MSP430x2xx 系列

用户指南

Literature Number: ZHCU032I
December 2004–Revised January 2012

内容

图片列表

图表列表

请先阅读

关于本手册

本手册介绍了 MSP430x2xx 器件系列的模块和外设。 每个讨论都给出了一般意义上的模块或外设。 目前所展示的并没有涵盖器件上所有模块或外围设备的所有特性和功能。 此外，在器件系列之间，在其严格的实施中模块或外围设备可能会有所不同，或在一个单独的器件或器件系列上可能无法完全实施。

引脚功能，内部信号连接和操作参数都因器件不同而各异。 有关这些细节，用户应该查阅《器件专用数据表》。

德州仪器 (TI) 提供的相关文档

有关相关文档，请参阅网站 http://www.ti.com/msp430。

FCC 警告

本设备仅限于在实验室测试环境中使用。 其会产生、使用并能够发出射频能量，且尚未经过测试，不确定是否符合 FCC 规则第 15 部分 J 子部分有关计算设备的限制，该限制可用于针对射频干扰提供合理的保护。 在其它环境中操作该设备可能会对无线电通讯造成干扰，在此情况下，用户必须自行承担为更正此干扰而需采取的任何相关措施的费用。

命名规则

程序示例，以一个特殊字体显示。

术语表

ACLK	辅助时钟	请参阅*基本时钟模块*
ADC	模数转换器	
BOR	掉电复位	请参阅*系统复位，中断，和运行模式*
引导加载程序 (BSL)	引导加载程序	有关应用报告请参阅*www.ti.com/msp430*
CPU	中央处理单元	请参阅*RISC 16 位 CPU*
DAC	数模转换器	
DCO	数字控制振荡器	请参阅*基本时钟模块*
dst	目的	请参阅*RISC 16 位 CPU*
FLL	频率锁定环路	请参阅 MSP430x4xx 系列用户指南中的*FLL+*
GIE	通用中断使能	请参阅*系统复位，中断，和操作模式*
INT (N/2)	N/2 的整数部分	
I/O	输入/输出	请参阅*数字 I/O*
ISR	中断服务子程序	
LSB	最低有效位	
LSD	最低有效位数	
LPM	低功耗模式	请参阅*系统复位，中断，和操作模式*
MAB	存储器地址总线	
MCLK	主时钟	请参阅*基本时钟模块*

MDB	存储器数据总线	
最高有效位	最高有效位	
MSD	最高有效位数	
NMI	（不）可屏蔽中断	请参阅*系统复位，中断，和操作模式*
PC	程序计数器	请参阅*RISC 16 位 CPU*
POR	加电复位	请参阅*系统复位，中断，和操作模式*
PUC	加电清零	请参阅*系统复位，中断，和操作模式*
RAM	随机存取存储器	
SCG	系统时钟发生器	请参阅*系统复位，中断，和操作模式*
SFR	特殊功能寄存器	
SMCLK	子系统主时钟	请参阅*基础时钟模块*
SP	堆栈指针	请参阅*RISC 16 位 CPU*
SR	状态寄存器	请参阅*RISC 16 位 CPU*
src	源	请参阅*RISC 16 位 CPU*
TOS	栈顶	请参阅*RISC 16 位 CPU*
WDT	安全装置定时器	请参阅*安全装置定时器*

寄存器位惯例

每个寄存器用一个键表示出每个单独的位的可用性，以及初始条件：

寄存器位的可用性和初始条件

键	可访问位
rw	读取/写入
r	只读
r0	读取为 0
r1	读取为 1
w	只写入
w0	写入为 0
w1	写入为 1
(w)	没有寄存器位被执行；在一个脉冲中写入一个 1 结果。 寄存器位始终读取为 0。
h0	由硬件清零
h1	由硬件置位
-0,-1	PUC 后的条件
-(0),-(1)	POR 后的条件

简介

本章对 MSP430 的架构进行了说明。

Topic **Page**

1.1 架构

MSP430 包含一个 16 位精简指令集 (RISC) CPU，外设，和一个灵活的时钟系统，这些器件通过一个非纽曼共同存储器地址总线 (MAB) 和一个存储器数据总线 (MDB) 互连（请见图 1-1）。 通过将一个最新 CPU 与模块化存储器映射模拟和数字外设搭配使用，MSP430 为要求严格的混合信号应用提供解决方案。

MSP430x2xx 系列的特性包括：

- 超低功耗架构延长了电池寿命
  - 0.1µA RAM 保持模式
  - 0.8µA 实时时钟模式
  - 250µA/MIPS 工作模式
- 非常适合于精准测量的高性能模拟
  - 针对测量电阻式元件的比较器栅极定时器
- 16 位 RISC CPU 可使用一小部分代码来实现全新应用。
  - 大寄存器文件去除了工作文件瓶颈
  - 紧凑内核设计减少了功耗和成本
  - 针对最新高级编程进行了优化
  - 只有 27 个内核指令和 7 个寻址模式
  - 扩展矢量化中断功能
- 系统内可编程闪存允许灵活代码变化、字段升级和数据记录

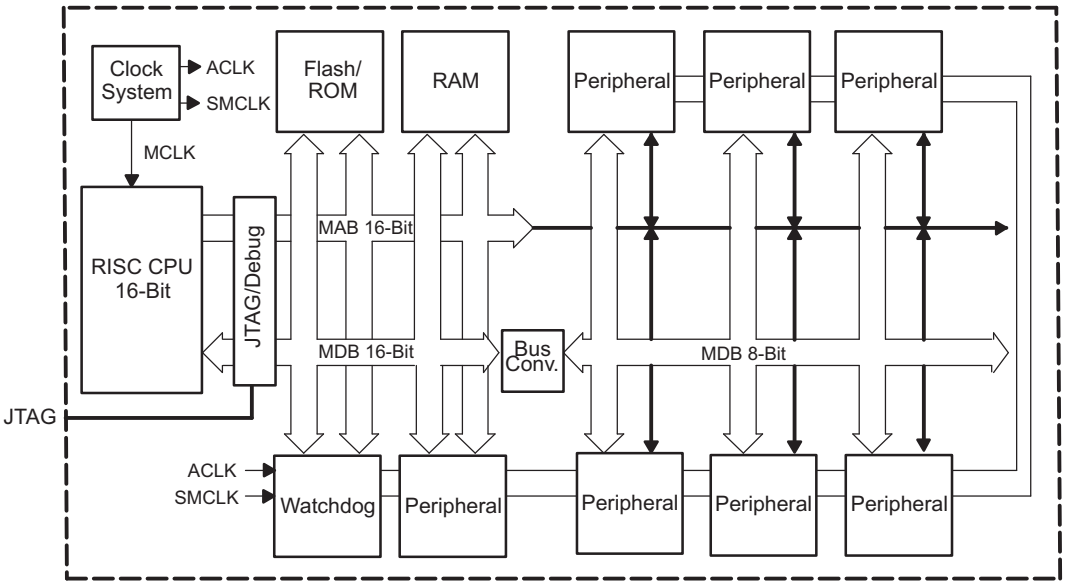


图 **1-1. MSP430** 架构

1.2 灵活的时钟系统

时钟系统专门为电池供电类应用而设计。 由一个普通 32kHz 时钟晶振来直接驱动一个低频辅助时钟 (ACLK)。 ACLK 可被用于一个后台实时时钟自我唤醒功能。 一个集成的高速数控振荡器 (DCO) 可为 CPU 和高度外设所使用的主控时钟 (MCLK) 供源。 通过设计，DCO 被激活并且 1MHz 时，稳定时间少于 2µs。 基于 MSP430 的解决方案在极短突发中高效使用高性能 16 位 RISC CPU。

- 低频辅助时钟=超低功耗待机模式
- 高速主控时钟=高性能信号处理

1.3 嵌入式仿真

专用嵌入式仿真逻辑位于器件本身并且在无需额外系统资源的情况下可通过 JTAG 进行访问。

嵌入式仿真的优势包括：

- 支持非侵入式开发和全速执行时的调试，断点，和一个应用中的单步执行。
- 系统内开发具有与最终应用一样的特性。
- 混合信号完整性被保留并且不受线缆接口的影响。

1.4 地址空间

MSP430 非纽曼架构具有一个与特殊功能寄存器 (SFR)，外设，RAM 和 闪存 / ROM 存储器共用的地址空间，显示在图 1-2中。 特定存储器映射请参阅器件专用数据表。 代码访问一直在偶数地址上执行。 数据可作为字节或字进行访问。

现在可寻址存储器空间为 128KB。

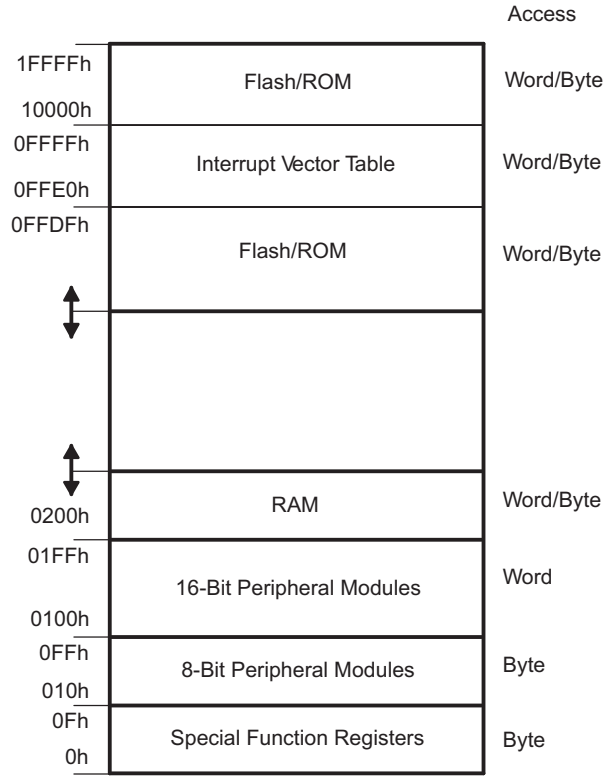


图 **1-2.** 内存映射

1.4.1 闪存 / ROM

闪存 / ROM 的开始地址取决于出现的 闪存 / ROM 的数量并且由器件改变。 针对具有少于 60KB 闪存 / ROM 的器件，闪存 / ROM 的末尾地址为 0x0FFFF。闪存可被用于代码和数据。 字或字节表可在闪存 / ROM 中存储和使用，而无需在使用前，将它们复制到 RAM。

中断矢量表被映射到闪存 / ROM 地址空间的上部 16 字，在这里，具有最高优先级的中断矢量位于最高闪存 / ROM 字地址(0x0FFFE)。

1.4.2 RAM

RAM 从 0200h 开始。 RAM 的末尾地址取决于出现的 RAM 的数量并且由器件改变。 RAM 可被用于代码和数据。

1.4.3 外设模块

外设模块被映射到地址空间。 从 0100 到01FFh 的地址空间为 16 位外设模块所保留。 这些模块应该通过字指令访问。 如果使用字节指令，那么只允许偶数地址，并且结果的高字节一直为 0。

从 010h 到 0FFh 的地址空间为 8 位外设模块所保留。 应该使用字节指令访问这些模块。 使用字指令的字节读取访问导致高字节内的无法预计的数据。 如果字数据被写入一个字节模块，那么只有低字节被写入外设寄存器，高字节被忽略。

1.4.4 特别功能寄存器(SFR)

在 SFR 中配置 某些外设功能。 SFR 位于地址空间的较低16 个字节内，并且采用字节的形式。 只能使用字节指令来访问 SFR。 适用的 SFR 位请参阅器件专用数据表。

1.4.5 内存组织

字节位于偶数或者奇数地址内。 字只位于图 1-3中显示的偶数地址内。 当使用字指令时，只可使用偶数地址。 一个字的低字节一直为一个偶数地址。 高字节位于下一个奇数地址。 例如，如果一个数据字位于地址 xxx4h 上，那么那个数据字的低字节位于地址xxx4h 上，而那个字的高字节位于地址 xxx5h 上。

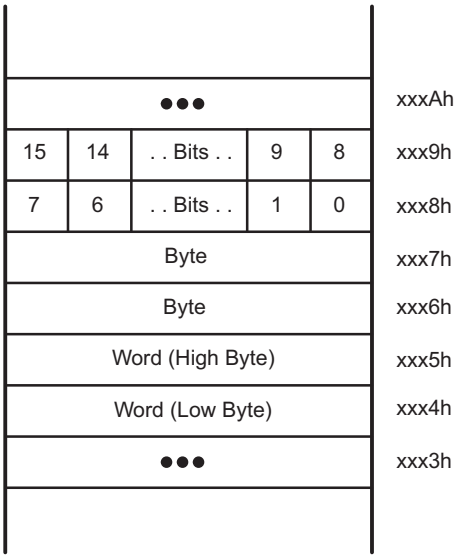


图 **1-3.** 位、字节和字位于字节格式的存储器内

1.5 MSP430x2xx 系列改进和增强

表 1-1突出强调了对 MSP430x2xx 系列的改进和增强。 在下面的章节中详细讨论了改进和增强，或者显示在器件专用数据表中的改进的器件参数。

表 1-1. MSP430x2xx 系列改进和增强

主题	改进和增强
复位	• 所有 MSP430x2xx 器件都包含掉电复位。 • PORIFG 和 RSTIFG 标志已经被添加到 IFG1 中来表明复位的原因。 • 从地址范围 0x0000-0x01FF 内的一个取指令将复位器件。
安全装置定时器	• 所有 MSP430x2xx 器件集成了安全装置 Timer+ 模块 (WDT+)。 WDT+ 确保针对定时器的时钟源永远不被禁用。
基本时钟系统	• LFXT1 振荡器有处于 LF模式中的可选的负载电容器。 • LFXT1 支持处于 LF 模式中的高达 16MHz 的晶振。 • LFXT1 包括处于 LF 模式中的振荡器故障检测。 • XIN 和 XOUT 引脚是 20 和 28 引脚器件上的共用功能引脚。 • 在某些器件上不支持 DCO 的外部 R_{OSC}特性。 在这个情况下，软件不应该设定 BCSCTL2 寄存器内的 LSB。详细信息请参阅器件专用数据表。 • DCO 运行频率已经被大大提升。 • DCO 温度稳定性已经被大大改进。
闪存存储器	• 信息存储器有 4 个每个大小为 64 字节的段。 • 段 A 用 LOCKA 位单独锁定。 • LOCKA位保护所有信息不被批量擦除。 • 段擦除可由一个中断来中断。 • 可使用一个中断来中止闪存升级。 • 闪存编程电压已经被降低到 2.2V。 • 编程/擦除电压已经被减少。 • 时钟故障中止一个闪存升级。
数字 I/O	• 所有端口有集成的上拉/下拉电阻器。 • P2.6 和 P2.7 功能已经被添加到 20 和 28 引脚器件。这些是与 XIN 和 XOUT 共用的功能。 如果需要晶振操作，那么软件一定不能清除用于这些引脚的 P2SELx 位。
Comparator_A	• Comparator_A 已经用一个全新的输入复用器扩展了输入功能。
低功率	• LPM3 在 3V 时的典型流耗已经被减少了大约 50%。 DCO 启动时间已经被大大减少。
运行频率	• 3.3V 时的最大运行频率为 16MHz。
引导加载程序 (BSL)	• 一个不正确的密码会引起一个批量擦除。 • BSL进入序列更加稳健耐用以防止意外进入和擦除。

系统复位、中断、和运行模式

本章对 MSP430x2xx 系统复位、中断、和运行模式进行了说明

Topic **Page**

2.1 系统复位和初始化

显示在图 2-1中的系统复位电路提供一个加电复位 (POR) 和一个加电清零 (PUC) 信号。不同的事件触发这些信号，而不同的初始条件的存储在取决于哪个信号被生成。

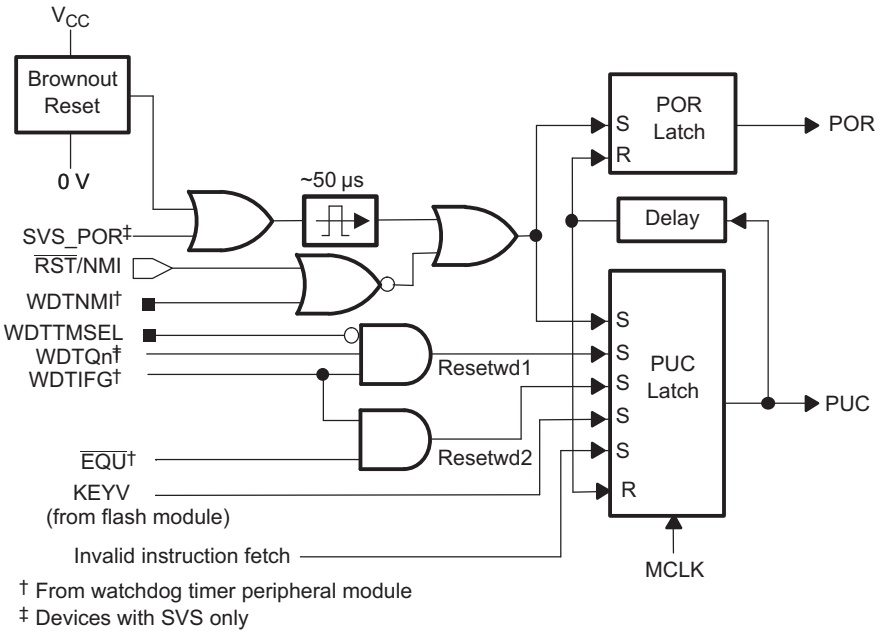


图 2-1. 加电复位和加电清零电路原理图

一个 POR 是一个器件复位。 只有下列事件生成一个 POR：

- 为器件加电
- 当配置复位模式中时，$\overline{\text{RST}}$/NMI 上的一个低电平信号
- 当 PORON=1 时，一个 SVS 低电平条件。

当一个 POR 被生成时，将一直生成一个 PUC，但是 PUC 不会生成一个 POR。以下事件触发一个 PUC：

- 一个 POR 信号
- 只在安全装置模式中时的安全装置定时器过期
- 安全定时器安全密钥违反
- 一个闪存存储器安全密钥违反
- 从 0h 到 01FFh 外设地址范围内的一个 CPU 取指令

2.1.1 欠压复位 (BOR)

当一个电源电压被应用或者从 V_{CC}端子上移除时，欠压复位电路检测低电源电压。 欠压复位电路通过在电源被应用或移除时触发一个 POR 信号来复位器件。 运行电平显示在图 2-2中。

当 V_{CC}超过$V_{CC\ (启动)}$电平时，POR 信号被激活。 它在 V_{CC}超过$V_{(B_IT+)}$阀值前保持有效并且推迟 $t_{(BOR)}$消失的时间。延迟 $t_{(BOR)}$可被延长以自使用一个缓慢的斜升V_{CC}。 滞后 $V_{hys(B_IT-)}$确保电源电压必下降至低于 $V_{(B_IT-)}$来从欠压复位电路中生成其它 POR 信号。

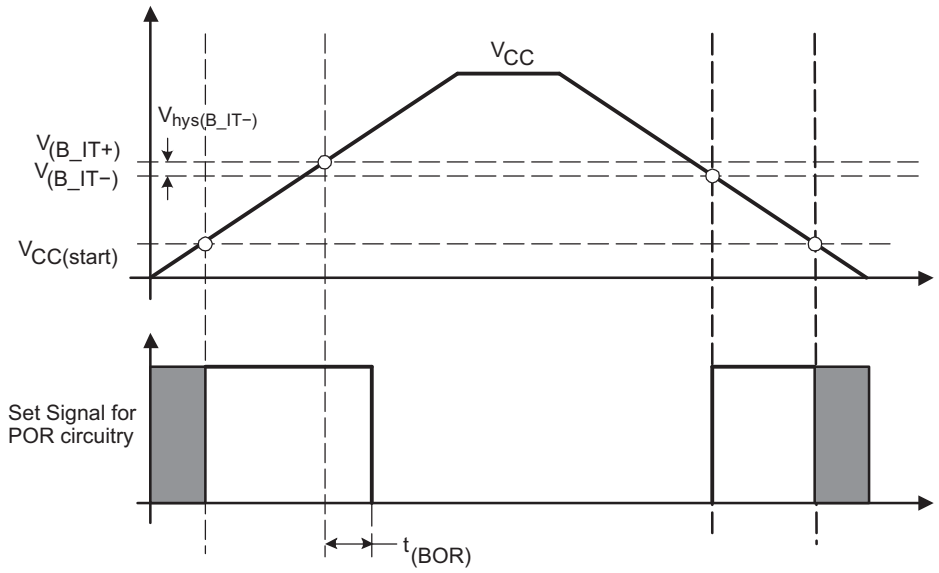


图 **2-2.** 欠压时序

由于 $V_{(B-IT-)}$电平远远高于 POR 电路的$V_{最小值}$电平，BOR 为电源故障提供一个复位，在电源故障中 V_{CC}没有下降到低于 $V_{最小值}$。 参数请参阅器件专用数据表。

2.1.2 系统复位后的器件初始条件

一个 POR 之后，初始 MSP430 条件为：

- $\overline{\text{RST}}$/NMI 引脚在复位模式中被配置。
- I/O 引脚被切换至数字 *I/O*一章所描述的输入模式。
- 其它外设模块和寄存器被如本手册中它们各自章节所描述的那样被初始化。
- 状态寄存器 (SR) 被复位。
- 在安全装置模式中，安全装置定时器加电有效。
- 程序计数器 (PC) 被载入包含在复位矢量位置(0FFFEh) 内的地址。 如果复位矢量内容为 0FFFFh，为了实现最小功耗，此器件将被禁用。

2.1.2.1 软件初始化

一个系统复位后，用户软件必须针对应用要求初始化 MSP430。 必须进行以下操作：

- 初始化 SP，通常至 RAM 的顶部
- 将安全装置初始化为应用要求的那样。
- 将外设模块配置为应用要求的那样。

此外，安全装置定时器、振荡器故障、和闪存存储器标志可被评估以确定复位源。

2.2 中断

中断优先级是固定的并且由图 2-3中显示的连接链中的模块安排来定义。 距离 CPU/NMIRS 越近的模块，其优先级越高。中断优先级确定当多于一个中断在同时等待时，采用哪一个中断。

中断有三个类型：

- 系统复位
- （不）可屏蔽 NMI
- 可屏蔽

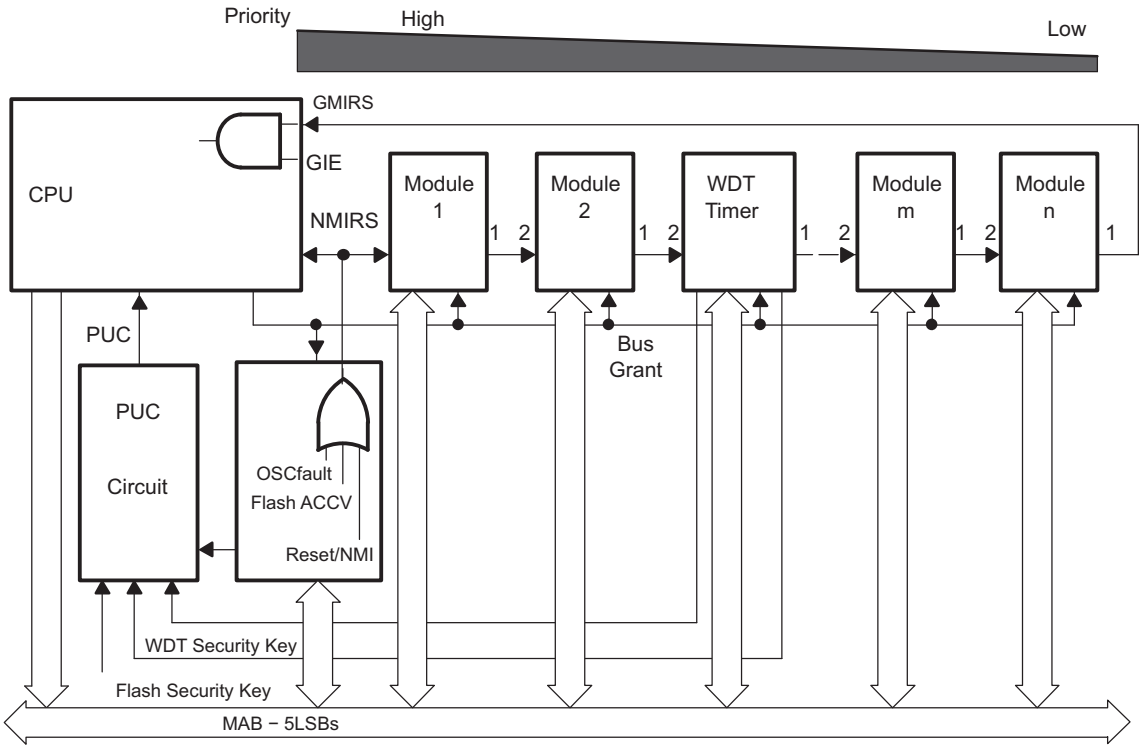


图 **2-3.** 中断优先级

2.2.1 （不）可屏蔽中断 (NMI)

（不）可屏蔽 NMI 中断不由通用中断使能位 (GIE) 屏蔽，但是由单独中断使能位 (NMIIE，ACCVIE，OFIE) 启用。 当一个 NMI 中断被接受时，所有 NMI 中断使能位被自动复位。 程序在存储在（不）可屏蔽中断矢量，0FFFCh 上开始执行。 用户软件必须设定中断所需的NMI 中断使能位来被再次启用。 针对 NMI 源的方框图显示在图 2-4中。

可由三个源来生成一个（不）可屏蔽 NMI 中断：

- 配置在 NMI模式时的，$\overline{\text{RST/NMI}}$上的边沿
- 一个振荡器故障出现
- 一个到闪存存储器的方位违反

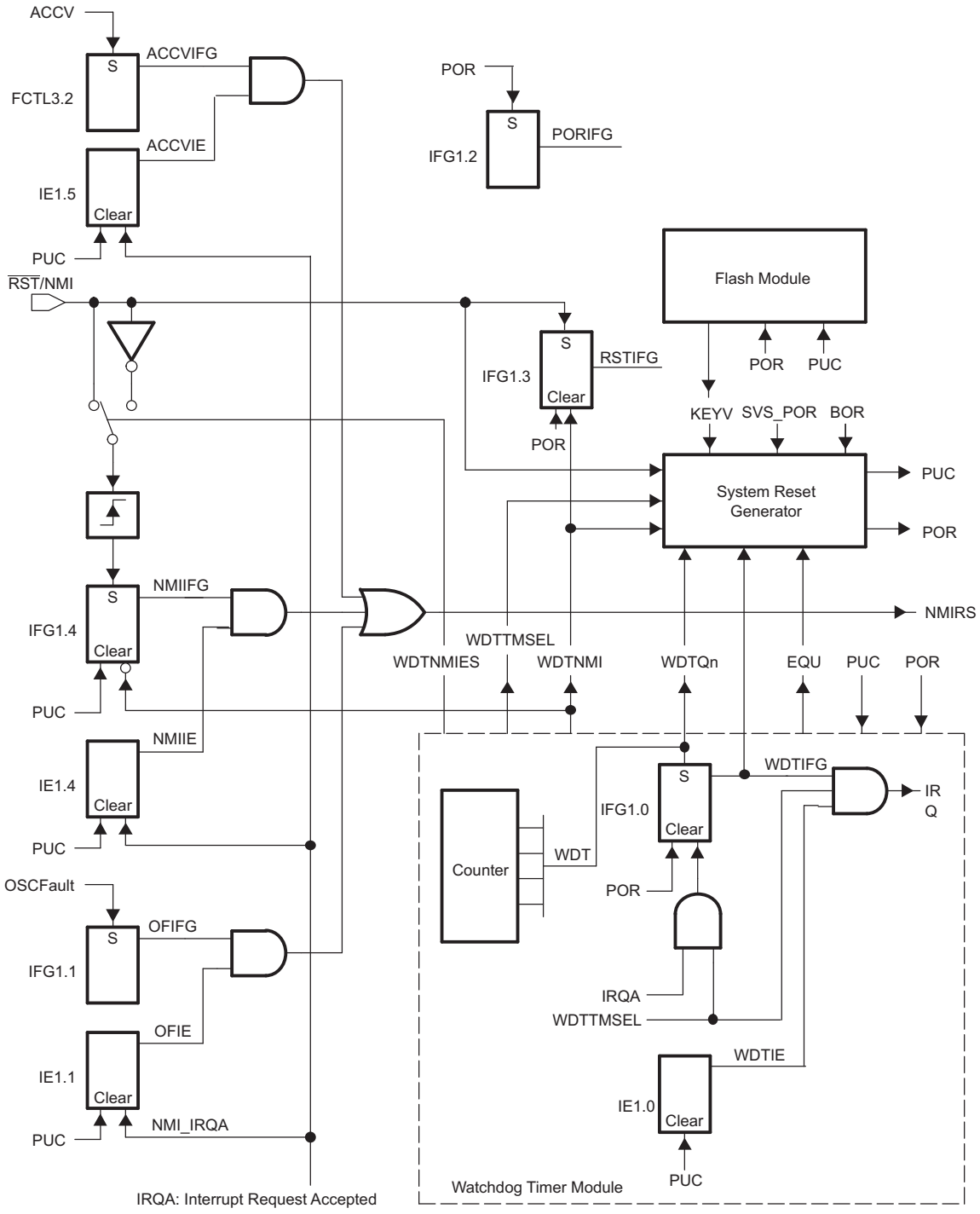


图 **2-4.** （不）可屏蔽中断源的方框图

2.2.1.1 复位 / NMI 引脚

加电时，RST/NMI 引脚在复位模式中被配置。 RST/NMI 引脚的功能在安全装置控制寄存器WDTCTL 中被选择。 如果RST/NMI 引脚被设定为复位功能，CPU 在RST/NMI 引脚被保持在低电平时被保持在复位状态。 在输入改变为一个高电平状态后，CPU 在存储于复位矢量，0FFFEh 上执行程序，并且 RSTIFG 标志被置位。

如果RST/NMI 引脚由用户软件配置为 NMI功能，一个由 WDTNMIIES 位选择的信号边沿在 NMIIE 位被置位时生成一个 NMI 中断。 RST/NMI 标志 NMIIFG 也被置位。

注: 保持RST/NMI 为低电平

当在 NMI 模式中进行配置时，一个生成 NMI 事件的信号不应将RST/NMI 保持在引脚低电平。如果一个 PUC 在 NMI信号为低电平时从一个不同源出现，器件将保持在复位状态，这是因为一个 PUC 将RST/NMI 引脚改为复位功能。

注: 修改 WDTNMIES

当 NMI 模式被选择并且 WDTNMIES 位被改变，可生成一个 NMI，这取决于 RST/NMI 引脚上的实际电平。 当 NMI 边沿选择位在选择 NMI 模式前被改变，那么 不生成 NMI。

2.2.1.2 闪存访问违反

当一个闪存访问违反出现时，一个 ACCVIFG 标志被设定。 通过设置 ACCVIE 位，闪存访问违反可被启用来生成一个 NMI 中断。 然后，ACCVIFG 标志可被 NMI 中断处理例程测试以确定 NMI 是否由一个闪存访问违反引起。

2.2.1.3 振荡器故障

振荡器故障信号警告一个晶体振荡器可能的错误条件。 振荡器故障可被启用来通过设置 OFIE位来生成一个 NMI 中断。 然后可用 NMI 中断处理例程来测试 OFIFG 标志以确定 NMI 是否由一个振荡器故障引起。

一个 PUC 信号能够触发一个振荡器故障，这是因为 PUC 将 LFXT1 切换为LF 模式，从而关闭了 HF 模式。 PUC 信号也关闭 XT2振荡器。

2.2.1.4 NMI 中断处理器示例

NMI 中断是一个多源中断。 一个 NMI 中断自动复位 NMIIE，OFIE 和ACCVIE 中断使能位。 用户 NMI 处理例程复位中断标志并且按照图 2-5中显示的应用需求来重新启用中断使能位

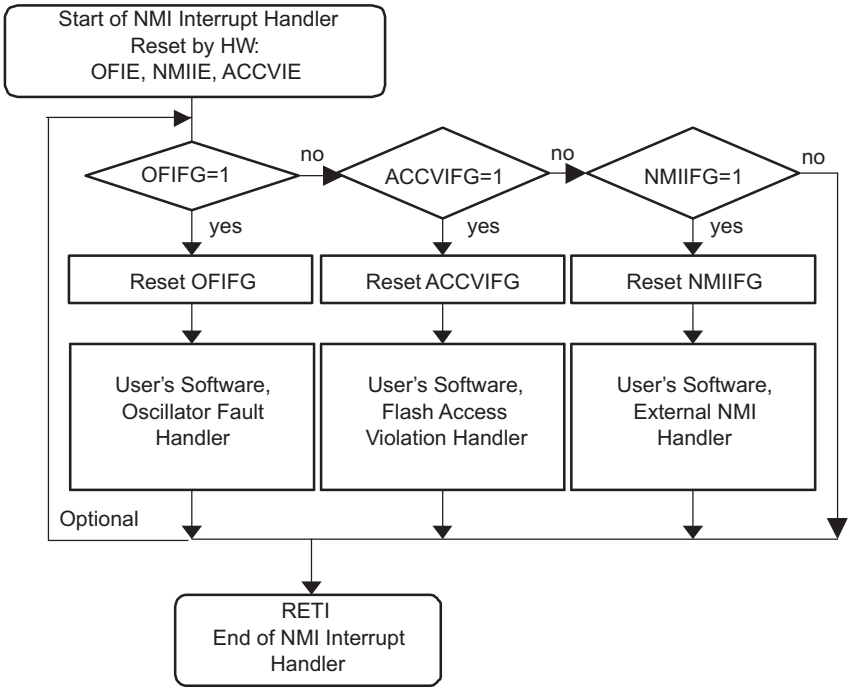


图 **2-5. NMI** 中断处理器

注: 用 **ACCVIE**，**NMIIE**，和**OFIE**来启用 NMI 中断

为了避免嵌套式 NMI 中断，ACCVIE，NMIIE，和 OFIE 使能位不应在一个 NMI 中断例程内被置位。

2.2.2 可屏蔽中断

可屏蔽中断由具有中断功能的外设引起，其中包括间隔定时器模式中的安全装置定时器上溢。 每个可屏蔽中断源可被一个中断使能位单独禁用，或者所有可屏蔽中断可由状态寄存器 (SR) 内的通用中断使能 (GIE) 位禁用。

在本手册中相关的外设模块章节中讨论了每个单独的外设中断。

2.2.3 中断处理

当外设请求一个中断并且外设中断使能位和 GIE 位被置位时，中断出了力例程被请求。 只需设定单独使能位即可请求（不）可屏蔽中断。

2.2.3.1 中断接受

中断延迟为 5 周期 (CPUx) 或 6 周期 (CPU)，从接受一个中断请求开始并且持续到中断处理例程的第一条指令开始执行，如图 2-6所示。 中断逻辑执行以下操作：

1. 任何当前执行的指令完成。
2. 指向下一条指令的 PC 被压入堆栈。
3. SR 被压入堆栈。
4. 如果在最后一个指令执行期间由多个中断出现，那么具有最高优先级的中断被选中并等待被处理。
5. 在单一源标志上，中断请求标志自动复位。对于软件处理，多个源标志保持被设定。
6. SR 被清除。 这将终止任何低功耗模式。 由于 GIE 位被清除，之后的中断被禁用。
7. 中断矢量的内容被载入到 PC：程序继续在中断处理例程所处的地址上执行。

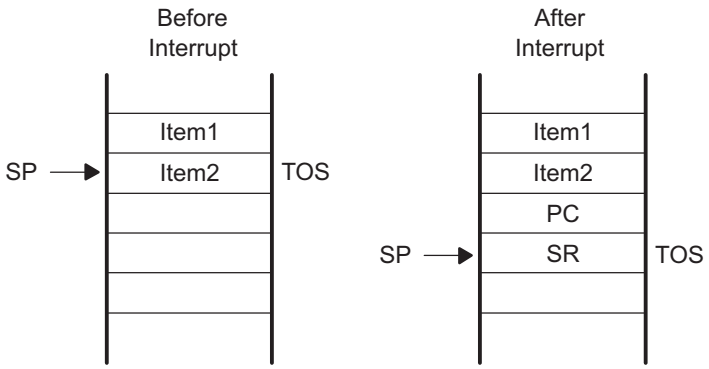


图 **2-6.** 中断处理

2.2.3.2 从中断返回

中断处理例程由以下指令终止：

RETI（从中断处理例程返回）

从中断返回需要花费 5 个周期 (CPU) 或者 3 个周期 (CPUx) 来执行下列操作并在图 2-7中进行了说明。

1. 带有所有之前设置的 SR 从堆栈中弹出。 所有GIE，COUOFF 等之前的设置现在有效，无论中断处理例程期间使用的是什么设置。
2. PC 从堆栈弹出并且开始在其被中断的位置开始执行。

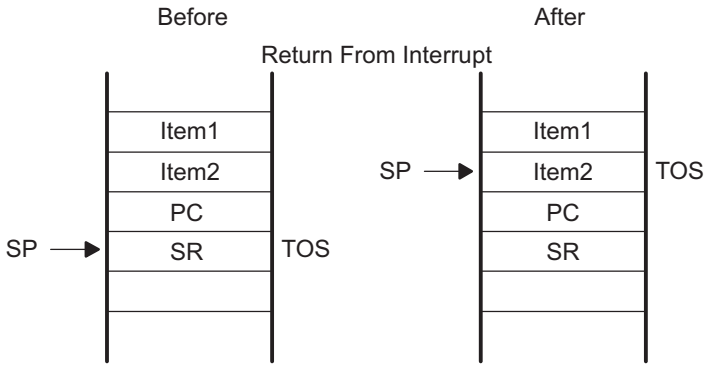


图 **2-7.** 从中断返回

2.2.3.3 中断嵌套

如果 GIE 位在中断处理例程中内置位，则中断嵌套被启用。 当中断嵌套被启用，任何在中断处理例程期间出现的中断将中断例程，无论中断的优先级是什么。

2.2.4 中断矢量

中断矢量和加电起动地址位于地址范围 0FFFFh 至 0FFC0h 内，与表 2-1中描述的相同。 一个矢量由用户使用相应中断处理例程的 16 位地址进行编辑。 完整中断矢量列表请参与器件专用数据表。

建议为每个分配给一个模块的中断矢量提供一个中断处理例程。 一个假中断处理例程可以只包含 RETI指令，并且几个中断矢量可指向它。

如果需要，未分配的中断矢量可被用于常规程序代码。

某些模块使能位、中断使能位、和中断标志位于SFR 内。 SFR 位于较低的地址范围并且用字节格式执行。必须使用字节指令来访问 SFR。 SFR配置请参阅器件专用数据表。

表 **2-1.** 中断源、标志、和矢量

中断源	中断标志	系统中断	字地址	优先级
加电、外部复位、安全装置、闪存密码、非法取指令	PORIFG RSTIFG WDTIFG KEYV	复位	0FFFEh	31，最高
NMI，振荡器故障，闪存存储器访问违法	NMIIFG OFIFG ACCVIFG	(不)可屏蔽 (不)可屏蔽 (不)可屏蔽	0FFFCh	30
器件专用			0FFFAh	29
器件专用			0FFF8h	28
器件专用			0FFF6h	27
安全装置定时器	WDTIFG	可屏蔽	0FFF4h	26
器件专用			0FFF2h	25
器件专用			0FFF0h	24
器件专用			0FFEEh	23
器件专用			0FFECh	22
器件专用			0FFEAh	21
器件专用			0FFE8h	20
器件专用			0FFE6h	19
器件专用			0FFE4h	18
器件专用			0FFE2h	17
器件专用			0FFE0h	16
设备专用			0FFDEh	15
器件专用			0FFDCh	14
器件专用			0FFDAh	13
器件专用			0FFD8h	12
器件专用			0FFD6h	11
器件专用			0FFD4h	10
器件专用			0FFD2h	9
器件专用			0FFD0h	8
器件专用			0FFCEh	7
器件专用			0FFCCh	6
器件专用			0FFCAh	5
器件专用			0FFC8h	4
器件专用			0FFC6h	3
器件专用			0FFC4h	2
器件专用			0FFC2h	1
器件专用			0FFC0h	0，最低

2.3 操作模式

MSP430 系列设计用于超低功耗应用并且使用不同的运行模式，这些模式显示在图 2-9中。

运行模式考虑了三个不同的模式：

- 超低功耗
- 速度和数据吞吐量
- 独立外设流耗最小化

图 2-8显示了 MSP430 典型流耗。

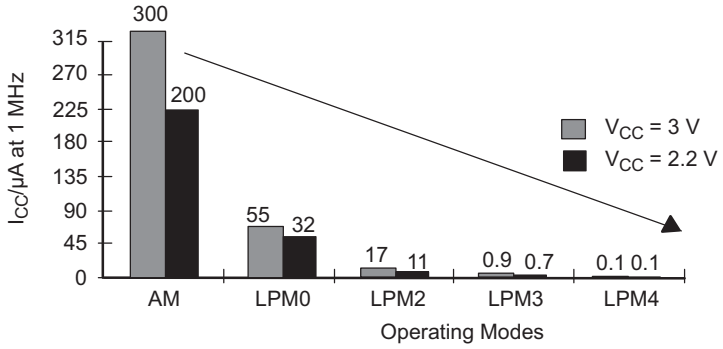


图 **2-8. 'F21x1** 器件的典型流耗与运行模式间的关系

使用状态寄存器内的 CPUOFF，OSCOFF，SCG0 和 SCG1 位来配置低功耗模式 0 到 4。在状态寄存器内包含 CPUOFF，OSCOFF，SCG0 和 SCG1 控制位的优势在于现有的运行模式在中断处理例程期间被保存在堆栈内。 如果中断处理例程期间保存的 SR值未改变，程序流返回到之前的运行模式。 通过操作堆栈内的保存的 SR 值而不是中断处理例程，程序流可返回至一个不同的运行模式。 可使用任何指令来访问模式控制位和堆栈。

当设置任一模式控制位时，所选的运行模式立即生效（请见图 2-9）。 在时钟被激活前，带有被禁用时钟的外设运行被禁用。 也可使用它们各自的控制寄存器设置来禁用外设。 所有 I/O 端口引脚和 RAM / 寄存器未改变。 通过所有被启用的中断可实现唤醒。

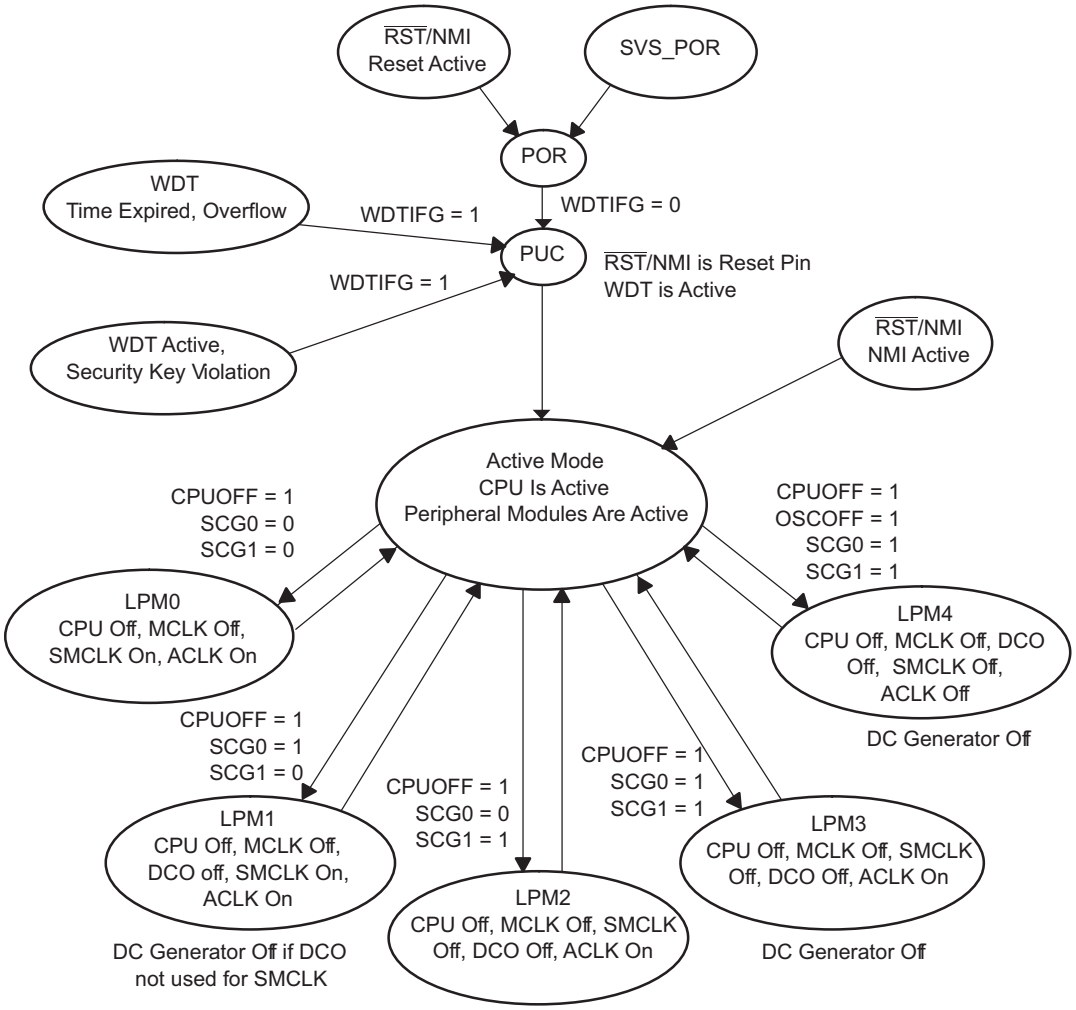


图 **2-9.** 针对基本时钟系统的运行模式

表 **2-2.** 针对基本时钟系统的运行模式

SCG1	SCG0	OSCOFF	CPUOFF	模式	CPU 和时钟状态
0	0	0	0	激活	CPU 被激活，所有被启用的时钟被激活
0	0	0	1	LPM0	CPU，MCLK 被禁用，SMCLK，ACLK 被激活
0	1	0	1	LPM1	CPU，MCLK 被禁用。 如果数控振荡器 (DCO) 不被用于 SMCLK，DCO 和 DC 生成器被禁用。 ACLK 被激活。
1	0	0	1	LPM2	CPU，MCLK，SMCLK，DCO 被禁用。 DC 生成器保持启用。ACLK 被激活。
1	1	0	1	LPM3	CPU，MCLK，SMCLK，DCO 被禁用。 DC 生成器被禁用。 ACLK被激活。
1	1	1	1	LPM4	CPU 和所有时钟被禁用。

2.3.1 进入和退出低功耗模式

一个被启用的中断事件将 MSP430 从任一低功耗模式中唤醒。程序流如下：

- 进入中断处理例程：
  - PC 和 SR 被存储在堆栈上
  - CPUOFF，SCG1 和 OSCOFF 位被自动复位
- 用于从中断处理例程返回的选项：
  - 原先的 SR 从堆栈中弹出，从而恢复之前的运行模式。
  - 存储在堆栈中的 SR 位可在中断处理例程内被修改，当 RETI 指令被执行时，返回之前的运行模式。

```
; Enter LPM0 ExampleBIS #GIE+CPUOFF,SR ; Enter LPM0; ... ; Program stops here;; Exit LPM0
Interrupt Service RoutineBIC #CPUOFF,0(SP) ; Exit LPM0 on RETIRETI; Enter LPM3 ExampleBIS
#GIE+CPUOFF+SCG1+SCG0,SR ; Enter LPM3; ... ; Program stops here;; Exit LPM3 Interrupt Service
RoutineBIC #CPUOFF+SCG1+SCG0,0(SP) ; Exit LPM3 on RETIRETI
```

2.4 低功耗应用的原则

通常，减少功耗的最重要的因素是使用 MSP430 时钟系统来大大增加 LPM3 内的时间。 具有一个实时时钟功能并且所有中断有效的 LPM3 功耗的典型值少于 2μA。 一个 32kHz 手表晶振用于 ACLK 并且CPU 由 DCO（通常关闭）计时，此 DCO 有一个 1μs 的唤醒时间。

- 使用中断来唤醒处理器并控制程序流。
- 应该只在需要时打开外设。
- 使用低功耗集成外设模块来取代软件启动的功能。例如 Timer_A 和 Timer_B 可自动生成 PWM 并且捕捉外部时序，而无需CPU 资源。
- 计算出的转移和快速表查询应该用来取代标志轮询和长软件计算。
- 由于开销，应避免频繁的子例程和函数调用。
- 对于较长的软件例程，应使用单周期 CPU 寄存器。

2.5 未使用引脚的连接

表 2-3中列出了所有未使用引脚的正确连接。

表 **2-3.** 未使用引脚的连接

引脚	电势	注释
AV_{CC}	DVCC	
AV_{SS}	DVSS	
V_{REF+}	断开	
V_{eREF+}	DVSS	
V_{REF-}/V_{eREF-}	DVSS	
XIN	DVCC	只适用于专用 XIN 引脚。 带有共用 GPIO 功能的 XIN 引脚应该被编程为 GPIO 并且按照 Px.0 至 Px.7 建议的那样设置。
XOUT	断开	只适用于 XOUT 引脚。 带有共用 GPIO功能的 XOUT 引脚应该被编程为 GPIO 并且按照 Px.0 至 Px.7 建议的那样设置。
XT2IN	DVSS	只适用于 X2IN 引脚。 带有共用 GPIO功能的 X2IN 引脚应该被编程为 GPIO 并且按照 Px.0 至 Px.7 建议的那样设置。
XT2OUT	断开	只适用于 X2OUT 引脚。 带有共用 GPIO功能的 X2OUT 引脚应该被编程为 GPIO 并且按照 Px.0 至 Px.7 建议的那样设置。
Px.0 至 Px.7	断开	用启用的上拉/下拉电阻器来切换端口功能、输出方向或输入
$\overline{RST}$/NMI	DVCC 或 VCC	带有 10nF (2.2nF[(1)]) 下拉电容的 47kΩ 上拉电阻器
测试	断开	20xx，21xx，22xx 器件
TDO	断开	
TDI	断开	
TMS	断开	
TCK	断开	

(1) 当使用带有 Spy-Bi-Wire（两线制）接口，处于 Spy-Bi-Wire 模式或者处于 4 线制模式，具有诸如FET 接口或者 GANG 编程器的 TI 工具的器件时，下拉电容器的值不应超过 2.2nF。

CPU

本章对 MSP430 CPU，寻址模式，和指令集进行了说明。

Topic **Page**

3.1 CPU 介绍

CPU 包含特别针对最新编程技术而设计的特性，例如计算分支，表处理，和高级语言，诸如 C 语言的使用。在不分页的情况下，CPU 能够寻址整个地址范围。

CPU 特性包括：

- 具有 27 条指令和 7 个寻址模式的 RISC
- 与可使用每个寻址模式的每条指令的正交架构。
- 包括程序计数器、状态寄存器、和堆栈指针的完全寄存器访问。
- 单周期寄存器运行。
- 大尺寸 16 位寄存器文件减少了到存储器的取指令。
- 16 位地址总线可实现直接访问和整个存储器范围上的分支。
- 16 位数据总线可实现对字宽自变量的操作。
- 常量发生器提供最多六个立即值并少了代码尺寸。
- 无需中间寄存器保持的直接存储器到存储器传输。
- 字和字节寻址和指令格式。

图 3-1中显示了 CPU 的方框图。

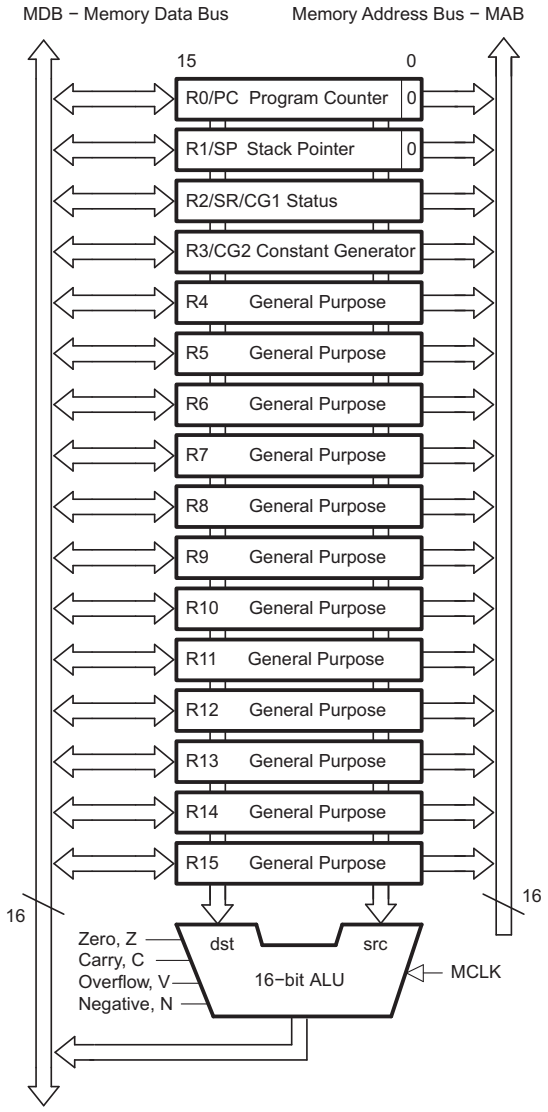


图 **3-1. CPU** 方框图

3.2 CPU 寄存器

CPU 包含十六个 16 位寄存器。 R0，R1，R2，和 R3 有专用功能。 R4 到 R15 是针对普通用途的工作寄存器。

3.2.1 程序计数器 (PC)

16 位程序计数器 (PC/R0) 指向将被执行的下一条指令。 每个指令使用偶数数量的字节（2 个，4 个或 6 个），并且 PC 相应的递增。 64KB 地址空间内的指令访问在字边界上执行，并且PC 与偶数地址对齐。 图 3-2显示了程序计数器。

图 **3-2.** 程序矜计数器

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
程序计数器位 15 至 1															0

可用所有指令和寻址模式对 PC 寻址。 几个示例：

```
MOV #LABEL,PC ; Branch to address LABELMOV LABEL,PC ; Branch to address contained in
LABELMOV @R14,PC ; Branch indirect to address in R14
```

3.2.2 堆栈指针 (SP)

堆栈指针 (SP/R1) 被 CPU 用来存储子例程调用和中断的返回地址。 它使用一个先递减、后递增机制。 此外，SP可由软件用所有指令和寻址模式使用。图 3-3显示了 SP。 SP 由用户初始化进 RAM，并且与偶数地址对齐。

图 3-4显示了堆栈用法。

图 3-3. 堆栈计数器

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
堆栈计数器位 15 至 1															0

```
MOV 2(SP),R6 ; Item I2 -
> R6MOV R7,0(SP) ; Overwrite TOS with R7PUSH #0123h ; Put 0123h onto TOSPOP R8 ; R8 =
0123h
```

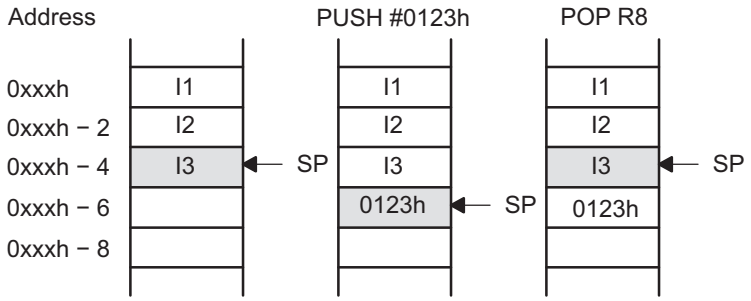


图 3-4. 堆栈用法

将 SP 用作一个到 PUSH 和 POP 指令的自变量的特殊情况在图 3-5中进行了说明和显示。

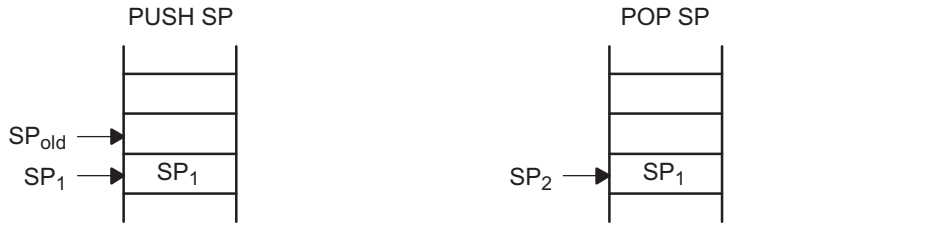


图 3-5. PUSH SP-POP SP序列

3.2.3 堆栈寄存器 (SR)

堆栈寄存器 (SR/R2)，被用作一个源或者目的寄存器，可被用在只能用字指令进行寻址的寄存器模式。 寻址模式的剩余组合被用来支持常量寄存器。图 3-6显示了 SR 位。

图 **3-6.** 状态寄存器位

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
被保留							V	SCG1	SCG0	OSC OFF	CPU OFF	GIE	N	Z	C
rw-0							rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

表 3-1描述了状态寄存器位。

表 **3-1.** 状态寄存器位的说明

位	说明
V	溢出位 当一个算术运算的结果溢出带符号变量范围时，这个位被置位。
	ADD(.B)和ADDC(.B) 在以下情况时置位： 正+正=负 负+负=负 否则复位
	SUB(.B)和SUBC(.B)和CMP(.B) 在以下情况时置位： 负-正=负 负-负=负 否则复位
SCG1	系统时钟生成器 1。当置位时，关闭 SMCLK。
SCG0	系统时钟生成器 0。当置位时，如果DCOCLK 未用于 MCLK 或 SMCLK，关闭 DCO dc 生成器。
OSCOFF	振荡器关闭。 当置位时，如果LFXT1CLK 没有被用于 MCLK 或 SMCLK，关闭 LFXT1 晶体振荡器。
CPUOFF	CPU 关闭。 当置位时，关闭 CPU。
GIE	通用中断使能。 当置位时，启用可屏蔽中断。 当置位时，所有可屏蔽中断被禁用。
N	负位。 当一个字节或者字运算的结果为负时置位，当结果不为负时清除。 字运算：N 被设定为结果的位 15 的值。 字节运算：N 被设定为结果的位 7 的值。
Z	零位。 当一个字节或字运算的结果为 0 时置位，当结果不为 0 时清除。
C	进位位。 当一个字节或字运算的结果产生一个进位时置位，并且当没有出现进位时清除。

3.2.4 常量发生器寄存器 CG1 和 CG2

常量发生器寄存器 R2和 R3 生成的六个常用常量，无需额外的 16 位程序代码字。 用源寄存器寻址模式 (As) 选择常量，如表 3-2所示。

表 **3-2.** 常量发生器 **CG1**，**CG2** 的值

寄存器	As	常量	注释
R2	00	– – – —	寄存器模式
R2	01	(0)	绝对地址模式
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

常量发生器的优势在于：

- 无需特殊指令
- 对于六个常量无需代码字
- 无需代码存储器访问来检索常量

如果六个常量中的一个被用作一个立即源操作数，汇编程序自动使用常量发生器。 寄存器 R2 和 R3，在常量模式中使用，不能被显式寻址；它们运行为只源寄存器。

3.2.4.1 常量发生器-扩展指令集

MSP430 的 RISC 指令集只有 27 条指令。 然而，常量发生器使得 MSP430 汇编程序支持 24 条附加，仿真指令。 例如，单操作数指令

```
CLR dst
```

用相同的长度仿真双操作数指令：

```
MOV R3,dst
```

其中 #0 被汇编程序取代，而 R3 在 As=00 时使用。

```
INC dst
```

被替换为：

```
ADD 0(R3),dst
```

3.2.5 通用寄存器 R4 至R15

12 个寄存器，R4-R15，为通用寄存器。 所有这些寄存器可被用作数据寄存器、地址指针、或者索引值并且可用字节或字指令进行访问，如图 3-7所示。

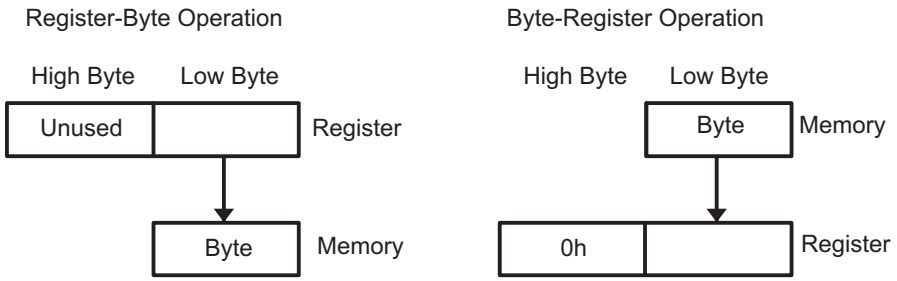


图 3-7. 寄存器字节/字节寄存器运行

示例寄存器字节运行
R5=0A28Fh
R6=0203h
Mem(0203h)=012h

```
ADD.B     R5,0(R6)
          08Fh
          +012h
          0A1h
```

Mem(0203h)=0A1h
C=0，Z=0，N=1
（寄存器的低字节）
+（已编址字节）
->（已编址字节）

示例字节寄存器运行
R5=01202h
R6=0223h
Mem(0223h)=05Fh

```
ADD.B     @R6,R5
          05Fh
          +002h
          00061h
```

R5=00061h
C=0，Z=0，N=0
（已编址字节）
+（寄存器的低字节）
->（寄存器的低字节，零至高字节）

3.3 寻址模式

针对源操作数的七个寻址模式和针对目的操作数的四个寻址模可在完整地址空间寻址。 表 3-3中的位数量描述了 As（源）和 Ad（目的）模式位的内容。

表 **3-3.** 源/目的操作数寻址模式

As/Ad	寻址模式	句法	说明
00/0	寄存器模式	Rn	寄存器内容是操作数
01/1	已索引模式	X(Rn)	(Rn+X) 指向操作数。 X 被存储在下一个字中。
01/1	符号模式	ADDR	(PC+X) 指向操作数。 X 被存储在下一个字中。 使用已索引模式 X(PC)。
01/1	绝对模式	&ADDR	这条指令后的字包含绝对地址。 X 被存储在下一个字中。 已索引模式 X (SR) 被使用。
10/-	间接寄存器模式	@Rn	Rn 被用作一个指向操作数的指针。
11/-	间接自动递增	@Rn+	Rn 被用作一个指向操作数的指针。 之后针对 .B 指令 Rn 递增 1，针对 .W指令，Rn 递增 2。
11/-	立即模式	#N	此指令之后的字包含立即常量 N。间接自动递增模式 @PC+ 被使用。

在下面的小节中详细解释了七个寻址模式。 大多数示例显示了针对源和目的的同样的寻址模式，但是在一个指令中可使用源和目的寻址模式的任一有效组合。

注: 标签***EDE，TONI，TOM 和 LEO 的使用***

在整个 MSP430 文档中，EDE，TONI，TOM 和 LEO 被用作普通标签。它们只是标签。 它们没有特别的含义。

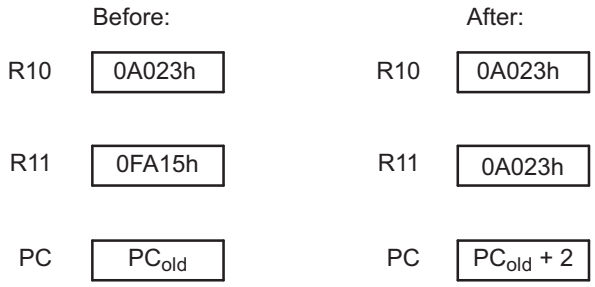
3.3.1 寄存器模式

在表 3-4中描述了寄存器模式。

表 **3-4.** 寄存器模式说明

汇编程序代码	**ROM** 内容
`MOV R10,R11`	`MOV R10,R11`

长度： 一个或两个字
运行： 将 R10 的内容移动到R11。 R10 不受影响。
注释： 对于源和目的有效
示例： `MOV R10,R11`



注: **寄存器中的数据**

寄存器中的数据可使用字或字节指令访问。 如果字节指令被使用，结果中高字节一直为 0。 根据字节指令的结果来处理状态位。

3.3.2 已索引模式

在表 3-5中描述了已索引模式。

表 **3-5.** 已索引模式说明

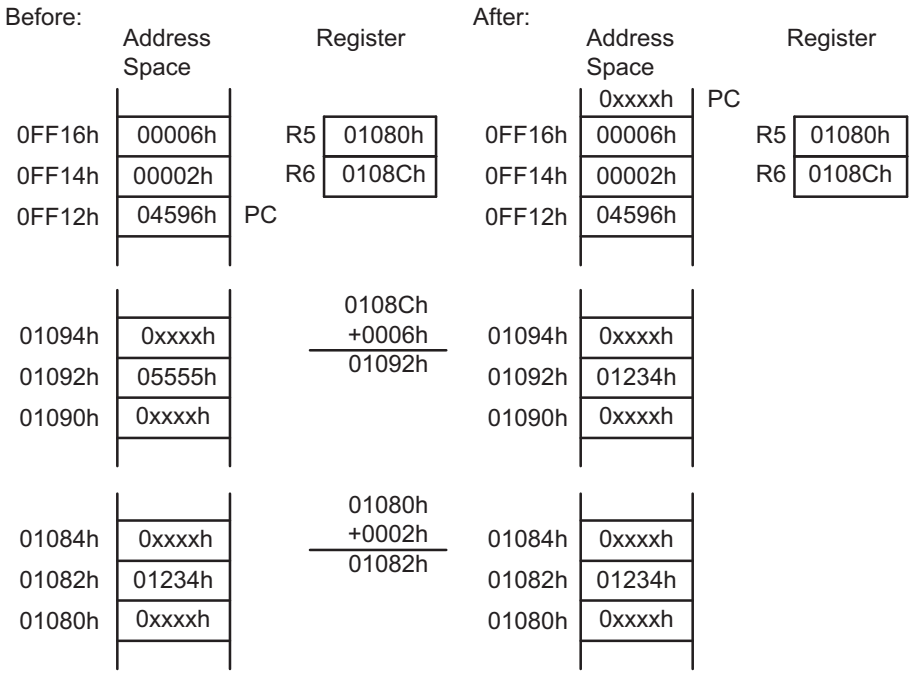
汇编程序代码	**ROM**的内容
MOV 2(R5)M6(R6)	MOV X(R5)MY(R6) X=2 Y=6

长度： 两个或三个字

运行： 移动源地址的内容（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。 在已索引模式中，程序计数器被自动递增，这样继续执行下一条程序指令。

注释： 针对源和目的有效

示例：
```
MOV 2(R5)M6(R6);
```



3.3.3 符号模式

在表 3-6中描述了符号模式。

表 **3-6.** 符号模式说明

汇编程序模式	**ROM**的内容
MOV EDEM TONI	MOV X(PC)M Y(PC) X=EDE-PC Y=TONI-PC

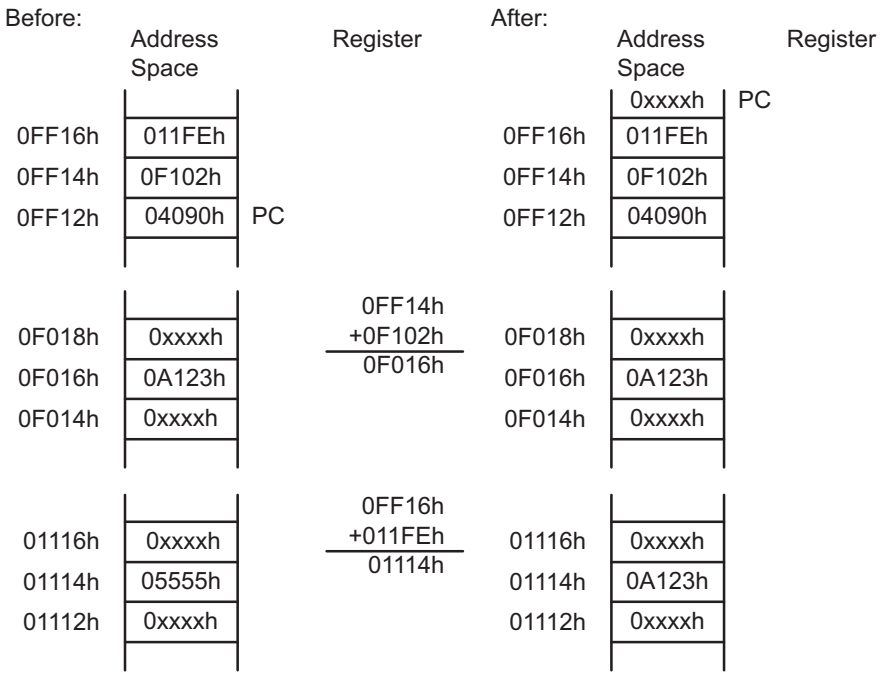
长度： 两个或三个字

运行： 移动源地址 EDE的内容（PC+X 的内容）到目的地址 TONI（PC+Y 的内容） 指令之后的字包含 PC 和源或目的地址间的差异。 汇编计算机并且自动插入偏移 X 和 Y。 借助于符号模式，程序计数器 (PC)被自动递增，这样继续执行下一条程序指令。

注释： 针对源和目的有效

示例：

```
MOV EDE,TONI ;Source address EDE = 0F016h;Dest. address TONI = 01114h
```



3.3.4 绝对模式

在表 3-7中对绝对模式进行了说明。

表 3-7. 绝对模式说明

汇编程序代码	ROM的内容
MOV &EDEM&TONI	MOV X(0)MY(0) X=EDE Y=TONI

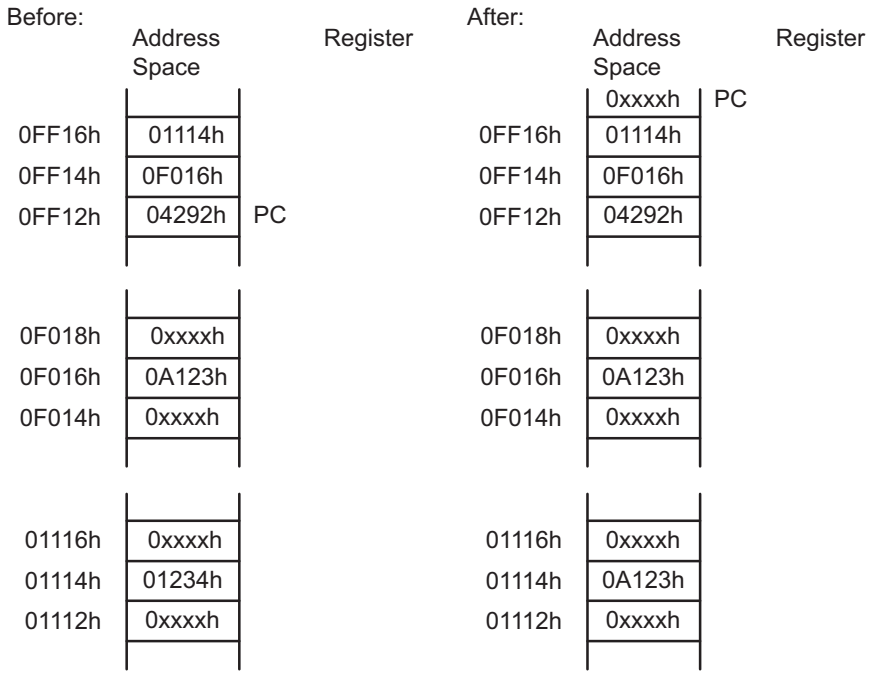
长度： 两个或三个字

运行： 移动源地址的内容到目的地址 TONI。 指令之后的字包含源和目的地址的绝对地址。 借助于绝对模式，PC 被自动递增，这样继续执行下一条程序指令。

注释： 针对源和目的有效

示例：

```
MOV &EDE,&TONI ;Source address EDE = 0F016h;Dest. address TONI = 01114h
```



这个寻址模式主要用于硬件外设模块，这些模块位于绝对、固定地址上。 这些是使用绝对模式的寻址来确保软件可移植性（例如，位置独立代码）。

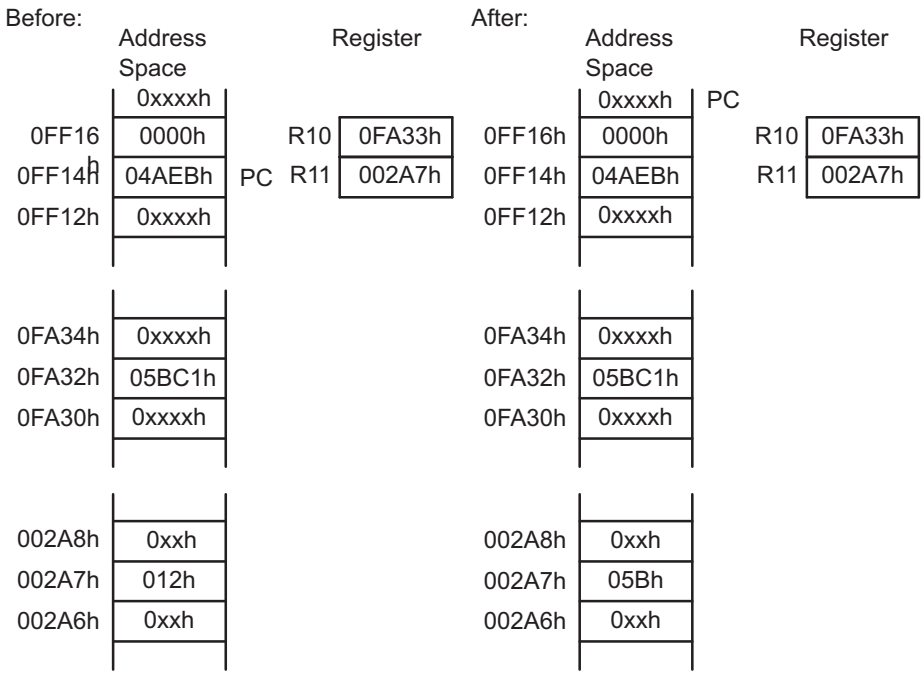
3.3.5 间接寄存器模式

在表 3-8中说明了间接寄存器模式。

表 **3-8.** 间接模式说明

汇编程序代码	**ROM**的内容
MOV @R10M0(R11)	MOV @R10M0(R11)

长度： 一个或两个字
运行： 移动源地址的内容（R10 的内容）到目的地址（R11 的内容）。 寄存器不被修改。
注释： 针对源操作数有效。 用 0(Rd) 替代目的操作数。
示例： MOV.B @R10M0(R11)



3.3.6 间接自动递增模式

在表 3-9中描述了间接自动递增。

表 3-9. 间接自动递增模式说明

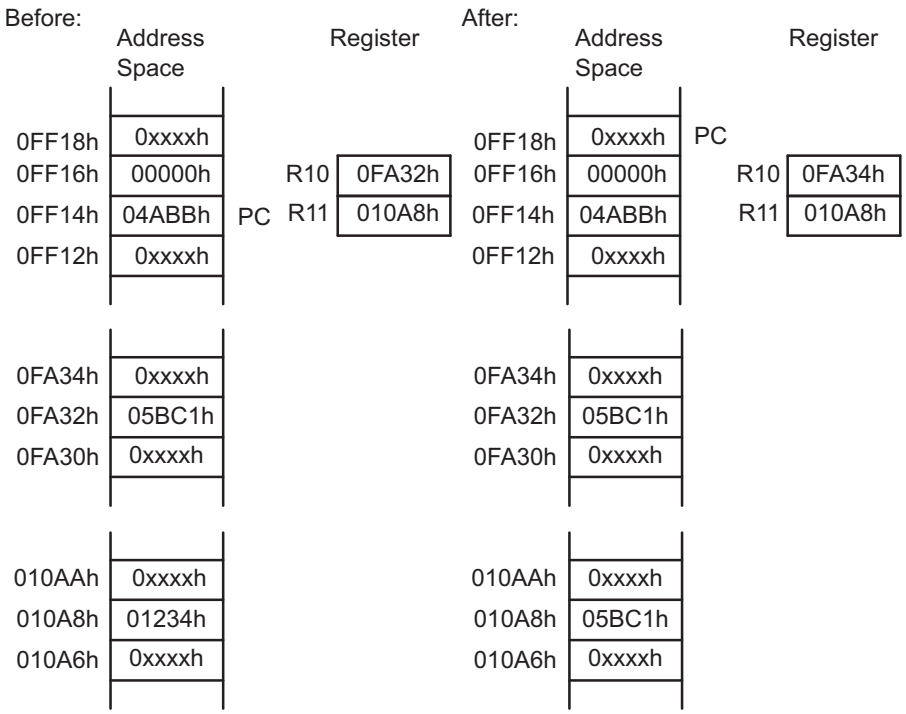
汇编程序代码	ROM 的内容
MOV @R10+M0(R11)	MOV @R10+M0(R11)

长度： 一个或两个字

运行： 移动源地址的内容（R10 的内容）到目的地址（R11 的内容）。 对于一个字节操作，寄存器 R10 被递增 1，或者在取指令之后针对字操作递增 2；它在无需开销的情况下指向下一个地址。 这对于表处理十分有用。

注释： 针对源操作数有效。 用 0(Rd) 加上第二指令 INCD Rd 替代目的操作数。

示例： MOV @R10+M0(R11)



在操作数被取出后，寄存器自动增量发生。图 3-8显示了这一过程。

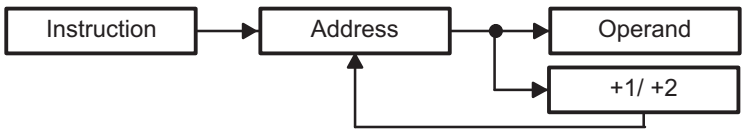


图 3-8. 操作数取操作

3.3.7 立即模式

在表 3-10中描述了立即模式。

表 3-10. 立即模式说明

汇编程序代码	ROM的内容
MOV #45hMTONI	MOV @PC+MX(PC) 45 X=TONI-PC

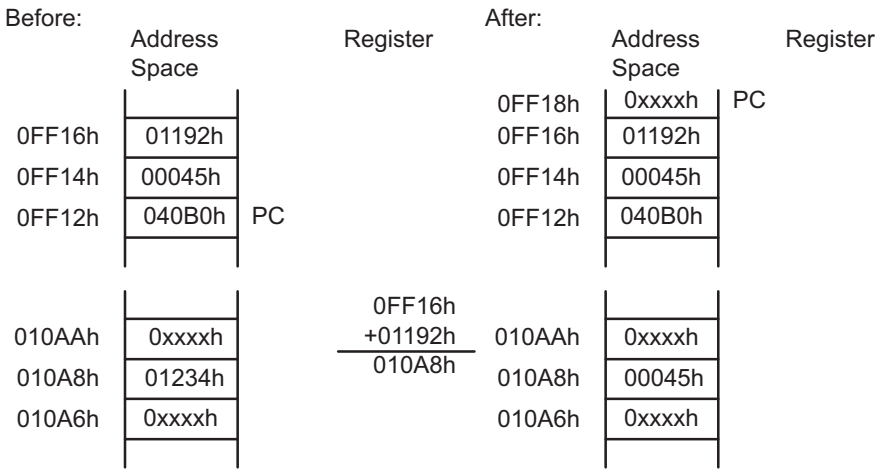
长度： 两个或三个字

如果 CG1 或 CG2 的内容可被使用，则少一个字。

运行： 移动包含在指令后的字中的立即内容45h 到目的地址 TONI。 当取源时，程序计数器指向指令后的字并且将内容移动到目的。

注释： 针对一个源操作数有效。

示例： MOV #45hMTONI



3.4 指令集

完整 MSP430 指令集包含 27 条内核指令和 24 个仿真指令。 内核指令是具有唯一运行代码（由 CPU 解码）的指令。仿真指令是简化写入和读取的指令，但是本身不带有运行代码，反之，它们自动被汇编程序用等效的内核指令所取代。 使用仿真指令不会影响代码或性能。

有三个内核指令格式：

- 双操作数
- 单操作数
- 跳转

通过使用 .B 或 .W 扩展名，所有单操作数和双操作数指令可以为字节或字指令。 字节指令可被用于访问字节数据或字节外设。 字指令被用于访问字数据或字外设。 如果不使用扩展名，指令是一个字指令。

一个指令的源和目的由以下字段定义：

src	源操作数由 As 和 S-reg 定义
dst	目的操作数由 Ad 和 D-seg 段定义
As	寻址位负责源 (src) 使用的寻址模式。
S-reg	针对源 (src) 的工作寄存器
Ad	寻址位负责用于目的 (dst) 的寻址模式
D-reg	针对目的 (dst) 的工作寄存器
B/W	字节或字操作： 0：字操作 1：字节操作

注： **目的地址**

在存储器映射的任一位置目的地址有效。 然而，当使用一个修改目的内容的指令时，用户必须确保目的地址可写入。 例如，一个被屏蔽的 ROM 位置将是一个有效的目的地址，但是内容不可修改，所有指令的结果将丢失。

3.4.1 双操作数（格式 I）指令

图 3-9解释了双操作数指令格式。

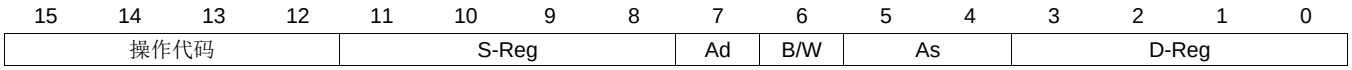


图 **3-9.** 双操作数指令格式

表 3-11列出并说明了双操作数指令。

表 **3-11.** 双操作数指令

助记符	**S-Reg，D-Reg**	运行	状态位			
			V	**N**	**Z**	**C**
MOV(.B)	src,dst	src→dst	-	-	-	-
ADD(.B)	src,dst	src+dst→dst	*	*	*	*
ADDC(.B)	src,dst	src+dst+C→dst	*	*	*	*
SUB(.B)	src,dst	dst+.not.src+1→dst	*	*	*	*
SUBC(.B)	src,dst	dst+.not.src+C→dst	*	*	*	*
CMP(.B)	src,dst	dst-src	*	*	*	*
DADD(.B)	src,dst	src+dst+C→dst（十进制）	*	*	*	*
BIT(.B)	src,dst	src .and. dst	0	*	*	*
BIC(.B)	src,dst	not.src.and. dst→dst	-	-	-	-
BIS(.B)	src,dst	src .or. dst→dst	-	-	-	-
XOR(.B)	src,dst	src .xor. dst→dst	*	*	*	*
AND(.B)	src,dst	src.and. dst→dst	0	*	*	*

* 状态位被影响
\- 状态位不受影响
0 状态位被清除
1 状态位被置位

注: 指令CMP 和SUB

指令CMP和SUB完全一样，除了存储结果。 对于BIT和AND指令也是如此。

3.4.2 单操作数（格式 II）指令

图 3-10解释了单操作数指令格式。

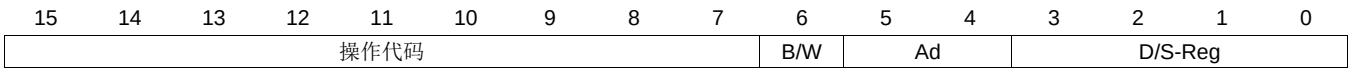


图 3-10. 单操作数指令格式

表 3-12列出并说明了单操作数指令。

表 3-12. 单操作数指令

助记符	S-Reg，D-Reg	运行	状态位			
			V	N	Z	C
RRC(.B)	dst	C→MSB→.....LSB→C	*	*	*	*
RRA(.B)	dst	MSB→MSB→....LSB→C	0	*	*	*
PUSH(.B)	src	SP-2→SP，src→@SP	-	-	-	-
SWPB	dst	交换字节	-	-	-	-
CALL	dst	SP-2→SP，PC+2→@SP dst→PC	-	-	-	-
RETI		TOS→SR，SP+2→SP TOS→PC，SP+2→SP	*	*	*	*
SXT	dst	位 7 → 位8.....位 15	0	*	*	*

* 状态位被影响
– 状态位不受影响
0 状态位被清除
1 状态位被置位

CALL 指令可使用所有寻址模式。 如果使用符号模式（地址）、立即模式 (#N)、绝对模式 (&EDE) 或者已索引模式 x(RN)，之后的字包含地址信息。

3.4.3 跳转

图 3-11显示了条件跳转指令格式。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
操作代码			C			10 为 PC 偏移									

图 3-11. 跳转指令格式

表 3-13列出并说明了跳转指令。

表 3-13. 跳转指令

助记符	S-Reg，D-Reg	运行
JEQ/JZ	Label	如果零位被置位则跳转至标签
JNE/JNZ	Label	如果零位被复位则跳转至标签
JC	Label	如果进位位被置位则跳转至标签
JNC	Label	如果进位位被复位则跳转至标签
JN	Label	如果负位被置位则跳转至标签
JGE	Label	如果(N .XOR. V)=0 则跳转到标签
JL	Label	如果(N .XOR. V)=1 则跳转到标签
JMP	Label	无条件跳转到标签

条件跳转支持相对于 PC 的程序分支，并且不影响状态位。 相对于跳转指令上的 PC 值，可能的跳转范围为 -511 到 +512 字之间。 10 位程序计数器偏移被认为是一个带符号的 10 位值，此值被加倍并且加到程序计数器中。

$PC_{新} = PC_{之前} + 2 + PC_{偏移} \times 2$

3.4.4 指令周期和长度

一条指令所需的 CPU 时钟周期取决于指令格式和使用的寻址模式-而非指令本身。 时钟周期的数量参考 MCLK。

3.4.4.1 中断和复位周期

表 3-14列出了用于中断开销和复位的 CPU 周期。

表 3-14. 中断和复位周期

操作	周期的数量	指令的长度
从中断 (RETI) 返回	5	1
接受的中断	6	-
WDT复位	4	-
复位($\overline{\text{RST}}$/NMI)	4	-

3.4.4.2 格式 II（单操作数）指令周期和长度

表 3-15列出了所有格式 II指令寻址模式的长度和 CPU 周期。

表 3-15. 格式 II 指令周期和长度

	周期的数量				
寻址模式	**RRA，RRC SWPB，SXT**	压栈	调用	指令的长度	示例
Rn	1	3	4	1	SWPB R5
@Rn	3	4	4	1	RRC @R9
@Rn+	3	5	5	1	SWPB @R10+
#N	（请见注释）	4	5	2	MM #0F000h
X(Rn)	4	5	5	2	MM 2(R7)
EDE	4	5	5	2	MM EDE
&EDE	4	5	5	2	SXT &EDE

注: **指令格式 II 立即模式**

不要在目的字段中使用带有立即模式的 RRA，RRC，SWPB和SXT 在立即模式中使用这些指令会导致一个无法预计的程序运行。

3.4.4.3 格式 II（跳转）指令周期和长度

所有跳转指令要求一个代码字，并且花费两个 CPU 周期来执行，无论是否发生跳转。

3.4.4.4 格式 I（双操作数）指令周期和长度

表 3-16列出了所有格式 I指令寻址模式的长度和 CPU 周期。

表 3-16. 格式 I 指令周期和长度

寻址模式					
Src	**Dst**	周期的数量	指令的长度		示例
Rn	Rm	1	1	MOV	R5M R8
	PC	2	1	BR	R9
	x(Rm)	4	2	M M	R5M 4(R6)
	EDE	4	2	M M	R8M EDE
	&EDE	4	2	MOV	R5M &EDE
@Rn	Rm	2	1	M	@R4M R5
	PC	2	1	BR	@R8
	x(Rm)	5	2	M M	@R5M 8(R6)
	EDE	5	2	MOV	@R5M EDE
	&EDE	5	2	M M	@R5M &EDE
@Rn+	Rm	2	1	M M	@R5+M R6
	PC	3	1	BR	@R9+
	x(Rm)	5	2	M M	@R5M 8(R6)
	EDE	5	2	MOV	@R9+M EDE
	&EDE	5	2	M M	@R9+M &EDE
#N	Rm	2	2	M M	#20M R9
	PC	3	2	BR	#2AEh
	x(Rm)	5	3	M M	#0300hM 0(SP)
	EDE	5	3	M M	#33M EDE
	&EDE	5	3	M M	#33M &EDE
x(Rn)	Rm	3	2	M M	2(R5)M R7
	PC	3	2	BR	2(R6)
	TONI	6	3	M M	4(R7)M TONI
	x(Rm)	6	3	M M	4(R4)M 6(R9)
	&TONI	6	3	M M	2(R4)M &TONI
EDE	Rm	3	2	M	EDEM R6
	PC	3	2	BR	EDE
	TONI	6	3	CMP	EDEM TONI
	x(Rm)	6	3	M M	EDEM 0(SP)
	&TONI	6	3	M M	EDEM &TONI
&EDE	Rm	3	2	M M	&EDEM R8
	PC	3	2	BRA	&EDE
	TONI	6	3	M M	&EDEM TONI
	x(Rm)	6	3	M M	&EDEM 0(SP)
	&TONI	6	3	M M	&EDEM &TONI

3.4.5 指令集说明

指令映射显示在图 3-12中，而完整指令集汇总于表 3-17中。

	000	040	080	0C0	100	140	180	1C0	200	240	280	2C0	300	340	380	3C0
0xxx																
4xxx																
8xxx																
Cxxx																
1xxx	RRC	RRC.B	SWPB		RRA	RRA.B	SXT		PUSH	PUSH.B	CALL		RETI			
14xx																
18xx																
1Cxx																
20xx	JNE/JNZ															
24xx	JEQ/JZ															
28xx	JNC															
2Cxx	JC															
30xx	JN															
34xx	JGE															
38xx	JL															
3Cxx	JMP															
4xxx	MOV, MOV.B															
5xxx	ADD, ADD.B															
6xxx	ADDC, ADDC.B															
7xxx	SUBC, SUBC.B															
8xxx	SUB, SUB.B															
9xxx	CMP, CMP.B															
Axxx	DADD, DADD.B															
Bxxx	BIT, BIT.B															
Cxxx	BIC, BIC.B															
Dxxx	BIS, BIS.B															
Exxx	XOR, XOR.B															
Fxxx	AND, AND.B															

图 3-12. 内核指令映射

表 3-17. MSP430 指令集

助记符		说明		V	N	Z	C
ADC(.B) (1)	dst	将 C 加至目的	dst+C→dst	*	*	*	*
ADD(.B)	srcMdst	将源加至目的	src+dst→dst	*	*	*	*
ADDC(.B)	srcMdst	将源和 C 加至目的	src+dst+C→dst	*	*	*	*
AND(.B)	srcMdst	将源和目的进行与操作	src.and. dst→dst	0	*	*	*
BIC(.B)	srcMdst	清除目的中的位	not.src.and.dst→dst	-	-	-	-
BIS(.B)	srcMdst	设定目的中的位	src.or. dst→dst	-	-	-	-
BIT(.B)	srcMdst	测试目的中的位	src.and.dst	0	*	*	*
BR (1)	dst	分支至目的	dst→PC	-	-	-	-
MM	dst	调用目的	PC+2→stack，dst→PC	-	-	-	-
CLR(.B) (1)	dst	清零目的	0→dst	-	-	-	-
CLRC (1)		清零 C	0→C	-	-	-	0
CLRN (1)		清零 N	0→N	-	0	-	-
CLRZ (1)		清零 Z	0→Z	-	-	0	-
CMP(.B)	srcMdst	比较源和目的	dst-src	*	*	*	*
DADC(.B) (1)	dst	十进制加 C到目的	dst+C→dst（十进制）	*	*	*	*
DADD(.B)	srcMdst	将源和 C 十进制加入dst	src+dst+C→dst（十进制）	*	*	*	*

(1) 仿真指令

表 3-17. MSP430 指令集 (continued)

助记符		说明		V	N	Z	C
DEC(.B) (1)	dst	递减目的	dst-1→dst	*	*	*	*
DECD(.B) (1)	dst	双递减目的	dst-2→dst	*	*	*	*
DINT (1)		禁用中断	0→GIE	-	-	-	-
EINT (1)		启用中断	1→GIE	-	-	-	-
INC(.B) (1)	dst	递增目的	dst+1→dst	*	*	*	*
INCD(.B) (1)	dst	双递增目的	dst+2→dst	*	*	*	*
INV(.B) (1)	dst	反转目的	.not.dst→dst	*	*	*	*
JC/JHS	MM	如果 C置位则跳转，如果高于或相等则跳转		-	-	-	-
JEQ/JZ	MM	如果等于则跳转，如果 Z 被置位则跳转		-	-	-	-
JGE	MM	如果大于或等于则跳转		-	-	-	-
JL	MM	如果少于则跳转		-	-	-	-
JMP	MM	跳转	PC+2×offset→PC	-	-	-	-
JN	MM	如果 N被置位则跳转		-	-	-	-
JNC/JLO	MM	如果 C 未被设定则跳转，如果低于则跳转		-	-	-	-
JNE/JNZ	MM	如果不等于则跳转，如果 Z 未被置位则跳转		-	-	-	-
MOV(.B)	srcMdst	将源移动到目的	src→dst	-	-	-	-
NOP (2)		无操作		-	-	-	-
POP(.B) (2)	dst	将项目从堆栈弹出至目的	@SP→dst，SP+2→SP	-	-	-	-
PUSH(.B)	src	将源压入堆栈	SP-2→SP，src→@SP	-	-	-	-
RET (2)		从目的返回	@SP→PC，SP+2→SP	-	-	-	-
RETI		从中断返回		*	*	*	*
RLA(.B) (2)	dst	算术左旋转		*	*	*	*
RLC(.B) (2)	dst	通过 C左旋转		*	*	*	*
RRA(.B)	dst	算术右旋转		0	*	*	*
RRC(.B)	dst	通过 C右旋转		*	*	*	*
SBC(.B) (2)	dst	将 非 (C)从目的中减去	dst+0FFFFh+C→dst	*	*	*	*
SETC (2)		置位 C	1→C	-	-	-	1
SETN (2)		置位 N	1→N	-	1	-	-
SETZ (2)		置位 Z	1→Z	-	-	1	-
SUB(.B)	srcMdst	从目的中减去源	dst+.not.src+1→dst	*	*	*	*
SUBC(.B)	srcMdst	从 dst 中减去源和非 (C)	dst+.not.src+C→dst	*	*	*	*
SWPB	dst	交换字节		-	-	-	-
SXT	dst	扩展符		0	*	*	*
TST(.B) (2)	dst	测试目的	dst+0FFFFh+1	0	*	*	1
XOR(.B)	srcMdst	异或源和目的	src .xor. dst→dst	*	*	*	*

(2) 仿真指令

3.4.6 指令集细节

3.4.6.1 ADC

***ADC[.W]**	将进位加至目的
***ADC.B**	将进位加至目的
句法	`ADC dst or ADC.W dst ADC.B dst`
运行	dst+C→dst
仿真	`ADDC #0,dst ADDC.B #0,dst`
说明	进位位 (C) 被加入到目的操作数中。 目的之前的内容丢失。
状态位	N：如果结果为负则置位，否则复位 Z：如果结果为零则置位，否则复位 C：如果 dst 从 0FFFFh 到 0000 递增则置位，否则复位 如果 dst 从 0FFh 到 00 递增则置位，否则复位 V：如果一个算术溢出发生则置位，否则复位
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	R13 指向的 16 位计数器被加至R12 指向的 32 位计数器。 `ADD @R13,0(R12) ; Add LSDsADC 2(R12) ; Add carry to MSD`
示例	R13 指向的 8 位计数器被加至R12 指向的 16 位计数器。 `ADD.B @R13,0(R12) ; Add LSDsADC.B 1(R12) ; Add carry to MSD`

3.4.6.2 加

ADD[.W]	将源加至目的
ADD.B	将源加至目的
句法	`ADD src,dst or ADD.W src,dst` `ADD.B src,dst`
运行	src+dst→dst
说明	源操作数被加至目的操作数。 源操作数不受影响。 目的之前的内容丢失。
状态位	N：如果结果为负则置位，否则复位 Z：如果结果为零则置位，否则复位 C：如果结果有一个进位则置位，否则清零 V：如果一个算术溢出发生则置位，否则复位
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	R5 增加 10。 进位时执行到 TONI 的跳转。 `ADD #10,R5JC TONI ; Carry occurred..... ; No carry`
示例	R5 增加 10。 进位时执行到 TONI 的跳转。 `ADD.B #10,R5 ; Add 10 to Lowbyte of R5JC TONI ; Carry occurred, if (R5) ≥ 246 [0Ah+0F6h]..... ; No carry`

3.4.6.3 ADDC

ADDC[.W]	将源和进位加至目的
ADDC.B	将源和进位加至目的
句法	`ADDC src,dst or ADDC.W src,dst ADDC.B src,dst`
运行	src+dst+C→dst
说明	源操作数和进位位 (C) 被加至目的操作数。源操作数不受影响。 目的之前的内容丢失。
状态位	N：如果结果为负则置位，否则复位 Z：如果结果为零则置位，否则复位 C：如果结果的 MSB 有一个进位则置位，否则复位 V：如果一个算术溢出发生则置位，否则复位
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	R13 指向的 32 位计数器被加至 32 位计数器，R13 内指针之上的 11 个字(20/2+2/2)。 `ADD @R13+,20(R13) ; ADD LSDs with no carry inADDC @R13+,20(R13) ; ADD MSDs with carry... ; resulting from the LSDs`
示例	R13 指向的 24 位计数器被加至 24 位计数器，R13 内指针之上的 11 个字。 `ADD.B @R13+,10(R13) ; ADD LSDs with no carry inADDC.B @R13+,10(R13) ; ADD medium Bits with carryADDC.B @R13+,10(R13) ; ADD MSDs with carry... ; resulting from the LSDs`

3.4.6.4 与

AND[.W]	源和目的进行与操作
AND.B	源和目的进行与操作
句法	`AND src,dst or AND.W src,dst AND.B src,dst`
运行	`src .AND. dst → dst`
说明	源操作数和目的操作数进行逻辑与操作。 结果被放置在目的中。
状态位	N：如果结果 MSB 被设定则置位，如果未被设定则复位 Z：如果结果为零则置位，否则复位 C：如果结果不为零则置位，否则复位 (=.NOT. 零位) V：复位
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	R5 中的位被用作一个针对由 TOM 寻址的字的掩码 (#0AA55h)。如果结果为零，一个分支指令将指向标签 TONI。 `MOV #0AA55h,R5 ; Load mask into register R5AND R5,TOM ; mask word addressed by TOM with R5JZ TONI ; ..... ; Result is not zero;;; or;;AND #0AA55h,TOMJZ TONI`
示例	掩码 #0A5h 的位于 TOM 的低字节逻辑与。 如果结果为零，一个分支指令被指向标签 TONI。 `AND.B #0A5h,TOM ; mask Lowbyte TOM with 0A5hJZ TONI ; ..... ; Result is not zero`

3.4.6.5 BIC

BIC[.W]	清除目的中的位
BIC.B	清除目的中的位
句法	`BIC src,dst or BIC.W src,dst BIC.B src,dst`
运行	.NOT.src .AND. dst→dst
说明	被反转的源操作数与目的操作数进行逻辑与。结果被放置在目的中。 源操作数不受影响。
状态位	状态位不受影响。
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	RAM 字 LEO 的六个 MSB 被清零。 `BIC #0FC00h,LEO ; Clear 6 MSBs in MEM(LEO)`
示例	RAM 字节 LEO 的五个 MSB 被清零。 `BIC.B #0F8h,LEO ; Clear 5 MSBs in Ram location LEO`

3.4.6.6 BIS

BIS[.W]	设定目的中的位
BIS.B	设定目的中的位
句法	`BIS src,dst or BIS.W src,dst BIS.B src,dst`
运行	src.OR. dst→dst
说明	源操作数和目的操作数被逻辑与。 结果被放置在目的中。 源操作数不受影响。
状态位	状态位不受影响。
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	RAM 字 TOM 的六个 LSB 被置位。 `BIS #003Fh,TOM ; set the six LSBs in RAM location TOM`
示例	RAM 字节 TOM 的三个 MSB 被置位。 `BIS.B #0E0h,TOM ; set the 3 MSBs in RAM location TOM`

3.4.6.7 位

BIT[.W]	测试目的中的位
BIT.B	测试目的中的位
句法	`BIT src,dst or BIT.W src,dst`
运行	src.AND. dst
说明	源和目的操作数被逻辑与。 结果至影响状态位。 源和目的操作数不受影响。
状态位	N：如果结果的 MSB 被设定则置位，否则复位 Z：如果结果为零则置位，否则复位 C：如果结果不为零则复位，否则复位 (.NOT. 零位) V：复位
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	如果 R8 的位 9 被设定，一个分支指令被指向标签 TOM。 `BIT #0200h,R8 ; bit 9 of R8 set?JNZ TOM ; Yes, branch to TOM... ; No, proceed`
示例	如果 R8 的位 3 被设定，一个分支指令被指向标签 TOM。 `BIT.B #8,R8JC TOM`
示例	一个串行通信接收位 (RCV) 被测试。 由于进位位在使用 BIT 指令来测试一个单一位时等于被测试位的状态，进位位被随后的指令使用；读取指令信息被移入寄存器RECBUF。 `; ; Serial communication with LSB is shifted first:; xxxx xxxx xxxx xxxxBIT.B #RCV,RCCTL ; Bit info into carryRRC RECBUF ; Carry -> MSB of RECBUF; cxxx xxxx..... ; repeat previous two instructions..... ; 8 times; cccc cccc; ^ ^; MSB LSB; Serial communication with MSB shifted first:BIT.B #RCV,RCCTL ; Bit info into carryRLC.B RECBUF ; Carry -> LSB of RECBUF; xxxx xxxc..... ; repeat previous two instructions..... ; 8 times; cccc cccc; | ; MSB LSB`

3.4.6.8 BR，BRANCH

***BR，BRANCH**	分支指令到..... 目的
句法	`BR dst`
运行	dst→PC
仿真	`MOV dst,PC`
说明	一个无条件分支指令被指向 64K 地址空间内的任一地址。 可使用所有源寻址模式。 分支指令是一个字指令。
状态位	状态位不受影响。
示例	给出了所有寻址模式的示例。

```
BR #EXEC ; Branch to label EXEC or direct branch (e.g. #0A4h); Core instruction
MOV @PC+,PCBR EXEC ; Branch to the address contained in EXEC; Core instruction
MOV X(PC),PC; Indirect addressBR &EXEC ; Branch to the address contained in
absolute ; address EXEC; Core instruction MOV X(0),PC; Indirect addressBR R5 ;
Branch to the address contained in R5; Core instruction MOV R5,PC; Indirect R5BR
@R5 ; Branch to the address contained in the word ; pointed to by R5.; Core
instruction MOV @R5+,PC; Indirect, indirect R5BR @R5+ ; Branch to the address
contained in the word pointed ; to by R5 and increment pointer in R5 afterwards.;
The next time--S/W flow uses R5 pointer--
it can ; alter program execution due to access to; next address in a table
pointed to by R5; Core instruction MOV @R5,PC; Indirect, indirect R5 with
autoincrementBR X(R5) ; Branch to the address contained in the address ; pointed
to by R5 + X (e.g. table with address ; starting at X). X can be an address or a
label; Core instruction MOV X(R5),PC; Indirect, indirect R5 + X
```

3.4.6.9 调用

调用	子例程
句法	`CALL dst`
运行	dst→tmp dst 被评估和保存 SP-2→SP PC→@SP PC 被更新至 TOS tmp→PC dst 被保存至 PC
说明	在 64K 地址空间内的任一地址上进行子例程调用。 所有寻址模式均可使用。 返回地址（之后指令的地址）被存储在堆栈上。 调用指令是一个字指令。
状态位	状态位不受影响。
示例	给出了所有寻址模式的示例。

```
CALL #EXEC ; Call on label EXEC or immediate address (e.g. #0A4h); SP-2 -
> SP, PC+2 -> @SP, @PC+ -
> PCCALL EXEC ; Call on the address contained in EXEC; SP-2 -> SP, PC+2 -
>SP, X(PC) -
> PC; Indirect addressCALL &EXEC ; Call on the address contained in
absolute address ; EXEC; SP-2 -> SP, PC+2 -> @SP, X(0) -
> PC; Indirect addressCALL R5 ; Call on the address contained in R5; SP-
2 -> SP, PC+2 -> @SP, R5 -
> PC; Indirect R5CALL @R5 ; Call on the address contained in the word ;
pointed to by R5; SP-2 -> SP, PC+2 -> @SP, @R5 -
> PC; Indirect, indirect R5CALL @R5+ ; Call on the address contained in
the word ; pointed to by R5 and increment pointer in R5.; The next time
S/W flow uses R5 pointer; it can alter the program execution due to;
access to next address in a table pointed to by R5; SP-2 -> SP, PC+2 -
> @SP, @R5 -
> PC; Indirect, indirect R5 with autoincrementCALL X(R5) ; Call on the
address contained in the address pointed; to by R5 + X (e.g. table with
address starting at X); X can be an address or a label; SP-2 -
> SP, PC+2 -> @SP, X(R5) -> PC; Indirect, indirect R5 + X
```

3.4.6.10 CLR

***CLR[.W]**	清零目的
***CLR.B**	清零目的
句法	`CLR dst or CLR.W dst CLR.B dst`
运行	0→dst
仿真	`MOV #0,dst MOV.B #0,dst`
说明	目的操作数被清零。
状态位	状态位不受影响。
示例	RAM 字 TONI 被清零。 `CLR TONI ; 0 -> TONI`
示例	寄存器 R5 被清零。 `CLR R5`
示例	RAM 字节 TONI 被清零。 `CLR.B TONI ; 0 -> TONI`

3.4.6.11 CLRC

***CLRC**	清零进位位
句法	`CLRC`
运行	0→C
仿真	`BIC #1,SR`
说明	进位位 (C) 被清零。 清零进位指令是一条字指令。
状态位	N：不受影响 Z：不受影响 C：被清零 V：不受影响
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	R13 指向的 16 位计数器被加至R12 指向的 32 位计数器。 `CLRC ; C=0: defines startDADD @R13,0(R12) ; add 16=bit counter to low word of 32=bit counterDADC 2(R12) ; add carry to high word of 32=bit counter`

3.4.6.12 CLRN

***CLRN**	清零负位
句法	`CLRN`
运行	0→N 或者 (.NOT.src .AND. dst→dst)
仿真	`BIC #4,SR`
说明	常量 04h 被反转 (0FFFBH) 并且与目的操作数逻辑与。 结果被放置在目的中。 清零负位的指令是一条字指令。
状态位	N：复位为0 Z：不受影响 C：不受影响 V：不受影响
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	状态寄存器内的负位被清零。 这避免了对被调用子例程负数的特殊处理。 `CLRNCALL SUBR.....SUBR JN SUBRET ; If input is negative: do nothing and return.....SUBRET RET`

3.4.6.13 CLRZ

***CLRZ**	清零零位
句法	`CLRZ`
运行	0→Z 或 (.NOT.src .AND. dst→dst)
仿真	`BIC #2,SR`
说明	常量 02h 被反转 (0FFFDH) 并且与目的操作数逻辑与。 结果被放置在目的中。 清零零位的指令是一条字指令。
状态位	N：不受影响 Z：复位至 0 C：不受影响 V：不受影响
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	状态寄存器内的零位被清零。 `CLRZ`

3.4.6.14 CMP

CMP[.W]	比较源和目的
CMP.B	比较源和目的
句法	`CMP src,dst or CMP.W src,dst CMP.B src,dst`
运行	dst+.NOT.src+1 或者 (dst-src)
说明	从目的操作数中减去目的操作数。 通过把源操作数的 1s 补数加 1 来完成此操作。由于结果不保存，两个操作数不受影响；只用状态位受影响。
状态位	N：如果结果为负则置位，如果为正则复位 (src≥dst) Z：如果结果为零则置位，否则复位 (src=dst) C：如果结果的 MSB 有一个进位则置位，否则复位 V：如果一个算术溢出发生则置位，否则复位
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	R5 和 R6 被比较。 如果它们相等，程序继续在标签EQUAL 上执行。 `CMP R5,R6 ; R5 = R6?JEQ EQUAL ; YES, JUMP`
示例	两个 RAM 块被比较。 如果它们不相等，程序分支到标签 ERROR。 `MOV #NUM,R5 ; number of words to be comparedMOV #BLOCK1,R6 ; BLOCK1 start address in R6MOV #BLOCK2,R7 ; BLOCK2 start address in R7L$1 CMP @R6+,0(R7) ; Are Words equal? R6 incrementsJNZ ERROR ; No, branch to ERRORINCD R7 ; Increment R7 pointerDEC R5 ; Are all words compared?JNZ L$1 ; No, another compare`
示例	由 EDE 和 TONI 寻址的 RAM 字节被比较。 如果它们相等，程序继续在标签 EQUAL 上执行。 `CMP.B EDE,TONI ; MEM(EDE) = MEM(TONI)?JEQ EQUAL ; YES, JUMP`

3.4.6.15 DADC

***DADC[.W]**	将十进制进位加至目的
***DADC.B**	将十进制进位加至目的
句法	`DADC dst or DADC.W src,dst DADC.B dst`
运行	dst+C→dst（用十进制）
仿真	`DADD #0,dst DADD.B #0,dst`
说明	十进制进位位 (C) 被加入到目的操作数中。
状态位	N：如果 MSB 为 1 则置位 Z：如果 dst 为零则置位，否则复位 C：如果目的从 9999 到 0000 递增则置位，否则复位 如果目的从 99 至 00 递增则置位，否则复位 V：未定义
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	包含在 R5 中的四位十进制数被加至 R8指向的一个八位十进制数。 `CLRC ; Reset carry; next instruction's start condition is definedDADD R5,0(R8) ; Add LSDs + CDADC 2(R8) ; Add carry to MSD`
示例	包含在 R5 中的两位十进制数被加至 R8 指向的一个四位十进制数。 `CLRC ; Reset carry; next instruction's start condition is definedDADD.B R5,0(R8) ; Add LSDs + CDADC.B 1(R8) ; Add carry to MSDs`

3.4.6.16 DADD

DADD[.W]	十进制源和进位被加至目的
DADD.B	十进制源和进位被加至目的
句法	`DADD src,dst or DADD.W src,dst DADD.B src,dst`
运行	src+dst+C→dst（用十进制）
说明	源操作数和目的操作数被认为是四个带符号的二进制编码的十进制 (BCD)。 十进制的源操作数和进位位 (C) 被加至目的操作数。 源操作数不受影响。 目的之前的内容丢失。 对于 BCD 数，此结果未定义。
状态位	N：如果 MSB 为 1 则置位，否则复位 Z：如果结果为零则置位，否则复位 C：如果结果大于 9999 则置位 如果结果大于 99 则置位 V：未定义
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	包含在 R5 和 R6 中的八位 BCD 数被用十进制的方法加至包含在 R3 和 R4（包含在 MSD 中的 R6 和 R4）的一个八位 BCD 数。 `CLRC ; clear carryDADD R5,R3 ; add LSDsDADD R6,R4 ; add MSDs with carryJC OVERFLOW ; If carry occurs go to error handling routine`

示例	RAM 字节 CNT 中的两位十进制计数器递增 1。
	`CLRC ; clear carryDADD.B #1,CNT`
	或者
	`SETCDADD.B #0,CNT ; equivalent to DADC.B CNT`

3.4.6.17 DEC

***DEC[.W]**	递减目的
***DEC.B**	递减目的
句法	`DEC dst or DEC.W dst DEC.B dst`
运行	dst-1→dst
仿真	`SUB #1,dstSUB.B #1,dst`
说明	目的操作数递减 1。 原先的内容丢失。
状态位	N：如果结果为负则置位，否则复位 Z：如果 dst 包含 1 则置位，否则复位 C：如果 dst 包含 0 则复位，否则置位 V：如果一个算术溢出发生则置位，否则复位 如果目的的初始值为 08000h 则置位，否则复位。 如果目的的初始值为 080h 则置位，否则复位。
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	R10 递减 1。 `DEC R10 ; Decrement R10; Move a block of 255 bytes from memory location starting with EDE to memory location starting with; TONI. Tables should not overlap: start of destination address TONI must not be within the range EDE; to EDE+0FEhMOV #EDE,R6MOV #255,R10L$1 MOV.B @R6+,TONI-EDE-1(R6)DEC R10JNZ L$1` 不要使用上面的具有图 3-13中显示的重叠的例程来传送表格。

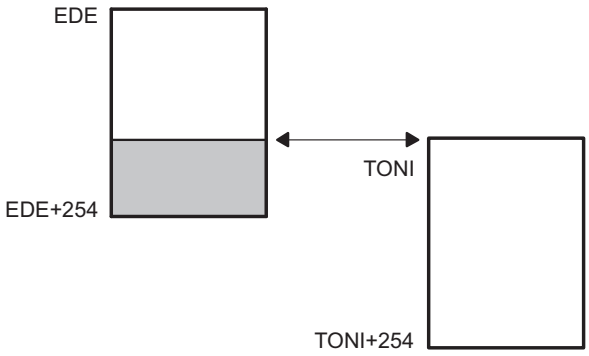


图 **3-13.** 递减重叠

3.4.6.18 DECD

***DECD[.W]**	双递减目的
***DECD.B**	双递减目的
句法	`DECD dst or DECD.W dst DECD.B dst`
运行	dst-2→dst
仿真	`SUB #2,dst`
仿真	`SUB.B #2,dst`
说明	目的操作数递减 2。 原先的内容丢失。
状态位	N：如果结果为负则置位，否则复位 Z：如果 dst 包含 2 则置位，否则复位 C：如果 dst 包含 0 或 1 则复位，否则置位 V：如果一个算术溢出发生则置位，否则复位 如果目的的初始值为 08001 或 08000h 则置位，否则复位。 如果目的的初始值为 081 或 080h 则置位，否则复位。
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	R10 递减 2。 `DECD R10 ; Decrement R10 by two; Move a block of 255 words from memory location starting with EDE to ; memory location starting with TONI; Tables should not overlap: start of destination address TONI must not be ; within the range EDE to EDE+0FEhMOV #EDE,R6MOV #510,R10L$1 MOV @R6+,TONI-EDE-2(R6)DECD R10JNZ L$1`

示例	位置 LEO 上的存储器递减 2。
	`DECD.B LEO ; Decrement MEM(LEO)`
	状态字节 STATUS 减 2。
	`DECD.B STATUS`

3.4.6.19 DINT

***DINT**	禁用（通用）中断
句法	`DINT`
运行	0→GIE 或 (0FFF7h .AND. SR→SR/.NOT.src.AND. dst→dst)
仿真	`BIC #8,SR`
说明	所有中断被禁用。 常量 08h 被反转并且与状态寄存器 (SR) 逻辑与。结果被放置在 SR 中。
状态位	状态位不受影响。
模式位	GIE 被清零。 OSCOFF 和 CPUOFF 不受影响。
示例	状态寄存器中的通用中断使能 (GIE) 位被清零来实现 32 位计数器的不中断移动。 这就确保移动器件，计数器不会被任何中断修改。 `DINT ; All interrupt events using the GIE bit are disabled NOP MOV COUNTHI,R5 ; Copy counter MOV COUNTLO,R6 EINT ; All interrupt events using the GIE bit are enabled`

注: 禁用中断

如果任何代码序列需要被保护不受中断影响，DINT 应该在不可中断序列开始前至少在一个指令上执行，或者在一个 NOP 指令后执行。

3.4.6.20 EINT

***EINT** 启用（通用）中断

句法 `EINT`

运行 1→GIE

或

(0008h.OR. SR→SR/.src.OR. dst→dst)

仿真 `BIS #8,SR`

说明 所有中断被启用。

常量 #08h 与状态寄存器 SR 逻辑与。 结果被放置在 SR 中。

状态位 状态位不受影响。

模式位 GIE 被置位。 OSCOFF 和 CPUOFF 不受影响。

示例 状态寄存器中的通用中断使能 (GIE) 位被置位。

```
; Interrupt routine of ports P1.2 to P1.7; P1IN is the address of the register
where all port bits are read. P1IFG is ; the address of the register where all
interrupt events are latched.PUSH.B &P1INBIC.B @SP,&P1IFG ; Reset only accepted
flagsEINT ; Preset port 1 interrupt flags stored on stack; other interrupts are
allowedBIT #Mask,@SPJEQ MaskOK ; Flags are present identically to mask:
jump......MaskOK BIC #Mask,@SP......INCD SP ; Housekeeping: inverse to PUSH
instruction ; at the start of interrupt subroutine. Corrects ; the stack
pointer.RETI
```

注: 启用中断

中断被启用时，使能中断指令 (EINT) 之后的指令一直被执行，即使一个中断处理例程被挂起时也是如此。

3.4.6.21 INC

***INC[.W]**	递增目的
***INC.B**	递增目的
句法	`INC dst or INC.W dst INC.B dst`
运行	dst+1→dst
仿真	`ADD #1,dst`
说明	目的操作数递增 1。 原先的内容丢失。
状态位	N：如果结果为负则置位，否则复位 Z：如果 dst 包含 0FFFFh 则置位，否则复位 如果 dst 包含 0FFh 则置位，否则复位 C：如果 dst 包含 0FFFFh 则置位，否则复位 如果 dst 包含 0FFh 则置位，否则复位 V：如果 dst 包含 07FFFh 则置位，否则复位 如果 dst 包含 07Fh 则置位，否则复位
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	一个过程的状态字节，STATUS 被递增。 当它等于 11 时，采用一个到 OVFL 的分支指令。 `INC.B STATUSCMP.B #11,STATUSJEQ OVFL`

3.4.6.22 INCD

***INCD[.W]**	双递增目的
***INCD.B**	双递增目的
句法	`INCD dst or INCD.W dst INCD.B dst`
运行	dst+2→dst
仿真	`ADD #2,dstADD.B #2,dst`
示例	目的操作数递增 2。 原先的内容丢失。
状态位	N：如果结果为负则置位，否则复位 Z：如果 dst 包含 0FFFEh 则置位，否则复位 如果 dst 包含 0FEh 则置位，否则复位 C：如果 dst 包含 0FFFEh 或 0FFFFh 则置位，否则复位 如果 dst 包含 0FEh 或 0FFh 则置位，否则复位 V：如果 dst 包含 07FFEh 或 07FFFh 则置位，否则复位 如果 dst 包含 07Eh 或 07Fh 则置位，否则复位
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	在不使用一个寄存器的情况下将堆栈顶部的项目 (TOS) 删除。 `PUSH R5 ; R5 is the result of a calculation, which is stored; in the system stackINCD SP ; Remove TOS by double-increment from stack; Do not use INCD.B, SP is a word-aligned registerRET`
示例	堆栈顶端的字节递增 2。 `INCD.B 0(SP) ; Byte on TOS is increment by two`

3.4.6.23 INV

***INV[.W]**	反转目的
***INV.B**	反转目的
句法	`INV dst INV.B dst`
运行	.NOT.dst→dst
仿真	`XOR #0FFFFh,dstXOR.B #0FFh,dst`
说明	目的操作数被反转。 原先的内容丢失。
状态位	N：如果结果为负则置位，否则复位 Z：如果 dst 包含 0FFFFh 则置位，否则复位 如果 dst 包含 0FFh 则置位，否则复位 C：如果结果非零则置位，否则复位 (=.NOT. 零位) 如果结果非零则置位，否则复位 (=.NOT. 零位) V：如果初始目的操作数为负则置位，否则复位
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	R5 的内容被求反（2 补码） `MOV #00AEh,R5 ; R5 = 000AEhINV R5 ; Invert R5, R5 = 0FF51hINC R5 ; R5 is now negated, R5 = 0FF52h`
示例	存储器字节的内容被求反。 `MOV.B #0AEh,LEO ; MEM(LEO) = 0AEhINV.B LEO ; Invert LEO, MEM(LEO) = 051hINC.B LEO ; MEM(LEO) is negated, MEM(LEO) = 052h`

3.4.6.24 JC，JHS

JC	如果进位被设定则跳转
JHS	如果高于或相等则跳转
句法	`JC label JHS label`
运行	如果 C=1：PC + 2 offset → PC 如果 C=0：执行之后的指令
说明	状态寄存器进位位 (C) 被测试。 如果它被置位，包含在指令 LSB 中的 10位带符号偏移被加至程序计数器。 如果 C 被复位，跳转之后的下一条指令被执行。 JC（如果进位/高于或一样则跳转）被用于不带符号数（0 至 65536）的比较。
状态位	状态位不受影响。

示例	P1IN.1 信号被用于定义或者控制程序流程。
	`BIT.B #02h,&P1IN ; State of signal -`
	`> CarryJC PROGA ; If carry=1 then execute program routine A..... ;`
	`Carry=0, execute program here`

示例	R5 与 15 相比较。 如果其内容高于或一样，分支至标签。 `CMP #15,R5JHS LABEL ; Jump is taken if R5 >= 15..... ; Continue here if R5 < 15`

3.4.6.25 JEQ，JZ

JEQ，JZ	如果相等则跳转，如果为零则跳转
句法	`JEQ label JZ label`
运行	如果 Z=1：PC + 2 offset → PC 如果 Z=0：执行之后的指令
说明	状态寄存器零位 (C) 被测试。 如果它被置位，包含在指令 LSB 中的 10位带符号偏移被加至程序计数器。 如果 Z 未被置位，跳转之后的指令被执行。
状态位	状态位不受影响。
示例	如果 R7 包含零，则跳转至地址 TONI。 `TST R7 ; Test R7JZ TONI ; if zero: JUMP`
示例	如果 R6 等于表内容，则跳转至地址 LEO。 `CMP R6,Table(R5) ; Compare content of R6 with content of ; MEM (table address + content of R5)JEQ LEO ; Jump if both data are equal..... ; No, data are not equal, continue here`
示例	如果 R5 为 0，则分支至标签。 `TST R5JZ LABEL.....`

3.4.6.26 JGE

JGE	如果大于或相等，则跳转。
句法	`JGE label`
运行	如果 (N .XOR. V)=0，则跳转至标签：PC + 2P offset → PC 如果 (N .XOR. V)=1，那么执行后面的指令。
说明	状态寄存器负位 (N) 和溢出位 (V) 被测试。 如果 N和 V 都被置位或复位，包含在指令 LSB 中的 10 位带符号偏移被加至程序计数器。 如果只有一个被置位，跳转之后的指令被执行。 这可实现带符号数的比较。
状态位	状态位不受影响。
示例	当 R6 的内容大于或等于R7 指向的存储器的时候，程序继续在标签 EDE 上执行。 `CMP @R7,R6 ; R6 >= (R7)?, compare on signed numbersJGE EDE ; Yes, R6 >= (R7)..... ; No, proceed.....`

3.4.6.27 JL

JL	如果少于则跳转
句法	`JL label`
运行	如果 (N .XOR. V)= 1，则跳转至标签：PC + 2 offset → PC 如果 (N .XOR. V)=0 那么执行之后的指令。
说明	状态寄存器负位 (N) 和溢出位被测试。 如果只有一个未被置位，包含在指令 LSB 中的 10 位带符号偏移被添加到程序计数器。 如果 N 和 V 都被置位或复位，执行跳转之后的指令。 这样可实现带符号数的比较。
状态位	状态位不受影响。
示例	当 R6 的内容少于 R7 指向的存储器时，程序化继续在标签 EDE 上继续执行。 `CMP @R7,R6 ; R6 < (R7)?, compare on signed numbersJL EDE ; Yes, R6 < (R7)..... ; No, proceed.....`

3.4.6.28 JMP

JMP	无条件跳转
句法	`JMP label`
运行	PC + 2 × offset → PC
说明	包含在指令 LSB 内的 10 位带符号偏移被加至程序计数器。
状态位	状态位不受影响。
提示	这个单字指令取代相对于程序计数器的在 -511至 +512 字范围内的分支 (BRANCH) 指令。

3.4.6.29 JN

JN	如果为负则跳转
句法	`JN label`
运行	如果 N= 1：PC + 2 x offset → PC 如果 N=0：执行之后的指令
说明	状态寄存器的负位 (N) 被测试。 如果它被置位，包含在指令 LSB 中的 10位带符号偏移被加至程序计数器。 如果 N 被复位，跳转之后的下一条指令被执行。
状态位	状态位不受影响。
示例	从计数 (COUNT) 中减去 R5 中的计算结果。 如果结果为负，COUNT 被清零并且程序继续在另外一条路径上执行。

```
SUB R5,COUNT ; COUNT - R5 -
> COUNTJN L$1 ; If negative continue with COUNT=0 at PC=L$1..... ;
Continue with COUNT>=0..................L$1 CLR COUNT.................
```

3.4.6.30 JNC，JLO

JNC	如果进位没被设定则跳转
JLO	如果低于则跳转
句法	`JNC label JLO label`
运行	如果 C=0：PC + 2 offset → PC 如果 C=1：执行之后的指令
说明	状态寄存器进位位 (C) 被测试。 如果它被复位，包含在指令 LSB 中的 10位带符号偏移被加至程序计数器。 如果 C 被置位，跳转之后的下一条指令被执行。 JNC（如果无进位/低于则跳转）用于 比较无符号数（0 至 65536）。
状态位	状态位不受影响。

示例	R6 中的结果被加至缓冲器 (BUFFER)。 如果一个溢出发生，使用位于地址 ERROR上的一个错误错误处理例程。
	`ADD R6,BUFFER ; BUFFER + R6 -`
	`> BUFFERJNC CONT ; No carry, jump to CONTERROR ..... ; Error handler start.................CONT ..... ; Continue with normal program flow...........`

示例	如果 STATUS 包含 1 或 0 则分支至 STL2。 `CMP.B #2,STATUSJLO STL 2 ; STATUS < 2..... ; STATUS >= 2, continue here`

3.4.6.31 JNE，JNZ

JNE	如果不相等则跳转
JNZ	如果不为零则跳转
句法	`JNE label JNZ label`
运行	如果 Z=0：PC + 2 a offset → PC 如果 Z=1：执行之后的指令
说明	状态寄存器零位 (C) 被测试。 如果它被复位，包含在指令 LSB 中的 10位带符号偏移被加至程序计数器。 如果 Z 被置位，跳转之后的指令被执行。
状态位	状态位不受影响。
示例	如果 R7 和 R8 的内容不同，则跳转至地址 TONI。 `CMP R7,R8 ; COMPARE R7 WITH R8JNE TONI ; if different: jump..... ; if equal, continue`

3.4.6.32 MOV

MOV[.W]	将源移动到目的
MOV.B	将源移动到目的
句法	`MOV src,dst or MOV.W src,dst MOV.B src,dst`
运行	src→dst
说明	源操作数被移动至目的操作数。 源操作数不受影响。 目的之前的内容丢失。
状态位	状态位不受影响。
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	表 EDE 的内容（数据字）被复制到表 TOM。 此表的长度必须为 020h 个位置。 `MOV #EDE,R10 ; Prepare pointerMOV #020h,R9 ; Prepare counterLoop MOV @R10+,TOM-EDE-2(R10) ; Use pointer in R10 for both tablesDEC R9 ; Decrement counterJNZ Loop ; Counter not 0, continue copying..... ; Copying completed.....`
示例	表 EDE 的内容（字节数据）被复制到表 TOM。 此表的长度必须为 020h 个位置。 `MOV #EDE,R10 ; Prepare pointerMOV #020h,R9 ; Prepare counterLoop MOV.B @R10+,TOM-EDE-1(R10) ; Use pointer in R10 for ; both tablesDEC R9 ; Decrement counterJNZ Loop ; Counter not 0, continue ; copying..... ; Copying completed.....`

3.4.6.33 NOP

***NOP**	无操作
句法	`NOP`
运行	无
仿真	`MOV #0, R3`
说明	无操作。 在软件检查期间或在定义的等待时间内，此指令可被用于删除指令。
状态位	状态位不受影响。

NOP 指令主要有两个用途：

- 填充一个、两个、或三个存储器字
- 跳转软件时序

注: 仿真误操作指令

其它指令可在提供不同指令周期和代码字数量的同时仿真 NOP 指令。 一些示例为：

```
MOV #0,R3 ; 1 cycle, 1 word 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 cycle, 1 word
```

然而，当使用这些示例应该小心以防止发生意外结果。 例如，如果`MOV 0(R4)`M`0(R4)`被使用并且 R4 中的值为 120h 的话，那么会发生一个到安装装置定时器（地址120h）的安全违反，这是由于安全密钥未被使用。

3.4.6.34 POP

***POP[.W]**	将字从堆栈弹出至目的
***POP.B**	将字节从堆栈弹出至目的
句法	`POP dst POP.B dst`
运行	@SP→temp SP+2→SP temp→dst
仿真	`MOV @SP+,dst or MOV.W @SP+,dstMOV.B @SP+,dst`
说明	堆栈指针 (TOS) 指向的堆栈位置被移动到目的。 之后，堆栈指针递增 2。
状态位	状态位不受影响。
示例	R7 的内容和状态寄存器被从堆栈中恢复。 `POP R7 ; Restore R7POP SR ; Restore status register`
示例	RAM 字节 LEO 的内容被从堆栈中恢复。 `POP.B LEO ; The low byte of the stack is moved to LEO.`
示例	R7 的内容被从堆栈中恢复。 `POP.B R7 ; The low byte of the stack is moved to R7,; the high byte of R7 is 00h`
示例	R7 指向的存储器的内容和状态寄存器被从堆栈中恢复。 `POP.B 0(R7) ; The low byte of the stack is moved to the; the byte which is pointed to by R7; Example: R7 = 203h; Mem(R7) = low byte of system stack; Example: R7 = 20Ah; Mem(R7) = low byte of system stackPOP SR ; Last word on stack moved to the SR`

注: **系统堆栈指针**

系统堆栈指针 (SP) 一直递增 2，与字节后缀无关。

3.4.6.35 PUSH

PUSH[.W]	将字压入堆栈
PUSH.B	将字节压入堆栈
句法	`PUSH src or PUSH.W src PUSH.B src`
运行	SP-2→SP src→@SP
说明	堆栈指针递减 2，随后源操作数被移动到由堆栈指针 (TOS) 寻址的 RAM 字。
状态位	状态位不受影响。
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	状态寄存器和 R8 的内容被保存在堆栈内。 `PUSH SR ; save status registerPUSH R8 ; save R8`

示例	外设 TCDAT 的内容被保存在堆栈上。
	`PUSH.B &TCDAT ; save data from 8-`
	`bit peripheral module,; address TCDAT, onto stack`

注: **系统堆栈指针**

系统堆栈指针 (SP) 一直递减 2，与字节后缀无关。

3.4.6.36 RET

***RET**	从子例程返回
句法	RET
运行	@SP→PC SP+2→SP
仿真	MOV @SP+,PC
说明	被一个调用指令压入堆栈的返回地址被移动到程序计数器。 程序继续在子例程调用之后的代码地址上执行。
状态位	状态位不受影响。

3.4.6.37 RETI

RETI	从中断返回
句法	`RETI`
运行	TOS→SR SP+2→SP TOS→PC SP+2→SP
说明	通过用 TOS 内容替代现有的 SR 内容，状态寄存器被恢复至中断处理例程开始位置的值。 堆栈指针 (SP)被递增 2。 程序计数器被恢复至中断处理开始位置的值。 这是被中断程序流程之后的连续步骤。 通过用 TOS 存储器内容替代现有的 PC 内容，恢复被执行。 堆栈指针 (SP) 被递增。
状态位	N：从系统堆栈中恢复 Z：从系统堆栈中恢复 C：从系统堆栈中恢复 V：从系统堆栈中恢复
模式位	OSCOFF，CPUOFF 和 GIE 从系统堆栈中恢复。
示例	图 3-14解释了主程序中断。

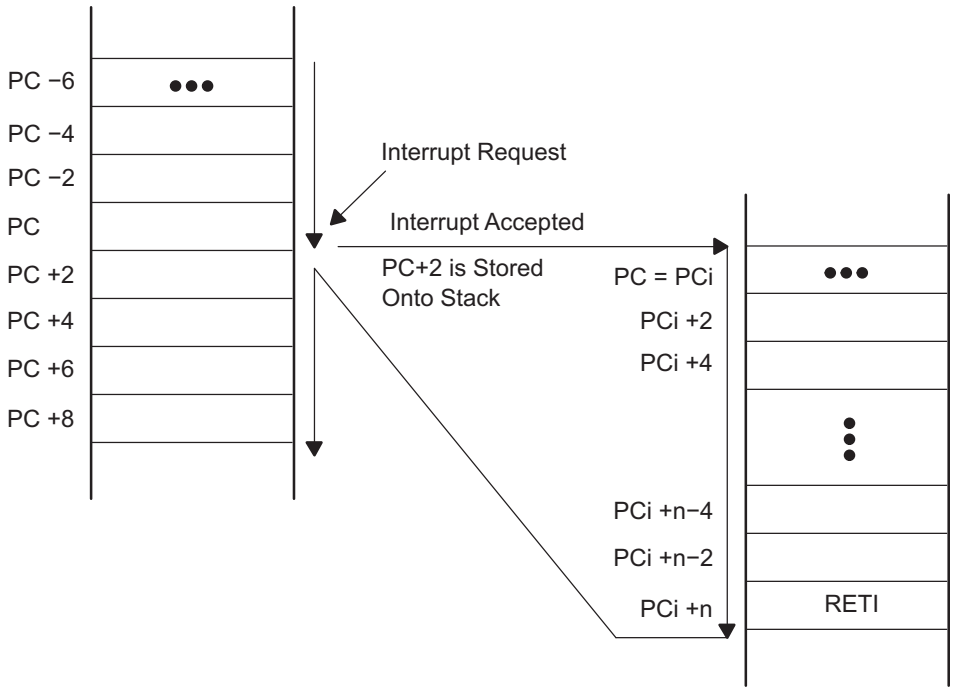


图 **3-14.** 主程序中断

3.4.6.38 RLA

***RLA[.W]** 算术左旋转

***RLA.B** 算术左旋转

句法 `RLA dst or RLA.W dst RLA.B dst`

运行 C<-MSB<-MSB-1 .... LSB+1<-LSB<-0

仿真 `ADD dst,dst ADD.B dst,dst`

说明 目的操作数如图 3-15中显示的那样左移一个位置。 MSB 被移入进位位 (C)，而 LSB 用 0 填充。RLA 指令作为一个带符号的乘以 2 的乘法。

如果在操作被执行前 dst≥04000h 和 dst<0C000h，一个溢出发生：结果的符号改变。

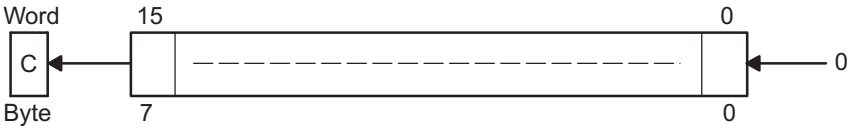


图 **3-15.** 目的操作数-算术左移

如果在操作被执行前 dst≥040h 和 dst<0C0h，一个溢出发生：结果的符号改变。

状态位 N：如果结果为负则置位，否则复位

Z：如果结果为零则置位，否则复位

C：从 MSB 载入

V：如果一个算术溢出发生则置位：

初始值为 04000h≤dst<0C000h；否则复位

如果一个算术溢出发生则置位：

初始值 040h ≤dst<0C0h；否则复位

模式位 OSCOFF，CPUOFF 和 GIE 不受影响。

示例 R7 乘以 2。

```
RLA R7 ; Shift left R7 (x 2)
```

示例 R7 的低字节乘以 4。

```
RLA.B R7 ; Shift left low byte of R7 (x 2) RLA.B R7 ; Shift left low
byte of R7 (x 4)
```

注： **RLA** 替代

汇编程序不识别此指令：

```
RLA @R5+, RLA.B @R5+, or RLA(.B) @R5
```

它可由

```
ADD @R5+,-2(R5), ADD.B @R5+,-1(R5), or ADD(.B) @R5
```

3.4.6.39 RLC 替代

***RLC[.W]** 通过进位左旋转

***RLC.B** 通过进位做选装

句法 `RLC dst or RLC.W dst RLC.B dst`

运行 C<-MSB<-MSB-1 .... LSB+1<-LSB<-C

仿真 `ADDC dst,dst`

说明 目的操作数被如图 3-16中显示的那样左移一个位置。 进位位 (C) 被移入 LSB，而 MSB 被移入进位位 (C)。

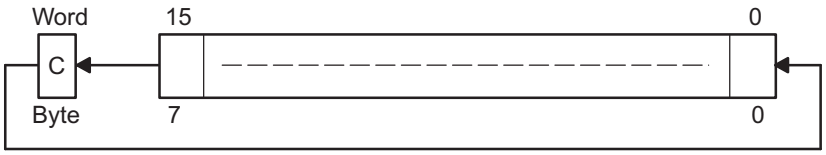


图 **3-16.** 目的操作数-进位左移

状态位 N：如果结果为负则置位，否则复位

Z：如果结果为零则置位，否则复位

C：从 MSB 载入

V：如果一个算术溢出发生则置位

初始值为 04000h≤dst<0C000h；否则复位

如果一个算术溢出发生则置位：

初始值为 040h≤dst<0C0h；否则复位

模式位 OSCOFF，CPUOFF 和 GIE 不受影响。

示例 R5 被左移一个位置。

```
RLC R5 ; (R5 x 2) + C -> R5
```

示例 输入 P1IN.1 信息被移入 R5 的 LSB。

```
BIT.B #2,&P1IN ; Information -> CarryRLC R5 ; Carry=P0in.1 -> LSB of R5
```

示例 MEM(LEO) 内容被左移一个位置。

```
RLC.B LEO ; Mem(LEO) x 2 + C -> Mem(LEO)
```

注: **RLC 和 RLC.B 替代**

汇编程序不识别此指令：

```
RLC @R5+, RLC @R5, or RLC(.B) @R5
```

它必须被

```
ADDC @R5+,-2(R5), ADDC.B @R5+,-1(R5), or ADDC(.B) @R5
```

3.4.6.40 RRA 所替代

RRA[.W] 算术右旋转

RRA.B 算术右旋转

句法 `RRA dst or RRA.W dst RRA.B dst`

运行 MSB→MSB，MSB→MSB-1, ... LSB+1→LSB，LSB→C

说明 目的操作数如图 3-17中显示的那样右移一个位置。 MSB 被移入 MSB，MSB 被移入 MSB-1，而LSB+1 被移入 LSB。

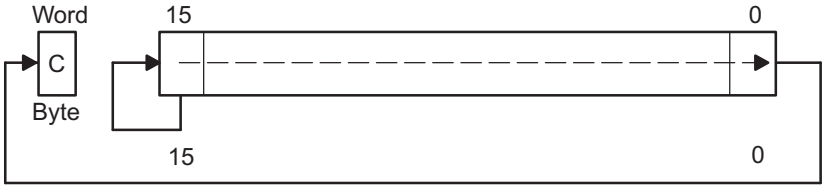


图 **3-17.** 目的操作数**-**算术右移

状态位 N：如果结果为负则置位，否则复位

Z：如果结果为零则置位，否则复位

C：从 MSB 载入

V：复位

模式位 OSCOFF，CPUOFF 和 GIE 不受影响。

示例 R5 被右移一个位置 MSB 保存之前的值。 它的操作与算术除以 2 等效。

```
RRA R5 ; R5/2 -
> R5; The value in R5 is multiplied by 0.75 (0.5 + 0.25).;PUSH R5 ; Hold
R5 temporarily using stackRRA R5 ; R5 x 0.5 -
> R5ADD @SP+,R5 ; R5 x 0.5 + R5 = 1.5 x R5 -
> R5RRA R5 ; (1.5 x R5) x 0.5 = 0.75 x R5 -> R5......
```

示例 R5 的低字节被右移一个位置。 MSB 保存之前的值。 它的操作与算术除以 2 等效。

```
RRA.B R5 ; R5/2 -
> R5: operation is on low byte only; High byte of R5 is resetPUSH.B R5 ;
R5 x 0.5 -> TOSRRA.B @SP ; TOS x 0.5 = 0.5 x R5 x 0.5 = 0.25 x R5 -
> TOSADD.B @SP+,R5 ; R5 x 0.5 + R5 x 0.25 = 0.75 x R5 -> R5......
```

3.4.6.41 RRC

RRC[.W]	通过进位右旋转
RRC.B	通过进位右旋转
句法	`RRC dst or RRC.W dst RRC dst`
运行	C→MSB→MSB-1 .... LSB+1→LSB→C
说明	目的操作数如图 3-18中显示的那样右移一个位置。 进位位 (C) 被移入 MSB，而 LSB 被移入进位位 (C)。

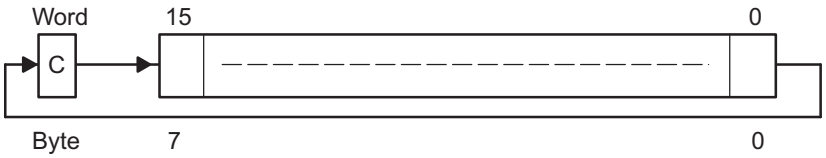


图 **3-18.** 目的操作数-进位右移

状态位	N：如果结果为负则置位，否则复位 Z：如果结果为零则置位，否则复位 C：从 LSB 载入 V：复位
模式位	OSCOFF，CPUOFF 和 GIEare 不受影响。
示例	R5 被右移一个位置。 MSB 被载入 1。 `SETC ; Prepare carry for MSBRRC R5 ; R5/2 + 8000h -> R5`
示例	R5 被右移一个位置。 MSB 被载入 1。 `SETC ; Prepare carry for MSBRRC.B R5 ; R5/2 + 80h -> R5; low byte of R5 is used`

3.4.6.42 SBC

***SBC[.W]**	减去源并且借位/.NOT.来自目的的进位
***SBC.B**	减去源并且借位/.NOT.来自目的的进位
句法	`SBC dst or SBC.W dst SBC.B dst`
运行	dst+0FFFFh+C→dst dst+0FFh+C→dst
仿真	`SUBC #0,dst SUBC.B #0,dst`
说明	进位位 (C) 被加入到目的操作数减一中。 目的之前的内容丢失。
状态位	N：如果结果为负则置位，否则复位 Z：如果结果为零则置位，否则复位 C：如果结果的 MSB 有一个进位则置位，否则复位。 如果无借位则置位为 1，如果有借位则复位。 V：如果一个算术溢出发生则置位，否则复位。
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	从 R12 指向的32 位计数器中减去 R13 指向的 16 位计数器。 `SUB @R13,0(R12) ; Subtract LSDsSBC 2(R12) ; Subtract carry from MSD`
示例	从 R12 指向的16 位计数器中减去 R13 指向的 8 位计数器。 `SUB.B @R13,0(R12) ; Subtract LSDsSBC.B 1(R12) ; Subtract carry from MSD`

注： 借位执行

借位被视为一个.NOT.进位：

借位	进位位
支持	0
否	1

3.4.6.43 SETC

***SETC**	设定进位位
句法	SETC
运行	1→C
仿真	BIS #1,SR
说明	进位位 (C) 被设定。
状态位	N：不受影响 Z：不受影响 C：设定 V：不受影响
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	十进制减法仿真。 用十进制减法从 R6 中减去 R5。 假定 R5=03987h 而 R6=04137h

```
DSUB ADD #06666h,R5 ; Move content R5 from 0-9 to 6-
0Fh; R5 = 03987h + 06666h = 09FEDhINV R5 ; Invert this (result back to 0-
9); R5 = .NOT. R5 = 06012hSETC ; Prepare carry = 1DADD R5,R6 ; Emulate
subtraction by addition of:; (010000h - R5 -
 1); R6 = R6 + R5 + 1; R6 = 0150h
```

3.4.6.44 SETN

***SETN**	设定负位
句法	SETN
运行	1→N
仿真	BIS #4,SR
说明	负位 (N) 被设定。
状态位	N：设定 Z：不受影响 C：不受影响 V：不受影响
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。

3.4.6.45 SETZ

***SETZ**	设定零位。
句法	SETZ
运行	1→Z
仿真	BIS #2,SR
说明	零位 (Z) 被设定。
状态位	N：不受影响 Z：设定 C：不受影响 V：不受影响
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。

3.4.6.46 SUB

SUB[.W]	从目的中减去源
SUB.B	从目的中减去源
句法	`SUB src,dst or SUB.W src,dst SUB.B src,dst`
运行	dst+.NOT.src+1→dst 或 [(dst-src→dst)]
说明	通过将源操作数 1 补码与常量 1 相加，可将源操作数从目的操作数中减去。源操作数不受影响。 目的之前的内容丢失。
状态位	N：如果结果为负则置位，否则复位 Z：如果结果为零则置位，否则复位 C：如果结果的 MSB 有一个进位则置位，否则复位。 如果无借位则置位，如果有借位则复位。 V：如果一个算术溢出发生则置位，否则复位
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	参见 SBC 介绍中的示例。
示例	参见 SBC.B 介绍中的示例。

注: **借位被视为一个 .NOT.**

借位被视为一个.NOT.进位：

借位	进位位
支持	0
否	1

3.4.6.47 SUBC，SBB

SUBC[.W]，SBB[.W]	减去源并借位 / .NOT来自目的的进位
SUBC.B，SBB.B	减去源并借位 / .NOT来自目的的进位
句法	`SUBC src,dst or SUBC.W src,dst or SBB src,dst or SBB.W src,dstSUBC.B src,dst or SBB.B src,dst`
运行	dst+.NOT.src+C→dst 或者 (dst-src-1+C→dst)
说明	通过将源操作数的 1 补码与进位位 (C) 相加，可将源操作数从目的操作数中减去。 源操作数不受影响。 目的之前的内容丢失。
状态位	N：如果结果为负则置位，否则复位。 Z：如果结果为零则置位，否则复位。 C：如果结果的 MSB 有一个进位则置位，否则复位。 如果无借位则置位，如果有借位则复位。 V：如果一个算术溢出发生则置位，否则复位。
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	减去两个浮点尾数（24 位）。 LSB 位于 R13 和 R10 中，MSB 位于 R12 和 R9 中。 `SUB.W R13,R10 ; 16-bit part, LSBsSUBC.B R12,R9 ; 8-bit part, MSBs`
示例	从 R10 指向的一个 16 位计数器和 R11(MSD) 中减去 R13 指向的 16 位计数器。 `SUB.B @R13+,R10 ; Subtract LSDs without carrySUBC.B @R13,R11 ; Subtract MSDs with carry... ; resulting from the LSDs`

注： 借位执行

借位被视为一个.NOT.进位；

借位	进位位
支持	0
否	1

3.4.6.48 SWPB

SWPB 交换字节

句法 `SWPB dst`

运行 位 15 至 8 ↔ 位 7 至 0

说明 如图 3-19所示，目的操作数高字节和低字节互换。

模式位 OSCOFF，CPUOFF 和 GIE 不受影响。

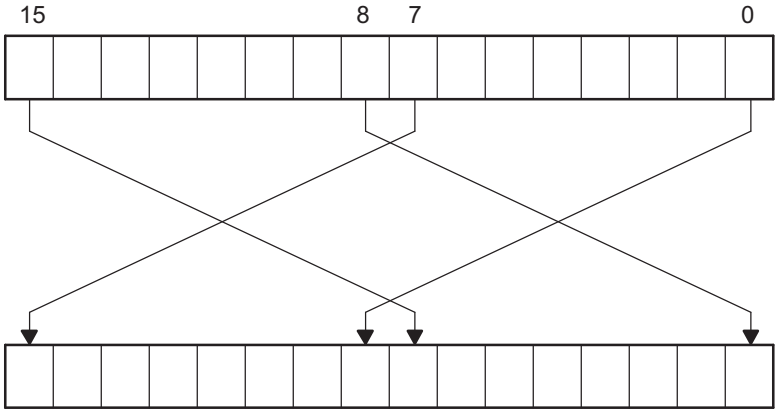


图 **3-19.** 目的操作数-字节交换

示例
```
MOV #040BFh,R7 ; 0100000010111111 -> R7SWPB R7 ; 1011111101000000 in R7
```

示例 R5 中的值乘以 256。 结果被保存在 R5，R4 中。
```
SWPB R5 ;MOV R5,R4 ; Copy the swapped value to R4BIC #0FF00h,R5 ;
Correct the resultBIC #00FFh,R4 ; Correct the result
```

3.4.6.49 SXT

SXT 扩展符

句法 `SXT dst`

运行 位 7 → 位 8..... 位 15

说明 如图 3-20所示，低字节的符号被扩展至高字节。

状态位 N：如果结果为负则置位，否则复位

Z：如果结果为零则置位，否则复位

C：如果结果不为零则复位，否则复位 (.NOT. 零位)

V：复位

模式位 OSCOFF，CPUOFF 和 GIE 不受影响。

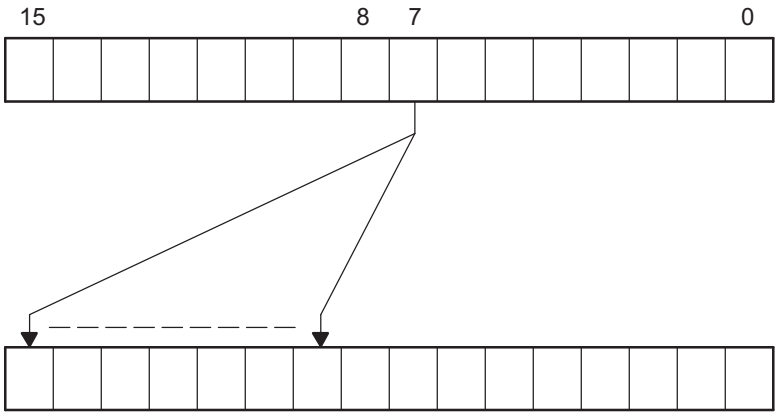


图 3-20. 目的操作数-符号扩展

示例 将 P1IN 值载入到 R7。 扩展符指令的运行用位 7 的值将位 8 扩展至位 15。

然后 R7 被加至 R6。

```
MOV.B &P1IN,R7 ; P1IN = 080h: .... .... 1000 0000SXT R7 ; R7 = 0FF80h:
1111 1111 1000 0000
```

3.4.6.50 TST

***TST[.W]**	测试目的
***TST.B**	测试目的
句法	`TST dst or TST.W dst TST.B dst`
运行	dst+0FFFFh+1 dst+0FFh+1
仿真	`CMP #0,dst CMP.B #0,dst`
说明	目的操作数与零进行比较。 状态位根据结果进行置位。 目的不受影响。
状态位	N：如果目的为负则置位，否则复位 Z：如果目的包含零则置位，否则复位 C：设定 V：复位
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	R7 被测试。 如果它为负，继续在 R7NEG 上执行；如果它为正但不为零，继续在 R7POS 上执行。 `TST R7 ; Test R7JN R7NEG ; R7 is negativeJZ R7ZERO ; R7 is zeroR7POS ..... ; R7 is positive but not zeroR7NEG ..... ; R7 is negativeR7ZERO ..... ; R7 is zero`
示例	R7 的低字节被测试。 如果它为负，继续在 R7NEG 上执行；如果它为正但不为零，继续在 R7POS 上执行。 `TST.B R7 ; Test low byte of R7JN R7NEG ; Low byte of R7 is negativeJZ R7ZERO ; Low byte of R7 is zeroR7POS ..... ; Low byte of R7 is positive but not zeroR7NEG ..... ; Low byte of R7 is negativeR7ZERO ..... ; Low byte of R7 is zero`

3.4.6.51 异或

XOR[.W]	将源与目的进行异或操作
XOR.B	将源与目的进行异或操作
句法	`XOR src,dst or XOR.W src,dst XOR.B src,dst`
运行	src.XOR. dst→dst
说明	源和目的操作数进行异或操作。 结果被放置在目的中。 源操作数不受影响。
状态位	N：如果结果 MSB 被设定则置位，如果未被设定则复位 Z：如果结果为零则置位，否则复位 C：如果结果非零则置位，否则复位 (=.NOT. 零位) V：如果两个操作数都为负则置位
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	R6 中被置位的位转换 RAM 字 TONI 中的位。 `XOR R6,TONI ; Toggle bits of word TONI on the bits set in R6`
示例	R6 中被置位的位转换 RAM 字节 TONI 中的位。 `XOR.B R6,TONI ; Toggle bits of byte TONI on the bits set in; low byte of R6`
示例	将 R7 低字节中与 RAM字节 EDE 中的位不同的那些位复位为 0。 `XOR.B EDE,R7 ; Set different bit to "1s" INV.B R7 ; Invert Lowbyte, Highbyte is 0h`

CPUX

本章介绍了支持 1MB存储器访问的扩展 MSP430X 16 位精简指令集 (RISC) CPU，它的寻址方式，和指令集。 MSP430XCPU 在所有地址空间超过 64KB 的 MSP430 器件上执行。

Topic **Page**

4.1 CPU 介绍

MSP430X CPU 包含有专门针对现代 编程技术，例如计算分支、表处理 和诸如 C 语言的高级语言使用而设计的特性。MSP430X CPU 可寻址1MB 地址范围而无需分页。 此外，在某些情况下，相对于MSP430 CPU，MSP430X CPU 具有较少的中断开销周期和更少的指令周期，而同时又保持了与MSP430 CPU 一样或者更佳的代码密度。 MSP430X CPU 与MSP430 CPU 向后兼容。

MSP430X CPU 的特性包括：

- RISC 架构
- 垂直架构
- 包括程序计数器、状态寄存器和堆栈指针的完全寄存器访问
- 单周期寄存器操作
- 较大的寄存器文件减少了到存储器的取指令操作
- 20 位地址总线可在无需分页的情况下在整个存储器范围内实现直接访问和分支指令
- 16 位数据总线允许字宽自变量的直接操作
- 常数发生器提供最多 6 个经常使用的立即值并减少了代码尺寸
- 无需中间寄存器保持的直接内存到内存传送
- 字节、字、和 20 位地址字寻址

图 4-1显示了 MSP430X CPU 的方框图。

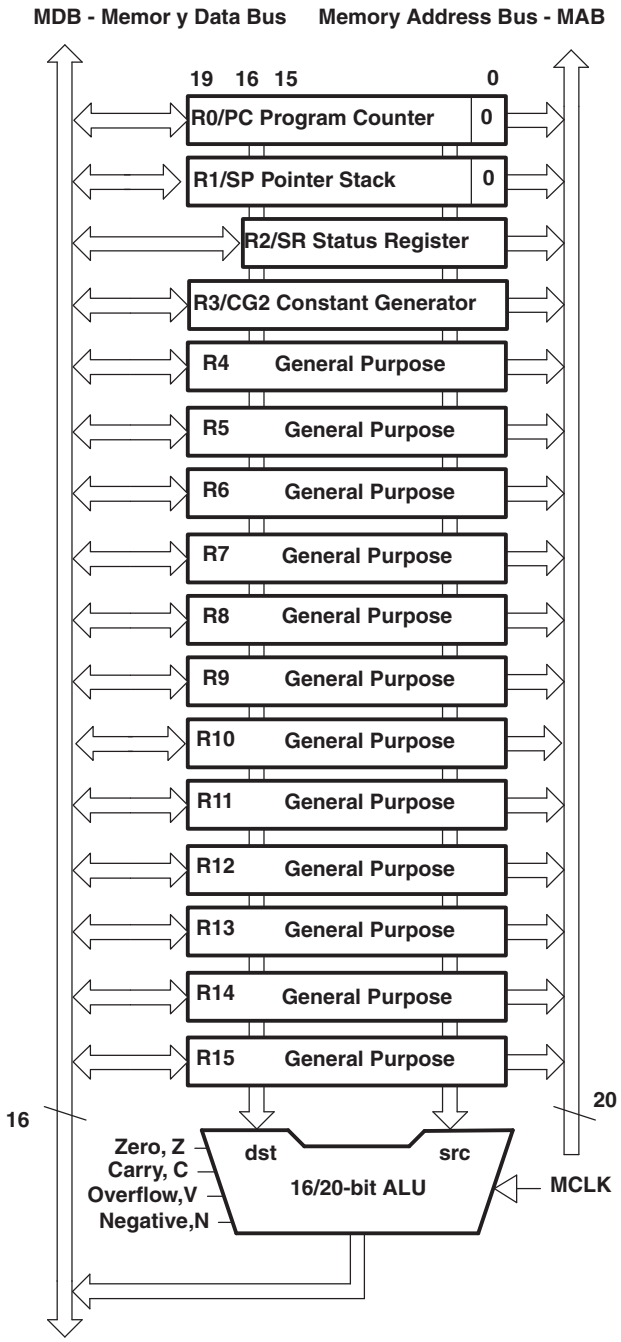


图 **4-1. MSP430X CPU** 方框图

4.2 中断

MSP430X 使用与 MSP430 一样的中断结构：

- 无需轮询的矢量中断
- 中断矢量位于地址 0FFFEh 以下的位置

*第二章 系统复位、中断、和运行模式第二节中断*对 MSP430 和 MSP430X CPU 的中断操作进行了说明。 中断矢量包含指向较低 64KB 存储器的 16 位地址。 这意味着即使在 MSP430X 器件中，所有中断处理器必须在较低 64KB 存储器内开始。

中断期间，如图 4-2所示，程序计数器和状态寄存器被压入堆栈。 MSP430X 架构通过自动将 PC 位 19:16 添加到堆栈上存储的 SR 值上来高效地存储完整的 20 位 PC 值。 当 RETI 指令被执行时，完整的 20 位 PC 被恢复，这样可从中断返回到存储器范围内的任一地址。

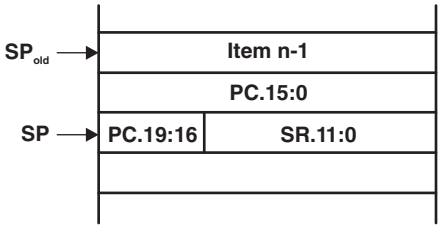


图 **4-2.** 存储在堆栈上用于中断的 **PC**

4.3 CPU 寄存器

CPU 包含有 16 个寄存器（R0 至 R15）。 寄存器 R0，R1，R2，和 R3 有专用功能。 寄存器 R4 至 R15 是用于普通用途的工作寄存器。

4.3.1 程序计数器 (PC)

指向下一条指令的 20 位 PC (PC/R0) 被执行。 每条指令使用一个偶数数量字节（2，4，6，或 8 字节），PC 相应递增。 在字边界上执行指令访问，而 PC 与偶数地址对其。图 4-3显示了 PC。

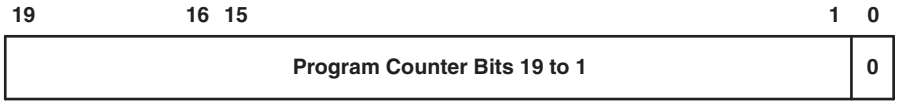


图 **4-3.** 程序计数器

可使用所有指令和寻址模式来对 PC 进行寻址。 几个示例：

```
MOV.W #LABEL,PC ; Branch to address LABEL (lower 64KB)MOVA #LABEL,PC ; Branch to
address LABEL (1MB memory)MOV.W LABEL,PC ; Branch to address in word LABEL ; (lower
64KB)MOV.W @R14,PC ; Branch indirect to address in ; R14 (lower 64KB)ADDA #4,PC ; Skip
two words (1MB memory)
```

BR 和 CALL 函数将 PC 的上四位复位为 0。BR 或者 CALL 指令只能对低 64KB 地址范围内的地址寻址。当分支指令或者调用时，低 64KB 地址范围以外的地址只能使用 BRA 或者 CALLA 指令寻址。 此外，根据所使用的寻址模式，任一直接修改 PC的指令进行类似操作。 例如，`MOV.W #valueM PC`清空 PC 的上四位，这是因为它是一个 .W 指令。

在一个中断处理例程期间，使用 CALL（或 CALLA）指令可将 PC 自动存储在堆栈内。图 4-4显示了一个 CALLA 指令后，具有返回地址的 PC 的存储。 一个 CALL 指令只存储 PC 的位 15:0。

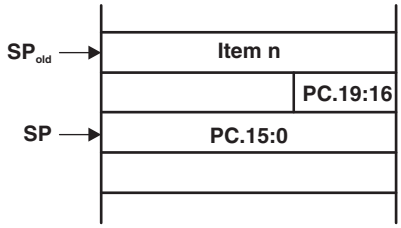


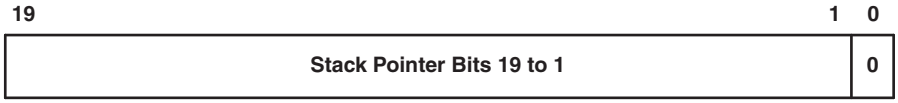
图 **4-4.** 针对 **CALLA** 的 **PC** 堆栈存储

RETA 指令恢复 PC 位 19:0 并且将堆栈指针 (SP) 加 4。 RET 指令恢复 PC 位 15:0 并且将 SP 加 2。

4.3.2 堆栈指针 (SP)

CPU 用 20 位 SP (SP/R1) 来存储子例程调用和中断的返回地址。 它使用一个先递减、后递增的机制。 此外，SP 可被具有所有指令和寻址模式的软件所使用。图 4-5显示了 SP。 SP 由用户初始化到 RAM 中，并且一直与偶数地址对齐。

图 4-6显示了堆栈用法。图 4-7显示了当 20 位地址字被压入时的堆栈用法。




```
MOV.W   2(SP),R6      ; Copy Item I2 to R6
MOV.W   R7,0(SP)      ; Overwrite TOS with R7
PUSH    #0123h        ; Put 0123h on stack
POP     R8            ; R8 = 0123h
```

图 **4-5.** 堆栈指针

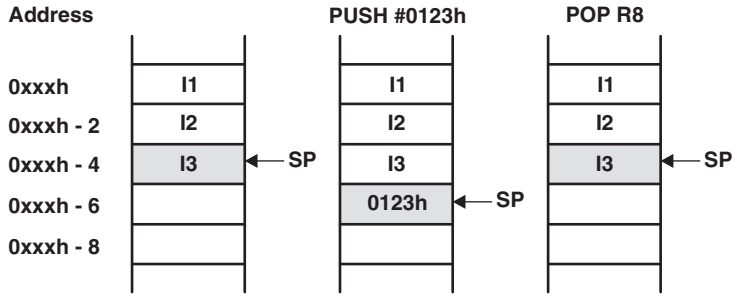


图 **4-6.** 堆栈用法

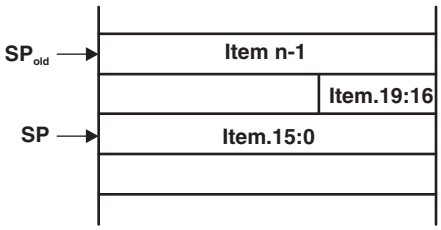


图 **4-7.** 堆栈上的 **PUSH.A** 格式

图 4-8中描述了将 SP 用作一个到 PUSH 和 POP 指令的自变量的特殊情况。

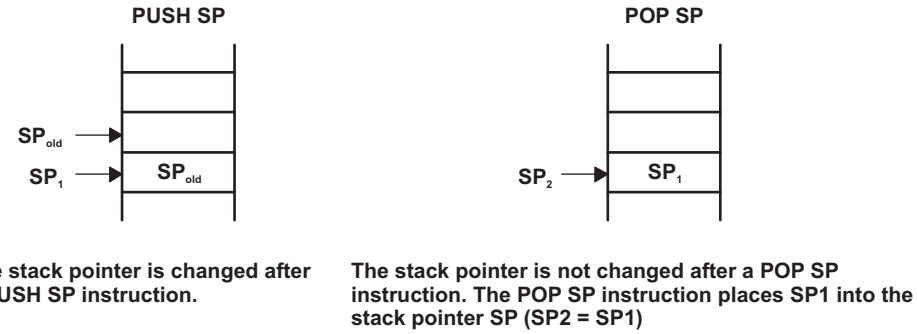


图 **4-8. PUSH SP，POP SP** 序列

4.3.3 状态寄存器 (SR)

用作一个源或者目标寄存器的 16 位 SR (SR/R2) 之可被用在由字指令寻址的寄存器模式中。 寻址描述的剩余组合被用来支持常数发生器。图 4-9显示了 SR 位。 不要将 20 位值写入 SR。 可导致不可预知的运行。

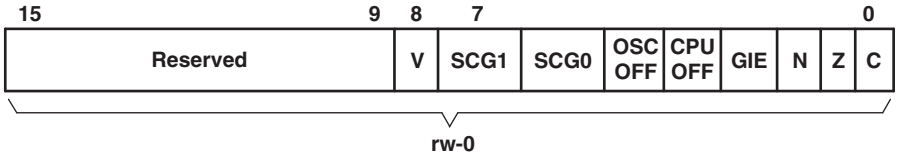


图 **4-9. SR** 位

表 4-1描述了 SR 位。

表 **4-1. SR** 位说明

位	说明	
被保留	被保留	
V	溢出。 当一个算术运算从信号变量范围内溢出时，这个位被置位。	
	ADD(.B), ADDX(.B,.A), ADDC(.B), ADDCX(.B.A), ADDA	当: 正 + 负 = 负 负 + 负 = 正 时置位，否则复位
	SUB(.B), SUBX(.B,.A), SUBC(.B), SUBCX(.B,.A), SUBA, CMP(.B), CMPX(.B,.A), CMPA	当: 正 - 负 = 负 负 - 正 = 负 时置位，否则复位
SCG1	系统时钟生成器 1。根据器件系列的不同，这个位可被用于启用/禁用时钟系统内的功能；例如，数控振荡器 (DCO) 偏置启用/禁用。	
SCG0	系统时钟生成器 0。根据器件系列的不同，这个位可被用于启用/禁用时钟系统内的功能；例如，锁相环 (FLL) 启用/禁用。	
OSCOFF	振荡器关闭。 当 LFXT1CLK 不被用于 MCLK 或者 SMCLK 时，这个位用来关闭 LFXT1 晶体振荡器。	
CPUOFF	CPU 关闭。 当被置位时，这个位关闭 CPU。	
GIE	通用中断启用。 当被置位时，这个位启用可屏蔽中断。 复位时，所有可屏蔽中断被禁用。	
N	负。 当运算结果为负时，这个位被置位，而当结果为正时，这个位被清零。	
Z	零位。 当运算结果为 0 时，这个位被置位，而当结果非 0 时， 这个位被清零。	
C	进位。 当运算结果产生一个进位时，这个位被置位，而当没有出现进位时，这个位被清零。	

4.3.4 常数发生器寄存器（CG1 和 CG2）

在无需额外的 16 位字程序代码的情况下，常数发生器寄存器 R2 (CG1) 和 R3 (CG2) 生成六个常用常数。这些常数用表 4-2中描述的源寄存器寻址模式 (As) 进行选择。

表 4-2. 常数发生器 CG1，CG2 的值

寄存器	作为	常量	备注
R2	00	-	寄存器模式
R2	01	(0)	绝对地址模式
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	FFh，FFFFh，FFFFFh	-1，字处理

常数发生器的优势在于：

- 无需特殊指令
- 无需用于六个常数的额外代码字
- 检索常数无需代码存储器访问

如果六个常数中的一个被用作一个中间源操作数，汇编程序自动使用常数发生器。 用在常数模式的寄存器 R2 和 R3，不能被明确寻址；它们只作为源寄存器运行。

4.3.4.1 常数发生器-扩展指令集

MSP430 的 RISC 指令集只有 27 条指令。 然而，常数发生器使得 MSP430 汇编程序支持 24 条附加仿真指令。 例如，单操作数指令：

```
CLR dst
```

由双操作数指令使用同样长度的：

```
MOV R3,dst进行仿真
```

其中的 #0 被汇编程序所取代，并且在 As=00 时使用 R3。

```
INC dst
```

被：

```
ADD 0(R3),dst所替代
```

4.3.5 通用寄存器（R4 至 R15）

12 个 CPU 寄存器（R4 至 R15）包含 8 位，16 位，或 20 位值。 任何到 CPU 寄存器的字节写入操作将清零位 19:8。任何到寄存器的字写入操作将清零位 19:16。 唯一的例外是 SXT 指令。 SXT 指令通过完整的 20 位寄存器来扩展此符号。

下面的图表显示了字节、字、和地址字数据的处理。 请注意，如果寄存器是一个字节或者字指令的目的地址，前缘最高有效位 (MSB) 被复位。

图 4-10显示了字节处理（8 位数据，.B 后缀）。 显示的是针对一个源寄存器和一个目的存储器字节，一个源存储器字节和一个目的寄存器的处理。

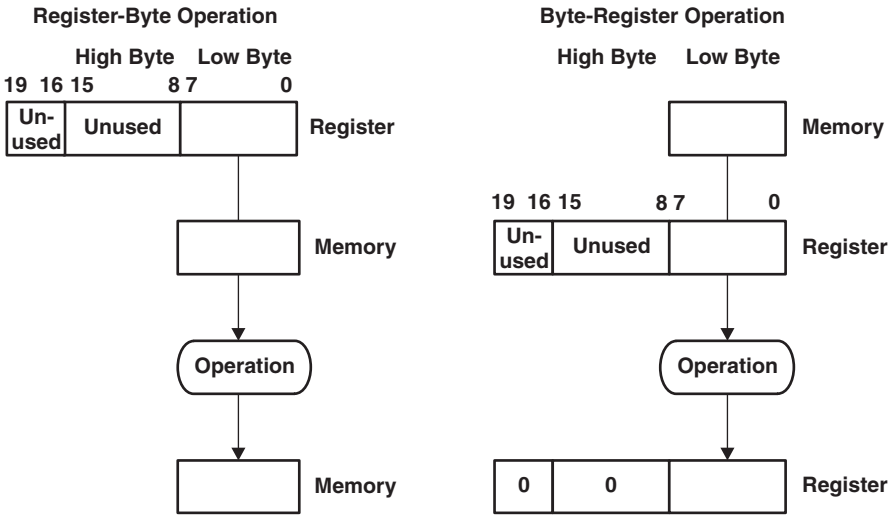


图 **4-10.** 寄存器-字节/字节-寄存器操作

图 4-11和图 4-12显示了 16 位字处理（.W 后缀）。 显示的是针对一个源寄存器和一个目的存储器字，一个源存储器字和个目的寄存器的处理。

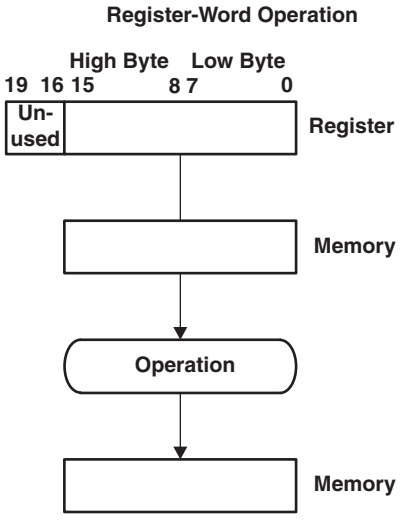


图 **4-11.** 寄存器-字操作

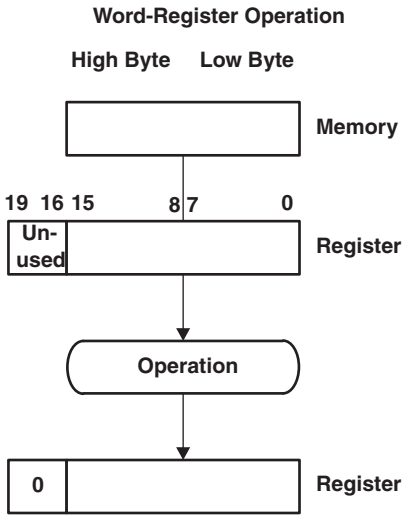


图 **4-12.** 字-寄存器操作

图 4-13和图 4-14显示了 20 位地址字处理（.A 后缀）。 显示的是针对一个源寄存器和一个目的存储器地址字，一个源存储器地址字和一个目的寄存器的处理。

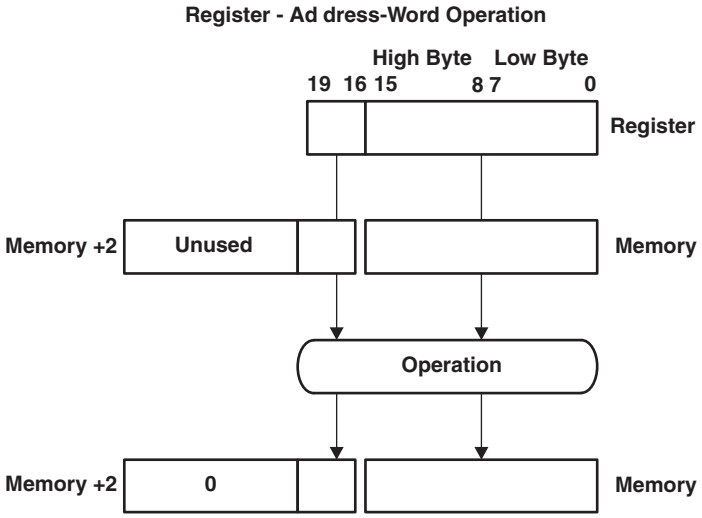


图 **4-13.** 寄存器-地址字操作

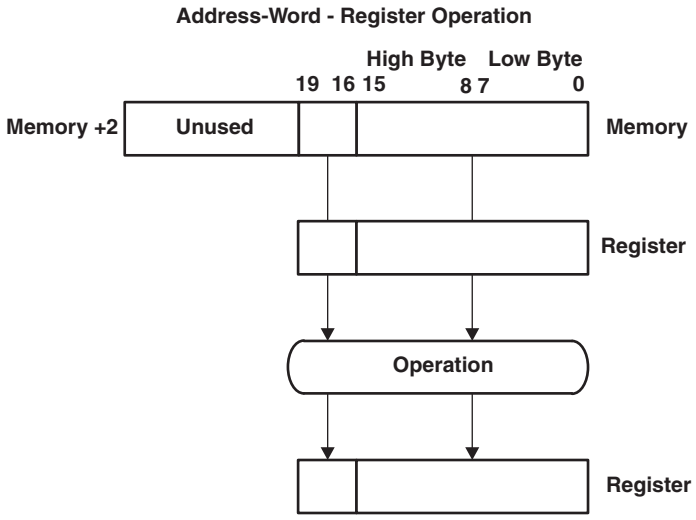


图 **4-14.** 地址-字-寄存器操作

4.4 寻址模式

针对源操作数七个寻址模式和针对目的操作数的四个寻址模式使用 16 位 或者 20 位地址（请见表 4-3）。MSP430 和 MSP430X 指令在整个 IMB 存储器范围内可用。

表 **4-3.** 源/目的寻址

As/Ad	寻址模式	句法	说明
00/0	寄存器	Rn	寄存器的内容为操作数。
01/1	加索引的	X(Rn)	(Rn + X) 指向操作数。 X 被存储在下一个字中，或者存储在之前扩展字和下一个字的组合中。
01/1	符号	ADDR	(PC + X) 指向操作数。 X 被存储在下一个字中，或者存储在之前扩展字和下一个字的组合中。 加索引的模式 X (PC) 被使用。
01/1	绝对	&ADDR	指令之后的字包含绝对地址。 X 被存储在下一个字中，或者存储在之前扩展字和下一个字的组合中。 加索引的 X(SR) 被使用。
10/-	间接寄存器	@Rn	Rn 被用作一个指向操作数的指针。
11/-	间接自动递增	@Rn+	Rn 被用作一个指向操作数的指针。 针对 .B 指令，Rn 随后加 1。 针对 .W 指令，加 2，针对 .A 指令加 4。
11/-	立即	#N	N 被存储在下一个字中，或者存储在之前扩展字和下一个字的组合中。 间接自动递增模式 @PC+ 被使用。

在下面的小节中详细解释了七个寻址模式。 大多数的示例显示了针对源和地址的同样的寻址模式，但是在一个指令中源和目的寻址模式的任何有效组合都是可能的。

注: **标签 EDE，TONI，TOM，和 LEO 标签的使用**

在整个 MSP430 文档中，EDE，TONI，TOM，和 LEO 被用作普通标签。 它们只是标签而并未特殊含义。

4.4.1 寄存器模式

运行： 操作数为所使用的 CPU 寄存器的 8，16，或者 20 位内容。

长度： 一个、两个、或者三个字

注释： 对于源和地址有效

字节操作： 字节操作只读取源寄存器 Rsrc 的八个最低有效位 (LSB)，而将结果写入目的寄存器 Rdst 的八个 LSB 中。 位 Rdst.19:8 被清零。 寄存器 Rsrc 未修改。

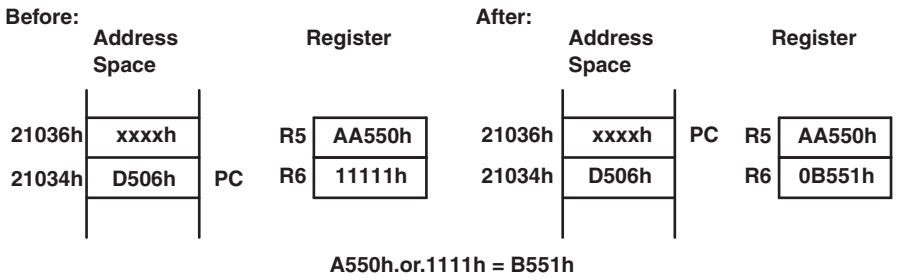
字操作： 字操作读取源寄存器 Rsrc 的 16 个 LSB，而将结果写入目的寄存器 Rdst 的 16 个 LSB 中。 位 Rdst.19:16 被清零。 寄存器 Rsrc 未被修改。

地址字操作： 地址字操作读取源寄存器 Rsrc 的 20 个位，而将结果写入目的寄存器 Rdst 的 20 个位中。 寄存器 Rsrc 未被修改。

SXT 例外： SXT 指令是寄存器操作的唯一一个例外情况。 位 7 中低字节的符号被扩展至位 Rdst.19:8。

示例： `BIS.W R5M R6M`

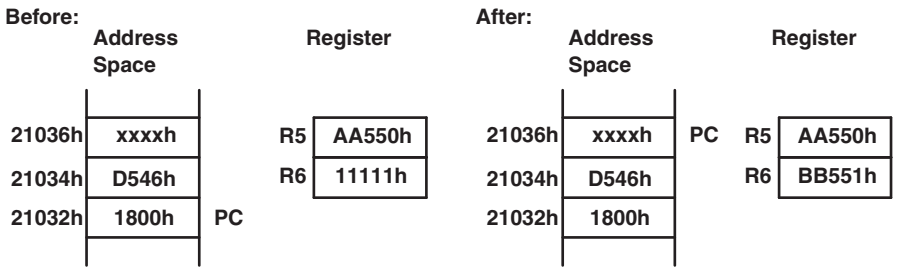
这条指令用 R6 的 16 位内容与 R5 包含的 16 位数据进行逻辑或 (OR) 操作。 R6.19:16 被清零。



A550h.or.1111h = B551h

示例： `BISX.A R5M R6M`

这条指令用 R6 的 20 位内容与 R5 包含的 20 位数据进行逻辑或 (OR) 操作。 扩展字包含针对 20 位数据的 A/L 位。 这条指令使用位 A/L:B/W=01 时的字节模式。 这条指令的结果为：



AA550h.or.11111h = BB551h

4.4.2 已索引的模式：

通过将符号化的索引添加到一个 CPU 寄存器中，已被索引的模式计算操作数的地址。 已索引模式具有三个寻址可能：

- 低 64KB 存储器中的已索引模式
- 具有已索引模式的 MSP430 指令对低 64KB 存储器以上的地址进行寻址。
- 具有已索引模式的 MSP430X 指令

4.4.2.1 低 64KB 存储器中的已索引模式

如果 CPU 寄存器 Rn 指向低 64KB 存储器范围中的一个地址，在添加了 CPU 寄存器 Rn 和符号化的 16 位索引后，计算得出的存储器地址位 19:16 被清零。 这意味着计算得出的地址一直位于低 64KB 存储器中，并且不会从低 64KB 存储器空间中上溢或者下溢。 RAM 和外设寄存器可用这个方法进行访问并且如图 4-15 所示，在无需修改的情况下现有的 MSP430 软件可用。

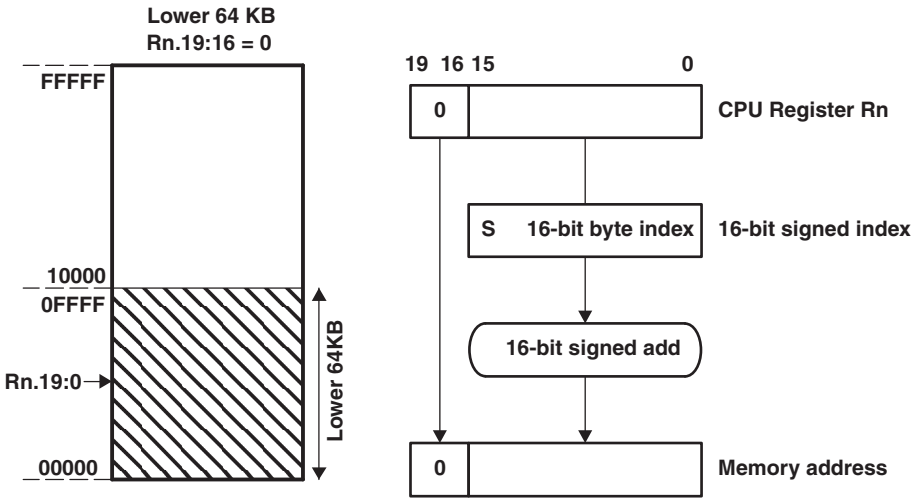
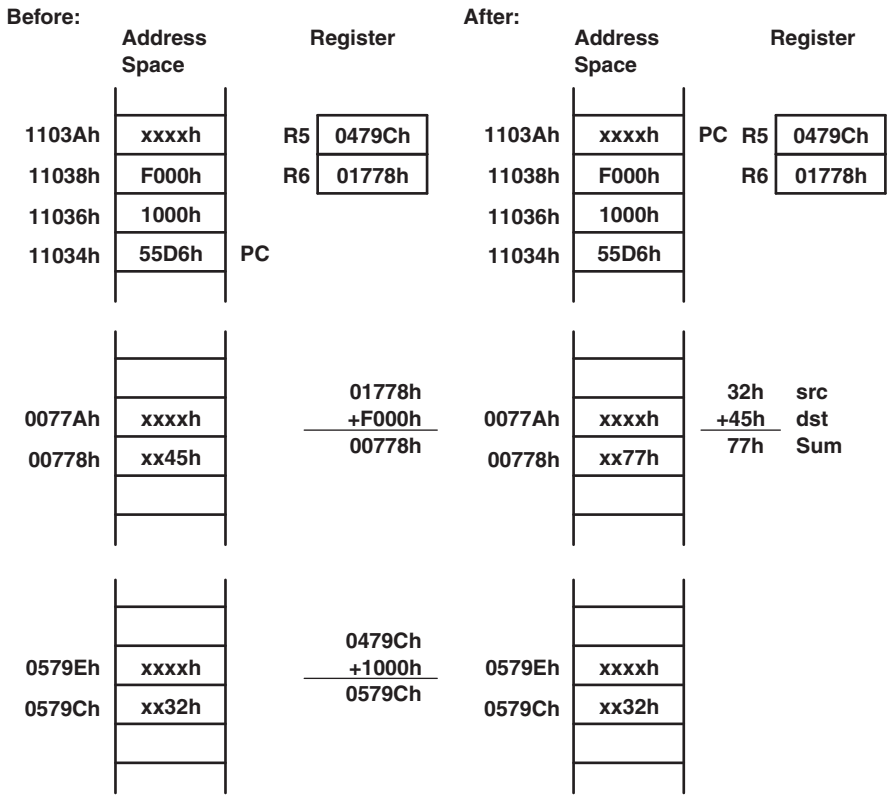


图 **4-15.** 低 **64KB** 中的已索引模式

长度：	两个或者三个字
运行：	符号化的 16 位索引位于此指令后的下一个字中并且被添加到 CPU 寄存器 Rn 中。 得出的位 19:16 被清零，从而给出了一个缩短了的 16 位存储器地址，此地址指向范围 00000h 至 0FFFFh 内的一个操作数地址。 此操作数是已寻址存储器位置的内容。
注释：	对于源和目的有效。 汇编程序计算寄存器索引并将其插入。
示例：	`ADD.B 1000h(R5)M0F000h(R6)M` 这条指令加上 包含在源字节 1000h(R5) 和目的字节 0F000h(R6) 中的 8 位数据并且将结果防止在目的字节中。 由于寄存器 R5 和 R6 中被清零的位 19:16，源和目的字节都位于低 64KB 位置中。
源：	在截断为一个 16 位地址后，R5 指向的字节 + 1000h 得到地址 0479Ch+1000h=0579Ch。
目标：	在截断为一个 16 位地址后，R6 指向的字节 + F000h 得到地址 01778h+F000h= 00778h。



4.4.2.2 上部存储器中的具有已索引模式的 MSP430 指令

如果 CPU 寄存器 Rn 指向一个低 64KB 存储器以上的地址，Rn 位 19:16 被用于操作数的地址计算。 操作数可位于范围 Rn±32KB 内的存储器内，这是因为索引，X，是一个符号化 16 位的值。 在这个情况下，操作数的地址会在低 64KB 存储器中上溢或者下溢（请见图 4-16和图 4-17）。

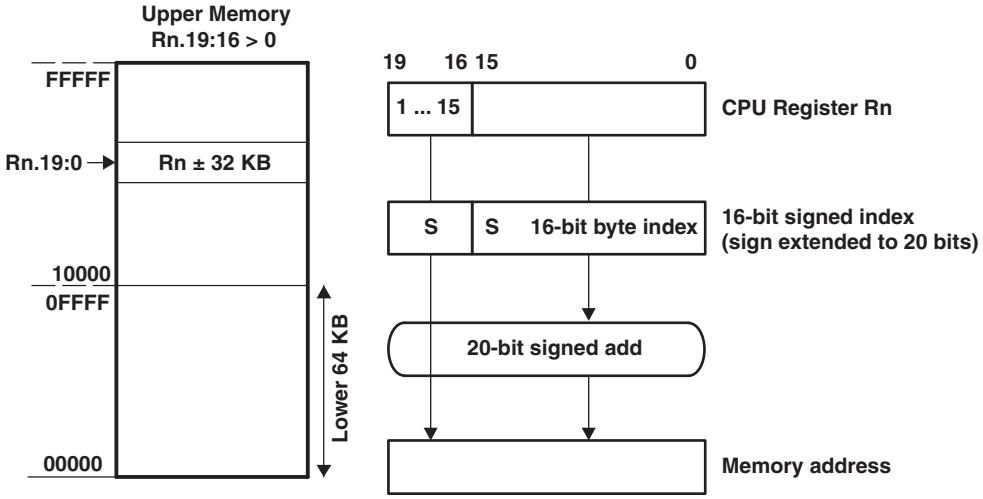


图 **4-16.** 上部存储器中的已索引模式

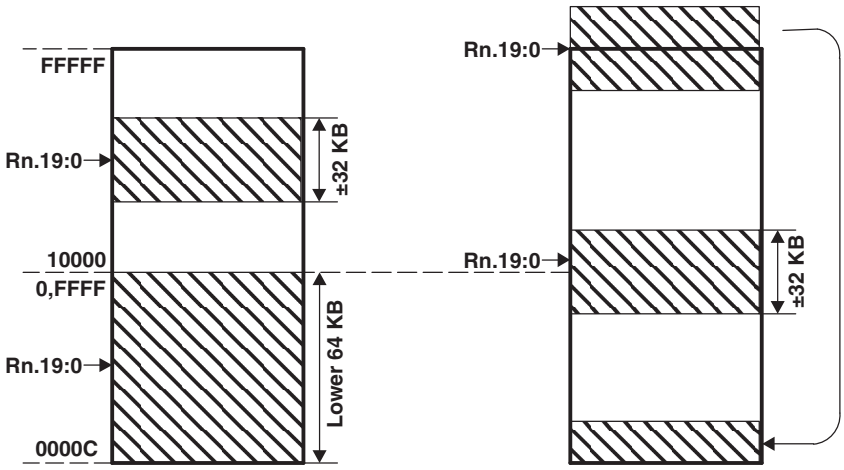


图 **4-17.** 针对已索引模式的上溢和下溢

长度： 两个或者三个字

运行： 指令之后，位于下一个字内的符号扩展 16 位索引被添加到 CPU 寄存器 Rn 的 20 个位内。 这传送了一个 20 位地址，此地址指向范围 0 至 FFFFFh 内的一个地址。 操作数是已寻址存储器位置的内容。

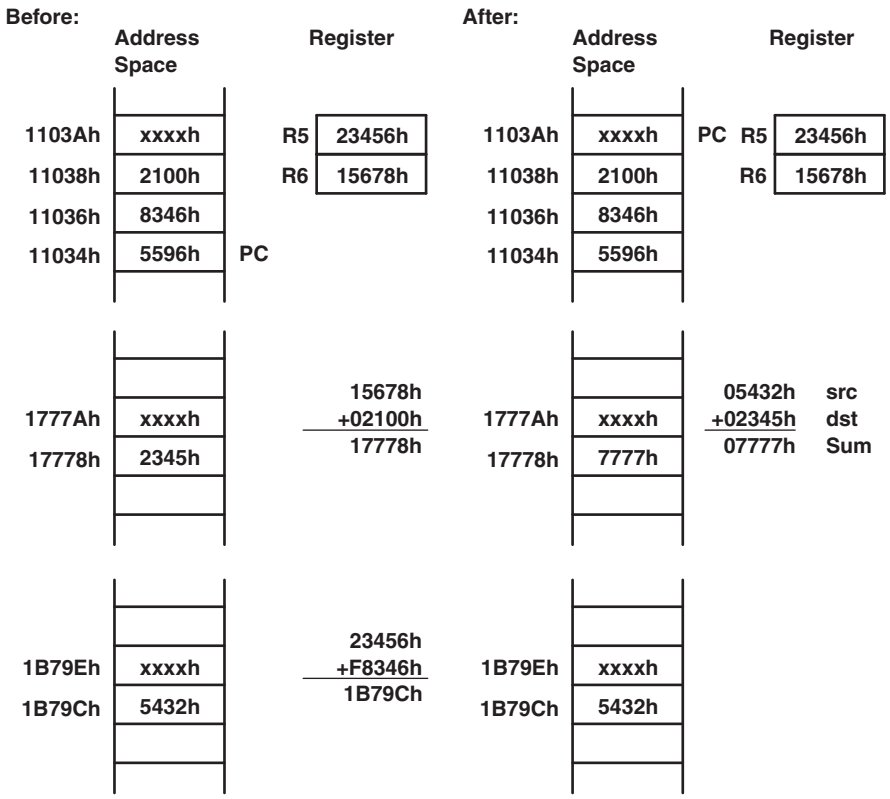
注释： 对于源和目的有效 汇编程序计算寄存器索引并将其插入。

示例： `ADD.W 8346h(R5)M2100h(R6)M`

这条指令加上包含在源和目的地址中的 16 位数据并且将 16 位结果放置在目的地址内。 源和目的操作数可在整个地址范围内被锁定。

源： R5 指向的字 + 8346h。 负索引 8346h 为被扩展的符号，得到地址 23456h+F8346h=1B79Ch。

目标： R6 指向的字 + 2100h 得到地址 15678h+2100h=17778h。

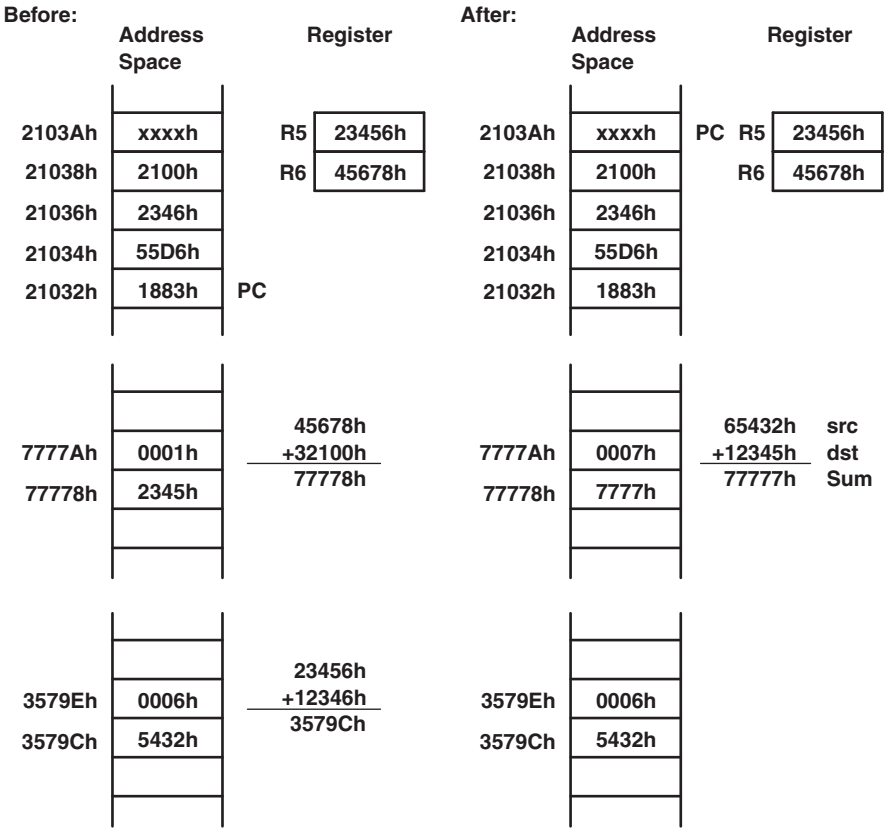


4.4.2.3 具有已索引模式的 MSP430 指令

当使用具有已索引模式的 MSP430X 指令时，操作数可位于 Rn 的范围 + 19 位的任一位置。

长度：	三个或者四个字
运行：	操作数地址是 20 位 CPU 寄存器内容和 20 位索引的和。 索引的 4 个 MSB 包含在扩展字中；16 个 LSB 被包含在随后指令的字中。 CPU 寄存器未修改
注释：	对于源和目的有效。 汇编程序计算寄存器索引并将其插入。
示例：	`ADDX.A 12346h(R5)M32100h(R6)M`
	这条指令加上包含在源和目的地址中的 20 位数据并且将结果放置在目的地址内。
源：	R5 指向的两个字 + 12346h 得到地址 23456h+12346h=3579Ch。
目标：	R6 指向的两个字 + 32100h 得到地址 45678h+32100h=77778h。

扩展字包含源索引和目的索引的 MSB以及针对 20 位数据的 A/L 位。 由于位 A/L:B/W=01 时的 20 位数据长度，这个指令字使用字节模式。



4.4.3 符号模式

通过将符号化索引添加到 PC 中，符号模式计算操作数的地址。 符号模式有三个寻址可能：

- 低 64KB 存储器中的符号模式
- 具有符号模式的 MSP430 指令在低 64KB 存储器之上的存储器内寻址
- 具有符号模式的 MSP430X 指令

4.4.3.1 低 64KB 中的符号模式

如果 PC 指向低 64KB 存储器范围中的一个地址，在增加了 PC 和符号化 16 位索引后，计算得出的存储器地址位 19:16 被清零。 这意味着计算得出的地址一直位于低 64KB 存储器中，并且不会从低 64KB 存储器空间中上溢或者下溢。 RAM 和外设寄存器可用这个方法进行访问并且如图 4-18所示，在无需修改的情况下现有的 MSP430 软件可用。

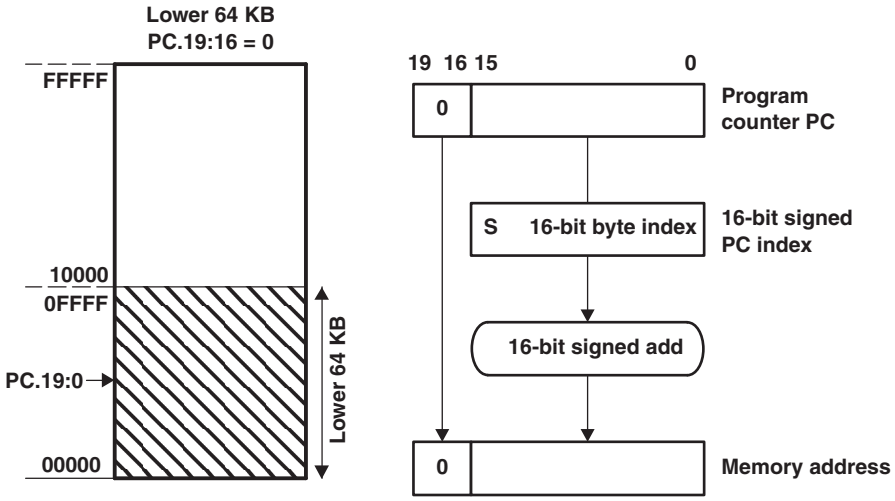


图 **4-18.** 低 **64KB** 中的符号模式运行那个

运行： 指令之后下一个字中的符号化 16 位索引被暂时增加到 PC 中。 得出的位 19:16 被清零，从而给出了一个缩短了的 16 位存储器地址，此地址指向范围 00000h 至 0FFFFh 内的一个操作数地址。 操作数是已寻址存储器位置的内容。

长度： 两个或者三个字

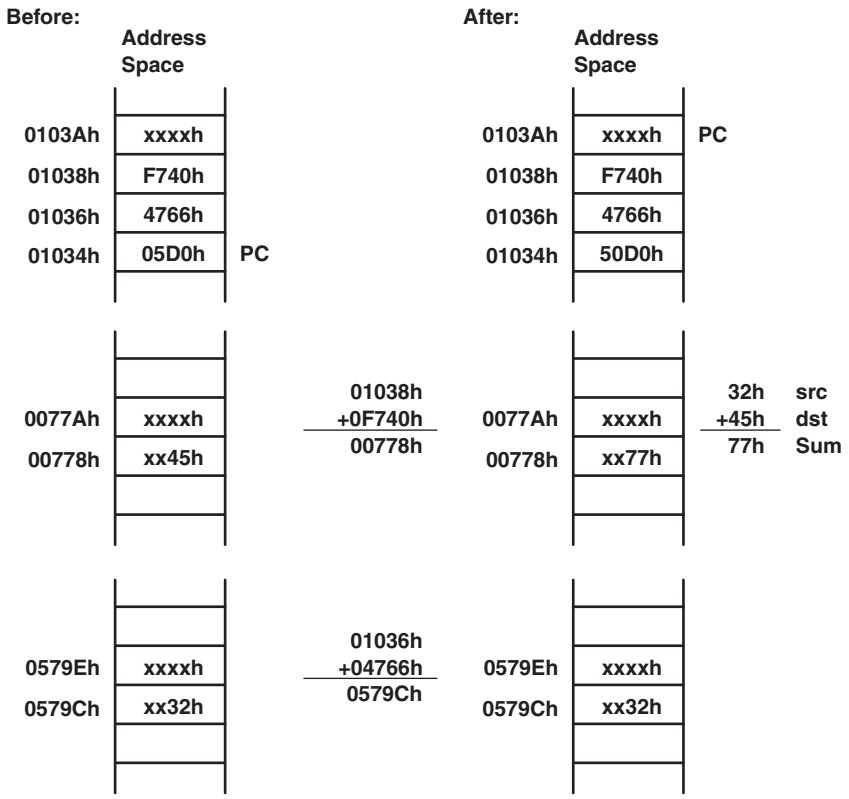
注释： 对于源和目的有效。 汇编程序计算寄 PC 索引并将其插入。

示例： `ADD.B EDEM TONIM`

这个指令加上包含在源字节 EDE 和目的字节 TONI 中的 8 位数据并且将结果放置在目的字节 TONI 中。 字节 EDE 和 TONI 以及程序位于低 64KB 内。

源： 由 PC 指向的位于地址 0579Ch 的字节 EDE+4766h，其中 PC 索引 4766h 是 0579Ch-01036h=04766h 的结果。 地址 01036h 是针对这个示例的索引的位置。

目标： 由 PC 指向的位于地址 00778h 的字节 TONI+F470h，是 00778h-1038h=FF740h 的截短的 16 位结果。 地址 01038h 是针对这个示例的索引的位置。



4.4.3.2 上部存储器中的具有符号模式的 MSP430 指令

如果 PC 指向一个低 64KB 存储器以上的地址，PC 位 19:16 被用于操作数的地址计算。 操作数可位于范围 PC±32KB 内的存储器内，这是因为索引，X，是一个符号化 16 位的值。 在这个情况下，如图 4-19和图 4-20所示，操作数的地址会在低 64KB 存储器中上溢或者下溢。

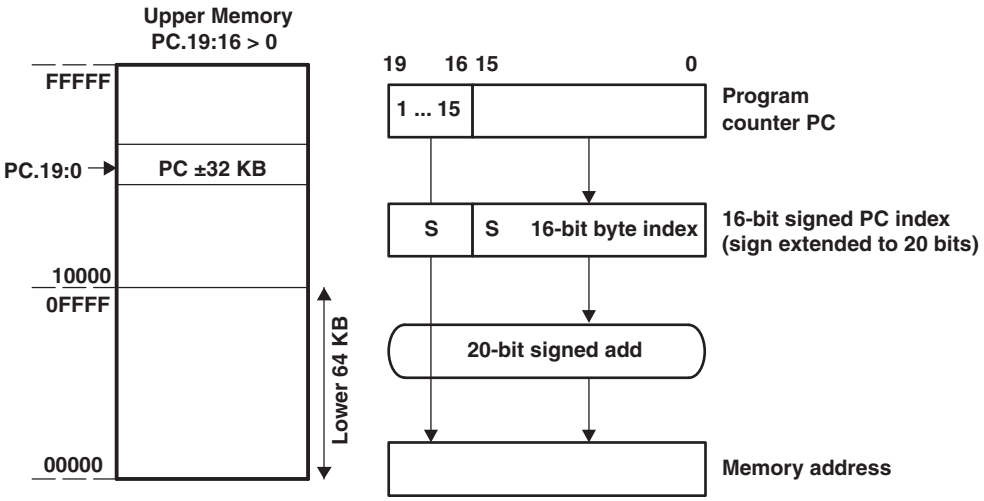


图 **4-19.** 上部存储器内的符号模式运行

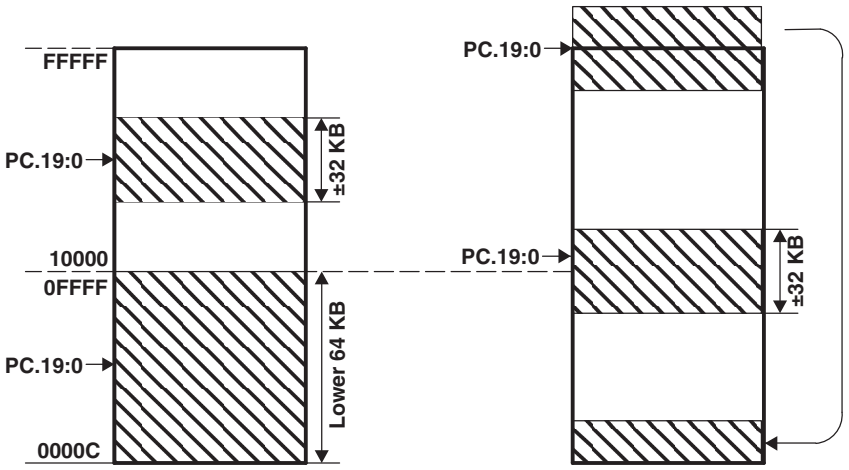


图 **4-20.** 针对符号模式的上溢和下溢

长度： 两个或者三个字

运行： 指令之后，位于下一个字内的符号扩展 16 位索引被添加到 PC 的 20 个位内。 这传送了一个 20 位地址，此地址指向范围 0 至 FFFFFh 内的一个地址。 操作数是已寻址存储器位置的内容。

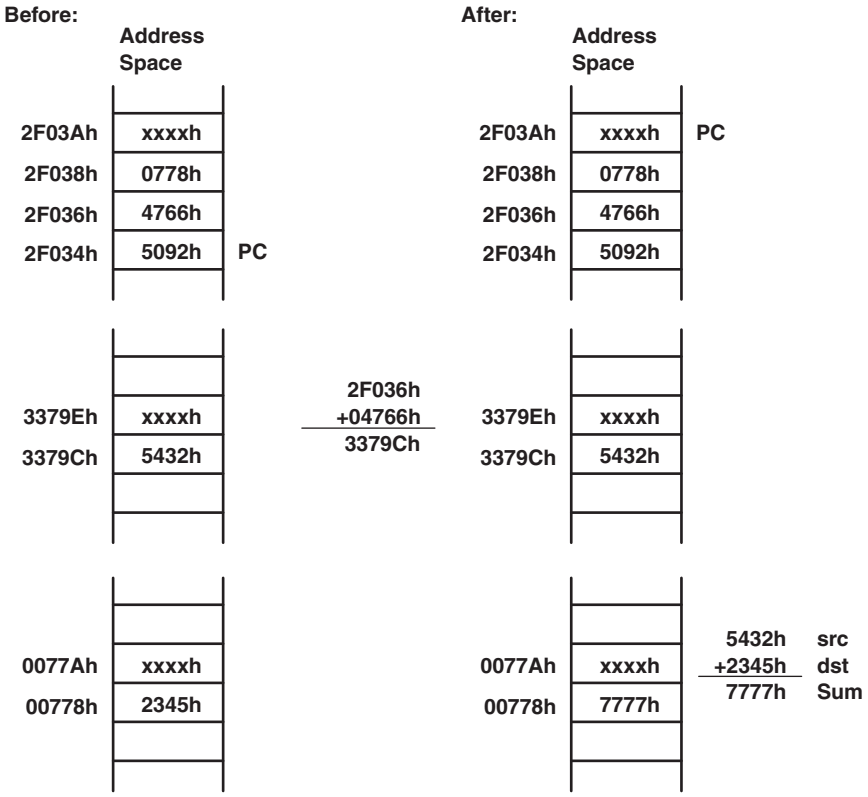
注释： 对于源和目的有效。 汇编程序计算 PC 索引并将其插入。

示例： `ADD.W EDEM&TONIM`

这个指令加上包含在源字 EDE 和目的字 TONI 中的 16 位数据并且将结果放置在目的字 TONI 中。 对于此示例，此指令位于地址 2F034h 中。

源： 由 PC 指向的位于地址 3379Ch 内的 EDE+4766h，是一个 3379Ch-2F036h=04766h 的 16 位结果。 地址 2F036h 是针对这个示例的索引位置。

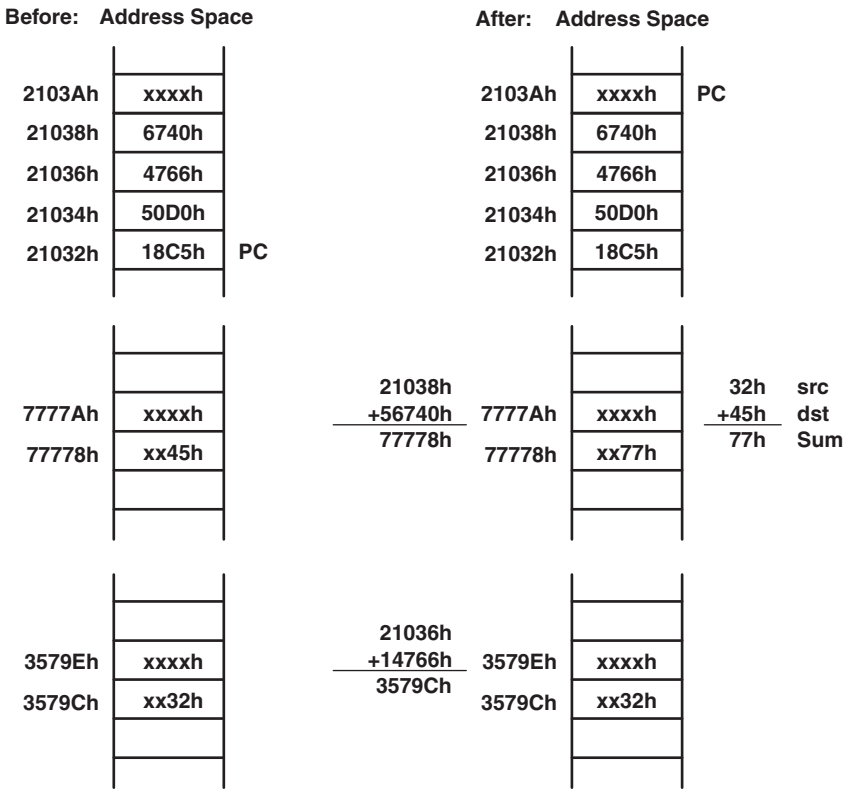
目标： 字 TONI 位于由绝对字 00778h 指向的地址 00778h 内。



4.4.3.3 具有符号模式的 MSP430X 指令

当使用具有符号模式的 MSP430X 指令时，操作数可位于 Rn 的范围 + 19 位的任一位置。

长度：	三个或者四个字
运行：	操作数地址是 20 位 PC 和 20 位索引的和。 索引的 4 个 MSB 包含在扩展字中；16 个 LSB 被包含在随后指令的字中。
注释：	对于源和目的有效。 汇编程序计算寄存器索引并将其插入。
示例：	`ADDX.B EDEM TONIM` 这个指令加上包含在源字节 EDE 和目的字节 TON 中的 8 位数据并且将结果放置在目的字节 TONI 中。
源：	由 PC 指向的位于地址 3579Ch 内的字节 EDE+14766h，是 3579Ch-21036h=14766h 的 20 位结果。 地址 21036h 是针对这个示例的索引的位置。
目标：	由 PC 指向的位于地址 77778h 的字节 TONI+F470h，是 77778h- 21038h=FF56740h 的截短的 20 位结果。 地址 21038h 是这个示例中的索引的地址。



4.4.4 绝对模式

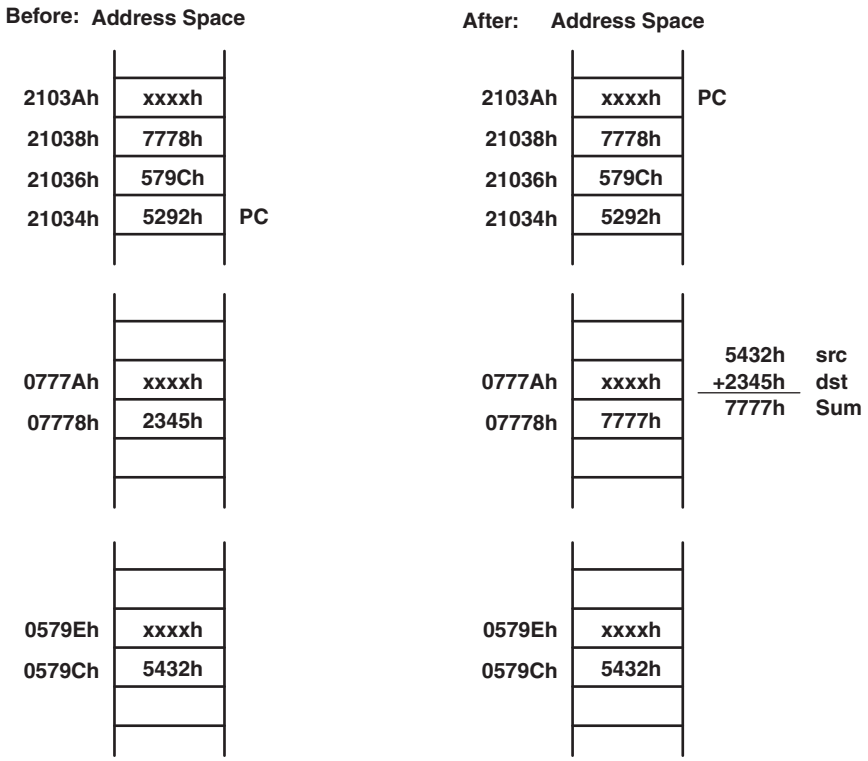
绝对模式使用指令之后的字的内容作为操作数的地址。 绝对模式有两个各不同的可能：

- 低 64KB 存储器中的绝对模式
- 具有绝对模式的 MSP430X 指令

4.4.4.1 低 64KB 中的绝对模式

如果 MSP430 指令在绝对寻址模式中使用，绝对地址是一个 16 位值并因此指向一个低 64KB 存储器范围的地址。 此地址被计算为一个来自 0 的索引并且被存储在指令后的字中。RAM 和外设寄存器可用这种方式访问并且现有的 MSP430 软件在不许修改的情况下即可用。

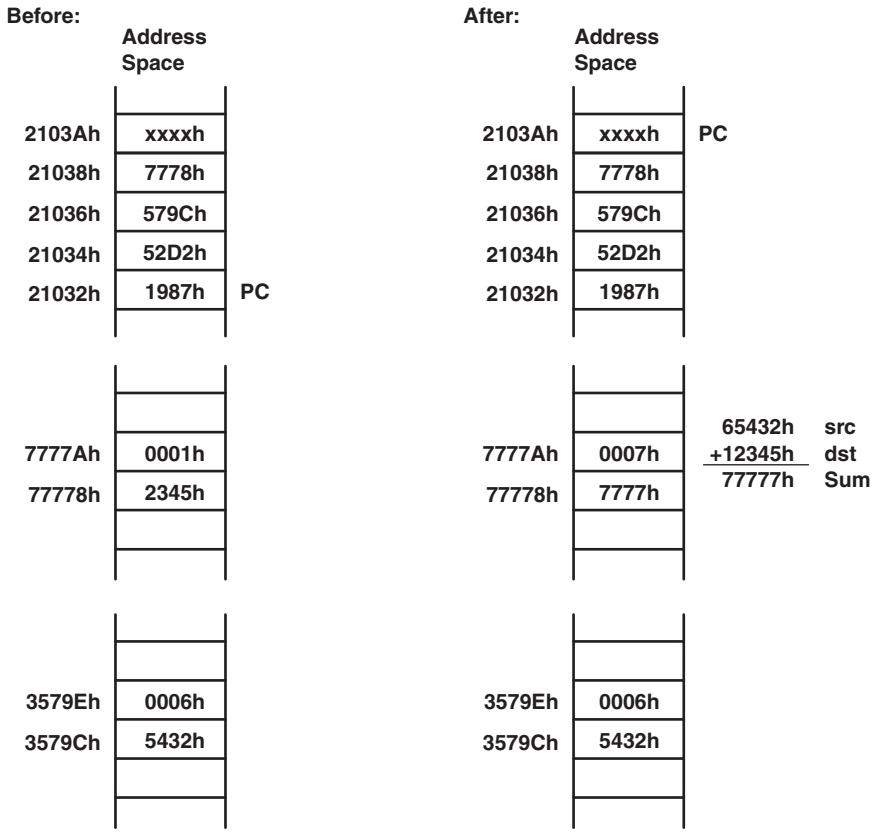
长度：	两个或者三个字
运行：	操作数是已寻址存储器位置的内容。
注释：	对于源和目的有效。 汇编程序从 0 开始计算索引并将其插入。
示例：	`ADD.W &EDEM&TONIM` 这条指令加上包含在绝对源和目的地址中的 16 位数据并且将结果放置在目的地址内。
源：	地址 EDE 上的字
目标：	地址 TONI 上的字



4.4.4.2 具有绝对模式的 MSP430X 指令

如果用绝对寻址模式使用 MSP430X 指令，绝对地址是一个 20 位值，并因指向存储器范围内的任一地址。地址值被计算为一个来自 0 索引。索引的 4 个 MSP 包含在扩展字中，并且 16 个 LSB 被包含在指令之后的字中。

长度：	三个或者四个字
运行：	操作数是已寻址存储器位置的内容。
注释：	对于源和目的有效。 汇编程序从 0 计算索引并且将其插入。
示例：	`ADDX.A &EDEM&TONIM`
	这条指令加上包含在源和目的地址中的 20 位数据并且将结果放置在目的地址内。
源：	从地址 EDE 开始的两个字
目的：	从地址 TONI 开始的两个字



4.4.5 间接寄存器模式

此间接寄存器模式将 CPU 寄存器 Rsrc 用作源操作数。 此间接寄存器模式一直使用一个 20 位地址。

长度： 一个、两个、或者三个字

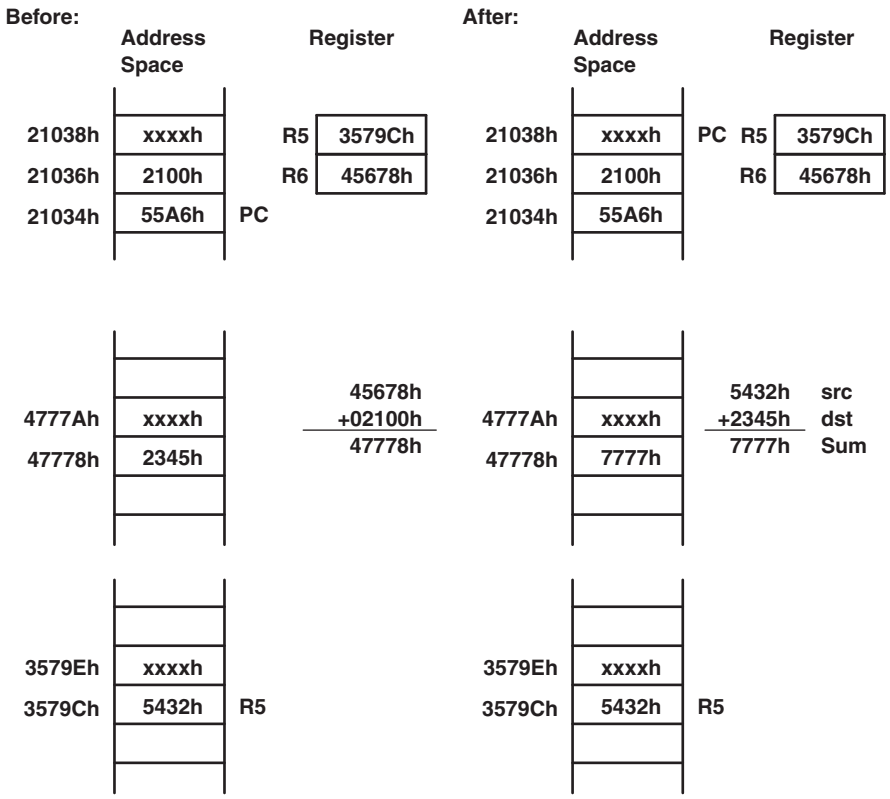
运行： 此操作数是已寻址地址位置的内容。 源寄存器 Rsrc 未修改。

注释： 只对源操作数有效。 目标操作数的替代值为 0 (Rdst)。

示例：
```
ADDX.W @R5M2100h(R6)
```
这条指令加上包含在源和目的地址中的 16 位操作数数据并且将结果放置在目的地址内。

源： R5 指向的字。 R5 包含针对这个示例的地址 3579Ch。

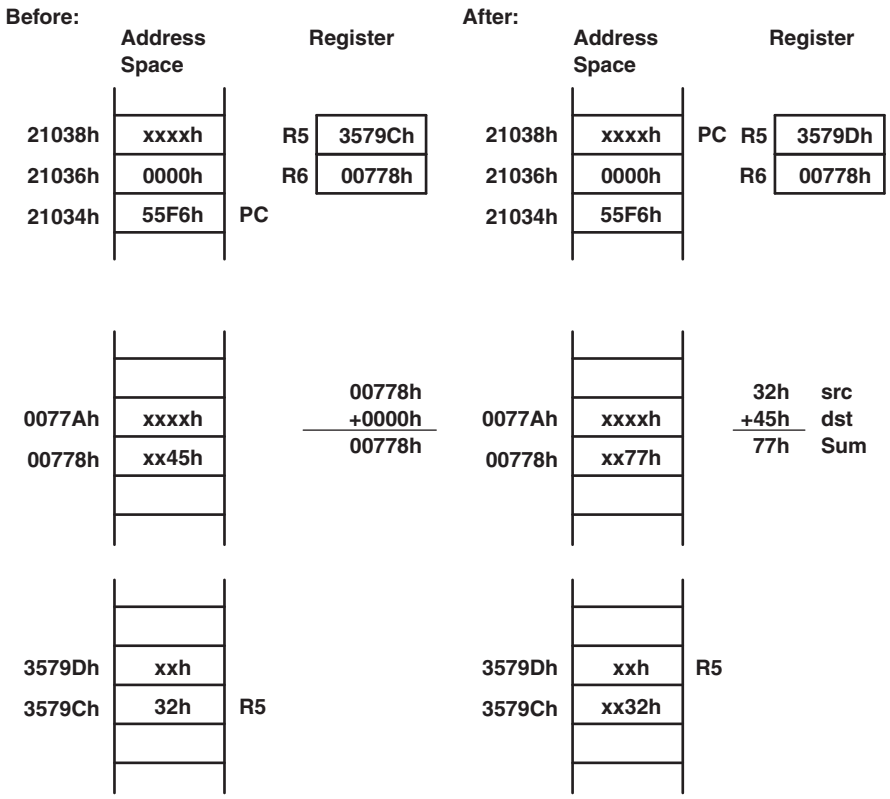
目的： R6 指向的字 + 2100h 得到地址 45678h+ 2100h=7778h。



4.4.6 间接自动递增模式

间接自动递增模式将 CPU 寄存器的内容用作源操作数。 然后，访问源操作数之后，Rsrc 立即针对字节指令自动加 1，针对字指令自动加 2，针对地址字指令加 4。 如果用于源和目的的寄存器是同一个寄存器，它包含针对目的地址的已增量地址。 间接自动递增模式一直使用 20 位地址。

长度：	一个、两个、或者三个字
运行：	操作数是已寻址存储器位置的内容。
注释：	只针对源操作数有效
示例：	`ADD.B @R5+M0(R6)` 这条指令加上包含在源和目的地址中的 8 位数据并且将结果放置在目的地址内。
源：	R5 指向的字节 在本示例中，R5 包含地址 3579Ch。
目的：	R6 指向的字节 + 0h，得到针对本示例的地址 0778h。



4.4.7 立即模式

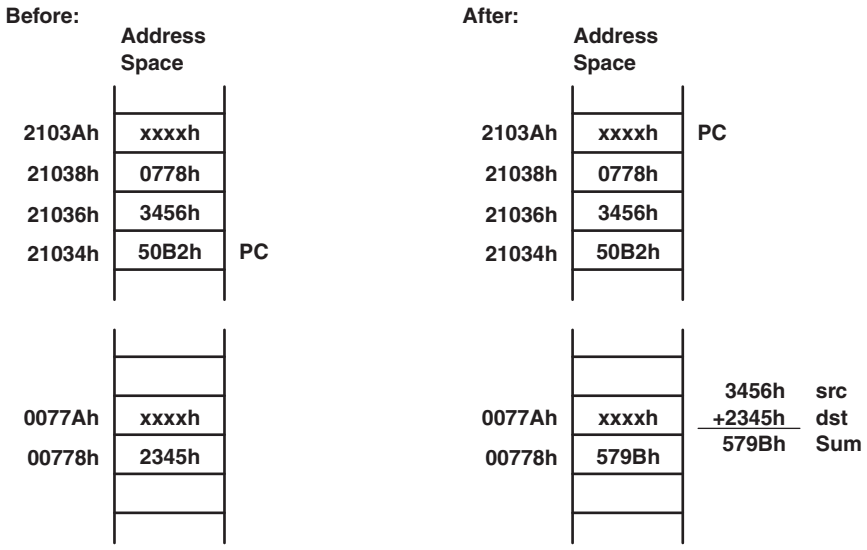
通过在指令之后将常数包含在存储器位置内，立即模式可实现将常数作为操作数进行访问。 PC 使用间接自动递增模式。 PC 指向下一个字中包含的立即值。 在取得立即操作数后，针对字节、字、或者地址-字指令，PC 增加 2。 立即模式有两个寻址可能：

- MSP430 指令时的 8 位或 16 位常数
- MSP430X 指令时的 20 位常数

4.4.7.1 支持立即模式的 MSP430 指令

如果 MSP430 指令使用立即寻址模式，常数为一个 8 或 16 位的值，并且存储在指令后的字中。

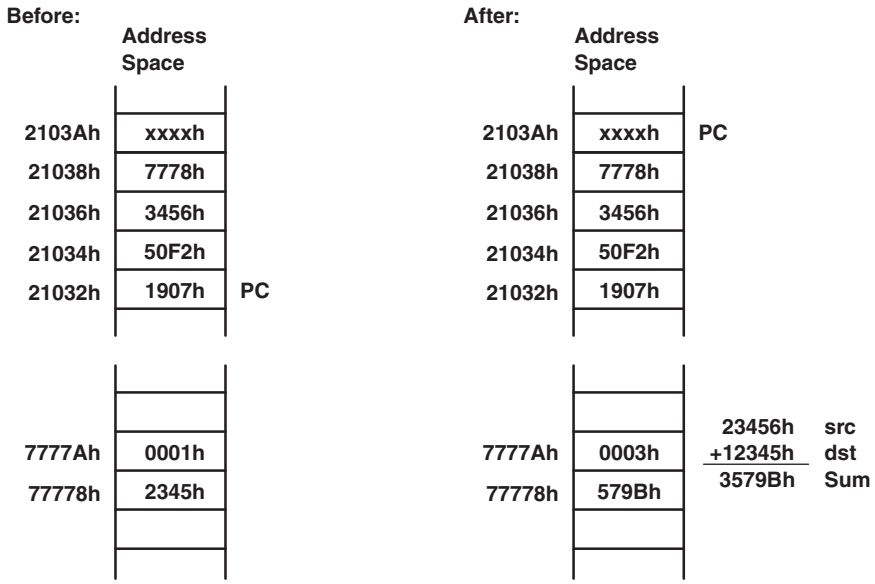
长度：	两个或者三个字。 如果常数发生器的一个常数可被用于立即操作数，则少一个字。
运行：	16 位立即源操作数与 16 位目的操作数一起使用。
注释：	只对源操作数有效
示例：	`ADD #3456hM&TONI` 这条指令将 16 位立即操作数 3456h 添加到目的地址 TONI 中的数据内。
源：	16 位立即值 3456h
目的：	地址 TONI 上的字



4.4.7.2 支持立即模式的 MSP430X 指令

如果一个 MSP430X 指令与立即寻址模式一起使用，常数为一个 20 位的值。 常数的 4 MSB 存储在扩展字中，而常数的 16 LSB 存储在指令后的字中。

长度： 三个或者四个字。 如果常数发生器的一个常数可被用于立即操作数，则少一个字。
运行： 20 位立即源操作数与 20 位目的操作数一起使用。
注释： 只对源操作数有效
示例： `ADDX.A #23456hM&TONIM`
这条指令将 20 位立即操作数 23456h 添加到目的地址 TONI 中的数据内。
源： 20 位立即值 23456h
目的： 以地址 TONI 为开始的两个字



4.5 MSP430 和 MSP430X 指令

MSP430 指令是 MSP430 CPU 执行的 27 条指令。 这些指令在 1MB 存储器范围内使用，除非超过了它们的 16 位能力。 当操作数寻址，或者数据长度超过 MSP430 指令的 16 位能力时，MSP430X 指令被使用。

当在 MSP430 和 MSP430X 指令间进行选择时，有三个可能：

- 只使用 MSP430 指令-唯一的例外是 CALLA 和 RETA 指令。 如果符合几个简单规则的话，可实现此目的：
  - 将所有常数、变量、数组、表格、和数据放置在低 64KB 空间内。 这样可针对所有数据访问的 16 位寻址使用 MSP430 指令。 无需具有 20 位地址的指针。
  - 将子例程常数紧接着子例程代码放置。 这样可使用符号寻址模式，此模式的 16 位索引能够达到 PC + 32KB 范围内的地址。
- 只使用 MSP430X 指令-这个方法的劣势是由额外 CPU 周期而导致的速度降低以及由于任一双操作数指令的所需扩展字而导致的程序空间增加。
- 按照需要选择最合适的指令。

下面的指令列表描述了 MSP430 和 MSP430X 指令。

4.5.1 MSP430 指令

无论程序是驻留在低 64KB 还是 驻留在其之上的空间，都可使用 MSP430 指令。 唯一的例外时是指令 CALL 和 RET，这两个指令被限制在低 64KB 地址范围。 CALLA 和 RETA 已经被添加到 MSP430X CPU 中来处理整个地址范围内的子例程，而又无需代码尺寸开销。

4.5.1.1 MSP430 双操作数（格式 I）指令

图 4-21显示了 MSP430 双操作数指令的格式。 针对已索引、符号、绝对、和立即模式，添加了源和目的字。表 4-4列出了 12 个 MSP430 双操作数指令。

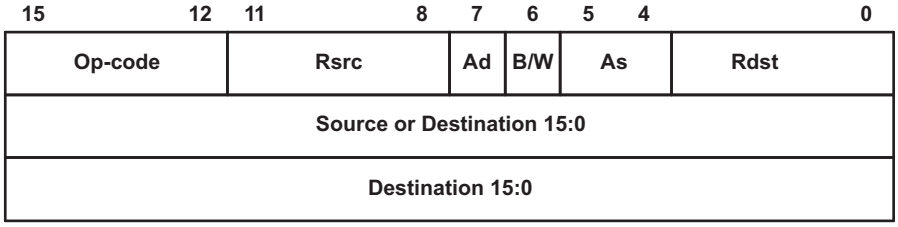


图 4-21. MSP430 双操作数指令格式

表 4-4. MSP430 双操作数指令

助记符	S-Reg，D-Reg	运行	状态位(1)			
			V	N	Z	C
MOV(.B)	src，dst	src→dst	-	-	-	-
ADD(.B)	src，dst	src+dst→dst	*	*	*	*
ADDC(.B)	src，dst	src+dst+C→dst	*	*	*	*
SUB(.B)	src，dst	dst+.not.src+1→dst	*	*	*	*
SUBC(.B)	src，dst	dst+.not.src+C→dst	*	*	*	*
CMP(.B)	src，dst	dst→src	*	*	*	*
DADD(.B)	src，dst	src+dst+C→dst（用十进制）	*	*	*	*
BIT(.B)	src，dst	src .and. dst	0	*	*	$\overline{Z}$
BIC(.B)	src，dst	.not.src .and. dst→dst	-	-	-	-
BIS(.B)	src，dst	src .or. dst→dst	-	-	-	-
XOR(.B)	src，dst	src .xor. dst→dst	*	*	*	$\overline{Z}$
AND(.B)	src，dst	src .and. dst→dst	0	*	*	$\overline{Z}$

(1) * = 状态位受影响。
- = 状态位未受影响。
0 = 状态位被清零。
1 = 状态位被置位。

4.5.1.2 MSP430 单操作数（格式 II）指令

图 4-22显示了针对 MSP430 单操作数指令的格式，除 RETI 之外。 针对已索引、符号、绝对、和立即模式附加了目的字。表 4-5列出了七条单操作数指令。

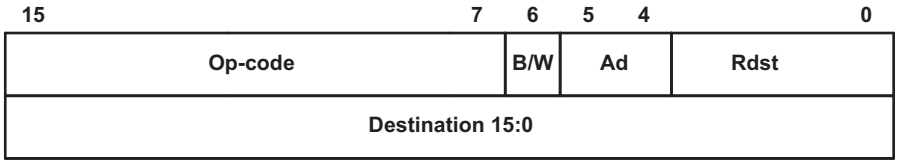


图 4-22. MSP430 单操作数指令

表 4-5. MSP430 单操作数指令

助记符	S-Reg，D-Reg	运行	状态位(1)			
			V	N	Z	C
RRC(.B)	dst	C→MSB→.....LSB→C	*	*	*	*
RRA(.B)	dst	MSB→MSB→....LSB→C	0	*	*	*
PUSH(.B)	src	SP-2→SP，src→SP	-	-	-	-
SWPB	dst	位 15...位 8 ↔ 位 7...位 0	-	-	-	-
CALL	dst	在低 64KB 中调用子例程	-	-	-	-
RETI		TOS→SR，SP+2→SP	*	*	*	*
		TOS→PC，SP+2→SP				
SXT	dst	寄存器模式：位 7 → 位 8...位 19 其它模式：位 7 → 位 8...位 15	0	*	*	$\overline{Z}$

(1) * = 状态位受影响。
– = 状态位未受影响。
0 = 状态位被清零。
1 = 状态位被置位。

4.5.1.3 跳转指令

图 4-23显示了 MSP430 和 MSP430X 跳转指令的格式。 跳转指令的带符号 10 位字偏移乘以 2，符号扩展至一个 20 位地址，并加至 20 位 PC。 这可实现相对于完全 20 位地址空间内的 PC 的 -511 至 +512 字范围内的跳转。 跳转并不影响状态位。表 4-6列出并描述了八个跳转指令。

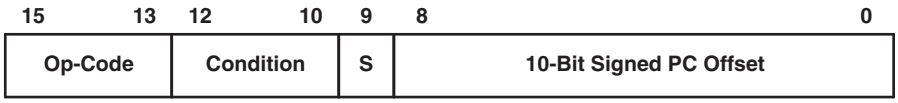


图 **4-23.** 条件跳转指令的格式

表 **4-6.** 条件跳转指令

助记符	S-Reg，D-Reg	运行
JEQ/JZ	标签	如果零位被置位则跳转至标签
JNE/JNZ	标签	如果零位被复位则跳转至标签
JC	标签	如果进位位被置位则跳转至标签
JNC	标签	如果进位位被复位则跳转至标签
JN	标签	如果负位被置位则跳转至标签
JGE	标签	如果 (N .XOR. V)=0，则跳转至标签
JL	标签	如果 (N .XOR. V)=1，则跳转至标签
JMP	标签	无条件跳转至标签

4.5.1.4 仿真指令

除了 MSP430 和 MSP430X 指令，仿真指令是使代码更容易进行写入和读取，但是本身不具有运算代码的指令。 作为替代，它们自动被具有一个核心指令的汇编程序所取代。 使用仿真指令并不会产生代码或者性能损失。 表 4-7中列出了仿真指令。

表 **4-7.** 仿真指令

指令	说明	仿真	状态位[1]			
			V	N	Z	C
ADC(.B) dst	将进位增加至 dst	ADDC(.B) #0,dst	*	*	*	*
BR dst	分支指令间接 dst	MOV dst,PC	-	-	-	-
CLR(.B) dst	清零 dst	MOV(.B) #0,dst	-	-	-	-
CLRC	清零进位位	BIC #1,SR	-	-	-	0
CLRN	清零负位	BIC #4,SR	-	0	-	-
CLRZ	清零零位	BIC #2,SR	-	-	0	-
DADC(.B) dst	用十进制将进位增加至 dst	DADD(.B) #0,dst	*	*	*	*
DEC(.B) dst	dst 减 1	SUB(.B) #1,dst	*	*	*	*
DECD(.B) dst	dst 减 2	SUB(.B) #2,dst	*	*	*	*
DINT	禁用中断	BIC #8,SR	-	-	-	-
EINT	启用中断	BIS #8,SR	-	-	-	-
INC(.B) dst	dst 增 1	ADD(.B) #1,dst	*	*	*	*
INCD(.B) dst	dst 增 2	ADD(.B) #2,dst	*	*	*	*
INV(.B) dst	反转 dst	XOR(.B) #-1,dst	*	*	*	*
NOP	无操作	MOV R3,R3	-	-	-	-

(1) * = Status bit is affected; 状态位受影响；– = 状态位未受影响；0 = 状态位被清零；1 = 状态位被置位。

表 4-7. 仿真指令 (continued)

指令	说明	仿真	状态位(1)			
			V	N	Z	C
POP dst	从堆栈中弹出操作数	MOV @SP+,dst	-	-	-	-
RET	从子例程返回	MOV @SP+,PC	-	-	-	-
RLA(.B) dst	算术左移 dst	ADD(.B) dst,dst	*	*	*	*
RLC(.B) dst	通过进位逻辑左移 dst	ADDC(.B) dst,dst	*	*	*	*
SBC(.B) dst	从 dst 中减去进位	SUBC(.B) #0,dst	*	*	*	*
SETC	置位进位位	BIS #1,SR	-	-	-	1
SETN	置位负位	BIS #4,SR	-	1	-	-
SETZ	置位零位	BIS #2,SR	-	-	1	-
TST(.B) dst	测试 dst（与 0 相比较）	CMP(.B) #0,dst	0	*	*	1

4.5.1.5 MSP430 指令执行

一个指令所需的 CPU 时钟周期的数量取决于指令格式和使用的寻址模式-而不是指令本身。 参考 MCLK 的时钟周期数量。

4.5.1.5.1 针对中断、复位、和子例程的指令周期和长度

表 4-8列出了针对复位、中断、和子例程的长度和 CPU 周期

表 4-8. 中断、返回、和复位周期以及长度

操作	执行时间（MCLK 周期）	指令长度（字）
从中断 RETI 返回	3(1)	1
从子例程 RET 返回	3	1
中断请求处理（第一个指令前需要的周期）	5(2)	-
WDT 复位	4	-
复位 (RST/NMI)	4	-

(1) MSP430 CPU 中的周期数量为 5。
(2) MSP430 CPU 中的周期数量为 6。

4.5.1.5.2 格式 II（单操作数）指令周期和长度

表 4-9列出了针对 MSP430 单操作数指令的所有寻址模式的长度和 CPU 周期。

表 4-9. MSP430 格式 II 指令周期和长度

寻址模式	周期的数量			指令的长度	示例
	RRA，RRC SWPB，SXT	PUSH	CALL		
Rn	1	3	3(1)	1	SWPB R5
@Rn	3	3(1)	4	1	RRC @R9
@Rn+	3	3(1)	4(2)	1	SWPB @R10+
#N	不可用	3(1)	4(2)	2	CALL #LABEL
X(Rn)	4	4(2)	4(2)	2	CALL 2(R7)
EDE	4	4(2)	4(2)	2	PUSH EDE

(1) MSP430 CPU 中的周期数量为 4。
(2) MSP430 CPU 中的周期数量为 5。此外，当 Rn=SP 时，X(Rn) 寻址模式的周期数量为 5。

表 4-9. MSP430 格式 II 指令周期和长度 (continued)

寻址模式	周期的数量			指令的长度	示例
	RRA，RRC SWPB，SXT	PUSH	CALL		
&EDE	4	4(2)	4(2)	2	SXT &EDE

4.5.1.5.3 跳转指令周期和长度

所有跳转指令要求一个代码字并且花费两个 CPU 周期来执行，无论跳转是否发生。

4.5.1.5.4 格式 I（双操作数）指令周期和长度

表 4-10列出了所有针对 MSP430 格式 I 指令寻址模式的长度和 CPU 周期。

表 4-10. MSP430 格式 I 指令周期和长度

寻址模式		周期的数量	指令长度	示例
Src	Dst			
Rn	Rm	1	1	MOV R5，R8
	PC	2	1	BR R9
	x(Rm)	4(1)	2	ADD R5，4(R6)
	EDE	4(1)	2	XOR R8，EDE
	&EDE	4(1)	2	MOV R5，&EDE
@Rn	Rm	2	1	和 @R4，R5
	PC	3	1	BR @R8
	x(Rm)	5(1)	2	XOR @R5，8(R6)
	EDE	5(1)	2	MOV @R5，EDE
	&EDE	5(1)	2	XOR @R5，&EDE
@Rn+	Rm	2	1	ADD @R5+，R6
	PC	3	1	BR @R9+
	x(Rm)	5(1)	2	XOR @R5，8(R6)
	EDE	5(1)	2	MOV @R9+，EDE
	&EDE	5(1)	2	MOV @R9+，&EDE
#N	Rm	2	2	MOV #20，R9
	PC	3	2	BR #2AEh
	x(Rm)	5(1)	3	MOV #0300h，0(SP)
	EDE	5(1)	3	ADD #33，EDE
	&EDE	5(1)	3	ADD #33，&EDE
x(Rn)	Rm	3	2	MOV 2(R5)，R7
	PC	3	2	BR 2(R6)
	TONI	6(1)	3	MOV 4(R7)，TONI
	x(Rm)	6(1)	3	ADD 4(R4)，6(R9)
	&TONI	6(1)	3	MOV 2(R4)，&TONI
EDE	Rm	3	2	AND EDE，R6
	PC	3	2	BR EDE
	TONI	6(1)	3	CMP EDE，TONI
	x(Rm)	6(1)	3	MOV EDE，0(SP)
	&TONI	6(1)	3	MOV EDE，&TONI

(1) MOV，BIT，和 CMP 指令在少一个周期内执行。

表 4-10. MSP430 格式 I 指令周期和长度 (continued)

寻址模式		周期的数量	指令长度	示例
Src	Dst			
&EDE	Rm	3	2	MOV &EDE，R8
	PC	3	2	BR &EDE
	TONI	6(1)	3	MOV &EDE，TONI
	x(Rm)	6(1)	3	MOV &EDE，0(SP)
	&TONI	6(1)	3	MOV &EDE，&TONI

4.5.2 MSP430 扩展指令

扩展 MSP430X 指令使得 MSP430X CPU 可完全访问其 20 位地址范围。 大多数 MSP430X 指令要求一个被称为扩展字的运算代码的附加字。 一些扩展指令无需附加字并且未在指令说明中注明。 当前面为扩展字时，所有地址、索引、和立即数有 20 位的值。

有两种类型的扩展字：

- 针对格式 I 指令的寄存器/寄存器模式和针对格式 II 的寄存器模式。
- 针对所有其它地址模式组合的扩展字

4.5.2.1 寄存器模式扩展字

此寄存器模式扩展字显示在图 4-24中并在表 4-11中进行了说明。 图 4-26显示了一个示例。

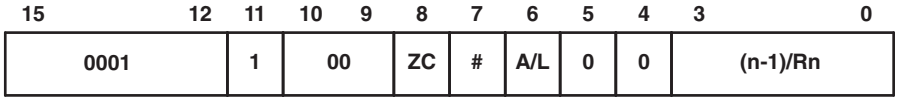


图 4-24. 针对寄存器模式的扩展字

表 4-11. 针对寄存器模式的扩展字的说明

位	说明		
15:11	扩展字运算代码。 运算代码 1800h 至 1FFFh 为扩展字。		
10:9	被保留		
ZC	零进位 0 被执行的指令使用进位位 C 的状态。 1 被执行的指令将进位位用作 0。指令执行后，进位位由最终运算的结果定义。		
#	重复 0 指令重复的次数由扩展字位 3:0 置位。 1 指令重复的次数由 Rn 的四个 LSB 的值定义。 位 3:0 请见说明。		
A/L	数据长度扩展。 与下面的 MSP430 指令的 B/W 位一起，AL 位定义了指令所使用的数据长度。		
	A/L	B/W	注释
	0	0	被保留
	0	1	20 位地址字
	1	0	16 位字
	1	1	8 位字节
5:4	被保留		
3:0	重复数量 # = 0 这四个位置位重复数量 n。 这些位包含 n-1。 # = 1 这四个位定义了 CPU 寄存器，此寄存器的位 3:0 置位重复的数量。 Rn 3:0 包含 n-1。		

4.5.2.2 非寄存器模式扩展位

针对非寄存器模式的扩展字显示在图 4-25中并在表 4-12中进行了说明。 图 4-27显示了一个示例。

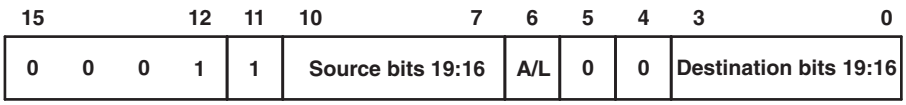


图 **4-25.** 针对非寄存器模式的扩展位

表 **4-12.** 针对非寄存器模式的扩展字的说明

位	说明		
15:11	扩展字运算代码。 运算代码 1800h 至 1FFFh 是扩展字。		
源位 19:16	20 位源的四个 MSB。 根据源寻址模式，这四个 MSB 有可能属于一个立即操作数，一个索引或者一个绝对地址。		
A/L	数据长度扩展。 与下面的 MSP430 指令的 B/W 一起，AL 位定义了指令所使用的数据长度。		
	A/L	**B/W**	注释
	0	0	被保留
	0	1	20 位地址字
	1	0	16 位字
	1	1	8 位字节
5:4	被保留		
目的位 19:16	20 位目的的四个 MSB。 根据目的寻址模式，这四个 MSB 有可能属于一个索引或者一个绝对地址。		

注: 针对 **SWPBX** 和 **SXTX** 的 **B/W** 和 **A/L** 位设置

A/L	**B/W**	
0	0	SWPBX.A，SXTX.A
0	1	不可用
1	0	SWPB.W，SXTX.W
1	1	不可用

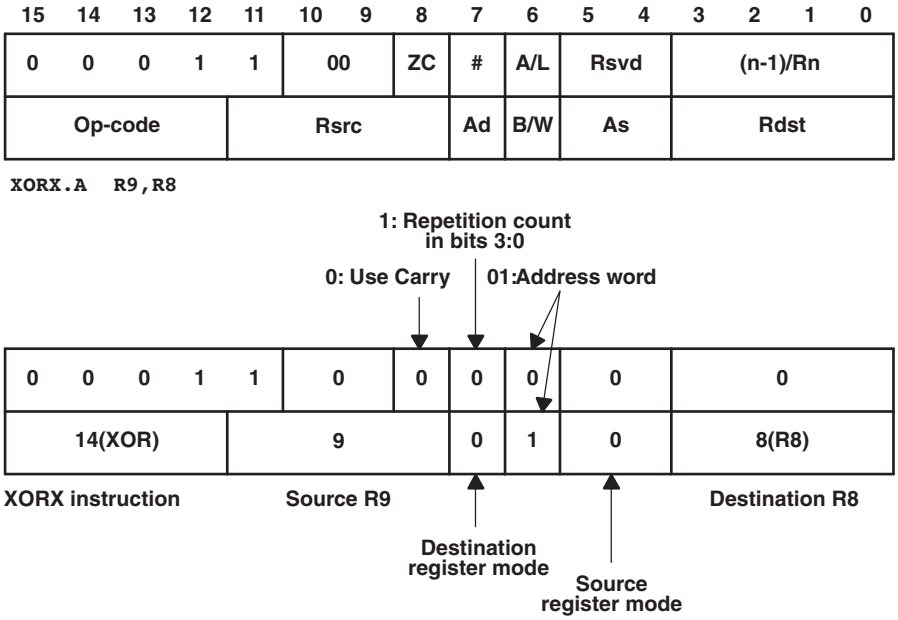


图 **4-26.** 针对扩展寄存器/寄存器指令的示例

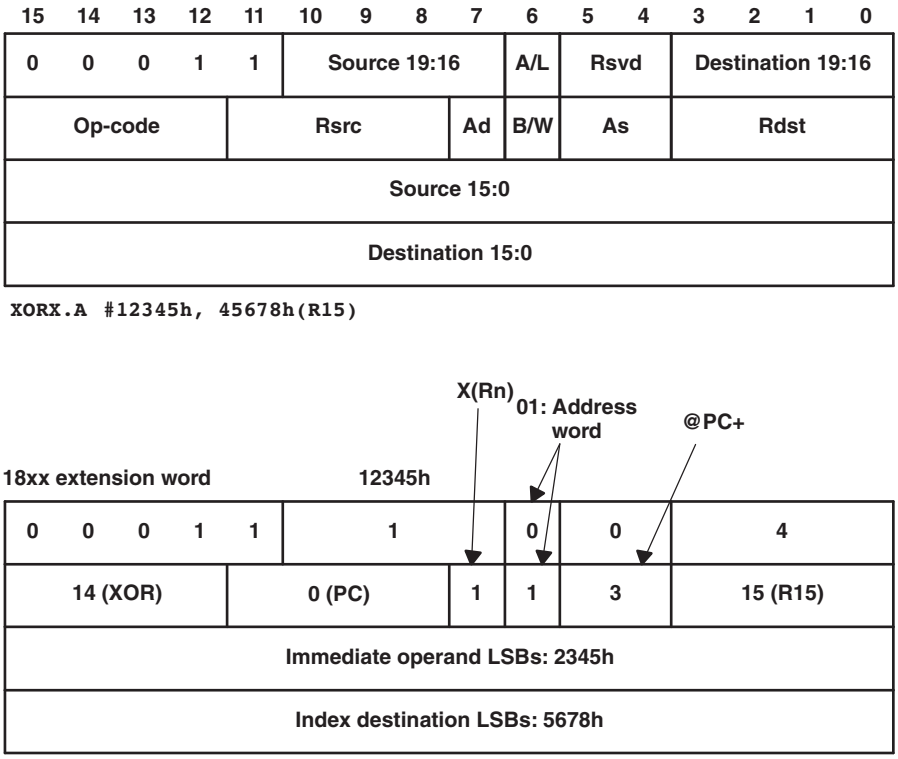


图 **4-27.** 针对扩展立即**/**已索引指令的示例

4.5.2.3 扩展双操作数（格式 I）指令

所有 12 个双操作数指令具有如表 4-13所列的扩展版本。

表 **4-13.** 扩展双操作数指令

助记符	操作数	运行	状态位[1]			
			V	**N**	**Z**	**C**
MOVX(.B,.A)	src，dst	src→dst	-	-	-	-
ADDX(.B,.A)	src，dst	src+dst→dst	*	*	*	*
ADDCX(.B,.A)	src，dst	src+dst+C→dst	*	*	*	*
SUBX(.B,.A)	src，dst	dst + .not.src + 1 → dst	*	*	*	*
SUBCX(.B,.A)	src，dst	dst + .not.src + C → dst	*	*	*	*
CMPX(.B,.A)	src，dst	dst-src	*	*	*	*
DADDX(.B,.A)	src，dst	src + dst + C → dst（十进制）	*	*	*	*
BITX(.B,.A)	src，dst	src .and. dst	0	*	*	$\overline{Z}$
BICX(.B,.A)	src，dst	.not.src .and. dst→dst	-	-	-	-
BISX(.B,.A)	src，dst	src .or. dst→dst	-	-	-	-
XORX(.B,.A)	src，dst	src .xor. dst→dst	*	*	*	$\overline{Z}$
ANDX(.B,.A)	src，dst	src .and. dst→dst	0	*	*	$\overline{Z}$

(1) * = 状态位受影响。
- = 状态位未受影响。
0 = 状态位被清零。
1 = 状态位被置位。

针对格式 I 指令扩展字的四个可能的寻址组合显示在图 4-28中。

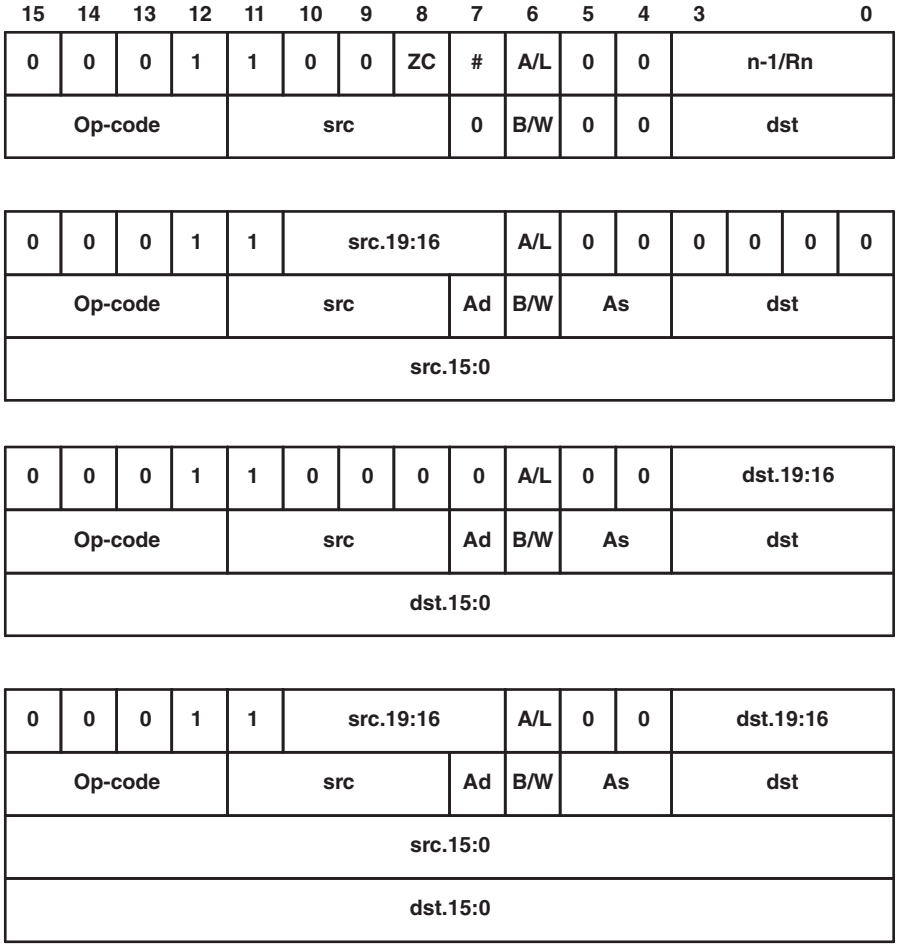


图 **4-28.** 扩展格式 **I** 指令格式

如果一个源或者目的操作数的 20 位地址被锁定在存储器中，而非一个 CPU 寄存器中，那么用于这个操作数的两个字显示在图 4-29中。

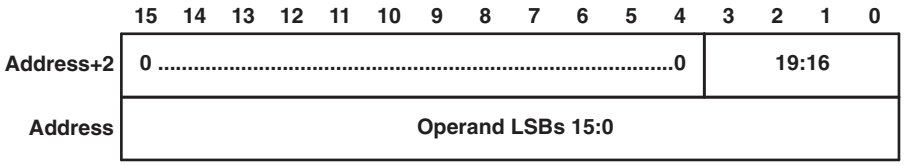


图 **4-29.** 存储器中的 **20** 位地址

4.5.2.4 扩展单操作数（格式 II）指令

表 4-14中列出了扩展 MSP430X 格式 II 指令。

表 4-14. 扩展单操作数指令

助记符	操作数	运行	n	状态位(1) V	N	Z	C
CALLA	dst	间接调用子例程（20 位地址）		-	-	-	-
POPM.A	#n，Rdst	从堆栈弹出 n 个 20 位寄存器	1 至 16	-	-	-	-
POPM.W	#n，Rdst	从堆栈弹出 n 个 16 位寄存器	1 至 16	-	-	-	-
PUSHM.A	#n，Rsrc	将 n 个 20 位寄存器压入堆栈	1 至 16	-	-	-	-
PUSHM.W	#n，Rsrc	将 n 个 16 位寄存器压入堆栈	1 至 16	-	-	-	-
PUSHX(.B,.A)	src	将 8/16/20 位源压入堆栈		-	-	-	-
RRCM(.A)	#n，Rdst	通过进位将 Rdst 右旋 n 位（16/20 位寄存器）	1 至 4	0	*	*	*
RRUM(.A)	#n，Rdst	将 Rdst 右旋 n 个无符号位（16/20 位寄存器）	1 至 4	0	*	*	*
RRAM(.A)	#n，Rdst	将 Rdst 算术右旋 n 个位（16/20 位寄存器）	1 至 4	*	*	*	*
RLAM(.A)	#n，Rdst	将 Rdst 算术左旋 n 个位（16/20 位寄存器）	1 至 4	*	*	*	*
RRCX(.B,.A)	dst	通过进位右旋 dst（8/16/20 位数据）	1	0	*	*	*
RRUX(.B,.A)	Rdst	右旋 dst 无符号位（8/16/20 位）	1	0	*	*	*
RRAX(.B,.A)	dst	算术右旋 dst	1	*	*	*	*
SWPBX(.A)	dst	用高字节交换低字节	1	-	-	-	-
SXTX(.A)	Rdst	位 7 → 位 8 ... 位 19	1	0	*	*	*
SXTX(.A)	dst	位 7 → 位 8 ... MSB	1	0	*	*	*

(1) * = 状态位受影响。
- = 状态位未受影响。
0 = 状态位被清零。
1 = 状态位被置位。

针对格式 II 指令的三个可能寻址模式组合显示在图 4-30中。

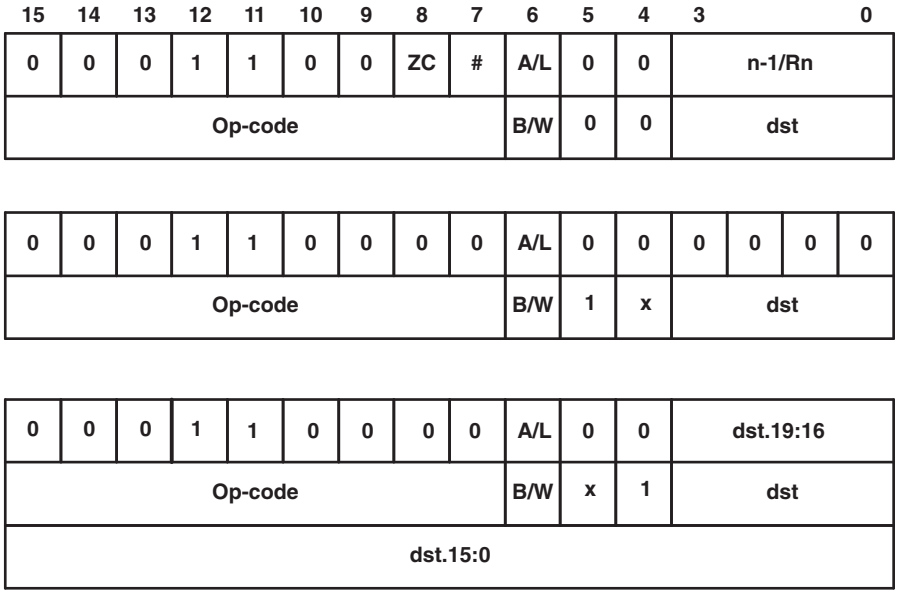


图 4-30. 扩展格式 II 指令格式

4.5.2.4.1 扩展格式 II 指令格式除外

针对格式 II 指令格式的例外显示在图 4-31至图 4-34中。

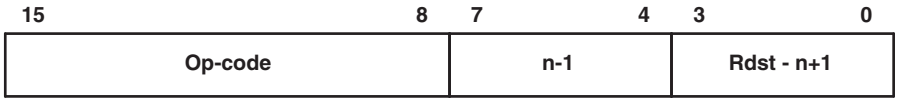


图 **4-31. PUSHM/POPM** 指令格式

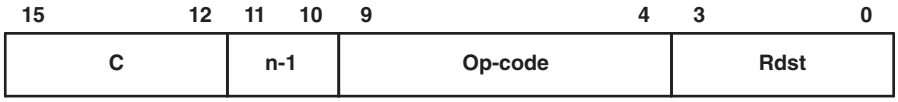


图 **4-32. RRCM，RRAM，RRUM** 和 **RLAM** 指令格式

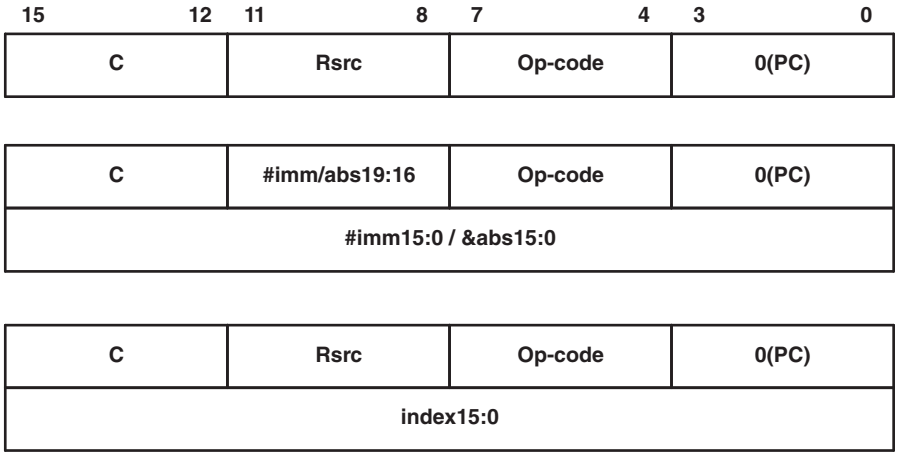


图 **4-33. BRA** 指令格式

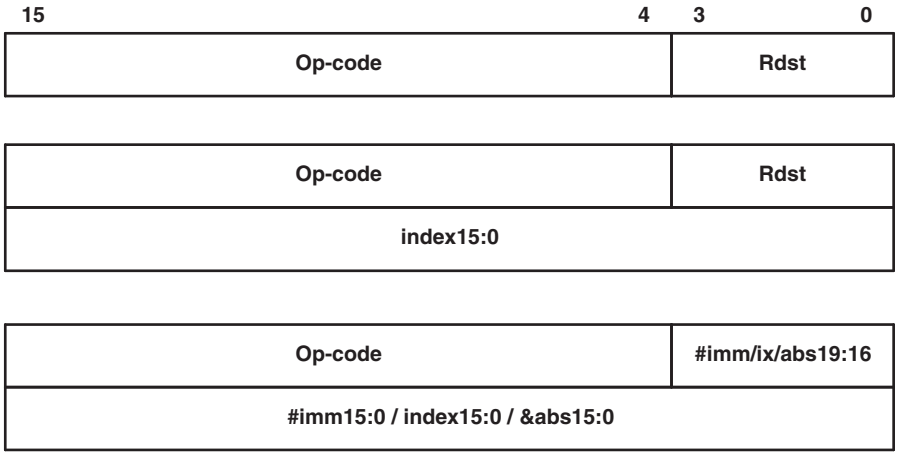


图 **4-34. CALLA** 指令格式

4.5.2.5 扩展仿真指令

扩展指令与常数发生器一起形成了扩展仿真指令。表 4-15列出了仿真指令。

表 4-15. 扩展仿真指令

指令	说明	仿真
ADCX(.B,.A) dst	将进位增加到 dst	ADDCX(.B,.A) #0M dst
BRA dst	分支指令 dst	MOVA dstM PC
RETA	从子例程返回	MOVA @SP+M PC
CLRA Rdst	清零 Rdst	MOV #0M Rdst
CLRX(.B,.A) dst	清零 dst	MOVX(.B,.A) #0M dst
DADCX(.B,.A) dst	用十进制为 dst 增加进位	DADDX(.B,.A) #0M dst
DECX(.B,.A) dst	dst 减 1	SUBX(.B,.A) #1M dst
DECDA Rdst	Rdst 减 2	SUBA #2M Rdst
DECDX(.B,.A) dst	dst 减 2	SUBX(.B,.A) #2M dst
INCX(.B,.A) dst	dst 增 1	ADDX(.B,.A) #1M dst
INCDA Rdst	Rdst 增 2	ADDA #2M Rdst
INCDX(.B,.A) dst	dst 增 2	ADDX(.B,.A) #2M dst
INVX(.B,.A) dst	反转 dst	XORX(.B,.A) #-1M dst
RLAX(.B,.A) dst	算术移位左侧 dst	ADDX(.B,.A) dstM dst
RLCX(.B,.A) dst	通过进位逻辑移位左侧 dst	ADDCX(.B,.A) dstM dst
SBCX(.B,.A) dst	从 dst 减去进位	SUBCX(.B,.A) #0M dst
TSTA Rdst	测试 Rdst（与 0 相比较）	CMPA #0M Rdst
TSTX(.B,.A) dst	测试 dst（与 0 相比较）	CMPX(.B,.A) #0M dst
POPX dst	弹出到 dst	MOVX(.B, .A) @SP+M dst

4.5.2.6 MSP430X 寻址指令

MSP430X 寻址指令支持 20 位操作数，但是具有受限的寻址模式。 寻址模式限制为寄存器模式和立即模式，除了表 4-16中列出的 MOVA 指令。 对寻址模式的限制免除了对于额外扩展字运算代码的需要，从而改进了代码密度和执行时间。 只要需要具有相应受限寻址模式的 MSP430X 指令，就应该使用寻址指令。

表 **4-16.** 寻址指令，在 **20** 位寄存器数据上运行

助记符	操作数	运行	状态位(1)			
			V	N	Z	C
ADDA	RsrcM Rdst	将源添加到目的寄存器	*	*	*	*
	#imm20M Rdst					
MOVA	RsrcM Rdst	将源移动到目的	-	-	-	-
	#imm20M Rdst					
	z16(Rsrc)M Rdst					
	EDEM Rdst					
	&abs20M Rdst					
	@RsrcM Rdst					
	@Rsrc+M Rdst					
	RsrcM z16(Rdst)					
	RsrcM &abs20					
CMPA	RsrcM Rdst	将源与目的寄存器相比较	*	*	*	*
	#imm20M Rdst					
SUBA	RsrcM Rdst	将源从目的寄存器中减去	*	*	*	*
	#imm20M Rdst					

(1) * = 状态位受影响。
- = 状态位未受影响。
0 = 状态位被清零。
1 = 状态位被置位。

4.5.2.7 MSP430X 指令执行

一个 MSP430X 指令所需的 CPU 时钟周期的数量取决于指令格式和使用的寻址模式，而不是指令本身。 参考 MCLK 的时钟周期数量。

4.5.2.7.1 MSP430X 格式 II（单操作数）指令周期和长度

表 4-17列出了针对 MSP430X 扩展单操作数指令的所有寻址模式的长度和 CPU 周期。

表 4-17. MSP430X 格式 II 指令周期和长度

指令	指令（字）的执行周期/长度						
	Rn	@Rn	@Rn+	#N	X(Rn)	EDE	&EDE
RRAM	n/1	-	-	-	-	-	-
RRCM	n/1	-	-	-	-	-	-
RRUM	n/1	-	-	-	-	-	-
RLAM	n/1	-	-	-	-	-	-
PUSHM	2+n/1	-	-	-	-	-	-
PUSHM.A	2+2n/1	-	-	-	-	-	-
POPM	2+n/1	-	-	-	-	-	-
POPM.A	2+2n/1	-	-	-	-	-	-
CALLA	4/1	5/1	5/1	4/2	6(1)/2	6/2	6/2
RRAX(.B)	1+n/2	4/2	4/2	-	5/3	5/3	5/3
RRAX.A	1+n/2	6/2	6/2	-	7/3	7/3	7/3
RRCX(.B)	1+n/2	4/2	4/2	-	5/3	5/3	5/3
RRCX.A	1+n/2	6/2	6/2	-	7/3	7/3	7/3
PUSHX(.B)	4/2	4/2	4/2	4/3	5(1)/3	5/3	5/3
PUSHX.A	5/2	6/2	6/2	6/3	7(1)/3	7/3	7/3
POPX(.B)	3/2	-	-	-	5/3	5/3	5/3
POPX.A	4/2	-	-	-	7/3	7/3	7/3

(1) 当 Rn=SP 时，增加一个周期

4.5.2.7.2 MSP430X 格式 I（双操作数）指令周期和长度

表 4-18列出了针对所有 MSP430X 扩展格式 I 指令寻址模式的长度和 CPU 周期。

表 4-18. MSP430X 格式 I 指令周期和长度

寻址模式		周期的数量		指令的长度	示例
源	目标	.B/.W	.A	.B/.W/.A	
Rn	Rm[1]	2	2	2	BITX.B R5，R8
	PC	3	3	2	ADDX R9，PC
	X(Rm)	5[2]	7[3]	3	ANDX.A R5，4(R6)
	EDE	5[2]	7[3]	3	XORX R8，EDE
	&EDE	5[2]	7[3]	3	BITX.W R5，&EDE
@Rn	Rm	3	4	2	BITX @R5，R8
	PC	3	4	2	ADDX @R9，PC
	X(Rm)	6[2]	9[3]	3	ANDX.A @R5，4(R6)
	EDE	6[2]	9[3]	3	XORX @R8，EDE
	&EDE	6[2]	9[3]	3	BITX.B @R5，&EDE
@Rn+	Rm	3	4	2	BITX @R5+，R8
	PC	4	5	2	ADDX.A @R9+，PC
	X(Rm)	6[2]	9[3]	3	ANDX @R5+，4(R6)
	EDE	6[2]	9[3]	3	XORX.B @R8+，EDE
	&EDE	6[2]	9[3]	3	BITX @R5+，&EDE
#N	Rm	3	3	3	BITX #20，R8
	PC[4]	4	4	3	ADDX.A #FE000h，PC
	X(Rm)	6[2]	8[3]	4	ANDX #1234，4(R6)
	EDE	6[2]	8[3]	4	XORX #A5A5h，EDE
	&EDE	6[2]	8[3]	4	BITX.B #12，&EDE
X(Rn)	Rm	4	5	3	BITX 2(R5)，R8
	PC[4]	5	6	3	SUBX.A 2(R6)，PC
	X(Rm)	7[2]	10[3]	4	ANDX 4(R7)，4(R6)
	EDE	7[2]	10[3]	4	XORX.B 2(R6)，EDE
	&EDE	7[2]	10[3]	4	BITX 8(SP)，&EDE
EDE	Rm	4	5	3	BITX.B EDE，R8
	PC[4]	5	6	3	ADDX.A EDE，PC
	X(Rm)	7[2]	10[3]	4	ANDX EDE，4(R6)
	EDE	7[2]	10[3]	4	ANDX EDE，TONI
	&TONI	7[2]	10[3]	4	BITX EDE，&TONI
&EDE	Rm	4	5	3	BITX &EDE，R8
	PC[4]	5	6	3	ADDX.A &EDE，PC
	X(Rm)	7[2]	10[3]	4	ANDX.B &EDE，4(R6)
	TONI	7[2]	10[3]	4	XORX &EDE，TONI
	&TONI	7[2]	10[3]	4	BITX &EDE，&TONI

(1) 重复指令要求 n+1 个周期，其中 n 是指令被执行的次数。
(2) 对于 MOV，BIT，和 CMP 指令，周期数量减 1。
(3) 对于 MOV，BIT，和 CMP 指令，周期数量减 2。
(4) 对于 MOV，ADD，和 SUB 指令，周期数量减 1。

4.5.2.7.3 MSP430X 寻址指令周期和长度

表 4-19列出了针对 MSP430 地址指令的所有寻址模式的长度和 CPU 周期。

表 4-19. 寻址指令周期和长度

寻址模式		执行时间（MCLK 周期）		指令长度（字）		示例
源	目标	MOVA BRA	CMPA ADDA SUBA	MOVA	CMPA ADDA SUBA	
Rn	Rn	1	1	1	1	CMPA R5,R8
	PC	2	2	1	1	SUBA R9,PC
	x(Rm)	4	-	2	-	MOVA R5,4(R6)
	EDE	4	-	2	-	MOVA R8,EDE
	&EDE	4	-	2	-	MOVA R5,&EDE
@Rn	Rm	3	–	1	-	MOVA @R5,R8
	PC	3	–	1	-	MOVA @R9,PC
@Rn+	Rm	3	–	1	-	MOVA @R5+,R8
	PC	3	–	1	-	MOVA @R9+,PC
#N	Rm	2	3	2	2	CMPA #20,R8
	PC	3	3	2	2	SUBA #FE000h,PC
x(Rn)	Rm	4	-	2	-	MOVA 2(R5),R8
	PC	4	-	2	-	MOVA 2(R6),PC
EDE	Rm	4	-	2	-	MOVA EDE,R8
	PC	4	-	2	-	MOVA EDE,PC
&EDE	Rm	4	-	2	-	MOVA &EDE,R8
	PC	4	-	2	-	MOVA &EDE,PC

4.6 指令集说明

表 4-20显示了所有可用指令：

表 **4-20. MSP430X** 的指令映射

	000	040	080	0C0	100	140	180	1C0	200	240	280	2C0	300	340	380	3C0
0xxx	MOVA，CMPA，ADDA，SUBA，RRCM，RRAM，RLAM，RRUM															
10xx	RRC	RRC.B	SWPB		RRA	RRA.B	SXT		PUSH	PUSH.B	CALL		RETI	CALLA		
14xx	PUSHM.A，POPM.A，PUSHM.W，POPM.W															
18xx	针对格式 I 和格式 II 指令的扩展字															
1Cxx																
20xx	JNE/JNZ															
24xx	JEQ/JZ															
28xx	JNC															
2Cxx	JC															
30xx	JN															
34xx	JGE															
38xx	JL															
3Cxx	JMP															
4xxx	MOV，MOV.B															
5xxx	ADD，ADD.B															
6xxx	ADDC，ADDC.B															
7xxx	SUBC，SUBC.B															
8xxx	SUB，SUB.B															
9xxx	CMP，CMP.B															
Axxx	DADD，DADD.B															
Bxxx	BIT，BIT.B															
Cxxx	BIC，BIC.B															
Dxxx	BIS，BIS.B															
Exxx	XOR，XOR.B															
Fxxx	AND，AND.B															

4.6.1 扩展指令二进制说明

详细的 MSP430X 指令二进制说明显示在以下的表中。

指令	指令组				src 或 data.19:16	指令标识符				dst	
	15			12	11 8	7			4	3 0	
MOVA	0	0	0	0	src	0	0	0	0	dst	MOVA @RsrcM Rdst
	0	0	0	0	src	0	0	0	1	dst	MOVA @Rsrc+M Rdst
	0	0	0	0	&abs.19:16	0	0	1	0	dst	MOVA &abs20M Rdst
	&abs.15:0										
	0	0	0	0	src	0	0	1	1	dst	MOVA x(Rsrc)M Rdst
	x.15:0										±15 位索引 x
	0	0	0	0	src	0	1	1	0	&abs.19:16	MOVA RsrcM &abs20
	&abs.15:0										
	0	0	0	0	src	0	1	1	1	dst	MOVA RsrcM X(Rdst)
	x.15:0										±15 位索引 x
	0	0	0	0	imm.19:16	1	0	0	0	dst	MOVA #imm20M Rdst
	imm.15:0										
CMPA	0	0	0	0	imm.19:16	1	0	0	1	dst	CMPA #imm20M Rdst
	imm.15:0										
ADDA	0	0	0	0	imm.19:16	1	0	1	0	dst	ADDA #imm20M Rdst
	imm.15:0										
SUBA	0	0	0	0	imm.19:16	1	0	1	1	dst	SUBA #imm20M Rdst
	imm.15:0										
MOVA	0	0	0	0	src	1	1	0	0	dst	MOVA RsrcM Rdst
CMPA	0	0	0	0	src	1	1	0	1	dst	CMPA RsrcM Rdst
ADDA	0	0	0	0	src	1	1	1	0	dst	ADDA RsrcM Rdst
SUBA	0	0	0	0	src	1	1	1	1	dst	SUBA RsrcM Rdst

指令	指令组				位位置	指令 ID		指令标识符				dst	
	15			12	11 10	9	8	7			4	3 0	
RRCM.A	0	0	0	0	n-1	0	0	0	1	0	0	dst	RRCM.A #nM Rdst
RRAM.A	0	0	0	0	n-1	0	1	0	1	0	0	dst	RRAM.A #nM Rdst
RLAM.A	0	0	0	0	n-1	1	0	0	1	0	0	dst	RLAM.A #nM Rdst
RRUM.A	0	0	0	0	n-1	1	1	0	1	0	0	dst	RRUM.A #nM Rdst
RRCM.W	0	0	0	0	n-1	0	0	0	1	0	1	dst	RRCM.W #nM Rdst
RRAM.W	0	0	0	0	n-1	0	1	0	1	0	1	dst	RRAM.W #nM Rdst
RLAM.W	0	0	0	0	n-1	1	0	0	1	0	1	dst	RLAM.W #nM Rdst
RRUM.W	0	0	0	0	n-1	1	1	0	1	0	1	dst	RRUM.W #nM Rdst

指令	指令标识符												dst				
	15			**12**	**11**			**8**	**7**	**6**	**5**	**4**	**3**			**0**	
RETI	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	
CALLA	0	0	0	1	0	0	1	1	0	1	0	0	dst				CALLA Rdst
	0	0	0	1	0	0	1	1	0	1	0	1	dst				CALLA x(Rdst)
	x.15:0																
	0	0	0	1	0	0	1	1	0	1	1	0	dst				CALLA @Rdst
	0	0	0	1	0	0	1	1	0	1	1	1	dst				CALLA @Rdst+
	0	0	0	1	0	0	1	1	1	0	0	0	&abs.19:16				CALLA &abs20
	&abs.15:0																
	0	0	0	1	0	0	1	1	1	0	0	1	x.19:16				CALLA EDE
	x.15:0																CALLA x(PC)
	0	0	0	1	0	0	1	1	1	0	1	1	imm.19:16				CALLA #imm20
	imm.15:0																
被保留	0	0	0	1	0	0	1	1	1	0	1	0	x	x	x	x	
被保留	0	0	0	1	0	0	1	1	1	1	x	x	x	x	x	x	
PUSHM.A	0	0	0	1	0	1	0	0	n-1				dst				PUSHM.A #nM Rdst
PUSHM.W	0	0	0	1	0	1	0	1	n-1				dst				PUSHM.W #nM Rdst
POPM.A	0	0	0	1	0	1	1	0	n-1				dst-n+1				POPM.A #nM Rdst
POPM.W	0	0	0	1	0	1	1	1	n-1				dst-n+1				POPM.W #nM Rdst

4.6.2 MSP430 指令

在下面的部分中对 MSP430 指令进行了说明。

MSP430X 扩展指令请参阅4.6.3 节，而MSP430X 寻址指令请参阅4.6.4 节。

4.6.2.1 ADC

*** ADC[.W]**	将进位增加到目的
*** ADC.B**	将进位增加到目的
句法	`ADC dst`或 `ADC.W dst` `ADC.B dst`
运行	dst+C→dst
仿真	`ADDC #0M dst` `ADDC.B #0M dst`
描述	进位位 (C) 被增加到目的操作数。 目的操作数的之前内容丢失。
状态位	N： 如果结果为负则置位，如果为正则复位。 Z： 如果结果为零则置位，否则复位 C： 如果 dst 被从 0FFFFh 递增至 0000，则置位，否则复位。 如果 dst 被从 0FFh 递增至 00，则置位，否则复位。 V： 如果一个算术溢出发生，则置位，否则复位
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	R13 指向的 16 位计数器被添加到 R12 指向的一个 32 位计数器内。

```
ADD @R13,0(R12) ; Add LSDsADC 2(R12) ; Add carry to MSD
```

示例	R13 指向的 8 位计数器被添加到 R12 指向的一个 16 位计数器内。

```
ADD.B @R13,0(R12) ; Add LSDsADC.B 1(R12) ; Add carry to MSD
```

4.6.2.2 ADD

ADD[.W]	将源字加入至目的字
ADD.B	将源字节加至目的字节
句法	`ADD srcMdst` 或`ADD.W srcMdst` `ADD.B srcMdst`
运行	src+dst→dst
描述	源操作数被添加到目的操作数。 目的操作数之前的内容丢失。
状态位	N： 如果结果为负 (MSB=1)，则置位，如果为正 (MSB=0)，则复位 Z： 如果结果为零，则置位，否则复位 C： 如果有一个来自结果的 MSB 的进位，则置位，否则复位 V： 如果两个正操作数的结果为负，或者如果两个负数的结果为正，则置位，否则复位
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	位于低 64KB 内的 16 位计数器 CNTR 加 10。

```
ADD.W #10,&CNTR ; Add 10 to 16-bit counter
```

示例 由 R5指向的表格字（R5 内的 20 位地址）被加入到 R6。 在一个进位上执行跳转到标签 TONI。

```
ADD.W @R5,R6 ; Add table word to R6. R6.19:16 = 0JC TONI ; Jump if carry... ; No carry
```

示例 R5（20 位地址）指向的一个表格字节被加入到 R6。 如果没有进位发生，执行到标签 TONI 的跳转。 表格指针自动加 1。R6.19:8=0

```
ADD.B @R5+,R6 ; Add byte to R6. R5 + 1. R6: 000xxhJNC TONI ; Jump if no carry... ;
Carry occurred
```

4.6.2.3 ADDC

ADDC[.W]	将源字和进位加入目的字
ADDC.B	将源字节和进位加入目的字节
句法	`ADDC srcMdst`或 `ADDC.W srcMdst` `ADDC.B srcMdst`
运行	src+dst+C→dst
说明	源操作数和进位位 C 被加入到目的操作数。 目的操作数之前的内容丢失。
状态位	N： 如果结果为负 (MSB=1)，则置位，如果为正 (MSB=0)，则复位 Z： 如果结果为零，则置位，否则复位 C： 如果有一个来自结果的 MSB 的进位，则置位，否则复位 V： 如果两个正操作数的结果为负，或者如果两个负数的结果为正，则置位，否则复位
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	常数值 15 和之前指令的进位被加入到位于低 64KB 内的 16 位计数器 CNTR 内。

```
ADDC.W #15,&CNTR ; Add 15 + C to 16-bit CNTR
```

示例 由 R5指向的一个表格字（20 位地址）和进位 C 被加入 R6。 在一个进位上执行跳转到标签 TONI。 R6.19:16=0

```
ADDC.W @R5,R6 ; Add table word + C to R6JC TONI ; Jump if carry... ; No carry
```

示例 由 R5（20 位地址）指向的表格字节和进位位 C 被加入到 R6。 如果没有进位发生，执行到标签 TONI 的跳转。 表格指针自动加 1。R6.19:8=0

```
ADDC.B @R5+,R6 ; Add table byte + C to R6. R5 + 1JNC TONI ; Jump if no carry... ;
Carry occurred
```

4.6.2.4 与

AND[.W]	源字与目的字的逻辑与 (AND)
AND.B	源字节与目的字节的逻辑 AND
句法	`AND srcMdst` 或 `AND.W srcMdst` `AND.B srcMdst`
Operation	src .and. dst→st
说明	源操作数和目的操作数被逻辑与。 结果被放置在目的操作数中。 源操作数不受影响。
状态位	N： 如果结果为负 (MSB=1)，则置位，如果为正 (MSB=0)，则复位 Z： 如果结果为零，则置位，否则复位 C： 如果结果不为零则置位，否则复位。 C=(.not. Z) V： 复位
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	R5（16 位数据）中置位的位被用作一个针对位于低 64KB 内字 TOM 的掩码(AA55h)。 如果结果为零，一个分支指令被带到标签 TONI。 R5.19:16=0

```
MOV #AA55h,R5 ; Load 16-bit mask to R5AND R5,&TOM ; TOM .and. R5 -
> TOMJZ TONI ; Jump if result 0... ; Result > 0
```

或更短：

```
AND #AA55h,&TOM ; TOM .and. AA55h -> TOMJZ TONI ; Jump if result 0
```

示例	由 R5（20 位地址）指向的一个表格字节被与 R6 逻辑与。 取字节后，R5 增 1。 R6.19:8=0

```
AND.B @R5+,R6 ; AND table byte with R6. R5 + 1
```

4.6.2.5 BIC

BIC[.W]	清零目的字中源字内置位的位
BIC.B	清零目的字节中源字节内置位的位
句法	`BIC srcMdst` 或 `BIC.W srcMdst` `BIC.B srcMdst`
运行	(.not. src) .and. dst→dst
说明	被反转的源操作数和目的操作数被逻辑与。 结果被放置在目的操作数内。 源操作数不受影响。
状态位	N： 不受影响 Z： 不受影响 C： 不受影响 V： 不受影响
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	R5 的位 15:14（16 位数据）被清零。 R5.19:16=0

```
BIC #0C000h,R5 ; Clear R5.19:14 bits
```

示例	由 R5 指向的一个表格字（20 位地址）被用于清零 R7 中的位。 R7.19:16=0

```
BIC.W @R5,R7 ; Clear bits in R7 set in @R5
```

示例	R5 （20 位地址）指向的一个表格字节被用于清零 Port1 中的位。

```
BIC.B @R5,&P1OUT ; Clear I/O port P1 bits set in @R5
```

4.6.2.6 BIS

BIS[.W] 置位在目的字中源字内置位的位
BIS.B 置位在目的字节中源字节内置位的位

句法 `BIS srcMdst` 或 `BIS.W srcMdst`
`BIS.B srcMdst`

运行 src .or. dst→dst

说明 源操作数与目的操作数被逻辑与。 结果被放置在目的操作数内。 源操作数不受影响。

状态位
N： 不受影响
Z： 不受影响
C： 不受影响
V： 不受影响

模式位 OSCOFF，CPUOFF 和 GIE 不受影响。

示例 R5 的位 15 和位 13（16 位数据）被置位为 1。 R5.19:16=0

```
BIS #A000h,R5 ; Set R5 bits
```

示例 R5 指向的一个表格字（20 位地址）被用于清零 R7 中的位。 R7.19:16=0

```
BIS.W @R5,R7 ; Set bits in R7
```

示例 R5（20 位地址）指向的一个表格字节被用来置位 Port1 中的位。 之后 R5 增 1。

```
BIS.B @R5+,&P1OUT ; Set I/O port P1 bits. R5 + 1
```

4.6.2.7 位

BIT[.W]	测试在目的字中源字内置位的位
BIT.B	测试在目的字节中源字节内置位的位
句法	`BIT srcMdst` 或 `BIT.W srcMdst` `BIT.B srcMdst`
运行	src .and. dst
说明	源操作数与目的操作数被逻辑与。 结果只影响 SR 中的状态位。 寄存器模式：寄存器位 Rdst.19:16 (.W) resp.Rdst. 19:8 (.B) 未被清零！
状态位	N： 如果结果为负 (MSB=1)，则置位，如果为正 (MSB=0)，则复位 Z： 如果结果为零，则置位，否则复位 C： 如果结果不为零则置位，否则复位。 C = (.not. Z) V： 复位
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	测试是否 R5 的位 15 和 14（16 位数据）中的一个（或两个）被置位。 如果被置位的话则跳转至标签 TONI。 R5.19:16 未受影响。

```
BIT #C000h,R5 ; Test R5.15:14 bitsJNZ TONI ; At least one bit is set in R5... ; Both
bits are reset
```

示例	R5 指向的一个表格字（20 位地址）被用于测试 R7 中的位。 如果至少一个位被置位，则跳转至标签 TONI。 R7.19:16 未受影响。

```
BIT.W @R5,R7 ; Test bits in R7JC TONI ; At least one bit is set... ; Both are reset
```

示例	由 R5（20 位地址）指向的一个表格字节被用来测试输出 Port1 中的位。 如果没有位被置位，则跳转至标签 TONI。 下一个表格字节被寻址。

```
BIT.B @R5+,&P1OUT ; Test I/O port P1 bits. R5 + 1JNC TONI ; No corresponding bit is
set... ; At least one bit is set
```

4.6.2.8 BR，BRANCH

* BR，BRANCH	到低 64KB 地址空间目的的分支指令
句法	BR dst
运行	dst→PC
仿真	MOV dstMPC
说明	一个无条件分支指令被指向低 64KB 地址空间的任一位置上的地址。 可使用所有源寻址模式。 分支指令是一个字指令。
状态位	状态位不受影响。
示例	给出了针对所有寻址模式的示例。

```
BR #EXEC ; Branch to label EXEC or direct branch (e.g. #0A4h); Core instruction MOV
@PC+,PCBR EXEC ; Branch to the address contained in EXEC; Core instruction MOV
X(PC),PC; Indirect addressBR &EXEC ; Branch to the address contained in absolute ;
address EXEC; Core instruction MOV X(0),PC; Indirect addressBR R5 ; Branch to the
address contained in R5; Core instruction MOV R5,PC; Indirect R5BR @R5 ; Branch to the
address contained in the word ; pointed to by R5.; Core instruction MOV @R5,PC;
Indirect, indirect R5BR @R5+ ; Branch to the address contained in the word pointed ;
to by R5 and increment pointer in R5 afterwards.; The next time-
S/W flow uses R5 pointer-
it can ; alter program execution due to access to; next address in a table pointed to
by R5; Core instruction MOV @R5,PC; Indirect, indirect R5 with autoincrementBR X(R5) ;
Branch to the address contained in the address ; pointed to by R5 + X (e.g. table with
address ; starting at X). X can be an address or a label; Core instruction MOV
X(R5),PC; Indirect, indirect R5 + X
```

4.6.2.9 CALL

CALL	调用一个低 64KB 内的子例程
句法	MM dst
运行	dst→PC 16 位 dst 被评估和存储 SP-2→SP PC→@SP 用到 TOS 的返回地址更新了 PC tmp→PC 将 16 位 dst 存储到 PC 中
说明	从一个低 64KB 内的地址到一个低 64KB 内的子例程地址进行子例程调用。 可使用所有七个源寻址模式。 此调用指令是一个字指令。 使用 RET 指令来完成返回。
状态位	状态位不受影响。 PC.19:16 被清零（低 64KB 内的地址）
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	给出了针对所有寻址模式的示例。

立即模式：调用一个标签 EXEC（低 64KB）上的子例程或者直接调用到地址。

```
CALL #EXEC ; Start address EXECCALL #0AA04h ; Start address 0AA04h
```

符号模式：调用一个包含在地址 EXEC 内的 16 位地址上的子例程。 EXEC 位于地址 (PC+X) 上，其中 X 在 PC+32K 内。

```
CALL EXEC ; Start address at @EXEC. z16(PC)
```

绝对模式：调用一个 16 位地址上的子例程，此地址包含在低 64KB 内的绝对地址 EXEC 内。

```
CALL &EXEC ; Start address at @EXEC
```

寄存器模式：调用一个包含在寄存器 R5.15:0 中的 16 位地址上的子例程。

```
CALL R5 ; Start address at R5
```

间接模式：调用一个 16 位地址上的子例程，此地址包含在由寄存器 R5 指向的字（20 位地址）内。

```
CALL @R5 ; Start address at @R5
```

4.6.2.10 CLR

*** CLR[.W]**	清零目的操作数
*** CLR.B**	清零目的操作数
句法	`CLR dst`或 `CLR.W dst` `CLR.B dst`
运行	0→dst
仿真	`MOV #0M dst` `MOV.B #0M dst`
说明	目的操作数被清零。
状态位	状态位不受影响。
示例	RAM 字 TONI 被清零。

```
CLR TONI ; 0 -> TONI
```

示例 寄存器 R5 被清零。

```
CLR R5
```

示例 RAM 字节 TONI 被清零。

```
CLR.B TONI ; 0 -> TONI
```

4.6.2.11 CLRC

*** CLRC**	清零进位位
句法	`CLRC`
运行	0→C
仿真	`BIC #1M SR`
说明	进位位 (C) 被清零。 清零进位指令是字指令。
状态位	N： 不受影响 Z： 不受影响 C： 被清零 V： 不受影响
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	R13 指向的 16 位十进制计数器被添加到 R12 指向的一个 32 位计数器内。

```
CLRC ; C=0: defines startDADD @R13,0(R12) ; add 16-bit counter to low word of 32-
bit counterDADC 2(R12) ; add carry to high word of 32-bit counter
```

4.6.2.12 CLRN

*** CLRN**	清零负位
句法	`CLRN`
运行	0→N 或 (.NOT.src .AND. dst→dst)
仿真	`BIC #4M SR`
说明	常数 04h 被反转 (0FFFBh) 并且与目的操作数进行逻辑与。 结果被放置在目的操作数内。 清零负位指令为字指令。
状态位	N： 复位为 0 Z： 不受影响 C： 不受影响 V： 不受影响
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	SR 中的负位被清零。 这避免了对子例程调用的负数的特别处理。

```
CLRNCALL SUBR......SUBR JN SUBRET ; If input is negative: do nothing and
return.........SUBRET RET
```

4.6.2.13 CLRZ

*** CLRZ**	清零零位
句法	CLRZ
运行	0→Z 或 (.NOT.src .AND. dst→dst)
仿真	BIC #2M SR
说明	常数 02h 被反转 (0FFFDh) 并且与目的操作数进行逻辑与。 结果被放置在目的操作数内。清零零位指令为字指令。
状态位	N： 不受影响 Z： 复位为 0 C： 不受影响 V： 不受影响
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	SR 中的零位被清零。

```
CLRZ
```

间接，自动增量模式：调用一个包含在由R5 指向的字（20 位地址）中的 16 位地址上的子例程，之后 R5 中的 16 位地址递增 2。下次软件使用 R5 作为一个指针，访问由 R5 指向表中的下一个字地址使得它能够改变程序执行。

```
CALL @R5+ ; Start address at @R5. R5 + 2
```

已索引模式：调用一个位于 16 位地址上的子例程，此地址包含在寄存器 (R5+X) 指向的 20 位地址内，例如，一个开始地址为 X 的表。此地址在低 64KB 内。 X 位于 +32KB 内。

```
CALL X(R5) ; Start address at @(R5+X). z16(R5)
```

4.6.2.14 CMP

CMP[.W]	将源字与目的字相比较
CMP.B	将源字节与目的字节相比较
句法	`CMP srcMdst` 或 `CMP.W srcMdst` `CMP.B srcMdst`
运行	(.not.src)+1+dst 或 dst-src
仿真	`BIC #2MSR`
说明	从目的操作数中减去源操作数。 通过在目的中增加源 + 1 的 1s 补充来完成。 结果只影响 SR 中的状态位。 寄存器模式：寄存器位 Rdst.19:16 (.W) resp.Rdst. 19:8 (.B) 未被清零。
状态位	N： 如果结果为负 (src>dst)，则置位，如果为正则复位 (src=dst) Z： 如果为零 (src=dst) 则置位，否则复位 (src≠dst) C： 如果有来自 MSB 的进位，则置位，否则复位 V： 如果从一个正目的操作数中减去一个负源操作数得到一个负结果，或者如果从一个负目的操作数中减去一个正源操作数得到一个正结果，则置位，否则复位（无溢出）。
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	将字 EDE 与一个 16 位常数 1800h 相比较。 如果 EDE 等于常数则跳转至标签 TONI。 EDE 的地址在 PC+32K 内。

```
CMP #01800h,EDE ; Compare word EDE with 1800hJEQ TONI ; EDE contains 1800h... ; Not
equal
```

示例	(R5+10) 指向的一个表格字与 R7 相比较。 如果 R7 包含一个较低的、带符号的 16 位数，则跳转至标签 TONI。 R7.19:16 未被清零。 源操作数的地址为完全地址范围内的一个 20 位地址。

```
CMP.W 10(R5),R7 ; Compare two signed numbersJL TONI ; R7 < 10(R5)... ; R7 >= 10(R5)
```

示例	由 R5（20 位地址）指向的一个表格字节与输出 Port1 中的值相比较。 如果这两个值相等，则跳转至标签 TONI。 下一个表格字节被寻址。

```
CMP.B @R5+,&P1OUT ; Compare P1 bits with table. R5 + 1JEQ TONI ; Equal contents... ;
Not equal
```

4.6.2.15 DADC

*** DADC[.W]** 将十进制进位增加到目的
*** DADC.B** 将十进制进位增加到目的

句法 `DADC dst` 或 `DADC.W dst`
`DADC.B dst`

运行 dst+C→dst（用十进制）

仿真
```
DADD #0M dst
DADD.B #0M dst
```

说明 进位位 (C) 被用十进制增加到目的操作数。

状态位
- N： 如果 MSB 为 1 则置位
- Z： 如果 dst 为 0 则置位，否则复位
- C： 如果目的从 9999 至 0000 递增则置位，否则复位
  如果目的从 99 至 00 递增，则置位，否则复位
- V： 未定义

模式位 OSCOFF，CPUOFF 和 GIE 不受影响。

示例 包含在 R5 中的四位十进制数被增加到由 R8 指向的一个八位十进制数上。

```
CLRC ; Reset carry; next instruction's start condition is definedDADD R5,0(R8) ; Add
LSDs + CDADC 2(R8) ; Add carry to MSD
```

示例 包含在 R5 中的两位十进制数被增加到由 R8 指向的一个四位十进制数上。

```
CLRC ; Reset carry; next instruction's start condition is definedDADD.B R5,0(R8) ; Add
LSDs + CDADC 1(R8) ; Add carry to MSDs
```

4.6.2.16 DADD

*** DADD[.W]** 增加源字和十进制进位至目的字

*** DADD.B** 增加源字节和十进制进位至目的字节

句法 `DADD srcMdst` 或 `DADD.W srcMdst`

`DADD.B srcMdst`

运行 src+dst+C→dst（用十进制）

说明 源操作数和目的操作数被视为具有正符号的两个 (.B) 或者四个 (.W) 的二进制编码的十进制 (BCD)。 源操作数和进位位 C 被用十进制加入到目的操作数。 源操作数不受影响。 目的操作数之前的内容丢失。 此结果不针对非 BCD 数定义。

状态位
- N： 如果结果的 MSB 为 1（字 > 7999h，字节 > 79h）则置位，如果 MSB 为 0 则复位
- Z： 如果结果为零，则置位，否则复位
- C： 如果 BCD 结果太大（字 > 9999h，字节 > 99h），则置位，否则复位
- V： 未定义

模式位 OSCOFF，CPUOFF 和 GIE 不受影响。

示例 十进制数 10 被增加 16 位 BCD 计数器 DECCNTR。

```
DADD #10h,&DECCNTR ; Add 10 to 4-digit BCD counter
```

示例 包含在 16 位 RAM 地址 BCD 和 BCD+2 中的 8 位 BCD 数被用十进制加入到包含在 R4 和 R5 中的一个 8 位 BCD 数中（BCD+2 和 R5 包含 MSD）。 进位 C 被增加、清零。

```
CLRC ; Clear carryDADD.W &BCD,R4 ; Add LSDs. R4.19:16 = 0DADD.W &BCD+2,R5 ; Add MSDs
with carry. R5.19:16 = 0JC OVERFLOW ; Result >9999,9999: go to error routine... ;
Result ok
```

示例 包含在字 BCD 中的两位 BCD 数（16 位地址）被用十进制增加到包含在 R4 中的一个两位 BCD 数中。 进位 C 也被加入。 R4.19:8=0CLRC；清零 carryDADD.B &BCD，R4；将 BCD 用十进制加入 R4。 R4: 0，00ddh

```
CLRC ; Clear carryDADD.B &BCD,R4 ; Add BCD to R4 decimally.R4: 0,00ddh
```

4.6.2.17 DEC

*** DEC[.W]**	递减目的
*** DEC.B**	递减目的
句法	`DEC dst`或 `DEC.W dst` `DEC.B dst`
运行	dst-1→dst
仿真	`SUB #1M dst` `SUB.B #1M dst`
说明	目的操作数减 1。 原先的内容丢失。
状态位	N： 如果结果为负则置位，如果为正则复位。 Z： 如果 dst 包含 1 则置位，否则复位 C： 如果 dst 包含 0 则置位，否则复位 V： 如果一个算术溢出发生，则置位，否则复位。 如果目的的初始值为 08000h 则置位，否则复位。 如果目的的初始值为 080h 则置位，否则复位。
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	R10 减 1。

```
DEC R10 ; Decrement R10; Move a block of 255 bytes from memory location starting with
EDE to ; memory location starting with TONI. Tables should not overlap: start of ;
destination address TONI must not be within the range EDE to EDE+0FEhMOV #EDE,R6MOV
#510,R10L$1 MOV @R6+,TONI-EDE-1(R6)DEC R10JNZ L$1
```

不要上面具有图 4-35中所显示的交迭的例程来传送表格。

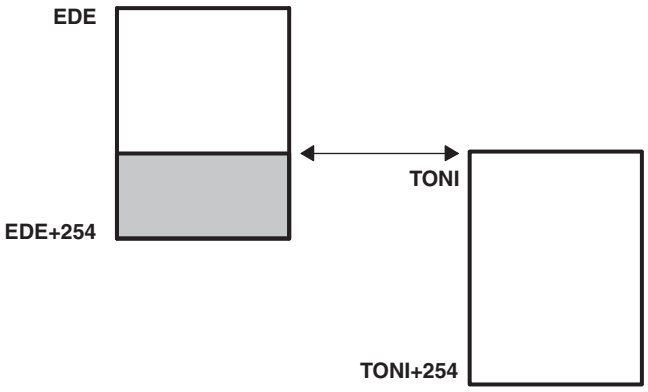


图 **4-35.** 递减交迭

4.6.2.18 DECD

*** DECD[.W]** 双递减目的
*** DECD.B** 双递减目的
句法 DECD dst或 DECD.W dst
DECD.B dst
运行 dst-2→dst
仿真 SUB #2M dst
SUB.B #2M dst
说明 目的操作数递减 2。 原先的内容丢失。
状态位 N： 如果结果为负则置位，如果为正则复位。
Z： 如果 dst 包含 2 则置位，否则复位
C： 如果 dst 包含 0 则复位，否则置位
V： 如果一个算术溢出发生，则置位，否则复位。
如果目的初始值为 08001 或 08000h 则置位，否则复位
如果目的的初始值为 081 或 080h 则置位，否则复位。
模式位 OSCOFF，CPUOFF 和 GIE 不受影响。
示例 R10 递减 2.

```
DECD R10 ; Decrement R10 by two; Move a block of 255 bytes from memory location
starting with EDE to ; memory location starting with TONI. ; Tables should not
overlap: start of destination address TONI must not ; be within the range EDE to
EDE+0FEhMOV #EDE,R6MOV #255,R10L$1 MOV.B @R6+,TONI-EDE-2(R6)DECD R10JNZ L$1
```

示例 位于 LEO 中的存储器递减 2。

```
DECD.B LEO ; Decrement MEM(LEO)
```

将状态字节 STATUS 递减 2

```
DECD.B STATUS
```

4.6.2.19 DINT

*** DINT**	禁用（通用）中断
句法	`DINT`
运行	0→GIE 或 (0FFF7h .AND. SR→SR / .NOT. src .AND. dst→dst)
仿真	`BIC #8M SR`
说明	所有中断被禁用。 常数 08h 被反转并且与 SR 进行逻辑与。 结果被放置在 SR 内。
状态位	状态位不受影响。
模式位	GIE 被复位。 OSCOFF 和 CPUOFF 不受影响。
示例	SR 中的通用中断启用 (GIE) 位被清零来实现一个 32 位计数器的非中断移动。 这就确保任一中断进行移动期间计数器不会被修改。

```
DINT ; All interrupt events using the GIE bit are disabledNOPMOV COUNTHI,R5 ; Copy
counterMOV COUNTLO,R6EINT ; All interrupt events using the GIE bit are enabled
```

注: 禁用中断

如果保护任一代码序列不被中断，在不可中断序列开始前，DINT 应该至少在一个指令上被执行，或者在它之后应该有一个 NOP 指令。

4.6.2.20 EINT

*** EINT**	启用（通用）中断
句法	EINT
运行	1→GIE 或 (0008h .OR. SR→SR / .src .OR. dst→dst)
仿真	BIS #8M SR
说明	所有中断被启用。 常数 08h 和 SR 被逻辑与。 结果被放置在 SR 内。
状态位	状态位不受影响。
模式位	GIE 被复位。 OSCOFF 和 CPUOFF 不受影响。
示例	SR 中的通用中断启用 (GIE) 位被置位。

```
; Interrupt routine of ports P1.2 to P1.7; P1IN is the address of the register where
all port bits are read. ; P1IFG is the address of the register where all interrupt
events are latched.PUSH.B &P1INBIC.B @SP,&P1IFG ; Reset only accepted flagsEINT ;
Preset port 1 interrupt flags stored on stack; other interrupts are allowedBIT
#Mask,@SPJEQ MaskOK ; Flags are present identically to mask: jump......MaskOK BIC
#Mask,@SP......INCD SP ; Housekeeping: inverse to PUSH instruction ; at the start of
interrupt subroutine. Corrects ; the stack pointer.RETI
```

注: **启用中断**

当中断被启用时，启用中断指令 (EINT) 后的指令一直被执行，即使一个中断处理请求在等待中也是如此。

4.6.2.21 INC

*** INC[.W]**	递增目的
*** INC.B**	递增目的
句法	`INC dst`或 `INC.W dst` `INC.B dst`
运行	dst+1→dst
仿真	`ADD #1M dst`
描述	目的操作数被递增 1。 原先的内容丢失。
状态位	N： 如果结果为负则置位，如果为正则复位。 Z： 如果 dst 包含 0FFFFh 则置位，否则复位 如果 dst 包含 0FFh 则置位，否则复位 C： 如果 dst 包含 0FFFFh 则置位，否则复位 如果 dst 包含 0FFh 则置位，否则复位 V： 如果 dst 包含 07FFFh 则置位，否则复位 如果 dst 包含 07Fh 则置位，否则复位
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	一个进程的状态字节，STATUS，被递增。 当它等于 11 时，采用一个到 OVFL 的分支指令。

```
INC.B STATUSCMP.B #11,STATUSJEQ OVFL
```

4.6.2.22 INCD

*** INCD[.W]** 双递增目的

*** INCD.B** 双递增目的

句法 `INCD dst`或 `INCD.W dst`

`INCD.B dst`

运行 dst+2→dst

仿真 `ADD#2M dst`

`ADD.B #2M dst`

说明 目的操作数被递增 2。 原先的内容丢失。

状态位 N： 如果结果为负则置位，如果为正则复位。

Z： 如果 dst 包含 0FFFEh 则置位，否则复位

如果 dst 包含 0FEh 则置位，否则复位

C： 如果 dst 包含 0FFFEh 或 0FFFFh 则置位，否则复位

如果 dst 包含 0FEh 或 0FFh 则置位，否则复位

V： 如果 dst 包含 07FFEh 或 07FFFh 则置位，否则复位

如果 dst 包含 07Eh 或 07Fh 则置位，否则复位

模式位 OSCOFF，CPUOFF 和 GIE 不受影响。

示例 堆栈顶部 (TOS) 的项目在不使用一个寄存器的情况下被删除。

```
.......PUSH R5 ; R5 is the result of a calculation, which is stored; in the system
stack INCD SP ; Remove TOS by double-
increment from stack; Do not use INCD.B, SP is a word-aligned registerRET
```

示例 堆栈顶部的字节递增 2。

```
INCD.B 0(SP) ; Byte on TOS is increment by two
```

4.6.2.23 INV

*** INV[.W]** 反转目的
*** INV.B** 反转目的
句法 `INV dst`或 `INV.W dst`
`INV.B dst`
运行 .not.dst→dst
仿真 `XOR #0FFFFhM dst`
`XOR.B #0FFhM dst`
说明 目的操作数被反转。 原先的内容丢失。
状态位 N： 如果结果为负则置位，如果为正则复位。
Z： 如果 dst 包含 0FFFFh 则置位，否则复位
如果 dst 包含 0FFh 则置位，否则复位
C： 如果结果不为零则置位，否则复位 (=.NOT. 零)
V： 如果初始目的操作数为负则置位，否则复位
模式位 OSCOFF，CPUOFF 和 GIE 不受影响。
示例 R5 的内容被求反（2s 补数）。

```
MOV #00AEh,R5 ; R5 = 000AEhINV R5 ; Invert R5, R5 = 0FF51hINC R5 ; R5 is now negated,
R5 = 0FF52h
```

示例 存储器字节 LEO 的内容被求反。

```
MOV.B #0AEh,LEO ; MEM(LEO) = 0AEhINV.B LEO ; Invert LEO, MEM(LEO) = 051hINC.B LEO ;
MEM(LEO) is negated, MEM(LEO) = 052h
```

4.6.2.24 JC，JHS

JC	如果进位则跳转
JHS	如果高于或同样则跳转（无符号）
句法	JC MM JHS MM
运行	如果 C=1：PC +（2 × 偏移）→PC 如果 C= 0：执行以下指令
说明	SR 中的进位位 C 被测试。 如果它被置位，包含在指令中的带符号 10 位字偏移被乘以 2，符号被扩展，并且被加入到 20 位 PC 中。 这意味着相对于全部存储器范围内的 PC 的 -511 至 +512 字的跳转。 如果 C 被复位，跳转之后的指令被执行。 JC 用于测试进位位 C。 JHS 用于无符号数的比较。
状态位	状态位不受影响。
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	端口 1 引脚 P1IN.1 位的状态定义程序流。

```
BIT.B #2,&P1IN ; Port 1, bit 1 set? Bit -
> CJC Label1 ; Yes, proceed at Label1... ; No, continue
```

示例 如果 R5≥R6（无符号），程序在 Label2 上继续执行。

```
CMP R6,R 5 ; Is R5 >= R6? Info to CJHS Label2 ; Yes, C = 1... ; No, R5 < R6. Continue
```

示例 如果 R5≥12345h（无符号操作数），程序在 Label2 上继续执行。

```
CMPA #12345h,R5 ; Is R5 >= 12345h? Info to CJHS Label2 ; Yes, 12344h < R5 <= F,FFFFh.
C = 1... ; No, R5 < 12345h. Continue
```

4.6.2.25 JEQ，JZ

JEQ	如果相等则跳转
JZ	如果为零则跳转
句法	JEQ MM JZ MM
运行	如果 Z = 1：PC +（2 × 偏移）→ PC 如果 Z = 0：执行下面的指令
说明	SR 中的零位 Z 被测试。 如果它被置位，包含在指令中的带符号 10 位字偏移被乘以 2，符号被扩展，并且被加入到 20 位 PC 中。 这意味着相对于全部存储器范围内的 PC 的 -511 至 +512 字的跳转。 如果 Z 被复位，执行跳转后的指令。 JZ 用于零位 Z 的测试。 JEQ 用于操作数比较。
状态位	状态位不受影响
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	P21N.0 位的状态定义了程序流程。

```
BIT.B #1,&P2IN ; Port 2, bit 0 reset?JZ Label1 ; Yes, proceed at Label1... ; No, set,
continue
```

示例 如果 R5=15000h（20 位数据），程序继续在 Label2 上执行。

```
CMPA #15000h,R5 ; Is R5 = 15000h? Info to SRJEQ Label2 ; Yes, R5 = 15000h. Z = 1... ;
No, R5 not equal 15000h. Continue
```

示例 R7（20 位计数器）被递增。 如果它的内容为零，程序继续在 Label4 上执行。

```
ADDA #1,R7 ; Increment R7JZ Label4 ; Zero reached: Go to Label4... ; R7 not equal 0.
Continue here.
```

4.6.2.26 JGE

JGE	如果大于或者相等则条状（无符号）
句法	JGE MM
运行	如果 (N .xor. V) = 0：PC +（2 × 偏移） → PC 如果 (N .xor. V) = 1：执行下一条指令
说明	SR 中的负位 N 和溢位 V 被测试。 如果两个位都被置位或被复位，包含在指令中的带符号 10 位字偏移被乘以 2，符号被扩展，并且被加入到 20 位 PC 中。 这意味着相对于全部存储器范围内的 PC 的 -511 至 +512 字的跳转。 如果只有一个为被置位，跳转之后的指令被执行。 JGE 被用于带符号操作数的比较：也用于由溢出造成的不正确结果的比较，JGE 指令做出的决定是正确的。 请注意：如果在指令 AND，BIT，RRA，SXTX 和 TST 之后使用的话，JGE 仿真非执行 JP（正则跳转）指令。 这些指令清零 V 位。
状态位	状态位不受影响。
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	如果字节 EDE（低 64KB）包含正数据，则转至 Label1。 软件可运行在完全存储器范围内。

```
TST.B &EDE ; Is EDE positive? V <-
 0JGE Label1 ; Yes, JGE emulates JP... ; No, 80h <= EDE <= FFh
```

示例	如果 R6 的内容大于或者等于由 R7 指向的存储器，程序继续在 Label5 上执行。 带符号数据。 完全存储器范围内的数据和程序。

```
CMP @R7,R6 ; Is R6 >= @R7?JGE Label5 ; Yes, go to Label5... ; No, continue here
```

示例	如果 R5≥12345h（带符号操作数），程序继续在 Label2 上执行。 完全存储器范围内的程序。

```
CMPA #12345h,R5 ; Is R5 >= 12345h?JGE Label2 ; Yes, 12344h < R5 <= 7FFFFh... ; No,
80000h <= R5 < 12345h
```

4.6.2.27 JL

JL	如果少于则跳转（带符号）
句法	JL MM
运行	如果 (N .xor. V) = 1：PC +（2 × 偏移）→ PC 如果 (N .xor. V) = 0：执行下一条指令
说明	SR 中的负位 N 和溢位 V 被测试。 如果只有一个被置位，包含在指令中的带符号 10 位字偏移被乘以 2，符号被扩展，并且被加入到 20 位 PC 中。 这意味着相对于全部存储器范围内的 PC 的 -511 至 +512 字的跳转。 如果位 N 和 V 都被置位或者被复位，执行跳转之后的指令。 JL 被用于带符号操作数的比较：也用于由溢出造成的不正确结果的比较，JL 指令做出的决定是正确的。
状态位	状态位不受影响。
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	如果字节 EDE 包含一个比字节 TONI 更小的、无符号操作数，则继续在 Label1 上。 地址 EDE 在 PC±32K 内。

```
CMP.B &TONI,EDE ; Is EDE < TONIJL Label1 ; Yes... ; No, TONI <= EDE
```

示例	如果 R6 的带符号内容少于由 R7 指向的存储器（20 位地址），程序继续在 Label5 上执行。 完全存储器范围内的数据和程序。

```
CMP @R7,R6 ; Is R6 < @R7?JL Label5 ; Yes, go to Label5... ; No, continue here
```

示例	如果 R5<12345h（带符号操作数），程序继续在 Label2 上执行。 完全存储器范围内的数据和程序。

```
CMPA #12345h,R5 ; Is R5 < 12345h?JL Label2 ; Yes, 80000h =< R5 < 12345h... ; No,
12344h < R5 <= 7FFFFh
```

4.6.2.28 JMP

JMP	无条件跳转
句法	JMP MM
运行	PC+（2 × 偏移）→ PC
说明	包含在指令中的带符号 10 位字偏移被乘以 2，符号被扩展，并且被加入到 20 位 PC 中。这意味着相对于全部存储器范围内的 PC 的 -511 至 +512 字的无条件跳转。 JMP 指令在其相对于 PC 的有限范围内可被用作一个 BR 或者 BRA 指令。
状态位	状态位不受影响
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	字节 STATUS 被置位为 10。 然后进行到标签 MAINLOOP 的跳转。 低 64KB 内的数据，完全存储器范围内的程序。

```
MOV.B #10,&STATUS ; Set STATUS to 10JMP MAINLOOP ; Go to main loop
```

示例	Timer_A3 的中断矢量 TAIV 被读取并用于程序流程。 完全存储器范围内的程序，但是中断处理器一直在低 64KB 内启动。

```
ADD &TAIV,PC ; Add Timer_A interrupt vector to PCRETI ; No Timer_A interrupt
pendingJMP IHCCR1 ; Timer block 1 caused interruptJMP IHCCR2 ; Timer block 2 caused
interruptRETI ; No legal interrupt, return
```

4.6.2.29 JN

JN	如果为负则跳转
句法	JN MM
运行	如果 N = 1：PC +（2 × 偏移）→ PC 如果 N = 0：执行下一个指令
说明	SR 中的负位 N 被测试。 如果它被置位，包含在指令中的带符号 10 位字偏移被乘以 2，符号被扩展，并且被加入到 20 位程序 PC 中。 这意味着相对于全部存储器范围内的 PC 的 -511 至 +512 字的跳转。 如果 N 被复位，跳转之后的指令被执行。
状态位	状态位不受影响。
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	字节 COUNT 被测试。 如果它为负，程序继续在 Label0 上执行。 低 64KB 内的数据，完全存储器范围内的程序。

```
TST.B &COUNT ; Is byte COUNT negative?JN Label0 ; Yes, proceed at Label0... ; COUNT >=
0
```

示例	从 R5 中减去 R6。 如果结果为负，程序继续在 Label2 上执行。 完全存储器范围内的程序。

```
SUB R6,R5 ; R5 - R6 -
> R5JN Label2 ; R5 is negative: R6 > R5 (N = 1)... ; R5 >= 0. Continue here.
```

示例	R7（20 位计数器）被递减。 如果它的内容为负，程序继续在 Label4 上执行。 完全存储器范围内的程序。

```
SUBA #1,R7 ; Decrement R7JN Label4 ; R7 < 0: Go to Label4... ; R7 >= 0. Continue here.
```

4.6.2.30 JNC，JLO

JNC	如果无进位则跳转
JLO	如果低于则跳转（无符号）
句法	JNC MM JLO MM
Operation	如果 C= 0：PC +（2 × 偏移）→PC 如果 C= 1：执行下一指令
说明	SR 中的进位位 C 被测试。 如果它被复位，包含在指令中的带符号 10 位字偏移被乘以 2，符号被扩展，并且被加入到 20 位 PC 中。 这意味着相对于全部存储器范围内的 PC 的 -511 至 +512 字的跳转。 如果 C 被置位，跳转之后的指令被执行。 JNC 用于测试进位位 C。 JLO 用于无符号数的比较。
状态位	状态位不受影响。
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	如果字节 EDE<15，程序继续在 Label2 上执行。 无符号数据。 低 64KB 内的数据，完全存储器范围内的程序。

```
CMP.B #15,&EDE ; Is EDE < 15? Info to CJLO Label2 ; Yes, EDE < 15. C = 0... ; No, EDE
>= 15. Continue
```

示例	字 TONI 被加入 R5。 如果无进位出现，继续在 Label0 上执行。 TONI 的地址在 PC±32K 内。

```
ADD TONI,R5 ; TONI + R5 -> R5. Carry -
> CJNC Label0 ; No carry... ; Carry = 1: continue here
```

4.6.2.31 JNZ，JNE

JNZ	如果非零则跳转
JNE	如果不相等则跳转
句法	JNZ MM JNE MM
运行	如果 Z = 0：PC +（2 × 偏移）→ PC 如果 Z = 1：执行下列指令
说明	SR 中的零位 Z 被测试。 如果它被复位，包含在指令中的带符号 10 位字偏移被乘以 2，符号被扩展，并且被加入到 20 位 PC 中。 这意味着相对于全部存储器范围内的 PC 的 -511 至 +512 字的跳转。 如果 Z 被置位，执行跳转后的指令。 JNZ 用于零位 Z 的测试。 JNE 用于操作数比较。
状态位	状态位不受影响。
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	字节 STATUS 被测试。 如果它不为零，程序继续在 Label3 上执行。 STATUS 的地址在 PC±32K 内。

```
TST.B STATUS ; Is STATUS = 0?JNZ Label3 ; No, proceed at Label3... ; Yes, continue here
```

示例	如果 EDE≠1500，程序继续在 Label2 上执行。 低 64KB 中的数据，完全存储器范围内的程序。

```
CMP #1500,&EDE ; Is EDE = 1500? Info to SRJNE Label2 ; No, EDE not equal 1500.... ;
Yes, R5 = 1500. Continue
```

示例	R7（20 位计数器）被递减。 如果它的内容非零，程序继续在 Label4 上执行。 完全存储器范围内的程序。

```
SUBA #1,R7 ; Decrement R7JNZ Label4 ; Zero not reached: Go to Label4... ; Yes, R7 = 0.
Continue here.
```

4.6.2.32 MOV

MOV[.W]	将源字移动到目的字
MOV.B	将源字节移动到目的字节
句法	`MOV srcMdst` 或 `MOV.W srcMdst` `MOV.B srcMdst`
运行	src→dst
说明	源操作数被复制到目的操作数。 源操作数不受影响。
状态位	N： 不受影响 Z： 不受影响 C： 不受影响 V： 不受影响
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	将一个 16 位常数 1800h 移动到绝对地址字 EDE（低 64KB）

```
MOV #01800h,&EDE ; Move 1800h to EDE
```

示例 表 EDE 的内容（字数据，16 位地址）被复制到表 TOM。 表的长度为 030h 字。 两个表都驻留在低 64KB 内。

```
MOV #EDE,R10 ; Prepare pointer (16-bit address)Loop MOV @R10+,TOM-EDE-
2(R10) ; R10 points to both tables.; R10+2CMP #EDE+60h,R10 ; End of table reached?JLO
Loop ; Not yet... ; Copy completed
```

示例 表 EDE 的内容（字节数据，16 位地址）被复制到表 TOM。 表的长度为 020h 字节。 两个表都驻留在完全存储器范围内，但是必须在 R10±32K 之内。

```
MOVA #EDE,R10 ; Prepare pointer (20-
bit)MOV #20h,R9 ; Prepare counterLoop MOV.B @R10+,TOM-EDE-
1(R10) ; R10 points to both tables.; R10+1DEC R9 ; Decrement counterJNZ Loop ; Not yet
done... ; Copy completed
```

4.6.2.33 NOP

*** NOP**	无操作
句法	NOP
运行	无
仿真	MOV #0M R3
说明	不执行操作。 软件检查期间或者定义的等待时间内，此指令可被用于指令的删除。
状态位	状态位不受影响。

4.6.2.34 POP

*** POP[.W]** 从堆栈中弹出字到目的
*** POP.B** 从堆栈弹出字节到目的

句法
```
POP dst
POP.B dst
```

运行
@SP→temp
SP+2→SP
temp→dst

仿真
```
MOV @SP+M dst 或 MOV.W @SP+M dst
MOV.B @SP+M dst
```

说明 SP 指向的堆栈位置被移动到目的。 之后，SP 被增加 2。

状态位 状态位不受影响。

示例 R7 和 SR 的内容被从堆栈中恢复。

```
POP R7 ; Restore R7POP SR ; Restore status register
```

示例 RAM 字节 LEO 的内容被从堆栈中恢复。

```
POP.B LEO ; The low byte of the stack is moved to LEO.
```

示例 R7 的内容被从堆栈中恢复。

```
POP.B R7 ; The low byte of the stack is moved to R7,; the high byte of R7 is 00h
```

示例 R7 和 SR 指向的存储器的内容被从堆栈中恢复。

```
POP.B 0(R7) ; The low byte of the stack is moved to the; the byte which is pointed to
by R7: Example: R7 = 203h; Mem(R7) = low byte of system stack: Example: R7 = 20Ah;
Mem(R7) = low byte of system stackPOP SR ; Last word on stack moved to the SR
```

注: **系统堆栈指针**

系统 SP 被一直加 2，而与字节后缀无关。

4.6.2.35 PUSH

PUSH[.W]	将一个字保存在堆栈上
PUSH.B	将一个字节保存在堆栈上
句法	`PUSH dstor PUSH.W dst` `PUSH.B dst`
运行	SP-2→SP dst→@SP
说明	20 位 SP 被减 2。 然后操作数被复制到由 SP 寻址的 RAM 字。 一个压入的字节被存储在低字节内；高字节不受影响。
状态位	状态位不受影响。
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	在堆栈上保存两个 16 位寄存器 R9 和 R10

```
PUSH R9 ; Save R9 and R10 XXXXhPUSH R10 ; YYYYh
```

示例	在堆栈上保存两个字节 EDE 和 TONI。 EDE 和 TONI 的地址在 PC±32K 内。

```
PUSH.B EDE ; Save EDE xxXXhPUSH.B TONI ; Save TONI xxYYh
```

4.6.2.36 RET

RET	从子例程返回
句法	RET
运行	@SP→PC.15:0 将 PC 保存至 PC.15:0. PC.19:16←0 SP+2→SP
说明	被一个 CALL 指令压入堆栈的 16 位返回地址（低 64KB）被恢复至 PC。 程序继续在子例程调用之后的地址上执行。 PC.19:16 的四个 MSB 被清零。
状态位	状态位不受影响。 PC.19:16：被清零
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	调用一个低 64KB 内的子例程 SUBR 并且在 CALL 之后返回到低 64KB 内的地址。

```
CALL #SUBR ; Call subroutine starting at SUBR... ; Return by RET to hereSUBR PUSH R14
; Save R14 (16 bit data)... ; Subroutine codePOP R14 ; Restore R14RET ; Return to
lower 64KB
```

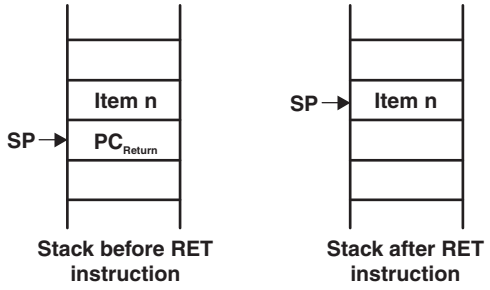


图 **4-36.** 一个 **RET** 指令之后的堆栈

4.6.2.37 RETI

RETI	从中断返回	
句法	RETI	
运行	@SP→SR.15:0 SP+2→SP	用 PC.19:16 恢复保存的 SR
	@SP→PC.15:0 SP+2→SP	恢复保存的 PC.15:0 常规事务
说明	SR 被恢复至中断处理例程的开始值。 这包括 PC.19:16 的四个 MSB。 之后 SP 被加 2。20 位 PC 从 PC.19:16（从与状态位一样的堆栈位置）和 PC.15:0 恢复。20 位 PC 被恢复至中断处理例程的开始值。 当中断被批准后，程序继续在最后一个被执行指令之后的地址上执行。 之后，SP 被加 2。	
状态位	N： 从堆栈恢复 C： 从堆栈恢复 Z： 从堆栈恢复 V： 从堆栈恢复	
模式位	OSCOFF，CPUOFF 和 GIE 从堆栈恢复。	
示例	低 64KB 内的中断处理器 一个 20 位返回地址被存储在堆栈上。	

```
INTRPT PUSHM.A #2,R14 ; Save R14 and R13 (20-
bit data)... ; Interrupt handler codePOPM.A #2,R14 ; Restore R13 and R14 (20-
bit data)RETI ; Return to 20-bit address in full memory range
```

4.6.2.38 RLA

*** RLA[.W]** 算术左旋

*** RLA.B** 算术左旋

句法 `RLA dst`或 `RLA.W dst`

`RLA.B dst`

运行 C←MSB←MSB-1 .... LSB+1←LSB←0

仿真
```
ADD dstMdst
ADD.B dstMdst
```

说明 如图 4-37所示，目的操作数被向左移位一个位置。 MSB 被移入进位位 (C)，而 LSB 被 0 填充。RLA 指令运行为一个带符号的 2 倍乘。

如果在操作被执行前，dst≥04000h 且 dst<0C000h，一个溢出发生。

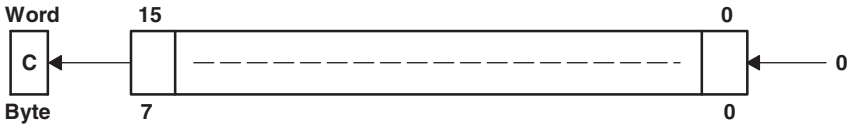


图 **4-37.** 目的操作数-算术左移位

如果在操作被执行前，dst≥ 040h 且 dst< 0C0h，一个溢出发生；此结果有一个经改变的符号。

状态位
- N： 如果结果为负则置位，如果为正则复位。
- Z： 如果结果为零，则置位，否则复位
- C： 从 MSB 载入
- V： 如果一个算术溢出发生：初始值为 04000h≤dst<0C000h，则置位；否则复位
  如果一个算术溢出发生：初始值为 040h≤dst<0C0h，则置位；否则复位

模式位 OSCOFF，CPUOFF 和 GIE 不受影响。

示例 R7 乘以 2。

```
RLA R7 ; Shift left R7 (x 2)
```

示例 R7 的低字节乘以 4。

```
RLA.B R7 ; Shift left low byte of R7 (x 2)RLA.B R7 ; Shift left low byte of R7 (x 4)
```

注: **RLA** 替代

汇编程序并不识别指令：

`RLA @R5+ RLA.B @R5+ RLA(.B) @R5`

它们必须由

`ADD @R5+,-2(R5) ADD.B @R5+,-1(R5) ADD(.B) @R5`

4.6.2.39 RLC 所取代：

*** RLC[.W]**	通过进位左旋
*** RLC.B**	通过进位左旋
句法	`RLC dst`或 `RLC.W dst` `RLC.B dst`
运行	C←MSB←MSB-1 .... LSB+1←LSB←C
仿真	`ADDC dstM dst`
说明	如图 4-38中所示，目的操作数向左移动一个位置。 进位位 (C) 被移入 LSB，而 MSB 被移入进位位 (C)。

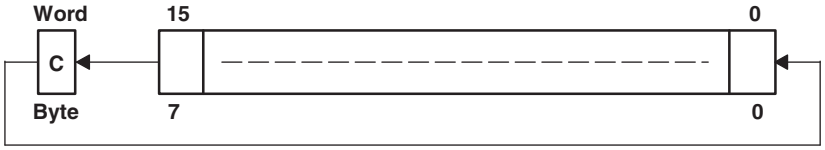


图 **4-38.** 目的操作数-进位左移位

状态位	N： 如果结果为负则置位，如果为正则复位。 Z： 如果结果为零，则置位，否则复位 C： 从 MSB 载入 V： 如果一个算术溢出发生：初始值为 04000h≤dst<0C000h，则置位；否则复位 如果一个算术溢出发生：初始值为 040h≤dst<0C0h，则置位；否则复位
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	R5 被向左移动一个位置。

```
RLC R5 ; (R5 x 2) + C -> R5
```

示例 输入 P1NI.1 信息被移入 R5 的 LSB 中。

```
BIT.B #2,&P1IN ; Information -> CarryRLC R5 ; Carry=P0in.1 -> LSB of R5
```

示例 MEM (LEO) 内容被向左移动一个位置。

```
RLC.B LEO ; Mem(LEO) x 2 + C -> Mem(LEO)
```

注： **RLA** 替代

汇编程序并不识别指令：

```
RLC @R5+ RLC.B @R5+ RLC(.B) @R5
```

它们必须被

```
ADDC @R5+,-2(R5) ADDC.B @R5+,-1(R5) ADDC(.B) @R5
```

4.6.2.40 RRA 所取代：

RRA[.W]	算术右旋目的字
RRA.B	算术右旋目的字节
句法	`RRA.B dst`或 `RRA.W dst`
运行	MSB→MSB→MSB-1 → ... LSB+1→LSB→C
说明	如图 4-39所示，目的操作数被用算术的方法向右移动一个位置。 MSB 保持其值（符号）。 BRA 的运行与带符号的被 2 除等效。MSB 被保持并且被移入 MSB-1。 LSB+1 被移入 LSB。 之前的 LSB 被移入进位位 C。
状态位	N： 如果结果为负 (MSB=1)，则置位，否则 (MSB=0)复位 Z： 如果结果为零，则置位，否则复位 C： 从 LSB 载入 V： 复位
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	R5 中带符号的 16 位数被用算术的方法向右移位一个位置。

```
RRA R5 ; R5/2 -> R5
```

示例	带符号的 RAM 字节 EDE 被用算术的方法向右移动一个位置。

```
RRA.B EDE ; EDE/2 -> EDE
```

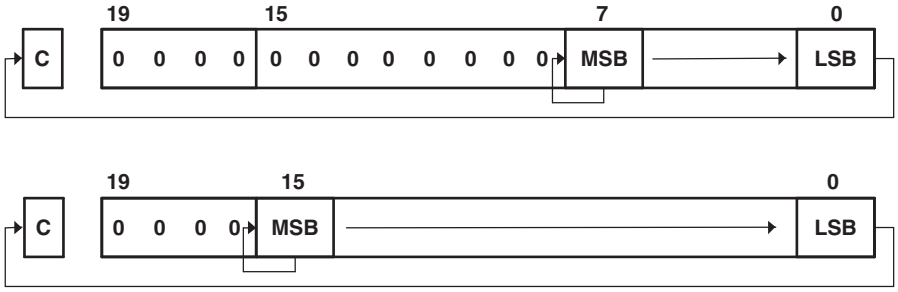


图 **4-39.** 算术右旋 **RRA.B** 和 **RRA.W**

4.6.2.41 RRC

RRC[.W]	通过进位右旋目的字
RRC.B	通过进位目的字节右旋
句法	`RRC dst`与 `RRC.W dst` `RRC.B dst`
运行	C→MSB→MSB–1→ ... LSB+1→LSB→C
说明	如图 4-40所示，目的操作数被向右移动一个位置。 进位位 C 被移入 MSB，而 LSB 被移入进位位 C。
状态位	N： 如果结果为负 (MSB=1)，则置位，否则 (MSB=0)复位 Z： 如果结果为零，则置位，否则复位 C： 从 LSB 载入 V： 复位
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	RAM 字 EDE 被向右移位一个位的位置。 将 1 载入 MSB。

```
SETC ; Prepare carry for MSBRRC EDE ; EDE = EDE >> 1 + 8000h
```

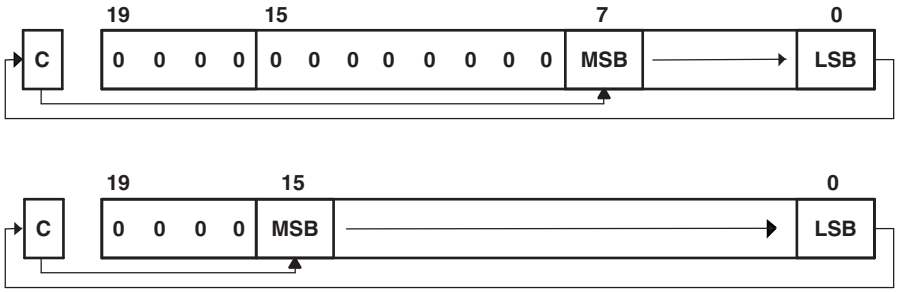


图 **4-40.** 通过进位 **RRC.B** 和 **RRC.W** 右旋

4.6.2.42 SBC

*** SBC[.W]**	将借位（.NOT. 进位）从目的中减去
*** SBC.B**	将借位（.NOT. 进位）从目的中减去
句法	`SBC dst`或 `SBC.W dst` `SBC.B dst`
运行	dst+0FFFFh+C→dst dst+0FFh+C→dst
仿真	`SUBC #0M dst` `SUBC.B #0M dst`
说明	进位位 (C) 被加至目的操作数减一。 目的操作数之前的内容丢失。
状态位	N： 如果结果为负则置位，如果为正则复位 Z： 如果结果为零则置位，否则复位 C： 如果有一个来自结果的 MSB 的进位，则置位，否则复位 如果无借位则置位为 1，如果有借位则复位 V: 如果一个算术溢出发生，则置位，否则复位
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	R13 指向的 16 位计数器被从一个 R12 指向的 32 位计数器内减去。

```
SUB @R13,0(R12) ; Subtract LSDsSBC 2(R12) ; Subtract carry from MSD
```

示例	R13 指向的 8 位计数器被从一个 R12 指向的 16 位计数器内。

```
SUB.B @R13,0(R12) ; Subtract LSDsSBC.B 1(R12) ; Subtract carry from MSD
```

注: **借位执行**

借位被视为一个 .NOT.进位：

借位	进位位
支持	0
否	1

4.6.2.43 SETC

*** SETC**	置位进位位
句法	SETC
运行	1→C
仿真	BIS #1M SR
说明	进位位 (C) 被置位。
状态位	N： 不受影响 Z： 不受影响 C: 设置 V： 不受影响
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	十进制减法的仿真： 用十进制的方法将 R5 从 R6 中减去。 假定 R5=03987h 和 R6=04137h。

```
DSUB ADD #06666h,R5 ; Move content R5 from 0-9 to 6-
0Fh; R5 = 03987h + 06666h = 09FEDhINV R5 ; Invert this (result back to 0-
9); R5 = .NOT. R5 = 06012hSETC ; Prepare carry = 1DADD R5,R6 ; Emulate subtraction by
addition of:; (010000h - R5 - 1); R6 = R6 + R5 + 1; R6 = 0150h
```

4.6.2.44 SETN

*** SETN**	置位负位
句法	SETN
运行	1→N
仿真	BIS #4M SR
说明	复位 (N) 被置位。
状态位	N： 设置
	Z： 不受影响
	C： 不受影响
	V： 不受影响
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。

4.6.2.45 SETZ

*** SETZ**	置位零位
句法	SETZ
运行	1→N
仿真	BIS #2M SR
说明	零位 (Z) 被置位。
状态位	N： 不受影响 Z： 设置 C： 不受影响 V： 不受影响
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。

4.6.2.46 SUB

SUB[.W]	将源字从目的字中减去
SUB.B	将源字节从目的字节中减去
句法	`SUB srcMdst`或 `SUB.W srcMdst` `SUB.B srcMdst`
运行	(.not.src)+1+dst→dst 或 dst-src→dst
说明	从目的操作数中减去源操作数。 通过在目的中增加源 + 1 的 1s 补数来完成。 源操作数不受影响，结果被写入目的操作数。
状态位	N： 如果结果为负 (src>dst)，则置位，如果为正则复位 (src≤dst) Z： 如果为零 (src=dst) 则置位，否则复位 (src≠dst) C： 如果有来自 MSB 的进位，则置位，否则复位 V： 如果从一个正目的操作数中减去一个负源操作数得到一个负结果，或者如果从一个负目的操作数中减去一个正源操作数得到一个正结果，则置位，否则复位（无溢出）。
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	从 RAM 字 EDE 中减去一个 16 位常数 7654h。

```
SUB #7654h,&EDE ; Subtract 7654h from EDE
```

示例	R5 指向的一个表格字（20 位地址）被从 R7 中减去。 之后，如果 R7 包含零，则跳转至标签 TONI。 然后 R5 自动增 2。R7.19:16=0

```
SUB @R5+,R7 ; Subtract table number from R7. R5 + 2JZ TONI ; R7 = @R5 (before
subtraction)... ; R7 <> @R5 (before subtraction)
```

示例	字节 CNT 被从 R12 指向的字节内减去。 CNT 的地址在 PC±32K 内。 R12 指向的地址为全部存储器范围。

```
SUB.B CNT,0(R12) ; Subtract CNT from @R12
```

4.6.2.47 SUBC

SUBC[.W]	从目的字中减去带有进位的源字
SUBC.B	从目的字节中减去带有进位的源字节
句法	SUBC srcMdst或 SUBC.W src,dst SUBC.B srcMdst
运行	(.not.src)+C+dst→dst 或 dst-(src-1)+C→dst
说明	从目的操作数中减去源操作数。 通过在目的中增加源 + 进位的 1s 补数来完成。 源操作数不受影响，结果被写入目的操作数。 用于 32，48，和 64 位操作数。
状态位	N： 如果结果为负 (MSB=1)，则置位，如果为正 (MSB=0)，则复位 Z： 如果结果为零则置位，否则复位 C： 如果有来自 MSB 的进位，则置位，否则复位 V： 如果从一个正目的操作数中减去一个负源操作数得到一个负结果，或者如果从一个负目的操作数中减去一个正源操作数得到一个正结果，则置位，否则复位（无溢出）。
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	一个 16 位常数 7654h 被从带有来自之前指令进位的 R5 中减去。 R5.19:16=0

```
SUBC.W #7654h,R5 ; Subtract 7654h + C from R5
```

示例 由 R5 指向的一个 48 位数（3 个字）（20 位地址）被从由 R7 指向的 RAM 中的一个 48 位计数器内减去。 之后，R5 指向下一个 48 位数。 R7 指向的地址为全部存储器范围。

```
SUB @R5+,0(R7) ; Subtract LSBs. R5 + 2SUBC @R5+,2(R7) ; Subtract MIDs with C. R5 +
2SUBC @R5+,4(R7) ; Subtract MSBs with C. R5 + 2
```

示例 从 R12 指向的字节中减去字节 CNT 。 使用之前指令的进位。 CNT 的地址位于低 64KB 内。

```
SUBC.B &CNT,0(R12) ; Subtract byte CNT from @R12
```

4.6.2.48 SWPB

SWPB	交换字节
句法	`SWPB dst`
运行	dst.15:8↔dst.7:0
说明	操作数的高字节和低字节被交换。 PC.19:16 位在寄存器模式中被清零。
状态位	状态位不受影响
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	交换 RAM 字 EDE 的字节（低 64KB）

```
MOV #1234h,&EDE ; 1234h -> EDESWPB &EDE ; 3412h -> EDE
```

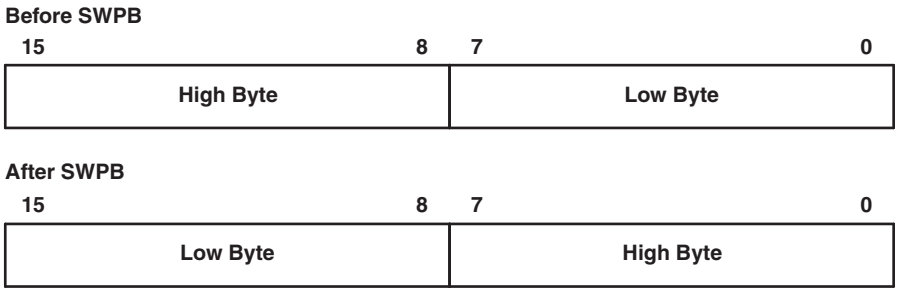


图 **4-41.** 交换存储器中的字节

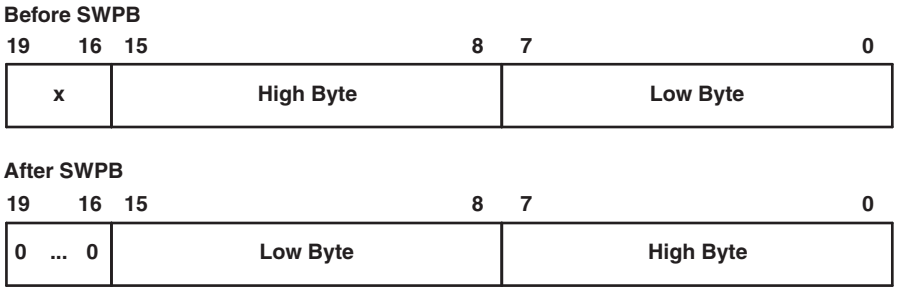


图 **4-42.** 交换寄存器中的字节

4.6.2.49 SXT

SXT	扩展符
句法	SXT dst
运行	dst.7→dst.15:8，dst.7→dst.19:8（寄存器模式）
说明	寄存器模式：操作数低字节的符号被扩展至位 Rdst.19:8 内。 之后 Rdst.7=0: Rdst.19:8=000h 之后 Rdst.7=1: Rdst.19:8=FFFh 其它模式：操作数低字节的符号被扩展至高字节。 之后 dst.7=0: 高字节 = 00h 之后 dst.7=1: 高字节 = FFh
状态位	N： 如果结果为负则置位，否则复位 Z： 如果结果为零则置位，否则复位 C： 如果结果不为零则置位，否则复位 (C=.NOT.Z) V： 复位
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	EDE 内带符号的 8 位数据（低 64KB）是扩展符并且被增加到 R7 内的 16 位带符号数据中。

```
MOV.B &EDE,R5 ; EDE -
> R5. 00XXhSXT R5 ; Sign extend low byte to R5.19:8ADD R5,R7 ; Add signed 16-bit values
```

示例	EDE 内带符号的 8 位数据 (PC+32KB) 是扩展符并且被增加到 R7 内的 20 位数据中。

```
MOV.B EDE,R5 ; EDE -
> R5. 00XXhSXT R5 ; Sign extend low byte to R5.19:8ADDA R5,R7 ; Add signed 20-
bit values
```

4.6.2.50 TST

*** TST[.W]**	测试目的操作数
*** TST.B**	测试目的操作数
句法	`TST dst`或 `TST.W dst` `TST.B dst`
运行	dst+0FFFFh+1 dst+0FFh+1
仿真	`CMP #0M dst` `CMP.B #0M dst`
说明	目的操作数与零相比较。 根据结果置位状态位。 目的操作数不受影响。
状态位	N： 如果目的操作数为负则置位，如果为正则复位 Z： 如果目的操作数包含零则置位，否则复位 C： 设置 V： 复位
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	R7 被测试。 如果它为负，则继续在 R7NEG 上执行；如果为正但又不为零，则继续在 R7POS 上执行。

```
TST R7 ; Test R7JN R7NEG ; R7 is negativeJZ R7ZERO ; R7 is zeroR7POS ..... ; R7 is
positive but not zeroR7NEG ..... ; R7 is negativeR7ZERO ..... ; R7 is zero
```

示例	R7 的低字节被测试。 如果它为负，则继续在 R7NEG 上执行；如果为正但又不为零，则继续在 R7POS 上执行。

```
TST.B R7 ; Test low byte of R7JN R7NEG ; Low byte of R7 is negativeJZ R7ZERO ; Low
byte of R7 is zeroR7POS ..... ; Low byte of R7 is positive but not zeroR7NEG ..... ;
Low byte of R7 is negativeR7ZERO ..... ; Low byte of R7 is zero
```

4.6.2.51 XOR

XOR[.W]	源字与目的字异或操作
XOR.B	源字节与目的字节异或操作
句法	`XOR srcMdst`或 `XOR.W srcMdst` `XOR.B srcMdst`
运行	src .xor. dst→dst
说明	源操作数和目的操作数被异或操作。 结果被放置在目的操作数内。 源操作数不受影响。 目的操作数之前的内容丢失。
状态位	N： 如果结果为负 (MSB=1)，则置位，如果为正 (MSB=0)，则复位 Z： 如果结果为零则置位，否则复位 C： 如果结果不为零则置位，否则复位 (C=.NOT. Z) V： 如果两个操作数在执行前均为负则置位，否则复位
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	将位切换为带有地址字 TONI 信息（位=1）的字 CNTR（16 位数据）。 两个操作数都位于低 64KB 内。

```
XOR &TONI,&CNTR ; Toggle bits in CNTR
```

示例	R5 指向的一个表格字（20 位地址）被用于切换 R6 中的位。 R6.19:16=0

```
XOR @R5,R6 ; Toggle bits in R6
```

示例	R7 中低字节内复位为零的那些位与位于字节 EDE 内的位不同。 R7.19:8=0。EDE 的地址在 PC±32K 内。

```
XOR.B EDE,R7 ; Set different bits to 1 in R7.INV.B R7 ; Invert low byte of R7, high
byte is 0h
```

4.6.3 MSP430X 扩展指令

MSP430X 扩展指令使得 MSP430X CPU 可完全访问其 20 位地址范围。 MSP430X 指令要求一个被称为扩展字的运算代码的附加字。 当前面为扩展字时，所有地址、索引、和立即数有 20 位的值。 在下面的部分中对 MSP430X 扩展指令进行了说明。 对于不需要扩展字的 MSP430X 指令，在指令说明中进行了注释。

MSP430X 指令请参阅4.6.2 节，而 MSP430X 寻址指令请参阅4.6.4 节。

4.6.3.1 ADCX

*** ADCX.A** 将进位增加到目的地址字
*** ADCX.[W]** 将进位增加到目的字节
*** ADCX.B** 将进位增加到目的字节

句法
```
ADCX.A dst
ADCX dst或 ADCX.W dst
ADCX.B dst
```

运行 dst+C→dst

仿真
```
ADDCX.A #0M dst
ADDCX #0M dst
ADDCX.B #0M dst
```

说明 进位位 (C) 被增加到目的操作数。 目的操作数之前的内容丢失。

状态位
- N： 如果结果为负 (MSB=1)，则置位，如果为正 (MSB=0)，则复位
- Z： 如果结果为零则置位，否则复位
- C： 如果有一个来自结果的 MSB 的进位，则置位，否则复位
- V： 如果两个正操作数的结果为负，或者如果两个负数的结果为正，则置位，否则复位

模式位 OSCOFF，CPUOFF 和 GIE 不受影响。

示例 R12 和 R13 指向的 40 位计数器被递增。

```
INCX.A @R12 ; Increment lower 20 bitsADCX.A @R13 ; Add carry to upper 20 bits
```

4.6.3.2 ADDX

ADDX.A 将源地址字加入目的地址字
ADDX.[W] 将源字加入至目的字
ADDX.B 将源字节加至目的字节

句法
```
ADDX.A srcMdst
ADDX srcMdst或 ADDX.W srcMdst
ADDX.B srcMdst
```

运行 src+dst→dst

说明 源操作数被添加到目的操作数。 目的操作数之前的内容丢失。 两个操作数都被放置在完全地址空间内。

状态位
- N： 如果结果为负 (MSB=1)，则置位，如果为正 (MSB=0)，则复位
- Z: 如果结果为零则置位，否则复位
- C: 如果有一个来自结果的 MSB 的进位，则置位，否则复位
- V: 如果两个正操作数的结果为负，或者如果两个负数的结果为正，则置位，否则复位

模式位 OSCOFF，CPUOFF 和 GIE 不受影响。

示例 位于两个字 CNTR (LSB) 和 CNTR+2 (MSB) 内的 20 位指针 CNTR 加 10。

```
ADDX.A #10,CNTR ; Add 10 to 20-bit pointer
```

示例 R5（20 位地址）指向的一个表格字节（16 位）被加入到 R6。 在一个进位上执行跳转到标签 TONI。

```
ADDX.W @R5,R6 ; Add table word to R6JC TONI ; Jump if carry... ; No carry
```

示例 R5（20 位地址）指向的一个表格字节被加入到 R6。 如果没有进位发生，执行到标签 TONI 的跳转。 表格指针自动加 1。

```
ADDX.B @R5+,R6 ; Add table byte to R6. R5 + 1. R6: 000xxhJNC TONI ; Jump if no
carry... ; Carry occurred
```

请注意：在下面两个情况中使用 ADDA 以实现更佳的代码密度和执行性能。

```
ADDX.A Rsrc,RdstADDX.A #imm20,Rdst
```

4.6.3.3 ADDCX

ADDCX.A	将源地址字和进位加入目的地址字
ADDCX.[W]	将源字和进位加入目的字
ADDCX.B	将源字节和进位加入目的字节

句法	`ADDCX.A src,dst`
	`ADDCX srcMdst`或 `ADDCX.W srcMdst`
	`ADDCX.B srcMdst`

运行	src+dst+C→dst
说明	源操作数和进位位 C 被加入到目的操作数。 目的操作数之前的内容丢失。 两个操作数都位于完全地址空间内。
状态位	N： 如果结果为负 (MSB=1)，则置位，如果为正 (MSB=0)，则复位 Z： 如果结果为零则置位，否则复位 C： 如果有一个来自结果的 MSB 的进位，则置位，否则复位 V： 如果两个正操作数的结果为负，或者如果两个负数的结果为正，则置位，否则复位
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	常数值 15 和之前指令的进位被加入到位于两个字内的 20 位计数器 CNTR 内。

```
ADDCX.A #15,&CNTR ; Add 15 + C to 20-bit CNTR
```

示例 由 R5（20 位地址）指向的一个表格字和进位 C 被加入 R6。 在一个进位上执行跳转到标签 TONI。

```
ADDCX.W @R5,R6 ; Add table word + C to R6JC TONI ; Jump if carry... ; No carry
```

示例 由 R5（20 位地址）指向的表格字节和进位位 C 被加入到 R6。 如果没有进位发生，执行到标签 TONI 的跳转。 表格指针自动加 1。

```
ADDCX.B @R5+,R6 ; Add table byte + C to R6. R5 + 1JNC TONI ; Jump if no carry... ;
Carry occurred
```

4.6.3.4 ANDX

ANDX.A 源地址字与目的地址字的逻辑与
ANDX.[W] 源字与目的字的逻辑与
ANDX.B 源字节与目的字节的逻辑与

句法
`ANDX.A srcM dst`
`ANDX srcM dst`或 `ANDX.W srcM dst`
`ANDX.B srcM dst`

运行 src .and. dst→dst

说明 源操作数与目的操作数被逻辑与。 结果被放置在目的操作数内。 源操作数不受影响。 两个操作数都位于完全地址空间内。

状态位
N： 如果结果为负 (MSB=1)，则置位，如果为正 (MSB=0)，则复位
Z： 如果结果为零则置位，否则复位
C： 如果结果不为零则置位，否则复位。 C=(.not. Z)
V： 复位

模式位 OSCOFF，CPUOFF 和 GIE 不受影响。

示例 R5（20 位数据）中置位的位被用作一个针对两个字内地址字 TOM 的掩码 (AAA55h)。 如果结果为零，一个分支指令被指向标签 TONI。

```
MOVA #AAA55h,R5 ; Load 20-bit mask to R5ANDX.A R5,TOM ; TOM .and. R5 -
> TOMJZ TONI ; Jump if result 0... ; Result > 0
```

或更短：

```
ANDX.A #AAA55h,TOM ; TOM .and. AAA55h -> TOMJZ TONI ; Jump if result 0
```

示例 由 R5（20 位地址）指向的一个表格字节被与 R6 逻辑与。 R6.19:8=0。 表格指针自动加 1。

```
ANDX.B @R5+,R6 ; AND table byte with R6. R5 + 1
```

4.6.3.5 BICX

BICX.A	清零目的地址字中源地址字内置位的位
BICX.[W]	清零目的字中源字内置位的位
BICX.B	清零目的字节中源字节内置位的位

句法	`BICX.A srcMdst`
	`BICX srcMdst` 或 `BICX.W srcMdst`
	`BICX.B srcMdst`

运行	(.not. src) .and. dst→dst
说明	被反转的源操作数和目的操作数被逻辑与。 结果被放置在目的操作数内。 源操作数不受影响。 两个操作数都位于完全地址空间内。
状态位	N： 不受影响 Z： 不受影响 C： 不受影响 V： 不受影响
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	R5（20 位数据）的位 19:15 被清零。

```
BICX.A #0F8000h,R5 ; Clear R5.19:15 bits
```

示例 由 R5（20 位地址）指向的一个表格字被用于清零 R7 中的位。 R7.19:16=0

```
BICX.W @R5,R7 ; Clear bits in R7
```

示例 由 R5（20 位地址）指向的一个表格字节被用来清零输出 Port1 中的位。

```
BICX.B @R5,&P1OUT ; Clear I/O port P1 bits
```

4.6.3.6 BISX

BISX.A	置位在目的地址字中源地址字内置位的位
BISX.[W]	置位在目的字中源字内置位的位
BISX.B	置位在目的字节中源字节内置位的位

句法	`BISX.A srcMdst`
	`BISX srcMdst` 或 `BISX.W srcMdst`
	`BISX.B srcMdst`

运行	src .or. dst→dst
说明	源操作数与目的操作数被逻辑与。 结果被放置在目的操作数内。 源操作数不受影响。 两个操作数都位于完全地址空间内。
状态位	N： 不受影响 Z： 不受影响 C： 不受影响 V： 不受影响
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	R5（20 位数据）的位 16 和位 15 被置位为 1。

```
BISX.A #018000h,R5 ; Set R5.16:15 bits
```

示例 R5（20 位地址）指向的一个表格字被用于置位 R7 中的位。

```
BISX.W @R5,R7 ; Set bits in R7
```

示例 由 R5（20 位地址）指向的一个表格字节被用来置位输出 Port1 中的位。

```
BISX.B @R5,&P1OUT ; Set I/O port P1 bits
```

4.6.3.7 BITX

BITX.A	测试在目的地址字中源地址字内置位的位
BITX.[W]	测试在目的字中源字内置位的位
BITX.B	测试在目的字节中源字节内置位的位

句法	`BITX.A srcMdst`
	`BITX srcMdst`或 `BITX.W srcMdst`
	`BITX.B srcMdst`

运行	src .and. dst→dst
说明	源操作数与目的操作数被逻辑与。 结果只影响状态位。 两个操作数都位于完全地址空间内。
状态位	N： 如果结果为负 (MSB=1)，则置位，如果为正 (MSB=0)，则复位 Z： 如果结果为零则置位，否则复位 C： 如果结果不为零则置位，否则复位。 C=(.not. Z) V： 复位
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	测试是否 R5（20 位数据）的为 16 和 15 被置位。 如果被置位则跳转至标签。

```
BITX.A #018000h,R5 ; Test R5.16:15 bitsJNZ TONI ; At least one bit is set... ; Both
are reset
```

示例	R5（20 位地址）指向的一个表格字被用于测试 R7 中的位。 如果至少一个位被置位，则跳转至标签 TONI。

```
BITX.W @R5,R7 ; Test bits in R7: C = .not.ZJC TONI ; At least one is set... ; Both are
reset
```

示例	由 R5（20 位地址）指向的一个表格字节被用来测试输入 Port1 中的位。 如果没有位被置位，则跳转至标签 TONI。 下一个表格字节被寻址。

```
BITX.B @R5+,&P1IN ; Test input P1 bits. R5 + 1JNC TONI ; No corresponding input bit is
set... ; At least one bit is set
```

4.6.3.8 CLRX

*** CLRX.A** 清零目的地址字
*** CLRX.[W]** 清零目的字
*** CLRX.B** 清零目的字节

句法	`CLRX.A dst`
	`CLRX dst`或 `CLRX.W dst`
	`CLRX.B dst`

运行	0→dst

仿真	`MOVX.A #0M,dst`
	`MOVX #0M,dst`
	`MOVX.B #0M,dst`

说明	目的操作数被清零。
状态位	状态位不受影响。
示例	RAM 地址字 TONI 被清零。

```
CLRX.A TONI ; 0 -> TONI
```

4.6.3.9 CMPX

CMPX.A	将源地址字与目的地址字相比较
CMPX.[W]	将源字与目的字相比较
CMPX.B	将源字节与目的字节相比较

句法	`CMPX.A srcMdst`
	`CMPX srcMdst`或 `CMPX.W srcMdst`
	`CMPX.B srcMdst`

运行	(.not. src)+1+dst 或 dst-src
说明	通过将源 + 1 的 1s 补数加入目的，源操作数被从目的操作数中刨除。 结果只影响状态位。 两个操作数都位于完全地址空间内。
状态位	N： 如果结果为负 (src>dst)，则置位，如果为正则复位 (src≤dst) Z： 如果为零 (src=dst) 则置位，否则复位 (src≠dst) C： 如果有来自 MSB 的进位，则置位，否则复位 V： 如果从一个正目的操作数中减去一个负源操作数得到一个负结果，或者如果从一个负目的操作数中减去一个正源操作数得到一个正结果，则置位，否则复位（无溢出）。
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	将 EDE 与一个 20 位常数 18000h 相比较。 如果 EDE 等于常数则跳转至标签 TONI。

```
CMPX.A #018000h,EDE ; Compare EDE with 18000hJEQ TONI ; EDE contains 18000h... ; Not
equal
```

示例 R5（20 位地址）指向的一个表格字与 R7 相比较。 如果 R7 包含一个较低的、带符号的 16 位数，则跳转至标签 TONI。

```
CMPX.W @R5,R7 ; Compare two signed numbersJL TONI ; R7 < @R5... ; R7 >= @R5
```

示例 由 R5（20 位地址）指向的一个表格字节与输入到I/O Port1 中的值相比较。 如果这两个值相等，则跳转至标签 TONI。 下一个表格字节被寻址。

```
CMPX.B @R5+,&P1IN ; Compare P1 bits with table. R5 + 1JEQ TONI ; Equal contents... ;
Not equal
```

请注意：在下面两个情况中使用 CMPA 以实现更佳的代码密度和执行性能。

```
CMPA Rsrc,RdstCMPA #imm20,Rdst
```

4.6.3.10 DADCX

*** DADCX.A** 将十进制进位增加到目的地址字
*** DADCX.[W]** 将十进制进位增加到目的字
*** DADCX.B** 将十进制进位增加到目的字节

句法
```
DADCX.A dst
DADCX dst或 DADCX.W dst
DADCX.B dst
```

运行 dst+C→dst（用十进制）

仿真
```
DADDX.A #0,dst
DADDX #0M dst
DADDX.B #0M dst
```

说明 进位位 (C) 被用十进制增加到目的操作数。

状态位
- N： 如果结果的 MSB 为 1（地址字 > 79999h，字 > 7999h，字节 > 79h）则置位，如果 MSB 为 0 则复位
- Z： 如果结果为零则置位，否则复位
- C： 如果 BCD 结果太大（地址字 > 99999h，字 > 9999h，字节 > 99h）则置位，否则复位
- V： 未定义

模式位 OSCOFF，CPUOFF 和 GIE 不受影响。

示例 R12 和 R13 指向的 40 位计数器被用十进制方法递增。

```
DADDX.A #1,0(R12) ; Increment lower 20 bitsDADCX.A 0(R13) ; Add carry to upper 20 bits
```

4.6.3.11 DADDX

DADDX.A 将源地址字和进位用十进制方法加入目的地址字

DADDX.[W] 将源字和进位用十进制的方法加入目的字

DADDX.B 将源字节和进位用十进制的方法加入目的字节

句法 `DADDX.A srcM dst`

`DADDX srcM dst`或 `DADDX.W srcM dst`

`DADDX.B src,dst`

运行 src+dst+C→dst（用十进制）

说明 源操作数和目的操作数被视为具有正符号的两个 (.B)，四个 (.W) 或五个 (.A) 的二进制编码的十进制 (BCD) 数。 源操作数和进位位 C 被用十进制加入到目的操作数。 源操作数不受影响。 目的操作数之前的内容丢失。 此结果不针对非 BCD 数定义。 两个操作数都位于完全地址空间内。

状态位
- N： 如果结果的 MSB 为 1（地址字 > 79999h，字 > 7999h，字节 > 79h）则置位，如果 MSB 为 0 则复位
- Z： 如果结果为零则置位，否则复位
- C： 如果 BCD 结果太大（地址字 > 99999h，字 > 9999h，字节 > 99h）则置位，否则复位
- V： 未定义

模式位 OSCOFF，CPUOFF 和 GIE 不受影响。

示例 十进制 10 被加入到位于两个字内的 20 位 BCD 计数器 DECCNTR 内。

```
DADDX.A #10h,&DECCNTR ; Add 10 to 20-bit BCD counter
```

示例 包含在 20 位地址 BCD 和 BCD+2 中的 8 位 BCD 数被用十进制加入到包含在 R4 和 R5 中的一个 8 位 BCD 数中（BCD+2 和 R5 包含 MSD）。

```
CLRC ; Clear carryDADDX.W BCD,R4 ; Add LSDsDADDX.W BCD+2,R5 ; Add MSDs with carryJC
OVERFLOW ; Result >99999999: go to error routine... ; Result ok
```

示例 包含在 20 位地址 BCD 中的两位 BCD 数被用十进制增加到包含在 R4 中的一个两位 BCD 数中。

```
CLRC ; Clear carryDADDX.B BCD,R4 ; Add BCD to R4 decimally.; R4: 000ddh
```

4.6.3.12 DECX

*** DECX.A**	递减目的地址字
*** DECX.[W]**	递减目的字
*** DECX.B**	递减目的字节

句法	`DECX.A dst`
	`DECX dst`或 `DECX.W dst`
	`DECX.B dst`

运行	dst-1→dst

仿真	`SUBX.A #1M,dst`
	`SUBX #1M,dst`
	`SUBX.B #1M,dst`

说明	目的操作数减 1。 原先的内容丢失。
状态位	N： 如果结果为负则置位，如果为正则复位 Z： 如果 dst 包含 1 则置位，否则复位 C 如果 dst 包含 0 则置位，否则复位 V： 如果一个算术溢出发生则置位，否则复位。
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	RAM 地址字 TONI 减 1。

```
DECX.A TONI ; Decrement TONI
```

4.6.3.13 DECDX

*** DECDX.A**	双递减目的地址字
*** DECDX.[W]**	双递减目的字
*** DECDX.B**	双递减目的字节

句法	`DECDX.A dst`
	`DECDX dst`或 `DECDX.W dst`
	`DECDX.B dst`

运行	dst-2→dst

仿真	`SUBX.A #2M,dst`
	`SUBX #2M,dst`
	`SUBX.B #2M,dst`

说明	目的操作数递减 2。 原先的内容丢失。
状态位	N： 如果结果为负则置位，如果为正则复位 Z： 如果 dst 包含 2 则置位，否则复位 C： 如果 dst 包含 0 则复位，否则置位 V： 如果一个算术溢出发生则置位，否则复位。
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	RAM 地址字 TONI 递减 2。

```
DECDX.A TONI ; Decrement TONI
```

4.6.3.14 INCX

*** INCX.A** 递增目的地址字
*** INCX.[W]** 递增目的字
*** INCX.B** 递增目的字节

句法 `INCX.A dst`
`INCX dst`或 `INCX.W dst`
`INCX.B dst`

运行 dst+1→dst

仿真 `ADDX.A #1M dst`
`ADDX #1M dst`
`ADDX.B #1M dst`

说明 目的操作数被递增 1。 原先的内容丢失。

状态位
N： 如果结果为负则置位，如果为正则复位
Z： 如果 dst 包含 0FFFFFh 则置位，否则复位
如果 dst 包含 0FFFFh 则置位，否则复位
如果 dst 包含 0FFh 则置位，否则复位
C： 如果 dst 包含 0FFFFFh 则置位，否则复位
如果 dst 包含 0FFFFh 则置位，否则复位
如果 dst 包含 0FFh 则置位，否则复位
V： 如果 dst 包含 07FFFh 则置位，否则复位
如果 dst 包含 07FFFh 则置位，否则复位
如果 dst 包含 07Fh 则置位，否则复位

模式位 OSCOFF，CPUOFF 和 GIE 不受影响。

示例 RAM 地址字 TONI 加 1。

```
INCX.A TONI ; Increment TONI (20-bits)
```

4.6.3.15 INCDX

*** INCDX.A**	双递增目的地址字
*** INCDX.[W]**	双递增目的字
*** INCDX.B**	双递增目的字节

句法	`INCDX.A dst`
	`INCDX dst`或 `INCDX.W dst`
	`INCDX.B dst`

运行	dst+2→dst

仿真	`ADDX.A #2M dst`
	`ADDX #2M dst`
	`ADDX.B #2M dst`

说明	目的操作数被递增 2。 原先的内容丢失。
状态位	N： 如果结果为负则置位，如果为正则复位 Z： 如果 dst 包含 0FFFFEh 则置位，否则复位 如果 dst 包含 0FFFEh 则置位，否则复位 如果 dst 包含 0FEh 则置位，否则复位 C： 如果 dst 包含 0FFFFEh 或 0FFFFFh 则置位，否则复位 如果 dst 包含 0FFFEh 或 0FFFFh 则置位，否则复位 如果 dst 包含 0FEh 或 0FFh 则置位，否则复位 V： 如果 dst 包含 07FFFEh 或 07FFFFh 则置位，否则复位 如果 dst 包含 07FFEh 或 07FFFh 则置位，否则复位 如果 dst 包含 07Eh 或 07Fh 则置位，否则复位
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	RAM 字节 LEO 递增 2；PC 指向上部存储器。

```
INCDX.B LEO ; Increment LEO by two
```

4.6.3.16 INVX

*** INVX.A** 反转目的操作数
*** INVX.[W]** 反转目的操作数
*** INVX.B** 反转目的操作数

句法
```
INVX.A dst
INVX dst或 INVX.W dst
INVX.B dst
```

运行 .not.dst→dst

仿真
```
XORX.A #0FFFFFhM dst
XORX #0FFFFhM dst
XORX.B #0FFhM dst
```

说明 目的操作数被反转。 原先的内容丢失。

状态位
- N： 如果结果为负则置位，如果为正则复位
- Z： 如果 dst 包含 0FFFFFh 则置位，否则复位
  如果 dst 包含 0FFFFh 则置位，否则复位
  如果 dst 包含 0FFh 则置位，否则复位
- C： 如果结果不为零则置位，否则复位 (=.NOT. 零位)
- V： 如果初始目的操作数为负则置位，否则复位

模式位 OSCOFF，CPUOFF 和 GIE 不受影响。

示例 R5 的 20 位内容被求反（2s 补数）。

```
INVX.A R5 ; Invert R5INCX.A R5 ; R5 is now negated
```

示例 存储器字节 LEO 的内容被求反。 PC 正指向上部存储器。

```
INVX.B LEO ; Invert LEOINCX.B LEO ; MEM(LEO) is negated
```

4.6.3.17 MOVX

MOVX.A	将源地址字移至目的地址字
MOVX.[W]	将源字移动到目的字
MOVX.B	将源字节移动到目的字节

句法	`MOVX.A src,dst`
	`MOVX srcMdst` 或 `MOVX.W srcMdst`
	`MOVX.B srcMdst`

运行	src→dst
描述	源操作数被复制到目的操作数。 源操作数不受影响。 两个操作数都位于完全地址空间内。
状态位	N： 不受影响 Z： 不受影响 C： 不受影响 V： 不受影响
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	将一个 20 位常数 18000h 移动到绝对地址字 EDE

```
MOVX.A #018000h,&EDE ; Move 18000h to EDE
```

示例 表 EDE 的内容（字数据，20 位地址）被复制到表 TOM。 表的长度为 030h 字。

```
MOVA #EDE,R10 ; Prepare pointer (20-bit address)Loop MOVX.W @R10+,TOM-EDE-
2(R10) ; R10 points to both tables.; R10+2CMPA #EDE+60h,R10 ; End of table reached?JLO
Loop ; Not yet... ; Copy completed
```

示例 表 EDE 的内容（字节数据，20 位地址）被复制到表 TOM。 表的长度为 020h 字节。

```
MOVA #EDE,R10 ; Prepare pointer (20-
bit)MOV #20h,R9 ; Prepare counterLoop MOVX.W @R10+,TOM-EDE-
2(R10) ; R10 points to both tables.; R10+1DEC R9 ; Decrement counterJNZ Loop ; Not yet
done... ; Copy completed
```

MOVX.A 指令的 28 个可能寻址组合中的 10 个可使用 MOVA 指令。 这节省了两个字节和代码周期。 寻址组合的示例如下：

```
MOVX.A Rsrc,Rdst MOVA Rsrc,Rdst ; Reg/RegMOVX.A #imm20,Rdst MOVA #imm20,Rdst ;
Immediate/RegMOVX.A &abs20,Rdst MOVA &abs20,Rdst ; Absolute/RegMOVX.A @Rsrc,Rdst MOVA
@Rsrc,Rdst ; Indirect/RegMOVX.A @Rsrc+,Rdst MOVA @Rsrc+,Rdst ; Indirect,Auto/RegMOVX.A
Rsrc,&abs20 MOVA Rsrc,&abs20 ; Reg/Absolute
```

只有当 16 位索引已能满足寻址需求时，下四个复位才可行：

```
MOVX.A z20(Rsrc),Rdst MOVA z16(Rsrc),Rdst ; Indexed/RegMOVX.A Rsrc,z20(Rdst) MOVA
Rsrc,z16(Rdst) ; Reg/IndexedMOVX.A symb20,Rdst MOVA symb16,Rdst ; Symbolic/RegMOVX.A
Rsrc,symb20 MOVA Rsrc,symb16 ; Reg/Symbolic
```

4.6.3.18 POPM

POPM.A	从堆栈中恢复 n 个 CPU 寄存器（20 位数据）	
POPM.[W]	从堆栈中恢复 n 个 CPU 寄存器（16 位数据）	
句法	`POPM.A #nM Rdst`	1≤n≤16
	`POPM.W #nM Rdst` 或 `POPM #nM Rdst`	1≤n≤16
运行	POPM.A：将堆栈内的寄存器值恢复至指定的 CPU 寄存器。 针对每个从堆栈中恢复的寄存器，SP 增 4。 堆栈（每寄存器两个字）的 20 位值被恢复至寄存器。 POPM.W：将堆栈中的 16 位寄存器值恢复至指定的 CPU 寄存器。 针对每个从堆栈中恢复的寄存器，SP 增 2。 堆栈（每寄存器一个字）的 16 位值被恢复至 CPU 寄存器。 请注意：这条指令并不使用扩展字。	
说明	POPM.A：被压入堆栈的 CPU 寄存器被移动至扩展 CPU 寄存器，从 CPU 寄存器开始（Rdst-n+1）。 运算后，SP 增加 (nx4)。 POPM.A：被压入堆栈的 16 位寄存器被移回至 CPU 寄存器，从 CPU 寄存器开始（Rdst-n+ 1）。 运算后，SP 增加 (nx 2)。 被恢复 CPU 寄存器的 MSB (Rdst.19:16) 被清零。	
状态位	状态位不受影响，除了包含在运算中的 SR。	
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。	
示例	从堆栈恢复 20 位寄存器 R9，R10，R11，R12，R13。	

```
POPM.A #5,R13 ; Restore R9, R10, R11, R12, R13
```

示例	从堆栈恢复 16 位寄存器 R9，R10，R11，R12，R13。

```
POPM.W #5,R13 ; Restore R9, R10, R11, R12, R13
```

4.6.3.19 PUSHM

PUSHM.A	在堆栈上保存 n 个 CPU 寄存器（20 位数据）	
PUSHM.[W]	在堆栈上保存 n 个 CPU 寄存器（16 位数据）	
句法	`PUSHM.A #nM Rdst`	1≤n≤16
	`PUSHM.W #nM Rdst` 或 `PUSHM #nM Rdst`	1≤n≤16
运行	PUSHM.A：将 20 位 CPU 寄存器值保存在堆栈上。 对于每个存储在堆栈上的寄存器，SP 减 4。 MSB 被首先存储（较高地址）。 PUSHM.W：将 16 位 CPU 寄存器值保存在堆栈上。 对于每个存储在堆栈上的寄存器，SP 减 2。	
说明	PUSHM.A：从 Rdst 开始向后的 n 个 CPU 寄存器被存储在堆栈上。 运算后，SP 减少 (nx4)。 被压入的 CPU 寄存器的数据 (Rn.19:0) 不受影响。 PUSHM.W：从 Rdst 开始向后的 n 个寄存器被存储在堆栈上。 运算后，SP 减少 (nx2)。 被压入的 CPU 寄存器的数据 (Rn.19:0) 不受影响。 请注意：这条指令不使用扩展字。	
状态位	状态位不受影响。	
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。	
示例	在堆栈上保存 5 个 20 位寄存器 R9，R10，R11，R12，R13。	

```
PUSHM.A #5,R13 ; Save R13, R12, R11, R10, R9
```

示例	在堆栈上保存 5 个 16 位寄存器 R9，R10，R11，R12，R13。

```
PUSHM.W #5,R13 ; Save R13, R12, R11, R10, R9
```

4.6.3.20 POPX

*** POPX.A**	从堆栈恢复单个地址字
*** POPX.[W]**	从堆栈恢复单个字
*** POPX.B**	从堆栈恢复单个字节

句法	`POPX.A dst`
	`POPX dst`或 `POPX.W dst`
	`POPX.B dst`

运算	将堆栈中的 8/16/20 位值恢复至目的操作数。可使用 20 位数据。 SP 加 2（字节和字操作数）和加 4（地址字操作数）。
仿真	`MOVX(.B,.A) @SP+M dst`
说明	TOS 上的项目被写入目的操作数。 可使用寄存器模式、已索引模式、符号模式、和绝对模式。 SP 加 2 或者加 4。 请注意：对于字节运算，SP 也加 2。
状态位	状态位不受影响。
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	将 TOS 上的 16 位值写入 20 位地址 &EDE。

```
POPX.W &EDE ; Write word to address EDE
```

示例	将 TOS 上的 20 位值写入 R9

```
POPX.A R9 ; Write address-word to R9
```

4.6.3.21 PUSHX

PUSHX.A	将单地址字保存至堆栈
PUSHX.[W]	将单字写入堆栈
PUSHX.B	将单字节写入堆栈

句法	`PUSHX.A src`
	`PUSHX src`或 `PUSHX.W src`
	`PUSHX.B src`

运算	保存 TOS 上的 8/16/20 位值。可使用 20 位地址。 写入操作前，SP 减 2（字节和字操作数）和减 4（地址字操作数）。
说明	SP 减 2（字节和字操作数）或减 4（地址字操作数）。 然后源操作数被写入 TOS。 对于源操作数，所有七个寻址模式均可使用。
状态位	状态位不受影响。
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	将字节保存在堆栈上的 20 位地址 &EDE 内。

```
PUSHX.B &EDE ; Save byte at address EDE
```

示例	将 20 位值保存在堆栈上的 R9 中。

```
PUSHX.A R9 ; Save address-word in R9
```

4.6.3.22 RLAM

RLAM.A	算术左旋 20 位 CPU 寄存器内容	
RLAM.[W]	算术左旋 16 位 CPU 寄存器内容	
句法	`RLAM.A #nM Rdst`	1≤n≤4
	`RLAM.W #nM Rdst`或 `RLAM #nM Rdst`	1≤n≤4
运算	C←MSB←MSB-1 .... LSB+1←LSB←0	
说明	如图 4-43所示，目的操作数被用算术的方法左移 1，2，3 或 4 个位位置。 RLAM 运行为一个 2，4，8 或 16 的倍乘（带符号和不带符号的）。 字指令 RLAM.W 清零位 Rdst.19:16。 请注意：这条指令不使用扩展字。	
状态位	N： 如果结果为负则置位 .A: Rdst.19=1，如果 Rdst.19=0 则复位 .W: Rdst.15=1，如果 Rdst.15=0 则复位 Z： 如果结果为零则置位，否则复位 C： 从 MSB (n=1)，MSB-1 (n=2)，MSB-2 (n=3)，MSB-3 (n=4) 载入 V： 未定义	
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。	
示例	R5 中的 20 位操作数被左移三个位置。 它的操作与一个算术 8 倍乘等效。	

```
RLAM.A #3,R5 ; R5 = R5 x 8
```

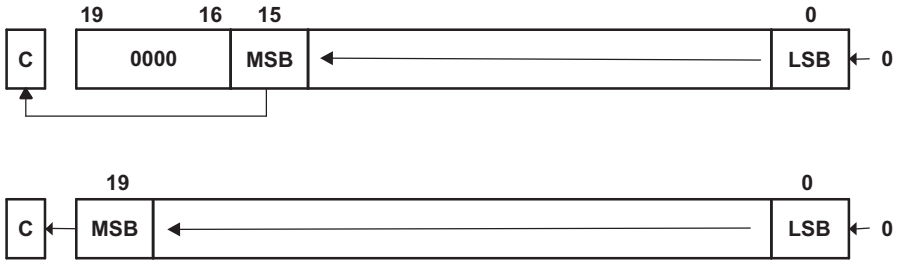


图 **4-43.** 用算术的方法左移 **- RLAM[.W]** 和 **RLAM.A**

4.6.3.23 RLAX

*** RLAX.A**	算术左旋地址字
*** RLAX.[W]**	算术左旋字
*** RLAX.B**	算术左旋字节

句法	`RLAX.A dst`
	`RLAX dst`或 `RLAX.W dst`
	`RLAX.B dst`

运算	C←MSB←MSB-1 .... LSB+1←LSB←0

仿真	`ADDX.A dstMdst`
	`ADDX dstMdst`
	`ADDX.B dstMdst`

说明	如图 4-44中所示，目的操作数向左移动一个位置。 MSB 被移入进位位 (C)，而 LSB 被 0 填充。RLAX 指令运行为一个带符号的 2 倍乘。
状态位	N： 如果结果为负则置位，如果为正则复位 Z： 如果结果为零则置位，否则复位 C： 从 MSB 载入 V： 如果一个算术溢出发生：初始值为 040000h≤dst<0C0000h，则置位；否则复位 如果一个算术溢出发生：初始值为 04000h≤dst<0C000h，则置位：否则复位 如果一个算术溢出发生：初始值为 040h≤dst<0C0h，则置位；否则复位
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	R7 中的 20 位值乘以 2。

```
RLAX.A R7 ; Shift left R7 (20-bit)
```

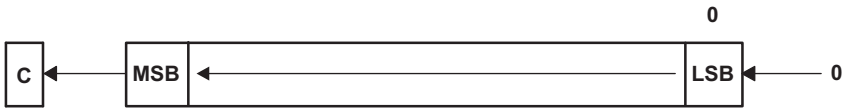


图 **4-44.** 目的操作数-算术左移位

4.6.3.24 RLCX

*** RLCX.A** 通过进位地址字左旋
*** RLCX.[W]** 通过进位字左旋
*** RLCX.B** 通过进位字节左旋

句法 `RLCX.A dst`
`RLCX dst`或 `RLCX.W dst`
`RLCX.B dst`

运算 C←MSB←MSB-1 .... LSB+1←LSB←C

仿真 `ADDCX.A dstMdst`
`ADDCX dstMdst`
`ADDCX.B dstMdst`

说明 如图 4-45中所示，目的操作数向左移动一个位置。 进位位 (C) 被移入 LSB，而 MSB 被移入进位位 (C)。

状态位
N： 如果结果为负则置位，如果为正则复位
Z： 如果结果为零则置位，否则复位
C： 从 MSB 载入
V： 如果一个算术溢出发生：初始值为 040000h≤dst<0C0000h，则置位；否则复位
如果一个算术溢出发生：初始值为 04000h≤dst<0C000h，则置位；否则复位
如果一个算术溢出发生：初始值为 040h≤dst<0C0h，则置位；否则复位

模式位 OSCOFF，CPUOFF 和 GIE 不受影响。

示例 R5 中的 20 位值被向左移动一个位置。

```
RLCX.A R5 ; (R5 x 2) + C -> R5
```

示例 RAM 字节 LEO 被向左移位一个位置。 PC 正指向上部存储器。

```
RLCX.B LEO ; RAM(LEO) x 2 + C -> RAM(LEO)
```

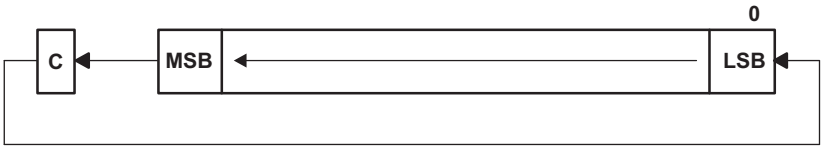


图 **4-45.** 目的操作数**-**进位左移位

4.6.3.25 RRAM

RRAM.A	算术右旋 20 位 CPU 内容	
RRAM.[W]	算术右旋 16 位 CPU 内容	
句法	`RRAM.A #n,Rdst`	1≤n≤4
	`RRAM.W #nM Rdst`或`RRAM #nM Rdst`	1≤n≤4
运算	MSB→MSB→MSB-1→ ... LSB+1→LSB→C	
说明	如图 4-46所示，目的操作数被用算术的方法右移 1，2，3 或 4 个位位置。 MSB 保持其值（符号）。 BRAM 的运行与一个带符号的 2/4/8/16 除法等效。 MSB 被保持并且被移入 MSB-1。 LSB+1 被移入 LSB，而 LSB 被移入进位位 C。字指令 RRAM.W 清零位 Rdst.19:16。 请注意：这条指令不使用扩展字。	
状态位	N： 如果值为负则置位 .A: Rdst.19=1，如果 Rdst.19=0 则复位 .W: Rdst.15=1，如果 Rdst.15=0 则复位 Z： 如果结果为零则置位，否则复位 C： 从 LSB (n=1)，LSB+1 (n=2)，LSB+2 (n=3)，或 LSB+3 (n=4) 载入 V： 复位	
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。	
示例	R5 中带符号的 20 位数被用算术的方法向右移位两个位置。	

```
RRAM.A #2,R5 ; R5/4 -> R5
```

示例 R15 中的 20 位值乘以 0.75。(0.5+0.25)×R15。

```
PUSHM.A #1,R15 ; Save extended R15 on stackRRAM.A #1,R15 ; R15 y 0.5 -
> R15ADDX.A @SP+,R15 ; R15 y 0.5 + R15 = 1.5 y R15 -
> R15RRAM.A #1,R15 ; (1.5 y R15) y 0.5 = 0.75 y R15 -> R15
```

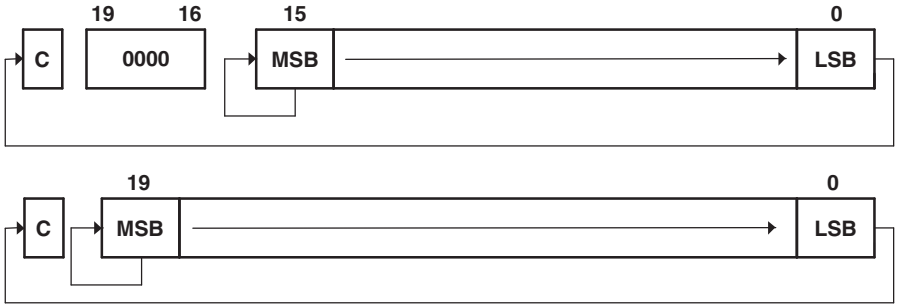


图 **4-46.** 算术右旋 **RRAM[.W]** 和 **RRAM.A**

4.6.3.26 RRAX

RRAX.A	算术右旋 20 位操作数
RRAX.[W]	算术右旋 16 位操作数
RRAX.B	算术右旋 8 位操作数

句法	`RRAX.A Rdst`
	`RRAX.W Rdst`
	`RRAX Rdst`
	`RRAX.B Rdst`
	`RRAX.A dst`
	`RRAX dst`或 `RRAX.W dst`
	`RRAX.B dst`

运算	MSB→MSB→MSB-1→ ... LSB+1→LSB→C
说明	针对目的操作数的寄存器模式：如图 4-47所示，目的操作数右移一个位位置。 MSB 保持其值（符号）。 字指令 RRAX.W 清零位 Rdst.19:16，字节指令 RRAX.B 清零位 Rdst.19:8。MSB 保持其值（符号），LSB 被移入进位位。 RRAX 的运行与带符号的被 2 除等效。 针对目的操作数的所有其它模式：如图 4-48所示，目的操作数被算术右移一个位位置。 MSB 保持其值（符号），LSB 被移入进位位。 这里 RRAX 的运行与带符号的被 2 除等效。除立即模式之外的所有寻址模式可在整个存储器内使用。
状态位	N： 如果结果为负则置位，如果为正则复位 .A: dst.19=1，如果 dst.19=0 则复位 .W: dst.15=1，如果 dst.15=0 则复位 .B: dst.7=1，如果 dst.7=0 则复位 Z： 如果结果为零则置位，否则复位 C： 从 LSB 载入 V： 复位
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	R5 中带符号的 20 位数被用算术右移位一个位置。

```
RPT #4RRAX.A R5 ; R5/16 -> R5
```

示例 EDE 中的带符号 8 位值乘以 0.5。

```
RRAX.B &EDE ; EDE/2 -> EDE
```

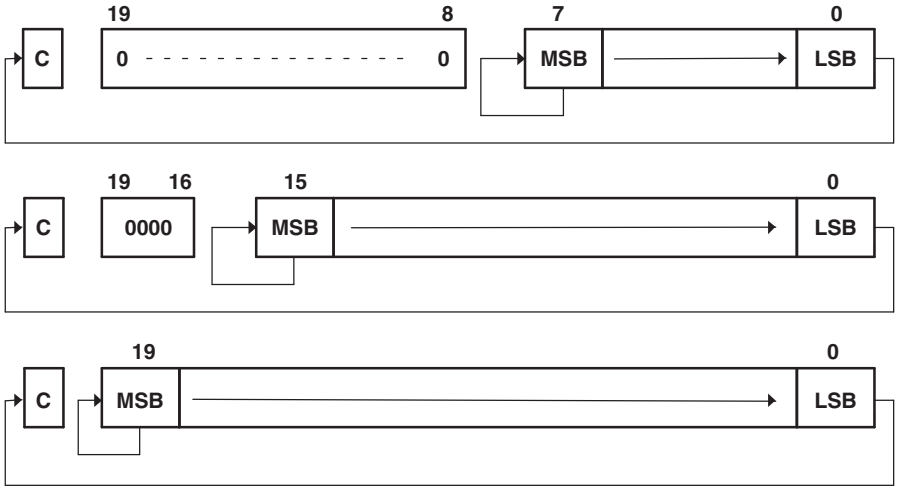


图 **4-47.** 算术右旋 **RRAX (.B，.A) -** 寄存器模式

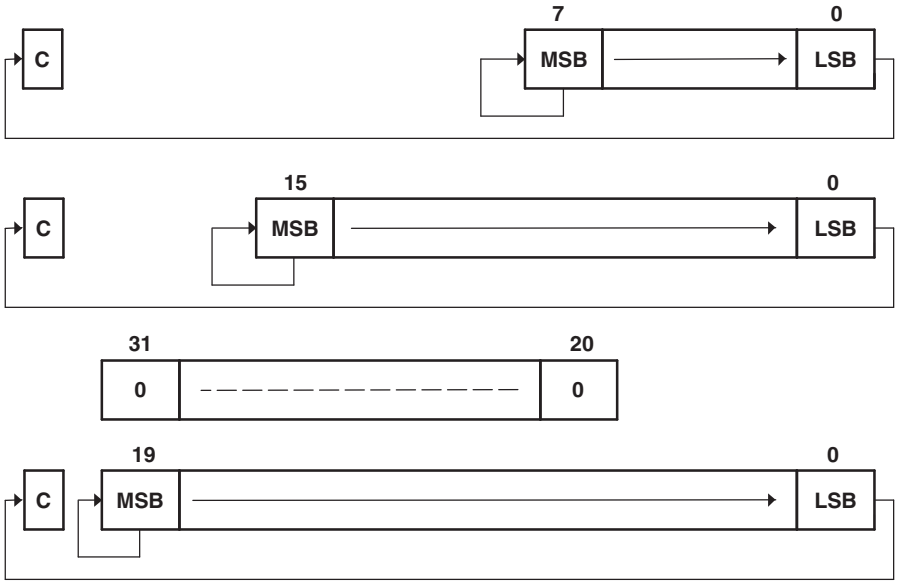


图 **4-48.** 算术右旋 **RRAX (.B，.A) -** 非寄存器模式

4.6.3.27 RRCM

RRCM.A	通过进位 20 位 CPU 寄存器内容右旋	
RRCM.[W]	通过进位 16 位 CPU 寄存器内容右旋	
句法	`RRCM.A #nM Rdst`	1≤n≤4
	`RRCM.W #nM Rdst`或`RRCM #nM Rdst`	1≤n≤4
运算	C→MSB→MSB-1→ ... LSB+1→LSB→C	
说明	如图 4-49所示，目的操作数右移 1，2，3 或 4 个位位置。 进位位 C 被移入 MSB，而 LSB 被移入进位位。 字指令 RRCM.W 清零位 Rdst.19:16。 请注意：这条指令不使用扩展字。	
状态位	N： 如果值为负则置位 .A: Rdst.19=1，如果 Rdst.19=0 则复位 .W: Rdst.15=1，如果 Rdst.15=0 则复位 Z： 如果结果为零则置位，否则复位 C： 从 LSB (n=1)，LSB+1 (n=2)，LSB+2 (n=3)，或 LSB+3 (n=4) 载入 V： 复位	
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。	
示例	R5 中的地址字被右移三个位置。 将 1 载入 MSB-2。	

```
SETC ; Prepare carry for MSB-2RRCM.A #3,R5 ; R5 = R5 » 3 + 20000h
```

示例 R6 中的地址字被右移两个位置。 将 LSB 载入 MSB。 将进位标志载入 MSB-1。

```
RRCM.W #2,R6 ; R6 = R6 » 2. R6.19:16 = 0
```

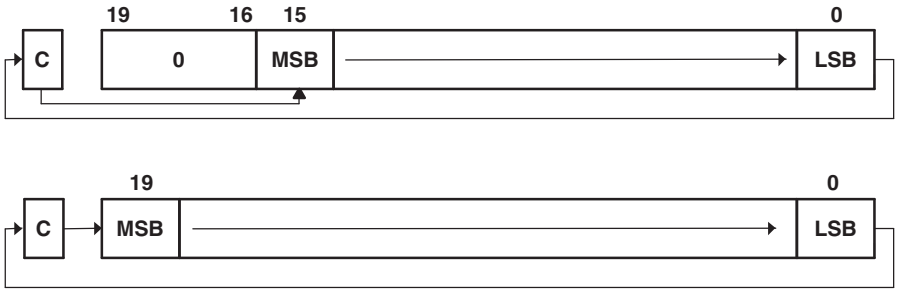


图 **4-49.** 通过进位 **RRCM[.W]** 和 **RRCM.A** 右旋

4.6.3.28 RRCX

RRCX.A	通过进位 20 位操作数右旋
RRCX.[W]	通过进位 16 位操作数右旋
RRCX.B	通过进位 8 位操作数右旋

句法	`RRCX.ARdst`
	`RRCX.WRdst`
	`RRCX Rdst`
	`RRCX.B Rdst`
	`RRCX.A dst`
	`RRCX dst`或 `RRCX.W dst`
	`RRCX.B dst`

运算	C→MSB→MSB-1→ ... LSB+1→LSB→C
说明	针对目的操作数的寄存器模式：如图 4-50所示，目的操作数右移一个位位置。 字指令 RRCX.W 清零位 Rdst.19:16，字节指令 RRCX.B 清零位 Rdst.19:8。进位位 C 被移入 MSB，而 LSB 被移入进位位。 针对目的操作数的所有其它模式：如图 4-51所示，目的操作数被算术右移一个位位置。 进位位 C 被移入 MSB，而 LSB 被移入进位位。 除立即模式之外的所有寻址模式可在整个存储器内使用。
状态位	N： 如果值为负则置位 .A: dst.19=1，如果 dst.19=0 则复位 .W: dst.15=1，如果 dst.15=0 则复位 .B: dst.7=1，如果 dst.7=0 则复位 Z： 如果结果为零则置位，否则复位 C： 从 LSB 载入 V： 复位
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	地址 EDE 上的 20 位操作数被右移一个位置。 将 1 载入 MSB。

```
SETC ; Prepare carry for MSBRRCX.A EDE ; EDE = EDE » 1 + 80000h
```

示例 R6 中的字被右移 12 个位置。

```
RPT #12RRCX.W R6 ; R6 = R6 » 12. R6.19:16 = 0
```

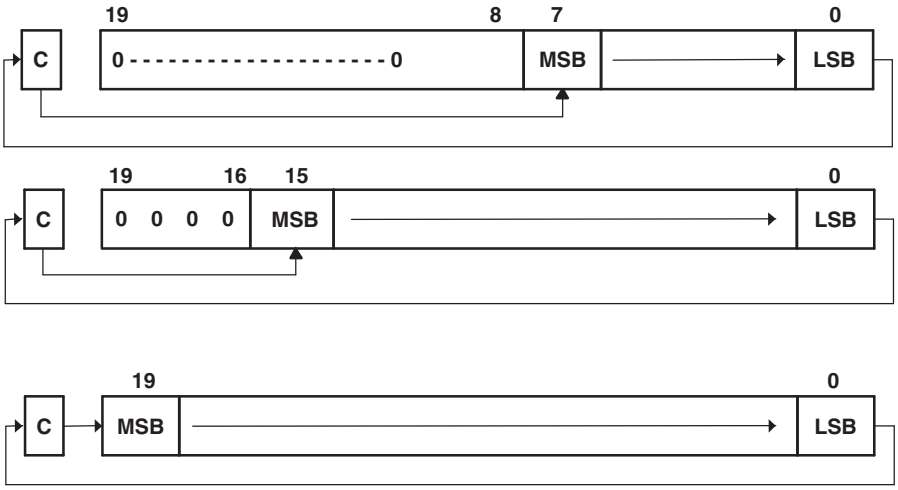


图 **4-50.** 通过进位 **RRCX (.B，.A)** 右旋-寄存器模式

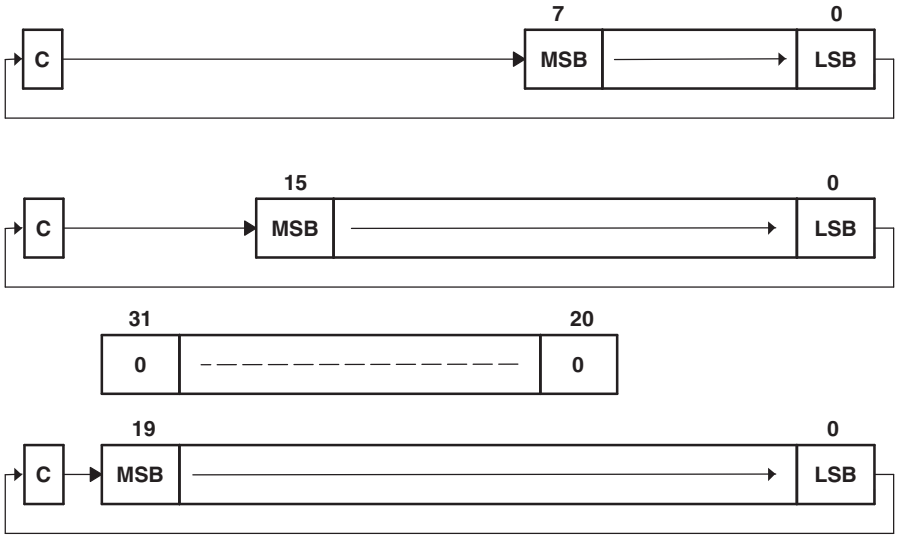


图 **4-51.** 通过进位 **RRCX (.B，.A)** 右旋-非寄存器模式

4.6.3.29 RRUM

RRUM.A	通过进位 20 位 CPU 寄存器内容右旋	
RRUM.[W]	通过进位 16 位 CPU 寄存器内容右旋	
句法	`RRUM.A #nM Rdst`	1≤n≤4
	`RRUM.W #nM Rdst`或`RRUM #nM Rdst`	1≤n≤4
运算	0→MSB→MSB-1.... LSB+1→LSB→C	
说明	如图 4-52所示，目的操作数右移 1，2，3 或 4 个位位置。 零被移入 MSB，而 LSB 被移入进位位。 RRUM 运行为一个无符号 2，4，8 或 16 除法。 字指令 RRUM.W 清零位 Rdst.19:16。 请注意：这条指令不使用扩展字。	
状态位	N： 如果值为负则置位 .A: Rdst.19=1，如果 Rdst.19=0 则复位 .W: Rdst.15=1，如果 Rdst.15=0 则复位 Z： 如果结果为零则置位，否则复位 C： 从 LSB (n=1)，LSB+1 (n=2)，LSB+2 (n=3)，或 LSB+3 (n=4) 载入 V： 复位	
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。	
示例	R5 中的无符号地址字被 16 除。	

```
RRUM.A #4,R5 ; R5 = R5 » 4. R5/16
```

示例 R6 中的字被右移一个位。 将 0 载入 MSB R6.15。

```
RRUM.W #1,R6 ; R6 = R6/2. R6.19:15 = 0
```

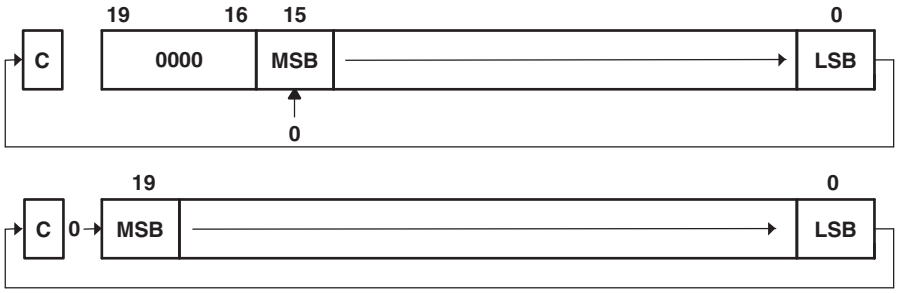


图 **4-52.** 右旋无符号 **RRUM[.W]** 和 **RRUM.A**

4.6.3.30 RRUX

RRUX.A	无符号右移 20 位 CPU 寄存器内容
RRUX.[W]	无符号右移 16 位 CPU 寄存器内容
RRUX.B	无符号右移 8 位 CPU 寄存器内容

句法	`RRUX.A Rdst`
	`RRUX.W Rdst`
	`RRUX Rdst`
	`RRUX.B Rdst`

运算	C=0→MSB→MSB-1 ... LSB+1→LSB→C
描述	RRUX 只对寄存器模式有效：如图 4-53所示，目的操作数右移一个位位置。 字指令 RRUX.W 清零位 Rdst.19:16。 字节指令 RRUX.B 清零位 Rdst.19:8。零被移入 MSB，而 LSB 被移入进位位。
状态位	N： 如果值为负则置位 .A: dst.19=1，如果 dst.19=0 则复位 .W: dst.15=1，如果 dst.15=0 则复位 .B: dst.7=1，如果 dst.7=0 则复位 Z： 如果结果为零则置位，否则复位 C： 从 LSB 载入 V： 复位
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	R6 中的字被右移 12 个位置。

```
RPT #12RRUX.W R6 ; R6 = R6 » 12. R6.19:16 = 0
```

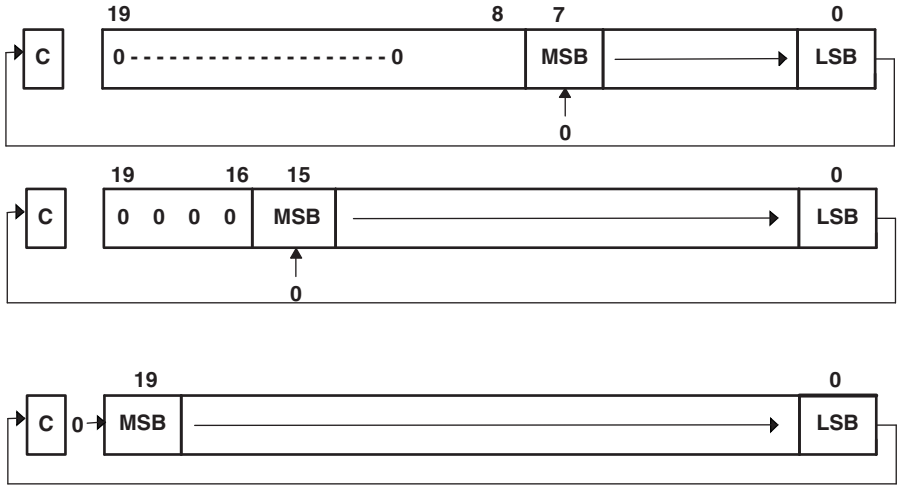


图 **4-53.** 右旋无符号 **RRUN (.B，.A) -** 寄存器模式

4.6.3.31 SBCX

*** SBCX.A**	将借位（.NOT. 进位）从目的地址字中减去
*** SBCX.[W]**	将借位（.NOT. 进位）从目的字中减去
*** SBCX.B**	将借位（.NOT. 进位）从目的字节中减去

句法	`SBCX.A dst`
	`SBCX dst`或`SBCX.W dst`
	`SBCX.B dst`

运算	dst+0FFFFFh+C→dst dst+0FFFFh+C→dst dst+0FFh+C→dst

仿真	`SBCX.A #0M dst`
	`SBCX #0M dst`
	`SBCX.B #0M dst`

说明	进位位 (C) 被加至目的操作数减一。 目的操作数之前的内容丢失。
状态位	N： 如果结果为负则置位，如果为正则复位 Z： 如果结果为零则置位，否则复位 C： 如果有一个来自结果的 MSB 的进位，则置位，否则复位 如果无借位则置位为 1，如果有借位则复位 V： 如果一个算术溢出发生，则置位，否则复位
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	R13 指向的 8 位计数器被从一个 R12 指向的 16 位计数器内。

```
SUBX.B @R13,0(R12) ; Subtract LSDsSBCX.B 1(R12) ; Subtract carry from MSD
```

注： 借位执行

借位被视为一个 .NOT.进位：

借位	进位位
支持	0
否	1

4.6.3.32 SUBX

SUBX.A 从目的地址字中减去源地址字
SUBX.[W] 从目的字中减去源字
SUBX.B 从目的字节中减去源字节

句法 `SUBX.A srcM dst`

`SUBX srcM dst` 或 `SUBX.W srcM dst`

`SUBX.B srcM dst`

运算 (.not. src)+1+dst→dst 或 dst-src→dst

说明 从目的操作数中减去源操作数。 这通过在目的中增加源 + 1 的 1s 补数来完成。 源操作数不受影响。 结果被写入目的操作数。 两个操作数都位于完全地址空间内。

状态位
- N： 如果结果为负 (src>dst)，则置位，如果为正则复位 (src≤dst)
- Z： 如果为零 (src=dst) 则置位，否则复位 (src≠dst)
- C： 如果有来自 MSB 的进位，则置位，否则复位
- V： 如果从一个正目的操作数中减去一个负源操作数得到一个负结果，或者如果从一个负目的操作数中减去一个正源操作数得到一个正结果，则置位，否则复位（无溢出）。

模式位 OSCOFF，CPUOFF 和 GIE 不受影响。

示例 从 EDE (LSB) 和 EDE+2 (MSB) 中减去一个 20 位 常数 87654h。

```
SUBX.A #87654h,EDE ; Subtract 87654h from EDE+2|EDE
```

示例 R5（20 位地址）指向的一个表格字被从 R7 中减去。 指令后，如果 R7 包含零，则跳转至标签 TONI。 R5 自动增量 2。 R7.19:16=0

```
SUBX.W @R5+,R7 ; Subtract table number from R7. R5 + 2JZ TONI ; R7 = @R5 (before
subtraction)... ; R7 <> @R5 (before subtraction)
```

示例 从指向完全地址空间的字节 R12 中减去字节 CNT。 地址 CNT 在 PC±512K 内。

```
SUBX.B CNT,0(R12) ; Subtract CNT from @R12
```

请注意：在下面两个情况中使用 SUBA 以实现更佳的代码密度和执行性能。

```
SUBX.A Rsrc,RdstSUBX.A #imm20,Rdst
```

4.6.3.33 SUBCX

SUBCX.A	从目的地址字中减去带有进位的源地址
SUBCX.[W]	从目的字中减去带有进位的源字
SUBCX.B	从目的字节中减去带有进位的源字节

句法	`SUBCX.A srcM dst`
	`SUBCX srcM dst`或`SUBCX.W srcM dst`
	`SUBCX.B srcM dst`

运算	(.not. src)+C+dst→dst 或 dst-(src-1)+C→dst
说明	从目的操作数中减去源操作数。 这通过在目的中增加源+进位的 1s 补数来完成。 源操作数不受影响，结果被写入目的操作数。 两个操作数都位于完全地址空间内。
状态位	N： 如果结果为负 (MSB=1)，则置位，如果为正 (MSB=0)，则复位 Z： 如果结果为零则置位，否则复位 C： 如果有来自 MSB 的进位，则置位，否则复位 V： 如果从一个正目的操作数中减去一个负源操作数得到一个负结果，或者如果从一个负目的操作数中减去一个正源操作数得到一个正结果，则置位，否则复位（无溢出）。
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	一个 20 位常数 87654h 被从带有来自之前指令进位的 R5 中减去。

```
SUBCX.A #87654h,R5 ; Subtract 87654h + C from R5
```

示例 从由 R7 指向的 RAM 中的一个 48 位计数器内减去由 R5（20 位地址）指向的一个 48 位数（3 个字）。 R5 自动增量来指向下一个 48 位数。

```
SUBX.W @R5+,0(R7) ; Subtract LSBs. R5 + 2SUBCX.W @R5+,2(R7) ; Subtract MIDs with C. R5
+ 2SUBCX.W @R5+,4(R7) ; Subtract MSBs with C. R5 + 2
```

示例 从 R12 指向的字节中减去字节 CNT 。 使用之前指令的进位。20 位地址。

```
SUBCX.B &CNT,0(R12) ; Subtract byte CNT from @R12
```

4.6.3.34 SWPBX

SWPBX.A	较低字的交换字节
SWPBX.[W]	字的交换字节
句法	`SWPBX.A dst` `SWPBX dst`或`SWPBX.W dst`
运算	dst.15:8↔dst.7:0
说明	寄存器模式：Rn.15:8 与 Rn.7:0 交换。当使用 .A 扩展名时，Rn.19:16 保持不变。 当使用 .W 扩展名时，Rn.19:16 被清零。 其它模式：当使用 .A 扩展名时，目的地址的位 31:20 被清零，位 19:16 保持不变，而位 15:8 与位 7:0 交换。当使用 .W 扩展名时，位 15:8 与被寻址字的位 7:0 交换。
状态位	状态位不受影响。
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	交换 RAM 地址字 EDE 的字节

```
MOVX.A #23456h,&EDE ; 23456h -> EDESWPBX.A EDE ; 25634h -> EDE
```

示例 交换 R5 的字节

```
MOVA #23456h,R5 ; 23456h -> R5SWPBX.W R5 ; 05634h -> R5
```

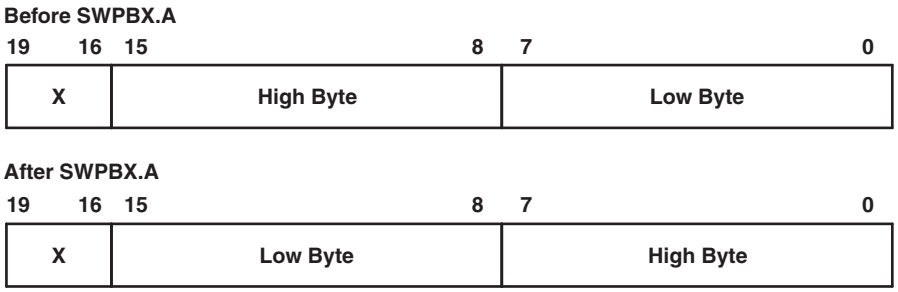


图 **4-54.** 交换字节 **SWPBX.A** 寄存器模式

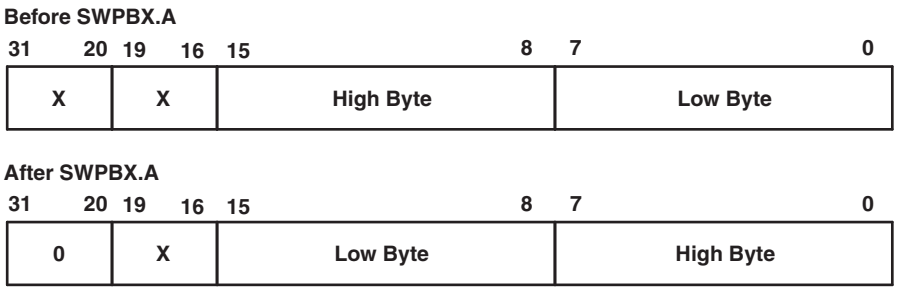


图 **4-55.** 在存储器中交换 **SWPBX.A** 字节

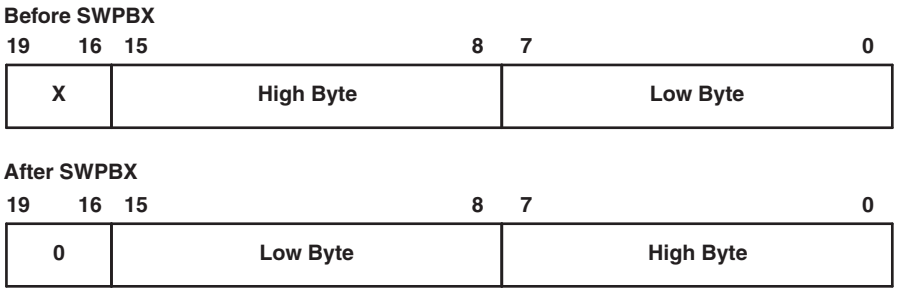


图 **4-56.** 交换字节 **SWPBX[.W]** 寄存器模式

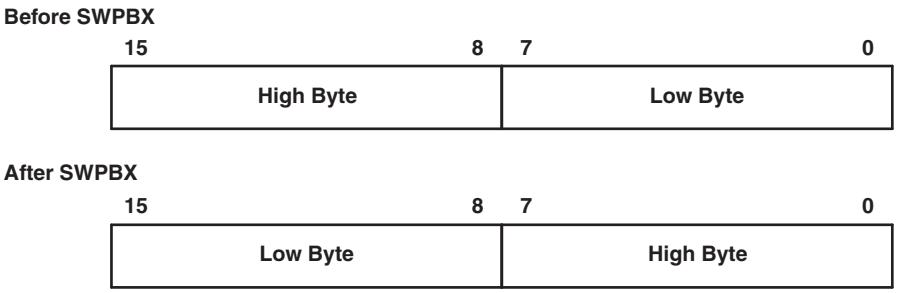


图 **4-57.** 在存储器中交换 **SWPBX[.W]** 字节

4.6.3.35 SXTX

SXTX.A	将较低字节的符号扩展为地址字
SXTX.[W]	将较低字节的符号扩展为字
句法	`SXTX.A dst` `SXTX dst`或`SXTX.W dst`
运算	dst.7→dst.15:8，Rdst.7→Rdst.19:8（寄存器模式）
说明	寄存器模式：操作数 (Rdst.7) 低字节的符号被扩展至位 Rdst.19:8 内。 其他模式：SXTX.A：操作数 (dst.7) 低字节的符号被扩展至位 dst.19:8 内。位 dst.31:20 被清零。 SXTX[.W]：操作数 (dst.7) 低字节的符号被扩展至位 dst.15:8 内。
状态位	N： 如果结果为负则置位，否则复位 Z： 如果结果为零则置位，否则复位 C： 如果结果不为零则置位，否则复位 (C=.NOT.Z) V： 复位
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	EDE.7:0 内的带符号的 8 位数据符号被扩展至 20 位：EDE.19:8。位于 EDE+2 内的位 31:20 被清零。

```
SXTX.A &EDE ; Sign extended EDE -> EDE+2/EDE
```

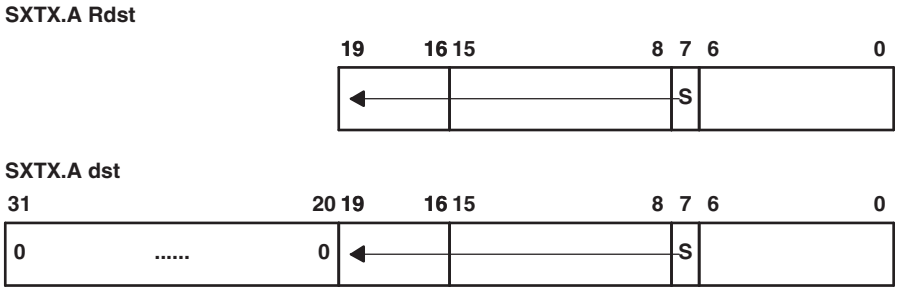


图 **4-58.** 符号扩展 **SXTX.A**

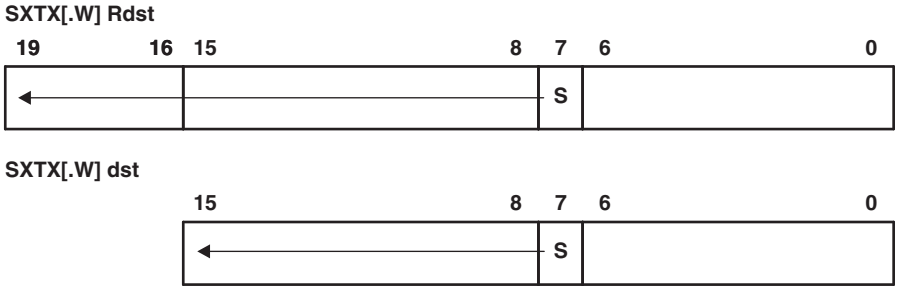


图 **4-59.** 符号扩展 **SXTX[.W]**

4.6.3.36 TSTX

*** TSTX.A**	测试目的地址字
*** TSTX.[W]**	测试目的字
*** TSTX.B**	测试目的字节

句法	`TSTX.A dst`
	`TSTX dst`或`TSTX.W dst`
	`TSTX.B dst`

运算	dst+0FFFFFh+1 dst+0FFFFh+1 dst+0FFh+1

仿真	`CMPX.A #0M dst`
	`CMPX #0M dst`
	`CMPX.B #0M dst`

说明	目的操作数与零相比较。 根据结果置位状态位。 目的操作数不受影响。
状态位	N： 如果目的操作数为负则置位，如果为正则复位 Z： 如果目的操作数包含零则置位，否则复位 C： 设置 V： 复位
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	RAM 字节 LEO 被测试；PC 正指向上部存储器。 如果它为负，则继续在 LEONEG 上执行；如果为正但又不为零，则继续在 LEOPOS 上执行。

```
TSTX.B LEO ; Test LEOJN LEONEG ; LEO is negativeJZ LEOZERO ; LEO is zeroLEOPOS ......
; LEO is positive but not zeroLEONEG ...... ; LEO is negativeLEOZERO ...... ; LEO is
zero
```

4.6.3.37 XORX

XORX.A	将源地址字与目的地址字进行异或操作
XORX.[W]	源字与目的字异或操作
XORX.B	源字节与目的字节异或操作

句法	`XORX.A srcMdst`
	`XORX srcMdst` 或 `XORX.W srcMdst`
	`XORX.B srcMdst`

运算	src .xor. dst→dst
说明	源操作数和目的操作数被异或操作。 结果被放置在目的操作数内。 源操作数不受影响。 目的操作数之前的内容丢失。 两个操作数都位于完全地址空间内。
状态位	N： 如果结果为负 (MSB=1)，则置位，如果为正 (MSB=0)，则复位 Z： 如果结果为零则置位，否则复位 C： 如果结果不为零则置位，否则复位 (C=.not. 零位) V： 如果两个操作数均为负(在执行前)则置位，否则复位
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	用地址字 TONI（20 位地址）内的信息切换地址字 CNTR（20 位数据）内的位。

```
XORX.A TONI,&CNTR ; Toggle bits in CNTR
```

示例 R5（20 位地址）指向的一个表格字被用于切换 R6 中的位。

```
XORX.W @R5,R6 ; Toggle bits in R6. R6.19:16 = 0
```

示例 R7 中低字节内复位为零的那些位与位于字节 EDE（20 位地址）内的位不同。

```
XORX.B EDE,R7 ; Set different bits to 1 in R7INV.B R7 ; Invert low byte of R7. R7.19:8
= 0.
```

4.6.4 MSP430X 寻址指令

MSP430X 寻址指令支持 20 位操作数，但是具有受限的寻址模式。 寻址模式限制为寄存器模式和立即模式，除了 MOVA 指令。 对寻址模式的限制免除了对于额外扩展字运算代码的需要，从而改进了代码密度和执行时间。 在下面的部分中对 MSP430X 寻址指令进行了说明。

MSP430X 扩展指令请参阅4.6.3 节，而标准MSP430X 指令请参阅4.6.2 节。

4.6.4.1 ADDA

ADDA	将一个 20 位源添加到一个 20 位地址寄存器
句法	`ADDA RsrcM Rdst` `ADDA #imm20M Rdst`
运算	src+Rdst→Rdst
说明	20 位源操作数被添加到 20 位目的 CPU 寄存器。 目的操作数之前的内容丢失。 源操作数不受影响。
状态位	N： 如果结果为负 (Rdst.19=1)，则置位，如果为正 (Rdst.19=0)，则复位 Z： 如果结果为零则置位，否则复位 C： 如果有来自 20 位结果的进位，则置位，否则复位 V： 如果两个正操作数的结果为负，或者如果两个负数的结果为正，则置位，否则复位
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	R5 增加 0A4320h。 如果进位发生，执行到标签 TONI 的跳转。

```
ADDA #0A4320h,R5 ; Add A4320h to 20-
bit R5JC TONI ; Jump on carry... ; No carry occurred
```

4.6.4.2 BRA

*** BRA**	到目的的分指令
句法	`BRA dst`
运算	dst→PC
仿真	`MOVA dst,PC`
说明	一个无条件分支指令被完全地址空间内任何位置的 20 位地址 可使用所有七个源寻址模式。 分支指令是一个地址字指令。 如果目的地址被包含在一个存储器位置 X，它被包含在两个上升字内：X (LSB) 和 (X + 2) (MSB)。
状态位	N： 不受影响 Z： 不受影响 C： 不受影响 V： 不受影响
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	给出了针对所有寻址模式的示例。 立即模式：分支至位于 20 位地址空间内任一位置的标签 EDE 或者到地址。

```
BRA #EDE ; MOVA #imm20,PCBRA #01AA04h
```

符号模式：分支至包含在地址 EXEC (LSB) 和 EXEC+2 (MSB) 内的 20 位地址。 EXEC 位于地址 (PC+X) 上，其中 X 在 +32K 内。间接寻址。

```
BRA EXEC ; MOVA z16(PC),PC
```

请注意：如果 16 位索引不能满足需要，可用下列指令使用一个 20 位索引。

```
MOVX.A EXEC,PC ; 1M byte range with 20-bit index
```

绝对模式：分支至包含在绝对地址 EXEC (LSB) 和 EXEC+2 (MSB) 内 20 位地址。 间接寻址

```
BRA &EXEC ; MOVA &abs20,PC
```

寄存器模式：分支至包含在寄存器 R5 中的 20 位地址。 间接 R5。

```
BRA R5 ; MOVA R5,PC
```

间接模式：分支至包含在由寄存器 R5 (LSB) 指向的字内的 20 位地址。 MSB 具有地址 (R5+2)。 间接，间接 R5。

```
BRA @R5 ; MOVA @R5,PC
```

间接、自动增量模式：分支至包含在由 R5 指向的字内的 20 位地址并且之后 R5 中的地址增 4。下次 S/W 使用 R5 作为一个指针，访问由 R5 指向表中的下一个字地址使得它能够改变程序执行。 间接，间接 R5。

```
BRA @R5+ ; MOVA @R5+,PC. R5 + 4
```

已索引模式：分支至包含在由寄存器 (R5+X) 指向的地址内 20 位地址（例如，开始地址为 X 的表）。 (R5+X) 指向 LSB，(R5+X+2) 指向地址的 MSB。 X 在 R5+32 内。间接，间接 (R5+X)。

```
BRA X(R5) ; MOVA z16(R5),PC
```

请注意：如果 16 位索引不能满足需要，可用下列指令使用一个 20 位索引 X。

```
MOVX.A X(R5),PC ; 1M byte range with 20-bit index
```

4.6.4.3 CALLA

CALLA	调用一个子例程
Syntax	`CALLA dst`
运算	dst→tmp 20 位 dst 被评估和存储 SP-2→SP PC.19:16→@SP 用到 TOS 的返回地址更新 PC (MSB) SP-2→SP PC.15:0→@SP 更新到 TOS 的 PC (LSB) tmp→PC 保存 20 位 dst 到 PC
说明	在完全地址空间内任何位置的一个 20 位地址进行子例程调用。 可使用所有七个源寻址模式。 调用指令是一个地址字指令。 如果目的地址被包含在存储器位置 X，它包含在两个上升字内，X (LSB) 和 (X+2) (MSB)。 返回地址需要堆栈上的两个字。 使用指令 RETA 来完成返回。
状态位	N： 不受影响 Z： 不受影响 C： 不受影响 V： 不受影响
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	给出了针对所有寻址模式的示例。 立即模式：调用一个标签 EXEC 上的子例程或者直接调用一个地址。

```
CALLA #EXEC ; Start address EXECCALLA #01AA04h ; Start address 01AA04h
```

符号模式：调用一个包含在地址 EXEC (LSB) 和 EXEC+2 (MSB) 内的 20 位地址上的子例程。 EXEC 位于地址 (PC+X) 上，其中 X 在 PC+32K 内。间接寻址。

```
CALLA EXEC ; Start address at @EXEC. z16(PC)
```

绝对模式：调用一个包含在绝对地址 EXEC (LSB) 和 EXEC+2 (MSB) 内的 20 位地址上的子例程。 间接寻址

```
CALLA &EXEC ; Start address at @EXEC
```

寄存器模式：调用一个包含在寄存器 R5 中的 20 位地址上的子例程。 间接 R5。

```
CALLA R5 ; Start address at @R5
```

间接模式：调用一个 20 位地址上的子例程，此地址包含在由寄存器 R5 指向的字 内。 MSB 具有地址 (R5+2)。 间接，间接 R5。

```
CALLA @R5 ; Start address at @R5
```

间接，自动增量模式：调用一个包含在由 R5 指向的字中的 20 位地址上的子例程，之后 R5 中的 20 位地址增加 4。下次 S/W 使用 R5 作为一个指针，访问由 R5 指向表中的下一个字地址使得它能够改变程序执行。 间接，间接 R5。

```
CALLA @R5+ ; Start address at @R5. R5 + 4
```

已索引模式：调用一个包含在由 (R5+X) 指向的地址中的 20 位地址上的子例程；例如，一个起始地址为 X 的表。(R5+X) 指向 LSB，(R5+X) 指向字地址的 MSB。 X 在 R5+32K 内。间接，间接 (R5+X)。

```
CALLA X(R5) ; Start address at @(R5+X). z16(R5)
```

4.6.4.4 CLRA

*** CLRA**	清零 20 位目的寄存器
句法	`CLRA Rdst`
运算	0→Rdst
仿真	`MOVA #0M Rdst`
说明	目的寄存器被清零。
状态位	状态位不受影响。
示例	R10 内的 20 位值被清零。

```
CLRA R10 ; 0 -> R10
```

4.6.4.5 CMPA

CMPA 将 20 位源与 20 位目的寄存器相比较。

句法
```
CMPA RsrcM Rdst
CMPA #imm20M Rdst
```

运算 (.not. src)+1+Rdst 或 Rdst-src

说明 从 20 位目的 CPU 寄存器中减去 20 位源操作数。 通过在目的寄存器中增加源 + 1 的 1s 补数来完成。 结果只影响状态位。

状态位
- N： 如果结果为负 (src>dst)，则置位，如果为正则复位 (src≤dst)
- Z： 如果为零 (src=dst) 则置位，否则复位 (src≠dst)
- C： 如果有来自 MSB 的进位，则置位，否则复位
- V： 如果从一个正目的操作数中减去一个负源操作数得到一个负结果，或者如果从一个负目的操作数中减去一个正源操作数得到一个正结果，则置位，否则复位（无溢出）。

模式位 OSCOFF，CPUOFF 和 GIE 不受影响。

示例 将一个 20 位直接操作数与 R6 相比较。 如果他们相等，程序继续在标签 EQUAL 上执行。

```
CMPA #12345h,R6 ; Compare R6 with 12345hJEQ EQUAL ; R5 = 12345h... ; Not equal
```

示例 比较 R5 和 R6 中的 20 位值。 如果 R5 大于（带符号）或等于 R6，程序继续在标签 GRE 上执行。

```
CMPA R6,R5 ; Compare R6 with R5 (R5 - R6)JGE GRE ; R5 >= R6... ; R5 < R6
```

4.6.4.6 DECDA

*** DECDA**	双递减 20 位目的寄存器
句法	`DECDA Rdst`
运算	Rdst-2→Rdst
仿真	`SUBA #2M Rdst`
说明	目的寄存器递减 2。 原先的内容丢失。
状态位	N： 如果结果为负则置位，如果为正则复位 Z： 如果 Rdst 包含 2 则置位，否则复位 C： 如果 Rdst 包含 0 或 1 则复位，否则置位 V： 如果一个算术溢出发生则置位，否则复位
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	R5 中的 20 位值被乘以 2。

```
DECDA R5 ; Decrement R5 by two
```

4.6.4.7 INCDA

*** INCDA**	双递增 20 位目的寄存器
句法	`INCDA Rdst`
运算	Rdst+2→Rdst
仿真	`ADDA #2M Rdst`
说明	目的寄存器被递增 2。 原先的内容丢失。
状态位	N： 如果结果为负则置位，如果为正则复位 Z： 如果 Rdst 包含 0FFFFFEh 则置位，否则复位 如果 Rdst 包含 0FFFEh 则置位，否则复位 如果 Rdst 包含 0FEh 则置位，否则复位 C： 如果 Rdst 包含 0FFFFEh 或 0FFFFFh 则置位，否则复位 如果 Rdst 包含 0FFFEh 或 0FFFFh 则置位，否则复位 如果 Rdst 包含 0FEh 或 0FFh 则置位，否则复位 V： 如果 Rdst 包含 07FFFEh 或 07FFFFh 则置位，否则复位 如果 Rdst 包含 07FFEh 或 07FFFh 则置位，否则复位 如果 Rdst 包含 07Eh 或 07Fh 则置位，否则复位
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	R5 中的 20 位值被乘以 2。

```
INCDA R5 ; Increment R5 by two
```

4.6.4.8 MOVA

MOVA	将 20 位源操作数移动到 20 位目的操作数
句法	MOVA RsrcM Rdst MOVA #imm20M Rdst MOVA z16(Rsrc)M Rdst MOVA EDEM Rdst MOVA &abs20M Rdst MOVA @RsrcM Rdst MOVA @Rsrc+M Rdst MOVA RsrcM z16(Rdst) MOVA RsrcM &abs20
运算	src→Rdst Rsrc→dst
说明	20 位源操作数被移到至 20 位目的操作数。 源操作数不受影响。 目的操作数之前的内容丢失。
状态位	N： 不受影响 Z： 不受影响 C: 不受影响 V： 不受影响
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	将 R9 的 20 位复制到 R8

```
MOVA R9,R8 ; R9 -> R8
```

将 20 位立即值 12345h 写入到 R12

```
MOVA #12345h,R12 ; 12345h -> R12
```

将由 (R9+100h) 寻址的 20 位值复制到 R8。 地址 (R9+100h) LSB 和 (R9+102h) MSB 中的源操作数。

```
MOVA 100h(R9),R8 ; Index: + 32 K. 2 words transferred
```

将 20 位绝对地址 EDE (LSB) 和 EDE+2 (MSB) 内的 20 位值移动到 R12

```
MOVA &EDE,R12 ; &EDE -> R12. 2 words transferred
```

将 20 位地址 EDE (LSB) 和 EDE+2 (MSB) 内的 20 位值移动到 R12。 PC 索引 ±32K。

```
MOVA EDE,R12 ; EDE -> R12. 2 words transferred
```

将指向（20 位地址）的 20 位值复制至 R8。 地址 @R9 LSB 和 @(R9+2) MSB 内的源操作数。

```
MOVA @R9,R8 ; @R9 -> R8. 2 words transferred
```

将指向（20 位地址）的 20 位值复制至 R8。 之后 R9 增加 4。 地址 @R9 LSB 和 @(R9 + 2) MSB 内的源操作数。

```
MOVA @R9+,R8 ; @R9 -> R8. R9 + 4. 2 words transferred.
```

将 R8 内的 20 位值复制到 (R9+100h) 寻址的目的操作数。 地址 @(R9+100h) LSB 和 @(R9+102h) MSB 内的目的操作数。

```
MOVA R8,100h(R9) ; Index: +- 32 K. 2 words transferred
```

将 R13 内的 20 位值移动到 20 位绝对地址 EDE (LSB) 和 EDE+2 (MSB)。

```
MOVA R13,&EDE ; R13 -> EDE. 2 words transferred
```

将 R13 内的 20 位值移动到 20 位地址 EDE (LSB) 和 EDE+2 (MSB)。 PC 索引 ± 32K。

```
MOVA R13,EDE ; R13 -> EDE. 2 words transferred
```

4.6.4.9 RETA

*** RETA**	从子例程返回
句法	RETA
运算	@SP → PC.15:0 LSBs (15:0) of saved PC to PC.15:0 SP+2→SP @SP → PC.19:16 MSBs (19:16) of saved PC to PC.19:16 SP+2→SP
仿真	MOVA @SP+MPC
说明	被一个 CALLA 指令压入堆栈的 20 位返回地址被恢复至 PC。 程序继续在子例程调用之后的地址上执行。 SR 位 SR.11:0 不受影响。 这可实现包含这些位的信息的传送。
状态位	N： 不受影响 Z： 不受影响 C： 不受影响 V： 不受影响
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	调用一个 20 位地址空间内的任一位置上的子例程 SUBR 并且在 CALLA 之后返回至地址

```
CALLA #SUBR ; Call subroutine starting at SUBR... ; Return by RETA to hereSUBR PUSHM.A
#2,R14 ; Save R14 and R13 (20 bit data)... ; Subroutine codePOPM.A #2,R14 ; Restore
R13 and R14 (20 bit data)RETA ; Return (to full address space)
```

4.6.4.10 TSTA

*** TSTA**	测试 20 位目的寄存器
句法	`TSTA Rdst`
运算	dst+0FFFFFh+1 dst+0FFFFh+1 dst+0FFh+1
仿真	`CMPA #0M Rdst`
说明	目的寄存器与零相比较。 根据结果置位状态位。 目的寄存器不受影响。
状态位	N： 如果目的寄存器为负则置位，如果为正则复位 Z： 如果目的寄存器包含零则置位，否则复位 C： 设置 V： 复位
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	R7 中的 20 位值被测试。 如果它为负，则继续在 R7NEG 上执行；如果为正但又不为零，则继续在 R7POS 上执行。

```
TSTA R7 ; Test R7JN R7NEG ; R7 is negativeJZ R7ZERO ; R7 is zeroR7POS ..... ; R7 is
positive but not zeroR7NEG ..... ; R7 is negativeR7ZERO ..... ; R7 is zero
```

4.6.4.11 SUBA

SUBA	从 20 位目的寄存器中减去 20 位源。
句法	`SUBA RsrcM Rdst` `SUBA #imm20M Rdst`
运算	(.not.src)+1+Rdst→Rdst 或 Rdst-src→Rdst
说明	从 20 位目的寄存器中减去 20 位源操作数。 通过在目的中增加源 + 1 的 1s 补数来完成。 结果被写入目的寄存器，源操作数不受影响。
状态位	N： 如果结果为负 (src>dst)，则置位，如果为正则复位 (src≤dst) Z： 如果为零 (src=dst) 则置位，否则复位 (src≠dst) C： 如果有来自 MSB (Rdst.19) 的进位，则置位，否则复位 V： 如果从一个正目的操作数中减去一个负源操作数得到一个负结果，或者如果从一个负目的操作数中减去一个正源操作数得到一个正结果，则置位，否则复位（无溢出）。
模式位	OSCOFF，CPUOFF 和 GIE 不受影响。
示例	从 R6 中减去 R5 中的 20 位值。 如果一个进位发生，程序继续在标签 TONI 上执行。

```
SUBA R5,R6 ; R6 - R5 -> R6JC TONI ; Carry occurred... ; No carry
```

基本时钟模块+

基本时钟模块+ 为 MSP430x2xx 系列提供了时钟。 本章阐述了 MSP430x2xx 器件系列的基本时钟模块+ 的操作。

Topic **Page**

5.1 基本时钟模块+ 介绍

基本时钟模块+ 支持低系统成本和超低功耗。 采用三种内部时钟信号，用户可以选择性能和低功耗的最佳平衡。 为了实现无任何外部元件操作，可在全软件控制下，用一个外部电阻、一个或两个外部晶振、或用振荡器来配置基本时钟模块+。

基本时钟模块+ 有 2 个，3 个或 4 个时钟源：

- LFXT1CLK：低频/高频振荡器可以与低频时钟晶振或外接 32768Hz 时钟源，或与标准晶振、振荡器，外部 400KHz～16MHz 的外部时钟源一起使用。
- XT2CLK：可以与标准晶振、振荡器，或外部 400KHz～16MHz 的外部时钟源一起使用的可供选择的高频振荡器。
- DCOCLK：内部数控振荡器 (DCO)。
- VLOCLK：内部超低功耗、12KHz 典型频率的低频振荡器。

基本时钟模块+ 可提供的三种时钟信号：

- ACLK：辅助时钟。 ACLK 是由软件选择来作为 LFXT1CLK 或 VLOCLK。 ACLK经 1，2，4，8 分频后得到。ACLK 可由软件选作各个外围模块。
- MCLK：主机时钟。 MCLK 由软件选择作 LFXT1CLK，VLOCLK，XT2CLK（如果片上提供），或 DCOCLK。 MCLK 由 1，2，4，8 分频得到。MCLK 用于 CPU 和系统。
- SMCLK：系统子时钟。 SMCLK 由软件选作 LFXT1CLK，VLOCLK，XT2CLK（如果片上提供），或 DCOCLK。 SMCLK 由 1，2，4，8 分频得到。SMCLK 可由软件选作各个外围模块。

MSP430F2xx 器件中的基本时钟模块+ 的方框图如图 5-1所示。

MSP430AFE2xx 器件中的基本时钟模块+ 的方框图如图 5-2所示。

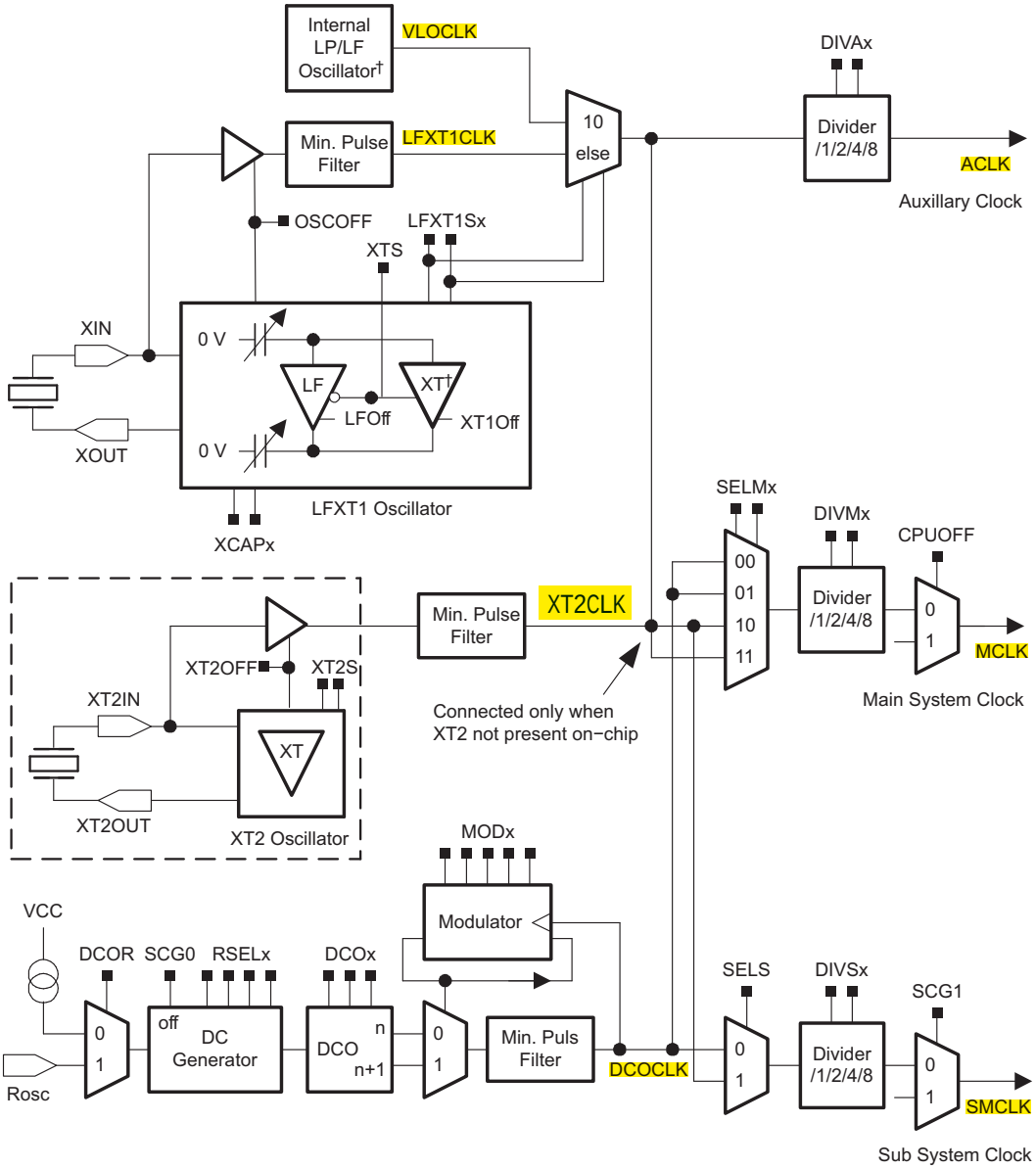


图 **5-1.** 基本时钟模块+ 框图 **— MSP430F2xx**

注: † 特定器件的时钟变化

并不是在所有的 MSP430x2xx 器件上所有的时钟特性都可用：
MSP430G22x0：没有 LFXT1，没有 XT2，不支持 ROSC。

MSP430F20xx，MSP430G2xx1，MSP430G2xx2，MSP430G2xx3：LFXT1 不支持 HF 模式，没有XT2，不支持 ROSC。
MSP430x21x1：没有内部 LP/LF 振荡器，没有 XT2，不支持ROSC。
MSP430x21x2：没有 XT2。
MSP430F22xx, MSP430x23x0：没有 XT2。

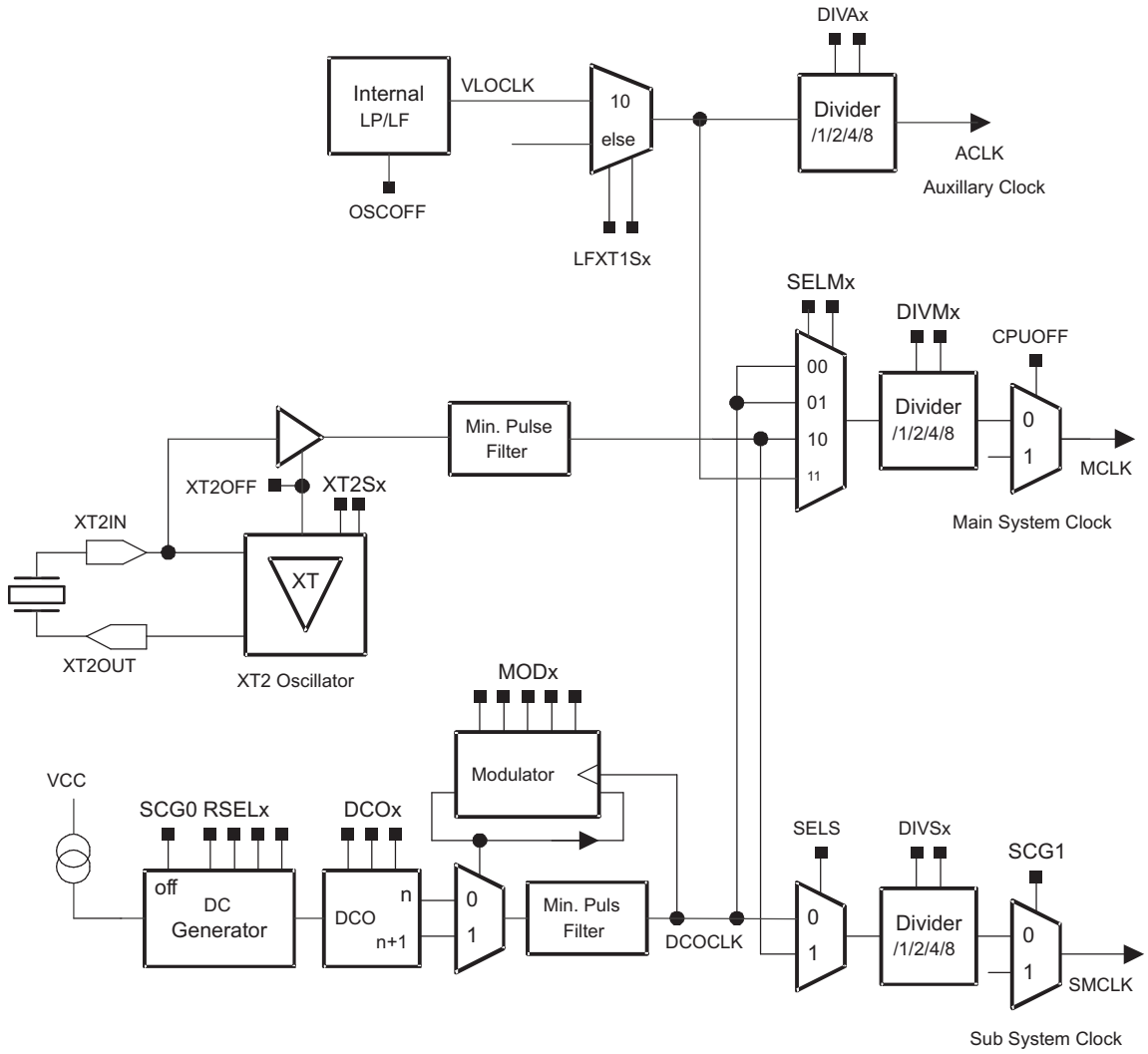


图 **5-2.** 基本时钟模块+ 框图 **— MSP430AFE2xx**

注: 在 MSP430AFE2xx 器件中没有 LFXT1。

5.2 基本时钟模块+ 的操作

在一个 PUC 后，MCLK 和 SMCLK 来自 1.1MHz 的 DCOCLK（有关数据请参阅《特定器件数据表》），ACLK 来自于带有一个 6pF 电容内部负载的处于高频模式的 LFXT1CLK。

状态寄存器中的位 SCG0，SCG1，OSCOFF，和 CPUOFF 配置 MSP430操作模式和使能或禁止部分基本时钟模块+（请参阅*系统复位，中断，和运行模式*章节）。 DCOCTL，BCSCTL1，BCSCTL2，和 BCSCTL3 寄存器配置基本时钟模块+。

在程序执行时，基本时钟模块+ 能够通过软件设置或重设置，例如：

```
CLR.B &DCOCTL ; Select lowest DCOx; and MODx settingsBIS.B #RSEL2+RSEL1+RSEL0,&BCSCTL1 ; Select range 7BIS.B #DCO2+DCO1+DCO0,&DCOCTL ; Select max DCO tap
```

5.2.1 低功耗应用的基本时钟模块+ 的特性

电池供电应用中通常存在相互矛盾的要求。

- 低时钟频率，以节约能源和测时
- 针对快速对事件做出反应及快速突发处理能力的高时钟频率。
- 运行温度和电源电压上的时钟稳定

基本时钟模块+通过允许用户从三个可用的时钟信号中做出选择来解决上述相互矛盾的要求：ACLK，MCLK 和 SMCLK。 对于理想的低功耗模式，ACLK 来自一个低功耗的 32768Hz 时钟晶振（如果可用的话），为系统和低功耗操作提供一个稳定的时钟基础，当晶振的精确度时间保持不被要求时，或可来自内部低频振荡器。 当请求中断驱动事件发时，为了由可激活的片上 DCO 运行，MCLK 可被配置。 为了由一个晶振或 DCO 运行，SMCLK可被配置，这取决于外围设备要求。 可提供一个灵活的时钟分配和分频系统来对各个时钟要求进行微调。

5.2.2 内部超低功耗低频振荡器 (VLO)

内部超低功耗、低频率振荡器 (VLO) 不需要一个晶振就能提供12kHz（有关参数请参阅《特定器件的数据手册》）的典型频率。 当 XTS = 0 时，可通过设置 LFXT1Sx = 10 来选定 VLOCLK源。 OSCOFF 位禁用 LPM4 的 VLO。当 VLO 被要求减小电流消耗时，LFXT1 晶体振荡器被关闭。VLO 在不使用时不消耗功率。

为了把VLO 用作 ACLK，应该配置没有 LFXT1（例如，MSP430G22x0）的器件。

5.2.3 LFXT1 振荡器

在 MSP430G22x0 器件系列中没有执行 LFXT1 振荡器。

LFXT1 振荡器通过在 LF 模式下 (XTS=0) 采用 32768Hz 时钟晶振来支持极小电流消耗。 一个时钟晶振连接到 XIN 和 XOUT 不需要任何其他外部组件。 在 LF 模式中，可选的软件 XCAPx 位为 LFXT1晶振配置内部提供的负载电容。 这个电容值可以被选作 1pF、6pF、10pF、或典型z值12.5pF。 必要的话还可添加额外的外部电容。

当在 HF 模式(XTS=1) 时，LFXT1 振荡器还支持高速晶振或者振荡器。 高速晶振或振荡器可接到 XIN 和 XOUT 端,且两端都需外部电容。 应当根据晶振或振荡器来决定电容的大小。 当 LFXT1 处于 HF 模式时，LFXT1Sx 位选择操作的范围。

当 LFXT1Sx＝11，OSCOFF=0 且 XCAPx = 00 时，可以把 LFXT1 与在 XIN 引脚上的处于 HF 或 LF 模式的外部时钟信号一起使用。 当与外部时钟信号一起使用时，外部频率必须满足已选模式的数据手册参数。当输入频率低于指定的最低限时，为了防止 CPU 采用被 LFXT1CLK 计时，必须置位 LFXT1OF 位。

如果 LFXT1CLK 不来自 SMCLK 或MCLK，软件可以通过设置 OSCOFF 来禁用 LFXT1，如图 5-3中所示。

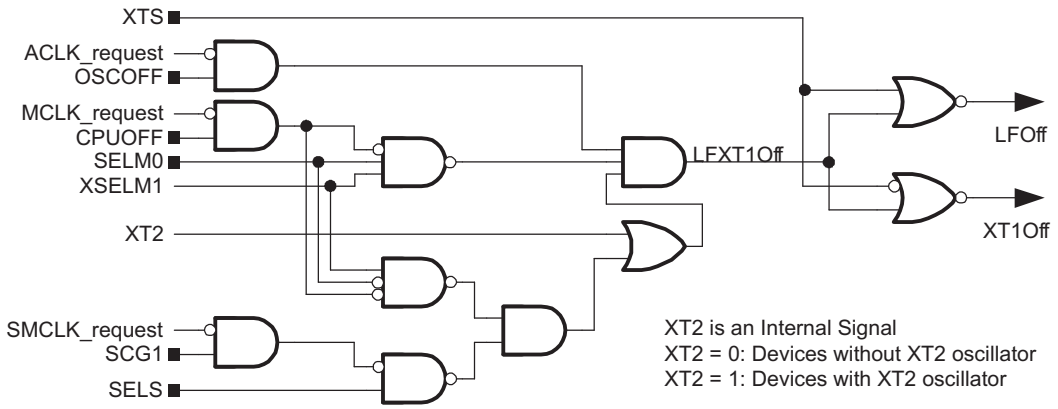


图 **5-3. LFXT1** 振荡器的关闭信号

注: **LFXT1 振荡器的特性**

低频率晶振通常需要几百毫秒的来启动，具体取决于晶振。

超低功耗振荡器如 LFXT1 在 LF 模式下应当远离其他来源的耦合的噪音干扰。 通过晶振外壳接地且用接地走线保护晶振走线来使晶振应尽量靠近 MSP430。

5.2.4 XT2 振荡器

有些器件有一个第二晶振振荡器，XT2。 XT2 来源于 XT2CLK，且特性和处于 HF 模式的 LFXT1 是相同的。 XT2Sx 位选择 XT2的操作范围。 如果 XT2CLK 没有被用作 MCLK 或 SMCLK 的时钟源，则 XT2OFF 位会禁用 XT2 振荡器，如图 5-4所示。

当 XT2Sx＝11 且 XT2OFF＝0 时，XT2 可以与 XT2IN 引脚上的外部时钟信号一起使用。当与外部信号一起使用时，外部频率必须满足 XT2 数据手册中的参数。 当输入频率低于指定的最低值时，为了防止 CPU 采用 XT2CLK 作为时钟，可对 XT2OF 进行置位。

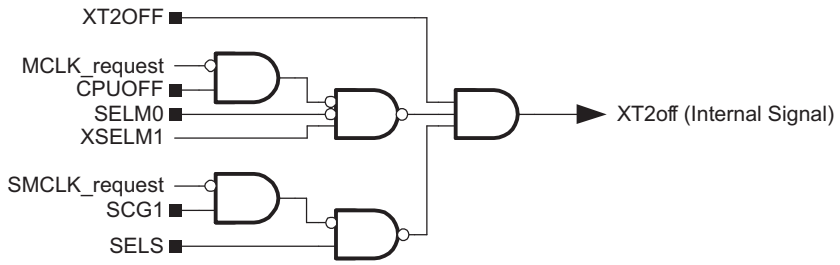


图 **5-4. XT2** 振荡器的关闭信号

5.2.5 数控振荡器(DCO)

DCO 是一个内置的数字控制振荡器。 DCO 的频率可通过软件使用 DCOx，MODx，和 RSELx 位来调节。

5.2.5.1 禁用 DCO

当 DCOCLK 在活动模式中没被用作 MCLK 或SMCLK 信号源时，可通过设置 SCG0 用软件使能 DCOCLK，如在图 5-5中所示。

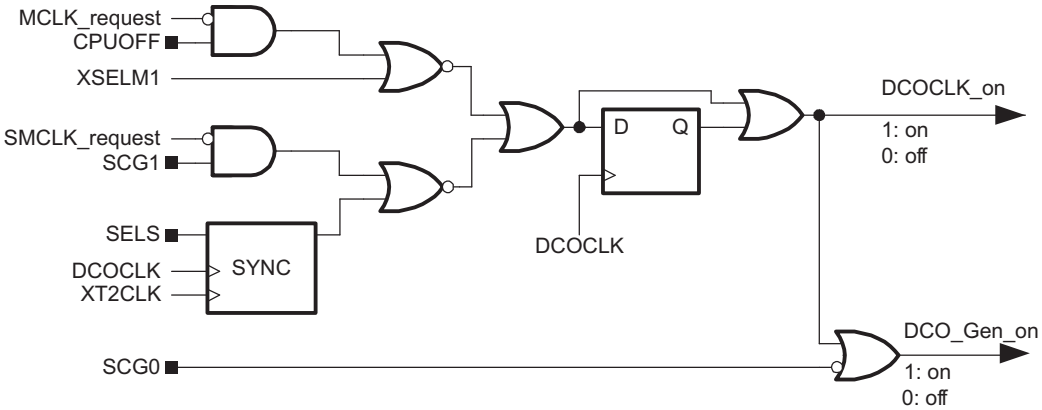


图 **5-5. DCO** 的开*/*关控制

5.2.5.2 调整 DCO 的频率

在一个 PUC 之后，RSELx = 7 且 DCOx = 3，可允许 DCO 在一个中段频率启动。 MCLK 和 SMCLK 都来自 DCO。 因为 CPU 执行来自 MCLK 的编码，MCLK来源于快速启动的 DCO，代码通常从 PUC 不到 2μs 时开始执行。图 5-6给出了典型的 DCOx 和 RSELx 的范围和阶跃。

COCLK 的频率根据以下功能设定：

- 四个 SELx 位选择 DCO 的 16 个标称频率范围其中的一个。在特定器件的数据手册中针对一个单独器件的对这些范围进行了定义。
- 3 个 DCOX 位把由 RSELx 位选择的 DCO的范围分频成 8 个频率阶跃，大约 10% 分频。
- 5 个 MODx 位，在由 DCOX 位选择的频率和下一个由 DCOX+1 设置的更高的频率之间的切换。 当 DCOx＝07H 时，由于DCOx 已经处于已选 RSELx 范围的最高设置，因此 MODx 位无效。

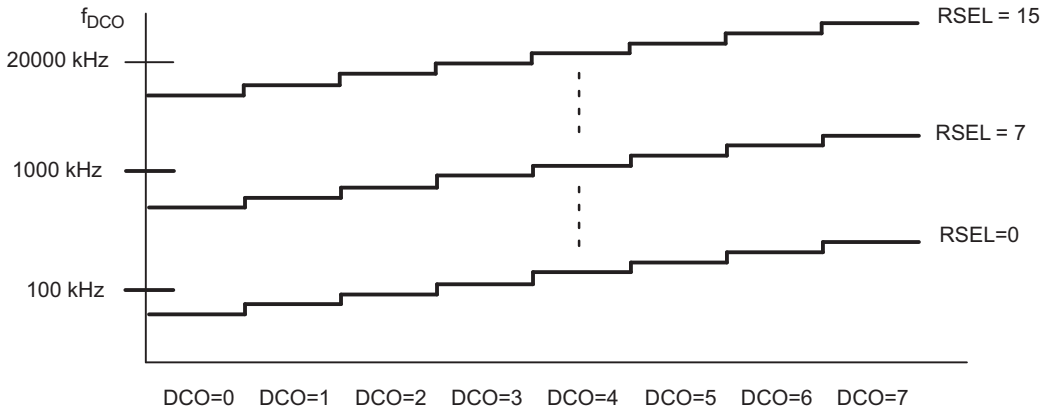


图 5-6. 典型的 DCOX 范围和 RSELx 的阶跃

每一个 MSP430F2xx 器件（以及大多数 MSP430G2xx 器件；请参阅《特定器件的数据手册》已为存放在信息存储段 A 中的特殊频率校准了 DCOCTL 和 BCSCTL1 寄存器的设置。为了使用校准设置，信息被复制到 DCOCTL和 BCSCTL1寄存器中。 已校准设置会影响DCOx，MODx，和 RSELx 位，除了 XT2OFF 位保持设置外，所有其它位都被清零。 剩余的 BCSCTL1 位能够根据需要通过 BIS.B 或 BIC.B 指令来置位或清零。

```
; Set DCO to 1 MHz:CLR.B &DCOCTL ; Select lowest DCOx; and MODx settingsMOV.B
&CALBC1_1MHZ,&BCSCTL1 ; Set rangeMOV.B &CALDCO_1MHZ,&DCOCTL ; Set DCO step + modulation
```

5.2.5.3 一个外部电阻(R_{osc}) 用于 DCO

当 DCOR=1 时，一些 MSP430F2xx 器件通过一个外部电阻，，R_{OSC}，接到 DV_{CC}来为 DCO 电流源提供选择。在这种情况下，DCO 与 MSP430x1xx 系列具有相同的特性，除 RSEL3 被忽略外，RSELX 设置被限定为 0～7。 为了通过改变电阻值来调整 DCO的频率，这个选择为此提供了另一种方法。 有关参数请参阅《特殊器件数据手册》。

5.2.6 DCO 调制器

为了 f_{DCO}和f_{DCO+1}能在f_{DCO}和 f_{DCO+1} 之间产生一个有效中段频率并发出时钟能量，减小电磁干扰 (EMI)，该调节器混合了两种 DCO 频率。 该调制器为 32 个 DCOCLK时钟周期混合了 f_{DCO}和f_{DCO+1}，且是用 MODx 位配置的。当MODx = 0 时，该调制器关闭。

调制器混频公式为：

$$t = (32\text{-}MODx) \times t_{DCO} + MODx \times t_{DCO+1}$$

由于 f_{DCO}低于有效频率而f_{DCO+1}高于有效频率，所以有效频率误差的积分为零。 它不会累积。 有效频率的误差每 32 个 DCOCLK就归零。图 5-7说明了调节器操作。

调节器设置和 DCO 控制都是用软件配置的。 DCOCLK 可以和一个已知值的稳定频率进行对比，并用 DCOx，RSELx 和 MODx 位进行调节。 有关应用说明和 DCO 配置代码示例请参见http://www.msp430.com。

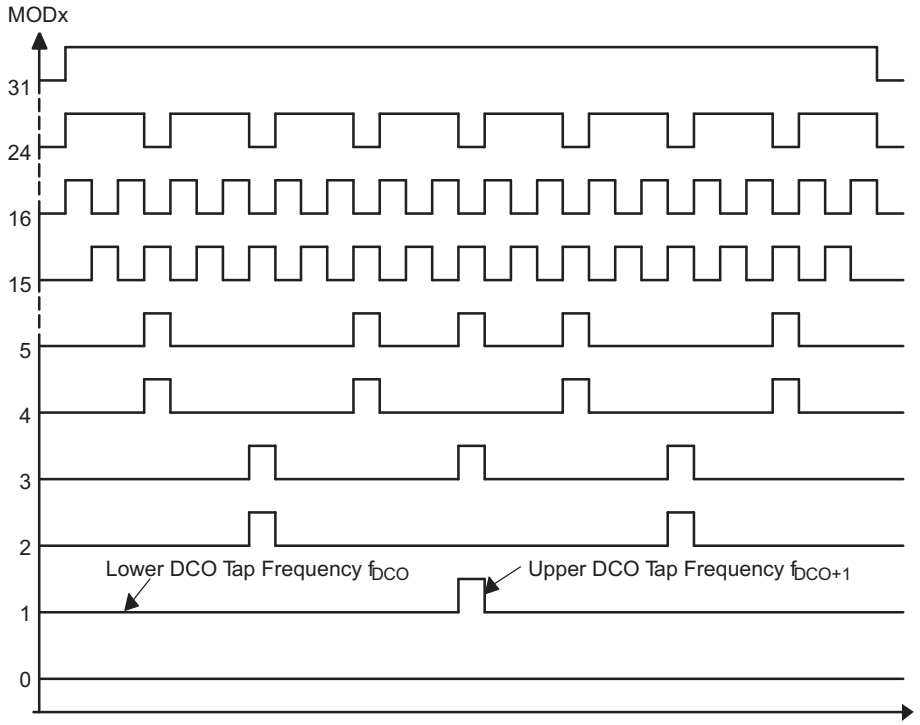


图 **5-7.** 调制器模式

5.2.7 基本时钟模块+ 的故障安全操作

基本时钟模块+ 集成了一个振荡器故障的故障安全功能。 该功能能够检测到 LFXT1 和 XT2 的振荡器故障，如在图 5-8中所示。 可能失效的情况有：

- LF 模式下 LFXT1 的低频振荡器故障 (LFXT1OF)
- HF 模式下 LFXT1 的高频振荡器故障 (LFXT1OF)
- XT2 的高频振荡器故障 (XT2OF)

如果相应的晶振振荡器打开且操作不当时，晶振振荡器故障位 LFXT1OF，和 XT2OF 就会被置位。 只要故障条件存在故障位就保持置位，直到使能振荡器得到正常操作才自动清零。

当测试到振荡器故障(LFXT1OF 或 XT2OF) 时，OFIFG 振荡器故障标志被置位，并且锁定到 POR。 当 OFIFG 被置位时，MCLK 以 DCO 为源，且如果 OFIE 被置位，则 OFIFG 会请求一个不可屏蔽 (NMI) 中断。 当中断得到响应，OFIE 会自动复位。 必须由软件清零 OFIFG 标志。 可以通过测试各个故障位确定失效源。

假如在以 MCLK 为源的晶振振荡器中检测到故障，则 MCLK会被自动切换到以 DCO 为时钟源。 这不会改变 SELMx 位的设置。 这种情况由用户软件操作。

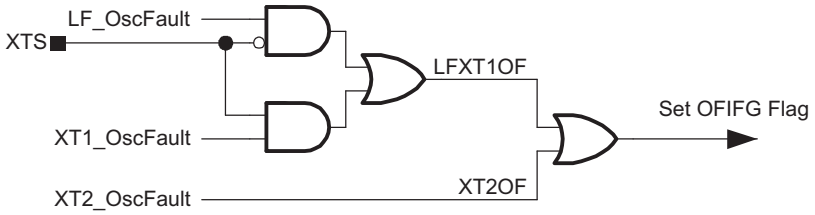


图 **5-8.** 振荡器故障逻辑

5.2.7.1 MCLK 以晶振为时钟源

一次 PUC 之后，基本时钟模块+ 将 DCOCLK 用于 MCLK。 如果需要，MCLK 也可以来自 LFXT1 或 XT2 - 如果可用的话。

把 MCLK 的源从 DCO 时钟转换成晶振时钟(LFXT1CLK 或 XT2CLK) 的顺序是：

1. 打开晶体振荡器并选择合适的模式
2. 清零 OFIFG 标志
3. 等待至少 50uS
4. 测试 OFIFG，并重复 2 至 4 的步骤，直到 OFIFG 保持被清零。

```
; Select LFXT1 (HF mode) for MCLKBIC.W #OSCOFF,SR ; Turn on osc.BIS.B #XTS,&BCSCTL1 ; HF
modeMOV.B #LFXT1S0,&BCSCTL3 ; 1-
3MHz CrystalL1 BIC.B #OFIFG,&IFG1 ; Clear OFIFGMOV.W #0FFh,R15 ; DelayL2 DEC.W R15 ; JNZ L2
;BIT.B #OFIFG,&IFG1 ; Re-
test OFIFGJNZ L1 ; Repeat test if neededBIS.B #SELM1+SELM0,&BCSCTL2 ; Select LFXT1CLK
```

5.2.8 时钟信号的同步

当把 MCLK 或 SMCLK 从一个时钟源切换到另一个时，为了避免临界竞争条件，该转换必须同步，如在图 5-9中所示。

- 当前时钟周期持续到下一个上升沿。
- 时钟保持高电平直到新时钟的下一个上升沿。
- 新时钟源被选择并且持续全高电平一段时间。

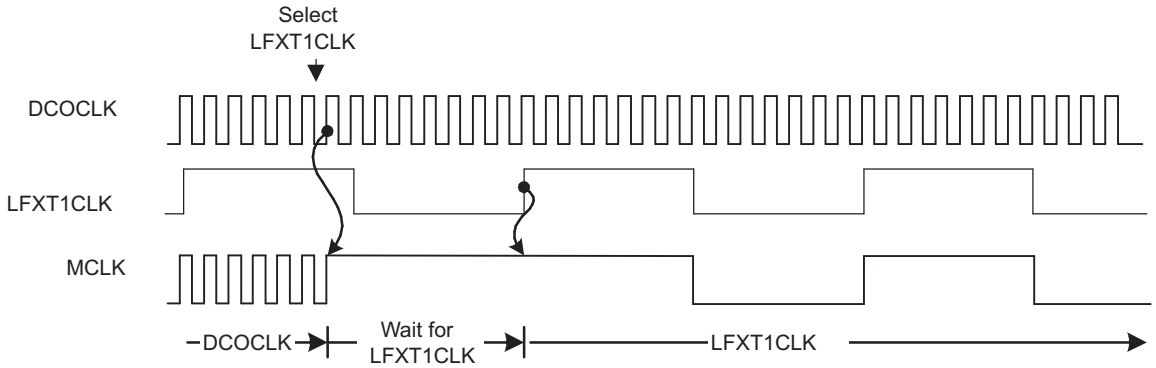


图 **5-9.** 把 **MCLK** 从 **DCOCLK** 切换至 **LFXT1CLK**

5.3 基本时钟模块+ 寄存器

在表 5-1中列出了基本时钟模块+ 寄存器。

表 5-1. 基本时钟模块+寄存器

寄存器	简表	寄存器类型	地址	初始化状态
DCO 控制寄存器	DCOCTL	读取/写入	056h	060h 与 PUC
基本时钟系统控制 1	BCSCTL1	读取/写入	057h	087h 与 POR(1)
基本时钟系统控制 2	BCSCTL2	读取/写入	058h	用 PUC 复位
基本时钟系统控制 3	BCSCTL3	读取/写入	053h	005h 与 PUC(2)
SFR 中断使能寄存器 1	IE1	读取/写入	000h	用 PUC 复位
SFR 中断标志寄存器 1	IFG1	读取/写入	002h	用 PUC 复位

(1) 一些寄存器位也被 PUC 初始化（请参见5.3.2 节）。
(2) 在 MSP430AFE2xx器件中 BCSCTL3 的初始状态是 000h。

5.3.1 DCOCTL，DCO 控制寄存器

7	6	5	4	3	2	1	0
DCOx			MODx				
rw-0	rw-1	rw-1	rw-0	rw-0	rw-0	rw-0	rw-0

DCOx	位 7-5	DCO 频率选择。 这些位选择是 RSELx 设置的界定范围中的八个离散 DCO 频率中的一个。
MODx	位 4-0	调制器选择。 这几位决定在一个 32 个 DCOCLK 周期内 f_{DCO+1}频率被用的次数。 在持续的时钟周期内 (32-MOD)，使用了 f_{DCO}频率。 当 DCOx=7时 不被采用。

5.3.2 BCSCTL1，基础时钟系统控制寄存器 1

7	6	5	4	3	2	1	0
XT2OFF	XTS[1][2]	DIVAx		RSELx			
rw-(1)	rw-(0)	rw-(0)	rw-(0)	rw-0	rw-1	rw-1	rw-1

XT2OFF	位 7	XT2 关闭。 此位关闭了 XT2 振荡器 0 XT2 被开启 1 如果不使用 MCLK 或 SMCLK 的话，XT2 就会被关闭。
XTS	位 6	LFXT1 模式选择。 0 低频模式 1 高频模式
DIVAx	位 5-4	ACLK 分压器 00 /1 01 /2 10 /4 11 /8
RSELx	位 3-0	范围选择。 16 个不同的频率范围可用。 通过设置 RSELx= 0 来选择最低频率范围。当 DCOR= 1 时 RSEL3 被忽略。

(1) 在 MSP430x20xx 和 MSP430G2xx器件中不支持 XTS = 1（有关所有器件所支持设置的详细信息，请参阅图 5-1和图 5-2）。
(2) 在 MSP430AFE2xx器件中保留了该位。

5.3.3 BCSCTL2，基础时钟系统控制寄存器 2

7	6	5	4	3	2	1	0
SELMx		DIVMx		SELS	DIVSx		DCOR(1)(2)
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

SELMx 位 7-6 选择 MCLK。 这些位选择 MCLK 源。
- 00 DCOCLK
- 01 DCOCLK
- 10 当片上有 XT2 振荡器时，XT2CLK。 当片上没有XT2 振荡器时，LFXT1CLK 或 VLOCLK。
- 11 LFXT1CLK或 VLOCLK

DIVMx 位 5-4 MCLK 分压器
- 00 /1
- 01 /2
- 10 /4
- 11 /8

SELS 位 3 选择 SMCLK。 该位选择 SMCLK 的来源。
- 0 DCOCLK
- 1 当有 XT2 振荡器时，XT2CLK。 当没有 XT2 振荡器时，LFXT1CLK 或 VLOCLK。

DIVSx 位 2-1 SMCLK 分压器
- 00 /1
- 01 /2
- 10 /4
- 11 /8

DCOR 位 0 DCO 电阻选择。 不适用于所有设备。 请参阅《特定器件数据手册》。
- 0 内部电阻
- 1 外部电阻

(1) 不适用于 MSP430x20xx 或 MSP430x21xx器件。
(2) 在 MSP430AFE2xx器件中保留了该位。

5.3.4 BCSCTL3，基础时钟系统控制寄存器 3

7	6	5	4	3	2	1	0
XT2Sx		LFXT1Sx(1)		XCAPx(2)		XT2OF(3)	LFXT1OF(2)
rw-0	rw-0	rw-0	rw-0	rw-0	rw-1	r0	r-(1)

XT2Sx 位7-6 XT2 范围选择。 这些位为 XT2 选择频率范围。

00 0.4 至 1MHz 的晶振或谐振器

01 1 至 13MHz 的晶振或谐振器

10 3 至 16MHz 的晶振或谐振器

11 数字化外部 0.4 至 16MHz 的时钟源

LFXT1Sx 位5-4 低频率时钟选择和 LFXT1 范围选择。 当 XTS = 0 时， 这些位在 LFXT1和 VLO 之间选择，且当 XTS = 1 时，会选择 LFXT1 频率范围。

当 XTS = 0 时：

00 LFXT1 上的 32768Hz 晶振

01 被保留

10 VLOCLK（保留在 MSP430F21x1 器件中）

11 数字外部时钟源

当 XTS =1 时（不适用于 MSP430x20xx器件，MSP430G2xx1/2/3）

00 0.4 至 1MHz 的晶振或谐振器

01 1 至 13MHz 的晶振或谐振器

10 3 至 16MHz 的晶振或谐振器

11 数字化外部 0.4 至 16MHz 的时钟源

MSP430AFE2xx器件的 LFXT1Sx 定义：

00 被保留

01 被保留

10 VLOCLK

11 被保留

XCAPx 位3-2 振荡器的电容选择。 当 XTS = 0 时，这些位选择有效电容，请参见 LFXT1 晶振。如果 XTS =1 或 LFXT1Sx= 11， XCAPx 应该是 00。

00 ~1pF

01 ~6pF

10 ~10pF

11 ~12.5pF

XT2OF 位 1 XT2 振荡器失效

0 不存在失效条件

1 存在失效条件

LFXT1OF 位 0 LFXT1 振荡器失效

0 不存在失效条件

1 存在失效条件

(1) MSP430G22x0：在初始化和启动代码选择 VLOCLK 期间，LFXT1Sx 位应该被编程为 10b（更多信息请参阅数字 I/O章节）。 其它位被保留且不应该改变。

(2) 在 MSP430AFE2xx器件中保留了该位。

(3) 不适用于 MSP430x2xx，MSP430x21xx，或MSP430x22xx 器件。

5.3.5 IE1，中断使能寄存器 1

7	6	5	4	3	2	1	0
						OFIE[1]	
						rw-0	

	位 7-2	这些位可以用于其他模块。 请参阅《特定器件数据手册》。
OFIE	位 1	振荡器故障中断使能. 该位使能 OFIFG 中断。 因为在 IE1 中其他位可用于其他模块，故建议使用 BIS.B或 BIC.B 指令，而非 MOV.B 或 CLR.B 指令来置位或清零此位。 0 中断未被启用 1 中断被启用
	位 0	该位可用于其他模块。 请参阅《特定器件数据手册》。

(1) MSP430G22x0：该位不应被置位。

5.3.6 IFG1，中断标志寄存器 1

7	6	5	4	3	2	1	0
						OFIFG[1]	
						rw-1	

	位 7-2	这些位可用于其他模块。 请参阅《特定器件数据手册》。
OFIFG	位 1	振荡器故障中断标志。 因为在 IFG1 中其他位可用于其他模块，故建议使用 BIS.B或 BIC.B 指令，而非MOV.B 或 CLR.B 指令来置位或清零此位。 0 无中断等待 1 中断等待
	位 0	该位可用于其他模块。 请参阅《特定器件数据手册》。

(1) MSP430G22x0：LFXT1 振荡器的引脚在这个器件中不可用。 振荡器故障标志将始终由硬件置位。 不应设置中断使能位。

DMA 控制器

在无需 CPU 干预的情况下，DMA 控制器模块可将数据从一个地址移动到另外一个地址。 这一章将介绍 MSP430x5xx 器件系列中 DMA 控制器的操作。

Topic **Page**

6.1 DMA 介绍

直接存储器存取 (DMA) 控制器可以在全部地址范围内把数据从一个地址传输到另外一个地址，而无须 CPU 干预。 例如，DMA 控制器可以把数据从 ADC12 转换存储器中直接传输到 RAM 中。

包含一个 DMA 控制器的器件可能有一个，两个，或三个可用的 DMA 通道。 因此，根据 DMA 通道数量的不同，在这一章中有些特性并不对适用于所有器件。

通过使用 DMA 控制器可增加外设模块的吞吐量。 通过使 CPU 保持在睡眠模式，而无需将其唤醒来从一个外设中移动数据，它也会减少系统功耗。

DMA 控制器的功能包括：

- 多达 8 个独立的传输通道
- 可配置的 DMA 通道的优先级
- 每次传输仅需要两个 MCLK 时钟周期
- 字节或字和混合字/字节传输能力
- 字区大小高达 65536 字节或字
- 可配置的传输触发选择
- 可选择的边沿或电平触发传输
- 四种寻址方式
- 单次，块，或突发块传输模式

在图 6-1中给出了 DMA 控制器的结构图。

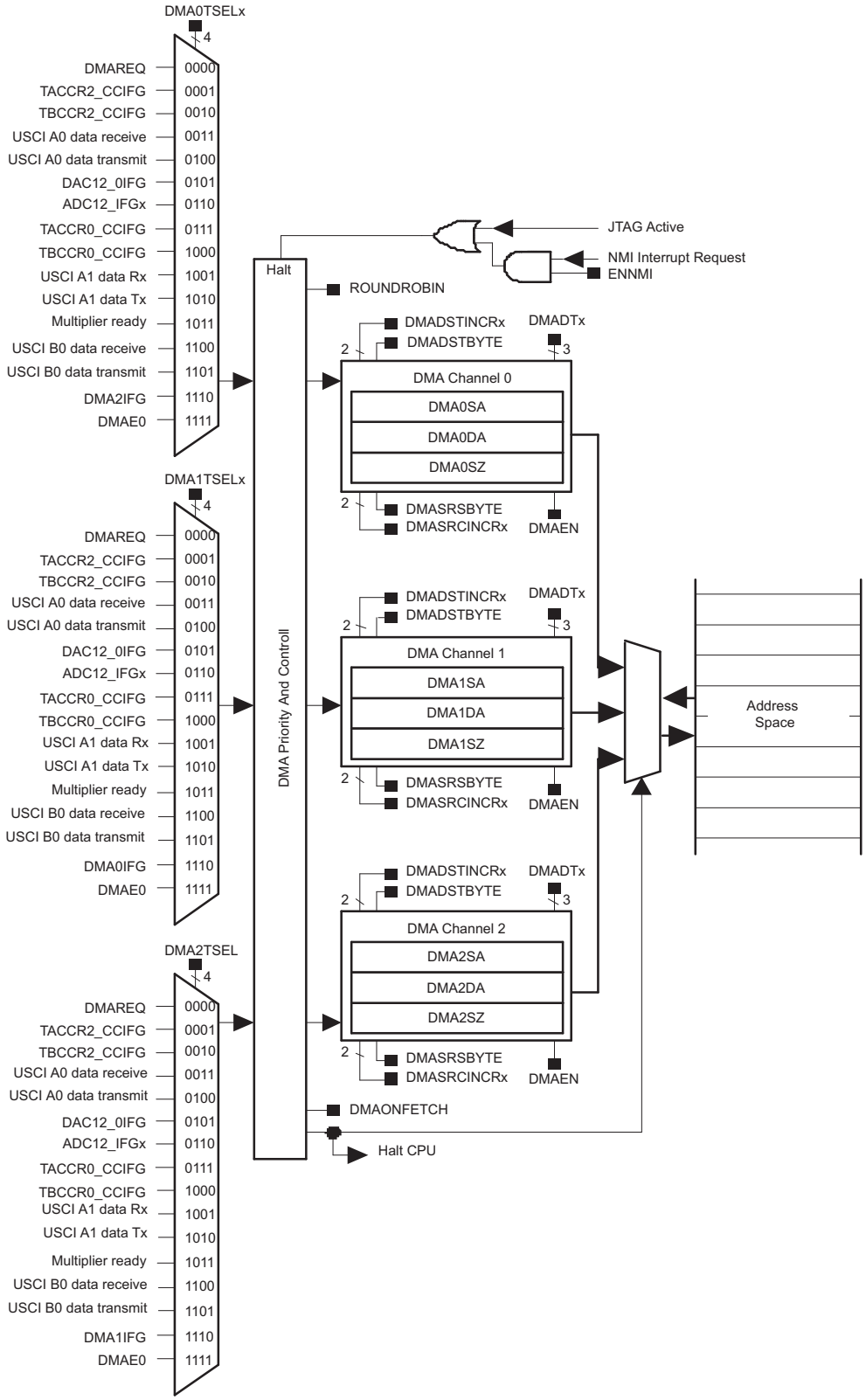


图 **6-1. CAN** 控制器结构图

6.2 DMA 操作

DMA 控制器由用户软件配置。 DMA 的建立和操作将在下面的部分进行讨论。

6.2.1 DMA 寻址模式

DMA 控制器有四种寻址模式。 对于每个 DMA 通道的寻址模式都是可独立可配置的。 例如，通道 0 可以在两个固定的地址间传输，而通道 1 可在地址的两个块间传输。 在图 6-2中给出了寻址模式。 这些寻址模式是：

- 固定的地址到固定的地址
- 固定的地址到地址块
- 地址块到固定的地址
- 地址块到地址块

寻址方式由 DMASRCINCRx 和 DMADSTINCRx 控制位配置。 DMASRCINCRx 位选择在每次传输结束后源地址是否不变、增加还是减少。 DMADSTINCRx 位选择在每次传输结束后目标地址是否不变、增加还是减少。

传输可以是字节到字节、字到字、字节到字、或字到字节。 当字到字节传输时，只有源字节的低字节会被传输。 当是字节到字传输时，目标字的高字节将会在传输发生的时候被清除。

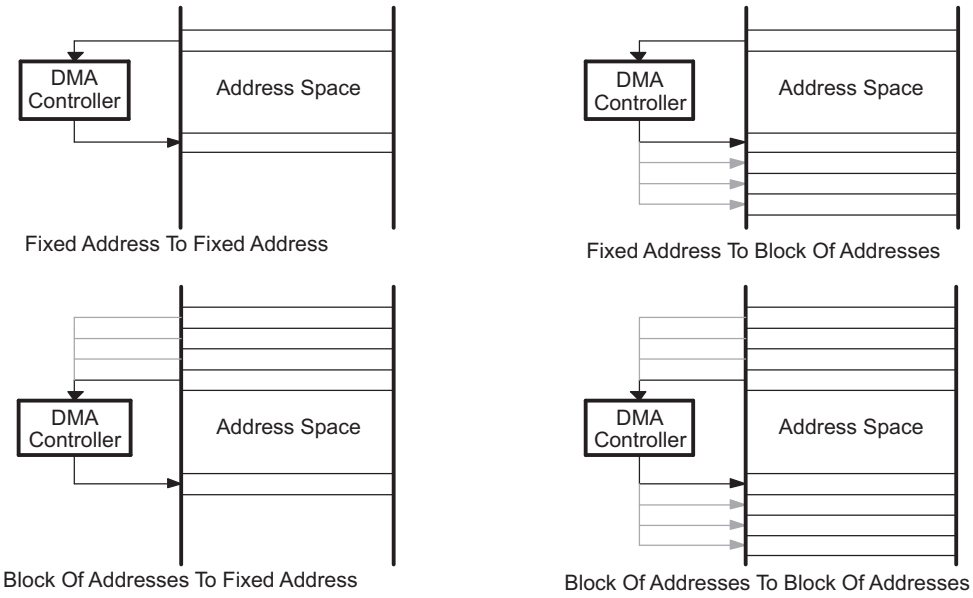


图 6-2. DMA 寻址模式

6.2.2 DMA 传输模式

如在表 6-1中所给，DMA 控制器有六种传输模式，这些模式都由 DMADTx 位选择。 每个通道都可以独立配置其传输模式。 例如，可在单次传输模式下配置通道 0，而通道 1 可以配置为突发块传输模式，且通道 2 可在重复块模式下操作。 传输模式独立于寻址模式进行配置。 任何寻址模式都可以使用任何传输模式。

由 DMAxCTL DSTBYTE 和 SRCBYTE 区域选择的数据的两种类型可以被传输。 源和/或目标位置都可以是字或字数据。 它也可以在字节到字节、字到字或任何组合之间的进行传输。

表 6-1. DMA 传输模式

DMADTx	传输模式	说明
000	单次传输	每次传输都需要一个单独的触发。 当 DMAxSZ 传输已经生成时 DMAEN 会被自动清零。
001	块传输	一个整块将会在一个触发后传输。 在块传输结束时 DMAEN 会被自动清零。
010, 011	突发块传输	CPU 操作与块传输交叉进行。 DMAEN 位会在突发块传输结束时自动清零。
100	重复单次传输	每次传输需要一个触发。 DMAEN 保持被启用。
101	重复块传输	一个完整块传输需要一个触发。 DMAEN 保持被启用。
110, 111	重复突发块传输	CPU 操作与块传输交叉进行。 DMAEN 保持被启用。

6.2.2.1 单次传输

在单次传输模式中，每字节/字的传输都需要一个单独的触发。 在图 6-3中给出了单次传输状态图。

DMAxSZ 寄存器用来定义每次传输的数目。 DMADSTINCRx 和 DMASRCINCRx 位用来选择在每次传输结束后目标地址和源地址是否增加或减少。 如果 DMAxSZ=0，则没有传输发生。

DMAxSA，DMAxDA，和 DMAxSZ 寄存器都会被复制到临时寄存器中。 在每次传输结束后，DMAxSA 和 DMAxDA 的临时值都会增加或者减少。 在每次传输结束后 DMAxSZ 寄存器中的值会减少。 当 DMAxSZ 寄存器的值减少至 0 时，将会从其临时寄存器中重载并且相应的 DMAIFG 标志将会置位。 当 DMADTx=0 时，DMAEN 位将会被自动清零，当 DMAxSZ 减至 0 时必须为下一次传输的产生重新设置。

在重复的单次传输模式中，DMAEN=1 时 DMA 控制器保持被启用，每当一个触发后就会发生一个传输。

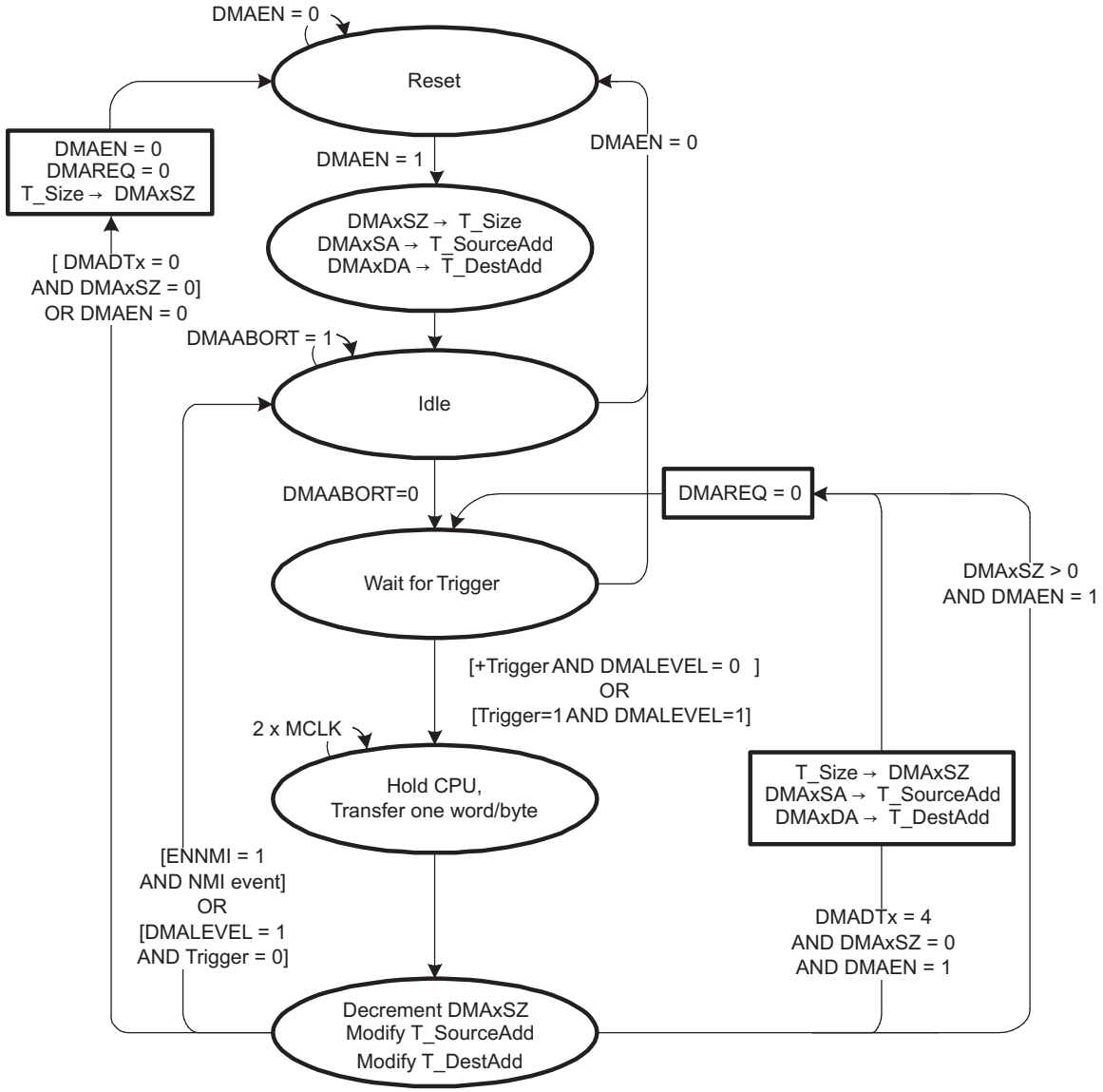


图 **6-3. DMA** 单次传输的状态图

6.2.2.2 块传输

在块传输模式中，数据的一个整块的一个传输将会在一个触发后开始传输。 当 DMADTx=1 时，在块传输结束后 DMAEN 位将会被清零并需在另一个块传输被触发前重新置位。 在一个块传输被触发后，在块传输的过程中接下来的触发信号将会被忽略。 在图 6-4中给出了块传的输状态图。

DMAxSZ 寄存器用来定义块的大小，DMADSTINCRx 和 DMASRCINCRx 位用来选择在每次块传输结束后目标地址和源地址是否增加或减少。 如果 DMAxSZ=0，则没有块传输发生。

DMAxSA，DMAxDA，和 DMAxSZ 寄存器都会被复制到临时寄存器中。 在每次块传输结束后，DMAxSA 和 DMAxDA 的临时值都会增加或者减少。 在每次块传输结束后 DMAxSZ 寄存器中的值会减少并且会指示块中还剩余多少数据。 当 DMAxSZ 寄存器的值减少至 0 时将会从其临时寄存器中重载并且相应的 DMAIFG 标志将被置位。

在一个块传输中，在块传输完成时 CPU 将会暂停。 块传输将用 2 x MCLK x DMAxSZ 个时钟周期来完成。在块传输完成后 CPU 将会以其先前的状态继续执行。

在重复块传输模式中，在每个块传输完成后 DMAEN 位将保持置位。 在一个重复块传输的一个完成后，下一个触发将触发另一个块传输。

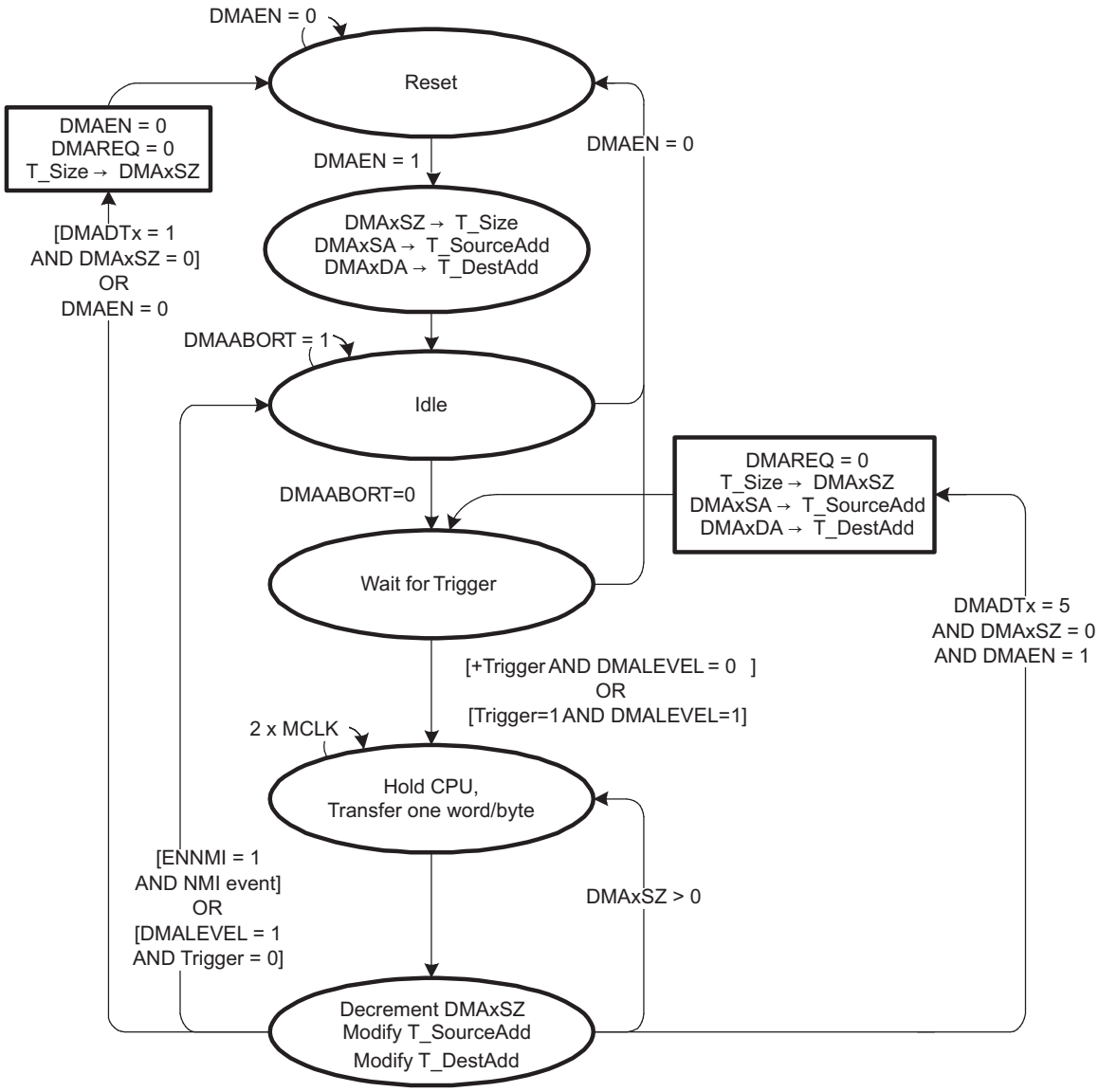


图 **6-4. DMA** 块传输的状态图

6.2.2.3 突发块传输

在突发模式中，传输是在 CPU 交叉存取下的块传输。 在每个块的四个字节/字传输后，CPU 将运行 2 个 MCLK 周期，如此导致了 20% 的 CPU 运行容量。 在突发块之后，CPU 将会在 100% 的容量下运行并且 DMAEN 位将被清零。 在另一个突发块传输被触发前 DMAEN 位必须重新置位。 在一个突发块传输被触发后，在突发块传输期间，接下来的触发信号将会被忽略。 在图 6-5中给出了突发块传输的状态图。

DMAxSZ 寄存器用来定义块的大小，且 DMADSTINCRx 和 DMASRCINCRx 用来选择在每次块传输后目标地址和源地址是否增加或者减少。 如果 DMAxSZ=0，则没有传输发生。

DMAxSA，DMAxDA，和 DMAxSZ 寄存器都会被复制到临时寄存器中。 在每次块传输结束后 DMAxSA 和 DMAxDA 的临时值都会增加或者减少。 在每次块传输结束后 DMAxSZ 寄存器中的值会减少并且会指示块中还剩余多少数据。 当 DMAxSZ 寄存器的值减少至 0 时将会从其临时寄存器中重载并且相应的 DMAIFG 标志将被置位。

在重复突发块模式中，在突发块传输完成后，DMAEN 位将保持置位且不再需要额外的触发信号来启动另一次突发块传输。 另一个突发块传输将在一个突发块传输完成后立即开始。 在这种情况下，通过清零 DMAEN 位必须停止传输，或当 ENNMI 被置位时由一个 NMI 中断引起。 在重复突发块模式中，CPU 持续在 20% 的容量运行直到重复突发块传输停止。

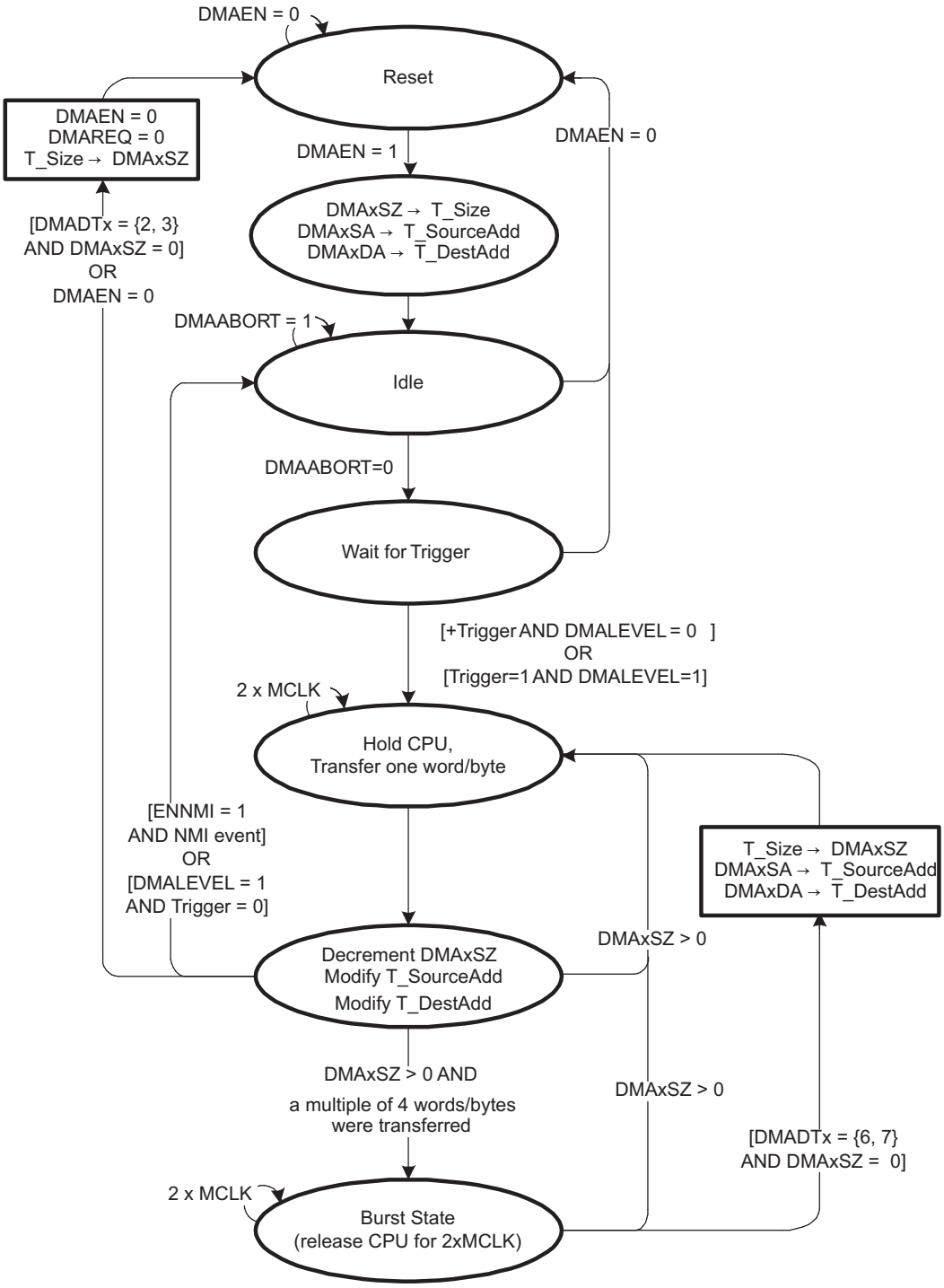


图 **6-5. DMA** 突发块传输的状态图

6.2.3 初始化 DMA 传输

每个 DMA 通道都可以独立的由 DMAxTSELx 位配置为自身的触发源，如在表 6-2中所述。 应该在 DMACTLx DMAEN 位为 0 时修改 DMAxTSELx 位。否则，不确定的 DMA 触发或许会发生。

当选择触发时，必须确保触发还没有发生，或传输将不会发生。 例如，如果 TACCR2 CCIFG 位被选作一个触发，并且它已经被置位时，将不会发生转移直到下一次 TACCR2 CCIFG 位被置位。

6.2.3.1 边沿触发的触发器

当 DMALEVEL=0 时，边沿触发将被使用并且有触发信号的上升沿初始化该传输。 在单次传输模式中，每次传输都需要其自身的触发。 当使用块或者突发块模式时，仅需要一个触发来启动块或者突发块传输。

6.2.3.2 电平触发的触发器

当 DMALEVEL=1 时，使用电平触发的触发器。 为了适当的操作，电平触发的触发器仅用在当外部触发 DMAE0 被选做触发时。 只要触发源信号为高电平就会有 DMA 传输被触发并且 DMAEN 位保持置位。

为了块或突发块传输的完成，触发信号必须保持为高电平 在一个块或突发块传输时，如果触发信号变低，则 DMA 控制器将会保持在当前状态直到触发变高或者直到 DMA 寄存器被软件修改。 如果 DMA 寄存器没被软件修改，当触发信号再次变高时，传输将会恢复到触发信号变低的那个状态。

当 DMALEVEL=1 时，建议当 DMADTx = {0，1，2，3} 时选择传输模式时， 因为 DMAEN 位是在传输配置后自动置位的。

6.2.3.3 DMA传输的暂停执行执指令

DMAONFETCH 位控制何时 CPU 为一个 DMA 传输暂停时。 当 DMAONFETCH=0 时，CPU 立即被暂停且当接受到一个触发信号时传输开始。 当 DMAONFETCH=1 时，CPU 将会在 DMA 控制器暂停 CPU 和传输开始前完成当前的执行指令。

注: **把 DMA 写入闪存时必须使用 DMAONFETCH**

如果 DMA 控制器是用来写入闪存存储器的，那么必须置位 DMAONFETCH 位。 否则，可能会导致不可预知的操作。

表 6-2. DMA 触发操作

DMAxTSELx	运行
0000	当 DMAREQ 位被置位时一个传输被触发。 当该传输开始时 DMAREQ 位自动复位。
0001	当 DMAxIFG 标志置位时一个传输被触发。 当该传输开始时 TACCR2 CCIFG 标志自动复位。 如果 TACCR2 CCIE 位被置位，TACCR2 CCIFG 标志将不会触发一个传输。
0010	当 TBCCR2 CCIFG 标志置位时一个传输被触发。 当该传输开始时 TBCCR2 CCIFG 标志自动复位。 如果 TBCCR2 CCIE 位被置位，TBCCR2 CCIFG 标志将不会触发一个传输。
0011	当串行接口接收到新的数据时会触发一个传输。 USCI_A0 模块的器件：当 USCI_A0 收到新的数据时会触发一个传输 传输开始后 UCA0RXIFG 自动复位。 如果 UCA0RXIE 被置位, UCA0RXIFG 标志将不会触发一个传输。
0100	当串行接口准备好传输一个新的数据时会触发一个传输。 USCI_A0 模块的器件：当 USCI_A0 准备好发送新的数据时会触发一个传输。 传输开始后 UCA0TXIFG 自动复位。 如果 UCA0TXIE 被置位, UCA0TXIFG 标志将不会触发一个传输。
0101	当 DAC12_0CTL DAC12IFG 标志被置位时会触发一个传输。 传输开始时 DAC12_0CTL DAC12IFG 标志会自动清零。 如果 DAC12_0CTL DAC12IE 位被置位，DAC12_0CTL DAC12IFG 标志将不会触发一个传输。
0110	用 ADC12IFGx 标志来触发一个传输。 当执行单通道转换后, 相应的 ADC12IFGx 会被触发。 如果用到序列转换, ADC12IFGx 在转换序列中的最后一次转换被触发。 在转换完成后传输被触发并且 ADC12IFGx 被置位。 软件设置 ADC12IFGx 不会触发一个传输。 当相关的 ADC12MEMx 寄存器被 DMA 控制器访问时，所有的 ADC12IFGx 标志会自动复位。

表 6-2. DMA 触发操作 (continued)

DMAxTSELx	运行
0111	当 TACCR0 CCIFG 标志被置位时一个传输被触发。 当该传输开始时 TACCR0 CCIFG 标志自动复位。 如果 TACCR0 CCIE 位被置位，TACCR0 CCIFG 标志将不会触发一个传输。
1000	当 TBCCR0 CCIFG 标志被置位时一个传输被触发。 当该传输开始时 TBCCR0 CCIFG 标志自动复位。 如果 TBCCR0 CCIE 位被置位，TBCCR0 CCIFG 标志将不会触发一个传输。
1001	当 UCA1RXIFG 标志被置位时一个传输被触发。 传输开始后 UCA1RXIFG 自动复位。 如果 URXIE1 被置位，UCA1RXIFG 标志将不会触发一个传输。
1010	当 UCA1TXIFG 标志被置位时一个传输被触发。 传输开始后 UCA1TXIFG 自动复位。 如果 UTXIE1 被置位，UCA1TXIFG 标志将不会触发一个传输。
1011	在硬件乘法器准备一个新的操作数时会触发一个传输。
1100	没有传输被触发。 USCI_B0 模块的器件：当 USCI_B0 收到新的数据时会触发一个传输。 传输开始后 UCB0RXIFG 自动复位。 如果 UCB0RXIE 被置位，UCB0RXIFG 标志将不会触发一个传输。
1101	没有传输被触发。 USCI_B0 模块的器件：当 USCI_B0 准备好发送新的数据时会触发一个传输。 传输开始后 UCB0TXIFG 自动复位。 如果 UCB0TXIE 被置位，UCB0TXIFG 标志将不会触发一个传输。
1110	当 DMAxIFG 标志被置位时一个传输被触发。 DMA0IFG 触发通道 1，DMA1IFG 触发通道 2，且 DMA2IFG 触发通道 0。当传输开始时没有 DMAxIFG 标志会自动复位。
1111	通过外部触发 DMAE0 来触发一个传输。

6.2.4 停止 DMA 传输

有两种方法可以停止正在进行的 DMA 传输：

- 如果 DMACTL1 寄存器的 ENNMI 位被置位，一个单次，块，或突发块传输可以被一个 NMI 中断所停止。
- 可以通过清零 DMAEN 位来停止一个突发块传输。

6.2.5 DMA 通道的优先级

默认的 DMA 通道优先级是 DMA0-DMA1-DMA2。 如果两个或三个触发同时发生或者挂起，拥有最高优先级的通道将会首先完成传输（单次，块或者突发块传输），然后是第二优先级的通道，最后是第三优先级的通道。 如果一个较高优先级的通道被触发，进行中的传输中将不会被暂停。 一直到进行中的传输完成后较高优先级的传输才开始。

DMA 通道的优先级由 ROUNDROBIN 位配置。 当 ROUNDROBIN 位被置位时，完成一个传输的通道的优先级会变为最低。 通道的优先级的*顺序*总保持相同，DMA0-DMA1-DMA2（请参阅表 6-3）。

表 **6-3.** 通道的优先级

DMA 优先级	传输发生	新的 **DMA** 优先级
DMA0 - DMA1 - DMA2	DMA1	DMA2 - DMA0 - DMA1
DMA2 - DMA0 - DMA1	DMA2	DMA0 - DMA1 - DMA2
DMA0 - DMA1 - DMA2	DMA0	DMA1 - DMA2 - DMA0

当 ROUNDROBIN 位被清零时，通道的优先级回到默认优先级。

6.2.6 DMA 传输周期

在每次单传输或者完整块或者突发块传输前 DMA 控制器需要一个或两个MCLK 时钟周期来同步。 同步后每个字节/字传输需要两个 MCLK 周期，且传输后有一个周期的等待时间。 因为 DMA 控制器使用 MCLK，所以 DMA 周期决定于 MSP430 的操作模式和时钟系统的设置。

如果 MCLK 源活动，但是 CPU 关闭，则 DMA 控制器将使用 MCLK 源来完成每次传输，而无需重新启用 CPU。 当 MCLK 源关闭时，DMA 控制器将临时重新开启 MCLK ，以 DCOCLK 为源，以便完成单次传输或者整块或者突发块传输。 且在传输完成后，CPU 保持关闭，MCLK 关闭。 各种操作模式下的最大 DMA 周期见表 6-4。

表 **6-4.** 最大单次传输 **DMA** 周期

CPU 操作模式	时钟源	最大 **DMA** 周期
激活模式	MCLK=DCOCLK	4 个 MCLK 周期
激活模式	MCLK=LFXT1CLK	4 个 MCLK 周期
低功耗模式 LPM0/1	MCLK=DCOCLK	5 个 MCLK 周期
低功耗模式 LPM3/4	MCLK=DCOCLK	5 个 MCLK 周期 + 6µs(1)
低功耗模式 LPM0/1	MCLK=LFXT1CLK	5 个 MCLK 周期
低功耗模式 LPM3	MCLK=LFXT1CLK	5 个 MCLK 周期
低功耗模式 LPM4	MCLK=LFXT1CLK	5 个 MCLK 周期 + 6µs(1)

(1) 额外的 6µs 用于需要启动 DCOCLK。 它是数据手册中的 $t_{(LPMx)}$参数。

6.2.7 使用带有系统中断的 DMA

DMA 传输不会被系统中断所打断。 系统中断将会保持挂起直到传输完成。 如果 ENNMI 位被置位的话，NMI 中断可以中断 DMA 控制器。

系统中断服务子程序将会被 DMA 传输打断。 如果系统中断服务子程序或者其他程序必须在没有中断的情况下运行，则 DMA 控制器必须在该子程序被执行前被禁止。

6.2.8 DMA 控制器中断

每个 DMA 通道都有自己的 DMAIFG 标志。 当相应的 DMAxSZ 寄存器计数到 0 时，每个 DMAIFG 标志都可以在任何模式下被设置。 如果相应的 DMAIE 和 GIE 位都被设置，则会产生一个中断请求。

所有的 DMAIFG 标志只源自 DMA 控制器中断向量且，在一些器件上，可以和其他模块分享该中断向量。进一步更多详细信息请参阅《特定器件的数据手册》。 对这些器件来说，软件必须检查 DMAIFG 和相应的模块标志来判断中断源。 DMAIFG 标志不会自动复位且必须由软件复位。

此外，一些器件使用 DMAIV 寄存器。 为了源自同一个中断向量源，所有 DMAIFG 标志都被优先级化，和正成为最高优先级的 DMA0IFG 一起，被结合在一起 。 最高优先级的被启用的中断产生在 DMAIV 寄存器中生成了一个数字。 为了自动进入相应的软件程序，可以对这个数字进行评估，或将其添加到程序计数器。禁用的 DMA 中断不影响 DMAIV 的值。

任何对 DMAIV 寄存器的访问，读取，或写入都将自动复位最高的正在挂起的中断标志。 如果另一个中断标志被置位，则另一个中断将会在最初的中断服务结束后立即产生。 例如，假设 DMA0 有最高的优先级。 如果 DMA0IFG 和 DMA2IFG 标志被置位，当中断服务子程序在访问 DMAIV 寄存器时，DMA0IFG 会自动复位。 在中断服务子程序执行完 RETI 指令后，DMA2IFG 将会生成另一个中断。

下面的软件示例是 DMAIV 和处理开销的推荐用法。 为了自动跳转到相应的程序，DMAIV 值将被添加到 PC。

在右边距的数字显示了每条指令所需的 CPU 周期。 不同中断源的这个软件开销包括中断响应和中断返回的周期，但不处理的任务本身。

Example 6-1. DMAIV 软件示例

```
;Interrupt handler for DMA0IFG, DMA1IFG, DMA2IFG CyclesDMA_HND ... ; Interrupt latency 6ADD &DMAIV,PC
; Add offset to Jump table 3RETI ; Vector 0: No interrupt 5JMP DMA0_HND ; Vector 2: DMA channel 0
2JMP DMA1_HND ; Vector 4: DMA channel 1 2JMP DMA2_HND ; Vector 6: DMA channel 2 2RETI ; Vector 8:
Reserved 5RETI ; Vector 10: Reserved 5RETI ; Vector 12: Reserved 5RETI ; Vector 14: Reserved
5DMA2_HND ; Vector 6: DMA channel 2... ; Task starts hereRETI ; Back to main program 5DMA1_HND ;
Vector 4: DMA channel 1... ; Task starts hereRETI ; Back to main program 5DMA0_HND ; Vector 2: DMA
channel 0... ; Task starts hereRETI ; Back to main program 5
```

6.2.9 在 DMA控制器下使用 USCI_B I²C 模块

USCI_B I²C 模块为 DMA 控制器提供了两个触发源。 当接收到 I²C 数据且当需要传输该数据时，USCI_B I²C 模块可以触发一个传输。

如果 UCB0RXIFG 被置位会触发一个传输。 当 DMA 控制器应答该传输时 UCB0RXIFG 会自动清零。 如果 UCB0RXIE 被置位, UCB0RXIFG 将不会触发一个传输。

如果 UCB0TXIFG 被置位会触发一个传输。 当 DMA 控制器应答该传输时 UCB0TXIFG 会自动清零。 如果 UCB0TXIE 被置位, UCB0TXIFG 将不会触发一个传输。

6.2.10 在 DMA 控制器下使用 ADC12

拥有一个集成的 DMA 控制器的 MSP430 器件可以自动地把数据从任何 ADC12MEMx 寄存器移动到任何位置。 DMA 传输可以在没有 CPU 的干预下完成并且不受任何低功耗模式的影响。 ADC12 模块增加了DMA 控制器的吞吐量，并且当数据传输发生时，通过允许 CPU 保持在关闭状态来提高低功耗应用的性能。

DMA 传输可以被任何 ADC12IFGx 标志触发。 当 CONSEQx={0，2} 时，被用作转换的 ADC12MEMx 的 ADC12IFGx 标志可以触发一个 DMA 传输。 当 CONSEQx={1，3} 时，在顺序转换中的最后一个 ADC12MEMx 的 ADC12IFGx 标志可以触发一个 DMA 传输。 当 DMA 控制器访问相应的 ADC12MEMx 的时候，任何 ADC12IFGx 标志都会被自动清零。

6.2.11 在 DMA 控制器下使用 DAC12

拥有一个集成的 DMA 控制器的 MSP430 器件可以自动地把数据移动到 DAC12_xDAT 寄存器。 DMA 传输可以在没有 CPU 的干预下完成并且不受任何低功耗模式的影响。 DMA 控制器增加了 DAC12 模块的吞吐量，并且当数据传输发生的时候，通过允许 CPU 保持在关闭状态来增强低功耗应用的性能。

需要周期性波形生成的应用程序可以受益于使用 DMA 控制器的 DAC12。 例如，一个产生正弦波的应用程序可以把正弦波的值存储在一个表格中。 为了产生正弦波，，DMA 控制器可以在特定的时间间隔内自动地并且连续不断地把这些值传输到 DAC12， 并且不需要 CPU 的执行。 当 DMA 控制器访问 DAC12_xDAT 寄存器时，DAC12_xCTL DAC12IFG 标志将会被自动清零。

6.2.12 在 DMA 控制器下写入闪存

带有一个集成的 DMA 控制器的 MSP430 器件可以自动地把数据移动到闪存存储器中。 DMA 传输可以在没有 CPU 的干预下完成并且不受任何低功耗模式的影响。 DMA 控制器会把数据字/字节移动到闪存。 写入时序控制是由闪存控制器完成的。 如果在闪存控制器被设置先于 DMA 传输且如果闪存不忙，转移到闪存存储器的写入就会成功，。 要设置闪存控制器的写入访问，请参阅《*闪存存储器控制器*章节》。

6.3 DMA 寄存器

在表 6-5中列出了 DMA 寄存器。

表 6-5. DMA 寄存器

寄存器	简表	寄存器类型	地址	初始化状态
DMA 控制 0	DMACTL0	读取/写入	0122h	用 POR 复位
DMA 控制 1	DMACTL1	读取/写入	0124h	用 POR 复位
DMA 中断向量	DMAIV	只读	0126h	用 POR 复位
DMA 通道 0 控制	DMA0CTL	读取/写入	01D0h	用 POR 复位
DMA 通道 0 源地址	DMA0SA	读取/写入	01D2h	未改变
DMA 通道 0 目标地址	DMA0DA	读取/写入	01D6h	未改变
DMA 通道 0 传送大小	DMA0SZ	读取/写入	01DAh	未改变
DMA 通道 1 控制	DMA1CTL	读取/写入	01DCh	用 POR 复位
DMA 通道 1 源地址	DMA1SA	读取/写入	01DEh	未改变
DMA 通道 1 目标地址	DMA1DA	读取/写入	01E2h	未改变
DMA 通道 1 发送尺寸	DMA1SZ	读取/写入	01E6h	未改变
DMA 通道 2 控制	DMA2CTL	读取/写入	01E8h	用 POR 复位
DMA 通道 2 源地址	DMA2SA	读取/写入	01EAh	未改变
DMA 通道 2 目标地址	DMA2DA	读取/写入	01EEh	未改变
DMA 通道的 2 个传输大小	DMA2SZ	读取/写入	01F2h	未改变

6.3.1 DDMACTL0，DMA 控制寄存器 0

15	14	13	12	11	10	9	8
被保留				DMA2TSELx			
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
DMA1TSELx				DMA0TSELx			
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

被保留 位 15-12 被保留

DMA2TSELx 位 11-8 DMA 的触发选择。 这些位选择 DMA 的传输触发。

- 0000 DMAREQ 位（软件触发）
- 0001 TACCR2 CCIFG 位
- 0010 TBCCR2 CCIFG 位
- 0011 接收的串行数据 UCA0RXIFG
- 0100 串行数据传输就绪 UCA0TXIFG
- 0101 DAC12_0CTL DAC12IFG 位
- 0110 ADC12 ADC12IFGx 位
- 0111 TACCR0 CCIFG 位
- 1000 TBCCR0 CCIFG 位
- 1001 接收的串行数据 UCA1RXIFG
- 1010 串行数据传输就绪 UCA1TXIFG
- 1011 乘法器就绪
- 1100 接收的串行数据 UCB0RXIFG
- 1101 串行数据传输就绪 UCB0TXIFG
- 1110 DMA0IFG 位触发 DMA 通道 1
DMA1IFG 位触发 DMA 通道 2
DMA2IFG 位触发 DMA 通道 0
- 1111 外部触发 DMAE0

DMA1TSELx 位 7-4 同 DMA2TSELx 一样

DMA0TSELx 位 3-0 同 DMA2TSELx 一样

6.3.2 DDMACTL1，DMA 控制寄存器 1

15	14	13	12	11	10	9	8
0	0	0	0	0	0	0	0
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
0	0	0	0	0	DMAON FETCH	ROUND ROBIN	ENNMI
r0	r0	r0	r0	r0	rw-(0)	rw-(0)	rw-(0)

被保留 位 15-3 被保留。 只读. 始终读为 0。

DMAONFETCH 位 2 DMA 上读取

- 0 立即发生 DMA 传输。
- 1 在触发后，DMA 传输发生在下一条指令的读取上。

ROUNDROBIN 位 1 轮循。 该位启用轮循 DMA 通道优先级。

- 0 DMA 通道的优先级是 DMA0 - DMA1 - DMA2
- 1 每次传输 DMA 通道优先级的变化

ENNMI 位 0 启用 NMI。 该位通过一个 NMI 中断来启用一个 DMA 传输中断。 当一个 NMI 中断一个 DMA 传输时，当前传输正常完成，接下来的传输被停止，DMAABORT 被置位。

- 0 NMI 中断不中断 DMA 传输
- 1 NMI 中断中断一个 DMA 传输

6.3.3 DMAxCTL，DMA通道 x 控制寄存器

15	14	13	12	11	10	9	8
被保留	DMADTx			DMADSTINCRx		DMASRCINCRx	
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
DMADST BYTE	DMASRC BYTE	DMALEVEL	DMAEN	DMAIFG	DMAIE	DMAABORT	DMAREQ
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

被保留 位 15 被保留

DMADTx 位 14-12 DMA 传输模式。

000 单次传输
001 块传输
010 突发块传输
011 突发块传输
100 重复单次传输
101 重复块传输
110 重复突发块传输
111 重复突发块传输

DMADSTINCRx 位 11-10 DMA 的目标增量。 在每个字节或字传输后该位选择目的地址的自动递增或递减。 当 DMADSTBYTE=1 时, 目标地址加/减 1。 当 DMADSTBYTE=0 时, 目标地址加/减 2。 DMAxDA 被复制到一个临时的寄存器中，且这个临时寄存器是递增或递减。 DMAxDA 不会增加或者减小。

00 目标地址不变
01 目标地址不变
10 目标地址递减
11 目标地址递增

DMASRCINCRx 位 9-8 DMA 源增量。 在每个字节/字传输完成后此位选择源地址自动递增或递减。 当 DMASRCBYTE=1 时, 源地址加/减 1。 当 DMASRCBYTE=0 时, 源地址加/减 2。 DMAxSA 被复制到一个临时的寄存器中，且这个临时寄存器将是递增或递减 。 DMAxSA 不会增加或者减少。

00 源地址不变
01 源地址不变
10 源地址递减
11 源地址递增

DMADSTBYTE 位 7 DMA 目标字节。 此位选择目标作为字节或字。

0 字
1 字节

DMASRCBYTE 位 6 DMA 源字节。 此位选择源作为字节或字。

0 字
1 字节

DMALEVEL 位 5 DMA 电平 此位在边沿敏感或电平敏感之间选择。

0 边沿敏感（上升沿)
1 电平敏感（高电平）

DMAEN 位 4 DMA 使能

0 被禁用
1 被启用

DMAIFG 位 3 DMA 中断标志

0 无中断挂起
1 中断挂起

DMAIE 位 2 DMA 中断使能

0 被禁用
1 被启用

DMAABORT	位 1	DMA 中断。 此位表明一个 DMA 传输被一个 NMI 中断。 0 DMA 传输没有被中断 1 DMA 传输被 NMI 中断
DMAREQ	位 0	DMA 请求。 软件控制的 DMA 启动。 DMAREQ 被自动复位。 0 没有 DMA 启动 1 启动 DM

6.3.4 DMAxSA，DMA 源地址寄存器

15	14	13	12	11	10	9	8
被保留							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
被保留				**DMAxSAx**			
r0	r0	r0	r0	rw	rw	rw	rw
15	14	13	12	11	10	9	8
DMAxSAx							
rw	rw	rw	rw	rw	rw	rw	rw
7	6	5	4	3	2	1	0
DMAxSAx							
rw	rw	rw	rw	rw	rw	rw	rw

DMAxSA	位 15-0	DMA 源地址 源地址寄存器指向单次传输 DMA 源地址或者指向块传输的第一个源地址。 源地址寄存器在块或者和突发块传输中保持不变。 有可寻址的内存范围为 64KB 或低于 64KB 的器件包含一个单 DMAxSA 字。 当用字操作写入时，上部字会被自动清零。 从这个位置读取总是读为 0。 有可寻址的内存范围超出 64KB 的器件包含一个额外的源地址的字。 新增加字的 15-4 位被保留且始终读为 0。 当用字格式写入 DMAxSA 时，这个额外的字会被自动清零。 使用字格式读取这个字的，且始终读为 0。

6.3.5 DMAxDA，DMA 目的地址寄存器

15	14	13	12	11	10	9	8
被保留							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
被保留				DMAxDAx			
r0	r0	r0	r0	rw	rw	rw	rw
15	14	13	12	11	10	9	8
DMAxDAx							
rw	rw	rw	rw	rw	rw	rw	rw
7	6	5	4	3	2	1	0
DMAxDAx							
rw	rw	rw	rw	rw	rw	rw	rw

DMAxDA 位 15-0 DMA 目标地址

目的地址寄存器指向单次传输 DMA 目的地址或者指向块传输的第一个目的地址。 目的地址寄存器在块或者和突发块传输中保持不变。

有可寻址的内存范围为 64KB 或低于 64KB 的器件包含一个单 DMAxDA 字。

有可寻址的内存范围超出 64KB 的器件包含一个额外的目的地址的字。 新增加字的 15-4 位被保留且始终读为 0。 当用字格式写入 DMAxDA 时，这个额外的字会被自动清零。 使用字格式读取这个字的，且始终读为 0。

6.3.6 DMAxSZ，DMA 大小地址寄存器

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

DMAxSZx 位 15-0 DMA 大小。 DMA 大小寄存器定义了每个块传输的字节/字的数量。 DMAxSZ 寄存器伴随着每个字/字节传输递减。 当 DMAxSZ 减至 0 时, 以前被初始化时的值会马上自动重载。

00000h 传输被禁用

00001h 将被传输的一个字节或者字

00002h 要传输的两个字节或者字

⋮

0FFFFh 必须传输 65536 个字节或者字

6.3.7 DMAIV，DMA 中断向量寄存器

15	14	13	12	11	10	9	8
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**DMAIVx**			**0**
r0	r0	r0	r0	r--(0)	r--(0)	r--(0)	r0

DMAIVx 位 15-0 DMA 中断向量值

DMAIV 内容	中断源	中断标志	中断优先级
00h	无中断等待	-	
02h	DMA 通道 0	DMA0IFG	最高
04h	DMA 通道 1	DMA1IFG	
06h	DMA 通道 2	DMA2IFG	
08h	被保留	-	
0Ah	被保留	-	
0Ch	被保留	-	
0Eh	被保留	-	最低

闪存存储器控制器

本章对 MSP430x2xx 闪存存储器控制器的运行进行了说明。

Topic **Page**

7.1 闪存存储器介绍

MSP430 闪存存储器位、字节、和 字可寻址并且可编程。 闪存存储器模块有一个控制编程和擦除操作的集成型控制器。 控制器有四个寄存器、一个时序发生器、和一个电压生成器来为编程和擦除供电。

MSP430 闪存存储器特性包括：

- 内部编程电压生成
- 位、字节或字可编程
- 超低功耗运行
- 段擦除和批量擦除
- 边界 0 和边界 1 测试模式（可选，请参与器件专用数据表）

图 7-1显示了闪存存储器和控制器的方框图。

注: 闪存写入或者擦除期间的最小 V_{cc}

闪存写入或者擦除操作期间的最小 V_{cc}电压为2.2V。如果 $_{cc}$在写入或擦除期间下降至低于2.2V，写入或擦除的结果则无法预计。

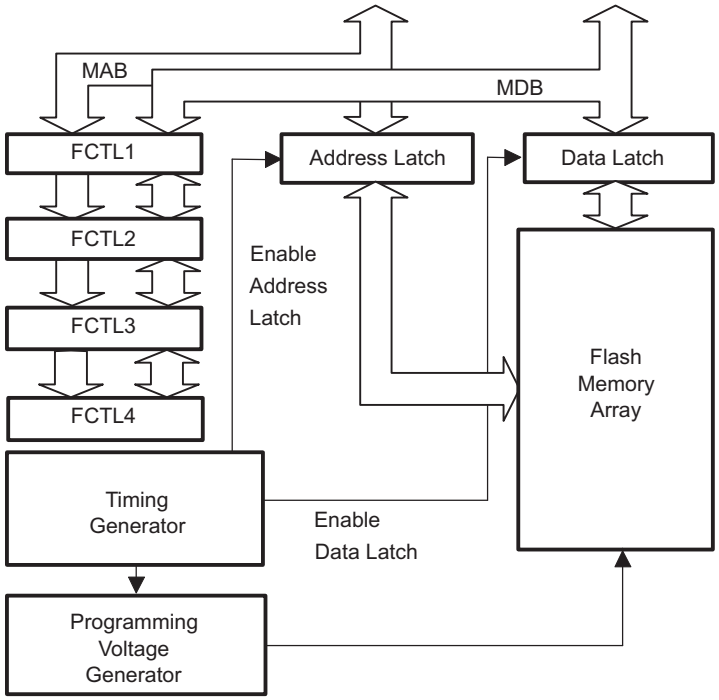


图 **7-1.** 闪存存储器模块方框图

7.2 闪存存储器分段

MSP430 闪存存储器被分为三个段。 单位、字节、或者 字被写入到闪存存储器，但是段是可被擦除的最小尺寸的闪存存储器。

闪存存储器被分为主和信息存储器部分。 主和信息存储器部分的运行没有不同。 代码或者数据可位于两个部分中的任何一个。 两个部分的区别在于段尺寸和物理地址。

信息存储器有四个 64 字节段。 主存储器有多于两个的512 字节段。 一个器件的完整存储器映射请参阅器件专用数据表。

这些段被进一步分为块。

图 7-2显示了使用一个 32KB 闪存示例的闪存分段，此示例有八个主段和四个信息段。

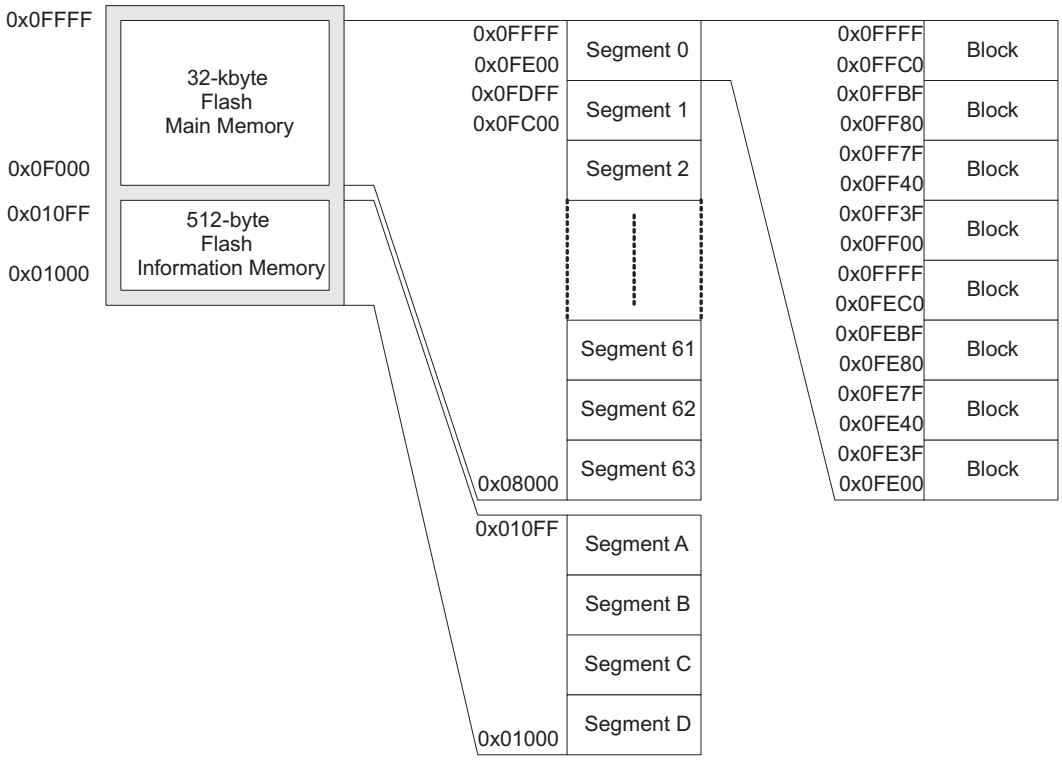


图 7-2. 闪存存储器段，32KB 示例

7.2.1 段 A

用 LOCKA 位将信息存储器的段 A 与所有其它段分开锁定。 当 LOCKA=1 时，段 A 不能被写入或者擦除并且所有信息存储器被保护，以防止一个批量擦除或者生产编程期间的擦除。 当 LOCKA=0 时，段 A作为任何其它闪存存储器段被擦除和写入，并且在一个批量擦除或者生产编程期间，所有信息存储器被擦除。

当一个 1 被写入 LOCKA 位时，它的状态被切换。 将一个 0 写入 LOCKA无效。 这样可在无需更改的情况下，使用现有的闪存编程例程。

```
; Unlock SegmentABIT #LOCKA,&FCTL3 ; Test LOCKAJZ SEGA_UNLOCKED ; Already unlocked?MOV
#FWKEY+LOCKA,&FCTL3 ; No, unlock SegmentASEGA_UNLOCKED ; Yes, continue; SegmentA is unlocked;
Lock SegmentABIT #LOCKA,&FCTL3 ; Test LOCKAJNZ SEGA_LOCKED ; Already locked?MOV
#FWKEY+LOCKA,&FCTL3 ; No, lock SegmentASEGA_LOCKED ; Yes, continue; SegmentA is locked
```

7.3 闪存存储器运行

闪存存储器缺省模式为读取模式。 在读取模式中，闪存存储器不被擦除或被写入，闪存时序发生器和电压生成器关闭，并且存储器运行方式与 ROM 完全一样。

MSP430 闪存存储器系统内可编程 (ISP)，而无需额外的外部电压。 CPU 可编辑它自己的闪存存储器。 用 BLKWRT，WRT，MERAS 和 ERASE 位来选择闪存存储器写入/擦除模式，这些模式为：

- 字节/字写入
- 块写入
- 段擦除
- 批量擦除（所有主存储器段）
- 所有擦除（所有段）

禁止在闪存存储器被编程或者擦除时对其进行读取或写入操作。如果在写入或擦除期间要求 CPU 执行，被执行的代码必须位于 RAM 中。 可从闪存存储器或 RAM 中启动对任一闪存的升级。

7.3.1 闪存存储器时序发生器

写入和擦除操作由图 7-3中显示的闪存时序发生器控制。 闪存时序发生器运行频率，f_{FTG}，必须在大约 257kHz 到大约 476kHz 的范围内（请见器件专用数据表。）

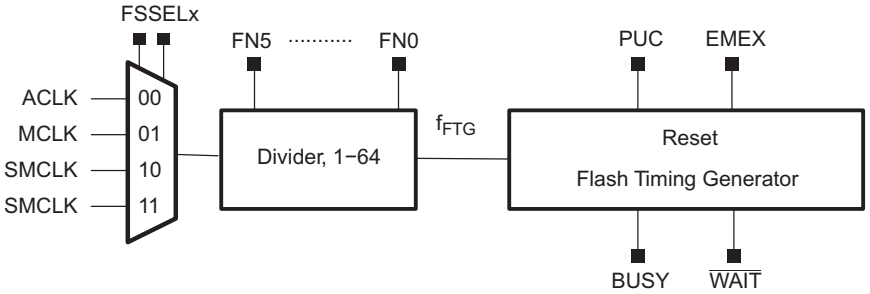


图 **7-3.** 闪存存储器时序发生器方框图

7.3.1.1 闪存时序发生器时钟选择

闪存时序发生器可由 ACLK，SMCLK，或 MCLK 供源。 选择的时钟源应该用 FNx 位进行分频来满足f_{FTG}的频率要求。 如果 f_{FTG}频率在写入或擦除操作期间偏离了额定值，写入或擦除的结果也许无法预计，或者闪存存储器的应力值也许会在可靠运行的限值以上。

如果在写入或擦除操作期间检测到时钟故障，操作被异常中断，故障 (FAIL) 标志被设定，并且运行的结果无法预计。

在一个写入或擦除操作有效时，不能通过将 MSP430置于一个功耗模式来禁用所选择的时钟源。 在被禁用前，所选择的时钟源将保持有效，直到操作完成。

7.3.2 擦除闪存存储器

一个闪存存储器位的被擦除电平为 1。每个为可被单独从 1 设定为 0，但是将 0 重编程为 1 则要求一个擦除周期。 可被擦除的闪存最小数量是一个扇区。 可用表 7-1中列出的 ERASE 和 MERAS 位选择三个擦除模式。

表 7-1. 擦除模式

MERAS	ERASE	擦除模式
0	1	段擦除
1	0	批量擦除（所有主存储器段）
1	1	LOCKA=0：擦除主和信息闪存存储器。 LOCKA=1：只擦除主闪存存储器。

所有的擦除由一个到将被擦除的地址范围内的假写入启动。 假写入启动闪存时序发生器和擦除操作。图 7-4显示了擦除周期时序。 BUSY 在假写入之后被立即置位并且在整个擦除周期内保持置位。 当周期完成时，BUSY，MERAS 和 ERASE 被自动清除。 擦除周期的时序并不取决于出现在器件的闪存存储器的数量。 对于所有 MSP430F2xx 和 MSP430G2xx器件，擦除周期时间相等。

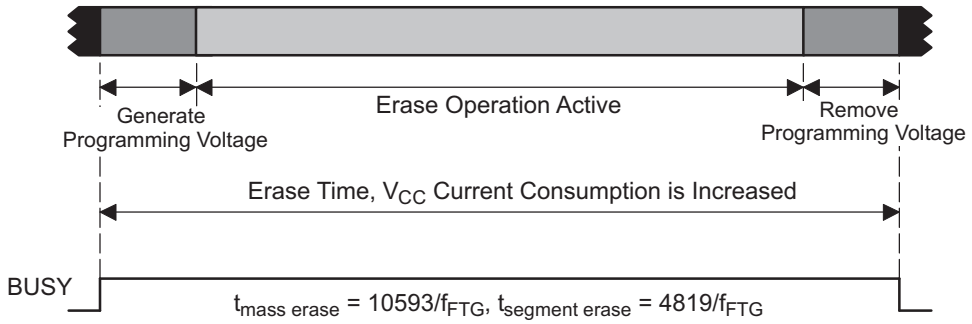


图 7-4. 擦除周期时序

到一个将被擦除的范围之外地址的假写入不会启动擦除周期，并不影响闪存存储器，并且无论如何也不会被标记。 这个错误的假写入被忽略。

7.3.2.1 从闪存存储器内启动一个擦除

可从闪存存储器或者 RAM 中启动一个擦除周期。 当一个闪存段擦除操作从闪存存储器内启动时，所有时序由闪存控制器控制，并且在擦除周期完成时被保持。 在擦除周期完成后，CPU 用假写入之后的指令来恢复代码执行。

当一个擦除周期从闪存存储器内部启动时，可在擦除之后来擦除代码执行所需的代码。 如果这个情况发生，那么在擦除周期之后，CPU 的执行将无法预计。

图 7-5中显示了从闪存启动一个擦除的流程。

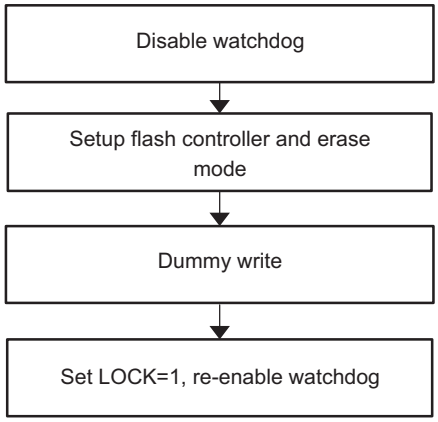


图 **7-5.** 闪存存储器内的擦除周期

```
; Segment Erase from flash. 514 kHz < SMCLK < 952 kHz; Assumes ACCVIE = NMIIE = OFIE = 0.MOV
#WDTPW+WDTHOLD,&WDTCTL ; Disable WDTMOV #FWKEY+FSSEL1+FN0,&FCTL2 ; SMCLK/2MOV #FWKEY, &FCTL3 ;
Clear LOCKMOV #FWKEY+ERASE, &FCTL1 ; Enable segment eraseCLR &0FC10h ; Dummy write, erase S1MOV
#FWKEY+LOCK, &FCTL3 ; Done, set LOCK ... ; Re-enable WDT?
```

7.3.2.2 从 RAM 启动一个擦除

任何擦除周期都可从 RAM 发起。 在这个情况下，CPU 不被保持并且可继续执行 RAM 内的代码。 必须轮询 BUSY 位来在 CPU 能够再次访问任一闪存地址之前确定擦除周期的末尾。 如果在 BUSY=1 时一个闪存访问发生，那么它是一个访问违反，ACCVIFG 将被置位，并且擦除结果不可预计。

图 7-6中显示了从 RAM 内的闪存中发起一个擦除的流程。

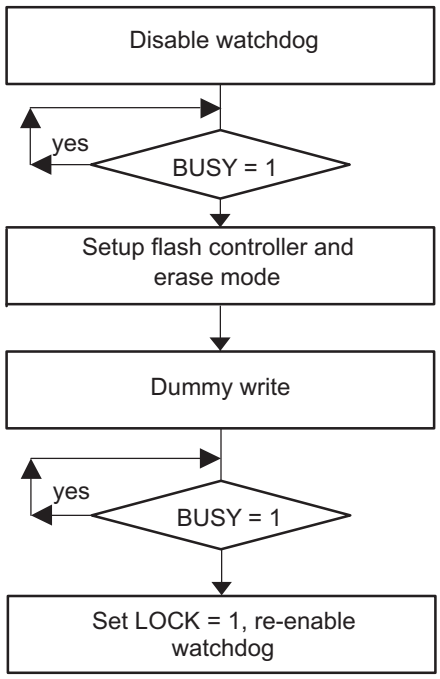


图 **7-6.** 来自 **RAM** 内的擦除周期

```
; Segment Erase from RAM. 514 kHz < SMCLK < 952 kHz; Assumes ACCVIE = NMIIE = OFIE = 0.MOV
#WDTPW+WDTHOLD,&WDTCTL ; Disable WDTL1 BIT #BUSY, &FCTL3 ; Test BUSYJNZ L1 ; Loop while busyMOV
#FWKEY+FSSEL1+FN0, &FCTL2 ; SMCLK/2MOV #FWKEY&FCTL3 ; Clear LOCKMOV #FWKEY+ERASE, &FCTL1 ; Enable
eraseCLR &0FC10h ; Dummy write, erase S1L2 BIT #BUSY, &FCTL3 ; Test BUSYJNZ L2 ; Loop while
busyMOV #FWKEY+LOCK&FCTL3 ; Done, set LOCK... ; Re-enable WDT?
```

7.3.3 写入闪存存储器

表 7-2中列出了由 WRT 和 BLKWRT 位选择的写入模式。

表 7-2. 写入模式

BLKWRT	WRT	写入模式
0	1	字节/字写入
1	1	块写入

两个写入模式都使用一个单独写入指令序列，但是使用块写入模式会比字节/字模式快大约一倍，这是因为在整个块写入时，电压生成器保持打开。 任何修改一个目的的指令可被用于修改字节/字模式或者块写入模式中的一个闪存位置。 在擦除之间，一个闪存字（低+高字节）一定不能被写入多于两次。 否则，会发生器件损坏。

在一个写入操作时，BUSY 位被置位，而当操作完成时，这个位被清零。 如果写入操作从 RAM 中发起，CPU 一定不能在 BUSY=1 时访问闪存。 否则，会出现一个访问违反，ACCVIFG 被置位，并且闪存写入不可预计。

7.3.3.1 字节/字写入

可从闪存存储器或者 RAM 内发起一个字节/字写入操作。当从闪存存储器内发起时，所有时序由闪存控制器控制，并且CPU 在写入完成时被保持。 在写入完成后，CPU 用写入之后的指令来恢复代码执行。 图 7-7中显示了字节/字写入时序。

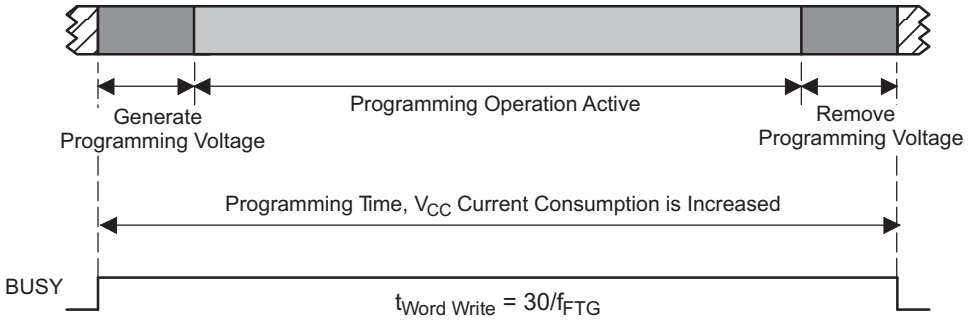


图 7-7. 字节/字写入时序

当一个字节/字写入从 RAM 中执行时，CPU 继续执行来自RAM 的代码。 在 CPU 再次访问闪存前，BUSY 位必须为零，否则一个访问违反会出现，ACCVIFG 被置位，并且写入结果不可预计。

在字节/字模式中，对于 $30f_{FTG}$周期，每次写入一个字节或字内部生成的编程电压被应用于完整 64 字节块，。 在每个字节或者字写入时，块的定时数量取决于积累的编程电压。 累积编程时间，t_{CPT}，一定不能超过任何块。 如果累积编程时间被满足，在执行任何到快内任何地址的进一步写入前，块必须被擦除。 技术规格请见器件专用数据表。

7.3.3.2 从闪存存储器内发起一个字节/字写入

图 7-8中显示了一个从闪存发起一个字节/字写入的例程。

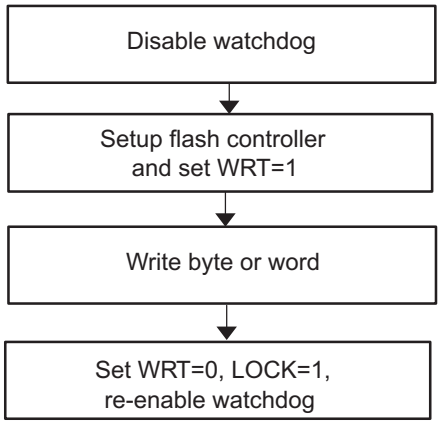


图 **7-8.** 从闪存发起一个字节/字写入

```
; Byte/word write from flash. 514 kHz < SMCLK < 952 kHz; Assumes 0FF1Eh is already erased;
Assumes ACCVIE = NMIIE = OFIE = 0.MOV #WDTPW+WDTHOLD,&WDTCTL ; Disable WDT MOV
#FWKEY+FSSEL1+FN0,&FCTL2 ; SMCLK/2MOV #FWKEY,&FCTL3 ; Clear LOCKMOV #FWKEY+WRT,&FCTL1 ; Enable
writeMOV #0123h,&0FF1Eh ; 0123h -
> 0FF1EhMOV #FWKEY,&FCTL1 ; Done. Clear WRT MOV #FWKEY+LOCK,&FCTL3 ; Set LOCK... ; Re-enable WDT?
```

7.3.3.3 从RAM 发起一个字节/字写入

图 7-9中显示了从 RAM 中发起一个字节/字写入的流程。

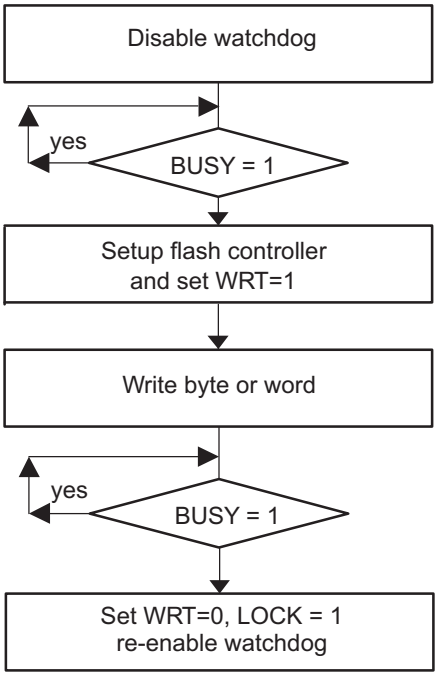


图 **7-9.** 从 **RAM** 中发起一个字节/字写入

```
; Byte/word write from RAM. 514 kHz < SMCLK < 952 kHz; Assumes 0FF1Eh is already erased; Assumes
ACCVIE = NMIIE = OFIE = 0.MOV #WDTPW+WDTHOLD,&WDTCTL ; Disable WDTL1 BIT #BUSY,&FCTL3 ; Test
BUSYJNZ L1 ; Loop while busyMOV #FWKEY+FSSEL1+FN0,&FCTL2 ; SMCLK/2MOV #FWKEY,&FCTL3 ; Clear
LOCKMOV #FWKEY+WRT,&FCTL1 ; Enable writeMOV #0123h,&0FF1Eh ; 0123h -
> 0FF1EhL2 BIT #BUSY,&FCTL3 ; Test BUSYJNZ L2 ; Loop while busyMOV #FWKEY,&FCTL1 ; Clear WRTMOV
#FWKEY+LOCK,&FCTL3 ; Set LOCK... ; Re-enable WDT?
```

7.3.3.4 块写入

当需要编辑很多连续的字节或者字时，块写入可被用于计算闪存写入进程。 在写入 64 字节块的持续时间内，闪存编程电压保持打开。 块写入期间，对于任何一个块写入，一定不能超过累积编程时间 t_{CPT}。

不能从闪存存储器内发起一个块写入。 块写入只能从RAM 发起。 在块写入的整个持续时间内，BUSY 位保持置位。 在块内写入每个字节或者字之间，WAIT 位必须被检查。 当 WAIT 被置位时，才可写入块的下一个字节或者字。 当写入连续的块时，在当前块写入完成时，BLKWRT位必须被检查。 在由 $t_{和}$指定所需的闪存恢复时间后，BLKWRT 可被置位为启动写一个块写入。 在一每一个块写入完成之后，BUSY 位被清零，这表明可写入下一个块。图 7-10显示了块写入时序。

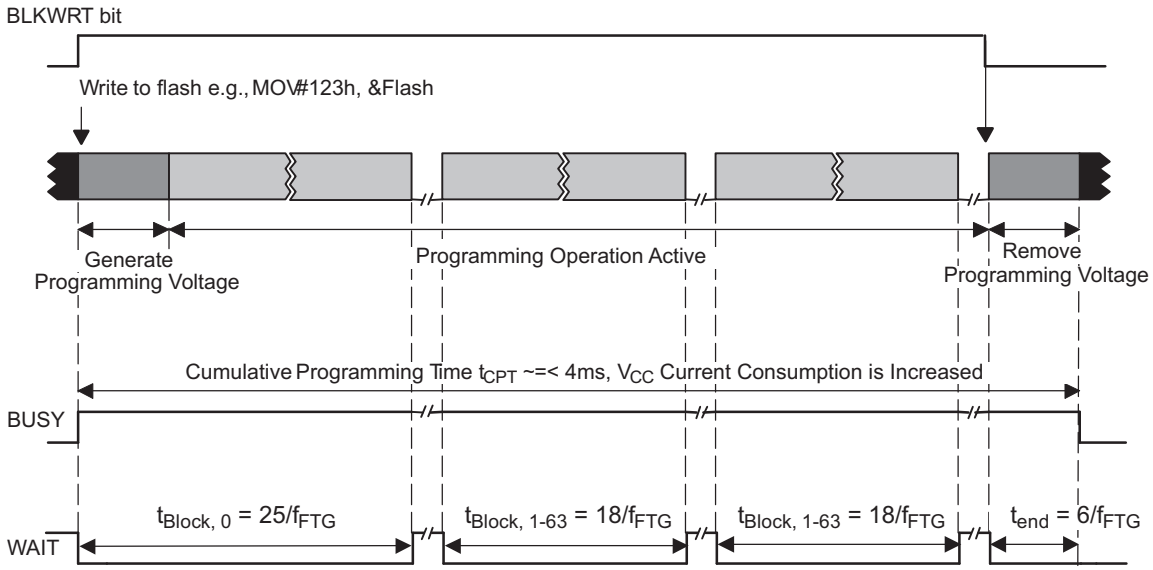


图 **7-10.** 块写入周期时序

7.3.3.5 块写入流程和示例

图 7-11中显示了一个块写入流程和以下示例。

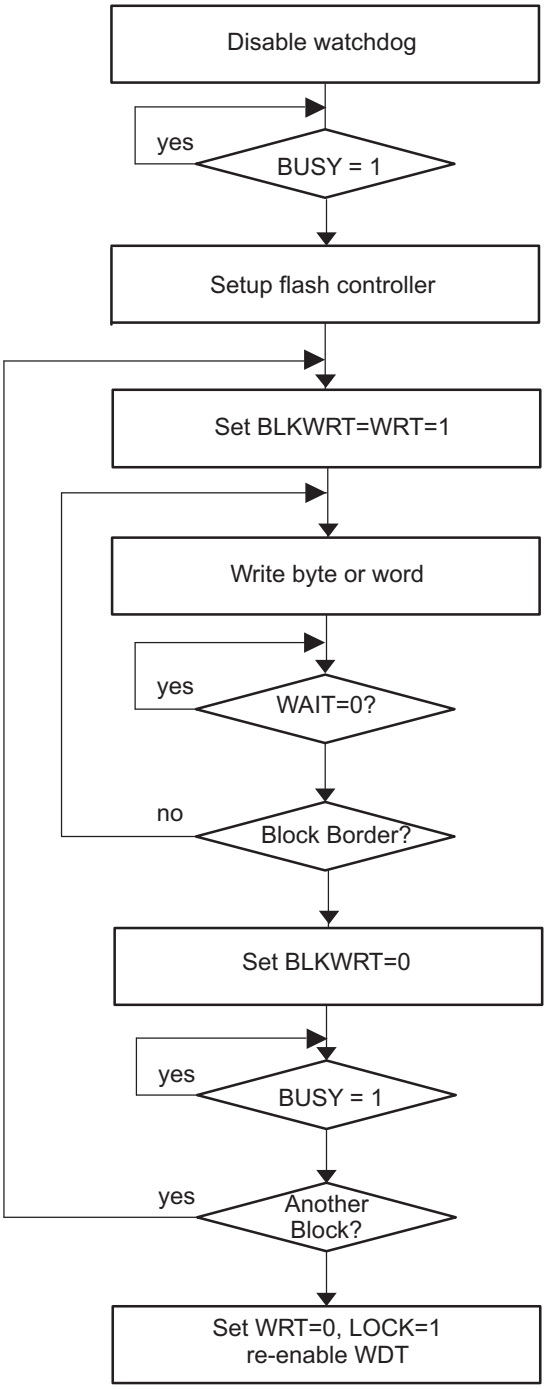


图 **7-11.** 块写入流程

```
; Write one block starting at 0F000h.; Must be executed from RAM, Assumes Flash is already
erased.; 514 kHz < SMCLK < 952 kHz; Assumes ACCVIE = NMIIE = OFIE = 0.MOV #32,R5 ; Use as write
counterMOV #0F000h,R6 ; Write pointerMOV #WDTPW+WDTHOLD,&WDTCTL ; Disable WDTL1 BIT #BUSY,&FCTL3
; Test BUSYJNZ L1 ; Loop while busyMOV #FWKEY+FSSEL1+FN0,&FCTL2 ; SMCLK/2MOV #FWKEY,&FCTL3 ;
Clear LOCKMOV #FWKEY+BLKWRT+WRT,&FCTL1 ; Enable block writeL2 MOV Write_Value,0(R6) ; Write
locationL3 BIT #WAIT,&FCTL3 ; Test WAITJZ L3 ; Loop while WAIT = 0INCD R6 ; Point to next wordDEC
R5 ; Decrement write counterJNZ L2 ; End of block?MOV #FWKEY,&FCTL1 ; Clear WRT,BLKWRTL4 BIT
#BUSY,&FCTL3 ; Test BUSYJNZ L4 ; Loop while busyMOV #FWKEY+LOCK,&FCTL3 ; Set LOCK... ; Re-
enable WDT if needed
```

7.3.4 写入或擦除期间的闪存存储器访问

当任一写入或者擦除操作从 RAM 内发起且 BUSY=1 时，CPU可以对任一闪存位置进行读取或者写入操作。否则，会出现一个访问违反，ACCVIFG被置位，并且结果不可预计。 同样的，如果在 WRT=0 时尝试一个到闪存的写入，ACCVIFG 中断标志被置位，而闪存存储器不受影响。

当一个字节/字写入或者任一擦除操作从闪存存储器内发起时，闪存控制器在写一个取指令时将运行代码 03FFFh 返回到 CPU。 运行代码03FFFh 是 JMP PC 指令。 这使得 CPU 循环执行，直到闪存操作被完成。 操作完成且 BUSY=0 时，闪存控制器将允许 CPU取合适的运行代码并且程序执行继续。

BUSY=1 时的闪存访问条件列于表 7-3中。

表 **7-3. BUSY=1** 时的闪存访问

闪存操作	闪存存取	WAIT	结果
任何擦除，或字节/字写入	读取	0	ACCVIFG=0。03FFFh 是被读取的值。
	写入	0	ACCVIFG=1。写入被忽略。
	取指令	0	ACCVIFG=0。CPU 取 03FFFh。 这条是 JMP PC指令。
块写入	任一	0	ACCVIFG=1，LOCK=1
	读取	1	ACCVIFG=0。03FFFh 是被读取的值。
	写入	1	ACCVIFG=0。被写入。
	取指令	1	ACCVIFG=1，LOCK=1

当 EEI=0 并且 EEIEX=0 时，在 EEI 和 EEIEX 不出现的 MSP430x20xx 和 MSP430G2xx 器件上，中断会在任一闪存操作期间被自动禁用。 在闪存操作完成完成后，中断被自动重新启用。 操作期间任一发生的中断的相关标志将被置位，并且在被重新启用时将生成一个中断请求。

当 EEIEX=1 且 GIE=1 时，一个中断将立即中断任何闪存操作并且FAIL 标志将被置位。 当 EEI=1，GIE=1，并且 EEIEX=0 时，一个段擦除将在每 32 个 f_{FTG}周期被一个等待的中断中断。 在处理中断后，段擦除继续至少 32 个 f_{FTG}周期或者直到它完成。 在中断处理期间，BUSY 保持被置位，但是闪存存储器可由 CPU 访问，而不会导致一个访问违反发生。 不支持嵌套中断和使用中断处理例程内的 RETI 指令的。

安全装置定时器（处于安全装置模式内）应该在一个闪存擦除周期前被禁用。 一个复位将中止擦除并且此结果将无法预计。 擦除周期被完成之后，安全装置可被重新启用。

7.3.5 停止一个写入或擦除周期

在写入或擦除操作正常完成前，通过设置紧急退出位 EMEX，此操作可被停止。 设置 EMEX 位将立即停止有效操作并且停止闪存控制器。 所有闪存操作停止，闪存返回到读取模式，并且所有FCTL1 寄存器内的位被复位。 目的操作的结果无法预计。

7.3.6 边界读取模式

边界读取模式可被用于验证闪存存储器内容的完整性。 这个特性在所选的 2xx 器件内执行；可用性请参阅器件专用数据表。 在边界读取模式，边界编辑的闪存存储器位位置可被检测到。 会产生这一情况的事件包括不适当的 f_{FTG}设置，或者在擦除/编程操作期间违反最小 V_{CC}。 识别此类存储器位置的一个方法将在闪存存储器的一个部分上（例如，一个闪存段）定期执行一个校验和计算并且在边界读取模式被启用时重复这一步骤。 如果它们不匹配，这将表明一个不充分已编辑闪存存储器位置。 可通过禁用边界读取模式，复制至 RAM，擦除闪存段，并且将其从 RAM 写回来刷新受影响的闪存存储器。

检查闪存存储器内容的程序必须从 RAM 执行。 从闪存执行代码将自动禁用边界读取模式。 边界读取模式由 MRG0 和 MRG1 寄存器位控制。 设置 MRG1 被用来检测未充分编辑的包含一个‘1’（被擦除的位）的闪存单元。 设置 MRG0 被用来检测未充分编辑的包含一个‘0’（已编辑的位）的闪存单元。 应该一次只设定这些位中的一个。 因此，一个完全边界读取检查将要求两次合格的闪存存储器内容完整性检查。 边界读取模式期间，闪存访问速度 (MCLK) 必须被限制在 1MHz（请参阅器件专用数据表）。

7.3.7 配置和访问闪存存储器控制器

FCLTx 寄存器是 16 位，受密码保护的，读取/写入寄存器。 任何读取或者写入访问必须使用字指令，而读取访问必须在上部字节中包括写入密码 0A5h。 上部字节中的值不是 0A5h 的任何到 FCTLx 寄存器的写入是一个安全密钥违反，将设定 KEYV 表中并且触发一个 PUC 系统复位。 到任何 FCTLx寄存器的读取将读取上部字节内的 096h 值。

在擦除或者字节/字写入操作期间，到 FCTL1 的任何写入操作是一个访问违反并且将 ACCVIFG 置位。 当 WAIT=1 时，允许块写入模式下的到 FCTL1 的写入，但是当 WAIT=0 时，块写入模式下的到FCTL1 的写入是一个访问违反并且将 ACCVIFG 置位。

当 BUSY=1 时，任何到 FCTL1 的写入是一个访问违反。

当 BUSY=1 时，可读取任一 FCTLx 寄存器。读取不会导致访问违反。

7.3.8 闪存存储器控制器中断

闪存控制器有两个中断源，KEYV，和 ACCVIFG。 当一个访问违法发生时，ACCVIFG 被置位。 当一个闪存写入或擦除之后，ACCVIFG 被重新启用时，一个设置ACCVIFG 标志的操作将生成一个中断请求。 ACCVIFG 为 NMI 中断矢量供源，所以对于ACCVIFG 中断请求没必要设定 GIE。 也可通过软件来检查 ACCVIFG 以确定是否发生了访问违反。 ACCVIFG 必须由软件复位。

当使用一个不正确的密码写入任一闪存控制寄存器时，密码违反标志 KEYV 被置位。 当这一情况发生时，通过复位器件，PUC 被立即生成。

7.3.9 编辑闪存存储器器件

有三个选项可用来编辑 MSP430 闪存器件 所有选项支持系统内编程：

- 通过 JTAG 编程
- 通过引导加载程序编程
- 听过一个定制解决方案编程

7.3.9.1 通过JTAG 编辑闪存存储器

MSP430 可通过 JTAG 端口编程。 JTAG 接口要求四个信号（在 20 引脚和 28 引脚上为五个信号），接地和可选的V_{CC}和$\overline{RST}$/NMI。

JTAG 端口由一个熔丝保护。 完全熔断熔丝将禁用 JTAG端口并且不可恢复。 无法再通过 JTAG 来访问此器件。 详细信息，请参阅《*通过 JTAG 接口进行 MSP430 编程用户指南*》(SLAU320)。

7.3.9.2 通过引导加载程序 (BSL) 编辑闪存存储器

大多数 MSP430 闪存器件包含一个引导加载程序。 执行细节请参阅器件专用数据表。 BSL 使用户能够采用一个 UART 串行接口来读取或编辑闪存存储器。 通过 BSL 对 MSP430 闪存存储器的访问由一个用户定义的 256 位密码保护。 更多细节，请参阅《*通过 引导加载程序进行 MSP430 编程用户指南*》(SLAU319)。

7.3.9.3 通过一个定制解决方案编辑闪存存储器

MSP430 CPU 向其自身闪存存储器的写入功能可实现系统内和外部定制编程解决方案，如图 7-12所示。 用户可以选择通过可用的任何方式（UART，SPI 等）来为 MSP430 通过数据。 用户开发的软件能够接收数据并且便捷闪存存储器。 由于这个解决方案类型由用户开发，它可被完全定制以符合针对编程、擦除、或升级闪存存储器的应用需要。

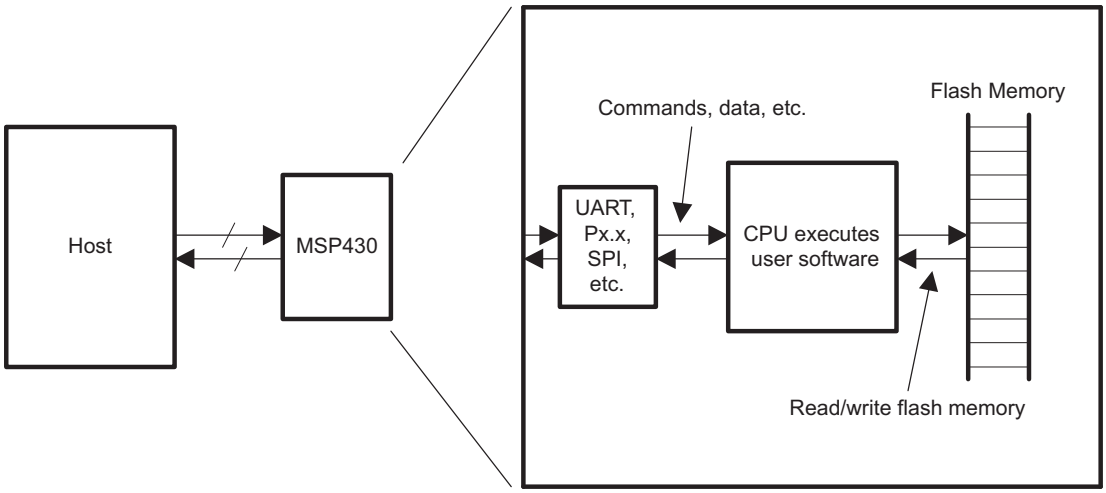


图 **7-12.** 用户开发的编程解决方案

7.4 闪存存储器寄存器

表 7-4中列出了闪存存储器寄存器。

表 7-4. 闪存存储器寄存器

寄存器	简氏	寄存器类型	地址	初始状态
闪存存储器控制寄存器 1	FCTL1	读取/写入	0x0128	带有 PUC 的 0x9600
闪存存储器控制寄存器 2	FCTL2	读取/写入	0x012A	带有 PUC 的 0x9642
闪存存储器控制寄存器 3	FCTL3	读取/写入	0x012C	带有 PUC 的 0x9658(1)
闪存存储器控制寄存器 4(2)	FCTL4	读取/写入	0x01BE	带有 PUC 的 0x0000
中断使能 1	IE1	读取/写入	0x0000	带有 PUC 的复位
中断标志 1	IFG1	读取/写入	0x0002	

(1) KEYV 由 POR 复位。
(2) 并不出现在所有器件上。 请见器件专用数据表。

7.4.1 FCTL1，闪存存储器控制寄存器

15	14	13	12	11	10	9	8
FRKEY读取为 096h **FWKEY**，必须被写为 0A5h							

7	6	5	4	3	2	1	0
BLKWRT	**WRT**	被保留	**EEIEX**(1)	**EEI**(1)	**MERAS**	**ERASE**	被保留
rw-0	rw-0	r0	rw-0	rw-0	rw-0	rw-0	r0

FRKEY
FWKEY 位 15-8 FCTLx 密码。 一直读取为 096h。 必须被写为 0A05h 或者一个 PUC被生成。

BLKWRT 位 7 块写入模式。 对于块写入模式，WRT 必须被设定。 当 EMEX 被置位时，BLKWRT 被自动复位。
- 0 块写入模式关闭
- 1 块写入模式打开

WRT 位 6 写入。 这个位被用于选择任一写入模式。 当EMEX 被置位时，WRT 被自动复位。
- 0 写入模式关闭
- 1 块写入模式打开

被保留 位 5 被保留。 一直读取为 0。

EEIEX 位 4 使能紧急中断退出。 当 GIE=1 时，设置这个位启用一个位来引起一个从闪存操作中的紧急退出。当 EMEX 被置位时，EEIEX 被自动复位。
- 0 退出中断被禁用。
- 1 在中断上退出被启用。

EEI 位 3 启用擦除中断。 对这个位的设置可通过一个中断请求来中断一个段擦除。 中断被处理后，擦除周期重新开始。
- 0 段擦除被禁用期间的中断。
- 1 段擦除被启用期间的中断。

MERAS
ERASE 位 2
位 1 批量擦除和擦除。 这些位被一起用来选择擦除模式。 当 EMEX 被置位时，MERAS和 ERASE 被自动复位。

MERAS	ERASE	擦除周期
0	0	无擦除
0	1	只擦除单独的段
1	0	擦除所有主存储器段
1	1	LOCKA= 0：擦除主和信息闪存存储器。 LOCKA= 1：只擦除主闪存存储器。

被保留 位 0 被保留。 一直读取为0。

(1) 在 MSP430x20xx 和 MSP430G2xx器件上不出现。

7.4.2 FCTL2，闪存存储器控制寄存器

15	14	13	12	11	10	9	8
FWKEYx，读取为 096h 必须被写为 0A5h							

7	6	5	4	3	2	1	0
FSSELx		**FNx**					
rw-0	rw-1	rw-0	rw-0	rw-0	rw-0	rw-1	rw-0

FWKEYx 位 15-8 FCTLx 密码。 一直读取为 096h。 必须被写为 0A05h 或者一个 PUC被生成。

FSSELx 位 7-6 闪存控制器时钟源选择
- 00 ACLK
- 01 MCLK
- 10 SMCLK
- 11 SMCLK

FNx 位 5-0 闪存控制器时钟分频器。 这些位为闪存控制器时钟选择分频器。 除数值为 FNx+1。例如，当 FNx=00h 时，除数为 1。当FNx=03Fh 时，除数为 64。

7.4.3 FCTL3，闪存存储器控制寄存器

15	14	13	12	11	10	9	8
FWKEYx，读取为 096h 必须被写为 0A5h							

7	6	5	4	3	2	1	0
FAIL	**LOCKA**	**EMEX**	**LOCK**	**WAIT**	**ACCVIFG**	**KEYV**	**BUSY**
r(w)-0	r(w)-1	rw-0	rw-1	r-1	rw-0	rw-(0)	r(w)-0

FWKEYx 位 15-8 FCTLx 密码。 一直读取为 096h。 必须被写为 0A05h 或者一个 PUC被生成。

FAIL 位 7 运行故障。 如果 fFTG 时钟源故障，则这个位被置位，或者当 EEIEX=1 时，一个闪存操作被一个中断中止。 必须用软件复位 FAIL。

0 无故障

1 故障

LOCKA 位 6 段 A 和信息锁定。 写入一个 1 到这个位来改变其状态。 写入 0 无效。

0 在批量擦除期间，段 A被解锁并且所有信息存储器被擦除。

1 在批量擦除期间，段 A被锁定并且所有信息存储器被保护不被擦除。

EMEX 位 5 紧急退出

0 无紧急退出

1 紧急退出

LOCK 位 4 锁定。 这个位解锁闪存存储器使其能够被写入或擦除。 在一个字节/字写入或者擦除操作期间 LOCK 位可随时被置位，并且操作将正常完成。在块写入模式中，如果 LOCK 位被置位且 BLKWRT=WAIT=1，那么 BLKWRT 和 WAIT 被复位并且模式正常终止。

0 未锁定

1 已锁定

WAIT 位 3 等待。 表明闪存存储器正在被写入。

0 闪存存储器没有为下一个字节/字写入做好准备

1 闪存存储器已为为下一个字节/字写入做好准备

ACCVIFG 位 2 访问违反中断标志

0 无中断挂起

1 中断挂起

KEYV 位 1 闪存安全密码违反。 当置位时，这个位表明一个不正确的 FCTLx 密码被写入任一闪存控制寄存器并且生成一个 PUC。 必须用软件来复位 KEYV。

0 FCTLx密码被正确写入

1 FCTLx密码写入不正确

BUSY 位 0 忙。 这个位表明闪存时序发生器的状态。

0 不忙

1 忙

7.4.4 FCTL4，闪存存储器控制寄存器

并不是所有器件都提供这个寄存器。 细节请参阅器件专用数据表。

15	14	13	12	11	10	9	8
FWKEYx，被读取为 096h 必须被写为 0A5h							

7	6	5	4	3	2	1	0
		MRG1	**MRG0**				
r-0	r-0	rw-0	rw-0	r-0	r-0	r-0	r-0

FWKEYx	位 15-8	FCTLx 密码。 一直读取为 096h。 必须被写为 0A05h 或者一个 PUC 将被生成。
被保留	位 7-6	被保留。 一直读取为 0。
MRG1	位 5	边界读取 1 模式。 这个位启用边界 1 读取模式。 如果 CPU 从闪存存储器开始执行，边界读取 1位被清除。 如果 MRG1 和 MRG0 都被置位，MRG1 有效，而 MRG0 被忽略。 0 边界 1读取模式被禁用。 1 边界 1读取模式被启用
MRG0	位 4	边界读取 0 模式 这个位启用边界 0 读取模式。 如果 CPU 从闪存存储器开始执行，边界读取模式 0被清除。 如果 MRG1 和 MRG0 都被置位，MRG1有效，而 MRG0 被忽略。 0 边界0 读取模式被禁用。 1 边界0 读取模式被启用
被保留	位 3-0	被保留。 一直读取为 0。

7.4.5 IE1，中断启用寄存器1

7	6	5	4	3	2	1	0
		ACCVIE					
		rw-0					

	位 7-6	其它模块可使用这些位。 请参阅器件专用数据表。
ACCVIE	位 5	闪存存储器非法访问中断启用. 这个位启用 ACCVIFG中断。 由于 IE1 中的其它位可被用于其它模块，建议使用 BIS.B 或者 BIC.B 指令来置位或者清除这个位，而不是使用 MOV.B 或 CLR.B 指令。 0 中断未被启用 1 中断被启用
	位 4-0	其它模块可使用这些位。 请参阅器件专用数据表。

数字 I/O

本章对数字 I/O 端口的运行进行了说明。

Topic **Page**

8.1 数字 I/O 介绍

MSP430 执行多达 8 个数字 I/O 端口，P1 至 P8。 每个端口有多达 8 个 I/O 引脚。 每个 I/O 引脚可针对输入或输出方向单独配置，并且可对每个 I/O 线路单独进行读取或写入操作。

端口 P1 和 P2 有中断功能。 每个针对 P1 和 P2 I/O 线路的中断可被单独启用并被配置成在一个输入信号的上升或者下降边沿上提供一个中断。 所有 I/O 线路提供一个单一中断矢量，并且所有 P2 I/O 线路提供一个不同的，单中断矢量。

数字 I/O 特性包括：

- 单独可编程独立 I/O
- 输入或输出的任意组合
- 单独可配置的 P1 和 P2 中断
- 独立输入和输出数据寄存器
- 单独可配置的上拉或者下拉电阻器
- 单独可配置的引脚振荡器功能（某些 MSP430 器件）

注： **MSP430G22x0**：这些器件特有数字 I/O 引脚P1.2，P1.5，P1.6 和 P1.7。通用输入输出 (GPIO) P1.0，P1.1，P1.3，P1.4，P2.6 和 P2.7 在这个器件上执行，但是不在器件上提供输出引脚。 为了避免悬空输入，这些 GPIO，这些数字 I/O 应该通过运行一个启动代码来适当的初始化。 初始化代码如下
：mov.b #0x1B, P1REN;; 适当终止不可用的端口 1；配置为输入且下拉电阻器被启用
xor.b #0x20, BCSCTL3;; 将 VLO 选为低频时钟
初始化代码配置 GPIO P1.0，P1.1，P1.3，和 P1.4 为输入，且下拉电阻器被启用（即，P1REN.x=1）并且通过将 VLOCLK 选为 ACLK 来中止 GPIO P2.6 和 P2.7，细节请参阅基本时钟系统一章。 在初始化代码被执行后，寄存器 P1OUT，P1DIR，P1IFG，P1IE，P1IES 和 P1REN 中的寄存器位P1.0，P1.1，P1.3 和 P1.4 不应被改变。 此外，所有端口 2 寄存器不应被改变。

8.2 数字 I/O 运行

使用用户软件来配置数字 I/O。 在下面的小节中讨论数字I/O 的设置和运行。

8.2.1 输入寄存器 PxIN

当引脚被配置为 I/O 功能时，PxIN 寄存器中的每个位反映相应 I/O 引脚上输入信号的值。

位 = 0：输入为低电平

位 = 1：输入为高电平

注： **写入只读寄存器 PxIN**

写入这些只读寄存器将在写入尝试被激活时增加流耗。

8.2.2 输出寄存器 PxOUT

当引脚被配置为 I/O 功能，输出方向，和上拉/下拉电阻器被禁用时，每个 PxOUT 寄存器中的每个位是相应 I/O 引脚上将被输出的值

位 = 0：输出为低电平

位 = 1：输出位高电平

如果引脚上的上拉/下拉电阻器被启用，PxOUT寄存器中的相应位选择上拉或下拉电阻器。

位 = 0：引脚被下拉

位 = 1：引脚被上拉

8.2.3 方向寄存器 PxDIR

每个 PxDIR 寄存器中的每个位选择相应 I/O 引脚的方向，这与为引脚选择的功能无关。 被选择用于其它功能的 I/O 引脚的 PxDIR 位必须按照其它功能的要求进行设定。

位 = 0：端口引脚被切换至输入方向

位 = 1：端口引脚被切换至输出方向

8.2.4 上拉/下拉电阻器使能寄存器PxREN

每个 PxREN 寄存器中的每个位启用或者禁用相应 I/O 引脚的上拉/下拉电阻器。 PxOUT 寄存器中的相应位选择是否上拉或下拉引脚。

位 = 0：上拉/下拉电阻器被禁用

位 = 1：上拉/下拉电阻器被启用

8.2.5 功能选择寄存器 PxSEL 和PxSEL2

端口引脚通常与其它外设模块功能复用。 请参见器件专用数据表来确定引脚功能。 每个 PxSEL 和 PxSEL2 位被用来选择引脚功能 - I/O 端口或者外设模块功能。

表 8-1. PxSEL 和 PxSEL2

PxSEL2	PxSEL	引脚功能
0	0	I/O 功能被选择。
0	1	主外设模块功能被选择。
1	0	被保留。 请参阅器件专用数据表。
1	1	第二外设模块功能被选择。

设置 PxSELx=1 不能自动设定引脚方向。 其它外设模块功能也许要求 PxDIRx 位被按照模块功能所需的方向进行配置。 请参阅器件专用数据表中的引脚电路原理图。

注： 当 PxSEL=1 时设置 PxREN=1

在 MSP430F261x 和 MSP430F2416/7/8/9 的某些 I/O 端口上，在模块功能被选择时 (Pxsel=1) 启用上拉/下拉电阻器 (PxREN=1) 不会禁用逻辑输出驱动器。 不建议采用这个组合，并且这个组合有可能导致流经电阻器的有害电流。 要获得更多信息，请参阅器件专用数据表。

```
;Output ACLK on P2.0 on MSP430F21x1BIS.B #01h,&P2SEL ; Select ACLK function for pinBIS.B
#01h,&P2DIR ; Set direction to output *Required*
```

注： 当 PxSEL=1 时，P1 和 P2 中断被禁用

当 P1SELx 或 P2SELx 位中的任何一个被置位时，相应引脚的中断功能被禁用。 因此，这些引脚上的信号将不会生成 P1 或者 P2 中断，这与相应 P1IE 或 P2IE 位的状态无关。

当一个端口引脚被选为一个到外设的输入时，到外设的输入信号表示器件引脚上信号被锁存。 当 PxSELx=1 时， 内部输入信号在引脚上的信号之后。 然而，如果 PxSELx=0，到外设的输入在 PxSELx 位被复位前保持器件引脚上输入信号的值。

8.2.6 引脚振荡器

某些 MSP430 器件有一个内置于某些信号内的引脚振荡器功能。 引脚振荡器功能可被用在电容触感应用中以免除对外部无源组件的需要。 此外，引脚振荡器可被用在传感器应用中。

无需使用外部组件生成振荡

电容式传感器可被直接连接至 MSP430 引脚

大约为 0.7V 的稳健，典型内置滞后

当引脚振荡器功能被启用时，其它的引脚配置被写覆盖。 在弱上拉/下拉电阻器被启用并且由引脚本身上的电压电平控制时，输入驱动器被关闭。I/O 上的电压被馈入引脚的施密特触发器，然后被路由至一个定时器。 到定时器的连接为器件专用，因此，在器件专用数据表中进行了定义。施密特触发器输出被反转，然后决定是否启用上拉或下拉电阻器。由于反转，一旦引脚振荡器引脚配置被选择，引脚开始振荡。 在路由至一个定时器时钟输出或定时器捕捉通道前，某些引脚振荡器输出被通过逻辑与操作组合在一起。 因此，应该每一次只启用一个引脚振荡器。 每个引脚的振荡器频率由引脚上的负载和 I/O类型定义。 带有模拟功能的 I/O 通常显示了比纯数字I/O 更低的振荡频率。 有关详细信息，请参阅器件专用数据表。 无外部负载的引脚显示的典型振荡频率在 1MHz 到 3MHz 之间。

电容式触摸应用中的引脚振荡器

图 8-1中显示了一个使用引脚振荡器的典型触摸板应用。

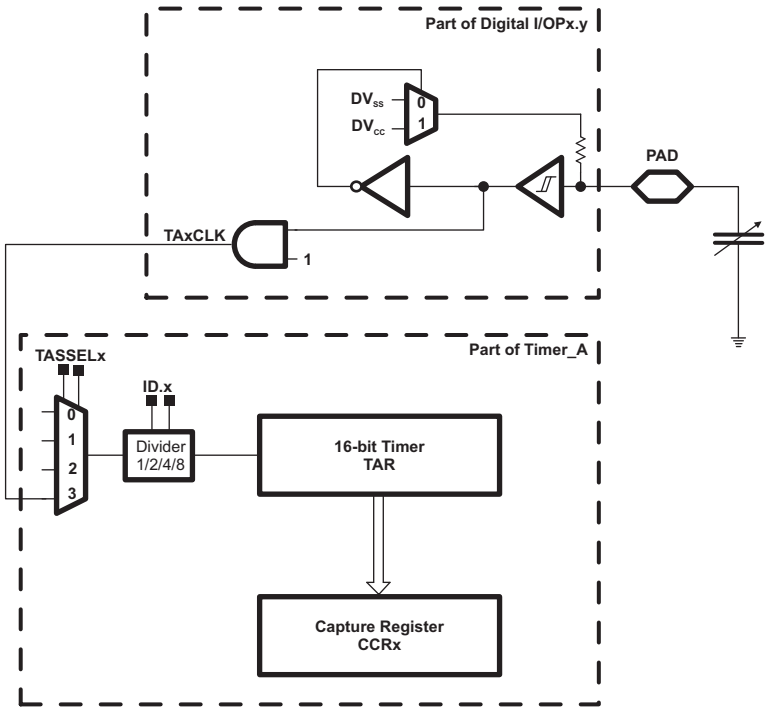


图 8-1. 使用引脚振荡器的示例电路和配置

触摸板电容的变化（外部电容负载）会影响引脚振荡器频率。 一个正在接近的指尖增加了触摸板的电容，从而导致一个由更长充电时间引起的较低自身振荡频率。 可在一个内置定时器通道中捕捉振荡频率。 一个引脚敏感度的典型值显示在图 8-2中。

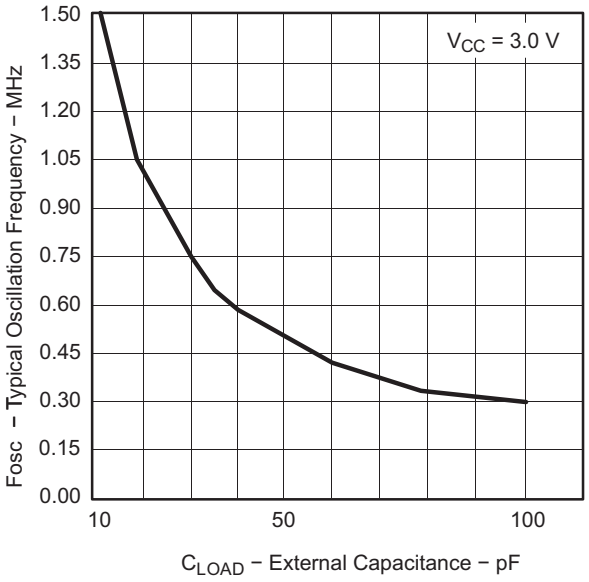


图 **8-2.** 典型引脚振荡频率

8.2.7 P1 和 P2 中断

端口 P1 和 P2 中的每个引脚有中断功能，此功能由 PxIFG，PxIE，和 PxIES 寄存器配置。 所有 P1 引脚提供一个单一中断矢量，而所有 P2 引脚提供一个不同的单一中断矢量。 可测试 PxIFG 寄存器来确定一个 P1 或 P2 中断的源。

8.2.7.1 中断标志寄存器 P1IFG，P2IFG

每个 PxIFGx 位是针对其相应 I/O 引脚的中断标志，并且当被选择的输入信号边沿出现在引脚上时被置位。 当它们相应的 PxIE 位和 GIE 位被置位时，所有 PxIFGx 中断标志要求一个中断。 每个 PxIFG 标志必须由软件复位。 软件也可设定每个 PxIFG 标志，从而提供了一个生成软件初始中断的方法。

位 = 0：无中断等待

位 = 1：一个中断等待

只有转换，而非静态电平，导致中断。 如果在一个 Px 中断处理例程期间任何 PxIFGx 标志被置位，或者在一个 Px 中断处理例程的 RETI 指令被执行后被置位的话，被置位的 PxIFGx 标志生成另外一个中断。 这确保每个转换被确认。

注： 当改变 **PxOUT** 或 **PxDIR** 时的 **PxIFG** 标志

写入 P1OUT，P1DIR，P2OUT，或 P2DIR 可导致设置相应的P1IFG 或者 P2IFG 标志。

8.2.7.2 中断边沿选择寄存器 P1IES，P2IES

每个 PxIES 位位相应的 I/O 引脚选择中断标志。

位 = 0：用一个低电平到高电平转换来设定 PxIFGx 标志

位 = 1：用一个高电平到低电平转换来设定 PxIFGx 标志

注: 写入 **PxIESx**

到 P1IES，或者 P2IES 的写入可导致相应中断标志的设置。

PxIESx	PxINx	PxIFGx
0→1	0	可被置位
0→1	1	未改变
1→0	0	未改变
1→0	1	可被置位

8.2.7.3 中断使能 P1IE，P2IE

每个 PxIE 位启用相关的 PxIFG 中断标志。

位 = 0：中断被禁用。

位 = 1：中断被启用。

8.2.8 配置未使用的端口引脚

未使用的 I/O 引脚应该被配置为 I/O 功能，输出方向，并在板上保持未连接状态，以防止一个悬空输入并减少流耗。 由于引脚未连接，PxOUT 位的值无关。 或者，集成的上拉/下拉电阻器可通过设定未使用引脚的 PxREN 位来启用以防止悬空输入。 未使用引脚的终止，请参阅*系统复位、中断、和运行模式*一章。

8.3 输入 I/O 寄存器

表 8-2中列出了数字 I/O 寄存器。

表 8-2. 数字 I/O 寄存器

端口	寄存器	简氏	地址	寄存器类型	初始状态
P1	输入	P1IN	020h	只读	-
	输出	P1OUT	021h	读取/写入	未改变
	方向	P1DIR	022h	读取/写入	由 PUC 复位
	中断标志	P1IFG	023h	读取/写入	由 PUC 复位
	中断边沿选择	P1IES	024h	读取/写入	未改变
	中断使能	P1IE	025h	读取/写入	由 PUC 复位
	端口选择	P1SEL	026h	读取/写入	由 PUC 复位
	端口选择 2	P1SEL2	041h	读取/写入	由 PUC 复位
	电阻器使能	P1REN	027h	读取/写入	由 PUC 复位
P2	输入	P2IN	028h	只读	-
	输出	P2OUT	029h	读取/写入	未改变
	方向	P2DIR	02Ah	读取/写入	由 PUC 复位
	中断标志	P2IFG	02Bh	读取/写入	由 PUC 复位
	中断边沿选择	P2IES	02Ch	读取/写入	未改变
	中断使能	P2IE	02Dh	读取/写入	由 PUC 复位
	端口选择	P2SEL	02Eh	读取/写入	带 PUC 的 0C0h
	端口选择 2	P2SEL2	042h	读取/写入	由 PUC 复位
	电阻器使能	P2REN	02Fh	读取/写入	由 PUC 复位
P3	输入	P3IN	018h	只读	-
	输出	P3OUT	019h	读取/写入	未改变
	方向	P3DIR	01Ah	读取/写入	由 PUC 复位
	端口选择	P3SEL	01Bh	读取/写入	由 PUC 复位
	端口选择 2	P3SEL2	043h	读取/写入	由 PUC 复位
	电阻器使能	P3REN	010h	读取/写入	由 PUC 复位
P4	输入	P4IN	01Ch	只读	-
	输出	P4OUT	01Dh	读取/写入	未改变
	方向	P4DIR	01Eh	读取/写入	由 PUC 复位
	端口选择	P4SEL	01Fh	读取/写入	由 PUC 复位
	端口选择 2	P4SEL2	044h	读取/写入	由 PUC 复位
	电阻器使能	P4REN	011h	读取/写入	由 PUC 复位
P5	输入	P5IN	030h	只读	-
	输出	P5OUT	031h	读取/写入	未改变
	方向	P5DIR	032h	读取/写入	由 PUC 复位
	端口选择	P5SEL	033h	读取/写入	由 PUC 复位
	端口选择 2	P5SEL2	045h	读取/写入	由 PUC 复位
	电阻器使能	P5REN	012h	读取/写入	由 PUC 复位
P6	输入	P6IN	034h	只读	-
	输出	P6OUT	035h	读取/写入	未改变
	方向	P6DIR	036h	读取/写入	由 PUC 复位
	端口选择	P6SEL	037h	读取/写入	由 PUC 复位
	端口选择 2	P6SEL2	046h	读取/写入	由 PUC 复位
	电阻器使能	P6REN	013h	读取/写入	由 PUC 复位

表 8-2. 数字 I/O 寄存器 (continued)

端口	寄存器	简氏	地址	寄存器类型	初始状态
P7	输入	P7IN	038h	只读	-
	输出	P7OUT	03Ah	读取/写入	未改变
	方向	P7DIR	03Ch	读取/写入	由 PUC 复位
	端口选择	P7SEL	03Eh	读取/写入	由 PUC 复位
	端口选择 2	P7SEL2	047h	读取/写入	由 PUC 复位
	电阻器使能	P7REN	014h	读取/写入	由 PUC 复位
P8	输入	P8IN	039h	只读	-
	输出	P8OUT	03Bh	读取/写入	未改变
	方向	P8DIR	03Dh	读取/写入	由 PUC 复位
	端口选择	P8SEL	03Fh	读取/写入	由 PUC 复位
	端口选择 2	P8SEL2	048h	读取/写入	由 PUC 复位
	电阻器使能	P8REN	015h	读取/写入	由 PUC 复位

电源电压监控器 (SVS)

本章描述了 SVS 的运行。 SVS 在已选 MSP430x2xx 器件上执行。

Topic **Page**

9.1 电源电压监控器 (SVS) 介绍

SVS 被用于监控 AV_{CC}电源电压或一个外部电压。 当电源电压或外部电压降至一个用户已选的阈值以下时，可以配置 SVS 来置位一个标志或产生一个 POR 复位。

SVS 的功能包括：

- AV_{CC}监控
- POR 的可选生成
- 软件可访问的 SVS 比较器输出
- 低电压条件下被锁存和可由软件访问
- 14 个可选择的阈值水平
- 外部通道管理外部电压

在图 9-1中显示了 SVS 结构图。

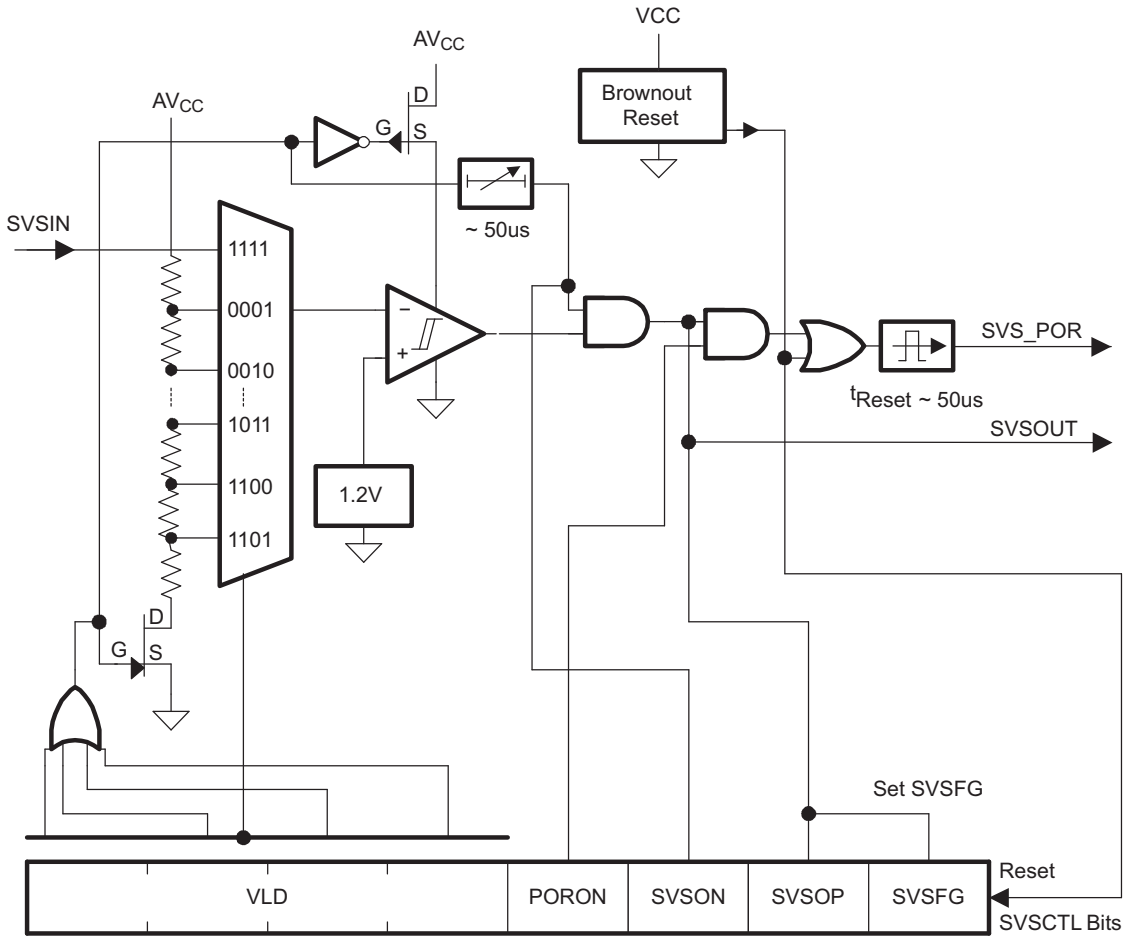


图 **9-1. SVS** 结构图

9.2 SVS 运行

SVS 检测 AV_{CC}电压是否降至一个可选择的水平以下。 发生一个低电压状况时，它可以被配置以便提供一个 POR 或置位一个标志。 在一个掉电复位后禁用 SVS，以减少电流消耗。

9.2.1 配置 SVS

VLDx 位被用于使能/禁用 SVS 和为与 AV_{CC}比较选择 14 个可选择的阈值水平 ($V_{(SVS\_IT-)}$)中的一个。 当 VLDx=0 时，SVS 关闭和 VLDx>0 时，SVS 打开。SVSON 不能打开 SVS。 相反，它反映了 SVS 的打开/关闭状态并且当 SVS 打开时，它可用于决定 SVS 的状态。

当 VLDx=1111 时，外部 SVSIN 通道被选用。 把 SVSIN 上的电压和一个约为 1.25V 的内部电平相比较。

9.2.2 SVS 比较器运行

当 AV_{CC}降至所选阈值以下时，或外部电压降至它的 1.25V 阈值以下时，会出现一个低电压状态。 任何低电压状态都会置位 SVSFG 位。

PORON 位使能或禁用 SVS 的器件复位功能。 如果 PORON=1，那么当 SVSFG 位被置位时，将会产生一个 POR。 如果 PORON=0，一个低电压状态置位 SVSFG，但不会产生一个 POR。

SVSFG 位被锁存。 这将允许用户软件确定之前是否发生了一个低电压状态。 SVSFG 位必须由软件复位。 如果 SVSFG 复位后，低电压状态仍然存在，那么立即被 SVS 再一次置位。

9.2.3 更改 VLDx 位

当 VLDx 位从 0 到其他非零值改变时，它将应用一个自动的稳定延迟 $t_{d\ (SVSon)}$从而允许 SVS 电路去结算。 $t_{d\ (SVSon)}$ 延迟约为 50µs。 在本次延迟期间，SVS 将不会标志一个低电压状态或复位器件，并且 SVSON 被清除。 软件可以检测 SVSON 位来确定延迟何时会过去及 SVS 何时正常的监控电压。 在 SVSON=0 期间，写入 SVSCTL 将会中止 SVS自动的结算延迟，$t_{d(SVSon)}$，并立即切换 SVS 到激活模式。 这样做，SVS 电路可能不会被结算，就会导致不可预知的行为。

当 VLDx 位从 0 到其他非零值改变时，电路需要时间 $t_{稳定}$去结算。 稳定时间 $t_{稳定}$最大为 ~12 µs。 设置特定器件数据表。 没有阻止 SVSFG 被置位或器件复位的自动延迟可使用。 在不同水平间的切换建议流程显示在下列的代码中。

```
; Enable SVS for the first time:MOV.B #080h,&SVSCTL ; Level 2.8V, do not cause POR; ...; Change
SVS levelMOV.B #000h,&SVSCTL ; Temporarily disable SVSMOV.B #018h,&SVSCTL ; Level 1.9V, cause
POR; ...
```

9.2.4 SVS 运行范围

当 AV_{CC}接近阈值时，每一个 SVS 水平都有滞后，以此来降低对小型电源电压改变的敏感度。 在图 9-2中显示了 SVS 运行和 SVS/掉电交互运行。

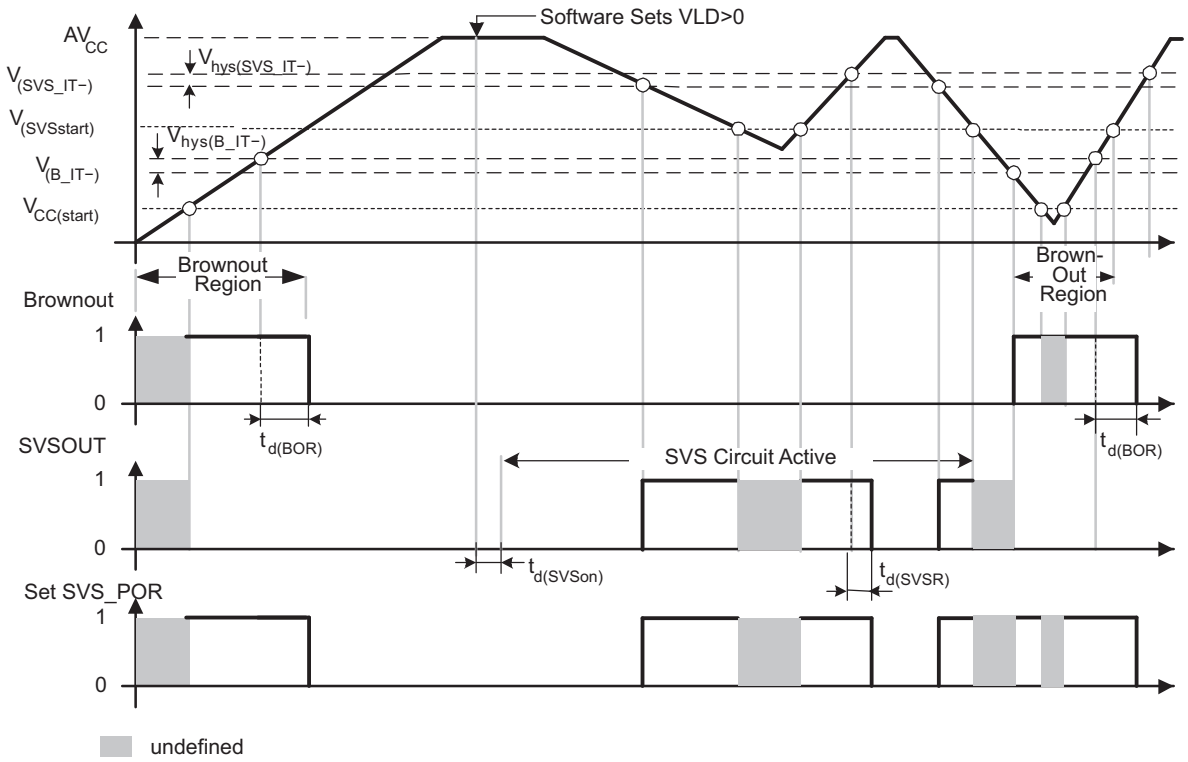


图 **9-2. SVS** 运行水平和掉电**/**复位电路

9.3 SVS 寄存器

在表 9-1中列出了 SVS 寄存器。

表 9-1. SVS 寄存器

寄存器	简式	寄存器类型	地址	初态
SVS 控制寄存器	SVSCTL	读取/写入	055h	用 BOR 复位

9.3.1 SVSCTL, SVS 控制寄存器

7	6	5	4	3	2	1	0
VLDx				PORON	SVSON	SVSOP	SVSFG
rw-0[1]	rw-0[1]	rw-0[1]	rw-0[1]	rw-0[1]	r[1]	r[1]	rw-0[1]

VLDx 位 7-4 电压电平检测 这些位打开 SVS 并选择标称 SVS 阈值电压电平。 有关参数请参阅《器件专用数据表》。

0000 SVS 关闭
0001 1.9V
0010 2.1V
0011 2.2V
0100 2.3V
0101 2.4V
0110 2.5V
0111 2.65V
1000 2.8V
1001 2.9V
1010 3.05V
1011 3.2V
1100 3.35V
1101 3.5V
1110 3.7V
1111 与 1.25V 比较外部输入电压 SVSIN 。

PORON 位 3 POR 打开。 该位通过使能 SVSFG 标志来引起 POR 器件复位。

0 SVSFG 没有导致一个 POR
1 SVSFG 导致一个 POR

SVSON 位 2 SVS 打开。 该位反映了 SVS 的运行状态。 该位没有打开 SVS。 通过设置 VLDx>0 打开 SVS。

0 SVS 被关闭
1 SVS 被打开

SVSOP 位 1 SVS 输出。 该位反映了 SVS 比较器的输出值。

0 SVS 比较器的输出是低电平
1 SVS 比较器的输出是高电平

SVSFG 位 0 SVS 标志。 该位表示一个低电压状态。 在一个低电压状态后 SVSFG 保持置位直到由软件复位。

0 无低电压情况发生
1 低电压情况出现或已经发生

(1) 只能由一个掉电复位来复位，不能由 POR 或 PUC 复位。

安全装置定时器+ (WDT+)

安全装置定时器+（WDT+）是一个 16 位定时器，可以用来作为安全装置或作为一个间隔定时器。 本章介绍了 WDT+。WDT+ 已在所有 MSP430x2xx 器件中执行。

Topic **Page**

10.1 安全装置定时器+ (WDT+) 介绍

安全装置定时器 (WDT+) 模块的主要功能是在软件问题发生后执行受控的系统重启。 如果选定的时间间隔结束，则产生一个系统复位。 如果在一个应用中不需要安全装置功能，则该模块可被禁用或配置为一个间隔定时器，并能在选定的时间间隔内产生中断。

安全装置定时器+ 模块的功能包括：

- 4 个软件可选时间间隔
- 安全装置
- 间隔模式
- WDT+ 控制寄存器的访问受密码保护
- $\overline{\text{RST}}$/NMI 引脚功能的控制
- 可选时钟源
- 可以停止来节省电能
- 时钟故障安全功能

在图 10-1中给出了 WDT+ 的结构图。

注: 安全装置定时器+ 加电有效

在一个 PUC 后，WDT+ 模块在安全装置模式下自动配置，通过使用 DCOCLK，用一个最初的 32768 个时钟周期复位间隔。 用户必须在初始复位间隔期满前设置或暂停 WDT+。

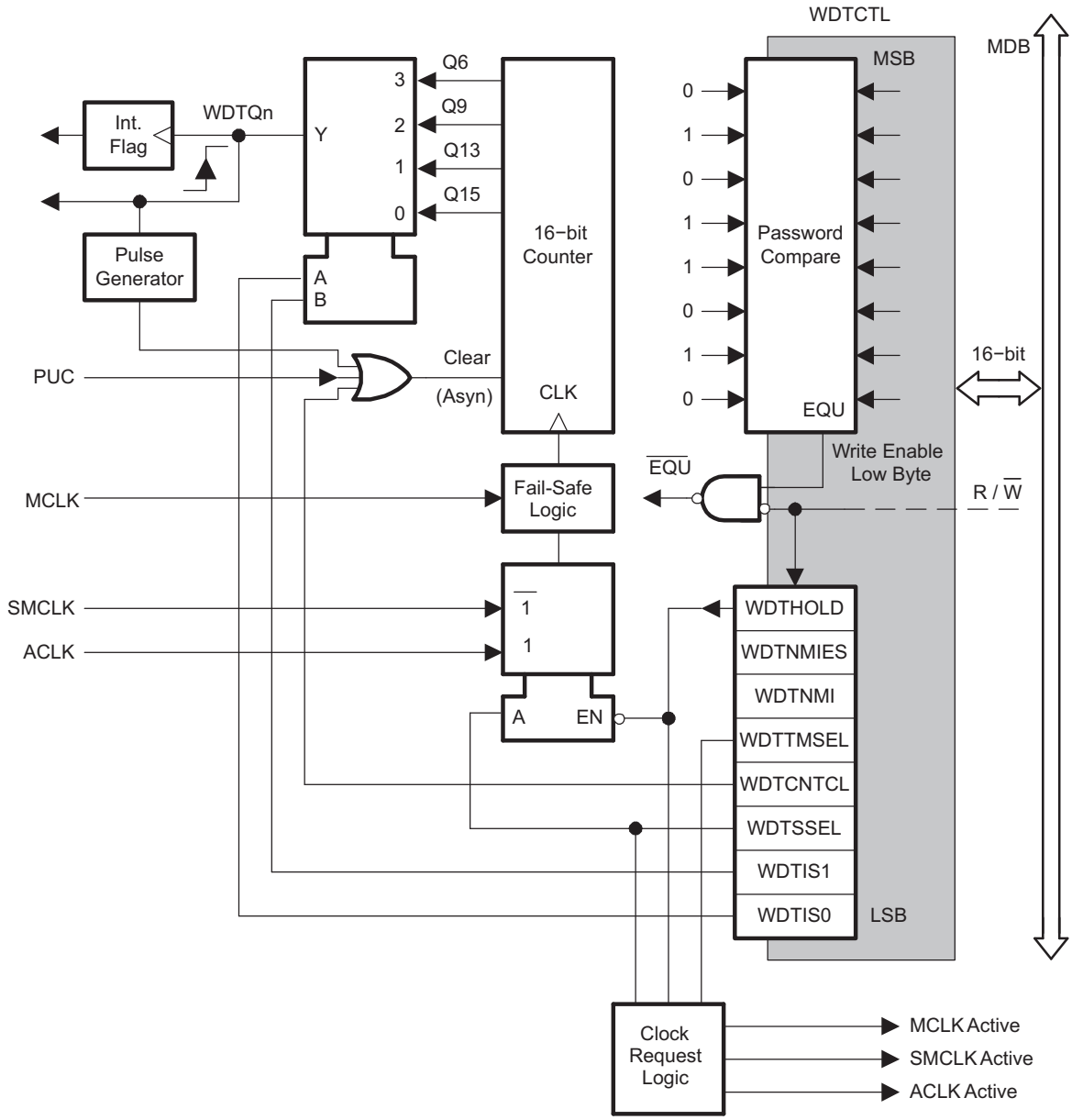


图 **10-1.** 安全装置定时器+ 方框图

10.2 安全装置定时器+ 操作

WDT+ 模块可以被配置为一个安全装置或带有 WDTCTL 寄存器间隔定时器。 WDTCTL寄存器中还包含控制位来配置RST/NMI 引脚。 WDTCTL是一个 16 位的，密码保护的，读取/写入寄存器。 任何读取或写入访问必须使用字指令且写入密码必须包括高字节中的写入密码 05Ah。 任何用除了 05Ah 以外的其他任何高字节值写入 WDTCTL 都是一个安全密钥违反，且会触发一个 PUC 系统复位而与定时器模式无关。 任何 WDTCTL 的读取会读取高字节中的 069h 。 WDT+ 计数器的时钟速度应当较慢或等于系统频率 (MCLK)。

10.2.1 安全装置定时器+ 计数器

安全装置定时器+ 计数器 (WDTCNT) 是一个 16 位的不能直接用软件访问递增计数器。 WDTCNT 通过安全装置定时器+ 控制寄存器 WDTCTL 来控制 WDTCNT 和选择时间间隔。

WDTCNT 可以来源于 ACLK 或 SMCLK。 用 WDTSSEL 位时钟源选择。

10.2.2 安全装置模式

在一个 PUC 条件后，通过使用 DCOCLK 来用一个最初的 32768 个周期复位间隔配置 WDT+ 模块。 用户必须在初始复位间隔或将产生另一个 PUC 到期满前设置、暂停、或清零 WDT+。 当 WDT+ 被配置为在安全装置模式下操作时，要么是用不正确的密码写入 WDTCTL，要么已选时间间隔到期时触发一个 PUC。 一个 PUC 将 WDT+ 复位到其缺省状态并把RST/NMI 引脚配置复位模式。

10.2.3 间隔定时器模式

把 WDTTMSEL 位设置为 1 能选择间隔定时器模式。 这种模式可用于提供周期性中断。 在间隔定时器模式下，在选定时间间隔期满时 WDTIFG 标志被置位。 在间隔定时器模式下，在选定定时器的间隔期满时不会产生一个 PUC，且 WDTIFG 使能位 WDTIE 保持不变。

当 WDTIE 位和 GIE 位被置位时，WDTIFG 标志会请求一个中断。 当中断标志 WDTIFG 的中断请求被服务时它会自动复位时，或能通过软件被置位。 在间隔定时器模式下的中断向量地址与在安全装置模式下是不同的。

注: 修改安全装置定时器+

为了即时 PUC 或中断避免意外，WDT+ 的间隔时间应该与在一个单指令中的 WDTCNTCL=1 一起改变。

为了避免可能出现不正确的时间间隔，在改变时钟源前应暂停 WDT+。

10.2.4 安全装置定时器+ 的中断

WDT+ 为中断控制使用了 SFR 中的两个位。

- WDT+ 中断标志，WDTIFG，位于 IFG1.0。
- WDT+ 中断使能，WDTIE，位于 IE1.0。

当在安全装置模式下使用 WDT+ 时，WDTIFG 标志源自一个复位向量中断。 通过复位中断服务子例程，该 WDTIFG 可用于以确定安全装置是否引起了器件复位。 如果该标志被置位，那么安全装置定时器+ 就会要么通过超时要么通过一个违反安全密钥来初始化复位状态。 如果 WDTIFG 被清零，那么复位就是由一个不同的源引起的。

在间隔定时器模式下使用 WDT+，如果 WDTIE 和 GIE 位被置位，在选定的时间间隔请求一个 WDT+ 间隔定时器中断后，WDTIFG 标志就会被置位。 间隔定时器中断向量与安全装置下模式下的复位向量是不同的。 在间隔定时器模式下，当中断得到服务时 WDTIFG 标志自动复位，或可以用软件复位。

10.2.5 安全装置定时器+ 时钟故障安全操作

WDT+ 模块提供了一个失效安全时钟功能，来确保在安全装置模式下 WDT+ 的时钟不能被禁用。 这意味着 WDT+ 时钟选择可能会受到影响低功耗模式。 例如，如果 ACLK 是 WDT+ 的时钟源，LPM4 将无法使用，因为 WDT+ 将防止 ACLK 被禁用。 另外，如果 ACLK 或 SMCLK 在来自 WDT+ 时失败，WDT+ 的时钟源会自动切换到 MCLK。 在这种情况下，如果 MCLK 来源于晶振，但晶体已失效，那么失效保护功能将激活 DCO 并把它用作为 MCLK 的源。

当 WDT+ 模块用于在间隔定时器模式下时，没有时钟源的失效安全功能。

10.2.6 在低功耗模式下的操作

MSP430 器件具有多种低功耗模式。 不同的时钟信号可在不同的低功耗模式中使用。 用户的应用程序要求和使用的时钟类型决定了应如何配置 WDT+。 例如，如果用户想使用低功耗模式 3，就不应该在安全装置模式下把 SMCLK 配置为 WDT+ 的时钟源，因为 WDT+ 将为保持 SMCLK 的其时钟源一直使能 SMCLK，这将增加了 LPM3 的电流消耗。 当对安全装置定时器+ 没有要求时，WDTHOLD 位可以用来保持 WDTCNT，这就降低了功耗。

10.2.7 软件示例

任何对 WDTCTL 的写入操作都必须为一个字的高字节 05Ah 操作 (WDTPW):

```
; Periodically clear an active watchdogMOV #WDTPW+WDTCNTCL,&WDTCTL;; Change watchdog timer+
intervalMOV #WDTPW+WDTCNTL+WDTSSEL,&WDTCTL;; Stop the watchdogMOV #WDTPW+WDTHOLD,&WDTCTL;; Change
WDT+ to interval timer mode, clock/8192 intervalMOV #WDTPW+WDTCNTCL+WDTTMSEL+WDTIS0,&WDTCTL
```

10.3 安全装置定时器+ 寄存器

在表 10-1中列出了 WDT+ 寄存器。

表 **10-1.** 安全装置定时器+ 寄存器

寄存器	简式	寄存器类型	地址	初始化状态
安全装置定时器+ 控制寄存器	WDTCTL	读取/写入	0120h	06900h 与 PUC
SFR 中断使能寄存器 1	IE1	读取/写入	0000h	用 PUC 复位
SFR 中断标志寄存器 1	IFG1	读取/写入	0002h	用 PUC 复位(1)

(1) 用 POR 复位 WDTIFG。

10.3.1 WDTCTL，安全装置定时器+ 寄存器

15	14	13	12	11	10	9	8
WDTPW，读为 069h 必须写为 05Ah							

7	6	5	4	3	2	1	0
WDTHOLD	**WDTNMIES**	**WDTNMI**	**WDTTMSEL**	**WDTCNTCL**	**WDTSSEL**	**WDTISx**	
rw-0	rw-0	rw-0	rw-0	r0(w)	rw-0	rw-0	rw-0

WDTPW 位 15-8 安全装置定时器+ 密码。 总是读为 069h。 必须写为 05Ah，否则就会产生一个 PUC。

WDTHOLD 位 7 安全装置定时器+ 保持。 该位阻止了安全装置定时器+。 设置 WDTHOLD=1 时， 在不使用 WDT+ 时节省了功耗。

0 安全装置定时器+ 没被停止。

1 安全装置定时器+ 被停止。

WDTNMIES 位 6 安全装置定时器+ 的 NMI 沿选择。 当 WDTNMI=1 时，该位为 NMI 中断选择中断边沿。修改该位可以触发一个 NMI。 为了避免引发意外 NMI，当 WDTIE=0 时，修改该位。

0 上升沿上的 NMI

1 下降沿上的 NMI

WDTNMI 位 5 安全装置定时器+ NMI 选择。 该位为$\overline{\text{RST}}$NMI 引脚选择功能。

0 复位功能

1 NMI 功能

WDTTMSEL 位 4 安全装置定时器+ 模式选择

0 安全装置模式

1 间隔定时器模式

WDTCNTCL 位 3 安全装置定时器+ 计数清零。 设置 WDTCNTCL=1，清零计数值到 0000h。 WDTCNTCL 被自动复位。

0 无操作

1 WDTCNT=0000h

WDTSSEL 位 2 安全装置定时器+ 时钟源选择。

0 SMCLK

1 ACLK

WDTISx 位 1-0 安全装置定时器+ 间隔选择。 这些位选择安全装置定时器+ 的时间间隔来设置 WDTIFG 的标志和/或来产生一个 PUC。

00 安全装置时钟源 /32768

01 安全装置时钟源 /8192

10 安全装置时钟源 /512

11 安全装置时钟源 /64

10.3.2 IE1，中断使能寄存器 1

7	6	5	4	3	2	1	0
			NMIIE				**WDTIE**
							rw-0

	位 7-5	这些位也可以被其他模块使用。 请参阅《器件专用数据表》。
NMIIE	位 4	NMI 中断使能。 该位启用 NMI 中断。 由于 IE1 中的其他位可以用于其他模块中，建议使用 BIS.B 或 BIC.B 指令，而不是 MOV.B 或 CLR.B 指令来设置或清零该位。 0 中断未被启用 1 中断被启用
	位 3-1	这些位也可以被其他模块使用。 请参阅《器件专用数据表》。
WDTIE	位 0	安全装置定时器+ 中断使能。 该位启用间隔定时器模式下的 WDTIFG 中断。 没有必要为安全装置模式设置此位。 由于 IE1 中的其他位可以用于其他模块中，建议使用 BIS.B 或 BIC.B 指令，而不是 MOV.B 或 CLR.B 指令来设置或清零该位。 0 中断未被启用 1 中断被启用

10.3.3 IFG1，中断标志寄存器 1

7	6	5	4	3	2	1	0
			NMIIFG				**WDTIFG**
			rw-0				rw-(0)

	位 7-5	这些位也可以被其他模块使用。 请参阅《器件专用数据表》。
NMIIFG	位 4	NMI 中断标志。 NMIIFG 必须由软件复位。 由于 IFG1 中的其他位可以用于其他模块中，建议使用 BIS.B 或 BIC.B 指令，而不是 MOV.B 或 CLR.B 指令清零 NMIIFG。 0 无中断等待 1 中断等待
	位 3-1	这些位也可以被其他模块使用。 请参阅《器件专用数据表》。
WDTIFG	位 0	安全装置定时器+ 中断使能。 在安全装置模式下，WDTIFG 保持直到被软件复位。 在间隔模式下，通过处理中断来自动复位 WDTIFG，或者可以由软件复位。 由于 IFG1 中的其他位可以用于其他模块中，建议使用 BIS.B 或 BIC.B 指令，而不是 MOV.B 或 CLR.B 指令来清零 WDTIFG 。 0 无中断等待 1 中断等待

硬件乘法器

本章介绍了硬件乘法器。 硬件乘法器用在一些 MSP430x2xx 器件中。

Topic **Page**

11.1 硬件乘法器介绍

硬件乘法器是一个外设和但不是 MSP430 CPU 的一部分。 这意味着，它的活动不干扰 CPU 的活动。 乘法器寄存器是用 CPU 指令加载和读取的外设寄存器。

硬件乘法器支持：

- 无符号的乘法
- 有符号的乘法
- 无符号乘法累加
- 有符号的乘法累加
- 16×16 位，16×8 位，8×16 位，8×8 位

在图 11-1中给出了硬件乘法器的方框图。

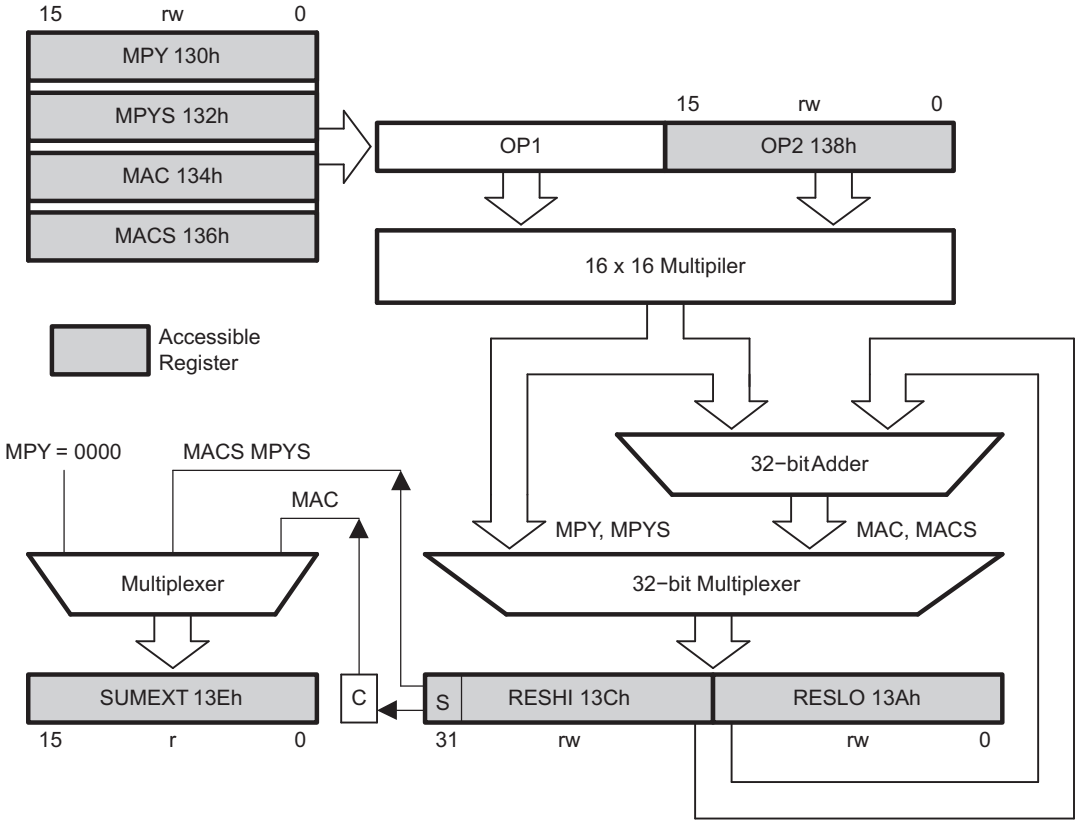


图 11-1. 硬件乘法器方框图

11.2 硬件乘法器操作

硬件乘法器支持无符号乘法，有符号乘法，无符号乘法累加，和有符号乘法累加操作。 通过寻址写入的第一个操作数来选择操作类型。

硬件乘法器有两个 32 位操作数寄存器，OP1 和 OP2，和三个结果寄存器，RESLO，RESHI，和 SUMEXT。 RESL0 用于存储结果的低字，RESHI 用于存储结果的高字，和 SUMEXT 用于存储结果的信息。 除非使用一个间接寻址模式来访问结果，在 3 个 MCLK 周期内结果就可以准备就绪且在写入 OP2 后可用下一条指令读取。 当使用对结果间接寻址时，在结果准备就绪之前需要一个 NOP。

11.2.1 操作数寄存器

一个操作数寄存器 OP1 有四个地址，已在表 11-1中给出，用于选择乘法器模式。 把第一个操作数写入指定的地址会选择乘法运算的类型，但不启动任何操作。 把第二个操作数写入两个操作数寄存器 OP2 会初始化乘法运算。 写入 OP2 会用存储在 OP1 和 OP2 中的值启动所选操作。 结果将被写入这 3 个结果寄存器：RESLO，RESHI，和 SUMEXT。

如果 OP1 的值用于连续操作，则重复的乘法操作可以无需重新加载 OP1 进行 。 没有必要为了执行操作而重新写入 OP1 值。

表 **11-1. OP1** 的各地址

OP1 的各地址	寄存器名称	运行
0130h	MPY	无符号乘法
0132h	MPYS	有符号乘法
0134h	MAC	无符号乘法累加
0136h	MACS	有符号倍乘加

11.2.2 结果寄存器

结果低寄存器 RESLO 保存了计算结果的低 16 位。 结果高位寄存器 RESHI 的内容取决于乘法运算，并在表 11-2列出。

表 **11-2. RESHI** 的内容

模式	**RESHI** 的内容
MPY	结果的高 16 位
MPYS	MSB 是结果的符号。 剩余位是结果的高 15 位。 二的补码表示法用于结果。
MAC	结果的高 16 位
MACS	结果的高 16 位。 二的补码表示法用于结果。

总和扩展寄存器 SUMEXT 的内容依赖于乘法运算，并在表 11-3列出。

表 **11-3. SUMEXT** 的内容

模式	**SUMEXT**
MPY	SUMEXT 始终是 0000H
MPYS	SUMEXT 包含结果的扩展符号 00000H = 结果为正数或零 0FFFFh = 结果为负
MAC	SUMEXT 包含的结果的进位 0000H = 结果无进位 0001H = 结果有进位
MACS	SUMEXT 包含结果的扩展符号 00000H = 结果为正数或零 0FFFFh = 结果为负

11.2.2.1 MACS 下溢和上溢

该乘法器不会自动检测 MACS 模式中的下溢和上溢。 累加器的为正数范围是 0 到7FFF FFFFh 且为负数范围是 0FFFF FFFFh 到 8000 0000H。 当两个负数的总和产生一个介于一个正数范围的结果时，发生下溢。 当两个正数的总和产生一个介于一个负数范围的结果时，发生上溢。 在这两种情况下，SUMEXT 寄存器包含结果的符号，0FFFFh 是上溢而 0000H 为下溢。 用户必须用软件检测并适当地处理这些条件。

11.2.3 软件示例

下面是所有乘法器模式的例子。 当使用标准定义文件上的标签时，所有 8x8 模式都使用寄存器的绝对地址，因为汇编器将不会允许 B 访问字寄存器。

在软件中没有必要进行符号扩展。 在有符号运行期间用一个字节指令访问乘法器将自动导致一个乘法器模块内的字节的符号扩展。

```
; 16x16 Unsigned MultiplyMOV #01234h,&MPY ; Load first operandMOV #05678h,&OP2 ; Load second
operand; ... ; Process results; 8x8 Unsigned Multiply. Absolute addressing.MOV.B #012h,&0130h ;
Load first operandMOV.B #034h,&0138h ; Load 2nd operand; ... ; Process results; 16x16 Signed
MultiplyMOV #01234h,&MPYS ; Load first operandMOV #05678h,&OP2 ; Load 2nd operand; ... ; Process
results; 8x8 Signed Multiply. Absolute addressing.MOV.B #012h,&0132h ; Load first operandMOV.B
#034h,&0138h ; Load 2nd operand; ... ; Process results; 16x16 Unsigned Multiply AccumulateMOV
#01234h,&MAC ; Load first operandMOV #05678h,&OP2 ; Load 2nd operand; ... ; Process results; 8x8
Unsigned Multiply Accumulate. Absolute addressingMOV.B #012h,&0134h ; Load first operandMOV.B
#034h,&0138h ; Load 2nd operand; ... ; Process results; 16x16 Signed Multiply AccumulateMOV
#01234h,&MACS ; Load first operandMOV #05678h,&OP2 ; Load 2nd operand; ... ; Process results; 8x8
Signed Multiply Accumulate. Absolute addressingMOV.B #012h,&0136h ; Load first operandMOV.B
#034h,R5 ; Temp. location for 2nd operandMOV R5,&OP2 ; Load 2nd operand; ... ; Process results
```

11.2.4 RESLO 的间接寻址

当使用间接或间接自动增量的寻址模式访问的结果寄存器时，在加载第二个操作数和访问一个结果寄存器之间至少需要一条指令。

```
; Access multiplier results with indirect addressingMOV #RESLO,R5 ; RESLO address in R5 for
indirectMOV &OPER1,&MPY ; Load 1st operandMOV &OPER2,&OP2 ; Load 2nd operandNOP ; Need one
cycleMOV @R5+,&xxx ; Move RESLOMOV @R5,&xxx ; Move RESHI
```

11.2.5 使用中断

如果在写入 OP1 后，但写入 OP2 之前发生了一个中断，且乘法器被用于服务该中断，那么就会失去原来的乘法器模式选择且结果是不可预知的。 为了避免这种情况，应在使用硬件乘法器前禁止中断或不在中断服务程序中使用乘法器。

```
; Disable interrupts before using the hardware multiplierDINT ; Disable interruptsNOP ; Required
for DINTMOV #xxh,&MPY ; Load 1st operandMOV #xxh,&OP2 ; Load 2nd operandEINT ; Interrupts may be
enable before ; Process results
```

11.3 硬件乘法器寄存器

在表 11-4中列出了硬件乘法器寄存器。

表 **11-4.** 硬件乘法器寄存器

寄存器	简式	寄存器类型	地址	初始状态
操作数 1 - 乘法	MPY	读取/写入	0130h	未改变
操作数 1 - 有符号乘法	MPYS	读取/写入	0132h	未改变
操作数 1 - 乘法累加	MAC	读取/写入	0134h	未改变
操作数 1 - 有符号乘法累加	MACS	读取/写入	0136h	未改变
操作数 2	OP2	读取/写入	0138h	未改变
结果低字	RESLO	读取/写入	013Ah	未定义
结果高字	RESHI	读取/写入	013Ch	未定义
总和扩展寄存器	SUMEXT	读取	013Eh	未定义

定时器_A

定时器_A 是一个带有复用捕捉/比较寄存器的 16 位定时器/计数器。 本章描述了 MSP430x2xx 器件系列的定时器_A 的运行。

Topic **Page**

12.1 定时器_A 介绍

定时器_A 是具有 3 个捕捉/比较寄存器的 16 位定时器/计数器。 定时器_A 能支持多个捕捉/比较，PWM 输出，和反相时序。 定时器_A 还有广泛的中断功能。 中断可由计数器在溢出条件上产生也可以由每一个捕捉/比较寄存器产生。

定时器_A 功能包括：

- 在四种运行模式下异步 16 位定时器/计数器
- 可选择和可配置的时钟源
- 两个或三个可配置的捕捉/比较寄存器
- 可配置的 PWM 输出功能
- 异步输入和输出锁存
- 对所有定时器_A 中断快速响应的中断向量寄存器

定时器_A 的框图如图 12-1所示。

注: *字数计数的使用*

在本章中使用计数。 这就意味着计数器要在发生操作的地方进行计数。 如果一个特殊值被直接写入计数器，那么相关的动作将不再发生。

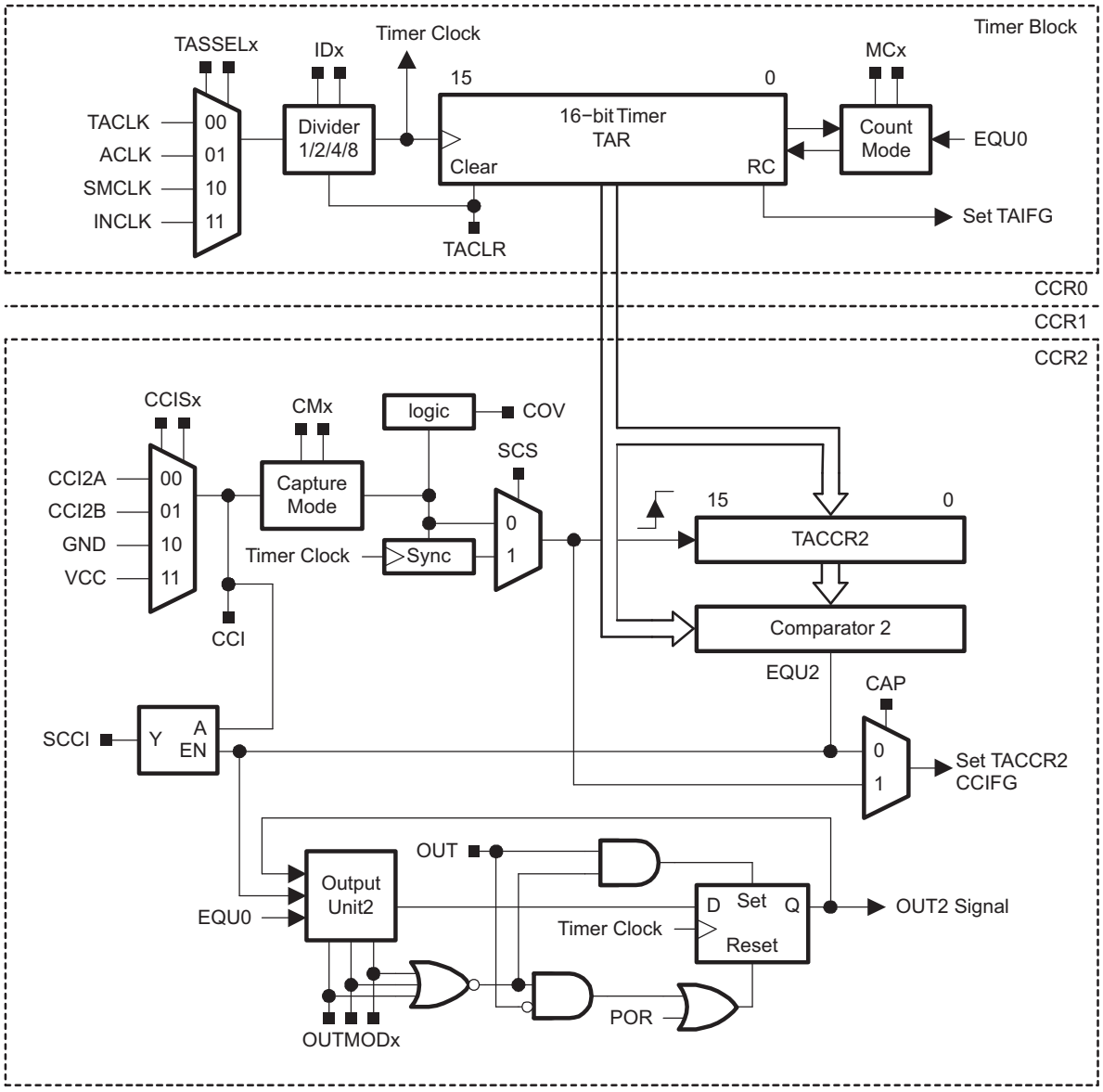


图 **12-1.** 定时器_A 的方框图

12.2 定时器_A 的运行

使用用户软件配置定时器_A 模块。 定时器_A 的建立和运行在下面的部分会进行讨论。

12.2.1 16 位定时计数器

16 位定时/计数器寄存器，TAR，随着每个时钟信号的上升沿增/减（这由操作模式决定）。 TAR 可以通过软件读取或写入。另外，当它溢出时，定时器可以产生一个中断。

TAR 可以通过设置 TACLR 位被清除。 在增/减模式下，设置 TACLR 也可以清除时钟分频器和计数器方向。

注: **修改定时器_A 寄存器**

建议在修改定时器的运行（除了中断使能、和中断标志）前，先停止定时器，以避免产生未知的错误操作。

当定时器时钟与 CPU 时钟异步时，任何对 TAR 的读取都会发生而定时器却不运行或者运行结果是不可信的。 另外，定时器在运行期间需要被多次读取，通过软件多数表决的方式来确定正确的读数。 对 TAR 的任何写入都将立即生效。

12.2.1.1 时钟源选择和分频

定时器的时钟源可以是内部时钟源 ACLK, SMCLK，或外部源 TACLK 和 INCLK。时钟源是由 TASSELx 位来选择的。 所选择的时钟可以通过 IDx 位直接传递给定时器或进行 2、4 或 8 分频。 当 TACLR 被置位时，定时器时钟分频器被复位。

12.2.2 启动定时器

定时器可以已下列的方法启动，或复位：

- 当 MCx＞0 时，定时器计数并且时钟源活跃。
- 当定时器模式为增/减的任一种时，定时器都可以通过把0 写入 TACCR0 而停止。 然后，可以通过将一个非零值写入 TACCR0 来重启定时器。 这种情况下，定时器开始从零向上递增。

12.2.3 定时器模式控制

定时器有四种运行模式，如表 12-1所描述的：停止，增，连续，和增/减。 运行模式由MCx 位选择。

表 12-1. 定时器模式

MCx	模式	说明
00	停止	定时器被暂停
01	向上	定时器从 0 开始到 TACCR0 的值重复计数。
10	连续	定时器从 0 开始到 0FFFFh 重复计数。
11	增加/减少	定时器从 0 开始递增加到 TACCR0的值并返回到 0 重复计数。

12.2.3.1 上数模式

如果定时器周期一定要和 0FFFFh 计数不同，那么就要用到上数模式了。 定时器重复计数增加至比较寄存器 TACCR0 的值，该值定义了周期，正如在图 12-2所示。 周期中定时器计数的数量是 TACCR0+1。 当定时器的值等于 TACCR0 的值时，定时器重新从 0 开始计数。 在上数模式下，当定时器的值大于 TACCR0 的值时，定时器立即从 0 开始重新计数。

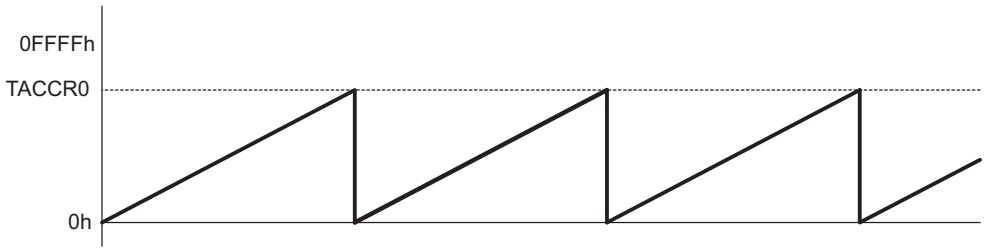


图 12-2. 上数模式

当定时器*计数*到 TACCR0 的值时，TACCR0 CCIFG 中断标志被置位。 当定时器从 *TACCR0 计数*到 0 时，TAIFG 中断标志位被置位。图 12-3显示了标志置位周期。

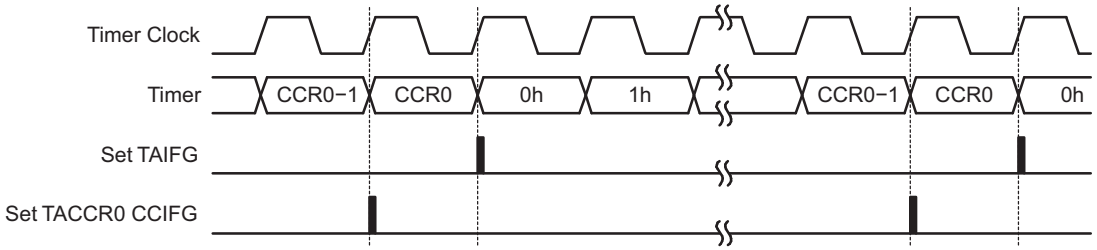


图 **12-3.** 上数模式标志设置

12.2.3.2 改变周期寄存器TACCR0

当在定时器运行时改变 TACCR0 时，如果新周期大于或等于旧周期，或者大于当前的计数值，那么定时器增加至新周期。 如果新周期低于当前计数值，则定时器返回到 0。 然而，在计数器返回到 0 前可能会出现一个额外的计数。

12.2.3.3 连续模式

在连续模式中，定时器重复计数增加至 0FFFFh 并且从0 重新开始，如图 12-4所示。 捕捉/比较寄存器 TACCR0 和其他的捕捉/比较寄存器一样以相同的方式工作。

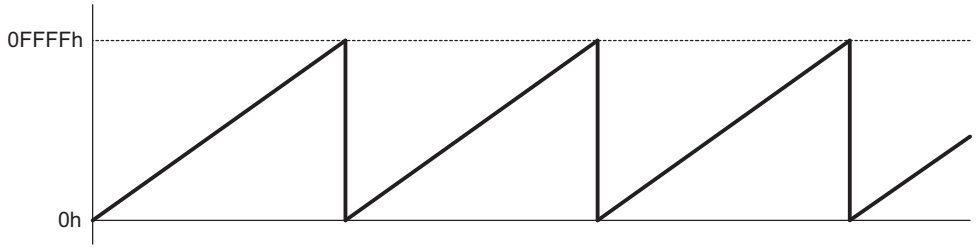


图 **12-4.** 连续模式

当定时器从 *0FFFFh 开始计数*到 0 时，TAIFG 中断标志被置位。图 12-5显示了标志置位周期。

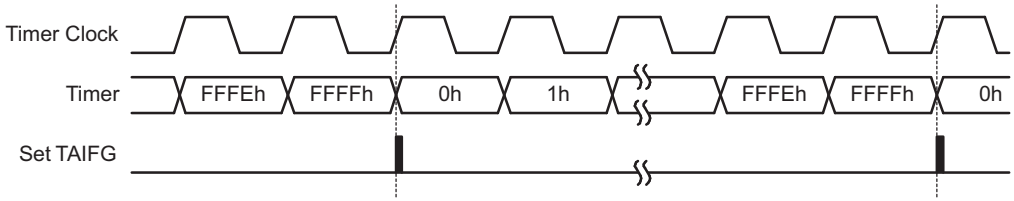


图 **12-5.** 连续模式标志置位

12.2.3.4 连续模式的使用

连续模式可以用于产生独立的时间间隔和输出频率。 当每个时间间隔完成时，就会产生一个中断。 在中断服务程序中，下一个时间间隔被添加到 TACCRx 寄存器中。图 12-6显示了两个独立的时间间隔 t_0和t_1被添加到捕捉/比较寄存器。 在该应用中，时间间隔被硬件控，而不是软件，对中断延迟无影响。 使用所有的 3 个捕捉/比较寄存器可以产生高达3 个独立的时间间隔或输出频率。

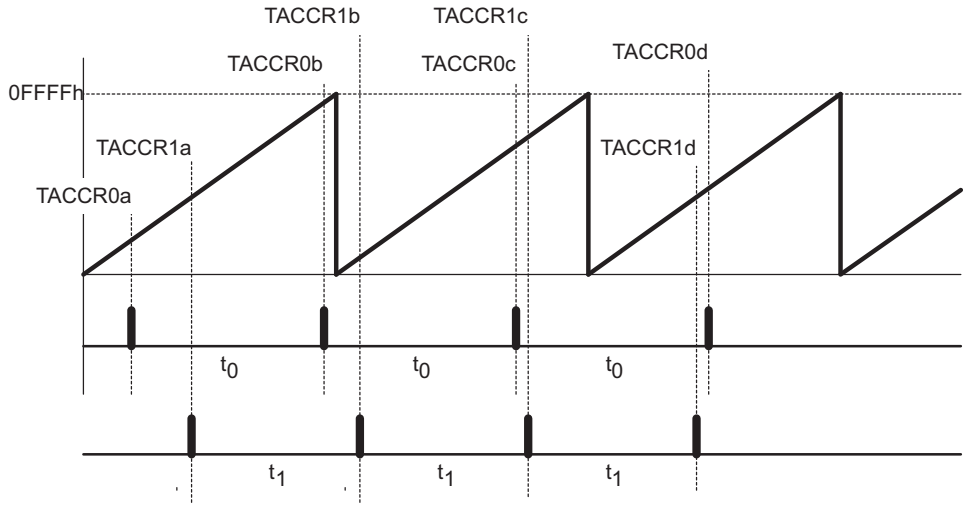


图 **12-6.** 连续模式下的时间间隔

时间间隔在其他模式下也能产生，其中 TACCR0 被作为周期寄存器。 由于旧的 TACCRx 数据和新的周期总和可能高于 TACCR0 值，所以它们的处理更加的复杂。 当以前的 TACCRx 值加上t_x高于 TACCR0 数据时，TACCR0 + 1 必须被减掉以此来获得正确的时间间隔。

12.2.3.5 增加/减少模式

增加/减少模式是在定时器周期不同于 0FFFFH 计数，且需要产生一个对称的脉冲时使用的。 定时器重复计数增加至比较寄存器 TACCR0 的值并且减少到 0，如图 12-7所示。 周期是 TACCR0 中值的两倍。

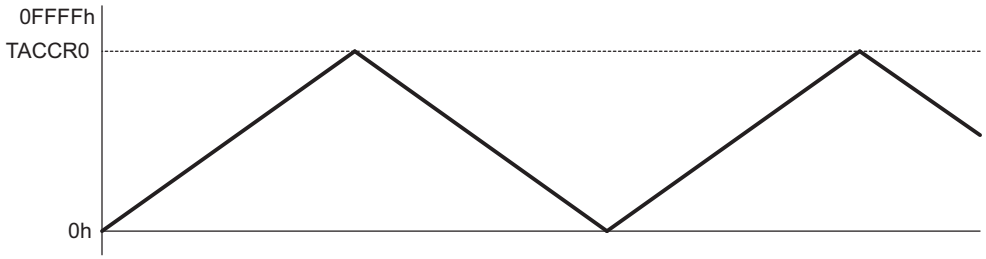


图 **12-7.** 上数/下数模式

计数方向被锁定。 这就允许定时器停止比并且能以它被停止以前的计数方向重新开始。 如果不需要这些，那么必须先将TACLR 置位以清零方向。 TACLR 位同样清除 TAR 值和定时器时钟分频。

在上数/下数模式中，TACCR0 CCIFG 中断标志和 TAIFG 中断标志在一个周期中只置位一次，由 1/2 定时器周期隔开。 当定时器*计数*由 TACCR0−1 到 TACCR0 时， TACCR0 CCIFG 中断标志被置位；而定时器完成*计数*从 0001h 下降到0000h 时，TAIFG被置位。图 12-8显示了标志置位周期。

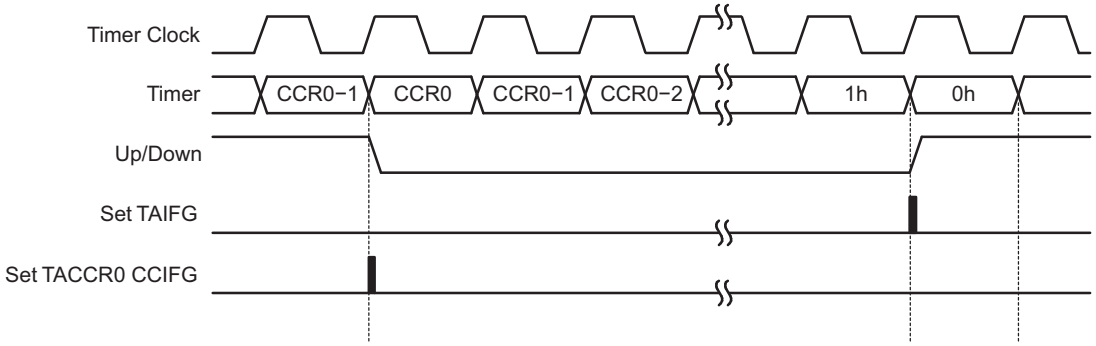


图 **12-8.** 上数/下数模式标志置位

12.2.3.6 改变周期寄存器TACCR0

当定时器运行时，改变 TACCR0 的值，如果正处于计数减少的方向，定时器会继续减少到 0。 TACCR0 中的值被立即锁进 TACL0，然而，新周期会在计数器减少到 0 后生效。

如果正处于增计数状态，并且新周期大于或等于原来的周期，或比当前计数值要大，定时器会在计数下降前增加计数到新周期。 如果正处于增加计数状态，并且新周期小于当前计数值，定时器立刻开始减少计数。但是，在定时器开始减少计数之前会有一个额外的计数出现。

12.2.3.7 上数/下数模式的使用

上数/下数模式支持那些在输出信号之间有空载时间的应用（请参阅《*定时器_A 输出单元*》部分）。 例如，为了避免过载情况，2 个输出驱动一个 H 桥不能同时为高。例子中显示了在图 12-9$t_{死区中}$是：

$t_{死区}=t_{定时器}$(TACCR1 –TACCR2)

其中，

$t_{死区}$=两种输出都不活动的时间段

$t_{定时器}$=定时器时钟的周期时间

TACCRx= 捕捉/比较寄存器 x 的内容

TACCRx 寄存器不被缓冲。 当被写入后，它们立即更新。因此，任何要求的空载时间都不会自动被保留。

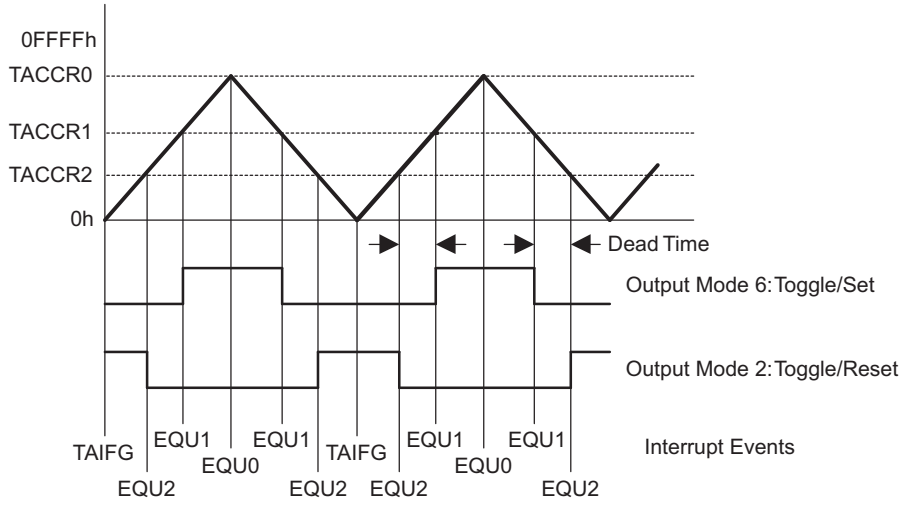


图 **12-9.** 在上数/下数模式中的输出单元

12.2.4 捕捉/比较区块

定时器_A 中有 2 个或 3 个相同的捕捉/比较模块 TACCRx。 任何区块都可能被用于捕捉定时器数据，或产生时间间隔。

捕捉模式

当 CAP=1 时，捕捉模式被选用。捕捉模式被用于记录时间事件。它可被用于速度估计或时间测量。 捕捉输入 CCIxA 和 CCIxB 被连接到外部引脚或内部信号并且由 CCISx 位选择。 CMx 位选择输入信号的捕捉沿作为上升沿，下降沿或两者都是。 捕捉发生在选择的输入信号沿上。 如果发生捕捉：

- 定时器的值被复制仅 TACCRx 寄存器
- 中断标志 CCIFG 被置位。

在任何时刻都可以通过 CCI 位读取输入信号的电平。 MSP430x2xx 系列器件有不同的信号连接到 CCIxA 和 CCIxB。 对于这些信号的连接，请参阅《器件专用数据表》。

捕捉信号可能会和定时器时钟不同步并导致竞争条件的发生。设置 SCS 位使其可以与下个定时器时钟捕捉信号同步。 设置 SCS 位以使其可以与建议的定时器时钟捕捉信号同步。 这显示在图 12-10中。

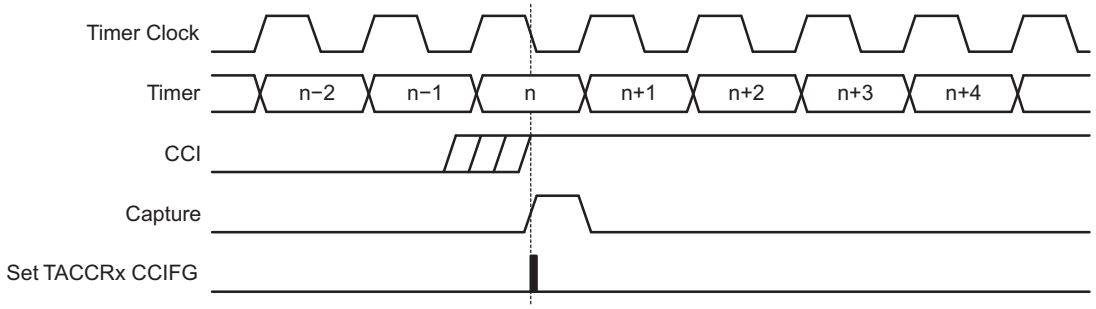


图 12-10. 捕捉信号 (SCS = 1)

如果在第一次捕捉的值被读取之前发生一个第二次捕捉，那么捕捉比较寄存器就会产生一个溢出逻辑。 当这种情况发生时，位 COV 被置位，如图 12-11所示。 COV 位必须有软件复位。

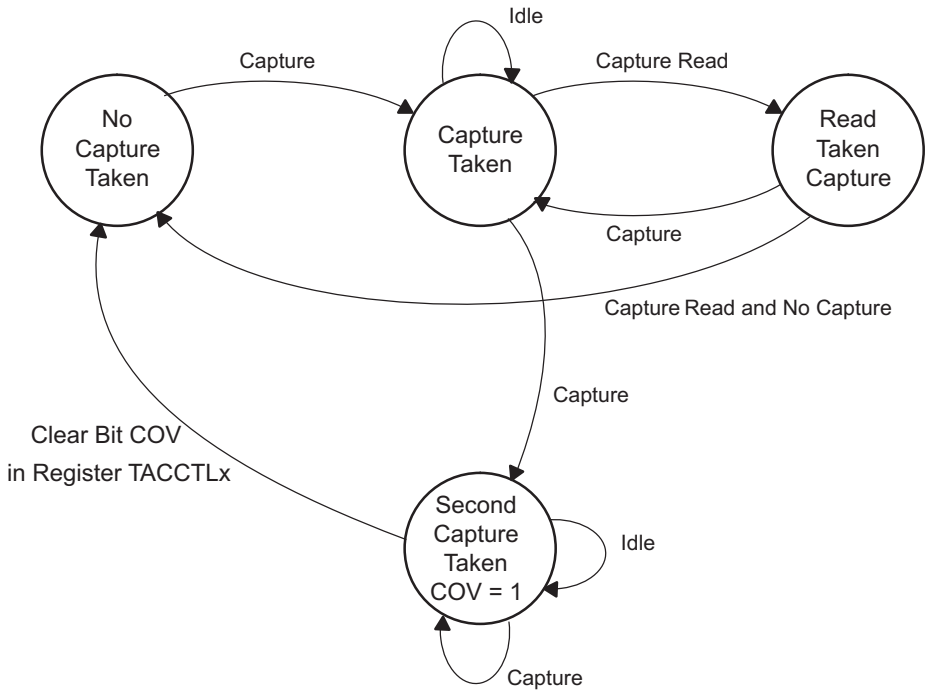


图 **12-11.** 捕捉周期

12.2.4.1 由软件初始化捕捉

捕捉可由软件初始化。 CMx 位可以在两种触发沿上配置捕捉 然后，软件设置 CCIS1=1 和切换位 CCIS0 来切换在V_{CC}和 GND，之间的捕捉信号，每次 CCIS0 改变状态时，都要初始化捕捉器：

```
MOV #CAP+SCS+CCIS1+CM_3,&TACCTLx ; Setup TACCTLxXOR #CCIS0,&TACCTLx ; TACCTLx = TAR
```

12.2.4.2 比较模式

当 CAP=0 时，选用比较模式。比较模式被用于产生 PWM输出信号或在特定的时间间隔上产生中断。 当 TAR*计数*到 TACCRx 中的值时：

- 中断标志 CCIFG 被置位
- 内部信号 EQUx=1
- EQUx 根据输出模式来影响输出信号
- 输入信号 CCI 锁存到 SCCI

12.2.5 输出单元

每一个捕捉/比较区块都包括一个输出单元。 该输出单元被用于产生输出信号，如 PWM 信号。 每个输出单元可以根据 EQU0 和 EQUx 产生8 种模式的信号。

12.2.5.1 输出模式

输出模式由 OUTMODx 位来确定，如表 12-2中所描述的。 对于除了模式 0 以外的所有模式来说，OUTx 信号都是随着定时器时钟的上升沿而改变的。输出模式 2，3，6，和 7 对于输出单元 0 无效，因为在这些模式下，EQUx =EQU0。

表 12-2. 输出模式

OUTMODx	模式	说明
000	输出	输出信号 OUTx 由 OUTx 位定义。 当 OUTx 位更新时，OUTx信号立刻更新。
001	置位	当定时器*计数*到 TACCRx 值时，输出被置位。 它保持置位直到定时器复位或选择了另一个输出模式并影响了输出。
010	切换/复位	当定时器*计数*到 TACCRx 值时，输出被切换。 当定时器*计数*到 TACCR0 值时，它被复位。
011	置位/复位	当定时器*计数*到 TACCRx 值时，输出被置位。 当定时器*计数*到 TACCR0 值时，它被复位。
100	切换	当定时器*计数*到 TACCRx 值时，输出被切换。 输出周期是定时器周期的二倍。
101	复位	当定时器*计数*到 TACCRx 值时，输出被复位。 它保持复位直到选用另外一种输出模式并且影响到了输出。
110	切换/置位	当定时器*计数*到 TACCRx 值时，输出被切换。 当定时器*计数*到 TACCR0 值时，它被置位。
111	复位/置位	当定时器*计数*到 TACCRx 值时，输出被复位。 当定时器*计数*到 TACCR0 值时，它被置位。

12.2.5.2 输出举例—在单调增加模式中的定时器

当定时器*计数*增加到 TACCRx 的值，并从 TACCR0 返回到 0 时，OUTx 信号根据输出模式而改变。 在图 12-12中显示了一个使用了 TACCR0 和 TACCR1的例子。

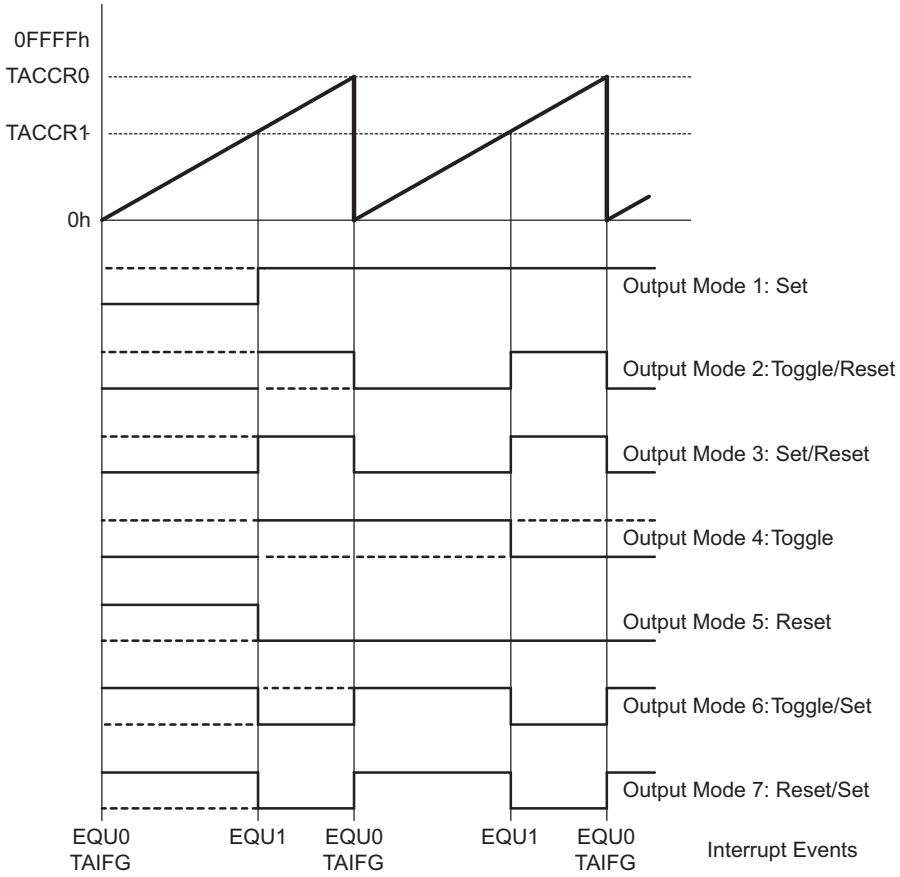


图 12-12. 输出举例—在单调增加模式中的定时器

12.2.5.3 输出举例—在连续模式中的定时器

当定时器计数达到到 TACCRx 和 TACCR0 时，OUTx 信号会根据选择的输出模式发生改变。 在图 12-13中显示了一个使用了 TACCR0 和 TACCR1 的例子。

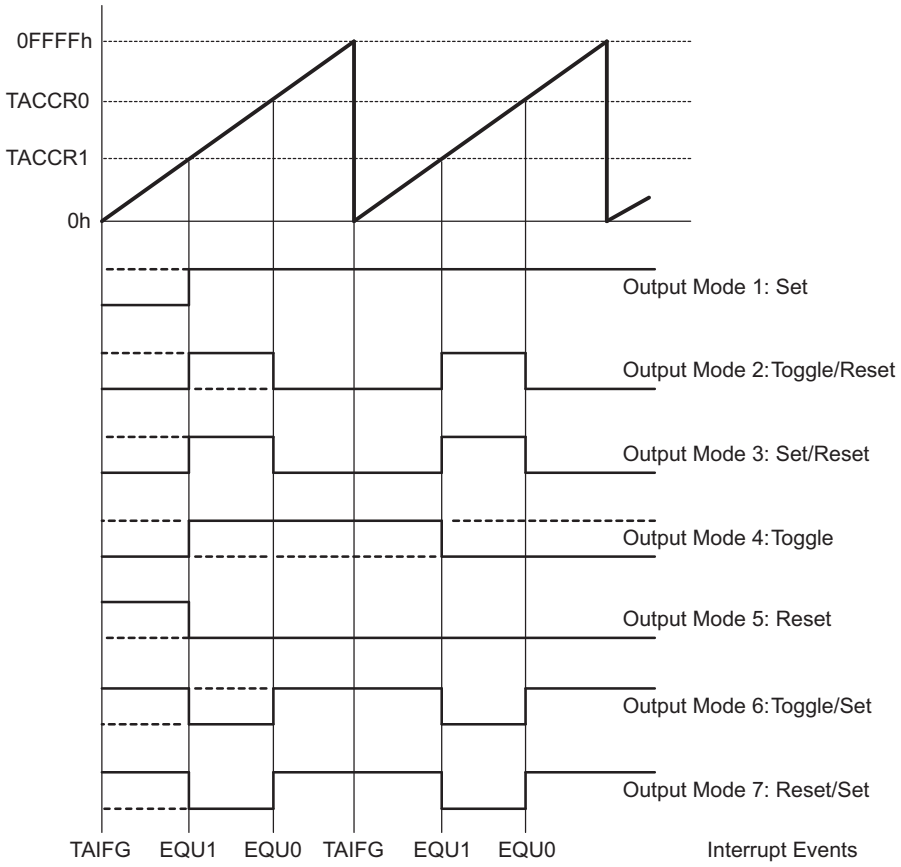


图 **12-13.** 输出举例—在连续模式中的定时器

12.2.5.4 输出举例—在增加/减少 模式中的定时器

当定时器的值在任一计数方向上出现了等于 TACCRx 和TACCR0 的值时，OUTx 信号都会按选择的输出模式发生改变。 在图 12-14中显示了一个使用了 TACCR0 和 TACCR2 的例子。

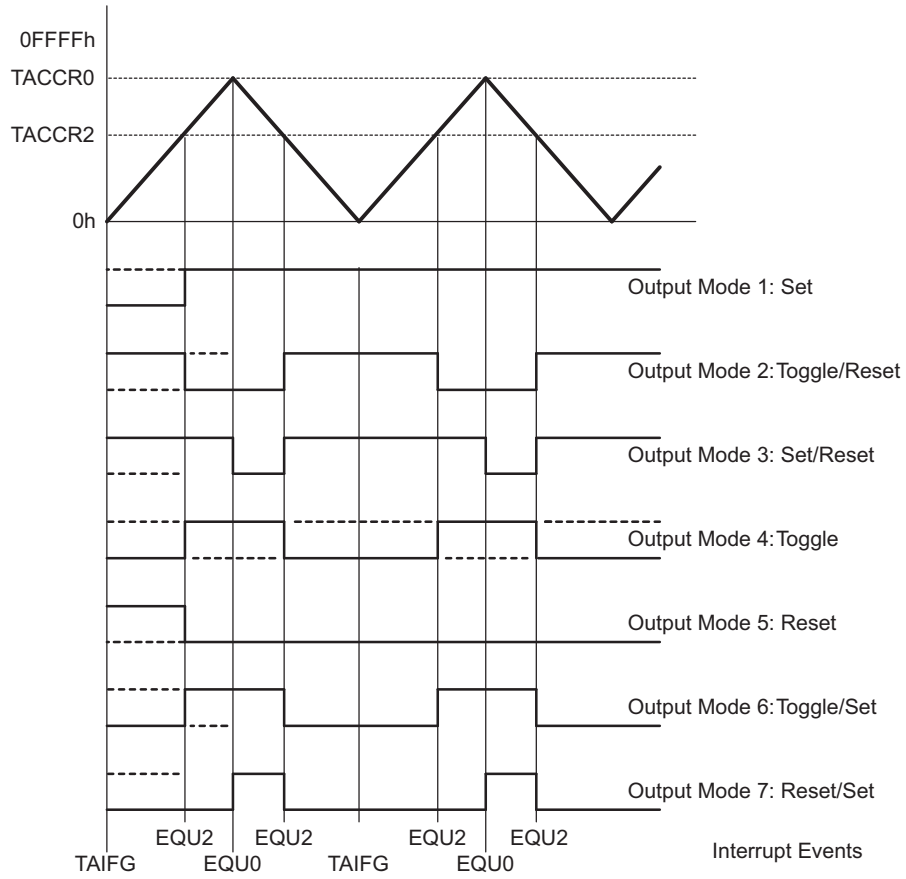


图 **12-14.** 输出举例—在上数/下数模式中的定时器

注: 在输出模式间切换

当在输出模式之间进行切换时，OUTMODx 的一个位必须在过渡时保持置位，除非是切换到模式 0。否则，会因为一个NOR 门解码输出模式 0 而导致出现输出干扰。在输出模式之间切换的一个安全方法就是用输出模式 7 作为过渡状态。

```
BIS #OUTMOD_7,&TACCTLx ; Set output mode=7BIC #OUTMODx, &TACCTLx ; Clear
unwanted bits
```

12.2.6 定时器_A 中断

16 位定时器_A 和 2 个中断向量相关联：

- TACCR0 CCIFG 的 TACCR0 中断向量
- 所有的其他 CCIFG 标志和 TAIFG 的 TAIV 中断向量

在捕捉模式下，当一个定时器的值在其相关的TACCRx 寄存器被捕捉时， CCIFG 标志被置位。 在比较模式下，如果 TAR*计数*到相应的 TACCRx 值时，CCIFG 标志被置位。 软件也可以清除或置位任何CCIFG 标志。 当相应的 CCIE 位和 GIE 位被置位时，所有的 CCIFG 标志都会请求一个中断。

12.2.6.1 TACCR0 中断

TACCR0 CCIFG 标志拥有最高的定时器_A 中断优先级，并有一个专用的中断向量，如图 12-15所示。 当进入 TACCR0 中断后，TACCR0 CCIFG 标志自动复位。

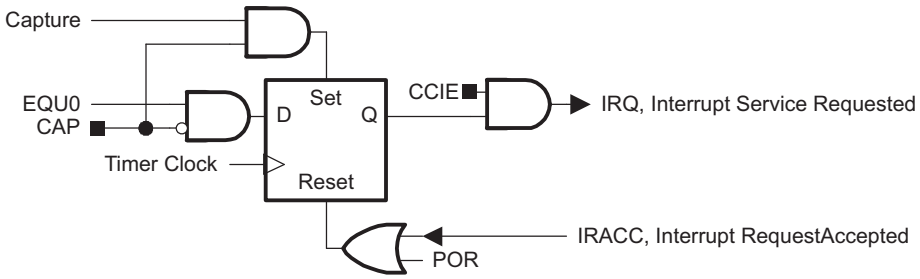


图 **12-15.** 捕捉/比较 **TACCR0** 中断标志

12.2.6.2 TAIV，中断向量发生器

TACCR1 CCIFG，TACCR2 CCIFG，和 TAIFG 标志被优先连接到一个单独的中断向量源。 中断向量寄存器 TAIV 用于确定它们中的哪个标志响应中断请求。

最高优先级在 TAIV 寄存器中产生一个数字使能中断（见寄存器说明）。 此数字被评估并被添加到项目计数器中从而自动的进入相应的子程序。 禁用定时器_A 不会影响TAIV 的值。

对 TAIV 寄存器的任何访问，读取或写入都会自动复位最高优先级的挂起中断标志。 如果另一个中断标志置位，在结束原有的中断响应后会，该中断响应立即发生。 例如，当中断服务子程序访问 TAIV 寄存器时，如果 TACCR1 和 TACCR2 CCIFG 标志被置位，则 TACCR1 CCIFG 自动复位。 在中断服务子程序的 RETI 命令执行后，TACCR2CCIFG 标志会产生另一个中断。

12.2.6.3 TAIV 软件举例

下列软件例子说明了 TAIV 和处理开销的建议方法。 TAIV 的值被加入到 PC 来自动跳转到相应的子程序。

右边空白处的数字表明了 CPU 的每条指令所需要的周期。不同的中断源的软件开销包括中断延迟和返回中断周期，但并不包含任务本身的执行时间。 延迟是：

- 捕捉比较/模块 TACCR0：11 个周期
- 捕捉比较/模块 TACCR1，TACCR2:16 个周期
- 定时器溢出 TAIFG：14 个周期

```
; Interrupt handler for TACCR0 CCIFG CyclesCCIFG_0_HND; ... ; Start of handler Interrupt latency
6RETI 5; Interrupt handler for TAIFG, TACCR1 and TACCR2 CCIFGTA_HND... ; Interrupt latency 6ADD
&TAIV,PC ; Add offset to Jump table 3RETI ; Vector 0: No interrupt 5JMP CCIFG_1_HND ; Vector 2:
TACCR1 2JMP CCIFG_2_HND ; Vector 4: TACCR2 2RETI ; Vector 6: Reserved 5RETI ; Vector 8: Reserved
5TAIFG_HND ; Vector 10: TAIFG Flag... ; Task starts hereRETI 5CCIFG_2_HND ; Vector 4: TACCR2... ;
Task starts hereRETI ; Back to main program 5CCIFG_1_HND ; Vector 2: TACCR1... ; Task starts
hereRETI ; Back to main program 5
```

12.3 定时器_A 寄存器

定时器_A 寄存器在表 12-3中列出。

表 12-3. 定时器_A3 寄存器

寄存器	简氏	寄存器类型	地址	初态
定时器_A 控制	TACTL	读取/写入	0160h	用 POR 复位
定时器_A 计数器	TAR	读取/写入	0170h	用 POR 复位
定时器_A 捕捉/比较控制 0	TACCTL0	读取/写入	0162h	用 POR 复位
定时器_A 捕捉/比较 0	TACCR0	读取/写入	0172h	用 POR 复位
定时器_A 捕捉/比较控制 1	TACCTL1	读取/写入	0164h	用 POR 复位
定时器_A 捕捉/比较 1	TACCR1	读取/写入	0174h	用 POR 复位
定时器_A 捕捉/比较控制 2	TACCTL2(1)	读取/写入	0166h	用 POR 复位
定时器_A 捕捉/比较 2	TACCR2(1)	读取/写入	0176h	用 POR 复位
定时器_A 中断矢量	TAIV	只读	012Eh	用 POR 复位

(1) 像MSP430F20xx 和其他器件一样，MSP430 器件上没有定时器_A2。

12.3.1 TACTL，定时器_A 控制寄存器

15	14	13	12	11	10	9	8
未使用						**TASSELx**	
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
IDx		**MCx**		未使用	**TACLR**	**TAIE**	**TAIFG**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

未使用	位 15-10	未使用
TASSELx	位 9-8	定时器_A 时钟源选择 00 TACLK 01 ACLK 10 SMCLK 11 INCLK（INCLK 是器件专用并且常被分配到反相的 TBCLK）（请参阅《器件专用数据表》）
IDx	位 7-6	输入分频器。 这些位为输入时钟选择分频器。 00 /1 01 /2 10 /4 11 /8
MCx	位 5-4	模式控制。 当定时器_A 在不使用时，设置 MCX=00h 从而节省了功耗。 00 停止模式：定时器被暂停。 01 上数模式：定时器计数增加至 TACCR0。 10 连续模式：定时器计数增加至 0FFFFh。 11 上数/下数模式：定时器增加至 TACCR0 然后减至 0000h。
未使用	位 3	未使用
TACLR	位 2	定时器_A 清零 将这些位置位复位 TAR，时钟分频器，和计数方向。 TACLR 位自动复位并且总是读取为 0。
TAIE	位 1	定时器_A 中断使能。 这些位启用 TAIFG 中断请求。 0 中断被禁用 1 中断被启用
TAIFG	位 0	定时器_A 中断标志 0 无中断等待 1 中断等待

12.3.2 TAR，定时器_A 寄存器

15	14	13	12	11	10	9	8
TARx							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
TARx							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

TARx	位 15-0	定时器_A 寄存器。 TAR寄存器是定时器_A 的计数。

12.3.3 TACCRx，定时器_A 捕捉/比较寄存器x

15	14	13	12	11	10	9	8
TACCRx							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
TACCRx							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

TACCRx	位 15-0	定时器_A 捕捉/比较寄存器。 比较模式：TACCRx 保存用于与定时器_A 寄存器中的定时器值进行比较的数据，TAR。 捕捉模式：当一个捕捉被执行时，定时器_A 寄存器，TAR，被复制进 TACCRx 寄存器。

12.3.4 TACCTLx，捕捉/比较控制寄存器

15	14	13	12	11	10	9	8
CMx		**CCISx**		**SCS**	**SCCI**	未使用	**CAP**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	r0	rw-(0)
7	6	5	4	3	2	1	0
OUTMODx			**CCIE**	**CCI**	**OUT**	**COV**	**CCIFG**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	rw-(0)	rw-(0)	rw-(0)

CMx 位 15-14 捕捉模式
- 00 无捕捉
- 01 上升沿上的捕捉
- 10 下降沿上的捕捉
- 11 上升沿和下降沿二者的捕捉

CCISx 位 13-12 捕捉/比较输入选择 这些位选择 TACCRx 输入信号。 有关特定信号的连接请参阅《器件专用数据表》。
- 00 CCIxA
- 01 CCIxB
- 10 GND
- 11 V_{CC}

SCS 位 11 同步捕捉源。 该位被用于使用定时器时钟同步捕捉输入信号。
- 0 不同步捕捉
- 1 同步捕捉

SCCI 位 10 同步捕捉/比较输入 所选择的 CCI 输入信号由EQUx 信号锁存，并可通过该位读取

未使用 位 9 未使用。 只读。 总是读取为 0。

CAP 位 8 捕捉模式
- 0 比较模式
- 1 捕捉模式

OUTMODx 位 7-5 输出模式。 模式 2，3，6 和 7 不能用于 TACCR0，这是因为EQUx = EQU0。
- 000 OUT 位值
- 001 置位
- 010 切换/复位
- 011 置位/复位
- 100 切换
- 101 复位
- 110 切换/置位
- 111 复位/置位

CCIE 位 4 捕捉/比较中断使能。 该位启用相应 CCIFG 标志的中断请求。
- 0 中断被禁用
- 1 中断被启用

CCI 位 3 捕捉/比较输出。 选择的输入信号可以通过该位读取。

OUT 位 2 输出。 在输出模式 0 中，该位直接控制输出的状态。
- 0 输出低电平
- 1 输出高电平

COV 位 1 捕捉溢出。 该位表明一个捕捉溢出的发生。 COV 位必须由软件复位。
- 0 无捕捉溢出发生
- 1 捕捉溢出发生

CCIFG 位 0 捕捉/比较中断标志
- 0 无中断等待
- 1 中断等待

12.3.5 TAIV，定时器_A 中断矢量寄存器

15	14	13	12	11	10	9	8
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**TAIVx**			**0**
r0	r0	r0	r0	r-(0)	r-(0)	r-(0)	r0

TAIVx 位 15-0 定时器_A 中断矢量值

TAIV 内容	中断源	中断标志	中断优先级
00h	无中断挂起	-	
02h	捕捉/比较 1	TACCR1 CCIFG	最高：
04h	捕捉/比较 2(1)	TACCR2 CCIFG	
06h	被保留	-	
08h	被保留	-	
0Ah	定时器溢出	TAIFG	
0Ch	被保留	-	
0Eh	被保留	-	最低

(1) 没有在 MSP430x20xx器件中执行。

定时器_B

定时器_B 是一个带有多个捕捉/比较寄存器的 16 位定时器/计数器。 本章主要讲述了 MSP430x2xx 器件系列定时器_B 的操作。

Topic **Page**

13.1 定时器_B 的介绍

定时器_B 是一个带有 3 个或 7 个捕捉/比较寄存器的 16 位定时器/计数器。 定时器_B 能支持多个捕获/比较寄存器，PWM 输出和间隔定时。 定时器_A 也具有扩展的中断功能。 计数器在溢出发生时可生成中断而每个捕获/比较寄存器也可生成中断。

定时器_B 的特性包括：

- 4 种操作模式和 4 个可选长度的异步 16 位定时器/计数器
- 可选和可配置的时钟源
- 3 个或 7 个可配置的捕获/比较寄存器
- 具有 PWM 功能的可配置输出
- 加载同步的双缓冲比较锁存器
- 对所有定时器_B 中断快速响应的中断向量寄存器

在图 13-1中给出了定时器_B 的方框图。

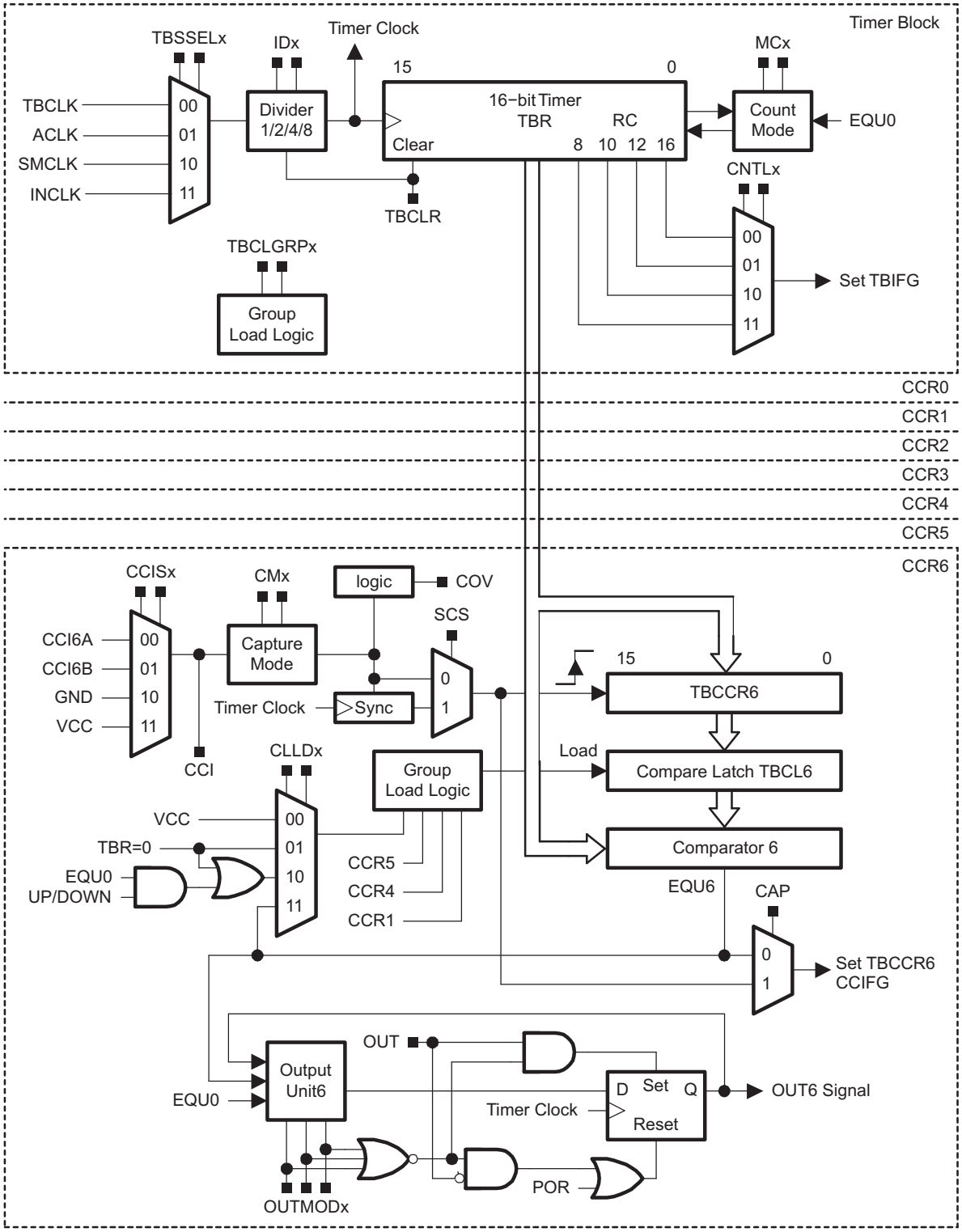
注: *字数的使用*

本章中使用了计数。 这意味着计数器必须在在计数动作的过程中计数。 如果一个特定的值被直接写入计数器，那么相应的操作就不会发生。

13.1.1 与定时器_A 的相似和不同之处

除了下列情况外，定时器_B 与定时器_A 都相同：

- 定时器_B 的可被编程到 8，10，12，或 16 位。
- 定时器_B 的 TBCCRx 寄存器是双缓冲的，且可以被集合。
- 所有的定时器_B 都可以被放入一个高阻抗状态。
- 在定时器_B 上未执行 SCCI 位的功能。



NOTE: INCLK 是器件特定的，通常被分配给反相的 TBCLK，请参阅《特定器件数据手册》。

图 **13-1.** 定时器_B 的方框图

13.2 定时器_B 的操作

定时器_B 的模块由用用户软件进行配置。 定时器_B 的设置和操作将在下面的章节讨论。

13.2.1 16 位定时器计数器

16 位定时器/计数器寄存器，TBR，随着时钟信号的每个上升沿增/减（由操作模式决定）。 TBR 可以由软件进行读取或写入。 此外，定时器溢出时，它可以产生一个中断。

可通过设置 TBCLR 位来清除 TBR。 设置 TBCLR 也会清除时钟分频器和针对上数/下数的计数方向。

注: 修改定时器_B 寄存器

为了避免产生错误的操作状况，建议在修改定时器的操作（中断使能、中断标志，和 TBCLR 除外）前先停止定时器。

当定时器时钟和 CPU 时钟异步时，任何对 TBR 的读取会由于定时器未运行而导致所读的结果是不可预料的。 因此，当定时器运行时，需要多读几次，通过软件多数表决来确定正确的读数。 对 TBR 的写入操作是立即生效的。

13.2.1.1 TBR 的长度

定时器_B 可以通过 CNTLx 位将它配置为 8，10，12 或 16 位定时器。 最大的计数数值，$TBR_{(最大)}$，可选长度可分别为 0FFh，03FFh，0FFFh，和 0FFFFh。 在 8，10，和 12 位模式下，写入 TBR 寄存器中的数据是右对齐，带前导零。

13.2.1.2 时钟源选择和分频器

定时器的时钟源可以是 ACLK，SMCLK，或外部通过 TBCLK 和 INCLK（INCLK 是特定于器件的，通常被分配给反相 TBCLK，请参见《器件专用数据表》）。 时钟源由 TBSSELx 位来选择。 所选择的时钟源可以直接被传递给计时器或通过使用 IDx 位进行 2，4 或 8 分频。 当 TBCLR 被置位时，时钟分频器复位。

13.2.2 启动定时器

定时器可以通过以下方式启动或重新启动：

- 当 MCX>0 时，定时器计数到并且时钟源处于活动状态时。
- 当定时器模式为增/减模式时，定时器可以通过把 0 载入 TBCL0 来停止计数。 之后定时器可以通过把一个非 0 的数值载入 TBCL0 来重新开始计数。 在这种情况下，计时器开始以增加的方向从零递增。

13.2.3 定时器模式控制

定时器有 4 种操作模式，见表 13-1：停止、增、连续和增/减。 操作模式由 MCx 位来选择。

表 13-1. 定时器模式

MCx	模式	说明
00	停止	该定时器暂停。
01	向上	定时器循环地从 0 增到比较寄存器 TBCL0 的值。
10	连续	定时器循环地从 0 连续增加到由 CNTLx 位选择的值。
11	增/减	定时器循环地从 0 增到 TBCL0 的值再连续减至 0。

13.2.3.1 增模式

用于如果计数周期不同于 TBR (最大) 计数，则使用增模式。 定时器重复递增计数比较锁存器 TBCL0 的值，该值定义了周期，如在图 13-2中所示。 在此期间的定时器计数的值是 TBCL0+1。 当定时器的值等于 TBCL0 时，定时器就回到 0 重新开始计数。 当定时器的值大于 TBCL0 时，如果选择增模式，定时器立即从 0 重新开始计数。

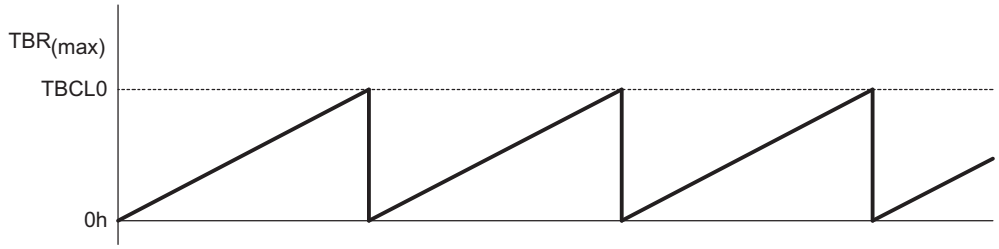


图 13-2. 增模式

当定时器*计数*到 TBCL0 值时，TBCCR0 CCIFG 中断标志被置位。 当定时器从 *TBCL0 至 0 计数*时，TBIFG 中断标志置位。图 13-3说明了标志置位循环。

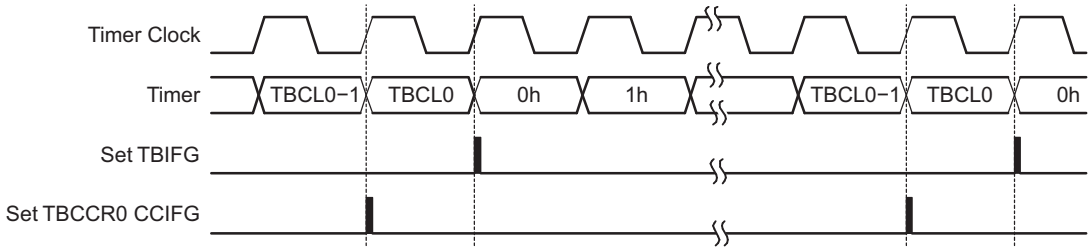


图 13-3. 增模式标志的置位

13.2.3.2 更改周期寄存器 TBCL0

当在定时器运行时修改 TBCL0，并且 TBCL0 加载事件为*立即*时，CLLD0=00，如果新的周期大于或等于旧的周期，或大于当前计数值，那么定时器就会计数至新的周期。 如果新周期小于当前的计数值，那么定时器回到 0。 但是，在计数器回到 0 之前会多出一个额外的计数。

13.2.3.3 连续模式

在连续模式中，定时器重复计数增到 TBR (最大) 然后重新从 0 开始增计数，如在图 13-4中所示。 比较锁存器 TBCL0 与其他捕获/比较寄存器的工作方式一样。

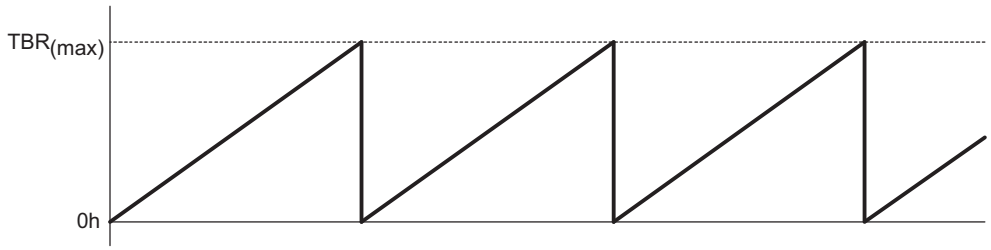


图 13-4. 连续模式

当定时器从 *TBR* (最大) 至 *0*计数时，TBIFG 中断标志被置位。图 13-5说明了标志置位循环。

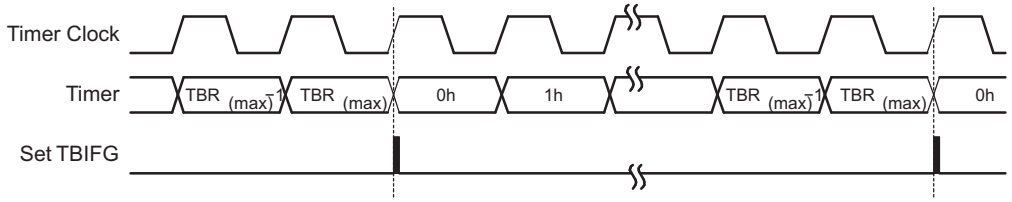


图 **13-5.** 连续模式标志的置位

13.2.3.4 连续模式的使用

连续模式可以用于产生独立的时间间隔和输出频率。 每次一个间隔完成时，就会产生一个中断。 下一个时间间隔被添加到中断服务子程序中的 TBCLx 锁存器。图 13-6显示了 2 个独立的时间间隔 t_0和 t_1正被添加至捕获/比较寄存器。 该时间间隔由硬件控制，而不是软件，不受中断延迟的影响。 多达 3 个（定时器_B3）或 7 个（定时器_B7）独立的时间间隔或输出频率可以通过使用捕获/比较寄存器生成。

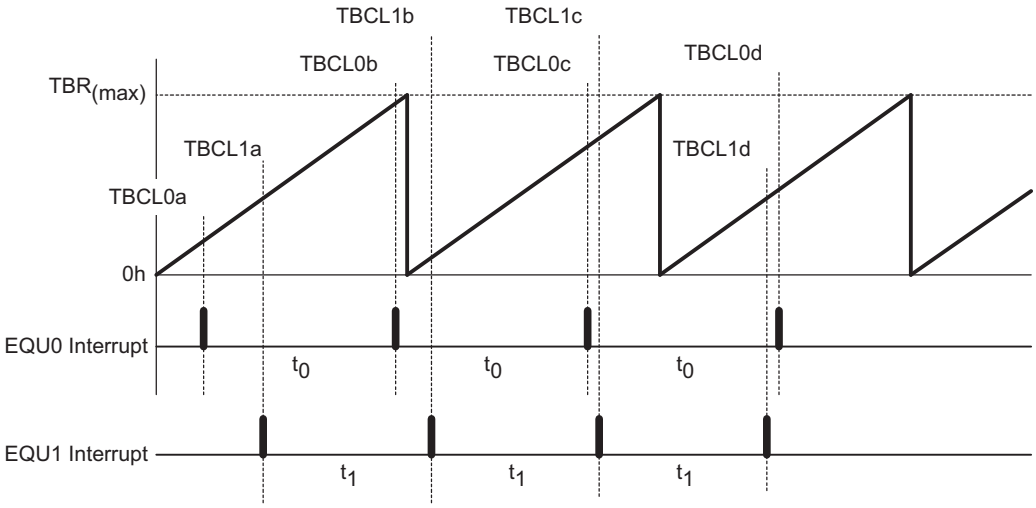


图 **13-6.** 连续模式时间间隔

时间间隔也可以由其他模式产生，在此 TBCL0 也可以被用作周期寄存器。 由于旧的 TBCLx 数据之和新周期的总和比 TBCL0 值大，所以操作就复杂得多。 当前一个 TBCLx 值加上 t_x比 TBCL0 数据大时，那么为了获得正确的时间间隔，必须减去 TBCL0+1。

13.2.3.5 增/减模式

如果定时器周期肯定不会与 TBR (最大) 计数相同， 且需要产生对称的脉冲时，才会使用增/减模式。 定时器重复递增计数比较锁存器 TBCL0 的值，之后再减至 0，如在图 13-7中所示。 该周期是 TBCL0 值的 2 倍。

注: **TBCL0 > TBR（最大）**

如果 TBCL0 > TBR (最大)，那么计数操作就和连续模式的配置一样。 不会从 TBR (最大) 减到 0。

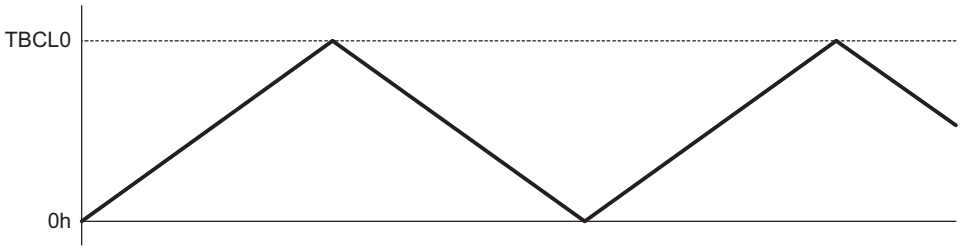


图 13-7. 增/减模式

计数方向是锁定的。 这就使得定时器停止后再能在停止前以相同的方向重新启动计数。 如果不想这样的话，就需要用 TBCLR 位来清零方向。 TBCLR 位也会清零 TBR 值和时钟分频。

在增减模式中，TBCCR0 CCIFG 中断标志和 TBIFG 中断标志在该周期中只被置位一次，它们相隔 1/2 个定时器周期。 当定时器从 *TBCL0−1* 至 *TBCL0* 计数时，TBCCR0 CCIFG 中断标志被置位；而定时器完成从 *0001h* 减至 *0000h* 的计数时，TBIFG 被置位。图 13-8说明了标志置位循环。

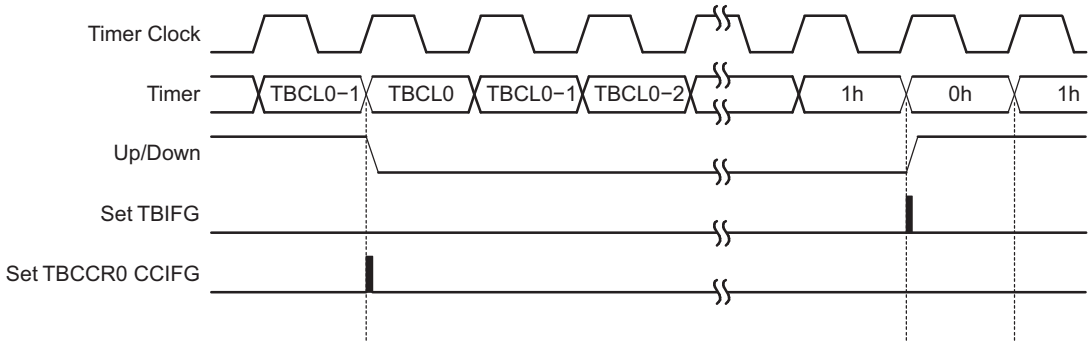


图 13-8. 增/模式标志的置位

13.2.3.6 改变周期寄存器 TBCL0 的值

如果在定时器正在计数，且以减的方向计数时改变 TBCL0，且当 TBCL0 加载事件为*立即的*时，定时器会继续下降直至减至 0。新的周期在减到0后开始。 TBCCR0 中的值被立即锁存到 TBCL0 中；然而，在计数器计数下降至 0 之后，新周期生效。

如果定时器正以增的方向计数，并且当新的周期已经锁存到 TBCL0 中时，且新的周期大于或等于旧的周期，或比当前计数值大，定时器会在减计数之前增计数到新的周期。 如果定时器正以增的方向计数，且当载入 TBCL0 时新周期小于当前计数值，定时器会开始减计数。 但是，在计数器开始减计数之前可能会产生一个额外的计数。

13.2.3.7 增/减模式的使用

增/减模式支持在输出信号之间需要死区时间的应用（请参阅《*定时器_B 输出单元*》小结部分）。 例如，为避免出现过载情况，2 个输出驱动一个 H 桥绝不能同时处于一个高状态。 在图 13-9该 $t_{空载}$给出的例子是;

$t_{空载}=t_{定时器}\times(TBCL1-TBCL3)$

其中，

$t_{空载}$=在此期间的两个输出需要处于非活动状态

$t_{定时器}$=定时器时钟周期时间

TBCLx=比较锁存器 x 的内容

可以同时加载集合的比较锁存器来保证死区时间。

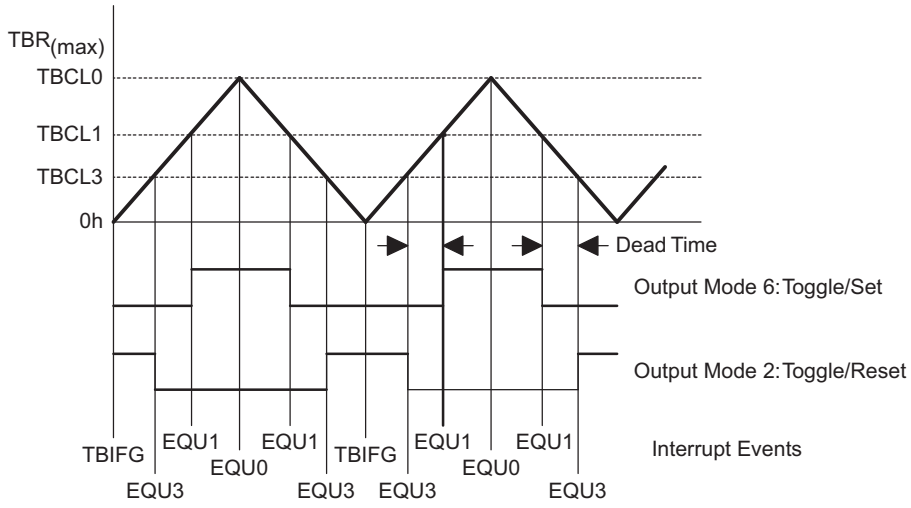


图 **13-9.** 增/减模式的输出单元

13.2.4 捕捉/比较块

在定时器_B 中有 3 个或 7 个相同的捕捉/比较块，TBCCRx。 这些块中的任一个都可用于捕获定时器数据或生成时间间隔。

13.2.4.1 捕获模式

当 CAP=1 时，就会选择捕获块。捕获块用于记录时间事件。 它可用于快速估计或时间测量。 捕获输入 CCIxA 和 CCIxB 被连接到一个外部引脚上或内部信号上，这通过 CCISx 位 选择。 CMx 位选择捕获输入信号的边沿如上升，下降，或两者兼而有之。 一个捕获发生在已选输入信号的边沿。 如果执行了一个捕获：

- 定时器的值就会被复制到 TBCCRx 寄存器。
- 中断标志 CCIFG 被置位。

在任何时刻，可以通过 CCI 位读取输入信号的电平。 MSP430x2xx 系列的器件可能会有被连接到 CCIxA 和 CCIxB 的不同信号。 有关这些信号连接的详细信息请参阅《特定器件数据手册》。

捕获信号可能会和定时器时钟异步，并导致一个竞争条件的发生。 置位 SCS 位可以在下个定时器时钟使捕获同步。 建议置位 SCS 位来使定时器时钟与捕获同步。 在图 13-10给出了有关这的图例说明。

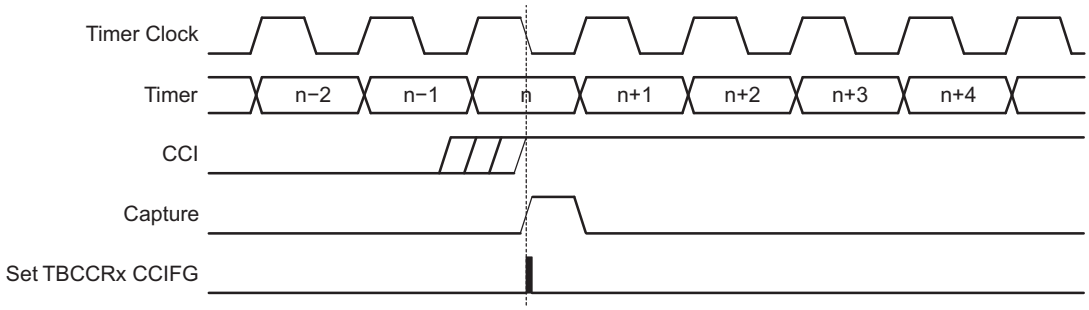


图 **13-10.** 捕获信号 **(SCS = 1)**

为了显示是否一个二次捕获在第一次捕获的值被读取之前发生，在每个捕获/比较寄存器中就会提供一个溢出逻辑。 如在图 13-11中所示，当这种情况发生时，位 COV 被置位。 COV 位必须由软件复位。

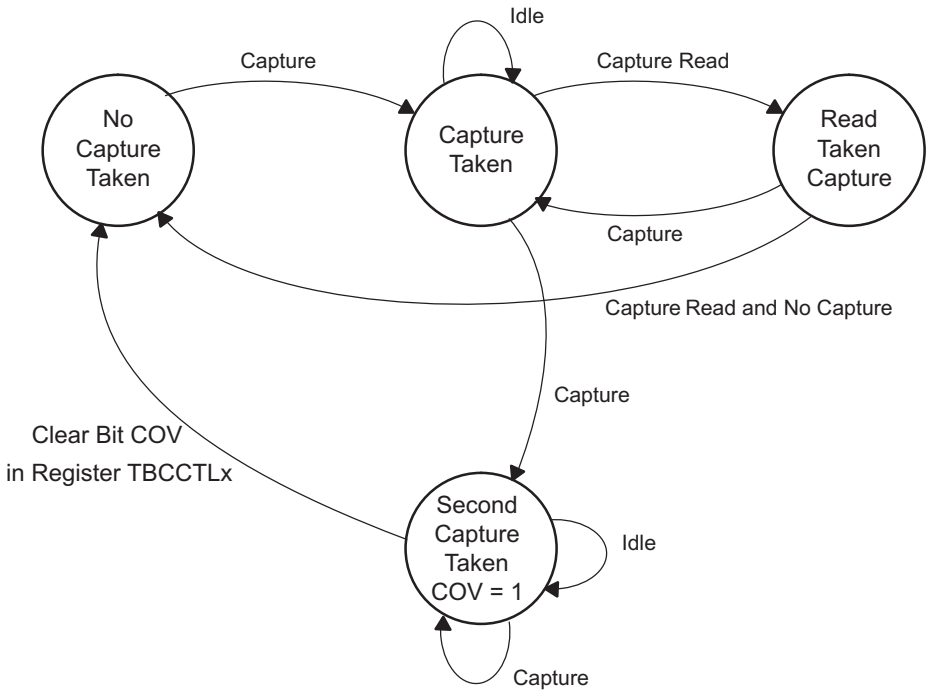


图 **13-11.** 捕获循环

13.2.4.1.1 通过软件初始化捕获

可通过软件初始化捕获。 CMx 位可以配置捕获的两个边沿。 之后为了在 V_{CC}和 GND 之间切换捕获信号，软件会置位位 CCIS1=1 且切换 CCIS0，每当 CCIS0 更改状态时就会初始化一个捕获：

```
MOV #CAP+SCS+CCIS1+CM_3,&TBCCTLx ; Setup TBCCTLxXOR #CCIS0, &TBCCTLx ; TBCCTLx = TBR
```

13.2.4.2 比较模式

当 CAP = 0 时，选择比较模式。比较模式用于生成 PWM 输出信号或在特定的时间间隔中断。 当 TBR*计数*到一个 TBCLx 值时：

- 中断标志 CCIFG 被置位。
- 内部信号 EQUx=1
- EQUx 根据输出模式影响输出

13.2.4.2.1 比较锁存器 TBCLx

比较锁存器 TBCLx，在比较模式中为定时器值的比较保持数据。 TBCLx 由 TBCCRx 缓冲。 当一个比较周期更新时，已缓冲的比较锁存器会为用户提供控制权。 用户不能直接访问 TBCLx。 比较数据被写入每个 TBCCRx 后自动传递到 TBCLx 中。 从 TBCCRx 到 TBCLx 传递的时间是用户可通过 CLLDx 位选择的，如在表 13-2中所述。

表 13-2. TBCLx 加载事件

CLLDx	说明
00	当 TBCCRx 被写入时，新数据将被立即从 TBCCRx 转移到 TBCLx 中。
01	当 TRB*计数*至 0 时，新数据将被从 TBCCRx 转移到 TBCLx 中。
10	当在增和连续模式中 TRB*计数*至 0 时，新数据将被从 TBCCRx 转移到 TBCLx 中。 当在增/减模式中 TRB*计数*至 0 时，新数据将被从 TBCCRx 转移到 TBCLx 中。
11	当 TRB*计数*至旧的 TBCLx 值时，新数据将被从 TBCCRx 转移到 TBCLx 中。

13.2.4.2.2 编组比较锁存器

多个比较锁存寄器可以通过 TBCLGRPx 位编组，以便于同步更新。 当使用编组时，组中序号最小的 TBCCRx 的 CLLDx 位决定该组的每个比较锁存器的加载事件，把 TBCLGRP=3 时除外，如在表 13-3中所示。 不得把 CLLDx 控制的 TBCCRx 位设置为零。 当把 CLLDx 控制的 TBCCRx 位设置为零时，所有比较锁存器就会在他们对应的 TBCCRx 被写入时立即更新；不存在比较锁存器编组。

当编组时，则需要加载的比较锁存器必须存在 2 个条件。 第一，即使是 TBCCRx 数据=原来的 TBCCRx 数据时，该组的所有 TBCCRx 寄存器必须也都更新。 第二，加载事件必须发生。

表 13-3. 比较锁存器的操作模式

TBCLGRPx	编组	已更新的控制
00	无	个人
01	TBCL1+TBCL2 TBCL3+TBCL4 TBCL5+TBCL6	TBCCR1 TBCCR3 TBCCR5
10	TBCL1+TBCL2+TBCL3 TBCL4+TBCL5+TBCL6	TBCCR1 TBCCR4
11	TBCL0+TBCL1+TBCL2+TBCL3+TBCL4+TBCL5+TBCL6	TBCCR1

13.2.5 输出单元

每个捕获/比较块包含一个输出单元。 输出单元用于产生如 PWM 这样的信号。 每个输出单元有 8 种可以根据 EQU0 和 EQUx 信号产生信号的操作模式。 TBOUTH 的引脚功能可以用于将所有的定时器_B 输出拉近一个高阻抗状态。 当为该引脚选择 TBOUTH 引脚功能时，且当该引脚被上拉时，所有的定时器_B 输出会处于一个高阻抗状态。

13.2.5.1 输出模式

输出模式由 OUTMODx 位来确定，如表 13-4在中所述。 对于除了模式 0 以外的所有模式来说，OUTx 信号随着定时器时钟的上升沿而改变。输出模式 2，3，6，和 7 对输出单元 0 来说是无用的，因为 EQUx = EQU0。

表 **13-4.** 输出模式

OUTMODx	模式	说明
000	输出	输出信号 OUTx 由 OUTx 位定义。 当 OUTx 位更新时，OUTx 信号会立刻更新。
001	置位	当定时器*计数*到 TBCLx 值时，输出被置位。 它保持置位直到一个定时器复位，或直到选择另一个输出模式并影响该输出。
010	切换/复位	当定时器*计数*到 TBCLx 值时，输出被切换。 当定时器*计数*至 TBCL0 值时，它被复位。
011	置位/复位	当定时器*计数*到 TBCLx 值时，输出被置位。 当定时器*计数*至 TBCL0 值时，它被复位。
100	切换	当定时器*计数*到 TBCLx 值时，输出被切换。 输出周期为双定时器周期。
101	复位	当定时器*计数*到 TBCLx 值时，输出被复位。 直到另一个输出模式被选择时且影响输出时，它才不保持复位状态。
110	切换/置位	当定时器*计数*到 TBCLx 值时，输出被切换。 当定时器*计数*至 TBCL0 值时，它被置位。
111	复位/置位	当定时器*计数*到 TBCLx 值时，输出被复位。 当定时器*计数*至 TBCL0 值时，它被置位。

13.2.5.1.1 输出示例，增模式中的定时器

当定时器计数增至 TBCLx 值，且从 TBCL0 降到 0 时，OUTx 信号根据输出模式而改变。 在图 13-12中给出了使用 TBCL0 和 TBCL1 的一个例子。

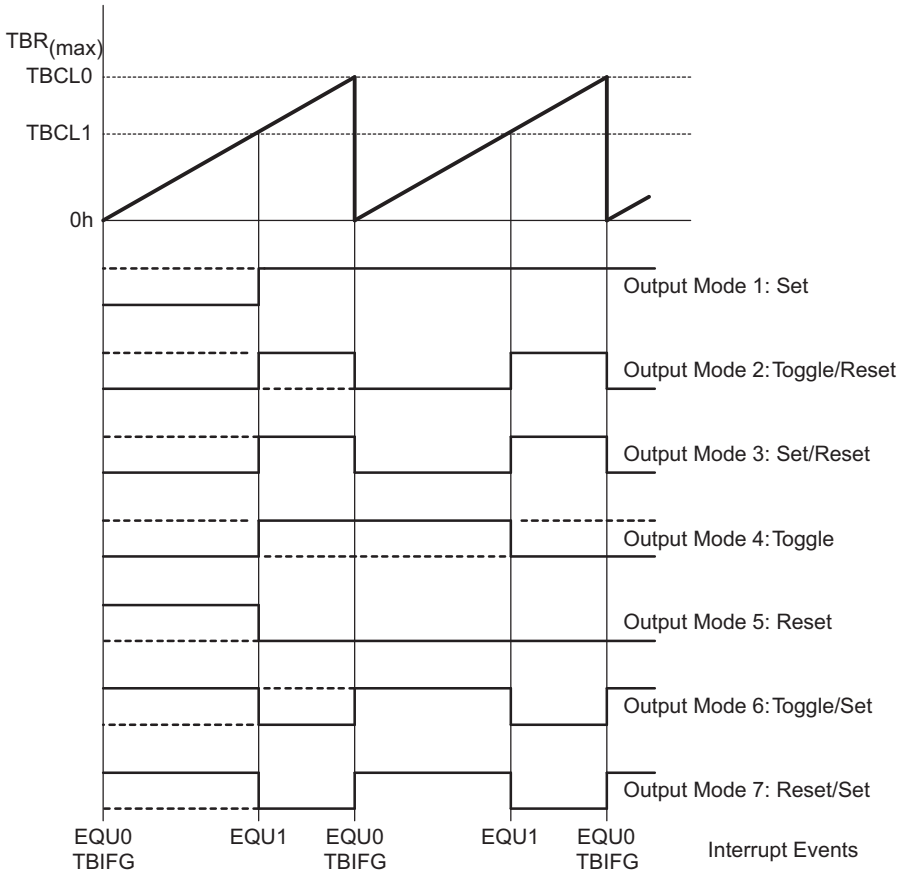


图 **13-12.** 输出示例，定时器处于增模式

13.2.5.1.2 输出示例，定时器处于连续模式

当定时器达到 TBCLx 和 TBCL0 值时，OUTx 信号根据输出模式而改变，在图 13-13中给出了使用 TBCL0 和 TBCL1 的一个例子。

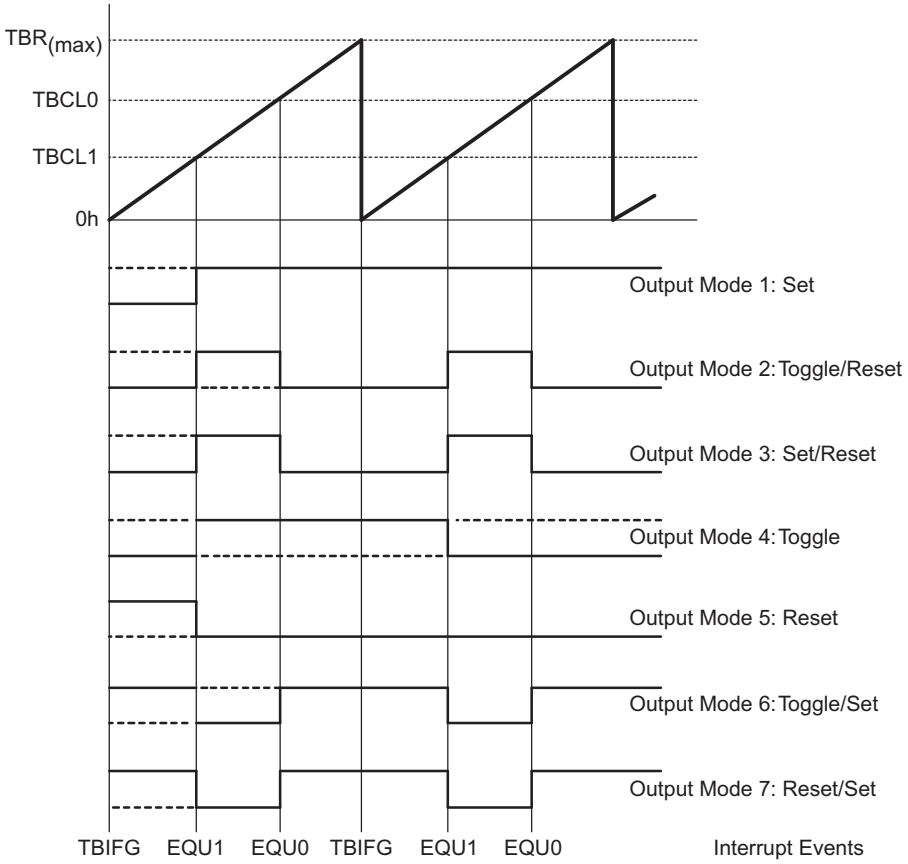


图 **13-13.** 输出示例，定时器处于连续模式

13.2.5.1.3 输出示例，定时器处于增/减模式

当定时器在任一计数方向上等于 TBCLx 和定时器等于 TBCL0 时，OUTx 信号会根据输出模式而改变。 在图 13-14中给出了使用 TBCL0 和 TBCL1 的一个例子。

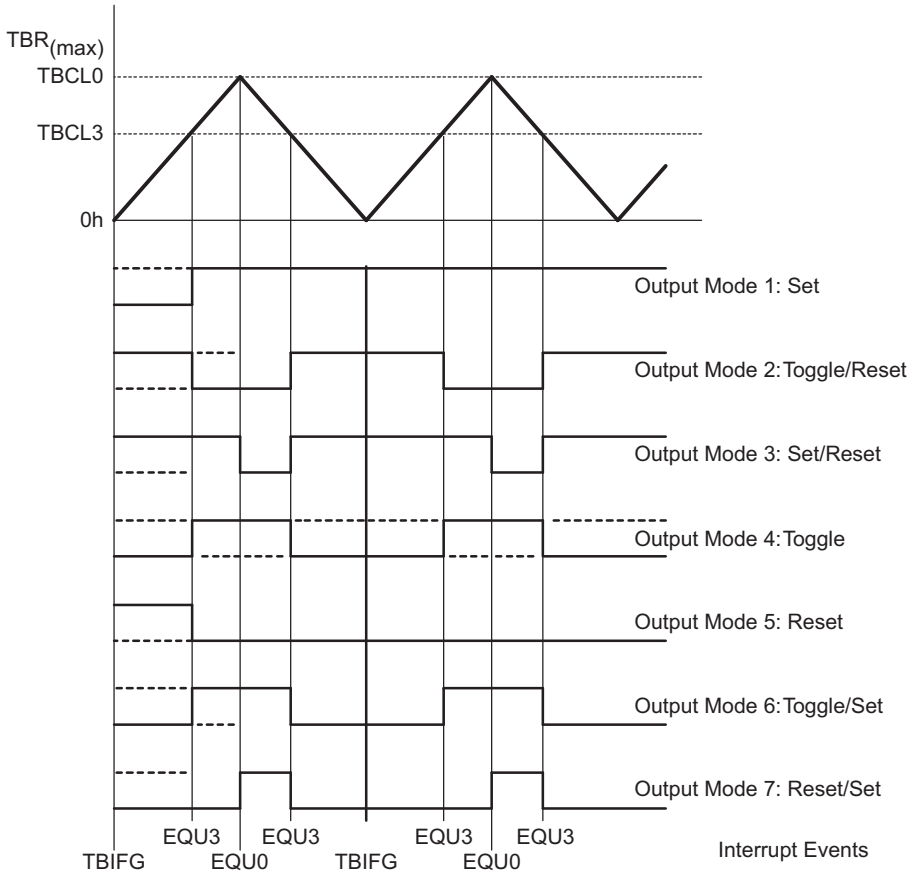


图 **13-14.** 输出示例，定时器处于增/减模式

注: 在输出模式间切换

当在输出模式之间切换时，在过渡期间 OUTMODx 位的其中一个应保持置位，除非切换至模式 0。否则会由于一个非门解码输出模式 0 而导致出现脉冲干扰。在输出模式之间安全切换的一个方法是用输出模式 7 作为一个过度状态：

```
BIS #OUTMOD_7,&TBCCTLx ; Set output mode=7BIC #OUTMODx, &TBCCTLx ; Clear
unwanted bits
```

13.2.6 定时器_B 的中断

2 个中断向量与 16 位定时器_B 相关联：

- TBCCR0 CCIFG 的 TBCCR0 中断向量
- 所有其他 CCIFG 标志和 TBIFG 的 TBIV 中断向量

在捕获模式下，当在相应的 TBCCRx 寄存器中捕获到一个定时器的值时， 任何 CCIFG 标志都被置位。 在比较模式下，当 TBR*计数*至相关的 TBCLx 值时，任何 CCIFG 标志都被置位。 软件也可以置位或清零任何 CCIFG 标志。 当它们相应的 CCIE 位和 GIE 位被置位时，所有 CCIFG 标志就会要求产生一个中断。

13.2.6.1 TBCCR0 的中断向量

TBCCR0 CCIFG 标志拥有定时器_B 的最高中断优先级，并有一个专用的中断向量，如在图 13-15中所示。当 TBCCR0 的中断要求被服务之后，TBCCR0 CCIFG 标志会自动复位。

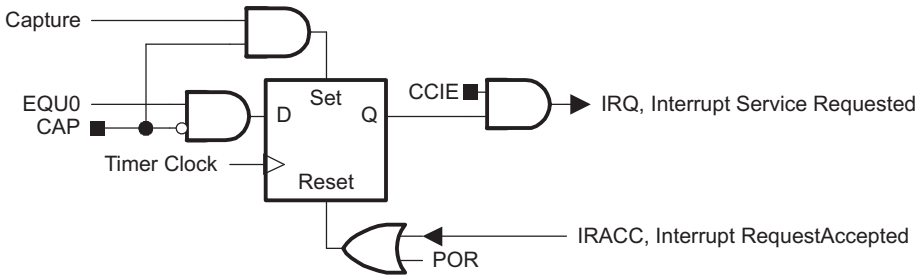


图 13-15. 捕捉/比较 TBCCR0 中断标志

13.2.6.2 TBIV，中断向量发生器

TBIFG 标志 和 TBCCRx CCIFG 标志（不包括 TBCCR0 CCIFG）被优先化且被组合在一起共用一个中断向量。 中断向量寄存器 TBIV 用于确定哪个标志要求了一个中断。

使能的最高优先级的中断（不包括 TBCCR0 CCIFG）在 TBIV 寄存器中产生了一个数字（见寄存器描述）。 为了自动进入相应的软件程序，可以对这个数字进行评估或将其添加到程序计数器中，。 禁止定时器_B 中断不会影响 TBIV 的值。

任何对 TBIV 寄存器的访问、读取或写入都会自动复位最高优先级的挂起中断标志。 如果另一个中断标志被置位，在服务完最初的中断后会立即产生另一个中断。 例如，当中断服务子程序访问 TBIV 寄存器时，如果 TBCCR1 和 TBCCR2 CCIFG 标志被置位，则 TBCCR1 CCIFG 会被自动复位。 在中断服务子程序的 RETI 命令被执行后，TBCCR2 CCIFG 标志会产生另一个中断。

13.2.6.3 TBIV，中断处理程序示例

以下软件示例给出了 TBIV 和处理开销的使用和操作 TBIV 的值被加入 PC 以便自动跳转到相应的子程序。

右边空白处的数字表明了每条指令所需的 CPU 时钟周期。 不同中断源的软件开销包含中断延迟时间和返回中断周期，但并不包含任务本身的执行时间。 延迟时间为：

- 捕获/比较块 CCR0： 11 个周期
- 捕获/比较模块 CCR1 至 CCR6： 16 个周期
- 定时器溢出 TBIFG：14 个周期

Example 13-1给出了建议的定时器_B3 的 TBIV 的使用。

Example 13-1. *建议的* ***TBIV*** *的使用*

```
; Interrupt handler for TBCCR0 CCIFG. CyclesCCIFG_0_HND... ; Start of handler Interrupt latency 6RETI
5; Interrupt handler for TBIFG, TBCCR1 and TBCCR2 CCIFG.TB_HND ... ; Interrupt latency 6ADD &TBIV,PC
; Add offset to Jump table 3RETI ; Vector 0: No interrupt 5JMP CCIFG_1_HND ; Vector 2: Module 1 2JMP
CCIFG_2_HND ; Vector 4: Module 2 2RETI ; Vector 6RETI ; Vector 8RETI ; Vector 10RETI ; Vector
12TBIFG_HND ; Vector 14: TIMOV Flag... ; Task starts hereRETI 5CCIFG_2_HND ; Vector 4: Module 2... ;
Task starts hereRETI ; Back to main program 5; The Module 1 handler shows a way to look if any other;
interrupt is pending: 5 cycles have to be spent, but ; 9 cycles may be saved if another interrupt is
pendingCCIFG_1_HND ; Vector 6: Module 3 ... ; Task starts hereJMP TB_HND ; Look for pending ints 2
```

13.3 定时器_B 的寄存器

在表 13-5列出了定时器_B 的寄存器。

表 13-5. 定时器_B 的寄存器

寄存器	简氏	寄存器类型	地址	初始化状态
定时器_B 控制	TBCTL	读取/写入	0180h	用 POR 复位
定时器_B 计数器	TBR	读取/写入	0190h	用 POR 复位
定时器_B 的捕获/比较控制 0	TBCCTL0	读取/写入	0182h	用 POR 复位
定时器_B 的捕获/比较 0	TBCCR0	读取/写入	0192h	用 POR 复位
定时器_B 的捕获/比较控制 1	TBCCTL1	读取/写入	0184h	用 POR 复位
定时器_B 的捕获/比较 1	TBCCR1	读取/写入	0194h	用 POR 复位
定时器_B 的捕获/比较控制 2	TBCCTL2	读取/写入	0186h	用 POR 复位
定时器_B 的捕获/比较 2	TBCCR2	读取/写入	0196h	用 POR 复位
定时器_B 的捕获/比较控制 3	TBCCTL3	读取/写入	0188h	用 POR 复位
定时器_B 的捕获/比较 3	TBCCR3	读取/写入	0198h	用 POR 复位
定时器_B 的捕获/比较控制 4	TBCCTL4	读取/写入	018Ah	用 POR 复位
定时器_B 的捕获/比较 4	TBCCR4	读取/写入	019Ah	用 POR 复位
定时器_B 的捕获/比较控制 5	TBCCTL5	读取/写入	018Ch	用 POR 复位
定时器_B 的捕获/比较 5	TBCCR5	读取/写入	019Ch	用 POR 复位
定时器_B 的捕获/比较控制 6	TBCCTL6	读取/写入	018Eh	用 POR 复位
定时器_B 的捕获/比较 6	TBCCR6	读取/写入	019Eh	用 POR 复位
定时器_B 中断矢量	TBIV	只读	011Eh	用 POR 复位

13.3.1 定时器_B 的控制寄存器 TBCTL

15	14	13	12	11	10	9	8
未使用	TBCLGRPx		CNTLx		未使用	TBSSELx	
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
IDx		MCx		未使用	TBCLR	TBIE	TBIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	w-(0)	rw-(0)	rw-(0)

未使用 位 15 未使用

TBCLGRP 位 14-13 TBCLx 编组

- 00 每个 TBCLx 锁存器都独立加载
- 01 TBCL1 + TBCL2（TBCCR1 CLLDx 位控制更新）
TBCL3 + TBCL4（TBCCR3 CLLDx 位控制更新）
TBCL5 + TBCL6（TBCCR5 CLLDx 位控制更新）
TBCL0 无关
- 10 TBCL1 + TBCL2 + TBCL3（TBCCR1 CLLDx 位控制更新）
TBCL4 + TBCL5 + TBCL6（TBCCR4 CLLDx 位控制更新）
TBCL0 无关
- 11 TBCL0 + TBCL1 + TBCL2 + TBCL3 + TBCL4 + TBCL5 + TBCL6（TBCCR1 CLLDx 位控制更新）

CNTLx 位 12-11 计数器的长度

- 00 16 位，TBR（最大）=0FFFFh
- 01 12 位，TBR（最大）=0FFFh
- 10 10 位，TBR（最大）=03FFh
- 11 8 位，TBR（最大）=0FFh

未使用 位 10 未使用

TBSSELx 位 9-8 定时器_B 的时钟源选择。

- 00 TBCLK
- 01 ACLK
- 10 SMCLK
- 11 INCLK（INCLK 是器件专用的，通常被分配给反相 TBCLK）（请参阅《器件专用数据表》）

IDx 位 7-6 输入分频。 这些位为输入时钟选择分频。00 /101 /210 /411 /8

MCx 位 5-4 模式控制。 当不使用定时器_B 时，设置 MCX=00H 能节省功耗。

- 00 停止模式：定时器被暂停
- 01 上数模式：定时器计数增至 TBCL0
- 10 连续模式：定时器计数到由 CNTLx 设定的值
- 11 上数/下数模式：定时器计数到 TBCL0 并下降至 0000h

未使用 位 3 未使用

TBCLR 位 2 定时器_B 清零。 设置该位会复位 TBR，时钟分频器，和计数方向。 TBCLR 位自动复位且始终读为 0。

TBIE 位 1 定时器_B 中断使能。 该位启用 TBIFG 的中断请求。

- 0 中断被禁止
- 1 中断被启用

TBIFG 位 0 定时器_B 中断标志。

- 0 无中断等待
- 1 中断等待

13.3.2 TBR，定时器_B 的寄存器

15	14	13	12	11	10	9	8
TBRx							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
TBRx							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

TBRx	位 15-0	定时器_B 寄存器. TBR 寄存器是定时器_B 的计数。

13.3.3 TBCCRx，定时器_B 的捕捉/比较寄存器 x

15	14	13	12	11	10	9	8
TBCCRx							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
TBCCRx							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

TBCCRx	位 15-0	定时器_B 的捕获/比较寄存器。 比较模式：比较数据被写入每个 TBCCRx 且被自动传递到 TBCLx 中。 TBCLx 保存了与定时器_B 的寄存器，TBR，定时器的值相比较的数据。 捕捉模式：定时器_B 的寄存器，TBR，当一个捕获被执行时，TBR 就被复制到 TBCCRx 寄存器中。

13.3.4 TBCCTLx，捕获/比较控制寄存器

15	14	13	12	11	10	9	8
CMx		**CCISx**		**SCS**	**CLLDx**		**CAP**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r-(0)	rw-(0)
7	6	5	4	3	2	1	0
OUTMODx			**CCIE**	**CCI**	**OUT**	**COV**	**CCIFG**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	rw-(0)	rw-(0)	rw-(0)

CMx 位 15-14 捕捉模式
- 00 无捕捉
- 01 在上升沿的捕捉
- 10 在下降沿的捕捉
- 11 在上升和下降沿都捕捉

CCISx 位 13-12 捕捉/比较输入选择。 这些位选择 TBCCRx 的输入信号。 有关特定信号的连接请参阅《器件专用数据表》。
- 00 CCIxA
- 01 CCIxB
- 10 GND
- 11 V_{CC}

SCS 位 11 同步捕捉源。 此位是用来同步定时器时钟与捕获输入信号的。
- 0 异步捕捉
- 1 同步捕捉

CLLDx 位 10-9 比较锁存载入。 该位选择比较锁存载入事件。
- 00 TBCCRx 写入时 TBCLx 载入
- 01 当 TBR*计数*至 0 时加载 TBCLx。
- 10 当 TBR*计数*至 0 时加载 TBCLx（上数或连续模式）
当 TBR*计数*至 TBCL0 或至 0 时加载 TBCLx（上数/下数模式）
- 11 当 TBR*计数*至 TBCLx 时加载 TBCLx

CAP 位 8 捕获模式
- 0 比较模式
- 1 捕捉模式

OUTMODx 位 7-5 输出模式。 模式 2，3，6，和 7 对 TBCL0 来说是无用的，这是因为 EQUx = EQU0。
- 000 输出位的值
- 001 置位
- 010 切换/复位
- 011 置位/复位
- 100 切换
- 101 复位
- 110 切换/置位
- 111 复位/置位

CCIE 位 4 捕捉/比较中断使能。 该位使能相应 CCIFG 标志的中断请求。
- 0 中断禁止
- 1 中断使能

CCI 位 3 捕捉/比较输入。 所选输入信号可以由该位读出。

OUT 位 2 输出。 对于输出模式 0，该位直接控制输出的状态。
- 0 输出低电平
- 1 输出高电平

COV 位 1 捕捉溢出。 该位表示一个已发生的捕捉溢出。 COV 位必须由软件复位。
- 0 没有捕捉溢出发生
- 1 捕捉溢出发生

CCIFG 位 0 捕捉/比较中断标志
- 0 无中断等待
- 1 中断等待

13.3.5 TBIV，定时器_B 的中断向量寄存器

15	14	13	12	11	10	9	8
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**TBIVx**			**0**
r0	r0	r0	r0	r-(0)	r-(0)	r-(0)	r0

TBIVx 位 15-0 定时器_B 中断矢量值

TBIV 的目录	中断源	中断标志	中断优先级
00h	无中断等待	-	
02h	捕捉/比较 1	TBCCR1 CCIFG	最高
04h	捕捉/比较 2	TBCCR2 CCIFG	
06h	捕捉/比较 3(1)	TBCCR3 CCIFG	
08h	捕捉/比较 4(1)	TBCCR4 CCIFG	
0Ah	捕捉/比较 5(1)	TBCCR5 CCIFG	
0Ch	捕捉/比较 6(1)	TBCCR6 CCIFG	
0Eh	定时器溢出	TBIFG	最低

(1) 不适用于所有设备

通用串行接口 (USI)

通用串行接口 (USI) 模块提供与一个硬件模块的 SPI 和I²C 串行通信。 本章讨论两个模式。

Topic **Page**

14.1 USI 介绍

USI 模块提供支持同步串行通信的基本功能性。 在其最简单的形式中，它是一个可被用于输出数据帧的 8 位或 16 位移位寄存器，或者当与最少软件组合使用时，它可执行串行通信。 此外，USI 包括简化 SPI 和 I^2C 通信的内置硬件功能性。 USI 模块还包括中断来进一步减少针对串行通信和保持 MSP430 超低功耗功能所需软件开销。

USI 模块特性包括：

- 三线制 SPI 模式支持
- I^2C 模式支持
- 可变数据长度
- LPM4 中的从器件运行；无需内部时钟
- 可选 MSB 或 LSB 数据顺序
- 针对带有自动 SCL 控制的 I^2C 模式的 START 和 STOP 检测
- 主控模式下的仲裁丢失检测
- 可编程时钟生成
- 可选时钟极性和相位控制

图 14-1显示了 SPI 模式中的 USI 模块。图 14-2显示了 I^2C 模式中的 USI 模块。

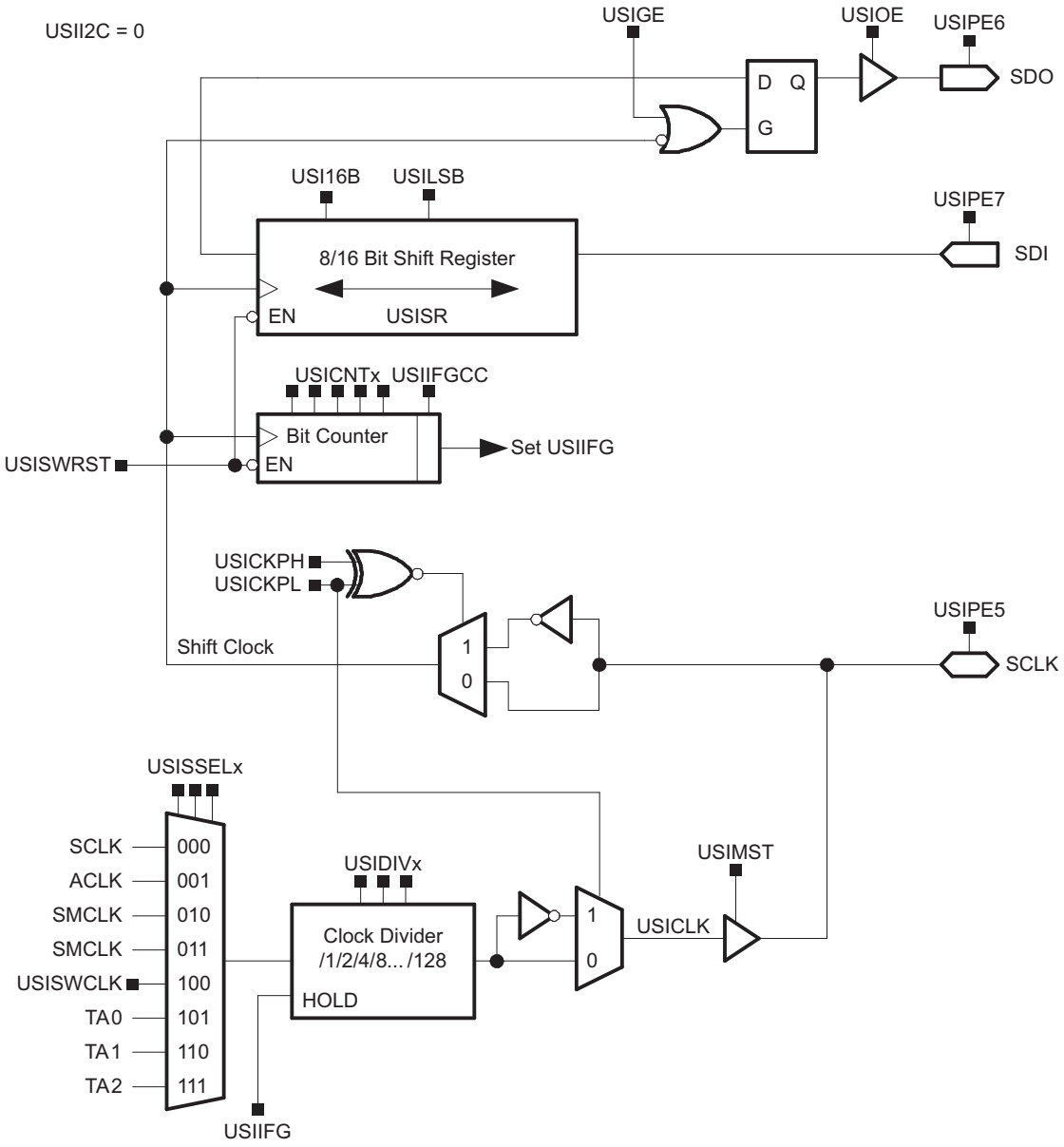


图 **14-1. USI** 方框图：**SPI** 模式

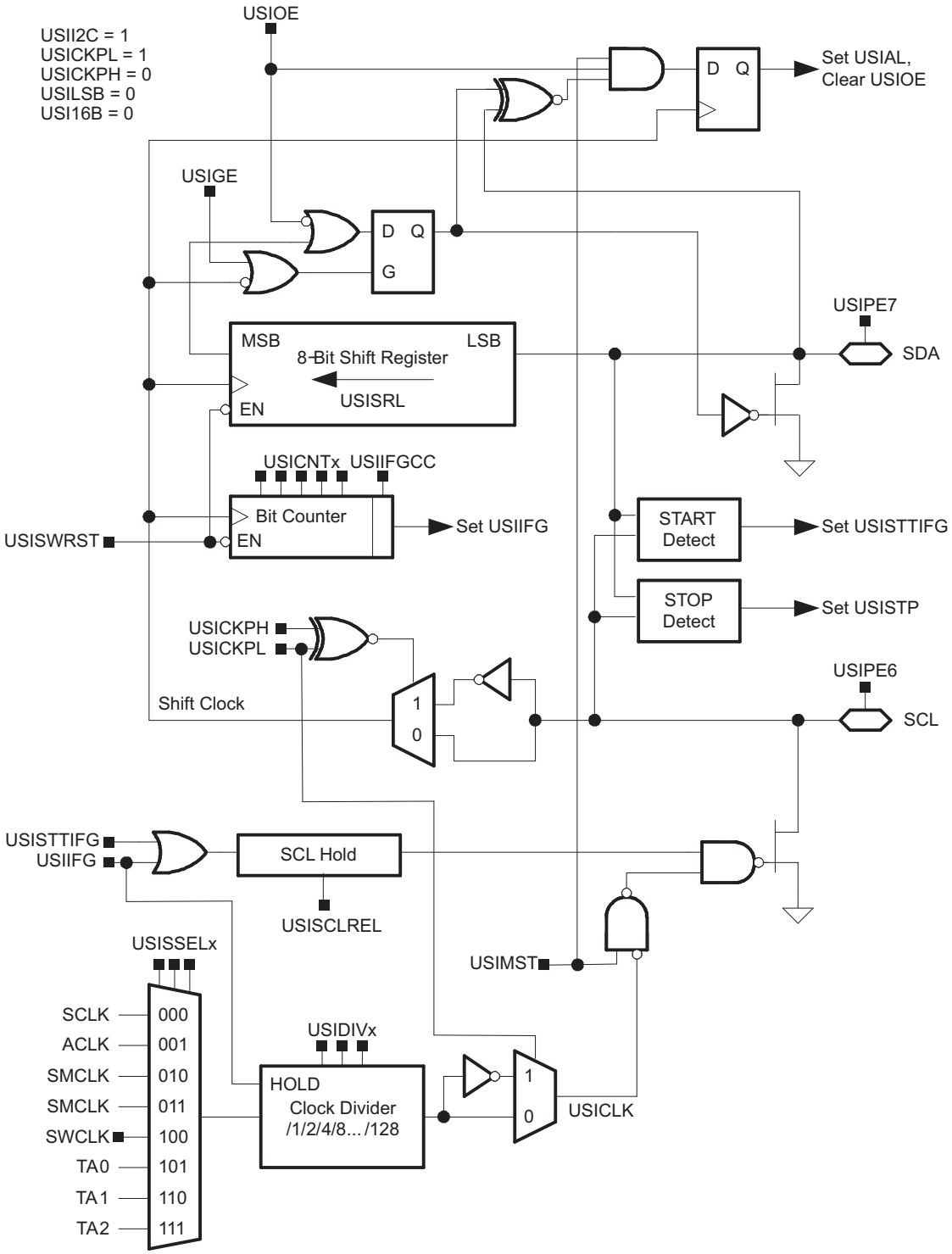


图 **14-2. USI** 方框图：**I²C** 模式

14.2 USI 运行

USI 模块是一个移位寄存器和包含支持 SPI 和 I²C 通信逻辑的位计数器。 USI 移位寄存器 (USISR) 可由软件直接访问并且包含将被传送或者已经被接收到的数据。

位计数器计算被采样位的数量并且当 USICNTx 值变为零时设定 USI 中断标志 USIIFG，通过递减或者直接将零写入 USICNTx 位。 当 USIIFG=0 时，将一个大于 0 值写入 USICNTx 将自动清除 USIIFG，否则 USIIFG 不受影响。 当它们变为 0 时，USICNTx 位停止递减。它们不会下溢至 0FFh。

计数器和移位寄存器由同一个移位时钟驱动。 在一个上升的移位时钟边沿，USICNTx 递减并且 USISR 采样下一个位输入。 被连接到移位寄存器的输出的锁存将输出的改变延迟至移位时钟的下降沿。 通过设置 USIGE 位，它可被透明完成。 根据 USILSB 位，这个设置将立即将 USISR 的 MSB 或者 LSB 输出到 SDO 引脚。

14.2.1 USI 初始化

当 USI 软件复位位，USISWRST 被置位，标志 USIIFG，USISTTIFG，USISTP 和 USIAL 将被保持在复位状态。 不对 USISR 和 USICNTx 计时，并且它们的内容不受影响。 在 I²C 模式中，通过 USI 硬件，SCL 线路也被释放至闲置状态。

为了激活 USI 端口功能性，在 USI 控制寄存器中的相应 USIPEx 位被置位。 这将为引脚选择 USI 功能并且也为引脚保持 PxIN 和 PxIFG 功能。 借助于这个特性，由软件通过 PxIN 寄存器读取端口输入电平并且进入的数据流能够生成数据传输上的端口中断。 例如，这将有助于在一个 START 边沿上生成一个端口中断。

14.2.2 USI 时钟生成

USI 时钟生成器包含一个时钟选择复用器、一个分频器、和选择图 14-1和图 14-2中显示的方框图内时钟极性的功能。

可从内部时钟 ACLK 或者 SMCLK，从一个外部时钟 SCLK，以及从 Timer_A 的捕捉/比较输出内选择时钟源。 此外，当 USISSELx=100 时，可使用 USISWCLK 位通过软件来为模块计时。

USIDIVx 位可被用于通过一个 2 至 128 的倍数来将所选的时钟的分频。 当 USIIFG=0 或者 当模块运行在受控模式下的时候，生成的时钟，USICLK 被停止。

USICKPL 位被用于选择 USICLK 的极性。 当 USICKPL=0 时，USICLK 的无效电平为低电平。 当 USICKPL=1 时，USICLK 的无效电平为高电平。

14.2.3 SPI 模式

当 USII2C=0 时，USI 模块被配置为 SPI 模式。控制位 USICKPL 选择 SPI 时钟的无效电平，而 USICKPH 选择 SDO 被更新和 SDI 被采样上的时钟边沿。图 14-3显示了针对一个 8 位，MSB 首先传送的时钟/数据关系。 USIPE5，USIPE6 和 USIPE7 被置位来启用 SCLK，SDO 和 SDI 端口功能。

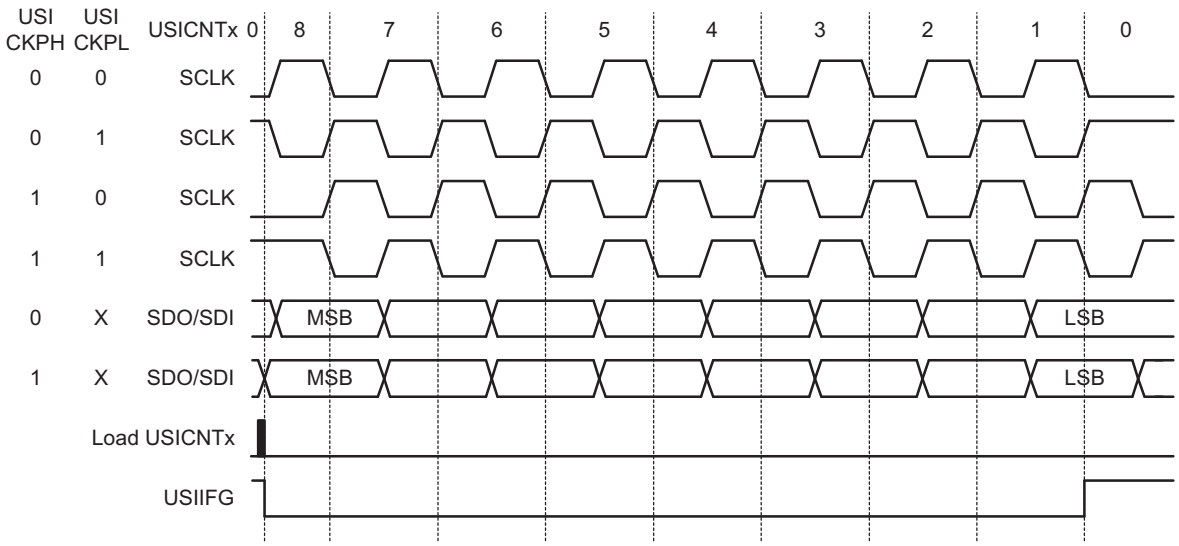


图 **14-3. SPI** 时序

14.2.3.1 SPI 主控模块

通过设置主控位 USIMST 和清除 I^2C 位 USII2C，USI 模块被配置为 SPI 主控。 由于主控为受控提供时钟，需要选择一个适当的时钟源并且 SCLK 被配置为输出。 当 USIPE5=1 时，SCLK 被配置为一个输出。

当 USIIFG=0 和 USICNTx>0 时，时钟生成被启用并且主控将使用 USISR 开始计时输入/输出数据。

在新数据被写入移位寄存器用于传输时，必须将接收到的数据从移位寄存器中读出。 在一个典型应用中，USI 软件将从 USISR 中读取接收到的数据，将被发送的数据写入 USISR，并且通过将被发送的一定数量的位写入 USICNTx 来启用用于下次传输的模块。

14.2.3.2 SPI 受控模式

通过清除 USIMST 和 USII2C 位，USI 模块被配置为 SPI 受控模式。 在这个模式下，当 USIPE5=1 时，SCLK 被自动配置为一个输入并且 USI 从外部接收来自主器件的时钟。

如果 USI 将传输数据，在主器件提供第一个时钟边沿之前，必须将数据载入到移位寄存器。 通过设置 USIOE，输出被启用。 当 USICKPH=1 时，载入移位寄存器后，MSB 将立即在 SDO 上可见。

通过清除 USIOE 位，SDO 位可被禁用。 如果从器件不在总线上有多个从器件的环境中寻址，这一功能将有所帮助。

一旦所有位被接收，在来自主控的下一个时钟边沿之前，数据必须从 USISR 中读取并且新数据被载入到 USISR 中。 在一个典型应用中，在接收数据后，USI 软件将读取 USISR 寄存器，将被传送的新数据写入 USISR，并且通过将一定数量的位传送至 USICNTx 来为下一次传输来启用 USI 模块。

14.2.3.3 USISR 操作

16 位 USISR 由两个 8 位寄存器，USISRL 和 USISR 组成。 控制位 USI16B 选择用于数据传输和接收的 USISR 的位的数量。 当 USI16B=0 时，只使用较低的 8 位，USISRL。

为了传输小于 8 位的数据，数据必须被载入到 USISRL，这样未使用的位没有被移除。 根据 USILSB，数据必须 MSB 或者 LSB 对齐。图 14-4显示了一个 7 位数据处理的示例。

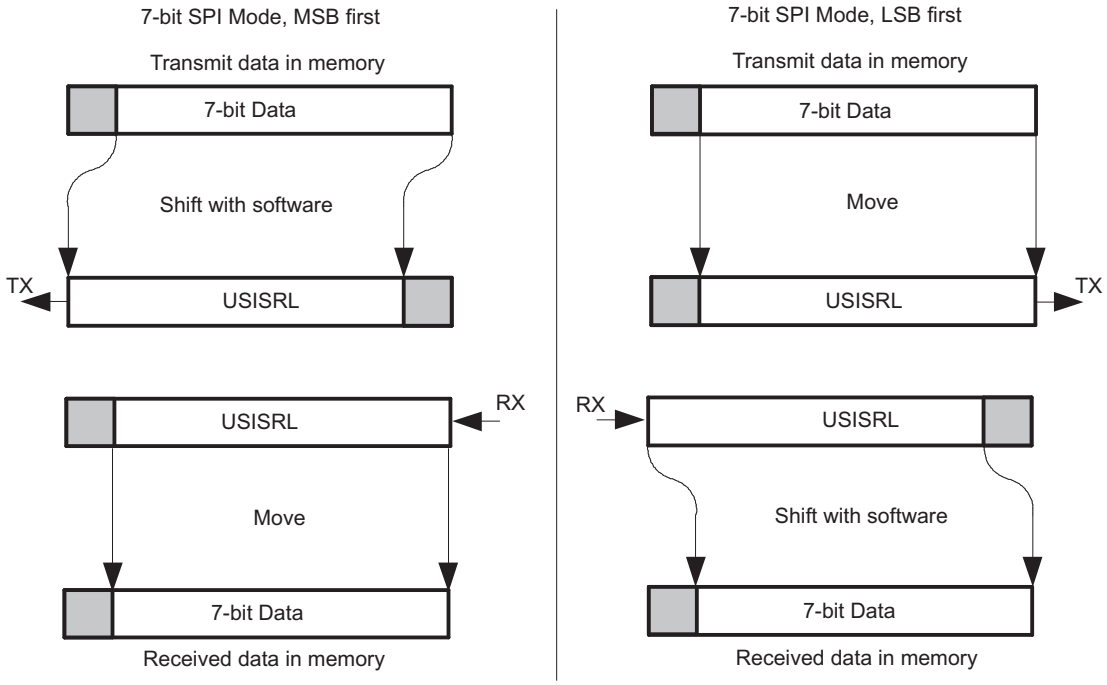


图 14-4. 针对 7 位 SPI 数据的数据调整

当 USI16B=1 时，所有 16 位被用于数据处理。 当使用 USISR 来访问 USISRL 和 USISRH 时，当小于 16 位的数据的使用方式与图 14-4中显示的方式一样时，数据需要被适当调整。

14.2.3.4 SPI 中断

有一个与 USI 模块相关的中断矢量，以及一个相对于 SPI 操作的中断标志，USIIFG。 当 USIIE 和 GIE 被置位时，中断标志将生成一个中断请求。

当 USICNTx 变为零时（被计算或者直接将 0 写入 USICNTx 位），USIIFG 被置位。 当 USIIFGCC=0，或者被软件直接写入时，通过将一个大于 0 的值写入 USICNTx 位，USIIFG 被清零。

14.2.4 I²C 模式

当 USII2C=1，USICKPL=1，并且 USICKPH=0 时，USI 模块在 I²C 模式中被配置。为了实现 I²C 数据兼容性，USILSB 和 USI16B 必须被清除。 USIPE6 和 USIPE7 必须被置位来启用 SCL 和 SDA 端口功能。

14.2.4.1 I²C 主控模式

为了将 USI 模块配置为一个 I²1C 主器件，必须将 USIMST 位置位。 在主控模式下，时钟由 USI 模块生成并且在 USIIFG=0 时输出至 SCL 线路。当 USIIFG=1 时，SCL 将在闲置，或高电平时停止。 如仲裁部分中描述的那样支持多主控运行。

只有当 USIDIVx>0 时，主控才支持将 SCL 线路保持在低电平的从器件。当 USIDIVx 被设定为 /1 时钟分频 (USIDIVx=0) 时，被连接的从器件在数据传输时不能将 SCL 线路保持在低电平。 否则通信可能失败。

14.2.4.2 I²C 受控模式

为了将 USI 模块配置为一个 I²C 从器件，USIMST 位被清零。 在受控模式，如果 USIIFG=1，USISTTIFG=1 或者如果 USICNTx=0，SCL 被保持在低电平。在从器件被设置并且已经为从一个主器件接收从地址做好准备后，USISTTIFG必须由软件清除。

14.2.4.3 I²C 发送器

在发送器模式中，数据被首先载入到 USISRL。 通过设置 USIOE，输出被启用，并且通过将 8 写入 USICNTx，传输被启动。 这清除了 USIIFG，并且 SCL 在主控模式中被清除，或者在受控模式中从被保持低电平释放。 在传输了所有 8 个位后，USIIFG 被置位，并且在主控模式中，SCL 上的时钟信号被停止或者在受控模式的下一个低相位上被保持低电平。

为了接收 I²C 确认位，USIOE 被用软件清零并且 USICNTx 被载入 1。这清除了 USIIFG 并且一个位被写入 USISRL。 当 USIIFG 再次被置位时，USISRL 的 LSB 是接收到的确认位，并且用软件测试。

```
; Receive ACK/NACKBIC.B #USIOE,&USICTL0 ; SDA inputMOV.B #01h,&USICNT ; USICNTx =
1TEST_USIIFGBIT.B #USIIFG,&USICTL1 ; Test USIIFGJZ TEST_USIIFGBIT.B #01h,&USISRL ; Test received
ACK bitJNZ HANDLE_NACK ; Handle if NACK...Else, handle ACK
```

14.2.4.4 I²C 接收器

在 I²C 接收器模式中，必须通过清除 USIOE 将输出禁用，并且通过写入 8 到 USICNTx，USI 模块为接收做好准备。 这清除了 USIIFG，并且在主控模式中生成 SCL 或者在受控模式中被从低电平保持中释放。 8 个时钟后，USIIFG 位将被置位。 这停止了主控模式中 SCL 上的时钟信号或者在受控模式中的下一个相位上将 SCL 保持在低电平。

为了传送一个确认位或者不确认位，将 0 或者 1 载入移位寄存器的 MSB，用软件将 USIOE 位置位来启用输出，并且将 1 写入 USICNTx 位。 只要 MSB 位被移出，USIIFG 将被置位，并且模块可为接收下一个 I²C 数据类型做好准备。

```
; Generate ACKBIS.B #USIOE,&USICTL0 ; SDA outputMOV.B #00h,&USISRL ; MSB = 0MOV.B #01h,&USICNT ;
USICNTx = 1TEST_USIIFGBIT.B #USIIFG,&USICTL1 ; Test USIIFGJZ TEST_USIIFG...continue...; Generate
NACKBIS.B #USIOE,&USICTL0 ; SDA outputMOV.B #0FFh,&USISRL ; MSB = 1MOV.B #01h,&USICNT ; USICNTx =
1TEST_USIIFGBIT.B #USIIFG,&USICTL1 ; Test USIIFGJZ TEST_USIIFG...continue...
```

14.2.4.5 START 条件

在 SCL 为高电平时，一个 START 条件是一个从高电平到低电平的转换。 通过将移位寄存器中的 MSB 设置为 0 可生成 START 条件。 设置 USIGE 和 USIOE 位使得输出锁存透明，并且移位寄存器中的 MSB 被立即提供给 SDA，并将线路拉至低电平。 清除 USIGE 可重新开始时钟锁存功能并在输出随 SCL 移除之前在 SDA 上保持 0。

```
; Generate STARTMOV.B #000h,&USISRL ; MSB = 0BIS.B #USIGE+USIOE,&USICTL0 ; Latch/SDA output
enabledBIC.B #USIGE,&USICTL0 ; Latch disabled...continue...
```

14.2.4.6 STOP 条件

SCL 为高电平时，一个 STOP 条件是在 SDA 上的一个低电平到高电平的转换。 为了完成确认位并且将 SDA 拉至低电平来为一个 STOP 条件生成做好准备，要求清除移位寄存器中的 MSB 并且将 1 载入 USICNTx。 这将在低相位 SDA 被拉至低电平期间在 SCL 上生成一个低相位。 由于模块处于主控模式，SCL 在闲置，或高电平状态中停止。 为了生成低电平到高电平的转换，MSB 在移位寄存器中被置位并且将 1 载入 USICNTx。设置 USIGE 和 USIOE 位使得输出锁存透明，并且 USISRL 的 MSB 将 SDA 释放至闲置状态。 清除 USIGE 将 MSB 存储在输出锁存中，并且通过清除 USIOE 来禁用输出。 SDA 保持高电平，直到有一个外部上拉电阻器导致的 START 条件生成。

```
; Generate STOPBIS.B #USIOE,&USICTL0 ; SDA=outputMOV.B #000h,&USISRL ; MSB = 0MOV.B #001h,&USICNT
; USICNT = 1 for one clockTEST_USIIFGBIT.B #USIIFG,&USICTL1 ; Test USIIFGJZ test_USIIFG ;MOV.B
#0FFh,&USISRL ; USISRL = 1 to drive SDA highBIS.B #USIGE,&USICTL0 ; Transparent latch
enabledBIC.B #USIGE+USIOE,&USICTL; Latch/SDA output disabled...continue...
```

14.2.4.7 释放 SCL

如果它被 USI 模块保持在低电平，在无需清除 USIIFG 的情况下，设置 USISCLREL 位将释放 SCL。 如果一个 START 条件被接收并且 SCL 线路将在下一个时钟上被保持低电平，USISCLREL 位将被自动清除。

在受控模式运行中，当从器件已经检测到它不由主器件编址时，这个位应该被用来防止 SCL 被保持低电平。 在下一个 START 条件时，USISCLREL 将被清除并且 USISTTIFG 将被置位。

14.2.4.8 仲裁

USI 可检测一个多主控 I^2C 系统中失败的仲裁条件。 I^2C 仲裁过程通过发送器竞争来使用出现在 SDA 上的数据。 第一个生成逻辑高电平的主控发射器在与相对生成逻辑低电平的主控发射器竞争时失败。 通过比较出现在总线上的值和从总线中读取的值可检测 USI 模块中的仲裁失败。 如果这两个值不相等，那么仲裁失败并且仲裁丢失标志，USIAL，被置位。 这也将清除输出使能位 USIOE 并且 USI 模块不再驱动总线。 在这个情况下，用户软件必须同时检查 USIAL 标志和 USIIFG 并且当仲裁失败时将 USI 配置为从接收器 USIAL 标志必须由软件清除。

为了防止其它更快速的主控在仲裁过程期间生成时钟，如果总线上的其它主控将 SCL 驱动为低电平并且 USIIFG 或 USISTTIFG 被置位，或者如果 USICNTx=0 的话，SCL 被保持在低电平。

14.2.4.9 I^2C 中断

这是一个中断矢量，它通过两个针对 I^2C 运行的中断标志，USIIFG 和 USISTTIFG，与 USI 模块相关联。 每个中断标志由其自己的中断使能位，USIIE 和 USISTTIE。 当一个中断被启用并且 GIE 位被置位时，一个被置位的中断标志将生成一个中断请求。

当 USICNTx 变为零（通过计数或者通过直接向 USICNTx 位写入 0），USIIFG 被置位。 当 USIIFGCC=0 时，通过写入一个大于 0 的值到 USICNTx 位，USIIFG 被清除，或者直接由软件清除。

当检测到一个 START 条件时，USISTTIFG 被置位。 USISTTIFG 必须由软件清除。

由 USISTP 标志表示接收到一个 START 条件，但是没有与 USISTP 标志相关的中断功能。 当 USIIFGCC=0 时，通过将一个大于 0 的值写入 USICNTx 位，USISTP 被清零，或者被软件直接清除。

14.3 USI 寄存器

表 14-1中列出了 USI 寄存器。

表 14-1. USI 寄存器

寄存器	简氏	寄存器类型	地址	初始状态
USI 控制寄存器 0	USICTL0	寄存器	078h	带有 PUC 的 01h
USI 控制寄存器 1	USICTL1	读取/写入	079h	带有 PUC 的 01h
USI 时钟控制	USICKCTL	读取/写入	07Ah	带有 PUC 的复位
USI 位计数器	USICNT	读取/写入	07Bh	带有 PUC 的复位
USI 低字节移位寄存器	USISRL	读取/写入	07Ch	未经改变
USI 高字节移位寄存器	USISRH	读取/写入	07Dh	未经改变

可使用表 14-2中显示的字指令访问 USI 寄存器。

表 14-2. 到 USI 寄存器的字访问

寄存器	简氏	高字节寄存器	低字节寄存器	地址
USI 控制寄存器	USICTL	USICTL1	USICTL0	078h
USI 时钟和计数器控制寄存器	USICCTL	USICNT	USICKCTL	07Ah
USI 移位寄存器	USISR	USISRH	USISRL	07Ch

14.3.1 USICTL0，USI 控制寄存器 0

7	6	5	4	3	2	1	0
USIPE7	**USIPE6**	**USIPE5**	**USILSB**	**USIMST**	**USIGE**	**USIOE**	**USISWRST**
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1

USIPE7 位 7 USI SDI/SDA 端口启用。 SPI 模式中的输入，I^2C 模式中的输入或者开漏输出。
- 0 USI 功能被禁用
- 1 USI 功能被启用

USIPE6 位 6 USI SDO/SCL 端口启用。 SPI 模式中的输出，I^2C 模式中的输入或者开漏输出。
- 0 USI 功能被禁用
- 1 USI 功能被启用

USIPE5 位 5 USI SCLK 端口启用。 SPI 受控模式或 I^2C 模式中的输入，SPI 主控模式中的输出。
- 0 USI 功能被禁用
- 1 USI 功能被启用

USILSB 位 4 首先选择 LSB。 这个位控制接收和发送移位寄存器的方向。
- 0 MSB 优先
- 1 最低有效位 (LSB)优先

USIMST 位 3 主器件选择
- 0 从器件模式
- 1 主器件模式

USIGE 位 2 输出锁存控制
- 0 输出锁存启用取决于移位时钟
- 1 输出锁存一直被启用并且透明

USIOE 位 1 数据输出启用
- 0 输出被禁用
- 1 输出被启用

USISWRST 位 0 USI 软件复位
- 0 USI 被释放进行操作。
- 1 USI 逻辑被保持在复位状态，

14.3.2 USICTL1，USI 控制寄存器 1

7	6	5	4	3	2	1	0
USICKPH	**USII2C**	**USISTTIE**	**USIIE**	**USIAL**	**USISTP**	**USISTTIFG**	**USIIFG**
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1

USICKPH 位 7 时钟相位选择
- 0 数据在第一个 SCLK 边沿上被改变并且在下一个边沿上被捕捉。
- 1 数据在第一个 SCLK 边沿上被捕捉并且在下一个边沿上被改变。

USII2C 位 6 I^2C 模式启用
- 0 I^2C 模式被禁用
- 1 I^2C 模式被启用

USISTTIE 位 5 START 条件中断-启用
- 0 START 条件上的中断被禁用
- 1 START 条件上的中断被启用

USIIE 位 4 USI 计数器中断启用
- 0 中断被禁用
- 1 中断被启用

USIAL 位 3 仲裁失败
- 0 无仲裁失败条件
- 1 仲裁失败

USISTP 位 2 STOP 条件被接收。 当 USIIFGCC=0 时，如果 USICNTx 被载入一个大于 0 的值，USISTP 被自动清除。
- 0 没有接收到 STOP 条件。
- 1 接收到 STOP 条件

USISTTIFG 位 1 START 条件中断标志
- 0 没有接收到 START 条件。 无中断挂起。
- 1 接收到 START 条件。 中断挂起

USIIFG 位 0 USI 计数器中断标志。 当 USICNTx=0 时置位。当 USIIFGCC=0 时，如果 USICNTx 被载入一个大于 0 的值，被自动清除。
- 0 无中断挂起
- 1 中断挂起

14.3.3 USICKCTL，USI 时钟控制寄存器

7	6	5	4	3	2	1	0
USIDIVx			**USISSELx**			**USICKPL**	**USISWCLK**
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

USIDIVx 位 7-5 时钟分频选择

000 除以 1
001 除以 2
010 除以 4
011 除以 8
100 除以 16
101 除以 32
110 除以 64
111 除以 128

USISSELx 位 4-2 时钟源选择。 未在受控模式下使用。

000 SCLK（未在 SPI 模式中使用）
001 ACLK
010 SMCLK
011 SMCLK
100 USISWCLK 位
101 TACCR0
110 TACCR1
111 TACCR2（在 MSP430F20xx 器件上被保留）

USICKPL 位 1 时钟极性选择

0 无效状态为低电平
1 无效状态为高电平

USISWCLK 位 0 软件时钟

0 输入时钟为低电平
1 输入时钟为高电平

14.3.4 USICNT，USI 位计数器寄存器

7	6	5	4	3	2	1	0
USISCLREL	**USI16B**	**USIIFGCC**	**USICNTx**				
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

USISCLREL 位 7 SCL 释放。 SCL 线路被从低电平释放到闲置状态。 如果检测到一个 START 条件，USISCLREL 被清除。

0 如果 USIIFG 被置位，SCL 线路被保持在低电平
1 SCL 线路被释放

USI16B 位 6 16 位移位寄存器启用

0 8 位移位寄存器模式。 低字节寄存器 USISRL 被使用。
1 16 位移位寄存器模式。 高字节和低字节寄存器 USISRL 和 USISRH 都被使用。 USISR 同时寻址所有 16 位。

USIIFGCC 位 5 USI 中断标志清除控制。 当 USIIFGCC=0 并且 USICNTx 被写入一个大于 0 的值时，USIIFG 将不能被自动置位。

0 USICNTx 更新时，USIIFG 被自动清除
1 USIIFG 不被自动清除

USICNTx 位 4-0 USI 位计数。 USICNTx 位设定将被接收或发出的位的数量。

14.3.5 USISRL，USI 低字节移位寄存器

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

USISRLx 位 7-0 USI 低字节移位寄存器的内容

14.3.6 USISRH，USI 高字节移位寄存器

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

USISRHx 位 7-0 USI 高字节移位寄存器的内容 当 USI16B=0 时被忽略。

通用串行通信接口，*UART* 模式

通用串行通信接口 (USCI) 在同一个硬件模块下支持多路串行通信模式。 本章讨论了异步 UART 模式的操作。

Topic **Page**

15.1 USCI 概述

通用串行通信接口 (USCI) 模块支持多路串行通信模式。 不同的 USCI 模块支持不同的模式。 每种不同的 USCI 模块以一个不同的字母命名。 例如，USCI_A 就不同于 USCI_B， 等等。 如果在一个设器件上执行了不止一个相同的 USCI 模块，那么这些模块将以递增的数字命名。 例如，当一个器件有两个 USCI_A 模块时，它们应该被命名为 USCI_A0 和 USCI_A1。 如有需要，请参阅《器件专用数据表》来确定哪些 USCI 模块可以在哪些设备上执行。

USCI_Ax 模块支持：

- UART 模式
- IrDA 通信的脉冲整形
- LIN 通信的自动波特率检测
- SPI 模式

USCI_Bx 模块支持：

- I^2C 模式
- SPI 模式

15.2 USCI 介绍：UART 模式

在异步模式中，USCI_Ax 模块通过两个外部引脚，UCAxRXD 和 UCAxTXD，把 MSP430 和一个外部系统连接起来。 当 UCSYNC 位被清零时就选择了 UART 模式。

UART 模式的特性包括：

- 7 或 8 位奇，偶，或无奇偶校验的数据
- 独立的发送和接收移位寄存器
- 独立的发送和接收缓冲寄存器
- 最低有效位 (LSB) 优先或最高有效位 (MSB) 优先的数据发送和接收
- 多处理器系统中内置空闲线和地址位通信协议
- 接收器开始边沿检测从 LMPx 模式中自动唤醒
- 支持分数波特率的可编程调制波特率
- 状态标志的错误检测和抑制
- 地址检测的状态标志
- 独立接收和发送中断的能力

图 15-1给出了配置为 UARG 模式的 USCI_Ax。

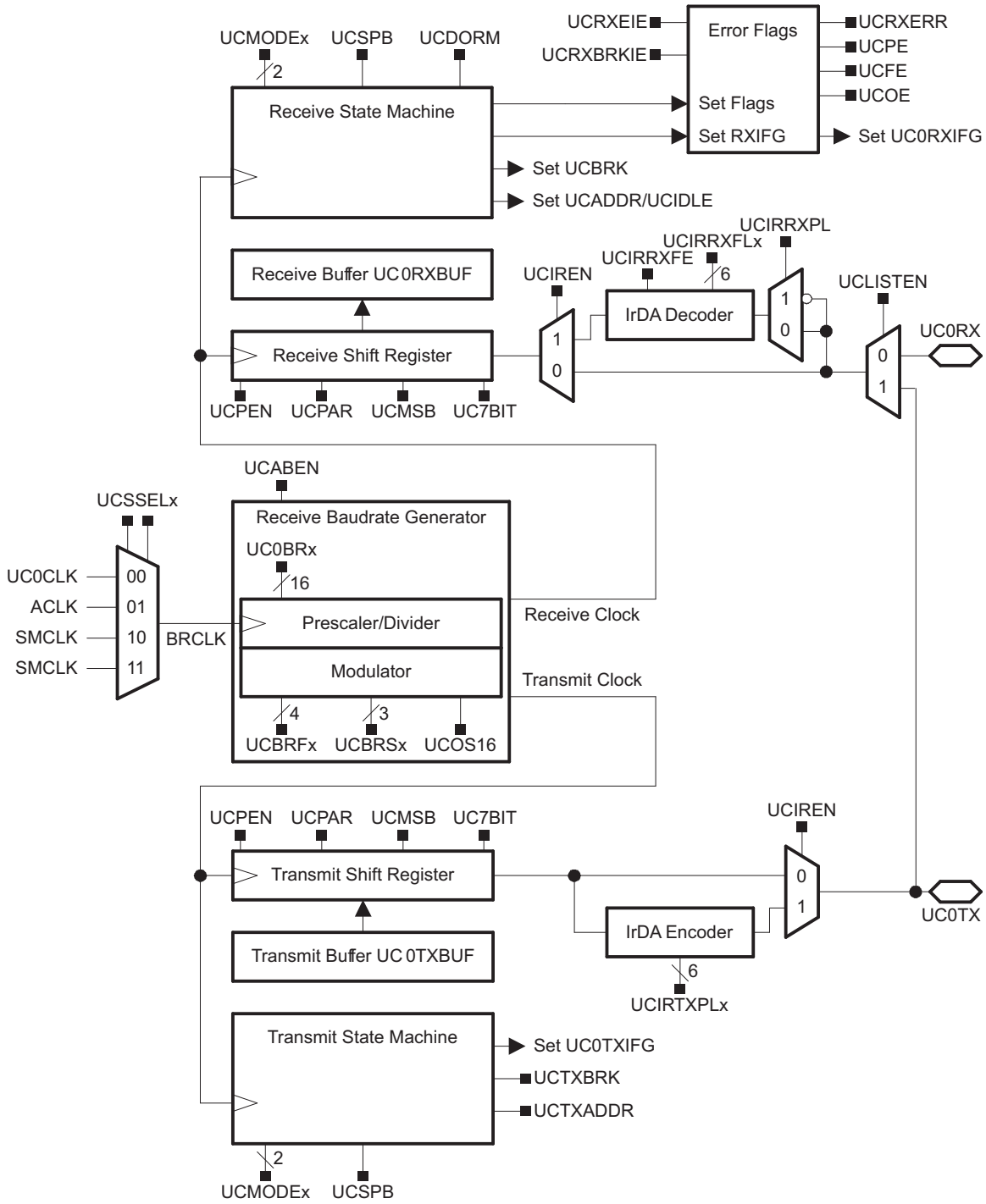


图 15-1. USCI_Ax 框图：UART 模式 (UCSYNC=0)

15.3 USCI 操作：UART 模式

在 UART 模式下，USCI 发送和接收的字符以每一个位速率异步于到另一个器件。 每个字符传输的计时取决于所选的 USCI 波特率。 传输和接收功能使用相同的波特率。

15.3.1 USCI 初始化和复位

USCI 在一个 PUC 后或者通过设置 UCSWRST 位来复位。 在一个 PUC 后，UCSWRST 位自动置位，保持了 USCI 处于一个复位状态。 当置位时，UCSWRST 位复位 UCAxRXIE，UCAxTXIE，UCAxRXIFG，UCRXERR，UCBRK，UCPE，UCOE，UCFE，UCSTOE 和 UCBTOE 位置位 UCAxTXIFG 位。 清零 UCSWRST 释放了 UCSI 的操作。

注: **初始化或者重新配置 USCI 模块**

建议的 USCI 初始化/重配置过程如下：

1. 置位 UCSWRST (BIS.B #UCSWRST，&UCAxCTL1)
2. UCSWRST=1 时初始化所有 UCSI 寄存器（包括 UCAxCTL1）
3. 配置端口
4. 软件清除 UCSWRSTBIC.B #UCSWRST，&UCAxCTL1)
5. 通过 UCAxRXIE 和/或 UCAxTXIE 是使能中断（可选）

15.3.2 字符格式

UART 的字符格式，展示在图 15-2中，包括一个开始位，7 个或 8 个数据位，一个奇/偶/无校验位，一个地址位（地址位模式），和一个或两个停止位。 UCMSB 位控制传输的方向以及优先选择 LSB 还是 MSB。 UART 通信典型地要求先发送 LSB。

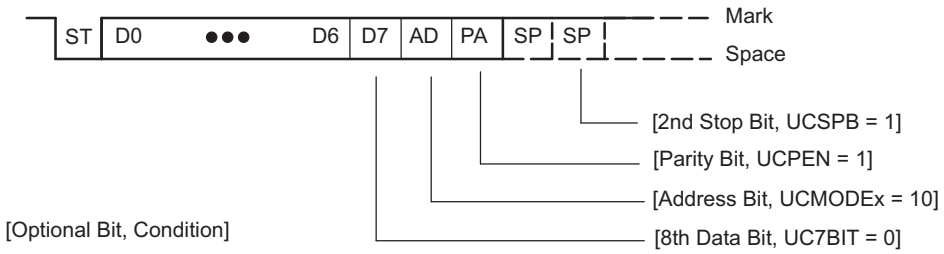


图 15-2. 字符格式

15.3.3 异步通信格式

当两个器件异步通信时，协议不需要多处理器格式。 当三个或更多的器件通信时，USCI 支持空闲线和地址位多处理器通信格式。

15.3.3.1 空闲线多处理器格式

当 UCMODEx=01 时，空闲线多处理器格式被选中. 数据块在发送或接收线上被一段空闲时间隔开，如图 15-3所示。 在一个字符的一个或两个停止位后，当接收到 10 个或更多的持续标志（标志）时，一条空闲接收线就会被监测。 在接收到一条空闲线后，直到下一次开始边沿被监测到时波特率发生器才被切断。 当检测到一条空闲线路时，UCIDLE 就置位。

在一个空闲周期之后接收的第一个字符是地址字符。 UCIDLE 位被用作每个字符块的地址标签。 在线路空闲多处理器模式下，当接收的一个字符是一个地址时该位就会被置位。

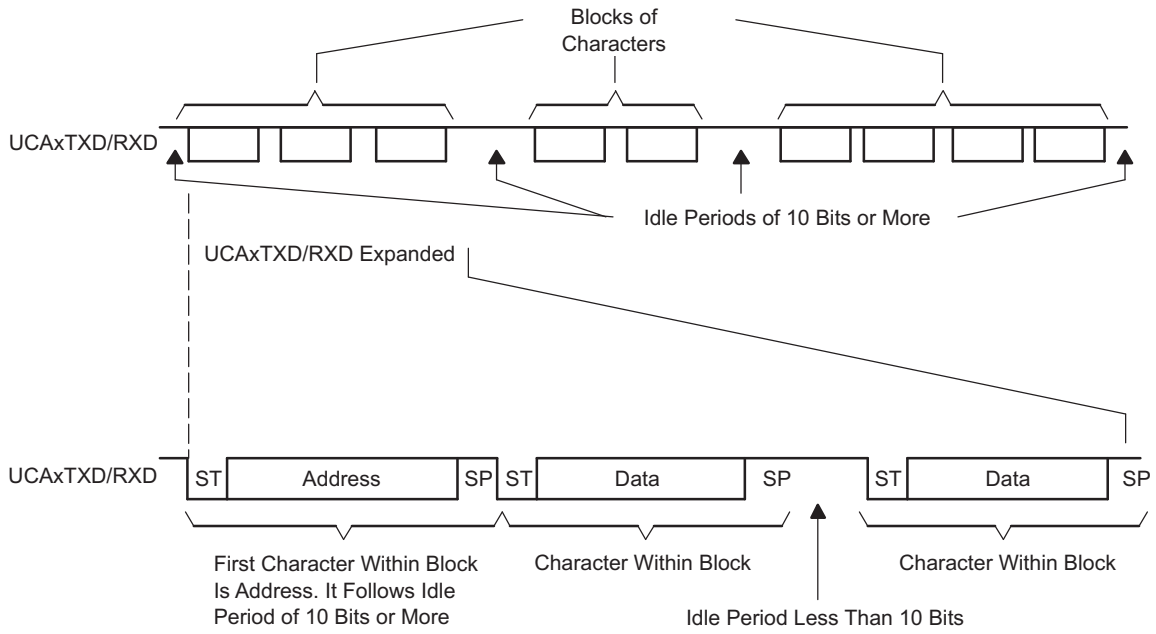


图 15-3. 空闲线格式

在多处理器模式下的 UCDORM 位被用于控制数据接收。 当 UCDORM=1 时，所有非地址字符被拼装起来但不会被移送到 UCAxRXBUF，也不会产生中断。 当接收到一个地址字符时，该字符被移送到 UCAxRXBUF，UCAxRXIFG 被置位，当 UCRXEIE=1 时任何可用的错误标志都被置位。当 UCRXEIE=0 并且接收到一个地址字符，但该字符发生了帧错误或奇偶错误，字符不会被移送到 UCAxRXBUF，UCRXIFG 也不会置位。

如果接收到一个地址，用户软件可以验证此地址且必须复位 UCDORM 才可以继续接收数。 如果 UCDORM 保持置位，将只能接收地址字符。 在接收一个字符期间若 UCDORM 被清零，则在接收完成后接收中断标志将被置位。 UCDORM 位不会被 USCI 硬件自动修改。

对在空闲线多处理器模式下发送地址而言，在 UCAxTXD 上产生地址字符标识的 USCI 会产生一个精确的空闲周期。 如果下一个字符先于 11 位的空闲线被载入 UCAxTXBUF，这将通过双缓存 UCTXADDR 标志展示出来。 当开始位发生时 UCTXADDR 将被自动清零。

15.3.3.2 发送一个空闲帧

以下程序通过发送一个空闲帧来表示一个地址字符及随后其关联的数据：

1. 置位 UCTXADDR，然后把地址字符写入 UCAxTXBUF。 UCAxTXBUF 必须为新数据做好准备 (UCAxTXIFG=1)。

   这可以产生一个地址字符之后的 11 位空闲周期。 当地址字符从 UCAxTXBUF 中传输到移位寄存器时，UCTXADDR 自动复位。
2. 在 UCAxTXBUF 中写入预期数据。 UCAxTXBUF 必须为新数据做好准备 (UCAxTXIFG=1)。

   写入 UCAxTXBUF 中的数据被传输到移位寄存器中并且一旦移位寄存器为新数据做好准备就开始发送。

   空闲周期不能超过地址和数据传输之间或者数据和数据传输的时间。 否则传输的数据将被误解为一个地址。

15.3.3.3 地址位多处理器格式

当 UCMODEx=10 时，地址位多处理器格式被选中。 如图 15-4所示，每个已处理的字符包含一个用作一个地址指示的额外位。 一个字符块的第一个字符带有一组指示字符是一个地址的地址位。 当一个接收到的字符有其自身的地址位组并被转移到 UCAxRXBUF 时，USCI UCADDR 位被置位。

在地址位多处理器模式下，UCDORM 位被用于控制数据接收。 当 UCDORM 被置位，地址位=0 的数据字符被接收器组装但，不会传输到 UCAxRXBUF 中，且没有产生中断。 当接收到一个包含一组地址位的字符时，该字符被移送到 UCAxRXBUF，UCAxRXIFG 被置位，且当 UCRXEIE=1 时任何可用的错误标志都被置位。当 UCRXEIE=0 并且接收到一个包含一组地址位的字符，但该字符发生了帧错误或奇偶错误，字符不会被移送到 UCAxRXBUF，UCRXIFG 也不会置位。

如果接收到一个地址，用户软件可以验证此地址且必须复位 UCDORM 才可以继续接收数。 如果 UCDORM 保持置位，将只能接收地址位=1 的地址字符。 UCDORM 位不会被 USCI 硬件自动修改。

当 UCDORM=0 时所有已接收的字符将置位中断标志 UCAxRXIFG。 在接收一个字符期间若 UCDORM 被清零，则在接收完成后接收中断标志将被置位。

对于在地址位多处理器模式下的地址传输，字符的地址位是由 UCTXADDR 位控制。 UCTXADDR 位的值被载进被从 UCAxTXBUF 传送到发送移位寄存器中的字符的地址位。 当开始位发生时 UCTXADDR 将被自动清零。

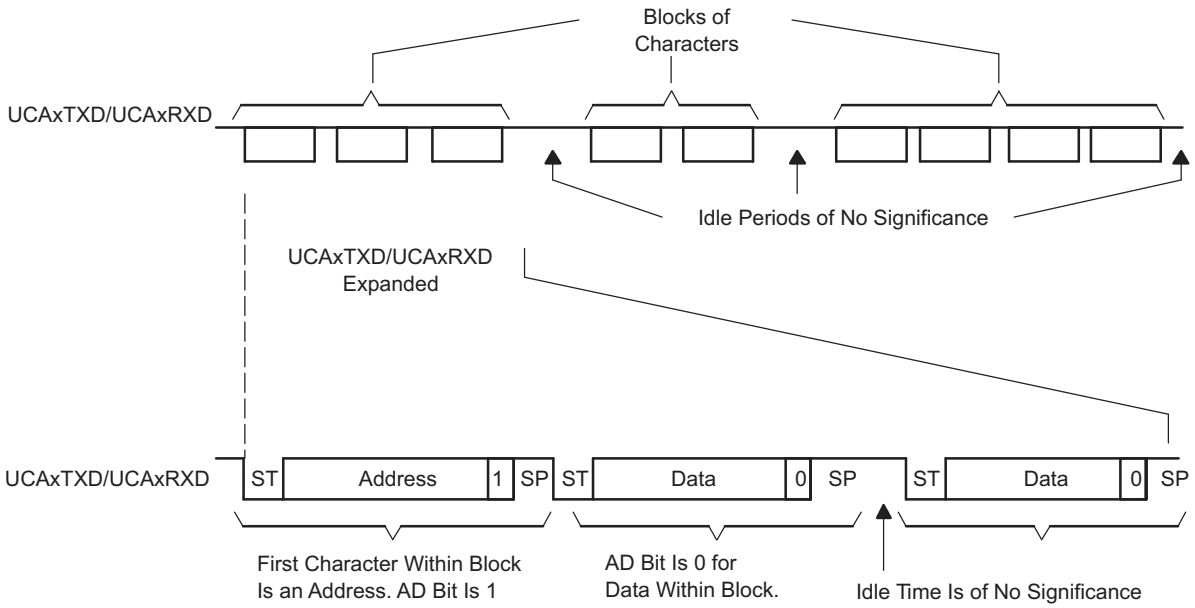


图 **15-4.** 地址位多处理器格式

15.3.3.4 中断接收和生成

当 UCMODEX=00，01 或 10 时，且不考虑奇偶位，地址模式，或其它字符值，当所有数据，奇偶校验，停止位都为低电平时，接收器在监测到一次中断。 当监测到一次中断时，UCBRK 位置位。 如果中断的中断使能位，UCBRKIE，被置位，接收中断标志 UCAxRXIFG 也将被置位。 在这种情况下，因为所有数据位是 0 所以 UCAxRXBUF 中的值也是 0。

为了发送一个中断，置位 UCTXBRK 位，然后把 0h 写到 UCAxTXBUF 中。 UCAxTXBUF 必须为新数据做好准备 (UCAxTXIFG=1)。 在所有位为低电平时会产生一个中断。 当开始位发生时 UCTXBRK 将被自动清零。

15.3.4 自动波特率检测

当 UCMODEx=11 时，选择了带自动波特率检测的 UART 模式。 对于自动波特率检测，数据帧在一个包含一个中断和一个同步域的同步序列的前面。 当 11 个或更多的 0（空格）被接收到时监测到一个中断。 如果中断长度超过 22 位的时间，暂停超时错误标志 UCBTOE 将被置位。 暂停的异步域如图 15-5所示。

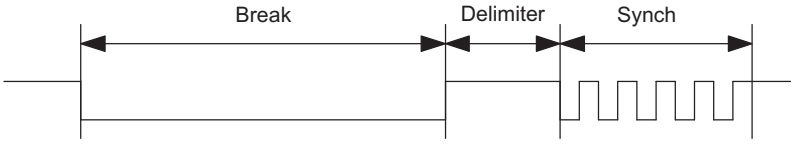


图 15-5. 自动波特率监测-暂停/同步序列

为了 LIN 一致字符格式应该设置为 8 数据位，LSB 优先，无奇偶校验位和停止位。 没有可用的地址位。

在一个字节域内同步域所包含的数据 055H 如图 15-6所示。 同步的是基于该模式的第一个下降沿和最后一个下降沿之间的时间测量。 如果自动波特率监测通过置位 UCABDEN 使能，发送波特率发生器就可以用于测量。 否则，这个模式只被接收但不被测量。 测量的结果被传送到波特率控制寄存器 UCAxBR0，UCAxBR1，和 UCAxMCTL 中。 如果同步域的长度超过了测量时间，同步超时错误标志 UCSTOE 将被置位。

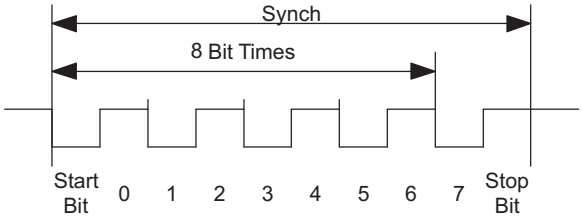


图 15-6. 自动波特率监测-同步域

在这种模式中 UCDORM 位被用来控制数据接收。 当 UCDORM 被置位时，所有字符被接收但不会被传输到 UCAxRXBUF 中，而且不会发生中断。 当一个中断/同步域被监测到时，UCBRK 标志被置位。 随后的中断/同步域的字符被传送到 UCAxRXBUF 中且 UCAxRXIFG 中断标志被置位。 任何可用的错误标志也会置位。 如果 UCBRKIE 位被置位，中断/同步的接收会置位 UCAXRXIFG。 通过读取接收缓存 UCAxRXBUF，或通过用户软件来置位 UCBRK 位。

当一个中断/同步域被接收时，为继续接收数据用户必须用软件置位 UCDORM。 如果 UCDORM 保持置位状态，只有在接受下一个中断/同步域后字符才能被接收。 UCDORM 位不会被 USCI 硬件自动修改。

当 UCDORM=0 时所有已接收的字符将置位中断标志 UCAxRXIFG。 在接收一个字符期间若 UCDORM 被清零，则在接收完成后接收中断标志将被置位。

可以在一个带有一些限制的全双工通信系统中使用自动波特率检测模式。 USCI 可以在接收中断/同步域期间不发送数据，并且如果一个帧错误字节 0h 被接收，那么在这段时间内任何传输的数据都会被损坏。 可以通过检查所接收的数据和 UCFE 位来发现后一种情况。

15.3.4.1 发送一个中断/同步域

以下为发送一个中断/同步域的程序流程：

- 在 UMODEX=11 时，置位 UCTXBRK。
- 把 055h 写入 UCAxTXBUF。 UCAxTXBUF 必须为新数据做好准备 (UCAxTXIFG=1)。

  伴随着中断分割符和同步字符将会产生一个 13 位的中断域。 中断定界符的长度由 UCDELIMX 位控制。 当同步字符从 UCAxTXBUF 传输到移位寄存器中时 UCTXBRK 将自动复位。
- 在 UCAxTXBUF 中写入所需的数据。 UCAxTXBUF 必须为新数据做好准备 (UCAxTXIFG=1)。

  写入 UCAxTXBUF 中的数据被传输到移位寄存器中并且一旦移位寄存器为新数据做好准备就开始发送。

15.3.5 IrDA 编码和解码

当 UCIREN 被置位时，IrDA 解码器和译码器被使能并提供修整 IrDA 通信的硬件位。

15.3.5.1 IrDA 编码

如图 15-7所示，在来自 UART 的发送位流中译码器为每个 0 位发送一个脉冲。 脉冲持续时间由 UCIRTXPLx 位决定，来指定被 UCIRTXCLK 选中的半个时钟周期的数目。

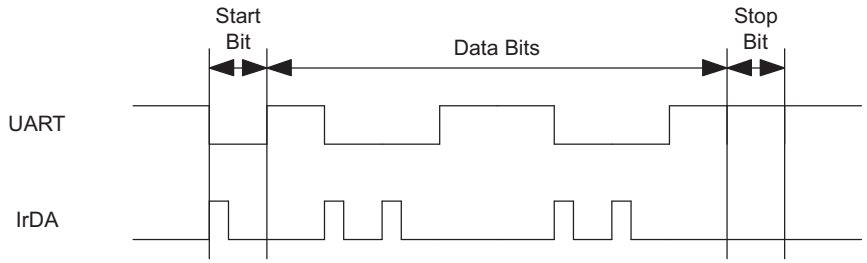


图 15-7. UART 与 IrDA 数据格式的关系

要求根据 IrDA 标准去设置 3/16 位的周期脉冲，通过置 UCIRTXCLK=1 来选中 BITCLK16 时钟，且脉冲长度由 UCIRTXPLx=6-1=5 来设置为 6 个半时钟循环。

当 UCIRTXCLK=0 时，基于 BRCLK 的脉冲长度 $t_{脉冲}$由以下公式计算：

$$UCIRTXPLx = t_{PULSE} \times 2 \times f_{BRCLK} - 1$$

当脉冲长度是基于 BRCLK 的时，计数器 UCBRx 必须被设置成一个大于或等于 5 的值.

15.3.5.2 IrDA 解码

当 UCIRRXPL=0 时解码器监测到高脉冲.否则就监测到低脉冲。 除模拟抗尖峰脉冲滤波器外，额外的可编程数字滤波器阶段也能通过设置 UCIRRXFE 来是使能。 当 UCIRRXFE 被置位时，只有比可编程滤波器长的脉冲才能通过。 较短脉冲被丢弃。 编程滤波器长度 UCIRRXFLx 的方程式如下：

$$UCIRRXFLx = (t_{PULSE} - t_{WAKE}) \times 2 \times f_{BRCLK} - 4$$

其中，

$t_{脉冲}$=接受脉冲的最小宽度

$t_{唤醒}$=从任何低功耗模式中唤醒。 当 MSP430 处于活动模式时为 0。

15.3.6 自动错误检测

毛刺抑制会阻止 USCI 的突然启动。 在 UCAxRXD 上的任何小于抗尖峰脉冲时间 t_τ（大约 150NS）的脉冲都被忽略. 有关参数请参阅《器件专用数据表》。

当在 UCAxRXD 上的一个低电平周期超过 t_τ时，多数票决都会被当作开始位。 如果多数票决没有监测到一个有效的开始位，USCI 将暂停字符接收同时等待 UCAxRXD 上的下一个低电平周期。 多数票决也用于一个字符的每个位来防止位错误。

当接收字符时，USCI 模块自动监测帧错误，奇偶校验错误，溢出错误，以及中断条件。 在各自的情况被监测到时，UCFE，UCPE，UCOE，以及 UCBRK 位被置位。 当 UCFE，UCPE 或 UCOE 错误标志被置位时，UCRXERR 也被置位。 在表 15-1中描述了错误条件。

表 15-1. 接收错误条件

错误条件	错误标志	说明
组帧错误	UCFE	当一个低电平停止位被监测到时发生一个组帧错误。 当使用两个停止位时，这两个位都会被检查是否有组帧错误。 当检测到一个组帧错误时，UCFE 位被置位。
奇偶校验错误	UCPE	一个奇偶校验错误是一个字符中 1 的个数和奇偶校验位的值之间的一个不匹配。 当地址位被包含在字符中时，它被包含在奇偶校验计算中。 当监测到一个奇偶错误时，UCPE 位被置位。
接收溢出	UCOE	当在读出前一个字符之前一个字符被载入 UCAxRXBUF 中时，会引发一个溢出错误。 当溢出错误发生时，UCOE 位被置位。
中断条件	UCBRK	当不使用自动波特率监测时，在所有数据，奇偶校验，和停止位为低电平时，监测到一个中断。 当监测到一次中断条件时，UCBRK 位置位。 如果暂停中断使能 UCBRKIE 位被置位，一个中断条件也可以置位其中断标志 UCAXRXIFG。

当 UCRXEIE=0 时且监测到一个帧错误，或奇偶校验错误时，不会有字符被接收到 UCAxRXBUF 中。 当UCRXEIE=1 时，字符被接收到 UCAxRXBUF 中且任何适用的错误位被置位。

当 UCFE，UCPE，UCOE，UCBRK，或 UCRXEER 被置位时，该位一直保持到用户用软件复位或 UCAxRXBUF 被读出。 UCOE 必须通过读取 UCAxRXBUF 复位。 否则，它将不能正常工作。 为了可靠地检测溢出，建议使用以下流程。 在一个字符被接收且 UCAxRXIFG 被置位后，首先读取 UCAxSTAT 来检查包括溢出标志 UCOE 在内的错误标志。 接下来读取 UCAxRXBUF。 如果 UCAxRXBUF 在 UCAxSTAT 和 UCAxRXBUF 的读取访问之间被覆写，则除了 UCOE 以外的所有错误标志都会被清零。 为了检测这个条件，在读完 UCAxRXBUF 后应该检查 UCOE 标志。 注意，在这种情况下，UCRXERR 标志不会被置位。

15.3.7 USCI 接收使能

通过清零 UCSWRST 位可以使能 USCI 模块，接收端准备就绪且处于一个空闲状态。 接收波特率发生器处于准备状态但是不计时也不产生任何时钟。

开始位的下降沿使能波特率发生器同时 UART 状态机检查一个有效的开始位。 如果没有监测到有效的开始位，UART 状态机返回到其空闲状态且波特率发生器再次被关掉。 如果一个有效的开始位被监测到，一个字符就能被接收。

UCMODEX=01，当空闲线多处理器模式被选中时，UART 状态机在接收一个字符后检查空闲线。 如果监测到一个开始位，则另一个字符会被接收。 否则在接收 10 个字符后 UCIDLE 标志会被置位，同时 UART 状态机返回到空闲态且波特率发生器被关掉。

15.3.7.1 接收数据毛刺脉冲抑制

毛刺脉冲抑制会阻止 USCI 的突然启动。 在 UCAxRXD 上的任何小于去毛刺脉冲时间 t_t（大约 150NS）的脉冲都被 USCI 忽略，同时进一步的行为会发生，如图图 15-8所示。 有关参数请参阅《器件专用数据表》。

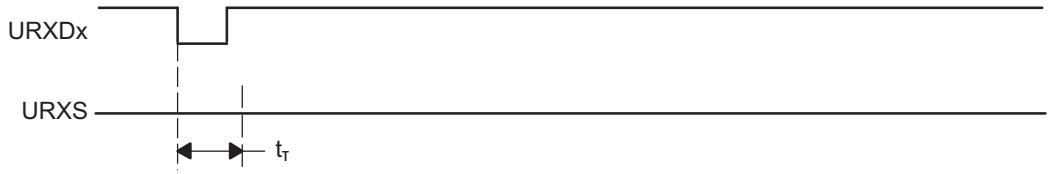


图 **15-8.** 去毛刺脉冲抑制，**USCI** 接收未开始

当一个毛刺脉冲时间大于 t_t或一个有效的开始位发生在 UCAxRXD 上时，USCI 接收操作开始，在图 15-9中给出了多数票决。 如果绝大多数表决没有监测到一个开始位，USCI 将停止字符的接收。

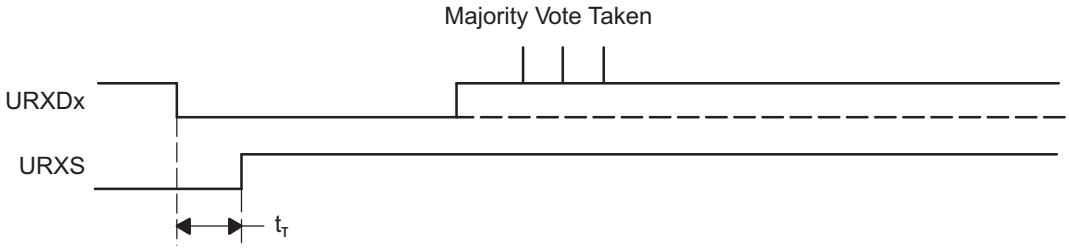


图 **15-9.** 毛刺脉冲抑制，**USCI** 启动

15.3.8 USCI 发送使能

通过清零 UCSWRST 位可以使能 USCI 模块，发送端准备就绪且处于一个空闲状态。 接收波特率发生器处于准备状态但是不被计时也不产生任何时钟。

通过把数据写入 UCAxTXBUF 中来初始化一次传输。 当这发生后，波特率发生器被使能，在发送移位寄存器为空后，在下一个BITCLK 周期 UCAxTXBUF 中的数据就被移到发送移位寄存器。 当新数据被写入 UCAxTXBUF 时 UCAxTXIFG 被置位。

在前一个字节发送结束时，只要在 UCAxTXBUF 中的新数据有用，发送就会持续进行。 在前一个字节已发送时，如果在 UCAxTXBUF 中没有新数据，发送器就返回到空闲态并且波特率发生器被关掉。

15.3.9 UART 波特率生成

USCI 波特率发生器能从非标准源频率中产生一个标准的波特率。 它通过 UCOS16 位提供了两种操作模式。

15.3.9.1 低频率波特率生成

当 UCOS16=0 时，低频模式被选中。这种模式允许波特率从低频时钟源中产生（比如，一个 32768HZ 晶振产生一个 9600 波特率） 通过使用一个较低的输入频率，能减少模块的能量消耗。 通过使用这种模式，更高频率和更高预分频器的设置将导致在一个不断增加的小窗口中绝大多数表决采用，这样就降低了绝大多数表决的优势。

在低频模式中波特率发生器使用一个预分频器和一个调节器产生位时钟时序。 这种组合支持波特率生成的分数除数。 在这种模式下，最大的 USCI 波特率是时钟源频率的 1/3。

在图 15-10中给出了每一位的时序。 对于接收到的每一位，采用了一个绝大多数表决来决定位值。 这些采样发生在 N/2-1/2，N/2，和 N/2+1/2 BRCLK 周期，其中 N 是每个 BITCLK 周期中 BRCLK 的数目。

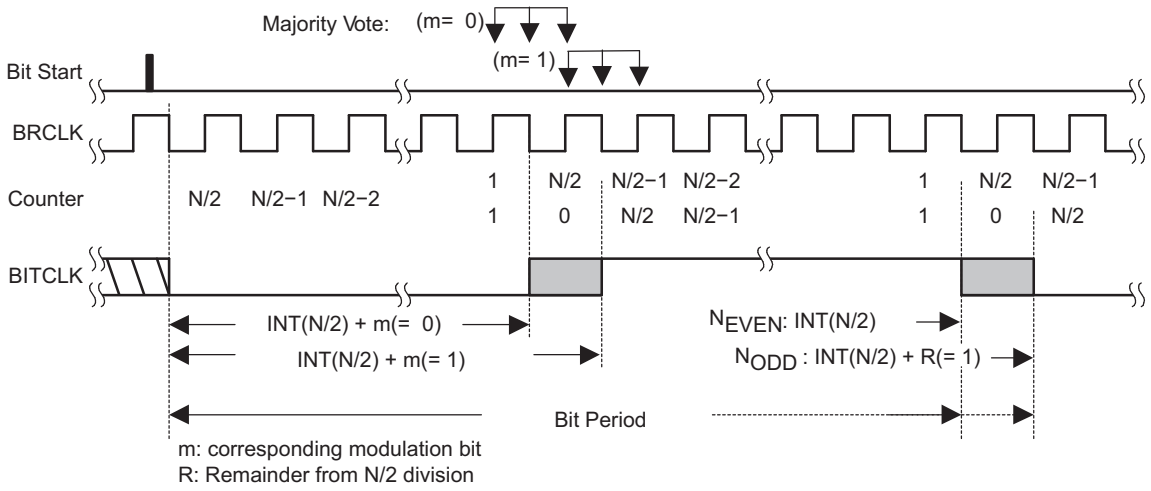


图 15-10. BITCLK 波特率用 UCOS16=0 定时

基于 UCBRSX 设置的调制如表 15-2所示。 表中 A1 表示 m=1 且相应的 BITCLK 周期是一个比 m=0 时的一个 BITCLK 周期长的周期。在 8 位后，调制互相环绕但是随着每一个新的开始位会重新启动。

表 15-2. BITCLK 的调制模式

UCBRSx	位 0（开始位）	位 1	位 2	位 3	位 4	位 5	位 6	位 7
0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0
2	0	1	0	0	0	1	0	0
3	0	1	0	1	0	1	0	0
4	0	1	0	1	0	1	0	1
5	0	1	1	1	0	1	0	1
6	0	1	1	1	0	1	1	1
7	0	1	1	1	1	1	1	1

15.3.9.2 过采样波特率生成

当 UCOS16=1 时选择过采样模式。该模式支持采样一个输入时钟频率较高的 UART 位流。 这就导致大数票决总是一个位时钟周期的 1/16 的结果。 当 IrDA 编码器和解码器被使能时，这种模式也很容易支持 3/16 位时间的 IrDA 脉冲。

该模式使用一个预分频器和调制器来产生比 BITCLK 快 16 倍的 BITCLK16。 一个额外的分频器和调制器级从 BITCLK16 中产生 BITCLK。 波特率产生该组合支持 BITCLK16 和 BITCLK。 在这种模式下，最大 USCI 波特率是 UART 源时钟频率 BRCLK 的 1/16。 当 UCBRx 被设置为 0 或 1 时，预分频器和调制器第一个阶段被旁路且 BRCLK 等于 BITCLK16。

在表 15-3中给出了基于 UCBRFx 设置的 BITCLK16 的调制。 表中 A 1 表示相应的 BITCLK16 是一个比 m=0 的周期长的 BRCLK 周期。 用每一个新的位定时重新启动调制。

在表 15-2中给出了如先前所述的基于 UCBRSx 设置的 BITCLK 调制。

表 15-3. BITCLK 的调制模式

UCBRFx	最后一个下降 BITCLK 边沿后的 BITCLK16 时钟的数量															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
00h	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
01h	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
02h	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
03h	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1
04h	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1
05h	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1
06h	0	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1
07h	0	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1
08h	0	1	1	1	1	0	0	0	0	0	0	0	1	1	1	1
09h	0	1	1	1	1	1	0	0	0	0	0	0	1	1	1	1
0Ah	0	1	1	1	1	1	0	0	0	0	0	1	1	1	1	1
0Bh	0	1	1	1	1	1	1	0	0	0	0	1	1	1	1	1
0Ch	0	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1
0Dh	0	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1
0Eh	0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
0Fh	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

15.3.10 设置一个波特率

对于一个给定的 BRCLK 时钟源，使用的波特率决定了所需的除法因子 N：

$$N = \frac{f_{BRCLK}}{\text{Baud rate}}$$

除发因子 N 通常不是一个整数值，因此至少需要一个除法器和一个调制器阶段尽可能的接近该因子。

如果 N 等于或大于 16，可以通过设置 UCOS16 来选择过采样波特率产生模式。

15.3.10.1 低频波特率模式的设置

在低频模式下，除数的整数部分是由预分频器实现的：

UCBRx=INT(N)

同时小数部分由调制器通过下面的公式实现：

UCBRSx=round((N-INT(N))×8)

对于任何给定的位，通过计算增加或减少 UCBRSX 设定可能会降低最大位误差。 为了决定是否是这种情况，在每一次 UCBRSx 的设定中必须对误差进行一个详细的计算。

15.3.10.2 过采样波特率模式的设置

在过采样模式中，计数器被设置为：

$$UCBRx = INT(\frac{N}{16})$$

且第一阶段的调制器被设置为：

$$UCBRFx = round\left(\left(\frac{N}{16} - INT(\frac{N}{16})\right) \times 16\right)$$

当要求更高的精度时，UCBRSx 调节器也可以通过从 0 至 7 调节实现。为了对任何给定位的在最坏情况下的最大误差进行查明，须通过初始化 UCBRFx 设置和 UCBRFx 的设置递增和递减一，对 USCBSX 所有设置的 0 至 7 位的误差进行一个详细的计算。

15.3.11 发送位的时序

每个字符的时序是单个位时序的总和。 使用波特率发生器的调节特性可以减少累计的位误差。 可以通过以下的步骤计算出单个位误差。

15.3.11.1 低频波特率模式位时序

在低频率模式中，计算位 i $T_{位的长度，TX}[i]$是基于 UCBRx 和 UCBRSx 的设置的。

$$T_{bit,TX}[i] = \frac{1}{f_{BRCLK}} (UCBRx + m_{UCBRSx}[i])$$

其中，

$m_{UCBRSx}[i]$=表 15-2中位 i 的调制

15.3.11.2 过采样波特率模式位时序

在过采样波特率模式中，计算位 i $T_{位的长度，TX}[i]$ 是基于 波特率发生器 UCBRx，UCBRFx 和 UCBRSx 的设置的。

$$T_{bit,TX}[i] = \frac{1}{f_{BRCLK}} \left((16 + m_{UCBRSx}[i]) \times UCBRx + \sum_{j=0}^{15} m_{UCBRFx}[j] \right)$$

其中，

$\sum_{j=0}^{15} m_{UCBRFx}[j]$ =表 15-3中相应行的总和

$m_{UCBRSx}[i]$=表 15-2中位 i 的调制

这导致了一个结束位时间 $t_{位，TX}[i]$等于所有以前的和当前位的时间：

$$t_{bit,TX}[i] = \sum_{j=0}^{i} T_{bit,TX}[j]$$

为了计算位误差，时间和理想位时间 $t_{位，理想，TX}[i]$：

$$t_{bit,ideal,TX}[i] = \frac{1}{\text{Baud rate}} (i + 1)$$

这就导致了一个误差被标准化成一个理想的位时间（1/波特率）：

误差$_{TX}[i]$= ($t_{bit,TX}[i]$–$t_{位，理想，TX}[i]$)× 波特率 × 100%

15.3.12 接收位时序

接收时序误差包括两个误差源。 第一个是位到位的时序误差，与发送位时序误差相似。 第二个是介于一个正出现的上升沿和被 UCSI 模块接受了的上升沿之间的误差图 15-11中所示的是介于 UCAxRXD 脚上的数据和内部波特率时钟之间的异步时钟误差。 这导致一个另外的异步误差。 异步误差 $t_{异步}$介于 -0.5BRCLKS 和 +0.5BRCLKS 之间。该误差取决于所选择的波特率发生模式。

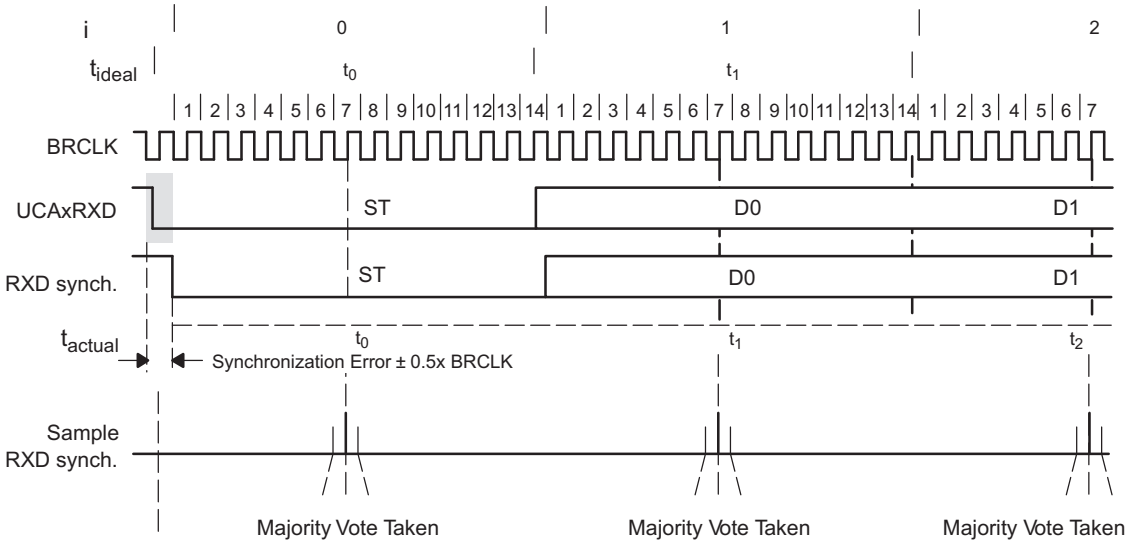


图 **15-11.** 接收错误

理想的采样时间是在一个位周期的中间：

$$t_{bit,ideal,RX}[i] = \frac{1}{\text{Baud rate}} (i + 0.5)$$

真实采样时间等于先前所有位的总和，根据在发送时序部分给出的公式，加上当前位 i 的 1/2 个 BITCLK，加上异步误差 $t_{异步}$。

对于低频波特率模式结果如下：

$$t_{bit,RX}[i] = t_{SYNC} + \sum_{j=0}^{i-1} T_{bit,RX}[j] + \frac{1}{f_{BRCLK}} \left(\text{INT}\left(\frac{1}{2}\text{UCBRx}\right) + m_{UCBRSx}[i] \right)$$

其中，

$$T_{bit,RX}[i] = \frac{1}{f_{BRCLK}} (\text{UCBRx} + m_{UCBRSx}[i])$$

$m_{UCBRSx}[i]$=表 15-2中位 i 的调制

对于过采样波特率模式，位 i 采样时间由以下公式计算：

$$t_{bit,RX}[i] = t_{SYNC} + \sum_{j=0}^{i-1} T_{bit,RX}[j] + \frac{1}{f_{BRCLK}} \left((8 + m_{UCBRSx}[i]) \times \text{UCBRx} + \sum_{j=0}^{7+m_{UCBRSx}[i]} m_{UCBRFx}[j] \right)$$

其中，

$$T_{bit,RX}[i] = \frac{1}{f_{BRCLK}} \left((16 + m_{UCBRSx}[i]) \times \text{UCBRx} + \sum_{j=0}^{15} m_{UCBRFx}[j] \right)$$

$\sum_{j=0}^{7+m_{UCBRSx}[i]} m_{UCBRFx}[j]$ =列 0 的总和 -表 15-3中的相应行的总和

$m_{UCBRSx}[i]$=表 15-2中位 i 的调制

根据以下公式，这导致了一个误差被标准化为一个理想位时间（1/波特率）：

误差$_{RX}[i]$= ($t_{bit,RX}[i]$− $t_{位，理想，RX}[i]$) × 波特率 × 100%

15.3.13 典型的波特率和错误

对于一个使用 32768HZ 的晶振源 ACLK 和典型的 SMCLK 频率，在表 15-4和表 15-5中列出了 UCBRSx 和 UCBRFx 的标准波特率数据。 要确保所选 BRCLK 频率不超过特定器件的最大 USCI 输入频率（请参阅《特定器件的数据手册》）。

相对于在每位中间的理想扫描时间，接收错误是累积时间错误。 为一个带奇偶校验的 8 位字符的接收和一个包括同步误差的停止位给出定了最糟糕的错误。

相对于理想位周期时间，发送错误是时间累积错误 为一个带奇偶校验的 8 位字符的发送和一个停止位给出定了最严重的错误。

表 15-4. 常用波特率，设置，和错误，UCOS16= 0

BRCLK 频率[Hz]	波特率[波特]	UCBRx	UCBRSx	UCBRFx	最大 TX 错误[%]		最大 RX 错误[%]	
32,768	1200	27	2	0	-2.8	1.4	-5.9	2.0
32,768	2400	13	6	0	-4.8	6.0	-9.7	8.3
32,768	4800	6	7	0	-12.1	5.7	-13.4	19.0
32,768	9600	3	3	0	-21.1	15.2	-44.3	21.3
1,048,576	9600	109	2	0	-0.2	0.7	-1.0	0.8
1,048,576	19200	54	5	0	-1.1	1.0	-1.5	2.5
1,048,576	38400	27	2	0	-2.8	1.4	-5.9	2.0
1,048,576	56000	18	6	0	-3.9	1.1	-4.6	5.7
1,048,576	115200	9	1	0	-1.1	10.7	-11.5	11.3
1,048,576	128000	8	1	0	-8.9	7.5	-13.8	14.8
1,048,576	256000	4	1	0	-2.3	25.4	-13.4	38.8
1,000,000	9600	104	1	0	-0.5	0.6	-0.9	1.2
1,000,000	19200	52	0	0	-1.8	0	-2.6	0.9
1,000,000	38400	26	0	0	-1.8	0	-3.6	1.8
1,000,000	56000	17	7	0	-4.8	0.8	-8.0	3.2
1,000,000	115200	8	6	0	-7.8	6.4	-9.7	16.1
1,000,000	128000	7	7	0	-10.4	6.4	-18.0	11.6
1,000,000	256000	3	7	0	-29.6	0	-43.6	5.2
4,000,000	9600	416	6	0	-0.2	0.2	-0.2	0.4
4,000,000	19200	208	3	0	-0.2	0.5	-0.3	0.8
4,000,000	38400	104	1	0	-0.5	0.6	-0.9	1.2
4,000,000	56000	71	4	0	-0.6	1.0	-1.7	1.3
4,000,000	115200	34	6	0	-2.1	0.6	-2.5	3.1
4,000,000	128000	31	2	0	-0.8	1.6	-3.6	2.0
4,000,000	256000	15	5	0	-4.0	3.2	-8.4	5.2
8,000,000	9600	833	2	0	-0.1	0	-0.2	0.1
8,000,000	19200	416	6	0	-0.2	0.2	-0.2	0.4
8,000,000	38400	208	3	0	-0.2	0.5	-0.3	0.8
8,000,000	56000	142	7	0	-0.6	0.1	-0.7	0.8
8,000,000	115200	69	4	0	-0.6	0.8	-1.8	1.1
8,000,000	128000	62	4	0	-0.8	0	-1.2	1.2
8,000,000	256000	31	2	0	-0.8	1.6	-3.6	2.0
12,000,000	9600	1250	0	0	0	0	-0.05	0.05
12,000,000	19200	625	0	0	0	0	-0.2	0
12,000,000	38400	312	4	0	-0.2	0	-0.2	0.2
12,000,000	56000	214	2	0	-0.3	0.2	-0.4	0.5
12,000,000	115200	104	1	0	-0.5	0.6	-0.9	1.2

表 **15-4.** 常用波特率，设置，和错误，**UCOS16= 0 (continued)**

BRCLK 频率[Hz]	波特率[波特]	UCBRx	UCBRSx	UCBRFx	最大 TX 错误[%]		最大 RX 错误[%]	
12,000,000	128000	93	6	0	-0.8	0	-1.5	0.4
12,000,000	256000	46	7	0	-1.9	0	-2.0	2.0
16,000,000	9600	1666	6	0	-0.05	0.05	-0.05	0.1
16,000,000	19200	833	2	0	-0.1	0.05	-0.2	0.1
16,000,000	38400	416	6	0	-0.2	0.2	-0.2	0.4
16,000,000	56000	285	6	0	-0.3	0.1	-0.5	0.2
16,000,000	115200	138	7	0	-0.7	0	-0.8	0.6
16,000,000	128000	125	0	0	0	0	-0.8	0
16,000,000	256000	62	4	0	-0.8	0	-1.2	1.2

表 **15-5.** 常用波特率，设置，和错误，**UCOS16=1**

BRCLK 频率[Hz]	波特率[波特]	UCBRx	UCBRSx	UCBRFx	最大 TX 错误[%]		最大 RX 错误[%]	
1,048,576	9600	6	0	13	-2.3	0	-2.2	0.8
1,048,576	19200	3	1	6	-4.6	3.2	-5.0	4.7
1,000,000	9600	6	0	8	-1.8	0	-2.2	0.4
1,000,000	19200	3	0	4	-1.8	0	-2.6	0.9
1,000,000	57600	1	7	0	-34.4	0	-33.4	0
4,000,000	9600	26	0	1	0	0.9	0	1.1
4,000,000	19200	13	0	0	-1.8	0	-1.9	0.2
4,000,000	38400	6	0	8	-1.8	0	-2.2	0.4
4,000,000	57600	4	5	3	-3.5	3.2	-1.8	6.4
4,000,000	115200	2	3	2	-2.1	4.8	-2.5	7.3
4,000,000	230400	1	7	0	-34.4	0	-33.4	0
8,000,000	9600	52	0	1	-0.4	0	-0.4	0.1
8,000,000	19200	26	0	1	0	0.9	0	1.1
8,000,000	38400	13	0	0	-1.8	0	-1.9	0.2
8,000,000	57600	8	0	11	0	0.88	0	1.6
8,000,000	115200	4	5	3	-3.5	3.2	-1.8	6.4
8,000,000	230400	2	3	2	-2.1	4.8	-2.5	7.3
8,000,000	460800	1	7	0	-34.4	0	-33.4	0
12,000,000	9600	78	0	2	0	0	-0.05	0.05
12,000,000	19200	39	0	1	0	0	0	0.2
12,000,000	38400	19	0	8	-1.8	0	-1.8	0.1
12,000,000	57600	13	0	0	-1.8	0	-1.9	0.2
12,000,000	115200	6	0	8	-1.8	0	-2.2	0.4
12,000,000	230400	3	0	4	-1.8	0	-2.6	0.9
16,000,000	9600	104	0	3	0	0.2	0	0.3
16,000,000	19200	52	0	1	-0.4	0	-0.4	0.1
16,000,000	38400	26	0	1	0	0.9	0	1.1
16,000,000	57600	17	0	6	0	0.9	-0.1	1.0
16,000,000	115200	8	0	11	0	0.9	0	1.6
16,000,000	230400	4	5	3	-3.5	3.2	-1.8	6.4
16,000,000	460800	2	3	2	-2.1	4.8	-2.5	7.3

15.3.14 在低功耗模式下 UART 模式中使用 USCI 模块

低功耗模式下，USCI 模块为 SMCLK 提供时钟自动激活。 因为器件处于一个低功耗模式，当 SMCLK 是 USCI 模块的时钟源，所以是无效的，必要时 USCI 模块将自动激活，而不管时钟源的控制位的设置如何。 直到 USCI 模块返回到它的空闲状态时时钟才会停止保持活动。 在 USCI 模块返回到空闲状态后，时钟源的控制又会依赖于它的控制位的设置。 自动激活模式不适用于 ACLK。

当 USCI 模块激活一个激活的时钟源时，整个器件的时钟源变得活跃且和任何被配置为使用此时钟源的外围可能会受到影响。 例如，在 USCI 模块强制激活 SMCLK 时，使用 SMCLK 的定时器将递增。

15.3.15 USCI 中断

USCI 有一个发送中断向量和接收中断向量。

15.3.15.1 USCI 发送中断操作

UCAxTXIFG 中断标志被发送器置位以便表示 UCAxTXBUF 已准备好接收另一个字符。 如果 UCAxTXIE 和 GIE 也被置位，则将产生一个中断请求。 如果一个字符被写入 UCAxTXBUF，那么 UCAxTXIFG 将自动复位。

在一个 PUC 后或当 UCSWRST=1 时，UCAxTXIFG 被置位。在一个 PUC 后或当 UCSWRST=1 时，UCAxTXIE 被置位。

15.3.15.2 USCI 接收中断操作

每当一个字符被接收并被被载进 UCAxRXBUF 中，UCAxRXIFG 中断标志都会置位。 如果 UCAxRXIE 和 GIE 也被置位，就会产生一个中断请求。 UCAxRXIFG 和 UCAxRXIE 由一个系统复位信号 PUC 复位或当 UCSWRST=1 复位。当 UCAxRXBUF 被读取时 UCAxRXIFG 自动复位。

其它中断控制特征包括：

- 当 UCAxRXEIE=0 时，错误字符将不会置位 UCAxRXIFG。
- 在 UCDORM=1 时，在多处理器模式下非地址字符不会置位 UCAxRXIFG。 在普通 UART 模式，没有字符 置位 UCAxRXIFG。
- 当 UCBRKIE=1 时，一个中断条件将置位 UCBRK 位和 UCAxRXIFG 标志。

15.3.15.3 USCI 的中断使用

USCI_Ax 和 USCI_Bx 共享同一个中断向量。 接收中断标志 UCAxRXIFG 和 UCBxRXIFG 被路由到一个中断向量，发送中断标志 UCAxTXIFG 和 UCBxTXIFG 共享另一个中断向量。

Example 15-1展示了处理数据接收中断的一个中断服务子程序的摘录，这些数据接收中断来自处于 UART 或 SPI 模式的 USCI_A0 和来自 SPI 模式的 USCI_B0。

Example 15-1. 共享中断向量的软件示例，数据接收

```
USCIA0_RX_USCIB0_RX_ISRBIT.B #UCA0RXIFG, &IFG2 ; USCI_A0 Receive Interrupt?JNZ
USCIA0_RX_ISRUSCIB0_RX_ISR?; Read UCB0RXBUF (clears UCB0RXIFG)...RETIUSCIA0_RX_ISR; Read UCA0RXBUF
(clears UCA0RXIFG)...RETI
```

Example 15-2展示了处理数据发送中断的一个中断服务子程序的摘录，这些数据接收中断来自处于 UART 或 SPI 模式的 USCI_A0 和来自 SPI 模式的 USCI_B0。

Example 15-2. *共享中断向量的软件示例，数据发送*

```
USCIA0_TX_USCIB0_TX_ISRBIT.B #UCA0TXIFG, &IFG2 ; USCI_A0 Transmit Interrupt?JNZ
USCIA0_TX_ISRUSCIB0_TX_ISR; Write UCB0TXBUF (clears UCB0TXIFG)...RETIUSCIA0_TX_ISR; Write UCA0TXBUF
(clears UCA0TXIFG)...RETI
```

15.4 USCI 寄存器：UART 模式

在表 15-6和表 15-7中列出了 UART 模式下的可用的 USCI 寄存器。

表 **15-6. USCI_A0** 控制和状态寄存器

寄存器	简式	寄存器类型	地址	初始化状态
USCI_A0 控制寄存器 0	UCA0CTL0	读取/写入	060h	用 PUC 复位
USCI_A0 控制寄存器 1	UCA0CTL1	读取/写入	061h	001h 与 PUC
USCI_A0 波特率控制寄存器 0	UCA0BR0	读取/写入	062h	用 PUC 复位
USCI_A0 波特率控制寄存器 1	UCA0BR1	读取/写入	063h	用 PUC 复位
USCI_A0 调制控制寄存器	UCA0MCTL	读取/写入	064h	用 PUC 复位
USCI_A0 状态寄存器	UCA0STAT	读取/写入	065h	用 PUC 复位
USCI_A0 接收缓冲寄存器	UCA0RXBUF	读取	066h	用 PUC 复位
USCI_A0 发送缓冲寄存器	UCA0TXBUF	读取/写入	067h	用 PUC 复位
USCI_A0 自动波特率控制寄存器	UCA0ABCTL	读取/写入	05Dh	用 PUC 复位
USCI_A0 IrDA 发送控制寄存器	UCA0IRTCTL	读取/写入	05Eh	用 PUC 复位
USCI_A0 IrDA 接收控制寄存器	UCA0IRRCTL	读取/写入	05Fh	用 PUC 复位
SFR 中断使能寄存器 2	IE2	读取/写入	001h	用 PUC 复位
SFR 中断标志寄存器 2	IFG2	读取/写入	003h	00Ah 与 PUC

注: 修改 **SF R**位

为了避免修改其他模块的控制位，建议使用 BIS.B 或 BIC.B指令，而非 MOV.B 或 CLR.B 指令来置位或清除 IEx 和 IFGx 位。

表 **15-7. USCI_A1** 控制和状态寄存器

寄存器	简式	寄存器类型	地址	初始化状态
USCI_A1 控制寄存器 0	UCA1CTL0	读取/写入	0D0h	用 PUC 复位
USCI_A1 控制寄存器 1	UCA1CTL1	读取/写入	0D1h	001h 与 PUC
USCI_A1 波特率控制寄存器 0	UCA1BR0	读取/写入	0D2h	用 PUC 复位
USCI_A1 波特率控制寄存器 1	UCA1BR1	读取/写入	0D3h	用 PUC 复位
USCI_A1 调制控制寄存器	UCA1MCTL	读取/写入	0D4h	用 PUC 复位
USCI_A1 状态寄存器	UCA1STAT	读取/写入	0D5h	用 PUC 复位
USCI_A1 接收缓冲寄存器	UCA1RXBUF	读取	0D6h	用 PUC 复位
USCI_A1 发送缓冲寄存器	UCA1TXBUF	读取/写入	0D7h	用 PUC 复位
USCI_A1 自动波特率控制寄存器	UCA1ABCTL	读取/写入	0CDh	用 PUC 复位
USCI_A1 IrDA 发送控制寄存器	UCA1IRTCTL	读取/写入	0CEh	用 PUC 复位
USCI_A1 IrDA 接收控制寄存器	UCA1IRRCTL	读取/写入	0CFh	用 PUC 复位
USCI_A1/B1 中断使能寄存器	UC1IE	读取/写入	006h	用 PUC 复位
USCI_A1/B1 中断标志寄存器	UC1IFG	读取/写入	007h	00Ah 与 PUC

15.4.1 UCAxCTL0，USCI_Ax 控制寄存器 0

7	6	5	4	3	2	1	0
UCPEN	**UCPAR**	**UCMSB**	**UC7BIT**	**UCSPB**	**UCMODEx**		**UCSYNC**
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

UCPEN 位 7 奇偶校验使能

0 奇偶校验被禁止。

1 奇偶校验被启用。 产生的 (UCAxTXD) 和预期 (UCAxRXD) 的奇偶校验位。 在地址位多处理器模式中，地址位被包括在奇偶校验计算中。

UCPAR 位 6 奇偶校验选择。 奇偶校验被禁用时，UCPAR 不能使用。

0 奇数校验

1 偶数校验

UCMSB 位 5 MSB 首先选择。 控制移位寄存器接收和发送的方向。

0 LSB 首先

1 MSB 首先

UC7BIT 位 4 字符长度。 选择 7 位或 8 位长度字符。

0 8 位数据

1 7 位数据

UCSPB 位 3 停止位选择。 停止位的个数。

0 1 个停止位

1 2 个停止位

UCMODEx 位 2-1 USCI 模式。 当 UCSYNC=0 时，UCMODEx 位选择异步模式。

00 UART 模式

01 空闲线多处理器模式

10 地址位多处理器模式

11 带有自动波特率检测的 UART 模式

UCSYNC 位 0 同步模式使能

0 异步模式

1 同步模式

15.4.2 UCAxCTL1，USCI_Ax 控制寄存器 1

7	6	5	4	3	2	1	0
UCSSELx		UCRXEIE	UCBRKIE	UCDORM	UCTXADDR	UCTXBRK	UCSWRST
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1

UCSSELx 位 7-6 USCI 时钟源选择。 这些位选择 BRCLK 时钟源。
- 00 UCLK
- 01 ACLK
- 10 SMCLK
- 11 SMCLK

UCRXEIE 位 5 接收错误的字符中断使能
- 0 拒绝的错误字符和 UCAxRXIFG 没有置位
- 1 接收到的错误字符将置位 UCAxRXIFG

UCBRKIE 位 4 接收中断字符中断使能
- 0 接收的中断字符不会置位 UCAxRXIFG。
- 1 接收的中断字符置位 UCAxRXIFG。

UCDORM 位 3 休眠。 使 USCI 进入休眠模式
- 0 没有处于休眠状态。 所有接收的字符都将置位 UCAxRXIFG。
- 1 休眠。 只有被空闲线或地址位设置在前面的字符才会置位 UCAxRXIFG。 在带有自动波特率检测的 UART 模式中，只有一个中断和同步字段的组合才会置位 UCAxRXIFG。

UCTXADDR 位 2 发送地址 发送的下一帧将会被被标记为取决于选择的多处理器模式的地址。
- 0 发送的下一帧是数据
- 1 发送的下一帧是地址

UCTXBRK 位 1 发送中断。 通过下一次写入发送缓冲器发送一个中断。 在带有自动波特率检测的 UART 模式中，必须将 055h 写入 UCAxTXBUF 以此来产生所需的中断/同步字段。 否则，必须将 0h 写入发送缓冲器。
- 0 发送的下一帧不是一个中断
- 1 发送的下一帧是一个中断或一个中断/同步

UCSWRST 位 0 软件复位使能
- 0 被禁用。 USCI 复位被释放用于运行。
- 1 被启用。 USCI 逻辑保持在复位状态。

15.4.3 UCAxBR0，USCI_Ax 波特率控制寄存器 0

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

15.4.4 UCAxBR1，USCI_Ax 波特率控制寄存器 1

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

UCBRx 7-0 波特率发生器的时钟预分频器设置。 (UCAxBR0 + UCAxBR1 × 256) 的 16 位值组成了分频值。

15.4.5 UCAxMCTL，USCI_Ax 调制控制寄存器

7	6	5	4	3	2	1	0
UCBRFx				UCBRSx			UCOS16
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

UCBRFx 位 7-4 第一调制阶段选择。 当 UCOS16=1 时，这些位决定了 BITCLK16 的调制模式。在 UCOS16=0 时忽略。表 15-3显示了调制模式

UCBRSx 位 3-1 第二调制阶段选择。 这些位决定了 BITCLK 的调制模式。表 15-2显示了调制模式。

UCOS16 位 0 过采样模式被启用

0 被禁用

1 被启用

15.4.6 UCAxSTAT，USCI_Ax 状态寄存器

7	6	5	4	3	2	1	0
UCLISTEN	UCFE	UCOE	UCPE	UCBRK	UCRXERR	UCADDR UCIDLE	UCBUSY
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	r-0

UCLISTEN 位 7 监听使能。 UCLISTEN 位选择回路模式。

0 被禁用

1 被启用。 UCAxTXD 被内部反馈到接收器。

UCFE 位 6 组帧错误标志

0 无错误

1 接收到的具有低停止位的字符

UCOE 位 5 溢出错误标志。 当在之前的一个字符被读取前随后一个字符被传输到 UCAxRXBUF 时，该位被置位。 当 UCxRXBUF 被读取时，UCOE 被自动清除，而且 UCOE 绝不能用软件清除。 否则，它将无法正常工作。

0 无错误

1 发生溢出错误

UCPE 位 4 奇偶校验错误标志。 当 UCPEN=0 时, UCPE 被读取为 0。

0 无错误

1 接收到的具有奇偶校验错误的字符

UCBRK 位 3 中断检测标志

0 无中断条件

1 中断条件发生

UCRXERR 位 2 接收错误标志。 该位表示收到一个错误字符。 当 UCRXERR=1 时, 一个或多个错误标志位 (UCFE, UCPE, UCOE) 也被置位。 当 UCAxRXBUF 被读取时，UCRXERR 被清除。

0 没有检测到接收错误

1 检测到接收错误

UCADDR 位 1 在地址位多处理器模式中接收到的地址

0 接收到的字符为数据

1 接收到的字符是一个地址

UCIDLE 在空闲线多处理器模式中检测到的空闲线路。

0 没有检测到空闲线路

1 检测到空闲线路

UCBUSY 位 0 USCI 忙。 该位表示是否有一个发送或接收操作正在进行。

0 USCI 未激活

1 USCI 发送或接收

15.4.7 UCAxRXBUF，USCI_Ax 接收缓冲寄存器

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

UCRXBUFx	位 7-0	接收数据缓冲区是用户可以访问的并且包含最后从接收移位寄存器中接收到的字符。 读取 UCAxRXBUF 复位接收错误位，UCADDR 或 UCIDLE 位，和 UCAxRXIFG。 在 7 位数据模式下，UCAxRXBUF 是 LSB 对齐的并且 MSB 总是复位。

15.4.8 UCAxTXBUF，USCI_Ax 发送缓冲寄存器

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

UCTXBUFx	位 7-0	发送数据缓冲区是用户可以访问的并且保存有等待被转移到发送移位寄存器和 UCAxTXD 上传输的数据。 写入到发送数据缓冲器清除 UCAxTXIFG。 UCAxTXBUF 的 MSB 不用于 7 位数据且被复位。

15.4.9 UCAxIRTCTL，USCI_Ax IrDA 发送缓冲寄存器

7	6	5	4	3	2	1	0
UCIRTXPLx						UCIRTXCLK	UCIREN
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

UCIRTXPLx	位 7-2	发送脉冲长度。 脉冲长度 $t_{脉冲}$=(UCIRTXPLx + 1)/ (2 × $f_{IRTXCLK}$)
UCIRTXCLK	位 1	IrDA 的发送脉冲时钟选择 0 BRCLK 1 当 UCOS16=1 时，为 BITCLK16。否则为 BRCLK
UCIREN	位 0	IrDA 编码器/解码器使能。 0 IrDA 编码器/解码器被禁用 1 IrDA 编码器/解码器被启用

15.4.10 UCAxIRRCTL，USCI_Ax IrDA 接收控制寄存器

7	6	5	4	3	2	1	0
UCIRRXFLx						UCIRRXPL	UCIRRXFE
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

UCIRRXFLx	位 7-2	接收过滤器长度。 对于接收的最短脉冲长度由：t_{MIN}= (UCIRRXFLx + 4) / (2 × $f_{IRTXCLK}$)给出
UCIRRXPL	位 1	IrDA 接收输入 UCAxRXD 极性 0 当一个轻脉冲出现时，IrDA 收发器传递了一个高脉冲 1 当一个轻脉冲出现时，IrDA 收发器传递了一个低脉冲
UCIRRXFE	位 0	IrDA 接收滤波器被启用 0 接收滤波器被禁用 1 接收滤波器被启用

15.4.11 UCAxABCTL，USCI_Ax 自动波特率控制寄存器

7	6	5	4	3	2	1	0
被保留		UCDELIMx		UCSTOE	UCBTOE	被保留	UCABDEN
r-0	r-0	rw-0	rw-0	rw-0	rw-0	r-0	rw-0

被保留	位 7-6	被保留
UCDELIMx	位 5-4	中断/同步定界符长度
		00 1 位时间
		01 2 位时间
		10 3 位时间
		11 4 位时间
UCSTOE	位 3	同步字段超时错误
		0 无错误
		1 超出可测量时间的同步字段长度。
UCBTOE	位 2	中断超时错误
		0 无错误
		1 超过 22 位时间的中断字段长度。
被保留	位 1	被保留
UCABDEN	位 0	自动波特率检测使能
		0 波特率检测被禁用。 中断和同步字段长度没有被测量。
		1 波特率检测被禁用。 中断和同步字段的长度被测量并且波特率设置也相应的改变。

15.4.12 IE2，中断使能寄存器 2

7	6	5	4	3	2	1	0
						UCA0TXIE	UCA0RXIE
						rw-0	rw-0

	位 7-2	这些位可用于其他模块（请参阅《器件专用数据表》）
UCA0TXIE	位 1	USCI_A0 发送中断启用
		0 中断被禁用
		1 中断被启用
UCA0RXIE	位 0	USCI_A0 接收中断启用
		0 中断被禁用
		1 中断被启用

15.4.13 IFG2，中断标志寄存器 2

7	6	5	4	3	2	1	0
						UCA0TXIFG	UCA0RXIFG
						rw-1	rw-0

	位 7-2	这些位可用于其他模块（请参阅《器件专用数据表》）
UCA0TXIFG	位 1	USCI_A0 发送中断标志。 UCA0TXBUF 为空时，UCA0TXIFG 被置位。
		0 无中断等待
		1 中断等待
UCA0RXIFG	位 0	USCI_A0 接收中断标志。 当 UCA0RXBUF 已经接收一个完整字符时，UCA0RXIFG 被置位。
		0 无中断等待
		1 中断等待

15.4.14 UC1IE，USCI_A1 中断使能寄存器

7	6	5	4	3	2	1	0
未被使用						**UCA1TXIE**	**UCA1RXIE**
rw-0	rw-0	rw-0	rw-0			rw-0	rw-0

未被使用	位 7-4	未使用
	位 3-2	这些位可用于其他 USCI 模块（请参阅《器件专用数据表》）
UCA1TXIE	位 1	USCI_A1 发送中断启用
		0 中断被禁用
		1 中断被启用
UCA1RXIE	位 0	USCI_A1 接收中断启用
		0 中断被禁用
		1 中断被启用

15.4.15 UC1IFG，USCI_A1 中断标志寄存器

7	6	5	4	3	2	1	0
未被使用						**UCA1TXIFG**	**UCA1RXIFG**
rw-0	rw-0	rw-0	rw-0			rw-1	rw-0

未被使用	位 7-4	未使用
	位 3-2	这些位可用于其他 USCI 模块（请参阅《器件专用数据表》）
UCA1TXIFG	位 1	USCI_A1 发送中断标志。 UCA1TXBUF 为空时，UCA1TXIFG 被置位。
		0 无中断等待
		1 中断等待
UCA1RXIFG	位 0	USCI_A1 接收中断标志。 当 UCA1RXBUF 已经接收一个完整字符时，UCA1RXIFG 被置位。
		0 无中断等待
		1 中断等待

通用串行通信接口，SPI 模式。

通用串行通信接口 (USCI) 采用一个硬件模块支持多路串行通信模式。 本章讨论了同步外设接口或 SPI 模式的操作。

Topic **Page**

16.1 USCI 概述

通用串行通信接口 (USCI) 支持多种串行通信模式。 不同的串行通信接口模块支持不同的模式。 每个不同的串行通信接口模块用不同的字母命名。（例如，USCI_A 和 USCI_B 是不同的）。 如果在一个器件里有多于一个的相同的 USCI 模块，这些模块以递增的数字命名。 例如，如果一个器件有两个 USCI_A 模块，它们被命名为 USCI_A0 和 USCI_A1。 请参阅《器件专用数据表》以确定在哪个器件里面应用了什么 USCI 模块。

USCI_Ax 模块支持：

- UART 模式
- 用于 IrDA 通信的整形脉冲
- 用于 LIN 通信的自动波特率检测
- SPI 模式

USCI_Bx 模块支持：

- I^2C 模式
- SPI 模式

16.2 USCI 介绍：SPI 模式

在同步模式中，通用串行通信接口通过三格或四个引脚把 MSP430 连接到一个外部系统：UCxSIMO，UCxSOMI，UCxCLK，和 UCxSTE。 当 UCSYNC 位被置位时，选用 SPI 模式并且用 UCMODEx 位选择 SPI 模式(3 引脚或 4 引脚)。

SPI 模式的特性包括：

- 数据长度为 7 或 8 位
- 最低有效位或最高有效位数据最先传送和接收
- 3 引脚和 4 引脚 SPI 操作
- 主控模式或受控模式
- 独立的发送和接收移位寄存器
- 独立的发送和接收缓存寄存器
- 连续发送和接收操作
- 可选的时钟极性和相位控制
- 主控模式下的可编程时钟频率
- 独立的接收中断和发送中断功能
- LPM4 模式下的从器件操作

图 16-1显示了 SPI 模式下的 USCI 的配置。

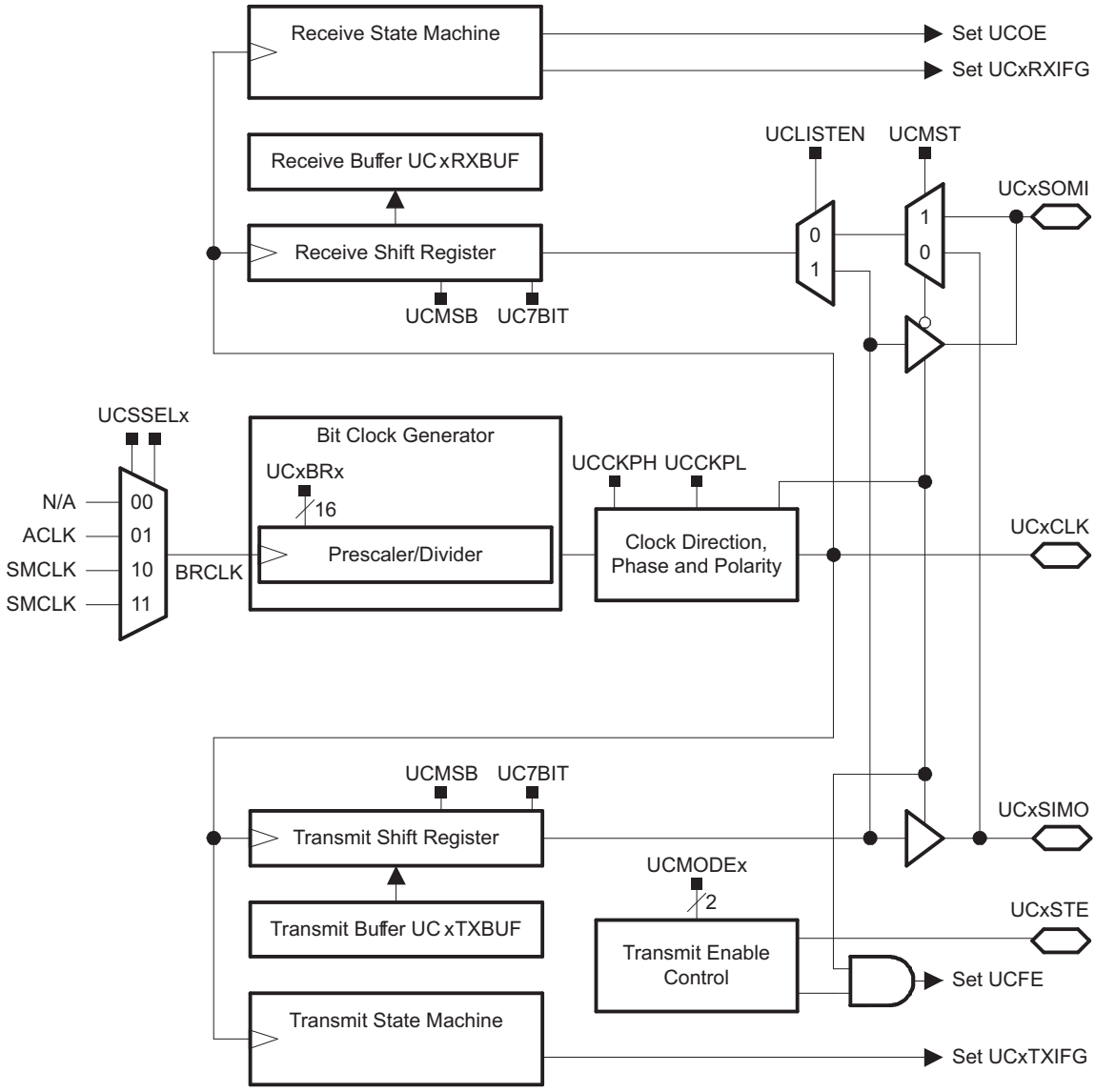


图 **16-1. USCI** 方框图**: SPI** 模式

16.3 USCI 操作: SPI 模式

在 SPI 模式中，串行数据可在多路个器件之间进行发送和接收，这些器件使用由主机提供的共用时钟。 一个额外引脚，UCxSTE，用于使能器件接收和发送数据并且由主器件控制。

三或四引脚信号被用于 SPI 数据交换：

- UCxSIMO:从器件输入，主器件输出
  - 主控模式：UCxSIMO 为数据的输出线。
  - 受控模式：UCxSIMO 为数据的输入线。
- UCxSIM1:从器件输出，主器件输入
  - 主控模式：UCxSOMI 为数据的输入线。
  - 受控模式：UCxSOMI 为数据的输出线。
- UCxCLK: USCI SPI 时钟
  - 主控模式：UCxCLK 是一个输出。
  - 受控模式：UCxCLK 是一个输入。
- UCxSTE：从器件发送使能
  被用于 4 引脚模式中以此来允许一个单总线上的多个主器件。 不用于 3 引脚模式。表 16-1描述了 UCxSTE 的操作。

表 16-1. UCxSTE 的操作

UCMODEx	UCxSTE 激活状态	UCxSTE	从器件	主器件
01	高	0	未激活的	激活的
		1	激活的	未激活的
10	低	0	激活	未激活的
		1	未激活的	激活

16.3.1 USCI 的初始化和复位

通用串行通信接口通过一个 PUC 或 UCSWRST 位来复位。 一个 PUC 后，UCSWRST 位会自动置位，以此来保持 USCI 在复位状态。 当置位时，UCSWRST 位复位 UCxRXIE，UCxTXIE，UCxRXIFG，UCOE 和 UCFE 位并置位 UCxTXIFG 标志。 清零 UCSWRST 会释放 USCI 从而使其运行。

注: 初始化或重新配置 USCI 模块

建议的 USCI 初始化/重新配置的过程为：

1. 置位 UCSWRST (BIS.B #UCSWRST，&UCxCTL1)
2. 使用 UCSWRST=1，初始化所有的通用串行通信接口寄存器（包括 UCxCTL1）
3. 配置端口
4. 通过软件清零 UCSWRST 位 (BIC.B #UCSWRST，&UCxCTL1)
5. 通过 UCxRXIE 和/或 UCxTXIE 使能中断（可选）

16.3.2 字符格式

SPI 模式下的 USCI 模块支持通过 UC7BIT 位选择的 7 和 8 位字符长度。 在 7 位数据模式下，UCxRXBUF 是对齐的 LSB 并且 MSB 总是复位。 UCMSB 位控制传输方向并选定最低有效位或最高有效位先发送或接收。

注: 缺省字符格式

缺省 SPI 字符传输是从 LSB 开始。 对于带有其他 SPI 接口的通信，它有可能需要 MSB 优先模式。

注: 字符格式图表

本章的所有图表均使用 MSB 优先的格式。

16.3.3 主控模式

图 16-2说明了 USCI 在 3 引脚和 4 引脚模式下作为主器件时的配置。 当数据被送到传输数据缓冲器 UCxTXBUF 时，USCI 开始数据传送。 当 TX 移位寄存器空了后，UCxTXBUF 缓冲区的数据被传送到其中，在 UCxSIMO 上传送初始化数据，起始位是最高位还是最低位，决定于 UCMSB 标志位的设置。 UCxSOMI 上的数据在反向时钟沿上被移入接收移位寄存器。 当接收到字符之后，接收数据从 RX 移位寄存器送入接收数据缓冲器 UCxRXBUF，并且置位接收中断标志 UCxRXIFG，表示接收/发送操作完成。

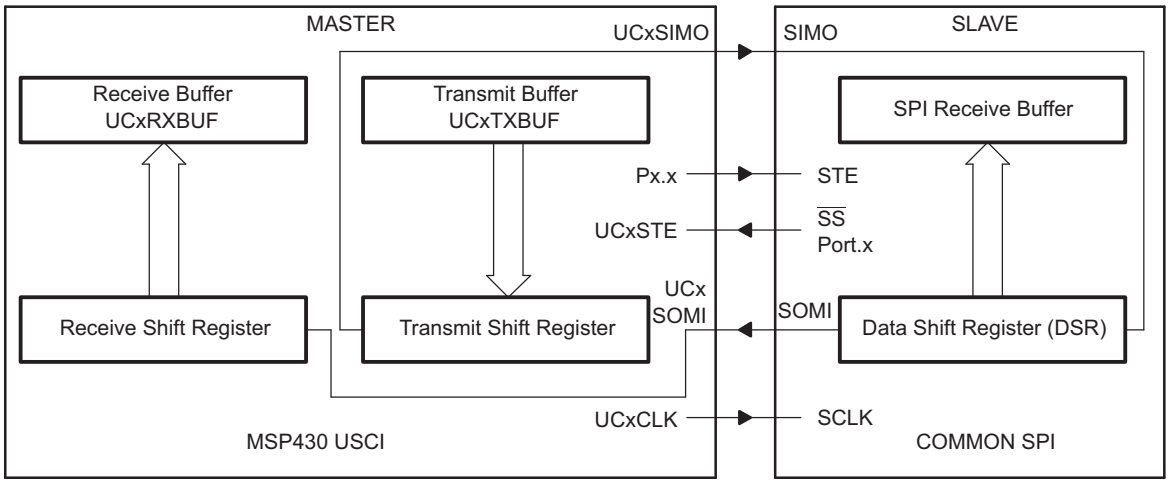


图 16-2. USCI 主控模式和外部受控模式

一个传输中断标志 UCxTXIFG 被置位，表示数据已经从 UCxTXBUF 移动到 TX 移位寄存器中并且 UCxTXBUF 准备传输新数据。 它并不代表 RX/TX 的完成。

为了在主控模式下接收 USCI 数据，数据必须事先写入 UCxTXBUF，因为接收和发送操作不是马上进行的。

16.3.3.1 4 引脚 SPI 主控模式

如在表 16-1中描述，在 4 引脚主控模式中，UCxSTE 被用来防止与其它主器件相冲突和控制主器件。 当 UCxSTE 处于主器件不活动状态时：

- UCxSIMO 和 UCxCLK 被设置为输入并且不再驱动总线。
- 出错位 UCFE 置位表明一个要由客户处理的通讯完整性操作的违规。
- 内部状态被复位并且移位操作取消。

如果数据写入 UCxTXBUF 而主器件通过 UCxSTE 位保持非激活状态，它将在 UCxSTE 转换为主器件激活状态时被立即发送。 如果一个激活的发送被正在转换为主器件未激活状态的 UCxSTE 中断，那么当 UCxSTE 转换为主器件激活状态时数据需要将数据重新写入 UCxTXBUF。 UCxSTE 输入信号不会在 3 引脚主空模式中使用。

16.3.4 受控模式

图 16-3显示了在 3 引脚和 4 引脚配置下的作为从器件的 USCI 。 UCxCLK 被用作 SPI 时钟输入而且它必须由外部主器件提供。 数据传送率取决于这个时钟而不是内部位时钟发生器。 在 UCxSOM1 上，UCxCLK 开始传输之前，数据被写入 UCxTXBUF 并被移动到 TX 移位寄存器。 当设定编号位数被接收到后，UCxSIMO 上的数据被移入 UCxCLK 反向沿上的移位寄存器并被移动到 UCxRXBUF。 当数据由 RX 移位寄存器转被送到 UCxRXBUF 时，UCxRXIFG 中断标志被置位，表明数据已被接收。 在新数据写入 UCxRXBUF 时，如果前一个接收的数据还未从 UCxRXBUF 中被读取，则溢出错误位，UCOE 被置位。

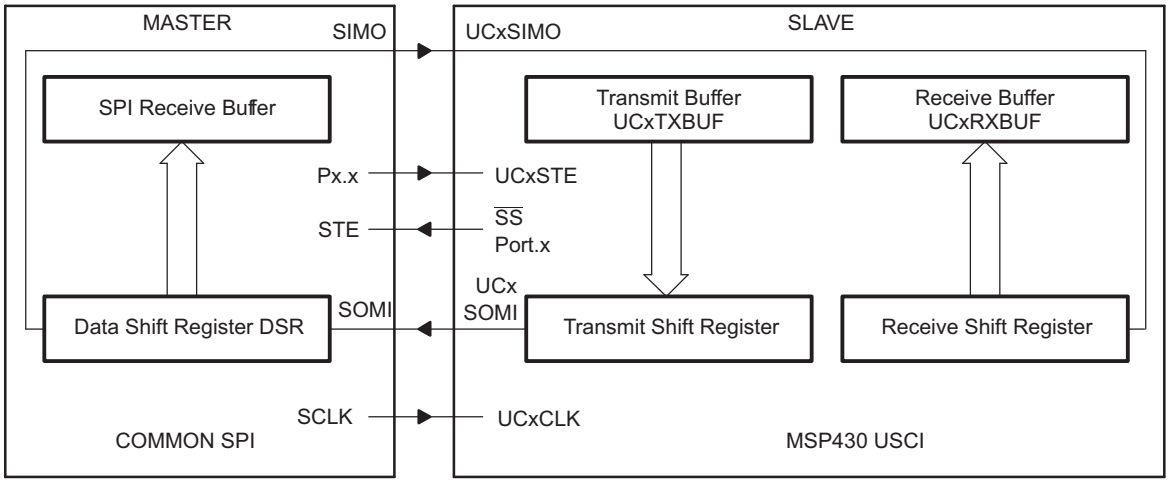


图 16-3. USCI 从器件和外部主器件

16.3.4.1 4 引脚 SPI 受控模式

在 4 引脚受控模式中，UCxSTE 被从器件用于使能发送和接收操作并由 SPI 主器件提供。 当 UCxSTE 处于从器件未激活状态时，从器件正常运行。 当 UCxSTE 处于从器件未激活状态时：

- UCxSIMO 上的所有进行中的操作都被暂停。
- UCxSIMO 被设置为输入方向
- 切换操作被暂停直到 UCxSIMO 线传输进入从器件传输状态。

UCxSTE 输入信号的不会在 3 引脚受控模式中使用。

16.3.5 SPI 使能

通过清零 UCxSIMO 位使能 USCI 模块时，该模块准备接收和发送数据。 在主控模式中，位时钟发生器准备，但既不计时也不产生任何时钟。 在受控模式中，位时钟发生器被禁用并且由主器件提供。

UCBUSY=1 标志着一个发送或接收操作。

一个 PUC 或置位 UCSWRST 位立即禁用 USCI 并且所有激活的传输都被终止。

16.3.5.1 发送使能

在主控模式中，写入 UCxTXBUF 将激活位时钟发生器并且数据开始发送。

在受控模式中，当一个主器件提供一个时钟时且，在 4 引脚模式中，UCxSTE 处于从器件激活状态中时，数据开始传输。

16.3.5.2 接收使能

当一个传输激活时，SPI 接收数据。 接收和发送操作不同时运行。

16.3.6 穿行时钟控制

在 SPI 总线上，主器件提供 UCxCLK。 当 UCMST=1 时，USCI 位时钟发生器通过 UCxCLK 引脚提供位时钟。 被用于产生位时钟的时钟由 UCSSELx 位进行选择。 当 UCMST=0 时，USCI 时钟由主器件通过 UCxCLK 引脚提供，不使用位时钟发生器，并且 UCSSELx 位无影响。 SPI 接收器和发生器并行操作并且在数据传输时使用同一时钟源。

在比特率控制寄存器 UCxxBR1 和 UCxxBR0 中 UCBRx 的 16 位值就是 USCI 时钟源，BRCLK 的分频系数。 在主控模式中能产生的最大位时钟是 BRCLK。 在 SPI 模式中不用调制并且当 USCI_A 应用在 SPI 模式时，UCAxMCTL 应该被清零。 UCAxCLK/UCBxCLK 频率由下列给出：

$$f_{BitClock} = \frac{f_{BRCLK}}{UCBRx}$$

16.3.6.1 串行时钟的极性和相位

UCxCLK 的相位和极性通过 USCI 的 UCCKPL 和 UCCKPH 控制位独立配置。 图 16-4给吃了每种情况的时序。

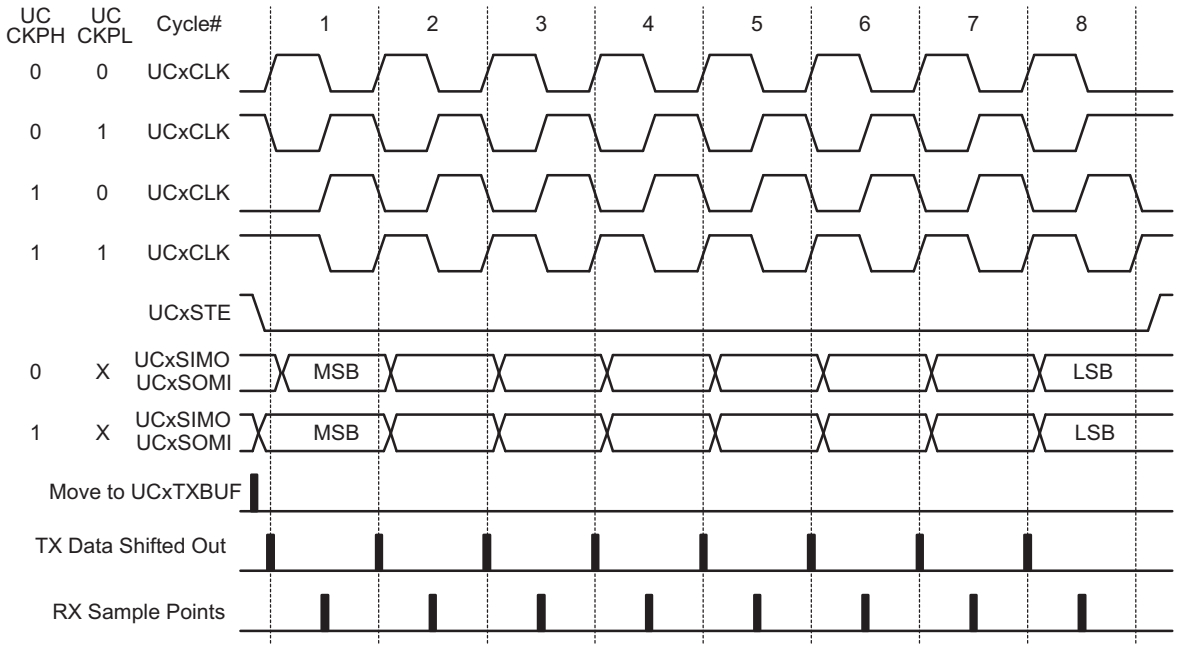


图 16-4. UCMSB=1 时的 USCI SPI 时序

16.3.7 使用 SPI 低功耗模式

对于应用在低功耗模式中的 SMCLK，USCI 模块提供时钟自动激活功能。 当 SMCLK 为 USCI 的时钟源时，它是未激活的，因为器件处于一个低功耗模式，当需要时，USCI 模块可以自动激活，而不用管时钟源的控制位状态。 时钟保持激活直到 USCI 模块返回到空闲状态。 USCI 模块返回到空闲状态后，时钟源的控制恢复到它控制位的设置。 不为 ACLK 提供自动时钟唤醒功能。

当 USCI 模块激活一个未激活时钟源时，时钟源激活，则应用该时钟源的整个器件和所有外设配置都会受到影响。 例如，在 USCI 模块强制 SMCLK 激活期间，将会增加一个使用 SMCLK 的定时器。

在 SPI 受控模式中，无需内部时钟源，因为所需时钟都是由外部主器件提供。 当器件是在 LPM4 中且所有时钟源都被禁用期间，有可能在 SPI 受控模式中运行 USCI。 接收或发送中断可以将 CPU 从任何低功耗模式中唤醒。

16.3.8 SPI 中断

USCI 有一个发送中断向量和接收中断向量。

16.3.8.1 SPI 发送中断操作

UCxTXIFG 中断标志被发送器置位来表示 UCxTXBUF 准备接收另一个字符。 如果 UCxTXIE 和 GIE 也被置位，将产生一个中断请求。 如果一个字符被写入 UCxTXBUF，则 UCxTXIFG 会自动复位。 一个 PUC 或 UCSWRST=1 后，UCxTXIFG 将置位。一个 PUC 或 UCSWRST= 1 后，UCxTXIE 将复位。

注: 在 SPI 模式中写入 UCxTXBUF

当 UCxTXIFG =0 时，数据被写入 UCxTXBUF 可能导致错误的数据传输。

16.3.8.2 SPI 接收中断操作

每当一个字符被接收并装载到 UCxRXBUF 中时 UCxRXIFG 中断标志就会被置位一次。 如果 UCxRXIE 和 GIE 也置位，将产生一个中断请求。 复位 PUC 信号或者 UCSWRST=1 时，UCxRXIFG 和 UCxRXIE 将被系统复位。当 UCxRXBUF 被读取时，UCxRXIFG 会自动复位。

16.3.8.3 USCI 中断用法

USCI_Ax 和 USCI_Bx 共用一个相同的中断向量。 接收中断的标志 UCAxRXIFG 和 UCBxRXIFG 被路由到一个中断向量，发送中断标志 UCAxTXIFG 和 UCBxTXIFG 共享另一个中断向量。

Example 16-1说明一个中断服务子程序处理来自 UART 或 SPI 模式下的 USCI_A0 和在 SPI 模式下 USCI_B0 的数据接收中断的一段摘录。

Example 16-1. 共用接受中断向量软件举例

```
USCIA0_RX_USCIB0_RX_ISRBIT.B #UCA0RXIFG, &IFG2 ; USCI_A0 Receive Interrupt?JNZ
USCIA0_RX_ISRUSCIB0_RX_ISR?; Read UCB0RXBUF (clears UCB0RXIFG)...RETIUSCIA0_RX_ISR; Read UCA0RXBUF
(clears UCA0RXIFG)...RETI
```

Example 16-2说明一个中断服务子程序处理来自 UART 或 SPI 模式下的 USCI_A0 和在 SPI 模式下 USCI_B0 的数据发送中断的一段摘录。

Example 16-2. 共用发送中断向量软件举例

```
USCIA0_TX_USCIB0_TX_ISRBIT.B #UCA0TXIFG, &IFG2 ; USCI_A0 Transmit Interrupt?JNZ
USCIA0_TX_ISRUSCIB0_TX_ISR; Write UCB0TXBUF (clears UCB0TXIFG)...RETIUSCIA0_TX_ISR; Write UCA0TXBUF
(clears UCA0TXIFG)...RETI
```

16.4 USCI 寄存器：SPI 模式

表 16-2列出了 SPI 模式下应用 USCI_A0 和 USCI_B0 的 USCI 寄存器。 表 16-3列出了 SPI 模式下应用 USCI_A1 和 USCI_B1 的 USCI 寄存器。

表 16-2. USCI_A0 和 USCI_B0 控制状态寄存器

寄存器	简氏	寄存器类型	地址	初始状态
USCI_A0 控制寄存器 0	UCA0CTL0	读取/写入	060h	用 PUC 复位
USCI_A0 控制寄存器 1	UCA0CTL1	读取/写入	061h	001h 与 PUC
USCI_A0 波特率控制寄存器 0	UCA0BR0	读取/写入	062h	用 PUC 复位
USCI_A0 波特率控制寄存器 1	UCA0BR1	读取/写入	063h	用 PUC 复位
USCI_A0 调制控制寄存器	UCA0MCTL	读取/写入	064h	用 PUC 复位
USCI_A0 状态寄存器	UCA0STAT	读取/写入	065h	用 PUC 复位
USCI_A0 接收缓冲器寄存器	UCA0RXBUF	阅读	066h	用 PUC 复位
USCI_A0 发送缓冲寄存器	UCA0TXBUF	读取/写入	067h	用 PUC 复位
USCI_B0 控制寄存器 0	UCB0CTL0	读取/写入	068h	001h 与 PUC
USCI_B0 控制寄存器 1	UCB0CTL1	读取/写入	069h	001h 与 PUC
USCI_B0 位速率控制寄存器 0	UCB0BR0	读取/写入	06Ah	用 PUC 复位
USCI_B0 比特率控制寄存器 1	UCB0BR1	读取/写入	06Bh	用 PUC 复位
USCI_B0 状态寄存器	UCB0STAT	读取/写入	06Dh	用 PUC 复位
USCI_B0 接收缓冲寄存器	UCB0RXBUF	读取	06Eh	用 PUC 复位
USCI_B0 发送缓冲寄存器	UCB0TXBUF	读取/写入	06Fh	用 PUC 复位
SFR 中断使能寄存器 2	IE2	读取/写入	001h	用 PUC 复位
SFR 中断标志寄存器 2	IFG2	读取/写入	003h	00Ah 与 PUC

注: 修改 SFR 位

为了避免修改其他模块的控制位，建议使用 BIS.B 或 BIC.B 指令置位或清除 IEX 和 IFGx 位，而非 MOV.B 或 CLR.B 指令。

表 16-3. USCI_A1 和 USCI_B1 控制状态寄存器

寄存器	简式	寄存器类型	地址	初态
USCI_A1 控制寄存器 0	UCA1CTL0	读取/写入	0D0h	用 PUC 复位
USCI_A1 控制寄存器 1	UCA1CTL1	读取/写入	0D1h	001h 与 PUC
USCI_A1 波特率控制寄存器 0	UCA1BR0	读取/写入	0D2h	用 PUC 复位
USCI_A1 波特率控制寄存器 1	UCA1BR1	读取/写入	0D3h	用 PUC 复位
USCI_A1 调制控制寄存器	UCA10MCTL	读取/写入	0D4h	用 PUC 复位
USCI_A1 状态寄存器	UCA1STAT	读取/写入	0D5h	用 PUC 复位
USCI_A1 接收缓冲寄存器	UCA1RXBUF	读取	0D6h	用 PUC 复位
USCI_A1 发送缓冲寄存器	UCA1TXBUF	读取/写入	0D7h	用 PUC 复位
USCI_B1 控制寄存器 0	UCB1CTL0	读取/写入	0D8h	001h 与 PUC
USCI_B1 控制寄存器 1	UCB1CTL1	读取/写入	0D9h	001h 与 PUC
USCI_B1 比特率控制寄存器 0	UCB1BR0	读取/写入	0DAh	用 PUC 复位
USCI_B1 比特率控制寄存器 1	UCB1BR1	读取/写入	0DBh	用 PUC 复位
USCI_B1 状态寄存器	UCB1STAT	读取/写入	0DDh	用 PUC 复位
USCI_B1 接收缓冲寄存器	UCB1RXBUF	读取	0DEh	用 PUC 复位
USCI_B1 发送缓冲寄存器	UCB1TXBUF	读取/写入	0DFh	用 PUC 复位
USCI_A1/B1 中断使能寄存器	UC1IE	读取/写入	006h	用 PUC 复位
USCI_A1/B1 中断标志寄存器	UC1IFG	读取/写入	007h	00Ah 与 PUC

16.4.1 UCAxCTL0，USCI_Ax 控制寄存器 0，UCBxCTL0，USCI_Bx 控制寄存器 0

7	6	5	4	3	2	1	0
UCCKPH	**UCCKPL**	**UCMSB**	**UC7BIT**	**UCMST**	**UCMODEx**		**UCSYNC=1**
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	

UCCKPH 位 7 时钟相位选择
- 0 在第一个 UCLK 边沿上变化的数据和在下列边沿上捕获的数据。
- 1 在第一个 UCLK 边沿上捕获的数据和在下列边沿上改变的数据。

UCCKPL 位 6 时钟极性选择。
- 0 未激活的状态是低电平。
- 1 未激活的状态是高电平。

UCMSB 位 5 MSB 优先选择。 控制移位寄存器接收和发送的方向。
- 0 LSB 优先
- 1 MSB 优先

UC7BIT 位 4 字符长度。 选择 7 位或 8 位字符长度。
- 0 8 位数据。
- 1 7 位数据。

UCMST 位 3 主控模式选择
- 0 受控模式
- 1 主控模式

UCMODEx 位 2-1 USCI 模式。 当 UCSYNC=1 时，UCMODEx 位选择同步模式。
- 00 3 引脚 SPI
- 01 UCxSTE 高电平有效时的 4 引脚 SPI：当 UCxSTE=1 时，从器件模式被启用
- 10 UCxSTE 低电平有效时的 4 引脚 SPI：当 UCxSTE=0 时，从器件模式被启用
- 11 I^2C 模式

UCSYNC 位 0 同步模式使能
- 0 异步模式
- 1 同步模式

16.4.2 UCAxCTL1，USCI_Ax 控制寄存器 1，UCBxCTL1，USCI_Bx 控制寄存器 1

7	6	5	4	3	2	1	0
UCSSELx		未被使用					**UCSWRST**
rw-0	rw-0	rw-0[1] r0[2]	rw-0	rw-0	rw-0	rw-0	rw-1

UCSSELx 位 7-6 USCI 时钟源选择。 这些位选择在主器件模式下的 BRCLK 时钟源。 UCxCLK 总是在从器件模式下使用。
- 00 不可用
- 01 ACLK
- 10 SMCLK
- 11 SMCLK

未被使用 位 5-1 未被使用

UCSWRST 位 0 软件复位使能
- 0 被禁用。 USCI 复位操作释放
- 1 被启用. USCI 逻辑保持在复位状态。

(1) UCAxCTL1(USCI_Ax)
(2) UCBxCTL1(USCI_Bx)

16.4.3 UCAxBR0，USCI_Ax 比特率控制寄存器 0，UCBxBR0，USCI_Bx 比特率控制寄存器 0

7	6	5	4	3	2	1	0
UCBRx-低字节							
rw	rw	rw	rw	rw	rw	rw	rw

16.4.4 UCAxBR1，USCI_Ax 比特率控制寄存器 1，UCBxBR1，USCI_Bx 比特率控制寄存器 1

7	6	5	4	3	2	1	0
UCBRx-高字节							
rw	rw	rw	rw	rw	rw	rw	rw

UCBRx 位时钟预分频器设置。 (UCxxBR0 + UCxxBR1 × 256) 的 16 值组成了预分频器值。

16.4.5 UCAxSTAT，USCI_Ax 状态寄存器，UCBxSTAT，USCI_Bx 状态寄存器

7	6	5	4	3	2	1	0
UCLISTEN	**UCFE**	**UCOE**	未被使用				**UCBUSY**
rw-0	rw-0	rw-0	rw-0[1] r0[2]	rw-0	rw-0	rw-0	r-0

UCLISTEN	位 7	监听使能。 UCLISTEN 位选择回路模式。 0 被禁用 1 被启用. 发送器输出从内部反馈到接收器。
UCFE	位 6	组帧错误标志。 该位表示了一个 4 线制主控模式中的总线冲突。 UCFE 不能用于 3 线制主控模式或任何受控模式。 0 无错误 1 发生总线冲突
UCOE	位 5	溢出错误标志。 当在之前的一个字符被读取前随后一个字符被传输到 UCxRXBUF 时，该位被置位。 当 UCxRXBUF 被读取时，UCOE 被自动清除，而且 UCOE 绝不能用软件清除。 否则，它将无法正常工作。 0 无错误 1 发生溢出错误
未被使用	位 4-1	未被使用
UCBUSY	位 0	USCI 忙。 该位表示一个发送或接收操作正在进行。 0 USCI 未激活 1 USCI 发送或接收

(1) UCAxSTAT (USCI_Ax)
(2) UCBxSTAT (USCI_Bx)

16.4.6 UCAxRXBUF，USCI_Ax 接收缓冲寄存器，UCBxRXBUF，USCI_Bx 接收缓冲寄存器

7	6	5	4	3	2	1	0
UCRXBUFx							
r	r	r	r	r	r	r	r

UCRXBUFx 位 7-0 接收数据缓冲区是用户可以访问的并且包含最后从接收移位寄存器中接收到的字符。 读取 UCxRXBUF 复位接收错误位，和 UCxRXIFG。 在 7 位数据模式下，UCxRXBUF 是对齐的 LSB 并且 MSB 总是复位。

16.4.7 UCAxTXBUF，USCI_Ax 发送缓冲寄存器，UCBxTXBUF，USCI_Bx 发送缓冲寄存器

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

UCTXBUFx	位 7-0	发送数据缓冲区是用户可以访问的并且保存有等待被转移到发送移位寄存器和传输的数据。 写入到发送数据缓冲器清除 UCxTXIFG。 UCxTXBUF 的 MSB 没有在 7 位数据模式下使用且被复位了。

16.4.8 IE2，中断使能寄存器 2

7	6	5	4	3	2	1	0
				UCB0TXIE	UCB0RXIE	UCA0TXIE	UCA0RXIE
				rw-0	rw-0	rw-0	rw-0

	位 7-4	这些位可能用于其他模块（请参阅《器件专用数据表》）。
UCB0TXIE	位 3	USCI_B0 发送中断启用 0 中断被禁用 1 中断被启用
UCB0RXIE	位 2	USCI_B0 接收中断启用 0 中断被禁用 1 中断被启用
UCA0TXIE	位 1	USCI_A0 发送中断启用 0 中断被禁用 1 中断被启用
UCA0RXIE	位 0	USCI_A0 接收中断启用 0 中断被禁用 1 中断被启用

16.4.9 IFG2，中断标志寄存器 2

7	6	5	4	3	2	1	0
				UCB0TXIFG	UCB0RXIFG	UCA0TXIFG	UCA0RXIFG
				rw-1	rw-0	rw-1	rw-0

	位 7-4	这些位可能用于其他模块（参阅《器件专用数据表》）。
UCB0TXIFG	位 3	USCI_B0 发送中断标志。 UCB0TXBUF 为空时，UCB0TXIFG 被置位。 0 无中断等待 1 中断等待
UCB0RXIFG	位 2	USCI_B0 接收中断标志. 当 UCB0RXBUF 已经接收一个完整字符时，UCB0RXIFG 被置位。 0 无中断等待 1 中断等待
UCA0TXIFG	位 1	USCI_A0 发送中断标志. UCA0TXBUF 为空时，UCA0TXIFG 被置位。 0 无中断等待 1 中断等待
UCA0RXIFG	位 0	USCI_A0 接收中断标志. 当 UCA0RXBUF 已经接收一个完整字符时，UCA0RXIFG 被置位。 0 无中断等待 1 中断等待

16.4.10 UC1IE，USCI_A1/USCI_B1 中断使能寄存器

7	6	5	4	3	2	1	0
未被使用				**UCB1TXIE**	**UCB1RXIE**	**UCA1TXIE**	**UCA1RXIE**
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

未被使用	位 7-4	未被使用
UCB1TXIE	位 3	USCI_B1 发送中断使能
		0 中断被禁用
		1 中断被启用
UCB1RXIE	位 2	USCI_B1 接收中断使能
		0 中断被禁用
		1 中断被启用
UCA1TXIE	位 1	USCI_A1 发送中断使能
		0 中断被禁用
		1 中断被启用
UCA1RXIE	位 0	USCI_A1 接收中断使能
		0 中断被禁用
		1 中断被启用

16.4.11 UC1IFG，USCI_A1/USCI_B1 中断标志寄存器

7	6	5	4	3	2	1	0
未被使用				**UCB1TXIFG**	**UCB1RXIFG**	**UCA1TXIFG**	**UCA1RXIFG**
rw-0	rw-0	rw-0	rw-0	rw-1	rw-0	rw-1	rw-0

未被使用	位 7-4	未被使用
UCB1TXIFG	位 3	USCI_B1 发送中断标志。 UCB1TXBUF 为空时，UCB1TXIFG 被置位。
		0 无中断等待
		1 中断等待
UCB1RXIFG	位 2	USCI_B1 接收中断标志。 当 UCB1RXBUF 已经接收一个完整字符时，UCB1RXIFG 被置位。
		0 无中断等待
		1 中断等待
UCA1TXIFG	位 1	USCI_A1 发送中断标志。 UCA1TXBUF 为空时，UCA1TXIFG 被置位。
		0 无中断等待
		1 中断等待
UCA1RXIFG	位 0	USCI_A1 接收中断标志。 当 UCA1RXBUF 已经接收一个完整字符时，UCA1RXIFG 被置位。
		0 无中断等待
		1 中断等待

通用串行通信接口，I²C 模式

通用串行通信接口 (USCI) 支持一个硬件模块下的多通道串行通信模式。 本章讨论了 I²C 模式的运行。

Topic **Page**

17.1 USCI 概述

通用串行通信接口 (USCI) 模块支持多种串行通信模式。 不同的 USCI 模块支持不同的模式。 每种不同的 USCI 模块用不同的字母命名。 例如，USCI_A 与 USCI_B 是不同的，等等。 如果在一台器件上执行多个相同的 USCI 模块，那么这些模块用递增的数字命名。 例如，如果一个设备有两个 USCI_A 模块，它们被命名为 USCI_A0 和 USCI_A1。 请参阅器件专用数据表来决定在哪种器件执行哪一个 USCI 模块。

USCI_Ax 模块支持：

- UART 模式
- 用于 IrDA 通信的脉冲整形
- 用于 LIN 通信的波特率自动检测
- SPI 模式

USCI-Bx 模块支持：

- I^2C 模式
- SPI 模式

17.2 USCI 介绍：I^2C 模式

在 I^2C 模式中，USCI 模块在 MSP430 和用两线式 I^2C 串行总线方式连接的 I^2C 兼容器件之间提供了一个接口。 串连至 I^2C 总线的外部组件通过双线 I^2C 接口与 USCI 模块相互传输串行数据。

I^2C 模式的特点包括：

- 符合 2.1 版本飞利浦半导体 I^2C 技术规格
  - 7 位和 10 位器件寻址模式
  - 常规调用
  - 起始/重新起始/停止
  - 多主控发送器/接收器模式
  - 受控接收器/发送器模式
  - 标准模式下可达 100kbps，支持快速模式下高达 400kbps
- 主控模式下可编程的 UCxCLK 频率
- 低功耗设计
- 受控接收器起始从 LPMx 模式中自动唤醒的检测
- LPM4 下的受控模式运行

当在 I^2C 模式中配置后，图 17-1就会给出USCI。

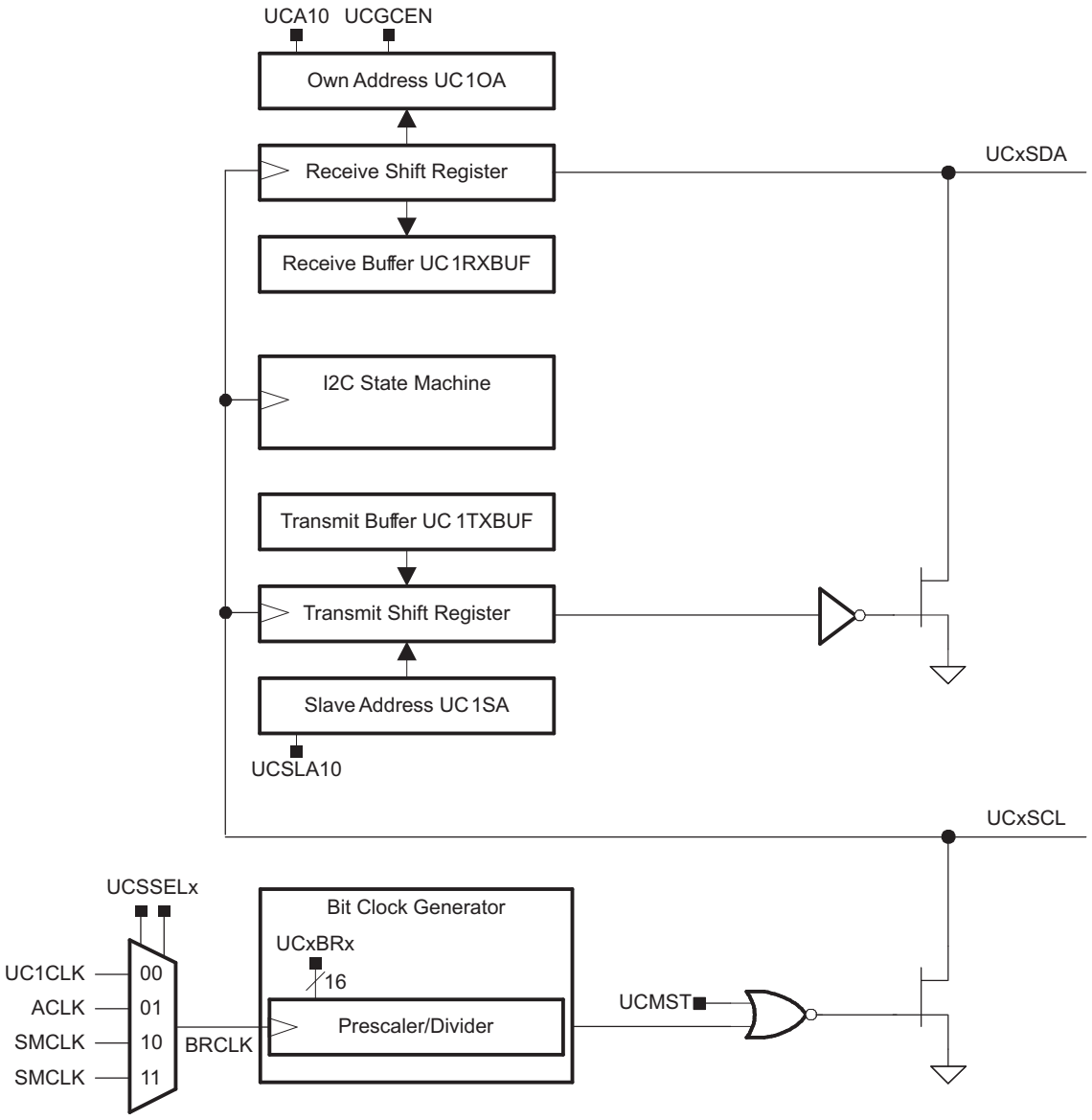


图 17-1. USCI 方框图：I²C 模式

17.3 USCI 运行：I²C 模式

I²C 模式支持任何从器件或主器件 I²C 相兼容器件。图 17-2给出了一个 I²C 总线的一个例子。 每一个 I²C 器件被一个唯一的地址识别，并可以作为发送器或接收器。 当进行数据传输时，一个连接到 I²C 总线的器件可被看作主器件或从器件。 一个主器件发起数据传输，并生成时钟信号 SCL。 任何由主器件寻址的器件被看作是一个从器件。

I²C 数据是用串行数据引脚 (SDA) 和串行时钟引 (SCL) 进行通信的。 SDA 和 SCL 是双向的，并且必须被连接到使用一个上拉电阻的正电源电压。

注： **SDA 和 SCL 电平**

MSP430 SDA 和 SCL 引脚不能被拉高至超过 MSP430 V_{CC}电平。

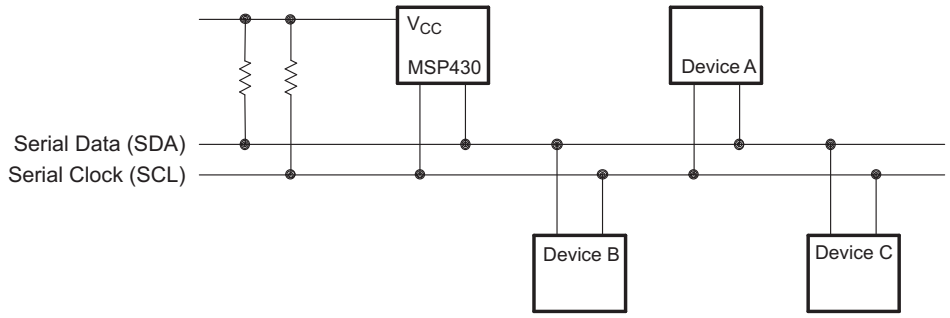


图 **17-2. I²C 总线连接框图**

17.3.1 USCI 初始化和复位

USCI由一个 PUC 进行复位或由 UCSWRST 位设置。 在一个 PUC 之后，UCSWRST 位被自动设置，从而保持了 USCI 处于复位状态。 为了选择 I²C 运行，UCMODEx 位必须设置为 11。 模块被初始化后，已准备好发送或接收操作。 清除 UCSWRST 位会使 USCI 处于运行状态。

当 UCSWRST 被设定以便避免不可预知的运行状态时，应完成 USCI 模块的配置和重新配置。 在 I²C 模式下设置 UCSWRST 具有以下作用：

- I²C 通信停止
- SDA 和 SCL 为高阻抗
- UCBxI2CSTAT，6-0 位被清零
- UCBxTXIE 和 UCBxRXIE 被清零
- UCBxTXIFG 和 UCBxRXIFG 被清零
- 所有其他位和寄存器保持不变。

注： 初始化或重新配置 USCI 模块

建议的 USCI 初始化或重新配置过程是：

1. 置位 UCSWRST (BIS.B＃UCSWRST，UCxCTL1)
2. UCSWRST= 1，初始化所有 USCI 寄存器（包括 UCxCTL1）
3. 配置端口。
4. 通过软件清零 UCSWRST (BIC.B＃UCSWRST，UCxCTL1)
5. 通过 UCxRXIE 和/或 UCxTXIE 使能中断（可选）

17.3.2 I²C 串行数据

由主器件为每个已传送的数据位产生一个时钟脉冲。 I²C 模式用字节数据运行。 传输的数据是最高有效位，如在图 17-3中所示。

在一个起始条件后的第一个字节包含一个 7 位从器件地址和 R/W 位。 当 R/W = 0 时，主器件向从器件发送数据。 当 R/W = 1 时，主器件从从器件处接收数据。 ACK 位由在第 9 个 SCL 时钟上的每个字节后的接收器发送。

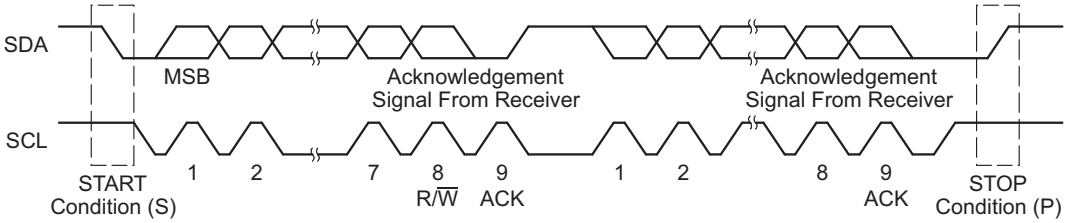


图 17-3. I²C 模块数据传输

起始和停止条件是由主器件生成，并显示在图 17-3中。 一个起始条件是，在 SCL 为高电平时，SDA 线上的由高电平至低电平的过渡。 一个停止条件是，在 SCL 为高电平时，SDA 线上的由低电平至高电平的过渡。 总线忙位，UCBBUSY，在起始后置位，在停止后清零。

SCL 为高电平期间，SDA 上的数据必须保持稳定，如图 17-4中所示。 只能在 SCL 为低时才可以改变 SDA 的高低电平状态，否则将会产生起始和停止条件。

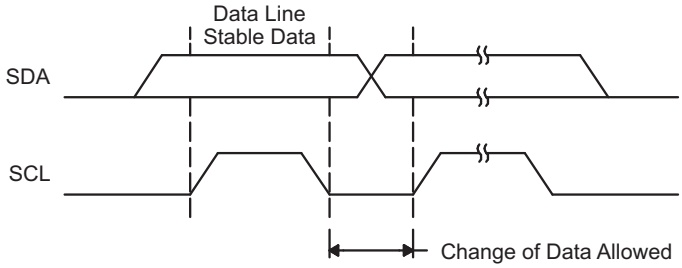


图 17-4. 在 I²C 总线上的位传输

17.3.3 I²C 寻址模式

I²C 模式支持 7 位和 10 位的寻址模式。

17.3.3.1 7 位寻址

7 位寻址的格式如在图 17-5中所示M 第一个字节是 7 位从器件地址和 R/w 位。 应答位 ACK 是在每个字节后由接收器发出的。

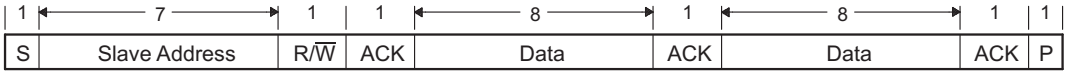


图 17-5. I²C 模块 7 位寻址格式

17.3.3.2 10 位寻址

10 位寻址的格式如在图 17-6中所示，第一个字节由 11110b 加上 10 位从器件地址的两个最高位 (MSB) 和 R/W 位构成。 应答位 ACK 是在每个字节后由接收器发出的。 下一个字节是 10 位从地址中剩下的低 8 位，而后是 ACK 应答位和 8 位数据。

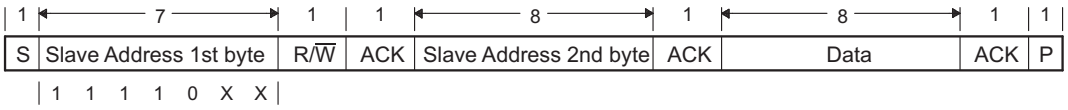


图 17-6. I²C 模块 10 位寻址格式

17.3.3.3 重复起始条件

主器件可以在不先停止一个传输的情况下，通过一个重复起始条件来改变 SDA 上数据流的方向。 这称为重新起始。 重新起始生成后，从器件地址被 R/W 位指定的新数据方向再次发送。 在图 17-7中给出了重新起始的条件。

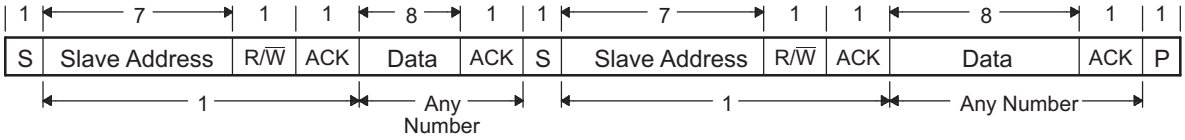


图 17-7. I²C 模块重复起始条件的寻址格式

17.3.4 I²C 模块的运行模式

在 I²C 模式下，USCI 模块可以在主器件发送模式，主器件接收模式，从器件发送模式或受控接收器模式下工作。 下面的章节对这些模式进行了讨论。 用时序线路来对这些模式进行阐明。

图 17-8给出了如何解释这些时序线路图表。 主器件发送的数据用灰色的矩形块表示，从器件发送的数据用白色的矩形块表示。 不管作为主器件还是从器件，稍高的矩形块表示的是由 USCI 模块发送的数据。

USCI 模块的行为用一个带箭头的灰色矩形块表示，其箭头所指的数据流位置就是动作发生的地方。 那些必须用软件处理的动作由带箭头的白色矩形方块表示，箭头指向的是数据流中动作必须发生的位置。

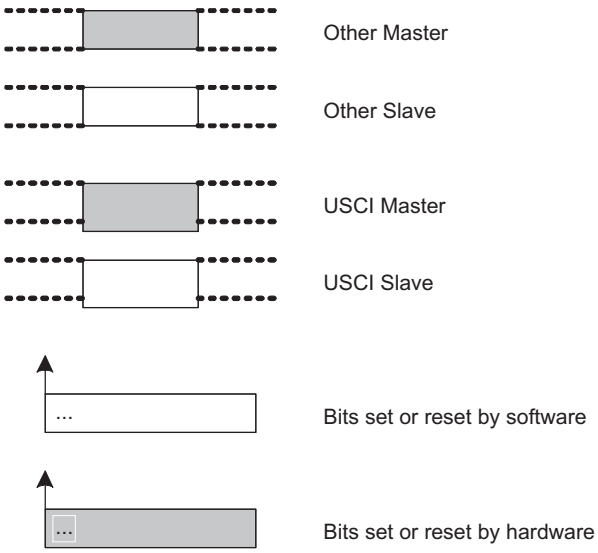


图 17-8. I²C 时序线路图例

17.3.4.1 受控模式

通过把 I²C 模式设置为 UCMODEx=11 和 USCYNC=1，并清除 UCMST 位，USCI 模块被配置为一个 I²C 从器件。

首先，USCI 模块必须在接收模式下通过清除 UCTR 位进行配制，以便接收 I²C 从器件地址。 之后，发送和接收操作是由与从器件一起接收到的 R/W 位自动决定。

USCI 从器件地址是由 UCBxI2COA 寄存器编程的。 当 UCA10=0 时，选用 7 位寻址方式。 当 UCA10=1 时，选用 10 位寻址方式。 如果从器件响应一个常规调用，则可以选择 UCGCEN 位。

当在总线上检测到起始条件时，USCI 模块将接收传送过来的地址，并将之与存储在 UCBxI2C0A 中的本器件地址相比较。 若接收地址与 USCI 从器件地址一致，则置位 UCSTTIFG 标志。

17.3.4.1.1 I²C 受控发送器模式

当主器件发送的从器件地址和带有一个设置 R/W 位的其自身地址相匹配时，从器件进入发送模式。 受控传输器依靠主器件产生的时钟脉冲信号在 SDA 上移位传输串行数据。 从器件不能产生时钟时钟M 但是当发送完一个字节后需要 CPU 的干预时，从器件能够保持 SCL 为低电平。

如果主器件向从器件器件请求数据，USCI 模块会自动配置为发送模式，并置位 UCTR 和 UCBxTXIFG。 在数据未写入发送缓存 UCBxTXBUF之前，SCL 时钟线一直保持低电平。 当地址被响应后，清除 UCSTTIFG 标志，然后开始传输数据。 一旦数据被转移到移位寄存器，UCTXIFG 将再次被置位。 被主器件确认之后，下一个被写入 UCBxTXBUF 中的字节数据开始传输，或发送缓冲区为空，通过一直保持 SCL 为低电平直到新的数据被写到 UCBxTXBUF 内，在应答周期内总线被挂起。 假如主器件通过一个停止条件成功发送了一个 NACK 信号，则 UCSTPIFG 被置位。 如若 NACK 被一个重复起始条件成功发送，则USCI I²C 状态机返回至其地址接收状态。

图 17-9给出了受控发送器运行。

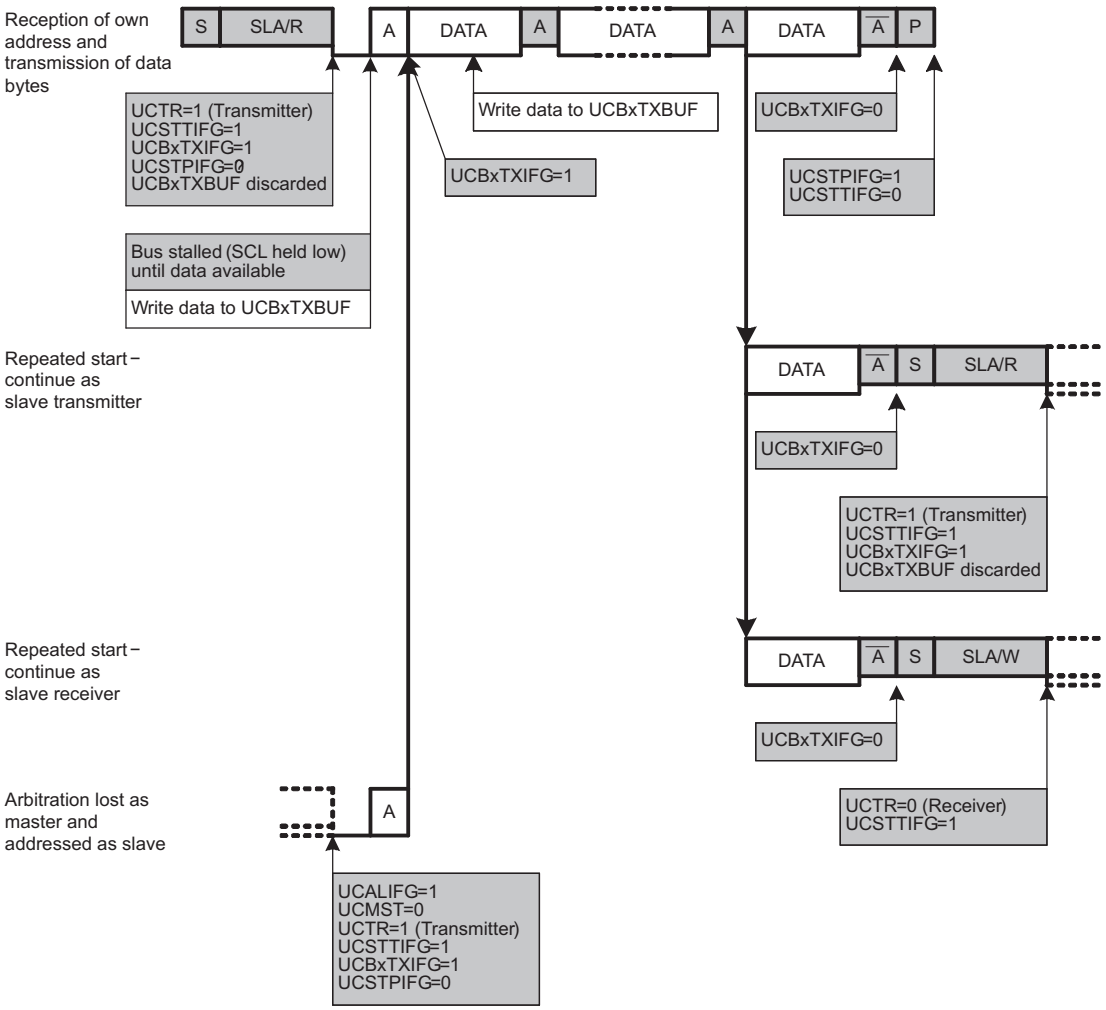


图 **17-9. I²C 受控发送器模式**

17.3.4.1.2 I²C 受控接收器模式

当主器件发送的从器件地址和其本地地址相匹配，且接收到被清零的 R/W 时，从器件进入接收模式。 在从器件接收模式中，从器件根据主器件产生的时钟脉冲信号在 SDA 上接收串行数据。 从设备不能产生时钟，但是当一个字节接收完毕需要 CPU 的干预时，从器件可保持 SCL 为低电平。

如果从器件需要接收主器件发送过来的数据，则 USCI 模块将自动配置为接收，并将 UCTR 清零。 在接收完第一个数据字节后，接收中断标志 UCBxRXIFG 被置位。 USCI 模块会自动应答接收到的数据并可接收下一个数据字节。

如果在一个接收完成之后没能从接收缓存 UCBxRXBUF 内读出前一个数据，则通过保持 SCL 为低电平，总线被停止。 一旦 UCBxRXBUF 被读取，新数据就会被传输到 UCBxRXBUF，就会把一个应答信号给主器件，然后开始下个数据的接收。

置位 UCTXNACK 会导致在下一个应答周期内发送一个 NACK 信号给主器件。 即使 UCBxRXBUF 没有准备好接收最新数据，也将会立即发送一个 NACK。 如果在 SCL 为低电平时置位 UCTXNACK 将会释放总线，并马上会发送一个NACK 信号，同时 UCBxRXBUF 将装载最后一次接收到的数据。 由于先前的数据还没有被读出，这将造成数据丢失。 为避免数据的丢失，应在 UCTXNACK 置位之前读出 UCBxRXBUF 中的数据。

当主设备产生一个停止停止条件时，UCSTPIFG 标志被置位。

如果主器件产生一个重复起始条件时，则 USCI I²C 将返回到地址接收状态。

图 17-10给出了 I²C 受控接收器操作。

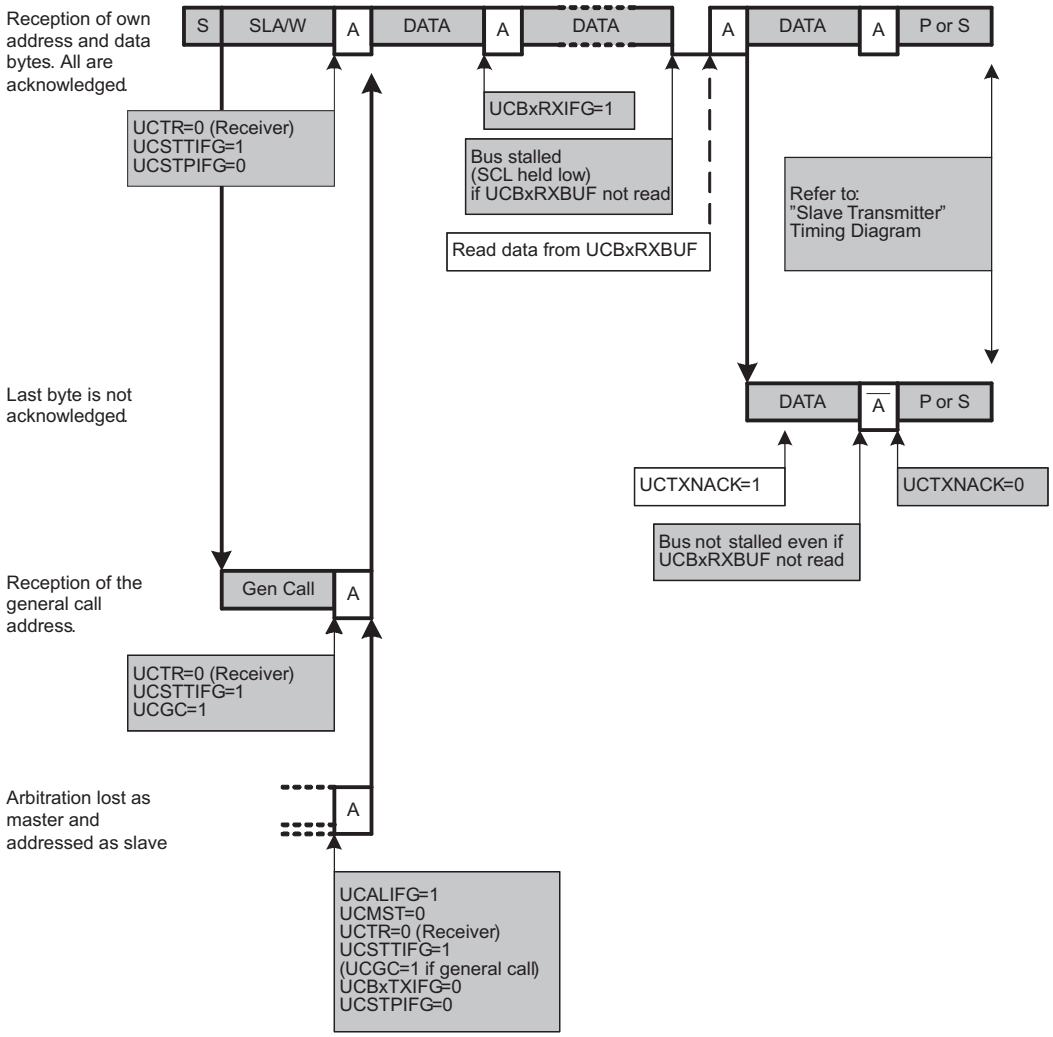


图 **17-10. I²C** 受控接收器模式

17.3.4.1.3 I²C 从器件 10 位寻址模式

如在图 17-11中所示，当 UCA 10=1 时选用 10 位寻址模式。 在 10 位寻址模式下，整个地址接收完毕后从器件处于接收模式。 USCI 模块会通过在清零 UCTR 位的同时置位 UCSTTIFG 来指明上述操作。 若需要将从器件切换到发送模式，则需要主器件在发送一个重复起始条件后紧跟着发送最开始的地址字节，同时发送 R/W 位置位。 若标志 UCSTTIFG 之前被软件清除，同时 USCI 模块转变为发送模式，且 UCTR =1，那么此时将被置位。

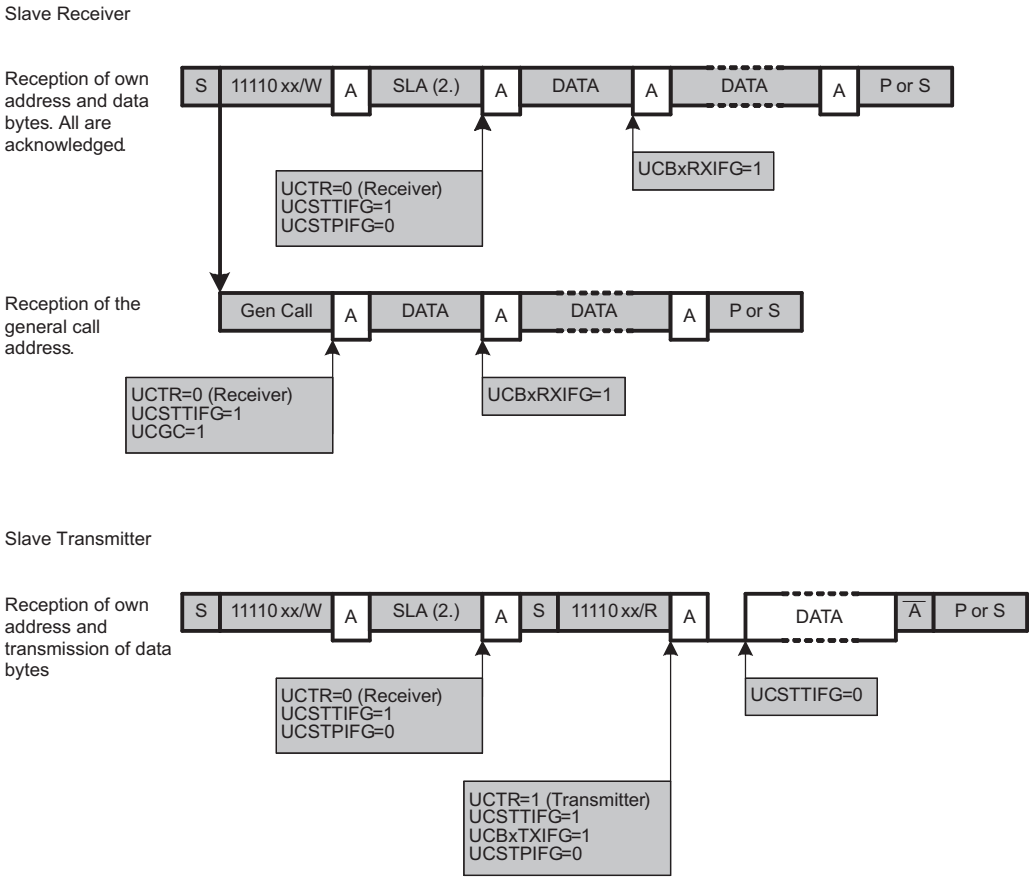


图 17-11. I²C 从器件 10 位寻址模式

17.3.4.2 主控模式

通过把 I²C 模式设置为 UCMODEx= 11 和 USCYNC=1，并置位 UCMST 位，USCI 模块被配置为一个 I²C 主器件。 当主器件是多主器件系统的一部分时，必须置位 UCMM，并且其自身地址必须被编入 UCBxI2COA 寄存器。 当 UCA10=0 时，选用 7 位寻址方式。 当 UCA10=1 时，选用 10 位寻址方式。 如果 USCI 模块响应常规调用，则可以选择 UCGCEN 位。

17.3.4.2.1 I²C 主控发送器模式

初始化之后，通过把目标从器件地址写入寄存器 UCBxI2CSA、用 UCSLA 10 位来选择从器件地址的位数、置位 UCTR 来选择发送模式、置位 UCTXSTT 来产生一个起始条件，主控发送器模式才被初始化。

USCI 模块先检测总线是否空闲，之后产生一个起始条件，并传送从器件地址。 当起始条件产生，CBxTXIFG 将被置位，并将要发送的第一个数据写入 UCBxTXBUF 中。 一旦从器件对地址作出应答，UCTXSTT 位会被清零。

在从器件地址的发送过程中，如果仲裁没有失效，那么会已发送写入到 UCBxTXBUF 中的数据。 一旦数据由缓冲区转移到移位寄存器，UCBxTXIFG 将再次置位。 如果在应答周期到来之前 UCBxTXBUF 中没有装载新数据，那么在应答周期过程中总线将被挂起，SCL 将保持拉低电平状态，直到数据写入缓存器 UCBxTXBUF 中。 只要 UCTXSTP 位或 UCTXSTT 位没被置位，数据就就会被传输或总线被保持。

在从器件下一个应答信号到来之后，置位 UCTXSTP 将会产生一个停止条件。 如果在从器件的地址传送过程或者是 USCI 模块等待把数据写入 UCBxTXBUF 的过程中置位 UCTXSTP，则即使没有数据被发送到从器件依旧会产生一个停止条件。 如果传送的是单字节数据，在字节传送过程中或者在数据传输开始后必须置位 UCTXSTP，不要将任何新的数据写入 UCBxTXBUF。 否则，会造成只传送地址。 当数据由缓冲器转移到发送移位寄存器时，UCBxTXIFG 将被置位，这表示着数据传输已经开始，可以置位 UCTXSTP 了。

置位 UCTXSTT 将会产生一个重复起始条件。 在这种情况下，可以通过置位或清零 UCTR，以便配置为发送器或接收器。

如果从器件没有响应发送的数据，则未响应中断标志 UCNACKIFG 会被置位。 主器件必须发送一个停止条件或者重复起始条件的方式来响应。 如果已经把数据写入 UCBxTXBUF，那么当前数据将被丢弃。 如果在一个重复起始条件后，这个数据还要发送出去，则必须重新将其写入 UCBxTXBUF。 任何置位 UCTXSTT 也会被丢弃。 若要触发一个重复起始条件，UCTXSTT 需要重新被置位。

图 17-12给出了 I²C 主器件发送操作。

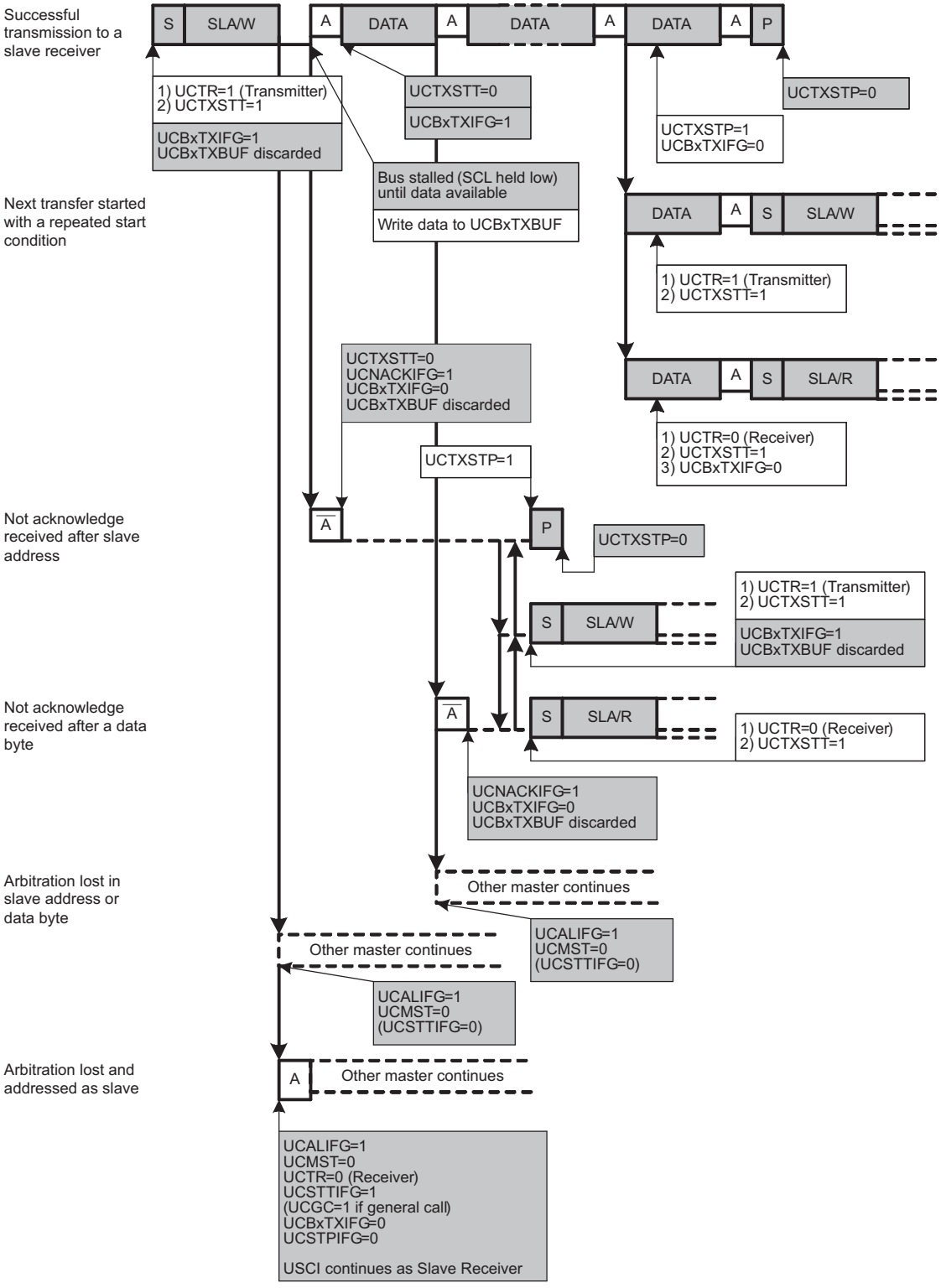


图 **17-12. I²C** 主控发送器模式

17.3.4.2.2 I²C 主控接收器模式

初始化之后，通过把目标从器件地址写入寄存器 UCBxI2CSA、用 UCSLA 10 位来选择从器件地址的位数、置位 UCTR 来选择发送模式、置位 UCTXSTT 来产生一个起始条件，主器件接收模式才被初始化。

USCI 模块先检测总线是否空闲，之后产生一个起始条件，并传送从器件地址。 一旦从器件对地址作出应答，UCTXSTT 位会被清零。

在从器件对地址应答后，将接收到从器件发送的第一个数据字节并发送应答信号，同时置位 UCBxRXIFG 标志。 只要 UCTXSTP 或 UCTXSTT 不被置位，就能接收到从器件发来的数据。 若没有读取 UCBxRXBUF，那么主器件将在接收最后到一个数据位后挂起总线直到 UCBxRXBUF 被读取。

如果从器件没有响应发送的地址，则未响应中断标志 UCNACKIFG 会被置位。 主器件必须发送一个停止条件或者重复起始条件的方式来响应。

置位 UCTXSTP 将会产生一个停止条件。 置位 UCTXSTP 后，主器件将在接收完从设备传送的数据后发出一个NACK，并紧接着发送一个停止，或者如果在 USCI 模块正在等待读取 UCBxRXBUF时，将立即产生停止。

如果主及想接收一个单字节数据，那么在接收字节的过程中必须将 UCTXSPT 位置位。 在这种情况下，可以通过查询 UCTXSTT 来确定何时将被清除：

```
BIS.B #UCTXSTT,&UCBOCTL1 ;Transmit START cond.POLL_STT BIT.B #UCTXSTT,&UCBOCTL1 ;Poll UCTXSTT
bitJC POLL_STT ;When cleared,BIS.B #UCTXSTP,&UCB0CTL1 ;transmit STOP cond.
```

置位 UCTXSTT 将会产生一个重复起始条件。 在这种情况下，可以通过置位或清零 UCTR，以便配置为发送器或接收器，如果需要的话，还可以把不同的地址写入 UCBxI2CSA。

图 17-13给出了 I²C 主控接收器操作。

注： **在不使用重复起始的情况下的连续主器件传输**

在不使用重复起始功能的情况下，当进行多个连续 I²C 主器件传输时，当前传输必须在下一个传输初始化完成之前结束。 这可以通过确保在下一个 I²C 传输初始化完成之前发送停止条件标志 UCTXSTP 被清零，并设置 UCTXSTT= 1来完成。否则，将会影响当前的传输。

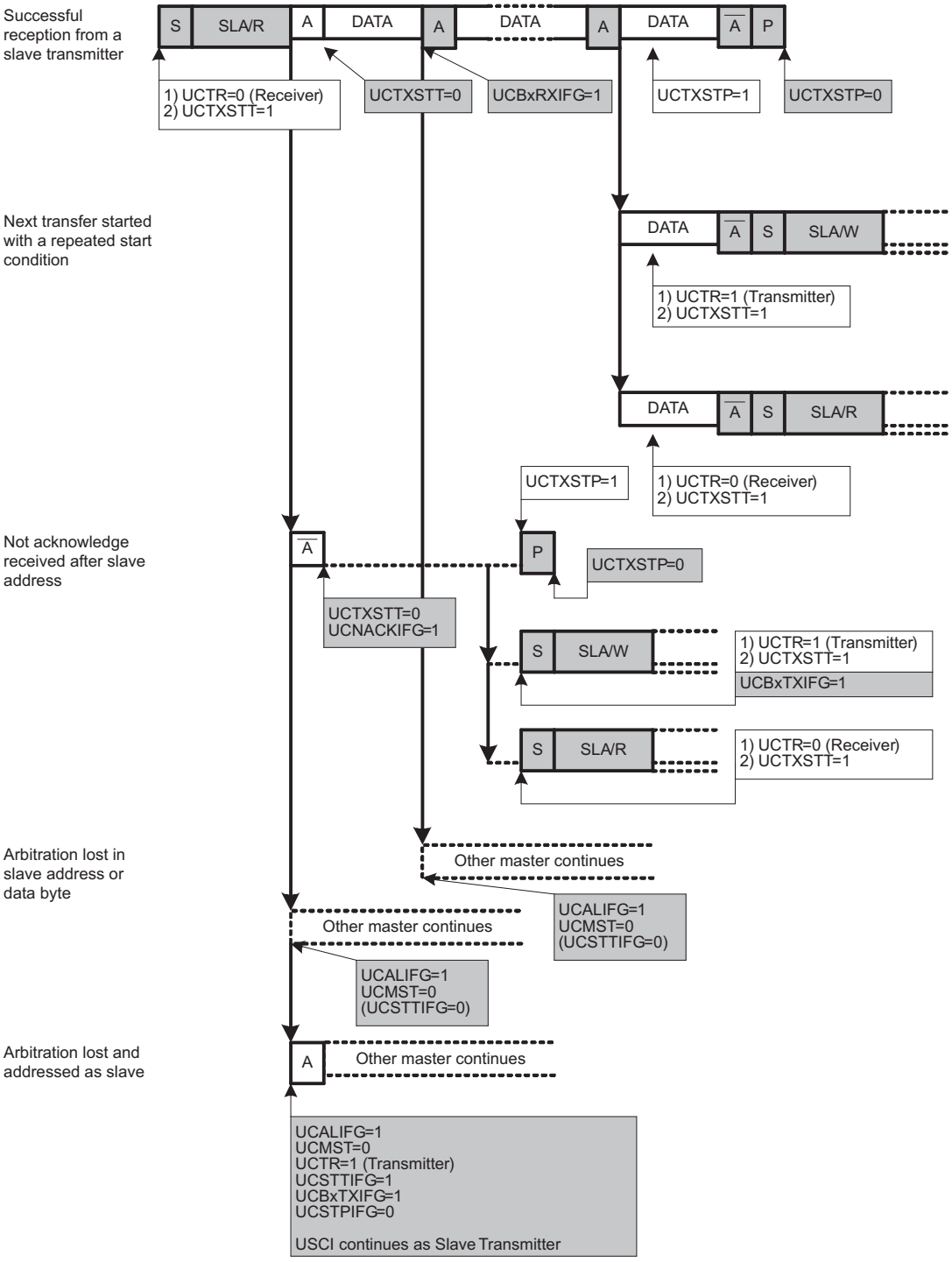


图 **17-13. I²C** 主控接收器模式

17.3.4.2.3 I²C 主器件 10 位寻址模式

如在图 17-14中所示，当 USCLA 10=1 时选用 10 位寻址模式。

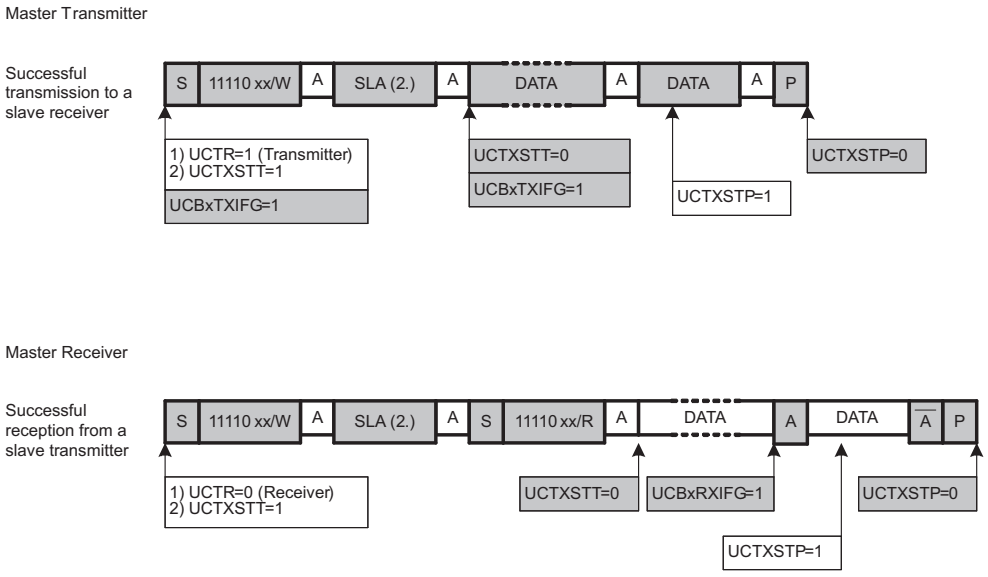


图 17-14. I²C 主器件 10 位寻址模式

17.3.4.2.4 仲裁

当两个或两个以上的主器件发送在总线上同时传输时，就会起始一个仲裁程序。图 17-15描述了对两个器件间的仲裁程序。 仲裁程序使用由相互竞争的发送器发送到 SDA 上的数据。 生成一个逻辑高电平的第一个主控发送器将被逻辑低电平的和其竞争的主器件发送其覆盖。 仲裁进程将优先权授予用最低二进制值传送串行数据流的器件。 失去仲裁的主控发送器将转换成受控接收器模式，并置位仲裁失去标志 UCALIFG。 如果两个或两个以上的器件发送相同的第一个字节，则仲裁会在后续字节中继续发生作用。

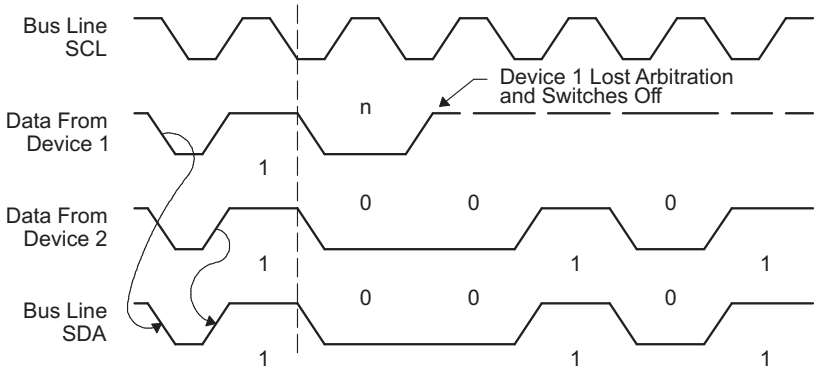


图 17-15. 在两个主控发送器之间的仲裁

如果在仲裁正在进行中，在 SDA 上一个有重复起始条件或者停止条件在传送时，那么在仲裁进程中的所有主控发送器都必须在帧格式中的同一个位置发送重复起始或者停止条件。 仲裁不会在下列几组间发生：

- 一个重复起始条件和数据位之间
- 一个停止条件和数据位之间
- 一个重复起始条件和一个停止条件之间

17.3.5 I²C 时钟的发生与同步

I²C 时钟 SCL 是由在 I²C 总线上的主器件提供。 当 USCI 处于主器件发送模式下时，BITCLK 由 USCI位时钟发生器提供，同时通过 UCSSELx 位选择时钟源。 在受控模式下，位时钟发生器不工作，且 UCSSELx位无效。

寄存器 UCBxBR1 和 UCBxBR0 中 UCBRx 的 16 位值是 USCI 时钟源 BRCLK 的分频因子。 在单主控模式下，可用的最大位时钟为 $f_{BRCLK}/4$。在多主控模式下，最大位时钟为 $f_{BRCLK}/8$。BITCLK 的频率可由以下得到：M

$$f_{BitClock} = \frac{f_{BRCLK}}{UCBRx}$$

生成的 SCL 的最小高电平和低电平周期是

$$t_{LOW,MIN} = t_{HIGH,MIN} = \frac{UCBRx / 2}{f_{BRCLK}}$$ 当 UCBRx 为偶数时且

$$t_{LOW,MIN} = t_{HIGH,MIN} = \frac{(UCBRx - 1) / 2}{f_{BRCLK}}$$ 当 UCBRx 为奇数时。

为了满足 I²C 总线协议规定的最小高低电平周期，必须选择 USCI 时钟源的频率和 UCBRx 的分频因子设置。

在仲裁进程中，来自不同主器件的时钟必须进行同步处理。 在 SCL 上第一个产生低电平周期的器件会驳回其他器件，以便迫使其他器件也起始其本地低电平周期。 之后 SCL 被低电平周期最长的器件保持为低电平。 在其他器件的高电平周期开始之前，其他器件必须等待释放 SCL。图 17-16显示了时钟的同步。 这允许低速器件把高速器件拉低。

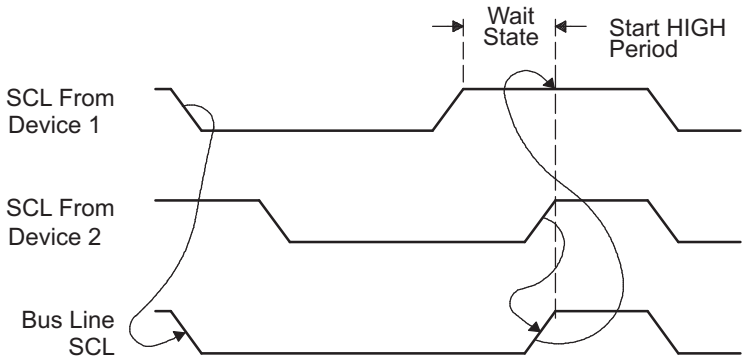


图 17-16. 在仲裁期间两个 I²C 时钟发生器的同步

17.3.5.1 时钟扩展

USCI 模块支持时钟扩展并可以和上述操作模式中讲述的一样进行使用。

在下列几种情况下，如果 USCI 模块已经释放了SCL，可以用 UCSCLLOW 位来检查是否其他的器件把 SCL 拉低。M

- USCI 作为主器件，且一个连接的从器件将 SCL 拉低。
- USCI 作为主器件，在仲裁进程中其他主器件把 SCL 拉低。

如果 USCI 模块由于作为发送器等待数据写入UCBxTXBUF 或者是作为接收器等待从 UCBxRXBUF 中读取数据而把 SCL 拉低时，UCSCLLOW 位同样可用。

由于逻辑检查外部 SCL，并把它与内部生成的 SCL 相比较之后才产生 SCL，所以在每一个 SCL 产生上升沿的瞬间 UCSCLLOW 位就有可能被置位。

17.3.6 在处于低功耗模式中的 I²C 模式中使用 USCI 模块

为了在低功耗模式下使用，USCI 模块为 SMCLK 提供了自动时钟激活。 当 SMCLK 是 USCI 的时钟源并无效时，由于器件处于一个低功耗模式，如果需要，USCI 模块都可忽略时钟源控制位设置而自动激活。 直到 USCI 模块回到其空闲状态，时钟都会保持激活条件。 USCI 模块恢复空闲条件后，时钟源的控制权会恢复到其控制位的设置。 不为时钟信号 (ACLK) 提供自动时钟激活。

当 USCI 模块激活一个无效的时钟源时，该时钟源将为整个设备变得活跃，并且这可能会影响到任何为了使用而被配置的外围设备时钟源。 例如，当 USCI 模块迫使 SMCLK 激活时，一个使用 SMCLK 的定时器将递增。

在 I²C 模式下，由于时钟是由外部主器件提供，所以就不需要内部时钟源。 在器件处于 LPM4 状态下并且所有内部时钟源被禁止时，就可以实现在 I²C 受控模式中操作 USCI。 接收或者发送中断可以将 CPU 从任何一种低功耗模式中唤醒。

17.3.7 I²C 模式下的 USCI 中断

在 I²C 模式中 USCI 模块有两个中断矢量。 一个中断矢量与发送和接收中断标志相关联。 另一个中断矢量的与状态变化中断标志相关联。 每个中断标志都有其本地中断使能位。 当一个中断被使能，且 GIE 位被置位时，该中断标志将会生成一个中断请求。 在有 DMA 控制器的器件上 DMA 传输将由 UCBxTXIFG 和 UCBxRXIFG 标志控制。

17.3.7.1 I²C 发送中断操作

为了说明 UCBxTXBUF 已经为接收下一个字符做好了准备，UCTXIFG 中断标志会被发送器置位。 如果此时 UCBxTXIE 和 GIE 也被置位，就会产生一个中断请求信号。 如果一个字符被写入 UCBxTXBUF 或者接收到 NACK ，UCBxTXIFG 会自动复位。 当选择 I²C 模式并且 UCSWRST=1 时，UCTXIFG 会被置位。 在一个 PUC 后或者当 UCSWRST=1 时，UCBxTXIE 被复位。

17.3.7.2 I²C 接收中断操作

当接收到一个字节并被装载到 UCBxRXBUF 中时，UCBxRXIFG 中断标志被置位。 如果此时 UCBxRXIE 和 GIE 也被置位，就会产生一个中断请求信号。 在一个 PUC 信号后或当 UCSWRST=1 时，UCBxRXIFG 和 UCBxRXIE 会被复位。在读取 UCxRXBUF 时，UCxRXIF 会自动复位。

17.3.7.3 I²C 状态更改中断操作

表 17-1描述了 I²C 状态更改中断标志。

表 17-1. 状态更改中断标志

中断标志	中断条件
UCALIFG	仲裁丢失。 在两个或两个以上的发送器同时开始发送数据时，或者是当 USCI 作为主器件工作，但被系统中其他主器件作为从器件来寻址时，可能会发生仲裁丢失。 当仲裁丢失时，UCALIFG 标志被置位。 当 UCALIFG 被置位时，UCMST 位被清零，同时 I²C 模块变成一个从器件。
UCNACKIFG	无应答中断。 当接收不到预期的应答时此标志被置位。 当接收到一个起始条件时，UCNACKIFG 被自动清零。
UCSTTIFG	起始条件检测到的中断。 在受控模式下，当 I²C 模块检测到一个带有其本地地址的起始条件时，该标志会被置位。 UCSTPIFG 只能在受控模式下使用，并且在接收到一个停止条件时被自动清零。
UCSTPIFG	停止条件检测到的中断。 在受控模式下，当 I²C 模块检测到一个停止条件时，该标志会被置位。 UCSTPIFG 只能在受控模式下使用，并且在接收到一个起始条件时被自动清零。

17.3.7.4 中断失量的分配

USCI_Ax 和 USCI_Bx 共享同一个中断矢量。 在 I²C 模式下，状态更改中断标志UCSTTIFG，UCSTPIFG，UCIFG，来自 USCI_Bx 的UCALIFG 和来自 USCI_Ax 的 UCAxRXIFG 被路由到一个中断矢量中。 I²C 发射和接收中断标志 UCBxTXIFG 和来自 USCI_Bx 的 UCBxRXIFG 以及来自 USCI_Ax 的 UCAxTXIFG 共享另一个中断矢量。

Example 17-1展示了一个中断处理例程的提取，该提取被用来处理处于 UART 或 SPI 模式下的 USCI_A0 的数据接收中断，并且用于处理处于 I²C 模式下的来自 USCI_B0 的更改中断。

Example 17-1. 共享的接收中断矢量软件示例

```
USCIA0_RX_USCIB0_I2C_STATE_ISRBIT.B #UCA0RXIFG, &IFG2 ; USCI_A0 Receive Interrupt?JNZ
USCIA0_RX_ISRUSCIB0_I2C_STATE_ISR; Decode I2C state changes ...; Decode I2C state changes
......RETIUSCIA0_RX_ISR; Read UCA0RXBUF ... - clears UCA0RXIFG...RETI
```

Example 17-2展示了一个中断处理例程的提取，该提取被用来处理处于 UART 或 SPI 模式下的 USCI_A0 的数据发送中断，并且用于处理处于 I²C 模式下的来自 USCI_B0 的传输中断。

Example 17-2. 共享的发送中断矢量软件示例

```
USCIA0_TX_USCIB0_I2C_DATA_ISRBIT.B #UCA0TXIFG, &IFG2 ; USCI_A0 Transmit Interrupt?JNZ
USCIA0_TX_ISRUSCIB0_I2C_DATA_ISRBIT.B #UCB0RXIFG, &IFG2JNZ USCIB0_I2C_RXUSCIB0_I2C_TX; Write
UCB0TXBUF... - clears UCB0TXIFG...RETIUSCIB0_I2C_RX; Read UCB0RXBUF... -
 clears UCB0RXIFG...RETIUSCIA0_TX_ISR; Write UCA0TXBUF ... - clears UCA0TXIFG...RETI
```

17.4 USCI 寄存器：I²C 模式

在 I²C 模式下可用于 USCI_B0 的 USCI 寄存器被列在表 17-2中，以及可用于 USCI_B1 的被列在表 17-3中。

表 17-2. USCI_B0 控制和状态寄存器

寄存器	简式	寄存器类型	地址	初始状态
USCI_BO 控制寄存器 0	UCB0CTL0	读取/写入	068h	001h 与 PUC
USCI_BO 控制寄存器 1	UCB0CTL1	读取/写入	069h	001h 与 PUC
USCI_B0 位率控制寄存器 0	UCB0BR0	读取/写入	06Ah	用 PUC 复位
USCI_B0 位率控制寄存器 1	UCB0BR1	读取/写入	06Bh	用 PUC 复位
USCI_B0 I²C 中断使能寄存器	UCB0I2CIE	读取/写入	06Ch	用 PUC 复位
USCI_B0 状态寄存器	UCB0STAT	读取/写入	06Dh	用 PUC 复位
USCI_B0 接收缓冲寄存器	UCB0RXBUF	读取	06Eh	用 PUC 复位
USCI_B0 发送缓冲寄存器	UCB0TXBUF	读取/写入	06Fh	用 PUC 复位
USCI_B0 I²C 本地址寄存器	UCB0I2COA	读取/写入	0118h	用 PUC 复位
USCI_B0 I²C 从器件地址寄存器	UCB0I2CSA	读取/写入	011Ah	用 PUC 复位
SFR 中断使能寄存器 2	IE2	读取/写入	001h	用 PUC 复位
SFR 中断标志寄存器 2	IFG2	读取/写入	003h	00Ah 与 PUC

注: 修改 SFR 位

为了避免修改其他模块的控制位，建议使用 BIS.B 或 BIC.B 指示，而非 MOV.B 或 CLR.B 指示来置位或清零 IEX 和 IFGx 位。

表 17-3. USCI_B1 控制和状态寄存器

寄存器	简式	寄存器类型	地址	初始状态
USCI_B1 控制寄存器 0	UCB1CTL0	读取/写入	0D8h	用 PUC 复位
USCI_B1 控制寄存器 1	UCB1CTL1	读取/写入	0D9h	001h 与 PUC
USCI_B1 波特率控制寄存器 0	UCB1BR0	读取/写入	0DAh	用 PUC 复位
USCI_B1 波特率控制寄存器 1	UCB1BR1	读取/写入	0DBh	用 PUC 复位
USCI_B1 I²C 中断使能寄存器	UCB1I2CIE	读取/写入	0DCh	用 PUC 复位
USCI_B1 状态寄存器	UCB1STAT	读取/写入	0DDh	用 PUC 复位
USCI_B1 接收缓冲寄存器	UCB1RXBUF	读取	0DEh	用 PUC 复位
USCI_B1 发送缓冲寄存器	UCB1TXBUF	读取/写入	0DFh	用 PUC 复位
USCI_B1 I²C 本地址寄存器	UCB1I2COA	读取/写入	017Ch	用 PUC 复位
USCI_B1 I²C 从器件地址寄存器	UCB1I2CSA	读取/写入	017Eh	用 PUC 复位
USCI_A1/B1 中断使能寄存器	UC1IE	读取/写入	006h	用 PUC 复位
USCI_A1/B1 中断标志寄存器	UC1IFG	读取/写入	007h	00Ah 与 PUC

17.4.1 UCBxCTL0，USCI_Bx 控制寄存器 0

7	6	5	4	3	2	1	0
UCA10	**UCSLA10**	**UCMM**	未使用	**UCMST**	**UCMODEx=11**		**UCSYNC=1**
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	r-1

UCA10 位 7 自身寻址模式选择
- 0 自身地址是一个 7 位地址
- 1 自身地址是一个 10 位地址

UCSLA10 位 6 从器件寻址模式选择
- 0 寻址具有 7 位地址的从器件
- 1 寻址具有 10 位地址的从器件

UCMM 位 5 多主器件环境选择
- 0 单主器件环境。 该系统内没有别的主器件。 地址比较单元被禁用。
- 1 多主器件环境

未被使用 位 4 未被使用

UCMST 位 3 主控模式选择。 当一个主器件在一个多主器件环境下 (UCMM = 1) 丢失仲裁时，UCMST 位就会自动清零，且该模块被视作从器件。
- 0 受控模式
- 1 主控模式

UCMODEx 位 2-1 USCI 模式。 当 UCSYNC=1 时，UCMODEx 位选择同步模式。
- 00 3 引脚 SPI
- 01 4 引脚 SPI（如果 STE=1，主器件/从器件被启用）
- 10 4 引脚 SPI（如果 STE= 0，主器件/从器件被启用）
- 11 I²C 模式

UCSYNC 位 0 同步模式使能
- 0 异步模式
- 1 同步模式

17.4.2 UCBxCTL1，USCI_Bx 控制寄存器 1

7	6	5	4	3	2	1	0
UCSSELx		未被使用	**UCTR**	**UCTXNACK**	**UCTXSTP**	**UCTXSTT**	**UCSWRST**
rw-0	rw-0	r0	rw-0	rw-0	rw-0	rw-0	rw-1

UCSSELx 位 7-6 USCI 时钟源选择。 这些位选择 BRCLK 时钟源。
- 00 UCLKI
- 01 ACLK
- 10 SMCLK
- 11 SMCLK

未被使用 位 5 未被使用

UCTR 位 4 发送器/接收器
- 0 接收器
- 1 发送器

UCTXNACK 位 3 发送一个 NACK。 在一个 NACK 发送完毕后，UCTXNACK 自动复位。
- 0 正常确认
- 1 生成 NACK

UCTXSTP 位 2 在主控模式下发送停止条件。 在受控模式下忽略。 在主控接收器模式下，一个 NACK 位于重复停止条件之前。 在停止生成后，UCTXSTP 自动清零。
- 0 无停止条件被生成
- 1 产生停止条件

UCTXSTT 位 1 在主控模式下发送起始条件。 在受控模式下忽略。 在主控接收器模式下，一个 NACK 位于一个重复起始条件之前。 在起始条件和地址信息被发送后，UCTXSTT 自动清零。 在受控模式下忽略。
- 0 不生成起始条件
- 1 生成起始条件

UCSWRST 位 0 软件复位使能
- 0 被禁用。 释放 USCI 复位以便进行操作。
- 1 被启用。 在复位状态中 USCI 逻辑状态被保持。

17.4.3 UCBxBR0, USCI_Bx 波特率控制寄存器 0

7	6	5	4	3	2	1	0
UCBRx- 低字节							
rw	rw	rw	rw	rw	rw	rw	rw

17.4.4 UCBxBR1, USCI_Bx 波特率控制寄存器 1

7	6	5	4	3	2	1	0
UCBRx- 高字节							
rw	rw	rw	rw	rw	rw	rw	rw

UCBRx 位时钟预分频器设置。 (UCBxBR0+ UCBxBR1×256) 的 16 位值构成预分频值。

17.4.5 UCBxSTAT，USCI_Bx 状态寄存器

7	6	5	4	3	2	1	0
未备使用	UCSCLLOW	UCGC	UCBBUSY	UCNACKIFG	UCSTPIFG	UCSTTIFG	UCALIFG
rw-0	r-0	rw-0	r-0	rw-0	rw-0	rw-0	rw-0

未被使用	位 7	未被使用。
UCSCLLOW	位 6	SCL 低电平 0 SCL 没被保持在低电平 1 SCL 被保持在低电平
UCGC	位 5	接收到常规调用地址。 当接收到一个起始条件时，UCGC 被自动清零。 0 没有接收到常规调用地址 1 接收到常规调用地址
UCBBUSY	位 4	总线忙 0 总线未激活 1 总线忙
UCNACKIFG	位 3	不应答接收到的中断标志。 当接收到一个起始条件时，UCNACKIFG 被自动清零。 0 无中断等待 1 中断等待
UCSTPIFG	位 2	停止条件中断标志。 当接收到一个起始条件时，UCSTPIFG 被自动清零。 0 无中断等待 1 中断等待
UCSTTIFG	位 1	起始条件中断标志。 当接收到一个停止条件时，UCSTTIFG 被自动清零。 0 无中断等待 1 中断等待
UCALIFG	位 0	仲裁失效中断标志 0 无中断等待 1 中断等待

17.4.6 UCBxRXBUF，USCI_Bx 接收缓冲寄存器

7	6	5	4	3	2	1	0
UCRXBUFx							
r	r	r	r	r	r	r	r

UCRXBUFx	位 7-0	接收数据缓冲是用户可以访问的，并包含从接收移位寄存器那里最后接收到的字符。 读取 UCBxRXBUF 将复位 UCBxRXIFG。

17.4.7 UCBxTXBUF，USCI_Bx 发送缓冲寄存器

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

UCTXBUFx	位 7-0	发送数据缓冲是用户可以访问的，并包含等待被移入发送移位寄存器的数据，并在移入之后将被发送。 写入发送数据缓冲将会清除 UCBxTXIFG。

17.4.8 UCBxI2COA，USCIBx I²C 本地地址寄存器

15	14	13	12	11	10	9	8
UCGCEN	**0**	**0**	**0**	**0**	**0**	**I2COAx**	
rw-0	r0	r0	r0	r0	r0	rw-0	rw-0
7	6	5	4	3	2	1	0
I2COAx							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

UCGCEN 位 15 通用常规地址使能
- 0 不响应一个常规调用
- 1 响应一个常规调用

I2COAx 位 9-0 I²C 自身地址。 I2COAx 位包含 USCI_Bx I²C 控制器的本地地址。 该地址是右对齐的。 在 7 位寻址模式中，位 6 是 MSB，位 9-7 被忽略。 在 10 位寻址模式中，位 9 是 MSB。

17.4.9 UCBxI2CSA，USCI_Bx I²C 从器件地址寄存器

15	14	13	12	11	10	9	8
0	**0**	**0**	**0**	**0**	**0**	**I2CSAx**	
r0	r0	r0	r0	r0	r0	rw-0	rw-0
7	6	5	4	3	2	1	0
I2CSAx							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

I2CSAx 位 9-0 I²C 从器件地址。 I2CSAx 位包含了由 USCI_Bx 模块寻址的外部器件的从器件地址。 它仅用于主控模式。 该地址是右对齐的。 在 7 位从器件寻址模式中，位 6 是 MSB，位 9-7 被忽略。 在 10 位从器件寻址模式中，位 9 是 MSB。

17.4.10 UCBxI2CIE，USCI_Bx I²C 中断使能寄存器

7	6	5	4	3	2	1	0
被保留				**UCNACKIE**	**UCSTPIE**	**UCSTTIE**	**UCALIE**
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

被保留 位 7-4 被保留

UCNACKIE 位 3 不应答中断使能
- 0 中断被禁用
- 1 中断被启用

UCSTPIE 位 2 停止条件中断使能
- 0 中断被禁用
- 1 中断被启用

UCSTTIE 位 1 起始条件中断使能
- 0 中断被禁用
- 1 中断被启用

UCALIE 位 0 仲裁丢失中断使能
- 0 中断被禁用
- 1 中断被启用

17.4.11 IE2，中断使能寄存器 2

7	6	5	4	3	2	1	0
				UCB0TXIE	**UCB0RXIE**		
				rw-0	rw-0		

	位 7-4	这些位可以用于其他模块（请参阅器件专用数据表）。
UCB0TXIE	位 3	USCI_B0 发送中断使能
		0 中断被禁用
		1 中断被启用
UCB0RXIE	位 2	USCI_B0 接收中断使能
		0 中断被禁用
		1 中断被启用
	位 1-0	这些位可以用于其他模块（请参阅器件专用数据表）。

17.4.12 IFG2，中断标志寄存器 2

7	6	5	4	3	2	1	0
				UCB0TXIFG	**UCB0RXIFG**		
				rw-1	rw-0		

	位 7-4	这些位可以被用于其他模块（请参阅特定器件专用数据表）。
UCB0TXIFG	位 3	USCI_B0 发送中断标志。 当 UCB0TXBUF 为空时，UCB0TXIFG 被置位。
		0 无中断等待
		1 中断等待
UCB0RXIFG	位 2	USCI_B0 接收中断标志。 当 UCB0RXBUF 收到一个完整字符时，UCB0RXIFG 被置位。
		0 无中断等待
		1 中断等待
	位 1-0	这些位可以被用于其他模块（请参阅特定器件专用数据表）。

17.4.13 UC1IE，USCI_B1 中断使能寄存器

7	6	5	4	3	2	1	0
未被使用				**UCB1TXIE**	**UCB1RXIE**		
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0		

未被使用	位 7-4	未被使用
UCB1TXIE	位 3	USCI_B1 发送中断启用
		0 中断被禁用
		1 中断被启用
UCB1RXIE	位 2	USCI_B1 接收中断使能
		0 中断被禁用
		1 中断被启用
	位 1-0	这些位可以被其他的 USCI 模块使用（请参阅器件专用数据表）。

17.4.14 UC1IFG，USCI_B1 中断标志寄存器

7	6	5	4	3	2	1	0
未被使用				**UCB1TXIFG**	**UCB1RXIFG**		
rw-0	rw-0	rw-0	rw-0	rw-1	rw-0		

未被使用	位 7-4	未被使用。
UCB1TXIFG	位 3	USCI_B1 发送中断标志. 当 UCB1TXBUF 为空时，UCB1TXIFG 被置位。 0 无中断等待 1 中断等待
UCB1RXIFG	位 2	USCI_B1 接收中断标志。 当 UCB1RXBUF 收到一个完整字符时，UCB1RXIFG 被置位。 0 无中断等待 1 中断等待
	位 1-0	这些位可被用于其它模块（请参阅器件专用数据表）。

USART 外设接口，USART 模式

通用同步/异步接收/发送器 (USART) 外设接口支持在同一硬件模块中的两个串行模式。 本章讨论异步 UART 模式的运行。 MSP430AFE2xx 器件上应用了 USART0。

Topic **Page**

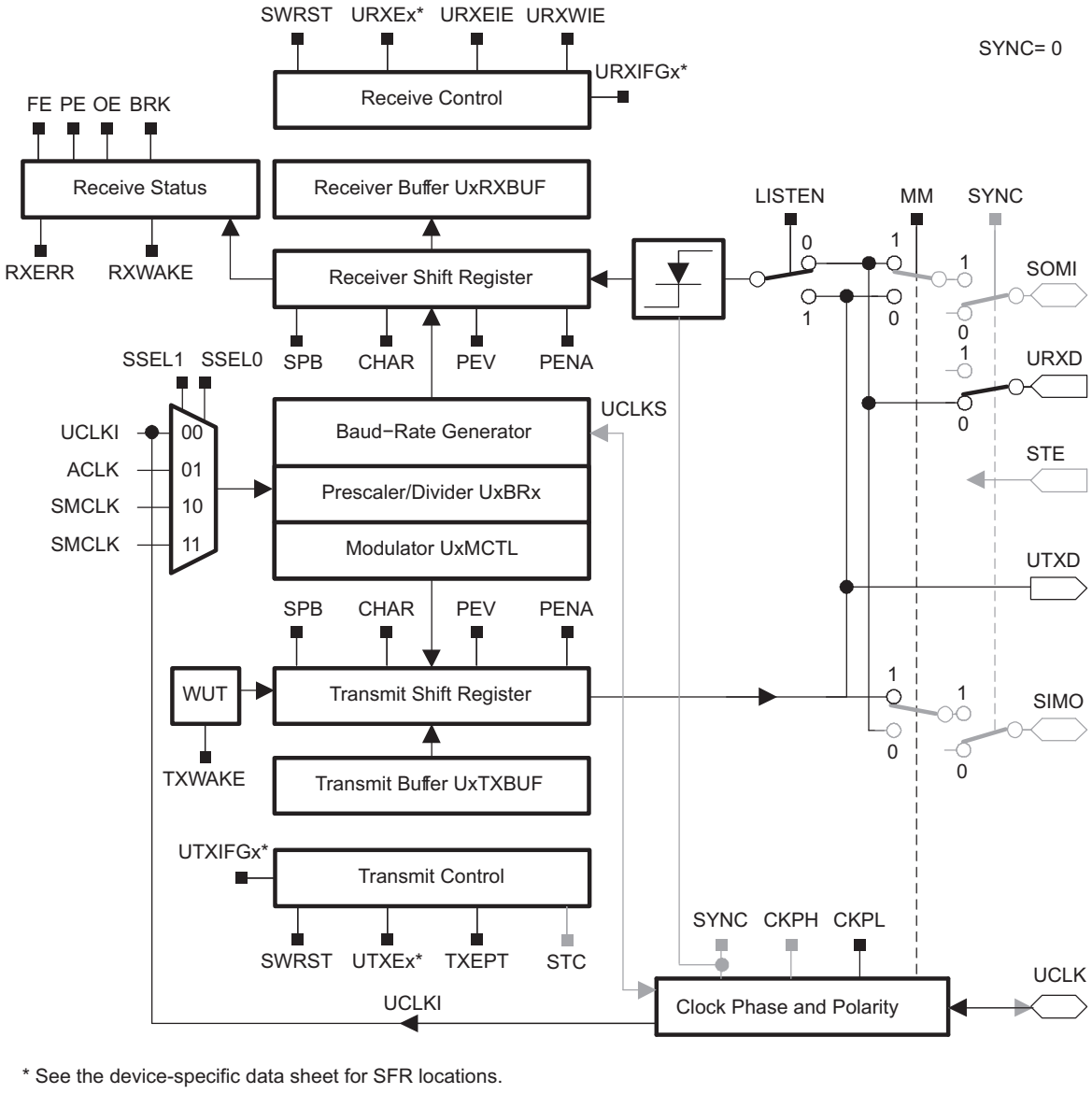
18.1 USART 介绍：USART 模式

在异步模式中，USART 通过两个外部引脚，URXD 和 UTXD，把 MSP430 连接到外部系统。 当 SYNC 位被清零时 UART 模式被选用。

UART 模式的特性包括：

- 7 或 8 位的奇，偶，或无校验数据
- 独立的发送和接收转换寄存器
- 单独的发送和接收缓存寄存器
- LSB 第一数据发送或接收
- 多处理器系统中内置空闲线和地址位通信
- 接收器开始边沿检测以从 LMPx 模式中自动唤醒
- 通过调制可编程波特率来支持分数位的波特率
- 错误检测和抑制及地址检测的状态标志
- 独立的接收中断和发送中断功能

图 18-1显示配置 UART 模式时的 USART。



* See the device-specific data sheet for SFR locations.

图 **18-1. USART** 方框图：**UART** 模式

18.2 USART 运行：UART 模式

在 UART 模式下，USART 的发送和接收字符以一个比特率异步传输到另一个设备。 每个字符的时序都是根据选定的 USART 的波特率来定的。 发送和接收功能使用相同的波特率频率。

18.2.1 USART 初始化和复位

USART 由 PUC 或通过置位 SWRST 位进行复位。 一个 PUC 后，SWRST 位会自动置位，以此来保持 USART 在复位状态。 当置位时，SWRST 位复位 URXIEx，UTXIEx，URXIFGx，RXWAKE，TXWAKE，RXERR，BRK，PE，O，和 FE 位并且置位 UTXIFGx 和 TXEPT 位。 接收和发送使能标志，URXEx 和 UTXEx，不会被 SWRST 改变。 为了运行，清除 SWRST 位释放 USART。 对于 USART0， 当从 I^2C 模式到 UART 模式重新配置时，也可参阅章节*USART 模块，I^2C 模式*。

注： **初始化或重新配置 USART 模块**

初始化/重新配置 USART 需要的过程如下：

1. 置位 SWRST (BIS.B #SWRST，&UxCTL)
2. UCSWRST=1 时初始化所有的 USART 寄存器（包括UCAXCTL）
3. 通过 MExSFRs (URXEx 和/或 UTXEx)使能 USART 模块
4. 通过软件(BIC.B #SWRST，&UxCTL)清除 SWRST
5. 通过 IExSFR（URXIEx 和/或 UTXIEx）启用中断（可选）

如果不按照这个过程，可能会导致不可预知的 USART 行为。

18.2.2 字符格式

UART 的字符格式展示在图 18-2中，包括一个开始位，7 或 8 个数据位，一个奇/偶/无校验位，一个地址位（地址位模式），和一个或两个停止位。 由选定的时钟源和波特率寄存器的设置来定义位周期。

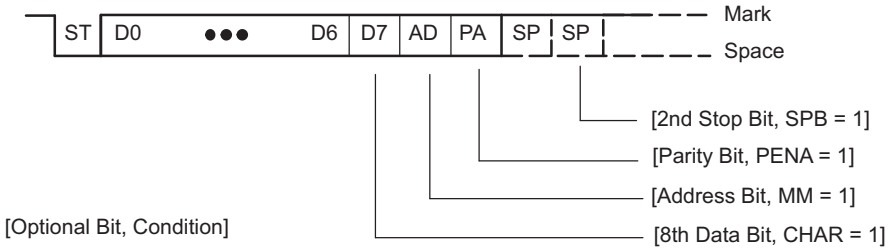


图 18-2. 字符格式

18.2.3 异步通信格式

当两个器件异步通信时，协议中需要用到空闲线格式。 当三个或更多的器件通信时，USART 支持空闲线和地址位多处理器通信格式。

18.2.3.1 空闲线多处理器格式

当 MM=0 时，空闲线多处理器格式将会被选用。 数据块在发送和接收线上被一段空闲时间隔开，如图 18-3所示。 当 10 个或多个持续标志在第一个字符的停止位之后被接收到时，一条空闲接收线被监测。 当两个停止位被用于空闲线时，第二个停止位被记为空闲时段的第一个标记位。

一段空闲时段后接收到的第一个字符为一个地址字符。 RXWAKE 位被用来作为针对每个字符块的地址标志。 在空闲线多处理器格式中，当接收到的字符是一个地址并被传输到 UxRXBUF 时，该位被置位。

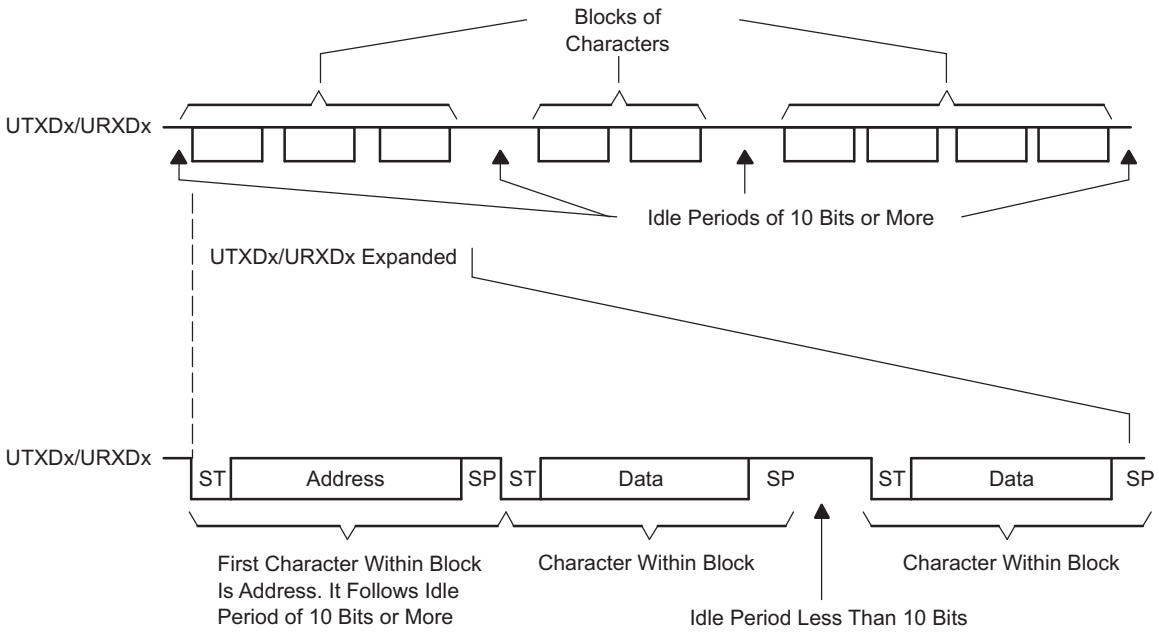


图 **18-3.** 空闲线路格式

在空闲线路多处理器格式中 URXWIE 位被用来控制数据接收。 当 URXWIE=1 时，所有的非地址字符被组装但不会传输到 UxRXBUF 中并且不会产生中断。 当一个地址字符被接收时，接收器被暂时激活并将这个字符传输到 UxRXBUF 中，同时 URXIFGx 中断标志被置位。 任何适用的错误标志也被置位。 然后，用户可以验证所接收到的地址。

如果接收到一个地址，用户软件可以验证该地址，并且必须复位 URXWIE 以继续接收数据。 如果 URXWIE 仍然置位，那么只有地址字符才能被接收。 URXWIE 位不会被 USART 硬件自动修改。

在空闲线多处理器格式中进行地址传输时，为了在 UTXDx 上产生地址字符标识符，可以由 USART 产生一个精确的空闲周期。 唤醒暂时 (WUT) 标志是一个带有用户可访问 TXWAKE 位的双缓冲内部标志。 当发送器从 UxTXBUF 中装载时，WUT 还从 TXWAKE 复位 TXWAKE 位中装载。

下列程序发送一个空闲帧以标明一个地址字符：

1. 置位 TXWAKE，然后向 UxTXBUF 中写入任何字符。 UxTXBUF 必须准备发送新数据 (UxTXIFG=1)。

   当移位寄存器准备发送新数据时，TXWAKE 的值被转移到 WUT 并且 UxTXBUF 的内容被转移到发送移位寄存器中。 它置位 WUT，抑制正常传输的起始位，数据位和奇偶校验位，然后发送正好为 11 位的空闲周期。 当空闲线中用有两个停止位时，第二个停止位作为空闲时段的第一个标记位计数。 TXWAKE 自动复位。
2. 向 UxTXBUF 写入所需的地址字符。 UxTXBUF 必须准备发送新数据 (UxTXIFG= 1)。

   在 UTXDx 上随着地址识别空闲时段后，新字符所代表的特定地址被移出。 向 UxTXBUF 写入的第一个“无影响”字符对于移出 TXWAKE 位到 WUT 和产生一个空闲线状态是必要的。 该数据将被丢弃，并不会出现在 UTXDx 上。

18.2.3.2 地址位多处理器格式

当 MM=1 时，地址位多处理器格式将会被选用。 如图 18-4所示，每个处理过的字符都包含一个用作地址指示的额外位。 字符块的第一个字符带有一个设置指示字符地址的地址位。 当接收到的字符是一个有效的地址字符并被传输到 UxRXBUF 时，USART RXWAKE 位被置位。

在地址位多处理器格式中 URXWIE 位被用来控制数据接收。 如果 URXWIE 被置位，数据字符（地址位=0）将由接收器组装，但不会传输到 UxRXBUF 而且不会产生中断。 当包含一组地址位的一个字符被接收时，接收器被暂时激活，并将字符传输至 UxRXBUF 而且将 URXIFGx 置位。 所有的适用错误标志也被置位。

如果接收到一个地址，用户软件必须复位 URXWIE 以继续接收数据。 如果 URXWIE 仍然置位，那么只有地址字符（地址位=1）才能被接收。 URXWIE 位不会被 USART 硬件自动修改。

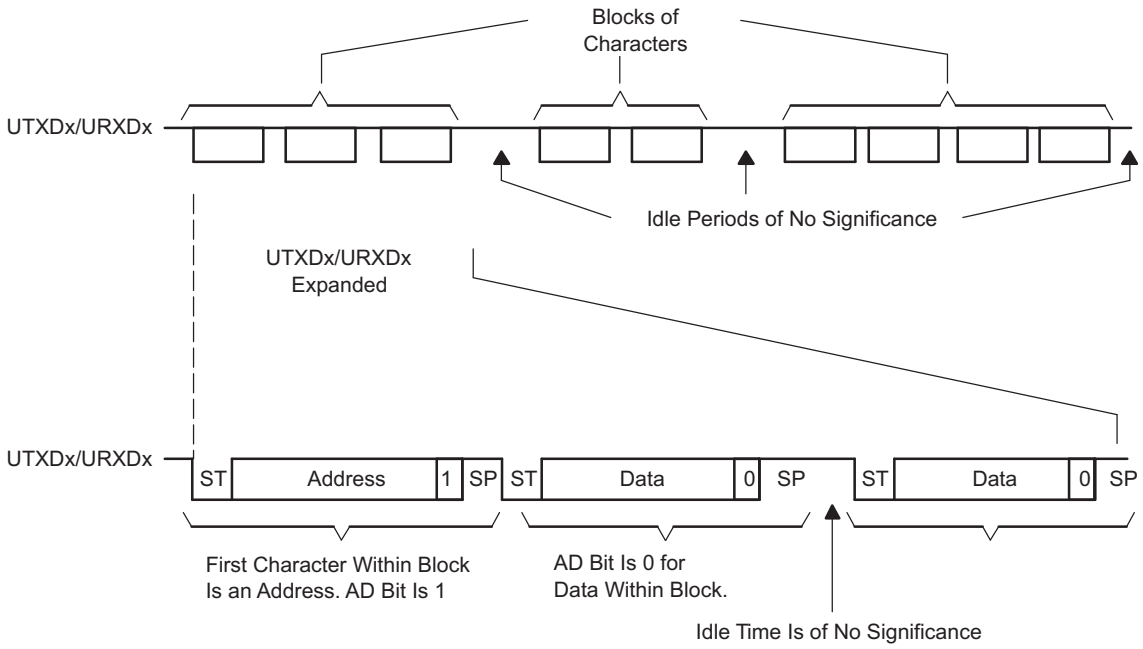


图 **18-4.** 地址位多处理器格式

对于在地址位多处理器模式中的地址传输，一个字符的地址位可以通过写入 TXWAKE 位来控制。 TXWAKE 位的值被装入字符的地址位从 UxTXBUF 转移到发送移位寄存器中，自动清除 TXWAKE 位。 TXWAKE 一定不能由软件清除。 在它被传输到 WUT 后由 USART 硬件或通过置位 SWRST 被清除。

18.2.3.3 自动错误检测

干扰抑制防止 USART 被意外启动。 URXDx 上任何低于抗尖峰脉冲的时间 t_τ（约 300ns）的低电平都将被忽略。 对于参数请参阅《器件专用数据表》。

在 URXDx 上，当一个低电平周期超过 t_τ时，对开始位将采取多数表决的方法检测。 如果多数表决未能检测到一个有效的起始位，则 USART 暂停字符接收并等待下一个 URXDx 上的低电平周期。 多数表决也可用于字符中的每个位，以防止位错误。

接收字符时，USART 模块将自动检测帧错误，奇偶校验错误，溢出错误，和中断条件。 当他们相应的条件被检测到时，FE，PE，OE，和 BRK 都将被置位。 当这些错误标志置位时，RXERR 也将置位。 错误条件在表 18-1中做出描述。

表 18-1. 接收错误条件

错误条件	说明
组帧错误	当一个组帧错误发生时，一个低低电平停止位被检测到。 当使用两个停止位时，只有第一个停止位被进行组帧错误检查。 当检测一个组帧错误时，FE 位被置位。
奇偶校验错误	奇偶校验错误也就是字符中 1s 的数量和奇偶校验位的值不匹配。 当一个地址位包含于字符时，它同时也被包含进奇偶校验计算中。 当一次奇偶校验错误被监测到时，PE 位置位。
接收溢出错误	在读取前一个字符之前另一个字符被装载到 UxRXBUF 中会引发一次溢出错误。 当溢出错误发生时，OE 位置位
中断状态	中断状态是 URXDx 上 10 个或更多的低位在丢失的停止位后收到的周期。 当检测到一个中断条件时，BRK 位被置位。 当 URXEIE=0 时，一个中断状态也可以置位中断标志 URXIFGx。

当 URXEIE=0 并且检测到一个帧错误，奇偶校验错误，或中断状态时，UxRXBUF 将不再接收字符。 当 URXEIE=1 时，UxRXBUF 开始接收字符并且所有的合适错误位都将被置位。

当 FE，PE，OE，BRK 或 RXEER 置位时，其状态保持到用户软件复位它或 UxRXBUF 中的数据被读出。

18.2.4 USART 接收使能

接收使能位，URXEx，使能或禁用 URXDx 上的数据接收如图 18-5中所示。 在当前任何字符接受完成后或在无接收操作是有效的后，立即禁用 USART 接收来停止接收运行。 接收数据缓冲器，UxRXBUF，包括在字符被接收后从 RX 移送寄存器中的移动。

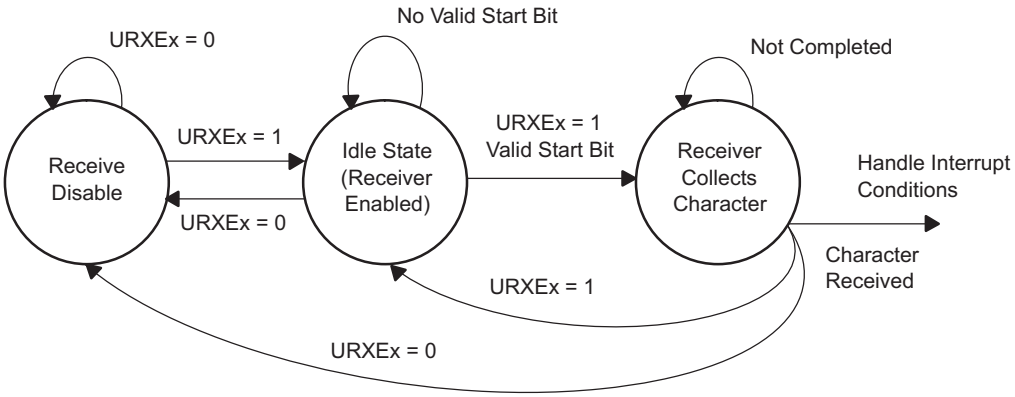


图 18-5. 接收器使能状态图

注: 重新使能接收器（置位 URXEx）：UART 模式

当接收器被禁用 (URXEx=0)时，重新使能接收器 (URXEx=1) 使其与当时出现在 URXDx 上的任何数据流异步。 接收到一个有效字符（请见 URXWIE）之前，可以进行同步测试空闲状态。

18.2.5 USART 发送使能

当 UTXEx 被置位时，UART 发送器被使能。 通过将数据写入到 UxTXBUF 启动传输。 然后，TX 移位寄存器空后，这些数据被转移到下一个 BITCLK 上的发送移位寄存器中，并开始传输。 图 18-6显示了这一过程。

当 UTXEx 位被复位时，发送器被停止。 在清除 UTXEx 前，目前在发送移位寄存器中的任何数据被移动到 UxTXBUF 且任何活跃的的数据传输都将继续，直到完成所有的数据传输。

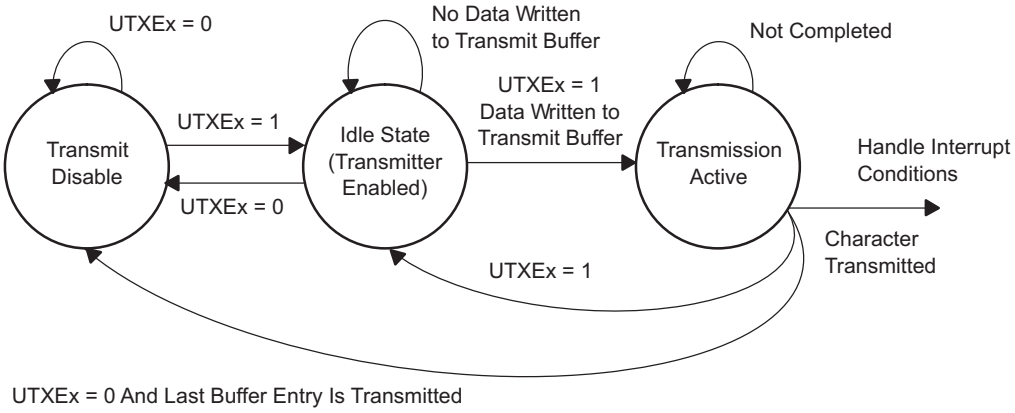


图 **18-6.** 发送器使能状态图

当发送器被使能 (UTXEx=1)，除非它已经准备好由 UTXIFGx=1 指明发送新数据，否则数据一定不要写入 UxTXBUF。 如果在 UxTXBUF 中数据被被移动到 TX 移位寄存器时被修改，那么冲突可能会导致一个错误的传输。

建议在完成所有已激活的传输后，再禁用 (UTXEx=0) 该发送器。 这是由一组发送器空位 (TXEPT=1)表示的。 在发送器被禁用期间写入 UxTXBUF 的任何数据都被保留在缓冲区中，但不会被移动到发送移位寄存器或被发送。 一旦 UTXEx 置位，则在发送缓冲区中的数据将被立即装载到发送移位寄存器中并且字符发送重新开始。

18.2.6 USART 波特率生成

USART 波特率发生器，能够从非标准源频率中产生标准的波特率。 波特率发生器使用一个预分频器/除法器和一个调制器，如图 18-7所示。 这样的组合支持了波特率产生小数约数。 最大 USART 的波特率是 UART 源时钟频率 BRCLK 的三分之一。

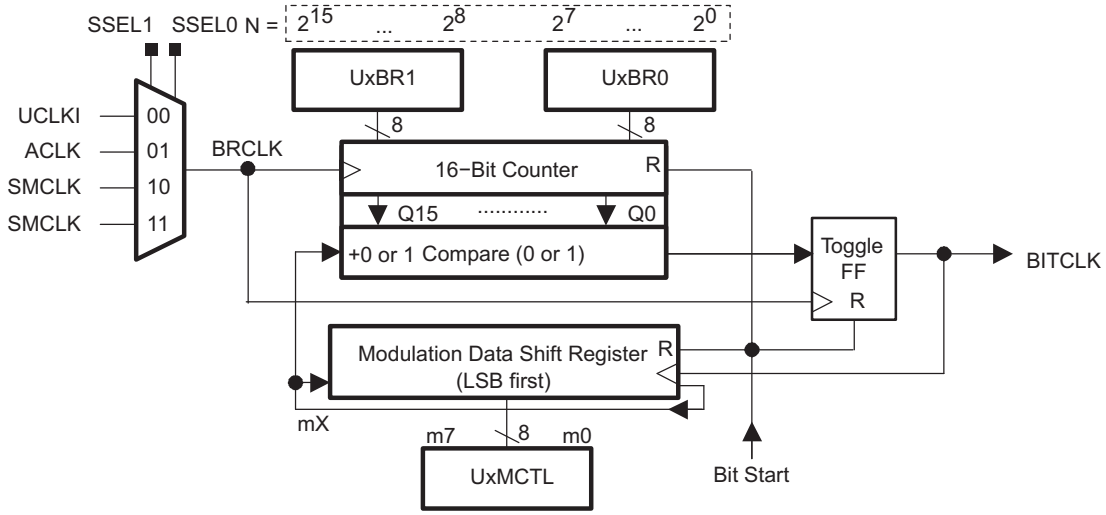


图 **18-7. MSP430** 波特率发生器

每个位的时序如图 18-8所示。 对于每一接收的位，采取多数表决的方法来确定该位的值。 在 N/2-1，N/2，和 N/2+1 BRCLK 周期中采样，其中 N 是每个 BITCLK 中 BRCLKs 的数量。

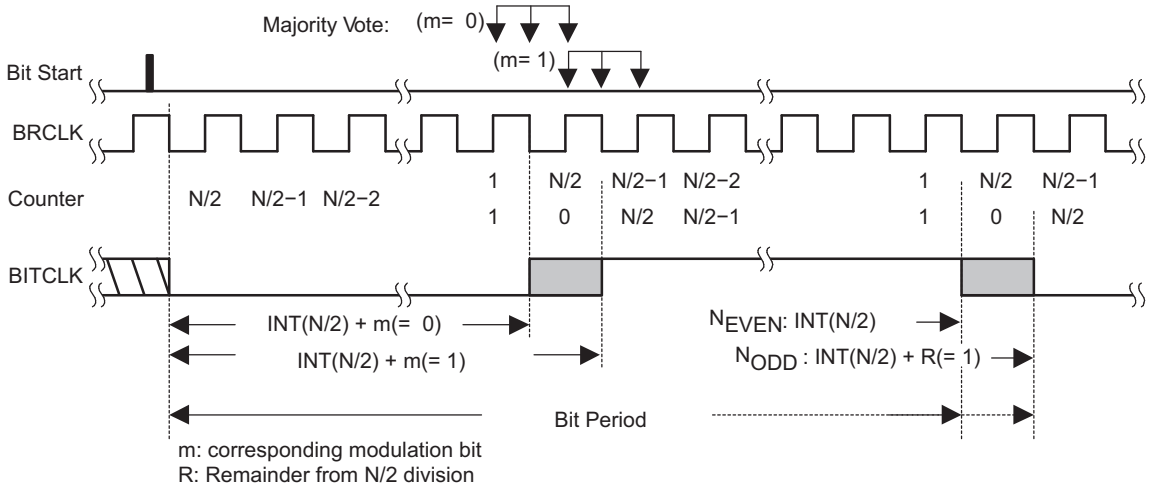


图 **18-8. BITCLK** 波特率时序

18.2.6.1 波特率位时序

波特率发生器的第一阶段是 16 位的计数器和比较器。 在发送或接收的每一位的开始，计数器都用 INT (N/2) 装载，其中 N 是存储的在 UxBR0 和 UxBR1 中的结合值。 为每个位周期的半周期计数器重新加载 INT (N/2)，给出 N BRCLKs 的一个总位周期。 对于一个给定的 BRCLK 时钟源，使用的波特率决定了需要的分频系数 N：

$$N = \frac{BRCLK}{Baud\ Rate}$$

分频系数 N 通常是一个非整数的值，它的整数部分可以由预分频器/除法器实现。 波特率发生器的第二阶段，调制器，被用于尽可能地满足小数部分。 然后系数*N*被定义为：

$$N = UxBR + \frac{1}{n}\sum_{i=0}^{n-1} m_i$$

其中，

N = 目标分频系数

UxBR = 寄存器 UxBR0 和 UxBR1 的16 位表示

i = 字符中的比特位置

n = 字符中的总位数

m_i= 每个对应的调制位的数据（1 或 0）

$$Baud\ rate = \frac{BRCLK}{N} + \frac{BRCLK}{UxBR + \frac{1}{n}\sum_{i=0}^{n-1} m_i}$$

当需要一个非整数分频时，BITCLK 可以通过调制器进行位到位的调整，以此来满足时序要求。 如果调制器位 m_i被置位，那么每个位的时序都将由一个 BRCLK 时钟周期进行扩展。 每当一个位被接收或发送时，在调制控制寄存器中的下一个位将决定该位的时序。 一个置位的调制位通过 1 来增加分频系数，而一个清除的调制位通过给定的 UxBR 来保持分频系数。

起始位的时序由 UxBR 加上 m0 来确定，下一位由UxBR 加上 m1 来确定，等等。 调制序列开始于最低有效位 (LSB)。 当字符大于 8 位时，调制序列用 m0 重新开始并继续进行直到所有的位都被处理。

18.2.6.2 决定调制值

决定调制值是一个互动的过程。 使用提供的时序误差公式，在起始位开始，单独的比特误差通过对应的调制位的设置和清除来计算。 使用较低误差置位的调制位被选用并且下一个比特错误被计算。。 持续这个过程，直到所有的位误差被最小化。 当一个字符包含多于 8 位时，调制比特重复进行。 例如，字符的第九位使用调制比特 0。

18.2.6.3 发送位时序

每个字符的时序是单独的位时序的总和。 通过调制每个位来减少累积的位误差。 单个位误差可以通过以下来计算：

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

其中，

波特率=期望的波特率

BRCLK=输入频率 - UCLKI，ACLK，或 SMCLK

j = 比特位置 - 0（起始位时），-1（数据位 D0 时），等等

UxBR = 寄存器 UxBR1 和 UxBR0 中的分频系数

例如，满足下列条件的发送误差的计算方法：

波特率 = 2400

BRCLK=32768Hz (ACLK)

UxBR=13，因为理想的分频系数是 13.65

UxMCTL=6Bh：m7=0，m6=1，m5=1，m4=0，m3=1，m2=0，m1=1，和 m0=1。 UxMCTL 的 LSB 被首先使用。

$$\text{Start bit Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \cdot ((0+1) \cdot \text{UxBR} + 1) - 1\right) \cdot 100\% = 2.54\%$$

$$\text{Data bit D0 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \cdot ((1+1) \cdot \text{UxBR} + 2) - 2\right) \cdot 100\% = 5.08\%$$

$$\text{Data bit D1 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \cdot ((2+1) \cdot \text{UxBR} + 2) - 3\right) \cdot 100\% = 0.29\%$$

$$\text{Data bit D2 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \cdot ((3+1) \cdot \text{UxBR} + 3) - 4\right) \cdot 100\% = 2.83\%$$

$$\text{Data bit D3 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \cdot ((4+1) \cdot \text{UxBR} + 3) - 5\right) \cdot 100\% = -1.95\%$$

$$\text{Data bit D4 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \cdot ((5+1) \cdot \text{UxBR} + 4) - 6\right) \cdot 100\% = 0.59\%$$

$$\text{Data bit D5 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \cdot ((6+1) \cdot \text{UxBR} + 5) - 7\right) \cdot 100\% = 3.13\%$$

$$\text{Data bit D6 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \cdot ((7+1) \cdot \text{UxBR} + 5) - 8\right) \cdot 100\% = -1.66\%$$

$$\text{Data bit D7 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \cdot ((8+1) \cdot \text{UxBR} + 6) - 9\right) \cdot 100\% = 0.88\%$$

$$\text{Parity bit Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \cdot ((9+1) \cdot \text{UxBR} + 7) - 10\right) \cdot 100\% = 3.42\%$$

$$\text{Stop bit 1 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \cdot ((10+1) \cdot \text{UxBR} + 7) - 11\right) \cdot 100\% = -1.37\%$$

结果显示每比特的误差最大为 BITCLK 周期的 5.08%。

18.2.6.4 接收位时序

接收时序有两个错误来源。 第一个是比特到比特的时序误差。 第二个是发生的起始沿和被 USART 接受的起始沿之间的误差。图 18-9显示了 URXDx 引脚上数据和内部波特率时钟之间的异步时序误差。

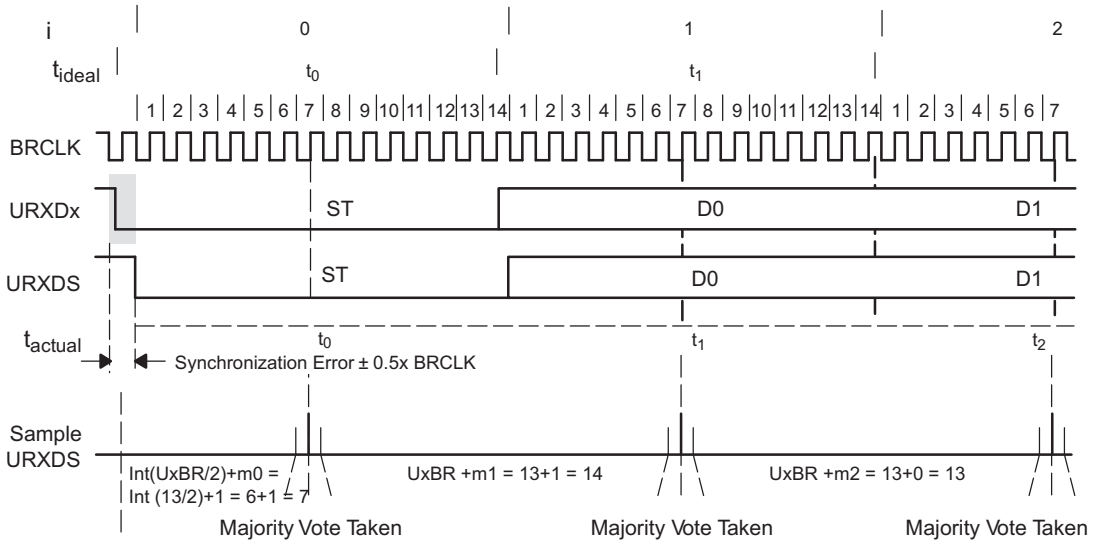


图 18-9. 接收错误

理想的起始位时序 $t_{理想\ (0)}$是波特率时序 $t_{波特率}$的一半，因为该位是在它周期的中间开始被检测的。 对于剩余的字符位，理想的波特率时序 $t_{理想(i)}$等于波特率时序 $t_{波特率}$ 单个位误差可以通过以下来计算：

$$\text{Error [\%]} = \left\{ \frac{\text{baud rate}}{\text{BRCLK}} \times \left(2 \times \left[m0 + \text{int}\left(\frac{\text{UxBR}}{2} \right) \right] + \left[i \times \text{UxBR} + \sum_{i=1}^{j} m_i \right] \right) - 1 - j \right\} \times 100\%$$

其中，

波特率=所需的波特率

BRCLK = 输入频率；UCLK，ACLK，或 SMCLK 的选择

j=0（起始位时），1（数据位 D0 时），等

UxBR=寄存器 UxBR1 和 UxBR0 中的分频系数

例如，满足下列条件的接收误差的计算方法：

波特率 = 2400

BRCLK=32768Hz (ACLK)

UxBR=13，因为理想的分频系数是 13.65

UxMCTL=6B：m7 = 0，m6=1，m5=1，m4=0，m3=1，m2=0，m1=1 和 m0=1。 UxMCTL 的 LSB 被首先使用。

$$\text{Data bit D1 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \cdot [2x(1+6)+2 \cdot \text{UxBR}+1]-1-2\right) \cdot 100\% = 0.29\%$$

$$\text{Data bit D2 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \cdot [2x(1+6)+3 \cdot \text{UxBR}+2]-1-3\right) \cdot 100\% = 2.83\%$$

$$\text{Data bit D3 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \cdot [2x(1+6)+4 \cdot \text{UxBR}+2]-1-4\right) \cdot 100\% = -1.95\%$$

$$\text{Data bit D4 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \cdot [2x(1+6)+5 \cdot \text{UxBR}+3]-1-5\right) \cdot 100\% = 0.59\%$$

$$\text{Data bit D5 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \cdot [2x(1+6)+6 \cdot \text{UxBR}+4]-1-6\right) \cdot 100\% = 3.13\%$$

$$\text{Data bit D6 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \cdot [2x(1+6)+7 \cdot \text{UxBR}+4]-1-7\right) \cdot 100\% = -1.66\%$$

$$\text{Data bit D7 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \cdot [2x(1+6)+8 \cdot \text{UxBR}+5]-1-8\right) \cdot 100\% = 0.88\%$$

$$\text{Parity bit Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \cdot [2x(1+6)+9 \cdot \text{UxBR}+6]-1-9\right) \cdot 100\% = 3.42\%$$

$$\text{Stop bit 1 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \cdot [2x(1+6)+10 \cdot \text{UxBR}+6]-1-10\right) \cdot 100\% = -1.37\%$$

$$\text{Start bit Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \cdot [2x(1+6)+0 \cdot \text{UxBR}+0]-1-0\right) \cdot 100\% = 2.54\%$$

$$\text{Data bit D0 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \cdot [2x(1+6)+1 \cdot \text{UxBR}+1]-1-1\right) \cdot 100\% = 5.08\%$$

结果显示每比特的误差最大为 BITCLK 周期的 5.08%。

18.2.6.5 典型的波特率和误差

对于一个 32 768-Hz 晶振 (ACLK) 和一个典型的 1 048 576-Hz SMCLK，UxBRx 和 UxMCTL 的标准波特率频率在表 18-2中列出。

接收误差是在每一个比特中间相对于理想扫描时间的累积时间。 发送误差是相对于位周期理想时间的累积时序误差。

表 18-2. 常用的波特率，波特率数据和误差

波特率	除以		A：BRCLK=32 768Hz						B：BRCLK=1 048 576Hz				
	A:	B:	UxBR1	UxBR0	UxMCTL	最大 TX 误差 %	最大 RX 误差 %	同步 RX 误差 %	UxBR1	UxBR0	UxMCTL	最大 TX 误差 %	最大 RX 误差 %
1200	27.31	873.81	0	1B	03	-4/3	-4/3	±2	03	69	FF	0/0.3	±2
2400	13.65	436.91	0	0D	6B	-6/3	-6/3	±4	01	B4	FF	0/0.3	±2
4800	6.83	218.45	0	06	6F	-9/11	-9/11	±7	0	DA	55	0/0.4	±2
9600	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		9.1							0	09	08	-5/7	±7

18.2.7 USART 中断

USART 有一个发送中断向量和接收中断向量。

18.2.7.1 USART 发送中断操作

UTXIFGx 中断标志被发送器置位表示 UxTXBUF 准备接收另一个字符。 如果 UTXIE 和 GIE 也置位，将产生一个中断请求。 如果一个中断请求被服务或一个字符被写到 UxTXBUF 中 UTXIFG 将自动复位。

一个 PUC 或 SWRST=1 后，UTXIFGx 将置位。一个 PUC 或 SWRST=1 后，UTXIEx 将复位。运行如图 18-10所示。

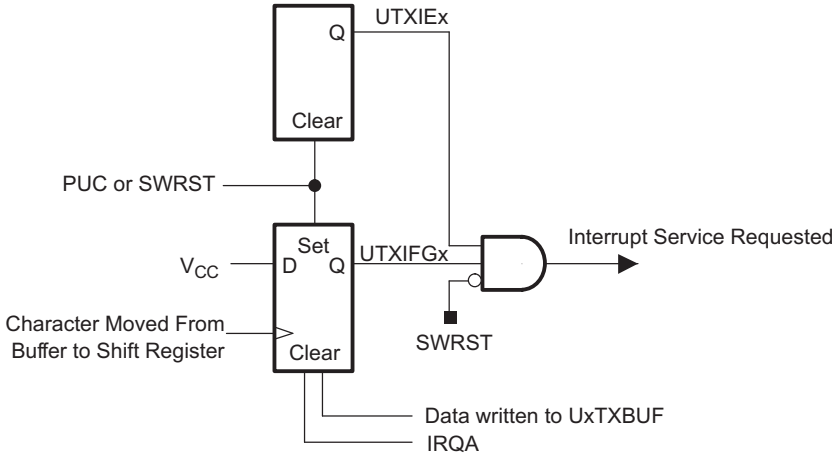


图 **18-10.** 发送中断操作

18.2.7.2 USART 接收中断操作

每次一个字符被接收并装载到 UxRXBUF 中去时 URXIFG 中断标志置位一次。 如果 URXIE 和 GIE 置位，将产生一个中断请求。 复位 PUC 信号或者 SWRST=1 时，URXIFG 和URXIE 将被系统复位。如果挂起中断被服务时（当 URXSE=0 时）或 当 UxRXBUF 被读取时 URXIFGx 将自动复位。 运行如图 18-11所示。

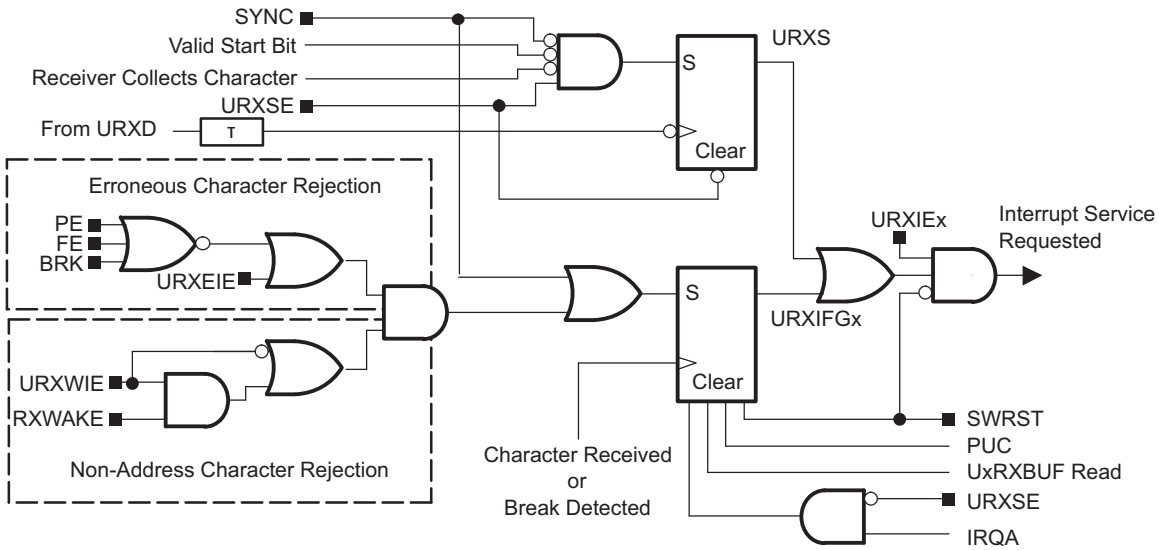


图 **18-11.** 接收中断操作

URXEIE 被用来使能或禁止错误字符置位 URXIFGx。 当使用多处理器寻址模式时，URXWIE 被用于自动检测有效的地址字符和拒绝不需要的数据字符。

有两种类型的字符不能置位 URXIFGx：

- 当 URXEIE=0 时，错误的字符
- 当 URXWIE=1 时，非地址字符

当 URXEIE=1 时，一个中断条件置位 BRK 位和URXIFGx 标志。

18.2.7.3 接收开始边沿检测操作

URXSE 位使能接收开始边缘检测功能。 推荐的接收开始边沿功能的用法是，BRCLK 由 DCO 做源并且关闭 DCO，因为这是低功耗模式操作。 DCO 的超高速打开使字符能在开始边沿检测后被接受。

当 URXSE，URXIEx 和 GIE 置位并且在 URXDx 上出现开始沿时，内部信号 URXS 被置位。 当 URXS 置位时，产生接收中断请求但 URXIFGx 没有置位。 在接收中断服务程序中的用户软件可以测试 URXIFGx 来确定中断源。 当 URXIFGx=0 时，检测到起始沿，当 URXIFGx=1 时，收到一个有效的字符（或中断）。

当 ISR 确定中断请求是从一个起始边缘发出时，用户软件切换 URXSE 从 ISR 返回到主动模式或者一个源是活动的低功率模式，并且必须使能 BRCLK 源。 如果 ISR 返回到 BRCLK 源无效的低功耗模式，将无法接收字符。 切换 URXSE 清除 URXS 信号，并为以后的字符重新使能开始边缘检测功能。 更多关于进入和退出低功耗模式的信息请参阅《*系统复位，中断和操作模式*》章节。

现在活跃的的 BRCLK 允许 USART 接收平衡的字符。 完整的字符被接收并移动到 UxRXBUF 后，URXIFGx 置位并再次发出中断服务请求。 ISR 进入后，URXIFGx=1 表示一个字符被接收。 U当用户软件读取 UxRXBUF 时，RXIFGx 标志被清除。

```
; Interrupt handler for start condition and ; Character receive. BRCLK = DCO.U0RX_Int BIT.B
#URXIFG0,&IFG1 ; Test URXIFGx to determineJZ ST_COND ; If start or characterMOV.B &UxRXBUF,dst ;
Read buffer... ;RETI ;ST_COND BIC.B #URXSE,&U0TCTL ; Clear URXS signalBIS.B #URXSE,&U0TCTL ; Re-
enable edge detectBIC #SCG0+SCG1,0(SP) ; Enable BRCLK = DCORETI ;
```

注: 带有暂停的 UART 时钟中断检测

使用接收开始边缘检测功能时，如果 BRCLK 源是关闭的，则不能检测到中断状态。

18.2.7.4 接收开始边沿检测条件

当 URXSE =1 时，干扰抑制防止 USART 被意外启动。 URXDx 上任何低电平小于抗尖峰脉冲时间 t_τ（约 300ns）都将被 USART 忽略并且不会产生中断请求（请见图 18-12)。 对于参数请参阅《器件专用数据表》

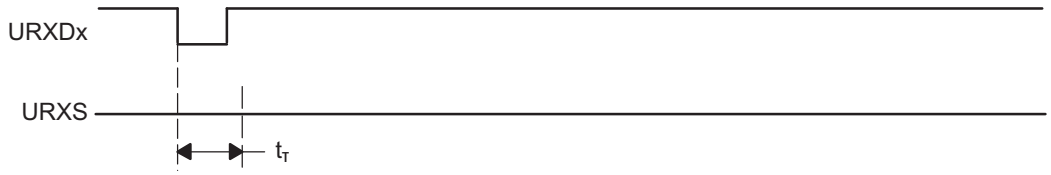


图 **18-12.** 干扰抑制，**USART** 接收未开始

当一个干扰大于 t_τ或在 URXDx 上出现一个有效开始位时，USART 接收操作被启动并且采取一次多数表决，如图 18-13所示。 如果多数表决未能检测到一个起始位，USART 暂停字符接收。

如果字符接收停止，则 BRCLK 无需处于激活状态。 一个超时周期大于字符的接收持续时间时，可以由软件使用它来表示在预期的时间内没有收到字符，并且该软件可以禁用 BRCLK。

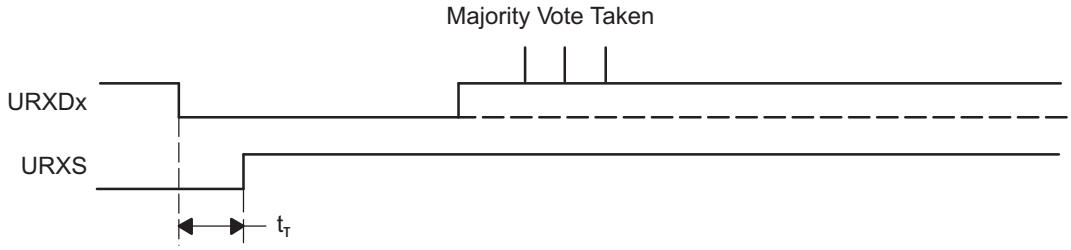


图 **18-13.** 干扰抑制，**USART** 激活

18.3 USART 寄存器：UART 模式

表 18-3列出了所有器件中，应用一个 USART 模块的寄存器。表 18-4只适用于带有一个第二 USART 模块的器件，USART1。

表 18-3. USART0 控制和状态寄存器

寄存器	简表	寄存器类型	地址	初态
USART 控制寄存器	U0CTL	读取/写入	070h	001h 与 PUC
发送控制寄存器	U0TCTL	读取/写入	071h	001h 与 PUC
接收控制寄存器	U0RCTL	读取/写入	072h	000h 与 PUC
调制控制寄存器	U0MCTL	读取/写入	073h	未改变
波特率控制寄存器 0	U0BR0	读取/写入	074h	未改变
波特率控制寄存器 1	U0BR1	读取/写入	075h	未改变
接收缓冲寄存器	U0RXBUF	读取	076h	未改变
发送缓冲寄存器	U0TXBUF	读取/写入	077h	未改变
SFR 模块使能寄存器 1	ME1	读取/写入	004h	000h 与 PUC
SFR 中断使能寄存器 1	IE1	读取/写入	000h	000h 与 PUC
SFR 中断标志寄存器 1	IFG1	读取/写入	002h	082h 与 PUC

表 18-4. USART1 控制和状态寄存器

寄存器	简式	寄存器类型	地址	初态
USART 控制寄存器	U1CTL	读取/写入	078h	001h 与 PUC
发送控制寄存器	U1TCTL	读取/写入	079h	001h 与 PUC
接收控制寄存器	U1RCTL	读取/写入	07Ah	000h 与 PUC
调制控制寄存器	U1MCTL	读取/写入	07Bh	未改变
波特率控制寄存器 0	U1BR0	读取/写入	07Ch	未改变
波特率控制寄存器 1	U1BR1	读取/写入	07Dh	未改变
接收缓冲寄存器	U1RXBUF	阅读	07Eh	未改变
发送缓冲寄存器	U1TXBUF	读取/写入	07Fh	未改变
SFR 模块使能寄存器 2	ME2	读取/写入	005h	000h 与 PUC
SFR 中断使能寄存器 2	IE2	读取/写入	001h	000h 与 PUC
SFR 中断标志寄存器 2	IFG2	读取/写入	003h	020h 与 PUC

注： 修改 **SFR** 位

为了避免修改其他模块的控制位，建议使用 BIS.B 或 BIC.B 指令置位或清除 IEX 和 IFGx 位，而非 MOV.B 或 CLR.B 指令。

18.3.1 UxCTL，USART 控制寄存器

7	6	5	4	3	2	1	0
PENA	**PEV**	**SPB**	**CHAR**	**LISTEN**	**SYNC**	**MM**	**SWRST**
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1

PENA 位 7 奇偶校验使能

0 奇偶校验被禁用

1 奇偶校验被启用。 生成 (UTXDx) 和预期 (URXDx) 的奇偶校验位。 在地址位多处理器模式中，地址位被包括在奇偶校验计算中。

PEV 位 6 奇偶校验选择。 奇偶校验被禁用时，PEV 不能使用。

0 奇数校验

1 偶数校验

SPB 位 5 停止位选择 发送的停止位个数。 接收器总是会检查一个停止位。

0 1 个停止位

1 2 个停止位

CHAR 位 4 字符长度。 选择 7 位或 8 位字符长度。

0 7 位数据

1 8 位数据

LISTEN 位 3 监听使能。 监听位选择回路模式。

0 被禁用

1 被启用。 UTXDx 被内部反馈到接收器。

SYNC 位 2 同步模式使能

0 UART 模式

1 SPI 模式

MM 位 1 多处理器模式选择

0 空闲线多处理器协议

1 地址位多处理器协议

SWRST 位 0 软件复位使能

0 被禁用。 USART 复位操作释放

1 被启用。 USART 逻辑保持在复位状态

18.3.2 UxTCTL，USART 发送控制寄存器

7	6	5	4	3	2	1	0
未使用	**CKPL**	**SSELx**		**URXSE**	**TXWAKE**	未使用	**TXEPT**
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1

未使用	位 7	未使用	
CKPL	位 6	时钟极性选择	
		0	UCLKI=UCLK
		1	UCLKI = 反相的 UCLK
SSELx	位 5-4	源选择。 这些位选择 BRCLK 时钟源。	
		00	UCLKI
		01	ACLK
		10	SMCLK
		11	SMCLK
URXSE	位 3	UART接收开始边沿。 该位使能 UART 接收开始边沿的功能。	
		0	被禁用
		1	被启用
TXWAKE	位 2	发送器唤醒	
		0	发送下一帧数据
		1	发送下一帧地址
未使用	位 1	未使用	
TXEPT	位 0	发送器空标志	
		0	UART 发送数据和/或数据在 UxTXBUF 中等待发送
		1	发送移位寄存器和 UxTXBUF 为空或 SWRST=1

18.3.3 UxRCTL，USART 接收控制寄存器

7	6	5	4	3	2	1	0
FE	**PE**	**OE**	**BRK**	**URXEIE**	**URXWIE**	**RXWAKE**	**RXERR**
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

FE 位 7 帧错误标志
- 0 无错误
- 1 接收到的低停止位字符

PE 位 6 奇偶校验错误标志。 当 PENA=0，PE 被读取为 0。
- 0 无错误
- 1 接收到带有奇偶校验错误的字符

OE 位 5 溢出错误标志。 当在之前的一个字符被读取前随后一个字符被传输到 UxRXBUF 时，该位被置位。
- 0 无错误
- 1 发生溢出错误

BRK 位 4 中断检测标志
- 0 无中断条件
- 1 发生的中断条件

URXEIE 位 3 接收错误的字符中断使能
- 0 错误字符被拒绝且 URXIFGx 没有被置位
- 1 接收到的错误字符置位 URXIFGx

URXWIE 位 2 接收唤醒中断使能。 当接收到一个地址字符时，此位启用 URXIFGx 进而置位。 当 URXEIE=0 时，如果接收有错误，则地址字符不再置位 URXIFGx。
- 0 所有接收到的字符置位 URXIFGx
- 1 只有接收到的地址字符置位 URXIFGx

RXWAKE 位 1 接收唤醒标志
- 0 接收到的字符为数据
- 1 接收到的字符是一个地址

RXERR 位 0 接收错误标志。 该位表示错误的收到一个字符。 当 RXERR=1 时，一个或更多的错误标志 (FE，PE，OE，BRK) 也被置位。 当 UxRXBUF 被读取时，RXERR 被清除。
- 0 没有检测到接收错误
- 1 检测到接收错误

18.3.4 UxBR0，USART 波特率控制寄存器 0

7	6	5	4	3	2	1	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

18.3.5 UxBR1，USART 波特率控制寄存器 1

7	6	5	4	3	2	1	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

UxBRx 有效的波特率控制范围为 3≤UxBR≤0FFFFh 之间，其中 UxBR=(UxBR1+ UxBR0)。 如果 UxBR<3，将会发生不可预知的接收和发送时序。

18.3.6 UxMCTL，USART 调制控制寄存器

7	6	5	4	3	2	1	0
m7	**m6**	**m5**	**m4**	**m3**	**m2**	**m1**	**m0**
rw	rw	rw	rw	rw	rw	rw	rw

UxMCTLx 调制比特。 这些位为 BRCLK 选择调制。

18.3.7 UxRXBUF，USART 接收缓冲寄存器

7	6	5	4	3	2	1	0
2^7	**2^6**	**2^5**	**2^4**	**2^3**	**2^2**	**2^1**	**2^0**
r	r	r	r	r	r	r	r

UxRXBUFx 位 7-0 接收数据缓冲区是用户可以访问的并且包含最后从接收移位寄存器中接收到的字符。 读取 UxRXBUF 复位接收错误位，RXWAKE 位，和 URXIFGx。 在 7 位数据模式下，UxRXBUF 是对齐的 LSB 并且 MSB 总是复位。

18.3.8 UxTXBUF，USART 发送缓冲寄存器

7	6	5	4	3	2	1	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

UxTXBUFx 位 7-0 发送数据缓冲区是用户可以访问的并且保存有等待被转移到发送移位寄存器和 UTXDx 上传输的数据。 写入到发送数据缓冲器清除 UTXIFGx。 UxTXBUF 的 MSB 没有在 7 位数据模式下使用且被复位了。

18.3.9 ME1，模块使能寄存器 1

7	6	5	4	3	2	1	0
UTXE0	**URXE0**						
rw-0	rw-0						

UTXE0 位 7 USART0 发送使能。 该位启用针对 USART0 的发送器。
0 未启用模块
1 启用模块

URXE0 位 6 USART0 接收使能。 该位启用针对 USART0 的接收器。
0 未启用模块
1 启用模块

位 5-0 这些位也可以被其他模块使用。 请参阅《器件专用数据表》。

18.3.10 ME2，模块使能寄存器 2

7	6	5	4	3	2	1	0
		UTXE1	**URXE1**				
		rw-0	rw-0				

位 7-6 这些位也可以被其他模块使用。 请参阅《器件专用数据表》。

UTXE1 位 5 USART1 发送使能。 该位启用针对 USART1 的发送器。
0 未启用模块
1 启用的模块

URXE1 位 4 USART1 接收使能。 该位启用针对 USART1 的接收器。
0 未启用模块
1 启用模块

位 3-0 这些位也可以被其他模块使用。 请参阅《器件专用数据表》。

18.3.11 IE1，中断使能寄存器 1

7	6	5	4	3	2	1	0
UTXIE0	**URXIE0**						
rw-0	rw-0						

UTXIE0 位 7 USART0 发送中断使能。 该位启用 UTXIFG0 中断。
0 中断未启用
1 中断启用

URXIE0 位 6 USART0 接收中断使能。 该位启用 URXIFG0 中断。
0 中断未启用
1 中断被启用

位 5-0 这些位也可以被其他模块使用。 请参阅《器件专用数据表》。

18.3.12 IE2，中断使能寄存器 2

7	6	5	4	3	2	1	0
		UTXIE1	**URXIE1**				
		rw-0	rw-0				

位 7-6 这些位也可以被其他模块使用。 请参阅《器件专用数据表》。

UTXIE1 位 5 USART1 发送中断使能。 该位启用 UTXIFG1 中断。
0 中断未启用
1 中断被启用

URXIE1 位 4 USART1 接收中断使能。 该位启用 URXIFG1 中断。
0 中断未启用
1 中断被启用

位 3-0 这些位也可以被其他模块使用。 请参阅《器件专用数据表》。

18.3.13 IFG1，中断标志寄存器 1

7	6	5	4	3	2	1	0
UTXIFG0	**URXIFG0**						
rw-1	rw-0						

UTXIFG0 位 7 USART0 发送中断标志。 当 U0TXBUF 为空时，UTXIFG0 被置位。
0 无中断等待
1 中断等待

URXIFG0 位 6 USART0 接收中断标志。 当 U0RXBUF 收到一个完整的字符时，URXIFG0 被置位。
0 无中断挂起
1 中断挂起

位 5-0 这些位也可以被其他模块使用。 请参阅《器件专用数据表》。

18.3.14 IFG2，中断标志寄存器 2

7	6	5	4	3	2	1	0
		UTXIFG1	**URXIFG1**				
		rw-1	rw-0				

	位 7-6	这些位也可以被其他模块使用。 请参阅《器件专用数据表》。
UTXIFG1	位 5	USART1 发送中断标志。 当 U1TXBUF 为空时，UTXIFG1 被置位。 0 无中断等待 1 中断等待
URXIFG1	位 4	USART1 接收中断标志。 当 U1RXBUF 收到一个完整的字符时，URXIFG1 被置位。 0 无中断等待 1 中断等待
	位 3-0	这些位也可以被其他模块使用。 请参阅《器件专用数据表》。

USART 外设接口，SPI 模式。

通用同步/异步接收/发送器 (USART) 外设接口采用一个硬件模块支持两个串行模式。 本章介绍了同步外设接口或 SPI 模式的操作。 在 MSP430AFE2xx 器件上执行了 USART0。

Topic **Page**

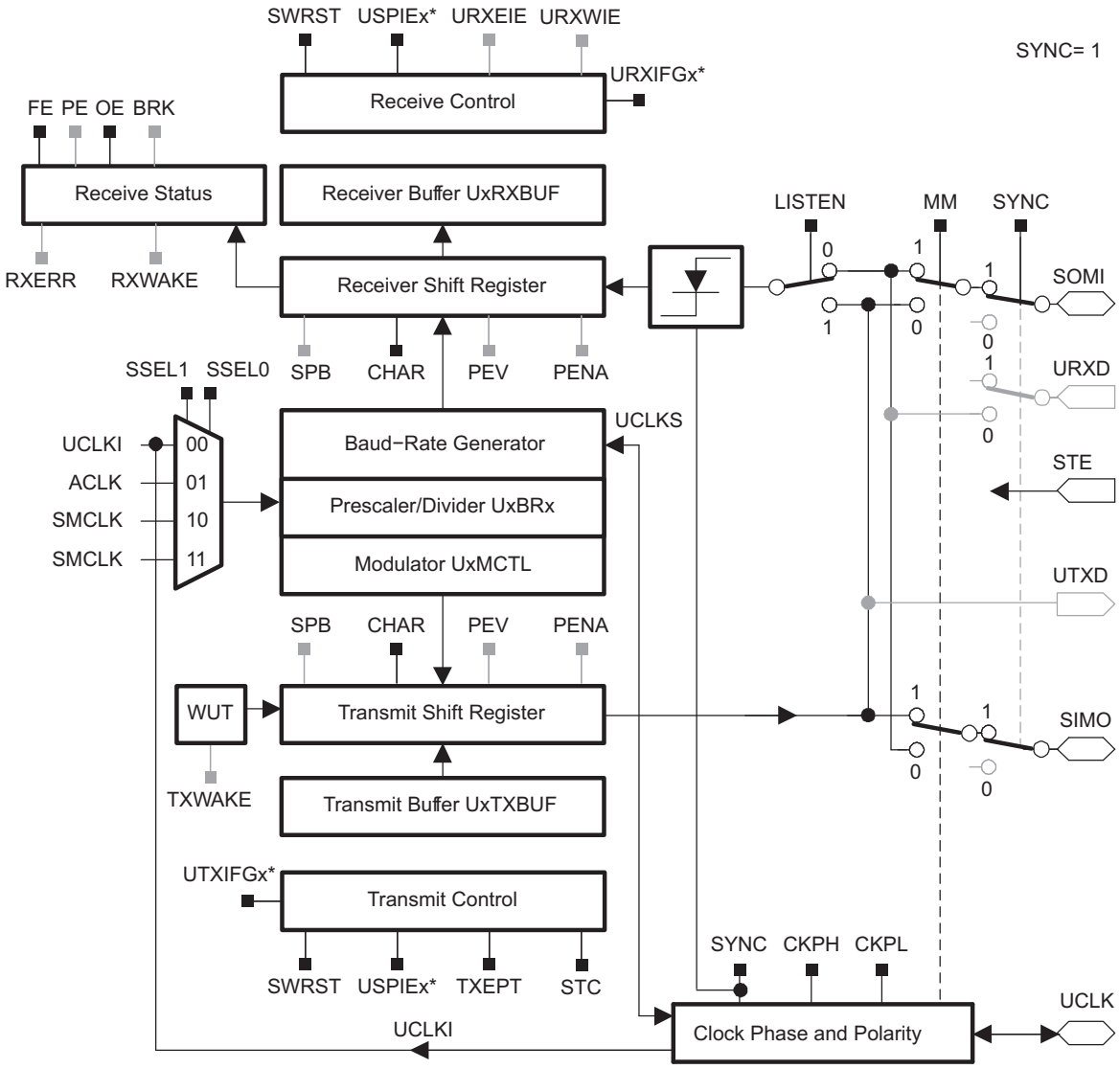
19.1 USART 介绍：SPI 模式

在同步模式中，USART 通过三个或四个脚把 MSP430 与一个外部系统连接起来，这几个引脚是：SIMO，OMI，UCLK，和 STE。 当 SYNC 被置位且 I2C 被清零时，选择 SPI 模式。

SPI 模式的特点包括：

- 7 位或 8 位的数据长度
- 3 引脚或 4 引脚 SPI 操作
- 主控或受控模式
- 独立的发送和接收移位寄存器
- 分开的发送和接收缓冲寄存器
- 可选择的 UCLK 极性和相位控制
- 在主模式下可编程的 UCLK 频率
- 独立的接收和发送中断功能

图 19-1给出了配置为 SPI 模式时的 USART。



* See the device-specific data sheet for SFR locations.

图 **19-1. USART** 方框图：**SPI** 模式

19.2 USART 操作：SPI 模式

在 SPI 模式下，串行数据可通过多路个器件进行发送和接收，这些器件使用一个由主机提供的共用时钟。一个附加的引脚，STE，用于使能器件接收和发送数据，并由主机控制。

三个或四个信号被用于 SPI 数据交换：

- SIMO：从器件输入，主器件输出
  - 主控模式：SIMO 为数据输出线。
  - 受控模式：SIMO 为数据输入线。
- SOMI：从器件输出，主器件输入
  - 主控模式：SOMI 为数据输入线。
  - 受控模式：SOMI 为数据输出线。
- UCLK：USART SPI 时钟
  - 主控模式：UCLK 是一个输出端。
  - 受控模式：UCLK 是一个输入端。
- STE：从机发送使能。 用于 4 引脚模式以便允许在一条单总线上的多个主器件。 不用于 3 引脚模式。
  - 4 引脚主控模式：
    - 当 STE 为高电平时，SIMO 和 UCLK 正常运行。
    - 当 STE 为低电平时，SIMO 和 UCLK 被设置到输入方向。
  - 4 引脚受控模式：
    - 当 STE 为高电平时，从器件的 RX/TX 被禁用且 SOMI 被强制为输入方向。
    - 当 STE 为低电平时，从器件的 RX/TX 被启用和且 SOMI 正常工作。

19.2.1 USART 的初始化和复位

通过一个 PUC 或 SWRST 位复位 USART。 一个 PUC 后，SWRST 位自动置位，保持 USART 处于一个复位条件下。 当置位时，SWRST 位复位 URXIEx，UTXIEx，URXIFGx，OE，和 FE 位并并置位 UTXIFGx 标志。 USPIEx 位没被 SWRST 改变。 清零 SWRST 会释放 USART 进行操作。

注： **初始化或重新配置 USART 模块**

所需的 USART 初始化/重新配置过程是：

1. 置位 SWRST (BIS.B #SWRST，&UxCTL)
2. UCSWRST=1，初始化所有 USART 寄存器（包括 UxCTL）
3. 通过 MEx SFRs (USPIEx) 启用 USART 模块
4. 通过软件清零 SWRST (BIC.B #SWRST，&UxCTL)
5. 通过 IExSFR（URXIEx 和/或 UTXIEx）启用中断（可选）

未能按照这个过程中可能会导致不可预知的 USART 运行方式。

19.2.2 主控模式

图 19-2给出了在 3 引脚和 4 引脚配置下的作为一个主器件的 USART。 当数据被移动到发送数据缓冲时 UxTXBUF 时，USART 初始化数据传输。 当 TX 移位寄存器为空时，UxTXBUF 数据被移动到 TX 移位寄存器，SIMO 上的初始化数据传输从最高有效位开始。 SOMI 上的数据被移入在反相的时钟沿上的接收移位寄存器，从最高有效位开始。 当字符接收到之后，接收数据被从 RX 移位寄存器送入接收到的数据缓冲器 UxRXBUF，并且接收中断标志，URXIFGx，被置位，完成初始化 RX/TX （接收/发送）操作。

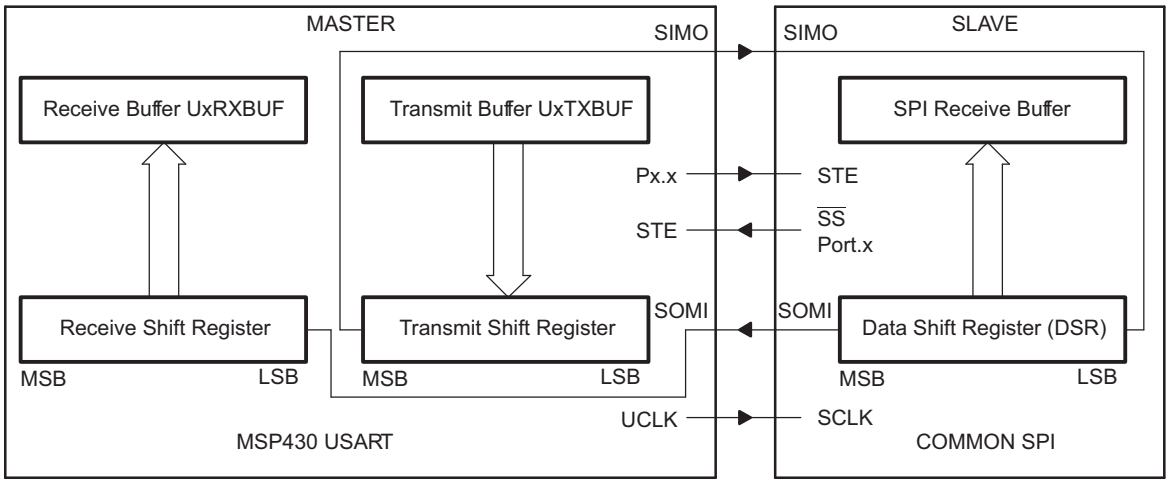


图 **19-2. USART** 主器件和外部从器件

一组传输中断标志，UTXIFGx，表明数据已经从 UxTXBUF 转移到 TX 移位寄存器且 UxTXBUF 已为新的数据准备就绪。 它并不表示 RX/TX 完成。 在主控模式下，一个有效传输的完成是用一组发送器的空位 TXEPT=1 表示的。

要在主控模式下把数据接收到 USART 中，鉴于接收和发送操作同时运行的，必须把数据写入 UxTXBUF。

19.2.2.1 4 引脚 SPI 主控模式

在 4 引脚主控模式下，STE 用于防止与另一个主器件发生冲突。 当 STE 为高电平时主器件正常操作。 当 STE 为低电平时：

- SIMO 和 UCLK 被设为输入，并不再驱动总线
- 错误位 FE 被置位表示了一个通信完整性违规将被用户处理。

一个低 STE 信号不会复位 USART 模块。 STE 输入信号不使用在 3 引脚主控模式中。

19.2.3 受控模式

图 19-3给出了在 3 引脚和 4 引脚配置下的作为一个从器件的 USART。 UCLK 被使作 SPI 时钟的输入且必须由外部主器件提供。 数据传输速率是由这个时钟确定而不是由内部波特率发生器确定。 在一个 UCLK 开始前，会在 SOMI 上发送写入 UxTXBUF 和被移到 TX 移位寄存器的数据。 SIMO 上的数据被移进在 UCLK 反相边沿上的接收移位寄存器，且当设定位数被接收时数据被移到 UxRXBUF。 当数据被从 RX 移位寄存器移动到 UxRXBUF 时，URXIFGx 中断标志被置位，表明数据已被接收。 溢出错误位，OE，当在新数据没被移到 UxRXBUF 前，接收到的数据无法从 UxRXBUF 中读取时，该位被置位。

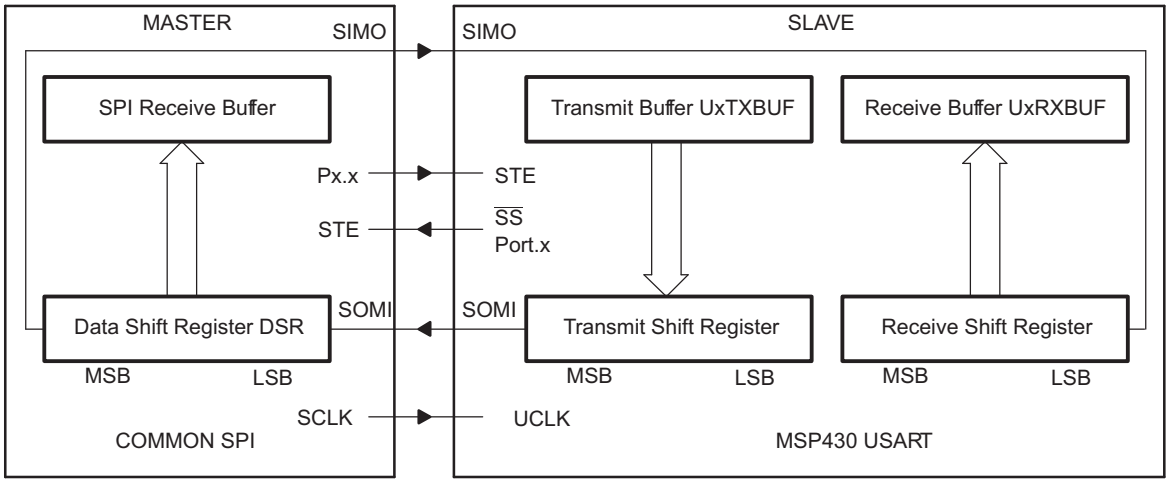


图 **19-3. USART** 从器件和外部主器件

19.2.3.1 4 引脚 SPI 受控模式

在 4 引脚受控模式中，STE 被从器件用于启用发送和接收操作，并由 SPI 主器件提供。 当 STE 为低电平时，从器件正常操作。 当 STE 为高电平时：

- 在 SIMO 上的任何进行中的接收操作都被暂停
- SOMI 被设置为输入方向

一个高 STE 信号不会复位 USART 模块。 STE 输入信号不使用在 3 引脚受控模式中。

19.2.4 SPI 使能

在 SPI 模式下，SPI 发送/接收使能位 USPIEx 使能或禁用 USART。 当 USPIEx=0 时，在当前数据传输完成后 USART 会停止操作，或如果没有操作处于活动状态的话会立即停止操作。 一个 PUC 或一组 SWRST 位立即禁止了 USART 且任何活动传输被终止。

19.2.4.1 发送使能

当 USPIEx=0 时，不发送任何更多写入 UxTXBUF 。 当 USPIEx=1 且 BRCLK 源处于活动状态时，开始发送写入 UxTXBUF 的数据。图 19-4和图 19-5给出了发送使能状态结构图。

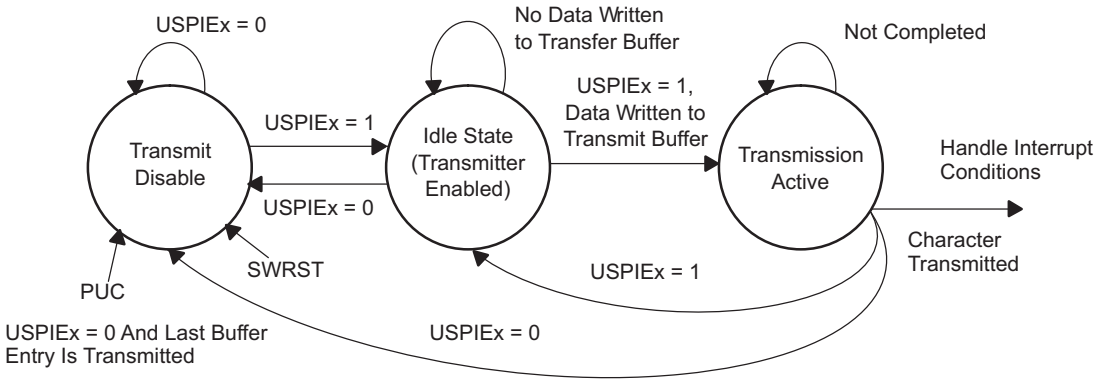


图 **19-4.** 主器件发送使能状态结构图

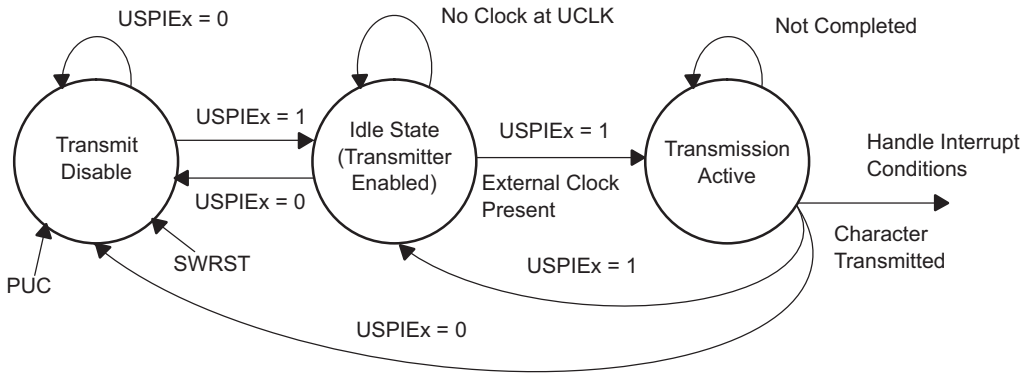


图 **19-5.** 从器件发送使能状态结构图

19.2.4.2 接收使能

在图 19-6和图 19-7中给出了 SPI 接收使能状态结构图。 当 USPIEx=0 时，UCLK 被禁止转移数据到 RX 移位寄存器。

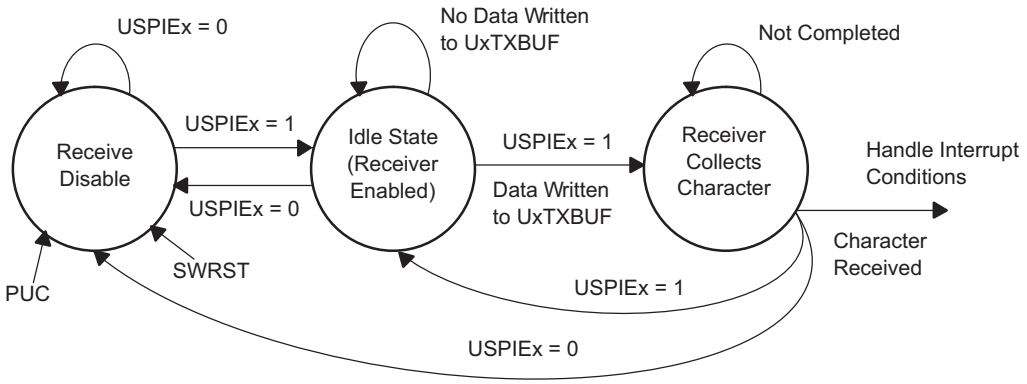


图 **19-6. SPI** 主器件接收使能状态结构图

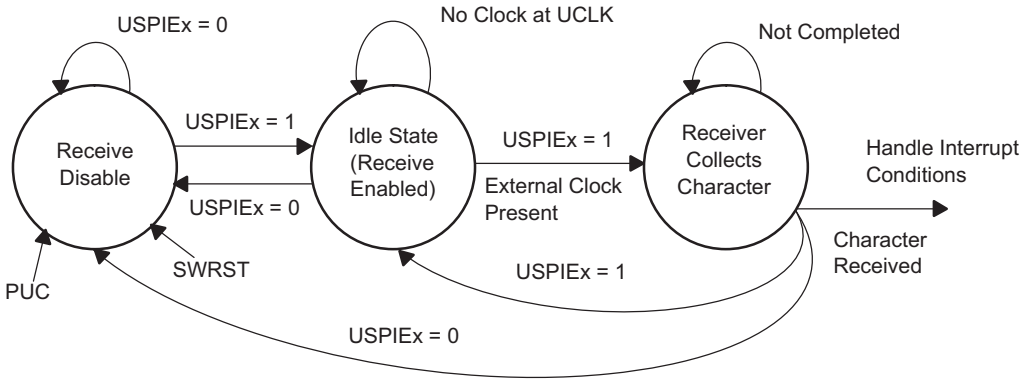


图 **19-7. SPI** 从器件接收使能状态结构图

19.2.5 串行时钟控制

UCLK 由 SPI 总线上的主器件提供。 当 MM=1 时，BITCLK 由在 UCLK 引脚上的 USART 波特率发生器提供，如在图 19-8中所示。 当 MM=0 时，USART 时钟由 UCLK 引脚上的主机提供，不使用波特率发生器，且 SSELx 位是“无关”。 SPI 接收器和传送器并行操作且并数据传输使用相同的时钟源。

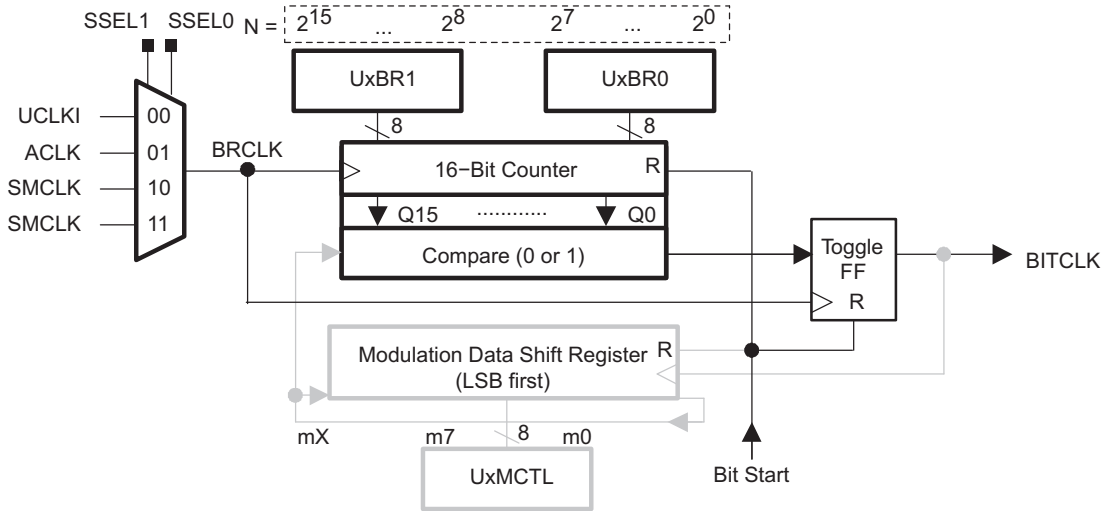


图 **19-8. SPI** 波特率发生器

UxBR0+ UxBR1 的 16 位值是 USART 时钟源的分频因子，BRCLK。 在主控模式下是可以产生的最大波特率是 BRCLK/2。在受控模式下可以产生的最大波特率是 BRCLK。在 USART 波特率发生器中的调制器不用于SPI模式，并建议设置到 000h。 UCLK 频率由下式给出：

$$\text{Baud rate} = \frac{\text{BRCLK}}{\text{UxBR}} \text{ with UxBR= [UxBR1, UxBR0]}$$

19.2.5.1 串行时钟的极性和相位

可通过 USART 的 CKPL 和 CKPH 控制位独立配置 UCLK 的极性和相位。 在图 19-9中给出了每种情况下的时序。

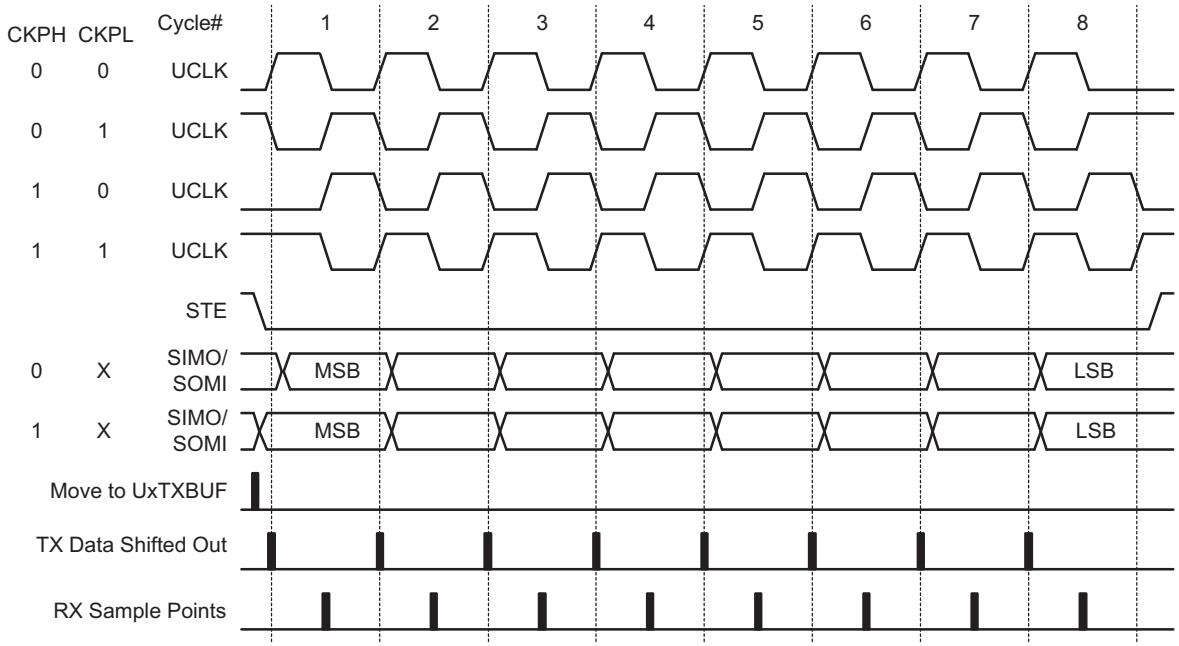


图 **19-9. USART SPI** 时序

19.2.6 SPI 中断

USART 有一个传输中断矢量的和一个接收中断矢量。

19.2.6.1 SPI 的发送中断操作

发射器置位 UTXIFGx 中断标志表示 UxTXBUF 已准备好接受另一个字符。 如果 UTXIEx 和 GIE 也被置位，会产生一个中断请求。 如果中断请求被服务或如果一个字符被写入 UxTXBUF，UTXIFGx 会自动复位。

在一个 PUC 后或当 SWRST=1 时，UTXIFGx 被置位。在一个 PUC 后或当 SWRST=1 时，UTXIFGx 被复位。在图 19-10中展示了该操作。

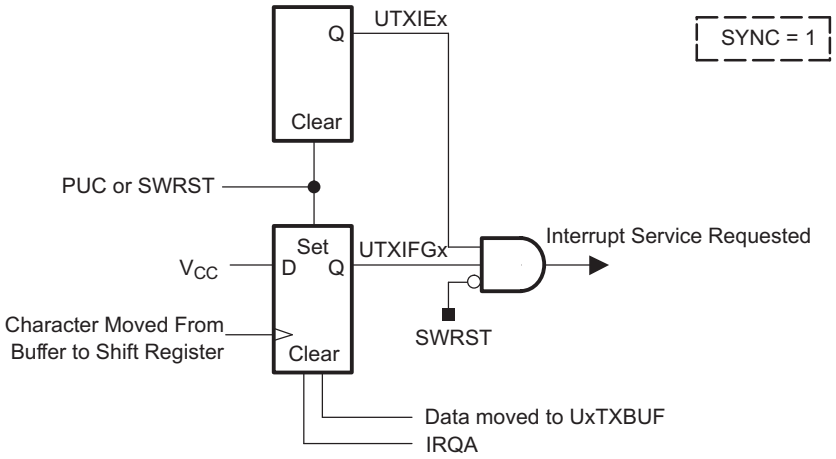


图 **19-10.** 发送中断操作

注: **在 SPI 模式下写入 UxTXBUF**

当 UTXIFGx=0 且 USPIEx=1 时，写入 UxTXBUF 的数据可能会导致错误的数据传输。

19.2.6.2 SPI 接收中断操作

在每次接收到一个字符并加载到 UxRXBUF 中时，URXIFGx 中断标志被置为，如在图 19-11和图 19-12中所示。 如果 URXIEx 和 GIE 也被置位，会产生一个中断请求。 URXIFGx 和 URXIEx 被一次系统复位 PUC 信号复位或当 SWRST=1 时被复位。如果挂起的中断被送达或当 UxRXBUF 被读取时，URXIFGx 会自动置位。

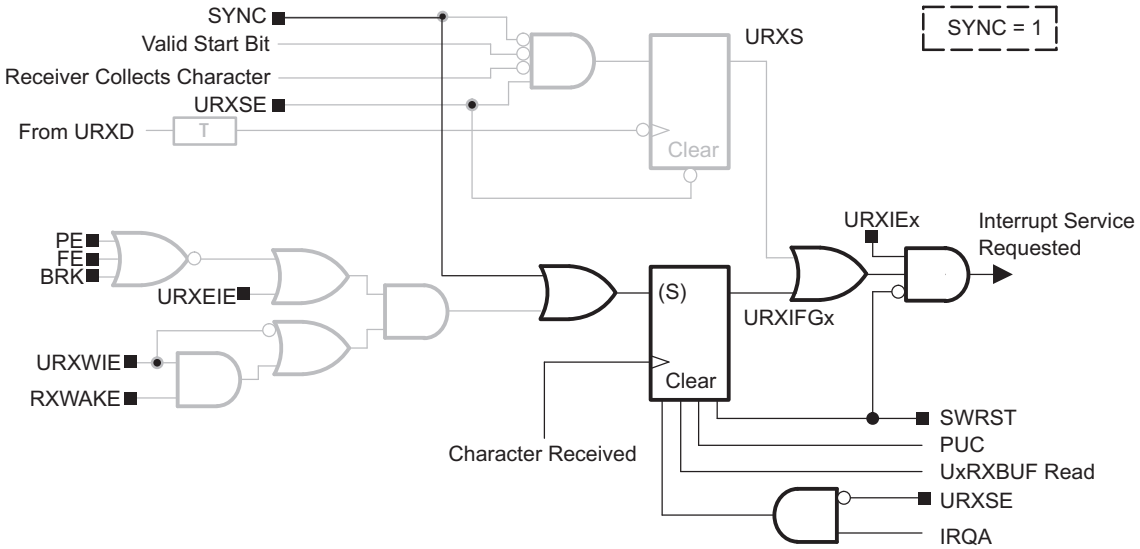


图 **19-11.** 接收中断操作

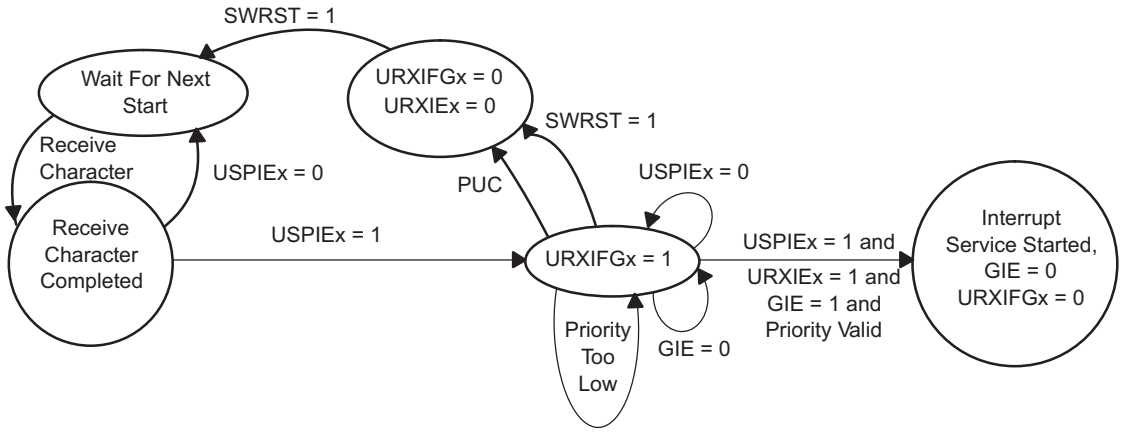


图 **19-12.** 接收中断状态图

19.3 USART 寄存器：SPI 模式

表 19-1列出了执行一个 USART 模块的所有器件的寄存器。表 19-2只适用于有第二 USART 模块，USART1 的器件。

表 19-1. USART0 控制和状态寄存器

寄存器	简式	寄存器类型	地址	初始化状态
USART 控制寄存器	U0CTL	读取/写入	070h	001H 与 PUC
发送控制寄存器	U0TCTL	读取/写入	071h	001h 与 PUC
接收控制寄存器	U0RCTL	读取/写入	072h	000h 与 PUC
调制控制寄存器	U0MCTL	读取/写入	073h	未改变
波特率控制寄存器 0	U0BR0	读取/写入	074h	未改变
波特率控制寄存器 1	U0BR1	读取/写入	075h	未改变
接收缓冲寄存器	U0RXBUF	读取	076h	未改变
发送缓冲寄存器	U0TXBUF	读取/写入	077h	未改变
SFR 模块使能寄存器 1	ME1	读取/写入	004h	000h 与 PUC
SFR 中断使能寄存器 1	IE1	读取/写入	000h	000h 与 PUC
SFR 中断标志寄存器 1	IFG1	读取/写入	002h	082h 与 PUC

表 19-2. USART1 控制和状态寄存器

寄存器	简式	寄存器类型	地址	初始化状态
USART 控制寄存器	U1CTL	读取/写入	078h	001h 与 PUC
发送控制寄存器	U1TCTL	读取/写入	079h	001h 与 PUC
接收控制寄存器	U1RCTL	读取/写入	07Ah	000h 与 PUC
调制控制寄存器	U1MCTL	读取/写入	07Bh	未改变
波特率控制寄存器 0	U1BR0	读取/写入	07Ch	未改变
波特率控制寄存器 1	U1BR1	读取/写入	07Dh	未改变
接收缓冲寄存器	U1RXBUF	读取	07Eh	未改变
发送缓冲寄存器	U1TXBUF	读取/写入	07Fh	未改变
SFR 模块使能寄存器 2	ME2	读取/写入	005h	000h 与 PUC
SFR 中断使能寄存器 2	IE2	读取/写入	001h	000h 与 PUC
SFR 中断标志寄存器 2	IFG2	读取/写入	003h	020h 与 PUC

注： 修改 SFR 位

为了避免修改其他模块的控制位，建议使用 BIS.B 或 BIC.B 指示，而非 MOV.B 或 CLR.B 指令置位或清除 IEX 和 IFGx 位。

19.3.1 UxCTL，USART 控制寄存器

7	6	5	4	3	2	1	0
未被使用		**I2C**	**CHAR**	**LISTEN**	**SYNC**	**MM**	**SWRST**
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1

未被使用	位 7-6	未被使用
I2C	位 5	I^2C 模式使能。 当 SYNC=1 时，该位选择 I^2C 或 SPI 操作。
		0 SPI 模式
		1 I^2C 模式
CHAR	位 4	字符长度
		0 传输的每个数据单元为 7 位。
		1 传输的每个数据单元为 8 位。
监听	位 3	监听使能。 监听位选择回路模式
		0 被禁用
		1 被启用。 发送信号被内部反馈到接收器。
SYNC	位 2	同步模式使能
		0 UART 模式
		1 SPI 模式
MM	位 1	主器件模式
		0 USART 是从器件
		1 USART 是主器件
SWRST	位 0	软件复位使能
		0 被禁用。 USART复位释放操作。
		1 被启用. USART 逻辑保持在复位状态。

19.3.2 UxTCTL，USART 发送控制寄存器

7	6	5	4	3	2	1	0
CKPH	**CKPL**	**SSELx**		未使用		**STC**	**TXEPT**
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1

CKPH	位 7	时钟相位选择。
		0 数据第一个 UCLK 边缘被改变且在下一个边沿被捕获。
		1 数据第一个 UCLK 边缘被捕获且在下一个边沿被改变。
CKPL	位 6	时钟极性选择
		0 不活动状态是低电平。
		1 不活动状态是高电平。
SSELx	位 5-4	源选择。 这些位选择 BRCLK 时钟源。
		00 外部 UCLK（仅在受控模式下有效）
		01 ACLK（仅在主控模式下有效）
		10 SMCLK（仅在主机控模式下有效）
		11 SMCLK（仅在主控模式下有效）
未被使用	位 3-2	未被使用
STC	位 1	从器件发送控制。
		0 4 引脚 SPI 模式：STE 被启用。
		1 3 引脚 SPI 模式：STE 被禁用。
TXEPT	位 0	发送器空标志。 TXEPT 标志不在受控模式中使用。
		0 在 UxTXBUF 中传输有效和/或数据等待
		1 UxTXBUF 和 TX 移位寄存器是空的

19.3.3 UxRCTL，USART 接收控制寄存器

7	6	5	4	3	2	1	0
FE	未被使用	**OE**	未被使用				
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

FE	位 7	帧出错标志。 当 MM=1 且 STC=0 时该位表示一个总线冲突。FE 在受控模式下未使用。 0 没有检测到冲突 1 在 STE 上发生一个负边缘，表示总线冲突
未被使用	位 6	未使用
OE	位 5	溢出错误标志。 在前一个字符被读取前，一个字符被传输进 UxRXBUF 时，该位被置位。 当 UxRXBUF 被读取时，OE 自动复位，当 SWRST=1 时，也可以由软件复位。 0 无错误 1 发生溢出错误
未使用	位 4-0	未被使用

19.3.4 UxBR0，USART 波特率控制寄存器 0

7	6	5	4	3	2	1	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

19.3.5 UxBR1，USART 波特率控制寄存器 1

7	6	5	4	3	2	1	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

UxBRx	波特率发生器使用 {UxBR1+ UxBR0} 的内容来设置波特率。 如果 UxBR<2，会发生不可预知的 SPI 操作。

19.3.6 UxMCTL，USART 调制控制寄存器

7	6	5	4	3	2	1	0
m7	**m6**	**m5**	**m4**	**m3**	**m2**	**m1**	**m0**
rw	rw	rw	rw	rw	rw	rw	rw

UxMCTLx	位 7-0	调制控制寄存器不用于 SPI 模式且应被设置到 000h。

19.3.7 UxRXBUF，USART 接收缓冲寄存器

7	6	5	4	3	2	1	0
2^7	**2^6**	**2^5**	**2^4**	**2^3**	**2^2**	**2^1**	**2^0**
r	r	r	r	r	r	r	r

UxRXBUFx	位 7-0	接收数据缓冲是用户可以访问并包含从接收移位寄存器处最后接收到的字符。 读取 UxRXBUF 将复位 OE 位和 URXIFGx 标志。 在 7 位数据模式下，UxRXBUF 是已对齐的 LSB 且 MSB 总是复位。

19.3.8 UxTXBUF，USART 发送缓冲寄存器

7	6	5	4	3	2	1	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

UxTXBUFx	位 7-0	发送数据缓冲是用户可访问的并且包含要被发送的当前数据。 当使用 7 位字符长度时，在被转移到 UxTXBUF 前数据应该是已对齐的 MSB。 数据被传输，MSB 首先被传输。 写入 UxTXBUF 将清零 UTXIFGx。

19.3.9 ME1，模块使能寄存器 1

7	6	5	4	3	2	1	0
	USPIE0						
	rw-0						

	位 7	该位也可以被其他模块使用。 请参阅《器件专用数据表》。
USPIE0	位 6	USART0 SPI 使能。 该位启用针对 USART0 的 SPI 模式。 0 模块未被启用 1 模块被启用
	位 5-0	这些位也可以被其他模块使用。 请参阅《器件专用数据表》。

19.3.10 ME2，模块使能寄存器 2

7	6	5	4	3	2	1	0
			USPIE1				
			rw-0				

	位 7-5	这些位也可以被其他模块使用。 请参阅《器件专用数据表》。
USPIE1	位 4	USART1 SPI 使能。 该位启用 USART1 的 SPI 模式。 0 模块未被启用 1 模块 被启用
	位 3-0	这些位也可以被其他模块使用。 请参阅《器件专用数据表》。

19.3.11 IE1，中断使能寄存器 1

7	6	5	4	3	2	1	0
UTXIE0	**URXIE0**						
rw-0	rw-0						

UTXIE0	位 7	USART0 发送中断使能。 该位使能 UTXIFG0 中断。 0 中断未被启用 1 中断被启用
URXIE0	位 6	USART0 接收中断使能。 该位启用 URXIFG0 中断。 0 中断未被启用 1 中断被启用
	位 5-0	这些位也可以被其他模块使用。 请参阅《器件专用数据表》。

19.3.12 IE2，中断使能寄存器 2

7	6	5	4	3	2	1	0
		UTXIE1	**URXIE1**				
		rw-0	rw-0				

	位 7-6	这些位也可以被其他模块使用。 请参阅《器件专用数据表》。
UTXIE1	位 5	USART1 发送中断使能。 该位启用 UTXIFG1 中断。
		0 中断未被启用
		1 中断被启用
URXIE1	位 4	USART1 接收中断使能。 该位启用 URXIFG1 中断。
		0 中断未被启用
		1 中断被启用
	位 3-0	这些位也可以被其他模块使用。 请参阅《器件专用数据表》。

19.3.13 IFG1，中断标志寄存器 1

7	6	5	4	3	2	1	0
UTXIFG0	**URXIFG0**						
rw-1	rw-0						

UTXIFG0	位 7	USART0 发送中断标志。 U0TXBUF 为空时置位 UTXIFG0。
		0 无中断等待
		1 中断等待
URXIFG0	位 6	USART0 接收中断标志。 当 U0RXBUF 已收到一个完整的字符时置位 URXIFG0。
		0 无中断等待
		1 中断等待
	位 5-0	这些位也可以被其他模块使用。 请参阅《器件专用数据表》。

19.3.14 IFG2，中断标志寄存器 2

7	6	5	4	3	2	1	0
		UTXIFG1	**URXIFG1**				
		rw-1	rw-0				

	位 7-6	这些位也可以被其他模块使用。 请参阅《器件专用数据表》。
UTXIFG1	位 5	USART1 发送中断标志。 当 U1TXBUF 为空时置位 UTXIFG1。
		0 无中断挂起
		1 中断挂起
URXIFG1	位 4	USART1 接收中断标志。 当 U1RXBUF 已收到一个完整的字符时置位 URXIFG1。
		0 无中断挂起
		1 中断挂起
	位 3-0	这些位也可以被其他模块使用。 请参阅《器件专用数据表》。

OA

OA 是一个通用的运算放大器。 本章介绍了 OA。 MSP430x22x4 器件中应用了两个 OA 模块。

Topic **Page**

20.1 OA 介绍

OA 运算放大器支持模数转换前的前端模拟信号调理。

OA 的特性包括：

- 单电源，低电流运行
- 轨至轨输出
- 可编程建立时间与功耗的关系
- 可选择的软件配置
- 用于 PGA（可编程增益放大器）应用的可选择的软件反馈电阻。

注： **多 OA 模块**

有些器件可以内置一个以上的 OA 模块。 如果在一个器件上出现一个以上的 OA 模块，则多 OA 模块在操作上完全相同。

本章中，将会命名 OAxCTL0 这样的术语来描述寄存器的名字。 这种情况下，x 被用于指代正在被讨论的 OA 模块。 在操作完全相同的情况下，寄存器被简写为 OAxCTL0。

OA 模块框图如图 20-1所示。

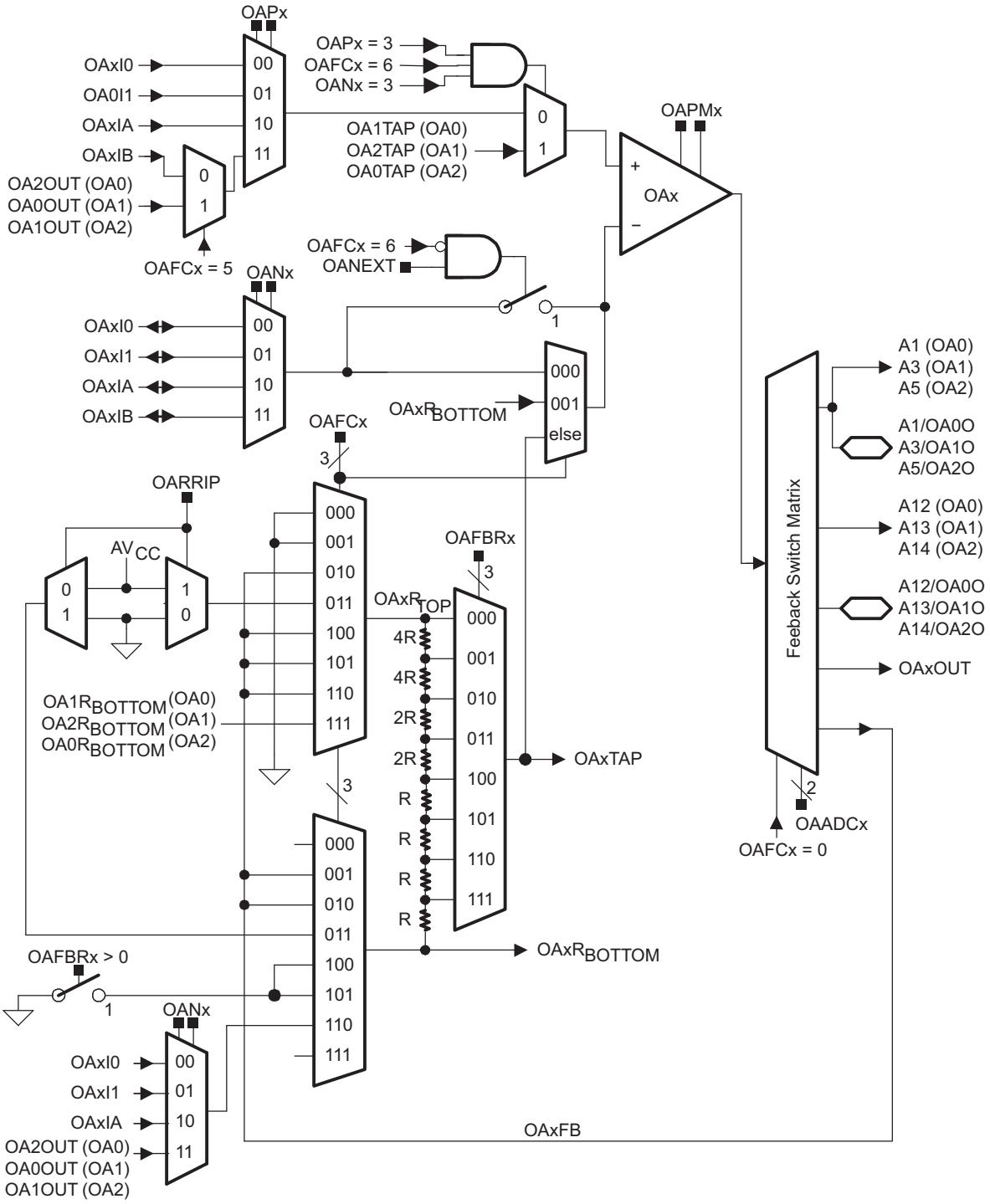


图 **20-1. OA** 方框图

20.2 OA 操作

OA 模块由用户软件配置。 下面部分将讨论 OA 的设置和操作。

20.2.1 OA 放大器

OA 是一个可配置、低电流、轨对轨输出的运算放大器。 它可以被配置成反相放大器，或同相放大器，或与其它 OA 模块组合形成差分放大器。 OA 的输出回转速率可以用 OAPMx 位来配置最佳的稳定时间与功耗。 当 OAPMx = 00 时，OA 关闭，输出为高阻态。 当 OAPMx > 0 时, OA 打开。 参数请参阅《特定器件的数据手册》。

20.2.2 OA 输入

OA 具有可配置的输入选择。 + 和 - 端的信号输入可以通过 OANx 和 OAPx 位进行独立选择并且可以被选择为外部信号或内部信号。 OAxI0 和 OAxI1 是为每一个 OA 模块提供的外部信号。 OA0I1 为所有 OA 模块都提供一个内部相连的同相输入。 OAxIA 和 OAxIB 提供相关器件输入。 信号连接请参阅《器件数据表》。

当一种模式不需要外部反相输入时，设置 OANEXT 位可使内部反相输入对外可用。

20.2.3 OA 输出和反馈路线

OA 具有可配置的输出选择，由 OAADCx 位和 OAFCx 位控制。 OA 输出信号可以内部传送到 ADC 输入 A12 (OA0), A13 (OA1) 和 A14 (OA2)，或者连接到这些 ADC 的输入和其外部引脚。 OA 输出信号也可以传送到 ADC 输入 A1 (OA0), A3 (OA1) 或 A5 (OA2)，和相应的外部引脚。 OA 输出也可以通过设置 OAFCx 位连接到一个内部梯形电阻网络。 梯形电阻网络抽头可由 OAFBRx 位选择以提供可编程的增益放大功能。

表 20-1显示了 OA 输出和回馈线路配置。 当 OAFCx = 0 时，OA 为通用模式，器件的反馈在外部完成。 当 OAFCx > 0 和 OAADCx = 00 或 11 时,OA 的输出被保持在内部连接到器件。 当 OAFCx > 0 和 OAADCx= 01 或 10 时, OA 输出由内部和外部两种路径连接到器件。

表 20-1. OA 输出配置

OAFCx	OAADCx	OA 输出和反馈路线
= 0	x0	OAxOUT 连接到外部引脚和 ADC 输入 A1, A3 或 A5。
= 0	x1	OAxOUT 连接到外部引脚和 ADC 输入 A12, A13 或 A14。
> 0	00	OAxOUT 只用作内部线路。
> 0	01	OAxOUT 连接到外部引脚和 ADC 输入 A12, A13 或 A14。
> 0	10	OAxOUT 连接到外部引脚和 ADC 输入 A1, A3 或 A5。
> 0	11	OAxOUT 内部连接到 ADC 输入 A12，A13，A14。 外部 A12，A13，A14 引脚连接与 ADC 断开。

20.2.4 OA 配置

OA 可以通过 OAFCx 位配置来拥有不同的放大功能，如表 20-2所列。

表 20-2. OA 模式选择

OAFCx	OA 模式
000	通用运算放大器
001	用于三运放差分放大器的单位增益缓冲器
010	单位增益缓冲器
011	比较器
100	同相 PGA（比例放大）放大器
101	级连同相 PGA 放大器
110	反相 PGA 放大器
111	差分放大器

20.2.4.1 通用运算放大器模式

在该模式下，反馈梯形电阻与 OAx 是分离的，并且由 OAxCTL0 位定义信号路径。 OAx 输入由 OAPx 和 OANx 位选择。 OAx 输出可以通过 OAxCTL0 位来选择连接到 ADC 输入通道。

20.2.4.2 用于差分放大器的单位增益缓冲器

该模式下，OAx 的输出被连接到 OAx 的反相输入端，以此来提供一个单位增益缓冲器。 同相输入由 OAPx 位选择。 反相输入端的外部连接被禁用并且 OANx 位可随意设置。 通过梯形电阻，OAx 的输出也被路由来作为三运放差分放大器的一部分。 该模式只用于三运放差分放大器结构。

20.2.4.3 单位增益模式

该模式下，OAx 的输出被连接到 OAx 的反相输入端以此来提供一个单位增益缓冲器。 同相输入由 OAPx 位选择。 反相输入端的外部连接被禁用并且 OANx 位随意设置。 OAx 输出可以通过 OAxCTL0 位来选择连接到 ADC 输入通道。

20.2.4.4 比较器模式

该模式下，反馈梯形电阻与 OAx 的输出是独立的。 当 OARRIP = 0 时，R_{TOP}被连接到 AV_{SS}而 $R_{底部}$被连接到 AV_{CC}。当 OARRIP = 1 时，梯形电阻的连接是相反的。 $R_{顶部}$被连接到 AV_{CC}而$R_{底部}$被连接到 AV_{SS}。OAxTAP 信号被连接到 OAx 的反相输入端以此来提供一个具有可编程门限电压的比较器，该电压由 OAFBRx 位确定。 同相输入由 OAPx 位选择。 通过一个外部正反馈电阻来增加迟滞。 反相输入端的外部连接被禁用并且 OANx 位随意设置。 OAx 输出可以通过 OAxCTL0 位来选择连接到 ADC 输入通道。

20.2.4.5 同相 PGA 模式

该模式下，OAx 的输出端被连接到 $R_{顶部}$并且 $R_{底部}$被连接到 AV_{SS}。 OAxTAP 信号端被连接到 OAx 的反相输入端以此来提供一个具有[1+OAxTAP 比值]增益的可编程同相放大器。 OAxTAP 比率由 OAFBRx 选择。 如果 OAFBRx 位＝0，增益为单位增益。 同相输入由 OAPx 位选择。 反相输入端的外部连接被禁用并且 OANx 位随意设置。 OAx 输出可以通过 OAxCTL0 位来选择连接到 ADC 输入通道。

20.2.4.6 级联同相 PGA 模式

该模式允许 OA 信号在内部按照反相模式级联 2 个或 3 个 OA。 该模式下，当 OAPx = 11 时，OAx 的同相输入端被连接到 OA2OUT (OA0)，OA0OUT (OA1) 或 OA1OUT (OA2)。 OAx 输出端可以通过 OAxCTL0 位来选择连接到 ADC 输入通道。

20.2.4.7 反相 PGA 模式

该模式下，OAx 的输出端被连接到 $R_{顶部}$和 $R_{底部}$被连接到一个模拟多路复用器，它可以复用 OAxI0, OAxI1, OAxIA或由 OANx 位选择的剩余 OAs 的一个输出端。 OAxTAP 信号端被连接到 OAx 的反相输入端以此来提供一个具有 -OAxTAP 比率增益的反相放大器。 OAxTAP 比值由 OAFBRx 位选择。 同相输入由 OAPx 位选择。 OAx 输出可以通过 OAxCTL0 位来选择连接到 ADC 输入通道。

注: 同时使用 OAx 负输入作为 ADC 输入

当引脚连接到反相输入多路复用器时，也被用作了 ADC 的输入，由于内部线路上的压降，转换误差可能上升到 5mV。

20.2.4.8 差分放大器模式

该模式允许 OA 信号在内部连接形成一个两运放或三运放的仪表放大器。图 20-2显示了一个具有 OA0 和 OA1 配置的双运放配置。 该模式下，通过经由反相 PGA 模式下的另一 OAx，OAx 的输出端被连接到 $R_{顶部}$。 $R_{底部}$被断开来提供一个单位增益缓冲器。 该缓冲器与剩下的 1 到 2 个 OAx 组合形成差分放大器。 OAx 输出可以通过 OAxCTL0 位来选择连接到 ADC 输入通道。

图 20-2显示了一个由 OA0 和 OA1 构成的两运放差分放大器。 控制寄存器的设置如表 20-3所示。 放大器的增益由 OA1 的 OAFBRx 位决定，如表 20-4所示。 OAx 的相互连接如图 20-3所示。

表 20-3. 两运放差分放大器控制寄存器设置

寄存器	设置 （二进制）
OA0CTL0	xx xx xx 00
OA0CTL1	000 111 0 x
OA1CTL0	11 xx xx x x
OA1CTL1	xxx 110 0 x

表 20-4. 两运放差分放大器增益设置

OA1 OAFBRx	增益
000	0
001	1/3
010	1
011	1 2/3
100	3
101	4 1/3
110	7
111	15

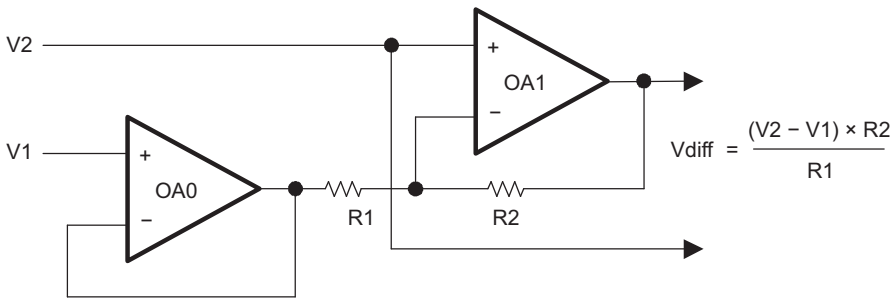


图 20-2. 两运放差分放大器

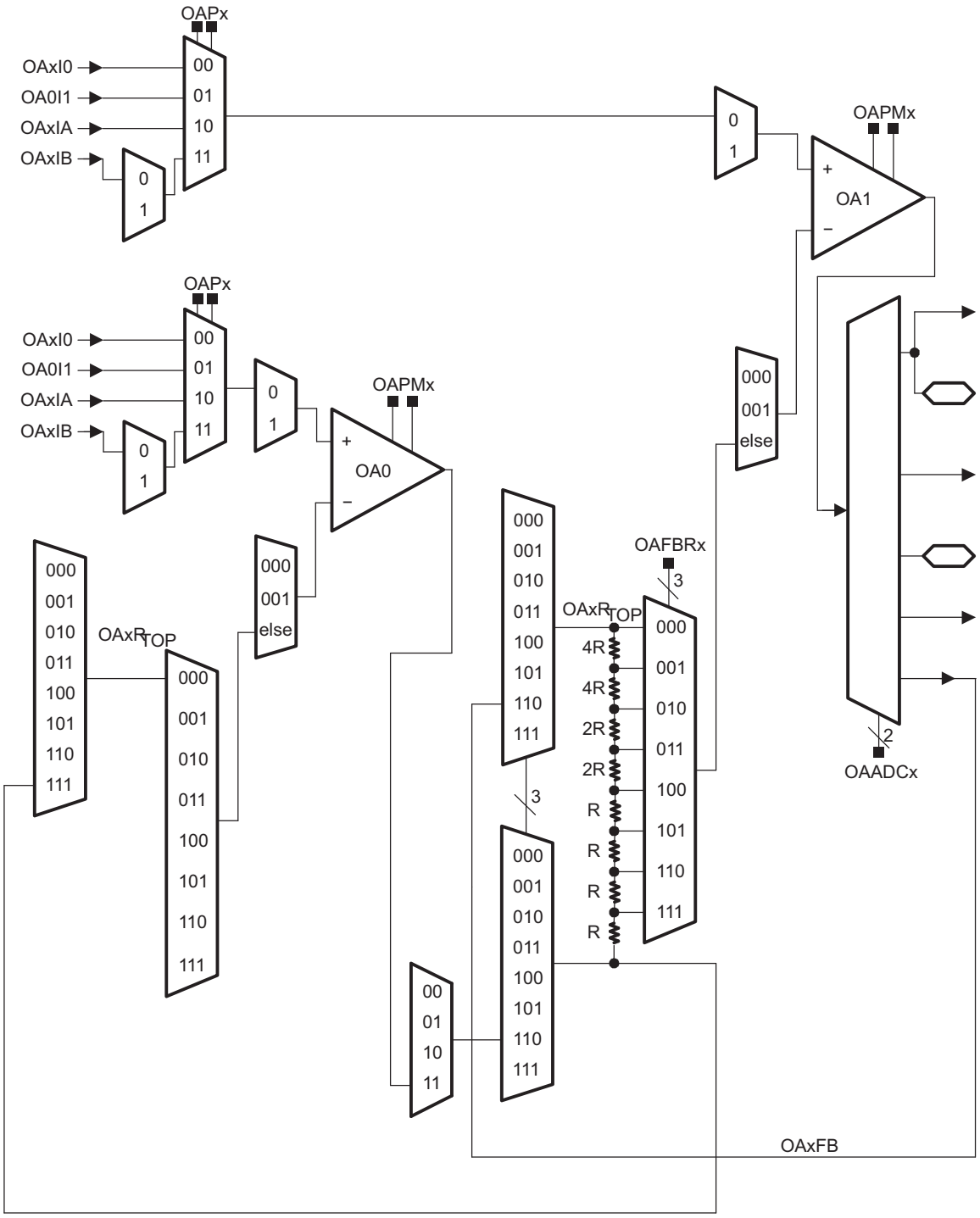


图 **20-3.** 两运放差分放大器 **OAx** 的相互连接

图 20-4显示了使用 OA0，OA1 和 OA2 构成的三运放差分放大器（三运放不是在所有的器件中都具有。 请参阅《特定器件数据手册》。）。 控制寄存器的设置如表 20-5所示。 放大器的增益由 OA0 和 OA2 的 OAFBRx 位决定。 OA0 和 OA2 的 OAFBRx 位设定必须相同。 增益设置如表 20-6所示。 OAx 的相互连接如图 20-5所示

表 **20-5.** 三运放差分放大器控制寄存器设置

寄存器	设置 （二进制）
OA0CTL0	xx xx xx 00
OA0CTL1	xxx 001 0 x
OA1CTL0	xx xx xx 0 0
OA1CTL1	000 111 0 x
OA2CTL0	11 11 xx x x
OA2CTL1	xxx 110 0 x

表 **20-6.** 三运放差分放大器增益设置

OA0/OA2 OAFBRx	增益
000	0
001	1/3
010	1
011	1 2/3
100	3
101	4 1/3
110	7
111	15

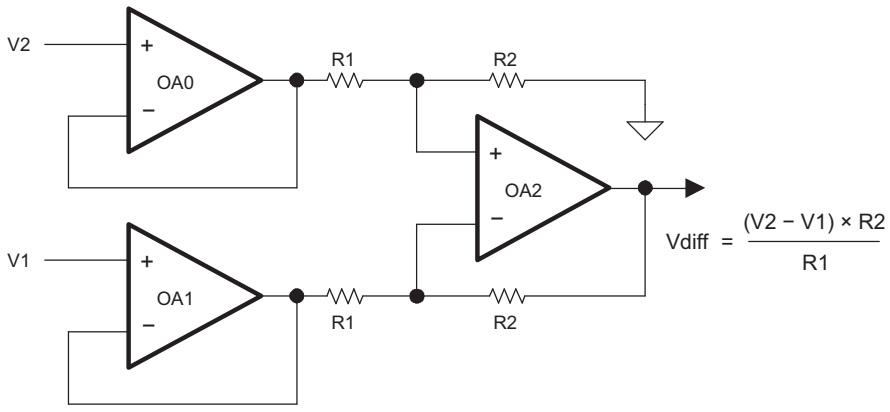


图 **20-4.** 三运放差分放大器

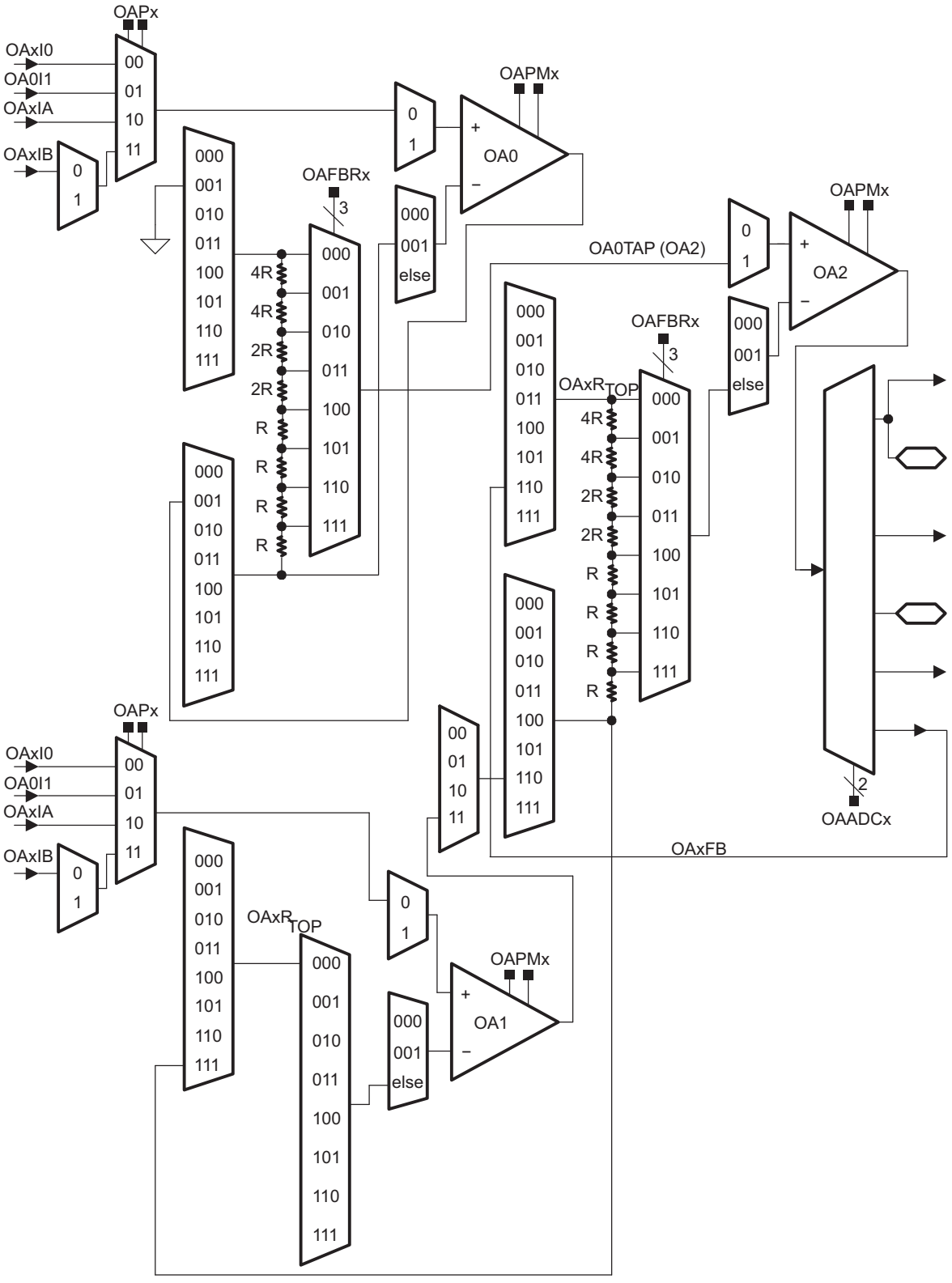


图 **20-5.** 三运放差分放大器 **OAx** 的相互连接

20.3 OA 寄存器

在表 20-7中列出了 OA 寄存器。

表 20-7. OA 寄存器

寄存器	简表	寄存器类型	地址	初态
OA0 控制寄存器 0	OA0CTL0	读取/写入	0C0h	用 POR 复位
OA0 控制寄存器 1	OA0CTL1	读取/写入	0C1h	用 POR 复位
OA1 控制寄存器 0	OA1CTL0	读取/写入	0C2h	用 POR 复位
OA1 控制寄存器 1	OA1CTL1	读取/写入	0C3h	用 POR 复位
OA2 控制寄存器 0	OA2CTL0	读取/写入	0C4h	用 POR 复位
OA2 控制寄存器 1	OA2CTL1	读取/写入	0C5h	用 POR 复位

20.3.1 OAxCTL0，运算放大器控制寄存器 0

7	6	5	4	3	2	1	0
OANx		**OAPx**		**OAPMx**		**OAADCx**	
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

OANx 位 7-6 反向输入选择 这些位为 OA 反相输入选择输入信号。

00 OAxI0

01 OAxI1

10 OAxIA（请参阅《连接信号的器件专用数据手册》）。

11 OAxIB（请参阅《连接信号的器件专用数据手册》）。

OAPx 位 5-4 同相输入选择 这些位为 OA 同相输入选择输入信号。

00 OAxI0

01 OA0I1

10 OAxIA（请参阅《连接信号的器件专用数据手册》）。

11 OAxIB（请参阅《连接信号的器件专用数据手册》）。

OAPMx 位 3-2 转换率选择。 这些位为 OA 选择回转率与电流消耗。

00 关闭, 输出高阻态 Z

01 慢

10 中

11 快

OAADCx 位 1-0 OA 输出选择。 当 OAPMx > 0 时，这些位，和 OAFCx 位一起，控制 OAx 的输出路径。

当 OAFCx = 0 时：

00 OAxOUT 连接到外部引脚和 ADC 输入 A1, A3 或 A5

01 OAxOUT 连接到外部引脚和 ADC 输入 A12, A13 或 A14

10 OAxOUT 连接到外部引脚和 ADC 输入 A1, A3 或 A5

11 OAxOUT 连接到外部引脚和 ADC 输入 A12, A13 或 A14

当 OAFCx ＞ 0 时：

00 OAxOUT 只用作内部线路。

01 OAxOUT 连接到外部引脚和 ADC 输入 A12, A13 或 A14

10 OAxOUT 连接到外部引脚和 ADC 输入 A1, A3 或 A5

11 OAxOUT 内部连接到 ADC 输入 A12，A13，A14。 外部 A12，A13，A14 引脚连接与 ADC 断开。

20.3.2 OAxCTL1，运算放大器控制寄存器 1

7	6	5	4	3	2	1	0
OAFBRx			OAFCx			OANEXT	OARRIP
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

OAFBRx 位 7-5 OAx 反馈电阻选择

000 抽头 0 - 0R/16R
001 抽头 1 - 4R/12R
010 抽头 2 - 8R/8R
011 抽头 3 - 10R/6R
100 抽头 4 - 12R/4R
101 抽头 5 - 13R/3R
110 抽头 6 - 14R/2R
111 抽头 7 - 15R/1R

OAFCx 位 4-2 OAx 功能控制。 该位选择 OAx 的功能

000 通用运算放大器
001 用于三运放差分放大器的单位增益缓冲器
010 单位增益缓冲器
011 比较器
100 同相 PGA（比例放大）放大器
101 级连同相 PGA 放大器
110 反相 PGA 放大器
111 差分放大器

OANEXT 位 1 OAx 反相输入外部可用。 当集成的电阻网络被使用时，该位，当被置位时，把反相 OAX 输入连接到外部引脚。

0 OAx 反相输入在外部不可用。
1 OAx 反相输入在外部可用。

OARRIP 位 0 在比较器模式中 OAX 的反向电阻器连接

0 当 OAFCx = 3 时，$R_{顶部}$被连接到 AV_{SS}而 $R_{底部}$被连接到 AV_{CC}
1 当 OAFCx = 3 时，$R_{顶部}$被连接到 AV_{CC}而 $R_{底部}$被连接到 AV_{SS}。

比较器_A+ (Comparator_A+)

比较器_A+ 是一个模拟电压比较器。 这一章阐述了 2xx 系列中比较器_A+ 的操作。

Topic **Page**

21.1 比较器_A+ 介绍

比较器_A+ 模块支持精确的斜率模数转换，电源电压监控，和外部模拟信号的监控。

比较器_A+ 的特性包括：

- 反向和非反向的端子输入复用器
- 比较器输出的软件可选的 RC 滤波器
- 为定时器_A 的捕获输入提供的输出端
- 端口输入缓冲的软件控制。
- 中断功能
- 可选择的基准电压发生器
- 比较器和基准电压发生器可关闭
- 输入多路复用器

比较器_A+ 的结构框图如图 21-1所示。

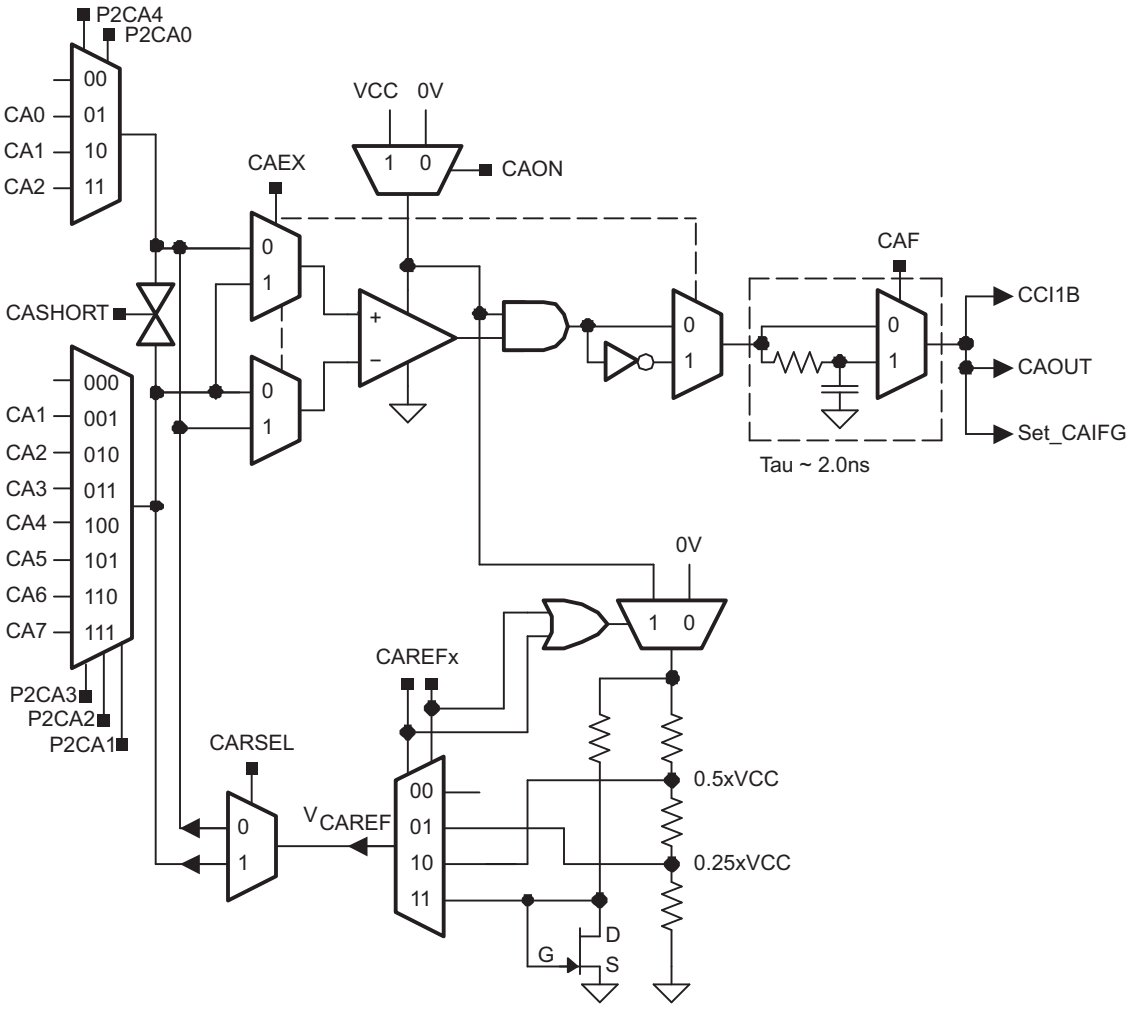


图 **21-1.** 比较器_A+ 方框图

注： **MSP430G2210**：通道 2，5，6，和 7 是可用的。其他通道不应该被启用。

21.2 比较器_A+ 的操作

可以通过用户软件对比较器_A+ 模块进行配置。 下面几个章节对比较器_A+的设置和操作进行了讨论。

21.2.1 比较器

比较器在 + 和 - 端子比较模拟电压。 如若 +端子的电压比 - 端子的高，则比较器输出 CAOUT 为高。 可以通过使用控制位 CAON 来打开或者关闭该比较器。 在不使用该比较器时，应该将其关闭以减小电流消耗。 当比较器关闭时，CAOUT 总是低电平。

21.2.2 输入模拟开关

通过使用 P2CAx 位，模拟输入开关可以把两个比较器的输入端连接到或者不连接相关的端口引脚。 可以单独控制比较器的两个输入端子。 P2CAx 位允许：

- 将一个外部信号应用到比较器的+ 和 - 两端子。
- 将一个内部基准电压路由到一个相应的输出端口引脚

在内部，为了抑制信号路径上的失真，输入开关被构造成一个 T 型开关。

注： **比较器输入连接**

当比较器打开时，其输入端子应该连接到一个信号、电源或者接地。 否则，悬空水平会产生意想不到的中断和增加电流消耗。

注： **MSP430G2210**：比较器通道 0，1，3，4被执行，但在器件引脚上不可用。 为了避免悬空输入，不应该使能这些比较器的输入端。

CAEX 位控制输入多路复用器，交换连接到比较器 + 和 - 端子的输入信号。 另外，当比较器两端的信号被交换时，比较器的输出信号会被反转。 这就使得用户可以测定或者补偿比较器输入的偏移电压。

21.2.3 输入短路开关

CASHORT 位短路比较器_A+ 的输入。 它可以用来为比较器建立一个简单的采样保持器，如图 21-2中所示。

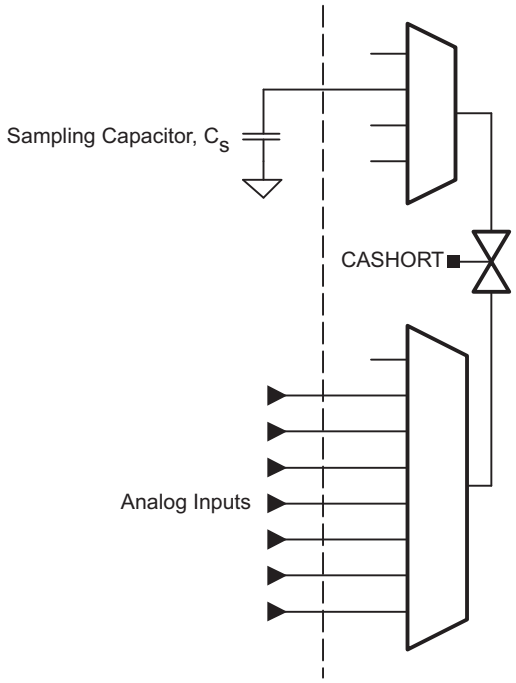


图 21-2. 比较器_A+ 的采样和保持

所需的采样时间与采样电容尺寸(C_S)，与短路开关(R_i) 串联的输入开关电阻，以及外部信号源 (R_S) 的电阻的大小成比例。总内部阻抗 (R_I) 处于典型值范围为 2 到 10kΩ。采样电容 C_S应该大于 100pF。 该时间常量，Tau，要改变采样电容 C_S，可以用以下公式计算：

$$Tau = (R_I + R_S) \times C_S$$

根据需要的精度，应该用 3 到 5 倍的 Tau 作为采样时间。 用3 倍的 Tau 采样电容大约可以对输入信号电压电平进行 95% 的充电，用5 倍的 Tau 可进行大于 99% 的充电，而用 10 倍的 Tau 采样的电压可以充分满足12 位的精度要求。

21.2.4 输出滤波器

比较器的输出可以使用内部的滤波器，也可以不使用。 当控制位 CAF 位被置位时，通过用一个片上电阻电容 (RC) 滤波器来对输出进行滤波。

如果输入端子间的电压差比较小，任何比较器的输出都会振荡。 内部和外部的寄生作用以及信号线、电源线，和系统的其他部分产生的耦合都会导致图 21-3中的行为。 比较器输出的振荡会降低比较结果的精度和分辨率。 选择输出滤波器可以减少与比较器振荡相关的错误。

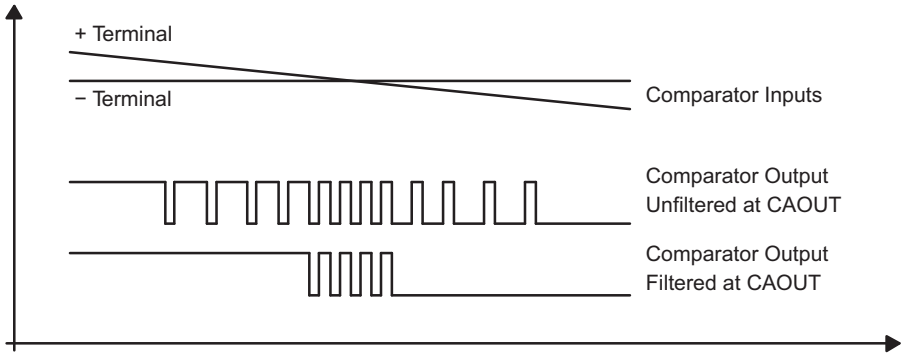


图 **21-3.** 在比较器输出端的 **RC** 滤波器响应

21.2.5 基准电压发生器

基准电压发生器用来产生电压V_{CAREF}，该电压被提供给比较器的输入端。CAREFx 位控制电压发生器的输出。 CARSEL 位选择比较器端子使用哪个 V_{CAREF}。 如果比较器的两个输入端都使用外部输入信号，那么为了降低电流的消耗，应该关闭内部基准电压。 该基准电压发生器能够产生器件的V_{CC}的一小部分或一个固定的 ～0.55V 的晶体管的阈值电压。

21.2.6 比较器A+，端口禁用寄存器CAPD

比较器的输入和输出功能和相关的 I/O 端口引脚复合使用，这些引脚都是数字 CMOS 门。 当模拟信号被加载到数字 CMOS 门时，会产生从 VCC 流向 GND 的寄生电流 如果输入电压接近门的转换电压，就会出现寄生电流。 禁止端口引脚缓冲能够消除寄生电流的流动，从而可以减少整个电流的消耗。

CAPDx 位，当被置位时，会禁用相应的引脚端口输入和输出缓冲，如图 21-4所示。 当电流消耗非常重要时，任何连接到模拟信号的端口引脚都应该通过其 CAPDx 位禁止。

通过 P2CAx 位为比较器复用器选择输入引脚时，不管其对应的 CAPDx 位状态如何，该引脚的输入和输出缓冲都会自动禁止。

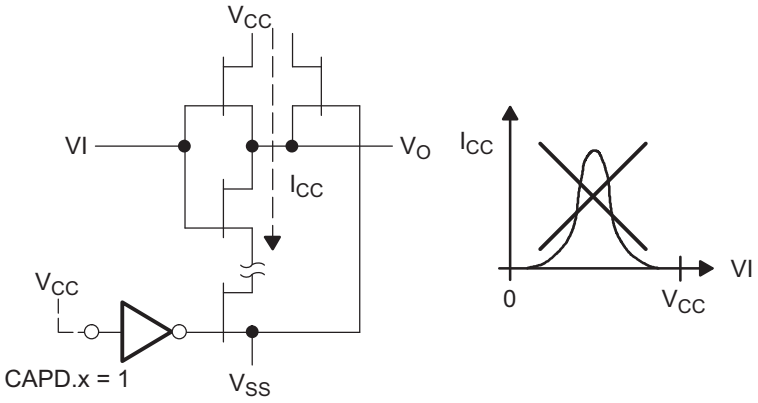


图 **21-4.** 一个 **CMOS** 反相器**/**缓冲器中的传输特性和功耗

注: **MSP430G2210**：通道 0，1，3 和 4由引脚上不可用来执行。 为了避免输入悬空，不应该使用这些输入。

21.2.7 比较器_A+ 的中断

如在图 21-5中所示，一个中断标志一个中断向量都与比较器_A+ 相关。 比较器输出端的上升沿或下降沿都会使中断标志 CAIFG 置位，该中断可以由 CAIES 位选择。 如果 CAIE 和 GIE 同时被置位，那么CAIFG 会产生一个中断请求。 当中断请求被响应时，CAIFG 位会自动复位，也可以通过软件复位。

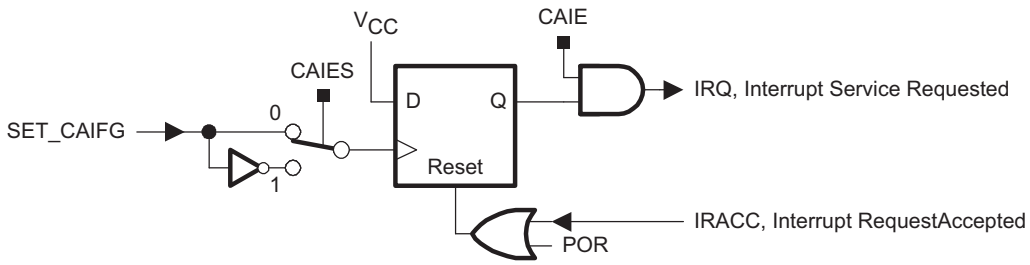


图 **21-5.** 比较器_A+ 的中断系统

21.2.8 比较器_A+ 用于测量电阻元件

为了精确地测量电阻元件，可以通过使用单一的斜率模数转换对比较器_A+ 进行优化。 例如，通过比较热敏电阻电容和一个基准电阻电容的放电时间，该基准电阻如在图 21-6中所示，可利用热敏电阻把温度转换成数字信号。把一个基准电阻 Rref 与 Rmeas 进行比较。

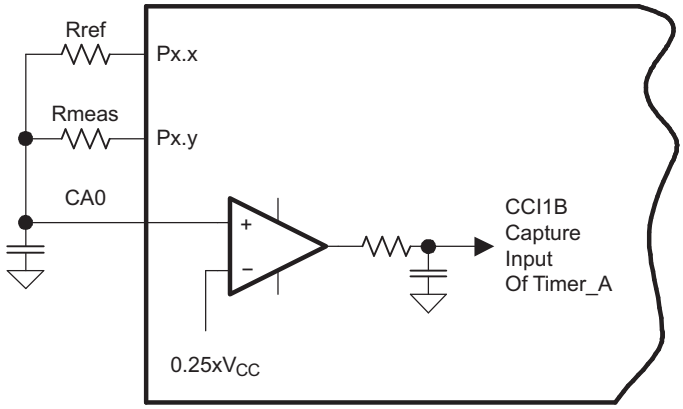


图 **21-6.** 温度测量系统

下面是通过 Rmeas 计算温度传感所使用的 MSP430 的资源：

- 用来对电容进行充电和放电的两个数字 I/O 端口。
- 置位 I/O 输出高电平 (V_{CC})对电容进行充电，复位则对其放电。
- 当不使用 I/O 时，通过置位 CAPDx 位使其切换到高阻态输出。
- 一个输出通过 Rref 对电容进行充放电。
- 一个端口通过 Rmeas 对电容进行充放电。
- + 端子被连接到比较器的正端。
- - 端子被连接到一个基准电平，例如 0.25 xV_{CC}。
- 使用输出滤波器最小化开关噪声。
- CAOUT 使用门控制定时器_A CCI1B，捕获电容放电时间。

可以测量一个以上的电阻元件。 通过可用的 I/O 引脚，把额外的元件连接到CA0，且当不进行测量时，可把其切换到高阻抗。

热敏电阻的测量是基于一个比例转换原理的。 两个电容放电时间的比率的计算显示在图 21-7中。

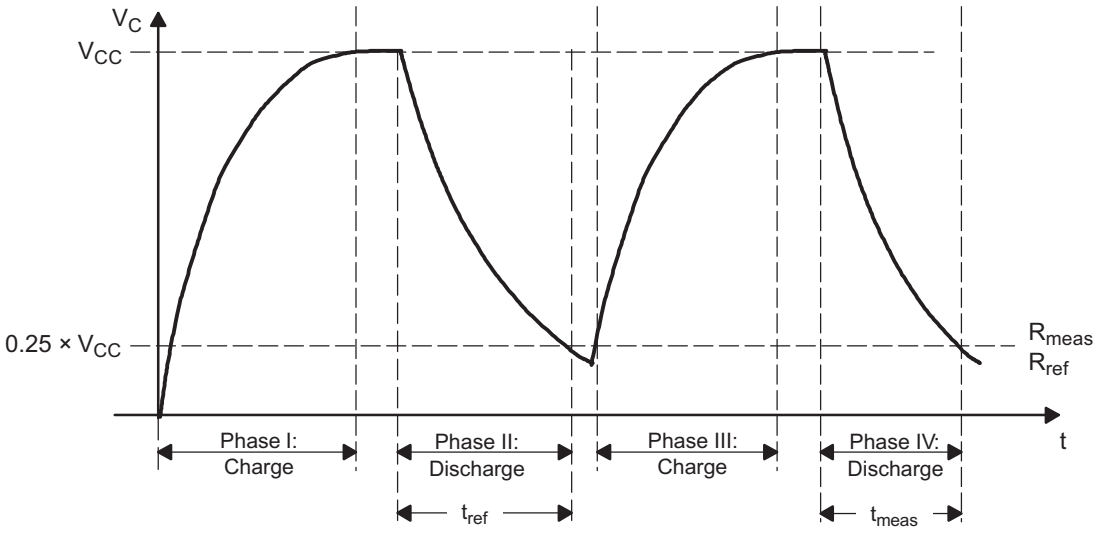


图 21-7. 温度测量系统的时序

在转换期间，V_{CC}电压和电容值应当保持恒定，但这不是很关键，因为在比率中它们可以抵消：

$$\frac{N_{meas}}{N_{ref}} = \frac{-R_{meas} \times C \times \ln \frac{V_{ref}}{V_{CC}}}{-R_{ref} \times C \times \ln \frac{V_{ref}}{V_{CC}}}$$

$$\frac{N_{meas}}{N_{ref}} = \frac{R_{meas}}{R_{ref}}$$

$$R_{meas} = R_{ref} \times \frac{N_{meas}}{N_{ref}}$$

21.3 比较器_A+ 寄存器

在表 21-1中列出了比较器_A+ 寄存器。

表 21-1. 比较器_A+ 寄存器

寄存器	简表	寄存器类型	地址	初始化状态
比较器_A+ 控制寄存器 1	CACTL1	读取/写入	059h	用 POR 复位
比较器_A+ 控制寄存器 2	CACTL2	读取/写入	05Ah	用 POR 复位
比较器_A+ 端口禁用	CAPD	读取/写入	05Bh	用 POR 复位

21.3.1 CACTL1，比较器_A+ 控制寄存器1

7	6	5	4	3	2	1	0
CAEX	**CARSEL**	**CAREFx**		**CAON**	**CAIES**	**CAIE**	**CAIFG**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

CAEX 位 7 比较器_A+ 的交换。 该位交换比较器的输入和反转比较器的输出。

CARSEL 位 6 比较器_A+ 的基准电压选择。 该位选择V_{CAREF}被用于哪一端。

当 CAEX= 0 时：

0 V_{CAREF}被应用于 + 端子。

1 V_{CAREF}被应用于 - 端子。

当 CAEX= 1 时：

0 V_{CAREF}被应用于 - 端子。

1 V_{CAREF}被应用于 + 端子。

CAREF 位 5-4 比较器_A+ 的基准电压 这些位选择基准电压V_{CAREF}。

00 内部基准电压关闭 使用一个外部基准电源。

01 0.25 ×V_{CC}

10 0.50 ×V_{CC}

11 二极管基准电压被选择

CAON 位 3 比较器_A+ 开启。 该位可以打开比较器。 当比较器关闭的时，它不消耗电流。 其参考电路可以被独立使能或禁止。

0 关闭

1 打开

CAIES 位 2 比较器_A+ 的中断沿选择

0 上升沿

1 下降沿

CAIE 位 1 比较器_A+ 的中断使能

0 被禁用

1 被启用

CAIFG 位 0 比较器_A+ 的中断标志

0 无中断等待

1 中断等待

21.3.2 CACTL2，比较器_A+，控制寄存器

7	6	5	4	3	2	1	0
CASHORT	**P2CA4**	**P2CA3**	**P2CA2**	**P2CA1**	**P2CA0**	**CAF**	**CAOUT**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r-(0)

CASHORT	位 7	输入短路。 该位将输入的 + 和 - 端短路。 0 输入没被短路 1 输入被短路
P2CA4	位 6	输入选择。 该位结合 P2CA0，在 CAEX=0时选择 + 端输入，在 CAEX=1 时选择 - 端输入。
P2CA3[1] **P2CA2** **P2CA1**	位 5-3	输入选择。 这些位在 CAEX=0时选择 - 端输入，在 CAEX=1 时选择 + 端输入。 000 无连接 001 CA1 010 CA2 011 CA3 100 CA4 101 CA5 110 CA6 111 CA7
P2CA0	位 2	输入选择。 该位结合 P2CA4，在 CAEX=0 时选择 + 端输入，在 CAEX= 1 时选择 - 端输入。 00 无连接 01 CA0 10 CA1 11 CA2
CAF	位 1	比较器_A+ 的输出滤波器 0 比较器_A+ 的输出没被滤波 1 比较器_A+ 的输出被滤波
CAOUT	位 0	比较器_A+ 的输出 该位反映比较器输出的值。 对该位进行写入操作无效。

(1) **MSP430G2210**：只有通道2，5，6，和 7 是可用的。 其他通道不应该被选择。

21.3.3 CAPD，比较器_A+，端口禁用寄存器

7	6	5	4	3	2	1	0
CAPD7	**CAPD6**	**CAPD5**	**CAPD4**	**CAPD3**	**CAPD2**	**CAPD1**	**CAPD0**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

CAPDx[1]	位 7-0	比较器_A+ 端口禁用。 这些位可以独立地禁用与比较器_A+ 相关的针对端口引脚的输入缓冲区。 例如，如果 CA0 是连在引脚 P2.3 上的，可以用 CAPDx 位独立地启用或禁用每个 P2.x 的引脚缓冲器。 CAPD0 禁用 P2.0，CAPD1禁用 P2.1，等等。 0 该输入缓冲器被启用。 1 该输入缓冲器被禁用。

(1) **MSP430G2210**：通道 2，5，6，和 7 是可用的。 其他通道不应该被禁用。

ADC10

ADC10 模块是一个高性能的 10 位模数转换器。 本章概括的描述了 2xx 系列中 ADC10 模块的运行。 器件中有不到 8 个外部输入通道。

Topic **Page**

22.1 ADC10 介绍

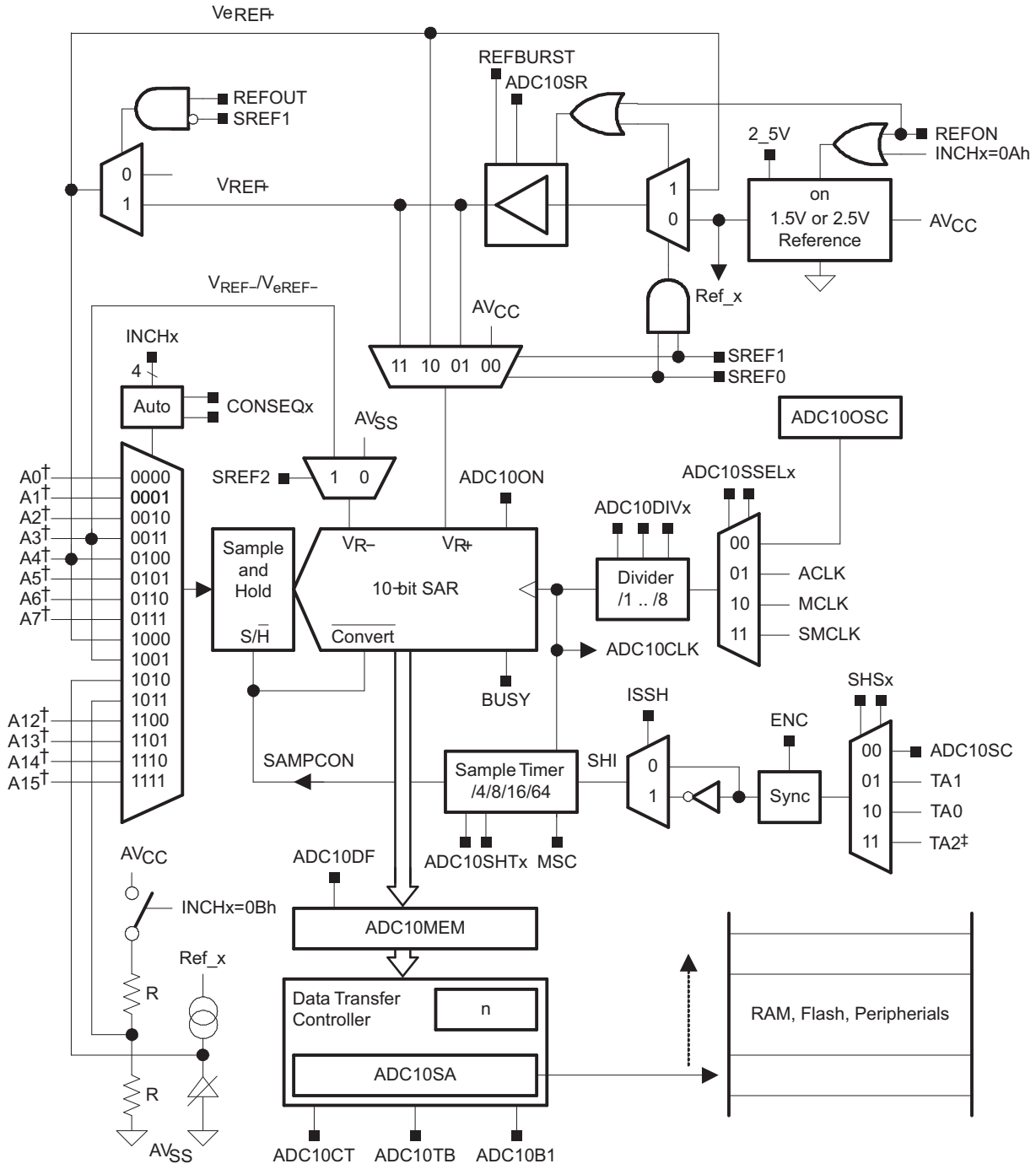
ADC10 模块支持快速，10 位模数转换。 该模块应用了一个 10 位逐次逼近 (SAR) 内核，采样选择控制，基准电压产生器和数据传递控制器 (DTC)。

DTC 允许 ADC10 样本被转换和存储在任何其它寄存器单元而无需 CPU的干预。 模块可以经过用户软件配置后支持不同的应用。

ADC10 模块特征如下：

- 大于 200ksps 的最大转换速率
- 无失码的单片 10 位转换器
- 带有可编程采样周期的采样保持功能
- 通过软件或 定时器_A 初始化转换
- 软件可选片上基准电压（1.5V 或 2.5V）
- 软件可选内部或外部基准电压
- 高达 8 个外部输入通道（MSP430F22xx 器件上 12 个）
- 内部温度传感器的转换通道，V_{CC}，和外部基准电压
- 可选的转换时钟源
- 单通道单次，单通道多次，序列通道单次和序列通道多次转换模式
- ADC 内核和基准电压都可以独立关闭
- 自动存储转换结果的数据转换控制器

ADC10 模块的芳框图如图 22-1所示。



†Channels A12-A15 are available in MSP430F22xx devices only. Channels A12-A15 tied to channel A11 in other devices. Not all channels are available in all devices.
‡TA1 on MSP430F20x2, MSP430G2x31, and MSP430G2x30 devices

图 **22-1. ADC10** 方框图

22.2 ADC10 的运行

ADC10 模块可由用户软件配置。 ADC10 的运行和建立在下列章节中进行讨论。

22.2.1 10 位 ADC 内核

ADC 内核将一个模拟量的输入转化成 10 位数字形式，结果保存在 ADC10MEM 寄存器中。 内核利用两个可编程/可选择的基准电平（V_{R+}和 V_{R-}）来定义转换范围的最大值和最小值。 当输入信号等于或高于 V_{R+} 时，数字输出 (N_{ADC}) 为满量程（03FF），当输入信号等于或低于 V_{R-}时，输出为零。 输入通道和基准电平（V_{R+}和 V_{R-}）在转换控制寄存器中进行了定义。 转换的结果为直接的二进制形式或二的补码形式。 在使用直接二进制形式时，ADC 结果的转换公式为：

$$N_{ADC} = 1023 \times \frac{V_{IN} - V_{R-}}{V_{R+} - V_{R-}}$$

ADC10 内核由 ADC10CTL0 和 ADC10CTL1 两个控制寄存器完成配置。 内核使能由 ADC10ON 位控制。大多数情况下，只有在 ENC=0 时，ADC10 的控制位才可以被修改。在进行任何转换前 ENC 位必须设为 1。

22.2.1.1 转换时钟选择

ADC10CLK 既可以作为转换时钟也可用于产生采样周期。 ADC10 源时钟可以用 ADC10SSELX 位来选择，也可以由 ADC10DIVx 位进行 1 至 8 分频。 可选的 ADC10CLK 源有 SMCLK，MCLK，ACLK 和内部的振荡器ADC10OSC。

ADC10OSC 由内部产生，在 5MHz 范围内，但会随着器件本身，供电电压，和温度而改变。 对于 ADC10OSC说明，请参阅《器件专用数据表》。

用户必须保证在转换结束前所选择的 ADC10CLK 都保持在活动状态。 如果在转换期间时钟被去除，转换将无法完成，并且结果无效。

22.2.2 ADC10 输入和多路器

模拟输入多路器可以选择 8 个外部和 4 个内部模拟信号接口作为转换通道。 输入模拟多路器是先关后开型开关以此来减少因通道切换而引入的噪声（请见图 22-2）。 输入模拟多路器也是一个 T 型开关，可以减少通道间的耦合。 未选择的通道与 A/D 分开，并且中间接点被连接到模拟接地 (V_{SS}) 端以便于寄生电容接地从而减少噪声。

ADC10 利用电荷再分配原理。 当输入在内部切换时，切换动作可能在输入信号上引起瞬变。 这些瞬变衰减和解决之前会导致错误的转换。

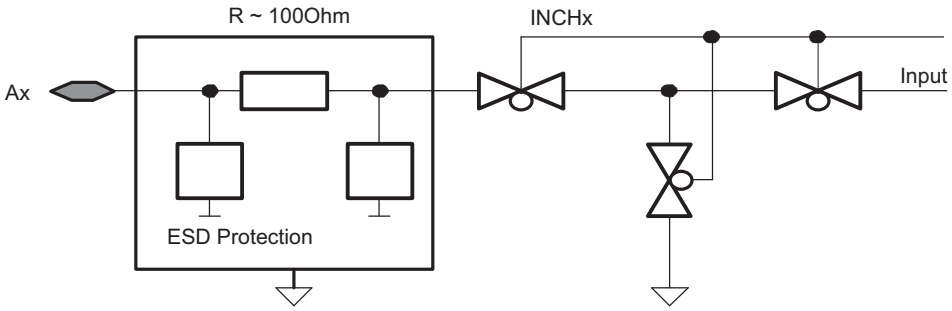


图 22-2. 模拟多路复用器

22.2.2.1 模拟端口选择

ADC10 外部输入 Ax，V_{eREF+}，和V_{REF-}和通用 I/O 端口共享端子，通用端口是数字 CMOS栅极（请参阅《器件专用数据表》）。 当模拟信号被应用到数字 CMOS 栅极时，寄生电流可以从 VCC 流向 GND。 如果输入电压接近该门的转换电平时，就会产生寄生电流。 禁用端口引脚缓冲器防止了寄生电流流过，因此可降低总电流消耗。 ADC10AEx 位提供了禁用端口引脚的输入和输出缓冲器的功能。

```
; P2.3 on MSP430F22xx device configured for analog inputBIS.B #08h,&ADC10AE0 ; P2.3 ADC10
function and enable
```

没有全部 ADC10 外部输入通道 Ax 或没有在器件引脚可用的V_{eREF+}/V_{REF+}和V_{eREF-}/$_{VREF-}$的器件不得更改不可用引脚的默认寄存器位的配置。 请参阅《器件专用数据表》。

22.2.3 基准电压产生器

ADC10 模块包含一个内置的电压基准带有两个可选的电压电平。 设置 REFON=1 使能内部基准。 当 REF2_5V=1 时，内部基准是 2.5V。当 REF2_5V=0 时，基准是 1.5V。内部基准电压可用于内部 (REFOUT=0) 并且，当REFOUT=1 时，引脚上外部地 V_{REF+}。如果引脚 V_{REF+}和V_{REF-}可用，REFOUT=1 应仅用于器件引脚。

外部基准电压可以分别通过引脚 A4 和 A3 应用于 V_{R+}和 V_{R-}。 当外部基准电压被使用时，或当V_{CC}被用作基准电压时，可以关闭内部基准电压以减少功耗。

一个外部正基准电压 V_{eREF+}可以通过设置SREF0=1 和 SREF1=1（仅适于带有 V_{REF+}引脚的器件）被缓冲。这就在缓冲电流的成本上允许了使用一个带有大的内部电阻的外部基准电压。 当REFBURST=1 时，增加的电流消耗受采样和转换周期的限制。

正如在ADC12 上一样，ADC10 基准源同样不需要外部存储电容。

22.2.3.1 内部参考电压低功耗特性

ADC10 的内部基准电压产生器是为低功耗应用而设计的。 该基准电压产生器包括一个带隙电源和一个独立的缓冲器。 每个器件的电流消耗在《特定的器件数据手册》中分别有详细说明。 当 REFON=1 时，两者都被启用并且当 REFON=0 时，二者都被禁用。 当 REFON 变为 1 时 的总设定时间约为 30μs。

当 REFON=1,所有转换都被禁用，缓冲器自动禁用并且在需要进行转换时自动重新使能。 当缓冲器被禁用时，无电流消耗。 这种情况下，带隙电源保持使能。

当 REFOUT=1 时，REFBURST 位控制内部基准缓冲器的运行。 当 REFBURST=0 时，缓冲器持续打开，允许基准电压持续存在于器件外部。 当 REFBURST=1 时，缓冲器自动被禁用ADC10 模块不进行活动转换并且需要时，自动重新打开。

内部基准缓冲器也可以对转换速度和功耗设置进行选择。 当最大转换率低于 500ksps 时，设置 ADC10SR=1 可以减少约 50% 的缓冲器的电流消耗。

22.2.4 自动关断

ADC10 是为低功耗应用而设计的。 当 ADC10 没有活动转换时，内核自动被禁用并且在需要时会自动重新使能。 该ADC10OSC 同样也是在需要时，自动使能而在不需要时，自动禁用。 当内核或振荡器被禁用时，无电流消耗。

22.2.5 采样和转换时序

一个数模转换由一个采样输入信号SHI 的上升沿启动。 SHI 信号源可以通过 SHSx 位来选择，包括如下：

- ADC10SC 位
- 定时器_A 输出单元 1

- 定时器_A 输出单元 0
- 定时器_A 输出单元 2

SHI 信号源的极性可以通过 ISSH 位来反转。 SHTx 位可以选择采样周期 $t_{采样}$为 4，8，16 或 64 个 ADC10CLK 周期。 在与ADC10CLK 同步后，采样定时器为选择的采样周期设置 SAMPCON 为高。总采样时间是 $t_{采样}$加上$t_{同步}$。SAMPCON 由高到低变化时开始模数转换，该转换需要 13 个 ADC10CLK 周期，如图 22-3所示。

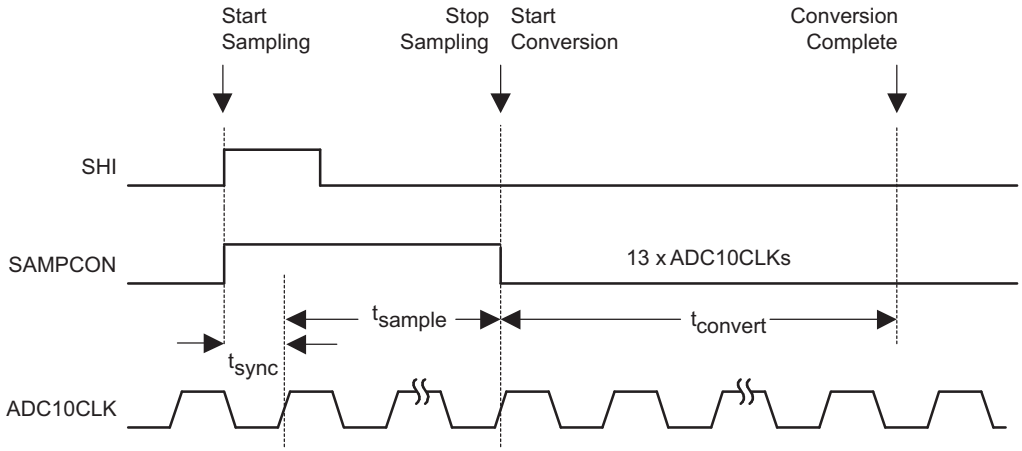


图 **22-3.** 采样时序

22.2.5.1 采样时序注意事项

当 SAMPCON=0 时，所有的 Ax 输入均为高阻态 当 SAMPCON=1 时，在采样时间$t_{采样}$期间，被选择的 Ax 输入相当于一个 RC 低通滤波器，如图 22-4所示。 一个内部 MUX 上输入电阻 R_I（最大2kΩ）被认为是由源极和电容器 C_I（最大 27pF）串连在一起。电容器 C_I电压 V_C必须被充电至电源电压 V_S的 ½ LSB 范围内，以此来进行精度为 10 位的转换。

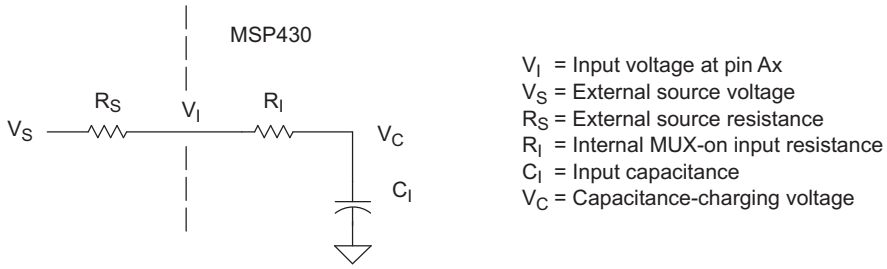


图 **22-4.** 模拟输入等效电路

源 R_S和 R_I的电阻影响 $t_{采样}$。下列公式可被用于计算一个 10 位转换的最小采样时间。

$$t_{采样} > (R_S + R_I) \times \ln(2^{11}) \times C_I$$

R_I和 C_I的代入值在上面已经给出，公式变为：

$$t_{采样} > (R_S + 2\ k\Omega) \times 7.625 \times 27pF$$

例如，如果 R_S是 10kΩ，那么 $t_{采样}$必须大于 2.47μs。

当基准缓冲器被用于突发模式时，采样时间必须大于计算出的采样时间和缓冲器的稳定时间$t_{REFBURST}$：

$$t_{sample} > \begin{cases} (R_S + R_I) \times \ln(2^{11}) \times C_I \\ t_{REFBURST} \end{cases}$$

例如，当 ADC10SR=0 时, 如果 V_{Ref}是 1.5V 和 R_S是10kΩ, $t_{采样}$必须大于 2.47μs，或当ADC10SR=1 时，采样时间必须大于2.5μs 参数请参阅《器件专用数据表》。

当使用一个外部基准电压时，为了计算缓冲器的稳定时间，使用的公式是：

$$t_{REFBURST} = S_R \times V_{Ref} - 0.5\ \mu s$$

其中：

SR = 缓冲器转换率（当 ADC10SR=0 时，为 ~1 μs/V 和当 ADC10SR=1 时，为 ~2 μs/V）

V_{Ref}=外部基准电压

22.2.6 转换时间

ADC10 有四个可由 CONSEQx 位选择的运行模式，在表 22-1中进行讨论。

表 22-1. 转换模式概述

CONSEQx	模式	运行
00	单通道单次转换	一个单通道被转换一次
01	通道序列	一个通道序列被转换一次
10	单通道重复转换	一个单通道被重复转换
11	通道序列重复转换	一个通道序列被重复转换

22.2.6.1 单通道单次转换模式

一个 INCHx 选择的单通道被采样和转换一次。 ADC 结果被写入 ADC10MEM。图 22-5显示了单通道单次转换模式的流程。 当ADC10SC 触发一次转换时，连续的转换可有 ADC10SC 位来触发。 当使用任何其它触发源时，必须在每次转换间切换 ENC。

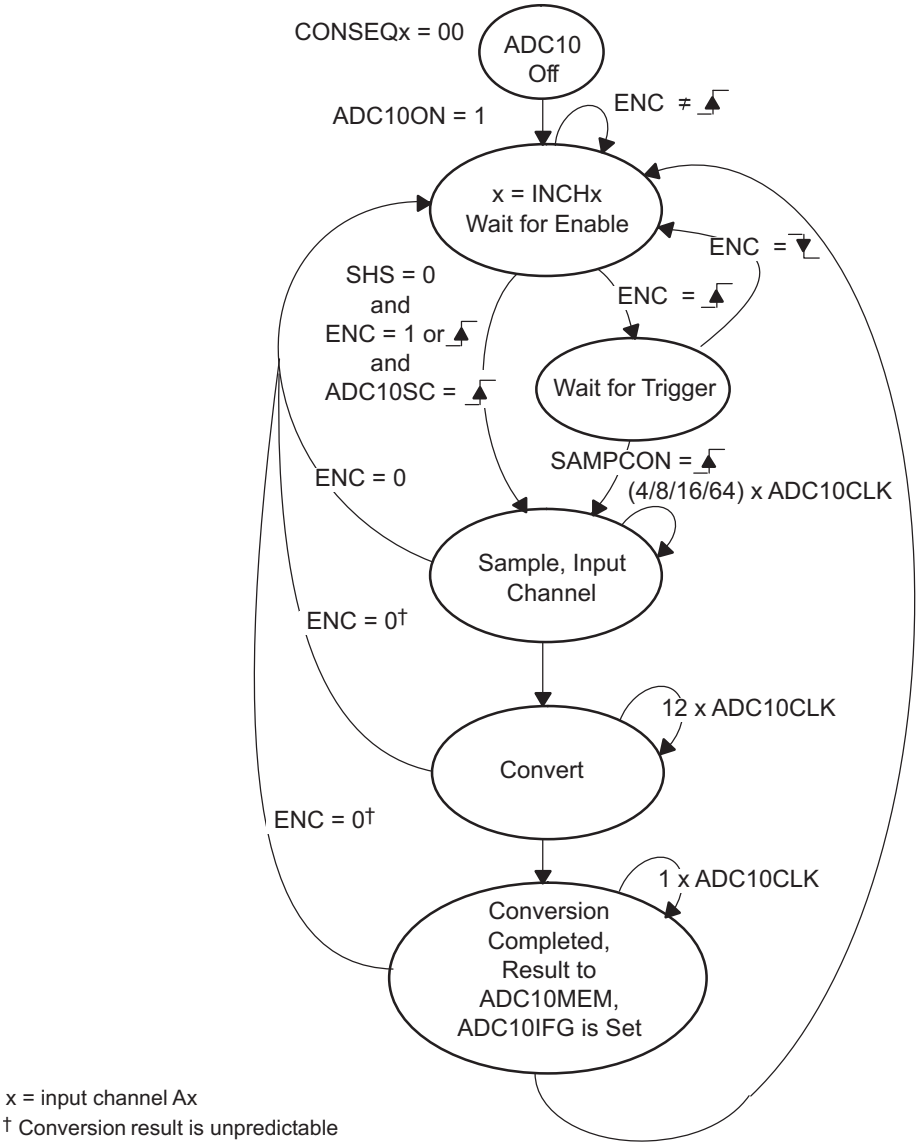


图 **22-5.** 单通道单次转换模式

22.2.6.2 通道序列模式

一个通道序列被采样和转换一次 序列从 INCHx 选择的通道开始并且递减到通道 A0。 每一个 ADC 结果都被写入 ADC10MEM。 该序列在通道 A0 转换后停止。图 22-6显示了通道序列模式。 当 ADC10SC 触发一个序列时，连续的序列可有 ADC10SC 位来触发。 当使用任何其它触发源时，必须在每个序列间切换 ENC。

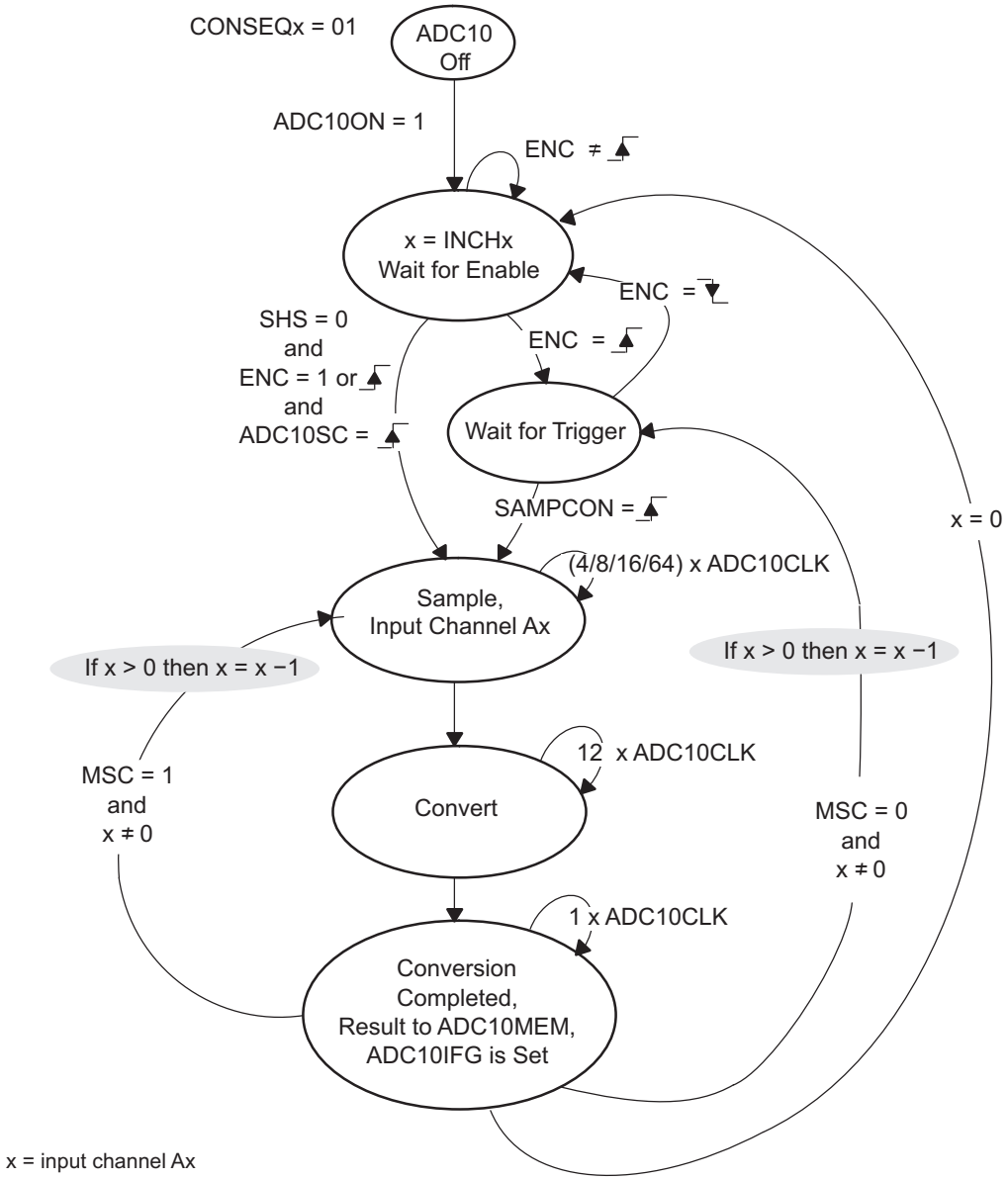


图 **22-6.** 通道序列模式

22.2.6.3 单通道重复模式

一个 INCHx 选择的单通道被连续采样和转换。 每个 ADC结果都被写入 ADC10MEM。图 22-7显示了单通道重复模式。

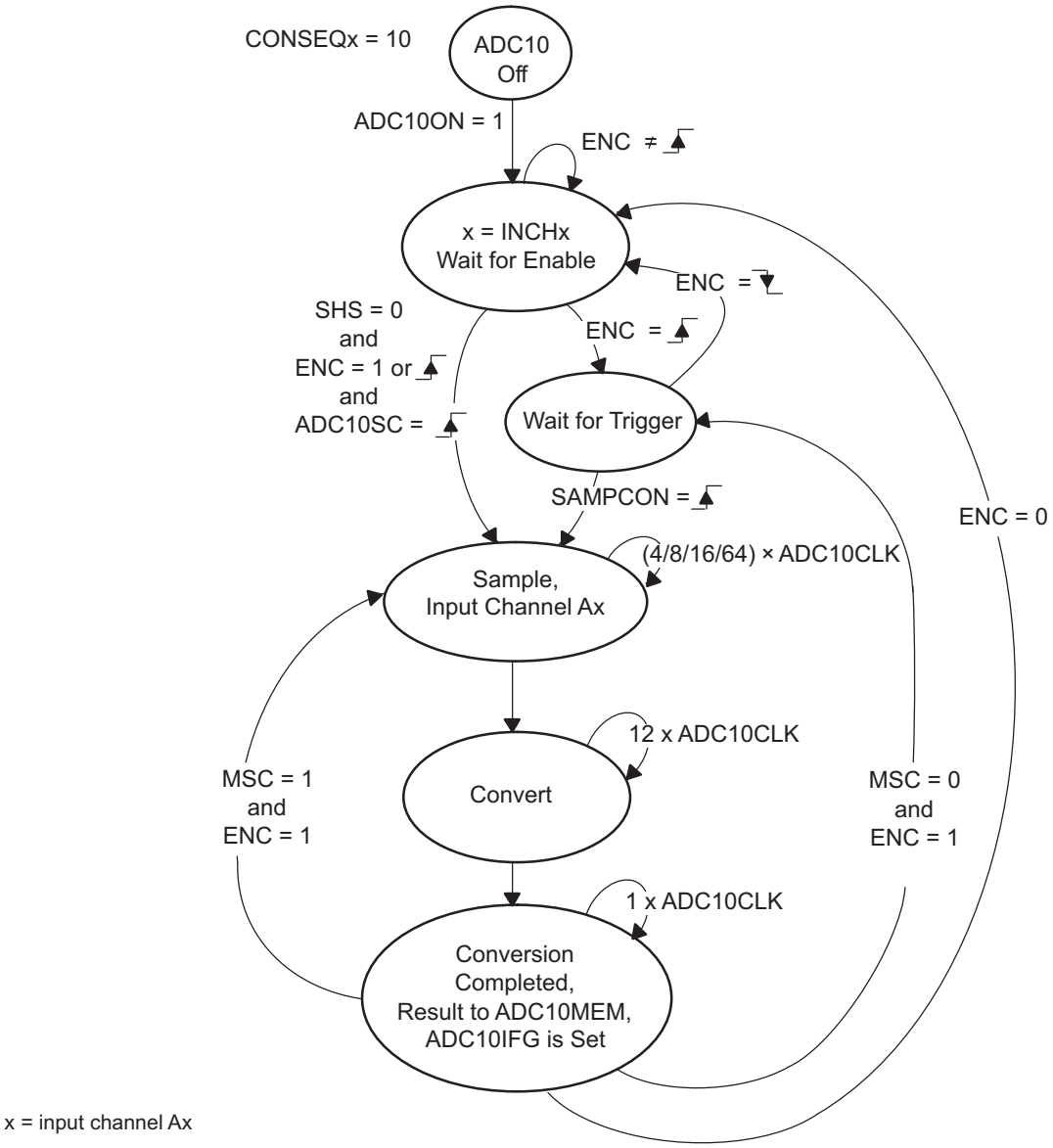


图 **22-7.** 单通道重复模式

22.2.6.4 通道的重复序列模式

一个通道序列被重复采样和转换。 序列从 INCHx 选择的通道开始并且递减到通道 A0。 每一个 ADC 结果都被写入 ADC10MEM。序列在通道 A0 转换后结束，并且下一个触发信号重启序列。图 22-8显示了通道的重复序列模式。

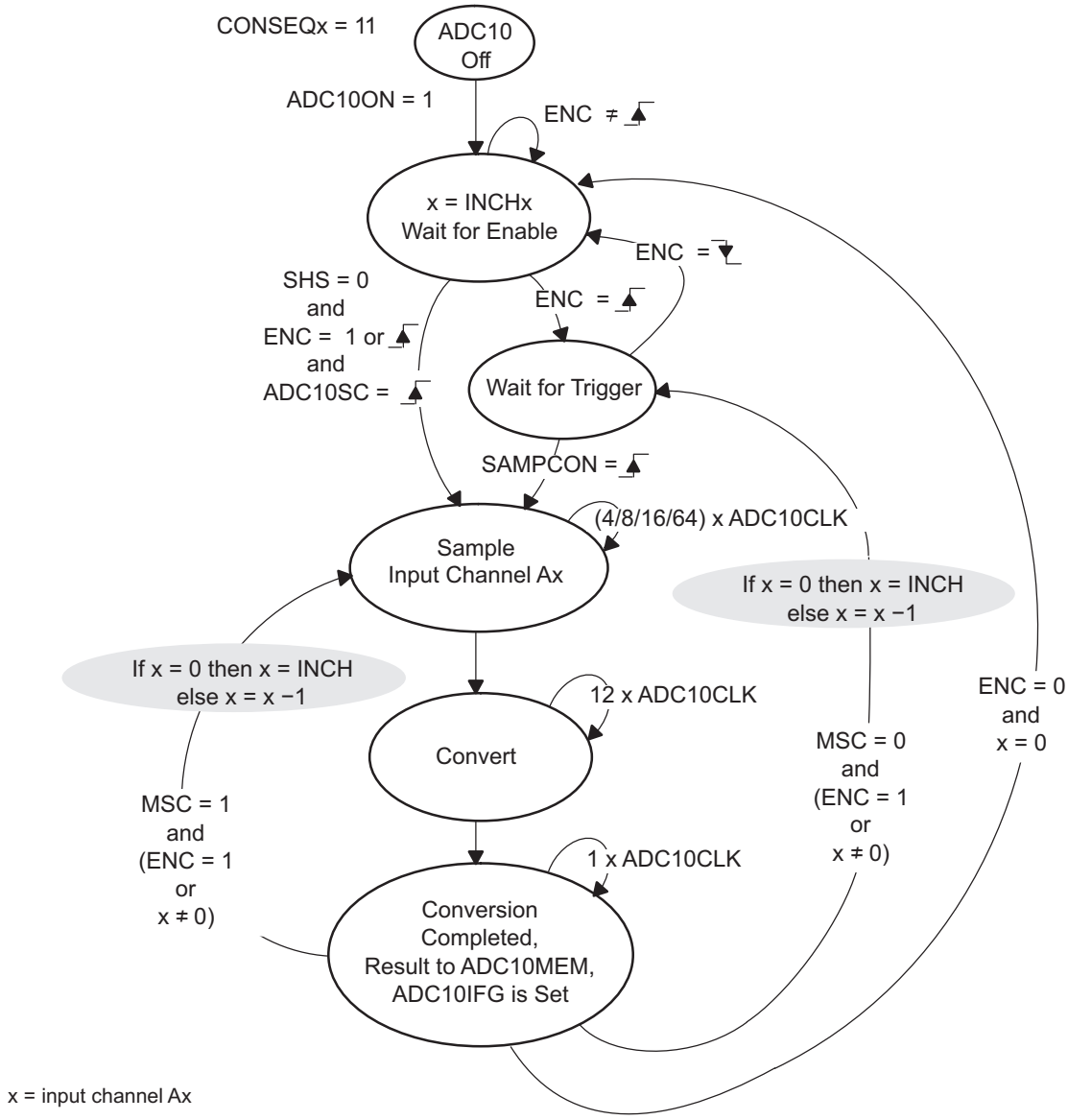


图 **22-8.** 通道的重复序列模式

22.2.6.5 使用 MSC 位

为了配置转换器以使其能自动的执行连续转换并且尽可能的快速，可以使用一个多路采样和转换功能。 当 MSC=1 且 CONSEQx>0 时，SHI 信号源的第一上升沿首先转换。 转换完成前，连续转换被快速并自动的触发。 在单序列模式中，序列完成前或在单通道重复模式或重复序列中，ENC 位被切换前，SHI 上的其他上升沿都被忽略。 当使用 MSC 位时，ENC 位的功能不能更改。

22.2.6.6 停止转换

停止 ADC10 活动取决于运行模式。 建议的停止一个活动转换或转换序列的方法是：

- 在单次转换模式中复位 ENC 可以立即停止一个转换并且结果是不可预知的。 想要获得正确的结果，在清除 ENC 前应轮询 ADC10BUSY 位直到复位开始。
- 在单通道重复运行器件复位 ENC 可以在当前转换结束时停止转换。
- 在一个序列或重复序列中复位 ENC 可以在序列结束时停止转换。
- 通过设置 CONSEQx=0 和复位 ENC 位可以立即停止任何转换模式。 转换数据不可信。

22.2.7 ADC10 数据传输控制器

ADC10 包括一个数据传输控制器 (DTC) 来自动传输从 ADC10MEM 到其他片上存储器位置的转换结果。 通过把ADC10DTC1 寄存器设置为一个非零值来使能 DTC。

当 DTC 被启用时，每次 ADC10 完成一个转换和下载结果到 ADC10MEM 时，一个数据传输都会被触发。预订量的转换数据被传输前，无需软件干预来管理ADC10。 每个 DTC 传输需要一个 CPU MCLK。 在 DTC 传输期间，为了避免总线内容，CPU 应该被暂停，如果没被暂停，传输器需要一个 MCLK。

在 ADC10 占用期间，不应启动一个 DTC 传输器。 当 DTC 被配置时，软件必须保证没有活动的转换或序列正在进行。

```
; ADC10 activity testBIC.W #ENC,&ADC10CTL0 ;busy_test BIT.W #BUSY,&ADC10CTL1 ;JNZ busy_test
;MOV.W #xxx,&ADC10SA ; SafeMOV.B #xx,&ADC10DTC1 ;; continue setup
```

22.2.7.1 一个数据块传输模式

如果 ADC10TB 被复位，那么一个块传输模式将会被选用。 对于一个数据块，ADC10DTC1 中的 n 值定义了传输的总数量。 可以利用 16 位寄存器 ADC10SA 在MSP430 任何地址范围内定义块的起始地址。 块结束地址为 ADC10SA+2n-2。该一个块传输模式如图 22-9所示。

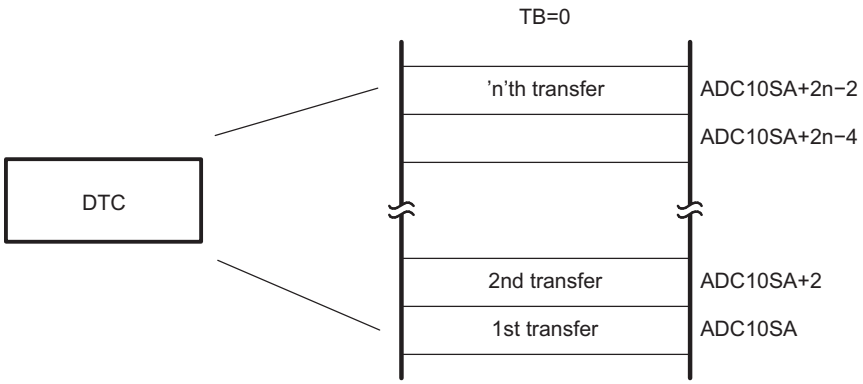


图 **22-9.** 一个块传输

内部地址指针最初为 ADC10SA，内部传输计数器等于 'n'。 内部指针和计数器对软件是不可见的。DTC 传输 ADC10MEM 的字值到地址指针 ADC10SA。 每次 DTC传输后，内部地址指针增加 2 同时内部传递计数器减 1。

DTC 随着 ADC10MEM 的每次装载连续传递，直到内部传递计数器变为 0。 直到一个值被写入 ADC10SA，DTC 才会停止其他的数据传递。 当在一个数据块模式中使用DTC 时，ADC10IFG 标志仅会在一个完整的块被传输后才会被设置。图 22-10显示了一个块模式的状态图表。

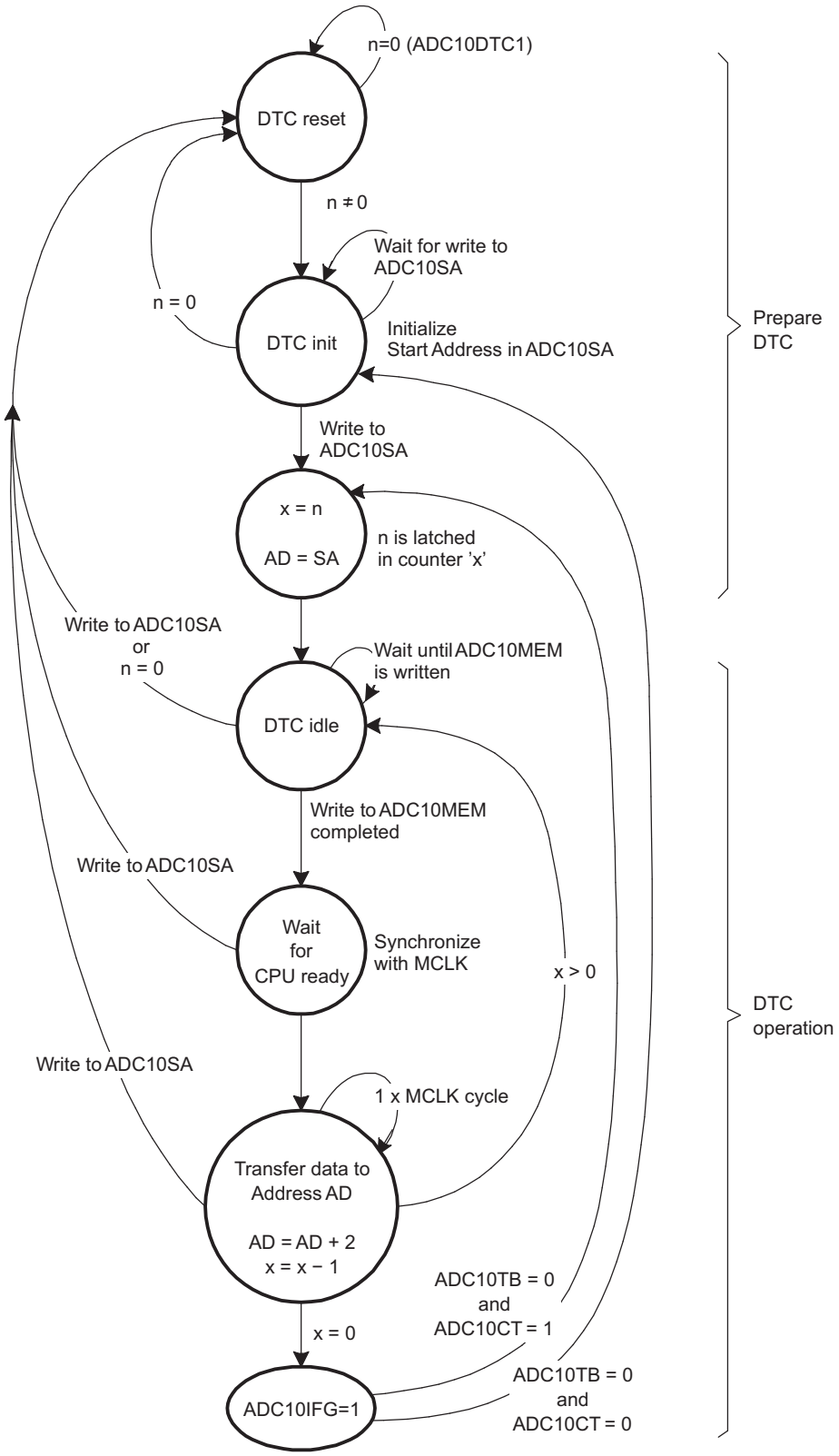


图 **22-10.** 在一个块传输模式中的数据传输控制状态图表

22.2.7.2 两个块传输模式

如果 ADC10TB 被设置，那么两个块传输模式将会被选用。 对于一个块，ADC10DTC1 中的 n 值定义了传输的数量。 可以利用 16 位寄存器 ADC10SA 在MSP430 任何地址范围内定义第一个块的地址范围。 第一个块结束地址为ADC10SA+2n-2。 第二个块的地址范围被定义为 SA+2n 到 SA+4n-2。 两个块传输模式如图 22-11所示。

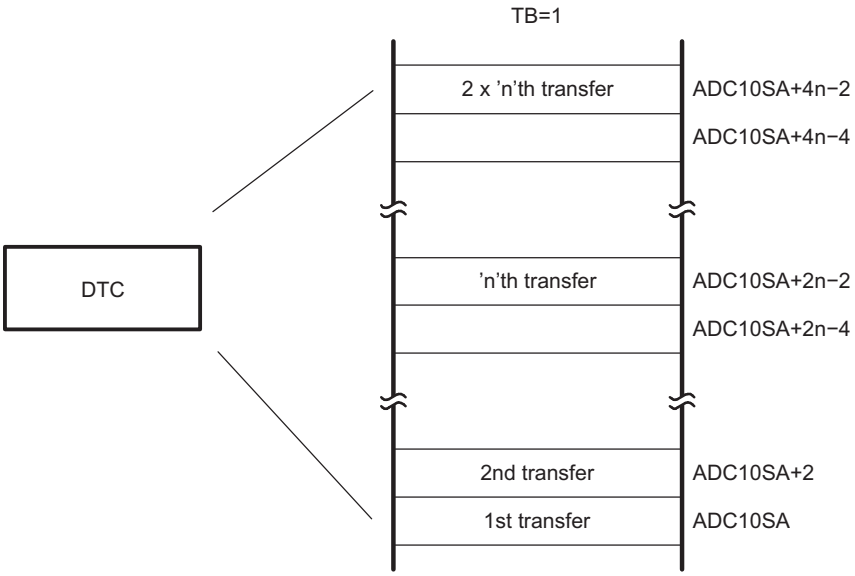


图 22-11. 两个块传输

内部地址指针最初为 ADC10SA，内部传递计数器等于 'n'。 内部指针和计数器对软件是不可见的。DTC 传递 ADC10MEM 的字值到地址指针 ADC10SA。 每次 DTC传递后，内部地址指针增加 2 同时内部传递计数器减 1。

DTC 随着 ADC10MEM 的每次装载连续传递，直到内部传递计数器变为 0。 在这时，块一已装满，ADC10IFG 标志ADC10B1 位都被设置 用户可以通过测试 ADC10B1 位来判断块一是否已装满。

DTC 继续传递块 2。 内部传递计数器自动重装'n' 在下次装载 ADC10MEM 时，DTC 开始传递转换结果到块 2。 完成 n 次传递后，块 2 被装满。 ADC10IFG 标志被设置和ADC10B1 位被清除。 用户软件可以通过测试 ADC10B1 位是否清除来判断块 2是否装满。图 22-12显示了两个块模式的状态图表。

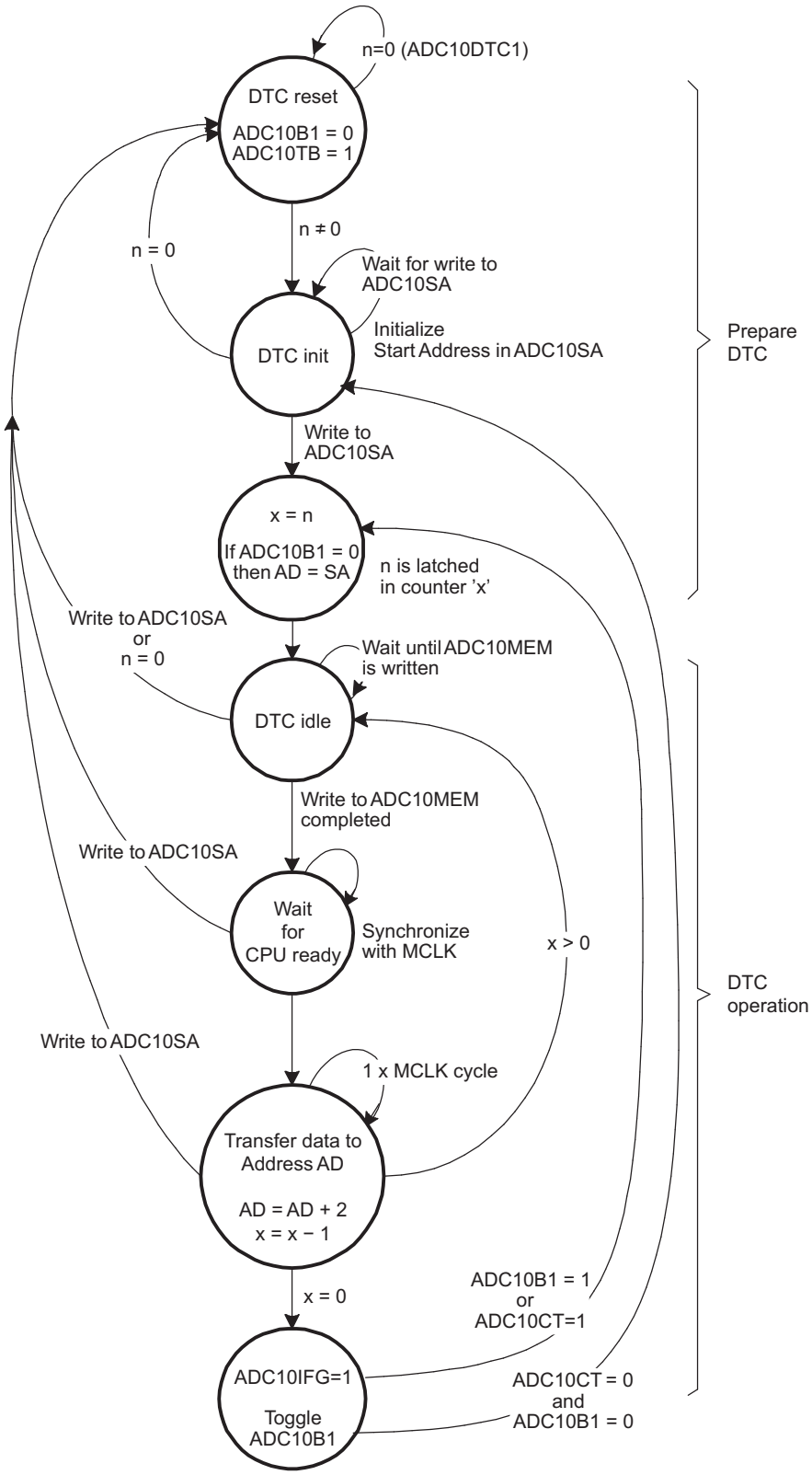


图 **22-12.** 在两个块传输模式中的数据传输控制状态图表

22.2.7.3 连续传输

如果 ADC10CT 被设置，那么连续传输模式将会被选用。 当块一（一个块传输模式）或块二（两个块模式）完成传递后，DTC 不停止。 内部地址指针和数据传递计数器分别被设置等于 ADC10SA 和 n 时。 块一开始后，进行连续传递。 如果 ADC10CT 位被复位，在当前的数据完整的传输进块一（一个块模式）或块二（两个块模式）后，DTC 传递停止。

22.2.7.4 DTC 传输周期时间

对于每个 ADC10MEM 传递，DTC 需要一或二个 MCLK 周期来实现同步，一个周期 用于实际的传递（CPU 暂停时）和一个周期的等待时间。因为 DTC 使用 MCLK，DTC 的周期时间取决于 MSP430 的工作模式和时钟系统设置。

如果 MCLK 处在活动状态，但 CPU 停止时，DTC 使用 MCLK 时钟源进行每次的传递，无需重新使能 CPU 。 如果 MCLK 时钟源关闭，只有在一次传递过程中，DTC 才暂时重新起始来自 DCOCLK 的MCLK。 CPU 仍然关闭，在 DTC 传递后，MCLK 又重新关闭。 在所有工作模式下的最大 DTC 周期时间如表 22-2所示。

表 **22-2.** 最大 **DTC** 周期时间

CPU 运行模式	时钟源	最大 DTC 周期时间
激活模式	MCLK=DCOCLK	3 个 MCLK 周期
激活模式	MCLK=LFXT1CLK	3 个 MCLK 周期
低功耗模式 LPM0/1	MCLK=DCOCLK	4 个 MCLK 周期
低功耗模式 LPM3/4	MCLK=DCOCLK	4 个 MCLK 周期 + 2μs(1)
低功耗模式 LPM0/1	MCLK=LFXT1CLK	4 个 MCLK 周期
低功耗模式 LPM3	MCLK=LFXT1CLK	4 个 MCLK 周期
低功耗模式 LPM4	MCLK=LFXT1CLK	4 个 MCLK 周期 + 2μs(1)

(1) 额外的 2μs 被用于启动 DCOCLK。参数请参阅《器件专用数据表》。

22.2.8 使用集成温度传感器

想要使用片上温度传感器，应选择模拟输入通道 INCHx=1010。就象选择一个外部通道一样完成了任何其它配置，包括基准电压选择，转换寄存器选择等等。

典型的温度传感器传递功能图 22-13所示。 使用温度传感器时，采样周期必须大于 30μs。 温度传感器偏置误差比较大。 在应用中所产生的绝对温度值需要进行校准。 参数请参阅《器件专用数据表》。

选择温度传感器后，将自动打开片上基准电平发生器作为温度传感器的电压源。 然而，对于转换，它却不会使能V_{REF+}输出或影响基准电平的选择。 用于温度传感器的基准电平的选择与其它通道一样。

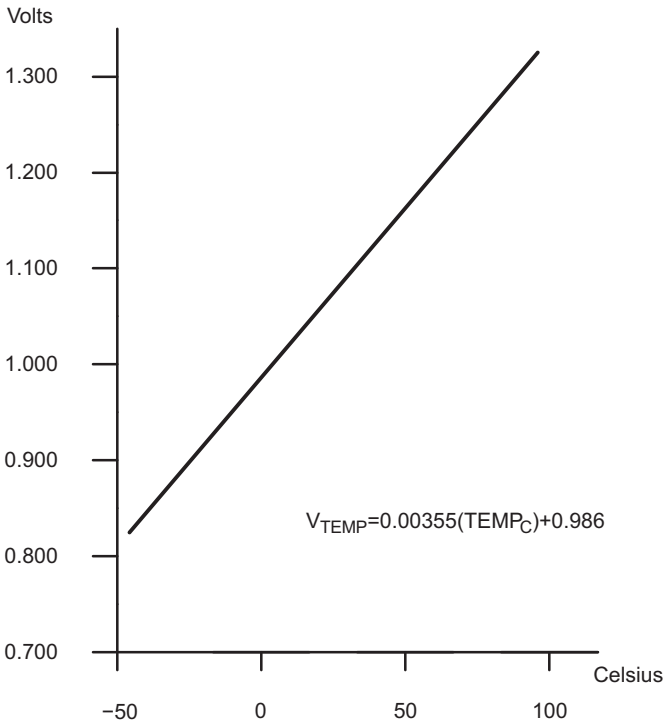


图 **22-13.** 典型的温度传感器传输功能

22.2.9 ADC10 接地和噪声考虑

就如高分辨率的 ADC，消除接地环路，有害的寄生效应，和噪音，需遵循印刷电路板布局和接地技术。

当回路电流从 A/D 流经其他的模拟或数字电路的公共回路时，会形成地接地环路。 如果不小心，这个电流会产生小的不必要的偏移电压，该电压可以增加或减少 A/D 转换器的基准电压或输入电压。 图 22-14和图 22-15所示的连接方法可以避免这些。

除了接地，由于切换数字或切换电源在电源线路上产生的纹波和噪声脉冲会影响转换的结果。 一个无噪声的设计对达到高精度的转换是非常重要的。

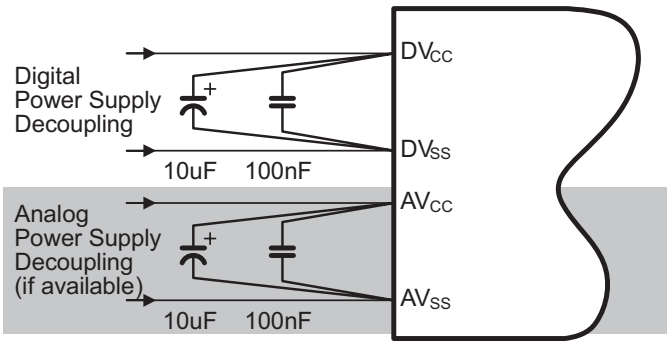


图 **22-14. ADC10** 接地和噪声考虑（内部**V_{REF}**）

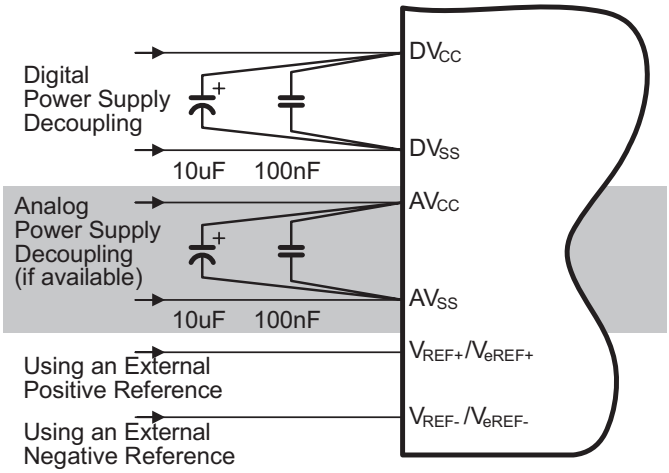


图 **22-15. ADC10** 接地和噪声考虑（外部**V_{REF}**）

22.2.10 ADC10 中断

与 ADC10 相关的一个中断和一个中断向量如图 22-16所示。 当不使用 DTC (ADC10DTC1=0)，转换结果装载到 ADC1OMEM 时，ADC10IFG 被设置。 当使用 DTC 时 (ADC10DTC1 > 0)，一个数据块传递完成和内部传递计数器'n' = 0 时，ADC10IFG 被设置。如果 ADC10IE 和GIE 位都被设置，然后ADC10IFG 标志产生一个中断请求。 当中断请求被响应后，ADC10IFG 标志自动复位，或者也可以通过软件复位。

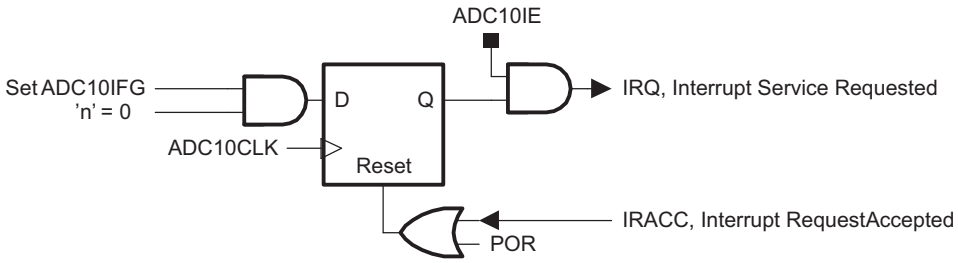


图 **22-16. ADC10** 中断系统

22.3 ADC10 寄存器

ADC10 寄存器在表 22-3中列出。

表 22-3. ADC10 寄存器

寄存器	简表	寄存器类型	地址	初态
ADC10 输入使能寄存器 0	ADC10AE0	读取/写入	04Ah	用 POR 复位
ADC10 输入使能寄存器 1	ADC10AE1	读取/写入	04Bh	用 POR 复位
ADC10 控制寄存器 0	ADC10CTL0	读取/写入	01B0h	用 POR 复位
ADC10 控制寄存器 1	ADC10CTL1	读取/写入	01B2h	用 POR 复位
ADC10 存储器	ADC10MEM	读取	01B4h	未改变
ADC10 数据传输控制寄存器 0	ADC10DTC0	读取/写入	048h	用 POR 复位
ADC10 数据传输控制寄存器 1	ADC10DTC1	读取/写入	049h	用 POR 复位
ADC10 数据传输起始地址	ADC10SA	读取/写入	01BCh	0200h 与 POR

22.3.1 ADC10CTL0，ADC10 控制寄存器0

15	14	13	12	11	10	9	8
SREFx			ADC10SHTx		ADC10SR	REFOUT	REFBURST
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
MSC	REF2_5V	REFON	ADC10ON	ADC10IE	ADC10IFG	ENC	ADC10SC
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

只有当ENC=0 时可以被修改

SREFx 位 15-13 选择基准

000 V_{R+}= V_{CC}和 V_{R-}= V_{SS}

001 V_{R+}= V_{REF+}和V_{R-}= V_{SS}

010 V_{R+}= V_{eREF+}和V_{R-}= V_{SS}。 只由 V_{eREF+}的器件。

011 V_{R+}= 缓冲的 V_{eREF+}和V_{R-}= V_{SS}。 只有 V_{eREF+}引脚的器件。

100 V_{R+}= V_{CC}和 V_{R-}= V_{REF-}/ V_{eREF-}。 只有 V_{eREF-}引脚的器件。

101 V_{R+}= V_{REF+}和V_{R-}= V_{REF-}/ V_{eREF-}。 只有 $V_{eREF+/-}$ 引脚的器件。

110 V_{R+}= V_{eREF+} 和V_{R-}= V_{REF-}/ V_{eREF-}。 只有 $V_{eREF+/-}$ 引脚的器件。

111 V_{R+}= 缓冲的 V_{eREF+} 和V_{R-}= V_{REF-}/ V_{eREF-}。 只有 $V_{eREF+/-}$ 引脚的器件。

ADC10SHTx 位 12-11 ADC10 采样保持时间

00 4 ×ADC10CLK

01 8 ×ADC10CLK

10 16 ×ADC10CLK

11 64 ×ADC10CLK

ADC10SR 位 10 ADC10 采样率。 该位为最大采样率选择基准缓冲器驱动性能。 设定 ADC10SR 来减少基准缓冲器的电流消耗。

0 基准缓冲器支持高达 ~200 ksps

1 基准缓冲器支持高达 ~50 ksps

REFOUT 位 9 基准输出

0 基准输出关闭

1 基准输出打开。 器件只有 V_{eREF+}/ V_{REF+} 引脚。

REFBURST 位 8 基准冲突。

0 连续的基准缓冲器

1 只用在采样和转换期间的基准缓冲器

MSC 位 7 多重采样和转换。 只在序列或重复模式中有效。

0 采样请求由 SHI信号的上升沿触发每次采样与转换。

1 第一个SHI 信号上升沿信号触发采样定时器，后面的采样与转换由前一次转换完成后立即被自动执行。

REF2_5V 位 6 基准电压产生器 REFON 也必须被置位。

0 1.5V

1 2.5V

REFON 位 5 基准电压生成器打开

0 基准电压生成器关闭

1 基准电压生成器打开

ADC10ON 位 4 ADC10 打开

0 ADC10关闭

1 ADC10打开

ADC10IE 位 3 ADC10 中断使能

0 中断禁用

1 中断使能

ADC10IFG	位 2	ADC10 中断标志。 如果 ADC10MEM 已经装满转换结果，该位被设置。当中断请求被接受时，它将自动复位，或者通过软件复位。当使用 DTC，一个数据块传输完成时，该标志被置位。 0 无中断等待 1 中断等待
ENC	位 1	使能转换 0 ADC10被禁用 1 ADC10被启用
ADC10SC	位 0	开始转换 软件控制的采样和转换启动。 ADC10SC 和 ENC用同一个指令一起被置位。 ADC10SC 被自动复位。 0 无采样转换启动 1 启动采样和转换

22.3.2 ADC10CTL1，ADC10 控制寄存器1

15	14	13	12	11	10	9	8
INCHx				**SHSx**		**ADC10DF**	**ISSH**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ADC10DIVx			**ADC10SSELx**		**CONSEQx**		**ADC10BUSY**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r-0

只有当ENC=0 时，才能被修改。

INCHx 位 15-12 输入通道选择。 这些位用来选择进行单次转换的通道或进行序列转换的最高通道。 只有可用的 ADC 通道才会被选择。 请参阅《器件专用数据表》。

0000 A0
0001 A1
0010 A2
0011 A3
0100 A4
0101 A5
0110 A6
0111 A7
1000 V_{eREF+}
1001 V_{REF-}/V_{eREF-}
1010 温度传感器
1011 $(V_{CC}- V_{SS})/ 2$
1100 $(V_{CC}- V_{SS})/ 2$, A12 在 MSP430F22xx器件上
1101 $(V_{CC}- V_{SS})/ 2$, A13 在 MSP430F22xx器件上
1110 $(V_{CC}- V_{SS})/ 2$, A14 在 MSP430F22xx器件上
1111 $(V_{CC}- V_{SS})/ 2$, A15 在 MSP430F22xx器件上

SHSx 位 11-10 采样和保持源选择。

00 ADC10SC位
01 定时器_A。OUT1(1)
10 定时器_A。OUT0(1)
11 定时器_A.OUT2（在 MSP430F20x0，MSP430G2x31，和 MSP430G2x30 器件上的定时器_A.OUT1）(1)

ADC10DF 位 9 ADC10 数据格式

0 直接二进制
1 2补码

ISSH 位 8 反相信号采样保持

0 采样输入信号未被反相。
1 采样输入信号被反相。

ADC10DIVx 位 7-5 ADC10 时钟分频器

000 /1
001 /2
010 /3
011 /4
100 /5
101 /6
110 /7
111 /8

ADC10SSELx 位 4-3 ADC10 时钟源选择

00 ADC10OSC
01 ACLK
10 MCLK
11 SMCLK

(1) 如果器件上存在一个以上的定时器模块，那么就由定时器 0_Ax 触发定时器。

CONSEQx	位 2-1	转换序列模式选择
		00 单通道单次转换
		01 通道序列
		10 重复单通道
		11 重复通道序列
ADC10BUSY	位 0	ADC10 忙。 该位标志着一个有效的采样和转换操作
		0 无操作活动。
		1 一个序列，采样，或转换活动。

22.3.3 ADC10AE0，模拟（输入）使能控制寄存器 0

7	6	5	4	3	2	1	0
ADC10AE0x							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

ADC10AE0x	位 7-0	ADC10 模拟使能。 该位为模拟输入使能相应的引脚。 BIT0对应于 A0，BIT1 对应于 A1，等等。 未执行通道的模拟使能位不应该被编程为 1。
		0 模拟输入被禁用
		1 模拟输入被启用

22.3.4 ADC10AE1，模拟（输入）使能控制寄存器 1（仅适用于 MSP430F22xx）

7	6	5	4	3	2	1	0
ADC10AE1x				被保留			
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

ADC10AE1x	位 7-4	ADC10 模拟使能。 该位为模拟输入使能相应的引脚。 BIT4对应于 A12，BIT5 对应于 A13，BIT6 对应于 A14，和 BIT7 对应于 A15。未使用通道的模拟使能位不应该被编程为 1。
		0 模拟输入禁用
		1 模拟输入使能
被保留	位3-0	被保留

22.3.5 ADC10MEM，转换存储寄存器，二进制格式

15	14	13	12	11	10	9	8
0	0	0	0	0	0	转换结果	
r0	r0	r0	r0	r0	r0	r	r

7	6	5	4	3	2	1	0
转换结果							
r	r	r	r	r	r	r	r

转换结果	位 15-0	10 位转换结果是右对齐，直接二进制格式。 位 9 是 MSB。 位 15-10 总是0。

22.3.6 ADC10MEM，转换存储寄存器，2 补码格式

15	14	13	12	11	10	9	8
转换结果							
r	r	r	r	r	r	r	r
7	6	5	4	3	2	1	0
转换结果		0	0	0	0	0	0
r	r	r0	r0	r0	r0	r0	r0

转换结果 位 15-0 10 位转换结果是左对齐，2 补码格式。 位 15 是 MSB。 位 5-0 总是 0。

22.3.7 ADC10DTC0，数据传输控制寄存器 0

7	6	5	4	3	2	1	0
被保留				**ADC10TB**	**ADC10CT**	**ADC10B1**	**ADC10FETCH**
r0	r0	r0	r0	rw-(0)	rw-(0)	r-(0)	rw-(0)

被保留 位 7-4 被保留。 总是读取为 0。

ADC10TB 位 3 ADC10 两个数据块模式

0 一个数据块传输模式

1 两个数据块传输模式

ADC10CT 位 2 ADC10 连续传输

0 当一个块（一块模式）或两个块（两块模式）传输完成时，数据传输停止。

1 数据是连续传输的。 DTC 的运行只在 ADC10CT 被清零，或 ADC10SA 被写入时才会停止。

ADC10B1 位 1 ADC 块一。 该位表明两个块模式下哪个块被ADC10 转换结果装入。 ADC10B1 只有当 ADC10IFG 位在DTC 工作期间第一次被置位后才有效。 ADC10TB 也必须被置位。

0 块 2 被填满

1 块 1 被填满

ADC10FETCH 位 0 该位应该被正常的复位。

22.3.8 ADC10DTC1，数据传输控制寄存器 1

7	6	5	4	3	2	1	0
DTC传输							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

DTC 传输 位 7-0 DTC传输 该位定义了每个块中的传输量。

0 DTC 被禁用

01h-0FFh 每个块中的传输量

22.3.9 ADC10SA，数据传输的开始地址寄存器

15	14	13	12	11	10	9	8
ADC10SAx							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ADC10SAx							0
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r0

ADC10SAx	位 15-1	ADC10 开始地址。 这些位是 DTC 的开始地址。 要启动 DTC传输就需要一个对寄存器 ADC10SA 的写入。
未使用	位 0	未使用，只读。 总是读取为 0。

ADC12

ADC12 模块是一个高性能的 12 位模数转换器。 本章介绍了 MSP430x2xx 器件系列中的 ADC12。

Topic **Page**

23.1 ADC12 介绍

ADC12 模块支持快速 12 位模数转换。 此模块运行一个 12 位逐次逼近 (SAR) 内核、样本选择控制、基准生成器、和一个 16 字转换和控制缓冲器。 在无需 CPU 干预的情况下，转换和控制缓冲器可转换并存储多达 16 个独立的 ADC 样本。

ADC12 模块特征如下：

- 大于 200ksps 的最大转换速率
- 单片的 12 位转换器无失码
- 由软件或定时器控制的可编程采样周期的采样保持
- 由软件，定时器_A，或定时器_B 启动的转换
- 软件可选片上基准电压生成（1.5V 或 2.5 V）
- 软件可选内部或外部基准
- 8 个可单独配置的外部输入通道
- 内部温度传感器的转换通道，AV_{CC}，和外部基准电压
- 独立的通道可选基准位自正和负基准供源
- 可选的转换时钟源
- 单信道、重复单信道、序列、和重复序列的转换模式
- ADC 内核和基准电压都可以独立断电
- ADC 中断的快速解码的 18 位中断矢量寄存器
- 16 转换结果存储寄存器

ADC12 模块的方框图如图 23-1所示。

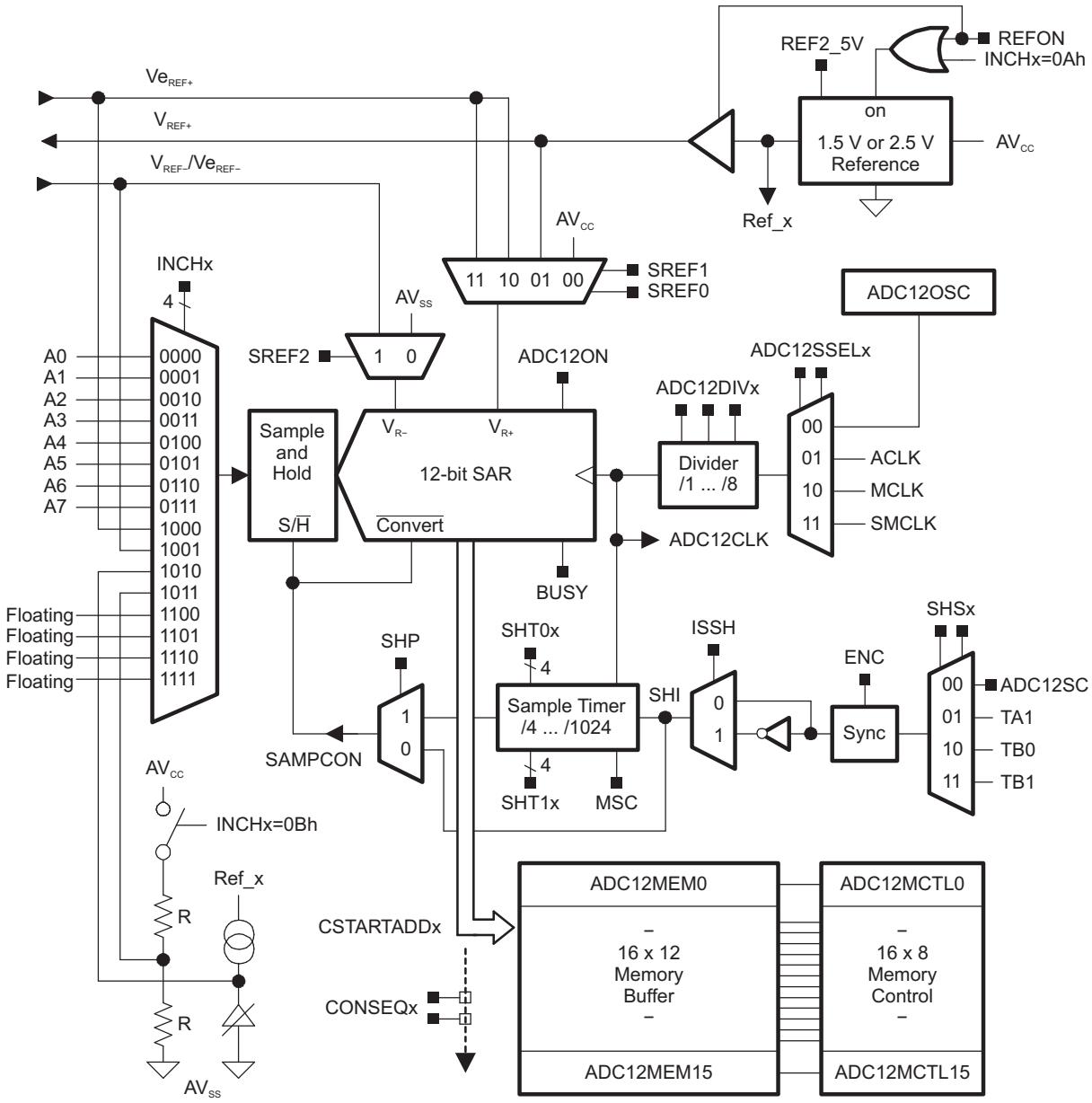


图 **23-1. ADC12** 方框图

23.2 ADC12 的操作

ADC12 模块可由用户软件配置。 ADC12 的运行和建立在下列章节中进行讨论。

23.2.1 12 位 ADC 内核

ADC 内核将一个模拟量的输入转化成 12 位数字形式，结果保存在转换存储器中。 内核利用两个可编程/可选的基准电压（V_{R+}和 V_{R-}）来定义转换范围的最大值和最小值。 当输入信号等于或高于 V_{R+} 时，数字输出 (N_{ADC}) 为满量程 (0FFFh)，当输入信号等于或低于 V_{R-}时，输出为零。 输入通道和基准电平（V_{R+}和 V_{R-}）在转换控制寄存器中进行了定义。 ADC 结果的换算公式 N_{ADC}是：

$$N_{ADC} = 4095 \times \frac{V_{IN} - V_{R-}}{V_{R+} - V_{R-}}$$

ADC12 内核由 ADC12CTL0 和 ADC12CTL1 两个控制寄存器完成配置。 内核使能由 ADC12ON 位控制。为了节省电力，可以在不使用时关闭 ADC12。 大多数情况下，只有在 ENC=0 时 ADC12 的控制位才可以被修改。在进行任何转换前 ENC 位必须设为 1。

23.2.1.1 转换时钟选择

该 ADC12CLK 被用作转换时钟并在采样模式被选择时产生脉冲采样周期。 ADC12 源时钟可以用 ADC12SSELX 位来选择，且可以用 ADC12DIVx 位进行 1 至 8 分频。 可选的 ADC12CLK 源有 SMCLK，MCLK，ACLK 和一个内的振荡器 ADC12OSC。

ADC12OSC 由内部产生，且位于 5MHz 范围内，但频率会随个别器件，电源电压，和温度而不同。 有关 ADC12OSC 的规格请参阅《器件专用数据表》。

应用程序必须保证在一个转换结束前为 ADC12CLK 所选的时钟都保持在活动状态。 如果在一个转换期间时钟被移除，那么转换将无法完成且结果无效。

23.2.2 ADC12 输入和多路复用器

模拟输入多路复用器可以选择 8 个外部和 4 个内部模拟信号接口作为转换通道。 输入模拟多路复用器是先关后开型开关以此来减少因通道切换而引入的噪声（见图 23-2）。 输入模拟多路复用器也是一个 T 型开关，可以减少通道间的耦合。 未选择的通道与 A/D 分开，并且中间接点被连接到模拟接地 (AV_{SS}) 端以便于寄生电容接地从而减少噪声。

ADC12 利用电荷再分配原理。 当输入在内部切换时，切换动作可能在输入信号上引起瞬变。 这些瞬变衰减和解决之前会导致错误的转换。

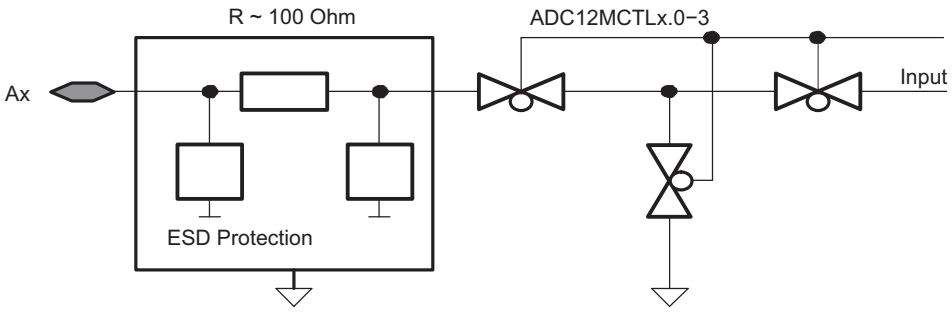


图 **23-2.** 模拟多路复用器

23.2.2.1 模拟端口选择

ADC12 输入是与端口 P6 引脚复用的，其中该引脚是数字 CMOS 门。 当模拟信号被应用到数字 CMOS 门时，寄生电流可以从 VCC 流向 GND。 如果输入电压接近该门的转换电平时，就会产生寄生电流。 禁用端口引脚缓冲器防止了寄生电流流过，因此，可降低总电流消耗。 P6SELx 位提供了禁用端口引脚的输入和输出缓冲器的功能。

```
; P6.0 and P6.1 configured for analog inputBIS.B #3h,&P6SEL ; P6.1 and P6.0 ADC12 function
```

23.2.3 *基准电压产生器*

ADC2.5 模块包含一个内置的电压基准带有两个可选的电压电平，1.5V 和 2.5V。 这些基准电压的任一个都可用于在引脚 V_{REF+}的内部和外部。

设置 REFON=1 使能内部基准。 当 REF2_5V=1 时，内部基准电压为 2.5V。当 REF2_5V=0 时，基准电压为1.5V。为了节省电能，可在不使用时关闭基准电压。

为了正确操作，必须给内部参考电压发生器提供整个 V_{REF+}和 AV_{SS}中的存储容量。 建议的存储电容是 10μF 和 0.1μF 电容$_{的一个并联组合。}$ 从开启开始上，必须允许一个最大为 17ms 的参考电压发生器来偏置建议的贮藏电容。 如果内部基准发生器还没有用于该转换，则不需要存储电容器。

注: 基准电压解耦

在一个转换期间，当正在解决两个 LSB 时*ADC12 使用的任何*基准电压中都需要大约 200μA。建议为*任何*基准电压连接一个 10μF 和 0.1μF 电容的并联组合，如在图 23-11中所示。

可分别把外部参考应用于 V_{R+}和 V_{R-}至引脚 V_{EREF+}和 V_{REF-}/ V_{EREF-}。

23.2.4 *采样和转换时序*

一个数模转换是由一个采样输入信号 SHI 的上升沿启动。 SHI 信号源可以通过 SHSx 位来选择， 包括如下：

- ADC12SC 位
- 定时器_A 输出单元 1
- 定时器_B 输出单元 0
- 定时器_B 输出单元 1

SHI 信号源的极性可以通过 ISSH 位来反转。 SAMPCON 信号控制转换的采样周期的开始。 当 SAMPCON 为高电平时，采样是有效的。 高至低 SAMPCON 过渡起始了模数转换，这需要 13 个 ADC12CLK 周期。两种不同的采样定时方法是控制位 SHP，扩展的采样模式和脉冲模式定义。

23.2.4.1 扩展的采样模式

当 SHP=0 时选择扩展的采样模式。SHI 信号直接控制 SAMPCON 并定义采样周期 $t_{采样}$的长度。 当 SAMPCON 为高电平时，采样是有效的。 在与 ADC12CLK 同步后，高至低 SAMPCON 过渡起始了该转换（请参见图 23-3）。

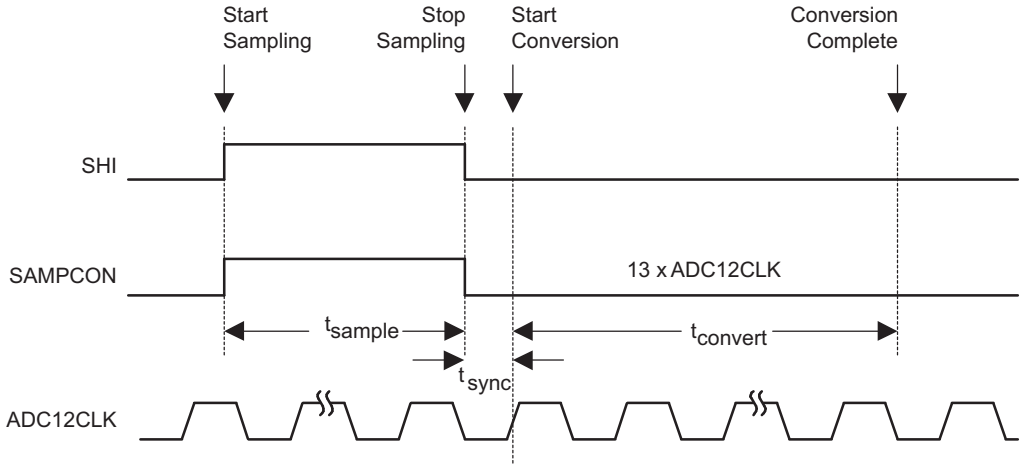


图 **23-3.** 扩展的采样模式

23.2.4.2 脉冲采样模式

当 SHP =1 时选择脉冲采样模式。SHI 信号用于触发采样定时器。 ADC12CTL0 中的 SHT0x 和 SHT1x 位控制采样定时器的时间间隔的，该定时器定义 SAMPCON 的采样周期$t_{采样}$。 在一个可编程时间间隔 $T_{采样}$与 AD12CLK 同步后，采样定时器保持 SAMPCON 为高电平。 总的采样时间为 $t_{采样}$加上 $t_{同步}$（请见图 23-4）。

SHTx 位选择 ADC12CLK 的 4 倍采样时间。 SHT0x 为 ADC12MCTL0 至 7 选择采样时间且 SHT1X 为 ADC12MCTL8 至 15 选择采样时间。

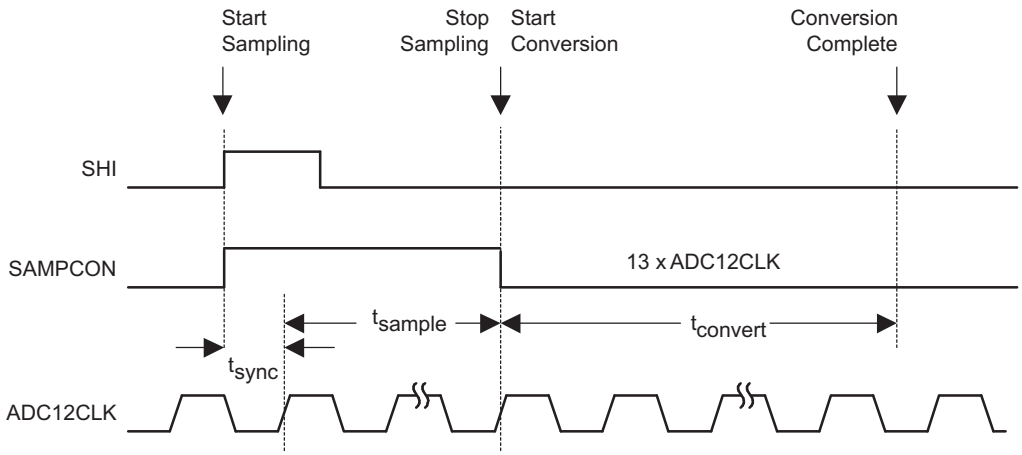


图 **23-4.** 脉冲采样模式

23.2.4.3 采样时序转换

当 SAMPCON =0 时，所有的 Ax 输入均为高阻态 。 当 SAMPCON =1 时，在采样时间$_t$采样图 23-5期间，已选的 Ax 输入相当于一个 RC 低通滤波器，如所示。 一个内部 MUX 上输入电阻 R_I（最大 2kΩ）被认为是由源极和电容器 C_I（最大 40pF）串连在一起。 电容器 C_I电压 (V_C必须被充电至电源电压 (V_S) 的 ½ LSB 范围内，以此来进行精度为 12 位的转换。

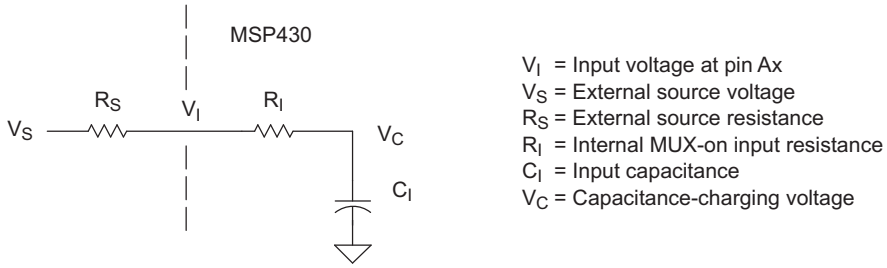


图 23-5. 模拟输入等效电路

源 R_S和 $R_{I电阻}$影响$_{采样}$。 可以用下面的公式来计算 12 位转换的最小采样时间 $t_{采样}$：

$$t_{采样} > (R_S + R_I) \times \ln(2^{13}) \times C_I + 800\ ns$$

R_I和 C_I的代入值在上面已经给出，公式变为：

$$t_{采样} > (R_S + 2\ k\Omega) \times 9.011 \times 40\ pF + 800\ ns$$

例如，如果 R_S是 10kΩ，那么 $t_{采样}$必须大于 5.13μs。

23.2.5 转换存储器

有 16 个 ADC12MEMx 转换内存寄存器用于存储转换结果。 每个 ADC12MEMx 都用一个相关的 ADC12MCTLx 控制寄存器配置。 SREFx 位定义电压基准，INCHx 位选择输入通道。 当使用顺序转换模式时，EOS 位定义序列末端。 当 ADC12MCTL15 中的 EOS 没被置位时，一个序列从 ADC12MEM15 翻转到 ADC12MEM0。

CSTARTADDx 位定义用于任何转换的第一个 ADC12MCTLx。 如果转换模式是单通道或重复单通道，那么 CSTARTADDx 会指向要使用的单 ADC12MCTLx。

如果所选择的转换模式是序列通道或重复序列通道中一个，CSTARTADDx 会指向在一个序列中第一个被使用到的 ADC12MCTLx 的位置。 一个指针，对软件不可见，当每次转换完成时，会自动递增至序列中的下一个 ADC12MCTLx。 序列继续进行直到 ADC12MCTLx 中的 EOS 位被处理；这是最后一个被处理的控制字节。

当转换结果被写入一个选定的 ADC12MEMx 中时，在 ADC12IFGx 寄存器中的相应标志被置位。

23.2.6 ADC12转换模式

ADC12 有四个可由 CONSEQx 位选择的运行模式，在表 23-1中给出了这些模式。

表 23-1. 转换模式概述

CONSEQx	模式	运行
00	单通道单次转换	一个单通道被转换一次
01	通道序列	一个通道序列被转换一次
10	单通道重复	一个单通道被重复转换
11	序列通道重复	一个通道序列被重复转换

23.2.6.1 单通道单次转换模式

一个单通道被采样和转换一次 ADC 结果被写入由 CSTARTADDx 位定义的 ADC12MEMx 中。图 23-6显示了单通道单次转换模式的流程。 当 ADC12SC 触发一次转换时，连续的转换可有 ADC12SC 位来触发。 当使用任何其它触发源时，ENC 必须在每次转换间被切换。

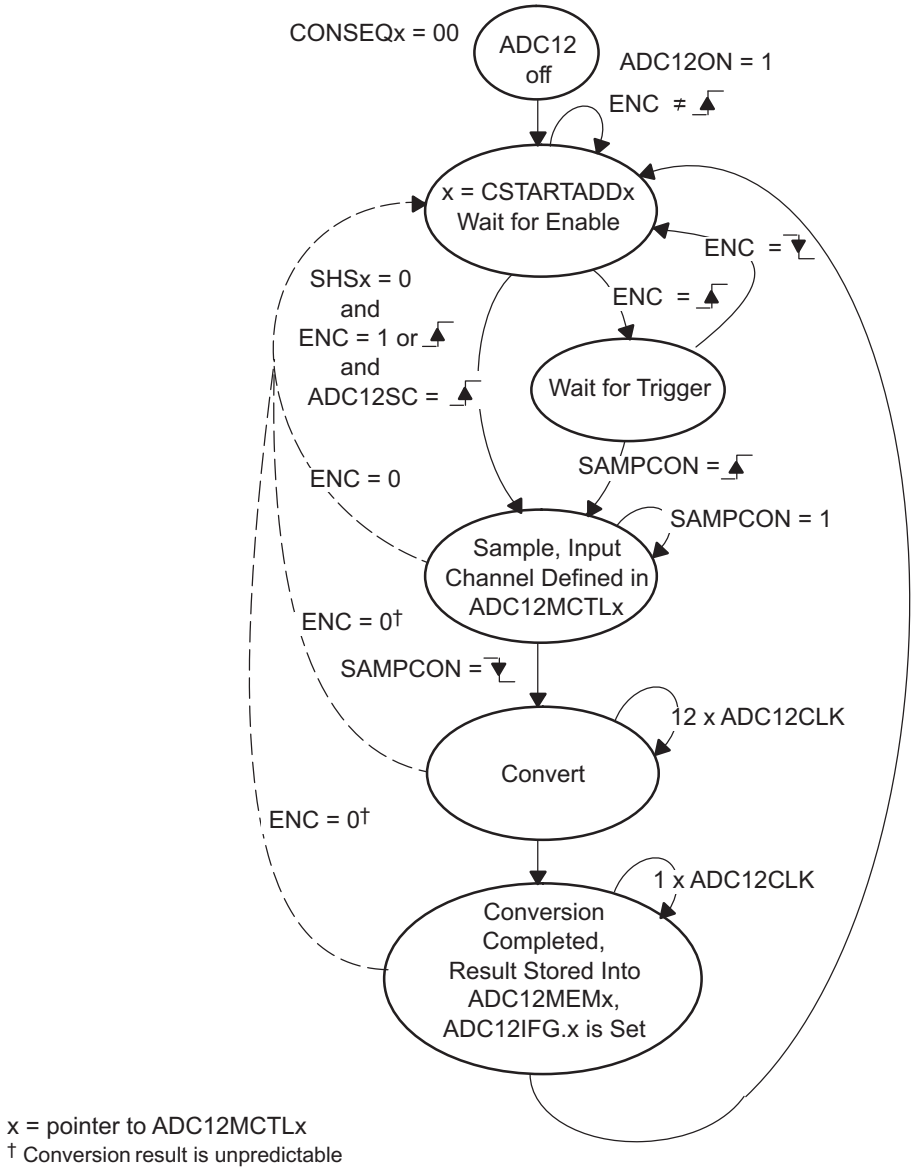


图 **23-6.** 单通道，单次转换模式

23.2.6.2 通道序列模式

一个通道序列被采样和转换一次 ADC 的结果被写入用 ADCMEMx 起始的转换存储器中，其中 ADCMEMx 由CSTARTADDx 位定义。 在用一组 EOS 位测量完通道后，序列停止。图 23-7显示了通道序列模式。 当 ADC12SC 触发一个序列时，连续的序列可由 ADC12SC 位来触发。 当使用任何其它触发源时，ENC 必须在每个序列间被切换。

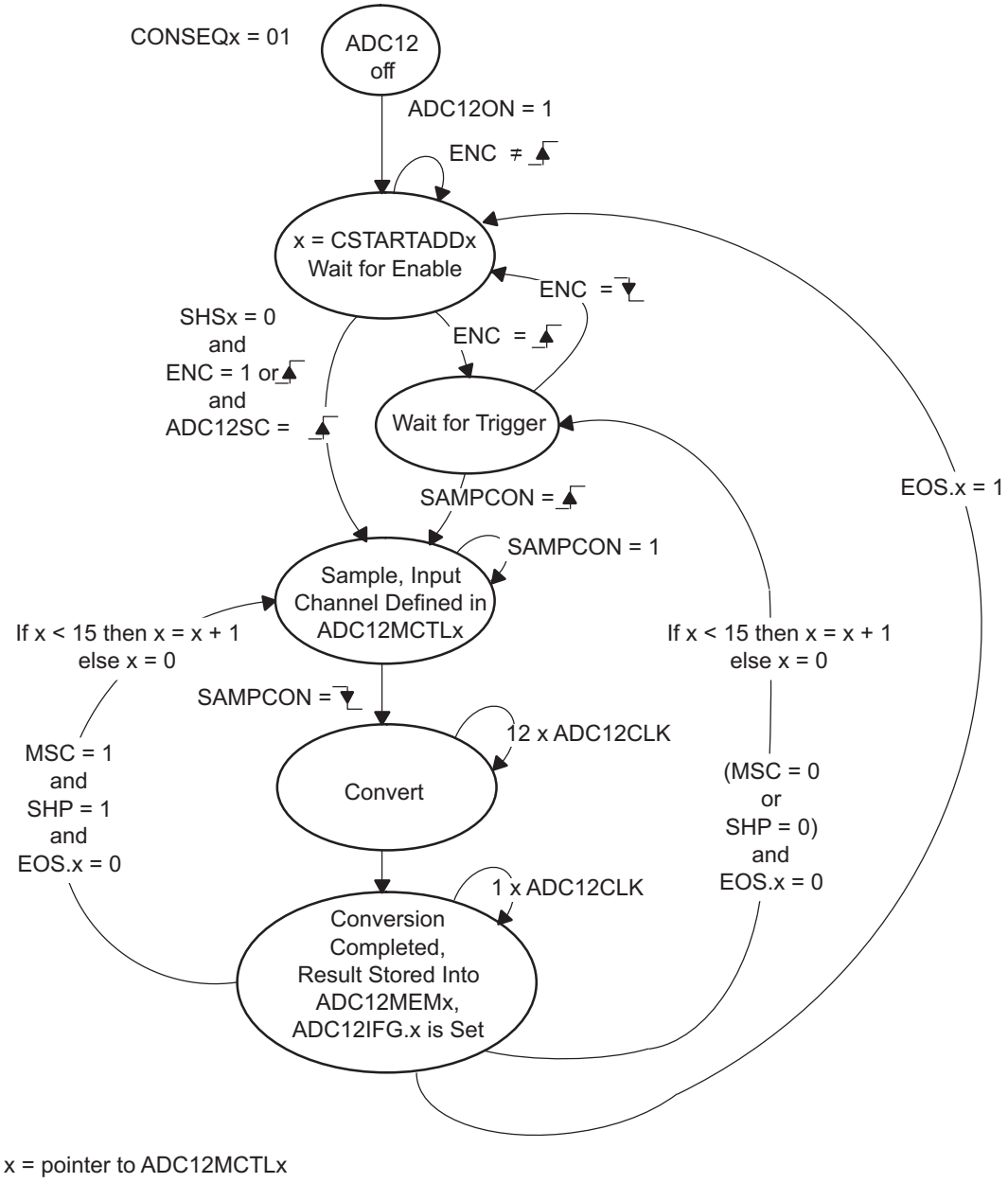


图 **23-7.** 通道序列模式

23.2.6.3 单通道重复模式

一个单通道被连续采样和转换。 ADC 结果被写入由 CSTARTADDx 位定义的 ADC12MEMx 中。 由于只有一个 ADC12MEMx 存储器被采用而且是被下一个转换覆写，有必要在完成一个转换后读取结果。图 23-8显示了单通道重复模式。

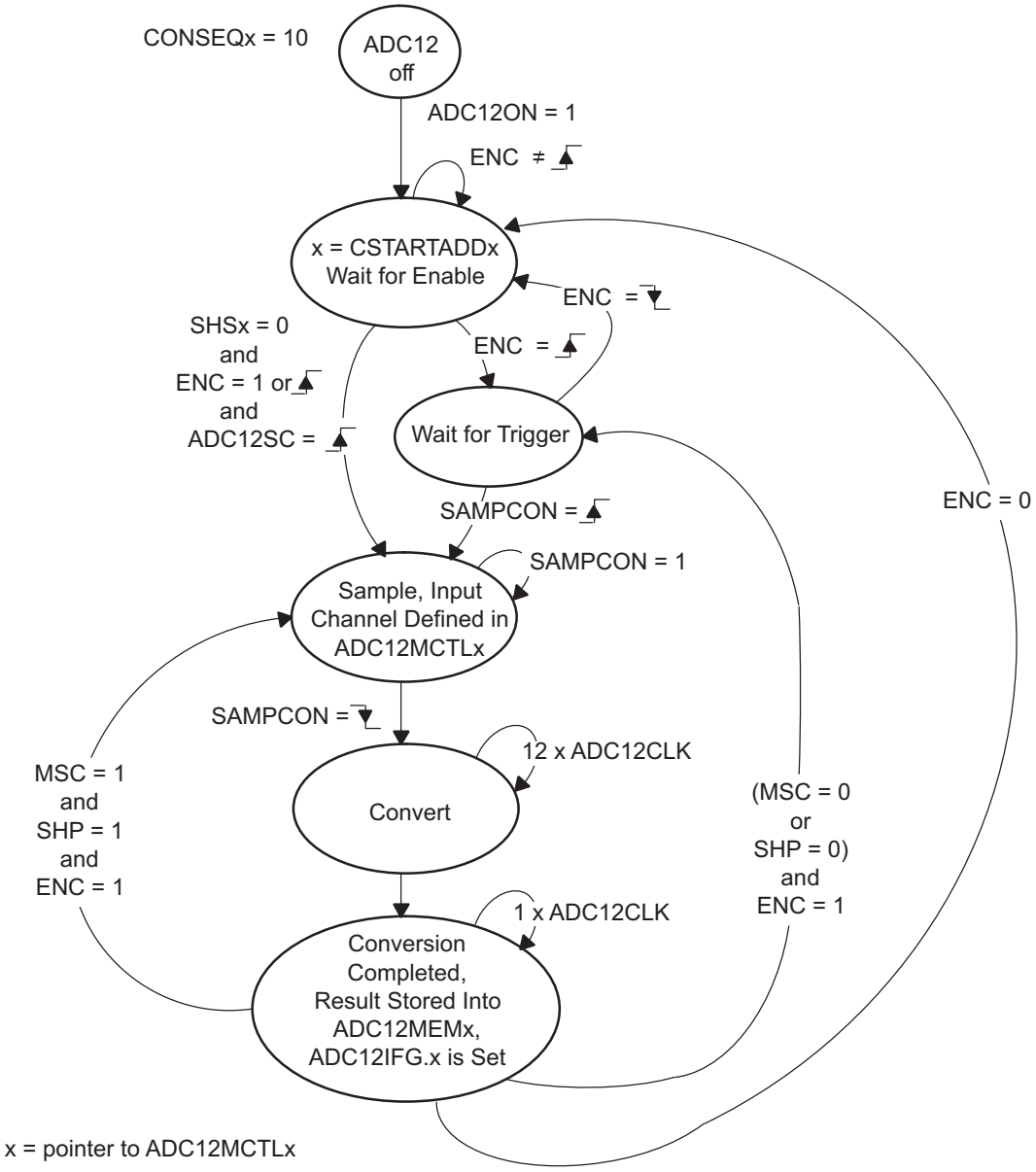


图 **23-8.** 单通道重复模式

23.2.6.4 通道的重复序列模式

一个通道序列被重复采样和转换。 ADC 的结果被写入用 ADCMEMx 起始的转换存储器中，其中 ADCMEMx 由CSTARTADDx 位定义。 在用一组 EOS 位测量完通道后该序列结束，下一个触发信号重新启动序列。图 23-9显示了通道的重复序列模式。

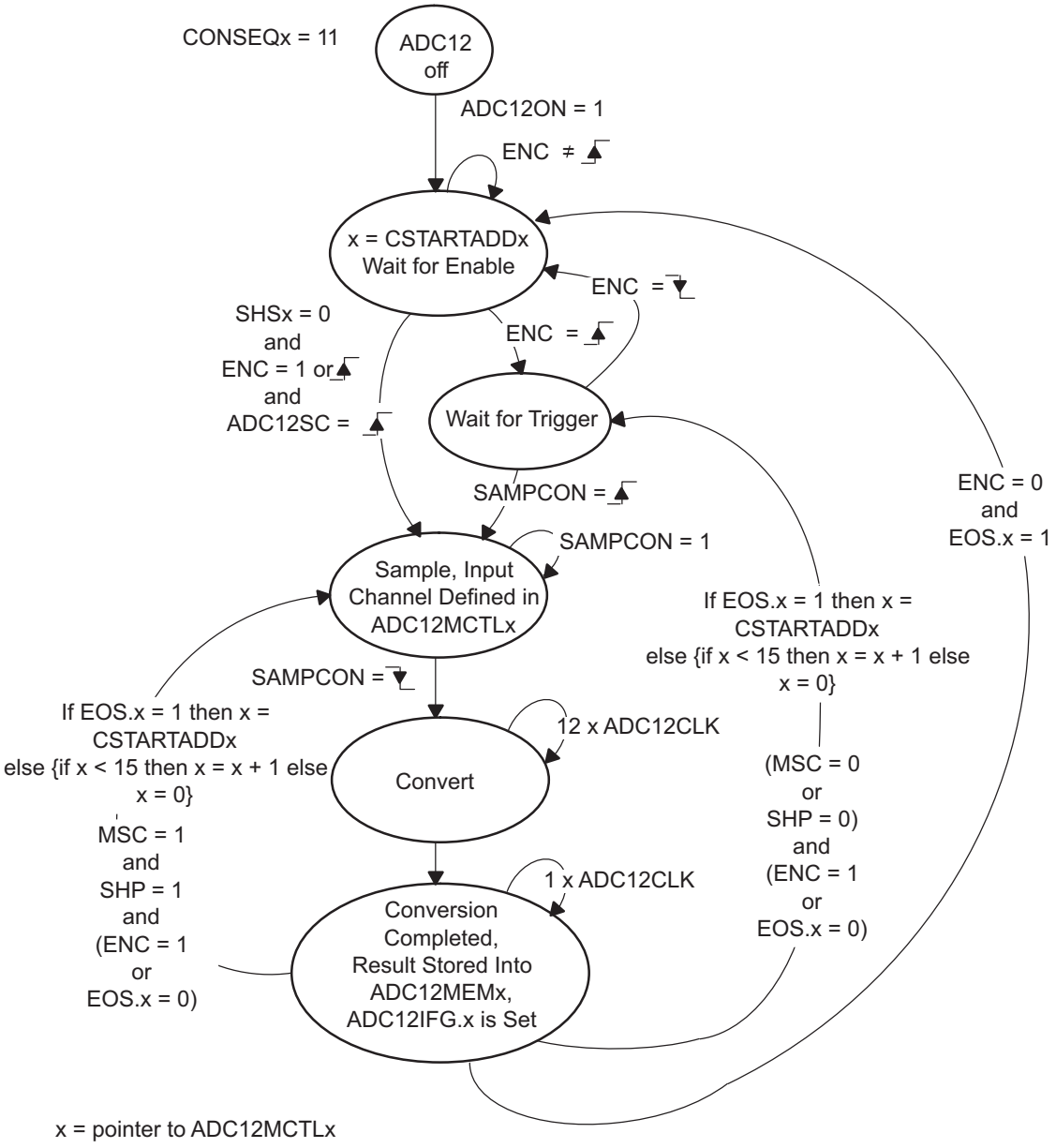


图 **23-9.** 通道的重复序列模式

23.2.6.5 使用多路采样和转换 (MSC) 位

为了配置转换器来使其能自动的执行连续转换并且尽可能的快速，可以使用一个多路采样和转换功能。 当 MSC=1，CONSEQx>0，且使用采样定时器时，SHI 信号的第一个上升沿会触发第一个转换。 转换完成前，连续转换被快速并自动的触发。 在单序列模式中，序列完成前或在单通道重复模式或重复序列模式中，ENC 位被切换前，SHI 上的其他上升沿都被忽略。 当使用 MSC 位时，ENC 位的功能不能更改。

23.2.6.6 停止转换

停止 ADC12 活动取决于运行模式。 建议的停止一个活动转换或转换序列的方法是：

- 在单通道单转换模式中复位 ENC 可以立即停止一个转换并且结果是不可预知的。 想要获得正确的结果，在清除 ENC 前应轮询忙位直到它被复位。
- 在单通道重复运行期间复位 ENC 可以在当前转换结束时停止转换。
- 在一个序列或重复序列模式器件复位 ENC 可以在序列结束时停止转换。
- 通过设置 CONSEQx=0 和复位 ENC 位可以立即停止任何转换模式。 在这种情况下，转换的数据是不可信的。

注: 针对序列的无 EOS 位位置

如果没有 EOS 位被置位但已选序列模式，复位 ENC 位不会停止该序列。 要停止该序列，首先要选择单通道模式并然后复位 ENC。

23.2.7 使用集成温度传感器

想要使用片上温度传感器，应选择模拟输入通道 INCHx=1010。 设置其它寄存器就象选择一个外部通道一样，包括基准电平选择，转换寄存器选择等等。

典型的温度传感器传递功能如图 23-10所示。 当使用温度传感器时，采样周期必须大于 30μs。 温度传感器的偏移误差可能会比较大，对于大多数应用程序来说都需要对其进行校准。 有关参数请参阅《器件专用数据表》。

选择温度传感器会自动打开片上作为温度传感器的一个电压源的基准电平发生器。 然而，对于转换，它不会使能 V_{REF+}输出或影响基准电平的选择。 用于转换温度传感器的基准电压的选择与其它通道一样。

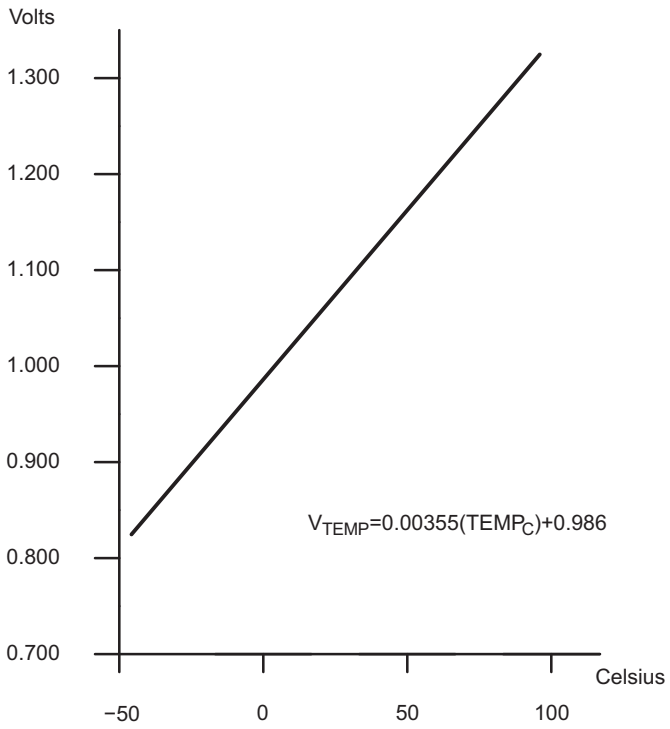


图 **23-10.** 典型的温度传感器传输功能

23.2.8 ADC12 接地和噪声考虑

就像高分辨率的 ADC，为了消除地电流环路，有害的寄生效应和噪声，应遵循合理的印刷电路板布局和接地技术。

当 A/D 中的回路电流流经其他模拟或数字电路的公共回路时，就会形成地电流环路。 如果不当心，这个电流会产生小的不必要的偏移电压，该电压可以增加或减少 A/D 转换器的基准电压或输入电压。 图 23-11中所示的连接方法可以避免这些。

除了接地，由于切换数字或切换电源会在电源线路上产生纹波和噪声脉冲，这也会影响转换的结果。 为了实现高精确度，推荐一个使用独立的模拟和带有一个单点连接的数字地平面的无噪音设计。

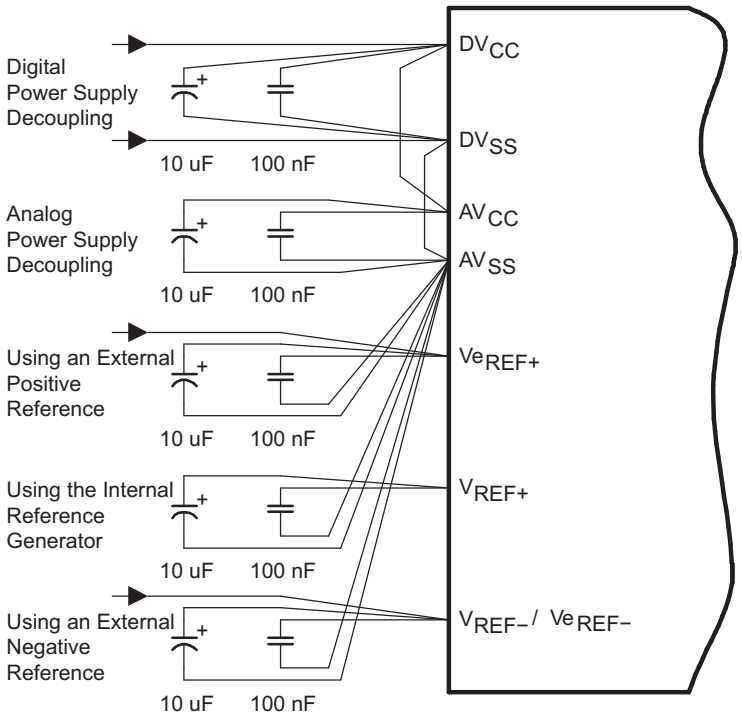


图 **23-11. ADC12** 接地和噪声考虑

23.2.9 ADC12 中断

ADC12 有 18 个中断源：

- ADC12IFG0 至 ADC12IFG15
- ADC12OV，ADC12MEMx 溢出
- ADC12TOV，ADC12 转换时间溢出

其相应的 ADC12MEMx 内存寄存器加载一个转换结果时 ADC12IFGx 位被置位。 如果相应的 ADC12IEx 位和 GIE 位被置位，会产生一个中断请求。 当转换结果 ADC12MEMx 的先前转换结果被读取之前被写入任何 ADC12MEMx 中时，ADC12OV 条件发生。 在当前转换完成之前请求另一个采样转换时，会产生 ADC12TOV 条件。 在转换后，该 DMA 在单信道模式下或在一个序列通道模式完成之后被触发。

23.2.9.1 ADC12IV，中断向量发生器

所有 ADC12 中断源被优先化并被结合起来共同来自一个中断向量源。 中断向量寄存器 ADC12IV 用于确定哪一个使能的 ADC12 中断源请求了一个中断。

最高优先级的使能的 ADC12 中断产生在 ADC12IV 寄存器中产生了一个数字（请见23.3.7 节）。 为了能自动进入相应的软件程序，可以对这个数字进行评估，或将其添加到程序计数器。 禁用的 ADC12 中断不影响 ADC12IV 的值。

如果是最高的正挂起的中断，对 ADC12IV 寄存器进行的任何访问（读取或写入）都会自动复位 ADC12OV 条件 或 ADC12TOV 条件。 中断条件也没有一个可访问的中断标志。 该 ADC12IFGx 标志不会被一个 ADC12IV 的访问复位。 通过访问相关的 ADC12MEMx 寄存器 能使 ADC12IFGx 位自动复位或可能通过软件复位。

如果在服务一个中断后另一个中断挂起，就会产生另一个中断。 例如，如果 ADC12OV 和 ADC12IFG3 中断挂起时，当中断服务子程序访问 ADC12IV 寄存器时，ADC12OV 中断条件会自动复位。 在中断服务程序的 RETI 指令被执行后，ADC12IFG3 会产生另一个中断。

23.2.9.2 ADC12 中断处理软件示例

Example 23-1给出了推荐的 ADC12IV 和处理开销的使用。 为了自动跳转到相应的子程序，ADC12IV 值将被添加到 PC。

在右边距的数字显示了每条指令所需的 CPU 周期。 不同中断源的软件开销包括中断延迟时间和从中断返回周期，但不处理任务本身。 这些延迟是：

- ADC12IFG0 至 ADC12IFG14，ADC12TOV，和 ADC12OV：16 个周期
- ADC12IFG15：14 个周期

如果在 ADC12IFG15 的处理过程中发生了一个更高优先级的中断，ADC12IFG15 的中断处理程序会给出一个立即检查的方法。 如果另一个 ADC12 中断挂起时，这样可以节省 9 个时钟周期。

Example 23-1. 中断处理

```
; Interrupt handler for ADC12.INT_ADC12 ; Enter Interrupt Service Routine 6ADD &ADC12IV,PC ; Add
offset to PC 3RETI ; Vector 0: No interrupt 5JMP ADOV ; Vector 2: ADC overflow 2JMP ADTOV ; Vector 4:
ADC timing overflow 2JMP ADM0 ; Vector 6: ADC12IFG0 2... ; Vectors 8-
32 2JMP ADM14 ; Vector 34: ADC12IFG14 2;; Handler for ADC12IFG15 starts here. No JMP required.;ADM15
MOV &ADC12MEM15,xxx ; Move result, flag is reset... ; Other instruction needed?JMP INT_ADC12 ; Check
other int pending;; ADC12IFG14-
ADC12IFG1 handlers go here;ADM0 MOV &ADC12MEM0,xxx ; Move result, flag is reset... ; Other
instruction needed?RETI ; Return 5;ADTOV ... ; Handle Conv. time overflowRETI ; Return 5;ADOV ... ;
Handle ADCMEMx overflowRETI ; Return 5
```

23.3 ADC12 寄存器

在表 23-2中列出了 ADC12 寄存器。

表 23-2. ADC12 寄存器

寄存器	简式	寄存器类型	地址	初始化状态
ADC12 控制寄存器 0	ADC12CTL0	读取/写入	01A0h	用 POR 复位
ADC12 控制寄存器 1	ADC12CTL1	读取/写入	01A2h	用 POR 复位
ADC12 中断标志寄存器	ADC12IFG	读取/写入	01A4h	用 POR 复位
ADC12 中断使能寄存器	ADC12IE	读取/写入	01A6h	用 POR 复位
ADC12 中断向量字	ADC12IV	读取	01A8h	用 POR 复位
ADC0 存储器 0	ADC12MEM0	读取/写入	0140h	未改变
ADC1 存储器 1	ADC12MEM1	读取/写入	0142h	未改变
ADC2 存储器 2	ADC12MEM2	读取/写入	0144h	未改变
ADC3 存储器 3	ADC12MEM3	读取/写入	0146h	未改变
ADC4 存储器 4	ADC12MEM4	读取/写入	0148h	未改变
ADC5 存储器 5	ADC12MEM5	读取/写入	014Ah	未改变
ADC6 存储器 6	ADC12MEM6	读取/写入	014Ch	未改变
ADC7 存储器 7	ADC12MEM7	读取/写入	014Eh	未改变
ADC8 存储器 8	ADC12MEM8	读取/写入	0150h	未改变
ADC9 存储器 9	ADC12MEM9	读取/写入	0152h	未改变
ADC10 存储器 10	ADC12MEM10	读取/写入	0154h	未改变
ADC11 存储器 11	ADC12MEM11	读取/写入	0156h	未改变
ADC12 存储器 12	ADC12MEM12	读取/写入	0158h	未改变
ADC13 存储器 13	ADC12MEM13	读取/写入	015Ah	未改变
ADC14 存储器 14	ADC12MEM14	读取/写入	015Ch	未改变
ADC15 存储器 15	ADC12MEM15	读取/写入	015Eh	未改变
ADC12 存储器控制 0	ADC12MCTL0	读取/写入	080h	用 POR 复位
ADC12 存储器控制 1	ADC12MCTL1	读取/写入	081h	用 POR 复位
ADC12 存储器控制 2	ADC12MCTL2	读取/写入	082h	用 POR 复位
ADC12 存储器控制 3	ADC12MCTL3	读取/写入	083h	用 POR 复位
ADC12 存储器控制 4	ADC12MCTL4	读取/写入	084h	用 POR 复位
ADC12 存储器控制 5	ADC12MCTL5	读取/写入	085h	用 POR 复位
ADC12 存储器控制 6	ADC12MCTL6	读取/写入	086h	用 POR 复位
ADC12 存储器控制 7	ADC12MCTL7	读取/写入	087h	用 POR 复位
ADC12 存储器控制 8	ADC12MCTL8	读取/写入	088h	用 POR 复位
ADC12 存储器控制 9	ADC12MCTL9	读取/写入	089h	用 POR 复位
ADC12 存储器控制 10	ADC12MCTL10	读取/写入	08Ah	用 POR 复位
ADC12 存储器控制 11	ADC12MCTL11	读取/写入	08Bh	用 POR 复位
ADC12 存储器控制 12	ADC12MCTL12	读取/写入	08Ch	用 POR 复位
ADC12 存储器控制 13	ADC12MCTL13	读取/写入	08Dh	用 POR 复位
ADC12 存储器控制 14	ADC12MCTL14	读取/写入	08Eh	用 POR 复位
ADC12 存储器控制 15	ADC12MCTL15	读取/写入	08Fh	用 POR 复位

23.3.1 ADC12CTL0，ADC12 控制寄存器 0

15	14	13	12	11	10	9	8
SHT1x				SHT0x			
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
MSC	REF2_5V	REFON	ADC120N	ADC12OVIE	ADC12TOVIE	ENC	ADC12SC
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

只有当 ENC=0 时可以被修改

SHT1x 位 15-12 采样保持时间。 这些位决定 ADC12MEM8 到 ADC12MEM15 寄存器在采样周期的 ADC12CLK 周期的数量。

- 0000 4 个 ADC12CLK 周期
- 0001 8 个 ADC12CLK 周期
- 0010 16 个 ADC12CLK 周期
- 0011 32 个 ADC12CLK 周期
- 0100 64 个 ADC12CLK 周期
- 0101 96 个 ADC12CLK 周期
- 0110 128 个 ADC12CLK 周期
- 0111 192 个 ADC12CLK 周期
- 1000 256 个 ADC12CLK 周期
- 1001 384 个 ADC12CLK 周期
- 1010 512 个 ADC12CLK 周期
- 1011 768 个 ADC12CLK 周期
- 1100 1024 个 ADC12CLK 周期
- 1101 1024 个 ADC12CLK 周期
- 1110 1024 个 ADC12CLK 周期
- 1111 1024 个 ADC12CLK 周期

SHT0x 位 11-8 采样保持时间。 这些位决定 ADC12MEM0 到 ADC12MEM7 寄存器在采样周期的 ADC12CLK 周期的数量。

- 0000 4 个 ADC12CLK 周期
- 0001 8 个 ADC12CLK 周期
- 0010 16 个 ADC12CLK 周期
- 0011 32 个 ADC12CLK 周期
- 0100 64 个 ADC12CLK 周期
- 0101 96 个 ADC12CLK 周期
- 0110 128 个 ADC12CLK 周期
- 0111 192 个 ADC12CLK 周期
- 1000 256 个 ADC12CLK 周期
- 1001 384 个 ADC12CLK 周期
- 1010 512 个 ADC12CLK 周期
- 1011 768 个 ADC12CLK 周期
- 1100 1024 个 ADC12CLK 周期
- 1101 1024 个 ADC12CLK 周期
- 1110 1024 个 ADC12CLK 周期
- 1111 1024 个 ADC12CLK 周期

MSC 位 7 多路采样和转换。 只在序列或重复模式中有效。

- 0 采样定时器需要 SHI 信号的一个上升沿来触发每个采样和转换。
- 1 第一个 SHI 信号的上升沿触发采样定时器，但后面的采样与转换在前一次转换完成后立即被自动执行。

REF2_5V 位 6 基准发生器电压。 REFON 也必须被置位。

- 0 1.5V
- 1 2.5V

REFON	位 5	基准电压发生器打开 0 基准关闭 1 基准打开
ADC12ON	位 4	ADC12 打开 0 ADC12 关闭 1 ADC12 打开
ADC12OVIE	位 3	ADC12MEMx 溢出中断使能。 也必须置位 GIE 位来启用该中断。 0 溢出中断被禁用 1 溢出中断被启用
ADC12TOVIE	位 2	ADC12 转换时间溢出中断使能。 也必须置位 GIE 位来启用该中断。 0 转换时间溢出中断被禁用 1 转换时间溢出中断被启用
ENC	位 1	启用转换 0 ADC12 被禁用 1 ADC12 被启用
ADC12SC	位 0	开始转换 软件控制的采样和转换启动。 ADC12SC 和 ENC 可以用同一个指令一起被置位。 ADC12SC 自动复位。 0 无采样和转换开始 1 启动采样和转换

23.3.2 ADC12CTL1，ADC12 控制寄存器 1

15	14	13	12	11	10	9	8
CSTARTADDx				**SHSx**		**SHP**	**ISSH**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ADC12DIVx			**ADC12SSELx**		**CONSEQx**		**ADC12BUSY**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

只有当 ENC=0 时可以被修改

CSTARTADDx 位 15-12 转换的起始地址。 这些位用于选择哪一个 ADC12 转换存储器寄存器是用于一个单次转换的或在一个序列中的第一个转换的。 CSTARTADDx 的值是 0 到 0Fh，与 ADC12MEM0 到 ADC12MEM15 相对应。

SHSx 位 11-10 采样保持源选择

- 00 ADC12SC 位
- 01 定时器_A.OUT1
- 10 定时器_B.OUT0
- 11 定时器_B.OUT1

SHP 位 9 采样保持脉冲模式选择。 此位选择采样信号 (SAMPCON) 源是作为采样定时器的输出还是直接作为采样输入信号输出。

- 0 SAMPCON 信号来源于采样输入信号。
- 1 SAMPCON 信号来源于采样定时器。

ISSH 位 8 反相信号采样保持

- 0 采样输入信号未反相。
- 1 采样输入信号被反相。

ADC12DIVx 位 7-5 ADC12 时钟分频器

- 000 /1
- 001 /2
- 010 /3
- 011 /4
- 100 /5
- 101 /6
- 110 /7
- 111 /8

ADC12SSELx 位 4-3 ADC12 时钟源选择

- 00 ADC12OSC
- 01 ACLK
- 10 MCLK
- 11 SMCLK

CONSEQx 位 2-1 转换序列模式选择

- 00 单通道，单次转换
- 01 序列通道
- 10 重复单通道
- 11 重复通道序列

ADC12BUSY 位 0 ADC12 忙。 该位标志着一个有效的采样或转换操作。

- 0 无操作被激活。
- 1 一个序列、采样、或转换是有效的。

23.3.3 ADC12MEMx，ADC12 转换存储器寄存器

15	14	13	12	11	10	9	8
0	0	0	0	转换结果			
r0	r0	r0	r0	rw	rw	rw	rw
7	6	5	4	3	2	1	0
转换结果							
rw	rw	rw	rw	rw	rw	rw	rw

转换结果 位 15-0 12 位转换结果是右对齐的。 位 11 是 MSB。 位 15-12 总是 0。写入转换存储寄存器会破坏结果。

23.3.4 ADC12MCTLx，ADC12转换存储控制寄存器

7	6	5	4	3	2	1	0
EOS	**SREFx**			**INCHx**			
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

只有当 ENC=0 时可以被修改

EOS 位 7 序列末尾。 表示在一个序列中的最后一个转换。
- 0 非序列末尾
- 1 序列结束。

SREFx 位 6-4 选择基准
- 000 V_{R+}= AV_{CC}和 V_{R-}= AV_{SS}
- 001 V_{R+}= V_{REF+} 和 V_{R-}= AV_{SS}
- 010 V_{R+}= V_{eREF+} 和 V_{R-}= AV_{SS}
- 011 V_{R+}= V_{eREF+} 和 V_{R-}= AV_{SS}
- 100 V_{R+}= AV_{CC}和 V_{R-}= V_{REF-}/ V_{eREF-}
- 101 V_{R+}= V_{REF+} 和 V_{R-}= V_{REF-}/ V_{eREF-}
- 110 V_{R+}= V_{eREF+} 和 V_{R-}= V_{REF-}/ V_{eREF-}
- 111 V_{R+}= V_{eREF+} 和 V_{R-}= V_{REF-}/ V_{eREF-}

INCHx 位 3-0 输入通道选择
- 0000 A0
- 0001 A1
- 0010 A2
- 0011 A3
- 0100 A4
- 0101 A5
- 0110 A6
- 0111 A7
- 1000 V_{eREF+}
- 1001 V_{REF-}/V_{eREF-}
- 1010 温度二极管
- 1011 (AV_{CC}- AV_{SS})/2
- 1100 GND
- 1101 GND
- 1110 GND
- 1111 GND

23.3.5 ADC12IE，ADC12 中断使能寄存器

15	14	13	12	11	10	9	8
ADC12IE15	**ADC12IE14**	**ADC12IE13**	**ADC12IE12**	**ADC12IE11**	**ADC12IE10**	**ADC12IFG9**	**ADC12IE8**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ADC12IE7	**ADC12IE6**	**ADC12IE5**	**ADC12IE4**	**ADC12IE3**	**ADC12IE2**	**ADC12IE1**	**ADC12IE0**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

ADC12IEx 位 15-0 中断使能。 这些位启用或禁用 ADC12IFGx 位的中断请求。

0 中断被禁用

1 中断被启用

23.3.6 ADC12IFG，ADC12 中断标志寄存器

15	14	13	12	11	10	9	8
ADC12IFG15	**ADC12IFG14**	**ADC12IFG13**	**ADC12IFG12**	**ADC12IFG11**	**ADC12IFG10**	**ADC12IFG9**	**ADC12IFG8**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ADC12IFG7	**ADC12IFG6**	**ADC12IFG5**	**ADC12IFG4**	**ADC12IFG3**	**ADC12IFG2**	**ADC12IFG1**	**ADC12IFG0**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

ADC12IFGx 位 15-0 ADC12MEMx 中断标志。 当用一个转换结果加载相应的 ADC12MEMx 时，这些位被置位。 如果相应的 ADC12MEMx 被访问，ADC12IFGx 位就被复位，或可以用软件复位。

0 无中断等待

1 中断等待

23.3.7 ADC12IV，ADC12 中断向量寄存器

15	14	13	12	11	10	9	8
0	0	0	0	0	0	0	0
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
0	0	**ADC12IVx**					0
r0	r0	r-(0)	r-(0)	r-(0)	r-(0)	r-(0)	r0

ADC12IVx 位 15-0 ADC12 中断矢量值

ADC12IV 内容	中断源	中断标志	中断优先级
000h	无中断等待	-	
002h	ADC12MEMx 溢出	-	最高
004h	转换时间溢出	-	
006h	ADC12MEM0 中断标志	ADC12IFG0	
008h	ADC12MEM1 中断标志	ADC12IFG1	
00Ah	ADC12MEM2 的中断标志	ADC12IFG2	
00Ch	ADC12MEM3 中断标志	ADC12IFG3	
00Eh	ADC12MEM4 中断标志	ADC12IFG4	
010h	ADC12MEM5 中断标志	ADC12IFG5	
012h	ADC12MEM6 中断标志	ADC12IFG6	
014h	ADC12MEM7 中断标志	ADC12IFG7	
016h	ADC12MEM8 中断标志	ADC12IFG8	
018h	ADC12MEM9 中断标志	ADC12IFG9	
01Ah	ADC12MEM10 中断标志	ADC12IFG10	
01Ch	ADC12MEM11 中断标志	ADC12IFG11	
01Eh	ADC12MEM12 中断标志	ADC12IFG12	
020h	ADC12MEM13 中断标志	ADC12IFG13	
022h	ADC12MEM14 中断标志	ADC12IFG14	
024h	ADC12MEM15 中断标志	ADC12IFG15	最低

TLV 结构

标签长度值 (TLV) 结构用在选定的 MSP430x2xx 器件中来提供器件的闪存存储器段 A 中的器件专用信息，例如校准数据。 有关与器件相关的执行信息，请参阅《器件专用数据表》。

Topic **Page**

24.1 TLV 介绍

TLV 结构段 A 中器件专用数据。 一个示例器件的段 A 的内容如在表 24-1中所示。

表 24-1. 示例区段 A结构

字地址	高位字节	低位字节	标签地址和偏移
0x10FE	CALBC1_1MHZ	CALDCO_1MHZ	0x10F6 + 0x0008
0x10FC	CALBC1_8MHZ	CALDCO_8MHZ	0x10F6 + 0x0006
0x10FA	CALBC1_12MHZ	CALDCO_12MHZ	0x10F6 + 0x0004
0x10F8	CALBC1_16MHZ	CALDCO_16MHZ	0x10F6 + 0x0002
0x10F6	0x08 (LENGTH)	TAG_DCO_30	0x10F6
0x10F4	0xFF	0xFF	
0x10F2	0xFF	0xFF	
0x10F0	0xFF	0xFF	
0x10EE	0xFF	0xFF	
0x10 EC	0x08 (LENGTH)	TAG_EMPTY	0x10 EC
0x10EA	CAL_ADC_25T85		0x10DA + 0x0010
8x10E0	CAL_ADC_25T30		0x10DA + 0x000E
6x10E0	CAL_ADC_25VREF_FACTOR		0x10DA + 0x000C
4x10E0	CAL_ADC_15T85		0x10DA + 0x000A
2x10E0	CAL_ADC_15T30		0x10DA + 0x0008
0x10E0	CAL_ADC_15VREF_FACTOR		0x10DA + 0x0006
0x10DE	CAL_ADC_OFFSET		0x10DA + 0x0004
0x10DC	CAL_ADC_GAIN_FACTOR		0x10DA + 0x0002
0x10DA	0x10 (LENGTH)	TAG_ADC12_1	0x10DA
0x10D8	0xFF	0xFF	
0x10D6	0xFF	0xFF	
0x10D4	0xFF	0xFF	
0x10D2	0xFF	0xFF	
0x10D0	0xFF	0xFF	
0x10CE	0xFF	0xFF	
0x10CC	0xFF	0xFF	
0x10CA	0xFF	0xFF	
0x10C8	0xFF	0xFF	
0x10C6	0xFF	0xFF	
0x10C4	0xFF	0xFF	
0x10C2	0x16 (LENGTH)	TAG_EMPTY	0x10C2
0x10C0	逐位异或运算的 2 的补码		0x10C0

段 A 的前两个字节 (0x10C0 和 0x10C1) 保持了段（地址 0x10C20x10FF）的余数的校验和。

第一标签位于地址 0x10C2 中，在这个例子中，是 TAG_EMPTY 标签。下面的字节 (0x10C3) 保持了以下结构的长度。 该TAG_EMPTY 结构的长度是 0x16 且，因此，下一个标签，TAG_ADC12_1，出现在 0x10DA 地址。同样，下面的字节保存了 TAG_ADC12_1 结构的长度。

TLV 结构将整个地址范围 0x10C2 映射到 段 A 的 0x10FF。 一个寻找开始于段 A 地址 0x10C2 的标签的程序例程，即使它存储在一个不同的（器件专用）绝对地址中，也提取所有的信息。

24.2 支持的标签

每个器件包含在表 24-2所示标签的一个子集。 有关详细信息请参阅《器件专用数据表》。

表 **24-2.** 支持的标签（器件专用）

标签	说明	值
TAG_EMPTY	识别一个未被使用的内存区域	0xFE
TAG_DCO_30	室温下 DCO 的校准值和DV_{CC}=3V	0x01
TAG_ADC12_1	ADC12 模块的校准值	0x08
TAG_ADC10_1	ADC10 模块的校准值	0x08

24.2.1 DCO 校准 TLV 结构

在对于 DCO 校准，使用了 BCS+ 寄存器（BCSCTL1 和 DCOCTL）。 存储在闪存信息存储区段 A 的值被写入 BCS+ 寄存器（请见表 24-3）。

表 **24-3. DCO** 校准数据（器件专用）

标签	说明	偏移量
CALBC1_1MHZ	1MHz BCSCTL1 寄存器的值，T_A=25℃	0x07
CALDCO_1MHZ	1MHz DCOCTL 寄存器的值，T_A=25℃	0x06
CALBC1_8MHZ	8MHz BCSCTL 寄存器的值，T_A=25℃	0x05
CALDCO_8MHZ	8MHz DCOCTL 寄存器的值，T_A=25℃	0x04
CALBC1_12MHZ	12MHz BCSCTL 寄存器的值，T_A=25℃	0x03
CALDCO_12MHZ	12MHz DCOCTL 寄存器的值，T_A=25℃	0x02
CALBC1_16MHZ	16MHz BCSCTL 寄存器的值，T_A=25℃	0x01
CALDCO_16MHZ	16MHz DCOCTL 寄存器的值，T_A=25℃	0x00

DCO 的校准数据适用于所有 2xx 器件中并存储在相同的绝对地址。 如果在Example 24-1中适用了采样编码，通过使用绝对寻址模式，就可以使用器件专用段 A 的内容。

***Example 24-1.** 使用绝对寻址模式的代码示例*

```
; Calibrate the DCO to 1 MHzCLR.B &DCOCTL ; Select lowest DCOx; and MODx settingsMOV.B
&CALBC1_1MHZ,&BCSCTL1 ; Set RSELxMOV.B &CALDCO_1MHZ,&DCOCTL ; Set DCOx and MODx
```

TLV 结构允许使用 TAG_DCO_30 标签的地址来寻址 DCO寄存器。Example 24-2显示了如何通过使用 TAG_DCO_30标签来寻址 DCO 校准数据。

***Example 24-2.** 使用**TLV** 结构的代码示例*

```
; Calibrate the DCO to 8 MHz; It is assumed that R10 contains the address of the TAG_DCO_30 tagCLR.B
&DCOCTL ; Select lowest DCOx and; MODx settingsMOV.B 7(R10),&BCSCTL1 ; Set RSELMOV.B 6(R10),&DCOCTL ;
Set DCOx and MODx
```

24.2.2 TAG_ADC12_1 校准 TLV结构

ADC12 模块的校准数据由八个字组成（请见表 24-4）。

表 24-4. TAG_ADC12_1 的校准数据（器件专用）

标签	说明	偏移量
CAL_ADC_25T85	VREF2_5=1，T_A= 85℃±2K，12 位转换结果	0x0E
CAL_ADC_25T30	VREF2_5=1，T_A=30℃± 2K，12 位转换结果	0x0C
CAL_ADC_25VREF_因子	VREF2_5=1, T_A=30℃±2K	0x0A
CAL_ADC_15T85	VREF2_5=0，T_A=85℃± 2K，12 位转换结果	0x08
CAL_ADC_15T30	VREF2_5=0，T_A=30℃±2K，12 位转换结果	0x06
CAL_ADC_15VREF_因子	VREF2_5=0，T_A=30℃±2K	0x04
CAL_ADC_偏移量	VeREF=2.5V，T_A=85℃±2K，$f_{ADC12CLK}$=5MHz	0x02
CAL_ADC_GAIN_FACTOR	VeREF=2.5V，T_A=85°C± 2K，$f_{ADC12CLK}$=5MHz	0x00

24.2.2.1 温度传感器的校准数据

通过使用内部基准电压来校准该温度传感器。 在VREF2_5=0 和 1 时，转换结果在 30°C 和 85°C 下被写在各自的区段地点（请见表 24-4）。

24.2.2.2 集成电压基准校准数据

基准电压（VREF2_5=0 和 1）是在室温下测定的。 在把测量的值存储到闪存中信息存储器段 A 中之前会将它标准化为 1.5V 或 2.5V。

$$CAL\_ADC\_15VREF\_FACTOR = (Ve_{REF} / 1.5\ V) \times 2^{15}$$

通过与 CAL_ADC_15VREF_FACTOR 相乘（或 CAL_ADC_25VREF_FACTOR）并把该结果除以 2^{15}来校正转换结果。

ADC（已校正） = ADC（原始的）× CAL_ADC_15VREF_因子 × $(1/2^{15})$

24.2.2.3 使用基准电压校准的示例

在下面的例子中，集成的 1.5V 基准电压用于转换过程。

- 转换结果：0X0100
- 基准电压校准因子 (CAL_ADC_15VREF_FACTOR)：0x7BBB

以下步骤演示了是一个例子是如何通过使用硬件乘法器校正 ADC12 转换结果的：

- 把转换的结果乘以 2（这一步简化了最后的除法）。
- 把该结果乘以CAL_ADC_15VREF_FACTOR。
- 把结果除以 2^{16}（使用32 位上部字的相乘结果 RESHI）。

在这个例子中：

- 0x0100 × 0x0002=0x0200
- 0x0200 × 0x7BBB=0x00F7_7600
- 0x00F7_7600 ÷ 0x0001_0000 = 0x0000_00F7 (=247)

以下是使用硬件乘法器的代码示例。

```
; The ADC conversion result is stored in ADC12MEM0; It is assumed that R9 contains the address of
the; TAG_ADC12_1.; The corrected value is available in ADC_CORMOV.W &ADC12MEM0,R10 ; move result
to R10RLA.W R10 ; R10 x 2MOV.W R10,&MPY ; unsigned multiply OP1MOV.W
CAL_ADC_15VREF_FACTOR(R9),&OP2; calibration value OP2MOV.W &RESHI,&ADC_COR ; result: upper 16-
```

```
bit MPY
```

24.2.2.4 偏移和增益校准数据

在段 A 中 ADC12 的偏移量被确定并存储为二补码数。 通过把 CAL_ADC_OFFSET 添加到转换结果中来完成偏移误差校正。

ADC（偏移量_已校正）= ADC（原始） + CAL_ADC_OFFSET

ADC12 的增益，存储在偏移量 0x00 中，用以下公式计算。

CAL_ADC_GAIN_FACTOR =（1 / 增益）× 2^{15}

是通过将它乘以CAL_ADC_GAIN_FACTOR 并把该结果除以 2^{15}来增益纠正该转换结果。

ADC（增益_已校正）=ADC（原始）× CAL_ADC_GAIN_FACTOR × ($1/2^{15}$)

如果增益和偏移量这两个都要进行校正，那么首先进行增益校正。

ADC（增益_已校正）=ADC（原始）× CAL_ADC_GAIN_FACTOR × ($1/2^{15}$)

ADC（最终）=ADC（增益_已校正）+ CAL_ADC_OFFSET

24.2.2.5 使用增益和偏移量校准的例子

在下面的例子中，在一个转换期间使用了一个外部集成的基准电压。

- 转换结果：0x0800 (=2048)
- 增益校准系数：0x7FE0（增益误差：+2 LSB）
- 偏移校准：0xFFFE（-2 的 2S补码）

以下步骤演示了一个例子是如何通过使用硬件乘法器校正 ADC12 转换结果的：

- 把转换的结果乘以 2（这一步简化了最后的除法）。
- 把该结果乘以CAL_ADC_GAIN_FACTOR。
- 把结果除以 2^{16}（使用32 位乘法结果 RESHI 的上部字）。
- 把 CAL_ADC_OFFSET 添加到该结果中。

在这个例子中：

- 0x0800+(0x*0002)=0x1000
- 0x1000×0x8010=0x0801_0000
- 0x0801_0000÷0x0001_0000=0x0000_0801 (=2049)
- 0x801+0xFFFE=0x07FF (=2047)

以下是使用硬件乘法器的代码示例。

```
; The ADC conversion result is stored in ADC12MEM0; It is assumed that R9 contains the address of
the TAG_ADC12_1.; The corrected value is available in ADC_CORMOV.W &ADC12MEM0,R10 ; move result
to R10RLA.W R10 ; R10 * 2MOV.W R10,&MPY ; unsigned multiply OP1MOV.W
CAL_ADC_GAIN_FACTOR(R9),&OP2; calibration value OP2MOV.W &RESHI,&ADC_COR ; use upper 16-
bit MPYADD.W CAL_ADC_OFFSET(R9),&ADC_COR; add offset correction
```

24.3 检查段 A 的完整性

64 字节段 A 包含一个 2 字节的存储在地址为 0x10C0 和 0x10C1 的 0x10C2 至 0x10FF处的数据的校验和。 校验和是一个存储在二补码数据格式中的 31 字的逐位异或运算。

一个计算以下校验和的代码示例。

```
; Checking the SegmentA integrity by calculating the 2's; complement of the 31 words at 0x10C2 -
 0x10FE.; It is assumed that the SegmentA Start Address is stored; in R10. R11 is initialized to
0x00.; The label TLV_CHKSUM is set to 0x10C0.ADD.W #2,R10 ; Skip the checksumLP0 XOR.W @R10+,R11
; Add a word to checksum CMP.W #0x10FF,R10 ; Last word included?JN LP0 ; No, add more dataADD.W
&TLV_CHKSUM,R11 ; Add checksumJNZ CSNOK ; Checksum not ok... ; Use SegmentA dataCSNOK ... ; Do
not use SegmentA Data
```

24.4 分解段 A 的 TLV 结构

分析以下区段 A 的代码示例。

```
; It is assumed that the SegmentA start address; is stored in R10.LP1 ADD.W #2,R10 ; Skip two
bytesCMP.W #0x10FF,R10 ; SegmentA end reached?JGE DONE ; Yes, doneCMP.B #TAG_EMPTY,0(R10) ;
TAG_EMPTY?JNZ T1 ; No, continueJMP LP2 ; Yes, done with TAG_EMPTYT1 CMP.B #TAG_ADC12_1,0(R10) ;
TAG_ADC12_1?JNZ T2 ; No, continue... ; Yes, found TAG_ADC12_1JMP LP2 ; Done with TAG_ADC12_1T2
CMP.B #DCO_30,0(R10) ; TAG_DCO_30?JNZ T3 ; No, continueCLR.B &DCOCTL ; Select lowest DCOxMOV.B
7(R10),&BCSCTL1 ; Yes, use e.g. 8MHz data andMOV.B 6(R10),&DCOCTL ; set DCOx and MODxJMP LP2 ;
Done with TAG_DCO_30T3 ... ; Test for "next tag"... ; JMP LP2 ; Done with "next tag"LP2 MOV.B
1(R10),R11 ; Store LENGTH in R11ADD.W R11,R10 ; Add LENGTH to R10JMP LP1 ; Jump to continue
analysisDONE ;
```

DAC12

DAC12 模块是一宽 12 位电压输出数模转换器 (DAC)。 本章描述了 MSP430x2xx 器件系列的 DAC12 模块的运行。

Topic **Page**

25.1 DAC12 介绍

DAC12 模块是一个 12 位电压输出数模转换器 (DAC)。 DAC12 可在 8 位或者 12 位模式中被配置并可与 DMA 控制器协同使用。 当多个 DAC12 模块出现时，可将它们编成一组进行同步更新操作。

DAC12 的特性包括：

- 12 位单片输出
- 8 位或 12 位电压输出分辨率
- 可编程的稳定时间与功耗间的关系
- 内部或者外部基准电压选择
- 直接二进制或 2 补码数据格式
- 针对偏移校正的自校准选项
- 多个 DAC12 模块的同步更新功能

注: 多个 DAC12 模块

有些器件内置了一个以上的 DAC12 模块。 如果一个器件上出现一个以上的 DAC12 模块，则多 DAC12 模块完全相同的运行。

在这一整章中，将会出现如 DAC12_xDAT 或 DAC12_xCTL 的命名来描述寄存器名称。 这种情况下，x 被用于指代正在被讨论的 DAC12 模块。 在操作完全相同的情况下，寄存器被简写为 DAC12_xCTL。

DAC12 模块反馈图如图 25-1所示。

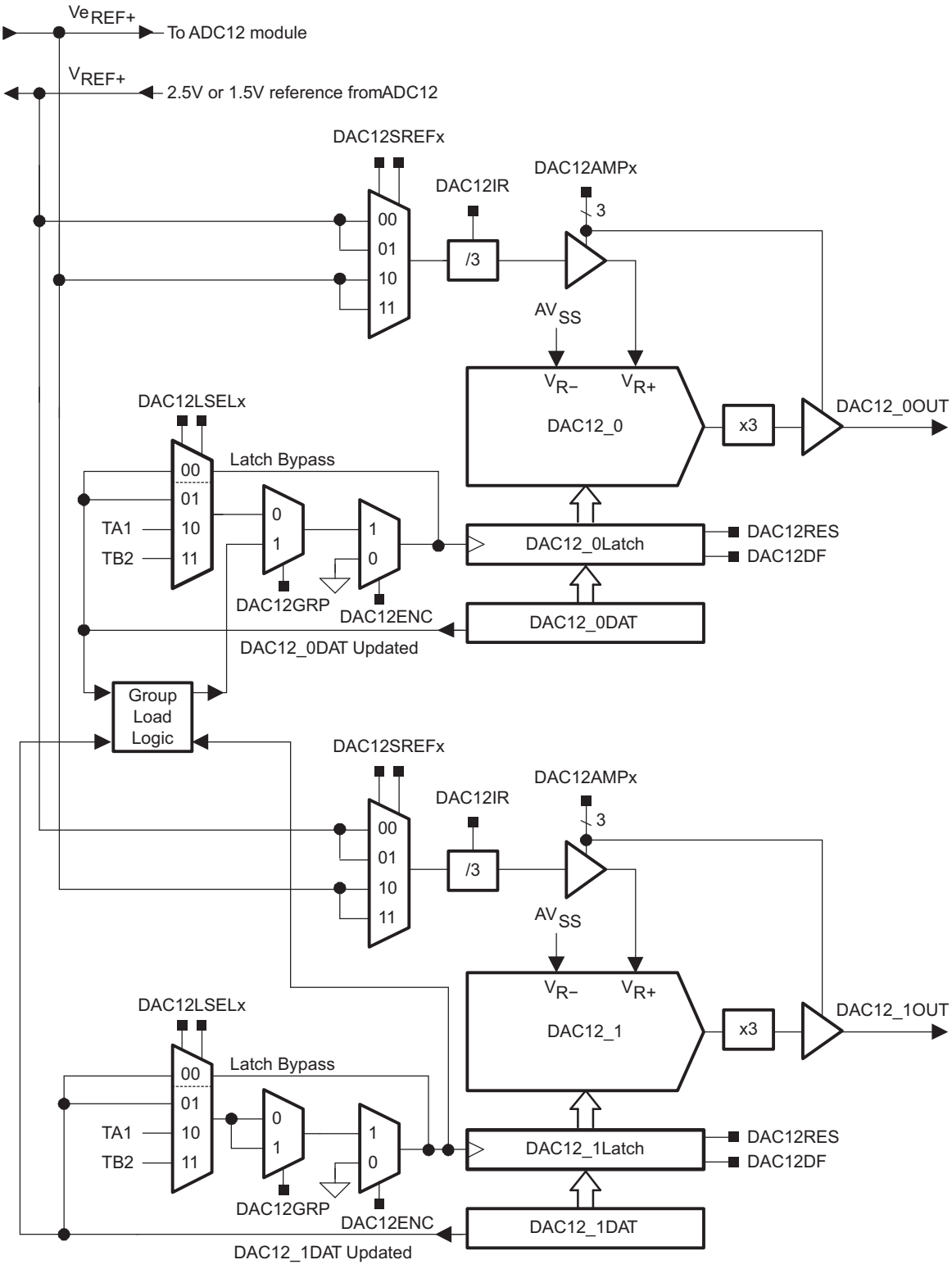


图 **25-1. DAC12** 反馈图

25.2 DAC12 运行

DAC12 模块可由用户软件配置。 DAC12 的运行和建立在下列章节中进行讨论。

25.2.1 DAC12 内核

DAC12 可以在 8 位或 12 位模式中使用 DAC12RES 位配置运作中。 满量程输出可以通过 DAC12IR 位被编程为 1x 或 3x 选择的基准电压。 此功能允许了用户控制 DAC12 的动态范围。 DAC12DF 位允许用户为 DAC 在直节二进制和 2 补码数据格式之间进行选择。 使用直节二进制数据格式时，输出电压公式在表 25-1 中给出。

表 25-1. DAC12 满量程范围（V_{REF}=V_{eREF+}或 V_{REF+}）

分辨率	DAC12RES	DAC12IR	输出电压公式
12 位	0	0	$V_{OUT} = V_{REF} \times 3 \times \frac{DAC12\_xDAT}{4096}$
12 位	0	1	$V_{OUT} = V_{REF} \times \frac{DAC12\_xDAT}{4096}$
8 位	1	0	$V_{OUT} = V_{REF} \times 3 \times \frac{DAC12\_xDAT}{256}$
8 位	1	1	$V_{OUT} = V_{REF} \times \frac{DAC12\_xDAT}{256}$

在 8 位模式中，DAC12_xDAT 的最大可用值值为 0FFh。 在 12 位模式中，DAC12_xDAT 的最大可用值值为 0FFFh。 大于这些的值可能会被写入寄存器，但前导位会被忽略。

25.2.1.1 DAC12 端口选择

DAC12 输出与端口 P6 引脚和 ADC12 模拟量输入，和 V_{EREF+}引脚是复用的。 当 DAC12AMPx>0 时，DAC12 引脚将自动选择功能，而不管相关的 PxSELx 和 PxDIRx 位状态。 DAC12OPS 位将在 P6 引脚和 V_{EREF+}引脚之间为 DAC 输出进行选择。 例如，当 DAC12OPS=0 时，DAC12_0 在 P6.6 上输出并且 DAC12_1 在 P6.7 上输出。当 DAC12OPS=1 时，DAC12_0 在 V_{eREF+}上输出并且 DAC12_1 在 P6.5 上输出。更多详细信息请参阅《器件专用数据表》中的端口引脚的原理图。

25.2.2 DAC12 基准

DAC12 的基准被配置为使用外部基准电压或使用来自于带有 DAC12SREFx 位的 ADC12 模块的内部 1.5-V/2.5-V 基准电压。 当 DAC12SREFx={0,1}时，V_{REF+}信号被用作基准电压并且当 DAC12SREFx={2,3} 时，V_{eREF+}信号被用作基准电压。

想要使用 ADC12 内部基准电压，必须通过合适的 ADC12 控制位启用和配置它。

25.2.2.1 DAC12 基准输入和电压输出缓冲器

DAC12 的基准输入和电压输出缓冲器可以配置优化的建立时间与功耗。 使用 DAC12AMPx 位选择 8 种组合。 在低/低设置中，建立时间是最慢的，并且两个缓冲器的电流消耗也是最低的。 中高级设置中比较快的建立时间，而且电流消耗也会增加。 有关参数请参阅《器件专用数据表》

25.2.3 更新 DAC12 电压输出

DAC12_xDAT 寄存器可直接连接到 DAC12 内核或被双缓冲。 最新的 DAC12 电压输出触发器由 DAC12LSELx 位选择。

当 DAC12LSELx=0 时，数据锁存器是透明的并且 DAC12_xDAT 寄存器被直接应用于 DAC12 内核。当新的 DAC12 数据被写入 DAC12_xDAT 寄存器时，不管 DAC12ENC 位的状态，DAC12 输出会立即更新。

当 DAC12LSELx=1 时，DAC12 的数据被锁存并且在新的数据被写入到 DAC12_xDAT 后该数据直接应用于 DAC12 内核。 当 DAC12LSELx=2 或 3 时，数据分别被锁存在从定时器_A CCR1 输出的或定时器_B CCR2 输出的上升沿上。 当 DAC12LSELx>0 时，DAC12ENC 必须被置位以锁存数据。

25.2.4 DAC12_xDAT 数据格式

DAC12 支持直接二进制和 2 补码两种数据格式。 当使用直二进制数据格式时，在 12 位模式中满量程输出值是 0FFFh（在 8 位模式中是0FFh）如图 25-2所示。

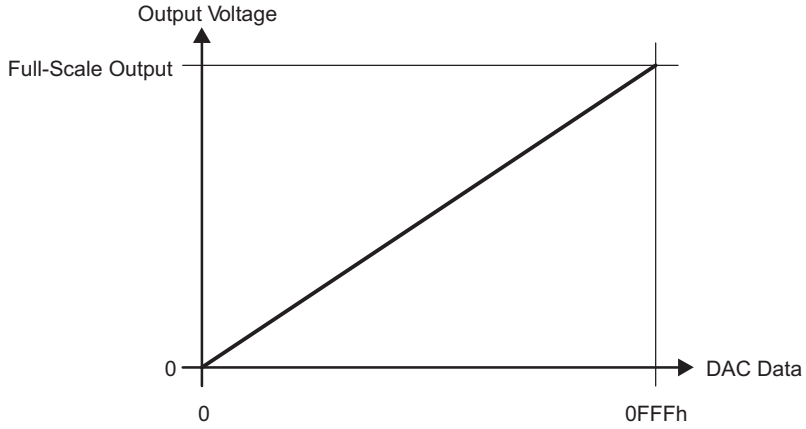


图 **25-2.** 输出电压与 **DAC12** 数据，**12** 位，直节二进制模式。

当使用 2 补码数据格式时，使得一个 DAC12_xDAT 的值在 0800h 范围内转换（在 8 位模式中是 0800h）导致了一个 0 输出电压，0000h 是半量程输出电压，和 07FFh（8 位模式为 007Fh）是满量程输出电压（见 图 25-3）。

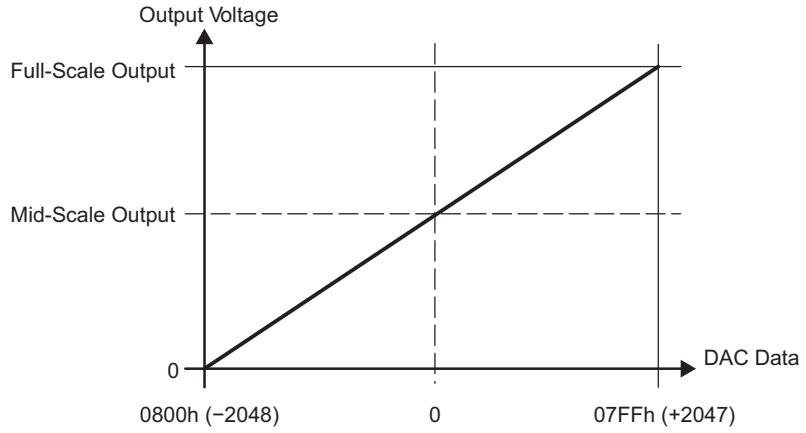


图 **25-3.** 输出电压与 **DAC12** 数据，**12** 位，**2** 补码模式

25.2.5 DAC12 输出放大器的失调校准

DAC12 输出放大器的偏置电压可以是正的或负的。 当偏移量为负时，输出放大器试图驱动负电压，但不能成功。 输出电压一直保持为 0 直到 DAC12 数字输入产生一个足够的输出电压来克服负偏移电压，从而产生了图 25-4中所示的传递函数。

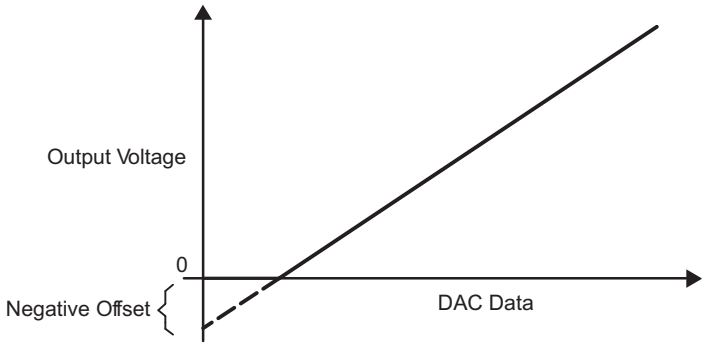


图 **25-4.** 负偏移

当输出放大器有一个正偏移时，一个值为 0 的数字输入不会导致输出电压为零。 DAC12 的输出电压会在 DAC12 数据达到最大代码前达到最大输出电平。 图 25-5显示了这一过程。

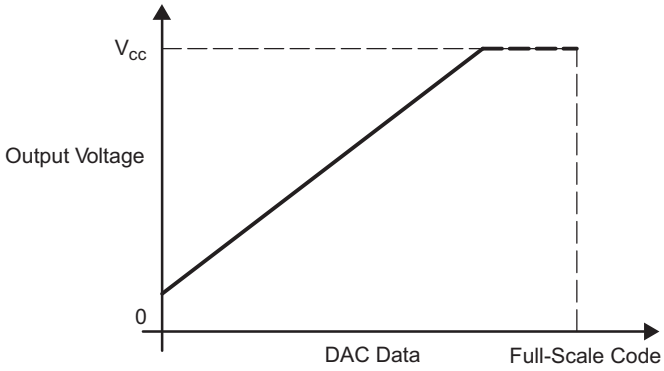


图 **25-5.** 正偏移

DAC12 具有校准输出放大器的偏移电压的功能。 设置 DAC12CALON 位来启动偏移校准。 使用 DAC12 之前应完成校准。 校准完成后，DAC12CALON 位自动复位。 校准前应先配置 DAC12AMPx 位。 为获得最佳校准结果，在校准过程中，应尽量减少端口和 CPU 的活动。

25.2.6 编组多个 DAC12 模块

多个 DAC12 可以被组合在一起使用 DAC12GRP 位同步更新每个 DAC12 输出。 硬件确保 DAC12 模块组中的所有更新，同时也确保了任何中断或 NMI 事件的独立。

通过设置 DAC12_0 的 DAC12GRP 位编组 DAC12_0 和 DAC12_1。 DAC12_1 中的 DAC12GRP 位不影响。 当 DAC12_0 和 DAC12_1 被编组时：

- DAC12_1 DAC12LSELx 位为两种 DAC 选择更新触发。
- 两种 DAC 的 DAC12LSELx 位都必须> 0。
- 两种 DAC 的 DAC12ENC 位都必须被设置为 1

当 DAC12_0 和 DAC12_1 被编组时，即使一种或两种 DAC 数据没有发生改变，在数据更新前两种 DAC12_xDAT 寄存器也必须被写入。图 25-6显示了 DAC12_0 和 DAC12_1 编组的一个锁存更新时序例子。

当 DAC12_0 DAC12GRP=1 和两种 DAC12_x DAC12LSELx>0 及任一 DAC12ENC=0 时，任一 DAC12 都不更新。

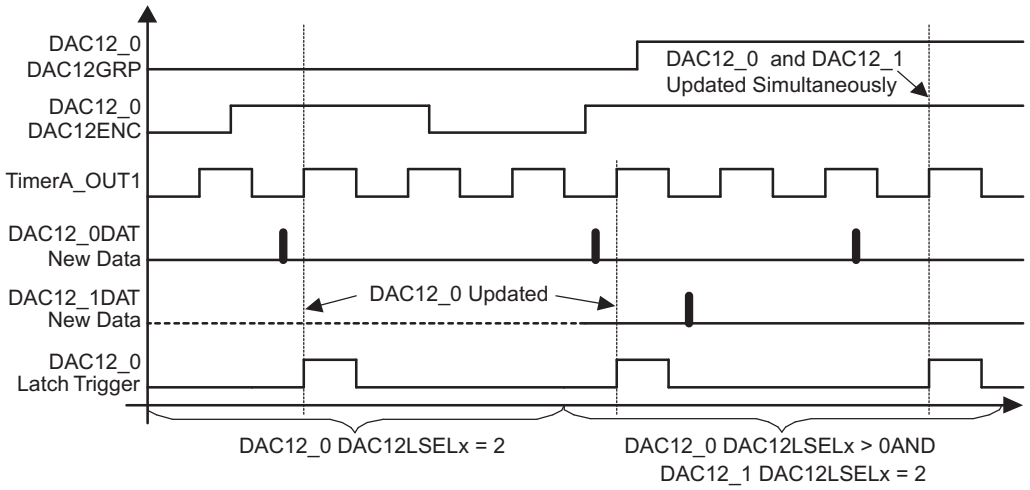


图 **25-6. DAC12** 组更新举例，定时器_**A3** 触发器

注: **DAC12** 稳定时间

DMA 控制器能够将 DAC12 输出可以解决数据更快的传输至 DAC12。 在使用 DMA 控制器时，用户必须保证 DAC12 稳定时间没有被占用。 对于参数请参阅《器件专用数据表》

25.2.7 DAC12 中断

在 某些器件上 DAC12 中断向量与 DMA 控制器共享（参见《器件专用数据表》的中断分配）。 在这种情况下，软件必须检查 DAC12IFG 和 DMAIFG 标志位来判断中断源。

当 DAC12LSELx>0，从 DAC12_xDAT 寄存器到数据锁存器的数据被锁存时，DAC12IFG 被置位。 当 DAC12LSELx=0 时，DAC12IFG 的标志没有被置位。

一个置位 DAC12IFG 位表明了 DAC12 正准备传输新数据。 如果 DAC12IE 和 GIE 都被置位，则 DAC12IFG 产生一个中断请求。 DAC12IFG 标志不会被自动复位。 它必须由软件复位。

25.3 DAC12 寄存器

在表 25-2中列出了 ADC12 寄存器。

表 25-2. DAC12 寄存器

寄存器	简式	寄存器类型	地址	初态
DAC12_0 控制	DAC12_0CTL	读取/写入	01C0h	用 POR 复位
DAC12_0 数据	DAC12_0DAT	读取/写入	01C8h	用 POR 复位
DAC12_1 控制	DAC12_1CTL	读取/写入	01C2h	用 POR 复位
DAC12_1 数据	DAC12_1DAT	读取/写入	01CAh	用 POR 复位

25.3.1 DAC12_xCTL，DAC12 控制寄存器

15	14	13	12	11	10	9	8
DAC12OPS	**DAC12SREFx**		**DAC12RES**	**DAC12LSELx**		**DAC12CALON**	**DAC12IR**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
DAC12AMPx			**DAC12DF**	**DAC12IE**	**DAC12IFG**	**DAC12ENC**	**DAC12GRP**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

只有当 DAC12ENC=0 时，才能被修改。

DAC12OPS 位 15 DAC12 输出选择

- 0 DAC12_0 在 P6.6 上输出，DAC12_1 在 P6.7 上输出
- 1 DAC12_0 在 V_{eREF+} 上输出，DAC12_1 在 P6.5 上输出。

DAC12SREFx 位 14-13 DAC12 选择基准电压

- 00 V_{REF+}
- 01 V_{REF+}
- 10 V_{eREF+}
- 11 V_{eREF+}

DAC12RES 位 12 DAC12 分辨率选择

- 0 12 位分辨率
- 1 8 位分辨率

DAC12LSELx 位 11-10 DAC12 负载选择。 为 DAC12 锁存器选择负载触发器。 为了 DAC 的更新，除了当 DAC12LSELx=0 时，DAC12ENC 必须被置位。

- 00 当 DAC12_xDAT 被写入时，DAC12 锁存负载（DAC12ENC 被忽略）
- 01 当 DAC12_xDAT 被写入时，DAC12 锁存负载，或者，当编组时，当组合内所有 DAC12_xDAT 寄存器都已经被写入时，DAC12 锁存负载。
- 10 定时器_A.OUT1 (TA1) 的上升沿
- 11 定时器_B.OUT2 (TB2) 的上升沿

DAC12CALON 位 9 DAC12 校准打开。 该位启动 DAC12 偏移校准序列并且校准完成后，会自动复位。

- 0 校准未激活
- 1 启动校准/校准正在进行中

DAC12IR 位 8 DAC12 的输入范围。 该位设置基准输入和输出电压范围。

- 0 DAC12 满量程输出 = 3 倍基准电压
- 1 DAC12 满量程输出 = 1 倍基准电压

DAC12AMPx 位 7-5 DAC12 放大器设置。 这些位为 DAC12 输入和输出放大器选择稳定时间与电流消耗。

DAC12AMPx	输入缓冲器	输出缓冲器
000	关闭	DAC12 关闭，输出高 Z
001	关闭	DAC12 关闭，输出 0V
010	低速/电流	低速/电流
011	低速/电流	中速/电流
100	低速/电流	高速/电流
101	中速/电流	中速/电流
110	中速/电流	高速/电流
111	高速/电流	高速/电流

DAC12DF 位 4 DAC12 数据格式

- 0 直接二进制
- 1 2 补码

DAC12IE 位 3 DAC12 中断使能

- 0 被禁用
- 1 被启用

DAC12IFG	位 2	DAC12 中断标志 0 无中断等待 1 中断等待
DAC12ENC	位 1	DAC12 使能转换。 当 DAC12LSELx>0 时，该位启用 DAC12 模块。当 DAC12LSELx=0 时，DAC12ENC 被忽略。 0 DAC12 被禁用 1 DAC12 被启用
DAC12GRP	位 0	DAC12 组。 将 DAC12_x 与下一个更高的 DAC12_x 编组。 不用于 DAC12_1。 0 没被编组 1 被编组

25.3.2 DAC12_xDAT，DAC12 数据寄存器

15	14	13	12	11	10	9	8
0	0	0	0	**DAC12 数据**			
r(0)	r(0)	r(0)	r(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
DAC12 数据							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

未被使用	位 15-12	未被使用。 这些位总是 0 并且不影响 DAC12 内核。
DAC12 数据	位 11-0	DAC12 数据

DAC12 数据格式	DAC12 数据
12 位的二进制	DAC12 数据是右对齐的。 位 11 是 MSB。
12 位 2 补码	DAC12 数据是右对齐的。 位 11 是 MSB（符号）。
8 位的二进制	DAC12 数据是右对齐的。 位 7 是 MSB。 位 11-8 无关并且不会影响 DAC12 内核。
8 位 2 补码	DAC12 数据是右对齐的。 位 7 是 MSB（符号）。 位 11-8 无关并且不会影响 DAC12 内核。

SD16_A

SD16_A 是一个单转换 16 位三角积分模数转换模块，它具有高阻抗输入缓冲器。 本章介绍了 SD16_A 模块。 SD16_A 模块在 MSP430x20x3 器件中执行。

Topic **Page**

26.1 SD16_A 介绍

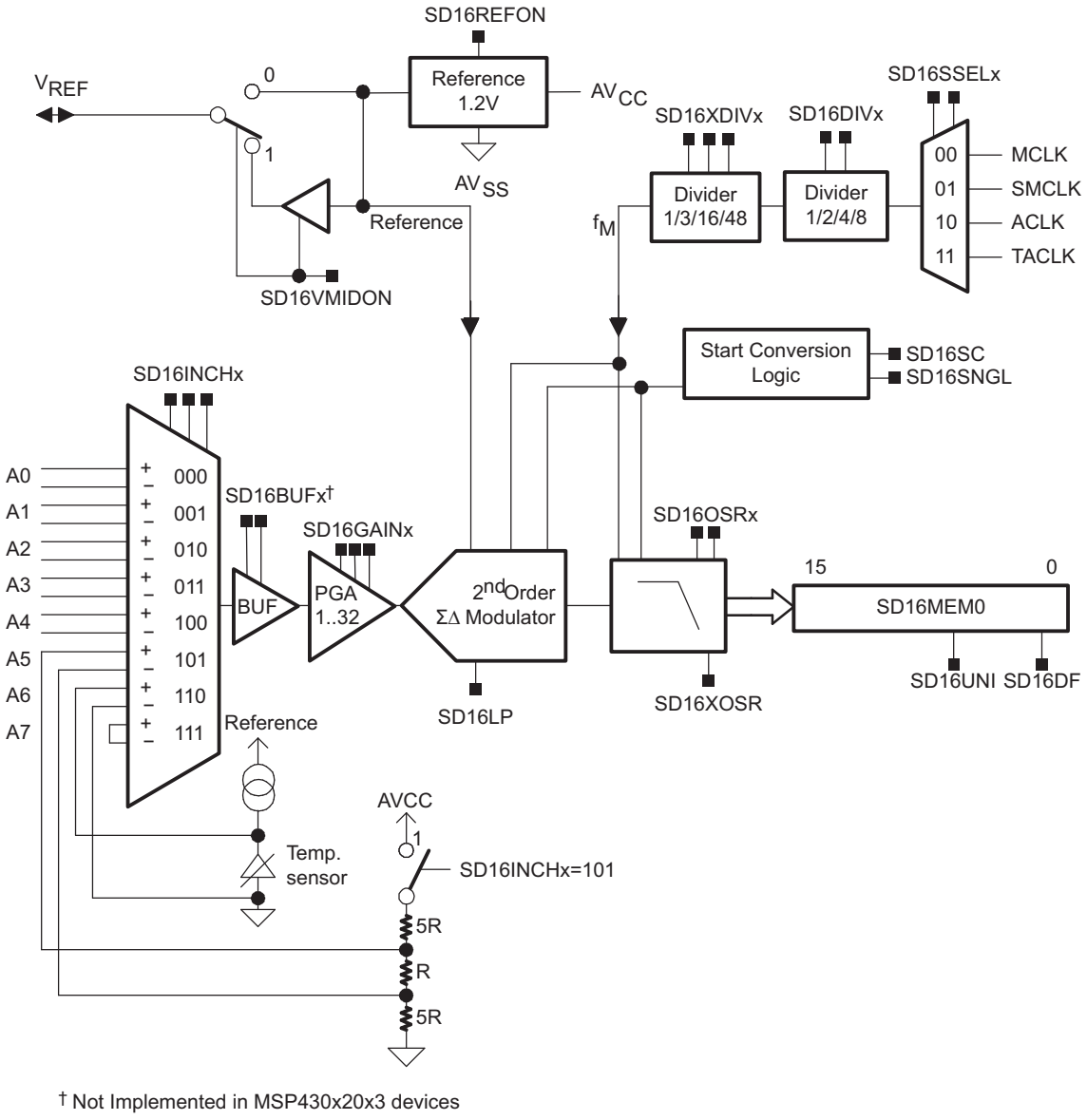
SD16_A 模块由一个三角积分模数转换器组成，此转换器带有一个高阻抗输入缓冲器和内部参考基准。 它拥有多达 8 个的差分式复合输入对，并包含一个内置温度传感器和一个分电源电压。 该模数转换器是基于二阶过采样的三角积分调节器和数字抽取滤波器。 该抽取滤波器是一种梳状滤波器，它的过采样率是可选择的，最高可以达到 1024。 额外的滤波可以用软件实现。

在 MSP430x20x3 器件中没有执行高阻抗输入缓冲。

SD16_A 的特点包括：

- 16 位三角积分结构
- 每通道多达 8 个多路复用差分模拟输入（输入端的数量取决于器件，请参阅《器件专用数据手册》。）
- 软件可选片上基准电压生成 (1.2V)
- 软件可选内部或外部基准电压
- 内置温度传感器
- 高达 1.1MHZ 的调节器输入频率
- 高阻抗输入缓（在所有器件上未执行，请参阅《器件专用数据表》）
- 可选低功耗转换模式

在图 26-1中给出了 SD16_A 的模块方框图。



† Not Implemented in MSP430x20x3 devices

图 **26-1. SD16_A** 方框图

26.2 SD16_A 操作

SD16_A 模块用用户软件进行配置。 在以下章节详细阐述了 SD16_A 的设置和操作。

26.2.1 ADC 芯片

模数转换是由一个 1 位二阶三角积分调节器实现的。 调节器中的一个单位比较器通过调节器频率 f_M量化输入信号。 所产生的 1 位数据流由数字滤波器平均分配作为转化结果。

26.2.2 模拟输入范围和 PGA

各模拟输入对的满量程输入电压范围由各通道的可编程放大器增益设置来决定。 最大满量程范围为 $\pm V_{FSR}$其中，V_{FSR}由以下公式定义：

$$V_{FSR} = \frac{V_{REF}/2}{GAIN_{PGA}}$$

对于一个 1.2V 的基准电压，在增益为 1 时，最大全程输入范围为：

$$\pm V_{FSR} = \frac{1.2\ V/2}{1} = \pm 0.6\ V$$

有关满量输入规格请参阅《器件专用数据表》。

26.2.3 基准电压发生器

SD16_A 模块有一个 1.2v 的内置基准电压 可以通过 SD16REFON 位来启用它。 当使用内部基准电压时，为了减少噪声，建议用一个外部 100nF 电容把 V_{REF}连接到 AV_{SS}。 当 SD16VMIDON = 1 时，内部基准电压可以在片外使用。缓冲输出可提供高达 1mA 的驱动。 当使用片外内部基准电压时，需要在 V_{REF}至 AV_{SS}之间接一个 470nF 的电容。 详细参数请参阅《器件专用数据表》。

当 SD16REFON 和 SD16VMIDON 都复位时，外部电压可以用于 V_{REF}输入端。

26.2.4 自动断电

SD16_A 是专为低功耗应用而设计的。 当 SD16_A 不进行转换时，它就会自动关闭，而当一个转换开始时又自动重新使能。 基准电压不会自动禁止，但是可以通过设置 SD16REFON = 0 来关闭。当 SD16_A 或基准电压被禁止时，它们不消耗电流。

26.2.5 模拟输入对选择

SD16_A 可将多达 8 个不同的差分输入对复路到 PGA。 在器件上的多达 5 个模拟输入对 (A0-A4) 对外是可用的。 通过使用 A5 多路复用器输入可以使一个用于检测电源电压的电阻分压器可用。 通过使用 A6 多路复用器输入可以使一个内部温度传感器可用。 A7 是 + 和 - 输入对之间的短路连接，且可用于校准 SD16A 的输入级的偏移。

26.2.5.1 模拟输入设置

模拟输入是通过使用 SD16INCTL0 和 SD16AE 寄存器来配置的。 SD16INCHx 位选择了模拟多路复用器的 8 个不同差分输入对中的其中一个。 PGA 的增益由 SD16GAINx 决定。 共有 6 个增益设置可用。 SD16AEx 位用于使能或禁止模拟输入引脚。 设置任何 SD16AEx 位可以禁止相应引脚的复用数字电路。 有关引脚图请参阅《器件专用数据表》。

在转换过程中，对 SD16INCHx 和 SD16GAINx 位的任何修改都会在下一个数字滤波器的采样周期生效。 由于数字滤波器的建立时间，在这些位被修改之后，接下来的 3 次转换可能会无效。 这可由 SD16INTDLYx 位自动处理。 当 SD16INTDLY = 00h 时，在一个转换开始条件后，转换中断请求将在第 4 次转换时才开始。

可以通过使用 SD16BUFx 位来使能器件上正在执行的高阻抗输入缓冲。 设置的速度根据 SD16A 调节器频率来选择，如在表 26-1中所示。

表 26-1. 高输入阻抗缓冲器

SD16BUFx	缓冲器	SD16 调制器频率 f_M
00	缓冲被禁止	
01	低速/电流	f_M< 200kHz
10	中速/电流	200kHz < f_M< 700kHz
11	高速/电流	700kHz < f_M< 1.1MHz

建议为 SD16_A 添加一个 RC 反锯齿滤波器来防止输入信号失真。 对于一个 1MHz 的调制器时钟且 OSR=256 时，截止频率应该 < 10kHz。 在带宽要求较低的应用中，截止频率可以设置为一个较低频率。

26.2.6 模拟输入特性

SD16_A 使用一个开关电容输入级，该输入极对外部电路来说就如一个阻抗，如在图 26-2中所示。

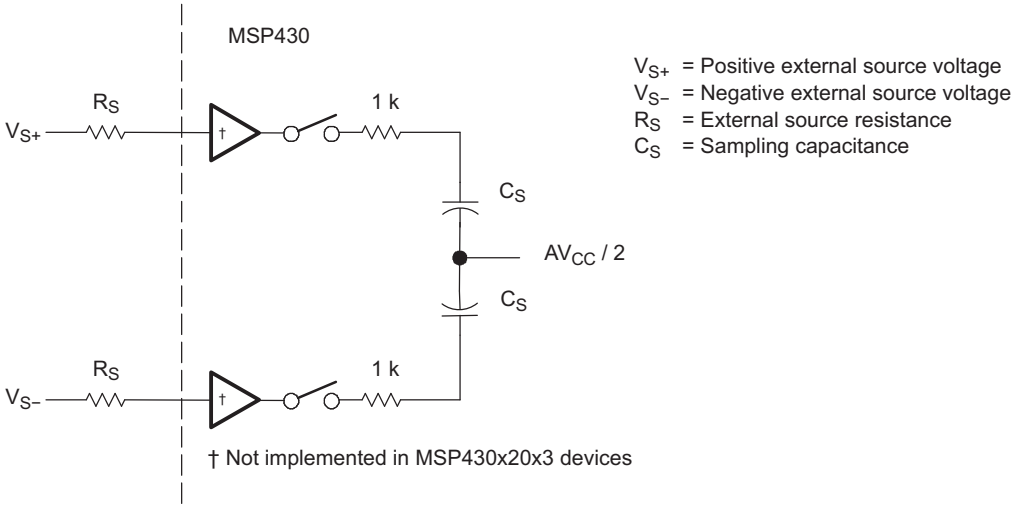


图 26-2. 模拟输入等效电路

当使用缓冲器时，R_S不影响采样频率 f_S。 然而，当不使用缓冲器时，或器件上没有缓冲器时，最大采样频率 f_S可以由采样电路的最小建立时间 $t_{稳定时间}$来计算：

$$t_{Settling} \geq (R_S + 1\,k\Omega) \times C_S \times \ln\left(\frac{GAIN \times 2^{17} \times V_{Ax}}{V_{REF}}\right)$$

其中

$$f_S = \frac{1}{2 \times t_{Settling}} \quad \text{and} \quad V_{Ax} = \max\left(\left|\frac{AV_{CC}}{2} - V_{S+}\right|, \left|\frac{AV_{CC}}{2} - V_{S-}\right|\right)$$

V_{S+}和 V_{S-}以 AV_{SS}为基准。

C_S随着如表 26-2中所示的增益设置而不同。

表 **26-2.** 采样电容

PGA 增益	采样电容，C_S
1	1.25pF
2, 4	2.5pF
8	5pF
16, 32	10pF

26.2.7 数字滤波器

数字滤波器用一个 SINC^{3}梳状滤波器来处理调制器的 1 位的数据流。 在 Z 域表述了该传递函数。

$$H(z) = \left(\frac{1}{OSR} \times \frac{1 - z^{-OSR}}{1 - z^{-1}} \right)^3$$

频域传递函数为：

$$H(f) = \left[\frac{\text{sinc}\left(OSR \times \pi \times \frac{f}{f_M} \right)}{\text{sinc}\left(\pi \times \frac{f}{f_M} \right)} \right]^3 = \left[\frac{1}{OSR} \times \frac{\sin\left(OSR \times \pi \times \frac{f}{f_M} \right)}{\sin\left(\pi \times \frac{f}{f_M} \right)} \right]^3$$

这里过采样频率，OSR，是调制器频率 f_M与采样频率 f_S的比率。图 26-3给出了 32 的一个 OSR 滤波器频率响应。 第一个滤波器陷波是在 $f_S = F_M/OSR$ 时。 可以通过更改调制器的频率，f_M，使用 SD16SSELx 和 SD16DIVx，且过采样率用 SD16OSRx 和 SD16XOSR 位来调节陷波的频率。

在采样频率为 f_S时，每个使能的 ADC 通道的数字滤波器完成数字位流的采样，并输出新的转换结果到 SD16MEM0 寄存器。

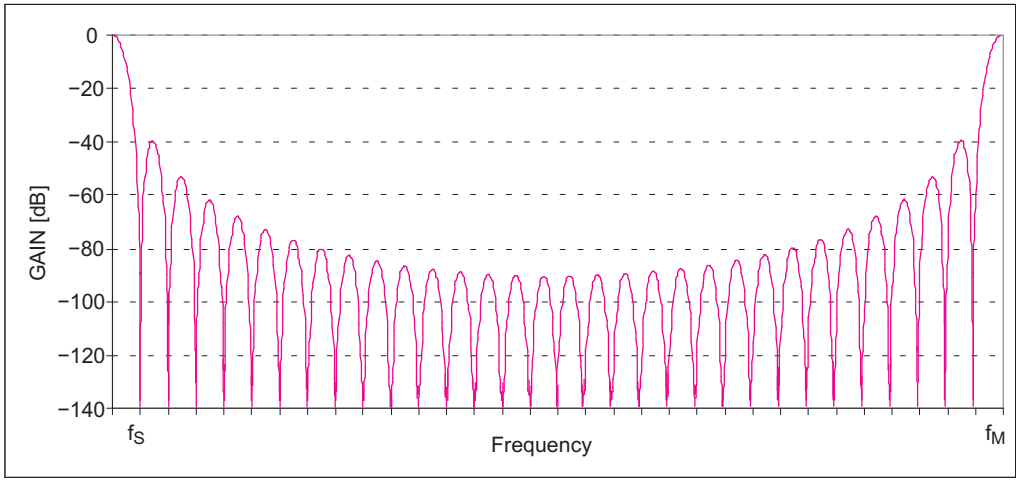


图 **26-3.** 梳状滤波器的频率响应，**OSR = 32**

图 26-4给出了数字滤波器的阶跃响应和转换点。 在开始转换后，对于在输入端的阶跃变化，必须在获得一个有效转换结果之前提供一段建立时间。 SD16INTDLYx 位可以为 ADC 输入的一个满量程变化提供足够的滤波建立时间。 如果阶跃和数字滤波器的采样同时进行，则有效数据会在第 3 次转换时可用。 一个异步阶跃会在有效数据可用之前需要多一次转换。

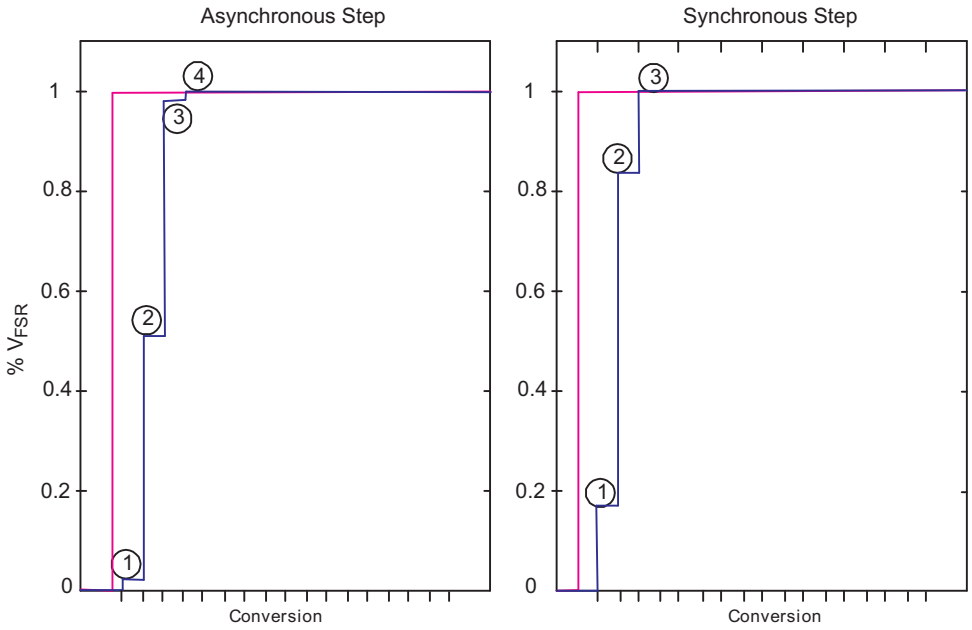


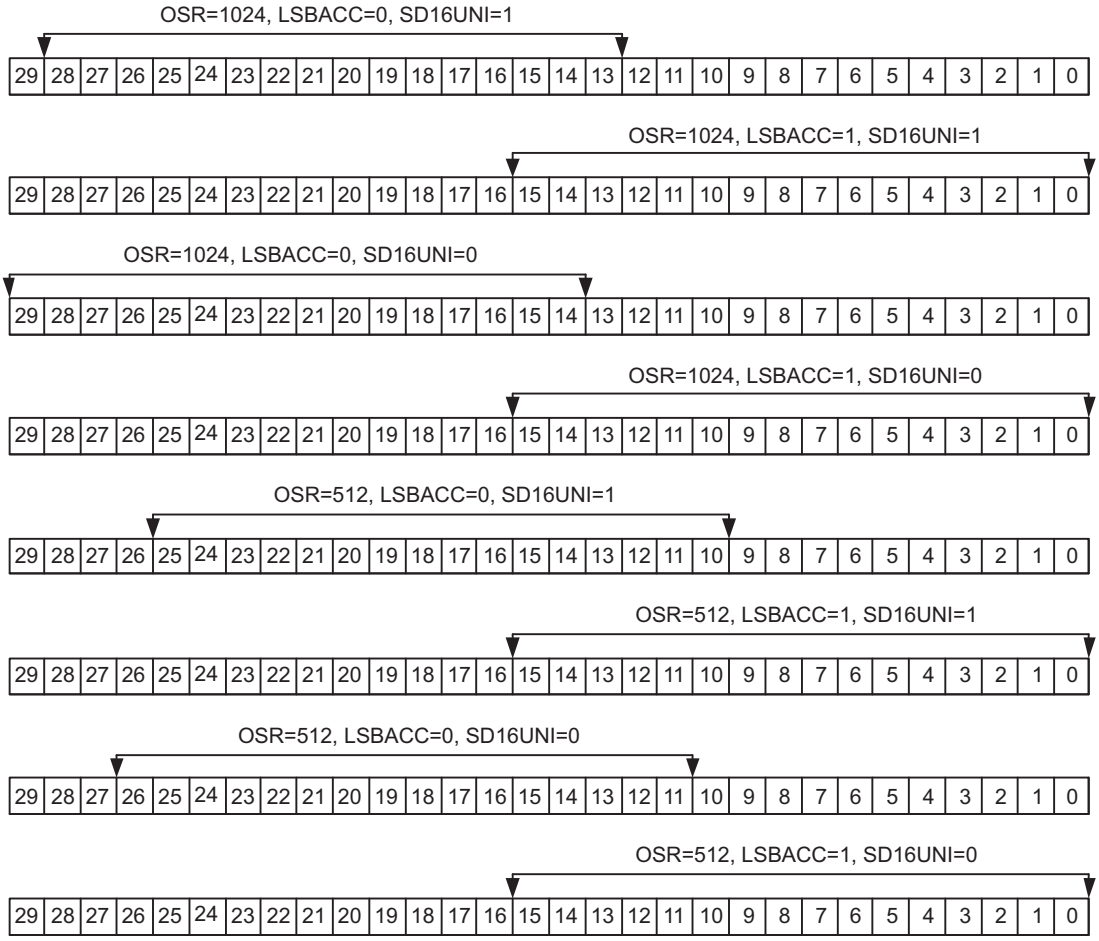
图 **26-4.** 数字滤波器阶跃响应和转换点

26.2.7.1 数字滤波器输出

数字滤波器输出的位数由过采样率决定，且范围为 15 至 30 位。图 26-5给出了数字滤波器输出和在每个 OSR，LSBACC, 和 SD16UNI 设置下，它们与 SD16MEM0 的关系。 例如，对于 OSR=1024，LSBACC = 0, 和 SD16UNI = 1，SD16MEM0 寄存器包含了数字滤波器输出的第 28 至 13 位。 当 OSR= 32 时, 1 个 (SD16UNI = 0) 或 2 个 (SD16UNI=1) LSB 总是为 0。

SD16LSBACC 和 SD16LSBTOG 位允许访问数字滤波器输出的最低有效位。 当 SD16LSBACC = 1 时，通过用字指令，数字滤波器的输出的最低有效位 16 位可以从 SD16MEM0 读出。 通过只返回数字滤波器输出的最低有效位 8 位，也可以通过字节指令访问SD16MEM0 寄存器。

当 SD16LSBTOG = 1 时，每次读取 SD16MEM0 时都会自动触发 SD16LSBACC 位。 这就使得数字滤波器的输出结果可以通过 2 次读取 SD16MEM0 来获得。 在下一次 SD16MEM0 访问前，置位或复位 SD16LSBTOG 并不会改变 SD16LSBACC。



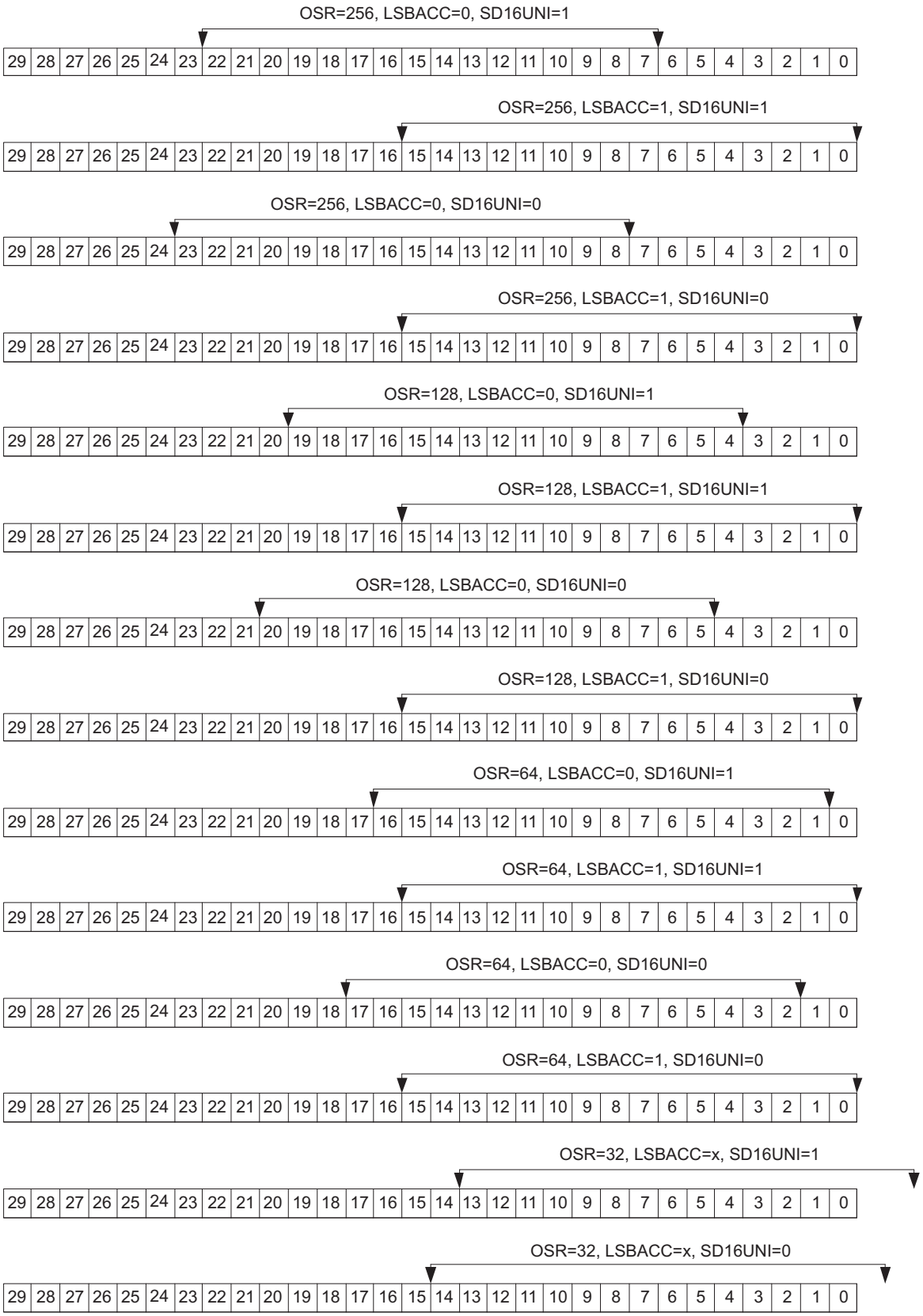


图 **26-5.** 数字滤波器输出的使用位

26.2.8 转换存储寄存器：SD16MEM0

SD16MEM0 寄存器和 SD16_A 通道相关联。 转换结果随着数字滤波器的每次采样阶跃被转移到 SD16MEM0 寄存器中。 当新数据被写入 SD16MEM0 时，SD16IFG 位就会被置位。 当 SD16MEM0 被 CPU 读取或被软件清除时，SD16IFG 会被自动清零。

26.2.8.1 输出数据格式

输出数据格式被配置为 2 的补码、偏移二进制或单极性模式，如在表 26-3中所示。 数据格式由 SD16DF 和 SD16UNI 位来选择。

表 **26-3.** 数据格式

SD16UNI	SD16DF	Format	模拟输入	SD16MEM0(1)	数字滤波器输出 (OSR = 256)
0	0	双极性偏移二进制	+FSR	FFFF	FFFFFF
			零	8000	800000
			-FSR	0000	000000
0	1	双极性二补码	+FSR	7FFF	7FFFFF
			零	0000	000000
			-FSR	8000	800000
1	0	单极性	+FSR	FFFF	FFFFFF
			零	0000	800000
			-FSR	0000	000000

(1) 不受 SD16OSRx 和 SD16XOSR 设置的影响；SD16LSBACC = 0。

注: 偏移测量和数据格式

只有当 SD16UNI= 0、通道在双极性模式下运行时，任何已完成的外部偏移测量或使用内部差分对 A7 才将是适当的。

图 26-6给出了范围为 $-V_{FRS}$至 $+V_{FRS}$的满量程输入电压和转换结果之间的关系。 已用图解形式给出了数据格式。

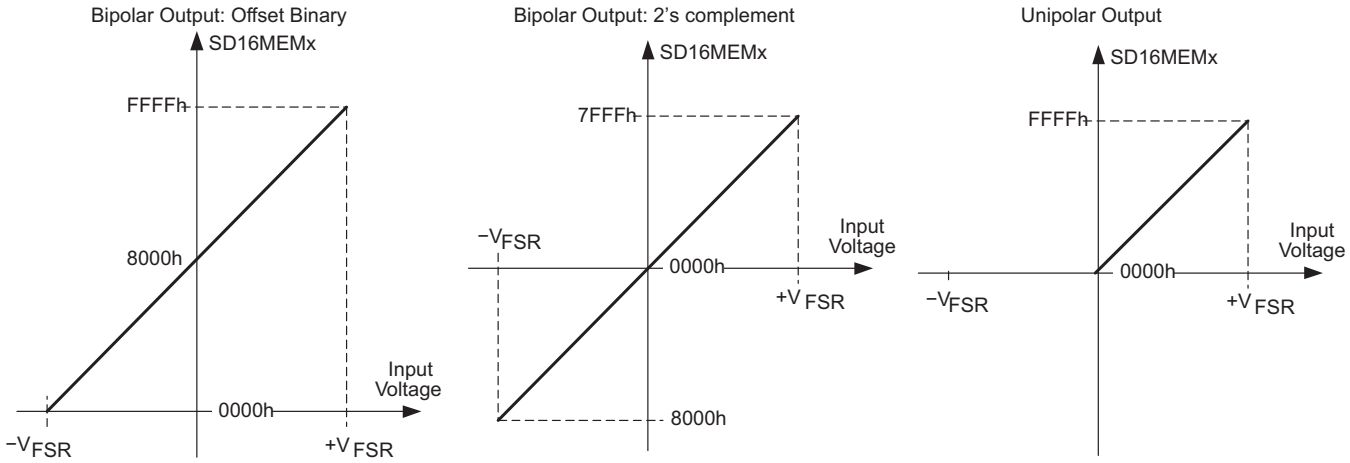


图 **26-6.** 输入电压与数字输出的关系

26.2.9 转换时间

SD16_A 模块可以被配置为两种操作模式，在表 26-4中列出了这两种模式。 SD16SNGL 位选择了转换模式。

表 26-4. 转换模式汇总

SD16SNGL	模式	运行
1	单次转换	该通道被转换一次。
0	连续转换	通道被连续转换。

26.2.9.1 单一转换

当 SD16SNGL = 1 时，置位该通道的 SD16SC 位会初始化该通道的一次转换。SD16SC 位在转换结束后会自动清零。

在转换完成之前清除 SD16SC 会立即停止该通道的转换，该通道会被断电并且相应的数字滤波器也会被关闭。 SD16MEM0 的值在 SD16SC 被清除时可能会变化。 因此建议在清除 SD16SC 前读取 SD16MEM0 值，以避免读到一个无效的结果。

26.2.9.2 连续转换

当 SD16SNGL = 0 时，连续转换模式将会被选择。 当 SD16SC 被置位时，该通道的转换开始，并连续转换到 SD16SC 位被软件清零。

清零 SD16SC 会立刻停止所选通道的转换，该通道会被断电，且相应的数字滤波器也会被关闭。 SD16MEM0 中的值在 SD16SC 被清零时可能会变化。 为了避免读到一个无效的结果，建议在清零 SD16SC 前读取 SD16MEM0 值。

图 26-7给出了转换操作。

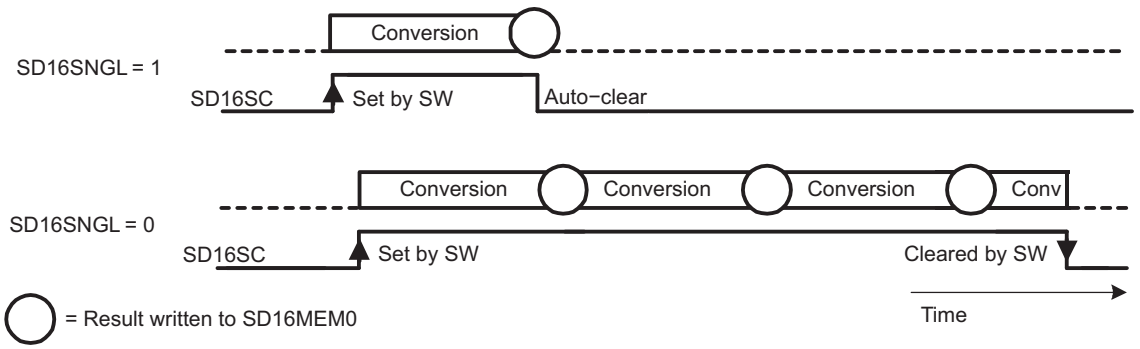


图 26-7. 单通道操作

26.2.10 使用集成的温度转换器。

为了使用片上温度传感器，用户应选择模拟输入对 SD16INCHx = 110 并且设置 SD16REFON = 1。任何其他的配置都和外部模拟输入对一样被选择，包括 SD16INTDLYx 和 SD16GAINx 的设置。 因为为了使用温度传感器，内部基准电压时必须打开，因此不能使用一个外部基准电压作为温度传感器电压的转换。 并且，内部基准电压会和任何使用的外部基准电压发生争用。 在这种情况下，为了最小化转换中争用的影响，可以设置 SD16VMIDON 位。

这种典型的温度传感器传递函数如图 26-8所示。 当把一个 SD16_A 通道的输入转换为温度传感器时，必须使用 SD16INTDLYx 提供适当的延时，以便允许数字滤波器建立和保证转换结果的正确性。 在大多数应用中，温度传感器误差偏移可能较大，但可以校准。 有关温度传感器的参数请参阅《器件专用数据表》。

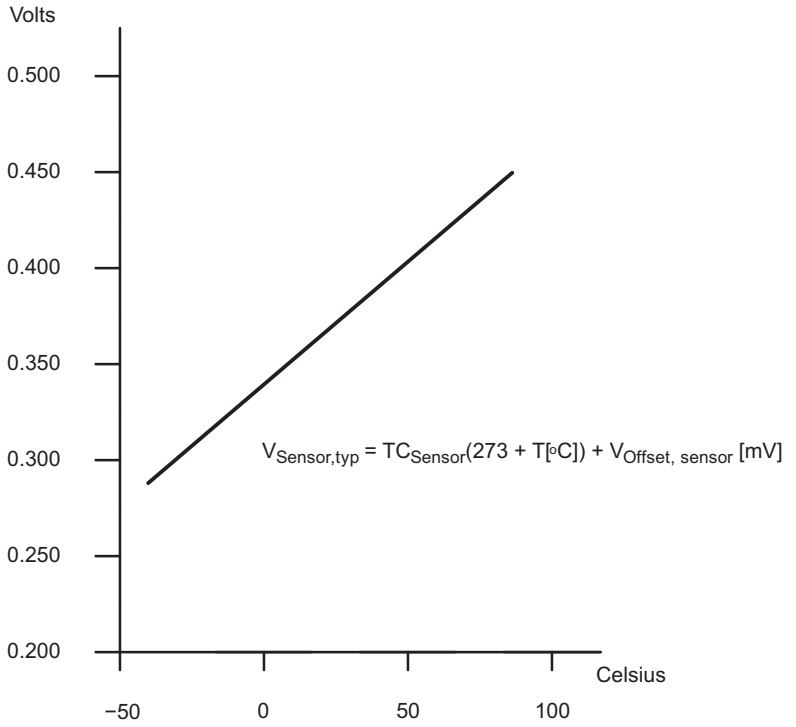


图 **26-8.** 典型的温度传感器传递函数

26.2.11 中断处理

它的 ADC 通道的 SD16_A 有 2 个中断源：

- SD16IFG
- SD16OVIFG

当 SD16MEM0 存储寄存器写入转换结果时，SD16IFG 位被设置。 如果相应的 SD16IE 位和 GIE 位都被设置，就会产生一个中断请求。 当在前一个结果被读取之前，新的转换结果写入 SD16MEM0 时，SD16_A 就会发生溢出状况。

26.2.11.1 SD16IV，中断向量发生器

所有的 SD16_A 中断源都被优先化，并被连接到一个中断向量上。 SD16IV 被用于确定哪一个使能的 SD16_A 中断源请求了一个中断。 被启用的最高优先级 SD16_A 的中断请求在 SD16IV 寄存器中产生一个数字（请参见寄存器描述）。 该数字可以被估计，或被添加到程序计数器相中以便自动进入相应的软件程序。 禁用 SD16-A 的中断不影响 SD16IV 的值。

任何对 SD16IV 寄存器的访问，读取或写入均不会影响到 SD16OVIFG 或 SD16IFG 的标志。 通过读取 SD16MEM0 寄存器或清零软件中的标志可以复位 SD16IFG 标志。 SD16OVIFG 位只能由软件复位。

如果在服务一个中断后另一个中断挂起，就会产生另外一个中断。 例如，当中断服务子程序访问 SD16IV 寄存器时，如果 SD16OVIFG 和一个或多个 SD16IFG 中断被挂起，会首先响应 SD16OVIFG 中断条件，并且相应的标志必须用软件清除。 在执行完中断子程序的 RETI 指令后，最高优先级 SD16IFG 的挂起会生成另一个中断请求。

26.2.11.2 中断延时操作

SD16INTDLYx 位控制为相应通道的第一次中断服务请求定时。 为了在产生一个中断请求之前允许数字滤波器建立，该功能把一次完整的转换中中断请求延迟了多达 4 个转换周期。 每次 SD16SC 被置位或当通道的 SD16GAINx 或 SD16INCHx 位被修改时就会发生延迟。 SD16INTDLYx 会为通道所选择的延时周期数禁止溢出中断的产生。 在延时期间，延时转换的中断请求不会产生中断。

26.3 SD16_A 寄存器

在表 26-5中列出了 SD16_A 寄存器。

表 26-5. SD16_A 寄存器

寄存器	简氏	寄存器类型	地址	初始化状态
SD16_A 的控制	SD16CTL	读取/写入	0100h	用 PUC 复位
SD16_A 中断向量	SD16IV	读取/写入	0110h	用 CPU 复位
SD16_A 通道 0 的控制	SD16CCTL0	读取/写入	0102h	用 PUC 复位
SD16_A 转换存储器	SD16MEM0	读取/写入	0112h	用 PUC 复位
SD16_A 的输入控制	SD16INCTL0	读取/写入	0B0h	用 PUC 复位
SD16_A 的模拟使能	SD16AE	读取/写入	0B7h	用 PUC 复位

26.3.1 SD16CTL，SD16_A 控制寄存器

15	14	13	12	11	10	9	8
被保留				SD16XDIVx			SD16LP
r0	r0	r0	r0	rw-0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
SD16DIVx		SD16SSELx		SD16VMIDON	SD16REFON	SD16OVIE	被保留
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	r0

被保留	位 15-12	被保留
SD16XDIVx	位 11-9	SD16_A 时钟分频器 000 /1 001 /3 010 /16 011 /48 1xx 被保留
SD16LP	位 8	低功耗模式。 该位选择了一个低速，低功耗模式。 0 低功耗模式被禁用 1 低功耗模式被使能。 SD16_A 的最大时钟频率被降低。
SD16DIVx	位 7-6	SD16_A 时钟分频器 00 /1 01 /2 10 /4 11 /8
SD16SSELx	位 5-4	SD16_A 时钟源选择 00 MCLK 01 SMCLK 10 ACLK 11 外部 TACLK
SD16VMIDON	位 3	VMID 缓冲打开 0 关闭 1 打开
SD16REFON	位 2	基准电压发生器打开 0 基准电压关闭 1 基准电压打开
SD16OVIE	位 1	SD16_A 溢出中断使能。 为了启用中断，必须把 GIE 位也置位。 0 溢出中断禁用 1 溢出中断被启用
被保留	位 0	被保留

26.3.2 SD16CCTL0, SD16_A 控制寄存器 0

15	14	13	12	11	10	9	8
被保留	SD16BUFx[1]		SD16UNI	SD16XOSR	SD16SNGL	SD16OSRx	
r0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
SD16LSBTOG	SD16LSBACC	SD16OVIFG	SD16DF	SD16IE	SD16IFG	SD16SC	被保留
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	r-0

被保留	位 15	被保留
SD16BUFx	位 14-13	高阻抗输入缓冲模式
		00 缓冲被禁止
		01 低速/电流
		10 中速/电流
		11 高速/电流
SD16UNI	位 12	单极性模式选择
		0 双极性模式
		1 单极性模式
SD16XOSR	位 11	已扩展的过采样率。 该位，与 SD16OSRx 位一起，选择过采样率。 有关设置请参阅《SD16OSRx 位说明》。
SD16SNGL	位 10	单次转换模式选择
		0 连续转换模式
		1 单次转换模式
SD16OSRx	位 9-8	过采样率
		当 SD16XOSR = 0 时
		00 256
		01 128
		10 64
		11 32
		当 SD16XOSR= 1 时
		00 512
		01 1024
		10 被保留
		11 被保留
SD16LSBTOG	位 7	LSB 触发。 该位，当置位时，每当读取 SD16MEM0 寄存器时都会引起 SD16LSBACC 触发。
		0 每当读取 SD16MEM0 时不会触发 SD16LSBACC
		1 每当读取 SD16MEM0 时会触发 SD16LSBACC
SD16LSBACC	位 6	LSB 访问。 该位允许访问高于或低于 SD16_A 转换结果的 16 位。
		0 SD16MEMx 包含转换的最高有效位 16 位。
		1 SD16MEMx 包含转换的最低有效位 16 位。
SD16OVIFG	位 5	SD16_A 溢出中断标志
		0 无溢出中断等待
		1 溢出中断等待
SD16DF	位 4	SD16_A 的数据格式
		0 偏移二进制
		1 二补码
SD16IE	位 3	SD16_A 的中断使能
		0 被禁用
		1 被启用
SD16IFG	位 2	SD16_A 的中断标志。 当新的转换结果可用时，可置位 SD16IFG 位。 当相应的 SD16MEMx 寄存器被读取或被软件清零时，SD16IFG 自动复位。
		0 无中断等待
		1 中断等待

(1) 被保留在 MSP430x20x3 器件中。

SD16SC	位 1	SD16_A 开始转换 0 无转换开始 1 转换开始
被保留	位 0	被保留

26.3.3 SD16INCTL0，SD16_A 输入控制寄存器

7	6	5	4	3	2	1	0
SD16INTDLYx		**SD16GAINx**			**SD16INCHx**		
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

SD16INTDLYx	位 7-6	转换开始后产生中断延迟。 在转换开始后，这些位选择第一次中断延迟。 00 第 4 次采样发生中断 01 第 3 次采样发生中断 10 第 2 次采样发生中断 11 第 1 次采样发生中断
SD16GAINx	位 5-3	SD16_A 前置放大器的增益 000 x1 001 x2 010 x4 011 x8 100 x16 101 x32 110 被保留 111 被保留
SD16INCHx	位 2-0	SD16_A 的通道差分对输入 000 A0 001 A1 010 A2 011 A3 100 A4 101 A5 - (AV_{CC}- AV_{SS}) / 11 110 A6 - 温度传感器 111 A7 - 短路 PGA 偏移测量

26.3.4 SD16MEM0，SD16_A 转换存储寄存器

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

转换结果 位 15-0 转换结果。 SD16MEMx 寄存器是否保持数字滤波器输出的高于或低于 16 位是由SD16LSBACC 位决定的。

26.3.5 SD16AE，SD16_A 模拟输入使能寄存器

7	6	5	4	3	2	1	0
SD16AE7	**SD16AE6**	**SD16AE5**	**SD16AE4**	**SD16AE3**	**SD16AE2**	**SD16AE1**	**SD16AE0**
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

SD16AEx 位 7-0 SD16_A 的模拟使能

0 禁用外部输入。 负极输入端被内连到 VSS。

1 使能外部输入。

26.3.6 SD16IV，SD16_A 中断向量寄存器

15	14	13	12	11	10	9	8
0	0	0	0	0	0	0	0
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
0	0	0	**SD16IVx**				0
r0	r0	r0					r0

SD16IVx 位 15-0 SD16_A 的中断向量值

SD16IV 的目录	中断源	中断标志	中断优先级
000h	无中断等待	-	
002h	SD16MEMx 溢出	SD16CCTLx SD16OVIFG	最高
004h	SD16_A 的中断	SD16CCTL0 SD16IFG	
006h	被保留	-	
008h	被保留	-	
00Ah	被保留	-	
00Ch	被保留	-	
00Eh	被保留	-	
010h	被保留	-	最低

SD24_A

SD24_A 模块是一个多通道 24 位积分三角模数转换器 (ADC)。 本章介绍了 MSP430x2xx 系列中的 SD24_A。

Topic **Page**

27.1 SD24_A 介绍

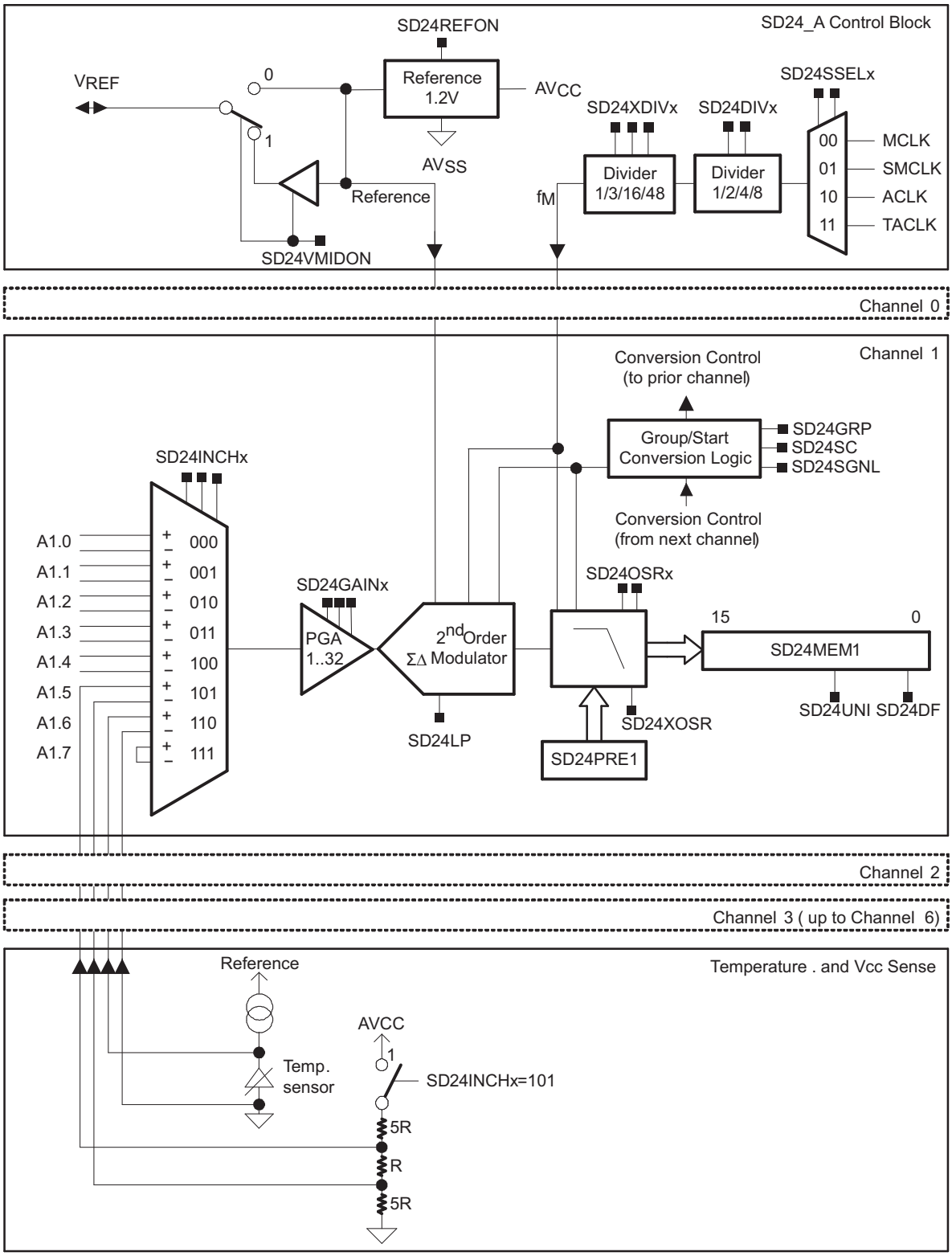
SD24_A 模块由多达七个独立的积分三角模数转换器和一个内部基准电压组成，其中模数转换器简称为通道。 每个通道都具有多达 8 个全差分多路复用模拟输入对，该模拟输入对包括一个内置的温度传感器和一个分割电源电压。 该转换器是基于二阶过采样 Σ-Δ 调制器和数字抽取滤波器的。 抽取滤波器是带有可选的过采样率高达 1024 的梳型过滤器。 额外的滤波电路可在软件中完成。

基于过采样率，SD24_A 的数字滤波器输出范围可以从 15 位达到 30 位。 默认的过采样率是 256，这将导致从数字滤波器中输出 24 位。 可在 SD24_A 转换存储寄存器中捕获滤波器的 16 个最高有效位，并通过设置 SD24LSBACC= 1，可以读取滤波器输出的 16 个最低有效位（更多细节请参阅27.2.7 节）。

SD24_A 的特性包括：

- 多达 7 个独立的、同步采样 ADC 通道（通道数与设备有关，请参阅《特定器件的数据手册》）。
- 多达 8 个多路复用、每通道差分模拟输入端（输入端的数量与器件有关，请参阅特定器件的数据手册）。
- 可选的片上软件参考电压产生 (1.2 V)
- 可用软件的内部或外部参考
- 内置所有渠道都可用的温度传感器
- 高达 1.1MHz 的调制器输入频率
- 高阻抗输入缓冲器（未实现在所有器件上实施，请参阅《特定器件的数据手册》）。
- 可选低功率转换模式

在图 27-1中展示了 SD24_A 模块框图。



NOTE: Ax.1 到 Ax.4 并不适用于所有设备。 请参阅特定器件的数据手册。

图 27-1. SD24_A 模块框图

27.2 SD24_A 的操作

SD24_A 模块被配置为带有用户的软件。 在下面的章节中讨论了 SD24_A 的设置和操作。

27.2.1 ADC 芯片

模数转换是由一个 1 位二阶三角积分调制器执行的。 调制器内的一个单个位比较器用调制器的频率 f_M量化了输入信号。 为了得出转换结果，所得的 1 位数据流被数字滤波器平均。

27.2.2 模拟输入范围和可编程增益放大器 (PGA)

每个模拟量输入对的满量程输入电压范围取决于每个通道的可编程增益放大器的增益设置。 最大的满量程范围为 $\pm V_{FSR}$，其中 V_{FSR}被以下内容所定义：

$$V_{FSR} = \frac{V_{REF}/2}{GAIN_{PGA}}$$

对于一个 1.2V 的基准，一个增益为 1 的最大满量程输入范围是：

$$\pm V_{FSR} = \frac{1.2\ V/2}{1} = \pm 0.6\ V$$

请参阅《特定器件的满量程输入规格数据手册》。

27.2.3 基准电压发电机

SD24_A 模块具有一个内置的 1.2V 基准。 它可以用于每个 SD24_A 通道且可由 SD24REFON 位使能。 当使用内部基准电压时，为了减少噪音，建议把一个外部 100nF 电容从 V连接到 AV_{SS}。 当 SD24VMIDON= 1 时，内部基准电压可用于片外。缓冲输出可提供高达 1mA 的驱动。 当使用内部片外基准电压，需要把一个 470nF 电容从 V_{REF}连接到 AV_{SS}。 请参阅《特定器件的数据手册的参数》。

当 SD24REFON 和 SD24VMIDON 都被复位时，可把外部参考电压应用到 V_{REF}输入。

27.2.4 自动断电

设计 SD24_A 专用于低功耗应用。 当 SD24_A 不进行转换时，它会自动禁用，当换开始时，它会自动重新启用。 该基准不会自动禁用，但可以通过设置 SD24REFON= 0 来禁用它。SD24_A 或基准被禁用时，不消耗电流。

27.2.5 模拟输入对的选择

SD24_A 可以将最多 8 个差分对输入复用到 PGA 中。 在器件上可对外使用多达 5 个模拟输入对 (A0∽A4)。 可通过 A5 多路调制器输入来使用一个测量供电电压的电阻分压器。 可通过 A6 多路复用器输入来使用内部温度传感器。 输入端 A7 是一个 + 和 - 输入对之间的短路连接，且可以用于校准 SD24_A 输入级的偏移量。

27.2.5.1 模拟输入设置

使用 SD24INCTLx 寄存器来配置每个通道的模拟输入端。 可以为每个SD24_A通道独立地配置这些设置。

SD24INCHx 位选择了模拟多路复用器的 8 个差分输入对的其中之一。 由 SD24GAINx 位为每个 PGA 选择增益。 共有 6 个增益设置可用。

在某些设备上 SD24AEx 位可启用或禁用模拟输入引脚。 设置任何 SD24AEx 位都会禁用关联引脚的复用数字电路。 有关引脚图请参阅《特定器件数据手册》。

在转换过程中，对 SD24INCHx 和 SD24GAINx 位的任何修改都将使下一个数字滤波器的抽取步骤变得有效。 这些位被修改之后，由于数字滤波器的建立时间，接下来的三个转换可能是无效的。 这可以用 SD24INTDLYx 位进行自动处理。 当 SD24INTDLY=00H 时，转换中断请求直到一个起始条件后的第四转换才会开始。

可通过使用 SD24BUFx 位来使能在器件上实施的高阻抗输入缓冲。 如在表 27-1中所示，基于 SD24_A 调制器频率的速度设置是可选的。

表 27-1. 高输入阻抗缓冲器

SD24BUFx	缓冲器	SD24 调制器频率，f_M
00	缓冲区禁用	
01	低速/电流	f_M< 200kHz
10	中速/电流	200kHz < f_M< 700kHz
11	高速/电流	700kHz < f_M< 1.1MHz

为了防止输入信号混叠，建议为 SD24_A 安装一个外部 RC 抗混叠滤波器。 1MHz 的调制器时钟的截止频率应该小于 10kHz，且 OSR = 256。 截止频率可能会调到有较低带宽需求的应用程序的一个较低频率。

27.2.6 模拟输入特性

SD24_A 使用了一个开关电容器输入级，该输入极被作为一个显示到外部电路的阻抗，如图 27-2所示。

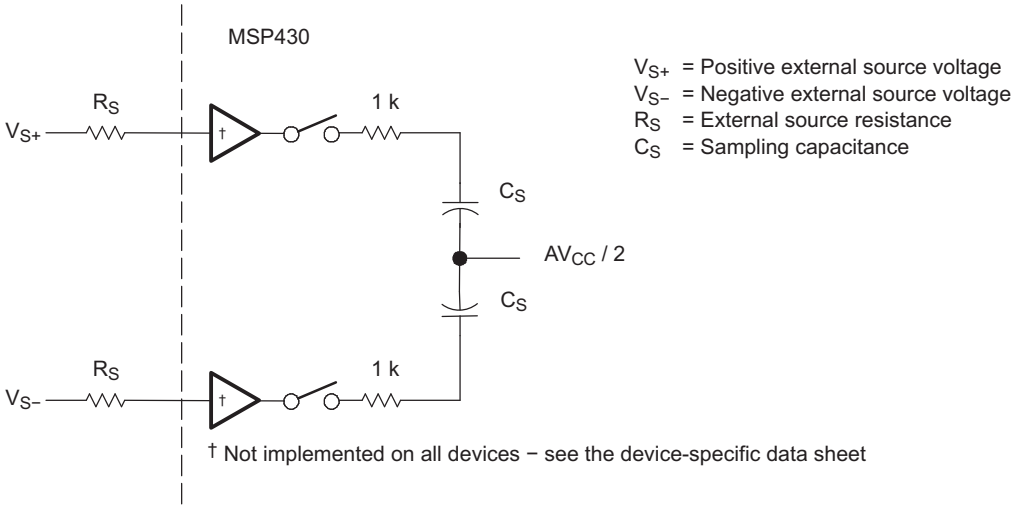


图 27-2. 模拟输入等效电路

当使用缓冲器时，R_S不影响采样频率 f_S。 然而，当不使用缓冲器或在设备上没有缓冲器时，最大调制器频率 f_M可以由采样电路的最小建立时间 $t_{建立时间}$根据以下公式计算：

$$t_{Settling} \geq (R_S + 1\ k\Omega) \times C_S \times \ln\left(\frac{GAIN \times 2^{17} \times V_{Ax}}{V_{REF}}\right)$$

其中，

$$f_M = \frac{1}{2 \times t_{Settling}} \quad \text{and} \quad V_{Ax} = \max\left(\left|\frac{AV_{CC}}{2} - V_{S+}\right|, \left|\frac{AV_{CC}}{2} - V_{S-}\right|\right)$$

用 V_{S+}和 V_{S-}参考 AV_{SS}。

如在表 27-2中所示，C_S随着增益设置变化。

表 **27-2.** 采样电容

PGA 增益	采样电容 (C_S)
1	1.25pF
2, 4	2.5pF
8	5pF
16, 32	10pF

27.2.7 数字滤波器

通过使用一个 SINC^{3}梳状滤波器，数字滤波器处理来自调制器的 1 位数据流。 在 z 域描述传递函数是根据：

$$H(z) = \left(\frac{1}{OSR} \times \frac{1 - z^{-OSR}}{1 - z^{-1}}\right)^3$$

和在频域中的是根据：

$$H(f) = \left[\frac{\text{sinc}\left(OSR \times \pi \times \frac{f}{f_M}\right)}{\text{sinc}\left(\pi \times \frac{f}{f_M}\right)}\right]^3 = \left[\frac{1}{OSR} \times \frac{\sin\left(OSR \times \pi \times \frac{f}{f_M}\right)}{\sin\left(\pi \times \frac{f}{f_M}\right)}\right]^3$$

其中过采样率，OSR，是调制器频率f_M与采样频率 f_S的比率。图 27-3为 32 的 OSR 显示了滤波器的频率响应。 第一个滤波器陷波是在 $f_S = F_M/OSR$。 可以通过改变调制器的频率，使用 SD24SSELx 和 SD24DIVx 以及使用 SD24OSRx 和 SD24XOSR 位的过采样率来调整陷波频率，f_M。

每个启用的 ADC 通道的数字滤波器完成了抽取的数字位流，并在采样频率为 f_S时把新的转换结果输入到 相应的 SD24MEMx 寄存器中。

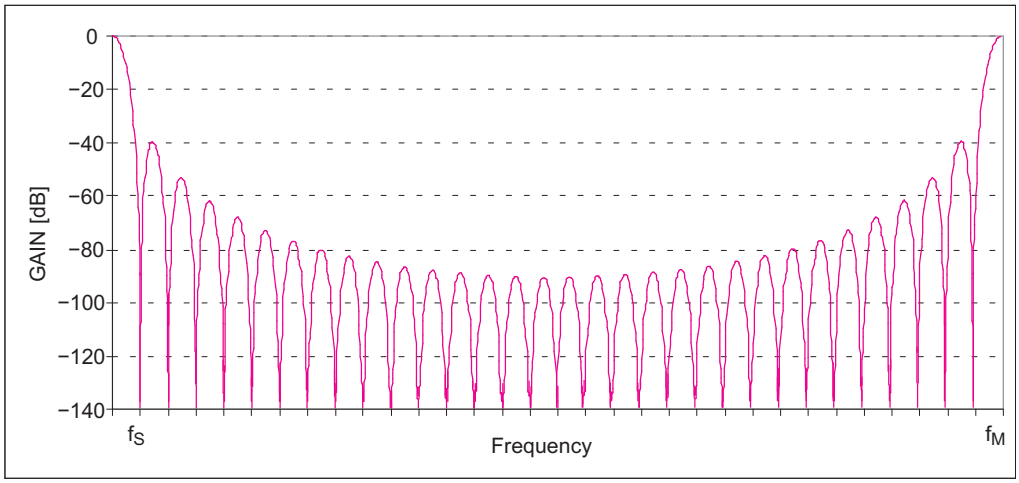


图 **27-3. OSR= 32** 的梳状滤波器的频率响应

图 27-4给出了数字滤波器的阶跃响应及转换点。 对于转换开始后的在输入端的阶跃变化，在一个有效的转换结果可用之前，必须允许一个建立时间。 SD24INTDLYx 位可以为在 ADC 输入端的满量程变化提供足够的滤波器建立时间。 如果该步骤发生与数字滤波器的抽取同步，则有效数据将在第三个转换上可用。 在有效数据可用之前，异步步骤将需要一个额外的转换。

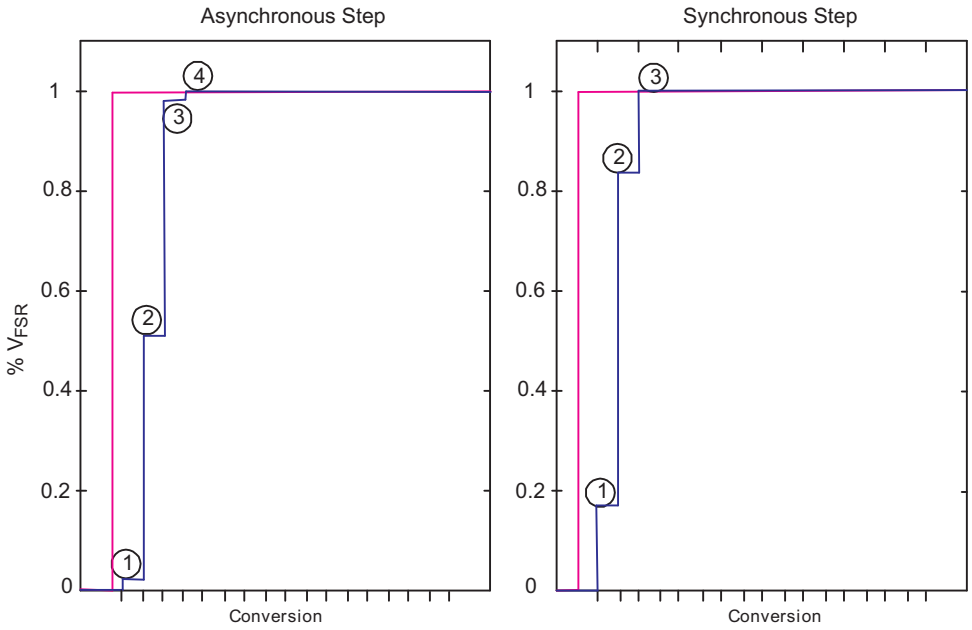


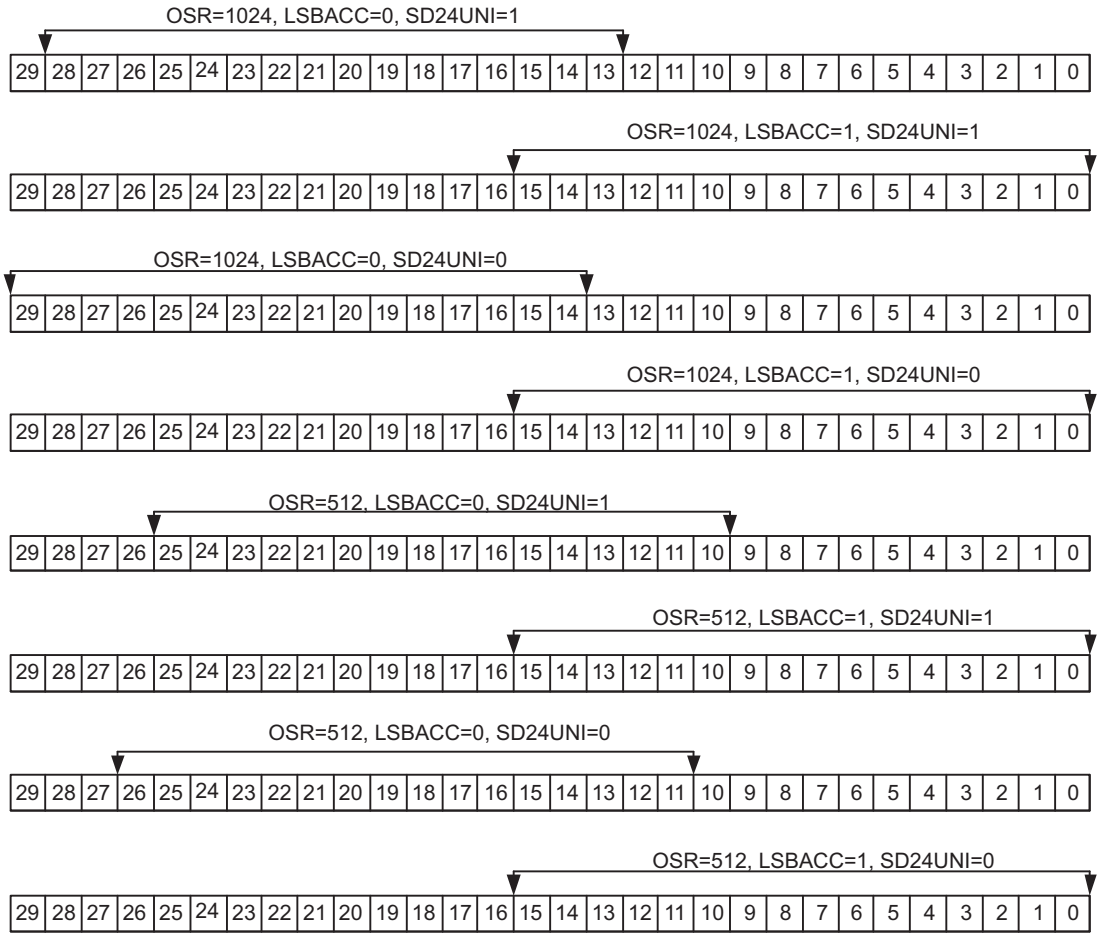
图 **27-4.** 数字滤波器的阶跃响应和转换点

27.2.7.1 数字滤波器输出

由数字滤波器输出的位的数目取决于过采样率和从 15 到 30 位的范围。图 27-5为每个 OSR，LSBACC，和 SD24UNI 设置给出了数字滤波器输出以及滤波器与 SD24MEMx 的关系。 例如，对于 OSR= 1024，LSBACC= 0，和 SD24UNI= 1，SD24MEMx 寄存器包含数字滤波器输出的位 28 至位 13。 当 OSR =32 时，一个 (SD24UNI= 0) 或 2 个 LSB (SD24UNI= 1) 始终为零。

SD24LSBACC 和 SD24LSBTOG 位给出了数字滤波器输出的最少有效位的访问权限。 当 SD24LSBACC=1 时，通过使用字指令，数字滤波器的输出端的 16 个最低有效位可被 SD24MEMx 读取。 通过只返回数字滤波器输出的 8 个最低有效位，可以用字节指令访问 SD24MEMx 寄存器。

当 SD24LSBTOG= 1 时，每次读取 SD24MEMx 都会自动切换 SD24LSBACC 位。 这就使得完整的数字滤波器的输出结果被 SD24MEMx 的两次读取所读取。 直到下一个 SD24MEMx 开始访问，设置或清零 SD24LSBTOG 才会改变 SD24LSBACC。



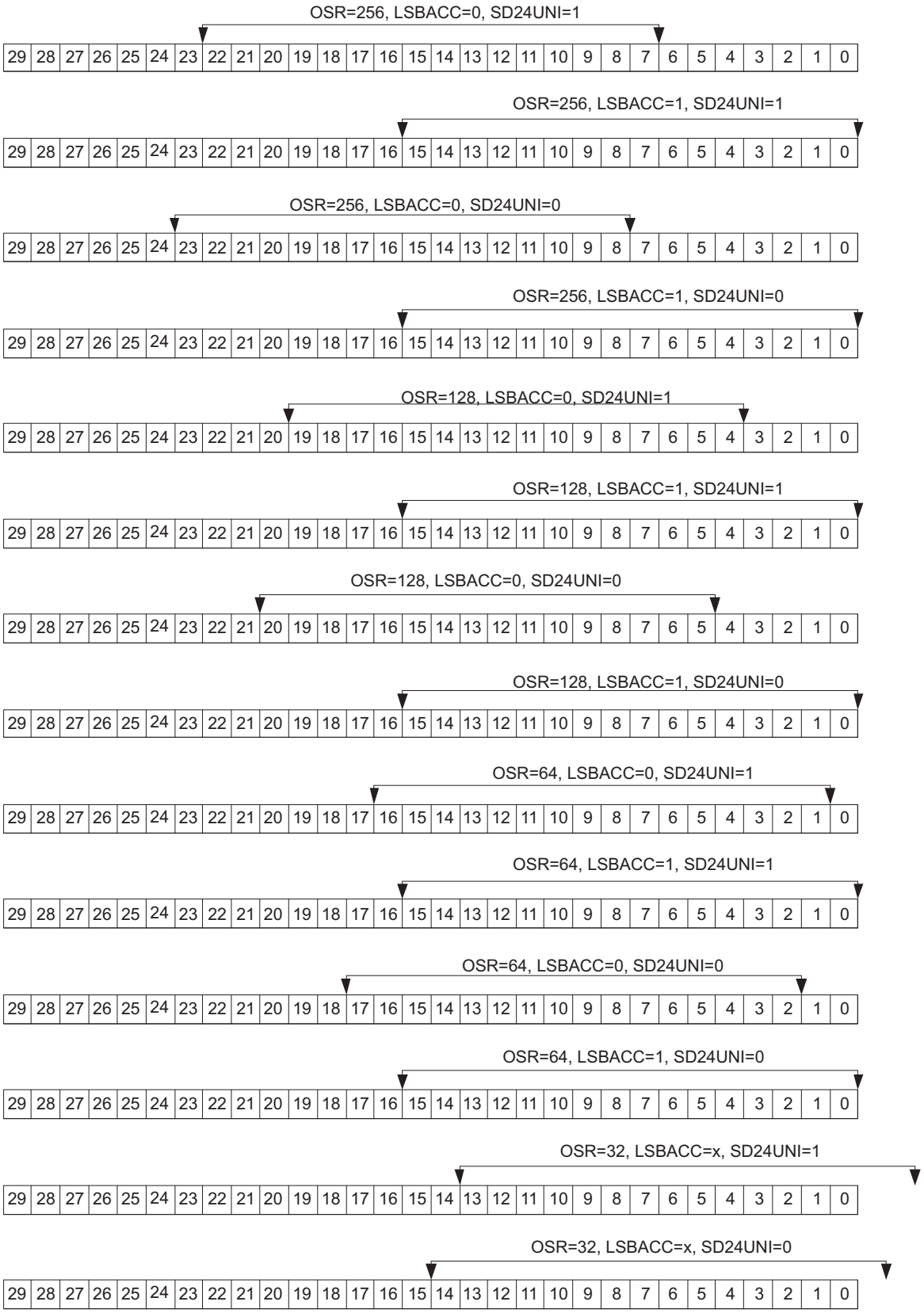


图 **27-5.** 已使用位的数字滤波器输出

27.2.8 转换存储寄存器：SD24MEMx

一个 SD24MEMx 寄存器与每个 SD24_A 通道是相关联的。 转换结果被移动到相应的带有数字滤波器每个抽取步骤的 SD24MEMx 寄存器中。 当新的数据被写入 SD24MEMx 时，SD24IFG 位被置位。 当 SD24MEMx 被 CPU 读取或可能被软件清除时，SD24IFG 被自动清零。

27.2.8.1 输出数据格式

如在表 27-3中所示，在二进制补码，偏移二进制或单极模式中输出数据格式是可配置的。 由 SD24DF 和 SD24UNI 位选择数据格式。

表 27-3. 数据格式

SD24UNI	SD24DF	Format	模拟输入	SD24MEMx (1)	数字滤波器输出 (OSR = 256)
0	0	双极性偏移二进制	+FSR	起始值也可以是 0，但是由于 SysTick 中断和 COUNTFLAG 在计数从 1 到 0 时都会被激活，所以没什么作用	FFFFFF
			零	8000	800000
			-FSR	0000	000000
0	1	双极性二进制补码	+FSR	7FFF	7FFFFF
			零	0000	000000
			-FSR	8000	800000
1	0	单极性	+FSR	起始值也可以是 0，但是由于 SysTick 中断和 COUNTFLAG 在计数从 1 到 0 时都会被激活，所以没什么作用	FFFFFF
			零	0000	800000
			-FSR	0000	000000

(1) SD24OSRx 和 SD24XOSR 设置的独立；SD24LSBACC= 0。

注： 偏置测量和数据格式

只有当 SD24UNI= 0，通道在双极性模式下运行时，任何已做的外部偏移测量或使用内部差分 A7 才是适当的。

如果要在单极化模式中把测得的值是用于偏移校正，那么需要把该值乘以 2。

图 27-6给出了从 $-V_{FSR}$到 $+V_{FRS}$ 的满量程输入电压范围与转换结果之间的关系。 给出了数据格式。

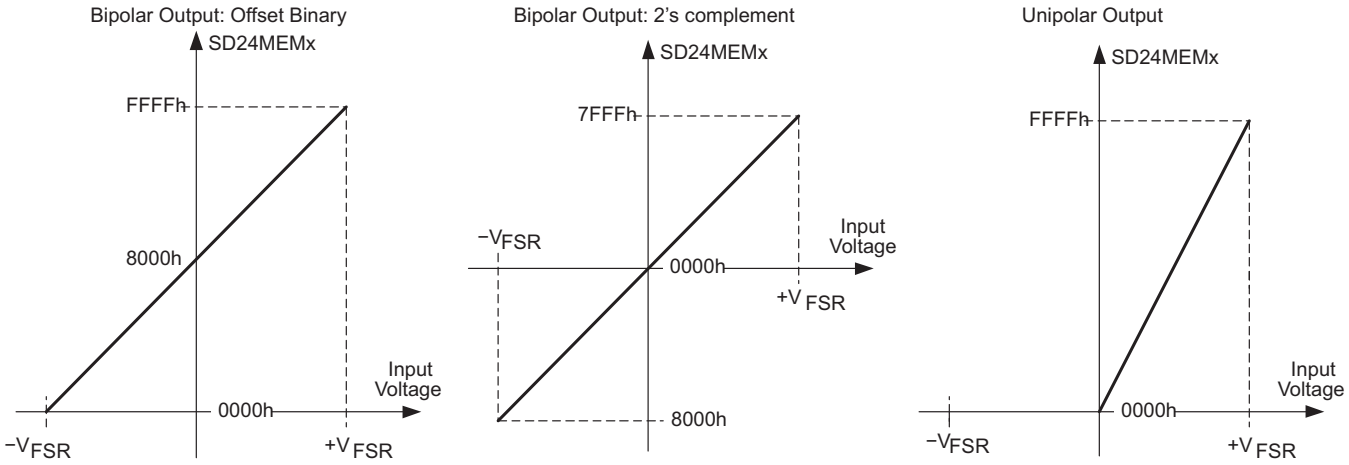


图 **27-6.** 输入电压与数字输出

27.2.9 转换时间

可以为四种操作模式配置 SD24_A 模块，在表 27-4中列出了这四种操作模式。 SD24SNGL 和 SD24GRP 位为每个通道选择了转换模式。

表 **27-4.** 转换模式总结

SD24SNGL	SD24GRP (1)	模式	运行
1	0	单通道，单次转换	一个单通道被转换一次。
0	0	单通道，连续转换	一个单通道被连续转换。
1	1	通道组，单次转换	一组通道被转换一次。
0	1	通道组，连续转换	一组通道被连续转换。

(1) 当 SD24GRP=0 时，如果前一通道 (S) 的 SD24GRP 被置位，一个通道被分组且是该组的主通道。

27.2.9.1 单通道，单次转换

当 SD24SNGL= 1，且它不与任何其它通道分组时，在该通道上设置 SD24SC 位通道会起始一个转换。 在转换完成后 SD24SC 位自动清零。

在转换完成之前清零 SD24SC 会立即停止所选通道的转换，通道被断电，且相应的数字滤波器被关闭。 在 SD24SC 被清零时，可以改变 SD24MEMx 中的值。 为了避免读取无效结果，建议在清除 SD24SC 之前读出 SD24MEMx 转换数据。

27.2.9.2 单通道，连续转换

当 SD24SNGL= 0 时，连续转换模式被选中。 选定通道的转换将在 SD24SC 被置位时开始，并一直持续到通道不与任何其他通道分组，SD24SC 位被软件清零。

清零 SD24SC 会立即停止所选通道的转换，通道被断电以及相应的数字滤波器被关闭。 在 SD24SC 被清零时，可以改变 SD24MEMx 中的值。 为了避免读取无效结果，建议在清除 SD24SC 之前读出 SD24MEMx 转换数据。

图 27-7给出了单次转换模式和连续转换模式中单信道的操作。

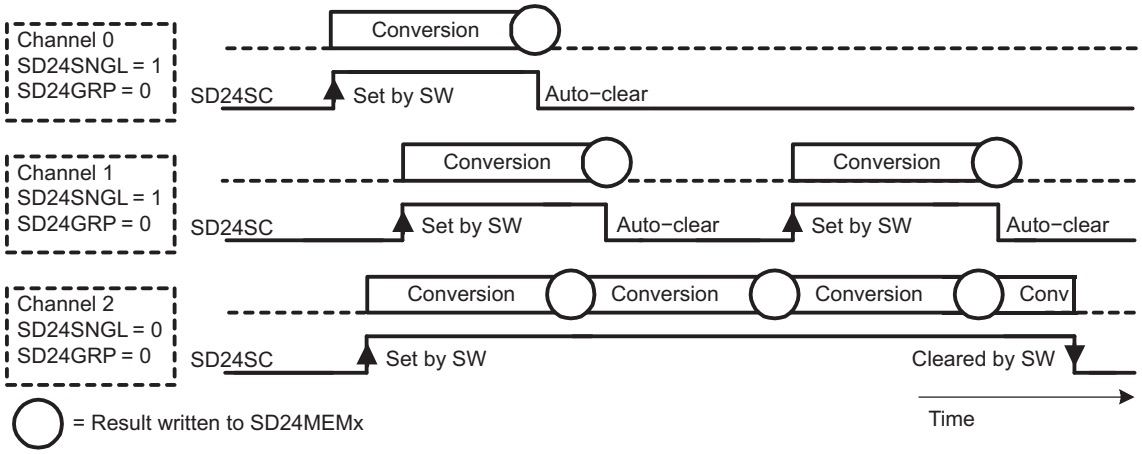


图 **27-7.** 单通道操作 **-** 示例

27.2.9.3 通道组，单次转换

连续的 SD24_A 通道可以用 SD24GRP 位组合在一起组成同步转换。 为在模块中带有下一个通道的一个通道组设置 SD24GRP。 例如， 为通道 0 组，即带有通道 1 的通道，设置 SD24GRP。在这种情况下，通道 1 是主通道，能用其 SD24SC 位启用和禁用组中所有通道的转换。 主通道的 SD24GRP 位始终为 0。SD24_A 的最后一个通道的 SD24GRP 位没有任何功能，并且始终为 0。

当在一组中的通道的 SD24SNGL=1 时，就会选择单次转换模式。 主通道 SD24SC 位被置位时，该通道的单次转换将同步发生。 该组中所有通道的 SD24SC 位将被自动设置，且被主通道的 SD24SC 清除。 每个通道的 SD24SC 也可以在软件中被独立清零。

在转换完成之前清零主通道的 SD24SC 会立即停止该组中所有通道的转换，通道被断电，且相应的数字过滤器被关闭。 在 SD24SC 被清零时，可以改变 SD24MEMx 中的值。 为了避免读取无效结果，建议在清除 SD24SC 之前读出 SD24MEMx 转换数据。

27.2.9.4 通道组，连续转换

当在一组中的通道的 SD24SNGL= 1 时，就会选择连续转换模式。 主通道 SD24SC 位被置位时，该通道的连续转换将同步发生。 主通道的 SD24SC 位将自动设置和清零所有集合通道的 SD24SC 位。 在该组中的每个通道的 SD24SC 也可以在软件中被独立清零。

当一个集合通道的 SD24SC 被主机的独立软件置位时，该通道的转换会自动与主通道的转换同步。 这就确保了集合通道的转换总是与主通道的转换同步。

清零主通道的 SD24SC 会立即停止该组中所有通道的转换，通道被断电，且相应的数字过滤器被关闭。 在 SD24SC 被清零时，可以改变 SD24MEMx 中的值。 为了避免读取无效结果，建议在清除 SD24SC 之前读出 SD24MEMx 转换数据。

图 27-8给出了 3 个 SD24_A 通道的集合通道的操作。 通道 0 被配置为单次转换模式，SD24SNGL= 1，且通道 1 和 2 被配置为连续转换模式，SD24SNGL= 0。通道 2，该组中的最后一个通道，是主通道。 不管是否用软件置位每个 SD24SC 位，该组中所有通道的转换与主通道的转换同步发生。

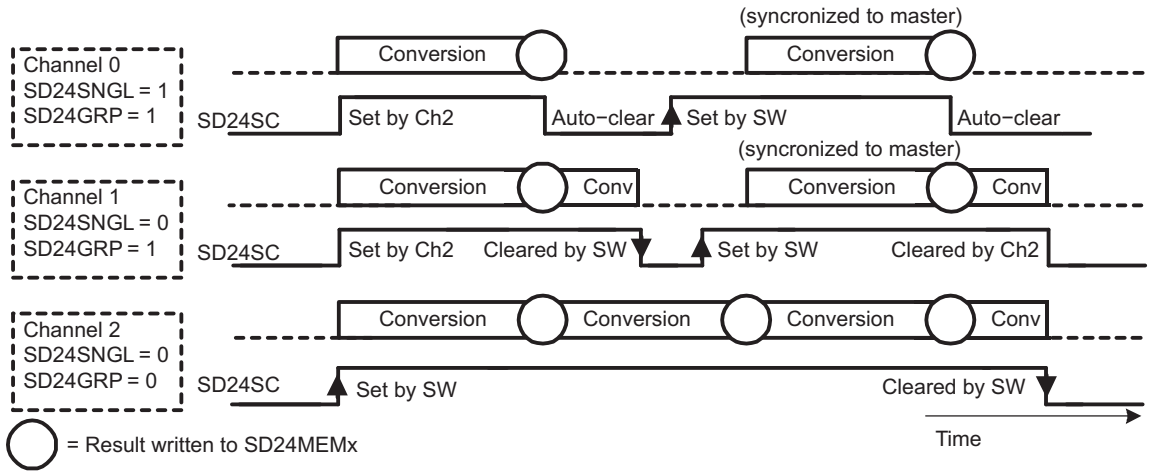


图 **27-8.** 集合通道操作 **-** 示例

27.2.10 使用预置的转换操作

当多个通道被集合时，SD24PREx 寄存器可以被用来延迟每个通道的转换时间帧。 使用 SD24PREx，f_M时钟周期的指定数目增加了数字滤波器的抽取时间，且范围可为 0 到 255。图 27-9给出了使用 SD24PREx 的一个例子。

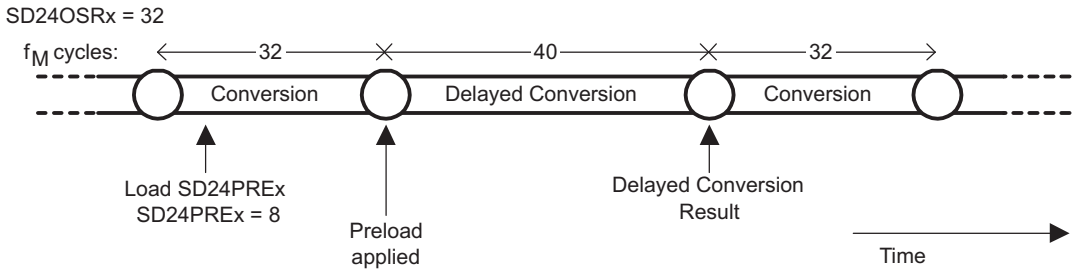


图 **27-9.** 转换延迟预置**-** 示例

SD24PREx 延迟被应用到正在写入后的下一个转换周期的开始。 在 SD24SC 被置位后的第一个转换上和紧跟着每一个写入 SD24PREx 的转换周期上使用该延迟。 下列转换不被延迟。 修改 SD24PREx 后，直到下一个转换周期完成才应发生下一个写入 SD24PREx，否则转换的结果可能不正确。

使用 SD24PREx 的已延迟转换周期结果的准确性取决于延迟的长度和正被采样的模拟信号的频率。 例如，当测量直流信号时，SD24PREx 延迟对转换结果没有影响，与持续时间无关。 用户必须确定延迟的转换结果何时对他们的应用程序是有用的。

图 27-10给出了集合通道 0 和 1 的操作。通道 1 的预置寄存器被零加载，这会致使立即转换，而通过设置SD24PRE0= 8，通道 0 的转换周期被延迟。 第一个通道 0 的转换使用 SD24PREx=8，用 8 个 f_M时钟周期移位所有后续转换。

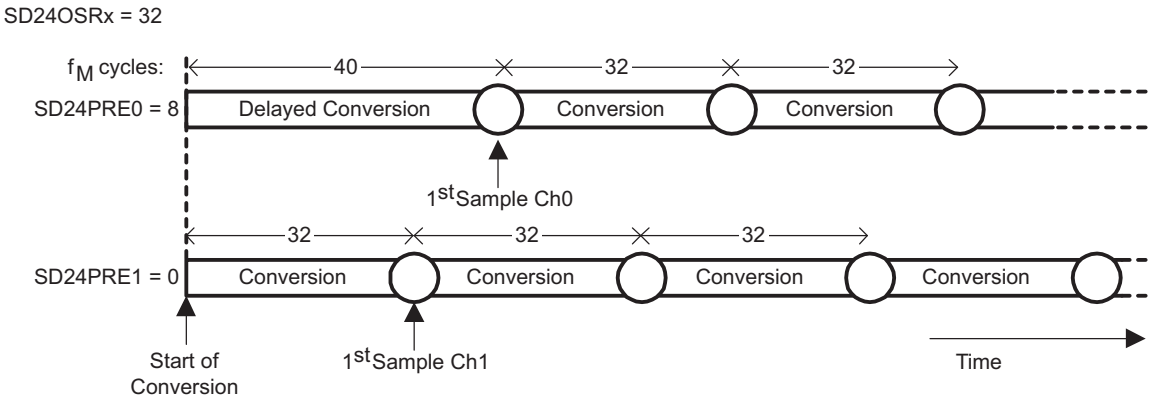


图 **27-10.** 使用预置的转换的开始 **-** 示例

当通道进行集合时，当一个或多个通道在单次转换模式中运行或在软件中被禁用、而主渠道仍然活跃时必须小心。 每次该组中的通道被重新启用，并重新与主通道同步时，该通道的预置延迟将被重新引入。图 27-11给出了在一个组中的通道重新同步和预置延迟。 当它们被重新启用时，为了在主通道和其他通道之间维持一个一致延迟，建议主通道的 SD24PREx= 0。

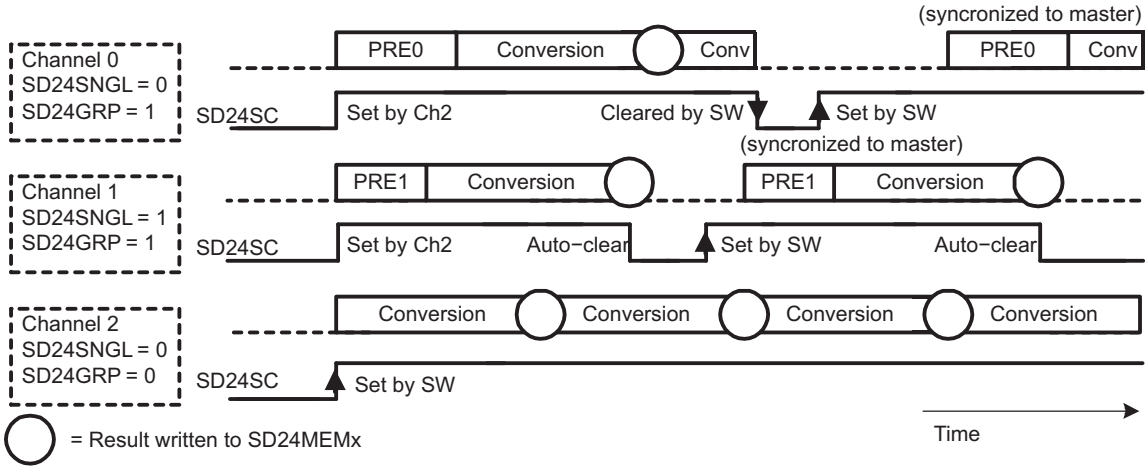


图 **27-11.** 预至和通道同步

27.2.11 使用集成温度传感器

为了使用的片上温度传感器，用户选择了的模拟输入对 SD24INCHx= 110，并设置 SD24REFON= 1。任何其他配置完成后，就如外部模拟输入对被选中，包括 SD24INTDLYx 和 SD24GAINx 设置。 由于内部参考必须在温度传感器上使用，因此是不能用做温度传感器电压转换的一个外部基准电压的。 此外，内部基准电压将争用任何使用的外部基准电压。 在这种情况下，为了最小化争用转换的影响，可以 SD24VMIDON 位被设置。

图 27-12给出了典型的温度传感器的传递函数。 当把一个 SD24_A 通道输入切换到温度传感器时，为了能使数字滤波器建立和保证转换结果是有效的，必须使用 SD24INTDLYx 提供足够的延迟。 对于大多数应用来说，该温度传感器的偏移误差可以有点大，并可能需要校准。 有关温度传感器参数请参阅《特定器件的数据手册的》。

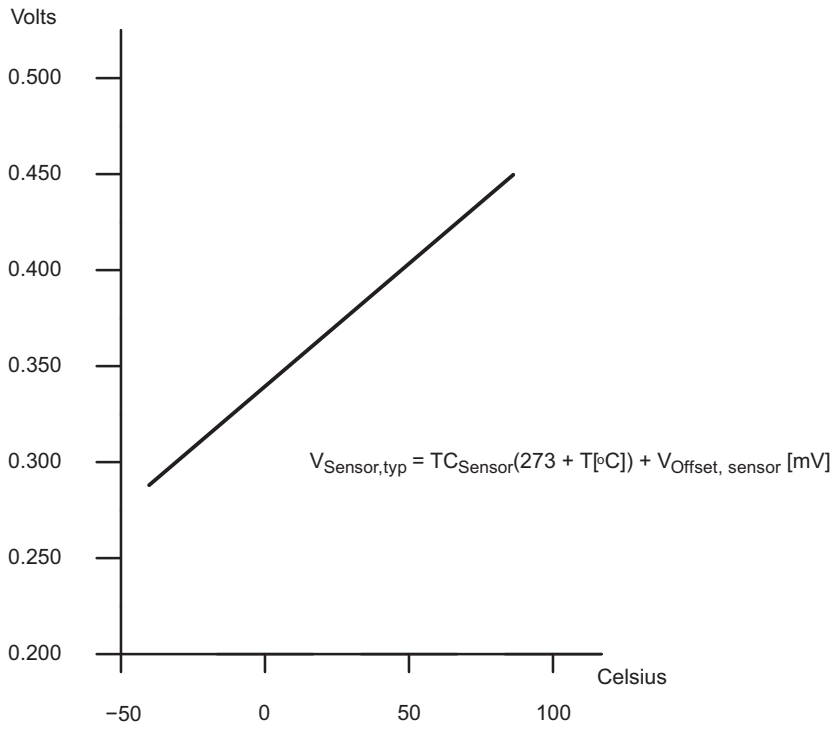


图 **27-12.** 典型的温度传感器传输函数

27.2.12 中断处理

每个 ADC 通道的 SD24_A 有 2 个中断源：

- SD24IFG
- SD24OVIFG

当 SD24IFG 位相应的 SD24MEMx 内存寄存器用转换结果进行写入时，它们会被置位。 如果相应的 SD24IE 位和 GIE 位被置位，那么就会产生一个中断请求。 当在上个转换结果被读取之前，一个转换结果被写入到任何 SD24MEMx 位置时，就会发生 SD24_A 溢出情况。

27.2.12.1 SD24IV，中断向量发生器

所有 SD24_A 中断源被优先化，并与一个中断向量源结合起来。 SD24IV 用于确定哪一个使能的 SD24_A 中断源请求了一个中断。 在 SD24IV 寄存器中，产生了大量的被使能最高优先级的 SD24_A 中断请求（见寄存器描述）。 这个数字可以评估，或将其添加到程序计数器，以便自动进入相应的软件程序。 禁用的 SD24_A 中断不影响 SD24IV 值。

SD24IV 寄存器的任何访问，读取或写入都对 SD24OVIFG 或 SD24IFG 标志没有影响。 可通过读取相关的 SD24ME Mx寄存器，或通过清除软件中的标志来复位 SD24IFG 标志。 SD24OVIFG 位只能用软件复位。

在一个中断服务之后，如果另一个中断挂起，则会产生另一个中断。 例如，如果 SD24OVIF G和一个或多个 SD24IFG 中断挂起，当中断服务例程访问 SD24IV 寄存器时，SD24OVIFG 中断条件会第一个接收服务，且其相应的标志 (s) 必须从软件中清除。 中断服务程序的 RETI 指令被执行后，挂起的优先级最高的 SD24IFG 会产生另一个中断请求。

27.2.12.2 中断延时操作

SD24INTDLYx 位控制为相应通道的第一个中断服务请求定时。 为了在产生一个中断请求之前使数字滤波器建立，此功能为一个完成的转换把中断请求延迟了高达 4 个转换周期。 每次 SD24SC 位被置位时，或者当通道的 SD24GAINx 或 SD24INCHx 位被修改时都会启用该延迟。 SD24INTDLYx 为所选通道延迟周期数禁用溢出中断的产生。 在延迟期间，没有生成延迟转换的中断请求。

27.2.12.3 SD24_A 中断处理软件示例

以下软件示例给出了建议的 SD24IV 和处理开销的使用。 为了自动跳转到相应的程序，SD24IV 值被添加到 PC。

右边距的数字显示了每条指令所需的 CPU 周期。 不同中断源的软件开销包括中断延迟时间和从中断返回周期，但不包含处理本身的任务。 延迟是：

- SD24OVIFG，SD24IFG CH0，CH1 SD24IFG：16 个周期
- CH2 SD24IFG：14 个周期

如果在 ISR 处理期间发生一个较高优先级中断，通道 2 SD24IFG 的中断处理程序会给出一个立即检查的方法。 如果另一个 SD24_A 中断正被挂起，这样可以节省 9 个时钟周期。

```
; Interrupt handler for SD24_A.INT_SD24 ; Enter Interrupt Service Routine 6ADD &SD24IV,PC ; Add
offset to PC 3RETI ; Vector 0: No interrupt 5JMP ADOV ; Vector 2: ADC overflow 2JMP ADM0 ; Vector
4: CH_0 SD24IFG 2JMP ADM1 ; Vector 6: CH_1 SD24IFG 2;; Handler for CH_2 SD24IFG starts here. No
JMP required.;ADM2 MOV &SD24MEM2,xxx ; Move result, flag is reset... ; Other instruction
needed?JMP INT_SD24 ; Check other int pending 2;; Remaining Handlers;ADM1 MOV &SD24MEM1,xxx ;
Move result, flag is reset... ; Other instruction needed?RETI ; Return 5;ADM0 MOV &SD24MEM0,xxx ;
Move result, flag is resetRETI ; Return 5;ADOV ... ; Handle SD24MEMx overflowRETI ; Return 5
```

27.3 SD24_A 寄存器

在表 27-5中列出了 SD24_A 寄存器（寄存器不适用于没被执行的通道，请参阅《特定器件的数据手册》）。

表 27-5. SD24_A 寄存器

寄存器	简表	寄存器类型	地址	初始状态
SD24_A 控制	SD24CTL	读取/写入	0100h	用 PUC 复位
SD24_A 中断向量	SD24IV	读取/写入	0110h	用 PUC 复位
SD24_A 模拟使能(1)	SD24AE	读取/写入	0B7h	用 PUC 复位
SD24_A 通道 0 的控制	SD24CCTL0	读取/写入	0102h	用 PUC 复位
SD24_A 通道 0 的转换存储器	SD24MEM0	读取/写入	0112h	用 PUC 复位
SD24_A 通道 0 的输入控制	SD24INCTL0	读取/写入	0B0h	用 PUC 复位
SD24_A 通道 0 的预置	SD24PRE0	读取/写入	0B8h	用 PUC 复位
SD24_A 通道 1 的控制	SD24CCTL1	读取/写入	0104h	用 PUC 复位
SD24_A 通道 1 的转换存储器	SD24MEM1	读取/写入	0114h	用 PUC 复位
SD24_A 通道 1 的输入控制	SD24INCTL1	读取/写入	0B1h	用 PUC 复位
SD24_A 通道 1 的预置	SD24PRE1	读取/写入	0B9h	用 PUC 复位
SD24_A 通道 2 的控制	SD24CCTL2	读取/写入	0106h	用 PUC 复位
SD24_A 通道 2 的转换存储器	SD24MEM2	读取/写入	0116h	用 PUC 复位
SD24_A 通道 2 的输入控制	SD24INCTL2	读取/写入	0B2h	用 PUC 复位
SD24_A 通道 2 的预置	SD24PRE2	读取/写入	0BAh	用 PUC 复位
SD24_A 通道 3 的控制	SD24CCTL3	读取/写入	0108h	用 PUC 复位
SD24_A 通道 3 的转换存储器	SD24MEM3	读取/写入	0118h	用 PUC 复位
SD24_A 通道 3 的输入控制	SD24INCTL3	读取/写入	0B3h	用 PUC 复位
SD24_A 通道 3 的预置	SD24PRE3	读取/写入	0BBh	用 PUC 复位
SD24_A 通道 4 的控制	SD24CCTL4	读取/写入	010Ah	用 PUC 复位
SD24_A 通道 4 的转换存储器	SD24MEM4	读取/写入	011Ah	用 PUC 复位
SD24_A 通道 4 的输入控制	SD24INCTL4	读取/写入	0B4h	用 PUC 复位
SD24_A 通道 4 的预置	SD24PRE4	读取/写入	0BCh	用 PUC 复位
SD24_A 通道 5 的控制	SD24CCTL5	读取/写入	010Ch	用 PUC 复位
SD24_A 通道 5 的转换存储器	SD24MEM5	读取/写入	011Ch	用 PUC 复位
SD24_A 通道 5 的输入控制	SD24INCTL5	读取/写入	0B5h	用 PUC 复位
SD24_A 通道 5 的预置	SD24PRE5	读取/写入	0BDh	用 PUC 复位
SD24_A 通道 6 的控制	SD24CCTL6	读取/写入	010Eh	用 PUC 复位
SD24_A 通道 6 的转换存储器	SD24MEM6	读取/写入	011Eh	用 PUC 复位
SD24_A 通道 6 的输入控制	SD24INCTL6	读取/写入	0B6h	用 PUC 复位
SD24_A 通道 6 的预置	SD24PRE6	读取/写入	0BEh	用 PUC 复位

(1) 在所有设备上没有执行；请参阅《特定器件的数据手册》。

27.3.1 SD24CTL，SD24_A 控制寄存器

15	14	13	12	11	10	9	8
被保留				SD24XDIVx			SD24LP
r0	r0	r0	r0	rw-0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
SD24DIVx		SD24SSELx		SD24VMIDON	SD24REFON	SD24OVIE	被保留
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	r0

被保留	位 15-12	被保留
SD24XDIVx	位 11-9	SD24_A 时钟分频器 00 /1 01 /3 10 /16 11 /48 1xx 被保留
SD24LP	位 8	低功耗模式。 该位选择了一个速度减少功耗降低模式 0 低功耗模式被禁用 1 低功耗模式被使能。 SD24_A 的最大时钟频率被减小。
SD24DIVx	位 7-6	SD24_A 时钟分频器 00 /1 01 /2 10 /4 11 /8
SD24SSELx	位 5-4	SD24_A 时钟源选择 00 MCLK 01 SMCLK 10 ACLK 11 外部 TACLK
SD24VMIDON	位 3	VMID 缓冲在 0 关闭 1 打开
SD24REFON	位 2	基准发电机在 0 基准关闭 1 基准开启
SD24OVIE	位 1	SD24_A 溢出中断使能。 GIE 位也必须设置为启用中断。 0 溢出中断禁用 1 溢出中断使能
被保留	位 0	被保留

27.3.2 SD24CCTLx，SD24_A 通道 x 控制寄存器

15	14	13	12	11	10	9	8
被保留	SD24BUFx[1]		SD24UNI	SD24XOSR	SD24SNGL	SD24OSRx	
r0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
SD24LSBTOG	SD24LSBACC	SD24OVIFG	SD24DF	SD24IE	SD24IFG	SD24SC	SD24GRP
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	r(w)-0

被保留 位 15 被保留

SD24BUFx 位 14-13 高阻抗输入缓冲模式
- 00 缓冲区禁用
- 01 慢速/电流
- 10 中速/电流
- 11 高速/电流

SD24UNI 位 12 单极性模式选择
- 0 双极性模式
- 1 单极性模式

SD24XOSR 位 11 已扩展过采样率。 该位，与 SD24OSRx 位，选择过采样率。 有关设置请参阅《SD24OSRx 位说明》。

SD24SNGL 位 10 单次转换模式选择
- 0 连续转换模式
- 1 单次转换模式

SD24OSRx 位 9-8 过采样率

当 SD24XOSR = 0 时
- 00 256
- 01 128
- 10 64
- 11 32

当 SD24XOSR = 1 时
- 00 512
- 01 1024
- 10 被保留
- 11 被保留

SD24LSBTOG 位 7 LSB 触发器。 该位，当置位时，每次读取 SD24MEMx 寄存器都会引起 SD24LSBACC 触发。
- 0 SD24LSBACC 不会触发每个 SD24MEMx 读取
- 1 SD24LSBACC 触发每个 SD24MEMx 读取

SD24LSBACC 位 6 LSB 访问。 此位允许访问 SD24_A 转换结果的高于或低于16 位。
- 0 SD24MEMx 包含转换的最有效 16 位。
- 1 SD24MEMx 包含转换的最低有效 16 位。

SD24OVIFG 位 5 SD24_A 溢出中断标志
- 0 无溢出中断挂起
- 1 溢出中断挂起

SD24DF 位 4 SD24_A 的数据格式
- 0 偏移二进制
- 1 2 的补码

SD24IE 位 3 SD24_A 中断使能
- 0 被禁用
- 1 被启用

[1] 在所有设备上未执行（请参阅《特定器件的数据手册》）。当没有执行高阻抗缓冲器时，带 r0 的访问被保留。

SD24IFG	位 2	SD24_A 中断标志。 当新的转换结果可用时才置位 SD24IFG。 当相应的 SD24MEMx 寄存器被读取时，SD24IFG 被自动复位或者被软件清零。
		0 无中断挂起
		1 中断挂起
SD24SC	位 1	SD24_A 开始转换
		0 无转换开始
		1 开始转换
SD24GRP	位 0	SD24_A 集合。 带有下一个较高通道的 SD24_A 集合通道。 不用于最后一个通道。
		0 没被集合
		1 被集合

27.3.3 SD24INCTLx，SD24_A 通道 x 输入控制寄存器

7	6	5	4	3	2	1	0
SD24INTDLYx		**SD24GAINx**			**SD24INCHx**		
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

SD24INTDLYx	位 7-6	转换开始后产生的中断延迟。 这些位选择转换开始后的第一个中断延迟。
		00 第四个采样引起中断
		01 第三个采样引起中断
		10 第二个采样引起中断
		11 第一个采样引起中断
SD24GAINx	位 5-3	SD24_A 前置放大器的增益
		000 x1
		001 x2
		010 x4
		011 x8
		100 x16
		101 x32
		110 被保留
		111 被保留
SD24INCHx	位 2-0	SD24_A 通道差分对输入。 可用的选择依赖于设备。 设置特定器件数据表。
		000 Ax.0
		001 Ax.1(1)
		010 Ax.2(1)
		011 Ax.3(1)
		100 Ax.4(1)
		101 (AV_{CC}- AV_{SS}) / 11
		110 温度传感器
		111 PGA 偏移测量的简称。

(1) 在所有设备上不可用 Ax.1 至 Ax.4（请参阅《特定器件的的数据手册》）。

27.3.4 SD24MEMx，SD24_A 通道 x 的转换存储寄存器

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

转换结果 位 15-0 转换结果。 SD24MEMx 寄存器是否保存高于或低于 16 位的数字滤波器的输出取决于 SD24LSBACC 位。

27.3.5 SD24PREx，SD24_A 通道 x 的预置寄存器

7	6	5	4	3	2	1	0
预置值							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

预置值 位 7-0 SD24_A 数字滤波器的预置值

27.3.6 SD24AE，SD24_A 的模拟输入使能寄存器

7	6	5	4	3	2	1	0
SD24AE7	**SD24AE6**	**SD24AE5**	**SD24AE4**	**SD24AE3**	**SD24AE2**	**SD24AE1**	**SD24AE0**
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

SD24AEx 位 7-0 SD24_A 模拟使能

0 禁用外部输入。 负极输入端被内置连接到 VSS。

1 使能的外部输入

27.3.7 SD24IV，SD24_A 中断向量寄存器

15	14	13	12	11	10	9	8
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
0	**0**	**0**	**SD24IVx**				**0**
r0	r0	r0	r-0	r-0	r-0	r-0	r0

SD24IVx 位 15-0 SD24_A 中断向量值

SD24IV 的目录	中断源	中断标志	中断优先级寄存器
000h	无中断挂起	-	
002h	SD24MEMx 溢出	SD24CCTLx SD24OVIFG(1)	最高
004h	SD24_A 通道 0 的中断	SD24CCTL0 SD24IFG	
006h	SD24_A 通道 1 的中断	SD24CCTL1 SD24IFG	
008h	SD24_A 通道 2 的中断	SD24CCTL2 SD24IFG	
00Ah	SD24_A 通道 3 的中断	SD24CCTL3 SD24IFG	
00Ch	SD24_A 通道 4 的中断	SD24CCTL4 SD24IFG	
00Eh	SD24_A 通道 5 的中断	SD24CCTL5 SD24IFG	
010h	SD24_A 通道 6 的中断	SD24CCTL6 SD24IFG	最低

(1) 当 SD24_A 发生溢出时，为了确定是哪个通道溢出的，用户必须检查所有SD24CCTLxSD24OVIFG 标志。

内嵌式仿真模块 (EEM)

本章描述了嵌入式仿真模块 (EEM) 在所有 MSP430 闪存器件中的应用。

Topic **Page**

28.1 EEM 说明

每个 MSP430 闪存型微控制器都应用了嵌入式仿真模块 (EEM)。 它是通过 JTAG 进行控制和访问的。 每种应用都与器件相关，在 1.3 节*EEM 配置*和器件数据手册中进行了详细描述。

概括的说，具有以下特征：

- 具有非侵入性执行代码的实时断点控制。
- 单步，单步进入和跳过功能
- 支持所有的低功耗模式
- 支持所有的系统频率和时钟源
- 最多可以在地址总线 (MAB) 和数据总线 (MDB) 上设置 8（器件相关）个硬件触发/断点
- 最多可以在 CPU 寄存器写入口设置 2（器件相关）个硬件触发/断点
- 地址总线，数据总线，和 CPU 寄存器触发器可被连接在一起，以便组成 8（器件相关）个硬件触发/断点
- 触发排序（器件相关）
- 用一个集成跟踪缓冲器（器件相关）来存储内部总线和控制信号
- 在一个仿真停顿时，对定时器，通信外围器件，和在其他通用器件或在一个前置基础模块上的其他模块进行时钟控制

图 28-1展示了目前最大的 2xx 嵌入式仿真模块应用的简化模块图。

更多关于怎样将 EEM 的功能和 IAR 嵌入式工作平台™调试增强型一起使用的详细信息，请参阅《*增强型仿真模块高级调试应用报告*》SLAA263在www.msp430.com。 代码调试器 (CCE) 和其他调试器支持 MSP430 拥有相同或相似的功能设置。 更多详细信息请参阅《适用调试器用户指南》。

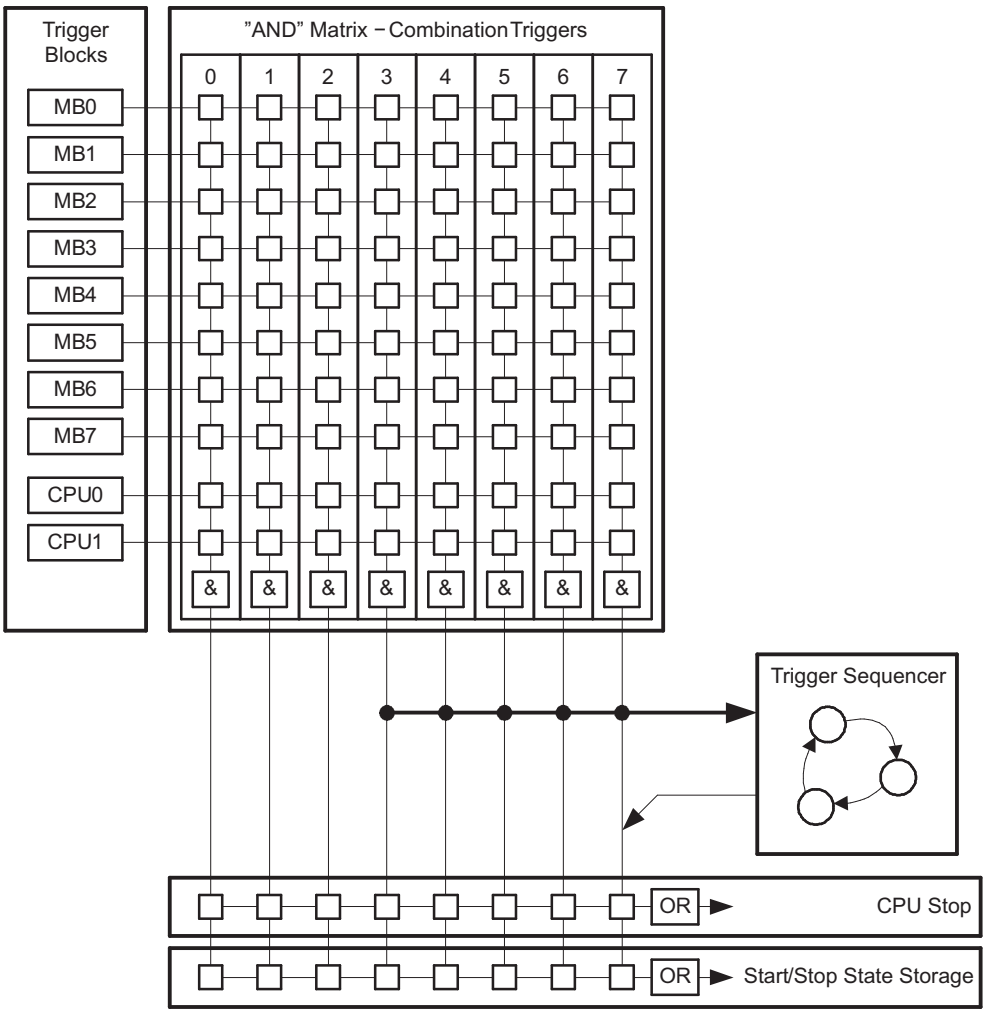


图 **28-1.** 嵌入式仿真模块 **(EEM)** 的大应用

28.2 EEM 构建模块

28.2.1 触发器

MSP430 系统的嵌入式仿真模块的事件控制由触发器组成，触发器是指示一个已发生特定事件的内部信号。可以将这些触发器用作简单的断点，但也可以结合两个或者更多的触发器来探测复杂事件，同时，除了可以停止 CPU 以外，也触发各种各样的反应。

一般来说，这些触发器可以用来控制嵌入式仿真模块的以下功能模块：

- 断点（CPU 停止）
- 状态存储器
- 序列发生器

有两种不同类型的触发器，存储触发器和 CPU 寄存器写入触发器。

为了将 MAB 或 MDB 上的数据和已知的数据进行比较，每一个存储触发器模块都可以被独立选中。 根据应用的嵌入仿真模块，比较方式可以是 =，≠，≥ 或 ≤。 通过使用一个掩码可以将这种比较限制到某些位。 掩码可以是位型的，也可以是字节型的，这由器件来决定。 除了可以选择总线和比较外，也可以选择触发器的触发条件。 这些条件包括读取访问，写写访问，DMA 访问和获取指令。

为了将写入选中寄存器的数据和已知数据进行比较，每一个 CPU 寄存器写入触发器模块都可以被独立地选中。 对每一个触发器都可以独立地选择被观察的寄存器。 比较方式可以是 =，≠，≥ 或 ≤。 通过使用一个位掩码可以将这种比较限制到某些位。

可以把这两种类型触发器组合在一起，以便组成更加复杂的触发器。 例如，一个复杂的触发器可以在一个特定数值被写入一个特定用户的地址时发出信号。

28.2.2 触发序列发生器

触发序列发生器允许在一个事件接收一个中断或状态存储事件前定义触发信号的一个特定顺序。 在触发序列发生器内可以实现以下功能：

- 四种状态（状态 0 到状态 3）
- 每一种状态切换到另一种状态时发生两次传递
- 把复位触发序列发生器的触发器复位为状态 0。

触发序列发生器总是从状态 0 开始，且为了产生一个动作，必须执行到状态 3 。 如果不需要状态 1 和状态 2，它们可以被忽略。

28.2.3 状态储存（内部跟踪缓冲器）

状态存储功能用一个内置缓冲器以一种没有侵占性的方式存储 MAB，MDB，和 CPU 控制信号信息（即：读，写或者获取指令）。 内置缓冲器可以保持多达八个入口。 灵活的配置可以允许用户非常有效地记录感兴趣的信息。

28.2.4 时钟控制

嵌入式仿真模块给器件提供了相关灵活的时钟控制。 该应用对于那些在 CPU 停止以后还需要一个运行时钟的外设是很有用。（例如：为了允许一个 UART 模块完成其一个字符的传送或者允许一个定时器继续产生一个 PWM 信号）。

该时钟控制是灵活的并且支持这两种模块：即需要一个运行时钟的模块和那些当由一个断点导致 CPU 停止时必须被停止的模块。

28.3 嵌入式仿真模块的配置

表 28-1概括了 MSP430 2xx 系列的嵌入式仿真模块的配置。 这些配置都与器件相关—请参阅《器件数据手册》。

表 28-1. 2xx 嵌入式仿真模块的配置

特性	XS	S	M	L
存储器总线触发	2（只有 =, ≠）	3	5	8
存储器总线触发掩码对于	1）低字节 2）高字节	1）低字节 2）高字节	1）低字节 2）高字节	所有 16 或 20 位
CPU 寄存器写入触发	0	1	1	2
组合触发	2	4	6	8
序列发生器	否	不支持	支持	支持
状态存储器	否	否	不支持	支持

基本上任一款 2xx 器件都包括了以下这些特性：

- 至少两个 MAB/MDB 触发器支持：
  - CPU，DMA，读取，和写入访问区分
  - =，≠，≥ 或 ≤ 的比较（在 XS 中仅有 =，≠）
- 至少两个触发结合寄存器
- 使用 CPU 停止反馈的硬件断点
- 带有模块时钟独立控制的时钟控制（在某些 XS 配置中模块时钟的控制是硬编码的）

修订历史记录

修订版本	注释
SLAU144G	第五章*基本时钟模块+*，向 MSP430AFE2xx 器件添加了特定信息的： 图表 5-2。 基本时钟模块+ 方框图 - MSP430AFE2xx 第 5.3 节，寄存器 BCSCTL3 缺省 第 5.3.2，5.3.3，5.3.4 节，可用的寄存器位，缺省，和定义。 添加的章节： 第 18 章*USART 外设接口，UART模式* 第 19 章*USART 外设接口，SPI模式* 第 27 章*SD24_A* 在这个文档中进行了编辑并更改了格式。
SHCU032	第 2.4 节，已修正了 DCO 启动时间。 第 8.2.6 节，已更新引脚振荡器的信息；已添加的图表 8-1。 第 3.4.6.5 节，在 BIC 说明中已修正了排版错误。 第 7.2.1 节，更正了代码示例中的排版错误。
SLAU144I	表 2-3，已更改了晶振引脚上的注释。 1.4.1 节，已修正了闪存 / ROM 末尾的地址。 3.3.5 节，已更改了示例图表。 在下面的章节中已更新的说明：5.1 节，5.2.1 节，5.2.2 节，5.2.3 节，节 5.2.5.2，节 5.2.7.1，5.3.3 节（DCOR 位），5.3.4 节（FLST1Sx 位）。 7.3.2 节和7.3.4 节，已添加了有关 MSP430G2xx 的信息。 8.1 节，已添加了有关 MSP430G22x0 的注释。 Chapter 21，已添加了有关整个 MSP430G2210 的注释。 图 22-1，已更新了结构图。 节 22.2.2.1，已更改了模拟端口选择说明。 22.2.3 节，已更改了基准电压发生器描述。 22.3.1 节，已更新了 SREF 位说明。 22.3.2 节，已更新了 INCHx 位说明。 图 23-1，已将中心偏左复用器上的四个引脚从 GND 改为悬空。 表 24-1，已更正了 CALDCO...名。 在整个文档中进行了编辑更改。

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

重要声明

德州仪器(TI) 及其下属子公司有权根据 JESD46 最新标准, 对所提供的产品和服务进行更正、修改、增强、改进或其它更改， 并有权根据 JESD48 最新标准中止提供任何产品和服务。客户在下订单前应获取最新的相关信息, 并验证这些信息是否完整且是最新的。所有产品的销售都遵循在订单确认时所提供的TI 销售条款与条件。

TI 保证其所销售的组件的性能符合产品销售时 TI 半导体产品销售条件与条款的适用规范。仅在 TI 保证的范围内，且 TI 认为 有必要时才会使用测试或其它质量控制技术。除非适用法律做出了硬性规定，否则没有必要对每种组件的所有参数进行测试。

TI 对应用帮助或客户产品设计不承担任何义务。客户应对其使用 TI 组件的产品和应用自行负责。为尽量减小与客户产品和应 用相关的风险，客户应提供充分的设计与操作安全措施。

TI 不对任何 TI 专利权、版权、屏蔽作品权或其它与使用了 TI 组件或服务的组合设备、机器或流程相关的 TI 知识产权中授予 的直接或隐含权限作出任何保证或解释。TI 所发布的与第三方产品或服务有关的信息，不能构成从 TI 获得使用这些产品或服 务的许可、授权、或认可。使用此类信息可能需要获得第三方的专利权或其它知识产权方面的许可，或是 TI 的专利权或其它 知识产权方面的许可。

对于 TI 的产品手册或数据表中 TI 信息的重要部分，仅在没有对内容进行任何篡改且带有相关授权、条件、限制和声明的情况 下才允许进行复制。TI 对此类篡改过的文件不承担任何责任或义务。复制第三方的信息可能需要服从额外的限制条件。

在转售 TI 组件或服务时，如果对该组件或服务参数的陈述与 TI 标明的参数相比存在差异或虚假成分，则会失去相关 TI 组件 或服务的所有明示或暗示授权，且这是不正当的、欺诈性商业行为。TI 对任何此类虚假陈述均不承担任何责任或义务。

客户认可并同意，尽管任何应用相关信息或支持仍可能由 TI 提供，但他们将独力负责满足与其产品及在其应用中使用 TI 产品 相关的所有法律、法规和安全相关要求。客户声明并同意，他们具备制定与实施安全措施所需的全部专业技术和知识，可预见 故障的危险后果、监测故障及其后果、降低有可能造成人身伤害的故障的发生机率并采取适当的补救措施。客户将全额赔偿因 在此类安全关键应用中使用任何 TI 组件而对 TI 及其代理造成的任何损失。

在某些场合中，为了推进安全相关应用有可能对 TI 组件进行特别的促销。TI 的目标是利用此类组件帮助客户设计和创立其特 有的可满足适用的功能安全性标准和要求的终端产品解决方案。尽管如此，此类组件仍然服从这些条款。

TI 组件未获得用于 FDA Class III（或类似的生命攸关医疗设备）的授权许可，除非各方授权官员已经达成了专门管控此类使 用的特别协议。

只有那些 TI 特别注明属于军用等级或“增强型塑料”的 TI 组件才是设计或专门用于军事/航空应用或环境的。购买者认可并同 意，对并非指定面向军事或航空航天用途的 TI 组件进行军事或航空航天方面的应用，其风险由客户单独承担，并且由客户独 力负责满足与此类使用相关的所有法律和法规要求。

TI 已明确指定符合 ISO/TS16949 要求的产品，这些产品主要用于汽车。在任何情况下，因使用非指定产品而无法达到 ISO/TS16949 要求，TI不承担任何责任。

	产品		应用
数字音频	www.ti.com.cn/audio	通信与电信	www.ti.com.cn/telecom
放大器和线性器件	www.ti.com.cn/amplifiers	计算机及周边	www.ti.com.cn/computer
数据转换器	www.ti.com.cn/dataconverters	消费电子	www.ti.com/consumer-apps
DLP® 产品	www.dlp.com	能源	www.ti.com/energy
DSP - 数字信号处理器	www.ti.com.cn/dsp	工业应用	www.ti.com.cn/industrial
时钟和计时器	www.ti.com.cn/clockandtimers	医疗电子	www.ti.com.cn/medical
接口	www.ti.com.cn/interface	安防应用	www.ti.com.cn/security
逻辑	www.ti.com.cn/logic	汽车电子	www.ti.com.cn/automotive
电源管理	www.ti.com.cn/power	视频和影像	www.ti.com.cn/video
微控制器 (MCU)	www.ti.com.cn/microcontrollers		
RFID 系统	www.ti.com.cn/rfidsys		
OMAP应用处理器	www.ti.com/omap		
无线连通性	www.ti.com.cn/wirelessconnectivity	德州仪器在线技术支持社区	www.deyisupport.com

邮寄地址： 上海市浦东新区世纪大道 1568 号，中建大厦 32 楼 邮政编码： 200122