

# MSP430 Advanced Technical Conference 2006



## Hands-On: Experiencing Enhanced Emulation Debugging

Stefan Schauer  
Product Application Engineer  
Texas Instruments

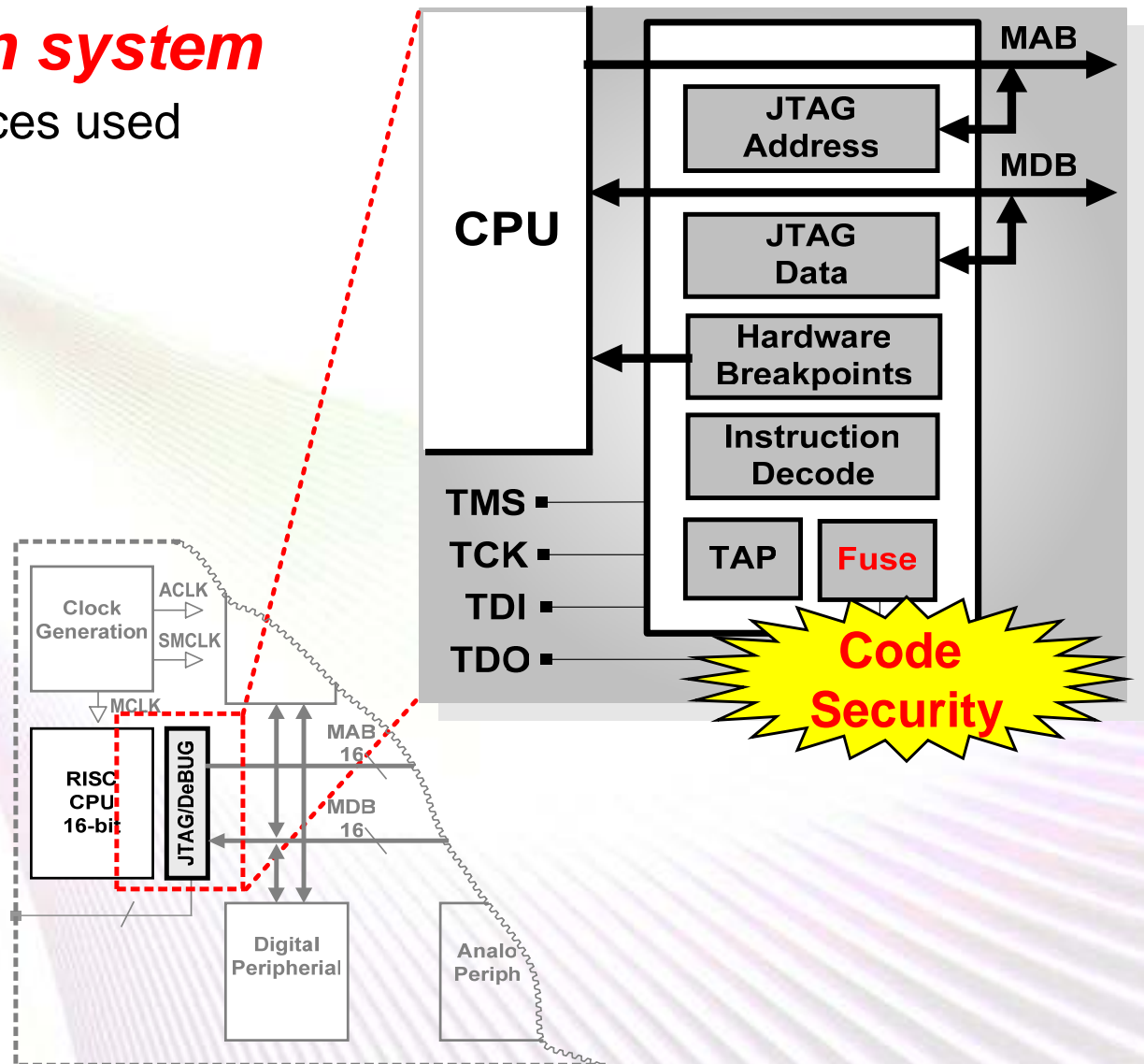
# Agenda

- Introduction to the Embedded Debug Logic (Enhanced Emulation Module: EEM)
  - **Show different implementation Levels of the EEM**
  - **EEM Limitations and Behaviors**
- Lab: Setting a Breakpoint on Stack Overflow
  - **Using On-Chip Trace Buffer to see where the problem did occur**
- Lab: Setting a Breakpoint on Fetch outside allowed Area
  - **Using On-Chip Trace Buffer to see where the problem did occur**
- Lab: Setting a Breakpoint on a Variable
  - **Stop on Write**
  - **Trace on Write**
  - **Stop on Write of a dedicated Value**
- Lab: Using the Trigger Sequencer
- Lab: Clock Control
- Lab: Using On-Chip Trace Buffer as Real-Time Watch
- Lab: Building own complex Breakpoints with combining of Triggers

# Embedded Emulation

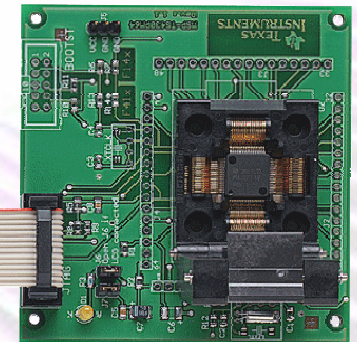
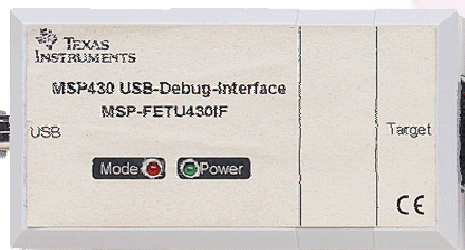
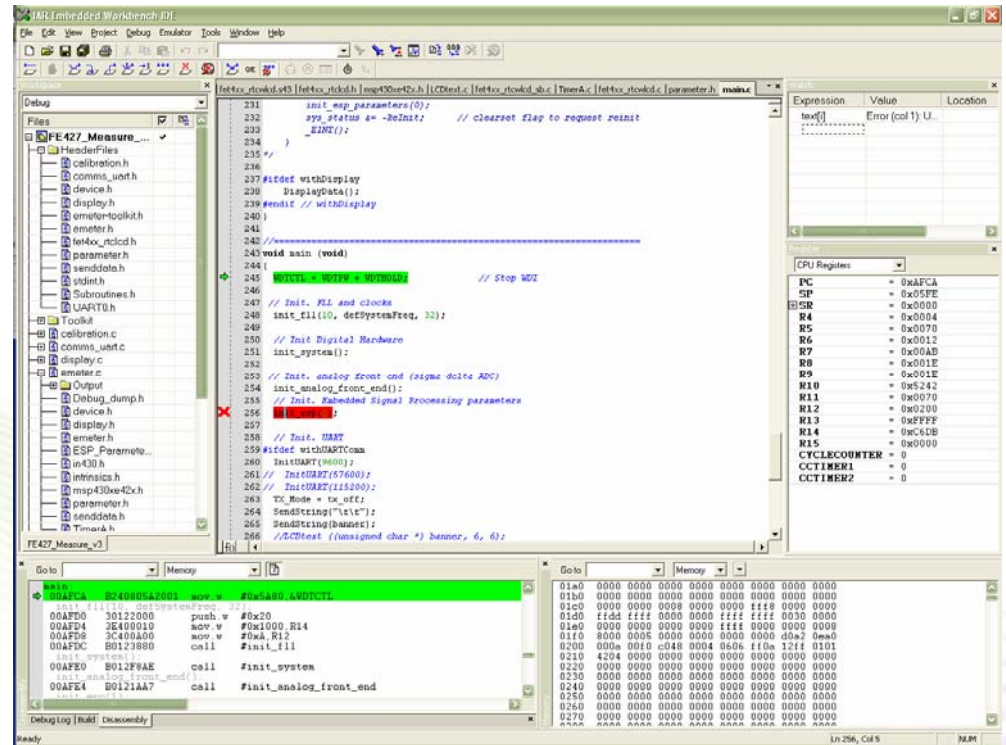
- **Debug real time in system**

- No application resources used
- Full speed
- Breakpoint
- Single step
- Complex trigger
- Trace



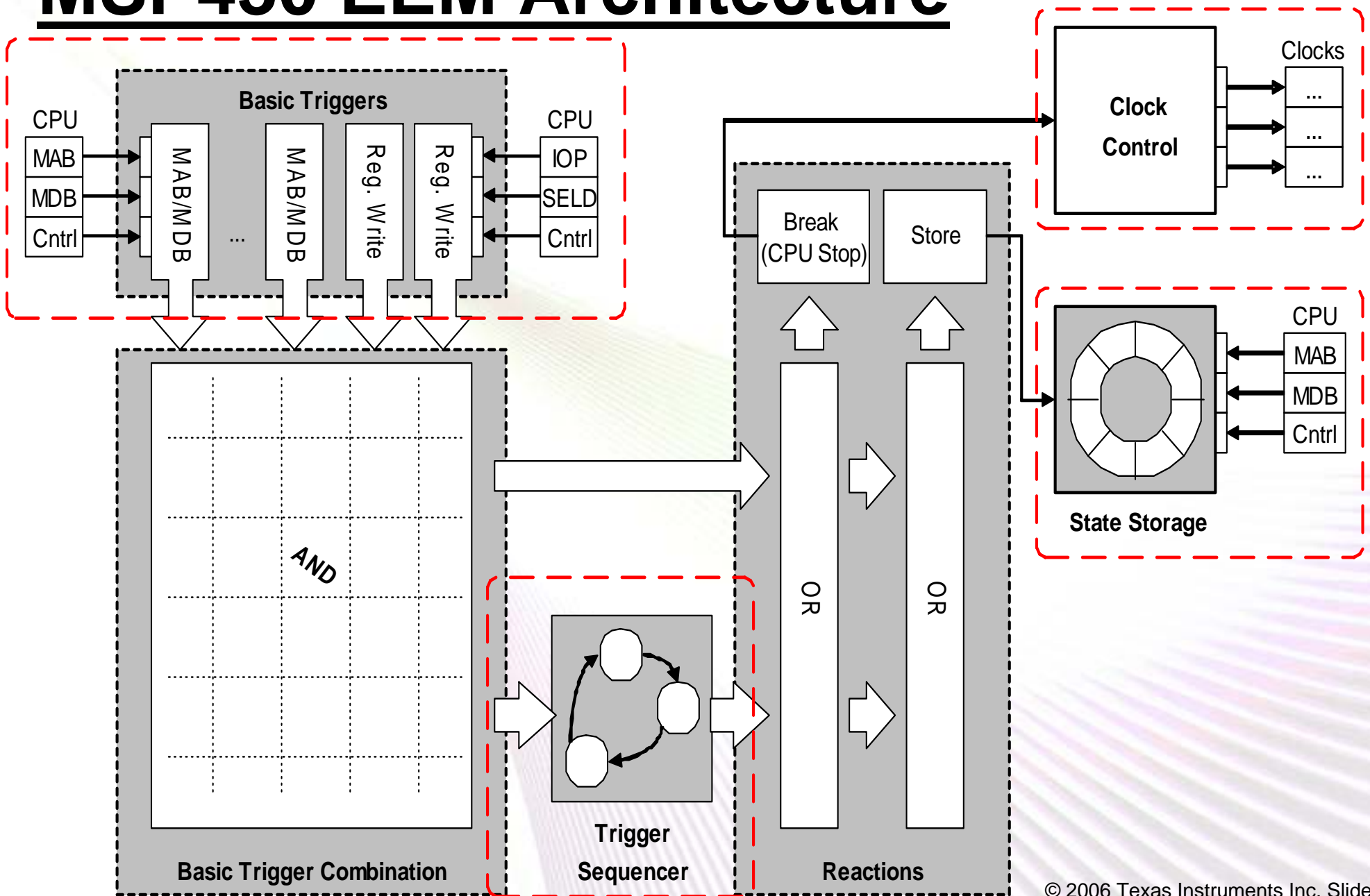
# FET – One Tool For Every Device

- Assembler/linker
- 4KB C compiler
- Common IDE
- JTAG interface
- Target Board
- \$149 USD



© 2006 Texas Instruments Inc, Slide 4

# MSP430 EEM Architecture



# Available EEM Resources

Device	F11x1 / F12x	F12x2	F13x / F14x	F15x / F16x	F20xx / F21x1 / F22xx / F23xx	F41x	FE42x / FW42x	FG43x	F43x / F44x / FG46x
<b>Triggers</b>									
<b>MAB/MDB-Trigger</b>	2	2	3	8	2	2	2	2	8
<=/>=	-	-	X	X	-	-	-	-	X
R/W	-	-	-	X	-	-	-	-	X
DMA	-	X	-	X	-	-	-	X	-
16bit Mask	-	-	-	X	-	-	-	-	X
<b>Reg.-Write-Trigger</b>	-	-	-	2	-	-	-	-	2
<=/>= 1	-	-	-	X	-	-	-	-	X
16bit Mask	-	-	-	X	-	-	-	-	X
<b>Combination</b>	2	2	3	8	2	2	2	2	8
<b>Trigger Sequencer</b>	-	-	-	1	-	-	-	-	1
<b>Reactions</b>									
Break	X	X	X	X	X	X	X	X	X
State Storage	-	-	-	X	-	-	-	-	X
<b>State Storage</b>									
Internal	-	-	-	X	-	-	-	-	X
<b>Clock Control</b>									
Global	-	-	-	X	X	X	X	X	X
Modules	-	-	-	X	-	-	-	X	X

Note: Flash devices only

© 2006 Texas Instruments Inc, Slide 6

# Influence and Resource Requirement

## **The EEM:**

- Does not use any internal CPU registers or memory
- Does not use interrupt vectors
- Does not insert debugging code or software breakpoints
- Has no influence on the program until a break event

## **Exception:**

- Devices  $\leq 28$  pin share the JTAG pins with port pins
- Spy Bi-Wire: use of RST/NMI pin

# Exceptions

- **Complex breakpoints stop the CPU after the instruction causing the break.**
- **When a break occurs, the execution of the current instruction will always be completed.**
- **EEM cannot prevent an invalid value from being written into an address or register.**
- **It is not possible to trigger on timer values. Only the values on the address or data bus can be observed.**



# Where to find the menus

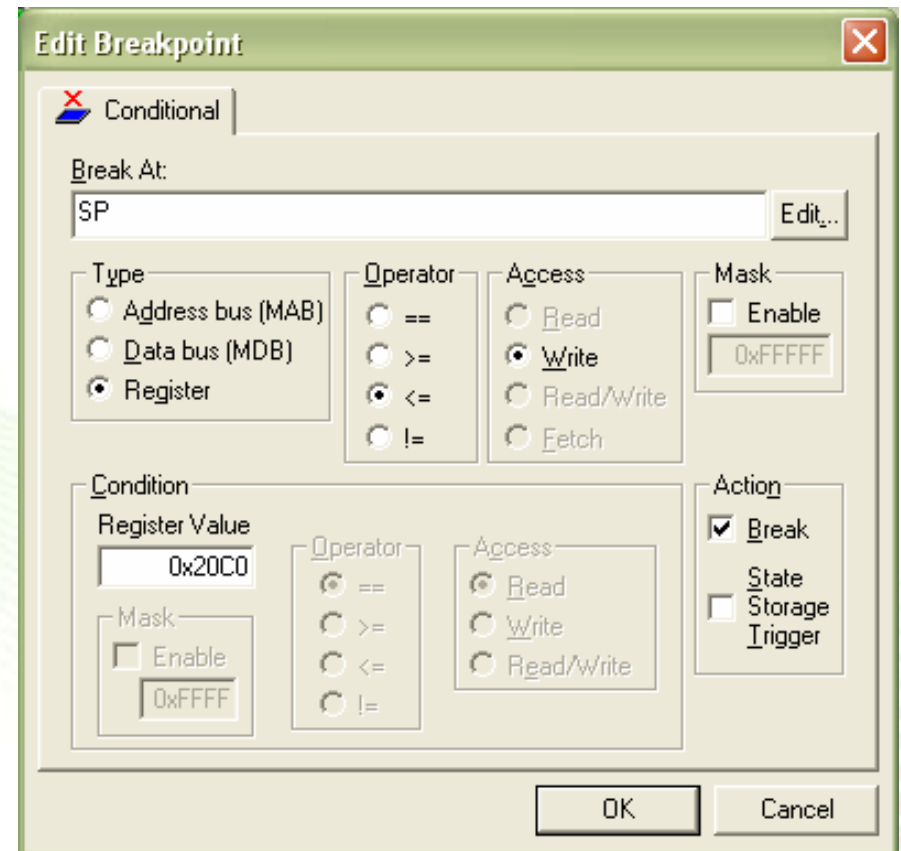
- **Breakpoint:**
  - View | Breakpoint
- **New Breakpoint:**
  - Right Click into the Breakpoint window and select New Breakpoint
- **State Storage Configuration**
  - Emulator | State Storage Control
- **State Storage Window**
  - Emulator | State Storage Window
- **Trigger Sequencer Control**
  - Emulator | Sequencer Control

# Agenda

- Introduction to the Embedded Debug Logic (Enhanced Emulation Module: EEM)
  - **Show different implementation Levels of the EEM**
  - **EEM Limitations and Behaviors**
- Lab: Setting a Breakpoint on Stack Overflow
  - **Using On-Chip Trace Buffer to see where the problem did occur**
- Lab: Setting a Breakpoint on Fetch outside allowed Area
  - **Using On-Chip Trace Buffer to see where the problem did occur**
- Lab: Setting a Breakpoint on a Variable
  - **Stop on Write**
  - **Trace on Write**
  - **Stop on Write of a dedicated Value**
- Lab: Using the Trigger Sequencer
- Lab: Clock Control
- Lab: Using On-Chip Trace Buffer as Real-Time Watch
- Lab: Building own complex Breakpoints with combining of Triggers

# Lab: Stack Observation

- **Nested functions or local declarations if arrays could easily lead to this problem**
- **Set a conditional breakpoint on the Stack Pointer so that the CPU stops if the SP decreases below 0x20C0.**



# Lab: Stack Observation

**Target:** Halt CPU if SP decrements below a certain level

**Demo Program:** Clock\_TB1.c

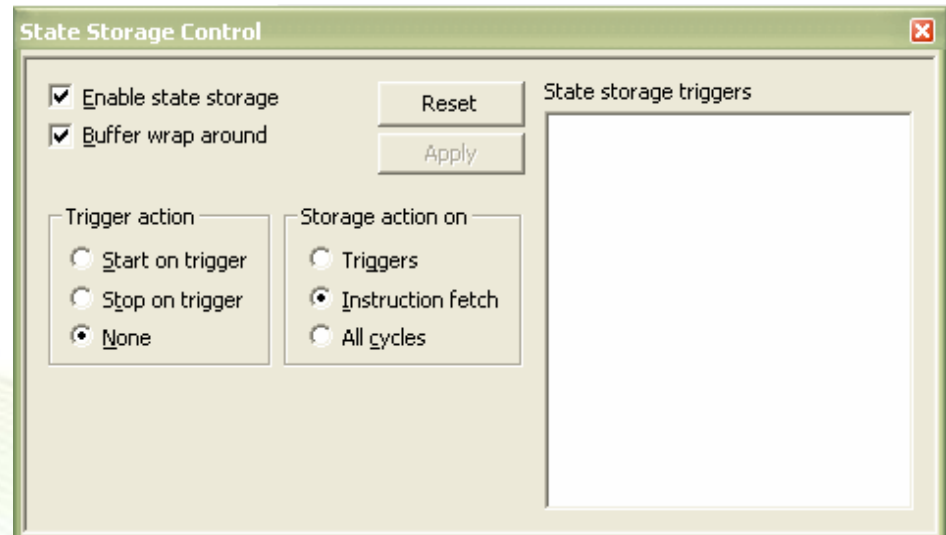
**Detailed Lab Instructions:**

- **Open breakpoint dialog: View | Breakpoints**
- **Clear all previous breakpoints**
- **Create new “Conditional” Breakpoint**
  - Break At: SP (for Stack Pointer – Note: ‘SP’ should be upper case !)
  - Type: Register
  - Operator: <=
  - Access: write
  - Mask: not enabled
  - Condition: 0x20C0
  - Action: Break
- **Close the dialog with OK**
- **Start program execution**
  - Program should stop in the function ‘foo’ after the 80 bytes have been allocated on the stack (This should take approximately 8 seconds.)

© 2006 Texas Instruments Inc, Slide 12

# Lab: Using Trace for Stack Observation

- **Open the State Storage Control**
  - Enable state Storage
  - Enable Buffer wrap around
  - Trigger action: None
  - Storage Action on: Instruction Fetch→ Apply
- **Open the State Storage Window**
- **Execute Program again:**
  - Push the reset Button and execute the program again
  - After the breakpoint was hit observe the output in the State Storage Window



The screenshot shows the 'State Storage Window' with a title bar and a close button. Below the title bar are four checkboxes: 'Update' (checked), 'Automatic update' (unchecked), 'Automatic restart' (unchecked), and 'Append data' (unchecked). Below these is a table with four columns: 'Address ...', 'Instr.', 'Mnemonic', and 'Data bus...'. The table contains several rows of instruction data, with the row at address 0x1000 highlighted in yellow.

Address ...	Instr.	Mnemonic	Data bus...
0x2192	B013010	calla #foo	0x13B0
0x1000	3180A00	sub.w #0xA0,SP	0x8031
0x1004	343	nop	0x4303
0x21C0	B290E83411	cmp.w #0x3E8,&u...	0x90B2
0x21C6	FA2B	jnc 0x21BC	0x2BFA
0x21C8	101	reta	0x0110
0x218A	B29002211	cmp.w #0x200,&u...	0x90B2
0x2190	F823	jne 0x2182	0x23F8

© 2006 Texas Instruments Inc, Slide 13

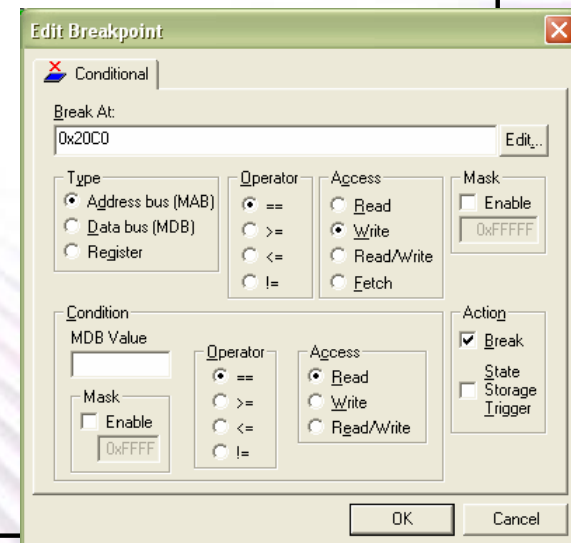
# Lab: Stack Observation (MSP430X)

**Due to the speed improvements in the MSP430X CPU an additional Breakpoint is required for this CPU to get all Stack overflows**

**Demo Program: Clock\_TB1.c**

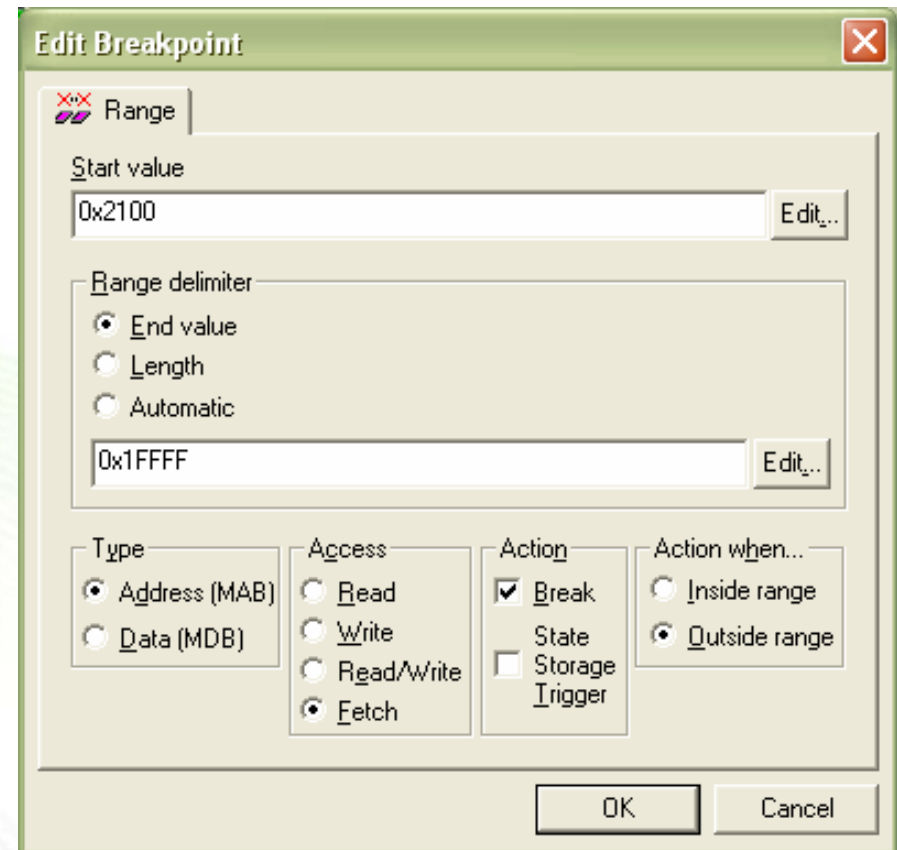
**Detailed Lab Instructions: (add this to the previous Lab)**

- **Open breakpoint dialog: View | Breakpoints**
- **Modify previous breakpoint to 0x20FA**
- **Start program execution**
  - Program should stop in 'delay' function when the return address is saved on the stack but this does not work.
  - Note: Program will also stop (3 times) during the initialization part (CStartup)
- **Create new "Conditional" Breakpoint**
  - **Break At:** 0x20FA
  - **Type:** MAB
  - **Operator:** ==
  - **Access:** write
  - **Mask:** not enabled
  - **Action:** Break
- **Close the dialog with OK**
- **Start program execution**
  - Program should stop now also in the function 'delay'



# Program Fetch Observation

- **A problem with function pointers to improve and optimize code or function tables could make the PC jump somewhere. Finding this problems is very hard because the origin of the problem could not be detected.**
- **Set a range breakpoint:  
Start: 0x2100  
End: 0x1FFFF  
Access on Fetch if outside range**



# Lab: Program Fetch Observation

**Target:** Halt CPU when loading an instruction in invalid range

**Demo Program:** Clock\_TB1.c

## **Detailed Lab Instructions:**

- **Open breakpoint dialog: View | Breakpoints**
- **Clear all previous breakpoints**
- **Create new “Range” Breakpoint**
  - **Start Value:** 0x2100
  - **Range delimiter:** End Value -> 0x1FFFF
  - **Type:** Address (MAB)
  - **Access:** Fetch
  - **Action:** Break
  - **Action when:** Outside range
- **Close the dialog with OK**
- **Reset and Start program execution**
  - Program should stop when the function ‘foo’ is called (because ‘foo’ is placed into info memory at 0x1000) (This should take approximately 8 seconds.)



# Lab: Using the Trace for Fetch Observation

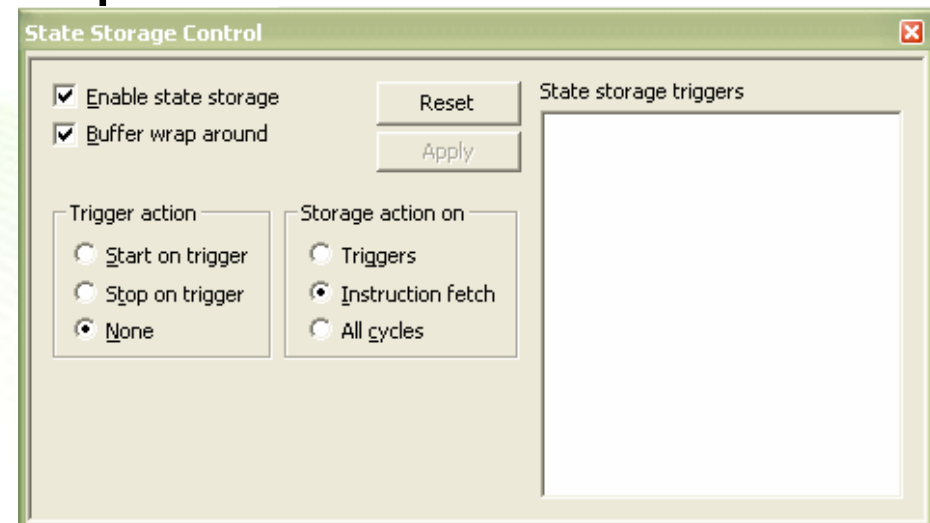
- **Open the State Storage Control**

- Enable state Storage
  - Enable Buffer wrap around
  - Trigger action: None
  - Storage Action on: Instruction Fetch
- Apply

- **Open the State Storage Window**

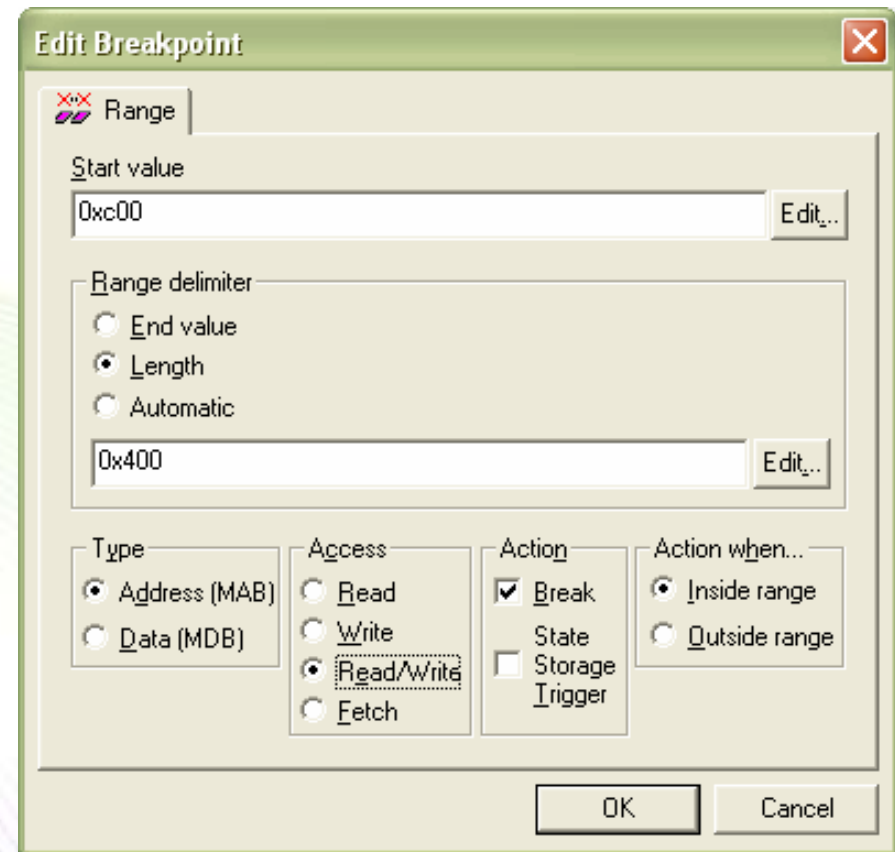
- **Execute Program again:**

- Push the reset Button and execute the program again
- After the breakpoint was hit observe the output in the State Storage Window



# Break on Read/Write to Invalid Memory

- **Example:**  
The CPU should stop if a read access from a specified memory area occurs (0xC00 to 0xFFF in this case).



# Lab: Break on Read/Write to Invalid Memory

**Target:** Halt CPU when accessing invalid memory

**Demo Program:** Clock\_TB1.c

## **Detailed Lab Instructions:**

- **Open breakpoint dialog: View | Breakpoints**
- **Clear all previous breakpoints**
- **Create new “Range” Breakpoint**
  - **Start Value:** 0xc00
  - **Range delimiter:** End Value -> 0xFFF
  - **Type:** Address (MAB)
  - **Access:** Read/Write
  - **Action:** Break
  - **Action when:** Inside range
- **Close the dialog with OK**
- **Reset and Start program execution**
  - Program should stop when the line :” \*(ptr + 2) = \*ptr + 0x1234;” is executed as this does access the Boot loader Memory.  
Additional step: Try to modify the trigger to Read or Write only. Check the difference within the disassembler window.

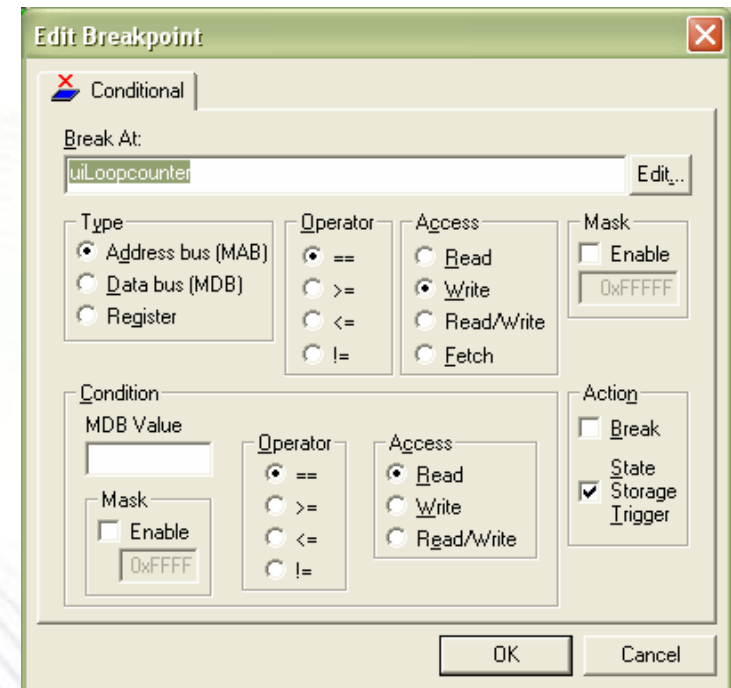
# Lab: Trace on Write to Memory

**Target:** Trace the information which is written into a dedicated memory address during program execution

**Demo Program:** Clock\_TB1.c

## Detailed Lab Instructions:

- Open breakpoint dialog: View | Breakpoints
- Clear all previous breakpoints
- Create New “Conditional” Breakpoint:
  - Break At: uiLoopcounter
  - Type: MAB
  - Operator: ==
  - Access: write
  - Mask: not enabled
  - Action: State Storage Trigger
- Close the dialog with OK
- Setup State Storage:
  - Enable state Storage
  - Enable Buffer wrap around
  - Trigger action: None (disabled)
  - Storage Action on: Triggers
- Apply
- Reset and Start program execution
  - Start and Stop Program execution or set a breakpoint on the call of the foo function. You should see the last view increments of the uiLoopcounter variable in the State storage window.



© 2006 Texas Instruments Inc, Slide 20

# Lab: Real-Time Watch

**Target:** Trace the information which is written into a dedicated memory address during program execution and read the data without stopping the CPU

**Demo Program:** Clock\_TB1.c

**Detailed Lab Instructions:** (add this to the previous)

- Start program execution
- Open the State Storage Window and press the update button
  - A snapshot of the last trace entries is read and displayed in the State Storage Window

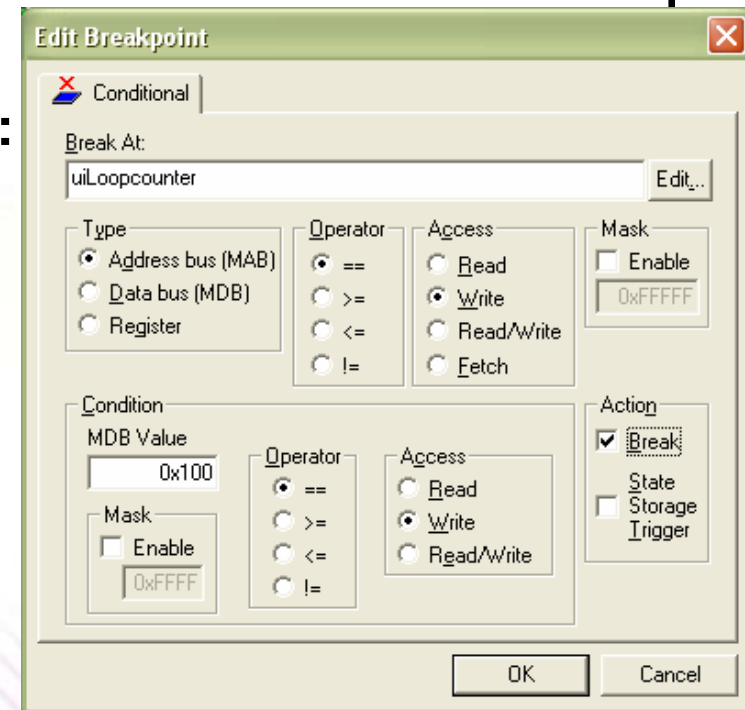
# Lab: Stop on Memory Access with dedicated Value

**Target:** Trace the information which is written into a dedicated memory address during program execution

**Demo Program:** Clock\_TB1.c

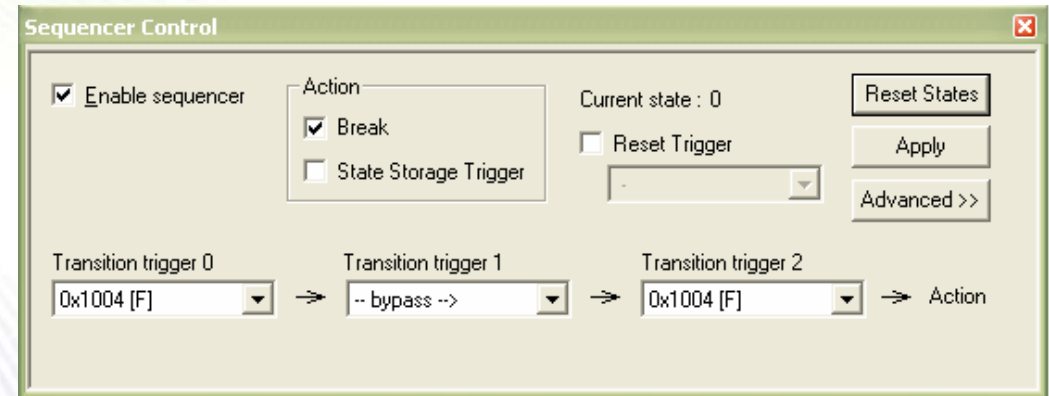
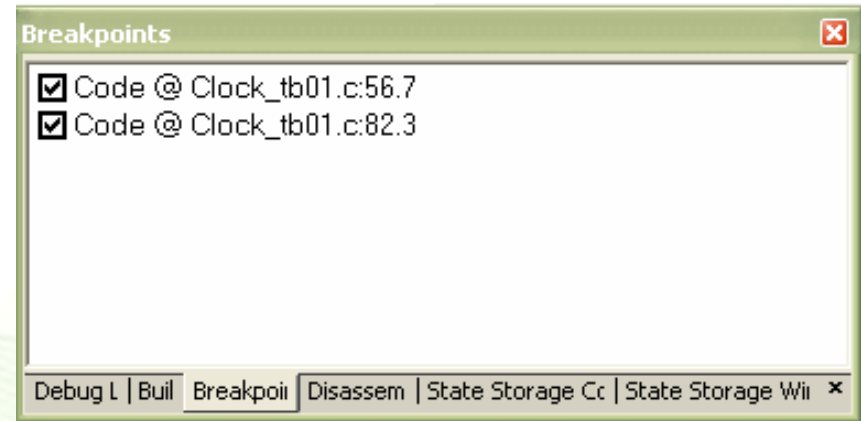
**Detailed Lab Instructions:**

- Open breakpoint dialog: View | Breakpoints
- Modify the previous “Conditional” Breakpoint:
  - Break At: uiLoopcounter
  - Type: MAB
  - Operator: ==
  - Access: write
  - Mask: not enabled
  - Action: Break
  - Condition MDB value: 0x100
  - Condition Operator: ==
  - Condition Access: write
- Close the dialog with OK
- Add the uiLoopCounter to the Watch Window
- Start program execution
  - Check the content of uiLoopcounter after program execution did stop. It should contain 0x100.



# Trigger Sequencer

- Can create a linear program sequence before a trigger is accepted for a break or state storage event
- Useful if an event occurs only after a given sequence in the program has taken place



# Lab: Trigger Sequencer

**Target:** Halt CPU if a certain program sequence was executed

**Demo Program:** Clock\_TB1.c

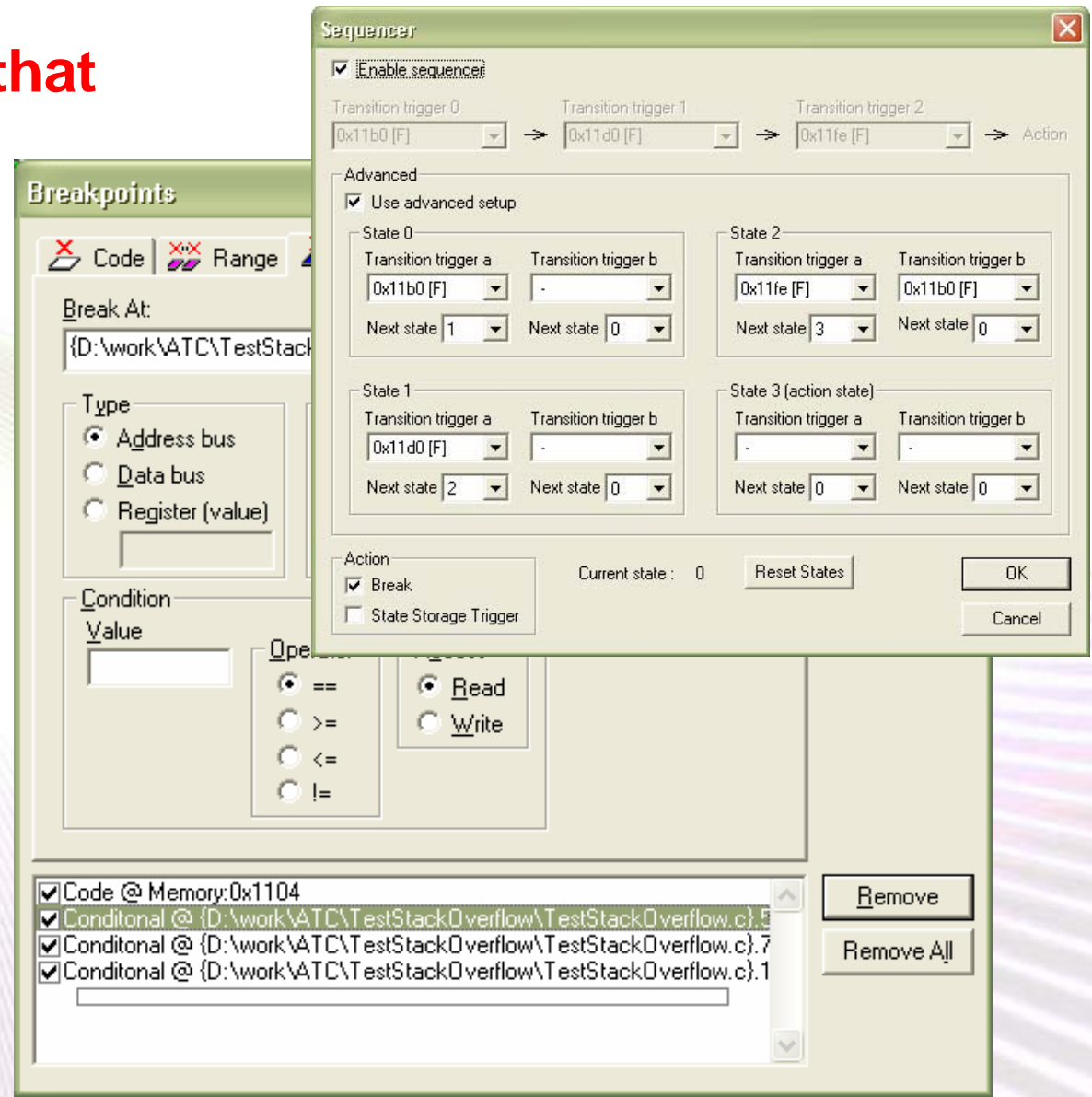
## **Detailed Lab Instructions:**

- Open breakpoint dialog: View | Breakpoints
- Clear all previous breakpoints
- Set a code breakpoint in line 56: “uiLoopcounter++;”
- Set a code breakpoint in line 82: “\_NOP();”
- Open the Trigger “Sequencer Control” Window
  - Enable Sequencer
  - Transition Trigger 0: 0x1004 [F]
  - Transition Trigger 1: Bypass
  - Transition Trigger 2: 0x2182 [F]
  - Action: Break
    - Reset States
    - Apply
- **Reset and Start program execution**
  - Program should stop after “uiLoopcounter++;” but the variable is already incremented to 513. So the Breakpoint is activated after the function foo was executed. To repeat the test goto the Trigger Sequencer window and push the “Reset States” Button



# Complex Trigger Sequencer

- **No Lab – just to show that this is also available**
- **Allows a trigger on complex system sequences**
- **Restart and reset conditions for the sequencer can also be defined**

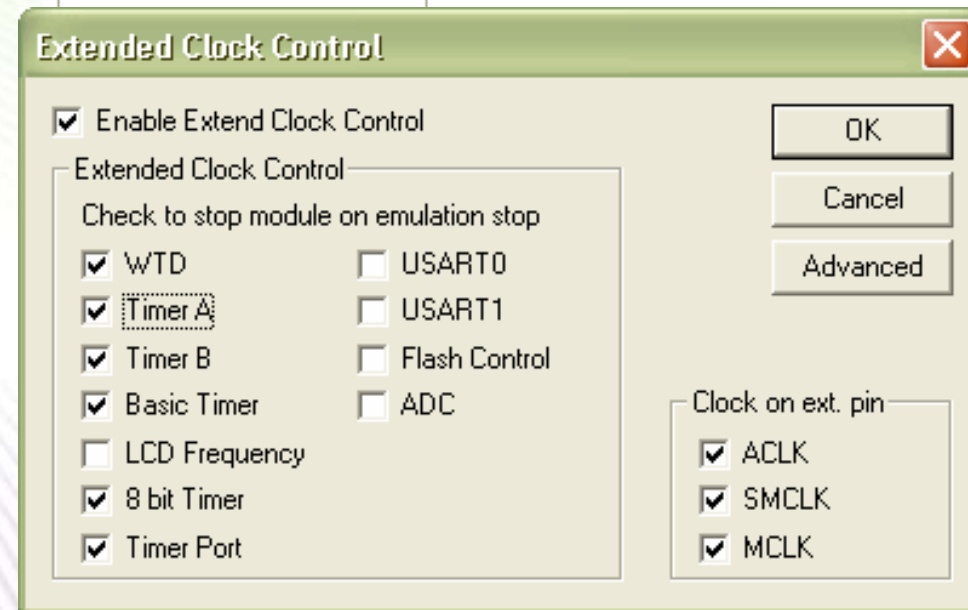
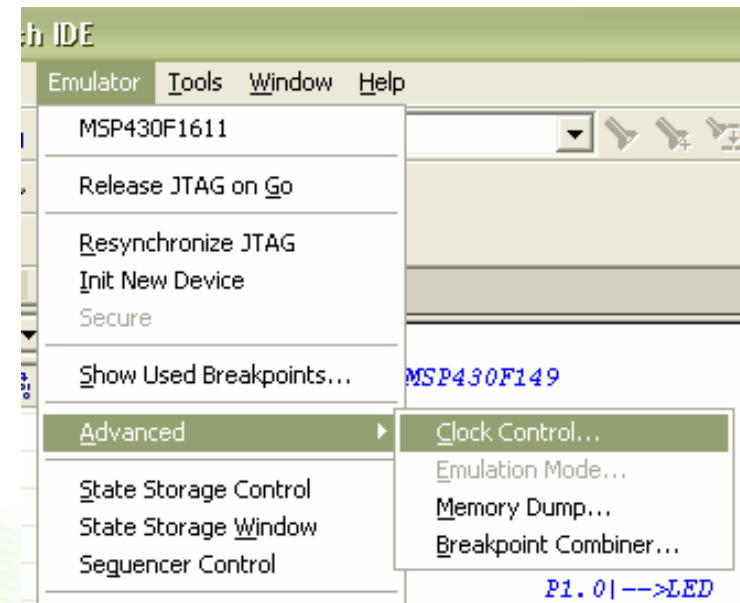


# Clock Control

- **Different applications have different requirements for the clock control during debug**
- **For instance, it might be dangerous to stop a clock for a timer which is generating a PWM signal for a motor.**
  - Similar requirements could exist for the Flash, UART, ADC, etc.
- **Clock control may be needed when the clock is triggering a counter which continuously requests interrupts during the stop time, for example an RTC**

# Clock Control

- Stop & release Clock for TimerB
- Check PWM output (P2.2)
- Check debugging. If ISR is active set a breakpoint in the ISR
- Test single stepping with Clock for TimerB stopped and released



# Lab: Clock Control

**Target:** Check device operation w/ different clock control setup

**Demo Program:** Clock\_TB1.c

**Detailed Lab Instructions:**

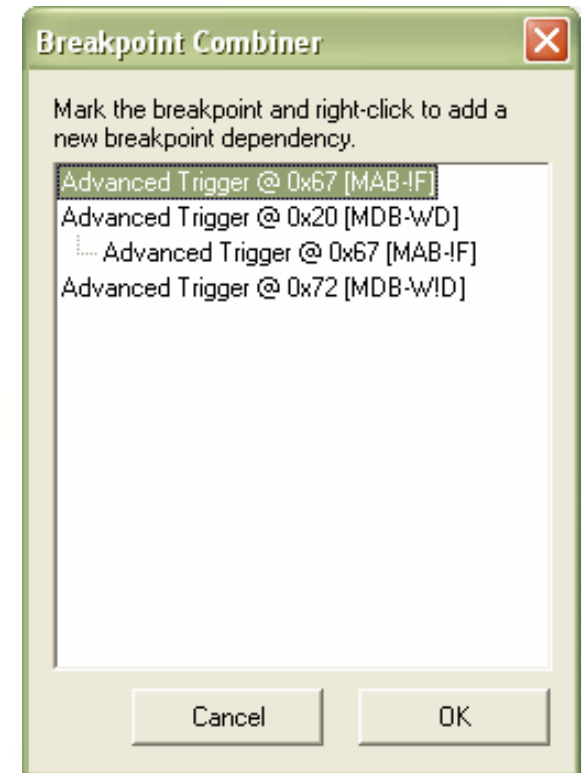
- **Open breakpoint dialog: View | Breakpoints**
- **Clear all previous breakpoints**
- **Close the dialog with OK**
- **Start Debugger**
- **Open Emulator | Advanced | Clock Control Dialog**
  - Click on the Advanced Button
  - Enable Extended Clock Control
  - Check TimerB → so that Clock for TimerB is stopped on Emulation hold
- **Close the dialog with OK**
- **Accept reset of the CPU**
- **Start program execution**
  - Check PWM output (P2.2 - LED1)
  - Check software toggled output (P2.1 - LED2)

# Lab: Clock Control

- **Stop program execution**
  - Check PWM output (P2.2 - LED1 )
  - Check software toggled output (P2.1 - LED2 )
- **Try to single step through the program (esp. main program)**
- **Open Emulator | Advanced | Clock Control Dialog**
- **Enable Extended Clock Control**
- **Uncheck TimerB → so that clock for TimerB is not stopped on Emulation hold**
- **Close the dialog with OK**
- **Accept reset of the CPU**
- **Start Program execution**
  - Check PWM output (P2.2 - LED1 )
  - Check software toggled output (P2.1 - LED2 )
- **Stop Program execution**
  - Check PWM output (P2.2 - LED1 )
  - Check software toggled output (P2.1 - LED2 )
- **Try to single step through the program (esp. main program)**

# Combining Breakpoints

- **The Breakpoint Combiner dialog (Emulator | Advanced) allows the combination of two or more individual breakpoints or triggers**
- **The Sub-Trigger is added to the Main-Trigger with an AND combination**
- **The Sub-Trigger stays unmodified in the system**
  - A break action set on the Sub-Trigger stops execution independent from the Main-Trigger
  - Normally the Break Action should not be set for the Sub-Trigger



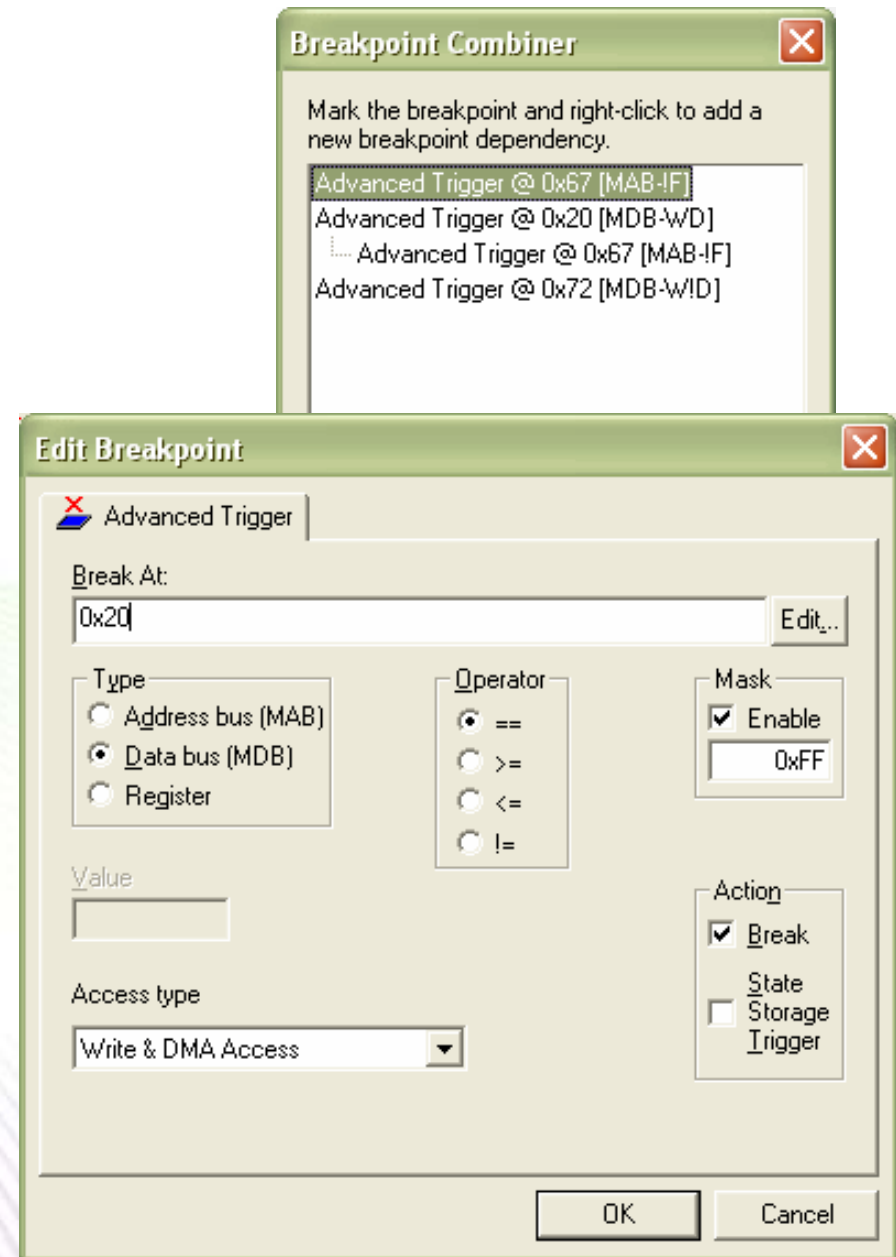
# DMA Trigger

- **During Program execution a single memory location could be accessed by the CPU and/or the DMA**
- **Allowing Trigger to detect between these two different types of accesses provides better control over software execution and maintaining real-time behavior of the system as much as possible without stopping the CPU**

# DMA Trigger

**Setting a break on a DMA transfer means that the CPU will stop only if a certain value is written into a dedicated address by the DMA**

- **Use Breakpoint Combiner to combine MAB & MDB Triggers**
  - Only the Main Trigger should have the Break Action set!
- **The CPU should stop if a DMA transfer of the Space Character into the UART TX Buffer is done**





# Lab: DMA Trigger

**Target:** Halt CPU the 0x20 moves to UCA0TXBUF via the DMA

**Demo Program:** ATC2006\_DMA\_Demo.c

## **Detailed Lab Instructions:**

- **Open breakpoint dialog: View | Breakpoints**
  - **Clear all previous breakpoints**
  - **Create new “Advanced Trigger”, set first trigger:**
    - **Break At:** UCA0TXBUF (0x6F)
    - **Type:** MAB
    - **Operator:** ==
    - **Mask:** not enabled
    - **Access Type:** No Instruction Fetch
    - **Action:** No Break
- OK

# Lab: DMA Trigger

- **Set second trigger:**
  - **Break At:** 0x20 (“space” character)
  - **Type:** MDB
  - **Operator:** ==
  - **Mask: Enable:** 0x00FF (only Byte access)
  - **Access Type:** Write & DMA Access
  - **Action:** Break→ OK
- **Close the dialog with OK**
- **Open “Breakpoint Combiner” dialog:**  
Emulator | Advanced | Breakpoint Combiner
- **Right click on ‘Advanced Trigger @ 0x20 [MDB-WD]’**
  - Add trigger ‘Advanced Trigger @ 0x67 [MAB-!f]’
- **Close the dialog with OK**
- **Start program execution**
- **Program should stop each time the DMA transfers the ‘space’ character to the UART TX buffer but. Note: It does not stop on the first transmitted character which is sent directly by the CPU.**

# Lab: DMA Trigger

- **Open breakpoint dialog: View | Breakpoints**
- **Create new “Advanced Trigger”**
  - **Break At:** 0x72 ('r' character)
  - **Type:** MDB
  - **Operator:** ==
  - **Mask:** Enable: 0x00FF (only Byte access)
  - **Access Type:** Write & No DMA Access
  - **Action:** Break
- Apply
- **Close the dialog with OK**
- **Open terminal program: 9600 / 8N1**
- **Open “Breakpoint Combiner” dialog :**
  - Emulator | Advanced | Breakpoint Combiner
- **Right click on ‘Advanced Trigger @ 0x72 [MDB-W!D]’**
  - Add trigger ‘Advanced Trigger @ 0x67 [MAB-!f]’
- **Close the dialog with OK, reset program, start program execution**
- **Program should stop when software transmits ‘r’ but not ‘space’ to the UART TX buffer. Also stops each time the DMA transfer ‘space’.**

# Summary

- The EEM logic allows powerful trigger and break settings making hard to find errors easily identifiable
- No additional hardware testing is necessary after development and evaluation with EEM
- Cost effective and efficient method of debugging
- Compatible across all products
- Facilitates true analog performance and behavior
- In-system and in-field debugging possible
- Observation of variables in a running system enables a deeper view into the application
- Given the flexibility of the EEM, implementation of additional features are possible and will be added in the near future including statistical code coverage and better implementation of real-time watches

## 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.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

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

<b>Products</b>		<b>Applications</b>	
Amplifiers	<a href="http://amplifier.ti.com">amplifier.ti.com</a>	Audio	<a href="http://www.ti.com/audio">www.ti.com/audio</a>
Data Converters	<a href="http://dataconverter.ti.com">dataconverter.ti.com</a>	Automotive	<a href="http://www.ti.com/automotive">www.ti.com/automotive</a>
DSP	<a href="http://dsp.ti.com">dsp.ti.com</a>	Broadband	<a href="http://www.ti.com/broadband">www.ti.com/broadband</a>
Interface	<a href="http://interface.ti.com">interface.ti.com</a>	Digital Control	<a href="http://www.ti.com/digitalcontrol">www.ti.com/digitalcontrol</a>
Logic	<a href="http://logic.ti.com">logic.ti.com</a>	Military	<a href="http://www.ti.com/military">www.ti.com/military</a>
Power Mgmt	<a href="http://power.ti.com">power.ti.com</a>	Optical Networking	<a href="http://www.ti.com/opticalnetwork">www.ti.com/opticalnetwork</a>
Microcontrollers	<a href="http://microcontroller.ti.com">microcontroller.ti.com</a>	Security	<a href="http://www.ti.com/security">www.ti.com/security</a>
RFID	<a href="http://www.ti-rfid.com">www.ti-rfid.com</a>	Telephony	<a href="http://www.ti.com/telephony">www.ti.com/telephony</a>
Low Power Wireless	<a href="http://www.ti.com/lpw">www.ti.com/lpw</a>	Video & Imaging	<a href="http://www.ti.com/video">www.ti.com/video</a>
		Wireless	<a href="http://www.ti.com/wireless">www.ti.com/wireless</a>

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265  
Copyright © 2007, Texas Instruments Incorporated