Chipsmall Limited consists of a professional team with an average of over 10 year of expertise in the distribution of electronic components. Based in Hongkong, we have already established firm and mutual-benefit business relationships with customers from,Europe,America and south Asia,supplying obsolete and hard-to-find components to meet their specific needs.

With the principle of “Quality Parts,Customers Priority,Honest Operation,and Considerate Service”,our business mainly focus on the distribution of electronic components. Line cards we deal with include Microchip,ALPS,ROHM,Xilinx,Pulse,ON,Everlight and Freescale. Main products comprise IC,Modules,Potentiometer,IC Socket,Relay,Connector.Our parts cover such applications as commercial,industrial, and automotives areas.

We are looking forward to setting up business relationship with you and hope to provide you with the best service and solution. Let us make a better world for our industry!

Contact us

Tel: +86-755-8981 8866 Fax: +86-755-8427 6832
Email & Skype: info@chipsmall.com Web: www.chipsmall.com
Address: A1208, Overseas Decoration Building, #122 Zhenhua RD., Futian, Shenzhen, China

Emulation Extension Pak (EEP) and Emulation Header User's Guide

ISBN: 978-1-63277-835-2

QUALITY MANAGEMENT SYSTEM CERTIFIED BY DNV

═ ISO/TS 16949 ═

Microchip received ISO/TS-16949:2009 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

Table of Contents

Chapter 1. EEP and Emulation Header Overview

NOTICE TO CUSTOMERS

All documentation becomes dated, and this manual is no exception. Microchip tools and documentation are constantly evolving to meet customer needs, so some actual dialogs and/or tool descriptions may differ from those in this document. Please refer to our web site (www.microchip.com) to obtain the latest documentation available.

Documents are identified with a "DS" number. This number is located on the bottom of each page, in front of the page number. The numbering convention for the DS number is "DSXXXXXA", where "XXXXX" is the document number and "A" is the revision level of the document.

For the most up-to-date information on development tools, see the MPLAB X IDE online Help (Help menu).

This chapter contains the following topics:

1.1 WHAT IS AN EMULATION EXTENSION PAK?

An Emulation Extension Pak (EEP) contains an emulation header; gold, single, in-line pins; a trace cable, and a trace adapter board. An EEP is what you purchase when you want an emulation header.

1.2 WHAT IS AN EMULATION HEADER?

An emulation header is a circuit board that allows a debug tool to debug code for a specific device. A special version of the device (-ME2) with on-board emulation circuitry is located on the header. Connectors on the side of the header allow it to connect directly to, or through, an adapter to the emulator. Connectors on the bottom of the header allow it to connect directly to, or through, a transition socket to a target board.

1.3 WHY WOULD I WANT TO USE AN EMULATION HEADER?

Although some devices have on-board debug circuitry to allow you to debug your code, you often lose device resources to debugging, i.e., debugging requires the use of two I/O lines, plus V_{DD}, V_{SS} and V_{PP}, to communicate with the device. Using an emulation header can free up these resources for your application, and give you new and powerful debugging features not found on debug headers.

For details on emulation header features, see Chapter 2. “Emulation Header Features”.

For details on debug header features, see the user’s guide or online help file for supported debug tools (listed in Section 1.5 “Support Information”). Also see the *Processor Extension Pak and Debug Header Specification* (DS51292). Find all documentation on the Microchip website (http://www.microchip.com).

For a comparison of emulation versus debug features, see Section 1.4 “Compare Emulation Header, Debug Header and Device Features (Future)”.

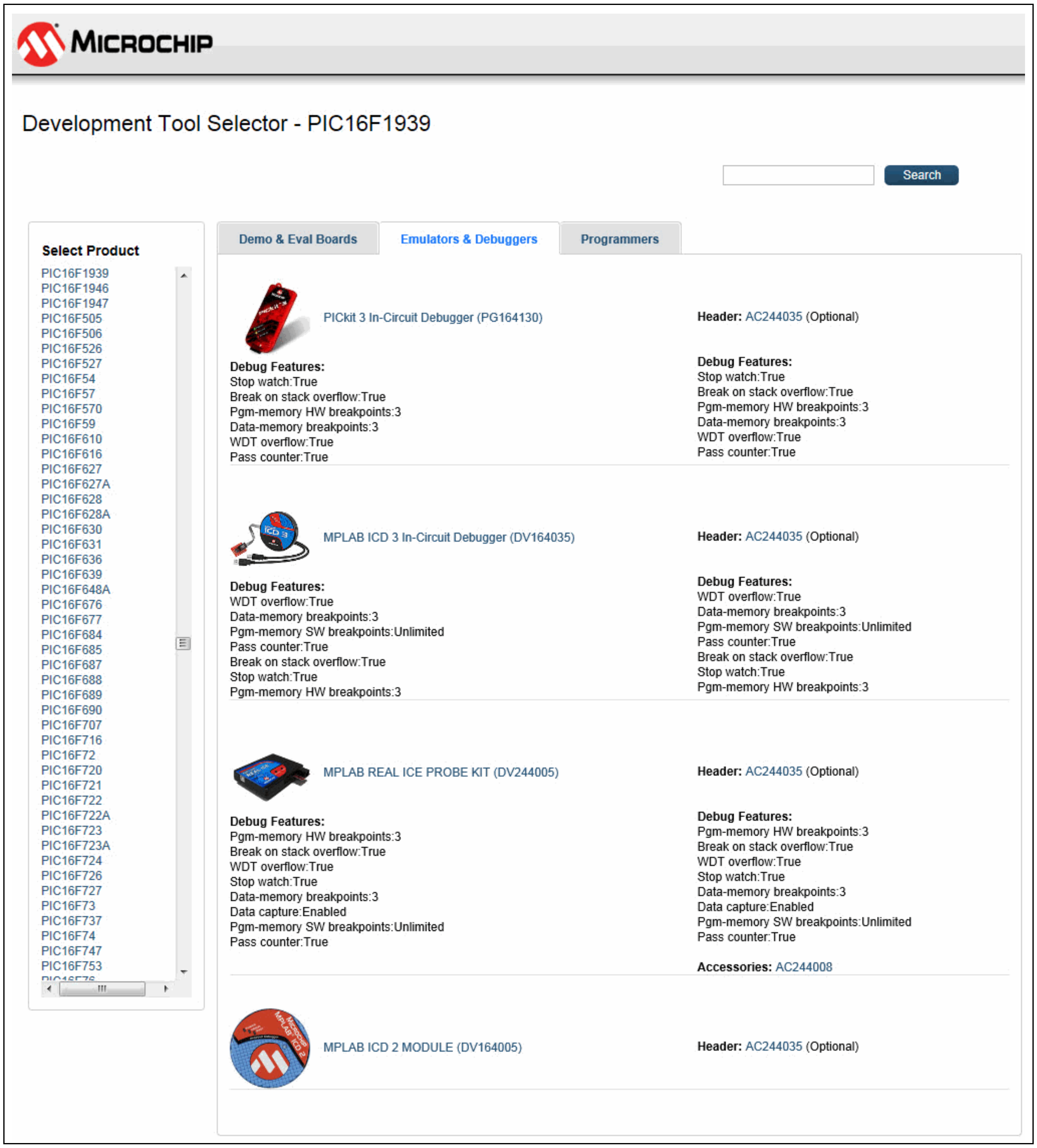
1.4 COMPARE EMULATION HEADER, DEBUG HEADER AND DEVICE FEATURES (FUTURE)

Use the Development Tool Selector (DTS) to compare the extra debug features of an emulation header to a supported device (with on-board debug circuitry) and supported debug header.

To find features by device:

1. In a web browser, go to: http://www.microchip.com/dtsapp/
2. Select your device from the “Select Product” list. Or type and search for the name of your device in the "Search" box and it will appear at the top of the “Select Product” list, where you can select it.
3. Click on the tab “Emulators & Debuggers” to see debug features.

FIGURE 1-1: DTS DEVICE INFORMATION



Development Tool Selector - PIC16F1939

Select Product

PIC16F1939
PIC16F1946
PIC16F1947
PIC16F505
PIC16F506
PIC16F526
PIC16F527
PIC16F54
PIC16F57
PIC16F570
PIC16F59
PIC16F610
PIC16F616
PIC16F627
PIC16F627A
PIC16F628
PIC16F628A
PIC16F630
PIC16F631
PIC16F636
PIC16F639
PIC16F648A
PIC16F676
PIC16F677
PIC16F684
PIC16F685
PIC16F687
PIC16F688
PIC16F689
PIC16F690
PIC16F707
PIC16F716
PIC16F72
PIC16F720
PIC16F721
PIC16F722
PIC16F722A
PIC16F723
PIC16F723A
PIC16F724
PIC16F726
PIC16F727
PIC16F73
PIC16F737
PIC16F74
PIC16F747
PIC16F753

PICkit 3 In-Circuit Debugger (PG164130)

Header: AC244035 (Optional)

Debug Features:
Stop watch:True
Break on stack overflow:True
Pgm-memory HW breakpoints:3
Data-memory breakpoints:3
WDT overflow:True
Pass counter:True

Debug Features:
Stop watch:True
Break on stack overflow:True
Pgm-memory HW breakpoints:3
Data-memory breakpoints:3
WDT overflow:True
Pass counter:True

MPLAB ICD 3 In-Circuit Debugger (DV164035)

Header: AC244035 (Optional)

Debug Features:
WDT overflow:True
Data-memory breakpoints:3
Pgm-memory SW breakpoints:Unlimited
Pass counter:True
Break on stack overflow:True
Stop watch:True
Pgm-memory HW breakpoints:3

Debug Features:
WDT overflow:True
Data-memory breakpoints:3
Pgm-memory SW breakpoints:Unlimited
Pass counter:True
Break on stack overflow:True
Stop watch:True
Pgm-memory HW breakpoints:3

MPLAB REAL ICE PROBE KIT (DV244005)

Header: AC244035 (Optional)

Debug Features:
Pgm-memory HW breakpoints:3
Break on stack overflow:True
WDT overflow:True
Stop watch:True
Data-memory breakpoints:3
Data capture:Enabled
Pgm-memory SW breakpoints:Unlimited
Pass counter:True

Debug Features:
Pgm-memory HW breakpoints:3
Break on stack overflow:True
WDT overflow:True
Stop watch:True
Data-memory breakpoints:3
Data capture:Enabled
Pgm-memory SW breakpoints:Unlimited
Pass counter:True

Accessories: AC244008

MPLAB ICD 2 MODULE (DV164005)

Header: AC244035 (Optional)

1.5 SUPPORT INFORMATION

Emulation headers require specific MPLAB X IDE versions and debug tools to operate. Acquire these before purchasing an emulation header in an emulation extension pak (EEP). Available EEPs are listed in Chapter 3. “Emulation Header List”.

To proceed with setting up emulation header hardware, see Section 1.6 “Emulation Header Hardware Setup”.

Contact Customer Support for issues with emulation headers.

1.5.1 Software Support

Emulation headers are supported on MPLAB X IDE v1.90 and greater.

1.5.2 Tool Support

Emulation headers are supported on the following tools:

- PICkit™ 3 in-circuit debugger
- MPLAB® ICD 3 in-circuit debugger
- MPLAB® REAL ICE® in-circuit emulator

Note: Not all features are supported on all tools.

1.5.3 Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Field Application Engineer (FAE)
- Technical Support

Technical support is available through the web site at: http://support.microchip.com

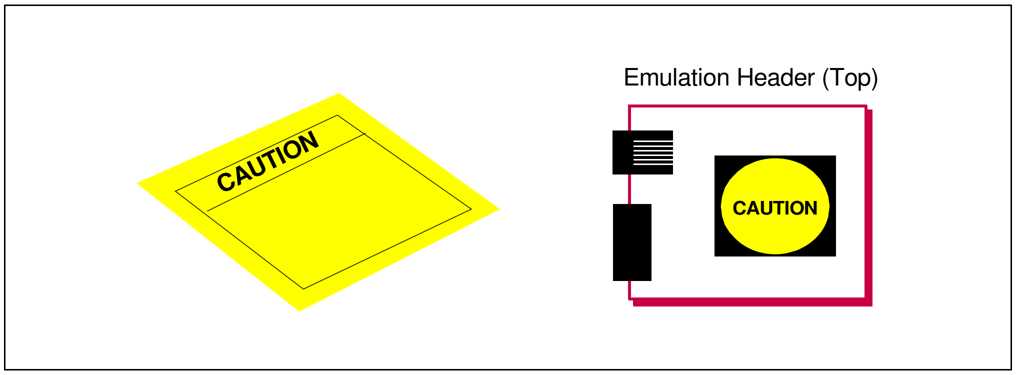
Documentation errors or comments may be sent to: docerrors@microchip.com.

1.6 EMULATION HEADER HARDWARE SETUP

To set up your header, follow these instructions before doing anything else:

1. Check the header box for any paper inserts that specify special operating instructions and the emulation header for any stickers (Figure 1-2).

FIGURE 1-2: SPECIAL HEADER INSTRUCTIONS



2. Set any jumpers or switches on the header to determine device functionality, or selection, as specified for that header. See the section "Emulation Header List" for information on how to set up individual headers.
3. Connect the header to your desired debug tool by consulting the tool documentation for connection options. Example connections are shown in Figure 1-3, Figure 1-4, and Figure 1-5.

FIGURE 1-3: PICkit™ 3 IN-CIRCUIT DEBUGGER CONNECTIONS

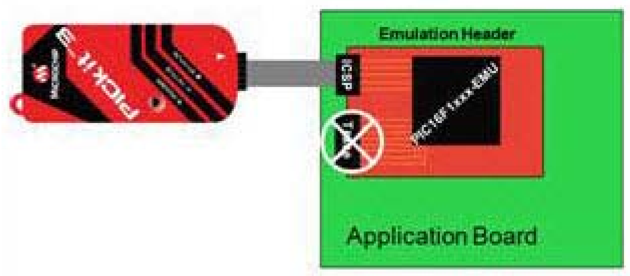


FIGURE 1-4: MPLAB® ICD 3 IN-CIRCUIT DEBUGGER CONNECTIONS

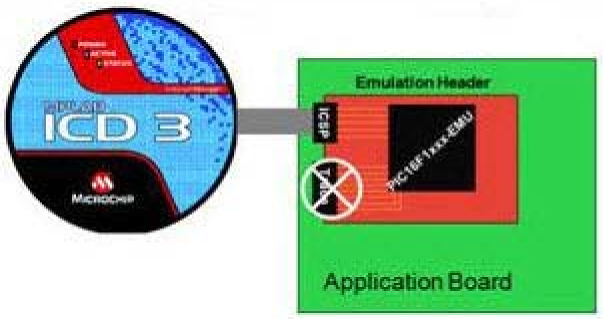
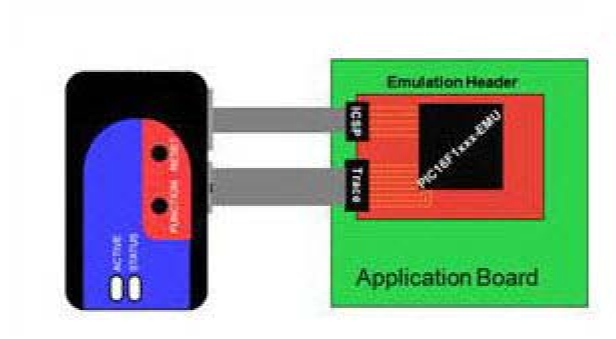


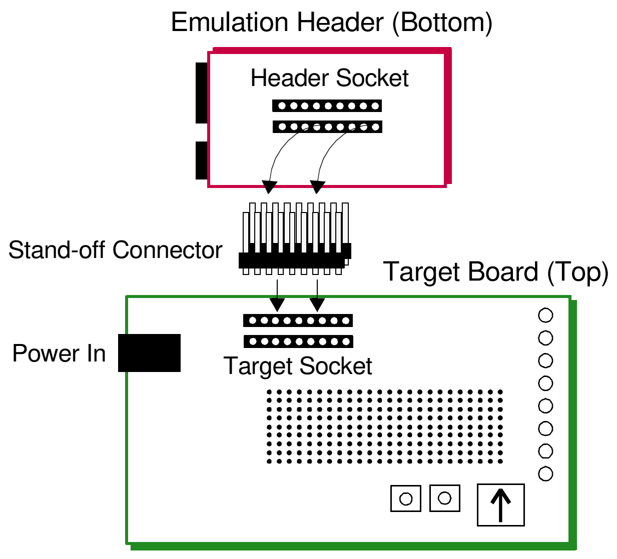
FIGURE 1-5: MPLAB® REAL ICE™ IN-CIRCUIT EMULATOR CONNECTIONS



4. Connect the header to the target board. On the bottom of the header is a socket that is used to connect to the target board. The header can be connected to the target board as follows:
   a) PDIP header socket to PDIP target socket with a stand-off (male-to-male) connector or single in-line pins. An example is shown in Figure 1-6.
   b) Header socket to plug on the target board
   c) Header socket to target socket with a transition socket (see the *Transition Socket Specification*, DS51194)

   The header socket will have the same pin count as your selected device. The -ME2 device on the top of the header usually has a larger pin count because it has additional pins that are dedicated to debug.

FIGURE 1-6: CONNECT HEADER TO TARGET



5. If using a debug tool that can power the target, power that tool now.
6. Power the target, if needed.

1.7 EMULATION HEADER SETUP FOR MPLAB X IDE

Emulation header functionality is supported on MPLAB X IDE 1.90 or greater, but **not** on MPLAB IDE v8. Please use debug headers, if you are still using MPLAB IDE v8.

You need to do the following to use an emulation header on MPLAB X IDE:

1. Set up the emulation header as specified in Section 1.6 "Emulation Header Hardware Setup".
2. Begin creating a project for a device supported by your emulation header using the Projects wizard (*File>New Project*). See MPLAB X IDE documentation for more on projects.
3. In one step of the wizard you will have an opportunity to specify the emulation header product number (AC######).
4. In another step you will specify the hardware (debug) tool to which the emulation header is attached.
5. Once the wizard is complete, write code for your project.
6. Select *Debug>Debug Project* to run and debug your code.

Note: An emulation header can only be used to debug (Debug menu), not to program (Run menu). See "Section 1.8.1 "Programming Notes"".

1.8 ADDITIONAL INFORMATION

The following additional information is useful when using an emulation header from an Emulation Extension Pak.

1.8.1 Programming Notes

The emulation header is designed to be used with the in-circuit emulator in debugger mode, *Debug>Debug Project,* (not in programmer mode, *Run>Run Project*), in MPLAB X IDE. Any programming of the special -ME2 device on the header is for debug purposes.

To program production (non-special) devices with your debug tool, use the *Universal Programming Module* (AC162049) or design a modular interface connector on the target. See the appropriate specification for connections. For the most up-to-date device programming specifications, see the Microchip website at http://www.microchip.com.

Also, production devices can be programmed with the following tools:

- MPLAB PM3 device programmer
- PICkit 3 development programmer
- MPLAB ICD 3 in-circuit debugger (select as a programmer)
- MPLAB REAL ICE in-circuit emulator (select as a programmer)

1.8.2 Calibration Bits

The calibration bits for the band gap and internal oscillator are always preserved to their factory settings.

1.8.3 Performance Issues

See the MPLAB X IDE help file regarding your debug tool for information on specific device limitations that could affect performance.

NOTES:

Chapter 2. Emulation Header Features

2.1 INTRODUCTION

Emulation header features depend on the debug tool used. The table below shows a list of all available emulation header features, and the features that are supported on each tool.

TABLE 2-1: EMULATION FEATURE SUPPORT BY HARDWARE TOOL

Features	RI	ICD3	PK3
Section 2.3 "Runtime Watches" - see Note	✓	✗	✗
Section 2.4 "Real Time Hardware Instruction Trace" - see Note	✓	✗	✗
Section 2.5 "Hardware Address/Data Breakpoints" - see Note Section 2.5.2 "Range Breakpoints" Section 2.5.3 "Data Value Comparison" Section 2.5.4 "Data Value Mask" Section 2.5.6 "Trigger Out Operation" Section 2.5.7 "Interrupt Context Detection"	✓	✓	✓
Section 2.6 "Enhanced Event Breakpoints" - see Note Section 2.6.1 "Execution Out-of-Bounds Detection" Section 2.6.2 "Break on Trigger In/Emit Trigger Out"	✓	✓	✓
Section 2.7 "Event Combiners"	✓	✓	✓
Section 2.8 "Stopwatch Cycle Counter"	✓	✓	✓
Section 2.9 "Trigger In/Out"	✓	✓	✓
Section 2.10 "View Hardware Stack On Halt"	✓	✓	✓
Section 2.11 "Previous Program Counter"	✓	✓	✓
Section 2.12 "Background Debug"	✓	✓	✓

Legend:
RI = MPLAB® REAL ICE™ In-Circuit Emulation
ICD3 = MPLAB® ICD 3 In-Circuit Debugger
PK3 = PICkit™ 3 In-Circuit Debugger

Note: See also Section 2.2 "Breakpoint, Runtime Watch, and Trace Resources".

2.2 BREAKPOINT, RUNTIME WATCH, AND TRACE RESOURCES

Emulation headers have 32 data capture resources, total, that may be used for breakpoints and runtime watches. For example, if you use a data capture resource for a breakpoint, you will have one less data capture resource for other breakpoints or runtime watches.

Runtime watches and trace are mutually exclusive. Data streaming from the emulation header may be used for either runtime watches or trace. For example, if you want to use trace, you cannot use runtime watches.

Selecting trace in the Project Properties takes precedence over other selections, such as in the Watches window or in any plug-ins.

2.3 RUNTIME WATCHES

In the MPLAB X IDE Watches window (*Window>Debugging>Watches*), you can create a watch for a symbol where the value changes at runtime.

Before you add a runtime watch to the Watches window, you need to set up the clock. Perform the following steps to set up the clock:

1. Right click on the project name and select "Properties".
2. Click on the debug tool name (e.g., Real ICE) and select option category "Clock".
3. Set the runtime instruction speed.

To add a global symbol or SFR as a runtime watch, do one of the following:

- Right click in the Watches window and select "New Runtime Watch" or select *Debug>New Runtime Watch*. Click the selection buttons to see either Global Symbols or SFRs. Click on a name from the list and then click **OK**.
- Select the symbol or SFR name in the Editor window and then select "New Runtime Watch" from the right click menu. The name is populated in the Watches window. Click **OK**.

Debug Run and watch the values change as the program executes. For more information on the Watches window, see MPLAB X IDE documentation.

2.3.1 Symbol/SFR Size

For all devices except PIC32 MCUs, symbols used in a runtime watch must be sized to match the device memory. That is, you need 8-bit symbols when using an 8-bit device.

2.3.2 SFR Caveats

Unless an SFR is explicitly read or written by code, no value change will be visible in the Watches window. As the runtime watch uses the trace mechanism, code must read or cause a change before a trace packet is output. Therefore any SFR value changes, without code explicitly reading it or causing the SFR value change, cannot be directly monitored as a Runtime Watch.

For example, setting up a Runtime Watch on the Analog-to-Digital Converter (ADC) conversion result SFRs `ADRESH` and `ADRESL` will not show any runtime changes, even when the input voltage to the ADC is changing. A workaround is code that copies the `ADRESH` and `ADRESL` register values to RAM variables (e.g., `ADRESH_copy` and `ADRESL_copy`') after an ADC conversion is complete. These variables, added to the Watches window as Runtime Watches, will then reflect the changes in ADC values.

Another example would be monitoring an input pin or an entire port that is configured as inputs. External changes on the input pin(s) will not trigger a trace watch packet; and, therefore, no runtime updates to the port will be displayed. The Runtime Watch display will only update when code reads the input pin or input port.

2.4 REAL TIME HARDWARE INSTRUCTION TRACE

Note: This feature is only supported on the MPLAB REAL ICE in-circuit emulator.

Real Time Hardware Instruction Trace is a real-time dump of the program execution stream that can be captured and analyzed. The functionality that is provided includes the following:

- full instruction execution information, up to 32 MHz
- trace through Reset conditions
- trace buffer with optional stall

Emulation header trace is similar to PIC32 instruction trace, which is a non-intrusive hardware instruction trace. You can use trace to capture every instruction executed by the device. The trace data is output from the device (using the pins TRCLK, TRDAT[6:0], and TRSTALL) to the emulator. The emulator streams this data to a trace buffer, that acts like a rolling first-in/first-out (FIFO) buffer, on the PC.

The amount of trace data is limited only by the size of the trace buffer. This buffer can fill quickly even when set to the maximum size, so it is wise to determine exactly what you need to capture.

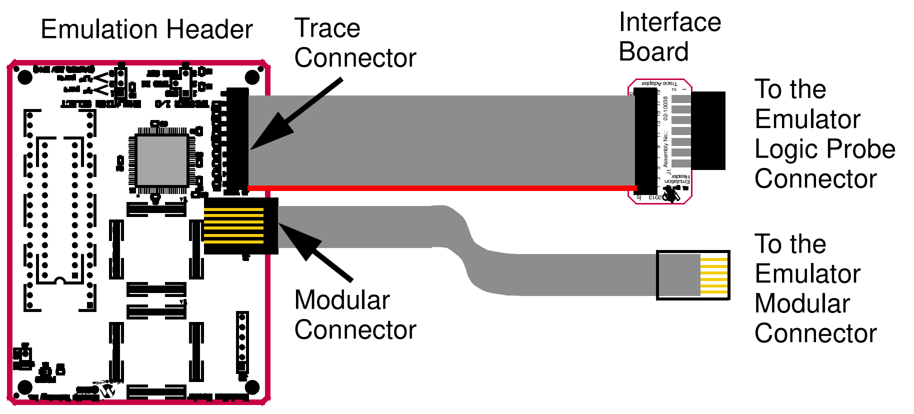
Note: Execution will NOT HALT when this external buffer is full. However, MPLAB X IDE will announce when the trace buffer has overflowed.

Trace requires hardware and software setup before trace data can be viewed in a window.

2.4.1 Set Up Trace Hardware

To use trace, you will need the ribbon cable and emulator interface board that comes with the emulation header. Connect one end to the emulation header (Figure 2-1). Connect the other end to the emulator. For more information on complete hardware setup and connections, refer to Section 1.6 "Emulation Header Hardware Setup". For details on trace connectors, see Section 2.4.4 "Trace Hardware".

FIGURE 2-1: EMULATION TRACE CABLE CONNECTED TO HEADER

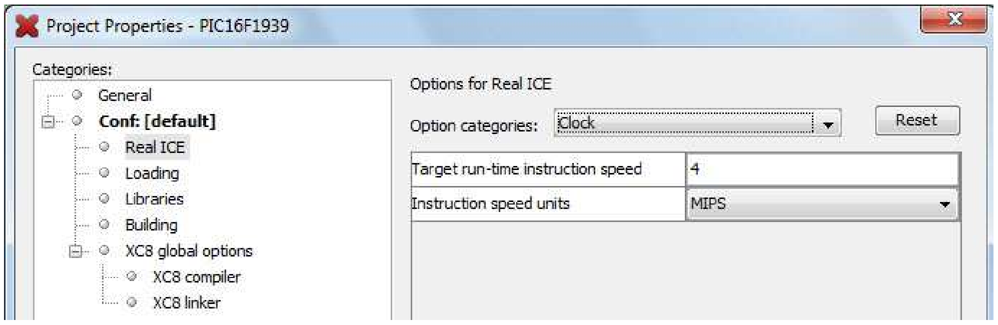

Emulation Header
Trace Connector
Interface Board
To the Emulator Logic Probe Connector
To the Emulator Modular Connector
Modular Connector

2.4.2 Set Up Trace for MPLAB X IDE

To set up MPLAB X IDE to use trace for the MPLAB REAL ICE in-circuit emulator, perform the following steps:

1. Right click on the project name and select "Properties" to open the Project Properties window.
2. Click on "Real ICE" under "Categories".
3. Under "Option categories", select "Clock". Specify the target run-time instruction speed and speed units.

FIGURE 2-2: CLOCK OPTION CATEGORY

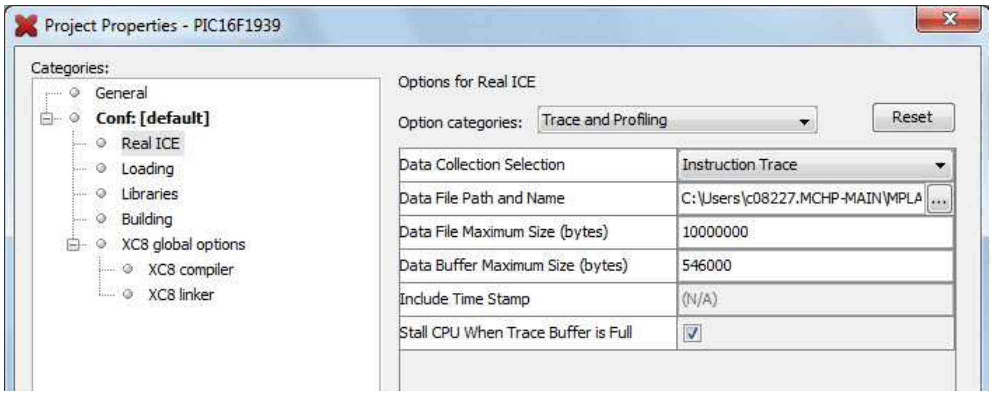


Option categories: Clock

Target run-time instruction speed	4
Instruction speed units	MIPS

4. Under "Option categories", select "Trace and Profiling".
5. Under "Data Collection Selection", choose "Instruction Trace/Profiling".
6. Set up any other trace-related options.

FIGURE 2-3: TRACE AND PROFILING OPTION CATEGORY



Option categories: Trace and Profiling

Data Collection Selection	Instruction Trace
Data File Path and Name	C:\Users\c08227.MCHP-MAIN\MPLA
Data File Maximum Size (bytes)	10000000
Data Buffer Maximum Size (bytes)	546000
Include Time Stamp	(N/A)
Stall CPU When Trace Buffer is Full	✓

7. Click **OK**.

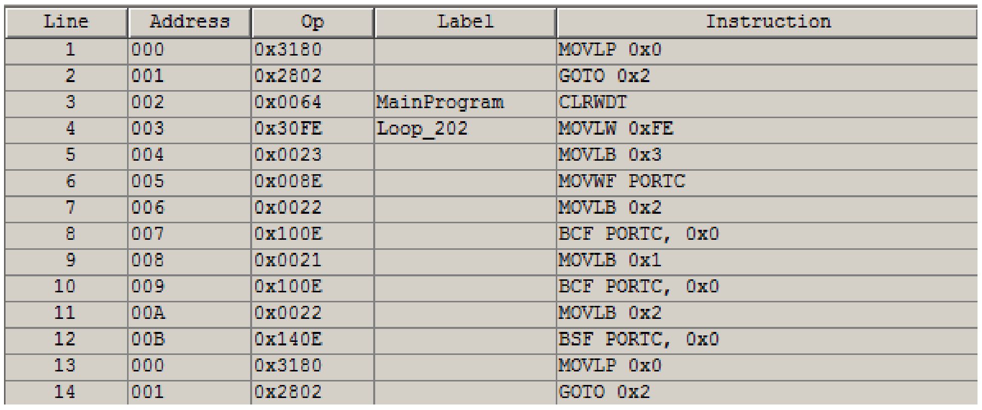
2.4.3 Using Trace

Once trace hardware and MPLAB X IDE are set up, you can begin using trace. On a Debug Run, trace will continue to fill the trace buffer with data, rolling over when the buffer is full, until a program Halt. For more on the trace buffer, see Section 2.4 "Real Time Hardware Instruction Trace".

2.4.3.1 VIEWING TRACE DATA

When trace is enabled and code is run, trace data will be collected by the emulator. Once the device is halted, trace data will be decoded and displayed in the Trace window (*Window>Debugging>Trace*).

FIGURE 2-4: TRACE WINDOW



Line	Address	Op	Label	Instruction
1	000	0x3180		MOVLP 0x0
2	001	0x2802		GOTO 0x2
3	002	0x0064	MainProgram	CLRWDT
4	003	0x30FE	Loop_202	MOVLW 0xFE
5	004	0x0023		MOVLB 0x3
6	005	0x008E		MOVWF PORTC
7	006	0x0022		MOVLB 0x2
8	007	0x100E		BCF PORTC, 0x0
9	008	0x0021		MOVLB 0x1
10	009	0x100E		BCF PORTC, 0x0
11	00A	0x0022		MOVLB 0x2
12	00B	0x140E		BSF PORTC, 0x0
13	000	0x3180		MOVLP 0x0
14	001	0x2802		GOTO 0x2

2.4.3.2 BANK SELECT AND TRACE

For PIC12F/16F1xxx devices and related headers, Trace can decode instructions resulting in a bank set. Once established, decoding with that bank continues until the next bank switch. Therefore, scrolling up in the window will cause the bank to be indeterminate, where as scrolling down sets the bank.

2.4.3.3 IMPROVING THE TRACE EXPERIENCE

Remove as many USB devices from your PC USB ports as you can. This should improve trace throughput.

2.4.4 Trace Hardware

Hardware details are shown in the following figures.

FIGURE 2-5: TRACE CONNECTOR ON EMULATION HEADER

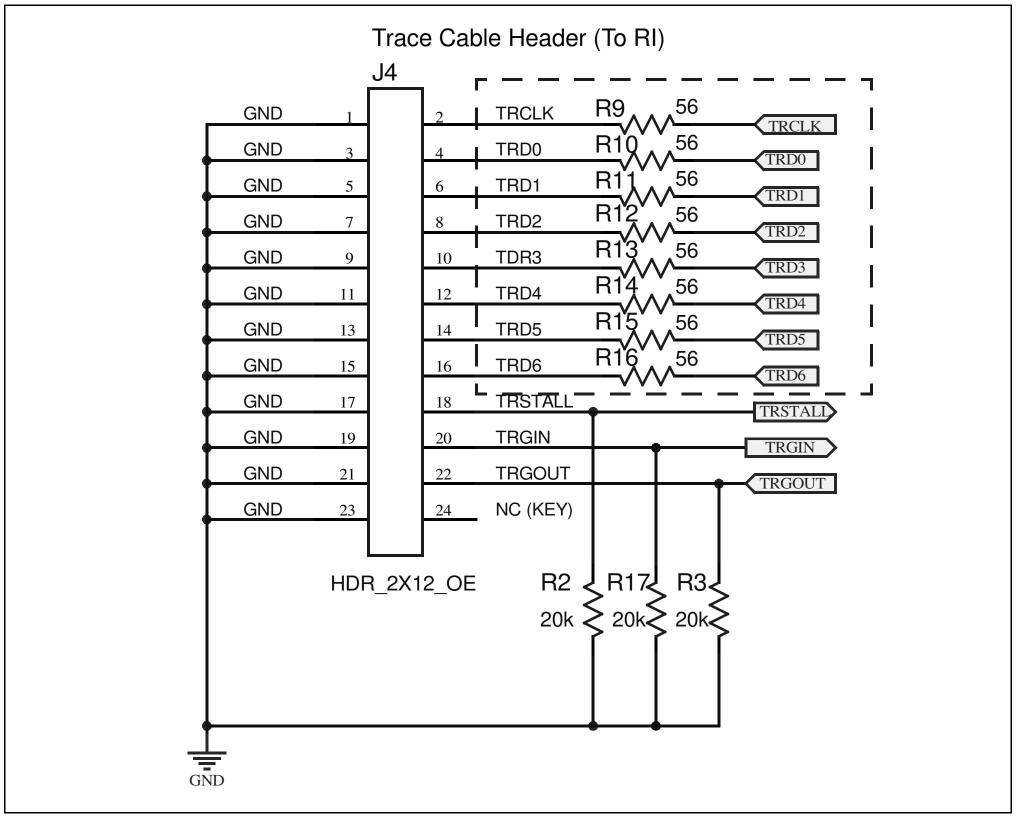
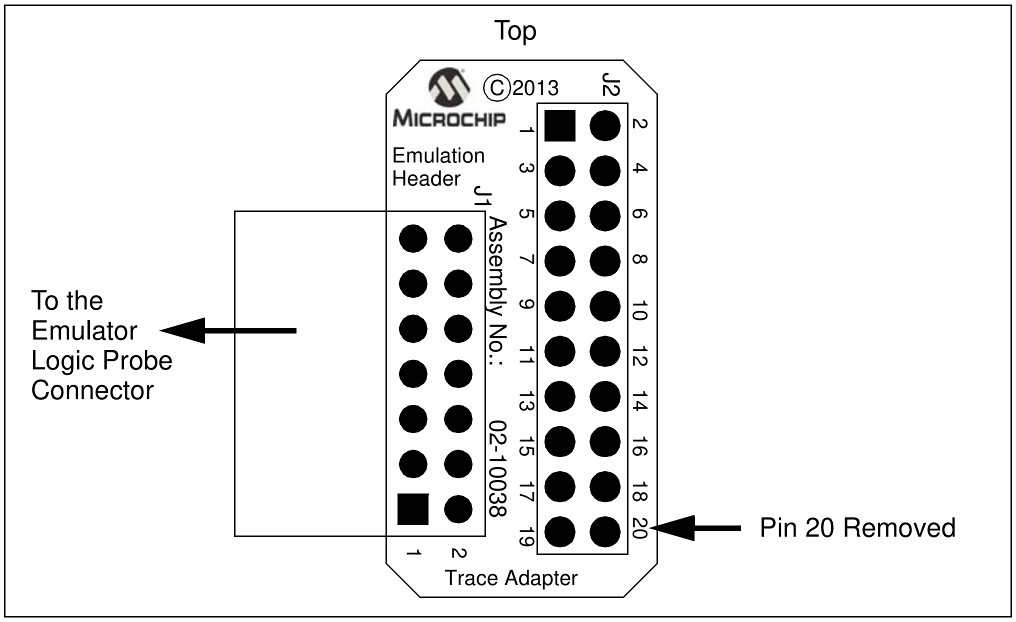


FIGURE 2-6: INTERFACE BOARD



2.5 HARDWARE ADDRESS/DATA BREAKPOINTS

Up to 32 hardware address/data breakpoints are available to use on the emulation header (for details see Section 2.2 "Breakpoint, Runtime Watch, and Trace Resources".). Software breakpoints are useful, but real hardware breakpoints are incomparable when you need unfettered control of qualifying the breakpoint/event conditions beyond the simple address matching. Consider the addition of a special interrupt contextual qualifier and these hardware breakpoints offer a high degree of configurability.

Emulation header hardware breakpoints can trigger without halting execution and could be used as trigger events to other features.

2.5.1 Breakpoint Setup

Address/Data Breakpoints may be found and set up on the New Breakpoint dialog (*Debug>New Breakpoint*) by choosing either "Address" or "Data" as the "Breakpoint Type". After the breakpoint is created, it can be edited by right clicking and selecting "Customize". For details, see the MPLAB X IDE documentation.

2.5.2 Range Breakpoints

A breakpoint may be specified to occur with a range of program or data memory. A start address, and then an end address, are specified for the breakpoint. Then the break condition is selected.

The dialog items used to set up range breakpoints are described below.

TABLE 2-2: ADDRESS/DATA BREAKPOINTS - SETTINGS

Item	Description
Enable Range Address	check to set a range breakpoint
Address (Start)	enter a starting hexadecimal memory address
Address (End)	enter an ending hexadecimal memory address
Breaks on	specifies the conditions of the break Conditions differ for address memory and data memory.

2.5.3 Data Value Comparison

Data memory breakpoints may be set up to compare a "Break on" value to another value before breaking.

The dialog items used to set up data value comparison are described below.

TABLE 2-3: DATA BREAKPOINTS - SETTINGS

Item	Description
Breaks on	specifies the conditions of the break
Value	for the "Breaks on" selection of: • Read Specific Value • Write Specific Value • Read or Write Specific Value Enter a hexadecimal value here.
Value Comparison	compare to "Value" as specified: **= Value**: Equal-to value **!= Value**: Not-equal-to value **> Value**: Greater-than value **< Value**: Less-than value

2.5.4 Data Value Mask

Data memory breakpoints may be set up to compare a masked "Break on" value to another value before breaking.

To set up a value comparison, see Section 2.5.3 "Data Value Comparison".

The dialog items used to set up a data value mask are described below.

TABLE 2-4: DATA BREAKPOINTS - SETTINGS

Item	Description
Data Value Mask	use mask when comparing to "Value" Enter a value in the range 0x00 to 0xhh, where: 0x00: no bits compared 0xhh: all bits compared

2.5.5 Pass Count Operation

Using a pass count allows you to delay breaking until after a specified count.

Event must occur Count times

Count is the number of times you will pass the breakpoint before stopping.

- 0 means execution will stop immediately
- 1 means execution will pass once then stop the next time (stop on second time)
- 10 means execution will pass 10 times and stop the next time (stop on eleventh time)

The dialog items used to set up a pass count operation are found in the "Pass count" section of the dialog and are described below.

TABLE 2-5: ADDRESS/DATA BREAKPOINTS - PASS COUNT

Item	Description
Condition	determines when the break specified under "Breaks on" occurs: **Always Break**: Always break when the "Breaks on" condition is met. **Event must occur Count times**: An event ("Breaks on" condition) must occur Count times before actually breaking.
Count*	according to the condition specified, enters the number of events

2.5.6 Trigger Out Operation

A trigger out pulse can be generated when an address or data breakpoint is reached. For more on triggers, see Section 2.9 "Trigger In/Out".

The dialog item used to set up trigger out operation is described below

TABLE 2-6: ADDRESS/DATA BREAKPOINTS - TRIGGER OUT OPTIONS

Item	Description
Trigger Options	selects when to trigger, either: • do not emit a trigger out pulse when breakpoint is hit • emits a trigger out pulse when breakpoint is hit

2.5.7 Interrupt Context Detection

An address or data breakpoint can be set based on the context of an interrupt. You can set up the breakpoint so it only breaks when it is in the interrupt section of code (ISR), only when it is in main line code, or when it is in either ISR or main code. This can assist when attempting to narrow down issues in code regions.

Address/Data Breakpoints may be found and set up on the New Breakpoint Dialog (*Debug>New Breakpoint*) by choosing either "Address" or "Data" as the "Breakpoint Type". After the breakpoint is created, it may be edited by right clicking and selecting "Customize".

The dialog item used to set up interrupt context is described below.

TABLE 2-7: ADDRESS/DATA BREAKPOINTS - INTERRUPT CONTEXT

Item	Description
Interrupt Context	Interrupt Context qualifier for address/data breakpoints select from: • Always break (break in both ISR and main code) • Break in main line (non-interrupt) context only - break in main code only • Break in interrupt context only - break in ISR code only

2.6 ENHANCED EVENT BREAKPOINTS

For a definition of event breakpoints, see the MPLAB X IDE Help, "New Breakpoint Dialog". Additional events for emulation headers are:

- Break on MCLR reset
- Break on execution out of bounds
- Break on trigger in signal

When creating a new breakpoint or customizing an existing breakpoint using an emulation header, additional actions are available for each event breakpoint:

Action	Description
Break	breaks (halts) execution per option specified
Trigger out	emits a trigger out pulse per option specified
Break and trigger out	breaks (halts) execution AND emits a trigger out pulse per option specified

Event breakpoints may be found and set up on the New Breakpoint Dialog (*Debug>New Breakpoint*) by choosing "Event" as the "Breakpoint Type". After the breakpoint is created, it may be edited by right clicking and selecting "Customize".

2.6.1 Execution Out-of-Bounds Detection

An emulation header can be used to detect out-of-bounds execution of code. Out-of-bounds code execution is detected by an event breakpoint that watches for Program Counter (PC) values that exceed the available program memory of the emulated MCU. The out-of-bounds code execution condition is typically caused by a computed GOTO or CALL that erroneously computes the index, or by loading PCLATH with an incorrect value. Once code is halted due to the execution out-of-bounds event breakpoint the 'Previous PC' functionality can be used to identify the offending instruction.

The Out-of-bounds break option may be found and set up on the New Breakpoint Dialog (*Debug>New Breakpoint*) by choosing "Event" as the "Breakpoint Type" and checking "Break on execution out of bounds". After the breakpoint is created, it may be edited by right clicking and selecting "Customize".

2.6.2 Break on Trigger In/Emit Trigger Out

An emulation header can be set up to break on a trigger in. For details on this type of trigger, see Section 2.9.2 "Trigger In Operation".

Also, the emulation header may be set to trigger out, or break and trigger out, when an enhanced even breakpoint is hit. For details on this type of trigger, see Section 2.9.3 "Trigger Out Operation".

See also: Section 2.9.1 "Trigger In/Out Hardware".

2.7 EVENT COMBINERS

An event combiner monitors multiple event inputs (currently breakpoints only) and can generate a halt or a trigger out that is based on combinations and sequences of those inputs.

Emulation headers have four (4) Event Combiners, and each combines eight (8) combinational or sequential events into a single event trigger. Individual breakpoints may be grouped into sequences, logical ‘AND’ lists, or a nested combination of these for more complex control. The Event Combiners were modeled after the legacy MPLAB® ICE 2000 complex triggers.

Set up the following complex breakpoints through selections in the Breakpoint window (*Window>Debugging>Breakpoints*).

2.7.1 Complex Breakpoint Sequence

A breakpoint sequence is a list of breakpoints that execute but do not halt until the last breakpoint is executed. Sequenced breakpoints can be useful when there are more than one execution path leading to a certain instruction, and you only want to exercise one specific path.

To create a Breakpoint Sequence:

1. Right click on an existing breakpoint or shift click to select a group of existing breakpoints and right click on the group.
2. From the pop-up menu, go to “Complex Breakpoint” and select “Add to New Sequence”.
3. Enter a name for your sequence in the dialog box, and click **OK**.
4. The breakpoint(s) will appear under the new sequence.

To add Existing Breakpoints to a Sequence:

1. Right click on an existing breakpoint or shift click to select a group of existing breakpoints and right click on the group.
2. From the pop-up menu, go to “Complex Breakpoint” and select “Move to *Name*”, where *Name* is the name of the sequence.

To add a New Breakpoint to a Sequence:

Right click on the sequence and select “New Breakpoint”.

For more on setting up a new breakpoint, see Section 2.5 “Hardware Address/Data Breakpoints” and Section 2.6 “Enhanced Event Breakpoints”.

To select the Sequence Order:

1. Expand on a sequence to see all items.
2. Right click on an item and select *Complex Breakpoints>Move Up* or *Complex Breakpoints>Move Down*. Sequence execution of breakpoints is bottom-up; the last breakpoint in the sequence occurs first.

To remove Breakpoints from a Sequence:

1. Right click on an existing breakpoint or shift click to select a group of existing breakpoints and right click on the group.
2. From the pop-up menu, go to “Complex Breakpoint” and select “Remove from *Name*”, where *Name* is the name of the sequence.

2.7.2 Complex Breakpoint Latched-And

In addition to breakpoint sequences, a Latched-And (hardware AND) is available to AND a list of breakpoints. ANDed breakpoints can be useful when a variable is modified in more than one location and you need to break only when that variable is modified in one particular location.

To create a Breakpoint Latch-And:

1. Right click on an existing breakpoint or shift click to select a group of existing breakpoints and right click on the group.
2. From the pop-up menu, go to “Complex Breakpoint” and select “Add a New Latched-And”.
3. Enter a name for your Latched-And in the dialog box and click **OK**.
4. The breakpoint(s) will appear under the new Latched-And.

To add Existing Breakpoints to a Latch-And:

1. Right click on an existing breakpoint or shift click to select a group of existing breakpoints and right click on the group.
2. From the pop-up menu, go to “Complex Breakpoint” and select “Move to *Name*”, where *Name* is the name of the Latched-And.

To add a New Breakpoint to a Latch-And:

Right click on the Latched-And and select “New Breakpoint”.

To remove Breakpoints from a Latch-And:

1. Right click on an existing breakpoint or shift click to select a group of existing breakpoints and right click on the group.
2. From the pop-up menu, go to “Complex Breakpoint” and select “Remove from *Name*”, where *Name* is the name of the Latched-And.

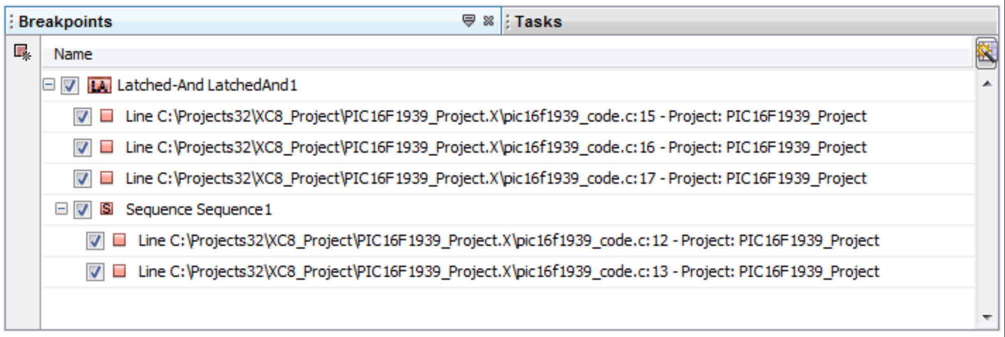
2.7.3 Complex Breakpoint Nesting

Complex breakpoints may be nested to create even more complex breaking schemes.

To nest one group of complex breakpoints into another:

1. Create two groups of complex breakpoints (Sequenced, Latched-And or one of each).
2. Right click on the complex breakpoint group you wish to nest.
3. From the pop-up menu, go to “Complex Breakpoint” and select “Move to *Name*”, where *Name* is the name of the other complex breakpoint group.
4. The first group will appear under the second group, thus creating a scheme.

FIGURE 2-7: NESTED COMPLEX BREAKPOINTS



```
☑ Latched-And LatchedAnd1
    ☑ Line C:\Projects32\XC8_Project\PIC16F1939_Project.X\pic16f1939_code.c:15 - Project: PIC16F1939_Project
    ☑ Line C:\Projects32\XC8_Project\PIC16F1939_Project.X\pic16f1939_code.c:16 - Project: PIC16F1939_Project
    ☑ Line C:\Projects32\XC8_Project\PIC16F1939_Project.X\pic16f1939_code.c:17 - Project: PIC16F1939_Project
    ☑ Sequence Sequence1
        ☑ Line C:\Projects32\XC8_Project\PIC16F1939_Project.X\pic16f1939_code.c:12 - Project: PIC16F1939_Project
        ☑ Line C:\Projects32\XC8_Project\PIC16F1939_Project.X\pic16f1939_code.c:13 - Project: PIC16F1939_Project
```

2.8 STOPWATCH CYCLE COUNTER

The Stopwatch Cycle Counter (32-bit-wide instruction cycle counter) has the ability to perform the existing basic instruction cycle counting (all instruction cycles counted) that exists on standard Enhanced Midrange parts..

Note: The count units are in instruction cycles, not in instructions (as not all instructions execute in a single cycle).

The stopwatch is available under *Window>Debugging>Stopwatch*.

The stopwatch uses two (2) breakpoint resources.

2.9 TRIGGER IN/OUT

The emulation header is capable of producing an output pulse for external triggering and detecting an input pulse for internal triggering.

- Trigger In/Out Hardware
- Trigger In Operation
- Trigger Out Operation

2.9.1 Trigger In/Out Hardware

Pins on the emulation header may be used for Trigger In/Out. Pin functions are labeled on the board silkscreen, namely:

- GND – ground
- TRIG IN – used for Trigger In
- TRIG OUT – used for Trigger Out

2.9.2 Trigger In Operation

A pulse on the Trigger In (TRIG IN) pin can be used to generate a trigger condition, halt and/or trigger out signal. Set up Trigger In by selecting *Window>Debugging>Triggers*.

Selection	Description
Polarity	Trigger-in pin pulse polarity: **Positive:** a positive-going pulse **Negative:** a negative-going pulse
Noise Reduction Filter	reduces the noise on the trigger-in pin using a filter, which helps mitigate spurious triggers from halting code Enables or disables this feature
Trigger Trace	Trigger trace when a pulse is received on the trigger-in pin Enables or disables this feature

An Event Breakpoint can be set up to break when a Trigger In pulse is detected. See Section 2.6 “Enhanced Event Breakpoints”.

2.9.3 Trigger Out Operation

A pulse can be output on the Trigger Out (TRIG OUT) pin based on the setting of breakpoint types:

- Line
- Data or Address (see Section 2.5 “Hardware Address/Data Breakpoints”)
- Event (see Section 2.6 “Enhanced Event Breakpoints”).

The breakpoint can be set up so that the pulse is emitted without halting device execution.