Errata

MSP432E4 SimpleLink™ 微控制器

TEXAS INSTRUMENTS

内容

1 MSP432E4 SimpleLink™ 微控制器

1.1 简介

本文档介绍了 SimpleLink™ MSP432E4 微控制器功能规格的已知例外情况。请注意，某些功能并不适用于该系列中的所有器件，因此并非所有勘误表都适用于您的器件。有关更多详细信息，请参见器件特定数据表。

有关 Arm® Cortex®-M4F CPU 公告的详细信息，请参阅《*Arm® Cortex®-M4F 勘误表*》。

1.2 器件命名规则

为了指明产品开发周期所处的阶段，TI 为所有 MSP432™ MCU 器件和支持工具的器件型号分配了前缀。每个 MSP432 MCU 商用系列产品都具有以下两个前缀之一：MSP 或 XMS（例如 MSP432E411Y）。每个支持工具都具有以下两个前缀中的其中一个：MSP 和 MSPX。这些前缀代表了产品从工程原型（其中 XMS 适用于器件，而 MSPX 适用于工具）直到完全合格的生产器件和工具（其中 MSP 适用于器件和工具）的产品开发演变阶段。

器件开发进化流程：

XMS — 实验器件，不一定代表最终器件的电气规格

MSP — 完全合格的生产器件

支持工具开发演变流程：

MSPX — 尚未完成 TI 内部合格性测试的开发支持工具

MSP — 完全合格的开发支持产品。

XMS 器件和 MSPX 器件开发支持工具在供货时附带如下免责条款：

“开发中的产品用于内部评估用途。”

MSP 器件和 MSP 开发支持工具的特征已得到充分体现，并且器件的质量和可靠性已得到充分证明。TI 的标准保修证书对该器件适用。

预测显示原型器件 (XMS) 的故障率大于标准生产器件。由于这些器件的预计最终使用故障率尚不确定，德州仪器 (TI) 建议不要将它们用于任何生产系统。请仅使用合格的生产器件。

1.3 器件标识

下面是微控制器封装符号化的示例。

MSP432E411Y，212 引脚 ZAD (NFBGA) 封装

```
+-----------------+
|     \TI/        |      \TI/ = TI Logo
| MSP432(TM)      |      (TM) = Trademark Symbol
| E411YT          |      #    = Die Revision
| xxxxxxxxxx      |      O    = Pin 1
| O  REV #        |
+-----------------+
```

MSP432E401Y，128 引脚 PDT (TQFP) 封装

```
+-----------------+
| \TI/ xxxxxxxxx  |      \TI/ = TI Logo
| MSP432          |      #    = Die Revision
| E401YT          |      O    = Pin 1
| REV #           |
| O               |
+-----------------+
```

1.4 勘误表概述

表 1-1 列出了器件勘误表。

表 1-1. 器件勘误表

名称	勘误表标题
	ADC
ADC#13	使用任何 ADC 模拟输入通道进行采样时，引脚 PE3 上可能会出现干扰
ADC#14	前两个 ADC 样本可能不正确
	EPI
EPI#01	使用 EPI 模块中的代码地址空间时，数据读取可能损坏
	GPIO
GPIO#09	在某些情况下，注入 GPIO 引脚 PB0 和 PB1 的噪声会导致高电流消耗
	通用计时器
GPTM#09	当针对 RTC 模式配置时，通用定时器不同步
GPTM#15	在边沿递增模式下达到 MATCH 时，计数器不会立即清零
	休眠
HIB#10	如果 MEMCLR 设置为非零值，篡改事件可能不会清除 HIBDATA 寄存器中的所有位
HIB#16	在清除期间，应用代码可能丢失新的篡改事件
HIB#18	在日历模式下每天可以获得两个匹配
HIB#19	休眠模块复位后，可能无法成功完成对 HIBCTL 寄存器的首次写入
	存储器
MEM#07	在 EEPROM 操作期间，软复位不应生效
MEM#15	任何扇区中的特定闪存位置都不会被擦除
MEM#16	BOOTCFG 在 NW=0 的情况下提交时出现 JTAG 解锁问题
	PWM
PWM#04	PWM 发生器中断只能在中断发生后的 1 个 PWM 时钟周期被清除
PWM#05	具有全局同步的发生器负载可能会导致错误的脉冲宽度
PWM#06	PWM 输出可能生成持续高电平而非低电平，或持续低电平而非高电平
	QEI
QEI#01	使用索引脉冲复位计数器时，QEI 模块中的一个特定初始条件会导致第一个计数的方向误读
	SSI
SSI#03	SSI1 只能在传统模式下使用
SSI#05	SSI 的双模式和四模式下的总线争用
SSI#06	在从机模式中，SSI 接收 FIFO 超时中断的生效时间可能比预期的要早
SSI#07	SSI 发送中断状态位不被锁存
SSI#08	SSI 从器件在双模式和四模式下交换 XDAT0 和 XDAT1
	系统控制
SYSCTL#03	在时钟成功运行后，MOSC 验证电路没有检测到时钟丢失
SYSCTL#18	DIVSCLK 输出的时钟频率与 DIV = 0x0 时的预期频率不同
SYSCTL#24	如果先关闭然后再打开电源域，模块也许没有为访问做好准备
	USB
USB#04	器件发送 SE0 以 响应 USB 总线复位
	看门狗计时器
WDT#08	当使用看门狗定时器 1 时，读取 WDTVALUE 寄存器可能会返回错误值

1.5 勘误说明

ADC#13

使用任何 ADC 模拟输入通道进行采样时，引脚 PE3 上可能会出现干扰

说明

使用任何 ADC 模拟输入通道 (AINx) 进行采样时，PE3 上可能会出现干扰。当 PE3 配置为输入通道时，可能会发生此干扰，并在 ADC 转换结束时发生时，可能会发生此干扰。当配置为 AIN0 时，只要满足指定的源电阻，这些干扰就不会影响 PE3 上的模拟测量。

权变措施

PE3 上的 1kΩ 外部上拉或下拉电阻有助于将干扰幅度降至 200mV 或更低。

ADC#14

前两个 ADC 样本可能不正确

说明

在 xCGCADC 寄存器中启用 ADC 时钟后，捕获的前两个 ADC 样本可能不正确。

权变措施

1. 在启用 ADC 外设时钟后，在初始化 ADC 并启用采样序列发生器之前，使用 SRADC 寄存器复位 ADC 外设。
2. 如果无法通过应用程序完成重新配置，则在处理数据之前丢弃前两个样本。

EPI#01

使用 EPI 模块中的代码地址空间时，数据的读取可能损坏

说明

使用 EPI 地址映射 (EPIADDRMAP) 中的 ECSZ 和 ECADR 字段来指定地址 0x1000.0000 处的外部代码地址空间以用于 EPI 模块。不过，使用该地址空间时，数据读取可能会损坏。

权变措施

无法从 0x1000.0000 地址空间执行代码。可以改用 0x6000.0000 和 0x8000.0000 处的 EPI 地址空间。

此外，从映射到 0x1000.0000 处代码地址空间的 EPI 存储器中读取数据时，应根据读操作的数据大小，将通过指针读取的直接 EPI 存储器替换为调用 EPIWorkaroundWordRead()、EPIWorkaroundHWordRead() 或 EPIWorkaroundByteRead() 函数。同样，在写入 EPI 代码地址空间时，将直接写入替换为调用 EPIWorkaroundWordWrite()、EPIWorkaroundHWordWrite() 或 EPIWorkaroundByteWrite() 函数。这些 API 是新的，可以在 节 1.7 中找到。对于 Keil、IAR、GCC 和 Code Bench，这些函数在 *driverlib* 文件夹中的 epi.h 文件中定义为内联函数。对于不支持此结构的 CCS，应该将它们添加到名为 epi_workaround_ccs.s 的 *\driverlib* 目录中的新文件中，并且此文件应添加到项目中。请注意，节 1.7 中提到的新 DriverLib API 和 CCS 文件均包含在 SimpleLink MSP432 SDK 中。

GPIO#09

在某些情况下，注入 GPIO 引脚 PB0 和 PB1 的噪声会导致高电流消耗

说明

PB0 或 PB1 上的快速转换可在一个或两个引脚与接地之间的低电阻路径上切换，可能导致高电流消耗。

当器件引脚上的信号的上升时间或下降时间（测得的从 10% VDD 到 90% VDD 的时间）长于 2ns 时，便观察到了这种情况。这种情况更有可能发生在高温或嘈杂环境中。当引脚处于输入或输出模式或使用任何引脚多路复用选项时，可能会发生这种情况。

GPIO#09（续）

在某些情况下，注入 GPIO 引脚 PB0 和 PB1 的噪声会导致高电流消耗

如果在引脚配置为输出 GPIO 时引发该情况，则将引脚状态更改为低电平、然后在较低温度下恢复到高电平状态即可解决该情况。

权变措施

1. 请勿使用 PB0 和 PB1。通过一个 1kΩ 电阻器将两者连接到 GND、并将其配置为 GPIO 输入。
2. 如果 PB0 和 PB1 用作 USB0ID 和 USB0VBUS，请参阅 *《MSP432E4 微控制器的系统设计指南》* 的 USB 部分、并确认遵守本文档中的所有指南。

GPTM#09

当配置为 RTC 模式时，通用定时器不同步

说明

在尝试使用 GPTM 同步 (GPTMSYNC) 寄存器同步通用定时器时，如果有定时器配置为 RTC 模式，它们不会同步。

权变措施

无。

GPTM#15

在边沿递增模式下达到 MATCH 时，计数器不会立即复位为 0

说明

当配置为输入边沿计数模式和递增模式时，在计数到匹配值之后，计数器使用一个额外的边沿将定时器复位为 0。因此，在第一次匹配事件发生后，所有后续匹配事件都将在编程的边沿事件数加一之后发生。

权变措施

在软件中，在收到第一个匹配中断后，计入已编程边沿计数中的一个额外边沿。

HIB#10

如果 MEMCLR 设置为非零值，篡改事件可能不会清除 HIBDATA 寄存器中的所有位

说明

如果 HIB 篡改控制 (HIBTPCTL) 寄存器的 MEMCLR 位字段设置为非零值，那么休眠数据 (HIBDATA) 寄存器可能不会清除指定的位。MEMCLR 位字段提供在发生篡改事件时擦除休眠存储器所有内容、上半部分、下半部分或者不擦除休眠存储器的选项。

权变措施

通过设置 TPCLR 位清除篡改实践后，应用程序应将 HIBDATA 寄存器的休眠存储器中的数据清零（将上半部分、下半部分或所有位全部写为零）。

HIB#16

在清除期间，应用代码可能丢失新的篡改事件

说明

清除篡改事件期间，可能丢失新的篡改事件或者篡改日志可能崩溃。 在清除篡改事件时，首先写入 HIB 篡改控制寄存器 (HIBTPCTL) 的防篡改清除 (TPCLR) 位。 该写入操作会在 32.768kHz 时钟的 3 个上升沿完成清除。

权变措施

为防止在这三个休眠时钟周期内丢失篡改事件并将篡改日志恢复至复位状态，必须在 NMI 处理程序中实施权变措施代码，如 节 1.6 所示。节 1.6 中提到的新 DriverLib API 包含在 SimpleLink MSP432E4 SDK 中。

HIB#18 *在日历模式下每天可以获得两个匹配*

说明

将休眠日历控制 (HIBCALCTL) 寄存器的 CAL24 位清除后，RTC 将在 12 小时 AM/PM 制下计数。休眠日历匹配 0 (HIBCALM0) 中的 AM/PM 位指定了该匹配是在 AM 还是在 PM 发生。但是，在确定是否发生匹配时，该位将被忽略。因此，RTC 匹配可能会在一天之内发生两次。

权变措施

在配置 HIBCALM0 寄存器并设置 CAL24 位之前，将匹配时间调整为 24 小时制。或者，当匹配发生时，请检查休眠日历 (HICAL0) 寄存器中的 AM/PM 位来确定匹配是否正确。

HIB#19 *休眠模块复位后，可能无法成功完成对 HIBCTL 寄存器的首次写入*

说明

如果休眠模块复位，那么可能不会对 HIBCTL 寄存器进行初始写入。HIBCTL 寄存器的 WRC 位可能不会置位。

权变措施

休眠模块复位后，检查 WRC 位是否置位，并执行以下操作：

- 如果 WRC 位未在振荡器最大启动时间内置位，那么对休眠模块执行软件复位，然后重试 HIBCTL 写操作。最大振荡器启动时间由数据表"电气特性"一章中"休眠外部振荡器 (XOSC) 输入特性"表内的休眠 XOSC 启动时间参数 TSTART 给出。
- 如果 WRC 位置位但 HIBCTL 写入没有成功，请重试 HIBCTL 写入。

MEM#07 *在 EEPROM 操作期间，软复位不应生效*

说明

如果在 EEPROM 编程或擦除操作期间执行以下任何软复位，则 EEPROM 数据可能损坏：

- 软件复位 (SYSRESREQ)
- EEPROM 模块的软件外设复位
- 看门狗复位（如果在 RESBEHAVCTL 寄存器中配置为系统复位）
- MOSC 故障复位
- BOR 复位（如果在 RESBEHAVCTL 寄存器中配置为系统复位）
- 外部复位（如果在 RESBEHAVCTL 寄存器中配置为系统复位）
- 写入 HSSR 寄存器

权变措施

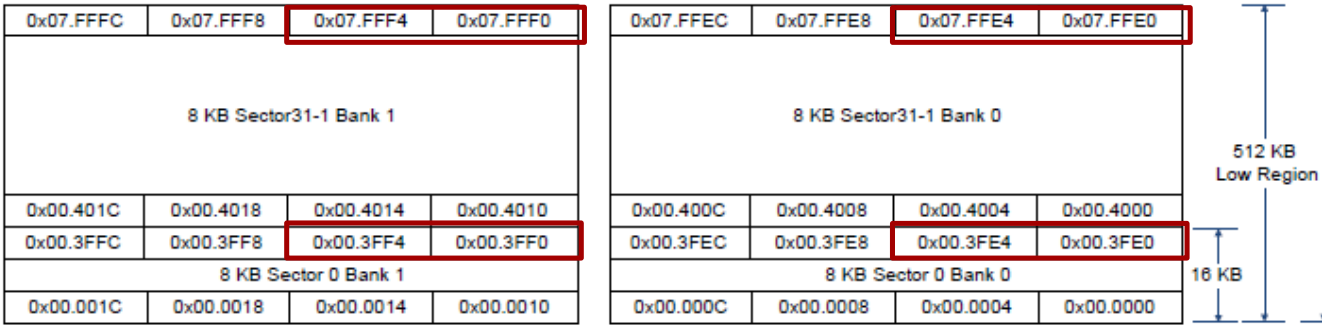
确保在 EEPROM 编程或擦除操作期间，上述任何软复位均不置位。可以在该复位生效之前检查 EEDONE 寄存器的 WORKING 位，以查看是否发生了 EEPROM 编程或擦除操作。使用调试器时可能会发生软复位，在 EEPROM 操作期间应避免软复位。如果时间问题不大，可以使用一个 GPIO 或者休眠模式，将一个诸如看门狗复位的复位映射到一个外部复位。

MEM#15

说明

如果只对组中一个扇区最后一行的前两个字中的一个或两个字进行编程，则擦除整个扇区、批量擦除器件或切换批量擦除不会将闪存擦除回全 1。下图显示了下部 512KB 闪存的扇区 0 和扇区 31 中受影响的字。

***MEM#15*（续）**



权变措施

对扇区组中的任何其他字进行编程以允许扇区擦除功能工作。

备注

切换批量擦除，或批量擦除仍将不起作用。

MEM#16

BOOTCFG 在 Nw=0 的情况下提交时出现 JTAG 解锁问题

说明

将 BOOTCFG 寄存器配置为 NW=0、DBG1=0 和 DBG=1 后，JTAG 调试器访问被锁定并且无法通过一次解锁序列运行来解锁。

权变措施

要解锁器件，必须运行两次解锁序列。

PWM#04

PWM 发生器中断只能在中断发生后的 1 个 PWM 时钟周期被清除

说明

向 PWMxISC 寄存器写入 1 应该会清除下一个系统时钟上的相应发生器中断状态。但是，写入操作将清除下一个 PWM 时钟的发生器中断状态。在下一个 PWM 时钟之前向 PWMxISC 写入来清除中断的任何操作都将被忽略，中断将重新置为有效。

权变措施

中断置为有效后，CPU 必须等待一个 PWM 时钟周期，然后再向 PWMxISC 写入 1 来清零相应的发生器中断状态。PWM 时钟分频器值越大，清除中断的系统延迟越长。

PWM#05

具有全局同步的发生器负载可能会导致错误的脉冲宽度

说明

在这种情况下，发生器定时器仍然可以用旧的值计数，但新的比较器值会被装载，从而产生错误的高电平脉冲宽度。

权变措施

应用程序必须根据中断状态写入新装载值和比较器值。当使用递减计数模式时，清除相应比较器递减计数匹配中断状态位的原始中断状态位，等待再次设置该位，然后更新加载和比较器匹配的新值。

PWM#06 ***PWM 输出可能生成持续高电平而非低电平，或持续低电平而非高电平***

说明

当在上/下双向计数模式下使用 PWM 时，在以下情况下，PWM 发生器可以生成持续高电平而不是低电平，或者持续低电平而不是高电平：

- PWM 发生器与比较器 A 或 B 的递增和递减操作配合使用，以控制 PWM 周期。
- PWM 比较器 A 或 B 从计数 2 或更多更改为 PWM 负载计数值。
- 如果启用反转选项，则会生成持续高电平。
- 如果禁用反转选项，则会生成持续低电平。

权变措施

- 使用 PWM 的向下计数模式
- 当比较器负载寄存器中的 PWM 负载计数值从任何值（1 除外）更改为 PWM 负载计数值时，在进入 PWM 负载计数值之前，始终变为 1。

QEI#01 ***使用索引脉冲复位计数器时，QEI 模块中的一个特定初始条件会导致第一个计数的方向误读***

说明

通过 QEI 控制 (QEICTL) 寄存器的以下配置使用索引脉冲复位计数器：

- SIGMODE 为 0 表示正交模式
- CAPMODE 为 1 表示 PhA 和 PhB 边沿均被计数

和以下初始条件：

- PhA 和 PhB 都为 0
- 下一个正交状态是逆时针方向

QEI 将状态变化解释为顺时针方向的更新，这会导致位置不匹配 2。

权变措施

无。

SSI#03 ***SSI1 只能在传统模式下使用***

说明

在指定的 SSI 模块上，双模式、四模式和高级运行模式无法正常工作。因此，任何受影响的模块都只能用于传统运行。

权变措施

SSI0、SSI2 或 SSI3 可用于双模式、四模式和高级模式运行。对于传统模式运行使用 SSI1。

SSI#05 ***SSI 的双模式和四模式下的总线争用***

说明

当 SSI 配置为双模式或四模式并且在 SSI 配置为接收模式后执行对外部存储器的读取时，第一次读取的数据可能会在 SSI 数据管脚上发生总线争用。

权变措施

在将 SSI 配置为接收模式（设置 QSSI 控制 (SSICR1) 寄存器的 DIR 位）之后，在第一个有效读取操作之前从存储器执行虚拟读取。例如：

```
SSIConfigSetExpClk(SSI0_BASE,SysCtlClockFreqSet (),
SSI_FRF_MOTO_MODE_0, SSI_MODE_MASTER, 1000000, 8);
SSIAdvModeSet(SSI0_BASE, SSI_ADV_BI_READ); //Receive Mode set
SSIDataPut(SSI0_BASE, &pui32DataRx[ui32Index]); //intentional dummy write
SSIDataGetNonBlocking(SSI-_BASE, ui32Dummy); //dummy read
```

SSI#05（续） ***SSI 的双模式和四模式下的总线争用***

```
    SSIDataPut(SSI0_BASE, &pui32DataRx[ui32Index]); //intentional dummy write
    SSIDataGetNonBlocking(SSI0_BASE, &pui32DataRx[0])); //first intentional
read
    SSIDataPut(SSI0_BASE, &pui32DataRx[ui32Index]); //intentional dummy write
    SSIDataGetNonBlocking(SSI0_BASE, &pui32DataRx[1])); //second intentional
read
```

如果传送通常需要任何虚拟操作，例如上面显示的有意虚拟写入，则虚拟读取应在正常虚拟操作之前发生。

请注意，如果您的应用对 SSIClk 很敏感，则虚拟读取会输出一个时钟周期。在执行虚拟读取的同时将 SSIClk 引脚重新配置为 GPIO 输入，以防止这种情况影响您的时钟敏感型应用。

SSI#06 ***在从机模式中，SSI 接收 FIFO 超时中断的生效时间可能比预期的要早***

说明

在从机模式中，如果 SSI 时钟预分频 (SSICPSR) 寄存器的 CPSDVSR 位字段被设为大于 0x2 的值，则 SSI 接收 FIFO 超时中断可能会早于 32 个系统时钟周期有效。主机模式不受该行为影响。

权变措施

某些情况下，软件可以将 SSI 控制 0 (SSICR0) 寄存器的 SCR 字段结合 CPSDVSR 字段值 0x2 来获得相同的 SSI 时钟频率。例如，如果所需的串行时钟速率为 SysClk/48，则可以在公式 SSInCLK = SysClk / (CPSDVSR * (1 + SCR)) 中使用 CPSDVSR = 0x2 和 SCR = 0x17，而不是 CPSDVSR = 0x18 和 SCR = 0x1 来实现相同的时钟速率。如果某个 SCR 值不能与 CPSDVSR = 0x2 结合使用以获得所需的串行时钟速率，则无法使用接收 FIFO 超时功能。

SSI#07 ***SSI 发送中断状态位不被锁存***

说明

SSI 传输中断状态位工作不正常。

- 适用于在传输 FIFO 半满或更少时带有中断的主机模式。即使使用了 SSIICR 寄存器清除中断条件，每当 TXFIFO 降至 FIFO 阈值的一半以下，导致中断一直有效时，SSIMIS 也会生效。

权变措施

在中断处理程序中，从缓冲区到 SSIDR 的传输完成后将清除 SSIIM.TXIM 以停止任何进一步的中断。当传输需要下一组中断时，设置 SSIIM.TXIM 位。

SSI#08 ***SSI 从器件在双模式和四模式下交换 XDAT0 和 XDAT1***

说明

处于双模式或四模式时，SSI 从器件会在 XDAT0 至 XDAT1 线路和 XDAT1 至 XDAT0 线路上发送数据。

权变措施

当在双模式和四模式下传输来自 SSI 从器件的数据时，CPU 必须在对数据寄存器 SSIDR 进行写入操作交换位来传输正确的位。使用 UDMA CPU 时，必须在 UDMA 的源缓冲区中写入交换的位。

SYSCTL#03 *在时钟成功运行后，MOSC 验证电路没有检测到时钟丢失*

说明

如果 MOSC 时钟源已经上电并正确运行，随后又被移除或为平整线，则 MOSC 验证电路不指示错误情况。

权变措施

使用脱离 PIOSC 运行的看门狗模块 1 在 MOSC 发生故障时复位系统。

SYSCTL#18 *DIVSCLK 输出的时钟频率与 DIV = 0x0 时的预期频率不同*

说明

在分频值和源时钟配置 (DIVSCLK) 寄存器中，如果 DIV 位字段为 0x0（除以 1），则 GPIO 的时钟输出不符合预期：

- 如果 DIV 位字段尚未在代码中调整，则时钟频率将是 SRC 位字段中定义的时钟源除以 32。例如，如果 SRC = 0x1 (PIOSC) 且 DIV = 0x0，则输出到 GPIO 的时钟输出频率将为 500kHz，该频率源自 PIOSC。
- 如果已在代码中调整 DIV 位字段，则时钟频率将是 SRC 位字段中定义的时钟源除以上一个 DIV 位字段值。例如，如果 SRC = 0x1 (PIOSC) 且 DIV 的上一个值 0x1（除以 2），然后写入为 0x0，则输出到 GPIO 的时钟输出频率将为 8MHz，该频率源自 PIOSC。

权变措施

如果源的时钟精度不是一个因素，则可以使用非零 DIV 值和不同的 SRC 值来实现某些频率。例如，要实现 16MHz 时钟，则不要使用 SRC = 0x1 (PIOSC) 和 DIV = 0x0，而要使用 SRC = 0x0（系统时钟）和相应的 DIV 值（对于 80MHz 的系统时钟，DIV = 0x4（除以 5））。

SYSCTL#24 *如果先关闭然后再打开电源域，模块也许没有为访问做好准备*

说明

器件上有五个模块（CAN、以太网 MAC、以太网 PHY、USB 和 CCM）驻留在单独的电源域中。这些模块可关闭其电源域，以节省额外的泄漏电流。当电源域关闭然后开启时，如果在其各自的外设就绪寄存器（如 PRUSB 寄存器）上轮询，这些模块可能会无限期地指示不能访问。

权变措施

无。请勿关闭电源域。

USB#04 *器件发送 SE0 以响应 USB 总线复位*

说明

USB 器件发送单端零 (SE0) 总线状态（USB0DP 和 USB0DM 驱动为低电平）以响应主机的 USB 总线复位。根据 USB 技术规范，在 USB 总线复位时，器件不应驱动这些引脚。这不影响 USB 认证。

权变措施

无。

WDT#08 *当使用看门狗定时器 1 时，读取 WDTVALUE 寄存器可能会返回错误值*

说明

使用看门狗定时器 1 时，可能会从看门狗定时器 1 基址的看门狗值 (WDTVALUE) 寄存器中读取不正确的值。

WDT#08（续）	*当使用看门狗定时器 1 时，读取 WDTVALUE 寄存器可能会返回错误值*
权变措施	无。

1.6 附录 1：

要解决错误 HIB#16 在清除期间，应用代码可能丢失新的篡改事件 ，必须将 HibernateTamperEventsClear() API 替换为以下 API：

```
HibernateTamperEventsClearNoLock();
HibernateTamperUnLock();
HibernateTamperLock();
```

API 定义如下：

```
//*****************************************************************************
//
//! Clears the tamper feature events without Unlock and Lock.
//!
//! This function is used to clear all tamper events without unlock/locking
//! the tamper control registers, so API HibernateTamperUnLock() should be
//! called before this function, and API HibernateTamperLock() should be
//! called after to ensure that tamper control registers are locked.
//!
//! This function doesn't block until the write is complete.
//! Therefore, care must be taken to ensure the next immediate write will
//! occure only after the write complete bit is set.
//!
//! This function is used to implement a software workaround in NMI interrupt
//! handler to fix an issue when a new tamper event could be missed during
//! the clear of current tamper event.
//!
//! \note The hibernate tamper feature is not available on all
//! devices.  Please consult the data sheet for the device that you
//! are using to determine if this feature is available.
//!
//! \return None.
//
//*****************************************************************************
void
HibernateTamperEventsClearNoLock(void)
{
    //
    // Wait for write completion.
    //
    _HibernateWriteComplete();
    //
    //
    // Set the tamper event clear bit.
    //
    HWREG(HIB_TPCTL) |= HIB_TPCTL_TPCLR;
}
//*****************************************************************************
//
//! Unlock temper registers.
//!
//! This function is used to unlock the temper control registers.  This
//! function should be only used before calling API
//! HibernateTamperEventsClearNoLock().
//!
//! \note The hibernate tamper feature is not available on all
//! devices.  Please consult the data sheet for the device that you
//! are using to determine if this feature is available.
//!
//! \return None.
//
//*****************************************************************************
void
HibernateTamperUnLock(void)
{
    //
    // Unlock the tamper registers.
    //
    HWREG(HIB_LOCK) = HIB_LOCK_HIBLOCK_KEY;
    _HibernateWriteComplete();
}
//*****************************************************************************
//
//! Lock temper registers.
```

```
//!
//! This function is used to lock the temper control registers.  This
//! function should be used after calling API
//! HibernateTamperEventsClearNoLock().
//!
//! \note The hibernate tamper feature is not available on all
//! devices.  Please consult the data sheet for the device that you
//! are using to determine if this feature is available.
//!
//! \return None.
//
//*****************************************************************************
void
HibernateTamperLock(void)
{
    //
    // Wait for write completion.
    //
    _HibernateWriteComplete();
    //
    // Lock the tamper registers.
    //
    HWREG(HIB_LOCK) = 0;
    _HibernateWriteComplete();
}
```

必须将软件权变措施添加到 NMI 处理程序中。代码主要在篡改清除同步期间轮询篡改日志条目。

具有此权变措施的 NMI 处理程序示例如下所示。

```
static uint32_t g_ui32RTCLog[4];
static uint32_t g_ui32EventLog[4];
//*****************************************************************************
//
// Handles an NMI interrupt generated by a Tamper event.
//
//*****************************************************************************
void
NMITamperEventHandler(void)
{
    uint32_t ui32NMIStatus, ui32TamperStatus;
    uint32_t pui32Buf[3];
    uint8_t ui8Idx, ui8StartIdx;
    bool     bDetectedEventsDuringClear;
    //
    // Get the cause of the NMI event.
    //
    ui32NMIStatus = SysCtlNMIStatus();
    //
    // We should have got the cause of the NMI event from the above function.
    // But in Snowflake RA0 the NMIC register is not set correctly when an
    // event occurs.  So as a work around check if the NMI event is caused by a
    // tamper event and append this to the return value from SysCtlNMIStatus().
    // This way only this section can be removed once the bug is fixed in next
    // silicon rev.
    //
    ui32TamperStatus = HibernateTamperStatusGet();
    if(ui32TamperStatus & (HIBERNATE_TAMPER_STATUS_EVENT |
                           HIBERNATE_TAMPER_STATUS_EXT_OSC_FAILED))
    {
        ui32NMIStatus |= SYSCTL_NMI_TAMPER;
    }
    //
    // Check if SysCtlNMIStatus() returned a valid value.
    //
    if(ui32NMIStatus)
    {
        //
        // Check if the NMI Interrupt is due to a Tamper event.
        //
        if(ui32NMIStatus & SYSCTL_NMI_TAMPER)
        {
            //
            // If the previous NMI event has not been processed by main
            // thread, we need to OR the new event along with the old ones.
            //
            if(g_ui32NMIEvent == 0)
```

```
        {
            //
            // Reset variables that used for tamper event.
            //
            g_ui32TamperEventFlag = 0;
            g_ui32TamperRTCLog = 0;
            //
            // Clean the log data for debugging purpose.
            //
            memset(g_ui32RTCLog, 0, (sizeof(g_ui32RTCLog))<<2);
            memset(g_ui32EventLog, 0, (sizeof(g_ui32EventLog))<<2);
        }
        //
        // Log the tamper event data before clearing tamper events.
        //
        for(ui8Idx = 0; ui8Idx< 4; ui8Idx++)
        {
            if(HibernateTamperEventsGet(ui8Idx,
                                        &g_ui32RTCLog[ui8Idx],
                                        &g_ui32EventLog[ui8Idx]))
            {
                //
                // Event in this log entry, store it.
                //
                g_ui32TamperEventFlag |= g_ui32EventLog[ui8Idx];
                g_ui32TamperRTCLog = g_ui32RTCLog[ui8Idx];
            }
            else
            {
                //
                // No event in this log entry. Done checking the logs.
                //
                break;
            }
        }
        //
        // Process external oscillator failed event.
        //
        if(ui32TamperStatus & HIBERNATE_TAMPER_STATUS_EXT_OSC_FAILED)
        {
            g_ui32TamperXOSCFailEvent++;
            g_ui32TamperEventFlag |= HIBERNATE_TAMPER_EVENT_EXT_OSC;
            g_ui32TamperRTCLog = HWREG(HIB_TPLOG0);
        }
```

备注

权变措施代码块由此开始。

```
        // The following block of code is to workaround hardware defect
        // which results in missing new tamper events during tamper clear
        // synchronization.
        //
        // There is a window after the application code writes the tamper
        // clear where a new tamper event can be missed if the application
        // requires more than one tamper event pins detection.
        //
        // The tamper Clear is synchronized to the hibernate 32kHz clock
        // domain.  The clear takes 3 rising edges of the 32KHz clock.
        // During this window, new tamper events could be missed.
        // A software workaround is to poll the tamper log during the
        // tamper event clear synchronization.
        //
        //
        // Clear the flag for the case there are events triggered
        // during clear execution.
        //
        bDetectedEventsDuringClear = false;
        //
        // Unlock the Tamper Control register. This is required before
        // calling HibernateTamperEventsClearNoLock().
        //
        HibernateTamperUnLock();
        do
```

```
        {
            //
            // We will start to poll the log registers at index 1 for
            // any new events.
            //
            ui8StartIdx = 1;
            //
            // Clear the Tamper event.
            // Note this API doesn't wait for synchronization, which
            // allows us to check the tamper log during
            // synchronization.
            //
            HibernateTamperEventsClearNoLock();
            //
            // Check new tamper event during tamper event clear
            // synchronization.
            // This will take about 92us(three clock cycles) at most.
            //
            while(HibernateTamperStatusGet() & HIBERNATE_TAMPER_STATUS_EVENT)
            {
                //
                // Clear execution isn't done yet , poll for new events.
                // If there were any new event, it will be logged in log 1
                // registers and so on.
                //
                for(ui8Idx = ui8StartIdx; ui8Idx< 4; ui8Idx++)
                {
                    if(HibernateTamperEventsGet(ui8Idx,
                                                &g_ui32RTCLog[ui8Idx],
                                                &g_ui32EventLog[ui8Idx]))
                    {
                        //
                        // detected new event, store it.
                        //
                        g_ui32TamperEventFlag |= g_ui32EventLog[ui8Idx];
                        //
                        // check for more event.
                        //
                        continue;
                    }
                    else
                    {
                        //
                        // no new event in this log, update the log index
                        // to be checked next, and break out of loop.
                        //
                        ui8StartIdx = ui8Idx;
                        break;
                    }
                }
                //
                // all last three logs have info. Check if all 4 logs
                // have the same info. This is to detect the case that
                // events happen during clear execution.
                //
                if(ui8Idx == 4)
                {
                    //
                    // If events happens during clear
                    // execution, all four log registers will be
                    // logged with the same event, to detect this
                    // condition, we will compare with all four log data.
                    //
                    if(HibernateTamperEventsGet(0, &g_ui32RTCLog[0], &g_ui32EventLog[0]))
                    {
                        if((g_ui32RTCLog[0] == g_ui32RTCLog[1])     &&
                           (g_ui32EventLog[0] == g_ui32EventLog[1]) &&
                          (g_ui32RTCLog[0] == g_ui32RTCLog[2])     &&
                           (g_ui32EventLog[0] == g_ui32EventLog[2]) &&
                           (g_ui32RTCLog[0] == g_ui32RTCLog[3])     &&
                           (g_ui32EventLog[0] == g_ui32EventLog[3]))
                        {
                            //
                            // Detected events during clear execution.
                            // Event logging takes priority, the clear
                            // will not be done in this case. We will need
                            // to go back to the beginning of the loop and
                            // clear the events.
```

```
                                //
                                if(bDetectedEventsDuringClear)
                                {
                                    //
                                    // This condition has already detected,
                                    // we have cleared the event,
                                    // clear the flag.
                                    //
                                    bDetectedEventsDuringClear = false;
                                }
                                else
                                {
                                    // This is the first time it has been
                                    // detected, set the flag.
                                    //
                                    bDetectedEventsDuringClear = true;
                                }
                                //
                                // Break out of while loop so that we can
                                // clear the events, and start the
                                // workaround all over again.
                                //
                                break;
                            }
                        }
                        else
                        {
                            //
                            // Log 0 didn't detect any events. So this is not
                            // the case of missing events during clear
                            // execution.
                            // Update the log index at which we will poll next.
                            // It should be the last log entry that OR all the
                            // new events.
                            //
                            ui8StartIdx = 3;
                        }
                    }
                }
            }
            while(bDetectedEventsDuringClear);
            //
            // Lock the Tamper Control register.
            //
            HibernateTamperLock();
```

备注

权变措施代码块由此结束。

```
            //
            // Save the tamper event and RTC log info in the Hibernate Memory
            //
            HibernateDataGet(pui32Buf, 3);
            pui32Buf[1] = g_ui32TamperEventFlag;
            pui32Buf[2] = g_ui32TamperRTCLog;
            HibernateDataSet(pui32Buf, 3);
            //
            // Signal the main loop that an NMI event occurred.
            //
            g_ui32NMIEvent++;
        }
        //
        // Clear NMI events
        //
        SysCtlNMIClear(ui32NMIStatus);
    }
}
```

1.7 附录 2：

要解决错误 EPI#01 使用 EPI 模块中的代码地址空间时，数据的读取可能损坏，应将以下代码添加到 epi.h 文件中。

```
#ifdef rvmdk
//*****************************************************************************
//
// Keil case.
//
//*****************************************************************************
inline void
EPIWorkaroundWordWrite(uint32_t *pui32Addr, uint32_t ui32Value)
{
  uint32_t ui32Scratch;
    __asm
    {
        //
        // Add a NOP to ensure we don't have a flash read immediately before
        // the EPI read.
        //
        NOP
        //
        // Perform the write we're actually interested in.
        //
        STR ui32Value, [pui32Addr]
        //
        // Read from SRAM to ensure that we don't have an EPI write followed by
        // a flash read.
        //
        LDR ui32Scratch, [__current_sp()]
    }
}
inline uint32_t
EPIWorkaroundWordRead(uint32_t *pui32Addr)
{
  uint32_t ui32Value, ui32Scratch;
    __asm
    {
        //
        // Add a NOP to ensure we don't have a flash read immediately before
        // the EPI read.
        //
        NOP
        //
        // Perform the read we're actually interested in.
        //
        LDR ui32Value, [pui32Addr]
        //
        // Read from SRAM to ensure that we don't have an EPI read followed by
        // a flash read.
        //
        LDR ui32Scratch, [__current_sp()]
    }
    return(ui32Value);
}
inline void
EPIWorkaroundHWordWrite(uint16_t *pui16Addr, uint16_t ui16Value)
{
    uint32_t ui32Scratch;
    __asm
    {
        //
        // Add a NOP to ensure we don't have a flash read immediately before
        // the EPI read.
        //
        NOP
        //
        // Perform the write we're actually interested in.
        //
        STRH ui16Value, [pui16Addr]
        //
        // Read from SRAM to ensure that we don't have an EPI write followed by
        // a flash read.
        //
        LDR ui32Scratch, [__current_sp()]
    }
```

```
}
inline uint16_t
EPIWorkaroundHWordRead(uint16_t *pui16Addr)
{
    uint32_t ui32Scratch;
    uint16_t ui16Value;
    __asm
    {
        //
        // Add a NOP to ensure we don't have a flash read immediately before
        // the EPI read.
        //
        NOP
        //
        // Perform the read we're actually interested in.
        //
        LDRH ui16Value, [pui16Addr]
        //
        // Read from SRAM to ensure that we don't have an EPI read followed by
        // a flash read.
        //
        LDR ui32Scratch, [__current_sp()]
    }
    return(ui16Value);
}
inline void
EPIWorkaroundByteWrite(uint8_t *pui8Addr, uint8_t ui8Value)
{
  uint32_t ui32Scratch;
    __asm
    {
        //
        // Add a NOP to ensure we don't have a flash read immediately before
        // the EPI read.
        //
        NOP
        //
        // Perform the write we're actually interested in.
        //
        STRB ui8Value, [pui8Addr]
        //
        // Read from SRAM to ensure that we don't have an EPI write followed by
        // a flash read.
        //
        LDR ui32Scratch, [__current_sp()]
    }
}
inline uint8_t
EPIWorkaroundByteRead(uint8_t *pui8Addr)
{
    uint32_t ui32Scratch;
    uint8_t ui8Value;
    __asm
    {
        //
        // Add a NOP to ensure we don't have a flash read immediately before
        // the EPI read.
        //
        NOP
        //
        // Perform the read we're actually interested in.
        //
        LDRB ui8Value, [pui8Addr]
        //
        // Read from SRAM to ensure that we don't have an EPI read followed by
        // a flash read.
        //
        LDR ui32Scratch, [__current_sp()]
    }
    return(ui8Value);
}
#else
#ifdef ccs
//*****************************************************************************
//
// Code Composer Studio versions of these functions can be found in separate
// source file epi_workaround_ccs.s.
//
```

```
//*****************************************************************************
extern void EPIWorkaroundWordWrite(uint32_t *pui32Addr, uint32_t ui32Value);
extern uint32_t EPIWorkaroundWordRead(uint32_t *pui32Addr);
extern void EPIWorkaroundHWordWrite(uint16_t *pui16Addr, uint16_t ui16Value);
extern uint16_t EPIWorkaroundHWordRead(uint16_t *pui16Addr);
extern void EPIWorkaroundByteWrite(uint8_t *pui8Addr, uint8_t ui8Value);
extern uint8_t EPIWorkaroundByteRead(uint8_t *pui8Addr);
#else
//*****************************************************************************
//
// GCC and IAR case.
//
//*****************************************************************************
inline void
EPIWorkaroundWordWrite(uint32_t *pui32Addr, uint32_t ui32Value)
{
    volatile register uint32_t ui32Scratch;
    __asm volatile (
        //
        // Add a NOP to ensure we don't have a flash read immediately before
        // the EPI read.
        //
        "    NOP\n"
        "    STR %[value],[%[addr]]\n"
        "    LDR %[scratch],[sp]\n"
         :  [scratch] "=r" (ui32Scratch)
         :  [addr] "r" (pui32Addr), [value] "r" (ui32Value)
     );
    //
    // Keep the compiler from generating a warning.
    //
    ui32Scratch = ui32Scratch;
}
inline uint32_t
EPIWorkaroundWordRead(uint32_t *pui32Addr)
{
    volatile register uint32_t ui32Data, ui32Scratch;
    //
    // ui32Scratch is not used other than to add a padding read following the
    // "real" read.
    //
    __asm volatile(
        //
        // Add a NOP to ensure we don't have a flash read immediately before
        // the EPI read.
        //
        "    NOP\n"
        "    LDR %[ret],[%[addr]]\n"
        "    LDR %[scratch],[sp]\n"
         : [ret] "=r" (ui32Data),
           [scratch] "=r" (ui32Scratch)
         : [addr] "r" (pui32Addr)
    );
    //
    // Keep the compiler from generating a warning.
    //
    ui32Scratch = ui32Scratch;
    return(ui32Data);
}
inline void
EPIWorkaroundHWordWrite(uint16_t *pui16Addr, uint16_t ui16Value)
{
    volatile register uint32_t ui32Scratch;
    __asm volatile (
        //
        // Add a NOP to ensure we don't have a flash read immediately before
        // the EPI read.
        //
        "    NOP\n"
        "    STRH %[value],[%[addr]]\n"
        "    LDR %[scratch],[sp]\n"
         :  [scratch] "=r" (ui32Scratch)
         :  [addr] "r" (pui16Addr), [value] "r" (ui16Value)
     );
    //
    // Keep the compiler from generating a warning.
    //
    ui32Scratch = ui32Scratch;
```

```
}
inline uint16_t
EPIWorkaroundHWordRead(uint16_t *pui16Addr)
{
    register uint16_t ui16Data;
    register uint32_t ui32Scratch;
    //
    // ui32Scratch is not used other than to add a padding read following the
    // "real" read.
    //
    __asm volatile(
        //
        // Add a NOP to ensure we don't have a flash read immediately before
        // the EPI read.
        //
        "    NOP\n"
        "    LDRH %[ret],[%[addr]]\n"
        "    LDR %[scratch],[sp]\n"
         : [ret] "=r" (ui16Data),
           [scratch] "=r" (ui32Scratch)
         : [addr] "r" (pui16Addr)
    );
    //
    // Keep the compiler from generating a warning.
    //
    ui32Scratch = ui32Scratch;
    return(ui16Data);
}
inline void
EPIWorkaroundByteWrite(uint8_t *pui8Addr, uint8_t ui8Value)
{
    volatile register uint32_t ui32Scratch;
    __asm volatile (
        //
        // Add a NOP to ensure we don't have a flash read immediately before
        // the EPI read.
        //
        "    NOP\n"
        "    STRB %[value],[%[addr]]\n"
        "    LDR %[scratch],[sp]\n"
         :  [scratch] "=r" (ui32Scratch)
         :  [addr] "r" (pui8Addr), [value] "r" (ui8Value)
     );
    //
    // Keep the compiler from generating a warning.
    //
    ui32Scratch = ui32Scratch;
}
inline uint8_t
EPIWorkaroundByteRead(uint8_t *pui8Addr)
{
    register uint8_t ui8Data;
    register uint32_t ui32Scratch;
    //
    // ui32Scratch is not used other than to add a padding read following the
    // "real" read.
    //
    __asm volatile(
        //
        // Add a NOP to ensure we don't have a flash read immediately before
        // the EPI read.
        //
        "    NOP\n"
        "    LDRB %[ret],[%[addr]]\n"
        "    LDR %[scratch],[sp]\n"
         : [ret] "=r" (ui8Data),
           [scratch] "=r" (ui32Scratch)
         : [addr] "r" (pui8Addr)
    );
    //
    // Keep the compiler from generating a warning.
    //
    ui32Scratch = ui32Scratch;
    return(ui8Data);
}
#endif
```

此外，如果使用 CCS，下列代码应该保存为名为 epi_workaround_ccs.s *driverlib* 目录的文件并包含在项目中：

```
;*******************************************************************************
;
; epi_workaround_ccs.s - EPI memory access functions.
;
; Copyright (c) 2013 Texas Instruments Incorporated.  All rights reserved.
; TI Information - Selective Disclosure
;
;*******************************************************************************
;*******************************************************************************
;
; void EPIWorkaroundWordWrite(uint32_t *pui32Addr, uint32_t ui32Value)
;
;*******************************************************************************
    .sect ".text:EPIWorkaroundWordWrite"
    .global EPIWorkaroundWordWrite
EPIWorkaroundWordWrite:
    ;
    ; Include a no-op to ensure that we don't have a flash data access
    ; immediately before the EPI access.
    ;
    nop
    ;
    ; Store the word in EPI memory.
    ;
    str r1, [r0]
    ;
    ; Make a dummy read from the stack to ensure that we don't have a flash
    ; data access immediately after the EPI access.
    ;
    ldr r1, [sp]
    ;
    ; Return to the caller.
    ;
    bx lr
    .align 4
;*******************************************************************************
;
; uint32_t EPIWorkaroundWordRead(uint32_t *pui32Addr)
;
;*******************************************************************************
    .sect ".text:EPIWorkaroundWordRead"
    .global EPIWorkaroundWordRead
EPIWorkaroundWordRead:
    ;
    ; Include a no-op to ensure that we don't have a flash data access
    ; immediately before the EPI access.
    ;
    nop
    ;
    ; Read the word from EPI memory.
    ;
    ldr r0, [r0]
    ;
    ; Make a dummy read from the stack to ensure that we don't have a flash
    ; data access immediately after the EPI access.
    ;
    ldr r1, [r13]
    ;
    ; Return to the caller.
    ;
    bx lr
    .align 4
;*******************************************************************************
;
; void EPIWorkaroundHWordWrite(uint16_t *pui16Addr, uint16_t ui16Value)
;
;*******************************************************************************
    .sect ".text:EPIWorkaroundHWordWrite"
    .global EPIWorkaroundHWordWrite
EPIWorkaroundHWordWrite:
    ;
    ; Include a no-op to ensure that we don't have a flash data access
    ; immediately before the EPI access.
    ;
    nop
```

```
    ;
    ; Store the word in EPI memory.
    ;
    strh r1, [r0]
    ;
    ; Make a dummy read from the stack to ensure that we don't have a flash
    ; data access immediately after the EPI access.
    ;
    ldr r1, [sp]
    ;
    ; Return to the caller.
    ;
    bx lr
    .align 4
;*******************************************************************************
;
; uint16_t EPIWorkaroundHWordRead(uint16_t *pui16Addr)
;
;*******************************************************************************
    .sect ".text:EPIWorkaroundHWordRead"
    .global EPIWorkaroundHWordRead
EPIWorkaroundHWordRead:
    ;
    ; Include a no-op to ensure that we don't have a flash data access
    ; immediately before the EPI access.
    ;
    nop
    ;
    ; Read the half word from EPI memory.
    ;
    ldrh r0, [r0]
    ;
    ; Make a dummy read from the stack to ensure that we don't have a flash
    ; data access immediately after the EPI access.
    ;
    ldr r1, [r13]
    ;
    ; Return to the caller.
    ;
    bx lr
    .align 4
;*******************************************************************************
;
; void EPIWorkaroundByteWrite(uint8_t *pui8Addr, uint8_t ui8Value)
;
;*******************************************************************************
    .sect ".text:EPIWorkaroundByteWrite"
    .global EPIWorkaroundByteWrite
EPIWorkaroundByteWrite:
    ;
    ; Include a no-op to ensure that we don't have a flash data access
    ; immediately before the EPI access.
    ;
    nop
    ;
    ; Store the byte in EPI memory.
    ;
    strb r1, [r0]
    ;
    ; Make a dummy read from the stack to ensure that we don't have a flash
    ; data access immediately after the EPI access.
    ;
    ldr r1, [sp]
    ;
    ; Return to the caller.
    ;
    bx lr
    .align 4
;*******************************************************************************
;
; uint8_t EPIWorkaroundByteRead(uint8_t *pui8Addr)
;
;*******************************************************************************
    .sect ".text:EPIWorkaroundByteRead"
    .global EPIWorkaroundByteRead
EPIWorkaroundByteRead:
    ;
    ; Include a no-op to ensure that we don't have a flash data access
```

```
    ; immediately before the EPI access.
    ;
    nop
    ;
    ; Read the byte from EPI memory.
    ;
    ldrb r0, [r0]
    ;
    ; Make a dummy read from the stack to ensure that we don't have a flash
    ; data access immediately after the EPI access.
    ;
    ldr r1, [r13]
    ;
    ; Return to the caller.
    ;
    bx lr
    .align 4
.end
```

2 商标

SimpleLink™ and MSP432™ are trademarks of Texas Instruments.
Arm® and Cortex® are registered trademarks of Arm Limited.
所有商标均为其各自所有者的财产。

3 修订历史记录

Changes from OCTOBER 1, 2017 to JUNE 30, 2025 (from Revision * (October 2017) to Revision A (June 2025)) **Page**

- 添加了 SYSCTL#24 公告 11

重要通知和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、某特定用途方面的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他功能安全、信息安全、监管或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的相关应用。 严禁以其他方式对这些资源进行复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款或 ti.com 上其他适用条款/TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

邮寄地址：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265