MPLAB® REAL ICE™ In-Circuit Emulator User's Guide For MPLAB X IDE

ISBN: 978-1-5224-1154-3

Object of Declaration: MPLAB REAL ICE In-Circuit Emulator

EU Declaration of Conformity

This declaration of conformity is issued by the manufacturer.
The development/evaluation tool is designed to be used for research and development in a laboratory environment. This development/evaluation tool is not intended to be a finished appliance, nor is it intended for incorporation into finished appliances that are made commercially available as single functional units to end users. This development/evaluation tool complies with EU EMC Directive 2004/108/EC and as supported by the European Commission's Guide for the EMC Directive 2004/108/EC (8th February 2010).
This development/evaluation tool complies with EU RoHS2 Directive 2011/65/EU.
This development/evaluation tool, when incorporating wireless and radio-telecom functionality, is in compliance with the essential requirement and other relevant provisions of the R&TTE Directive 1999/5/EC and the FCC rules as stated in the declaration of conformity provided in the module datasheet and the module product page available at www.microchip.com.
For information regarding the exclusive, limited warranties applicable to Microchip products, please see Microchip's standard terms and conditions of sale, which are printed on our sales documentation and available at www.microchip.com.
Signed for and on behalf of Microchip Technology Inc. at Chandler, Arizona, USA.

Derek Carlson

Derek Carlson
VP Development Tools

11-NOV-16
Date

NOTES:

Table of Contents

NOTES:

Preface

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 "DSXXXXXXXXA", where "XXXXXXXX" 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 help. Select the Help menu, and then Topics to open a list of available help files.

INTRODUCTION

This chapter contains general information that will be helpful to know before using the MPLAB® REAL ICE™ in-circuit emulator.

Items discussed here include:

- Document Layout
- Conventions Used in this Guide
- Recommended Reading - Emulator
- Recommended Reading - Emulator Accessories

DOCUMENT LAYOUT

This document describes how to use the MPLAB REAL ICE in-circuit emulator as a development tool to emulate and debug firmware on a target board, as well as how to program devices. The document is organized as follows:

Part 1 – Overview & Operation

- **Chapter 1: About the Emulator** – What the MPLAB REAL ICE in-circuit emulator is, and how it can help you develop your application.
- **Chapter 2: Device and Feature Support** – Tables of supported features by device families.
- **Chapter 3: Operation** – The theory of MPLAB REAL ICE in-circuit emulator operation and descriptions of configuration options.

Part 2 – **Features**

- **Chapter 4: General Setup** – How to set up MPLAB X IDE to use the emulator.
- **Chapter 5: Common Debug Functions** – A description of basic emulator features available in MPLAB X IDE when the MPLAB REAL ICE in-circuit emulator is chosen as the debug tool. This includes debug features such as breakpoints, stopwatch, and external triggering.
- **Chapter 6: Specific Debug Functions: 8- and 16-Bit Devices** – A description of data capture, runtime watches, trace for 8- and 16-bit (data memory) devices, PC sampling, Application in/out and other debug features.
- **Chapter 7: Specific Debug Functions: 32-Bit Devices** – A description of data capture, runtime watches, trace for 32-bit (data memory) devices, PC sampling, Application in/out and other debug features.

Part 3 – **Troubleshooting**

- **Chapter 8: Troubleshooting First Steps** – The first things you should try if you are having issues with emulator operation.
- **Chapter 9: Frequently Asked Questions (FAQ)** – A list of frequently asked questions about emulator operation and issues.
- **Chapter 10: Messages** – A list of error messages and suggested resolutions.
- **Chapter 11: Engineering Technical Notes (ETNs)** – Any resolvable hardware issues are listed.

Part 4 – **Software & Hardware Reference**

- **Chapter 12: Emulator Function Summary** – A summary of emulator functions available in MPLAB IDE when the MPLAB REAL ICE emulator is chosen as the debug or program tool.
- **Chapter 13: Hardware Specification** – The hardware and electrical specifications of the emulator system. Includes a description of how to use the loopback test board.

CONVENTIONS USED IN THIS GUIDE

The following conventions may appear in this documentation:

TABLE 1: DOCUMENTATION CONVENTIONS

Description	Represents	Examples
Arial font:		
Italic	Referenced books	*MPLAB® X IDE User's Guide*
	Emphasized text	...is the *only* compiler...
Initial caps	A window	the Output window
	A dialog	the Settings dialog
	A menu selection	select Enable Programmer
Quotes	A field name in a window or dialog	"Save project before build"
Underlined, italic text with right angle bracket	A menu path	<u>*File>Save*</u>
Bold	A dialog button	Click **OK**
	A tab	Click the **Power** tab
Text in angle brackets < >	A key on the keyboard	Press <Enter>, <F1>
Courier font:		
Plain	Sample source code	`#define START`
	Filenames	`autoexec.bat`
	File paths	`c:\mcc18\h`
	Keywords	`_asm, _endasm, static`
	Command-line options	`-Opa+, -Opa-`
	Bit values	`0, 1`
	Constants	`0xFF, 'A'`
Italic	A variable argument	*`file`*`.o`, where *`file`* can be any valid filename
Square brackets [ ]	Optional arguments	`mpasmwin [options]` *`file`* `[options]`
Curly brackets and pipe character: { | }	Choice of mutually exclusive arguments; an OR selection	`errorlevel {0|1}`
Ellipses...	Replaces repeated text	`var_name [, var_name...]`
	Represents code supplied by user	`void main (void) { ... }`

RECOMMENDED READING - EMULATOR

The following Microchip documents are available and recommended as supplemental reference resources.

Multi-Tool Design Advisory (DS51764)

A small document on guidelines and implementation considerations to ensure proper interfacing to the various development tools.

Release Notes for MPLAB REAL ICE In-Circuit Emulator

For the latest information on using the MPLAB REAL ICE in-circuit emulator, read the “`Readme for MPLAB REAL ICE Emulator.htm`” file (an HTML file) by clicking on “Release Notes, User’s Guide and Support Docs” on the Start Page. The release notes (Readme) contain update information and known issues that may not be included in this document.

Using the MPLAB REAL ICE In-Circuit Emulator (DS51997)

This poster shows you how to hook up the hardware and install the software for the MPLAB REAL ICE in-circuit emulator.

MPLAB REAL ICE In-Circuit Emulator Help

An on-line version of the comprehensive emulator user’s guide in MPLAB X IDE. Usage, troubleshooting and hardware specifications are included.

Processor Extension Pak (PEP) and Debug Header Specification (DS50001292), Emulation Extension Pak (EEP) and Emulation Header User’s Guide (DS50002243)

These booklets describe how to install and use debug and emulation headers. Headers are used to better debug selected devices using special -ME2/-ICE/-ICD device versions, without the loss of pins or resources. Extension Paks contain headers. See also the related help files.

Transition Socket Specification (DS51194)

Consult this document for information on transition sockets available for use with headers.

RECOMMENDED READING - EMULATOR ACCESSORIES

The following Microchip documents describe available accessories for the emulator system.

Performance Pak User's Guide (DS50002528)

A Performance Pak (AC244002) contains the connectors and cables for high-speed/LVDS communications. This document describes how to connect the hardware for the emulator to target communications, shows target circuits that should and should not be used, and provides details on the hardware components.

MPLAB REAL ICE Trace Interface Kit Specification (DS50002531)

A trace interface kit (AC244006) provides a cable and adapter board for Instruction Trace using supporting PIC32 MCUs/PIMs or emulation headers. This document contains a list of supported PIC32 PIMs, an example of a PIC32 PIM configuration, and hardware details of the kit components.

MPLAB REAL ICE Isolation Unit Instruction Sheet (DS50001858)
MPLAB REAL ICE Isolation Unit Specification (DS50002529)

An opto-isolation unit (AC244005) is used to protect the emulator when developing high-power applications.The instruction sheet shows you how to hook up the unit hardware. The specification provides additional information about the unit hardware.

MPLAB REAL ICE JTAG Adapter Instruction Sheet (DS52094)
MPLAB REAL ICE JTAG Adapter Specification (DS50002530)

The JTAG adapter board (AC244007) provides JTAG functionality to the emulator. (Be aware you will not have the full suite of debug features when using this board.) The instruction sheet shows you how to hook up the JTAG adapter board. The specification provides additional information about the board hardware.

MPLAB REAL ICE Power Monitor Instruction Sheet (DS50002156)
MPLAB REAL ICE Power Monitor User's Guide (DS50002532)

The power monitor board (AC244008) allows you to view the power usage on your target board.The instruction sheet shows you how to hook up the power monitor. The user's guide provides additional information on software setup, all hardware configurations, and using the power monitor.

NOTES:

Part 1 – Overview & Operation

NOTES:

Chapter 1. About the Emulator

1.1 INTRODUCTION

The MPLAB REAL ICE in-circuit emulator is a modern emulator that supports hardware and software development for selected Microchip PIC® microcontrollers (MCUs) and dsPIC® Digital Signal Controllers (DSCs).

An overview of the emulator is provided in this chapter:

- Emulator Features
- How the Emulator Helps You
- Emulator Kit Components
- Emulator System

1.2 EMULATOR FEATURES

The MPLAB REAL ICE emulation concept provides these features:

- Processors run at maximum speeds
- Debugging can be done with the device in-circuit
- No emulation load on the processor bus
- Simple interconnection
- Capability to incorporate I/O data
- Instrumented Trace (MPLAB X IDE and Compiler Assisted)
- Instruction Trace (Hardware trace)

In addition to emulator functions, the MPLAB REAL ICE in-circuit emulator system can also be used as a production programmer.

1.3 HOW THE EMULATOR HELPS YOU

The MPLAB REAL ICE in-circuit emulator is an integral part of the development engineer's toolsuite. Application usage can vary from software development to hardware integration, manufacturing test, or field service.

The MPLAB REAL ICE in-circuit emulator system enables you to:

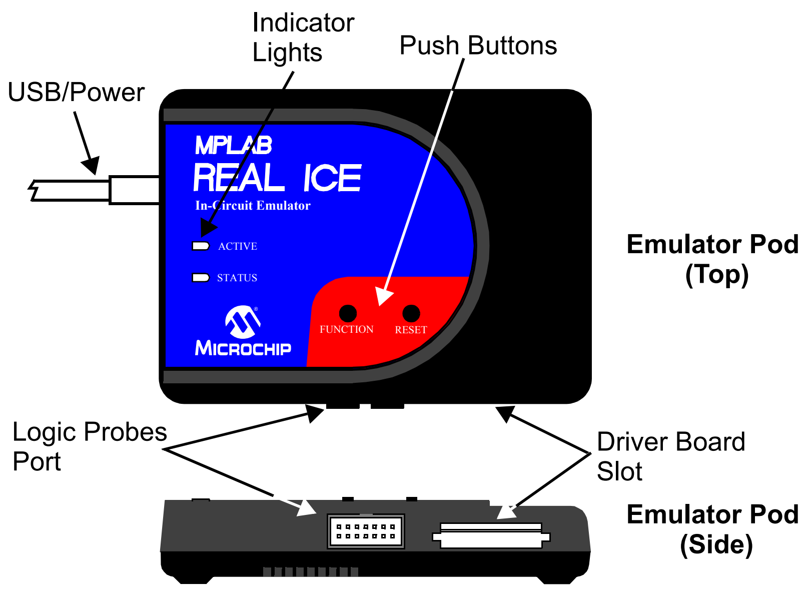
- Debug an application on hardware in real time
- Debug with hardware breakpoints
- Debug with software breakpoints (device-dependent)
- Halt, based on internal events and/or external signals
- Monitor internal file registers
- Emulate full speed
- Program devices as a production programmer
- Trace lines of code or log variable/expression values

1.4 EMULATOR KIT COMPONENTS

The components of the MPLAB REAL ICE In-Circuit Emulator System Kit (DV244005) are listed below.

- Emulator pod – See Section 12.4 "Emulator Pod".
- USB cable – Provides communications between the emulator and a PC, and provides power to the emulator. See Section 12.3 "USB Port/Power".
- Standard driver board and cable – Connects the emulator pod to a header module or target board. See Section 12.5 "Standard Communication Hardware".
- Loopback test board – Verifies emulator operation. See Section 12.7 "Loopback Test Board".

FIGURE 1-1: BASIC EMULATOR POD



Additional hardware may be ordered separately:

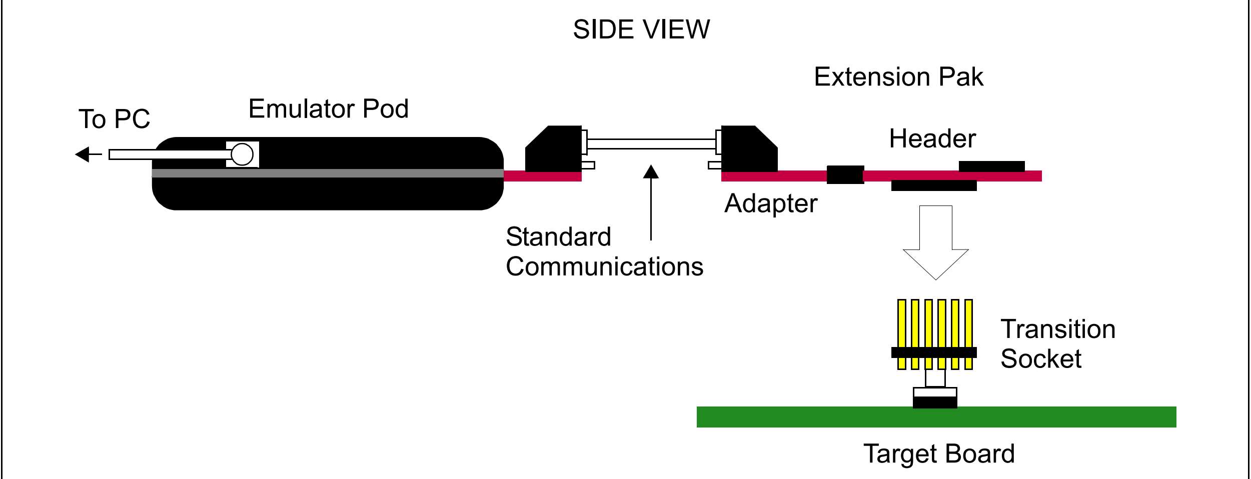
- Performance Pak (AC244002) – Used for high-speed/LVDS communications and SPI Trace.
- Processor/Emulation Extension Pak – Provides a device-specific debug or emulation header and other hardware.
- Transition socket – Connects a header to the target.
- Logic probes (ACICE0104) – Used for external triggers and I/O port trace.
- MPLAB REAL ICE Trace Interface Kit (AC244006) - Used with device-specific Plug-In Modules (PIMs) that support Instruction Trace.
- MPLAB REAL ICE Isolator unit (AC244005) – Optically isolates the emulator from the target, which is useful for high-power applications.
- MPLAB REAL ICE JTAG Adapter (AC244007) – Provides JTAG communication between the emulator and a target PIC32 device.
- MPLAB REAL ICE Power Monitor (AC244008) – Allows the emulator to monitor current and voltage of the target board or device.

1.5 EMULATOR SYSTEM

The MPLAB REAL ICE in-circuit emulator is an in-circuit emulator that is controlled by a PC running the MPLAB X IDE integrated development environment. The emulator communicates with a debug/emulation header or a device that has on-board debug/emulation circuitry. A device is usually connected directly to a target board, whereas a header may be connected directly to or through a transition socket.

An example emulator system configuration is shown in Figure 1-2.

FIGURE 1-2: EXAMPLE EMULATOR SYSTEM SETUP



NOTES:

Chapter 2. Device and Feature Support

Information on feature support by device and tool may be found under the:

- Development Tool Selector (DTS) - by individual device
- Device & Feature Support HTML Tables - for all devices

2.1 DEVELOPMENT TOOL SELECTOR (DTS)

Find the Development Tool Selector (DTS) on the Development Tools section of the Microchip website (copy and paste into a browser):

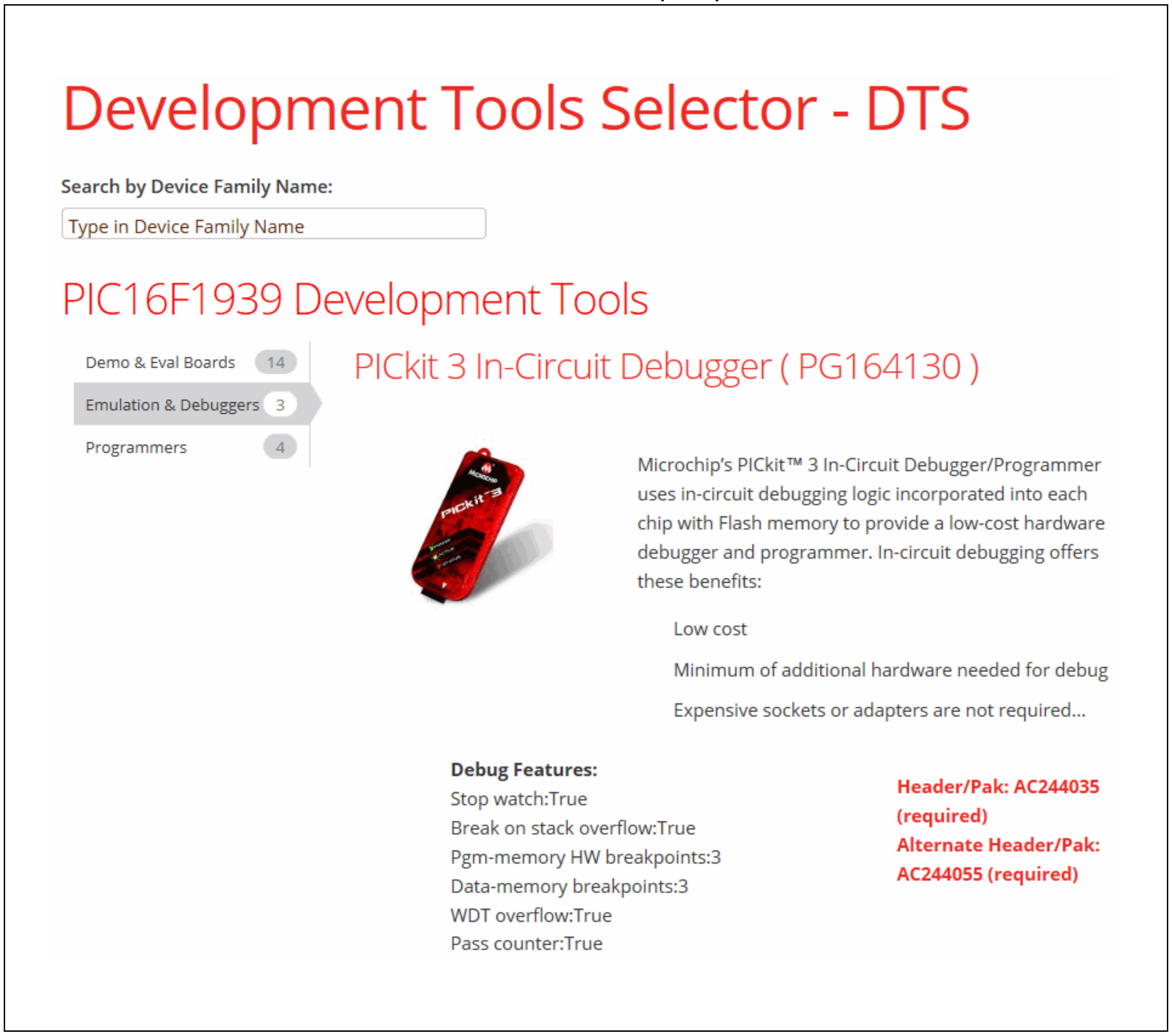
http://www.microchip.com/dts

On the DTS web page, start typing a device family name and select from the list that appears below the search box. After your device has been selected, click on the "Emulators and Debuggers" tab for a list of supported features by tool (and header).

Caveat

For devices and tools supporting Date Capture, Runtime Watches, and Native Trace: at speeds higher than 15 MIPS, the Performance Pak may be needed.

FIGURE 2-1: DEVELOPMENT TOOLS SELECTOR (DTS) WEB PAGE



Development Tools Selector - DTS

Search by Device Family Name:

Type in Device Family Name

PIC16F1939 Development Tools

Demo & Eval Boards	14
Emulation & Debuggers	3
Programmers	4

PICkit 3 In-Circuit Debugger (PG164130)

Microchip's PICkit™ 3 In-Circuit Debugger/Programmer uses in-circuit debugging logic incorporated into each chip with Flash memory to provide a low-cost hardware debugger and programmer. In-circuit debugging offers these benefits:

Low cost

Minimum of additional hardware needed for debug

Expensive sockets or adapters are not required...

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

Header/Pak: AC244035 (required)
Alternate Header/Pak: AC244055 (required)

2.2 DEVICE & FEATURE SUPPORT HTML TABLES

To view HTML tables, with feature support, for all devices and tools:

1. In MPLAB X IDE, go to the Start Page, “Learn & Discover” tab, and click the link “Users Guide & Release Notes”. Your default browser will open containing the document, *User’s Guides, Release Notes and Support Documentation in the MPLAB® X IDE Installation*.
2. On the browser page, go to the “Device & Feature Support” section. Select:
   a) Hardware Tool Debug Features by Device - view debug features (breakpoints, trace, etc.) for Microchip hardware tools (e.g., PICkit 3) by device.
   b) Simulator Debug Features by Device - view debug features (breakpoints, trace, etc.) for the MPLAB X IDE Simulator by device.
   c) Simulator Peripheral Support by Device - view the peripherals supported in the MPLAB Simulator by device.

   Note: These tables are large and make take some time to load into your browser window.

3. A CSV (comma separated values) document version of each HTML table is available to support importing data into spread sheet programs and sorting the data. Click the link on the top of each HTML table to get to the CSV file.

NOTES:

Chapter 3. Operation

3.1 INTRODUCTION

A simplified description of how the MPLAB REAL ICE in-circuit emulator system works is provided here. It is intended to provide enough information so that a target board can be designed that is compatible with the emulator for both emulation and programming operations. The basic theory of in-circuit emulation and programming is described so that problems, if encountered, are quickly resolved.

- Tools Comparison
- Operational Overview
- Emulator Communication with the PC
- Emulator Communication with the Target
- Trace Connections
- Debugging with the Emulator
- Requirements For Debugging
- Programming with the Emulator
- Resources Used by the Emulator

3.2 TOOLS COMPARISON

The MPLAB REAL ICE in-circuit emulator system differs physically and operationally from the other Microchip debug tools shown below. Specific features may vary by device (see the online help file for "Device and Feature Support".)

TABLE 3-1: DEBUG TOOLS COMPARISON

Features	MPLAB® REAL ICE™ In-Circuit Emulator	MPLAB ICD 3 In-Circuit Debugger	PICkit™ 3 In-Circuit Debugger
USB Speed	High and Full	High and Full	Full Only
USB Driver	Microchip	Microchip	HID
USB Powered	✓ (1)	✓	✓
Power to Target	✗	✓	✓
Programmable VPP and VDD	✓	✓	✓
VDD Drain from Target	< 50 µA	< 50 µA	20 mA
Max current to Target	None(1)	100 mA	30 mA
Overvoltage/ Overcurrent Protection	✓ (Hardware)	✓ (Hardware)	✓ (Software)
HW Breakpoints	Complex	Complex	Simple
Stopwatch	✓	✓	✓
SW Breakpoints	✓	✓	✗
Non-Volatile Program Image	✗	✗	✓ (512K bytes)
Serialized USB	✓	✓	✓
Trace	✓	✗	✗
Data Capture	✓	✗	✗
Logic Probe Triggers	✓	✗	✗
High Speed/LVDS Connection	✓	✗	✗
Production Programmer	✓	✓	✗

Note 1: Target can be powered if the Power Monitor accessory is used.

3.3 OPERATIONAL OVERVIEW

The emulator is connected to the PC via a USB port for communication and emulator power (but, not target power). The emulator is connected to the target application for communication and data collection, such as trace. Below is a summary of possible connection configurations.

FIGURE 3-1: EMULATOR CONNECTIONS

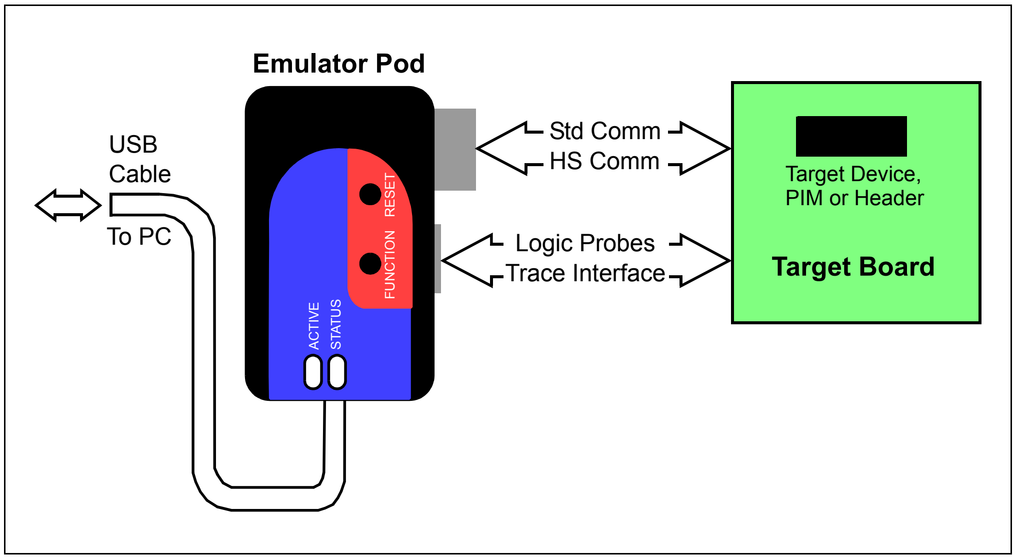


TABLE 3-2: CONNECTIONS FOR EMULATING DEVICES

Connection	Debug Support [1,2]	Trace Support [1,3]	Support Speed
Standard Communications	Data Capture, Runtime Watch	Native Trace	15 MIPS or less PIC32: Device Dependent
High-Speed Communications	Data Capture, Runtime Watch	Native Trace, SPI Trace	Greater than 15 MIPS PIC32: Device Dependent
Logic Port Probes [4]	N/A	I/O Port Trace	Device Dependent
MPLAB REAL ICE Trace Interface Kit (Logic Port)	N/A	Instruction Trace	Device Dependent

Note 1: Support is device dependent. See online help.
2: For details, see Section 6.2 “Data Capture and Runtime Watches”.
3: For details, see.Section 6.3.3 “Types of Trace” or Section 6.4 “PIC32 Instruction Trace – PIC32 MCUs Only”.
4: For details, see Section 3.6.3 “I/O Port Trace Connections (Logic Port)”.

3.4 EMULATOR COMMUNICATION WITH THE PC

The emulator is powered, and communicates with the PC, through the USB cable. The emulator cannot power the target from USB power. To power the target, use the MPLAB REAL ICE Power Monitor (AC244008).

CAUTION

Communication Failure.
Do not connect the hardware before installing the software and USB drivers.

3.5 EMULATOR COMMUNICATION WITH THE TARGET

The emulator communicates with the target via configurations that are described in the following sections:

- Standard Communication
- High-Speed/LVDS Communication (Performance Pak)

CAUTION

Emulator or Target Damage.
Do not change hardware connections while the pod or target is powered.

⚠ DANGER

Potential Electrical Hazard.
If your application uses AC line or high voltage power not referenced to ground, you should use an isolation circuit and the MPLAB REAL ICE Isolator unit (AC244005).

3.5.1 Standard Communication

The emulator system can be configured to use the standard connection for communicating debug and programming instructions to the target. This 6-pin connection is the same one used by other Microchip in-circuit debuggers.

3.5.1.1 STANDARD CONNECTION

The standard driver board is inserted into the emulator pod to configure the system for communication with the target. One end of the modular cable is plugged into the standard driver board and the other end is plugged into one of the following locations:

1. A matching socket on the target, where the target device (or device-specific plug-in module) is on the target board, as shown in Figure 3-2
2. A standard adapter/header combo – available as an Extension Pak – which is then plugged into the target board, as shown in Figure 3-3.

Note: Older debug headers used a 6-pin (RJ-11) connector instead of an 8-pin connector. In the case of these older headers, the cable can be connected directly to the emulator.

To see the debug features of a device versus those of a header, see the DTS at http://www.microchip.com/dts. For more on the hardware, see Section 12.5 “Standard Communication Hardware”.

FIGURE 3-2: STANDARD CONNECTION – DEVICE WITH ON-BOARD ICE CIRCUITRY

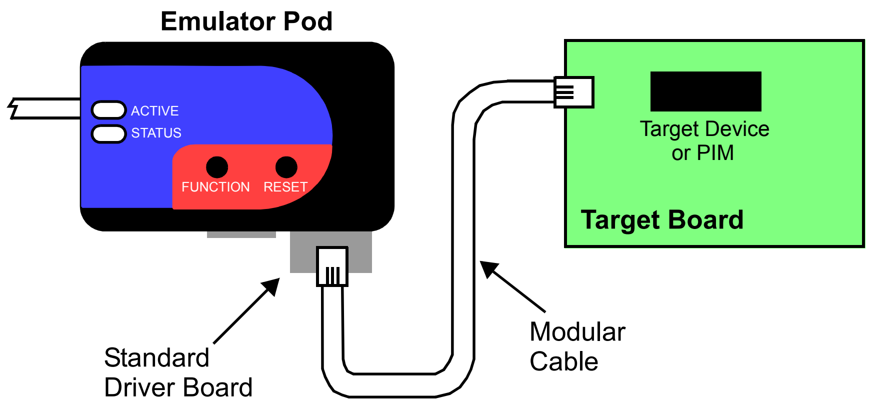
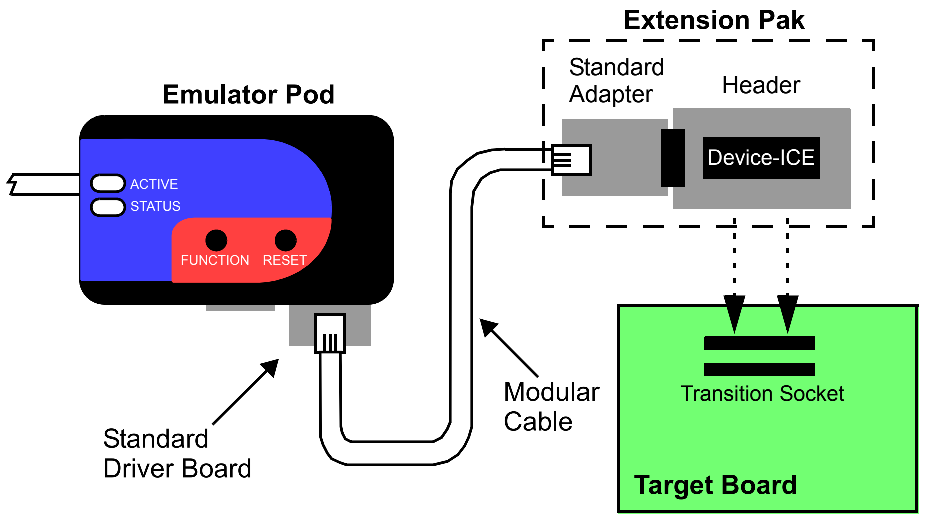


FIGURE 3-3: STANDARD CONNECTION – ICE DEVICE

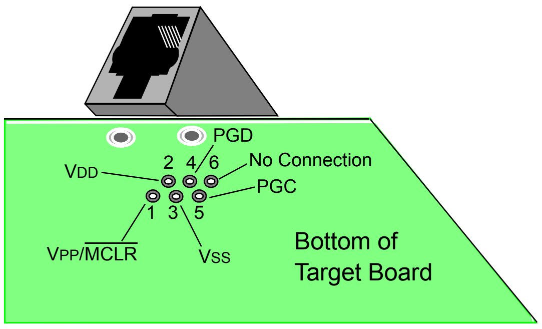


3.5.1.2 STANDARD CONNECTOR AT THE TARGET

To use the standard driver board, the MPLAB REAL ICE in-circuit emulator is connected to the target device with the modular interface (six-conductor) cable. The pin numbering for the connector is shown from the perspective of the bottom of the target PC board in Figure 3-4.

Note:	Cable connections at the emulator and the target are mirror images of each other; i.e., pin 1 on one end of the cable is connected to pin 6 on the other end of the cable. See Section 12.5.2.1 “Modular Cable Specification”.

FIGURE 3-4: STANDARD CONNECTION AT TARGET



3.5.2 High-Speed/LVDS Communication (Performance Pak)

The emulator system can be configured to use the high-speed/LVDS connection for communicating debug and programming instructions to the target. Compared to standard communication, this form of communication provides the following features.

- Noise cancellation from the low-voltage differential signal (LVDS) technology, which allows:
  - Communication speeds greater than 15 MIPS for data capture, runtime watches, and Native trace
  - Longer distances between the emulator and the target
  - Operation in noisy environments
- Two additional pins used for SPI trace

The Performance Pak (AC244002) is necessary for high-speed/LVDS communications. For connection information, see the “*Performance Pak User’s Guide*” (DS50002528).

3.5.3 Target Connection Circuitry

Figure 3-5 shows the interconnections of the MPLAB REAL ICE in-circuit emulator through the target board connector to a device on the target board. The interconnection is very simple. Any problems experienced are often caused by other connections or components on these critical lines that interfere with emulator operation, as discussed in Section 3.5.4 "Target Circuit Design Precautions".

FIGURE 3-5: STANDARD CONNECTION TO TARGET CIRCUITRY

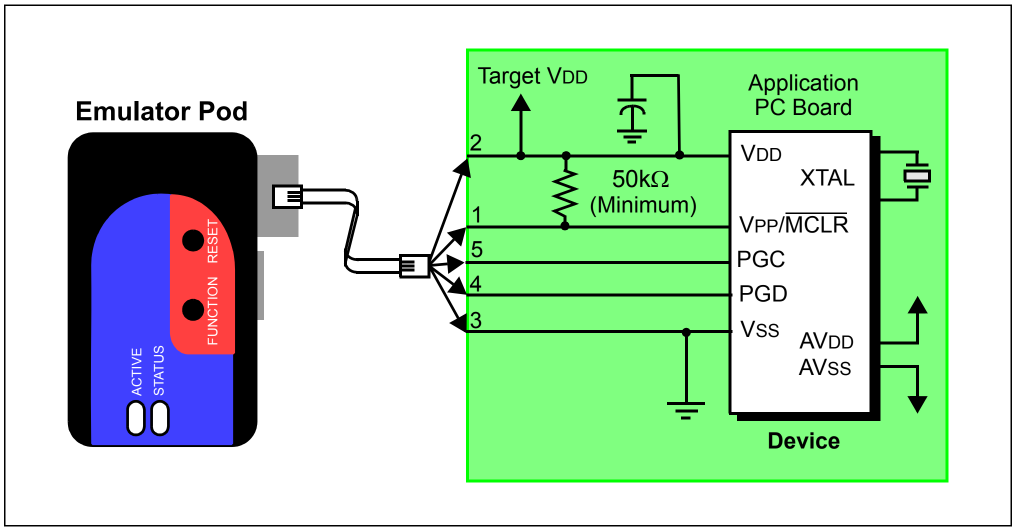


TABLE 3-3: TARGET CONNECTOR AND RELATED DEVICE I/O

Pin No.	Device I/O	Description
1	VPP/$\overline{\text{MCLR}}$	The emulator requires access to VPP for programming and debugging the device.
2	VDD	Target VDD is sensed by the emulator to allow level translation for target low-voltage operation and to detect a device. If the emulator does not sense voltage on its Vdd line, it will not connect with the device. **Note:** The emulator DOES NOT provide target power.
3	VSS	Target VSS is sensed by the emulator. **Note:** The emulator DOES NOT provide target VSS or ground.
4	PGD	The emulator requires access to PGD and PGC for programming and debugging the device.
5	PGC	

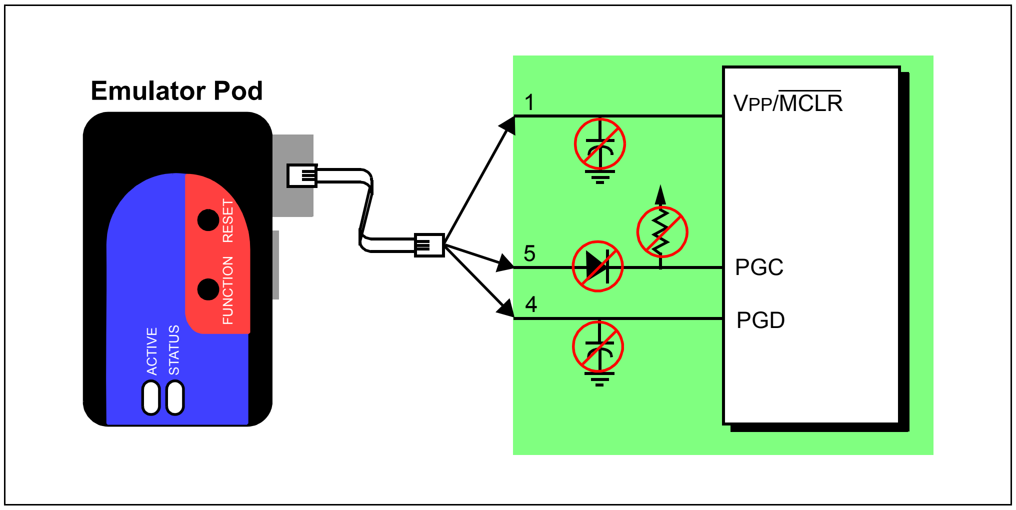
TABLE 3-4: CIRCUITRY ON DEVICE I/O

Device I/O	Description
VPP/$\overline{\text{MCLR}}$ and VDD	A pull-up resistor (minimum 50kΩ) should be connected from the VPP/MCLR line to VDD so that the line may be strobed low to reset the device.
XTAL	The target device must be running with an oscillator for the emulator to function as a debugger.
AVDD, AVSS	Not all devices have the AVDD and AVSS lines, but if they are present on the target device, all must be connected to the appropriate levels in order for the emulator to operate. This also applies to voltage regulator pins (e.g., ENVREG/DISVREG on PIC24FJ MCUs).
VDD, VSS, AVDD, AVSS	In general, it is recommended per device data sheet that all VDD/AVDD and VSS/AVSS lines be connected to the appropriate levels. For devices with a VCAP pin (e.g., PIC18FXXJ devices), the appropriately-valued capacitor should be placed as close to the Vcap pin as possible.

3.5.4 Target Circuit Design Precautions

Figure 3-6 shows the active emulator lines with some example components that will prevent the MPLAB REAL ICE in-circuit emulator system from functioning.

FIGURE 3-6: IMPROPER CIRCUIT COMPONENTS



- **Do not use capacitors on MCLR** – they will prevent fast transitions of VPP.
- **Do not use pull-ups on PGC/PGD** – they will divide the voltage levels because these lines have 4.7 kΩ pull-down resistors in the emulator.
- **Do not use multiplexing on PGC/PGD** – they are dedicated for communications to the emulator.
- **Do not use capacitors on PGC/PGD** – they will prevent fast transitions on data and clock lines during programming and debug communications.
- **Do not use diodes on PGC/PGD** – they will prevent bidirectional communication between the emulator and the target device.
- **Do not exceed recommended cable lengths** – for acceptable cable lengths, see Section 12.5.2 “Modular Cable and Connector”.

Additional design information is available in the “*Development Tools Design Advisory*” (DS51764).

For other operational issues, see these locations in this document:

- Chapter 8. “Frequently Asked Questions (FAQ)”
- Chapter 9. “Messages”
- Section 9.3.6 “Debug Failure Actions” (Top Reasons Why You Can’t Debug)
- Section 12.7 “Loopback Test Board”

3.6 TRACE CONNECTIONS

Depending on your selected device, one or more trace capabilities may be available when the emulator is selected as the debug tool.

3.6.1 Native Trace Connections

No additional connections are necessary to use Native trace. The communications connection will carry the trace information using the PGD/PGC/EMUC/EMUD pins. However, the selected device must have this feature. If it does not, one of the other trace methods may be used.

For more on this type of trace, see Section 6.3.3.1 "Native Trace".

3.6.2 SPI Trace Connections (High-Speed/LVDS Connection)

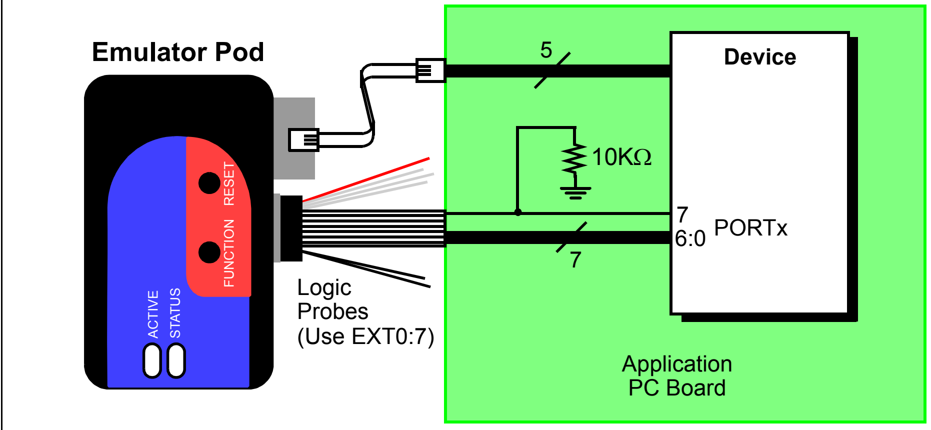
Serial trace is an optional trace that is only available by using the Performance Pak (AC244002). For details, see the "*Performance Pak User's Guide*" (DS50002528).

For more on this type of trace, see Section 6.3.3.2 "SPI Trace".

3.6.3 I/O Port Trace Connections (Logic Port)

Parallel trace is possible using a device 8-pin I/O port and the emulator logic probes. This provides greater trace speed and data quantity, but limits emulator-to-target distance by the length of the logic probes. Figure 3-7 shows these additional connections.

FIGURE 3-7: PARALLEL TRACE CONNECTIONS



For this trace configuration, seven (7) lines of data and one (1) line for clock are transmitted. PORTx must be a port with 8 pins that has all 8 pins available for trace. The port does not have to be one physical port but can be made up of pins from more than one port. The port pins must not be multiplexed with the currently-used PGC and PGD pins.

For allowable PORTx configurations:

1. Right click on your project in the Projects window and select "Properties".
2. In the Project Properties window, click on the "REAL ICE" category.
3. Select the "Trace and Profiling" option category from the drop down list.
4. Under "Data Collection Selection", select the trace that is supported for your device, e.g., "User Instrumented Trace".
5. Under "Communications Medium", select "I/O Port".
6. Under "I/O Port Selection", select your port configuration from the list.

A basic configuration is shown in the following table.

TABLE 3-5: I/O PORT TRACE CONNECTION EXAMPLE

PORTx pin	Logic Probe pin(1)	Content
0	EXT0	Data
1	EXT1	Data
2	EXT2	Data
3	EXT3	Data
4	EXT4	Data
5	EXT5	Data
6	EXT6	Data
7	EXT7(2)	Clock

Note 1: For pin descriptions, see Section 12.4.4 "Logic Probe/External Trigger Interface".

2: Use a 10KΩ pull-down resistor for noise reduction.

As in Section 3.5.4 "Target Circuit Design Precautions", do not use pull-up or pull-down resistors, capacitors or diodes on port pins, except as specified.

For more on this type of trace, see Section 6.3.3.3 "I/O Port Trace".

3.6.4 Instruction Trace Connections

Instruction trace is a non-intrusive hardware trace used to capture every instruction executed by the device. There are two types of Instruction Trace:

- PIC32 Instruction Trace
- Real Time Hardware Instruction Trace

3.6.4.1 PIC32 INSTRUCTION TRACE

PIC32 Instruction Trace is only available for PIC32 MCU devices. Also, only some PIC32 MCU devices have the trace feature. Consult your device data sheet for details.

To use this trace, you will need the following hardware:

- PIC32 Plug-In Module (PIM) that contains a device that supports trace and has a trace port
- MPLAB® REAL ICE™ Trace Interface Kit (AC244006) that contains a 12-inch trace cable and a trace adapter board

If you do not have a trace cable, you can use the logic probes. Connect them as indicated below.

TABLE 3-6: LOGIC PROBE CONNECTIONS

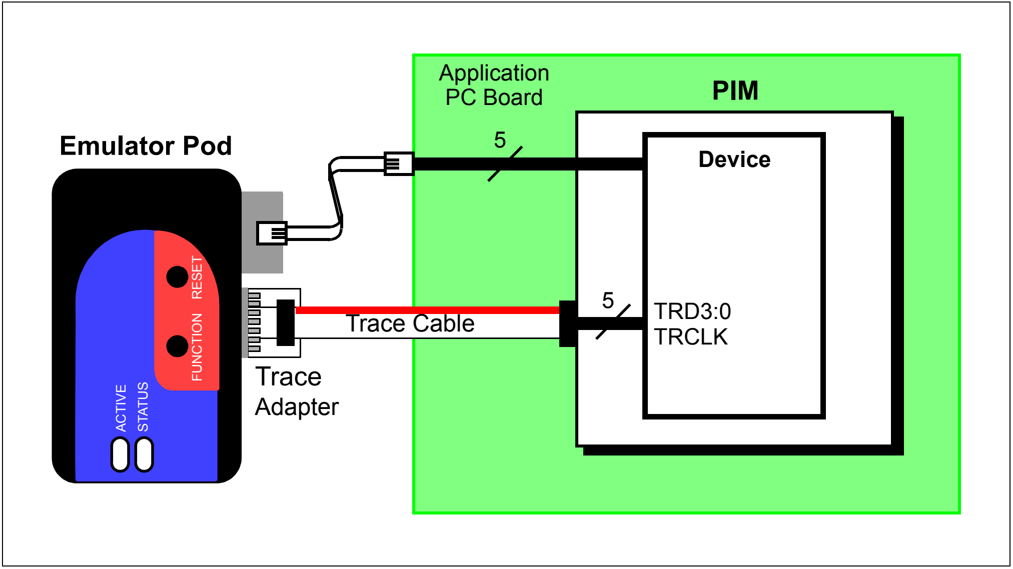
Logic Probe Port Pins(1)		**PIM Trace Pins**(2)	
No.	**Name**	**No.**	**Name**
4	TCLK	1	TRCLK
12	EXT0 (TRIG1)	3	TRD0
11	EXT1 (TRIG2)	5	TRD1
10	EXT2 (TRIG3)	7	TRD2
9	EXT3 (TRIG4)	9	TRD3

Note 1: For more information, see Section 12.4.4 "Logic Probe/External Trigger Interface".

2: For more information, see Section 6.4.4 "Trace Hardware Specifications".

To use the PIC32 Instruction Trace feature, see Section 6.4 "PIC32 Instruction Trace – PIC32 MCUs Only".

FIGURE 3-8: PIC32 TRACE CONNECTION WITH PIM



3.6.4.2 REAL TIME HARDWARE INSTRUCTION TRACE

Real Time Hardware Instruction Trace is a real-time dump of the program execution stream that can be captured and analyzed. This type of trace is available on selected emulation headers (-ME2 devices).

For details, see the "*Emulation Extension Pak (EEP) and Emulation Header User's Guide*" (DS50002243).

3.7 DEBUGGING WITH THE EMULATOR

There are two steps to using the MPLAB REAL ICE in-circuit emulator system as a debugger.

1. The first requires that an application be programmed into the target device.
2. The second uses the internal in-circuit debug hardware of the target Flash device to run and test the application program.

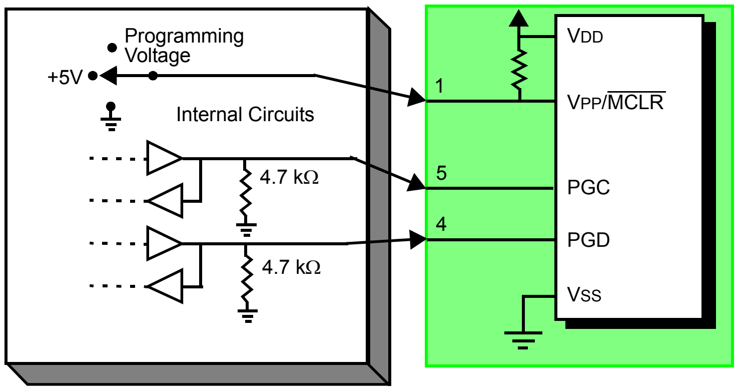
Those two steps are directly related to the following MPLAB IDE operations:

- Programming the code into the target and activating special debug functions (see the next section for details).
- Debugging the code using features such as breakpoints.

If the target device cannot be programmed correctly, the MPLAB REAL ICE in-circuit emulator will not be able to debug.

Figure 3-9 shows the basic interconnections required for programming and debugging. Note that this is the same as Figure 3-5, but for the sake of clarity, the VDD and VSS lines from the emulator are not shown.

FIGURE 3-9: PROPER CONNECTIONS FOR PROGRAMMING



A simplified diagram of some of the internal interface circuitry of the MPLAB REAL ICE in-circuit emulator pod is shown. For programming, no clock is needed on the target device, but power must be supplied. When programming, the emulator puts programming levels on VPP, sends clock pulses on PGC and serial data via PGD. To verify that the part has been programmed correctly, clocks are sent to PGC and data is read back from PGD. This conforms to the In-Circuit Serial Programming™ (ICSP™) protocol of the device under development. See the device programming specification for details.

3.8 REQUIREMENTS FOR DEBUGGING

To debug (set breakpoints, see registers, etc.) with the MPLAB REAL ICE in-circuit emulator system, there are critical elements that must be working correctly:

- The emulator must be connected to a PC. It must be powered by the PC via the USB cable, and it must be communicating with MPLAB IDE software via the USB cable. See Chapter 5. "Basic Debug Functions" for details.
- The emulator must be connected (as shown) to the VPP, PGC and PGD pins of the target device with the modular interface cable (or equivalent). VSS and VDD are also required to be connected between the emulator and target device.
- The target device must have power and a functional, running oscillator – either internal or external. If the target device will not run (for whatever reason), the MPLAB REAL ICE in-circuit emulator cannot debug.
- The target device must have its Configuration words programmed correctly:
  - The oscillator Configuration bits should correspond to RC, XT, etc., depending on the target design.
  - On some devices, the Watchdog Timer is enabled by default and needs to be disabled.
  - The target device must not have code protection enabled.
  - The target device must not have table read protection enabled.
  - For some devices with more than one PGC/PGD pair, the correct pair needs to be configured. This is only needed for debugging since programming will work over any PGC/PGD pair.

3.8.1 Sequence of Operations Leading to Debugging

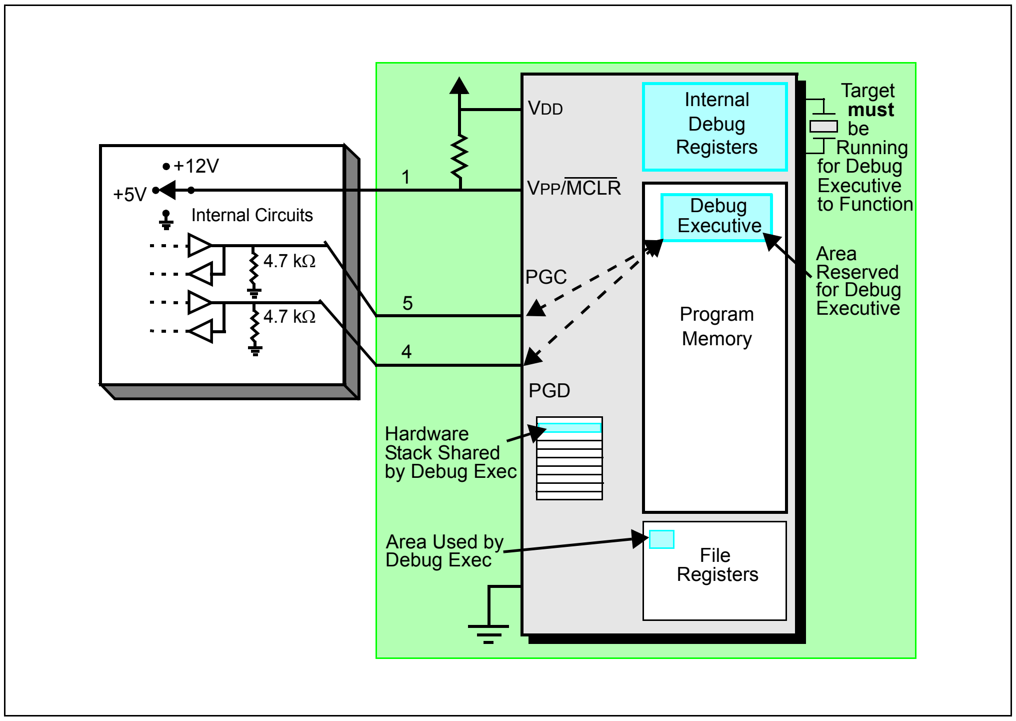
Given that the Requirements For Debugging (from the section above) are met, the following actions can be performed when the MPLAB REAL ICE in-circuit emulator is set as the current tool (*File>Project Properties*, "Hardware Tool" category):

- When *Debug>Debug Project* is selected, the application code is programmed into the device's memory via the ICSP protocol, as described earlier.
- A small "debug executive" program is loaded into high program memory of the target device. Because the debug executive must reside in program memory, the application program must not use this reserved space. Some devices have special memory areas dedicated to the debug executive. Check your device data sheet for details.
- Special "in-circuit debug" registers in the target device are enabled by MPLAB IDE. These allow the debug executive to be activated by the emulator. For more on device reserved resources, see the online help file.
- The target device is run in Debug mode.

3.8.2 Debugging Details

Figure 3-10 illustrates the MPLAB REAL ICE in-circuit emulator system when it is ready for debugging.

FIGURE 3-10: MPLAB® REAL ICE™ IN-CIRCUIT EMULATOR



Typically, in order to find out if an application program will run correctly, a breakpoint is set early in the program code. When a breakpoint is set from the user interface of MPLAB IDE, the address of the breakpoint is stored in the special internal debug registers of the target device. Commands on PGC and PGD communicate directly to these registers to set the breakpoint address.

Next, the *Debug>Debug Project* function is usually selected in MPLAB IDE. The emulator then tells the debug executive to run. The target starts from the Reset vector and executes until the Program Counter reaches the breakpoint address that was stored previously in the internal debug registers.

After the instruction at the breakpoint address is executed, the in-circuit debug mechanism of the target device "fires" and transfers control to the debug executive (much like an interrupt) and the user's application is effectively halted. The emulator communicates with the debug executive via PGC and PGD, gets the breakpoint status information and sends it back to MPLAB IDE. MPLAB IDE then sends a series of queries to the emulator to get information about the target device, such as file register contents and the state of the CPU. These queries are ultimately performed by the debug executive.

The debug executive runs just like an application in program memory. It uses some locations on the stack for its temporary variables. If the device does not run, for whatever reason, such as no oscillator, a faulty power supply connection, shorts on the target board, etc., then the debug executive cannot communicate to the MPLAB REAL ICE in-circuit emulator and MPLAB IDE will issue an error message.

Another way to stop execution is to select *Debug>Pause*. This toggles the PGC and PGD lines so that the in-circuit debug mechanism of the target device switches execution from the user's code in program memory to the debug executive. Again, the target application program is effectively halted, and MPLAB IDE uses the emulator communications with the debug executive to interrogate the state of the target device.

3.9 PROGRAMMING WITH THE EMULATOR

Use the MPLAB REAL ICE in-circuit emulator as a programmer to program a production device, i.e., a device that is not on a debug header. Set the MPLAB REAL ICE in-circuit emulator as the current tool (*File>Project Properties*, Hardware Tool) to perform these actions:

- When *Run>Run Project* is selected, the application code is programmed into the device's memory via the ICSP protocol as described in the previous section. No target oscillator clock is required while programming, and all modes of the processor can be programmed, including code protect, Watchdog Timer enabled, and table read protect.
- A small "program executive" program may be loaded into the high area of program memory for some target device. This increases programming speeds for devices with large memories.
- Special "in-circuit debug" registers in the target device are disabled by MPLAB IDE, along with all debug features. This means that a breakpoint cannot be set, and register contents cannot be seen or altered.
- The target device is run in Release mode. As a programmer, the emulator can only toggle the $\overline{\text{MCLR}}$ line to reset and start the target.

3.10 RESOURCES USED BY THE EMULATOR

For a complete list of resources used by the emulator for your device, please see the online help file in MPLAB IDE for the MPLAB REAL ICE in-circuit emulator.

NOTES:

Part 2 – Features

NOTES:

Chapter 4. Working with the Emulator

4.1 INTRODUCTION

How to get started using the MPLAB REAL ICE in-circuit emulator is discussed.

4.2 INSTALLATION AND SETUP

Refer to the Help file "Getting Started with MPLAB X IDE" for details on installing the IDE and setting up the emulator to work with it.

In summary:

1. Install MPLAB X IDE.
2. Install the USB drivers as specified.
3. Connect to the PC. For information on target and trace connections, see Chapter 3. "Operation".

> **Note:** The emulator CANNOT power a target board.
> However, you can use the Power Monitor accessory with the emulator to power your target. See the "*MPLAB REAL ICE Power Monitor User's Guide*" (DS50002532).

4. Install the language toolsuite/compiler you want to use for development.
5. Launch MPLAB X IDE.
6. Use the New Project wizard (*File>New Project*) to add your "Real ICE" emulator to your project.
7. Use the project Properties dialog (*File>Project Properties*) to set up emulator options.
8. Run the project (build and run) from *Run>Run Project*.

Items of note are:

1. Specific instructions apply for installing USB drivers on a Windows® system. See MPLAB X IDE documentation for details.
2. Each emulator contains a unique identifier which, when first installed, will be recognized by the operating system (OS), regardless of which computer USB port is used.
3. MPLAB X IDE operation connects to the hardware tool at runtime (Run or Debug Run). To always be connected to the hardware tool (i.e. MPLAB IDE v8), click in the *Tools>Options*, **Embedded** button, **Generic Settings** tab, "Keep hardware tool connected" checkbox.
4. Configuration bits must now be set in code. You can set up Configuration bits in the Configuration window (*Window>PIC Memory Views>Configuration Bits*) and then click "Generate Code".

4.3 FIRMWARE UPGRADES

IN GENERAL, YOU DO NOT NEED TO PERFORM MANUAL FIRMWARE UPDATES

When you install a version of MPLAB X IDE, the associated version of the MPLAB REAL ICE in-circuit emulator firmware (operating system) will be installed on your system. The next time you use the emulator, MPLAB X IDE will automatically upgrade the firmware on the emulator. You can see the progress in the Output window.

If you want to use a version of MPLAB X IDE with a firmware version other than the one MPLAB X IDE uses by default, you will have to manually install the firmware as described in the following sections.

4.3.1 Default Firmware File Location

The default firmware file (.jam) is stored in a Java archive (JAR) file for the debug tool (i.e., REALICE.jar).

The default location of the firmware is:

```
<install path>MPLABX/vx.xx/mplab_ide/mlablibs/modules/ext/REALICE.jar
```

4.3.2 Manual Firmware Installation

For an active project that uses the emulator:

1. Select *File>Project Properties* to open the Project Properties window.
2. Click on the "Real ICE" category and select "Firmware" from the drop-down Option Categories.
3. Uncheck "Use Latest Firmware".
4. Click in the "Firmware File" field that says, "Press to browse for a specific firmware file". A long button will appear. Click it.
5. In the dialog under "Directories", browse to the location of the file containing the firmware file you want (`REALICE.jar`). Select this file.
6. In the dialog under "Firmware Files", select the firmware file (`.jam`) and click **OK**.
7. In the Project Properties window, click **Reset**.

4.3.3 Communication Interruption When Installing Firmware

If the emulator becomes disconnected from the PC when firmware is being installed, reconnect the emulator. It will begin to erase what had been written so it can restart. This erasing will take about 75 seconds (1:15 min.). Please be patient.

You will see:

- the orange busy light turn on for approximately 25 seconds
- the light turn red for another 25 seconds
- the light turn orange again for another 25 seconds, or so

When it turns back to red again, MPLAB X IDE will recognize the device and start the recovery process, i.e., begin the firmware download.

4.4 EMULATOR FUNCTIONS

Information about using those emulator functions that are common to all devices is provided in Chapter 5. "Basic Debug Functions".

Information about using emulator functions that are specific to particular device families is provided in Chapter 6. "Specific Debug Functions".

4.5 QUICK DEBUG/PROGRAM REFERENCE

The following table is a quick reference for using the MPLAB REAL ICE in-circuit emulator as a debug or as a program tool. See previous chapters for information about proper emulator setup and configuration.

TABLE 4-1: DEBUG VS. PROGRAM OPERATION

Item	Debug	Program
Needed Hardware	A PC and target application (Microchip demo board or your own design.)	
	Emulator pod, USB cable, communication driver board(s) and cable(s).	
	Device with on-board debug circuitry or debug header with special -ME2/-ICE/-ICD device.	Device (with or without on-board debug circuitry.)
MPLAB X IDE selection	Project Properties, REAL ICE as Hardware Tool	
	Debug>Debug Run	Program Target Project toolbar button
Program operation	Programs application code into the device. Depending on the selections on the Project Properties dialog, this can be any range of program memory. In addition, other debug resources are reserved. On the MPLAB X IDE Start Page, click "Release Notes and Support Documentation" to find the reserved resources for the emulator.	Programs application code into the device. Depending on the selections on the Project Properties dialog, this can be any range of program memory.
Debug features available	All debug features available for the target device – breakpoints, trace, etc.	N/A.
Command-line operation	N/A	Use REALICECMD, found by default in: `<install path>\MPLABX\vx.xx\mplab_ipe`
Serial Quick-Time Programming (SQTP)	N/A	Right click on a debugged project and select "Export Hex". Import the hex file into MPLAB IPE and use MPLAB PM3 to SQTP the device.

4.6 DEBUGGER/PROGRAMMER LIMITATIONS

For a complete list of emulator limitations for your device, see the online help file in MPLAB X IDE for the MPLAB REAL ICE in-circuit emulator.

NOTES:

Chapter 5. Basic Debug Functions

5.1 INTRODUCTION

Basic debug functions are discussed below.

- Starting and Stopping Emulation
- Viewing Processor Memory and Files
- Breakpoints and Stopwatch
- Device Debug Functions

Debug functions that are only available specifically for the emulator are discussed later, in **Chapter 6. "Specific Debug Functions"**.

5.2 STARTING AND STOPPING EMULATION

To debug an application in MPLAB X IDE, you must create a project containing your source code so that the code may be built, programmed into your device, and executed as specified below:

- To run your code, select either *Debug>Debug Project* or **Debug Project** from the Run toolbar.
- To halt your code, select either *Debug>Pause* or **Pause** from the Debug toolbar.
- To run your code again, select either *Debug>Continue* or **Continue** from the Debug toolbar.
- To step through your code, select either *Debug>Step Into* or **Step Into** from the Debug toolbar. Be careful not to step into a Sleep instruction or you will have to perform a processor Reset to resume emulation.
- To step over a line of code, select either *Debug>Step Over* or **Step Over** from the Debug toolbar.
- To end code execution, select either *Debug>Finish Debugger Session* or **Finish Debugger Session** from the Debug toolbar.
- To perform a processor Reset on your code, select either *Debug>Reset* or **Reset** from the Debug toolbar. Additional Resets, such as POR/BOR, MCLR and System, may be available, depending on device.

5.3 VIEWING PROCESSOR MEMORY AND FILES

MPLAB X IDE provides several windows for viewing debug and various processor memory information. These are selectable from the Window menu. See MPLAB X IDE online help for assistance on using these windows.

- *Window>PIC Memory Views* – view the different types of device memory

  Depending on the selected device, memory types include Program Memory, SFRs, Configuration Memory, etc.
- *Window>Debugging* – view debug information

  Select from variables, watches, call stack, breakpoints, stopwatch, and trace. (Trace is discussed more thoroughly in upcoming chapters.)

To view your source code, find the source-code file you wish to view in the Projects window and double click it to open it in a Files window. Code in this window is color-coded according to the processor and build tool selected. To change the style of color-coding, select *Tools>Options*, **Fonts & Colors**, **Syntax** tab.

For more on the Editor, see MPLAB X IDE online help, Editor section.

5.4 BREAKPOINTS AND STOPWATCH

Use breakpoints to halt code execution at specified lines in your code. Use the stopwatch with breakpoints to time code execution.

- Breakpoint Resources
- Hardware or Software Breakpoint Selection
- Breakpoint and Stopwatch Usage

5.4.1 Breakpoint Resources

For 16-bit devices, breakpoints, data captures, and runtime watches use the same resources. So, the available number of breakpoints is actually the available number of combined breakpoints/triggers.

For 32-bit devices, breakpoints use different resources than data captures and runtime watches. So, the available number of breakpoints is independent of the available number of triggers.

The number of hardware and software breakpoints available and/or used is displayed in the Dashboard window (*Window>Dashboard*). See the MPLAB IDE documentation for more on this feature. Not all devices have software breakpoints.

For limitations on breakpoint operation, including the general number of hardware breakpoints per device and hardware breakpoint skidding amounts, see “Limitations” in the online help file.

5.4.2 Hardware or Software Breakpoint Selection

To select hardware or software breakpoints:

1. Select your project in the Projects window. Then, select either *File>Project Properties* or right click and select “Properties”.
2. In the Project Properties dialog, select “REAL ICE” under “Categories”.
3. Under “Option Categories” select “Debug Options”.
4. Check “Use software breakpoints” to use software breakpoints. Uncheck to use hardware breakpoints.

Note: Using software breakpoints for debug impacts device endurance. Therefore, it is recommended that devices used in this manner not be used as production parts.

To help you decide which type of breakpoints to use (hardware or software) the following table compares the features of each.

TABLE 5-1: HARDWARE VS. SOFTWARE BREAKPOINTS

Feature	Hardware Breakpoints	Software Breakpoints
Number of breakpoints	Limited	Unlimited
Breakpoints written to*	Internal debug registers	Flash Program Memory
Breakpoints applied to**	Program Memory/Data Memory	Program Memory only
Time to set breakpoints	Minimal	Dependent on oscillator speed, time to program Flash Memory and page size
Breakpoint skidding	Most devices. See the online help, Limitations section, for details.	No

* Where information about the breakpoint is written in the device.
** What kind of device feature applies to the breakpoint. This is where the breakpoint is set.

5.4.3 Breakpoint and Stopwatch Usage

Breakpoints halt execution of code. To determine the time between the breakpoints, use the stopwatch.

Refer to the MPLAB X IDE Help for how to set up and use breakpoints and the stopwatch.

5.5 DEVICE DEBUG FUNCTIONS

In addition to the debug features covered previously, devices have the other debug features, such as “Freeze on Halt”, which allows you to freeze selected peripherals on a halt. For more on these functions, see MPLAB X IDE documentation.

NOTES:

Chapter 6. Specific Debug Functions

6.1 INTRODUCTION

The following debug functions or the implementations of the debug functions are specific to the emulator. Some debug functions are dependent on other debug functions. For information on basic debug functions, see Chapter 5. "Basic Debug Functions".

- Data Capture and Runtime Watches
- Instrumented Trace
- PIC32 Instruction Trace – PIC32 MCUs Only
- Jump Trace – EP Devices Only
- PC Sampling – 8-Bit and 16-Bit MCUs Only
- PC Profiling – 32-Bit MCUs Only
- Function Level Profiling
- Application In/Out
- External Triggers
- Additional Debug Features

6.2 DATA CAPTURE AND RUNTIME WATCHES

Data capture is an on-board debug feature of the device. When a value in the Capture Data Address register matches an SFR register address, a trigger and data capture occurs. You cannot access data capture directly; you must select an application that uses data capture, such as the DMCI plug-in.

Runtime Watches are selected symbols in a Watches window that change as the program runs. You can select a symbol to be runtime in a Watches window. See MPLAB X IDE documentation for details.

8- and 16-Bit Devices

Not all 8- and 16-bit devices support data capture and/or runtime watches. A list of supported features by device is available in online help or the Development Tools Selector (DTS) at http://www.microchip.com/developmenttools/.

When watching symbols, an 8-bit PIC device can only watch 8-bit variables and a 16-bit PIC device can only watch 16-bit variable.

For 8- and 16-bit devices, data captures, runtime watches and hardware breakpoints use the same registers/resources. For example, if you use a data capture resource for a symbol, you will not be able to use a hardware breakpoint or runtime watch resource for that symbol.

For data captures and runtime watches at speeds higher than 15 MIPS, the Performance Pak may be needed. The actual speed may vary depending on layout, noise, and similar considerations.

32-Bit Devices

Not all 32-bit devices support data capture and/or runtime watches. A list of supported features by device is available in online help.

For PIC32 devices, hardware breakpoints **do not** use data capture or runtime watch resources. However data captures and runtime watches **do** use the same resources. Therefore, if you use a data capture resource for a symbol, you will not be able to use a runtime watch resource for that symbol, and vice versa.

Note:	The Performance Pak is not required for debugging at maximum speeds as the capture clock speed is independent of the target system clock.

- Data Capture and DMCI
- Runtime Watches and DMCI – PIC32 MCUs Only
- Runtime Watches and the Watches Window

6.2.1 Data Capture and DMCI

Data capture provides streaming data from a device to the following:

Data Monitoring and Control Interface (DMCI) – Plug-in

To install the DMCI plug-in:

1. Select *Tools>Plugins*. The Plugins window will open.
2. Click on the **Available Plugins** tab.
3. Find DMCI and check the checkbox next to it.
4. Click **Install** and follow the screens.

See also the "Add Plug-in Tools" in the "Additional Tasks" section of the MPLAB X IDE help file.

To set up data capture:

1. Build the project (In the Projects window, right click on the project name and select "Build"). The project must be built to see the available symbols.
2. Select *Window>Debugging>Watches* to open the Watches window.
3. Right click in the window and select "New Watch". Select the symbol or SFR wish to watch in the New Watch window. Click **OK**.
4. Select *Tools>Embedded>DMCI>DMCI Window* to open the DMCI dialog.
5. Set up the DMCI for this data capture. See the DMCI help file for details (*Help>Help Contents*, "Plug-In Tools" section).
6. Begin a debug session (*Debug>Debug Project*). Input data using DMCI controls or view data in a DMCI graphical window.

6.2.2 Runtime Watches and DMCI – PIC32 MCUs Only

For devices without data capture, the runtime watch can provide data polling to the following:

Data Monitoring and Control Interface (DMCI) – Plug-in

To install the DMCI plug-in:

1. Select *Tools>Plugins*. The Plugins window will open.
2. Click on the **Available Plugins** tab.
3. Find DMCI and check the checkbox next to it.
4. Click **Install** and follow the screens.

See also the "Add Plug-in Tools" in the "Additional Tasks" section of the MPLAB X IDE help file.

To set up runtime watches:

1. Build the project (In the Projects window, right click on the project name and select “Build”). The project must be built to see the available symbols.
2. Select *Window>Debugging>Watches* to open the Watches window.
3. Right click in the window and select “New Runtime Watch”. Select the symbol or SFR wish to watch in the New Run Time Watch window. Click **OK**.
4. Select *Tools>Embedded>DMCI>DMCI Window* to open the DMCI dialog.
5. Set up the DMCI for this runtime watch. See the DMCI help file for details (*Help>Help Contents*, “Plug-In Tools” section).
6. Begin a debug session (*Debug>Debug Project*). Input data using DMCI controls or view data in a DMCI graphical window.

6.2.3 Runtime Watches and the Watches Window

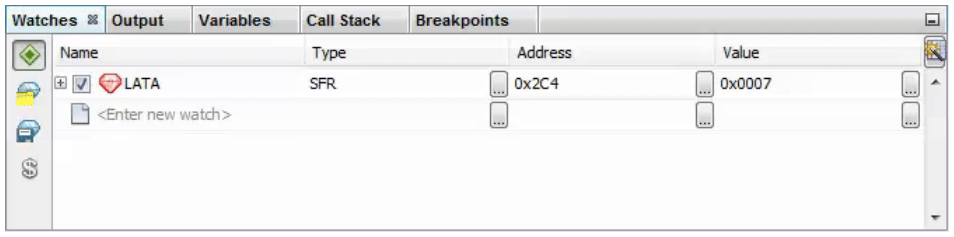
A runtime watch provides updating of a variable in the following windows during program execution instead of on halt:

- Watches – Window > Debugging menu
- Memory – Window > PIC Memory Views menu

To set up runtime watches:

1. Build the project (In the Projects window, right click on the project name and select “Build”). The project must be built to see the available symbols.
2. Select *Window>Debugging>Watches* to open the Watches window.
3. Right click in the window and select “New Runtime Watch”. Select the symbol or SFR you wish to watch in the New Run Time Watch window. Click **OK**.
4. Begin a debug session (*Debug>Debug Project*). Watch variable values change in the Watches window.
5. To see the watched variable change in a Memory window, Pause (halt) and open a Memory window containing the watched variable. Continue the program and watch the values change in this window.

FIGURE 6-1: RUNTIME WATCH



Name	Type	Address	Value
LATA	SFR	0x2C4	0x0007
<Enter new watch>			

6.3 INSTRUMENTED TRACE

This section discusses the available types of instrumented trace and how to use them.

6.3.1 Requirements for Trace

The following is required to use trace:

Devices that support trace – see the Development Tool Selector (DTS) on the Development Tools section of the Microchip web site.

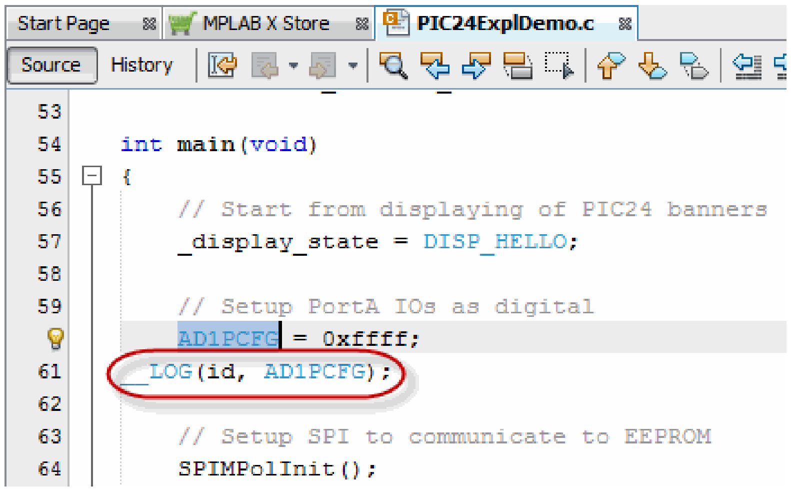
The following are limitations of trace:

- For 8- and 16-bit devices, in-line assembly code (assembly code within C code) cannot be traced.
- For additional limitations, see the Limitations section of MPLAB REAL ICE In-Circuit Emulator Help in MPLAB X IDE.

6.3.2 How Trace Works

Trace for the MPLAB REAL ICE in-circuit emulator (Instrumented Trace) is a solution for providing basic trace information. Through the use of `TRACE()` and `LOG()` macros, you can report program locations or variable values to MPLAB IDE while the application is running and examine them via the Trace window once the application halts. You may type these macro names in manually or right click in the editor and select the macro to be inserted from the context menu. To log a variable value, the variable should be highlighted before selecting from the context menu.

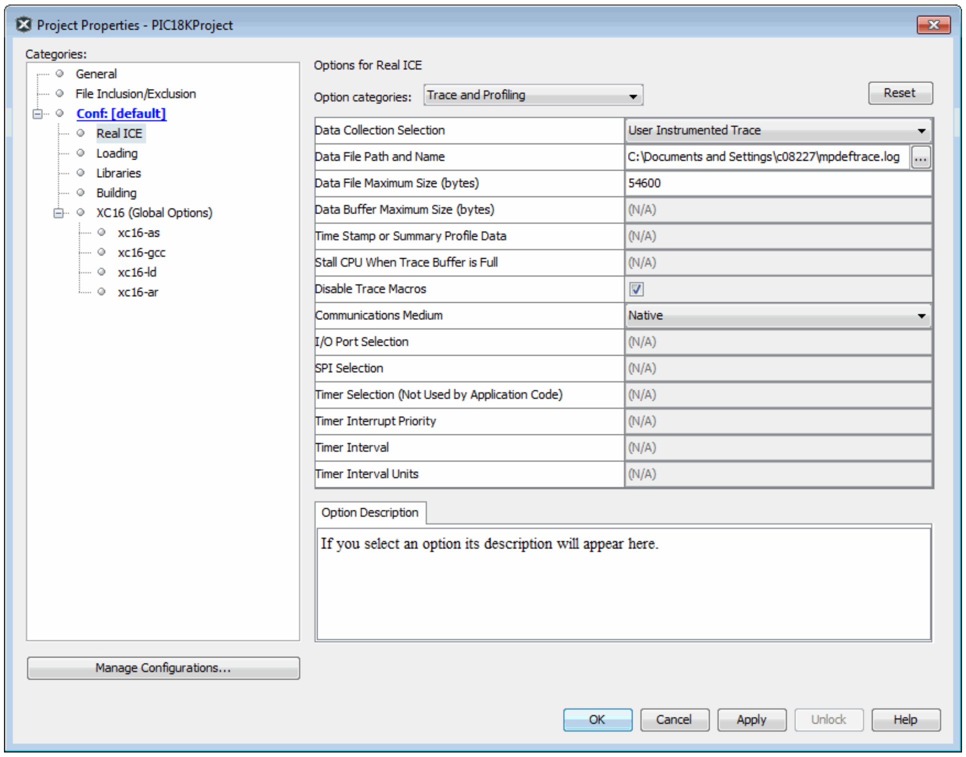
FIGURE 6-2: EXAMPLE OF INSERTED LOG MACRO



```
int main(void)
{
    // Start from displaying of PIC24 banners
    _display_state = DISP_HELLO;

    // Setup PortA IOs as digital
    AD1PCFG = 0xffff;
  __LOG(id, AD1PCFG);

    // Setup SPI to communicate to EEPROM
    SPIMPolInit();
```

There are three trace methods available at this time (see Section 6.3.3 “Types of Trace”). The methods are located on the Project Properties dialog, “REAL ICE” Category, “Trace and Profiling” Options Category page. The choices found under “Communications Medium” include “Native Trace” (utilizes PGC/PGD communication lines), “SPI Trace”, and “I/O Port Trace”. Not every method is available on every part; i.e., the options are device specific. The Instrumented Trace library supports C and assembly projects on PIC18F MCU devices; but, only C projects on 16-bit devices.

FIGURE 6-3: PROJECT PROPERTIES – NATIVE TRACE SELECTION



Option categories: Trace and Profiling

Data Collection Selection	User Instrumented Trace
Data File Path and Name	C:\Documents and Settings\c08227\mpdeftrace.log
Data File Maximum Size (bytes)	54600
Data Buffer Maximum Size (bytes)	(N/A)
Time Stamp or Summary Profile Data	(N/A)
Stall CPU When Trace Buffer is Full	(N/A)
Disable Trace Macros	✓
Communications Medium	Native
I/O Port Selection	(N/A)
SPI Selection	(N/A)
Timer Selection (Not Used by Application Code)	(N/A)
Timer Interrupt Priority	(N/A)
Timer Interval	(N/A)
Timer Interval Units	(N/A)

If you select an option its description will appear here.

The trace and log information transmitted is identical regardless of the trace method used. For `TRACE()`, a single value in the range of 64-127 is sent. A label generated using this number is automatically inserted into the code, so MPLAB IDE can identify in the trace buffer the location that sent the value. For `LOG()`, a two-byte header is sent, followed by the value of the variable being logged. The first byte indicates the variable type, and is a value between 0 and 63. The second byte indicates the location that sent the variable. Here, the location is represented by a value between 0 and 127. (See Section 6.3.9 “More on Trace/Log ID Numbers”.)

Interrupts are disabled during every `TRACE()` and `LOG()` call. This is to ensure that trace or log statements at an interrupt level do not interfere with a trace or log statement that may already be in progress at the application level. A similar argument holds for protecting statements within a low priority interrupt from being corrupted by those from a high priority interrupt.

6.3.3 Types of Trace

Currently there are three types of trace. All types are language-tool-version dependent, and stream data in real time to MPLAB IDE.

The pluses and minuses of using each trace type, as well as the type of communication available (standard and/or high-speed), are summarized below.

Type of Trace	Speed	Code Size Impact	Real Time Op	Pin Usage	Device Feature Needed	Communication	
						Std	**HS**
Native Trace	Fast[1]	Large	Close	None	Built-in debug	15 MIPS or less	Greater than 15 MIPS
SPI Trace	Faster[1]	Medium	Closer	SPI pins	SPI	No	Yes
I/O Port Trace	Fastest	Small	Closest	8-pin port	None	Yes[2]	Yes[2]

Note 1: For Native trace running at speeds higher than 15 MIPS and for SPI Trace capability, the Performance Pak may be needed. The actual cutoff speed may vary depending on layout, noise, and similar considerations.

2: Also requires connection from device port to emulator logic probe port.

6.3.3.1 NATIVE TRACE

Native trace can be used with either standard or high-speed communications, with no additional connections – the information is conveyed via the PGD/PGC/EMUC/EMUD pins. This two-wire interface uses the trace macro format (see Section 6.3.4 "Setting Up Trace in MPLAB X IDE").

For 8-bit and 16-bit devices, Native trace requires that you enter the clock speed (Project Properties dialog, REAL ICE category, Clock options category.)

If Native trace is used, then data captures cannot be used as these features use the same device resource. Breakpoints are still available; but, be aware that Native trace will use one breakpoint for tracing. This will NOT be reflected on the Device Debug Resource toolbar.

To use data capture triggers, you must disable Native trace (see Section 6.3.7 "Disabling Trace").

6.3.3.2 SPI TRACE

SPI trace can be used only with high-speed/LVDS communication hardware, purchased as the Performance Pak (AC244002). Trace clock and data are provided through pins 7 (DAT) and 8 (CLK) on the receiver board of the LVDS connection. The device does not have to be operating at high speeds to use this feature.

When you dedicate these pins to tracing, any multiplexed function on these pins cannot be used by the application.

For devices with remappable peripheral pins, be aware that the SPI trace macro does not touch any PPS register and does not need to know how the peripheral is mapped to a certain pin – it will write to the SPI1 or SPI2 selected in MPLAB X IDE.

SPI trace does require that you enter the clock speed (on the Properties dialog, REAL ICE category, and the Clock options category.)

For hardware connections, see the "*Performance Pak User's Guide*" (DS50002528).

The SPI interface uses the trace macro format (see Section 6.3.4 "Setting Up Trace in MPLAB X IDE").

6.3.3.3 I/O PORT TRACE

I/O Port trace can be used with either standard or high-speed communications. Trace clock and data are provided from a device 8-pin I/O port through the MPLAB REAL ICE in-circuit emulator logic probe connector.

The I/O port must have all 8 pins available for trace. The port must not be multiplexed with the currently-used PGC and PGD pins. Therefore, review the data sheet of the selected device to determine the uninitialized/default port pin states and change them as necessary.

For hardware connections, see Section 3.6.3 "I/O Port Trace Connections (Logic Port)".

The port interface uses the trace macro format (see Section 6.3.4 "Setting Up Trace in MPLAB X IDE").

6.3.4 Setting Up Trace in MPLAB X IDE

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

1. Right click on the project name and select "Properties". In the Project Properties dialog click on "Real ICE" under "Categories".
2. Under "Option categories", select "Clock". For data capture and trace, the emulator needs to know the instruction cycle speed.
3. Under "Option categories", select "Trace and Profiling".
4. Under "Data Collection Selection", choose "User Instrumented Trace".
5. Under "Communications Medium" choose either "Native", "I/O PORT" or "SPI" trace.
6. Set up any other trace-related options. (See Section 11.3.5 "Trace and Profiling".)
7. Click **OK**.

Next, enter trace macros into your application code.

- To record a PC location, click on or highlight a line of code and then right click to select "Insert *Language* Line Trace" from the pop-up menu, where *Language* can be either C or ASM. This causes the following macro line to be inserted above the selected line:

  ```
  __TRACE(id);
  ```

  where `id` is a line trace number auto-generated during the build. For more information, see Section 6.3.9 "More on Trace/Log ID Numbers".

  > **Note:** Inserting a macro into code may modify the logic flow of the program. Check that braces are present wherever they are necessary.

- The recording of a variable value is performed similarly. First, highlight the variable name, or expression. Then, right click to select "Log Selected *Language* Value" from the pop-up menu, where *Language* can be either C or ASM. This causes the following macro line to be inserted above the line containing the variable:

  ```
  __LOG(id,selected variable);
  ```

  where `id` is a log number that was auto-generated during build and *`selected variable`* is the highlighted variable. For more information, see Section 6.3.9 "More on Trace/Log ID Numbers".
- To remove a trace point, simply highlight and then delete the Trace/Log macro.

6.3.5 Running Trace

1. Debug Run (*Debug>Debug Project*) your application.
2. Pause the application.
3. View the trace data in the Trace window (*Window>Debugging>Trace*). For each `__TRACE` macro, the line of code following the macro will appear in the trace window each time it is passed. For each `__LOG` macro, the selected variable in the line of code following the macro will appear in the trace window each time it is passed.

> **Note:** To trace multiple lines of code or variables, you must place a macro before each line/variable that you wish to trace.

Repeat these steps each time you change a trace point.

6.3.6 Tracing Tips

When using `__TRACE` and `__LOG` macros in your code, consider the following:

- Focus on one area of an application and place `__TRACE` and `__LOG` macros so that they form a "flow" in the Trace window. That way, you can follow the execution flow and debug the application based on missing/incorrect trace points or an abrupt end to the trace flow.
- Use `__TRACE` and `__LOG` macros with conditional statements in your code to aid in debugging. Example: When a variable reaches a certain value, start logging it.

```
If(var > 5)
{
   __LOG(ID, var)
}
```

- Leave `__TRACE` and `__LOG` macros in your code for future debugging, if this is allowable. (For Project Properties dialog, REAL ICE Category, Trace Options Category page, select "Disable Trace Macros".)

6.3.7 Disabling Trace

To temporarily turn off trace data collection:

1. Select *File>Project Properties* dialog, Categories: REAL ICE, Options categories: Trace and Profiling.
2. Check "Disable Trace Macros".
3. Click **OK**.

To disable the full trace capability:

1. Remove all trace and log macros from code.
2. Select *File>Project Properties* dialog, Categories: REAL ICE, Options categories: Trace and Profiling.
3. Under "Data Collection Selection", choose "Off".
4. Click **OK**.

6.3.8 Resource Usage Examples

The following examples are for illustration only. Your results may vary based upon compiler/assembler version, command line options, MPLAB IDE version, size of data variable being logged, interrupt state, and device in use. All examples include argument setup, function call, and return time in their cycle counts.

The PIC18FXXJ MCU examples are compiled/assembled for non-priority interrupt usage (30 instructions). For priority interrupt usage, the value is 57; and for no interrupt usage, the value is 15.

The dsPIC33F DSC examples show nine instructions specified in the 16-bit library size for `memcpy()`.

EXAMPLE 6-1: PIC18FXXJ DEVICE RUNNING AT 4MHZ (1 MIPS) WITH ASSEMBLY PROJECT

	Native	SPI	I/O Port
Library Size (in instructions)	23 + 30	37 + 30	25 + 30
GPRs Used (in bytes)	8	6	6
`__TRACE(id)` instruction cycles	80	54	42
`__LOG(id, BYTE)` instruction cycles	168	90	57

EXAMPLE 6-2: PIC18FXXJ DEVICE RUNNING AT 40MHZ (10 MIPS) WITH C PROJECT

	Native	SPI	I/O Port
Library Size (in instructions)	75 + 30	87 + 30	112 + 30
GPRs Used (in bytes)	10	8	8
`__TRACE(id)` instruction cycles	79	71	55
`__LOG(id, INT)` instruction cycles	225	169	162

EXAMPLE 6-3: dsPIC33F DEVICE RUNNING AT 10 MIPS WITH C PROJECT

	Native	SPI	I/O Port
Library Size (in instructions)	87 + 9	92 + 9	93 + 9
GPRs Used (in bytes)	18	14	0
`__TRACE(id)` instruction cycles	80	53	32
`__LOG(id, INT)` instruction cycles	212	124	106

EXAMPLE 6-4: dsPIC33F DEVICE RUNNING AT 16 MIPS WITH C PROJECT

	Native	SPI	I/O Port
`__TRACE(id)` instruction cycles	88	53	32
`__LOG(id, INT)` instruction cycles	227	138	106

EXAMPLE 6-5: dsPIC33F DEVICE RUNNING AT 34 MIPS WITH C PROJECT

	Native	SPI	I/O Port
`__TRACE(id)` instruction cycles	100	53	32
`__LOG(id, INT)` instruction cycles	251	152	106

6.3.9 More on Trace/Log ID Numbers

MPLAB IDE will automatically generate the ID numbers required for a trace or log macro. However, to understand the method behind the numbering, read further.

You can have 64 trace points and 64 log points. These limits are determined by port trace (8 bits). Bit 7 is used as a clock and Bit 6 is used as a flag which indicates either a trace record (1) or a log record (0).

For a trace record, the low order bits represent the trace number (nnnnnn). You could say 0-63 are the legal trace numbers and require the trace flag be set, but it was just easier to combine the flag with the number and say the valid numbers are 64-127.

clock	1	n	n	n	n	n	n
bit 7	bit 6						bit 0

For a log record, the low order bits represent the log number (nnnnnnn).

clock	0	n	n	n	n	n	n
bit 7	bit 6						bit 0

6.3.10 Quick Trace Reference

If you are new to using the MPLAB REAL ICE in-circuit emulator trace feature, it is recommended that you read through the entire trace section for a full understanding.

Use this section as a quick reference for trace.

1. Select File>Project Properties dialog, Categories: REAL ICE, Options categories: Trace. Enable trace, choose the type of trace and set other trace options.
2. Select Window>Debugging>Trace to open the trace window in which to view trace data.
3. Right click in your code to enter trace macros (`__TRACE`, `__LOG`) as desired.
4. Debug Run your project.

6.4 PIC32 INSTRUCTION TRACE – PIC32 MCUs ONLY

This section will discuss trace for 32-bit devices and how to use it. Not all PIC32 devices have instruction trace so refer to your device data sheet.

- Requirements for Trace
- How Instruction Trace Works
- Setting Up and Using Trace
- Trace Hardware Specifications

6.4.1 Requirements for Trace

The following is required to use trace for 32-bit (PIC32) devices:

- MPLAB X IDE and a compatible 32-bit C compiler (e.g., MPLAB XC32 C compiler)
- PIC32 Plug-In Module (PIM) containing a device that supports trace and a trace port
- MPLAB® REAL ICE™ Trace Interface Kit (AC244006) containing a 12-inch trace cable and a trace adapter board

6.4.2 How Instruction Trace Works

PIC32 instruction trace uses a MIPS32 iFlowtrace™ mechanism, which is a non-intrusive hardware instruction trace. You can use this trace to capture every instruction executed by the device. The trace data is sent from the device using the pins TRCLK and TRD3:0 to the emulator. The emulator streams this data to a trace buffer on the PC that acts like a rolling FIFO.

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.

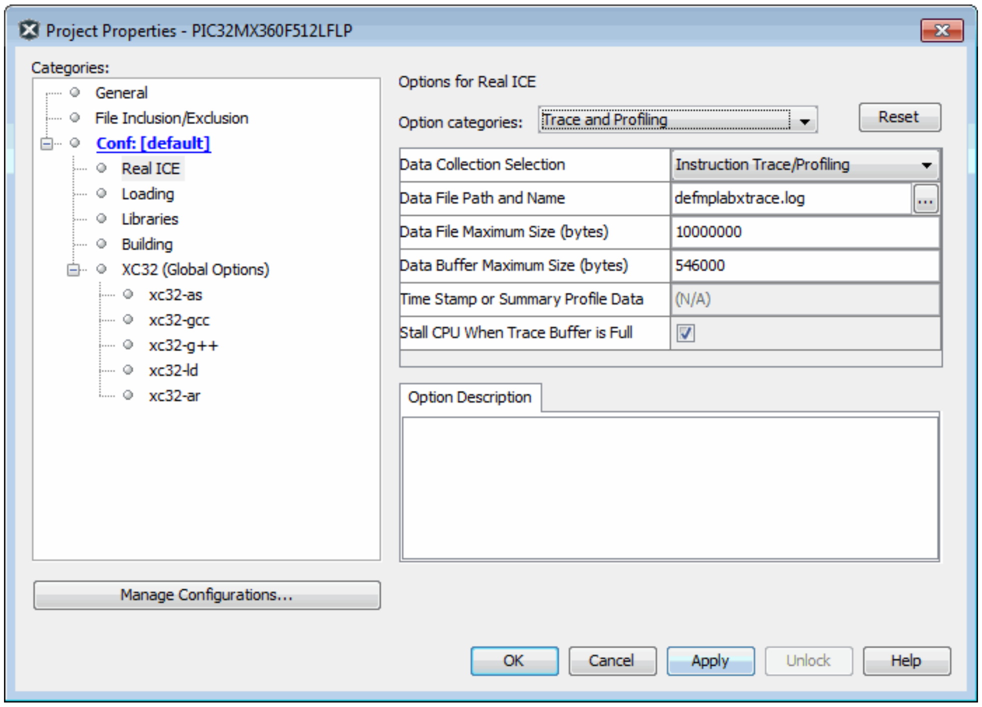
Enable and set trace options in the Project Properties dialog, Categories: REAL ICE, Option categories: Trace and Profiling. Here you may set:

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

The maximum off-chip (PC) trace buffer size is 22MB. Execution will not halt when this external buffer is full.

See also Section 11.3.5 “Trace and Profiling”.

FIGURE 6-4: PIC32 INSTRUCTION TRACE OPTIONS



Option categories: Trace and Profiling

Data Collection Selection	Instruction Trace/Profiling
Data File Path and Name	defmplabxtrace.log
Data File Maximum Size (bytes)	10000000
Data Buffer Maximum Size (bytes)	546000
Time Stamp or Summary Profile Data	(N/A)
Stall CPU When Trace Buffer is Full	✓

6.4.3 Setting Up and Using Trace

PIC32 Instruction Trace requires hardware and software setup before you can trace.

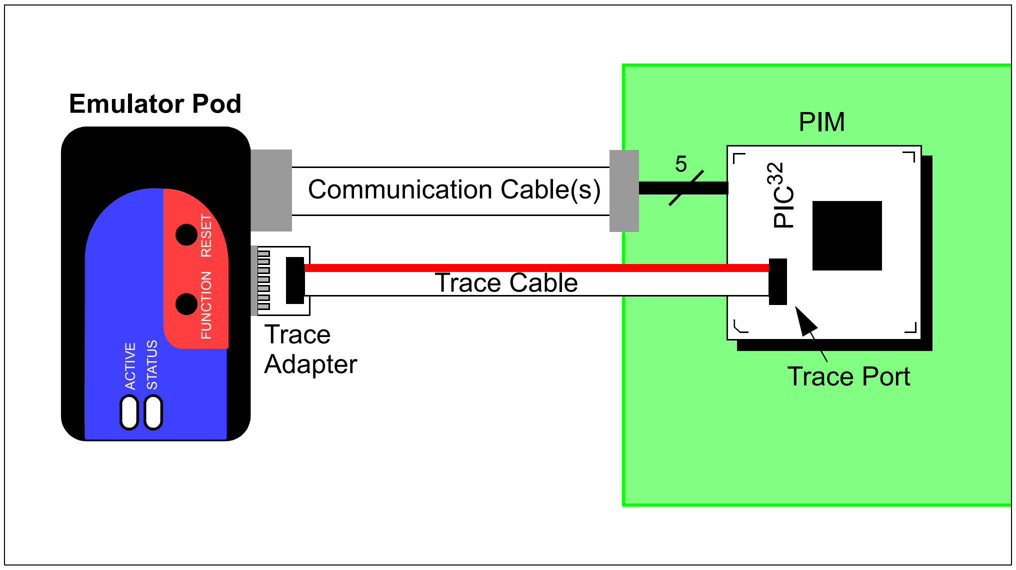
6.4.3.1 HARDWARE SETUP – PIC32 MCU ON PIM

To use the PIC32 Instruction Trace feature with the PIM do the following:

1. Plug the PIM into an **unpowered** target board.
2. Install communication cable(s) between the emulator and your target board. See Section 3.5 "Emulator Communication with the Target" or the "*Performance Pak User's Guide*" (DS50002528).
3. Connect the trace cable from the trace port on the PIM to the trace adapter board. Orient the cable as show in Figure 5. For hardware details, see the "*MPLAB REAL ICE Trace Interface Kit Specification*" (DS50002331).
4. Plug the trace adapter board into the MPLAB REAL ICE in-circuit emulator logic probe port. The top of the adapter board contains the connectors and should be oriented upwards when plugging the board into the logic probe port (Figure 5).
5. Power the target.

Note: When using trace, pins TRCLK and TRD3:0 are used. Therefore, you cannot use the other functions multiplexed on these pins.
For PIC32MX360F512L, multiplexed functions are RG14:12 and RA7:6. Check your device data sheet for details.

FIGURE 6-5: TRACE CONNECTION WITH PIM



6.4.3.2 HARDWARE SETUP – PIC32 MCU ON TARGET

When designing instruction trace capability onto your own board, the following provisions will need to be made.

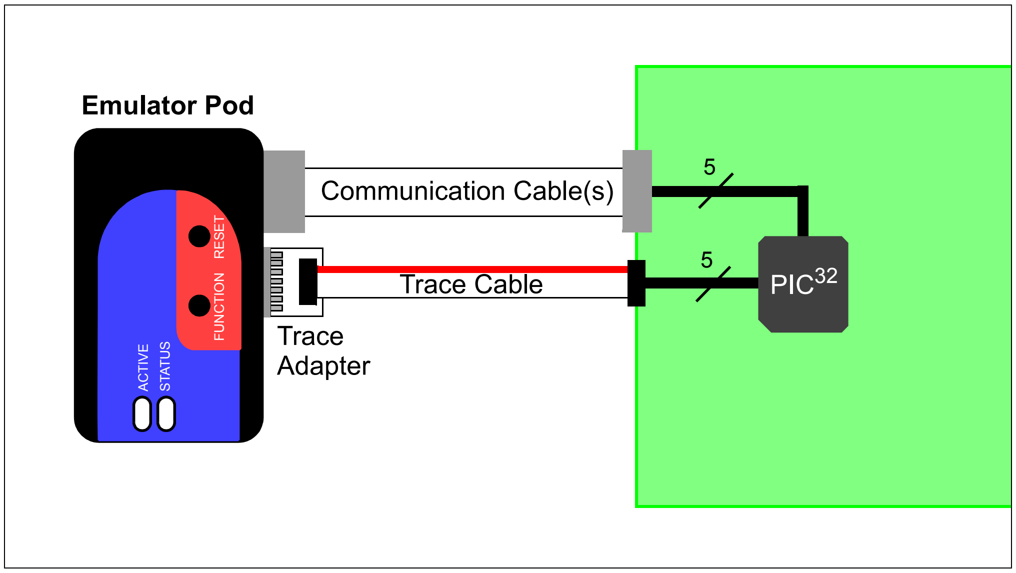
- Termination series resistors will need to be added. For details, see the "*MPLAB REAL ICE Trace Interface Kit Specification*" (DS50002331).
- Depending on your board routing and loading of the signals used for trace, it is a good idea to place 0 ohm resistors that can be unpopulated to isolate the trace signals TRCLK and TRD3:0.

To use the PIC32 Instruction Trace feature with your own board do the following:

1. The target board should initially be **unpowered**.
2. Install communication cable(s) between the emulator and your target board. See Section 3.5 "Emulator Communication with the Target" or the "*Performance Pak User's Guide*" (DS50002528).
3. Connect the trace cable from the target board to the trace adapter board. Make sure the PIC32 MCU on your target board is connected to accommodate trace, as per Figure 6-6. For hardware details, see "*MPLAB REAL ICE Trace Interface Kit Specification*" (DS50002331).
4. Plug the trace adapter board into the MPLAB REAL ICE in-circuit emulator logic probe port. The top of the adapter board contains the connectors and should be oriented upwards when plugging the board into the logic probe port (Figure 6-6).
5. Power the target.

Note: When using trace, pins TRCLK and TRD3:0 are used. So, you cannot use the other functions multiplexed on these pins. For PIC32MX360F512L, multiplexed functions are RG14:12 and RA7:6.

FIGURE 6-6: TRACE CONNECTION WITH DEVICE ON TARGET



6.4.3.3 MPLAB X IDE SETUP

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

1. Right click on the project name and select “Properties”. In the Project Properties dialog, click on “Real ICE” (under “Categories”).
2. Under “Option categories”, select “Trace and Profiling”.
3. Under “Data Collection Selection”, choose “Instruction Trace/Profiling”.
4. Set up any other trace-related options. (See Section 11.3.5 “Trace and Profiling”.)
5. 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.

6.4.3.4 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*).

6.4.4 Trace Hardware Specifications

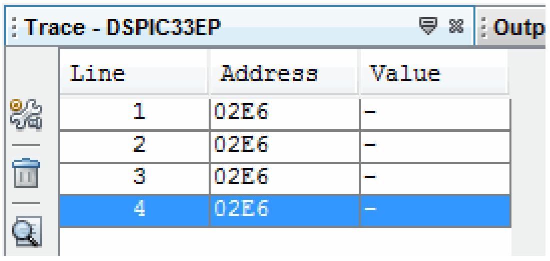
Specifications for hardware that supports PIC32 Instruction Trace may be found in “*MPLAB REAL ICE Trace Interface Kit Specification*” (DS50002331) and on the web pages of some PIC32 PIMs.

6.5 JUMP TRACE – EP DEVICES ONLY

Jump trace is a feature on some (EP) devices that records up-to-the-last four instruction flow changes in the code, allowing you to navigate back up a call chain. Branch/jump addresses are displayed in the Trace window (up to four entries).

This feature is selected in the Project Properties window, Real ICE category, "Trace and Profiling" option category, under "Data Collection Selection".

FIGURE 6-7: JUMP TRACE IN TRACE WINDOW



Line	Address	Value
1	02E6	–
2	02E6	–
3	02E6	–
4	02E6	–

6.6 PC SAMPLING – 8-BIT AND 16-BIT MCUs ONLY

PC sampling is a method of examining C code to determine the percentage of time that is spent in each function. This information can show you where your program time is being spent so you can work to optimize your code.

For PC sampling, a device timer is set up to take samples of program execution and display the results in the PC profiling window.

PC profiling is similar to PC sampling. For details see Section 6.7 “PC Profiling – 32-Bit MCUs Only”.

6.6.1 Requirements

Currently, to use PC sampling, your project must be set up for:

- a supported device:
  (Note: PIC16F1 devices are NOT supported)
  - PIC18F with data capture – To find out if your device has data capture, please see the online Development Tool Selector: http://www.microchip.com/dts.
  - PIC24F, PIC24EP
  - dsPIC33FJ, dsPIC33E
  -
- an MPLAB XC8 or MPLAB XC16 C Compiler version 1.10 or above

6.6.2 Clock Setup

To set up the clock:

1. Open the Project properties window (*File>Project Properties*).
2. Click on “REAL ICE” under “Categories” and select “Clock” from the “Options categories” drop-down box. Ensure this value matches the actual target speed, i.e., as set by the Configuration bits in code.
3. Click **Apply**.

6.6.3 Sampling Setup

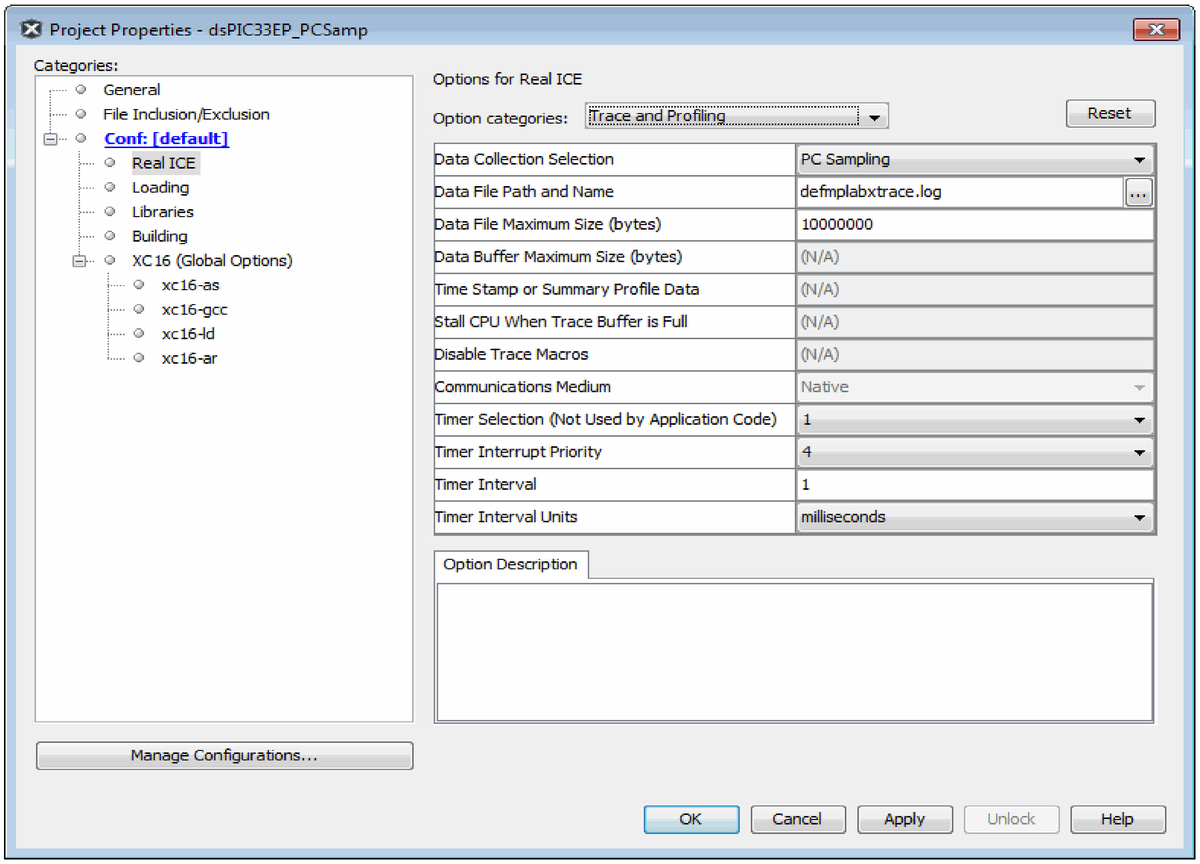
To set up sampling:

1. Click on “REAL ICE” under “Categories” and select “Trace and Profiling” from the “Options categories” drop-down box.
2. Under “Data Collection Selection”, select “PC Sampling”.
3. Set up your data file and timer in this window. For reference, see Section 11.3.5 “Trace and Profiling”.

Note: Ensure the timer you select is not used in your program.

4. Click **OK**.

FIGURE 6-8: PC SAMPLING – SELECTION AND SETUP



Option categories: Trace and Profiling

Data Collection Selection	PC Sampling
Data File Path and Name	defmplabxtrace.log
Data File Maximum Size (bytes)	10000000
Data Buffer Maximum Size (bytes)	(N/A)
Time Stamp or Summary Profile Data	(N/A)
Stall CPU When Trace Buffer is Full	(N/A)
Disable Trace Macros	(N/A)
Communications Medium	Native
Timer Selection (Not Used by Application Code)	1
Timer Interrupt Priority	4
Timer Interval	1
Timer Interval Units	milliseconds

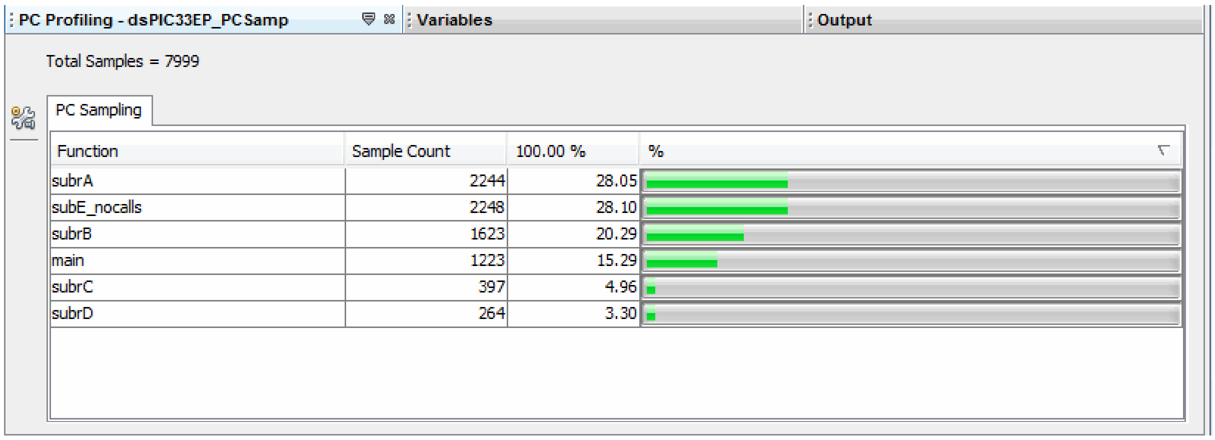
6.6.4 Operation

To generate data:

1. Select *Window>Debugging>PC Profiling*. This will open the PC Profiling window.
2. Run your code and then halt.
3. View the sampling data in the window. Data is only displayed on halt.
4. Right click in the window to pop up a menu to either clear the data or reload the data.

You may also display this data in the Code Profiling plugin, available for purchase at http://www.embeddedcodesource.com

FIGURE 6-9: PC SAMPLING – PC PROFILING WINDOW



Total Samples = 7999

Function	Sample Count	100.00 %	%
subrA	2244	28.05	
subE_nocalls	2248	28.10	
subrB	1623	20.29	
main	1223	15.29	
subrC	397	4.96	
subrD	264	3.30	

6.7 PC PROFILING – 32-BIT MCUs ONLY

PC profiling is a method for examining C code to determine the percentage of time that is spent in each function. This information can show you where your program time is being spent so you can work to optimize your code.

For PC profiling, every program counter (PC) trace sample is taken from the trace (data) buffer and profiled. This data is displayed in the PC Profiling window, as shown in Figure 6-10. In this example, the trace buffer contains 21,000 PC sample data points. Of these, 7859 (or 37.42%) were associated with the `main()` function, 6023 (or 28.68%) with the `subrA()` function, etc.

PC profiling is similar to PC sampling. For details see Section 6.6 "PC Sampling – 8-Bit and 16-Bit MCUs Only".

6.7.1 Requirements

Currently, to use PC profiling your project must be set up for:

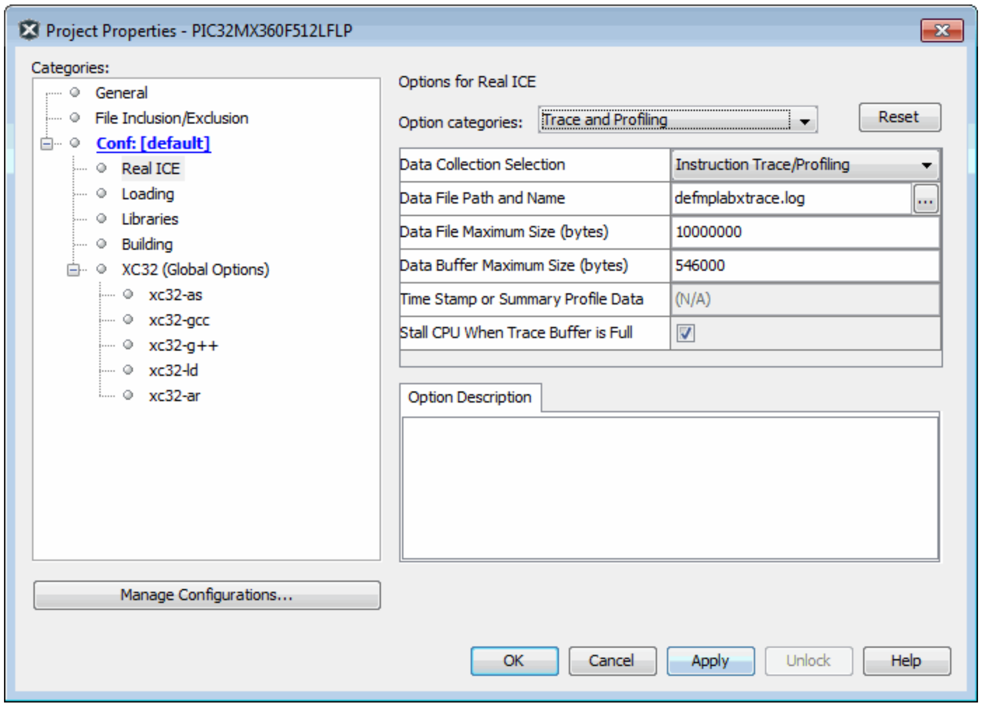
- a supported device:
  - PIC32MX with data capture – To find out if your device has data capture, please see the online Development Tool Selector: http://www.microchip.com/dts.
  - **Note:** PIC32MZ devices are NOT supported

6.7.2 Profiling Setup

To set up profiling:

1. Set up your hardware for PIC32 Instruction trace (see Section 6.4 "PIC32 Instruction Trace – PIC32 MCUs Only".)
1. Open the Project properties window (*File>Project Properties*).
2. Click on "REAL ICE" under "Categories" and select "Trace and Profiling" from the "Options categories" drop-down box.
3. Under "Data Collection Selection", select "Instruction Trace/Profiling".
4. Set up your data file in this window. For reference, see Section 11.3.5 "Trace and Profiling". Then click **OK**.

FIGURE 6-10: PC PROFILING – SELECTION AND SETUP



Data Collection Selection	Instruction Trace/Profiling
Data File Path and Name	defmplabxtrace.log
Data File Maximum Size (bytes)	10000000
Data Buffer Maximum Size (bytes)	546000
Time Stamp or Summary Profile Data	(N/A)
Stall CPU When Trace Buffer is Full	✓

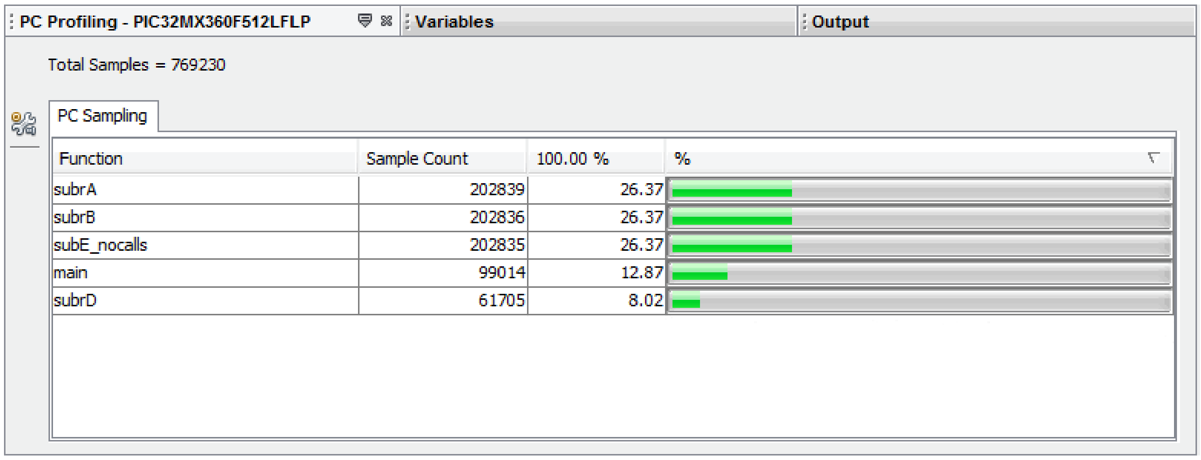
6.7.3 Operation

To generate data:

1. Select *Window>Debugging>PC Profiling*. This will open the PC Profiling window.
2. Run your code and then pause/halt.
3. View the profiling data in the window. Data is only displayed on halt.
4. Right click in the window to pop up a menu to either clear the data or reload the data.

You may also display this data in the Code Profiling plugin, available for purchase at http://www.embeddedcodesource.com

FIGURE 6-11: PC PROFILING WINDOW



Total Samples = 769230

Function	Sample Count	100.00 %	%
subrA	202839	26.37	
subrB	202836	26.37	
subE_nocalls	202835	26.37	
main	99014	12.87	
subrD	61705	8.02	

6.8 FUNCTION LEVEL PROFILING

Function Level Profiling (FLP) is a method for examining C code to determine the percentage of time (summary on halt of streaming data) that is being spent in each function. This information can show you where your program time is being spent so you may work to optimize your code.

6.8.1 Requirements

Currently, to use FLP your project must be set up for:

- a **device that supports data capture**:
  - PIC18F with data capture – To find out if your device has data capture, please see the online Development Tool Selector: http://www.microchip.com/dts.
  - PIC24F, PIC24EP
  - dsPIC33FJ, dsPIC33E
  - PIC32MX with data capture – To find out if your device has data capture, please see the online Development Tool Selector: http://www.microchip.com/dts.
  - **Note:** PIC16F1 devices are NOT supported
  - **Note:** PIC32MZ devices are NOT supported
- an **MPLAB XC C Compiler version 1.20 or above**
- the **Code Profiling Plugin** (optional purchase)
  - the plug-in may be purchased at: http://www.embeddedcodesource.com
  - Install the plug-in (called Code Profiling Viewer) by following the instructions in MPLAB X IDE Help, “Additional Tasks”, “Add Plug-In Tools”.
  - Once the plugin is installed, the “Code Profiling” window then may be accessed under *Tools>Embedded>CodeProfiling*.

6.8.2 Clock Setup (8- and 16-Bit MCUs Only)

To set up the clock:

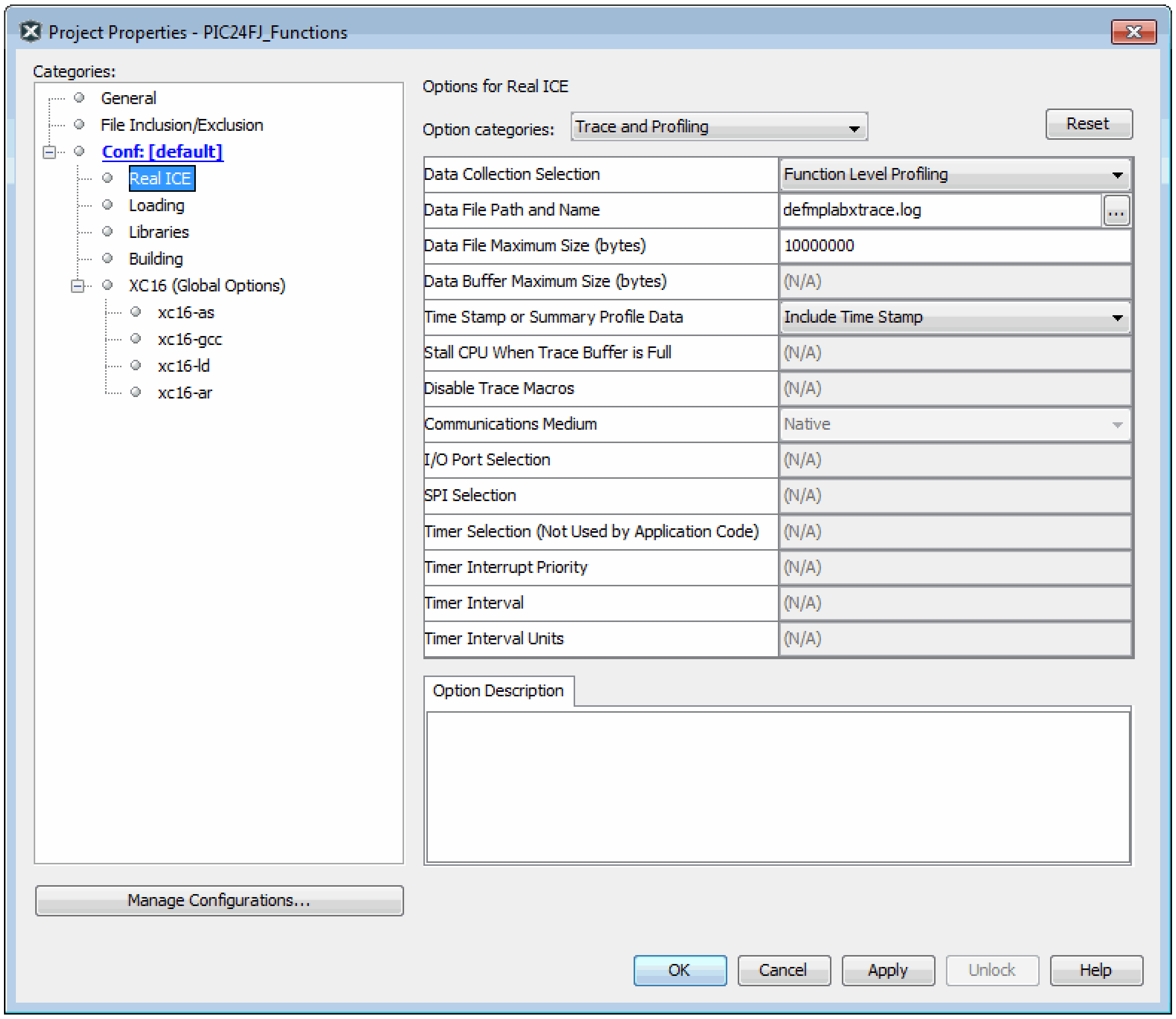
1. Open the Project properties window (*File>Project Properties*).
2. Click on “REAL ICE” under “Categories” and select “Clock” from the “Options categories” drop-down box. Ensure this value matches the actual target speed, i.e., as set by the Configuration bits in code.
3. Click **Apply**.

6.8.3 Sampling Setup

To set up sampling (Figure 6-12):

1. Click on “REAL ICE” under “Categories” and select “Trace and Profiling” from the “Options categories” drop-down box.
2. Under “Data Collection Selection”, select “Function Level Profiling”.
3. Set up your data file information, i.e., Data File Path and Name and Data File Maximum Size. For details see Section 11.3.5 “Trace and Profiling”.
4. Select the type of Function Level Profiling:
   a) Include Time Stamp: Time stamp data used
   b) Summary Profile Data Only: Summary data used
5. Click **OK**.

FIGURE 6-12: FUNCTION LEVEL PROFILING – SELECTION AND SETUP



Option categories: Trace and Profiling

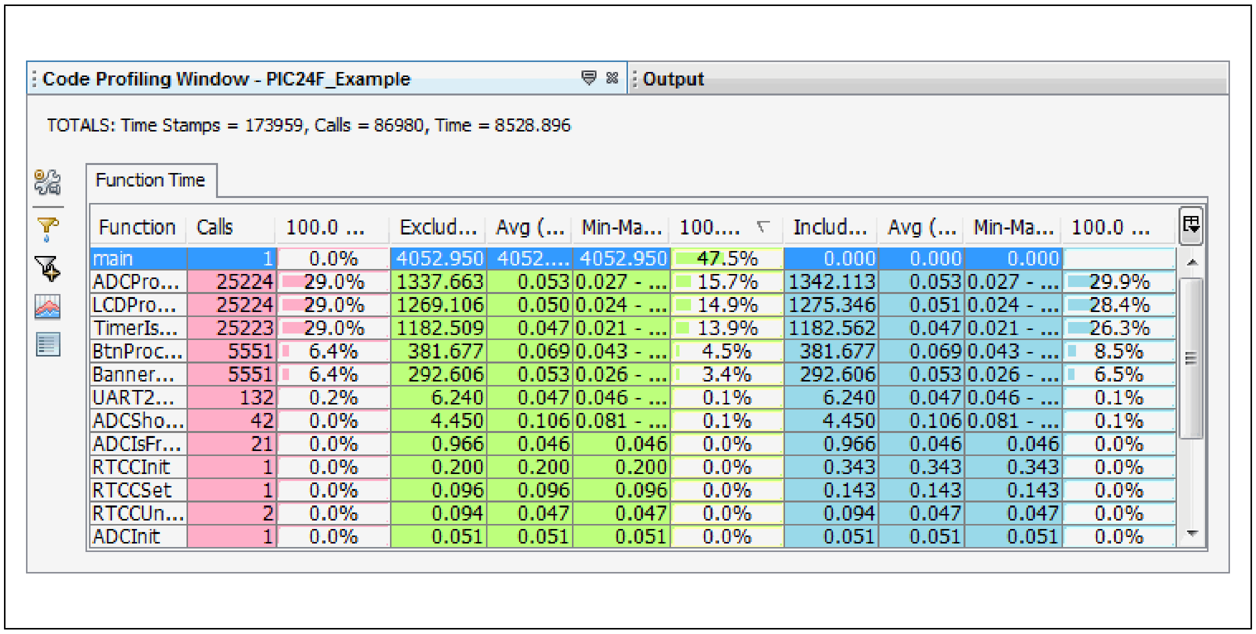
Data Collection Selection	Function Level Profiling
Data File Path and Name	defmplabxtrace.log
Data File Maximum Size (bytes)	10000000
Data Buffer Maximum Size (bytes)	(N/A)
Time Stamp or Summary Profile Data	Include Time Stamp
Stall CPU When Trace Buffer is Full	(N/A)
Disable Trace Macros	(N/A)
Communications Medium	Native
I/O Port Selection	(N/A)
SPI Selection	(N/A)
Timer Selection (Not Used by Application Code)	(N/A)
Timer Interrupt Priority	(N/A)
Timer Interval	(N/A)
Timer Interval Units	(N/A)

6.8.4 Operation

To generate data:

1. Debug Run and then Pause your program to collect data into a file.
2. Depending on whether or not you have the Code Profiling Plugin:
   a) No plugin: no IDE display of data available.
   b) Plugin available: Time Stamped (Figure 6-13) or Summary data will be displayed in the Plugin window. A graph is also available. See the plugin help file for details.

FIGURE 6-13: CODE PROFILING PLUGIN WINDOW



Code Profiling Window - PIC24F_Example | Output

TOTALS: Time Stamps = 173959, Calls = 86980, Time = 8528.896

Function Time

Function	Calls	100.0 ...	Exclud...	Avg (...	Min-Ma...	100....	Includ...	Avg (...	Min-Ma...	100.0 ...
main	1	0.0%	4052.950	4052....	4052.950	47.5%	0.000	0.000	0.000	
ADCPro...	25224	29.0%	1337.663	0.053	0.027 - ...	15.7%	1342.113	0.053	0.027 - ...	29.9%
LCDPro...	25224	29.0%	1269.106	0.050	0.024 - ...	14.9%	1275.346	0.051	0.024 - ...	28.4%
TimerIs...	25223	29.0%	1182.509	0.047	0.021 - ...	13.9%	1182.562	0.047	0.021 - ...	26.3%
BtnProc...	5551	6.4%	381.677	0.069	0.043 - ...	4.5%	381.677	0.069	0.043 - ...	8.5%
Banner...	5551	6.4%	292.606	0.053	0.026 - ...	3.4%	292.606	0.053	0.026 - ...	6.5%
UART2...	132	0.2%	6.240	0.047	0.046 - ...	0.1%	6.240	0.047	0.046 - ...	0.1%
ADCSho...	42	0.0%	4.450	0.106	0.081 - ...	0.1%	4.450	0.106	0.081 - ...	0.1%
ADCIsFr...	21	0.0%	0.966	0.046	0.046	0.0%	0.966	0.046	0.046	0.0%
RTCCInit	1	0.0%	0.200	0.200	0.200	0.0%	0.343	0.343	0.343	0.0%
RTCCSet	1	0.0%	0.096	0.096	0.096	0.0%	0.143	0.143	0.143	0.0%
RTCCUn...	2	0.0%	0.094	0.047	0.047	0.0%	0.094	0.047	0.047	0.0%
ADCInit	1	0.0%	0.051	0.051	0.051	0.0%	0.051	0.051	0.051	0.0%

6.9 APPLICATION IN/OUT

> **Note:** This window is only available for devices that support the application in/out function used with the MPLAB ICD 3 or MPLAB REAL ICE in-circuit emulator. For details, see online help, “Device and Feature Support”. The Application In/Out function cannot be used with PIC32 instruction trace.

The Application In/Out window allows you to interact with a running application using the Application Input/Output (App I/O) feature supported on some devices. Using this feature, data may be sent serially to and from the target application and, during runtime, over the same PGC/PGD lines used for debugging. Data written to the APPOUT register will be displayed to the window, while user input will be sent to the target application and available in the APPIN register.

6.9.1 Using App I/O with 8- and 16-Bit MCUs

To use the Application In/Out window:

- To directly write to App I/O, use the device-specific header file when building your application (per Example 6-6.)
- In the future, the 16-bit device library will have `printf()` and `scanf()` functions for use with the Application In/Out feature.

EXAMPLE 6-6: APPLICATION IN/OUT REGISTER USAGE

The following application code reads the application input register and writes out a padded value of the input to the application output register.

```
// include file
#include <p33Exxxx.h>

// set up config bits
_FWDT( FWDTEN_OFF )

// initialize variables
unsigned int val;
int i;
unsigned int oval;

int main(void)
{
   while(1)
   {
    if(APPSbits.APIFUL) // APPI is full?
      {
       val = _APPIN;          // Read User Input
       for(i=0; i<4; i++)
         {
          while(APPSbits.AROFUL); // APPO is full?
          oval =  val&0xFF;
          if(oval < 0x20)
             oval = 0x20;
          oval |= 0x20202000;
          _APPO = oval;                 // Send to MPLAB X IDE
          val >>= 8;
         }
      }
   }
}
```

6.9.2 Using App I/O with 32-Bit MCUs

To use the Application In/Out window:

- Use the device-specific header file when building your application to use the macros assigned in the header (per Example 6-6.)
- Alternatively, the PIC32 MCU library has `printf()` and `scanf()` functions for use with the Application In/Out feature. Refer to the "PIC32 Debug-Support Library" chapter in *32-Bit Language Tools Libraries* (DS51685).

EXAMPLE 6-7: APPLICATION IN/OUT MACRO USAGE

The following application code reads the application input register and writes out a padded value of the input to the application output register.

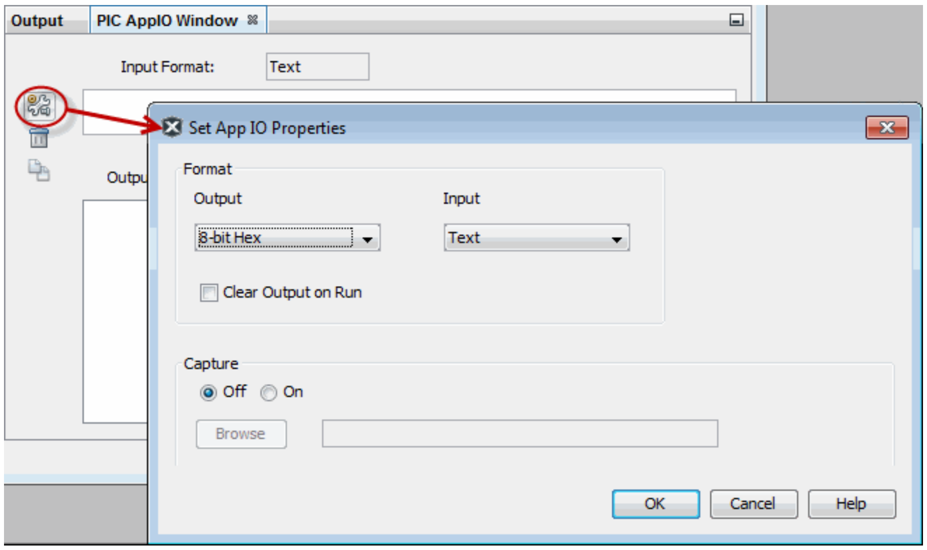
```
// include file
#include <p32xxxx.h>

// set up config bits
#pragma config FWDTEN = OFF

// initialize variables
unsigned int val;
int i;
unsigned int oval;

int main(void)
{
   while(1)
   {
    if(_DDPSTATbits.APIFUL) // APPI is full?
      {
       val = _APPI;         // Read User Input
       for(i=0; i<4; i++)
         {
          while(_DDPSTATbits.APOFUL); // APPO is full?
          oval =  val&0xFF;
          if(oval < 0x20)
             oval = 0x20;
          oval |= 0x20202000;
          _APPO = oval;               // Send to MPLAB X IDE
          val >>= 8;
         }
      }
   }
}
```

6.9.3 Running the Application

To run the example application:

1. Create a project using the Project wizard:
   a) Select a supported device.
   b) Select “REAL ICE” or “ICD 3” as the hardware tool.
   c) Use an MPLAB XC C Compiler as the language toolsuite.
2. Select *File>New File* to create and name a new C Source File. In the Editor window that opens, add the previous code and save.
3. Right click on the project name in the Projects window and select “Properties”. In the Project Properties window, click on “xc*nn*-gcc” under “Categories”. Under “Option Categories”, select “General” and check the “Enable App IO” checkbox.
4. Build the project (Right click on the project name and select “Build”).
5. Select *Window>Debugging>PIC App IO* to open the Application In/Out window.
6. Click on the “Properties” button (on the left of the window) to launch the Set App IO Properties dialog. Under “Capture” click “On”.
7. Debug Run or Run the project.
8. Enter a text value in the Input text box. Press Enter.
9. View the output in the Output text box.

FIGURE 6-14: SET APP I/O PROPERTIES

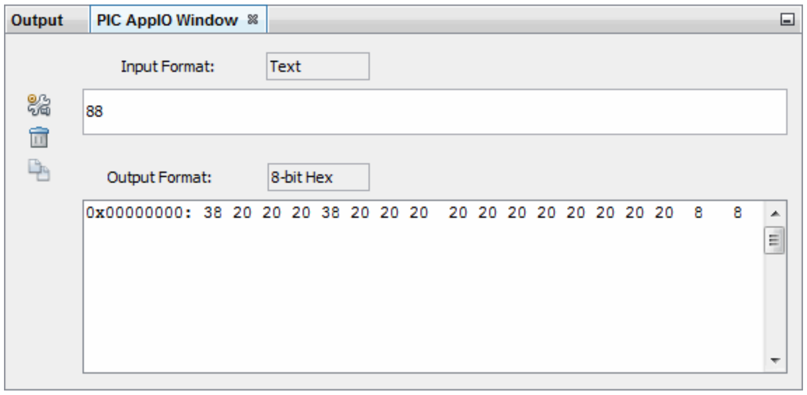


Input Format: Text

Output: 8-bit Hex
Input: Text
[ ] Clear Output on Run

Capture: [✓ Off] On

FIGURE 6-15: APP IN/OUT WINDOW



Input Format: Text

88

Output Format: 8-bit Hex

```
0x00000000: 38 20 20 20 38 20 20 20  20 20 20 20 20 20 20 20  8   8
```

Change Input and Output Formats by opening Set App IO Properties dialog and running the program again to see what outputs result for different inputs.

For more on options available in this window, see Section 11.4.2 "Application In/Out Window and Related Dialogs".

6.10 EXTERNAL TRIGGERS

Use external triggers to set up hardware triggers using the logic probe cables. External triggers can be used as input to halt program execution when an external event occurs. They can also be used as output to control external devices.

You will not be able to use external triggers if you are using the emulator logic probe port for another debug feature such as:

- I/O Port trace (see Section 6.3.3.3 "I/O Port Trace".)
- PIC32 Instruction trace (see Section 6.4 "PIC32 Instruction Trace – PIC32 MCUs Only".)

6.10.1 Hardware Connections

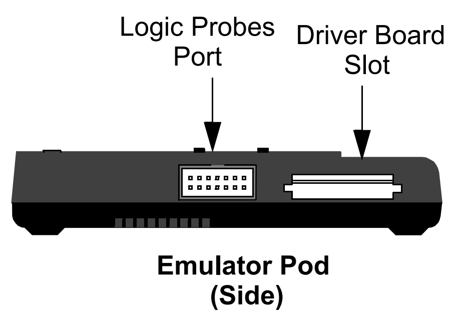
To connect the logic probe cables:

1. Plug the connector on the logic probes bundle into the logic probes port on the emulator (Figure 6-16). If you do not have the logic probes that came with your emulator, you can order them at Microchip Direct (ACICE0104).
2. Connect probes to your target. All pins, used or unused, should be pulled up or be grounded. Floating pins may produce false triggers.

To use probe pins as inputs, you must provide the circuitry to drive them (see Table 12-3: "Logic Probe Electrical Specifications" for drive levels.)

For more on external trigger hardware, see Section 12.4.4 "Logic Probe/External Trigger Interface".

FIGURE 6-16: LOGIC PROBE CONNECTOR



6.10.2 Software Setup

To select functions on logic probe port pins:

1. Right click on your MPLAB X IDE project and select "Properties".
2. Click on the "Real ICE" under "Categories" and select the "External Triggers" option category.
3. Use the drop down list next to a Trigger to set up the function. See also Section 11.3.8 "External Triggers".

6.11 ADDITIONAL DEBUG FEATURES

For a complete list of debug functions, windows and dialogs, see Chapter 11. "Emulator Function Summary".

Part 3 – Troubleshooting

NOTES:

Chapter 7. Troubleshooting First Steps

7.1 INTRODUCTION

If you are having problems with MPLAB REAL ICE in-circuit emulator operation, start here.

- The 5 Questions to Answer First
- Top Reasons Why You Can't Debug
- Other Things to Consider

7.2 THE 5 QUESTIONS TO ANSWER FIRST

1. What device are you working with? Often an upgrade to a newer version of MPLAB X IDE is required to support newer devices.
2. Are you using a Microchip demo board or one of your own design? Have you followed the guidelines for resistors/capacitors for communications connections? See Chapter 3. "Operation".
3. Have you powered the target? The emulator cannot power the target.
4. Are you using a USB hub in your set up? Is it powered? If you continue to have problems, try using the emulator without the hub (plugged directly into the PC.)
5. Are you using the standard communication cable (RJ-11) shipped with emulator? If you have made a longer cable, it may have communications errors. If longer cables are required, you should consider using high-speed communications. See Section 3.5.2 "High-Speed/LVDS Communication (Performance Pak)".

7.3 TOP REASONS WHY YOU CAN'T DEBUG

1. **Oscillator not working**. Check your Configuration bits setting for the oscillator. If you are using an external oscillator, try using an internal oscillator. If you are using an internal a PLL, make sure your PLL settings are correct.
2. **No power to the target board**. Check the power cable connection.
3. **Incorrect VDD voltage**. The VDD voltage is outside the specifications for this device. See the device programming specification for details.
4. **Physical disconnect**. The emulator has somehow become physically disconnected from the PC and/or the target board. Check the communications cables' connections.
5. **Communications lost**. Emulator to PC communications has somehow been interrupted. Reconnect to the emulator in MPLAB IDE.
6. **Device not seated**. The device is not properly seated on the target board. If the emulator is properly connected and the target board is powered, but the device is absent or not plugged in completely, you may get the message:
   Target Device ID (0x0) does not match expected Device ID (0x%x)
   where %x is the expected device ID.

7. **Device is code-protected**. Check your Configuration bits setting for code protection.
8. **No device debug circuitry**. You are trying to debug a production device that does not have debugging capabilities. Use a debug header instead. (See the "*Processor Extension Pak and Debug Header Specification*" (DS51292) in **"**Recommended Reading - Emulator**"**.)
9. **Application code corrupted**. The target application has somehow become corrupted or contains errors. Try rebuilding and reprogramming the target application. Then initiate a Power-On-Reset of the target.
10. **Application code runs but does not debug.** If your application debug builds and runs, but does not debug (no halt arrow in gutter, broken breakpoint, etc.), the source files may not be able to be located. Check your path and file names for non-ANSI characters or a length that is too long (See MPLAB X IDE documentation for path, file and folder restrictions).
11. **Incorrect programming pins**. You do not have the correct PGC/PGD pin pairs programmed in your Configuration bits (for devices with multiple PGC/PGD pin pairs).
12. **Additional setup required.** Other configuration settings are interfering with debugging. Any configuration setting that would prevent the target from executing code will also prevent the emulator from putting the code into Debug mode.
13. **Incorrect brown-out voltage**. Brown-out Detect voltage is greater than the operating voltage VDD. This means the device is in Reset and cannot be debugged.
14. **Incorrect connections**. You have not followed the guidelines in Chapter 3. "Operation".

7.4 OTHER THINGS TO CONSIDER

1. Use the Loopback Test board to verify that the emulator itself is functioning correctly (Section 12.7 "Loopback Test Board".)
2. There may be a problem programming in general. As a test, switch to Run mode (*Run>Run Project*) and program the target with the simplest application possible (e.g., a program to blink an LED). If the program will not run, then you know that something is wrong with the target setup.
3. It is possible that the target device has been damaged in some way (e.g., over current). Development environments are notoriously hostile to components. Consider trying another target device.
4. Review emulator debug operation to ensure proper application setup (see Chapter 3. "Operation".)
5. If the problem persists contact Microchip.

Chapter 8. Frequently Asked Questions (FAQ)

8.1 INTRODUCTION

Look here for answers to frequently asked questions about the MPLAB REAL ICE in-circuit emulator system.

- Emulator Operation
- Instrumented Trace Operation
- General Issues

8.2 EMULATOR OPERATION

The questions below concern emulator operation.

8.2.1 What's in the device that allows it to communicate with the MPLAB REAL ICE in-circuit emulator?

Most silicon contains a debug module that can communicate with the emulator. A small debug executive program also needs to be programmed in the high area of memory (sometimes in a separate area of memory called test memory). On some small devices without on-chip debugging capability, a Processor Extension Pak or an Emulation Extension Pak may be purchased to allow debugging.

8.2.2 How is the throughput of the processor affected by having to run the debug executive?

The debug executive doesn't run while code is running, so there is no throughput reduction, i.e., the emulator doesn't 'steal' any cycles from the target device.

8.2.3 How does the MPLAB REAL ICE in-circuit emulator compare with other in-circuit emulators/debuggers?

Please refer to Section 3.2 "Tools Comparison".

8.2.4 How does MPLAB X IDE interface with the MPLAB REAL ICE in-circuit emulator to allow more features than in-circuit debuggers?

For some devices, the MPLAB REAL ICE in-circuit emulator communicates using the debug executive located in a special area of memory that does not use application program memory. Also, the debug exec is streamlined for more efficient communication. The emulator contains an FPGA, large SRAM Buffers (1Mx8), and a high speed USB interface. The program memory image is downloaded and is contained in the SRAM to allow faster programming. The FPGA in the emulator serves as an accelerator for interfacing with the device in-circuit debugger modules.

8.2.5 Does the MPLAB REAL ICE in-circuit emulator have complex breakpoints?

Yes. You can break based on pass counts, program memory execution at a specific address or data memory reads/writes at a specific address. You can also do event breakpoints (break on an event) sequenced breakpoints (break after events occur in a specific order), or ANDed breakpoint (break when events occur all together).

8.2.6 One of the probe pins is labeled 5V. How much drive capability does this probe have?

This pin provides a monitoring function (allows you to see what VDD is actually being applied and used on the driver buffers). Attaching any circuitry to this pin can affect emulator operation.

The MPLAB REAL ICE in-circuit emulator cannot provide power to the target, unless you are using the Power Monitor (AC244008). See the "*MPLAB REAL ICE Power Monitor User's Guide*" (DS50002532).

8.2.7 Are any of the driver boards opto-isolated or electrically isolated?

They are DC opto-isolated. To isolate against AC voltage (120V), you will need the MPLAB® REAL ICE™ Isolator Unit (AC244005). See the "*MPLAB REAL ICE Isolator Unit Specification*" (DS50002529).

8.2.8 What precautions are there with the programming clock (PGC) or programming data (PGD) pins?

The following are precautions for programming:

- The standard modular (ICSP) RJ-11 cable does not allow for communications clock speeds greater than about 15 Mb/sec. For these high-speed applications, the Performance Pak (high-speed/LVDS) cable interface is required.
- Some circuitry should not be used on these pins. See Section 3.5.4 "Target Circuit Design Precautions". Also refer to "Development Tools Design Advisory" (DS51764).

8.2.9 How do I connect the Performance Pak pins CLK and DAT?

These connections are optional and used for SPI trace. For more information, see Section 3.6.2 "SPI Trace Connections (High-Speed/LVDS Connection)".

8.2.10 What is meant by, "The data rate is limited to 15 MIPS, when using the Standard Board?" Is this caused by the core processor or transfer rate?

The Standard Board uses the RJ-11 cable and that cable has a limitation on how fast data can reliably be transmitted when using trace, runtime watches, and data capture. The top end is when the processor has an operational speed of 15 MIPS. The trace clock is derived from the main system clock of the device.

The 15 MIPS limit is a worst-case limit. For well designed boards with less signal integrity problems, the limit may be higher.

8.2.11 Does the 15 MIPS limit apply to PIC32 devices?

No. The PIC32 MCU uses a fixed rate clock that doesn't derive from the target system clock. It also uses a more robust communication protocol.

8.2.12 To debug a dsPIC® DSC running at 30 MIPS, are high-speed communications necessary to do even basic debugging?

Basic debugging at any device frequency can be accomplished with either standard or high-speed (Performance Pak) communications.

8.2.13 My target board connector is for standard communications but I want to use high-speed communications. Can I use the high-speed/LVDS communications (Performance Pak) cables, high-speed-to-standard converter board, and standard communication (ICSP) cable?

No. You cannot use the standard/ICSP cable for high-speed communications, i.e., high device operational frequencies. This introduces signal integrity issues, due to the lower quality of cable transmission, when using the RJ-11 converter board.

8.2.14 If high-speed/LVDS communications is used, do pins 7-8 on the receiver board have to be connected, or can they just be left open?

They can be left open. The high-speed receiver board weakly pulls them down.

8.2.15 What is the function of pin 6, the auxiliary pin, on the high-speed receiver board?

This pin is reserved for use by the emulator. Do not use pin 6.

8.3 INSTRUMENTED TRACE OPERATION

Instrumented trace has three types: Native trace, SPI trace and I/O Port trace. For details, see Section 6.3.3 "Types of Trace". Not all devices support each type of instrumented trace.

The following questions relate to these trace types.

8.3.1 When using instrumented trace, is the connection electrically isolated in any way?

If you are using Native trace, you can use the opto-isolator, or MPLAB® REAL ICE™ Isolator Unit (AC244005), to isolate from high-voltage AC. There is no equivalent for SPI trace or I/O Port trace.

8.3.2 Can we do trace by using the 5 or 6 ICSP pins only?

Native trace is possible using the standard ICSP interface. SPI trace and I/O Port trace require additional hardware. See Section 3.6 "Trace Connections".

8.3.3 Does inputting execution speed in the Project Properties dialog (*File>Project Properties*, REAL ICE category, Clock option category) actually set communication clock?

No. The input box in this tab merely reports the speed to the emulator so that it can accurately control timing. Reporting the clock is needed only for Native trace, data capture, and runtime watches. If you are using clock switching, this can cause issues with these features.

8.3.4 When would SPI trace be used? What extra advantage does this have?

SPI trace can be used on devices that do not have Native trace. Also, SPI trace is faster than Native trace and impacts code size less. See Section 6.3.3 "Types of Trace" for comparison details.

8.3.5 In order to use the SPI trace, what is the hardware connection?

For serial SPI port trace, the device SPI SDO (serial data output) and SCK (serial clock) are required. These pins must be connected, respectively, to the DAT and CLK pin interface on the Performance Pak receiver board.
See Section 3.6.2 "SPI Trace Connections (High-Speed/LVDS Connection)" for more information.

8.3.6 For SPI trace, which two pins are used?

The pins are:

- SDO (Serial Data Output) → DAT (pin 7)
- SCK (Serial Clock Output) → CLK (pin 8)

See Section 3.6.2 "SPI Trace Connections (High-Speed/LVDS Connection)" for more information.

8.3.7 What are the correct port settings to use SPI trace, i.e., mode, sync/async, etc.?

The setup is taken care of by MPLAB X IDE, so you will not need to be concerned about the code required for setting this. Trace will support 64 trace points and 64 log points. For details, see Section 6.3.9 "More on Trace/Log ID Numbers".

SPI – Comm Protocol MODE1, clock high, sampled falling edge.

8.3.8 What is the correct connection for using I/O Port trace?

The connection varies depending on the PORT used. There are port assignments in MPLAB X IDE that are displayed when the PORT is selected in the property sheet. See Section 3.6.3 “I/O Port Trace Connections (Logic Port)” for more information.

8.3.9 Can I use any port?

The port must be available on the device and not multiplexed with the currently used PGC and PGD pins.

8.3.10 Of the eight (8) port pins, which one is the clock?

For details, see Section 6.3.9 “More on Trace/Log ID Numbers”.

8.3.11 Are these I/O ports used for trace available as general I/O during debugging?

For 8-bit devices, once the ports are defined to be used for trace, you **should not** access them in your code.

For 16-bit devices, you may write to the opposing 8-bit part of the port provided byte write operations are used. The following example will write to the high side of the port only.

```
#define high(num)   (((BYTE *)&num)[1])
#define low(num)    (((BYTE *)&num)[0])
high(PORTA) = 0x12;
```

8.3.12 I cannot get trace to work. What’s wrong?

Consider the following:

- Certain tool versions are required to use trace. Refer to Chapter 6. “Specific Debug Functions”.
- Only C code can be used with trace, not assembly.
- In-line assembly code (assembly code within C code) cannot be traced.
- Code must be rebuilt and reprogrammed when trace macros are added.
- Ensure you do not have trace disabled, i.e., Project Properties window, REAL ICE category, “Trace and Profiling” options category, “Data Collection Selection” should not be “Off”.
- Native trace, data captures and runtime watches cannot be used together.
- The target clock frequency must be reported for Native and SPI trace in the Project Properties window, REAL ICE category, “Clock” options category.
- For Port I/O Trace:
  - All 8 pins must be dedicated to trace (i.e., not multiplexed with the currently used PGC and PGD pins.)
  - Ensure that the chosen port is able to output 0x00 and 0xFF. As a test, set the port TRIS to 0 (all outputs) and set the LAT to a value in the Watches window. The value written to LAT should appear on the port pins.
- Ensure pins are set as digital I/O and not analog.

8.4 GENERAL ISSUES

The following questions cover general emulator functions.

8.4.1 My device keeps failing to program. What is wrong?

If you have code that changes Flash memory, using Run (*Run>Run Project*) will run your code immediately after programming and the verify operation could fail. To prevent the code from running after programming:

1. Keep the debug tool connected – *Tools>Options*, **Embedded** button, **Generic Settings** tab, check “Maintain active connection to hardware tool”.
2. Select ‘Hold in Reset’.

8.4.2 I have set a breakpoint, but my program doesn’t stop there. Why?

Consider the following:

1. Are you executing code in Run mode, i.e., selecting *Run>Run Project*? Breakpoints are ineffective in this mode.
2. Are these breakpoints part of a sequence? Remember, sequence breakpoints will only halt execution if they are followed in the exact sequence.
3. Are you seeing breakpoint skid? See the MPLAB X IDE Help file for the emulator, Limitations section.
4. Are you using compiler optimizations? If so, your program may not have executed the instruction at which you wish to break. Try setting a breakpoint earlier in your code and single step from there to see the actual code flow executed by the device.

8.4.3 I can’t set a breakpoint on a particular line of code. Why?

There may be many reasons why you cannot set a breakpoint:

1. The line you are trying to break on is not executable code. MPLAB X IDE will allow setting breakpoints only on executable code.
2. The line you are trying to break on may have been optimized out by the compiler. In that case, try turning optimizations off.
3. The line you are trying to break on may be a compiler directive, and not actual code.
4. You may have used up all resources available for breakpoints.
5. You may be in the wrong file. If you have multiple projects open that contain files with similar names, ensure you used the file you were expecting.
6. If you are using `ifdef` code, ensure the code was preprocessed to appear in executable code.

You can use the Breakpoints window to help determine the state of all your breakpoints.

8.4.4 I didn’t set a breakpoint, yet I have one in my code. What’s going on?

What you are seeing is a phantom breakpoint. Occasionally, a breakpoint can become enabled when it shouldn’t be. Simply disable or delete the breakpoint. You may need to go to the disassembly window to do so, or simply right click and choose delete all breakpoints. If this does not work, try closing and reopening the disassembly window.

8.4.5 Can I use the debugger with optimized code?

It is strongly recommended that C compiler optimization is turned off during debugging (see the Project Properties window). Optimization will greatly alter how the executable code corresponds to the source files and hence the debugger may appear to behave strangely with some code.

8.4.6 Data capture is not working correctly. What is going on?

The speed of data capture may be too high for USB communications or target environment (noise). There are several things you can do:

1. Reduce the clock speed. Refer to Section 11.3.6 “Clock”.
2. Reduce the number of runtime watches in the Watches window.
3. Check for device errata to ensure that there are not issues with data capture.

8.4.7 I cannot get PIC32 instruction trace to work. What’s wrong?

Consider the following:

- Certain tool versions are required to use trace. Please refer to Chapter 6. “Specific Debug Functions”.
- Ensure you do not have trace disabled, i.e., Project Properties window, REAL ICE category, “Trace and Profiling” options category, “Data Collection Selection” should not be “Off”.
- Ensure pins are set as digital I/O and not analog.

8.4.8 My program halts while I am tracing, corrupting my trace data. What’s happening?

The Watchdog timer can cause this behavior. Ensure it is disabled by properly setting the Configuration bits.

8.4.9 My PC went into power-down/hibernate mode, and now my emulator won’t work. What happened?

When using the emulator for prolonged periods of time, and especially as a debugger, be sure to disable the Hibernate mode in the Power Options Dialog window of your PC’s operating system. Go to the Hibernate tab and clear or uncheck the “Enable hibernation” check box. This will ensure that all communication is maintained across all the USB subsystem components.

8.4.10 I set my peripheral to NOT freeze on halt, but it is suddenly freezing. What’s going on?

For 16- and 32-bit devices, a reserved bit in the peripheral control register (usually either bit 14 or 5) is used as a Freeze bit by the debugger. If you have performed a write to the entire register, you may have overwritten this bit. (The bit is user accessible in Debug mode.)

To avoid this problem, write only to the bits you wish to change for your application (`BTS`, `BTC`) instead of to the entire register (`MOV`).

8.4.11 An unexpected Reset occurred. How do I determine what caused it?

Consider the following actions:

- Ensure the Watchdog Timer is disabled in the Configuration bits.
- To determine a Reset source, check the RCON register.
- Handle traps/interrupts in an interrupt service routine (ISR). See below.

  16-Bit Devices: You should include `trap.c` style code:

```
void __attribute__((__interrupt__)) _OscillatorFail(void);
          :
void __attribute__((__interrupt__)) _AltOscillatorFail(void);
          :
void __attribute__((__interrupt__)) _OscillatorFail(void)
{
     INTCON1bits.OSCFAIL = 0;        //Clear the trap flag
     while (1);
}
          :
void __attribute__((__interrupt__)) _AltOscillatorFail(void)
{
     INTCON1bits.OSCFAIL = 0;
     while (1);
}
          :
```

32-Bit Devices: You should include `exception.c` style code:

```
void __attribute__((interrupt())) _OscillatorFail(void);
          :
void __attribute__((interrupt())) _AltOscillatorFail(void);
          :
void __attribute__((interrupt())) _OscillatorFail(void)
{
     INTCON1bits.OSCFAIL = 0;        //Clear the trap flag
     while (1);
}
          :
void __attribute__((interrupt())) _AltOscillatorFail(void)
{
     INTCON1bits.OSCFAIL = 0;
     while (1);
}
          :
```

8.4.12 How can I manually download the firmware?

In general, you do not need to manually download the firmware; MPLAB X IDE will do this automatically. For details, see Section 4.3 “Firmware Upgrades”.

8.4.13 I accidentally disconnected my emulator while firmware was downloading. What do I do now?

See Section 4.3.3 “Communication Interruption When Installing Firmware”.

8.4.14 My code updates Flash memory; why I don’t I see changes in the memory window?

In order to see these changes, you must do a read of the memory.

8.4.15 I don’t see my problem here. Now what?

Try the following resources:

- Section 3.10 “Resources Used by the Emulator”
- Section 9.2 “Error, Warning and Informational Messages”
- Section 9.3 “General Corrective Actions”

NOTES:

Chapter 9. Messages

9.1 INTRODUCTION

The MPLAB REAL ICE in-circuit emulator produces many different output messages. Many of the error messages can be resolved with general corrective actions.

- Error, Warning and Informational Messages
- General Corrective Actions

9.2 ERROR, WARNING AND INFORMATIONAL MESSAGES

The MPLAB REAL ICE in-circuit emulator produces the following types of output messages:

- Error Message – Your application will not continue to build/execute until the specified error is corrected.
- Warning Message – Your application will build/execute, but may not operate properly until the specified warning is addressed.
- Informational Message – Information about a build, execution or test is displayed.

Many error or warning messages contain instructions about how you might resolve the issue raised by the message. If you have additional problems, you should consult Section 9.3 "General Corrective Actions".

A few error or warning messages may require specific corrective actions, as specified below.

The Debug Executive is found but can't be communicated with. Please ensure your oscillator settings are correct. If the device supports internal RC try to connect via that mode first.

See also Section 9.3.2 "Emulator-to-Target Communication Error Actions".

Failed to program device

See Section 8.4.1 "My device keeps failing to program. What is wrong?".

Failed to send database

If you receive this error, perform the following steps:

1. Try downloading again. It may be a one-time error.
2. Try manually downloading the highest-number `.jam` file.

Failed to download firmware.

If the Hex file does exist:

- Reconnect and try again.
- If this does not work, the file may be corrupted. Reinstall MPLAB X IDE.

If the Hex file does not exist, reinstall MPLAB X IDE.

Invalid streaming data was been detected. Run time watch or trace data may no longer be valid. Is recommended that you restart your debug session.

See the FAQ, "Data capture is not working correctly. What is going on?".

Loopback test completed successfully. Your REAL ICE is functioning properly. If you are still having problems with your target circuit please check the Target Board Considerations section of the online help.

See Section 12.8 “Target Board Considerations”.

REAL ICE is busy. Please wait for the current operation to finish.

If you receive this error when attempting to deselect the emulator as a debugger or programmer:

1. Wait – give the emulator time to finish any application tasks. Then try to deselect the emulator again.
2. Select Halt to stop any running applications. Then try to deselect the emulator again.
3. Unplug the emulator from the PC. Then try to deselect the emulator again.
4. Shut down MPLAB X IDE.

Target Device ID (0x%x) does not match expected Device ID (0x%x).

If the Target Device ID is 0x0, your device may not be properly connected to, or may be absent from, the target board.

The target device is not ready for debugging. Please check your Configuration bit settings and program the device before proceeding. The most common causes for this failure are oscillator and/or PGC/PGD settings.

You will receive this message when you have not programmed your device for the first time and try to Run. If you receive this message after this, or immediately after programming your device, please refer to Section 9.3.6 “Debug Failure Actions”.

Target device was not found (could not detect target voltage V_{DD}). You must connect to a target device to use MPLAB REAL ICE.

See also Section 9.3.2 “Emulator-to-Target Communication Error Actions”.

Unable to download debug/program executive.

If you receive this error while attempting to debug, perform the following steps:

1. Deselect the emulator as the debug tool.
2. Close your project, and then close MPLAB X IDE.
3. Restart MPLAB X IDE, and re-open your project.
4. Reselect the emulator as your debug tool, and attempt to program your target device again.

9.3 GENERAL CORRECTIVE ACTIONS

These general corrective actions may solve your problem:

- Read/Write Error Actions
- Emulator-to-Target Communication Error Actions
- Emulator-to-PC Communication Error Actions
- Corrupted Installation Actions
- USB Port Communication Error Actions
- Debug Failure Actions
- Internal Error Actions

9.3.1 Read/Write Error Actions

If you receive a read or write error:

1. Did you hit Abort? This may produce read/write errors.
2. Try the action again. It may be a one-time error.
3. Ensure that the target is powered and at the correct voltage levels for the device. See the device data sheet for required device voltage levels.
4. Ensure that the emulator-to-target connection is correct (PGC and PGD are connected.)
5. For write failures, ensure that “Erase all before Program” is checked on the **Program Memory** tab of the Settings dialog.
6. Ensure that the cable(s) used are of the correct length - maximum 6” for standard communications and 10’ for high-speed communications.

9.3.2 Emulator-to-Target Communication Error Actions

The MPLAB REAL ICE in-circuit emulator and the target device are out-of-sync with each other.

1. Select **Reset** and then try the action again.
2. Ensure that the cable(s) used are of the correct length – maximum 6" for standard communications and 10' for high-speed communications.

9.3.3 Emulator-to-PC Communication Error Actions

The MPLAB REAL ICE in-circuit emulator and MPLAB X IDE are out of sync with each other.

1. Unplug and then plug in the emulator.
2. Reconnect to the emulator.
3. Try the operation again. It is possible that the error was a one time glitch.
4. The version of MPLAB X IDE installed may be incorrect for the version of firmware loaded on the MPLAB REAL ICE in-circuit emulator. Follow the steps outlined in Section 9.3.4 “Corrupted Installation Actions”.
5. There may be an issue with the PC USB port. See Section 9.3.5 “USB Port Communication Error Actions”.

9.3.4 Corrupted Installation Actions

The problem is most likely caused by a incomplete or corrupted installation of MPLAB X IDE.

1. Uninstall all versions of MPLAB X IDE from the PC.
2. Reinstall the desired MPLAB X IDE version.
3. If the problem persists, contact Microchip.

9.3.5 USB Port Communication Error Actions

The problem is most likely caused by a faulty or non-existent communications port.

1. Reconnect to the MPLAB REAL ICE in-circuit emulator
2. Make sure the emulator is physically connected to the PC on the appropriate USB port.
3. Make sure the appropriate USB port has been selected in the emulator Settings.
4. Make sure the USB port is not in use by another device.
5. If using a USB hub, make sure it is powered.
6. Make sure the USB drivers are loaded.

9.3.6 Debug Failure Actions

The MPLAB REAL ICE in-circuit emulator was unable to perform a debugging operation. There are numerous reasons why this could occur. See Chapter 7. “Troubleshooting First Steps”.

9.3.7 Internal Error Actions

Internal errors are unexpected and should not happen. They are primarily used for internal Microchip development.

The most likely cause is a corrupted installation (Section 9.3.4 “Corrupted Installation Actions”).

Another likely cause is exhausted system resources.

1. Try rebooting your system to free up memory.
2. Make sure you have a reasonable amount of free space on your hard drive (and that it is not overly fragmented).

If the problem persists, contact Microchip.

Chapter 10. Engineering Technical Notes (ETNs)

The following ETNs are related to the MPLAB REAL ICE in-circuit emulator. See the product web page for details.

ETN-30: Applies to Assembly #10-00401-R1 or below.

NOTES:

Part 4 – Software & Hardware Reference

NOTES:

Chapter 11. Emulator Function Summary

11.1 INTRODUCTION

A summary of the MPLAB REAL ICE in-circuit emulator functions is listed here.

- Emulator Selection and Switching
- Emulator Options Selection
- Emulator Windows & Dialogs

11.2 EMULATOR SELECTION AND SWITCHING

Use the Project Properties dialog to select or switch emulators for a project. To switch you must have more than one MPLAB REAL ICE in-circuit emulator connected to your computer. MPLAB X IDE will differentiate between the two by displaying two different serial numbers.

To select or change the emulator used for a project:

1. Open the Project Properties dialog by doing one of the following:
   a) Click on the project name in the Projects window and select *File>Project Properties*.
   b) Right click on the project name in the Projects window and select "Properties".
2. Under "Categories", click on "Conf: [default]"
3. Under "Hardware Tools", find "REAL ICE" and click on a serial number (SN) to select an emulator for use in the project.

11.3 EMULATOR OPTIONS SELECTION

Set up emulator options on the emulator property pages of the Project Properties dialog.

1. Open the Project Properties dialog by doing one of the following:
   a) Click the project name in the Projects window, select *File>Project Properties*.
   b) Right click the project name in the Projects window, select "Properties".
2. Under "Categories", click on "REAL ICE"
3. Select property pages from "Options categories". Click on an option to see its description in the text box below it. Click to the right of an option to change it.

Available option categories are:

- Memories to Program
- Debug Options
- Program Options
- Freeze Peripherals
- Trace and Profiling
- Clock
- Firmware
- External Triggers

11.3.1 Memories to Program

Select the memories to be programmed into the target.

If “Erase All Before Program” is selected under Section 11.3.3 “Program Options”, then all device memory will be erased before programming. To select only certain memories to program after erase, check the specific memory type. To preserve the value of memory of different types, check to preserve that memory type *and* check the specific memory type; checking “Preserve *Memory*” writes the current contents to a buffer before erase, and checking “*Memory*” writes the contents back into that memory after erase, where *Memory* is the type of memory, such as EEPROM.

TABLE 11-1: MEMORIES TO PROGRAM OPTION CATEGORY

Auto select memories and ranges	**Allow REAL ICE to Select Memories** – The emulator uses your selected device and default settings to determine what to program. **Manually select memories and ranges** – You select the type and range of memory to program (see below.)
Memory	Check to program *Memory*, where *Memory* is the type of memory. Types include: EEPROM, ID, Boot Flash, Auxiliary.
Program Memory	Check to program the target program memory range specified below.
Program Memory Range(s) (hex)*	The starting and ending hex address range in program memory for programming, reading, or verification. **Note:** The address range does not apply to the Erase function. The Erase function will erase all data on the device.
Preserve Program Memory	Check to not program the target program memory range specified below. Ensure code is NOT code protected.
Preserve Program Memory Range(s) (hex)*	The starting and ending hex address range in target program memory to preserve when programming, reading, or verifying. This memory is read from the target and overlayed with existing MPLAB X IDE memory.
Preserve *Memory*	Check to preserve *Memory* for reprogramming, where *Memory* is the type of memory. Types include: EEPROM, ID, Boot Flash, Auxiliary. Ensure code is NOT code protected.
Preserve *Memory* Range(s) (hex)*	The starting and ending hex address range in target *Memory* to preserve when programming, reading, or verifying. *Memory* is the type of memory, which includes EEPROM, ID, Boot Flash, Auxiliary. This memory is read from the target and overlayed with existing MPLAB X IDE memory. Ensure code is NOT code protected.

* If you receive a programming error due to an incorrect range, ensure the range does not exceed available/remaining device memory.

11.3.2 Debug Options

Use software breakpoints, if available for the project device.

TABLE 11-2: DEBUG OPTIONS OPTION CATEGORY

Use Software Breakpoints	Check to use software breakpoints. Uncheck to use hardware breakpoints. See the discussion below to determine which type is best for your application.

TABLE 11-3: SOFTWARE VS HARDWARE BREAKPOINTS

Features	Software Breakpoints	Hardware Breakpoints
Number of breakpoints	unlimited	limited
Breakpoints are written to	program memory	debug registers
Time to set breakpoints	oscillator speed dependent, it can take minutes	minimal
Skidding	no	yes

Note: Using software breakpoints for debug impacts device endurance. So, it is recommended that devices used in this manner should not be used as production parts.

11.3.3 Program Options

Choose to erase all memory before programming, or to merge code.

TABLE 11-4: PROGRAM OPTIONS OPTION CATEGORY

Erase All Before Program	Check to erase all memory before programming begins. Unless programming new or already erased devices, it is important to have this box checked. If it is not checked, the device is not erased and program code will be merged with the code already in the device.
Enable Low Voltage Programming	*For Programmer Settings only, PIC12F/16F1xxx devices:* • For the LVP Configuration bit set to “Low-voltage programming enabled”, you may program in either high-voltage (default) or low-voltage (enabled here.) • For the LVP Configuration bit set to “High-voltage on MCLR/VPP must be used for programming”, you may only program in high-voltage.
Do Not Erase Auxiliary Memory	*For devices that support auxiliary memory:* Check to not erase aux memory when programming. Uncheck to erase aux memory when programming.

11.3.4 Freeze Peripherals

Select peripherals to freeze, or not freeze, on program halt. Options available depend on device or header chosen.

TABLE 11-5: FREEZE PERIPHERALS OPTION CATEGORY

Freeze Peripherals	Check to freeze all peripherals on halt. Uncheck to unfreeze all peripherals. This options applies to PIC12/16/18 MCUs.
Peripheral Freeze Enable *Peripheral List*	Check to select which peripheral(s) to freeze. Uncheck to unfreeze all peripherals. This option applies to AC244066.
Peripheral List	Check to freeze the peripheral *Peripheral* on halt. Uncheck to unfreeze the peripheral *Peripheral*. This options applies to 16- and 32-bit MCUs.

PIC12/16/18 MCU Devices

To freeze/unfreeze all device peripherals on halt, check/uncheck the “Freeze Peripherals” checkbox. If your desired peripheral does not halt, be aware that some peripherals have no freeze on halt capability and cannot be controlled by the emulator.

AC244066 Emulation Extension Pak (PIC16F1619-ME2)

Check the “Peripheral Freeze Enable” checkbox to select which peripherals to freeze. All are selected by default. Uncheck this checkbox unfreeze all peripherals. If your desired peripheral does not halt, be aware that some peripherals have no freeze on halt capability and cannot be controlled by the emulator.

dsPIC, PIC24 and PIC32 Devices

To freeze/unfreeze a peripherals on halt, check/uncheck the peripheral from the list. If you do not see a peripheral on the list, check/uncheck “Freeze All Other Peripherals”. If your desired peripheral does not halt, be aware that some peripherals have no freeze on halt capability and cannot be controlled by the emulator.

11.3.5 Trace and Profiling

Depending on the device you have selected for your project, you may be able to use trace, PC sampling/profiling or other data collection features when debugging. Enable and set up these features as specified in the following sections.

8-Bit and 16-Bit Devices

Options available on this page depend on the trace/profiling features of the project device. For more on trace and profiling, see **Chapter 6. "Specific Debug Functions"**.

TABLE 11-6: TRACE/PROFILING OPTION CATEGORY

Data Collection Selection	Enable/Disable data collection. • Off - Do not collect target data. • User Instrumented Trace - see Section 6.3 "Instrumented Trace". • PC Sampling - see Section 6.6 "PC Sampling – 8-Bit and 16-Bit MCUs Only". • Function Level Profiling - see Section 6.8 "Function Level Profiling". • Jump Trace - see Section 6.5 "Jump Trace – EP Devices Only". • Power Monitor (Target Power Sampling) - see the "*MPLAB REAL ICE Power Monitor User's Guide*" (DS50002532)
Data File Path and Name	Enter or change the path and/or name of the file used to store data. • Enter file name (path will be relative to project) – Recommended • Enter a path and file name (path will be absolute) • Browse (...) to a file, select "Absolute", select the file, and click **Save** (path will be absolute) **Note:** Do not select "Relative" when browsing to a file or MPLAB X IDE will not be able to find the file. When you run, you will receive a warning message that the path does not exist.
Data File Maximum Size (bytes)	Set the maximum size of the data file. Target power sampling will take 12 bytes or 18 bytes (with PC data) per sample. The file size may be adjusted down to be a multiple of one of those byte sizes depending upon the trace type selected. Other trace data types may use record byte sizes that are different from those described above.
Data Buffer Maximum Size (bytes)	Set the size of the data buffer, up to 54600 bytes (on board the emulator unit.) For trace/sampling data that is buffered in memory while the target is running, individual trace or sample entry sizes vary depending upon the trace/sample type and the device and tool being used. It is normally good to make this buffer as large as possible. For example, the enhanced PIC16 with instruction trace uses 1 to 3 bytes for each in-memory entry. Each of those will generate a 13-byte REAL ICE instruction trace entry as well. Each such in-memory record will normally be converted to a trace data file entry line, as detailed in the data file size description (refer to the data file size description for trace/sampling file entry sizes).
Stall CPU When Trace Buffer is Full	Stop execution when the trace buffer is full. Set the buffer size in the option described above.
User Instrumented Trace Items	
Disable Trace Macros	Check to temporarily disable trace macros or uncheck to enable trace macros. To disable trace, remove all macros and select "Off" under "Data Collection Selection".
Communications Medium	Select the trace medium, if available, from the following (device-dependent): Native, I/O Port, SPI.
I/O Port Selection	Specify the device port to be used for I/O port trace. The available combinations for the selected device will be listed.
SPI Selection	Specify the device SPI pins to be used for SPI trace. The available pins for the selected device will be listed.

TABLE 11-6: TRACE/PROFILING OPTION CATEGORY (CONTINUED)

PC Sampling Items	
Timer Selection (Not Used by Application Code)	Select a device timer to use to count PC samples. **Note:** You will no longer be able to use this timer in your application, it will be dedicated to PC sampling. **Note:** You may select only one timer; you cannot combine two timers to get a 32-bit timer. Using one timer of a 32-bit-timer pair will prohibit that pair from operating as a 32-bit timer.
Timer Interrupt Priority	Select an interrupt priority for the timer. **Note:** Select a priority that is higher than other priorities you have set in your application. If you do not, the other priorities will preempt the sampling priority and you will not capture these samples.
Timer Interval	Enter a sampling interval. This must be integer values (1, 2, 3, and so forth). If you are not capturing data, you may be missing samples (given your current interval). Try adjusting the unit selection and interval, for example, if you had 1 millisecond, try 990 microseconds.
Timer Interval Units	Select a sampling interval unit: • microseconds • milliseconds • seconds • instruction cycles
Function Level Profiling – Works with Code Profiling Plugin	
Time Stamp or Summary Profile Data	Select to generate either time stamped data or summary profile data. Summary profile data is selected by default. **Note:** If "Power Monitor" is the "Data Collection Selection", this item will default to "Include Time Stamp" as timestamped power data is collected.

32-Bit Devices

Options available on this page depend on the trace/profiling features of the project device. For more on trace and profiling, see **Chapter 6. "Specific Debug Functions"**.

TABLE 11-7: TRACE/PROFILING OPTION CATEGORY

Data Collection Selection	Enable/Disable data collection. • Off – Do not collect target data. • Instruction Trace/Profiling - see Section 6.4 "PIC32 Instruction Trace – PIC32 MCUs Only" and Section 6.7 "PC Profiling – 32-Bit MCUs Only". • User Instrumented Trace - see Section 6.3 "Instrumented Trace". • Power Monitor (Target Power Sampling) - see the "*MPLAB REAL ICE Power Monitor User's Guide*" (DS50002532)
Data File Path and Name	Enter or change the path and/or name of the file used to store data. • Enter file name (path will be relative to project) – Recommended • Enter a path and file name (path will be absolute) • Browse (...) to a file, select "Absolute", select the file, and click **Save** (path will be absolute) **Note:** Do not select "Relative" when browsing to a file or MPLAB X IDE will not be able to find the file. When you run, you will receive a warning message that the path does not exist.
Data File Maximum Size (bytes)	Set the maximum size of the data file. Each line of instruction trace data in a trace data file requires 13 bytes when using the emulator. Target power sampling will take 12 bytes or 18 bytes (with PC data) per sample. The file size may be adjusted down to be a multiple of one of those byte sizes depending upon the trace type selected. Other trace data types may use record byte sizes different from those described above.
Data Buffer Maximum Size (bytes)	Set the size of the data buffer, up to 54600 bytes (on board the emulator unit.) For trace/sampling data that is buffered in memory while the target is running, individual trace or sample entry sizes vary depending on the trace/sample type and the device and tool being used. It is normally good to make this buffer as large as possible. For example, PIC32 instruction trace takes 8 bytes per "frame" which can produce over 50 13-byte REAL ICE instruction trace entries in a trace file.
User Instrumented Trace Items	
Disable Trace Macros	Check to temporarily disable trace macros or uncheck to enable trace macros. To disable trace, remove all macros and select "Off" under "Data Collection Selection".
Communications Medium	Select the trace medium, if available (device dependent): Native.
Function Level Profiling – Works with Code Profiling Plugin	
Time Stamp or Summary Profile Data	Select to generate either time stamped data or summary profile data. Summary profile data is selected by default. **Note:** If "Power Monitor" is the "Data Collection Selection", this item will default to "Include Time Stamp" as timestamped power data is collected.

11.3.6 Clock

Enter the runtime clock (instruction) speed under this option category. This does not set the speed, but informs the emulator of its value for runtime watch, data capture and trace.

> **Note:** Clock switching is available for data capture and trace, but you must set the clock correctly or you may see issues. Enter the fastest instruction speed you will be using.

TABLE 11-8: CLOCK OPTION CATEGORY

Use FRC in Debug mode (dsPIC33E/F and PIC24E/F/H devices only)	When debugging, use the device fast internal RC (FRC) for clocking instead of the oscillator specified for the application. This is useful when the application clock is slow. Checking this checkbox will let the application run at the slow speed but debug at the faster FRC speed. Reprogram after changing this setting. **Note:** Peripherals that are not frozen will operate at the FRC speed while debugging.
Target run-time instruction speed	Enter a value for the "Speed unit" selected. **Example 1:** For a PIC24 MCU and a target clock oscillator at 32 MHz (HS), instruction speed = 32 MHz/2 = 16 MIPS. **Example 2:** For a PIC18F8722 MCU and a target clock oscillator at 10 MHz (HS) making use of the PLL (x4 = 40 MHz), instruction speed = 40 MHz/4 = 10 MIPS.
Instruction speed units	Select either: KIPS – Thousands (10^3) of instructions per second MIPS – Millions (10^6) of instructions per second

11.3.7 Firmware

Select and load emulator firmware. MPLAB X IDE automatically downloads the correct firmware for your project. Only change this setting if you are having issues.

TABLE 11-9: FIRMWARE OPTION CATEGORY

Use Latest Firmware	Check to use the latest firmware. Uncheck to select the firmware version below.
Firmware File	Click in the right-hand text box to search for a firmware file to associate with the emulator. For details, see Section 4.3 "Firmware Upgrades".

11.3.8 External Triggers

Select external triggers functions. For more on triggers, see Section 6.10 "External Triggers".

TABLE 11-10: EXTERNAL TRIGGERS OPTION CATEGORY

Trigger 0 : Trigger 7	Select trigger function: Off Input – Positive Edge Triggered – Halt On Trigger Input – Positive Edge Triggered – Reset On Trigger Input – Negative Edge Triggered – Halt On Trigger Input – Negative Edge Triggered – Reset On Trigger Output – High-to-Low Pulse – Assert on Halt Output – High-to-Low Pulse – Assert on Run Output – Low-to-High Pulse – Assert on Halt Output – Low-to-High Pulse – Assert on Run

11.4 EMULATOR WINDOWS & DIALOGS

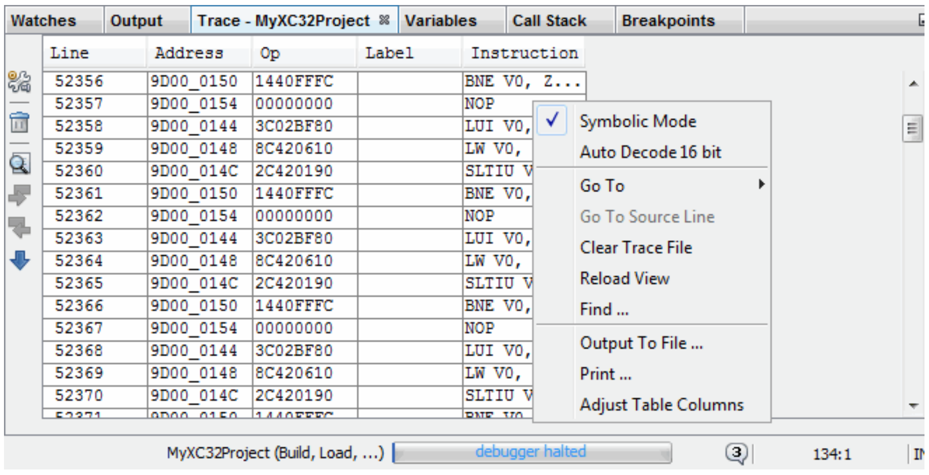
The following windows and dialogs are used specifically for the emulator and/or other debug tools.

- Trace Window and Related Dialogs
- Application In/Out Window and Related Dialogs
- PC Sampling Window and Related Dialogs

11.4.1 Trace Window and Related Dialogs

The trace window displays the results of a trace. This window is available for the emulator and the simulator.

FIGURE 11-1: TRACE WINDOW



Line	Address	Op	Label	Instruction
52356	9D00_0150	1440FFFC		BNE V0, Z...
52357	9D00_0154	00000000		NOP
52358	9D00_0144	3C02BF80		LUI V0,
52359	9D00_0148	8C420610		LW V0,
52360	9D00_014C	2C420190		SLTIU V
52361	9D00_0150	1440FFFC		BNE V0,
52362	9D00_0154	00000000		NOP
52363	9D00_0144	3C02BF80		LUI V0,
52364	9D00_0148	8C420610		LW V0,
52365	9D00_014C	2C420190		SLTIU V
52366	9D00_0150	1440FFFC		BNE V0,
52367	9D00_0154	00000000		NOP
52368	9D00_0144	3C02BF80		LUI V0,
52369	9D00_0148	8C420610		LW V0,
52370	9D00_014C	2C420190		SLTIU V

Right clicking in a column of the window shown above will pop up a menu with a list of functions. For more on these functions, see the MPLAB X IDE User's Guide (DS52027), "MPLAB X IDE Windows and Dialogs", "Trace Window".

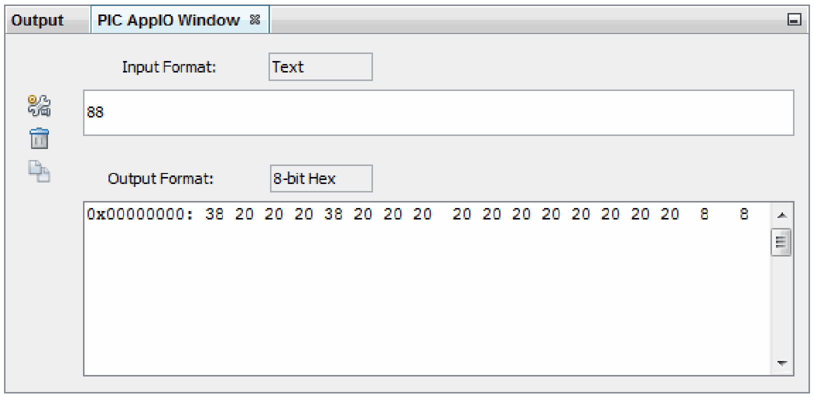
For more on using trace, see:

- Section 6.3 "Instrumented Trace"
- Section 6.4 "PIC32 Instruction Trace – PIC32 MCUs Only"

11.4.2 Application In/Out Window and Related Dialogs

The Application In/Out window supports the App IO function in which runtime control information can be sent to an application through MPLAB IDE (APPIN) and status information can be sent by the application to MPLAB IDE (APPOUT).

FIGURE 11-2: APP IN/OUT WINDOW



Text

88

8-bit Hex

```
0x00000000: 38 20 20 20 38 20 20 20  20 20 20 20 20 20 20 20  8   8
```

Other actions are available from the buttons. Right clicking in the Output text box will pop-up a menu with the same functions.

TABLE 11-11: APP IN/OUT WINDOW BUTTONS

Button	Description
Properties	Open the Set App IO Properties dialog. Set the format of the input and output: Text, 8-bit Hex, 16-bit Hex, or 32-bit Hex. Enable/disable data capture. Browse to a location to save the output to a file.
Clear App IO Output	Clear the output content from the Output text box.
Copy App IO Output to Output View	Copy the Output text box content to the Output window.

For more on using this window and its related dialogs, see Section 6.9 "Application In/Out".

11.4.3 PC Sampling Window and Related Dialogs

Program Counter (PC) sampling is a method of determining the amount of time spent in each application function for use in code optimization. The PC Sampling window will show the function name, sample count for that function, percentage sample count is of the total samples, and a bar graph of the count.

For more on PC Sampling and Profiling, see:

- Section 6.6 "PC Sampling – 8-Bit and 16-Bit MCUs Only"
- Section 6.7 "PC Profiling – 32-Bit MCUs Only"

Chapter 12. Hardware Specification

12.1 INTRODUCTION

The hardware and electrical specifications of the basic MPLAB REAL ICE in-circuit emulator system are detailed.

12.2 HIGHLIGHTS

This chapter discusses:

- USB Port/Power
- Emulator Pod
- Standard Communication Hardware
- High-Speed/LVDS Communication Hardware
- Loopback Test Board
- Target Board Considerations

12.3 USB PORT/POWER

The MPLAB REAL ICE in-circuit emulator is connected to the host PC via a USB port, version 2.0 compliant. The USB connector is located on the back of the pod.

The system is capable of reloading the firmware via the USB interface.

System power is derived from the USB interface. The emulator is classified as a high power system per the USB specification, and requires 300 mA of power from the USB to function in all operational modes (emulator/programmer).

Note: The MPLAB REAL ICE in-circuit emulator is powered through its USB connection. The target board is powered from its own supply. The emulator cannot provide power to the target board.

Cable Length – The PC-to-emulator cable length for proper operation has been tested for each driver board and is shipped in the emulator kit.

Powered Hubs – If you are going to use a USB hub, make sure it is powered. Also, USB ports on PC keyboards do not have enough power for the emulator to operate.

12.4 EMULATOR POD

The emulator pod (DV244005) consists of a main board enclosed in the casing with a port for either of two driver boards (for standard or high-speed communication with a target). On the emulator enclosure are push buttons, indicator lights (LEDs) and a logic probe connector interface.

12.4.1 Main Board

This component has an interface processor (dsPIC DSC), a USB 2.0 interface capable of USB speeds of 480 Mb/sec, a Field Programmable Gate Array (FPGA) for general system control and increased communication throughput, an SRAM for holding the program code image for programming into the emulation device on-board Flash, the external trigger logic, user interface push buttons and LED indicators.

The MPLAB REAL ICE in-circuit emulator system supports two types of interfaces to the target processor. They consist of the standard driver board and an optional high-speed driver board. These boards are inserted into the emulator pod via a card guide.

Durability/insertion life cycle of the card guide: 10,000 cycles

12.4.2 Push Buttons

The push buttons have the following significance.

Push Button	Related LED	Description
Reset	Status	Push to Reset the device.
Function	Status	Halt – when running, push to put the emulator in the Break or halted condition.

12.4.3 Indicator Lights (LEDs)

The indicator lights have the following significance.

TABLE 12-1: LED INDICATORS

Type	Color	Condition	Description
Active	Blue	Lit	Power has been applied or target has been connected.
Status	Green	Lit	The emulator is operating normally – standby.
	Red	Lit	An operation has failed.
		Blinking	USB Comm error or driver not installed.
	Orange	Lit	The emulator is busy.

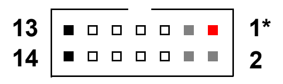
12.4.4 Logic Probe/External Trigger Interface

Logic probes (ACICE0104) can be connected to the 14-pin header on the side of the unit for processing external signals that are used for triggering external equipment. This header contains 8 input/output connections that are user selectable as inputs or outputs with logic levels that are proportional to the target operating voltage.

The outputs can be used for triggering an external logic analyzer or oscilloscope to allow the developer to capture events of interest based on trigger criteria set within MPLAB X IDE. The external trigger is a pulse of approximately 1.5 μs. This value is not deterministic and the external tool should be triggered on a pulse edge.

The inputs are part of a trigger bus. Inputs can have a latency up to 700 μs and will vary in length depending on architecture emulated (8-bit, 16-bit, 32-bit) and whether trace or streaming data is active.

FIGURE 12-1: LOGIC PROBE PINOUT ON EMULATOR



Logic probes may be attached to this connector to give the functionality described in Table 12-2. The probes are color coded and labeled for easy identification.

TABLE 12-2: LOGIC PROBE PINOUT DESCRIPTION

Pin	I/O	Name	Function	Color
1	O	VDD(1)	VDD reference	Red
2	O	NC	No connection	Gray
3	O	NC	No connection	Gray
4	I	TCLK	External synchronous clock	Gray
5	I/O	EXT7(2)	External input/output bit 7	White
6	I/O	EXT6	External input/output bit 6	White
7	I/O	EXT5	External input/output bit 5	White
8	I/O	EXT4	External input/output bit 4	White
9	I/O	EXT3	External input/output bit 3	White
10	I/O	EXT2	External input/output bit 2	White
11	I/O	EXT1	External input/output bit 1	White
12	I/O	EXT0(2)	External input/output bit 0	White
13	Gnd	GND	System Ground	Black
14	Gnd	GND	System Ground	Black

Note 1: Do not connect VDD to the target.

2: EXT0 and EXT7 are temporarily used during loopback test. Ensure that they are not connected together.

The electrical specifications for logic probes are listed in Table 12-3.

TABLE 12-3: LOGIC PROBE ELECTRICAL SPECIFICATIONS

Logic Inputs	VIH = VDD x 0.7V (min)			
	VIL = VDD x 0.3V (max)			
Logic Outputs	VDD = 5V	VDD = 3V	VDD = 2.3V	VDD = 1.65V
	VOH = 3.8V min	VOH = 2.4V min	VOH = 1.9V min	VOH = 1.2V min
	VOL = 0.55V max	VOL = 0.55V max	VOL = 0.3V max	VOL = 0.45V max

12.5 STANDARD COMMUNICATION HARDWARE

For standard emulator communication with a target (Section 3.5.1 "Standard Communication"), use the standard driver board.

To use this type of communication with a debug header, you may need a device-specific Extension Pak, which includes an 8-pin connector debug header that contains the desired -ICE/-ICD device and a standard adapter board (8-pin to 6-pin connection).

Note: Older debug headers used a 6-pin (RJ-11) connector instead of an 8-pin connector, so these headers may be connected directly to the emulator.

For more on available debug headers, see the "*Processor Extension Pak and Debug Header Specification"* in "Recommended Reading".

12.5.1 Standard Driver Board

The standard driver board (AC244001), included with the emulator but can be purchased separately, is the main interface to the target processor. It contains the connections to the high voltage (VPP), VDD sense lines, and clock and data connections required for programming and connecting with the target devices.

The VPP high-voltage lines can produce a variable voltage that can swing from 14 to 0 volts to satisfy the voltage requirements for the specific emulation processor.

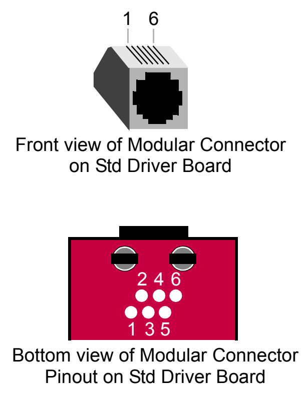
The VDD sense connection draws very little current from the target processor. The actual power comes from the MPLAB REAL ICE in-circuit emulation system as the VDD sense line is used as a reference only to track the target voltage. The VDD connection is isolated with an optical switch.

The clock and data connections are interfaces with the following characteristics:

- Clock and data signals are in High-Impedance mode (even when no power is applied to the MPLAB REAL ICE in-circuit emulator system)
- Clock and data signals are protected from high voltages caused by faulty targets systems, or improper connections
- Clock and data signals are protected from high current caused from electrical shorts in faulty target systems

Note: When using the standard driver board, the rate for real-time streaming data and tracing is limited to 15 MIPS.

FIGURE 12-2: MODULAR CONNECTOR PINOUT OF STANDARD DRIVER BOARD


1
6
Front view of Modular Connector on Std Driver Board
2 4 6
1 3 5
Bottom view of Modular Connector Pinout on Std Driver Board

Modular Connector Pin	Microcontroller Pin
1	Not Used
2	RB6
3	RB7
4	Ground
5	VDD Target
6	VPP

12.5.2 Modular Cable and Connector

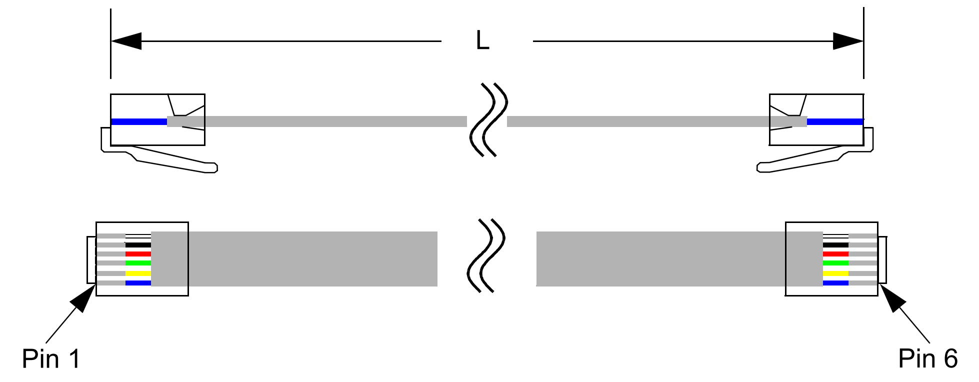
For standard communications, a modular (ICSP) cable connects the emulator and the target application. The specifications for this cable and its connectors are listed below.

12.5.2.1 MODULAR CABLE SPECIFICATION

Manufacturer, Part Number – Microchip Technology, 07-00024

The length for this cable (L) is 6 inches. It is not recommended that you use a modular cable longer than 6 inches or you may experience communication problems. If you require a longer cable, consider purchasing the Performance Pak (AC244002).

FIGURE 12-3: MODULAR CABLE



12.5.2.2 MODULAR PLUG SPECIFICATION

- Manufacturer, Part Number – AMP Incorporated, 5-554710-3
- Distributor, Part Number – Digi-Key, A9117ND

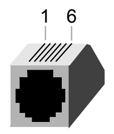
12.5.2.3 MODULAR CONNECTOR SPECIFICATION

- Manufacturer, Part Number – AMP Incorporated, 555165-1
- Distributor, Part Number – Digi-Key, A9031ND

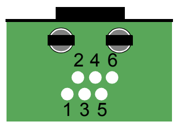
The following table shows how the modular connector pins on an application correspond to the microcontroller pins. This configuration provides full ICD functionality.

FIGURE 12-4: MODULAR CONNECTOR PINOUT OF TARGET BOARD

Modular Connector Pin	Microcontroller Pin
6	Not Used
5	RB6
4	RB7
3	Ground
2	V_{DD} Target
1	V_{PP}



Front view of Modular Connector on Target Board



Bottom view of Modular Connector Pinout on Target Board

12.6 HIGH-SPEED/LVDS COMMUNICATION HARDWARE

Information on high-speed/LVDS communication hardware may be found in the "*Performance Pak User's Guide*" (DS50002528).

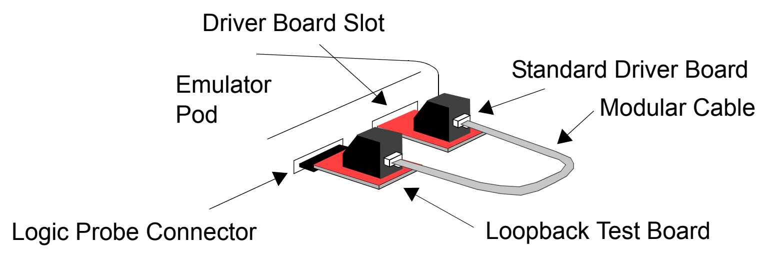
12.7 LOOPBACK TEST BOARD

This board (AC244003), included with the emulator but can be purchased separately, can be used to verify that the emulator is functioning properly. To use this board:

1. Disconnect the emulator from the target and the PC.
2. Insert the standard driver board if it is not already installed.
3. Plug the loopback test board into the pod's logic probe connector.
4. Connect the loopback test board to the standard driver board using the modular cable.
5. Reconnect the emulator to the computer.
6. Launch MPLAB X IDE. Ensure that all existing projects are closed.
7. Select *Debug>Run Debugger/Programmer Self Test*, then, select the specific "REAL ICE" you want to test and click **OK**.
8. Ensure the loopback test board and cable are connected and click **Yes** to continue.
9. View the self test results in the emulator's Output window.
10. After the emulator passes the self test, disconnect the loopback test board from the emulator.

MPLAB IDE will detect and run the complete loopback test and give you a status (PASS/FAIL). The loopback test board detection works by applying a short pulse on EXT0 and detecting it on EXT7 on the logic probe connector (Section 12.4.4 "Logic Probe/External Trigger Interface"). Once the board is detected, the emulator applies stimulus to the clock/data and VPP lines and reads the sequence back from the logic probe connector interface, thus confirming proper signals levels and connectivity down to the connector interfaces.

FIGURE 12-5: LOOPBACK TEST BOARD CONNECTIONS



12.8 TARGET BOARD CONSIDERATIONS

The target board should be powered according to the requirements of the selected device (1.6V-5.5V) and the application.

Note: The emulator cannot power the target.

The emulator does sense target power. There is a 10 kΩ load on VDD_TGT.

Depending on the type of emulator-to-target communications used, there will be some considerations for target board circuitry:

- Section 3.5.3 "Target Connection Circuitry"
- Section 3.5.4 "Target Circuit Design Precautions"

NOTES:

Appendix A. Revision History

Revision A (January 2013)

- Initial release of this document.

Revision B (May 2014)

- Any MPLAB IDE v8 links or references fixed or removed.
- Warning boxes updated throughout to match updated standards.
- **Preface** "Recommended Reading" split into two parts, Emulator and Accessory, and updated for new documents.
- **Chapter 2. "Device and Feature Support - Emulators and Debuggers"** – Table data can be ported to DTS.
- **Section 3.2 "Tools Comparison"** – Comparison table updated.
- **Section 3.3 "Operational Overview"** – Tables updated to clarify speed.
- **Section 3.5.3 "Target Connection Circuitry"** – Pull-up value changed.
- **Section 3.6.3 "I/O Port Trace Connections (Logic Port)"** – Added allowable PORTx configurations.
- **Section 4.3 "Firmware Upgrades"** – Section added.
- **Section 4.4 "Emulator Functions"** – Replaces previous sections 4.3 and 4.4.
- **Section 5.4.2 "Hardware or Software Breakpoint Selection"** – Table updated.
- **Section 5.5 "Instrumented Trace"** – Instrumented trace supported for all families for many devices. Place holders still left in Specific Debug Functions chapters.
- **Section 6.2 "Data Capture and Runtime Watches"** and **Section 8.2 "Data Capture and Runtime Watches"**– Data capture and runtime watches defined. Watch variables need to be native sized.
- Remove clock requirement from runtime watches: **Section 6.2.3 "Runtime Watches and the Watches Window"** and **Section 8.2 "Data Capture and Runtime Watches"**.
- **Section 6.5 "Jump Trace – EP Devices Only"** and **Section 6.8 "Function Level Profiling"** added.
- **Chapter 8. "Specific Debug Functions: 32-Bit Devices"** – Updated PIC32MX to PIC32 to include all 32-bit devices.
- **Section 8.3.3 "Setting Up and Using Trace"** – Diagrams added.
- **Chapter 7. "Troubleshooting First Steps"** and **Chapter 8. "Frequently Asked Questions (FAQ)"** updated.
- **Part 4 – "Software & Hardware Reference"** – "Reference" section name changed to "Software & Hardware Reference".
- **Part 5 – "Emulator Accessories"** – "Emulator Accessories" chapter removed from "Reference" section to create chapter in this part. Accessory chapters added for available emulator accessories.

Revision C (May 2015)

- Substituted references to PIC32MX with PIC32 (to include PIC32MZ parts).
- Updated screen captures as needed.
- **Section 1.4 "Emulator Kit Components"** – Added information on additional hardware.
- **Section 3.2 "Tools Comparison"** – Updated "VDD Drain from Target".
- **Section 3.5.3 "Target Connection Circuitry"** – Updated pull-up resistor value.
- **Chapter 6. "Specific Debug Functions"** – Merged Chapter 6. "Specific Debug Functions: 8- and 16-Bit Devices" and Chapter 7."Specific Debug Functions: 32-Bit Devices" into one chapter. Many features now available for all devices.
- **Section 6.6 "PC Sampling – 8-Bit and 16-Bit MCUs Only"**, **Section 6.7 "PC Profiling – 32-Bit MCUs Only"**, and **Section 6.8 "Function Level Profiling"** – Updated for use with Code Profiling plugin.
- **Section 11.3.1 "Memories to Program"** – Added effect of selecting "Erase All Before Program" on memory.
- **Section 11.3.5 "Trace and Profiling"** – Updated for use with Code Profiling plugin.
- **Section 11.3.8 "External Triggers"** Added triggers detail.
- **Section 12.4.4 "Logic Probe/External Trigger Interface"** – Added information on trigger inputs.
- **Section 12.5.2.1 "Modular Cable Specification"** – Fixed cable length description.
- **Chapter 15. "PIC32 PIMs with Trace"** – Updated PIMs with trace list.
- **Chapter 17. "MPLAB REAL ICE Isolator Unit"** – Rearranged sections and text for better understanding of content.
- **Chapter 18. "MPLAB REAL ICE JTAG Adapter (PIC32)"** – Added suggested JTAG adapter board and specified part of schematic as "Optional".

Revision D (May 2015)

- **Section 3.3 "Operational Overview"** – Updated text and table as instrumented trace now available on some PIC32 MCUs.
- **Section 4.3 "Firmware Upgrades"** – Firmware upgrade process changed; firmware version now synced to MPLAB X IDE version.
- **Section 6.8 "Function Level Profiling"** – Figure 6-12 updated to show "Time Stamp" checked.

Revision E (December 2016)

- Removed Accessories section and created separate documents for each accessory. Referenced these new documents as required throughout.
- Updated graphics for MPLAB X IDE v3.25 and later (shield motif).
- **Section 11.3.1 "Memories to Program"** - Updated information on ranges to program and/or preserve.
- **"Support"** - Added information about MySoftware account.

Support

INTRODUCTION

Please refer to the items discussed here for support issues.

- Warranty Registration
- myMicrochip Personalized Notification Service
- MySoftware Account
- The Microchip Web Site
- Microchip Forums
- Customer Support
- About Microchip Technology

WARRANTY REGISTRATION

Registering your development tool entitles you to receive new product updates. Interim software releases are available at the Microchip web site:

http://www.microchipdirect.com

myMICROCHIP PERSONALIZED NOTIFICATION SERVICE

Microchip's personal notification service helps keep customers current on their Microchip products of interest. Subscribers will receive e-mail notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool.

Please visit http://www.microchip.com/pcn to begin the registration process and select your preferences to receive personalized notifications. A FAQ and registration details are available on the page, which can be opened by selecting the link above.

When you are selecting your preferences, choosing "Development Systems" will populate the list with available development tools. The main categories of tools are listed below:

- **Compilers** – The latest information on Microchip C compilers, assemblers, linkers and other language tools. These include all MPLAB C compilers; all MPLAB assemblers (including MPASM™ assembler); all MPLAB linkers (including MPLINK™ object linker); and all MPLAB librarians (including MPLIB™ object librarian).
- **Emulators** – The latest information on Microchip in-circuit emulators.This includes the MPLAB REAL ICE™ in-circuit emulator.
- **In-Circuit Debuggers** – The latest information on Microchip in-circuit debuggers. These include the PICkit™ 2, PICkit 3 and MPLAB ICD 3 in-circuit debuggers.
- **MPLAB® IDE** – The latest information on Microchip MPLAB IDE, the Windows® Integrated Development Environment for development systems tools. This list is focused on the MPLAB IDE, MPLAB IDE Project Manager, MPLAB Editor and MPLAB SIM simulator, as well as general editing and debugging features.

- **Programmers** – The latest information on Microchip programmers. These include the device (production) programmers MPLAB REAL ICE In-Circuit Emulator, MPLAB ICD 3 In-Circuit Debugger, MPLAB PM3 and development (non-production) programmers PICkit 2 and 3.
- **Starter/Demo Boards** – These include MPLAB Starter Kit boards, PICDEM demo boards, and various other evaluation boards.

MYSOFTWARE ACCOUNT

Manage Microchip software downloads, keys and licensing from your MySoftware account. Access your account via:

- the MPLAB X IDE desktop, My MPLAB X IDE tab, "Microchip Login". Once logged in, click on "View mySoftware Account".
- microchipDirect, select My Account>My Software Products. Log into your account.
- the MySoftware URL is: https://www.microchip.com/mySoftware.

THE MICROCHIP WEB SITE

Microchip provides online support via our web site at http://www.microchip.com. This web site is used as a means to make files and information easily available to customers. Accessible by using your favorite Internet browser, the web site contains the following information:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip consultant program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

MICROCHIP FORUMS

Microchip provides additional online support by way of our web forums at http://www.microchip.com/forums. Forums that are currently available include the following subjects:

- Development Tools
- 8-bit PIC MCUs
- 16-bit PIC MCUs
- 32-bit PIC MCUs

CUSTOMER SUPPORT

Users of Microchip products can receive assistance through several channels:

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

Customers should contact their distributor, representative or field application engineer (FAE) for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in the back of this document. See our web site for a complete, up-to-date listing of sales offices.

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

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

ABOUT MICROCHIP TECHNOLOGY

Microchip Technology Inc. is a leading provider of microcontroller and analog semiconductors, providing low-risk product development, lower total system cost and faster time to market for thousands of diverse customer applications worldwide. Headquartered in Chandler, Arizona, Microchip offers outstanding technical support along with dependable delivery and quality.

Voice: (480) 792-7200

Fax: (480) 792-7277

myMicrochip: http://www.microchip.com/pcn

MySoftware: https://www.microchip.com/mySoftware

Web site: http://www.microchip.com

Forums: http://www.microchip.com/forums

Support: http://support.microchip.com

NOTES:

Glossary

A

Absolute Section

A GCC compiler section with a fixed (absolute) address that cannot be changed by the linker.

Absolute Variable/Function

A variable or function placed at an absolute address using the OCG compiler's `@ address` syntax.

Access Memory

PIC18 Only – Special registers on PIC18 devices that allow access regardless of the setting of the Bank Select Register (BSR).

Access Entry Points

Access entry points provide a way to transfer control across segments to a function which may not be defined at link time. They support the separate linking of boot and secure application segments.

Address

Value that identifies a location in memory.

Alphabetic Character

Alphabetic characters are those characters that are letters of the Roman alphabet (a, b, …, z, A, B, …, Z).

Alphanumeric

Alphanumeric characters are comprised of alphabetic characters and decimal digits (0,1, …, 9).

ANDed Breakpoints

Set up an ANDed condition for breaking, i.e., breakpoint 1 AND breakpoint 2 must occur at the same time before a program halt. This can only be accomplished if a data breakpoint and a program memory breakpoint occur at the same time.

Anonymous Structure

16-bit C Compiler **–** An unnamed structure.

PIC18 C Compiler **–** An unnamed structure that is a member of a C union. The members of an anonymous structure may be accessed as if they were members of the enclosing union. For example, in the following code, `hi` and `lo` are members of an anonymous structure inside the union `caster`.

```
union castaway
 int intval;
 struct {
  char lo; //accessible as caster.lo
  char hi; //accessible as caster.hi
 };
} caster;
```

ANSI

American National Standards Institute is an organization responsible for formulating and approving standards in the United States.

Application

A set of software and hardware that may be controlled by a PIC® microcontroller.

Archive/Archiver

An archive/library is a collection of relocatable object modules. It is created by assembling multiple source files to object files, and then using the archiver/librarian to combine the object files into one archive/library file. An archive/library can be linked with object modules and other archives/libraries to create executable code.

ASCII

American Standard Code for Information Interchange is a character set encoding that uses 7 binary digits to represent each character. It includes upper and lowercase letters, digits, symbols and control characters.

Assembly/Assembler

Assembly is a programming language that describes binary machine code in a symbolic form. An assembler is a language tool that translates assembly language source code into machine code.

Assigned Section

A GCC compiler section which has been assigned to a target memory block in the linker command file.

Asynchronously

Multiple events that do not occur at the same time. This is generally used to refer to interrupts that may occur at any time during processor execution.

Asynchronous Stimulus

Data generated to simulate external inputs to a simulator device.

Attribute

GCC Characteristics of variables or functions in a C program which are used to describe machine-specific properties.

Attribute, Section

GCC Characteristics of sections, such as “executable”, “readonly”, or “data” that can be specified as flags in the assembler `.section` directive.

B

Binary

The base two numbering system that uses the digits 0-1. The rightmost digit counts ones, the next counts multiples of 2, then $2^2 = 4$, etc.

Bookmarks

Use bookmarks to easily locate specific lines in a file.

Select Toggle Bookmarks on the Editor toolbar to add/remove bookmarks. Click other icons on this toolbar to move to the next or previous bookmark.

Breakpoint

Hardware Breakpoint: An event whose execution will cause a halt.

Software Breakpoint: An address where execution of the firmware will halt. Usually achieved by a special break instruction.

Build

Compile and link all the source files for an application.

C

C\C++

C is a general purpose programming language which features economy of expression, modern control flow and data structures, and a rich set of operators. C++ is the object-oriented version of C.

Calibration Memory

A special function register or registers used to hold values for calibration of a PIC microcontroller on-board RC oscillator or other device peripherals.

Central Processing Unit

The part of a device that is responsible for fetching the correct instruction for execution, decoding that instruction, and then executing that instruction. When necessary, it works in conjunction with the arithmetic logic unit (ALU) to complete the execution of the instruction. It controls the program memory address bus, the data memory address bus, and accesses to the stack.

Clean

Clean removes all intermediary project files, such as object, hex and debug files, for the active project. These files are recreated from other files when a project is built.

COFF

Common Object File Format. An object file of this format contains machine code, debugging and other information.

Command Line Interface

A means of communication between a program and its user based solely on textual input and output.

Compiled Stack

A region of memory managed by the compiler in which variables are statically allocated space. It replaces a software or hardware stack when such mechanisms cannot be efficiently implemented on the target device.

Compiler

A program that translates a source file written in a high-level language into machine code.

Conditional Assembly

Assembly language code that is included or omitted based on the assembly-time value of a specified expression.

Conditional Compilation

The act of compiling a program fragment only if a certain constant expression, specified by a preprocessor directive, is true.

Configuration Bits

Special-purpose bits programmed to set PIC MCU and dsPIC DSC modes of operation. A Configuration bit may or may not be preprogrammed.

Control Directives

Directives in assembly language code that cause code to be included or omitted based on the assembly-time value of a specified expression.

CPU

See Central Processing Unit.

Cross Reference File

A file that references a table of symbols and a list of files that references the symbol. If the symbol is defined, the first file listed is the location of the definition. The remaining files contain references to the symbol.

D

Data Directives

Data directives are those that control the assembler's allocation of program or data memory and provide a way to refer to data items symbolically; that is, by meaningful names.

Data Memory

On Microchip MCU and DSC devices, data memory (RAM) is comprised of General Purpose Registers (GPRs) and Special Function Registers (SFRs). Some devices also have EEPROM data memory.

Data Monitor and Control Interface (DMCI)

The Data Monitor and Control Interface, or DMCI, is a tool in MPLAB X IDE. The interface provides dynamic input control of application variables in projects. Application-generated data can be viewed graphically using any of four dynamically-assignable graph windows.

Debug/Debugger

See ICE/ICD.

Debugging Information

Compiler and assembler options that, when selected, provide varying degrees of information used to debug application code. See compiler or assembler documentation for details on selecting debug options.

Deprecated Features

Features that are still supported for legacy reasons, but will eventually be phased out and no longer used.

Device Programmer

A tool used to program electrically programmable semiconductor devices such as microcontrollers.

Digital Signal Controller

A A digital signal controller (DSC) is a microcontroller device with digital signal processing capability, i.e., Microchip dsPIC DSC devices.

Digital Signal Processing\Digital Signal Processor

Digital signal processing (DSP) is the computer manipulation of digital signals, commonly analog signals (sound or image) which have been converted to digital form (sampled). A digital signal processor is a microprocessor that is designed for use in digital signal processing.

Directives

Statements in source code that provide control of the language tool's operation.

Download

Download is the process of sending data from a host to another device, such as an emulator, programmer or target board.

DWARF

Debug With Arbitrary Record Format. DWARF is a debug information format for ELF files.

E

EEPROM

Electrically Erasable Programmable Read Only Memory. A special type of PROM that can be erased electrically. Data is written or erased one byte at a time. EEPROM retains its contents even when power is turned off.

ELF

Executable and Linking Format. An object file of this format contains machine code. Debugging and other information is specified in with DWARF. ELF/DWARF provide better debugging of optimized code than COFF.

Emulation/Emulator

See ICE/ICD.

Endianness

The ordering of bytes in a multi-byte object.

Environment

MPLAB PM3 **–** A folder containing files on how to program a device. This folder can be transferred to a SD/MMC card.

Epilogue

A portion of compiler-generated code that is responsible for deallocating stack space, restoring registers and performing any other machine-specific requirement specified in the runtime model. This code executes after any user code for a given function, immediately prior to the function return.

EPROM

Erasable Programmable Read Only Memory. A programmable read-only memory that can be erased usually by exposure to ultraviolet radiation.

Error/Error File

An error reports a problem that makes it impossible to continue processing your program. When possible, an error identifies the source file name and line number where the problem is apparent. An error file contains error messages and diagnostics generated by a language tool.

Event

A description of a bus cycle which may include address, data, pass count, external input, cycle type (fetch, R/W), and time stamp. Events are used to describe triggers, breakpoints and interrupts.

Executable Code

Software that is ready to be loaded for execution.

Export

Send data out of the MPLAB IDE/MPLAB X IDE in a standardized format.

Expressions

Combinations of constants and/or symbols separated by arithmetic or logical operators.

Extended Microcontroller Mode

In Extended Microcontroller mode, on-chip program memory as well as external memory is available. Execution automatically switches to external if the program memory address is greater than the internal memory space of the PIC18 device.

Extended Mode (PIC18 MCUs)

In Extended mode, the compiler will utilize the extended instructions (i.e., `ADDFSR`, `ADDULNK`, `CALLW`, `MOVSF`, `MOVSS`, `PUSHL`, `SUBFSR` and `SUBULNK`) and the indexed with literal offset addressing.

External Label

A label that has external linkage.

External Linkage

A function or variable has external linkage if it can be referenced from outside the module in which it is defined.

External Symbol

A symbol for an identifier which has external linkage. This may be a reference or a definition.

External Symbol Resolution

A process performed by the linker in which external symbol definitions from all input modules are collected in an attempt to resolve all external symbol references. Any external symbol references which do not have a corresponding definition cause a linker error to be reported.

External Input Line

An external input signal logic probe line (TRIGIN) for setting an event based upon external signals.

External RAM

Off-chip Read/Write memory.

F

Fatal Error

An error that will halt compilation immediately. No further messages will be produced.

File Registers

On-chip data memory, including General Purpose Registers (GPRs) and Special Function Registers (SFRs).

Filter

Determine by selection what data is included/excluded in a trace display or data file.

Fixup

The process of replacing object file symbolic references with absolute addresses after relocation by the linker.

Flash

A type of EEPROM where data is written or erased in blocks instead of bytes.

FNOP

Forced No Operation. A forced NOP cycle is the second cycle of a two-cycle instruction. Since the PIC microcontroller architecture is pipelined, it prefetches the next instruction in the physical address space while it is executing the current instruction. However, if the current instruction changes the program counter, this prefetched instruction is explicitly ignored, causing a forced NOP cycle.

Frame Pointer

A pointer that references the location on the stack that separates the stack-based arguments from the stack-based local variables. Provides a convenient base from which to access local variables and other values for the current function.

Free-Standing

An implementation that accepts any strictly conforming program that does not use complex types and in which the use of the features specified in the library clause (ANSI '89 standard clause 7) is confined to the contents of the standard headers `<float.h>`, `<iso646.h>`, `<limits.h>`, `<stdarg.h>`, `<stdbool.h>`, `<stddef.h>` and `<stdint.h>`.

G

GPR

General Purpose Register. The portion of device data memory (RAM) available for general use.

H

Halt

A stop of program execution. Executing Halt is the same as stopping at a breakpoint.

Heap

An area of memory used for dynamic memory allocation where blocks of memory are allocated and freed in an arbitrary order determined at runtime.

Hex Code\Hex File

Hex code is executable instructions stored in a hexadecimal format code. Hex code is contained in a hex file.

Hexadecimal

The base 16 numbering system that uses the digits 0-9 plus the letters A-F (or a-f). The digits A-F represent hexadecimal digits with values of (decimal) 10 to 15. The rightmost digit counts ones, the next counts multiples of 16, then $16^2 = 256$, etc.

High Level Language

A language for writing programs that is further removed from the processor than assembly.

I

ICE/ICD

In-Circuit Emulator/In-Circuit Debugger: A hardware tool that debugs and programs a target device. An emulator has more features than an debugger, such as trace.

In-Circuit Emulation/In-Circuit Debug: The act of emulating or debugging with an in-circuit emulator or debugger.

-ICE/-ICD: A device (MCU or DSC) with on-board in-circuit emulation or debug circuitry. This device is always mounted on a header board and used to debug with an in-circuit emulator or debugger.

ICSP

In-Circuit Serial Programming. A method of programming Microchip embedded devices using serial communication and a minimum number of device pins.

IDE

Integrated Development Environment, as in MPLAB IDE/MPLAB X IDE.

Identifier

A function or variable name.

IEEE

Institute of Electrical and Electronics Engineers.

Import

Bring data into the MPLAB IDE/MPLAB X IDE from an outside source, such as from a hex file.

Initialized Data

Data which is defined with an initial value. In C,

```
int myVar=5;
```

defines a variable which will reside in an initialized data section.

Instruction Set

The collection of machine language instructions that a particular processor understands.

Instructions

A sequence of bits that tells a central processing unit to perform a particular operation and can contain data to be used in the operation.

Internal Linkage

A function or variable has internal linkage if it can not be accessed from outside the module in which it is defined.

International Organization for Standardization

An organization that sets standards in many businesses and technologies, including computing and communications. Also known as ISO.

Interrupt

A signal to the CPU that suspends the execution of a running application and transfers control to an Interrupt Service Routine (ISR) so that the event may be processed. Upon completion of the ISR, normal execution of the application resumes.

Interrupt Handler

A routine that processes special code when an interrupt occurs.

Interrupt Service Request (IRQ)

An event which causes the processor to temporarily suspend normal instruction execution and to start executing an interrupt handler routine. Some processors have several interrupt request events allowing different priority interrupts.

Interrupt Service Routine (ISR)

Language tools – A function that handles an interrupt.

MPLAB IDE/MPLAB X IDE – User-generated code that is entered when an interrupt occurs. The location of the code in program memory will usually depend on the type of interrupt that has occurred.

Interrupt Vector

Address of an Interrupt Service Routine or interrupt handler.

L

L-value

An expression that refers to an object that can be examined and/or modified. An l-value expression is used on the left-hand side of an assignment.

Latency

The time between an event and its response.

Library/Librarian

See Archive/Archiver.

Linker

A language tool that combines object files and libraries to create executable code, resolving references from one module to another.

Linker Script Files

Linker script files are the command files of a linker. They define linker options and describe available memory on the target platform.

Listing Directives

Listing directives are those directives that control the assembler listing file format. They allow the specification of titles, pagination and other listing control.

Listing File

A listing file is an ASCII text file that shows the machine code generated for each C source statement, assembly instruction, assembler directive, or macro encountered in a source file.

Little Endian

A data ordering scheme for multibyte data whereby the Least Significant Byte is stored at the lower addresses.

Local Label

A local label is one that is defined inside a macro with the LOCAL directive. These labels are particular to a given instance of a macro's instantiation. In other words, the symbols and labels that are declared as local are no longer accessible after the ENDM macro is encountered.

Logic Probes

Up to 14 logic probes can be connected to some Microchip emulators. The logic probes provide external trace inputs, trigger output signal, +5V, and a common ground.

Loopback Test Board

Used to test the functionality of the MPLAB REAL ICE in-circuit emulator.

LVDS

Low Voltage Differential Signaling. A low noise, low-power, low amplitude method for high-speed (gigabits per second) data transmission over copper wire.

With standard I/O signaling, data storage is contingent upon the actual voltage level. Voltage level can be affected by wire length (longer wires increase resistance, which lowers voltage). But with LVDS, data storage is distinguished only by positive and negative voltage values, not the voltage level. Therefore, data can travel over greater lengths of wire while maintaining a clear and consistent data stream.

Source: http://www.webopedia.com/TERM/L/LVDS.html

M

Machine Code

The representation of a computer program that is actually read and interpreted by the processor. A program in binary machine code consists of a sequence of machine instructions (possibly interspersed with data). The collection of all possible instructions for a particular processor is known as its "instruction set".

Machine Language

A set of instructions for a specific central processing unit, designed to be usable by a processor without being translated.

Macro

Macro instruction. An instruction that represents a sequence of instructions in abbreviated form.

Macro Directives

Directives that control the execution and data allocation within macro body definitions.

Makefile

Export to a file the instructions to Make the project. Use this file to Make your project outside of MPLAB IDE/MPLAB X IDE, i.e., with a `make`.

Make Project

A command that rebuilds an application, recompiling only those source files that have changed since the last complete compilation.

MCU

Microcontroller Unit. An abbreviation for microcontroller. Also uC.

Memory Model

For C compilers, a representation of the memory available to the application. For the PIC18 C compiler, a description that specifies the size of pointers that point to program memory.

Message

Text displayed to alert you to potential problems in language tool operation. A message will not stop operation.

Microcontroller

A highly integrated chip that contains a CPU, RAM, program memory, I/O ports and timers.

Microcontroller Mode

One of the possible program memory configurations of PIC18 microcontrollers. In Microcontroller mode, only internal execution is allowed. Thus, only the on-chip program memory is available in Microcontroller mode.

Microprocessor Mode

One of the possible program memory configurations of PIC18 microcontrollers. In Microprocessor mode, the on-chip program memory is not used. The entire program memory is mapped externally.

Mnemonics

Text instructions that can be translated directly into machine code. Also referred to as opcodes.

Module

The preprocessed output of a source file after preprocessor directives have been executed. Also known as a translation unit.

MPASM™ Assembler

Microchip Technology’s relocatable macro assembler for PIC microcontroller devices, KeeLoq® devices and Microchip memory devices.

MPLAB *Language Tool* for *Device*

Microchip’s C compilers, assemblers and linkers for specified devices. Select the type of language tool based on the device you will be using for your application, e.g., if you will be creating C code on a PIC18 MCU, select the MPLAB C Compiler for PIC18 MCUs.

MPLAB ICD

Microchip in-circuit debugger that works with MPLAB IDE/MPLAB X IDE. *See* ICE/ICD.

MPLAB IDE/MPLAB X IDE

Microchip's Integrated Development Environment. MPLAB IDE/MPLAB X IDE comes with an editor, project manager and simulator.

MPLAB PM3

A device programmer from Microchip. Programs PIC18 microcontrollers and dsPIC digital signal controllers. Can be used with MPLAB IDE/MPLAB X IDE or stand-alone. Replaces PRO MATE II.

MPLAB REAL ICE™ In-Circuit Emulator

Microchip's next-generation in-circuit emulator that works with MPLAB IDE/MPLAB X IDE. *See* ICE/ICD.

MPLAB SIM

Microchip's simulator that works with MPLAB IDE/MPLAB X IDE in support of PIC MCU and dsPIC DSC devices.

MPLAB Starter Kit for *Device*

Microchip's starter kits contains everything needed to begin exploring the specified device. View a working application and then debug and program your own changes.

MPLIB™ Object Librarian

Microchip's librarian that can work with MPLAB IDE/MPLAB X IDE. MPLIB librarian is an object librarian for use with COFF object modules created using either MPASM assembler (mpasm or mpasmwin v2.0) or MPLAB C18 C Compiler.

MPLINK™ Object Linker

MPLINK linker is an object linker for the Microchip MPASM assembler and the Microchip C18 C compiler. MPLINK linker also may be used with the Microchip MPLIB librarian. MPLINK linker is designed to be used with MPLAB IDE/MPLAB X IDE, though it does not have to be.

MRU

Most Recently Used. Refers to files and windows available to be selected from MPLAB IDE/MPLAB X IDE main pull down menus.

N

Native Data Size

For Native trace, the size of the variable used in a Watches window must be of the same size as the selected device's data memory: bytes for PIC18 devices and words for 16-bit devices.

Nesting Depth

The maximum level to which macros can include other macros.

Node

MPLAB IDE/MPLAB X IDE project component.

Non-Extended Mode (PIC18 MCUs)

In Non-Extended mode, the compiler will not utilize the extended instructions nor the indexed with literal offset addressing.

Non Real Time

Refers to the processor at a breakpoint or executing single-step instructions or MPLAB IDE/MPLAB X IDE being run in Simulator mode.

Non-Volatile Storage

A storage device whose contents are preserved when its power is off.

NOP

No Operation. An instruction that has no effect when executed except to advance the program counter.

O

Object Code/Object File

Object code is the machine code generated by an assembler or compiler. An object file is a file containing machine code and possibly debug information. It may be immediately executable or it may be relocatable, requiring linking with other object files, e.g., libraries, to produce a complete executable program.

Object File Directives

Directives that are used only when creating an object file.

Octal

The base 8 number system that only uses the digits 0-7. The rightmost digit counts ones, the next digit counts multiples of 8, then $8^2 = 64$, etc.

Off-Chip Memory

Off-chip memory refers to the memory selection option for the PIC18 device where memory may reside on the target board, or where all program memory may be supplied by the emulator. The **Memory** tab accessed from *Options>Development Mode* provides the Off-Chip Memory selection dialog box.

Opcodes

Operational Codes. *See* Mnemonics.

Operators

Symbols, like the plus sign '+' and the minus sign '-', that are used when forming well-defined expressions. Each operator has an assigned precedence that is used to determine order of evaluation.

OTP

One Time Programmable. EPROM devices that are not in windowed packages. Since EPROM needs ultraviolet light to erase its memory, only windowed devices are erasable.

P

Pass Counter

A counter that decrements each time an event (such as the execution of an instruction at a particular address) occurs. When the pass count value reaches zero, the event is satisfied. You can assign the Pass Counter to break and trace logic, and to any sequential event in the complex trigger dialog.

PC

Personal Computer or Program Counter.

PC Host

Any PC running a supported Windows operating system.

Persistent Data

Data that is never cleared or initialized. Its intended use is so that an application can preserve data across a device Reset.

Phantom Byte

An unimplemented byte in the dsPIC architecture that is used when treating the 24-bit instruction word as if it were a 32-bit instruction word. Phantom bytes appear in dsPIC hex files.

PIC MCUs

PIC microcontrollers (MCUs) refers to all Microchip microcontroller families.

PICkit 2 and 3

Microchip's developmental device programmers with debug capability through Debug Express. See the Readme files for each tool to see which devices are supported.

Plug-ins

The MPLAB IDE/MPLAB X IDE has both built-in components and plug-in modules to configure the system for a variety of software and hardware tools. Several plug-in tools may be found under the Tools menu.

Pod

The enclosure for an in-circuit emulator or debugger. Other names are "Puck", if the enclosure is round, and "Probe", not be confused with logic probes.

Power-on-Reset Emulation

A software randomization process that writes random values in data RAM areas to simulate uninitialized values in RAM upon initial power application.

Pragma

A directive that has meaning to a specific compiler. Often a pragma is used to convey implementation-defined information to the compiler.

Precedence

Rules that define the order of evaluation in expressions.

Production Programmer

A production programmer is a programming tool that has resources designed in to program devices rapidly. It has the capability to program at various voltage levels and completely adheres to the programming specification. Programming a device as fast as possible is of prime importance in a production environment where time is of the essence as the application circuit moves through the assembly line.

Profile

For MPLAB SIM simulator, a summary listing of executed stimulus by register.

Program Counter

The location that contains the address of the instruction that is currently executing.

Program Counter Unit

16-bit assembler – A conceptual representation of the layout of program memory. The program counter increments by 2 for each instruction word. In an executable section, 2 program counter units are equivalent to 3 bytes. In a read-only section, 2 program counter units are equivalent to 2 bytes.

Program Memory

MPLAB IDE/MPLAB X IDE – The memory area in a device where instructions are stored. Also, the memory in the emulator or simulator containing the downloaded target application firmware.

16-bit assembler/compiler – The memory area in a device where instructions are stored.

Project

A project contains the files needed to build an application (source code, linker script files, etc.) along with their associations to various build tools and build options.

Prologue

A portion of compiler-generated code that is responsible for allocating stack space, preserving registers and performing any other machine-specific requirement specified in the runtime model. This code executes before any user code for a given function.

Prototype System

A term referring to a user's target application, or target board.

Psect

The OCG equivalent of a GCC section, short for program section. A block of code or data which is treated as a whole by the linker.

PWM Signals

Pulse Width Modulation Signals. Certain PIC MCU devices have a PWM peripheral.

Q

Qualifier

An address or an address range used by the Pass Counter or as an event before another operation in a complex trigger.

R

Radix

The number base, hex, or decimal, used in specifying an address.

RAM

Random Access Memory (Data Memory). Memory in which information can be accessed in any order.

Raw Data

The binary representation of code or data associated with a section.

Read Only Memory

Memory hardware that allows fast access to permanently stored data but prevents addition to or modification of the data.

Real Time

When an in-circuit emulator or debugger is released from the Halt state, the processor runs in Real Time mode and behaves exactly as the normal chip would behave. In Real Time mode, the real time trace buffer of an emulator is enabled and constantly captures all selected cycles, and all break logic is enabled. In an in-circuit emulator or debugger, the processor executes in real time until a valid breakpoint causes a halt, or until the user halts the execution.

In the simulator, real time simply means execution of the microcontroller instructions as fast as they can be simulated by the host CPU.

Recursive Calls

A function that calls itself, either directly or indirectly.

Recursion

The concept that a function or macro, having been defined, can call itself. Great care should be taken when writing recursive macros; it is easy to get caught in an infinite loop where there will be no exit from the recursion.

Reentrant

A function that may have multiple, simultaneously active instances. This may happen due to either direct or indirect recursion or through execution during interrupt processing.

Relaxation

The process of converting an instruction to an identical, but smaller instruction. This is useful for saving on code size. MPLAB XC16 currently knows how to `relax` a `CALL` instruction into an `RCALL` instruction. This is done when the symbol that is being called is within +/- 32k instruction words from the current instruction.

Relocatable

An object whose address has not been assigned to a fixed location in memory.

Relocatable Section

16-bit assembler – A section whose address is not fixed (absolute). The linker assigns addresses to relocatable sections through a process called relocation.

Relocation

A process performed by the linker in which absolute addresses are assigned to relocatable sections and all symbols in the relocatable sections are updated to their new addresses.

ROM

Read Only Memory (Program Memory). Memory that cannot be modified.

Run

The command that releases the emulator from halt, allowing it to run the application code and change or respond to I/O in real time.

Run-time Model

Describes the use of target architecture resources.

Runtime Watch

A Watch window where the variables change in as the application is run. See individual tool documentation to determine how to set up a runtime watch. Not all tools support runtime watches.

S

Scenario

For MPLAB SIM simulator, a particular setup for stimulus control.

Section

The GCC equivalent of an OCG psect. A block of code or data which is treated as a whole by the linker.

Section Attribute

A GCC characteristic ascribed to a section (e.g., an `access` section).

Sequenced Breakpoints

Breakpoints that occur in a sequence. Sequence execution of breakpoints is bottom-up; the last breakpoint in the sequence occurs first.

Serialized Quick Turn Programming

Serialization allows you to program a serial number into each microcontroller device that the Device Programmer programs. This number can be used as an entry code, password or ID number.

Shell

The MPASM assembler shell is a prompted input interface to the macro assembler. There are two MPASM assembler shells: one for the DOS version and one for the Windows operating system version.

Simulator

A software program that models the operation of devices.

Single Step

This command steps though code, one instruction at a time. After each instruction, MPLAB IDE/MPLAB X IDE updates register windows, watch variables, and status displays so you can analyze and debug instruction execution. You can also single step C compiler source code, but instead of executing single instructions, MPLAB IDE/MPLAB X IDE will execute all assembly level instructions generated by the line of the high level C statement.

Skew

The information associated with the execution of an instruction appears on the processor bus at different times. For example, the executed opcodes appears on the bus as a fetch during the execution of the previous instruction, the source data address and value and the destination data address appear when the opcodes is actually executed, and the destination data value appears when the next instruction is executed. The trace buffer captures the information that is on the bus at one instance. Therefore, one trace buffer entry will contain execution information for three instructions. The number of captured cycles from one piece of information to another for a single instruction execution is referred to as the skew.

Skid

When a hardware breakpoint is used to halt the processor, one or more additional instructions may be executed before the processor halts. The number of extra instructions executed after the intended breakpoint is referred to as the skid.

Source Code

The form in which a computer program is written by the programmer. Source code is written in a formal programming language which can be translated into machine code or executed by an interpreter.

Source File

An ASCII text file containing source code.

Special Function Registers (SFRs)

The portion of data memory (RAM) dedicated to registers that control I/O processor functions, I/O status, timers or other modes or peripherals.

SQTP

See Serialized Quick Turn Programming.

Stack, Hardware

Locations in PIC microcontroller where the return address is stored when a function call is made.

Stack, Software

Memory used by an application for storing return addresses, function parameters, and local variables. This memory is dynamically allocated at runtime by instructions in the program. It allows for reentrant function calls.

Stack, Compiled

A region of memory managed and allocated by the compiler in which variables are statically assigned space. It replaces a software stack when such mechanisms cannot be efficiently implemented on the target device. It precludes reentrancy.

Static RAM or SRAM

Static Random Access Memory. Program memory you can read/write on the target board that does not need refreshing frequently.

Status Bar

The Status Bar is located on the bottom of the MPLAB IDE/MPLAB X IDE window and indicates such current information as cursor position, development mode and device, and active tool bar.

Step Into

This command is the same as Single Step. Step Into (as opposed to Step Over) follows a CALL instruction into a subroutine.

Step Over

Step Over allows you to debug code without stepping into subroutines. When stepping over a `CALL` instruction, the next breakpoint will be set at the instruction after the `CALL`. If for some reason the subroutine gets into an endless loop or does not return properly, the next breakpoint will never be reached. The Step Over command is the same as Single Step except for its handling of `CALL` instructions.

Step Out

Step Out allows you to step out of a subroutine which you are currently stepping through. This command executes the rest of the code in the subroutine and then stops execution at the return address to the subroutine.

Stimulus

Input to the simulator, i.e., data generated to exercise the response of simulation to external signals. Often the data is put into the form of a list of actions in a text file. Stimulus may be asynchronous, synchronous (pin), clocked and register.

Stopwatch

A counter for measuring execution cycles.

Storage Class

Determines the lifetime of the memory associated with the identified object.

Storage Qualifier

Indicates special properties of the objects being declared (e.g., `const`).

Symbol

A symbol is a general purpose mechanism for describing the various pieces which comprise a program. These pieces include function names, variable names, section names, file names, struct/enum/union tag names, etc. Symbols in MPLAB IDE/MPLAB X IDE refer mainly to variable names, function names and assembly labels. The value of a symbol after linking is its value in memory.

Symbol, Absolute

Represents an immediate value such as a definition through the assembly `.equ` directive.

System Window Control

The system window control is located in the upper left corner of windows and some dialogs. Clicking on this control usually pops up a menu that has the items “Minimize,” “Maximize,” and “Close.”

T

Target

Refers to user hardware.

Target Application

Software residing on the target board.

Target Board

The circuitry and programmable device that makes up the target application.

Target Processor

The microcontroller device on the target application board.

Template

Lines of text that you build for inserting into your files at a later time. The MPLAB Editor stores templates in template files.

Tool Bar

A row or column of icons that you can click on to execute MPLAB IDE/MPLAB X IDE functions.

Trace

An emulator or simulator function that logs program execution. The emulator logs program execution into its trace buffer which is uploaded to the MPLAB IDE/MPLAB X IDE trace window.

Trace Memory

Trace memory contained within the emulator. Trace memory is sometimes called the trace buffer.

Trace Macro

A macro that will provide trace information from emulator data. Since this is a software trace, the macro must be added to code, the code must be recompiled or reassembled, and the target device must be programmed with this code before trace will work.

Trigger Output

Trigger output refers to an emulator output signal that can be generated at any address or address range, and is independent of the trace and breakpoint settings. Any number of trigger output points can be set.

Trigraphs

Three-character sequences, all starting with ??, that are defined by ISO C as replacements for single characters.

U

Unassigned Section

A section which has not been assigned to a specific target memory block in the linker command file. The linker must find a target memory block in which to allocate an unassigned section.

Uninitialized Data

Data which is defined without an initial value. In C,

```
int myVar;
```

defines a variable which will reside in an uninitialized data section.

Upload

The Upload function transfers data from a tool, such as an emulator or programmer, to the host PC or from the target board to the emulator.

USB

Universal Serial Bus. An external peripheral interface standard for communication between a computer and external peripherals over a cable using bi-serial transmission. USB 1.0/1.1 supports data transfer rates of 12 Mbps. Also referred to as high-speed USB, USB 2.0 supports data rates up to 480 Mbps.

V

Vector

The memory locations that an application will jump to when either a Reset or interrupt occurs.

Volatile

A variable qualifier which prevents the compiler applying optimizations that affect how the variable is accessed in memory.

W

Warning

MPLAB IDE/MPLAB X IDE – An alert that is provided to warn you of a situation that would cause physical damage to a device, software file, or equipment.

16-bit assembler/compiler – Warnings report conditions that may indicate a problem, but do not halt processing.

Watch Variable

A variable that you may monitor during a debugging session in a Watches window.

Watch Window

Watch windows contain a list of watch variables that are updated at each breakpoint.

Watchdog Timer (WDT)

A timer on a PIC microcontroller that resets the processor after a selectable length of time. The WDT is enabled or disabled and set up using Configuration bits.

Workbook

For MPLAB SIM stimulator, a setup for generation of SCL stimulus.

NOTES:

Index

MICROCHIP

Worldwide Sales and Service

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://www.microchip.com/support
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Austin, TX
Tel: 512-257-3370

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Novi, MI
Tel: 248-848-4000

Houston, TX
Tel: 281-894-5983

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453
Tel: 317-536-2380

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608
Tel: 951-273-7800

Raleigh, NC
Tel: 919-844-7510

New York, NY
Tel: 631-435-6000

San Jose, CA
Tel: 408-735-9110
Tel: 408-436-4270

Canada - Toronto
Tel: 905-695-1980
Fax: 905-695-2078

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon

Hong Kong
Tel: 852-2943-5100
Fax: 852-2401-3431

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8569-7000
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Chongqing
Tel: 86-23-8980-9588
Fax: 86-23-8980-9500

China - Dongguan
Tel: 86-769-8702-9880

China - Guangzhou
Tel: 86-20-8755-8029

China - Hangzhou
Tel: 86-571-8792-8115
Fax: 86-571-8792-8116

China - Hong Kong SAR
Tel: 852-2943-5100
Fax: 852-2401-3431

China - Nanjing
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-3326-8000
Fax: 86-21-3326-8021

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8864-2200
Fax: 86-755-8203-1760

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xian
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

ASIA/PACIFIC

China - Xiamen
Tel: 86-592-2388138
Fax: 86-592-2388130

China - Zhuhai
Tel: 86-756-3210040
Fax: 86-756-3210049

India - Bangalore
Tel: 91-80-3090-4444
Fax: 91-80-3090-4123

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-3019-1500

Japan - Osaka
Tel: 81-6-6152-7160
Fax: 81-6-6152-9310

Japan - Tokyo
Tel: 81-3-6880- 3770
Fax: 81-3-6880-3771

Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-5778-366
Fax: 886-3-5770-955

Taiwan - Kaohsiung
Tel: 886-7-213-7830

Taiwan - Taipei
Tel: 886-2-2508-8600
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

Finland - Espoo
Tel: 358-9-4520-820

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

France - Saint Cloud
Tel: 33-1-30-60-70-00

Germany - Garching
Tel: 49-8931-9700

Germany - Haan
Tel: 49-2129-3766400

Germany - Heilbronn
Tel: 49-7131-67-3636

Germany - Karlsruhe
Tel: 49-721-625370

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Germany - Rosenheim
Tel: 49-8031-354-560

Israel - Ra'anana
Tel: 972-9-744-7705

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Italy - Padova
Tel: 39-049-7625286

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Norway - Trondheim
Tel: 47-7289-7561

Poland - Warsaw
Tel: 48-22-3325737

Romania - Bucharest
Tel: 40-21-407-87-50

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

Sweden - Gothenberg
Tel: 46-31-704-60-40

Sweden - Stockholm
Tel: 46-8-5090-4654

UK - Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820

11/07/16