = 1011111101000000
```

例 R5 × 256。结果存入 R5、R4。

```
SWPB    R5                      ;
MOV     R5,R4                   ; 字节交换后的值复制到 R4
BIC     #0FF00h,R5              ; 修正结果
BIC     #00FFh,R4               ; 修正结果
```

SXT　　符号扩展

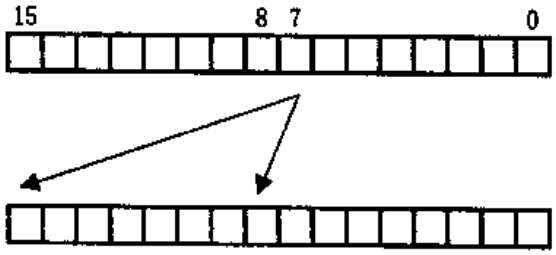
语法　SXT　　目的操作数

操作　Bit7 → Bit8 … Bit15

说明　低字节的符号扩展到高字节。

状态位　N：结果为负数时置位，否则复位。
Z：结果为零时置位，否则复位。
C：结果非零时置位，否则复位。
V：复位。

模式位　OSCOff、CPUOff 和 GIE 不受影响。



例 R7 装入定时器/计数器的值。SXT 指令将 b7 扩展到 b8 ～15。然后 R7 加入 R6。

```
MOV.B   &TCDAT,R7               ; TCDAT = 080h; .... .... 1000 0000
```

```
    SXT     R7                      ; R7 = 0FF80h: 1111 1111 1000 0000
    ADD     R7,R6                   ; 累加入 R6
```

*** TST[.W]　测试目的操作数**

*** TST.B　测试目的操作数**

语法　TST　目的操作数　或　TST.W　目的操作数
　　TST.B　目的操作数

操作　目的操作数 + 0FFFFh + 1
　　目的操作数 + 0FFh + 1

模拟　CMP　#0,目的操作数
　　CMP.B　#0,目的操作数

说明　目的操作数与"0"比较。结果影响状态位。目的操作数不变。

状态位　N：结果为负数时置位，否则复位。
　　Z：目的操作数为零时置位，否则复位。
　　C：置位。
　　V：复位。

模式位　OSCOff、CPUOff 和 GIE 不受影响。

例　测试 R7。当为负时跳转到 R7NEG；为"0"时跳转到 R7ZERO；为正数时执行 R7POS。

```
            TST     R7          ; 测试 R7
            JN      R7NEG       ; R7 为负数,跳转
            JZ      R7ZERO      ; R7 为"0",跳转
R7POS       …                   ; R7 为正数
R7NEG       …                   ; R7 为负数
R7ZERO      …                   ; R7 为"0"
```

例　测试 R7 的低字节。当为负时跳转到 R7NEG；为"0"时跳转到 R7ZERO；为正数时执行 R7POS。

```
            TST.B   R7          ; 测试 R7 的低字节
            JN      R7NEG       ; R7 的低字节为负数,跳转
            JZ      R7ZERO      ; R7 的低字节为"0",跳转
R7POS       …                   ; R7 的低字节为正数
```

```
R7NEG   …                    ; R7 的低字节为负数
R7ZERO  …                    ; R7 的低字节为"0"
```

XOR[.W]　源操作数和目的操作数异或
XOR.B　源操作数和目的操作数异或

语法　XOR　源操作数,目的操作数　或　XOR.W　源操作数,目的操作数
　　XOR.B　源操作数,目的操作数

操作　源操作数 .XOR. 目的操作数 →目的操作数

说明　源操作数和目的操作数异或。结果存入目的操作数。源操作数不变。

状态位　N：结果的 MSB=1 时置位,否则复位。
　　Z：结果为零时置位,否则复位。
　　C：结果为零时置位,否则复位。
　　V：两个操作数都为负数时置位,否则复位。

模式位　OSCOff、CPUOff 和 GIE 不受影响。

例　按 R6 的各位将 RAM 字 TONI 的各位翻转。

```
XOR     R6,TONI              ; 按 R6 中为"1"的各位使字 TONI 的位翻转
```

例　按 R6 的各位将 RAM 字节 TONI 的各位翻转。

```
XOR     R6,TONI              ; 按 R6 的低字节中为"1"的各位使字 TONI 的位翻转
```

例　R7 的低字节中与 RAM 字节 EDE 中不同的各位复位。

```
XOR.B   EDE,R7               ; 将不相同的各位置为"1"
INV.B   R7                   ; 低字节取反,高字节为"0"
```

B.5　用几条指令模拟的宏指令

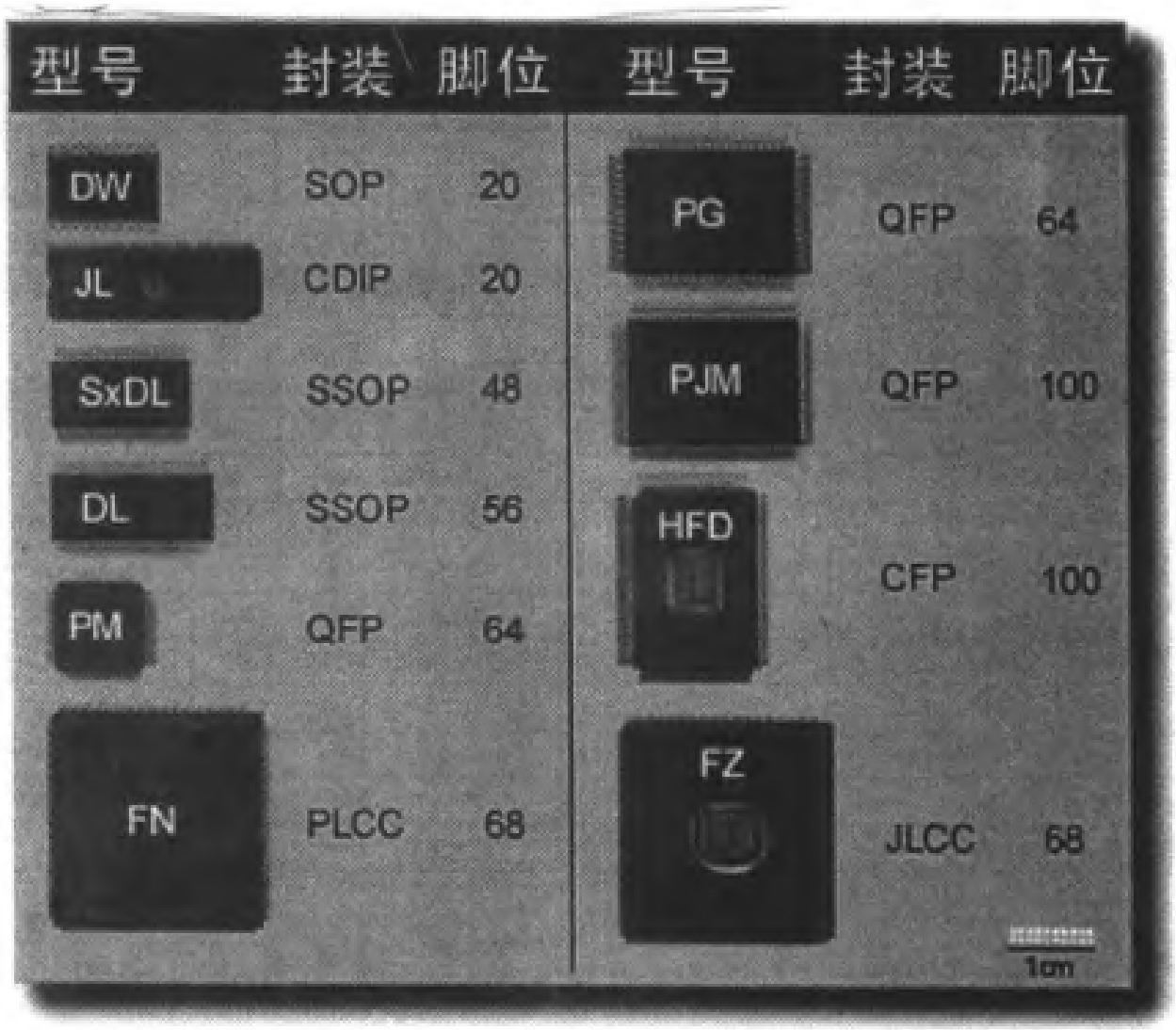
以下是需要更多的 RISC 指令模拟的指令,它们很少用到。立即数－1、0、+1、2、4 和 8 用常数发生器 R2/CG1 和 R3/CG2 产生。

模拟指令	指令序列		注释
ABS 目的操作数			
	TST	目的操作数	；求目的操作数的绝对值
	JN	L$0	；目的操作数为负
L$1	…		；目的操作数非负
	…		
	…		
L$0	INV	目的操作数	；转换目的操作数为正
	INC	目的操作数	
	JMP	L$1	
DSUB 源操作数,目的操作数			
ADD	#6666h,源操作数		；十进制减法
INV	源操作数		；源操作数被改变!
SETC			
DADD	源操作数,目的操作数		；源操作数－目的操作数（十进制）
NEG 目的操作数			
INV	目的操作数		；求目的操作数的相反数
INC	目的操作数		
RL 目的操作数			
ADD	目的操作数,目的操作数		；循环左移
ADDC	#0,目的操作数		
RR 目的操作数			
	CLRC		；循环右移
	RRC	目的操作数	
	JNC	L$1	
	BIS	#8000h,目的操作数	
L$1	…		

附录 C　MSP430 系列单片机参数表

型　号	OTP	ROM	FLASH	Memory/KB	RAM	A/D	LCD 驱动	定时器	封　装
MSP430F1101IDW			F1101	1	128 字节	slope	none	2	20SOP
MSP430F1111IDW			F1111	2	128 字节	slope	none	2	20SOP
MSP430F1121IDW			F1121	4	256 字节	slope	none	2	20SOP
MSP430F133IPM			F133	8	256 字节	12 Bit	none	3	64QFP
MSP430F135IPM			F135	16	512 字节	12 Bit	none	3	64QFP
MSP430F147IPM			F147	32	1KB	12 Bit	none	3	64QFP
MSP430F148IPM			F148	48	2KB	12 Bit	none	3	64QFP
MSP430F149IPM			F149	60	2KB	12 Bit	none	3	64QFP
MSP430C311IDL		C311		2	128 字节	slope	64	6	56SSOP
MSP430C312IDL		C312		4	256 字节	slope	92	6	56SSOP
MSP430C313IDL		C313		8	256 字节	slope	92	6	48SSOP
MSP430C315IDL	P315			16	512 字节	slope	92	6	48SSOP
MSP430C323IDG		C323		8	256 字节	14 Bit	84	6	64QFP
MSP430C325IPM	P325			16	512 字节	14 Bit	84	6	64QFP
MSP430C337IPM	P337			32	1 KB	slope	120	7	100QFP
MSP430F412IPM			F412	4	256 字节	slope	96	5	64QFP
MSP430F4137IPM			413	8	256 字节	slope	96	5	64QFP

附录 D　MSP430 系列单片机封装形式



Powered by xiaoguo’s publishing studio
QQ:8204136