

Advanced Debugging Using the Enhanced Emulation Module (EEM) With IAR Embedded Workbench 5.60

Stefan Schauer

MSP430

ABSTRACT

This document describes the benefits of the Enhanced Emulation Module (EEM) advanced debug features of the MSP430™ devices and how they can be used in IAR Embedded Workbench™ version 5.60 software development tool.

The EEM advanced debugging features support both precision analog and full-speed digital debugging. The configuration of the debug environment for maximum control and the use of the embedded trace capability are described. Some techniques that allow rapid development and design-for-testability are shown.

Project collateral and source code discussed in this application report can be downloaded from www.ti.com/lit/zip/slaa263.

Contents

1	Introduction	2
2	Triggers	2
3	Breakpoint	2
4	State Storage.....	6
5	Advanced Trigger Options.....	7
6	Clock Control	7
7	Cycle Counter	8
8	Attach to Running Target	9
9	Considerations.....	10
10	Emulation Module Implementation Summary.....	10
11	C-Spy Internal Breakpoint Use	11
12	References	11
Appendix A Examples		12

List of Figures

1	Conditional Breakpoint Dialog.....	3
2	Conditional Breakpoint Dialog for Register.....	4
3	Setting a Range Breakpoint	5
4	State Storage.....	6

List of Tables

1	Emulation Module Overview	10
---	---------------------------------	----

MSP430, LaunchPad are trademarks of Texas Instruments.
IAR Embedded Workbench is a trademark of IAR Systems.
All other trademarks are the property of their respective owners.

1 Introduction

Every MSP430 flash-based microcontroller includes on-chip debug logic. This Enhanced Emulation Module (EEM) provides different levels of debug features, depending on the selected device. This application report describes how the EEM can be used to solve typical debugging problems.

In general, the following features are available:

- Two to eight hardware breakpoints
- Complex breakpoints
- Break on read or write at specified address
- Protection of read and write areas within memory
- All timers and counters can be stopped (device dependent)
- PWM generation may not be stopped on emulation hold
- Single step, step into, step over, and run in real-time
- Full support of all low-power modes
- Support for digitally controlled oscillator (DCO) dependencies such as temperature and voltage

The EEM logic in the MSP430 devices works nonintrusively, meaning that it does not use or lock any resources targeted for the application, such as registers, memory, or interrupts.

NOTE: All examples in this application report are based on IAR version 5.60. Many of the other debuggers have the same or similar features. For details about using these other debuggers, see the user's guide of the dedicated debugger.

2 Triggers

The event control in the EEM of the MSP430 system consists of Triggers, which are internal signals indicating that a certain event has happened. These Triggers may be used as simple breakpoints, but it is also possible to combine two or more Triggers to allow detection of complex events. In general, the Triggers could be used to control the following functional blocks of the EEM:

- Breakpoints
- State storage

There are two fundamental different types of Triggers, one for the address and data bus and the other for the CPU registers. It is also possible to define under which condition the Trigger is active. Such conditions include reading, writing, or fetching an instruction. These Triggers can also be combined, so that a Trigger gives a signal if a particular value is written into a specified address.

3 Breakpoint

Triggers are used to configure breakpoints. This very flexible system allows the definition of various powerful breakpoints.

3.1 Address Breakpoints

A simple code breakpoint in this context would be a Trigger with a certain value (instruction address) on the address bus combined with the fetch signal of the CPU.

For the address breakpoint, one Trigger is used.

3.2 Data Breakpoints

Another type of breakpoint, called a data breakpoint, can be configured by using one or two Triggers. A data breakpoint can be used to check for a certain value on the address bus (memory address of the variable) combined with a read or write signal. It can also be enhanced, so that the break only occurs if a specific value is read or written into this address. This value is then checked on the data bus.

For a data breakpoint without a value, one Trigger is used. For a data breakpoint with a value, two Triggers are used.

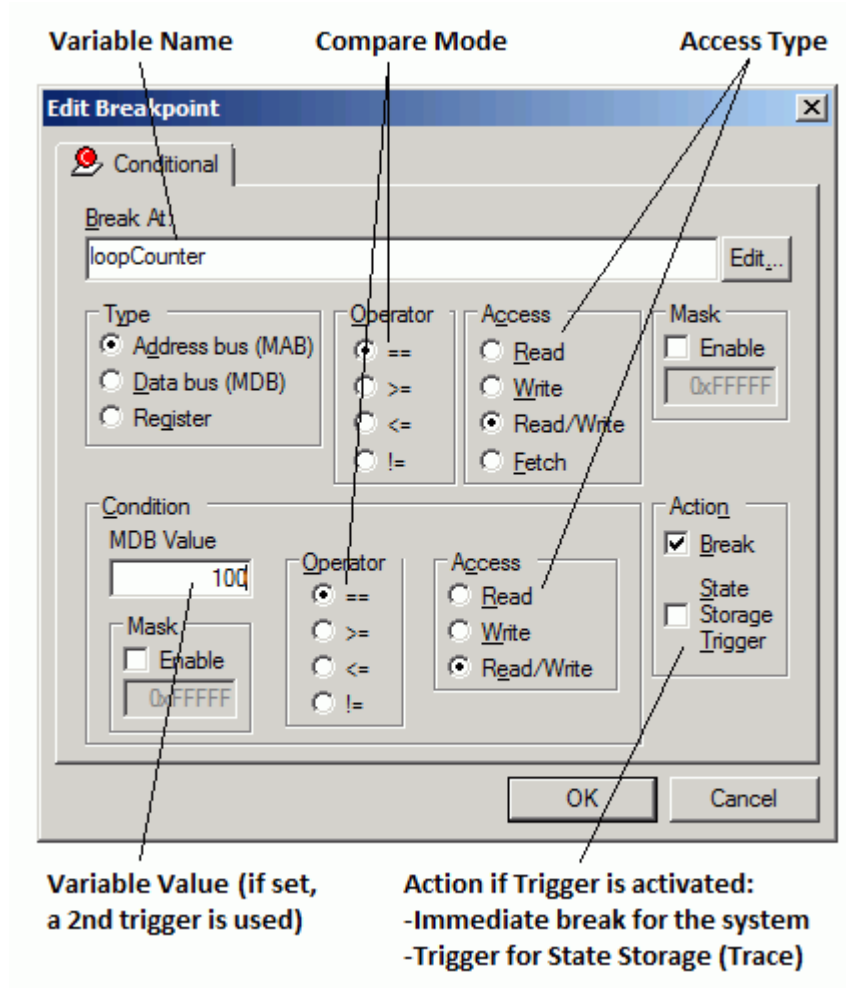


Figure 1. Conditional Breakpoint Dialog

3.3 Register Breakpoints

The same observation methods can be used for the CPU registers. This can be a very powerful tool if the program is written in assembler, where the programmer has complete control over the use of the registers. Dedicated registers might be used for a variable or a system state flag. The stack pointer register is critical and worth observing very carefully. If there is a problem within the program that allows the stack to run into the data area, it is often very difficult to find the problem with normal debugging features, as the symptoms may change each time program execution is started. A simple breakpoint that stops the microcontroller when the stack pointer is equal to or below a certain value, helps detect such problems quite easily. To set up this Trigger, one of the Register Triggers is used.

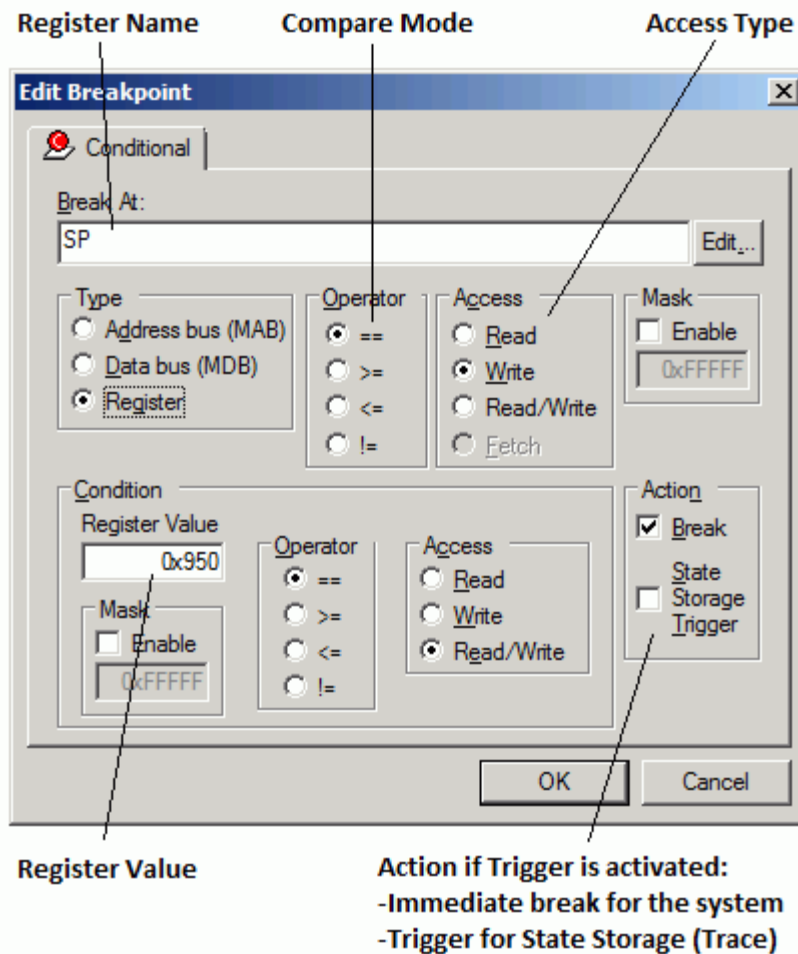


Figure 2. Conditional Breakpoint Dialog for Register

NOTE: When using an MSP430X CPU (for example, MSP430FG46x), a second breakpoint is required for observing the stack pointer (SP) due to the speed improvement on the MSP430X CPU. Set a Conditional Breakpoint on an MAB write access with the address of the SP limit.

3.4 Mask Register

The Mask Register in the Conditional Breakpoint dialog defines the bits of the MDB value being written to the register to be compared against the specified value. This could be used, for example, to check if only a certain bit is being set and cleared in the specified register.

3.5 Range Breakpoints

Range breakpoints are necessary to check for an access to a specific memory range. Some possible conditions could be:

- Break on Write to Flash – This allows a check to determine if a write access into the Flash memory area occurs during program execution. In many cases, this is not allowed (or is allowed only under certain circumstances) and, therefore, could be considered as an erroneous write.
- Break on Read or Write to Invalid Memory – This check could be used to evaluate if any attempt to access data in invalid memory range occurs during program execution.
- Break on Instruction Fetch out of Range – This breakpoint could stop the CPU if it fetches an instruction from a memory address where no program is stored.
- Break on Data out of Range – This check gives a signal if the value at the data bus is inside or outside the specified range. This could be used if the value in a certain variable should be observed for this data range. To use this feature, the Data Range Trigger must be combined with a write or read Trigger on the variable address. Otherwise, any value that appears on the data bus and is in this range (for example, an instruction) could stop the CPU.

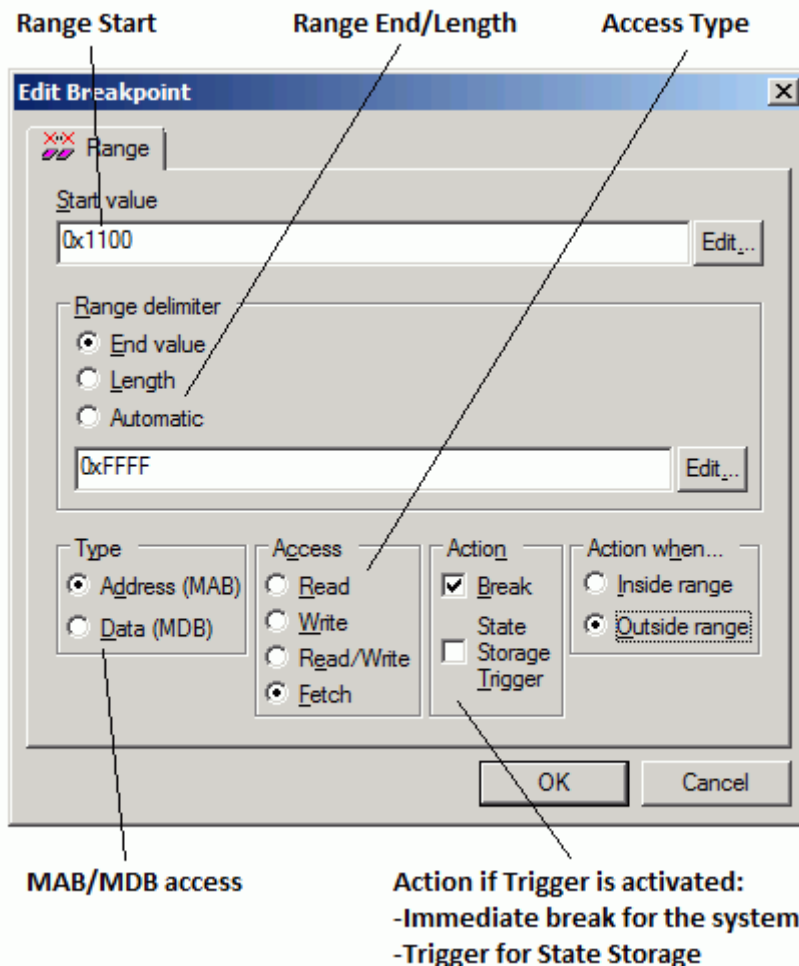


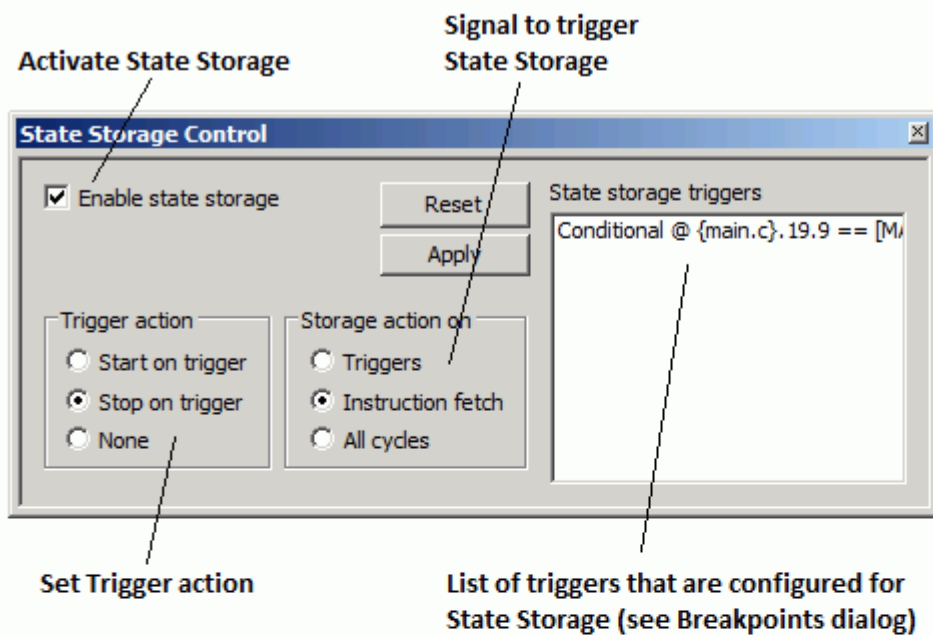
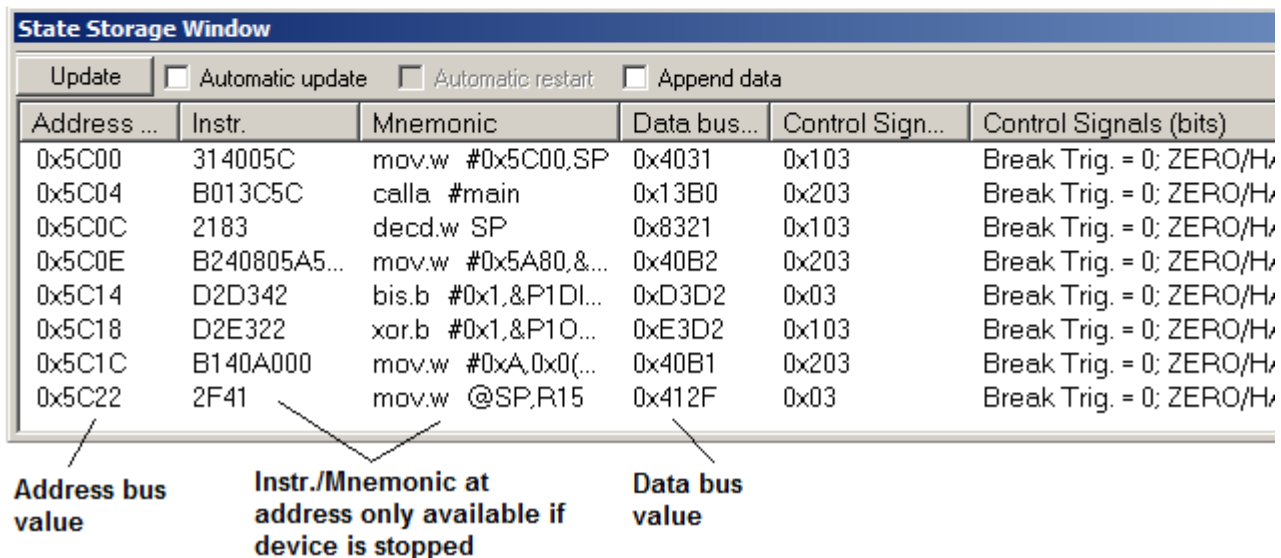
Figure 3. Setting a Range Breakpoint

4 State Storage

State storage can be used to save the information that is on the address or data bus. Additionally, some flags of the CPU, such as Read/Write or Instruction Fetch are stored. There is a total depth of eight entries available in the State Storage buffer. The flexible configuration of this system makes it possible to record the required information into the State Storage buffer very efficiently, so that the information can be saved and displayed.

4.1 Default Configuration

A useful default configuration for state storage is an instruction trace of the last few cycles before a code location. To enable this, select the state storage action on instruction fetch and set State Storage to Stop on trigger. Note that this required a Trigger to be configured as State Storage Trigger. The result can then be seen in the State Storage Window.

Address ...	Instr.	Mnemonic	Data bus...	Control Sign...	Control Signals (bits)
0x5C00	314005C	mov.w #0x5C00,SP	0x4031	0x103	Break Trig. = 0; ZERO/H...
0x5C04	B013C5C	calla #main	0x13B0	0x203	Break Trig. = 0; ZERO/H...
0x5C0C	2183	dec.d.w SP	0x8321	0x103	Break Trig. = 0; ZERO/H...
0x5C0E	B240805A5...	mov.w #0x5A80,&...	0x40B2	0x203	Break Trig. = 0; ZERO/H...
0x5C14	D2D342	bis.b #0x1,&P1DI...	0xD3D2	0x03	Break Trig. = 0; ZERO/H...
0x5C18	D2E322	xor.b #0x1,&P1O...	0xE3D2	0x103	Break Trig. = 0; ZERO/H...
0x5C1C	B140A000	mov.w #0xA,0x0(...	0x40B1	0x203	Break Trig. = 0; ZERO/H...
0x5C22	2F41	mov.w @SP,R15	0x412F	0x03	Break Trig. = 0; ZERO/H...

Figure 4. State Storage

5 Advanced Trigger Options

5.1 Manually Combining Triggers

With the Breakpoint Combiner dialog (Emulator | Advanced menu), two or more individual Triggers can be combined. When defining a complex Trigger with the manual combination of Triggers, the following points should be considered:

- One Trigger (Sub-Trigger) is added to another Trigger (Main-Trigger) with an AND combination. The Main-Trigger then contains the combination of the two Triggers.
- In a combination, only the reaction of the Main-Trigger is used. Sub-Triggers will no longer trigger a reaction on their own.

5.2 Trigger on DMA Events

The Triggers are also able to differentiate between a memory access hosted by the CPU or by the DMA. If a Trigger should be setup for the DMA, this must be done within the Advanced Trigger dialog. It is possible to select from all possible access types and gain full control of all features of the Trigger.

6 Clock Control

A very important part of the debugging system is the flexible clock control (Emulator | Advanced | Clock Control). The clock should, of course, be stopped on emulation hold, especially the main clock for the CPU. Depending on the application, there could be different requirements to the clock for the peripheral modules, such as the UART module that might transfer a character or a timer that is generating a PWM signal for a motor. Merely stopping these peripherals could cancel a communication or even destroy the high-power circuit of the motor control unit. The different clock control modules are listed below, and it is shown how they could be used. [Section 10](#) shows which device has which implementation.

6.1 No Clock Control

Devices with no clock control include the F11x1, F12x, F13x, and F14x.

Modules may be clocked while the CPU is stopped by reading and writing to memory. For example, if a timer interrupt is enabled while the program is executed in single step, the program remains permanently in the interrupt service routine.

NOTE: The only solution to single step with an enabled timer interrupt is to clear the GIE bit in the status register before starting to single step.

6.2 Standard Clock Control

Devices with standard clock control include the F41x.

Standard clock control stops clocks for selected global clocks completely for all modules using these clocks; other modules maintain a continuously running clock. Clock control selection is hardwired. This means that it is possible to select if all of ACLK, MCLK, or SMCLK on the device should be stopped on emulation hold or not.

6.3 Extended Clock Control

Devices with extended clock control include the F15x, F16x, F43x, and F44x.

Extended clock control allows the same control as the standard clock control but additionally the clock could be controlled on the module level. A generally recommended setting for this system is to stop all clocks except the USART, ADC, and flash modules. This setting allows a data transmission, ADC measurement, or a write into the flash memory that was already started to be completed while all other peripheral modules are stopped on a break condition.

7 Cycle Counter

The cycle counter can be used to profile code and count the number of clock cycles required to execute a piece of code.

Open the register view by selecting "View" → "Register" during an active debug session. In the register window, select "CPU Registers" from the drop-down list.

CYCLECOUNTER will show the total number of clock cycles so far. This value cannot be reset. Values CCTIMER1 and CCTIMER2 will also show the number of clocks, but these can be changed or reset manually. CCSTEP will only show the number of clocks spent on the last step or code execution (until the target is either manually halted or a breakpoint is hit).

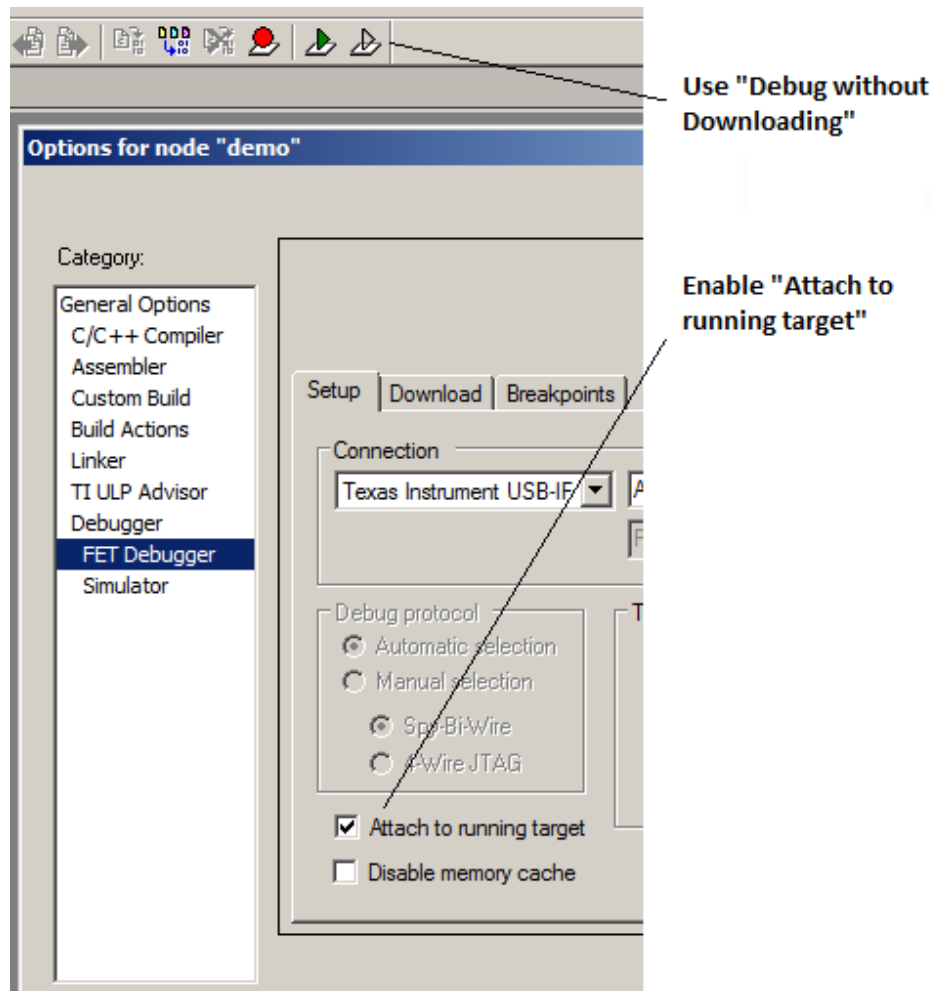
Register	
CPU Registers	
PC	= 0x08020
SP	= 0x02BFA
+ SR	= 0x0000
R4	= 0xFFFF
R5	= 0xFFFF
R6	= 0xFFFF
R7	= 0xFFFF
R8	= 0xFFFF
R9	= 0xFFFF
R10	= 0xFFFF
R11	= 0xFFFF
R12	= 0x00000
R13	= 0x00002
R14	= 0x00182
R15	= 0xFFFF
CYCLECOUNTER	= 40
CCTIMER1	= 13
CCTIMER2	= 17
CCSTEP	= 3

NOTE: On devices with a hardware cycle counter (F5xx, F6xx, FR5xx, FR6xx), the measured cycles can be slightly off due to releasing the device from debug control and then taking it back under debug control. Devices without a hardware cycle counter will require single stepping in the disassembly view.

8 Attach to Running Target

It is possible to attach a debug session to a target that is already executing code without resetting it.

1. Open the project options (right click on the project and then click "Options...") and select "FET Debugger" in the "Category" panel on the left.
2. Check "Attach to running target".



3. Start the debug session without programming the target. Click "Debug without Downloading" to start debugging.

NOTE:

- Breakpoints cannot be used before the target is halted at least once, because the EEM is not enabled before the first halt.
- Attach to Running Target is possible only with external power supply of the MSP MCU.
- If the debug probe powers the MSP MCU, Attach to Running Target is not supported.
- Attach to Running Target is supported only when using the MSP-FET or the MSP-FET430UIF.
- MSP LaunchPad™ development kits do not support this feature.

9 Considerations

- If the JTAG fuse has been blown, access to the emulation logic is disabled.
- When using a complex breakpoint, the CPU is stopped after the instruction causing the break has been executed.
- When a break occurs, the current instruction is always completed before stopping execution.
- The EEM logic cannot prevent an invalid value from being written into a given address or register.
- Hardware registers, like the Timer_A counter (TAR), cannot be used for Triggers unless the CPU is accessing this register during the time the required value are stored in the register.

10 Emulation Module Implementation Summary

The device-dependent EEM implementation level can be found the device-specific data sheet section for the EEM.

Table 1 is an overview of the possible options and lists older devices for which this information is not available in the data sheet. For all other devices, refer to the device-specific data sheet for the EEM details.

Table 1. Emulation Module Overview⁽¹⁾

Device or EEM Version:	F11x1 F12x	F12x2	F13x F14x	F15x F16x F16xx F26xx	F20xx F21x1 F22xx F23xx	F24x	F41x F42x FE42x FW42x F42x0	FG43x	F43x F44x	FG46xx	EEM Version L ⁽²⁾	EEM Version S ⁽²⁾
Triggers (MAB/MDE)	2	2	3	8	2	3	2	2	8	8	8	3
<=/>= ⁽³⁾	–	–	X	X	–	X	–	–	X	X	X	X
R/W	–	–	–	X	–	X	–	–	X	X	X	X
DMA	–	X	–	X	–	–	–	X	–	X	X	X
16/20-bit Mask	–	–	–	X	–	X	–	–	X	X	X	X
Reg.-Write-Trigger	–	–	–	2	–	1	–	–	2	2	2	1
<=/>= ^{(3) (4)}	–	–	–	X	–	X	–	–	X	X	X	X
16/20-bit Mask	–	–	–	X	–	X	–	–	X	X	X	X
Combination	2	2	3	8	2	4	2	2	8	8	8	4
Trigger Sequencer	–	–	–	1	–	–	–	–	1	1	1	–
Reactions												
Break	X	X	X	X	X	X	X	X	X	X	X	X
State Storage	–	–	–	X	–	–	–	–	X	X	X	–
Trace												
Internal	–	–	–	X	–	–	–	–	X	X	X	–
Cycle Counter (HW)	–	–	–	–	–	–	–	–	–	–	2	1
Clock Control												
Global	–	–	–	X	X	X	X	X	X	X	X	X
Modules	–	–	–	X	–	X	–	X	X	X	X	X

⁽¹⁾ Flash devices only

⁽²⁾ The values shown here are examples only—refer to the device-specific data sheet for the values that apply to each device.

⁽³⁾ <=/>= is compare for ==, <>, <=, and >=

⁽⁴⁾ Standard comparison within all devices

11 C-Spy Internal Breakpoint Use

C-SPY itself consumes breakpoints. C-SPY sets a breakpoint if:

- The debugger option Run to has been selected, and any step command is used. These are temporary breakpoints that are set only when the debugger system is running. This means that they are not visible in the Breakpoint Usage window.
- The linker options With I/O emulation modules has been selected. These types of breakpoint consumers are displayed in the Breakpoint Usage dialog box as C-SPY Terminal I/O and libsupport module. The number of options used for this purpose depends on which library you are using. If CLIB is used, three breakpoints are used on putchar, getchar, and exit. The user can disable these in the Breakpoint options dialog box. If DLIB is used, only one breakpoint is required for terminal I/O, and this breakpoint cannot be disabled.
- C-SPY plugin modules, for example modules for real-time operating systems, can also consume additional breakpoints. Specifically, by default, the Stack window consumes one breakpoint. This can be disabled.
- Some user breakpoints consume several low-level breakpoints and conversely, several user breakpoints can share one low-level breakpoint. User breakpoints are displayed in the same way both in the Breakpoint Usage dialog box and in the Breakpoints window, for example Data @[R] callCount. Breakpoints are treated in the same way on all MSP430 devices.

12 References

1. [MSP430 Product Brochure](#)

Examples

The appendix contains some samples of the EEM features that are discussed in this application report. The [code provided with this report](#) does not necessarily present a good coding style or have practical application for a specific part. The code has been designed for easy use of the EEM features. Some features, particularly allowing of nested interrupts and a delay loop inside an Interrupt Service Routine should never be used in a production application.

The menus and dialogs mentioned in the following sections can be accessed at these locations:

- Breakpoint
View | Breakpoint
- New Breakpoint
Right click in the Breakpoint window and select New Breakpoint.
- State Storage Configuration
Emulator | State Storage Control
- State Storage Window
Emulator | State Storage Window

A.1 Break on Write to Address

Breakpoint on write to variable (AdvancedDebugging1.c)

- Set up breakpoint
 - Open the breakpoint window (View | Breakpoints)
 - Right click in breakpoint window
 - In the context menu, select New Breakpoint -> Conditional...
 - Break At: wLoopCounter
 - Type: Address Bus
 - Operator: ==
 - Access: Write
 - Action: Break
 - Other fields unchanged
 - Close the dialog by clicking OK
- Run
- The CPU stops on the first write access to the variable wLoopCounter

Breakpoint on writing specific value to variable (AdvancedDebugging1.c)

- Modify previously set breakpoint
 - Condition - MDB Value: 50
 - Condition - Operator: ==
 - Condition - Access: Write
 - Close the dialog by clicking OK
- Reset the program
- Run

- The CPU stops after the value 50 has been written to wLoopCounter (due to the pipeline CPU, this might not be immediately after the instruction that causes the write access)

A.2 Break on Write to Register

Break on stack overflow (AdvancedDebugging2.c)

- Setup breakpoint
 - Open the breakpoint window (View | Breakpoints)
 - Right click in breakpoint window
 - In the context menu, select New Breakpoint -> Conditional...
 - Breakpoint At: SP
 - Type: Register
 - Operator: <=
 - Access: Write
 - Action: Break
 - Condition - Register Value: 0x4350
 - Other fields unchanged
 - Close the dialog by clicking OK
- Run
- The CPU stops when the function 'DummyStackFill' is called, because during the initialization, 100 words are pushed to the Stack and the stack pointer goes below the set value of 0x4350

A.3 Break on Write to Flash

Break on write access to Flash (AdvancedDebugging3.c)

- Set up breakpoint
 - Open the breakpoint window (View | Breakpoints)
 - Right click in breakpoint window
 - In the context menu, select New Breakpoint -> Range...
 - Start: 0x4400
 - Range Delimiter: End Value - 0xFFFF
 - Type: Address
 - Access: Write
 - Action: Break
 - Action when: Inside range
 - Close the dialog by clicking OK
- Run
- The CPU stops after the first write access to the Flash memory (= 'wDataInFlash++' instruction, due to the pipeline CPU, this might not be immediately after the instruction)

NOTE: The break occurs after a delay of approximately 5 seconds.

A.4 Break on Access of Invalid Memory

Break on any access to BSL memory (AdvancedDebugging3.c)

- Set up breakpoint
 - Open the breakpoint window (View | Breakpoints)
 - Right click in breakpoint window
 - In the context menu, select New Breakpoint -> Range...
 - Start: 0x1000
 - Range Delimiter: End Value - 0x1800
 - Type: Address
 - Access: Read/Write
 - Action: Break
 - Action when: Inside range
 - Close the dialog by clicking OK
- Run
- The CPU stops after the first read access to the BSL memory (= `'*(unsigned int *)BSL_START'`, due to the pipeline CPU, this might not be immediately after the instruction)

NOTE: The break occurs after a delay of approximately 10 seconds.

A.5 Break if Fetch is Out of Allowed Area

Break on instruction fetch outside of main memory (AdvancedDebugging3.c)

- Set up breakpoint
 - Open the breakpoint window (View | Breakpoints)
 - Right click in breakpoint window
 - In the context menu, select New Breakpoint -> Range...
 - Start: 0x4400
 - Range Delimiter: Length - 0xFFFF
 - Type: Address
 - Access: Fetch
 - Action: Break
 - Action when: Outside range
 - Close the dialog by clicking OK
- Run
- The CPU stops inside the Delay function located in Info memory

A.6 State Storage: Trace

Get trace of the last eight instructions (AdvancedDebugging4.c)

- Set up State Storage
 - Open State Storage Control menu (Emulator | State Storage Control)
 - Select Enable State Storage
 - Storage action on: Instruction Fetch
 - Trigger action: None
 - Confirm settings by clicking Apply
- Open the State Storage Window (Emulator | State Storage Window)
- Run
- Break
- The State Storage window displays the last eight instructions executed by the CPU

Trace variable writes (AdvancedDebugging5.c)

- Set up breakpoint or storage trigger
 - Open the breakpoint window (View | Breakpoints)
 - Right click in breakpoint window
 - In the context menu, select New Breakpoint -> Conditional...
 - Break At: wLoopCounter
 - Type: Address Bus
 - Operator: ==
 - Access: Write
 - Action: State Storage Trigger (do not select Break)
 - Other fields unchanged
 - Close the dialog by clicking OK
- Set up State Storage
 - Open State Storage Control menu (Emulator | State Storage Control)
 - Select Enable State Storage
 - Storage action on: Trigger
 - Confirm settings by clicking Apply
 - Open the State Storage Window (Emulator | State Storage Window)
 - Check Automatic update
 - Check Automatic restart
- Run
- The Trace Buffer inside the MSP430 is read and displayed
- The information written to the variable wLoopCounter can be displayed without stopping or influencing the program execution
- Some values might not be filled when the buffer is displayed and contain MAB and MDB values of 0.

Revision History

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

Changes from July 29, 2015 to September 6, 2016

Page

- Removed "Section 4.2 Real Time Watch" and "Section 5 Trigger Sequencer" (features not supported in this version).... 7
- Added the last four list items to the note at the end of [Section 8, Attach to Running Target](#) 9

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as "components") are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI's terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers' products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers' products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI's goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or "enhanced plastic" are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have **not** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products

Audio	www.ti.com/audio
Amplifiers	amplifier.ti.com
Data Converters	dataconverter.ti.com
DLP® Products	www.dlp.com
DSP	dsp.ti.com
Clocks and Timers	www.ti.com/clocks
Interface	interface.ti.com
Logic	logic.ti.com
Power Mgmt	power.ti.com
Microcontrollers	microcontroller.ti.com
RFID	www.ti-rfid.com
OMAP Applications Processors	www.ti.com/omap
Wireless Connectivity	www.ti.com/wirelessconnectivity

Applications

Automotive and Transportation	www.ti.com/automotive
Communications and Telecom	www.ti.com/communications
Computers and Peripherals	www.ti.com/computers
Consumer Electronics	www.ti.com/consumer-apps
Energy and Lighting	www.ti.com/energy
Industrial	www.ti.com/industrial
Medical	www.ti.com/medical
Security	www.ti.com/security
Space, Avionics and Defense	www.ti.com/space-avionics-defense
Video and Imaging	www.ti.com/video

TI E2E Community

e2e.ti.com