Application Report
SLAA393F–June 2008–Revised September 2016

Advanced Debugging Using the Enhanced Emulation Module (EEM) With Code Composer Studio Version 6

Stefan Schauer *MSP430*

ABSTRACT

This document describes the benefits of the Enhanced Emulation Module (EEM) advanced debugging features that are available in the MSP430™ devices and how they can be used with Code Composer Studio™ IDE(CCS) version 6 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 demonstrated.

Project collateral and source code discussed in this application report can be downloaded from www.ti.com/lit/zip/slaa393.

Contents

List of Figures

List of Tables

MSP430, Code Composer Studio, LaunchPad are trademarks of Texas Instruments.
All other trademarks are the property of their respective owners.

Copyright © 2008–2016, Texas Instruments Incorporated

1 Introduction

Within every MSP430 flash-based microcontroller, there is on-chip debug logic. This Enhanced Emulation Module (EEM) provides different levels of debug features, depending on the device being used. This application report describes how the EEM can be used to solve typical debug 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 or 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 and over/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 works nonintrusively, meaning that it does not use or lock any resources targeted for the application, such as registers, memory, or interrupts.

CCS v6 supports the setting of frequently used debug features (particularly the complex ones) with use cases, such as stack overflow. These features are also described in this application report.

NOTE: All examples in this application report are based on CCS v6. Many of the other debuggers have the same or similar features. For details about using these other debuggers, see the user's guide for the specific 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
- Trace

There are two fundamentally 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 a 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 Breakpoints

Triggers are used to configure breakpoints. This 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.

Copyright © 2008–2016, Texas Instruments Incorporated

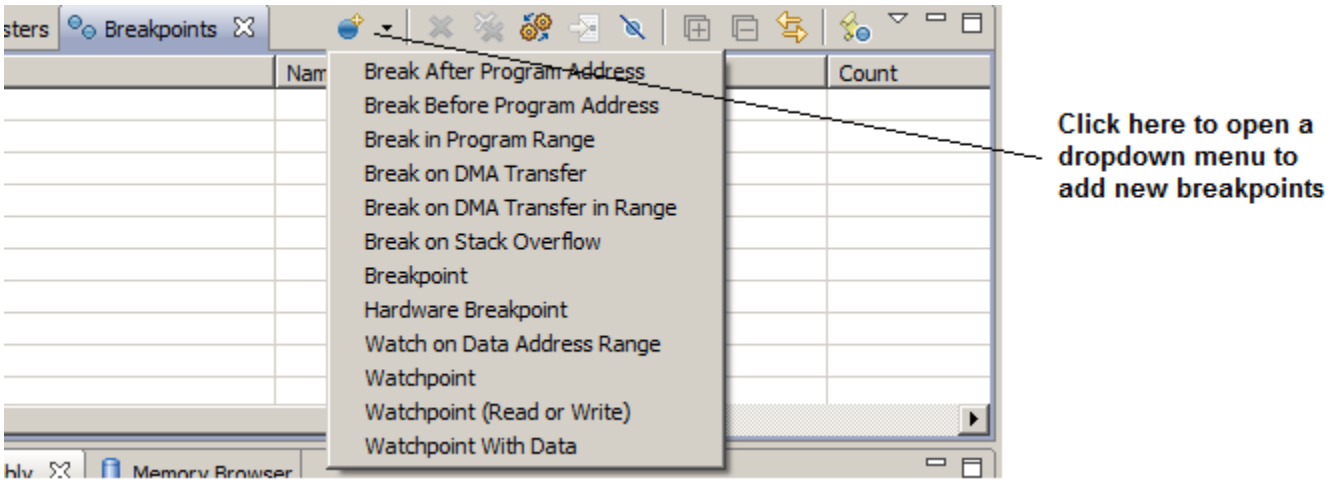

Figure 1. Breakpoint Dialog

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 and/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.

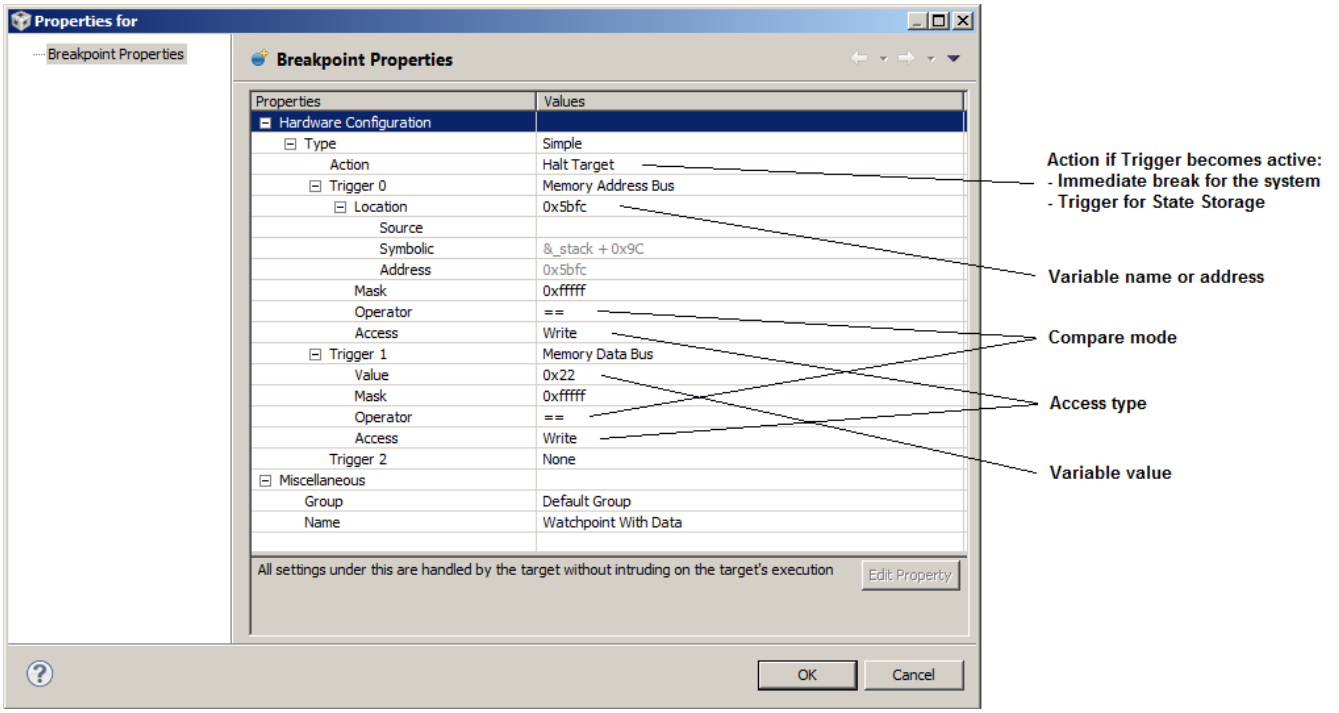

Figure 2. Conditional Breakpoint Properties Dialog

Copyright © 2008–2016, Texas Instruments Incorporated

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 assembly, where the programmer has complete control over the usage of the registers. Dedicated registers might be used for a variable or a system state flag. One register is critical and worth observing very carefully, the stack pointer. 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, which stops the microcontroller when the stack pointer is equal or below a certain value, helps detect such problems quite easily. To set up this Trigger, one of the register Triggers is used.

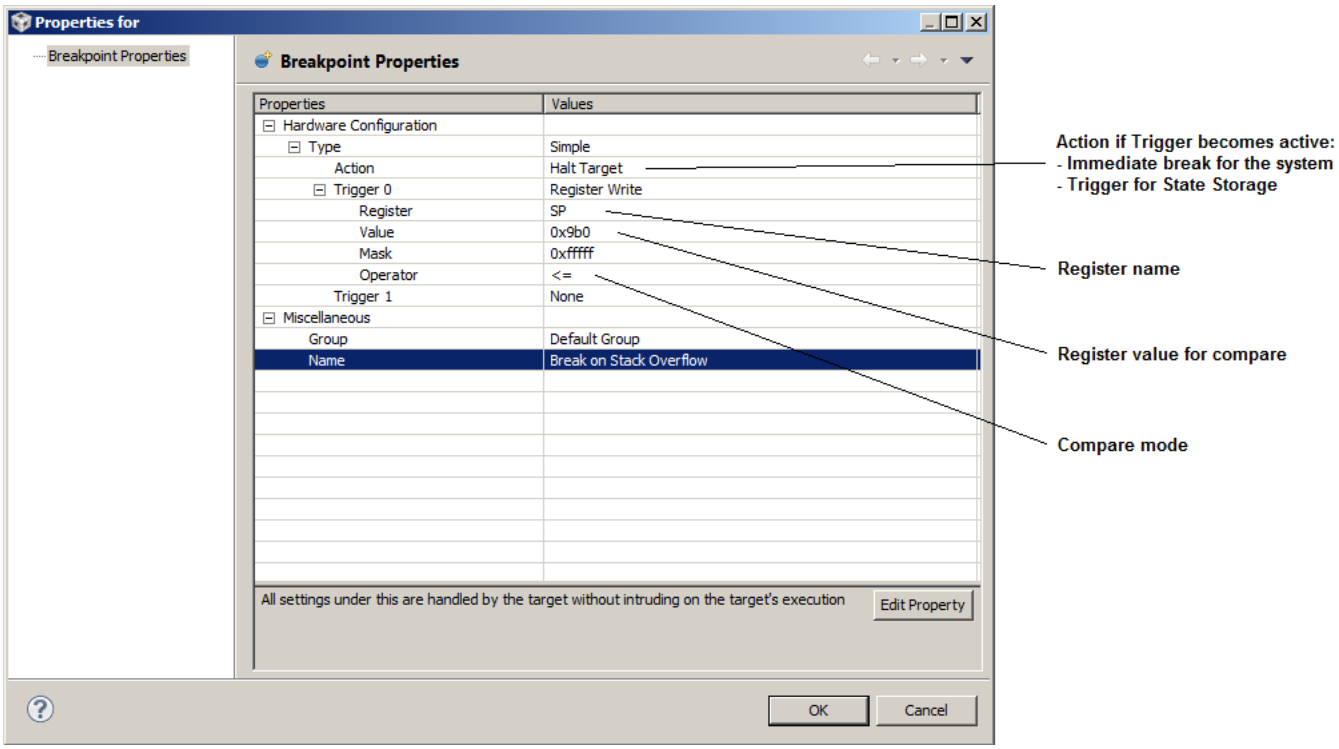

Figure 3. Break on Stack Properties Dialog

NOTE: When using an MSP430X CPU (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/cleared in the specified register.

Copyright © 2008–2016, Texas Instruments Incorporated

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, during program execution, a write access into the flash memory area occurs. 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/Write to Invalid Memory – This check could be used to evaluate if, during program execution, any attempt to access data in invalid memory range occurs.
- 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 Trigger 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.

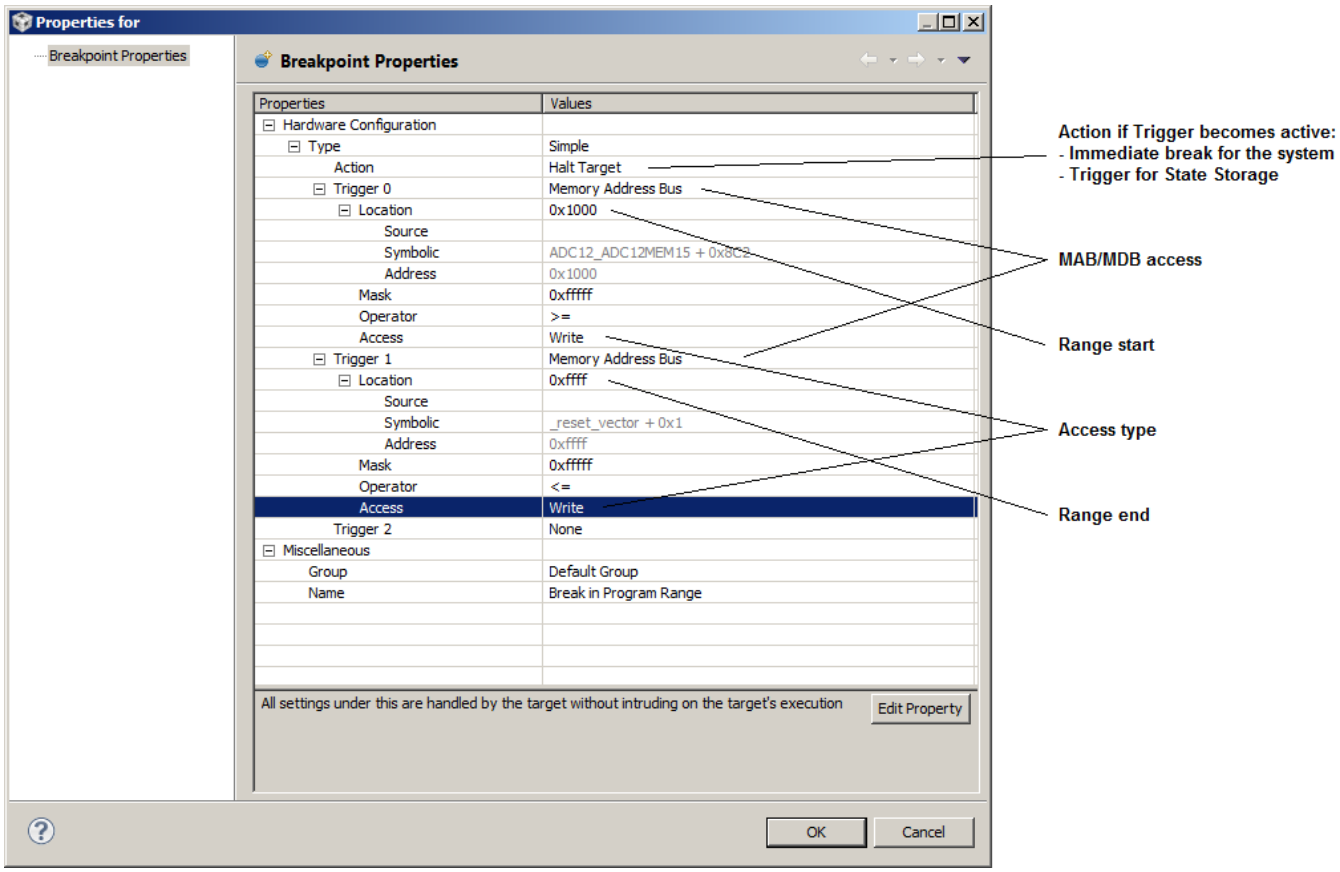

Figure 4. Break on Data Address Range Properties Dialog

Copyright © 2008–2016, Texas Instruments Incorporated

4 Trace

Trace can be used to save the information that is on the address and 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 Trace buffer. The flexible configuration of this system makes it possible to record the required information into the Trace buffer very efficiently, so that the required information can be saved and displayed.

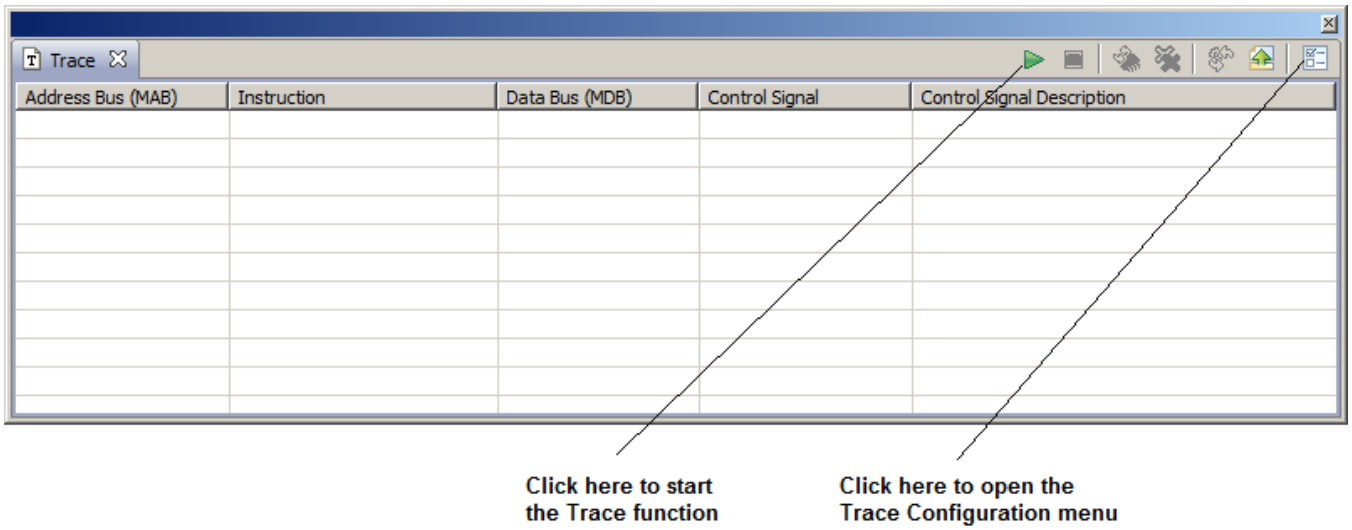

Figure 5. Trace Window

4.1 Default Configuration

A useful default configuration would be an instruction trace of the last few cycles before a code location. To enable this, configure Trace to Start Immediately and Stop On Trigger and select Filter on Instruction Fetch. Note that this requires a Trigger to be configured to either Trigger Storage or Halt and Trigger Storage. The result can then be seen in the Trace Window.

5 Advanced Trigger Options

5.1 Manually Combining Triggers

With the Breakpoint Properties dialog, two or more individual Triggers can be combined by defining additional Triggers (for example, Set Trigger1 to Memory Data Bus). 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 do not 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 set up 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.

Copyright © 2008–2016, Texas Instruments Incorporated

6 Clock Control

A very important part of the debugging system is the flexible 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. Table 1 shows which device has which implementation.

The clock control settings could be changed under

Project | Properties

TI Debug Settings | Target Tab

MSP430 Properties | Clock Control

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 (Global)

Devices with standard clock control include the F41x.

Standard clock control stops the 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 (Modules)

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 in a break condition.

Copyright © 2008–2016, Texas Instruments Incorporated

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.

Enable display of clock cycles by selecting "Run" → "Clock" → "Enable" during an active debug session (see Figure 6).

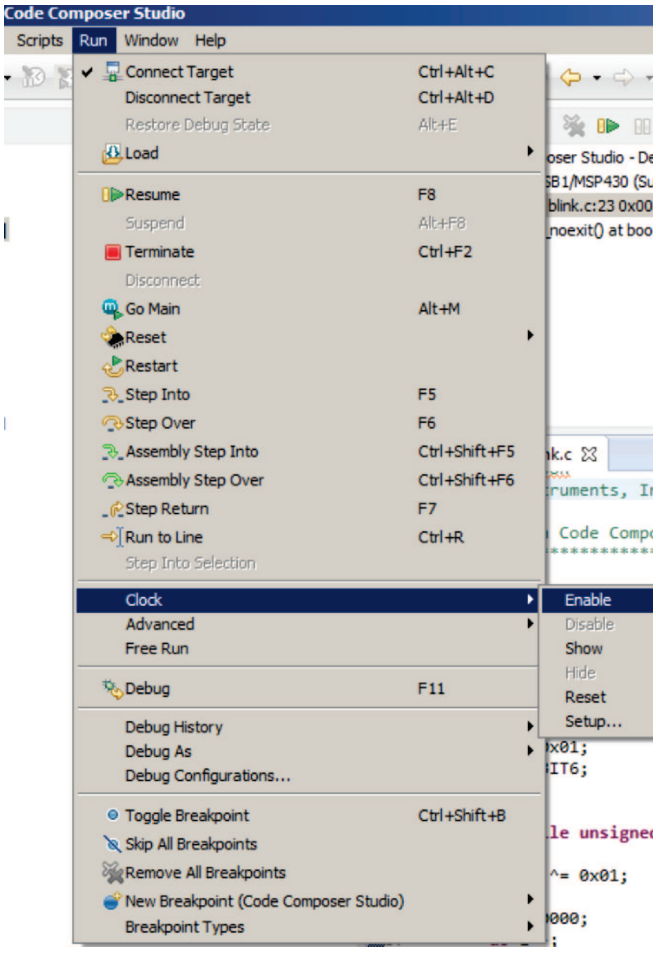

Figure 6. Cycle Counter

A clock icon will appear at the bottom in the status bar (see Figure 7), showing the number of cycles accumulated so far.

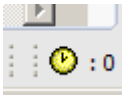

Figure 7. Cycle Counter Icon

The counter can be reset ("Run" → "Clock" → "Reset") and will keep track of clock cycles when executing/stepping through code.

NOTE: On devices with a hardware cycle counter ('F(R)5xx/6xx), the measured cycles can be slightly off due to releasing the device from and taking it back under debug control. Enabling the cycle counter will cause CCS to single step during code execution.

Copyright © 2008–2016, Texas Instruments Incorporated

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. Click "Run" and then click "Debug configurations...".
2. Right click the configuration in the panel on the left, and then click "Duplicate" in the popup menu (see Figure 8).

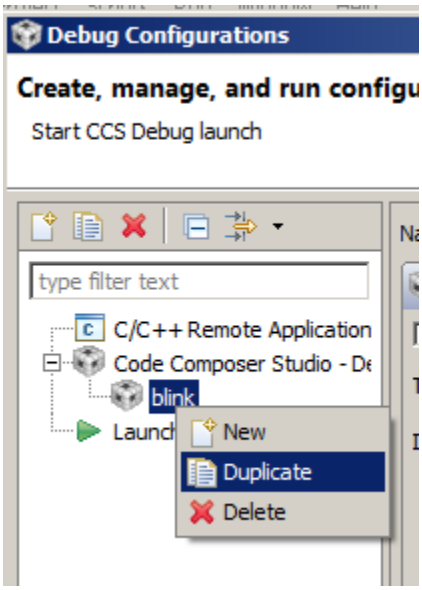

Figure 8. Attach Duplicate

3. Select the "Program" tab.
4. Select "Load symbols only" (see Figure 9).

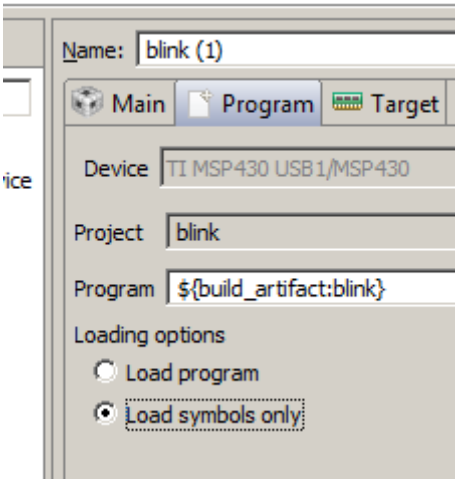

Figure 9. Attach Symbols

5. Select the "Target" tab.

Copyright © 2008–2016, Texas Instruments Incorporated

6. In "Program/Memory Load Options", uncheck "Halt the target on a connect" (see Figure 10).

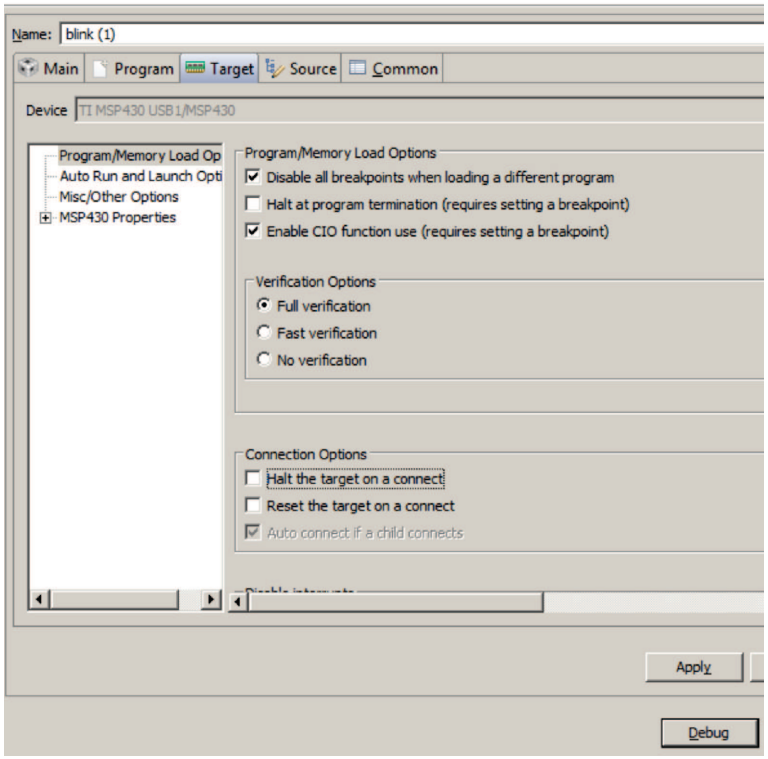

Figure 10. Attach Halt

7. Click the "Debug" button.
8. For later debug sessions, select the new configuration.

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.

Copyright © 2008–2016, Texas Instruments Incorporated

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

Copyright © 2008–2016, Texas Instruments Incorporated

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 been used in a production application.

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

- Breakpoint
  Window | Show View | Breakpoints
- New Breakpoint
  Click on down arrow right from new Breakpoint icon
- Trace Window
  Window | Show View | Trace
- Trace Configuration
  Click on Configuration Icon in Trace Window

NOTE: The examples have been set up for a MSP430F5529. If using another device, make sure to use the correct addresses for the available memory.

A.1 Break on Write to Address

Breakpoint on write to variable (AdvancedDebugging1.c)

- Set up breakpoint
  - Open the Breakpoint menu (Window | Show View | Breakpoints)
  - Click drop down menu for new Breakpoint and select Watchpoint (Read or Write)
    - Location: &wLoopCounter
    - Access Type: Write
  - 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)

- Set up breakpoint
  - Open the Breakpoint menu (Window | Show View | Breakpoints)
  - Click drop down menu for new Breakpoint and select Watchpoint with Data
    - Location: &wLoopCounter
    - Data Value: 50
    - Access Type: Write
  - Close the dialog by clicking OK
- Run
- The CPU stops after the value 50 has been written to the variable wLoopCounter (due to the pipeline

Copyright © 2008–2016, Texas Instruments Incorporated

CPU, this might not be immediately after the instruction causing the write access).

A.2 Break on Write to Register

Break on stack overflow (AdvancedDebugging2.c)

- Set up breakpoint
  - Open the Breakpoint menu (Window | Show View | Breakpoints)
  - Click drop down menu for new Breakpoint and select Break on Stack Overflow
    - Stack Min: &_stack
  - 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 the stack size.

A.3 Break on Write to Flash

Break on write access to Flash (AdvancedDebugging3.c)

- Set up breakpoint
  - Open the Breakpoint menu (Window | Show View | Breakpoints)
  - Click drop down menu for new Breakpoint and select Watch on Data Address Range
    - Range Start: 0x4400 (Start of Main Flash Memory)
    - Range End: 0xFFFF
    - Access Type: Write
  - Close the dialog by clicking OK
- Run
- The CPU stops after the first write access to the Flash memory (= '(*pwDataInFlash++)' 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 menu (Window | Show View | Breakpoints)
  - Click drop down menu for new Breakpoint and select Watch on Data Address Range
    - Range Start: 0x1000
    - Range End: 0x1800
    - Access Type: No Instruction Fetch (= Read/Write)
  - 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.

Copyright © 2008–2016, Texas Instruments Incorporated

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 menu (Window | Show View | Breakpoints)
  - Click drop down menu for new Breakpoint and select Break Before Program Address
    - Location: 0x4400 (start of main Flash memory)
  - Close the dialog by clicking OK
- Run
- The CPU stops inside the Delay function located in Info memory

A.6 Trace

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

- Set up breakpoint
  - Place a breakpoint on line 72 by double clicking the line number
- Configure Storage action
  - Modify the breakpoint (right click on breakpoint | Properties...)
    - Action: Halt and Trigger Storage
  - Close the dialog by clicking OK
- Setup Trace
  - Open the trace menu (Window | Show View | Trace)
    - Click "Configure Properties" in the trace view
    - Select: Start immediately and stop on trigger
    - Select: Instruction Fetch
  - Close the dialog by clicking OK
  - Click Start Tracing
- Run
- Trace information displays when the breakpoint is hit.

Trace variable writes (AdvancedDebugging5.c)

- Set up breakpoint
  - Open the Breakpoint menu (Window | Show View | Breakpoints)
  - Click drop down menu for new Breakpoint and select Watchpoint (Read or Write)
    - Location: &wLoopCounter
    - Access Type: Write
  - Close the dialog by clicking OK
- Configure Storage action
  - Modify the breakpoint (right click on breakpoint | Properties...)
    - Action: Trigger Storage
  - Close the dialog by clicking OK
- Setup Trace
  - Open the trace menu (Window | Show View | Trace)
    - Click "Configure Properties" in the trace view
    - Select: Store on Trigger and stop when buffer full
  - Close the dialog by clicking OK
  - Select: Restart Automatically
  - Click Start Tracing

Copyright © 2008–2016, Texas Instruments Incorporated

- Run
- The Trace Buffer inside the MSP430 is read and displayed
- The information written to the variable wLoopCounter can be displayed without stopping of influencing the program execution

Copyright © 2008–2016, Texas Instruments Incorporated

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
• Added the last four list items to the note at the end of Section 8, *Attach to Running Target*	10

Copyright © 2008–2016, Texas Instruments Incorporated

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		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2016, Texas Instruments Incorporated