TEXAS INSTRUMENTS

*Application Report*
*SLAA263−September 2005*

# *Using the MSP430 EEM debug feature with IAR 3.30*

*Stefan Schauer* *AEC/MSP430*

**ABSTRACT**

This document shows the benefits of the Enhanced Emulation Module(EEM) advanced debug features of the MSP430 devices and how they can be used using IAR embedded workbench V3.30 (full and Kickstart) software development tool.

EEM advanced debugging features supports both precision analog and full speed digital debug. The configuration of the debug environment for maximum control as well as the usage of the embedded trace capability is shown. Some techniques that allow rapid development and design-for-testability are demonstrated.

## 1 Introduction

Within every MSP430 Flash based Microcontroller there is an On-Chip Debug Logic. This Enhanced Emulation Module (EEM) provides different levels of debug features, depending on the device being used. This document will show how it can be used to solve typical debug problems.

In general the following features are available:

- 2 to 8 hardware breakpoints
- Complex breakpoints
- Break if read/write at specified address
- Protection of read/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 DCO dependencies such as temperature and voltage

**Note:**

All shown examples are based on IAR version 3.21 and 3.30. Many of the other Debuggers have the same or similar features. For details about using this please have a look into the Users guide of the dedicated debugger.

## 2 Tigger

The event control in the EEM of the MSP430 system consist of so called ‘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 use for the control of the following functional blocks of the EEM:

- Breakpoints
- State Storage
- Sequencer

There are two fundamental different types of Triggers, one for the Address und Data bus and the second one for the CPU Registers. It is also possible to define under which condition the trigger should be active. Such conditions could be read, write or fetch of an instruction. These triggers can also be combined so that a trigger gives a signal if a particular value is written into a dedicated address.

## 3 Breakpoint

Triggers will be used to configure breakpoints. This very flexible system allows the definition of various different but 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 Breakpoints one Trigger will be used.

### *3.2 Data Breakpoints*

Another type of breakpoints – so called data breakpoints can be configured by using one or two triggers. A data break could be used to check of a certain value on the address bus (memory address of the variable) combined with a read and/or write signal. It could also be enhanced so that the break only occurs if a dedicated value is read/ written into this address. This value will then be checked on the data bus.

For a Data Breakpoint without a value one Trigger and for a Data Breakpoint with Value two Triggers will be used.

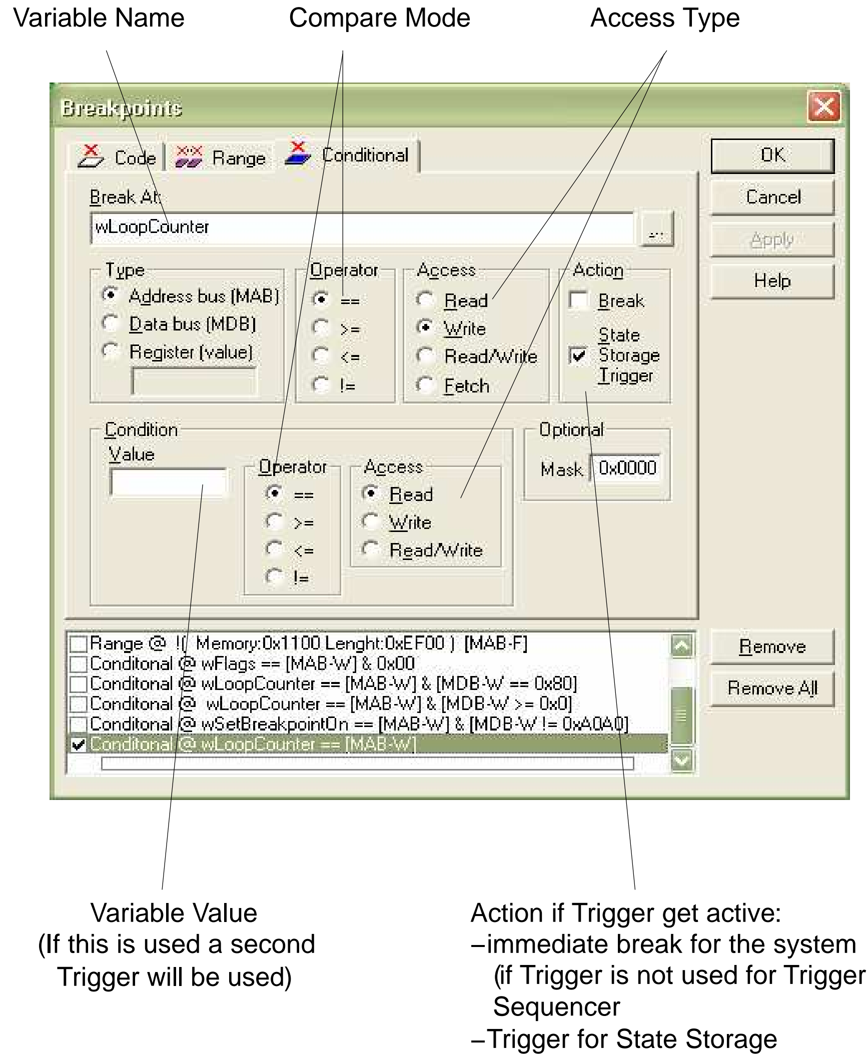


**Figure 1. Conditional Breakpoint Dialog**

### 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 usage of the registers. Dedicated registers might be used for a variable or a system state flag. One register is very critical and worth observing very carefully - the stack pointer. If there is a problem within the program, which allows the stack to run into the data area, it is often very difficulty to find the problem with normal debugging features as the symptoms may change each time the program execution is started. A simple breakpoint, which does stop the microcontroller when the stack pointer reaches a certain value, or is below a certain value will help to detect such problems quite easily. To setup this Trigger one of the Register Triggers used.

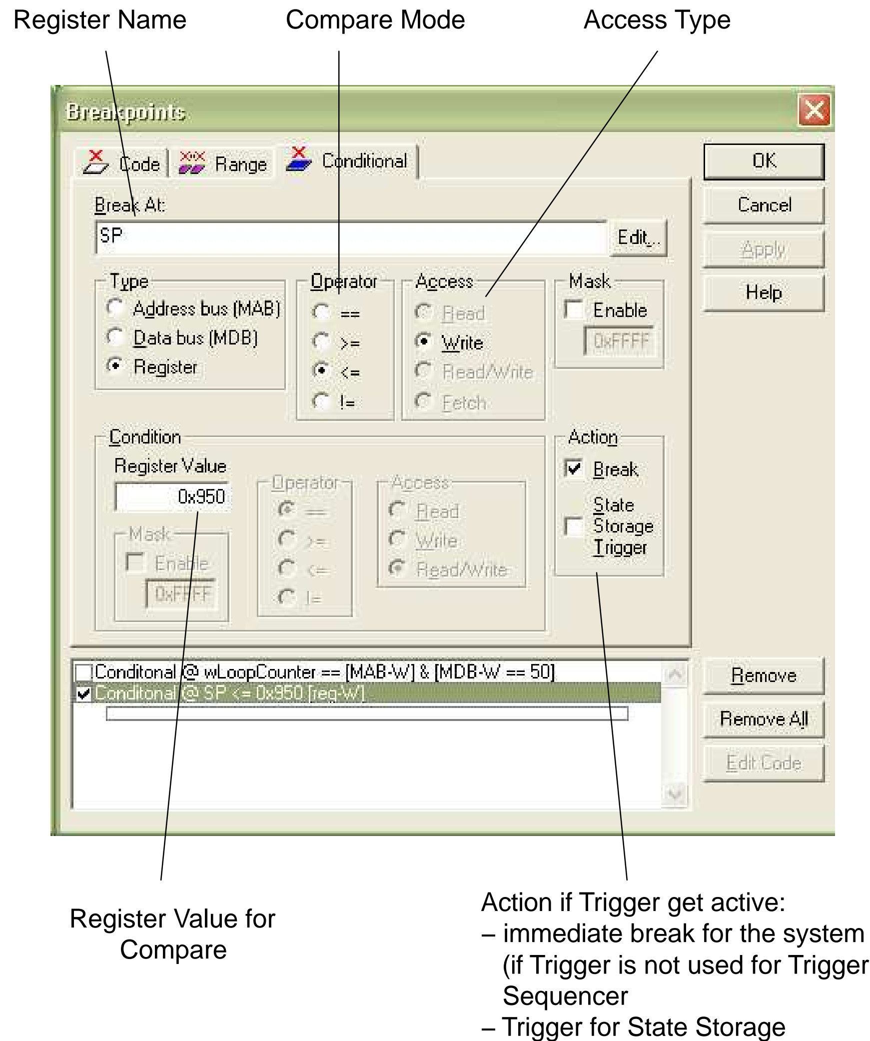


**Figure 2. Conditional Breakpoint Dialog for Register**

## 3.4 Mask Register

The Mask Register in the Conditional Breakpoint Dialog allows to limit the check only to certain bits (according to the mask) of the register. This could be e.g. a check if only a certain bit has been set/reset in the specified Register.

## 3.5 Range Breakpoints

Range Breakpoints are necessary to check an access to the dedicated memory range. Some possible conditions could be:

- Break on Write to Flash – This allows to check if during the program execution a write access into the Flash memory area occurs. In many cases this is not or only under certain circumstances allowed 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 the program execution any attempt to access data in invalid memory range happens.
- Break on Instruction Fetch out of Range – This breakpoint could stop the CPU if it does fetch 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 has to be combined with a write or read Trigger on the Variable address or the Trigger Sequencer. Otherwise any value which does appear on the Data Bus and is in this range, e.g. an instruction, could stop the CPU.

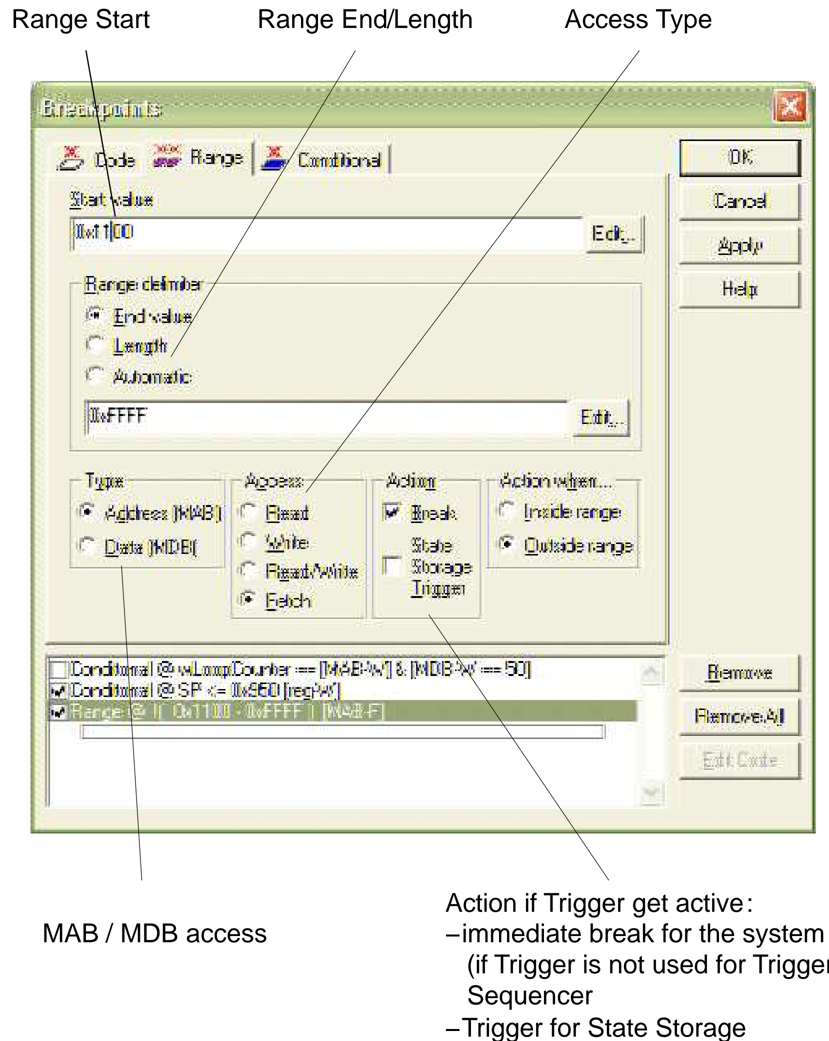


**Figure 3. Setting a Range Breakpoint**

## 4 State Storage

The State Storage could be used to save the information which is on the Address and Data Bus. Additionally some Flags of the CPU, like Read/Write or Instruction Fetch will be stored. There is a total depth of 8 entries available in the State Storage Buffer. The very flexible configuration of this system makes it possible to record the required information into the State Storage Buffer very efficient so that the required information could be saved and displayed.

### *4.1 Default Configuration*

A useful default configuration would be an instruction trace of the last few cycles. To enable this you need to select the state storage action on Instruction Fetch and enable the Buffer wrap around option to continuously save the executed instructions. The result can then be seen in the State storage window. It is possible to update this window during the program execution. This could give some feedback on the current program execution and program status.

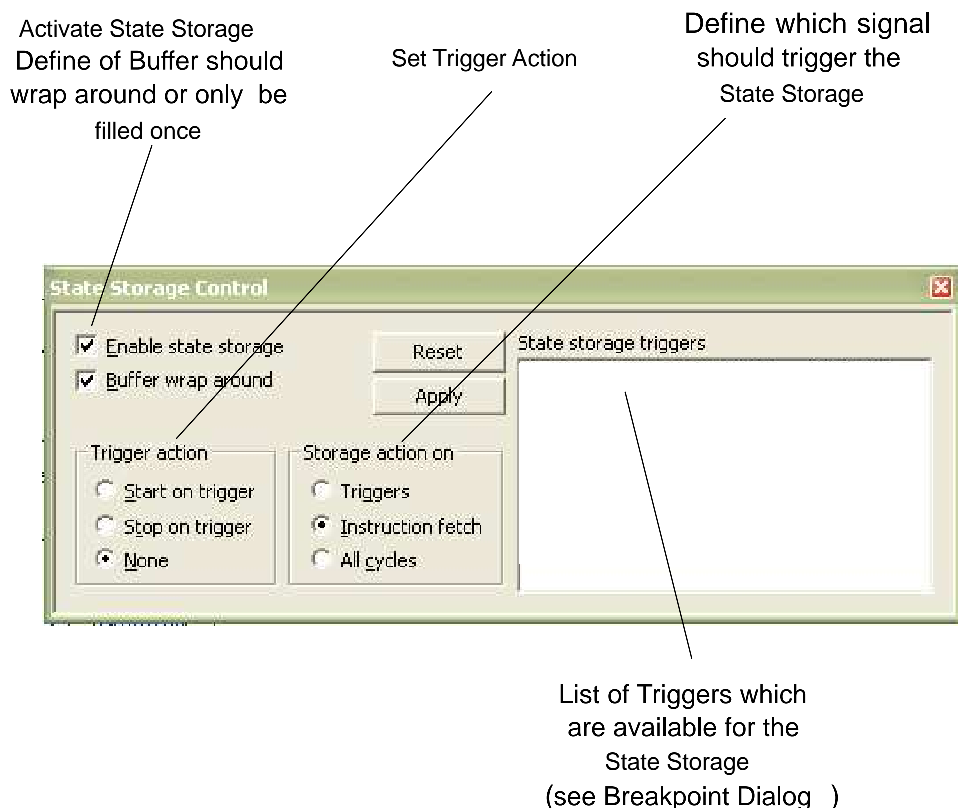


**Figure 4. State Storage Configuration Window**

## 4.2 Real Time Watch

A special configuration of the state storage system allows some kind of Real Time Watch implementation. A trace on a trigger which is set to variable read and/or write creates this real-time watch system.

A Real Time Watch for a Variable could be setup with the following steps:

Set State Storage Trigger on Variable read or write in the Breakpoint Menu (Note: disable the Break for this Trigger)

In the State Storage Dialog following options have to be set:

- Buffer wrap around
- Storage should trigger at Triggers

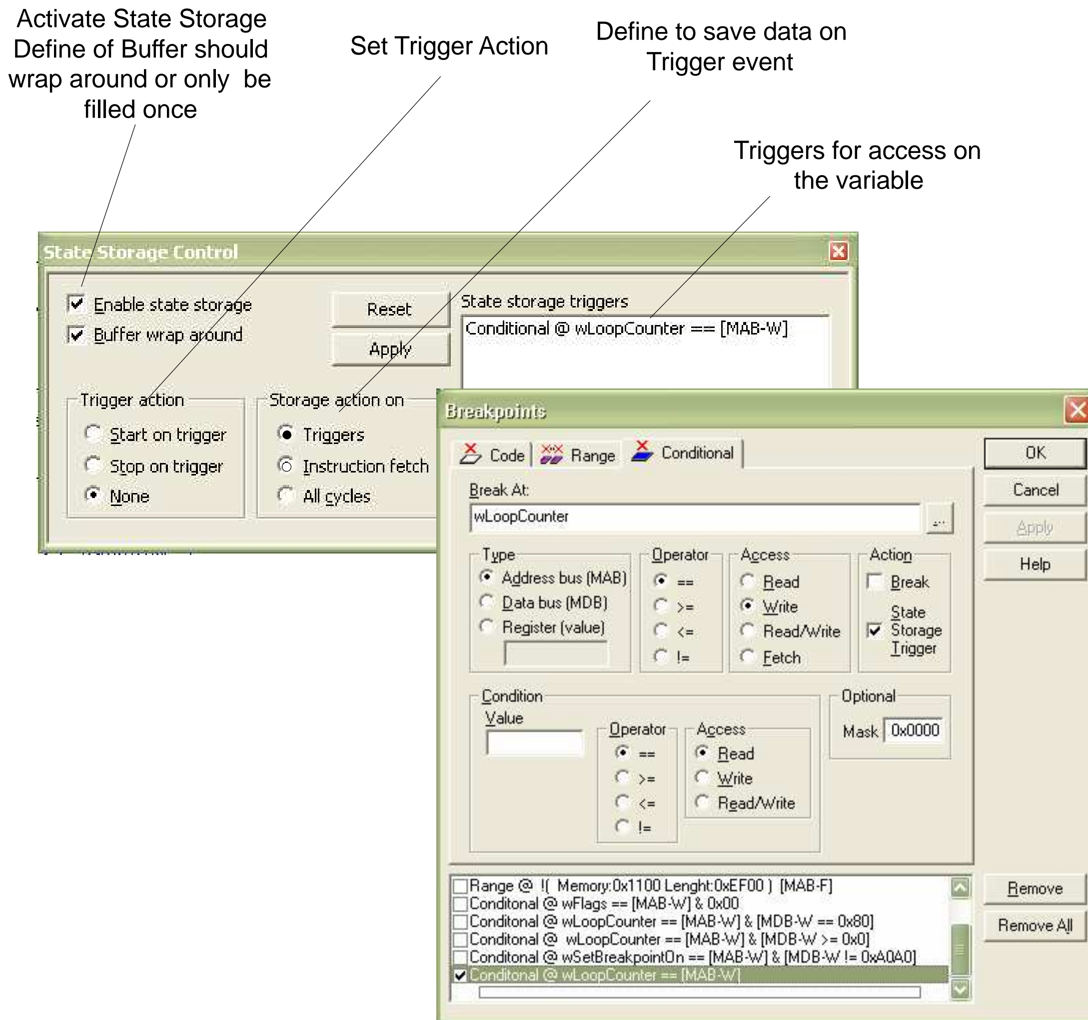


**Figure 5. Setup of the Real Time Watch Feature**

Caution: With more than one variable watch, a variable which is addressed infrequently may not be captured as it may not be captured by the Debugger before it is overwritten by other entries into the State Storage Buffer after a wrap around.

## 5 Trigger Sequencer

The Trigger Sequencer allows the definition of a certain sequence of Trigger signals before an event is accepted for a break or state storage event. Within the Trigger Sequencer it is possible to use the following features:

- 4 states (State 0 to State 3)
- 2 transitions per state (Transition Trigger a and b)
- Each transition can be programmed to move to any state

The Trigger Sequencer always starts at State 0 and has to execute to State 3 to generate an action. If State 1 and State 2 or not required for the function they can be bypassed.

## 5.1 Simple Trigger Sequencer

With this simple dialog a linear program sequence can be setup which has to be executed before a trigger is accepted for a break or state storage event.

This is useful if a certain event only occurs after a given sequence in the program has been executed.

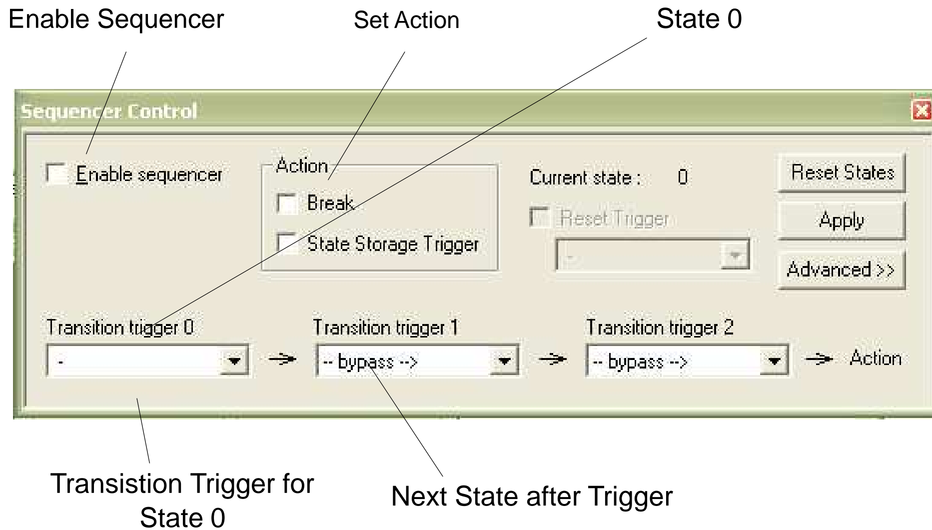


**Figure 6. Simple Trigger Sequencer Dialog**

## 5.2 Advanced Trigger Sequencer

The advanced dialog allows the full configuration of the Sequencer. This is especially useful if a Trigger may activate a sequence but another Trigger could reset or deactivate the sequence again. This may be in a communication routine where only after a certain command has been received a Trigger in the transmit code should stop the CPU but if a 3rd trigger happens it should first wait again for the first trigger.

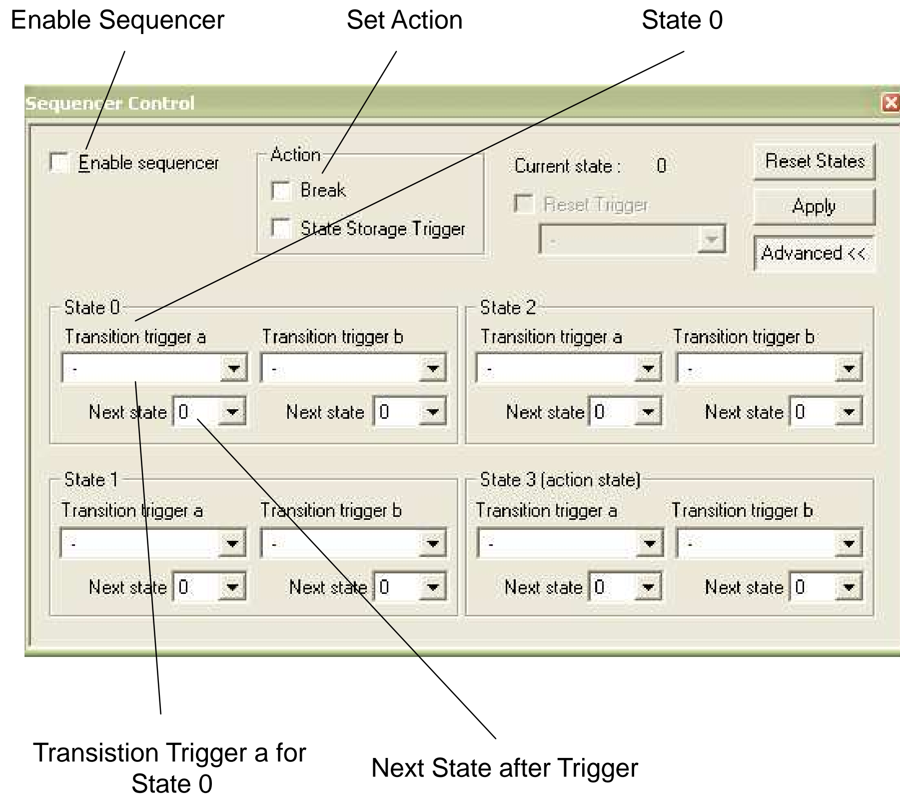


**Figure 7. Advanced Trigger Sequencer Dialog**

## 6 Advanced Trigger Options

### *6.1 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 will then contain the combination of the two Triggers.
- The Sub-Trigger stays unmodified in the system. A break action set on the Sub-Trigger stops execution independent from the Main-Trigger. This means that in most cases the Break event should be disabled for all Sub-Triggers.

### *6.2 Trigger on DMA Events*

The Triggers are also able to differ between a memory access hosted by the CPU or by the DMA. If a Trigger should be setup for the DMA this has to be done within the Advanced Trigger Dialog. It is possible to select from all possible Access Types and get the full control of all features of the Trigger.

# 7 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, but depended on the application there could be different requirements to the clock for the peripheral modules, like the UART module which might transfer a character, or a timer which 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. Below the different Clock Control modules are listed and it is shown how they could be used. Table 1 shows which device has which implementation.

## 7.1 No Clock Control (e.g., 'F11x1,'F12x,'F13x/14x)

Modules may be clocked while CPU is stopped by reading and writing to memory. So if e.g. a Timer interrupt is enabled while the program is executed in single step the program will be permanently in the ISR.

**Note:**

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

## 7.2 Standard Clock Control (e.g., 'F41x)

Standard Clock Control stops clocks for selected modules completely; 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.

## 7.3 Extended Clock Control (e.g., 'F15x/16x,'F43x/44x)

Extended Clock Control allows the same control as the standard clock control but additionally the clock could be controlled on module level. A general recommended setting for this system is to stop all clocks except the USART, ADC and Flash modules. With this setting it does allow that a started data transmission or ADC measurement or write into the Flash memory could be finished by the system while all other peripheral modules are stopped on a break condition.

# 8 Considerations

- If the JTAG Fuse has been blown the access to the Emulation Logic is disabled.
- When using a complex breakpoint, the CPU will be stopped after the instruction causing the break has been executed.
- When a break occurs, the current instruction will always be completed first before stopping the 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), can not be used for Triggers unless the CPU is accessing this register during the time the required value are stored in the register.

# 9 Emulation Module Implementation Summary

The following table shows in a summary the differences of the implementations of the Emulation modules in the different devices.

**Table 1. Emulation Module Overview[(1)]**

| DEVICE | F11x1 F12x | F12x2 | F13x F14x | F15x F16x | F21x1 | F41x F42x F42x0 | FE42x FW42x | FG43x | F43x F44x |
|---|---|---|---|---|---|---|---|---|---|
| **Triggers (MAB/MDE)** | 2 | 2 | 3 | 8 | 2 | 2 | 2 | 2 | 8 |
| <=/>=[(2)] | – | – | X | X | – | – | – | – | X |
| R/W | – | – | – | X | – | – | – | – | X |
| DMA | – | X | – | X | – | – | – | X | – |
| 16bit Mask | – | – | – | X | – | – | – | – | X |
| **Reg.-Write-Trigger** | – | – | – | 2 | – | – | – | – | 2 |
| <=/>=[(2)(3)] | – | – | – | X | – | – | – | – | X |
| 16bit Mask | – | – | – | X | – | – | – | – | X |
| **Combination** | 2 | 2 | 3 | 8 | 2 | 2 | 2 | 2 | 8 |
| **Trigger Sequencer** | – | – | – | 1 | – | – | – | – | 1 |
| **Reactions** | | | | | | | | | |
| Break | X | X | X | X | X | X | X | X | X |
| State Storage | – | – | – | X | – | – | – | – | X |
| **Trace** | | | | | | | | | |
| Internal | – | – | – | X | – | – | – | – | X |
| Clock Control | | | | | | | | | |
| Global | – | – | – | X | X | X | X | X | X |
| Modules | – | – | – | X | – | – | – | X | X |

(1) Flash devices only.
(2) <=/>= is Compare for ==, <>, <=, and >=
(3) Standard Comparision within all devices

## 10 References

1. MSP430 Product Brochure (SLAB034)

# Appendix A Examples

The appendix contains some samples which could be used for a simple try of the EEM features which have been discussed above. The code provided with this report does not present a good coding style or have any useful application specific part. It has been designed for an easy use of the EEM features. Esp. allowing of nested interrupts and a delay loop inside an Interrupt Service Routine should never been used in any application.

## A.1 Break on Write to Address

1. Open Breakpoint dialog from Edit menu
2. Select Conditional tab
3. Enter name of the variable for break in field: wSetBreakpointOn
4. Select Address bus, operator '==' and Write Access
5. Action should be Break
6. In the Condition Field set a Value of 0xA0A0, Operator '!=' and Write Access (Do not change the mask value)
   - System will stop if a value not equal to 0xA0A0 is written into the variable wSetBreakpointOn
   - To test, press button several times in fast sequence (>20)

## A.2 Break on Write to Register

1. Delete or disable previous Breakpoints
2. Open Breakpoint dialog from Edit menu
3. Select Conditional tab
4. Enter name of the variable for Break in field: SP (for Stack Pointer)
5. Select Register , operator '<=' and Write Access
6. Action should be Break
7. Set Register value to 0x09A0
8. No change is required in the Condition or Mask fields
   - System will halt if the stack pointer decreases to a value below 0x09A0
   - To test, press button several times in a fast sequence (>20)

## A.3 Break on Write to Flash

1. Delete or disable previous Breakpoints
2. Open Breakpoint dialog from Edit menu
3. Select Range tab
4. Enter Start Address: 0xE000 and End Address: 0xFFDE
5. For Write protection select:
   - Type: Address
   - Access: Write
   - Action: Break
   - Action when: Inside range

**Note:**

To test, Reset the device via the GUI and execute GO. After a certain time (approximately 20–30 sec) the break will occur.

## A.4 Break on Access of Invalid Memory

1. Delete or disable previous Breakpoints
2. Open Breakpoint dialog from Edit menu
3. Select Range tab
4. Enter Start Address: 0xC00 and End Address: 0xFFF
5. For Read protection select:
   – Type: Address
   – Access: Read/Write
   – Action: Break
   – Action when: Inside range

**Note:**

To test, Reset the device via the GUI and execute GO. After a certain time the break will occur.

## A.5 Break if Fetch is Out of Allowed Area

1. Delete or disable previous Breakpoints
2. Open Breakpoint dialog from Edit menu
3. Select Range tab
4. Enter Start Address: 0x1100 and Length: 0xEEFE
5. For Read protection select:
   – Type: Address
   – Access: Fetch
   – Action: Break
   – Action when: Outside range

**Note:**

To test, Reset the device via the GUI and execute GO. After a certain time NOTHING should will occur.

## A.6 State Storage: Trace

1. Delete or disable previous Breakpoints
2. Open State Storage Dialog via Emulator Menu to configure the state storage function
3. Switch on via Enable state storage
4. Configure the buffer to wrap around
5. Configure how the State Storage function should trigger to save events- Instruction Fetch
6. Run the code for some time then halt the execution. The last eight instructions are stored in the buffer
   – For a most configurations, the following settings could be used:
     – Check ‘Enable state storage’
     – Check ‘Buffer wrap around’
     – Storage action on ‘Instruction fetch’
   – Open State Storage Window via the Emulator Menu and examine the data after the CPU was running and is stopped.

## A.7 State Storage: Real-Time Watch

1. Delete or disable previous Breakpoints and Triggers
2. Setting up a Real-Time Watch with State Storage:
3. Setup a State Storage Trigger in the Breakpoint Menu

- Enter name of the variable for break in field: wCounter
- Select Address bus, operator '==' and Write Access
- Select as Action: State Storage Trigger
- Leave the Condition Field empty

4. Open the State Storage dialog via the Emulator Menu:
   - Switch on with Enable State Storage
   - Configure the buffer to wrap around
   - Configure the State Storage to execute on triggers

**Note:**

During the Free Run with the Update Button in the State Storage Window, the State Storage Buffer is read and displayed.

## A.8 Trigger Sequencer

1. Delete or disable previous Breakpoints and Triggers
2. Setup the following Trigger:
   - Beginning of main (e.g. Disable Watchdog)
   - Instruction after P1.1 is set
   - Instruction in P1 Interrupt Service Routine
3. Open Sequencer Dialog via Emulator Menu to configure the Sequencer
4. Configure Sequencer:
   - Switch on via Enable Sequencer
   - Transition Trigger 0 : Trigger on Beginning of Main
   - Transition Trigger 1 : Trigger on Instruction in P1 Interrupt Service Routine
   - Transition Trigger 2 : Instruction after P1.1 is set
   - Action : Break
5. Click Reset Button in GUI
6. Start execution
7. No action until S1 is pressed and P1.1 is set again
8. Open Sequencer and push the Reset State button
9. Continue Running without Reset
10. Device will not be stopped because Trigger 0 is never executed.

## A.9 Advanced Trigger Sequencer

1. Delete or disable previous Breakpoints and Triggers
2. Setup the following Trigger:
   - On wFlags |= 0x01 instruction of P1.3 (S1) ISR
   - On wFlags |= 0x02 instruction of P1.4 (S2) ISR
   - Last Instruction (RETI)in P1 Interrupt Service Routine in Disassembly window
3. Configure Sequencer:
   - Transition Trigger 0/a : Trigger on S1 / Next State : 1
   - Transition Trigger 1/a : Trigger on S2 / Next State : 3 (=Event)
   - Transition Trigger 1/b : Last Instruction in P1 ISR / Next State : 0
   - Action : Break
4. Click Reset Button in GUI
5. Start execution:
   - No action until S1, then S2 are pressed while executing the ISR delay loop
   - Try Sequences:
     - S2 then S1
     - S1 and wait until LED's start Flashing again (ISR is finished) then press S2

## IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

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

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

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI 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 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. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

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

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

| **Products** | | **Applications** | |
|---|---|---|---|
| Amplifiers | amplifier.ti.com | Audio | www.ti.com/audio |
| Data Converters | dataconverter.ti.com | Automotive | www.ti.com/automotive |
| DSP | dsp.ti.com | Broadband | www.ti.com/broadband |
| Interface | interface.ti.com | Digital Control | www.ti.com/digitalcontrol |
| Logic | logic.ti.com | Military | www.ti.com/military |
| Power Mgmt | power.ti.com | Optical Networking | www.ti.com/opticalnetwork |
| Microcontrollers | microcontroller.ti.com | Security | www.ti.com/security |
| | | Telephony | www.ti.com/telephony |
| | | Video & Imaging | www.ti.com/video |
| | | Wireless | www.ti.com/wireless |

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265

Copyright © 2005, Texas Instruments Incorporated