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

ISBN: 978-1-62077-860-9

EEP AND EMULATION HEADER USER'S GUIDE

Table of Contents

Chapter 1. EEP and Emulation Header Overview

This chapter contains the following topics:

- Emulation Extension Pak and Emulation Header Defined
- Why Do I Need An Emulation Header?
- Programming Notes
- General Emulation Header Setup
- Device vs. Optional Header Features
- MPLAB X IDE Use with Headers
- Calibration Bits
- Performance Issues
- Related Debug Tools
- Customer Support

1.1 EMULATION EXTENSION PAK AND EMULATION HEADER DEFINED

An emulation header is a circuit board that allows an emulator 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.

An Emulation Extension Pak (EEP) contains an emulation header, gold single in-line pins, and a trace cable and trace adapter board.

1.2 WHY DO I NEED 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 Vdd, Vss and Vpp, to communicate with the device. Using a debug header can free up these resources for your application.

Using an emulation header gives you the benefits of a debug header plus new and powerful debugging features. These features give you more choices to pick the right debugging feature(s) to efficiently solve the debugging task at hand.

1.2.1 Advances Debug Features

Advanced debug features and benefits are:

- Real-time Hardware Instruction Trace (**RI**)
  - Provides full instruction execution information up to 32 MHz
  - Trace through Reset conditions
  - Trace buffer with optional stall
- Hardware Address/Data Breakpoints - 32 maximum
  - Break on program memory fetch or data read/writes
  - Program or data address range breakpoints
  - Data masking for bit-field breakpoints
  - Data comparison breakpoints
  - Break on ISR and/or main code
  - Break on Pass Count
  - Break and Trigger Out or just Trigger Out
  - Data break on normal or linear modes
  - Break without halting or as a trigger for other events
- Enhanced Event Breakpoints
  - Break on execution out of bounds - watches PC
  - Break on MCLR Reset
  - Break on trigger in signal
- Background Debug
  - Settings and breakpoints may be changed in runtime or sleep time
  - Faster stepping for lower MCU frequencies compared to -ICE/-ICD parts
- Event Combiners - Four (4) available
  - Each event combiner can combine up to eight (8) events
  - Generates a halt or Trigger Out
  - Modeled on MPLAB ICE 2000 In-Circuit Emulator complex triggers
- Execution Out-Of-Bounds Detection
  - Watch for PC values that exceed the available program memory
- Enhanced Stopwatch Cycle Counter
  - 32-bit instruction cycle counter
- External Trigger In/Out
  - Trigger In: Signal falling edge can cause a Halt or Trigger Out
  - Trigger Out: On an event with an enabled trigger, positive pulse is generated

RI = This feature applies to the MPLAB REAL ICE in-circuit emulator only.

For more information on these features, see: **Chapter 2. “Emulation Header Features”**.

1.2.2 Standard Debug Features

For information on standard debug features, see the user’s guide or online help file for your hardware debug tool:

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

Also see the *Processor Extension Pak and Debug Header Specification* (DS51292).

1.3 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 www.microchip.com.

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

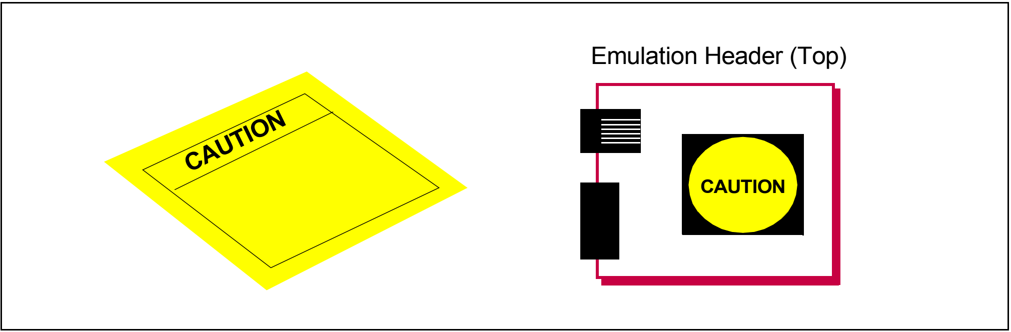
- MPLAB PM3 device programmer
- PICkit 3 development programmer
- MPLAB ICD 3 in-circuit debugger (select as a programmer)
- MPLAB REAL ICE in-circuit emulator (select as a programmer)

1.4 GENERAL EMULATION HEADER 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 1-1).

FIGURE 1-1: 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 Figures 1-2 through 1-4.

FIGURE 1-2: PICkit 3™ IN-CIRCUIT DEBUGGER CONNECTIONS

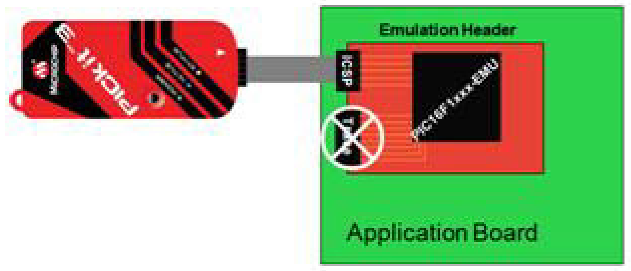


FIGURE 1-3: MPLAB ICD® 3 IN-CIRCUIT DEBUGGER CONNECTIONS

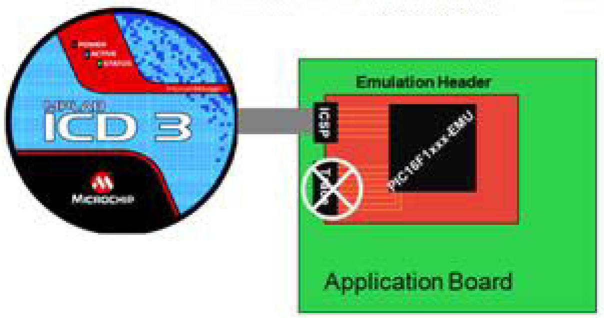
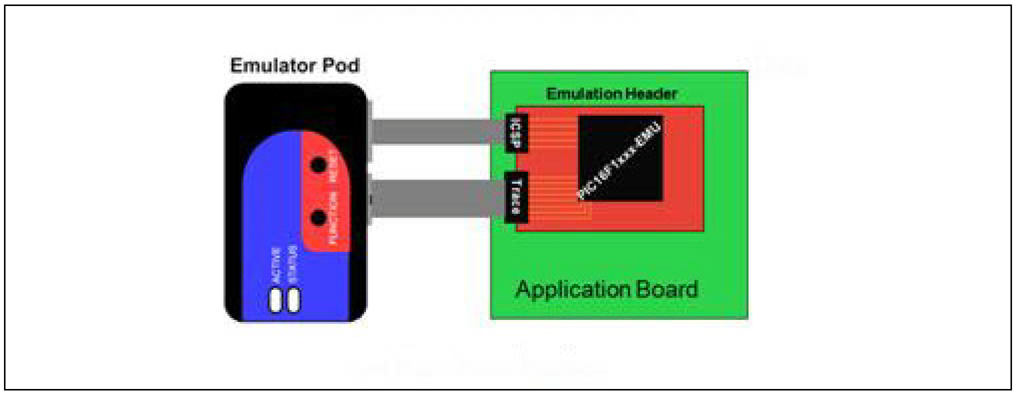


FIGURE 1-4: MPLAB® REAL ICE™ IN-CIRCUIT EMULATOR CONNECTIONS

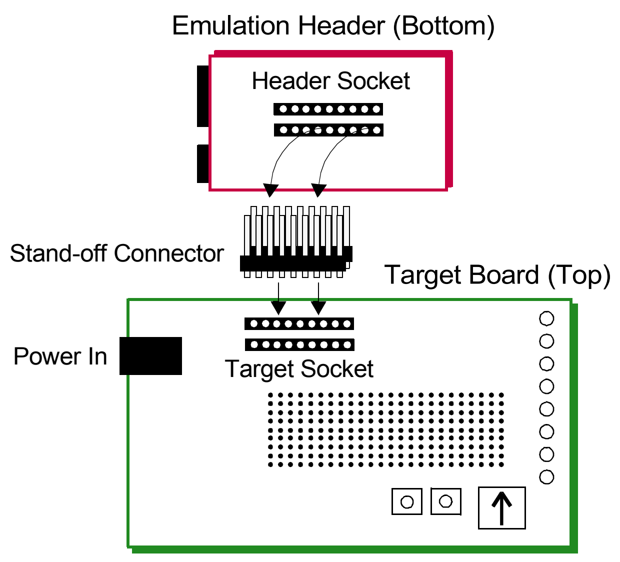


4. Connect the header to the target board. On the bottom of the header is a socket that is used to connect to the target board. The header can be connected to the target board as follows:
   a) PDIP header socket to PDIP target socket with a stand-off (male-to-male) connector or single in-line pins
   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)

   An example connection is shown in Figure 1-5.

   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 1-5: 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.

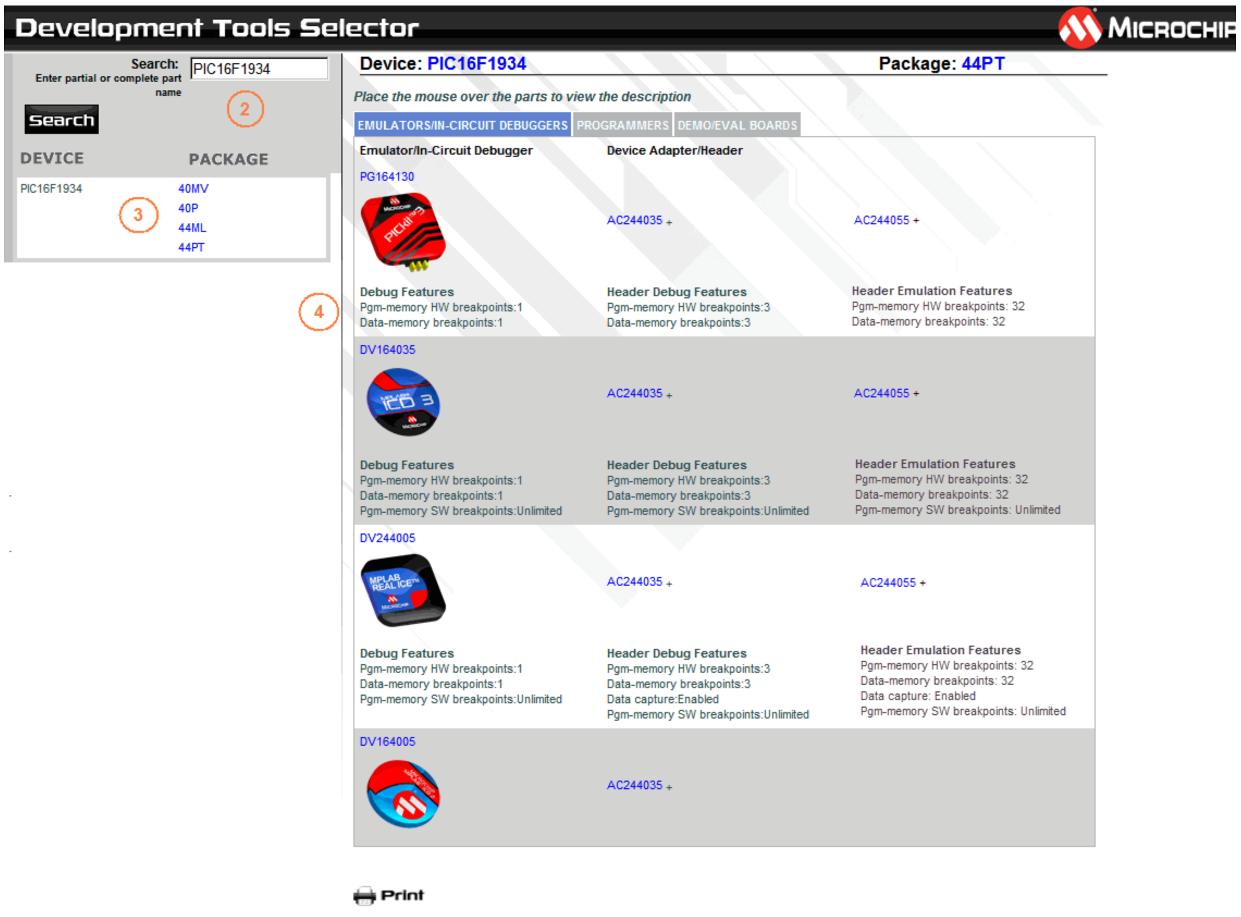
1.5 DEVICE VS. OPTIONAL HEADER FEATURES

This document discusses the benefits of using an emulation header versus devices that have on-board debug capability or debug headers. Another resource for determining the differences in debug features is the Development Tool Selector (DTS).

To find features by device:

1. In a web browser, go to: http://www.microchip.com/dtsapp/
2. Enter your device and click the **Search** button.
3. Select the package you will use.
4. Compare the device under “Debug Features”, “Header Debug Features”, and “Header Emulation Features”.

FIGURE 1-6: DTS DEVICE INFORMATION



Device: PIC16F1934 Package: 44PT

Place the mouse over the parts to view the description

EMULATORS/IN-CIRCUIT DEBUGGERS | PROGRAMMERS | DEMO/EVAL BOARDS

Emulator/In-Circuit Debugger	Device Adapter/Header	
PG164130	AC244035 +	AC244055 +
Debug Features Pgm-memory HW breakpoints:1 Data-memory breakpoints:1	Header Debug Features Pgm-memory HW breakpoints:3 Data-memory breakpoints:3	Header Emulation Features Pgm-memory HW breakpoints: 32 Data-memory breakpoints: 32
DV164035	AC244035 +	AC244055 +
Debug Features Pgm-memory HW breakpoints:1 Data-memory breakpoints:1 Pgm-memory SW breakpoints:Unlimited	Header Debug Features Pgm-memory HW breakpoints:3 Data-memory breakpoints:3 Pgm-memory SW breakpoints:Unlimited	Header Emulation Features Pgm-memory HW breakpoints: 32 Data-memory breakpoints: 32 Pgm-memory SW breakpoints: Unlimited
DV244005	AC244035 +	AC244055 +
Debug Features Pgm-memory HW breakpoints:1 Data-memory breakpoints:1 Pgm-memory SW breakpoints:Unlimited	Header Debug Features Pgm-memory HW breakpoints:3 Data-memory breakpoints:3 Data capture:Enabled Pgm-memory SW breakpoints:Unlimited	Header Emulation Features Pgm-memory HW breakpoints: 32 Data-memory breakpoints: 32 Data capture: Enabled Pgm-memory SW breakpoints: Unlimited
DV164005	AC244035 +	

1.6 MPLAB X IDE USE WITH HEADERS

Emulation header functionality is supported on MPLAB X IDE, 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 “General Emulation Header 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 header.
4. In another step you will specify the hardware (debug) tool to which your 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 “Programming Notes”.

1.7 CALIBRATION BITS

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

1.8 PERFORMANCE ISSUES

The PIC® MCU devices do not support partial program memory erase; therefore, users may experience slower performance than with other devices.

Also, see the in-circuit emulator Help file for information on specific device limitations that could affect performance.

1.9 RELATED DEBUG TOOLS

The following tools support the use of emulation headers:

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

See the Microchip website for the latest documentation: http://www.microchip.com

1.10 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.

NOTES:

Chapter 2. Emulation Header Features

2.1 INTRODUCTION

Emulation headers provide an array of debug features. The following requirements and advantages are discussed here:

- Hardware and Software Requirements
- Real Time Hardware Instruction Trace
- Hardware Address/Data Breakpoints
- Enhanced Event Breakpoints
- Background Debug Mode
- Event Combiners
- Stopwatch Cycle Counter
- Interrupt Context Detection
- Trigger In/Out

2.2 HARDWARE AND SOFTWARE REQUIREMENTS

To use emulation header features, the following are required.

Hardware

- an emulation header that is set up as per **Section 1.4 “General Emulation Header Setup”**.
- for Trace, the trace cable and interface board (included with emulation header) also need to be connected as per above.

Software

- MPLAB X IDE v1.90 or greater
- In the Project Properties window (right click on the project name and select “Properties”), and under “Supported Debug Header”, ensure that the emulation header is selected.

2.3 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:

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

2.3.1 How Trace Works

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.

Enable and set trace options in the Project Properties dialog in the following menu selections:

Categories – "REAL ICE"

Option categories – "Trace and Profiling"
From there, you can set:

"Data selection" – enable/disable trace and select the type of trace.
"Data file path and name" – location of trace file
"Data file maximum size" – size of trace file
"Data Buffer maximum size" – size of the trace buffer

MPLAB X IDE will announce when the trace buffer has overflowed.

Note: Execution will **not** halt when this external buffer is full.

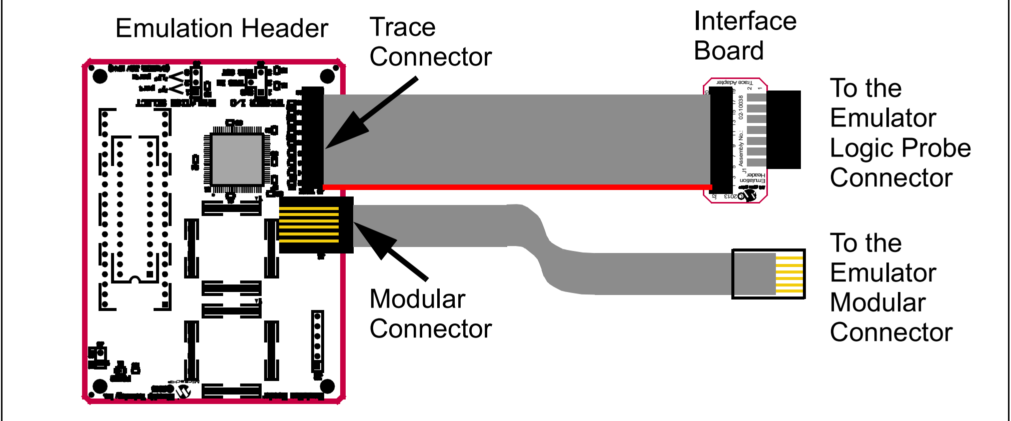
2.3.2 Setting Up and Using Trace

Trace requires hardware and software setup:

2.3.2.1 HARDWARE SETUP

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 1.4 "General Emulation Header Setup"**.

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

2.3.2.2 MPLAB X IDE SETUP

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

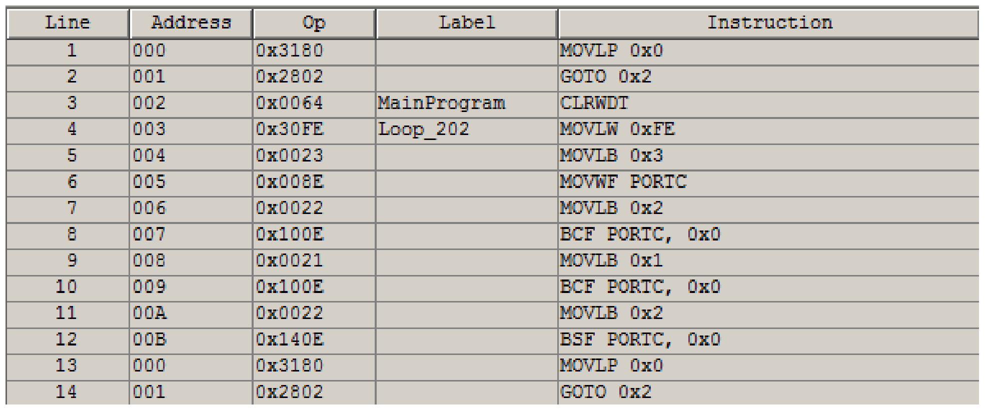
1. Right click on the project name and select "Properties" to open the Project Properties window.
2. Click on "Real ICE" under "Categories".
3. Under "Option categories", select "Clock".
   For data capture and trace, the emulator needs to know the instruction cycle speed.
4. Under "Option categories", select "Trace and Profiling".
5. Under "Data Collection Selection", choose "Instruction Trace/Profiling".
6. Set up any other trace-related options.
7. Click **OK**.

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.

2.3.2.3 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 2-2: 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

2.3.3 Improving the Trace Experience

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

2.3.4 Trace Hardware

Hardware details are shown in the following figures.

FIGURE 2-3: TRACE CONNECTOR ON EMULATION HEADER

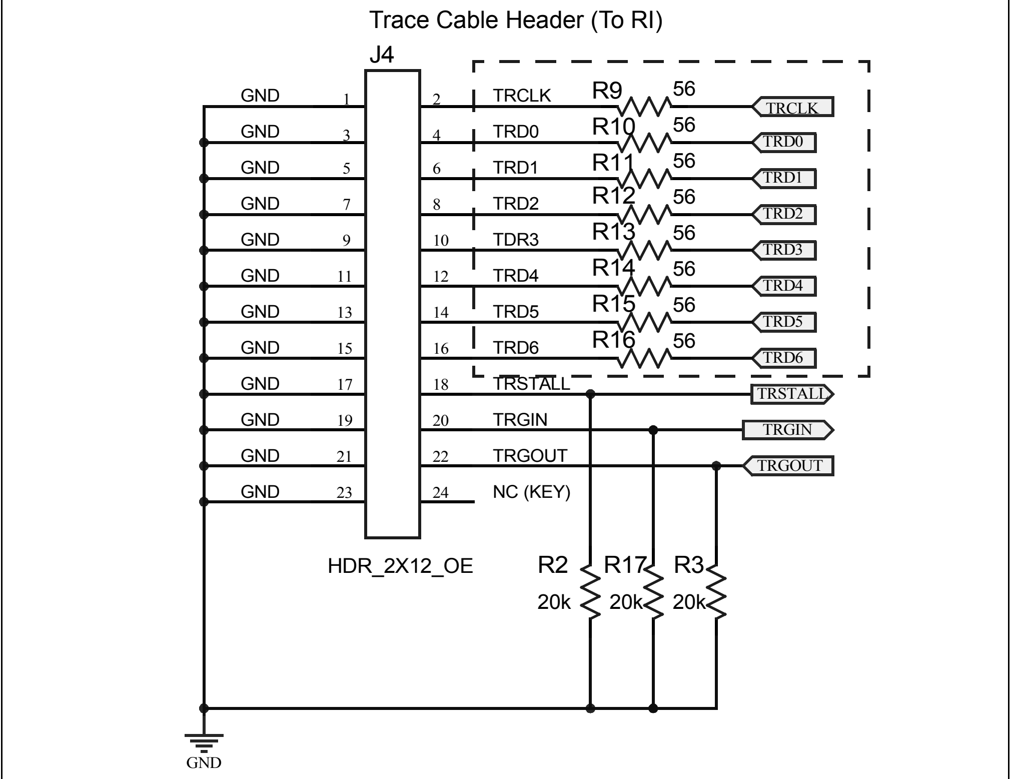
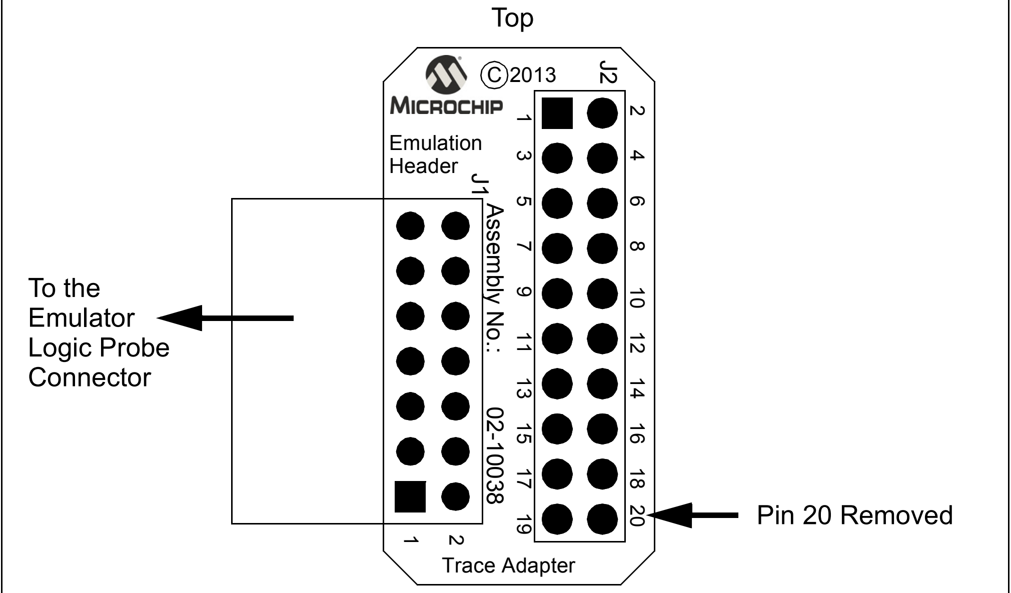

Trace Cable Header (To RI)
J4
GND 1 2 TRCLK R9 56 TRCLK
GND 3 4 TRD0 R10 56 TRD0
GND 5 6 TRD1 R11 56 TRD1
GND 7 8 TRD2 R12 56 TRD2
GND 9 10 TDR3 R13 56 TRD3
GND 11 12 TRD4 R14 56 TRD4
GND 13 14 TRD5 R15 56 TRD5
GND 15 16 TRD6 R16 56 TRD6
GND 17 18 TRSTALL TRSTALL
GND 19 20 TRGIN TRGIN
GND 21 22 TRGOUT TRGOUT
GND 23 24 NC (KEY)
HDR_2X12_OE
R2 20k
R17 20k
R3 20k
GND

FIGURE 2-4: INTERFACE BOARD



2.4 HARDWARE ADDRESS/DATA BREAKPOINTS

Up to 32 hardware address/data breakpoints are available to use on the emulation header. 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.

Some notable breakpoint features are listed here:

- 32 available address/data hardware breakpoints
- Address range breakpoints (break within data or program memory address ranges)
- Data-Masking qualifier for data breakpoints (allows bit-field breakpoints)
- Data-Comparison qualifier for data breakpoints (equality to)
- 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
- Trigger Out qualifier for address/data breakpoints (generate a trigger out signal on event condition) –
  Select from:
  - Do not trigger out when breakpoint is hit
  - Trigger out when breakpoint is hit
- Pass Count qualifier for address/data breakpoints (break on event condition occurring N times)
- Data breakpoints trigger on both normal and linear address modes
- Breakpoints and other events can trigger without halting execution and could be used as trigger events to other features

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”.

2.5 ENHANCED EVENT BREAKPOINTS

For a definition of event breakpoints, see the MPLAB X IDE Help, “New Breakpoint Dialog”. When creating a new breakpoint or customizing an existing breakpoint using an emulation header, additional actions are available for event breakpoints:

Action	Description
Break	break (halt) execution per option specified
Trigger out	emit a trigger out pulse per option specified
Break and trigger out	break (halt) execution AND emit 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 BACKGROUND DEBUG MODE

The emulation header’s on-board -ME2 device contains a Background Debug Mode control interface that allows you read/write access to RAM memory, SFRs, and emulation registers while your program is running or even sleeping.

Background Debug Mode capability includes the following advantages:

- Allows runtime/sleep time changes of In-Circuit Debug (ICD) settings and breakpoints (i.e., runtime address/data/complex/event breakpoints).
- Compared to debug headers (with -ICE or -ICD devices), yields noticeably faster single-stepping speeds at lower MCU operating frequencies.

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*).

- Complex Breakpoint Sequence
- Complex Breakpoint Latched-And
- Complex Breakpoint Nesting

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 2.4 "Hardware Address/Data Breakpoints"** and **Section 2.5 "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.

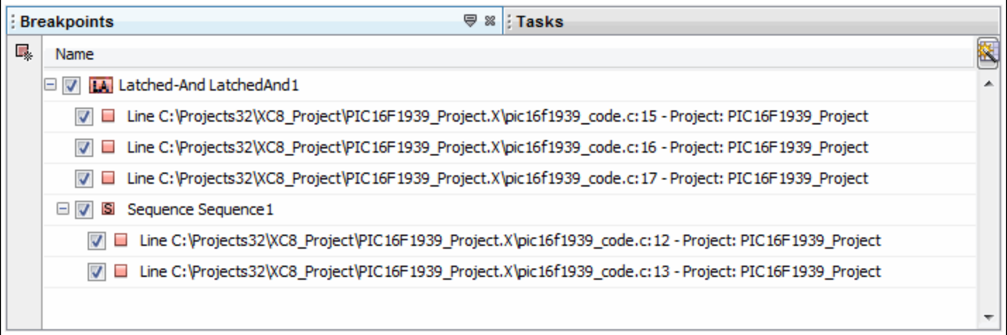
2.7.3 Complex Breakpoint Nesting

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 2-5: 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 (32-bit 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*.

2.9 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 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".

See also, **Section 2.5 "Enhanced Event Breakpoints"**.

2.10 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".

See also, **Section 2.4 "Hardware Address/Data Breakpoints"**.

2.11 TRIGGER IN/OUT

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

2.11.1 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

2.11.2 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	Reduce the noise on the trigger-in pin using a filter, which helps mitigate spurious triggers from halting code. Enable or disable this feature.
Trigger Trace	Trigger trace when a pulse is received on the trigger-in pin. Enable or disable this feature.

An Event Breakpoint can be set up to break when a Trigger In pulse is detected.

See **Section 2.5 “Enhanced Event Breakpoints”**.

2.11.3 Trigger Out Operation

A pulse can be output on the Trigger Out (TRIG OUT) pin based on a setting of either a Program or Data Breakpoint (see **Section 2.4 "Hardware Address/Data Breakpoints"**). The breakpoint may be set up so that the pulse is emitted without halting device execution.

Set up Trigger Out by selecting *Window>Debugging>Triggers*.

Selection	Description
Polarity	Trigger-in pin pulse polarity: **Positive:** a positive-going pulse **Negative:** a negative-going pulse
Slew Rate Limiting	Limit the slew rate on the Trigger Out pulse. **Enable:** slew rate is slow and limited **Disable:** Slew rate is as fast as possible
One Shot	The trigger out pulse duration is: **Enable:** a fixed width, regardless of MCU operating frequency **Disable:** the length of the event itself, which is MCU frequency-dependent
Force Trigger Out	Click this button to force a Trigger Out pulse.

A handy feature of the trigger out signal is that its duration can last as long as the occurring event. For example, if the customer needs to time the duration of the watchdog timer (whose timeout period is user-programmable); the following code, in conjunction with the trigger out and SLEEP event breakpoint features, can make timing an event very simple without employing the old-school technique of writing a single line of (special, non-production) code to wiggle an I/O pin.

The following code is an example:

```
    ; -----------------------------------------------------
    ;  T R I G G E R   O U T   T E S T :
    ;  T I M I N G   T H E   W A T C H D O G   T I M E R
    ; -----------------------------------------------------

    ; 1) Ensure the watchdog timer configuration bit is enabled.
    ; 2) Set an Event Breakpoint (Break on SLEEP) to initiate a
    :    'Trigger out' action only.
    ; 3) Connect your oscilloscope probe to the TRIGGER OUT pin and
    ;    set up your oscilloscope to trigger on the rising edge.
    ; 4) Run the following code:

    CLRWDT
    NOP
    NOP
    SLEEP

    ; The expiration if the watchdog timer will wake up the MCU from
    : sleep after ~2 seconds. The trigger out pulse high-time
    ; duration is the duration of the watchdog timer: ___---___
    ; Since the default watchdog timer period value is 2 seconds
    : typical, the trigger out pulse measured on the oscilloscope
    ; should be approximately 2 seconds.
    NOP
    NOP
    NOP
Loop_101:
    BRA         Loop_101
    ; -----------------------------------------------------
```

Chapter 3. Emulation Header List

3.1 INTRODUCTION

Currently available emulation headers and their associated -ME2 devices are shown below, organized by supported device.

TABLE 1: OPTIONAL DEBUG HEADERS - PIC12/16 DEVICES

Device	Pin Count	Header Part Number	-ME2 Device Used	VDD Max
PIC16F1782	28	AC244064	PIC16F1789-ME2	5.5V
PIC16F1783				
PIC16F1784				
PIC16F1786				
PIC16F1787				
PIC16F1788				
PIC16F1789				
PIC16LF1782	28	AC244064	PIC16F1789-ME2	3.6V
PIC16LF1783				
PIC16LF1784				
PIC16LF1786				
PIC16LF1787				
PIC16LF1788				
PIC16LF1789				
PIC12F1822	8	AC244063	PIC16F1829-ME2	5.5V
PIC12F1840	8			
PIC16F1823	14			
PIC16F1824	14			
PIC16F1825	14			
PIC16F1826	18			
PIC16F1827	18			
PIC16F1828	20			
PIC16F1829	20			
PIC16F1847	18			
PIC12LF1822	8			3.6V
PIC12LF1840	8			
PIC16LF1823	14			
PIC16LF1824	14			
PIC16LF1825	14			
PIC16LF1826	18			
PIC16LF1827	18			
PIC16LF1828	20			
PIC16LF1829	20			
PIC16LF1847	18			