

IAR Embedded Workbench® IDE Version 7+ for MSP430™ MCUs

User's Guide



Literature Number: SLAU138AS
JUNE 2004 – REVISED DECEMBER 2020

Read This First	5
How to Use This Manual	5
Information About Cautions and Warnings	5
Related Documentation From Texas Instruments	5
If You Need Assistance	6
Trademarks	6
1 Get Started Now!	7
1.1 Software Installation	7
1.2 Flashing the LED	7
2 Development Flow	11
2.1 Overview	11
2.2 Project Settings	12
2.3 Using Math Library for MSP430 (MSPMathlib) in IAR EW430 5.60.1 and Newer	13
2.4 Additional Project Settings for MSP430L092 and MSP430C092	13
2.4.1 MSP430L092 Loader Code	14
2.4.2 Password Protection of MSP430C092	14
2.5 Creating a Project From Scratch	15
2.6 Additional Project Settings for Ultra-Low-Power Mode (LPMx.5) Debugging	16
2.6.1 What is LPMx.5	16
2.6.2 Debugging LPMx.5 Mode on MSP430 Devices That Support the Ultra-Low-Power Debug Mode	16
2.6.3 Debugging LPMx.5 Mode on MSP430 Devices That Do Not Support the Ultra-Low-Power Debug Mode	17
2.7 Download Options for MSP430 Devices	18
2.8 Password Protection for MSP430 Devices	19
2.9 Stack Management and .xcl Files	20
2.10 How to Generate TI .TXT (and Other Format) Files	20
2.11 Overview of Example Programs	20
2.12 Using C-SPY	20
2.12.1 Breakpoint Types	20
2.12.2 Using Breakpoints	23
2.12.3 Using Single Step	23
2.12.4 Using Watch Windows	24
3 EnergyTrace™ Technology	25
3.1 Introduction	25
3.2 Energy Measurement	25
3.3 IAR Embedded Workbench® for MSP430 Integration	25
3.3.1 Debugging Devices With EnergyTrace++ Technology Support	26
3.3.2 Debugging Devices Without EnergyTrace++ Technology Support	31
3.4 Measuring Low-Power Currents	35
3.5 EnergyTrace Technology FAQs	36
4 MSP430 FRAM Memory Protection Mechanisms	39
4.1 Memory Protection Unit	39
4.2 Intellectual Property Encapsulation (IPE)	40
4.2.1 IPE Debug Settings	42
4.3 FRAM Write Protection (FRWP)	43
A Frequently Asked Questions (FAQs)	45
A.1 Hardware	45
A.2 Program Development (Assembler, C-Compiler, Linker)	45
A.3 Debugging (C-SPY)	47
B FET-Specific Menus	51
B.1 Menus	51

B.1.1 Emulator → Device Information.....	51
B.1.2 Emulator → Release JTAG on Go.....	51
B.1.3 Emulator → Resynchronize JTAG.....	51
B.1.4 Emulator → Init New Device.....	51
B.1.5 Emulator → Secure - Blow JTAG Fuse.....	51
B.1.6 Emulator → Breakpoint Usage.....	51
B.1.7 Emulator → Advanced → Clock Control.....	51
B.1.8 Emulator → Advanced → Emulation Mode.....	52
B.1.9 Emulator → Advanced → Memory Dump.....	52
B.1.10 Emulator → Advanced → Breakpoint Combiner.....	52
B.1.11 Emulator → State Storage Control.....	52
B.1.12 Emulator → State Storage Window.....	52
B.1.13 Emulator → Sequencer Control.....	52
B.1.14 Emulator → "Power on" Reset.....	52
B.1.15 Emulator → GIE on/off.....	52
B.1.16 Emulator → Leave Target Running.....	52
B.1.17 Emulator → Force Single Stepping.....	52
B.1.18	52
Revision History.....	53

List of Figures

Figure 1-1. Activate Project.....	8
Figure 1-2. Activate Project in Workspace Overview.....	9
Figure 2-1. L092 Mode.....	13
Figure 2-2. C092 Emulation Mode.....	14
Figure 2-3. C092 Password.....	14
Figure 2-4. Enable Ultra-Low-Power Debug Mode.....	16
Figure 2-5. LPMx.5 Notifications.....	17
Figure 2-6. Download Options.....	19
Figure 2-7. JTAG Password.....	19
Figure 3-1. Pulse Density and Current Flow.....	25
Figure 3-2. Debug Session With EnergyTrace++ Windows.....	26
Figure 3-3. Debug Options.....	27
Figure 3-4. Emulator Pulldown Menu With EnergyTrace++-Related Functions.....	28
Figure 3-5. Enabling the State Log Window.....	28
Figure 3-6. State Log Window With EnergyTrace++ Data.....	29
Figure 3-7. State Log Summary With EnergyTrace++ Data.....	29
Figure 3-8. Power Log Setup Window.....	29
Figure 3-9. Power Log Window With EnergyTrace++ Data.....	30
Figure 3-10. Timeline With Power Log and State Graphs Disabled.....	30
Figure 3-11. Timeline With EnergyTrace++ Data.....	31
Figure 3-12. Function Profiler With EnergyTrace++ Data.....	31
Figure 3-13. Debug Session With EnergyTrace Windows.....	32
Figure 3-14. Emulator Pulldown Menu With EnergyTrace-Related Functions.....	33
Figure 3-15. Power Log Setup Window.....	33
Figure 3-16. Power Log Window With EnergyTrace Data.....	34
Figure 3-17. Timeline With Power Log Graph Disabled.....	34
Figure 3-18. Timeline With EnergyTrace Data.....	35
Figure 3-19. LPM3 Current When Executing Under Debug Control.....	35
Figure 3-20. Release JTAG on Go Option in Emulator Pulldown Menu.....	36
Figure 3-21. LPM3 Current When Executing with JTAG Signals Released.....	36
Figure 4-1. MPU Configuration Dialog.....	40
Figure 4-2. IPE Configuration Dialog.....	41
Figure 4-3. IPE Debug Settings.....	42
Figure 4-4. FRWP Configuration Dialog.....	43

How to Use This Manual

Read and follow the instructions in [Chapter 1](#). This chapter provides instructions on installing the software, and describes how to run the demonstration programs. After you see how quick and easy it is to use the development tools, TI recommends that you read all of this manual.

This manual describes only the setup and basic operation of the software development environment, but it does not fully describe the MSP430 microcontrollers or the complete development software and hardware systems. For details of these items, see the appropriate TI and IAR™ documents listed in [Section Related Documentation From Texas Instruments](#).

This manual applies to the use of the TI MSP-FET, TI MSP-FET430UIF, and eZ430 development tools series.

These tools contain the most up-to-date materials available at the time of packaging. For the latest materials (including data sheets, user's guides, software, and application information), visit the [TI MSP430 website](#) or contact your local TI sales office.

Information About Cautions and Warnings

This book may contain cautions and warnings.

CAUTION

This is an example of a caution statement.

A caution statement describes a situation that could potentially damage your software or equipment.

WARNING

This is an example of a warning statement.

A warning statement describes a situation that could potentially cause harm to you.

The information in a caution or a warning is provided for your protection. Read each caution and warning carefully.

Related Documentation From Texas Instruments

MSP430 development tools documentation:

[MSP430 Hardware Tools User's Guide](#)

[eZ430-F2013 Development Tool User's Guide](#)

[eZ430-RF2480 User's Guide](#)

[eZ430-RF2500 Development Tool User's Guide](#)

[eZ430-RF2500-SEH Development Tool User's Guide](#)

[eZ430-Chronos Development Tool User's Guide](#)

[MSP Debugger's User's Guide](#)

MSP430 device documentation:

[MSP430F1xx Family User's Guide](#)

[MSP430F2xx Family User's Guide](#)

[MSP430F3xx Family User's Guide](#)

[MSP430F4xx Family User's Guide](#)

[MSP430F5xx and MSP430F6xx Family User's Guide](#)

[MSP430FR57xx Family User's Guide](#)

[MSP430FR58xx, MSP430FR59xx, and MSP430FR6xx User's Guide](#)

CC430 device documentation:

[CC430 Family User's Guide](#)

If You Need Assistance

Support for the MSP430 devices and the FET development tools is provided by the TI Product Information Center (PIC). Contact information for the PIC can be found on the [TI website](#). The [TI E2E™ Community](#) for the [MSP430](#) is available to provide open interaction with peer engineers, TI engineers, and other experts. Additional device-specific information can be found on the [MSP430 website](#).

Trademarks

MSP430™, E2E™, EnergyTrace™, EnergyTrace++™, and are trademarks of Texas Instruments.

Motorola™ is a trademark of Motorola Corporation.

ThinkPad™ is a trademark of IBM.

IAR Embedded Workbench® is a registered trademark of IAR Systems.

Intel® is a registered trademark of Intel Corporation.

All trademarks are the property of their respective owners.

This chapter provides instruction on installing the software, and shows how to run the demonstration programs.

1.1 Software Installation

Follow the instructions on the supplied READ ME FIRST document to install the IAR Embedded Workbench™. Read the file <Installation Root>\Embedded Workbenchx.x\common\doc\EW_QuickReference_LMS2.ENU.pdf from IAR for the latest information about the Workbench.

To install and run IAR Embedded Workbench, you need the following:

- A Pentium-compatible PC with Microsoft Windows Vista (SP2), Windows 7, Windows 8, Windows 8.1, or Windows 10. Both 32- and 64-bit variants of Windows are supported.
- Internet Explorer 7 or higher
- At least 2GB of RAM and 10GB of free disk space
- Adobe Acrobat Reader to access the product documentation

Note

If the MSP-FET or eZ-FET debugger driver install fails:

Under certain conditions (depending on the hardware and operating system that is used), the MSP-FET or eZ-FET driver install may fail on the first attempt. This can lead to unresponsive behavior on IDEs. To resolve this issue, disconnect the MSP-FET or eZ-FET and then plug it again, or plug it in to a different USB port, and restart the IDE.

1.2 Flashing the LED

This section demonstrates on the FET the equivalent of the C-language "Hello World!" introductory program. An application that flashes the LED is developed and downloaded to the FET, and then run.

1. Start the Workbench (**Start** → **Programs** → **IAR Systems** → **IAR Embedded Workbench for MSP430 Vxx** → **IAR Embedded Workbench**).
2. Click **File** → **Open Workspace** to open the file at: <Installation Root>\Embedded Workbench x.x\430\examples\Flashing the LED\Flashing the LED.eww. The workspace window opens.
3. Click on the tab at the bottom of the workspace window that corresponds to the MSP430 device (MSP430xxxx) and desired language (assembler or C) to set a project active (see [Figure 1-1](#)).

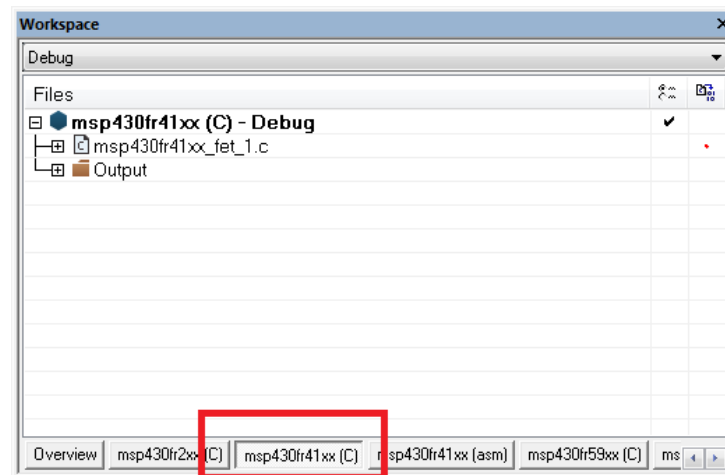


Figure 1-1. Activate Project

Alternatively, right-click to activate a project in the Workspace Overview tab (see [Figure 1-2](#)).

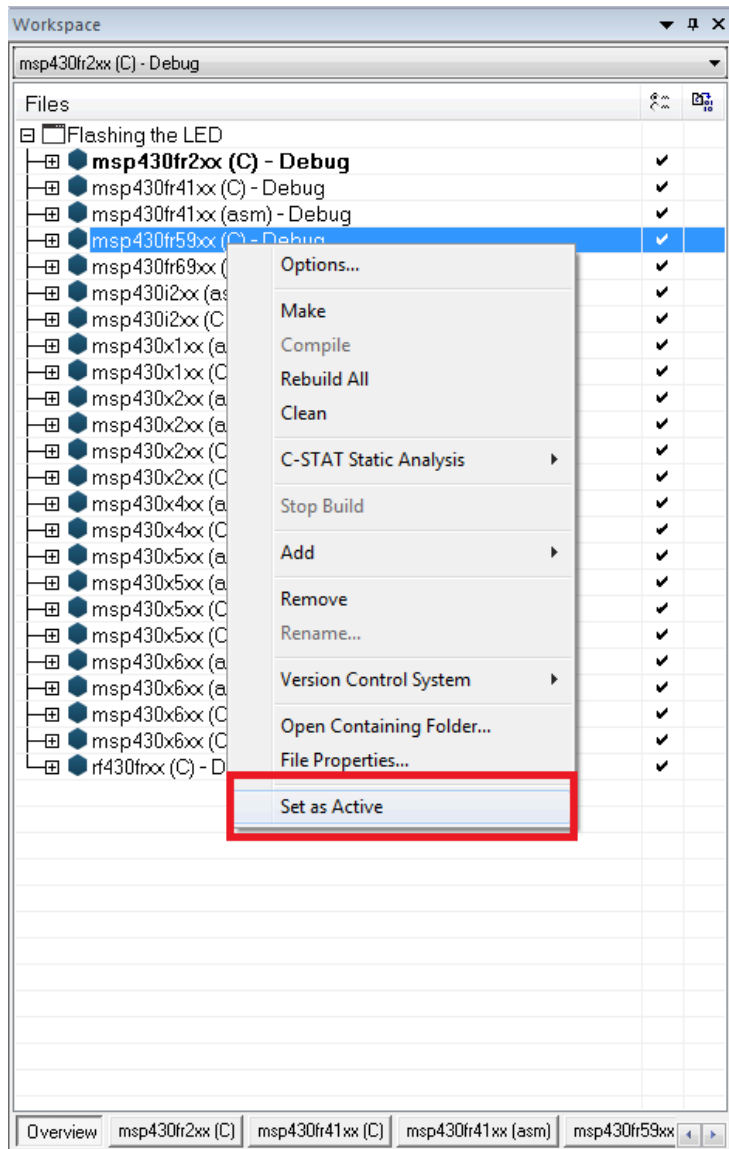


Figure 1-2. Activate Project in Workspace Overview

4. Click **Project** → **Options** → **Debugger** → **Setup** → **FET-Debugger**
5. Click **Project** → **Rebuild All** to build and link the source code. You can view the source code by double-clicking on the project, and then double-clicking on the displayed source file.
6. Click **Project** → **Download and Debug** (CTRL+D) to start the C-SPY debugger. C-SPY erases the device flash and then downloads the application object file to the device flash.

See FAQ 1 if C-SPY is unable to communicate with the device.

7. Click **Debug** → **Go** to start the application. The LED should flash.
8. Click **Debug** → **Stop Debugging** to stop debugging, to exit C-SPY, and to return to the Workbench.
9. Click **File** → **Exit** to exit the Workbench.

Congratulations, you have just built and tested an MSP430 application!

This page intentionally left blank.

This chapter describes how to use IAR EW430 to develop application software and how to use C-SPY to debug it.

2.1 Overview

Applications are developed in assembler or C using the Workbench, and they are debugged using C-SPY. C-SPY is seamlessly integrated into the Workbench. However, it is more convenient to make the distinction between the code development environment (Workbench) and the debugger (C-SPY). C-SPY can be configured to operate with the FET (that is, an actual MSP430 device) or with a software simulator of the device. IAR EW430 refers to the Workbench and C-SPY collectively.

Documentation for the MSP430 family and IAR EW430 is extensive. The [MSP430 home page](#) is another source of MSP430 information.

The components of IAR EW430 (IDE, debugger, assembler, compiler, linker) are fully documented in <Installation Root>\Embedded Workbench x.x\common\doc and <Installation Root>\Embedded Workbench \430\doc.

.htm files located throughout the EW430 directory tree contain the most up-to-date information and supplement the PDF files. In addition, EW430 documentation is available online through Help.

Tool	User's Guide
IDE	EW430_IDEGuide.pdf
Assembler	EW430_AssemblerReference.pdf
Compiler	EW430_CompilerReference.pdf
C library	clib.pdf
Linker and Librarian	xlink.ENU.pdf
Debugging	EW430_DebuggingGuide.pdf

2.2 Project Settings

The settings required to configure the Workbench and C-SPY are numerous and detailed. Read and thoroughly understand the documentation supplied by IAR when dealing with project settings. Review the project settings of the supplied assembler and C examples (the project settings are accessed using **Project** → **Options** with the project name selected). Use these project settings as templates when developing your own projects. Note that if the project name is not selected when settings are made, the settings are applied to the selected file (not to the project).

The following project settings are recommended or required:

- Specify the target device (**General Options** → **Target** → **Device**).
- Enable the generation of an executable output file (**General Options** → **Output** → **Output file** → **Executable**).
- To most easily debug a C project, disable optimization [**C/C++ Compiler** → **Optimizations** → **Size** → **None (Best debug support)**].
- Enable the generation of debug information in the compiler output (**C/C++ Compiler** → **Output** → **Generate debug information**).
- Specify the search path for the C preprocessor (**C/C++ Compiler** → **Preprocessor** → **Include Paths**).
- Enable the generation of debug information in the assembler output (**Assembler** → **Output** → **Generate Debug Info**).
- Specify the search path for the assembler preprocessor (**Assembler** → **Preprocessor** → **Include Paths**).
- To debug the project using C-SPY, specify a compatible format [**Linker** → **Output** → **Format** → **Debug information for C-SPY** (with runtime control modules or with I/O emulation modules)].
- Specify the search path for any used libraries (**Linker** → **Config** → **Search paths**).
- Specify the C-SPY driver. Select **Project** → **Options** → **Debugger** → **Setup** → **Driver** → **FET Debugger** to debug on the FET (that is, MSP430 device). Select Simulator to debug on the simulator. If FET Debugger is selected, use **Project** → **Options** → **FET Debugger** → **Setup** → **Connection** to select the appropriate port: Texas Instruments USB-IF for the USB Interface (MSP-FET430UIF) or for the eZ430.
- Enable the Device Description file. This file makes C-SPY "aware" of the specifics of the device it is debugging. This file corresponds to the specified target device (**Debugger** → **Setup** → **Device description file** → **Override default**).
- Enable the erasure of the Main and Information memories before object code download (**FET Debugger** → **Download** → **Erase main and Information memory**).
- To maximize system performance during debug, disable Virtual Breakpoints (**FET Debugger** → **Breakpoints** → **Use virtual breakpoints**) and disable all System Breakpoints (**FET Debugger** → **Breakpoints** → **System breakpoints on**).

Note

Use Factory Settings to quickly configure a project.

Use the Factory Settings button to quickly configure a project to a usable state.

The following steps can be used to quickly configure a project. Note that the General Options tab does not have a Factory Settings button.

1. Specify the target device (**General Options** → **Target** → **Device**).
2. Enable the generation of an executable output file (**General Options** → **Output** → **Output file** → **Executable**).
3. Accept the factory settings for the compiler (**C/C++ Compiler** → **Factory Settings**).
4. Accept the factory settings for the assembler (**Assembler** → **Factory Settings**).
5. Accept the factory settings for the linker (**Linker** → **Factory Settings**).
6. Accept the factory settings for C-SPY (**Debugger** → **Factory Settings**).

7. Debug on the hardware (**Debugger** → **Setup** → **Driver** → **FET Debugger**).
8. Specify the USB port (**FET Debugger** → **Setup** → **Connection** → **Texas Instruments USB-IF**).

Note

Avoid the use of absolute path names when referencing files.

Instead, use the relative pathname keywords \$TOOLKIT_DIR\$ and \$PROJ_DIR\$. See the IAR documentation for a description of these keywords. The use of relative path names permits projects to be moved easily, and projects do not require modification when IAR systems are upgraded (for example, from Limited or Baseline to Full).

2.3 Using Math Library for MSP430 (MSPMathlib) in IAR EW430 5.60.1 and Newer

TI's MSPMathlib is part of EW430 5.60.1 and newer releases. This optimized library provides up to 26x better performance in applications that use floating point scalar math. For details, see the [MSPMathlib web page](#).

MSPMathlib may be enabled for new and existing projects on all supported devices. Enable or disable MSPMathlib in the project options (**General Options** → **Library Configuration** → **Use TI MathLib**).

2.4 Additional Project Settings for MSP430L092 and MSP430C092

The MSP430L092 can operate in two different modes: L092 mode and C092 emulation mode. The purpose of the C092 emulation mode is to behave like a C092 with up to 1920 bytes of code at its final destination for mask generation.

The operation mode is determined by EW430 before starting the debugger. Two radio buttons are available for the mode selection. By default the L092 mode is selected (see [Figure 2-1](#) and [Figure 2-2](#)).

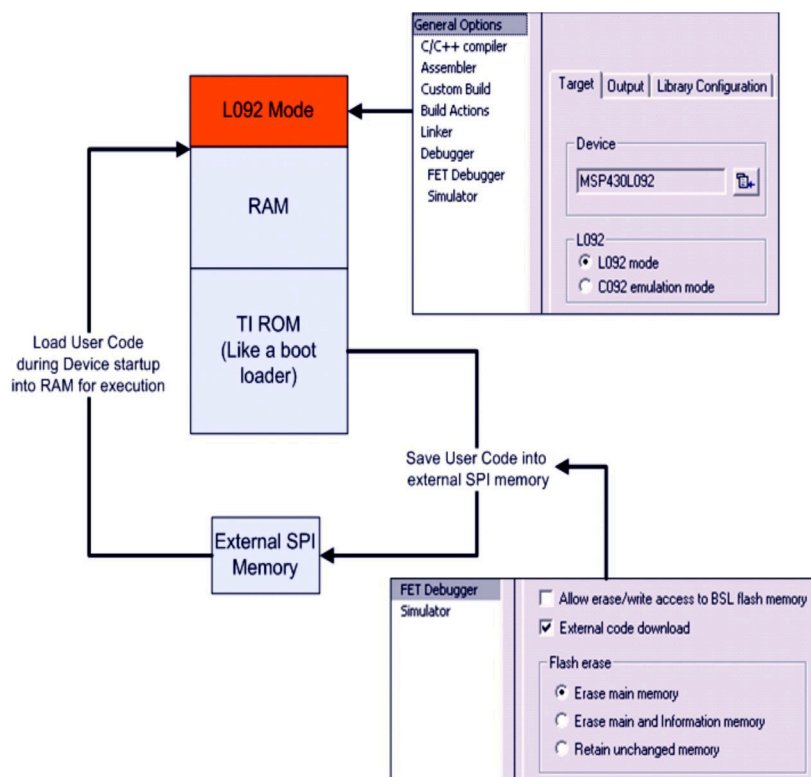


Figure 2-1. L092 Mode

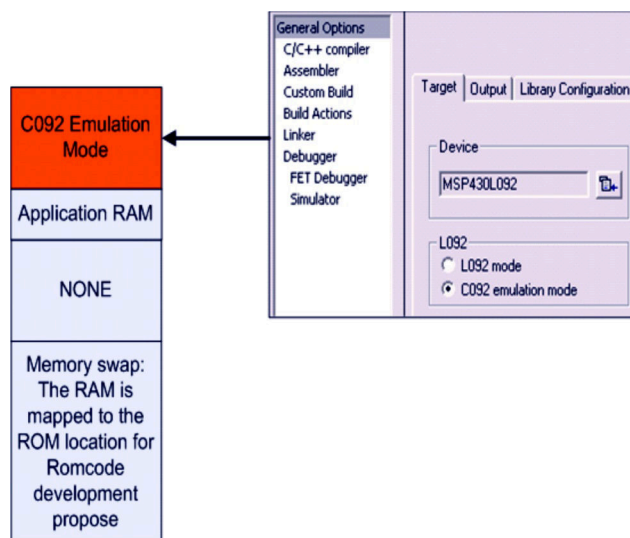


Figure 2-2. C092 Emulation Mode

2.4.1 MSP430L092 Loader Code

The Loader Code in the MSP430L092 is a ROM-code from TI that provides a series of services. It enables customers to build autonomous applications without needing to develop a ROM mask. Such an application consists of an MSP430 device containing the loader (for example, MSP430L092) and an SPI memory device (for example, '95512 or '25AA40); these and similar devices are available from various manufacturers.

Most use cases for an application with a loader device and external SPI memory for native 0.9-V supply voltage are late development, prototyping, and small series production.

Figure 2-1 shows the selection for loading the application into the external SPI memory.

2.4.2 Password Protection of MSP430C092

The MSP430C092 is a customer-specific ROM device that is protected by a password. To start a debug session, the password must be provided to EW430. Figure 2-3 shows how to provide a HEX password in EW430.

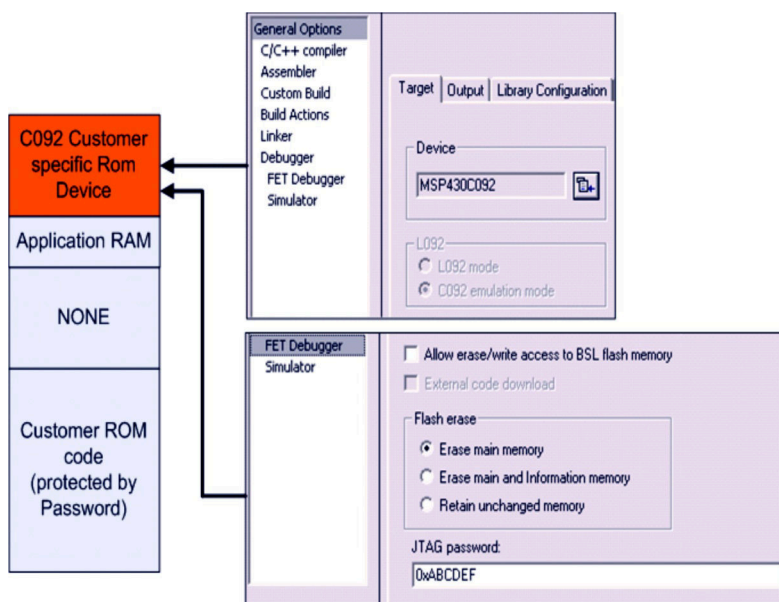


Figure 2-3. C092 Password

2.5 Creating a Project From Scratch

This section presents step-by-step instructions to create an assembler or C project from scratch, and to download and run the application on the MSP430 (see also [Section 2.2](#), Project Settings). The *MSP430 IAR Embedded Workbench IDE User's Guide* presents a more comprehensive overview of the process.

1. Start the Workbench (**Start** → **Programs** → **IAR Systems** → **IAR Embedded Workbench for MSP430** → **IAR Embedded Workbench**).
2. Create a new text file (**File** → **New** → **File**).
3. Enter the program text into the file.

Note

Use .h files to simplify your code development.

IAR EW430 is supplied with files that define the device registers and the bit names for each device. These files can greatly simplify the task of developing your program. The files are located in <Installation Root>\Embedded Workbench x.x\430\inc. Include the .h file corresponding to your target device in your text file (#include "msp430xyyy.h"). Additionally, files io430xxxx.h are provided and are optimized to be included by C source files.

4. Save the program text file (**File** → **Save**).

It is recommended that assembler text files be saved with a file-type suffix of ".s43" and that C text files be saved with a file-type suffix of ".c".

5. Create a new workspace (**File** → **New** → **Workspace**).
6. Create a new project (**Project** → **Create New Project**). Select Tool chain: MSP430, Project Templates: Empty project and click OK. Specify a project name and click Save.
7. Add the program text file to the project (**Project** → **Add Files**). Select the program text file and click Open. Alternatively, double-click on the file to add it to the project.

Note

How to add assembler source files to your project

The default file type presented in the Add Files window is **C/C++ Files**. To view assembler files (.s43), select **Assembler Files** in the **Files of type** drop-down menu.

8. Save the workspace (**File** → **Save Workspace**). Specify a workspace name and click **Save**.
9. Configure the project options (**Project** → **Options**). For each of the subcategories (**General Options**, **C/C++ Compiler**, **Assembler**, **Linker**, **Debugger**), accept the default Factory Settings with the following exceptions:
 - Specify the target device (**General Options** → **Target** → **Device**).
 - Enable the generation of an executable output file (**General Options** → **Output** → **Output file** → **Executable**).
 - To debug on the FET (that is, the MSP430), click **Debugger** → **Setup** → **Driver** → **FET Debugger**.
 - Specify the active port used to interface to the FET (**FET Debugger** → **Setup** → **Connection**).
10. Build the project (**Project** → **Rebuild All**).
11. Debug the application using C-SPY (**Project** → **Debug**). This starts C-SPY, and C-SPY takes control of the target, erases the target memory, programs the target memory with the application, and resets the target.

See [FAQ 1](#) if C-SPY is unable to communicate with the device.
12. Click **Debug** → **Go** to start the application.
13. Click **Debug** → **Stop Debugging** to stop the application, to exit C-SPY, and to return to the Workbench.
14. Click **File** → **Exit** to exit the Workbench.

2.6 Additional Project Settings for Ultra-Low-Power Mode (LPMx.5) Debugging

2.6.1 What is LPMx.5

LPMx.5 is an ultra-low-power mode in which the entry and exit is handled differently than the other low-power modes.

LPMx.5 gives the lowest power consumption available on a device. To achieve this, entry to LPMx.5 disables the LDO of the PMM module, which removes the supply voltage from the core and the JTAG module of the device. Because the supply voltage is removed from the core, all register contents and SRAM contents are lost. Exit from LPMx.5 causes a BOR event, which forces a complete reset of the system.

2.6.2 Debugging LPMx.5 Mode on MSP430 Devices That Support the Ultra-Low-Power Debug Mode

To enable the ultra-low-power debug mode feature, enable the **Enable ULP / LPMx.5 debug** checkbox by clicking **FET Debugger** → **Setup** → **Enable ULP / LPMx.5 debug** (see [Figure 2-4](#)). When the ultra-low power debug mode is enabled a notification is displayed in the Debugger log every time the target device enters and leaves LPMx.5 mode (see [Figure 2-5](#)).

Press the Halt or Reset button in Embedded Workbench to wake up the target device from LPMx.5. Execution of the code is halted at the start of the program. All breakpoints that had been active before LPMx.5 are restored and reactivated automatically.

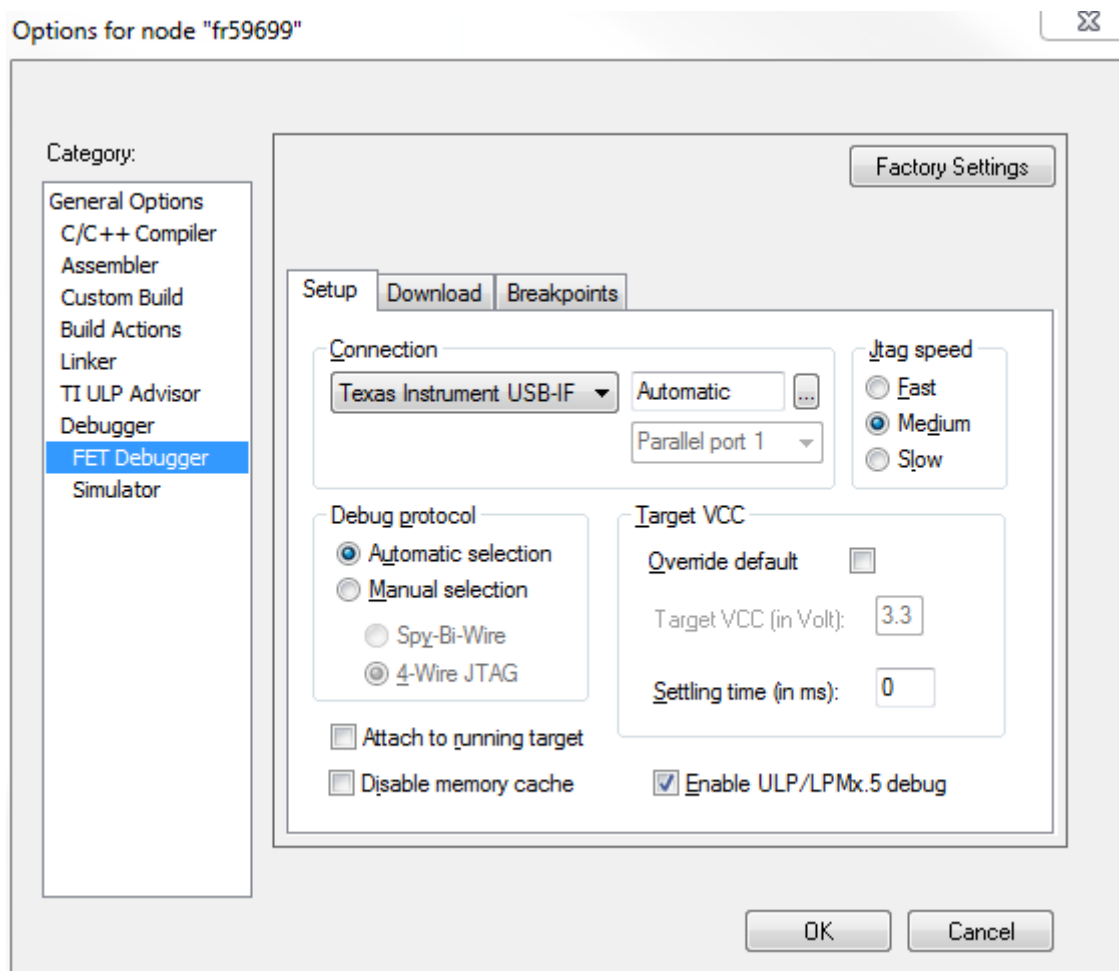


Figure 2-4. Enable Ultra-Low-Power Debug Mode

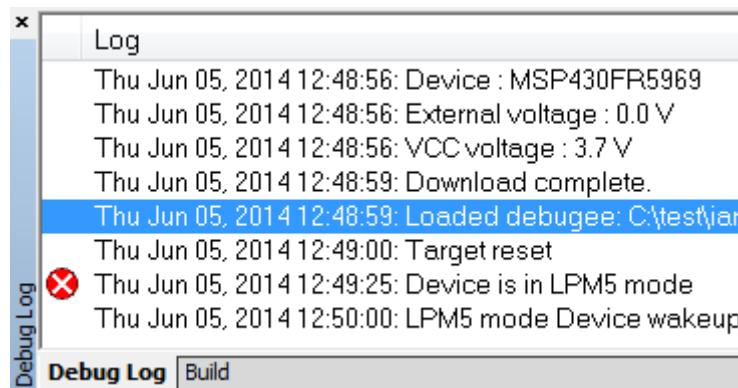


Figure 2-5. LPMx.5 Notifications

2.6.2.1 Limitations

When a target device is in LPMx.5 mode, it is not possible to set or remove advanced conditional or software breakpoints. It is, however, possible to set hardware breakpoints. Only hardware breakpoints that were set during LPMx.5 can be removed in the LPMx.5 mode. Attach to running target is not possible in combination with LPMx.5 mode debugging, as this results in a device reset.

2.6.3 Debugging LPMx.5 Mode on MSP430 Devices That Do Not Support the Ultra-Low-Power Debug Mode

On MSP430 devices that do not support the ultra-low-power mode, the LPMx.5 low-power mode can be debugged using the RELEASE JTAG ON GO option. This configuration provides the absolute current and energy consumption of MSP430 LPMx.5 low-power mode.

2.6.3.1 Limitations

Using this configuration presents some limitations:

1. Breakpoint
 - a. Setting or erasing any kind of breakpoint is not possible when the device is in LPMx.5.
2. Device State
 - a. There are no notifications about the current device state. From the perspective of the IDE, the device is running.
3. Pause
 - a. The pause button might not work reliably when the device is in LPMx.5 mode. The device might not leave LPMx.5 mode when the pause button is pressed. In this case, the debug session must be restarted. During debugging, an option is to trap the device in active mode after wake-up from LPMx.5, so that the device can be paused or suspend reliably when it is in a known power mode other than LPMx.5.
4. Debugger connection
 - a. To make sure that the debugger can always connect and synchronize to the MSP430 device.
 - i. Do not enter LPMx.5 directly after code start. A 500-ms delay is required between code start and LPMx.5 entry to ensure reliable debugger synchronization.
 - ii. If 4-wire JTAG shows connection and synchronization errors, use 2-wire SBW instead of the 4-wire JTAG protocol.
 - iii. Make sure that the code removes the lock I/O setting for all MSP430 port pins.
5. Release JTAG on Go
 - a. When using the "Release JTAG on Go" option in combination with LPMx.5 mode, the target device does not resume code execution after LPMx.5 wakeup. In this case, suspend the debug session by clicking the "Halt" button, and then resume the session by clicking the "Go" button.

2.7 Download Options for MSP430 Devices

By default, C-SPY downloads the application to RAM or flash when a debug session starts. The Download options (see [Figure 2-6](#)) let you modify the behavior of the download.

- **Verify download**
Verifies that the downloaded code image can be read back from target memory with the correct contents.
- **Allow erase/write access to locked flash memory**
Enables erase/write access to Info Segment A. This option can only be used with devices that have the Info Segment A memory.
- **Allow erase/write access to BSL flash memory**
Enables erase/write access to BSL flash memory. This option can only be used with MSP430F5xx devices that have BSL flash memory.
- **External code download**
Saves user code to external SPI memory.
- **Erase main memory**
Erases only the main flash memory before download. The Information memory is not erased.
- **Erase main and Information memory**
Erases both flash memories—main and Information memory—before download.
- **Retain unchanged memory**
Reads the main and Information memories into a buffer. Only the flash segments that are needed are erased.
- **Compare with image on target**
Compares the data that is to be written into a segment with the image on the target. If the data matches the image, the data on the target is left as is, and nothing is downloaded. The new data effectively replaces the old data, and unchanged old data is retained.
- **Compare with image cached on PC**
Compares the data that is to be written into a segment with the image that is cached on the host computer.
- **Erase main and Information memory inc. IP PROTECTED area**
Erases the main and Information flash memories, including the IP protected area before download.
- **JTAG password**
If a JTAG device is password-protected, supply the needed password here.

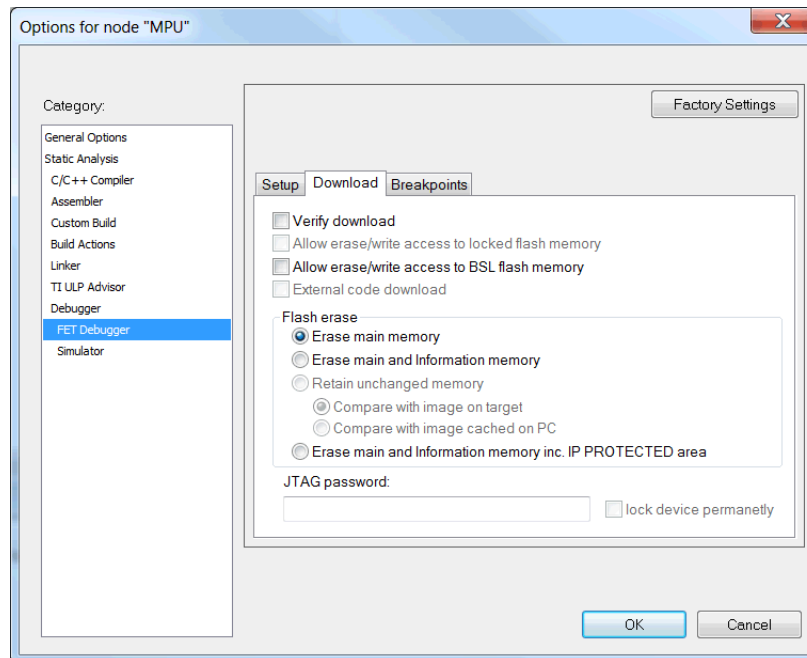


Figure 2-6. Download Options

2.8 Password Protection for MSP430 Devices

When debugging an MSP430 device that supports protection by a user password, the hexadecimal JTAG password must be provided to start a debug session.

Set JTAG password by clicking **FET Debugger** → **Download** → **JTAG password** (see [Figure 2-7](#)).

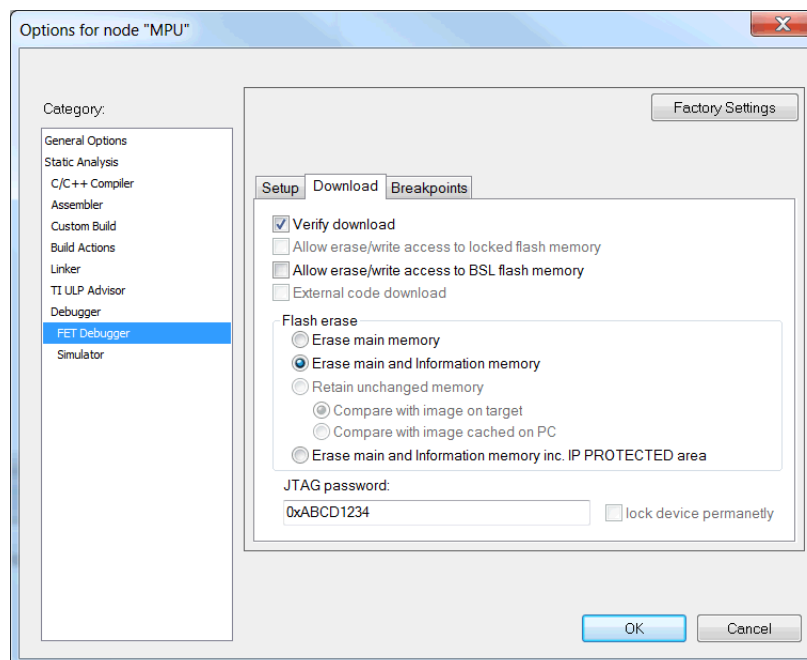


Figure 2-7. JTAG Password

2.9 Stack Management and .xcl Files

The reserved stack size can be configured through either the project options dialog (**General Options** → **Stack/Heap**) or through direct modification of the .xcl linker control files. These files are input to the linker and contain statements that control the allocation of device memory (RAM, flash). See the IAR XLINK documentation for a complete description of these files. The .xcl files provided with the IAR EW430 (<Installation Root>\Embedded Workbench x.x\430\config\lnk430xxxx.xcl) define a relocatable segment (RSEG) called CSTACK. CSTACK is used to define the region of RAM that is used for the system stack within C programs. CSTACK can also be used in assembler programs (`MOV.W #SFE(CSTACK), SP`). CSTACK is defined to extend from the last location of RAM for 50 bytes (that is, the stack extends downward through RAM for 50 bytes).

Other statements in the .xcl file define other relocatable regions that are allocated from the first location of RAM to the bottom of the stack.

Note

- The supplied .xcl files reserve 50 bytes of RAM for the stack, whether or not this amount of stack is actually required (or if it is sufficient).
- There is no runtime checking of the stack. The stack can overflow the 50 reserved bytes and possibly overwrite the other segments. No error is output.

The supplied .xcl files can be modified to tune the size of the stack to the needs of the application; edit - `D_STACK_SIZE=xx` to allocate xx bytes for the stack. The .xcl file also reserves 50 bytes for the heap if required [for example, by `malloc()`].

2.10 How to Generate TI .TXT (and Other Format) Files

The linker can be configured to output objects in TI .TXT format for use with the MSP-GANG programmers. Click **Project** → **Options** → **Linker** → **Output** → **Format** → **Other** → **msp430-txt**. Intel® and Motorola™ formats also can be selected.

For more information, see FAQ 6 in [Appendix A](#).

2.11 Overview of Example Programs

Example programs for MSP430 devices are provided in <Installation Root>\Embedded Workbench x.x\430\examples. Each tool folder contains folders that contain the assembler and C sources.

<Installation Root>\Embedded Workbench\430\examples\Flashing the LED\Flashing the LED.eww conveniently organizes the FET_1 demonstration code into a workspace. The workspace contains assembler and C projects of the code for each of the MSP430 device families. Debug and Release versions are provided for each of the projects.

Additional code examples can be found on the [MSP430 home page](#) under Code Examples. Note that some example programs require a 32-kHz crystal on LFXT1, and not all FETs are supplied with a 32-kHz crystal.

2.12 Using C-SPY

See [Appendix B](#) for a description of FET-specific menus within C-SPY.

2.12.1 Breakpoint Types

The C-SPY breakpoint mechanism uses a limited number of on-chip debugging resources (specifically, N breakpoint registers, see [Table 2-1](#)). When N or fewer breakpoints are set, the application runs at full device speed (or realtime). When greater than N breakpoints are set and Use Virtual Breakpoints is enabled (**FET Debugger** → **Breakpoints** → **Use virtual breakpoints**), the application runs under the control of the host PC; the system operates at a much slower speed but offers unlimited software breakpoints (or non-realtime). During non-realtime mode, the PC, in effect, repeatedly single steps the device and interrogates the device after each operation to determine if a breakpoint has been hit.

Both (code) address and data (value) breakpoints are supported. Data breakpoints and range breakpoints each require two MSP430 hardware breakpoints.

Table 2-1. Device Architecture, Breakpoints, and Other Emulation Features

Device	MSP430 Architecture	4-Wire JTAG	2-Wire JTAG ⁽¹⁾	Break-points (N)	Range Break-points	Clock Control	State Sequencer	Trace Buffer	LPMx.5 Debugging Support
CC430F512x	MSP430Xv2	X	X	2	X	X			X ⁽²⁾
CC430F513x	MSP430Xv2	X	X	2	X	X			
CC430F514x	MSP430Xv2	X	X	2	X	X			X ⁽²⁾
CC430F612x	MSP430Xv2	X	X	2	X	X			
CC430F613x	MSP430Xv2	X	X	2	X	X			
CC430F614x	MSP430Xv2	X	X	2	X	X			X ⁽²⁾
MSP430AFE2xx	MSP430	X	X	2		X			
MSP430BT5190	MSP430Xv2	X	X	8	X	X	X	X	
MSP430F11x1	MSP430	X		2					
MSP430F11x2	MSP430	X		2					
MSP430F12x	MSP430	X		2					
MSP430F12x2	MSP430	X		2					
MSP430F13x	MSP430	X		3	X				
MSP430F14x	MSP430	X		3	X				
MSP430F15x	MSP430	X		8	X	X	X	X	
MSP430F161x	MSP430	X		8	X	X	X	X	
MSP430F16x	MSP430	X		8	X	X	X	X	
MSP430F20xx	MSP430	X	X	2		X			
MSP430F21x1	MSP430	X		2		X			
MSP430F21x2	MSP430	X	X	2		X			
MSP430F22x2	MSP430	X	X	2		X			
MSP430F22x4	MSP430	X	X	2		X			
MSP430F23x	MSP430	X		3	X	X			
MSP430F23x0	MSP430	X		2		X			
MSP430F2410	MSP430	X		3	X	X			
MSP430F241x	MSP430X	X		8	X	X	X	X	
MSP430F24x	MSP430	X		3	X	X			
MSP430F261x	MSP430X	X		8	X	X	X	X	
MSP430F41x	MSP430	X		2		X			
MSP430F41x2	MSP430	X	X	2		X			
MSP430F42x	MSP430	X		2		X			
MSP430F42x0	MSP430	X		2		X			
MSP430F43x	MSP430	X		8	X	X	X	X	
MSP430F43x1	MSP430	X		2		X			
MSP430F44x	MSP430	X		8	X	X	X	X	
MSP430F44x1	MSP430	X		8	X	X	X	X	
MSP430F461x	MSP430X	X		8	X	X	X	X	
MSP430F461x1	MSP430X	X		8	X	X	X	X	
MSP430F471xx	MSP430X	X		8	X	X	X	X	
MSP430F47x	MSP430	X		2		X			
MSP430F47x3	MSP430	X		2		X			
MSP430F47x4	MSP430	X		2		X			
MSP430F51x1	MSP430Xv2	X	X	3	X	X			
MSP430F51x2	MSP430Xv2	X	X	3	X	X			

Table 2-1. Device Architecture, Breakpoints, and Other Emulation Features (continued)

Device	MSP430 Architecture	4-Wire JTAG	2-Wire JTAG ⁽¹⁾	Break-points (N)	Range Break-points	Clock Control	State Sequencer	Trace Buffer	LPMx.5 Debugging Support
MSP430F52xx	MSP430Xv2	X	X	3	X	X			
MSP430F530x	MSP430Xv2	X	X	3	X	X			
MSP430F5310	MSP430Xv2	X	X	3	X	X			
MSP430F532x	MSP430Xv2	X	X	8	X	X	X	X	
MSP430F534x	MSP430Xv2	X	X	8	X	X	X	X	
MSP430F535x	MSP430Xv2	X	X	8	X	X	X	X	
MSP430F543x	MSP430Xv2	X	X	8	X	X	X	X	
MSP430F54xx	MSP430Xv2	X	X	8	X	X	X	X	
MSP430F54xxA	MSP430Xv2	X	X	8	X	X	X	X	
MSP430F550x	MSP430Xv2	X	X	3	X	X			
MSP430F5510	MSP430Xv2	X	X	3	X	X			
MSP430F552x	MSP430Xv2	X	X	8	X	X	X	X	
MSP430F563x	MSP430Xv2	X	X	8	X	X	X	X	
MSP430F565x	MSP430Xv2	X	X	8	X	X	X	X	
MSP430F643x	MSP430Xv2	X	X	8	X	X	X	X	
MSP430F645x	MSP430Xv2	X	X	8	X	X	X	X	
MSP430F663x	MSP430Xv2	X	X	8	X	X	X	X	
MSP430F665x	MSP430Xv2	X	X	8	X	X	X	X	
MSP430F67xx	MSP430Xv2	X	X	3	X	X			
MSP430F67xx1	MSP430Xv2	X	X	3	X	X			
MSP430F67xx1A	MSP430Xv2	X	X	3	X	X			
MSP430F67xxA	MSP430Xv2	X	X	3	X	X			
MSP430FE42x	MSP430	X		2		X			
MSP430FE42x2	MSP430	X		2		X			
MSP430FG42x0	MSP430	X		2		X			
MSP430FG43x	MSP430	X		2		X			
MSP430FG461x	MSP430X	X		8	X	X	X	X	
MSP430FG47x	MSP430	X		2		X			
MSP430FG642x	MSP430Xv2	X	X	8	X	X	X	X	
MSP430FG662x	MSP430Xv2	X	X	8	X	X	X	X	X ⁽²⁾
MSP430FR20xx	MSP430Xv2	X	X	3	X	X			
MSP430FR21xx	MSP430Xv2	X	X	3	X	X			
MSP430FR23xx	MSP430Xv2	X	X	3	X	X			
MSP430FR24xx	MSP430Xv2	X	X	3	X	X			
MSP430FR25xx	MSP430Xv2	X	X	3	X	X			
MSP430FR26xx	MSP430Xv2	X	X	3	X	X			
MSP430FR41xx	MSP430Xv2	X	X	3	X	X			
MSP430FR57xx	MSP430Xv2	X	X	3	X	X			
MSP430FR58xx	MSP430Xv2	X	X	3	X	X			X
MSP430FR59xx	MSP430Xv2	X	X	3	X	X			X
MSP430FR60xx	MSP430Xv2	X	X	3	X	X			X
MSP430FR68xx	MSP430Xv2	X	X	3	X	X			X
MSP430FR69xx	MSP430Xv2	X	X	3	X	X			X
MSP430FW42x	MSP430	X		2		X			

Table 2-1. Device Architecture, Breakpoints, and Other Emulation Features (continued)

Device	MSP430 Architecture	4-Wire JTAG	2-Wire JTAG ⁽¹⁾	Break-points (N)	Range Break-points	Clock Control	State Sequencer	Trace Buffer	LPMx.5 Debugging Support
MSP430G22xx	MSP430		X	2		X			
MSP430G2xxx	MSP430	X	X	2		X			
MSP430i20xx	MSP430	X	X	2		X			
MSP430L092	MSP430Xv2	X		2		X			
MSP430SL5438A	MSP430Xv2	X	X	8	X	X	X	X	
MSP430TCH5E	MSP430	X	X	2		X			
RF430FRL15xH	MSP430Xv2	X	X	2		X			

- (1) The 2-wire JTAG debug interface is also referred to as Spy-Bi-Wire (SBW) interface. Note that this interface is supported only by the USB emulators (eZ430-xxxx and MSP-FET430UIF USB JTAG emulator) and the MSP-GANG430 production programming tool. The MSP-FET430PIF parallel port JTAG emulator does not support communication in 2-wire JTAG mode.
- (2) Support is limited to Spy-Bi-Wire (SBW) on MSP-FET430UIF. No limitations on MSP-FET.

2.12.2 Using Breakpoints

If C-SPY is started with greater than N breakpoints set and virtual breakpoints are disabled, a message is output to inform the user that only N (real-time) breakpoints are enabled (and one or more breakpoints are disabled). Note that the workbench permits any number of breakpoints to be set, regardless of the Use Virtual Breakpoints setting of C-SPY. If virtual breakpoints are disabled, a maximum of N breakpoints can be set within C-SPY.

Resetting a program temporarily requires a breakpoint if **Project** → **Options** → **Debugger** → **Setup** → **Run To** is enabled (see FAQ 32).

The Run To Cursor operation temporarily requires a breakpoint. Consequently, only (N – 1) breakpoints can be active when Run To Cursor is used if virtual breakpoints are disabled (see FAQ 33).

If, while processing a breakpoint, an interrupt becomes active, C-SPY stops at the first instruction of the interrupt service routine (see FAQ 26).

Note

Do not set a breakpoint on a RETI instruction if the previous instruction modifies the stack pointer. Program execution will not work properly after reaching the break point.

2.12.3 Using Single Step

When debugging an assembler file, Step Over, Step Out, and Next Statement operate like Step Into; that is, the current instruction is executed at full speed.

When debugging an assembler file, a step operation of a CALL instruction stops at the first instruction of the called function.

When debugging an assembler file, a (true) Step Over a CALL instruction that executes the called function at full device speed can be synthesized by placing a breakpoint after the CALL and using GO (to the breakpoint in realtime mode).

When debugging a C file, a single step (Step) operation executes the next C statement. Thus, it is possible to step over a function reference. If possible, a hardware breakpoint is placed after the function reference, and a GO is implicitly executed. This causes the function to be executed at full speed. If no hardware breakpoints are available, the function is executed in non-realtime mode. Step Into is supported. Step Out is supported.

Within Disassembly mode (**View** → **Disassembly**), a step operation of a non-CALL instruction executes the instruction at full device speed.

Within Disassembly mode (**View** → **Disassembly**), a step operation of a CALL instruction places, if possible, a hardware breakpoint after the CALL instruction, and then executes Go. The called function executes at full

device speed. If no hardware breakpoint is available prior to the Go, the called function is executed in non-realtime mode. In either case, execution stops at the instruction following the CALL.

It is possible to single step only when source statements are present. Breakpoints must be used when running code for which there is no source code (that is, place the breakpoint after the CALL to the function for which there is no source, and then Go to the breakpoint in realtime mode).

If, during a single step operation, an interrupt becomes active, the current instruction is completed and C-SPY stops at the first instruction of the interrupt service routine (see FAQ 26).

2.12.4 Using Watch Windows

The C-SPY Watch Window mechanism permits C variables to be monitored during the debugging session. Although not originally designed to do so, the Watch Window mechanism can be extended to monitor assembler variables.

Assume that the variables to watch are defined in RAM, for example:

```
RSEG DATA16_I varword ds 2 ; two bytes per word
vchar ds 1 ; one byte per character
```

In C-SPY:

1. Open the Watch Window (**View** → **Watch**).
2. Click **Debug** → **Quick Watch**.
3. To watch varword, enter in the Expression box:
(__data16 unsigned int *) varword
4. To watch vchar, enter in the **Expression** box:
(__data16 unsigned char *) vchar
5. Click the **Add Watch** button.
6. Close the Quick Watch window.
7. For the created entry in the Watch Window, click on the + symbol to display the contents (or value) of the watched variable.

To change the format of the displayed variable (default, binary, octal, decimal, hex, char), select the type, click the right mouse button, and then select the desired format. The value of the displayed variable can be changed by selecting it, and then entering the new value.

In C, variables can be watched by selecting them and then dragging and dropping them into the Watch Window.

Because the MSP430 peripherals are memory mapped, it is possible to extend the concept of watching variables to watching peripherals. Be aware that there may be side effects when peripherals are read and written by C-SPY (see FAQ 24).

CPU core registers can be specified for watching by preceding their name with '#' (that is, #PC, #SR, #SP, #R5, and so forth).

Variables watched within the Watch Window are updated only when C-SPY gets control of the device (for example, following a breakpoint hit, a single step, or a stop or escape).

Although registers can be monitored in the Watch Window, **View** → **Register** is the preferred method.

3.1 Introduction

EnergyTrace™ Technology is an energy-based code analysis tool that measures and displays the application's energy profile and helps to optimize it for ultra-low-power consumption.

MSP430 devices with built-in **EnergyTrace+[CPU State]+[Peripheral States]** (or in short **EnergyTrace++™**) support allow real-time monitoring of many internal device states while user program code executes. EnergyTrace++ technology is supported on selected MSP430 devices and debuggers.

EnergyTrace mode (without the "++") is the base of **EnergyTrace Technology** and enables analog energy measurement to determine the energy consumption of an application but does not correlate it to internal device information. The EnergyTrace mode is available for all MSP430 devices with selected debuggers, include CCS.

3.2 Energy Measurement

Debuggers with EnergyTrace Technology support include a new and unique way of continuously measuring the energy supplied to a target microcontroller that differs considerably from the well-known method of amplifying and sampling the voltage drop over a shunt resistor at discrete times. A software-controlled dc-dc converter is used to generate the target power supply. The time density of the dc-dc converter charge pulses equals the energy consumption of the target microcontroller. A built-in on-the-fly calibration circuit defines the energy equivalent of a single dc-dc charge pulse.

Figure 3-1 shows the energy measurement principle. Periods with a small number of charge pulses per time unit indicate low energy consumption and thus low current flow. Periods with a high number of charge pulses per time unit indicate high energy consumption and also a high current consumption. Each charge pulse leads to a rise of the output voltage V_{OUT} , which results in an unavoidable voltage ripple common to all dc-dc converters.

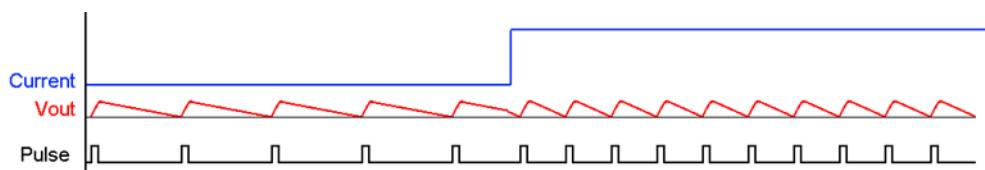


Figure 3-1. Pulse Density and Current Flow

The benefit of sampling continuously is evident: even the shortest device activity that consumes energy contributes to the overall recorded energy. No shunt-based measurement system can achieve this.

3.3 IAR Embedded Workbench® for MSP430 Integration

EnergyTrace Technology is available as part of IAR Embedded Workbench for MSP430 microcontrollers Version 6.10.0 or higher. During debugging of an application, additional windows are available if the debug probe and the target device support EnergyTrace Technology.

3.3.1 Debugging Devices With EnergyTrace++ Technology Support

Devices that support EnergyTrace++™ Technology allow sampling of internal device states while an application is executing (see [Figure 3-2](#)).

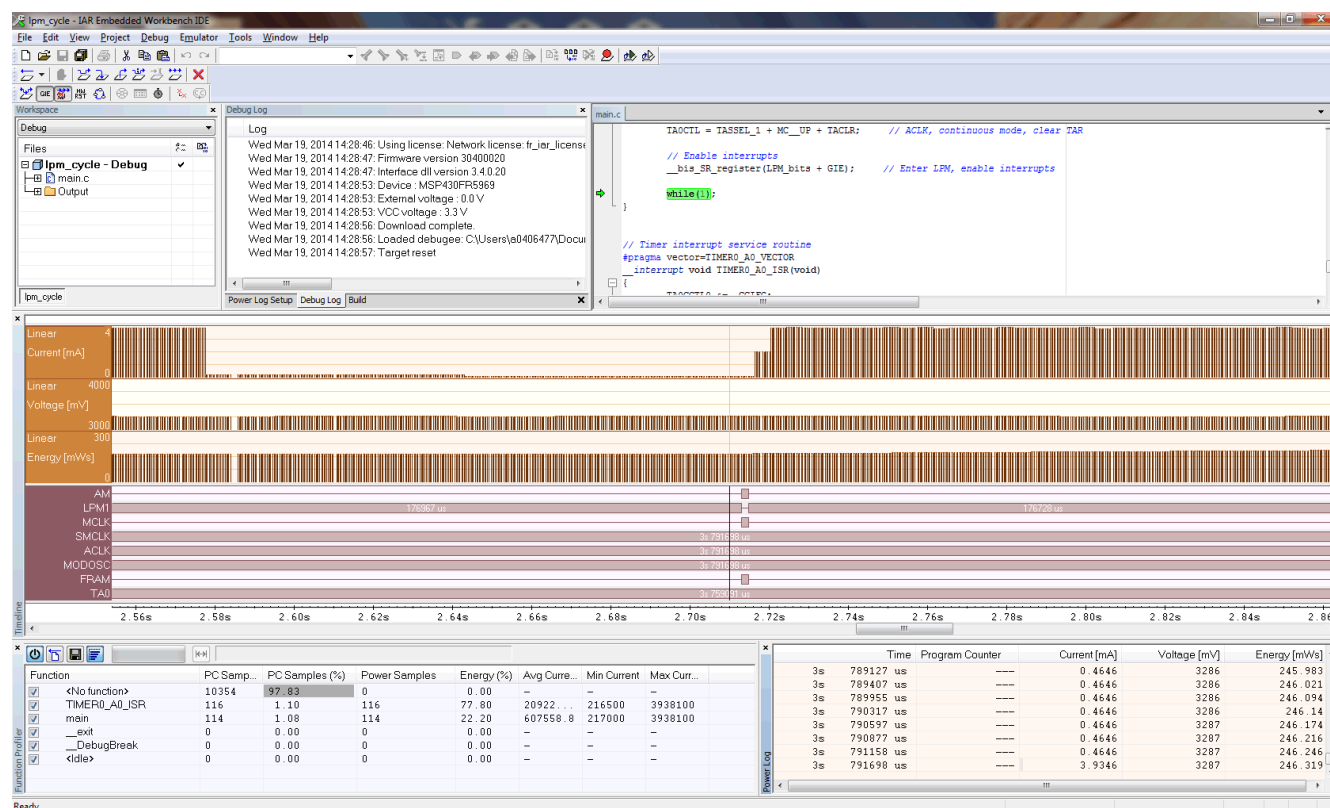


Figure 3-2. Debug Session With EnergyTrace++ Windows

Before downloading and debugging a program, enable the "Enable ULP / LPMx.5 debug" option: right click on the project, select the FET Debugger tab, and check the "Enable ULP / LPMx.5 debug" option (see [Figure 3-3](#)). If this option is not checked, no digital data can be collected from the device. Analog measurements, however, are still possible.

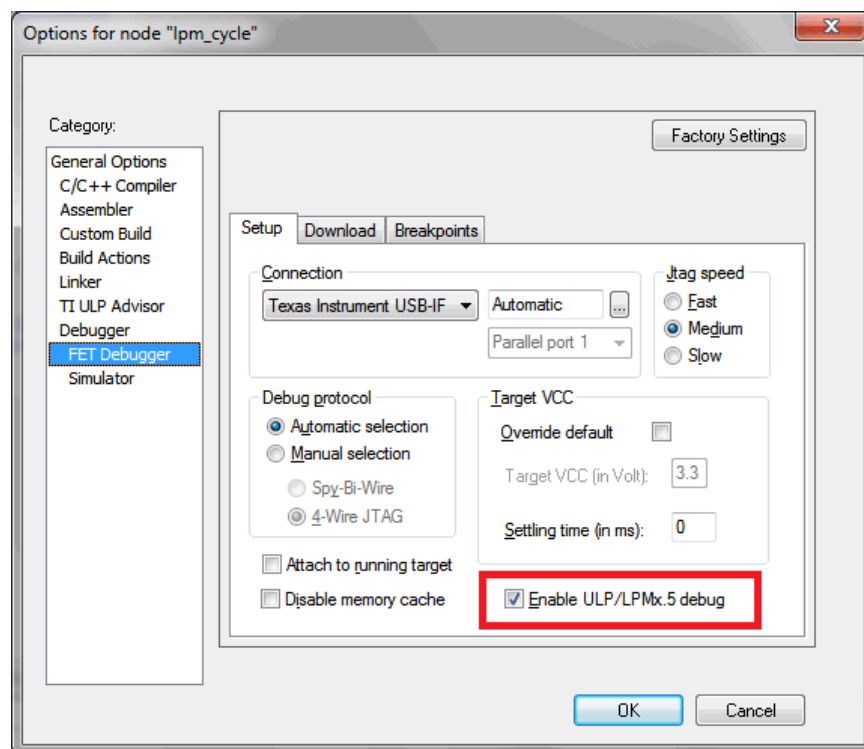


Figure 3-3. Debug Options

Click the **Emulator** menu for EnergyTrace++-related entries (see [Figure 3-4](#)).

- State Log
- Power Log
- Timeline
- Function Profiler

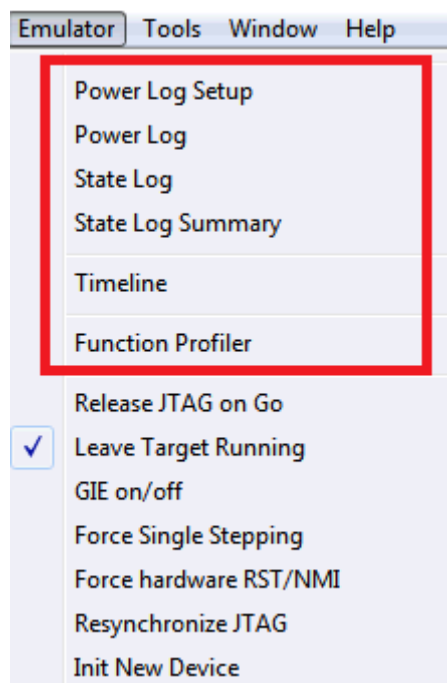


Figure 3-4. Emulator Pulldown Menu With EnergyTrace++-Related Functions

All EnergyTrace-related functions need to be enabled first by right-clicking into the respective window and selecting "Enable" (see [Figure 3-5](#)).

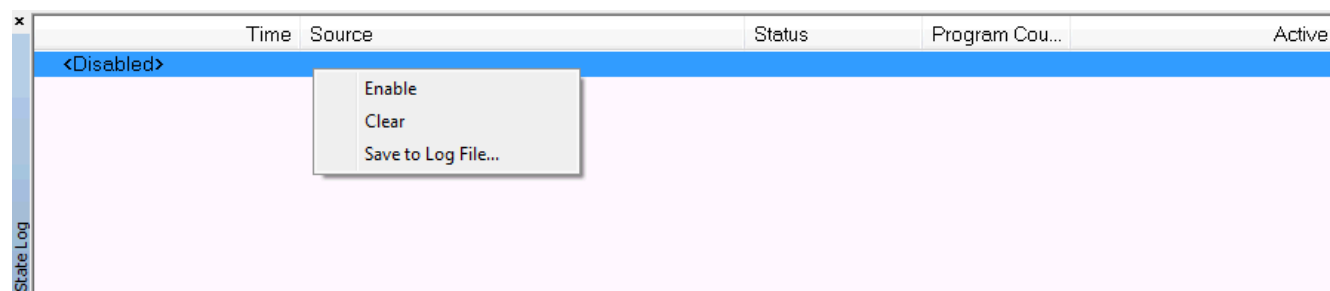
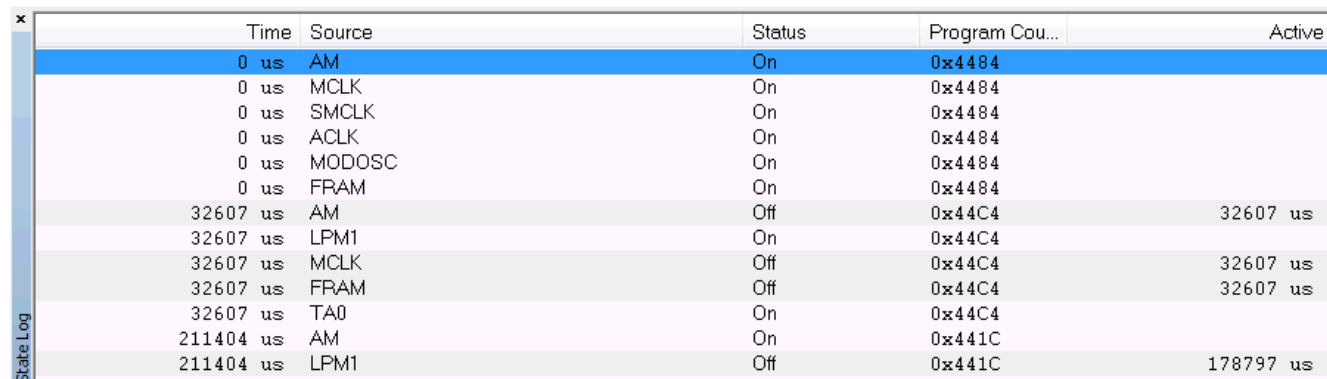


Figure 3-5. Enabling the State Log Window

3.3.1.1 State Log

All peripheral and clock activity is grouped under States. When a program executes, digital data is collected from the target device and displayed in list format (see [Figure 3-6](#)).

The State Log shows at what point in time a specific peripheral or clock has been activated and gives a reference to the program counter location where this happened.

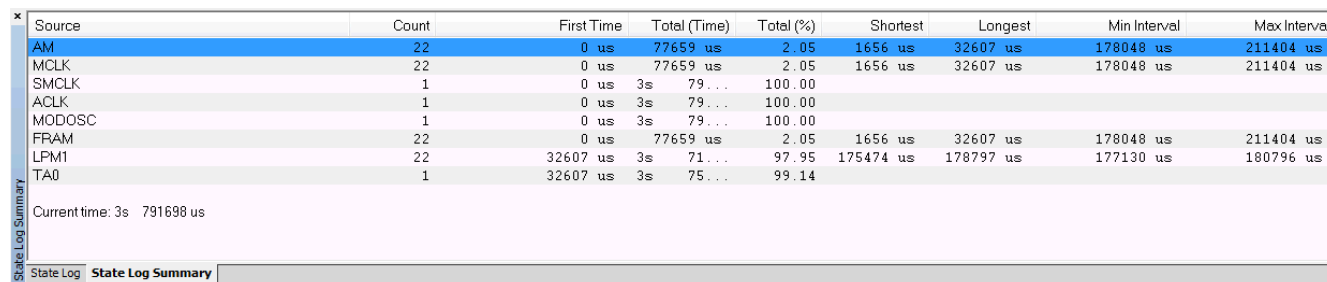


	Time	Source	Status	Program Cou...	Active
	0 us	AM	On	0x4484	
	0 us	MCLK	On	0x4484	
	0 us	SMCLK	On	0x4484	
	0 us	ACLK	On	0x4484	
	0 us	MODOSC	On	0x4484	
	0 us	FRAM	On	0x4484	
	32607 us	AM	Off	0x44C4	32607 us
	32607 us	LPM1	On	0x44C4	
	32607 us	MCLK	Off	0x44C4	32607 us
	32607 us	FRAM	Off	0x44C4	32607 us
	32607 us	TA0	On	0x44C4	
	211404 us	AM	On	0x441C	
	211404 us	LPM1	Off	0x441C	178797 us

Figure 3-6. State Log Window With EnergyTrace++ Data

3.3.1.2 State Log Summary

The State Log Summary window shows a condensed view of the peripheral and clock activity of a profiled program (see [Figure 3-7](#)). Click on the column headers to sort the data.



Source	Count	First Time	Total (Time)	Total (%)	Shortest	Longest	Min Interval	Max Interval
AM	22	0 us	77659 us	2.05	1656 us	32607 us	178048 us	211404 us
MCLK	22	0 us	77659 us	2.05	1656 us	32607 us	178048 us	211404 us
SMCLK	1	0 us	3s 79...	100.00				
ACLK	1	0 us	3s 79...	100.00				
MODOSC	1	0 us	3s 79...	100.00				
FRAM	22	0 us	77659 us	2.05	1656 us	32607 us	178048 us	211404 us
LPM1	22	32607 us	3s 71...	97.95	175474 us	178797 us	177130 us	180796 us
TA0	1	32607 us	3s 75...	99.14				

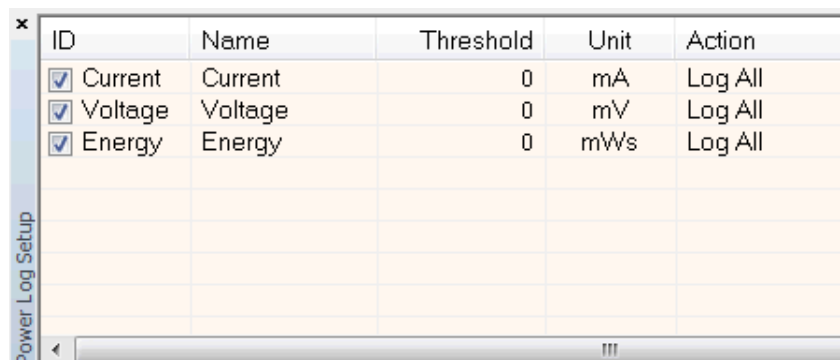
Current time: 3s 791698 us

State Log | State Log Summary

Figure 3-7. State Log Summary With EnergyTrace++ Data

3.3.1.3 Power Log Setup

The Power Log Setup can be used to control the analog measurement (see [Figure 3-8](#)). Check each parameter to enable data collection.



ID	Name	Threshold	Unit	Action
☒	Current	0	mA	Log All
☒	Voltage	0	mV	Log All
☒	Energy	0	mWs	Log All

Figure 3-8. Power Log Setup Window

3.3.1.4 Power Log Window

Similar to the State Log window, the Power Log window shows the current, voltage, and energy profile over time, with reference to the program counter that was sampled at the given time stamp (see [Figure 3-9](#)).

Time	Program Counter	Current [mA]	Voltage [mV]	Energy [mWs]
0 μ s	0x004484	0.3235	3289	0.018
268 μ s	0x004484	0.3235	3289	0.022
613 μ s	0x004486	0.3235	3289	0.025
882 μ s	0x004486	0.3235	3289	0.029
1150 μ s	0x004486	0.3235	3289	0.033
1921 μ s	0x004486	0.3235	3289	0.044
2266 μ s	0x004484	0.3235	3289	0.047
2534 μ s	0x004486	0.3235	3289	0.051
2802 μ s	0x004486	0.3235	3289	0.055
3071 μ s	0x004486	0.3235	3289	0.055
3635 μ s	0x004484	0.3235	3289	0.062
3986 μ s	0x004486	0.3235	3289	0.066
4255 μ s	0x004486	0.3235	3289	0.069
4524 μ s	0x004486	0.3235	3289	0.073

Figure 3-9. Power Log Window With EnergyTrace++ Data

3.3.1.5 Timeline

When invoking the Timeline for the first time, both Power Log and State graphs are disabled. Right click on each section to enable it, and use the mouse wheel to zoom in and out (see [Figure 3-10](#) and [Figure 3-11](#)).

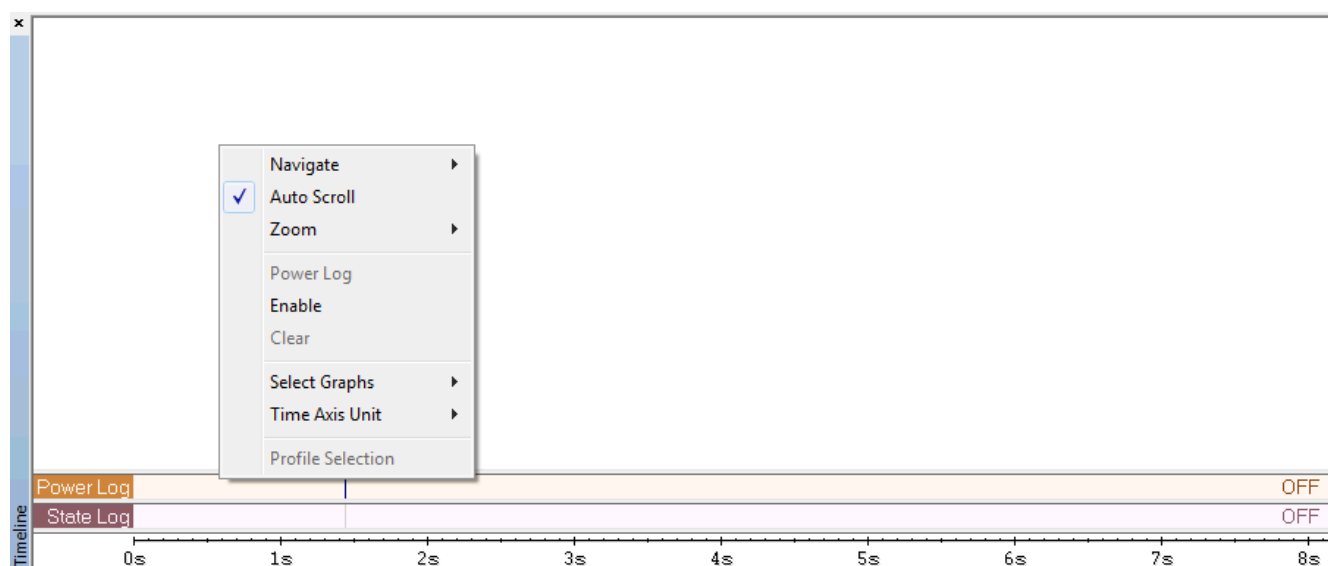


Figure 3-10. Timeline With Power Log and State Graphs Disabled

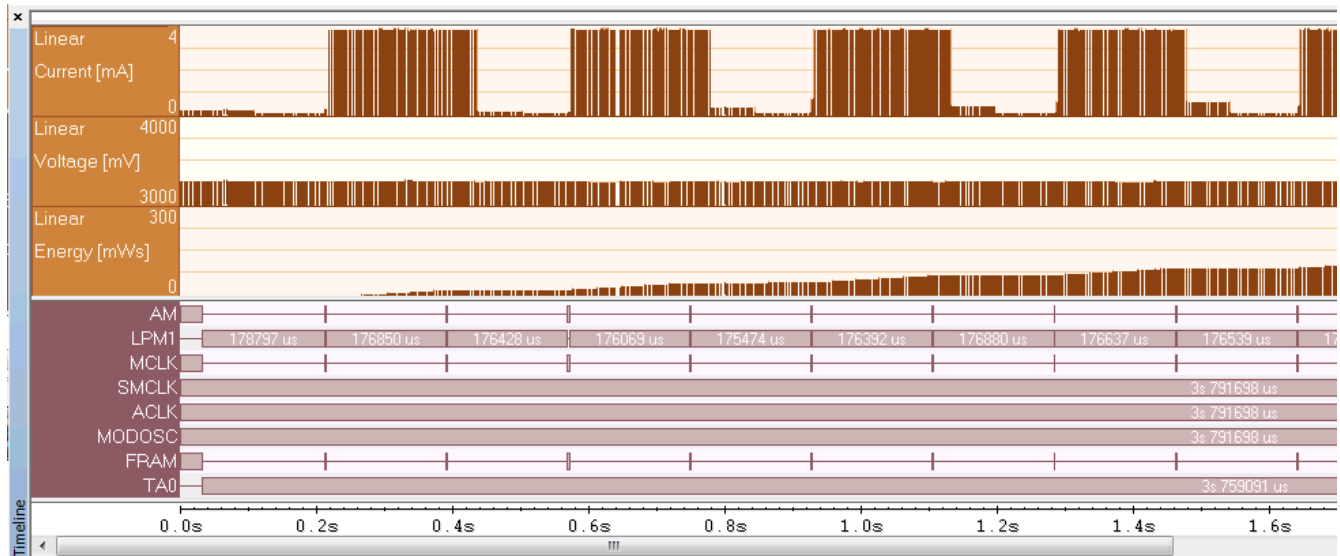


Figure 3-11. Timeline With EnergyTrace++ Data

3.3.1.6 Function Profiler

The Function Profiler gives a condensed overview of the functions and their contribution to total run time and energy consumption (see [Figure 3-12](#)).

Function	PC Samp...	PC Samples (%)	Power Samples	Energy (%)	Avg Curre...	Min Current	Max Current
☒ TIMER0_A0_ISR	116	1.10	116	77.80	20922....	216500	3938100
☒ main	114	1.08	114	22.20	607558.8	217000	3938100
☒ __exit	0	0.00	0	0.00	—	—	—
☒ __DebugBreak	0	0.00	0	0.00	—	—	—
☒ <No function>	10354	97.83	0	0.00	—	—	—
☒ <Idle>	0	0.00	0	0.00	—	—	—

Figure 3-12. Function Profiler With EnergyTrace++ Data

3.3.2 Debugging Devices Without EnergyTrace++ Technology Support

Devices without EnergyTrace++ Technology support still can take advantage of the base EnergyTrace technology analog measurements (see [Figure 3-13](#)).

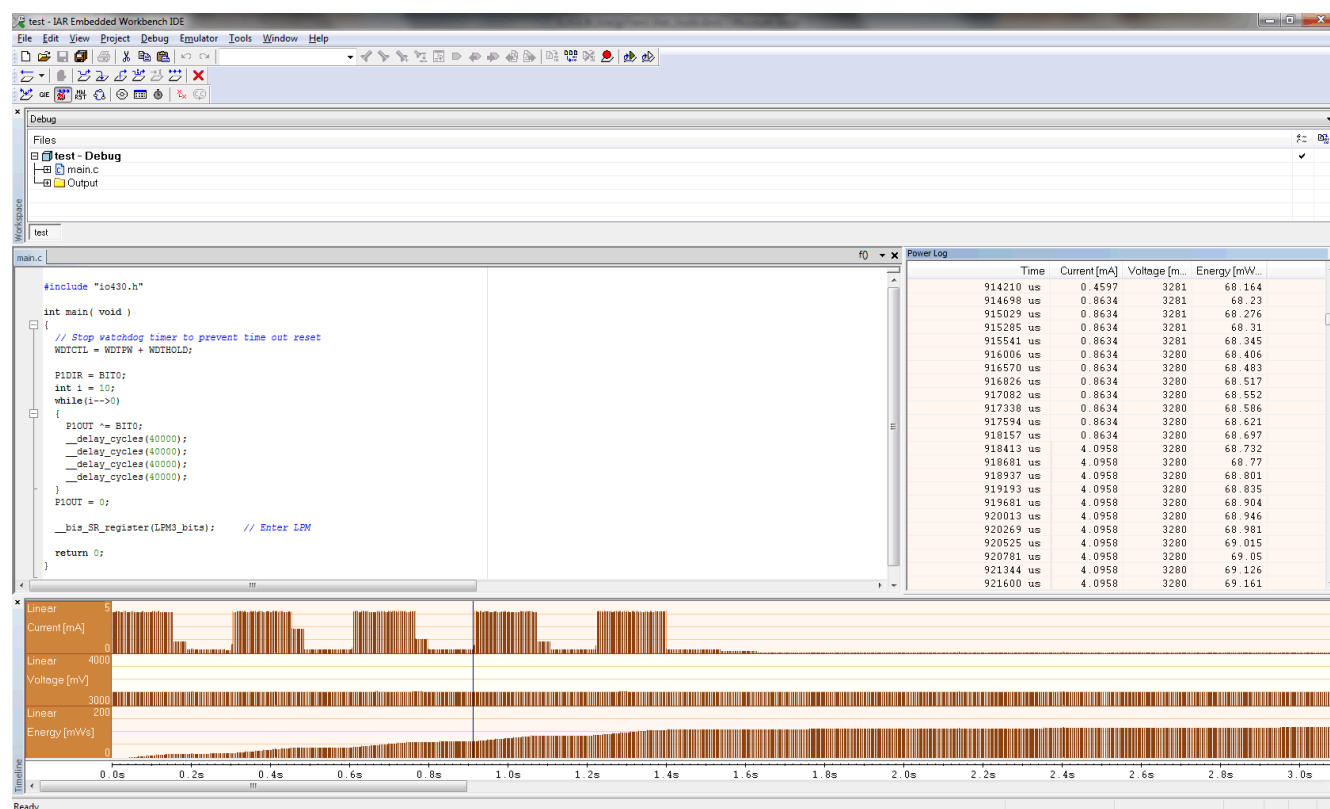


Figure 3-13. Debug Session With EnergyTrace Windows

Click the **Emulator** menu for EnergyTrace-related entries (see [Figure 3-14](#)).

- Power Log
- Timeline

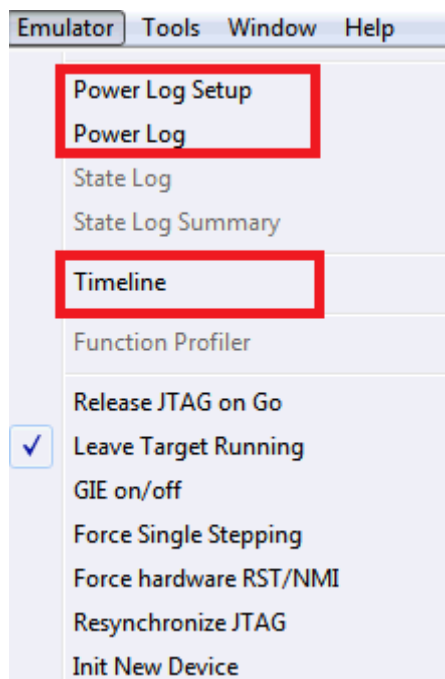


Figure 3-14. Emulator Pulldown Menu With EnergyTrace-Related Functions

3.3.2.1 Power Log Setup

The Power Log Setup can be used to control the analog measurement (see [Figure 3-15](#)). Check each parameter to enable data collection.

ID	Name	Threshold	Unit	Action
☒ Current	Current	0	mA	Log All
☒ Voltage	Voltage	0	mV	Log All
☒ Energy	Energy	0	mWs	Log All

Figure 3-15. Power Log Setup Window

3.3.2.2 Power Log Window

Similar to the State Log window, the Power Log window shows the current, voltage, and energy profile over time (see [Figure 3-16](#)).

Power Log				
Time	Current [mA]	Voltage [mV]	Energy [mW...]	
916570 us	0.8634	3280	68.483	
916826 us	0.8634	3280	68.517	
917082 us	0.8634	3280	68.552	
917338 us	0.8634	3280	68.586	
917594 us	0.8634	3280	68.621	
918157 us	0.8634	3280	68.697	
918413 us	4.0958	3280	68.732	
918681 us	4.0958	3280	68.77	
918937 us	4.0958	3280	68.801	
919193 us	4.0958	3280	68.835	
919681 us	4.0958	3280	68.904	
920013 us	4.0958	3280	68.946	
920269 us	4.0958	3280	68.981	
920525 us	4.0958	3280	69.015	
920781 us	4.0958	3280	69.05	
921344 us	4.0958	3280	69.126	

Figure 3-16. Power Log Window With EnergyTrace Data

3.3.2.3 Timeline

When invoking the Timeline for the first time, the Power Log graph is disabled. Right click to enable it, and use the mouse wheel to zoom in and out (see [Figure 3-17](#) and [Figure 3-18](#)).

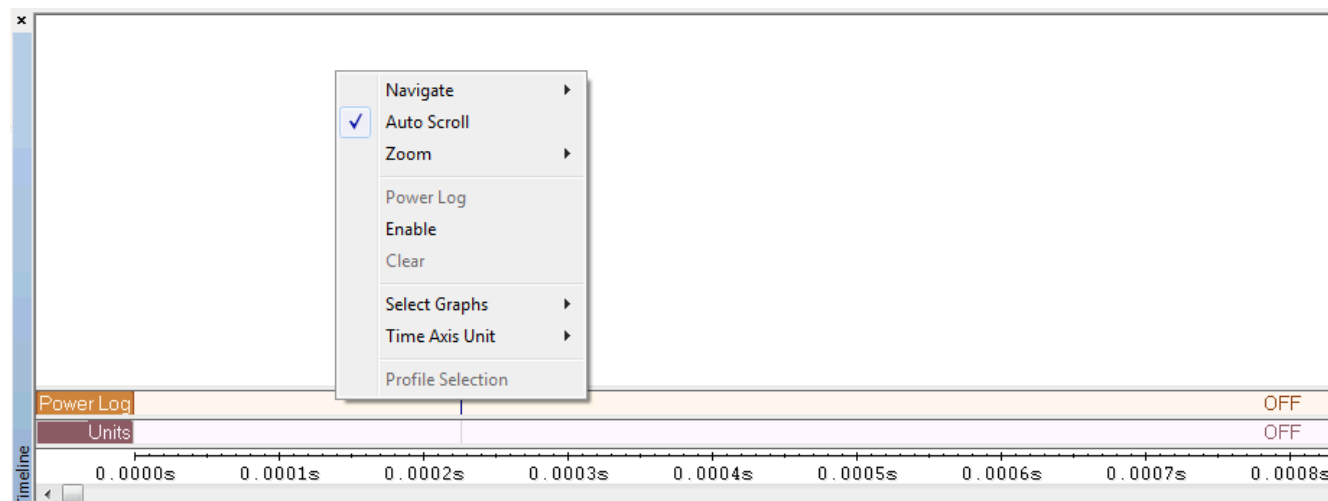


Figure 3-17. Timeline With Power Log Graph Disabled

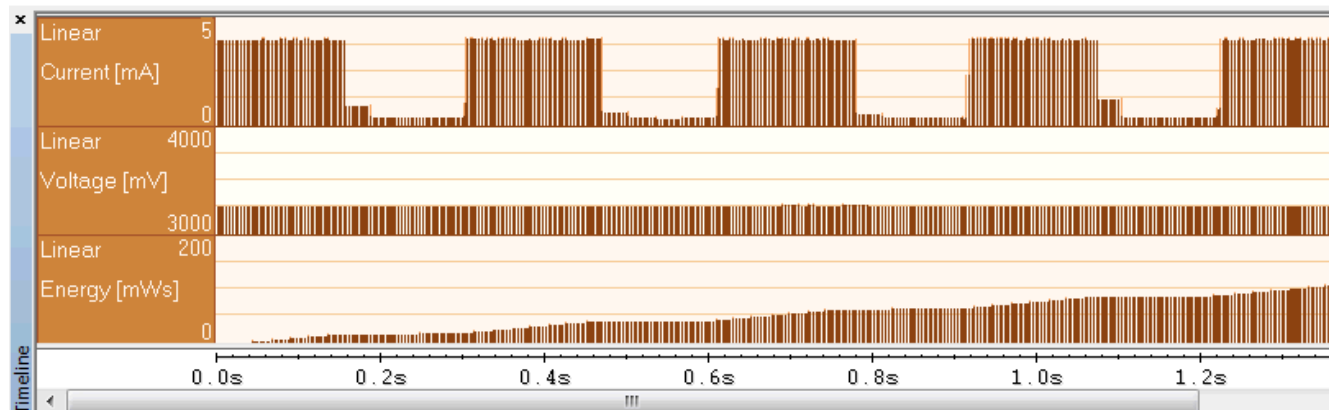


Figure 3-18. Timeline With EnergyTrace Data

3.4 Measuring Low-Power Currents

During the capture of the internal states or even when simply executing until breakpoint halt, the target microcontroller is constantly accessed by the JTAG or Spy-Bi-Wire debug logic. These debug accesses consume energy that is included in the numbers shown in the Power Log window and graph. To measure the absolute power consumption of the application, it is recommended to use the EnergyTrace mode in combination with the Release JTAG on Go option. This combination makes sure that the debug logic of the target microcontroller is not accessed while measuring energy consumption.

Figure 3-19 shows the current consumption of an application sleeping in LPM3, measured when the device is under debug control. The current column indicates approximately 152 μ A current consumption, which is significantly higher than the value stated in the device data sheet and is mostly caused by the current consumption of the device debug logic for breakpoint polling.

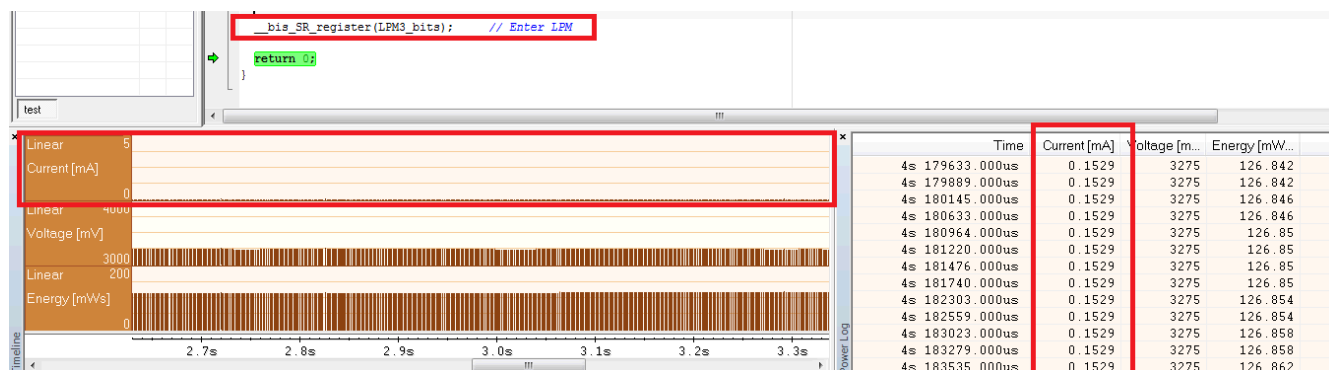


Figure 3-19. LPM3 Current When Executing Under Debug Control

After checking the Release JTAG on Go option in the Emulator pulldown menu (see Figure 3-20), the debugger puts the JTAG or Spy-Bi-Wire lines in Hi-Z state after letting the application go.

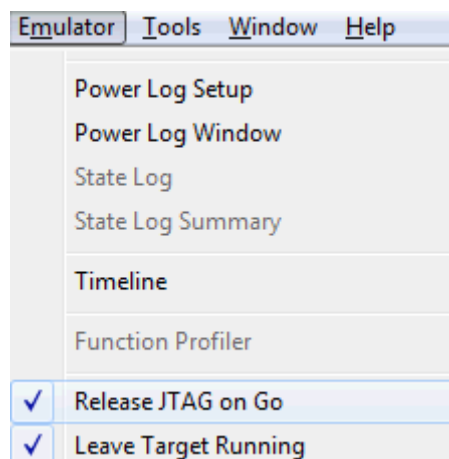


Figure 3-20. Release JTAG on Go Option in Emulator Pulldown Menu

As shown in Figure 3-21, the current consumption for the same code has reduced significantly to less than 4 μA . If this value still deviates considerably from the data sheet value, consider checking GPIO termination or peripheral activity.

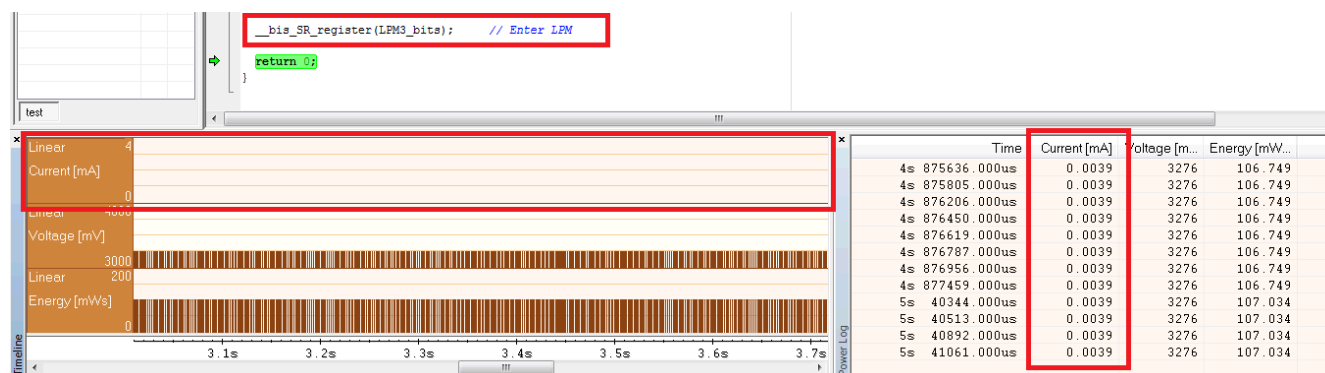


Figure 3-21. LPM3 Current When Executing with JTAG Signals Released

3.5 EnergyTrace Technology FAQs

Q: What is the sampling frequency of EnergyTrace++ technology?

A: The sampling frequency depends on the debugger and the selected debug protocol and its speed setting. It typically ranges from 1 kHz (for example, when using the Spy-Bi-Wire interface set to SLOW) up to 3.2 kHz (for example, when using the JTAG interface set to FAST). The debugger polls the state information of EnergyTrace++ from the device status information. Depending on the sampling frequency, a short or fast duty cycle active peripheral state may not be captured on the State graph. In addition, the higher sampling frequency affects the device energy consumption under EnergyTrace.

Q: What is the sampling frequency of EnergyTrace technology?

A: The sampling frequency to measure the energy consumption is the same independent of which debug protocol or speed and is approximately 4.2 kHz in Free Run mode.

Q: My Power graph seems to include noise. Is my board defective?

A: The power values shown in the Power graph are derived (that is, calculated) from the accumulated energy counted by the measurement system. When the target is consuming little energy, a small number of energy packets over time are supplied to the target, and the software needs to accumulate the dc-dc charge pulses over time before a new current value can be calculated. For currents under 1 μ A, this can take up to one second, while for currents in the milliamp range, a current can be calculated every millisecond. Additional filtering is not applied so that detail information is not lost. Another factor that affects the energy (and with it, the current) that is consumed by the target is periodic background debug access during normal code execution, either through capturing of States information or through breakpoint polling. Try recording in Free Run mode to see a much smoother Power graph.

Q: I have a code that repeatedly calls functions that have the same size. I would expect the function profile to show an equal distribution of the run time. In reality, I see some functions having slightly more run time than expected, and some functions slightly less.

A: During program counter trace, various factors affect the number of times a function is detected by the profiler over time. The microcontroller code could benefit from the internal cache, thus executing some functions faster than others. Another influencing factor is memory wait states and CPU pipeline stalls, which add time variance to the code execution. An outside factor is the sampling frequency of the debugger itself, which normally runs asynchronous to the microcontroller's code execution speed, but in some cases shows overlapping behavior, which also results in an unequal function run time distribution.

Q: My power mode profile sometimes shows short periods of power modes that I haven't used anywhere in my code. For example, I'm expecting a transition from active mode to LPM3, but I see a LPM2 during the transition.

A: When capturing in EnergyTrace++ mode, digital information is continuously collected from the target device. One piece of this information is the power mode control signals. Activation of low-power modes requires stepping through a number of intermediate states. Usually this happens too quickly to be captured by the trace function, but sometimes intermediate states can be captured and are displayed for a short period of time as valid low-power modes.

Q: My profile sometimes includes an <Undetermined> low-power mode, and there are gaps in the States graph Power Mode section. Where does the <Undetermined> low-power mode originate from?

A: During transitions from active mode to low-power mode, internal device clocks are switched off, and occasionally the state information is not updated completely. This state is displayed as <Undetermined> in the Profile window, and the States graph shows a gap during the time that the <Undetermined> low-power mode persists. The <Undetermined> state is an indication that your application has entered a low-power mode, but which mode cannot be accurately determined. If your application is frequently entering low-power modes, the <Undetermined> state will probably be shown more often than if your application only rarely uses low-power modes.

Q: What are the influencing factors for the accuracy of the energy measurement?

A: The energy measurement circuit is directly supplied from the USB bus voltage, and thus it is sensitive to USB bus voltage variations. During calibration, the energy equivalent of a single dc-dc charge pulse is defined, and this energy equivalent depends on the USB voltage level. To ensure a good repeatability and accuracy, power the debugger directly from an active USB port, and avoid using bus-powered hubs and long USB cables that can lead to voltage drops, especially when other consumers are connected to the USB hub. Furthermore the LDO and resistors used for reference voltage generation and those in the calibration circuit come with a certain tolerance and ppm rate over temperature, which also influences accuracy of the energy measurement.

Q: I am trying to capture in EnergyTrace++ mode or EnergyTrace mode with a MSP430 device that is externally powered, but there is no data shown in the Profile, Energy, Power and States window.

A: Both EnergyTrace++ mode and EnergyTrace mode require the target to be supplied from the debugger. No data can be captured when the target microcontroller is externally powered.

Q: I cannot measure LPM currents when I am capturing in EnergyTrace++ mode. I am expecting a few microamps but measure more than 150 μ A.

A: Reading digital data from the target microcontroller consumes energy in the JTAG domain of the microcontroller. Hence, an average current of approximately 150 μ A is measured when connecting an ampere meter to the device power supply pins. If you want to eliminate energy consumption through debug communication, switch to EnergyTrace mode, and let the target microcontroller execute with "Release JTAG on Go" option checked.

Q: My LPM currents seem to be wrong. I am expecting a few microamps but measure more, even in Free Run mode or when letting the device execute without debug control from an independent power supply.

A: The most likely cause of this extra current is improper GPIO termination, as floating pins can lead to extra current flow. Also check the JTAG pins again, especially when the debugger is still connected (but idle), as the debugger output signal levels in idle state might not match how the JTAG pins have been configured by the application code. This could also lead to extra current flow.

The available memory of an FRAM-based microcontroller can be seen as unified memory, which means the memory can be arbitrarily divided between code and data sections. As a consequence, a single FRAM-based microcontroller can be customized for a wide range of application use cases. MSP430 devices support two memory protection methods:

- Memory Protection Unit (MPU) and Intellectual Property Encapsulation (IPE)
- FRAM Memory Write Protection (FRWP). The protection granularity (1k) can be configured on some devices.

See the device-specific data sheet to determine which method is available on a particular device. For instructions on the efficient use of this technology, see [MSP430™ FRAM Technology – How To and Best Practices](#).

4.1 Memory Protection Unit

To prevent accidental overwrites of the program by application data or other forms of data corruption, the MPU can partition the available memory and define access rights for each of the partitions. Thus, it is possible to prevent accidental writes to memory sections that contain application code or to prevent the microcontroller from executing instructions that are located in the data section of the application.

[Figure 4-1](#) shows the MPU and IPE configuration dialog, which is available for FRAM devices that have the MPU feature. To access this dialog, select the menu **Project** → **Options** → **General Options** → **MPU/IPE/FRWP**. For a detailed description of the possible configurations that are provided by the dialog, see the *IAR C/C++ Compiler User Guide*. This dialog allows enabling or disabling of the MPU. The compiler tool chain generates two memory segments (read-write memory and executable memory). The segment borders of these two segments and their respective access bits are placed into the according control registers during device start-up. The compiler also sets the bit for read access of the MPU Info Memory segment. The MPUSEGxVS bit, which selects if a PUC must be executed on illegal access to a segment, is also set by default for each of the segments.

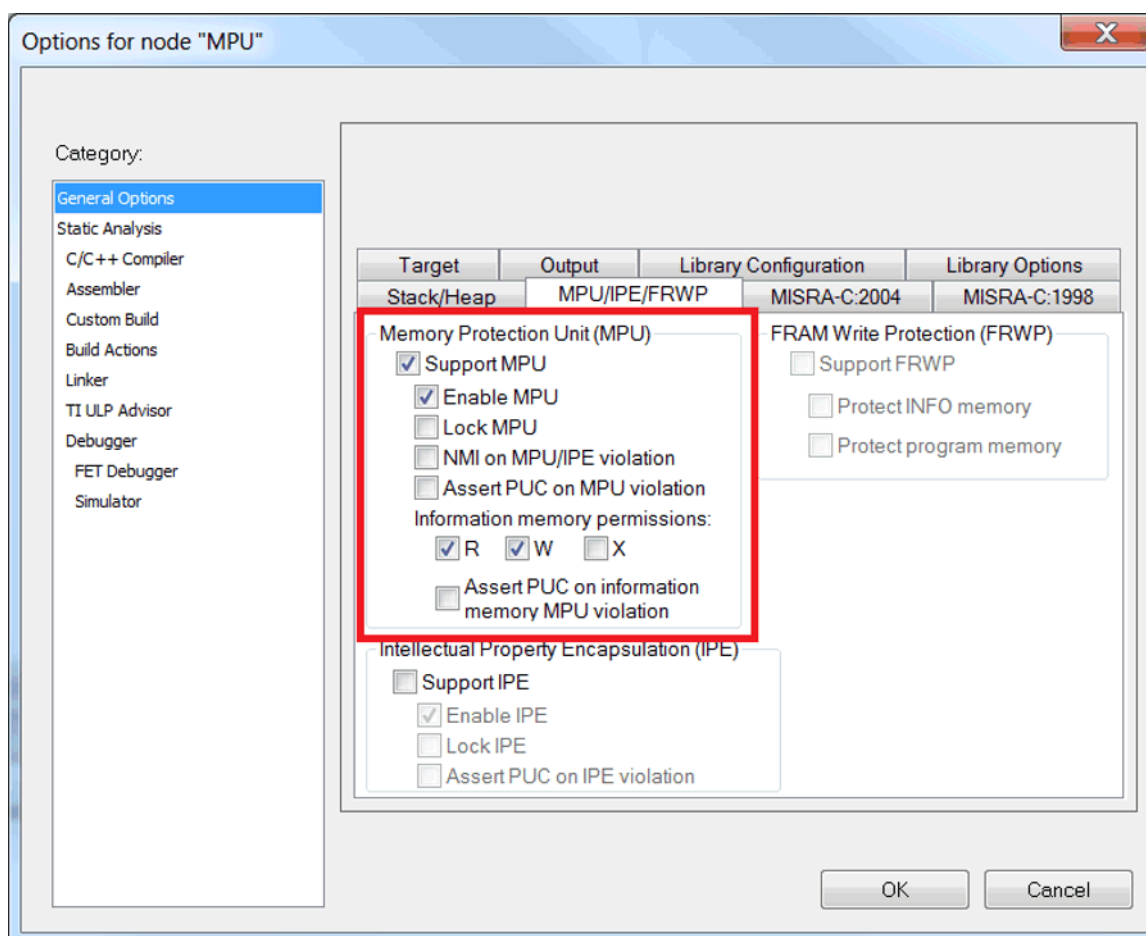


Figure 4-1. MPU Configuration Dialog

4.2 Intellectual Property Encapsulation (IPE)

The memory of many microcontroller applications contains information that should not be accessible by the public. This may include both the application code itself as well as configuration settings for certain peripherals. The IPE module allows the protection of memory that contains this kind of sensitive information. The IPE ensures that only program code that is itself placed in the IPE protected area has access to this memory segment. The access rights are evaluated with each code access, and even JTAG or DMA transfers cannot access the IPE segment. The IPE module is initialized by the boot code before the start of the application code, which ensures that the encapsulation is active before any user-controlled access to the memory can be performed.

Figure 4-2 shows the dialog for configuration of IPE memory, which is accessible through the menu **Project** → **Options** → **General Options** → **MPU/IPE/FRWP**. The runtime library provides support for automatically initializing the IPE to protect the data and code in certain predefined memory segments. If IPE support is included, special data structures are placed in the IPESIGNATURE memory that is read by the boot code to initialize the IPE functionality. The linker configuration files for devices with IPE functionality define the segments IPEDATA16_N, IPEDATA16_C and IPECODE16 in FRAM, surrounded by the border segments IPE_B1 and IPE_B2. Make sure to place writable `__no_init` data to be protected in the IPEDATA16_N segment, constant data in IPEDATA16_C, and the code to read the data in IPECODE16. Additionally, any user-defined segment linked between the border segments IPE_B1 and IPE_B2 is protected.

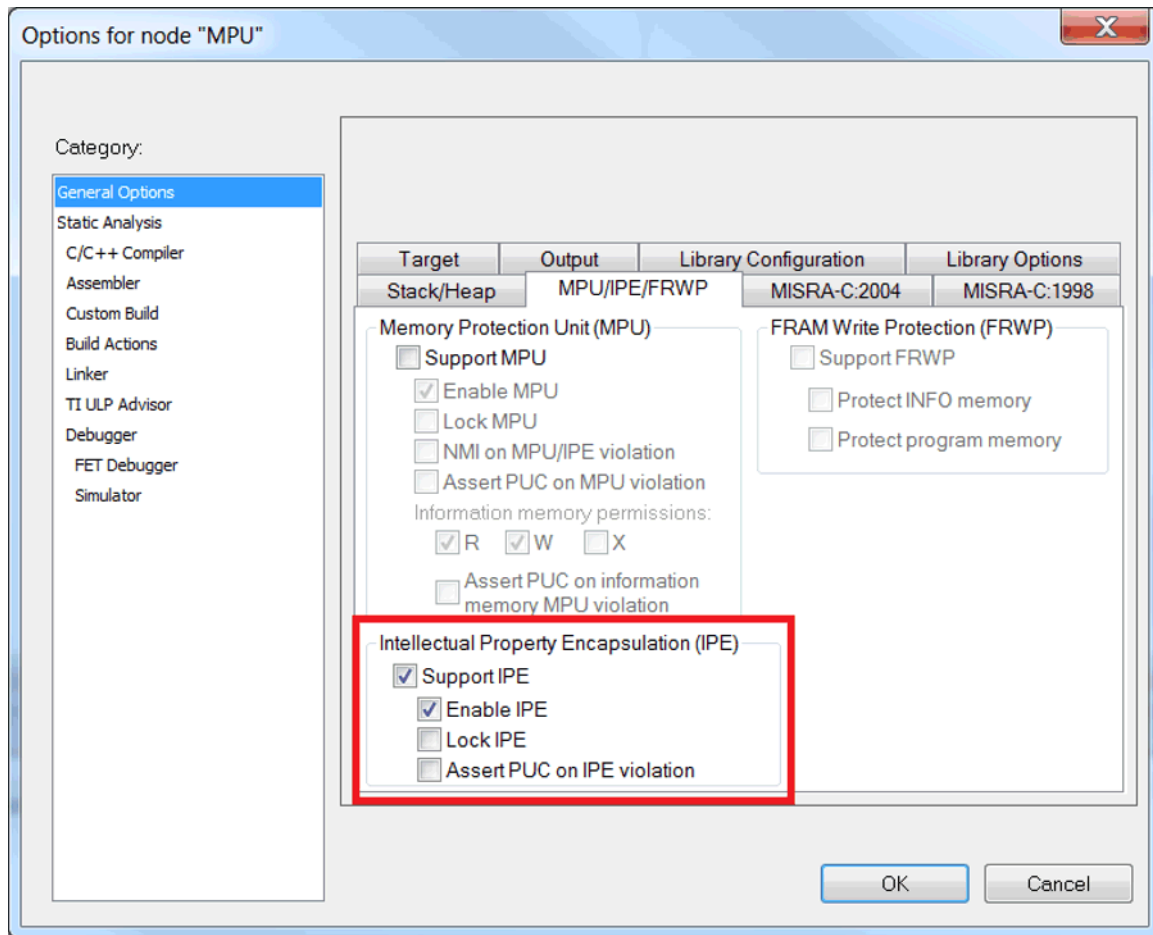


Figure 4-2. IPE Configuration Dialog

For a more detailed description on how to allocate space for certain code or data symbols inside sections, see the *IAR C/C++ Compiler for MSP430 User's Guide*.

4.2.1 IPE Debug Settings

Because it is possible to lock out the debugger from accessing certain memory regions (including downloading new software to the device), it is advisable to enable the option for erasing the IP protected area while the target is under debugger control. The corresponding option can be found under **Project Options** → **Debugger** → **FET Debugger** → **Download**, as shown in Figure 4-3.

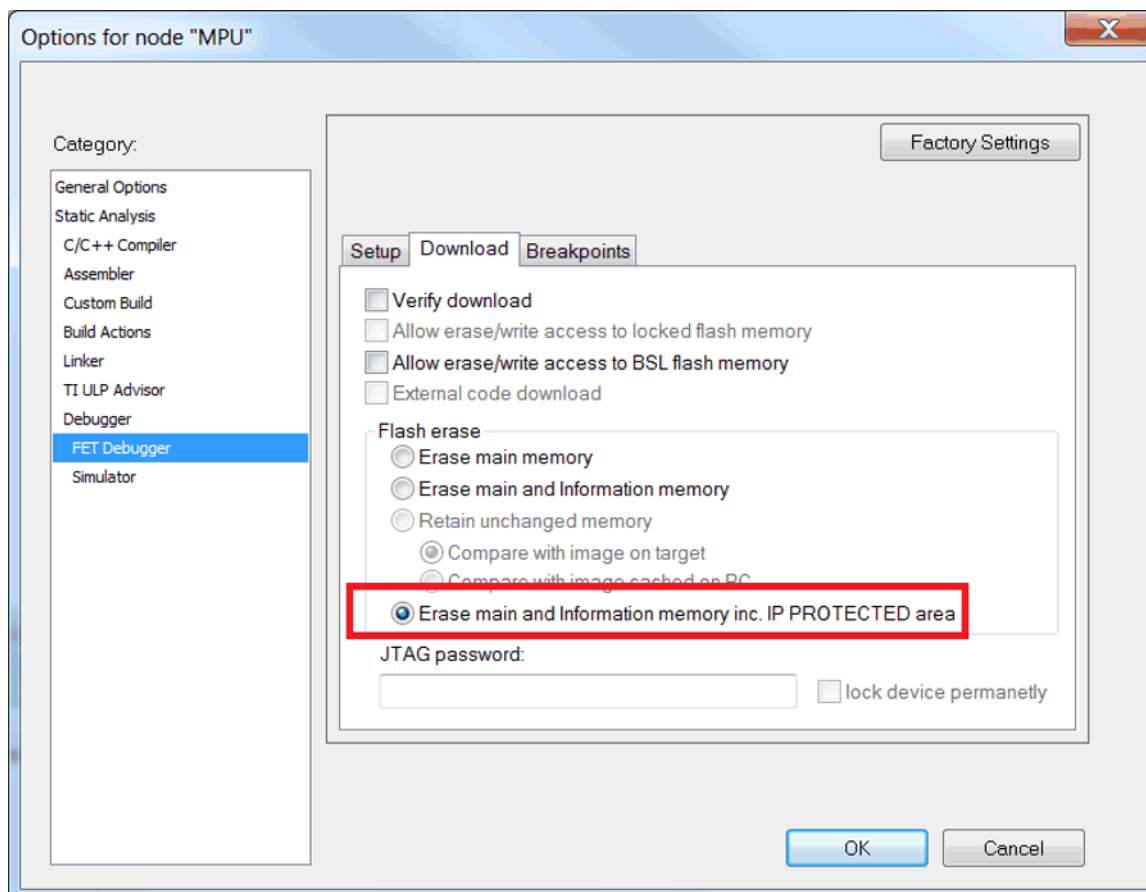


Figure 4-3. IPE Debug Settings

4.3 FRAM Write Protection (FRWP)

The FRWP prevents unintended programming on FRAM code section. For MSP430FR2xx and MSP430FR4xx MCUs, the FRAM memory is protected by setting the bits control in SYSCFG0 register. Some of the MSP430 devices implement the control to protect and unprotect the whole memory at once, and some of the devices such as MSP430FR2355, MSP430FR2353, MSP430FR2155, and MSP430FR2153 can unprotect some region and protect the rest of the memory.

IAR 7.12.1 and newer versions provide a GUI to configure the FRAM write protection. By default, a new IAR project has the Support FRWP option not configured, as the protection for information and program memory by default is enabled by hardware (the entire FRAM memory is protected).

When the Support FRWP option is selected, you can protect or unprotect the information memory and program memory. When the application code uses `__persistent` data type, the size of `__persistent` data is automatically calculated and aligned with 1kB size. These data are then placed in the unprotected program main memory section. The code is placed after the unprotected program main memory.

Figure 4-4 shows the FRWP configuration dialog, which is available for FRAM devices that have the FRWP feature. To access this dialog, select the menu **Project** → **Options** → **General Options** → **MPU/IPE/FRWP**. For a detailed description of the possible configurations that are provided by the dialog, see the *IAR C/C++ Compiler User Guide*.

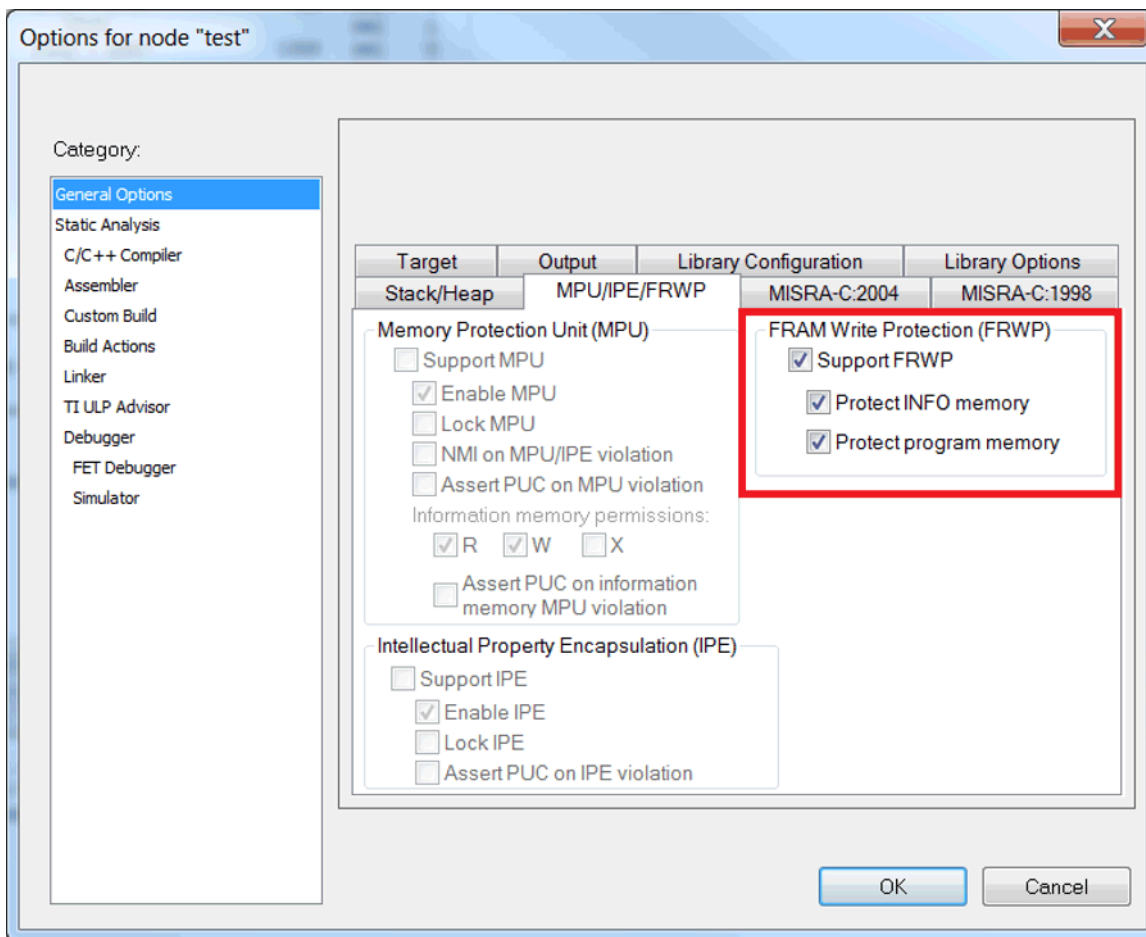


Figure 4-4. FRWP Configuration Dialog

This page intentionally left blank.

This section gives solutions to frequently asked questions regarding program hardware development and debugging tools.

A.1 Hardware

For a complete list of hardware-related FAQs, see the [MSP430 Hardware Tools User's Guide](#).

A.2 Program Development (Assembler, C-Compiler, Linker)

1. **The files supplied in the 430\tutor folder work only with the simulator.** Do not use the files with the FET (see FAQ 11).
2. **A common MSP430 "mistake" is to fail to disable the Watchdog mechanism;** the Watchdog is enabled by default, and it resets the device if not disabled or properly handled by the application (see FAQ 14).
3. **When adding source files to a project, do not add files that are included by source files that have already been added to the project** (for example, an .h file within a .c or .s43 file). These files are added to the project file hierarchy automatically.
4. **In assembler, enclosing a string in double quotes ("string") automatically appends a zero byte** to the string (as an end-of-string marker). Enclosing a string in single-quotes ('string') does not.
5. When using the compiler or the assembler, **if the last character of a source line is backslash (\), the subsequent carriage return or line feed is ignored** (that is, it is as if the current line and the next line are a single line). When used in this way, the backslash character is a "line continuation" character.
6. **The linker output format must be "Debug information for C-SPY" (.d43) for use with C-SPY.** C-SPY does not start otherwise, and an error message is output. C-SPY cannot input a .TXT file.
7. **Position-independent code can be generated** using Project → Options → General Options → Target → Position-Independent Code.
8. **Within the C libraries, GIE (Global Interrupt Enable) is disabled before** (and restored after) **the hardware multiplier is used.** To disable this behavior, contact TI for the source code for these libraries.
9. **It is possible to mix assembler and C programs within the Workbench.** See the Assembler Language Interface chapter of the C/C++ Compiler Reference Guide from IAR.
10. The Workbench can produce an object file in TI .TXT format. **C-SPY cannot input an object file in TI.TXT format.** An error message is output in this case.
11. **The example programs given in the KickStart documentation (that is, Demo, Tutor, and so forth) are not correct.** The programs work only in the simulator. The programs do not function correctly on an actual device, because the watchdog mechanism is active. The programs need to be modified to disable the watchdog mechanism. Disable the watchdog mechanism with this C-statement:

```
WDTCTL = WDTPW + WDTHOLD;
```

 or with this assembler statement:

```
mov.w # WDTPW+WDTHOLD, &WDTCTL
```
12. **Access to MPY using an 8-bit operation is flagged as an error.** Within the .h files, 16-bit registers are defined in such a way that 8-bit operations upon them are flagged as an error. This feature is normally beneficial and can catch register access violations. However, in the case of MPY, it is also valid to access this register using 8-bit operators. If 8-bit operators are used to access MPY, the access violation check mechanism can be defeated by using "MPY_" to reference the register. Similarly, 16-bit operations on 8-bit registers are flagged.

13. **Constant definitions (#define) used within the .h files are effectively reserved** and include, for example, C, Z, N, and V. Do not create program variables with these names.

14. **The CSTARTUP that is implicitly linked with all C applications does not disable the Watchdog timer.** Use `WDCTL = WDTWP + WDTOLD;` to explicitly disable the Watchdog. This statement is best placed in the `__low_level_init()` function that is executed before `main()`.

If the Watchdog timer is not disabled, and the Watchdog triggers and resets the device during CSTARTUP, **the source screen goes blank**, as C-SPY is not able to locate the source code for CSTARTUP. Be aware that CSTARTUP can take a significant amount of time to execute if a large number of initialized global variables are used.

```
int __low_level_init(void)
{
    /* Insert your low-level initializations here */

    WDCTL = WDTWP + WDTOLD; // Stop Watchdog timer

    /*=====*/
    /* Choose if segment initialization */
    /* should be done or not. */
    /* Return: 0 to omit seg_init */
    /*          1 to run seg_init */
    /*=====*/
    return (1);
}
```

15. **Compiler optimization can remove unused variables and statements that have no effect** and can affect debugging. Optimization: NONE is supported within Project → Options → C/C++ Compiler → Code → Optimizations. Alternatively, variables can be declared volatile.

16. **The IAR tutorial assumes a Full or Baseline version of the Workbench.** Within a KickStart system, it is not possible to configure the C compiler to output assembler mnemonics.

17. **Existing projects from an IAR 1.x system can be used within the new IAR 2.x or 3.x system;** see the IAR document migration guide for EW430 x.x. This document is located in <Installation Root>\Embedded Workbench x.x\430\doc\migration.htm

18. **Assembler projects must reference the code segment (RSEG CODE) to use the Linker → Processing → Fill Unused Code Memory** mechanism. No special steps are required to use Linker → Processing → Fill Unused Code Memory with C projects.

19. **Ensure that the proper C runtime library is selected for C-only and mixed C and assembly language projects** (Project → General Options → Library Configuration → Library). For assembly-only projects, the runtime library must not get linked in, otherwise the build fails and a linker error is output (for example, that the RESET vector is allocated twice).

20. **Numerous C and C++ runtime libraries are provided with the Workbench:**

cl430d: C, 64-bit doubles

cl430dp: C, 64-bit doubles, position independent

cl430f: C, 32-bit doubles

cl430fp: C, 32-bit doubles, position independent

dl430d: C++, 64-bit doubles

dl430dp: C++, 64-bit doubles, position independent

dl430f: C++, 32-bit doubles

dl430fp: C++, 32-bit doubles, position independent

See the IAR MSP430 C/C++ compiler reference guide for more information on which library to use.

A.3 Debugging (C-SPY)

1. **Debugging with C-SPY does not seem to affect an externally connected MSP430 device.** Should this be the case, check whether the main debugger menu bar contains a menu item called Simulator. If so, an actual C-SPY MSP430 core simulator session is running, and no actual communication with the target device is established. **Solution: Ensure that the C-SPY driver is set to FET Debugger** (Project → Options → Debugger → Setup → Driver).
2. **C-SPY reports that it cannot communicate with the device.** Possible solutions to this problem include:
 - Ensure the correct debug interface is selected; use Project → Options → FET Debugger → Connection.
 - Ensure the correct parallel port (LPT1, 2, or 3) is being specified in the C-SPY configuration if a parallel port MSP-FET430PIF interface is used; use Project → Options → FET Debugger → Connection → Parallel Port → LPT1 (default) or LPT2 or LPT3. Check the PC BIOS for the parallel port address (0x378, 0x278, 0x3BC), and the parallel port configuration (ECP, Compatible, Bidirectional, or Normal) (see FAQ 8). For users of IBM ThinkPad™ computers, try port specifications LPT2 and LPT3, even if the operating system reports the parallel port is at LPT1.
 - Ensure that the jumper settings are configured correctly on the target hardware.
 - Ensure that no other software application has reserved or taken control of the COM or parallel port (for example, printer drivers, ZIP drive drivers,) if a parallel port MSP-FET430PIF interface is used. Such software can prevent the C-SPY or FET driver from accessing the parallel port and, hence, communicating with the device.
 - Open the Device Manager and determine if the driver for the FET tool has been correctly installed and if the COM or parallel port is successfully recognized by the Windows OS.
 - It may be necessary to reboot the computer to complete the installation of the required port drivers.
 - Ensure that the MSP430 device is securely seated in the socket (so that the "fingers" of the socket completely engage the pins of the device), and that its pin 1 (indicated with a circular indentation on the top surface) aligns with the "1" mark on the PCB.

CAUTION

Possible Damage to Device

Always handle MSP430 devices using a vacuum pick-up tool only; do not use your fingers, as they can easily bend the device pins and render the device useless. Also, always observe and follow proper ESD precautions.

3. **C-SPY reports that the device JTAG security fuse is blown.** With current MSP-FET430PIF and MSP430-FET430UIF JTAG interface tools, there is a weakness when adapting target boards that are powered externally. This leads to an accidental fuse check in the MSP430 and results in the JTAG security fuse being recognized as blown although it is not. This is valid for MSP-FET430PIF and MSP-FET430UIF but is mainly seen on MSP-FET430UIF.

Workarounds:

- Connect the device $\overline{\text{RST}}/\text{NMI}$ pin to JTAG header (pin 11), MSP-FET430PIF or MSP-FET430UIF interface tools are able to pull the $\overline{\text{RST}}$ line, this also resets the device internal fuse logic.
- Do not connect both V_{CC} Tool (pin 2) and V_{CC} Target (pin 4) of the JTAG header and also specify a value for V_{CC} in the debugger that is equal to the external supply voltage.

Note

When the V_{CC} voltage is not high enough when trying to erase or write flash memory, the following message displays in the console: "Target device supply voltage is too low for Flash erase/programming". If this occurs, try to change your supply voltage.

4. **C-SPY can download data into RAM, information, and flash main memories.** A warning message is output if an attempt is made to download data outside of the device memory spaces.
5. **C-SPY can debug applications that use interrupts and low-power modes** (see FAQ 26).
6. **C-SPY cannot access the device registers and memory while the device is running.** C-SPY displays "-" to indicate that a register or memory field is invalid. The user must stop the device to access device registers and memory. Any displayed register or memory fields are then updated.
7. **When C-SPY is started, the flash memory is erased and the opened file is programmed** in accordance with the download options as set in Project → Options → FET Debugger → Download Control. This initial erase and program operations can be disabled selecting Project → Options → FET Debugger → Download Control → Suppress Download. Programming of the flash can be initiated manually with Emulator → Init New Device.
8. **The parallel port designators (LPTx) have the following physical addresses: LPT1: 378h, LPT2: 278h, LPT3: 3BCh.** The configuration of the parallel port (ECP, Compatible, Bidirectional, Normal) is not significant; ECP seems to work well (see FAQ 1 for additional hints on solving communication problems between C-SPY and the device).
9. **C-SPY may assert $\overline{\text{RST}}/\text{NMI}$ to reset the device** when C-SPY is started and when the device is programmed. The device is also reset by the C-SPY RESET button, and when the device is manually reprogrammed (Emulator → Init New Device), and when the JTAG is resynchronized (Emulator → Resynchronize JTAG). When $\overline{\text{RST}}/\text{NMI}$ is not asserted (low), C-SPY sets the logic driving $\overline{\text{RST}}/\text{NMI}$ to high-impedance, and $\overline{\text{RST}}/\text{NMI}$ is pulled high through a resistor on the PCB.

$\overline{\text{RST}}/\text{NMI}$ may get asserted and negated after power is applied when C-SPY is started. $\overline{\text{RST}}/\text{NMI}$ may then get asserted and negated a second time after device initialization is complete.

Within C-SPY, Emulator → "Power on" Reset cycles power to the target to generate a power-on reset.
10. **C-SPY can debug a device whose program reconfigures the function of the $\overline{\text{RST}}/\text{NMI}$ pin to NMI.**
11. **The level of the XOUT/TCLK pin is undefined when C-SPY resets the device.** The logic driving XOUT/TCLK is set to high-impedance at all other times.
12. **When making current measurements of the device, ensure that the JTAG control signals are released** (Emulator → Release JTAG on Go), otherwise the device is powered by the signals on the JTAG pins and the measurements are erroneous (see FAQ 14).
13. **Most C-SPY settings** (breakpoints,) **are preserved between sessions.**
14. **When C-SPY has control of the device, the CPU is ON** (that is, it is not in low-power mode) regardless of the settings of the low-power mode bits in the status register. Any low-power mode conditions are restored prior to Step or Go. Consequently, do not measure the power consumed by the device while C-SPY has control of the device. Instead, run your application using Go with JTAG released (see FAQ 12).
15. The View → Memory → Memory Fill dialog of C-SPY requires **hexadecimal values** for Starting Address, Length, and Value to be **preceded with "0x"**. Otherwise the values are interpreted as decimal.
16. The Memory debug view of C-SPY (View → Memory) can be used to view the RAM, the information memory, and the flash main memory. The Memory utility of C-SPY can be used to modify the RAM; **the information memory and flash main memory cannot be modified using the Memory utility.** The information memory and flash main memory can be programmed only when a project is opened and the data is downloaded to the device, or when Emulator → Init New Device is selected.
17. **C-SPY does not permit the individual segments of the information memory and the flash main memory to be manipulated separately;** consider the information memory to be one contiguous memory, and the flash main memory to be a second contiguous memory.
18. The Memory window correctly displays the contents of memory where it is present. However, **the Memory window incorrectly displays the contents of memory where there is none present.** Memory should be used only in the address ranges specified by the device data sheet.

19. C-SPY uses the system clock to control the device during debugging. Therefore, **device counters, , that are clocked by the Main System Clock (MCLK) are affected when C-SPY has control of the device.** Special precautions are taken to minimize the effect upon the Watchdog Timer. The CPU core registers are preserved. All other clock sources (SMCLK, ACLK) and peripherals continue to operate normally during emulation. In other words, **the Flash Emulation Tool is a partially intrusive tool.**

Devices that support clock control (Emulator → Advanced → Clock Control) can further minimize these effects by selecting to stop the clock(s) during debugging (see FAQ 24).

20. **There is a time after C-SPY performs a reset of the device** [when the C-SPY session is first started, when the flash is reprogrammed (via Init New Device), and when JTAG is resynchronized (Resynchronize JTAG)] and before C-SPY has regained control of the device **that the device executes code normally.** This behavior may have side effects. Once C-SPY has regained control of the device, it performs a reset of the device and retains control.
21. When programming the flash, **do not set a breakpoint on the instruction immediately following the write to flash operation.** A simple workaround to this limitation is to follow the write to flash operation with a NOP, and set a breakpoint on the instruction following the NOP (see FAQ 23).
22. The **Dump Memory length specifier is restricted to four hexadecimal digits (0 to FFFF).** This limits the number of bytes that can be written from 0 to 65535. Consequently, it is not possible to write memory from 0 to 0xFFFF inclusive, as this would require a length specifier of 65536 (or 10000h).
23. Multiple internal machine cycles are required to clear and program the flash memory. **When single stepping over instructions that manipulate the flash,** control is given back to C-SPY before these operations are complete. Consequently, **C-SPY updates its memory window with erroneous information.** A workaround to this behavior is to follow the flash access instruction with a NOP, and then step past the NOP before reviewing the effects of the flash access instruction (see FAQ 21).
24. **Peripheral bits that are cleared when read during normal program execution** (that is, interrupt flags) **are cleared when read while being debugged** (that is, memory dump, peripheral registers).
- When using certain MSP430 devices (such as MSP430F15x, MSP430F16x, MSP430F43x, and MSP430F44x devices), bits do not behave this way (that is, the bits are not cleared by C-SPY read operations).
25. **C-SPY cannot be used to debug programs that execute in the RAM of MSP430F12x and MSP430F41x devices.** A workaround to this limitation is to debug programs in flash.
26. **While single stepping with active and enabled interrupts, it can appear that only the interrupt service routine (ISR) is active** (that is, the non-ISR code never appears to execute, and the single step operation always stops on the first line of the ISR). However, this behavior is correct because the device always processes an active and enabled interrupt before processing non-ISR (that is, mainline) code. A workaround for this behavior is, while within the ISR, to disable the GIE bit on the stack so that interrupts are disabled after exiting the ISR. This permits the non-ISR code to be debugged (but without interrupts). Interrupts can later be reenabled by setting GIE in the status register in the Register window.
- On devices with the clock control emulation feature, it may be possible to suspend a clock between single steps and delay an interrupt request (Emulator → Advanced → Clock Control).
27. **The base (decimal, hexadecimal,) property of Watch Window variables is not preserved between C-SPY sessions;** the base reverts to Default Format.
28. On devices equipped with a Data Transfer Controller (DTC), **the completion of a data transfer cycle preempts a single step of a low-power mode instruction.** The device advances beyond the low-power mode instruction only after an interrupt is processed. Until an interrupt is processed, it appears that the single step has no effect. A workaround to this situation is to set a breakpoint on the instruction following the low-power mode instruction, and then execute (Go) to this breakpoint.
29. **The transfer of data by the Data Transfer Controller (DTC) may not stop precisely when the DTC is stopped in response to a single step or a breakpoint.** When the DTC is enabled and a single step is performed, one or more bytes of data can be transferred. When the DTC is enabled and configured for two-block transfer mode, the DTC may not stop precisely on a block boundary when stopped in response to a single step or a breakpoint.

30. The C-SPY **Register window supports instruction cycle length counters**. The cycle counter is active only while single stepping. The count is reset when the device is reset, or the device is run (Go). The count can be edited (normally set to zero) at any time.
31. **It is possible to use C-SPY to get control of a running device whose state is unknown.** Simply use C-SPY to program a dummy device, and then start the application with Release JTAG on Go selected. Remove the JTAG connector from the dummy device and connect to the unknown device. Select Debug → Break (or the Stop hand) to stop the unknown device. The state of the device can then be interrogated.
32. Resetting a program temporarily requires a breakpoint if Project → Options → Debugger → Setup → Run To is enabled. If N or more breakpoints are set, Reset sets a virtual breakpoint and runs to the Run To function. Consequently, **it may require a significant amount of time before the program resets** (that is, stops at the Run To function). During this time the C-SPY indicates that the program is running, and C-SPY windows may be blank (or may not be correctly updated).
33. Run To Cursor temporarily requires a breakpoint. If N breakpoints are set and virtual breakpoints are disabled, **Run To Cursor incorrectly uses a virtual breakpoint**. This results in very slow program execution.
34. **The simulator is a CPU core simulator only**; peripherals are not simulated, and interrupts are statistical events.
35. On devices without data breakpoint capabilities, it is possible to associate with an instruction breakpoint an (arbitrarily complex) expression that C-SPY evaluates when the breakpoint is hit. **This mechanism can be used to synthesize a data breakpoint**. See the C-SPY documentation for a description of this complex breakpoint mechanism.
36. **The ROM Monitor** referenced by the C-SPY documentation applies only to older MSP430Exxx (EPROM) based devices; it **can be ignored** when using the FET and the flash-based MSP430F devices.
37. **Special function registers** (SFRs) and the peripheral registers **are displayed in View → Register**.
38. **The putchar() and getchar() breakpoints are set only if these functions are present** (and the mechanism is enabled). Note that putchar() and getchar() could be indirectly referenced by a library function.
39. **The flash program and download progress bar does not update gradually**. This behavior is to be expected. The progress bar updates whenever a "chunk" of memory is written to flash. The development tools attempt to minimize the number of program chunks to maximize programming efficiency. Consequently, it is possible, for example, for a 60Kbyte program to be reduced to a single chunk, and the progress bar is not updated until the entire write operation is complete.
40. **After moving a complete EW430 project** (including workspace, project, source and generated object files) **to a different storage location** (for example, a different PC) a rebuild of the object files (rebuild project) is required before starting C-Spy. The Linker stores absolute path names in the object files, which probably do not match the new storage location. C-Spy can show a message that the source files cannot be located or can show strange artifacts during debugging.

Note

Linux and OS X do not support the MSP-FET430UIF if it has an old firmware image (MSP Debug Stack v2) on it.

Customers who buy a new MSP-FET430UIF will encounter this issue on OS X or Linux, because the MSP Debug Stack v2 is programmed on the debugger during production. To resolve this problem, connect the debugger to a Windows PC and use IAR, CCS, or the MSP430 Flasher to update the firmware on the debugger to the latest version (v3 or newer).

Note

Do not connect through a USB hub when performing a firmware update on the MSP-FET, the MSP-FET430UIF, or a LaunchPad™ development kit.

B.1 Menus

B.1.1 Emulator → Device Information

Opens a window with information about the target device being used. Also, this window allows adjusting the target voltage in the case an MSP-FET430UIF interface is used to supply power to the target by performing a right-click inside this window. The supply voltage can be adjusted between 1.8 V and 5 V. This voltage is available on pin 2 of the 14-pin target connector to supply the target from the MSP-FET430UIF. If the target is supplied externally, the external supply voltage should be connected to pin 4 of the target connector, so the MSP-FET430UIF can set the level of the output signals accordingly.

B.1.2 Emulator → Release JTAG on Go

C-SPY uses the device JTAG signals to debug the device. On some MSP430 devices, these JTAG signals are shared with the device port pins. Normally, C-SPY maintains the pins in JTAG mode so that the device can be debugged. During this time the port functionality of the shared pins is not available.

However, when Release JTAG On Go is selected, the JTAG drivers are set to three-state, and the device is released from JTAG control (TEST pin is set to GND) when Go is activated. Any active on-chip breakpoints are retained, and the shared JTAG port pins revert to their port functions.

At this time, C-SPY has no access to the device and cannot determine if an active breakpoint (if any) has been reached. C-SPY must be manually commanded to stop the device, at which time the state of the device is determined (that is, was a breakpoint reached?) (see [FAQ 12](#)).

B.1.3 Emulator → Resynchronize JTAG

Regain control of the device.

It is not possible to Resynchronize JTAG while the device is operating.

B.1.4 Emulator → Init New Device

Initialize the device according to the settings in the Download Options. Basically, the current program file is downloaded to the device memory. The device is then reset. This option can be used to program multiple devices with the same program from within the same C-SPY session.

It is not possible to select Init New Device while the device is operating.

B.1.5 Emulator → Secure - Blow JTAG Fuse

Blows the fuse on the target device. After the fuse is blown, no communication with the device is possible.

B.1.6 Emulator → Breakpoint Usage

List all used hardware and virtual breakpoints, as well as all currently defined EEM breakpoints.

B.1.7 Emulator → Advanced → Clock Control

Disable the specified system clock while C-SPY has control of the device (following a Stop or breakpoint). All system clocks are enabled following a Go or a single step (Step or Step Into) (see [FAQ 19](#)).

B.1.8 Emulator → Advanced → Emulation Mode

Specify the device to be emulated. The device must be reset (or reinitialized through Init New Device) following a change to the emulation mode.

B.1.9 Emulator → Advanced → Memory Dump

Write the specified device memory contents to a specified file. A conventional dialog is displayed that permits the user to specify a file name, a memory starting address, and a length. The addressed memory is then written in a text format to the named file. Options permit the user to select word or byte text format, and address information and register contents also can be appended to the file.

B.1.10 Emulator → Advanced → Breakpoint Combiner

Open the Breakpoint Combiner dialog box. The Breakpoint Combiner dialog box permits one to specify breakpoint dependencies. A breakpoint is triggered when the breakpoints are encountered in the specified order.

B.1.11 Emulator → State Storage Control

Open the State Storage dialog box. The State Storage dialog box permits the user to use the state storage module. The State Storage Module is not present on all MSP430 derivatives. See [Table 2-1](#) for implementation details

See the IAR C-SPY FET Debugger section in the MSP430 IAR Embedded Workbench IDE User Guide.

B.1.12 Emulator → State Storage Window

Open the State Storage window, and display the stored state information as configured by the State Storage dialog.

See the IAR C-SPY FET Debugger section in the MSP430 IAR Embedded Workbench IDE User Guide.

B.1.13 Emulator → Sequencer Control

Open the Sequencer dialog box. The Sequencer dialog box permits the user to configure the sequencer state machine.

See the IAR C-SPY FET Debugger section in the MSP430 IAR Embedded Workbench IDE User Guide.

B.1.14 Emulator → "Power on" Reset

Cycle power to the device to effect a reset.

B.1.15 Emulator → GIE on/off

Enables or disables all interrupts. Needs to be restored manually before Go.

B.1.16 Emulator → Leave Target Running

If C-SPY is closed, the target keeps running the user program.

B.1.17 Emulator → Force Single Stepping

On Go the program is executed by single steps. The cycle counter works correctly only in this mode.

B.1.18

Note**Availability of Emulator → Advanced menus**

Not all Emulator → Advanced menus are supported by all MSP430 devices. These menus are grayed out.

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from May 12, 2018 to December 14, 2020	Page
• Removed support for FET430PIF.....	5
• Deleted listing of MSP-FET430PIF debug port.....	12
• Deleted listing of PIF for configuring a project.....	12

This page intentionally left blank.

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES "AS IS" AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI's products are provided subject to [TI's Terms of Sale](#) or other applicable terms available either on [ti.com](#) or provided in conjunction with such TI products. TI's provision of these resources does not expand or otherwise alter TI's applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2022, Texas Instruments Incorporated