Texas Instruments

应用报告

ZHCAA02 – December 2015

MSP 代码保护 特性

Katie Pier *MSP Applications*

摘要

MSP 微控制器 (MCU) 具备多种 特性 ，可帮助控制器件中的代码可访问性，并增加不同层的代码访问管理和保护策略。这些 特性 包括：锁定或用密码保护 JTAG/SBW 访问、利用不同于程序其余部分的权限借助 IP 封装 (IPE) 隔离敏感代码，以及用于现场固件更新的引导加载程序 (BSL) 访问 特性 。本应用报告将详细介绍各个 MSP 器件系列中的一些可用 特性 ，以及在为器件增额外加保护层时可以考虑的一些重要事项。

如需了解相关源代码和更多信息，请登录 http://www.ti.com/cn/lit/zip/slaa685。

内容

1 简介 2
2 不同 MSP 系列的 JTAG 锁定 2
3 IP 封装 (IPE) 9
4 引导加载程序 (BSL) 安全性 特性 29
5 参考文献 32

附图目录

1 目标配置 6
2 首次加载时的 JTAG 密码 8
3 CCS 中的 IPE 工具 14
4 CCS 中的调试配置 16
5 CCS 中的 IPE 加载配置 17
6 CCS 中的 IPE 测试配置 18
7 CCS 中的 IPE 内存视图 - 无保护 19
8 CCS 中的 IPE 内存视图 - 受保护 19
9 IAR 中的 IPE 工具 23
10 在 IAR 中生成 .map 文件 24
11 用于 IPE_FR59xx 的选项 26
12 在 IAR 中生成 .txt 二进制文件 26
13 IAR 中的 IPE 调试配置 28
14 IAR 中的 IPE 内存视图 - 受保护 28

附表目录

1 不同 MSP 系列的 JTAG 锁定 特性 2
2 F5xx/F6xx 上的 JTAG 锁定 3
3 FR5xx/FR6xx 上的 JTAG 锁定 3
4 FR2xx/FR4xx 上的 JTAG 锁定 4
5 i2xx 上的 JTAG 锁定 4
6 FR5xx/FR6xx 上的 JTAG 密码锁定 5
7 IPE 初始化结构 16
8 IPE 初始化结构 27

SLAA685 — http://www-s.ti.com/sc/techlit/SLAA685
版权 © 2015, Texas Instruments Incorporated

9 MSP430FR5xx/FR6xx 器件上的 BSL 签名功能 31
10 MSP430F1xx/F2xx/F4xx 器件上的 BSL 签名功能 31

商标

MSP430, MSP432, Code Composer Studio are trademarks of Texas Instruments.
IAR Embedded Workbench is a trademark of IAR Systems.

1 简介

随着越来越多的产品采用微控制器 (MCU) 或其他嵌入式器件，嵌入式安全变得越来越重要。嵌入式系统 应用 通常需要解决的一个问题是防止读出器件信息，以保护专有代码的知识产权 (IP)。但是，没有恰当的解决方案能够防止未经授权而访问代码或对代码进行逆向工程。设计人员可以采取预防措施，防止外界使用常见的技术或设备轻松访问其代码。为增加安全保护层、防止他人试图读出器件信息，将分层方法与其他方法相结合来控制代码访问权限是一种不错的方法。本应用报告 将 讨论 MSP430™和 MSP432™微控制器系列中的部分特性，这些特性有助于提供以下保护层：锁定 JTAG、（在支持 IP 封装功能的器件上）使用 IP 封装以及引导加载程序 (BSL) 安全选项。综合使用这些方法有助于增强器件的分层保护，还会使他人难以读出器件信息。

注: 本应用报告主要介绍了 MSP430 微控制器的 特性 。有关保护 MSP432 微控制器或在 MSP432 微控制器上使用引导加载程序的信息，请参阅应用报告*Configuring Security and Bootloader (BSL) on MSP432P4xx* (SLAA659)。有关 MSP432 器件的 IP 保护安全区特性，请参阅应用报告 *Software IP Protection on MSP432P4xx Microcontrollers* (SLAA660)。

2 不同 MSP 系列的 JTAG 锁定

表 1 列出了每个 MSP 系列中可用的锁定 特性 。

表 1. 不同 MSP 系列的 JTAG 锁定 特性

器件系列	物理保险丝	引导覆盖	电子保险丝	JTAG 签名	可逆	利用密码访问 JTAG
MSP430F1xx/F2xx/F4xx	✓					
MSP430F5xx/F6xx			✓	0x17FC-0x17FF	✓ 使用引导加载程序	
MSP430FR5xx/FR6xx			✓	0xFF80-0xFF83	✓ 使用引导加载程序	✓
MSP430FR2xx/FR4xx			✓	0xFF80-0xFF83	✓ 使用引导加载程序	
MSP430i2xx			✓	0xFFDC-0xFFDF	出厂时未提供引导加载程序	
MSP432Pxx		✓			✓ 可以将引导覆盖功能恢复出厂设置	

2.1 *物理 JTAG 保险丝 (F1xx/F2xx/F4xx)*

可通过物理 JTAG 安全保险丝保护 MSP430F1xx、F2xx 和 F4xx 系列器件的 JTAG。进行 JTAG 或 SBW 编程之后，保险丝将由编程工具熔断。此工具会对 TEST 引脚（位于带有 TEST 引脚的器件）或 TDI 引脚（位于带有 TEST 引脚的器件）施加保险丝熔断电压 (6.5V ±0.5V) [请参阅*MSP430 Programming Via the JTAG Interface* (SLAU320)，了解更多详细信息]。保险丝熔断是一个不可逆的过程，会彻底禁用 JTAG 端口，而且无法再访问 JTAG 或 SBW。当物理保险丝熔断后，将只能通过设置密码保护的引导加载程序（如果支持并启用了此功能）来访问器件。

SLAA685 — http://www-s.ti.com/sc/techlit/SLAA685
版权 © 2015, Texas Instruments Incorporated

2.2 电子保险丝或无密码锁定

大多数 MSP430 器件都有一个电子保险丝，此保险丝有时被称为 e-Fuse 或"JTAG 无密码锁定"。通过在内存中将一个值设置为 2 个字（JTAG 签名），可以锁定 JTAG/SBW 接口。对 JTAG 锁定签名进行编程并复位器件之后，将无法再通过 JTAG 或 SBW 来访问器件。在这种情况下，只能通过设置密码保护的引导加载程序来访问器件。可以利用 BSL 来重新访问器件并清除 JTAG 签名，但这样需要输入正确的 BSL 密码或者批量擦除器件（取决于器件系列）。MSP430 不同子系列的电子保险丝实施方式略有不同，以下部分将对此进行说明。

2.2.1 F5xx/F6xx 电子保险丝的实施方式

在 MSP430 F5xx/F6xx 器件上，JTAG 签名位于引导加载程序的内存区域中（F5xx/F6xx 器件的闪存受保护区域中包含一个基于闪存的 BSL），地址为 17FCh-17FFh。如果将除 00000000h 或 FFFFFFFFh 以外的任何内容编程到这些地址，将会锁定 JTAG/SBW 接口。要对这些地址进行编程，必须先清除 SYSBSLC 寄存器中的 SYSBSLPE 位，以解锁 BSL 闪存的受保护区域。对签名编程之后，应重新启用 BSL 保护。

要清除 JTAG/SBW 锁定保护，可以使用 BSL 将 JTAG 签名清除为 00000000h。BSL 由中断矢量表的最后 32 个字节 FFE0h-FFFFh 提供密码保护（请参见4 节）。由于 JTAG 签名位于受保护的 BSL 区域中，因此 BSL 必须先清除 SYSBSLC 寄存器中的 SYSBSLPE 位（向地址 0182h 写入 0003h），才能向 JTAG 签名写入 00000000h。

表 **2. F5xx/F6xx** 上的 **JTAG** 锁定

名称	地址	值	器件安全
JTAG/SBW 签名	17FCh-17FFh[(1)]	FFFF_FFFFh	JTAG/SBW 未锁定。
		0000_0000h	
		任何其他值	JTAG/SBW 锁定。

(1) 这些地址位于受保护的 BSL 闪存区域，此区域必须通过写入 0003h 将 SYSBSLPE 清除为地址 0182h（SYSBSLC 寄存器地址）来解锁，这样才能更改签名。

2.2.2 FR5xx/FR6xx 电子保险丝的实施方式（无密码锁定）

在 MSP430FR5xx/FR6xx 器件上，JTAG 签名位于 FRAM 的主区域中，地址为 FF80h-FF83h。通过向 JTAG 签名写入 55555555h，可以对 JTAG/SBW 接口执行无密码锁定。

要清除 JTAG/SBW 锁定保护，可以使用引导加载程序将 JTAG 签名清除为除 5555h 和 AAAAh 以外的任何值。BSL 受密码保护，中断矢量表的最后 32 个字节 (FFE0h-FFFFh) 用作 BSL 密码（请参见4 节）。通过在这些 FRAM 器件上使用 BSL 批量擦除命令，可轻松解锁 JTAG/SBW 访问，因为 JTAG 签名位于主存储器中（而不是像 F5xx/F6xx 那样位于受保护的 BSL 区域中）。

表 **3. FR5xx/FR6xx** 上的 **JTAG** 锁定

名称	地址	值	器件安全
JTAG/SBW 签名	FF80h-FF83h	5555h_5555h	JTAG/SBW 无密码锁定。
		@FF80h = AAAAh @FF82h = 密码文字长度	JTAG/SBW 有密码锁定。[(1)]
		任何其他值	JTAG/SBW 未锁定。

(1) 请参阅2.3 节，以了解更多详细信息。

2.2.3 FR2xx/FR4xx 电子保险丝的实施方式

在 MSP430FR2xx/FR4xx 器件上，JTAG 签名位于 FRAM 的主区域中，地址为 FF80h-FF83h（与其他 FRxx 器件相同）。通过向 JTAG 签名写入除 00000000h 或 FFFFFFFFh 以外的任何内容，可以对 JTAG/SBW 接口执行无密码锁定（这不同于其他的 FRxx 器件）。

SLAA685 — http://www-s.ti.com/sc/techlit/SLAA685
版权 © 2015, Texas Instruments Incorporated

要清除 JTAG/SBW 锁定保护，可以使用引导加载程序将 JTAG 签名清除为 00000000h 或 FFFFFFFFh。BSL 受密码保护，中断矢量表的最后 32 个字节 (FFE0h-FFFFh) 用作 BSL 密码（请参见4 节）。通过在这些 FRAM 器件上使用 BSL 批量擦除命令，可轻松解锁 JTAG/SBW 访问，因为 JTAG 签名位于主存储器中（而不是像 F5xx/F6xx 那样位于某个受保护的 BSL 区域中）。

表 **4. FR2xx/FR4xx** 上的 **JTAG** 锁定

名称	地址	值	器件安全
JTAG/SBW 签名	FF80h-FF83h	FFFF_FFFFh	JTAG/SBW 未锁定。
		0000_0000h	
		任何其他值	JTAG/SBW 锁定。

2.2.4 **MSP430i2xx** 电子保险丝的实施方式 **-** 启动代码 **(SUC)**

在 MSP430i2xx 器件上，JTAG/SBW 器件安全性由启动代码 (SUC) 控制。在执行 BOR 或 POR 复位之后，SUC 必须在前 64 个 MCLK 时钟周期内将器件设置为安全或不安全状态。SUC 会检查地址 0xFFDC-0xFFDF 的签名，以确定应当启用还是禁用 JTAG/SBW 访问。

通过向地址 FFDCh-FFDFh 写入除 00000000h 或 FFFFFFFFh 以外的任何值，可以禁用 JTAG/SBW 接口。在执行 POR 或 BOR 复位之后，SUC 会检查这些地址中的值。如果此值并非 00000000h 或 FFFFFFFFh，则 SUC 会向 SYSJTAGDIS 寄存器写入 A5A5h，以禁用 JTAG/SBW 接口并保护器件。在执行 BOR 或 POR 复位之后，SUC 必须在前 64 个 MCLK 周期内完成此操作。

由于 MSP430i2xx 器件没有引导加载程序，因此无法清除地址 0xFFDC-0xFFDF 以重新启用 JTAG/SBW 访问。如果用户在主存储器中实施了定制的引导加载程序，他们可以使用此程序向地址 FFDCh-FFDFh 写入 00000000h。在执行 POR 或 BOR 复位之后，SUC 会检查这些地址中的值。如果此值为 00000000h 或 FFFFFFFFh，SUC 将不执行任何操作，并继续执行校准例程 - 在 64 个 MCLK 周期之后，会启用 JTAG，器件将自动处于不安全状态。

请参阅 *MSP430i2xx Family User's Guide* (SLAU335) 和器件代码示例，了解更多有关 SUC 的信息。

表 **5. i2xx** 上的 **JTAG** 锁定

名称	地址	值	器件安全
JTAG/SBW 签名	FFDCh-FFDFh	FFFF_FFFFh	JTAG/SBW 未锁定。[1]
		0000_0000h	
		任何其他值	JTAG/SBW 锁定。[1]

[1] 必须实施正确的 SUC 代码以检查这些地址，并在启动后的 64 个 MCLK 周期内执行锁定或解锁。使用 i2xx 代码示例附带的 SUC 代码（该代码已经实施此操作）。

SLAA685 — http://www-s.ti.com/sc/techlit/SLAA685
版权 © 2015, Texas Instruments Incorporated

2.3 JTAG 密码锁定 (FR5xx/FR6xx)

有些 MSP430 FRAM 器件还允许使用 JTAG 密码锁定选项。这不同于电子保险丝/JTAG 无密码锁定，因为使用这种机制时，无需使用引导加载程序，只需通过工具链提供用户定义的密码即可重新访问。

在 MSP430FR5xx/FR6xx 器件上，JTAG 签名位于 FRAM 的主区域中，地址为 FF80h-FF83h。通过向 JTAG 签名 1 (FF80h) 写入 AAAAh，并向 JTAG 签名 2 (FF82h) 写入除 5555h 以外的任何内容，即可使用密码锁定 JTAG/SBW。写入到 JTAG 签名 2 的值用来定义所需密码的文字长度。密码从地址 FF88h 开始，并持续 JTAG 签名 2 所设置的字数。请注意，如果密码足够长，可能会与中断矢量表所使用的地址重叠。这些地址的密码值应是中断矢量表值，只有这样才能正确处理中断和执行代码。密码保护会对下一个 BOR 事件生效。

表 6. FR5xx/FR6xx 上的 JTAG 密码锁定

名称	地址	值	器件安全
JTAG/SBW 签名	FF80h-FF83h	5555h_5555h	JTAG/SBW 无密码锁定。
		@FF80h = AAAAh @FF82h = 密码文字长度	JTAG/SBW 有密码锁定。
		任何其他值	JTAG/SBW 未锁定。
JTAG 密码	FF88h 长度	用户定义 + 矢量表配置	如果使用密码锁定了 JTAG/SBW，则必须由工具链通过 JTAG 邮箱提供这些地址中定义的密码值。

当使用 JTAG 密码锁定功能保护器件时，工具链必须提供正确的密码才能访问器件。提供了正确的密码之后，器件会解锁，直到发生下一个 BOR 事件为止。

2.3.1 在 CCS 中使用 JTAG 密码锁定

相关 zip 文件 (http://www.ti.com/cn/lit/zip/slaa685) 中的代码示例 JTAG_lock_FR5xx_with_password.c 显示了如何在 Code Composer Studio™IDE (CCS) 中为 MSP430FR5969 器件设置 JTAG 密码锁定。但请注意，有些生产编程工具可能还提供了一个选项，以便从 GUI 界面设置密码，而不必将密码硬编码到项目中。

对于 MSP430 TI C/C++ 编译器，使用 #pragma DATA_SECTION 将正确的签名放到链接器文件的 .jtagsignature 部分，并将密码放到 .jtagpassword 部分。使用 #pragma RETAIN 防止编译器优化数据（因为程序不会使用这些数据）。更多有关这些程序的信息，请参阅 *MSP430 TI C/C++ Compiler guide* (SLAU132)。

```
#pragma RETAIN(JTAG_signatures)
#pragma DATA_SECTION(JTAG_signatures, ".jtagsignature")
const uint16_t JTAG_signatures[] = {0xAAAA, 0x0002};

...

#pragma RETAIN(JTAG_password)
#pragma DATA_SECTION(JTAG_password, ".jtagpassword")
const uint8_t JTAG_password[] = {0x12, 0x34, 0x56, 0x78};
```


版权 © 2015, Texas Instruments Incorporated

如下示例显示了使用 2 字密码 12h 34h 56h 78h 锁定的器件的 TI-txt 二进制文件的一部分：

```
@ff80
AA AA 02 00 FF FF FF FF 12 34 56 78 FF FF FF FF
```

当器件由密码锁定并完成 BOR 复位（此时设置将会生效）之后，必须提供密码，才能重新访问器件以进行重新编程。在 CCS 中提供密码，然后在 targetConfigs 文件夹中打开器件的 .ccxml 文件。在“Advanced Setup”下方，单击链接“Target Configuration”。然后单击器件型号下方的“MSP430”，并在“Password: (HEX format)”框中输入要在解锁器件时提供的密码。对于上述示例，密码为 12h 34h 56h 78h。应以文字形式写入密码，先写入最低有效文字，因此对于此示例，输入 0x34127856。要找到为工具链提供密码的正确顺序，一种简单的方法是使用内存浏览器并将视图设置为“16-Bit Hex – TI Style”，然后在首次加载时查看编程到器件中的密码。

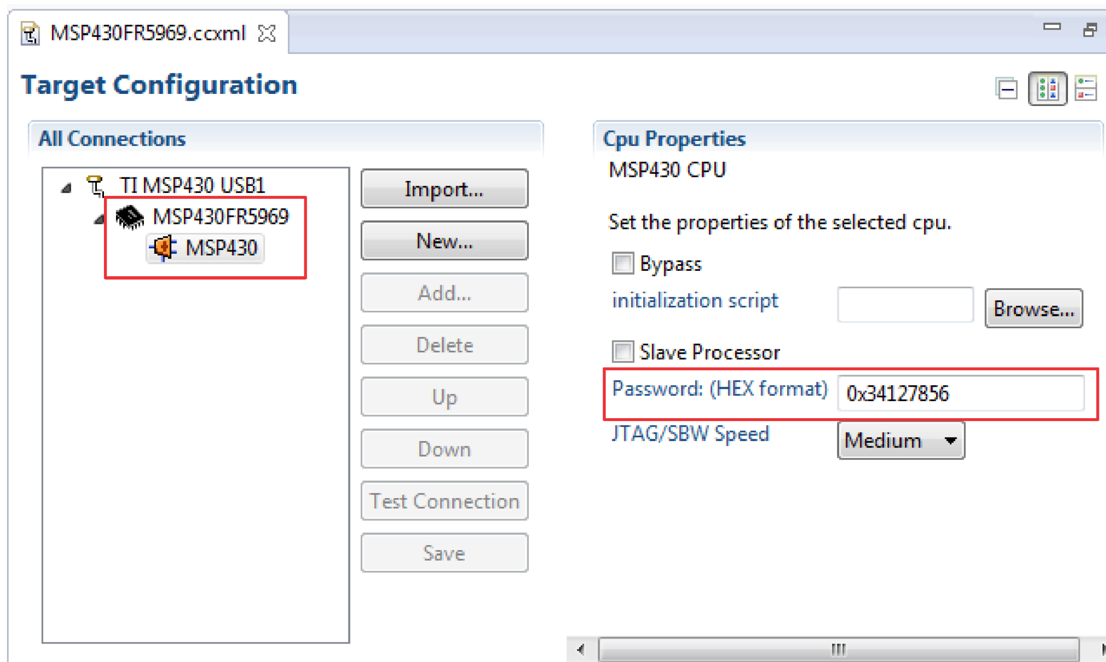

图 **1.** 目标配置

注: 当加载不同的用户项目时，如果在器件此前加载的代码中启用了 JTAG 密码锁定，则首次进入器件时，必须在项目设置中提供正确的密码并擦除器件。擦除器件之后，如果没有在新项目中启用 JTAG 密码锁定，则会解锁器件并能够正常对其进行调试。

SLAA685 — http://www-s.ti.com/sc/techlit/SLAA685
版权 © 2015, Texas Instruments Incorporated

2.3.2 在 IAR 中使用 JTAG 密码锁定

相关 zip 文件中的代码示例 JTAG_lock_FR5xx_with_password.c 显示了如何在 IAR 中的 MSP430FR5969 器件上设置 JTAG 密码锁定。但请注意，有些生产编程工具可能还提供了一个选项，以便从 GUI 界面设置密码，而不必将密码硬编码到项目中。

对于 IAR C/C++ 编译器，使用 #pragma 位置从地址 FF80h 开始放置正确的签名，从地址 FF88h 开始放置密码。使用 __root 关键字防止编译器优化数据（因为程序不会使用这些数据）。更多有关这些 pragma 的信息，请参阅 IAR 中的“Help”菜单下的 IAR C/C++ Compiler User's Guide。

```
#pragma location = 0xFF80
__root const uint16_t JTAG_signatures[] = {0xAAAA, 0x0002};

...

#pragma location = 0xFF88
__root const uint8_t JTAG_password[] = {0x12, 0x34, 0x56, 0x78};
```

如下示例显示了使用 2 字密码 12h 34h 56h 78h 锁定的器件的 TI-txt 二进制文件的一部分：

```
@FF80
AA AA 02 00
@FF88
12 34 56 78
```

要在 IAR 中提供密码，请在Project > Options 菜单中转到 "Debugger" > "FET Debugger"，然后选择" Download" 选项卡。在"JTAG password"框中，输入要在解锁器件时提供的密码。对于上述示例，密码为 12h 34h 56h 78h。应以文字形式写入密码，先写入最低有效文字，因此对于此示例，输入 0x34127856。要找到为工具链提供密码的正确顺序，一种简单的方法是使用“内存”视图并将格式设置为“2x Units”，然后在首次加载时查看编程到器件中的密码。

SLAA685 — http://www-s.ti.com/sc/techlit/SLAA685
版权 © 2015, Texas Instruments Incorporated

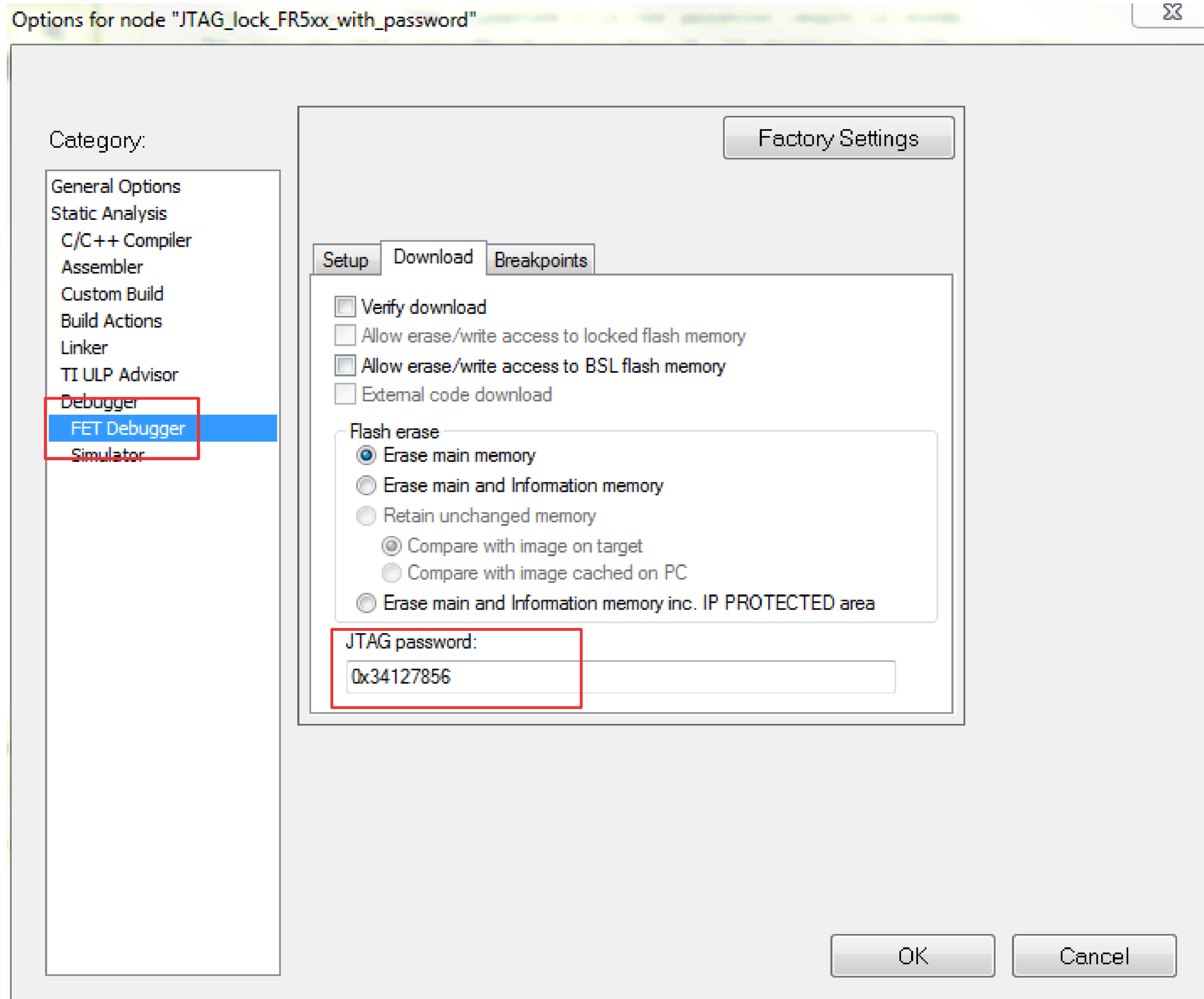

图 **2.** 首次加载时的 **JTAG** 密码

注: 当加载不同的用户项目时，如果在器件此前加载的代码中启用了 JTAG 密码锁定，则首次进入器件时，必须在项目设置中提供正确的密码并擦除器件。擦除器件之后，如果没有在新项目中启用 JTAG 密码锁定，则会解锁器件并能够正常对其进行调试。

版权 © 2015, Texas Instruments Incorporated

3 IP 封装 (IPE)

IP 封装 (IPE) 是诸如 MSP430FR59xx/69xx 系列 MSP430 FRAM 微控制器上提供的一种功能。（注意：对于 MSP432 器件上的 IP 保护安全区特性，请参阅应用报告*Software IP Protection on MSP432P4xx Microcontrollers* (SLAA660)。）

IP 封装允许用户封装 FRAM 存储器的某个区域以防止他人读取。当 IPE 激活时，即使使用 JTAG，也无法从 IP 封装区域以外的任何位置对 IP 封装区域中的任何代码或数据进行读取或写入访问。要去除 IPE 保护，需要执行由工具链提供支持的特殊批量擦除序列。任何形式的代码保护都不是无懈可击的，但此功能会为 JTAG/SBW 或引导加载程序的安全性提供额外一层保护，适用于密钥等敏感数据或者被用户视为知识产权 (IP) 的专有代码。

IP 封装的安全性和存储在其内的代码的安全性是一样的 - 即使代码位于封装区域内，糟糕的代码安全性措施也会让代码变得更容易受到攻击。需要特别注意可以读取/写入地址的代码或者不遵循可靠编码实践的代码。当封装执行结束时，如果需要隐藏代码对所使用的任何硬件模块和外设寄存器或者 RAM 执行的操作，则还必须将这些模块清除为它们的原始状态。

例如，IPE 示例项目包含一个示例，说明了如何清除 IPE 代码部分所使用的硬件模块寄存器：

```
TA0CCTL0 = 0;
TA0CCR0 = 0;
TA0CCR1 = 0;
TA0CCR2 = 0;
TA0CTL = 0;
TA0R = 0;
P4DIR &= ~BIT6;
P4OUT &= ~BIT6;
```

它还包含一个示例，说明了如何清除通用 CPU 寄存器 R4 到 R15：

```
__asm(" mov.w #0, R4");
__asm(" mov.w #0, R5");
__asm(" mov.w #0, R6");
__asm(" mov.w #0, R7");
__asm(" mov.w #0, R8");
__asm(" mov.w #0, R9");
__asm(" mov.w #0, R10");
__asm(" mov.w #0, R11");
__asm(" mov.w #0, R12");
__asm(" mov.w #0, R13");
__asm(" mov.w #0, R14");
__asm(" mov.w #0, R15");
```

注: 如果将参数传回给 IPE 区域以外的代码，则可能需要保留寄存器 R12 到 R15。请参阅 *MSP430 Optimizing C/C++ Compiler User's Guide* 中的“How a Function Makes a Call”和“How a Called Function Responds”部分，了解更多信息。

对于不同的 应用，可能需要采取更多的代码安全性措施；例如清除在函数期间分配的 RAM。另一种可以采取的预防措施是，在 IPE 执行时对 IPE 未使用的函数禁用中断（如果可能），然后在 IPE 结束时重新启用它们。这样可以确保在服务于 IPE 区域以外的任何 ISR 之前清除寄存器或 RAM。这些类型的高级预防措施与应用相关，而且需要隐藏的内容因需求而异，但在所有情况下都应特别注意。

SLAA685 — http://www-s.ti.com/sc/techlit/SLAA685
版权 © 2015, Texas Instruments Incorporated

3.1 使用 CCS 中的 IPE 工具进行 IP 封装

CCS 具有一个内置的 IPE 工具，并在链接器文件中提供了预定义的区段，以便于在项目中启用 IPE。尽管*《MSP430FR58xx、MSP430FR59xx、MSP430FR68xx 和 MSP430FR69xx 系列用户指南》* 详细说明了如何设置 IPE 初始化结构以及如何利用 MPU 寄存器手动启用 IPE，但使用 CCS 中的内置 IPE 工具仍不失为更轻松、更自动化的替代方法。随后的几节将讨论如何使用 CCS 中的这些工具在项目中启用 IPE。另请参阅相关的来源 (http://www.ti.com/cn/lit/zip/slaa685)，以了解 CCS 中使用 IPE 的示例项目。此代码将在 MSP-EXP430FR5969 LaunchPad 开发套件上运行。更多有关如何在 CCS 中使用此工具启用 IPE 的信息，请参阅*Code Composer Studio v6.1 for MSP430 User's Guide* (SLAU157)。

3.1.1 用于 IPE 的 CCS 链接器文件 特性

注: 这些链接器文件 特性 在 CCSv6.1.1 发布时是最新的版本。早期版本的 CCS 链接器文件并不具备所有这些 特性，因此建议在通过这种方法启用 IPE 时使用 CCSv6.1.1 或更高版本。

对于支持 IPE 的器件，CCS 链接器 .cmd 文件包含一些用来放置 IPE 代码和数据的部分。用户代码只需告诉编译器应将哪些变量和函数放到这些预定义的部分中。如下代码片段来自 MSP430FR5969 链接器 cmd 文件：

```
GROUP(IPENCAPSULATED_MEMORY)
{
   .ipestruct     : {}               /* IPE Data structure               */
   .ipe           : {}               /* IPE                              */
   .ipe_const     : {}               /* IPE Protected constants          */
   .ipe:_isr      : {}               /* IPE ISRs                         */
   .ipe_vars      : type = NOINIT{} /* IPE variables                    */
} PALIGN(0x0400), RUN_START(fram_ipe_start) RUN_END(fram_ipe_end)
RUN_END(fram_rx_start)
```

- **.ipestruct** - 此代码段用于包含 IPE 初始化结构，此结构包含有关 IP 封装存储器边界和控制设置的地址位置信息。*《MSP430FR58xx、MSP430FR59xx、MSP430FR68xx 和 MSP430FR69xx 系列用户指南》* 中提供了更多有关 IPE 初始化结构的信息。用户不应将任何数据放在此部分中 - 当使用内置的 CCS IPE 工具时，此工具会生成所需的值并将这些值放在这里。
- **.ipe** - 此代码段用于用户希望放在 IP 封装区域中的任何代码函数。
- **.ipe_const** - 此代码段用于需要封装的任何常量数据。这些数据可以是加密密钥或校准信息、封装的代码版本号或者应当隐藏的封装代码所使用的任何常数。取决于应用。
- **.ipe: _isr** - 此代码段用于需要成为 IP 封装用户代码的一部分的任何中断服务例程 (ISR)。
- **.ipe_vars** - 此代码段用于需要封装的任何数据变量。为了获得最高的安全性，只在 IP 封装代码内使用的任何变量还应放到 IP 封装区域中，而不是放到 RAM 中 - 否则有人可能利用所观察到的 RAM 变化尝试对封装代码的功能进行逆向工程。当然，这取决于应用。

继续转到链接器文件的底部，可以看到有一个部分用于为 IPE 结构计算 IPE 初始化值：

```
/****************************************************************************/
/* MPU/IPE Specific memory segment definitions                              */
/****************************************************************************/

#ifdef _IPE_ENABLE
   #define IPE_MPUIPLOCK 0x0080
   #define IPE_MPUIPENA 0x0040
```


SLAA685 — http://www-s.ti.com/sc/techlit/SLAA685
版权 © 2015, Texas Instruments Incorporated

```
    #define IPE_MPUIPPUC 0x0020

    // Evaluate settings for the control setting of IP Encapsulation
    #if defined(_IPE_ASSERTPUC1)
         #if defined(_IPE_LOCK ) && (_IPE_ASSERTPUC1 == 0x08))
          fram_ipe_enable_value = (IPE_MPUIPENA | IPE_MPUIPPUC |IPE_MPUIPLOCK);
         #elif defined(_IPE_LOCK )
          fram_ipe_enable_value = (IPE_MPUIPENA | IPE_MPUIPLOCK);
       #elif (_IPE_ASSERTPUC1 == 0x08)
          fram_ipe_enable_value = (IPE_MPUIPENA | IPE_MPUIPPUC);
       #else
          fram_ipe_enable_value = (IPE_MPUIPENA);
       #endif
    #else
       #if defined(_IPE_LOCK )
          fram_ipe_enable_value = (IPE_MPUIPENA | IPE_MPUIPLOCK);
       #else
          fram_ipe_enable_value = (IPE_MPUIPENA);
       #endif
    #endif

    // Segment definitions
    #ifdef _IPE_MANUAL                  // For custom sizes selected in the GUI
       fram_ipe_border1 = (_IPE_SEGB1>>4);
       fram_ipe_border2 = (_IPE_SEGB2>>4);
    #else                           // Automated sizes generated by the Linker
       fram_ipe_border2 = fram_ipe_end >> 4;
       fram_ipe_border1 = fram_ipe_start >> 4;
    #endif

    fram_ipe_settings_struct_address = Ipe_settingsStruct >> 4;
    fram_ipe_checksum = ~((fram_ipe_enable_value & fram_ipe_border2 &
fram_ipe_border1) | (fram_ipe_enable_value & ~fram_ipe_border2 & ~fram_ipe_border1)
| (~fram_ipe_enable_value & fram_ipe_border2 & ~fram_ipe_border1) |
(~fram_ipe_enable_value & ~fram_ipe_border2 & fram_ipe_border1));
#endif
```

用户不必修改这些设置的任何内容，但这展示了链接器文件如何与 CCS IPE 工具设置协作，以确定要进入到 IPE 初始化结构中的值。

SLAA685 — http://www-s.ti.com/sc/techlit/SLAA685
版权 © 2015, Texas Instruments Incorporated

3.1.2 在 CCS 中将代码和数据放到 IPE 部分中

用户必须为编译器指明应将哪些代码和数据放到链接器文件定义的 IPE 部分中。对于 MSP430 TI C/C++ 编译器，通过使用 CODE_SECTION 和 DATA_SECTION 这两个 pragma 来启用此操作。更多有关这些 pragma 的信息，请参阅*MSP430 Optimizing C/C++ Compiler User's Guide* (SLAU132)。更多有关如何在 CCS 中使用此工具启用 IPE 的信息，请参阅*Code Composer Studio v6.1 for MSP430 User's Guide* (SLAU157)。

必须在一个或一组函数中包含应当放到 IPE 区域中的代码。应将这些函数全部放到 .ipe 部分中（请参见如下示例）。

```
#pragma CODE_SECTION(IPE_encapsulatedInit, ".ipe")
void IPE_encapsulatedInit (void)
{
        ...
        ...

}

#pragma CODE_SECTION(IPE_encapsulatedBlink, ".ipe")
void IPE_encapsulatedBlink (void)
{
        ...
        ...
}
```

还应将 IP 封装代码所使用的任何中断服务例程 (ISR) 放到 IPE 区域中，以便同时将它们封装。应将这些 ISR 放到 .ipe:_isr 部分中（请参见如下示例）。

```
#pragma CODE_SECTION(TIMER0_A0_ISR, ".ipe:_isr")
#pragma vector=TIMER0_A0_VECTOR
__interrupt
void TIMER0_A0_ISR(void)
{
        ...
        ...

}
```

某些 应用 可能具有加密密钥等常数，建议将这些常数存储在 IPE 区域中。CCS 链接器文件不允许将常数与变量或代码放在同一个部分中。应将这些常数值放到 .ipe_const 部分中（请参见如下示例）。

```
#pragma DATA_SECTION(IPE_encapsulatedKeys, ".ipe_const")
const uint16_t IPE_encapsulatedKeys[] = {0x0123, 0x4567, 0x89AB, 0xCDEF,
                                         0xAAAA, 0xBBBB, 0xCCCC, 0xDDDD};
```

最后，还应将 IPE 代码使用的变量全部放到 IPE 区域中而不是 RAM 中，以防止代码的其他部分无法读取或访问这些变量。

```
#pragma DATA_SECTION(IPE_encapsulatedCount, ".ipe_vars")
```


SLAA685 — http://www-s.ti.com/sc/techlit/SLAA685
版权 © 2015, Texas Instruments Incorporated

```
unsigned char IPE_encapsulatedCount;

...

#pragma DATA_SECTION(IPE_i, ".ipe_vars")
uint8_t IPE_i;
```

在链接器文件中，将 .ipe_vars 部分设置为 NOINIT，以表明放在这里的任何变量不应有任何初始化值，而且不应被启动代码初始化为 0。这是因为，C 启动代码和 .cinit 初始化表并未放在 IPE 区域中，因此无法写入到 IPE 部分以及设置 IPE 部分中的变量。对用户来说，这意味着应在放在 .ipe 部分中的定制 init 函数中设置放在 .ipe_vars 部分中的任何变量，以使用在 IPE 区域内运行的代码来初始化这些变量。随后，应在启动时由用户的主代码调用此 init 函数，以便在使用这些变量之前（例如在停止看门狗定时器之后）将它们初始化。

```
#pragma DATA_SECTION(IPE_encapsulatedCount, ".ipe_vars")
unsigned char IPE_encapsulatedCount;

/*
 * main.c
 */
void main(void) {
    WDTCTL = WDTPW | WDTHOLD;    // Stop watchdog timer

    IPE_encapsulatedInit();
        ...
        ...
}

#pragma CODE_SECTION(IPE_encapsulatedInit, ".ipe")
void IPE_encapsulatedInit (void)
{
        IPE_encapsulatedCount = 1;
}
```

3.1.3 在 CCS 项目选项中启用 IPE

只需转到Project > Properties > General，然后使用 MSP430 IPE 选项卡，即可在 CCS 项目中启用 IPE。

要实现自动化的 IPE 处理，请选择“Enable Intellectual Property Encapsulation (IPE)”和“Let compiler handle IPE memory partitioning based on user-code and data placement”。单击“OK”，这样就设置了 IPE（请参见图 3）。

SLAA685 — http://www-s.ti.com/sc/techlit/SLAA685
版权 © 2015, Texas Instruments Incorporated

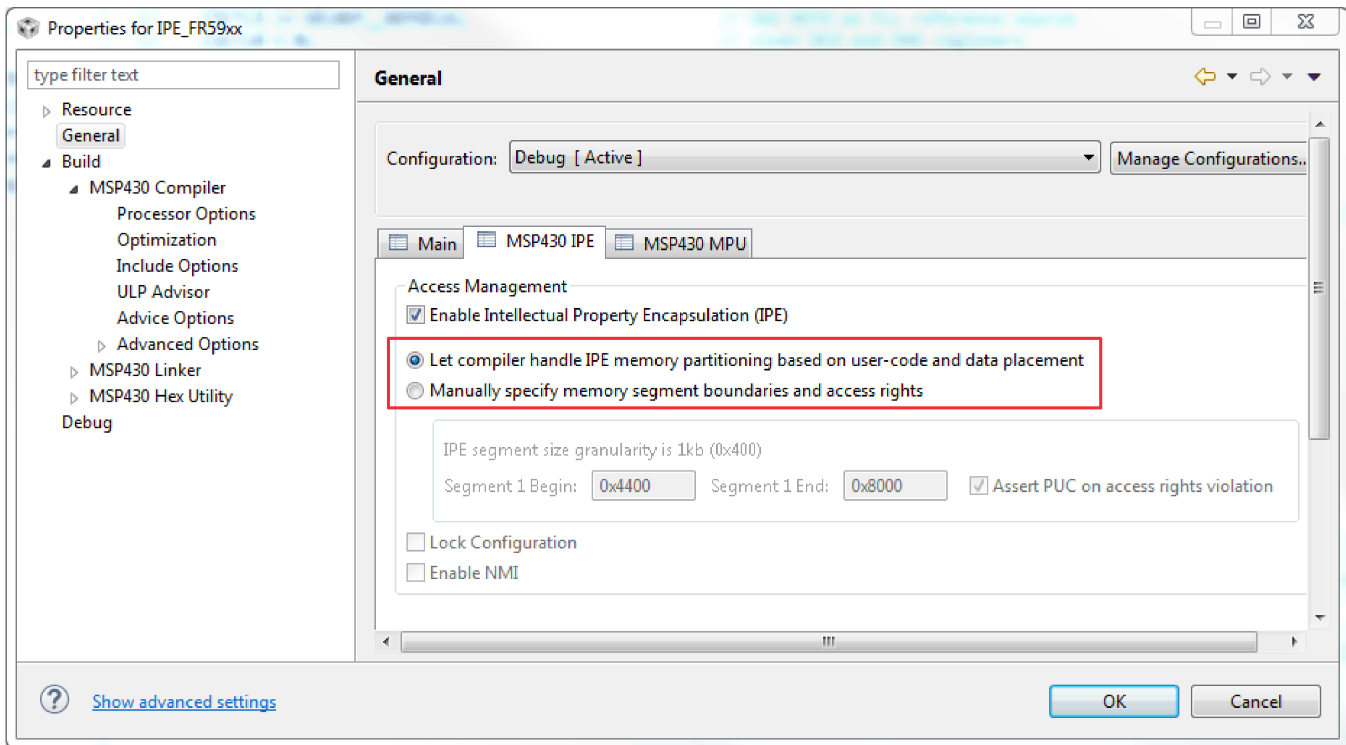

图 **3. CCS** 中的 **IPE** 工具

3.1.4 在 CCS 中查看编译器选择的 IPE 分区

在启用了 IPE 的 CCS 中构建了项目之后，如果需要，用户可以使用 Debug 文件夹中的 .map 文件，查看设置了哪些 IPE 边界以及不同的代码段（例如 .ipe）和代码段内的函数放在了哪些位置。此外，还可以找到 fram_ipe_border1、fram_ipe_settings_struct_address 等参数的值。

```
SECTION ALLOCATION MAP

 output                                  attributes/
section   page    origin      length       input sections
--------  ----  ----------  ----------   ----------------
...
...

.ipestruct
*          0    00004800    00000008
                  00004800    00000008
MSPIPE_INIT_LIB_CCS_msp430_large_code_restricted_data.lib : ipe_init.o
(.ipestruct:retain)

.ipe       0    00004808    000000d2
                  00004808    000000cc     IPE_FR59xx.obj
(.ipe:IPE_encapsulatedBlink)
                  000048d4    00000006     IPE_FR59xx.obj (.ipe:IPE_encapsulatedInit)

.ipe_const
*          0    000048da    00000010
                  000048da    00000010     IPE_FR59xx.obj (.ipe_const)
```


SLAA685 — http://www-s.ti.com/sc/techlit/SLAA685
版权 © 2015, Texas Instruments Incorporated

```
.ipe:_isr
*          0    000048ea    0000000a
                  000048ea    0000000a     IPE_FR59xx.obj (.ipe:_isr:TIMER0_A0_ISR)

.ipe_vars
*          0    000048f4    00000002    UNINITIALIZED
                  000048f4    00000002     IPE_FR59xx.obj (.ipe_vars)...

GLOBAL SYMBOLS: SORTED ALPHABETICALLY BY Name

address   name
-------   ----
...
...
00004808  IPE_encapsulatedBlink
000048f4  IPE_encapsulatedCount
000048d4  IPE_encapsulatedInit
000048da  IPE_encapsulatedKeys
000048f5  IPE_i
0000ff88  Ipe_enableSignature
00004800  Ipe_settingsStruct
0000ff8a  Ipe_structureAddress
...
...
00000480  fram_ipe_border1
000004c0  fram_ipe_border2
ffffffff  fram_ipe_checksum
00000040  fram_ipe_enable_value
00004c00  fram_ipe_end
00000480  fram_ipe_settings_struct_address
00004800  fram_ipe_start
00004400  fram_rw_start
00004c00  fram_rx_start
...
...
```

IPE 结构内置在代码的二进制映像中（.txt 或 .hex），以便能够在启动时被器件启动代码找到，如《*MSP430FR58xx、MSP430FR59xx、MSP430FR68xx 和 MSP430FR69xx 系列用户指南*》(SLAU367) 中所述。也可以转到 Project > Properties > Build > MSP430 Hex Utility 来查看此二进制映像。选择“Enable MSP430 Hex Utility”，然后在“Output Format Options”下方选择“Output TI-TXT hex format (--ti_txt)”。随后会进行构建，Debug 文件夹中应当会出现一个 .txt 文件。

当查看此 txt 文件以查看 IPE 初始化结构时，请验证以下各项：

1. 检查 0xFF8A 处的值。这是 IPE 签名中包含 IPE 结构地址的那一部分。请记住，此值已右移 4 位。在此示例中，0xFF8A 处的值为 0x0480，指向 IPE 结构的地址 0x4800。注意此值如何与 .map 文件匹配。

```
@ff80
FF FF FF FF FF FF FF FF AA AA 80 04 FF FF FF FF
```

2. 转到 IPE 初始化结构的地址（我们在步骤 1 中发现的地址，在本例中为 0x4800），并查看此结构中的值。它将具有来自用户指南的格式。

SLAA685 — http://www-s.ti.com/sc/techlit/SLAA685
版权 © 2015, Texas Instruments Incorporated

表 7. IPE 初始化结构

字段名称	地址偏移量	长度	说明
MPUIPC0	0h	文字	用于 IP 封装的控制设置。值将写入到 MPUIPC0
MPUIPB2	2h	文字	IP 封装段的上边界。值将写入到 MPUIPSEGB2
MPUIPB1	4h	文字	IP 封装段的下边界。值将写入到 MPUIPSEGB1
MPUCHECK	6h	文字	偶位交错奇偶校验

在下面的示例中，您可以看到：

- MPUIPC0 = 0x0040 → MPUIPENA = 1 = IPE 启用
- MPUIPB2 = 0x04C0 → 地址 0x4C00 是 IPE 终点
- MPUIPB1 = 0x0480 → 地址 0x4800 是 IPE 起点
- MPUCHECK = 0xFFFF → 以前的数据的校验和。在校验和低位字节 = INV(Byte 0 XOR Byte 2 XOR Byte 4) = INV(40 XOR C0 XOR 80) = 0xFF、校验和高位字节 = INV(Byte 1 XOR Byte 3 XOR Byte 5) = INV(00 XOR 04 XOR 04) = 0xFF 时计算得出

```
@4800
40 00 C0 04 80 04 FF FF
```

3.1.5 在 CCS 中运行和测试 IPE 代码

此代码将在 MSP-EXP430FR5969 LaunchPad 开发套件上运行。

1. 下载示例 (http://www.ti.com/cn/lit/zip/slaa685)
2. 将项目导入到 CCSv6 或更高版本中：
   - Project > Import CCS Projects…
   - 浏览... 并选择您下载并解压缩的代码所在的位置。确保选中“Copy projects into workspace”复选框，然后单击“OK”。
3. 构建项目。

3.1.5.1 在 CCS 中使用 IPE 调试设置

如《*MSP430FR58xx、MSP430FR59xx、MSP430FR68xx 和 MSP430FR69xx 系列用户指南*》(SLAU367) 中所述，只有当复位器件以使启动代码能够运行之后，IPE 设置才会生效。为了测试 IPE 是否能够防止 JTAG 访问受保护区域，示例项目提供了两个不同的调试配置，以简化加载器件和测试 IPE 特性的过程：

- IPE_FR59xx_load - 使用新的代码（包括 IPE 代码）对器件进行编程。
- IPE_FR59xx_test - 调试器件，但不对器件重新编程。用于测试 IPE 保护。

在 CCS 中，选择调试错误图标旁边的下拉箭头，然后选择“Debug configurations”（请参见图 4）。

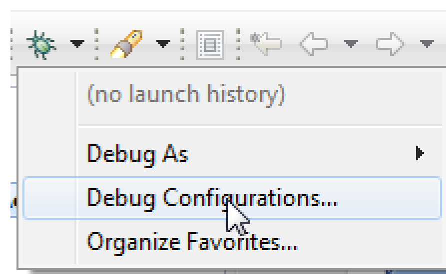

图 4. CCS 中的调试配置

版权 © 2015, Texas Instruments Incorporated

在新窗口（请参见图 5）中，此项目有两个调试配置：IPE_FR59xx_load 和 IPE_FR59xx_test。这两个配置都与默认调试配置有所不同，这样有助于在启用了 IPE 的情况下进行调试。

IPE_FR59xx_load 会将器件配置为在尝试对器件进行编程之前擦除 IP 受保护区域。否则，如果存在 IPE 代码，程序加载将失败，原因在于 IPE 禁止从 JTAG 进行编程访问。不过，工具链可以执行特殊擦除以批量擦除器件，还将擦除受到保护并存储着所保存的 IPE 结构指针的非易失性系统数据区域 [请参阅 *MSP430FR58xx, MSP430FR59xx, MSP430FR68xx, and MSP430FR69xx Family User's Guide* (SLAU367) 中的"Trapdoor Mechanism for IP Structure Pointer Transfer"]。要进行此设置，请在 MSP430 Properties > Connection Options 下方的 Target 选项卡上的debug configuration中，选择"On connect, erase main, information, and IP protected area"。

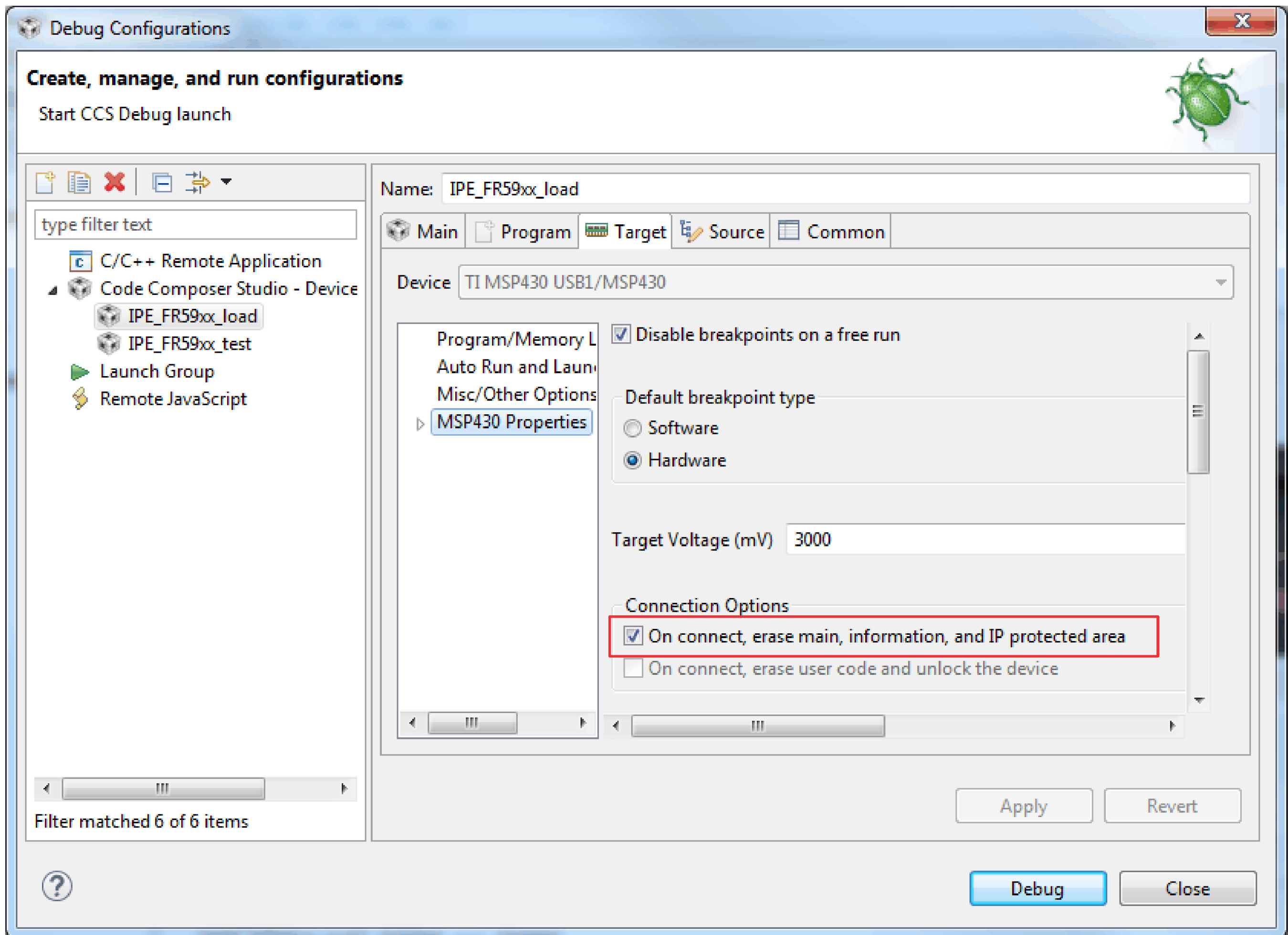

图 5. CCS 中的 IPE 加载配置

另一方面，IPE_FR59xx_test 可确保不执行此连接时擦除操作，以使用户能够观察 IPE 的行为（请参见图 6）。此构建配置不会选中"On connect, erase main, information, and IP protected area"设置。相反，IPE_FR59xx_test 与默认配置之间的区别在于，IPE_FR59xx_test 不会尝试加载程序（当启用了 IPE 之后，此操作会失败）。在本例中，用户需要在不下载的情况下进行调试。要进行此设置，请在调试配置中的"Program"选项卡上的"Loading options"下方，选择"Load symbols only"。

SLAA685 — http://www-s.ti.com/sc/techlit/SLAA685
版权 © 2015, Texas Instruments Incorporated

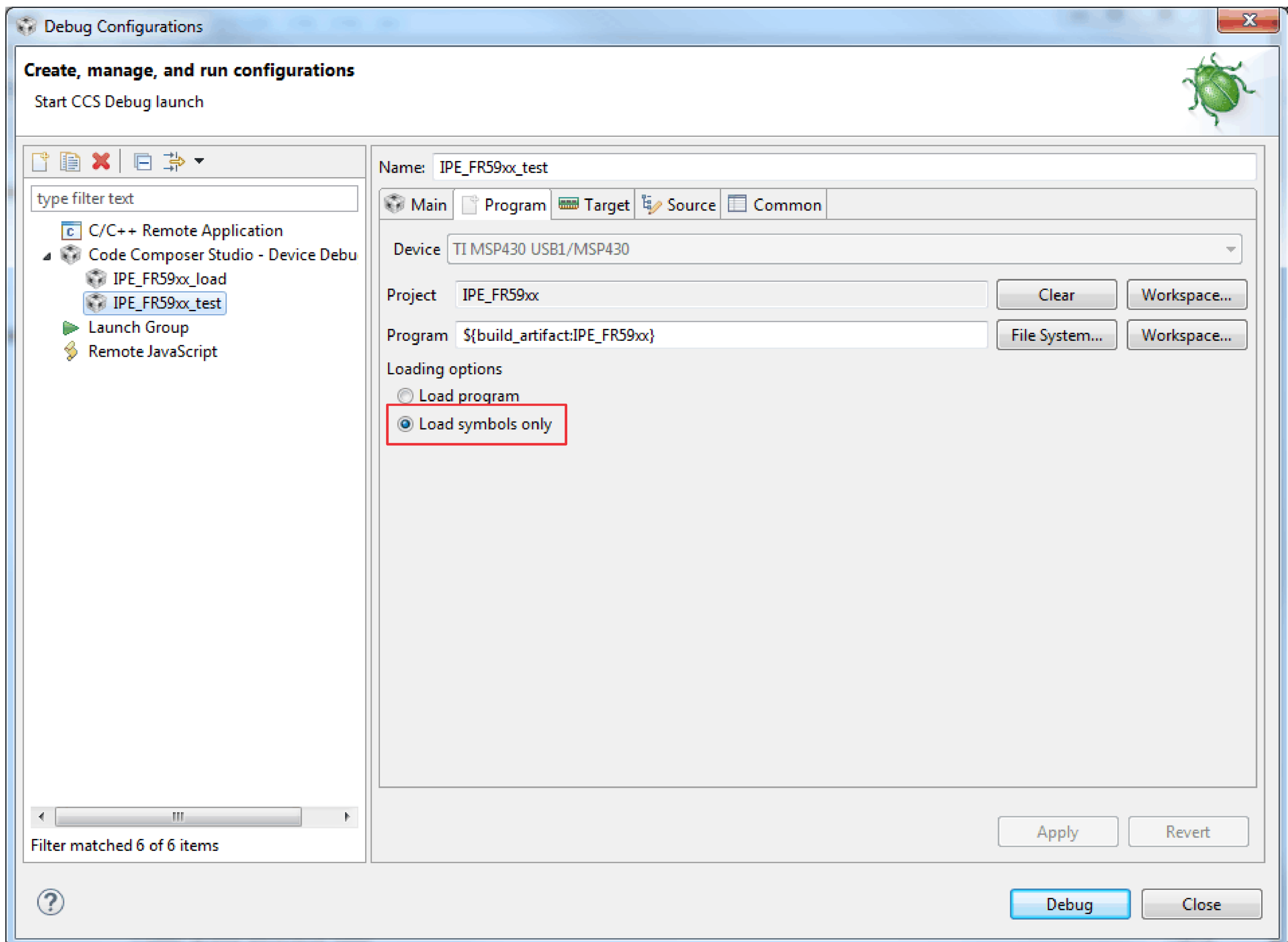

图 **6. CCS** 中的 **IPE** 测试配置

为了便于使用 IPE 开发启用了 IPE 的任何 CCS 项目，TI 建议使用此处所示的设置建立两个构建配置。

3.1.5.2 在 CCS 中测试 IPE

1. 单击调试图标旁边的下拉箭头，转到“Debug Configurations”，然后选择 IPE_FR59xx_load 调试配置。进行调试。
2. 此程序将加载到器件中。运行代码。LaunchPad 上的 LED 应当闪烁。
3. 暂停代码。
    1. 选择“View > Memory Browser以打开“Memory”视图。在“Memory”视图中，输入 IPE 受保护变量或函数的名称（IPE_encapsulatedInit、IPE_encapsulatedBlink 等等）。
    2. 注意 IP 封装变量和函数仍然可见。这是因为，启动代码必须在首次运行时将 IPE 结构指针加载到受到保护的内部非易失性系统数据区域，然后从 IPE 结构加载 IPE 寄存器，IPE 才会生效（需要复位以使启动代码能够运行）。

SLAA685 — http://www-s.ti.com/sc/techlit/SLAA685
版权 © 2015, Texas Instruments Incorporated

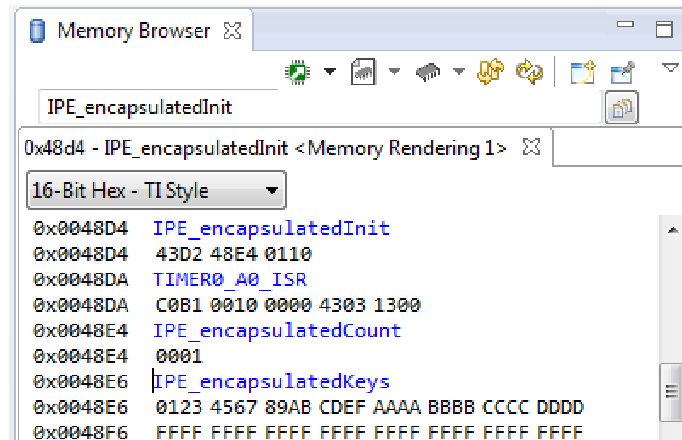

图 **7. CCS** 中的 **IPE** 内存视图 **-** 无保护

4. 终止调试会话。
5. 对器件执行循环通电。IPE 现在应处于激活状态，并禁止对器件的 IPE 区域执行读取和写入访问。
6. 单击调试图标旁边的下拉箭头，转到“Debug Configurations”，然后选择 IPE_FR59xx_test 调试配置。进行调试。
7. 此时不会加载任何程序，只会加载调试符号。运行代码。LaunchPad 上的 LED 应当闪烁。
8. 暂停代码。
   - 选择View > Memory Browser 以打开“Memory”视图。在“Memory”视图中，输入 IPE 受保护变量或函数的名称（IPE_encapsulatedInit、IPE_encapsulatedBlink 等等）。
   - 注意 IP 封装变量和函数不再可见。在这些区域中，此工具只能看见 0x3FFF (JMP$)。IP 封装处于激活状态，并禁止读取内存区域。

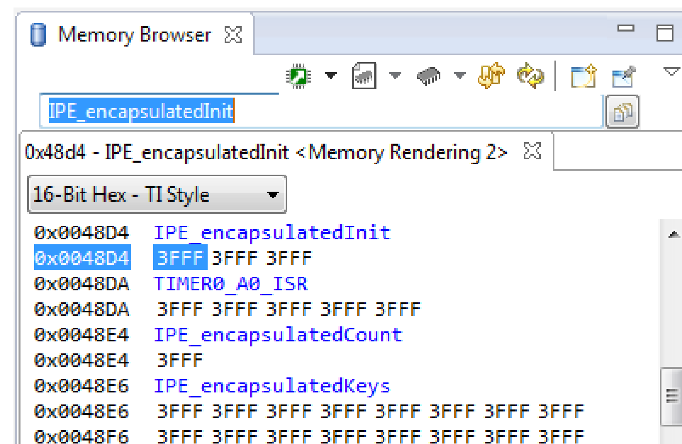

图 **8. CCS** 中的 **IPE** 内存视图 **-** 受保护

注: 当加载另一个用户项目时，如果在此前加载到器件中的代码中启用了 IPE，则 IPE 区域会导致无法正确加载新项目，并在 CCS 中报告错误（因为无法擦除或写入到 IPE 区域）。要首次加载新项目，请在调试设置中为首次加载的新项目选择“On connect, erase main, information, and IP protected area”，以擦除 IPE 代码和设置。

SLAA685 — http://www-s.ti.com/sc/techlit/SLAA685
版权 © 2015, Texas Instruments Incorporated

3.2 使用 IAR 中的 IPE 工具进行 IP 封装

IAR 具有一个内置的 IPE 工具，并在链接器文件中提供了预定义的区段，以便于在项目中启用 IPE。尽管《MSP430FR58xx、MSP430FR59xx、MSP430FR68xx 和 MSP430FR69xx 系列用户指南》详细说明了如何设置 IPE 初始化结构以及如何利用 MPU 寄存器手动启用 IPE，但使用 IAR 中的内置 IPE 工具仍不失为更轻松、更自动化的替代方法。随后的几节将讨论如何使用 IAR 中的这些工具在项目中启用 IPE。另请参阅 zip 文件 (http://www.ti.com/cn/lit/zip/slaa685)，以了解 IAR 中使用 IPE 的示例项目。此代码将在 MSP-EXP430FR5969 LaunchPad 开发套件上运行。更多有关如何使用此工具在 IAR 中启用 IPE 的信息，请参阅 IAR Embedded Workbench™IDE“Help”菜单中的 IAR C/C++ Compiler User Guide。

3.2.1 用于 IPE 的 IAR 链接器文件 特性

注: 这些链接器文件 特性 在 IAR v6.40.1 发布时是最新的版本。早期版本的 IAR 链接器文件并不具备所有这些 特性，因此建议在通过这种方法启用 IPE 时使用 IAR v6.40.1 或更高版本。

对于支持 IPE 的器件，CCS 链接器 .xcl 文件包含一些用来放置 IPE 代码和数据的部分。用户代码只需告诉编译器应将哪些变量和函数放到这些预定义的部分中。如下代码片段来自 MSP430FR5969 链接器 xcl 文件：

```
// ---------------------------
// Intellectual Property Encapsulation (IPE)
//

-Z(CONST)IPE_B1=4400-FF7F
-Z(DATA)IPEDATA16_N
-Z(CODE)IPECODE16
-Z(CONST)IPEDATA16_C,IPE_B2
```

- **(CONST)IPE_B1** - 此代码段是 IPE 区域的起点，还包含 IPE 初始化结构，此结构包含有关 IP 封装存储器边界和控制设置的地址位置信息。*《MSP430FR58xx、MSP430FR59xx、MSP430FR68xx 和 MSP430FR69xx 系列用户指南》* (SLAU367) 中提供了更多有关 IPE 初始化结构的信息。用户不应将任何数据放在此部分中。当使用内置的 IAR IPE 工具时，此工具会生成所需的值，并将它们放在此位置。
- **(DATA)IPEDATA16_N** - 此代码段用于需要封装的任何数据变量（应使用 __no_init 选项来声明这些变量）。为了获得最高的安全性，还应将只在 IP 封装代码内使用的任何变量放到 IP 封装区域中，而不是放到 RAM 中。否则有人可能利用所观察到的 RAM 变化尝试对封装代码的功能进行逆向工程。当然，这取决于应用。
- **(CODE)IPECODE16** - 此代码段用于用户希望放在 IP 封装区域中的任何代码函数。需要成为 IP 封装用户代码的一部分的任何中断服务例程 (ISR) 也要放在此位置。
- **(CONST)IPEDATA16_C** - 此代码段用于需要封装的任何常量数据。这可能包括加密密钥或密码等内容或者其他敏感数据。
- **(CONST)IPE_B2** - 这是代表 IPE 区域终点的边界。

3.2.2 在 IAR 中将代码和数据放到 IPE 部分中

用户必须为编译器指明应将哪些代码和数据放到链接器文件定义的 IPE 部分中。对于 IAR C/C++ 编译器，通过使用 #pragma 位置来启用此操作。更多有关此 pragma 以及如何在 IAR 中启用 IPE 的信息，请参阅 IAR 中的“Help”菜单下的IAR C/C++ Compiler User Guide。

版权 © 2015, Texas Instruments Incorporated

必须在一个或一组函数中包含应当放到 IPE 区域中的代码。应将这些函数全部放到 IPECODE16 部分中（请参见如下示例）。

```
#pragma location = "IPECODE16"
void IPE_encapsulatedInit(void)
{
        ...
        ...

}

#pragma location = "IPECODE16"
void IPE_encapsulatedBlink (void)
{
        ...
        ...
}
```

还应将 IP 封装代码所使用的任何中断服务例程 (ISR) 放到 IPE 区域中，以便同时将它们封装。还应将这些 ISR 放到 IPECODE16 部分中（请参见如下示例）。

```
#pragma location = "IPECODE16"
#pragma vector=TIMER0_A0_VECTOR
__interrupt
void TIMER0_A0_ISR(void)
{
        ...
        ...
}
```

某些 应用 可能具有加密密钥等常数，建议将这些常数存储在 IPE 区域中。IAR 链接器文件不允许将常数与变量或代码放在同一个部分中。应将这些常数值放到 IPEDATA16_C 部分中（请参见如下示例）。

```
#pragma location = "IPEDATA16_C"
const uint16_t IPE_encapsulatedKeys[] = {0x0123, 0x4567, 0x89AB, 0xCDEF,
                                         0xAAAA, 0xBBBB, 0xCCCC, 0xDDDD};
```

最后，还应将 IPE 代码使用的变量全部放到 IPEDATA16_N 部分中而不是 RAM 中，以防止代码的其他部分无法读取或访问这些变量。请注意，应使用 __no_init 关键字标记这些变量。

```
#pragma location = "IPEDATA16_N"
__no_init unsigned char IPE_encapsulatedCount;
```


版权 © 2015, Texas Instruments Incorporated

应使用 __no_init 关键字标记这些变量，以表明放在这里的任何变量不应有任何初始化值，而且不应被启动代码初始化为 0。这是因为，C 启动代码和 .cinit 初始化表并未放在 IPE 区域中，因此无法写入到 IPE 部分以及设置 IPE 部分中的变量。对用户来说，这意味着应在放在 IPECODE16 部分中的定制 init 函数中设置放在 IPEDATA16_N 部分中的任何变量，以使用在 IPE 区域内运行的代码来初始化这些变量。随后，应在启动时由用户的主代码调用此 init 函数，以便在使用这些变量之前（例如在停止看门狗定时器之后）将它们初始化。

```
#pragma location = "IPEDATA16_N"
__no_init unsigned char IPE_encapsulatedCount;

/*
 * main.c
 */
void main(void) {
    WDTCTL = WDTPW | WDTHOLD;    // Stop watchdog timer

    IPE_encapsulatedInit();
        ...
        ...
}

#pragma location = "IPECODE16"
void IPE_encapsulatedInit(void)
{
        IPE_encapsulatedCount = 1;
}
```


SLAA685 — http://www-s.ti.com/sc/techlit/SLAA685
版权 © 2015, Texas Instruments Incorporated

3.2.3 在 IAR 项目选项中启用 IPE

只需转到 Project > Options > General Options，然后使用 MPU/IPE 选项卡，即可在 IAR 项目中启用 IPE。

要实现自动化的 IPE 处理，请选择“Support IPE”和“Enable IPE”。单击“OK”，这样就设置了 IPE。

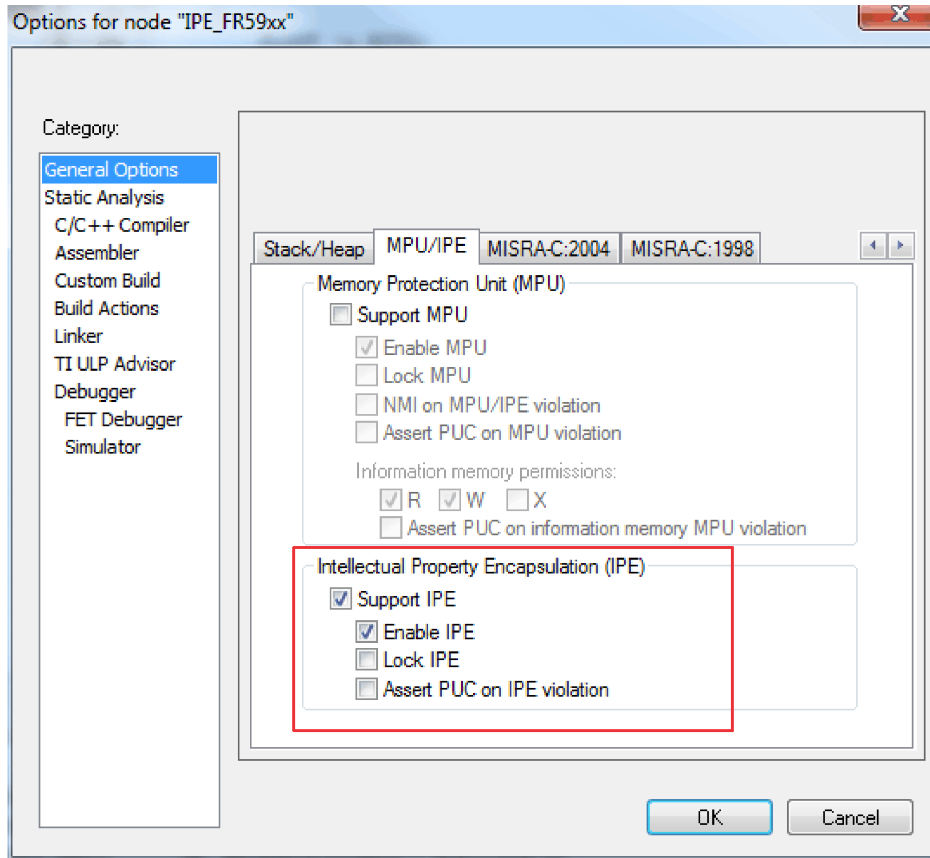

图 **9. IAR** 中的 **IPE** 工具

3.2.4 在 IAR 中查看编译器选择的 IPE 分区

在启用了 IPE 的 IAR 中构建了项目之后，如果需要，用户可以使用 Output 文件夹中的 .map 文件，查看设置了哪些 IPE 边界以及不同的代码段（例如 IPECODE16）和代码段内的函数放在了哪些位置。此外，还可以找到 __iar_430_MPUIPC0_value（此参数会显示 IPE 寄存器设置值）、IPE_B1 和 IPEB2 边界等参数的值。

要生成 .map 文件，请转到 Project > Options > Linker，然后在“List”选项卡上选择“Generate linker listing”。“Segment map”应当也已被选中。随后会进行构建，Output 文件夹中应当会出现一个 .map 文件。

SLAA685 — http://www-s.ti.com/sc/techlit/SLAA685
版权 © 2015, Texas Instruments Incorporated

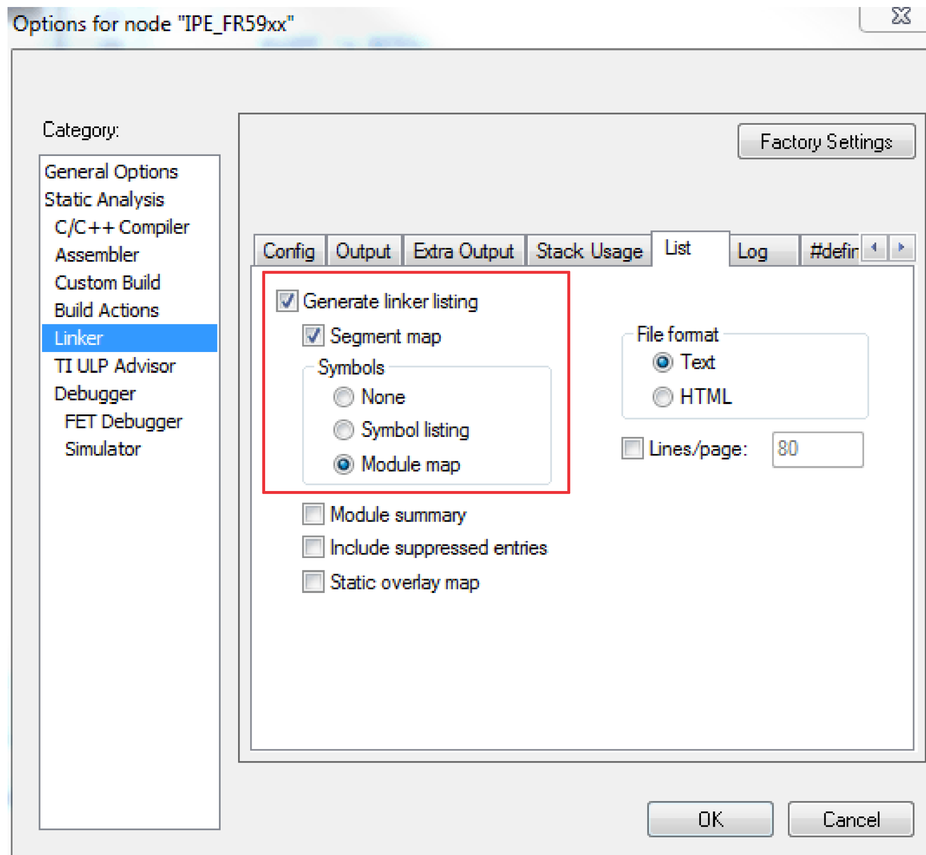

图 **10.** 在 **IAR** 中生成 **.map** 文件

```
                 ****************************************
                 *                                      *
                 *              MODULE MAP              *
                 *                                      *
                 ****************************************
...
...

    -------------------------------------------------------------------------
IPEDATA16_N
  Relative segment, address: 4808 - 4808 (0x1 bytes), align: 0
  Segment part 17.            Intra module refs:   IPE_encapsulatedBlink
                                                   IPE_encapsulatedInit
           ENTRY                    ADDRESS        REF BY
           =====                    =======        ======
           IPE_encapsulatedCount    4808
    -------------------------------------------------------------------------
IPEDATA16_N
  Relative segment, address: 4809 - 4809 (0x1 bytes), align: 0
  Segment part 18.            Intra module refs:   IPE_encapsulatedBlink
           ENTRY                    ADDRESS        REF BY
           =====                    =======        ======
           IPE_i                    4809
    -------------------------------------------------------------------------
IPEDATA16_C
  Relative segment, address: 48DE - 48ED (0x10 bytes), align: 1
```


SLAA685 — http://www-s.ti.com/sc/techlit/SLAA685
版权 © 2015, Texas Instruments Incorporated

```
  Segment part 19.            Intra module refs:   IPE_encapsulatedBlink
           ENTRY                        ADDRESS          REF BY
           =====                        =======          ======
           IPE_encapsulatedKeys         48DE
    -------------------------------------------------------------------------...
...
...
    -------------------------------------------------------------------------
IPECODE16
  Relative segment, address: 480A - 480F (0x6 bytes), align: 1
  Segment part 25.            Intra module refs:   main
           ENTRY                        ADDRESS          REF BY
           =====                        =======          ======
           IPE_encapsulatedInit         480A
    -------------------------------------------------------------------------
IPECODE16
  Relative segment, address: 4810 - 48D3 (0xc4 bytes), align: 1
  Segment part 24.            Intra module refs:   main
           ENTRY                        ADDRESS          REF BY
           =====                        =======          ======
           IPE_encapsulatedBlink        4810
    -------------------------------------------------------------------------
IPECODE16
  Relative segment, address: 48D4 - 48DD (0xa bytes), align: 1
  Segment part 23.            Intra module refs:   TIMER0_A0_ISR::??INTVEC 90
           ENTRY                        ADDRESS          REF BY
           =====                        =======          ======
           TIMER0_A0_ISR                48D4
               interrupt function
    -------------------------------------------------------------------------...
...
...
                *****************************************
                *                                       *
                *       SEGMENTS IN ADDRESS ORDER       *
                *                                       *
                *****************************************


SEGMENT                SPACE    START ADDRESS   END ADDRESS     SIZE  TYPE  ALIGN
=======                =====    =============   ===========     ====  ====  =====
...
...
IPE_B1                                   4800 - 4807               8   rel    10
IPEDATA16_N                              4808 - 4809               2   rel     0
IPECODE16                                480A - 48DD              D4   rel     1
IPEDATA16_C                              48DE - 48ED              10   rel     1
IPE_B2                                      4C00                       rel    10
```

IPE 结构内置在代码的二进制映像中（.txt 或 .hex），以便能够在启动时被器件启动代码找到，如 *《MSP430FR58xx、MSP430FR59xx、MSP430FR68xx 和 MSP430FR69xx 系列用户指南》* (SLAU367) 中所述。也可以转到Project > Options > Linker，然后在“Output”选项卡上选择“Allow C-SPY-specific extra output file”，以查看此二进制文件。

SLAA685 — http://www-s.ti.com/sc/techlit/SLAA685
版权 © 2015, Texas Instruments Incorporated

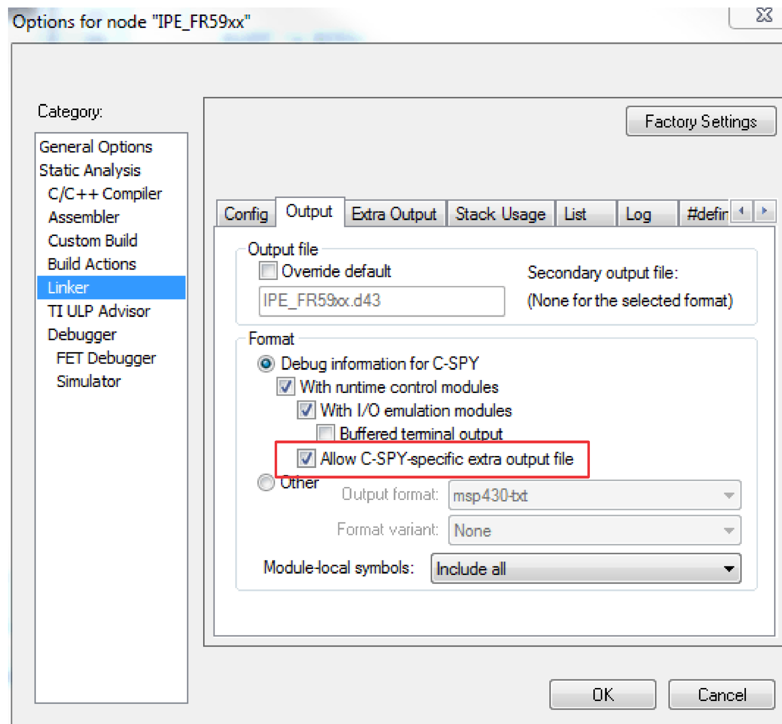

图 **11.** 用于 **IPE_FR59xx** 的选项

在“Extra Output”选项卡上，选择“Generate extra output file”，并将“输出”格式设置为“msp430-txt”。随后会进行构建，Output 文件夹中应当会出现一个 .txt 文件。

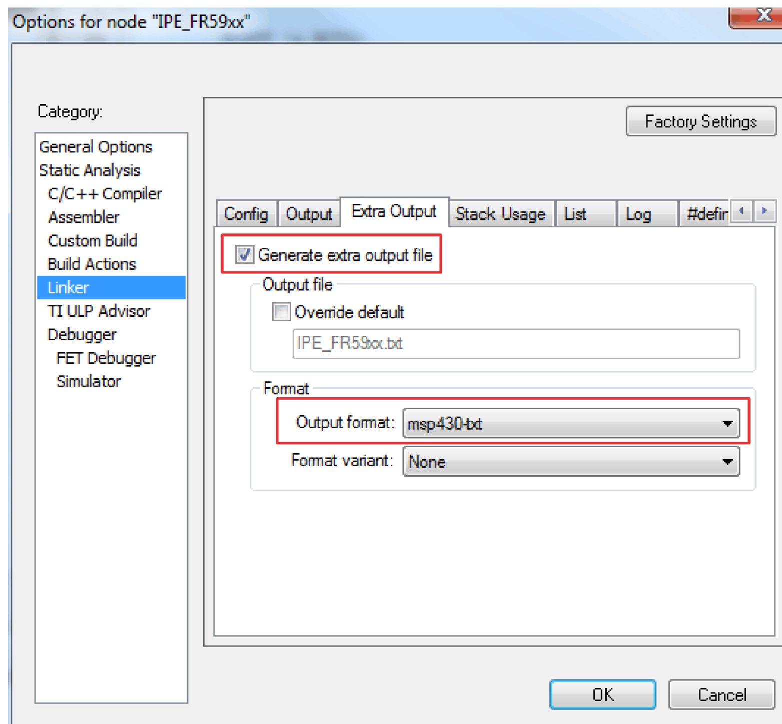

图 **12.** 在 **IAR** 中生成 **.txt** 二进制文件

SLAA685 — http://www-s.ti.com/sc/techlit/SLAA685
版权 © 2015, Texas Instruments Incorporated

当查看此 txt 文件以查看 IPE 初始化结构时，请验证以下各项：

1. 检查 0xFF8A 处的值。这是 IPE 签名中包含 IPE 结构地址的那一部分。请记住，此值已右移 4 位。在此示例中，0xFF8A 处的值为 0x0480，指向 IPE 结构的地址 0x4800。

```
@FF88
AA AA 80 04
```

2. 转到 IPE 初始化结构的地址（在步骤 1 中发现的地址，在本例中为 0x4800），并查看此结构中的值。它具有来自用户指南的格式。

表 8. IPE 初始化结构

字段名称	地址偏移量	长度	说明
MPUIPC0	0h	文字	用于 IP 封装的控制设置。值将写入到 MPUIPC0
MPUIPB2	2h	文字	IP 封装段的上边界。值将写入到 MPUIPSEGB2
MPUIPB1	4h	文字	IP 封装段的下边界。值将写入到 MPUIPSEGB1
MPUCHECK	6h	文字	偶位交错奇偶校验

在下面的示例中，您可以看到：

- MPUIPC0 = 0x0040 - MPUIPENA = 1 = IPE 启用
- MPUIPB2 = 0x04C0 → 地址 0x4C00 是 IPE 终点
- MPUIPB1 = 0x0480 → 地址 0x4800 是 IPE 起点
- MPUCHECK = 0xFFFF → 以前的数据的校验和。在校验和低位字节 = INV(Byte 0 XOR Byte 2 XOR Byte 4) = INV(40 XOR C0 XOR 80) = 0xFF、校验和高位字节 = INV(Byte 1 XOR Byte 3 XOR Byte 5) = INV(00 XOR 04 XOR 04) = 0xFF 时计算得出

```
@4800
40 00 C0 04 80 04 FF FF
```

3.2.5 在 IAR 中运行和测试 IPE 代码

此代码将在 MSP-EXP430FR5969 LaunchPad 开发套件上运行。

1. 下载示例 (http://www.ti.com/cn/lit/zip/slaa685)。
2. 在 IAR EW430 6.30.2 或更高版本中打开工作区 IPE_FR59xx.eww。
3. 构建项目。

3.2.5.1 在 IAR 中使用 IPE 调试设置

项目必须将器件配置为在尝试对器件进行编程之前擦除 IP 受保护区域。否则，如果存在 IPE 代码，程序加载将失败，原因在于 IPE 禁止从 JTAG 进行编程访问。不过，工具链可以执行特殊擦除以批量擦除器件，还将擦除受到保护并存储着所保存的 IPE 结构指针的非易失性系统数据区域 [请参阅*MSP430FR58xx, MSP430FR59xx, MSP430FR68xx, and MSP430FR69xx Family User's Guide* (SLAU367) 中的“Trapdoor Mechanism for IP Structure Pointer Transfer”]。请在“Download”选项卡上的Project > Options > Debugger > FET Debugger中进行此设置。选择“Erase main and Information memory inc，包括IP PROTECTED area”。

SLAA685 — http://www-s.ti.com/sc/techlit/SLAA685
版权 © 2015, Texas Instruments Incorporated

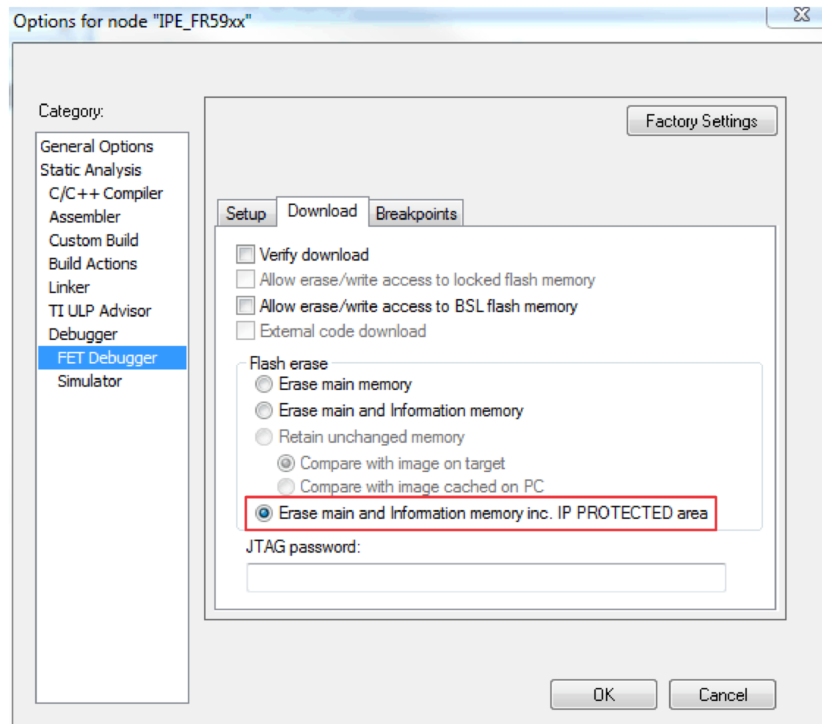

图 **13. IAR** 中的 **IPE** 调试配置

3.2.5.2 在 IAR 中测试 IPE

1. 单击“Download and Debug”图标。
2. 此程序将加载到器件中，并运行代码。LaunchPad 上的 LED 应当闪烁。
3. 暂停代码。
    1. 选择“View > Memory以打开“Memory”视图。在“Memory”视图中，输入 IPE 受保护变量或函数的名称（IPE_encapsulatedInit、IPE_encapsulatedBlink 等等）或者诸如 0x4800 的地址。
    2. 注意 IP 封装变量和函数不可见。在这些区域中，此工具只能看见 0x3FFF (JMP$)。IP 封装处于激活状态，并禁止读取内存区域。

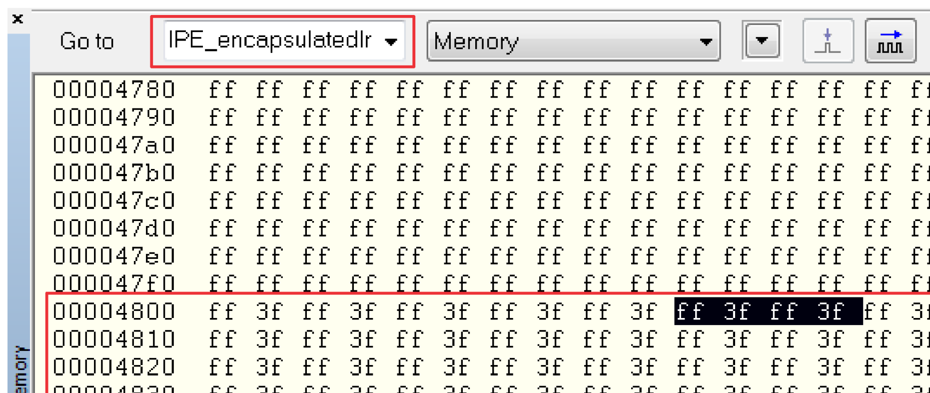

图 **14. IAR** 中的 **IPE** 内存视图 **-** 受保护

SLAA685 — http://www-s.ti.com/sc/techlit/SLAA685
版权 © 2015, Texas Instruments Incorporated

注： 当加载另一个用户项目时，如果在此前加载到器件中的代码中启用了 IPE，则 IPE 区域会导致无法正确加载新项目，并在 IAR 中报告错误（因为无法擦除或写入到 IPE 区域）。要首次加载新项目，请在调试设置中为首次加载的新项目选择“Erase main and Information memory inc. 包括IP PROTECTED area”，以擦除 IPE 代码和设置。

4 引导加载程序 (BSL) 安全性 特性

即使已锁定或禁用 JTAG/SBW 访问，用户也可以使用 MSP430 引导加载程序执行现场固件更新。当用户尝试在最终客户现场退回的器件（已在生产时锁定或禁用 JTAG/SBW 访问）上读取存储器或执行测试时，此工具也会非常有用。BSL 提供了多种 特性 ，有助于防止此 BSL 访问功能被滥用。更多有关特定器件上的 BSL 实施方法的信息，请参阅特定器件的相应 BSL 用户指南 [*MSP430 Programming With the Bootloader (BSL)* (SLAU319) 或*MSP430FR57xx, FR58xx, FR59xx, FR68xx, and FR69xx Bootloader (BSL) User's Guide*(SLAU550)]。另请参阅器件数据表，以确定器件是否具有内置的默认 BSL。大多数 MSP430 器件都实施了 BSL，但有些器件（例如 MSP430G2xx1/G2xx2 和 MSP430i2040）不支持内置的 BSL。

注： 有关如何保护 MSP432 器件或使用 MSP432 器件 BSL 的信息，请参阅应用报告*Configuring Security and Bootloader (BSL) on MSP432P4xx* (SLAA659)。

4.1 密码保护

所有 MSP430 引导加载程序都为一些命令提供了密码保护。任何允许读取器件的命令（例如 TX_DATA_BLOCK）或控制器件的命令（例如 Load PC 或 RX_DATA_BLOCK）通常都受到保护，除非已经提供正确的密码，否则它们不会执行。仅有的几个不受保护的命令通常包括 RX_PASSWORD（用于接收 BSL 密码和解锁其他命令）、CHANGE_BAUD_RATE（用于更改 BSL 波特率以便于进行 BSL 通信）和 MASS_ERASE（用于彻底擦除器件存储器）。

密码由多个字节（通常 32 个字节）组成，位于中断矢量表的末尾，往前不超过存储器地址 0xFFFF（请参阅 BSL 用户指南，详细了解器件的具体地址信息）。几乎所有 MSP430 BSL 的密码都由地址 0xFFE0-0xFFFF 的数据组成（唯一的例外是 MSP430F54xx 非 A 器件，这些器件只有一个 16 字节的密码）。

由于 BSL 密码由来自器件中断矢量表 (IVT) 的值组成，因此在已编程的器件上，对于地址 0xFFFE 处的复位矢量（除应用程序中使用的任何其他中断矢量以外），IVT 始终具有至少一个除 0xFFFF 以外的值，这样就总是有一个 BSL 密码是通过将程序加载到器件中而设置的。

在 CCS 中，如果任何中断矢量没有用户定义的 ISR，则 TI C/C++ 编译器会自动添加一个陷阱 ISR，以捕捉所有这些未使用的矢量。因此，未使用的矢量具有一个除 0xFFFF 以外的值，但此值对于所有未使用的矢量都是相同的。这样就为密码添加了更多的值，但如果用户代码未使用太多中断，则它可能仍未出现明显的变化。

```
@FFE0
1A 4C 1A 4C 1A 4C 1A 4C 1A 4C 9C 48 1A 4C 1A 4C
1A 4C 1A 4C 1A 4C 1A 4C 1A 4C 1A 4C 1A 4C 00 4C
```


版权 © 2015, Texas Instruments Incorporated

在上述的 TI-txt 文件代码片段中，观察包含 BSL 密码的 IVT 区域 - 在 0xFFE0-0xFFFF 之间有 32 个字节。本例中的 489Ch 和 4C00h 这两个值以绿色文本突出显示（这两个值用在中断矢量中），分别用于计时器 ISR 之一以及复位矢量。注意 FFFEh 处的复位矢量表明启动代码始于 4C00h。所有其他未使用的 IVT 条目中填充了 4C1Ah - 这是陷阱 ISR 的地址。在 .map 文件中也可以找到陷阱 ISR 的地址，此地址标记为 __TI_ISR_TRAP。在 _isr 部分的下方还可以找到其他 ISR 地址。

```
.text:_isr
*          0     00004c00     00000020
                   00004c00     0000001a     rts430x_lc_rd_eabi.lib : boot_special.obj
(.text:_isr:_c_int00_noargs_noexit)
                   00004c1a     00000006                            : isr_trap.obj
(.text:_isr:__TI_ISR_TRAP)
```

如果需要定制 BSL 密码，可以将 IVT 的未使用条目修改为密码的其他值。但这种情况的危险性在于，如果发生中断，会将 IVT 条目用作执行代码时要跳转到的地址，因此在这里放置随机值可能会导致代码胡乱运行或者将器件复位，当出现错误的中断时尤其如此（例如当意外启用了中断时）。尽管这种情况不太可能出现，但仍会构成风险，因此用户必须根据自己的系统和应用做出决定。用户也可以使用 BSL 密码区域 FFE0h-FFFFh 中的 IVT 条目，为所有未使用的中断定义单独的 ISR，并利用链接器文件修改和 pragma 将这些 ISR 放到所需的地址，以创建所需的 BSL 密码。

4.2 密码错误时批量擦除

默认情况下，在大多数 MSP430 引导加载程序上，错误的 BSL 密码会导致器件批量擦除。批量擦除操作会擦除全部代码。此特性有助于防止他人利用“蛮力破解”尝试每一组的 32 个字节以猜测密码，因此提供了额外的一层安全性。如果首次尝试时的密码不正确，将会批量擦除器件，这样现在就没有任何可读取的代码了。请参阅相应器件的用户指南，以确定特定的器件是否支持此特性。BSL 批量擦除操作并不会擦除 IPE 区域，IPE 安全性设置（如果已启用）将会保留下来。

批量擦除操作还会擦除 IVT，因此只要输入一次错误的密码，固件就将消失，BSL 密码的所有 32 个字节会随即恢复为默认的 FFh 空值。这意味着，输入一次错误的密码之后，可以使用默认密码来访问器件 - 当器件中可能存在错误的负载或损坏的固件、BSL 密码可能不正确但仍允许加载新固件时，这样会有所帮助。但如果担心未经授权的来源能够将新固件加载到器件中（在这种情况下，可能需要采取其他措施），则应将这种情况视为一种可能性。

支持“密码错误时批量擦除”功能的引导加载程序还可以在必要时禁用此功能。要执行此操作，请在器件存储器中设置一个 BSL 签名，以表明不应对错误的 BSL 密码执行批量擦除。在 BSL 用户指南或器件用户指南中，可以找到更多有关如何禁用此功能的信息。请注意，在这种情况下，器件可能容易受到更多的“蛮力破解”型密码猜测攻击。

SLAA685 — http://www-s.ti.com/sc/techlit/SLAA685
版权 © 2015, Texas Instruments Incorporated

表 9. MSP430FR5xx/FR6xx 器件上的 BSL 签名功能

名称	地址	值	器件安全
BSL 密码	FFE0h-FFFFh	用户定义 + 矢量表配置	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。已禁用“BSL 密码错误时批量擦除”功能。
		任何其他值	BSL 受密码保护。BSL 密码错误时执行大量擦除操作。

表 10. MSP430F1xx/F2xx/F4xx 器件上的 BSL 签名功能

名称	地址	值	器件安全
BSL 密码	FFE0h-FFFFh	用户定义 + 矢量表配置	必须首先由 BSL 主机提供该密码，BSL 才能访问器件。
BSL 签名	IVT 下方的数据字 [(1)]	AA55h	BSL 禁用。
		0000h	BSL 受密码保护。已禁用“BSL 密码错误时批量擦除”功能。
		任何其他值	BSL 受密码保护。BSL 密码错误时执行大量擦除操作。

(1) IVT 的起点因器件而异。请参阅器件特定数据表的“存储器组织”部分。例如，MSP430F2131 IVT 始于 FFE0h。因此，此器件上的 BSL 签名位于 FFDEh-FFDFh。

4.3 禁用引导加载程序 (BSL)

如果应用程序中未使用引导加载程序，或者担心未经授权的用户无视上述密码功能通过 BSL 获得访问权限，则也可以彻底禁用 BSL。在具有基于闪存的 BSL 的器件上，可以在生产时对器件进行编程，以擦除闪存的 BSL 区域。在具有基于 ROM 的 BSL 的器件上，可以在器件主存储器中设置一个 BSL 签名以彻底禁用 BSL，如表 9 和表 10 中所示。更多信息，请参阅相应的 BSL 用户指南：

MSP430 Programming With the Bootloader (BSL) (SLAU319)

《MSP430FR57xx、FR58xx、FR59xx、FR68xx 和 FR69xx 引导加载程序 (BSL) 用户指南》 (SLAU550)

注: 如果彻底禁用了引导加载程序，并且禁用了 JTAG/SBW 访问，则不仅无法执行现场固件更新或修补，当发生内存损坏等问题时也无法通过任何方式加载新代码以恢复器件。这还意味着无法在现场诊断所遇到的任何问题，因为无法读取或加载用于执行诊断测试的新代码。当决定禁用 BSL 而不是利用密码保护和其他 特性 时，需要认真考虑这一点 - 它在特定情况下的优势是否超过无法现场更新或诊断代码问题而面临的风险。

SLAA685 — http://www-s.ti.com/sc/techlit/SLAA685
版权 © 2015, Texas Instruments Incorporated

5 参考文献

1. *MSP430 FRAM Technology – How To and Best Practices* (SLAA628)
2. *Code Composer Studio v6.1 for MSP430 User's Guide* (SLAU157)
3. *MSP430 Optimizing C/C++ Compiler v4.4 User's Guide* (SLAU132)
4. *MSP430 Programming Via the JTAG Interface* (SLAU320)
5. *MSP430 Programming With the Bootloader (BSL)* (SLAU319)
6. *MSP430FR57xx, MSP430FR58xx, MSP430FR59xx, MSP430FR68xx, and MSP430FR69xx Bootstrap Loader (BSL)* (SLAU550)
7. *MSP430FR58xx, MSP430FR59xx, MSP430FR68xx, and MSP430FR69xx Family User's Guide* (SLAU367)
8. *MSP430x5xx and MSP430x6xx Family User's Guide* (SLAU208)
9. *MSP430x2xx Family User's Guide* (SLAU144)
10. *MSP430i2xx Family User's Guide* (SLAU335)

版权 © 2015, Texas Instruments Incorporated

重要声明和免责声明

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 产品的应用。严禁对这些资源进行其他复制或展示。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
Copyright © 2022，德州仪器 (TI) 公司