MSP430FR4xx 和 MSP430FR2xx 系列

用户指南

Literature Number: ZHCU089A
October 2014–Revised August 2015

内容

附图目录

附表目录

请先阅读

关于本手册

本手册介绍 MSP430FR4xx 和 MSP430FR2xx 系列器件的模块和外设。每个说明都给出了一般意义上的模块或外设。并不是所有模块或外设的特性和功能会存在于全部器件上。此外，不同的产品系列，其外设模块的功能实现也会有所不同，或者，在某个单独的产品，或者系列上，没有完全实现某些功能。

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，用户应查阅具体器件的数据表。

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。

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
ADC	模数转换器
BOR	掉电复位
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
DAC	数模转换器
DCO	数字控制振荡器
dst	目的
FLL	频率锁定环路
GIE 模式	通用中断使能
INT (N/2)	N/2 的整数部分
I/O	输入/输出
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；功耗模式简称为 PM
MAB	存储器地址总线
MCLK	主时钟
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；具体细分为 UNMI 和 SNMI
PC	程序计数器
PM	功耗模式
POR	加电复位

PUC	上电清零
RAM	随机存取存储器
SCG	系统时钟发生器
SFR	特殊功能寄存器
SMCLK	子系统主时钟
SNMI	系统 NMI
SP	堆栈指针
SR	状态寄存器
src	源
TOS	栈顶
UNMI	用户 NMI
WDT	看门狗定时器
z16	16 位地址空间

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	位可访问性
rw	读取/写入
r	只读
r0	读为 0
r1	读为 1
w	只写入
w0	写为 0
w1	写为 1
(w)	没有寄存器位被执行；在一个脉冲中写入一个 1 结果。寄存器位始终读为 0。
h0	由硬件清零
h1	由硬件置 1
-0,-1	PUC 后的条件
-(0),-(1)	POR 后的条件
-[0]，-[1]	BOR 后的条件
-{0}，-{1}	欠压后的条件

系统复位，中断，工作模式，系统控制模块 (SYS)

所有器件上都提供有系统控制模块 (SYS)。SYS 的基本特性：

- 掉电复位 (BOR) 和上电复位 (POR) 的处理
- 上电清零 (PUC) 的处理
- （不）可屏蔽中断（SNMI 和 UNMI）事件源的选择和管理
- 通过 JTAG 邮箱 (JMB) 实现的用户数据交换机制
- 引导加载程序 (BSL) 的条目机制
- 配置管理（器件描述符）
- 为复位和 NMI 提供中断向量发生器
- FRAM 写保护
- 片上模块间信令控制

Topic **Page**

1.1 系统控制模块 (SYS) 的介绍

SYS 负责整个系统中各种模块之间的相互作用。SYS 提供的功能并非外设模块本身所固有的。SYS 提供多种功能，例如地址解码、总线仲裁、中断事件汇总和复位生成。

1.2 系统复位和初始化

图 1-1 显示的是系统复位电路，涉及欠压复位 (BOR)、上电复位 (POR) 和上电清零 (PUC)。这些复位信号由不同的事件触发，并且由于各自的信号不同，初始化条件也不同。

BOR 是器件复位。BOR 只能由下列事件生成：

- 器件的上电
- 配置为复位模式时，$\overline{\text{RST}}$/NMI 引脚出现低电平信号
- LPMx.5（LPM3.5 或 LPM4.5）模式下的唤醒事件
- SVS_H 使能后出现低电平（详细信息请参见 “PMM”一章）
- 软件 BOR 事件

当 BOR 产生时，总是会生成一个 POR，但一个 POR 不会生成一个 BOR。以下事件触发POR：

- BOR 信号
- 软件 POR 事件

当 POR 产生时，总是会产生 PUC，但是 PUC 不会生成一个 POR。以下事件触发 PUC：

- POR 信号
- 仅在看门狗模式下出现的看门狗定时器到期（详细信息请参见 WDT_A 章节）
- 看门狗密码违规（详细信息请参阅WDT_A 章节）
- FRAM 存储器密码违规（详细信息请参见 “FRAM 控制器”一章）
- 电源管理模块密码违规（详细信息请参阅“PMM”一章）
- 提取自外设区域

注： 可用的复位的数量和类型可能会因器件的不同而不同。关于所有可用的复位源，请参见具体器件的数据表。

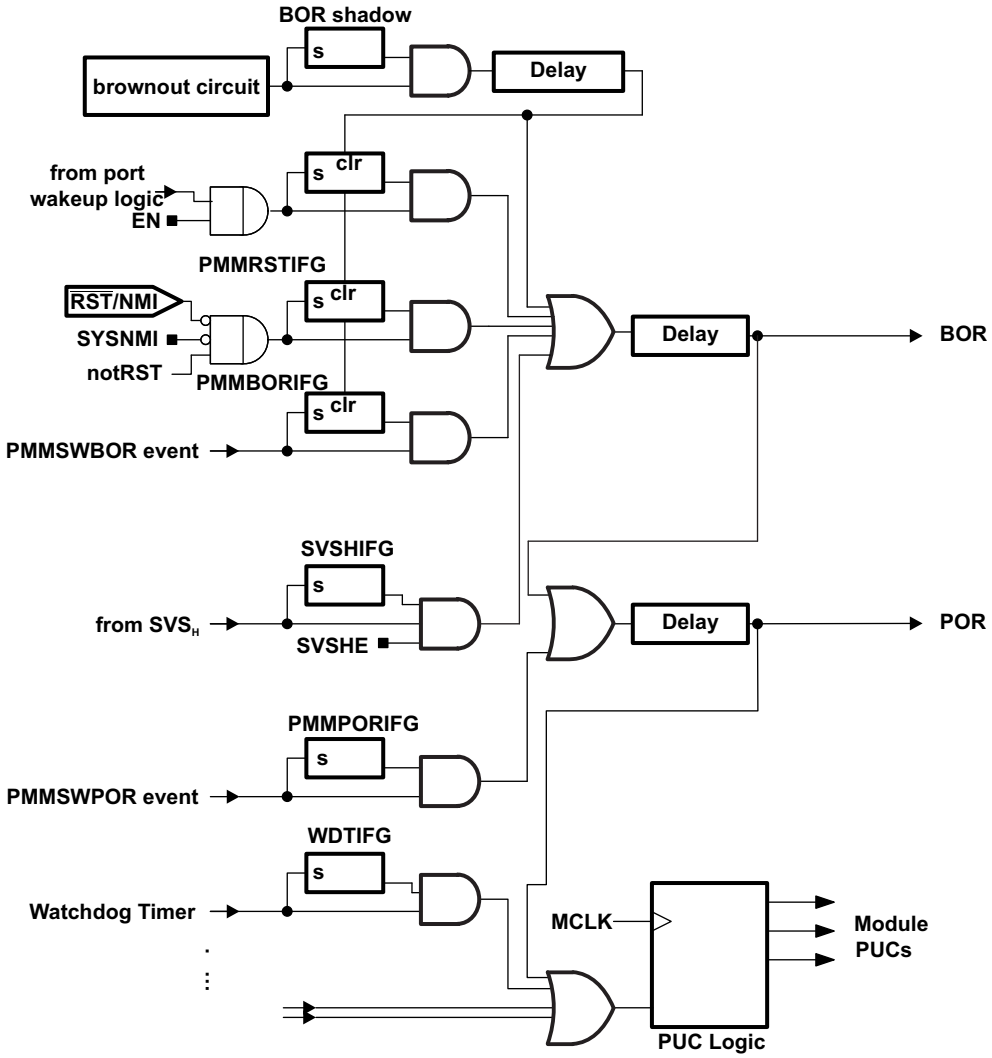


图 **1-1. BOR**、**POR** 和 **PUC** 复位电路

1.2.1 系统复位后的器件初始条件

一个 BOR 之后，初始器件的条件是：

- RST/NMI 引脚在复位模式中被配置。请参阅1.7 节《有关RST/NMI 引脚的配置》。
- I/O 引脚按数字 I/O 章节所述设为输入模式。
- 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。
- 在看门狗模式中，看门狗定时器加电有效。
- 程序计数器 (PC) 被引导代码的地址加载，且引导代码在该地址处开始执行。 引导代码完成之后，PC 将装载包含在 SYSRSTIV 复位地址 (0FFFEh) 中的地址。

一个系统复位后，用户软件须为应用要求来初始化器件。 必须进行以下操作：

- 初始化堆栈指针 (SP)，通常至 RAM 的顶部
- 将看门狗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。

注: 未编程器件或空白器件定义为 FFFEh 寄存器地址中包含复位向量值，等于 FFFFh。空白器件系统复位后，器件将自动进入工作模式 LPM4。有关工作模式的信息，请参见1.4 节。有关中断向量的详细信息，请参见1.3.6 节。

1.3 中断

中断优先级是固定的并且由图 1-2中显示的连接链中的模块安排来定义。中断优先级用于在多个中断挂起时决定优先执行哪个中断。

中断有三个类型：

- 系统复位
- （不）可屏蔽
- 可屏蔽

注: 可用的中断源和他们各自的优先事项的类型会因器件的不同而改变。关于所有中断源及其优先级，请参见具体器件的数据表。

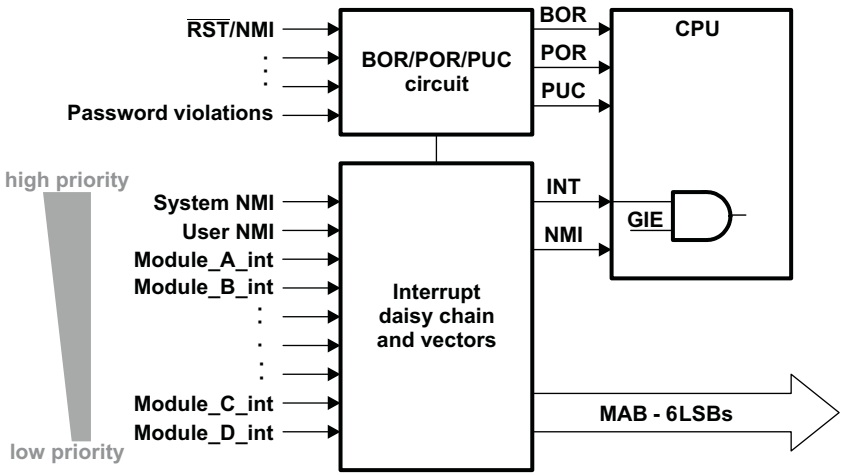


图 1-2. 中断优先级

1.3.1 （不）可屏蔽中断 (NMI)

一般情况下，NMIS 是没被中断使能位 (GIE) 屏蔽的。该系列支持两级 NMI：系统 NMI (SNMI) 和用户 NMI (UNMI)。NMI 源是由中断使能位使能的。当一个 NMI 中断被接受时，该级别的其他 NMIS 将被自动禁用以便能防止相同级别的连续 NMIS 的嵌套。如在表 1-1中所示，程序执行是在存储在 NMI 向量中的地址处开始的。为了使软件具备与较早 MSP430 系列的用户软件的向后兼容性，软件可能会重新启用 NMI 请求源，但这并不需要。

可由以下来信号源生成一个 UNMI 中断：

- 配置在 NMI模式时的，$\overline{\text{RST/NMI}}$上的边沿
- 振荡器发生故障

可由以下信号源生成一个 SNMI 中断：

- FRAM 错误（详细信息请参见 “FRAM 控制器”一章）
- 空置的存储访问
- JTAG邮箱 (JMB) 事件

注： NMI 源的数量和类型可能会因器件的不同而不同。有关所有可用 NMI 源的信息请参阅具体器件的数据表。

1.3.2 SNMI 定时

在 SNMI 处理程序完成一个 RETI 指令后、SNMI 处理程序再次执行前，以一个比它们可以处理的高的 (中断风暴) 速率发生的连续的 SNMIs 可让主程序执行一条指令。在这种情况下，连续的 SNMIs 不会被 UNMIs 中断。这样就避免了在高 SNMI 率上的行为。

1.3.3 可屏蔽中断

可屏蔽中断是由具有中断功能的外设造成的。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。

在每个单独外设中断的各自模块章节中本手册对这些中断进行了讨论。

1.3.4 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 1 时，中断出了力例程被请求。对于请求的（不）可屏蔽中断 (NMI)，必须仅将单独的使能位置 1。

1.3.4.1 中断接受

中断延迟为六个周期，从接受中断请求开始到开始执行中断服务程序的第一条指令，如图 1-3 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. 除了 SCG0，SR 的所有位都被清零，从而结束了任何低功耗模式。由于 GIE 位被清除，之后的中断被禁用。
7. 中断向量的内容被载入 PC；该程序与中断服务程序继续在该地址上执行。

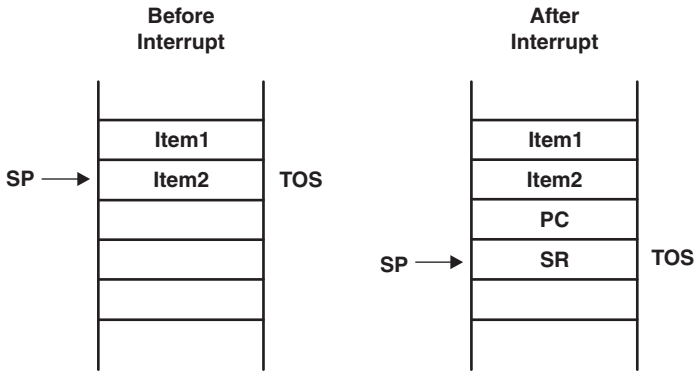


图 **1-3.** 中断处理

注: 使能和禁用中断

由于采用流水线 CPU 架构，即使在使能中断后有中断服务请求待处理，也始终会执行使能中断指令 (EINT) 后的指令。

如果使能中断指令 (EINT) 后紧跟禁用中断指令 (DINT)，则可能不会处理挂起中断。DINT 后的更多指令可能会被错误执行，进而导致 CPU 执行意外操作。建议始终在 EINT 和 DINT 之间插入至少一条指令。请注意：任何可以将 CPU 状态寄存器 GIE 位置 1 并立即清零的指令必须采用同样的方式进行处理。

1.3.4.2 从中断返回

中断处理例程由以下指令终止：

```
RETI //return from an interrupt service routine
```

中断返回需要五个周期来执行以下动作，如图 1-4 所示。

1. 包含之前所有设置的 SR 出栈。无论中断服务程序中使用何种设置，之前的所有 GIE、CPUOFF 和其他位设置此时都将生效。
2. PC 出栈并在中断位置开始执行。

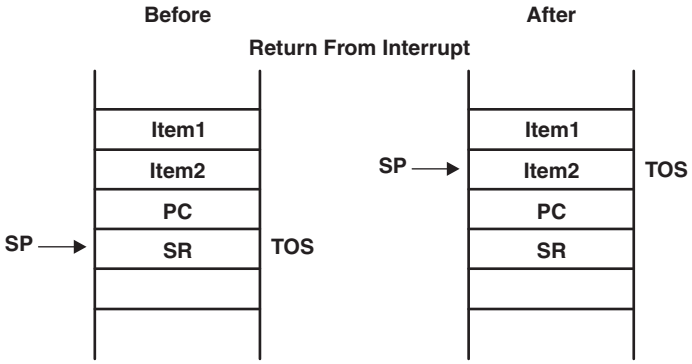


图 **1-4.** 从中断返回

1.3.5 中断嵌套

如果 GIE 位在中断处理例程中内置 1，则中断嵌套被启用。当中断嵌套被启用时，无论中断的优先级是什么，任何在一个中断服务程序期间出现的中断将中断该程序。

1.3.6 中断向量

中断向量位于在地址范围 OFFFFh 至 0FF80h 中，该向量是 64 中断源的最大的一个。一个向量是由用户编程的并指向相应中断服务程序的开始位置。表 1-1 为可用的中断向量示例。有关完整中断向量列表的信息请参阅具体器件的数据表。

表 **1-1.** 中断源、标志、和向量

中断源	中断标志	系统中断	字地址	优先级
复位： 上电，外部复位， 看门狗	… WDTIFG KEYV	… 复位	… 0FFFEh	… 最高
系统 NMI： JTAG 邮箱	JMBINIFG，JMBOUTIFG	（不）可屏蔽	0FFFCh	…
用户 NMI： NMI，振荡器故障， FRAM 存储器访问违规	… NMIIFG OFIFG	… （不）可屏蔽 （不）可屏蔽	… 0FFFAh	… …
特定于器件			0FFF8h	…
…			…	…
看门狗定时器	WDTIFG	可屏蔽	…	…
…			…	…
特定于器件			…	…
保留		可屏蔽	…	最低

某些中断使能位、中断标志和 $\overline{\text{RST}}$/NMI 引脚的控制位位于特殊功能寄存器 (SFR) 中。SFR 位于外设地址范围内，可按字节和字访问。SFR 配置请参阅具体器件的数据表。

1.3.6.1 备用中断向量

在包含 RAM 的器件上，RAM 可用作存储中断向量的备选位置。将 SYSCTL 中的 SYSRIVECT 位置 1 会导致中断向量被重新映射到 RAM 顶部。该位置 1 后，任何中断向量都会保存到备选位置 RAM 中。由于在一个 BOR 上 SYSRIVECT 被自动清零，至关重要的是在 0FFFEH 位置处的复位向量仍是可用的并在固件中被妥善处理。

1.3.7 SYS 中断向量发生器

SYS 会收集所有系统 NMI (SNMI) 源、用户 NMI (UNMI) 源以及所有其他模块的 BOR、POR、PUC（复位）源。它们被组合成 3 个中断向量。中断向量寄存器 SYSRSTIV，SYSSNIV，SYSUNIV 被用来确定哪些标志请求了一个中断或者一复位。当一组具有最高优先级的中断被使能时，就会在相应的 SYSRSTIV，SYSSNIV，SYSUNIV 寄存器中生成一个数字。可以直接把这个数字添加到程序计数器中，从而形成中断服务程序适当部分的一个分支。被禁用的中断不会影响 SYSRSTIV，SYSSNIV，SYSUNIV 的值。读取 SYSRSTIV，SYSSNIV，SYSUNIV 寄存器会自动复位该寄存器的最高挂起的中断标志。如果有另一个中断标志置 1，则在处理完最初的中断后会立即产生另一个中断。写入 SYSRSTIV，SYSSNIV，SYSUNIV 寄存器会自动复位该组所有挂起的中断标志。

1.3.7.1 SYSSNIV 软件示例

下列软件示例展示了 SYSSNIV 的建议使用。SYSSNIV 的值被加入 PC 以便自动跳转到相应程序。对于 SYSRSTIV 和 SYSUNIV，可以使用一个类似的软件方法。下面是一个通用器件的一个示例。向量可以为一个给定器件改变优先级。如需了解向量位置，请参见具体器件的数据表。为了方便移植代码，所有向量应均采用符号编码。

```
SNI_ISR: ADD   &SYSSNIV,PC  ; Add offset to jump table
         RETI               ; Vector 0: No interrupt
         JMP   VMA_ISR      ; Vector 10: VMAIFG
         JMP   JMBI_ISR     ; Vector 12: JMBINIFG
JMBO_ISR:                   ; Vector 14: JMBOUTIFG
        ...                 ; Task_E starts here
        RETI                ; Return
VMA_ISR:                    ; Vector A
       ...                  ; Task_A starts here
        RETI                ; Return
JMBI_ISR:                   ; Vector C
       ...                  ; Task_C starts here
       RETI                 ; Return
```

1.4 工作模式

MSP430 系列针对低功耗应用而设计，并且提供了不同的工作模式，如图 1-5 所示。

工作模式考虑了三种不同需求：

- 低功耗
- 速度和数据吞吐量
- 将各个外设的电流消耗降至最低

低功耗模式 LPM0 到 LPM4 通过 SR 中的 CPUOFF、OSCOFF、SCG0 和 SCG1 位进行配置。包括 SR 中 CPUOFF，OSCOFF，SCG0，SCG1 模式控制位优点在内的优点是，在一个中断服务程序期间目前的运行模式被保存到了堆栈中。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。当设置任一模式控制位时，所选的运行模式会立即生效。在时钟被激活前，带有被禁用时钟的外设运行被禁用。也可用外围设备自身的控制寄存器的设置来禁用外设。所有 I/O 端口引脚、RAM 和寄存器均保持不变。可通过所有已使能中断实现从 LPM0 至 LPM4 唤醒。

当进入 LPMx.5（LPM3.5 或 LPM4.5）时，电源管理模块 (PMM) 的稳压器会被禁用。所有 RAM 和寄存器的内容都丢失了。虽然 I/O寄存器的内容丢失了，但 I/O 引脚状态却被锁定在 LPMx.5 条目。更多细节请参阅数字 I/O 章节。通过上电序列、$\overline{\text{RST}}$ 事件或特定 I/O 可实现从 LPM4.5 唤醒。通过上电序列、$\overline{\text{RST}}$ 事件、RTC 事件、LF 晶振故障或特定 I/O 可实现从 LPM3.5 唤醒。

注: TEST/SBWTCK 引脚用于通过 Spy-Bi-Wire 连接开发工具。TEST/SBWTCK 引脚为高电平时从 LPM2（器件特定）、LPM3 和 LPM4 唤醒的时间与 TEST/SBWTCK 引脚为低电平时可能有所不同。请注意器件与开发工具（例如，MSP-FET430UIF）连接的情况下从 LPM2（特定于器件）、LPM3 和 LPM4 退出时的实时行为。有关详细信息，请参见“PMM”一章。

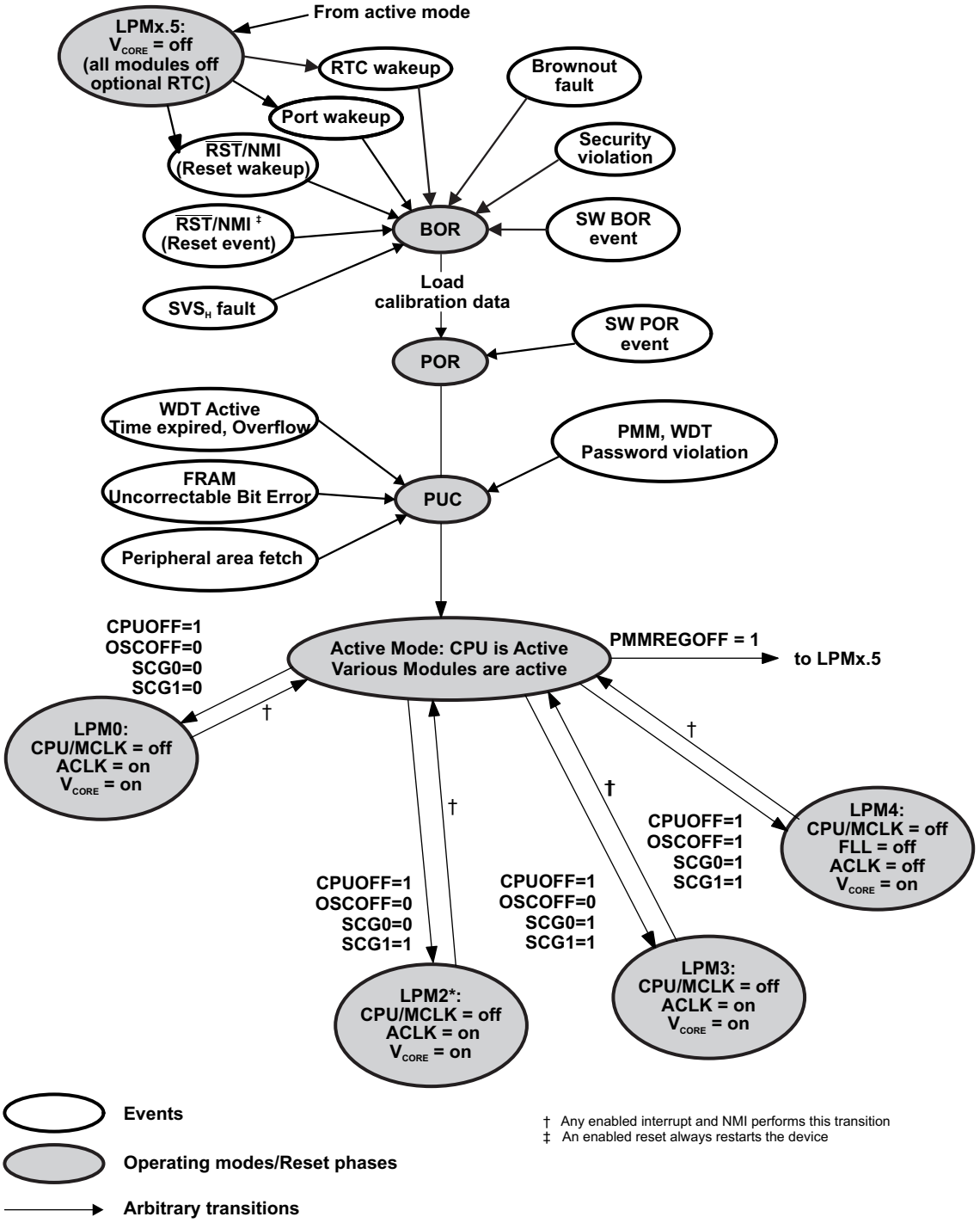


图 **1-5.** 运行模式

表 1-2. 运行模式

SCG1[1]	SCG0	OSCOFF[1]	CPUOFF[1]	模式	CPU 和时钟状态[2]
0	0	0	0	激活	CPU，MCLK 是有效的。 ACLK 被激活。SMCLK 可选地有效 (SMCLKOFF = 0)。 如果源自 ACLK，MCLK，或 SMCLK (SMCLKOFF = 0)，DCO 就会被使能。 如果 DCO 被使能或 DCO 源自 MCLK 或 SMCLK (SMCLKOFF = 0)，那么 DCO 偏压就会被使能。 如果 DCO 被使能了，那么 FLL 也会被使能。
0	0	0	1	LPM0	CPU，MCLK 被禁用。 ACLK 被激活。SMCLK 可选地有效 (SMCLKOFF = 0)。 如果源自 ACLK 或 SMCLK (SMCLKOFF = 0)，DCO 就会被使能。 如果 DCO 被使能或 DCO 源自 MCLK 或 SMCLK (SMCLKOFF = 0)，那么 DCO 偏压就会被使能。 如果 DCO 被使能了，那么 FLL 也会被使能。
1	0	0	1	LPM2（特定于器件）	CPU、MCLK 和 FLL 禁用。 ACLK 被激活。SMCLK 被禁用。 FLL 被禁用。
1	1	0	1	LPM3	CPU、MCLK 和 FLL 禁用。 ACLK 被激活。SMCLK 被禁用。 FLL 被禁用。
1	1	1	1	LPM4	CPU 和所有时钟被禁用。
1	1	1	1	LPM3.5	PMMREGOFF = 1 时，禁用稳压器。RAM 保留为备用存储器。在此模式下，RTC 和 LCD 经正确配置可正常工作。更多详细信息，请参见 RTC 和 LCD 模块。
1	1	1	1	LPM4.5	PMMREGOFF = 1 时，禁用稳压器。无记忆保留。在这种模式下，所有的时钟源都被禁止的，即 RTC 的运行是不可能的。

(1) 按照1.4.2 节中定义的正确进入序列进入 LPMx.5 模式。
(2) 系统时钟和低功耗模式可能会受到时钟请求系统的影响。有关详细信息，请参见“CS”一章。

1.4.1 低功耗模式和时钟请求

如果外设模块需要时钟源以便正常运转，则无论当前是何种低功耗工作模式，它都会自动从时钟系统 (CS) 模块请求时钟源。有关详细信息，请参见3.2.11 节，*低功耗模式运行，外设模块请求*。

由于时钟请求机制，系统可能无法达到 CPU 状态寄存器 (SR) 中位设置请求的低功耗模式，如表 1-3 所示。

表 1-3. 请求与实际 LPM

请求（SR 位，符合表 1-2）	实际 LPM ...		
	如果未请求时钟	如果仅请求 ACLK	如果请求 SMCLK
LPM0	LPM0	LPM0	LPM0
LPM2（特定于器件）	LPM2	LPM2	LPM0
LPM3	LPM3	LPM3	LPM0
LPM4	LPM3	LPM3	LPM0

1.4.2 进入和退出低功耗模式 LPM0 至 LPM4

一个使能的中断事件把器件从低功耗运行模式 LPM0 至 LPM4 中唤醒。退出 LPM0 至 LPM4 的程序流程是：

- 进入中断服务程序
  - PC 和 SR 被存储在堆栈上.
  - CPUOFF，SCG1 和 OSCOFF 位被自动复位.
- 用于从中断服务程序返回的选项：
  - 原先的 SR 从堆栈中弹出，从而恢复之前的运行模式。
  - 执行 RETI 指令时，可在中断服务程序中修改存储在堆栈中的 SR 位，从而返回至不同的工作模式。

```
; Enter LPM0 Example
   BIS   #GIE+CPUOFF,SR                    ; Enter LPM0
;  ...                                     ; Program stops here
;
; Exit LPM0 Interrupt Service Routine
   BIC   #CPUOFF,0(SP)                     ; Exit LPM0 on RETI
   RETI


; Enter LPM3 Example
   BIS   #GIE+CPUOFF+SCG1+SCG0,SR          ; Enter LPM3
;  ...                                     ; Program stops here
;
; Exit LPM3 Interrupt Service Routine
   BIC   #CPUOFF+SCG1+SCG0,0(SP)           ; Exit LPM3 on RETI
   RETI

; Enter LPM4 Example
   BIS #GIE+CPUOFF+OSCOFF+SCG1+SCG0,SR ; Enter LPM4
; ... ; Program stops here
;
; Exit LPM4 Interrupt Service Routine
   BIC #CPUOFF+OSCOFF+SCG1+SCG0,0(SP) ; Exit LPM4 on RETI
   RETI
```

1.4.3 低功耗模式 LPM3.5 和 LPM4.5 (LPMx.5)

低功耗模式 LPM3.5 和 LPM4.5 (LPMx.5 [1]) 可使器件实现最低功耗。在 LPMx.5 下，器件的内核 LDO 关闭。这会产生以下影响：

- 大多数模块断电。
  - 在 LPM3.5 下，只有 RTC LDO 供电的模块才会继续运行。至少有一个 RTC 模块连接 RTC LDO。如需了解连接 RTC LDO 的其他模块（如果有），请参见器件数据表。
  - 在 LPM4.5 下，RTC LDO 及其所连模块均关闭。
- 所有模块和 CPU 的寄存器内容均丢失。
- SRAM 内容丢失。
- 从 LPMx.5 唤醒会导致内核完全复位。
- 应用程序必须在从 LPMx.5 唤醒后初始化整个器件。

[1] 本文档使用缩写“LPMx.5”表示 LPM3.5 和 LPM4.5。

从 LPMx.5 唤醒的时间比任何其他功耗模式都长（请参见具体器件的数据表）。这是因为内核域必须上电，并且器件内部必须完成初始化。此外，应用程序还必须再次初始化。因此，仅当应用程序长时间处于 LPMx.5 模式下时才使用 LPMx.5。

1.4.3.1 进入 LPMx.5

要进入 LPMx.5，请按照以下步骤操作：

1. 将从 LPMx.5 唤醒后所需的全部信息存储到 FRAM 中。
2. 对于 LPM4.5，将所有端口设为通用 I/O（PxSEL0 = 00h 和 PxSEL1 = 00h）。
   对于 LPM3.5，如果使用 LF 晶振，则不更改 LF 晶振共享的 I/O 的设置。这些引脚必须配置为 LFXIN 和 LFXOUT。将所有其他端口引脚设为通用 I/O（PxSEL0 = 00h 和 PxSEL1 = 00h）。
3. 根据应用需要设置端口引脚方向和输出位。
4. 要通过 I/O 实现唤醒，请按以下步骤操作：
   (a) 选择唤醒边沿 (PxIES)
   (b) 清零中断标志 (PxIFG)
   (c) 将中断使能位 (PxIE) 置 1
5. 对于 LPM3.5，必须使能保持激活状态的模块。例如，必要时需使能 RTC。只有连接到 RTC LDO 的模块可保持激活状态。
6. 对于 LPM3.5，根据需要将这些模块中的所有中断源作为唤醒源使能。请参见相应的模块章节。
7. 如果看门狗定时器 WDT 使能且处于看门狗模式，请将其禁用。如果 WDT 使能且处于看门狗模式，则器件不会进入 LPMx.5。
8. 清零 GIE 位：
   ```
   BIC #GIE, SR
   ```
9. 执行以下步骤，将 PMMCTL0 寄存器中的 PMMREGOFF 位置 1：
   (a) 写入正确的 PMM 密码，获取 PMM 控制寄存器的写访问权限。
   ```
   MOV.B #PMMPW_H, &PMMCTL0_H
   ```
   (b) 将 PMMCTL0 寄存器中的 PMMREGOFF 位置 1。
   ```
   BIS.B #PMMREGOFF, &PMMCTL0_L
   ```
   (c) 要在 LPMx.5 下禁用 SVS，将 PMMCTL0 寄存器中的 SVSHE 位清零。
   ```
   BIC.B #SVSHE, &PMMCTL0_L
   ```
   (d) 写入不正确的 PMM 密码，禁用 PMM 控制寄存器的写访问权限。
   ```
   MOV.B #000h, &PMMCTL0_H
   ```
10. 使用如下指令进入 LPMx.5：
    ```
    BIS #CPUOFF+OSCOFF+SCG0+SCG1, SR
    ```

如果连接 RTC LDO 的任意模块已使能，器件将进入 LPM3.5。如果连接 RTC LDO 的任意模块均未使能，器件进入 LPM4.5。

1.4.3.2 从 LPMx.5 退出

以下条件将导致从 LPMx.5 退出：

- I/O 上发生唤醒事件（如果已配置和使能）。相应端口引脚的中断标志 (PxIFG) 置 1。PMMLPM5IFG 位置 1。
- RTC 唤醒事件（如果使能）。RTC 的相应中断标志置 1。PMMLPM5IFG 位置 1。
- RST 引脚唤醒信号。
- 重新启动。SVSHIFG 置 1，或者 PMMIFG 均不置 1。

从 LPMx.5 退出将导致 BOR。程序从复位向量指向的地址开始执行。PMMLPM5IFG = 1 指示从 LPMx.5 唤醒或者系统复位向量字寄存器 SYSRSTIV 可用于解码复位条件（请参见具体器件的数据表）。

从 LPMx.5 唤醒后，I/O 和连接 RTC LDO 的模块的状态均锁定，在 PM5CTL0 寄存器中的 LOCKLPM5 位清零之前保持不变。

1.4.3.3 从 LPM3.5 唤醒

从 LPM3.5 唤醒后执行以下步骤：

1. 按照器件进入 LPM3.5 之前的配置方式对连接 RTC LDO 的模块的寄存器进行初始化，但不使能中断。
2. 按照器件进入 LPM3.5 之前的配置方式对端口寄存器进行初始化，但不使能端口中断。
3. 如果在 LPM3.5 下使用 LF 晶振，则相应的 I/O 必须配置为 LFXIN 和 LFXOUT。必须在时钟系统中使能 LF 晶振（请参见“时钟系统 (CS)”一章）。
4. 清零 PM5CTL0 寄存器中的 LOCKLPM5 位。
5. 根据需要使能端口中断。
6. 使能模块中断。
7. 使能端口和模块中断后，唤醒中断将被视为正常中断进行处理。

1.4.3.4 从 LPM4.5 唤醒

从 LPM4.5 唤醒后执行以下步骤：

1. 按照器件进入 LPM4.5 之前的配置方式对端口寄存器进行初始化，但不使能端口中断。
2. 清零 PM5CTL0 寄存器中的 LOCKLPM5 位。
3. 根据需要使能端口中断。
4. 使能端口中断后，唤醒中断将被视为正常中断进行处理。

如果从 LPM4.5 唤醒后需要晶振，则配置相应引脚，并在清零 LOCKLPM5 位后启动振荡器。

1.4.4 在低功耗模式下扩展时间

当在扩展的低功耗模式期间 DCO 被禁用时，应把 DCO 的温度系数考虑在内。如果温度大幅变化，唤醒时的 DCO 频率可能与进入低功耗模式时偏差甚大，甚至可能超出指定范围。为避免发生这种情况，可在进入低功耗模式之前对 DCO 输出进行二分频，以延长温度变化的时间。

```
; Enter LPM3 Example with DCO/2 settings (to be updated upon the completion of CS module)
   MOV   #FLLD0+FLLN                       ; Set DCO Output divided by 2
   BIS   #GIE+CPUOFF+OSCOFF+SCG1+SCG0,SR   ; Enter LPM3
;  ...                                     ; Program stops
;

;  Interrupt Service Routine
   BIC   #CPUOFF+OSCOFF+SCG1+SCG0,0(SR)    ; Exit LPM3 on RETI
   RETI
```

1.5 低功耗应用原理

通常，降低功耗最重要的因素是使用器件时钟系统尽可能延长 LPM3 或 LPM4 模式的时间。

- 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代替软件驱动的功能。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。

如果应用程序占空比低且响应动作慢，最大化 LPMx.5 的时间可以进一步大幅降低功耗。

1.6 未使用引脚的连接

表 1-4 给出了未使用引脚的正确端接。

表 1-4. 未使用引脚的连接(1)

引脚	电势	注释
AVCC	DV_{CC}	
AVSS	DV_{SS}	
Px.0 至 Px.7	断开	切换到端口功能，输出方向 (PxDIR.n = 1)
$\overline{RST}$/NMI	DV_{CC}或 V_{CC}	47kΩ 上拉电阻或内部上拉电阻，通过 10nF (1.1nF) 下拉电容选择(2)
TDO TDI TMS TCK	开路	JTAG 引脚与通用 I/O 功能共用。如果未用作 JTAG 引脚，应将其切换为端口功能。当用作 JTAG 引脚时，这些引脚应该保持开放。
TEST	断开	该引脚总是有一个使能的内部下拉。

(1) 任何具有第二功能（与通用 I/O 共用）的引脚都应遵循 Px.0 到 Px.7 未使用引脚连接指南。
(2) 当所用器件的 Spy-Bi-Wire 接口处于 Spy-Bi-Wire 模式，并且使用 FET 接口或 GANG 编程器等 TI 工具时，下拉电容不得超过 1.1nF。

1.7 复位引脚 ($\overline{RST}$/NMI) 配置

复位引脚可通过特殊功能寄存器 (SFR) SFRRPCR 配置为复位功能（默认）或者 NMI 功能。将 SYSNMI 置 1 可将 $\overline{RST}$/NMI 引脚配置为外部 NMI 源。外部 NMI 由边沿触发，可通过 SYSNMIIES 选择触发边沿。设置 NMIIE 能使能外部 NMI 的中断。在一个外部 NMI 事件后，NMIIFG 会被设置。

该$\overline{RST}$/NMI 引脚可以有一个上拉或下拉，而不管它们是否有存在。SYSRSTUP 选择上拉或下拉而 SYSRSTRE 选择要使能或不使能的上拉或下拉。如果未使用$\overline{RST}$/NMI 引脚，它就会要求要么有一个选择并使能的内部上拉或一个连接到$\overline{RST}$/NMI 引脚的外部电阻，如在中表 1-4所示。

数字滤波器会抑制复位引脚上的短脉冲，避免器件意外复位。关于最短复位脉冲时间，请参见具体器件的数据表。滤波器仅在引脚配置为复位功能时才有效。如果引脚用作外部 NMI 源，则禁用滤波器。

1.8 配置 JTAG 引脚

JTAG 引脚与通用 I/O 引脚共用。在软件中为 4 线制 JTAG 模式选择 JTAG 引脚的方法有多种。正常情况下，BOR 之后会清零 SYSJTAGPIN。SYSJTAGPIN 清零时，JTAG 配置为通用 I/O。有关控制 JTAG 引脚作为通用 I/O 的详细信息，请参见数字 I/O 章节。如果 SYSJTAG = 1，JTAG 引脚配置为四线制 JTAG 模式，并在出现另一个 BOR 条件之前保持此模式。因此，SYSJTAGPIN 只是一个一次功能的写入。该位无法用软件清零，器件也不会从四线制 JTAG 模式切换到通用 I/O。

1.9 存储器映射 - 用途和功能

1.9.1 存储器映射

1.9.1.1 FR4xx 存储器映射

该存储器映射代表 MSP430FR4xx 器件。尽管各个器件的地址范围不同，但总体行为是相同的。

地址范围	名称及用途	属性						
		可以生成 NMI 读取/写入/提取	在提取访问上生成 PUC	可保护的读取/写入访问	总是能够访问(1)的 PMM 寄存器；可由用户操作的整体擦除	可由用户操作的整体擦除	可由用户操作的存储体擦除	可由用户操作的分块擦除
00000h-00FFFh	外设与间隙							
00000h-000FFh	为系统扩展保留							
00100h-00FEFh	外设						x	
00FF0h-00FF3h	描述符类型(2)						x	
00FF4h-00FF7h	描述符结构的起始地址						x	
01800h-019FFh	信息存储器 B	x	x	x	x	x		
02000h-027FFh	RAM 2KB							
0C400h-0FFFFh	程序 15KB	x	x(3)	x	x	x		
0FF80h-0FFFFh	中断向量							

(1) SYS 和 PMM 的访问权限可单独编程。
(2) 在空存储空间内，数据总线上驱动值 03FFFh。
(3) 所有 MSP430 器件的固定 ID。更多详细信息请参阅1.13.1 节。

1.9.2 空存储空间

空存储区这种存储空间不存在。当被使能时 (VMAIE = 1)，访问空置存储空间会生成一个系统（不）可屏蔽中断 (SNMI)。读取空置存储会引致值 3FFFH。在一个提取的情况下，这被当作 JMP $。提取空置的外设空间的访问会引致一个 PUC。在引导代码被执行后，它的行为就像空置的存储空间并也会导致一个 NMI 访问。

1.9.3 FRAM 写保护

用户可利用 FRAM 写保护防止对 FRAM 内容执行意外写操作。SYS 模块提供两种单独的写保护。

- 用户程序 FRAM 保护 – 始终用于存储用户主程序和常量数据，由 SYSCFG0 寄存器中 PFWP 位保护
- 用户数据 FRAM 保护 – 始终固定在 1800h 到 19FFh，由 SYSCFG0 寄存器中的 DFWP 位保护

使能写保护后，对受保护的 FRAM 执行任何写访问都会导致无效写操作，但不会生成非法中断或复位。建议在初始化程序开始时使能写保护。如果要向 FRAM 中写入某些数据，请在禁用写保护后立即写入数据，然后在完成写操作后立即重新使能写保护。

CAUTION

为防止 FRAM 中存储的程序被意外改写，除非需要执行写操作，否则必须始终使能 FRAM 写保护。写操作应在禁用中断后尽快完成，以便降低执行意外写操作的风险。

1.9.4 引导加载程序 (BSL)

当某个 BSL 进入条件被应用时，BSL 是在启动后被执行的软件。BSL 使得用户能够在原型设计阶段、最终生产阶段、以及运行阶段用微控制器中的嵌入式存储器进行通信。所有存储器映射资源、可编程存储器（FRAM 存储器）、数据存储器 (RAM) 和外设均可根据需要通过 BSL 修改。用户可为基于 FRAM 的器件定制 BSL 代码，使其免遭擦除以及意外或未经授权的访问。

在没有 USB 的器件上，一个基本的 BSL 程序是由 TI 提供的。这支持了常用的 RS232 接口的 UART 协议，使得硬件和软件两方面都能灵活使用。若要使用 BSL，一个特定的 BSL 的条目顺序就须适用于特定器件的引脚。正确的的条目顺序会产生要设置的 SYSBSLIND。所需的功能由一个添加的命令序列初始化。可以通过在一个已定义的用户程序地址上持续运行或通过标准复位序列来退出一个引导加载程序。

用户定义的密码可防止误用 BSL 访问器件存储器。带 USB 的器件含有一个基于 USB 的 BSL 程序（由 TI 提供）。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)。

可用的 BSL 存储空间特定于器件。BSL 存储器分段存储。关于可用段的数量和大小，请参见具体器件的数据表。可以把少量的 RAM 分配到 BSL 存储器中。设置 SYSBSLR 会为 BSL 分配 RAM 的最低的 16 个字节。当 BSL 内存受到保护时，访问这些 RAM 的位置是唯一可能进入受保护的 BSL 内存分部内的办法。

在一些 BSL 应用程序中只允许改变受保护的 BSL 分部的电源管理模块设置可能是可取的。用 SYSPMMPE 位来改变是可能的。通常情况下，该位会被清除并允许访问任何内存位置处的 PMM 控制寄存器。将 SYSPMMPE 置 1 允许仅通过受保护的 BSL 存储器访问 PMM 控制寄存器。SYSPMMPE 置 1 后，只能通过 BOR 事件清零。

1.10 JTAG 邮箱 (JMB) 系统

SYS 模块能够通过普通的 JTAG 或 SBW 测试/调试接口交换用户数据。JMB 旨在调试、编程和测试期间直连 CPU（这适用于此系列的所有 MSP430 器件），并且仅使用少量甚至不使用用户应用程序资源。选择 JTAG 接口的原因在于所有 MSP430 器件上均提供有该接口，并且该接口是调试、编程和测试的专用资源。

JMB 的应用包括：

- 提供输入密码以锁定或解锁器件保护
- 运行时的数据交换 (RTDX)

1.10.1 JMB 配置

JMB 支持两种传输模式，16 位和 32 位。将 JMBMODE 置 1 会使能 32 位传输模式。将 JMBMODE 清零会使能 16 位传输模式。

1.10.2 SYSJMBO0 和 SYSJMBO1 发件箱

有两个 16 位寄存器可用于向 JTAG/SBW 端口发出消息。在 16 位传输模式 (JMBMODE = 0) 下，仅使用 SYSJMBO0。在 32 位传输模式 (JMBMODE = 1) 下，同时使用 SYSJMBO0 和 SYSJMBO1。当应用程序想要向 JTAG 端口发送消息时，在 16 位模式下，它会将数据写入 SYSJMBO0；在 32 位模式下，它会将数据写入 SYSJMBO0 和 SYSJMBO1。

JMBOUT0FG 和 JMBOUT1FG 为只读标志，分别指示 SYSJMBO0 和 SYSJMBO1 的状态。JMBOUT0FG 置 1 时，SYSJMBO0 已被 JTAG 端口读取，并准备好接收新数据。JMBOUT0FG 复位时，SYSJMBO0 未准备好接收新数据。JMBOUT1FG 的行为与 JMBOUT0FG 类似。

1.10.3 SYSJMBI0 和 SYSJMBI1 收件箱

有两个 16 位寄存器可用于从 JTAG 端口接收消息。在 16 位传输模式 (JMBMODE = 0) 下，仅使用 SYSJMBI0。在 32 位传输模式 (JMBMODE = 1) 下，同时使用 SYSJMBI0 和 SYSJMBI1。当 JTAG 端口想要向应用程序发送消息时，在 16 位模式下，它会将数据写入 SYSJMBI0；在 32 位模式下，它会将数据写入 SYSJMBI0 和 SYSJMBI1。

JMBIN0FG 和 JMBIN1FG 标志分别指示 SYSJMBI0 和 SYSJMBI1 的状态。JMBIN0FG 置 1 时，SYSJMBI0 中的数据可供读取。JMBIN0FG 复位时，SYSJMBI0 中没有可用的新数据。JMBIN1FG 的表现一样。

可通过分别清零 JMBCLR0OFF 和 JMBCLR1OFF 来把 JMBIN0FG 和 JMBIN1FG 配置为自动清零。否则，须由软件清零这些标志。

1.10.4 JMB NMI 的使用

如果需要的话，可把 JMB 握手机制配置为使用中断来避免不必要的轮询。在 16 位模式下，JMBOUT0FG 置 1 时，SYSJMBO0 已被 JTAG 端口读取，并准备好接收数据。在 32 位模式下，JMBOUT0FG 置 1 时，SYSJMBO0 和 SYSJMBO1 均已被 JTAG 端口读取，并准备好接收数据。如果 JMBOUTIE 置 1，这些事件会导致一个系统 NMI。在 16 位模式下，JMBOUTIFG 在数据写入 SYSJMBO0 后自动清零。在 32 位模式下，JMBOUTIFG 在数据写入 SYSJMBO0 和 SYSJMBO1 后自动清零。此外，在读取 SYSSNIV 时可以清零 JMBOUTIFG。清零 JMBOUTIE 会禁用 NMI 中断。

在 16 位模式下，JMBINIFG 在 SYSJMBI0 可供读取时置 1。在 32 位模式下，JMBINIFG 在 SYSJMBI0 和 SYSJMBI1 可供读取时置 1。如果 JMBOUTIE 置 1，这些事件会导致一个系统 NMI。在 16 位模式下，JMBINIFG 在 SYSJMBI0 被读取后自动清零。在 32 位模式下，JMBINIFG 在 SYSJMBI0 和 SYSJMBI1 均被读取后自动清零。此外，在读取 SYSSNIV 时可以清零 JMBINIFG。清零 JMBINIE 会禁用 NMI 中断。

1.11 器件安全

本节将介绍几种用于保护器件的安全选项，这些选项可防止 JTAG/SBW 或 BSL 未经授权访问器件存储器。表 1-5 汇总了这些安全选项。

表 1-5. BSL 和 JTAG/SBW 签名

名称	地址	值	器件安全
BSL 密码	FFE0h-FFFFh	用户定义 + 向量表配置	必须首先由 BSL 主机提供该密码，BSL 才能访问器件。
BSL 签名	FF84h-FF87h	5555_5555h	BSL 禁用
		AAAA_AAAAh	BSL 受密码保护。禁用 BSL 密码错误时执行大量擦除操作的特性。
		任意其他值	BSL 受密码保护。BSL 密码错误时执行大量擦除操作。
JTAG/SBW 签名	FF80h-FF83h	FFFF_FFFFh	JTAG/SBW 未锁定
		0000_0000h	
		任意其他值	JTAG/SBW 锁定

1.11.1 JTAG 和 SBW 锁定机制（电子熔丝）

通过限制 JTAG 命令（可通过 JTAG 和 SBW 接口传输到器件）的可访问性可防止器件发生未经授权的访问。这是通过编程电子熔丝实现的。当器件受保护时，JTAG 和 SBW 接口仍保持正常工作，但可直接访问器件的 JTAG 命令被完全禁用。锁定器件需要对位于 FRAM 中的两个签名进行编程。JTAG 签名 1（存储单元 0FF80h）和 JTAG 签名 2（存储单元 0FF82h）控制器件锁定机制的行为。

注: 当器件受保护时，德州仪器 (TI) 不能访问客户退回的器件。只有为 BSL 提供了相应的密钥或者客户提供了解锁机制，才可以访问。

对 JTAG 签名 1 和 JTAG 签名 2 写入除 0000h 或 FFFFh 以外的任意值均可锁定器件。在这种情况下，JTAG 和 SBW 接口授权访问有限的 JTAG 命令集，限制对器件的访问能力。这种情况下解锁器件的唯一方法是，通过 BSL 使用除 0000h 或 FFFFh 以外的任意值来覆盖 JTAG 签名。某些 JTAG 命令在器件受保护后仍可使用，其中包括 BYPASS 命令（请参见 IEEE 标准 1149-2001）和 JMB_EXCHANGE 命令，后者允许访问 JTAG 邮箱系统（详情请参见 1.10.4 节）。

输入的签名在发生下一个 BOR 事件（此时会检查签名）后才会生效。

1.11.2 BSL 安全机制

BSL 签名 1（存储单元 FF84h）和 BSL 签名 2（存储单元 FF86h）这两个 BSL 签名均位于 FRAM 中，可用于控制 BSL 的行为。将 5555h 写入 BSL 签名 1 和 BSL 签名 2 可禁用 BSL 功能，对 BSL 存储空间执行任何访问都会导致空存储器访问，如1.9 节所述。大多数 BSL 命令要求使用用户定义的密码解锁 BSL。输入的密码不正确将擦除器件存储器，这是一种安全机制。将 AAAAh 写入 BSL 签名 1 和 BSL 签名 2 可禁用此安全特性。这样会使 BSL 返回密码错误，但器件存储器不会被擦除。在这种情况下，可以无限制地尝试输入密码。

更多详细信息，请参见《*MSP430 通过引导加载程序 (BSL) 编程用户指南*》（文献编号：SLAU319）和《MSP430FR4xx 与 *MSP430FR2xx 引导加载程序 (BSL)*》（文献编号：SLAU610）。

1.12 器件特定配置

本节将指定器件特定的配置。每节都将介绍器件的唯一配置。

1.12.1 MSP430FR413x 和 MSP430FR203x 配置

本节将介绍特定于 MSP430FR413x 和 MSP430FR203x 器件的配置。

1.12.1.1 FRAM 写保护

用户可通过 FRAM 写保护防止对用户代码和数据执行意外写操作。主代码 FRAM 和信息 FRAM 分别由 SYSCFG0 寄存器中的 PFWP 和 DFWP 位提供写保护。PUC 复位之后，这两个位默认为 1，FRAM 写访问被禁用。必须先清零相应位之后才能对用户代码执行写操作。MSP430FR413x 器件的相关信息请参见节 1.16.2.1；MSP430FR203x 器件的相关信息请参见节 1.16.1.1。

1.12.1.2 红外调制功能

SYS 模块包含红外 (IR) 调制逻辑，器件可使用它直接在外部输出引脚上轻松生成精确调制的 IR 波形，例如 RC-5 数据格式。图 1-6 详细显示了此电路实现。将 SYSCFG1 寄存器中的 IREN 位置 1 可使能该逻辑。如果 IREN 清零，该功能将被忽略，外部引脚默认为通用 I/O。

该功能具有两种不同的 PWM 输入信号，用以支持幅移键控 (ASK) 或移频键控 (FSK) 调制。在 ASK 调制中，第一个 PWM 用于生成载波，第二个 PWM 用于生成包络。在 FSK 调制中，第一个 PWM 和第二个 PWM 表示两种不同的偏移频率。SYSCFG1 寄存器中的 IRMSEL 位指定选定模式。将调制数据输出到外部引脚之前，可通过将 SYSCFG1 寄存器中的 IRPSEL 位置 1 以取反调制信号的极性，从而适应不同的外部驱动电路。

IR 调制功能可与硬件或软件生成的数据结合使用。在硬件数据生成中，eUSCI_A 输出数据，8 位数据自动串行发送。在软件数据生成中，SYSCFG1 寄存器中的 IRDATA 位用于控制要发送逻辑 0 还是逻辑 1。SYSCFG1 寄存器中的 IRDSSEL 位控制数据流来自硬件或固件。

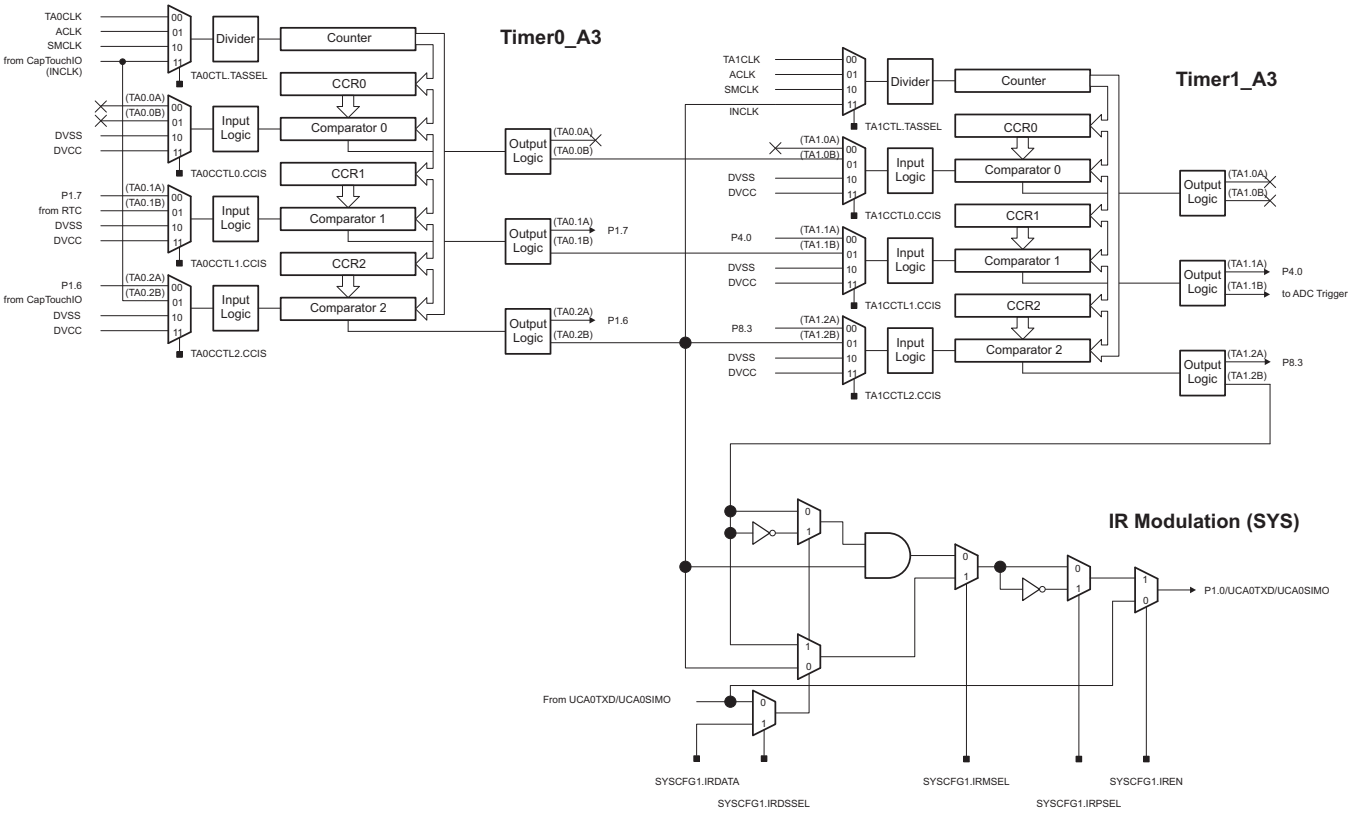


图 **1-6. IR** 调制组合逻辑

1.12.1.3 ADC 引脚使能和 1.2V 基准电压设置

ADC 引脚与 I/O 功能复用。使用 ADC 通道时，必须禁用 I/O 功能来避免 A0 至 A11 这些引脚的功能冲突。将 SYSCFG2 寄存器中的 ADCPTCLx 位置 1 可禁用 I/O 功能。MSP430FR413x 器件的相关信息请参见节 1.16.2.3；MSP430FR203x 器件的相关信息请参见节 1.16.1.3。

当 ADC A4 通道使能时，设置 PMM 寄存器可使 1.2V 片上基准电压输出至 P1.4。请参见图 1-7 和“PMM”一章。

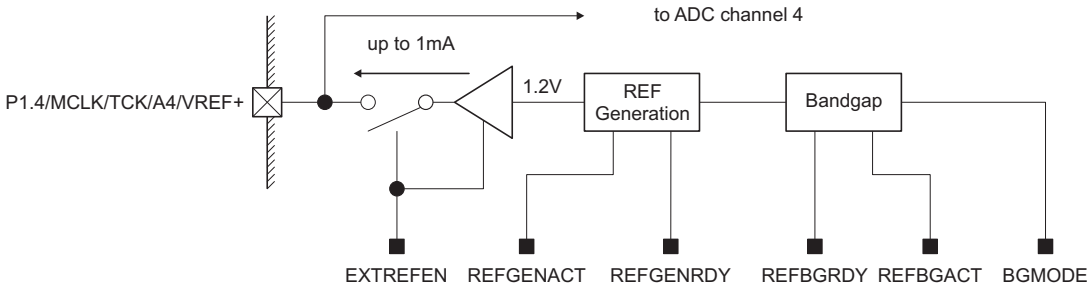


图 1-7. A4 上的 1.2V 基准电压输出

1.12.1.4 LCD 电源引脚使能

在 MSP430FR413x 器件中，LCD 电源引脚与 I/O 功能复用。使用 LCD 时，必须禁用 I/O 功能以避免 LCDCAP0、LCDCAP1、R13、R23、R33 这些引脚的功能冲突。将 SYSCFG2 寄存器中的 LCDPCTL 位置 1 可禁用 I/O 功能并使能 LCD 电源功能（请参见节 1.16.2.3）。

1.13 器件描述符表

每个器件的存储器数据结构可明确标识器件并描述指定器件上的可用模块。SYS 提供此信息并可用于器件适用的软件工具和库，从而明确标识特定器件及其所有模块和功能。器件描述符的有效性可以通过循环冗余校验 (CRC) 来验证。CRC 校验和涵盖器件特定的 TLV 范围。相关定义，请参见具体器件数据表中的 TLV 表。图 1-8 显示了器件描述符表的逻辑顺序和结构。要获取完整器件描述符表及其内容，请参见具体器件的数据表。

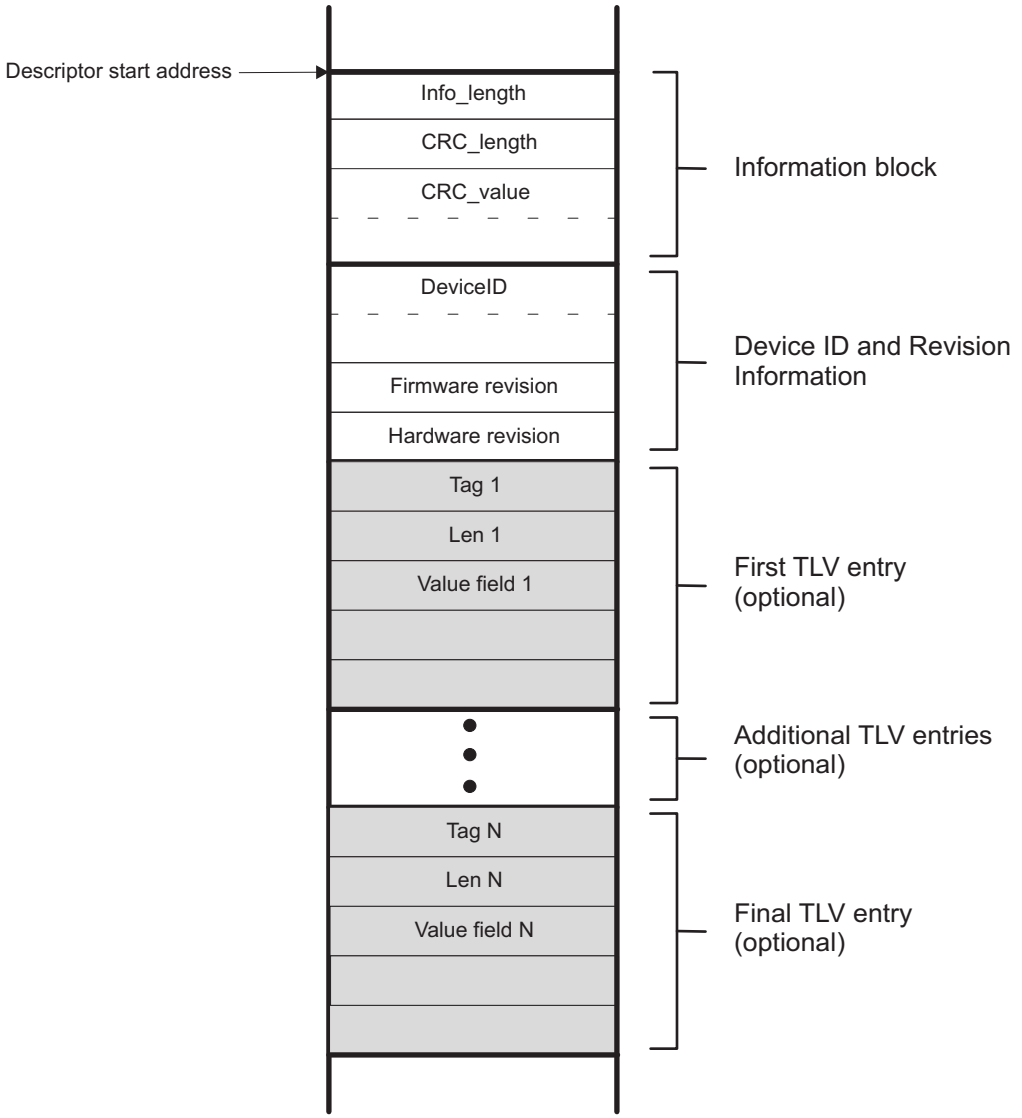


图 **1-8.** 器件描述符表

1.13.1 识别器件类型

地址单元 00FF0h 的值标识器件所属系列。以 80h 开头的所有值均指示一个包含信息块的层级结构和一个包含不同描述符的标签长度值 (TLV) 结构。如果地址单元 00FF0h 的值不为 80h，则表示器件属于早期产品系列，包含从地址单元 0FF0h 开始的平面描述符。图 1-8 中给出的信息块包含器件 ID、芯片版本、固件版本以及其他制造商和工具的相关信息。描述符包含可用外设及其子类型和地址等相关信息，为操作系统构建适用的硬件驱动程序提供所需信息。

描述符长度由 Info_length 表示，计算公式如公式 1 所示。

在 32 位字中，长度 = 2^{Info_length} (1)

例如，如果 Info_length = 5，则描述符的长度等于 128 个字节。

1.13.2 TLV 描述符

TLV 描述符遵循该信息块。由于信息块始终是一个固定的长度，对于一个给定器件系列来说，TLV 描述符的起始位置也是固定的。有关完整的 TLV 结构和哪些描述符是可用的信息请参阅具体器件的数据表。

该 TLV 描述符各自的 TLV 块是唯一的，并总是遵循描述符块的长度。

每个 TLV 描述符包含一个标签字段，用于标识描述符类型。表 1-6列出了目前支持的标记。

表 **1-6.** 标记值

简称	值	说明
LDTAG	01h	传统描述符
PDTAG	02h	外设发现描述符
保留	03h	供将来使用
保留	04h	供将来使用
空白	05h	空白描述符
保留	06h	供将来使用
ADCCAL	11h	ADC 校准
REFCAL	12h	REF 校准
保留	13h-FDh	供将来使用
TAGEXT	FEh	标记扩展器

每个标记字段对其各自的描述符来说都是唯一的，并总是遵循一个长度字段。如果标记值 01h 至 0FDh 并代表了以字节为单位的描述符的长度，则该长度字段是一个字节。如果标记值等于 0FEH (TAGEXT)，则下一个字节会扩展该标记值，之后的两个字节代表以字节为单位的描述符的长度。这样，用户便可以使用类似如下例程（用伪代码编写）在 TLV 描述符表中搜索特定标签值：

```
// Identify the descriptor ID (d_ID_value) for the TLV descriptor of interest:
descriptor_address = TLV_START address;

while ( value at descriptor_address != d_ID_value && descriptor_address != TLV_TAGEND &&
descriptor_address < TLV_END)
{
  // Point to next descriptor
  descriptor_address = descriptor_address + (length of the current TLV block) + 2;
}

if (value at descriptor_address == d_ID_value) {
  // Appropriate TLV descriptor has been found!
  Return length of descriptor & descriptor_address as the location of the TLV descriptor
} else {
  // No TLV descriptor found with a matching d_ID_value
  Return a failing condition
}
```

1.13.3 校准值

TLV 结构包含可用于提高各种功能的测量能力的校准值。在具体器件数据表的 TLV 结构中给出了在一个给定器件上可用的校准值。

1.13.3.1 1.5V 基准电压校准

校准数据包含一个表示可用基准电压 (1.5V) 的字。基准电压在室温下测得。测量值在存储到 TLV 结构之前按 1.5V 实现归一化：

$$Factor_{gain\_1.5Vref} = \frac{V_{REF+}}{1.5V} \times 2^{15} \quad (2)$$

这样，转换结果乘以 $Factor_{gain\_1.5Vref}$ 然后除以 2^{15} 即可得到校正，如对应基准电压公式所示：

$$ADC_{calibrated} = ADC_{raw} \times Factor_{gain\_1.5Vref} \times \frac{1}{2^{15}} \quad (3)$$

在下面的例子中，集成的 1.5V 基准电压用于转换过程。

- 换算结果：0X0100 = 256 十进制
- 基准电压校准系数 ($Factor_{gain\_1.5Vref}$)：0x7BBB

下列步骤演示了如何校正 ADC 转换结果：

- 转换结果乘以 2（该步骤简化了最终的除法）：0x0100 × 0x0002 = 0x0200
- 结果乘以 $Factor_{gain\_1.5Vref}$：0x200 × 0x7BBB = 0x00F7_7600
- 将结果除以 2^{16}： 0x00F7_7600 / 0x0001_0000 = 0x0000_00F7 = 247 十进制

1.13.3.2 ADC 偏移和增益校准

在 TLV 结构中 ADC 的偏移被确定并存储为二补码数。将 ADC_{offset} 与转换结果相加即可完成偏移误差校正。

$$ADC_{offset\_calibrated} = ADC_{raw} + ADC_{offset} \quad (4)$$

ADC 的增益系数按公式 5 计算：

$$Factor_{gain} = \frac{1}{Gain} \times 2^{15} \quad (5)$$

转换结果乘以 $Factor_{gain}$ 后再除以 2^{15} 即可实现增益校正：

$$ADC_{gain\_calibrated} = ADC_{raw} \times Factor_{gain} \times \frac{1}{2^{15}} \quad (6)$$

如果增益和偏移量这两者都要进行校正，那么应首先进行增益校正。

$$ADC_{calibrated} = ADC_{raw} \times Factor_{gain} \times \frac{1}{2^{15}} + ADC_{offset} \quad (7)$$

1.13.3.3 温度传感器的校准

通过使用内部基准电压来校准该温度传感器。1.5V 基准电压包含两个温度下的测量值：室温（数值通常为 30°C ± 3°C）和高温 (85°C ± 3°C)，这两个值存储在 TLV 结构中。温度传感器电压的特征方程（以 mV 为单位）：

$$V_{sense} = TC_{sensor} \times Temperature + V_{sensor} \quad (8)$$

温度系数 TC_{SENSOR}（单位为 mV/°C）表示等式斜率。V_{SENSOR}（单位为 mV）表示等式 Y 轴截距。*Temp*（单位为 °C）表示目标温度。

可使用如下公式为 ADC 测量中使用的每个基准电压计算温度（Temp，°C）：

$$Temperature = (ADC_{raw} - ADC_{30^{\circ}C\_1.5Vref}) \times \left(\frac{55^{\circ}C}{ADC_{85^{\circ}C\_1.5Vref} - ADC_{30^{\circ}C\_1.5Vref}} \right) + 30^{\circ}C \quad (9)$$

1.13.3.4 DCO 校准

可通过存储 DCO 校准值快速设置室温下的最大 DCO 频率（例如，16MHz）。将此值装载到 CSCTL0 寄存器可显著缩短 MCU 重启或退出 LPM 后的 FLL 锁定时间。如果预计退出 LPM 之后可能因温度偏移导致频率过冲，建议在使用前对 DCO 频率进行分频。有关详细信息，请参见1.4.4 节。

1.14 SFR 寄存器

表 1-8中列出了 SFR。SFR 基址列于表 1-7中。SFR 中的很多位在本用户指南中的其它章节中进行了说明。这些位被标记位注释和参考。详细信息请见各自模块的特定章节。

注: 所有寄存器都支持字或字节访问。对于通用寄存器*ANYREG*来说，后缀"_L"(*ANYREG_L*) 是指寄存器的低字节（位 0 到 7）。后缀 "_H" (*ANYREG_H*) 是指寄存器的较高字节（位8 至 位 15）。

表 1-7. SFR 基址

模块	基址
SFR	00100h

表 1-8. SFR 寄存器

偏移量	首字母缩写词	寄存器名称	类型	访问	复位	部分
00h	SFRIE1	中断使能	读取/写入	字	0000h	1.14.1 节
00h	SFRIE1_L (IE1)		读取/写入	字节	00h	
01h	SFRIE1_H (IE2)		读取/写入	字节	00h	
02h	SFRIFG1	中断标志	读取/写入	字	0082h	1.14.2 节
02h	SFRIFG1_L (IFG1)		读取/写入	字节	82h	
03h	SFRIFG1_H (IFG2)		读取/写入	字节	00h	
04h	SFRRPCR	复位引脚控制	读取/写入	字	001Ch	1.14.3 节
04h	SFRRPCR_L		读取/写入	字节	1Ch	
05h	SFRRPCR_H		读取/写入	字节	00h	

1.14.1 SFRIE1 寄存器（偏移 = 00h）[复位 = 0000h]

中断使能寄存器

图 1-9. SFRIE1 寄存器

15	14	13	12	11	10	9	8
保留							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
JMBOUTIE	JMBINIE	保留	NMIIE	VMAIE	保留	OFIE(1)	WDTIE
rw - 0	rw - 0	r0	rw - 0	rw - 0	r0	rw - 0	rw - 0

(1) 详细信息请见"CS"一章。

表 1-9. SFRIE1 寄存器说明

位	字段	类型	复位	说明
15-8	保留	R	0h	保留。始终读为 0。
7	JMBOUTIE	RW	0h	JTAG 邮箱输出中断使能标志 0b = 中断被禁用 1b = 中断被启用
6	JMBINIE	RW	0h	JTAG 邮箱输入中断使能标志 0b = 中断被禁用 1b = 中断被启用
5	保留	R	0h	保留。始终读为 0。
4	NMIIE	RW	0h	NMI 引脚中断使能标志 0b = 中断被禁用 1b = 中断被启用
3	VMAIE	RW	0h	空内存访问中断使能标志 0b = 中断被禁用 1b = 中断被启用
2	保留	R	0h	保留。始终读为 0。
1	OFIE	RW	0h	振荡器故障中断使能标志 0b = 中断被禁用 1b = 中断被启用
0	WDTIE	RW	0h	看门狗定时器中断使能。该位启用间隔定时器模式下的 WDTIFG 中断。没有必要为看门狗模式设置此位。由于 SFRIE1 中的其他位可能用于其他模块，建议使用 BIS.B 或 BIC.B 指令代替 MOV.B 或 CLR.B 指令将此位置 1 或清零。 0b = 中断被禁用 1b = 中断被启用

1.14.2 SFRIFG1 寄存器（偏移 = 02h）[复位 = 0082h]

中断标志寄存器

图 1-10. SFRIFG1 寄存器

15	14	13	12	11	10	9	8
保留							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
JMBOUTIFG	JMBINIFG	保留	NMIIFG	VMAIFG	保留	OFIFG(1)	WDTIFG
rw-(1)	rw-(0)	r0	rw - 0	rw - 0	r0	rw-(1)	rw-0

(1) 详细信息请见"CS"一章。

表 1-10. SFRIFG1 寄存器说明

位	字段	类型	复位	说明
15-8	保留	R	0h	保留。始终读为 0。
7	JMBOUTIFG	RW	1h	JTAG 邮箱输出中断标志 0b = 无中断挂起。在 16 位模式中时 (JMBMODE = 0)，当 JMBO0 已经被 CPU 写入了一个新消息到 JTAG 模块时，这个位被自动清零。在 32 位模式中时 (JMBMODE = 1)，当 JMBO0 和 JMBO1 已经被 CPU 写入新消息到 JTAG 模块时，这个位被清零。当 SYSUNIV 中的相关向量已经被读取时， 这个位也被清零。 1b = 中断挂起，JMBO 已经准备好接收新消息。在 16 位模式中时 (JMBMODE = 0)，JMBO0 已经由 JTAG 模块接收并为从 CPU 接收新消息做好准备。在 32 位模式中时 (JMBMODE = 1)，JMBO0 和 JMBO1 已经被 JTAG 模块接收并已准备好接收来自 CPU 的新消息。
6	JMBINIFG	RW	0h	JTAG 邮箱输入中断标志 0b = 无中断挂起。在 16 位模式中时 (JMBMODE = 0)，当 JMBI0 由 CPU 读取时，这个位被自动清零。在 32 位模式中时 (JMBMODE = 1)，当 JMBI0 和 JMBI1 已经由 CPU 读取时，这个位被自动清零。当 SYSUNIV 中的相关向量已经被读取时，这个位也被清零 1b = 中断挂起，在 JMBIN 寄存器中有一个消息在等待。当 JMBI0 已经由 JTAG 模块写入时，为 16 位模式 (JMBMODE = 0)。当 JMBI0 和 JMBI1 已经由 JTAG 模块写入时，为 32 位模式 (JMBMODE = 1)。
5	保留	R	0h	保留。始终读为 0。
4	NMIIFG	RW	0h	NMI 引脚中断标志 0b = 无中断挂起 1b = 中断挂起
3	VMAIFG	RW	0h	空内存访问中断标志 0b = 无中断挂起 1b = 中断挂起
2	保留	R	0h	保留。始终读为 0。
1	OFIFG	RW	1h	振荡器故障中断标志 0b = 无中断挂起 1b = 中断挂起
0	WDTIFG	RW	0h	看门狗定时器中断标志。在看门狗模式下，WDTIFG 会在发生看门狗超时事件时自清零。可读取 SYSRSTIV 来确定复位是否由一个看门狗超时事件引起。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 SFRIFG1 中的其他位可能用于其他模块，建议使用 BIS.B 或 BIC.B 指令代替 MOV.B 或 CLR.B 指令将 WDTIFG 位置 1 或清零。 0b = 无中断挂起 1b = 中断挂起

1.14.3 SFRRPCR 寄存器（偏移 = 04h）[复位 = 001Ch]

复位引脚控制寄存器

图 1-11. SFRRPCR 寄存器

15	14	13	12	11	10	9	8
保留							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
保留			SYSFLTE	SYSRSTRE	SYSRSTUP	SYSNMIIES	SYSNMI
r0	r0	r0	rw-1	rw-1	rw - 1	rw - 0	rw - 0

表 1-11. SFRRPCR 寄存器说明

位	字段	类型	复位	说明
15-5	保留	R	0h	保留。始终读为 0。
4	SYSFLTE	RW	1h	复位引脚滤波器使能 0b = 禁用复位引脚上的数字滤波器 1b = 使能复位引脚上的数字滤波器
3	SYSRSTRE	RW	1h	复位引脚电阻器使能 0b = $\overline{\text{RST}}$/NMI 引脚上的上拉/下拉电阻器被禁用 1b = $\overline{\text{RST}}$/NMI 引脚上的上拉/下拉电阻器被启用
2	SYSRSTUP	RW	1h	复位电阻器引脚上拉/下拉 0b = 下拉电阻器被选择 1b = 上拉电阻器被选择
1	SYSNMIIES	RW	0h	NMI 边沿选择。当 SYSNMI= 1 时，该位为 NMI 选择中断边沿。修改该位可以触发一个 NMI。为了避免触发意外 NMI，当 SYSNMI = 0 时，修改该位。 0b = NMI 在上升边沿上 1b = NMI 在下降边沿上
0	SYSNMI	RW	0h	NMI 选择。该位为 RST/NMI 引脚选择功能。 0b = 复位功能 1b = NMI 功能

1.15 SYS 寄存器

表 1-12 中列出了 SYS 寄存器。后续小节将详细介绍各个寄存器及其位。每个寄存器都以字为单位对齐。字或字节的数据可以被写入 SYS 配置寄存器。

注: 所有寄存器都支持字或字节访问。对于通用寄存器*ANYREG*来说，后缀“_L”(*ANYREG_L*) 是指寄存器的低字节（位 0 到 7）。后缀“_H”(*ANYREG_H*) 是指寄存器的高字节（位 8 到 15）。

表 **1-12. SYS** 寄存器

偏移量	首字母缩写词	寄存器名称	类型	访问	复位	部分
00h	SYSCTL	系统控制	读取/写入	字	0000h	1.15.1 节
00h	SYSCTL_L		读取/写入	字节	00h	
01h	SYSCTL_H		读取/写入	字节	00h	
02h	SYSBSLC	引导装载程序的配置	读取/写入	字	0000h	1.15.2 节
02h	SYSBSLC_L		读取/写入	字节	00h	
03h	SYSBSLC_H		读取/写入	字节	00h	
06h	SYSJMBC	JTAG 邮箱控制	读取/写入	字	000Ch	1.15.3 节
06h	SYSJMBC_L		读取/写入	字节	0Ch	
07h	SYSJMBC_H		读取/写入	字节	00h	
08h	SYSJMBI0	JTAG 邮箱输入 0	读取/写入	字	0000h	1.15.4 节
08h	SYSJMBI0_L		读取/写入	字节	00h	
09h	SYSJMBI0_H		读取/写入	字节	00h	
0Ah	SYSJMBI1	JTAG 邮箱输入 1	读取/写入	字	0000h	1.15.5 节
0Ah	SYSJMBI1_L		读取/写入	字节	00h	
0Bh	SYSJMBI1_H		读取/写入	字节	00h	
0Ch	SYSJMBO0	JTAG 邮箱输出 0	读取/写入	字	0000h	1.15.6 节
0Ch	SYSJMBO0_L		读取/写入	字节	00h	
0Dh	SYSJMBO0_H		读取/写入	字节	00h	
0Eh	SYSJMBO1	JTAG 邮箱输出 1	读取/写入	字	0000h	1.15.7 节
0Eh	SYSJMBO1_L		读取/写入	字节	00h	
0Fh	SYSJMBO1_H		读取/写入	字节	00h	
1Ah	SYSUNIV	用户 NMI 向量发生器	读取	字	0000h	1.15.8 节
1Ch	SYSSNIV	系统 NMI 向量发生器	读取	字	0000h	1.15.9 节
1Eh	SYSRSTIV	复位向量发生器	读取	字	0002h	1.15.10 节

1.15.1 SYSCTL 寄存器（偏移 = 00h）[复位 = 0000h]

SYS 控制寄存器

图 1-12. SYSCTL 寄存器

15	14	13	12	11	10	9	8
保留							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
保留		SYSJTAGPIN	SYSBSLIND	保留	SYSPMMPE	保留	SYSRIVECT
r0	r0	rw-[0]	rw-[0]	r0	rw-[0]	r0	rw-[0]

表 1-13. SYSCTL 寄存器说明

位	字段	类型	复位	说明
15-8	保留	R	0h	保留。始终读为 0。
7-6	保留	R	0h	保留。始终读为 0。
5	SYSJTAGPIN	RW	0h	专用 JTAG 引脚使能。将此位置 1 可禁用 JTAG 引脚复用的数字功能，并永久使能 JTAG 功能。该位只能被置 1 一次。此位置 1 后将保持置 1 状态，直到发生 BOR。 0b = 共享 JTAG 引脚（JTAG 模式可通过 JTAG/SBW 序列选择） 1b = 专用 JTAG 引脚（显式 4 线式 JTAG 模式选择）
4	SYSBSLIND	RW	0h	BSL 输入指示。该位表示一个在两线制引脚上检测到的 BSL 输入顺序。 0b = 未检测到 BSL 输入顺序 1b = 检测到 BSL 输入顺序
3	保留	R	0h	保留。始终读为 0。
2	SYSPMMPE	RW	0h	PMM 的访问保护。它控制着 PMM 控制寄存器的可访问性。此位置 1 后，只能通过 BOR 清零。 0b = 访问存储器中的任何地方 1b = 只访问受保护的 BSL 分段
1	保留	R	0h	保留。始终读为 0。
0	SYSRIVECT	RW	0h	基于 RAM 的中断向量 0b = 低 64KB 的 FRAM FFFFh 终止地址 TOP 产生的中断向量 1b = 用RAM 的终止地址 TOP 产生的中断向量

1.15.2 SYSBSLC 寄存器（偏移 = 02h）[复位 = 0000h]

引导装载程序的配置寄存器

图 1-13. SYSBSLC 寄存器

15	14	13	12	11	10	9	8
SYSBSLPE	SYSBSLOFF	保留					
rw-[0]	rw-[0]	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
保留					SYSBSLR	保留	
r0	r0	r0	r0	r0	rw-[0]	r0	r0

表 1-14. SYSBSLC 寄存器说明

位	字段	类型	复位	说明
15	SYSBSLPE	RW	0h	引导加载程序存储器保护使能。默认情况下，此位在发生 BOR 事件后由硬件清零（如前文所述）；不过，用于检查有效 BSL 的引导代码可在软件中将此位置 1，以保护 BSL。由于器件随附的 TI BSL 通常都已经过预编程且受保护，因此引导代码会将此位置 1。 0b = 不受保护区。BSL 存储器的读取，编程，和擦除是可能的。 0b = 受保护区。
14	SYSBSLOFF	RW	0h	引导加载程序存储器禁用 0b = 在此区域被读取时 BSL 存储器被寻址。 1b = BSL 存储器的行为就像空白存储。读取会引起 3FFFH 被读取。获取会引起 JMP $ 被执行。
13-3	保留	R	0h	保留。始终读为 0。
2	SYSBSLR	RW	0h	被分配给 BSL 的 RAM 0b = 无 RAM 被分配到 BSL 区 1b = 最低 16 字节的 RAM 被分配到 BSL。
1-0	保留	R	0h	保留。始终读为 0。

1.15.3 SYSJMBC 寄存器（偏移 = 06h）[复位 = 001Ch]

JTAG 邮箱控制寄存器

图 1-14. SYSJMBC 寄存器

15	14	13	12	11	10	9	8
保留							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
JMBCLR1OFF	JMBCLR0OFF	保留	JMBMODE	JMBOUT1FG	JMBOUT0FG	JMBIN1FG	JMBIN0FG
rw-(0)	rw-(0)	r0	rw-0	r-(1)	r-(1)	rw-(0)	rw-(0)

表 1-15. SYSJMBC 寄存器说明

位	字段	类型	复位	说明
15-8	保留	R	0h	保留。始终读为 0。
7	JMBCLR1OFF	RW	0h	传入 JTAG 邮箱 1 标志自动清零禁用 0b = 读取 SYSJMBI1 寄存器后清零 JMBIN1FG 1b = 由软件清零 JMBIN1FG
6	JMBCLR0OFF	RW	0h	传入 JTAG 邮箱 0 标志自动清零禁用 0b = 读取 SYSJMBI0 寄存器后清零 JMBIN0FG 1b = 由软件清零 JMBIN0FG
5	保留	R	0h	保留。始终读为 0。
4	JMBMODE	RW	0h	该位定义了 JMB（SYSJMBI0、SYSJMBI1、SYSJMBO0 和 SYSJMBO1）的工作模式。更改此位之前，填补或取出不完整内容以避免数据丢失。 0b = 16 位传输仅使用 SYSJMBO0 和 SYSJMBI0 1b = 32 位传输使用 SYSJMBI0、SYSJMBI1、SYSJMBO0 和 SYSJMBO1
3	JMBOUT1FG	RW	1h	传出 JTAG 邮箱 1 的标志。 该位在报文写入 SYSJMBO1 的高位字节或（CPU 或其他源）执行字访问后自动清零，在 JTAG 读取报文后置 1。 0b = SYSJMBO1 未准备好接收新数据。 1b = SYSJMBO1 准备好接收新数据。
2	JMBOUT0FG	RW	1h	传出 JTAG 邮箱 0 的标志。 该位在报文写入 SYSJMBO0 的高位字节或（CPU 或其他源）执行字访问后自动清零，在 JTAG 读取报文后置 1。 0b = SYSJMBO0 未准备好接收新数据。 1b = SYSJMBO0 准备好接收新数据。
1	JMBIN1FG	RW	0h	传入 JTAG 邮箱 1 的标志。 该位在 SYSJMBI1 中有一条新报文（JTAG 提供）时置 1。 当 JMBCLR1OFF = 0（自动清零模式）时，读取 SYSJMBI1 后会自动清零该标志。如果 JMBCLR1OFF = 1，JMBIN1FG 必须通过软件清零。 0b = SYSJMBI1 无新数据 1b = SYSJMBI1 有新数据
0	JMBIN0FG	RW	0h	传入 JTAG 邮箱 0 的标志。 该位在 SYSJMBI0 中有一条新报文（JTAG 提供）时置 1。 当 JMBCLR0OFF = 0（自动清零模式）时，读取 SYSJMBI0 后会自动清零该标志。如果 JMBCLR0OFF = 1，JMBIN0FG 必须通过软件清零。 0b = SYSJMBI1 无新数据 1b = SYSJMBI1 有新数据

1.15.4 SYSJMBI0 寄存器（偏移 = 08h）[复位 = 0000h]

JTAG 邮箱输入 0 寄存器

图 **1-15. SYSJMBI0** 寄存器

15	14	13	12	11	10	9	8
MSGHI							
r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
7	6	5	4	3	2	1	0
MSGLO							
r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0

表 **1-16. SYSJMBI0** 寄存器说明

位	字段	类型	复位	说明
15-8	MSGHI	R	0h	JTAG 邮箱传入消息的高字节
7-0	MSGLO	R	0h	JTAG 邮箱传入消息的低字节

1.15.5 SYSJMBI1 寄存器（偏移 = 0Ah）[复位 = 0000h]

JTAG 邮箱输入 1 寄存器

图 **1-16. SYSJMBI1** 寄存器

15	14	13	12	11	10	9	8
MSGHI							
r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
7	6	5	4	3	2	1	0
MSGLO							
r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0

表 **1-17. SYSJMBI1** 寄存器说明

位	字段	类型	复位	说明
15-8	MSGHI	R	0h	JTAG 邮箱传入消息的高字节
7-0	MSGLO	R	0h	JTAG 邮箱传入消息的低字节

1.15.6 SYSJMBO0 寄存器（偏移 = 0Ch）[复位 = 0000h]

JTAG 邮箱输出 0 寄存器

图 1-17. SYSJMBO0 寄存器

15	14	13	12	11	10	9	8
MSGHI							
rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0
7	6	5	4	3	2	1	0
MSGLO							
rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0

表 1-18. SYSJMBO0 寄存器说明

位	字段	类型	复位	说明
15-8	MSGHI	RW	0h	JTAG 邮箱传出消息的高字节
7-0	MSGLO	RW	0h	JTAG 邮箱传出消息的低字节

1.15.7 SYSJMBO1 寄存器（偏移 = 0Eh）[复位 = 0000h]

JTAG 邮箱输出 1 寄存器

图 1-18. SYSJMBO1 寄存器

15	14	13	12	11	10	9	8
MSGHI							
rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0
7	6	5	4	3	2	1	0
MSGLO							
rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0

表 1-19. SYSJMBO1 寄存器说明

位	字段	类型	复位	说明
15-8	MSGHI	RW	0h	JTAG 邮箱传出消息的高字节
7-0	MSGLO	RW	0h	JTAG 邮箱传出消息的低字节

1.15.8 SYSUNIV 寄存器（偏移 = 1Ah）[复位 = 0000h]

用户 NMI 向量寄存器

注: 后续会有更为复杂的器件的附加事件添加到此表格中；删除事件源可缩短此表格的长度。只能通过所用器件的相应包含文件对向量进行符号访问。

图 1-19. SYSUNIV 寄存器

15	14	13	12	11	10	9	8
SYSUNIV							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
SYSUNIV							
r0	r0	r0	r-0	r-0	r-0	r-0	r0

表 1-20. SYSUNIV 寄存器说明

位	字段	类型	复位	说明
15-0	SYSUNIV	R	0h	用户 NMI 向量。生成了一个值，该值可以用作快速中断服务程序处理的地址偏移量。写入该寄存器将清零所有挂起的用户 NMI 标志。 具体值列表请参见具体器件的数据表。

1.15.9 SYSSNIV 寄存器（偏移 = 1Ch）[复位 = 0000h]

系统 NMI 向量寄存器

注: 后续会有更为复杂的器件的附加事件添加到此表格中；删除事件源可缩短此表格的长度。只能通过所用器件的相应包含文件对向量进行符号访问。

图 1-20. SYSSNIV 寄存器

15	14	13	12	11	10	9	8
SYSSNIV							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
SYSSNIV							
r0	r0	r0	r-0	r-0	r-0	r-0	r0

表 1-21. SYSSNIV 寄存器说明

位	字段	类型	复位	说明
15-0	SYSSNIV	R	0h	系统 NMI 向量。生成了一个值，该值可以用作快速中断服务程序处理的地址偏移量。写入该寄存器将清零所有挂起的系统 NMI 标志。 具体值列表请参见具体器件的数据表。

1.15.10 SYSRSTIV 寄存器（偏移 = 1Eh）[复位 = 0002h]

复位中断向量寄存器

注: 后续会有更为复杂的器件的附加事件添加到此表格中；删除事件源可缩短此表格的长度。只能通过所用器件的相应包含文件对向量进行符号访问。

图 1-21. SYSRSTIV寄存器

15	14	13	12	11	10	9	8
SYSRSTIV							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
SYSRSTIV							
r0	r0	r-0	r-0	r-0	r-0	r-1	r0

表 1-22. SYSRSTIV 寄存器说明

位	字段	类型	复位	说明
15-0	SYSRSTIV	R	0h	复位中断向量。生成一个可用作偏移地址的值，以实现快速中断服务程序处理来标识引起复位的最终原因（BOR、POR 或 PUC）。写入该寄存器将清零所有挂起复位源标志。 具体值列表请参见具体器件的数据表。

1.16 系统配置寄存器

系统配置寄存器特定于器件，仅适用于特定的器件系列。每个寄存器都以字为单位对齐。字或字节的数据可以被写入 SYS 配置寄存器。

关于 MSP430FR203x 配置寄存器，请参见1.16.1 节。

关于 MSP430FR413x 配置寄存器，请参见1.16.2 节。

1.16.1 FR203x 系统配置寄存器

表 1-23 列出了所有寄存器。后续小节将详细介绍各个寄存器及其位。

表 1-23. FR203x 系统配置寄存器

偏移量	首字母缩写词	寄存器名称	类型	访问	复位	部分
20h	SYSCFG0	系统配置 0	读取/写入	字	0003h	节 1.16.1.1
20h	SYSCFG0_L		读取/写入	字节	03h	
21h	SYSCFG0_H		读取/写入	字节	00h	
22h	SYSCFG1	系统配置 1	读取/写入	字	0000h	节 1.16.1.2
22h	SYSCFG1_L		读取/写入	字节	00h	
23h	SYSCFG1_H		读取/写入	字节	00h	
24h	SYSCFG2	系统配置 2	读取/写入	字	0000h	节 1.16.1.3
24h	SYSCFG2_L		读取/写入	字节	00h	
25h	SYSCFG2_H		读取/写入	字节	00h	

1.16.1.1 FR203x SYSCFG0 寄存器（偏移 = 00h）[复位 = 0003h]

系统配置寄存器 0

图 **1-22. SYSCFG0** 寄存器

15	14	13	12	11	10	9	8
保留							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
保留						DFWP	PFWP
r0	r0	r0	r0	r0	r0	rw-1	rw-1

表 **1-24. SYSCFG0** 寄存器说明

位	字段	类型	复位	说明
15-3	保留	R	0h	保留。始终读为 0。
1	DFWP	RW	1h	数据 FRAM 写保护 0b = 使能数据 FRAM 写操作 1b = 数据 FRAM 受写保护（不可写）
0	PFWP	RW	1h	程序 FRAM 写保护 0b = 使能程序 FRAM 写访问 1b = 程序 FRAM 受写保护（不可写）

1.16.1.2 FR203x SYSCFG1 寄存器（偏移 = 02h）[复位 = 0000h]

系统配置寄存器 1

图 **1-23. SYSCFG1** 寄存器

15	14	13	12	11	10	9	8
保留							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
保留			IRDATA	IRDSSEL	IRMSEL	IRPSEL	IREN
r0	r0	r0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0

表 **1-25. SYSCFG1** 寄存器说明

位	字段	类型	复位	说明
15-5	保留	R	0h	保留。始终读为 0。
4	IRDATA	RW	0h	红外数据 0b = 红外数据逻辑 0 1b = 红外数据逻辑 1
3	IRDSSEL	RW	0h	红外数据源选择 0b = 来自硬件外设（器件配置后） 1b = 来自 IRDATA 位
2	IRMSEL	RW	0h	红外模式选择 0b = ASK 模式 1b = FSK 模式
1	IRPSEL	RW	0h	红外极性选择 0b = 正极性 1b = 反极性
0	IREN	RW	0h	红外使能 0b = 禁用红外功能 1b = 使能红外功能

1.16.1.3 FR203x SYSCFG2 寄存器（偏移 = 04h）[复位 = 0000h]

系统配置寄存器 2

图 **1-24. SYSCFG2** 寄存器

15	14	13	12	11	10	9	8
保留						ADCPCTL9	ADCPCTL8
r0	r0	r0	r0	r0	r0	rw - 0	rw - 0
7	6	5	4	3	2	1	0
ADCPCTL7	ADCPCTL6	ADCPCTL5	ADCPCTL4	ADCPCTL3	ADCPCTL2	ADCPCTL1	ADCPCTL0
rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0

表 **1-26. SYSCFG2** 寄存器说明

位	字段	类型	复位	说明
15-10	保留	R	0h	保留。始终读为 0。
9	ADCPCTL9	RW	0h	ADC 输入 A9 引脚选择 0b = 禁用 ADC 输入 A9 1b = 使能 ADC 输入 A9
8	ADCPCTL8	RW	0h	ADC 输入 A8 引脚选择 0b = 禁用 ADC 输入 A8 1b = 使能 ADC 输入 A8
7	ADCPCTL7	RW	0h	ADC 输入 A7 引脚选择 0b = 禁用 ADC 输入 A7 1b = 使能 ADC 输入 A7
6	ADCPCTL6	RW	0h	ADC 输入 A6 引脚选择 0b = 禁用 ADC 输入 A6 1b = 使能 ADC 输入 A6
5	ADCPCTL5	RW	0h	ADC 输入 A5 引脚选择 0b = 禁用 ADC 输入 A5 1b = 使能 ADC 输入 A5
4	ADCPCTL4	RW	0h	ADC 输入 A4 引脚选择 0b = 禁用 ADC 输入 A4 1b = 使能 ADC 输入 A4
3	ADCPCTL3	RW	0h	ADC 输入 A3 引脚选择 0b = 禁用 ADC 输入 A3 1b = 使能 ADC 输入 A3
2	ADCPCTL2	RW	0h	ADC 输入 A2 引脚选择 0b = 禁用 ADC 输入 A2 1b = 使能 ADC 输入 A2
1	ADCPCTL1	RW	0h	ADC 输入 A1 引脚选择 0b = 禁用 ADC 输入 A1 1b = 使能 ADC 输入 A1
0	ADCPCTL0	RW	0h	ADC 输入 A0 引脚选择 0b = 禁用 ADC 输入 A0 1b = 使能 ADC 输入 A0

1.16.2 FR413x 系统配置寄存器

表 1-27 列出了所有寄存器。后续小节将详细介绍各个寄存器及其位。

表 1-27. FR413x 系统配置寄存器

偏移量	首字母缩写词	寄存器名称	类型	访问	复位	部分
20h	SYSCFG0	系统配置 0	读取/写入	字	0003h	节 1.16.2.1
20h	SYSCFG0_L		读取/写入	字节	03h	
21h	SYSCFG0_H		读取/写入	字节	00h	
22h	SYSCFG1	系统配置 1	读取/写入	字	0000h	节 1.16.2.2
22h	SYSCFG1_L		读取/写入	字节	00h	
23h	SYSCFG1_H		读取/写入	字节	00h	
24h	SYSCFG2	系统配置 2	读取/写入	字	0000h	节 1.16.2.3
24h	SYSCFG2_L		读取/写入	字节	00h	
25h	SYSCFG2_H		读取/写入	字节	00h	

1.16.2.1 FR413x SYSCFG0 寄存器（偏移 = 00h）[复位 = 0003h]

系统配置寄存器 0

图 **1-25. SYSCFG0** 寄存器

15	14	13	12	11	10	9	8
保留							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
保留						DFWP	PFWP
r0	r0	r0	r0	r0	r0	rw-1	rw-1

表 **1-28. SYSCFG0** 寄存器说明

位	字段	类型	复位	说明
15-3	保留	R	0h	保留。始终读为 0。
1	DFWP	RW	1h	数据 FRAM 写保护 0b = 使能数据 FRAM 写操作 1b = 数据 FRAM 受写保护（不可写）
0	PFWP	RW	1h	程序 FRAM 写保护 0b = 使能程序 FRAM 写访问 1b = 程序 FRAM 受写保护（不可写）

1.16.2.2 FR413x SYSCFG1 寄存器（偏移 = 02h）[复位 = 0000h]

系统配置寄存器 1

图 **1-26. SYSCFG1** 寄存器

15	14	13	12	11	10	9	8
保留							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
保留			IRDATA	IRDSSEL	IRMSEL	IRPSEL	IREN
r0	r0	r0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0

表 **1-29. SYSCFG1** 寄存器说明

位	字段	类型	复位	说明
15-5	保留	R	0h	保留。始终读为 0。
4	IRDATA	RW	0h	红外数据 0b = 红外数据逻辑 0 1b = 红外数据逻辑 1
3	IRDSSEL	RW	0h	红外数据源选择 0b = 来自硬件外设（器件配置后） 1b = 来自 IRDATA 位
2	IRMSEL	RW	0h	红外模式选择 0b = ASK 模式 1b = FSK 模式
1	IRPSEL	RW	0h	红外极性选择 0b = 正极性 1b = 反极性
0	IREN	RW	0h	红外使能 0b = 禁用红外功能 1b = 使能红外功能

1.16.2.3 FR413x SYSCFG2 寄存器（偏移 = 04h）[复位 = 0000h]

系统配置寄存器 2

图 1-27. SYSCFG2 寄存器

15	14	13	12	11	10	9	8
保留			LCDPCTL	保留	保留	ADCPCTL9	ADCPCTL8
r0	r0	r0	rw-0	r0	r0	rw - 0	rw - 0
7	6	5	4	3	2	1	0
ADCPCTL7	ADCPCTL6	ADCPCTL5	ADCPCTL4	ADCPCTL3	ADCPCTL2	ADCPCTL1	ADCPCTL0
rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0

表 1-30. SYSCFG2 寄存器说明

位	字段	类型	复位	说明
15-13	保留	R	0h	保留。始终读为 0。
12	LCDPCTL	RW	0h	LCD 电源引脚（LCDCAP0、LCDCAP1、R23、R33）控制。 0b = 禁用 LCD 电源引脚 1b = 使能 LCD 电源引脚
11-10	保留	R	0h	保留。始终读为 0。
9	ADCPCTL9	RW	0h	ADC 输入 A9 引脚选择 0b = 禁用 ADC 输入 A9 1b = 使能 ADC 输入 A9
8	ADCPCTL8	RW	0h	ADC 输入 A8 引脚选择 0b = 禁用 ADC 输入 A8 1b = 使能 ADC 输入 A8
7	ADCPCTL7	RW	0h	ADC 输入 A7 引脚选择 0b = 禁用 ADC 输入 A7 1b = 使能 ADC 输入 A7
6	ADCPCTL6	RW	0h	ADC 输入 A6 引脚选择 0b = 禁用 ADC 输入 A6 1b = 使能 ADC 输入 A6
5	ADCPCTL5	RW	0h	ADC 输入 A5 引脚选择 0b = 禁用 ADC 输入 A5 1b = 使能 ADC 输入 A5
4	ADCPCTL4	RW	0h	ADC 输入 A4 引脚选择 0b = 禁用 ADC 输入 A4 1b = 使能 ADC 输入 A4
3	ADCPCTL3	RW	0h	ADC 输入 A3 引脚选择 0b = 禁用 ADC 输入 A3 1b = 使能 ADC 输入 A3
2	ADCPCTL2	RW	0h	ADC 输入 A2 引脚选择 0b = 禁用 ADC 输入 A2 1b = 使能 ADC 输入 A2
1	ADCPCTL1	RW	0h	ADC 输入 A1 引脚选择 0b = 禁用 ADC 输入 A1 1b = 使能 ADC 输入 A1
0	ADCPCTL0	RW	0h	ADC 输入 A0 引脚选择 0b = 禁用 ADC 输入 A0 1b = 使能 ADC 输入 A0

电源管理模块 (PMM) 和电源电压监控器 (SVS)

本章对电源管理模块 (PMM) 和电源电压监控器 (SVS) 的操作进行了说明。

Topic **Page**

2.1 电源管理模块 (PMM) 介绍

PMM 的特点包括：

- 宽电源电压 (DV_{CC}) 范围：1.8V 至 3.6V
- 为器件内核生成电压 (V_{CORE})
- 适用于 DV_{CC} 的电源电压监控器 (SVS)
- 欠压复位 (BOR)
- 软件可访问的电源故障指示灯
- 在电源故障条件期间的 I/O 保护
- 外部引脚上的基准电压输出

PMM 管理与器件的电源及其监控有关的所有功能。主要有两项功能，即为内核逻辑生成电源电压，以及提供若干机制来监控施加到器件的电压 (DV_{CC})。

PMM 使用一个集成低压降稳压器 (LDO) 生成一个二次内核电压 (V_{CORE})，即通过施加到器件的一次内核电压 (DV_{CC}) 来生成。通常，V_{CORE} 为 CPU、存储器和数字模块供电，而 DV_{CC} 为 I/O 和模拟模块供电。通过专用基准电压来维持 V_{CORE} 输出。本章节中调节器的输入或一次侧指的是其高电平侧。本章节中调节器的输出或二次侧指的是其低电平侧。

在图 2-1中给出了 PMM 的框图。

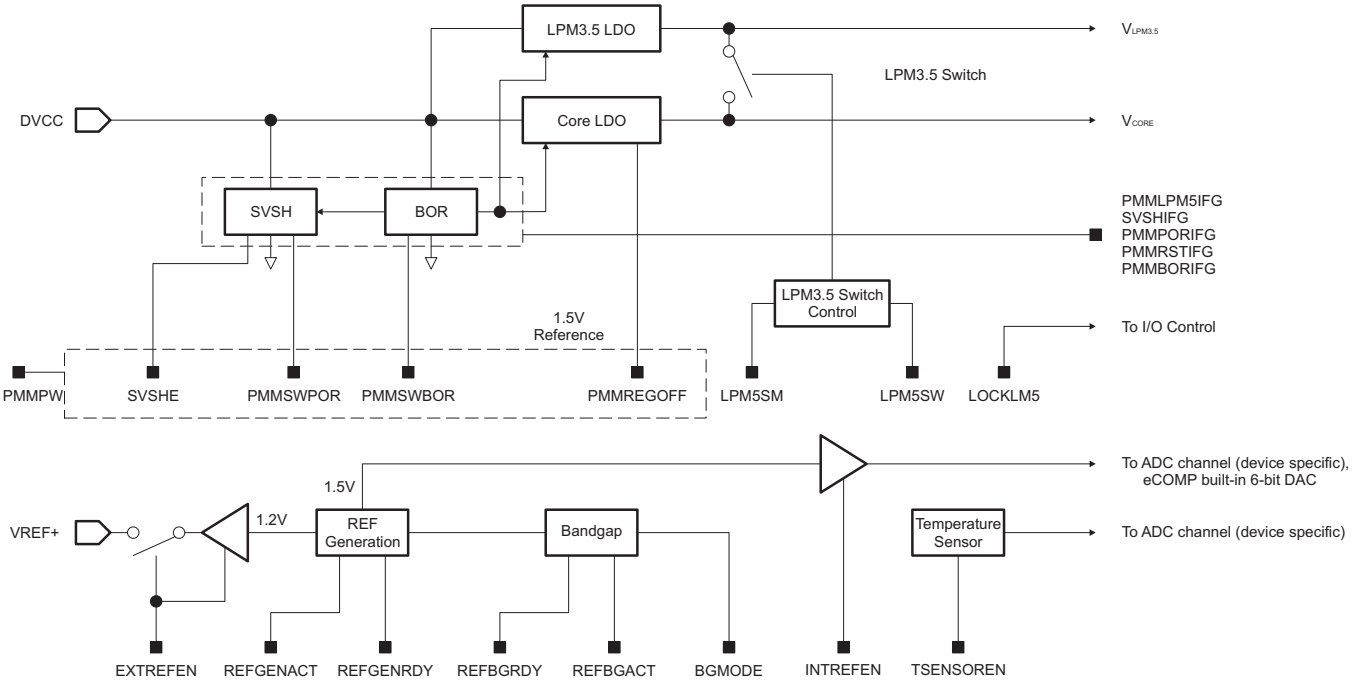


图 2-1. PMM 框图

2.2 PMM 的运行

2.2.1 $V_{内核}$和调节器

DV_{CC}可用一个宽输入电压范围来供电，但器件的芯片逻辑须保持在一个低于该范围允许的电压处。因此，在 PMM 中集成了一个稳压器 (LDO)。该稳压器将所需的内核电压 (V_{CORE}) 从 DV_{CC} 中派生出来。

稳压器支持不同的负载设置以优化功耗。硬件可根据以下标准自动控制负载设置：

- 所选和激活的功耗模式
- 所选和激活的时钟
- 取决于时钟系统 (CS) 设置的时钟频率
- JTAG 或 SBW 激活

除了主 LDO，超低功耗稳压器 (RTC LDO) 可为在主 LDO 关闭时，LPM3.5 期间保持激活状态的实时时钟模块（包括 32kHz 晶振）和其他超低功耗模块提供稳定电压。

2.2.2 电源电压监控器

高侧监控器 (SVSH) 监视 DV_{CC}。其在所有的功耗模式中默认激活。在 LPM3、LPM4、LPM3.5 和 LPM4.5 中，设置 SVSHE = 0 可使其禁用。

2.2.2.1 SVS 阈值

如图 2-2 所示，监控阈值中内置了滞后，因此有效的阈值取决于电压轨是上升还是下降。

这些阈值与 SVS 的特性可通过图形实现最佳呈现。图 2-2 给出了监控器对各种电源故障情况的响应。

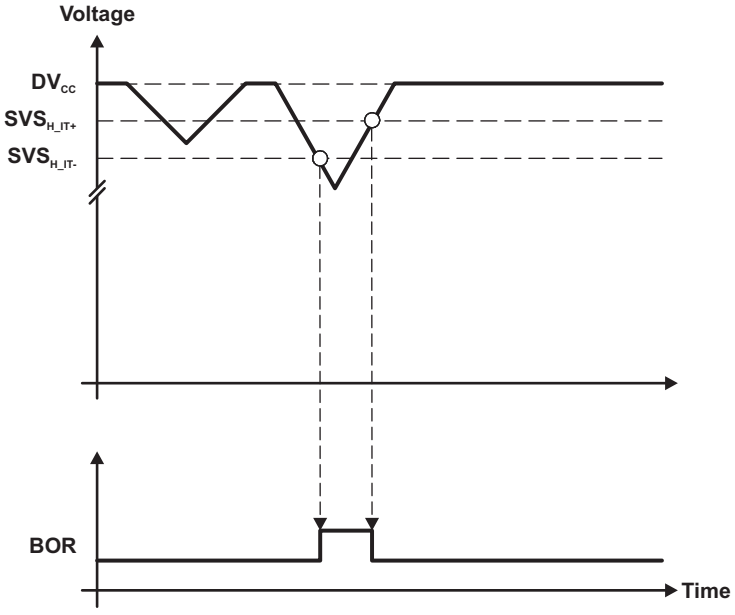


图 **2-2.** 电压故障以及伴随的 **PMM** 动作

2.2.3 上电期间的电源电压监控器

器件上电时，默认使能 SVSH 功能。开始时，DV_{CC} 较低，因此 PMM 使器件保持在 BOR 复位状态。达到 SVSH 电平时，释放复位。图 2-3展示了该过程。

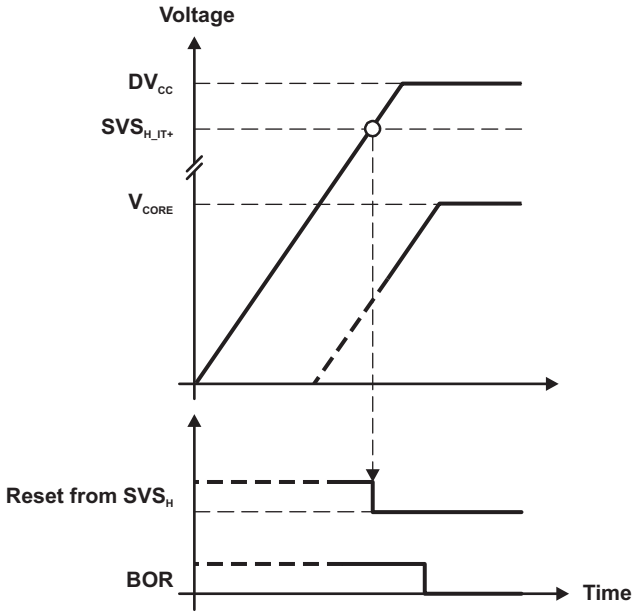


图 **2-3.** 器件上电时的 **PMM** 的动作

2.2.4 LPM3.5 和 LPM4.5 (LPMx.5)

LPM3.5 和 LPM4.5 为低功耗模式，该模式中的 PMM 内核稳压器完全被禁用以实现额外的节能。因为在 LPMx.5 中不为 V_{CORE} 供电，所以 CPU 和所有的数字模块（包括 RAM）将断电。这基本上禁用了整个器件，寄存器和 RAM 中的内容因此会丢失。在进入 LPMx.5 前，应将所有重要的值存储到 FRAM 中。有关 LPMx.5 的完整说明和使用，请参见 SYS 模块。

LPM3.5 和 LPM4.5 可配置成使能 SVS (SVSHE = 1) 或禁用 SVS (SVSHE = 0)。禁用 SVS 可实现更低的功耗，而使能 SVS 则可提供电源压降检测功能，并可实现在电源电压降至低于 SVS 阈值时“唤醒”设备。注意，因电源故障导致的“唤醒”不会标记为 LPMx.5 唤醒，而是标记为 SVS 复位事件。在 LPM4.5 中，使能 SVS 还可实现比禁用 SVS 快 10 倍的启动时间。

2.2.5 低功耗复位

在电池供电的应用中，可能希望在电源电压降至低于 SVS 断电电平后，最大程度限制设备的电流消耗。默认情况下，只要电源电压降至低于 SVS 断电电平，整个设备便复位并准备在电源电压再次可用时快速返回激活模式。此状态大约会产生 50 µA 到 100 µA（典型值）的电流消耗。

在 LPM4.5 低功耗复位状态中上拉复位引脚将导致设备进入默认复位状态（更高的电流消耗），且设备会在电源电压升到 SVS 上电电平以上时启动。

如果设备在电源电压降至 SVS 阈值以下前已处于 LPMx.5（使能 SVS）中，则设备将自动进入低功耗复位状态（即，设备进入 LPM4.5 状态，同时禁用 SVS、RTC 域和所有唤醒事件）。（在 LPMx.5 中，I/O 已处于已定义的状态。因此，无需 NMI 处理来定义 I/O 状态。）

2.2.6 欠压复位 (BOR)

BOR 电路的主要功能在设备上电时发挥作用。该功能在上电斜升早期便开始发挥作用，生成初始化系统的 BOR。该功能还将在 SVS 未使能和欠压条件出现时起作用。它将保持此复位状态，直到输入功率符合逻辑需要及正确复位系统时为止。

在应用中，可能希望通过软件实现 BOR。设置 PMMSWBOR 会导致一个软件驱动的 BOR。据此，PMMBORIFG 被置 1。需要注意的是，一个 BOR 也会启动一个 POR 和 PUC。可以由软件或通过读取 SYSRSTIV 来清零 PMMBORIFG。

同样，设置 PMMSWPOR 可通过软件实现 POR。据此，PMMPORIFG 被置 1。一个 POR 也会启动一个 PUC。可以由软件或通过读取 SYSRSTIV 来清零 PMMPORIFG。PMMSWBOR 和 PMMSWPOR 均是自清零。有关 BOR，POR，PUC 复位的完整说明，请参阅 SYS 模块。

2.2.7 LPM3.5 开关

LPM3.5 开关使用主 LDO 输出为 LPM3.5 电源域供电，从而允许外设消耗更多电流以在高频率下运行。当设备进入 LPM3.5，LPM3.5 域中的所有外设将从内核域隔离，并完全由 LPM3.5 LDO 供电。LPM3.5 开关既可手动控制也可自动控制。LPM3.5 开关模式可由 PM5CTL0 寄存器中的 LPM5SM 设置。

在自动控制模式下（LPM5SM 位清零），LPM3.5 开关在设备进入 LPM3.5 模式时断开。在从 LPM3.5 退出进入激活模式 (AM) 时，设备自动接通 LPM3.5 开关，且之前由 LPM3.5 供电的外设将直接由主 LDO 供电。在电源切换完成前，请勿以高频率读/写这些外设的寄存器。PM5CTL0 寄存器中的 LPM5SW 位报告 LPM3.5 开关的状态，并允许软件在高频工作前查看 LPM3.5 开关的连接。在该模式中，无法写入 LPM5SW 位。

在手动控制模式中（LPM5SM 位置 1），LPM3.5 开关由 PM5CTL0 寄存器中的 LPM5SW 位指定。在 LPM5SW 置 1 时连接 LPM3.5 开关。在 LPM5SW 清零时断开 LPM3.5 开关。建议在设备进入 LPM3.5 前关闭开关以避免漏电。当设备从 LPM3.5 模式中恢复时，应该接通开关以为高频工作提供足够的电流。

LPM5SW 默认为逻辑 1，也就是说 LPM3.5 开关在 BOR、POR 或 PUC 复位后始终处于连接状态。

2.2.8 基准电压生成和输出

PMM 模块具有一个高精度带隙，可用于片上的各种电压基准。该带隙会根据工作模式自动开关。PMMCTL2 寄存器中的 REFBGRDY 位报告带隙的就绪情况。在 REFBGRDY 置 1 时，带隙基准就绪，可以使用。

分别生成两个电压基准供内部 (1.5 V) 和外部 (1.2 V) 使用。电压发生器由响应电压基准请求（内部或是外部）的器件自动控制。如果输出在指定电压下正常工作，REFGENACT 和 REFGENRDY 位可表示发生器状态。

内部基准电压 (1.5V) 从内部连接到 ADC 通道（请参见具体器件的数据表配置）。PMMCTL2 中的 INTREFEN 位可控制是否将 1.5V 电压注入指定的 ADC 通道。

外部基准电压 (1.2V) 连接到给定的外部 ADC 通道（请参见具体器件的数据表配置）。如果此 ADC 通道与其他功能复用，则只有选择 ADC 作为此引脚上的功能时，1.2V 输出功能才有效。PMMCTL2 中的 EXTREFEN 位控制是否为外部 ADC 通道提供 1.2V 电压。外部基准电压支持最高 1mA 的驱动能力。

2.2.9 温度传感器

PMM 包含一个内置温度传感器，软件可利用该传感器来监测芯片温度，以实现高温环境下的故障防护。温度传感器从内部连接到 ADC 通道。该连接取决于具体器件，可在数据表的 ADC 部分查看。PMMCTL2 寄存器中的 TSENSOREN 位必须置 1 才能在使用传感器前将其接通。制造过程中温度会调整为 25°C。因此，测得的任何温度均可通过公式 10 来计算。

$$T = 0.00355 \times (V_T - V_{25ºC}) + 25ºC \quad (10)$$

2.2.10 $\overline{\text{RST}}$/NMI

外部 $\overline{\text{RST}}$/NMI 引脚在 BOR 复位条件下拉低。$\overline{\text{RST}}$/NMI 可作为其余应用的复位源使用。

2.2.11 PMM 中断

PMM 产生的中断标志被路由到系 统NMI 中断向量发生器寄存器，SYSSNIV。当 PMM 导致一个复位时，在系统复位中断向量发生器寄存器 SYSRSTIV 中就会生成一个值，该值对应于复位源。在 SYS 模块给出了这些寄存器的定义。有关 PMM 和 SYS 模块之间的关系的更多信息可在 SYS 章节找到。

2.2.12 端口 I/O 控制

PMM 可确保在出现欠压事件期间，I/O 引脚的行为不会不受控制。在此期间会禁用输出，包括常规驱动以及弱上拉和下拉功能。如果欠压事件发生前 CPU 工作正常，则任何配置为输入的引脚会在欠压事件发生时，锁存 PxIN 寄存器值，直到电压恢复正常。在欠压事件发生期间，内部不记录引脚上的外部电压变化。这有助于防止不确定行为。

2.3 PMM 寄存器

表 2-1 给出了 PMM 寄存器及其地址偏移量。关于 PMM 模块的基址，请参见具体器件的数据表。

PMMCTL0 寄存器中定义的密码控制对 PM5CTL0 以外的所有 PMM 寄存器的访问。对 PM5CTL0 的访问无需密码。写入正确的密码后，将使能写访问（包括对 PMMCTL0 低字节的字节访问）。向 PMMCTL0 高字节写入字节模式的错误密码，将禁用写访问。使用错误的密码对 PMMCTL0 进行字访问将触发 PUC。在未使能写访问的情况下，对除 PMMCTL0 以外的寄存器进行写访问将导致 PUC。

注: 所有寄存器都支持字或字节访问。对于通用寄存器*ANYREG*来说，后缀"_L"(*ANYREG_L*) 是指寄存器的低字节（位 0 到 7）。后缀为"_H"的*ANYREG_H*是指寄存器的高位字节（位 8 到 15）。

表 2-1. PMM 寄存器

偏移量	首字母缩写词	寄存器名称	类型	访问	复位	部分
00h	PMMCTL0	PMM 控制寄存器 0	读取/写入	字	9640h	2.3.1 节
00h	PMMCTL0_L		读取/写入	字节	40h	
01h	PMMCTL0_H		读取/写入	字节	96h	
02h	PMMCTL1	PMM 控制寄存器 1	读取/写入(1)	字	9600h	2.3.2 节
02h	PMMCTL1_L		读取(1)	字节	00h	
03h	PMMCTL1_H		读取(1)	字节	96h	
04h	PMMCTL2	PMM 控制寄存器 2	读取/写入	字	3200h	2.3.3 节
04h	PMMCTL2_L		读取/写入	字节	00h	
05h	PMMCTL2_H		读取/写入	字节	33h	
0Ah	PMMIFG	PMM 中断标志寄存器	读取/写入	字	0000h	2.3.4 节
0Ah	PMMIFG_L		读取/写入	字节	00h	
0Bh	PMMIFG_H		读取/写入	字节	00h	
10h	PM5CTL0	功耗模式 5 控制寄存器 0	读取/写入	字	0011h	2.3.5 节
10h	PM5CTL0_L		读取/写入	字节	11h	
11h	PM5CTL0_H		读取/写入	字节	00h	

(1) PMMCTL1 只能写为字。

2.3.1 PMMCTL0 寄存器（偏移 = 00h）[复位 = 9640h]

电源管理模块控制寄存器 0

图 2-4. PMMCTL0 寄存器

15	14	13	12	11	10	9	8
PMMPW							
rw - 1	rw - 0	rw - 0	rw - 1	rw - 0	rw - 1	rw - 1	rw - 0
7	6	5	4	3	2	1	0
保留	SVSHE	保留	PMMREGOFF	PMMSWPOR	PMMSWBOR	保留	
rw-[0]	rw-[1]	r0	rw-[0]	rw-(0)	rw-[0]	r0	r0

表 2-2. PMMCTL0 寄存器说明

位	字段	类型	复位	说明
15-8	PMMPW	RW	96h	PMM 密码。始终读为 096h。写入 0A5h 可解锁 PMM 寄存器。
7	保留	RW	0h	保留。必须写为 0。
6	SVSHE	RW	1h	高侧 SVS 使能。 0b = 在 LPM2、LPM3、LPM4、LPM3.5 和 LPM4.5 中禁用高侧 SVS (SVSH)。在激活模式、LPM0 和 LPM1 中使能 SVSH。 1b = 始终使能 SVSH。
5	保留	R	0h	保留。始终读为 0。
4	PMMREGOFF	RW	0h	稳压器关闭 0b = 进入 LPM3 或 LPM4 时稳压器保持打开。 1b = 进入 LPM3 或 LPM4 时稳压器关闭。系统会分别进入 LPM3.5 或 LPM4.5。
3	PMMSWPOR	RW	0h	软件 POR。将此位置 1 可触发 POR。此位可自行清零。 0b = 正常工作 1b = 置 1 可触发 POR
2	PMMSWBOR	RW	0h	软件掉电复位。将此位置 1 可触发 BOR。此位可自行清零。 0b = 正常工作 1b = 置 1 可触发 BOR
1-0	保留	R	0h	保留。始终读为 0。

2.3.2 PMMCTL1 寄存器（偏移 = 02h）[复位 = 0000h]

电源管理模块控制寄存器 1

图 **2-5. PMMCTL1** 寄存器

15	14	13	12	11	10	9	8
保留							
rw - 1	rw - 0	rw - 0	rw - 1	rw - 0	rw - 1	rw - 1	rw - 0
7	6	5	4	3	2	1	0
保留							
rw-[0]	rw-[0]	rw-[0]	rw-[0]	rw-[0]	rw-[0]	rw-[0]	r0

表 **2-3. PMMCTL1** 寄存器说明

位	字段	类型	复位	说明
15-0	保留	R	9600h	保留。始终读为 9600h。

2.3.3 PMMCTL2 寄存器（偏移 = 04h）[复位 = 3200h]

电源管理模块控制寄存器 2

图 2-6. PMMCTL2 寄存器

15	14	13	12	11	10	9	8
保留	保留	REFBGRDY	REFGENRDY	BGMODE	保留	REFBGACT	REFGENACT
r0	r0	r-(1)	r-(1)	r-(0)	r0	r-(1)	r-(0)
7	6	5	4	3	2	1	0
保留	保留	保留	保留	TSENSOREN	保留	EXTREFEN	INTREFEN
r0	r0	r0	r0	rw-(0)	r0	rw-(0)	rw-(0)

表 2-4. PMMCTL2 寄存器说明

位	字段	类型	复位	说明
15-14	保留	R	0h	保留。始终读为 0。
13	REFBGRDY	R	1h	缓冲带隙电压就绪状态。 0b = 缓冲带隙电压未就绪，不可使用。 1b = 缓冲带隙电压就绪，可以使用。
12	REFGENRDY	R	1h	可变基准电压就绪状态。 0b = 基准电压输出未就绪，不可使用。 1b = 基准电压输出就绪，可以使用。
11	BGMODE	R	0h	带隙模式。仅就绪。 0b = 静态模式（精度更高） 1b = 采样模式（功耗更低）
10	保留	R	0h	保留。始终读为 0。
9	REFBGACT	R	1h	基准带隙激活。仅就绪。 0b = 基准带隙缓冲器未激活 1b = 基准带隙缓冲器激活
8	REFGENACT	R	0h	基准发生器激活。只读。 0b = 基准发生器未激活 1b = 基准发生器激活
7-4	保留	R	0h	保留。始终读为 0。
3	TSENSOREN	RW	0h	温度传感器使能 0b = 禁用温度传感器 1b = 使能温度传感器
2	保留	R	0h	保留。始终读为 0。
1	EXTREFEN	RW	0h	外部基准输出使能 0b = 禁用外部基准输出 1b = 使能内部基准输出
0	INTREFEN	RW	0h	内部基准使能 0b = 禁用内部基准 1b = 使能内部基准

2.3.4 PMMIFG 寄存器（偏移 = 0Ah）[复位 = 0000h]

电源管理模块中断标志寄存器

图 2-7. PMMIFG 寄存器

15	14	13	12	11	10	9	8
PMMLPM5IFG	保留	SVSHIFG	保留		PMMPORIFG	PMMRSTIFG	PMMBORIFG
rw-{0}	r0	rw-{0}	r0	r0	rw-[0]	rw-{0}	rw-{0}
7	6	5	4	3	2	1	0
保留							
r0	r0	r0	r0	r0	r0	r0	r0

表 2-5. PMMIFG 寄存器说明

位	字段	类型	复位	说明
15	PMMLPM5IFG	RW	0h	LPMx.5 标志。 此位具有特定的复位条件。仅当系统在复位前处于 LPMx.5 中，此位才可置 1。 此位可通过软件或读取复位向量字进行清零。高侧 SVS（如果使能）或欠压触发 DVCC 域上的电源故障时，会将此位清零。 0b = 复位不是因从 LPMx.5 中唤醒所致 1b = 复位是因从 LPMx.5 中唤醒所致
14	保留	R	0h	保留。始终读为 0。
13	SVSHIFG	RW	0h	高侧 SVS 中断标志。 此位具有特定的复位条件。 SVSHIFG 中断标志仅在 SVSH 为复位源时置 1；也就是说，DVCC 降至高侧 SVS 电平以下，但仍高于欠压电平。此位可通过软件或读取复位向量字 SYSRSTIV 进行清零。 0b = 复位不是因 SVSH 所致 1b = 复位是因 SVSH 所致
12-11	保留	R	0h	保留。始终读为 0。
10	PMMPORIFG	RW	0h	PMM 软件 POR 中断标志。 此位具有特定的复位条件。此中断标志只有在触发软件 POR (PMMSWPOR) 后置 1。 此位可通过软件或读取复位向量字进行清零。 0b = 复位不是因 PMMSWPOR 所致 1b = 复位是因 PMMSWPOR 所致
9	PMMRSTIFG	RW	0h	PMM 复位引脚中断标志。 此位具有特定的复位条件。此中断标志只有在 $\overline{\text{RST}}$/NMI 引脚为复位源时置 1。 此位可通过软件或读取复位向量字进行清零。 0b = 复位不是因复位引脚所致 1b = 复位是因复位引脚所致
8	PMMBORIFG	RW	0h	PMM 软件掉电复位中断标志。 此位具有特定的复位条件。此中断标志只有在触发软件 BOR (PMMSWBOR) 后置 1。 此位可通过软件或读取复位向量字进行清零。 0b = 复位不是因 PMMSWBOR 所致 1b = 复位是因 PMMSWBOR 所致
7-0	保留	R	0h	保留。始终读为 0。

2.3.5 PM5CTL0 寄存器（偏移 = 10h）[复位 = 0011h]

功耗模式 5 控制寄存器 0

图 **2-8. PM5CTL0** 寄存器

15	14	13	12	11	10	9	8
保留							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
保留		LPM5SM	LPM5SW	保留			LOCKLPM5
r0	r0	rw-[0]	rw-[1]	r0	r0	r0	rw-[1]

表 **2-6. PM5CTL0** 寄存器说明

位	字段	类型	复位	说明
15-6	保留	R	0h	保留。始终读为 0。
5	LPM5SM	RW	0h	指定 LPM3.5 开关的工作模式。 0b = 自动模式。在模式切换期间完全通过电路处理 LPM3.5 开关。 1b = 手动模式。通过软件中的 LPM5SW 位指定 LPM3.5 开关。
4	LPM5SW	RW	1h	根据 LPM5SM 设置的开关模式报告或设置 LPM3.5 开关连接。若 LPM5SW = 1，$V_{LPM3.5}$ 域可接受 CPU MCLK 的全速读/写操作。如果开关断开 (LPM5SW = 0)，则此域内的所有外设可接受速度不超过 40 kHz 的时钟操作。 在自动模式下 (LPM5SM = 0)，该位代表 V_{core} 和 $V_{LPM3.5}$ 之间的开关连接。写入此位的操作均无效。 在手动模式下 (LPM5SM = 1)，可通过软件读/写该位。若该位置 1，将接通 V_{core} 和 $V_{LPM3.5}$ 之间的开关连接。否则，开关断开。 0b = 断开 LPMx.5 开关 1b = 连接 LPMx.5 开关
3-1	保留	R	0h	保留。始终读为 0。
0	LOCKLPM5	RW	1h	在进入或退出 LPMx.5 时，可锁定 I/O 引脚和其他 LPMx.5 相关（例如 RTC）配置。LOCKLPM5 位置 1 后，只能通过软件或重启清零。 此位可通过重启复位；亦即，在 SVSH（如果使能）或欠压触发了复位的情况下。 0b = LPMx.5 配置未锁定且默认为其复位条件。 1b = LPMx.5 配置保持锁定状态。在 LPMx.5 进入和退出期间保持引脚状态。

时钟系统 (CS)

时钟系统 (CS) 模块提供 MCU 使用的各种时钟。本章对所有器件中采用的 CS 模块的操作进行了说明。

Topic **Page**

3.1 CS 简介

CS 模块可降低系统成本并降低功耗。该模块支持四个内部时钟源和两个外部时钟源，用户可通过这些时钟源对不同的设计目标优化时钟配置。同一器件中不会兼具所有时钟源。有关指定器件配置的详细说明，请参见具体器件的数据表。所有时钟源均可通过软件选择。外部时钟源可使用晶振或陶瓷振荡器或者谐振器。

CS 模块包含多达六个时钟源：

- XT1CLK：高频或低频振荡器，可与高频陶瓷振荡器、高频晶振或者低频 32768Hz 晶振搭配使用。XT1CLK 可作为进入 FLL 的时钟基准。一些器件仅支持 XT1CLK 使用低频振荡器。更多详细信息，请参见具体器件的数据表。
- VLOCLK：频率典型值为 10kHz 的内部超低功耗低频振荡器
- REFOCLK：频率典型值为 32768Hz 的内部修整低频振荡器。可作为进入 FLL 的时钟基准。
- DCOCLK：内部数控振荡器 (DCO)，可使用 FLL 使其保持稳定。
- MODCLK：频率典型值为 5MHz 的内部高频振荡器。

CS 模块提供三个时钟信号：

- ACLK：辅助时钟。ACLK 可用于外设低频运行。此时钟可通过软件选择作为 XT1CLK 或 REFOCLK。所选时钟源的频率必须始终保持大约 32kHz，最高不得超过 40kHz（典型值）。ACLK 可由独立外设模块通过软件进行选择。
- MCLK：主时钟。MCLK 可作为下列设备的主时钟源：CPU、CRC 以及一些直接由 CPU 或其时钟操作的外设。该时钟可通过软件选择作为 REFOCLK、DCOCLK、XT1CLK 或 VLOCLK。当所选时钟源可用时，可对其进行 1、2、4、8、16、32、64 或 128 预分频。
- SMCLK：子系统主时钟。SMCLK 是独立于 CPU 工作的外设时钟。该时钟始终从 MCLK 获取。当 SMCLK 可用时，可对其进行 1、2、4 或 8 分频。SMCLK 可由独立外设模块通过软件进行选择。

图 3-1 显示了 CS 模块的框图。

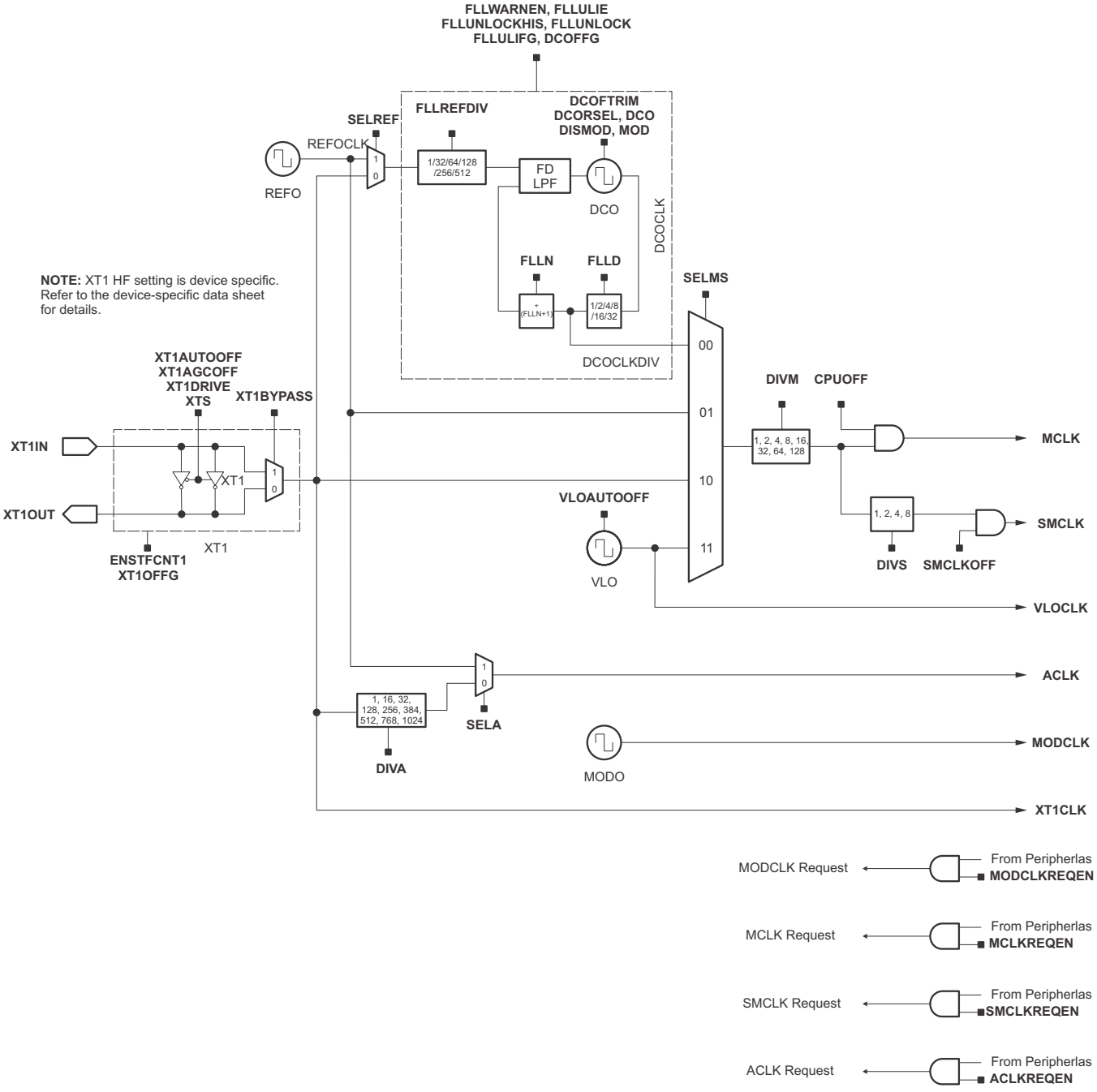


图 **3-1.** 时钟系统 **(CS)** 框图

3.2 CS 运行

经过 PUC 后，CS 模块的默认配置为：

- MCLK 和 SMCLK 使用 DCOCLKDIV，在不提供 XT1 的情况下，DCOCLKDIV 由 FLL 锁定并作为 REFO 的基准。
- ACLK 使用 REFO。
- XT1 外部晶振被选为 XT1CLK 时钟源。在配置 I/O 端口进行 XT1 操作前，XT1IN 和 XT1OUT 引脚设为通用 I/O 并且 XT1 保持禁用状态。

经过 PUC 后，DCO 由 FLL 操作锁定，同时默认选择 XT1CLK。FLL 将 MCLK 和 SMCLK 的频率稳定为 1MHz 并且 $f_{DCOCLKDIV}$ = 1MHz

外部 32768Hz 晶振可用作 FLL 基准。在默认情况下，晶振引脚（XT1IN 和 XT1OUT）与通用 I/O 共用。要使能 XT1，与晶振引脚关联的 PSEL 位必须置 1，从而将外部 32768Hz 晶振作为时钟源。晶振启动并稳定下来后，FLL 基准时钟在 XT1OFFG、DCOFFG 和 OFIFG 清零时会自动切换为 XT1CLK。

模块在运行过程中默认监视 XT1 振荡。如果使用的 XT1 无法正常工作，故障保护逻辑强制将 REFO 作为 FLL 基准时钟。

状态寄存器控制位（SCG0、SCG1、OSCOFF 和 CPUOFF）配置 MSP430 的工作模式，并且可使能或禁用 CS 模块的各部分。寄存器 CSCTL0 至 CSCTL8 配置 CS 模块。

在程序运行过程中，CS 模块可随时通过软件进行配置或重新配置。

3.2.1 面向低功耗应用的 CS 模块功能

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
- 高时钟频率，以获得快速响应时间和快速触发脉冲处理功能
- 运行温度和电源电压上的时钟稳定
- 时钟精度要求限制较小的低成本应用

CS 模块允许用户在提供的三种时钟信号 ACLK、MCLK 和 SMCLK 中进行选择，能够妥善处理这些互相冲突的要求。

MCLK 可选用任一可用的时钟源（DCOCLK、REFOCLK、XT1CLK 或 VLOCLK）。SMCLK 来源于 MCLK，始终使用与 MCLK 相同的时钟源。

ACLK 可通过 REFO 或 XT1CLK 提供时钟源。

3.2.2 内部超低低功率低频振荡器 (VLO)

内部 VLO 可在不使用晶振的情况下提供典型值为 10kHz 的频率（相关参数请参见具体器件的数据表）。VLO 可为不要求精确时基的应用提供低成本低功耗时钟源。

VLOCLK 在下列情况下激活：

- VLO 选为 MCLK 和 SMCLK (SELMS = {3}) 的时钟源并且 MCLK 或 SMCLK 处于激活状态。
- VLOAUTOOFF 位清零并且 MCU 处于从 AM 到 LPM4 的转换过程中。
- 至少一个外设请求将 VLO 作为时钟源。

3.2.3 内部修整低频基准振荡器 (REFO)

内部修整低频 REFO 可用于不要求或无需晶振的成本敏感型应用。REFO 在内部修整为 32.768kHz（典型值）并提供可作为 FLLREFCLK 的稳定基准频率。REFO 与 FLL 相结合，可在无需晶振的情况下提供灵活的系统时钟设置范围。如果不使用 REFO，则其不产生功耗。

以下条件下均可使能 REFO：

- REFO 为 MCLK 和 SMCLK 的时钟源 (SELMS = {1}) 并且 MCLK 或 SMCLK 处于激活状态。
- REFO 为 ACLK 的时钟源 (SELA = {1}) 并且 ACLK 处于激活状态。
- REFO 为 FLLREFCLK 的时钟源 (SELREF = {1}) 并且 DCO 处于激活状态。

3.2.4 XT1 振荡器

XT1 振荡器可通过 32768Hz 手表晶振在低频 (LF) 模式下以低流耗运行。手表晶振与 XIN 和 XOUT 相连，并且需要在这两个引脚上连接外部电容。这些电容的大小取决于晶振或谐振器的规格。

XT1 的驱动设置能够随着 XT1DRIVE 位而提升。在上电过程中，XT1 以最高驱动设置启动，以实现快速可靠的启动。启动后，用户软件可通过降低驱动强度来降低功耗。

当一些器件处于 (HF) 模式 (XTS = 1) 时，XT1 振荡器支持高频晶振或谐振器。高频晶振或谐振器与 XT1IN 和 XT1OUT 相连，这两个引脚需连接外部电容。这些电容的大小取决于晶振或谐振器的规格。

XT1 引脚与通用 I/O 端口共用。在上电过程中，XT1 默认用作通用 I/O 端口。在与 XT1 共用的端口配置为进行 XT1 操作前，XT1 保持禁用状态。共用 I/O 的配置取决于与 XT1IN 引脚相关的 Px.SEL 位以及 XT1BYPASS 位。Px.SEL 位置 1 会导致 XT1IN 和 XT1OUT 端口配置为进行 XT1 操作。

如果 XT1BYPASS 同时置 1，XT1 将配置为旁路运行模式，并且与 XT1 关联的振荡器也会掉电。在旁路运行模式下，XT1IN 可接收外部时钟输入信号并且 XT1OUT 配置为通用 I/O。不必考虑与 XT1OUT 关联的 Px.SEL 位。

如果与 XT1IN 关联的 Px.SEL 位清零，XT1IN 和 XT1OUT 端口配置为通用 I/O 并且 XT1 禁用。

以下条件下均可使能 XT1：

- XT1 为 MCLK 和 SMCLK 的时钟源 (SELMS = {2}) 并且 MCLK 或 SMCLK 处于激活状态。
- XT1 为 ACLK 的时钟源 (SELA = {0}) 并且 ACLK 处于激活状态。
- XT1 为 FLLREFCLK 的时钟源 (SELREF = {0}) 并且 DCO 处于激活状态。
- XT1AUTOOFF 位清零并且 MCU 处于从 AM 到 LPM4 的转换过程中。
- 至少一个外设请求将 XT1 作为时钟源。

注: 采用 HF 模式配置的 XT1

ACLK 为辅助时钟。ACLK 的频率必须为约 32kHz 并且不得高于 40kHz（典型值）。如果 ACLK 源于 HF 模式下的 XT1，则需采用分频器 (DIVA)。分频器设置取决于外部高频振荡器的值。

如果 ACLK 源于 LF 模式下的 XT1，该分频器始终被旁路。

3.2.5 数控振荡器 (DCO)

DCO 是一个内置的数字控制振荡器。DCO 频率可由软件通过使用 DCORSEL，DCO，和 MOD 位来调整。可使用 FLL 选择将 DCO 频率稳定为 FLLREFCLK÷n 的倍频。FLL 可接受通过 SELREF 位选择的不同基准源。基准源包括 XT1CLK 和 REFOCLK。n 的值由 FLLREFDIV 位确定（n = 1、32、64、128、256 或 512）。当 XT1 仅支持 32kHz 时钟时，FLLREFDIV 始终读/写为 0 (n = 1)。默认 n = 1。某些情况下不要求或无需运行 FLL，因此 FLLREFCLK 并非必不可少。

FLLD 位将 FLL 预分频值配置为 1、2、4、8、16、或 32。在默认情况下，FLLD = 1，MCLK 和 SMCLK 由 DCOCLKDIV 提供，提供的时钟频率为 DCOCLK÷2。

分频器 (FLLN + 1) 和 FLLD 的分频器值决定了 DCOCLK 和 DCOCLKDIV 的频率，其中 FLLN > 0。写入 FLLN = 0 会导致分频器置 1。

$f_{DCOCLK} = 2^{FLLD} \times (FLLN + 1) \times (f_{FLLREFCLK} \div n)$

$f_{DCOCLKDIV} = (FLLN + 1) \times (f_{FLLREFCLK} \div n)$

3.2.5.1 调整 DCO 的频率

默认情况下，FLL 操作被使能。可以通过设置 SCG0 或 SCG1 来禁用 FLL 操作。当 FLL 禁用时，DCO 以在 CSCTL0 和 CSCTL1 中定义的当前设置继续运行。想要的话，可以手动调整 DCO 频率。另外，DCO 频率是由 FLL 操作进行稳定的。

经过 PUC 后，DCORSEL = {1} 并且 DCO = {0}。MCLK 和 SMCLK 都以 DCOCLKDIV 作为时钟源。由于 CPU 执行源于 MCLK（以快速启动的 DCO 作为时钟源）的代码，因此代码在不到 5µs 的时间内从 PUC 开始执行。

COCLK 的频率根据以下功能设定：

- 3 个 DCORSEL 位为 DCO 选择 8 个标称频率范围中的一个。具体器件的数据表确定了器件各自的对应范围。
- 九个 DCO 位将 DCORSEL 位选定的 DCO 范围分为 512 个频率步长，相邻间隔约 0.1%。
- 五个 MOD 位可切换 DCO 位选定的频率与 {DCO + 1} 设置的下一较高频率（请参见3.2.7 节）。当 DCO = {511}，MOD 位没有任何作用，因为 DCO 已针对选定的 DCORSEL 范围采用了最高设置。

3.2.6 频率锁相环 (FLL)

FLL 不断的加或减频率积分器。驱动 DCO 的频率积分器的输出可在 CSCTL0 中读取（MOD 和 DCO 位）。

九个积分器位（CSCTL0 位 8 至 0）设置 DCO 频率节拍。DCO 实现了 512 个节拍，各节拍的频率均比前一节拍高约 0.1%。调制器混合两个相邻的 DCO 频率以生成分数节拍。

对于给定的 DCO 偏差范围设置，必须为 DCO 提供足够的时间来选择正常运行所需的合适节拍。n 的值由 FLLREFDIV 位确定（n = 1、32、64、128、256 或 512）。当 XT1 仅支持 32kHz 时钟时，FLLREFDIV 始终读/写为 0 (n = 1)。对于典型的 32768Hz 时钟源，FLLREFDIV 应始终置为 0（即 n = 1）。

3.2.7 DCO 调制器

调制器通过混合两个 DCO 频率 f_{DCO} 和 f_{DCO}+1 生成介于 f_{DCO} 和 f_{DCO}+1 之间的中间有效频率、传递时钟能量和降低电磁干扰 (EMI)。该调制器为 32 个 DCOCLK 时钟周期混合了 f_{DCO}和 f_{DCO}+1 并且是用 MOD 位配置的。当 MOD = {0} 时，调制器关闭。

调制器混频公式为：

$t = (32 - MOD) \times t_{DCO} + MOD \times t_{DCO+1}$

图 3-2说明了调节器操作。

当 FLL 操作被禁用时，调制器设置和 DCO 都将被 FLL 软件控制。如果不想要 FLL 操作，那么可以用软件配置调制器设置和 DCO 控制。

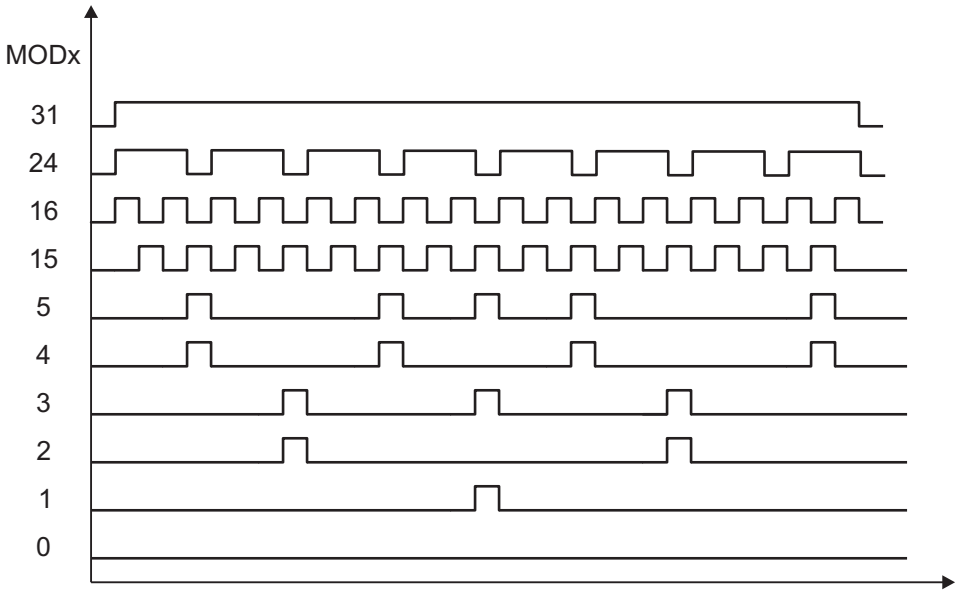


图 **3-2.** 调制器模式

3.2.8 禁用 FLL 硬件和调制器

当状态寄存器 SCG0 或 SCG1 被置 1时，FLL 被禁用。 当 FLL 禁用时，DCO 以之前选定的节拍运行，DCOCLK 无法自动稳定。

当 DISMOD 被置 1 时，DCO 调制器被禁用。当 DCO 调制器被禁用时，DCOCLK 被调整以适应 DCO 位将选择的 DCO 频拍。

注: 无 **FLL** 时的 **DCO** 运行

当 FLL 操作被禁用时，DCO 将在当前的设置上继续运行。因为它不是被 FLL 给稳定的，所以温度和电压的不同都会影响运行的频率。请参阅具体器件的数据表保证稳定运行的电压和温度系数部分。

3.2.9 FLL 解锁检测

当分频后的 DCO 输出无法锁定基准时钟时，FLL 解锁检测功能可能产生 PUC 复位或中断。

当 FLL 使能时，FLLUNLOCK 位反映 DCO 的状态：锁定、过慢、过快或超出 DCO 范围。当 FLL 恢复为锁定状态，FLLUNLOCK 位自动清零并且 FLLUNLOCKHIS 位将自动记录之前的解锁状态。

要重新配置 DCO 频率或 FLL 基准时钟，建议首先将 CSCTL0 清零。此举可确保 DCO 由最低频开始斜升，避免因温度或电源电压漂移导致频率随时间推移超出规范的限制。完成该操作后，必须等待至少两个 MCLK 周期方可重新使能 FLL。等待周期结束后，对 FLLUNLOCK 位进行轮询，确定 FLL 是否锁定在目标频率范围内。如果未在重新配置过程中清零 CSCTL0 寄存器，在轮询 FLLUNLOCK 位前应等待七个 REFCLK 周期。然后，对 FLLUNLOCK 进行轮询，确保 FLL 处于锁定状态。

建议通过以下过程重新配置 FLL：

1. 禁用 FLL (BIS.W #SCG0, SR)

2. 根据需要切换 FLL 基准时钟
3. 清零 CSCTL0 寄存器 (CLR.B CSCTL0)
4. 将 DCO 或 FLL 重新配置为目标范围
5. 至少等待两个 MCLK 周期，直到 DCO 和 FLL 稳定。
6. 重新使能 FLL (BIC.W #SCG0, SR)
7. 对 FLLUNLOCK 位进行轮询，直到 FLL 锁定。

如果 FLLULPUC 位置 1 (FLLULPUC = 1)，当 DCO 运行过快 (FLLUNLOCK = 10b) 时，FLLULIFG 位标志置 1 会导致 PUC 复位。

如果 FLLWARNEN 位置 1，当 FLLUNLOCKHIS 变为解锁状态时，OFIFG 标志置 1。

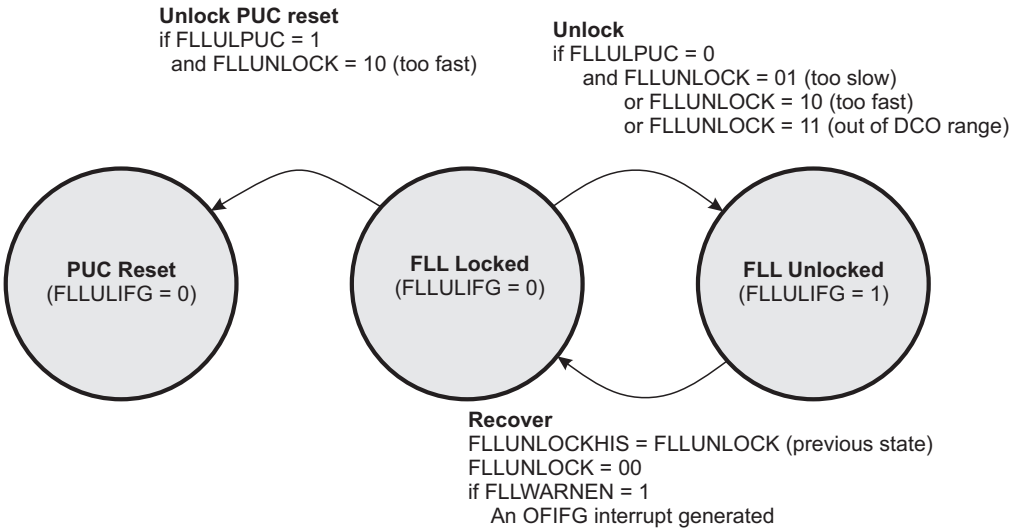


图 **3-3. FLL** 解锁检测

3.2.10 在低功耗模式下 FLL 的操作

如果被置 1，则会有一个中断服务请求清除 SCG1，CPUOFF，OSCOFF ，但没有清除 SCG0。这意味着，在从 LPM1、LPM3 或 LPM4 输入的中断服务程序中运行 FLL 时，它将保持禁用状态，DCO 以之前在 CSCTL0 和 CSCTL1 中定义的设置继续运行。如果需要 FLL 操作时，可通过用户软件清除 SCG0。

3.2.11 外设模块唤醒低功耗模式运行

如果要求外设模块正确运行，其将自动从 CS 模块请求时钟源，与当前运行模式无关（请参见图 3-4）。

外设模块根据其控制位将三种可能的时钟请求信号之一置为有效：ACLK_REQ、MCLK_REQ 或 SMCLK_REQ。这些请求信号基于模块的配置和时钟选择。例如，如果定时器处于使能状态并且选择 ACLK 作为时钟源，该定时器为 CS 系统生成 ACLK_REQ 信号。CS 进而使能 ACLK，无论 LPM 设置如何。

外设模块的任何时钟请求都会导致其相应的时钟关闭信号被覆盖，但不会更改时钟关闭控制位的设置。例如，外设模块可能请求 ACLK，即便其当前由 OSCOFF 位 (OSCOFF = 1) 禁用。该模块通过生成 ACLK_REQ 请求 ACLK。这导致 OSCOFF 位不产生任何作用并使得 ACLK 可用于发出请求的外设模块。OSCOFF 位仍保持当前设置 (OSCOFF = 1)。

如果该请求源未激活，那么软件 NMI 处理器就会执行该请求的行为。对于之前的示例，如果 ACLK 选择 XT1 作为时钟源并且 XT1 处于禁用状态，则振荡器发生故障时软件必须对该事件进行处理。如果原来选择的时钟源不可用，那么出于安全要求考虑，安全装置将选择 VLOCLK 源。

为了省电而进入低功耗模式时，必须在应用中考虑到时钟的请求特征。尽管该器件进入了选定的低功耗模式，但时钟请求产生的流耗可能高于器件数据表中的额定值。

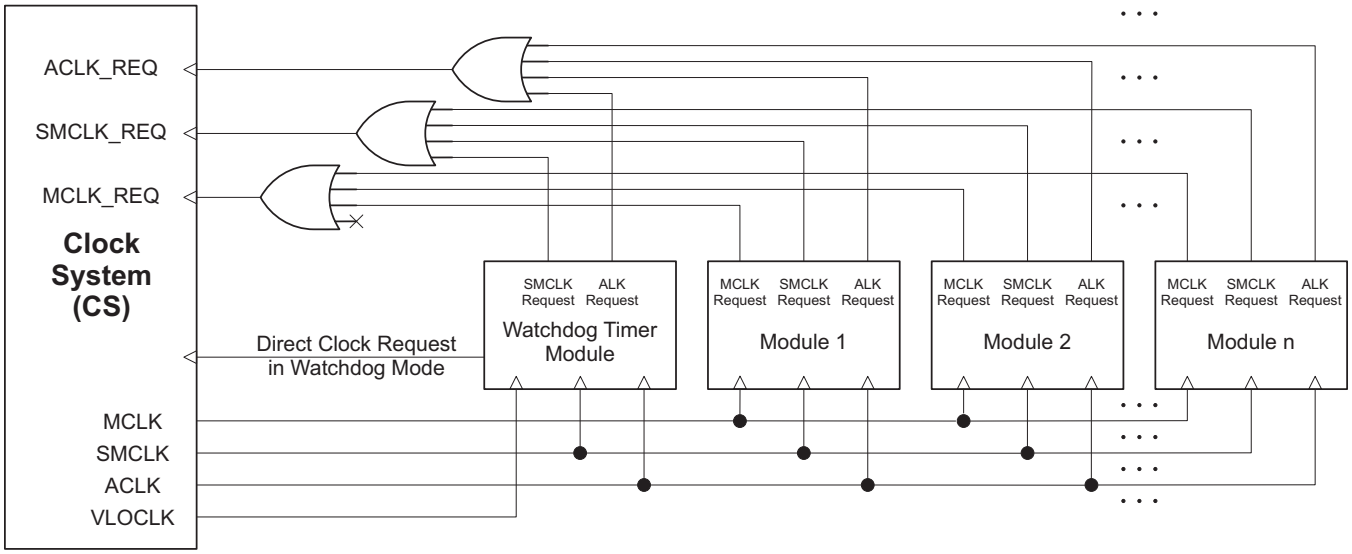


图 **3-4.** 模块请求时钟系统

默认情况下，时钟请求逻辑是使能的。时钟请求逻辑可以通过清除 ACLKREQEN，MCLKREQEN，或 SMCLKREQEN 被禁用，它们分别对应了各自的系统时钟。当 ACLKREQEN 或 MCLKREQEN 被置 1 或激活时，时钟对系统将是可用的并且它就将在所有模块请求的时钟都被禁用前阻止进入低功耗模式。 当 ACLKREQEN 或 MCLKREQEN 位被清零或禁用时，时钟将根据低功耗模式规定的那样一直停用。SMCLKREQEN 逻辑的行为与之相似，但还会受到 CSCTL5 寄存器的 SMCLKOFF 位的影响。表 3-1显示了系统时钟与时钟请求逻辑相关的低功耗模式之间的关系。

表 **3-1.** 时钟请求系统与功耗模式

模式	ACLK		MCLK		SMCLK			
					SMCLKOFF = 0		SMCLKOFF = 1	
	ACLKREQEN = 0	ACLKREQEN = 1	MCLKREQEN = 0	MCLKREQEN = 1	SMCLKREQEN = 0	SMCLKREQEN = 1	SMCLKREQEN = 0	SMCLKREQEN = 1
AM	激活	激活	激活	激活	激活	激活	被禁用	激活
LPM0	激活	激活	被禁用	激活	激活	激活	被禁用	激活
LPM3	激活	激活	被禁用	激活	被禁用	激活	被禁用	激活
LPM3.5	禁用	禁用	禁用	禁用	禁用	禁用	禁用	禁用
LPM4.5	禁用	禁用	禁用	禁用	禁用	禁用	禁用	禁用

3.2.11.1 LPM3.5 和 LPM4.5 时钟请求处理

在清零 ACLK 请求使能信号 (ACLKREQEN = 0) 后，器件即可进入 LPMx.5。有关进入 LPMx.5 的要求详情，请参见 PMM 一章。

3.2.12 故障安全运行

CS 模块具有振荡器故障安全功能。该功能可检测 XT1 和 DCO 的振荡器故障，如图 3-5 所示。可能出现的故障情况包括：

- XT1 的高频或低频振荡器故障 (XT1OFFG)
- DCO 的 DCO 故障标志 (DCOFFG)

如果相应的晶振开启但并未正常运行，晶振故障位 XT1OFFG 置 1。故障位在置 1 后保持不变，直至软件将其复位，即使故障情况不再出现也是如此。如果软件将故障位清零但故障情况仍然存在，故障位将再次自动置 1；否则保持清零状态。

当在 LF 模式下将 XT1 作为 FLL 的基准时钟 (SELREF = {0}) 时，发生晶振故障将自动导致 REFO 选择 FLL 基准时钟源 FLLREFCLK。XT1OFFG 置 1。当在 HF 模式下将 XT1 作为 FLL 基准时钟时，晶振故障导致没有 FLLREFCLK 信号生成，FLL 尝试通过继续递减计数至零来锁定 FLLREFCLK ÷ n 以及 DCOCLK ÷ [2^{FLLD} × (FLLN + 1)]。DCO 节拍移至最低位置（DCO 位清零）并且 DCOFFG 置 1。如果设置的 N 乘法器值对于选定的 DCO 频率范围过大，DCOFFG 也会置 1，导致 DCO 节拍移至最高位置（CSCTL0.8 至 CSCTL0.0 置 1）。在用户清零前，DCOFFG 位将一直保持置 1。如果用户清除了 DCOFFG 并且故障依然存在，那么它将自动置 1，否则保持 0 位。XT1LFOFFG 被置 1。

在 POR 过程中或检测到振荡器故障（XT1OFFG 或 DCOFFG）时，OFIFG 振荡器故障中断标志置 1 并被锁存。当 OFIFG 和 OFIE 被置 1 时，OFIFG 将请求一个 NMI。在中断发生时，OFIE 不会再像原来 MSP430 系列那样被自动复位。 MSP430 系列后期，OFIE 将不再需要复位。NMI 进入/退出电路无需此要求。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。

如果在 LF 模式下选择 XT1 作为 MCLK 的时钟源，振荡器故障导致 MCLK 自动切换至 REFO，以使用其时钟源 (REFOCLK)。如果在 HF 模式下选择 XT1 作为 MCLK 的时钟源，振荡器故障导致 MCLK 自动切换至 DCO，以使用其时钟源 (DCOCLKDIV)。此故障切换不会改变 SELMS 位的设置。这种情况必须由用户软件处理。

如果在 LF 模式下选择 XT1 作为 SMCLK 的时钟源，振荡器故障导致 SMCLK 自动切换至 REFO，以使用其时钟源 (REFOCLK)。如果在 HF 模式下选择 XT1 作为 MCLK 的时钟源，振荡器故障导致 MCLK 自动切换至 DCO，以使用其时钟源 (DCOCLKDIV)。此故障切换不会改变 SELMS 位的设置。这种情况必须由用户软件处理。

如果在 LF 或 HF 模式下选择 XT1 作为 ACLK 的时钟源，振荡器故障导致 ACLK 自动切换至 REFO，以使用其时钟源 (REFOCLK)。这不改变 SELM 位设置。这种情况必须由用户软件处理。

注: 振荡器故障期间活跃的 DCO

在最低 DCO 频拍上，DCOCLKDIV 是活跃时间在振荡器故障期间，时钟信号对 CPU 执行编码和服务 NMI 仍是有效的。

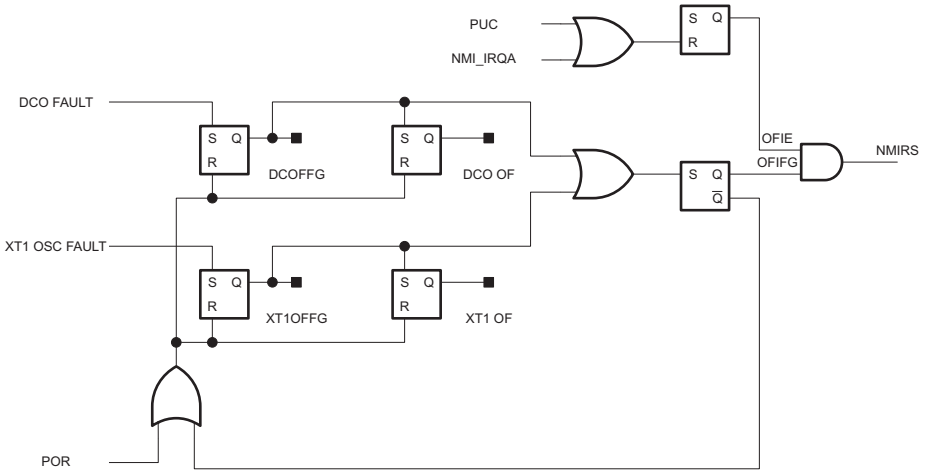


图 **3-5.** 振荡器故障逻辑

注: 故障状态

DCO_Fault：如果 CSCTL0 寄存器 DCO 位的值等于 {0} 或 {511} 并且 DCO 处于解锁状态，DCOFFG 置 1。当 FLL 禁用时，忽略 DCO_Fault。建议在禁用 FLL 前清零 DCOFFG。

XT1_OscFault：该信号在 XT1 振荡器停止运行后置 1；在 XT1 振荡器恢复运行后清零。故障状态导致 XT1OFFG 置 1 并保持该状态。如果在用户清零 XT1OFFG 后依然存在故障，XT1OFFG 保持置 1 状态。

故障逻辑

请注意，只要故障情况依然存在，则 OFIFG 保持置 1 状态。清除 OFIFG 信号时，该应用一定要被特别考虑。如果 OFIFG 信号被清除后故障消失，则时钟逻辑切换回故障发生前用户设置。

故障逻辑计数器

每一个晶振振荡器电路都有一个硬件计数器。该计数器在每一次其对应的振荡器发生故障时都会被复位，从而使得故障标志位置 1。在故障消失后，计数器开始计数。当计数值达到上限时，故障标志删除。

在 XT1 LF 模式下，最大计数值为 8192。在 XT1 HF 模式下，最大计数值为 1024。在旁路模式中不管 LF 或 HF 设置，最大计数都为 8192。

3.2.13 时钟信号的同步

在切换 MCLK 或 SMCLK 时钟源的过程中，为避免发生严重的竞争，切换操作按照图 3-6 给出的方式进行同步。

- 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。

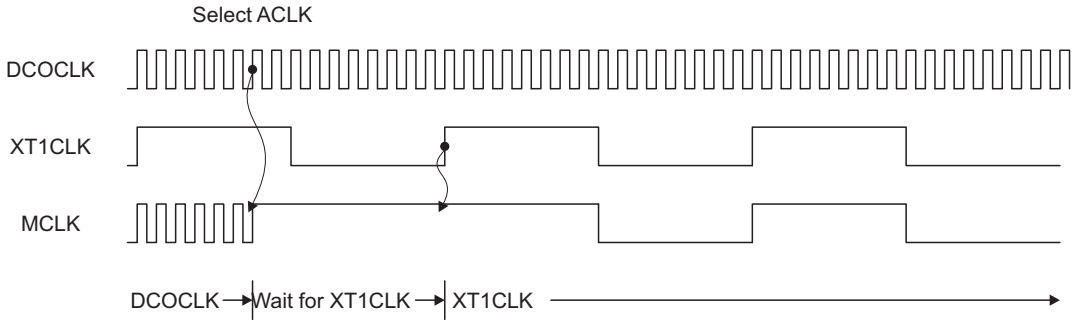


图 **3-6.** 把 **MCLK** 从 **DCOCLK** 切换至 **XT1CLK**

3.2.14 模块振荡器 (MODOSC)

CS 模块还支持内部振荡器 MODOSC，该振荡器可为 ADC 或系统中的其他模块所用。MODOSC 源自 MODCLK。

3.2.14.1 MODOSC 运行

为了节省电能，MODOSC 在不需要时会被关闭并指在需要时启用。需要时，会有相应的模块向 MODOSC 源发出请求。MODOSC 根据无条件和有条件请求被启用。设置 MODOSCREQEN 启用条件请求。无条件请求总是被启用。不必针对使用无条件请求的模块（例如 ADC）将 MODOSCREQEN 置 1。

ADC 可以选择将 MODOSC 作为其转换时钟的时钟源。用户选择将 MODOSC 作为转换时钟源。在转换过程中，ADC 模块发出关于 MODOSC 时钟源的无条件请求。如果之前另一模块的请求尚未使能 MODOSC 时钟源，完成上述操作即可将其使能。

3.3 CS 寄存器

表 3-2 列出了 CS 寄存器及相关偏移量。有关寄存器的基址，请参见具体器件的数据表。

注: 所有寄存器都支持字或字节访问。对于通用寄存器 *ANYREG* 而言，后缀“_L”(*ANYREG_L*) 表示寄存器的低字节（bit 0 到 bit 7）。后缀“_H”(*ANYREG_H*) 表示寄存器的高字节（bit 8 到 bit 15）。

表 3-2. CS 寄存器

偏移量	首字母缩写词	寄存器名称	类型	访问	复位	部分
00h	CSCTL0	时钟系统控制寄存器 0	读取/写入	字	0000h	3.3.1 节
02h	CSCTL1	时钟系统控制寄存器 1	读取/写入	字	0033h	3.3.2 节
04h	CSCTL2	时钟系统控制寄存器 2	读取/写入	字	101Fh	3.3.3 节
06h	CSCTL3	时钟系统控制寄存器 3	读取/写入	字	0000h	3.3.4 节
08h	CSCTL4	时钟系统控制寄存器 4	读取/写入	字	0100h	3.3.5 节
0Ah	CSCTL5	时钟系统控制寄存器 5	读取/写入	字	1000h	3.3.6 节
0Ch	CSCTL6	时钟系统控制寄存器 6	读取/写入	字	08C1h	3.3.7 节
0Eh	CSCTL7	时钟系统控制寄存器 7	读取/写入	字	0740h	3.3.8 节
10h	CSCTL8	时钟系统控制寄存器 8	读取/写入	字	0007h	3.3.9 节

3.3.1 CSCTL0 寄存器

时钟系统控制寄存器 0

图 3-7. CSCTL0 寄存器

15	14	13	12	11	10	9	8
保留		MOD					DCO
r0	r0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0
7	6	5	4	3	2	1	0
DCO							
rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0

表 3-3. CSCTL0 寄存器说明

位	字段	类型	复位	说明
15-14	保留	R	0h	保留。始终读为 0。
13-9	MOD	RW	0h	调制位计数器。这些位选择调制模式。所有 MOD 引脚在 FLL 运行期间被自动修改。当调制位计数器从 31 滚动回 0 时，DCO 寄存器的值如果调制位计数器由 0 递增至计数上限值，DCO 寄存器的值减小。
8-0	DCO	RW	0h	数控振荡器 (DCO) 抽头选择。这些位选择 DCO 抽头并在 FLL 运行期间被自动修改。

3.3.2 CSCTL1 寄存器

时钟系统控制寄存器 1

图 3-8. CSCTL1 寄存器

15	14	13	12	11	10	9	8
保留							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
DCOFTRIMEN	DCOFTRIM			DCORSEL			DISMOD
rw-[0]	rw - 0	rw - 1	rw - 1	rw - 0	rw - 0	rw - 1	rw-1

表 3-4. CSCTL1 寄存器说明

位	字段	类型	复位	说明
15-8	保留	R	0h	保留。始终读为 0。
7	DCOFTRIMEN	RW	0h	DCO 频率修整使能。当该位置 1 时，选择 DCOFTRIM 值设置 DCO 频率。否则，DCOFTRIM 值被旁路，DCO 在生产中应用默认设置。 0b = 禁用频率修整 1b = 使能频率修整
6-4	DCOFTRIM	RW	3h	DCO 频率修整。这些位可对 DCO 频率进行修整。在默认情况下，该过程针对具体芯片进行修整。这些位也可通过用户代码进行修整。
3-1	DCORSEL	RW	1h	DCO 范围选择 000b = 1MHz 001b = 2MHz（默认） 010b = 4MHz 011b = 8MHz 100b = 12MHz 101b = 16MHz 110b = 保留 111b = 保留
0	DISMOD	RW	1h	调制。该位使能/禁用调制功能。 0b = 调制被启用 1b = 调制被禁用

3.3.3 CSCTL2 寄存器

时钟系统控制寄存器 2

图 3-9. CSCTL2 寄存器

15	14	13	12	11	10	9	8
保留	FLLD			保留		FLLN	
r0	rw - 0	rw - 0	rw - 1	r0	r0	rw - 0	rw - 0
7	6	5	4	3	2	1	0
FLLN							
rw - 0	rw - 0	rw - 0	rw - 1	rw-1	rw-1	rw-1	rw-1

表 3-5. CSCTL2 寄存器说明

位	字段	类型	复位	说明
15	保留	R	0h	保留。始终读为 0。
14-12	FLLD	RW	1h	FLL 环路分频器。这些位在 FLL 反馈环路中将 f(DCOCLK) 分频。这将引起一个针对乘法器位的附加乘法器。也请见乘法器位。 000b = f_{DCOCLK} ÷ 1 001b = f_{DCOCLK} ÷ 2（默认） 010b = f_{DCOCLK} ÷ 4 011b = f_{DCOCLK} ÷ 8 100b = f_{DCOCLK} ÷ 16 101b = f_{DCOCLK} ÷ 32 110b = 保留供后续使用 111b = 保留供后续使用
11-10	保留	R	0h	保留。始终读为 0。
9-0	FLLN	RW	1Fh	乘法器位。这些位设定 DCO 的乘法器值 N。N 必须大于 0。将零写入 FLIN 将使 N 被设定为 1。

3.3.4 CSCTL3 寄存器

时钟系统控制寄存器 3

图 3-10. CSCTL3 寄存器

15	14	13	12	11	10	9	8
保留							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
保留		SELREF		保留	FLLREFDIV(1)		
r0	r0	rw - 0	rw - 0	r0	rw - 0	rw - 0	rw - 0

(1) 当 XT1 仅支持 32kHz 时，这些位始终读/写为 000b。

表 3-6. CSCTL3 寄存器说明

位	字段	类型	复位	说明
15-6	保留	R	0h	保留。始终读为 0。
5-4	SELREF	RW	0h	FLL 基准选择。这些位选择 FLL 基准时钟源。 00b = XT1CLK 01b = REFOCLK 10b = 保留供后续使用 11b = 保留供后续使用。
3	保留	R	0h	保留。始终读为 0。
2-0	FLLREFDIV	RW	0h	FLL 基准分频器。这些位定义针对 f(FLLREFCLK) 的分频因子。 如果 XT1 支持的高频输入超过 32kHz，分频后的频率可作为 FLL 的基准频率。 000b = $f_{FLLREFCLK}$ ÷ 1 001b = $f_{FLLREFCLK}$ ÷ 32 010b = $f_{FLLREFCLK}$ ÷ 64 011b = $f_{FLLREFCLK}$ ÷ 128 100b = $f_{FLLREFCLK}$ ÷ 256 101b = $f_{FLLREFCLK}$ ÷ 512 110b = 保留供后续使用 111b = 保留供后续使用 如果 XT1 仅支持 32kHz 时钟，FLLREFDIV 始终读/写为零。 000b = $f_{FLLREFCLK}$ ÷ 1

3.3.5 CSCTL4 寄存器

时钟系统控制寄存器 4

图 3-11. CSCTL4 寄存器

15	14	13	12	11	10	9	8
保留							SELA
r0	r0	r0	r0	r0	r0	r0	rw-1
7	6	5	4	3	2	1	0
保留					SELMS		
r0	r0	r0	r0	r0	rw - 0	rw - 0	rw - 0

表 3-7. CSCTL4 寄存器说明

位	字段	类型	复位	说明
15-9	保留	R	0h	保留。始终读为 0。
8 个	SELA	RW	1h	选择 ACLK 源 0b = 具有分频器的 XT1CLK（不得超过 40kHz） 1b = REFO（内部 32kHz 时钟源）
7-3	保留	R	0h	保留。始终读为 0。
2-0	SELMS	RW	0h	选择 MCLK 和 SMCLK 时钟源 000b = DCOCLKDIV 001b = REFOCLK 010b = XT1CLK 011b = VLOCLK 1xxb = 保留供后续使用

3.3.6 CSCTL5 寄存器

时钟系统控制寄存器 5

图 3-12. CSCTL5 寄存器

15	14	13	12	11	10	9	8
保留			VLOAUTOOFF	保留			SMCLKOFF
r0	r0	r0	rw-1	r0	r0	r0	rw-0
7	6	5	4	3	2	1	0
保留		DIVS		保留	DIVM		
r0	r0	rw - 0	rw - 0	r0	rw - 0	rw - 0	rw - 0

表 3-8. CSCTL5 寄存器说明

位	字段	类型	复位	说明
15-13	保留	R	0h	保留。始终读为 0。
12	VLOAUTOOFF	RW	1h	VLO 自动关闭使能。该位在 VLO 不使用时将其关闭。 0b = VLO 始终开启 1b = VLO 在不使用时自动关闭（默认）
11-9	保留	R	0h	保留。始终读为 0。
8 个	SMCLKOFF	R/W	0h	SMCLK 关闭。该位关闭 SMCLK 时钟 0b = SMCLK 打开 1b = SMCLK 关闭
7-6	保留	R	0h	保留。始终读为 0。
5-4	DIVS	RW	0h	SMCLK 时钟源分频器。SMCLK 由 MCLK 直接提供。SMCLK 频率为所选时钟源之外的 DIVM 和 DIVS 的组合。 00b = ÷ 1 01b = ÷ 2 10b = ÷ 4 11b = ÷ 8
3	保留	R	0h	保留。始终读为 0。
2-0	DIVM	RW	0h	MCLK 源分频器。 000b = ÷ 1 001b = ÷ 2 010b = ÷ 4 011b = ÷ 8 100b = ÷ 16 101b = ÷ 32 110b = ÷ 64 111b = ÷ 128

3.3.7 CSCTL6 寄存器

时钟系统控制寄存器 6

图 3-13. CSCTL6 寄存器

15	14	13	12	11	10	9	8
保留				DIVA			
r0	r0	r0	r0	rw - 1	rw - 0	rw - 0	rw - 0
7	6	5	4	3	2	1	0
XT1DRIVE		XTS[1]	XT1BYPASS	XT1HFFREQ		XT1AGCOFF	XT1AUTOOFF
rw-1	rw - 1	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 1

(1) 如果器件不支持 XT1 HF 模式，该位只读为 0。相关配置信息，请参见具体器件的数据表。

表 3-9. CSCTL6 寄存器说明

位	字段	类型	复位	说明
15-12	保留	R	0h	保留。始终读为 0。
11-8	DIVA	RW	8h	ACLK 时钟源分频器。(1)(2) 0000b = ÷1 0001b = ÷16 0010b = ÷32 0011b = ÷64 0100b = ÷128 0101b = ÷256 0110b = ÷384 0111b = ÷512 1000b = ÷768 1001b= ÷1024 1111b 至 1010b = 保留
7-6	XT1DRIVE	RW	3h	XT1 振荡器电流可被调整为它的驱动需要。在开始的时候，为了实现可靠和快速启动，它从最高电源电流开始。如果需要，用户软件能够减少驱动强度。 这些位的配置在处于 LPM3.5 时保持不变，直至 LOCKLPM5 清零，而非寄存器位本身清零；因此，在清零 LOCKLPM5 前，需在器件从 LPM3.5 唤醒后对其进行重新配置。 00b = 最低驱动强度和流耗 01b = 较低驱动强度和流耗 10b = 较高驱动强度和流耗 11b = 最高驱动强度和流耗
5	XTS	RW(3)	0h(4)	XT1 模式选择 0b = 低频模式。 1b = 高频模式。
4	XT1BYPASS	RW	0h	XT1 旁路选择 0b = XT1 作为内部时钟源 1b = XT1 通过引脚作为外部时钟源
3-2	XT1HFFREQ	RW	0h	XT1 高频选择。这些位必须设为合适的频率以实现晶振工作模式或旁路工作模式。(1) 00b = 1MHz 至 4MHz 01b = 介于 4MHz 至 6MHz 之间 10b = 介于 6MHz 至 16MHz 之间 11b = 介于 16MHz 至 24MHz 之间

(1) 这些位仅在 XT1 HF 模式下有效。分频器设置取决于外部高频振荡器的值，因为 ACLK 的频率固定为不高于 40kHz（典型值）。详细信息请参见具体器件的数据表。
(2) 如果 ACLK 选用 HF 模式下的 XT1 作为时钟源，该分频器始终被旁路。
(3) 如果器件不支持 XT1 HF 模式，这些位为只读位。相关配置信息，请参见具体器件的数据表。
(4) 如果器件不支持 XT1 HF 模式，这些位只读为 0。相关配置信息，请参见具体器件的数据表。

表 3-9. CSCTL6 寄存器说明 (continued)

位	字段	类型	复位	说明
1	XT1AGCOFF	RW	0h	自动增益控制 (AGC) 禁用。 0b = AGC 开启 1b = AGC 关闭
0	XT1AUTOOFF	RW	1h	XT1 自动关闭使能。该位允许在不使用 XT1 时将其关闭。 0b = 如果端口选择 XT1 并且 XT1 并未以旁路模式工作，则 XT1 开启。 1b = 如果 XT1 不用作 ACLK、MCLK 或 SMCLK 的时钟源或不用作 FLL 的基准时钟源，则 XT1 关闭。

3.3.8 CSCTL7 寄存器

时钟系统控制寄存器 7

图 3-14. CSCTL7 寄存器

15	14	13	12	11	10	9	8
保留		FLLWARNEN	FLLULPUC	FLLUNLOCKHIS		FLLUNLOCK	
r0	r0	rw - 0	rw - (0)	rw - (0)	rw - (1)	r-1	r-1
7	6	5	4	3	2	1	0
保留	ENSTFCNT1	保留	FLLULIFG	保留		XT1OFFG	DCOFFG
r-0	rw-(1)	r0	rw-(0)	r0	r0	rw-(0)	rw-(0)

表 3-10. CSCTL7 寄存器说明

位	字段	类型	复位	说明
15-14	保留	R	0h	保留。始终读为 0。
13	FLLWARNEN	RW	0h	警报启用。如果这个位被置 1，根据 FLUNLOCKHIS 位生成一个中断。如果 FLUNLOCKHIS 不等于 00，一个 OFIFG 被生成。 0b = FLLUNLOCKHIS 状态不能将 OFIFG 置 1。 1b = FLLUNLOCKHIS 状态能够将 OFIFG 置 1。
12	FLLULPUC	RW	0h	FLL 解锁 PUC 使能。如果 FLLULPUC 位置 1，那么在 FLLULIFG 置 1 的情况下将触发复位 (PUC)。FLLULIFG 指示 FLLUNLOCK 位何时等于 10（过快）。FLLULPUC 在处理事件时自动清零。如果 FLLULPUC 清零 (0)，FLLULIFG 不会触发 PUC。
11-10	FLLUNLOCKHIS	RW	1h	结果历史位。这些位表示 FLL 解锁条件历史记录。一旦任一结果条件发生，相应的位被置 1 并且在由软件通过写入 0 清零或者由一个加电复位 (POR) 清零前保持置 1 状态。 00b = FLL 被锁定。这些位最后一次复位后未检测到解锁情况。 01b = 自从这些位被清零后，DCOCLK 的速度过慢。 10b = 自从这些位被清零后，DCOCLK 电平已经过快 11b = 自从这些位被清零后，DCOCLK 电平已经过快和过慢
9-8	FLLUNLOCK	R	3h	取消锁定。这些位表示当前的 FLL 解锁条件。只要 DCOFF 标志被置 1，这两个位就都被置 1。 00b = FLL 被锁定。当前没有激活的锁定条件。 01b = DCOCLK 当前过慢。 10b = 当前 DCOCLK 过快。 11b = DCOERROR。DCO 在范围之外。
7	保留	R	0h	保留。始终读为 0。
6	ENSTFCNT1	RW	1h	使能 XT1 的启动计数器。 0b = 启动故障计数器禁用。计数器清零。 1b = 启动故障计数器使能。
5	保留	R	0h	保留。始终读为 0。
4	FLLULIFG	RW	0h	FLL 解锁中断标志。当 FLLUNLOCK 位等于 01b 时这个标志被置 1（DCO 过快时）。如果 FLLULPUC 同时置 1，FLLUIFG 置 1 时将触发 PUC。 0b = FLLUNLOCK 位不等于 10b 1b = FLLUNLOCK 位等于 10b
3-2	保留	R	0h	保留。始终读为 0。
1	XT1OFFG	RW	0h	XT1 振荡器故障标志。如果该位置 1，OFIFG 标志也会置 1。如果出现 XT1 故障，XT1OFFG 置 1。XT1OFFG 可由软件清零。如果 XT1 故障情况仍然存在，XT1OFFG 置 1。 0b = 未发生故障 1b = XT1 故障。发生 XT1 故障

表 3-10. CSCTL7 寄存器说明 (continued)

位	字段	类型	复位	说明
0	DCOFFG	RW	0h	DCO 故障标志。如果该位置 1，OFIFG 标志也会置 1。如果 DCO {0} 或 DCO = {511}，DCOFFG 位置 1。DCOFFG 可由软件清零。如果 DCO 故障条件仍然存在，DCOFFG 被置 1。只要 DCOFFG 被置 1，FLLUNLOCK 显示 DCOERROR 条件。 0b = 未发生故障 1b = DCO 故障。发生 DCO 故障

3.3.9 CSCTL8 寄存器

时钟系统控制寄存器 8

图 3-15. CSCTL8 寄存器

15	14	13	12	11	10	9	8
保留							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
保留				MODOSCREQEN	SMCLKREQEN	MCLKREQEN	ACLKREQEN
r0	r0	r0	r0	rw - (0)	rw - (1)	rw-(1)	rw-(1)

表 3-11. CSCTL8 寄存器说明

位	字段	类型	复位	说明
15-4	保留	R	0h	保留。始终读为 0。
3	MODOSCREQEN	RW	0h	MODOSC 时钟请求使能。将这个位置 1 将启用针对 MODOSC 的条件模块请求。 0b = MODOSC 条件请求被禁用。 1b = MODOSC 条件请求被启用。
2	SMCLKREQEN	RW	1h	SMCLK 时钟请求使能。将这个位置 1 将启用针对 SMCLK 的条件模块请求。 0b = SMCLK 条件请求被禁用。 1b = SMCLK 条件请求被启用。
1	MCLKREQEN	RW	1h	MCLK 时钟请求使能。将这个位置 1 将启用针对 MCLK 的条件模块请求。 0b = MCLK 条件请求被禁用。 1b = MCLK 条件请求被启用。
0	ACLKREQEN	RW	1h	ACLK 时钟请求使能。将这个位置 1 将启用针对 ACLK 的条件模块请求。 0b = ACLK 条件请求被禁用。 1b = ACLK 条件请求被启用。

CPUX

本章将介绍支持 1MB 存储器访问的扩展型 MSP430X 16 位精简指令集 (RISC) CPU (CPUX) 及其寻址模式和指令集。

注: 在某些情况下，该器件系列中应用的 MSP430X CPUX（正式名称为 CPUXV2）的周期数与 2xx 和 4xx 系列中应用的 MSP430X CPUX 的周期数略有差异。

Topic **Page**

4.1 MSP430X CPU (CPUX) 说明

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。MSP430X CPU 与 MSP430 CPU 完全兼容。

MSP430X CPU 的特性包括：

- RISC 架构
- 垂直架构
- 包括程序计数器 (PC)、状态寄存器 (SR) 和堆栈指针 (SP) 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 位数据总线可实现对字宽自变量的操作。
- 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 的框图。

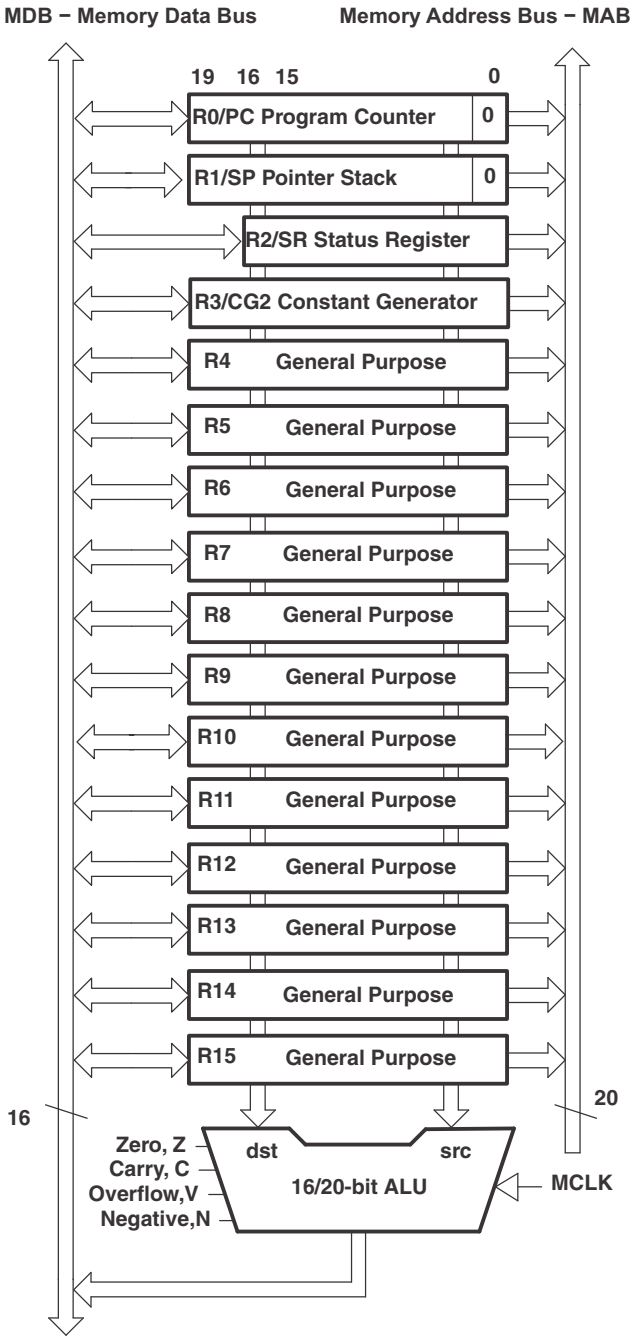


图 **4-1. MSP430X CPU** 框图

4.2 中断

MSP430X 具有下列中断结构：

- 无需轮询的向量中断
- 中断向量位于地址 0FFFEh 以下的位置.

中断向量包含指向低 64KB 存储空间的 16 位地址。这意味着所有中断处理程序必须从低 64KB 存储空间开始。

如图 4-2所示，中断期间，程序计数器 (PC) 和状态寄存器 (SR) 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。

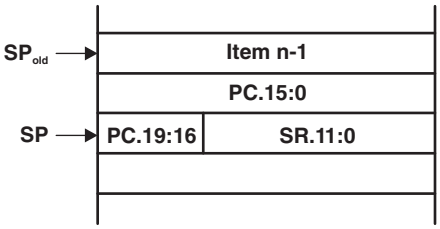


图 **4-2.** 存储在堆栈上用于中断的 **PC**

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)

20 位程序计数器 (PC，也叫 R0) 指向将被执行的下一条指令。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。

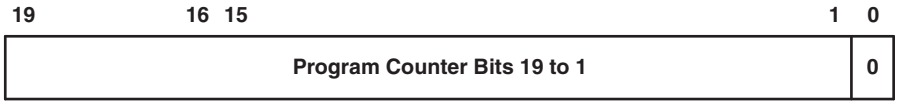


图 4-3. 程序计数器

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 指令将 PC 的高四位复位为 0。利用 BR 或 CALL 指令，只能到达低 64KB 地址范围内的地址。执行跳转或调用时，只能利用 BRA 或 CALLA 指令到达低 64KB 范围以外的地址。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。

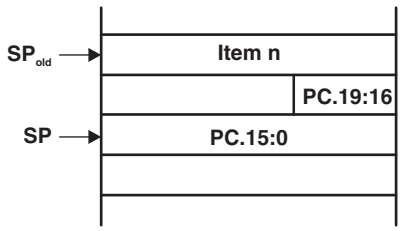


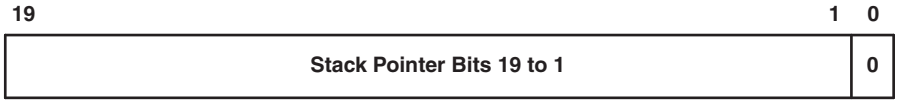
图 4-4. 针对 CALLA 的 PC 堆栈存储

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 位堆栈指针(SP，也叫 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

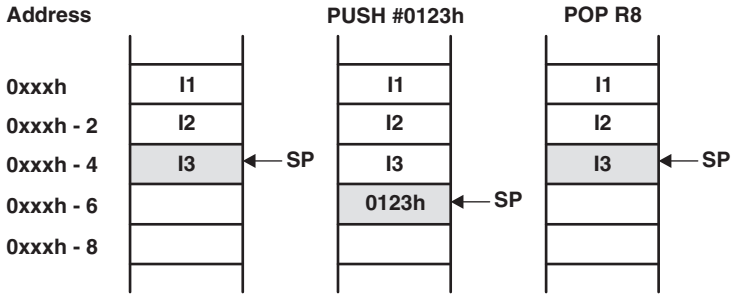


图 **4-6.** 堆栈用法

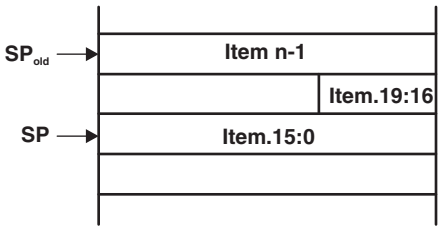


图 **4-7.** 堆栈上的 **PUSH.A** 格式

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。

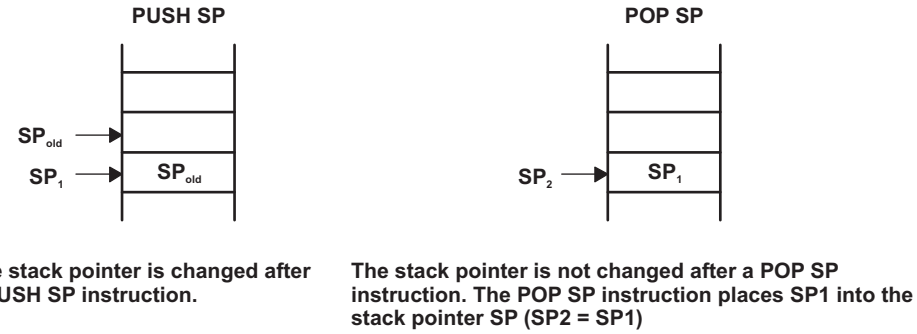


图 **4-8. PUSH SP，POP SP** 序列

4.3.3 状态寄存器 (SR)

用作一个源或者目标寄存器的 16 位栈堆指针（SR，也叫 R2）只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。

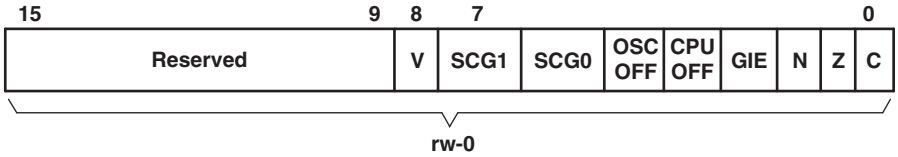


图 **4-9. SR** 位

表 4-1描述了 SR 位。

表 **4-1. SR** 位说明

位	说明	
保留	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 1。	
	ADD(.B), ADDX(.B,.A), ADDC(.B), ADDCX(.B.A), ADDA	当： 正 + 负 = 负 负 + 负 = 正 时置 1，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 1，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或禁用时钟系统内的功能；例如，数控振荡器 (DCO) 偏置启用或禁用。	
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或禁用时钟系统内的功能；例如，锁相环 (FLL) 启用或禁用。	
OSCOFF	振荡器关闭。当该位置 1 时，如果 LFXT1CLK 不用于 MCLK 或 SMCLK，则会关闭 LFXT1 晶振。	
CPUOFF	CPU 关闭。当该位置 1 时，将关闭 CPU。	
GIE	通用中断启用。当该位置 1 时，将使能可屏蔽的中断。当该位复位时，将禁用所有可屏蔽的中断。	
N	负。当运算结果为负时，这个位被置 1，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 1，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 1，而当没有出现进位时，这个位被清零。	

注： SR 的位处理应由下列指令来完成：MOV，BIS，BIC。

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 #1,dst
```

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。

图 4-10通过图 4-14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。

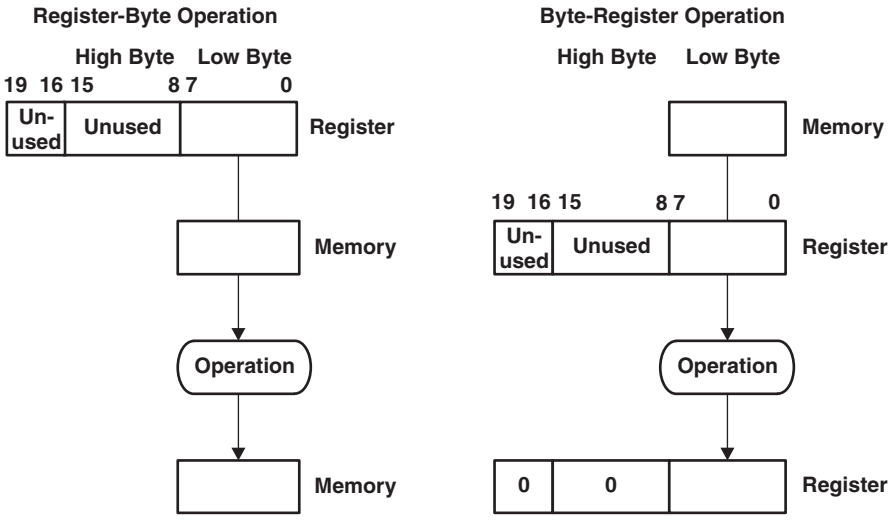


图 **4-10.** 寄存器字节和字节寄存器的操作

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。

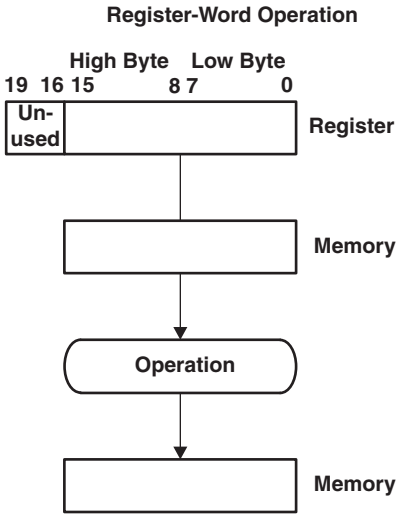


图 **4-11.** 寄存器-字操作

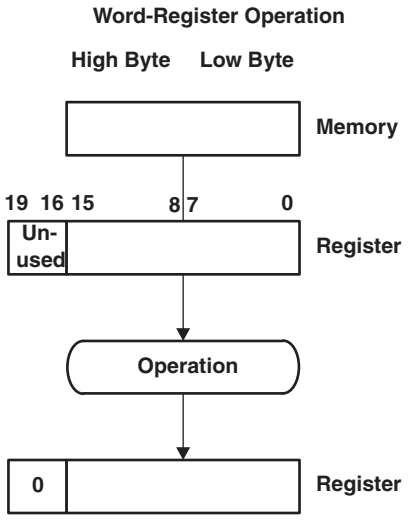


图 **4-12.** 字-寄存器操作

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。

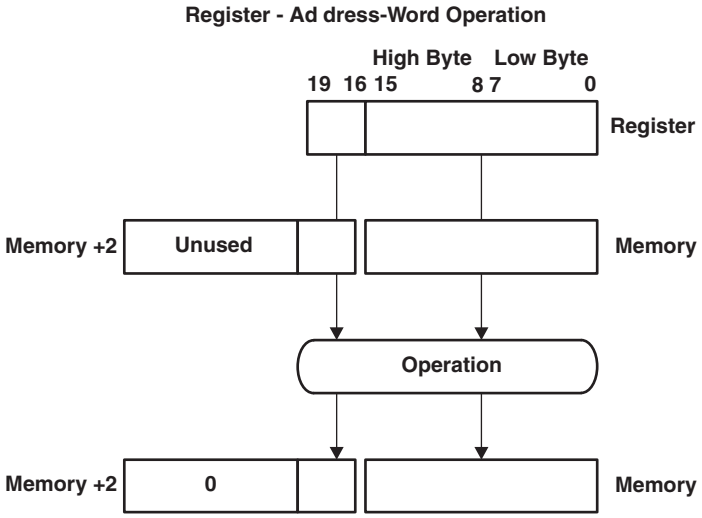


图 **4-13.** 寄存器-地址字操作

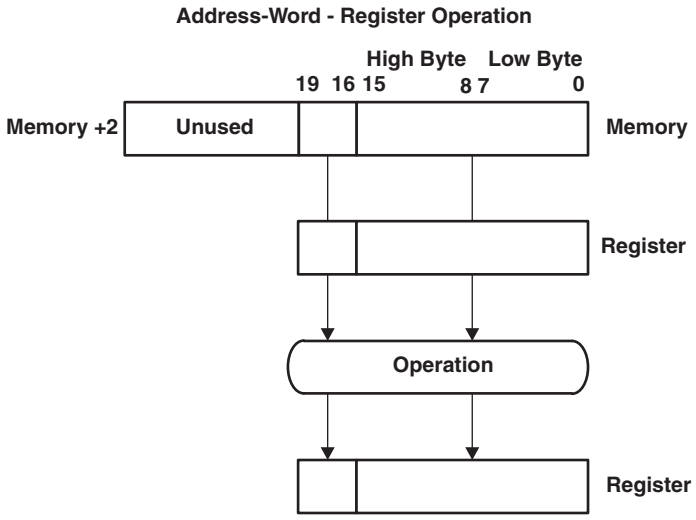


图 **4-14.** 地址-字-寄存器操作

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.** 源和目的地址

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。.B 指令使 Rn 逐 1 递增，.W 使其逐 2 递增，.A 使其逐 4 递增。
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。

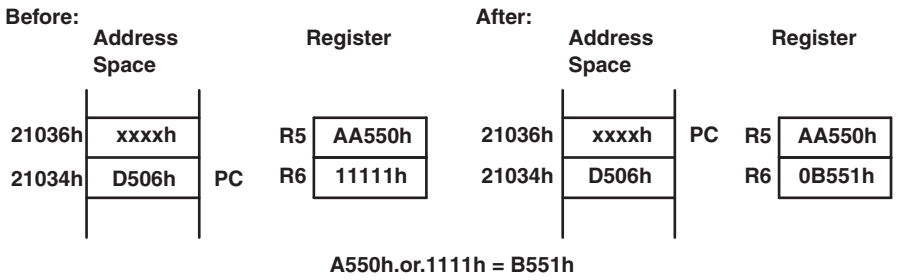
字操作： 字操作读取源寄存器 Rsrc 的 16 个 LSB，而将结果写入目的寄存器 Rdst 的 16 个 LSB 中。位 Rdst.19:16 被清零。寄存器 Rsrc 未被修改。

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`

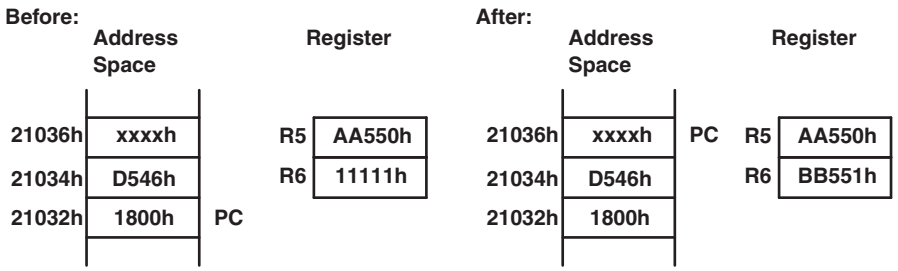
这条指令用 R6 的 16 位内容与 R5 包含的 16 位数据进行逻辑或 (OR) 操作。R6.19:16 被清零。



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。在索引模式下，有四种寻址方式可供选择：

- 索引模式搭配 MSP430 指令在低 64KB 存储空间内寻址
- 索引模式搭配 MSP430 指令在低 64KB 存储空间以外的存储空间内寻址
- 索引模式搭配 MSP430X 指令
- 索引模式搭配 MSP430X 寻址指令

4.4.2.1 索引模式搭配 MSP430 指令在低 64KB 存储空间内寻址

如果 CPU 寄存器 Rn 指向低 64KB 存储范围内的某一地址，则对 CPU 寄存器 Rn 和有符号 16 位索引执行相加计算后，存储器地址位 bit 19:16 将清零。这意味着计算得到的存储器地址始终位于低 64KB 存储空间内而不会溢出。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。

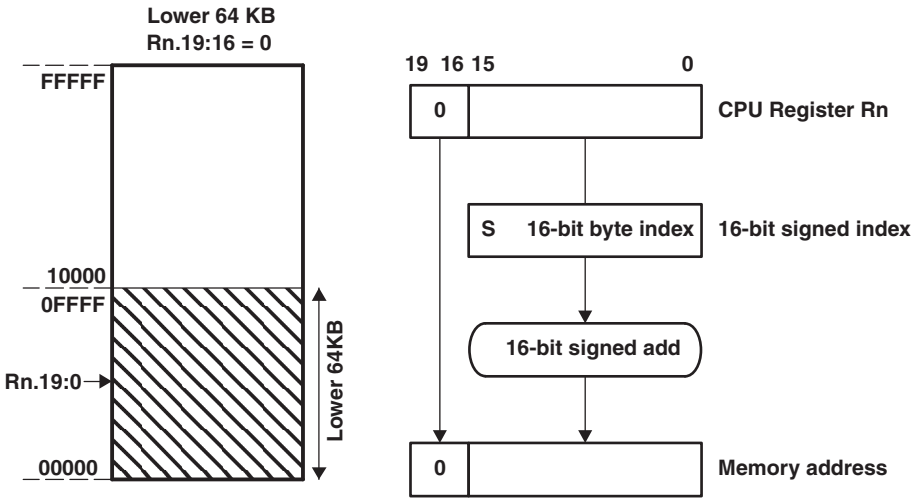


图 **4-15.** 低 **64KB** 中的已索引模式

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。

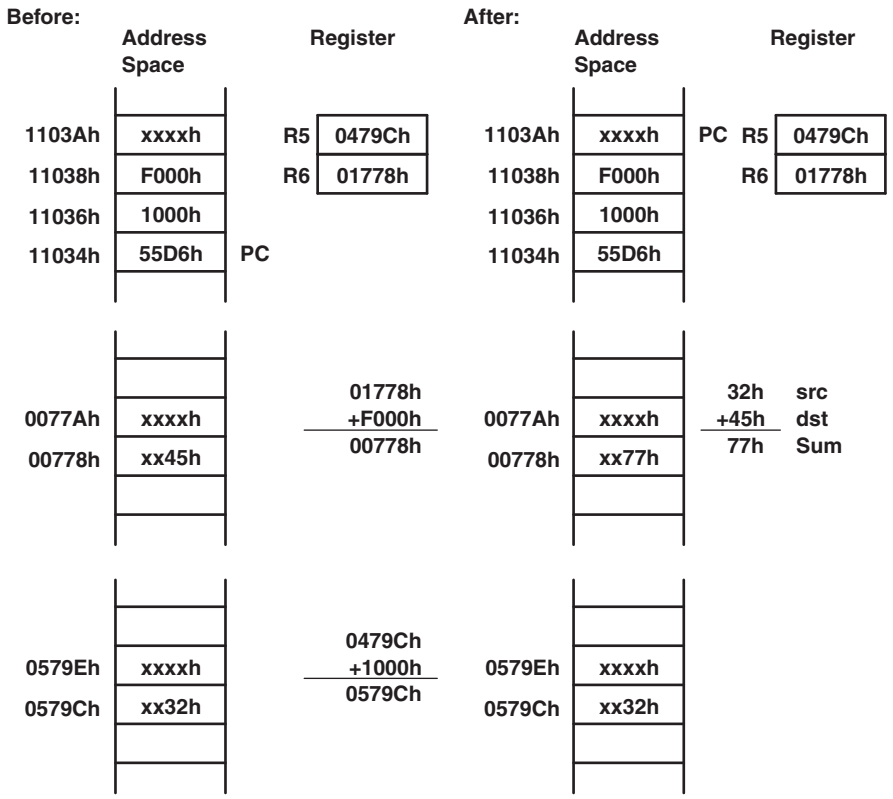
注释： 对于源和目的有效。汇编程序计算寄存器索引并将其插入。

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 指向低 64KB 存储空间以外的某一地址，则 Rn bit 19:16 用于计算操作数的地址。操作数可位于范围为 Rn±32KB 的存储空间内，这是因为索引 X 是一个有符号 16 位值。在这种情况下，操作数的地址可能会溢出低 64KB 存储空间（见图 4-16 和图 4-17）。

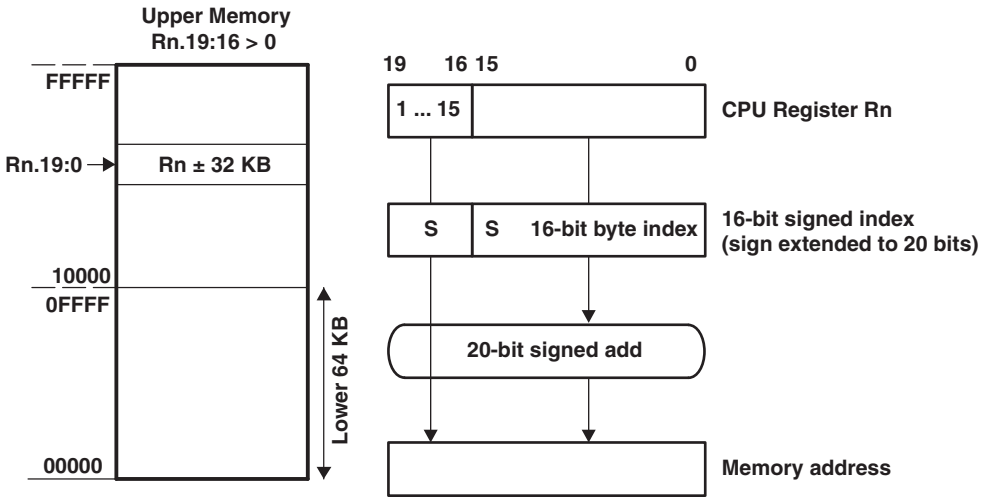


图 **4-16.** 上部存储器中的已索引模式

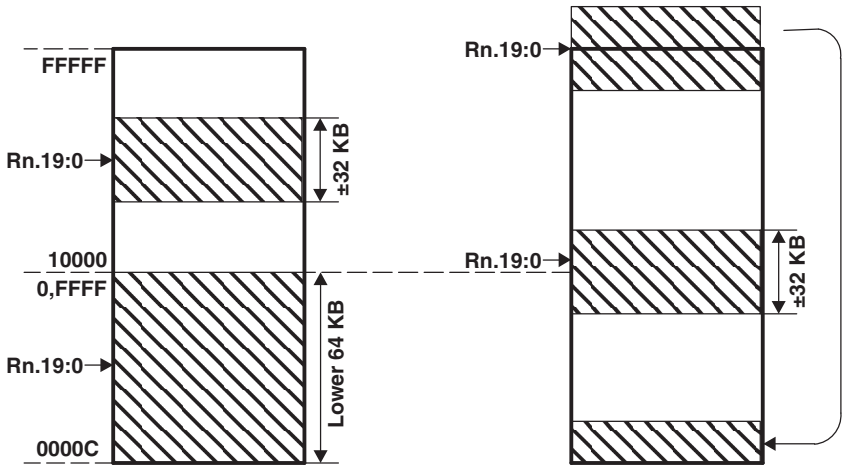


图 **4-17.** 针对已索引模式的上溢和下溢

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。

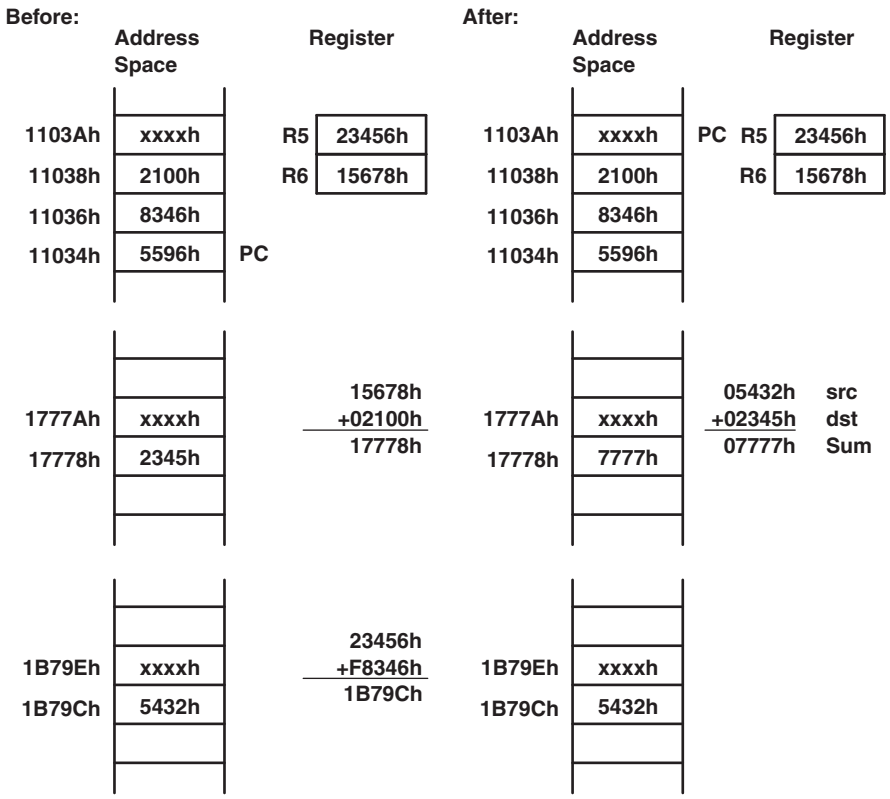


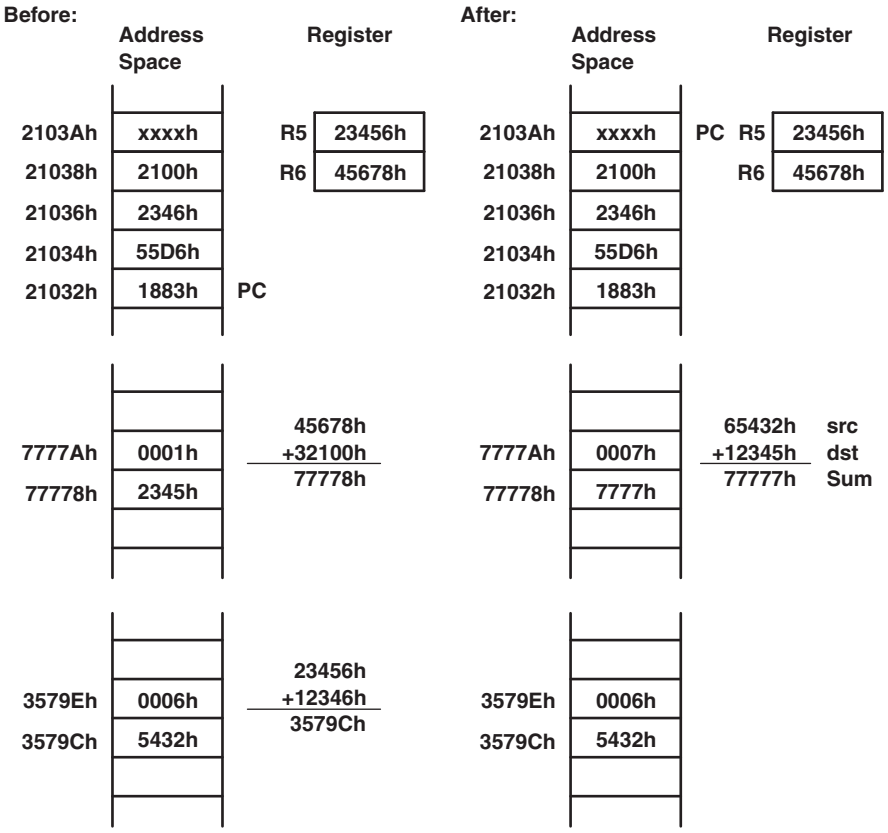
图 **4-18.** 索引模式示例

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。



4.4.2.4 索引模式搭配 MSP430X 寻址指令

当搭配使用索引模式和 MSP430X 寻址指令时，操作数位于范围为 Rn±32KB 的存储空间内，这是因为索引 X 是有符号 16 位值。

长度： 两个字

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：
```
MOVA 8002h(R5),R6 ; // R5 = 0x100
```
该指令将源地址中包含的 20 位数据装入目标寄存器。

源地址： 两个字，由 R5 + 8002h 和 R5 + 8002h + 2h 的计算结果指向的地址，即 00100h + F8002h (+2h) = F8102h 和 F8104h。

目标地址： 寄存器 R6

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：

- 通过符号模式在低 64KB 存储空间内寻址
- 符号模式搭配 MSP430 指令在低 64KB 存储空间以外的存储空间内寻址。
- 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。这意味着计算得到的存储器地址始终位于低 64KB 存储空间内而不会溢出。RAM 和外设寄存器可用这个方法进行访问并且如图 4-19所示，在无需修改的情况下现有的 MSP430 软件可用。

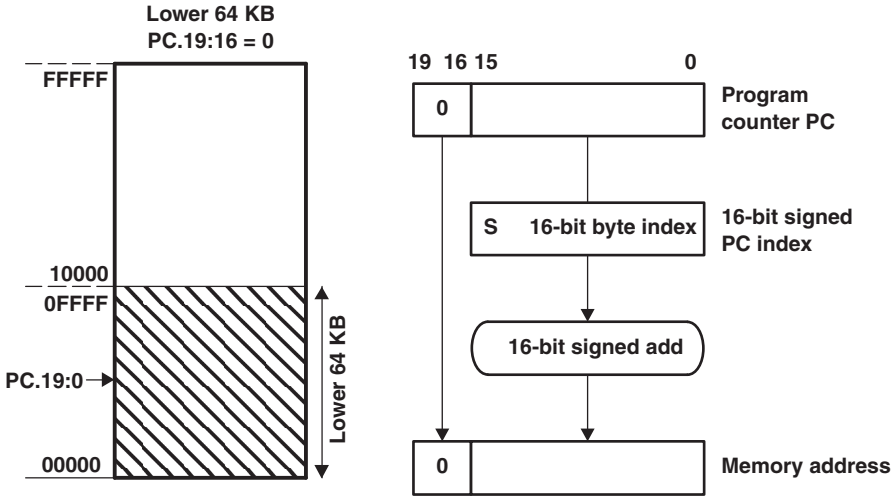


图 4-19. 低 64KB 中的符号模式运行那个

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

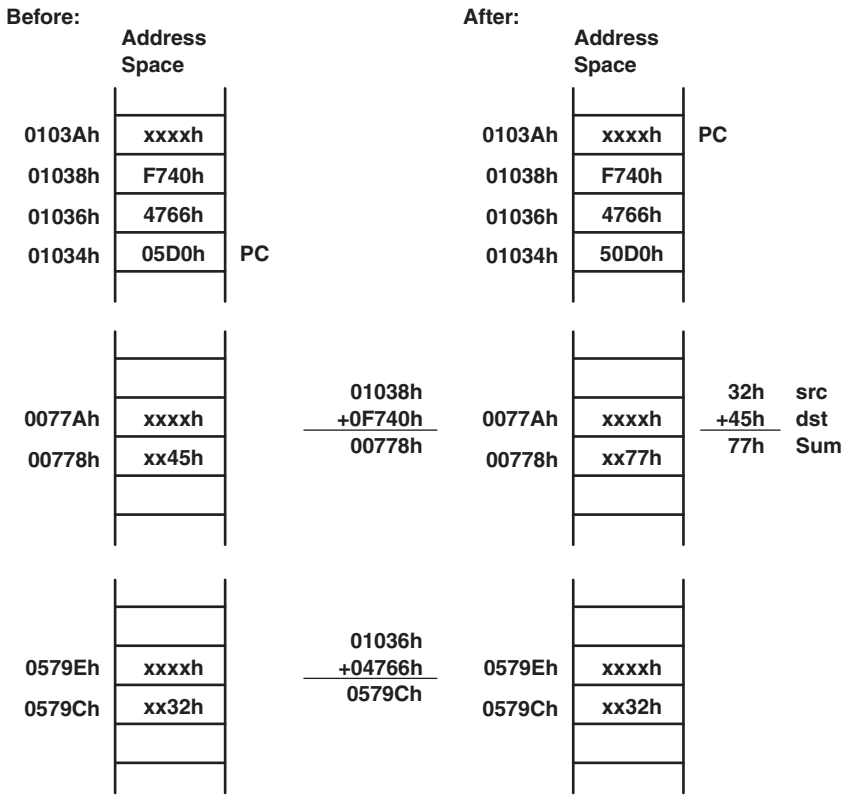
注释： 对于源和目的有效。汇编程序计算寄 PC 索引并将其插入。

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 指向低 64KB 存储空间以外的某一地址，则 PC bit 19:16 用于计算操作数的地址。操作数可位于范围为 PC±32KB 的存储空间内，这是因为索引 X 是一个有符号 16 位值。在这种情况下，操作数的地址可能会溢出低 64KB 存储空间，如图 4-20 和图 4-21 所示。

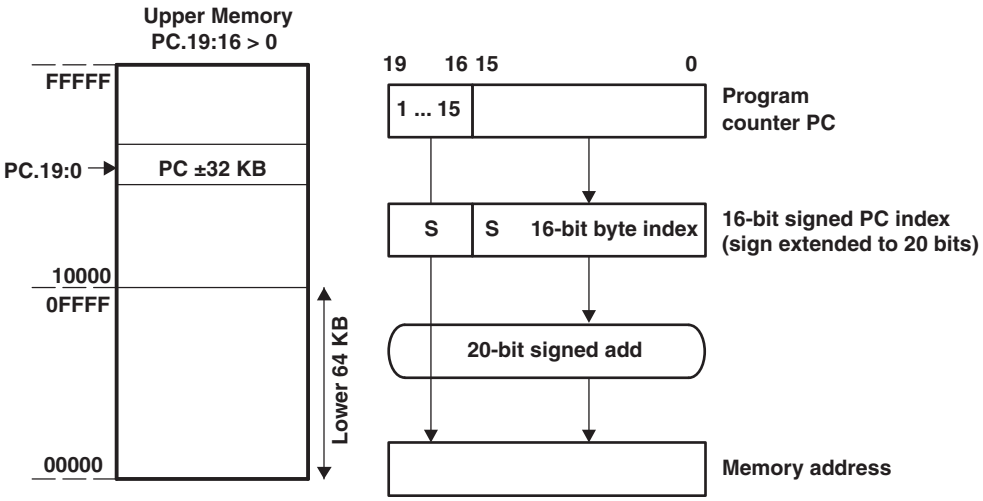


图 **4-20.** 上部存储器内的符号模式运行

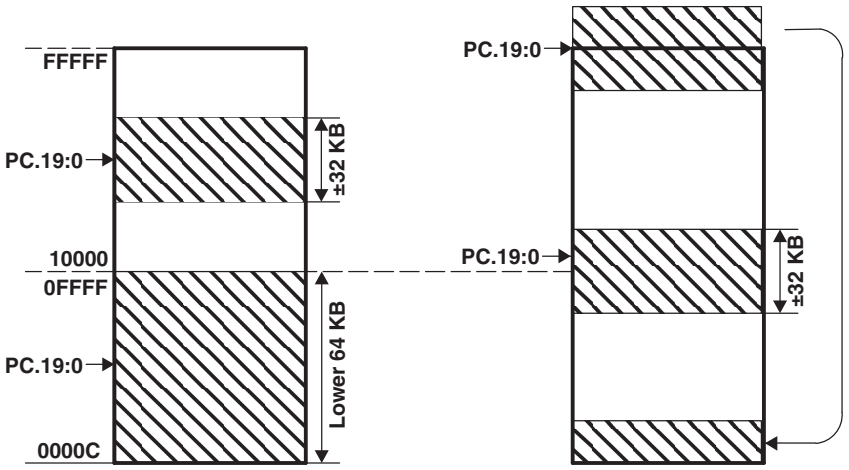


图 **4-21.** 针对符号模式的上溢和下溢

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。

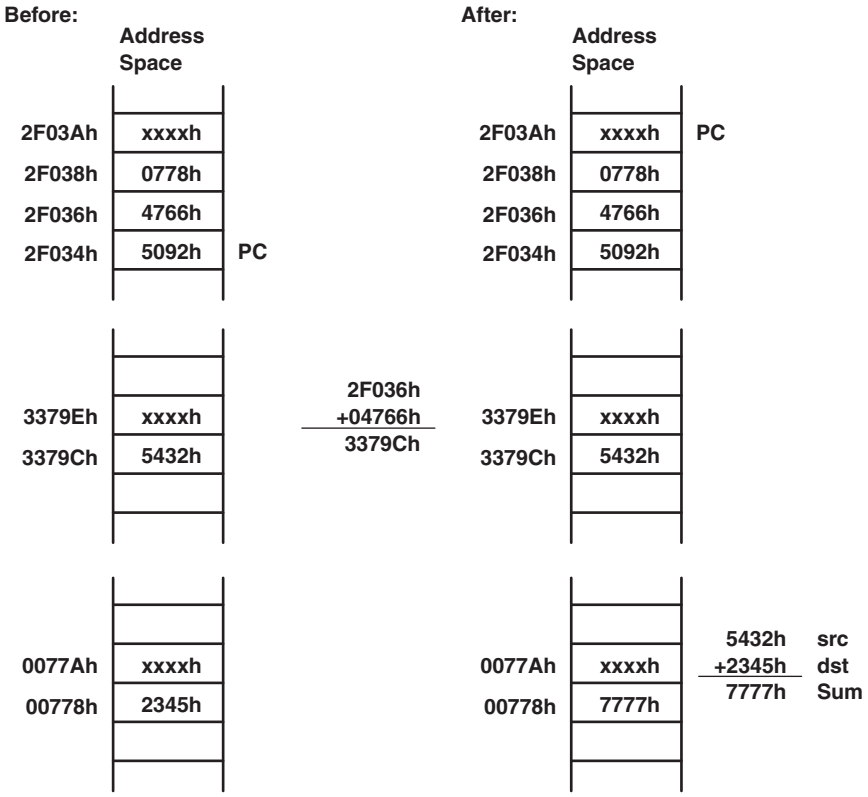
注释： 对于源和目的有效。汇编程序计算 PC 索引并将其插入。

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。

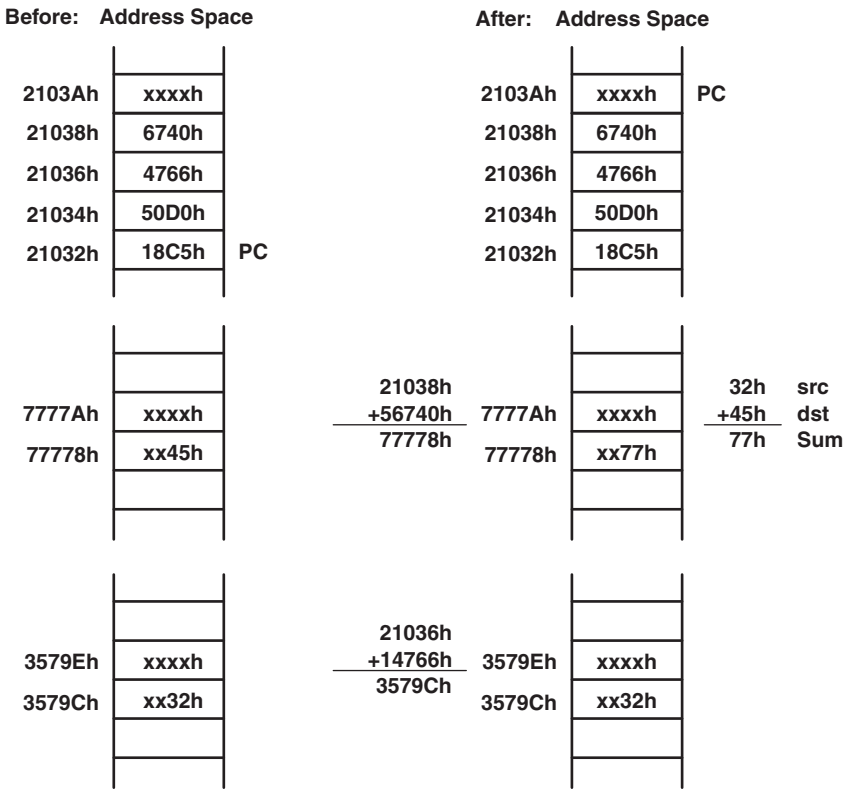
目标： 字 TONI 位于由绝对字 00778h 指向的地址 00778h 内。



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

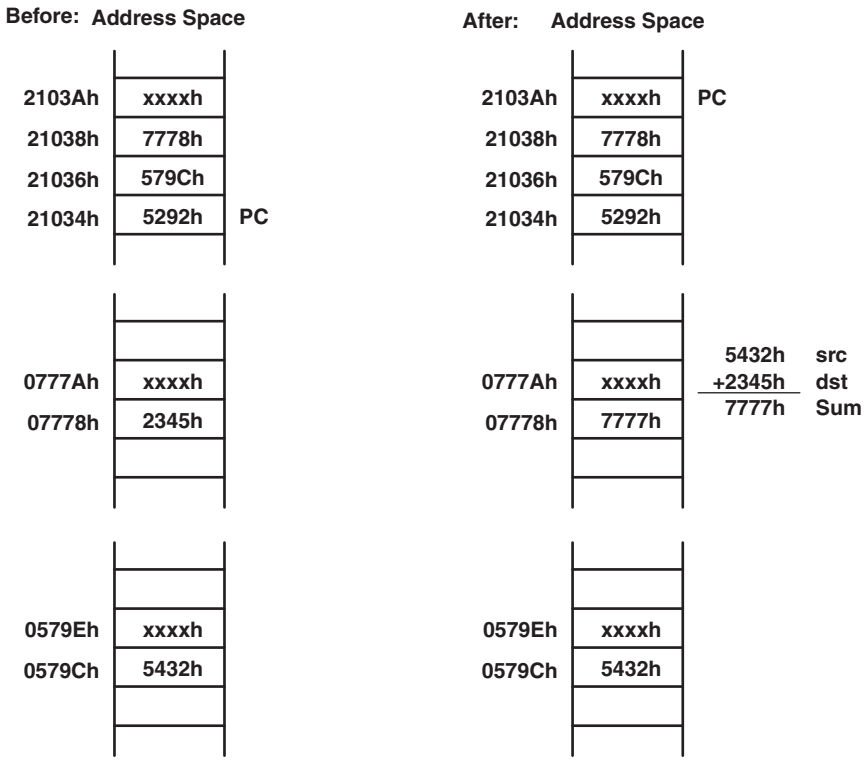
绝对模式使用指令之后的字的内容作为操作数的地址。绝对模式有两个各不同的可能：

- 通过绝对模式在低 64KB 存储空间内寻址
- 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。

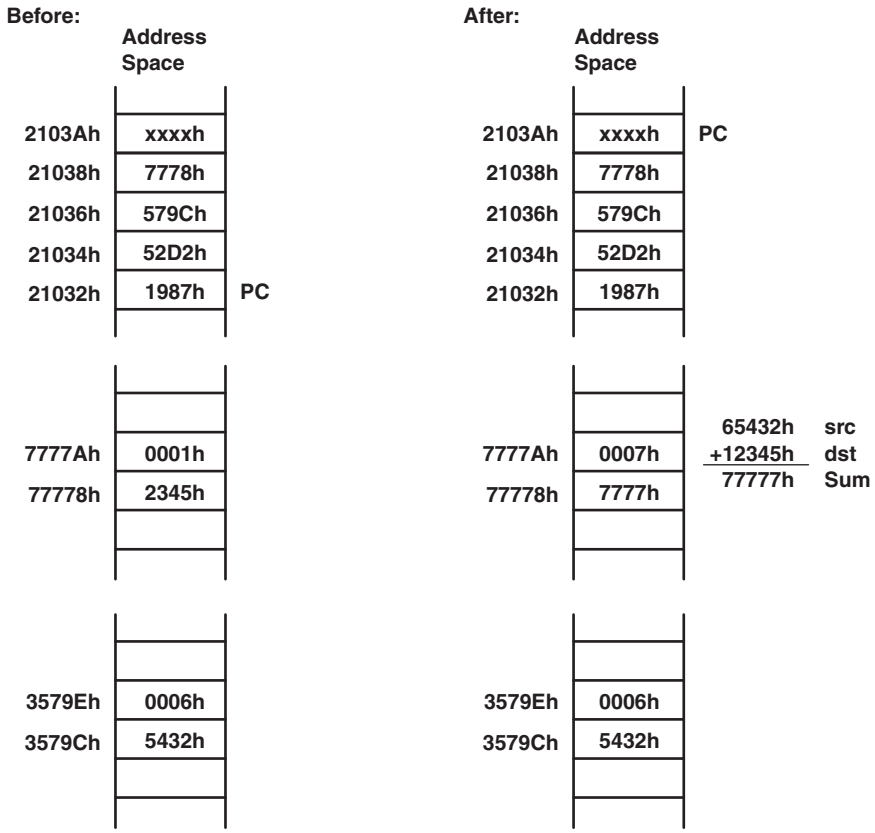
长度：	两个或者三个字
运行：	操作数是已寻址存储器位置的内容。
注释：	对于源和目的有效。汇编程序从 0 开始计算索引并将其插入。
示例：	`ADD.W &EDEM&TONIM` 这条指令加上包含在绝对源和目的地址中的 16 位数据并且将结果放置在目的地址内。
源：	地址 EDE 上的字
目标：	地址 TONI 上的字



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 MSB 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 开始计算索引并将其插入。
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。

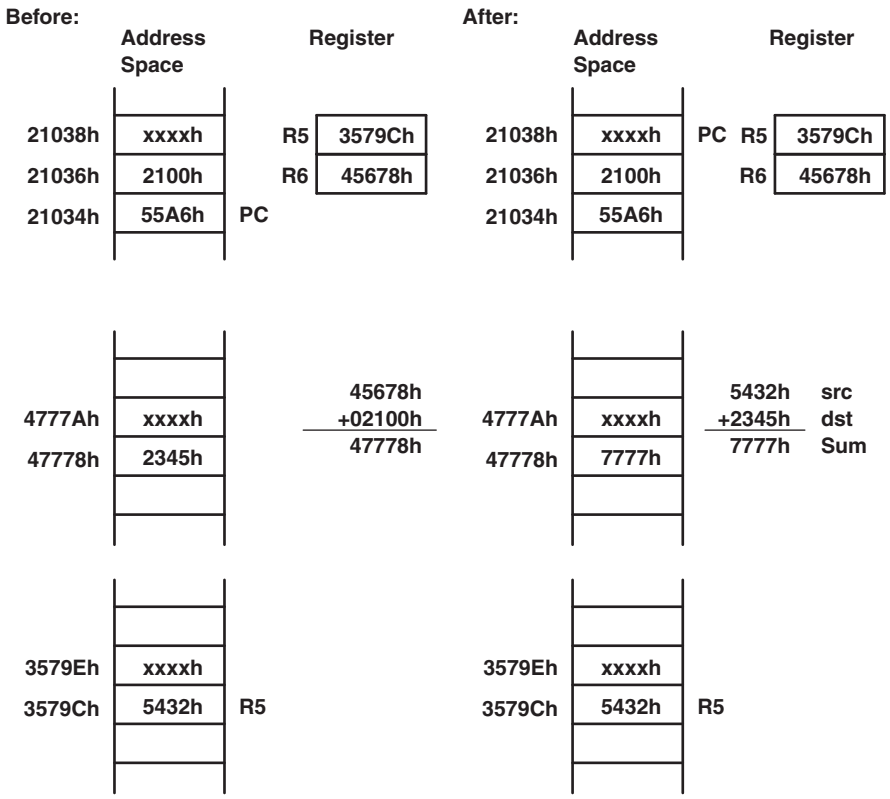
注释： 只对源操作数有效。目标操作数的替代值为 0 (Rdst)。

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。

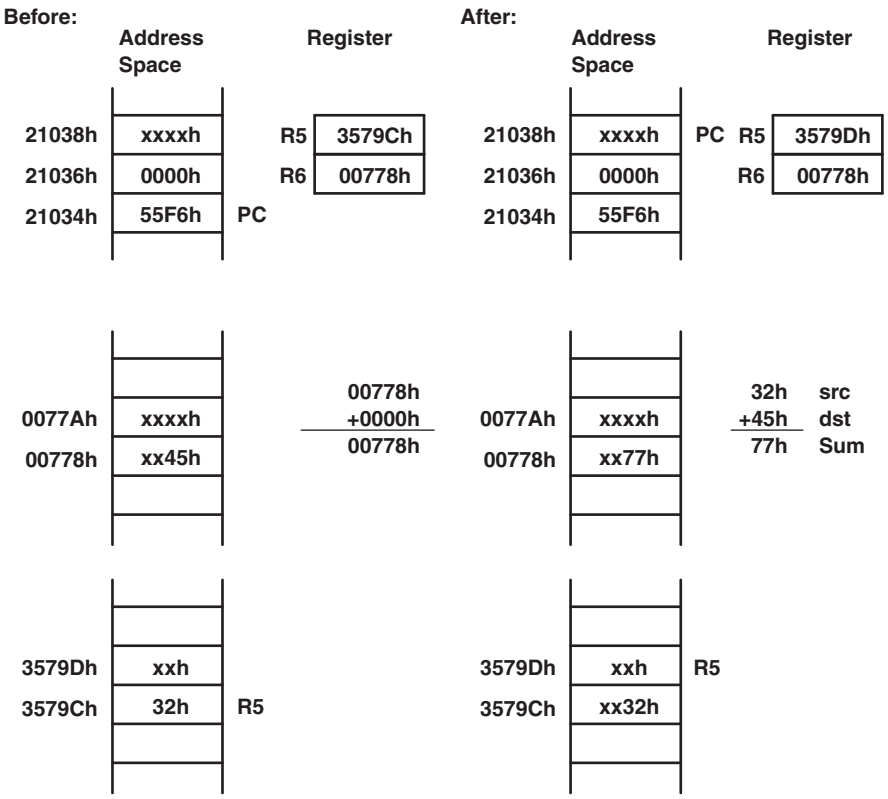
目的： R6 指向的字 + 2100h 得到地址 45678h+ 2100h=7778h。



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

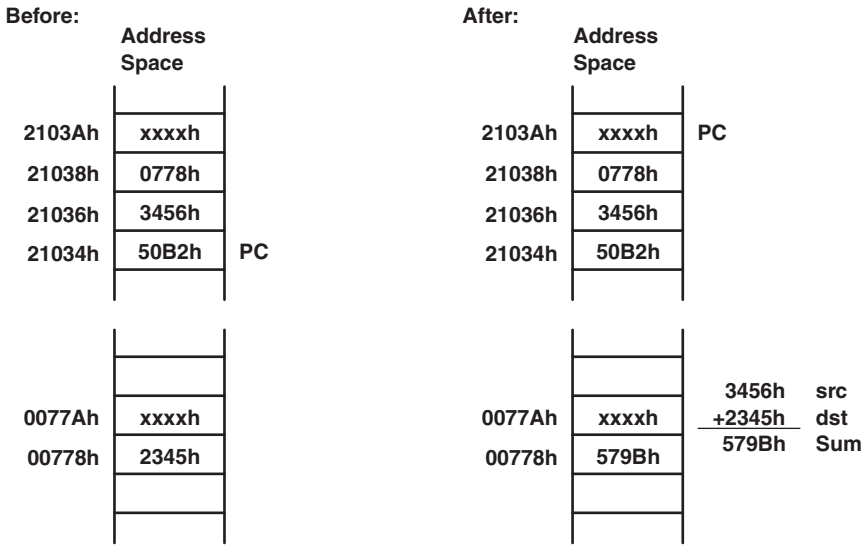
通过在指令之后将常数包含在存储器位置内，立即模式可实现将常数作为操作数进行访问。PC 使用间接自动递增模式。PC 指向下一个字中包含的立即值。在取得立即操作数后，针对字节、字、或者地址-字指令，PC 增加 2。立即模式有两个寻址可能：

- 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。

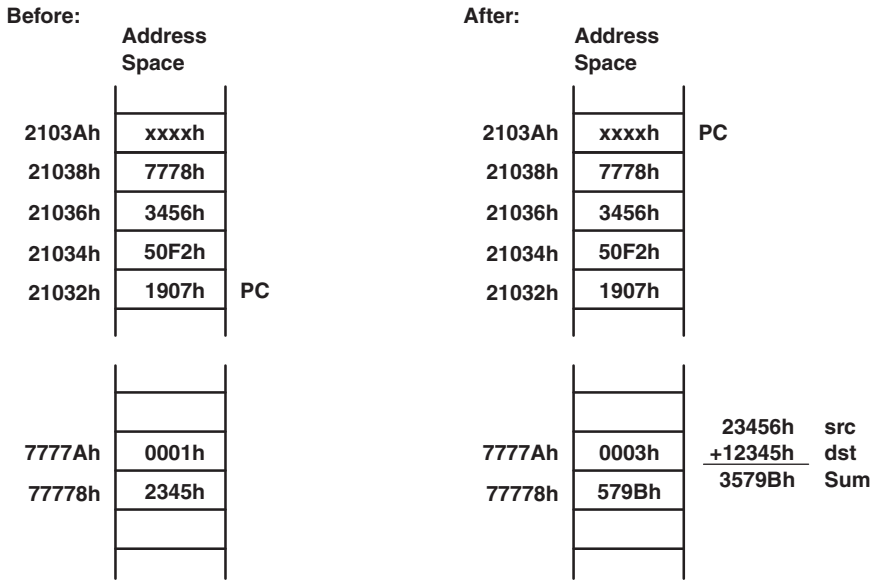
长度：	两个或者三个字。如果常数发生器的一个常数可被用于立即操作数，则少一个字。
运行：	16 位立即源操作数与 16 位目的操作数一起使用。
注释：	只针对源操作数有效
示例：	`ADD #3456hM&TONI` 这条指令将 16 位立即操作数 3456h 添加到目的地址 TONI 中的数据内。
源：	16 位立即值 3456h
目的：	地址 TONI 上的字



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： 只针对源操作数有效
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、表格和数据置于低 64KB 存储空间内。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及其 16 位索引到达 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。

4.5.1 节列出并说明了 MSP430 指令，和4.5.2 节列出并说明了 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。唯一的例外情况是 CALL 和 RET 指令，这两条指令限制在低 64KB 地址范围内。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 4-22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。

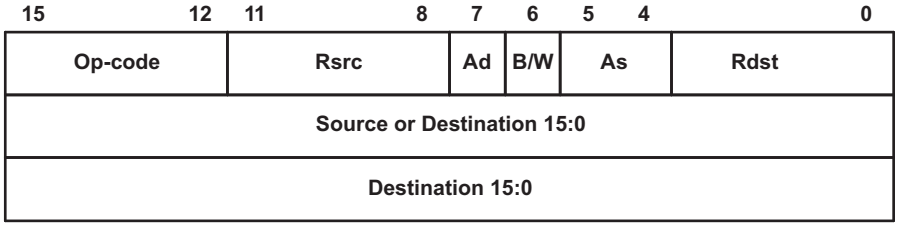


图 **4-22. MSP430** 双操作数指令格式

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 1。

4.5.1.2 MSP430 单操作数（格式 II）指令

图 4-23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。

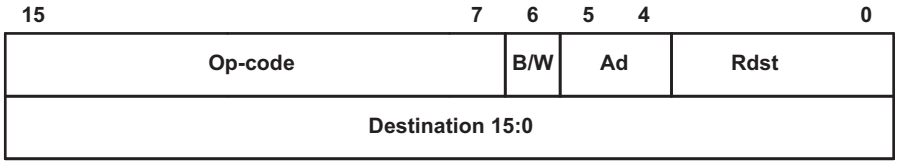


图 4-23. MSP430 单操作数指令

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	0	*	*	*
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 1。

4.5.1.3 跳转指令

图 4-24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。

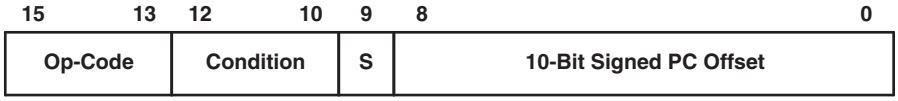


图 **4-24.** 条件跳转指令的格式

表 **4-6.** 条件跳转指令

助记符	S-Reg，D-Reg	操作
JEQM JZ	标签	如果零位被置 1 则跳转至标签
JNEM 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 1 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 1 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) #0M dst	*	*	*	*
BR dst	分支指令间接 dst	MOV dstM PC	-	-	-	-
CLR(.B) dst	清零 dst	MOV(.B) #0M dst	-	-	-	-
CLRC	清零进位位	BIC #1M SR	-	-	-	0
CLRN	清零负位	BIC #4M SR	-	0	-	-
CLRZ	清零零位	BIC #2M SR	-	-	0	-
DADC(.B) dst	用十进制将进位增加至 dst	DADD(.B) #0M dst	*	*	*	*
DEC(.B) dst	dst 减 1	SUB(.B) #1M dst	*	*	*	*
DECD(.B) dst	dst 减 2	SUB(.B) #2M dst	*	*	*	*
DINT	禁用中断	BIC #8M SR	-	-	-	-
EINT	启用中断	BIS #8M SR	-	-	-	-
INC(.B) dst	dst 增 1	ADD(.B) #1M dst	*	*	*	*
INCD(.B) dst	dst 增 2	ADD(.B) #2M dst	*	*	*	*

(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 1。

表 4-7. 仿真指令 (continued)

指令	说明	仿真	状态位(1)			
			V	N	Z	C
INV(.B) dst	反转 dst	XOR(.B) #-1,dst	*	*	*	*
NOP	无操作	MOV R3,R3	-	-	-	-
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	将进位位置 1	BIS #1,SR	-	-	-	1
SETN	将负位置 1	BIS #4,SR	-	1	-	-
SETZ	将零位置 1	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	5	1
从子例程 RET 返回	4	1
中断请求处理（第一个指令前需要的周期）	6	-
WDT 复位	4	-
复位 (RST/NMI)	4	-

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	4	1	SWPB R5
@Rn	3	3	4	1	RRC @R9
@Rn+	3	3	4	1	SWPB @R10+
#N	不可用	3	4	2	CALL #LABEL
X(Rn)	4	4	5	2	CALL 2(R7)
EDE	4	4	5	2	PUSH EDE
&EDE	4	4	6	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	指令的长度	示例
源	目标			
Rn	Rm	1	1	MOV R5M R8
	PC	3	1	BR R9
	x(Rm)	4(1)	2	ADD R5M 4(R6)
	EDE	4(1)	2	XOR R8M EDE
	&EDE	4(1)	2	MOV R5M &EDE
@Rn	Rm	2	1	AND @R4M R5
	PC	4	1	BR @R8
	x(Rm)	5(1)	2	XOR @R5M 8(R6)
	EDE	5(1)	2	MOV @R5M EDE
	&EDE	5(1)	2	XOR @R5M &EDE
@Rn+	Rm	2	1	ADD @R5+M R6
	PC	4	1	BR @R9+
	x(Rm)	5(1)	2	XOR @R5M 8(R6)
	EDE	5(1)	2	MOV @R9+M EDE
	&EDE	5(1)	2	MOV @R9+M &EDE
#N	Rm	2	2	MOV #20M R9
	PC	3	2	BR #2AEh
	x(Rm)	5(1)	3	MOV #0300hM 0(SP)
	EDE	5(1)	3	ADD #33M EDE
	&EDE	5(1)	3	ADD #33M &EDE
x(Rn)	Rm	3	2	MOV 2(R5)M R7
	PC	5	2	BR 2(R6)
	TONI	6(1)	3	MOV 4(R7)M TONI
	x(Rm)	6(1)	3	ADD 4(R4)M 6(R9)
	&TONI	6(1)	3	MOV 2(R4)M &TONI
EDE	Rm	3	2	AND EDEM R6
	PC	5	2	BR EDE
	TONI	6(1)	3	CMP EDEM TONI
	x(Rm)	6(1)	3	MOV EDEM 0(SP)
	&TONI	6(1)	3	MOV EDEM &TONI
&EDE	Rm	3	2	MOV &EDEM R8
	PC	5	2	BR &EDE
	TONI	6(1)	3	MOV &EDEM TONI
	x(Rm)	6(1)	3	MOV &EDEM 0(SP)
	&TONI	6(1)	3	MOV &EDEM &TONI

(1) MOV，BIT，和 CMP 指令在少一个周期内执行。

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 指令的寄存器或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 4-25中并在表 4-11中进行了说明。图 4-27显示了一个示例。

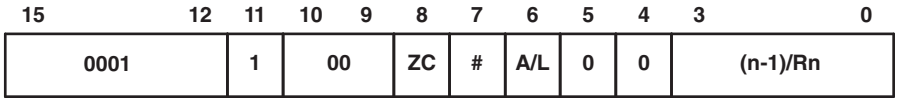


图 **4-25.** 针对寄存器模式的扩展字

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	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 1。 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 保留 0 1 20 位地址字 1 0 16 位字 1 1 8 位字节
5:4	保留
3:0	重复数量 # = 0 这四个位设置重复数量 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 设置重复的数量。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 4-26中并在表 4-12中进行了说明。图 4-28显示了一个示例。

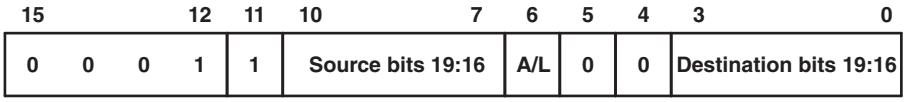


图 **4-26.** 针对非寄存器模式的扩展位

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 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 保留 0 1 20 位地址字 1 0 16 位字 1 1 8 位字节
5:4	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

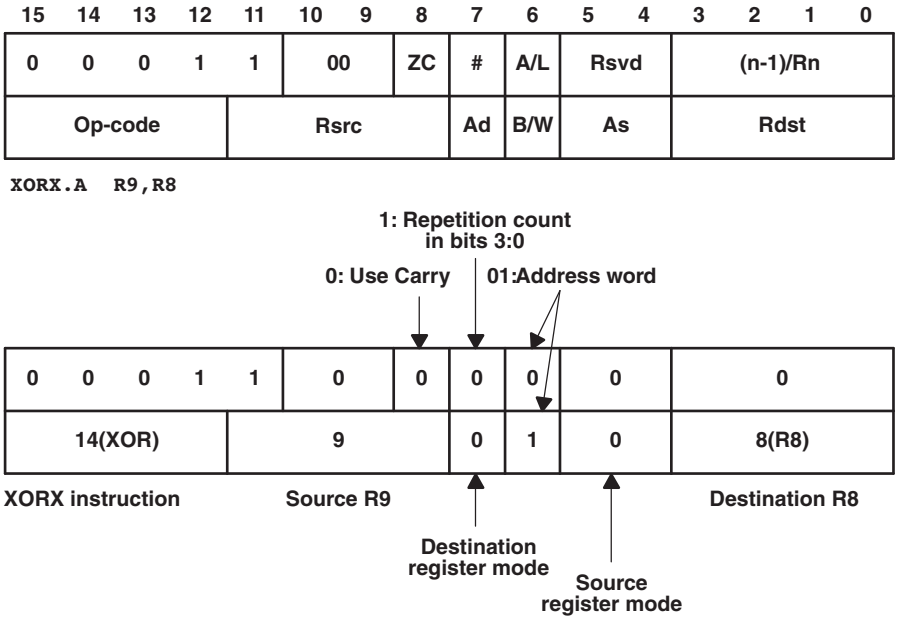


图 **4-27.** 针对扩展寄存器或寄存器指令的示例

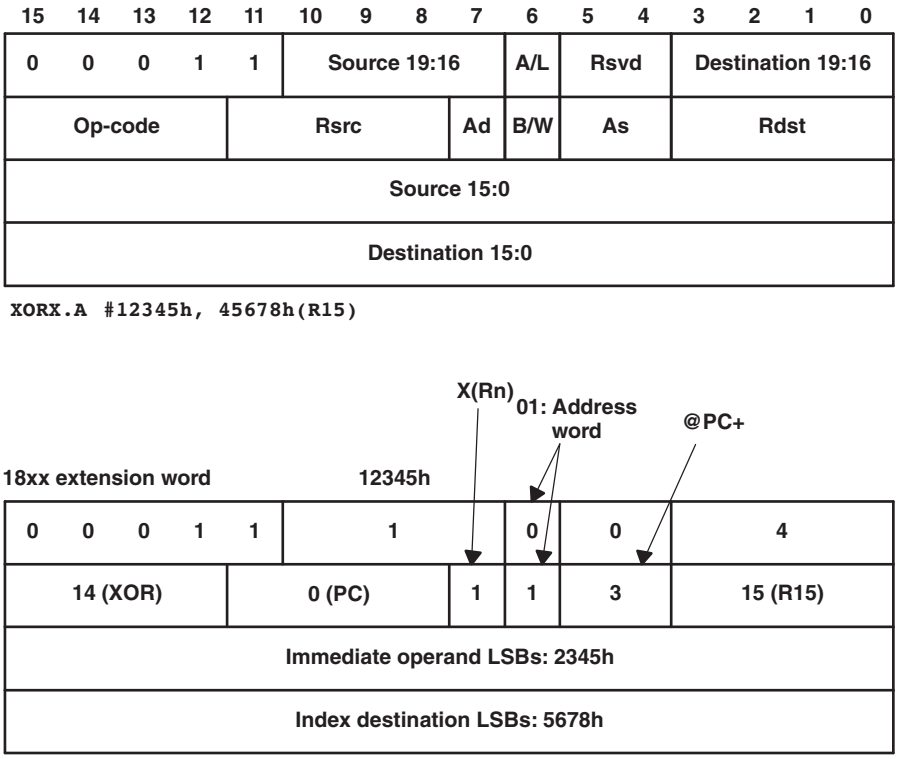


图 **4-28.** 针对扩展立即或已索引指令的示例

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 1。

针对格式 I 指令扩展字的四个可能的寻址组合显示在图 4-29中。

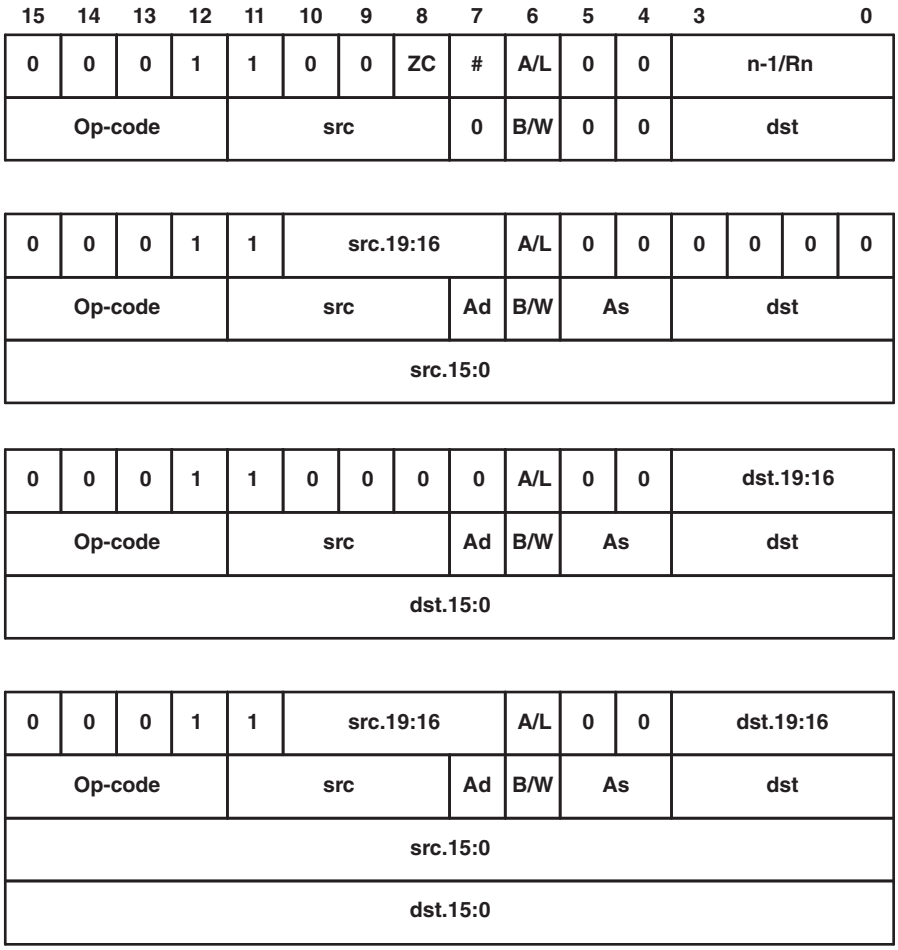


图 **4-29.** 扩展格式 **I** 指令格式

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 4-30中。

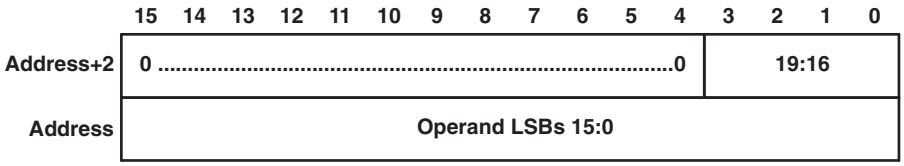


图 **4-30.** 存储器中的 **20** 位地址

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	0	*	*	*
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	0	*	*	*
SWPBX(.A)	dst	用高字节交换低字节	1	-	-	-	-
SXTX(.A)	Rdst	位 7 → 位 8 ... 位 19	1	0	*	*	Z
SXTX(.A)	dst	位 7 → 位 8 ... MSB	1	0	*	*	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 1。

针对格式 II 指令的三个可能寻址模式组合显示在图 4-31中。

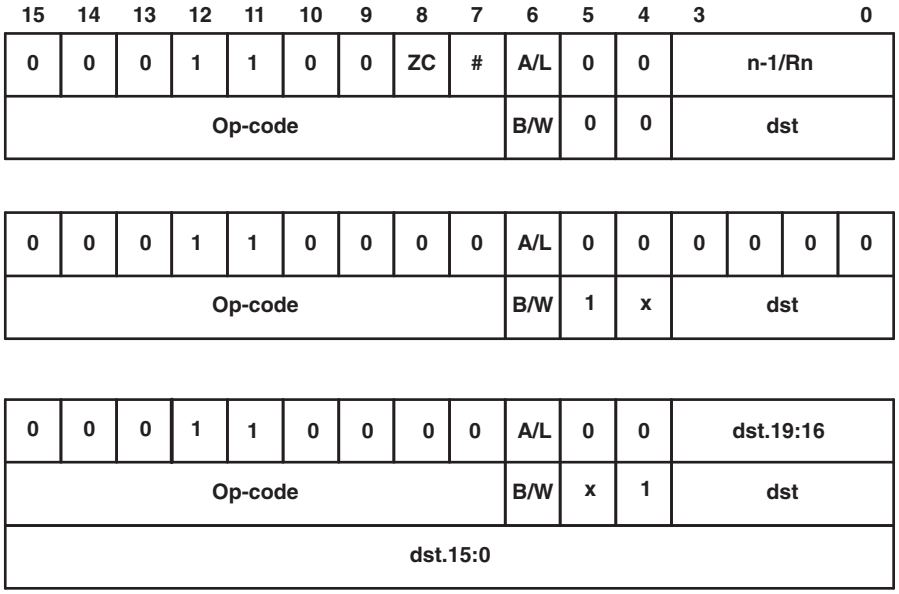


图 4-31. 扩展格式 II 指令格式

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 4-32至图 4-35中。

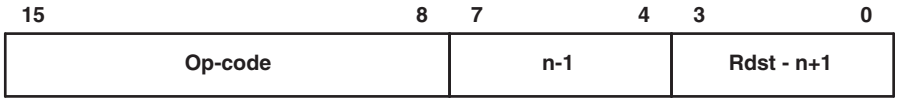


图 4-32. PUSHM 和 POPM 指令格式

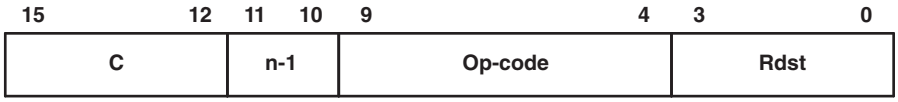


图 4-33. RRCM，RRAM，RRUM 和 RLAM 指令格式

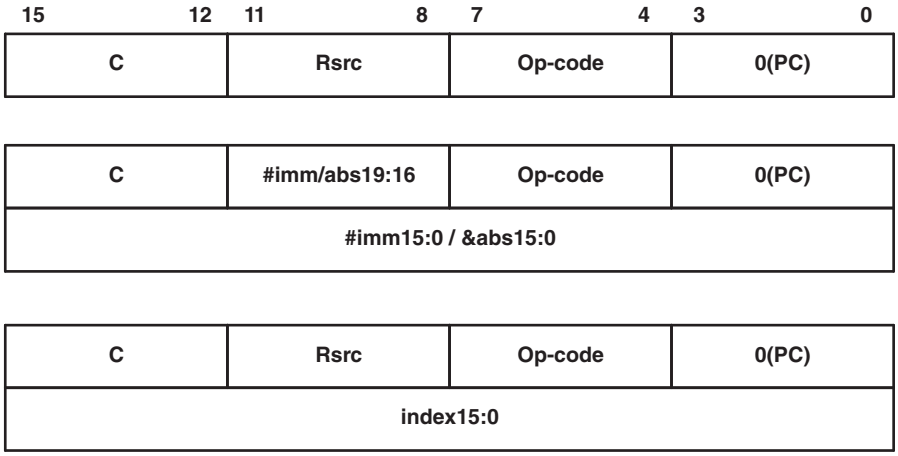


图 4-34. BRA 指令格式

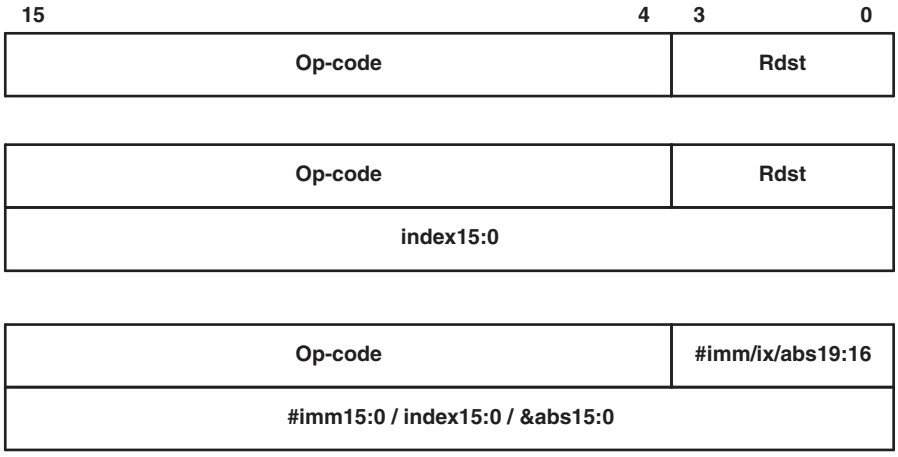


图 4-35. CALLA 指令格式

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	算术左移 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 1。

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	执行周期，指令长度（字）						
	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	5, 1	6, 1	6, 1	5, 2	5(1), 2	7, 2	7, 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	5，3	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 R5M R8
	PC	4	4	2	ADDX R9M PC
	x(Rm)	5[2]	7[3]	3	ANDX.A R5M 4(R6)
	EDE	5[2]	7[3]	3	XORX R8M EDE
	&EDE	5[2]	7[3]	3	BITX.W R5M &EDE
@Rn	Rm	3	4	2	BITX @R5M R8
	PC	5	6	2	ADDX @R9M PC
	x(Rm)	6[2]	9[3]	3	ANDX.A @R5M 4(R6)
	EDE	6[2]	9[3]	3	XORX @R8M EDE
	&EDE	6[2]	9[3]	3	BITX.B @R5M &EDE
@Rn+	Rm	3	4	2	BITX @R5+M R8
	PC	5	6	2	ADDX.A @R9+M PC
	x(Rm)	6[2]	9[3]	3	ANDX @R5+M 4(R6)
	EDE	6[2]	9[3]	3	XORX.B @R8+M EDE
	&EDE	6[2]	9[3]	3	BITX @R5+M &EDE
#N	Rm	3	3	3	BITX #20M R8
	PC[4]	4	4	3	ADDX.A #FE000hM PC
	x(Rm)	6[2]	8[3]	4	ANDX #1234M 4(R6)
	EDE	6[2]	8[3]	4	XORX #A5A5hM EDE
	&EDE	6[2]	8[3]	4	BITX.B #12M &EDE
x(Rn)	Rm	4	5	3	BITX 2(R5)M R8
	PC[4]	6	7	3	SUBX.A 2(R6)M PC
	TONI	7[2]	10[3]	4	ANDX 4(R7)M 4(R6)
	x(Rm)	7[2]	10[3]	4	XORX.B 2(R6)M EDE
	&TONI	7[2]	10[3]	4	BITX 8(SP)M &EDE
EDE	Rm	4	5	3	BITX.B EDEM R8
	PC[4]	6	7	3	ADDX.A EDEM PC
	TONI	7[2]	10[3]	4	ANDX EDEM 4(R6)
	x(Rm)	7[2]	10[3]	4	ANDX EDEM TONI
	&TONI	7[2]	10[3]	4	BITX EDEM &TONI
&EDE	Rm	4	5	3	BITX &EDEM R8
	PC[4]	6	7	3	ADDX.A &EDEM PC
	TONI	7[2]	10[3]	4	ANDX.B &EDEM 4(R6)
	x(Rm)	7[2]	10[3]	4	XORX &EDEM TONI
	&TONI	7[2]	10[3]	4	BITX &EDEM &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 R5M R8
	PC	3	3	1	1	SUBA R9M PC
	x(Rm)	4	-	2	-	MOVA R5M 4(R6)
	EDE	4	-	2	-	MOVA R8M EDE
	&EDE	4	-	2	-	MOVA R5M &EDE
@Rn	Rm	3	–	1	-	MOVA @R5M R8
	PC	5	-	1	-	MOVA @R9M PC
@Rn+	Rm	3	–	1	-	MOVA @R5+M R8
	PC	5	-	1	-	MOVA @R9+M PC
#N	Rm	2	3	2	2	CMPA #20M R8
	PC	3	3	2	2	SUBA #FE000hM PC
x(Rn)	Rm	4	-	2	-	MOVA 2(R5)M R8
	PC	6	-	2	-	MOVA 2(R6)M PC
EDE	Rm	4	-	2	-	MOVA EDEM R8
	PC	6	-	2	-	MOVA EDEM PC
&EDE	Rm	4	-	2	-	MOVA &EDEM R8
	PC	6	-	2	-	MOVA &EDEM 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 1Cxx	针对格式 I 和格式 II 指令的扩展字															
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 z16(Rsrc),Rdst
	x.15:0																
	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 Rsrc,z16(Rdst)
	x.15:0																
	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 #n,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																
保留	0	0	0	1	0	0	1	1	1	0	1	0	x	x	x	x	
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

下面将列出 MSP430 指令并对其进行描述。

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 1，如果为正则复位
Z： 如果结果为零则置 1，否则复位
C： 如果 dst 被从 0FFFFh 递增至 0000，则置 1，否则复位。
如果 dst 被从 0FFh 递增至 00，则置 1，否则复位。
V： 如果一个算术溢出发生则置 1，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 LSDs
ADC     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 LSDs
ADC.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 1，如果为正 (MSB=0)，则复位
Z： 如果结果为零则置 1，否则复位
C： 如果有一个来自结果的 MSB 的进位，则置 1，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 1，否则复位
模式位 OSCOFF，CPUOFF 和 GIE 不受影响。
示例 位于低 64K 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 = 0
JC      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: 000xxh
JNC     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

描述 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 1，如果为正 (MSB=0)，则复位

Z： 如果结果为零则置 1，否则复位

C： 如果有一个来自结果的 MSB 的进位，则置 1，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 1，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 64K 内的 16 位计数器 CNTR 内。

```
ADDC.W    #15,&CNTR      ; Add 15 + C to 16-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。R6.19:16=0

```
ADDC.W    @R5,R6         ; Add table word + C to R6
JC        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 + 1
JNC       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`

运行 src .and. dst→dst

描述 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 1，如果为正 (MSB=0)，则复位

Z： 如果结果为零则置 1，否则复位

C： 如果结果不为零则置 1，否则复位。C=(.not. Z)

V： 复位

模式位 OSCOFF，CPUOFF 和 GIE 不受影响。

示例 在 R5（16 位数据） 中被置 1 的位将被用作低 64K 内字 TOM 的一个掩码 (AA55h)。如果结果为 0，则会引入一个分支来标记 TON1。 R5.19:16=0

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

或更短：

```
AND     #AA55h,&TOM    ; TOM .and. AA55h -> TOM
JZ      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 1 的位
BIC.B 清零目的字节中源字节内置 1 的位
句法 `BIC srcMdst` 或 `BIC.W srcMdst`
`BIC.B srcMdst`
运行 (.not. src) .and. dst→dst
描述 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] 将在目的字中源字内置 1 的位置 1
BIS.B 将在目的字节中源字节内置 1 的位置 1
句法 `BIS srcMdst` 或 `BIS.W srcMdst`
`BIS.B srcMdst`
运行 src .or. dst→dst
描述 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 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将 Port1 中的位置 1。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 1 的位

BIT.B 测试在目的字节中源字节内置 1 的位

句法 `BIT srcMdst` 或 `BIT.W srcMdst`

`BIT.B srcMdst`

运行 src .and. dst

描述 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 1，如果为正 (MSB=0)，则复位

Z： 如果结果为零则置 1，否则复位

C： 如果结果不为零则置 1，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 1。如果被置 1 的话则跳转至标签 TONI。R5.19:16 未受影响。

```
BIT     #C000h,R5       ; Test R5.15:14 bits
JNZ     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 1，则跳转至标签 TONI。R7.19:16 未受影响。

```
BIT.W   @R5,R7          ; Test bits in R7
JC      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 1，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 + 1
JNC     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
描述	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 (for example #0A4h)
                  ; Core instruction MOV @PC+,PC

BR      EXEC      ; Branch to the address contained in EXEC
                  ; Core instruction MOV X(PC),PC
                  ; Indirect address

BR      &EXEC     ; Branch to the address contained in absolute
                  ; address EXEC
                  ; Core instruction MOV X(0),PC
                  ; Indirect address

BR      R5        ; Branch to the address contained in R5
                  ; Core instruction MOV R5,PC
                  ; Indirect R5

BR      @R5       ; Branch to the address contained in the word
                  ; pointed to by R5.
                  ; Core instruction MOV @R5,PC
                  ; Indirect, indirect R5

BR      @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 autoincrement

BR      X(R5)     ; Branch to the address contained in the address
                  ; pointed to by R5 + X (for example 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 64K 内的子例程

句法 MM dst

运行 dst → tmp 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

描述 在低 64K 中将一个子例程从一个地址调用到另一个地址。共有 7 个源寻址模式可用。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 64K 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 64K）上的子例程或者直接调用到地址。

```
CALL   #EXEC              ; Start address EXEC
CALL   #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 64K 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`
描述 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`

描述 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 start
DADD    @R13,0(R12)     ; add 16-bit counter to low word of 32-bit counter
DADC    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

描述 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。

```
          CLRN
          CALL   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`

描述 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
```

索引模式：调用位于某 16 位地址处的子程序，该 16 位地址包含在由寄存器 (R5 + X) 指向的 20 位地址中；例如，一个起始地址为 X 的表。该地址位于低 64KB 存储空间内。X 位于 ±32KB 范围内。

```
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

描述 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 1，如果为正则复位 (src≤dst)
- Z： 如果为零 (src=dst) 则置 1，否则复位 (src≠dst)
- C： 如果有来自 MSB 的进位，则置 1，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 1，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 1800h
JEQ     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 numbers
JL      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 + 1
JEQ     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`

描述 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 1

Z： 如果 dst 为 0 则置 1，否则复位

C： 如果目的从 9999 至 0000 递增则置 1，否则复位

如果目的从 99 至 00 递增，则置 1，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 defined
DADD   R5,0(R8)     ; Add LSDs + C
DADC   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 defined
DADD.B   R5,0(R8)   ; Add LSDs + C
DADC     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）

描述 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 1，如果 MSB 为 0 则复位
- Z： 如果结果为零则置 1，否则复位
- C： 如果 BCD 结果太大（字 > 9999h，字节 > 99h），则置 1，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 carry
DADD.W   &BCD,R4        ; Add LSDs. R4.19:16 = 0
DADD.W   &BCD+2,R5      ; Add MSDs with carry. R5.19:16 = 0
JC       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 = 0

```
CLRC                    ; Clear carry
DADD.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
描述 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 1，如果为正则复位
Z： 如果 dst 包含 1 则置 1，否则复位
C： 如果 dst 包含 0 则置 1，否则复位
V： 如果一个算术溢出发生，则置 1，否则复位。
如果目的的初始值为 08000h 则置 1，否则复位。
如果目的的初始值为 080h 则置 1，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+0FEh

        MOV     #EDE,R6
        MOV     #255,R10
L$1     MOV.B   @R6+,TONI-EDE-1(R6)
        DEC     R10
        JNZ     L$1
```

不要上面具有图 4-36中所显示的交迭的例程来传送表格。

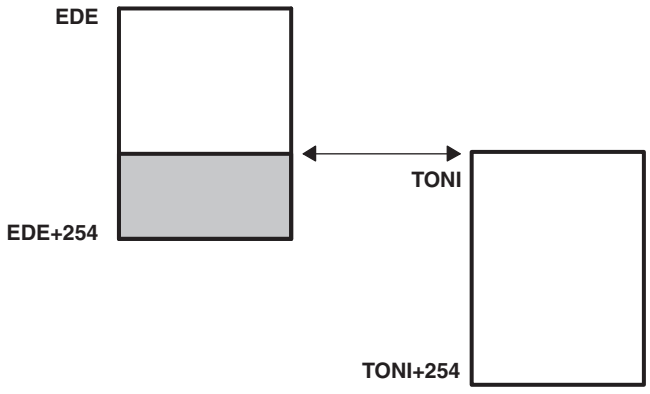


图 **4-36.** 递减交迭

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

描述 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 1，如果为正则复位

Z： 如果 dst 包含 2 则置 1，否则复位

C： 如果 dst 包含 0 则复位，否则置 1

V： 如果一个算术溢出发生，则置 1，否则复位。

如果目的初始值为 08001 或 08000h 则置 1，否则复位

如果目的的初始值为 081 或 080h 则置 1，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+0FEh

        MOV     #EDE,R6
        MOV     #255,R10
L$1     MOV.B   @R6+,TONI-EDE-2(R6)
        DECD    R10
        JNZ     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
描述	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 disabled
NOP                 ; Required due to pipelined CPU architecture
MOV    COUNTHI,R5   ; Copy counter
MOV    COUNTLO,R6
EINT                ; All interrupt events using the GIE bit are enabled
```

注: 禁用中断

由于流水线型 CPU 架构的原因，清零通用中断使能 (GIE) 位时需要特别小心。

- DINT 与需要中断保护的代码序列的起始代码之间至少包含一条指令。示例：在 DINT 之后插入一条 NOP 指令。
- 切勿在将通用中断使能 (GIE) 位置 1 后立即将其清零。此类序列之间至少插入一条指令。

上述规则适用于所有将通用中断使能位清零的指令。如果不遵循这些规则，则可能导致意外的 CPU 操作。

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
描述	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	将 SR 中的通用中断使能 (GIE) 位置 1。

```
          PUSH.B   &P1IN
          BIC.B    @SP,&P1IFG  ; Reset only accepted flags
          NOP                  ; Required due to pipelined CPU architecture
          EINT                 ; Preset port 1 interrupt flags stored on stack
                               ; other interrupts are allowed
          BIT   #Mask,@SP
          JEQ   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: 使能中断

由于流水线型 CPU 架构的原因，将通用中断使能 (GIE) 位置 1 时需要特别小心。

- 务必在使能中断指令 (EINT) 后立即执行该指令，即使有中断服务请求等待处理也如此。
- 中断使能位/中断标志位的清零与 EINT 指令之间至少包含一条指令。示例：在 EINT 指令之前插入一条 NOP 指令。
- 切勿在将通用中断使能 (GIE) 位置 1 后立即将其清零。此类序列之间至少插入一条指令。

上述规则适用于所有将通用中断使能位置 1 的指令。如果不遵循这些规则，则可能导致意外的 CPU 操作。

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 1，如果为正则复位

Z： 如果 dst 包含 0FFFFh 则置 1，否则复位

如果 dst 包含 0FFh 则置 1，否则复位

C： 如果 dst 包含 0FFFFh 则置 1，否则复位

如果 dst 包含 0FFh 则置 1，否则复位

V： 如果 dst 包含 07FFFh 则置 1，否则复位

如果 dst 包含 07Fh 则置 1，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   STATUS
CMP.B   #11,STATUS
JEQ     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`
描述 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 1，如果为正则复位
Z： 如果 dst 包含 0FFFEh 则置 1，否则复位
如果 dst 包含 0FEh 则置 1，否则复位
C： 如果 dst 包含 0FFFEh 或 0FFFFh 则置 1，否则复位
如果 dst 包含 0FEh 或 0FFh 则置 1，否则复位
V： 如果 dst 包含 07FFEh 或 07FFFh 则置 1，否则复位
如果 dst 包含 07Eh 或 07Fh 则置 1，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 register
RET
```

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
描述 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 1，如果为正则复位
Z： 如果 dst 包含 0FFFFh 则置 1，否则复位
如果 dst 包含 0FFh 则置 1，否则复位
C： 如果结果不为零则置 1，否则复位 (=.NOT. 零)
V： 如果初始目的操作数为负则置 1，否则复位
模式位 OSCOFF，CPUOFF 和 GIE 不受影响。
示例 将 R5 的内容取反（两进制补码）。

```
MOV     #00AEh,R5    ;                      R5 = 000AEh
INV     R5           ; Invert R5,           R5 = 0FF51h
INC     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) = 0AEh
INV.B   LEO          ; Invert LEO,          MEM(LEO) = 051h
INC.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

描述 SR 中的进位位 C 被测试。如果它被置 1，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 -> C
JC      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 C
JHS     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 C
JHS     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

描述 SR 中的零位 Z 被测试。如果它被置 1，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 SR
JEQ     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 R7
JZ      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

描述 SR 中的负位 N 和溢位 V 被测试。如果两个位都被置 1 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。如果只有一个位被置 1，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 64K）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 <- 0
JGE     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

描述 SR 中的负位 N 和溢位 V 被测试。如果只有一个被置 1，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 1 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 < TONI
JL      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

描述 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 被设置为 10。然后进行到标签 MAINLOOP 的跳转。低 64K 中的数据，完全存储器范围内的程序。

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

示例 Timer_A3 的中断向量 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 64K 内启动。

```
ADD     &TAIV,PC        ; Add Timer_A interrupt vector to PC
RETI                    ; No Timer_A interrupt pending
JMP     IHCCR1          ; Timer block 1 caused interrupt
JMP     IHCCR2          ; Timer block 2 caused interrupt
RETI                    ; 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

描述 SR 中的负位 N 被测试。如果它被置 1，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 64K 中的数据，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 -> R5
JN      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 R7
JN      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

运行 如果 C= 0：PC +（2 × 偏移）→PC
如果 C= 1：执行下一指令

描述 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 1，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 64K 中的数据，完全存储器范围内的程序。

```
CMP.B   #15,&EDE     ; Is EDE < 15? Info to C
JLO     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 -> C
JNC     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

描述 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 1，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 64K 中的数据，完全存储器范围内的程序。

```
CMP     #1500,&EDE      ; Is EDE = 1500? Info to SR
JNE     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 R7
JNZ     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

描述 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 64K）

```
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 64K 内。

```
        MOV     #EDE,R10                ; Prepare pointer (16-bit address)
Loop    MOV     @R10+,TOM-EDE-2(R10)    ; R10 points to both tables.
                                        ; R10+2
        CMP     #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 counter
Loop    MOV.B   @R10+,TOM-EDE-1(R10)    ; R10 points to both tables.
                                        ; R10+1
        DEC     R9                      ; Decrement counter
        JNZ     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
描述	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`

描述 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 R7
POP     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 stack
POP     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

描述 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 XXXXh
PUSH    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   xxXXh
PUSH.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

描述 被一个 CALL 指令压入堆栈的 16 位返回地址（低 64K）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 64K 内的子例程 SUBR 并且在 CALL 之后返回到低 64K 内的地址。

```
        CALL    #SUBR     ; Call subroutine starting at SUBR
        ...               ; Return by RET to here
SUBR    PUSH    R14       ; Save R14 (16 bit data)
        ...               ; Subroutine code
        POP     R14       ; Restore R14
        RET               ; Return to lower 64 K
```

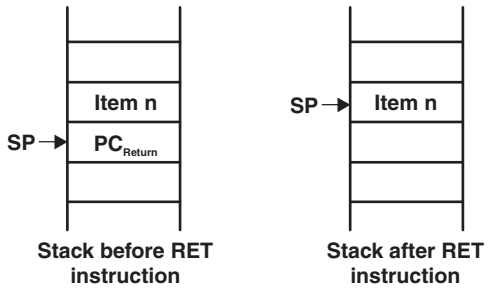


图 **4-37.** 一个 **RET** 指令之后的堆栈

4.6.2.37 RETI

RETI	从中断返回
句法	RETI
运行	@SP→SR.15:0 用 PC.19:16 恢复保存的 SR SP+2→SP @SP→PC.15:0 恢复保存的 PC.15:0 SP+2→SP 常规事务
描述	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（与状态位的堆栈位置相同）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。该命令未修改任何中断标志。
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 64K 中的中断处理程序。一个 20 位的返回地址被保存在堆栈中。

```
INTRPT  PUSHM.A   #2,R14    ; Save R14 and R13 (20-bit data)
        ...                 ; Interrupt handler code
        POPM.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`

描述 如图 4-38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。

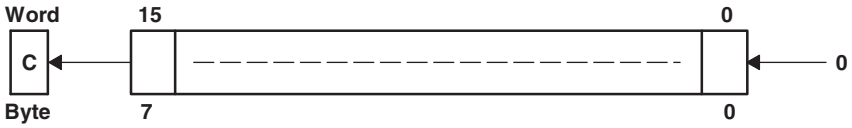


图 **4-38.** 目的操作数-算术左移位

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 1，如果为正则复位

Z： 如果结果为零则置 1，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 1；否则复位

如果一个算术溢出发生：初始值为 040h≤dst<0C0h，则置 1；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** 通过进位做选装

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`

描述 如图 4-39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)。

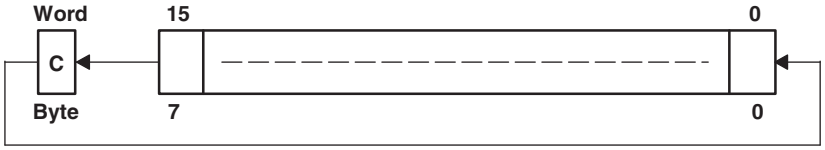


图 **4-39.** 目的操作数-进位左移位

状态位 N： 如果结果为负则置 1，如果为正则复位

Z： 如果结果为零则置 1，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 1；否则复位

如果一个算术溢出发生：初始值为 040h≤dst<0C0h，则置 1；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 -> Carry
RLC     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
描述	如图 4-40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 1，否则 (MSB=0)复位 Z： 如果结果为零则置 1，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
```

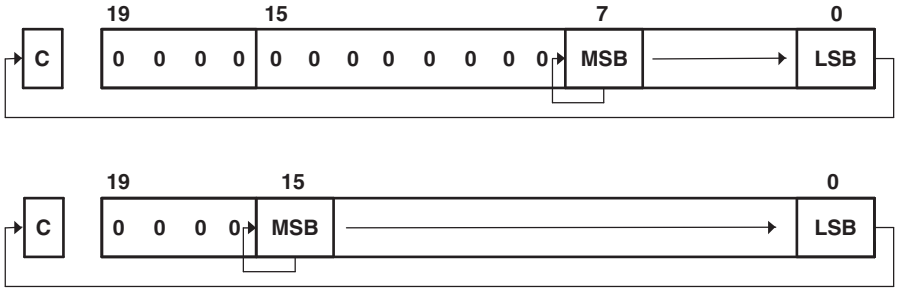


图 **4-40.** 算术右旋 **RRA.B** 和 **RRA.W**

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

描述 如图 4-41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 1，否则 (MSB=0)复位

Z： 如果结果为零则置 1，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 MSB
RRC     EDE     ; EDE = EDE >> 1 + 8000h
```

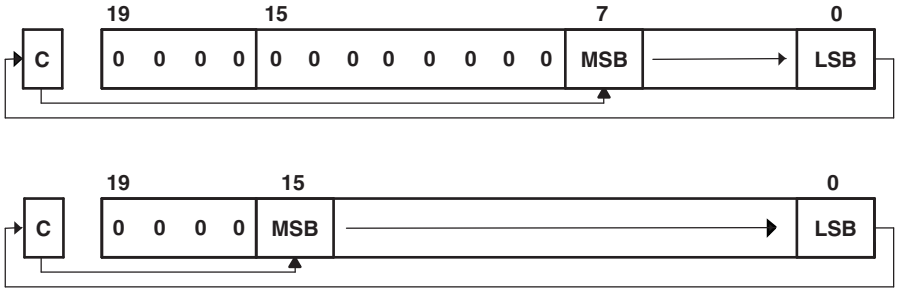


图 **4-41.** 通过进位 **RRC.B** 和 **RRC.W** 右旋

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`
描述 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 1，如果为正则复位
Z： 如果结果为零则置 1，否则复位
C： 如果有一个来自结果的 MSB 的进位，则置 1，否则复位
如果无借位则置 1，如果有借位则复位
V: 如果一个算术溢出发生，则置 1，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 LSDs
SBC     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 LSDs
SBC.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** 将进位位置 1

句法 SETC

运行 1→C

仿真 BIS #1M SR

描述 进位位 (C) 被置 1。

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 = 09FEDh
        INV     R5              ; Invert this (result back to 0-9)
                                ; R5 = .NOT. R5 = 06012h
        SETC                    ; Prepare carry = 1
        DADD    R5,R6           ; Emulate subtraction by addition of:
                                ; (010000h - R5 - 1)
                                ; R6 = R6 + R5 + 1
                                ; R6 = 0150h
```

4.6.2.44 SETN

*** SETN**	将负位置 1
句法	SETN
运行	1→N
仿真	BIS #4M SR
描述	复位 (N) 被置 1。
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**	将零位置 1
句法	SETZ
运行	1→N
仿真	BIS #2M SR
描述	零位 (Z) 被置 1。
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

描述 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 1，如果为正则复位 (src≤dst)
- Z： 如果为零 (src=dst) 则置 1，否则复位 (src≠dst)
- C： 如果有来自 MSB 的进位，则置 1，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 1，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 + 2
JZ      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

描述 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 1，如果为正 (MSB=0)，则复位
- Z： 如果结果为零则置 1，否则复位
- C： 如果有来自 MSB 的进位，则置 1，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 1，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 + 2
SUBC     @R5+,2(R7)      ; Subtract MIDs with C. R5 + 2
SUBC     @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 64K 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

描述 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 64K）

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

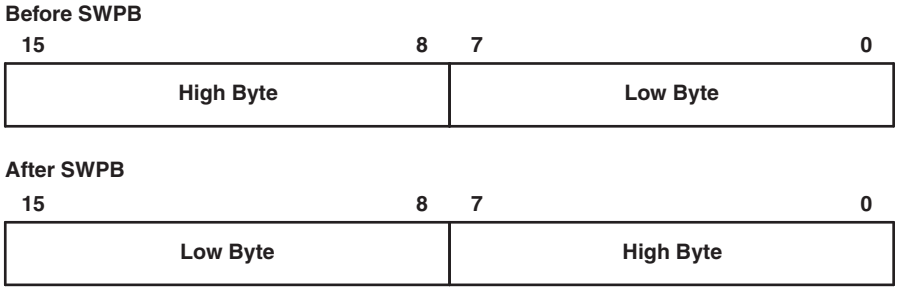


图 **4-42.** 交换存储器中的字节

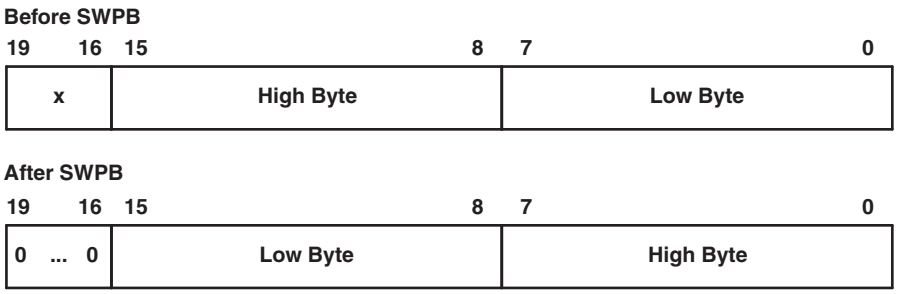


图 **4-43.** 交换寄存器中的字节

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）

描述 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 1，否则复位
Z： 如果结果为零则置 1，否则复位
C： 如果结果不为零则置 1，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 64K）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. 00XXh
SXT     R5           ; Sign extend low byte to R5.19:8
ADD     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. 00XXh
SXT     R5           ; Sign extend low byte to R5.19:8
ADDA    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`

描述 目的操作数与零相比较。根据结果设置状态位。目的操作数不受影响。

状态位 N： 如果目的操作数为负则置 1，如果为正则复位

Z： 如果目的操作数包含零则置 1，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 R7
            JN      R7NEG     ; R7 is negative
            JZ      R7ZERO    ; R7 is zero
R7POS       ......            ; R7 is positive but not zero
R7NEG       ......            ; R7 is negative
R7ZERO      ......            ; 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 R7
            JN      R7NEG     ; Low byte of R7 is negative
            JZ      R7ZERO    ; Low byte of R7 is zero
R7POS       ......            ; Low byte of R7 is positive but not zero
R7NEG       .....             ; Low byte of R7 is negative
R7ZERO      ......            ; 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

描述 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 1，如果为正 (MSB=0)，则复位

Z： 如果结果为零则置 1，否则复位

C： 如果结果不为零则置 1，否则复位 (C=.NOT. Z)

V： 如果两个操作数在执行前均为负则置 1，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 64K 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 拓展指令

扩展 MSP430X 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。下面将列出 MSP430 扩展指令并对其进行描述。

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`

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 (MSB=1)，则置 1，如果为正 (MSB=0)，则复位
Z： 如果结果为零则置 1，否则复位
C： 如果有一个来自结果的 MSB 的进位，则置 1，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 1，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 bits
ADCX.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

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。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 1，如果为正 (MSB=0)，则复位
- Z： 如果结果为零则置 1，否则复位
- C： 如果有一个来自结果的 MSB 的进位，则置 1，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 1，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 R6
JC       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: 000xxh
JNC      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,Rdst
ADDX.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

描述 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 1，如果为正 (MSB=0)，则复位
- Z： 如果结果为零则置 1，否则复位
- C： 如果有一个来自结果的 MSB 的进位，则置 1，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 1，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 R6
JC        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 + 1
JNC       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 (AND)
ANDX.B 源字节与目的字节的逻辑 AND

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

运行 src .and. dst→dst

描述 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。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 1，如果为正 (MSB=0)，则复位
Z： 如果结果为零则置 1，否则复位
C： 如果结果不为零则置 1，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 1 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 R5
ANDX.A   R5,TOM         ; TOM .and. R5 -> TOM
JZ       TONI           ; Jump if result 0
...                     ; Result > 0
```

或更短：

```
ANDX.A   #AAA55h,TOM    ; TOM .and. AAA55h -> TOM
JZ       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 1 的位
BICX.[W] 清零目的字中源字内置 1 的位
BICX.B 清零目的字节中源字节内置 1 的位

句法 `BICX.A srcMdst`

`BICX srcMdst` 或 `BICX.W srcMdst`

`BICX.B srcMdst`

运行 (.not. src) .and. dst→dst

描述 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 将在目的地址字中源地址字内置 1 的位置 1
BISX.[W] 置 1在目的字中源字内置 1的位
BISX.B 将在目的字节中源字节内置 1 的位置 1

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

运行 src .or. dst→dst

描述 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 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将 R7 中的位置 1。

```
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将输出 Port1 中的位置 1。

```
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 1 的位
BITX.[W] 测试在目的字中源字内置 1 的位
BITX.B 测试在目的字节中源字节内置 1 的位

句法 `BITX.A srcMdst`

`BITX srcMdst`或 `BITX.W srcMdst`

`BITX.B srcMdst`

运行 src .and. dst→dst

描述 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 1，如果为正 (MSB=0)，则复位
Z： 如果结果为零则置 1，否则复位
C： 如果结果不为零则置 1，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 1。如果被置 1 则跳转至标签。

```
BITX.A   #018000h,R5     ; Test R5.16:15 bits
JNZ      TONI            ; At least one bit is set
...                      ; Both 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 1，则跳转至标签 TONI。

```
BITX.W   @R5,R7          ; Test bits in R7: C = .not.Z
JC       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 1，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 + 1
JNC      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
```
描述 目的操作数被清零。

状态位 状态位不受影响。

模式位 OSCOFF，CPUOFF 和 GIE 不受影响。

示例 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

描述 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 1，如果为正则复位 (src≤dst)

Z： 如果为零 (src=dst) 则置 1，否则复位 (src≠dst)

C： 如果有来自 MSB 的进位，则置 1，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 1，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 18000h
JEQ      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 numbers
JL       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 + 1
JEQ      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,Rdst
CMPA     #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
```

描述 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 1，如果 MSB 为 0 则复位
- Z： 如果结果为零则置 1，否则复位
- C： 如果 BCD 结果太大（地址字 > 99999h，字 > 9999h，字节 > 99h）则置 1，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 bits
DADCX.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] 增加源字和十进制进位至目的字

DADDX.B 增加源字节和十进制进位至目的字节

句法

```
DADDX.A srcMdst
DADDX srcMdst或 DADDX.W srcMdst
DADDX.B src,dst
```

运行 src+dst+C→dst（用十进制）

描述 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 1，如果 MSB 为 0 则复位

Z： 如果结果为零则置 1，否则复位

C： 如果 BCD 结果太大（地址字 > 99999h，字 > 9999h，字节 > 99h）则置 1，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 carry
DADDX.W   BCD,R4            ; Add LSDs
DADDX.W   BCD+2,R5          ; Add MSDs with carry
JC        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 carry
DADDX.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`

描述 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 1，如果为正则复位
Z： 如果 dst 包含 1 则置 1，否则复位
C： 如果 dst 包含 0 则置 1，否则复位
V： 如果一个算术溢出发生则置 1，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
```

描述 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 1，如果为正则复位
- Z： 如果 dst 包含 2 则置 1，否则复位
- C： 如果 dst 包含 0 则复位，否则置 1
- V： 如果一个算术溢出发生则置 1，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`

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 1，如果为正则复位
Z： 如果 dst 包含 0FFFFFh 则置 1，否则复位
如果 dst 包含 0FFFFh 则置 1，否则复位
如果 dst 包含 0FFh 则置 1，否则复位
C： 如果 dst 包含 0FFFFFh 则置 1，否则复位
如果 dst 包含 0FFFFh 则置 1，否则复位
如果 dst 包含 0FFh 则置 1，否则复位
V： 如果 dst 包含 07FFFh 则置 1，否则复位
如果 dst 包含 07FFFh 则置 1，否则复位
如果 dst 包含 07Fh 则置 1，否则复位

模式位 OSCOFF，CPUOFF 和 GIE 不受影响。

示例 RAM 地址字 TON1 逐 1 递增。

```
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
```

描述 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 1，如果为正则复位
- Z： 如果 dst 包含 0FFFFEh 则置 1，否则复位
  如果 dst 包含 0FFFEh 则置 1，否则复位
  如果 dst 包含 0FEh 则置 1，否则复位
- C： 如果 dst 包含 0FFFFEh 或 0FFFFFh 则置 1，否则复位
  如果 dst 包含 0FFFEh 或 0FFFFh 则置 1，否则复位
  如果 dst 包含 0FEh 或 0FFh 则置 1，否则复位
- V： 如果 dst 包含 07FFFEh 或 07FFFFh 则置 1，否则复位
  如果 dst 包含 07FFEh 或 07FFFh 则置 1，否则复位
  如果 dst 包含 07Eh 或 07Fh 则置 1，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
*** INVX.[W]** 反转目的
*** INVX.B** 反转目的

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`

描述 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 1，如果为正则复位
Z： 如果 dst 包含 0FFFFFh 则置 1，否则复位
如果 dst 包含 0FFFFh 则置 1，否则复位
如果 dst 包含 0FFh 则置 1，否则复位
C： 如果结果不为零则置 1，否则复位 (=.NOT. 零)
V： 如果初始目的操作数为负则置 1，否则复位

模式位 OSCOFF，CPUOFF 和 GIE 不受影响。

示例 将 R5 的 20 位内容取反（两进制补码）。

```
INVX.A   R5      ; Invert R5
INCX.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 LEO
INCX.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+2
       CMPA     #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 counter
Loop   MOVX.W   @R10+,TOM-EDE-2(R10)   ; R10 points to both tables.
                                       ; R10+1
       DEC      R9                     ; Decrement counter
       JNZ      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/Reg
MOVX.A   #imm20,Rdst        MOVA   #imm20,Rdst      ; Immediate/Reg
MOVX.A   &abs20,Rdst        MOVA   &abs20,Rdst      ; Absolute/Reg
MOVX.A   @Rsrc,Rdst         MOVA   @Rsrc,Rdst       ; Indirect/Reg
MOVX.A   @Rsrc+,Rdst        MOVA   @Rsrc+,Rdst      ; Indirect,Auto/Reg
MOVX.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/Reg
MOVX.A   Rsrc,z20(Rdst)     MOVA   Rsrc,z16(Rdst)   ; Reg/Indexed
MOVX.A   symb20,Rdst        MOVA   symb16,Rdst      ; Symbolic/Reg
MOVX.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。

描述

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。	
描述	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`

运行 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`

描述 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 也逐 2 递增。

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`

运行 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）。

描述 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
运行	C←MSB←MSB-1 .... LSB+1←LSB←0
描述	如图 4-44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 1 .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 1，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
```

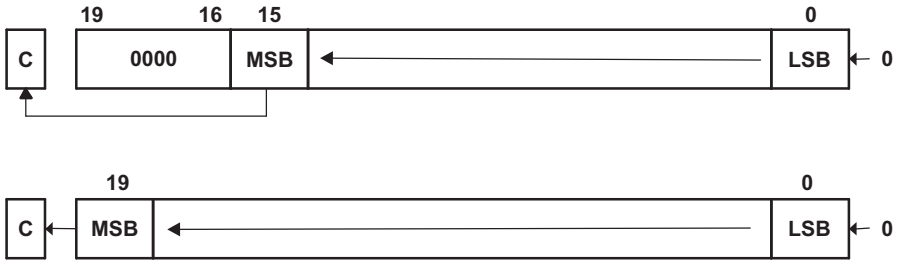


图 **4-44.** 用算术的方法左移 **- RLAM[.W]** 和 **RLAM.A**

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`

运行	C←MSB←MSB-1 .... LSB+1←LSB←0

仿真	`ADDX.A dst,dst`
	`ADDX dst,dst`
	`ADDX.B dst,dst`

描述	如图 4-45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。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 1，如果为正则复位 Z： 如果结果为零则置 1，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 1；否则复位 如果一个算术溢出发生：初始值为 04000h≤dst<0C000h，则置 1：否则复位 如果一个算术溢出发生：初始值为 040h≤dst<0C0h，则置 1；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)
```

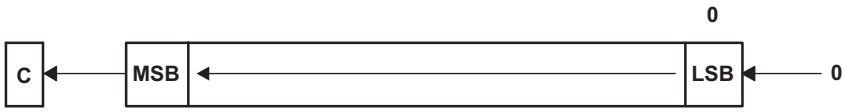


图 **4-45.** 目的操作数-算术左移位

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`

运行 C←MSB←MSB-1 .... LSB+1←LSB←C

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

描述 如图 4-46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 1，如果为正则复位
Z： 如果结果为零则置 1，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 1；否则复位
如果一个算术溢出发生：初始值为 04000h≤dst<0C000h，则置 1：否则复位
如果一个算术溢出发生：初始值为 040h≤dst<0C0h，则置 1；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)
```

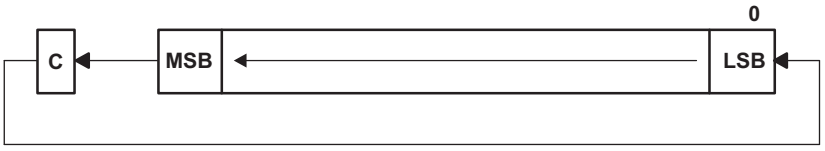


图 **4-46.** 目的操作数**-**进位左移位

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

运行 MSB→MSB→MSB-1→ ... LSB+1→LSB→C

描述 如图 4-47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： 如果结果为负则置 1

.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 1，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 stack
RRAM.A    #1,R15        ; R15 y 0.5 -> R15
ADDX.A    @SP+,R15      ; R15 y 0.5 + R15 = 1.5 y R15 -> R15
RRAM.A    #1,R15        ; (1.5 y R15) y 0.5 = 0.75 y R15 -> R15
```

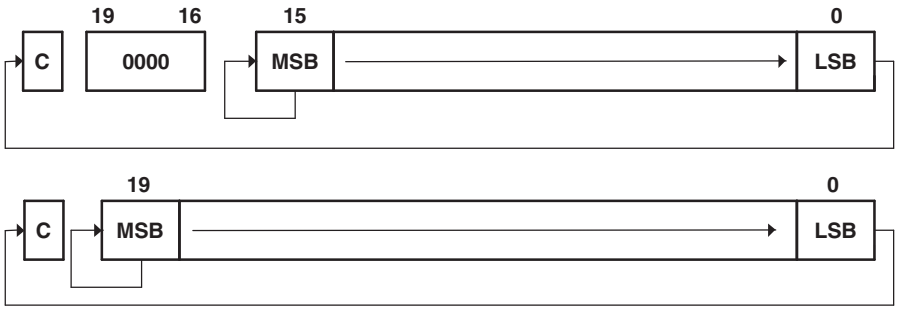


图 **4-47.** 算术右旋 **RRAM[.W]** 和 **RRAM.A**

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
```

运行 MSB→MSB→MSB-1→ ... LSB+1→LSB→C

描述 针对目的操作数的寄存器模式：如图 4-48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 4-49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 1，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 1，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     #4
RRAX.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
```

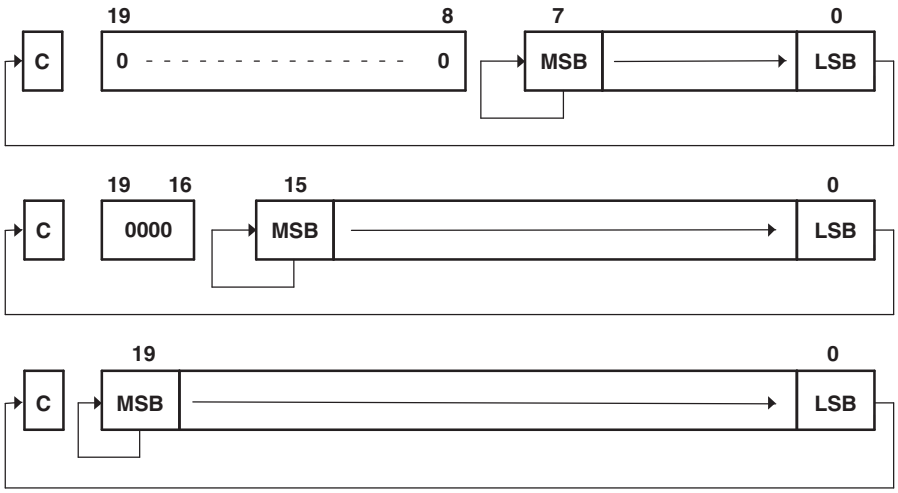


图 **4-48.** 算术右旋 **RRAX (.B，.A) -** 寄存器模式

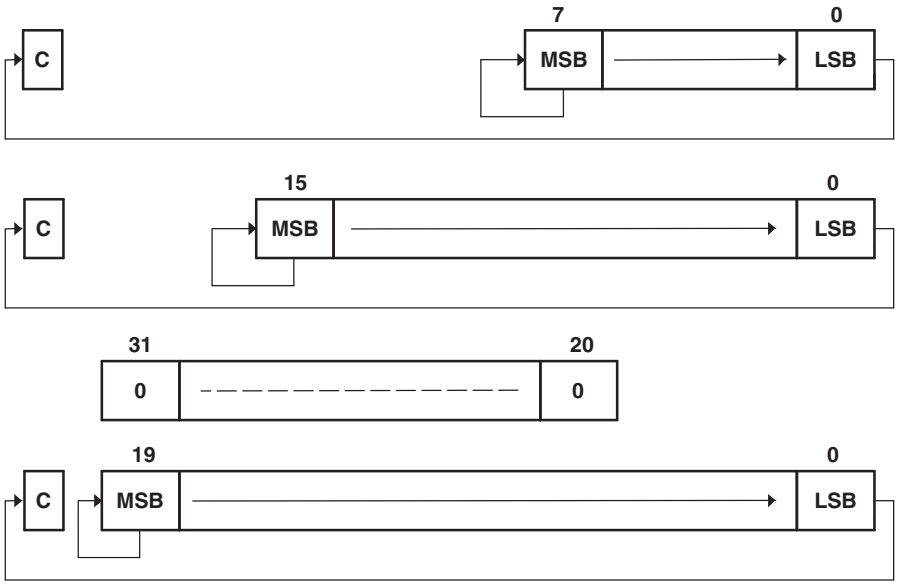


图 **4-49.** 算术右旋 **RRAX (.B，.A) -** 非寄存器模式

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
运行	C→MSB→MSB-1→ ... LSB+1→LSB→C	
描述	如图 4-50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： 如果结果为负则置 1 .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 1，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-2
RRCM.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
```

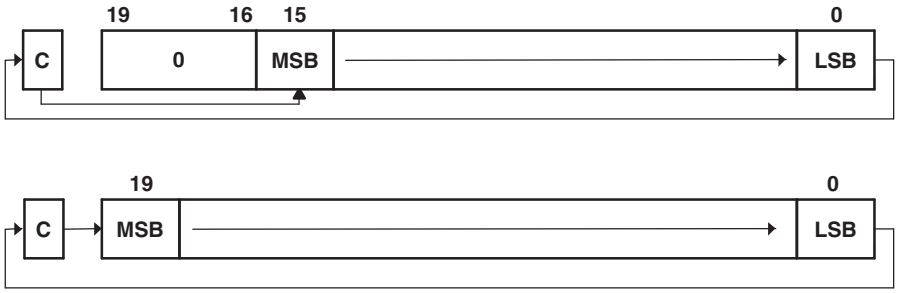


图 **4-50.** 通过进位 **RRCM[.W]** 和 **RRCM.A** 右旋

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
```

运行 C→MSB→MSB-1→ ... LSB+1→LSB→C

描述 针对目的操作数的寄存器模式：如图 4-51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 4-52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 1
  - .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 1，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 MSB
RRCX.A   EDE      ; EDE = EDE » 1 + 80000h
```

示例 R6 中的字被右移 12 个位置。

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

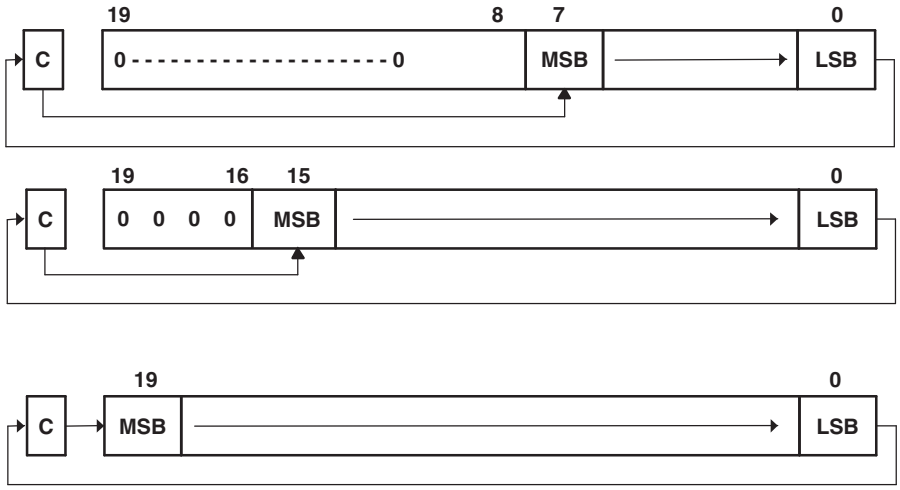


图 **4-51.** 通过进位 **RRCX (.B，.A)** 右旋-寄存器模式

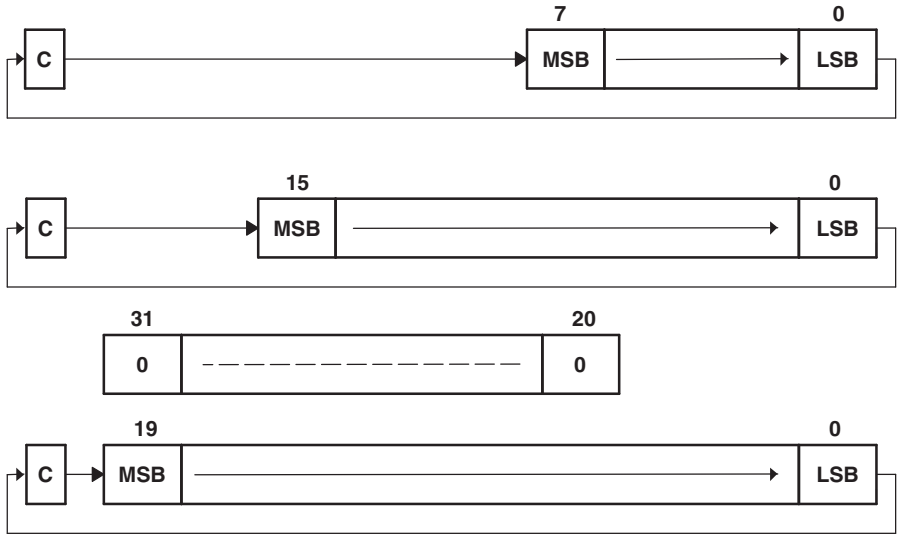


图 **4-52.** 通过进位 **RRCX (.B，.A)** 右旋-非寄存器模式

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

运行 0→MSB→MSB-1.... LSB+1→LSB→C

描述 如图 4-53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： 如果结果为负则置 1

.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 1，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
```

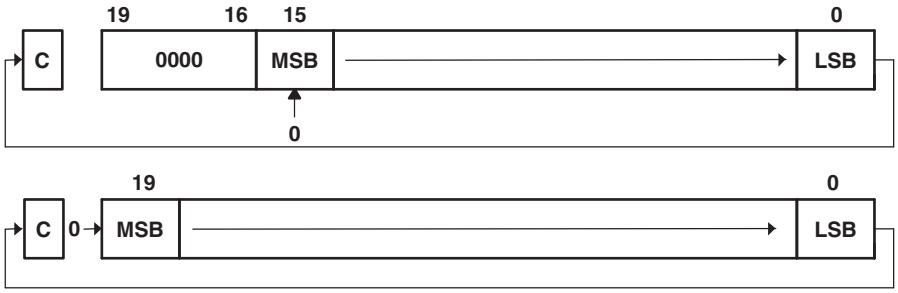


图 **4-53.** 右旋无符号 **RRUM[.W]** 和 **RRUM.A**

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
```

运行 C=0→MSB→MSB-1 ... LSB+1→LSB→C

描述 RRUX 只对寄存器模式有效：如图 4-54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 1
  - .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 1，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     #12
RRUX.W  R6      ; R6 = R6 » 12. R6.19:16 = 0
```

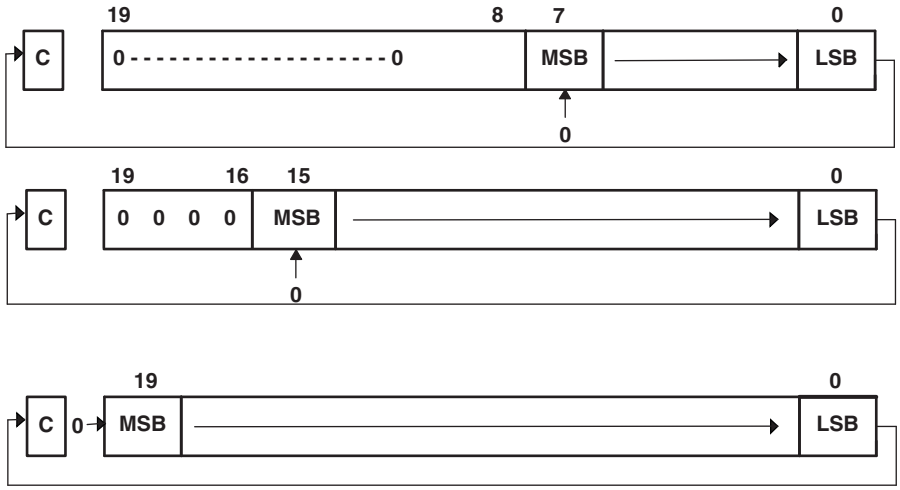


图 **4-54.** 右旋无符号 **RRUN (.B，.A) -** 寄存器模式

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
```

运行
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
```

描述 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 1，如果为正则复位
Z： 如果结果为零则置 1，否则复位
C： 如果有一个来自结果的 MSB 的进位，则置 1，否则复位
如果无借位则置 1，如果有借位则复位
V: 如果一个算术溢出发生，则置 1，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 LSDs
SBCX.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] 将源字从目的字中减去

SUBX.B 将源字节从目的字节中减去

句法

```
SUBX.A srcMdst
SUBX srcMdst 或 SUBX.W srcMdst
SUBX.B srcMdst
```

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

描述 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 1，如果为正则复位 (src≤dst)

Z： 如果为零 (src=dst) 则置 1，否则复位 (src≠dst)

C： 如果有来自 MSB 的进位，则置 1，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 1，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 + 2
JZ       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,Rdst
SUBX.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 srcMdst`
`SUBCX srcMdst`或`SUBCX.W srcMdst`
`SUBCX.B srcMdst`

运行 (.not. src)+C+dst→dst 或 dst-(src-1)+C→dst

描述 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 1，如果为正 (MSB=0)，则复位
Z： 如果结果为零则置 1，否则复位
C： 如果有来自 MSB 的进位，则置 1，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 1，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 + 2
SUBCX.W   @R5+,2(R7)      ; Subtract MIDs with C. R5 + 2
SUBCX.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`

运行 dst.15:8↔dst.7:0

描述 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 -> EDE
SWPBX.A   EDE                ; 25634h -> EDE
```

示例 交换 R5 的字节

```
MOVA      #23456h,R5         ; 23456h -> R5
SWPBX.W   R5                 ; 05634h -> R5
```

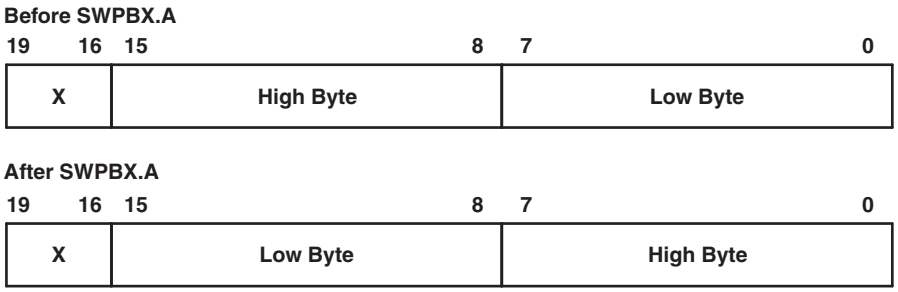


图 **4-55.** 交换字节 **SWPBX.A** 寄存器模式

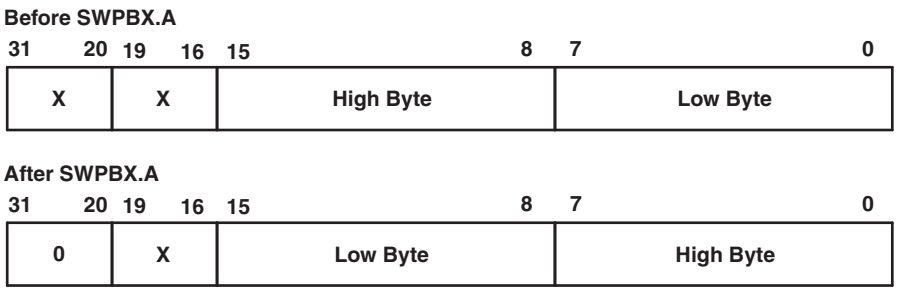


图 **4-56.** 在存储器中交换 **SWPBX.A** 字节

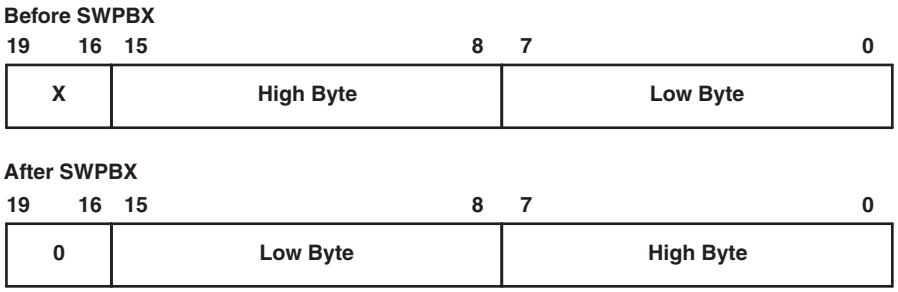


图 **4-57.** 交换字节 **SWPBX[.W]** 寄存器模式

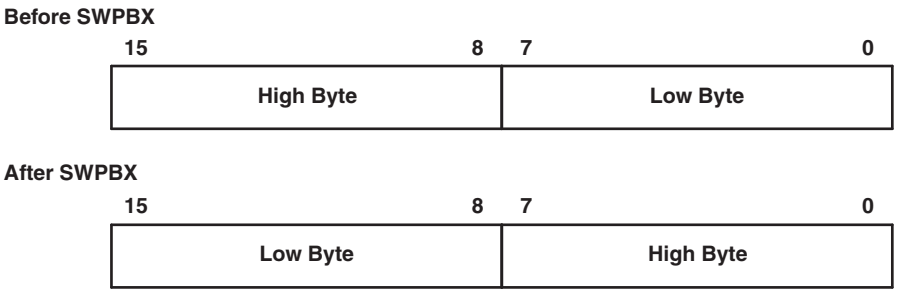


图 **4-58.** 在存储器中交换 **SWPBX[.W]** 字节

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`
运行	dst.7→dst.15:8，Rdst.7→Rdst.19:8（寄存器模式）
描述	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 1，否则复位 Z： 如果结果为零则置 1，否则复位 C： 如果结果不为零则置 1，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
```

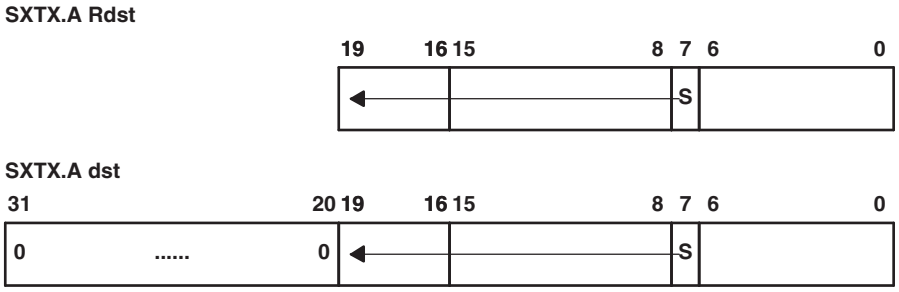


图 **4-59.** 符号扩展 **SXTX.A**

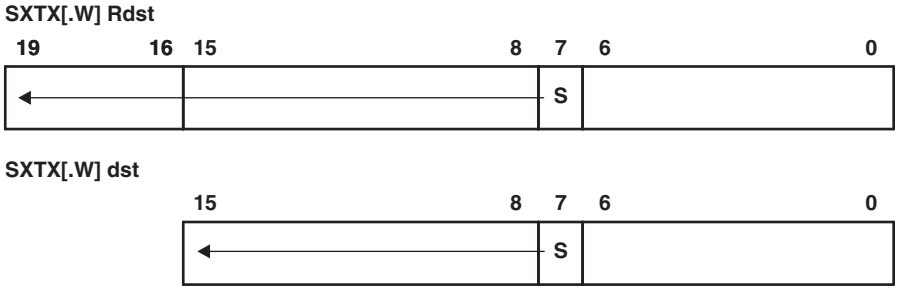


图 **4-60.** 符号扩展 **SXTX[.W]**

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
运行	dst+0FFFFFh+1 dst+0FFFFh+1 dst+0FFh+1
仿真	CMPX.A #0M dst CMPX #0M dst CMPX.B #0M dst
描述	目的操作数与零相比较。根据结果设置状态位。目的操作数不受影响。
状态位	N： 如果目的操作数为负则置 1，如果为正则复位 Z： 如果目的操作数包含零则置 1，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 LEO
            JN       LEONEG       ; LEO is negative
            JZ       LEOZERO      ; LEO is zero
LEOPOS      ......                ; LEO is positive but not zero
LEONEG      ......                ; LEO is negative
LEOZERO     ......                ; 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`

运行 src .xor. dst→dst

描述 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 1，如果为正 (MSB=0)，则复位
- Z： 如果结果为零则置 1，否则复位
- C： 如果结果不为零则置 1，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 1，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 R7
INV.B    R7              ; Invert low byte of R7. R7.19:8 = 0.
```

4.6.4 地址指令

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。下面将列出 MSP430X 地址指令并对其进行描述。

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`
运行	src+Rdst→Rdst
描述	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 1，如果为正 (Rdst.19=0)，则复位 Z： 如果结果为零则置 1，否则复位 C： 如果有来自 20 位结果的进位，则置 1，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 1，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 R5
JC      TONI             ; Jump on carry
...                      ; No carry occurred
```

4.6.4.2 BRA

*** BRA** 到目的的分指令

句法 `BRA dst`

运行 dst→PC

仿真 `MOVA dstMPC`

描述 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,PC
BRA      #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) 处，其中 X 的取值范围为 ±32 K。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。下次软件流使用 R5 作为一个指针，访问由 R5 指向表中的下一个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 的范围为 R5 ± 32 K。间接寻址，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

句法 CALLA dst

运行 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

描述 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 EXEC
CALLA   #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) 处，其中 X 的取值范围为 ±32 K。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。下次软件流使用 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 的范围为 R5 ± 32 K。间接寻址，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`
运行	0→Rdst
仿真	`MOVA #0M Rdst`
描述	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
```

运行 (.not. src)+1+Rdst 或 Rdst-src

描述 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 1，如果为正则复位 (src≤dst)
- Z： 如果为零 (src=dst) 则置 1，否则复位 (src≠dst)
- C： 如果有来自 MSB 的进位，则置 1，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 1，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 12345h
JEQ    EQUAL           ; R6 = 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`

运行 Rdst-2→Rdst

仿真 `SUBA #2M Rdst`

描述 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 1，如果为正则复位
- Z： 如果 Rdst 包含 2 则置 1，否则复位
- C： 如果 Rdst 包含 0 或 1 则复位，否则置 1
- V： 如果一个算术溢出发生则置 1，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`

运行 Rdst+2→Rdst

仿真 `ADDA #2M Rdst`

描述 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 1，如果为正则复位
- Z： 如果 Rdst 包含 0FFFFFEh 则置 1，否则复位
  如果 Rdst 包含 0FFFEh 则置 1，否则复位
  如果 Rdst 包含 0FEh 则置 1，否则复位
- C： 如果 Rdst 包含 0FFFFEh 或 0FFFFFh 则置 1，否则复位
  如果 Rdst 包含 0FFFEh 或 0FFFFh 则置 1，否则复位
  如果 Rdst 包含 0FEh 或 0FFh 则置 1，否则复位
- V： 如果 Rdst 包含 07FFFEh 或 07FFFFh 则置 1，否则复位
  如果 Rdst 包含 07FFEh 或 07FFFh 则置 1，否则复位
  如果 Rdst 包含 07Eh 或 07Fh 则置 1，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
```

运行
src→Rdst
Rsrc→dst

描述 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

运行 @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+M PC

描述 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 here
SUBR    PUSHM.A   #2,R14      ; Save R14 and R13 (20 bit data)
        ...                   ; Subroutine code
        POPM.A    #2,R14      ; Restore R13 and R14 (20 bit data)
        RETA                  ; Return (to full address space)
```

4.6.4.10 SUBA

SUBA 从 20 位目的寄存器中减去 20 位源。

句法
```
SUBA RsrcM Rdst
SUBA #imm20M Rdst
```

运行 (.not.src)+1+Rdst→Rdst 或 Rdst-src→Rdst

描述 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 1，如果为正则复位 (src≤dst)
- Z： 如果为零 (src=dst) 则置 1，否则复位 (src≠dst)
- C： 如果有来自 MSB (Rdst.19) 的进位，则置 1，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 1，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 -> R6
JC    TONI      ; Carry occurred
...             ; No carry
```

4.6.4.11 TSTA

*** TSTA** 测试 20 位目的寄存器

句法 `TSTA Rdst`

运行 dst+0FFFFFh+1
dst+0FFFFh+1
dst+0FFh+1

仿真 `CMPA #0M Rdst`

描述 目的寄存器与零相比较。根据结果设置状态位。目的寄存器不受影响。

状态位 N： 如果目的寄存器为负则置 1，如果为正则复位
Z： 如果目的寄存器包含零则置 1，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 R7
          JN     R7NEG       ; R7 is negative
          JZ     R7ZERO      ; R7 is zero
R7POS     ......             ; R7 is positive but not zero
R7NEG     ......             ; R7 is negative
R7ZERO    ......             ; R7 is zero
```

FRAM 控制器 (FRCTL)

本章对 FRAM 控制器的操作进行了说明。

Topic **Page**

5.1 FRAM 简介

FRAM 是一种非易失性存储器，其读取和写入的方式与标准 SRAM 类似。MSP430 FRAM 特性包括：

- 字节或字写访问
- 自动且可编程的等待状态控制，对于访问和周期时间具有独立的等待状态设置
- 错误校正码 (ECC)，支持位纠错、扩展位错误检测并且可通过标志指示错误
- 缓存数据以便快速读取
- 电源控制功能，可在不使用 FRAM 时将其禁用

图 5-1 显示了 FRAM 控制器的框图。

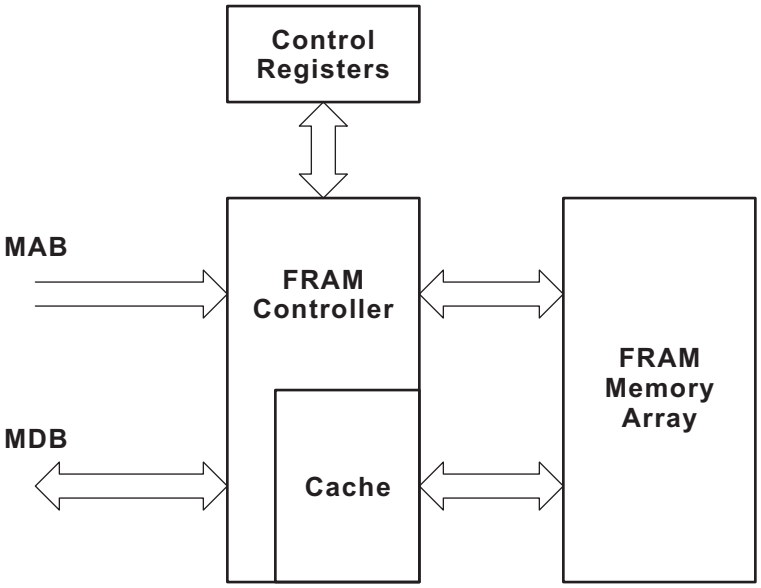


图 **5-1. FRAM** 控制器框图

5.2 FRAM 的结构

FRAM 的地址空间是线性的，但用户信息存储器和器件描述符信息 (TLV) 除外。

5.3 FRCTL 模块的工作方式

读取 FRAM 模块的方式与 SRAM 类似，没有特殊要求。

FRAM 的读取操作始终要求将同一读取信息写回同一存储器位置。此回写操作属于 FRAM 模块自身操作的一部分，无需与用户进行交互。此类回写操作不同于通过应用程序代码完成的正常写访问。

FRAM 模块内置错误校正码 (ECC) 逻辑，可进行位纠错并能检测多位错误。该逻辑提供两个标志，用于指示是否存在错误。

CBDIFG 在检测到可纠正的位错误时置 1。发生系统 NMI 事件 (SYSNMI) 时，CBDIE 也会置 1。

UBDIFG 在检测到不可纠正的多位错误时置 1。发生系统 NMI 事件 (SYSNMI) 时，UBDIE 也会置 1。

发生可纠正或不可纠正的位错误时，如果已使能 NMI，程序指向 SYSSNIV。如果需要，可通过将 UBDRSTEN 位置 1 生成系统复位事件 (SYSRST)。如果检测到不可纠正的错误，则启动 PUC 并且程序指向 SYSRSTIV。

5.4 对 FRAM 器件进行编程

可通过以下三种方式对 MSP430 FRAM 器件进行编程。所有方式都支持系统内编程。

- 通过 JTAG 或 Spy-Bi-Wire 接口编程
- 通过 BSL 编程
- 通过自定义解决方案编程

5.4.1 通过 JTAG 或 Spy-Bi-Wire 接口对 FRAM 进行编程

可通过 JTAG 端口或 Spy-Bi-Wire 端口对器件进行编程。JTAG 接口需要接入 TDI、TDO、TMS、TCK、TEST、接地引脚并可选择接入 VCC 和 $\overline{\text{RST}}$/NMI。Spy-Bi-Wire 接口需要接入 TEST、$\overline{\text{RST}}$/NMI、接地引脚并可选择接入 VCC。更多详细信息，请参见《*通过 JTAG 接口编程 MSP430 用户指南*》（文献编号：SLAU320）。

5.4.2 通过引导加载程序 (BSL) 对 FRAM 进行编程

每个器件在 ROM 中都存储有 BSL。BSL 支持用户使用 UART 串行接口读取 FRAM 或 RAM 或对其进行编程。通过 BSL 访问 FRAM 时受 256 位用户定义的密码保护。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)。

5.4.3 通过自定义解决方案对 FRAM 进行编程

CPU 能够对内部 FRAM 执行写入操作，因此可使用系统内部和外部的自定义编程解决方案。用户可以选择通过任何可用的方式（例如 UART 或 SPI）将数据发送至器件。用户开发的软件可接收数据并可对 FRAM 进行编程。由于此类解决方案由用户开发，因此可为完全自定义的解决方案，从而满足对 FRAM 进行编程或更新的应用需求。

5.5 等待状态控制

CPU 的系统时钟可能超出 FRAM 访问和周期时间要求。针对这些情况，可实施等待状态发生器机制。具体器件的数据表中，*建议运行条件*列出了频率范围以及所需的等待状态设置。等待状态的数量由 FRCTL0 寄存器中的 NWAITS[2:0] 位进行控制。

要增加系统时钟频率，使其超过当前等待状态设置允许的最大频率，需要执行以下步骤：

1. 通过根据目标频率配置 NWAITS[2:0] 来增加等待状态的数量。
2. 将频率增至新目标频率。

要将系统时钟频率降至支持较少等待状态的频率范围内，需按以下步骤进行操作：

1. 将频率降至新目标频率。
2. 根据新频率设置配置 NWAITS[2:0]，以此减少等待状态数。

为确保存储器的完整性，可实施一种在系统时钟频率以及等待状态设置违反 FRAM 访问时序时通过 PUC 将器件复位的机制。

注： **等待状态设置**

- 器件的起始等待状态为零。
- 必须确保等待状态设置正确，否则会生成 PUC 来避免不稳定的 FRAM 访问。

5.5.1 等待状态和缓存命中

FRAM 控制器的缓存具有两个缓存集。每个缓存集都包含两个缓存行，每个缓存行在一个访问周期中都预装载四个字（64 位）。智能逻辑选择其中一个缓存行来预装载 FRAM 数据并将最近访问的数据保存在另一缓存行。如果请求任一缓存行中存储的四个字中的任意一个字（缓存命中），则不会访问 FRAM；而是产生缓存请求。缓存请求无需等待状态，因此系统以全速访问数据。然而，如果请求的缓存中无任何字可用（缓存未命中），则等待状态控制 CPU，以确保对 FRAM 进行正确访问。

5.6 FRAM ECC

FRAM 支持位纠错以及不可纠正的位错误检测。如果 FRAM 错误检测逻辑检测到不可纠正的位错误，则 UBDIFG FRAM 不可纠正的位错误标志将置 1。如果检测到可纠正位错误并已进行纠正，则 CBDIFG FRAM 可纠正位错误标志置 1。如果检测到不可纠正的位错误，UBDRSTEN 使能上电清零 (PUC) 复位，而 UBDIE 使能 NMI 事件。如果检测到无足轻重的可纠正位错误并已进行纠正，CBDIE 将使能 NMI 事件。

5.7 FRAM 回写

FRAM 的所有读操作均需回写之前读取的内容。在任何情况下，都会执行这种回写操作，而无需与用户进行交互。

5.8 FRAM 电源控制

FRAM 控制器可禁用 FRAM 阵列的电源。设置 FRPWR = 0 后，FRAM 阵列电源被禁用，但仍能访问 FRAM 控制器中的寄存器。指向 FRAM 地址空间的存储器访问将 FRPWR = 1 自动复位并重新使能 FRAM 电源。第二个控制位 FRLPMPWR 用于在 LPM 退出后延迟 FRAM 上电操作。FRLPMPWR = 1 时，器件在退出 LPM 后直接激活 FRAM。FRLPMPWR = 0 将 FRAM 的激活延迟到对 FRAM 地址空间的第一次访问。对于 LPM0，LPM0 期间的 FRAM 电源状态由激活模式下的上一状态决定和记忆。如果 FRAM 电源已禁用，存储器访问自动插入等待状态，从而确保具有足够的时序进行 FRAM 上电和访问。可通过缓存进行处理的 FRAM 访问不会更改 FRAM 电源控制的电源状态。

PUC 复位强制状态机在 FRAM 使能的情况下激活。

图 5-2 显示了 FRAM 控制器的激活流程。

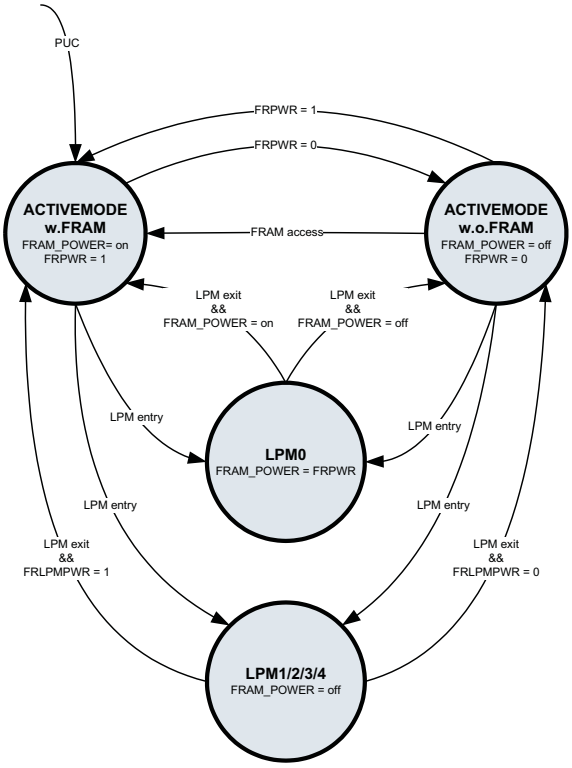


图 **5-2. FRAM** 电源控制示意图

5.9 FRAM 缓存

当 CPU 的运行速度高于 FRAM 支持的速度且没有等待状态时，FRAM 控制器可实现一种读缓存来实现速度优势。实现的这种缓存是一种 2 路关联缓存，具有 4 个缓存行，每个缓存行的容量为 64 位。如果存储器的连续地址处于同一缓存行时，可在没有等待状态的情况下对这些连续地址进行读访问。

5.10 FRCTL 寄存器

表 5-1 列出了 FRCTL 寄存器及其地址偏移量。关于 FRCTL 模块的基址，可参见具体器件的数据表。

FRCTL0 寄存器中定义的密码控制对所有 FRCTL 寄存器的访问。写入正确的密码后，将使能对寄存器的写访问。向 FRCTL 高字节写入字节模式的错误密码，将禁用写访问。使用错误的密码对 FRCTL 进行字访问将触发 PUC。在未使能写访问的情况下，对除 FRCTL 以外的寄存器进行写访问将导致 PUC。

注: 所有寄存器都支持字或字节访问。对于通用寄存器 *ANYREG* 而言，后缀“_L”(*ANYREG_L*) 表示寄存器的低字节（bit 0 到 bit 7）。后缀“_H”(*ANYREG_H*) 表示寄存器的高字节（bit 8 到 bit 15）。

表 5-1. FRCTL 寄存器

偏移量	首字母缩写词	寄存器名称	类型	访问	复位	部分
00h	FRCTL0	FRAM 控制器控制 0	读取/写入	字	9600h	5.10.1 节
00h	FRCTL0_L		读/写	字节	00h	
01h	FRCTL0_H		读/写	字节	96h	
04h	GCCTL0	通用控制 0	读取/写入	字	0006h	5.10.2 节
04h	GCCTL0_L		读/写	字节	06h	
05h	GCCTL0_H		读/写	字节	00h	
06h	GCCTL1	通用控制 1	读取/写入	字	0000h	5.10.3 节
06h	GCCTL1_L		读/写	字节	00h	
07h	GCCTL1_H		读/写	字节	00h	

5.10.1 FRCTL0 寄存器

FRAM 控制器控制寄存器 0

图 5-3. FRCTL0 寄存器

15	14	13	12	11	10	9	8
FRCTLPW							
rw	rw	rw	rw	rw	rw	rw	rw
7	6	5	4	3	2	1	0
保留	NWAITS			保留			
r-0	rw-[0]	rw-[0]	rw-[0]	r-0	r-0	r-0	r-0

表 5-2. FRCTL0 寄存器说明

位	字段	类型	复位	说明
15-8	FRCTLPW	RW	96h	FRCTLPW 密码。始终读为 96h。 要使能对 FRCTL 寄存器的写访问，写入 A5h。任何其他值的字写入都会导致 PUC。 写入正确的密码使能寄存器访问后，可通过写入字节模式的错误密码来禁用访问。在这种情况下，不会产生 PUC。
7	保留	R	0h	保留。始终读为 0。
6-4	NWAITS	RW	0h	等待状态控制。指定 FRAM 访问所需的等待状态数（0 到 7）（缓存丢失）。0 表示无等待状态。
3	保留	R	0h	保留。必须写为 0。
2-0	保留	R	0h	保留。始终读为 0。

5.10.2 GCCTL0 寄存器

通用控制寄存器 0

图 5-4. GCCTL0 寄存器

15	14	13	12	11	10	9	8
保留							
r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
7	6	5	4	3	2	1	0
UBDRSTEN	UBDIE	CBDIE	保留	保留	FRPWR	FRLPMPWR	保留
rw-[0]	rw-[0]	rw-[0]	r-0	rw - 0	rw - 1	rw-1	r-0

表 5-3. GCCTL0 寄存器说明

位	字段	类型	复位	说明
15-8	保留	R	0h	保留。始终读为 0。
7	UBDRSTEN	RW	0h	检测到 FRAM 不可纠正的位错误时，将使能 PUC 复位。 UBDRSTEN 位和 UBDIE 位彼此互斥，不允许同时置 1。一次只能选择处理一个错误。 0b = 出现不可纠正的位检测标志时不启动 PUC。 1b = 出现不可纠正的位检测标志时启动 PUC。在 SYSRSTIV 中生成向量。
6	UBDIE	RW	0h	检测到不可纠正的位错误时，将使能 NMI 事件。 UBDRSTEN 位和 UBDIE 位彼此互斥，不允许同时置 1。一次只能选择处理一个错误。 0b = 禁用不可纠正的位检测中断。 1b = 使能不可纠正的位检测中断。在 SYSSNIV 中生成向量。
5	CBDIE	RW	0h	检测到可纠正的位错误时，将使能 NMI 事件。 0b = 禁用可纠正的位检测中断。 1b = 使能可纠正的位检测中断。在 SYSSNIV 中生成向量。
4	保留	R	0h	保留。始终读为 0。
3	保留	RW	0h	保留。必须写为 0。
2	FRPWR	RW	1h	FRAM 电源控制。 写入该寄存器可使能或禁用 FRAM 电源。读取该寄存器会返回 FRAM 电源的实际状态，同时也会反映使能电源后的可能延迟。FRPWR = 1 表示 FRAM 电源已开启且准备就绪。 0b = 禁用 FRAM 电源 1b = 使能 FRAM 电源
1	FRLPMPWR	RW	1h	LPM 之后使能 FRAM 自动上电 0b = 退出 LPM 之后，FRAM 延迟到第一次 FRAM 访问时启动 1b = 退出 LPM 之后，FRAM 立即上电。
0	保留	R	0h	保留。始终读为 0。

5.10.3 GCCTL1 寄存器

通用控制寄存器 1

图 5-5. GCCTL1 寄存器

15	14	13	12	11	10	9	8
保留							
r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
7	6	5	4	3	2	1	0
保留				ACCTEIFG	UBDIFG	CBDIFG	保留
r-0	r-0	r-0	r-0	rw-0	rw-[0]	rw-[0]	r-0

表 5-4. GCCTL1 寄存器说明

位	字段	类型	复位	说明
15-3	保留	R	0h	保留。始终读为 0。
3	ACCTEIFG	RW	0h	访问时间错误标志。如果 NWAITS 设置有误且违反了 FRAM 访问时间，则该标志置 1 并产生复位 PUC。该位用软件清零，如果该位为最高挂起标志，还可通过读取系统复位向量字 SYSRSTIV 来清零。此位只可写 0，写 1 不起作用。 **注：**当系统频率配置为 8MHz 以上时，可在调试模式下将 ACCTEIFG 位置 1，与等待状态 (NWAITS) 无关。在这种情况下，并未违反 FRAM 访问。ACCTEIFG 位不会触发 PUC 或更改 SYSRSTIV 寄存器值。ACCTEIFG 位只能通过写 0 来清零。建议使用 SYSRESTIV 寄存器来检查 FRAM 访问违规错误，以避免产生混乱。
2	UBDIFG	RW	0h	FRAM 不可纠正的位错误标志。如果在 FRAM 存储器错误检测逻辑中检测到不可纠正的位错误，则该中断标志置 1。该位用软件清零，如果该位为最高挂起中断标志，还可通过读取系统 NMI 向量字 SYSSNIV 来清零。此位只可写 0，写 1 不起作用。 0b = 无中断挂起 1b = 中断挂起可由用户清零，或通过读取 SYSSNIV 来清零。
1	CBDIFG	RW	0h	FRAM 可纠正的位错误标志。如果在 FRAM 存储器错误检测逻辑中检测到可纠正的位错误且已将其纠正，则该中断标志置 1。该位用软件清零，如果该位为最高挂起中断标志，还可通过读取系统 NMI 向量字 SYSSNIV 来清零。此位只可写 0，写 1 不起作用。 0b = 无中断挂起 1b = 中断挂起可由用户清零，或通过读取 SYSSNIV 来清零
0	保留	R	0h	保留。始终读为 0。

备用存储器 (BKMEM)

备用存储器可在 LPM3.5 期间保存高达 256 个字节。其大小与具体器件相关，详细信息请参见具体器件的数据表。本章介绍了备用存储器的功能及特性。

Topic **Page**

6.1 备用存储器简介

备用存储器的特性包括：

- 可配置为 32 字节至 256 字节
- 支持 AM 到 LPM3.5 模式
- 支持字或字节访问

6.2 BKMEM 寄存器

表 6-1 列出了备用存储器寄存器。关于备用存储器模块模块的基址，可参见具体器件的数据表。

表 6-1. BKMEM 寄存器

偏移量	首字母缩写词	寄存器名称	类型	访问	复位
00h	BAKMEM0	备用存储器 0	读取/写入	字	未定义
00h	BAKMEM0_L		读取/写入	字节	
01h	BAKMEM0_H		读取/写入	字节	
02h	BAKMEM1	备用存储器 1(1)	读取/写入	字	未定义
02h	BAKMEM1_L		读取/写入	字节	
03h	BAKMEM1_H		读取/写入	字节	

(1) 字 2 至 127，如果可用，遵循相同格式。更多详细信息请参见具体器件的数据表。

数字 I/O

本章对所有器件内数字 I/O 端口的运行进行了说明。

Topic **Page**

7.1 数字 I/O 介绍

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。某些器件也许包含额外的端口中断。
- 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

系列内的器件也许有多达 12 个数字 I/O 端口被执行（P1 至 P11 以及 PJ）。大多数端口包含八条 I/O 线路；不过，也有部分端口的线路条数较少（关于可用端口，请参见具体器件的数据表。）每条 I/O 线路可单独配置为输入或输出方向，并且可单独进行读取或写入。每条 I/O 线路可单独配置上拉或下拉电阻。

端口 P1 和 P2 一直有中断功能。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 P1 I/O 线路共用一个中断向量 (P1IV)，而所有 P2 I/O 线路共用另一个中断向量 (P2IV)。器件可能提供具有中断功能的附加端口（有关详细信息，请参见具体器件的数据表），这些端口各自具有中断向量。

各端口可作为字节宽端口按字节访问，也可以组合成字宽端口按字访问。P1 和 P2、P3 和 P4、P5 和 P6、P7 和 P8 等端口对分别对应于 PA、PB、PC、PD 等名称。除中断向量寄存器 P1IV 和 P2IV 以外，所有端口寄存器均遵循这种命名约定，即不存在 PAIV。

当通过字操作写入端口 PA 时，全部 16 位均写入该端口。当通过字节操作写入端口 PA 的低字节时，高字节保持不变。同样，当通过字节指令写入端口 PA 的高字节时，低字节保持不变。当要写入的端口包含的位数少于可能的最大位数时，未使用的位为无关位。端口 PB、PC、PE 和 PF 的行为与端口 PA 类似。

通过字操作读取端口 PA 会将全部 16 位传送至目标地址。如果通过字节操作读取端口 PA（P1 或 P2）的低字节或高字节并将读取的字节存储于存储器中，则分别只会将低字节或高字节传送至目标地址。如果通过字节操作读取端口 PA 并将读取的字节存储于通用寄存器中，则会写入传送至寄存器最低有效字节的字节。目的寄存器的上部有效位被自动清零。端口 PB、PC、PE 和 PF 的行为与端口 PA 类似。当要读取的端口包含的位数少于可能的最大位数时，未使用的位读为零（端口 PJ 与之类似）。

7.2 数字 I/O 操作

数字 I/O 可通过用户软件进行配置。数字 I/O 的设置和操作在后续各小节进行讨论。

7.2.1 输入寄存器 (PxIN)

当引脚被配置为 I/O 功能时，每个 PxIN 寄存器中的每个位反映相应 I/O 引脚上输入信号的值。这些寄存器是只读的。

- 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。

7.2.2 输出寄存器 PxOUT)

当引脚被配置为 I/O 功能，输出方向时，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

如果引脚配置为 I/O 功能，将使能输入方向以及上拉或下拉电阻；PxOUT 寄存器中相应的位选择上拉或下拉。

- 位 = 0：引脚被下拉
- 位 = 1：引脚被上拉

7.2.3 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

7.2.4 上拉或下拉电阻器使能寄存器 (PxREN)

每个 PxREN 寄存器中的每个位用于使能或禁用相应 I/O 引脚的上拉或下拉电阻。PxOUT 寄存器中相应的位选择引脚是否包含上拉或下拉电阻。

- 位 = 0：禁用上拉或下拉电阻
- 位 = 1：使能上拉或下拉电阻

表 7-1 总结了使用 PxDIR、PxREN 和 PxOUT 进行正确 I/O 配置的方法。

表 7-1. I/O 配置

PxDIR	PxREN	PxOUT	I/O 配置
0	0	x	输入
0	1	0	具有下拉电阻器的输入
0	1	1	具有上拉电阻器的输入
1	x	x	输出

7.2.5 功能选择寄存器（PxSEL0 和 PxSEL1）

端口引脚通常与其它外设模块功能复用。请参见具体器件的数据表来确定引脚功能。每个端口引脚使用两个位选择引脚功能：I/O 端口或三种可能的外设模块功能之一。表 7-3 列出了选择各种模块功能的方法。请参见具体器件的数据表来确定引脚功能。每个 PxSEL 位用于选择引脚功能：I/O 端口或外设模块功能。该系列中的器件可能仅有 PxSEL0，也可能同时具有 PxSEL0 和 PxSEL1。

表 7-2. 只有一个 SEL 位的器件的 I/O 功能选择 – PxSEL0

PxSEL0	I/O 功能
0	选择通用 I/O
1	选择主要模块功能

表 7-3. 具有两个 SEL 位的器件的 I/O 功能选择 – PxSEL0 和 PxSEL1

PxSEL1	PxSEL0	I/O 功能
0	0	选择通用 I/O
0	1	选择主要模块功能
1	0	选择第二模块功能
1	1	选择第三模块功能

将 PxSEL1 或 PxSEL0 位设置为模块功能不会自动设置引脚方向。其它外设模块功能也许要求 PxDIR 位被按照模块功能所需的方向进行配置。请参阅具体器件数据表中的引脚电路原理图。

当选择端口引脚作为外设模块的输入时，相应外设模块的输入信号将呈现为锁存后的器件引脚信号。当 PxSEL1 和 PxSEL0 不为 00 时，所有已连接模块的内部输入信号均为器件引脚上的信号。但是，如果 PxSEL1 和 PxSEL0 为 00，则外设的输入将保持 PxSEL1 和 PxSEL0 位复位之前器件引脚的输入信号值。

由于 PxSEL1 和 PxSEL0 位的地址不连续，因此无法同时更改这两位。例如，应用可能需要将 P1.0 的功能由通用 I/O 更改为第三模块功能。初始条件下，P1SEL1 = 00h，P1SEL0 = 00h。要更改功能，需要写入 P1SEL1 = 01h 和 P1SEL0 = 01h。但如果不首先进行相应的中间配置，则无法实现，而且从应用角度来看，这种配置可能也并不理想。PxSELC 补充寄存器可用于处理此类情况。PxSELC 寄存器始终读为 0。PxSELC 寄存器中的每个置 1 位会补充 PxSEL1 和 PxSEL0 寄存器的相应位。在本示例中，初始条件为 P1SEL1 = 00h 且 P1SEL0 = 00h，写入 P1SELC = 01h 后将同时写入 P1SEL1 = 01h 和 P1SEL0 = 01h。

注： **PxSEL1 = 1 或 PxSEL0 = 1 时禁用中断**

当任意 PxSEL 位置 1 时，都会禁用对应的引脚中断功能。因此，无论相应的 PxIE 位状态如何，这些引脚上的信号都不会生成中断。

7.2.6 端口中断

通过 PxIFG、PxIE 和 PxIES 寄存器进行配置后，至少端口 P1 和 P2 中的每个引脚都具有中断功能。某些器件可能还提供除 P1 和 P2 以外的其他端口中断。请参见具体器件的数据表来确定器件提供哪些端口中断。

所有 Px 中断标志都进行了优先级排序（PxIFG.0 的优先级最高）且共用一个中断向量。优先级最高的已使能中断会在 PxIV 寄存器中生成一个数字。可评估该数字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 Px 中断不会影响 PxIV 值。PxIV 寄存器支持字或字节访问。

每个 PxIFG 位均为其对应 I/O 引脚的中断标志，当引脚上出现选定的输入信号边沿时，相应的标志置 1。当它们相应的 PxIE 位和 GIE 位被置 1 时，所有 PxIFG 中断标志请求一个中断。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 PxIFG 标志在中断服务程序期间或者 Px 中断服务程序的 RETI 指令被执行后置 1，则会生成另一个中断。这确保每个转换被确认。

注: 当改变 **PxOUT**，**PxDIR** 或 **PxREN** 时的 **PxIFG** 标志

写入 PxOUT、PxDIR 或 PxREN 会将相应的 PxIFG 标志置 1。

对 PxIV 寄存器低字节的任何访问（读/写访问，字/字节访问）都会自动复位最高挂起中断标志。如果有另一个中断标志置 1，则在处理完最初的中断后会立即产生另一个中断。

例如，假设 P1IFG.0 有最高的优先级。如果 P1IFG.0 和 P1IFG.2 标志被置 1，当中断处理例程访问 P1IV 寄存器时，P1IFG.0 会被自动复位。在中断处理例程的 RETI 指令被执行后，P1IFG.2 会生成另一个中断。

7.2.6.1 P1IV 软件示例

以下软件示例给出了建议的对 P1IV 和处理开销的使用方法。为了自动跳转到适当的例程，P1IV 值被添加到 PC。处理任何其他 PxIV 寄存器的代码是类似的。

最右侧的数字给出了每条指令所需的 CPU 周期数。不同中断源的软件开销包含中断延迟和中断返回周期，但不涉及任务处理本身。

```
;Interrupt handler for P1                                  Cycles
P1_HND   ...                ; Interrupt latency             6
         ADD      &P1IV,PC  ; Add offset to Jump table      3
         RETI               ; Vector  0: No interrupt       5
         JMP      P1_0_HND  ; Vector  2: Port 1 bit 0       2
         JMP      P1_1_HND  ; Vector  4: Port 1 bit 1       2
         JMP      P1_2_HND  ; Vector  6: Port 1 bit 2       2
         JMP      P1_3_HND  ; Vector  8: Port 1 bit 3       2
         JMP      P1_4_HND  ; Vector 10: Port 1 bit 4       2
         JMP      P1_5_HND  ; Vector 12: Port 1 bit 5       2
         JMP      P1_6_HND  ; Vector 14: Port 1 bit 6       2
         JMP      P1_7_HND  ; Vector 16: Port 1 bit 7       2

P1_7_HND                   ; Vector 16: Port 1 bit 7
         ...               ; Task starts here
         RETI              ; Back to main program          5

P1_6_HND                   ; Vector 14: Port 1 bit 6
         ...               ; Task starts here
         RETI              ; Back to main program          5

P1_5_HND                   ; Vector 12: Port 1 bit 5
         ...               ; Task starts here
         RETI              ; Back to main program          5

P1_4_HND                   ; Vector 10: Port 1 bit 4
         ...               ; Task starts here
         RETI              ; Back to main program          5

P1_3_HND                   ; Vector 8: Port 1 bit 3
         ...               ; Task starts here
         RETI              ; Back to main program          5

P1_2_HND                   ; Vector 6: Port 1 bit 2
         ...               ; Task starts here
         RETI              ; Back to main program          5

P1_1_HND                   ; Vector 4: Port 1 bit 1
```

```
          ...                 ; Task starts here
          RETI                ; Back to main program        5
P1_0_HND                      ; Vector 2: Port 1 bit 0
          ...                 ; Task starts here
          RETI                ; Back to main program        5
```

7.2.6.2 中断边沿选择寄存器 (PxIES)

每个 PxIES 位为相应的 I/O 引脚选择中断边沿。

- 位 = 0：各 PxIFG 标志在电平由低到高转换时置 1
- 位 = 1：各 PxIFG 标志在电平由高到低转换时置 1

注： 写入 **PxIES**

针对每个相应 I/O 对 P1ES 或 P2ES 的写入会将相应的中断标志置 1。

PxIES	PxIN	PxIFG
0→1	0	可被置 1
0→1	1	未改变
1→0	0	未改变
1→0	1	可被置 1

7.2.6.3 中断使能寄存器 (PxIE)

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

7.3 I/O 配置

7.3.1 复位后的配置

BOR 复位后，所有端口引脚均呈高阻态，并且将禁用施密特触发器及其模块功能，以免产生交叉电流。应用必须根据自身需要相应地配置 PxDIR、PxREN、PxOUT 和 PxIES，从而将包括未使用引脚（7.3.2 节）在内的所有引脚初始化为输入高阻抗、带下拉电阻的输入、带上拉电阻的输入、输出高电平或输出低电平等状态。该初始化在 PM5CTL 寄存器中的 LOCKLPM5 位（请参见 PMM 一章）清零时立即生效；在此之前，I/O 保持高阻态，施密特触发器输入处于禁用状态。请注意，该 I/O 初始化通常与从 LPMx.5 唤醒后所需的 I/O 初始化相同。LOCKLPM5 清零后，应将所有中断标志清零（请注意，该操作与从 LPMx.5 唤醒的流程不同）。之后可通过将相应 PxIE 位置 1 使能端口中断。

POR 或 PUC 复位之后，所有端口引脚均配置为输入，并且其模块功能被禁用。为了避免输入悬空，应根据应用需要在初始化过程中尽早配置所有端口引脚，包括未使用的引脚（7.3.2 节）。

请注意，除 PxIFG 之外，所有复位情况和从 LPMx.5 唤醒均可采用相同的 I/O 初始化过程：

1. 初始化端口：PxDIR、PxREN、PxOUT 和 PxIES
2. 清零 LOCKLPM5
3. 如果不是从 LPMx.5 唤醒：清零所有 PxIFG，避免产生错误的端口中断
4. 使能 PxIE 中的端口中断

7.3.2 未使用端口引脚的配置

为了避免输入悬空并降低功耗，未使用的 I/O 引脚应配置为 I/O 功能、输出方向并与 PC 主板保持未连接状态。由于引脚处于未连接状态，因此 PxOUT 位的值无关紧要。或者，也可以将未使用引脚的 PxREN 位置 1 来使能集成的上拉或下拉电阻，从而避免输入悬空。有关未使用引脚的端接信息，请参见*"系统复位，中断，工作模式，系统控制模块 (SYS)"*一章。

注： **配置端口 PJ 和共用的 JTAG 引脚：**

应用应确保端口 PJ 配置正确，从而防止输入悬空。由于端口 PJ 与 JTAG 功能共用，当在仿真环境中时，也许不会注意到悬空的输入。缺省情况下，端口 J 被初始化至高阻抗输入。

7.3.3 LPMx.5 低功耗模式的配置

注： 有关 LPMx.5 低功耗模式的详细信息，请参见1.4.3 节，*低功耗模式 LPM3.5 和 LPM4.5 (LPMx.5)*（在*系统复位，中断，工作模式，系统控制模块 (SYS)* 一章中）。

请参见具体器件的数据表以确定器件支持的 LPMx.5 低功耗模式以及能够在 LPM3.5 模式下工作的模块（如果有）。

对于数字 I/O，以下说明适用于 LPM3.5 和 LPM4.5。

进入 LPMx.5 模式（LPM3.5 或 LPM4.5）后会禁用 PMM 模块的 LDO，从而停止为器件内核供电。这将导致所有 I/O 寄存器配置丢失，因此必须相应地处理各个 I/O 引脚的配置，以确保应用中所有引脚在进入和退出 LPMx.5 后的行为均能得到控制。为了能够在 LPMx.5 下获得尽可能低的功耗并避免应用中出现输入或输出 I/O 状态失控情况，正确设置 I/O 引脚是至关重要的。应用能够完全控制 I/O 引脚的状态，从而防止器件在进入和退出 LPMx.5 后发生意外的虚假活动。

进入 LPMx.5 之前，需要为 I/O 执行以下操作：

(a) 将所有 I/O 设置为通用 I/O（PxSEL0 = 000h 且 PxSEL1 = 000h）并根据需要进行配置。每个 I/O 均可设置为输入高阻抗、带下拉电阻的输入、带上拉电阻的输入、输出高电平或输出低电平。务必确保应用中无输入悬空；否则可能导致 LPMx.5 下的电流消耗过大。

这样配置 I/O 可确保每个引脚在进入 LPMx.5 之前处于安全状态。

(b) 或者，也可以选择配置输入中断引脚从 LPMx.5 唤醒。要使器件从 LPMx.5 唤醒，通用 I/O 端口必须包含一个具有中断和唤醒功能的输入端口。并非所有带中断功能的输入均提供从 LPMx.5 唤醒的功能。请参见具体器件的数据表以确定器件是否具有唤醒功能。要唤醒器件，必须在进入 LPMx.5 之前正确配置端口引脚。每个端口都应配置为通用输入。如果需要的话，可采用下拉或上拉电阻器。相应寄存器 PxIES 位的设置将确定唤醒器件的边沿转换。最后，必须为端口使能 PxIE 并使能通用中断。

注： 如果相应的端口中断标志已置为有效，则无法通过端口中断将器件唤醒。德州仪器 (TI) 建议在进入 LPMx.5 之前清零这些标志。另外，德州仪器 (TI) 还建议在进入 LPMx.5 之前设置 GIE = 1。这样可使任何挂起标志在器件进入 LPMx.5 之前得到处理。

以上便是进入 LPMx.5 之前需要为 I/O 执行的操作。

在 LPMx.5 期间，I/O 引脚的状态将保持进入 LPMx.5 之前的设置并锁定。请注意，仅引脚状态保持不变。其他端口配置寄存器设置（例如，PxDIR、PxREN、PxOUT、PxIES 和 PxIE）的内容均丢失。

退出 LPMx.5 之后，所有外设寄存器均设为其默认状态，但 I/O 引脚保持锁定状态（LOCKLPM5 保持置 1）。I/O 引脚保持锁定状态可确保在器件进入工作模式时，无论默认 I/O 寄存器设置如何，所有引脚的状态都能保持稳定。

当器件返回工作模式时，I/O 配置和 I/O 中断配置（例如，PxDIR、PxREN、PxOUT 和 PxIES）应恢复为进入 LPMx.5 之前的值。然后可清零 LOCKLPM5 位，从而释放 I/O 引脚状态以及 I/O 中断配置。在 LOCKLPM5 置 1 时更改端口配置寄存器不会对 I/O 引脚造成任何影响。

通过配置 PxIE 使能 I/O 中断后，可按照 PxIFG 标志的指示处理引起唤醒的 I/O 中断。这些标志可被直接使用，或者可使用相应的 PxIV 寄存器。请注意，在 LOCKLPM5 位被清零时前，PxIFG 标志不能被清零。

注: 多个事件可出现在多个端口上。在这类情况下，会有多个 PxIFG 标志置 1，此时无法确定哪个端口引起 I/O 唤醒。

7.4 输入 I/O 寄存器

表 7-4中列出了数字 I/O 寄存器。基址可在具体器件的数据表中找到。每个端口分组都开始于其基址。在表 7-4中给出了地址偏移量。

注: 所有寄存器都支持字或字节访问。对于通用寄存器*ANYREG*来说，后缀“_L”(*ANYREG_L*) 是指寄存器的低字节（位 0 到 7）。后缀“_H”(*ANYREG_H*) 是指寄存器的高字节（位 8 到 15）。

表 7-4. 输入 I/O 寄存器

偏移量	首字母缩写词	寄存器名称	类型	访问	复位	部分
0Eh	P1IV	端口 1 的中断向量	只读	字	0000h	7.4.1 节
0Eh	P1IV_L		只读	字节	00h	
0Fh	P1IV_H		只读	字节	00h	
1Eh	P2IV	端口 2 的中断向量	只读	字	0000h	7.4.2 节
1Eh	P2IV_L		只读	字节	00h	
1Fh	P2IV_H		只读	字节	00h	
2Eh	P3IV	端口 3 中断向量	只读	字	0000h	7.4.3 节
2Eh	P3IV_L		只读	字节	00h	
2Fh	P3IV_H		只读	字节	00h	
3Eh	P4IV	端口 4 中断向量	只读	字	0000h	7.4.4 节
3Eh	P4IV_L		只读	字节	00h	
3Fh	P4IV_H		只读	字节	00h	
00h	P1IN 或 PAIN_L	端口 1 输入	只读	字节	未定义	7.4.5 节
02h	P1OUT 或 PAOUT_L	端口 1 输出	读取/写入	字节	未定义	7.4.6 节
04h	P1DIR 或 PADIR_L	端口 1 的方向	读取/写入	字节	00h	7.4.7 节
06h	P1REN 或 PAREN_L	端口 1 电阻器使能	读取/写入	字节	00h	7.4.8 节
0Ah	P1SEL0 或 PASEL0_L	端口 1 选择 0	读取/写入	字节	00h	7.4.9 节
0Ch	P1SEL1 或 PASEL1_L	端口 1 选择 1	读取/写入	字节	00h	7.4.10 节
16h	P1SELC 或 PASELC_L	端口 1 补充选择	读取/写入	字节	00h	7.4.11 节
18h	P1IES 或 PAIES_L	端口 1 的中断边沿选择	读取/写入	字节	未定义	7.4.12 节
1Ah	P1IE 或 PAIE_L	端口 1 的中断使能	读取/写入	字节	00h	7.4.13 节
1Ch	P1IE 或 PAIE_L	端口 1 的中断标志	读取/写入	字节	00h	7.4.14 节

表 7-4. 输入 I/O 寄存器 (continued)

偏移量	首字母缩写词	寄存器名称	类型	访问	复位	部分
01h	P2IN 或 PAIN_H	端口 2 的输入	只读	字节	未定义	7.4.5 节
03h	P2OUT 或 PAOUT_H	端口 2 的输出	读取/写入	字节	未定义	7.4.6 节
05h	P2DIR 或 PADIR_H	端口 2 的方向	读取/写入	字节	00h	7.4.7 节
07h	P2REN 或 PAREN_H	端口 2 的电阻器使能	读取/写入	字节	00h	7.4.8 节
0Bh	P2SEL0 或 PASEL0_H	端口 2 选择 0	读取/写入	字节	00h	7.4.9 节
0Dh	P2SEL1 或 PASEL1_H	端口 2 选择 1	读取/写入	字节	00h	7.4.10 节
17h	P2SELC 或 PASELC_L	端口 2 补充选择	读取/写入	字节	00h	7.4.11 节
19h	P2IES 或 PAIES_H	端口 2 的中断边沿选择	读取/写入	字节	未定义	7.4.12 节
1Bh	P2IE 或 PAIE_H	端口 2 的中断使能	读取/写入	字节	00h	7.4.13 节
1Dh	P2IFG 或 PAIFG_H	端口 2 的中断标志	读取/写入	字节	00h	7.4.14 节
00h	P3IN 或 PBIN_L	端口 3 的输入	只读	字节	未定义	7.4.5 节
02h	P3OUT 或 PBOUT_L	端口 3 的输出	读取/写入	字节	未定义	7.4.6 节
04h	P3DIR 或 PBDIR_L	端口 3 的方向	读取/写入	字节	00h	7.4.7 节
06h	P3REN 或 PBREN_L	端口 3 的电阻器使能	读取/写入	字节	00h	7.4.8 节
0Ah	P3SEL0 或 PBSEL0_L	端口 3 选择 0	读取/写入	字节	00h	7.4.9 节
0Ch	P3SEL1 或 PBSEL1_L	端口 3 选择 1	读取/写入	字节	00h	7.4.10 节
16h	P3SELC 或 PBSELC_L	端口 3 补充选择	读取/写入	字节	00h	7.4.11 节
18h	P3IES 或 PBIES_L	端口 3 中断边沿选择	读取/写入	字节	未定义	7.4.12 节
1Ah	P3IE 或 PBIE_L	端口 3 中断使能	读取/写入	字节	00h	7.4.13 节
1Ch	P3IFG 或 PBIFG_L	端口 3 中断标志	读取/写入	字节	00h	7.4.14 节

表 7-4. 输入 I/O 寄存器 (continued)

偏移量	首字母缩写词	寄存器名称	类型	访问	复位	部分
01h	P4IN 或 PBIN_H	端口 4 的输入	只读	字节	未定义	7.4.5 节
03h	P4OUT 或 PBOUT_H	端口 4 的输出	读取/写入	字节	未定义	7.4.6 节
05h	P4DIR 或 PBDIR_H	端口 4 的方向	读取/写入	字节	00h	7.4.7 节
07h	P4REN 或 PBREN_H	端口 4 的电阻器使能	读取/写入	字节	00h	7.4.8 节
0Bh	P4SEL0 或 PBSEL0_H	端口 4 选择 0	读取/写入	字节	00h	7.4.9 节
0Dh	P4SEL1 或 PBSEL1_H	端口 4 选择 1	读取/写入	字节	00h	7.4.10 节
17h	P4SELC 或 PBSELC_L	端口 4 补充选择	读取/写入	字节	00h	7.4.11 节
19h	P4IES 或 PBIES_H	端口 4 中断边沿选择	读取/写入	字节	未定义	7.4.12 节
1Bh	P4IE 或 PBIE_H	端口 4 中断使能	读取/写入	字节	00h	7.4.13 节
1Dh	P4IFG 或 PBIFG_H	端口 4 中断标志	读取/写入	字节	00h	7.4.14 节
00h	P5IN 或 PCIN_L	端口 5 的输入	只读	字节	未定义	7.4.5 节
02h	P5OUT 或 PCOUT_L	端口 5 的输出	读取/写入	字节	未定义	7.4.6 节
04h	P5DIR 或 PCDIR_L	端口 5 的方向	读取/写入	字节	00h	7.4.7 节
06h	P5REN 或 PCREN_L	端口 5 的电阻器使能	读取/写入	字节	00h	7.4.8 节
0Ah	P5SEL0 或 PCSEL0_L	端口 5 选择 0	读取/写入	字节	00h	7.4.9 节
0Ch	P5SEL1 或 PCSEL1_L	端口 5 选择 1	读取/写入	字节	00h	7.4.10 节
16h	P5SELC 或 PCSELC_L	端口 5 补充选择	读取/写入	字节	00h	7.4.11 节
18h	P5IES 或 PCIES_L	端口 5 中断边沿选择	读取/写入	字节	未定义	7.4.12 节
1Ah	P5IE 或 PCIE_L	端口 5 中断使能	读取/写入	字节	00h	7.4.13 节
1Ch	P5IFG 或 PCIFG_L	端口 5 中断标志	读取/写入	字节	00h	7.4.14 节

表 7-4. 输入 I/O 寄存器 (continued)

偏移量	首字母缩写词	寄存器名称	类型	访问	复位	部分
01h	P6IN 或 PCIN_H	端口 6 的输入	只读	字节	未定义	7.4.5 节
03h	P6OUT 或 PCOUT_H	端口 6 的输出	读取/写入	字节	未定义	7.4.6 节
05h	P6DIR 或 PCDIR_H	端口 6 的方向	读取/写入	字节	00h	7.4.7 节
07h	P6REN 或 PCREN_H	端口 6 的电阻器使能	读取/写入	字节	00h	7.4.8 节
0Bh	P6SEL0 或 PCSEL0_H	端口 6 选择 0	读取/写入	字节	00h	7.4.9 节
0Dh	P6SEL1 或 PCSEL1_H	端口 6 选择 1	读取/写入	字节	00h	7.4.10 节
17h	P6SELC 或 PCSELC_L	端口 6 补充选择	读取/写入	字节	00h	7.4.11 节
19h	P6IES 或 PCIES_H	端口 6 中断边沿选择	读取/写入	字节	未定义	7.4.12 节
1Bh	P6IE 或 PCIE_H	端口 6 中断使能	读取/写入	字节	00h	7.4.13 节
1Dh	P6IFG 或 PCIFG_H	端口 6 中断标志	读取/写入	字节	00h	7.4.14 节
00h	P7IN 或 PDIN_L	端口 7 的输入	只读	字节	未定义	7.4.5 节
02h	P7OUT 或 PDOUT_L	端口 7 的输出	读取/写入	字节	未定义	7.4.6 节
04h	P7DIR 或 PDDIR_L	端口 7 的方向	读取/写入	字节	00h	7.4.7 节
06h	P7REN 或 PDREN_L	端口 7 的电阻器使能	读取/写入	字节	00h	7.4.8 节
0Ah	P7SEL0 或 PDSEL0_L	端口 7 选择 0	读取/写入	字节	00h	7.4.9 节
0Ch	P7SEL1 或 PDSEL1_L	端口 7 选择 1	读取/写入	字节	00h	7.4.10 节
16h	P7SELC 或 PDSELC_L	端口 7 补充选择	读取/写入	字节	00h	7.4.11 节
18h	P7IES 或 PDIES_L	端口 7 中断边沿选择	读取/写入	字节	未定义	7.4.12 节
1Ah	P7IE 或 PDIE_L	端口 7 中断使能	读取/写入	字节	00h	7.4.13 节
1Ch	P7IFG 或 PDIFG_L	端口 7 中断标志	读取/写入	字节	00h	7.4.14 节

表 7-4. 输入 I/O 寄存器 (continued)

偏移量	首字母缩写词	寄存器名称	类型	访问	复位	部分
01h	P8IN 或 PDIN_H	端口 8 的输入	只读	字节	未定义	7.4.5 节
03h	P8OUT 或 PDOUT_H	端口 8 的输出	读取/写入	字节	未定义	7.4.6 节
05h	P8DIR 或 PDDIR_H	端口 8 的方向	读取/写入	字节	00h	7.4.7 节
07h	P8REN 或 PDREN_H	端口 8 的电阻器使能	读取/写入	字节	00h	7.4.8 节
0Bh	P8SEL0 或 PDSEL0_H	端口 8 选择 0	读取/写入	字节	00h	7.4.9 节
0Dh	P8SEL1 或 PDSEL1_H	端口 8 选择 1	读取/写入	字节	00h	7.4.10 节
17h	P8SELC 或 PDSELC_L	端口 8 补充选择	读取/写入	字节	00h	7.4.11 节
19h	P8IES 或 PDIES_H	端口 8 中断边沿选择	读取/写入	字节	未定义	7.4.12 节
1Bh	P8IE 或 PDIE_H	端口 8 中断使能	读取/写入	字节	00h	7.4.13 节
1Dh	P8IFG 或 PDIFG_H	端口 8 中断标志	读取/写入	字节	00h	7.4.14 节
00h	P9IN 或 PEIN_L	端口 9 的输入	只读	字节	未定义	7.4.5 节
02h	P9OUT 或 PEOUT_L	端口 9 的输出	读取/写入	字节	未定义	7.4.6 节
04h	P9DIR 或 PEDIR_L	端口 9 的方向	读取/写入	字节	00h	7.4.7 节
06h	P9REN 或 PEREN_L	端口 9 的电阻器使能	读取/写入	字节	00h	7.4.8 节
0Ah	P9SEL0 或 PESEL0_L	端口 9 选择 0	读取/写入	字节	00h	7.4.9 节
0Ch	P9SEL1 或 PESEL1_L	端口 9 选择 1	读取/写入	字节	00h	7.4.10 节
16h	P9SELC 或 PESELC_L	端口 9 补充选择	读取/写入	字节	00h	7.4.11 节
18h	P9IES 或 PEIES_L	端口 9 中断边沿选择	读取/写入	字节	未定义	7.4.12 节
1Ah	P9IE 或 PEIE_L	端口 9 中断使能	读取/写入	字节	00h	7.4.13 节
1Ch	P9IFG 或 PEIFG_L	端口 9 中断标志	读取/写入	字节	00h	7.4.14 节

表 7-4. 输入 I/O 寄存器 (continued)

偏移量	首字母缩写词	寄存器名称	类型	访问	复位	部分
01h	P10IN 或 PEIN_H	端口 10 的输入	只读	字节	未定义	7.4.5 节
03h	P10OUT 或 PEOUT_H	端口 10 的输出	读取/写入	字节	未定义	7.4.6 节
05h	P10DIR 或 PEDIR_H	端口 10 的方向	读取/写入	字节	00h	7.4.7 节
07h	P10REN 或 PEREN_H	端口 10 的电阻器使能	读取/写入	字节	00h	7.4.8 节
0Bh	P10SEL0 或 PESEL0_H	端口 10 选择 0	读取/写入	字节	00h	7.4.9 节
0Dh	P10SEL1 或 PESEL1_H	端口 10 选择 1	读取/写入	字节	00h	7.4.10 节
17h	P10SELC 或 PESELC_L	端口 10 补充选择	读取/写入	字节	00h	7.4.11 节
19h	P10IES 或 PEIES_H	端口 10 中断边沿选择	读取/写入	字节	未定义	7.4.12 节
1Bh	P10IE 或 PEIE_H	端口 10 中断使能	读取/写入	字节	00h	7.4.13 节
1Dh	P10IFG 或 PEIFG_H	端口 10 中断标志	读取/写入	字节	00h	7.4.14 节
00h	P11IN 或 PFIN_L	端口 11 的输入	只读	字节	未定义	7.4.5 节
02h	P11OUT 或 PFOUT_L	端口 11 的输出	读取/写入	字节	未定义	7.4.6 节
04h	P11DIR 或 PFDIR_L	端口 11 的方向	读取/写入	字节	00h	7.4.7 节
06h	P11REN 或 PFREN_L	端口 11 的电阻器使能	读取/写入	字节	00h	7.4.8 节
0Ah	P11SEL0 或 PFSEL0_L	端口 11 选择 0	读取/写入	字节	00h	7.4.9 节
0Ch	P11SEL1 或 PFSEL1_L	端口 11 选择 1	读取/写入	字节	00h	7.4.10 节
16h	P11SELC 或 PFSELC_L	端口 11 补充选择	读取/写入	字节	00h	7.4.11 节
18h	P11IES 或 PFIES_L	端口 11 中断边沿选择	读取/写入	字节	未定义	7.4.12 节
1Ah	P11IE 或 PFIE_L	端口 11 中断使能	读取/写入	字节	00h	7.4.13 节
1Ch	P11IFG 或 PFIFG_L	端口 11 中断标志	读取/写入	字节	00h	7.4.14 节

表 7-4. 输入 I/O 寄存器 (continued)

偏移量	首字母缩写词	寄存器名称	类型	访问	复位	部分
00h	PAIN	端口 A 的输入	只读	字	未定义	
00h	PAIN_L		只读	字节	未定义	
01h	PAIN_H		只读	字节	未定义	
02h	PAOUT	端口 A 的输出	读取/写入	字	未定义	
02h	PAOUT_L		读取/写入	字节	未定义	
03h	PAOUT_H		读取/写入	字节	未定义	
04h	PADIR	端口 A 的方向	读取/写入	字	0000h	
04h	PADIR_L		读取/写入	字节	00h	
05h	PADIR_H		读取/写入	字节	00h	
06h	PAREN	端口 A 的电阻使能	读取/写入	字	0000h	
06h	PAREN_L		读取/写入	字节	00h	
07h	PAREN_H		读取/写入	字节	00h	
0Ah	PASEL0	端口 A 选择 0	读取/写入	字	0000h	
0Ah	PASEL0_L		读取/写入	字节	00h	
0Bh	PASEL0_H		读取/写入	字节	00h	
0Ch	PASEL1	端口 A 选择 1	读取/写入	字	0000h	
0Ch	PASEL1_L		读取/写入	字节	00h	
0Dh	PASEL1_H		读取/写入	字节	00h	
16h	PASELC	端口 A 补充选择	读取/写入	字	0000h	
16h	PASELC_L		读取/写入	字节	00h	
17h	PASELC_H		读取/写入	字节	00h	
18h	PAIES	端口 A 的中断边沿选择	读取/写入	字	未定义	
18h	PAIES_L		读取/写入	字节	未定义	
19h	PAIES_H		读取/写入	字节	未定义	
1Ah	PAIE	端口 A 的中断使能	读取/写入	字	0000h	
1Ah	PAIE_L		读取/写入	字节	00h	
1Bh	PAIE_H		读取/写入	字节	00h	
1Ch	PAIFG	端口 A 的中断标志	读取/写入	字	0000h	
1Ch	PAIFG_L		读取/写入	字节	00h	
1Dh	PAIFG_H		读取/写入	字节	00h	

表 7-4. 输入 I/O 寄存器 (continued)

偏移量	首字母缩写词	寄存器名称	类型	访问	复位	部分
00h	PBIN	端口 B 的输入	只读	字	未定义	
00h	PBIN_L		只读	字节	未定义	
01h	PBIN_H		只读	字节	未定义	
02h	PBOUT	端口 B 的输出	读取/写入	字	未定义	
02h	PBOUT_L		读取/写入	字节	未定义	
03h	PBOUT_H		读取/写入	字节	未定义	
04h	PBDIR	端口 B 的方向	读取/写入	字	0000h	
04h	PBDIR_L		读取/写入	字节	00h	
05h	PBDIR_H		读取/写入	字节	00h	
06h	PBREN	端口 B 的电阻使能	读取/写入	字	0000h	
06h	PBREN_L		读取/写入	字节	00h	
07h	PBREN_H		读取/写入	字节	00h	
0Ah	PBSEL0	端口 B 选择 0	读取/写入	字	0000h	
0Ah	PBSEL0_L		读取/写入	字节	00h	
0Bh	PBSEL0_H		读取/写入	字节	00h	
0Ch	PBSEL1	端口 B 选择 1	读取/写入	字	0000h	
0Ch	PBSEL1_L		读取/写入	字节	00h	
0Dh	PBSEL1_H		读取/写入	字节	00h	
16h	PBSELC	端口 B 补充选择	读取/写入	字	0000h	
16h	PBSELC_L		读取/写入	字节	00h	
17h	PBSELC_H		读取/写入	字节	00h	
18h	PBIES	端口 B 中断边沿选择	读取/写入	字	未定义	
18h	PBIES_L		读取/写入	字节	未定义	
19h	PBIES_H		读取/写入	字节	未定义	
1Ah	PBIE	端口 B 中断使能	读取/写入	字	0000h	
1Ah	PBIE_L		读取/写入	字节	00h	
1Bh	PBIE_H		读取/写入	字节	00h	
1Ch	PBIFG	端口 B 中断标志	读取/写入	字	0000h	
1Ch	PBIFG_L		读取/写入	字节	00h	
1Dh	PBIFG_H		读取/写入	字节	00h	

表 7-4. 输入 I/O 寄存器 (continued)

偏移量	首字母缩写词	寄存器名称	类型	访问	复位	部分
00h	PCIN	端口 C 的输入	只读	字	未定义	
00h	PCIN_L		只读	字节	未定义	
01h	PCIN_H		只读	字节	未定义	
02h	PCOUT	端口 C 的输出	读取/写入	字	未定义	
02h	PCOUT_L		读取/写入	字节	未定义	
03h	PCOUT_H		读取/写入	字节	未定义	
04h	PCDIR	端口 C 的方向	读取/写入	字	0000h	
04h	PCDIR_L		读取/写入	字节	00h	
05h	PCDIR_H		读取/写入	字节	00h	
06h	PCREN	端口 C 的电阻使能	读取/写入	字	0000h	
06h	PCREN_L		读取/写入	字节	00h	
07h	PCREN_H		读取/写入	字节	00h	
0Ah	PCSEL0	端口 C 选择 0	读取/写入	字	0000h	
0Ah	PCSEL0_L		读取/写入	字节	00h	
0Bh	PCSEL0_H		读取/写入	字节	00h	
0Ch	PCSEL1	端口 C 选择 1	读取/写入	字	0000h	
0Ch	PCSEL1_L		读取/写入	字节	00h	
0Dh	PCSEL1_H		读取/写入	字节	00h	
16h	PCSELC	端口 C 补充选择	读取/写入	字	0000h	
16h	PCSELC_L		读取/写入	字节	00h	
17h	PCSELC_H		读取/写入	字节	00h	
18h	PCIES	端口 C 中断边沿选择	读取/写入	字	未定义	
18h	PCIES_L		读取/写入	字节	未定义	
19h	PCIES_H		读取/写入	字节	未定义	
1Ah	PCIE	端口 C 中断使能	读取/写入	字	0000h	
1Ah	PCIE_L		读取/写入	字节	00h	
1Bh	PCIE_H		读取/写入	字节	00h	
1Ch	PCIFG	端口 C 中断标志	读取/写入	字	0000h	
1Ch	PCIFG_L		读取/写入	字节	00h	
1Dh	PCIFG_H		读取/写入	字节	00h	

表 7-4. 输入 I/O 寄存器 (continued)

偏移量	首字母缩写词	寄存器名称	类型	访问	复位	部分
00h	PDIN	端口 D 的输入	只读	字	未定义	
00h	PDIN_L		只读	字节	未定义	
01h	PDIN_H		只读	字节	未定义	
02h	PDOUT	端口 D 的输出	读取/写入	字	未定义	
02h	PDOUT_L		读取/写入	字节	未定义	
03h	PDOUT_H		读取/写入	字节	未定义	
04h	PDDIR	端口 D 的方向	读取/写入	字	0000h	
04h	PDDIR_L		读取/写入	字节	00h	
05h	PDDIR_H		读取/写入	字节	00h	
06h	PDREN	端口 D 的电阻使能	读取/写入	字	0000h	
06h	PDREN_L		读取/写入	字节	00h	
07h	PDREN_H		读取/写入	字节	00h	
0Ah	PDSEL0	端口 D 选择 0	读取/写入	字	0000h	
0Ah	PDSEL0_L		读取/写入	字节	00h	
0Bh	PDSEL0_H		读取/写入	字节	00h	
0Ch	PDSEL1	端口 D 选择 1	读取/写入	字	0000h	
0Ch	PDSEL1_L		读取/写入	字节	00h	
0Dh	PDSEL1_H		读取/写入	字节	00h	
16h	PDSELC	端口 D 补充选择	读取/写入	字	0000h	
16h	PDSELC_L		读取/写入	字节	00h	
17h	PDSELC_H		读取/写入	字节	00h	
18h	PDIES	端口 D 中断边沿选择	读取/写入	字	未定义	
18h	PDIES_L		读取/写入	字节	未定义	
19h	PDIES_H		读取/写入	字节	未定义	
1Ah	PDIE	端口 D 中断使能	读取/写入	字	0000h	
1Ah	PDIE_L		读取/写入	字节	00h	
1Bh	PDIE_H		读取/写入	字节	00h	
1Ch	PDIFG	端口 D 中断标志	读取/写入	字	0000h	
1Ch	PDIFG_L		读取/写入	字节	00h	
1Dh	PDIFG_H		读取/写入	字节	00h	

表 7-4. 输入 I/O 寄存器 (continued)

偏移量	首字母缩写词	寄存器名称	类型	访问	复位	部分
00h	PEIN	端口 E 的输入	只读	字	未定义	
00h	PEIN_L		只读	字节	未定义	
01h	PEIN_H		只读	字节	未定义	
02h	PEOUT	端口 E 的输出	读取/写入	字	未定义	
02h	PEOUT_L		读取/写入	字节	未定义	
03h	PEOUT_H		读取/写入	字节	未定义	
04h	PEDIR	端口 E 的方向	读取/写入	字	0000h	
04h	PEDIR_L		读取/写入	字节	00h	
05h	PEDIR_H		读取/写入	字节	00h	
06h	PEREN	端口 E 的电阻使能	读取/写入	字	0000h	
06h	PEREN_L		读取/写入	字节	00h	
07h	PEREN_H		读取/写入	字节	00h	
0Ah	PESEL0	端口 E 选择 0	读取/写入	字	0000h	
0Ah	PESEL0_L		读取/写入	字节	00h	
0Bh	PESEL0_H		读取/写入	字节	00h	
0Ch	PESEL1	端口 E 选择 1	读取/写入	字	0000h	
0Ch	PESEL1_L		读取/写入	字节	00h	
0Dh	PESEL1_H		读取/写入	字节	00h	
16h	PESELC	端口 E 补充选择	读取/写入	字	0000h	
16h	PESELC_L		读取/写入	字节	00h	
17h	PESELC_H		读取/写入	字节	00h	
18h	PEIES	端口 E 中断边沿选择	读取/写入	字	未定义	
18h	PEIES_L		读取/写入	字节	未定义	
19h	PEIES_H		读取/写入	字节	未定义	
1Ah	PEIE	端口 E 中断使能	读取/写入	字	0000h	
1Ah	PEIE_L		读取/写入	字节	00h	
1Bh	PEIE_H		读取/写入	字节	00h	
1Ch	PEIFG	端口 E 中断标志	读取/写入	字	0000h	
1Ch	PEIFG_L		读取/写入	字节	00h	
1Dh	PEIFG_H		读取/写入	字节	00h	

表 7-4. 输入 I/O 寄存器 (continued)

偏移量	首字母缩写词	寄存器名称	类型	访问	复位	部分
00h	PFIN	端口 F 的输入	只读	字	未定义	
00h	PFIN_L		只读	字节	未定义	
01h	PFIN_H		只读	字节	未定义	
02h	PFOUT	端口 F 的输出	读取/写入	字	未定义	
02h	PFOUT_L		读取/写入	字节	未定义	
03h	PFOUT_H		读取/写入	字节	未定义	
04h	PFDIR	端口 F 的方向	读取/写入	字	0000h	
04h	PFDIR_L		读取/写入	字节	00h	
05h	PFDIR_H		读取/写入	字节	00h	
06h	PFREN	端口 F 的电阻使能	读取/写入	字	0000h	
06h	PFREN_L		读取/写入	字节	00h	
07h	PFREN_H		读取/写入	字节	00h	
0Ah	PFSEL0	端口 F 选择 0	读取/写入	字	0000h	
0Ah	PFSEL0_L		读取/写入	字节	00h	
0Bh	PFSEL0_H		读取/写入	字节	00h	
0Ch	PFSEL1	端口 F 选择 1	读取/写入	字	0000h	
0Ch	PFSEL1_L		读取/写入	字节	00h	
0Dh	PFSEL1_H		读取/写入	字节	00h	
16h	PFSELC	端口 F 补充选择	读取/写入	字	0000h	
16h	PFSELC_L		读取/写入	字节	00h	
17h	PFSELC_H		读取/写入	字节	00h	
18h	PFIES	端口 F 中断边沿选择	读取/写入	字	未定义	
18h	PFIES_L		读取/写入	字节	未定义	
19h	PFIES_H		读取/写入	字节	未定义	
1Ah	PFIE	端口 F 中断使能	读取/写入	字	0000h	
1Ah	PFIE_L		读取/写入	字节	00h	
1Bh	PFIE_H		读取/写入	字节	00h	
1Ch	PFIFG	端口 F 中断标志	读取/写入	字	0000h	
1Ch	PFIFG_L		读取/写入	字节	00h	
1Dh	PFIFG_H		读取/写入	字节	00h	

表 7-4. 输入 I/O 寄存器 (continued)

偏移量	首字母缩写词	寄存器名称	类型	访问	复位	部分
00h	PJIN	端口 J 的输入	只读	字	未定义	
00h	PJIN_L		只读	字节	未定义	
01h	PJIN_H		只读	字节	未定义	
02h	PJOUT	端口 J 的输出	读取/写入	字	未定义	
02h	PJOUT_L		读取/写入	字节	未定义	
03h	PJOUT_H		读取/写入	字节	未定义	
04h	PJDIR	端口 J 的方向	读取/写入	字	0000h	
04h	PJDIR_L		读取/写入	字节	00h	
05h	PJDIR_H		读取/写入	字节	00h	
06h	PJREN	端口 J 的电阻使能	读取/写入	字	0000h	
06h	PJREN_L		读取/写入	字节	00h	
07h	PJREN_H		读取/写入	字节	00h	
0Ah	PJSEL0	端口 J 选择 0	读取/写入	字	0000h	
0Ah	PJSEL0_L		读取/写入	字节	00h	
0Bh	PJSEL0_H		读取/写入	字节	00h	
0Ch	PJSEL1	端口 J 选择 1	读取/写入	字	0000h	
0Ch	PJSEL1_L		读取/写入	字节	00h	
0Dh	PJSEL1_H		读取/写入	字节	00h	
16h	PJSELC	端口 J 补充选择	读取/写入	字	0000h	
16h	PJSELC_L		读取/写入	字节	00h	
17h	PJSELC_H		读取/写入	字节	00h	

7.4.1 P1IV 寄存器

端口 1 的中断向量寄存器

图 7-1. P1IV 寄存器

15	14	13	12	11	10	9	8
P1IV							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
P1IV							
r0	r0	r0	r-0	r-0	r-0	r-0	r0

表 7-5. P1IV 寄存器说明

位	字段	类型	复位	说明
15-0	P1IV	R	0h	端口 1 的中断向量寄存器 00h = 无中断挂起 02h = 中断源：端口 1.0 中断；中断标志：P1IFG.0；中断优先级：最高 04h = 中断源：端口 1.1 中断；中断标志：P1IFG.1 06h = 中断源：端口 1.2 中断；中断标志：P1IFG.2 08h = 中断源：端口 1.3 中断；中断标志：P1IFG.3 0Ah = 中断源：端口 1.4 中断；中断标志：P1IFG.4 0Ch = 中断源：端口 1.5 中断；中断标志：P1IFG.5 0Eh = 中断源：端口 1.6 中断；中断标志：P1IFG.6 10h = 中断源：端口 1.7 中断；中断标志：P1IFG.7；中断优先级：最低

7.4.2 P2IV 寄存器

端口 2 的中断向量寄存器

图 7-2. P2IV 寄存器

15	14	13	12	11	10	9	8
P2IV							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
P2IV							
r0	r0	r0	r-0	r-0	r-0	r-0	r0

表 7-6. P2IV 寄存器说明

位	字段	类型	复位	说明
15-0	P2IV	R	0h	端口 2 的中断向量寄存器 00h = 无中断挂起 02h = 中断源：端口 2.0 中断；中断标志：P2IFG.0；中断优先级：最高 04h = 中断源：端口 2.1 中断；中断标志：P2IFG.1 06h = 中断源：端口 2.2 中断；中断标志：P2IFG.2 08h = 中断源：端口 2.3 中断；中断标志：P2IFG.3 0Ah = 中断源：端口 2.4 中断；中断标志：P2IFG.4 0Ch = 中断源：端口 2.5 中断；中断标志：P2IFG.5 0Eh = 中断源：端口 2.6 中断；中断标志：P2IFG.6 10h = 中断源：端口 2.7 中断；中断标志：P2IFG.7；中断优先级：最低

7.4.3 P3IV 寄存器

端口 3 中断向量寄存器

图 7-3. P3IV 寄存器

15	14	13	12	11	10	9	8
P3IV							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
P3IV							
r0	r0	r0	r-0	r-0	r-0	r-0	r0

表 7-7. P3IV 寄存器说明

位	字段	类型	复位	说明
15-0	P3IV	R	0h	端口 3 中断向量值 00h = 无中断挂起 02h = 中断源：端口 3.0 中断；中断标志：P3IFG.0；中断优先级：最高 04h = 中断源：端口 3.1 中断；中断标志：P3IFG.1 06h = 中断源：端口 3.2 中断；中断标志：P3IFG.2 08h = 中断源：端口 3.3 中断；中断标志：P3IFG.3 0Ah = 中断源：端口 3.4 中断；中断标志：P3IFG.4 0Ch = 中断源：端口 3.5 中断；中断标志：P3IFG.5 0Eh = 中断源：端口 3.6 中断；中断标志：P3IFG.6 10h = 中断源：端口 3.7 中断；中断标志：P3IFG.7；中断优先级：最低

7.4.4 P4IV 寄存器

端口 4 中断向量寄存器

图 7-4. P4IV 寄存器

15	14	13	12	11	10	9	8
P4IV							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
P4IV							
r0	r0	r0	r-0	r-0	r-0	r-0	r0

表 7-8. P4IV 寄存器说明

位	字段	类型	复位	说明
15-0	P4IV	R	0h	端口 4 中断向量值 00h = 无中断挂起 02h = 中断源：端口 4.0 中断；中断标志：P4IFG.0；中断优先级：最高 04h = 中断源：端口 4.1 中断；中断标志：P4IFG.1 06h = 中断源：端口 4.2 中断；中断标志：P4IFG.2 08h = 中断源：端口 4.3 中断；中断标志：P4IFG.3 0Ah = 中断源：端口 4.4 中断；中断标志：P4IFG.4 0Ch = 中断源：端口 4.5 中断；中断标志：P4IFG.5 0Eh = 中断源：端口 4.6 中断；中断标志：P4IFG.6 10h = 中断源：端口 4.7 中断；中断标志：P4IFG.7；中断优先级：最低

7.4.5 PxIN 寄存器

端口 x 的输入寄存器

图 7-5. PxIN 寄存器

7	6	5	4	3	2	1	0
PxIN							
r	r	r	r	r	r	r	r

表 7-9. PxIN 寄存器说明

位	字段	类型	复位	说明
7-0	PxIN	R	未定义	端口 x 输入 0b = 输入为低电平 1b = 输入为高电平

7.4.6 PxOUT 寄存器

端口 x 的输出寄存器

图 7-6. PxOUT 寄存器

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

表 7-10. PxOUT 寄存器说明

位	字段	类型	复位	说明
7-0	PxOUT	RW	未定义	端口 x 的输出 当 I/O 被配置为输出模式时： 0b = 输出为低电平。 1b = 输出为高电平。 当 I/O 被配置为输入模式且被使能上拉/下拉时： 0b = 选择下拉 1b = 选择下拉

7.4.7 PxDIR 寄存器

端口 x 的方向寄存器

图 7-7. PxDIR 寄存器

7	6	5	4	3	2	1	0
PxDIR							
rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0

表 7-11. P1DIR 寄存器说明

位	字段	类型	复位	说明
7-0	PxDIR	RW	0h	端口 x 的方向 0b = 端口被配置为输入 1b = 端口被配置为输出

7.4.8 PxREN 寄存器

端口 x 上拉或下拉电阻使能寄存器

图 7-8. PxREN 寄存器

7	6	5	4	3	2	1	0
PxREN							
rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0

表 7-12. PxREN 寄存器说明

位	字段	类型	复位	说明
7-0	PxREN	RW	0h	端口 x 上拉或下拉电阻使能。当端口配置为输入时，可通过设置该位使能或禁用上拉或下拉电阻。 0b = 上拉或下拉被禁用 1b = 上拉或下拉被使能

7.4.9 PxSEL0 寄存器

端口 x 功能选择寄存器 0

图 7-9. PxSEL0 寄存器

7	6	5	4	3	2	1	0
PxSEL0							
rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0

表 7-13. PxSEL0 寄存器说明

位	字段	类型	复位	说明
7-0	PxSEL0	RW	0h	端口功能选择。每个位对应端口 x 上的一条通道。 搭配使用 PxSEL1 和 PxSEL0 中每个位的值来指定功能。例如，如果 P1SEL1.5 = 1 且 P1SEL0.5 = 0，则为 P1.5 选择第二模块功能。 关于每个值的定义，请参见 PxSEL1。

7.4.10 PxSEL1 寄存器

端口 x 功能选择寄存器 1

图 7-10. PxSEL1 寄存器

7	6	5	4	3	2	1	0
PxSEL1							
rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0

表 7-14. PxSEL1 寄存器说明

位	字段	类型	复位	说明
7-0	PxSEL1	RW	0h	端口功能选择。每个位对应端口 x 上的一条通道。 搭配使用 PxSEL1 和 PxSEL0 中每个位的值来指定功能。例如，如果 P1SEL1.5 = 1 且 P1SEL0.5 = 0，则为 P1.5 选择第二模块功能。 00b = 选择通用 I/O 01b = 选择主要模块功能 10b = 选择第二模块功能 11b = 选择第三模块功能

7.4.11 PxSELC 寄存器

端口 x 补充选择

图 7-11. PxSELC 寄存器

7	6	5	4	3	2	1	0
PxSELC							
rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0

表 7-15. PxSELC 寄存器说明

位	字段	类型	复位	说明
7-0	PxSELC	RW	0h	端口选择补充。 PxSELC 中每个置 1 的位都会补充 PxSEL1 和 PxSEL0 寄存器中对应的位；即，当 PxSELC 中的任一位置 1 时，PxSEL1 和 PxSEL0 中对应的位也会同时发生改变。始终读为 0。

7.4.12 PxIES 寄存器

端口 x 中断边沿选择寄存器

图 7-12. PxIES 寄存器

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

表 7-16. PxIES 寄存器说明

位	字段	类型	复位	说明
7-0	PxIES	RW	未定义	端口 x 中断边沿选择 0b = PxIFG 标志在电平由低到高转换时置 1 1b = PxIFG 标志在电平由高到低转换时置 1

7.4.13 PxIE 寄存器

端口 x 中断使能寄存器

图 7-13. PxIE 寄存器

7	6	5	4	3	2	1	0
PxIE							
rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0

表 7-17. PxIE 寄存器说明

位	字段	类型	复位	说明
7-0	PxIE	RW	0h	端口 x 中断使能 0b = 相应端口的中断被禁止 1b = 相应端口的中断被使能

7.4.14 PxIFG 寄存器

端口 x 中断标志寄存器

图 7-14. PxIFG 寄存器

7	6	5	4	3	2	1	0
PxIFG							
rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0

表 7-18. PxIFG 寄存器说明

位	字段	类型	复位	说明
7-0	PxIFG	RW	0h	端口 x 中断标志 0b = 无中断挂起。 1b = 中断挂起。

电容式触摸 IO

本章介绍了电容式触摸 IO 的功能及相关控制。

Topic **Page**

8.1 电容式触摸 IO 简介

通过电容式触摸 IO 模块可以实现简单的电容式触摸感测应用。该模块使用集成上拉和下拉电阻以及外部电容，通过将输入施密特触发器感测到的反向输入电压反馈给上拉和下拉控制构成振荡器。图 8-1 给出了电容式触摸 IO 原理

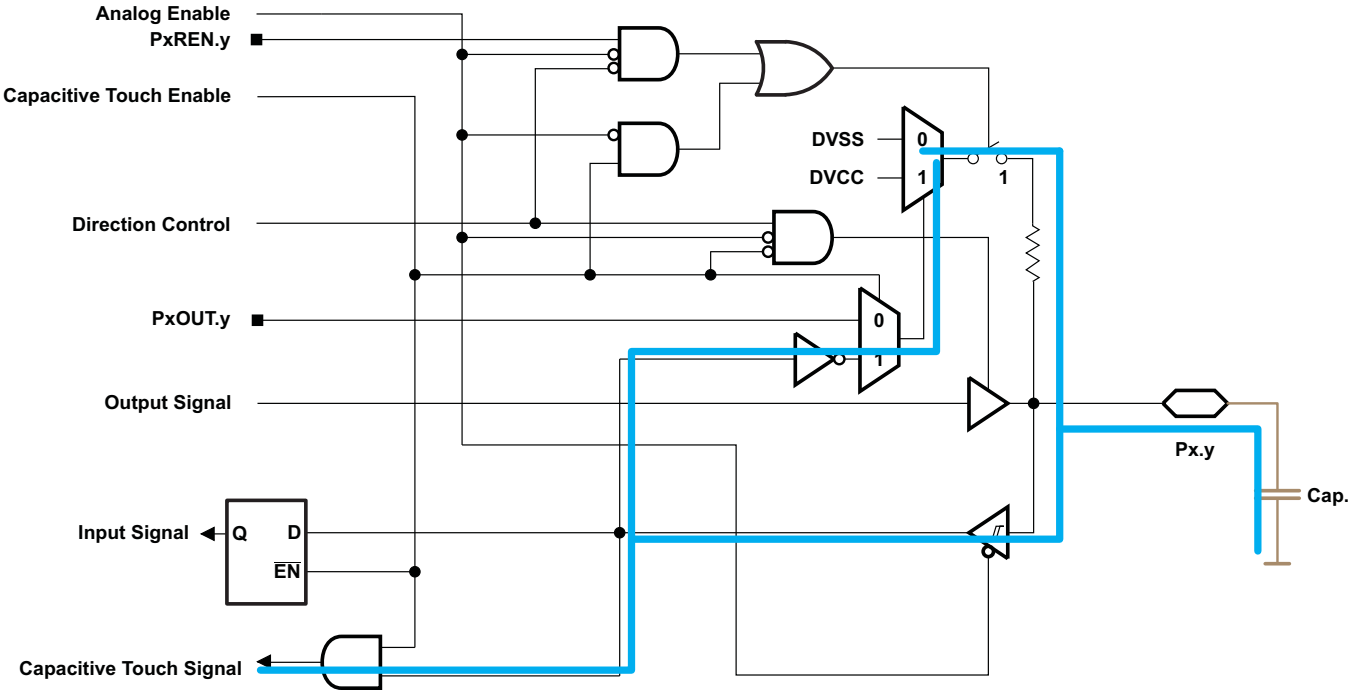


图 8-1. 电容式触摸 IO 原理

图 8-2 给出了电容式触摸 IO 模块的框图。

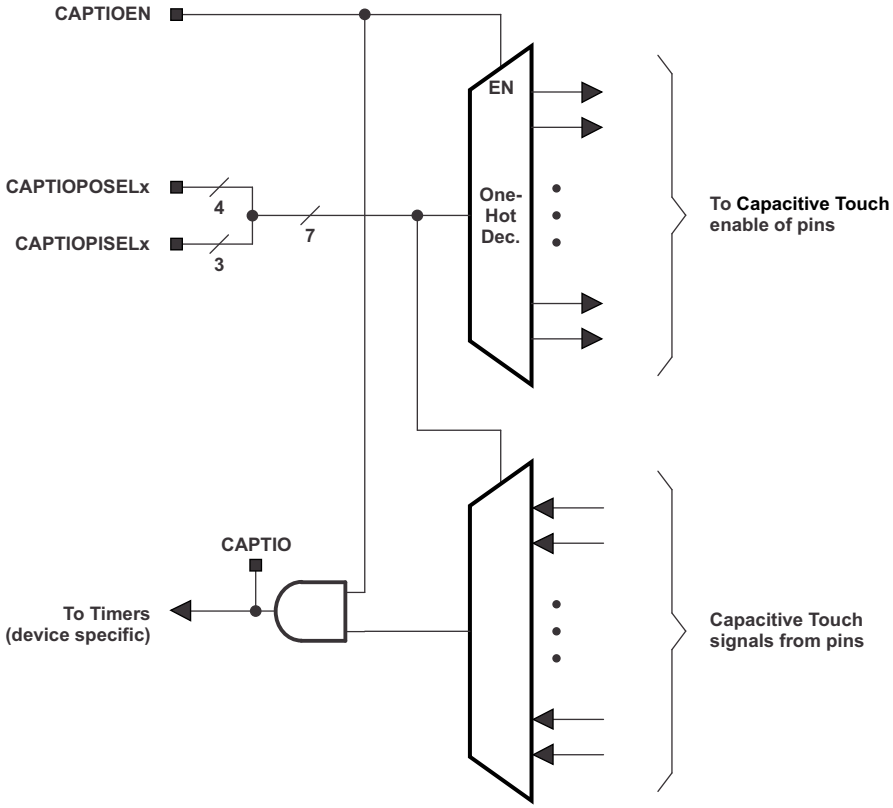


图 **8-2.** 电容式触摸 **IO** 框图

8.2 电容式触摸 IO 操作

通过 CAPTIOEN = 1 使能电容式触摸 IO 功能并使用 CAPTIOPOSELx 和 CAPTIOPISELx 选择端口引脚。选定的端口引脚会切换至电容式触摸状态，生成的振荡信号由定时器进行测量。连接的定时器特定于器件（请参见具体器件的数据表）。

可通过将电容式触摸 IO 控制寄存器 CAPTIOCTL_L 的低字节递增 2 扫描连续的端口引脚。

8.3 CapTouch 寄存器

表 8-1 列出了电容式触摸 IO 寄存器及其地址偏移量。给定的器件中可能提供多个电容式触摸 IO 寄存器。关于每个电容式触摸 IO 模块的基址，请参见具体器件的数据表。

注: 所有寄存器都支持字或字节访问。对于通用寄存器*ANYREG*来说，后缀“_L”(*ANYREG_L*) 是指寄存器的低字节（位 0 到 7）。后缀“_H”(*ANYREG_H*) 是指寄存器的高字节（位 8 到 15）。

表 8-1. CapTouch 寄存器

偏移量	首字母缩写词	寄存器名称	类型	访问	复位	部分
0Eh	CAPTIOxCTL	电容式触摸 IO x 控制寄存器	读取/写入	字	0000h	8.3.1 节
0Eh	CAPTIOxCTL_L		读取/写入	字节	00h	
0Fh	CAPTIOxCTL_H		读取/写入	字节	00h	

8.3.1 CAPTIOxCTL 寄存器（偏移 = 0Eh）[复位 = 0000h]

电容式触摸 IO x 控制寄存器

图 8-3. CAPTIOxCTL 寄存器

15	14	13	12	11	10	9	8
保留						CAPTIO	CAPTIOEN
r0	r0	r0	r0	r0	r0	r-0	rw-0
7	6	5	4	3	2	1	0
CAPTIOPOSELx				CAPTIOPISELx			保留
rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	r0

表 8-2. CAPTIOxCTL 寄存器说明

位	字段	类型	复位	说明
15-10	保留	R	0h	保留。始终读为 0。
9	CAPTIO	R	0h	电容式触摸 IO 状态。报告所选电容式触摸 IO 的当前状态。如果电容式触摸 IO 被禁用，则读为 0。 0b = 当前状态为 0 或电容式触摸 IO 已禁用 1b = 当前状态为 1
8	CAPTIOEN	RW	0h	电容式触摸 IO 使能 0b = 禁用所有电容式触摸 IO。信号方向定时器为 0。 1b = 使能所选电容式触摸 IO
7-4	CAPTIOPOSELx	RW	0h	电容式触摸 IO 端口选择。选择端口 Px。选择所用器件上不可用的端口引脚将产生无法预测的结果。 0000b = Px = PJ 0001b = Px = P1 0010b = Px = P2 0011b = Px = P3 0100b = Px = P4 0101b = Px = P5 0110b = Px = P6 0111b = Px = P7 1000b = Px = P8 1001b = Px = P9 1010b = Px = P10 1011b = Px = P11 1100b = Px = P12 1101b = Px = P13 1110b = Px = P14 1111b = Px = P15
3-1	CAPTIOPISELx	RW	0h	电容式触摸 IO 引脚选择。选择所选端口 Px 中的引脚（请参见 CAPTIOPOSELx）。选择所用器件上不可用的端口引脚将产生无法预测的结果。 000b = Px.0 001b = Px.1 010b = Px.2 011b = Px.3 100b = Px.4 101b = Px.5 110b = Px.6 111b = Px.7
0	保留	R	0h	保留。始终读为 0。

CRC 模块

循环冗余校验 (CRC) 模块为一个给定的数据序列提供了一个签名。本章介绍了 CRC 模块的操作和使用。

Topic **Page**

9.1 循环冗余校验 (CRC) 模块介绍

CRC 模块为一个给定的数据序列提供了一个信号。信号是通过数据位为 0，4，11，和 15 的反馈路径生成的（参见图 9-1）。CRC 信号基于 CRC-CCITT-BR 多项式中给出的多项式的（参见公式 11）。

$$f(x) = x^{16} + x^{12} + x^{5} + 1 \quad (11)$$

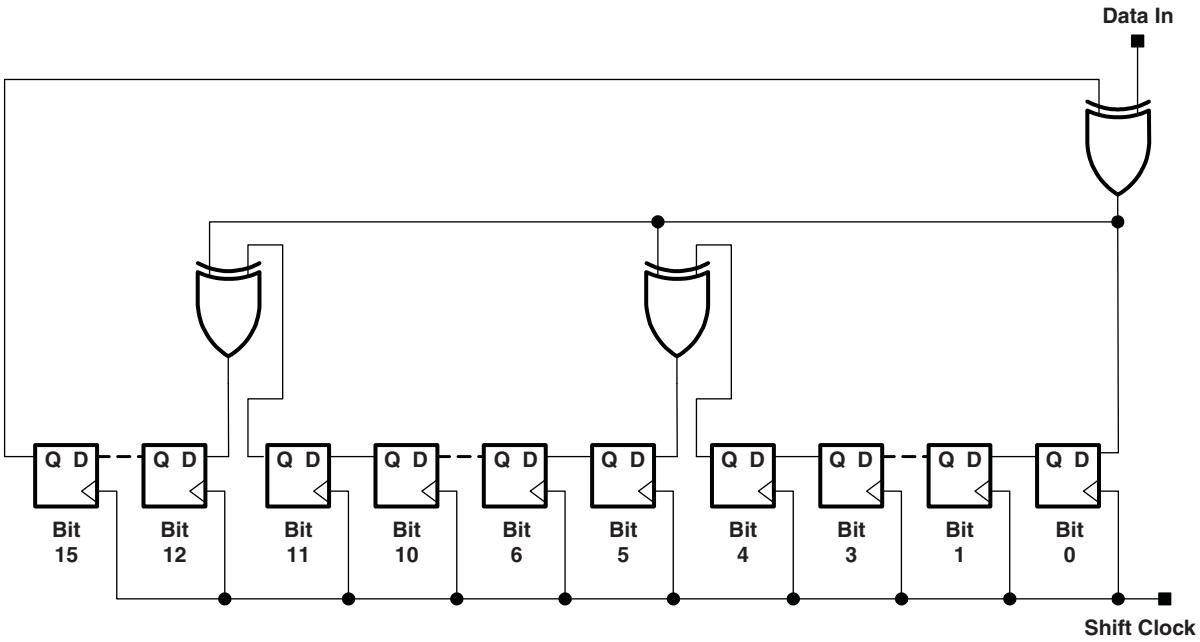


图 **9-1. LFSR** 的执行的标准是 **CRC-CCITT** 标准的，位 **0** 是结果的 **MSB**

当 CRC 被一个固定种子值初始化时，相同的输入数据序列会产生在相同的信号，而输入数据的序列不同时，在一般情况下，会生成不同的信号。

9.2 CRC 标准和位序

主框架计算机时代定义了不同的 CRC 标准，而按照惯例，BIT0 被看作 MSB。今天，在大多数微控制器中，如 MSP430，BIT 0 通常表示 LSB。在图 9-1中，给出了原来的位惯例标准，即，BIT0 就是 MSB。事实上，BIT0在某些模块上被用作LSB，在其他模块上被用作MSB，这一直引起了混乱。因此，CRC16 模块为 CRC16 的操作提供了一个位反转寄存器用来支持两种不同的序列。

9.3 CRC 的校验和生成

CRC 发生器的初始化，是由将一个 16 位字（种子）写入 CRC 初始化和结果寄存器 (CRCINIRES) 发起的。任何需要包括在 CRC 计算中的数据都须以与原来 CRC 信号被计算相同的顺序被写入 CRC 数据输入（CRCDI 或 CRCDIRB）到寄存器中。可以从CRCINIRES 寄存器中读取实际的信号，以便将计算出来的校验和与期望的校验和进行比较。

签名生成介绍了一种签名操作结果的计算方法。由一个外部工具所计算出的信号，以下文本中被称为校验和。校验和被存储在产品的存储器中，并用来检查 CRC 运算结果的正确性。

9.3.1 CRC 的执行

为了允许 CRC 的并行处理，线性反馈移位寄存器 (LFSR) 的功能是用 XOR 实现的。在8位数据被移入后，并且LSB 是第一个被“转移”的时候，此执行的实现与 LFSR 相同。通过把一个种子写入 CRCINIRES 寄存器来初始化该寄存器，从而存储一个信号计算的生成。可通过软件或硬件（例如，直接存储器访问 (DMA)）将数据（例如，存储器中的数据）传送至 CRCDI 或 CRCDIRB 寄存器。之后把 CRCDI 或 CRCDIRB 中的值列入信号中，在进行下一个读取访问（CRCINIRES 和 CRCRESR）时，其结果在信号结果寄存器中是可用的。字或字节数据都可用于生成信号。

如果一个字数据被处理时，在偶数地址上的低字节会在第一个时钟 (MCLK) 周期中被处理。在第二时钟周期期间，高字节会被处理。因此，它需要两个时钟周期来处理字数据，而它只需要一个时钟 (MCLK) 周期来处理字节数据。

使用字模式被写入 CRCDIRB 的数据字节，或者使用字节模式写入的数据字节，在 CRC 引擎它们添加到信号中前会很明智的进行反转。每个字节之间的位是相反的。使用字模式被写入 CRCDIRB 的数据字节，或者使用字节模式写入的数据字节，在被 CRC 引擎使用它们之前是不会进行位反转的。

如果校验和本身（采用反向的位顺序）包含于 CRC 操作中（与写入 CRCDI 或 CRCDIRB 的数据一样），CRCINIRES 和 CRCRESR 寄存器中的结果必须为零。

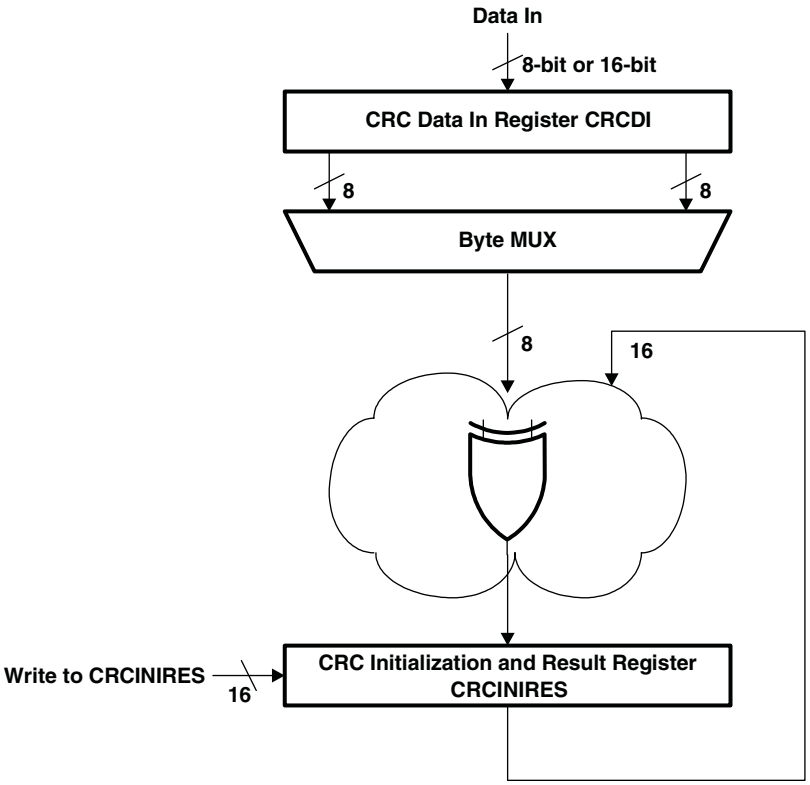


图 **9-2. CRC-CCITT** 的执行使用到**CRCDI** 和 **CRCINIRES** 寄存器

9.3.2 汇编程序示例

Example 9-1 演示了片上 CRC 的工作原理。

Example 9-1. 通用汇编程序示例

```
   ...
   PUSH   R4                ; Save registers
   PUSH   R5
   MOV    #StartAddress,R4  ; StartAddress < EndAddress
   MOV    #EndAddress,R5
   MOV    &INIT, &CRCINIRES ; INIT to CRCINIRES
L1 MOV    @R4+,&CRCDI       ; Item to Data In register
   CMP    R5,R4             ; End address reached?
   JLO    L1                ; No
   MOV    &Check_Sum,&CRCDI ; Yes, Include checksum
   TST    &CRCINIRES        ; Result = 0?
   JNZ    CRC_ERROR         ; No, CRCRES <> 0: error
   ...                      ; Yes, CRCRES=0:
                            ; information ok.
   POP    R5                ; Restore registers
   POP    R4
```

有关实现的 CRC 算法的详细信息，请参见Example 9-2 中的数据序列。该序列采用字或字节访问，并且使用了 CRC 数据输入寄存器和 CRC 数据输入反向字节寄存器。

Example 9-2. *参考数据序列*

```
   ...
mov     #0FFFFh,&CRCINIRES  ; initialize CRC
mov.b   #00031h,&CRCDI_L    ; "1"
mov.b   #00032h,&CRCDI_L    ; "2"
mov.b   #00033h,&CRCDI_L    ; "3"
mov.b   #00034h,&CRCDI_L    ; "4"
mov.b   #00035h,&CRCDI_L    ; "5"
mov.b   #00036h,&CRCDI_L    ; "6"
mov.b   #00037h,&CRCDI_L    ; "7"
mov.b   #00038h,&CRCDI_L    ; "8"
mov.b   #00039h,&CRCDI_L    ; "9"

cmp     #089F6h,&CRCINIRES  ; compare result
                            ; CRCRESR contains 06F91h
jeq     &Success            ; no error
br      &Error              ; to error handler
mov     #0FFFFh,&CRCINIRES  ; initialize CRC
mov.w   #03231h,&CRCDI      ; "1" & "2"
mov.w   #03433h,&CRCDI      ; "3" & "4"
mov.w   #03635h,&CRCDI      ; "5" & "6"
mov.w   #03837h,&CRCDI      ; "7" & "8"
mov.b   #039h,  &CRCDI_L    ; "9"

cmp     #089F6h,&CRCINIRES  ; compare result
                               ; CRCRESR contains 06F91h
jeq     &Success            ; no error
br      &Error              ; to error handler
   ...
mov     #0FFFFh,&CRCINIRES  ; initialize CRC
mov.b   #00031h,&CRCDIRB_L  ; "1"
mov.b   #00032h,&CRCDIRB_L  ; "2"
mov.b   #00033h,&CRCDIRB_L  ; "3"
mov.b   #00034h,&CRCDIRB_L  ; "4"
mov.b   #00035h,&CRCDIRB_L  ; "5"
mov.b   #00036h,&CRCDIRB_L  ; "6"
mov.b   #00037h,&CRCDIRB_L  ; "7"
mov.b   #00038h,&CRCDIRB_L  ; "8"
mov.b   #00039h,&CRCDIRB_L  ; "9"

cmp     #029B1h,&CRCINIRES  ; compare result
                            ; CRCRESR contains 08D94h
jeq     &Success            ; no error
br      &Error              ; to error handler
...
mov     #0FFFFh,&CRCINIRES  ; initialize CRC
mov.w   #03231h,&CRCDIRB    ; "1" & "2"
mov.w   #03433h,&CRCDIRB    ; "3" & "4"
mov.w   #03635h,&CRCDIRB    ; "5" & "6"
mov.w   #03837h,&CRCDIRB    ; "7" & "8"
mov.b   #039h,  &CRCDIRB_L  ; "9"

cmp     #029B1h,&CRCINIRES  ; compare result
                            ; CRCRESR contains 08D94h
jeq     &Success            ; no error
br      &Error              ; to error handler
```

9.4 CRC 寄存器

表 9-1中列出了 CRC 模块寄存器。有关寄存器的基址，请参见具体器件的数据表。表 9-1中给出了地址偏移。

注: 所有寄存器都支持字或字节访问。对于通用寄存器*ANYREG*来说，后缀“_L”(*ANYREG_L*) 是指寄存器的低字节（位 0 到 7）。后缀 "_H" (*ANYREG_H*) 是指寄存器的高字节（位 8 至 15）。

表 9-1. CRC 寄存器

偏移量	首字母缩写词	寄存器名称	类型	访问	复位	部分
00h	CRCDI	CRC 数据输入	读取/写入	字	0000h	9.4.1 节
00h	CRCDI_L		读取/写入	字节	00h	
01h	CRCDI_H		读取/写入	字节	00h	
02h	CRCDIRB	CRC 数据输入反向字节	读取/写入	字	0000h	9.4.2 节
02h	CRCDIRB_L		读取/写入	字节	00h	
03h	CRCDIRB_H		读取/写入	字节	00h	
04h	CRCINIRES	CRC 初始化和结果	读取/写入	字	FFFFh	9.4.3 节
04h	CRCINIRES_L		读取/写入	字节	FFh	
05h	CRCINIRES_H		读取/写入	字节	FFh	
06h	CRCRESR	CRC 结果反向	只读	字	FFFFh	9.4.4 节
06h	CRCRESR_L		读取/写入	字节	FFh	
07h	CRCRESR_H		读取/写入	字节	FFh	

9.4.1 CRCDI 寄存器

CRC 数据输入寄存器

图 **9-3. CRCDI** 寄存器

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

表 **9-2. CRCDI** 寄存器说明

位	字段	类型	复位	说明
15-0	CRCDI	RW	0h	CRC 数据输入。按照 CRC-CCITT 标准，被写入 CRCDI 寄存器的数据包括在 CRCINIRES 寄存器的现有签名中。

9.4.2 CRCDIRB 寄存器

CRC 数据输入反向寄存器

图 **9-4. CRCDIRB** 寄存器

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

表 **9-3. CRCDIRB** 寄存器说明

位	字段	类型	复位	说明
15-0	CRCDIRB	RW	0h	CRC 数据输入反向字节。按照 CRC-CCITT 标准，被写入 CRCDIRB 寄存器的数据包括在 CRCINIRES 和 CRCRESR 寄存器的现有签名中。对此寄存器的读取将返回寄存器 CRCDI 的内容。

9.4.3 CRCINIRES 寄存器

CRC 初始化和结果寄存器

图 9-5. CRCINIRES 寄存器

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

表 9-4. CRCINIRES 寄存器说明

位	字段	类型	复位	说明
15-0	CRCINIRES	RW	FFFFh	CRC 初始化和结果。这个寄存器保存了当前 CRC 结果（按照 CRC_CCITT 标准）。对这个寄存器的写入将用写入它的值初始化 CRC 计算。刚刚被写入的值可从 CRCINIRES 寄存器中读取。

9.4.4 CRCRESR 寄存器

CRC 反向结果寄存器

图 9-6. CRCRESR 寄存器

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

表 9-5. CRCRESR 寄存器说明

位	字段	类型	复位	说明
15-0	CRCRESR	R	FFFFh	CRC 反向结果。这个寄存器保存当前 CRC 结果（按照 CRC-CCITT 标准）。位顺序（例如，CRCINIRES[15] = CRCRESR[0]）与 CRCINIRES 寄存器中的位顺序反向（见示例代码）。

看门狗定时器 (WDT_A)

看门狗定时器是一个 32 位定时器，可以用来作为看门狗或作为一个间隔定时器。本章介绍了看门狗定时器。增强型看门狗定时器，WDT_A，在所有器件上都已执行。

Topic **Page**

10.1 WDT_A 介绍

看门狗定时器 (WDT_A) 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看门狗功能，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。

看门狗定时器模块的功能包括：

- 8 个软件可选时间间隔
- 看门狗模式
- 间隔模式
- 对看门狗定时器控制 (WDTCTL) 寄存器进行受密码保护的访问
- 可选时钟源
- 可以停止来节省电能
- 时钟故障安全功能

图 10-1 给出了看门狗定时器框图。

注： **看门狗定时器上电即激活。**

PUC 之后，WDT_A 模块会自动配置为看门狗模式，初始复位时间间隔约为 32ms（采用 SMCLK 时）。用户必须在初始复位时间间隔结束之前设置或暂停 WDT_A。

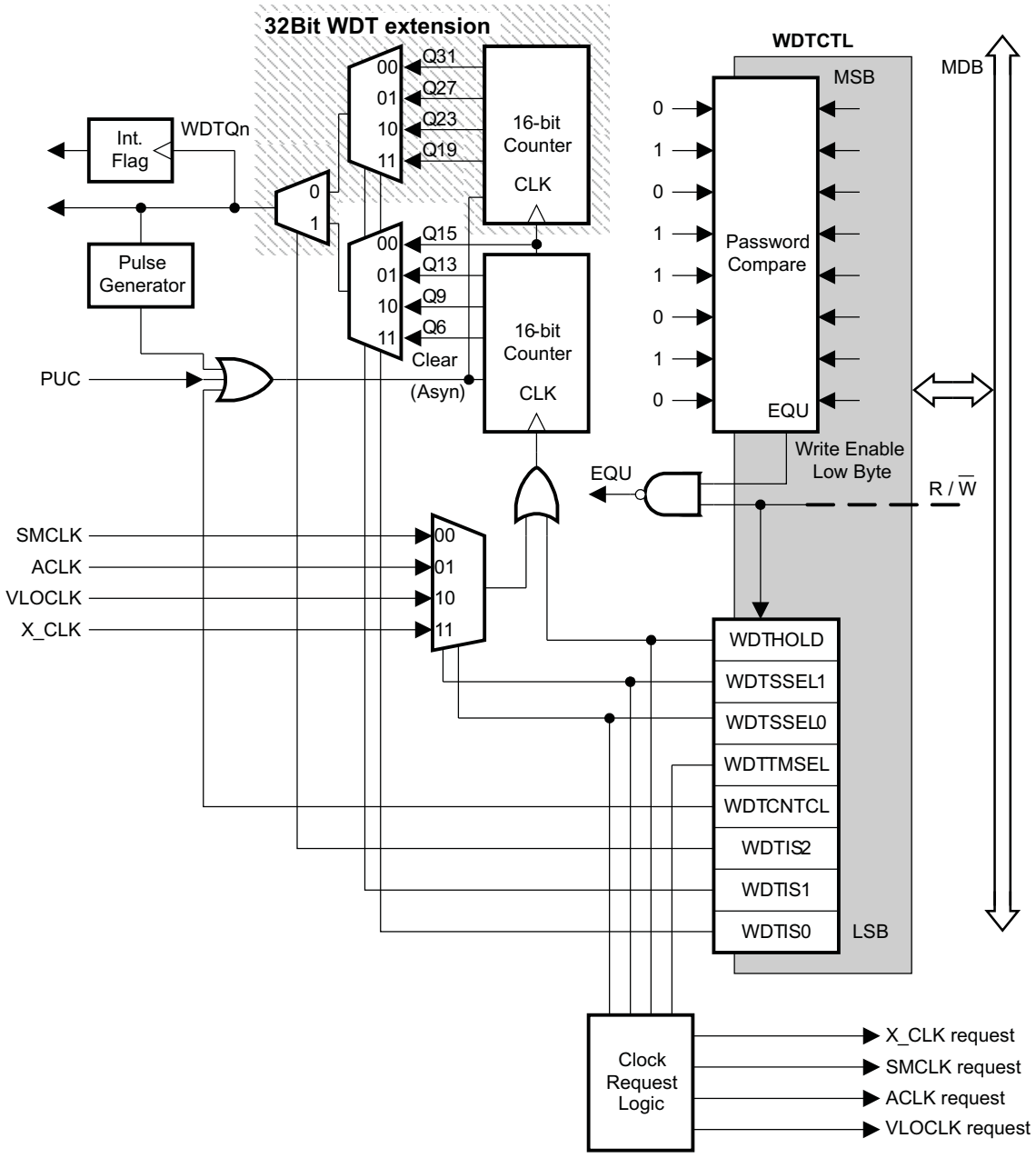


图 **10-1.** 看门狗定时器框图

10.2 WDT_A 的运行

看门狗定时器模块可以被配置为一个看门狗或带有 WDTCTL 寄存器的间隔定时器。WDTCTL是一个密码保护的 16 位读取/写入寄存器。任何读/写访问都必须采用字指令，写访问必须在高字节中包含写密码 05Ah。如果 WDTCTL 高字节写入的值不是 05Ah，则属于密码违规操作。无论当时为哪种定时器模式，该操作将都导致 PUC 系统复位。任何 WDTCTL 的读取会读取高字节中的 069h 。在 WDTCTL 的高字节或低字节部分读取的字节会生成一个低字节值。写入 WDTCTL 的高位宽字节或低位宽字节部分会导致一个 PUC。

10.2.1 看门狗计数器 (WDTCNT)

WDTCNT是一个软件不能直接访问的 32 位计数器。WDTCNT 是被看门狗定时控制寄存器 (WDTCTL) 控制并由该寄存器选择其时间间隔。某些器件上的 SMCLK、ACLK、VLOCLK 和 X_CLK 可为 WDTCNT 提供时钟。时钟源由 WDTSSEL 位来选择。时间间隔由 WDTIS 位来选择。

10.2.2 看门狗模式

PUC 之后，WDT_A 模块会配置为看门狗模式，初始复位时间间隔约为 32ms（采用 SMCLK 时）。在初始复位时间间隔结束或者另一个 PUC 产生之前，用户必须设置、暂停或清零看门狗定时器。当看门狗定时器配置为在看门狗模式下工作时，写入 WDTCTL 的密码不正确或者所选时间间隔结束都会触发 PUC。一个 PUC 看门狗定时器被复位到其默认状态。

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 1。在间隔定时器模式下，选定的定时器时间间隔结束时不会生成 PUC，并且 WDTIFG 使能位 WDTIE 保持不变。

当 WDTIE 位和 GIE 位被置 1 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 1。在间隔定时器模式下的中断向量地址与在看门狗模式下是不同的。

注: 修改看门狗定时器

为了避免意想不到的即时 PUC 或中断的发生，看门狗定时器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看门狗定时器。

10.2.4 看门狗定时器的中断

看门狗定时器为中断控制使用了 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。

当在看门狗模式下使用看门狗定时器时，WDTIFG 标志提供复位向量中断。复位中断服务程序可使用 WDTIFG 确定器件复位是否由看门狗引起。如果该标志置 1，则复位状态是由看门狗定时器超时或密码违规操作引起。如果 WDTIFG 清零，则复位是由其他源引起。

当在间隔定时器模式下使用看门狗定时器时，在选定的时间间隔后，WDTIFG 标志就会被置 1，并且，如果 WDTIE 和 GIE 位都被置 1，那么 WDTIFG 标志就会请求一个看门狗定时器间隔定时器中断。间隔定时器中断向量与看门狗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 时钟故障安全功能

WDT_A 模块提供了一个故障安全时钟功能，以此来确保在看门狗模式下 WDT_A 的时钟不能被禁用。这意味着低功耗模式可能会受 WDT_A 时钟选择的影响。

如果把 SMCLK 或 ACLK 作为 WDT_A 的时钟源，那么 VLOCLK 会被自动为 WDT_A 的时钟源。

当在间隔定时器模式下使用 WDT_A 模块时，在时钟源的 WDT_A 中就不存在故障安全功能。

10.2.6 在低功耗模式下的操作

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。应用程序的要求和使用的时钟类型决定了应如何配置 WDT_A。例如，如果用户希望使用低功耗模式 3，则不应将 WDT_A 配置为看门狗模式并由 DCO、XT1（高频模式）或 XT2（使用 SMCLK 或 ACLK）提供时钟源。在这种情况下，SMCLK 或 ACLK 将被保持使能，从而致使 LPM3 的电流消耗增加。当对看门狗定时器没有要求时，WDTHOLD 位可以用来保持 WDTCNT，从而降低了功耗。

对 WDTCTL 的任何写操作必须为字操作，并且高字节 (WDTPW) 为 05Ah（请参见Example 10-1）。

Example 10-1. 写入 WDTCTL

```
; Periodically clear an active watchdog
MOV #WDTPW+WDTIS2+WDTIS1+WDTCNTCL,&WDTCTL
;
; Change watchdog timer interval
MOV #WDTPW+WDTCNTCL+SSEL,&WDTCTL
;
; Stop the watchdog
MOV #WDTPW+WDTHOLD,&WDTCTL
;
; Change WDT to interval timer mode, clock/8192 interval
MOV #WDTPW+WDTCNTCL+WDTTMSEL+WDTIS2+WDTIS0,&WDTCTL
```

10.3 WDT_A 寄存器

表 10-1中列出了看门狗定时器模块。关于看门狗定时器模块寄存器和特殊功能寄存器 (SFR) 的基址，可参见具体器件的数据表。表 10-1中给出了地址偏移。

注: 所有寄存器都支持字或字节访问。对于通用寄存器*ANYREG*来说，后缀“_L”(*ANYREG_L*) 是指寄存器的低字节（位 0 到 7）。后缀“_H”(*ANYREG_H*) 是指寄存器的高字节（位 8 到 15）。

表 10-1. WDT_A 寄存器

偏移量	首字母缩写词	寄存器名称	类型	访问	复位	部分
00h	WDTCTL	看门狗定时器控制	读取/写入	字	6904h	10.3.1 节
00h	WDTCTL_L		读取/写入	字节	04h	
01h	WDTCTL_H		读取/写入	字节	69h	

10.3.1 WDTCTL 寄存器

看门狗定时器控制寄存器

图 10-2. WDTCTL 寄存器

15	14	13	12	11	10	9	8
WDTPW							
rw	rw	rw	rw	rw	rw	rw	rw
7	6	5	4	3	2	1	0
WDTHOLD	WDTSSEL		WDTTMSEL	WDTCNTCL	WDTIS		
rw - 0	rw - 0	rw - 0	rw - 0	r0(w)	rw - 1	rw - 0	rw - 0

表 10-2. WDTCTL 寄存器说明

位	字段	类型	复位	说明
15-8	WDTPW	RW	69h	看门狗定时器+密码。总是读取为 069h。必须写为 05Ah，否则就会产生一个 PUC。
7	WDTHOLD	RW	0h	看门狗定时器保持。该位停止看门狗定时器。为了省电，在不使用 WDT 时设置 WDTHOLD = 1。 0b = 不停止看门狗定时器 1b = 停止看门狗定时器
6-5	WDTSSEL	RW	0h	看门狗定时器时钟源选择 00b = SMCLK 01b = ACLK 10b = VLOCLK 11b = X_CLK
4	WDTTMSEL	RW	0h	看门狗定时器模式选择 0b = 看门狗模式 1b = 间隔定时器模式
3	WDTCNTCL	RW	0h	看门狗定时器计数器清零。设置 WDTCNTCL=1，清零计数值到 0000h。WDTCNTCL 被自动复位。 0b = 无动作 1b = WDTCNT = 0000h
2-0	WDTIS	RW	4h	看门狗定时器间隔选择。这些位选择用于将 WDTIFG 标志置 1 或生成 PUC 的看门狗定时器间隔。 000b = 看门狗时钟源 / 2^{31}（32.768kHz 时为 18:12:16） 001b = 看门狗时钟源 / 2^{27}（32.768kHz 时为 01:08:16） 010b = 看门狗时钟源 / 2^{23}（32.768kHz 时为 00:04:16） 011b = 看门狗时钟源 / 2^{19}（32.768kHz 时为 00:00:16） 100b = 看门狗时钟源 / 2^{15}（32.768kHz 时为 1s） 101b = 看门狗时钟源 / 2^{13}（32.768kHz 时为 250ms） 110b = 看门狗时钟源 / 2^{9}（32.768kHz 时为 15.625ms） 111b = 看门狗时钟源 / 2^{6}（32.768kHz 时为 1.95ms）

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。在一个给定的器件上可以有多种 定时器_A 模块（请参阅具体器件的数据表）。本章描述了 定时器_A 的使用和运行

Topic **Page**

11.1 定时器_A 介绍

定时器_A 是一个带有多达 7 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。计数器在溢出发生时可生成中断而每个捕捉/比较寄存器也可生成中断。

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
- 多达 7 个可配置捕捉/比较寄存器
- 带有脉冲宽度调制 (PWM) 功能的可配置输出
- 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 11-1所示。

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。

注: 命名规则

在一个给定的器件上有多种 定时器_A 的实例。使用了前缀 TAx，其中 x 大于或等于 0 表示 定时器_A 实例。 对于只有一个实例的器件，x = 0。后缀 n，其中 n = 0 到 6，代表与 定时器_A 相关的具体捕捉/比较寄存器。

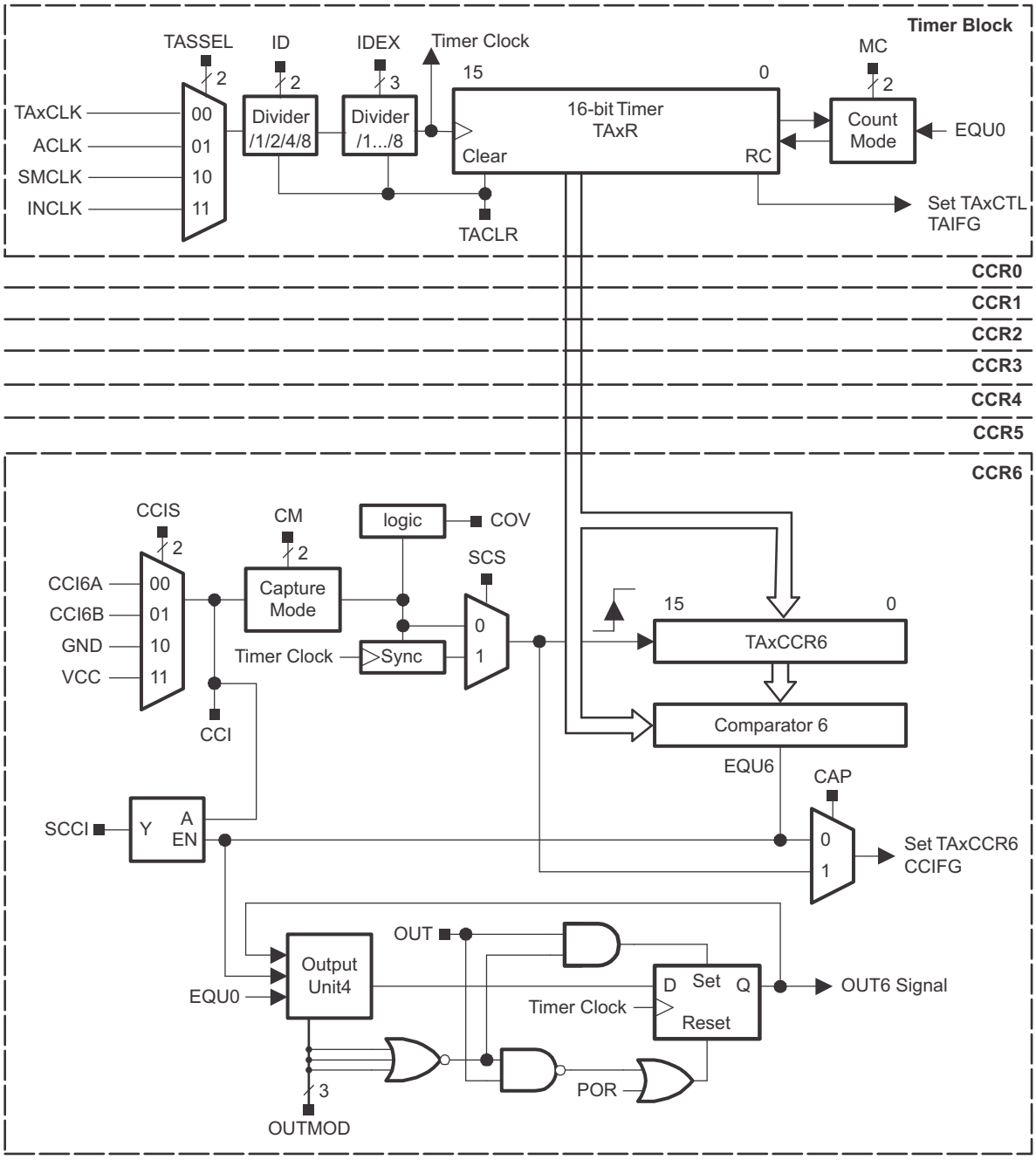


图 **11-1.** 定时器_A 的框图

11.2 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。

11.2.1 16 位定时器计数器

该 16 位定时器/计数器寄存器 () 在时钟信号的上升沿递增或递减（具体取决于工作模式）。TAxR 可以通过软件读取或写入。此外，定时器溢出时，它可以产生一个中断。

注: 访问 TAxR

访问 TAxR 时必须小心谨慎。如果 CPU 在定时器运行期间访问（读取或写入）TAxR，则读取/写入 TAxR 的值可能无法预测。为了避免这种不确定性，应在访问 TAxR 之前向 MC 位写入 0 以使定时器停止工作。对于读访问，可在定时器运行期间多次读取 TAxR，同时软件中会进行多数表决以确定正确读数。

11.2.1.1 时钟源选择和分频器

定时器时钟可由 ACLK 或 SMCLK 提供，或者由外部的 TAxCLK 或 INCLK 提供。时钟源由 TASSEL 位来选择。所选择的时钟源可以直接被传递给定时器或通过使用 ID 位进行 2，4 或 8 分频。所选时钟源可以使用 TAIDEX 位进一步 2，3，4，5，6，7，或 8 倍分频。当 TACLR 被置 1 时，定时器时钟分频器逻辑被复位。

注: 定时器_A 分频器

定时器时钟分频器通过 TACLR 位复位。时钟分频器实现为递减计数器。要复位递减计数器的状态，需在停止模式下向 TACLR 位写入 1。当定时器开始计数时，定时器时钟在 Timer_A 时钟源（通过 TASSEL 位选择）的第一个上升沿开始计时，然后以分频器设置（通过 ID 位和 TAIDEX 位设置）继续计时。

在定时器运行期间，不应更改时钟分频器（ID 位和 TAIDEX 位）。这可能引起意外行为。更改 ID 位或 TAIDEX 位时，应首先停止定时器 (MC = 0)。

11.2.2 启动定时器

当器件复位（BOR 或 POR）后，定时器处于停止状态，所有寄存器均为默认值。要从默认状态下启动定时器，请按照以下步骤操作。

1. 向 TACLR 位写入 1 (TACLR = 1) 来清零 TAxR、时钟分频器状态和计数器方向
2. 如有必要，向 TAxR 写入初始计数器值
3. 初始化 TAxCCRn
4. 对 TAxIV、TAIDEX 和 TAxCCTLn 应用所需配置
5. 对 TAxCTL（包括 MC 位）应用所需配置

11.2.3 定时器模式控制

定时器有四种运行模式：停止，增，连续，和增/减（参见表 11-1）。操作模式由 MC 位来选择。

表 11-1. 定时器模式

MC	模式	说明
00	停止	该定时器暂停。
01	向上	定时器从 0 开始到 TAxCCR0 的值重复计数。
10	连续	定时器从 0 开始到 0FFFFh 重复计数。
11	增/减	定时器从 0 开始递增加到 TAxCCR0 的值并返回到 0 重复计数。

要切换模式，首先向 MC 位写入 0 (MC = 0) 以停止定时器，然后根据所需模式设置 MC 位（有关详细信息，请参见表 11-1）。

11.2.3.1 递增模式

如果定时器周期必须与 0FFFFh 计数不同，则采用递增模式。定时器将反复递增计数至比较寄存器 TAxCCR0 的值，该值定义周期（请参见图 11-2）。在此期间的定时器计数的值是 TAxCCR0 + 1。当定时器的值等于 TAxCCR0时，定时器就回到 0 重新开始计数。如果选择递增模式，当定时器值大于 TAxCCR0 时，定时器会立即从 0 开始重新计数。

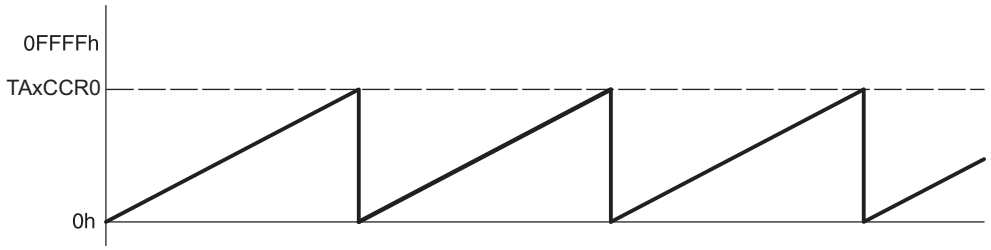


图 11-2. 递增模式

TAxCCR0 CCIFG 中断标志会在定时器*计数*至 TAxCCR0 值时置 1。当定时器从 *TAxCCR0 开始计数*到 0 时，TAIFG 中断标志被置 1。图 11-3说明了标志置 1 周期。

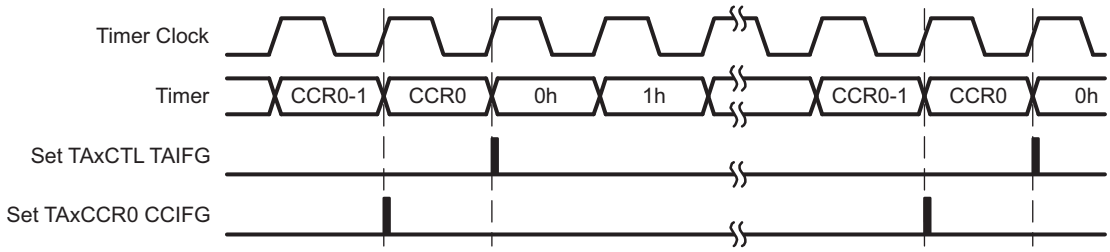


图 11-3. 增模式标志的置 1

11.2.3.1.1 更改周期寄存器 TAxCCR0

当 MC 位由停止模式 (MC = 0) 配置为递增模式 (MC = 1) 时，如果 TAxCCR0 大于 TAxR，定时器将从 TAxR 中的值开始递增计数。如果 TAxCCR0 小于或等于 TAxR，定时器会先返回 0，然后再递增计数至 TAxCCR0。定时器在返回 0 之前会额外进行一次计数。

在定时器运行期间更改 TAxCCR0 可能会引起意外行为。为了避免这种不确定性，应在停止模式 (MC = 0) 下更新 TAxCCR0。

11.2.3.2 连续模式

在连续模式下，定时器反复从 0 递增计数至 0FFFFh，如图 11-4 所示。捕捉/比较寄存器 TAxCCR0 与其他捕捉/比较寄存器的工作方式相同。

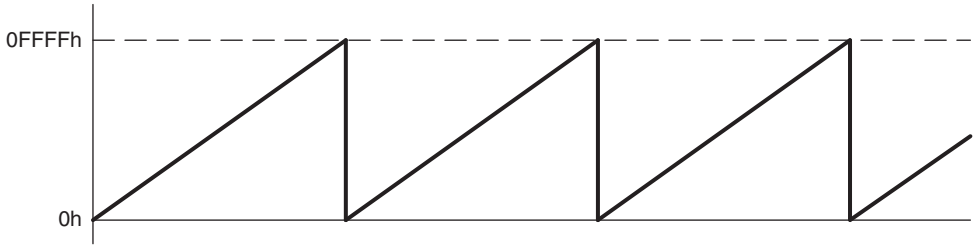


图 **11-4.** 连续模式

当定时器从 *0FFFFh 开始计数*到 0 时，TAIFG 中断标志被置 1。图 11-5显示了标志置 1 周期。

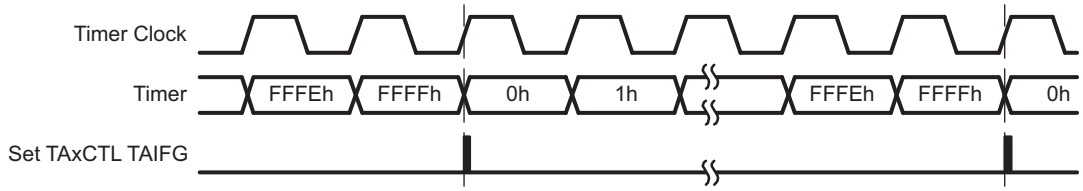


图 **11-5.** 连续模式标志的置 **1**

11.2.3.3 连续模式的使用

可以使用连续模式来生成单独的时间间隔和输出频率。每段时间间隔结束时，便会产生一个中断。下一个时间间隔被添加到中断服务子程序中的 TAxCCRn 寄存器。图 11-6显示了 2 个独立的时间间隔，t_0和 t_1正被添加至捕捉/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。高达 n（其中 n = 0 到 6），可以使用捕捉/比较寄存器来产生独立的时间间隔或输出频率。

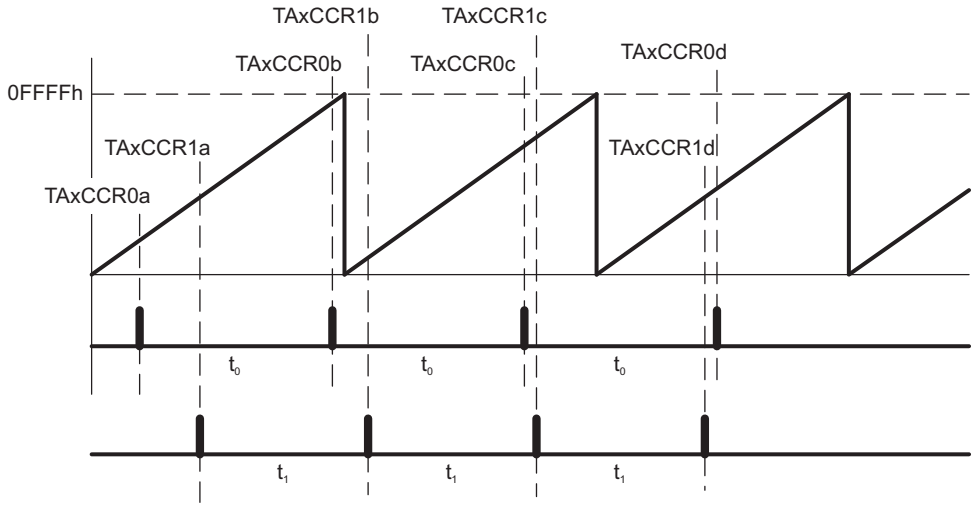


图 **11-6.** 连续模式时间间隔

其他模式也可以产生时间间隔，其中 TAxCCR0 用作周期寄存器。这些模式的处理方式更为复杂，因为先前的 TAxCCRn 数据与新周期之和可能大于 TAxCCR0 值。当先前的 TAxCCRn 值加上 t_x总和比 TAxCCR0 数据大时，为了获得正确的时间间隔，必须减去旧的 TDCL0 值。

11.2.3.4 递增/递减模式

如果定时器周期必须与 0FFFFh 计数不同并且需要生成对称脉冲，则采用递增/递减模式。定时器将反复递增计数至比较寄存器 TAxCCR0 的值并折回递减计数至 0（见图 11-7）。周期是 TAxCCR0 中值的两倍。

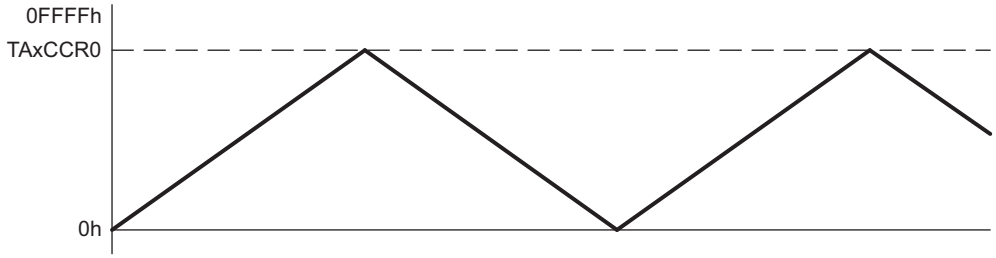


图 11-7. 递增/递减模式

计数方向会被锁存。这样可使停止的定时器以之前的计数方向重新启动。如果不希望如此，则必须在停止模式下将 TACLR 位置 1 以清除方向。TACLR 位还会清零 TAxR 值和定时器时钟分频器（分频器设置保持不变）。

在递增/递减模式下，TAxCCR0 CCIFG 中断标志和 TAIFG 中断标志在一个周期内均只置 1 一次，二者置 1 时刻相隔半个定时器周期。当定时器从 *TAxCCR0-1* 至 *TAxCCR0* *计数*时，TAxCCR0 CCIFG 中断标志被置 1；而定时器完成从 *0001h* *减至* *0000h* *的计数*时，TAIFG 被置 1。图 11-8说明了标志置 1 周期。

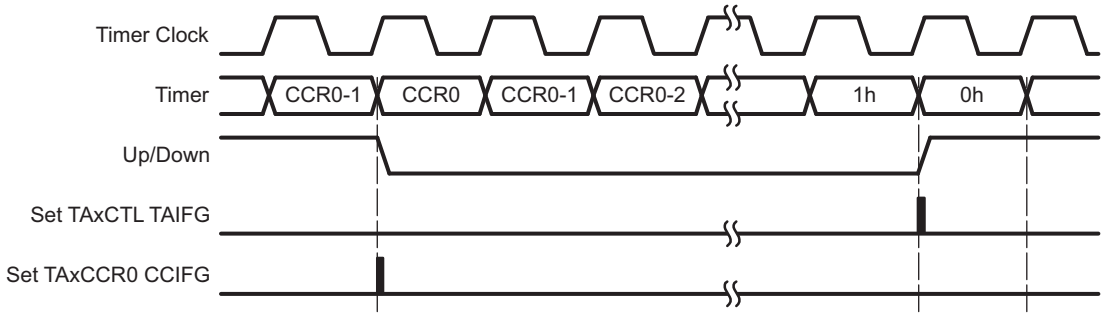


图 11-8. 增/模式标志的置 1

11.2.3.4.1 更改周期寄存器 TAxCCR0

当 MC 位由停止模式配置为递增/递减模式 (MC = 3) 时，定时器将根据之前的方向开始递增或递减计数。无论之前采用哪种模式，定时器都保持之前的计数方向。在停止模式下将 TACLR 位置 1 可将计数方向强制为递增；但是当定时器以递增方向开始计数时，无法将计数方向强制为递减。如果 TAxCCR0 大于 TAxR，则定时器将递增计数至 TAxCC0。如果 TAxCCR0 小于或等于 TAxR，则定时器开始递减计数。但是，定时器在开始递减计数之前会额外进行一次计数。

在递增/递减模式下，如果在定时器运行期间更新 TAxCCR0，则可能会引起意外行为。为了避免这种不确定性，应在停止模式 (MC = 0) 下更新 TAxCCR0。

11.2.3.5 递增/递减模式的使用

递增/递减模式支持输出信号间需要死区时间的应用（见11.2.5 节）。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 11-9所示的例子中，$t_{死区时间}$是：

$t_{dead} = t_{timer} \times (TAxCCR1 - TAxCCR2)$

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

TAxCCRn = 捕捉/比较寄存器 n 的内容

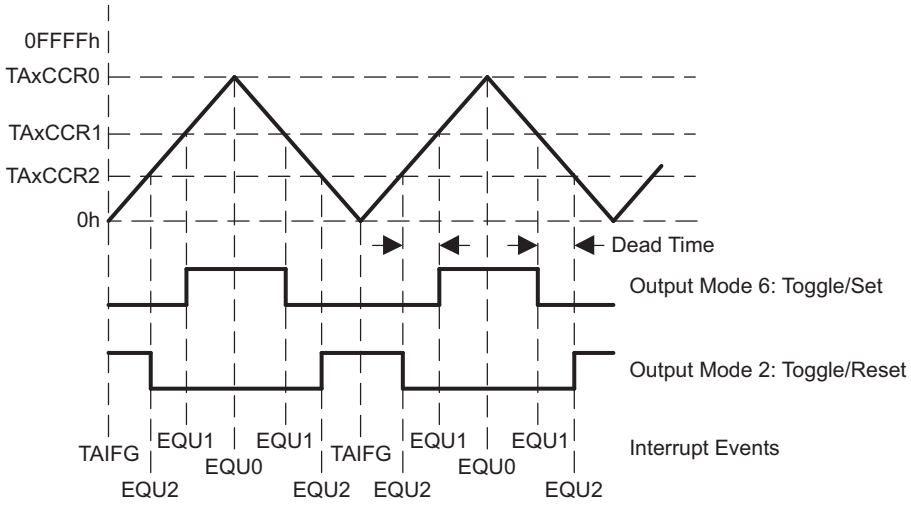


图 **11-9.** 增/减模式的输出单元

11.2.4 捕捉/比较块

定时器_A 中存在高达 7 个识别捕捉/比较模块，TAxCCRn（其中 n = 0 到 7）。这些块中的任一个都可用于捕捉定时器数据或生成时间间隔。

11.2.4.1 捕捉模式

当 CAP = 1 时，就会选择捕捉块。捕捉块用于记录时间事件。它可用于快速估计或时间测量。捕捉输入 CCIxA 和 CCIxB 被连接到一个外部引脚上或内部信号上，这通过 CCIS 位选择。CM 位选择捕捉输入信号的边沿如上升，下降，或两者兼而有之。一个捕捉发生在已选输入信号的边沿。如果发生捕捉：

- 定时器的值被复制仅 TAxCCRn 寄存器.
- 中断标志 CCIFG 被置 1。

可随时通过 CCI 位读取输入信号电平。器件可能会有被连接到 CCIxA 和 CCIxB 的不同信号。对于这些信号的连接，请参阅具体器件的数据表。

注: 在捕捉模式下读取 TAxCCRn

在捕捉模式下，如果 CPU 在发生捕捉事件（即，定时器计数器值复制到 TAxCCRn 中）时读取 TAxCCRn，则读取的值可能无效。为了避免出现这种意外情况，必须在 CCIFG 标志置 1 之后而下一个捕捉事件发生之前读取 TAxCCRn。

捕捉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。设置 SCS 位使下一个定时器时钟与捕捉信号同步。建议设置 SCS 位来使定时器时钟与捕捉信号同步。图 11-10

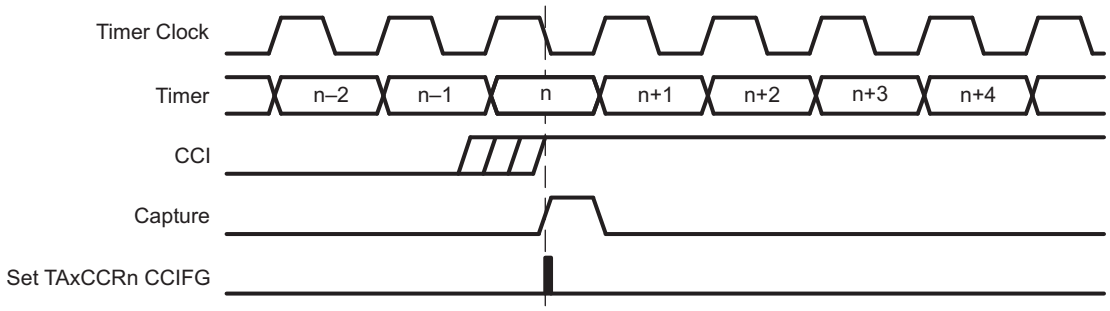


图 **11-10.** 捕捉信号 **(SCS = 1)**

注: 更改捕捉输入源（**CCIS** 位）

如果在捕捉模式下于 CCIxA 和 CCIxB 之间切换，可能会导致意外的捕捉事件。为了避免发生这种情况，应仅在禁用捕捉模式（CM = {0} 或 CAP = 0）时更改捕捉输入。请注意，可以随时在 GND 和 VCC 之间进行切换。有关详细信息，请参见节 11.2.4.1.1。

为了显示是否一个二次捕捉在第一次捕捉的值被读取之前发生，在每个捕捉/比较寄存器中就会提供一个溢出逻辑。如在图 11-11中所示，当这种情况发生时，位 COV 被置 1。COV 位必须由软件复位。

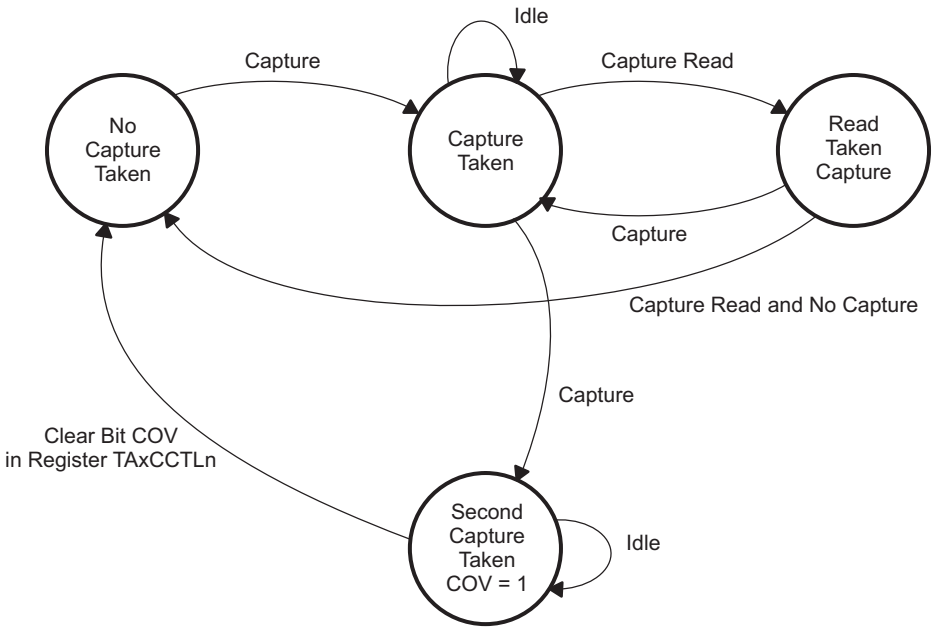


图 **11-11.** 捕捉循环

11.2.4.1.1 通过软件初始化捕捉

可通过软件初始化捕捉。CM 位可以配置捕捉的两个边沿。随后，软件会将 CCIS1 置 1 并翻转位 CCIS0 以在 V_{CC} 和 GND 之间切换捕捉信号，从而在每次 CCIS0 更改状态时发起一次捕捉：

```
MOV  #CAP+SCS+CCIS1+CM_3,&TA0CCTL1  ; Setup TA0CCTL1, synch. capture mode
                                    ; Event trigger on both edges of capture input.
XOR  #CCIS0,&TA0CCTL1               ; TA0CCR1 = TA0R
```

11.2.4.2 比较模式

当 CAP = 0 时，选择比较模式。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 TAxR *计数*到 TAxCCRn 中的值（其中，n 代表特定的捕捉/比较寄存器）时：

- 中断标志 CCIFG 被置 1.
- 内部信号 EQUn = 1。
- EQUn 根据输出模式影响输出.
- 输入信号 CCI 锁存到 SCCI.

注: 更新 TAxCCRn 寄存器

在捕捉模式下，向 TAxCCRn 写入新数据之前应向 MC 位写入 0 (MC = 0) 以使定时器停止工作。在定时器运行期间更新 TAxCCRn 可能会引起意外行为。

11.2.5 输出单元

每个捕捉/比较块包含一个输出单元。输出单元被用于生成输出信号，如 PWM 这样的信号。每个输出单元有 8 种可以根据 EQU0 和 EQUn 信号产生信号的操作模式。

11.2.5.1 输出模式

输出模式由 OUTMOD 位来确定，如表 11-2在中所述。对于除了模式 0 以外的所有模式来说，OUTn 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 EQUn = EQU0。

表 11-2. 输出模式

OUTMOD	模式	说明
000	输出	输出信号 OUTn 由 OUT 位定义。当 OUT 位更新时，OUTn 信号会立刻更新。
001	置 1	当定时器*计数*到 TAxCCRn 值时，输出被置 1。它保持置 1 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 TAxCCRn 值时，输出被切换。当定时器*计数*到 TAxCCR0 值时，它被复位。
011	置 1/复位	当定时器*计数*到 TAxCCRn 值时，输出被置 1。当定时器*计数*到 TAxCCR0 值时，它被复位。
100	切换	当定时器*计数*到 TAxCCRn 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 TAxCCRn 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 1	当定时器*计数*到 TAxCCRn 值时，输出被切换。当定时器*计数*到 TAxCCR0 值时，它被置 1。
111	复位/置 1	当定时器*计数*到 TAxCCRn 值时，输出被复位。当定时器*计数*到 TAxCCR0 值时，它被置 1。

11.2.5.1.1 输出举例—在单调增加模式中的定时器

当定时器递增计数至 TAxCCRn 值并从 TAxCCR0 返回 0 时，OUTn 信号会发生变化，具体变化取决于输出模式。图 11-12 给出了使用 TAxCCR0 和 TAxCCR1 时的示例。

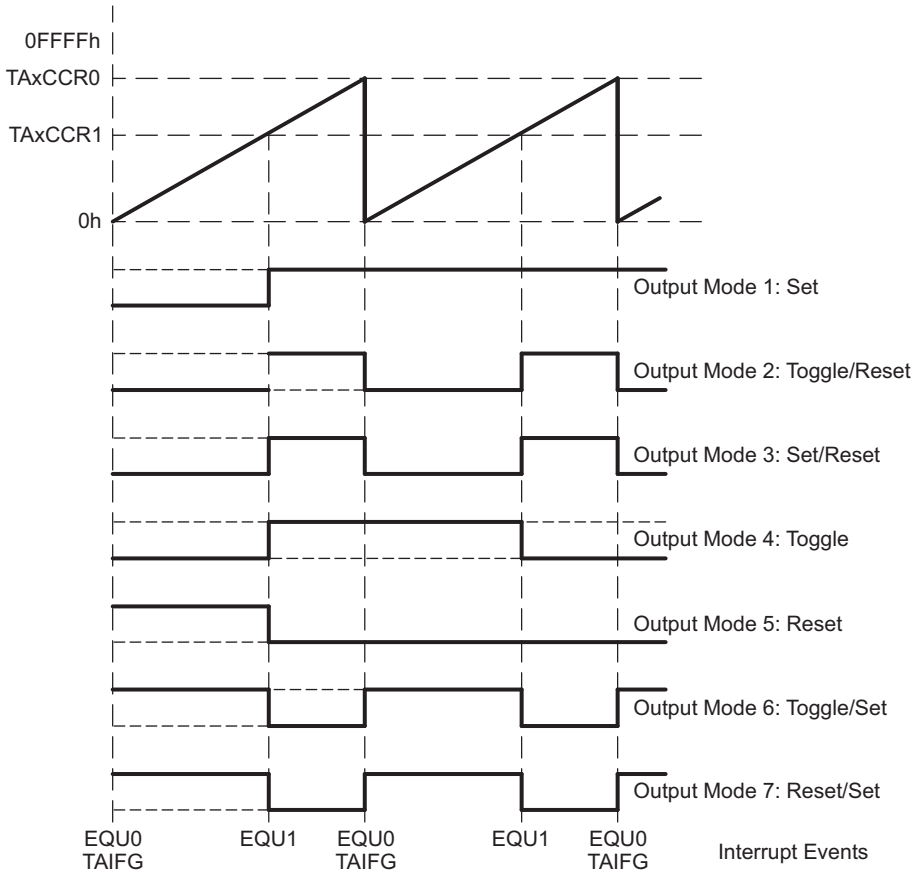


图 **11-12.** 输出示例 **-** 定时器处于增模式

11.2.5.1.2 输出示例 - 定时器处于连续模式

当定时器到达 TAxCCRn 和 TAxCCR0 值时，OUTn 信号会发生变化，具体变化取决于输出模式。图 11-13 给出了使用 TAxCCR0 和 TAxCCR1 时的示例。

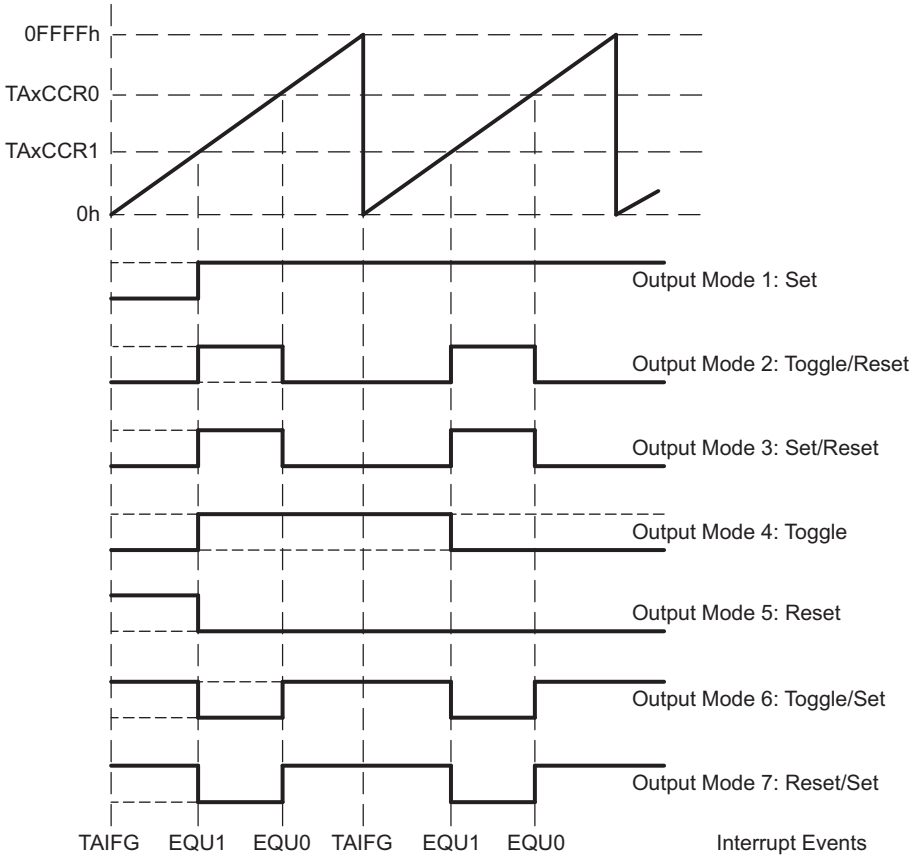


图 **11-13.** 输出示例 **-** 定时器处于连续模式

11.2.5.1.3 输出示例 - 定时器处于增/减模式

当定时器在任一计数方向的值等于 TAxCCRn 时以及定时器值等于 TAxCCR0 时，OUTn 信号会发生变化，具体变化取决于输出模式。图 11-14 给出了使用 TAxCCR0 和 TAxCCR2 时的示例。

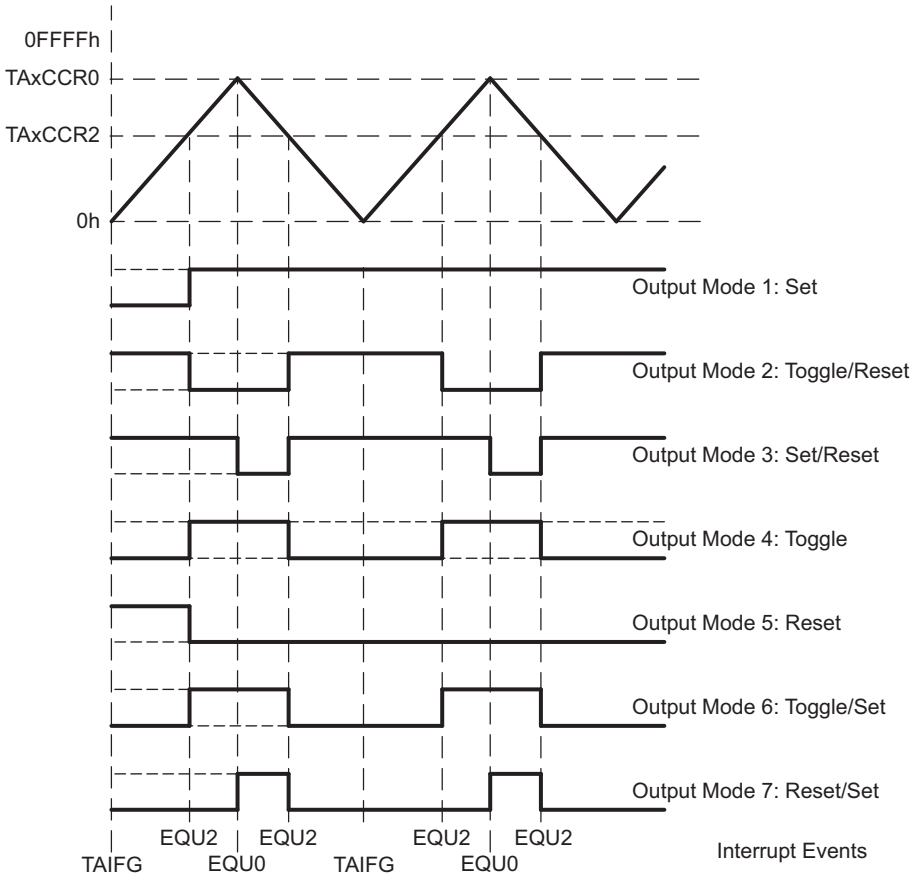


图 **11-14.** 输出示例 **-** 定时器处于增**/**减模式

注: 在输出模式间切换

德州仪器 (TI) 建议您在更改 OUTMOD 位之前停止定时器 (MC = 0)。但是，如果需要在定时器运行期间更改 OUTMOD 位，则其中一个 OUTMOD 位应在切换过程中保持置 1 状态（切换至模式 0 时除外）。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 作为一个过渡状态：

```
BIS    #OUTMOD_7,&TA0CCTL1      ; Set output mode=7
BIC    #OUTMOD,&TA0CCTL1        ; Clear unwanted bits
```

11.2.6 定时器_A 中断

16 位定时器_A 和 2 个中断向量相关联：

- TAxCCR0 中断向量，与 TAxCCR0 CCIFG 相关联
- TAxIV 中断向量，与所有其他 CCIFG 标志和 TAIFG 相关联

在捕捉模式下，当相关的 TAxCCRn 寄存器捕捉到定时器值时，任何 CCIFG 标志都将置 1。在比较模式下，如果 TAxR *计数*到相关的 TAxCCRn 值，则任何 CCIFG 标志都将置 1。软件也可以置 1 或清零任何 CCIFG 标志。当它们相应的 CCIE 位和 GIE 位被置 1 时，所有 CCIFG 标志就会要求产生一个中断。

11.2.6.1 TAxCCR0 中断

TAxCCR0 CCIFG 标志具有最高的 Timer_A 中断优先级和专用的中断向量，如图 11-15 所示。当 TAxCCR0 中断请求被处理后，TAxCCR0 CCIFG 标志会自动复位。

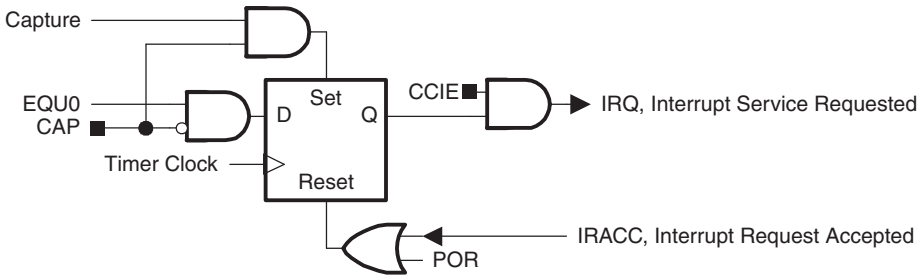


图 11-15. 捕捉/比较 TAxCCR0 中断标志

11.2.6.2 TAxIV，中断向量发生器

TAxCCRy CCIFG 标志和 TAIFG 标志被优先化并被结合发出一个中断向量。中断向量寄存器 TAxIV 用于确定请求中断的标志。

优先级最高的已使能中断会在 TAxIV 寄存器中生成一个数字（请参见寄存器说明）。可评估该数字或将其添加到程序计数器，以便自动进入相应的软件程序。禁止定时器_A 中断不会影响 TAxIV 的值。

对 TAxIV 寄存器执行任何访问（读或写）都会自动将最高挂起中断标志复位。如果有另一个中断标志置 1，则在处理完最初的中断后会立即产生另一个中断。例如，当中断服务程序访问 TAxIV 寄存器时，如果 TAxCCR1 和 TAxCCR2 CCIFG 标志均置 1，则 TAxCCR1 CCIFG 会自动复位。在中断服务子程序的 RETI 命令执行后，TAxCCR2 CCIFG 标志会产生另一个中断。

11.2.6.2.1 TAxIV 软件示例

以下软件示例给出了 TAxIV 的建议用法以及处理开销。TAxIV 的值被加入 PC 以便自动跳转到相应的子程序。本例假定了最大的定时器的单一实例配置可用。

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：

- 捕捉/比较块 TA0CCR0： 11 个周期
- 捕捉/比较块 TA0CCR1，TA0CCR2，TA0CCR3，TA0CCR4，TA0CCR5，TA0CCR6：16 周期
- 定时器溢出 TA0IFG：14 周期

```
; Interrupt handler for TA0CCR0 CCIFG.                    Cycles
CCIFG_0_HND
;        ...          ; Start of handler Interrupt latency    6
         RETI                                                 5

; Interrupt handler for TA0IFG, TA0CCR1 through TA0CCR6 CCIFG.

TA0_HND      ...                ; Interrupt latency           6
         ADD      &TA0IV,PC     ; Add offset to Jump table    3
         RETI                   ; Vector  0: No interrupt     5
         JMP      CCIFG_1_HND   ; Vector  2: TA0CCR1          2
         JMP      CCIFG_2_HND   ; Vector  4: TA0CCR2          2
         JMP      CCIFG_3_HND   ; Vector  6: TA0CCR3          2
         JMP      CCIFG_4_HND   ; Vector  8: TA0CCR4          2
         JMP      CCIFG_5_HND   ; Vector 10: TA0CCR5          2
         JMP      CCIFG_6_HND   ; Vector 12: TA0CCR6          2

TA0IFG_HND                     ; Vector 14: TA0IFG Flag
         ...                   ; Task starts here
         RETI                                                 5

CCIFG_6_HND                    ; Vector 12: TA0CCR6
         ...                   ; Task starts here
         RETI                  ; Back to main program        5

CCIFG_5_HND                    ; Vector 10: TA0CCR5
         ...                   ; Task starts here
         RETI                  ; Back to main program        5

CCIFG_4_HND                    ; Vector 8: TA0CCR4
         ...                   ; Task starts here
         RETI                  ; Back to main program        5

CCIFG_3_HND                    ; Vector 6: TA0CCR3
         ...                   ; Task starts here
         RETI                  ; Back to main program        5

CCIFG_2_HND                    ; Vector 4: TA0CCR2
         ...                   ; Task starts here
         RETI                  ; Back to main program        5

CCIFG_1_HND                    ; Vector 2: TA0CCR1
         ...                   ; Task starts here
         RETI                  ; Back to main program        5
```

注: 更改定时器时钟源

德州仪器 (TI) 建议您不要在定时器运行期间修改其工作方式，而应该首先使其停止工作。

如果定时器时钟源与 MCLK 异步，则在重新启动定时器之前至少需要 1.5 个定时器时钟的延迟才能重新完成同步，因为定时器状态机需要在这段时间内按照新的配置来同步时钟源。（如果定时器采用 1MHz 时钟，则建议延迟 1.5µs 后重启定时器。）

11.2.7 更新 Timer_A 配置

对 TAxCTL、TAxCTLn 或 TAxEX0 应用新配置时务必小心谨慎。下列控制位被设计为不应在定时器运行期间动态更新，在定时器运行期间更改下列控制位可能引起意外行为。请注意，下面未列出的控制位可在定时器运行期间进行读取或更新。

- TAxCTL 寄存器
  - 时钟源选择 (TASSEL)
  - 输入分频器 (ID)
  - 模式控制 (MD)（注：可以随时切换至停止模式）
  - Timer_A 清零 (TACLR)
- TAxCCTLn 寄存器
  - 捕捉控制 (CM)（注：可以随时切换至非捕捉模式）
  - 捕捉/比较输入选择 (CCIS)（注：可以随时在 GND 与 VCC 之间切换）
  - 同步捕捉源 (SCS)
  - 捕捉模式 (CAP)
  - 输出模式 (OUTMOD)
- TAxEX0 寄存器
  - 输入分频器扩展 (TAIDEX)

要更新 Timer_A 配置，请按照以下步骤操作：

1. 向模式控制位写入 0 (MC = 0)（注： 请勿使用 TACLR 位来复位模式控制位）。
2. 如有必要，向 TACLR 位写入 1 (TACLR = 1) 来清零 TAxR、时钟分频器状态和计数器方向。
3. 如有必要，将计数器值更新到 TAxR。
4. 如需在定时器处于捕捉模式时更新 CM、CCIS、SCS 位或 TAxCCRn，应首先向 CAP 位或 CM 位写入 0（CAP = 0 或 CM = 0）来禁用捕捉模式。
5. 对 TAxCCRn、TAIDEX 和 TAxCCTLn 应用所需配置。
6. 对 TAxCTL（包括 MC 位）应用所需配置。

11.3 Timer_A 寄存器

在表 11-3中列出了最大配置的 Timer_A 寄存器。有关寄存器的基址，请参见具体器件的数据表。

表 11-3. Timer_A 寄存器

偏移量	首字母缩写词	寄存器名称	类型	访问	复位	部分
00h	TAxCTL	Timer_Ax 控制	读取/写入	字	0000h	11.3.1 节
02h	TAxCCTL0	Timer_Ax 的捕捉/比较控制 0	读取/写入	字	0000h	11.3.3 节
04h	TAxCCTL1	Timer_Ax 的捕捉/比较控制 1	读取/写入	字	0000h	11.3.3 节
06h	TAxCCTL2	Timer_Ax 的捕捉/比较控制 2	读取/写入	字	0000h	11.3.3 节
08h	TAxCCTL3	Timer_Ax 的捕捉/比较控制 3	读取/写入	字	0000h	11.3.3 节
0Ah	TAxCCTL4	Timer_Ax 的捕捉/比较控制 4	读取/写入	字	0000h	11.3.3 节
0Ch	TAxCCTL5	Timer_Ax 的捕捉/比较控制 5	读取/写入	字	0000h	11.3.3 节
0Eh	TAxCCTL6	Timer_Ax 的捕捉/比较控制 6	读取/写入	字	0000h	11.3.3 节
10h	TAxR	Timer_Ax 计数器	读取/写入	字	0000h	11.3.2 节
12h	TAxCCR0	Timer_Ax 的捕捉/比较 0	读取/写入	字	0000h	11.3.4 节
14h	TAxCCR1	Timer_Ax 的捕捉/比较 1	读取/写入	字	0000h	11.3.4 节
16h	TAxCCR2	Timer_Ax 的捕捉/比较 2	读取/写入	字	0000h	11.3.4 节
18h	TAxCCR3	Timer_Ax 的捕捉/比较 3	读取/写入	字	0000h	11.3.4 节
1Ah	TAxCCR4	Timer_Ax 的捕捉/比较 4	读取/写入	字	0000h	11.3.4 节
1Ch	TAxCCR5	Timer_Ax 的捕捉/比较 5	读取/写入	字	0000h	11.3.4 节
1Eh	TAxCCR6	Timer_Ax 的捕捉/比较 6	读取/写入	字	0000h	11.3.4 节
2Eh	TAxIV	Timer_Ax 的中断向量	只读	字	0000h	11.3.5 节
20h	TAxEX0	Timer_Ax 的扩展 0	读取/写入	字	0000h	11.3.6 节

11.3.1 TAxCTL 寄存器

Timer_Ax 的控制寄存器

图 11-16. TAxCTL 寄存器

15	14	13	12	11	10	9	8
保留						TASSEL	
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ID		MC		保留	TACLR	TAIE	TAIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	w-(0)	rw-(0)	rw-(0)

表 11-4. TAxCTL 寄存器说明

位	字段	类型	复位	说明
15-10	保留	RW	0h	保留
9-8	TASSEL	RW	0h	Timer_A 时钟源选择 00b = TAxCLK 01b = ACLK 10b = SMCLK 11b = INCLK
7-6	ID	RW	0h	输入分频。这些位与 TAIDEX 位一起选择输入时钟的分频器。 00b = /1 01b = /2 10b = /4 11b = /8
5-4	MC	RW	0h	模式控制。不使用 Timer_A 时，将 MC 设置为 0 可实现节能。 00b = 停止模式：定时器暂停 01b = 递增模式：定时器递增计数至 TAxCCR0 10b = 连续模式：定时器递增计数至 0FFFFh 11b = 递增/递减模式：定时器递增计数至 TAxCCR0 后再折回递减计数至 0000h
3	保留	RW	0h	保留
2	TACLR	RW	0h	Timer_A 清零。将该位置 1 会复位 TAxR、定时器时钟分频器逻辑（分频器设置保持不变）以及计数方向。TACLR 位自动复位并始终读为 0。
1	TAIE	RW	0h	Timer_A 中断使能。这些位启用 TAIFG 中断请求。 0b = 中断被禁用 1b = 中断被启用
0	TAIFG	RW	0h	Timer_A 中断标志 0b = 无中断挂起 1b = 中断挂起

11.3.2 TAxR 寄存器

Timer_Ax 的计数器寄存器

图 **11-17. TAxR** 寄存器

15	14	13	12	11	10	9	8
TAxR							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
TAxR							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

表 **11-5. TAxR** 寄存器说明

位	字段	类型	复位	说明
15-0	TAxR	RW	0h	Timer_A 寄存器。TAxR 寄存器是 Timer_A 的计数。

11.3.3 TAxCCTLn 寄存器

Timer_A 的捕捉/比较控制 n 寄存器

图 11-18. TAxCCTLn 寄存器

15	14	13	12	11	10	9	8
CM		CCIS		SCS	SCCI	保留	CAP
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r-(0)	r-(0)	rw-(0)
7	6	5	4	3	2	1	0
OUTMOD			CCIE	CCI	OUT	COV	CCIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	rw-(0)	rw-(0)	rw-(0)

表 11-6. TAxCCTLn 寄存器说明

位	字段	类型	复位	说明
15-14	CM	RW	0h	捕捉模式 00b = 无捕捉 01b = 在上升沿的捕捉 10b = 在下降沿的捕捉 11b = 在上升和下降沿都捕捉
13-12	CCIS	RW	0h	捕捉/比较输入选择。这些位选择 TAxCCR0 输入信号。有关特定信号连接的信息，请参见具体器件的数据表。 00b = CCIxA 01b = CCIxB 10b = GND 11b = VCC
11	SCS	RW	0h	同步捕捉源。此位是用来同步定时器时钟与捕捉输入信号的。 0b = 异步捕捉 1b = 同步捕捉
10	SCCI	RW	0h	同步捕捉/比较输入选定的 CCI 输入信号与 EQUx 信号被锁存，并且可通过该位进行读取。
9	保留	R	0h	保留。读为 0。
8 个	CAP	RW	0h	捕捉模式 0b = 比较模式 1b = 捕捉模式
7-5	OUTMOD	RW	0h	输出模式。由于 EQUx = EQU0，模式 2、3、6 和 7 不能用于 TAxCCR0。 000b = OUT 位的值 001b = 置 1 010b = 触发/复位 011b = 置 1/复位 100b = 触发 101b = 复位 110b = 触发/置 1 111b = 复位/置 1
4	CCIE	RW	0h	捕捉/比较中断使能。该位使能相应 CCIFG 标志的中断请求。 0b = 中断被禁用 1b = 中断被启用
3	CCI	R	0h	捕捉/比较输入。所选输入信号可以由该位读出。
2	OUT	RW	0h	输出。当 OUTMOD = 0 时，该位直接控制输出的状态。 0b = 输出低电平 1b = 输出高电平
1	COV	RW	0h	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。 0b = 没有捕捉溢出发生 1b = 捕捉溢出发生

表 11-6. TAxCCTLn 寄存器说明 (continued)

位	字段	类型	复位	说明
0	CCIFG	RW	0h	捕捉/比较中断标志 0b = 无中断挂起 1b = 中断挂起

11.3.4 TAxCCRn 寄存器

Timer_A 捕捉/比较 n 寄存器

图 11-19. TAxCCRn 寄存器

15	14	13	12	11	10	9	8
TAxCCRn							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
TAxCCRn							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

表 11-7. TAxCCRn 寄存器说明

位	字段	类型	复位	说明
15-0	TAxCCRn	RW	0h	比较模式：TAxCCRn 保存用于与 Timer_A 寄存器中的定时器值进行比较的数据，TAR。 捕捉模式： Timer_A 的寄存器，TAR，当一个捕捉被执行时，TBR 就被复制到 TAxCCRn 寄存器中。

11.3.5 TAxIV 寄存器

Timer_Ax 的中断向量寄存器

图 11-20. TAxIV 寄存器

15	14	13	12	11	10	9	8
TAIV							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
TAIV							
r0	r0	r0	r0	r-(0)	r-(0)	r-(0)	r0

表 11-8. TAxIV 寄存器说明

位	字段	类型	复位	说明
15-0	TAIV	R	0h	Timer_A 中断矢量值 00h = 无中断挂起 02h = 中断源：捕捉/比较 1；中断标志：TAxCCR1 CCIFG；中断优先级：最高 04h = 中断源：捕捉/比较 2；中断标志：TAxCCR2 CCIFG 06h = 中断源：捕捉/比较 3；中断标志：TAxCCR3 CCIFG 08h = 中断源：捕捉/比较 4；中断标志：TAxCCR4 CCIFG 0Ah = 中断源：捕捉/比较 5；中断标志：TAxCCR5 CCIFG 0Ch = 中断源：捕捉/比较 6；中断标志：TAxCCR6 CCIFG 0Eh = 中断源：定时器溢出；中断标志：TAxCTL TAIFG；中断优先级：最低

11.3.6 TAxEX0 寄存器

Timer_Ax 的扩展 0 寄存器

图 11-21. TAxEX0 寄存器

15	14	13	12	11	10	9	8
保留							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
保留					TAIDEX(1)		
r0	r0	r0	r0	r0	rw-(0)	rw-(0)	rw-(0)

(1) 在编程 TAIDEX 位和配置定时器后，置 1 TACLR 位，来确保定时分频器逻辑适当的复位。

表 11-9. TAxEX0 寄存器说明

位	字段	类型	复位	说明
15-3	保留	R	0h	保留。读为 0。
2-0	TAIDEX	RW	0h	输入分频器扩展。这些位与 ID 位一起选择输入时钟的分频器。 000b = 不分频 001b = 2 分频 010b = 3 分频 011b = 4 分频 100b = 5 分频 101b = 6 分频 110b = 7 分频 111b = 8 分频

实时时钟 (RTC) 计数器

实时时钟 (RTC) 计数器为 16 位计数器，工作于激活模式 (AM) 和多种低功耗模式 (LPM) 下。RTC 计数器接受多个由控制寄存器设置进行选择的时钟源，以生成从小于 1µs 至长达数小时的时序。本章介绍了 RTC 计数器模块的操作和使用。

Topic **Page**

12.1 RTC 计数器简介

RTC 计数器是一个 16 位计数器，可在 AM 模式和除 LPM4 和 LPM4.5 之外所有 LPM 模式下工作。该模块的时钟源可以是下述三个之一：

1. 器件特定时钟源：SMCLK（最大工作频率取决于器件配置）或 ACLK（约 32kHz）
2. XT1CLK（约 32kHz）
3. VLOCLK（约 10kHz）

在 LPM3.5 模式下，RTC 计数器仅接受 XT1CLK 或 VLOCLK 作为其时钟源来定期唤醒器件。在对 16 位主计数器分频前，可以预分频所选时钟源。该 16 位计数器通过用户可访问的 16 位模寄存器和用户不可访问的 16 位影子寄存器支持连续计数。若 RTC 计数器在计数器值达到预设的影子寄存器值时溢出，则会生成中断。RTC 计数器特性包括：

- 16 位模计数器架构
  - 16 位基本计数器
  - 用户可进行读写访问的 16 位模寄存器
  - 用户不可访问的 16 位影子寄存器，用于在更新模值时支持继续运行
  - 16 位比较逻辑，用于检测影子寄存器值边界的计数器溢出
- 可通过设置 RTCSS 位来选择三个可用时钟源：XT1CLK、VLOCLK 或器件特定时钟源（SMCLK 或 ACLK）。
  - SMCLK 仅可在 AM 和 LPM0 模式下工作
  - ACLK 可在 AM 至 LPM3 模式下工作
  - XT1CLK 和 VLOCLK 可在 AM 以及除 LPM4 和 LPM4.5 外的全部 LPM 模式下工作
- 源时钟输入的可配置预分频器由 RTCPS 位设置
  - 直通：÷1；输入时钟源直接驱动 16 位计数器
  - 预分频器：÷10、÷100、÷1000、÷16、÷64、÷256 或 ÷1024；输入时钟源驱动 16 位计数器前按所选值预分频
- 计数器值达到影子寄存器值时，触发硬件中断（若已通过 RTCIE 位使能）。
- 溢出事件可能是其它模块硬件中的触发信号。有关支持该触发的模块的详细信息，请参见具体器件的数据表。
- 通过将 RTCSR 位置 1，软件可以复位计数器。

图 12-1 给出了 RTC 计数器的框图。

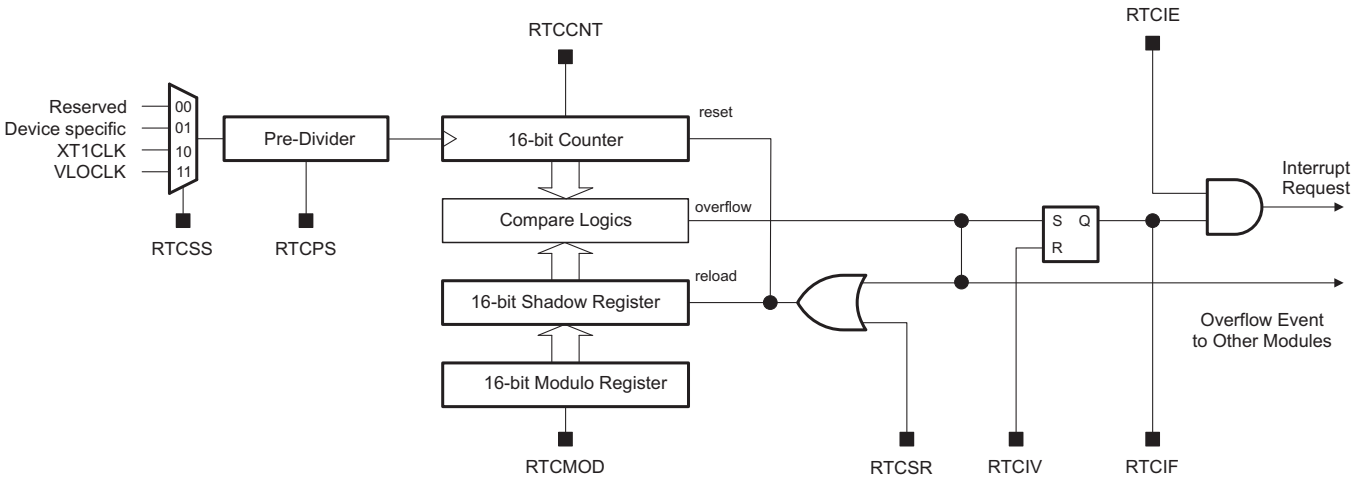


图 **12-1. RTC** 计数器框图

12.2 RTC 计数器操作

通过用户软件配置 RTC 计数器模块。RTC 计数器的设置和操作在后续各小节进行讨论。

12.2.1 16 位定时器计数器

该 16 位定时器计数器寄存器 RTCCNT 在源时钟信号的上升沿递增。只能通过软件读取 RTCCNT。计数器值达到影子寄存器值时，RTC 计数器会生成溢出信号，计数器值复位为零，并且计数器继续计数而不中断。只要 RTCSS 位指定的计数器时钟源有效，计数器就工作。

RTCCNT 可由溢出事件清零，或可通过软件将逻辑 1 写入 RTCCTL 寄存器中的 RTCSR 位复位。通过软件复位计数器时，在分频时钟的下一周期使用模寄存器中的值更新影子寄存器，但不生成溢出事件或中断。

LPM3.5 期间计数器的最大输入频率为 40kHz。因此，必须配置预分频器使得分频时钟频率不超过 40kHz。

12.2.2 时钟源选择和分频器

在 AM 和 LPM0 模式下，RTC 计数器时钟可由器件特定时钟（SMCLK 或 ACLK）、XT1CLK 或 VLOCLK 提供。在 LPM3 模式下，可选择 ACLK、XT1CLK 或 VLOCLK。在 LPM3.5 模式下，只能选择 XT1CLK 或 VLOCLK。时钟源由 RTCCTL 寄存器中的 RTCSS 位指定。复位后，RTCSS 默认为 00b（禁止），表示未选择时钟源。

计数器使用所选时钟源前，可以预分频所选时钟源。如果选择了直通模式 (÷1)，则预分频器被旁路，所选时钟源直接提供给计数器。预分频器选项 ÷16、÷64、÷256 和 ÷1024 允许对值为 2 次幂（例如来自 32768Hz 晶振）的时钟源频率进行简单分频。预分频器选项 ÷10、÷100 和 ÷1000 允许对值为 10 的倍数（例如来自 4MHz 或 8MHz 时钟输入）的时钟源频率进行简单分频。

注: **LPM3.5 下的所选时钟源**

在 LPM3.5 模式下，RTC 计数器的功耗极低，经过分频驱动计数器的时钟源可以有最大频率 40kHz。

12.2.3 模寄存器 (RTCMOD) 和影子寄存器

模寄存器 (RTCMOD) 是由用户软件设置的 16 位寄存器。RTCMOD 中的值是锁定的，直到装载到影子寄存器中才生效。影子寄存器也是 16 位寄存器，它存储 RTC 计数器用来与计数器值进行逻辑比较的模值。影子寄存器用作 RTCMOD 寄存器的缓冲器，使得软件可以设置新模值而无需中断计数器。用户可以对 RTCMOD 寄存器进行读写访问。用户无法访问影子寄存器。

在以下两种情况下，RTCMOD 中的值将装载到影子寄存器中：

1. 计数器值达到影子寄存器值时，此时还将触发溢出信号并将计数器值清零。
2. 通过软件将逻辑 1 写入 RTCCTL 寄存器的 RTCSR 位触发软件复位时。

因为影子寄存器始终通过 RTCMOD 更新值，所以软件必须在硬件溢出发生前将 RTCMOD 置 1。通过软件复位，软件可以立即将目标模值置入影子寄存器而无需等待下次溢出。如果在硬件溢出发生时未更新 RTCMOD 中的值，则影子寄存器将取出存储在 RTCMOD 中的上一个模值。如果在溢出前多次更新 RTCMOD，则仅最后一个值装载到影子寄存器中。

RTCMOD 置为 x0000 或 0x0001 时，RTC 计数器总是生成溢出。

将 RTCMOD 置 1 时应该注意避免溢出事件发生过快而无法加以处理。当所选 RTC 计数器时钟源频率接近 CPU 时钟频率时，模值必须足够长以便 CPU 在下次 RTC 计数器中断发生前能够及时响应 RTC 计数器中断服务程序 (ISR)。此外，频繁写入 RTCSR 位（软件复位）可能导致丢失溢出事件，因为每次都会复位计数器，从而 RTCMOD 设置会覆盖当前的影子寄存器设置。

图 12-2 显示了硬件溢出事件将新值 (0x2000) 从 RTCMOD 装载到影子寄存器中，替代上一个值 (0x4000) 的过程。

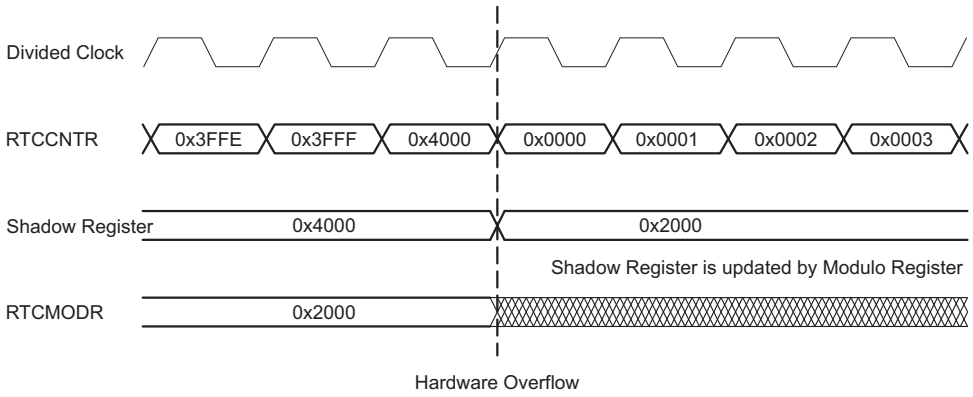


图 **12-2.** 影子寄存器示例

12.2.4 RTC 计数器中断和外部事件/触发器

存在与 16 位 RTC 计数器模块中断标志 (RTCIFG) 相关的中断向量 (RTCIV)。

发生溢出时，RTCCTL 寄存器的 RTCIFG 位置 1，直到 RTCIV 寄存器读操作将其清零。同时，如果 RTCCTL 寄存器的 RTCIE 位置 1，则向 CPU 发出中断以进行后处理。读取 RTCIV 寄存器会清除中断标志。

TI 建议在使能 RTC 计数器中断前通过读取 RTCIV 寄存器将 RTCIFG 位清零。否则，如果 RTCIFG 已由上一次溢出置 1，则可能生成中断。

除中断外，硬件溢出还将向其它片上模块提交外部触发信号作为同步信号。有关特定器件提供的模块触发器的更多详细信息，请参见具体器件的数据表。

12.3 RTC 计数器寄存器

表 12-1 列出了 RTC 计数器寄存器和每个寄存器的地址偏移量。有关模块的基址，请参见具体器件的数据表。

表 12-1. RTC 计数器寄存器

偏移量	首字母缩写词	寄存器名称	类型	访问	复位	部分
00h	RTCCTL	实时时钟控制	读取/写入	字	0000h	12.3.1 节
00h	RTCCTL_L		读取/写入	字节	00h	
01h	RTCCTL_H		读取/写入	字节	00h	
04h	RTCIV	实时时钟中断向量	读取/写入	字	0000h	12.3.2 节
04h	RTCIV_L		读取/写入	字节	00h	
05h	RTCIV_H		读取/写入	字节	00h	
08h	RTCMOD	实时计时器时钟模	读取/写入	字	BEEFh	12.3.3 节
08h	RTCMOD_L		读取/写入	字节	EFh	
09h	RTCMOD_H		读取/写入	字节	BEh	
0Ch	RTCCNT	实时时钟计数器	读取	字	0000h	12.3.4 节
0Ch	RTCCNT_L		读取	字节	00h	
0Dh	RTCCNT_H		读取	字节	00h	

12.3.1 RTCCTL 寄存器

RTC 计数器控制寄存器

图 12-3. RTCCTL 寄存器

15	14	13	12	11	10	9	8
保留		RTCSS		保留	RTCPS		
r0	r0	rw-{0}	rw-{0}	r0	rw-{0}	rw-{0}	rw-{0}
7	6	5	4	3	2	1	0
保留	RTCSR	保留				RTCIE	RTCIFG
r0	w-{0}	r0	r0	r0	r0	rw-{0}	r-{0}

表 12-2. RTCCTL 寄存器说明

位	字段	类型	复位	说明
15-14	保留	R	0h	保留
13-12	RTCSS	RW	0h	实时时钟源选择 00b = 保留 01b = 特定于器件 10b = XT1CLK 11b = VLOCLK
11	保留	R	0h	保留
10-8	RTCPS	RW	0h	实时时钟预分频器选择 000b = /1 001b = /10 010b = /100 011b = /1000 100b = /16 101b = /64 110b = /256 111b = /1024
7	保留	R	0h	保留
6	RTCSR	W	0h	实时软件复位。该位为只读位，始终通过逻辑 0 读取。 0b = 写 0 无效 1b = 向该位写 1 会将计数器值清零，并在所选源时钟的下个周期从模寄存器重新装载影子寄存器值。不生成溢出事件或中断。
5-2	保留	R	0h	保留
1	RTCIE	RW	0h	实时中断使能 0b = 中断被禁用 1b = 中断被启用
0	RTCIFG	R	0h	实时中断标志。该位报告挂起中断的状态。可以通过读取 RTCIV 寄存器清零该只读位。 0b = 无中断挂起 1b = 中断挂起

12.3.2 RTCIV 寄存器

RTC 计数器中断向量寄存器

图 12-4. RTCIV 寄存器

15	14	13	12	11	10	9	8
RTCIV							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
RTCIV							
r0	r0	r0	r0	r0	r0	r-{0}	r0

表 12-3. RTCIV 寄存器说明

位	字段	类型	复位	说明
15-0	RTCIV	R	0h	低功耗计数器中断向量。 00h = 无中断挂起 02h = 中断源：RTC 计数器溢出；中断标志：RTCIFG

12.3.3 RTCMOD 寄存器

RTC 计数器模寄存器

图 **12-5. RTCMOD** 寄存器

15	14	13	12	11	10	9	8
RTCMOD							
rw-{1}	rw-{0}	rw-{1}	rw-{1}	rw-{1}	rw-{1}	rw-{1}	rw-{0}
7	6	5	4	3	2	1	0
RTCMOD							
rw-{1}	rw-{1}	rw-{1}	rw-{0}	rw-{1}	rw-{1}	rw-{1}	rw-{1}

表 **12-4. RTCMOD** 寄存器说明

位	字段	类型	复位	说明
15-0	RTCMOD	RW	BEEFh	RTC 模值

12.3.4 RTCCNT 寄存器

RTC 计数器寄存器

图 **12-6. RTCCNT** 寄存器

15	14	13	12	11	10	9	8
RTCCNT							
r-{0}	r-{0}	r-{0}	r-{0}	r-{0}	r-{0}	r-{0}	r-{0}
7	6	5	4	3	2	1	0
RTCCNT							
r-{0}	r-{0}	r-{0}	r-{0}	r-{0}	r-{0}	r-{0}	r-{0}

表 **12-5. RTCCNT** 寄存器说明

位	字段	类型	复位	说明
15-0	RTCCNT	R	0h	RTC 计数器。该寄存器为只读寄存器，反映当前的计数器值。

ADC 模块

ADC 模块是一款高性能 10 位模数转换器 (ADC)。本章对 ADC 模块的操作进行了说明。

Topic **Page**

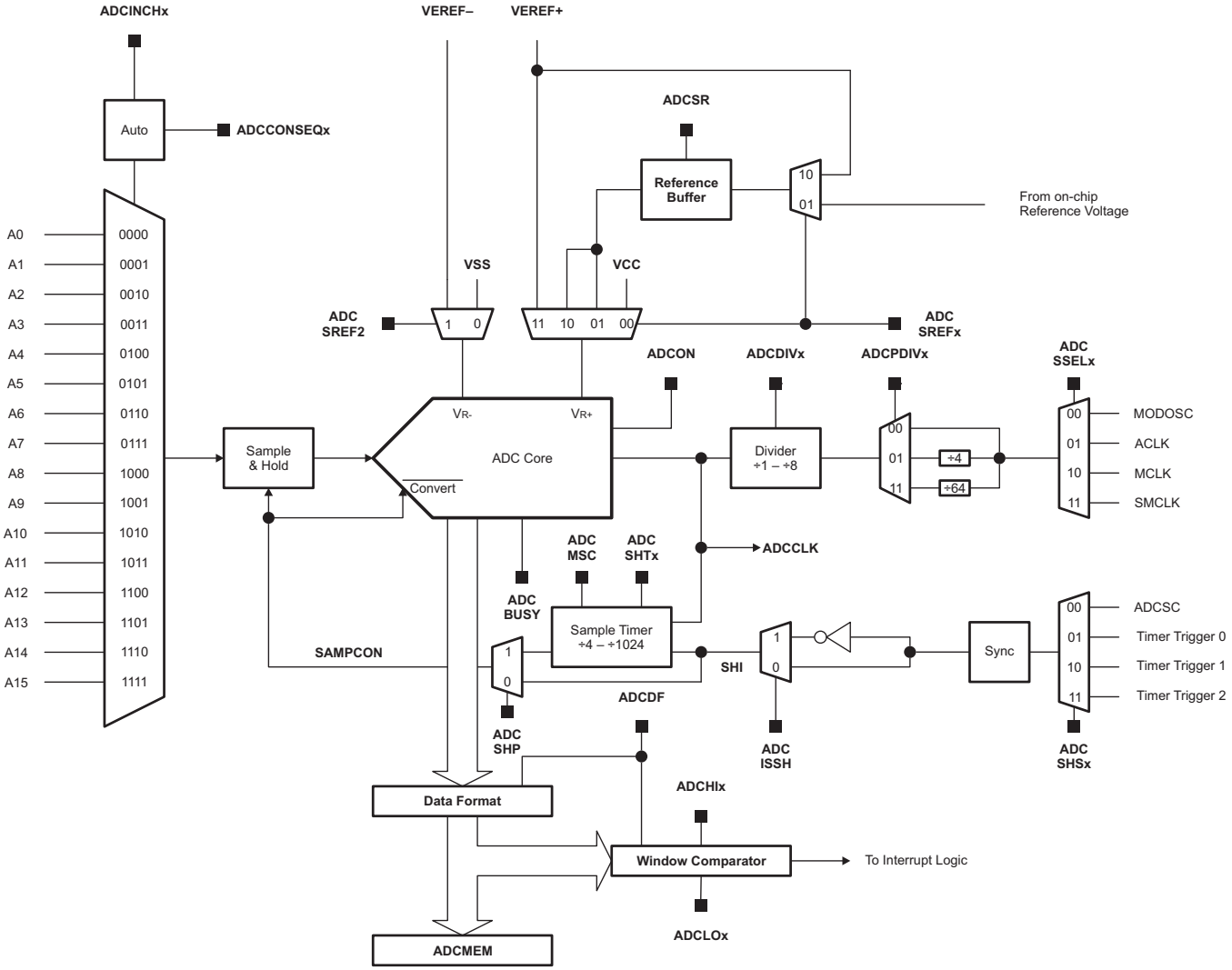
13.1 ADC 简介

ADC 模块支持高速 10 位模数转换。该模块具有一个 10 位 SAR 内核、采样选择控制功能和一个窗口比较器。

ADC 模块特性包括：

- 大于 200ksps 的最大转换速率
- 无丢码的单调10 位转换器，
- 由软件或定时器控制的采样周期可编程的采样保持功能
- 由软件或不同的定时器启动转换
- 软件可选片上基准或外部基准
- 十二个可独立配置的外部输入通道
- 适用于片上温度传感器的转换通道
- 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
- 适用于输入信号低功耗监视的窗口比较器
- 用于对六个 ADC 中断（ADCIFG0、ADCTOVIFG、ADCOVIFG、ADCLOIFG、DCINIFG、ADCHIIFG）快速解码的中断向量寄存器。

图 13-1 显示了 ADC 模块的框图。



A MODOSC 是 CS 的一部分。更多信息，请参见 CS 一章。

B 当使用 ADCSHP = 0 时，触发输入不会实现同步。

图 **13-1. ADC** 框图

13.2 ADC 操作

通过用户软件配置 ADC 模块。ADC 的设置和操作在后续各小节进行讨论。

13.2.1 10 位 ADC 内核

该 ADC 内核将模拟输入转换为 10 位数字表示，并将结果存储在转换寄存器 ADCMEM0 中。该内核利用两个可编程和可选的基准电压（V_{R+} 和 V_{R-}）来定义转换的上限和下限。数字输出 (N_{ADC}) 在输入信号大于等于 V_{R+} 时为满量程 (03FFh)，在输入信号小于等于 V_{R-} 时为零。输入通道和基准电压（V_{R+} 和 V_{R-}）在转换控制存储器中定义。ADC 结果 N_{ADC} 的转换公式为：

$$N_{ADC} = 1023 \times \frac{Vin - V_{R-}}{V_{R+} - V_{R-}}$$

ADC 内核由控制寄存器 ADCCTL0、ADCCTL1 和 ADCCTL2 配置。使用 ADCON 位使能内核。可以在不使用时关闭 ADC 以实现节能。除少数情况外，通常仅能在 ADCENC = 0 时修改这几个 ADC 控制位。在执行任何转换前，ADCENC 必须置 1。

13.2.1.1 转换时钟选择

ADCCLK 可作为转换时钟使用，也可在选择脉冲采样模式后用来生成采样周期。使用 ADCSSELx 位选择 ADC 时钟源。可用的 ADCCLK 时钟源有 SMCLK、ACLK 和 MODOSC。使用 ADCDIVx 位和 ADCPDIVx 位可实现输入时钟的 1 到 512 分频。

CS 内部生成的 MODOSC 在 5 MHz 范围内，但会因器件、电压和温度而变化。MODOSC 的技术规格请参见具体器件的数据表。

用户必须确保为 ADCCLK 选择的时钟在转换结束前一直有效。如果该时钟在转换期间被移除，则会导致转换操作不完整，转换结果无效。

13.2.2 ADC 输入和多路复用器

12 个外部和 4 个内部模拟信号由模拟输入多路复用器选为转换通道。输入多路复用器是先关后开型来减少因通道切换而引入的输入到输入噪声（请见图 13-2）。输入多路复用器也是一个 T 型开关，可大大减少通道间的耦合。未选择的通道与 A/D 隔离开，并且中间节点被连接到模拟接地 (AV_{SS})，这样寄生电容接地从而减少串扰。

ADC 利用电荷再分配方法。当内部切换输入的开关状态时，开关动作可能导致输入信号瞬变。在引起错误转换前，这些瞬变会衰减并稳定下来。

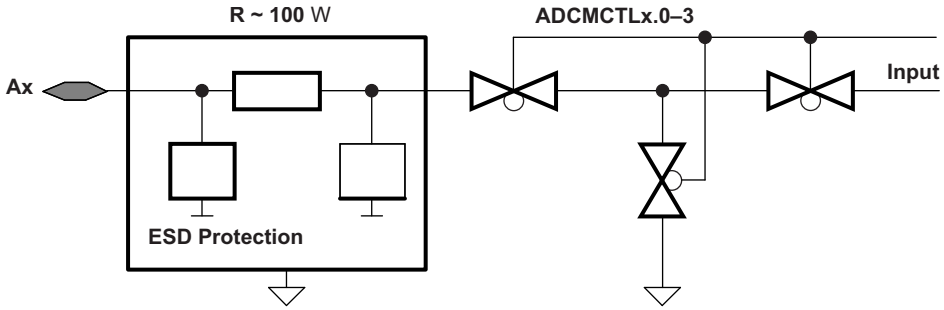


图 **13-2.** 模拟多路复用器

13.2.2.1 模拟端口选择

ADC 输入与数字端口引脚复用。当模拟信号被应用到数字栅极时，寄生电流可以从 V_{CC}流向 GND。如果输入电压接近该栅极的转换电平时，就会产生寄生电流。禁用端口引脚的数字部分可消除寄生电流，因此，可降低总流消。PySELx 位提供了禁用端口引脚输入和输出缓冲器的功能。

```
; Py.0 and Py.1 configured for analog input
BIS.B #3h,&PySEL ; Py.1 and Py.0 ADC function
```

13.2.3 基准电压发生器

ADC 模块可使用片上基准电压或外部引脚上提供的外部基准电压。

片上基准电压在具体器件的数据表中给出。

外部基准电压可通过引脚 VEREF+ 和 VEREF- 分别为 V_{R+} 和 V_{R-} 供电。

13.2.3.1 内部基准电压低功耗特性

片载基准被设计用于低功耗应用。此基准包括 PMM 模块中的带隙基准电压源。每个基准的电流消耗分别在具体器件的数据表中给出。ADC 还包含一个内部基准电压缓冲器。为 V_{REF+} 选择内部基准电压后，该缓冲器自动使能，且其还可选择性用于 Ve_{REF+}。PMM 模块的片上基准电压需用通过软件使能。其稳定时间小于等于 30µs。关于片上基准电压的更多信息，请参见 PMM 模块章节。

ADC 的基准缓冲器还有相对于功率设置的可选速度。当最大转换速率小于 50ksps 时，ADCSR 置 1 可将缓冲器电流消耗减少约 50%。

13.2.4 自动断电

ADC 旨在用于低功耗应用。当 ADC 未积极执行转换时，内核被自动禁用并在需要时自动重新使能。MODOSC 也是在需要时被自动启用而在不需要时自动被禁用。

13.2.5 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 的信号源可通过 ADCSHSx 位进行选择，其包括以下内容：

- ADCSC 位
- 三个定时器输出

SHI 信号源可通过 ADCISSH 位来实现极性反相。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。SAMPCON 由高到低切换会启动模数转换，这要求在 10 位分辨率模式中有 11 个 ADCCLK 周期。窗口比较器需要另一个额外的 ADCCLK 周期。通过控制位 ADCSHP 定义两种不同的采样定时方法：扩展采样模式和脉冲采样模式。

13.2.5.1 扩展采样模式

当 ADCSHP = 0 时选择扩展采样模式。SHI 信号直接控制 SAMPCON 并定义采样周期 t_{sample}的长度。当 SAMPCON 为高电平时，采样是激活的。在与 ADCCLK 同步后，SAMPCON 由高到低切换会启动转换（请参见图 13-3）。SHI 信号至少需要 4 个 ADCCLK 周期。

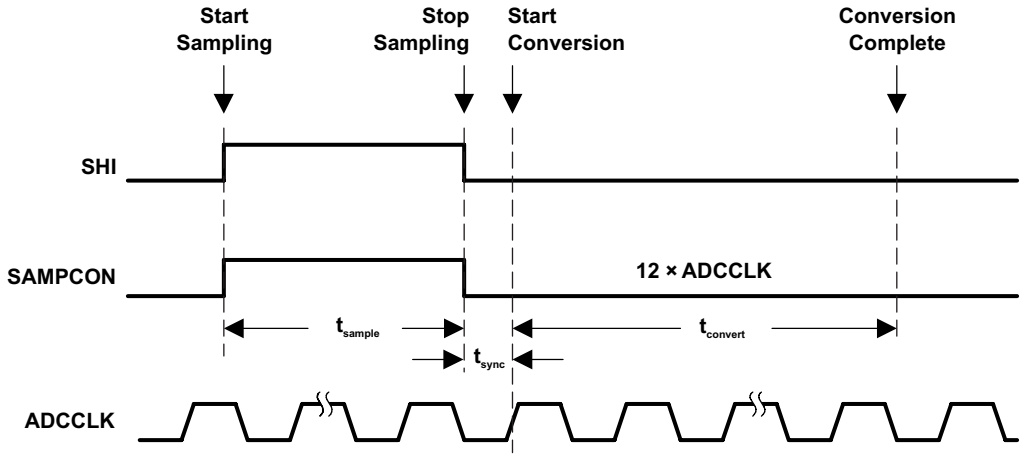


图 **13-3.** 扩展采样模式

13.2.5.2 脉冲采样模式

当 ADCSHP = 1 时选择脉冲采样模式。SHI 信号用于触发采样定时器。ADCCTL0 中的 ADCSHTx 位控制采样定时器的间隔，以定义 SAMPCON 采样时长 t_{sample}。。在一个已设定的时间间隔 t_{sample}内，当与 AD10CLK 同步后，采样定时器保持 SAMPCON 为高电平。总的采样时间为 t_{sample}加上 t_{sync}（请见图 13-4）。

ADCSHTx 位选择 4 倍于 ADCCLK 的采样时间。如果 ADCSC 位在该模式中作为采样保持源使用，它将自动清零。

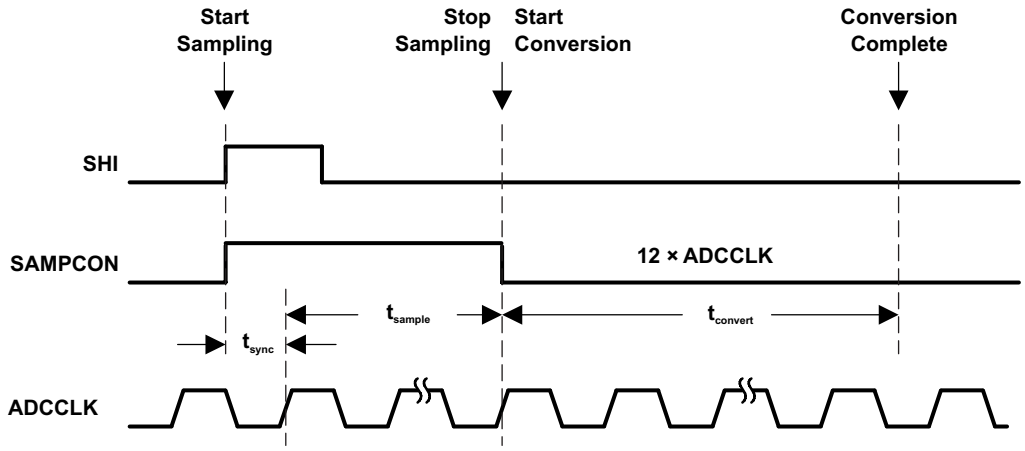


图 **13-4.** 脉冲采样模式

13.2.5.3 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_{sample}期间，已选的 Ax 输入相当于一个 RC 低通滤波器（请见图 13-5）。从源端看，内部多路复用接通输入电阻 R_I（请参见具体器件的数据表）与电容 C_I（请参见具体器件的数据表）串联。为了进行准确的 10 位转换，电容器 C_I电压 V_C必须被充电至电源电压 V_S的一半 LSB 范围内。

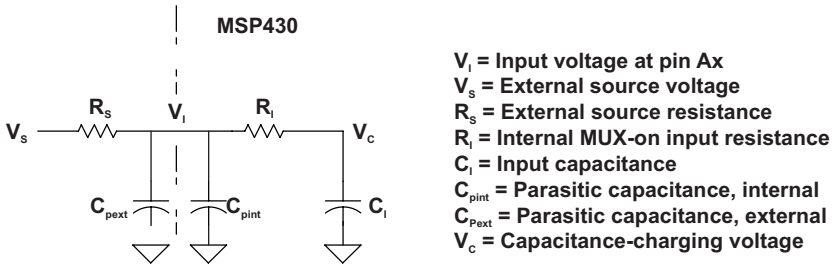


图 13-5. 模拟输入等效电路

源 R_S 和 R_I 的电阻影响 t_{sample}。有关 t_{sample} 限制，请参见具体器件的数据表。

13.2.6 转换结果

使用 ADCMEM0 寄存器可访问转换结果，转换结果与用户所选转换模式无关。转换结果写入 ADCMEM0 时，ADCIFG0 置 1。

13.2.7 ADC 转换模式

ADC 有四种运行模式，可通过 CONSEQx 位进行选择（见表 13-1）。

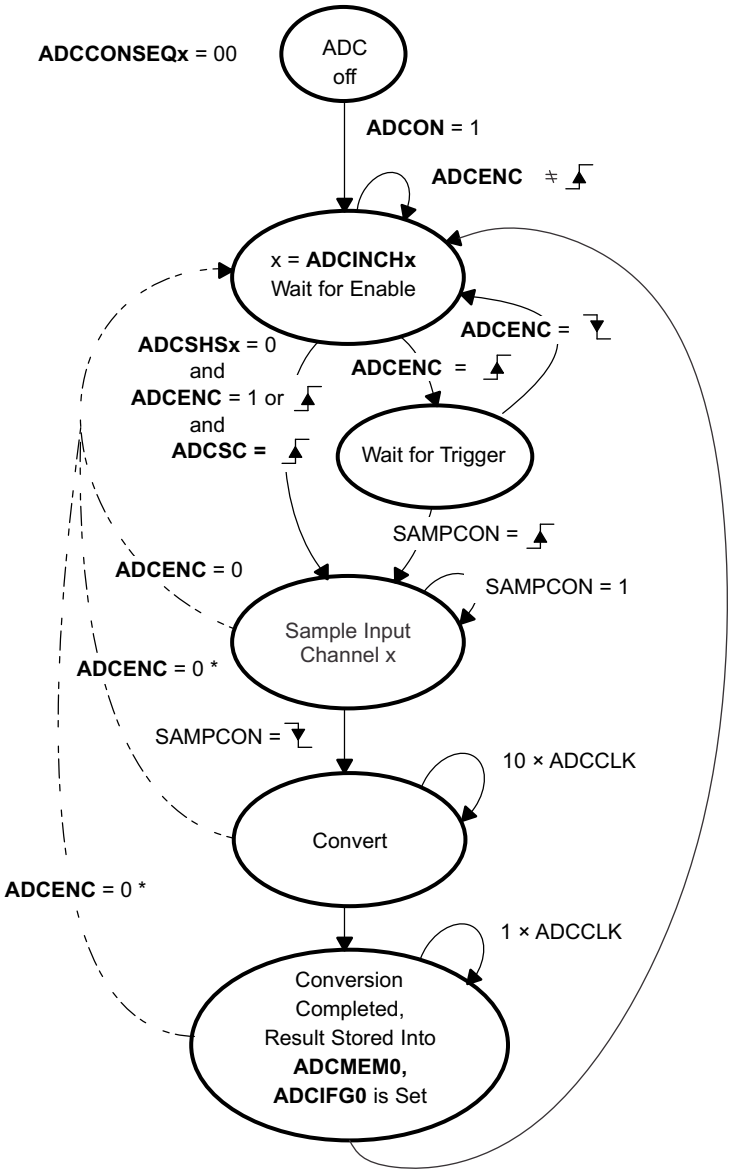
表 13-1. 转换模式汇总

ADCCONSEQx	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

13.2.7.1 单通道单次转换模式

ADCINCHx 选择的单通道被采样和转换一次。ADC 结果写入 ADCMEM0。图 13-6 显示了单通道单次转换模式的流程。ADCSC 触发转换后，由该 ADCSC 位触发后继的转换。若使用其他触发源，则在每次转换之间必须切换 ADCENC。

转换期间复位 ADCON 位，会导致 ADC 回到“ADC 关闭”状态。这种情况下，转换寄存器的值和中断标志的值无法预测。

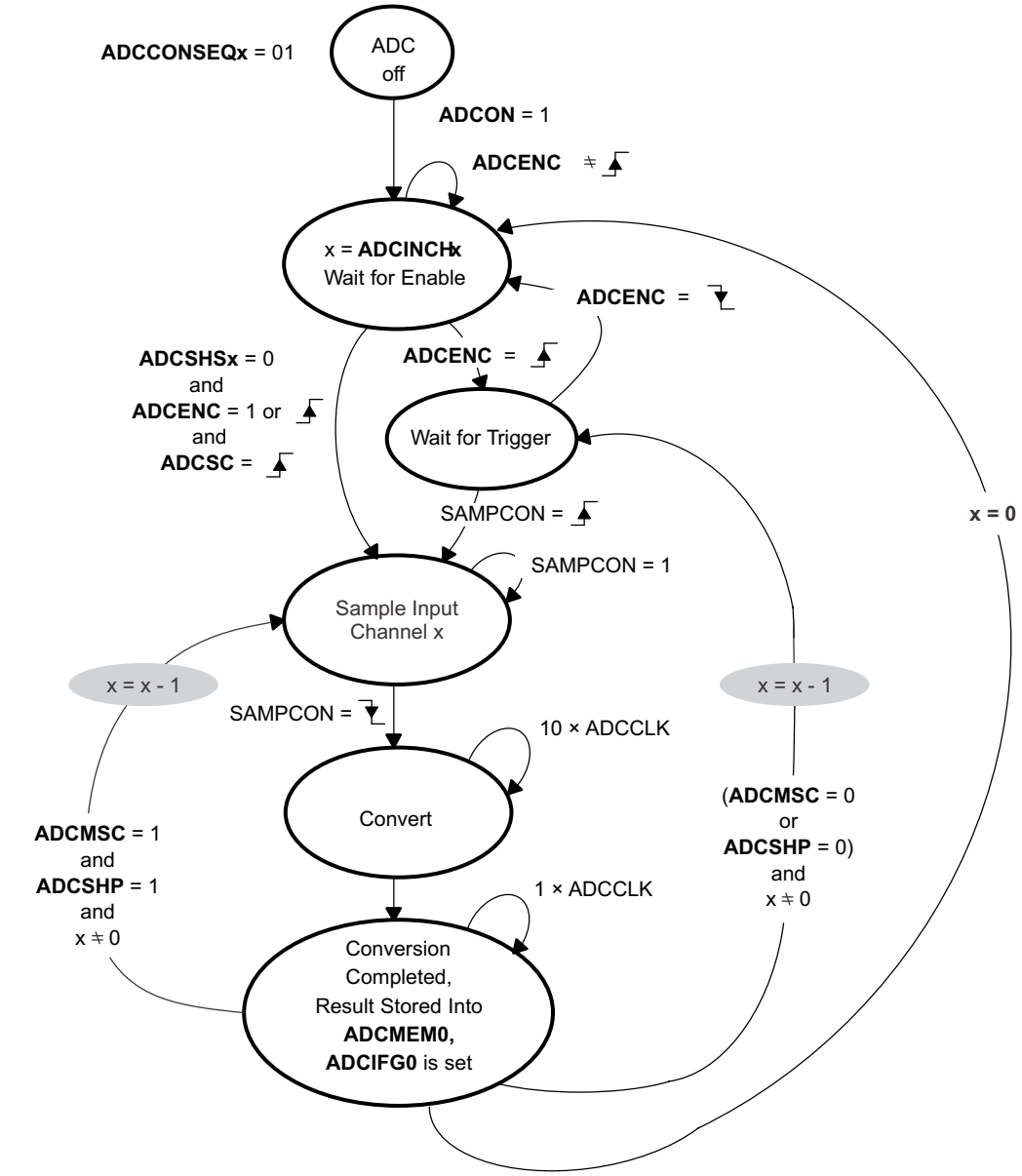


* = Conversion result is unpredictable
x = Pointer to the selected ADC channel defined by **ADCINCHx**
All bit and register names are in bold font; signals are in normal font.

图 **13-6.** 单通道单次转换模式

13.2.7.2 通道序列模式

一个通道序列被采样和转换一次序列以 ADCINCHx 位选择的通道开始，递减到通道 A0 为止。每个 ADC 结果都写入 ADCMEM0。该序列在通道 A0 转换后停止。图 13-7显示了通道序列模式。ADCSC 触发序列后，由该 ADCSC 位触发后继的序列。若使用其他触发源，则在每个序列之间必须切换 ADCENC。如同在所有转换模式中那样，转换期间复位 ADCON 位，会导致 ADC 回到“ADC 关闭”状态。

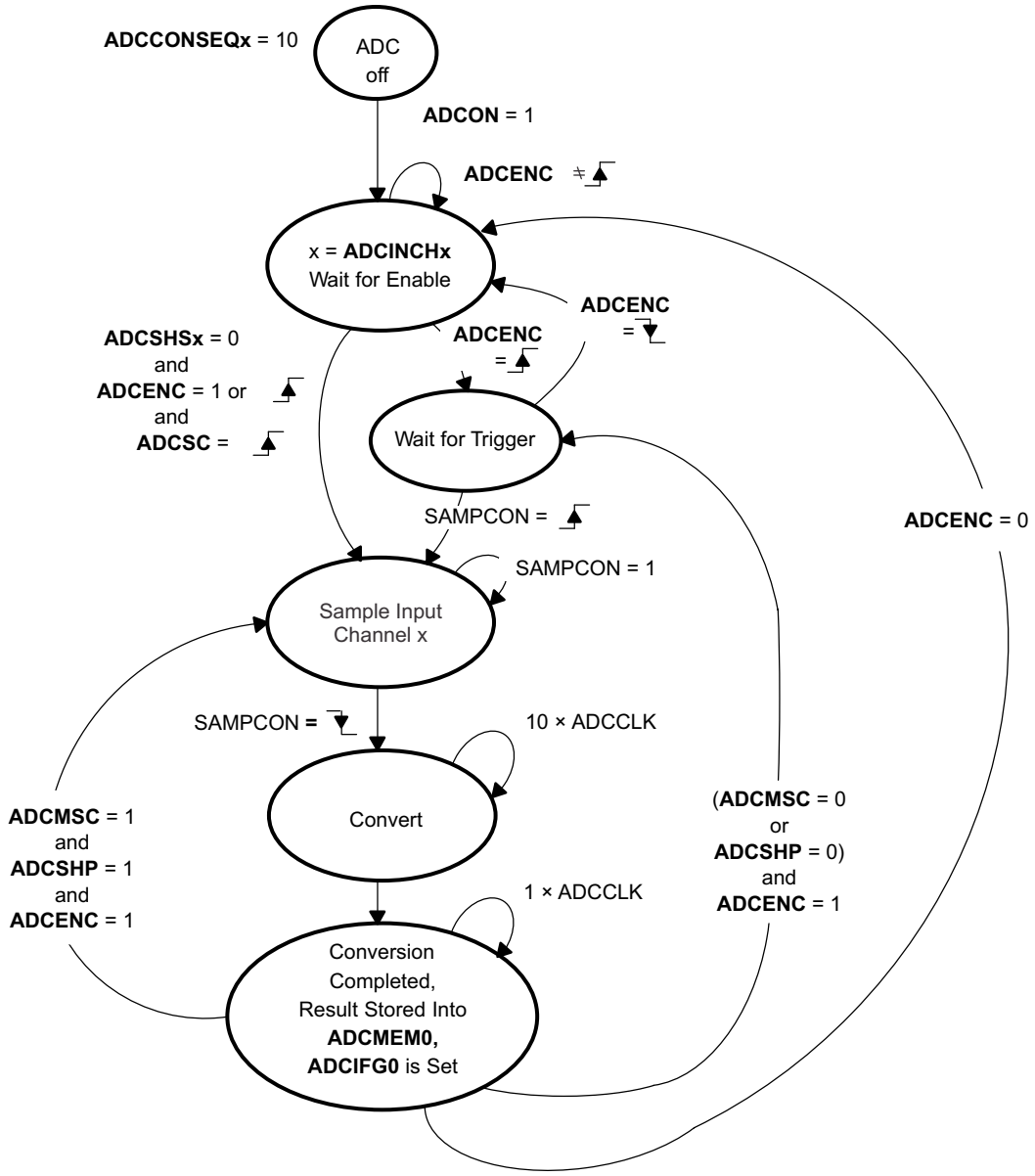


x = Input channel Ax
All bit and register names are in bold font; signals are in normal font.

图 13-7. 通道序列模式

13.2.7.3 单通道重复模式

ADCINCHx 选择的单通道连续被采样和转换。每个 ADC 结果都写入 ADCMEM0。图 13-8显示了单通道重复模式。

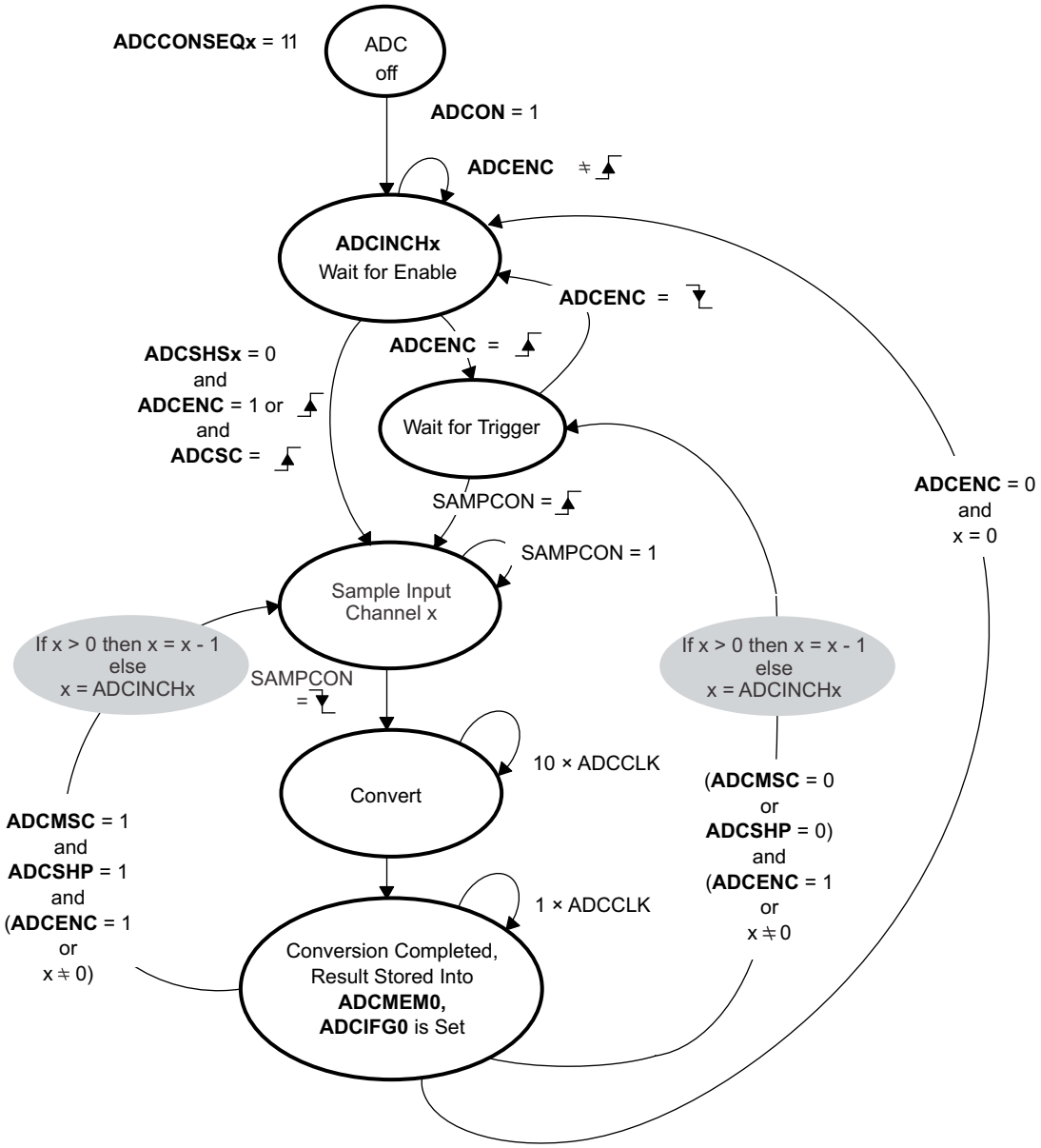


x = Pointer to the selected ADC channel defined by **ADCINCHx**
All bit and register names are in bold font; signals are in normal font.

图 **13-8.** 单通道重复模式

13.2.7.4 通道的重复序列模式

一个通道序列被重复采样和转换。序列以 ADCINCHx 位选择的通道开始，递减到通道 A0 为止。每个 ADC 结果都写入 ADCMEM0。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 13-9显示了通道的重复序列模式。



x = Input channel Ax
All bit and register names are in bold font; signals are in normal font.

图 **13-9.** 通道的重复序列模式

13.2.7.5 使用多重采样和转换 (ADCMSC) 位

为配置转换器尽快自动执行连续转换，可以使用多重采样和转换功能。当 ADCMSC = 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次转换。前面的转换一旦完成，则立即自动触发相继的转换。单个序列模式下的序列完成之前，或者单通道重复或重复序列模式下的 ADCENC 位切换之前，将忽略 SHI 的其他上升沿。使用 ADCMSC 位时，ADCENC 位的功能不变。

13.2.7.6 停止转换

是否停止 ADC 活动取决于运行模式。停止活动转换或转换序列的建议方法是：

- 在单通道单次转换模式下复位 ADCENC 可立即停止转换，但结果不可预测。要获得正确的结果，需在 ADCENC 清零前轮询繁忙位。
- 在单通道重复运行期间复位 ADCENC 可在当前转换结束时停止转换器。
- 在序列或重复序列模式下复位 ADCENC 可在序列结束时停止转换器。
- 设置 CONSEQx = 0 并复位 ADCENC 位可立即停止任何转换模式。转换数据不可信靠。

13.2.7.7 窗口比较器

窗口比较器可在无需 CPU 干预的情况下监视模拟信号。在以下列表中可查看可用的中断标志及其中断条件：

- 如果当前 ADC 转换结果小于寄存器 ADCLO 中定义的阈值下限，则 ADCLO 中断标志 (ADCLOIFG) 置 1。
- 如果当前 ADC 转换结果大于寄存器 ADCHI 中定义的阈值上限，则 ADCHI 中断标志 (ADCHIIFG) 置 1。
- 如果当前 ADC 转换结果介于寄存器 ADCLO 中定义的阈值下限与寄存器 ADCHI 中定义的阈值上限之间，则 ADCIN 中断标志 (ADCINIFG) 置 1。

这些中断的生成与用户所选转换模式无关。

用户始终需要确保 ADCHI 和 ADCLO 寄存器中的值采用了正确的数据格式。例如，如果选择二进制数据格式 (ADCDF = 0)，则阈值寄存器 ADCHI 和 ADCLO 中的阈值也需要以二进制编码的格式输入。更改 ADCDF 或 ADCRES 可复位阈值寄存器。

中断标志需要通过用户软件复位。ADC 仅在每次 ADCMEM0 中提供新值时更新中断标志。该更新只涉及相应的一组中断标志。当使用窗口比较器时，软件必须根据应用需要来复位标志。

13.2.7.8 使用集成温度传感器

为使用片上温度传感器，需选择模拟输入通道 ADCINCHx = 1100b。如同选择了外部通道那样完成其他配置，包括基准选择、转换模式选择及所有其他设置。温度传感器必须通过软件激活。

图 13-10 给出了典型温度传感器的传递函数。使用温度传感器时，采样周期必须大于 30µs。温度传感器偏移误差可能较大，对于大多数应用来说需要进行校正（参见具体器件的数据表了解相关参数）。

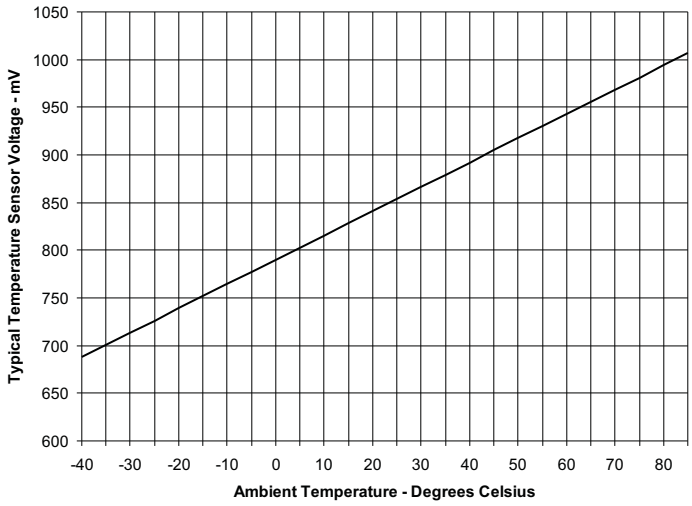


图 **13-10.** 典型温度传感器传递函数

13.2.7.9 ADC 接地和噪声注意事项

与任何高分辨率 ADC 一样，应遵循适当的印刷电路板布局和接地技术，以消除接地环路，不必要的寄生效应和噪声。

来自 ADC 的返回电流流经与其他模拟或数字电路共用的路径时，就会形成接地环路。如果不小心，这个电流会产生小的、有害的偏移电压，该电压可以增加或减少 ADC 的基准或输入电压。图 13-11中显示的连接可防止此情况。

除了接地，由数字切换或切换电源所引起的电源线路上的纹波和噪声尖峰脉冲会破坏转换结果。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接地层的无噪音设计。

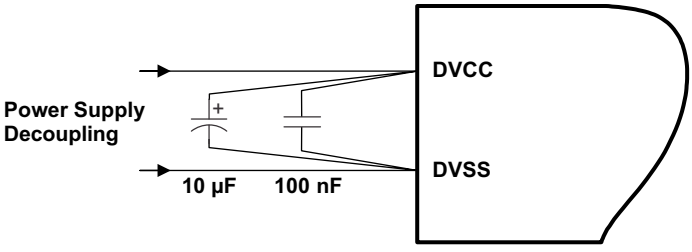


图 **13-11. ADC** 接地和噪声注意事项

13.2.7.10 ADC 中断

ADC 有六个中断源：

- ADCIFG0：转换就绪中断
- ADCOVIFG：ADCMEM0 溢出
- ADCTOVIFG：ADC 转换时间溢出
- ADCLOIFG、ADCINIFG 和 ADCHIIFG：窗口比较器中断标志

当转换结果加载到 ADCMEM0 存储寄存器时，ADCIFG0 位置 1。如果 ADCIE0 位和 GIE 位置 1，则生成中断请求。将转换结果写入 ADCMEM0 时，如果尚未读取其前一个转换结果，则出现 ADCOV 条件。如果在当前的转换完成前请求另一个采样和转换，则生成 ADCTOV 条件。

根据窗口比较器章节中的说明将窗口比较器中断标志置 1（参见节 13.2.7.7）。

13.2.7.10.1 ADCIV，中断向量发生器

全部 ADC 中断源进行了优先级排序且共用一个中断向量。中断向量寄存器 ADCIV 用于确定以使能的、请求中断的 ADC 中断源。

优先级最高的已使能 ADC 中断会在 ADCIV 寄存器中生成一个数字（请参见寄存器说明）。可评估该数字或将其添加到程序计数器 (PC)，以便自动进入相应的软件程序。禁用的 ADC 中断不影响 ADCIV 值。

对 ADCIV 寄存器执行读访问会自动将最高挂起中断条件和标志复位。仅 ADCIFG0 不会由此 ADCIV 读访问复位。ADCIFG0 可通过读取 ADCMEM0 寄存器来实现自动复位，或使用软件实现复位。

对 ADCIV 寄存器执行写访问会将所有挂起中断条件和标志清零。

在处理一个中断后，如果有另一个中断挂起，则会产生另一个中断。例如，当中断服务程序访问 ADCIV 寄存器时，如果 ADCOV、ADCHIIFG 和 ADCIFG0 中断挂起，优先级最高的中断（ADCOV 中断条件）会自动复位。在中断服务程序的 RETI 指令执行结束后，ADCHIIFG 会生成另一个中断。

13.2.7.10.2 ADC 中断处理软件示例

以下软件示例给出了 ADCIV 的建议用法。将 ADCIV 值添加到 PC 以自动跳转到相应的程序。

- ADCIFG0、ADCTOV 和 ADCOV：16 个周期

```
; Interrupt handler for ADC.
INT_ADC                            ; Enter Interrupt Service Routine
 ADD   &ADCIV,PC                     ; Add offset to PC
 RETI                                ; Vector  0: No interrupt
 JMP   ADOV                          ; Vector  2: ADC overflow
 JMP   ADTOV                         ; Vector  4: ADC timing overflow
 JMP   ADHI                          ; Vector  6: ADC window comparator high
interrupt
 JMP   ADLO                          ; Vector  8: ADC window comparator low interrupt
 JMP   ADIN                          ; Vector 10: ADC window comparator in interrupt
;
; Handler for ADCIFG0 starts here. No JMP required.
;
ADMEM  MOV &ADCMEM0,xxx              ; Move result, flag is reset
       ...                           ; Other instruction needed?
       RETI                          ; Return ;
ADOV   ...                           ; Handle ADCMEM0 overflow
       RETI                          ; Return ;
ADTOV  ...                           ; Handle Conv. time overflow
       RETI                          ; Return ;
ADHI   ...                           ; Handle window comparator high interrupt
       RETI                          ; Return ;
ADLO   ...                           ; Handle window comparator low interrupt
       RETI                          ; Return ;
ADIN   ...                           ; Handle window comparator in window interrupt
       RETI                          ; Return
```

13.3 ADC 寄存器

表 13-2 中列出了 ADC 寄存器。有关 ADC 的基址的详细信息，请参见具体器件的数据表。表 13-2 中给出了每个 ADC 寄存器的地址偏移量。

表 13-2. ADC 寄存器

偏移量	首字母缩写词	寄存器名称	类型	复位	部分
00h	ADCCTL0	ADC 控制寄存器 0	读取/写入	0000h	13.3.1 节
02h	ADCCTL1	ADC 控制寄存器 1	读取/写入	0000h	13.3.2 节
04h	ADCCTL2	ADC 控制寄存器 2	读取/写入	1000h	13.3.3 节
06h	ADCLO	ADC 窗口比较器阈值下限寄存器	读取/写入	0000h	13.3.9 节
08h	ADCHI	ADC 窗口比较器的阈值上限寄存器	读取/写入	FF03h	13.3.7 节
0Ah	ADCMCTL0	ADC 存储器控制寄存器	读取/写入	00h	13.3.6 节
12h	ADCMEM0	ADC 转换存储器寄存器	读取/写入	未定义	13.3.4 节
1Ah	ADCIE	ADC 中断使能寄存器	读取/写入	0000h	13.3.11 节
1Ch	ADCIFG	ADC 中断标志寄存器	读取/写入	0000h	13.3.12 节
1Eh	ADCIV	ADC 中断向量寄存器	读取/写入	0000h	13.3.13 节

13.3.1 ADCCTL0 寄存器

ADC 控制寄存器 0

图 13-12. ADCCTL0 寄存器

15	14	13	12	11	10	9	8
保留				ADCSHTx			
r0	r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ADCMSC	保留		ADCON	保留		ADCENC	ADCSC
rw-(0)	r0	r0	rw-(0)	r0	r0	rw-(0)	rw-(0)

仅在 ADCENC = 0 时可修改。在转换激活情况下，通过软件复位 ADCENC = 0 和更改这些字段会立即显示相关结果。

表 13-3. ADCCTL0 寄存器说明

位	字段	类型	复位	说明
15-12	保留	R	0h	保留。始终读为 0。
11-8	ADCSHTx	RW	0h	ADC 采样保持时间。这些位定义 ADC 采样周期中的 ADCCLK 周期个数。 仅在 ADCENC = 0 时可修改。在转换激活情况下，通过软件复位 ADCENC = 0 和更改这些字段会立即显示相关结果。 0000b = 4 个 ADCCLK 周期 0001b = 8 个 ADCCLK 周期 0010b = 16 个 ADCCLK 周期 0011b = 32 个 ADCCLK 周期 0100b = 64 个 ADCCLK 周期 0101b = 96 个 ADCCLK 周期 0110b = 128 个 ADCCLK 周期 0111b = 192 个 ADCCLK 周期 1000b = 256 个 ADCCLK 周期 1001b = 384 个 ADCCLK 周期 1010b = 512 个 ADCCLK 周期 1011b = 768 个 ADCCLK 周期 1100b = 1024 个 ADCCLK 周期 1101b = 1024 个 ADCCLK 周期 1110b = 1024 个 ADCCLK 周期 1111b = 1024 个 ADCCLK 周期
7	ADCMSC	RW	0h	ADC 多重采样和转换只在序列或重复模式下有效。 仅在 ADCENC = 0 时可修改。在转换激活情况下，通过软件复位 ADCENC = 0 和更改这些字段会立即显示相关结果。 0b = 采样定时器需要 SHI 信号的上升沿来触发每个采样和转换。 1b = 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。
6-5	保留	R	0h	保留。始终读为 0。
4	ADCON	RW	0h	ADC 使能 仅在 ADCENC = 0 时可修改。在转换激活情况下，通过软件复位 ADCENC = 0 和更改这些字段会立即显示相关结果。 0b = ADC 禁用 1b = ADC 使能
3-2	保留	R	0h	保留。始终读为 0。
1	ADCENC	RW	0h	ADC 转换使能 0b = 禁用 ADC 1b = 使能 ADC
0	ADCSC	RW	0h	ADC 启动转换。软件控制的采样和转换启动。可使用同一指令来设置 ADCSC 和 ADCENC。ADCSC 可自动复位。 0b = 不启动采样和转换 1b = 启动采样和转换

13.3.2 ADCCTL1 寄存器

ADC 控制寄存器 1

图 13-13. ADCCTL1 寄存器

15	14	13	12	11	10	9	8
保留				ADCSHSx		ADCSHP	ADCISSH
r0	r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ADCDIVx			ADCSSELx		ADCCONSEQx		ADCBUSY
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r-(0)

仅在 ADCENC = 0 时可修改。在转换激活情况下，通过软件复位 ADCENC = 0 和更改这些字段会立即显示相关结果。

表 13-4. ADCCTL1 寄存器说明

位	字段	类型	复位	说明
15-12	保留	R	0h	保留。始终读为 0。
11-10	ADCSHSx	RW	0h	ADC 采样保持源选择 仅在 ADCENC = 0 时可修改。在转换激活情况下，通过软件复位 ADCENC = 0 和更改这些字段会立即显示相关结果。 00b = ADCSC 位 01b = 定时器触发器 0（请参见具体器件的数据表） 10b = 定时器触发器 1（请参见具体器件的数据表） 11b = 定时器触发器 2（请参见具体器件的数据表）
9	ADCSHP	RW	0h	ADC 采样保持脉冲采样模式选择。此位将采样定时器的输出或直接是采样输入信号选为采样信号 (SAMPCON) 源。 仅在 ADCENC = 0 时可修改。在转换激活情况下，通过软件复位 ADCENC = 0 和更改这些字段会立即显示相关结果。 0b = SAMPCON 信号来源于采样输入信号。 1b = SAMPCON 信号来源于采样定时器。
8	ADCISSH	RW	0h	ADC 反相信号采样保持 仅在 ADCENC = 0 时可修改。在转换激活情况下，通过软件复位 ADCENC = 0 和更改这些字段会立即显示相关结果。 0b = 采用输入信号不取反。 1b = 采用输入信号取反。
7-5	ADCDIVx	RW	0h	ADC 时钟分频器 仅在 ADCENC = 0 时可修改。在转换激活情况下，通过软件复位 ADCENC = 0 和更改这些字段会立即显示相关结果。 000b = 1 分频 001b = 2 分频 010b = 3 分频 011b = 4 分频 100b = 5 分频 101b = 6 分频 110b = 7 分频 111b = 8 分频
4-3	ADCSSELx	RW	0h	ADC 时钟源选择 仅在 ADCENC = 0 时可修改。在转换激活情况下，通过软件复位 ADCENC = 0 和更改这些字段会立即显示相关结果。 00b = MODCLK 01b = ACLK 10b = SMCLK 11b = SMCLK

表 13-4. ADCCTL1 寄存器说明 (continued)

位	字段	类型	复位	说明
2-1	ADCCONSEQx	RW	0h	ADC 转换序列模式选择 仅在 ADCENC = 0 时可修改。在转换激活情况下，通过软件复位 ADCENC = 0 和更改这些字段会立即显示相关结果。 00b = 单通道单次转换 01b = 通道序列 10b = 重复单通道 11 = 重复通道序列
0	ADCBUSY	R	0h	ADC 忙。该位标志着一个有效的采样或转换操作。 0b = 无运行是激活的 1b = 一个序列、采样、或转换是激活的。

13.3.3 ADCCTL2 寄存器

ADC 控制寄存器 2

图 13-14. ADCCTL2 寄存器

15	14	13	12	11	10	9	8
保留						ADCPDIVx	
r0	r0	r0	r0	r0	r0	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
保留			ADCRES	ADCDF	ADCSR	保留	
r0	r0	r0	rw-(1)	rw-(0)	rw-(0)	r0	rw-(0)

仅在 ADCENC = 0 时可修改。在转换激活情况下，通过软件复位 ADCENC = 0 和更改这些字段会立即显示相关结果。

表 13-5. ADCCTL2 寄存器说明

位	字段	类型	复位	说明
15-10	保留	R	0h	保留。始终读为 0。
9-8	ADCPDIVx	RW	0h	ADC 预分频器。此位用于对所选的 ADC 时钟源进行预分频，然后再使用 ADCDIVx 分频。 00b = 不预分频 01b = 4 倍预分频 10b = 64 倍预分频 11b = 保留
7-5	保留	R	0h	保留。始终读为 0。
4	ADCRES	RW	1h	ADC 分辨率。该位定义转换结果的分辨率。 0b = 8 位（10 个时钟周期转换时间） 1b = 10 位（12 个时钟周期转换时间）
3	ADCDF	RW	0h	ADC 数据读回格式。数据总是以二进制的无符号格式存储。 0b = 二进制无符号理论上模拟输入电压 $-V_{REF}$ 为 0000h，模拟输入电压 $+V_{REF}$ 为 03FFh。 1b = 有符号二进制（二进制补码）数，左对齐。理论上模拟输入电压 $-V_{REF}$ 为 8000h，模拟输入电压 $+V_{REF}$ 为 7FC0h。
2	ADCSR	RW	0h	ADC 采样率。该位可选择 ADC 基准缓冲器的驱动能力以获得最大采样率。ADCSR 置 1 可降低此缓冲器的电流消耗。 0b = ADC 缓冲器最高支持约 200 ksps 的采样率 1b = ADC 缓冲区最高支持约 50 ksps 的采样率
1	保留	R	0h	保留。始终读为 0。
0	保留	RW	0h	保留。必须写为 0。

13.3.4 ADCMEM0 寄存器

ADC 转换存储器寄存器

图 **13-15. ADCMEM0 寄存器**

15	14	13	12	11	10	9	8
转换结果							
r0	r0	r0	r0	r0	r0	rw	rw
7	6	5	4	3	2	1	0
转换结果							
rw	rw	rw	rw	rw	rw	rw	rw

表 **13-6. ADCMEM0 寄存器说明**

位	字段	类型	复位	说明
15-0	转换结果	RW	未定义	此数据格式在 ADCDF = 0 时使用（无符号二进制数）。10 位转换结果右对齐。位 9 是 MSB。10 位模式下 bit 15-10 为 0，8 位模式下位 bit 15-8 为 0。写入转换存储寄存器会损坏结果。

13.3.5 ADCMEM0 寄存器，二进制补码格式

ADC 转换存储寄存器，二进制补码格式

图 **13-16. ADCMEM0 寄存器**

15	14	13	12	11	10	9	8
转换结果							
rw	rw	rw	rw	rw	rw	rw	rw
7	6	5	4	3	2	1	0
转换结果							
rw	rw	r0	r0	r0	r0	r0	r0

表 **13-7. ADCMEM0 寄存器说明**

位	字段	类型	复位	说明
15-0	转换结果	RW	未定义	此数据格式在 ADCDF = 1 时使用（二进制补码）。10 位转换结果是左对齐的，二进制补码格式。位 15 是 MSB。10 位模式下 bit 5-0 为 0，8 位模式下位 bit 7-0 为 0。该数据以右对齐格式被存储并在回读期间被转换为左对齐的二进制补码格式。写入转换存储寄存器会破坏结果。

13.3.6 ADCMCTL0 寄存器

ADC 转换存储器控制寄存器

图 13-17. ADCMCTL0 寄存器

7	6	5	4	3	2	1	0
保留	ADCSREFx			ADCINCHx			
r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

仅在 ADCENC = 0 时可修改。在转换激活情况下，通过软件复位 ADCENC = 0 和更改这些字段会立即显示相关结果。

表 13-8. ADCMCTL0 寄存器说明

位	字段	类型	复位	说明
7	保留	R	0h	保留。始终读为 0。
6-4	ADCSREFx	RW	0h	选择基准转换进行期间，不建议更改这些设置。 仅在 ADCENC = 0 时可修改。在转换激活情况下，通过软件复位 ADCENC = 0 和更改这些字段会立即显示相关结果。 000b = V_{R+} = AVCC 且 V_{R-} = AVSS 001b = V_{R+} = VREF 且 V_{R-} = AVSS 010b = V_{R+} = VEREF+ 缓冲电压且 V_{R-} = AVSS 011b =V_{R+} = VEREF+ 且 V_{R-} = AVSS 100b = V_{R+} = AVCC 且 V_{R-} = VEREF- 101b = V_{R+} = VREF 且 V_{R-} = VEREF- 110b = V_{R+} = VEREF+ 缓冲电压且 V_{R-} = VEREF- 111b = V_{R+} = VEREF+ 且 V_{R-} = VEREF-
3-0	ADCINCHx	RW	0h	输入通道选择。写入这些位来选择进行单次转换的通道或进行转换序列的最高通道。在 ADCCONSEQ = 01,11 的模式下读取这些位会返回当前转换的通道。 仅在 ADCENC = 0 时可修改。在转换激活情况下，通过软件复位 ADCENC = 0 和更改这些字段会立即显示相关结果。 0000b = A0（请参见具体器件的数据表） 0001b = A1（请参见具体器件的数据表） 0010b = A2（请参见具体器件的数据表） 0011b = A3（请参见具体器件的数据表） 0100b = A4（请参见具体器件的数据表） 0101b = A5（请参见具体器件的数据表） 0110b = A6（请参见具体器件的数据表） 0111b = A7（请参见具体器件的数据表） 1000b = A8（请参见具体器件的数据表） 1001b = A9（请参见具体器件的数据表） 1010b = A10（请参见具体器件的数据表） 1011b = A11（请参见具体器件的数据表） 1100b = A12（请参见具体器件的数据表） 1101b = A13（请参见具体器件的数据表） 1110b = A14（请参见具体器件的数据表） 1111b = A15（请参见具体器件的数据表）

13.3.7 ADCHI 寄存器

ADC 窗口比较器的阈值上限寄存器

图 13-18. ADCHI 寄存器

15	14	13	12	11	10	9	8
高电平阈值							
r0	r0	r0	r0	r0	r0	rw-(1)	rw-(1)
7	6	5	4	3	2	1	0
高电平阈值							
rw-(1)	rw-(1)	rw-(1)	rw-(1)	rw-(1)	rw-(1)	rw-(1)	rw-(1)

表 13-9. ADCHI 寄存器说明

位	字段	类型	复位	说明
15-0	高电平阈值	RW	3FFh	此数据格式可在 ADCDF = 0 时使用（无符号二进制数）。10 位的阈值需要进行右对齐。位 9 是 MSB。在 10 位模式中，位 15 - 10 为 0，在 8 位模式中，位 15 - 8 为 0。

13.3.8 ADCHI 寄存器，二进制补码格式

ADC 窗口比较器的阈值上限寄存器，二进制补码格式

图 13-19. ADCHI 寄存器

15	14	13	12	11	10	9	8
高电平阈值							
rw - (0)	rw - (1)	rw-(1)	rw-(1)	rw-(1)	rw-(1)	rw-(1)	rw-(1)
7	6	5	4	3	2	1	0
高电平阈值							
rw-(1)	rw-(1)	r0	r0	r0	r0	r0	r0

表 13-10. ADCHI 寄存器说明

位	字段	类型	复位	说明
15-0	高电平阈值	RW	1FFh	此数据格式可在 ADCDF = 1 时使用（二进制补码）。10 位阈值需要左对齐。位 15 是 MSB。在 10 位模式中，位 5 - 0 为 0，在 8 位模式中，位 7 - 0 为 0。

13.3.9 ADCLO 寄存器

ADC 窗口比较器阈值下限寄存器

图 13-20. ADCLO 寄存器

15	14	13	12	11	10	9	8
低电平阈值							
r0	r0	r0	r0	r0	r0	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
低电平阈值							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

表 13-11. ADCLO 寄存器说明

位	字段	类型	复位	说明
15-0	低电平阈值	RW	0h	此数据格式在 ADCDF = 0 时使用（无符号二进制数）。10 位的阈值需要进行右对齐。位 9 是 MSB。在 10 位模式中，位 15 - 10 为 0，在 8 位模式中，位 15 - 8 为 0。

13.3.10 ADCLO 寄存器，二进制补码格式

ADC 窗口比较器阈值下限寄存器，二进制补码格式

图 13-21. ADCLO 寄存器

15	14	13	12	11	10	9	8
低电平阈值							
rw-(1)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
低电平阈值							
rw-(0)	rw-(0)	r0	r0	r0	r0	r0	r0

表 13-12. ADCLO 寄存器说明

位	字段	类型	复位	说明
15-0	低电平阈值	RW	200h	此数据格式在 ADCDF = 1 时使用（二进制补码）。如果选择二进制补码格式，则 10 位的阈值需要进行左对齐。位 15 是 MSB。10 位模式下 bit 5-0 为 0，8 位模式下位 bit 7-0 为 0。

13.3.11 ADCIE 寄存器

ADC 中断使能寄存器

图 13-22. ADCIE 寄存器

15	14	13	12	11	10	9	8
保留							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
保留		ADCTOVIE	ADCOVIE	ADCHIIE	ADCLOIE	ADCINIE	ADCIE0
r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

表 13-13. ADCIE 寄存器说明

位	字段	类型	复位	说明
15-6	保留	R	0h	保留。始终读为 0。
5	ADCTOVIE	RW	0h	ADC 转换时间溢出中断使能。 0b = 禁用转换时间溢出中断 1b = 使能转换时间溢出中断
4	ADCOVIE	RW	0h	ADCMEM0 溢出中断使能。 0b = 禁用溢出中断 1b = 使能溢出中断
3	ADCHIIE	RW	0h	窗口比较器的阈值上限中断使能。 0b = 禁用阈值上限中断 1b = 使能阈值上限中断
2	ADCLOIE	RW	0h	窗口比较器的阈值下限中断使能。 0b = 禁用阈值下限中断 1b = 使能阈值下限中断
1	ADCINIE	RW	0h	窗口比较器窗口内部中断的中断使能。 0b = 禁用窗口内部中断 1b = 使能窗口内部中断
0	ADCIE0	RW	0h	中断使能。此位可使能或禁用已完成 ADC 转换的中断请求。 0b = 禁用中断 1b = 使能中断

13.3.12 ADCIFG 寄存器

ADC 中断标志寄存器

图 13-23. ADCIFG 寄存器

15	14	13	12	11	10	9	8
保留							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
保留		ADCTOVIFG	ADCOVIFG	ADCHIIFG	ADCLOIFG	ADCINIFG	ADCIFG0
r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

表 13-14. ADCIFG 寄存器说明

位	字段	类型	复位	说明
15-6	保留	R	0h	保留。始终读为 0。
5	ADCTOVIFG	RW	0h	在实际转换完成前触发 ADC 转换时，ADCTOVIFG 置 1。 0b = 无中断挂起 1b = 中断挂起
4	ADCOVIFG	RW	0h	如果在读取上一次转换结果前对 ADCMEM0 触发器执行写入，则 ADCOVIFG 置 1。 0b = 无中断挂起 1b = 中断挂起
3	ADCHIIFG	RW	0h	在当前 ADC 转换结果大于窗口比较器阈值寄存器定义的阈值上限时，ADCHIIFG 置 1。 0b = 无中断挂起 1b = 中断挂起
2	ADCLOIFG	RW	0h	在当前 ADC 转换结果小于窗口比较器阈值寄存器定义的阈值下限时，ADCLOIFG 置 1。 0b = 无中断挂起 1b = 中断挂起
1	ADCINIFG	RW	0h	在当前 ADC 转换结果介于窗口比较器阈值寄存器定义的阈值范围内时，ADCINIFG 置 1。 0b = 无中断挂起 1b = 中断挂起
0	ADCIFG0	RW	0h	在 ADC 转换完成时 ADCIFG0 置 1。该位在读取 ADCMEM 后复位，或可使用软件实现复位。 0b = 无中断挂起 1b = 中断挂起

13.3.13 ADCIV 寄存器

ADC 中断向量寄存器

图 13-24. ADCIV 寄存器

15	14	13	12	11	10	9	8
ADCIVx							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
ADCIVx							
r0	r0	r0	r0	r-(0)	r-(0)	r-(0)	r0

表 13-15. ADCIV 寄存器说明

位	字段	类型	复位	说明
15-0	ADCIVx	R	0h	ADC 中断向量值。它生成一个值，这个值可被用作针对快速中断处理程序的地址偏移。对这个寄存器的写入将清除所有挂起的中断标志。 00h = 无中断挂起 02h = 中断源：ADCMEM0 溢出；中断标志：ADCOVIFG；中断优先级：最高 04h = 中断源：转换时间溢出；中断标志：ADCTOVIFG 06h = 中断源：ADCHI 中断标志；中断标志：ADCHIIFG 08h = 中断源：ADCLO 中断标志；中断标志：ADCLOIFG 0Ah = 中断源：ADCIN 中断标志；中断标志：ADCINIFG 0Ch = 中断源：ADC 存储器中断标志；中断标志：ADCIFG0；中断优先级：最低

13.3.14 MSP430FR413x SYSCFG2 寄存器（绝对地址 = 0164h）[复位 = 0000h]

系统配置寄存器 2。在 MSP430FR413x 器件中，通过系统配置寄存器 2 控制 ADC 的引脚。相关寄存器说明，请参见 SYS 一章.

图 13-25. SYSCFG2 寄存器

15	14	13	12	11	10	9	8
保留			LCDPCTL	保留	保留	ADCPCTL9	ADCPCTL8
r0	r0	r0	rw-0	r0	r0	rw - 0	rw - 0
7	6	5	4	3	2	1	0
ADCPCTL7	ADCPCTL6	ADCPCTL5	ADCPCTL4	ADCPCTL3	ADCPCTL2	ADCPCTL1	ADCPCTL0
rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0

表 13-16. SYSCFG2 寄存器说明

位	字段	类型	复位	说明
15-13	保留	R	0h	保留。始终读为。
12	LCDPCTL	RW	0h	LCD 电源引脚 (LCDCAP0、LCDCAP1、R13、R23、R33) 控制。 0b = 禁用 LCD 电源引脚 1b = 使能 LCD 电源引脚
11-10	保留	R	0h	保留。始终读为 0。
9	ADCPCTL9	RW	0h	ADC 输入 A9 引脚选择 0b = 禁用 ADC 输入 A9 1b = 使能 ADC 输入 A9
8	ADCPCTL8	RW	0h	ADC 输入 A8 引脚选择 0b = 禁用 ADC 输入 A8 1b = 使能 ADC 输入 A8
7	ADCPCTL7	RW	0h	ADC 输入 A7 引脚选择 0b = 禁用 ADC 输入 A7 1b = 使能 ADC 输入 A7
6	ADCPCTL6	RW	0h	ADC 输入 A6 引脚选择 0b = 禁用 ADC 输入 A6 1b = 使能 ADC 输入 A6
5	ADCPCTL5	RW	0h	ADC 输入 A5 引脚选择 0b = 禁用 ADC 输入 A5 1b = 使能 ADC 输入 A5
4	ADCPCTL4	RW	0h	ADC 输入 A4 引脚选择 0b = 禁用 ADC 输入 A4 1b = 使能 ADC 输入 A4
3	ADCPCTL3	RW	0h	ADC 输入 A3 引脚选择 0b = 禁用 ADC 输入 A3 1b = 使能 ADC 输入 A3
2	ADCPCTL2	RW	0h	ADC 输入 A2 引脚选择 0b = 禁用 ADC 输入 A2 1b = 使能 ADC 输入 A2
1	ADCPCTL1	RW	0h	ADC 输入 A1 引脚选择 0b = 禁用 ADC 输入 A1 1b = 使能 ADC 输入 A1
0	ADCPCTL0	RW	0h	ADC 输入 A0 引脚选择 0b = 禁用 ADC 输入 A0 1b = 使能 ADC 输入 A0

LCD_E 控制器

LCD_E 控制器用于驱动静态 LCD 以及 2 路复用到 8 路复用 LCD。本章将介绍 LCD_E 控制器。表 14-1 列出了 LCD_B、LCD_C 和 LCD_E 的区别。

Topic **Page**

14.1 LCD_E 简介

LCD_E 控制器通过自动创建交流段和公共电压信号的方式直接驱动 LCD 显示。LCD_E 控制器可支持静态和 2 路复用到 8 路复用的 LCD 玻璃。

LCD_E 控制器的特性包括：

- 显示存储器
- 支持 LPM3.5
- 可配置 SEG 和 COM 引脚
- 自动生成信号
- 可配置的帧频率
- 静态，和 2 线至 4 线多路复用的 LCD 的独立闪烁存储的单个段的闪烁
- 5 线至 8 线多路复用 LCD 的完整显示的闪烁
- 稳压电荷泵最高可达 3.44V（典型值）
- 由软件控制的对比度
- 支持以下类型的 LCD
  - 静态
  - 2 路复用，1/3 偏置
  - 3 路复用，1/3 偏置
  - 4 路复用，1/3 偏置
  - 5 路复用，1/3 偏置
  - 6 路复用，1/3 偏置
  - 7 路复用，1/3 偏置
  - 8 路复用，1/3 偏置

表 14-1 列出了 LCD_B、LCD_C 和 LCD_E 之间的差别。

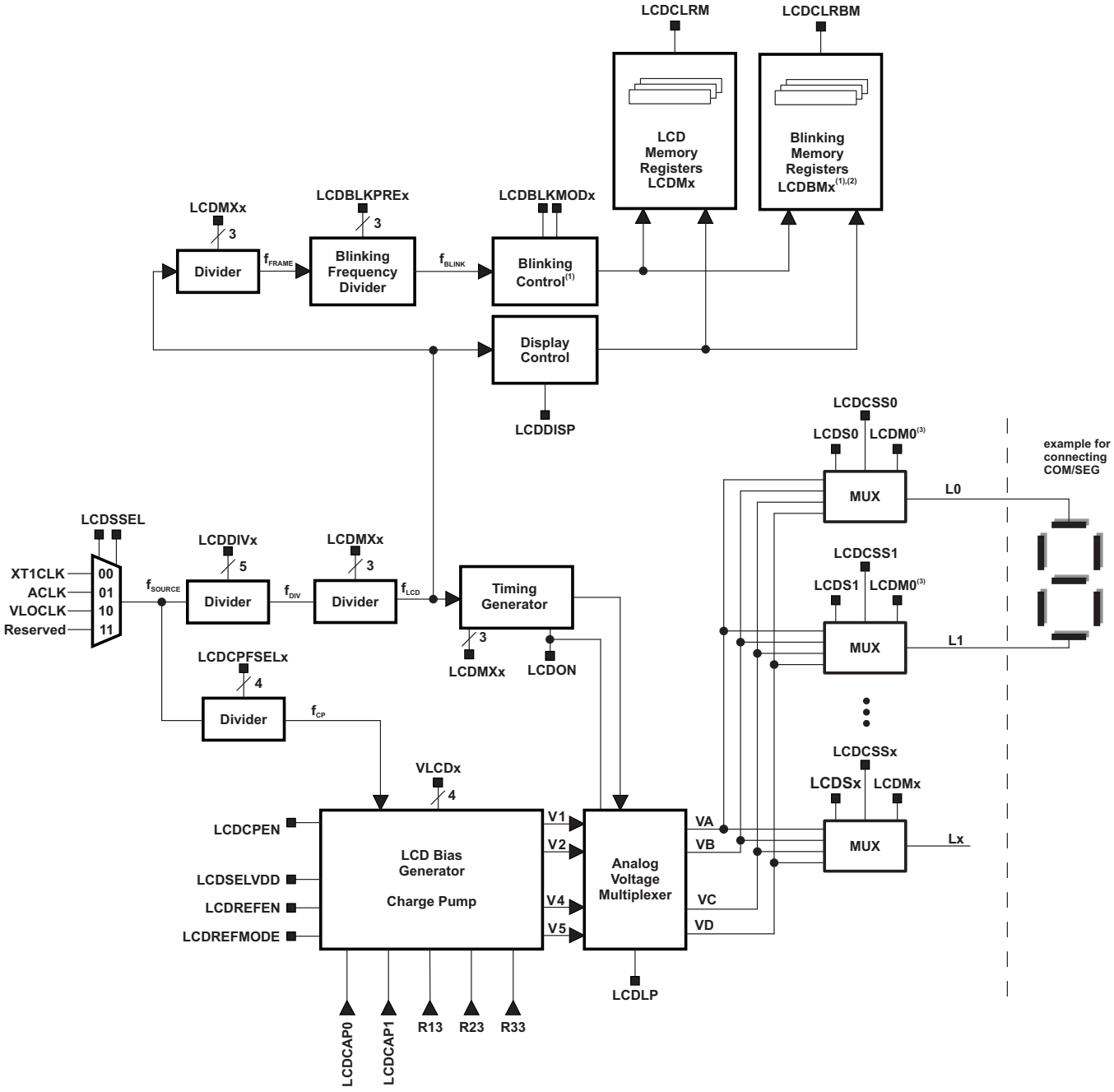
表 14-1. LCD_B、LCD_C 和 LCD_E 之间的差别

特性	LCD_B	LCD_C	LCD_E
支持的 LCD 的类型	静态，2，3，4 线多路复用	静态，2，3，4，5，6，7，8 线多路复用	静态，2、3、4 路复用 5、6、7、8 路复用（特定于器件）
LCD 偏置模式	1/2 偏置和 1/3 偏置	1/2 偏置和 1/3 偏置	1/3 偏置
LCD 闪烁存储器	是	是	特定于器件
SEG/COM mux	固定的 COM	固定的 COM	每个 LCD 驱动引脚
外部引脚	R03、R13、R23、R33	R03、R13、R23、R33	R13、R23、R33、LCDCAP0、LCDCAP1
LPM3.5	不支持	不支持	受支持
最大 VLCDx 设置	001111b	001111b	001111b
最大 LCD 电压（$V_{LCD,\ 典型值}$）	3.44V	3.44V	3.44V
LCD 引脚数	最多 4 x 46	最多 4 x 50 或 8 x 46	最多 4 x 60 或 8 x 56

图 14-1 显示了 LCD 控制器框图。

注: 最大 **LCD** 段控制

段线和可用存储寄存器的最大数量因器件不同而不同。有关可用的段引脚和支持的段的最大数量的信息，请参阅具体器件的数据表。



(1) device specific
(2) only static, 2- to 4-mux)
(3) used LCDMx depends on selected MUX mode (LCDMXx)

图 **14-1. LCD** 控制器框图

14.2 LCD_E 操作

LCD 控制器由用户软件配置。LCD 控制器的建立和运行将在下面的部分进行讨论。

14.2.1 LCD 的存储

LCD 的存储结构因模式不同而有轻微的差别。每个存储器位都对应一个 LCD 段、LCD 公共或不使用，具体取决于模式。要接通 LCD 段，需将其对应的存储器位置 1。此外，还可以使用从 LCDM0W、LCDM2W 等开始的偶数地址按字访问存储器。将 LCDCLRM 位置 1 会在下一帧边界清零所有 LCD 显示存储寄存器。寄存器被清零后它会自动复位。

14.2.1.1 静态和 2 线至 4 线多路复用模式

对于静态和 2 线至 4 线多路复用模式，LCD 存储的一个字节包含两个段线的信息。

在静态模式和 2 路到 4 路复用模式下，最大可用 LCD 段数如下：

- 静态：最多 63 个段（一条 COM 线）
- 2 路复用：最多 124 个段（两条 COM 线）
- 3 路复用：最多 183 个段（三条 COM 线）
- 4 路复用：最多 240 个段（四条 COM 线）

图 14-2 显示了 4 路复用模式下使用 240 个段的 LCD 存储器映射示例。

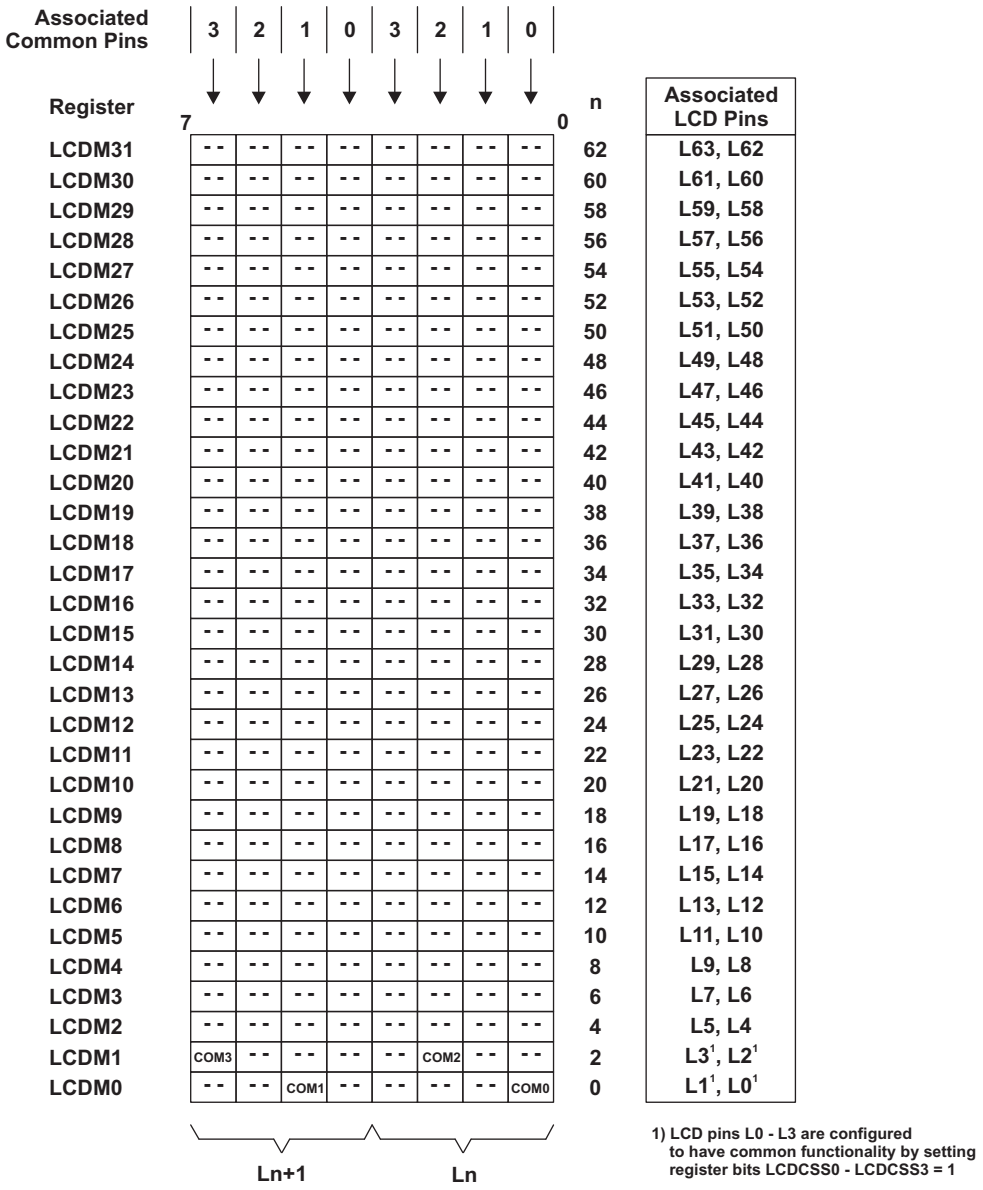


图 **14-2.** 静态到 **4** 路复用模式的 **LCD** 存储器 **- 4** 路复用模式下使用 **240** 个段的示例

14.2.1.2 5 线至 8 线多路复用模式

对于 5 线至 8 线多路复用模式，LCD 存储的一个字节包含一个段线的信息。

在 5 路到 8 路复用模式下，最大可用 LCD 段数如下：

- 5 路复用：最多 295 个段（五条 COM 线）
- 6 路复用：最多 348 个段（六条 COM 线）
- 7 路复用：最多 399 个段（七条 COM 线）
- 8 路复用：最多 448 个段（八条 COM 线）

图 14-3 显示了 8 路复用模式下使用 96 个段的 LCD 存储器映射示例。

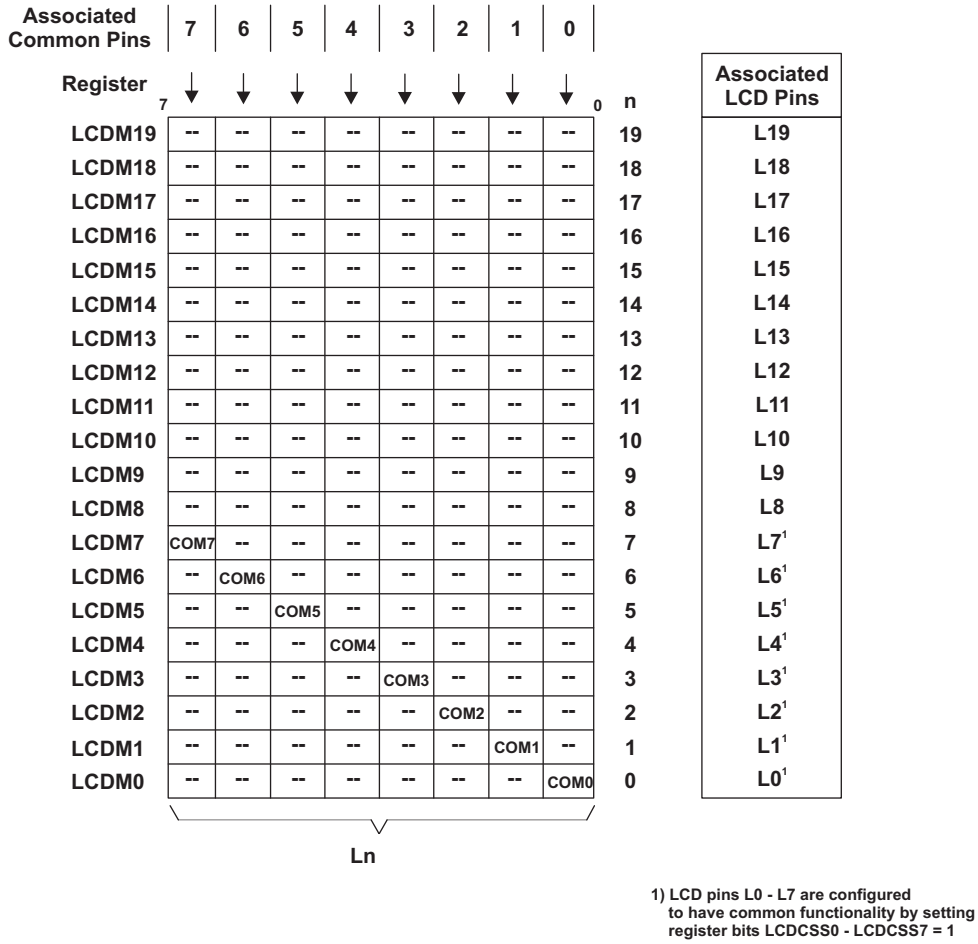


图 **14-3. 5** 路到 **8** 路复用模式的 **LCD** 存储器 **- 8** 路复用模式下使用 **96** 个段的示例

14.2.2 端口引脚配置为 LCD 输出

LCD 段和公共功能与数字 I/O 功能复用。这些引脚可用作数字 I/O 或 LCD 功能。与数字 I/O 复用时，可使用 LCDPCTLx 寄存器中的 LCDSx 位选择 LCD 段/公共功能。将 LCDSx 位置 1 可为每个引脚选择 LCD 功能。当 LCDSx = 0 时，一个多路复用引脚会被设置成数字 I/O 的功能。当 LCDSx = 1 时，一个多路复用引脚会被选作 LCD 的功能。有关控制引脚功能的详细信息，请参见具体器件数据表的端口原理图章节。

注: **LCDSx 位不会影响专用 LCD 段/公共引脚**

LCDSx 位仅会影响 LCD 段/公共功能与数字 I/O 功能复用的引脚。专用 LCD 段/公共引脚不受 LCDSx 位的影响。

14.2.3 LCD 引脚配置为 COM 或 SEG

为了简化电路板布局以及段和公共线的布线，可将每个 LCD 引脚定义为 LCD 段 (SEG) 或公共线 (COM)。LCDCSSx 位定义了如何解读 LCDMx 寄存器的内容。如果将 LCDCSSx 设为 0，LCD 引脚 Lx 会用作 LCD 段。如果将 LCDCSSx 设为 1，LCDMx 的内容会定义在相应的 LCD 引脚 Lx 上使用哪条公共线（COM0 到 COM7）。

后续章节将介绍此功能的使用方法。

14.2.3.1 将 LCD 引脚定义为段

静态、2、3、4 路复用模式

在静态、2、3、4 路复用模式下，LCDMx 寄存器用于存储两个段引脚。例如，LCDM1 包含 L3 和 L2（请参见14.2.1 节）。要将 LCD 引脚定义为 LCD 段，必须将 LCDCSSELx 寄存器中的相应位设为 0（默认值）。随后才可以使用 LCDMx 寄存器使能或禁用 LCD 段。例如，要将 LCD 引脚 L14 定义为 LCD 段，需将 LCDCSSEL0 寄存器中的 LCDCSS14 设为 0。

5、6、7 和 8 路复用模式

在5、6、7 和 8 路复用模式下，每个 LCDMx 寄存器都用于存储一个段引脚。要将 LCD 引脚定义为 LCD 段，必须将 LCDCSSELx 寄存器中的相应位设为 0（默认值）。随后才可以使用 LCDMx 寄存器使能或禁用 LCD 段。例如，LCDM7 用于存储 L7，LCDM29 用于存储 L29，以此类推。

注: 要确定某款器件是否支持 5、6、7 或 8 路复用模式，请参见具体器件的数据表。

14.2.3.2 将 LCD 引脚定义为公共线

注: 每个 LCD 引脚只能选择一个公共 (COMx) 引脚。向一个 LCD 引脚分配两个或多个公共功能可能导致意外操作。

要将 LCD 引脚定义为具有 LCD 公共功能，必须将 LCDCSSELx 寄存器中的相应位设为 1。随后才可以使用 LCDMx 寄存器将关联的 LCD 引脚配置为具有 COMx 功能。

LCDMx 的设置有所不同，具体取决于是使用静态到 4 路复用模式还是 5 路复用到 8 路复用模式。后续章节将介绍具体差别。

14.2.3.2.1 静态、2、3 或 4 路复用模式下的 COM 分配

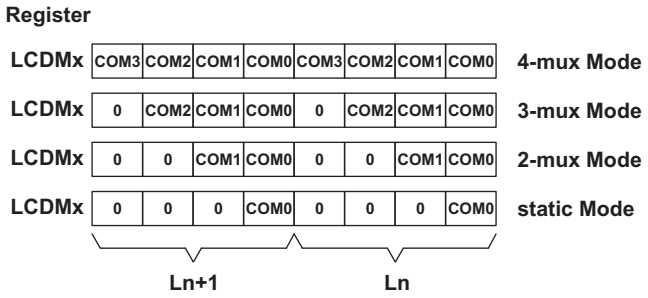
在静态、2、3 或 4 路复用模式下，每个 LCDMx 均用于控制两个 LCD 引脚的公共功能。与14.2.1 节中介绍的段功能类似，LCDMx 的低半字节用于控制编号为偶数的 LCD 引脚（L0、L2...）。编号为奇数的 LCD 引脚（L1、L3..）由 LCDMx 的高半字节控制。每个 LCD 引脚选择两个或多个 COM 引脚可能导致 LCD 意外操作，因此必须加以避免。

在静态模式下，仅可使用 COM0。

在 2 路复用模式下，可选择 COM0 和 COM1。

在 3 路复用模式下，可选择 COM0、COM1 和 COM2。

在 4 路复用模式下，可选择 COM0、COM1、COM2 和 COM3。



SEG_{MAX} = Number of Segment Pins

n = 0 ... $\frac{SEG_{MAX} - 1}{2}$

x = n

图 **14-4.** 静态、**2**、**3** 或 **4** 路复用模式下的 **LCDMx**

示例：

要将 LCD 引脚 L4 用作 COM2，请进行以下配置：

```
LCDPCTL0 = BIT4;   // configure I/O pad as LCD pin
LCDCSSEL0 = BIT4;  // configure LCD pin L4 as COM
LCDM2 = BIT2;      // define L4 as COM2
```

要将 LCD 引脚 L23 用作 COM0，请进行以下配置：

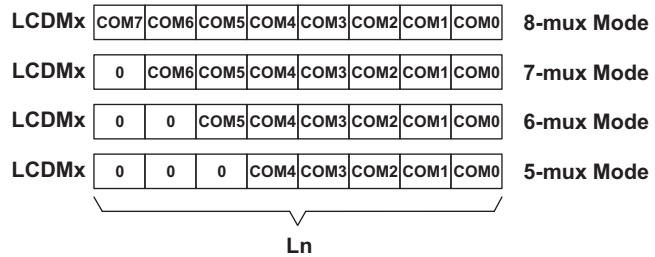
```
LCDPCTL1 = BIT7;   // configure I/O pad as LCD pin
LCDCSSEL1 = BIT7;  // configure LCD pin L23 as COM
LCDM11 = BIT4;     // define L23 as COM0
```

14.2.3.2.2 5、6、7 或 8 路复用模式下的 COM 分配

在 5、6、7 和 8 路复用模式下，每个 LCDMx 均用于控制一个 LCD 引脚的公共功能。

要将 LCD 引脚定义为 LCD 公共功能，必须将 LCDCSSELx 寄存器中的相应位设为 1。在 5、6、7 和 8 路复用模式下，每个 LCDMx 寄存器均控制一个 LCD 引脚的公共功能。每个 LCD 引脚选择两个或多个 COM 引脚可能导致 LCD 意外操作，因此必须加以避免。

注： 要确定某款器件是否支持 5、6、7 或 8 路复用模式，请参见具体器件的数据表。



SEG_{MAX} = Number of Segment Pins

n = 0 ... SEG_{MAX} - 1

x = n

图 **14-5. 5**、**6**、**7** 或 **8** 路复用模式下的 **LCDMx**

示例：

要将 LCD 引脚 L4 用作 COM6，请进行以下配置：

```
LCDPCTL0 |= BIT4;   // configure I/O pad as LCD pin
LCDCSSEL0 |= BIT4;   // configure LCD pin L4 as COM
LCDM4 = BIT6;       // define L4 as COM6
```

要将 LCD 引脚 L23 用作 COM5，请进行以下配置：

```
LCDPCTL1 = BIT7;    // configure I/O pad as LCD pin
LCDCSSEL1 = BIT7;    // configure LCD pin L23 as COM
LCDM23 = BIT5;      // define L23 as COM5
```

14.2.4 LCD 时序生成

LCD_E 控制器使用集成时钟分频器发出的 f_{LCD} 信号为公共和段线生成时序。LCDSSEL 位会将源频率 f_{SOURCE} 设为 ACLK（30kHz 到 40kHz）、XT1CLK (32.768kHz) 或 VLOCLK (≈ 10kHz)。f_{LCD} 频率通过 LCDDIVx 和 LCDMXx 位进行选择，具体视所选多路复用模式而定。表 14-2 列出了多路复用模式与分频器的对应关系。

表 **14-2.** 分频器与多路复用模式的对应关系

多路复用模式	多路复用分频器
1（静态）	64
2	32
3	16
4	16
5	12
6	8
7	8
8	8

获得的 f_{LCD} 频率的计算方法如下：

$$f_{LCD} = \frac{f_{SOURCE}}{(LCDDIVx + 1) \times MUXDIVDER}$$

示例 1：

f_{LCD} 频率是否合适取决于 LCD 对帧频和 LCD 多路复用率的要求。为了避免 LCD 上出现重影效应，f_{LCD} 应大约在 30Hz 到 60Hz 范围内。其计算方法如下：

$f_{LCD} = 2 \times mux \times f_{FRAME}$

例如，计算帧频为 25Hz 到 80Hz 的 3 路复用 LCD 的 f_{LCD}:

f_{FRAME}（查阅 LCD 数据表）= 25Hz 到 80Hz

$f_{LCD} = 2 \times 3 \times f_{FRAME}$

f_{LCD}（最小值）= 150Hz

f_{LCD}（最大值）= 480Hz

如果 f_{SOURCE} = 32768Hz，LCDDIVx = 01101，LCDMXx = 010：

f_{LCD}= 32768Hz / ((13+) × 161) = 32768Hz / 224 = 146Hz

如果 LCDDIVx = 00100，LCDMXx = 010：

f_{LCD} = 32768Hz / ((4+1) × 16) = 32768Hz / 56 = 409Hz

最低频率具有最低的电流消耗。最高频率具有最少的闪烁。

示例 2：

表 14-3 列出了可用的 f_{LCD}、f_{FRAME} 和 f_{BLINK} 频率（针对给定的 f_{SOURCE} = 32.768kHz），具体视所选多路复用模式而定。

表 14-3. 可用 LCD 频率示例

f_{SOURCE} (Hz)	多路复用模式	LCDDIVx(1)	f_{DIV} (Hz)	f_{LCD} (Hz)	f_{FRAME} (Hz)	f_{BLINK} (Hz)
32768	静态	4-16	8192 ... 2048	(8192 ... 2048) / 64 = (128 ... 32)	(128 ... 32) / 2 / 1 = (64 ... 16)	f_{LCD} / ((LCDMx + 1) × $2^{(LCDBLKPREx + 2)}$)
32768	2	4-16	8192 ... 2048	(8192 ... 2048) / 32 = (256 ... 64)	(256 ... 64) / 2 / 2 = (64 ... 16)	f_{LCD} / ((LCDMx + 1) × $2^{(LCDBLKPREx + 2)}$)
32768	3	4-16	8192 ... 2048	(8192 ... 2048) / 16 = (512 ... 128)	(512 ... 128) / 2 / 3 = (85 ... 21)	f_{LCD} / ((LCDMx + 1) × 2 ^ (LCDBLKPREx + 2))
32768	4	4-16	8192 ... 2048	(8192 ... 2048) / 16 = (512 ... 128)	(512 ... 128) / 2 / 4 = (64 ... 16)	f_{LCD} / ((LCDMx + 1) × $2^{(LCDBLKPREx + 2)}$)
32768	5	4-16	8192 ... 2048	(8192 ... 2048) / 12 = (683 ... 171)	(682 ... 172) / 2 / 5 = (68 ... 17)	f_{LCD} / ((LCDMx + 1) × $2^{(LCDBLKPREx + 2)}$)
32768	6	4-16	8192 ... 2048	(8192 ... 2048) / 8 = (1024 ... 256)	(1024 ... 256) / 2 / 6 = (85 ... 21)	f_{LCD} / ((LCDMx + 1) × $2^{(LCDBLKPREx + 2)}$)
32768	7	4-16	8192 ... 2048	(8192 ... 2048) / 8 = (1024 ... 256)	(1024 ... 256) / 2 / 7 = (73 ... 18)	f_{LCD} / ((LCDMx + 1) × $2^{(LCDBLKPREx + 2)}$)
32768	8	4-16	8192 ... 2048	(8192 ... 2048) / 8 = (1024 ... 256)	(1024 ... 256) / 2 / 8 = (64 ... 16)	f_{LCD} / ((LCDMx + 1) × $2^{(LCDBLKPREx + 2)}$)

(1) 不建议使 LCDDIVx < 4，因为这样会导致 f_{LCD}、f_{FRAME} 和 f_{BLINK} 的频率较高

14.2.5 消隐 LCD

LCD 控制器允许消隐整个 LCD。LCDSON 位于每个段的内存位进行与操作。当 LCDSON = 1 时，按照位的值将每个段打开或关闭。当 LCDSON= 0 时，每个 LCD 段是关闭的。

14.2.6 LCD 的闪烁

LCD 控制器还支持闪烁。在静态和 2 线至 4 线多路复用模式中，闪烁模式 LCDBLKMODx = 01 允许单独段的闪烁；令 LCDBLKMODx = 10，所有段都闪烁；令 LCDBLKMODx = 00，闪烁被禁用。 在 5 线多路复用模式和上述模式中，只有允许所有段可闪烁的闪烁模式 LCDBLKMODx = 10 是可用的；如果选择了其他模式，闪烁会被禁用。

14.2.6.1 闪烁存储

在静态和 2 线至 4 线多路复用模式中，为了选择闪烁段，一个独立的闪烁存储被执行。为了使能单个段闪烁，需将闪烁内存 LCDBMx 寄存器中的相应位置 1。该存储使用相同的结构作为 LCD 存储，如图 14-2在中所示。根据多路复用模式 LCDMXx，每个存储位对应于一个 LCD 段或不被使用。若要为一个 LCD 段使能闪烁，则需将其相应的存储位置 1。

此外，还可以使用从 LCDBM0W、LCDBM2W 等开始的偶数地址按字访问闪烁存储器。

设置 LCDCLRBM 位会清零所有下一帧边界处的闪烁存储寄存器。寄存器被清零后它会自动复位。

14.2.6.2 闪烁模式下的 COM 配置

如果将 LCD 段配置为闪烁，必须特别小心。如节 14.2.3.2中所述，显示存储器 LCDMx 的一部分会用于 COM 配置。具体视闪烁模式 LCDBLKMODx、显示存储器 LCDMx 和闪烁存储器 LCDBMx 的配置方式而定。有关详细信息，请参见表 14-4。

表 **14-4.** 闪烁模式下的 **COM** 配置概述

闪烁模式 **LCDBLKMOXx**	说明
00b	禁用闪烁，用户可通过设置 LCDMEMCTL 寄存器中的 LCDDISP 位选择显示哪个存储器 **LCDMx**：COM 相关的配置位应进行相应设置 **LCDBMx**：COM 相关的配置位应根据 LCDMx 配置进行相应设置
01b	各个段按照闪烁存储寄存器 LCDBMx 中的使能情况闪烁 **LCDMx**：COM 相关的存储器位应进行相应设置 **LCDBMx**：COM 相关的存储器位应设为 0
10b	所有段闪烁 **LCDMx**：COM 相关的存储器位应进行相应设置 **LCDBMx**：该闪烁模式下不使用该存储器，不需要对 LCDBMx 进行编程
11b	在 LCDMx 和 LCDBMx 存储寄存器中存储的显示内容之间切换 **LCDMx**：COM 相关的存储器位应进行相应设置 **LCDBMx**：COM 相关的存储器位应根据 LCDMx 配置进行相应设置

“LCDBMx 必须根据 LCDMx 进行相应配置”是指二者必须使用相同的存储器编号“x”。例如，如果 LCDM2 = 02h（LCD 引脚 L2 = COM1），那么 LCDBM2 也必须编程为 02h。

示例：

LCD 配置为使用 4 路复用模式，20 个 LCD 引脚，4 个引脚配置为公共功能，16 个引脚配置为段

L0 = COM0，L1 = COM1，L2 = COM2，L3 = COM3；L4 ... L19 = SEG0 ... SEG19

必须进行以下配置：

```
LCDPCTL0 = 0xFFFF;  // configure I/O pad of L0 to L15 as LCD pin
LCDPCTL1 = 0x000F;  // configure I/O pad of L16 to L19 as LCD pin
LCDCSSEL0 = 0x000F; // configure LCD pin L0-L3 as common
```

图 14-6 显示了如何在不同闪烁模式期间配置显示存储器 LCDMx 和闪烁存储器 LCDBMx。

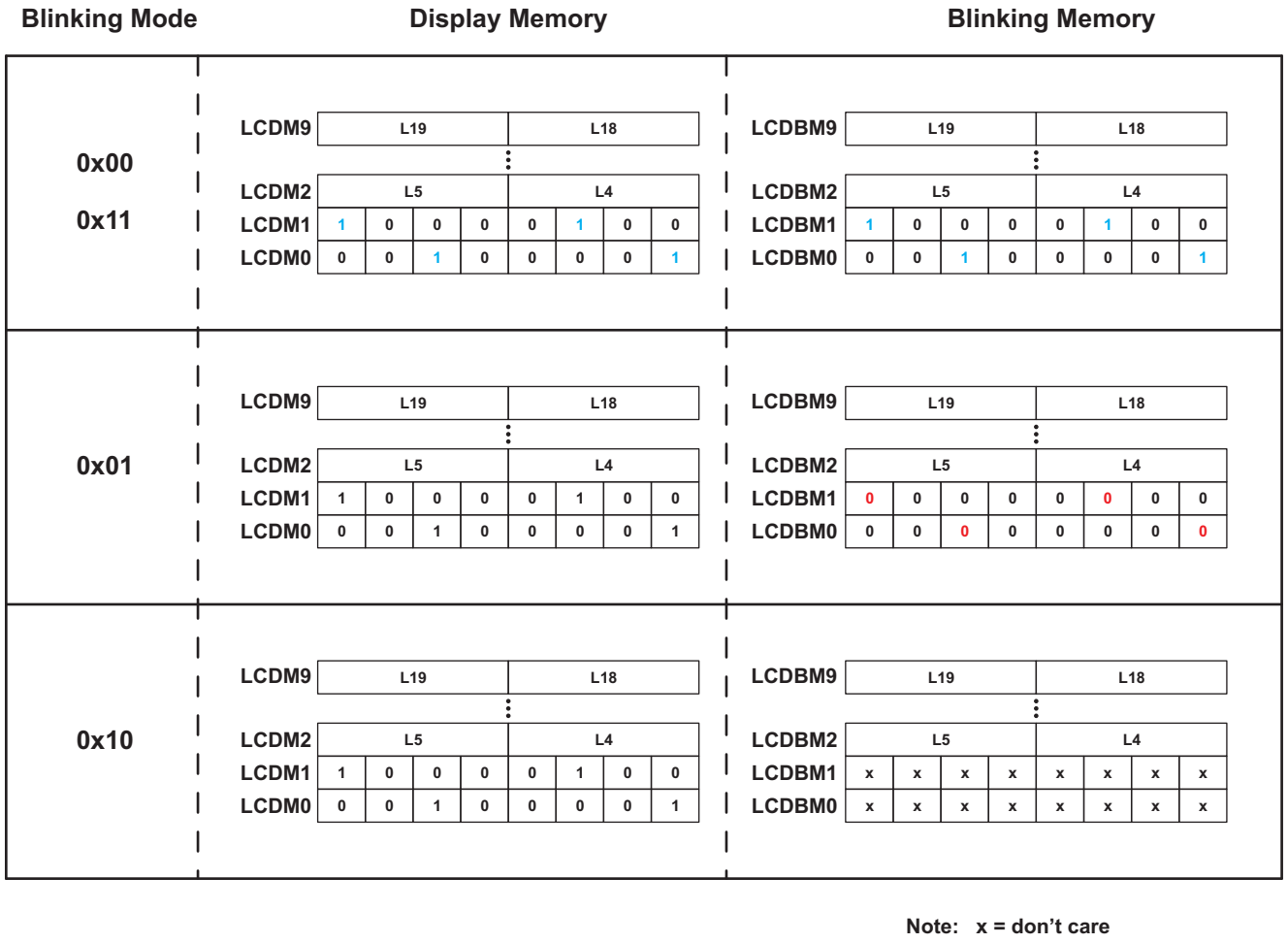


图 **14-6.** 不同闪烁模式下的 **LCDMx** 和 **LCDBMx** 配置示例

14.2.6.3 闪烁频率

闪烁频率 f_{BLINK} 通过 CDBLKDIVx 和 LCDMXx 位进行选择，因此取决于所选多路复用模式。由此所得的 f_{BLINK}频率的计算方法如下：

$$f_{BLINK} = \frac{f_{LCD}}{(LCDMXx + 1) \times 2^{LCDBLKPREx+2}}$$

当 LCDBLKMODx = 00 时，生成闪烁频率 f_{BLINK}的分频器被复位。在一个闪烁模式 LCDBLKMODx = 01 或 10 被选择后，被使能的段或全部分在下一帧边界处段会被消隐并保持关闭状态半个 BLKCLK 周期。之后，在于帧边界处它们再次被消隐前，在下一帧边界处它们会生效并保持开启状态为另个半 BLKCLK 周期。

注： **闪烁频率的限制**

闪烁频率必须小于帧频 f_{FRAME}。

当 LCDBLKMODx = 00 时，只应更改闪烁频率。

14.2.6.4 双显示存储

在静态模式到 4 路复用模式下，如果未选择闪烁模式（LCDBLKMODx = 01 或 10），闪烁存储器 LCDBMx 也可用作第二显示存储器。

如果 LCDBLKMODx = 00，LCDDISP 位可用于手动选择要显示的存储器。如果 LCDDISP = 0，说明选择了 LCD 存储器 LCDMx，如果 LCDDISP = 1，说明闪烁存储器 LCDBMx 被选作显示存储器。存储之间的切换与帧边界是同步的。

通过令 LCDBLKMODx = 11，LCD 控制器会在用分频器产生闪烁频率的存储之间自动切换。LCDDISP 位可用作静态位，指示所选存储器。在 LCDBLKMODx = 11 被选定后，在第一个半个 BLKCLK 期间要显示的存储是 LCD 存储。在第二个半个 BLKCLK 期间，闪烁存储被用作显示存储。存储之间的切换与帧边界是同步的。

14.2.7 LCD 电压和偏置生成

LCD_E 模块允许为峰值输出波形电压 V1 以及分数 LCD 偏置电压 V2、V4 和 V5 选择电压源。V_{LCD} 可由 V_{CC} 或内部电荷泵提供，也可从外部提供。

14.2.7.1 LCD 电压选择

当 LCDSELVDD = 1 且 LCDREFEN = 0 时，V_{LCD} 由 V_{CC} 提供。当 LCDSELVDD = 0 且 LCDCPEN = 1 时，V_{LCD} 由内部电荷泵提供。内部电荷泵由 V_{EXT} 或 V_{DD} 通过 R33 提供，或者由外部基准电压 $V_{REF,EXT}$ 或内部基准电压通过 R13 提供。VLCDx 位提供可通过软件选择的 LCD 电压，电压大小为 2.6V 到 3.5V（典型值），与 V_{DD} 无关。具体规范请参见具体器件的数据表。

如果使用内部电荷泵，则必须在 LCDCAP0 和 LCDCAP1 引脚之间连接一个不小于 100nF 的电容。为了降低系统噪声，可通过设置 LCDCPEN = 0，VLCDx > 0 来暂时禁用电荷泵。通过将 LCDVCTL 寄存器中的相应位置 1 可在某些时段自动禁用电荷泵。在这种情况下，在外部电容器上的被用作 LCD 电压直到电荷被重新使能。

注： **内部电荷泵所需的电容器**

如果使能内部电荷泵，则必须在 LCDCAP0 和 LCDCAP1 引脚之间连接一个不小于 100nF 的电容。

14.2.7.2 LCD 偏置生成

分数 LCD 偏置电压 V2 和 V4 可不依赖于 V_{LCD} 源在内部或外部生成。V5 始终接地。图 14-7 给出了 LCD_E 静态和 2 路复用到 8 路复用模式的偏置生成框图。

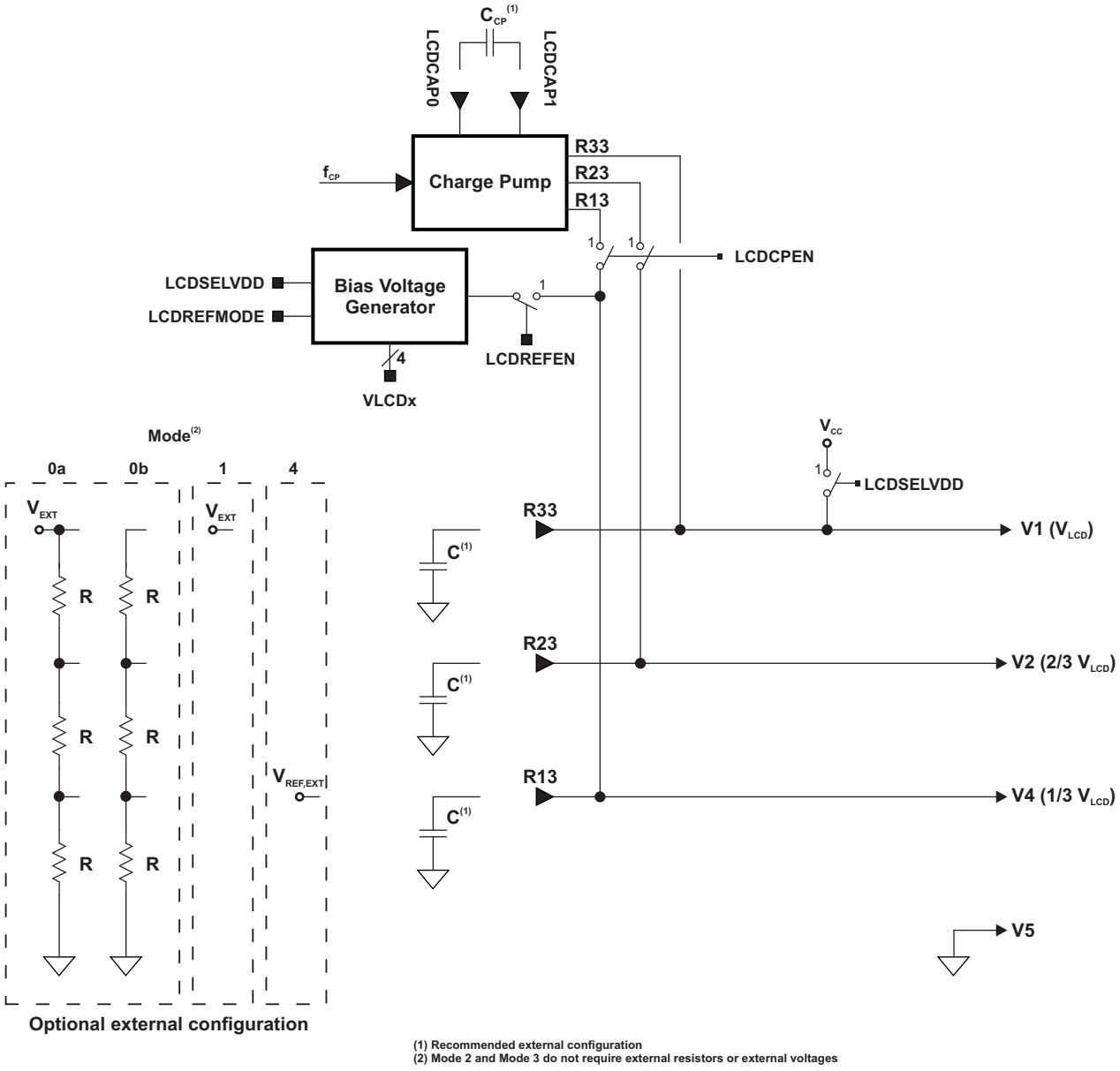


图 **14-7.** 偏置生成

偏置电压 V1、V2 和 V4 在引脚 R33、R23 和 R13 上提供。要从外部获得偏置电压 V1、V2 和 V4，需要使用等权电阻分压器，电阻大小从几千欧姆到一百万欧姆，具体视显示大小而定。 如果使用内部电荷泵，则偏置电压 V1、V2 和 V4 可由多种电压源提供。可以将外部电压 V_{EXT} 或内部电压 V_{DD} 连接至 R33，以生成 V2 和 V4。请参见节 14.2.8.1（模式 1 和模式 2）。第三种可行方法是由内部或外部基准电压通过 R13 提供。请参见节 14.2.8.2（模式 3）和节 14.2.8.3（模式 4）。

14.2.7.3 LCD 对比度控制

带有选定模式的输出波形的峰值电压和偏置决定了 LCD 的对比度和对比度比。在软件中，可通过调整由使用 VLCDx 设置的集成电荷泵生成的 LCD 电压来控制 LCD 对比度。

对比度取决于使用的 LCD 显示器。表 14-5给出了偏压的配置，该配置与被开启的（$V_{RMS,\ 开启}$）和关断（$V_{RMS,\ 关闭}$）的段电压 RMS 一起被应用到不同模式中，它们是 V_{LCD}的功能。也给出了由此生成的开启和关闭状态之间的对比度。

表 14-5. LCD 电压和偏压特性

模式	偏压配置	LCDMXx	串行通讯 (COM) 线	电压电平	$V_{RMS,\ 关闭}$/ V_{LCD}	$V_{RMS,\ 开启}$/ V_{LCD}	对比度比 $V_{RMS,\ 开启}$/ $V_{RMS,\ 关闭}$
静态	静态	0000	1	V1，V5	0	1	1/0
2 线多路复用	1/3	0001	2	V1，V2，V4，V5	0.333	0.745	2.236
3 线多路复用	1/3	0010	3	V1，V2，V4，V5	0.333	0.638	1.915
4 线多路复用	1/3	0011	4	V1，V2，V4，V5	0.333	0.577	1.732
5 线多路复用	1/3	0100	5	V1，V2，V4，V5	0.333	0.537	1.612
6 线多路复用	1/3	0101	6	V1，V2，V4，V5	0.333	0.509	1.528
7 线多路复用	1/3	0110	7	V1，V2，V4，V5	0.333	0.488	1.464
8 线多路复用	1/3	0111	8	V1，V2，V4，V5	0.333	0.471	1.414

一个确定所需 V_{LCD}的典型方法是使 $V_{RMS,\ OFF}$与一个已定义的 LCD 阈值电压相等，LCD 通常在此时的显示为大约 10% 对比度 ($V_{th,\ 10\%}$)：$V_{RMS,\ OFF} = V_{th,10\%}$。使用表中提供的 $V_{RMS,\ 关闭}/V_{LCD}$的值会致使 $V_{LCD} = V_{th,\ 10\%}/(V_{RMS,\ 关闭}/V_{LCD})$。在静态模式中，一个合适的选择是 V_{LCD}大于或等于 3 倍的 $V_{th,\ 10\%}$。

14.2.8 LCD 工作模式

本节将介绍 LCD 的不同工作模式。

14.2.8.1 内部电荷泵使能，内部 V_{REF} 禁用（模式 1、模式 2）

图 14-8 对模式 1 进行了描述。LCD 电压由连接至引脚 R33 的外部电压 V_{EXT} 提供。内部电荷泵用于生成 LCD 电压 V1、V2、V4 和 V5。可通过更改 V_{EXT} 来调整对比度。

```
LCDSELVDD  = 0;       // Pin R33 is connected to external supply voltage
LCDCPEN    = 1;       // internal charge pump enabled
LCDREFEN   = 0;       // internal reference voltage at R13 is disabled
LCDCPFSELx = 0b1111;  // charge pump frequency select, slowest value
VLCDx      = 0b0000;  // not used, set to reset value
LCDON      = 1;       // enable LCD
```

图 14-9 显示的是模式 2。R33 连接至内部电源电压 V_{DD}。内部电荷泵用于生成 LCD 电压 V1、V2、V4 和 V5。可通过在 1.8V 与 3.6V 之间更改 V_{DD} 来调整对比度。

```
LCDSELVDD  = 1;       // Pin R33 is connected to internal supply voltage
LCDCPEN    = 1;       // internal charge pump enabled
LCDREFEN   = 0;       // internal reference voltage at R13 is disabled
LCDCPFSELx = 0b1111;  // charge pump frequency select, slowest value
VLCDx      = 0b0000;  // not used, set to reset value
LCDON      = 1;       // enable LCD
```

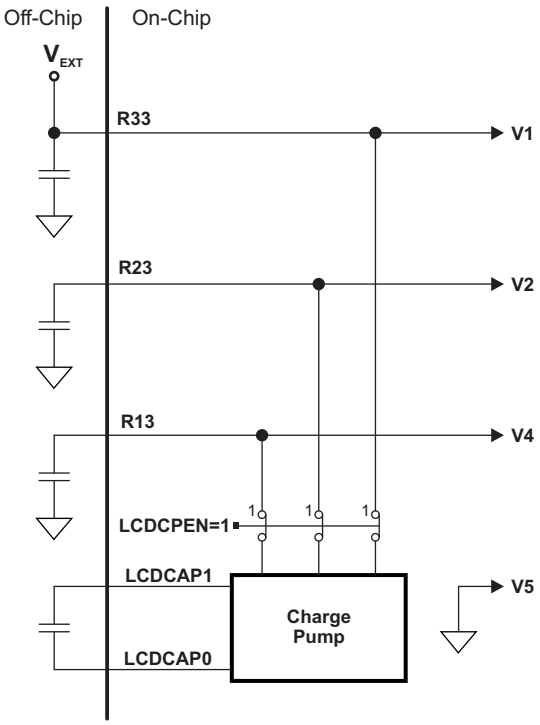


图 **14-8. LCD** 工作模式 **1**

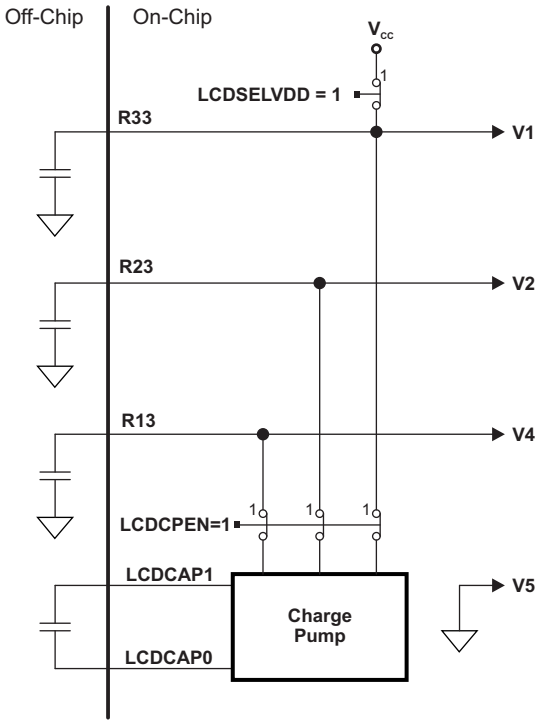


图 **14-9. LCD** 工作模式 **2**

14.2.8.2 内部电荷泵使能，内部 V_{REF} 使能（模式 3）

图 14-10 对模式 3 进行了描述。LCD 电压由连接到引脚 R13 的偏置电压发生器提供。内部电荷泵用于生成 LCD 电压 V1 和 V2，V5 接地。可在软件中更改 LCDVCTL 寄存器中的 VLCDx 位调整对比度。如果将 LCDREFMODE 设为 1，偏置电压发生器处于开关模式。因此，偏置电压发生器会在 1 个时钟周期内处于打开状态，其余 256 个时钟周期内处于关闭状态，以节约电能。如果将 LCDREFMODE 设为 0，则会将偏置电压发生器设为静态模式，从而能够驱动更大的 LCD 面板。

```
LCDSELVDD  = 0;       // Pin R33 is connected to external supply voltage
LCDCPEN    = 1;       // internal charge pump enabled
LCDREFEN   = 1;       // internal reference voltage at R13 is enabled
LCDCPFSELx = 0b1111;  // charge pump frequency select, slowest value
VLCDx      = 0b1000;  // VLCDx set to mid position
LCDON      = 1;       // enable LCD
```

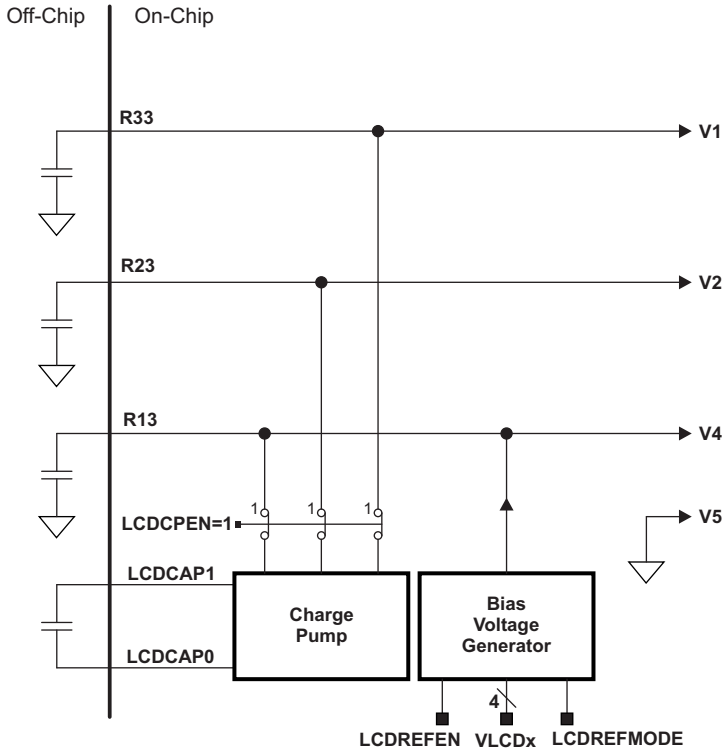


图 **14-10. LCD** 工作模式 **3**

注: 模式 3 是建议使用的工作模式，因为该模式的外部组件成本最低，工作电流也非常低。

14.2.8.3 内部电荷泵使能，内部 V_{REF} 禁用（模式 4）

图 14-11 显示的是模式 4。LCD 电压由连接到引脚 R13 的外部基准电压提供。内部电荷泵用于生成 LCD 偏置电压 V1 和 V2。V5 接地。可通过在 0.8V 和 1.2V 之间更改外部电压 $V_{REF,EXT}$ 来调整对比度。

```
LCDSELVDD  = 0;       // Pin R33 is connected to external supply voltage
LCDCPEN    = 1;       // internal charge pump enabled
LCDREFEN   = 0;       // internal reference voltage at R13 is disabled
LCDCPFSELx = 0b1111;  // charge pump frequency select, slowest value
VLCDx      = 0b0000;  // not used, set to reset value
LCDON      = 1;       // enable LCD
```

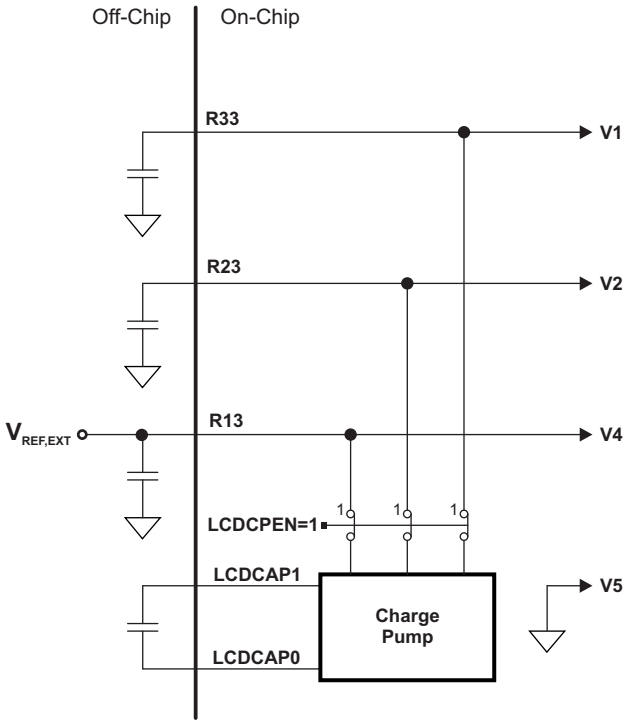


图 **14-11. LCD** 工作模式 **4**

14.2.9 LCD 中断

LCD_E 模块有三个可用中断源，每个中断源都有单独的使能信号和标志。

LCDFRMIFG、LCDBLKOFFIFG 和 LCDBLKONIFG 这三个中断标志进行了优先级排序且共用一个中断向量。中断向量寄存器 LCDIV 用于确定请求中断的标志。

优先级最高的已使能中断会在 LCDIV 寄存器中生成一个数字（请参见寄存器说明）。可评估该数字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 LCD 中断不会影响 LCDIV 值。

对 LCDIV 寄存器执行任何读访问都会自动将最高挂起中断标志复位。如果有另一个中断标志置 1，则在处理完最初的中断后会立即产生另一个中断。对 LCDIV 寄存器执行写访问会自动将所有挂起中断标志复位。此外，所有标志均可通过软件清零。

当使能闪烁（LCDBLKMODx = 01 或 10）时，LCDBLKONIFG 会在 BLKCLK 上升沿置 1，LCD 会切换为闪烁状态。当 LCDBLKMODx = 11 时，LCDBLKONIFG 也会在 BLKCLK 边沿置 1，选择闪烁存储器作为显示存储器。当一个 LCD 或闪烁存储寄存器被写入时，它会被自动清零。把 LCDBLKONIE 位置 1 来使能中断。

当使能闪烁（LCDBLKMODx = 01 或 10）时，LCDBLKOFFIFG 会在 BLKCLK 下降沿置 1，LCD 会切换为非闪烁状态。当 LCDBLKMODx = 11 时，LCDBLKONIFG 也会在 BLKCLK 边沿置 1，选择 LCD 存储器作为显示存储器。当一个 LCD 或闪烁存储寄存器被写入时，它会被自动清零。把 LCDBLKOFFIE 位置 1 来使能中断。

在一个帧边界处将 LCDFRMIFG 置 1。 当一个 LCD 或闪烁存储寄存器被写入时，它会被自动清零。把 LCDFRMIFGIE 位置 1 来使能中断。

14.2.9.1 LCDIV 软件示例

以下软件示例给出了 LCDIV 的建议用法以及处理开销。将 LCDIV 值添加到 PC 以自动跳转到相应的程序。

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。

```
; Interrupt handler for LCD_E interrupt flags.
LCDB_HND             ; Interrupt latency          6
   ADD &LCDBIV,PC    ; Add offset to Jump table   3
   RETI              ; Vector 0: No interrupt     5
   JMP LCDBLKON_HND  ; Vector 4: LCDBLKONIFG      2
   JMP LCDBLKOFF_HND ; Vector 6: LCDBLKOFFIFG     2
LCDFRM_HND           ; Vector 8: LCDFRMIFG
   ...    ; Task starts here
   RETI                                          5
LCDBLKON_HND ; Vector 4: LCDBLKONIFG
   ... ; Task starts here
   RETI ; Back to main program                   5
LCDBLKOFF_HND ; Vector 6: LCDBLKOFFIFG
   ... ; Task starts here
   RETI ; Back to main program                   5
```

14.2.10 静态模式

在静态模式中，每个 MSP430 段引脚驱动一个 LCD 段，并使用了一个串行通讯线 (COM0)。图 14-12给出了一些静态波形的示例。

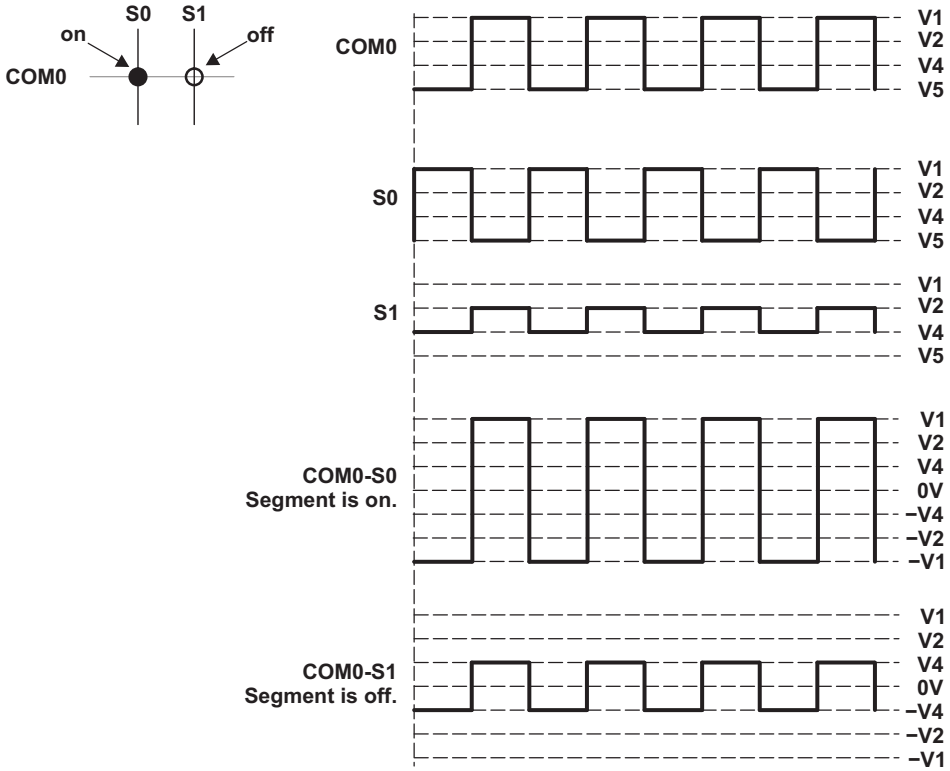


图 **14-12.** 静态波形的示例

14.2.11 2 线多路复用模式

在 2 线多路复用模式中，每个 MSP430 段引脚驱动两个 LCD 段，并使用了两个串行通讯线（COM0 和 COM1）。图 14-13 给出了一些 2 路复用 1/3 偏置波形的示例。

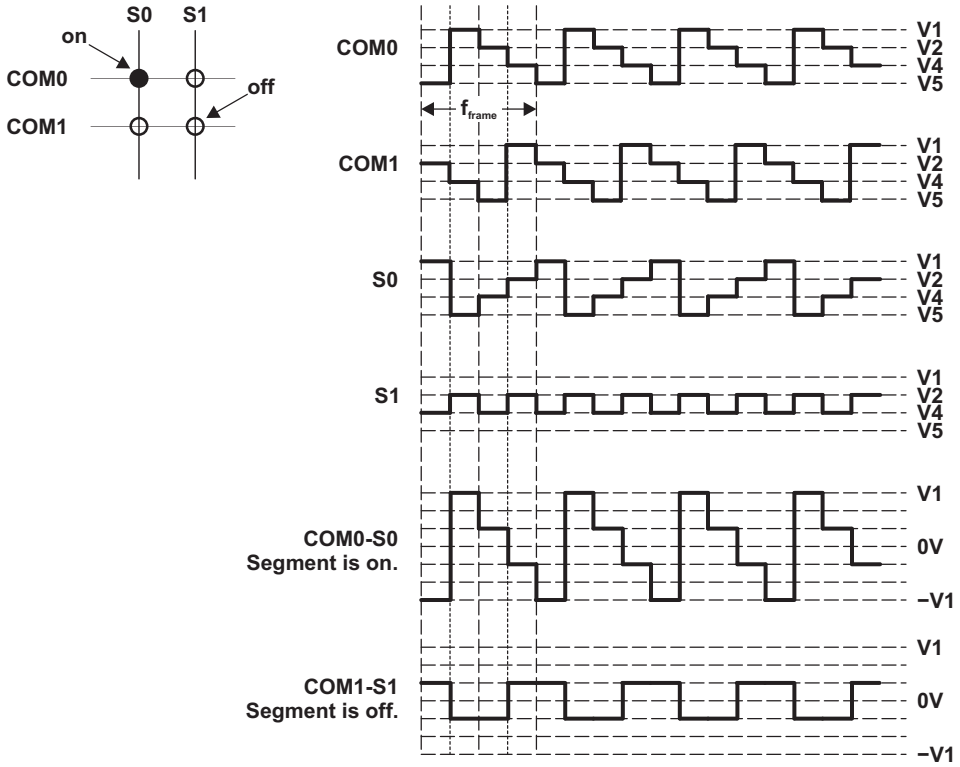


图 **14-13. 2** 线多路复用波形示例

14.2.12 3 线多路复用模式

在 430 线多路复用模式中，每个 MSP430 段引脚驱动三个 LCD 段，并使用了三个串行通讯线（COM0，COM1，和 COM2）。图 14-14给出了一些 3 线多路复用 1/3 偏压波形的示例。

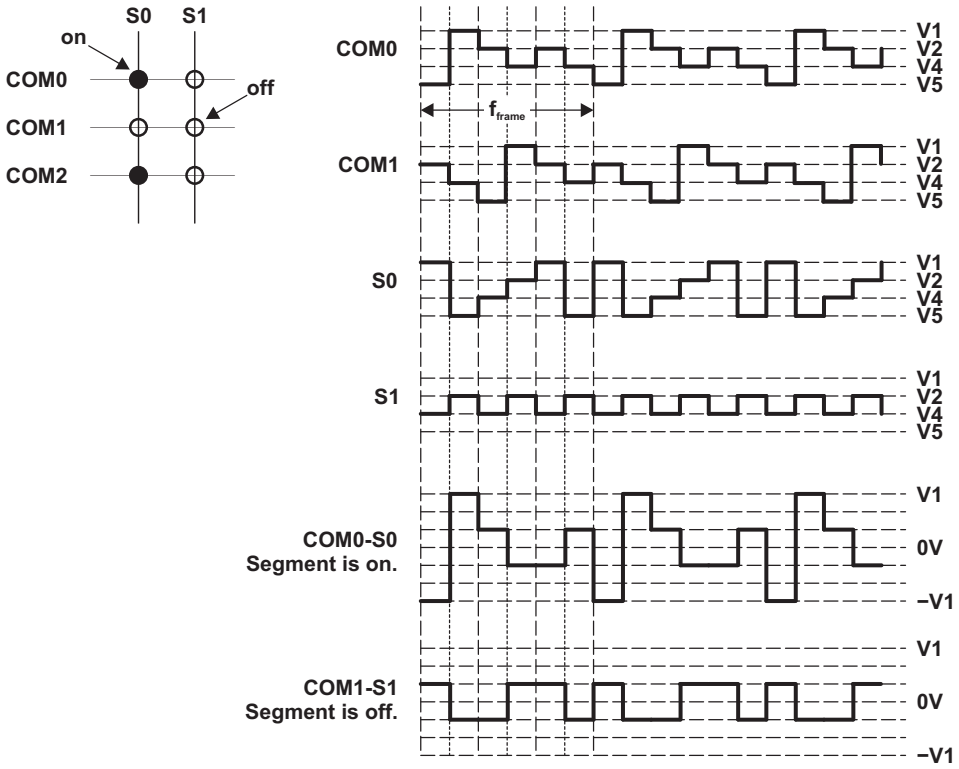


图 **14-14. 3** 线多路复用波形示例

14.2.13 4 线多路复用模式

在 4 线多路复用模式中，每个 MSP430 段引脚驱动四个 LCD 段，并使用了四个串行通讯线（COM0，COM1，COM2，和 COM3）。图 14-15给出了一些 4 线多路复用 1/3 偏压波形的示例。

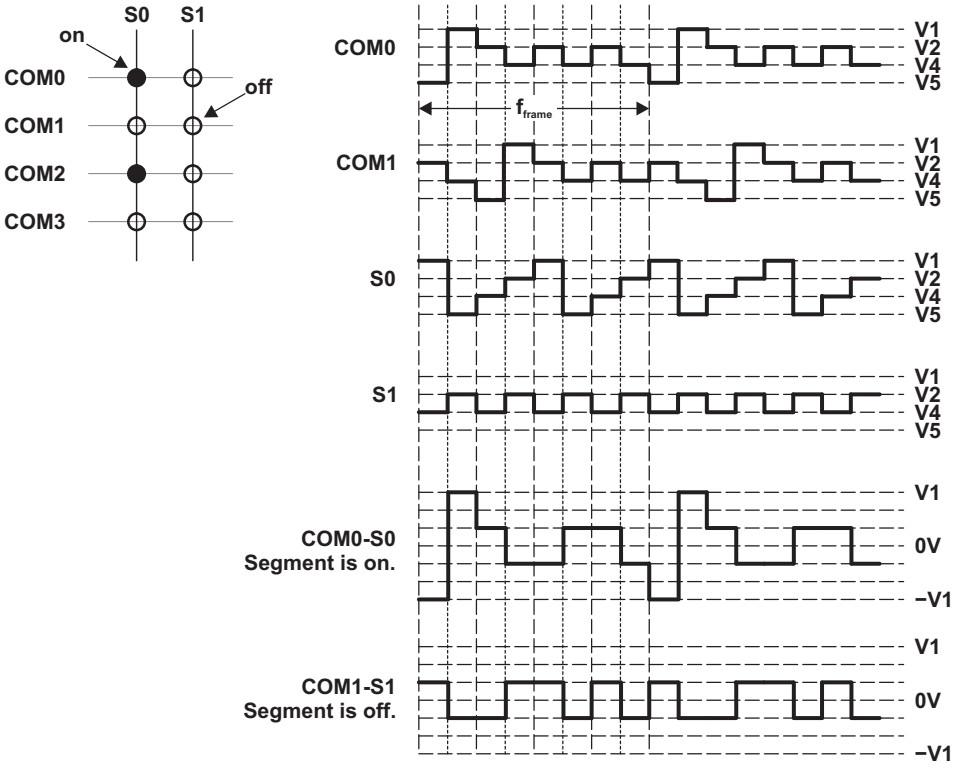


图 **14-15. 4** 线多路复用波形示例

14.2.14 6 线多路复用模式

在 6 线多路复用模式中，每个 MSP430 段引脚驱动六个 LCD 段，并使用了刘个串行通讯线（COM0，COM1，COM2，COM3，COM4，和 COM5）。图 14-16给出了一些 6 线多路复用 1/3 偏压波形的示例。

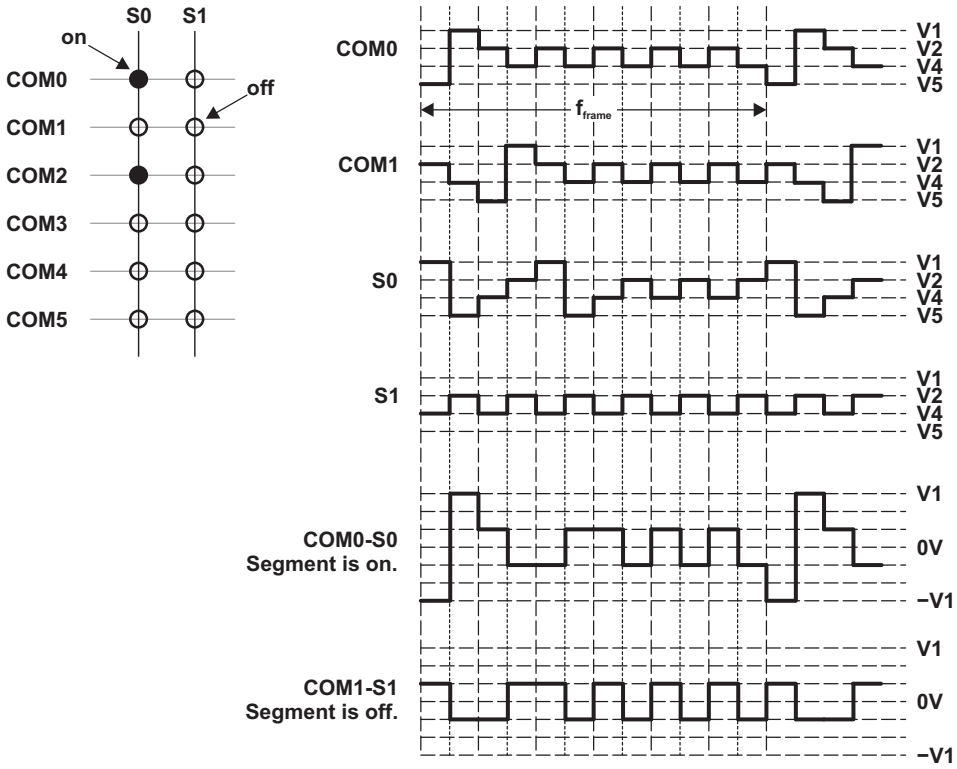


图 **14-16. 6** 线多路复用波形示例

14.2.15 8 Mux 模式

在 8 线多路复用模式中，每个 MSP430 段引脚驱动八个 LCD 段，并使用了八个串行通讯线（COM0 至 COM7）。图 14-17给出了一些 8 线多路复用 1/3 偏压波形的示例。

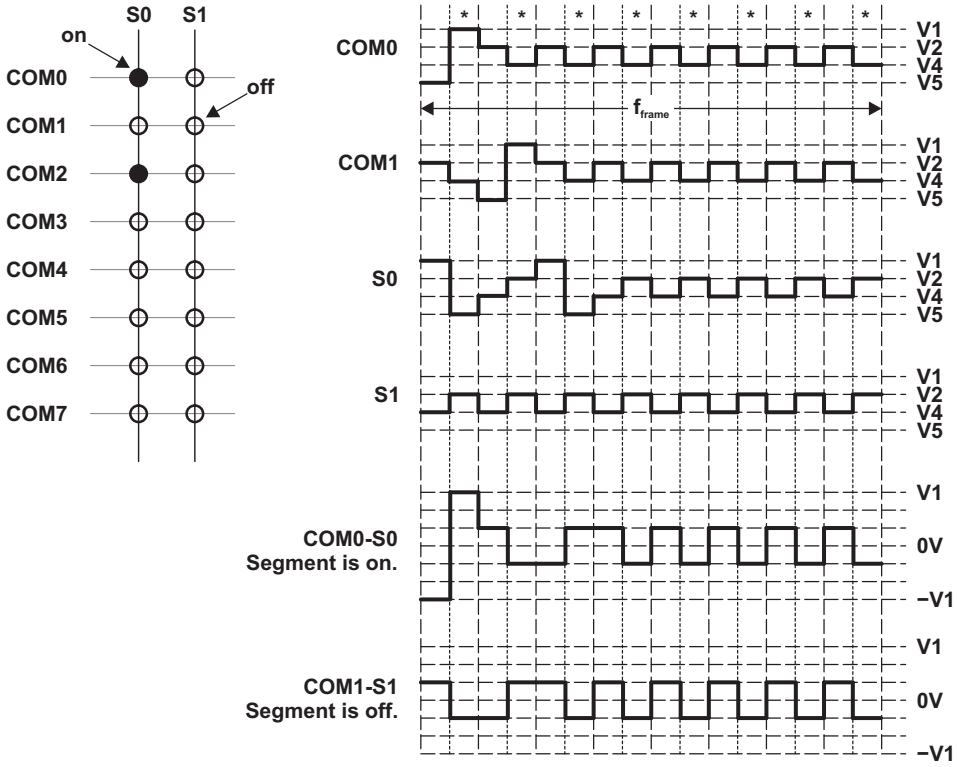


图 **14-17. 8** 线多路复用示例，**1/3** 偏压波形 **(LCDLP = 0)**

图 14-18给出了 LCDLP = 1 时 8 线多路复用 1/3 偏压波形的示例。令 LCDLP = 1，与低功率波形相比的电压序列被重新排序；即，图 14-17中所有标有“*”的时段被组合在了一起。同样的原则也适用于所有多路复用模式。

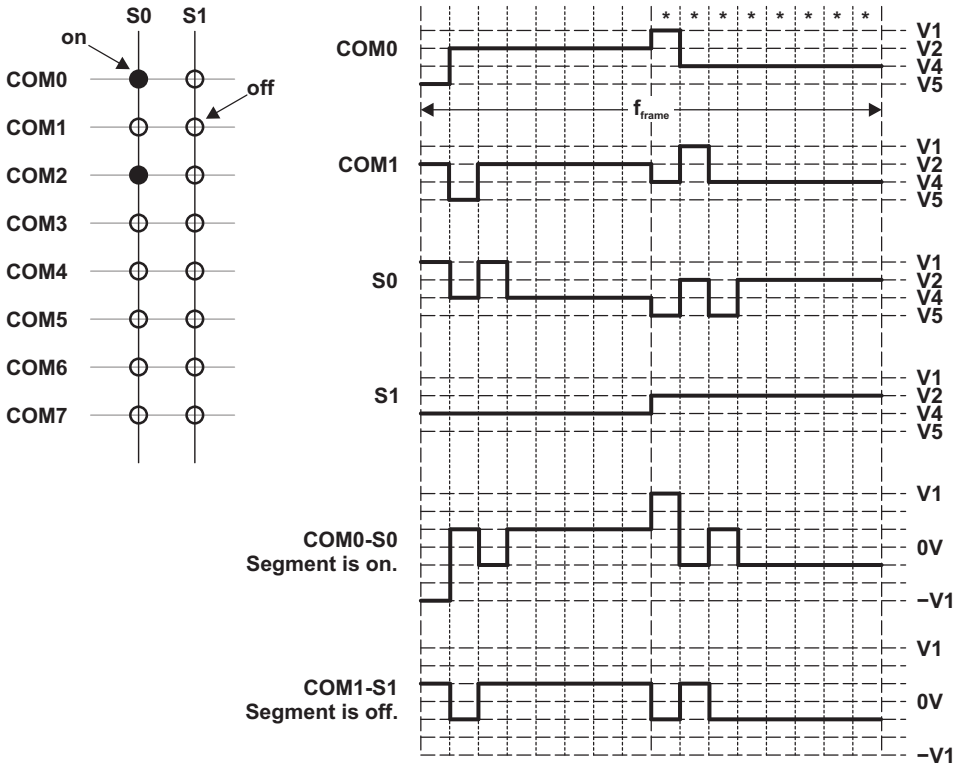


图 **14-18. 8** 线多路复用示例，**1/3** 偏压低功耗波形 **(LCDLP = 1)**

14.3 LCD_E 寄存器

表 14-6 到表 14-9 列出了 LCD_E 控制器寄存器。LCD 存储器和闪烁存储寄存器也可按字访问。

给定器件上的可用存储寄存器数量取决于可用段引脚的数量（请参见具体器件的数据表）。

表 14-6. LCD_E 寄存器

偏移量	首字母缩写词	寄存器名称	类型	复位	部分
000h	LCDCTL0	LCD_E 控制寄存器 0	读取/写入	3800h	14.3.1 节
002h	LCDCTL1	LCD_E 控制寄存器 1	读取/写入	0000h	14.3.2 节
004h	LCDBLKCTL	LCD_E 闪烁控制寄存器	读取/写入	0000h	14.3.3 节
006h	LCDMEMCTL	LCD_E 存储器控制寄存器	读取/写入	0000h	14.3.4 节
008h	LCDVCTL	LCD_E 电压控制寄存器	读取/写入	0000h	14.3.5 节
00Ah	LCDPCTL0	LCD_E 端口控制 0	读取/写入	0000h	14.3.6 节
00Ch	LCDPCTL1	LCD_E 端口控制 1	读取/写入	0000h	14.3.7 节
00Eh	LCDPCTL2	LCD_E 端口控制 2（256 个段）	读取/写入	0000h	14.3.8 节
010h	LCDPCTL3	LCD_E 端口控制 3（384 个段）	读取/写入	0000h	14.3.9 节
012h		保留	读取/写入	0000h	
014h	LCDCSSEL0	LCD_E COM/SEG 选择寄存器 0	读取/写入	0000h	14.3.10 节
016h	LCDCSSEL1	LCD_E COM/SEG 选择寄存器 1	读取/写入	0000h	14.3.11 节
018h	LCDCSSEL2	LCD_E COM/SEG 选择寄存器 2	读取/写入	0000h	14.3.12 节
01Ah	LCDCSSEL3	LCD_E COM/SEG 选择寄存器 3	读取/写入	0000h	14.3.13 节
01Ch		保留	读取/写入	0000h	
01Eh	LCDIV	LCD_E 中断向量	只读	0000h	14.3.16 节

表 14-7. 用于静态和 2 路复用到 4 路复用模式的 LCD 存储寄存器(1) (2)

偏移量	首字母缩写词	寄存器名称	类型	复位
020h	LCDM0W	LCD 存储器 0 字（S3、S2、S1 和 S0）	读取/写入	未改变
020h	LCDM0	LCD 存储器 0（S1 和 S0）	读取/写入	未改变
021h	LCDM1	LCD 存储器 1（S3 和 S2）	读取/写入	未改变
022h	LCD2W	LCD 存储器 2 字（S7、S6、S5 和 S4）	读取/写入	未改变
022h	LCDM2	LCD 存储器 2（S5 和 S4）	读取/写入	未改变
023h	LCDM3	LCD 存储器 3（S7 和 S6）	读取/写入	未改变
024h	LCD4W	LCD 存储器 4 字（S11、S10、S9 和 S8）	读取/写入	未改变
024h	LCDM4	LCD 存储器 4（S9 和 S8）	读取/写入	未改变
025h	LCDM5	LCD 存储器 5（S11 和 S10）	读取/写入	未改变
026h	LCDM6W	LCD 存储器 6 字（S15、S14、S13 和 S12）	读取/写入	未改变
026h	LCDM6	LCD 存储器 6（S13 和 S12）	读取/写入	未改变
027h	LCDM7	LCD 存储器 7（S15 和 S14）	读取/写入	未改变
028h	LCDM8W	LCD 存储器 8 字（S19、S18、S17 和 S16）	读取/写入	未改变
028h	LCDM8	LCD 存储器 8（S17 和 S16）	读取/写入	未改变
029h	LCDM9	LCD 存储器 9（S19 和 S18）	读取/写入	未改变
02Ah	LCDM10W	LCD 存储器 10 字（S23、S22、S21 和 S20）	读取/写入	未改变
02Ah	LCDM10	LCD 存储器 10（S21 和 S20）	读取/写入	未改变
02Bh	LCDM11	LCD 存储器 11（S23 和 S22）	读取/写入	未改变
02Ch	LCDM12W	LCD 存储器 12 字（S27、S26、S25 和 S24）	读取/写入	未改变
02Ch	LCDM12	LCD 存储器 12（S25 和 S24）	读取/写入	未改变
02Dh	LCDM13	LCD 存储器 13（S27 和 S26）	读取/写入	未改变
02Eh	LCDM14W	LCD 存储器 14 字（S31、S30、S29 和 S28）	读取/写入	未改变
02Eh	LCDM14	LCD 存储器 14（S29 和 S28）	读取/写入	未改变
02Fh	LCDM15	LCD 存储器 15（S31 和 S30）	读取/写入	未改变
030h	LCDM16W	LCD 存储器 16 字（S35、S34、S33 和 S32）	读取/写入	未改变
030h	LCDM16	LCD 存储器 16（S33 和 S32）	读取/写入	未改变
031h	LCDM17	LCD 存储器 17（S35 和 S34）	读取/写入	未改变
032h	LCDM18W	LCD 存储器 18 字（S39、S38、S37 和 S36）	读取/写入	未改变
032h	LCDM18	LCD 存储器 18（S37 和 S36）	读取/写入	未改变
033h	LCDM19	LCD 存储器 19（S39 和 S38）	读取/写入	未改变
034h	LCDM20W	LCD 存储器 20 字（S43、S42、S41 和 S40）	读取/写入	未改变
034h	LCDM20	LCD 存储器 20（S41 和 S40）	读取/写入	未改变
035h	LCDM21	LCD 存储器 21（S43 和 S42）	读取/写入	未改变
036h	LCDM22W	LCD 存储器 22 字（S47、S46、S45 和 S44）	读取/写入	未改变
036h	LCDM22	LCD 存储器 22（S45 和 S44）	读取/写入	未改变
037h	LCDM23	LCD 存储器 23（S47 和 S46）	读取/写入	未改变
038h	LCDM24W	LCD 存储器 24 字（S51、S50、S49 和 S48）	读取/写入	未改变
038h	LCDM24	LCD 存储器 24（S49 和 S48）	读取/写入	未改变
039h	LCDM25	LCD 存储器 25（S51 和 S50）	读取/写入	未改变
03Ah	LCDM26W	LCD 存储器 26 字（S55、S54、S53 和 S52）	读取/写入	未改变
03Ah	LCDM26	LCD 存储器 26（S53 和 S52）	读取/写入	未改变
03Bh	LCDM27	LCD 存储器 27（S55 和 S54）	读取/写入	未改变
03Ch	LCDM28W	LCD 存储器 28 字（S59、S58、S57 和 S56）	读取/写入	未改变
03Ch	LCDM28	LCD 存储器 28（S57 和 S56）	读取/写入	未改变
03Dh	LCDM29	LCD 存储器 29（S59 和 S58）	读取/写入	未改变
03Eh	LCDM30W	LCD 存储器 30 字（S63、S62、S61 和 S60）	读取/写入	未改变

(1) LCD 内存寄存器也可以作为字被访问。
(2) 给定器件上的可用存储寄存器数量取决于可用段引脚的数量。请参见具体器件的数据表。

表 **14-7.** 用于静态和 **2** 路复用到 **4** 路复用模式的 **LCD** 存储寄存器[(1)] [(2)] **(continued)**

偏移量	首字母缩写词	寄存器名称	类型	复位
03Eh	LCDM30	LCD 存储器 30（S61 和 S60）	读取/写入	未改变
03Fh	LCDM31	LCD 存储器 31（S63 和 S62）	读取/写入	未改变

表 14-8. 用于静态和 2 Mux 至 4 Mux 模式的 LCD 闪烁存储器寄存器(1)(2)

偏移量	首字母缩写词	寄存器名称	类型	复位
040h	LCDBM0W	LCD 闪烁存储器 0 字	读取/写入	未改变
040h	LCDBM0	LCD 闪烁存储器 0	读取/写入	未改变
041h	LCDBM1	LCD 闪存 1	读取/写入	未改变
042h	LCDBM2W	LCD 闪烁存储器 2 字	读取/写入	未改变
042h	LCDBM2	LCD 闪存 2	读取/写入	未改变
043h	LCDBM3	LCD 闪存 3	读取/写入	未改变
044h	LCDBM4W	LCD 闪烁存储器 4 字	读取/写入	未改变
044h	LCDBM4	LCD 闪存 4	读取/写入	未改变
045h	LCDBM5	LCD 闪存 5	读取/写入	未改变
046h	LCDBM6W	LCD 闪烁存储器 6 字	读取/写入	未改变
046h	LCDBM6	LCD 闪存 6	读取/写入	未改变
047h	LCDBM7	LCD 闪存 7	读取/写入	未改变
048h	LCDBM8W	LCD 闪烁存储器 8 字	读取/写入	未改变
048h	LCDBM8	LCD 闪存 8	读取/写入	未改变
049h	LCDBM9	LCD 闪存 9	读取/写入	未改变
04Ah	LCDBM10W	LCD 闪烁存储器 10 字	读取/写入	未改变
04Ah	LCDBM10	LCD 闪存 10	读取/写入	未改变
04Bh	LCDBM11	LCD 闪存 11	读取/写入	未改变
04Ch	LCDBM12W	LCD 闪烁存储器 12 字	读取/写入	未改变
04Ch	LCDBM12	LCD 闪存 12	读取/写入	未改变
04Dh	LCDBM13	LCD 闪存 13	读取/写入	未改变
04Eh	LCDBM14W	LCD 闪烁存储器 14 字	读取/写入	未改变
04Eh	LCDBM14	LCD 闪存 14	读取/写入	未改变
04Fh	LCDBM15	LCD 闪烁存储器 15	读取/写入	未改变
050h	LCDBM16W	LCD 闪烁存储器 16 字	读取/写入	未改变
050h	LCDBM16	LCD 闪烁存储器 16	读取/写入	未改变
051h	LCDBM17	LCD 闪烁存储器 17	读取/写入	未改变
052h	LCDBM18W	LCD 闪烁存储器 18 字	读取/写入	未改变
052h	LCDBM18	LCD 闪烁存储器 18	读取/写入	未改变
053h	LCDBM19	LCD 闪烁存储器 19	读取/写入	未改变
054h	LCDBM20W	LCD 闪烁存储器 20 字	读取/写入	未改变
054h	LCDBM20	LCD 闪烁存储器 20	读取/写入	未改变
055h	LCDBM21	LCD 闪烁存储器 21	读取/写入	未改变
056h	LCDBM22W	LCD 闪烁存储器 22 字	读取/写入	未改变
056h	LCDBM22	LCD 闪烁存储器 22	读取/写入	未改变
057h	LCDBM23	LCD 闪烁存储器 23	读取/写入	未改变
058h	LCDBM24W	LCD 闪烁存储器 24 字	读取/写入	未改变
058h	LCDBM24	LCD 闪烁存储器 24	读取/写入	未改变
059h	LCDBM25	LCD 闪烁存储器 25	读取/写入	未改变
05Ah	LCDBM26W	LCD 闪烁存储器 26 字	读取/写入	未改变
05Ah	LCDBM26	LCD 闪烁存储器 26	读取/写入	未改变
05Bh	LCDBM27	LCD 闪烁存储器 27	读取/写入	未改变
05Ch	LCDBM28W	LCD 闪烁存储器 28 字	读取/写入	未改变
05Ch	LCDBM28	LCD 闪烁存储器 28	读取/写入	未改变
05Dh	LCDBM29	LCD 闪烁存储器 29	读取/写入	未改变
05Eh	LCDBM30W	LCD 闪烁存储器 30 字	读取/写入	未改变

(1) LCD 闪存寄存器也可以作为字被访问。
(2) 给定器件上的可用存储寄存器数量取决于可用段引脚的数量（请参见具体器件的数据表）。

表 **14-8.** 用于静态和 **2 Mux** 至 **4 Mux** 模式的 **LCD** 闪烁存储器寄存器[(1)(2)] **(continued)**

偏移量	首字母缩写词	寄存器名称	类型	复位
05Eh	LCDBM30	LCD 闪烁存储器 30	读取/写入	未改变
05Fh	LCDBM31	LCD 闪烁存储器 31	读取/写入	未改变

表 14-9. 用于 5 路复用到 8 路复用模式的 LCD 存储寄存器(1)(2)

偏移量	首字母缩写词	寄存器名称	类型	复位
020h	LCDM0W	LCD 存储器 0 字（S1 和 S0）	读取/写入	未改变
020h	LCDM0	LCD 存储器 0 (S0)	读取/写入	未改变
021h	LCDM1	LCD 存储器 1 (S1)	读取/写入	未改变
022h	LCDM2W	LCD 存储器 2 字（S3 和 S2）	读取/写入	未改变
022h	LCDM2	LCD 存储器 2 (S2)	读取/写入	未改变
023h	LCDM3	LCD 存储器 3 (S3)	读取/写入	未改变
024h	LCDM4W	LCD 存储器 4 字（S5 和 S4）	读取/写入	未改变
024h	LCDM4	LCD 存储器 4 (S4)	读取/写入	未改变
025h	LCDM5	LCD 存储器 5 (S5)	读取/写入	未改变
026h	LCDM6W	LCD 存储器 6 字（S7 和 S6）	读取/写入	未改变
026h	LCDM6	LCD 存储器 6 (S6)	读取/写入	未改变
027h	LCDM7	LCD 存储器 7 (S7)	读取/写入	未改变
028h	LCDM8W	LCD 存储器 8 字（S9 和 S8）	读取/写入	未改变
028h	LCDM8	LCD 存储器 8 (S8)	读取/写入	未改变
029h	LCDM9	LCD 存储器 9 (S9)	读取/写入	未改变
02Ah	LCDM10W	LCD 存储器 10 字（S11 和 S10）	读取/写入	未改变
02Ah	LCDM10	LCD 存储器 10 (S10)	读取/写入	未改变
02Bh	LCDM11	LCD 存储器 11 (S11)	读取/写入	未改变
02Ch	LCDM12W	LCD 存储器 12 字（S13 和 S12）	读取/写入	未改变
02Ch	LCDM12	LCD 存储器 12 (S12)	读取/写入	未改变
02Dh	LCDM13	LCD 存储器 13 (S13)	读取/写入	未改变
02Eh	LCDM14W	LCD 存储器 14 字（S15 和 S14）	读取/写入	未改变
02Eh	LCDM14	LCD 存储器 14 (S14)	读取/写入	未改变
02Fh	LCDM15	LCD 存储器 15 (S15)	读取/写入	未改变
030h	LCDM16W	LCD 存储器 16 字（S17 和 S16）	读取/写入	未改变
030h	LCDM16	LCD 存储器 16 (S16)	读取/写入	未改变
031h	LCDM17	LCD 存储器 17 (S17)	读取/写入	未改变
032h	LCDM18W	LCD 存储器 18 字（S19 和 S18）	读取/写入	未改变
032h	LCDM18	LCD 存储器 18 (S18)	读取/写入	未改变
033h	LCDM19	LCD 存储器 19 (S19)	读取/写入	未改变
034h	LCDM20W	LCD 存储器 20 字（S21 和 S20）	读取/写入	未改变
034h	LCDM20	LCD 存储器 20 (S20)	读取/写入	未改变
035h	LCDM21	LCD 存储器 21 (S21)	读取/写入	未改变
036h	LCDM22W	LCD 存储器 22 字（S23 和 S22）	读取/写入	未改变
036h	LCDM22	LCD 存储器 22 (S22)	读取/写入	未改变
037h	LCDM23	LCD 存储器 23 (S23)	读取/写入	未改变
038h	LCDM24W	LCD 存储器 24 字（S25 和 S24）	读取/写入	未改变
038h	LCDM24	LCD 存储器 24 (S24)	读取/写入	未改变
039h	LCDM25	LCD 存储器 25 (S25)	读取/写入	未改变
03Ah	LCDM26W	LCD 存储器 26 字（S27 和 S26）	读取/写入	未改变
03Ah	LCDM26	LCD 存储器 26 (S26)	读取/写入	未改变
03Bh	LCDM27	LCD 存储器 27 (S27)	读取/写入	未改变
03Ch	LCDM28W	LCD 存储器 28 字（S29 和 S28）	读取/写入	未改变
03Ch	LCDM28	LCD 存储器 28 (S28)	读取/写入	未改变
03Dh	LCDM29	LCD 存储器 29 (S29)	读取/写入	未改变
03Eh	LCDM30W	LCD 存储器 30 字（S31 和 S30）	读取/写入	未改变

(1) LCD 内存寄存器也可以作为字被访问。
(2) 给定器件上的可用存储寄存器数量取决于可用段引脚的数量（请参见具体器件的数据表）。

表 14-9. 用于 5 路复用到 8 路复用模式的 LCD 存储寄存器[(1)(2)] (continued)

偏移量	首字母缩写词	寄存器名称	类型	复位
03Eh	LCDM30	LCD 存储器 30 (S30)	读取/写入	未改变
03Fh	LCDM31	LCD 存储器 31 (S31)	读取/写入	未改变
040h	LCDM32W	LCD 存储器 32 字（S33 和 S32）	读取/写入	未改变
040h	LCDM32	LCD 存储器 32 (S32)	读取/写入	未改变
041h	LCDM33	LCD 存储器 33 (S33)	读取/写入	未改变
042h	LCDM34W	LCD 存储器 34 字（S35 和 S34）	读取/写入	未改变
042h	LCDM34	LCD 存储器 34 (S34)	读取/写入	未改变
043h	LCDM35	LCD 存储器 35 (S35)	读取/写入	未改变
044h	LCDM36W	LCD 存储器 36 字（S37 和 S36）	读取/写入	未改变
044h	LCDM36	LCD 存储器 36 (S36)	读取/写入	未改变
045h	LCDM37	LCD 存储器 37 (S37)	读取/写入	未改变
046h	LCDM38W	LCD 存储器 38 字（S39 和 S38）	读取/写入	未改变
046h	LCDM38	LCD 存储器 38 (S38)	读取/写入	未改变
047h	LCDM39	LCD 存储器 39 (S39)	读取/写入	未改变
048h	LCDM40W	LCD 存储器 40 字（S41 和 S40）	读取/写入	未改变
048h	LCDM40	LCD 存储器 40 (S40)	读取/写入	未改变
049h	LCDM41	LCD 存储器 41 (S41)	读取/写入	未改变
04Ah	LCDM42W	LCD 存储器 42 字（S43 和 S42）	读取/写入	未改变
04Ah	LCDM42	LCD 存储器 42 (S42)	读取/写入	未改变
04Bh	LCDM43	LCD 存储器 43 (S43)	读取/写入	未改变
04Ch	LCDM44W	LCD 存储器 44 字（S45 和 S44）	读取/写入	未改变
04Ch	LCDM44	LCD 存储器 44 (S44)	读取/写入	未改变
04Dh	LCDM45	LCD 存储器 45 (S45)	读取/写入	未改变
04Eh	LCDM46W	LCD 存储器 46 字（S47 和 S46）	读取/写入	未改变
04Eh	LCDM46	LCD 存储器 46 (S46)	读取/写入	未改变
04Fh	LCDM47	LCD 存储器 47 (S47)	读取/写入	未改变
050h	LCDM48W	LCD 存储器 48 字（S49 和 S48）	读取/写入	未改变
050h	LCDM48	LCD 存储器 48 (S48)	读取/写入	未改变
051h	LCDM49	LCD 存储器 49 (S49)	读取/写入	未改变
052h	LCDM50W	LCD 存储器 50 字（S51 和 S50）	读取/写入	未改变
052h	LCDM50	LCD 存储器 50 (S50)	读取/写入	未改变
053h	LCDM51	LCD 存储器 51 (S51)	读取/写入	未改变
054h	LCDM52W	LCD 存储器 52 字（S53 和 S52）	读取/写入	未改变
054h	LCDM52	LCD 存储器 52 (S52)	读取/写入	未改变
055h	LCDM53	LCD 存储器 53 (S53)	读取/写入	未改变
056h	LCDM54W	LCD 存储器 54 字（S55 和 S54）	读取/写入	未改变
056h	LCDM54	LCD 存储器 54 (S54)	读取/写入	未改变
057h	LCDM55	LCD 存储器 55 (S55)	读取/写入	未改变
058h	LCDM56W	LCD 存储器 56 字（S57 和 S56）	读取/写入	未改变
058h	LCDM56	LCD 存储器 56 (S56)	读取/写入	未改变
059h	LCDM57	LCD 存储器 57 (S57)	读取/写入	未改变
05Ah	LCDM58W	LCD 存储器 58 字（S59 和 S58）	读取/写入	未改变
05Ah	LCDM58	LCD 存储器 58 (S58)	读取/写入	未改变
05Bh	LCDM59	LCD 存储器 59 (S59)	读取/写入	未改变
05Ch	LCDM60W	LCD 存储器 60 字（S61 和 S60）	读取/写入	未改变
05Ch	LCDM60	LCD 存储器 60 (S60)	读取/写入	未改变
05Dh	LCDM61	LCD 存储器 61 (S61)	读取/写入	未改变

表 **14-9.** 用于 **5** 路复用到 **8** 路复用模式的 **LCD** 存储寄存器[(1)(2)] **(continued)**

偏移量	首字母缩写词	寄存器名称	类型	复位
05Eh	LCDM62W	LCD 存储器 62 字（S63 和 S62）	读取/写入	未改变
05Eh	LCDM62	LCD 存储器 62 (S62)	读取/写入	未改变
05Fh	LCDM63	LCD 存储器 63 (S63)	读取/写入	未改变

14.3.1 LCDCTL0 寄存器

LCD_E 控制寄存器 0

图 **14-19. LCDCTL0** 寄存器

15	14	13	12	11	10	9	8
LCDDIVx					保留		
rw-{0}	rw-{0}	rw-{1}	rw-{1}	rw-{1}	r-{0}	r-{0}	r-{0}
7	6	5	4	3	2	1	0
LCDSSEL		LCDMXx			LCDSON	LCDLP	LCDON
rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}

表 **14-10. LCDCTL0** 寄存器说明

位	字段	类型	复位	说明
15-11	LCDDIVx	RW	00111b	LCD 分频器。搭配使用 LCDMXx 时可计算 LCD 频率 f_{LCD}，计算公式为：$f_{LCD} = f_{SOURCE} / ((LCDDIVx + 1) \times 值[LCDMXx])$。应仅在 LCDON = 0 时更改。 00000b = 不分频 00001b = 2 分频 ⋮ 11110b = 31 分频 11111b = 32 分频
10-8	保留	R	0h	保留
7-6	LCDSSEL	RW	0h	LCD 和闪烁频率的时钟频率 f_{SOURCE} 选择。应仅在 LCDON = 0 时更改。 00b = XT1CLK 01b = ACLK（30kHz 到 40kHz） 10b = VLOCLK 11b = 保留
5-3	LCDMXx	RW	0h	LCD 多路复用率。这些位选择 LCD 模式。应仅在 LCDON = 0 时更改。 000b = 静态 001b = 2 mux 010b = 3 mux 011b = 4 mux 100b = 5 mux 101b = 6 mux 110b = 7 mux 111b = 8 mux
2	LCDSON	RW	0h	LSD 段打开该位通过关闭所有段线，但却保持 LCD 时序发生器和 R33 使能来支持闪存 LCD 应用。 0b = 所有的 LCD 段都关闭。 1b = 所有的 LCD 段都是能并且根据它们相应的内存位置打开或关闭。
1	LCDLP	RW	0h	LCD 低功率波形 0b = 段上为标准 LCD 波形并且共通线被选择. 1b = 选择段和公共线上的低功率 LCD 波形。
0	LCDON	RW	0h	LCD 打开。此位会接通或关闭 LCD_E 模块。 0b = 关闭 LCD_E 模块 1b = 接通 LCD_E 模块

14.3.2 LCDCTL1 寄存器

LCD_E 控制寄存器 1

图 **14-20. LCDCTL1** 寄存器

15	14	13	12	11	10	9	8
保留					LCDBLKONIE	LCDBLKOFFIE	LCDFRMIE
r0	r0	r0	r0	r0	rw-{0}	rw-{0}	rw-{0}
7	6	5	4	3	2	1	0
保留					LCDBLKONIFG	LCDBLKOFFIFG	LCDFRMIFG
r0	r0	r0	r0	r0	rw-{0}	rw-{0}	rw-{0}

表 **14-11. LCDCTL1** 寄存器说明

位	字段	类型	复位	说明
15-11	保留	R	0h	保留
10	LCDBLKONIE	RW	0h	LCD 闪存中断使能，段切换到开启状态 0b = 中断被禁用 1b = 中断被启用
9	LCDBLKOFFIE	RW	0h	LCD 闪存中断使能，段切换到关闭状态 0b = 中断被禁用 1b = 中断被启用
8 个	LCDFRMIE	RW	0h	LCD 帧中断使能 0b = 中断被禁用 1b = 中断被使能
7-3	保留	R	0h	保留
2	LCDBLKONIFG	RW	0h	LCD 闪烁中断标志，在 BLKCLK 的上升沿置 1。当数据被写入内存寄存器时，这些标志将会被自动清除。 0b = 无中断挂起 1b = 中断挂起
1	LCDBLKOFFIFG	RW	0h	LCD 闪烁中断标志，在 BLKCLK 的下降沿置 1。当数据被写入内存寄存器时，这些标志将会被自动清除。 0b = 无中断挂起 1b = 中断挂起
0	LCDFRMIFG	RW	0h	LCD 帧中断标志。当数据被写入内存寄存器时，这些标志将会被自动清除。 0b = 无中断挂起 1b = 中断挂起

14.3.3 LCDBLKCTL 寄存器

LCD_E 闪烁控制寄存器

图 **14-21. LCDBLKCTL** 寄存器

15	14	13	12	11	10	9	8
保留							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
保留			LCDBLKPREx			LCDBLKMODx	
r0	r0	r0	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}

表 **14-12. LCDBLKCTL** 寄存器说明

位	字段	类型	复位	说明
15-5	保留	R	0h	
4-2	LCDBLKPREx	RW	0h	闪烁频率的时钟预分频器。搭配使用 LCDMXx 时可计算闪烁频率 f_{BLINK}，计算公式为：$f_{BLINK} = f_{LCD} / ((LCDMXx + 1) \times 2^{(LCDBLKPREx + 2)})$。 应仅在 LCDBLKMODx = 00 时更改 LCDMXx 和 LCDBLKPREx 的设置。 000b = 4 分频 001b = 8 分频 010b = 16 分频 011b = 32 分频 100b = 64 分频 101b = 128 分频 110b = 256 分频 111b = 512 分频
1-0	LCDBLKMODx	RW	0h	闪烁模式 00b = 闪烁模式被禁用. 01b = 单个段的闪烁如闪烁存储器寄存器 LCDBMx 中那样被启用。在 mux 模式大于 5 时，闪烁被禁用。 10b = 所有段闪烁 11b = 在 LCDMx 和 LCDBMx 存储器寄存器中所存储的显示内容之间切换在 mux 模式大于 5 时，闪烁被禁用。

14.3.4 LCDMEMCTL 寄存器

LCD_E 存储器控制寄存器

图 14-22. LCDMEMCTL 寄存器

15	14	13	12	11	10	9	8
保留							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
保留					LCDCLRBM	LCDCLRM	LCDDISP
r0	r0	r0	r0	r0	rw-{0}	rw-{0}	rw-{0}

表 14-13. LCDMEMCTL 寄存器说明

位	字段	类型	复位	说明
15-3	保留	R	0h	保留
2	LCDCLRBM	RW	0h	清除 LCD 闪烁存储 清除所有的闪烁存储寄存器 LCDBMx。当闪烁存储被清除时，该位自动复位。 在 5 路复用模式及更高模式下将该位置 1 不起作用。该位会立即再次复位。 0b = 闪烁存储寄存器 LCDBMx 的内容保持不变 1b = 清除闪烁存储寄存器 LCDBMx 所有内容
1	LCDCLRM	RW	0h	清除 LCD 存储 清除所有的 LCD 寄存器 LCDMx。当 LCD 存储被清除时，该位自动复位。 0b = LCD 存储寄存器 LCDMx 的内容保持不变 1b = 清除 LCD 存储寄存器 LCDMx 所有内容
0	LCDDISP	RW	0h	选择用于显示的 LCD 存储寄存器 当 LCDBLKMODx = 00 时，LCDDISP 可通过软件置 1。 这个位在 LCDBLKMODx = 01 and LCDBLKMODx = 10 时被清零或复用模式 ≥ 5 时被选择并且不能由软件更改。 当 LCDBLKMODx = 11 时，这个位反映了当前的显示存储器，但是不能被软件更改。当返回到 LCDBLKMODx = 00 时，该位被清除。 0b = 显示 LCD 存储寄存器 LCDMx 内容 1b = 显示 LCD 存储寄存器 LCDBMx 内容

14.3.5 LCDVCTL 寄存器

LCD_E 电压控制寄存器

图 14-23. LCDVCTL 寄存器

15	14	13	12	11	10	9	8
LCDCPFSELx				VLCDx			
rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}
7	6	5	4	3	2	1	0
LCDCPEN	LCDREFEN	LCDSELVDD	保留				LCDREFMODE
rw-{0}	rw-{0}	rw-{0}	r0	r0	r0	r0	rw-{0}

表 14-14. LCDVCTL 寄存器说明

位	字段	类型	复位	说明
15-12	LCDCPFSELx	RW	0h	电荷泵频率选择。时钟源可以是 XT1、ACLK 或 VLO（4 位，如果 $f_{SOURCE} = f_{ACLK} = 32.768kHz$） 0000b = 32.768kHz / 1 / 8 = 4.096kHz 0001b = 32.768kHz / 2 / 8 = 2.048kHz 0010b = 32.768kHz / 3 / 8 = 1.365kHz 0011b = 32.768kHz / 4 / 8 = 1.024kHz 0100b = 32.768kHz / 5 / 8 = 819Hz 0101b = 32.768kHz / 6 / 8 = 682Hz 0110b = 32.768kHz / 7 / 8 = 585Hz 0111b = 32.768kHz / 8 / 8 = 512Hz 1000b = 32.768kHz / 9 / 8 = 455Hz 1001b = 32.768kHz / 10 / 8 = 409Hz 1010b = 32.768kHz / 11 / 8 = 372Hz 1011b = 32.768kHz / 12 / 8 = 341Hz 1100b = 32.768kHz / 13 / 8 = 315Hz 1101b = 32.768kHz / 14 / 8 = 292Hz 1110b = 32.768kHz / 15 / 8 = 273Hz 1111b = 32.768kHz / 16 / 8 = 256Hz
11-8	VLCDx	RW	0h	R13 上的内部基准电压选择。仅当 LCDCPEN = 1 且 LCDREFEN = 1 时有效。 0000b = 2.60V 0001b = 2.66V 0010b = 2.72V 0011b = 2.78V 0100b = 2.84V 0101b = 2.90V 0110b = 2.96V 0111b = 3.02V 1000b = 3.08V 1001b = 3.14V 1010b = 3.20V 1011b = 3.26V 1100b = 3.32V 1101b = 3.38V 1110b = 3.44V 1111b = 3.50V
7	LCDCPEN	RW	0h	电荷泵使能 0b = 电荷泵被禁用(1) 1b = 当 V_{LCD} 在内部生成 (VLCDEXT = 0)，且 VLCDx > 0 或 VLCDREFx > 0 时，使能电荷泵。

(1) 要使用 LCD，必须将外部电阻分压器连接至 R13、R23 和 R33。

表 **14-14. LCDVCTL** 寄存器说明 **(continued)**

位	字段	类型	复位	说明
6	LCDREFEN	RW	0h	R13 上的内部基准电压使能 0b = 禁用内部基准电压 1b = 使能内部基准电压
5	LCDSELVDD	RW	0h	选择 R33 是由内部 V_{CC} 供电还是由电荷泵供电 0b = R33 连接至外部电源 1b = R33 内部连接至 V_{CC}
4-1	保留	R	0h	保留
0	LCDREFMODE	RW	0h	选择 R13 电压处于开关模式还是静态模式 0b = 静态模式 1b = 开关模式

14.3.6 LCDPCTL0 寄存器

LCD_E 端口控制寄存器 0

应仅在 LCDON = 0 时更改 LCDSx 的设置。

图 14-24. LCDPCTL0 寄存器

15	14	13	12	11	10	9	8
LCDS15	LCDS14	LCDS13	LCDS12	LCDS11	LCDS10	LCDS9	LCDS8
rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}
7	6	5	4	3	2	1	0
LCDS7	LCDS6	LCDS5	LCDS4	LCDS3	LCDS2	LCDS1	LCDS0
rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}

表 14-15. LCDPCTL0 寄存器说明

位	字段	类型	复位	说明
15	LCDS15	RW	0h	LCD 引脚 15 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
14	LCDS14	RW	0h	LCD 引脚 14 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
13	LCDS13	RW	0h	LCD 引脚 13 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
12	LCDS12	RW	0h	LCD 引脚 12 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
11	LCDS11	RW	0h	LCD 引脚 11 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
10	LCDS10	RW	0h	LCD 引脚 10 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
9	LCDS9	RW	0h	LCD 引脚 9 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
8	LCDS8	RW	0h	LCD 引脚 8 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
7	LCDS7	RW	0h	LCD 引脚 7 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
6	LCDS6	RW	0h	LCD 引脚 6 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。

表 14-15. LCDPCTL0 寄存器说明 (continued)

位	字段	类型	复位	说明
5	LCDS5	RW	0h	LCD 引脚 5 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
4	LCDS4	RW	0h	LCD 引脚 4 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
3	LCDS3	RW	0h	LCD 引脚 3 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
2	LCDS2	RW	0h	LCD 引脚 2 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
1	LCDS1	RW	0h	LCD 引脚 1 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
0	LCDS0	RW	0h	LCD 引脚 0 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。

14.3.7 LCDPCTL1 寄存器

LCD_E 端口控制寄存器 1

应仅在 LCDON = 0 时更改 LCDSx 的设置。

图 14-25. LCDPCTL1 寄存器

15	14	13	12	11	10	9	8
LCDS31	LCDS30	LCDS29	LCDS28	LCDS27	LCDS26	LCDS25	LCDS24
rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}
7	6	5	4	3	2	1	0
LCDS23	LCDS22	LCDS21	LCDS20	LCDS19	LCDS18	LCDS17	LCDS16
rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}

表 14-16. LCDPCTL1 寄存器说明

位	字段	类型	复位	说明
15	LCDS31	RW	0h	LCD 引脚 31 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
14	LCDS30	RW	0h	LCD 引脚 30 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
13	LCDS29	RW	0h	LCD 引脚 29 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
12	LCDS28	RW	0h	LCD 引脚 28 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
11	LCDS27	RW	0h	LCD 引脚 27 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
10	LCDS26	RW	0h	LCD 引脚 26 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
9	LCDS25	RW	0h	LCD 引脚 25 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
8	LCDS24	RW	0h	LCD 引脚 24 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
7	LCDS23	RW	0h	LCD 段线 23 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
6	LCDS22	RW	0h	LCD 段线 22 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。

表 14-16. LCDPCTL1 寄存器说明 (continued)

位	字段	类型	复位	说明
5	LCDS21	RW	0h	LCD 段线 21 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
4	LCDS20	RW	0h	LCD 段线 20 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
3	LCDS19	RW	0h	LCD 段线 19 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
2	LCDS18	RW	0h	LCD 段线 18 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
1	LCDS17	RW	0h	LCD 段线 17 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
0	LCDS16	RW	0h	LCD 段线 16 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。

14.3.8 LCDPCTL2 寄存器

LCD_E 端口控制寄存器 2（256 个段）

应仅在 LCDON = 0 时更改 LCDSx 的设置。

图 14-26. LCDPCTL2 寄存器

15	14	13	12	11	10	9	8
LCDS47	LCDS46	LCDS45	LCDS44	LCDS43	LCDS42	LCDS41	LCDS40
rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}
7	6	5	4	3	2	1	0
LCDS39	LCDS38	LCDS37	LCDS36	LCDS35	LCDS34	LCDS33	LCDS32
rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}

表 14-17. LCDPCTL2 寄存器说明

位	字段	类型	复位	说明
15	LCDS47	RW	0h	LCD 引脚 47 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
14	LCDS46	RW	0h	LCD 引脚 46 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
13	LCDS45	RW	0h	LCD 引脚 45 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
12	LCDS44	RW	0h	LCD 引脚 44 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
11	LCDS43	RW	0h	LCD 引脚 43 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
10	LCDS42	RW	0h	LCD 引脚 42 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
9	LCDS41	RW	0h	LCD 引脚 41 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
8	LCDS40	RW	0h	LCD 引脚 40 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
7	LCDS39	RW	0h	LCD 引脚 39 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
6	LCDS38	RW	0h	LCD 引脚 38 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。

表 14-17. LCDPCTL2 寄存器说明 (continued)

位	字段	类型	复位	说明
5	LCDS37	RW	0h	LCD 引脚 37 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
4	LCDS36	RW	0h	LCD 引脚 36 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
3	LCDS35	RW	0h	LCD 引脚 35 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
2	LCDS34	RW	0h	LCD 引脚 34 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
1	LCDS33	RW	0h	LCD 引脚 33 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
0	LCDS32	RW	0h	LCD 引脚 32 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。

14.3.9 LCDPCTL3 寄存器

LCD_E 端口控制寄存器 3（384 个段，COM 与段共享）

应仅在 LCDON = 0 时更改 LCDSx 的设置。

图 14-27. LCDPCTL3 寄存器

15	14	13	12	11	10	9	8
LCDS63	LCDS62	LCDS61	LCDS60	LCDS59	LCDS58	LCDS57	LCDS56
rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}
7	6	5	4	3	2	1	0
LCDS55	LCDS54	LCDS53	LCDS52	LCDS51	LCDS50	LCDS49	LCDS48
rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}

表 14-18. LCDPCTL3 寄存器说明

位	字段	类型	复位	说明
15	LCDS63	RW	0h	LCD 引脚 63 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
14	LCDS62	RW	0h	LCD 引脚 62 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
13	LCDS61	RW	0h	LCD 引脚 61 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
12	LCDS60	RW	0h	LCD 引脚 60 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
11	LCDS59	RW	0h	LCD 引脚 59 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
10	LCDS58	RW	0h	LCD 引脚 58 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
9	LCDS57	RW	0h	LCD 引脚 57 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
8	LCDS56	RW	0h	LCD 引脚 56 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
7	LCDS55	RW	0h	LCD 引脚 55 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
6	LCDS54	RW	0h	LCD 引脚 54 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。

表 14-18. LCDPCTL3 寄存器说明 (continued)

位	字段	类型	复位	说明
5	LCDS53	RW	0h	LCD 引脚 53 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
4	LCDS52	RW	0h	LCD 引脚 52 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
3	LCDS51	RW	0h	LCD 引脚 51 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
2	LCDS50	RW	0h	LCD 引脚 50 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
1	LCDS49	RW	0h	LCD 引脚 49 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。
0	LCDS48	RW	0h	LCD 引脚 48 使能。该位仅影响多路复用功能引脚。专用的 LCD 引脚一直执行 LCD 功能。 0b = 复用引脚执行端口功能。 1b = 引脚执行 LCD 功能。

14.3.10 LCDCSSEL0 寄存器

LCD_E COM/SEG 选择寄存器 0

图 **14-28. LCDCSSEL0** 寄存器

15	14	13	12	11	10	9	8
LCDCSS15	LCDCSS14	LCDCSS13	LCDCSS12	LCDCSS11	LCDCSS10	LCDCSS9	LCDCSS8
rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}
7	6	5	4	3	2	1	0
LCDCSS7	LCDCSS6	LCDCSS5	LCDCSS4	LCDCSS3	LCDCSS2	LCDCSS1	LCDCSS0
rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}

表 **14-19. LCDCSSEL0** 寄存器说明

位	字段	类型	复位	说明
15	LCDCSS15	RW	0h	选择引脚 L15 作为公共线或段线。 0b = 段线 1b = 公共线
14	LCDCSS14	RW	0h	选择引脚 L14 作为公共线或段线。 0b = 段线 1b = 公共线
13	LCDCSS13	RW	0h	选择引脚 L13 作为公共线或段线。 0b = 段线 1b = 公共线
12	LCDCSS12	RW	0h	选择引脚 L12 作为公共线或段线。 0b = 段线 1b = 公共线
11	LCDCSS11	RW	0h	选择引脚 L11 作为公共线或段线。 0b = 段线 1b = 公共线
10	LCDCSS10	RW	0h	选择引脚 L10 作为公共线或段线。 0b = 段线 1b = 公共线
9	LCDCSS9	RW	0h	选择引脚 L9 作为公共线或段线。 0b = 段线 1b = 公共线
8	LCDCSS8	RW	0h	选择引脚 L8 作为公共线或段线。 0b = 段线 1b = 公共线
7	LCDCSS7	RW	0h	选择引脚 L7 作为公共线或段线。 0b = 段线 1b = 公共线
6	LCDCSS6	RW	0h	选择引脚 L6 作为公共线或段线。 0b = 段线 1b = 公共线
5	LCDCSS5	RW	0h	选择引脚 L5 作为公共线或段线。 0b = 段线 1b = 公共线
4	LCDCSS4	RW	0h	选择引脚 L4 作为公共线或段线。 0b = 段线 1b = 公共线

表 **14-19. LCDCSSEL0** 寄存器说明 **(continued)**

位	字段	类型	复位	说明
3	LCDCSS3	RW	0h	选择引脚 L3 作为公共线或段线。 0b = 段线 1b = 公共线
2	LCDCSS2	RW	0h	选择引脚 L2 作为公共线或段线。 0b = 段线 1b = 公共线
1	LCDCSS1	RW	0h	选择引脚 L1 作为公共线或段线。 0b = 段线 1b = 公共线
0	LCDCSS0	RW	0h	选择引脚 L0 作为公共线或段线。 0b = 段线 1b = 公共线

14.3.11 LCDCSSEL1 寄存器

LCD_E COM/SEG 选择寄存器 1

图 **14-29. LCDCSSEL1** 寄存器

15	14	13	12	11	10	9	8
LCDCSS31	LCDCSS30	LCDCSS29	LCDCSS28	LCDCSS27	LCDCSS26	LCDCSS25	LCDCSS24
rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}
7	6	5	4	3	2	1	0
LCDCSS23	LCDCSS22	LCDCSS21	LCDCSS20	LCDCSS19	LCDCSS18	LCDCSS17	LCDCSS16
rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}

表 **14-20. LCDCSSEL1** 寄存器说明

位	字段	类型	复位	说明
15	LCDCSS31	RW	0h	选择引脚 L31 作为公共线或段线。 0b = 段线 1b = 公共线
14	LCDCSS30	RW	0h	选择引脚 L30 作为公共线或段线。 0b = 段线 1b = 公共线
13	LCDCSS29	RW	0h	选择引脚 L29 作为公共线或段线。 0b = 段线 1b = 公共线
12	LCDCSS28	RW	0h	选择引脚 L28 作为公共线或段线。 0b = 段线 1b = 公共线
11	LCDCSS27	RW	0h	选择引脚 L27 作为公共线或段线。 0b = 段线 1b = 公共线
10	LCDCSS26	RW	0h	选择引脚 L26 作为公共线或段线。 0b = 段线 1b = 公共线
9	LCDCSS25	RW	0h	选择引脚 L25 作为公共线或段线。 0b = 段线 1b = 公共线
8	LCDCSS24	RW	0h	选择引脚 L24 作为公共线或段线。 0b = 段线 1b = 公共线
7	LCDCSS23	RW	0h	选择引脚 L23 作为公共线或段线。 0b = 段线 1b = 公共线
6	LCDCSS22	RW	0h	选择引脚 L22 作为公共线或段线。 0b = 段线 1b = 公共线
5	LCDCSS21	RW	0h	选择引脚 L21 作为公共线或段线。 0b = 段线 1b = 公共线
4	LCDCSS20	RW	0h	选择引脚 L20 作为公共线或段线。 0b = 段线 1b = 公共线

表 **14-20. LCDCSSEL1** 寄存器说明 **(continued)**

位	字段	类型	复位	说明
3	LCDCSS19	RW	0h	选择引脚 L19 作为公共线或段线。 0b = 段线 1b = 公共线
2	LCDCSS18	RW	0h	选择引脚 L18 作为公共线或段线。 0b = 段线 1b = 公共线
1	LCDCSS17	RW	0h	选择引脚 L17 作为公共线或段线。 0b = 段线 1b = 公共线
0	LCDCSS16	RW	0h	选择引脚 L16 作为公共线或段线。 0b = 段线 1b = 公共线

14.3.12 LCDCSSEL2 寄存器

LCD_E COM/SEG 选择寄存器 0

图 **14-30. LCDCSSEL2** 寄存器

15	14	13	12	11	10	9	8
LCDCSS47	LCDCSS46	LCDCSS45	LCDCSS44	LCDCSS43	LCDCSS42	LCDCSS41	LCDCSS40
rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}
7	6	5	4	3	2	1	0
LCDCSS39	LCDCSS38	LCDCSS37	LCDCSS36	LCDCSS35	LCDCSS34	LCDCSS33	LCDCSS32
rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}

表 **14-21. LCDCSSEL2** 寄存器说明

位	字段	类型	复位	说明
15	LCDCSS47	RW	0h	选择引脚 L47 作为公共线或段线。 0b = 段线 1b = 公共线
14	LCDCSS46	RW	0h	选择引脚 L46 作为公共线或段线。 0b = 段线 1b = 公共线
13	LCDCSS45	RW	0h	选择引脚 L45 作为公共线或段线。 0b = 段线 1b = 公共线
12	LCDCSS44	RW	0h	选择引脚 L44 作为公共线或段线。 0b = 段线 1b = 公共线
11	LCDCSS43	RW	0h	选择引脚 L43 作为公共线或段线。 0b = 段线 1b = 公共线
10	LCDCSS42	RW	0h	选择引脚 L42 作为公共线或段线。 0b = 段线 1b = 公共线
9	LCDCSS41	RW	0h	选择引脚 L41 作为公共线或段线。 0b = 段线 1b = 公共线
8	LCDCSS40	RW	0h	选择引脚 L40 作为公共线或段线。 0b = 段线 1b = 公共线
7	LCDCSS39	RW	0h	选择引脚 L39 作为公共线或段线。 0b = 段线 1b = 公共线
6	LCDCSS38	RW	0h	选择引脚 L38 作为公共线或段线。 0b = 段线 1b = 公共线
5	LCDCSS37	RW	0h	选择引脚 L37 作为公共线或段线。 0b = 段线 1b = 公共线
4	LCDCSS36	RW	0h	选择引脚 L36 作为公共线或段线。 0b = 段线 1b = 公共线

表 **14-21. LCDCSSEL2** 寄存器说明 **(continued)**

位	字段	类型	复位	说明
3	LCDCSS35	RW	0h	选择引脚 L35 作为公共线或段线。 0b = 段线 1b = 公共线
2	LCDCSS34	RW	0h	选择引脚 L34 作为公共线或段线。 0b = 段线 1b = 公共线
1	LCDCSS33	RW	0h	选择引脚 L33 作为公共线或段线。 0b = 段线 1b = 公共线
0	LCDCSS32	RW	0h	选择引脚 L32 作为公共线或段线。 0b = 段线 1b = 公共线

14.3.13 LCDCSSEL3 寄存器

LCD_E COM/SEG 选择寄存器 0

图 **14-31. LCDCSSEL3** 寄存器

15	14	13	12	11	10	9	8
LCDCSS63	LCDCSS62	LCDCSS61	LCDCSS60	LCDCSS59	LCDCSS58	LCDCSS57	LCDCSS56
rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}
7	6	5	4	3	2	1	0
LCDCSS55	LCDCSS54	LCDCSS53	LCDCSS52	LCDCSS51	LCDCSS50	LCDCSS49	LCDCSS48
rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}

表 **14-22. LCDCSSEL3** 寄存器说明

位	字段	类型	复位	说明
15	LCDCSS63	RW	0h	选择引脚 L63 作为公共线或段线。 0b = 段线 1b = 公共线
14	LCDCSS62	RW	0h	选择引脚 L62 作为公共线或段线。 0b = 段线 1b = 公共线
13	LCDCSS61	RW	0h	选择引脚 L61 作为公共线或段线。 0b = 段线 1b = 公共线
12	LCDCSS60	RW	0h	选择引脚 L60 作为公共线或段线。 0b = 段线 1b = 公共线
11	LCDCSS59	RW	0h	选择引脚 L59 作为公共线或段线。 0b = 段线 1b = 公共线
10	LCDCSS58	RW	0h	选择引脚 L58 作为公共线或段线。 0b = 段线 1b = 公共线
9	LCDCSS57	RW	0h	选择引脚 L57 作为公共线或段线。 0b = 段线 1b = 公共线
8	LCDCSS56	RW	0h	选择引脚 L56 作为公共线或段线。 0b = 段线 1b = 公共线
7	LCDCSS55	RW	0h	选择引脚 L55 作为公共线或段线。 0b = 段线 1b = 公共线
6	LCDCSS54	RW	0h	选择引脚 L54 作为公共线或段线。 0b = 段线 1b = 公共线
5	LCDCSS53	RW	0h	选择引脚 L53 作为公共线或段线。 0b = 段线 1b = 公共线
4	LCDCSS52	RW	0h	选择引脚 L52 作为公共线或段线。 0b = 段线 1b = 公共线

表 **14-22. LCDCSSEL3** 寄存器说明 **(continued)**

位	字段	类型	复位	说明
3	LCDCSS51	RW	0h	选择引脚 L51 作为公共线或段线。 0b = 段线 1b = 公共线
2	LCDCSS50	RW	0h	选择引脚 L50 作为公共线或段线。 0b = 段线 1b = 公共线
1	LCDCSS49	RW	0h	选择引脚 L49 作为公共线或段线。 0b = 段线 1b = 公共线
0	LCDCSS48	RW	0h	选择引脚 L48 作为公共线或段线。 0b = 段线 1b = 公共线

14.3.14 LCDM[索引] 寄存器 – 静态、2 路复用、3 路复用、4 路复用模式

LCD_E 存储器 [索引] 寄存器

对于静态、2 路复用、3 路复用、4 路复用模式：**索引 = 0 到 31**

图 **14-32. LCDM[索引] 寄存器**

7	6	5	4	3	2	1	0
MBIT7	MBIT6	MBIT5	MBIT4	MBIT3	MBIT2	MBIT1	MBIT0
rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}

表 **14-23. LCDM[索引] 寄存器说明**

位	字段	类型	复位	说明
7	MBIT7	RW	0h	如果 LCD 引脚 L[2*索引+1] 被选为段线 (LCDCSS[2*索引+1] = 0b)，并且 LCD 复用率为 4 路复用 (LCDMXx=011b)： 0b = LCD 段关闭 1b = LCD 段打开 如果 LCD 引脚 L[2*索引+1] 被选为公共线 (LCDCSS[2*索引+1] = 1b)： 0b = 引脚 L[2*索引+1] 不用作 COM3 1b = 引脚 L[2*索引+1] 用作 COM3
6	MBIT6	RW	0h	如果 LCD 引脚 L[2*索引+1] 被选为段线 (LCDCSS[2*索引+1] = 0b)，并且 LCD 复用率为 3 路或 4 路复用 (010b <= LCDMXx <= 011b)： 0b = LCD 段关闭 1b = LCD 段打开 如果 LCD 引脚 L[2*索引+1] 被选为公共线 (LCDCSS[2*索引++1] = 1b)： 0b = 引脚 L[2*索引+1] 不用作 COM2 1b = 引脚 L[2*索引+1] 用作 COM2
5	MBIT5	RW	0h	如果 LCD 引脚 L[2*索引+1] 被选为段线 (LCDCSS[2*索引+1] = 0b)，并且 LCD 复用率为 2 路、3 路或 4 路复用 (001b <= LCDMXx <= 011b)： 0b = LCD 段关闭 1b = LCD 段打开 如果 LCD 引脚 L[2*索引+1] 被选为公共线 (LCDCSS[2*索引+1] = 1b)： 0b = 引脚 L[2*索引+1] 不用作 COM1 1b = 引脚 L[2*索引+1] 用作 COM1
4	MBIT4	RW	0h	如果 LCD 引脚 L[2*索引+1] 被选为段线 (LCDCSS[2*索引+1] = 0b)，并且 LCD 复用率为静态、2 路、3 路或 4 路复用 (000b <= LCDMXx <= 011b)： 0b = LCD 段关闭 1b = LCD 段打开 如果 LCD 引脚 L[2*索引+1] 被选为公共线 (LCDCSS[2*索引+1] = 1b)： 0b = 引脚 L[2*索引+1] 不用作 COM0 1b = 引脚 L[2*索引+1] 用作 COM0
3	MBIT3	RW	0h	如果 LCD 引脚 L[索引] 被选为段线 (LCDCSS[索引] = 0b)，并且 LCD 复用率为 4 路复用 (LCDMXx=011b)： 0b = LCD 段关闭 1b = LCD 段打开 如果 LCD 引脚 L[2*索引] 被选为公共线 (LCDCSS[2*索引] = 1b)： 0b = 引脚 L[2*索引] 不用作 COM3 1b = 引脚 L[2*索引] 用作 COM3
2	MBIT2	RW	0h	如果 LCD 引脚 L[2*索引] 被选为段线 (LCDCSS[2*索引] = 0b)，并且 LCD 复用率为 3 路或 4 路复用 (010b <= LCDMXx <= 011b)： 0b = LCD 段关闭 1b = LCD 段打开 如果 LCD 引脚 L[2*索引] 被选为公共线 (LCDCSS[2*索引] = 1b)： 0b = 引脚 L[2*索引] 不用作 COM2 1b = 引脚 L[2*索引] 用作 COM2

表 **14-23. LCDM[索引] 寄存器说明 (continued)**

位	字段	类型	复位	说明
1	MBIT1	RW	0h	如果 LCD 引脚 L[*2*index*] 被选为段线 (LCDCSS[2*索引] = 0b)，并且 LCD 复用率为 2 路、3 路或 4 路复用 (001b <= LCDMXx <= 011b)： 0b = LCD 段关闭 1b = LCD 段打开 如果 LCD 引脚 L[2*索引] 被选为公共线 (LCDCSS[2*索引] = 1b)： 0b = 引脚 L[2*索引] 不用作 COM1 1b = 引脚 L[2*索引] 用作 COM1
0	MBIT0	RW	0h	如果 LCD 引脚 L[2*索引] 被选为段线 (LCDCSS[2*索引] = 0b)，并且 LCD 复用率为 2 路、3 路或 4 路复用 (000b <= LCDMXx <= 011b)： 0b = LCD 段关闭 1b = LCD 段打开 如果 LCD 引脚 L[2*索引] 被选为公共线 (LCDCSS[2*索引] = 1b)： 0b = 引脚 L[2*索引] 不用作 COM0 1b = 引脚 L[2*索引] 用作 COM0

14.3.15 LCDM[索引] 寄存器 – 5 路复用、6 路复用、7 路复用、8 路复用模式

LCD_E 存储器 [索引] 寄存器

5 路复用、6 路复用、7 路复用、8 路复用模式：**索引 = 0 到 63**

图 **14-33. LCDM[索引] 寄存器**

7	6	5	4	3	2	1	0
MBIT7	MBIT6	MBIT5	MBIT4	MBIT3	MBIT2	MBIT1	MBIT0
rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}	rw-{0}

表 **14-24. LCDM[索引] 寄存器说明**

位	字段	类型	复位	说明
7	MBIT7	RW	0h	如果 LCD 引脚 L[索引] 被选为段线 (LCDCSS[索引] = 0b)，并且 LCD 复用率为 8 路复用 (LCDMXx = 111b)： 0b = LCD 段关闭 1b = LCD 段打开 如果 LCD 引脚 L[索引] 被选为公共线 (LCDCSS[索引] = 1b)： 0b = 引脚 L[索引] 不用作 COM7 1b = 引脚 L[索引] 用作 COM7
6	MBIT6	RW	0h	如果 LCD 引脚 L[索引] 被选为段线 (LCDCSS[索引] = 0b)，并且 LCD 复用率为 7 路或 8 路复用 (LCDMXx >= 110b)： 0b = LCD 段关闭 1b = LCD 段打开 如果 LCD 引脚 L[索引] 被选为公共线 (LCDCSS[索引] = 1b)： 0b = 引脚 L[索引] 不用作 COM6 1b = 引脚 L[索引] 用作 COM6
5	MBIT5	RW	0h	如果 LCD 引脚 L[索引] 被选为段线 (LCDCSS[索引] = 0b)，并且 LCD 复用率为 6 路、7 路或 8 路复用 (LCDMXx >= 101b)： 0b = LCD 段关闭 1b = LCD 段打开 如果 LCD 引脚 L[索引] 被选为公共线 (LCDCSS[索引] = 1b)： 0b = 引脚 L[索引] 不用作 COM5 1b = 引脚 L[索引] 用作 COM5
4	MBIT4	RW	0h	如果 LCD 引脚 L[索引] 被选为段线 (LCDCSS[索引] = 0b)，并且 LCD 复用率为 5 路、6 路、7 路或 8 路复用 (LCDMXx >= 100b)： 0b = LCD 段关闭 1b = LCD 段打开 如果 LCD 引脚 L[索引] 被选为公共线 (LCDCSS[索引] = 1b)： 0b = 引脚 L[索引] 不用作 COM4 1b = 引脚 L[索引] 用作 COM4
3	MBIT3	RW	0h	如果 LCD 引脚 L[索引] 被选为段线 (LCDCSS[索引] = 0b)，并且 LCD 复用率为 5 路、6 路、7 路或 8 路复用 (LCDMXx >= 100b)： 0b = LCD 段关闭 1b = LCD 段打开 如果 LCD 引脚 L[索引] 被选为公共线 (LCDCSS[索引] = 1b)： 0b = 引脚 L[索引] 不用作 COM3 1b = 引脚 L[索引] 用作 COM3
2	MBIT2	RW	0h	如果 LCD 引脚 L[索引] 被选为段线 (LCDCSS[索引] = 0b)，并且 LCD 复用率为 5 路、6 路、7 路或 8 路复用 (LCDMXx >= 100b)： 0b = LCD 段关闭 1b = LCD 段打开 如果 LCD 引脚 L[索引] 被选为公共线 (LCDCSS[索引] = 1b)： 0b = 引脚 L[索引] 不用作 COM2 1b = 引脚 L[索引-1] 用作 COM2

表 14-24. LCDM[索引] 寄存器说明 (continued)

位	字段	类型	复位	说明
1	MBIT1	RW	0h	如果 LCD 引脚 L[索引] 被选为段线 (LCDCSS[索引] = 0b)，并且 LCD 复用率为 5 路、6 路、7 路或 8 路复用 (LCDMXx >= 100b)： 0b = LCD 段关闭 1b = LCD 段打开 如果 LCD 引脚 L[索引] 被选为公共线 (LCDCSS[索引] = 1b)： 0b = 引脚 L[索引] 不用作 COM1 1b = 引脚 L[索引] 用作 COM1
0	MBIT0	RW	0h	如果 LCD 引脚 L[索引] 被选为段线 (LCDCSS[索引] = 0b)，并且 LCD 复用率为 5 路、6 路、7 路或 8 路复用 (LCDMXx >= 100b)： 0b = LCD 段关闭 1b = LCD 段打开 如果 LCD 引脚 L[索引] 被选为公共线 (LCDCSS[索引] = 1b)： 0b = 引脚 L[索引] 不用作 COM0 1b = 引脚 L[索引] 用作 COM0

14.3.16 LCDIV 寄存器

LCD_E 中断向量寄存器

图 **14-34. LCDIV** 寄存器

15	14	13	12	11	10	9	8
LCDIVx							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
LCDIVx							
r0	r0	r0	r0	r0	r0	r0	r0

表 **14-25. LCDIV** 寄存器说明

位	字段	类型	复位	说明
15-0	LCDIVx	R	0h	LCD_E 中断向量值 00h = 无中断挂起 04h = 中断源：闪烁，段关闭；中断标志： LCDBLKOFFIFG；中断优先级：最高 06h = 中断源：闪烁，段打开；中断标志LCDBLKONIFG 08h = 中断源：帧中断；中断标志：LCDFRMIFG；中断优先级：最低

增强型通用串行通信接口 (eUSCI) – UART 模式

增强型通用串行通信接口 A (eUSCI_A) 通过一个硬件模块支持多种串行通信模式。本章对异步 UART 模式的操作进行了说明。

Topic **Page**

15.1 增强型通用串行通信接口 A (eUSCI_A) 概述

eUSCI_A 模块支持两个串行通信模式：

- UART 模式
- SPI 模式

15.2 eUSCI_A 介绍– UART 模式

在异步模式下，器件通过 eUSCI_Ax 模块的两个外部引脚（UCAxRXD 和 UCAxTXD）与外部系统相连。当 UCSYNC 位被清零时就选择了 UART 模式。

UART 模式的特性包括：

- 支持奇校验、偶校验或无奇偶校验的 7 位或 8 位数据
- 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
- 接收器启动边沿检测，用于从 LPMx 模式自动唤醒（不支持从 LPMx.5 唤醒）
- 用于支持分数波特率的可编程波特率
- 状态标志的错误检测和抑制
- 地址检测的状态标志
- 针对接收、发送、起始位已接收和发送完成的独立中断能力

图 15-1显示了配置为 UART 模式时的 USCI_Ax。

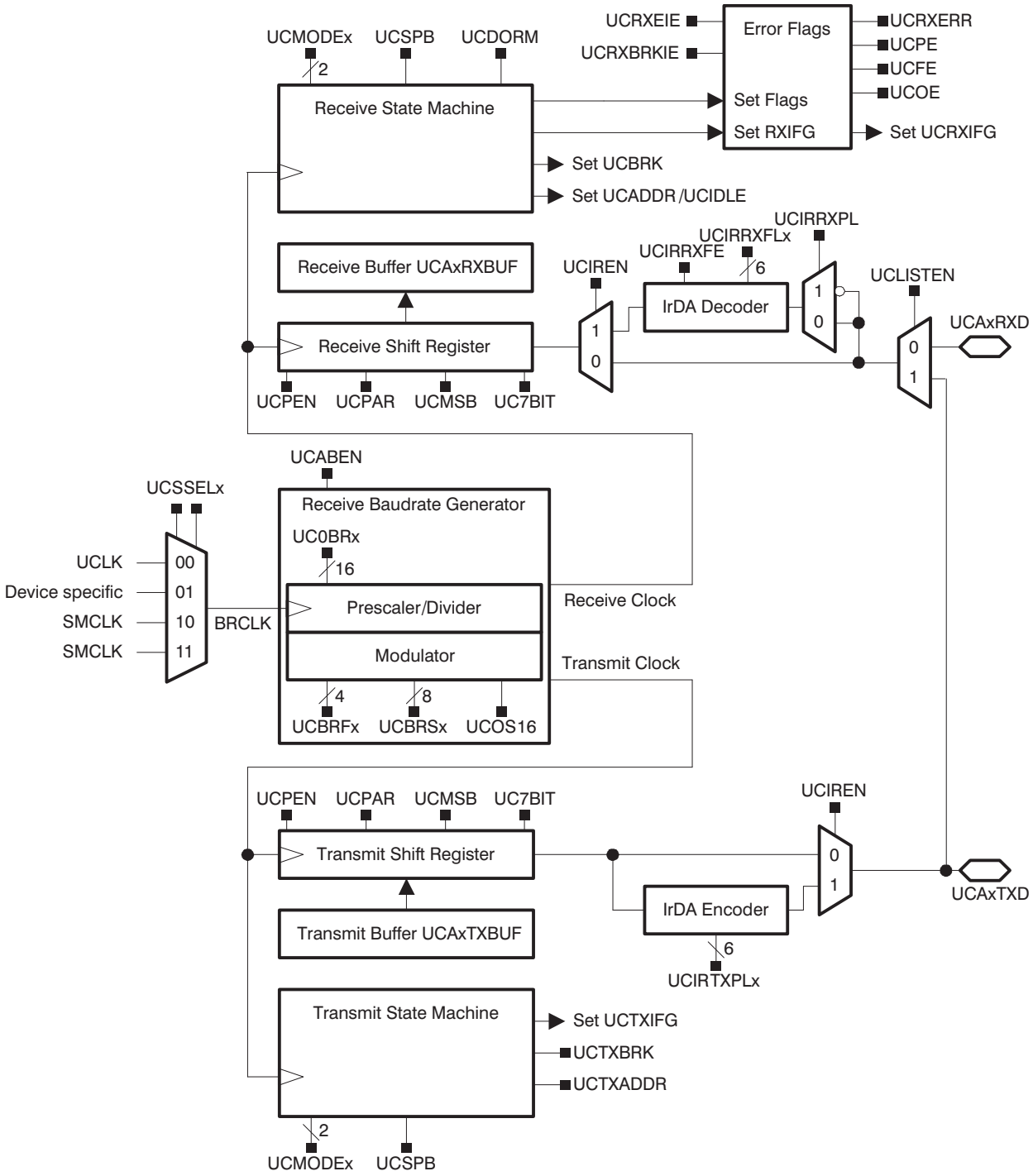


图 **15-1. eUSCI_Ax 框图：UART 模式 (UCSYNC= 0)**

15.3 eUSCI_A 运行 - UART 模式

在 UART 模式下，eUSCI_A 发送和接收字符时的比特率与另一器件不同步。每个字符的时序基于 eUSCI_A 的所选波特率。传输和接收功能使用相同的波特率。

15.3.1 eUSCI_A 初始化和复位

eUSCI_A 由一个 PUC 后或者通过设置 UCSWRST 位来复位。一个 PUC 后，UCSWRST 位会自动置 1，以此来将 eUSCI_A 保持在复位状态。 置 1 时，UCSWRST 位将 UCTXIFG 置 1 并将 UCRXIE，UCTXIE，UCRXIFG，UCRXERR，UCBRK，UCPE，UCOE，UCFE，UCSTOE 和 UCBTOE 位复位。UCSWRST 位清零会使 eUSCI_A 处于运行状态。

为避免发生无法预测的行为，应在 UCSWRST 置 1 时配置或重新配置 eUSCI_A 模块。

注： **初始化或重新配置 eUSCI_A 模块**

建议的 eUSCI_A 初始化/重配置过程为：

1. 将 UCSWRST 置 1 (`BIS.B #UCSWRST,&UCAxCTL1`)。
2. 在 UCSWRST = 1 时初始化所有 eUSCI_A 寄存器（包括 UCAxCTL1）。
3. 配置端口。
4. 通过软件将 UCSWRST 清零 (`BIC.B #UCSWRST,&UCAxCTL1`)。
5. 通过 UCRXIE 或 UCTXIE 使能中断（可选）。

15.3.2 字符格式

UART 的字符格式（请见图 15-2）包括一个开始位，7 或 8 个数据位，一个奇/偶/无奇偶校验位，一个地址位（地址位模式），和一个或两个停止位。UCMSB 位控制传输方向并选定最低有效位 (LSB) 或最高有效位 (MSB) 先发送或接收。UART 通信通常要求 LSB 优先发送。

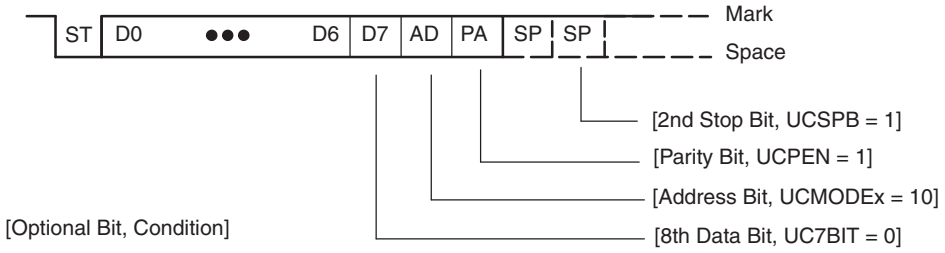


图 15-2. 字符格式

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，eUSCI_A 支持空闲线路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路上被一段空闲时间隔开（请见如图 15-3）。在一个字符的一个或两个停止位后，当接收到 10 个或更多的连续数据块（标志）时，一条空闲接收线路会被检测到。在接收到一条空闲线后，在下一次开始边沿被检测到前波特率发生器被切断。当检测到一条空闲线路时，UCIDLE 被置 1。

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 1。

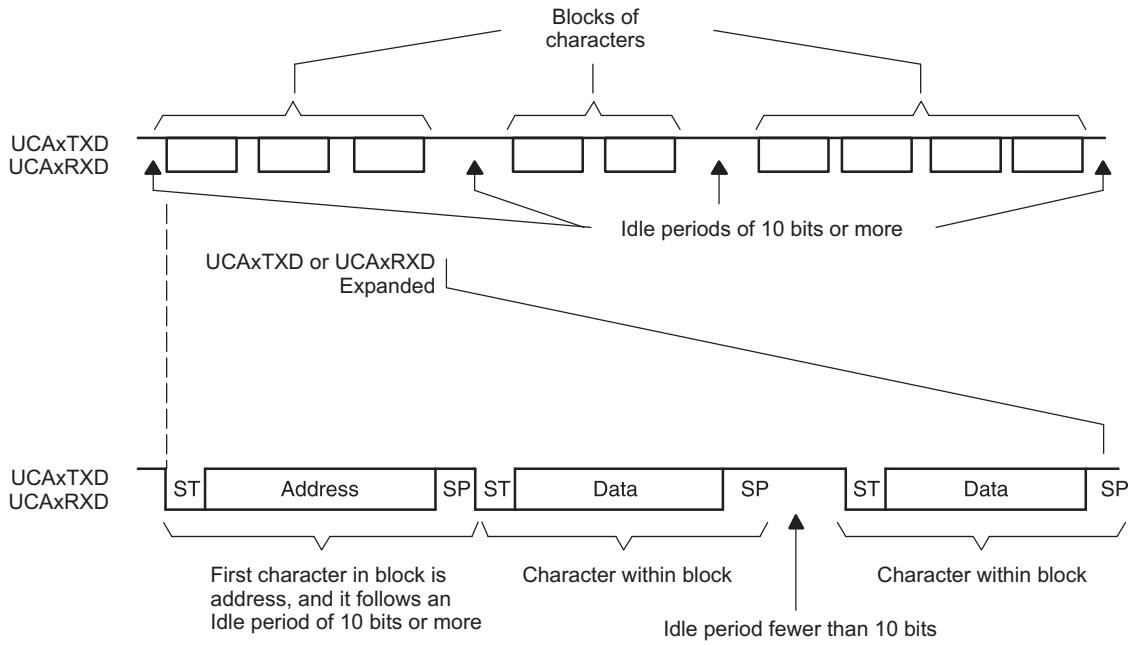


图 **15-3.** 空闲线格式

在多处理器模式下的 UCDORM 位被用于控制数据接收。当 UCDORM = 1 时，所有非地址字符均将被汇编但不会传送至 UCAxRXBUF，并且不会有中断生成。当接收到地址字符时，该字符会被传送到 UCAxRXBUF，UCRXIFG 置 1，并且当 UCRXEIE= 1 时，任何适用的错误标志均置 1。当 UCRXEIE = 0 并且接收到地址字符，但存在帧错误或奇偶校验错误时，字符不会传送至 UCAxRXBUF 并且 UCRXIFG 不会置 1。

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 仍然置 1，那么只有地址字符才能被接收。如果 UCDORM 在接收字符的过程中清零，接收中断标志会在接收完成后置 1。UCDORM 位不是由 eUSCI_A 硬件自动修改。

对在空闲线多处理器模式下发送地址而言，eUSCI_A 会产生一个精准的空闲周期以在 UCAxTXD 上产生地址字符标识符。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。

15.3.3.1.1 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 1 UCTXADDR，然后把地址字符写入 UCAxTXBUF。UCAxTXBUF 必须为新数据做好准备 (UCTXIFG = 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. 向 UCAxTXBUF 中写入所需数据。UCAxTXBUF 必须为新数据做好准备 (UCTXIFG = 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。

15.3.3.2 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。每个已处理的字符包含一个用作地址标识符的额外位（请见图 15-4）。字符块的第一个字符带有一个表示字符是一个地址的置 1 地址位。当一个接收到的字符的地址位被置 1 并被传递到 UCAxRXBUF 时，eUSCI_A UCADDR 位被置 1。

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 1，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。当接收到一个包含置 1 地址位的字符时，该字符被传递到 UCAxRXBUF，UCRXIFG 被置 1，且当 UCRXEIE= 1 时，任何可用的错误标志都被置 1。当 UCRXEIE = 0 并且接收到一个包含置 1 地址位的字符，但该字符发生了组帧错误或奇偶错误时，字符不会被移送到 UCAxRXBUF，UCRXIFG 也不会置 1。

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 1，将只能接收地址位 = 1 的地址字符。UCDORM 位不会被 eUSCI_A 硬件自动修改。

当 UCDORM = 0 时，所有已接收的字符将置 1 中断标志 UCRXIFG。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 1。

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。

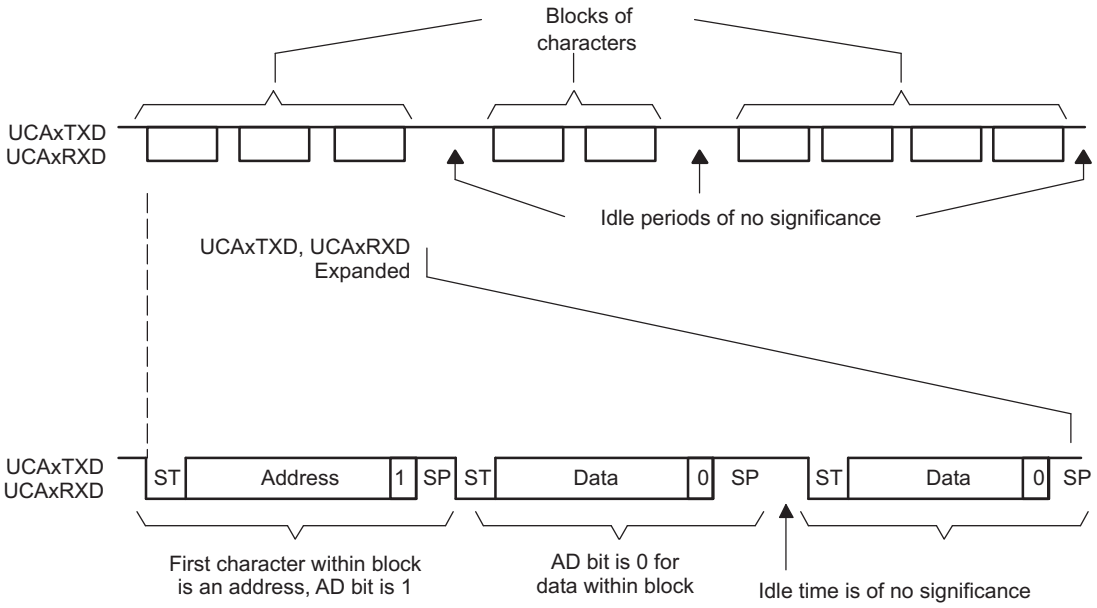


图 15-4. 地址位多处理器格式

15.3.3.2.1 中断接收和生成

当 UCMODEX = 00，01 或 10 时，无论奇偶校验，地址模式或其它字符设置，当所有数据，奇偶校验，停止位都为低电平时，接收器在检测到一次中断。当监测到一次中断时，UCBRK 位置 1。如果中断的中断使能位 (UCBRKIE) 被置 1，接收中断标志 UCRXIFG 也将被置 1。在这种情况下，因为所有数据位是 0 所以 UCAxRXBUF 中的值是 0h。

为了发送一个中断，将 UCTXBRK 位置 1，然后把 0h 写到 UCAxTXBUF 中。UCAxTXBUF 必须为新数据做好准备 (UCTXIFG = 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，在数据帧前面有一个包含一个中断和一个同步域的同步序列。当 11 个或更多的连续 0（空格）被接收到时会检测到一个中断。如果中断长度超过 21 位的时间，中断超时错误标志 UCBTOE 将被置 1。eUSCI_A 不能在接收到中断/同步字段时发送数据。中断后的同步字段如图 15-5中所示。

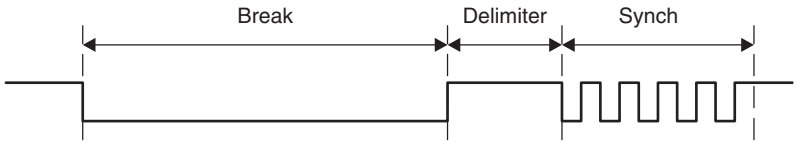


图 **15-5.** 自动波特率检测-中断/同步序列

为了 LIN 一致性，字符格式应该设置为 8 个数据位，LSB 优先，无奇偶校验位和一个停止位。没有可用的地址位。

在一个字节字段内的同步字段包含数据 055h（请见图 15-6）。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。如果自动波特率检测通过置 1 UCABDEN 启用，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。测量的结果被传递给波特率控制寄存器（UCAxBRW 和 UCAxMCTLW）。如果同步字段的长度超过了可测量时间，同步超时错误标志 UCSTOE 将被置 1。此结果可在接收中断标志 UCRXIFG 被置 1 后读取。

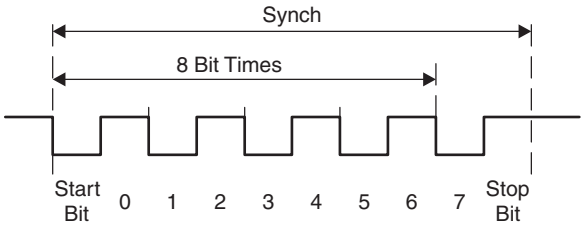


图 **15-6.** 自动波特率检测 - 同步字段

在这种模式中 UCDORM 位被用来控制数据接收。当 UCDORM 被置 1 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/同步字段被检测到时，UCBRK 标志被置 1。中断/同步字段后的字符被传递到 UCAxRXBUF 中且 UCRXIFG 中断标志被置 1。任何适用的错误标志也被置 1。如果 UCBRKIE 位被置 1，中断/同步的接收会将 UCRXIFG 置 1。通过读取接收缓存 UCAxRXBUF，或通过用户软件来将 UCBRK 位置 1。

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 1 UCDORM。如果 UCDORM 保持置 1 状态，只有在接受下一个中断/同步字段后字符才能被接收。UCDORM 位不会被 eUSCI_A 硬件自动修改。

当 UCDORM = 0 时，所有已接收的字符将置 1 中断标志 UCRXIFG。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 1。

此计数器用于检测波特率被限制在 0FFFFh (2^{16}) 计数。这意味着在过采样模式中可检测到的最小波特率为 244 波特，在低频模式中为 15 波特。可检测的最高波特率为 1Mbaud。

可以在一个带有某些限制的全双工通信系统中使用自动波特率检测模式。eUSCI_A 可以在接收中断/同步字段期间不发送数据，并且如果一个组帧错误字节 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：

1. 在 UMODEX=11 时，将 UCTXBRK 置 1。
2. 把 055h 写入 UCAxTXBUF。UCAxTXBUF 必须为新数据做好准备 (UCTXIFG = 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。
3. 向 UCAxTXBUF 中写入所需数据。UCAxTXBUF 必须为新数据做好准备 (UCTXIFG = 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 1 时，IrDA 解码器和译码器被启用并提供修整 IrDA 通信的硬件位。

15.3.5.1 IrDA 编码

在来自 UART 的发送位流中编码器为每个 0 位发送一个脉冲（请见图 15-7）。脉冲持续时间由 UCIRTXPLx 位决定，此位用来指定被 UCIRTXCLK 选中的半时钟周期的数目。

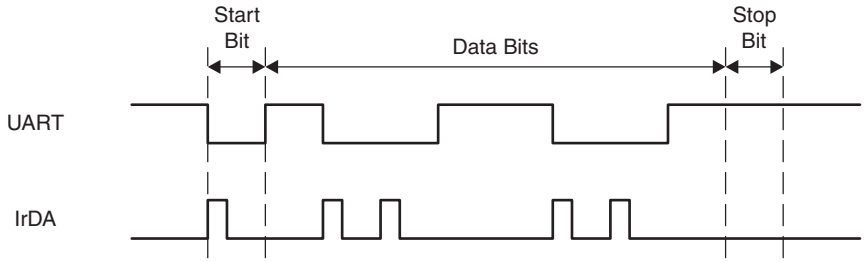


图 **15-7. UART 与 IrDA 数据格式间的关系**

为了按 IrDA 标准的要求去设置 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 来设置为六个半时钟周期。

当 UCIRTXCLK = 0 时，脉冲宽度 $t_{脉冲}$基于 BRCLK 并由以下公式计算：

$$UCIRTXPLx = t_{PULSE} \times 2 \times f_{BRCLK} - 1$$

当 UCIRTXCLK = 0时，预分频值 UCBRx 必须被设定为一个大于 6 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级也能通过设置 UCIRRXFE 来是启用。当 UCIRRXFE 被置 1 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_{PULSE}=最小接受脉冲宽度

t_{WAKE}=从任何低功耗模式中唤醒。当器件处于激活模式时为零。

15.3.6 自动错误检测

尖峰脉冲抑制防止 eUSCI_A 被意外启动。UCAxRXD 上任何短于去尖峰脉冲时间 t_t（由 UCGLITx 选中）的脉冲将被忽略（参数请参见具体器件的数据表）。

当在 UCAxRXD 上的一个低电平周期超过 t_t时，多数票决都会被当作开始位。如果多数票决没有检测到一个有效的开始位，eUSCI_A 将暂停字符接收并等待 UCAxRXD 上的下一个低电平周期。多数表决也可用于字符中的每个位，以防止位错误。

当接收字符时，eUSCI_A 模块自动检测组帧错误、奇偶校验错误、溢出错误，以及中断条件。在它们各自的条件被监测到时，UCFE，UCPE，UCOE 以及 UCBRK 位被置 1。当错误标志 UCFE，UCPE 或 UCOE 被置 1 时，UCRXERR 也被置 1。错误条件在表 15-1中做出描述。

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 1。
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 1。
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 1。
中断条件	UCBRK	当不使用自动波特率检测时，在所有数据、奇偶校验和停止位为低电平时，检测到一个中断。当检测到一次中断条件时，UCBRK 位置 1。如果中断使能 UCBRKIE 位被置 1，一个中断条件也可以将中断标志 UCRXIFG 置 1。

当 UCRXEIE = 0 时且检测到一个组帧错误，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 1。

当 UCFE，UCPE，UCOE，UCBRK 或 UCRXEER 中的任何一个被置 1 时，该位一直保持被置 1状态，直到用户用软件将其复位或 UCAxRXBUF 将其读取。UCOE 必须通过读取 UCAxRXBUF 复位。否则，它不能正常工作。为了可靠地检测溢出情况，建议使用以下流程。在接收到字符且 UCAxRXIFG 置 1 后，首先读取 UCAxSTATW 以检查错误标志（包括溢出标志 UCOE）。然后读取 UCAxRXBUF。如果在对 UCAxSTATW 和 UCAxRXBUF 的两次读访问之间 UCAxRXBUF 被覆盖，则该读操作将清零除 UCOE 以外的所有错误标志。因此，为了检测这个条件，在读完 UCAxRXBUF 后应该检查 UCOE 标志。请注意，在这种情况下，UCRXERR 标志不会被置 1。

15.3.7 eUSCI_A 接收使能

通过清零 UCSWRST 位可以启用 eUSCI_A 模块，接收器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检测到，一个字符就能被接收。

UCMODEx = 01 时，当空闲线路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 1，同时 UART 状态机返回到空闲态且波特率发生器被关闭。

15.3.7.1 接收数据尖峰脉冲抑制

尖峰脉冲抑制防止 eUSCI_A 被意外启动。UCAxRCD 上任何短于去尖峰脉冲时间 t_t的尖峰脉冲将被 eUSCI_A 忽略，而图 15-8中显示了采取的进一步操作（参数请参见具体器件的数据表）。使用 UCGLITx 位可将去尖峰脉冲时间 t_t设定为四个不同的值。

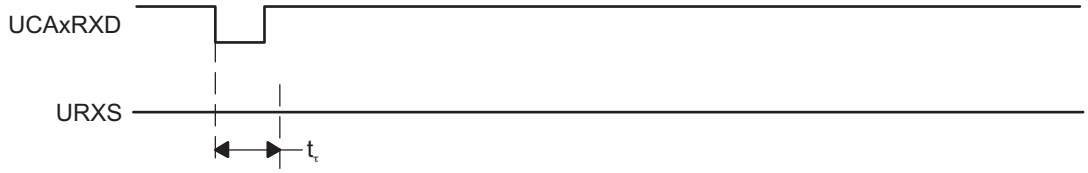


图 15-8. 尖峰脉冲抑制，eUSCI_A 接收不启动

当一个尖峰脉冲时间大于 t_t或一个有效的开始位出现在 UCAxRXD 上时，eUSCI_A 接收操作开始并采取多数票决（请见图 15-9）。如果多数票决未能检测到一个起始位，eUSCI_A 暂停字符接收。

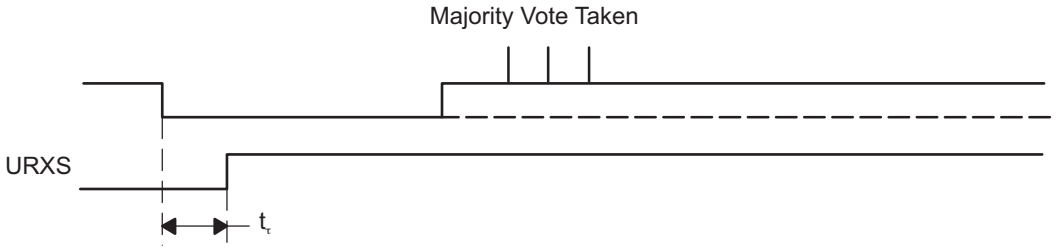


图 15-9. 尖峰脉冲抑制，eUSCI_A 被激活

15.3.8 eUSCI_A 发送使能

通过清零 UCSWRST 位可以启用 eUSCI_A 模块，发送器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 中来启动一次传输。当这发生后，波特率发生器被启用，在发送移位寄存器为空后，在下一个BITCLK 后，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 UCTXIFG 被置 1。

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闭。

15.3.9 UART 波特率发生器

eUSCI_A 波特率发生器，能够从非标准源频率中产生标准的波特率。它通过 UCOS16 位提供了两种操作模式。

15.3.10 节介绍了找出 eUSCI_A 正确波特率设置的快速设置方法。

15.3.9.1 低频波特率生成

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，从一个 32768HZ 晶振产生一个 9600 波特率）。通过使用一个较低的输入频率，能减少模块的能耗。使用这种具有更高频率和更高预分频器值设置的模式将导致在一个不断增加的小窗口中采用多数票决，从而降低了多数票决的优势。

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 eUSCI_A 波特率是 UART 时钟源频率 BRCLK 的 1/3。

每个位的时序如图 15-10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。这些采样发生在 N/2-1/2，N/2 和 N/2+ 1/2 BRCLK 周期上，其中 N 是每个 BITCLK 周期中 BRCLK 的数目。

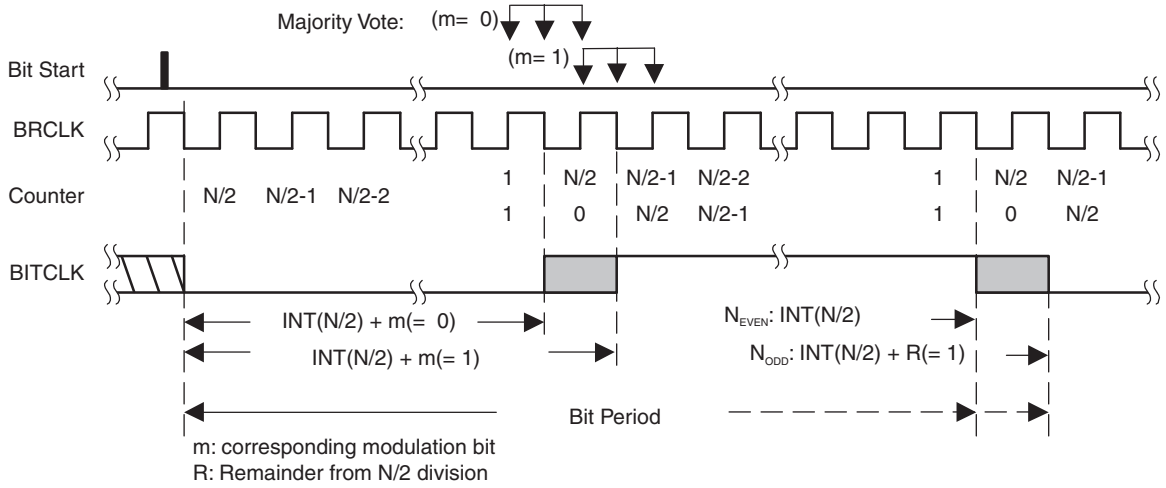


图 15-10. UCOS16 = 0 时的 BITCLK 波特率

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. 调制模式示例

UCBRSx	位 0（开始位）	位 1	位 2	位 3	位 4	位 5	位 6	位 7
0x00	0	0	0	0	0	0	0	0
0x01	0	0	0	0	0	0	0	1
				⋮				
0x35	0	0	1	1	0	1	0	1
0x36	0	0	1	1	0	1	1	0
0x37	0	0	1	1	0	1	1	1
				⋮				
0xff	1	1	1	1	1	1	1	1

UCBRSx 的正确设置如15.3.10 节中所述。

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 编码器和解码器被启用时，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 16 分频并且调制器级从 BITCLK16 中产生 BITCLK。这个组合支持针对 BITCLK16 和用于波特率生成的 BITCLK 的分数分频。在这种模式下，最大 eUSCI_A 波特率是 UART 源时钟频率 BRCLK 的 1/16。

BITCLK16 的调制是基于 UCBRFx 设置的（请见表 15-3）。表中 A 1 表示相应的 BITCLK16 是一个比 m=0 的周期长的 BRCLK 周期。用每一个新的位时序重新启动调制。

如前所述，BITCLK 调制是基于 UCBRSx 设置的。

表 15-3. BITCLK 的调制模式

UCBRFx	最后一个 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：

$N = f_{BRCLK} / 波特率$

分频因子 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级来尽可能的符合该因子。

如果 N 等于或大于 16，建议通过置 1 UCOS16 来使用过采样波特率生成模式。

注： **快速设置波特率**

要为生成波特率计算正确的波特率设置，请按以下步骤操作：

1. 计算 $N = f_{BRCLK} / 波特率$ [如果 N > 16，继续执行步骤 3；否则执行步骤 2]
2. OS16 = 0, UCBRx = INT(N) [跳到第 4 步]
3. OS16 = 1, UCBRx = INT(N/16), UCBRFx = INT([(N/16) – INT(N/16)] × 16)
4. 通过在表表 15-4中查找 N (= N - INT(N)) 的分小数部分可确定 UCBRSx。
5. 如果 OS16 = 0，建议执行一个详细的误差计算。

表 15-4可被用作检查表来为相应的 N 值小数部分寻找正确的 UCBRSx 调制模式。这些值已针对传输进行了优化。

表 15-4. N 的小数部分与 UCBRSx 设置的对应关系（$N = f_{BRCLK} /$ 波特率）

N 的小数部分	UCBRSx[1]	N 的小数部分	UCBRSx[1]
0.0000	0x00	0.5002	0xAA
0.0529	0x01	0.5715	0x6B
0.0715	0x02	0.6003	0xAD
0.0835	0x04	0.6254	0xB5
0.1001	0x08	0.6432	0xB6
0.1252	0x10	0.6667	0xD6
0.1430	0x20	0.7001	0xB7
0.1670	0x11	0.7147	0xBB
0.2147	0x21	0.7503	0xDD
0.2224	0x22	0.7861	0xED
0.2503	0x44	0.8004	0xEE
0.3000	0x25	0.8333	0xBF
0.3335	0x49	0.8464	0xDF
0.3575	0x4A	0.8572	0xEF
0.3753	0x52	0.8751	0xF7
0.4003	0x92	0.9004	0xFB
0.4286	0x53	0.9170	0xFD
0.4378	0x55	0.9288	0xFE

(1) 一行中的 UCBRSx 设置在采用那一行中给出的小数部分和下一行中的小数部分时有效

15.3.10.1 低频波特率模式设置

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)

小数部分由调制器的 UCBRSx 设置实现。确定正确 UCBRSx 的建议的方法是执行详细的误差计算，如以下部分所述。然而，也可在具有典型晶振的表中查找正确的设置（请见表 15-5）。

15.3.10.2 过采样波特率模式设置

在过采样模式中，预分频器被设置为：

UCBRx = INT(N/16)

且第一级调制器被设置为：

UCBRFx = INT([(N/16) – INT(N/16)] × 16)

可通过执行详细的误差计算或者使用表 15-4 和 N 的小数部分（N = f_{BRCLK} / 波特率）找出第二级调制设置 (UCBRSx)。

15.3.11 发送位时序-误差计算

每个字符的时序是单独的位时序的总和。使用波特率发生器的调制特性可以减少累积的位误差。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

在低频模式下，根据 UCBRx 和 UCBRSx 设置计算 bit i 的长度 $t_{bit,TX}[i]$：

$t_{bit,TX}[i] = (1/f_{BRCLK})(UCBRx + m_{UCBRSx}[i])$

其中：

$m_{UCBRSx}[i]$ = UCBRSx 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。

$$t_{bit,TX}[i] = \frac{1}{f_{BRCLK}}\left((16 \times UCBRx) + \sum_{j=0}^{15} m_{UCBRFx}[j] + m_{UCBRSx}[i]\right)$$

其中：

$\sum_{j=0}^{15} m_{UCBRFx}[j]$ =表 15-3中相应行的总和

$m_{UCBRSx}[i]$ = UCBRSx 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 $t_{位，理想，TX}[i]$：

$t_{bit,ideal,TX}[i]$ = (1 / 波特率)(i + 1)

这样可将误差相对一个理想位时间（1 / 波特率）进行归一化：

$Error_{TX}[i] = (t_{bit,TX}[i] – t_{bit,ideal,TX}[i])$ × 波特率 × 100%

15.3.12 接收位时序 – 误差计算

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起始沿和正在被 eUCSI_A 模块接受的起始沿之间的误差。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。同步误差 t_{SYNC} 介于 –0.5 BRCLK 和 +0.5 BRCLK 之间，与所选波特率生成模式无关。

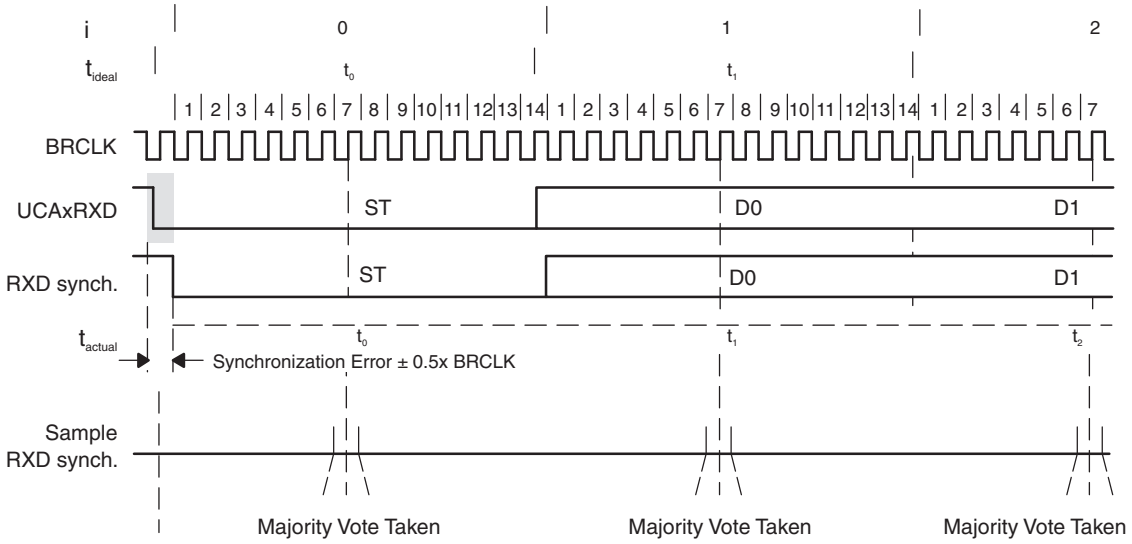


图 **15-11.** 接收错误

理想采样时间 $t_{位，理想值，RX}[i]$ 位于一个位周期的中间：

$t_{bit,ideal,RX}[i] = (1 / 波特率)(i + 0.5)$

真实采样时间，$t_{位，RX}[i]$，等于之前所有位的总和，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_{SYNC}。

这导致低频波特率模式中的下列 $t_{位，RX}[i]$ 值：

$$t_{bit,RX}[i] = t_{SYNC} + \sum_{j=0}^{i-1} T_{bit,RX}[j] + \frac{1}{f_{BRCLK}} \left(INT(½UCBRx) + m_{UCBRSx}[i] \right)$$

其中：

$t_{bit,RX}[i] = (1/f_{BRCLK})(UCBRx + m_{UCBRSx}[i])$

$m_{UCBRSx}[i]$ = UCBRSx 中位 i 的调制

对于过采样波特率模式，位 i 的采样时间 $t_{位，RX}[i]$ 计算方法如下：

$$t_{bit,RX}[i] = t_{SYNC} + \sum_{j=0}^{i-1} T_{bit,RX}[j] + \frac{1}{f_{BRCLK}} \left((8 * UCBRx) + \sum_{j=0}^{7} m_{UCBRFx}[j] + m_{UCBRSx}[i] \right)$$

其中：

$$t_{bit,RX}[i] = \frac{1}{f_{BRCLK}} \left((16 \times UCBRx) + \sum_{j=0}^{15} m_{UCBRFx}[j] + m_{UCBRSx}[i] \right)$$

$\sum_{j=0}^{7 + m_{UCBRSx}[i]} m_{UCBRFx}[j]$ = 从 0 至 $(7 + m_{UCBRSx}[i])$（来自表 15-3中的相应列）中某些列的总和。

$m_{UCBRSx}[i]$ = UCBRSx 中位 i 的调制

这样可根据如下公式将误差相对一个理想位时间（1 / 波特率）进行归一化：

$Error_{RX}[i] = (t_{bit,RX}[i] – t_{bit,ideal,RX}[i]) \times 波特率 \times 100\%$

15.3.13 典型的波特率和误差

对于一个为 ACLK 和典型的 SMCLK 频率供源的 32768HZ 晶振，在表 15-5和中列出了针对 UCBRSx 和 UCBRFx 的标准波特率数据。请确保所选择的 BRCLK 频率不要超过器件专用最大 eUSCI_A 输入频率（请参见具体器件的数据表）。

接收误差是在每一个位中间相对于理想扫描时间的累积时间。给出了针对带奇偶校验的 8 位字符的接收和包括同步误差的停止位的最差情况下的误差。

发送误差是相对于位周期理想时间的累积时序误差。针对带奇偶校验的 8 位字符的发送和停止位给出了最差情况下的误差。

表 **15-5.** 典型晶振和波特率的建议设置

BRCLK	波特率	UCOS16	UCBRx	UCBRFx	UCBRSx	TX 误差 (%)		RX 误差 (%)	
						负	正	负	正
32768	1200	1	1	11	0x25	-2.29	2.25	-2.56	5.35
32768	2400	0	13	-	0xB6	-3.12	3.91	-5.52	8.84
32768	4800	0	6	-	0xEE	-7.62	8.98	-21	10.25
32768	9600	0	3	-	0x92	-17.19	16.02	-23.24	37.3
1000000	9600	1	6	8	0x20	-0.48	0.64	-1.04	1.04
1000000	19200	1	3	4	0x2	-0.8	0.96	-1.84	1.84
1000000	38400	1	1	10	0x0	0	1.76	0	3.44
1000000	57600	0	17	-	0x4A	-2.72	2.56	-3.76	7.28
1000000	115200	0	8	-	0xD6	-7.36	5.6	-17.04	6.96
1048576	9600	1	6	13	0x22	-0.46	0.42	-0.48	1.23
1048576	19200	1	3	6	0xAD	-0.88	0.83	-2.36	1.18
1048576	38400	1	1	11	0x25	-2.29	2.25	-2.56	5.35
1048576	57600	0	18	-	0x11	-2	3.37	-5.31	5.55
1048576	115200	0	9	-	0x08	-5.37	4.49	-5.93	14.92
4000000	9600	1	26	0	0xB6	-0.08	0.16	-0.28	0.2
4000000	19200	1	13	0	0x84	-0.32	0.32	-0.64	0.48
4000000	38400	1	6	8	0x20	-0.48	0.64	-1.04	1.04
4000000	57600	1	4	5	0x55	-0.8	0.64	-1.12	1.76
4000000	115200	1	2	2	0xBB	-1.44	1.28	-3.92	1.68
4000000	230400	0	17	-	0x4A	-2.72	2.56	-3.76	7.28
4194304	9600	1	27	4	0xFB	-0.11	0.1	-0.33	0
4194304	19200	1	13	10	0x55	-0.21	0.21	-0.55	0.33
4194304	38400	1	6	13	0x22	-0.46	0.42	-0.48	1.23
4194304	57600	1	4	8	0xEE	-0.75	0.74	-2	0.87
4194304	115200	1	2	4	0x92	-1.62	1.37	-3.56	2.06
4194304	230400	0	18	-	0x11	-2	3.37	-5.31	5.55
8000000	9600	1	52	1	0x49	-0.08	0.04	-0.1	0.14
8000000	19200	1	26	0	0xB6	-0.08	0.16	-0.28	0.2
8000000	38400	1	13	0	0x84	-0.32	0.32	-0.64	0.48
8000000	57600	1	8	10	0xF7	-0.32	0.32	-1	0.36
8000000	115200	1	4	5	0x55	-0.8	0.64	-1.12	1.76
8000000	230400	1	2	2	0xBB	-1.44	1.28	-3.92	1.68
8000000	460800	0	17	-	0x4A	-2.72	2.56	-3.76	7.28
8388608	9600	1	54	9	0xEE	-0.06	0.06	-0.11	0.13
8388608	19200	1	27	4	0xFB	-0.11	0.1	-0.33	0
8388608	38400	1	13	10	0x55	-0.21	0.21	-0.55	0.33

表 **15-5.** 典型晶振和波特率的建议设置 **(continued)**

BRCLK	波特率	UCOS16	UCBRx	UCBRFx	UCBRSx	TX 误差 (%)		RX 误差 (%)	
						负	正	负	正
8388608	57600	1	9	1	0xB5	-0.31	0.31	-0.53	0.78
8388608	115200	1	4	8	0xEE	-0.75	0.74	-2	0.87
8388608	230400	1	2	4	0x92	-1.62	1.37	-3.56	2.06
8388608	460800	0	18	-	0x11	-2	3.37	-5.31	5.55
12000000	9600	1	78	2	0x0	0	0	0	0.04
12000000	19200	1	39	1	0x0	0	0	0	0.16
12000000	38400	1	19	8	0x65	-0.16	0.16	-0.4	0.24
12000000	57600	1	13	0	0x25	-0.16	0.32	-0.48	0.48
12000000	115200	1	6	8	0x20	-0.48	0.64	-1.04	1.04
12000000	230400	1	3	4	0x2	-0.8	0.96	-1.84	1.84
12000000	460800	1	1	10	0x0	0	1.76	0	3.44
16000000	9600	1	104	2	0xD6	-0.04	0.02	-0.09	0.03
16000000	19200	1	52	1	0x49	-0.08	0.04	-0.1	0.14
16000000	38400	1	26	0	0xB6	-0.08	0.16	-0.28	0.2
16000000	57600	1	17	5	0xDD	-0.16	0.2	-0.3	0.38
16000000	115200	1	8	10	0xF7	-0.32	0.32	-1	0.36
16000000	230400	1	4	5	0x55	-0.8	0.64	-1.12	1.76
16000000	460800	1	2	2	0xBB	-1.44	1.28	-3.92	1.68
16777216	9600	1	109	3	0xB5	-0.03	0.02	-0.05	0.06
16777216	19200	1	54	9	0xEE	-0.06	0.06	-0.11	0.13
16777216	38400	1	27	4	0xFB	-0.11	0.1	-0.33	0
16777216	57600	1	18	3	0x44	-0.16	0.15	-0.2	0.45
16777216	115200	1	9	1	0xB5	-0.31	0.31	-0.53	0.78
16777216	230400	1	4	8	0xEE	-0.75	0.74	-2	0.87
16777216	460800	1	2	4	0x92	-1.62	1.37	-3.56	2.06
20000000	9600	1	130	3	0x25	-0.02	0.03	0	0.07
20000000	19200	1	65	1	0xD6	-0.06	0.03	-0.1	0.1
20000000	38400	1	32	8	0xEE	-0.1	0.13	-0.27	0.14
20000000	57600	1	21	11	0x22	-0.16	0.13	-0.16	0.38
20000000	115200	1	10	13	0xAD	-0.29	0.26	-0.46	0.66
20000000	230400	1	5	6	0xEE	-0.67	0.51	-1.71	0.62
20000000	460800	1	2	11	0x92	-1.38	0.99	-1.84	2.8

15.3.14 在 UART 模式中使用具有低功率模式的 eUSCI_A 模块

eUSCI_A 模块提供针对与低功率模式一起使用时的自动时钟激活。当 eUSCI_A 时钟源未激活时，因为器件处于一个低功耗模式，eUSCI_A 在需要时将其自动激活，而不论时钟源的控制位是如何设置的。直到 eUSCI_A 模块回到其空闲状态，时钟都会保持激活。在 eUSCI_A 模块返回空闲条件后，时钟源的控制权会恢复到其控制位的设置。

15.3.15 eUSCI_A 中断

eUSCI_A 只有一个中断矢量，此矢量由传输和接收共用。

15.3.15.1 eUSCI_A 发送中断运行

UCTXIFG 中断标志由发送器置 1 以表示 UCAxTXBUF 已准备好接收另一个字符。如果 UCTXIE 和 GIE 也被置 1，就会产生一个中断请求信号。如果一个字符被写入 UCAxTXBUF，那么 UCTXIFG 将自动复位。

在一个 PUC 后或当 UCSWRST = 1 时，UCTXIFG 会被置 1。在一个 PUC 后或者当 UCSWRST = 1 时，UCTXIE 被复位。

15.3.15.2 eUSCI_A 接收中断运行

每当一个字符被接收并被被载入 UCAxRXBUF 中，UCRXIFG 中断标志都会置 1。如果 UCRXIE 和 GIE 也被置 1，就会产生一个中断请求信号。UCRXIFG 和 UCRXIE 由一个系统复位 PUC 信号复位或当 UCSWRST = 1 时复位。当 UCAxRXBUF 被读取时 UCRXIFG 自动复位。

附加中断控制特性包括：

- 当 UCRXEIE = 0 时，错误字符不会将 UCRXIFG 置 1。
- 当 UCDORM = 1 时，非地址字符不会在多处理器模式下将 UCRXIFG 置 1。在普通 UART 模式下，字符不能将 UCRXIFG 置 1。
- 当 UCBRKIE= 1 时，一个中断条件将置 1 UCBRK 位和 UCRXIFG 标志。

15.3.15.3 eUSCI_A 接收中断运行

表 15-6描述了 I²C 状态更改中断标志。

表 15-6. UART 状态更改中断标志

中断标志	中断条件
UCSTTIFG	接收到的 START 字节中断。 当 UART 模块接收到一个 START 字节时，这个标志被置 1。
UCTXCPTIFG	发送完成中断。在内部移位寄存器中包括 STOP 位在内的完整 UART 字节被移出并且 UCAxTXBUF 被清空时，这个标志被置 1。

15.3.15.4 UCAxIV，中断矢量生成器

所有 eUSCI_A 中断标志被按照优先级排序，并与一个单一中断矢量组合在一起。中断矢量寄存器 UCAxIV 用于确定哪个标志请求了一个中断。被启用的具有最高优先级的中断在 UCAxIV 寄存器中生成一个数，这个数可被评估或添加到程序计数器中来自动进入适当的软件例程。禁用的中断不影响 UCAxIV 的值。

UCAxIV 寄存器的读取访问自动将最高挂起中断条件和标志复位。UCAxIV 寄存器的写入访问清除所有挂起的中断条件和标志。如果另一个中断标志被置 1，则另一个中断将会在处理最初中断后立即产生另一个中断。

Example 15-1显示 UCAxIV 的建议使用方法。UCAxIV 的值被加入 PC 以便自动跳转到适当的例程。以下示例针对 eUSCI_A0 给出。

Example 15-1. UCAxIV 软件示例

```
#pragma vector = USCI_A0_VECTOR __interrupt void USCI_A0_ISR(void) {
    switch(__even_in_range(UCA0IV,18)) {
        case 0x00:      // Vector 0: No interrupts
                 break;
        case 0x02: ...  // Vector 2: UCRXIFG
                 break;
        case 0x04: ...  // Vector 4: UCTXIFG
                 break;
        case 0x06: ...  // Vector 6: UCSTTIFG
```

Example 15-1. UCAxIV 软件示例 (continued)

```
                break;
        case 0x08: ...  // Vector 8: UCTXCPTIFG
                break;
        default: break;
    }
}
```

15.4 eUSCI_A UART 寄存器

在表 15-7中列出了适用于 UART 模式的 eUSCI_A 寄存器和其各自的地址偏移量。有关寄存器的基址，请参见具体器件的数据表。

表 15-7. eUSCI_A UART 寄存器

偏移量	首字母缩写词	寄存器名称	类型	访问	复位	部分
00h	UCAxCTLW0	eUSCI_Ax 控制字0	读取/写入	字	0001h	15.4.1 节
01h	UCAxCTL0(1)	eUSCI_Ax 控制 0	读取/写入	字节	00h	
00h	UCAxCTL1	eUSCI_Ax 控制 1	读取/写入	字节	01h	
02h	UCAxCTLW1	eUSCI_Ax 控制字 1	读取/写入	字	0003h	15.4.2 节
06h	UCAxBRW	eUSCI_Ax 波特率控制字	读取/写入	字	0000h	15.4.3 节
06h	UCAxBR0(1)	eUSCI_Ax 波特率控制 0	读取/写入	字节	00h	
07h	UCAxBR1	eUSCI_Ax 波特率控制 1	读取/写入	字节	00h	
08h	UCAxMCTLW	eUSCI_Ax 调制控制字	读取/写入	字	00h	15.4.4 节
0Ah	UCAxSTATW	eUSCI_Ax 状态	读取/写入	字	00h	15.4.5 节
0Ch	UCAxRXBUF	eUSCI_Ax 接收缓冲区	读取/写入	字	00h	15.4.6 节
0Eh	UCAxTXBUF	eUSCI_Ax 传输缓冲区	读取/写入	字	00h	15.4.7 节
10h	UCAxABCTL	eUSCI_Ax 自动波特率控制	读取/写入	字	00h	15.4.8 节
12h	UCAxIRCTL	eUSCI_Ax IrDA 控制	读取/写入	字	0000h	15.4.9 节
12h	UCAxIRTCTL	eUSCI_Ax IrDA 传输控制	读取/写入	字节	00h	
13h	UCAxIRRCTL	eUSCI_Ax IrDA 接收控制	读取/写入	字节	00h	
1Ah	UCAxIE	eUSCI_Ax 中断使能	读取/写入	字	00h	15.4.10 节
1Ch	UCAxIFG	eUSCI_Ax 中断标志	读取/写入	字	02h	15.4.11 节
1Eh	UCAxIV	eUSCI_Ax 中断向量	读取	字	0000h	15.4.12 节

(1) 建议使用 16 位来访问这些寄存器。如果使用 8 位访问，则相应的位名称后缀应为“_H”。

15.4.1 UCAxCTLW0 寄存器

eUSCI_Ax 控制字寄存器 0

图 **15-12. UCAxCTLW0** 寄存器

15	14	13	12	11	10	9	8
UCPEN	UCPAR	UCMSB	UC7BIT	UCSPB	UCMODEx		UCSYNC
rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0
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

当 UCSWRST = 1 时，只进行修改。

表 **15-8. UCAxCTLW0** 寄存器说明

位	字段	类型	复位	说明
15	UCPEN	RW	0h	奇偶校验使能 0b = 奇偶校验被禁用 1b = 奇偶校验被使能。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。
14	UCPAR	RW	0h	奇偶校验选择。奇偶校验被禁用时，UCPAR 不能使用。 0b = 奇数校验 1b = 偶数奇偶校验
13	UCMSB	RW	0h	MSB 优先选择。控制移位寄存器接收和发送的方向。 0b = LSB 首先 1b = 最高有效位优先 (MSB)
12	UC7BIT	RW	0h	字符长度。选择 7 位或 8 位字符长度。 0b = 8 位数据 1b = 7 位数据
11	UCSPB	RW	0h	停止位选择。停止位的个数。 0b = 一个停止位 1b = 两个停止位
10-9	UCMODEx	RW	0h	eUSCI_A 模式。当 UCSYNC=0 时，UCMODEx 位选择异步模式。 00b = UART 模式 01b = 空闲线多处理器模式 10b = 地址位多处理器模式 11b = 带有自动波特率检测的 UART 模式
8 个	UCSYNC	RW	0h	同步模式使能 0b = 异步模式 1b = 同步模式
7-6	UCSSELx	RW	0h	eUSCI_A 时钟源选择。这些位选择 BRCLK 时钟源。 00b = UCLK 01b = 特定于器件 10b = SMCLK 11b = SMCLK
5	UCRXEIE	RW	0h	接收错误字符中断使能 0b = 拒绝的错误字符和 UCRXIFG 没有置 1。 1b = 接收的错误字符将 UCRXIFG 置 1。
4	UCBRKIE	RW	0h	接收暂停字符中断使能 0b = 接收到的中断字符没有将 UCRXIFG 置 1。 1b = 接收到的中断字符将 UCRXIFG 置 1。

表 **15-8. UCAxCTLW0** 寄存器说明 **(continued)**

位	字段	类型	复位	说明
3	UCDORM	RW	0h	休眠。令 eUSCI_A 进入休眠模式。 0b = 没处于休眠状态。所有接收到的字符将 UCRXIFG 置 1。 1b = 休眠。只有被空闲线或地址位设置在前面的字符才会将 UCRXIFG 置 1。在再有自动波特率检测的 UART 模式下，只有中断和同步字段的组合才能置 1 UCRXIFG。
2	UCTXADDR	RW	0h	发送地址取决于根据选择的多处理器模式，发送的下一帧会被标记为地址。 0b = 发送的下一帧是数据。 1b = 发送的下一帧是一个地址。
1	UCTXBRK	RW	0h	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。 0b = 发送的下一帧不是一个中断。 1b = 发送的下一帧是一个中断或一个中断/同步。
0	UCSWRST	RW	1h	软件复位使能 0b = 被禁用。为了运行而释放的 eUSCI_A 复位。 1b = 被使能。eUSCI_A 逻辑保持在复位状态。

15.4.2 UCAxCTLW1 寄存器

eUSCI_Ax 控制字寄存器 1

图 **15-13. UCAxCTLW1** 寄存器

15	14	13	12	11	10	9	8
保留							
r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
7	6	5	4	3	2	1	0
保留						UCGLITx	
r-0	r-0	r-0	r-0	r-0	r-0	rw-1	rw-1

表 **15-9. UCAxCTLW1** 寄存器说明

位	字段	类型	复位	说明
15-2	保留	R	0h	保留
1-0	UCGLITx	RW	3h	去毛刺脉冲时间 00b = 约为 2ns 01b = 约为 50ns 10b = 约为 100ns 11b = 约为 200ns

15.4.3 UCAxBRW 寄存器

eUSCI_Ax 波特率控制字寄存器

图 15-14. UCAxBRW 寄存器

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

当 UCSWRST = 1 时，只进行修改。

表 15-10. UCAxBRW 寄存器说明

位	字段	类型	复位	说明
15-0	UCBRx	RW	0h	波特率发生器的时钟预分频器设置

15.4.4 UCAxMCTLW 寄存器

eUSCI_Ax 调制控制字寄存器

图 15-15. UCAxMCTLW 寄存器

15	14	13	12	11	10	9	8
UCBRSx							
rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0
7	6	5	4	3	2	1	0
UCBRFx				保留			UCOS16
rw - 0	rw - 0	rw - 0	rw - 0	r0	r0	r0	rw-0

当 UCSWRST = 1 时，只进行修改。

表 15-11. UCAxMCTLW 寄存器说明

位	字段	类型	复位	说明
15-8	UCBRSx	RW	0h	第二调制阶段选择。这些位为 BITCLK 保持了一个自由调制模式。
7-4	UCBRFx	RW	0h	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。在“过采样波特率的生成”一节中给出了调制模式。
3-1	保留	R	0h	保留
0	UCOS16	RW	0h	过采样模式被启用 0b = 被禁用 1b = 被使能

15.4.5 UCAxSTATW 寄存器

eUSCI_Ax 状态寄存器

图 15-16. UCAxSTATW 寄存器

15	14	13	12	11	10	9	8
保留							
r0	r0	r0	r0	r0	r0	r0	r0
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

只在 UCSWRST = 1 时修改。

表 15-12. UCAxSTATW 寄存器说明

位	字段	类型	复位	说明
15-8	保留	R	0h	保留
7	UCLISTEN	RW	0h	监听使能。UCLISTEN 位选择回路模式。 0b = 被禁用 1b = 被使能。UCAxTXD 被内部反馈到接收器。
6	UCFE	RW	0h	帧错误标志。读取 UCAxRXBUF 后将清零 UCFE。 0b = 无错误 1b = 接收到的低停止位字符
5	UCOE	RW	0h	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 1。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，它无法正常工作。 0b = 无错误 1b = 发生溢出错误。
4	UCPE	RW	0h	奇偶校验错误标志。当 UCPEN = 0 时，UCPE 读为 0。读取 UCAxRXBUF 后将清零 UCPE。 0b = 无错误 1b = 接收到的具有奇偶校验错误的字符
3	UCBRK	RW	0h	中断检测标志。读取 UCAxRXBUF 后将清零 UCBRK。 0b = 无中断条件 1b = 出现无中断条件。
2	UCRXERR	RW	0h	接收错误标志。该位指示接收到的字符中存在一个或多个错误。当 UCRXERR = 1 时，一个或多个错误标志位 (UCFE，UCPE，UCOE) 也被置 1。当 UCAxRXBUF 被读取时，UCRXERR 被清除。 0b = 没有检测到接收错误 1b = 检测到接收错误
1	UCADDR UCIDLE	RW	0h	UCADDR: 在地址位多处理器模式中接收到的地址。当 UCAxRXBUF 被读取时，UCADDR 被清零。 UCIDLE：在空闲线多处理器模式中检测到的空闲线路。当 UCAxRXBUF 被读取时，UCIDLE 被清零。 0b = UCADDR：接收到的字符为数据。UCIDLE：未检测到空闲线路 1b = UCADDR：接收到的字符是一个地址。UCIDLE：检测到空闲线
0	UCBUSY	R	0h	eUSCI_A 忙线。该位表示一个发送或接收操作正在进行。 0b = eUSCI_A 处于非活动状态 1b = eUSCI_A 发送或接收中

15.4.6 UCAxRXBUF 寄存器

eUSCI_Ax 接收缓冲寄存器

图 **15-17. UCAxRXBUF** 寄存器

15	14	13	12	11	10	9	8
保留							
r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
7	6	5	4	3	2	1	0
UCRXBUFx							
r	r	r	r	r	r	r	r

表 **15-13. UCAxRXBUF** 寄存器说明

位	字段	类型	复位	说明
15-8	保留	R	0h	保留
7-0	UCRXBUFx	R	0h	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 UCAxRXBUF 复位接收错误位，UCADDR 或 UCIDLE 位，和 UCRXIFG。在 7 位数据模式下，UCAxRXBUF 是 LSB 对齐的并且 MSB 总是复位。

15.4.7 UCAxTXBUF 寄存器

eUSCI_Ax 发送缓冲寄存器

图 **15-18. UCAxTXBUF** 寄存器

15	14	13	12	11	10	9	8
保留							
r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
7	6	5	4	3	2	1	0
UCTXBUFx							
rw	rw	rw	rw	rw	rw	rw	rw

表 **15-14. UCAxTXBUF** 寄存器说明

位	字段	类型	复位	说明
15-8	保留	R	0h	保留
7-0	UCTXBUFx	RW	0h	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 UCTXIFG。UCAxTXBUF 的 MSB 不用于 7 位数据且被复位。

15.4.8 UCAxABCTL 寄存器

eUSCI_Ax 自动波特率控制寄存器

图 15-19. UCAxABCTL 寄存器

15	14	13	12	11	10	9	8
保留							
r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
7	6	5	4	3	2	1	0
保留		UCDELIMx		UCSTOE	UCBTOE	保留	UCABDEN
r-0	r-0	rw - 0	rw - 0	rw - 0	rw - 0	r-0	rw-0

只在 UCSWRST = 1 时修改。

表 15-15. UCAxABCTL 寄存器是

位	字段	类型	复位	说明
15-6	保留	R	0h	保留
5-4	UCDELIMx	RW	0h	中断/同步定界符长度 00b = 1 个位时间 01b = 2 个位时间 10b = 3 个位时间 11b = 4 个位时间
3	UCSTOE	RW	0h	同步字段超时错误 0b = 无错误 1b = 超出可测量时间的同步字段长度。
2	UCBTOE	RW	0h	中断超时错误 0b = 无错误 1b = 超过 22 位时间的中断字段长度。
1	保留	R	0h	保留
0	UCABDEN	RW	0h	自动波特率检测使能 0b = 波特率检测被禁用。中断和同步字段长度没有被测量。 1b = 波特率检测被使能。测量中断和同步字段的长度并也相应改变波特率的设置。

15.4.9 UCAxIRCTL 寄存器

eUSCI_Ax IrDA 控制字寄存器

图 15-20. UCAxIRCTL 寄存器

15	14	13	12	11	10	9	8
UCIRRXFLx						UCIRRXPL	UCIRRXFE
rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0
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

只在 UCSWRST = 1 时修改。

表 15-16. UCAxIRCTL 寄存器说明

位	字段	类型	复位	说明
15-10	UCIRRXFLx	RW	0h	接收过滤器长度。要接收的最短脉冲长度是由下式给出的： t_{MIN} = (UCIRRXFLx + 4) / (2 × $f_{IRTXCLK}$)
9	UCIRRXPL	RW	0h	IrDA 接收输入 UCAxRXD 极性 0b = 当一个轻脉冲出现时，IrDA 收发器传递了一个高脉冲。 1b = 当一个轻脉冲出现时，IrDA 收发器传递了一个低脉冲。
8 个	UCIRRXFE	RW	0h	IrDA 接收滤波器被启用 0b = 接收过滤器被禁用 1b = 接收过滤器被使能
7-2	UCIRTXPLx	RW	0h	发送脉冲长度。 脉冲长度 $t_{脉冲}$=(UCIRTXPLx + 1)/ (2 × $f_{IRTXCLK}$)
1	UCIRTXCLK	RW	0h	IrDA 的发送脉冲时钟选择 0b = BRCLK UCOS16 = 1 时 1b = BITCLK16 。否则为 BRCLK。
0	UCIREN	RW	0h	IrDA 编码器/解码器使能 0b = IrDA 编码器/解码器被禁用 1b = IrDA 编码器/解码器被启用

15.4.10 UCAxIE 寄存器

eUSCI_Ax 中断使能寄存器

图 15-21. UCAxIE 寄存器

15	14	13	12	11	10	9	8
保留							
r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
7	6	5	4	3	2	1	0
保留				UCTXCPTIE	UCSTTIE	UCTXIE	UCRXIE
r-0	r-0	r-0	r-0	rw - 0	rw - 0	rw - 0	rw - 0

表 15-17. UCAxIE 寄存器说明

位	字段	类型	复位	说明
15-4	保留	R	0h	保留
3	UCTXCPTIE	RW	0h	发送完成中断使能 0b = 中断被禁用 1b = 中断被启用
2	UCSTTIE	RW	0h	起始位中断使能 0b = 中断被禁用 1b = 中断被启用
1	UCTXIE	RW	0h	发送中断使能 0b = 中断被禁用 1b = 中断被启用
0	UCRXIE	RW	0h	接收中断使能 0b = 中断被禁用 1b = 中断被使能

15.4.11 UCAxIFG 寄存器

eUSCI_Ax 中断标志寄存器

图 15-22. UCAxIFG 寄存器

15	14	13	12	11	10	9	8
保留							
r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
7	6	5	4	3	2	1	0
保留				UCTXCPTIFG	UCSTTIFG	UCTXIFG	UCRXIFG
r-0	r-0	r-0	r-0	rw - 0	rw - 0	rw - 1	rw - 0

表 15-18. UCAxIFG 寄存器说明

位	字段	类型	复位	说明
15-4	保留	R	0h	保留
3	UCTXCPTIFG	RW	0h	发送就绪中断使能。当内部移位寄存器中的整个字节被移出且在 UCAxTXBUF 为空时，UCTXRDYIFG时 会被置 1。 0b = 无中断挂起 1b = 中断挂起
2	UCSTTIFG	RW	0h	起始位中断标志。在接收到一个起始位后 UCSTTIFG 被置 1 0b = 无中断挂起 1b = 中断挂起
1	UCTXIFG	RW	1h	发送中断标志。在 UCAxTXBUF 为空时 UCTXIFG 被置 1。 0b = 无中断挂起 1b = 中断挂起
0	UCRXIFG	RW	0h	接收中断标志。当 UCAxRXBUF 收到一个完整的字符时，UCRXIFG 被置 1。 0b = 无中断挂起 1b = 中断挂起

15.4.12 UCAxIV 寄存器

eUSCI_Ax 中断向量寄存器

图 15-23. UCAxIV 寄存器

15	14	13	12	11	10	9	8
UCIVx							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
UCIVx							
r0	r0	r0	r0	r-(0)	r-(0)	r-(0)	r0

表 15-19. UCAxIV 寄存器说明

位	字段	类型	复位	说明
15-0	UCIVx	R	0h	eUSCI_A 的中断向量值 00h = 无中断挂起 02h = 中断源：接收缓冲区已满；中断标志：UCRXIFG；中断优先级：最高 04h = 中断源：发送缓冲区为空；中断标志：UCTXIFG 06h = 中断源：接收到的起始位；中断标志：UCSTTIFG 08h = 中断源：发送完成；中断标志：UCTXCPTIFG；中断优先级：最低

增强型通用串行通信接口 (eUSCI) – SPI 模式

增强型通用串行通信接口 eUSCI_A 和 eUSCI_B 可在一个硬件模块上支持多种串行通信模式。本章对同步外设接口 (SPI) 模式的操作进行了说明。

Topic **Page**

16.1 增强型通用串行通信接口 (eUSCI_A，eUSCI_B) 概述

eUSCI_A 和 eUSCI_B 这两个接口都支持 SPI 模式下的串行通信。

16.2 eUSCI 介绍 - SPI 模式

在同步模式下，器件通过 eUSCI 的三个或四个引脚（UCxSIMO、UCxSOMI、UCxCLK 和 UCxSTE）与外部系统相连。当 UCSYNC 位置 1 时，SPI模式被选中，通过 UCMODEx 位的可以选择SPI模式(3线或4线)。

SPI 模式的特点包括：

- 7 位或 8 位的数据长度
- LSB 或 MSB 优先的数据发送和接收
- 3 引脚或 4 引脚 SPI 操作
- 主或从模式
- 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
- 连续发送和接收操作
- 可选的时钟极性和相位控制
- 主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 模式中的从器件操作

图 16-1给出了配置为 SPI 模式时的 eUSCI。

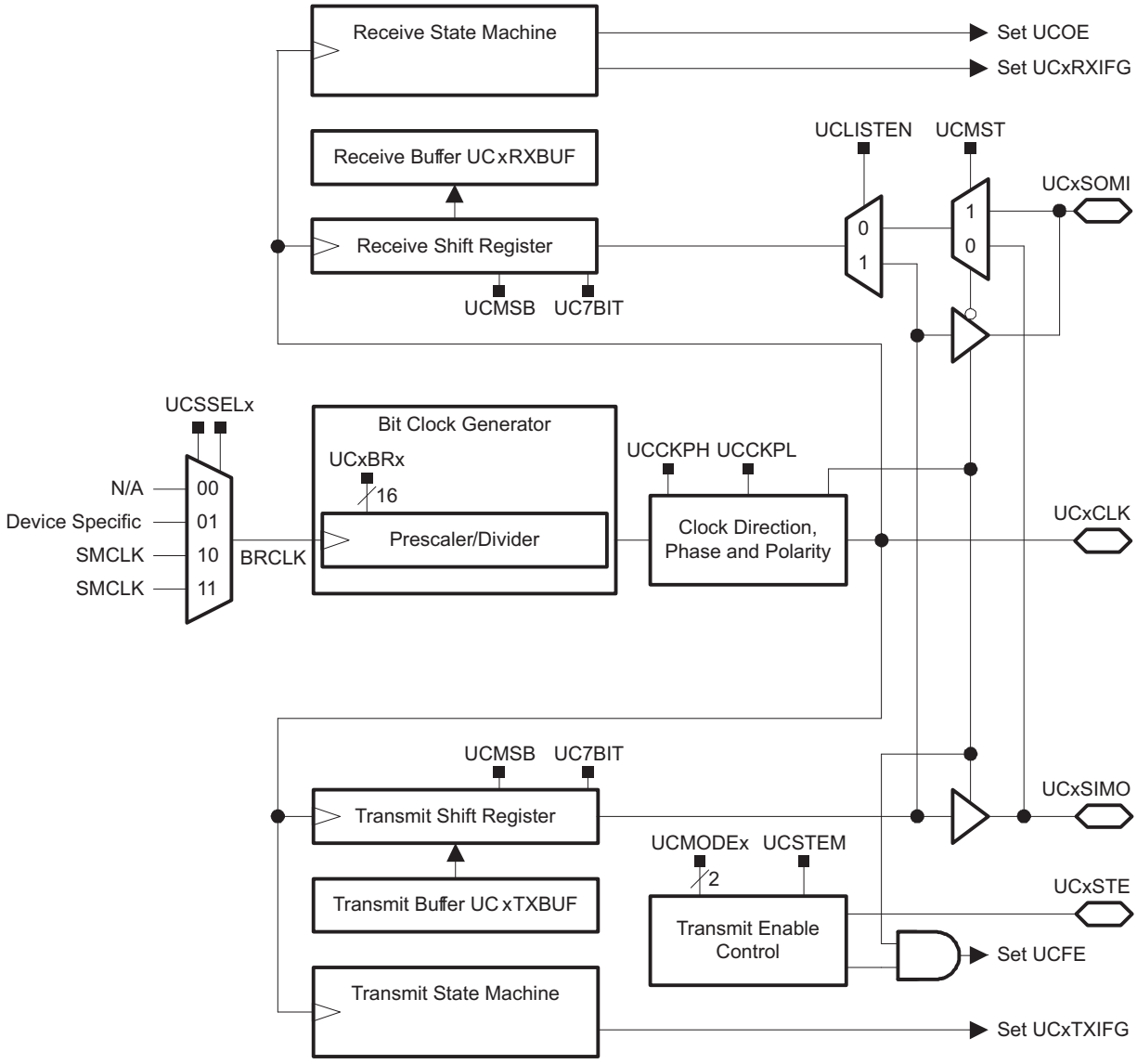


图 **16-1. eUSCI** 框图 **- SPI** 模式

16.3 eUSCI 的运行 - SPI 模式

在 SPI 模式下，串行数据可在多个从器件中进行通信。这些从器件使用同一个由主器件提供的时钟。一个由主器件控制的额外引脚，UCxSTE，被用于使能从器件的接收和发送操作。

SPI 数据交换可以使用三个或四个信号线：

- UCxSIMO:从器件输入，主器件输出

  主模式：UCxSIMO 为数据的输出线。
  从模式：UCxSIMO 为数据的输入线。

- UCxSIM1：从器件输出，主器件输入

  主模式：UCxSOMI 为数据的输入线。
  从模式：UCxSOMI 为数据的输出线。

- UCxCLK – eUSCI SPI 时钟

  主模式：UCxCLK 为输出。
  从模式：UCxCLK 为输入。

- UCxSTE - 从器件发送使能

  用于4线模式，以便允许在一条总线上存在多个主器件。不用于 3 线模式。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	激活
		1	激活	未激活
10	低	0	激活	未激活
		1	未激活	激活

16.3.1 eUSCI 的初始化和复位

eUSCI 通过一个 PUC 或 UCSWRST 位来复位。 PUC 之后，UCSWRST 位会自动置 1，以使 eUSCI 保持复位状态。当置 1 时，UCSWRST 位复位 UCRXIE，UCTXIE，UCRXIFG，UCOE，和 UCFE 位并置 1 UCTXIFG 标志。清零 UCSWRST 会使 eUSCI 处于运行状态。

为避免发生无法预测的行为，应在 UCSWRST 置 1 时配置或重新配置 eUSCI 模块。

注: 初始化或重新配置 eUSCI 模块

推荐的 eUSCI 初始化或重新配置过程如下：

1. 将 UCSWRST 置 1。

   ```
   BIS.B #UCSWRST,&UCxCTL1
   ```

2. 在 UCSWRST = 1 时初始化所有 eUSCI 寄存器（包括 UCxCTL1）。
3. 配置端口。
4. 用软件清零 UCSWRST。

   ```
   BIC.B #UCSWRST,&UCxCTL1
   ```

5. 通过将 UCRXIE 或 UCTXIE 置 1 使能中断（可选）。

16.3.2 字符格式

SPI 模式下的 eUSCI 模块支持通过 UC7BIT 位选择的 7 位和 8 位字符长度。 在 7 位数据模式下，UCxRXBUF 是 LSB 对齐，而 MSB 始终复位。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： **默认的字符格式**

默认 SPI 字符传输是从 LSB 开始。 与其他 SPI 接口进行通信时，可能需要采用 MSB 优先模式。

注： **图表的字符格式**

本章的所有图表均使用 MSB 优先的格式。

16.3.3 主模式

图 16-2 给出了 eUSCI 作为主器件时的 3 引脚和 4 引脚配置。

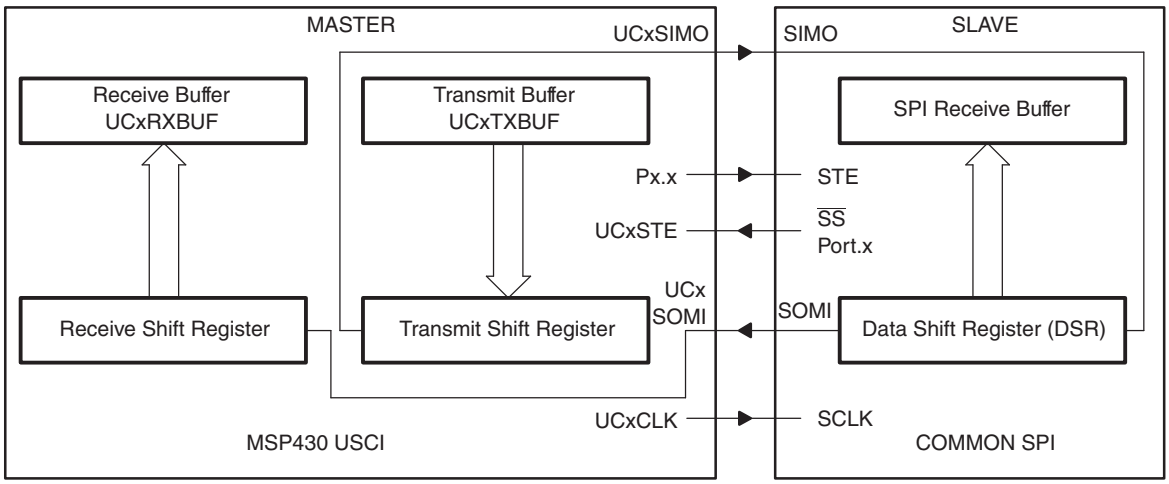


图 16-2. eUSCI 主器件和外部从器件 (UCSTEM = 0)

当数据被送到传输数据缓冲器 UCxTXBUF 时，eUSCI 开始数据传送。 当发送 (TX) 移位寄存器为空时，UCxTXBUF 的数据就会被传送到该移位寄存器中，从而在 UCxSIMO 上传送初始化数据，起始位是最高位还是最低位，取决于 UCMSB 的设置。 UCxSOMI 上的数据在相反的时钟沿上被移入接收移位寄存器。收到字符后，接收数据由接收 (RX) 移位寄存器移至接收数据缓冲区 UCxRXBUF，同时接收中断标志 UCRXIFG 置 1 以指示 RX 或 TX 操作完成。

发送中断标志 UCTXIFG 置 1 时，表示数据已从 UCxTXBUF 移至 TX 移位寄存器并且 UCxTXBUF 已准备好接收新数据。它不指示 RX 或 TX 操作完成。

要使 eUSCI 在主模式下接收数据，必须将数据写入 UCxTXBUF，因为接收和发送操作同时发生。

以下是将 eUSCI 配置为 4 引脚主器件的两种不同选项，详细内容将在后续章节中进行介绍：

- 第四个引脚用作输入以避免与其他主器件发生冲突 (UCSTEM = 0)。
- 第四个引脚用作输出以生成从器件使能信号 (UCSTEM = 1)。

位 UCSTEM 用于选择相应的模式。

16.3.3.1 4 引脚 SPI 主模式 (UCSTEM = 0)

在 4 引脚主模式下，如果 UCSTEM = 0，则 UCxSTE 作为数字输入，可用于避免与其他主器件发生冲突并控制主器件（见表 16-1）。当 UCxSTE 处于主器件不活动状态且 UCSTEM = 0 时：

- UCxSIMO 和 UCxCLK 被设置为输入并不再驱动总线。
- 出错位 UCFE 置 1 表明出现一个将由用户处理的通讯错误。
- 内部状态被复位并且移位操作取消。

如果数据被写入 UCxTXBUF 而主器件被 UCxSTE 保持在非激活状态，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 转回为主器件激活状态时须把数据重新写入 UCxTXBUF 以便发送。 UCxSTE 输入信号不会在 3 引脚主模式中使用。

16.3.3.2 4 引脚 SPI 主模式 (UCSTEM = 1)

在 4 引脚模式下，如果 UCSTEM = 1，则 UCxSTE 作为数字输出。在该模式下，UCxSTE 会自动为单独的从器件生成从器件使能信号。相应的行为可参见图 16-4。

该功能并不适用于需要多个从器件的情况，在这种情况下，软件需要转为使用通用 I/O 引脚单独为每个从器件生成 STE 信号。

16.3.4 从模式

图 16-3 给出了 eUSCI 作为从器件时的 3 引脚和 4 引脚配置。

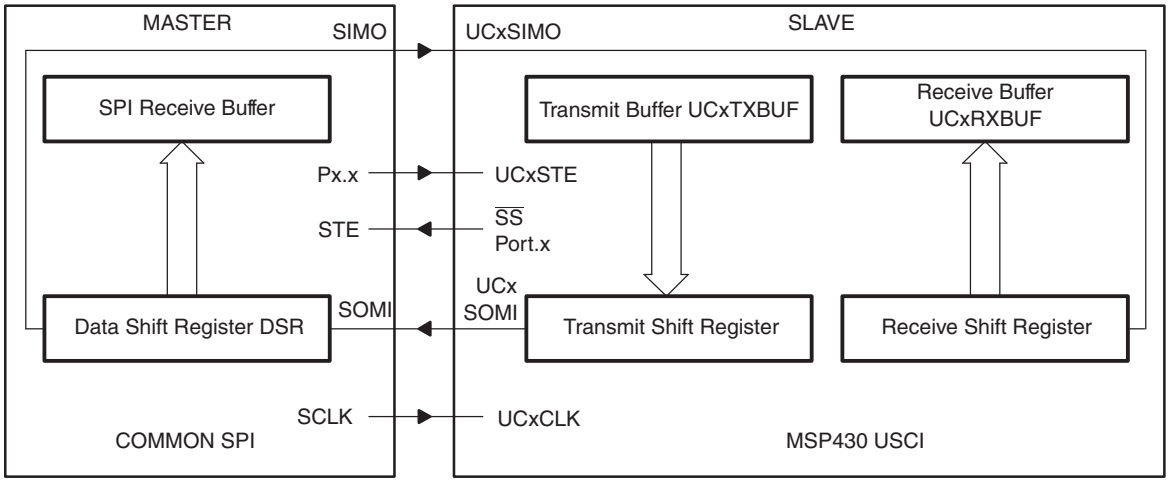


图 16-3. eUSCI 从器件和外部主器件

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UCxCLK被传输到UCxSOM1之前，数据被写入 UCxTXBUF 并被移动到 TX 移位寄存器。 当指定数量的数据被接收到后，UCxSIMO 上的数据在UCxCLK 的相反时钟沿上被移入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 中时，UCxRXIFG 中断标志被置 1，这表明数据已被接收。 当之前接收到的数据在新数据被移送到 UCxRXBUF 前未被 UCxRXBUF 读取时，则溢出错误位 UCOE 就会被置 1。

16.3.4.1 4 引脚 SPI 从模式：

在 4 引脚从模式下，UCxSTE 作为数字输入供从器件使用以使能发送和接收操作，并由 SPI 主器件驱动。当 UCxSTE 处于从器件激活状态时，从器件正常运行。当 UCxSTE 处于从器件未激活状态时：

- UCxSIMO 上任何正在进行中的接收操作被暂停。
- UCxSOMI 被设置为输入方向。
- 移位操作被暂停直到 UCxSIMO 线进入从器件发送有效状态。

在 3 引脚从模式中，不会使用UCxSTE 输入信号。

16.3.5 SPI 使能

通过清零 UCSWRST 位使能 eUSCI 模块时，该模块准备接收和发送数据。 在主模式中，位时钟发生器就绪，但既不计时也不产生任何时钟。 在从模式中，位时钟发生器被禁用并由主器件提供。

UCBUSY=1 标志着一个发送或接收操作。

一个 PUC 或置 1 UCSWRST 位将立即禁用 eUSCI 并且所有激活的传输都被终止。

16.3.5.1 发送使能

在主模式中，写入 UCxTXBUF 将激活位时钟发生器，且数据开始发送。

在从模式中，当一个主器件提供一个时钟时，数据即开始传输，而在 4 引脚模式中，需要UCxSTE 处于从器件激活状态中。

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 串行时钟控制

在 SPI 总线上，主器件提供 UCxCLK。 当 UCMST= 1 时，在 UCxCLK 引脚上的位时钟将由 eUSCI 位发生器提供。被用于产生位时钟的时钟由 UCSSELx 位进行选择。 当 UCMST = 0 时，eUSCI 时钟由主器件通过 UCxCLK 引脚提供，位时钟发生器不被使用，且 UCSSELx 位无影响。 SPI 接收器和传送器并行运行且数据传输使用相同的时钟源。

比特率控制寄存器 UCxxBRW 中 UCBRx 位的 16 位值是 eUSCI 时钟源 (BRCLK) 的分频系数。如果 UCBRx = 0，则主模式下可生成的最大位时钟为 BRCLK。SPI 模式下不使用调制，并且当 eUSCI_A 采用 SPI 模式时，应将 UCAxMCTL 清零。UCAxCLK 或 UCBxCLK 的频率可通过公式 12 进行计算。

$$f_{BitClock} = f_{BRCLK} / (UCBRx + 1) \quad (12)$$

当 UCBRx 设置为奇数时，分频值为偶数，生成的位时钟占空比为 50%。

当 UCBRx 设置为偶数时，分频值为奇数。在这种情况下，位时钟的高相位比低相位长一个 BRCLK 周期。

当 UCBRx = 0 时，不对 BRCLK 进行分频，位时钟等于 BRCLK。

16.3.6.1 串行时钟的极性和相位

UCxCLK 的极性和相位分别通过 eUSCI 的 UCCKPL 和 UCCKPH 控制位单独进行配置。在图 16-4中给出了每种情况下的时序。

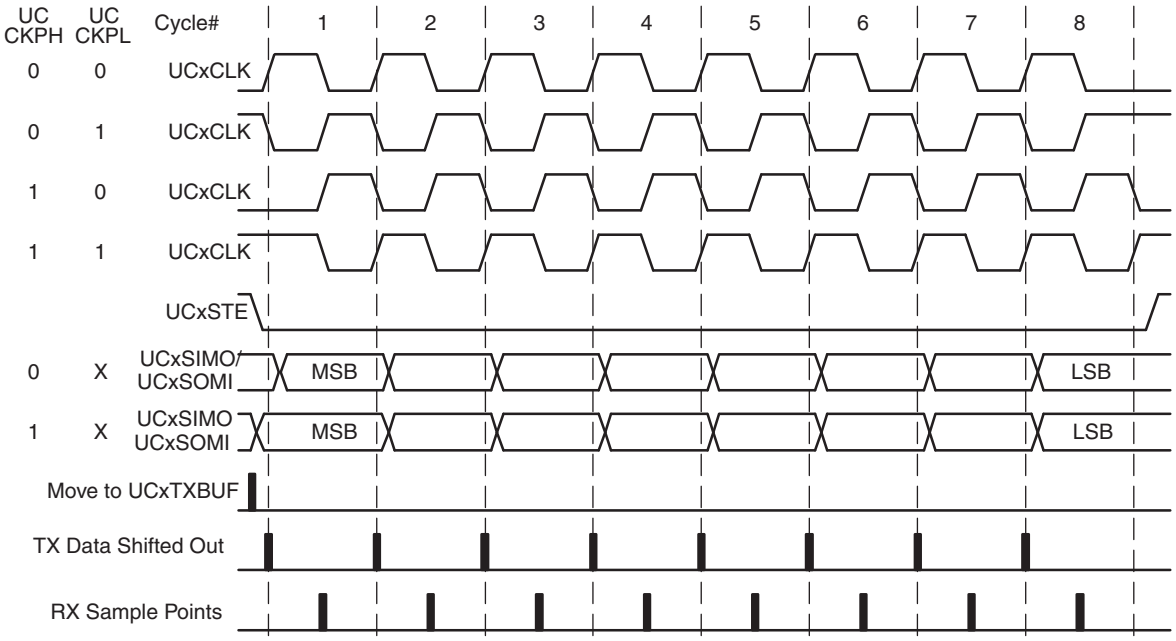


图 16-4. UCMSB = 1 时的 eUSCI SPI 时序

16.3.7 使用 SPI 低功耗模式

eUSCI 模块为低功耗模式下的使用提供了时钟自动激活。因为器件处于一个低功耗模式，当 eUSCI 的时钟源是无效的时，必要时，不管时钟源的控制位的设置如何，eUSCI 模块都将自动激活。时钟会一直保持激活直到 eUSCI 模块回到其空闲状态。eUSCI 模块恢复空闲条件后，时钟源的控制会恢复到其控制位的设置。

在 SPI 从模式中，无需内部时钟源，因为所需时钟都是由外部主器件提供。即使器件是在 LPM4 中且所有时钟源都被禁用，仍然可以使用从模式下的eUSCI。接收或发送中断可将 CPU 从任何一种低功耗模式中唤醒。

在LPM4模式下作为从器件接收多个字节，需要把 CPU 的唤醒时间考虑在内。如果 CPU 的唤醒时间是，例如，150µs（参见特定器件数据手册），就需要要确保，在第二个字节完全被 eUSCI_A 或 eUSCI_B 接收前，CPU 已为第一次接收到的字节 TXIFG 提供服务。否则，将发生溢出错误。

16.3.8 SPI 中断

eUSCI 只有一个中断向量是传输和接收共享的。eUSCI_Ax 和 eUSCI_Bx 不共享同一个中断向量。

16.3.8.1 SPI 的发送中断操作

发送器通过置 1 UCTXIFG 中断标志来表示 UCxTXBUF 准备接收另一个字符。如果 UCBRXIE 和 GIE 也被置 1，就会产生一个中断请求信号。如果一个字符被写入 UCxTXBUF，则 UCTXIFG 会自动复位。在一个 PUC 后或当 UCSWRST = 1 时，UCTXIFG 会被置 1。在一个 PUC 后或当 UCSWRST = 1 时，UCTXIE 被置 1。

注： 在 **SPI** 模式中写入 **UCxTXBUF**

当 UCTXIFG = 0 时，写入 UCxTXBUF 的数据可能会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 UCRXIFG 中断标志就会被置 1。如果 UCBRXIE 和 GIE 也被置 1，就会产生一个中断请求信号。UCRXIFG 和 UCRXIE 由一个系统复位 PUC 信号复位或当 UCSWRST = 1 时复位。当 UCxRXBUF 被读取时，UCRXIFG 会自动复位。

16.3.8.3 UCxIV，中断向量发生器

eUSCI 中断标志被优先化并被结合以便发起一个中断向量。中断向量寄存器 UCxIV 用于确定哪个标志要求了一个中断。被使能的具有最高优先级的中断在 UCxIV 中生成了一个数，该数可被评估或是被添加到编程计数器 (PC) 中，以便自动进入相应的软件程序中。禁用的中断不影响 UCxIV 的值。

对 UCxIV 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。如果有另一个中断标志置 1，则在处理完最初的中断后会立即产生另一个中断。

16.3.8.3.1 UCxIV 软件示例

下列软件示例给出了 UCxIV 的推荐用法。UCxIV 的值被加入 PC 以便自动跳转到相应的程序。为 eUSCI0_B 给出了下列示例。

```
USCI_SPI_ISR
         ADD      &UCB0IV, PC  ; Add offset to jump table
         RETI                  ; Vector 0: No interrupt
         JMP      RXIFG_ISR    ; Vector 2: RXIFG
TXIFG_ISR                      ; Vector 4: TXIFG
         ...                   ; Task starts here
         RETI                  ; Return
RXIFG_ISR                      ; Vector 2
         ...                   ; Task starts here
         RETI                  ; Return
```

16.4 eUSCI_A SPI 寄存器

表 16-2中列出了可用于 SPI 模式中的 eUSCI_A 寄存器。基址可在具体器件的数据表中找到。

表 16-2. eUSCI_A SPI 寄存器

偏移量	首字母缩写词	寄存器名称	类型	访问	复位	部分
00h	UCAxCTLW0	eUSCI_Ax 控制字0	读取/写入	字	0001h	16.4.1 节
00h	UCAxCTL1	eUSCI_Ax 控制 1	读取/写入	字节	01h	
01h	UCAxCTL0	eUSCI_Ax 控制 0	读取/写入	字节	00h	
06h	UCAxBRW	eUSCI_Ax 位速率控制字	读取/写入	字	0000h	16.4.2 节
06h	UCAxBR0	eUSCI_Ax 位速率控制 0	读取/写入	字节	00h	
07h	UCAxBR1	eUSCI_Ax 位速率控制 1	读取/写入	字节	00h	
0Ah	UCAxSTATW	eUSCI_Ax 状态	读取/写入	字	00h	16.4.3 节
0Ch	UCAxRXBUF	eUSCI_Ax 接收缓冲区	读取/写入	字	00h	16.4.4 节
0Eh	UCAxTXBUF	eUSCI_Ax 传输缓冲区	读取/写入	字	00h	16.4.5 节
1Ah	UCAxIE	eUSCI_Ax 中断使能	读取/写入	字	00h	16.4.6 节
1Ch	UCAxIFG	eUSCI_Ax 中断标志	读取/写入	字	02h	16.4.7 节
1Eh	UCAxIV	eUSCI_Ax 中断向量	读取	字	0000h	16.4.8 节

16.4.1 UCAxCTLW0 寄存器

eUSCI_Ax 控制寄存器 0

图 16-5. UCAxCTLW0 寄存器

15	14	13	12	11	10	9	8
UCCKPH	UCCKPL	UCMSB	UC7BIT	UCMST	UCMODEx		UCSYNC
rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0
7	6	5	4	3	2	1	0
UCSSELx		保留				UCSTEM	UCSWRST
rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 1

只在 UCSWRST = 1 时修改。

表 16-3. UCAxCTLW0 寄存器说明

位	字段	类型	复位	说明
15	UCCKPH	RW	0h	时钟相位选择。只在 UCSWRST = 1 时修改。 0b = 数据在第一个 UCLK 边沿发生变化并在下一个边沿进行捕捉。 1b = 数据在第一个 UCLK 边沿上被捕捉且在下一个边沿上被改变。
14	UCCKPL	RW	0h	时钟极性选择。只在 UCSWRST = 1 时修改。 0b = 未激活的状态为低电平。 1b = 未激活的状态是高电平。
13	UCMSB	RW	0h	MSB 优先选择。控制移位寄存器接收和发送的方向。只在 UCSWRST = 1 时修改。 0b = LSB 优先 1b = 最高有效位优先 (MSB)
12	UC7BIT	RW	0h	字符长度。选择 7 位或 8 位字符长度。只在 UCSWRST = 1 时修改。 0b = 8 位数据 1b = 7 位数据
11	UCMST	RW	0h	主模式选择。只在 UCSWRST = 1 时修改。 0b = 从模式 1b = 主模式
10-9	UCMODEx	RW	0h	eUSCI 模式。当 UCSYNC=1 时，UCMODEx 位选择同步模式。只在 UCSWRST = 1 时修改。 00b = 3 引脚 SPI 01b = UCxSTE 高电平有效时的 4 引脚 SPI：当 UCxSTE = 1 时，从器件模式被启用 10b = UCxSTE 低电平有效时的 4 引脚 SPI：当 UCxSTE = 0 时，从器件模式被启用 11b = I2C 模式
8	UCSYNC	RW	0h	同步模式使能。只在 UCSWRST = 1 时修改。 0b = 异步模式 1b = 同步模式
7-6	UCSSELx	RW	0h	eUSCI 时钟源选择。这些位用于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。只在 UCSWRST = 1 时修改。 00b = 保留 01b = 特定于器件 10b = SMCLK 11b = SMCLK
5-2	保留	R	0h	保留
1	UCSTEM	RW	0h	主模式中的 STE 模式选择。这个字节在从模式或 3 线制模式中被忽略。只在 UCSWRST = 1 时修改。 0b = STE 引脚被用来防止与其它主器件的冲突 1b = STE 引脚被用来生成针对 4 线制从器件的使能信号

表 16-3. UCAxCTLW0 寄存器说明 (continued)

位	字段	类型	复位	说明
0	UCSWRST	RW	1h	软件复位使能 0b = 被禁用。eUSCI 被复位以进行操作。 1b = 被禁用。在复位状态中 eUSCI 逻辑状态被保持。

16.4.2 UCAxBRW 寄存器

eUSCI_Ax 位速率控制寄存器 1

图 16-6. UCAxBRW 寄存器

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

只在 UCSWRST = 1 时修改。

表 16-4. UCAxBRW 寄存器说明

位	字段	类型	复位	说明
15-0	UCBRx	RW	0h	位时钟预分频器设置。只在 UCSWRST = 1 时修改。 $f_{BitClock} = f_{BRCLK} / (UCBRx + 1)$

16.4.3 UCAxSTATW 寄存器

eUSCI_Ax 状态寄存器

图 16-7. UCAxSTATW 寄存器

15	14	13	12	11	10	9	8
保留							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
UCLISTEN	UCFE	UCOE	保留				UCBUSY
rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	r-0

只在 UCSWRST = 1 时修改。

表 16-5. UCAxSTATW 寄存器说明

位	字段	类型	复位	说明
15-8	保留	R	0h	保留
7	UCLISTEN	RW	0h	监听使能。UCLISTEN 位选择回路模式。只在 UCSWRST = 1 时修改。 0b = 被禁用 1b = 被启用。发送器输出从内部反馈到接收器。
6	UCFE	RW	0h	帧错误标志。该位表示了一个 4 线制主模式中的总线冲突。 UCFE 不能用于 3 线制主模式或任何从模式。 0b = 无错误 1b = 发生了总线冲突
5	UCOE	RW	0h	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 1。当读取 UCxRXBUF 时，UCOE 自动清零，并且不得用软件清零。否则，它无法正常工作。 0b = 无错误 1b = 发生了超限错误
4-1	保留	RW	0h	保留
0	UCBUSY	R	0h	eUSCI 忙。该位表示一个发送或接收操作正在进行。 0b = eUSCI 未激活 1b = eUSCI 正在发送或接收

16.4.4 UCAxRXBUF 寄存器

eUSCI_Ax 接收缓冲寄存器

图 16-8. UCAxRXBUF 寄存器

15	14	13	12	11	10	9	8
保留							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
UCRXBUFx							
rw	rw	rw	rw	rw	rw	rw	rw

表 16-6. UCAxRXBUF 寄存器说明

位	字段	类型	复位	说明
15-8	保留	R	0h	保留
7-0	UCRXBUFx	R	0h	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。对 UCxRXBUF 的读取将复位接收错误位和 UCRXIFG。在 7 位数据模式下，UCxRXBUF 是 LSB 对齐，而 MSB 始终复位。

16.4.5 UCAxTXBUF 寄存器

eUSCI_Ax 发送缓冲寄存器

图 16-9. UCAxTXBUF 寄存器

15	14	13	12	11	10	9	8
保留							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
UCTXBUFx							
rw	rw	rw	rw	rw	rw	rw	rw

表 16-7. UCAxTXBUF 寄存器说明

位	字段	类型	复位	说明
15-8	保留	R	0h	保留
7-0	UCTXBUFx	RW	0h	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以及已经被发送的数据。到发送数据缓冲器的写入将清除 UCTXIFG。UCxTXBUF 的 MSB 不用于 7 位数据且被复位。

16.4.6 UCAxIE 寄存器

eUSCI_Ax 中断使能寄存器

图 16-10. UCAxIE 寄存器

15	14	13	12	11	10	9	8
保留							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
保留						UCTXIE	UCRXIE
r-0	r-0	r-0	r-0	r-0	r-0	rw - 0	rw - 0

表 16-8. UCAxIE 寄存器说明

位	字段	类型	复位	说明
15-2	保留	R	0h	保留
1	UCTXIE	RW	0h	发送中断使能 0b = 中断被禁用 1b = 中断被启用
0	UCRXIE	RW	0h	接收中断使能 0b = 中断被禁用 1b = 中断被使能

16.4.7 UCAxIFG 寄存器

eUSCI_Ax 中断标志寄存器

图 16-11. UCAxIFG 寄存器

15	14	13	12	11	10	9	8
保留							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
保留						UCTXIFG	UCRXIFG
r-0	r-0	r-0	r-0	r-0	r-0	rw - 1	rw - 0

表 16-9. UCAxIFG 寄存器说明

位	字段	类型	复位	说明
15-2	保留	R	0h	保留
1	UCTXIFG	RW	1h	发送中断标志。在 UCAxTXBUF 为空时 UCTXIFG 被置 1。 0b = 无中断挂起 1b = 中断挂起
0	UCRXIFG	RW	0h	接收中断标志。当 UCAxRXBUF 收到一个完整的字符时，UCRXIFG 被置 1。 0b = 无中断挂起 1b = 中断挂起

16.4.8 UCAxIV 寄存器

eUSCI_Ax 中断向量寄存器

图 16-12. UCAxIV 寄存器

15	14	13	12	11	10	9	8
UCIVx							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
UCIVx							
r0	r0	r0	r-0	r-0	r-0	r-0	r0

表 16-10. UCAxIV 寄存器说明

位	字段	类型	复位	说明
15-0	UCIVx	R	0h	eUSCI 中断矢量值 000h = 无中断挂起 002h = 中断源：数据被接收到；中断标志：UCRXIFG；中断优先级：最高 004h = 中断源：发送缓冲器空：中断标志：UCTXIFG；中断优先级：最低

16.5 eUSCI_B SPI 寄存器

表 16-11中列出了适用于 SPI 模式的 eUSCI_B 寄存器以及它们的地址偏移。基址可在具体器件的数据表中找到。

表 16-11. eUSCI_B SPI 寄存器

偏移量	首字母缩写词	寄存器名称	类型	访问	复位	部分
00h	UCBxCTLW0	eUSCI_Bx 控制字 0	读取/写入	字	01C1h	16.5.1 节
00h	UCBxCTL1	eUSCI_Bx 控制 1	读取/写入	字节	C1h	
01h	UCBxCTL0	eUSCI_Bx 控制 0	读取/写入	字节	01h	
06h	UCBxBRW	eUSCI_Bx 位速率控制字	读取/写入	字	0000h	16.5.2 节
06h	UCBxBR0	eUSCI_Bx 位速率控制 0	读取/写入	字节	00h	
07h	UCBxBR1	eUSCI_Bx 位速率控制 1	读取/写入	字节	00h	
08h	UCBxSTATW	eUSCI_Bx 状态	读取/写入	字	00h	16.5.3 节
0Ch	UCBxRXBUF	eUSCI_Bx 接收缓冲器	读取/写入	字	00h	16.5.4 节
0Eh	UCBxTXBUF	eUSCI_Bx 发送缓冲器	读取/写入	字	00h	16.5.5 节
2Ah	UCBxIE	eUSCI_Bx 中断使能	读取/写入	字	00h	16.5.6 节
2Ch	UCBxIFG	eUSCI_Bx 中断标志	读取/写入	字	02h	16.5.7 节
2Eh	UCBxIV	eUSCI_Bx 中断矢量	读取	字	0000h	16.5.8 节

16.5.1 UCBxCTLW0 寄存器

eUSCI_Bx 控制寄存器 0

图 16-13. UCBxCTLW0 寄存器

15	14	13	12	11	10	9	8
UCCKPH	UCCKPL	UCMSB	UC7BIT	UCMST	UCMODEx		UCSYNC
rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 1
7	6	5	4	3	2	1	0
UCSSELx		保留				UCSTEM	UCSWRST
rw-1	rw-1	r0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 1

只在 UCSWRST = 1 时修改。

表 16-12. UCBxCTLW0 寄存器说明

位	字段	类型	复位	说明
15	UCCKPH	RW	0h	时钟相位选择。只在 UCSWRST = 1 时修改。 0b = 数据在第一个 UCLK 边沿发生变化并在下一个边沿进行捕捉。 1b = 数据在第一个 UCLK 边沿上被捕捉且在下一个边沿上被改变。
14	UCCKPL	RW	0h	时钟极性选择。只在 UCSWRST = 1 时修改。 0b = 未激活的状态为低电平。 1b = 未激活的状态是高电平。
13	UCMSB	RW	0h	MSB 优先选择。控制移位寄存器接收和发送的方向。只在 UCSWRST = 1 时修改。 0b = LSB 优先 1b = 最高有效位优先 (MSB)
12	UC7BIT	RW	0h	字符长度。选择 7 位或 8 位字符长度。只在 UCSWRST = 1 时修改。 0b = 8 位数据 1b = 7 位数据
11	UCMST	RW	0h	主模式选择。只在 UCSWRST = 1 时修改。 0b = 从模式 1b = 主模式
10-9	UCMODEx	RW	0h	eUSCI 模式。当 UCSYNC=1 时，UCMODEx 位选择同步模式。只在 UCSWRST = 1 时修改。 00b = 3 引脚 SPI 01b = UCxSTE 高电平有效时的 4 引脚 SPI：当 UCxSTE = 1 时，从器件模式被启用 10b = UCxSTE 低电平有效时的 4 引脚 SPI：当 UCxSTE = 0 时，从器件模式被启用 11b = I2C 模式
8	UCSYNC	RW	1h	同步模式使能。只在 UCSWRST = 1 时修改。 0b = 异步模式 1b = 同步模式
7-6	UCSSELx	RW	3h	eUSCI 时钟源选择。这些位用于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。只在 UCSWRST = 1 时修改。 00b = 保留 01b = 特定于器件 10b = SMCLK 11b = SMCLK
5-2	保留	R	0h	保留
1	UCSTEM	RW	0h	主模式中的 STE 模式选择。这个字节在从模式或 3 线制模式中被忽略。只在 UCSWRST = 1 时修改。 0b = STE 引脚被用来防止与其它主器件的冲突 1b = STE 引脚被用来生成针对 4 线制从器件的使能信号

表 16-12. UCBxCTLW0 寄存器说明 (continued)

位	字段	类型	复位	说明
0	UCSWRST	RW	1h	软件复位使能 0b = 被启用。eUSCI 被复位以进行操作。 1b = 被启用。在复位状态中 eUSCI 逻辑状态被保持。

16.5.2 UCBxBRW 寄存器

eUSCI_Bx 位速率控制寄存器 1

图 16-14. UCBxBRW 寄存器

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

只在 UCSWRST = 1 时修改。

表 16-13. UCBxBRW 寄存器说明

位	字段	类型	复位	说明
15-0	UCBRx	RW	0h	位时钟预分频器设置。只在 UCSWRST = 1 时修改。 $f_{BitClock} = f_{BRCLK} / (UCBRx + 1)$

16.5.3 UCBxSTATW 寄存器

eUSCI_Bx 状态寄存器

图 16-15. UCBxSTATW 寄存器

15	14	13	12	11	10	9	8
保留							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
UCLISTEN	UCFE	UCOE	保留				UCBUSY
rw - 0	rw - 0	rw - 0	r0	r0	r0	r0	r-0

只在 UCSWRST = 1 时修改。

表 16-14. UCBxSTATW 寄存器说明

位	字段	类型	复位	说明
15-8	保留	R	0h	保留
7	UCLISTEN	RW	0h	监听使能。UCLISTEN 位选择回路模式。只在 UCSWRST = 1 时修改。 0b = 被禁用 1b = 被启用。发送器输出从内部反馈到接收器。
6	UCFE	RW	0h	帧错误标志。该位表示了一个 4 线制主模式中的总线冲突。 3 线制主模式或任何从模式下都不使用 UCFE。 0b = 无错误 1b = 发生了总线冲突
5	UCOE	RW	0h	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 1。当读取 UCxRXBUF 时，UCOE 自动清零，并且不得用软件清零。否则，它无法正常工作。 0b = 无错误 1b = 发生了超限错误
4-1	保留	R	0h	保留
0	UCBUSY	R	0h	eUSCI 忙。该位表示一个发送或接收操作正在进行。 0b = eUSCI 未激活 1b = eUSCI 正在发送或接收

16.5.4 UCBxRXBUF 寄存器

eUSCI_Bx 接收缓冲寄存器

图 16-16. UCBxRXBUF 寄存器

15	14	13	12	11	10	9	8
保留							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
UCRXBUFx							
rw	rw	rw	rw	rw	rw	rw	rw

表 16-15. UCBxRXBUF 寄存器说明

位	字段	类型	复位	说明
15-8	保留	R	0h	保留
7-0	UCRXBUFx	R	0h	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。对 UCxRXBUF 的读取将复位接收错误位和 UCRXIFG。在 7 位数据模式下，UCxRXBUF 是 LSB 对齐，而 MSB 始终复位。

16.5.5 UCBxTXBUF 寄存器

eUSCI_Bx 发送缓冲寄存器

图 16-17. UCBxTXBUF 寄存器

15	14	13	12	11	10	9	8
保留							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
UCTXBUFx							
rw	rw	rw	rw	rw	rw	rw	rw

表 16-16. UCBxTXBUF 寄存器说明

位	字段	类型	复位	说明
15-8	保留	R	0h	保留
7-0	UCTXBUFx	RW	0h	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以及已经被发送的数据。到发送数据缓冲器的写入将清除 UCTXIFG。UCxTXBUF 的 MSB 不用于 7 位数据且被复位。

16.5.6 UCBxIE 寄存器

eUSCI_Bx 中断使能寄存器

图 16-18. UCBxIE 寄存器

15	14	13	12	11	10	9	8
保留							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
保留						UCTXIE	UCRXIE
r-0	r-0	r-0	r-0	r-0	r-0	rw - 0	rw - 0

表 16-17. UCBxIE 寄存器说明

位	字段	类型	复位	说明
15-2	保留	R	0h	保留
1	UCTXIE	RW	0h	发送中断使能 0b = 中断被禁用 1b = 中断被启用
0	UCRXIE	RW	0h	接收中断使能 0b = 中断被禁用 1b = 中断被启用

16.5.7 UCBxIFG 寄存器

eUSCI_Bx 中断标志寄存器

图 16-19. UCBxIFG 寄存器

15	14	13	12	11	10	9	8
保留							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
保留						UCTXIFG	UCRXIFG
r-0	r-0	r-0	r-0	r-0	r-0	rw - 1	rw - 0

表 16-18. UCBxIFG 寄存器说明

位	字段	类型	复位	说明
15-2	保留	R	0h	保留
1	UCTXIFG	RW	1h	发送中断标志。当 UCBxTXBUF 为空时，UCTXIFG 被置 1。 0b = 无中断挂起 1b = 中断挂起
0	UCRXIFG	RW	0h	接收中断标志。当 UCBxRXBUF 已经收到一个完整的字符时，UCRXIFG 被置 1。 0b = 无中断挂起 1b = 中断挂起

16.5.8 UCBxIV 寄存器

eUSCI_Bx 中断矢量寄存器

图 16-20. UCBxIV 寄存器

15	14	13	12	11	10	9	8
UCIVx							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
UCIVx							
r0	r0	r0	r-0	r-0	r-0	r-0	r0

表 16-19. UCBxIV 寄存器说明

位	字段	类型	复位	说明
15-0	UCIVx	R	0h	eUSCI 中断矢量值 0000h = 无中断挂起 0002h = 中断源：数据已经被接收到；中断标志：UCRXIFG；中断优先级：最高 0004h = 中断源：发送缓冲器空；中断标志：UCTXIFG；中断优先级：最低

增强型通用串行通信接口 (eUSCI) – I²C 模式

增强型通用串行通信接口 B(eUSCI_B) 通过一个硬件模块支持多种串行通信模式。本章对 I²C 模式的操作进行了说明。

Topic **Page**

17.1 增强型通用串行通信接口 B(eUSCI_B) 概述

eUSCI_B 模块两个串行通信模式：

- I^2C 模式
- SPI 模式

如果在一个产品上实现了多个 eUSCI_B 模块，那么这些模块用递增的数字命名。例如，如果一个产品实现两个 eUSCI_B 模块，那么它们会被命名为 eUSCI0_B 和 eUSCI1_B。

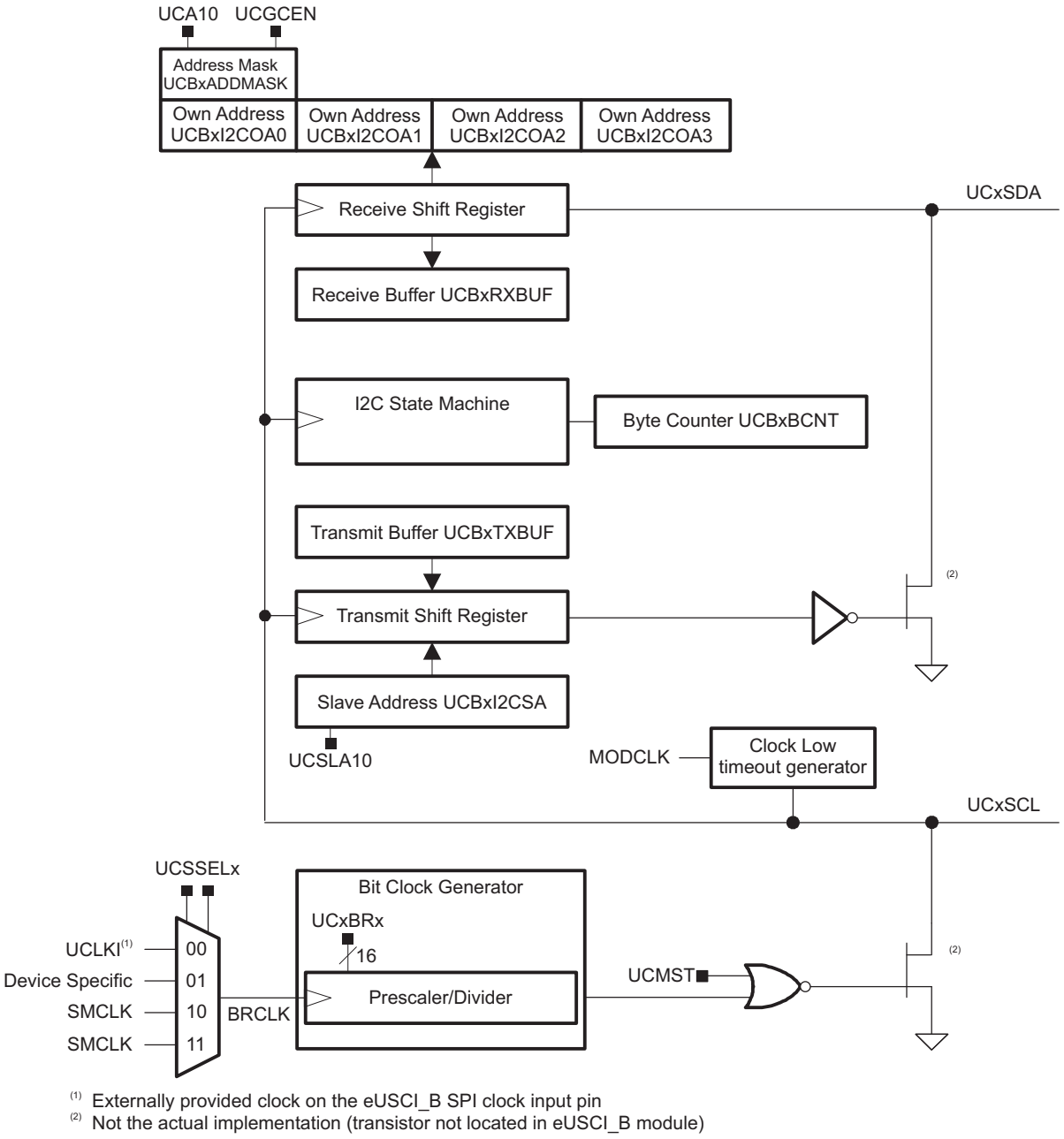
17.2 eUSCI_B 介绍 - I^2C 模式

在 I^2C 模式中，eUSCI_B 模块在器件和用两线式 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。连接至 I^2C 总线的外部元件通过 2 线制 I^2C 接口与 eUSCI_B 模块以串行方式交换数据。

eUSCI_B I^2C 模式的特点包括：

- 7 位和 10 位器件寻址模式
- 常规调用
- 启动、重启、停止
- 多主发送器或接收器模式
- 从接收器或发送器模式
- 支持速率高达 100kbps 的标准模式和速率高达 400kbps 的快速模式
- 主模式下可编程的 UCxCLK 频率
- 低功耗设计
- 带有中断功能的8位字节计数器，并能自动插入STOP
- 支持多达四个硬件从地址，每个都具有自己的中断
- 从地址掩码寄存器以及接收中断的地址
- 时钟低电平超时中断以避免总线停转
- LPM4 中的从器件操作
- 从接收器启动检测，以从 LPMx 模式（非 LPM3.5 和 LPM4.5）自动唤醒

图 17-1 显示了在 I^2C 模式中配置的 eUSCI_B。



(1) Externally provided clock on the eUSCI_B SPI clock input pin
(2) Not the actual implementation (transistor not located in eUSCI_B module)

图 17-1. eUSCI_B 框图 – I²C 模式

17.3 eUSCI_B 运行 – I²C 模式

I²C 模式支持任何从或主 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 (SDA) 引脚和串行时钟 (SCL) 引脚进行通信的。SDA 和 SCL 是双向的且须连接到使用一个上拉电阻的正电源电压上。

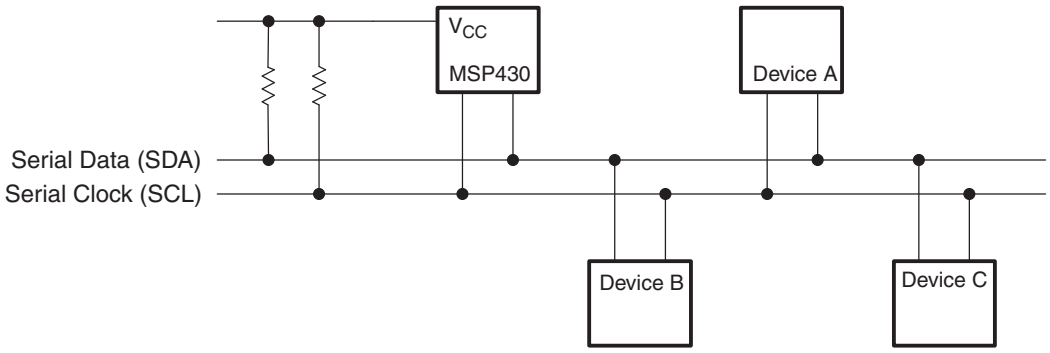


图 17-2. I²C 总线连接框图

注: **SDA 和 SCL 电平**

SDA 和 SCL 引脚均不能被拉高至超过器件 V_{CC}电平。

17.3.1 eUSCI_B 的初始化和复位

eUSCI_B 借助一个 PUC 或者通过设置 UCSWRST 位来复位。一个 PUC 后，UCSWRST 位会自动置 1，以此把 eUSCI_B 保持在复位状态。 为了选择 I²C 运行，须把 UCMODEx 位设置为 11。模块被初始化后，已准备好发送或接收操作。清除 UCSWRST 会使 eUSCI_B 处于运行状态。

将 UCSWRST 置 1 时，应完成 USCI 模块的配置和重新配置，以避免出现不可预知的行为。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。
- UCBxSTAT，清零 15-9 和 6-4 位。
- 寄存器 UCBxIE 和 UCBxIFG 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 eUSCI_B 模块**

建议的 eUSCI_B 初始化/重新配置过程是：

1. 将 UCSWRST 置 1 (`BIS.B #UCSWRST,&UCxCTL1`)。
2. UCSWRST= 1，初始化所有 eUSCI_B 寄存器（包括 UCxCTL1）。
3. 配置端口。
4. 通过软件将 UCSWRST 清零 (`BIC.B #UCSWRST,&UCxCTL1`)。
5. 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。首先传输最高有效位数据，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 位从地址和 R/$\overline{W}$ 位。当 R/$\overline{W}$= 0 时，主器件向从器件发送数据。当 R/$\overline{W}$= 1 时，主器件从从器件处接收数据。ACK信号由接收方在每个字节接收完后的第九个SCL时钟时发送。

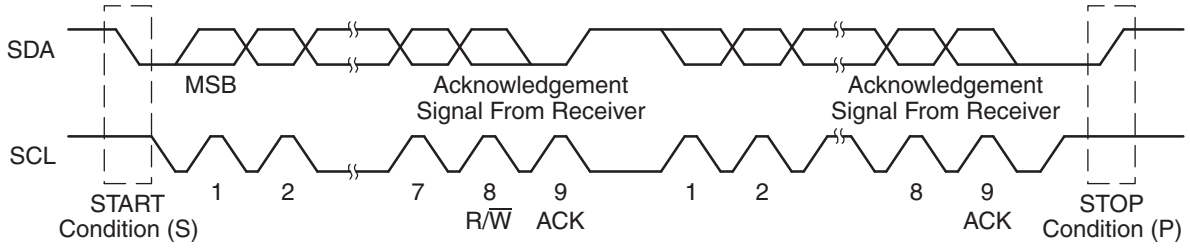


图 **17-3. I²C** 模块数据传输

起始和停止条件是由主器件产生，显示在图 17-3中。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。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START信号后置 1，在STOP信号后清零。

在 SCL 为高电平期间，SDA 上的数据必须保持稳定（参见图 17-4）。只能在 SCL 为低电平时才可以改变 SDA 的高低电平状态，否则将会产生起始和停止条件。

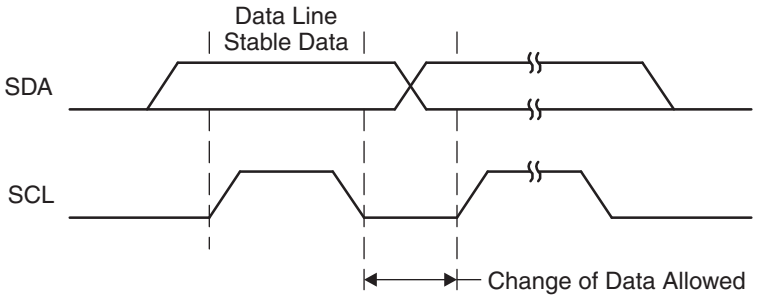


图 **17-4. I²C** 总线上的位传输

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

在 7 位寻址格式中（参见图 17-5），第一个字节是 7 位从地址和 R/$\overline{W}$位。应答位 ACK 是在每个字节后由接收器发出的。

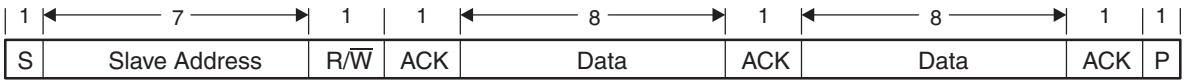


图 **17-5. I²C** 模块 **7** 位寻址格式

17.3.3.2 10 位寻址

在 10 位寻址格式中（参见图 17-6），第一个字节是由 11110b 加上 10 位从地址的两个最高有效位 (MSB) 和 R/$\overline{W}$位构成的。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8 个位，之后是 ACK 位和 8 位数据。有关如何使用 eUSCI_B 模块的 10 位寻址模式的详细信息请参阅I2C 从器件 10 位寻址模式和I2C 主器件 10 位寻址模式。

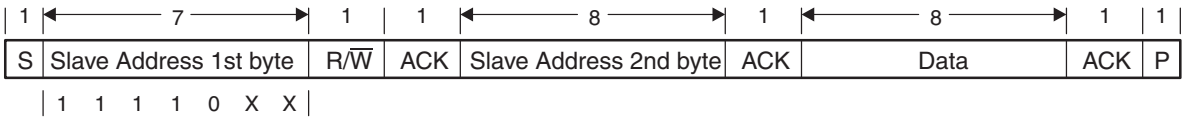


图 **17-6. I²C 模块 10 位寻址格式**

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。在一个重新起始生成后，从地址被 R/$\overline{W}$ 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。

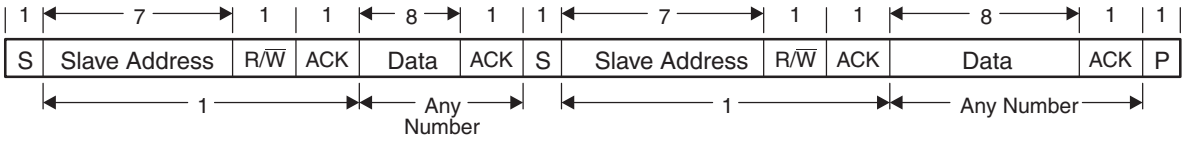


图 **17-7. 带有重复启动条件的 I²C 模块寻址格式**

17.3.4 I²C 快速安装

本节简单介绍了在 I2C 模式下 eUSCI_B 的运行。以一个软件为例子对启动通信的基本步骤进行了描述。更多有关可能的配置和细节的信息可在17.3.5 节中找到。

在 MSP430 网站的“代码示例”下可找到最新的代码示例。

要把 eUSCI_B 设置成一个向从器件发送 0x12h 地址主发送器，只需几个步骤就可以了（参见Example 17-1）。

Example 17-1. 带有 7 位 地址的 TX 主器件

```
UCBxCTL1 |= UCSWRST;           // put eUSCI_B in reset state
UCBxCTLW0 |= UCMODE_3 + UCMST; // I2C master mode
UCBxBRW = 0x0008;              // baud rate = SMCLK / 8
UCBxCTLW1 = UCASTP_2;          // automatic STOP assertion
UCBxTBCNT = 0x07;              // TX 7 bytes of data
UCBxI2CSA = 0x0012;            // address slave is 12hex
P2SEL |= 0x03;                 // configure I2C pins (device specific)
UCBxCTL1 &= ^UCSWRST;          // eUSCI_B in operational state
UCBxIE |= UCTXIE;              // enable TX-interrupt
GIE;                           // general interrupt enable
...
// inside the eUSCI_B TX interrupt service routine
UCBxTXBUF = 0x77;              // fill TX buffer
```

如在代码示例中所示，所有配置都必须在 UCSWRST 置 1 时完成。若要选择 eUSCI_B 的 I²C 运行，就须对 UCMODE 进行相应的设置。通过在 UCBxBRW 寄存器中写入正确的分频器设置传输波特率。选择的默认的时钟是 SMCLK。在一个帧中发送多少字节是由字节计数器阈值寄存器 UCBxTBCNT 和 UCASTPx 位一起控制的。

在 UCBxI2CSA 寄存器中要发送的从地址是指定的。最后，必须对端口进行配置。这一步取决于器件；请参阅必须使用的引脚数据手册。

必须将要传输的每个字节写入中断服务程序内的 UCBxTXBUF。Example 17-3 给出了建议的中断服务程序结构。

Example 17-2给出了把 eUSCI_B 设置为具有 0x12h 地址的从器件所需的步骤，该地址能够从主器件处接收数据和向主器件发送数据。

Example 17-2. 带有 7 位地址的 RX 从器件

```
UCBxCTL1 |= UCSWRST;        // eUSCI_B in reset state
UCBxCTLW0 |= UCMODE_3;      // I2C slave mode
UCBxI2COA0 = 0x0012;        // own address is 12hex
P2SEL |= 0x03;              // configure I2C pins (device specific)
UCBxCTL1 &= ^UCSWRST;       // eUSCI_B in operational state
UCBxIE |= UCTXIE + UCRXIE;  // enable TX&RX-interrupt
GIE;                        // general interrupt enable
...
// inside the eUSCI_B TX interrupt service routine
UCBxTXBUF = 0x77;           // send 077h
...
// inside the eUSCI_B RX interrupt service routine
data = UCBxRXBUF;           // data is the internal variable
```

如在Example 17-2中所示，所有配置都必须在 UCSWRST 置 1 时完成。对于从器件来说，是通过置 1 UCMODE 来选择 I²C 运行的。在 UCBxI2COA0 寄存器中从地址是指定的。为了使能接收和发送请求中断，最终，UCBxIE 和 GIE 中相应的位都需要进行置 1。 最终必须对端口进行配置。这一步取决于器件；请参阅已使用的引脚数据手册。

RX 中断服务例程被调用用于由器件接收的每一个字节。每当主器件请求一个字节时都要执行 RX 中断服务例程。可在Example 17-3中找到建议的中断服务例程结构。

17.3.5 I²C 模块的运行模式

在 I²C 模式下，eUSCI_B 模块可以在主发送器，主接收器，从发送器或从接收器模式下运行。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给出了如何解释时序线路图表。主器件发送的数据用灰色矩形表示；从器件发送的数据用白色矩形表示。不管作为主器件还是从器件，用比其他矩形高的矩形表示的是由 eUSCI_B 模块发送的数据。

eUSCI_B 模块采取的行为是用一个带箭头的灰色矩形表示的，其箭头所指的地方就是数据流中动作发生的地方。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。

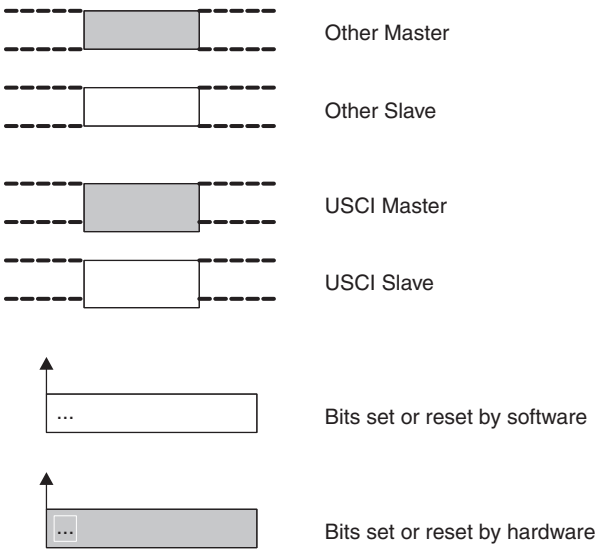


图 **17-8. I²C** 时序线图例

17.3.5.1 从模式

通过设置 UCMODEx = 11、UCSYNC = 1 和 UCMST = 0，将 eUSCI_B 模块配置为 I²C 从器件。

首先，eUSCI_B 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 地址。之后，根据与从地址一起接收到的 R/$\overline{W}$位，对发送和接收操作进行自动控制。

eUSCI_B 从地址是由 UCBxI2COA0 寄存器编程的。在17.3.9 节中说明了对多从地址的支持。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。

当在总线上检测到一个起始条件时，eUSCI_B 模块将接收传送过来的地址，并将之与存储在 UCBxI2COA0 中的自身地址进行比较。当接收到的地址与 eUSCI_B 从地址一致时会将 UCSTTIFG 标志置 1。

17.3.5.1.1 I²C 从发送器模式

当主器件发送的从地址与带有一个设置 R/$\overline{W}$位的自身地址相匹配时，就会使能从发送模式。从发送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，但是当在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请求数据，eUSCI_B 模块会自动配置为发送模式，并置 1 UCTR 和 UCTXIFG0。在数据未写入发送缓存 UCBxTXBUF之前，SCL 时钟线一直保持低电平。然后，地址会被应答且数据被发送。一旦数据被转移到移位寄存器，UCTXIFG0 会再次被置 1。在数据被主器件应答后，下一个被写入 UCBxTXBUF 中的字节数据开始传输，或如果缓冲区为空，通过一直保持 SCL 为低电平直到新的数据被写入 UCBxTXBUF，在应答周期内总线会被挂起。如果主器件在一个停止条件后发送了一个 NACK 信号，则 UCSTPIFG 标志会被置 1。如若一个重复起始条件之后是 NACK，则 USCI I²C 状态机会返回至其地址接收状态。

图 17-9给出了从发送器运行。

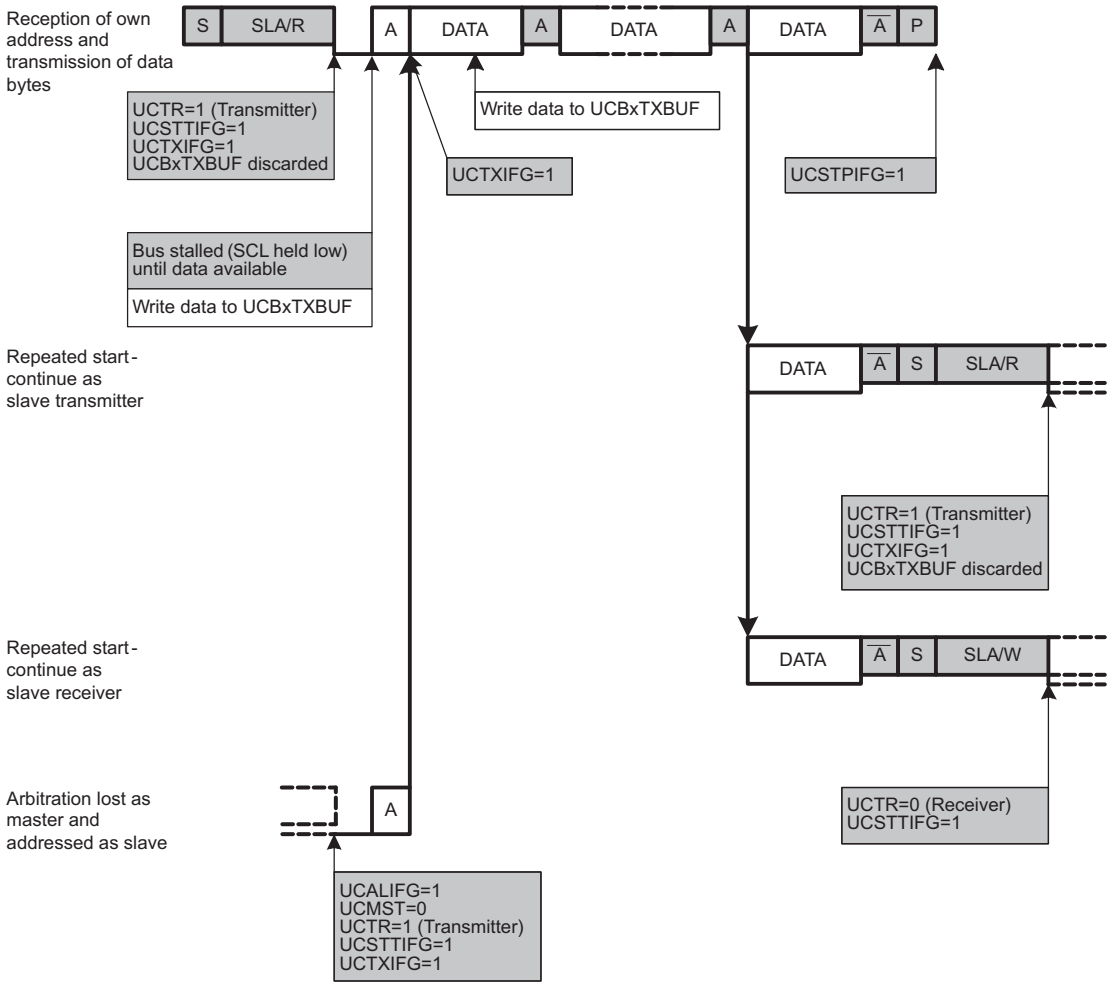


图 **17-9. I²C 从发送器模式**

17.3.5.1.2 I²C 从接收器模式

当主器件发送的从地址和其自身地址相匹配，且接收到一个被清零的 R/$\overline{W}$时，就会使能从器件接收模式。在从接收模式中，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。

如果从器件从主器件处接收了数据，则 eUSCI_B 模块将被自动配置为一个接收器且 UCTR 会被清零。在接收完第一个数据字节后，接收中断标志 UCRXIFG0 会被置 1。eUSCI_B 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，就会把一个应答发送给主器件，然后开始下个数据的接收。

置 1 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 1 UCTXNACK，将会释放总线，并马上会发送一个 NACK，且 UCBxRXBUF 将加载最后一次接收到的数据。因为上一个数据没有被读出，该数据丢失了。为避免数据的丢失，须在 UCTXNACK 置 1 之前读出 UCBxRXBUF。

当主器件产生一个停止条件时，UCSTPIFG 标志被置 1。

如果主器件产生了一个重复起始条件，eUSCI_B I²C 状态机会返回至其地址接收状态。

图 17-10给出了 I²C 从接收器操作。

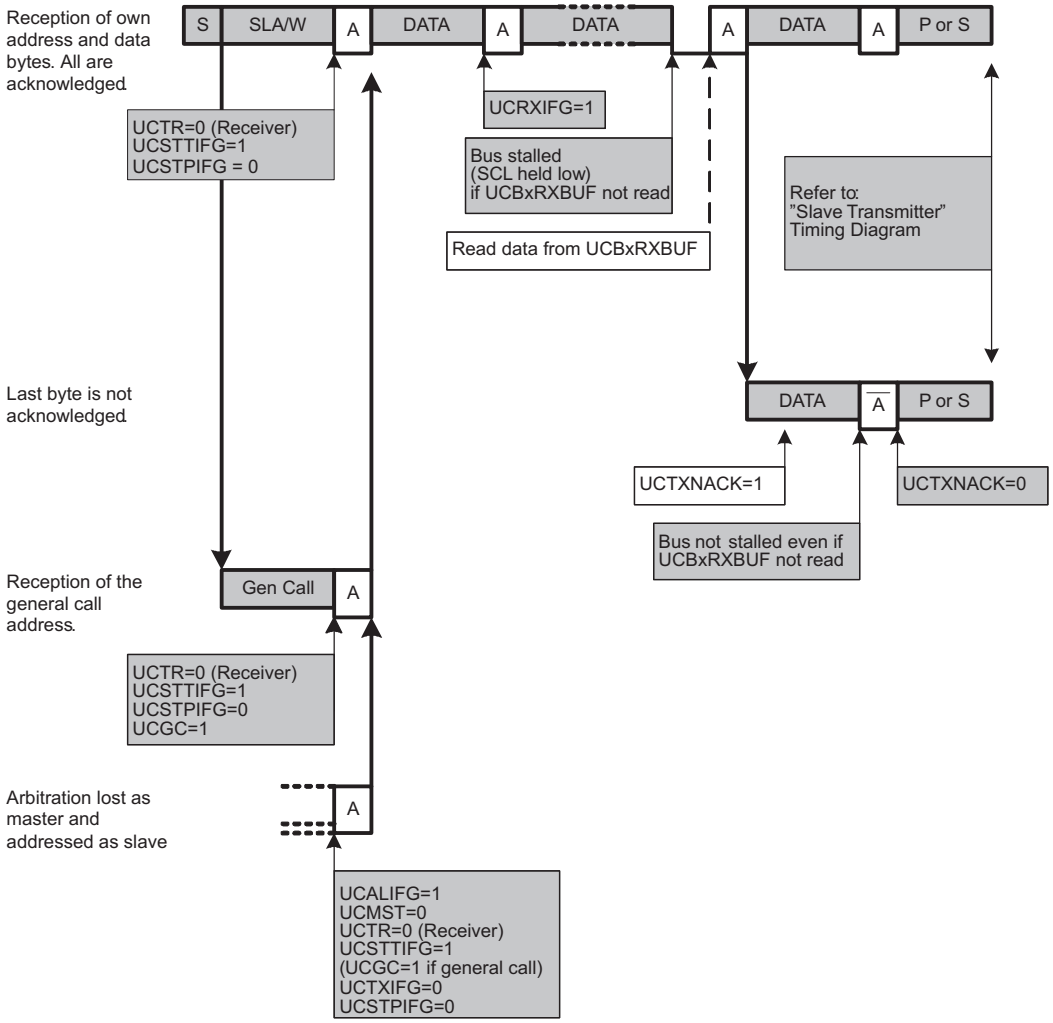


图 17-10. I²C 从接收器模式

17.3.5.1.3 I²C 从器件 10 位寻址模式

如在图 17-11中所示，当 UCA10 = 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。eUSCI_B 模块会通过在清零 UCTR 位时置 1 UCSTTIFG 标志来指明上述操作。若要把从器件切换到发送模式，就需要主器件一起发送一个重复起始条件和地址的第一个字节，并同时发送 R/$\overline{W}$ 位置 1。若 UCSTTIFG 标志先被软件清除，同时 eUSCI_B 模块切换到发送模式，且 UCTR = 1，那么该标志将被置 1。

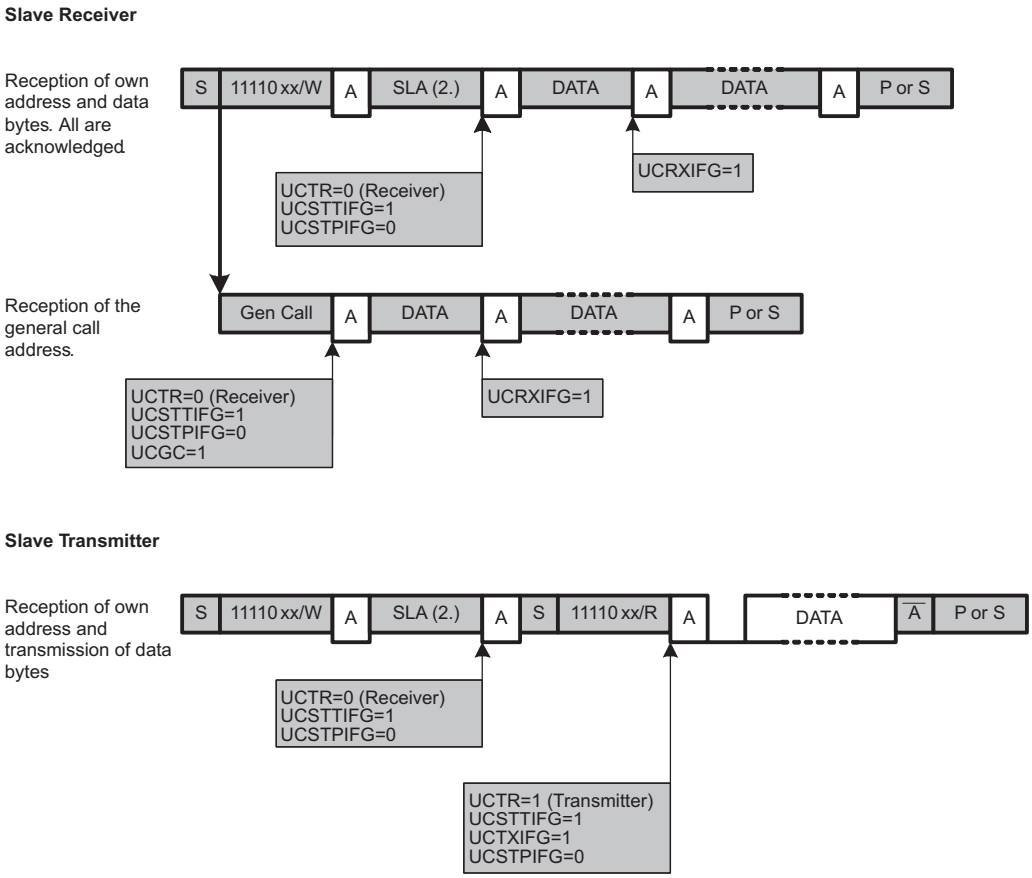


图 **17-11. I²C 从器件 10 位寻址模式**

17.3.5.2 主模式

通过选择 I²C，令 UCMODEx = 11 且 USCYNC = 1，并置 1 UCMST 位，把 eUSCI_B 模块被配置为一个 I²C 主器件。当该主器件为多主系统的一部分时，UCMM 必须置 1 且必须将其自身地址编入 UCBxI2COA0 寄存器。17.3.9 节 中介绍了多个从地址支持。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 eUSCI_B 模块响应常规调用，则可以选择 UCGCEN 位。

注: **地址和多主系统**

在主模式下，只要使能了自身地址检测 (UCOAEN = 1)，特别是在多主系统中，便不允许在自己的地址和从地址寄存器中指定同一地址 (UCBxI2CSA = UCBxI2COAx)。这意味着 eUSCI_B 将自身进行寻址。

用户软件须确保不会发生这种情况。现在还没有检测此类情况的硬件，其后果就是 eUSCI_B 的不可预知的行为。

17.3.5.2.1 I²C 主发送器模式

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 1 UCTR 来选择发送模式、置 1 UCTXSTT 来产生一个起始条件，主发送器模式才被初始化。

eUSCI_B 模块等到总线可用，然后生成启动条件并传输从地址。当起始条件产生且要发送的第一个数据可以写入 UCBxTXBUF 中时， UCTXIFG0 会被置 1。一旦完整的地址被发送出去，**UCTXSTT**标志就会被清零。

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，UCTXIFG0 将被再次置 1。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。只要是以下情况，就传输数据或保持总线：

- 没有生成自动停止
- UCTXSTP 位没被置 1
- UCTXSTT 位没被置 1

在从器件下一个应答信号到来之后，置 1 UCTXSTP 将会产生一个停止条件。如果在从器件地址的传送期间或是 USCI 模块等待把数据写入 UCBxTXBUF 的过程中置 1 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。**在这种情况下，UCSTPIFG 会被置 1。**在传送数据的单字节时，在字节传送过程中或者在数据传输开始后、没把任何新数据写入 UCBxTXBUF 中的任意时间内必须置 1 UCTXSTP。否则，将只发送地址。当数据由缓冲器转移到移位寄存器时，UCBxTXIFG 将被置 1，这就意味着数据传输已经开始，且 UCTXSTP 位 可能被置 1 了。当设置为 UCASTPx = 10 时，字节计数器用于产生停止且用户也就不需要置 1 UCTXSTP了。**在只传输一个字节时建议这么做。**

置 1 UCTXSTT 将会产生一个重复起始条件。在这种情况下，可以通过置 1 或清零 UCTR 来配置为发送器或接收器，且如果需要，一个不同的从器件地址可被写入 UCBxI2CSA。

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 1。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，则必须将其重新写入 UCBxTXBUF。任何置 1 UCTXSTT 或 UCTXSTP 也会被丢弃。

图 17-12给出了 I²C 主发送操作。

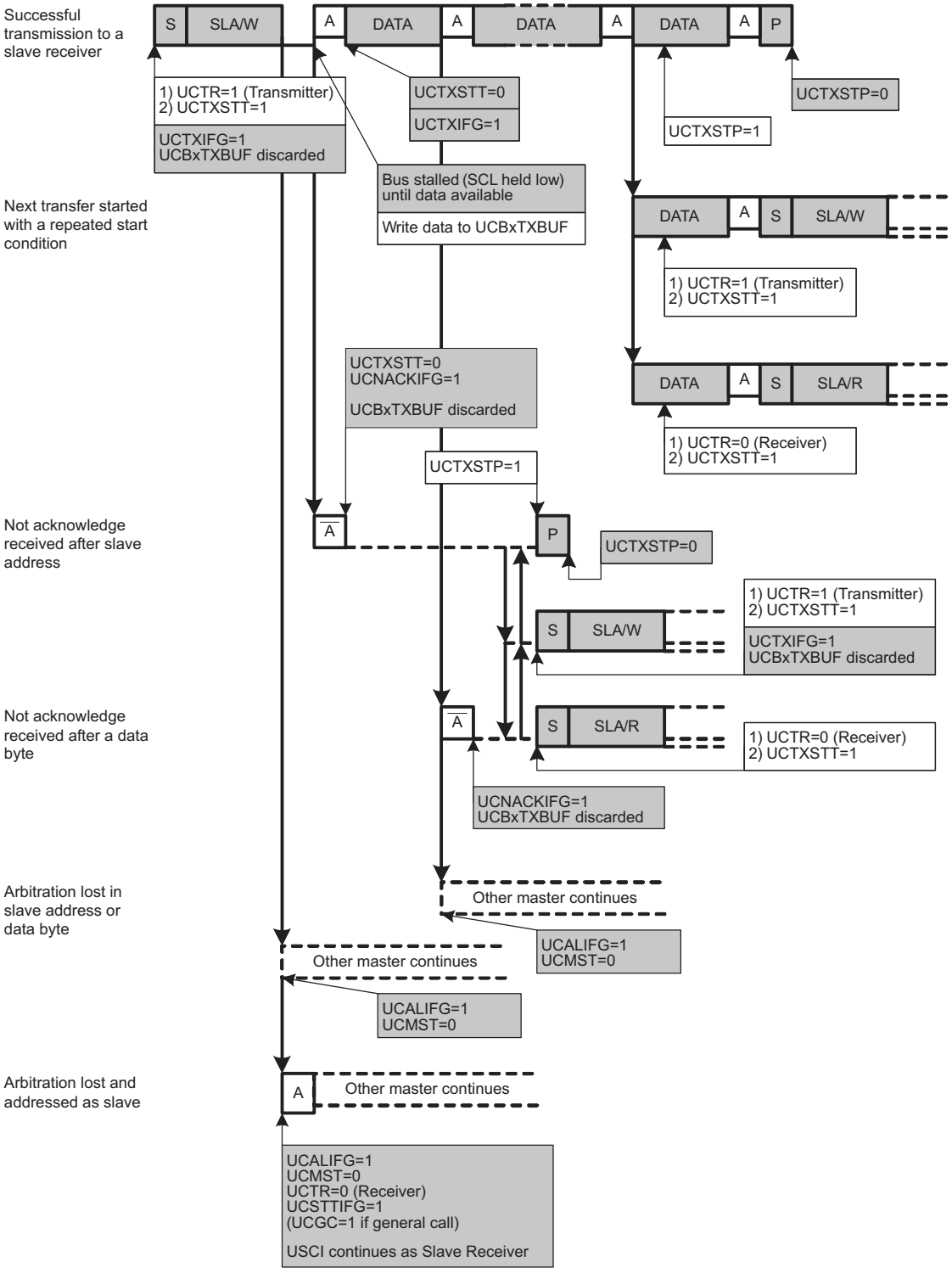


图 **17-12. I²C** 主发送器模式

17.3.5.2.2 I²C 主接收器模式

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 1 UCTR 来选择发送模式、置 1 UCTXSTT 来产生一个起始条件，主接收模式才被初始化。

eUSCI_B 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。一旦完整的地址被发送出去，UCTXSTT 标志就会被清零。

在从器件对地址应答后，将接收到从器件的第一个数据字节并对之做出应答，同时置 1 UCBxRXIFG 标志。数据会从从器件处接收到，只要：

- 没有生成自动停止
- UCTXSTP 位没被置 1
- UCTXSTT 位没被置 1

如果 eUSCI_B 模块生成了一个停止条件，则 UCSTPIFG 会被置 1。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。

如果从器件没有应答发送的地址，则未应答中断标志 UCNACKIFG 会被置 1。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。

一个停止条件要么是由自动停止生成生成的要么是通过置 1 UCTXSTP 位生成的。在一个 NACK 和一个停止条件之后是从从器件处接收下一个字节。如果目前 eUSCI_B 模块正在等待 UCBxRXBUF 被读取，NACK 会立即发生。

如果发送了一个重新启动，可以通过置 1 或清零 UCTR 来把其配置为发送器或接收器，如果需要的话，还可以把一个不同的从器件地址写入 UCBxI2CSA。

图 17-13给出了 I²C 主接收器操作。

注： **无重复起始的连续主器件传输**

在无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。

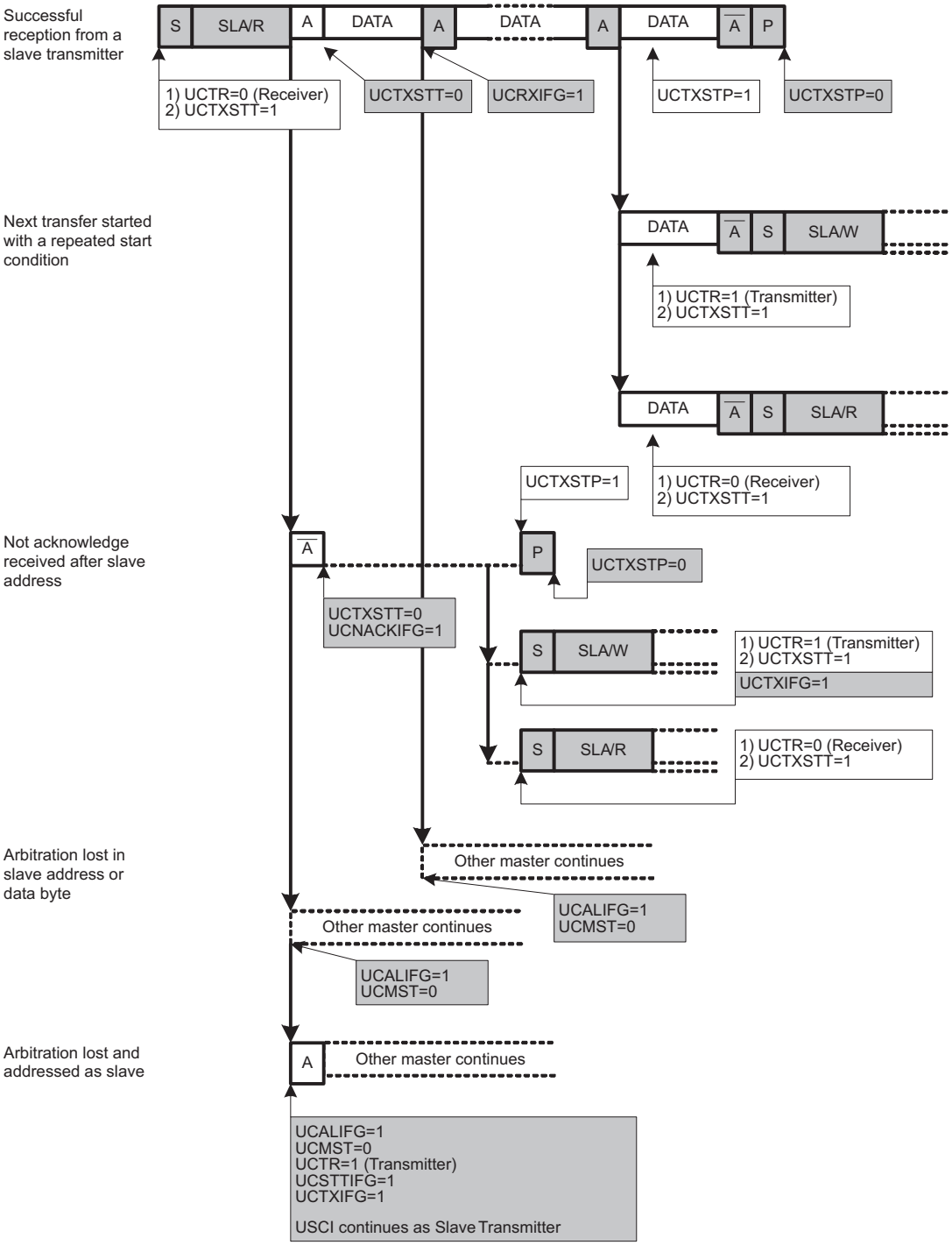


图 **17-13. I²C 主接收器模式**

17.3.5.2.3 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。

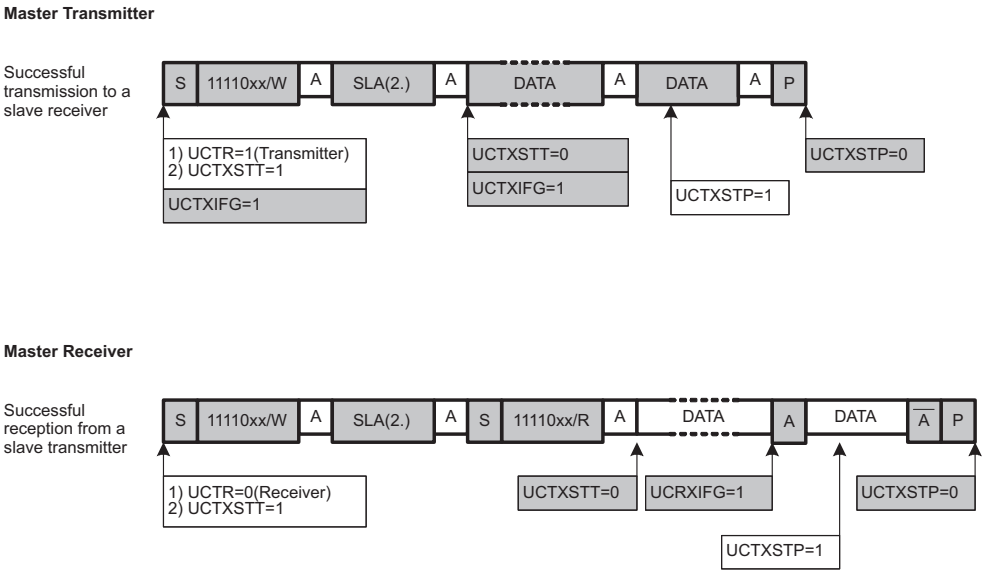


图 17-14. I²C 主器件 10 位寻址模式

17.3.5.3 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发送器将被逻辑低电平的和其竞争的主发送器覆盖。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。失去仲裁的主发送器将切换到从接收器模式，并置 1 仲裁丢失标志 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。

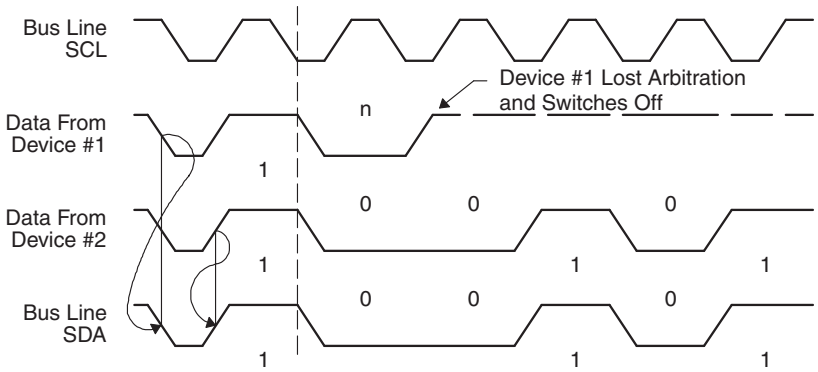


图 17-15. 在两个主发送器之间的仲裁

如果当一个主器件发送了一个重复启动或停止条件而其他主器件仍在发送数据时仲裁程序仍在进行中，就会有一个不确定条件。换言之，以下组合会导致一个未定义条件：

- 主器件 1 发送了一个重复起始条件且主器件 2 发送了一个数据位。
- 主器件 1 发送了一个停止条件且主器件 2 发送了一个数据位。
- 主器件 1 发送了一个重复起始条件且主器件 2 发送了一个停止条件。

17.3.6 干扰滤波

根据 I²C 标准，SDA 和 SCL 线都需要干扰过滤。eUSCI_B 模块提供了 UCGLITx 位来配置该干扰滤波器的长度：

表 17-1. 干扰滤波器长度选择位

UCGLITx	SDA 和 SCL 上的相应的干扰滤波器长度	根据 I²C 标准
00	最大 50ns 长度的脉冲会被过滤掉。	是
01	最大 25ns 长度的脉冲会被过滤掉。	no
10	最大 12.5ns 长度的脉冲会被过滤掉。	no
11	最大 6.25ns 长度的脉冲会被过滤掉。	no

17.3.7 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 eUSCI_B 处于主模式下时，BITCLK 由 eUSCI_B 位时钟发生器提供，同时通过 UCSSELx 位选择时钟源。在从模式下，不使用位时钟发生器且 UCSSELx 位无效。

寄存器 UCBxBR1 和 UCBxBR0 中 UCBRx 的 16 位值是 eUSCI_B 时钟源 BRCLK 的分频因子。在单主模式下，可用的最大位时钟为 $f_{BRCLK}/4$。在多主模式下，最大位时钟为 $f_{BRCLK}/8$。BITCLK 的频率可由以下得到：M

$$f_{位时钟} = f_{BRCLK}/UCBRx$$

生成的 SCL 的最小高电平和低电平周期是:

当 UCBRx 是偶数时，$t_{低电平，最小} = t_{高电平，最小} = (UCBRx/2)/f_{BRCLK}$

当 UCBRx 是奇数时，$t_{低电平，最小} = t_{高电平，最小} = ((UCBRx – 1)/2)/f_{BRCLK}$

为了满足 I²C 最小高低电平周期规格，必须选择 eUSCI_B 时钟源频率和 UCBRx 的预分频器设置。

仲裁过程进行期间，来自不同主器件的时钟必须同步。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。

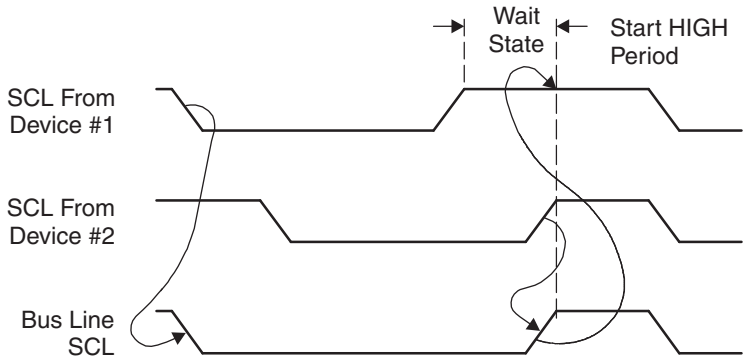


图 17-16. 在仲裁期间两个 I²C 时钟发生器的同步

17.3.7.1 时钟扩展

eUSCI_B 模块支持时钟扩展功能，同时还会利用此特性，如工作模式部分所述。

在下列几种情况下，如果 eUSCI_B 模块已经释放了SCL，可以用 UCSCLLOW 位来检查其他器件是否把 SCL 拉低了。M

- eUSCI_B 充当主器件且一个连接的从器件将 SCL 拉低。
- eUSCI_B 充当主器件，在仲裁进程中其他主器件把 SCL 拉低。

如果 eUSCI_B 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 进行比较，在每一个 SCL 产生上升沿的瞬间 UCSCLLOW 位就有可能被置 1。

17.3.7.2 避免时钟扩展

即使时钟扩展是 I2C 规范的一部分，在一些应用程序中应避免时钟扩展。

以下情况下，由 eUSCI_B 扩展时钟：

- 内部移位寄存器正在等待数据，而 TXIFG 还在挂起
- 内部移位寄存器内数据已满，而 TXIFG 还在挂起
- 仲裁丢失中断挂起
- UCSWACK 被选中 且 UCBxI2COA0 确实是匹配的

为了避免时钟扩展，在际时钟扩展之前要么避免时钟扩展的所有这些情况要么就处理相应的中断标志。

软件必须确保在时钟扩展前及时处理相应的中断。

在从发送器模式下，只在接收到方向位之后置 1 TXIFG；因此，只有很短的时间内为软件写的的 在发生一个时钟扩展前软件写入 TXBUF 的时间很短。这种情况可以通过使用早期的发送中断来补救（参见节 17.3.11.2）。

17.3.7.3 时钟低电平超时

UCCLTOIFG 中断允许软件反应时钟低电平时间是否比规定的长。当一个时钟被一个主器件或从器件扩展了很长时间时，这种情况有可能能检测到。那么，例如，用户可以通过使用 UCSWRST 位来复位 eUSCI_B 模块。

通过 UCCLTO 位使能时钟低电平超时功能。可以为时钟低电平超时功能选择三个预先定义的时间之一。如果时钟低电平时间比 UCCLTO 位定义的时间长且 eUSCI_B 正在有效地进行接收或发送，UCCLTOIFG 就会置 1，且如果 UCCLTOIE 和 GIE 也被置 1 的话就会产生一个中断请求。即使时钟扩展了 UCCLTO 中定义的时间的多倍，UCCLTOIFG 也只置 1 一次。

17.3.8 字节计数器

eUSCI_B 模块支持硬件对接收或发送到的字节进行计数。计数器自动活跃并可对在主模式和从模式中总线上看到的每个字节进行计数。

在独立于以下 ACK 或 NACK 的每个字节的第二比特位置处字节计数器增加。一个起始或重复起始条件会把计数器值复位到零。地址字节不会使计数器增加。如果在传输数据的第一个比特期间失去仲裁，则也会在第二比特位置递增字节计数器。

17.3.8.1 字节计数器中断

如果 UCASTPx= 01 或 10，在主从两种模式中都达到了字节计数器阈值 UCBxTBCNT 时，UCBCNTIFG 将被置 1。 把零写入 UCBxTBCNT 不会产生一个中断。

17.3.8.2 自动停止生成

当 eUSCI_B 模块被配置为一个主器件时，字节计数器可以通过设置 UCASTPx=10 来自动停止的生成。在使用 UCTXSTT 开始发送前，字节计数器阈值 UCBxTBCNT 须设置为要发送或接收的字节数。在已经发送了在 UCBxTBCNT 中配置过的字节数后，eUSCI_B 会自动生成一个停止条件。

如果用户想要发送不带任何数据的从器件地址，那么就不能使用 UCBxTBCNT 。在这种情况下，建议同时将 UCTXSTT 和 UCTXSTP 置 1。

17.3.9 多从器件地址

eUSCI_B 模块支持同时执行多从器件地址的两种不同方式：

- 硬件支持多达 4 个不同的从地址，每个都具有自己的中断标志
- 软件支持多达 2^{10}个不同的从器件地址，它们共用一个中断。

17.3.9.1 多从器件地址寄存器

寄存器 UCBxI2COA0，UCBxI2COA1，UCBxI2COA2，和 UCBxI2COA3 包含四个从器件地址：把多达四个地址寄存器与收到的 7 位或 10 位地址进行比较。通过设置在对应的 UCBxI2COAx 寄存器中的 UCAOEN 位，每个从器件地址都须被激活。如果总线上接收到的地址与多个从器件地址寄存器相匹配，寄存器 UCBxI2COA3 就具有最高优先级。优先级降低了地址寄存器索引号，以便与地址掩码相组合的 UCBxI2COA0 拥有最低优先级。

当其中一个从器件寄存器与在总线上看到的 7 位或 10 位地址相匹配时，该地址会被应答。下列与接收地址相对应的接收 — 或发送 — 中断标志（UCTXIFGx 或 UCRXIFGx）都是最新的。状态更改中断标志位与地址的比较结果无关。根据总线条件对它们进行更新。

17.3.9.2 地址掩码寄存器

在从模式或多主模式下配置 eUSCI_B 时，可以使用地址掩码寄存器。为了激活该功能，须至少清零 UCBxADDMASK 寄存器中地址掩码的一个位。

如果接收的地址与 UCBxI2COA0 中的自身地址在所有未被 UCBxADDMASK 掩码的比特位置上均匹配，则 eUSCI_B 会将接收的地址视为其自身的地址。UCSWACK = 0 时，模块自动发送确认。UCSWACK = 1 时，用户软件必须在 UCSTTIFG 置 1 后评估寄存器中接收的地址。要确认接收的地址，软件必须将 UCTXACK 置 1。

如果从地址与 UCBxI2COA1 至 UCBxI2COA3 中定义的任一已使能从地址匹配时，eUSCI_B 模块在总线上发现该从地址时，还会自动进行确认。

注: **UCSWACK 和从发送器**

用户选择手动确认从地址后，若从器件寻址为发送器，则 TXIFG 置 1。如果软件决定不确认地址，则必须复位 TXIFG0。

17.3.10 在处于低功耗模式中的 I²C 模式中使用 eUSCI_B 模块

eUSCI_B 模块为低功耗模式的使用提供了时钟自动激活。因为器件处于一个低功耗模式，当 eUSCI_B 模块的时钟源是无效的时，必要时，不管时钟源的控制位的设置如何， eUSCI_B 模块都将自动激活。时钟会一直保持激活直到 eUSCI_B 模块回到其空闲状态。eUSCI_B 模块恢复空闲条件后，时钟源的控制会恢复到其控制位的设置。

在 I²C 模式下，由于时钟是由外部主器件提供的，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 从模式中操作 eUSCI_B。接收或发送中断可以将 CPU 从任何一种低功耗模式中唤醒。

17.3.11 I²C 模式下的 eUSCI_B 中断

该 eUSCI_B 只有一个中断向量，该向量是传输，接收和状态变化共享的。

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 1 时，该中断标志会生成一个中断请求。

所有中断标志位被自动清零，但它们需要一起被用户交互清零（例如，读取 UCRXBUF 会清零 UCRXIFGx）。如果用户希望使用一个中断标志，那么在相应的中断被使能之前他就需要确保该标志有正确的状态。

17.3.11.1 I²C 发送中断操作

只要发送器能够接受一个新的字节，那么 UCTXIFG0 中断标志都会被置 1。当被作为一个带有多个从器件地址的从器件运行时，根据之前接收到的地址，UCTXIFGx 标志会被置 1。如果，例如，在寄存器 UCBxI2COA3 中指定的从器件地址确实与在总线上看到的地址相匹配，UCTXIFG3 会表明 UCBxTXBUF 已经准备好接受一个新的字节。

当在带有自动停止生成的主模式下运行时 (UCASTPx = 10)，UCTXIFG0 会被和在 UCBxTBCNT 中定义的次数一样被多次置 1。

如果 UCTXIEx 和 GIE 也被置 1，就会产生一个中断请求。如果 UCBxTXBUF 中发生了一个写入或 UCALIFG 被清零了，UCTXIFGx 就会被自动复位。当发生以下情况时，UCTXIFGx 会被置 1：

- 主模式：UCTXSTT 被用户置 1
- 从模式：接收到自身的地址 (UCETXINT = 0) 或接收到起始 (UCETXINT = 1)

在一个 PUC 后或当 UCSWRST= 1 时，UCTXIEx 被置 1。

17.3.11.2 早期 I²C 发送中断

当一个起始条件被发送且 eUSCI_B 被配置作为一个从器件时，置 1 UCETXINT 会引起 UCTXIFG0 被自动发送。在这种情况下，不允许使能其他从器件地址 UCBxI2COA1-UCBxI2COA3。在检测到从器件地址匹配后，当 UCTXIFG0 被发送出去时，这也就为软件提供了更多的时间来处理与正常情况比较过的 UCTXIFG0。 UCTXIFG0 被置 1 且之后无从器件地址匹配发生的这种情况需要在软件中进行处理。建议使用字节计数器来处理此种情况。

17.3.11.3 I²C 接收中断操作

当接收到一个字节并被装载到 UCBxRXBUF 中时，UCRXIFG0 中断标志被置 1。当被作为一个带有多个从器件地址的从器件运行时，根据之前接收到的地址，UCRXIFGx 标志会被置 1。

如果 UCRXIEx 和 GIE 也被置 1，就会产生一个中断请求。在一个 PUC 信号后或当 UCSWRST= 1 时，UCRXIFGx 和 UCRXIEx 会被复位。当 UCxRXBUF 被读取时，UCRXIFGx 会自动复位。

17.3.11.4 I²C 状态更改中断操作

表 17-2描述了 I²C 状态更改中断标志。

表 17-2. I²C 状态更改中断标志

中断标志	中断条件
UCALIFG	仲裁丢失中断。在两个或两个以上发送器同时开始发送数据时，或者是当 eUSCI_B 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，就可能会发生仲裁丢失。当仲裁丢失时，UCALIFG 标志被置 1。当 UCALIFG 被置 1 时，UCMST 位被清零且 I²C 模块变成了一个从器件。
UCNACKIFG	不确认中断。在需要确认但未收到时，此标志置 1。UCNACKIFG 只被用在主模式中。
UCCLTOIFG	时钟低电平超时。如果该时钟比 UCCLTO 位定义的低电平时间长，则该中断标志位被置 1。
UCBIT9IFG	eUSCI_B 每次发送一字节数据的第九个时钟周期都会生成此中断标志。这使用户能够根据需要在软件中跟踪 I²C 通信。发送地址信息时，UCBIT9IFG 不会置 1。
UCBCNTIFG	字节计数器中断。当字节计数器的值达到 UCBxTBCNT 和 UCASTPx = 01 或 10 中定义的值时，该标志就会被置 1。该位允许编组以下通信，特别是如果一个重新启动被发出时。
UCSTTIFG	起始条件检测到中断。当 I²C 模块检测到一个带有其自身地址(1)的起始条件时，该标志会被置 1。UCSTTIFG 只用于从模式。
UCSTPIFG	停止条件检测到中断。当在总线上 I²C 模块检测到一个停止条件时，该标志会被置 1。UCSTPIFG 用于从模式和主模式。

(1) 如果用到了地址掩码寄存器的话，地址评估也包括该寄存器。

17.3.11.5 UCBxIV，中断向量发生器

eUSCI_B 中断标志被优先化并被结合以便发起一个中断向量。中断向量寄存器 UCBxIV 用于确定哪个标志要求了一个中断。被使能的具有最高优先级的中断在 UCBxIV 中生成了一个数，该数可被评估或是被添加到 PC 中，以便自动进入相应的软件程序中。被禁用的中断不影响 UCBxIV 的值。

UCBxIV 寄存器的读取访问会自动复位最高挂起中断标志。如果有另一个中断标志置 1，则在处理完最初的中断后会立即产生另一个中断。

UCBxIV 寄存器的写入访问会清零所有挂起中断条件和标志。

Example 17-3给出了所推荐的 UCBxIV 的使用。UCBxIV 的值被加入 PC 以便自动跳转到相应的程序中。为 eUSCI0_B 给出了示例。

Example 17-3. UCBxIV 软件示例

```
#pragma vector = USCI_B0_VECTOR __interrupt void USCI_B0_ISR(void) {
    switch(__even_in_range(UCB0IV,0x1e))    {
        case 0x00:     // Vector 0: No interrupts
                    break;
        case 0x02: ... // Vector 2: ALIFG
                    break;
        case 0x04: ... // Vector 4: NACKIFG
                    break;
        case 0x06: ... // Vector 6: STTIFG
                    break;
        case 0x08: ... // Vector 8: STPIFG
                    break;
        case 0x0a: ... // Vector 10: RXIFG3
                    break;
        case 0x0c: ... // Vector 12: TXIFG3
                    break;
        case 0x0e: ... // Vector 14: RXIFG2
                    break;
        case 0x10: ... // Vector 16: TXIFG2
                    break;
        case 0x12: ... // Vector 18: RXIFG1
                    break;
        case 0x14: ... // Vector 20: TXIFG1
                    break;
        case 0x16: ... // Vector 22: RXIFG0
                    break;
        case 0x18: ... // Vector 24: TXIFG0
                    break;
        case 0x1a: ... // Vector 26: BCNTIFG
                    break;
        case 0x1c: ... // Vector 28: clock low time-out
                    break;
        case 0x1e: ... // Vector 30: 9th bit
                    break;
        default:   break;
    }
}
```

17.4 eUSCI_B I2C 寄存器

表 17-3中列出了适用于 I2C 模式的 eUSCI_B 寄存器和它们的地址偏移。有关寄存器的基址，请参见具体器件的数据表。

表 17-3. eUSCI_B 寄存器

偏移量	首字母缩写词	寄存器名称	类型	访问	复位	部分
00h	UCBxCTLW0	eUSCI_Bx 控制字 0	读取/写入	字	01C1h	17.4.1 节
00h	UCBxCTL1	eUSCI_Bx 控制 1	读取/写入	字节	C1h	
01h	UCBxCTL0	eUSCI_Bx 控制 0	读取/写入	字节	01h	
02h	UCBxCTLW1	eUSCI_Bx 控制字 1	读取/写入	字	0000h	17.4.2 节
06h	UCBxBRW	eUSCI_Bx 位速率控制字	读取/写入	字	0000h	17.4.3 节
06h	UCBxBR0	eUSCI_Bx 位速率控制字 0	读取/写入	字节	00h	
07h	UCBxBR1	eUSCI_Bx 位速率控制字 1	读取/写入	字节	00h	
08h	UCBxSTATW	eUSCI_Bx 状态字	读取	字	0000h	17.4.4 节
08h	UCBxSTAT	eUSCI_Bx 状态	读取	字节	00h	
09h	UCBxBCNT	eUSCI_Bx 字节计数器寄存器	读取	字节	00h	
0Ah	UCBxTBCNT	eUSCI_Bx 字节计数器阈值寄存器	读/写	字	00h	17.4.5 节
0Ch	UCBxRXBUF	eUSCI_Bx 接收缓冲器	读取/写入	字	00h	17.4.6 节
0Eh	UCBxTXBUF	eUSCI_Bx 发送缓冲器	读取/写入	字	00h	17.4.7 节
14h	UCBxI2COA0	eUSCI_Bx I2C 自有地址 0	读取/写入	字	0000h	17.4.8 节
16h	UCBxI2COA1	eUSCI_Bx I2C 自有地址 1	读取/写入	字	0000h	17.4.9 节
18h	UCBxI2COA2	eUSCI_Bx I2C 自有地址 2	读取/写入	字	0000h	17.4.10 节
1Ah	UCBxI2COA3	eUSCI_Bx I2C 自有地址 3	读取/写入	字	0000h	17.4.11 节
1Ch	UCBxADDRX	eUSCI_Bx 已接收地址寄存器	读取	字		17.4.12 节
1Eh	UCBxADDMASK	eUSCI_Bx 地址掩码寄存器	读取/写入	字	03FFh	17.4.13 节
20h	UCBxI2CSA	eUSCI_Bx I2C 从地址	读取/写入	字	0000h	17.4.14 节
2Ah	UCBxIE	eUSCI_Bx 中断使能	读取/写入	字	0000h	17.4.15 节
2Ch	UCBxIFG	eUSCI_Bx 中断标志	读取/写入	字	2A02h	17.4.16 节
2Eh	UCBxIV	eUSCI_Bx 中断矢量	读取	字	0000h	17.4.17 节

17.4.1 UCBxCTLW0 寄存器

eUSCI_Bx 控制字寄存器 0

图 17-17. UCBxCTLW0 寄存器

15	14	13	12	11	10	9	8
UCA10	UCSLA10	UCMM	保留	UCMST	UCMODEx		UCSYNC
rw - 0	rw - 0	rw - 0	r0	rw - 0	rw - 0	rw - 0	r1
7	6	5	4	3	2	1	0
UCSSELx		UCTXACK	UCTR	UCTXNACK	UCTXSTP	UCTXSTT	UCSWRST
rw-1	rw - 1	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 1

只在 UCSWRST = 1 时修改。

表 17-4. UCBxCTLW0 寄存器说明

位	字段	类型	复位	说明
15	UCA10	RW	0h	自身寻址模式选择。 只在 UCSWRST = 1 时修改。 0b = 自有地址为一个 7 位地址。 1b = 自有地址为一个 10 位地址。
14	UCSLA10	RW	0h	从器件寻址模式选择 0b = 寻址具有 7 位地址的从器件 1b = 寻址具有 10 位地址的从器件
13	UCMM	RW	0h	多主器件环境选择。 只在 UCSWRST = 1 时修改。 0b = 单一主器件环境。该系统内没有别的主器件。地址比较单元被禁用。 1b = 多主环境
12	保留	R	0h	保留
11	UCMST	RW	0h	主模式选择。在多主环境中的某个主器件失去仲裁时 (UCMM = 1)，UCMST 位将自动清零，相应模块用作从器件。 0b = 从模式 1b = 主模式
10-9	UCMODEx	RW	0h	eUSCI_B 模式。当 UCSYNC=1 时，UCMODEx 位选择同步模式。 只在 UCSWRST = 1 时修改。 00b = 3 引脚 SPI 01b = 4 引脚 SPI（STE = 1 时使能主器件或从器件） 10b = 4 引脚 SPI（STE = 1 时使能主器件或从器件） 11b = I^2C 模式
8	UCSYNC	RW	1h	同步模式使能。对于 eUSCI_B 一直读取和写入为 1 的情况。
7-6	UCSSELx	RW	3h	eUSCI_B 时钟源选择。这些位选择 BRCLK 时钟源。 这些位在从模式中被忽略。 只在 UCSWRST = 1 时修改。 00b = UCLKI 01b = 特定于器件 10b = SMCLK 11b = SMCLK
5	UCTXACK	RW	0h	在已启用地址掩码寄存器的从模式中发送 ACK 条件。在 UCSTTIFG 已经被置 1 后，用户需要置 1或复位 UCTXACK 标志以继续使用 I2C 协议。这个时钟在 UCBxCTL1 寄存器已被写入前被延展。这个位在 ACK 已经被发出后被自动清零。 0b = 不确认从器件地址 1b = 确认从器件地址
4	UCTR	RW	0h	发送器/接收器 0b = 接收器 1b = 发送器

表 **17-4. UCBxCTLW0 寄存器说明 (continued)**

位	字段	类型	复位	说明
3	UCTXNACK	RW	0h	发送一个 NACK。在一个 NACK 发送完毕后，UCTXNACK 自动复位。只适用于从接收器模式。 0b = 正常确认 1b = 生成 NACK
2	UCTXSTP	RW	0h	在主模式下发送停止条件。在从模式下忽略。在主接收器模式下，一个 NACK 位于停止条件之前。在停止生成后，UCTXSTP 被自动清零。如果自动 UCASTPx 不为 01 或 10，这个位是一个无关位。 0b = 无停止生成 1b = 生成停止
1	UCTXSTT	RW	0h	在主模式下发送起始条件。在从模式下忽略。在主接收器模式下，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。在从模式下忽略。 0b = 不生成起始条件 1b = 生成起始条件
0	UCSWRST	RW	1h	软件复位使能。 只在 UCSWRST = 1 时修改。 0b = 被禁用。eUSCI_B 被释放用于运行。 1b = 被启用。eUSCI_B 逻辑保持在复位状态。

17.4.2 UCBxCTLW1 寄存器

eUSCI_Bx 控制字寄存器 1

图 17-18. UCBxCTLW1 寄存器

15	14	13	12	11	10	9	8
保留							UCETXINT
r0	r0	r0	r0	r0	r0	r0	rw-0
7	6	5	4	3	2	1	0
UCCLTO		UCSTPNACK	UCSWACK	UCASTPx		UCGLITx	
rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0

只在 UCSWRST = 1 时修改。

表 17-5. UCBxCTLW1 寄存器说明

位	字段	类型	复位	说明
15-9	保留	R	0h	保留
8	UCETXINT	RW	0h	早期 UCTXIFG0。只适用于从模式中。当这个位被置 1 时，UCxI2COA1 至 UCxI2COA3 中定义的从地址必须被禁用。 只在 UCSWRST = 1 时修改。 0b = UCTXIFGx 在一个地址与 UCxI2COAx 和表示从器件发送的方向位相匹配后被置 1 1b = 对每个启动条件将 UCTXIFG0 置 1
7-6	UCCLTO	RW	0h	时钟低电平超时功能选择。 只在 UCSWRST = 1 时修改。 00b = 禁止时钟低电平超时计数器 01b = 135 000 MODCLK 周期（大约 28ms） 10b = 150 000 MODCLK 周期（大约 31ms） 11b = 165 000 MODCLK 周期（大约 34ms）
5	UCSTPNACK	RW	0h	UCSTPNACK 位还能够使 eUSCI_B 主器件确认主接收器模式中的最后一位。这不符合 I2C 技术规范并且应该只用于从器件，它在一个固定的数据包长度后自动释放 SDA。 只在 UCSWRST = 1 时修改。 0b = 配置为主器件时，在停止条件前发送一个不确定（符合 I2C 标准） 1b = 配置为主接收器时 eUSCI_B 确认所有字节
4	UCSWACK	RW	0h	这个位的使用可选择是由 eUSCI_B 模块触发地址 ACK 的发送还是由软件控制。 0b = 从器件的地址确认由 eUSCI_B 模块控制 1b = 用户需要发布 UCTXACK 来触发地址 ACK 的发送
3-2	UCASTPx	RW	0h	自动停止条件生成。在从模式中，只有 UCBCNTIFG 可用。 只在 UCSWRST = 1 时修改。 00b = 无自动停止生成。停止条件在用户将 UCTXSTP 位置 1 后生成。UCBxTBCNT 内的值无关。 01b = 字节计数器值达到 UCBxTBCNT 中定义的阈值时 UCBCNTIFG 将置 1。 10b = 一个停止条件在字节计数器值达到 UCBxTBCNT 时自动生成。UCBCNTIFG 在字节计数器达到阈值时被置 1。 11b = 保留
1-0	UCGLITx	RW	0h	去毛刺脉冲时间 00b = 50ns 01b = 25ns 10b = 12.5ns 11b = 6.25ns

17.4.3 UCBxBRW 寄存器

eUSCI_Bx 比特率控制字寄存器

图 17-19. UCBxBRW 寄存器

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

只在 UCSWRST = 1 时修改。

表 17-6. UCBxBRW 寄存器说明

位	字段	类型	复位	说明
15-0	UCBRx	RW	0h	位时钟预分频器。 只在 UCSWRST = 1 时修改。

17.4.4 UCBxSTATW

eUSCI_Bx 状态字寄存器

图 17-20. UCBxSTATW 寄存器

15	14	13	12	11	10	9	8
UCBCNTx							
r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
7	6	5	4	3	2	1	0
保留	UCSCLLOW	UCGC	UCBBUSY	保留			
r0	r-0	r-0	r-0	r-0	r0	r0	r0

表 17-7. UCBxSTATW 寄存器说明

位	字段	类型	复位	说明
15-8	UCBCNTx	R	0h	硬件字节计数器值。对这个寄存器的读取返回自最后一个起始或重新开始在 I2C 总线上所接收到或发送出的字节数。这寄存器的同步未完成。当在第一个位位置期间读取 UCBxBCNT 时，会发生一个故障回读。
7	保留	R	0h	保留
6	UCSCLLOW	R	0h	SCL 低电平 0b = SCL 没被保持在低电平 1b = SCL 被保持在低电平
5	UCGC	R	0h	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。 0b = 没有接收到常规调用地址 1b = 接收到常规调用地址
4	UCBBUSY	R	0h	总线忙 0b = 总线未激活 1b = 总线忙
3-0	保留	R	0h	保留

17.4.5 UCBxTBCNT 寄存器

eUSCI_Bx 字节计数器阈值寄存器

图 17-21. UCBxTBCNT 寄存器

15	14	13	12	11	10	9	8
保留							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
UCTBCNTx							
rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0

只在 UCSWRST = 1 时修改。

表 17-8. UCBxTBCNT 寄存器说明

位	字段	类型	复位	说明
15-8	保留	R	0h	保留
7-0	UCTBCNTx	RW	0h	字节计数器阈值被用来设定在多少个 I2C 数据字节后应该发生自动停止或 UCSTPIFG。这个值只在 UCASTPx 不为 00 时才被评估。 只在 UCSWRST = 1 时修改。

17.4.6 UCBxRXBUF 寄存器

eUSCI_Bx 接收缓冲寄存器

图 17-22. UCBxRXBUF 寄存器

15	14	13	12	11	10	9	8
保留							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
UCRXBUFx							
r	r	r	r	r	r	r	r

表 17-9. UCBxRXBUF 寄存器说明

位	字段	类型	复位	说明
15-8	保留	R	0h	保留
7-0	UCRXBUFx	R	0h	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。对 UCBxRXBUF 的读取将复位 UCRXIFGx 标志。

17.4.7 UCBxTXBUF

eUSCI_Bx 发送缓冲寄存器

图 17-23. UCBxTXBUF 寄存器

15	14	13	12	11	10	9	8
保留							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
UCTXBUFx							
rw	rw	rw	rw	rw	rw	rw	rw

表 17-10. UCBxTXBUF 寄存器说明

位	字段	类型	复位	说明
15-8	保留	R	0h	保留
7-0	UCTXBUFx	RW	0h	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以及已经发出的数据。对发送数据缓冲器的写入将清除 UCTXIFGx 标志。

17.4.8 UCBxI2COA0 寄存器

eUSCI_Bx I2C 自有地址 0 寄存器

图 **17-24. UCBxI2COA0** 寄存器

15	14	13	12	11	10	9	8
UCGCEN	保留				UCOAEN	I2COA0	
rw-0	r0	r0	r0	r0	rw - 0	rw - 0	rw - 0
7	6	5	4	3	2	1	0
I2COA0							
rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0

只在 UCSWRST = 1 时修改。

表 **17-11. UCBxI2COA0** 寄存器说明

位	字段	类型	复位	说明
15	UCGCEN	RW	0h	常规调用地址使能。这个位只在 UCBxI2COA0 中可用。 只在 UCSWRST = 1 时修改。 0b = 不响应一个常规调用 1b = 响应一个常规调用
14-11	保留	R	0h	保留
10	UCOAEN	RW	0h	自有地址使能寄存器。借助于这个寄存器，能够选择是否评估与这个寄存器 UCBxI2COA0 相关的 I2C 从地址。 只在 UCSWRST = 1 时修改。 0b = I2COA0 中定义的从地址被禁用 1b = I2COA0 中定义的从地址被启用
9-0	I2COAx	RW	0h	I2C 自身地址。I2COA0 位包含 eUSCIx_B I2C 控制器的本地地址。此地址为右对齐。在 7 位寻址模式中，位 6 是 MSB，而位 9-7 被忽略。在 10 位寻址模式中，位 9 是 MSB。 只在 UCSWRST = 1 时修改。

17.4.9 UCBxI2COA1 寄存器

eUSCI_Bx I2C 自有地址 1 寄存器

图 17-25. UCBxI2COA1 寄存器

15	14	13	12	11	10	9	8
保留					UCOAEN	I2COA1	
rw-0	r0	r0	r0	r0	rw - 0	rw - 0	rw - 0
7	6	5	4	3	2	1	0
I2COA1							
rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0

只在 UCSWRST = 1 时修改。

表 17-12. UCBxI2COA1 寄存器说明

位	字段	类型	复位	说明
15-11	保留	R	0h	保留
10	UCOAEN	RW	0h	自有地址使能寄存器。借助于这个寄存器，能够选择是否评估与这个寄存器 UCBxI2COA1 相关的 I2C 从地址。 只在 UCSWRST = 1 时修改。 0b = I2COA1 中定义的从地址被禁用 1b = I2COA1 中定义的从地址被启用
9-0	I2COA1	RW	0h	I2C 自身地址。I2COAx 位包含 eUSCIx_B I2C 控制器的本地地址。此地址为右对齐。在 7 位寻址模式中，位 6 是 MSB，而位 9-7 被忽略。在 10 位寻址模式中，位 9 是 MSB。 只在 UCSWRST = 1 时修改。

17.4.10 UCBxI2COA2 寄存器

eUSCI_Bx I2C 自有地址 2 寄存器

图 17-26. UCBxI2COA2 寄存器

15	14	13	12	11	10	9	8
保留					UCOAEN	I2COA2	
rw-0	r0	r0	r0	r0	rw - 0	rw - 0	rw - 0
7	6	5	4	3	2	1	0
I2COA2							
rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0

只在 UCSWRST = 1 时修改。

表 17-13. UCBxI2COA2 寄存器说明

位	字段	类型	复位	说明
15-11	保留	R	0h	保留
10	UCOAEN	RW	0h	自有地址使能寄存器。借助于这个寄存器，能够选择是否评估与这个寄存器 UCBxI2COA2 相关的 I2C 从地址。 只在 UCSWRST = 1 时修改。 0b = I2COA2 中定义的从地址被禁用 1b = I2COA2 中定义的从地址被启用
9-0	I2COA2	RW	0h	I2C 自身地址。I2COAx 位包含 eUSCIx_B I2C 控制器的本地地址。此地址为右对齐。在 7 位寻址模式中，位 6 是 MSB，而位 9-7 被忽略。在 10 位寻址模式中，位 9 是 MSB。 只在 UCSWRST = 1 时修改。

17.4.11 UCBxI2COA3 寄存器

eUSCI_Bx I2C 自有地址 3 寄存器

图 17-27. UCBxI2COA3 寄存器

15	14	13	12	11	10	9	8
保留					UCOAEN	I2COA3	
rw-0	r0	r0	r0	r0	rw - 0	rw - 0	rw - 0
7	6	5	4	3	2	1	0
I2COA3							
rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0

只在 UCSWRST = 1 时修改。

表 17-14. UCBxI2COA3 寄存器说明

位	字段	类型	复位	说明
15-11	保留	R	0h	保留
10	UCOAEN	RW	0h	自有地址使能寄存器。借助于这个寄存器，能够选择是否评估与这个寄存器 UCBxI2COA3 相关的 I2C 从地址。 只在 UCSWRST = 1 时修改。 0b = I2COA3 中定义的从地址被禁用 1b = I2COA3 中定义的从地址被启用
9-0	I2COA3	RW	0h	I2C 自身地址。I2COA3 位包含 eUSCIx_B I2C 控制器的本地地址。此地址为右对齐。在 7 位寻址模式中，位 6 是 MSB，而位 9-7 被忽略。在 10 位寻址模式中，位 9 是 MSB。 只在 UCSWRST = 1 时修改。

17.4.12 UCBxADDRX 寄存器

eUSCI_Bx I2C 已接收地址寄存器

图 17-28. UCBxADDRX 寄存器

15	14	13	12	11	10	9	8
保留						ADDRXx	
r-0	r0	r0	r0	r0	r0	r-0	r-0
7	6	5	4	3	2	1	0
ADDRXx							
r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0

表 17-15. UCBxADDRX 寄存器说明

位	字段	类型	复位	说明
15-10	保留	R	0h	保留
9-0	ADDRXx	R	0h	已接收地址寄存器。这个寄存器包含总线上最后接收到的从地址。通过使用这个寄存器和地址掩码寄存器，可对多个从地址使用同一 eUSCI_B 模块的情况作出反应。

17.4.13 UCBxADDMASK 寄存器

eUSCI_Bx I2C 地址掩码寄存器

图 17-29. UCBxADDMASK 寄存器

15	14	13	12	11	10	9	8
保留						ADDMASKx	
r-0	r0	r0	r0	r0	r0	rw-1	rw-1
7	6	5	4	3	2	1	0
ADDMASKx							
rw-1	rw-1	rw-1	rw-1	rw-1	rw-1	rw-1	rw-1

只在 UCSWRST = 1 时修改。

表 17-16. UCBxADDMASK 寄存器说明

位	字段	类型	复位	说明
15-10	保留	R	0h	保留
9-0	ADDMASKx	RW	3FFh	地址掩码寄存器。通过将自身地址的相应位清零，这个位在将总线上的地址与自身地址相比较时是一个无关位。使用这个方法，可对多个从器件地址作出反应。当 ADDMASKx 的所有位被置 1时，地址掩码特性失效。 只在 UCSWRST = 1 时修改。

17.4.14 UCBxI2CSA 寄存器

eUSCI_Bx I2C 从器件地址寄存器

图 17-30. UCBxI2CSA 寄存器

15	14	13	12	11	10	9	8
保留						I2CSAx	
r-0	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

表 17-17. UCBxI2CSA 寄存器说明

位	字段	类型	复位	说明
15-10	保留	R	0h	保留
9-0	I2CSAx	RW	0h	I2C 从器件地址。I2CSAx 位包含了由 eUSCIx_B 模块寻址的外部器件的从器件地址。它仅用于主模式。此地址为右对齐。在 7 位从器件寻址模式中，位 6 是 MSB，而位 9-7 被忽略。在 10 位从器件寻址模式中，位 9 是 MSB。

17.4.15 UCBxIE 寄存器

eUSCI_Bx I2C 中断使能寄存器

图 17-31. UCBxIE 寄存器

15	14	13	12	11	10	9	8
保留	UCBIT9IE	UCTXIE3	UCRXIE3	UCTXIE2	UCRXIE2	UCTXIE1	UCRXIE1
r0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0
7	6	5	4	3	2	1	0
UCCLTOIE	UCBCNTIE	UCNACKIE	UCALIE	UCSTPIE	UCSTTIE	UCTXIE0	UCRXIE0
rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0

表 17-18. UCBxIE 寄存器说明

位	字段	类型	复位	说明
15	保留	R	0h	保留
14	UCBIT9IE	RW	0h	位位置 9 中断使能 0b = 中断被禁用 1b = 中断被启用
13	UCTXIE3	RW	0h	发送中断使能 3 0b = 中断被禁用 1b = 中断被启用
12	UCRXIE3	RW	0h	接收中断使能 3 0b = 中断被禁用 1b = 中断被启用
11	UCTXIE2	RW	0h	发送中断使能 2 0b = 中断被禁用 1b = 中断被启用
10	UCRXIE2	RW	0h	接收中断使能 2 0b = 中断被禁用 1b = 中断被启用
9	UCTXIE1	RW	0h	发送中断使能 1 0b = 中断被禁用 1b = 中断被启用
8	UCRXIE1	RW	0h	接收中断使能 1 0b = 中断被禁用 1b = 中断被启用
7	UCCLTOIE	RW	0h	时钟低电平超时中断使能。 0b = 中断被禁用 1b = 中断被启用
6	UCBCNTIE	RW	0h	字节计数器中断使能。 0b = 中断被禁用 1b = 中断被启用
5	UCNACKIE	RW	0h	不确认中断使能。 0b = 中断被禁用 1b = 中断被启用
4	UCALIE	RW	0h	仲裁丢失中断使能 0b = 中断被禁用 1b = 中断被启用
3	UCSTPIE	RW	0h	停止条件中断使能 0b = 中断被禁用 1b = 中断被启用

表 **17-18. UCBxIE 寄存器说明 (continued)**

位	字段	类型	复位	说明
2	UCSTTIE	RW	0h	起始条件中断使能 0b = 中断被禁用 1b = 中断被启用
1	UCTXIE0	RW	0h	发送中断使能 0 0b = 中断被禁用 1b = 中断被启用
0	UCRXIE0	RW	0h	接收中断使能 0 0b = 中断被禁用 1b = 中断被启用

17.4.16 UCBxIFG 寄存器

eUSCI_Bx I2C 中断标志寄存器

图 17-32. UCBxIFG 寄存器

15	14	13	12	11	10	9	8
保留	UCBIT9IFG	UCTXIFG3	UCRXIFG3	UCTXIFG2	UCRXIFG2	UCTXIFG1	UCRXIFG1
r0	rw - 0	rw - 1	rw - 0	rw - 1	rw - 0	rw - 1	rw - 0
7	6	5	4	3	2	1	0
UCCLTOIFG	UCBCNTIFG	UCNACKIFG	UCALIFG	UCSTPIFG	UCSTTIFG	UCTXIFG0	UCRXIFG0
rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 0	rw - 1	rw - 0

表 17-19. UCBxIFG 寄存器说明

位	字段	类型	复位	说明
15	保留	R	0h	保留
14	UCBIT9IFG	RW	0h	位位置9 中断标志 0b = 无中断挂起 1b = 中断挂起
13	UCTXIFG3	RW	1h	eUSCI_B 发送中断标志 3。在从模式中，如果在 UCBxI2COA3 内定义的从地址在总线上的同一数据帧内，当 UCBxTXBUF 为空时，UCTXIFG3 被置 1。 0b = 无中断挂起 1B = 中断挂起
12	UCRXIFG3	RW	0h	接收中断标志 2。在从模式中，如果在 UCBxI2COA2 内定义的从地址在总线上的同一数据帧内，当 UCBxRXBUF 已经接收到一个完整字节时，UCRXIFG2 被置 1。 0b = 无中断挂起 1B = 中断挂起
11	UCTXIFG2	RW	0h	eUSCI_B 发送中断标志 2。在从模式中，如果在 UCBxI2COA2 内定义的从地址在总线上的同一数据帧内，当 UCBxTXBUF 为空时，UCTXIFG2 被置 1。 0b = 无中断挂起 1b = 中断挂起
10	UCRXIFG2	RW	0h	接收中断标志 2。在从模式中，如果在 UCBxI2COA2 内定义的从地址在总线上的同一数据帧内，当 UCBxRXBUF 已经接收到一个完整字节时，UCRXIFG2 被置 1。 0b = 无中断挂起 1B = 中断挂起
9	UCTXIFG1	RW	1h	eUSCI_B 发送中断标志 1。在从模式中，如果在 UCBxI2COA1 内定义的从地址在总线上的同一数据帧内，当 UCBxTXBUF 为空时，UCTXIFG1 被置 1。 0b = 无中断挂起 1B = 中断挂起
8	UCRXIFG1	RW	0h	接收中断标志 1。在从模式中，如果在 UCBxI2COA1 内定义的从地址在总线上的同一数据帧内，当 UCBxRXBUF 已经接收到一个完整字节时，UCRXIFG1 被置 1。 0b = 无中断挂起 1b = 中断挂起
7	UCCLTOIFG	RW	0h	时钟低电平超时中断标志 0b = 无中断挂起 1b = 中断挂起
6	UCBCNTIFG	RW	0h	字节计数器中断标志。当时用这个中断时，用户需要确保有足够的处理带宽（请见字节计数器中断一节）。 0b = 无中断挂起 1b = 中断挂起
5	UCNACKIFG	RW	0h	不应答接收到的中断标志。这个标志只有当运行在主模式下时才被更新。 0b = 无中断挂起 1b = 中断挂起

表 17-19. UCBxIFG 寄存器说明 (continued)

位	字段	类型	复位	说明
4	UCALIFG	RW	0h	仲裁丢失中断标志 0b = 无中断挂起 1b = 中断挂起
3	UCSTPIFG	RW	0h	停止条件中断标志。 0b = 无中断挂起 1b = 中断挂起
2	UCSTTIFG	RW	0h	启动条件中断标志 0b = 无中断挂起 1b = 中断挂起
1	UCTXIFG0	RW	0h	eUSCI_B 发送中断标志 0。 在主模式或从模式中，如果在 UCBxI2COA0 内定义的从地址在总线上的同一数据帧内，当 UCBxTXBUF 为空时，UCTXIFG0 被置 1。 0b = 无中断挂起 1b = 中断挂起
0	UCRXIFG0	RW	0h	eUSCI_B 接收中断标志 0。 在从模式中，如果在 UCBxI2COA0 内定义的从地址在总线上的同一数据帧内，当 UCBxRXBUF 已经接收到一个完整字节时，UCRXIFG0 被置 1。 0b = 无中断挂起 1b = 中断挂起

17.4.17 UCBxIV 寄存器

eUSCI_Bx 中断矢量寄存器

图 17-33. UCBxIV 寄存器

15	14	13	12	11	10	9	8
UCIVx							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
UCIVx							
r0	r0	r0	r0	r-0	r-0	r-0	r0

表 17-20. UCBxIV 寄存器说明

位	字段	类型	复位	说明
15-0	UCIVx	R	0h	eUSCI_B 中断矢量值。它生成一个值，这个值可被用作针对快速中断处理例程操作的地址偏移。对这个寄存器的写入将清除所有挂起的中断标志。 00h = 无中断挂起 02h = 中断源：仲裁丢失；中断标志：UCALIFG；中断优先级：最高 04h = 中断源：无确认；中断标志：UCNACKIFG 06h = 中断源：接收到开始条件；中断标志：UCSTTIFG 08h = 中断源：接收到停止条件；中断标志：UCSTPIFG 0Ah = 中断源：接收到从器件 3 数据；中断标志：UCRXIFG3 0Ch = 中断源：从器件 3 发送缓冲器空；中断标志：UCTXIFG3 0Eh = 中断源：接收到从器件 2 数据；中断标志：UCRXIFG2 10h = 中断源：从器件 2 发送缓冲器空；中断标志：UCTXIFG2 12h = 中断源：接收到从器件 1 数据；中断标志：UCRXIFG1 14h = 中断源：从器件 1 发送缓冲器空；中断标志：UCTXIFG1 16h = 中断源：数据被接收到；中断标志：UCRXIFG0 18h = 中断源：发送缓冲器空；中断标志：UCTXIFG0 1Ah = 中断源：字节计数器错误；中断标志：UCBCNTIFG 1Ch = 中断源：时钟低电平超时；中断标志：UCCLTOIFG 1Eh = 中断源：第九比特位置；中断标志：UCBIT9IFG；中断优先级：最低

嵌入式仿真模块 (EEM)

本章介绍了在所有器件中都包含的嵌入式仿真模块 (EEM)。

Topic **Page**

18.1 嵌入式仿真模块 (EEM) 简介

该系列中的每个器件都包含一个 EEM。通过 4 线制 JTAG 模式或 Spy-Bi-Wire 模式对其进行访问和控制。具体实现取决于具体器件，并在18.3 节和具体器件的数据表中进行介绍。

总的来说，具有以下特性：

- 具有非侵入性执行代码的实时断点控制
- 单步、单步执行入函数和单步执行出函数
- 完全支持所有低功耗模式
- 支持所有系统频率和各种时钟源
- 存储器地址总线 (MAB) 或存储器数据总线 (MDB) 上多达八个（取决于器件）硬件触发信号或断点
- CPU 寄存器写访问时多达两个（取决于器件）硬件触发信号或断点
- MAB，MDB 和 CPU 寄存器访问触发器可被组合在一起组成组多10个(取决于具体器件)复杂硬件触发或断点
- 最多2个(取决于具体器件)周期计数器
- 触发排序(取决于具体器件)
- 用一个集成跟踪缓冲器(取决于具体器件)来存储内部总线和控制信号
- 仿真停止期间在全局器件级或基于每个模块对定时器、通信外设和其它模块进行时钟控制

图 18-1 显示了目前最大的可用 EEM 的简化框图。

有关 EEM 功能如何与 IAR Embedded Workbench ™调试器或与 Code Composer Studio™IDE (CCS) 一起使用的更多详细信息，请参见应用报告*《使用增强型仿真模块进行高级调试》*（文献号：SLAA393），该应用报告在 www.msp430.com 上提供。大多数支持 MSP430 器件的其它调试器都具有相同或相似的功能集。更多详细信息请参阅适用调试器用户指南。

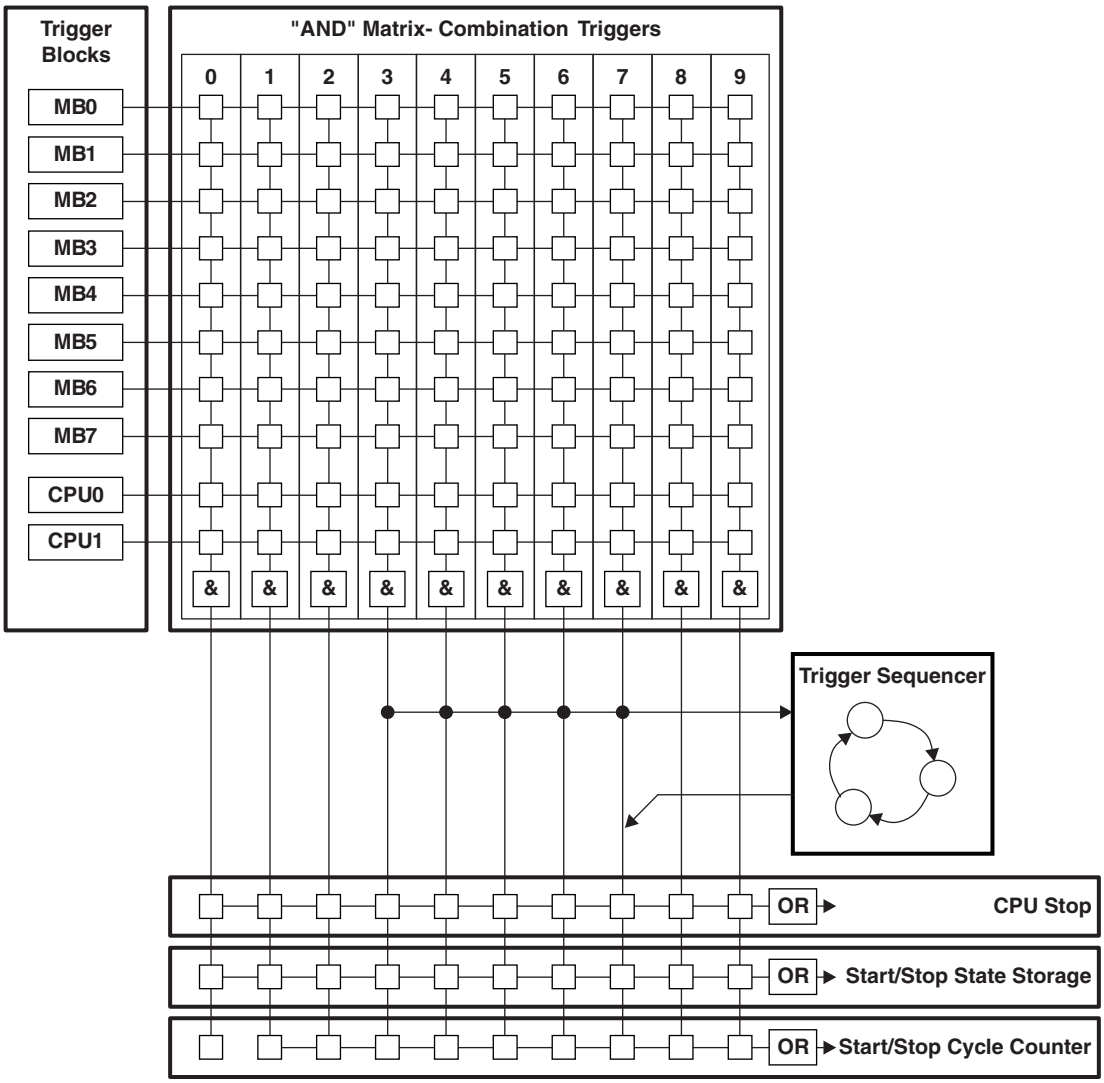


图 **18-1. EEM** 的大型应用

18.2 EEM 构建模块

18.2.1 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检测复杂事件并导致除停止 CPU 以外的多种反应。

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
- 周期计数器

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 上的数据和给定的数据进行比较，每一个存储触发器块都可以被独立选中。根据被执行的 EEM，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。

18.2.2 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，它们可以被旁通。

18.2.3 状态存储（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入性的方式存储 MAB，MDB 和 CPU 控制信号信息（即：读、写或者指令提取）。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。

18.2.4 周期计数器

周期计数器提供一个或两个 64 位计数器来测量 CPU 用来执行特定任务的周期。在某些器件上，可使用触发器对周期计数器的运行进行控制。例如，这可实现条件配置，诸如配置代码的特定部分。

18.2.5 时钟控制

EEM 提供了与器件相关的灵活的时钟控制。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。

18.3 EEM 配置

表 18-1给出了 EEM 配置的概述。实施的配置取决于器件，相关信息请参见具体器件的数据表和以下文档：

《使用具有 CCS 版本 4 的增强型仿真模块 (EEM) 进行高级调试》(SLAA393)

《用于 MSP430 的 IAR 嵌入式工作平台用户指南》（文献号：SLAU138）

《用于 MSP430 的 Code Composer Studio 用户指南》（文献号：SLAU157）

表 18-1. EEM 配置

特性	XS	S	M	L
存储器总线触发	2 （只在 =，≠ 时）	3	5	8
对于以下情况的存储器总线触发屏蔽	1) 低字节 2) 高字节 3) 四个高位	1) 低字节 2) 高字节 3) 四个高位	1) 低字节 2) 高字节 3) 四个高位	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	10
序列发生器	否	否	是	是
状态存储器	否	否	否	是
周期计数器	1	1	1	2 （包括已触发开始或停止）

总的来说，任一器件都包括了以下这些特性：

- 至少两个 MAB 或 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 =，≠）
- 至少两个触发组合寄存器
- 使用 CPU 停止反应的硬件断点
- 至少一个 40 位周期计数器
- 具有模块时钟单独控制的增强型时钟控制

修订历史记录

注：之前版本的页码可能与当前版本有所不同。

Changes from October 3, 2014 to August 17, 2015 **Page**

- 在图 16-1，*eUSCI 框图 – SPI 模式*的 UCSSELx 位复用中，已将 01b 枚举选项由“MODCLK”更改为“特定于器件” 470
- 在表 16-3，*UCAxCTLW0 寄存器说明*的 UCSSELx 位说明中，已将 01b 枚举选项由“MODCLK”更改为“特定于器件” ... 478
- 在图 17-1，*eUSCI_B 框图 – I²C 模式*的 UCSSELx 位复用中，已将 01b 枚举选项由“MODCLK”更改为“特定于器件” ... 493
- 修正了软件复位 (UCSWRST = 1) 条件。 494
- 在表 17-4，*UCBxCTLW0 寄存器说明*的 UCSSELx 位说明中，已将 01b 枚举选项由“MODCLK”更改为“特定于器件” ... 514

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

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

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265