User’s Guide

适用于 MSP430™ MCU 的 Code Composer Studio（代码调试器）™ IDE v10.x

摘要

本手册介绍了 TI Code Composer Studio™ IDE v10.x (CCS v10.x) 与 MSP430™ 超低功耗微控制器的结合使用。该文档仅适用于 Code Composer Studio IDE 的 Windows 版本。Linux® 版本与此相似，因此不单独说明。

内容

1 请先阅读 4
- 1.1 如何使用本手册 4
- 1.2 有关注意事项和警告的信息 4
- 1.3 德州仪器 (TI) 提供的相关文档 5
- 1.4 如果您需要协助 5

2 开始着手！ 6
- 2.1 软件安装 6
- 2.2 LED 闪烁 6
- 2.3 重要 MSP430™ 文档 7

3 开发流程 8
- 3.1 使用 Code Composer Studio(代码调试器)™ IDE (CCS) 8
- 3.2 使用集成调试器 10

4 EnergyTrace™ 技术 17
- 4.1 引言 17
- 4.2 电能测量 17
- 4.3 Code Composer Studio™ 集成 17
- 4.4 EnergyTrace 技术常见问题解答 32

5 MSP430 FRAM 存储器保护机制 35
- 5.1 存储器保护单元 (MPU) 35
- 5.2 知识产权封装 (IPE) 36
- 5.3 FRAM 写入保护 (FRWP) 38

6 常见问题和解答 39
- 6.1 硬件 39
- 6.2 程序开发（汇编器、C 语言编译器、链接器、IDE） 39
- 6.3 调试 40

7 将 C 代码从 IAR 2.x、3.x、4.x、5.x、6.x 或 7.x 迁移到 CCS 43
- 7.1 中断矢量定义 43
- 7.2 内在功能 43
- 7.3 数据和函数放置 43
- 7.4 数据放置到命名的段中。 43
- 7.5 函数放置到命名段中 44
- 7.6 C 调用约定 45
- 7.7 其它差异 45

8 将汇编器代码从 IAR 2.x、3.x、4.x、5.x、6.x 或 7.x 迁移到 CCS 48
- 8.1 与汇编源代码共享 C/C++ 头文件 48
- 8.2 段控制 48
- 8.3 将 A430 汇编程序指令转化为 Asm430 指令 49

9 为 CCS 编写 可移植 C 语言代码和为 MSP430 编写 MSP430-GCC 56
- 9.1 中断矢量定义 56

10 FET 专用菜单.....57
10.1 菜单.....57
11 器件专用菜单.....58
11.1 MSP430L092.....58
11.2 MSP430F5xx 和 MSP430F6xx BSL 支持.....61
11.3 MSP430FR5xx 和 MSP430FR6xx 密码保护.....62
11.4 MSP430 超低功耗 LPMx.5 模式.....63
12 修订历史记录.....65

插图清单

图 3-1. 断点.....14
图 3-2. 断点属性.....14
图 3-3. 下载选项.....16
图 4-1. 脉冲密度和电流流量.....17
图 4-2. 工具栏菜单中的 EnergyTrace 按钮.....18
图 4-3. 退出 EnergyTrace 模式.....18
图 4-4. EnergyTrace™ Technology 首选项.....19
图 4-5. 项目属性.....20
图 4-6. 调试属性.....21
图 4-7. 电池选择.....22
图 4-8. 自定义电池类型.....22
图 4-9. 目标连接.....22
图 4-10. “EnergyTrace™ Technology”控制栏.....23
图 4-11. 使用 EnergyTrace++ 图表的调试会话.....24
图 4-12. “Profile”（**配置**）窗口.....25
图 4-13. “States”（**状态**）窗口.....26
图 4-14. “Power”（**电源**）窗口.....26
图 4-15. “Energy”（**电能**）窗口.....27
图 4-16. 使用 EnergyTrace 图表的调试会话.....28
图 4-17. EnergyTrace 系统配置窗口.....29
图 4-18. 放大电源窗口.....29
图 4-19. 放大电能窗口.....29
图 4-20. 同一程序在恢复（黄线）和自由运行（绿线）中的电能配置.....30
图 4-21. 在 EnergyTrace++ 模式下比较配置.....31
图 4-22. 在 EnergyTrace 模式下比较配置.....32
图 5-1. MPU 配置对话框.....35
图 5-2. IPE 配置对话框.....36
图 5-3. IPE 调试设置.....37
图 5-4. FRWP 配置对话框.....38
图 11-1. MSP430L092 模式.....58
图 11-2. C092 仿真模式下的 MSP430L092。.....59
图 11-3. MSP430C092 密码访问.....60
图 11-4. 允许对 BSL 的访问.....61
图 11-5. MSP430 密码访问.....62
图 11-6. 启用超低功耗调试模式.....63

表格清单

表 2-1. 系统要求.....6
表 3-1. 器件架构、断点和其他仿真特性.....11
表 4-1. EnergyTrace 和 EnergyTrace++ 模式的可用性.....18
表 4-2. “EnergyTrace™ Technology”控制栏图标.....23

商标

所有商标均为其各自所有者的财产。

1 请先阅读

1.1 如何使用本手册

阅读并遵循节 2 中的指示。本节包含软件安装说明，并介绍了如何运行演示程序。在了解了如何快速轻松使用开发工具后，TI 建议您通读本手册。

这本手册只描述了软件开发环境的设置和基本操作，并没有全面介绍 MSP430 微控制器或者完整的开发软件和硬件系统。有关这些项目的详细信息，请参阅节 2.3 *网络上的重要 MSP430 文档* 和节 1.3 中列出的相应 TI 文档。

本手册适用于将 CCS 与 TI MSP-FET、MSP-FET430UIF、eZ-FET 和 eZ430 开发工具系列搭配使用的情形。

这些工具包含封装时可用的最新材料。有关最新资料（包括数据表、用户指南、软件、应用信息等），请访问 TI MSP430 网站 www.ti.com/msp430 或者联系当地的 TI 销售办事处。

1.2 有关注意事项和警告的信息

本文档可能包含注意事项和警告。

CAUTION

这是注意事项声明示例。

注意事项申明描述了一种可能对软件或者器件造成潜在损坏的情况。

WARNING

这是警告声明示例。

警告申明描述了一种可能对您造成伤害的情况。

注意事项或者警告中所提供的信息是为了保护您的安全。请仔细阅读每一条注意事项和警告。

1.3 德州仪器 (TI) 提供的相关文档

CCS 文档

MSP430™ 汇编语言工具用户指南

MSP430™ 优化 C/C++ 编译器用户指南

MSP430 开发工具文档：

MSP 调试器用户指南

MSP430™ 硬件工具用户指南

eZ430-F2013 开发工具用户指南

eZ430-RF2480 用户指南

eZ430-RF2500 开发工具用户指南

eZ430-RF2500-SEH 开发工具用户指南

eZ430-Chronos™ 开发工具用户指南

MSP-EXP430G2 LaunchPad™ 实验板用户指南

使用增强型仿真模块 (EEM) 与 Code Composer Studio IDE 进行高级调试

MSP430 器件系列用户指南

MSP430F1xx 系列用户指南

MSP430F2xx 系列用户指南

MSP430F3xx 系列用户指南

MSP430F4xx 系列用户指南

MSP430F5xx 和 MSP430x6xx 系列用户指南

《MSP430FR4xx 和 MSP430FR2xx 系列用户指南》

《MSP430FR57xx 系列用户指南》

《MSP430FR58xx、MSP430FR59xx 和 MSP430FR6xx 系列用户指南》

CC430 器件系列用户指南

CC430 系列产品用户指南

1.4 如果您需要协助

TI 产品信息中心 (PIC) 提供了对 MSP430 微控制器和 FET 开发工具的支持。PIC 的联系信息可在 TI 网站 www.ti.com.cn/support 上找到。Code Composer Studio （代码调试器）专用 Wiki 页面 (FAQ)可用，而 MSP430 微控制器和 Code Composer Studio（代码调试器） IDE 的 TI E2E™ 支持论坛提供了一个与同行工程师、TI 工程师和其他专家互动的平台。其他器件专用信息可在 MSP430 网站上找到。

2 开始着手！

本节提供了软件安装说明，并说明了如何运行演示程序。

2.1 软件安装

要安装 Code Composer Studio（代码调试器）™ IDE (CCS)，请将相应的 CCS 软件包下载到 HostOS 平台上，并解压完整的压缩包，然后再运行 ccs_setup_x.x.x.x。用户可以选择使用离线或在线安装程序（如果连接速度缓慢和不稳定，TI 建议使用离线安装程序）。请按照屏幕显示的指令操作。安装 CCS 时，自动安装 USB JTAG 仿真器（MSP-FET、MSPFET430UIF、eZ-FET 和 eZ430 系列）的硬件驱动程序。此版本的 CCS 不再支持并口 FET (MSP-FET430PIF) 的传统调试接口。

备注

此版本的 CCS 不再支持传统的 MSP-FET430PIF（并口仿真器）。

备注

完全解压压缩包 (CCSx.x.x.x_y.zip)，然后再运行 ccs_setup_x.x.x.x。

备注

如果 MSP-FET 或 eZ-FET 调试器驱动程序安装失败：

在一些条件下（取决于所使用的硬件和操作系统），MSP-FET 或 eZ-FET 驱动程序安装可能会在首次安装时失败。这可能会导致 IDE 上出现无响应行为。要解决此问题，请断开 MSP-FET 或 eZ-FET，再重新插接或者将其插入其他的 USB 端口，然后重新启动 IDE。

表 2-1. 系统要求

	建议系统要求	最低系统要求
处理器	双核 x86 兼容处理器	1.0GHz x86 兼容处理器
RAM	6GB	2GB
可用磁盘空间	平均 2GB（1 或 2 个器件系列）；所有特性都为 3.5GB	900MB（取决于安装期间所选择的特性）
操作系统	• Windows®：Windows 7（SP1 或更高版本）、Windows 8.x 和 Windows 10 • Linux：关于受支持 Linux 分发的详细信息可在以下位置找到：http://software-dl.ti.com/ccs/esd/documents/ccsv10_linux_host_support.html • Mac：CCS 发布时支持当前最新版本及之前的版本。	

2.2 LED 闪烁

本节在 FET 程序上演示了相当于 C 语言中“Hello world!”的入门程序。CCS 包括可使 LED 立即闪烁的 C 语言代码模板文件。着手实施时，请执行以下步骤：

1. 通过点击 **Start → All Programs → Texas Instruments → Code Composer Studio → Code Composer Studio，启动 Code Composer Studio**。
2. 通过点击 **File → New → CCS Project**，创建新项目。
3. 输入项目名称。
4. 将器件系列 (Device Family) 设定为 MSP430，并选择要使用的器件变型 (Device Variant)（例如，MSP430F2274）。
5. 在 "Project template and example"（项目模板和示例）部分选择 "Blink The LED"（使 LED 闪烁）。
6. 单击完成。

备注

预定义示例适用于大多数 MSP430 板。为 MSP430G221x，MSP430L092 和 MSP430FR59xx 器件自动选择特定示例。某些 MSP430F4xx 板使用端口 P5.0 进行 LED 连接，必须在代码中进行手工更改。

7. 要编译代码并将应用程序下载至目标器件，请点击 **Run（运行）→ Debug（调试）(F11)**。

CAUTION

永远不要在活动调试会话期间断开 JTAG 或仿真器 USB 电缆。在断开目标器件前，请始终正确地终止正在运行的调试会话（通过点击“Terminate（终止）”图标）。

8. 要启动应用程序，请点击 **Run（运行）→ Resume（恢复）(F8)** 或点击工具栏上的 Play（播放）按钮。

如果 CCS 调试器无法与器件通信，请参阅常见问题解答 1。

恭喜，您刚刚构建并测试了一个 MSP430 应用程序！

2.3 重要 MSP430™ 文档

MSP430 和 CCS 信息的主要来源是器件专用数据表和用户指南。MSP430 网站 (www.ti.com/msp430) 包含这些文档的最新版本。

描述 Code Composer Studio（代码调试器）工具（IDE、汇编器、C 语言编译器、链接器和库）的文档可以在 https://www.ti.com.cn/tool/cn/CCSTUDIO 中找到。代码调试器专用 Wiki 页面 (FAQ)在 processors.wiki.ti.com/index.php/Category:CCS 上可用，而 e2e.ti.com 上的 TI E2E 支持论坛提供了额外的帮助。第三方工具的文档，例如 MSP430 的 IAR 嵌入式工作台，通常可以在各自的第三方网站上找到。

3 开发流程

本节介绍了如何使用 Code Composer Studio(代码调试器)(CCS) 来开发应用软件以及如何调试该软件。

3.1 使用 Code Composer Studio(代码调试器)™ IDE (CCS)

以下部分概括了如何使用 CCS。有关使用汇编语言或者 C 语言中的 CCS 进行软件开发流程的完整说明，请参阅 *MSP430 汇编语言工具用户指南* 和 *MSP430 优化 C/C++ 编译器用户指南*。

3.1.1 从零开始创建项目

本节逐步说明了如何从零开始创建汇编语言项目或者 C 语言项目，以及如何在 MSP430 上下载和运行应用程序（请参阅节 3.1.2）。此外，MSP430 Code Composer Studio（代码调试器）帮助对该过程进行了更全面的概括。

1. 启动 CCS (**Start → All Programs → Texas Instruments → Code Composer Studio → Code Composer Studio**)。
2. 创建新项目 (**File → New → CCS Project**)。输入项目名称，点击下一步并将器件系统设置为 MSP430 。
3. 选择适当的器件变量。对于仅汇编项目，请在 **Project template and examples** （项目模板和示例）部分中选择的 **Empty Assembly-only Project**。
4. 如果使用 USB 闪存仿真工具，例如 MSP-FET、MSP-FET430UIF、eZ-FET 或者 eZ430 开发工具，这些工具应该都已默认配置好。
5. 对于 C 语言项目，至此设置已经完成，显示 main.c 将被显示,并可输入代码。对于仅汇编项目，显示 main.asm。相反，如果想要在项目中使用现有的源文件，请点击 **Project → Add Files...** 并浏览查找所需文件。单击此文件并单击打开 (Open) 或者双击文件名来将此文件添加到项目文件夹。
6. 单击完成 (Finish)。
7. 将程序文本输入到文件中。

备注

使用 MSP430 标头文件（*.h 文件）来简化代码开发。

CCS 为每个器件提供了定义器件寄存器和位名的文件。TI 建议使用这些文件，这些可以显著简化程序开发任务。如需包含与目标器件对应的 .h 文件，请在 C 语言中添加 **#include <msp430xyyy.h>** 一行或在汇编代码中添加 **.cdecls C,LIST,"msp430xyyy.h"** 一行，其中 xyyy 指定了 MSP430 器件型号。

8. 构建项目 (**Project → Build Project**)。
9. 调试应用程序 (**运行 → 调试 (F11)**)。这样将启动调试器，从而获得对器件的控制，擦除目标内存，使用应用程序对目标内存进行编辑，并复位目标器件。

   如果调试器无法与器件通信，请参阅常见问题解答 1。
10. 点击 **Run → Resume (F8)** 以启动应用程序。
11. 点击 **Run → Terminate** 以停止应用程序并退出调试器。CCS 自动返回至 C/C++ 视图（代码编辑器）。

CAUTION

在活动的调试会话期间,永远不要断开 JTAG 或仿真器 USB 电缆。在断开目标器件前，请始终正确地终止正在运行的调试会话（通过点击“Terminate”图标）。

12. 点击 **File → Exit** 以退出 CCS。

3.1.2 项目设置

配置 CCS 所需的设置非常多且十分详细。大多数项目都可以使用默认的出厂设置来进行编译和调试。对于活动项目，通过点击 **Project → Properties** 可访问项目设置。建议使用或要求使用以下项目设置：

- 为调试会话指定目标器件 (**Project → Properties → General → Device → Variant**)。自动选择相应的链接器命令文件和运行时间支持库。
- 为了更轻松地调试 C 项目，请禁用优化 (**Project → Properties → Build → MSP430 Compiler → Optimization → Optimization level**)。
- 为 C 预处理器指定搜索路径 (**Project → Properties → Build → MSP430 Compiler → Include Options**)。
- 为任一正在使用的库指定搜索路径 (**Project → Properties → Build → MSP430 Compiler → File Search Path**)。
- 指定调试程序接口 (**Project → Properties → General → Device → Connection**)。为并行 FET 接口选择 TI MSP430 LPTx 或者为 USB 接口选择 TI MSP430 USBx。
- 在项目代码下载之前，启用擦除主内存和信息内存 (**Project → Properties → Debug → MSP430 Properties → Download Options → Erase Main and Information Memory**)。
- 为了确保正确的独立运行，请选择 Hardware Breakpoints”(**Project → Properties → Debug → MSP430 Properties**)。如果启用了“Software Breakpoints”（软件断点）(**Project → Properties → Debug → Misc/ Other Options → Allow software breakpoints to be used**)，请确保在目标连接时正确地终止每个调试会话；否则，由于器件上的应用程序仍包含有软件断点指令，因此目标可能无法独立运行。

3.1.3 在 CCS v5.5 和更新的版本中使用 MSP430 的数学函数库 (MSPMathlib)。

TI 的 MSPMathlib 是 CCSv5.5 和更新版本中的一部分。经过优化的此库在使用浮点标量数学运算中将性能提升了 26 倍多。有关详细信息，请参阅 MSPMathlib 网页 (http://www.ti.com.cn/tool/cn/mspmathlib)。

对于所有受支持的器件上的所有新项目，CCSv5.5 以上版本中默认激活 MSPMathlib。对于导入的项目，只有当此项目已经使用 MSPMathlib 或者它已经手动启用时才使用。

要禁用 MSPMathlib，请按照以下步骤操作：在 **Project → Properties → Build → MSP430 Linker → File Search Path** 下，删除“Include library file or command file as input (--library, -l)”字段中的 libmath.a。

要启用 MSPMathlib，请按照以下步骤操作：在 **Project → Properties → Build → MSP430 Linker → File Search Path** 下的“Include library file or command file as input (--library, -l)”字段中，添加 libmath.a。重要提示：请将 libmath.a 放在此处可能列出的其他库之前。

3.1.4 使用现有的 CCE v2.x、CCE v3.x、CCS v4.x、CCS v5.x、CCS v6.x、CCS v7.x、CCS v8.x 或 CCS v9.x 项目

CCS v10.x 支持将版本 CCE v2.x、CCE v3.x、CCS v4.x、CCS v5.x、CCS v6.x、CCS v7.x、CCS v8.x 或 CCS v9.x 中创建的工作区和项目转换为 CCS v10.x 格式 (**File → Import → General → Existing Projects into Workspace → Next)**。浏览包含待导入的项目的遗留 CCE 或 CCS 工作区。导入向导列出了给定工作区内的所有项目。然后可以选择并转换具体项目。CCEv2 和 CCEv3 项目可能需要在导入后手动更改目标配置文件 (*.ccxml) 。

根据之前的 CGT 版本，CCS 有可能返回一个警告，表示导入的项目是由另一个版本的代码生成工具 (CGT) 构建的。

虽然对汇编项目的支持并未改变，但对 C 语言代码的标头文件进行了些许改动，以便提高与 IAR Embedded Workbench（嵌入式工作台）® IDE（中断矢量定义）的兼容性。仍旧对 CCE 2.x 中使用的定义进行指定，但是已经在所有头文件中添加了对于这些定义的注释。为了支持 CCE 2.x 代码，删除"Interrupt Vectors"（中断矢量部分中的每个 .h 文件末尾的 #define statements 之前的 "//"。

3.1.5 堆栈管理

可以通过项目选项 **Project → Properties → Build → MSP430 Linker → Basic Options → Set C System Stack Size**) 来配置保留的堆栈大小。堆栈大小被定义为从 RAM 最后的位置扩展 50 至 80 字节（也就是说，堆栈从 RAM 向下扩展 50 至 80 字节，具体扩展的字节数取决于所选器件的 RAM 大小）。

堆栈可能会因为尺寸太小或者应用程序错误而溢出。有关跟踪堆栈大小的方法，请参阅节 3.2.2.1。

3.1.6 如何生成二进制格式文件（TI-TXT 和 INTEL-HEX）

CCS 安装包含 hex430.exe 转换工具。其可以配置为生成 TI-TXT 格式的输出项目，以便与 MSP-GANG 一起使用，也能生成 INTEL-HEX 格式文件以用于 TI 厂家器件编程。该工具可在命令行中独立使用（位于 <安装根目录>\ccsv6\tools\compiler\ti-cgt-msp430_x.x.x\bin）或者在 CCS 内直接使用。如需在每次构建后自动生成文件，请使用 MSP430 Hex Utility 菜单 (**Project → Properties → Build → MSP430 Hex Utility**) 并选择用于生成二进制文件的选项。生成的文件存储在 <Workspace>\<Project>\Debug\directory 内。

3.2 使用集成调试器

有关 CCS 内的 FET 专用菜单的说明，请参阅节 10。

3.2.1 断点类型

调试程序断点机制使用有限数量的片载调试资源（具体为 N 个断点寄存器，请参阅表 3-1）。当设置 N 个或者更少的断点时，应用程序必须以全速（或者“实时”）运行。当设置超过 N 个断点并启用了“Use Software Breakpoints”时 (**Project → Properties → Debug → Misc/Other Options → Allow software breakpoints to be used**)，可以设定无限数量的软件断点，同时仍然满足实时约束。

备注

软件断点将中断地址处的指令替换为中断代码执行的调用。因此，设置软件断点时会有较小的延迟。此外，使用软件断点时始终要求正确终止每个调试会话；否则，由于器件上的应用程序仍然包含软件断点指令，应用程序可能无法独立运行。

同时支持地址（代码）和数据（值）断点。数据断点和范围断点均要求两个 MSP430 硬件断点。

表 3-1. 器件架构、断点和其他仿真特性

器件	MSP430 架构	4 线制 JTAG	2 线制 JTAG[1]	断点 (N)	范围断点	时钟控制	状态程序设置	跟踪缓冲器	LPMx.5 调试支持
CC430F512x	MSP430Xv2	✓	✓	3	✓	✓			
CC430F513x	MSP430Xv2	✓	✓	3	✓	✓			
CC430F514x	MSP430Xv2	✓	✓	3	✓	✓			
CC430F612x	MSP430Xv2	✓	✓	3	✓	✓			
CC430F613x	MSP430Xv2	✓	✓	3	✓	✓			
CC430F614x	MSP430Xv2	✓	✓	3	✓	✓			
MSP430AFE2xx	MSP430	✓	✓	2		✓			
MSP430BT5190	MSP430Xv2	✓	✓	8	✓	✓	✓	✓	
MSP430F11x1	MSP430	✓		2					
MSP430F11x2	MSP430	✓		2					
MSP430F12x	MSP430	✓		2					
MSP430F12x2	MSP430	✓		2					
MSP430F13x	MSP430	✓		3	✓				
MSP430F14x	MSP430	✓		3	✓				
MSP430F15x	MSP430	✓		8	✓	✓	✓	✓	
MSP430F161x	MSP430	✓		8	✓	✓	✓	✓	
MSP430F16x	MSP430	✓		8	✓	✓	✓	✓	
MSP430F20xx	MSP430	✓	✓	2		✓			
MSP430F21x1	MSP430	✓		2		✓			
MSP430F21x2	MSP430	✓	✓	2		✓			
MSP430F22x2	MSP430	✓	✓	2		✓			
MSP430F22x4	MSP430	✓	✓	2		✓			
MSP430F23x	MSP430	✓		3	✓	✓			
MSP430F23x0	MSP430	✓		2		✓			
MSP430F2410	MSP430	✓		3	✓	✓			
MSP430F241x	MSP430X	✓		8	✓	✓	✓	✓	
MSP430F24x	MSP430	✓		3	✓	✓			
MSP430F261x	MSP430X	✓		8	✓	✓	✓	✓	
MSP430F41x	MSP430	✓		2		✓			
MSP430F41x2	MSP430	✓	✓	2		✓			
MSP430F42x	MSP430	✓		2		✓			
MSP430F42x0	MSP430	✓		2		✓			
MSP430F43x	MSP430	✓		8	✓	✓	✓	✓	
MSP430F43x1	MSP430	✓		2		✓			
MSP430F44x	MSP430	✓		8	✓	✓	✓	✓	
MSP430F44x1	MSP430	✓		8	✓	✓	✓	✓	
MSP430F461x	MSP430X	✓		8	✓	✓	✓	✓	
MSP430F461x1	MSP430X	✓		8	✓	✓	✓	✓	
MSP430F471xx	MSP430X	✓		8	✓	✓	✓	✓	
MSP430F47x	MSP430	✓		2		✓			
MSP430F47x3	MSP430	✓		2		✓			
MSP430F47x4	MSP430	✓		2		✓			
MSP430F51x1	MSP430Xv2	✓	✓	3	✓	✓			
MSP430F51x2	MSP430Xv2	✓	✓	3	✓	✓			
MSP430F52xx	MSP430Xv2	✓	✓	3	✓	✓			
MSP430F530x	MSP430Xv2	✓	✓	3	✓	✓			

表 3-1. 器件架构、断点和其他仿真特性 (continued)

器件	MSP430 架构	4 线制 JTAG	2 线制 JTAG[(1)]	断点 (N)	范围断点	时钟控制	状态程序设置	跟踪缓冲器	LPMx.5 调试支持
MSP430F5310	MSP430Xv2	✓	✓	3	✓	✓			
MSP430F532x	MSP430Xv2	✓	✓	8	✓	✓	✓	✓	
MSP430F533x	MSP430Xv2	✓	✓	8	✓	✓	✓	✓	
MSP430F534x	MSP430Xv2	✓	✓	8	✓	✓	✓	✓	
MSP430F535x	MSP430Xv2	✓	✓	8	✓	✓	✓	✓	
MSP430F54xx	MSP430Xv2	✓	✓	8	✓	✓	✓	✓	
MSP430F54xxA	MSP430Xv2	✓	✓	8	✓	✓	✓	✓	
MSP430F550x	MSP430Xv2	✓	✓	3	✓	✓			
MSP430F5510	MSP430Xv2	✓	✓	3	✓	✓			
MSP430F552x	MSP430Xv2	✓	✓	8	✓	✓	✓	✓	
MSP430F563x	MSP430Xv2	✓	✓	8	✓	✓	✓	✓	
MSP430F565x	MSP430Xv2	✓	✓	8	✓	✓	✓	✓	
MSP430F643x	MSP430Xv2	✓	✓	8	✓	✓	✓	✓	
MSP430F645x	MSP430Xv2	✓	✓	8	✓	✓	✓	✓	
MSP430F663x	MSP430Xv2	✓	✓	8	✓	✓	✓	✓	
MSP430F665x	MSP430Xv2	✓	✓	8	✓	✓	✓	✓	
MSP430F67xx	MSP430Xv2	✓	✓	3	✓	✓			
MSP430F67xx1	MSP430Xv2	✓	✓	3	✓	✓			
MSP430F67xx1A	MSP430Xv2	✓	✓	3	✓	✓			
MSP430F67xxA	MSP430Xv2	✓	✓	3	✓	✓			
MSP430FE42x	MSP430	✓		2		✓			
MSP430FE42x2	MSP430	✓		2		✓			
MSP430FG42x0	MSP430	✓		2		✓			
MSP430FG43x	MSP430	✓		2		✓			
MSP430FG461x	MSP430X	✓		8	✓	✓	✓	✓	
MSP430FG47x	MSP430	✓		2		✓			
MSP430FG642x	MSP430Xv2	✓	✓	8	✓	✓	✓	✓	
MSP430FG662x	MSP430Xv2	✓	✓	8	✓	✓	✓	✓	
MSP430FR20xx	MSP430Xv2	✓	✓	3	✓	✓			
MSP430FR21xx	MSP430Xv2	✓	✓	3	✓	✓			
MSP430FR23xx	MSP430Xv2	✓	✓	3	✓	✓			
MSP430FR24xx	MSP430Xv2	✓	✓	3	✓	✓			
MSP430FR25xx	MSP430Xv2	✓	✓	3	✓	✓			
MSP430FR26xx	MSP430Xv2	✓	✓	3	✓	✓			
MSP430FR41xx	MSP430Xv2	✓	✓	3	✓	✓			
MSP430FR57xx	MSP430Xv2	✓	✓	3	✓	✓			
MSP430FR58xx	MSP430Xv2	✓	✓	3	✓	✓			✓
MSP430FR59xx	MSP430Xv2	✓	✓	3	✓	✓			✓
MSP430FR60xx	MSP430Xv2	✓	✓	3	✓	✓			✓
MSP430FR68xx	MSP430Xv2	✓	✓	3	✓	✓			✓
MSP430FR69xx	MSP430Xv2	✓	✓	3	✓	✓			✓
MSP430FW42x	MSP430	✓		2		✓			
MSP430G2xxx	MSP430	✓	✓	2		✓			
MSP430i20xx	MSP430	✓	✓	2		✓			
MSP430L092	MSP430Xv2	✓		2		✓			
MSP430SL54xxA	MSP430Xv2	✓	✓	8	✓	✓	✓	✓	

表 3-1. 器件架构、断点和其他仿真特性 (continued)

器件	MSP430 架构	4 线制 JTAG	2 线制 JTAG(1)	断点 (N)	范围断点	时钟控制	状态程序设置	跟踪缓冲器	LPMx.5 调试支持
MSP430TCH5E	MSP430	✓	✓	2		✓			
RF430FRL15xH	MSP430Xv2	✓	✓	2		✓			

(1) 此 2 线制 JTAG 调试接口也被称为 Spy-Bi-Wire (SBW) 接口。只有 USB 仿真器（eZ430-xxxx，eZ-FET 和 MSP-FET430UIF USB JTAG 仿真器）和 MSP-GANG430 生产编程工具支持该接口。

3.2.2 使用断点

如果调试程序启动时设置的断点数量超过 N 个并且软件断点被禁用（**Project → Properties → Debug → Misc/Other Options → Allow software breakpoints to be used 选项处于未选中状态**），则会显示一条消息，通知用户并非所有断点都可以启用。不管 CCS 的使用软件断点设置如何，CCS 都允许设置任何数量的断点。如果禁用了软件断点，则在调试程序中最多可设置 N 个断点。

程序复位需要断点，此断点设定在 **Project → Properties → Debug → Auto Run and Launch Options → Auto Run Options → Run to symbol** 中定义的地址上。

运行至光标 (Run to Cursor) 操作临时需要一个断点。

控制台 I/O (CIO) 功能，正如 printf，需要使用一个断点。如果希望这些功能都被编译，但又不想使用断点，请通过更改 **Project → Properties → Debug → Program/Memory Load Options → Program/Memory Load Options → Enable CIO function use** 中的选项（要求设置一个断点）来禁用 CIO 功能。

备注

如果上一个指令修改了堆栈指针，请勿在 RETI 指令中设置断点。到达断点后，程序将不能正常执行。

3.2.2.1 CCS 中的断点

CCS 支持许多预定义断点类型，可以通过打开“Breakpoint”（断点）窗口中的断点图标旁边的菜单来选择这些类型 (**Window → Show View → Breakpoints**)。除了传统的断点之外，CCS 允许设置观察点，以在数据地址访问而非地址访问上中断。通过右键点击断点并选择“Properties”（属性），可在调试器中更改断点和观察点的属性（请参阅图 3-1 和图 3-2）。

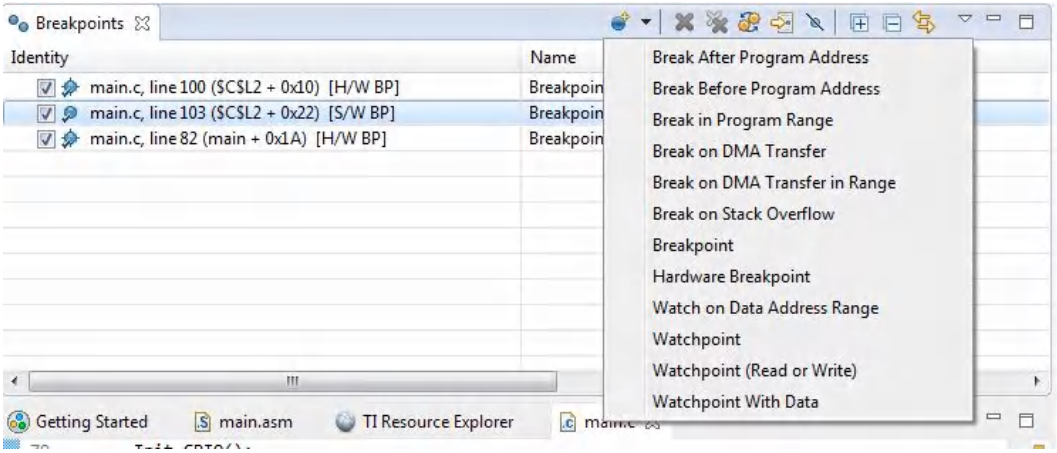

图 3-1. 断点

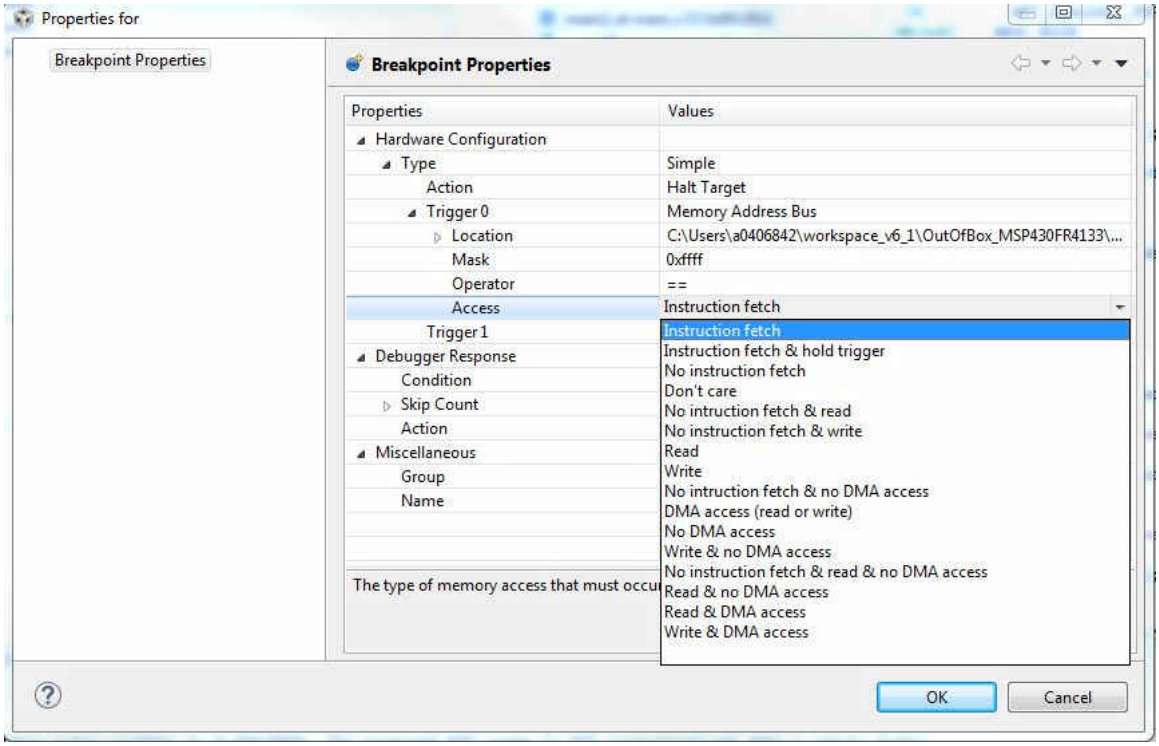

图 3-2. 断点属性

- **Break after program address**

  当程序尝试在特定地址之后执行代码时，停止代码执行。
- **Break before program address**

  当程序尝试在特定地址之前执行代码时，停止代码执行。
- **Break in program range**

  当程序尝试在特定范围内执行代码时，停止代码执行。
- **Break on DMA transfer**
- **Break on DMA transfer in range**

  当指定地址范围内的 DMA 访问发生时，就会中断。
- **Break on stack overflow**

  可以对引起堆栈溢出的应用程序进行调试。设置 Break on Stack Overflow（堆栈溢出中断）（右键点击“Breakpoints”窗口，然后在上下文菜单中选择“Break on Stack Overflow”）。在引起堆栈溢出的指令上停止执行程序。可在 Project → Properties → C/C++ Build → MSP430 Linker → Basic Options 中调整堆栈的大小。
- **Breakpoint**

  设定断点。
- **Hardware breakpoint**

  如果软件断点未被禁用，则强制使用硬件断点。
- **Watch on data address range**

  当对特定范围内的地址进行数据访问时，停止代码执行。
- **Watchpoint**

  如果对特定地址进行特定的数据访问，则停止代码执行。
- **Watchpoint (Read or Write)**

  如果对特定地址进行数据读取或写入访问，则停止代码执行。
- **Watchpoint with data**

  如果使用特定值对特定地址进行特定的数据访问,则停止代码执行。

  约束 1：观察点适用于全局变量和非寄存器局部变量。在后一种情况下，设置断点 (BP) 以在需要观察变量的函数中停止执行（在此处设置代码断点）。然后设置观察点并删除（或禁用）函数中的代码断点,然后运行或重新启动应用程序。

  约束 2：观察点适用于 8 位和 16 位宽的变量。

备注

并非所有选项都适用于每个 MSP430 衍生器件（请参阅表 3-1）。因此，断点菜单中预定义的断点类型的数量随所选器件的不同而不同。

有关使用 CCS 进行高级调试的更多信息，请参阅使用增强型仿真模块 (EEM) 与 Code Composer Studio IDE 进行高级调试。

3.2.3 用于 MSP430 器件的下载选项

默认情况下，在调试会话开始时，CCS 调试器会将应用程序下载到 RAM 或闪存上。通过“下载选项”（请参阅图 3-3），可以修改下载行为。

- **Copy application to external SPI memory after program load（程序加载后将应用复制到外部 SPI 存储器）**

  将用户代码保存到外部 SPI 存储器。
- **Allow Read/Write/Erase access to BSL memory（允许对 BSL 存储器进行读取/写入/擦除访问）**

  启用对 BSL 闪存的擦除和写入访问。
- **Erase main memory only（仅擦除主内存）**

  下载前仅擦除主闪存。信息存储器未被擦除。
- **Erase main and Information memory（擦除主内存和信息内存）**

  在下载前擦除主闪存和信息闪存。
- **Erase main, information and protected information memory（擦除主内存和、信息内存和受保护信息内存）**

  在下载前擦除主闪存和信息闪存，包括 IP 受保护区域。
- **Erase and download necessary segments only (Differential Download)（只擦除和下载必要的片段（差动下载））**

  跟踪程序映像的更改，并仅写入两次程序加载之间映像发生更改的部分。理论上，小幅更改应该能够提升负载性能。较大更改可能导致性能下降。其依赖于编译器和链接器来从根本上更改二进制映像，以实现对源代码的微小更改。
- **Replace written memory locations, retain unwritten memory locations（替换写入的内存位置，保留未写入的内存位置）**

  仅写入正在写入的闪存段。这不会跟踪已加载映像中的差异，并始终会写入包含在已加载映像中的片段。
- **By Address Range (specify below)（按地址范围（请在下述指明））**

  下载前仅擦除指定的闪存段。

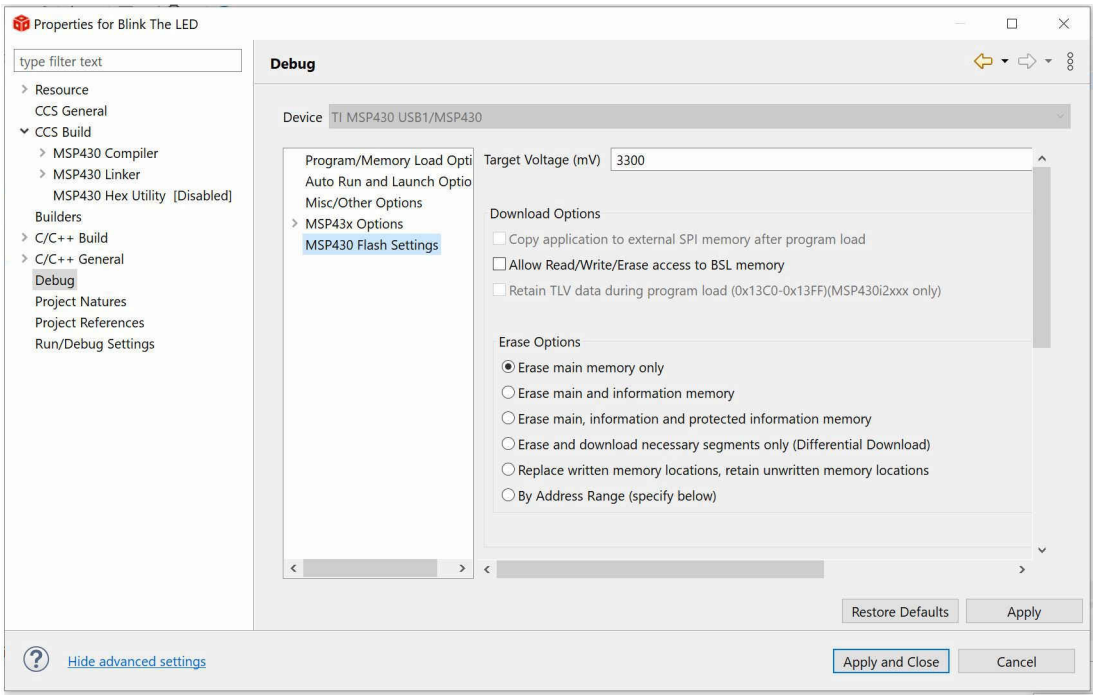

图 3-3. 下载选项

4 EnergyTrace™ 技术

4.1 引言

EnergyTrace™ 技术是基于电能的代码分析工具，用于测量和显示应用的电能曲线，并帮助优化应用以实现超低功耗。

具有内置 **EnergyTrace+[CPU 状态]+[外设状态]**（简称 **EnergyTrace++**）技术的 MSP430 器件可在用户程序代码执行期间实时监控多个内部器件状态。所选 MSP430 器件和调试器上支持 EnergyTrace++ 技术。

EnergyTrace 模式（不带“++”）是 **EnergyTrace 技术**的基础，其允许模拟电能测量来确定应用的能耗，但不将其与内部器件信息相关联。EnergyTrace 模式适用于所有带有所选调试器的 MSP430 器件（包含 CCS）。

4.2 电能测量

支持 EnergyTrace 技术支持的调试器包含一个全新且独特的方法来持续测量施加到目标微控制器的电能，该方法与众所周知的在不连续的时间对分流电阻器上的压降进行放大和采样的方法有很大不同。由软件控制的 dc-dc 转换器用于生成目标电源。DC-DC 转换器充电脉冲的时间密度等于目标微控制器的能耗。内置高速数据传输错误纠正校准电路定义与单个 DC-DC 充电脉冲等效的电能。

图 4-1 显示电能测量原理。每个时间单位内充电脉冲数量少的周期表示能耗低，因此电流低。每个时间单位内充电脉冲数量多的周期表示能耗高，因此电流高。每个电荷脉冲都会导致输出电压 $V_{输出}$上升，从而导致所有 DC-DC 转换器都不可避免的产生电压纹波。

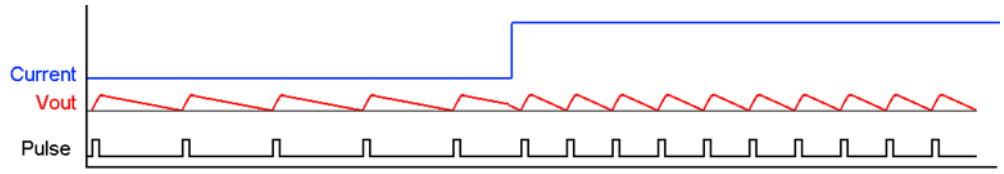

图 4-1. 脉冲密度和电流流量

持续采样的优势显而易见：即使是消耗电能的最短器件活动也会对总体记录能耗产生影响。任何基于分流器的测量系统都无法实现这一点。

4.3 Code Composer Studio™ 集成

EnergyTrace 技术是德州仪器 (TI) 的 Code Composer Studio（代码调试器）IDE 的一部分,用于 MSP430 微控制器。如果硬件支持 EnergyTrace 技术，则可以使用其他控件和窗口。

EnergyTrace 可以在调试应用程序（调试会话）期间使用，或仅用于测量独立运行的应用程序（在无调试会话的情况下）的电流消耗。在无调试会话的情况下使用 EnergyTrace 可以测量正在运行的应用程序的电流消耗，而无需更改代码内容或 CPU 状态。

在调试会话期间，可以使用 EnergyTrace 和 EnergyTrace++ 模式，具体取决于目标器件上支持的硬件特性。只有 EnergyTrace 模式可以用于独立运行的应用程序（请参阅表 4-1）。

表 4-1. EnergyTrace 和 EnergyTrace++ 模式的可用性

	EnergyTrace	EnergyTrace++
调试会话	x	x
独立应用程序	x	

在无调试会话的情况下使用 EnergyTrace 技术

1. 连接内嵌固件的目标板
2. 按下工具栏菜单中的“EnergyTrace Technology”按钮（请参阅图 4-2）。您无需构建或启动调试会话。

图 4-2. 工具栏菜单中的 EnergyTrace 按钮

3. 点击启动跟踪收集按钮 (▶) 以启动 EnergyTrace 技术测量。
4. 点击停止跟踪收集按钮 (●) 以停止 EnergyTrace 技术测量。
5. 有关更多详细信息，请参阅节 4.3.4。

如需退出此模式，请点击“EnergyTrace™ Technology”窗口中的 ⊠（请参阅图 4-3）。

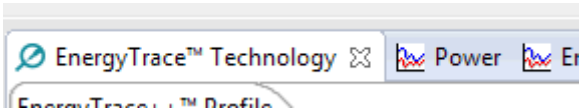

图 4-3. 退出 EnergyTrace 模式

为了在调试会话中使用 EnergyTrace 模式或 EnergyTrace++ 模式

1. 连接您的目标板。
2. 选择并构建您的项目
3. 开始调试会话。
4. 如果已选择 EnergyTrace，则显示 EnergyTrace 窗口。如果没有显示，请点击工具栏菜单中的 EnergyTrace 按钮。
5. 有关更多详细信息，请参阅节 4.3.3 和节 4.3.4。

4.3.1 EnergyTrace 技术设置

EnergyTrace 设置在“Code Composer Studio 的“Preferences”中。转到 **窗口 → 首选项 → 代码调试器 → 高级工具 → EnergyTrace™ 技术**（请参阅图 4-4）。

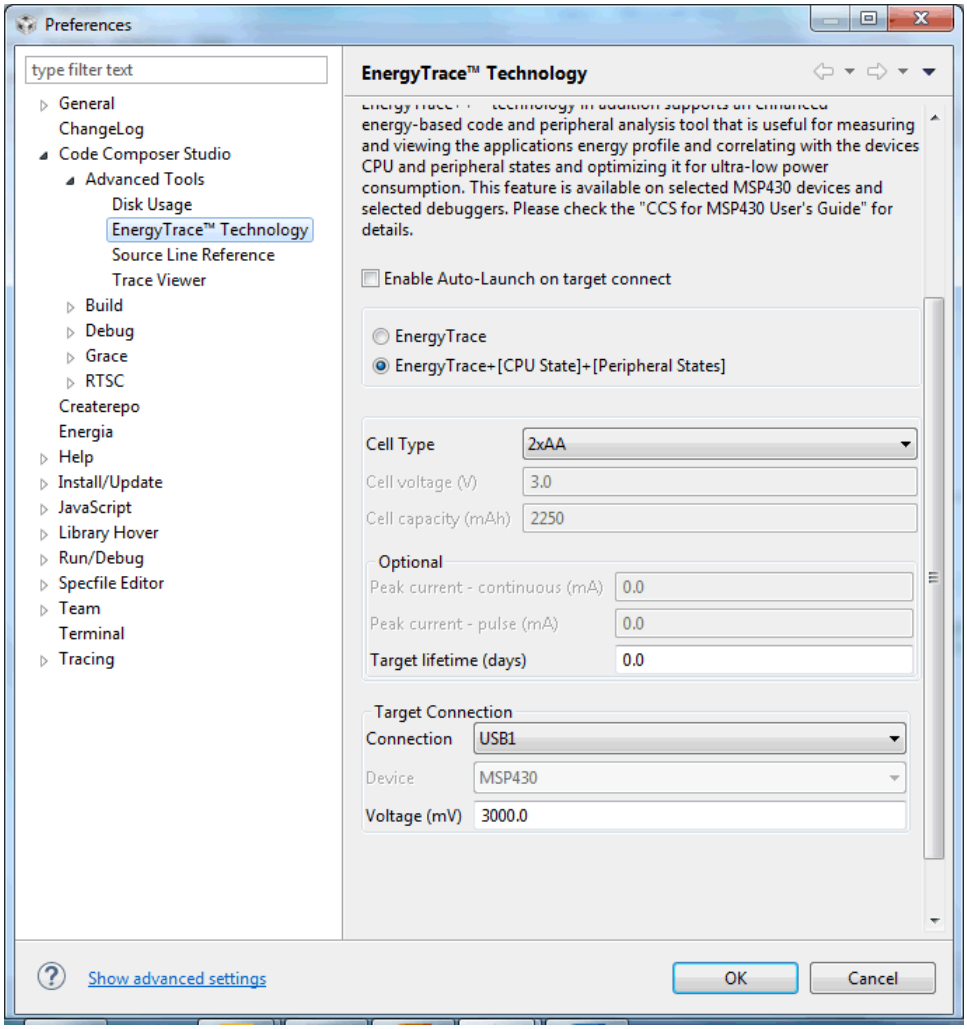

图 4-4. EnergyTrace™ Technology 首选项

- **Enable Auto-Launch on target connect**（**启用目标连接时的自动启动**）：选中此复选框可在进入调试会话时启用 EnergyTrace 模式。
- **支持以下两种采集模式**：
  - 全功能 **EnergyTrace+[CPU State]+[Peripheral States]**模式，提供实时器件状态信息以及电能测量数据
  - 只提供电能测量数据的 **EnergyTrace** 模式
  - 使用单选按钮选择启动调试会话时待启用的模式。如果 MSP430 器件不支持器件状态采集，则该选择将被忽略，Code Composer Studio（代码调试器）将以 EnergyTrace 模式启动。

当调试会话处于活动状态时，点击“Profile”窗口中的 图标可以在两种模式间切换。

如需使用 **EnergyTrace+[CPU State]+[Peripheral States]** 模式采集应用执行时的实时器件状态信息，还必须修改项目的默认调试属性。右键点击“Project Explorer”（项目浏览器）中的当前项目，然后点击“Properties”（属性）（请参阅 图 4-5）。

在调试 (Debug) 部分中，启用低功耗模式设置(Low Power Mode Settings) 中的**启用超低功耗调试/调试 LPMx.5 (Enable Ultra Low Power debug / Debug LPMx.5)**选项（请参阅图 4-6）。如果未启用该选项，则 **EnergyTrace+[CPU State]+[Peripheral States]**模式无法从器件采集数据。

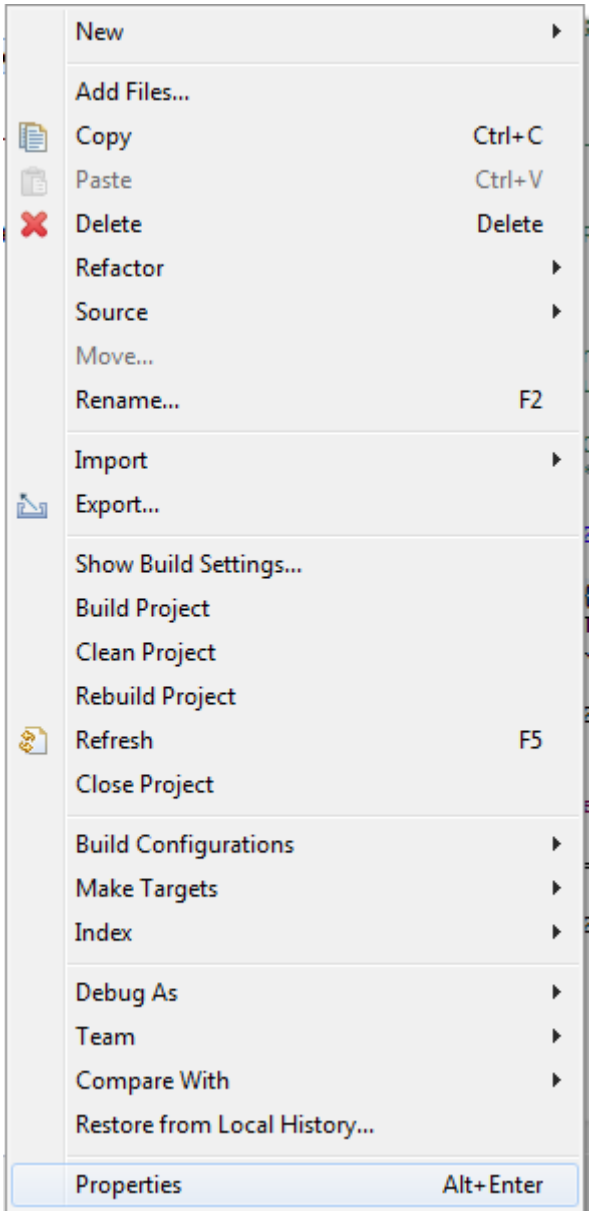

图 4-5. 项目属性

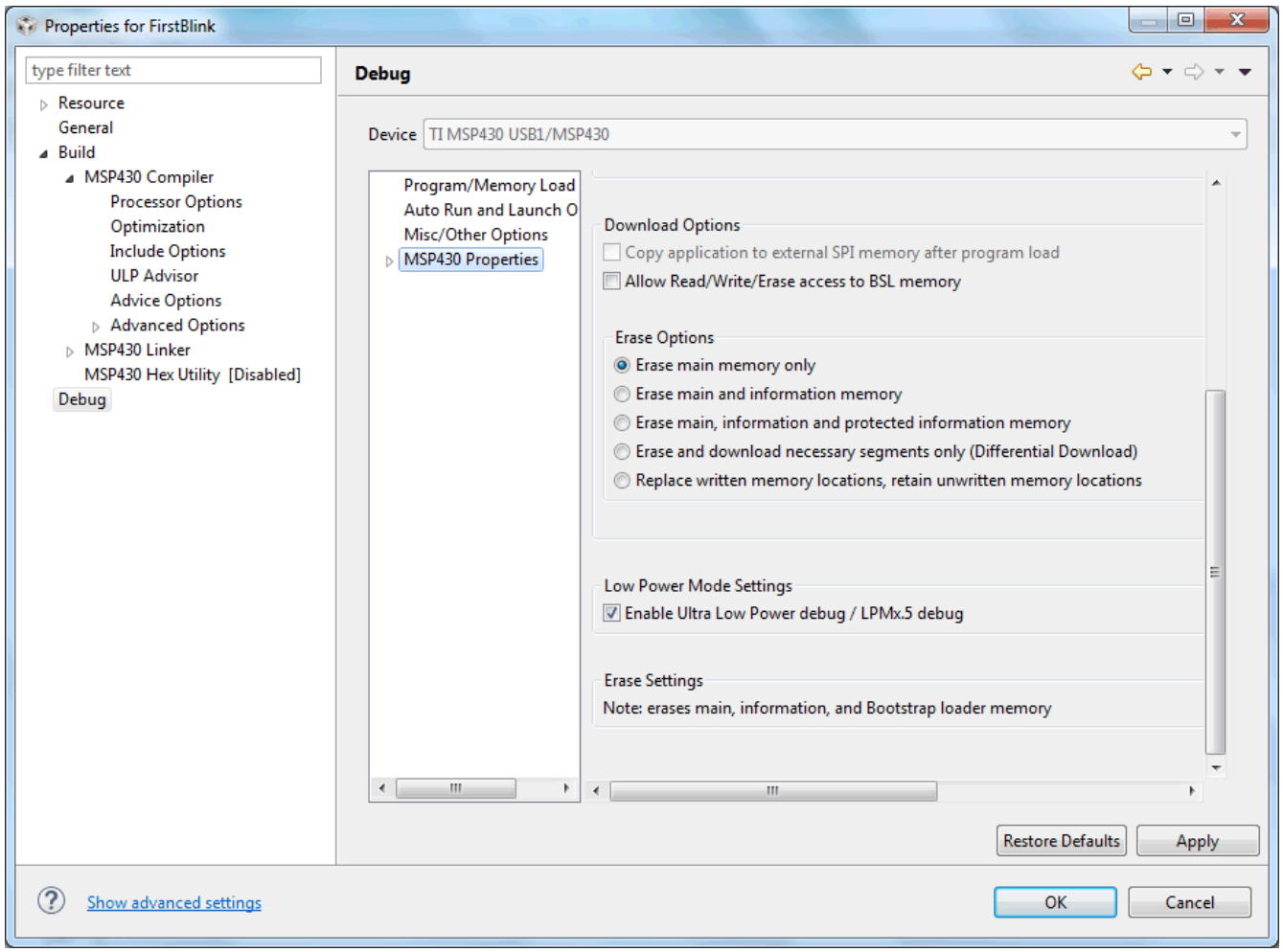

图 4-6. 调试属性

备注

如果在调试会话启动时未打开 EnergyTrace 技术窗口，请验证以下各项：

- 硬件（调试程序和器件）是否支持 EnergyTrace 技术？要确定所选器件是否支持 EnergyTrace 技术，请参阅器件专用数据表、*MSP430 硬件工具用户指南* 或评估板随附的用户指南。
- 是否在 **Window → Preferences → Code Composer Studio → Advanced Tools → EnergyTrace ™ Technology** 中全面启用了 EnergyTrace 技术？
- 是否在 **Project → Properties → Debug → Low Power Mode Settings** 中启用了“Enable Ultra Low Power debug / Debug LPMx.5”选项（仅在选择 EnergyTrace 模式时需要启用）？

- **电池选择**（请参阅图 4-7）：该窗口用于选择一个可用的标准电池或定义一个自定义电池。EnergyTrace 将使用电池特性来根据测得的电流消耗计算当前应用的预估选定电池寿命。可用的标准电池为 CR2032、2xAAA 或 2xAA。

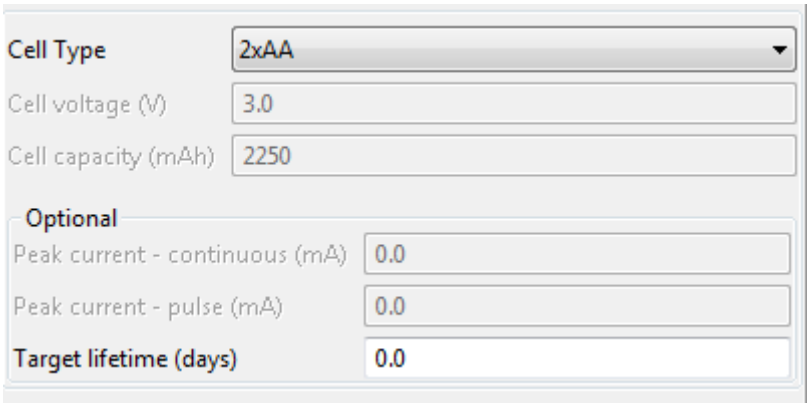

图 4-7. 电池选择

也可以选择自定义电池，并可以输入其特性（请参阅图 4-8）。

- 电池电压 (V)
- 电池容量 (mAh)
- 峰值电流 - 持续 (mA)
- 峰值电流 - 脉冲 (mA)
- 目标寿命（天）

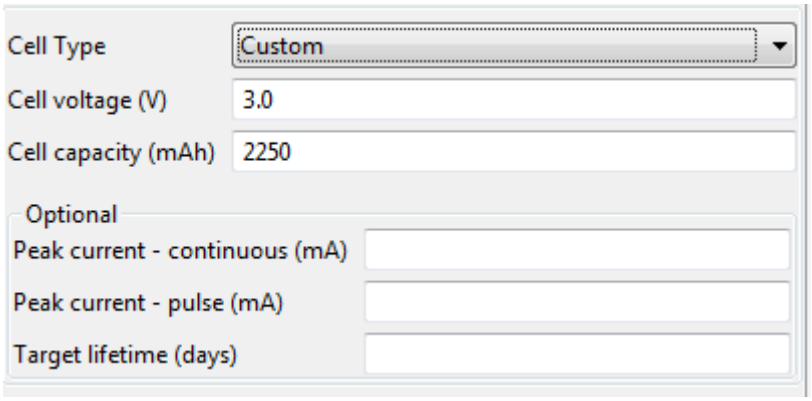

图 4-8. 自定义电池类型

- **目标连接**（请参阅图 4-9）：该菜单用于选择将哪个调试探针用于 EnergyTrace 测量。还可以调节电压。

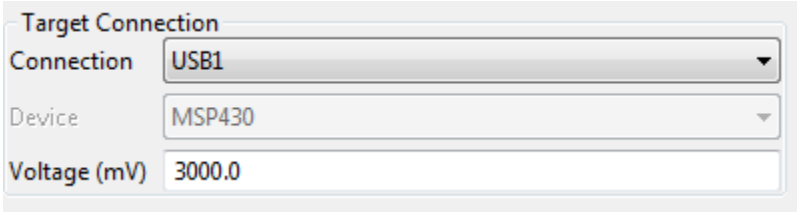

图 4-9. 目标连接

4.3.2 控制 EnergyTrace 技术

EnergyTrace 技术可以通过 **Profile （配置）**窗口中的控制栏图标进行控制（请参阅图 4-10）。表 4-2 介绍了各个按钮的功能。

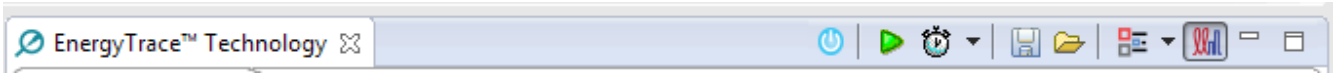

图 4-10. “EnergyTrace™ Technology”控制栏

表 4-2. “EnergyTrace™ Technology”控制栏图标

	启用或禁用 EnergyTrace 技术。当被禁用时，图标变成灰色。
	开始跟踪收集。
	停止跟踪收集。
	设置采集周期：5 秒、10 秒、30 秒、1 分钟或 5 分钟。经过该时间后，数据收集会停止。但是，在点击调试控制窗口中的“Pause”（暂停）按钮前，程序会继续执行。
	将配置保存至项目目录。保存 EnergyTrace++ 配置时，默认文件名以“MSP430_D”开头，后跟一个时间戳。保存 EnergyTrace 配置时，默认文件名以“MSP430”开头，后跟一个时间戳。
	加载入之前存入的配置进行比较。
	恢复曲线图或打开“Preferences”（首选项）窗口。
	在 **EnergyTrace++** 模式和 **EnergyTrace** 模式之间切换

4.3.3 EnergyTrace++ 模式

当调试具有内置 EnergyTrace++ 支持的器件时，**EnergyTrace++ 模式**提供有关目标微控制器的能耗和内部状态的信息。以下窗口会在调试会话启动时被打开（另请参阅图 4-11）：

- Profile
- States
- Power
- Energy

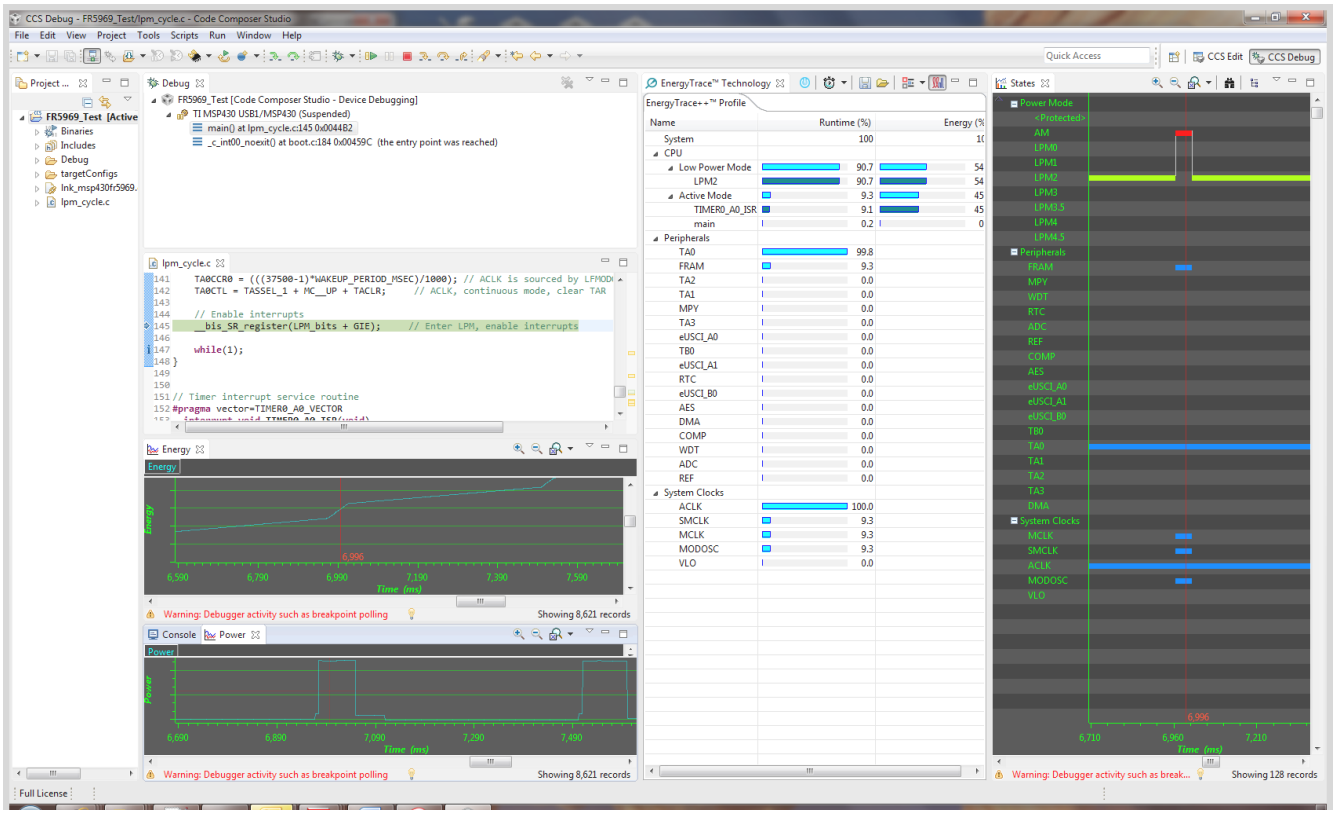

图 4-11. 使用 EnergyTrace++ 图表的调试会话

Profile（**配置**）窗口（请参阅图 4-12）是 EnergyTrace++ 的控制界面。其可用于设定采集时间或保存采集的数据供以后参考。**Profile**（**配置**）窗口还显示了采集数据的压缩视图，并允许与之前数据进行比较。

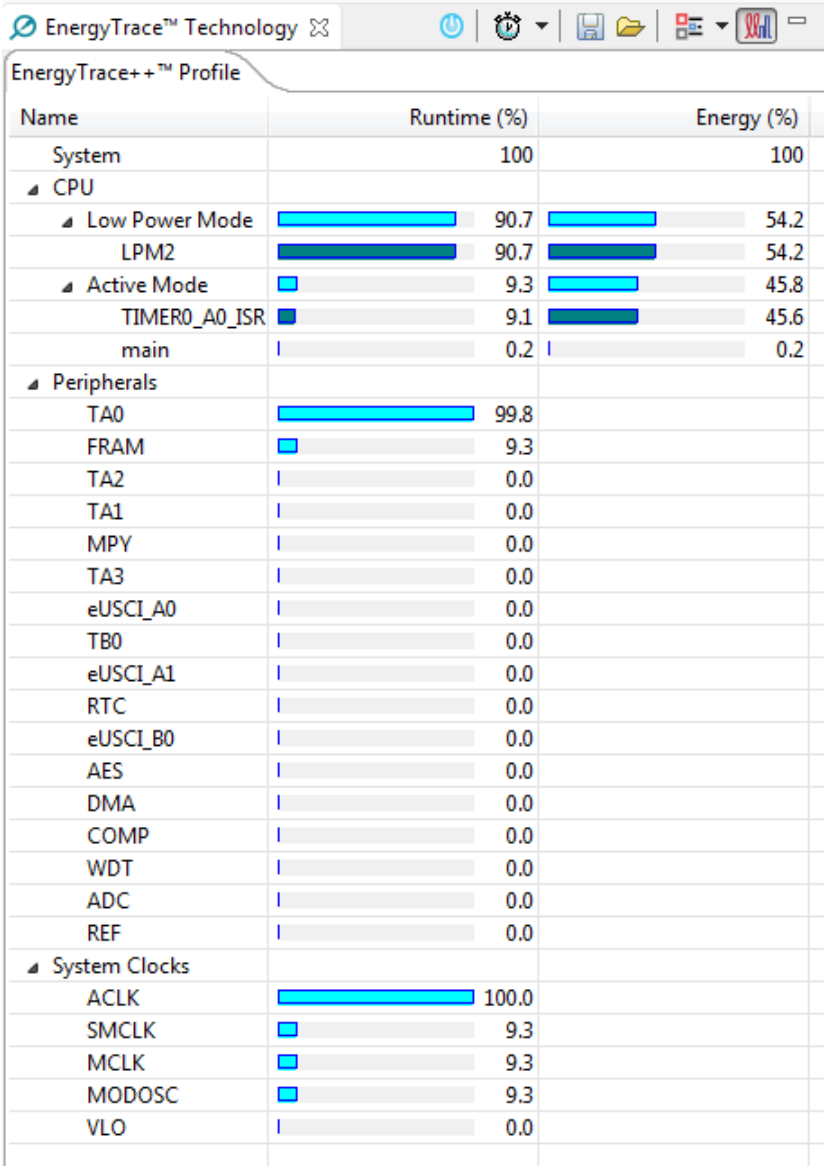

图 4-12. “Profile”（配置）窗口

Profile（**配置**）窗口允许快速浏览已配置应用的资源使用情况。这些资源被分为三类：

- CPU: 显示与程序执行相关的信息
  - 低功耗模式：显示低功耗模式使用摘要。有效的低功耗模式为 LPM0、LPM1、LPM2、LPM3、LPM4、LPM3.5 和 LPM4.5。如果无法正确确定低功耗模式，则会显示标记为“<Undetermined>”的行，以指示处于该模式下所花费的时间。
  - 活动模式：显示活动模式期间执行了哪些函数。在 _RTS_ subcategory 中分别列出运行时库中的函数。如果器件支持 IP 封装，将显示一条标记为“<Protected>”的行，指示在 IP 封装存储器外执行的时间。
- 外设：显示器件外设的相对时间
- 系统时钟：显示系统时钟的相对时间

States（**状态**）窗口（请参阅图 4-13）显示采集会话期间目标微控制器内部状态的实时跟踪。状态信息包括功率模式、外设模块的开关状态和系统时钟的状态。

图 4-13 显示了器件从低功耗模式 LPM2 唤醒到活动模式，其中活动期间启用了 FRAM 存储器。可以清楚地看到，器件高速时钟 MCLK 和 SMCLK 以及 MODOSC 仅在器件处于活动模式时才有效。**States**（**状态**）窗口允许直接验证应用是否表现出预期行为，例如，某个外设在特定活动后被禁用。

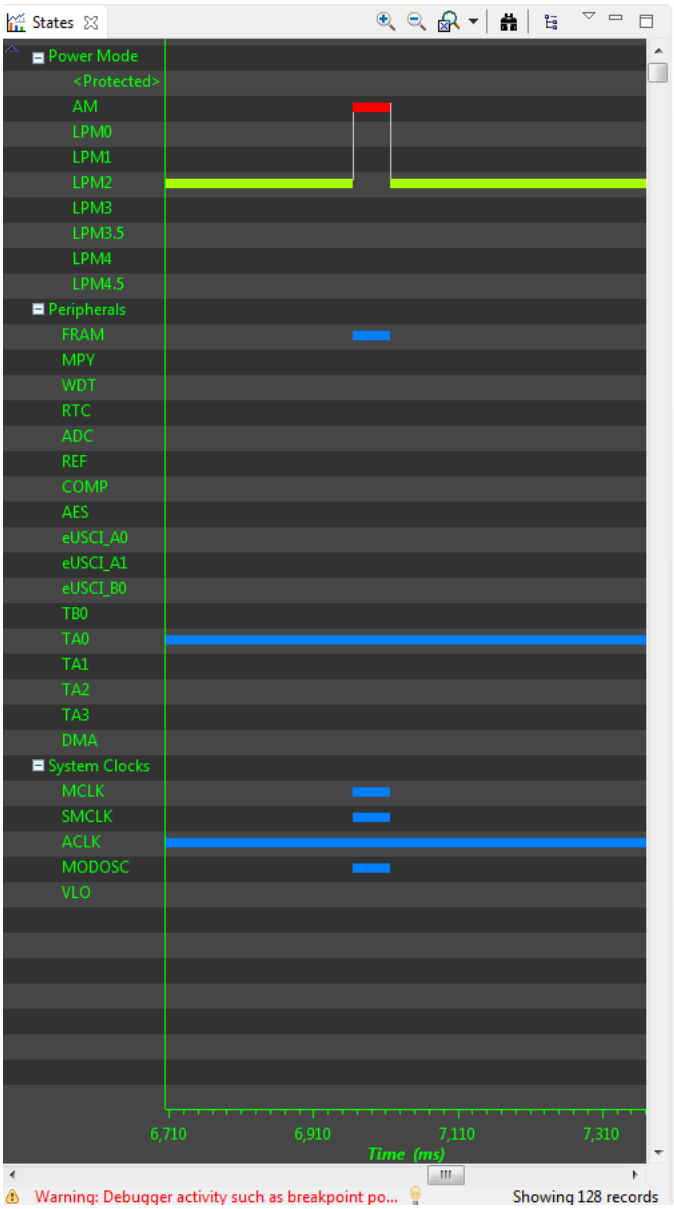

图 4-13. “States”（状态）窗口

Power（**电源**）窗口（请参阅图 4-14）显示目标随时间变化的动态功耗。当前配置用淡蓝色标出，而之前为进行比较而重新加载的配置用黄色标出。

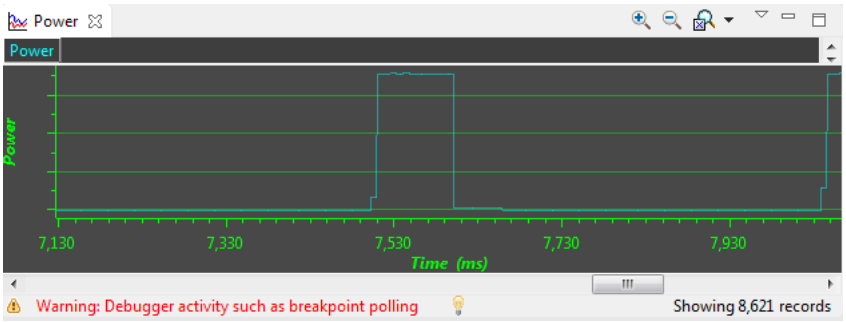

图 4-14. “Power”（电源）窗口

Energy（**电能**）窗口（请参阅图 4-15）显示目标随时间变化的累积能耗。当前配置用淡蓝色标出，而之前为进行比较而重新加载的配置用黄色标出。

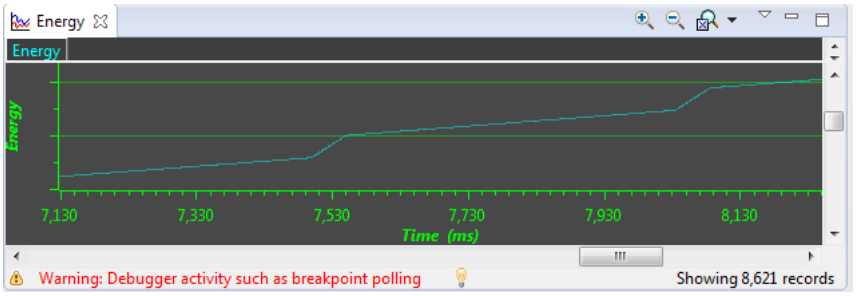

图 4-15. “Energy”（电能）窗口

备注

内部状态采集期间，通过 JTAG 或 Spy-Bi-Wire 调试逻辑持续访问目标微控制器。这些调试访问会消耗电能；因此，“Power”（电源）和“Energy”（电能）图的纵轴上未显示绝对功率值。要查看应用的绝对功率值，TI 建议结合使用 EnergyTrace 模式与 Free Run（自由运行）选项。在此模式下，测量能耗时不会访问目标微控制器的调试逻辑。

4.3.4 EnergyTrace 模式

此模式允许独立使用不支持内置 EnergyTrace++ 的 MSP430 微控制器的电能测量功能。其还可以在没有调试器活动的情况下验证应用的能耗。如果在首选项 (Preferences) 窗口中选择 **EnergyTrace** 模式，则在调试会话启动时将打开以下窗口（另请参阅图 4-16）：

- 配置
- 电源
- 电能

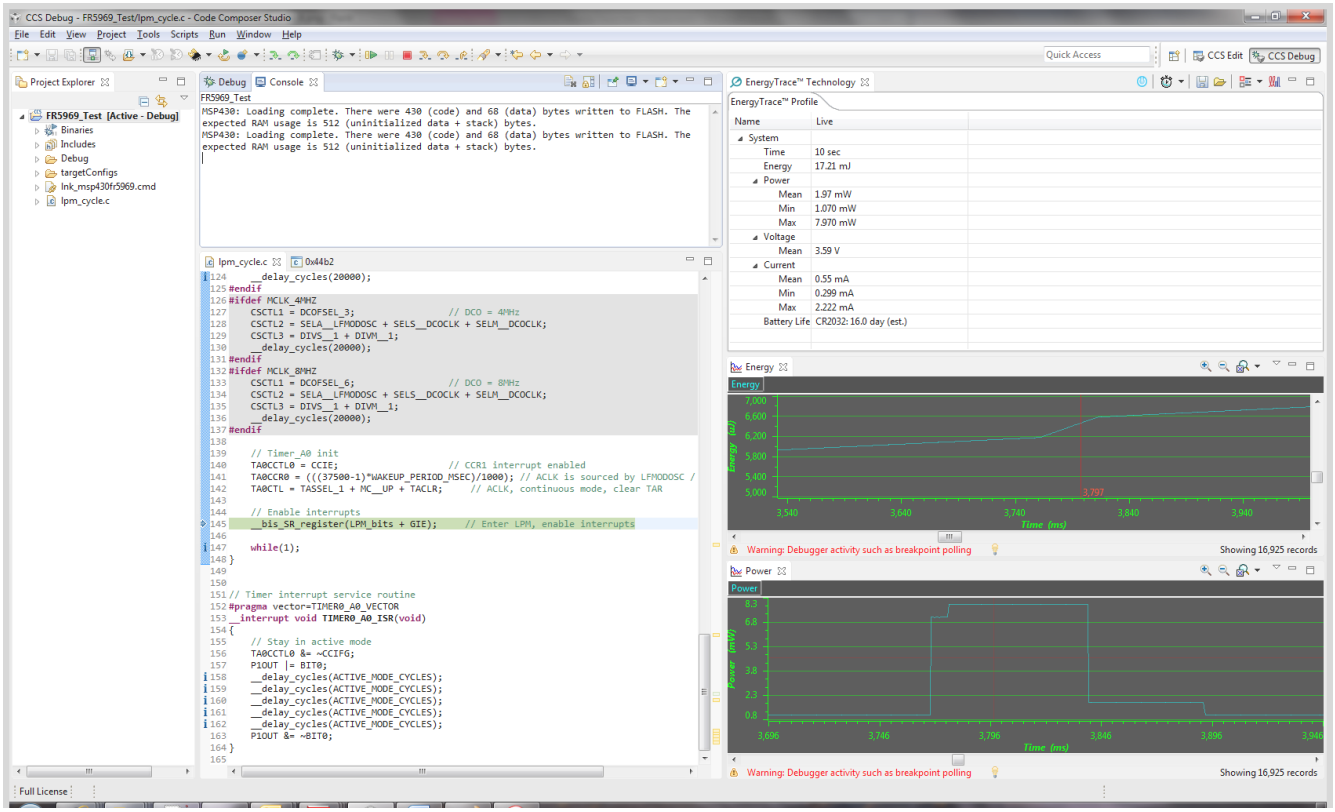

图 4-16. 使用 EnergyTrace 图表的调试会话

在 **EnergyTrace** 模式下，**系统配置 (Profile)**窗口显示已配置应用的统计数据（请参阅图 4-17）。显示下述参数：

- 采集时间
- 应用消耗的总电能（单位为 mJ ）
- 最小功率、平均功率和最大功率（单位为 mW ）
- 平均电压（单位为 V ）
- 最小电流、平均电流和最大电流（单位为 mA ）
- 采集电能曲线上所选电池的估计使用寿命（单位为天 ）

备注

计算电池使用寿命的公式假定一个理想的 3V 电池，并且未将温度、老化、峰值电流和其它会对电池容量产生负面影响的因素考虑在内。还应注意的是，更改目标电压（例如，从 3.6V 到 3V ）可能会使模拟电路的行为有所不同，电路将以更高效或更低效的状态运行，从而减少或增加能耗。系统配置 (Profile) 窗口中显示的值不能替代真实硬件上的测量值。

EnergyTrace™ Profile

Name	Live
System	
Time	10 sec
Energy	14.61 mJ
Power	
Mean	1.75 mW
Min	1.068 mW
Max	7.943 mW
Voltage	
Mean	3.59 V
Current	
Mean	0.49 mA
Min	0.298 mA
Max	2.213 mA
Battery Life	CR2032: 18.8 day (est.)

图 4-17. EnergyTrace 系统配置窗口

功率 (Power) 窗口（请参阅图 4-18）显示目标随时间变化的动态功耗。当前配置用淡蓝色标出，而之前为用于比较而重新加载的配置用黄色标出。

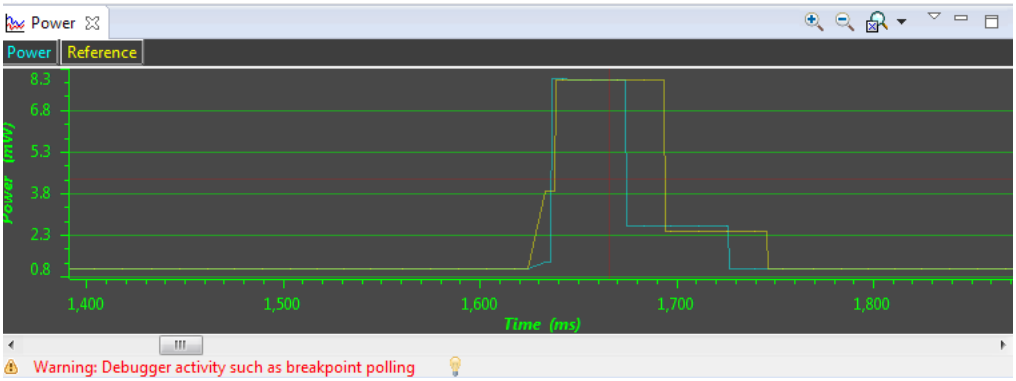

图 4-18. 放大电源窗口

Energy（电能）窗口（请参阅图 4-19）显示目标随时间变化的累积能耗。当前配置用淡蓝色标出，而之前为用于比较而重新加载的配置用黄色标出。

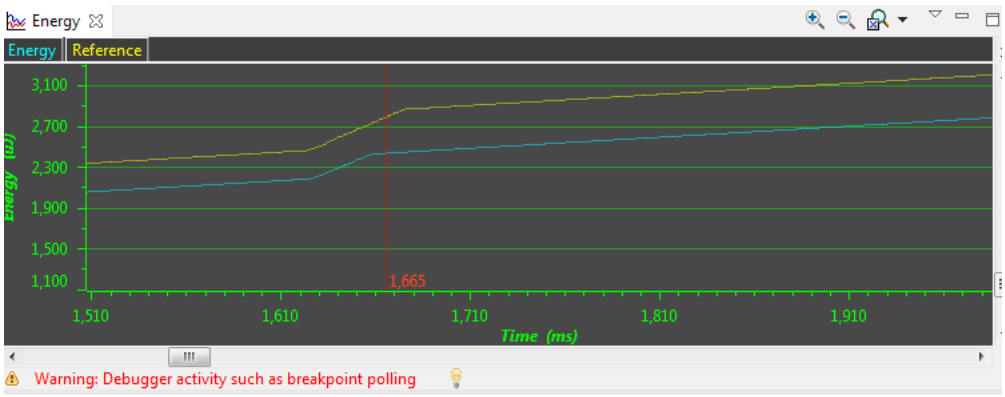

图 4-19. 放大电能窗口

备注

在使用调试器视图的**恢复 (Resume)**按钮执行程序期间，通过 JTAG 或 Spy-Bi-Wire 协议持续访问目标微控制器，以检测何时触及断点。不可避免地，这些调试访问会消耗目标域内的电能，并更改电能和功率图形所示的结果。要查看某个应用的绝对功耗，TI 建议使用 **Free Run（自由运行）**模式。在 **Free Run（自由运行）**模式下不访问目标微控制器的调试逻辑。有关调试访问对能耗的影响的示例，请参阅图 4-20。黄色曲线是在**恢复 (Resume)** 模式下记录的，绿色曲线是在**自由运行 (Free Run)**模式下记录的。

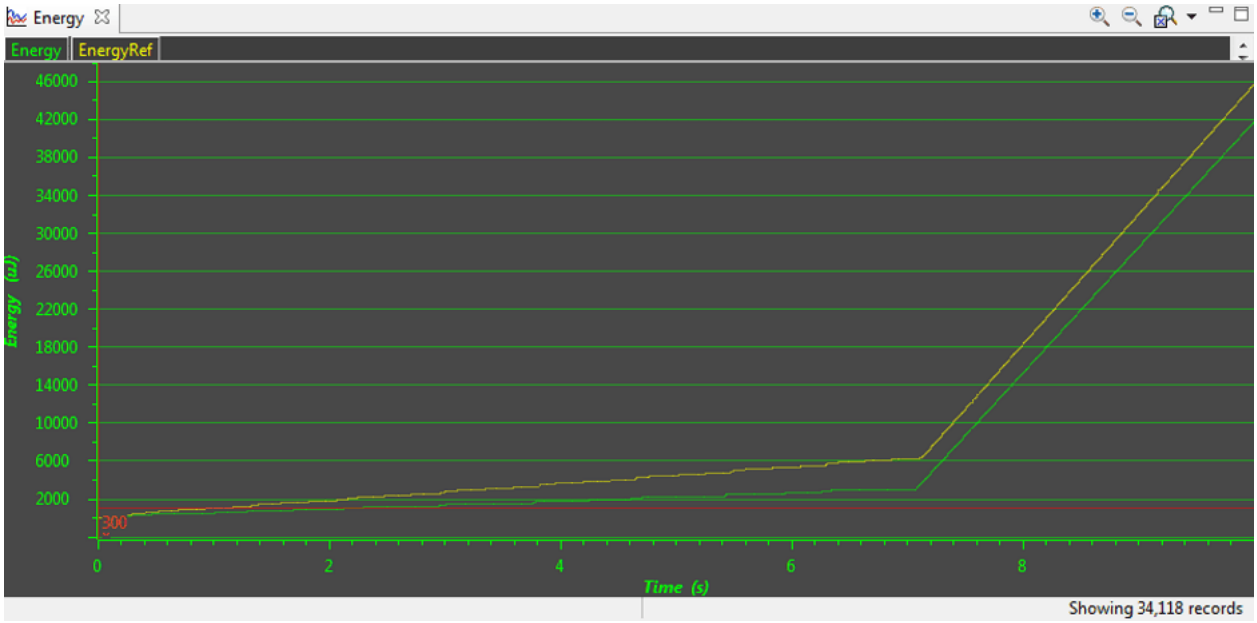

图 4-20. 同一程序在恢复（黄线）和自由运行（绿线）中的电能配置

4.3.5 将采集数据与基准数据进行比较

EnergyTrace 技术可以以多种方法使用。一种方法根据器件的预期行为检查器件的内部状态,并且纠正任何错误行为；例如，由定期使用后未将外设禁用所导致的错误。另一种方法是将采集数据与之前采集的数据进行比较。在以下讨论中，之前采集的数据称为*基准数据*。

加载基准数据后，将在功率 (Power) 和电能 (Energy) 窗口中绘制黄色基准图形。功率 (Power) 窗口显示了两个数据集随时间的功率曲线，可有助于确定静态功耗中的任何变化；例如，由于使用更深的低功耗模式或禁用未使用的外设而引起的变化。它还显示了动态功耗是如何从一个测量值更改为另外一个测量值；例如，由于执行了超低功耗顾问提示 (ULP advisor hint)。电能 (Energy) 窗口显示了随着时间的推移而累积的能耗,并指示哪种系统配置更节能。

在 **EnergyTrace++** 模式下，在系统配置 (Profile) 窗口中显示了采集数据和参考数据的压缩视图（请参阅图 4-21）。您可以快速查看两次采集会话之间总能耗以及功率模式、外设和时钟的使用是如何变化的。一般来说，变好的参数用绿条显示，而变坏的参数用红条显示。例如，主动模式中花费的时间通常被视为负数。因此，如果代码更改使得应用在主动模式下花费的时间变少，那么负增量显示为绿条，而在低功耗模式下花费的额外时间也显示为绿条。

EnergyTrace™ Technology

Reference: MSP430_D_2014_02_27_115349

EnergyTrace++™ Profile

Name	Runtime (%)	Energy (%)	Delta Runtime (%)	Delta Energy (%)	Reference Runtime (%)	Reference Energy (%)
System	100	83.5	0	-16.5	100	100
CPU						
Low Power Mode	94.3	67.2	3.5	12.9	90.8	54.3
LPM2	94.3	67.2	3.5	12.9	90.8	54.3
LPM0	0.0	0.0	-0.0	-0.0	0.0	0.0
Active Mode	5.7	32.8	-3.5	-12.9	9.2	45.7
TIMER0_A0_ISR	5.5	32.6	-3.6	-13.0	9.1	45.7
main	0.2	0.2	0.1	0.1	0.1	0.1
Peripherals						
TA0	99.8		-0.1		99.9	
FRAM	5.7		-3.5		9.2	
TA2	0.0		0.0		0.0	
TA1	0.0		0.0		0.0	
MPY	0.0		0.0		0.0	
TA3	0.0		0.0		0.0	
eUSCI_A0	0.0		0.0		0.0	
TB0	0.0		0.0		0.0	
eUSCI_A1	0.0		0.0		0.0	
RTC	0.0		0.0		0.0	
eUSCI_B0	0.0		0.0		0.0	
AES	0.0		0.0		0.0	
DMA	0.0		0.0		0.0	
COMP	0.0		0.0		0.0	
WDT	0.0		0.0		0.0	
ADC	0.0		0.0		0.0	
REF	0.0		0.0		0.0	
System Clocks						
ACLK	100.0		-0.0		100.0	
SMCLK	5.7		-3.5		9.2	
MCLK	5.7		-3.5		9.2	
MODOSC	5.7		-3.5		9.2	
VLO	0.0		0.0		0.0	

图 4-21. 在 EnergyTrace++ 模式下比较配置

在 **EnergyTrace 模式**下，没有可用的状态信息来生成详尽的报告。但是，测量过程中总能耗会与之比较,功率和电流的最小值、平均值和最大值也会与之比较。变好的参数用绿条显示，而变坏的参数用红条显示（请参阅图 4-22）。

EnergyTrace™

Reference: MSP430_2014_01_21_103959

EnergyMonitor Profile

Name	Live	Delta (%)	Reference
System			
Time	10 sec		10 sec
Energy	7.84 mJ	-11.5	8.86 mJ
Power			
Mean	0.79 mW	-10.8	0.89 mW
Min	0.582 mW	-27.3	0.800 mW
Max	1.938 mW	-0.2	1.942 mW
Voltage			
Mean	2.99 V	0.0	2.99 V
Current			
Mean	0.26 mA	-10.8	0.30 mA
Min	0.195 mA	-27.0	0.268 mA
Max	0.646 mA	-0.4	0.648 mA
Battery Life	CR2032: 35.1 day (est.)	13.0	CR2032: 31.0 day (est.)

图 4-22. 在 EnergyTrace 模式下比较配置

0% 至 50% 的区间线性绘制增量条。增量大于 50% 时不会产生更大的增量条。

4.4 EnergyTrace 技术常见问题解答

问：EnergyTrace++ 技术的采样频率是多少？

答：采样频率取决于调试程序和所选调试协议及其速度设置。其范围通常为 1kHz（例如，使用 Spy-Bi-Wire 接口时设置为慢速）到 3.2kHz（例如，使用 JTAG 接口时设置为快速）。调试程序从器件状态信息中轮询 EnergyTrace++ 的状态信息。由于采样频率不同，可能无法在状态图中采集具有短暂或快速占空比的活动外设状态。此外，采样频率越高，对 EnergyTrace 模式下的器件能耗影响越大。

问：EnergyTrace 技术的采样频率是多少？

答：无论使用何种调试协议或速度，用于测量能耗的采样频率均相同，在自由运行模式下约为 4.2kHz。

问：我的功率曲线图似乎包含噪声。是我的电路板有缺陷吗？

答：功率曲线图中显示的功率值根据测量系统计算出的累积电能推导得出（即计算得出）。当目标消耗很少的电能时，随着时间的推移，只有少量电能包提供给目标，软件需要随着时间段的推移累积 dc-dc 充电脉冲，然后才能计算新的电流值。对于低于 1μA 的电流，可能需要最多 1 秒的时间，而对于 mA 范围内电流，每毫秒都可以计算一次电流。未应用其它滤波，因此详细信息不会丢失。无论是通过采集状态信息还是通过断点轮询，另一个影响目标消耗的电能（以及电流）的因素是正常代码执行期间的周期性后台调试访问。尝试在自由运行模式中记录，以查看更平滑的功率曲线图。

问：我有一个代码会重复调用相同大小的函数。我希望函数配置能显示平均分配的运行时间。实际上，我发现某些函数的运行时间比预期的时间略长，而某些函数的运行时间稍短。

答：程序计数器跟踪期间，随着时间的推移，各种因素会影响分析器检测函数的次数。微控制器代码可以从内部高速缓存中获益，从而使一些函数的执行速度快于其他函数。另一个影响因素是内存等待状态和 CPU 流水线停顿，这给代码执行增加了时间差异。一个外部因素是调试器本身的采样频率，其通常与微控制器代码执行速度是异步运行的，但是在某些情况下显示出重叠行为，这也会导致函数运行时分布不均。

问：我的功耗模式曲线中有时会短时间显示某些代码中没有使用的功耗模式。例如，我期望从活动模式转换到 LPM3，但在转换过程中却看到了 LPM2。

答：当在 EnergyTrace++ 模式下进行采集时，会从目标器件持续收集数字信息。其中一条信息是功率模式控制信号。激活低功耗模式需要逐步通过中间状态。通常这种情况下发生地太快而无法被跟踪函数采集到，但有时可以采集中间状态并在短时间内显示为有效的低功耗模式。

问：我的配置有时包含“<待定>”低功耗模式，并且状态图电源模式部分中存在空白。“<待定>”低功耗模式源自何处？

答：从活动模式向低功耗模式转换期间，内部器件时钟被关闭，偶尔会出现状态信息未完全更新的情况。该状态在“Profile”窗口中显示为“<Undetermined>”，并且在“<Undetermined>”低功耗模式持续期间，“States”图中会显示一个缺口。“<Undetermined>”状态表示您的应用已进入低功耗模式，但无法准确确定是哪个模式。如果您的应用频繁进入低功耗模式，很有可能比很少使用低功耗模式时更频繁地显示“<Undetermined>”状态。

问：在 EnergyTrace 模式下进行采集时，即使我的程序是相同的，功率和电流的最小值与最大值也会显示偏差。我期望是完全一样的值。

答：硬件上使用的电能测量方法会随着时间的推移对 dc-dc 电荷脉冲进行计数。根据随时间变化的电能计算出电能和功率。由于统计采样效应和输出电压缓冲电容器的充放电效应，即使程序是完全一样的，电流的最小和最大值也有可能相差几个百分点。然而，采集的电能几乎相等（在指定的精度范围内）。

问：影响电能测量精度的因素有哪些？

答：电能测量电路直接由 USB 总线电压供电，因此对 USB 总线电压的变化非常敏感。校准期间，定义了单个 dc-dc 充电脉冲的电能等效值，该电能等效值取决于 USB 电压。为了确保良好的可重复性和准确性，应由活动的 USB 端口直接为调试器供电，并且避免使用总线供电的集线器以及较长的 USB 电缆，因为这会导致电压下降，特别是将其他用户连接到 USB 集线器时。而且，用于生成参考电压的低压降稳压器 (LDO) 以及校准电路中的电阻都有一定的公差和随时间变化的 ppm 速率，这些也会影响电能测量的准确性。

问：我正在尝试通过外部供电的 MSP430 器件在 EnergyTrace++ 模式或 EnergyTrace 模式下进行采集，但是“Profile”、“Energy”、“Power”和“States”窗口中未显示任何数据。

答：EnergyTrace++ 模式和 EnergyTrace 模式都要求目标由调试器供电。当目标微控制器由外部供电时，则无法采集任何数据。

问：我在 EnergyTrace++ 模式下采集数据时，无法测量 LPM 电流。我期望有几 µA，但测量结果却超过 150 µA。

答：从目标微控制器读取数字数据会消耗微控制器的 JTAG 域中的电能。因此，将电流表连接到器件电源引脚时，测得的平均电流大约为 150 µA。如果要通过调试通信消除能耗，请切换到 EnergyTrace 模式，并使目标微控制器在自由运行模式下执行。

问：我的 LPM 电流似乎是错的。我期望有几 µA，但测量结果更大，即使在自由运行模式下或使器件在没有独立电源供电的调试控制下执行也是如此。

答：产生此额外电流的最可能原因是 GPIO 端接不当，因为悬空引脚会产生额外电流。此外再次检查 JTAG 引脚，尤其是当调试器仍处于连接状态（但空闲）时，因为调试器在空闲状态下的输出信号电平可能与应用代码配置的 JTAG 引脚情况不匹配。这也可能产生额外电流。

问：当启动调试会话之前通过“View”→“Other”→“MSP430-EnergyTrace”启动 EnergyTrace++ 窗口时，数据捕获有时不能启动。

答：通过 **Window → Preferences → Code Composer Studio → Advanced Tools → EnergyTrace™ Technology** 启用 EnergyTrace。启动调试会话时，EnergyTrace++ 窗口自动打开，并且在器件执行时开始采集数据。如果在调试会话期间意外关闭所有 EnergyTrace++ 窗口，可以通过 **View → Other → MSP430-EnergyTrace** 重新打开这些窗口。

5 MSP430 FRAM 存储器保护机制

基于 FRAM 的微控制器的可用存储器可被视为统一存储器，这意味着存储器可随意分为代码区和数据区。因此，一个基于 FRAM 的微控制器可为广泛的应用使用用例进行定制。MSP430 器件支持两种存储器保护方法：

- 存储器保护单元 (MPU) 和知识产权封装 (IPE)
- FRAM 存储器写入保护 (FRWP)。一些器件上可以配置保护粒度 (1k)。

请参阅器件专用数据表以确定特定器件支持哪种方法。有关有效使用此技术的说明，请参阅 MSP430™ FRAM 技术 – 如何操作与最佳实践。

5.1 存储器保护单元 (MPU)

为了防止应用数据意外改写程序或其它形式的数据损坏，存储器保护单元允许对可用存储器分区，并为每个分区定义访问权限。这样，可以防止意外写入包含应用程序代码的存储器区域，或防止微控制器执行位于应用数据部分的指令。

图 5-1 显示了 MPU 配置对话框，此对话框适用于具有 MPU 特性的 FRAM 器件。要访问此对话框，请选择菜单 **Project → Properties → General → MPU**。此对话框允许您启用或禁用 MPU 并在自动和手动配置模式之间进行选择。对于自动配置，编译器工具链生成两个存储器段（读写存储器和可执行存储器）。器件启动期间，这两个段的段边界及其各自的访问位均位于相应的控制寄存器中。自动模式还设置了 MPU 信息存储器段的读访问位。MPUSEGxVS 位用于选择在对某个段进行非法访问时是否必须执行 PUC，默认时也为每个段进行认置。

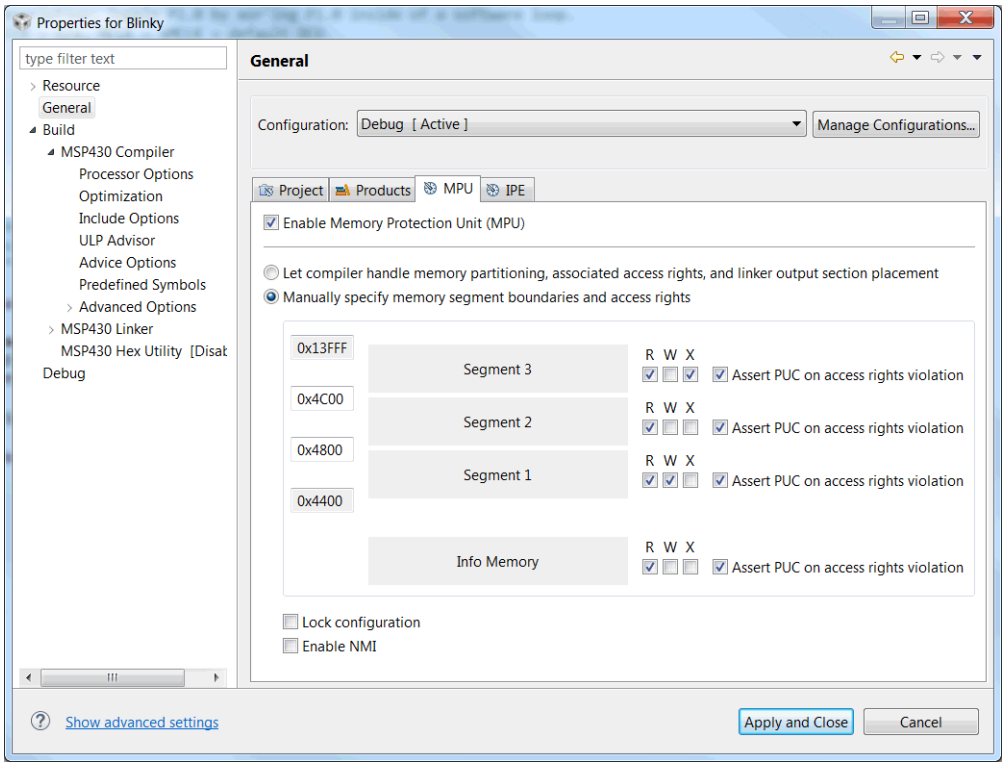

图 5-1. MPU 配置对话框

如图 5-1 所示， MPU 对话框允许对存储器保护区域进行完整的手动配置。由于段 1 的起始固定在 FRAM 存储器的起始地址上，而段 3 的结束固定在 FRAM 存储器的结束地址上，因此只需调整段 2 的起始地址和结束地址。由于这些地址分别等于段 1 的结束地址和段 3 的起始地址，因此由 GUI 自动调整。在手动配置中，可以完全独立地配置存储器及其相关访问权限。因此，用户有责任将相应代码段和数据段置于正确的存储器位置。可能还需要对链接过程进行额外配置，才能将代码和数据正确置于所需存储器位置。

5.2 知识产权封装 (IPE)

许多微控制器应用的存储器中包含了不应被公众访问的信息。这可能包含应用代码本身以及某些外设的配置设置。IPE 模块允许保护包含此类敏感信息的存储器。IPE 确保只有放在 IPE 保护区域中的程序代码才能访问此存储器段。每次代码访问时均会评估访问权限，甚至 JTAG 或 DMA 传输也无法访问 IPE 段。IPE 模块在应用程序代码启动前由引导代码进行初始化，以确保在可以执行任何用户控制的存储器访问之前，封装是有效的。

图 5-2 显示了 IPE 存储器的配置对话框，可通过菜单 **Project → Properties → General → IPE** 进行访问。IPE 对话框还提供手动和自动配置选项。在自动模式下，存储器段“.ipe”由编译器工具链生成并置于输出文件中。可通过以下源代码直接将变量置于此段中：

```
#pragma DATA_SECTION(primeNumbers, ".ipe")
const unsigned int primeNumbers[5] = {2, 3, 5, 7, 11};
```

有关如何在段内为某些代码或数据符号分配空间的更多详细说明，请参阅 MSP430™ 优化 C/C++ 编译器用户指南。

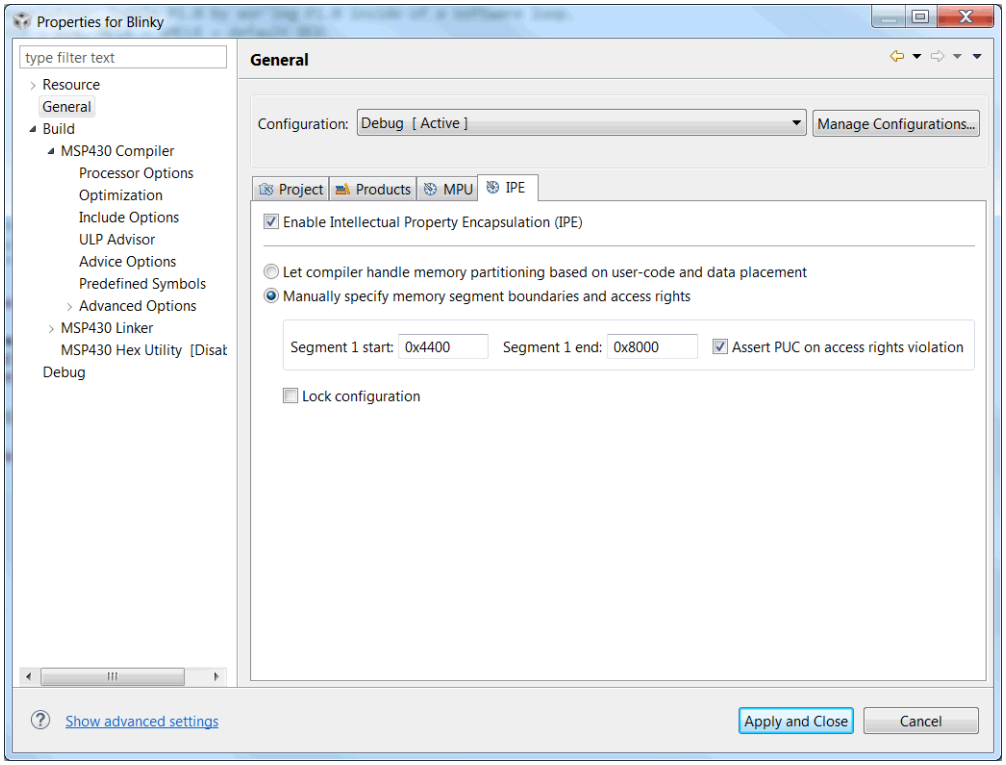

图 5-2. IPE 配置对话框

手动 IPE 模式允许用户配置 IPE 段边界和控制设置。因此，可能还需要对编译器或链接器进行额外配置，才能将代码和数据正确置于存储器中。为了防止 IPE 被修改，请将“.ipestruct”段置于 IP 封装的存储器部分。本节包含用于在器件启动期间初始化 IPE 相关寄存器的段边界和控制设置。

5.2.1 IPE 调试设置

由于可以锁定调试器而访问某些存储器区域（包括将新软件下载到器件），因此建议在目标器件处于调试器控制下时启用擦除 IP 受保护区域的选项。对应的选项可以在 **Project Properties → Debug → MSP430 Properties** 下找到（请参阅图 5-3）。

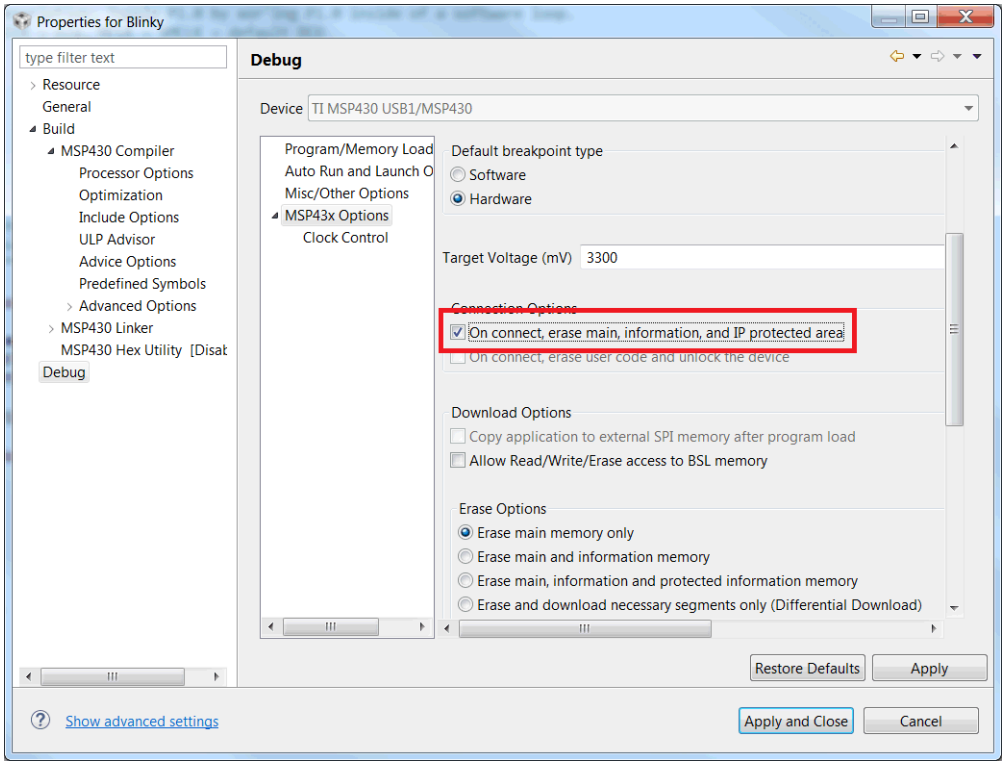

图 5-3. IPE 调试设置

5.3 FRAM 写入保护 (FRWP)

FRWP 防止对 FRAM 代码部分进行意外编程。对于 MSP430FR2xx 和 MSP430FR4xx MCU，通过设置 SYSCFG0 寄存器中的位控制来保护 FRAM 存储器。一些 MSP430 器件可以一次保护整个存储器和解除对整个存储器的保护，而 MSP430FR2355、MSP430FR2353、MSP430FR2155 和 MSP430FR2153 等一些器件可以解除对某些区域的保护并保护其余部分。

CCS 8.1 及更新版本提供了 GUI 来配置 FRAM 写入保护。默认情况下，新建 CCS 项目选择“Enable FRAM Write Protection (FRWP)“选项。

当选择了“启用 FRAM Write Protection (FRWP)”选项时，您可以保护信息存储器或解除对信息存储器的保护。当应用程序代码采用持久数据类型时，会自动计算持久数据的大小并与 1kB 大小对齐。然后，这些数据将被放置在不受保护的程序主存储器区域。而代码被放置在不受保护的程序主存储器之后。

图 5-4 显示了 FRWP 配置对话框，此对话框适用于具有该特性的 FRAM 器件。要访问此对话框，请选择菜单 **Project → Properties → General → FRWP**。

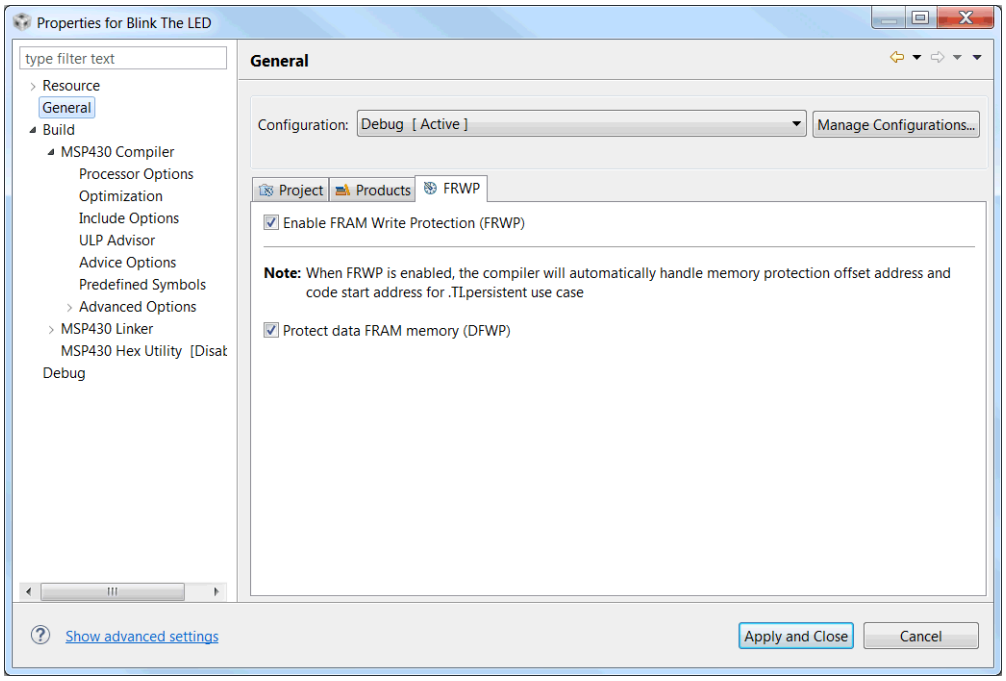

图 5-4. FRWP 配置对话框

6 常见问题和解答

本附录介绍了有关硬件、程序开发和调试工具的常见问题的解决方案。

6.1 硬件

有关硬件相关的常见问题解答的完整列表，请参阅 *MSP430 硬件工具用户指南*。

6.2 程序开发（汇编器、C 语言编译器、链接器、IDE）

备注

注意 CCS 发布注释

对于出现意外行为的情况，请参阅 CCS 发布注释文档，了解当前 CCS 版本的已知错误和限制。此信息可以通过菜单项 **Start → All Programs → Texas Instruments → Code Composer Studio → Release Notes** 进行访问。

1. **一个常见的 MSP430“错误”是未禁用看门狗机制。**看门狗默认是启用的，如果应用程序未禁用或正确管理看门狗，则会将器件复位。请使用 `WDTCL = WDTPW + WDTHOLD;` 显式禁用看门狗。该语句最好放置于在 main() 之前执行的 _system_pre_init() 函数中。如果看门狗定时器未被禁用，并且在 CSTARTUP 期间看门狗触发并复位了器件，则由于调试器不能定位用于 CSTARTUP 的源代码，**源屏幕变成空白**。请注意，如果使用了大量的初始化全局变量，CSTARTUP 则可能需要大量的时间来执行。

```
int _system_pre_init(void)
{
/* 此处插入低电平初始化 */
WDTCTL = WDTPW + WDTHOLD; // Stop Watchdog timer
/*===================================*/
/* 选择是否进行分段初始化 */
/* 是否该做.            */
/* 返回: 0 表示省略初始化 */
/*       1 表示运行初始化  */
/*===================================*/
返回 (1);
}
```

2. **在 C 库内，GIE（全局中断启用）在 使用硬件乘法器之前禁用**（并在使用硬件乘法器之后恢复）。
3. **可以在 CCS 内混合使用汇编语言程序和 C 语言程序。**请参阅 *MSP430 优化 C/C++ 编译器用户指南* 中的“使用汇编语言连接 C/C++ ”一章。
4. **.h 文件内使用的常量定义 (#define) 被有效地保留** 并且包含，例如，C，Z，N，和 V。请勿使用这些名称创建程序变量。
5. **编译器优化可以删除未使用的变量语句，这些变量和语句没有效果**能够对调试产生影响。为了防止发生这种情况，这些变量可以被声明为 `volatile`；例如：

```
volatile int i;.
```

备注

Tools Insider 博客提供了一些关于 TI MSP430 编译器和链接器的实用技巧与窍门。有关详细信息，请访问 Tools Insider 博客并阅读 *专家分享* 的系列文章。

一些实用帖子：

专家分享： 使用 TI 编译器从 RAM 执行代码

专家分享： 从链接器命令文件 (LCF) 访问文件和库

6.3 调试

调试器是 CCS 的一部分并可作为独立的应用程序使用。本节适用于独立使用和从 CCS IDE 使用调试器这两种情况。

备注

注意 CCS 版本说明

如果出现意外行为，请参阅 CCS 版本注释文档以了解当前 CCS 发布的已知缺陷限制。要访问这些信息，请点击 **Start → All Programs → Texas Instruments → Code Composer Studio → Release Notes**。

1. **调试器报告其无法与器件通信。**该问题的可能解决方案包含：
   - 确保已在 **Project → Properties → General → Device → Connection** 中选择正确的调试接口和相应的端口号。
   - 确保在目标硬件上正确配置跳线设置。
   - 确保没有其他软件应用（例如，打印机驱动程序）保留或者控制了 COM 或并行端口，这将妨碍调试服务器与器件的通信。
   - 打开器件管理器,并确定用于 FET 工具的驱动程序是否已正确安装以及 Windows 操作系统 (OS) 是否成功识别 COM 或并行端口。检查 PC BIOS 中的并行端口设置（请参阅常见问题解答 5）。对于使用 IBM 或联想 ThinkPad® 计算机的用户，即使操作系统报告并行端口位于 LPT1，也请尝试设置端口 LPT2 和 LPT3。
   - 重启计算机。

   确保 MSP430 器件牢靠地安装在插槽中（使插槽的“金手指”与器件的引脚完全接合），并且其引脚 1（在顶部表面有一个环形凹口）与印刷电路板 (PCB) 上的标记“1”对齐。

   CAUTION

   可能对器件造成损坏

   始终只使用真空拾取工具处理 MSP430；请勿使用手指，因为您能很容易地弯曲引脚致使器件无用。此外，时刻遵守并遵循正确的静电放电 (ESD) 预防措施。
2. **调试器能够调试使用中断和低功耗模式的应用程序。**请参阅常见问题解答 17。
3. **当器件运行时，调试器无法访问器件寄存器和存储器。**用户必须停止器件才能访问器件寄存器和存储器。

4. **调试器报告器件 JTAG 安全保险丝被熔断**。借助现有的 MSP430-FET430UIF JTAG 接口工具，在对外部供电的目标板进行适配时会存在一个弱点。这会导致 MSP430 意外检查保险丝并导致 JTAG 安全保险丝被识别为已被熔断，尽管其未被熔断。

   权变措施：
   - 将器件 $\overline{\text{RST}}$/NMI 引脚连接到 JTAG 接头（引脚 11），MSP-FET430UIF 接口工具能够拉动 $\overline{\text{RST}}$ 线路，这也是对器件内部保险丝逻辑的复位。
   - 请勿同时连接 V_{CC} 工具（引脚 2）和 JTAG 接头的 V_{CC} 目标（引脚 4）。在为调试器中指定与电源电压相等的 V_{CC} 值。

   备注

   如果尝试擦除或写入闪存时 V_{CC} 电压不够高，控制台中会显示以下消息：“目标器件电源电压太低，无法进行闪存擦除/编程”。如果出现此情况，请尝试更改电源电压。
5. **并行端口标识符 (LPTx) 具有以下物理地址：LPT1 = 378h，LPT2 = 278h，LPT3 = 3BCh。**并行端口的配置（ECP、兼容、双向、正常）并不重要；ECP 似乎运行良好。有关解决调试器与器件间通信问题的其他提示，请参阅常见问题解答 1。
6. 当调试器被启动并且器件被编程时，**调试器将 $\overline{\text{RST}}$/NMI 变为高电平来使器件复位。**当器件被手工重新编辑（使用重新载入 (Reload)），当 JTAG 被重新同步时（使用再同步 (Resynchronize) JTAG），此器件也可由调试器的复位 (Reset) 按钮复位。当 $\overline{\text{RST}}$/NMI 未置位（低电平）时，调试器将驱动 $\overline{\text{RST}}$/NMI 的逻辑置于高阻抗状态，并通过 PCB 上的电阻器将 $\overline{\text{RST}}$/NMI 上拉至高电平。

   当调试器启动时，$\overline{\text{RST}}$/NMI 会置位并在加电后取负。然后 $\overline{\text{RST}}$/NMI 被置成有效并在器件初始化完成之后第二次被置成无效。
7. **调试器可以调试将 $\overline{\text{RST}}$/NMI 引脚功能重新配置为 NMI 的器件。**
8. **当调试器复位器件时，XOUT/TCLK 引脚的电平是未定义的。**驱动 XOUT/TCLK 的逻辑电路会在所有其他时候都被设置为高阻抗。
9. **当对器件的电流进行测量时，必须确保释放 JTAG 控制信号**，否则器件由 JTAG 引脚上的信号供电，且测量结果错误。请参阅常见问题解答 10。
10. **当调试器控制器件时，无论状态寄存器中的低功耗模式位的设置如何，CPU 都是接通的**（也就是说，它不处于低功耗模式）。在 STEP 或者 GO 之前恢复任一低功耗模式条件。因此，在调试器控制器件时，请勿测量器件的功耗。相反，使用运行时释放 JTAG 来运行应用程序。
11. MEMORY（内存）窗口正确显示了其所在存储器的内容。然而，**“MEMORY”（内存）窗口错误地显示了其不在的存储器的内容。**存储器只能在器件数据表指定的地址范围内使用。
12. 调试期间，调试器使用系统时钟来控制器件。因此，**当调试器能够控制此器件时，由主系统时钟 (MCLK) 提供时钟信号的器件计数器和其他组件会受到影响。**应当采取特别的预防措施来尽可能地减少对看门狗计时器的影响。保留 CPU 内核寄存器。所有其他时钟源（SMCLK 和 ACLK）和外设在仿真期间继续正常运行。换句话说，**闪存仿真工具是部分侵入性工具。**

    支持时钟控制的器件可以通过在调试期间停止时钟 (**Project → Properties → CCS Debug Settings → Target → Clock Control**)来进一步最大程度地减少这些影响。

13. 当对闪存进行编程时，请勿**在紧跟写入闪存操作之后的指令上设置断点。**解决这一限制简单的权变措施是使用 NOP 执行写入闪存操作，并在 NOP 之后的指令上设置断点。
14. 需要多个内部机器周期来清除和编辑闪存。**当单步执行控制闪存的指令时，**在这些操作完成前将控制权交还给调试程序。因此，**调试器使用错误信息更新其内存窗口。**该行为的权变措施是使用 NOP 跟随闪存访问指令，然后在检查闪存访问指令的效果之前跳过此 NOP。
15. 在**正常程序执行期间读取被清除的位**（也就是说，中断标志）在调试时**被清除**

    使用带有增强型仿真逻辑的某些 MSP430 器件，例如 MSP430F43x 和 MSP430F44x 器件，数据位并不以这种方式表示（即，调试器读取操作并不清除数据位）。
16. **调试器不能用于调试在 F12x 和 F41x 器件的 RAM 中执行的程序。**解决该限制的权变措施是在闪存中调试程序。
17. 当**单步激活并启用中断时，似乎只有中断服务例程 (ISR) 处于活动状态**（也就是说，非 ISR 代码似乎永远不会执行，并且单步运行在 ISR 第一行上停止）。然而，该行为是正确的，因为器件在处理非 ISR（即主线路）代码之前处理了一个有效且被启用的中断。解决该行为的的权变措施是，在 ISR 中，禁用堆栈上的 GIE 位，这样能够在退出 ISR 之后将中断禁用。这允许调试非 ISR 代码（但不中断）。稍后可以通过在寄存器窗口中的状态寄存器中设置 GIE 来重新启用中断。

    在带有时钟控制的器件上，或许可以在单步运行之间将时钟挂起，并推迟中断请求 (**Project → Properties → CCS Debug Settings → Target → Clock Control**)。
18. 在配备了数据传输控制器 (DTC) 的器件上，**数据传输周期的完成会抢占低功耗模式指令的一个步骤。**只有在处理完中断后，器件才能超过低功耗模式指令继续运行。在处理中断之前，单个步骤似乎没有任何作用。解决这种情况的权变措施是在低功耗模式指令之后的指令上设置一个断点，然后执行（运行）至这个断点。
19. **数据传输控制器 (DTC) 的数据传输可能不会在 DTC 响应单个步骤或者断点而停止时准确地停止。**当 DTC 被启用且单个步骤被执行时，可以传送一个或者多个字节的数据。当 DTC 被启用且被配置用于双块传送模式时，当响应单步或断点而停止时，DTC 可能不会恰好停止在数据块边界上。
20. **断点。**CCS 支持许多预定义的断点和观察点类型。有关详细的概述，请参阅节 3.2.2。

备注

如果 MSP-FET430UIF 上具有旧的固件映像 (MSP Debug Stack v2)，则 Linux 和 OS X 不支持 MSP-FET430UIF。

购买新款 MSP-FET430UIF 的客户将会在 OS X 或 Linux 上遇到此问题，因为生产期间在调试器上编程设置了 MSP Debug Stack v2。要解决此问题，请将调试器连接到 Windows PC，然后使用 IAR、CCS 或 MSP430 刷写工具来将调试器上的固件更新到最新版本（v3 或更新版本）。

备注

在 MSP-FET、MSP-FET430UIF 或 LaunchPad™ 开发套件上执行固件更新时，请勿通过 USB 集线器进行连接。

7 将 C 代码从 IAR 2.x、3.x、4.x、5.x、6.x 或 7.x 迁移到 CCS

TI CCS C 编译器的源代码和 IAR Embedded Workbench（嵌入式工作台）C 编译器的源代码并不完全兼容。标准的 ANSI/ISO C 代码在这些工具间可移植，但实现专用扩展有所不同，必须被移植。本附录描述了两个编译器之间的主要差异。

7.1 中断矢量定义

现在 CCS 完全支持 IAR ISR 声明（使用 #pragma vector=）。然而，并非所有其他 IAR 编译指令都是如此。

7.2 内在功能

CCS 和 IAR 使用相同的指令实现 MSP430 处理器专用的内在功能。

7.3 数据和函数放置

7.3.1 数据放置在绝对位置上

在 IAR 编译器使用 @ 运算符或 #pragma location 指令实现的方案 CCS 编译器中是不支持的：

```
/* IAR C 代码 */
__no_init char alpha @ 0x0200;  /* 将“α”放置在地址 0x200 处 */
#pragma location = 0x0202
const int beta;
```

如果需要绝对数据放置，可以通过在链接器命令文件输入条目来实现，然后在 C 代码中将变量声明为 extern：

```
/* CCS 链接器命令文件条目 */
alpha = 0x200;
beta = 0x202;
/* CCS C Code */
extern char alpha;
extern int beta;
```

绝对 RAM 位置必须从 RAM 段中排除；否则，当链接器自动分配地址时，其内容可能会被覆盖。RAM 模块的开始地址和长度必须在链接器命令文件中进行修改。对于之前的例子，RAM 起始地址必须从 0x0200 移动 4 个字节至 0x0204 ，这样长度就从 0x0080 减少至 0x007C（对于具有 128 字节 RAM 的 MSP430 器件）：

```
/* CCS 链接器命令文件条目 */
/*************************************************************************/
/* 指定系统内存映射图 */
/*************************************************************************/
MEMORY /* 假设器件具有 128 字节的 RAM */
{
...
RAM    :origin = 0x0204, length = 0x007C  /* was: origin = 0x200, length = 0x0080 */
...
}
```

链接器命令文件 (lnk_msp430xxxx.cmd) 与 CCS 随附的器件专用标头文件(msp430xxxx.h 中的外设寄存器映射的定义 (msp430xxxx.h) 是将数据置于绝对位置上的一个示例。

备注

创建项目时，CCS 从包含目录（<安装根目录>\ccsv5\ccs_base\tools\compiler\MSP430\include）中拷贝与所选 MSP430 衍生器件相对应的链接器命令文件到项目目录。因此，确保在项目目录中完成所有链接器命令文件的更改。这允许对使用同一器件的不同项目使用项目专用链接器命令文件。

7.4 数据放置到命名的段中。

在 IAR 中，可以使用 @ 运算符或者 #pragma directive 来将变量放置到命名的分段中：

```
/* IAR C 代码 */
__no_init int alpha @ "MYSEGMENT";  /* 将“α”放入“MYSEGMENT”中 */
```

```
#pragma location="MYSEGMENT"        /* 将“β”放入“MYSEGMENT”中 */
const int beta;
```

借助于 CCS 编译器时，必须使用 #pragma DATA_SECTION() 指令：

```
/* CCS C 代码 */
#pragma DATA_SECTION(α, "MYSEGMENT")
int alpha;
#pragma DATA_SECTION(β, "MYSEGMENT")
int beta;
```

有关如何在 IAR 和 CCS 之间转换存储器段名称的信息，请参阅节 7.7.3。

7.5 函数放置到命名段中

借助 IAR 编译器，可使用 @ 运算符或 #pragma location 指令将函数放置到命名段中：

```
/* IAR C 代码 */
void g(void) @ "MYSEGMENT"
{
}
#pragma location="MYSEGMENT"
void h(void)
{
}
```

借助 CCS 编译器，必须使用以下带有 #pragma CODE_SECTION() 指令的方案：

```
/* CCS C 代码 */
#pragma CODE_SECTION(g, "MYSEGMENT")
void g(void)
{
}
```

有关如何在 IAR 和 CCS 之间转换存储器段名称的信息，请参阅节 7.7.3。

7.6 C 调用约定

CCS 和 IAR C 语言编译器使用不同的调用约定将参数传递给函数。当将混合的 C 语言和编译项目移植到 TI CCS 代码生成工具时，需修改编译函数以反映这些更改。有关调用约定的详细信息，请参阅 *TI MSP430 优化 C/C++ 编译器用户指南* 和 *IAR MSP430 C/C++ 编译器参考指南*。

以下示例函数将 32 位字 数据 以大端字节序格式写入指定存储器位置中。可以看出,参数 数据是使用不同的 CPU 寄存器传递的。

IAR 版本：

```
;-------------------------------------------------------------------------------
; void WriteDWBE(unsigned char *Add, unsigned long Data)
;
; Writes a DWORD to the given memory location in big-endian format.The
; memory address MUST be word-aligned.
;
; IN:  R12    Address      (Add)
;      R14    Lower Word   (Data)
;      R15    Upper Word   (Data)
;-------------------------------------------------------------------------------
WriteDWBE
   swpb    R14          ; Swap bytes in lower word
   swpb    R15          ; Swap bytes in upper word
   mov.w   R15,0(R12)   ; Write 1st word to memory
   mov.w   R14,2(R12)   ; Write 2nd word to memory
   ret
```

CCS 版本：

```
;-------------------------------------------------------------------------------
; void WriteDWBE(unsigned char *Add, unsigned long Data)
;
; Writes a DWORD to the given memory location in big-endian format.The
; memory address MUST be word-aligned.
;
; IN:  R12    Address     (Add)
;      R13    Lower Word  (Data)
;      R14    Upper Word  (Data)
;-------------------------------------------------------------------------------
WriteDWBE
   swpb    R13          ; Swap bytes in lower word
   swpb    R14          ; Swap bytes in upper word
   mov.w   R14,0(R12)   ; Write 1st word to memory
   mov.w   R13,2(R12)   ; Write 2nd word to memory
   ret
```

7.7 其它差异

7.7.1 初始化静态和全局变量

ANSI/ISO C 标准规定没有明确初始化的静态和全局（伪指令）变量必须被预初始化为 0（在程序开始运行之前）。通常在程序被加载且在 IAR 编译器中被执行时进行此任务：

```
/* IAR, 全局变量,程序启动时初始化为 0  */
int Counter;
```

然而，TI CCS 编译器并不预初始化这些变量；因此，由应用程序满足这一要求：

```
/* CCS, 全局变量, 手动零初始化 */
int Counter = 0;
```

7.7.2 自定义启动例程

借助 IAR 编译器，可以自定义 C 语言启动函数，使应用程序有机会执行配置外设等初期初始化，或者忽略数据段初始化。这是通过提供 customized __low_level_init() 函数来实现的：

```
/* IAR C 代码 */
int __low_level_init(void)
{ =
/* 此处插入低电平初始化 */
/*================================== */
/* 选择是否分段初始化  */
/* 是否该做.              */
/* 返回:  0 表示省略初始化 */
/*          1 表示运行初始化  */
/*================================== */
返回 (1);
}
```

返回值控制数据段是否由 C 语言启动代码进行初始化：借助于 CCS 编译器，自定义启动例程名为 _system_pre_init()。其使用方法与在 IAR 编译器中的使用方法相同。

```
/* CCS C 代码 */
int _system_pre_init(void)
{
/* 此处插入低电平初始化 */
/*================================== */
/* 选择是否分段初始化  */
/* 是否该做.              */
/* 返回:  0 表示省略初始化 */
/*          1 表示运行初始化  */
/*================================== */
返回 (1);
}
```

在两个编译器都省略分段初始化会同时省略显式和非显式初始化。用户必须确保在使用重要的变量之前,在运行时对变量初始化。

7.7.3 预定义的存储器段名称

数据和函数放置的存储器段名称由 CCS 和 IAR 工具中器件专用链接器命令文件控制。然而，使用了不同的段名称。有关更多详细信息，请参阅链接器命令文件。下述表格显示了如何转换最常用的段名称。

说明	CCS 段名称	IAR 段名称
RAM	.bss	DATA16_N DATA16_I DATA16_Z
堆栈 (RAM)	.stack	CSTACK
主存储器 (闪存或者 ROM)	.text	代码
信息存储器 (闪存或者 ROM)	.infoA .infoB	INFOA INFOB INFO
中断矢量 (闪存或者 ROM)	.int00 .int01 ... .int14	INTVEC
复位矢量 (闪存或者 ROM)	.reset	复位

7.7.4 预定义的宏名称

IAR 和 CCS 编译器均支持一些非 ANSI/ISO 标准预定义宏名称，这有助于创建可以在不同编译器平台上编译和使用的代码。检查是否使用 #ifdef directive 定义宏名称。

说明	CCS 宏名称	IAR 宏名称
MSP430 是否是目标？MSP430 是否是所使用的特定编译器平台？	__MSP430__	__ICC430__
是否是所使用的特定编译器平台吗？	__TI_COMPILER_VERSION__	__IAR_SYSTEMS_ICC__
C 头文件是否包含在编译源代码？	__ASM_HEADER__	__ IAR_SYSTEMS_ASM__

8 将汇编器代码从 IAR 2.x、3.x、4.x、5.x、6.x 或 7.x 迁移到 CCS

TI CCS 汇编器的源代码和 IAR 汇编器的源代码并不是 100% 兼容。指令助记符相同，但汇编器指令有些不同。本附录描述了 CCS 汇编器指令和 IAR 汇编器指令之间的差异。

8.1 与汇编源代码共享 C/C++ 头文件

IAR A430 汇编器直接支持某些 C/C++ 预处理器指令，从而允许将诸如 MSP430 器件专用头文件 (msp430xxxx.h) 的 C/C++ 头文件直接放入汇编代码中：

```
#include "msp430x14x.h" // Include device header file
```

借用 CCS Asm430 汇编器，必须使用采用 .cdecls 指令的不同方案。该指示允许混合汇编和 C/C++ 环境的程序设计者在 C/C++ 头文件和汇编代码之间共享包含 C/C++ 标头文件的语句和原型：

```
.cdecls C, LIST, "msp430x14x.h"; 包括器件头文件
```

有关 .cdecls 指令的更多信息，请参阅 *MSP430 汇编语言工具用户指南*。

8.2 段控制

CCS Asm430 汇编器不支持任一 IAR A430 段控制指令，诸如 ORG，ASEG，RSEG，和 COMMON。

说明	Asm430 指令 (CCS)
在 .bss 未初始化区域中保留空间	.bss
在命名的未初始化区域中保留空间	.usect
将程序分配至默认程序区域中（初始化）	.text
将数据分配至已命名的初始化区域中	.sect

为了使用 CCS 汇编器将代码和数据部分分配至指定的地址，有必要创建并使用在链接器命令文件中定义的存储器部分。下述示例演示了在 IAR 和 CCS 汇编中的中断矢量分配，以突出它们之间的差异。

```
;------------------------------------------------------------------------
; Interrupt Vectors Used MSP430x11x1 and 12x(2) - IAR Assembler
;------------------------------------------------------------------------
   ORG     0FFFEh      ; MSP430 RESET Vector
   DW      RESET       ;
   ORG     0FFF2h      ; Timer_A0 Vector
   DW      TA0_ISR     ;
;------------------------------------------------------------------------ ;
Interrupt Vectors Used MSP430x11x1 and 12x(2) - CCS Assembler
;------------------------------------------------------------------------
   .sect   ".reset"    ; MSP430 RESET Vector
   .short  RESET       ;
   .sect   ".int09"    ; Timer_A0 Vector
   .short  TA0_ISR     ;
```

两个例子都假定使用了标准的器件支持文件（头文件、链接器命令文件）。IAR 和 CCS 的链接器命令文件是不同的，并且不能重复使用。有关如何在 IAR 和 CCS 之间转换存储器段名称的信息，请参阅节 7.7.3。

8.3 将 A430 汇编程序指令转化为 Asm430 指令

8.3.1 引言

下述各节大致介绍了如何将 IAR A430 汇编器 (A430) 的汇编器指令转换为 TI CCS Asm430 汇编器 (Asm430) 指令。这些章节仅作为转换指南。有关每条指令的详细说明，请参阅 TI 提供的 *MSP430 汇编语言工具用户指南*，或者 IAR 提供的 *MSP430 IAR 汇编器参考指南*。

备注

仅汇编器 *指令* 需要转换

仅汇编器指令需要转换，汇编器指令无需转换。两种汇编器均使用相同的指令助记符、操作数、运算符和特殊符号(例如，段程序计数器 ($) 和注释分隔符 (;)。

默认情况下，A430 汇编器不区分大小写。这些章节以大写形式显示 A430 指令，以区别于小写形式显示的 Asm430 指令。

8.3.2 字符串

除了使用不同的指令外，每个汇编器还对不同的字符串使用不同的语法。A430 对字符串采用 C 语言语法：引号由反斜杠字符作为转义字符加上引号 (\") 来表示，而反斜杠本身由两个连续的反斜杠表示 (\\)。在 Asm430 语法中，一个引号用两个连续的引号 ("") 表示；请参阅示例：

字符串	Asm430 语法 (CCS)	A430 语法 (IAR)
计划 "C"	"计划 ""C"""	"计划 \"C\""
\dos\command.com	"\dos\command.com"	"\\dos\\command.com"
串联字符串（例如，错误 41）	-	“错误” "41"

8.3.3 区域控制指示

Asm430 有三个预定义区域，在这些区域中一个程序的不同部分被汇编在一起。未初始化数据被汇编至 .bss 区域，初始化数据被汇编至 .data 区域，而可执行代码被汇编至 .text 区域。

A430 也使用区域或者分段，但是没有预定义的段名称。通常，遵循 C 编译器使用的名称是很方便的。DATA16_Z 表示未初始化数据，CONST 表示常量（已初始化）数据，而 CODE 表示可执行代码。下述表格使用这些名称：

一对分段可被用于对经过初始化的、可更改的数据进行 PROM 编程。ROM 分段将包含初始化程序，并由启动程序复制到 RAM 中。在这种情况下，所有这些段必须大小一致且布局一样。

说明	Asm430 指令 (CCS)	A430 指令 (IAR)
在 .bss（未初始化的数据）区域中保留大小字节	.bss(1)	(2)
汇编至 .data（初始化数据）区域	.data	RSEG 常量
汇编至命名（初始化）区域	.sect	RSEG
汇编至 .text（可执行代码）区域	.text	RSEG 代码
在命名（未初始化）区域中保留空间	.usect(1)	(2)
在字节边界上对齐	.align 1	(3)
在字边界上对齐	.align 2	EVEN

(1) .bss 和 .usect 不需要在初始区域和未经初始化区域间来回切换。例如：

```
; IAR Assembler Example
RSEG DATA16_N ; Switch to DATA segment
EVEN ; Ensure proper alignment
ADCResult: DS 2 ; Allocate 1 word in RAM
Flags: DS 1 ; Allocate 1 byte in RAM
RSEG CODE ; Switch back to CODE segment
; CCS Assembler Example #1
ADCResult .usect ".bss",2,2 ; Allocate 1 word in RAM
Flags .usect ".bss",1 ; Allocate 1 byte in RAM
; CCS Assembler Example #2
.bss ADCResult,2,2 ; Allocate 1 word in RAM
.bss Flags,1 ; Allocate 1 byte in RAM
```

(2) 通过首先切换到未初始化段，然后定义适当的内存块，再切换回初始化段，可以在未初始化段中保留空间。例如：

```
RSEG DATA16_Z
LABEL: DS 16 ; Reserve 16 byte
RSEG CODE
```

(3) 不支持位字段常量 (.field) 的初始化，因此，段计数器总是与字节对齐。

8.3.4 常量初始化指令

说明	Asm430 指令 (CCS)	A430 指令 (IAR)
初始化一个或多个连续的字节或者文本字符串	.byte 或者 .string	DB
初始化 32 位 IEEE 浮点常量	.double 或者 .float	DF
初始化长度可变域	.field	(1)
在当前字段中保留大小字节	.space	DS
初始化一个或者多个文本字符串	初始化一个或者多个文本字符串	DB
初始化一个或者多个 16 位整数	.word	DW
初始化一个或者多个 32 位整数	.long	DL

(1) 不支持初始化位域常量 (.field)。必须使用 DW 将常量合并成完整的字。

```
; Asm430 code ; A430 code
.field 5,3 \
.field 12,4 | -> DW (30<<(4+3))|(12<<3)|5 ; equals 3941
.field 30,8 /
```

8.3.5 列出控制指令

说明	Asm430 指令 (CCS)	A430 指令 (IAR)
允许错误的条件代码块列表	.fclist	LSTCND-
禁止出错误的条件代码块列表	.fcnolist	LSTCND+

说明	Asm430 指令 (CCS)	A430 指令 (IAR)
设定源列表的页长度	.length	PAGSIZ
设定源列表的页宽度	.宽度	COL
重新启动源列表	.list	LSTOUT+
停止源列表	.nolist	LSTOUT-
允许宏列表和环路阻止	.mlist	LSTEXP+（宏） LSTREP+（循环块）
禁止宏列表和环路阻止	.mnolist	LSTEXP-（宏） LSTREP-（循环块）
选择输出列表选项	.option	(1)
在源码表中弹出页面	.page	（页面）
允许扩展的替代符号列表	.sslist	(2)
禁止扩展的替代符号列表	.ssnolist	(2)
在列表页面标头内打印标题	.title	(3)

(1) 没有与 .option 直接相对应的 A430 指令。单独的列表控制指令（上述的）或者命令行选项 -c（带有子选项）应该被用于替代 .option 指示。
(2) 没有与 .sslist 和 .ssnolist 直接相对应的指令。
(3) 列表页面村头中的标题为源文件名。

8.3.6 文件参考指令

说明	Asm430 指令 (CCS)	A430 指令 (IAR)
包含来自另一个文件的源语句	.copy 或者 .include	#include or $
识别一个或多个在当前模块中定义且在其它模块中使用的符号	.def	PUBLIC 或者 EXPORT
识别一个或多个局（外部）符号	.global	(1)
定义宏库	.mlib	(2)
识别一个或多个在当前模块中使用但在其它模块中定义的符号	.ref	EXTERN 或者 IMPORT

(1) 如果符号是在当前模块中定义的，指令 .global 函数为 .def，否则为 .ref。PUBLIC 或者 EXTERN 必须适用于 A430 汇编器，以替代 .global 指令。
(2) 不支持宏库概念。包含带有宏定义的文件必须用于此功能。

模块可与 Asm430 汇编器一起使用以创建可单独链接的例程。一个文件可能包含多个模块或者例程。除 DEFINE ，#define（IAR 预处理器指令）或 MACRO 创建的符号外，所有符号在模块结束时都是“未定义”的。此外，库模块有条件链接。这意味着只有当模块中的公共符号被外部引用时，库模块才包含在链接的可执行代码中。下列指示用于标记 A430 汇编器内模块的开始和结束。

附加的 A430 指令 (IAR)	A430 指令 (IAR)
开始程序模块	NAME（名称）或者 PROGRAM（程序）
开始库模块	MODULE（模块）或者 LIBRARY（库）
终止当前程序或者库模块	ENDMOD（结束模式）

8.3.7 条件汇编指令

说明	Asm430 指令 (CCS)	A430 指令 (IAR)
可选可重复模块汇编	.break	(1)
开始进行条件汇编	.if	IF
可选条件汇编	.else	ELSE
可选条件汇编	.elseif	ELSEIF
终止条件汇编	.endif	ENDIF
终止可重复模块汇编	.endloop	ENDR
开始可重复模块汇编	.loop	REPT

(1) 没有与 .break 直接相对应的指令。然而，如果在宏中使用可重复模块汇编，EXITM 指令可与其他条件一起使用，如下所示：

```
SEQ MACRO FROM,TO ; Initialize a sequence of byte constants
LOCAL X
X SET FROM
REPT TO-FROM+1 ; Repeat from FROM to TO
IF X>255 ; Break if X exceeds 255
EXITM
ENDIF
DB X ; Initialize bytes to FROM...TO
X SET X+1 ; Increment counter
ENDR
ENDM
```

8.3.8 符号控制指令

汇编时间符号的范围在两个汇编器中是不同的。在 Asm430 中，定义可以是文件的全局定义，也可以是模块或者宏的局部定义。局部符号可以使用 .newblock 指令进行未定义。在 A430 中，符号要么是宏 (局部)的局部符号，要么是模块 (EQU)的局部符号，要么是文件 (DEFINE)的全局符号。此外，预处理器指示 #define 还可用于定义局部符号。

说明	Asm430 指令 (CCS)	A430 指令 (IAR)
将字符串分配给替代符号	.asg	SET 或者 VAR 或者 ASSIGN
未定义局部符号	.newblock	(1)
将符号等同于一个值	.equ 或者 .set	EQU 或者 =
对数字替代符号进行算术运算	.eval	SET 或者 VAR 或者 ASSIGN
结束结构定义	.endstruct	(2)
开始结构定义	.struct	(2)
将结构属性分配给标签	.tag	(2)

(1) 没有与 .newblock 直接相对应的 A430 指示。然而，#undef 可用于复位一个使用 #define 指令定义的符号。此外，因为在宏或者模块结束时隐式未定义局部符号，宏或者模块可被用于实现 .newblock 功能性。

(2) 不支持结构类型的定义。通过使用宏分配聚合数据和基础地址以及符号偏移量可以实现类似的功能，如下所示：

```
MYSTRUCT: MACRO
DS 4
ENDM
LO DEFINE 0
HI DEFINE 2
RSEG DATA16_Z
X MYSTRUCT
RSEG CODE
MOV X+LO,R4
...
```

8.3.9 宏指令

说明	Asm430 指令 (CCS)	A430 指令 (IAR)
定义一个宏	.macro	宏
提前退出宏	.mexit	EXITM(退出)
结束宏定义	.endm	ENDM(结束宏)

8.3.10 混合指令

说明	Asm430 指令 (CCS)	A430 指令 (IAR)
发送用户定义的错误消息至输出器件	.emsg	#error
发送用户定义的消息至输出器件	.mmsg	#消息(1)
发送用户定义的警告消息至输出器件	.wmsg	(2)
定义加载地址标签	.label	(3)
由绝对列表器产生的指令	.setsect	ASEG(4)
由绝对列表器产生的指令	.setsym	EQU 或者 =(4)
程序结束	.end	END（结束）

(1) #message 指令的语法为：#message "<string>"
这会导致汇编和编译期间将 “#message <string>” 输出到项目构建窗口中。

(2) 警告消息无法由用户定义。可以使用 #消息（message），但是警告计数器不会增加。

(3) 不支持加载时间地址的概念假定运行时间和加载时间地址相同。要实现相同的效果，可以使用 EQU 指令为标签指定绝对（运行时）地址。

```
; Asm430 code ; A430 code
.label load_start load_start:
Run_start: <code>
<code> load_end:
Run_end: run_start: EQU 240H
.label load_end run_end: EQU run_start+load_end-load_start
```

(4) 尽管不是由绝对列表器生成的，但 ASEG 定义绝对段，而 EQU 可用于定义绝对符号。

```
MYFLAG EQU 23EH ; MYFLAG is located at 23E
ASEG 240H ; Absolute segment at 240
MAIN: MOV #23CH, SP ; MAIN is located at 240
...
```

8.3.11 Asm430 指令的字母顺序列表和交叉参考

Asm430 指令 (CCS)	A430 指令 (IAR)	Asm430 指令 (CCS)	A430 指令 (IAR)
.align	ALIGN	.loop	REPT
.asg	SET 或者 VAR 或者 ASSIGN	.macro	MACRO
.break	请参阅 节 8.3.7	.mexit	EXITM
.bss	请参阅 节 8.3.8	.mlib	请参阅 节 8.3.6
.byte 或 .string	DB	.mlist	LSTEXP+ (宏)
.cdecls	对 C 预处理器声明的固有支持。		LSTREP+ (环路阻止)
.copy 或 .include	#include or $	.mmsg	#message (XXXXXX)
.data	RSEG	.mnolist	LSTEXP- (宏)
.def	PUBLIC 或 EXPORT		LSTREP- (环路阻止)
.double	不支持。	.newblock	请参阅 节 8.3.8
.else	ELSE	.nolist	LSTOUT-
.elseif	ELSEIF	.option	请参阅 节 8.3.5
.emsg	#error	.page	PAGE
.end	END	.ref	EXTERN 或 IMPORT
.endif	ENDIF	.sect	RSEG
.endloop	ENDR	.setsect	请参阅 节 8.3.10
.endm	ENDM	.setsym	请参阅 节 8.3.10
.endstruct	请参阅 节 8.3.8	.space	DS
.equ 或 .set	EQU 或 =	.sslist	不支持
.eval	SET 或 VAR 或 ASSIGN	.ssnolist	不支持
.even	EVEN	.string	DB
.fclist	LSTCND-	.struct	请参阅 节 8.3.8
.fcnolist	LSTCND+	.tag	请参阅 节 8.3.8
.field	请参阅 节 8.3.4	.text	RSEG
.float	请参阅 节 8.3.4	.title	请参阅 节 8.3.5
.global	请参阅 节 8.3.6	.usect	请参阅 节 8.3.8
.if	IF	.width	COL
.label	请参阅 节 8.3.10	.wmsg	请参阅 节 8.3.10
.length	PAGSIZ	.word	DW
.list	LSTOUT+		

8.3.12 不受支持的 A430 指令 (IAR)

CCS Asm430 汇编器中不支持下述 IAR 汇编器指令：

条件汇编指令	宏指令	
REPTC(1)	LOCAL(2)	
REPTI		
文件参考指令	**其他指令**	**符号控制指令**
(名称) 或 (程序)	RADIX	DEFINE
(模块) 或 (库)	CASEON	SFRB
ENDMOD	CASEOFF	SFRW
列表控制指令	**C 类型预处理器指令(3)**	**符号控制指令**
LSTMAC (±)	#define	ASEG
LSTCOD (±)	#undef	RSEG
LSTPAG (±)	#if，#else，#elif	COMMON
LSTXREF (±)	#ifdef，#ifndef	STACK
	#endif	ORG
	#include	
	#error	

(1) CCS 中没有对 IAR REPTC 和 REPTI 指令的直接支持。然而，可以使用 CCS .macro 指令来实现同等的功能：

```
; IAR Assembler Example
REPTI zero,"R4","R5","R6"
MOV #0,zero
ENDR
; CCS Assembler Example
zero_regs .macro list
.var item
.loop
.break ($ismember(item, list) = 0)
MOV #0,item
.endloop
.endm
```

Code that is generated by calling "zero_regs R4,R5,R6":

```
MOV #0,R4
MOV #0,R5
MOV #0,R6
```

(2) 在 CCS 中，局部标签是通过使用 $n （其中 n=0...9 ）或者使用名称定义的吗？示例是 $4，$7，或者测试吗？

(3) 通过使用 .cdecls 间接支持 C 类型预处理器指令的使用。有关 .cdecls 指令的更多信息，请参阅 *MSP430 汇编语言工具用户指南*。

9 为 CCS 编写 可移植 C 语言代码和为 MSP430 编写 MSP430-GCC

TI CCS C 编译器的源代码和 MSP430 GCC 编译器的源代码并不完全兼容。标准的 ANSI/ISO C 代码在这些工具间可移植，但实现专用扩展有所不同，必须被移植。本附录描述了两个编译器之间的主要差异。

9.1 中断矢量定义

MSP430 的 GCC 不支持 CCS C 编译器所使用的 ISR 声明（使用 #pragma vector = ）的语法。然而，通过将不同的声明包含在预处理器指令中，可以为两个编译器编写正确的 C 语言代码：

```
#if defined(__TI_COMPILER_VERSION__) || defined(__IAR_SYSTEMS_ICC__)
#pragma vector=PORT1_VECTOR
__interrupt void Port1_ISR(void)
#elif defined(__GNUC__)
void __attribute__ ((interrupt(PORT1_VECTOR))) Port1_ISR (void)
#else
#error Compiler not supported!
#endif
```

有关使用 GCC 编译器的更多信息可在文档使用 GNU Compiler Collection（编译器集合）中找到。

10 FET 专用菜单

该附录说明了 FET 专用 CCS 菜单。

10.1 菜单

10.1.1 “Debug（调试）”视图：Run（运行）→ Free Run（自由运行）

调试器使用器件 JTAG 信号来调试器件。在某些 MSP430 器件上，这些 JTAG 信号与器件端口引脚共享。正常情况下，调试器将引脚保持在 JTAG 模式下，这样就可以调试器件。在此期间，共享引脚的端口功能不可用。

然而，当选择 Free Run（通过打开调试视图顶部的运行图标旁的下拉菜单进行选择）时，JTAG 驱动器被设置为 3 态，并且当 GO 被激活时此器件从 JTAG 控制（TEST 引脚被设置为 GND）中释放。任何有效的片上断点都被保留，而 JTAG 端口引脚恢复到其端口功能。

这时，调试器无法访问此器件,并且无法确定是否到达有效的断点（如果有的话）。必须手动控制调试器以停止此器件，此时确定器件的状态（也就是说，是否达到了断点？）。

请参阅常见问题解答 9。

10.1.2 Run（运行）→ Connect Target（连接目标器件）

勾选后重新获得对此器件的控制权。

10.1.3 Run → Advanced → Make Device Secure

在目标器件上熔断 JTAG 保险丝。保险丝熔断后，无法通过 JTAG 与器件继续通信。

10.1.4 Project → Properties → Debug → MSP430 Properties → Clock Control

当调试器控制器件时，禁用特定的系统时钟（在 STOP 或是断点之后。）在 GO 或者单个步骤之后，启用所有的系统时钟 (STEP 或 STEP INTO)。只有当调试程序不工作时才能更改。请参阅常见问题解答 12。

10.1.5 Window → Show View → Breakpoints

打开 MSP430 断点视图窗口。该窗口可用于设置基本断点和高级断点。可以通过访问属性（右键单击相应的断点）为每个断点单独选择高级设置，例如条件触发器 (Conditional Triggers) 和寄存器触发器 (Register Triggers)。通过打开断点 (Breakpoint) 下拉菜单（此菜单位于窗口顶部 Breakpoint 图标旁）来选择预定义断点，例如堆栈溢出中断 (Break on Stack Overflow)。可以通过在 Breakpoint View （断点视图）窗口中拖放断点来组合断点。当所有的断点条件都满足时，就会触发组合的断点。

10.1.6 Window → Show View → Other...Debug → Trace Control

跟踪视图 (Track View)允许使用状态存储模块。状态存储模块只存在于包含完整版的增强型仿真模块 (EEM) 的器件中（请参阅表 3-1）。断点被定义后，状态存储视图 (State Storage View) 会显示配置的跟踪信息。当点击窗口右上角的配置属性 (Configuration Properties)图标时，可以选择各种跟踪模式。关于 EEM 的详细信息可以在应用报告 *使用增强型仿真模块 (EEM) 与 Code Composer Studio（代码调试器）IDE 进行高级调试* 中找到。

10.1.7 Project → Properties → Debug → MSP430 Properties → Target Voltage

MSP-FET430UIF 的目标电压可以在 1.8V 至 3.6V 之间调节。此电压在 14 引脚目标连接器的引脚 2 上可用，以便从 USB FET 为目标供电。如果目标器件由外部供电，外部电源电压应该连接到目标连接器的引脚 4 上，以便 USB FET 能够相应地设置输出信号的电平。只有当调试程序不工作时才能更改。

11 器件专用菜单

11.1 MSP430L092

11.1.1 仿真模式

MSP430L092 可在两种不同的模式下运行：L092 模式和 C092 仿真模式。C092 仿真模式的目的是模拟高达 1920 代码字节的 C092，其最终目标是使用 L092 生成掩码。在运行调试程序之前，必须在 CCS 中设置工作模式。如图 11-1 中所示，在选择 MSP430L092 作为“Device Variant”（器件变量）之后，在底部的“Device Options”（器件选项）下的项目属性进行选择。图 11-2 显示了如何选择 C092 模式。

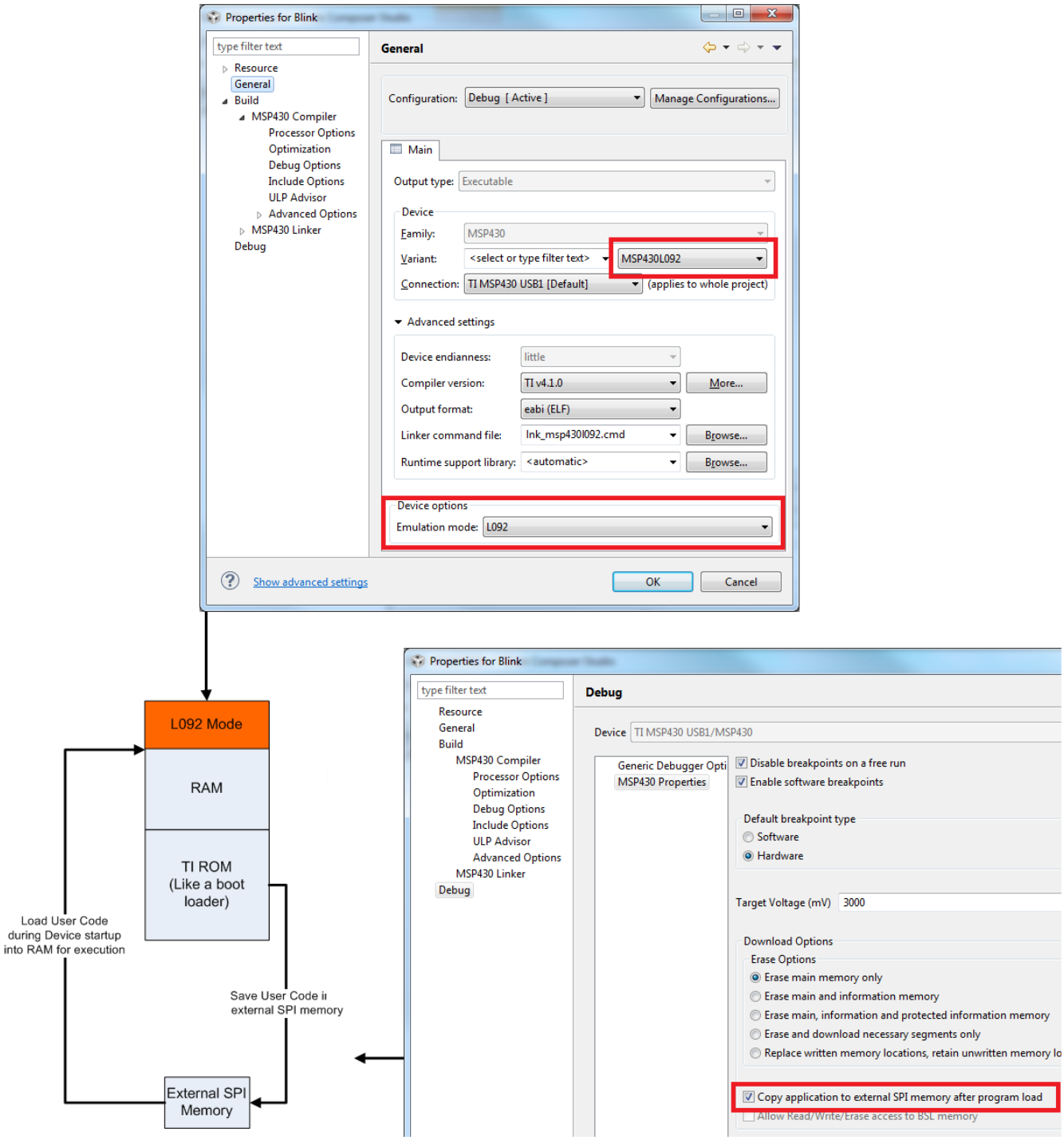

图 11-1. MSP430L092 模式

11.1.2 加载程序代码

MSP430L092 中的加载程序代码是 TI 公司生产的 ROM 代码，此代码提供了一系列的服务。其使得用户能够构建自主应用，而无需开发 ROM 掩码。此各应用由包含加载程序（例如，MSP430L092）的 MSP430 器件和 SPI 存储器件（例如，'95512 或者‘25640）组成。那些器件和相似器件可由不同的制造商提供。带有加载程序器件和外部 SPI 存储器（用于本地 0.9V 电源电压）的大多数应用用例都是后期开发，原型开发，和小批量试产可在 CCS Project Properties→ Debug → MSP430 Properties → Download Options → Copy application to external SPI memory after program load 中设置外部代码下载（请参阅图 11-1）。

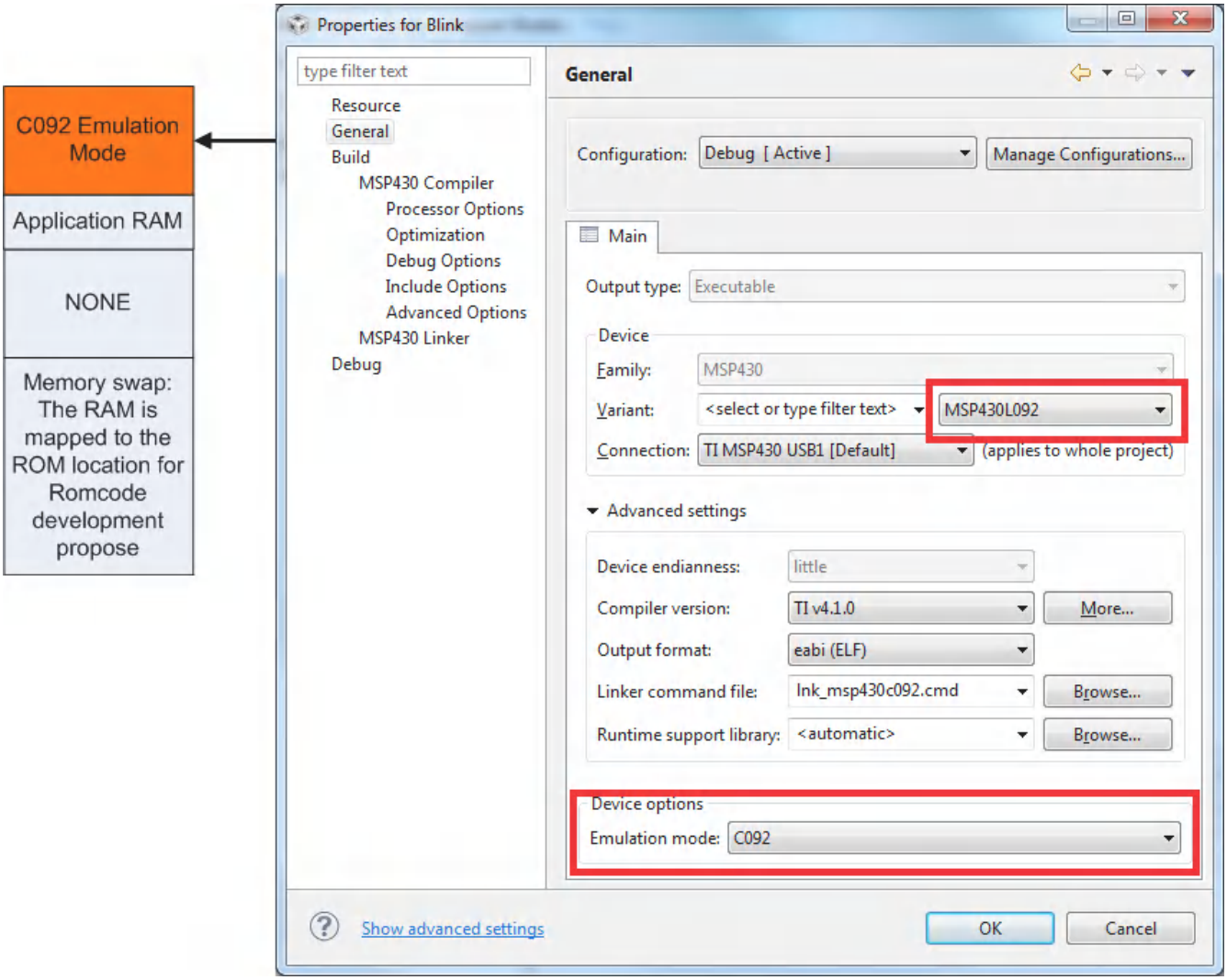

图 11-2. C092 仿真模式下的 MSP430L092。

11.1.3 C092 密码保护

MSP430C092 是一款用户专用 ROM 器件，其受密码保护。为了启动一个调试会话，必须向 CCS 提供密码。打开项目中的 MSP430C092.CCXML 文件，点击高级设置部分,高级目标配置中的目标配置选项。选择 MSP430 后，便会显示“CPU roperties”（CPU 属性）。图 11-3 显示了如何在 CCS v6.1 目标配置中提供 HEX 密码。

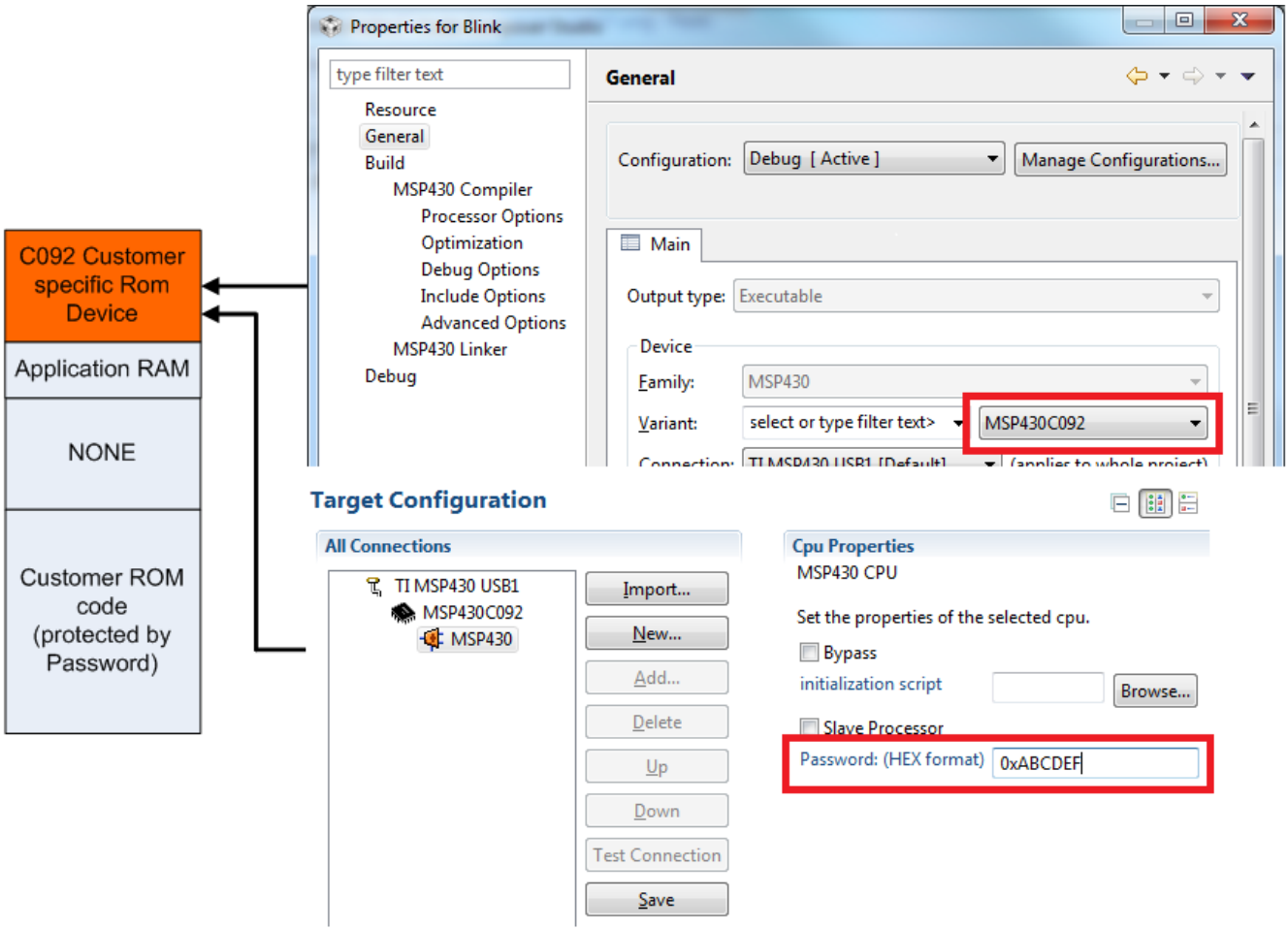

图 11-3. MSP430C092 密码访问

11.2 MSP430F5xx 和 MSP430F6xx BSL 支持

大多数 MSP430F5xx 和 MSP430F6xx 器件支持默认情况下受到保护的自定义 BSL。如需对自定义 BSL 进行编程，必须在 CCS Project Properties → Debug → MSP430 Properties → Download Options → Allow Read/Write/Erase access to BSL memory 中禁用该保护（请参阅图 11-4）。

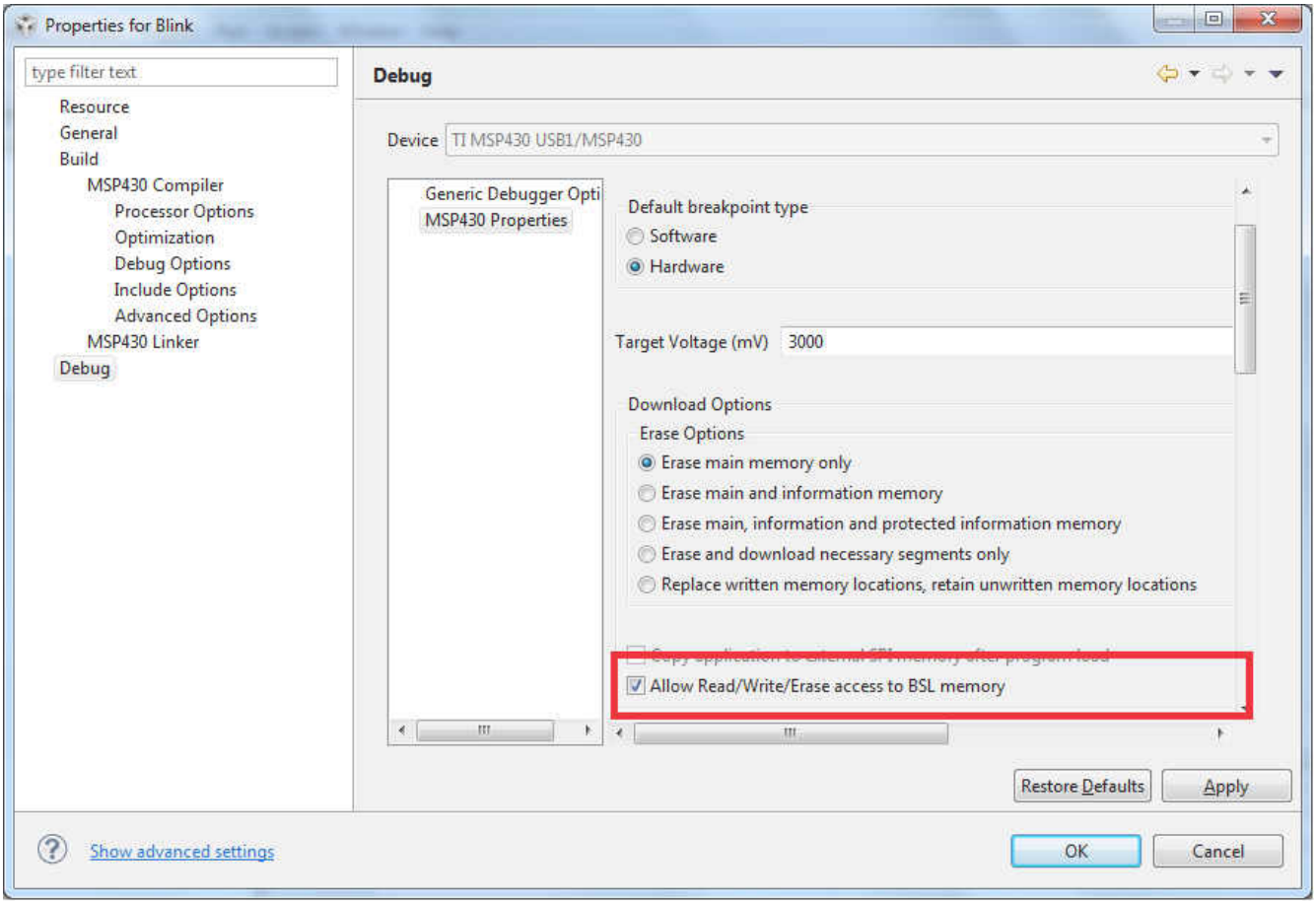

图 11-4. 允许对 BSL 的访问

11.3 MSP430FR5xx 和 MSP430FR6xx 密码保护

所选的 MSP430FR5xx 和 MSP430FR6xx 器件通过用户密码提供 JTAG 保护。在调试这些 MSP430 衍生器件时，必须提供十六进制的 JTAG 密码以启动调试会话。打开您项目中的 MSP430Fxxxx.CCXML 文件，点击 Advanced Setup section，Advanced Target Configuration 中的 Target Configurations。在选择了 MSP430 之后，CPU 属性变得清晰可见（请参阅图 11-5）。关于 MSP430FR5xx 和 MSP430FR6xx 器件的密码保护的详细信息可在器件用户指南中找到。

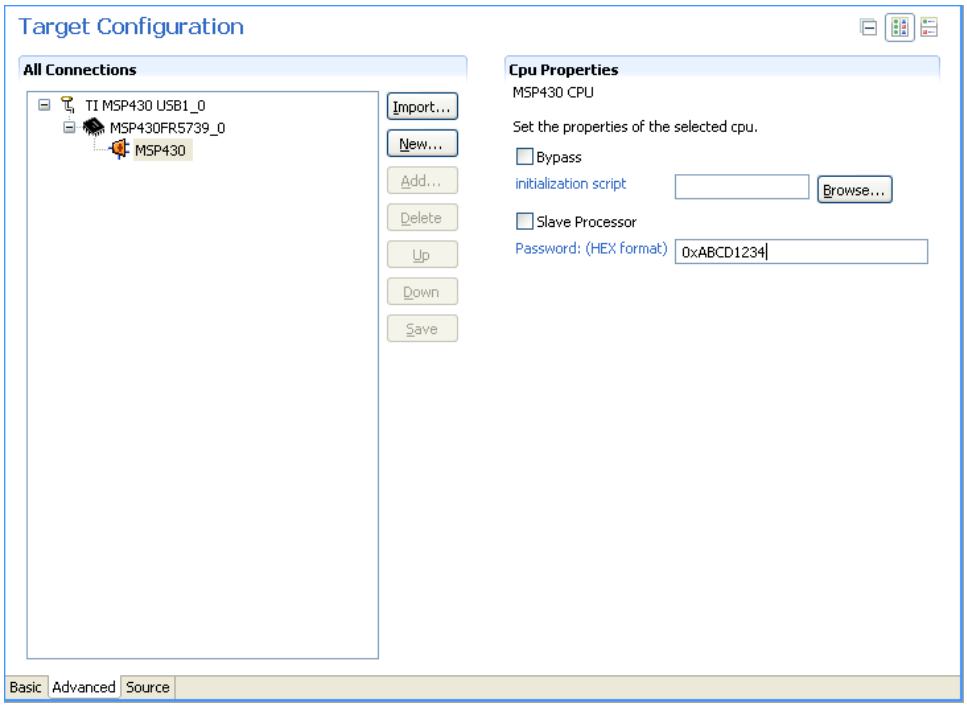

图 11-5. MSP430 密码访问

11.4 MSP430 超低功耗 LPMx.5 模式

11.4.1 LPMx.5 是什么

LPMx.5 是一种超低功耗模式，其进入和退出该模式的方式与其它低功耗模式不同。

LPMx.5 提供了器件上可用的最低功耗。为了实现这一点，进入 LPMx.5 模式后将禁用 PMM 模块的低压降稳压器 (LDO)，这将会从内核和器件的 JTAG 模块上移除电源电压。由于内核上的电源电压被移除，所有寄存器内容和 SRAM 内容都将丢失。从 LPMx.5 模式中退出会引起 BOR 事件，强制系统完全复位。

备注

有关 LPMx.5 和超低功耗调试模式的更多详细信息，请参阅相应的 MSP430 器件系列用户指南。

11.4.2 在支持超低功耗调试模式的 MSP430 器件上调试 LPMx.5 模式

为了启用超低功耗调试模式特性，必须通过点击“Project Properties -> Debug -> MSP430 Properties -> Enable Ultra Low Power debug / LPMx.5 debug”来启用“Enable Ultra Low Power debug（启用超低功耗调试）/ LPMx.5 debug（调试）”复选框（请参阅图 11-6）。

启用超低功耗调试模式时，每次目标器件进入和退出 LPMx.5 模式，均会在调试器日志中显示一条通知。

按下 Halt CCS 或 Reset CCS 按钮以将器件从 LPMx.5 唤醒。程序开始时暂停执行代码。在 LPMx.5 模式之前有效的所有断点都会自动恢复并重新激活。

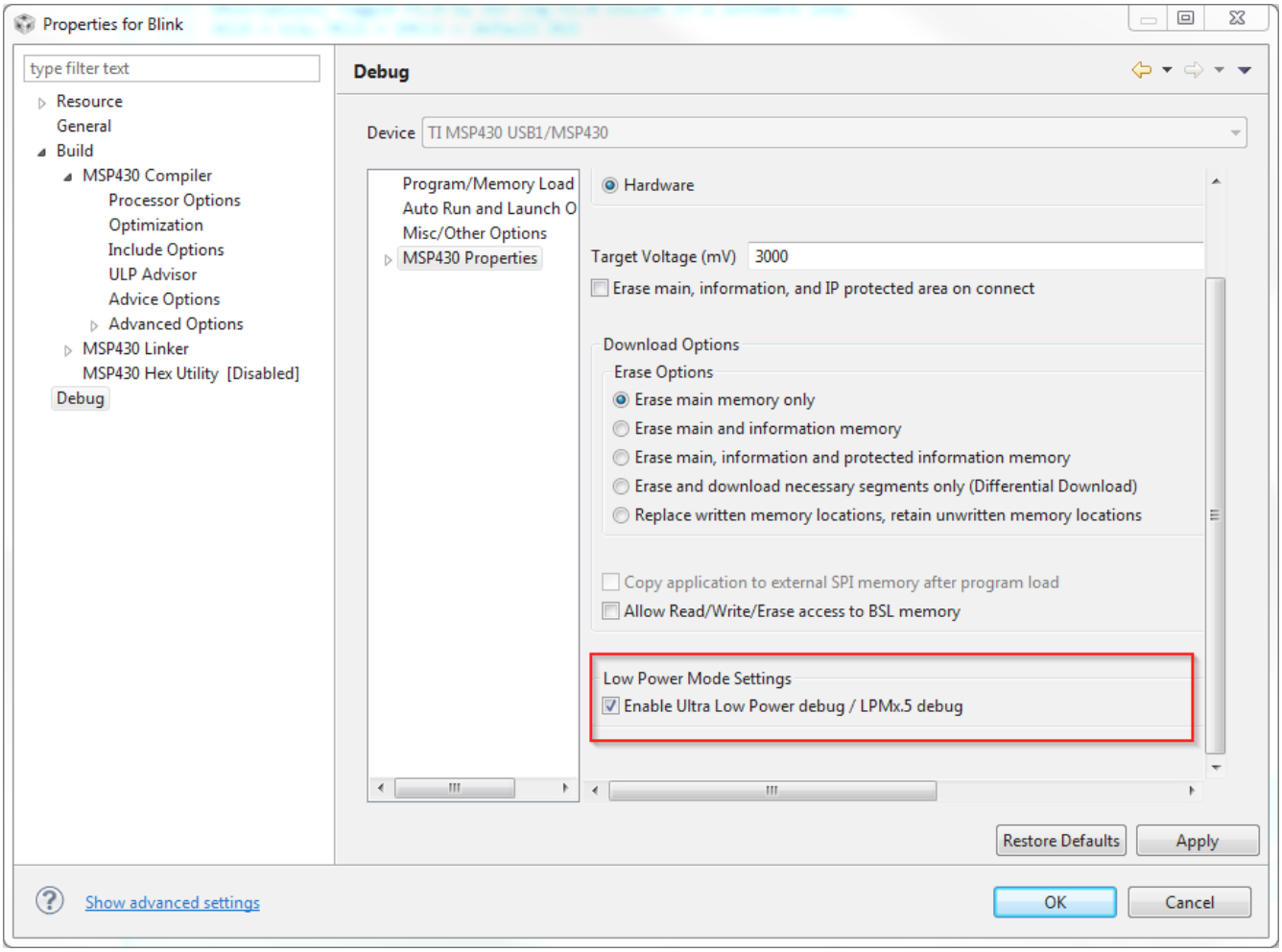

图 11-6. 启用超低功耗调试模式

11.4.2.1 限制

当目标器件处于 LPMx.5 模式时，不能设置或移除高级条件或者软件断点。但可以设置硬件断点。此外，只有在 LPMx.5 模式期间设置的硬件断点可以在 LPMx.5 模式中被移除。由于会引起器件复位，正在运行的目标不能与 LPMx.5 模式调试组合使用。

当将“Free Run”（自由运行）选项与 LPMx.5 模式结合使用时，目标器件在 LPMx.5 唤醒后不会恢复代码执行。在这种情况下，请点击“Suspend”（暂停）按钮来暂停调试会话，然后点击“Resume”（恢复）按钮来恢复会话。

11.4.3 在不支持超低功耗调试模式的 MSP430 器件上调试 LPMx.5 模式

在不支持超低功耗模式的 MSP430 器件上，可以使用“Free Run”(自由运行)选项来调试 LPMx.5 低功耗模式。此配置提供了 MSP430 LPMx.5 低功耗模式的绝对电流和能耗。

11.4.3.1 限制

使用此配置时存在一些限制：

1. 断点
   a. 当器件处于 LPMx.5 时，无法设置或擦除任何类型的断点。
2. 器件状态
   a. 没有关于当前器件状态的通知。从 IDE 的角度来看，器件正在运行。
3. 暂停
   a. 当器件处于 LPMx.5 模式时，暂停按钮可能无法可靠地工作。按下暂停按钮时，器件可能不会退出 LPMx.5 模式。在这种情况下，必须重新启动调试会话。在调试期间，一种选项是从 LPMx.5 唤醒后将器件置于活动模式，以便在器件处于 LPMx.5 之外的已知电源模式时能够可靠地暂停/恢复器件。
4. 调试程序连接
   a. 为了确保调试器始终可以连接 MSP430 器件并与其保持同步：
      i. 请勿在代码启动后直接进入 LPMx.5。代码启动和进入 LPMx.5 之间需要 500ms 的延迟，以确保调试程序可靠地同步。
      ii. 如果 4 线制 JTAG 显示连接和同步错误，请使用 2 线制 SBW 而非 4 线制 JTAG 协议。
      iii. 确保代码删除了所有 MSP430 端口引脚的锁定 I/O 设置。

12 修订历史记录

注：以前版本的页码可能与当前版本的页码不同

Changes from MAY 12, 2018 to MAY 4, 2020	Page
• 更改了文档标题并将所示示例的 Code Composer Studio (代码调试器)v8.x 更改为 v10.x	4
• 将 MSP-GANG430 编程器更改为 MSP-GANG 并删除了节 3.1.6 *如何生成二进制格式文件（TI-TXT 和 INTEL-HEX）* 中的 MSP-PRGS430	10
• 更改了图 3-3 *下载选项*	16
• 更改了节 7 *将 C 代码从 IAR 2.x、3.x、4.x、5.x、6.x 或 7.x 迁移到 CCS* 中迁移所用 IAR Embedded Workbench（嵌入式工作台）IDE 的版本	43
• 更改了节 8 *将汇编器代码从 IAR 2.x、3.x、4.x、5.x、6.x 或 7.x 迁移到 CCS* 中迁移所用的 IAR Embedded Workbench（嵌入式工作台）IDE 版本	48

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