Freescale Semiconductor
Application Note

AN2596/D
Rev. 2, 11/2004

Using the HCS08 Family On-Chip In-Circuit Emulator (ICE)

By Eduardo Montañez
Applications Engineering
Austin, Texas

Introduction

This application note describes the on-chip ICE system available in the HCS08 Family of microcontrollers. This system, also known as the debug module (DBG), is a unique silicon solution that:

- Improves existing software debugging techniques
- Implements an on-chip trace buffer for real-time bus capture
- Defines flexible triggers and capture options
- Provides nonintrusive debugging of application software
- Integrates emulator and bus-state analyzer functionality on silicon

This application note will show the user how to navigate through the Metrowerks® CodeWarrior® on-chip ICE interface and how to debug software using the on-chip ICE. Several software debugging scenarios will be scripted and resolved using the analysis capability provided by the appropriate on-chip ICE trigger settings. Throughout, on-chip ICE basics will be covered giving the user a general overview of the many on-chip ICE features. For a detailed explanation of the on-chip ICE system, see the HCS08 Family Reference Manual (Freescale Semiconductor document *HCS08RMv1/D*) or the appropriate MCU data sheet.

NOTE

The on-chip ICE is not the same hardware as the background debug controller (BDC). The on-chip ICE is an independent module whose control registers and FIFO buffer can be accessed through the BDC.

Overview

The on-chip ICE system captures real-time bus information into an on-chip FIFO buffer based on a user-determined trigger point. Then, the captured information can be displayed or used to rebuild the software execution path through a user-friendly development tool (such as Metrowerks' CodeWarrior True-Time Simulator & Real-Time Debugger).

Enhanced Debugging Methods

Traditional debugging methods (such as breakpoints or software profiling) use software instructions to toggle general-purpose I/O or store data in RAM and halt software execution or introduce latency. Unlike these traditional methods, the on-chip ICE system is transparent to code execution. Therefore, the on-chip ICE can capture bus information in real time without stopping CPU execution. This enables the user to retrieve this information later for analysis.

On-Chip Trace Buffer

The on-chip trace buffer is used to record real-time bus information. The content captured is dependent on the on-chip ICE trigger settings. The FIFO buffer can either store change-of-flow addresses during software execution, or it can log data for events that occur at a memory location. This unique method of capturing only the relevant software changes-of-flow prevents irrelevant information from taking up space in the FIFO buffer. The development tool could then use the captured software change-of-flow addresses to decode the complete software execution path.

Flexible Triggers and Capture Options

Flexible triggers and capture options allow the user to specify when and what bus information to capture. These predetermined trigger conditions are used to qualify when a trigger will occur. The trigger condition also specifies what type of bus information will be captured. Trigger conditions are comprised of hardware comparators that are given a value (address or data) by the user. After the hardware comparators are matched during software execution, the user-determined trigger condition will be satisfied, and a trigger action will occur.

The trigger action is user-determined by the capture options. One capture option allows the trigger to continuously record bus information up to the point where the trigger condition is qualified (end trigger). The inverse of this option is to start recording bus information when the trigger condition is qualified (begin trigger). Another capture option determines whether to force the CPU execution to halt or not halt when the trigger capture has completed. For an end trigger, the halt point occurs when the trigger point is matched. For a begin trigger, the halt point occurs when the FIFO buffer becomes full.

Traditional emulators have large memory resources and allow you to perform a large code capture. After you capture a large section of code, you must then sort through thousands of lines of program code. This process can be inefficient and cost valuable engineering time.

Using on-chip ICE is a more efficient approach to debugging. Instead of capturing one large section of code, the user can capture several smaller sections. This method requires less time sorting through excess code. Using the flexible triggers and capture options available with the on-chip ICE allows you to achieve more efficient captures and enjoy the benefits of on-chip debugging.

Eliminates Traditional Emulator and Bus-State Analysis Tools

By integrating bus-trace capabilities in silicon, the on-chip ICE eliminates the need for expensive traditional emulator and bus-state analysis tools. This nonintrusive system has no limitations on operating voltage or frequency — unlike traditional emulators. Also, the system does not require any MCU pins, which eliminates the need for elaborate cables. In the HCS08 Family of microcontrollers, the on-chip ICE system is essential for debugging because there is no expanded data and address bus. In addition, the system gives the user access to the internal core, which is not accessible through a traditional external bus-state analyzer.

System Details

The following HCS08 Family on-chip ICE system details will be discussed in later sections:

- Total of three (breakpoint/trigger) hardware comparators:
  - Two 16-bit address comparators OR One 16-bit address plus 8-bit data comparator available in the on-chip ICE module (can be used as trace-capture triggers or typical breakpoints)
  - Third comparator can be used as an additional 16-bit address comparator available through the BDC (used only as a typical breakpoint)
- 8-stage (16-bit wide) FIFO buffer captures software change-of-flow addresses or event data
- Nine trigger modes control when a trigger condition is satisfied and what data is captured in the FIFO
  - A only
  - A or B
  - A then B
  - A and B data
  - A and NOT B data
  - Event Only B
  - A then Event Only B
  - Inside range (A<= address <=B)
  - Outside Range (address <=A or address >=B)
- Begin or end trigger capture options
  - Begin mode: Trigger starts filling buffer at beginning until full
  - End mode: Trigger stops filling circular buffer
- Tag type (opcode) or force triggers and breakpoints
  - Tag: Takes trigger/breakpoint only when tagged opcode is about to execute
  - Force: Takes trigger/breakpoint at the next instruction boundary after an address appears on bus
- BDM or SWI post-trigger action options
  - MCU enters BDM if a BDM interface is connected and ENBDM = 1
  - MCU executes a software interrupt (SWI) if in user mode (no BDM connected) ENBDM = 0
- Halt or not halt CPU execution following trigger option

CodeWarrior Tools (On-Chip ICE Support Interface)

Metrowerks CodeWarrior on-chip ICE system support was used to debug the various software scenarios described in the following sections. To follow these examples, CodeWarrior Development Studio for HC(S)08 Microcontrollers (V3.0 and greater) must be installed. Earlier CodeWarrior editions do not support the HCS08 Family or include on-chip ICE support.

Trace Components

To perform an on-chip ICE trigger capture, the user must first open the trace component window. This window will be used by the debugger to display event data or rebuilt software execution data stored in the FIFO buffer. This information will be displayed when the trigger sequence is complete and the target CPU returns to background mode or the serial monitor. The completion of a trigger sequence and data display is dependent on the chosen capture options.

If the user halts CPU execution manually prior to completing a trigger sequence, the trace component window will display the data captured in the FIFO buffer up to that halt point. For more information on the capture options and how they affect the trace buffer rebuild, refer to Trigger Conditions and Capture Options.

To open the trace component window, choose the BDM_HCS08 option at the top of the CodeWarrior debugger window. Then select the bus trace option from the menu list. An empty trace component window will open. The BDM_HCS08 option will be labeled differently if you are using another form of communication (serial monitor, inDart08, etc.). This application note will use a P&E MultiLink™ BDM interface for HCS08 and HCS12 microcontrollers.

Figure 1 illustrates opening the trace component window.

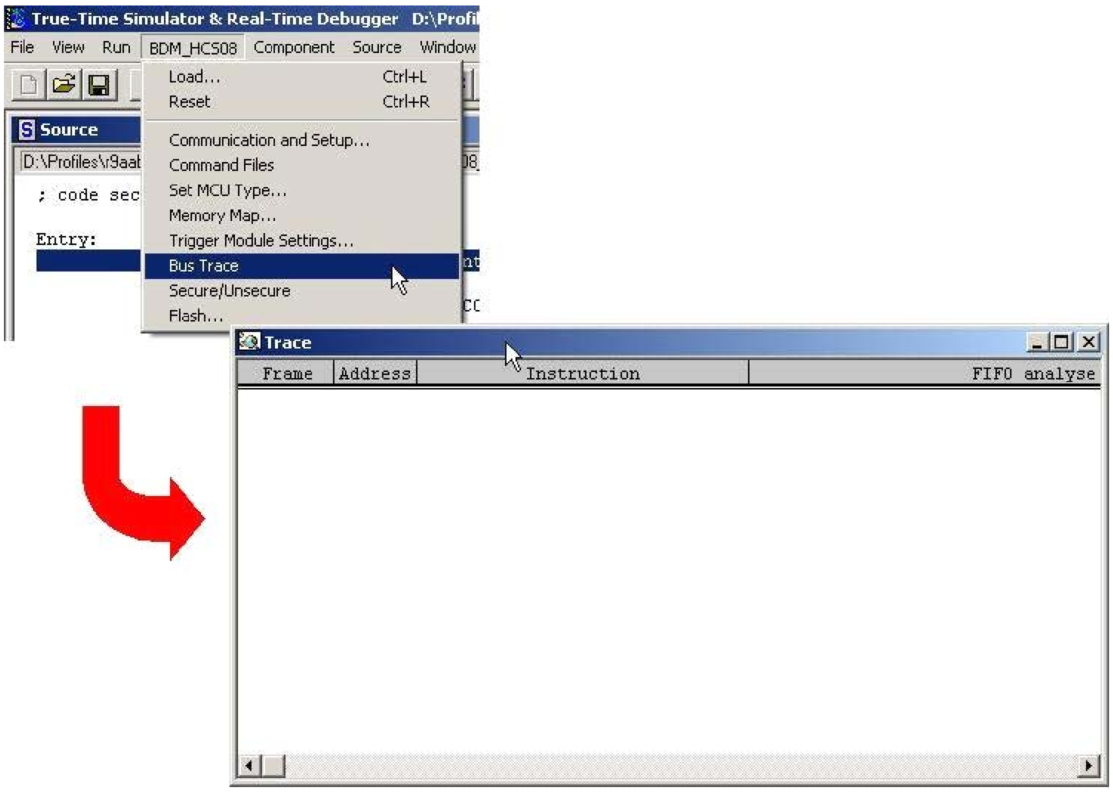

True-Time Simulator & Real-Time Debugger D:\Profi
File View Run BDM_HCS08 Component Source Window
Load... Ctrl+L
Reset Ctrl+R
Communication and Setup...
Command Files
Set MCU Type...
Memory Map...
Trigger Module Settings...
Bus Trace
Secure/Unsecure
Flash...
Source
D:\Profiles\r9aat
; code sec
Entry:
Trace
Frame Address Instruction FIFO analyse

Figure 1. Open Trace Component Window

NOTE

The trace component window must be opened prior to running a trigger capture on your application software. If opened after running your application, the trace window will not display event data or rebuilt program flow.

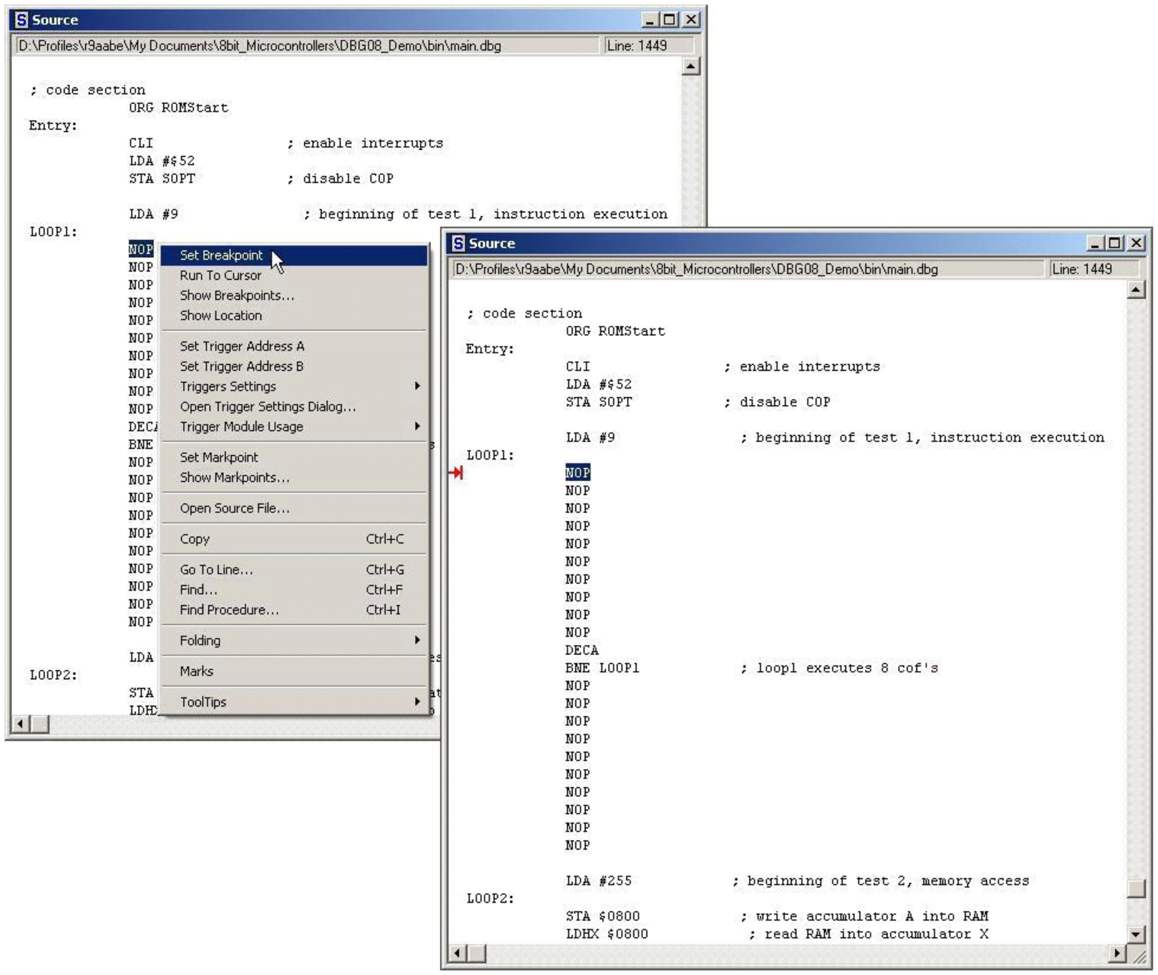
Breakpoints and Triggers

The on-chip ICE support interface allows the user to set and remove breakpoints and triggers quickly. This allows the user to adjust the breakpoint and trigger positions in their software repeatedly to acquire the information needed to troubleshoot software.

Setting Trace Breakpoints

To set a breakpoint, right click next to the instruction or software line and choose “Set Breakpoint” from the popup menu. The breakpoint will be represented by a red arrow. The 16-bit address marked by the red arrow will be set in the available hardware comparator. This breakpoint will halt software execution prior to executing that instruction or software line without performing a capture. These steps can be repeated to set up as many as three breakpoints (total of three hardware comparators, see System Details) with no triggers.

Figure 2 illustrates setting a breakpoint.



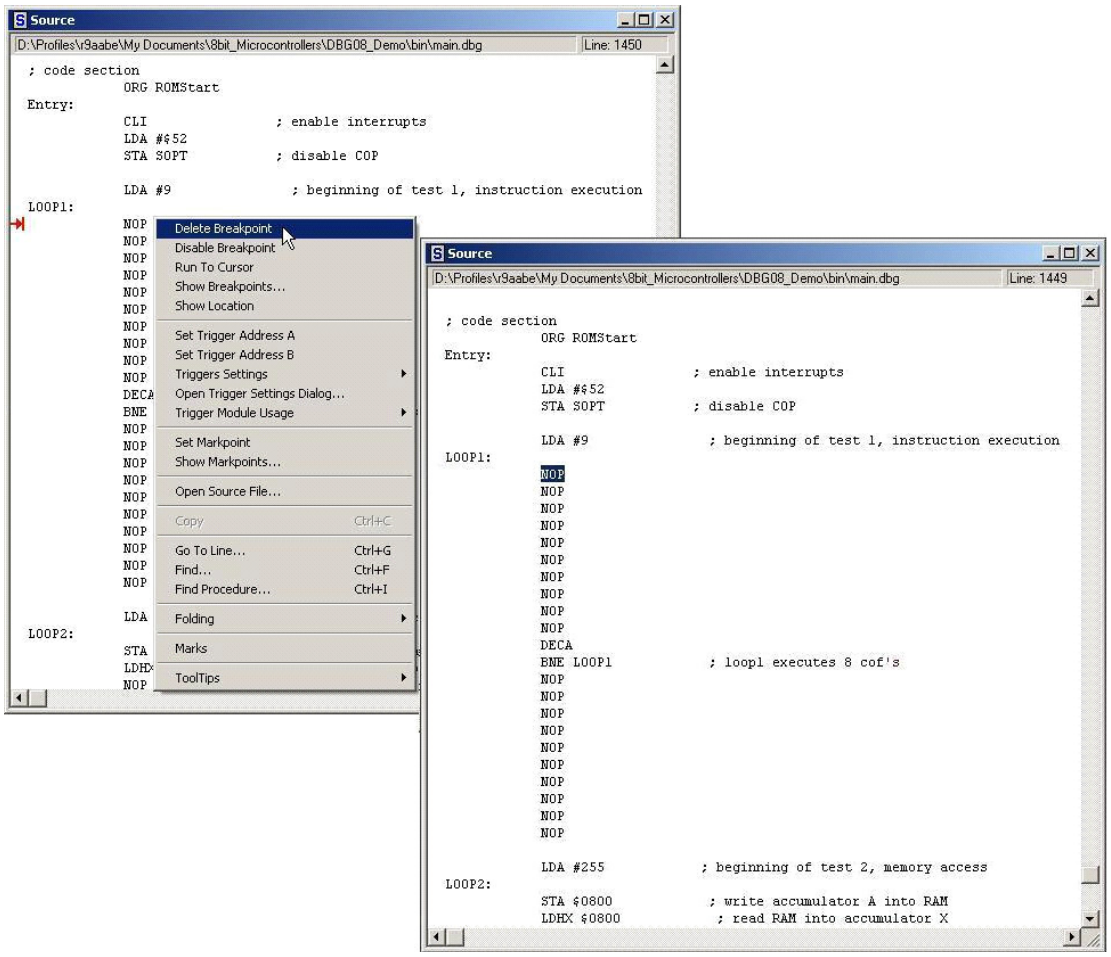
```
; code section
            ORG ROMStart
Entry:
            CLI                 ; enable interrupts
            LDA #$52
            STA SOPT            ; disable COP

            LDA #9                ; beginning of test 1, instruction execution
LOOP1:
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            DECA
            BNE LOOP1             ; loop1 executes 8 cof's
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP

            LDA #255             ; beginning of test 2, memory access
LOOP2:
            STA $0800             ; write accumulator A into RAM
            LDHX $0800             ; read RAM into accumulator X
```

Figure 2. Setting a Breakpoint

Removing Trace Breakpoints

To remove a breakpoint, right click next to the instruction or software line with the breakpoint and choose “Delete Breakpoint” from the popup menu. The red arrow representing the breakpoint will disappear. This action disassociates the 16-bit address set to the corresponding hardware comparator.

Figure 3 illustrates removing a breakpoint.



```
D:\Profiles\r9aabe\My Documents\8bit_Microcontrollers\DBG08_Demo\bin\main.dbg    Line: 1450

; code section
            ORG ROMStart
Entry:
            CLI                ; enable interrupts
            LDA #$52
            STA SOPT           ; disable COP

            LDA #9               ; beginning of test 1, instruction execution
LOOP1:
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            DECA
            BNE
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP

            LDA
LOOP2:
            STA
            LDHX
            NOP
```

```
D:\Profiles\r9aabe\My Documents\8bit_Microcontrollers\DBG08_Demo\bin\main.dbg    Line: 1449

; code section
            ORG ROMStart
Entry:
            CLI                ; enable interrupts
            LDA #$52
            STA SOPT           ; disable COP

            LDA #9               ; beginning of test 1, instruction execution
LOOP1:
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            DECA
            BNE LOOP1            ; loop1 executes 8 cof's
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP

            LDA #255           ; beginning of test 2, memory access
LOOP2:
            STA $0800            ; write accumulator A into RAM
            LDHX $0800            ; read RAM into accumulator X
```

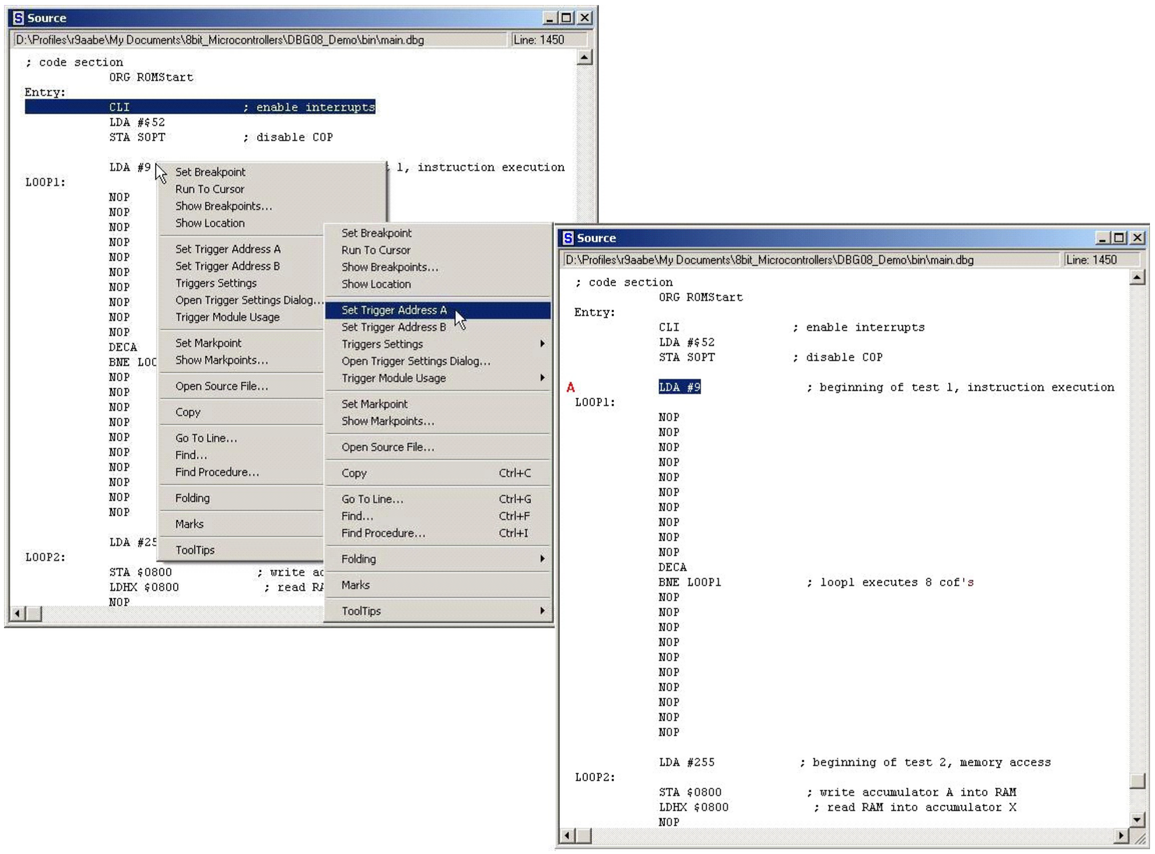
Figure 3. Removing a Breakpoint

Note that a breakpoint menu is available by right clicking anywhere in the source or assembly component window and selecting “Show Breakpoints”. This menu lists all the breakpoint addresses set by the user. Also, it allows the user to delete multiple breakpoints that might not be visible in the currently opened source or assembly component window. This menu is another way to verify that your breakpoints have been set.

Setting Source Triggers

To set a trigger, right click next to the instruction or software line and choose “Set Trigger Address X” from the popup menu. The trigger will be represented by a red letter A or B, depending on which of the two triggers the user sets. This action takes the 16-bit address associated with the instruction at the A or B symbol and sets it in the corresponding hardware comparator. This trigger will execute a real-time bus capture based on that instruction or software line. (Capture is dependent on trigger settings.) These same steps can be repeated to set the second available trigger.

Figure 4 illustrates setting a trigger in software.



```
; code section
            ORG ROMStart
Entry:
            CLI                   ; enable interrupts
            LDA #$52
            STA SOPT              ; disable COP

            LDA #9                ; 1, instruction execution
LOOP1:
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            DECA
            BNE LOC
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP

            LDA #25
LOOP2:
            STA $0800             ; write ac
            LDHX $0800             ; read RA
            NOP
```

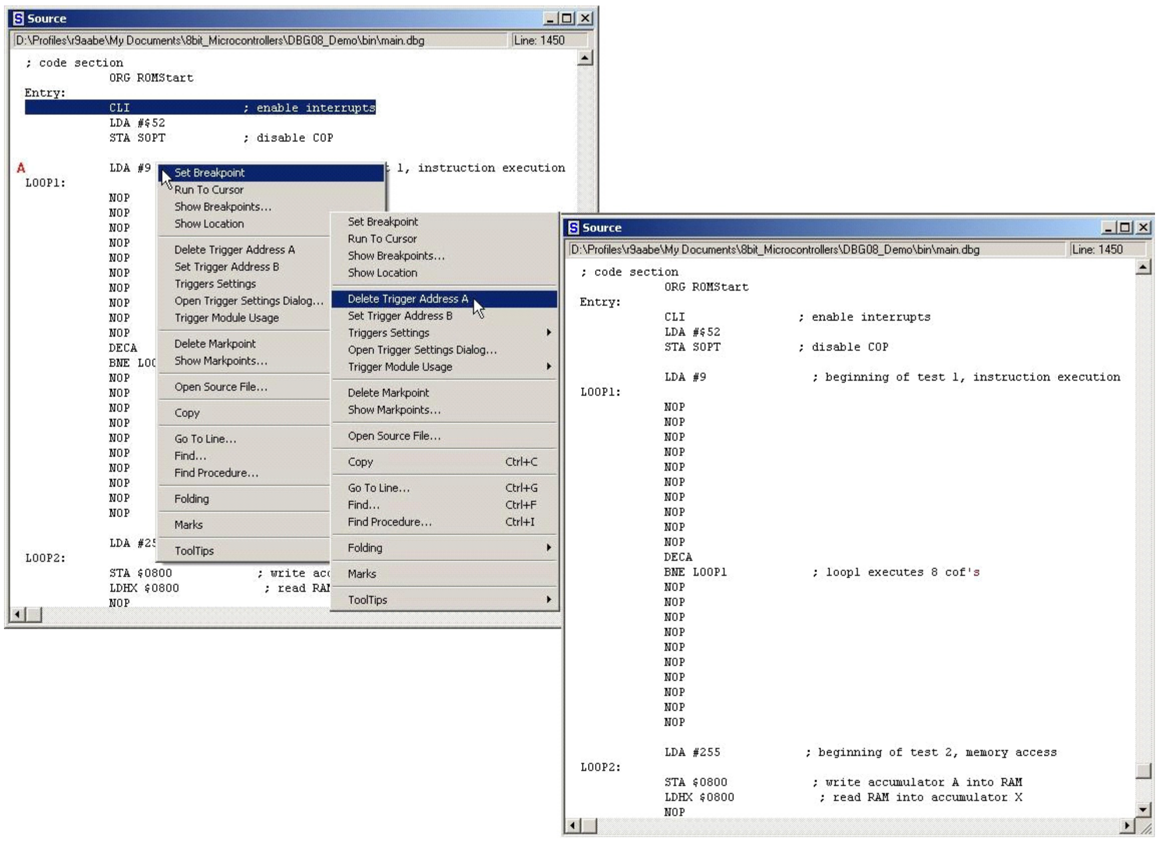
```
; code section
            ORG ROMStart
Entry:
            CLI                   ; enable interrupts
            LDA #$52
            STA SOPT              ; disable COP

            LDA #9                 ; beginning of test 1, instruction execution
LOOP1:
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            DECA
            BNE LOOP1              ; loop1 executes 8 cof's
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP

            LDA #255              ; beginning of test 2, memory access
LOOP2:
            STA $0800              ; write accumulator A into RAM
            LDHX $0800              ; read RAM into accumulator X
            NOP
```

Figure 4. Setting a Trigger

Removing Source Triggers

To remove a trigger, right click anywhere on the assembly or source component window and choose “Delete Trigger Address X” from the popup menu. If more than one trigger is set, choose the one to be deleted. The red letter representing the appropriate trigger will disappear. This action disassociates the 16-bit address set to the corresponding hardware comparator.

Figure 5 illustrates removing a trigger.



D:\Profiles\r9aabe\My Documents\8bit_Microcontrollers\DBG08_Demo\bin\main.dbg   Line: 1450

```
 ; code section
            ORG ROMStart
 Entry:
            CLI                   ; enable interrupts
            LDA #$52
            STA SOPT              ; disable COP

            LDA #9                  ; beginning of test 1, instruction execution
 LOOP1:
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            DECA
            BNE LOOP1               ; loop1 executes 8 cof's
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP

            LDA #255              ; beginning of test 2, memory access
 LOOP2:
            STA $0800               ; write accumulator A into RAM
            LDHX $0800               ; read RAM into accumulator X
            NOP
```

Figure 5. Removing a Trigger

Memory and Data Components

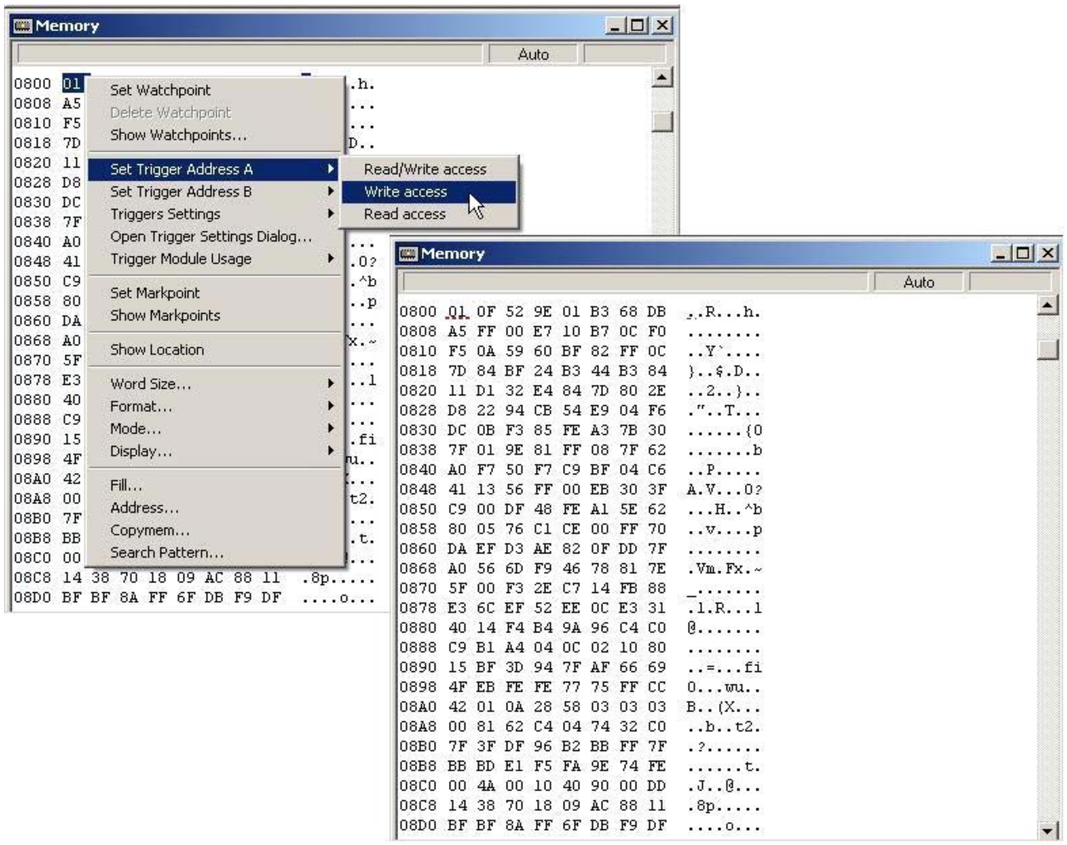
Note that the same procedures for setting and removing triggers can be applied in the memory and data component windows. The difference is that in the memory window, the user will typically trigger on a memory access, instead of an opcode address, at the chosen memory location. In the data component window, the user can set a trigger based on an access to a local or global variable.

Setting Memory Triggers

To set a trigger in the memory component window, right click on the desired memory location to be triggered on. From the popup menu choose "Set Trigger Address X." Then a popup menu will appear with the options: "Read/Write access," "Write access," and "Read access." This selection will tell the debugger what type of memory access to trigger on for the selected memory location.

The trigger will be represented by a red or blue hashed line below the chosen memory location (trigger address A is red and B is blue). This action takes the 16-bit address associated with the selected memory location and sets it in the corresponding hardware comparator. This trigger will execute a real-time bus capture based on the access to this memory location (capture is dependent on trigger settings). These same steps can be repeated to set the second available trigger.

Figure 6 illustrates setting a trigger in the memory component window.



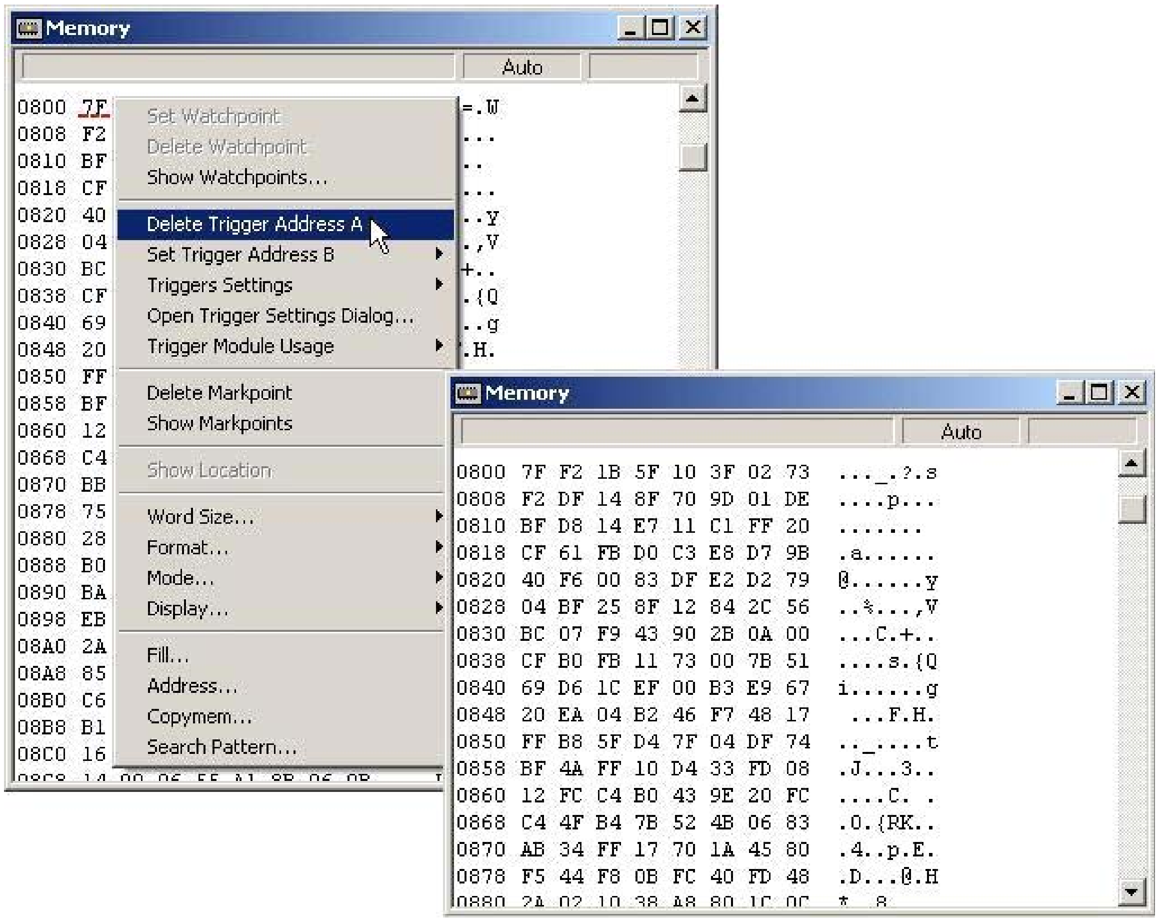
```
0800 01 0F 52 9E 01 B3 68 DB   ..R...h.
0808 A5 FF 00 E7 10 B7 0C F0   ........
0810 F5 0A 59 60 BF 82 FF 0C   ..Y`....
0818 7D 84 BF 24 B3 44 B3 84   }..$.D..
0820 11 D1 32 E4 84 7D 80 2E   ..2..}..
0828 D8 22 94 CB 54 E9 04 F6   ."..T...
0830 DC 0B F3 85 FE A3 7B 30   ......{0
0838 7F 01 9E 81 FF 08 7F 62   .......b
0840 A0 F7 50 F7 C9 BF 04 C6   ..P.....
0848 41 13 56 FF 00 EB 30 3F   A.V...0?
0850 C9 00 DF 48 FE A1 5E 62   ...H..^b
0858 80 05 76 C1 CE 00 FF 70   ..v....p
0860 DA EF D3 AE 82 0F DD 7F   ........
0868 A0 56 6D F9 46 78 81 7E   .Vm.Fx.~
0870 5F 00 F3 2E C7 14 FB 88   _.......
0878 E3 6C EF 52 EE 0C E3 31   .l.R...1
0880 40 14 F4 B4 9A 96 C4 C0   @.......
0888 C9 B1 A4 04 0C 02 10 80   ........
0890 15 BF 3D 94 7F AF 66 69   ..=...fi
0898 4F EB FE FE 77 75 FF CC   O...wu..
08A0 42 01 0A 28 58 03 03 03   B..(X...
08A8 00 81 62 C4 04 74 32 C0   ..b..t2.
08B0 7F 3F DF 96 B2 BB FF 7F   .?......
08B8 BB BD E1 F5 FA 9E 74 FE   ......t.
08C0 00 4A 00 10 40 90 00 DD   .J..@...
08C8 14 38 70 18 09 AC 88 11   .8p.....
08D0 BF BF 8A FF 6F DB F9 DF   ....o...
```

Figure 6. Setting a Memory Access Trigger

Removing Memory Triggers

To remove a trigger from the memory component window, right click anywhere on the window and choose “Delete Trigger Address X” from the popup menu. If more than one trigger is set, choose the ones to be deleted. The colored hashed line representing the appropriate trigger will disappear. This action disassociates the 16-bit address set to the corresponding hardware comparator.

Figure 7 illustrates removing a trigger in the memory component window.



```
0800  7F
0808  F2
0810  BF
0818  CF
0820  40
0828  04
0830  BC
0838  CF
0840  69
0848  20
0850  FF
0858  BF
0860  12
0868  C4
0870  BB
0878  75
0880  28
0888  B0
0890  BA
0898  EB
08A0  2A
08A8  85
08B0  C6
08B8  B1
08C0  16
```

```
0800  7F F2 1B 5F 10 3F 02 73   ..._.?.s
0808  F2 DF 14 8F 70 9D 01 DE   ....p...
0810  BF D8 14 E7 11 C1 FF 20   .......
0818  CF 61 FB D0 C3 E8 D7 9B   .a......
0820  40 F6 00 83 DF E2 D2 79   @......y
0828  04 BF 25 8F 12 84 2C 56   ..%...,V
0830  BC 07 F9 43 90 2B 0A 00   ...C.+..
0838  CF B0 FB 11 73 00 7B 51   ....s.{Q
0840  69 D6 1C EF 00 B3 E9 67   i......g
0848  20 EA 04 B2 46 F7 48 17    ...F.H.
0850  FF B8 5F D4 7F 04 DF 74   .._....t
0858  BF 4A FF 10 D4 33 FD 08   .J...3..
0860  12 FC C4 B0 43 9E 20 FC   ....C. .
0868  C4 4F B4 7B 52 4B 06 83   .O.{RK..
0870  AB 34 FF 17 70 1A 45 80   .4..p.E.
0878  F5 44 F8 0B FC 40 FD 48   .D...@.H
```

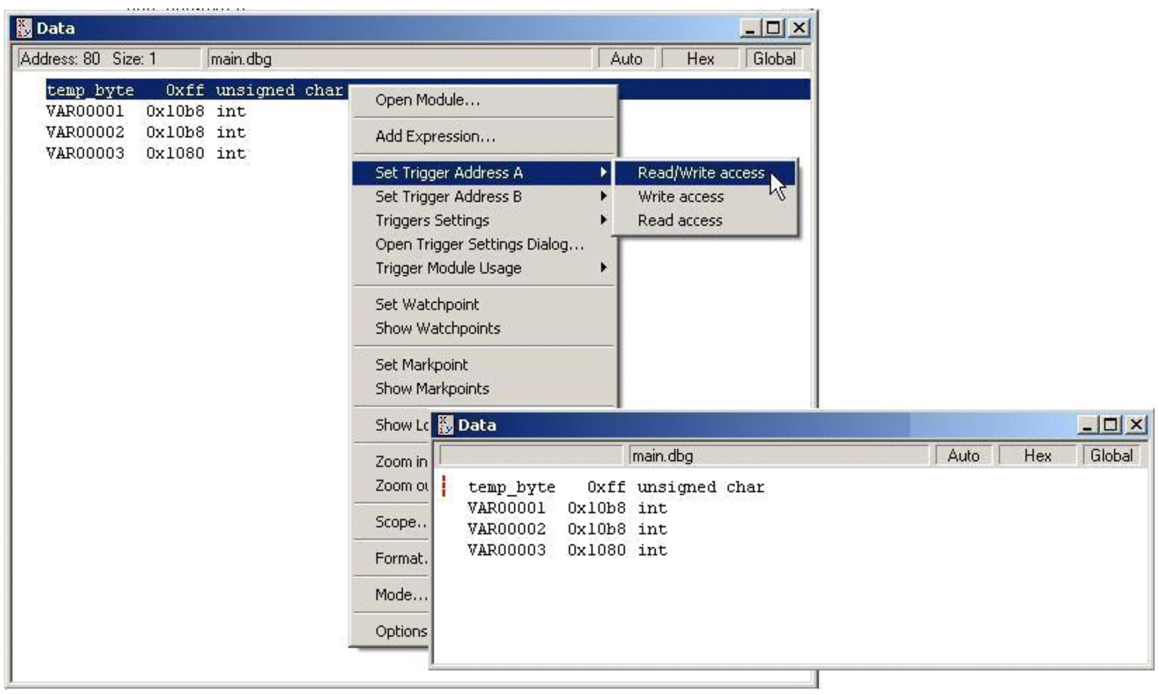
Figure 7. Removing a Memory Access Trigger

Setting Data Triggers

To set a trigger in the data component window, right click on the variable to be triggered on. From the popup menu choose “Set Trigger Address X.” Then a popup menu will appear with the options: “Read/Write access,” “Write access,” and “Read access.” This selection will tell the debugger what type of memory access to trigger on for the selected variable.

The trigger will be represented by a red or blue hashed line next to the chosen variable label (trigger address A is red and B is blue). This action takes the 16-bit address associated with the selected variable and sets it in the corresponding hardware comparator. This trigger will execute a real-time bus capture based on the access to this memory location. (Capture is dependent on trigger settings.) These same steps can be repeated to set the second available trigger.

Figure 8 illustrates setting a trigger in the data component window.



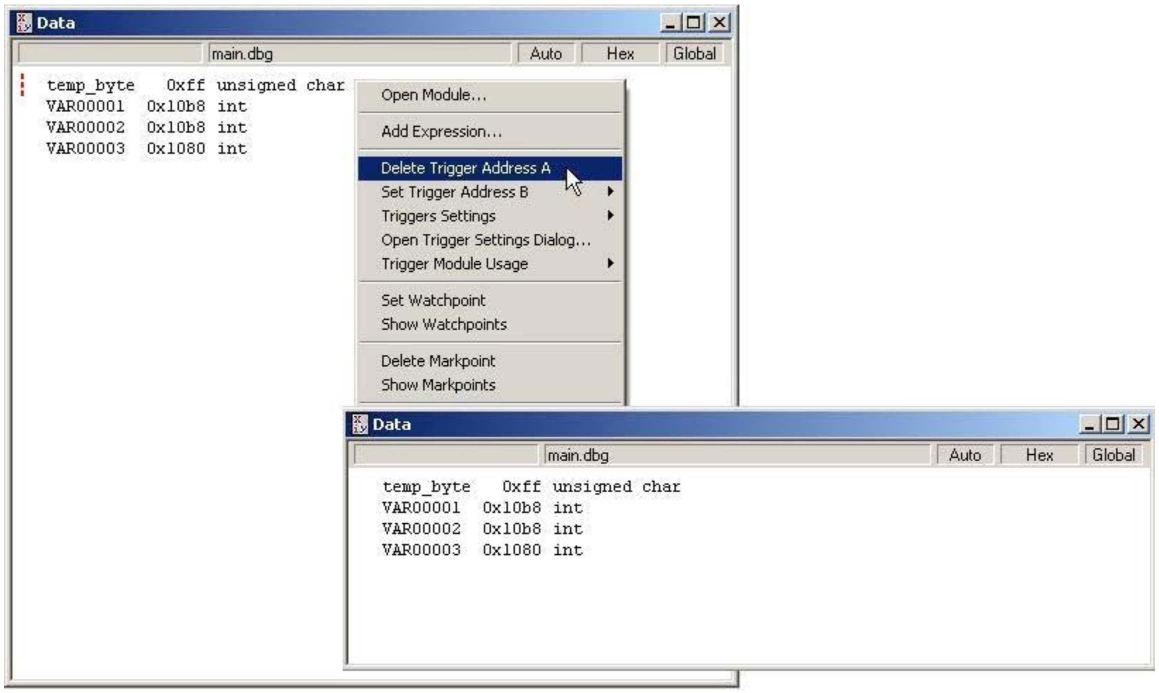
```
temp_byte   0xff  unsigned char
VAR00001   0x10b8  int
VAR00002   0x10b8  int
VAR00003   0x1080  int
```

Figure 8. Setting a Memory Access Trigger on a Variable

Removing Data Triggers

To remove a trigger from the data component window, right click anywhere on the window and choose “Delete Trigger Address X” from the popup menu. If more than one trigger is set, choose the one to be deleted. The colored hashed line representing the appropriate trigger will disappear. This action disassociates the 16-bit address set to the corresponding hardware comparator.

Figure 9 illustrates removing a trigger in the data component window.



```
temp_byte   0xff  unsigned char
VAR00001   0x10b8  int
VAR00002   0x10b8  int
VAR00003   0x1080  int
```

```
temp_byte   0xff  unsigned char
VAR00001   0x10b8  int
VAR00002   0x10b8  int
VAR00003   0x1080  int
```

Figure 9. Removing a Memory Access Trigger on a Variable

Trigger Conditions and Capture Options

Trigger settings must be configured in order to perform a useful trace capture. This involves choosing a trigger condition and setting the capture options. These settings will determine when the capture will occur and what information the trace buffer will capture. The on-chip ICE support interface allows the user to easily set or change these settings.

After a trigger has been set, right click anywhere on the component window. Then scroll down and select “Trigger Settings.” A popup menu will appear displaying the applicable trigger conditions (top part of popup menu). The options displayed depend on the number of trigger points set and whether this was an opcode or memory access trigger.

The capture options are displayed in the bottom part of the popup menu. Notice that a checkmark beside the settings indicates the default trigger settings. To change the settings, select the appropriate trigger condition and/or capture option in the menu. After selecting the setting, the checkmark should adjust to your selection.

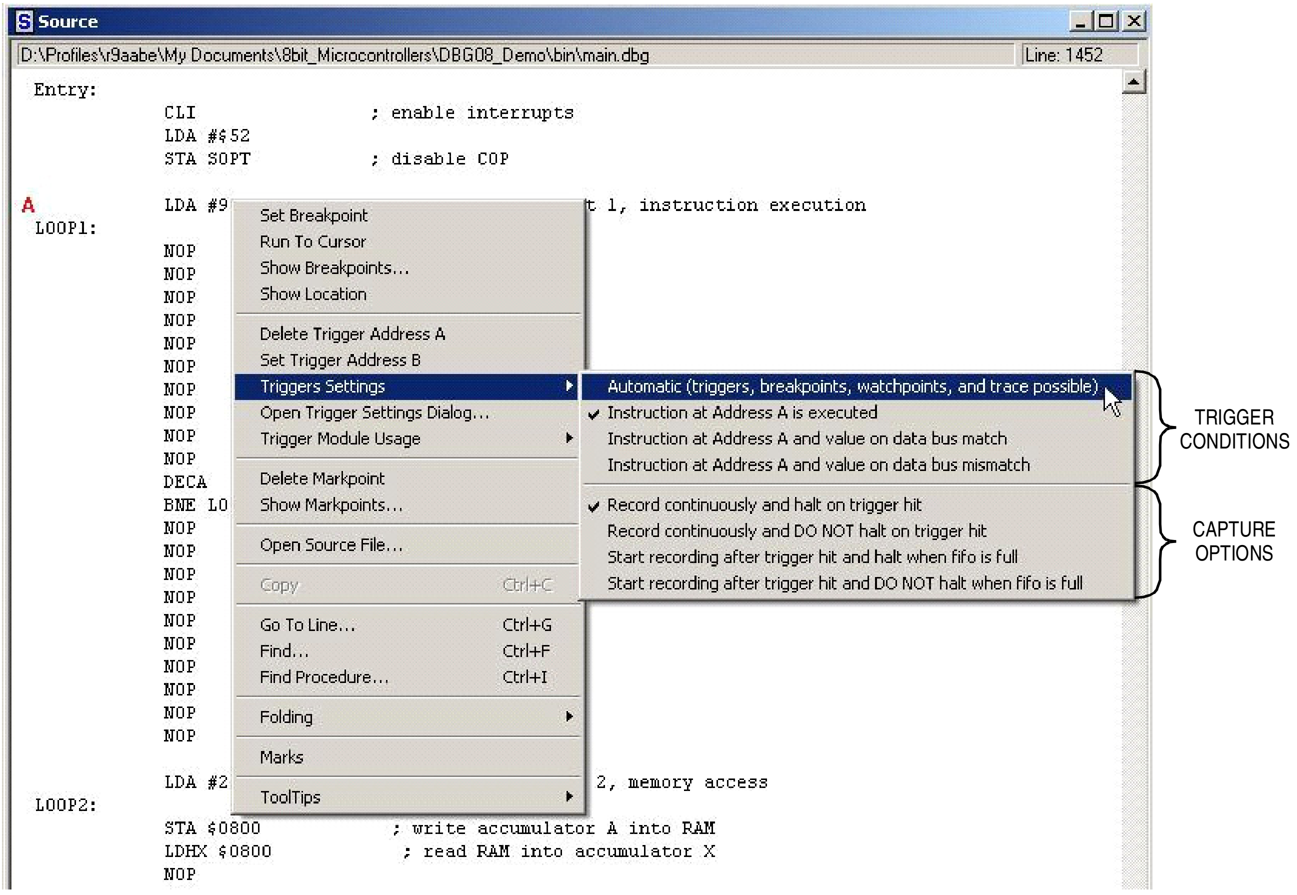
To change both the trigger condition and capture option, perform this step twice. Set the trigger condition first, and set the capture option second.

NOTE

Check the trigger settings before performing the trace on application software to ensure that the debugger applied the correct settings.

Figure 10 illustrates how to set or change your trigger condition and capture options in a **source or assembly component window**.

Figure 11 illustrates how to set or change your trigger condition and capture options in a **memory or data component window**.


Source
D:\Profiles\r9aabe\My Documents\8bit_Microcontrollers\DBG08_Demo\bin\main.dbg
Line: 1452
A
Set Breakpoint
Run To Cursor
Show Breakpoints...
Show Location
Delete Trigger Address A
Set Trigger Address B
Triggers Settings
Open Trigger Settings Dialog...
Trigger Module Usage
Delete Markpoint
Show Markpoints...
Open Source File...
Copy Ctrl+C
Go To Line... Ctrl+G
Find... Ctrl+F
Find Procedure... Ctrl+I
Folding
Marks
ToolTips
Automatic (triggers, breakpoints, watchpoints, and trace possible)
✓ Instruction at Address A is executed
Instruction at Address A and value on data bus match
Instruction at Address A and value on data bus mismatch
✓ Record continuously and halt on trigger hit
Record continuously and DO NOT halt on trigger hit
Start recording after trigger hit and halt when fifo is full
Start recording after trigger hit and DO NOT halt when fifo is full
TRIGGER CONDITIONS
CAPTURE OPTIONS

```
Entry:
            CLI                 ; enable interrupts
            LDA #$52
            STA SOPT            ; disable COP

            LDA #9                           t 1, instruction execution
LOOP1:
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            DECA
            BNE LO
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP

            LDA #2                           2, memory access
LOOP2:
            STA $0800             ; write accumulator A into RAM
            LDHX $0800             ; read RAM into accumulator X
            NOP
```

Figure 10. Setting/Changing Trigger Settings for Opcode Triggers

NOTE

Figure 10 represents the trigger settings available for a single instruction trigger. The number of trigger conditions available will vary according to the number of triggers set (maximum two triggers). Also, the trigger settings will vary if the trigger is placed in the memory component window triggering on memory accesses or data events as shown in Figure 11.

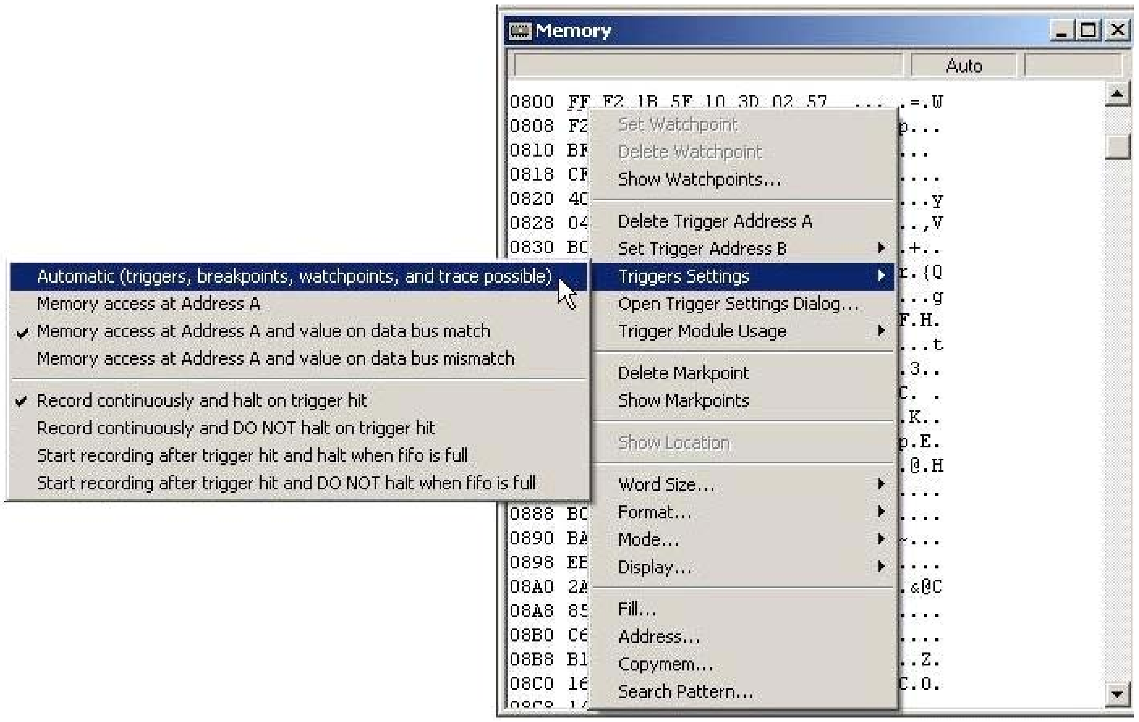

Memory
Auto
Set Watchpoint
Delete Watchpoint
Show Watchpoints...
Delete Trigger Address A
Set Trigger Address B
Triggers Settings
Open Trigger Settings Dialog...
Trigger Module Usage
Delete Markpoint
Show Markpoints
Show Location
Word Size...
Format...
Mode...
Display...
Fill...
Address...
Copymem...
Search Pattern...
Automatic (triggers, breakpoints, watchpoints, and trace possible)
Memory access at Address A
✓ Memory access at Address A and value on data bus match
Memory access at Address A and value on data bus mismatch
✓ Record continuously and halt on trigger hit
Record continuously and DO NOT halt on trigger hit
Start recording after trigger hit and halt when fifo is full
Start recording after trigger hit and DO NOT halt when fifo is full

Figure 11. Set/Change Trigger Settings for Memory Access Triggers

The listed trigger conditions will determine when the trigger is qualified. In Figure 10, “Instruction at Address A is executed” is selected. This condition means that a capture will be performed when the instruction marked by “Trigger Address A” is executed.

In Figure 11, “Memory access at Address A and value on data bus match” is selected. This condition means that a capture will be performed when a user-set value (comparator B) matches the data bus value at the memory location associated with Trigger Address A.

Similar rules follow for more complex conditions listed at System Details. Notice that the CodeWarrior tool trigger conditions are more elaborate in writing, but perform the same action as nine listed trigger conditions at System Details.

Table 1 describes the meaning behind the CodeWarrior tool labels, along with the corresponding MCU trigger condition name:

Table 1. Trigger Definitions

CodeWarrior Tool Trigger Setting and Description	On-Chip ICE Trigger Mode	Trigger Type
1. Instruction at Address A is executed	A only	Tagged
A capture will be performed when the instruction marked by trigger address A is executed		
2. Instruction at Address A or Address B is executed	A or B	Tagged
A capture will be performed when the instruction marked by trigger address A or trigger address B is executed		
3. Instruction execution inside Address A – Address B range	Inside range (A<= address <=B)	Tagged
A capture will be performed when any instruction within the trigger address A and trigger address B boundaries is executed		
4. Instruction execution outside Address A – Address B range	Outside range (address <=A or address >=B)	Tagged
A capture will be performed when any instruction outside the trigger address A and trigger address B boundaries are executed		
5. Instruction at Address A then at Address B were executed	A then B	Tagged
A capture will be performed when the instructions at trigger address A and trigger address B occur in sequence		
6. Memory access at Address A	A Only	Forced
A capture will be performed when the memory address marked by trigger address A is accessed		
7. Memory access at Address A or at Address B	A Only	Forced
A capture will be performed when the memory address marked by trigger address A or trigger address B is accessed		
8. Memory access inside Address A – Address B range	Inside range (A<= address <= B	Forced
A capture will be performed when a memory address within the trigger address A and trigger address B boundaries is accessed		
9. Memory access outside Address A – Address B range	Outside range (address <=A or address >=B	Forced
A capture will be performed when a memory address within the trigger address A and trigger address B boundaries is accessed		
10. Memory access at Address A then memory access at Address B	A then B	Forced
A capture will be performed when a memory address outside the trigger address A and trigger address B boundaries is accessed		

Table 1. Trigger Definitions (Continued)

CodeWarrior Tool Trigger Setting and Description	On-Chip ICE Trigger Mode	Trigger Type
11. Memory access at Address and value on data bus match	A and B data	Forced
A capture will be performed when a user-set value (comparator B) matches the data bus value at the memory location associated with trigger address A		
12. Memory access at Address and value on data bus mismatch	A and B data	Forced
A capture will be performed when a user-set value (comparator B) does not match the data bus value at the memory location associated with trigger address A		
13. Capture the read/write values at Address B	Event Only B	
Any read/write event at the memory location associated with trigger address B will be captured		
14. Capture read/write values at Address B after access at Address A	A then Event Only B	
Any read/write event at memory location associated with trigger address B will be captured after any access has been performed at trigger address A		

With each trigger condition, the user must choose the capture option that will determine when the buffer will capture data based on the trigger point. The capture option will also define the completion of a trigger sequence. When the trigger sequence is complete and the target CPU returns to BDM, the captured data will be displayed in the trace component window.

Table 2 describes when a trigger sequence will terminate and when captured data will be displayed for each capture option.

Table 2. Capture Option Definitions

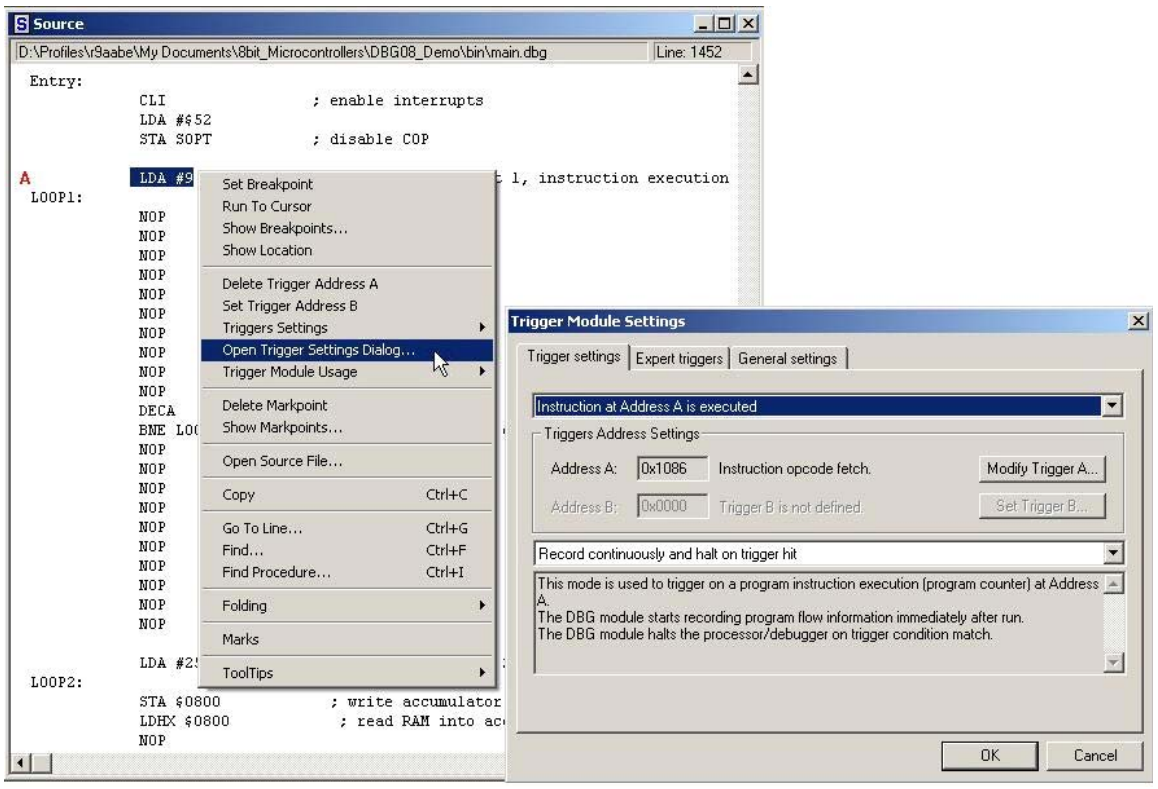
Codewarrior Capture Option	Trigger and CPU Action
1. Record continuously and halt on trigger hit	End trigger and halt CPU
Trigger sequence is complete when the trigger condition is qualified. The data will be displayed on the trace window immediately after the CPU halts and returns to BDM	
2. Record continuously and DO NOT halt on trigger hit	End trigger and DO NOT halt CPU
Trigger sequence is complete when the trigger condition is qualified. The data will not be displayed immediately since CPU execution did not halt. The display of data will occur when the user manually halts the target CPU and returns to BDM	
3. Start recording after trigger hit and halt when FIFO is full	Begin trigger and halt CPU
Trigger sequence is complete when the FIFO becomes full following a qualified trigger condition. The data will be displayed on the trace window immediately after the FIFO becomes full and the CPU halts returning to BDM	
4. Start recording after trigger hit and DO NOT halt when FIFO is full	Begin trigger and DO NOT halt CPU
Trigger sequence is complete when the FIFO becomes full following a qualified trigger condition. The data will not be displayed immediately since the CPU execution did not halt. The display of data will occur when the user manually halts the target CPU and returns to BDM	

Verifying Trigger Settings

One way to verify that the trigger condition and capture options have been set correctly is to repeat the steps described in Trigger Conditions and Capture Options. However, a second method is to open the Trigger Module Settings menu. This menu gives you detailed information about the currently configured trigger.

To bring up the trigger module settings menu, right click anywhere on the data component window. On the popup menu, scroll down and select “Open Trigger Settings Dialog.” A fixed trigger module settings menu will appear that displays the current trigger condition, trigger comparator addresses, and capture options. The menu also has a short description of what this trigger combination will perform.

Figure 12 demonstrates how to open the trigger module settings menu.



D:\Profiles\r9aabe\My Documents\8bit_Microcontrollers\DBG08_Demo\bin\main.dbg Line: 1452

```
Entry:
            CLI                ; enable interrupts
            LDA #$52
            STA SOPT           ; disable COP

A           LDA #9                                   1, instruction execution
LOOP1:
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            DECA
            BNE LOO
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP

            LDA #2
LOOP2:
            STA $0800             ; write accumulator
            LDHX $0800             ; read RAM into ac
            NOP
```

Instruction at Address A is executed

Triggers Address Settings

Address A: 0x1086 Instruction opcode fetch.

Address B: 0x0000 Trigger B is not defined.

Record continuously and halt on trigger hit

This mode is used to trigger on a program instruction execution (program counter) at Address A.
The DBG module starts recording program flow information immediately after run.
The DBG module halts the processor/debugger on trigger condition match.

Figure 12. Opening Trigger Module Settings Menu

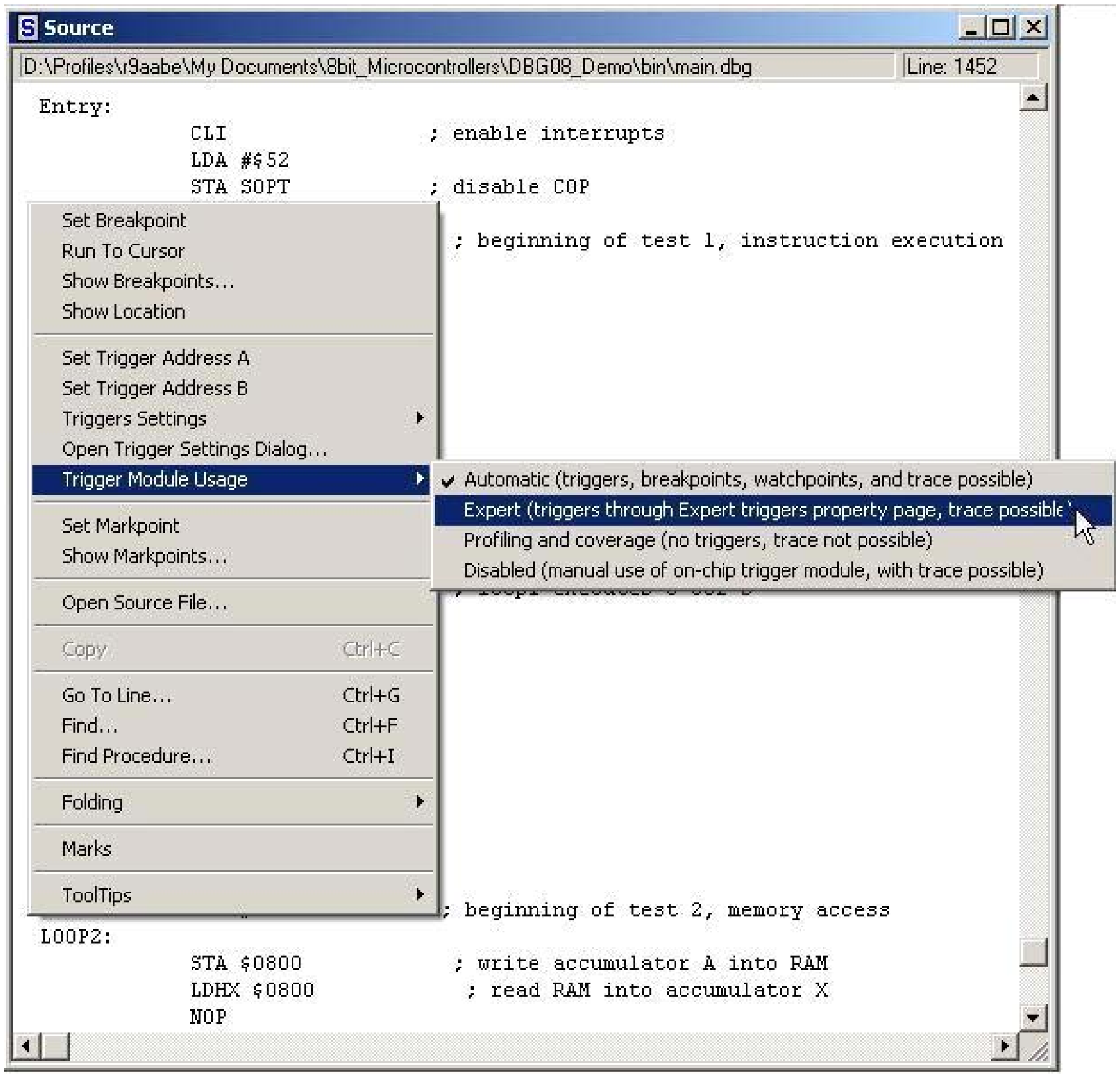
Trigger Module Modes

The CodeWarrior on-chip ICE user interface for setting up trace captures has two modes. The level described up to this point is automatic mode. The CodeWarrior tools will default to automatic mode. This level provides the user with an easy-to-navigate point-and-click solution for setting up trace captures. Automatic mode configures detailed capture options (such as tagged versus forced triggers) depending the component window of the trigger (tagged — source or assembly window; forced — memory or data window).

This mode also provides the user with a list of predetermined trigger conditions depending on the component window of the trigger (instruction execution — source or assembly window; memory access — memory or data window). Automatic mode allows the user to select settings as they would in an ordinary bus-state analyzer tool. It does not require the user to know the bit-level controls of the module. Automatic mode will be used for the software debug scenarios discussed later in this application note.

Expert Mode

In expert mode, the user has bit-level control of the on-chip ICE system. To change between automatic and expert mode, right click anywhere in any component window. In the popup menu, scroll down and select “Trigger Module Usage.” In the secondary popup menu, select the “Expert (….)” mode option. The checkmark set on “Automatic (….)” will move to “Expert (….).” Figure 13 demonstrates the switch to expert mode in the source window.



Source

D:\Profiles\r9aabe\My Documents\8bit_Microcontrollers\DBG08_Demo\bin\main.dbg   Line: 1452

```
Entry:
            CLI             ; enable interrupts
            LDA #$52
            STA SOPT        ; disable COP
                              ; beginning of test 1, instruction execution
                             ; beginning of test 2, memory access
LOOP2:
            STA $0800         ; write accumulator A into RAM
            LDHX $0800         ; read RAM into accumulator X
            NOP
```

Set Breakpoint
Run To Cursor
Show Breakpoints...
Show Location
Set Trigger Address A
Set Trigger Address B
Triggers Settings
Open Trigger Settings Dialog...
Trigger Module Usage
Set Markpoint
Show Markpoints...
Open Source File...
Copy Ctrl+C
Go To Line... Ctrl+G
Find... Ctrl+F
Find Procedure... Ctrl+I
Folding
Marks
ToolTips
✓ Automatic (triggers, breakpoints, watchpoints, and trace possible)
Expert (triggers through Expert triggers property page, trace possible)
Profiling and coverage (no triggers, trace not possible)
Disabled (manual use of on-chip trigger module, with trace possible)

Figure 13. Switching to Expert Mode

In expert mode, the user must configure the “Trigger Module Settings” menu. The user will set and remove the triggers in the software using the same steps explained in Breakpoints and Triggers. However, the trigger selection now reads “Set DBGCX” instead of “Set Trigger Address X”. DBGCX represents the hardware comparator registers within the module. Also, the trigger will be marked with a different symbol than in automatic mode.

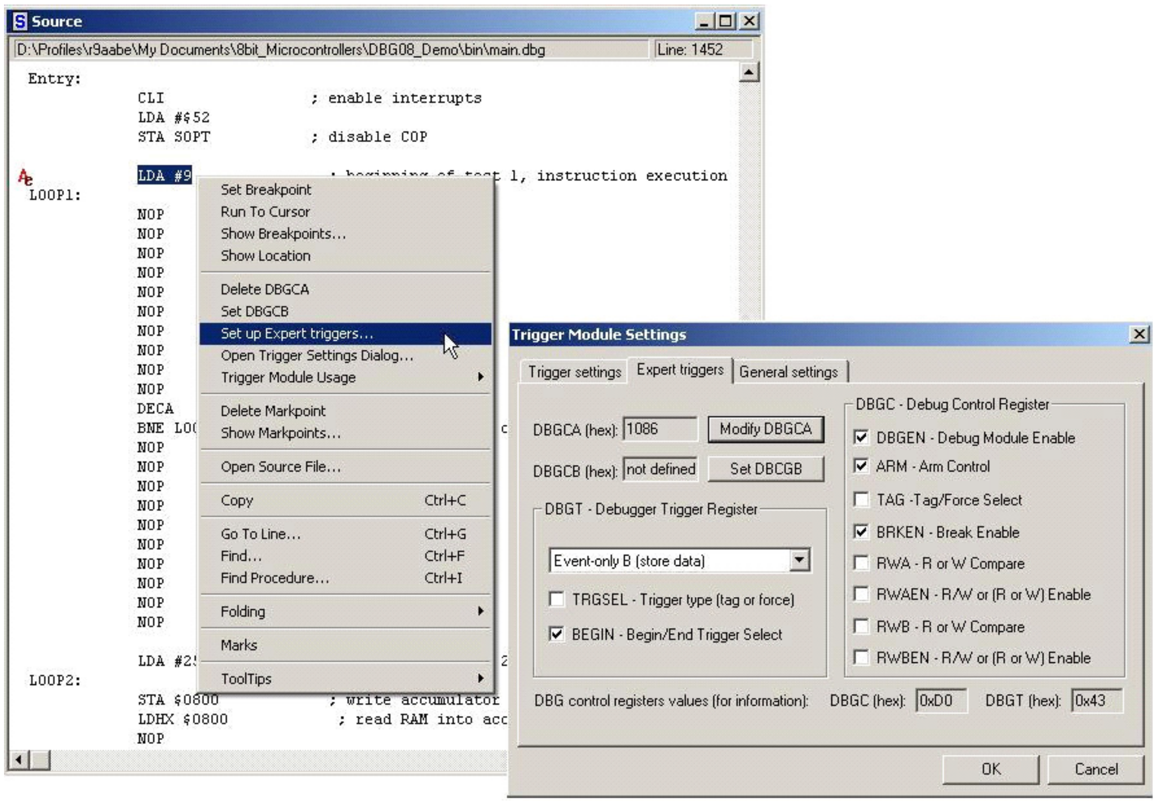
In the source and assembly component windows, the trigger will label trigger address A and B, but an “e” subscript will be added to represent expert mode. In the memory and data component windows, the hashed line labeling the respective trigger address A or B will change colors from red and blue to magenta and cyan.

Expert Triggers

After the triggers are placed, the user can set the trigger condition and capture options. To do this, right click the component window and select “Set up Expert triggers.” This will bring up the trigger module settings menu for expert triggers. The menu has a pull-down menu for the trigger conditions and several selection boxes for capture options (labeled by bit name). After configuring your desired trigger, select “OK.”

The trace capture is now configured for expert mode. To change the trigger settings, user must re-enter the trigger module settings menu.

Figure 14 demonstrates how to set or change the trigger condition or capture options in expert mode.



D:\Profiles\r9aabe\My Documents\8bit_Microcontrollers\DBG08_Demo\bin\main.dbg   Line: 1452

```
Entry:
            CLI                 ; enable interrupts
            LDA #$52
            STA SOPT            ; disable COP

            LDA #9              ; beginning of test 1, instruction execution
LOOP1:
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            DECA
            BNE LO(
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP

            LDA #2!
LOOP2:
            STA $0800           ; write accumulator
            LDHX $0800           ; read RAM into acc
            NOP
```

DBGCA (hex): 1086

DBGCB (hex): not defined

DBGT - Debugger Trigger Register: Event-only B (store data)

- [ ] TRGSEL - Trigger type (tag or force)
- [x] BEGIN - Begin/End Trigger Select

DBGC - Debug Control Register

- [x] DBGEN - Debug Module Enable
- [x] ARM - Arm Control
- [ ] TAG -Tag/Force Select
- [x] BRKEN - Break Enable
- [ ] RWA - R or W Compare
- [ ] RWAEN - R/W or (R or W) Enable
- [ ] RWB - R or W Compare
- [ ] RWBEN - R/W or (R or W) Enable

DBG control registers values (for information): DBGC (hex): 0xD0 DBGT (hex): 0x43

Figure 14. Setting/Changing Trigger Settings in Expert Mode

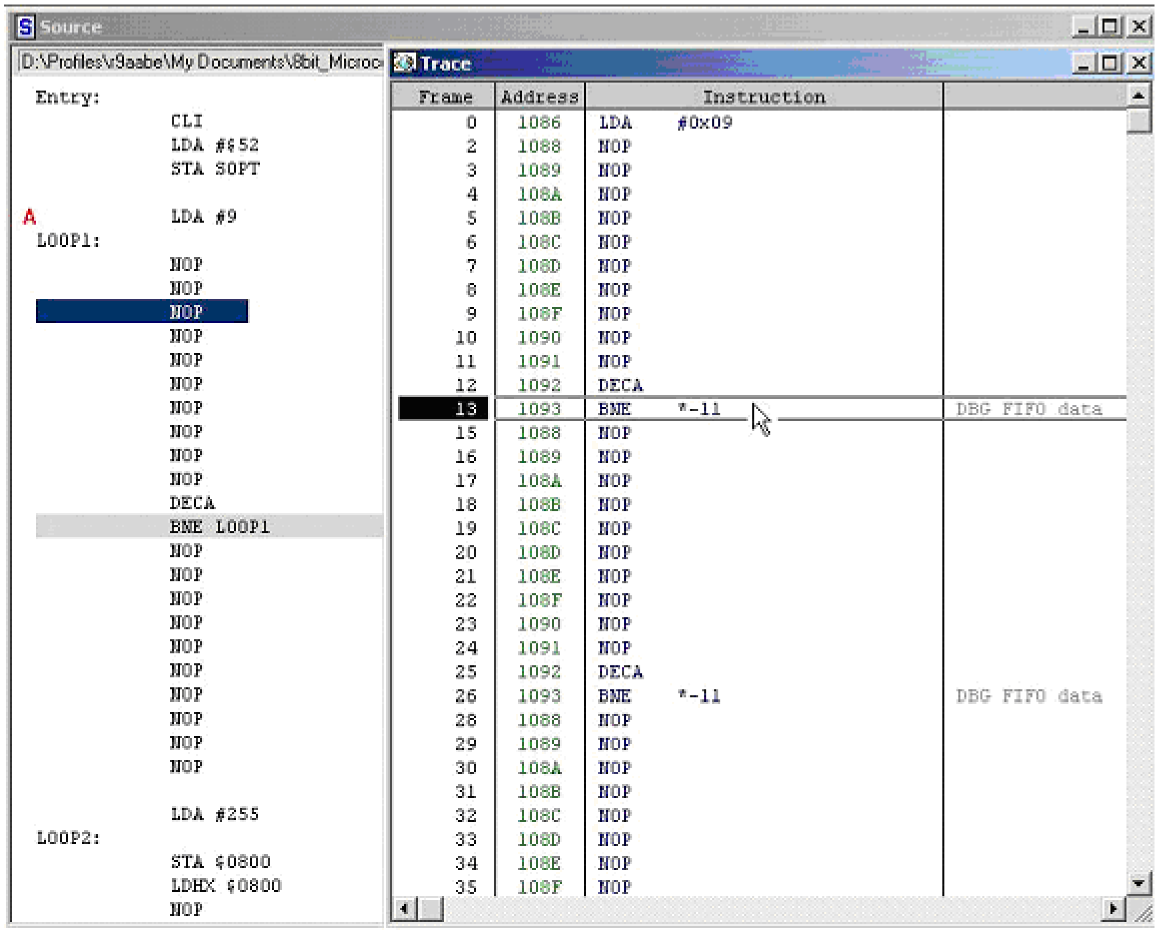
Performing the Bus Trace

Whether using automatic or expert mode to perform the bus trace capture, the method of initiating the trace is the same. After configuring the trigger point and settings, run the application software. Run the software by clicking on the green arrow located on the debugger toolbar. (Run can also be performed by pressing the F5 key.) When the trigger is condition is satisfied and the actions are completed, the FIFO buffer will display the event data or rebuild the software path from the recorded software change-of-flow addresses.

Analyzing the Bus Trace

To analyze the captured bus trace, select the trace component window. Following a successful bus trace, the window will display a list of frames that include event data or rebuilt instruction path. To observe the software execution path, click and hold the left mouse button and scroll through the rebuilt instruction frames. The selected instruction in the trace window will be highlighted gray in the source or assembly window. This feature allows you to observe the captured trace execution in the source window. The instruction highlighted blue in the source or assembly window marks the program counter at halt.

The program counter is marked to give you a point of reference in the captured data. From this information, the user can determine whether the software performed the intended sequence. Figure 15 illustrates the association between the rebuilt software execution path in the trace window and the actual software in the source window.



Source

D:\Profiles\v9aabe\My Documents\8bit_Microc

```
Entry:
                CLI
                LDA #$52
                STA SOPT
A               LDA #9
LOOP1:
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                DECA
                BNE LOOP1
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP

                LDA #255
LOOP2:
                STA $0800
                LDHX $0800
                NOP
```

Trace

Frame	Address	Instruction		
0	1086	LDA	#0x09	
2	1088	NOP		
3	1089	NOP		
4	108A	NOP		
5	108B	NOP		
6	108C	NOP		
7	108D	NOP		
8	108E	NOP		
9	108F	NOP		
10	1090	NOP		
11	1091	NOP		
12	1092	DECA		
13	1093	BNE	*-11	DBG FIFO data
15	1088	NOP		
16	1089	NOP		
17	108A	NOP		
18	108B	NOP		
19	108C	NOP		
20	108D	NOP		
21	108E	NOP		
22	108F	NOP		
23	1090	NOP		
24	1091	NOP		
25	1092	DECA		
26	1093	BNE	*-11	DBG FIFO data
28	1088	NOP		
29	1089	NOP		
30	108A	NOP		
31	108B	NOP		
32	108C	NOP		
33	108D	NOP		
34	108E	NOP		
35	108F	NOP		

Figure 15. Analyzing the Trace Component Window

Notice in Figure 15 that the recorded change-of-flow information is marked with “DBG FIFO data” in the trace component window. The remaining non-marked frames are then rebuilt instructions. Right clicking on the trace component window changes the type of information displayed on the window.

Debug Scenarios

The following sections will discuss the individual strengths of each trigger method available in the on-chip ICE and how each is used in different software debug scenarios.

Hardware Breakpoints

Trigger features in the on-chip ICE can be configured to perform traditional breakpoints. The difference between a breakpoint and an end trigger is that breakpoints do not record bus information.

With three hardware comparators (see System Details) available in the on-chip ICE, the user can set as many as three breakpoints. Recall that the three hardware comparators are shared resources with the two available triggers. Therefore, with no triggers set, the user can halt software execution prior to executing the instruction associated with three individual addresses.

General uses of breakpoints include:

- To verify that user software reaches a specific point
- To execute user software up to a specific point so that the user can trace through the questionable software area
- To set and remove breakpoints in user software refer to Breakpoints and Triggers

Single 16-Bit Address Trigger

The most common real-time bus capture is the single 16-bit address trigger. This trigger allows the user to capture eight changes of software flow before or after the 16-bit address trigger. This can reveal what occurred before or after this point in software. The direction (before or after) is determined by the capture options.

Only one hardware comparator is needed to set up a single 16-bit address trigger associated with the A only trigger condition. However, a second 16-bit address trigger can be set up with the A OR B trigger condition. If either trigger address A or B is met, a capture will be performed.

General practices for using a single 16-bit address trigger include:

- To resolve where user software went following the trigger point
- To figure out how user software reached the trigger point
- To verify that user software reaches a trigger point

Scenario 1:

For example, the user had a software routine that was performed periodically. However, during every other execution of this software routine, the result from this routine would be delayed. To determine why this unexpected delay is being introduced, the user would first perform a single 16-bit address trigger (A only) placed on the first instruction of the questionable routine. Using a begin capture option, the user can verify the actual versus expected software execution path. Depending on the number of changes-of-flow that occur from the beginning of the software routine until the unexpected delay, the trigger point might need to be repositioned to a point later in the routine.

Using this approach, the user might determine that an unexpected interrupt service routine (ISR) is occurring during every other instance of this routine, which introduces the unexpected delay. From this result, the user could take corrective measures to ensure that interrupts are disabled during a time-critical routine. Further trigger captures could be performed to determine what causes the interrupt to occur.

Scenario 2:

A different type of single 16-bit address trigger can be applied. For example, the user knew that the software was executing an ISR, but at an unexpected time. The user could set up an end, single 16-bit address trigger (A only), on the first instruction in the ISR. From the resulting trace capture, the user could then determine where in the software the ISR was occurring. From this data, the user might find that the software was not correctly clearing an interrupt flag.

To set and remove single 16-bit address triggers in user software refer to Breakpoints and Triggers. To configure the trigger condition and capture options for this trigger refer to Trigger Conditions and Capture Options*.*

Dual 16-Bit Address Triggers

These more elaborate real-time bus captures involve 16-bit address comparators. These also know as conditional triggers. These triggers allow the user to capture eight software changes-of-flow before or after a condition involving two 16-bit address comparisons. The direction (before or after) is determined by the capture options.

In addition to setting two trigger addresses, the user selects one of the following trigger conditions:

- A OR B
- Inside range A to B
- Outside range A to B
- A THEN B

If the trigger condition associated with trigger address A and B is met, a capture will be performed.

General uses of a dual 16-bit address trigger include:

- To observe what caused user software to enter or exit a subroutine/memory area
- To know what comes before or follows in user software at two reference points
- To verify sequential software execution between two points

Scenario 3:

For example, the user suspects code runaway is occurring in the application software. The problem is noticed when a static calibration data table in memory was being corrupted. Not knowing what part in the software was changing the values in the calibration data table, the user can set a dual 16-bit address trigger to catch the responsible instruction. Because the user does not know where in the calibration data table the corruption is first occurring, the trigger can be made within an address range.

The Inside Range A to B trigger condition can be used for this scenario because it will capture data the moment any location within address range A to B is written. Comparator A would be placed at the beginning of the calibration data table. Comparator B would be placed at the end of the calibration data table. To locate the instruction that is causing the corruption in the calibration data table, the user would configure the capture option to perform an end trigger. This setting would allow the user to see the path that led to the first memory corruption in the calibration data table.

To set and remove dual 16-bit address triggers in memory, refer to Breakpoints and Triggers. To configure the trigger condition and capture options for this trigger, refer to Trigger Conditions and Capture Options.

Single 16-Bit Address plus 8-Bit Data Triggers

A data-specific, real-time bus capture (also know as a full mode trigger) is the single 16-bit address plus 8-bit data trigger. This trigger allows the user to capture eight software changes-of-flow before or after a specified data value is read or modified at a specific memory address. The direction (before or after) is determined by the capture options. This trigger uses two comparators. The first is set to a single 16-bit address and the second is set to an 8-bit data value being observed. After setting the address and before setting the data value, the user sets one of the following trigger conditions:

- A AND B (data)
- A AND NOT B (data)

If the data value (comparator B) at Address A is read or modified, a capture will be performed.

General uses of a single 16-bit address plus 8-bit data trigger include:

- To capture the instruction that unexpectedly performs a read or write access at a specified memory location
- To observe what leads to or follows in software after the initialization of a register address
- To halt user software when a memory location changes to/from a specific data value

Scenario 4:

For example, the user wants to verify the execution path in the software after enabling a peripheral. (For this scenario, writing 0x80 to the corresponding peripheral control register enables the peripheral.) The user can use the A AND B (data) trigger in the begin trigger mode to capture the software changes-of-flow following the write of 0x80 to the peripheral control register.

Comparator A would be set to the 16-bit address associated with the peripheral control register and comparator B would be set to the expected data value (0x80) to be written in the peripheral control register. When the write of 0x80 occurs at the peripheral control register, the on-chip ICE would begin recording the software changes of flow. This would provide the user with the data necessary to evaluate the software.

Scenario 5:

Another scenario could be that a user suspects that a peripheral control register is being overwritten unexpectedly. The user's software maintains a 0x55 value in a peripheral control register. However, the software condition statement that checks for the 0x55 value in the peripheral control register unexpectedly fails. The value appears to have changed because the condition statement is not being satisfied as expected. To determine the cause of the unexpected data value modification, the user can set up an A AND NOT B (data) trigger, in end trigger mode.

The end trigger mode will allow the user to record the instructions performed prior to the change-in-value at the peripheral control register. Comparator A will be set to the 16-bit peripheral control register address in question. Comparator B will be set to the 8-bit data value (0x55), which the user expects the register to maintain. Since this trigger condition is qualified by a write of any value other than 0x55 (NOT B), the first instance that changes the peripheral register will freeze the contents of the previous eight changes-of-flow that are responsible for the erroneous write to the peripheral control register.

NOTE

For a single 16-bit address plus 8-bit data trigger, make sure to specify in the trigger settings whether you are qualifying on a read or a write of a specified data value match (or mismatch) at the specified address location.

To set and remove a single 16-bit address plus 8-bit data trigger in user software refer to Breakpoints and Triggers. To configure the trigger condition and capture options for this trigger refer to Trigger Conditions and Capture Options.

Single 16-Bit Address Event Triggers

The single 16-bit address event trigger records software change-of-flow addresses. Unlike the other trigger methods, this trigger allows the user to capture eight event data values read and/or written at a specified 16-bit memory location. Also, this trigger method does not care about the recording direction option (begin/end trigger modes) because it always performs a begin trigger.

This trigger can be set up to use one or two comparators. One hardware comparator is needed to set up a single 16-bit address event trigger associated with the Event Only B trigger condition. For this condition, if any read and/or write access is performed at the comparator B address, the event data value will be captured.

However, a second 16-bit address event trigger can be set up with the A then Event Only B trigger condition. This condition is identical to the Event Only B trigger condition with an additional comparator A that must be matched to a 16-bit address prior to recording event data values associated with a read and/or write access at comparator B.

General uses of a single 16-bit address event trigger include:

- To log the data values associated with a read and/or write event at a specified memory location
- To verify if a data value at a specified memory location is ever modified and to what value
- To watch the changes to a data value at a specified register location

Scenario 6:

For example, the user wants to verify an 8-bit data result stored in memory. If there is any doubt about the accuracy in the software algorithm that calculates the data result, the user can set up an Event Only B trigger. For this example, comparator B would be set to the 16-bit address associated with 8-bit data result that the user wants to log. This trigger would record the first eight values written at the memory location where the result resides. The user can then verify that his expected data result pattern is satisfied.

Scenario 7:

A variation to *Scenario 6* would be if the user wanted to capture accesses to a data result register after the software has completed polling for a status ready flag. For this example, the user can use the A then Event Only B trigger condition, which postpones the recording of data accesses until the software line that indicates transmission ready status. Without this trigger condition, a simple Event Only B trigger could not differentiate data accesses between the idle and ready status periods.

For this condition, comparator A would be set to the 16-bit address that contains the instruction that validates the ready status flag. Comparator B would be set to the 16-bit address that is associated with the data result register. From this trigger, the user can verify that correct data is being transmitted after shifting from the idle to ready state.

Conclusion

The HCS08 on-chip ICE system enables you to debug application software faster and with more ease. The various enhancements to previous debugging techniques allow you to set up predetermined triggers and capture options that execute a real-time bus capture within silicon and analyze the results at your own convenience.

With the on-chip ICE support in the Metrowerks' CodeWarrior True-time Simulator & Real-Time Debugger and the instructions explained in this application note, you can troubleshoot debug scenarios without expensive emulators and bus-state analyzers.

Overall, the HCS08 on-chip ICE system improves on the limitations of previous software debugging methods and is complemented by the new on-chip ICE support user interface in the CodeWarrior toolset.

NOTE

With the exception of mask set errata documents, if any other Freescale Semiconductor document contains information that conflicts with the information in the device data sheet, the data sheet should be considered to have the most current and correct data.

This page is intentionally blank.

This page is intentionally blank.