



**MPLAB REAL ICE  
In-Circuit Emulator  
User's Guide**

---

**Note the following details of the code protection feature on Microchip devices:**

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

---

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights.

**Trademarks**

The Microchip name and logo, the Microchip logo, dsPIC, KEELOQ, KEELOQ logo, MPLAB, PIC, PICmicro, PICSTART, rfPIC and UNI/O are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.


FilterLab, Hampshire, HI-TECH C, Linear Active Thermistor, MXDEV, MXLAB, SEEVAL and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICworks, dsSPEAK, ECAN, ECONOMONITOR, FanSense, HI-TIDE, In-Circuit Serial Programming, ICSP, Mindi, MiWi, MPASM, MPLAB Certified logo, MPLIB, MPLINK, mTouch, Octopus, Omniscient Code Generation, PICC, PICC-18, PICDEM, PICDEM.net, PICKit, PICtail, PIC<sup>32</sup> logo, REAL ICE, rLAB, Select Mode, Total Endurance, TSHARC, UniWinDriver, WiperLock and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

© 2009, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

 Printed on recycled paper.

**QUALITY MANAGEMENT SYSTEM**  
**CERTIFIED BY DNV**  
**== ISO/TS 16949:2002 ==**

*Microchip received ISO/TS-16949:2002 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC<sup>®</sup> MCUs and dsPIC<sup>®</sup> DSCs, KEELOQ<sup>®</sup> code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.*



---

---

## Table of Contents

---

---

|  |    |
|--|----|
| Preface .....  | 7  |
| <b>Part 1 – Overview</b>                                 |    |
| <b>Chapter 1. About the Emulator</b>                     |    |
| 1.1 Introduction .....                                   | 15 |
| 1.2 Emulator Defined .....                               | 15 |
| 1.3 How the Emulator Helps You .....                     | 16 |
| 1.4 Emulator Kit Components .....                        | 16 |
| 1.5 Device and Feature Support .....                     | 16 |
| <b>Chapter 2. Operation</b>                              |    |
| 2.1 Introduction .....                                   | 17 |
| 2.2 Tool Comparisons .....                               | 17 |
| 2.3 Emulator Communications with the PC and Target ..... | 18 |
| 2.4 Target Communication Connections .....               | 21 |
| 2.5 Trace Connections .....                              | 24 |
| 2.6 Debugging with the Emulator .....                    | 27 |
| 2.7 Requirements For Debugging .....                     | 27 |
| 2.8 Programming with the Emulator .....                  | 30 |
| 2.9 Resources Used by the Emulator .....                 | 30 |
| <b>Part 2 – Getting Started</b>                          |    |
| <b>Chapter 3. Installation</b>                           |    |
| 3.1 Introduction .....                                   | 33 |
| 3.2 Installing the Software .....                        | 33 |
| 3.3 Installing the USB Device Drivers .....              | 33 |
| 3.4 Selecting Target Communications .....                | 34 |
| 3.5 Setting Up the Target .....                          | 35 |
| 3.6 Connecting the Logic Probes .....                    | 36 |
| 3.7 Setting Up MPLAB IDE .....                           | 36 |
| <b>Chapter 4. Tutorial</b>                               |    |
| 4.1 Introduction .....                                   | 37 |
| 4.2 Setting Up The Environment .....                     | 37 |
| 4.3 Creating the Application Code .....                  | 38 |
| 4.4 Running the Project Wizard .....                     | 41 |
| 4.5 Viewing the Project .....                            | 42 |
| 4.6 Creating a Hex File .....                            | 43 |
| 4.7 Viewing Debug Options .....                          | 44 |

|  |    |
|--|----|
| 4.8 Setting Up the Demo Board .....                      | 46 |
| 4.9 Loading Program Code For Debugging .....             | 46 |
| 4.10 Running Debug Code .....                            | 47 |
| 4.11 Debugging Code Using Breakpoints .....              | 47 |
| 4.12 Debugging Code Using A RunTime Watch .....          | 53 |
| 4.13 Debugging Code Using Native Trace .....             | 54 |
| 4.14 Programming the Application .....                   | 57 |
| 4.15 Other Trace Methods – SPI or I/O Port Trace .....   | 58 |
| 4.16 Other Trace Methods – PIC32 Instruction Trace ..... | 63 |

## Part 3 – Features

---

### Chapter 5. General Setup

|  |    |
|--|----|
| 5.1 Introduction .....                                       | 67 |
| 5.2 Starting the MPLAB IDE Software .....                    | 67 |
| 5.3 Creating a Project .....                                 | 68 |
| 5.4 Viewing the Project .....                                | 68 |
| 5.5 Building the Project .....                               | 69 |
| 5.6 Setting Configuration Bits .....                         | 69 |
| 5.7 Setting the Emulator as the Debugger or Programmer ..... | 69 |
| 5.8 Quick Debug/Program Reference .....                      | 70 |
| 5.9 Debugger/Programmer Limitations .....                    | 71 |

### Chapter 6. Basic Debug Functions

|                                     |    |
|-------------------------------------|----|
| 6.1 Introduction .....              | 73 |
| 6.2 Breakpoints and Stopwatch ..... | 73 |
| 6.3 External Triggers .....         | 74 |

### Chapter 7. Debug for 8- and 16-Bit Devices

|  |    |
|--|----|
| 7.1 Introduction .....                     | 75 |
| 7.2 Data Capture and Runtime Watches ..... | 75 |
| 7.3 Trace .....                            | 76 |

### Chapter 8. Debug for 32-Bit Devices

|  |    |
|--|----|
| 8.1 Introduction .....                     | 83 |
| 8.2 Data Capture and Runtime Watches ..... | 83 |
| 8.3 PIC32 Instruction Trace .....          | 84 |

## Part 4 – Troubleshooting

---

### Chapter 9. Troubleshooting First Steps

|   |    |
|---|----|
| 9.1 Introduction .....                    | 91 |
| 9.2 The 5 Questions to Answer First ..... | 91 |
| 9.3 Top Reasons Why You Can't Debug ..... | 91 |
| 9.4 Other Things to Consider .....        | 92 |

## Chapter 10. Frequently Asked Questions (FAQ)

|   |    |
|---|----|
| 10.1 Introduction .....                           | 93 |
| 10.2 How The Emulator Works .....                 | 93 |
| 10.3 How Trace Works – 8 and 16 Bit Devices ..... | 95 |
| 10.4 General Issues .....                         | 96 |

## Chapter 11. Error Messages

|                                       |     |
|---------------------------------------|-----|
| 11.1 Introduction .....               | 99  |
| 11.2 Specific Error Messages .....    | 99  |
| 11.3 General Corrective Actions ..... | 103 |

## Part 5 – Reference

---

## Chapter 12. Emulator Function Summary

|                                      |     |
|--------------------------------------|-----|
| 12.1 Introduction .....              | 109 |
| 12.2 Debugging Functions .....       | 109 |
| 12.3 Debugging Dialogs/Windows ..... | 112 |
| 12.4 Programming Functions .....     | 120 |
| 12.5 Settings Dialog .....           | 121 |
| 12.6 Saved Information .....         | 125 |

## Chapter 13. Hardware Specification

|  |     |
|--|-----|
| 13.1 Introduction .....                      | 127 |
| 13.2 Highlights .....                        | 127 |
| 13.3 Declaration of Conformity .....         | 127 |
| 13.4 USB Port/Power .....                    | 128 |
| 13.5 Emulator Pod .....                      | 128 |
| 13.6 Standard Communication Hardware .....   | 130 |
| 13.7 High-Speed Communication Hardware ..... | 132 |
| 13.8 MPLAB REAL ICE Isolator unit .....      | 137 |
| 13.9 Loop-Back Test Board .....              | 141 |
| 13.10 Target Board Considerations .....      | 141 |

## Appendix A. Revision History .....

## Glossary .....

## Index .....

## Worldwide Sales and Service .....

# MPLAB® REAL ICE™ In-Circuit Emulator User's Guide

---

NOTES:



# MPLAB® REAL ICE™ IN-CIRCUIT EMULATOR USER'S GUIDE

---

---

## 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](http://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 “DSXXXXA”, where “XXXX” 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® IDE on-line help. Select the Help menu, and then Topics to open a list of available on-line 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 include:

- Document Layout
- Conventions Used in this Guide
- Warranty Registration
- Recommended Reading
- The Microchip Web Site
- Development Systems Customer Change Notification Service
- Customer Support

## 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

- **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: Operation** – The theory of MPLAB REAL ICE in-circuit emulator operation. Explains configuration options.

### Part 2 – Getting Started

- **Chapter 3: Installation** – How to install the emulator software and hardware.
- **Chapter 4: Tutorial** – A brief tutorial on using the emulator.

### Part 3 – Features

- **Chapter 5: General Setup** – How to set up MPLAB IDE to use the emulator.
- **Chapter 6: Basic Debug Functions** – A description of basic emulator features available in MPLAB IDE when the MPLAB REAL ICE in-circuit emulator is chosen as the debug tool. This includes the debug features breakpoints, stopwatch, and external triggering.
- **Chapter 7: Debug for 8- and 16-Bit Devices** – A description of data capture, runtime watches and trace for 8- and 16-bit (data memory) devices. Includes the types of trace available and how to setup and use trace.
- **Chapter 8: Debug for 32-Bit Devices** – A description of data capture, runtime watches and trace for 32-bit devices. Includes hardware and software setup for use of PIC32 instruction trace.

### Part 4 – Troubleshooting

- **Chapter 9: Troubleshooting First Steps** – First steps to take if you have problems with the MPLAB REAL ICE in-circuit emulator.
- **Chapter 10: Frequently Asked Questions** – A list of frequently-asked questions, useful for troubleshooting.
- **Chapter 11: Error Messages** – A list of error messages and suggested resolutions.

### Part 5 – 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 loop-back test board.



## CONVENTIONS USED IN THIS GUIDE

The following conventions may appear in this documentation:

**TABLE 1: DOCUMENTATION CONVENTIONS**

| Description                                      | Represents  | Examples  |
|--|---|---|
| <b>Arial font:</b>                               |   |   |
| Italic   | Referenced books  | <i>MPLAB<sup>®</sup> IDE User's Guide</i>                   |
|  | Emphasized text   | ...is the <i>only</i> 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><i>File&gt;Save</i></u>                                  |
| Bold   | A dialog button   | Click <b>OK</b>   |
|  | A tab   | Click the <b>Power</b> tab                                  |
| Text in angle brackets < >                       | A key on the keyboard                                   | Press <Enter>, <F1>   |
| <b>Courier font:</b>                             |   |   |
| 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                                     | <i>file.o</i> , where <i>file</i> can be any valid filename |
| Square brackets [ ]                              | Optional arguments                                      | mpasmwin [options]<br><i>file</i> [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)<br>{ ...<br>}                              |

## WARRANTY REGISTRATION

Please complete the enclosed Warranty Registration Card and mail it promptly. Sending in your Warranty Registration Card entitles you to receive new product updates. Interim software releases are available at the Microchip web site.

## RECOMMENDED READING

This document describes how to use the MPLAB REAL ICE in-circuit emulator. Other useful documents are listed below. The following Microchip documents are available and recommended as supplemental reference resources.

### **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) in the Readmes subdirectory of the MPLAB IDE installation directory. The release notes (Readme) contains update information and known issues that may not be included in this user's guide.

### **Using the MPLAB REAL ICE In-Circuit Emulator (DS51749)**

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 Isolation Unit Setup (DS51858)**

This poster shows you how to hook up the opto-isolation unit hardware for high power applications.

### **MPLAB REAL ICE In-Circuit Emulator On-line Help File**

A comprehensive help file for the emulator is included with MPLAB IDE. Usage, troubleshooting and hardware specifications are covered. This may be more up-to-date than the printed documentation. Also, emulator reserved resources and limitations are listed for various devices.

### **Header Board Specification (DS51292)**

This booklet describes how to install and use MPLAB REAL ICE in-circuit emulator headers. Headers are used to better debug selected devices using special -ICE device versions, without the loss of pins or resources. See also the Header on-line help file.

### **Transition Socket Specification (DS51194)**

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

## THE MICROCHIP WEB SITE

Microchip provides online support via our web site at [www.microchip.com](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

## DEVELOPMENT SYSTEMS CUSTOMER CHANGE NOTIFICATION SERVICE

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

To register, access the Microchip web site at [www.microchip.com](http://www.microchip.com), click on Customer Change Notification and follow the registration instructions.

The Development Systems product group categories are:

- **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. These include the MPLAB REAL ICE™ and MPLAB ICE 2000 in-circuit emulators
- **In-Circuit Debuggers** – The latest information on Microchip in-circuit debuggers. These include the MPLAB ICD 2 and 3 in-circuit debuggers and PICkit™ 2 and 3 debug express.
- **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 PRO MATE® II and development (nonproduction) programmers MPLAB ICD 2 in-circuit debugger, PICSTART® Plus and PICkit 1, 2 and 3.

## CUSTOMER SUPPORT

Users of Microchip products can receive assistance through several channels:

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

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.

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



# MPLAB® REAL ICE™ IN-CIRCUIT EMULATOR USER'S GUIDE

---

---

## Part 1 – Overview

---

---

|                                     |    |
|-------------------------------------|----|
| Chapter 1. About the Emulator ..... | 15 |
| Chapter 2. Operation.....           | 17 |

# MPLAB® REAL ICE™ In-Circuit Emulator User's Guide

---

NOTES:



# MPLAB® REAL ICE™ IN-CIRCUIT EMULATOR USER'S GUIDE

---

---

## Chapter 1. About the Emulator

---

---

### 1.1 INTRODUCTION

The MPLAB REAL ICE in-circuit emulator system is described.

- Emulator Defined
- How the Emulator Helps You
- Emulator Kit Components
- Device and Feature Support

### 1.2 EMULATOR DEFINED

The MPLAB REAL ICE in-circuit emulator is an in-circuit emulator that is controlled by a PC running MPLAB IDE software on a Windows® platform. The MPLAB REAL ICE in-circuit emulator is an integral part of the development engineer's toolsuite. The application usage can vary from software development to hardware integration to manufacturing test to field service.

The MPLAB REAL ICE in-circuit emulator is a modern emulator system that supports hardware and software development for selected Microchip PIC® microcontrollers (MCUs) and dsPIC® Digital Signal Controllers (DSCs) that are based on In-Circuit Serial Programming™ (ICSP™) programming capability and Standard DUT Programming (STDP) 2-wire serial interfaces.

The emulator system will execute code in an actual device because these Microchip devices have built-in emulation circuitry, instead of a special emulator chip, for emulation. All available features of a given device are accessible interactively, and can be set and modified by the MPLAB IDE interface.

The MPLAB REAL ICE emulation concept has 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 port data input
- Trace (MPLAB IDE and Compiler Assisted) – 8 and 16-bit devices
- Hardware Trace – 32-bit devices

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

## 1.3 HOW THE EMULATOR HELPS YOU

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

- Debug application on hardware in real time
- Debug with hardware breakpoints
- Debug with software breakpoints (device-dependent)
- Set breakpoints based on internal and/or external signals
- Monitor internal file registers
- Emulate full speed
- Program device
- 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 are listed below.

1. CD-ROM with MPLAB IDE software and on-line documentation
2. Emulator pod
3. USB cable to provide communications between the emulator and a PC and to provide power to the emulator
4. Standard driver board and cable to connect the emulator pod to a header module or target board
5. Logic probes
6. Loop-back test board

Additional hardware that may be ordered separately:

7. Processor Extension Pak: High-speed driver board, ICE header/receiver board, high-speed to standard converter board, and cables to connect the emulator pod to a target board
8. Performance Pak: High-speed driver board, high-speed receiver board, high-speed to standard converter board, and cables to connect the emulator pod to a target board
9. Transition socket
10. MPLAB REAL ICE Isolator unit

## 1.5 DEVICE AND FEATURE SUPPORT

For a complete list of device and feature support, please see the on-line help file in MPLAB IDE for the MPLAB REAL ICE in-circuit emulator.



## Chapter 2. Operation

### 2.1 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 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.

- Tool Comparisons
- Emulator Communications with the PC and Target
- Target Communication Connections
- Trace Connections
- Debugging with the Emulator
- Requirements For Debugging
- Programming with the Emulator
- Resources Used by the Emulator

### 2.2 TOOL COMPARISONS

The MPLAB REAL ICE in-circuit emulator system differs physically and operationally from a classic in-circuit emulator system. It is similar to an in-circuit debugger system.

**TABLE 2-1: TOOL COMPARISONS - HARDWARE CONFIGURATIONS**

| Hardware Tool         | Function | Device-Specific Hardware  |
|-----------------------|----------|---|
| Classic emulator unit | Debug    | Processor module (-ME device)   |
| Programmer unit       | Program  | Socket module   |
| MPLAB REAL ICE unit   | Debug    | Header board (-ICE device) or<br>Regular device with on-board emulation circuitry |
|                       | Program  | Regular device  |
| In-circuit debug unit | Debug    | Header board (-ICD device)<br>Regular device with on-board debug circuitry        |
|                       | Program  | Regular device  |

Classic emulator systems have speed bottlenecks caused by bringing internal busses off-chip and using external memories. The MPLAB REAL ICE in-circuit emulator system eliminates these bottlenecks by using the actual device (if it has on-board emulation circuitry) or a special -ICE version of the device on a header (see the *Header Board Specification* in Recommended Reading for more information) for emulation.

The MPLAB REAL ICE in-circuit emulator programs debug code into a device or -ICE version of a device to perform emulation. Therefore, it may also be used as a programmer to program finished application code into a device. So, unlike classic emulator systems, the additional purchase of a programmer is not necessary.

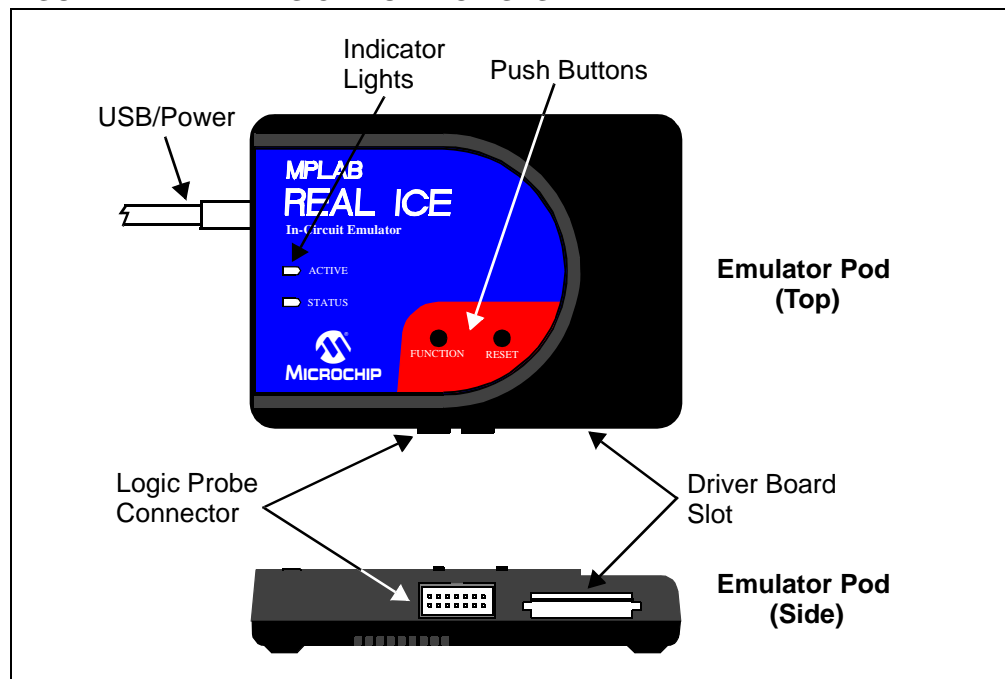
The MPLAB REAL ICE in-circuit emulator system surpasses in-circuit debugger systems in speed and functionality (e.g., trace).

## 2.3 EMULATOR COMMUNICATIONS WITH THE PC AND TARGET

The MPLAB REAL ICE in-circuit emulator system consists of these items:

- Emulator pod with indicator lights, push buttons and a logic probe connector
- USB cable to connect a PC to the emulator pod and power the pod
- Driver board and modular cable(s) to connect the emulator pod to an ICE header or target board

**FIGURE 2-1: BASIC EMULATOR SYSTEM**



The emulator communicates with the PC and is powered through the USB cable.

The emulator communicates with the target through the configurations discussed in the following sections.

### CAUTION

**Do not connect the hardware before installing the software and USB drivers. Also, do not change hardware connections when the pod or target is powered.**

### 2.3.1 Standard Communication

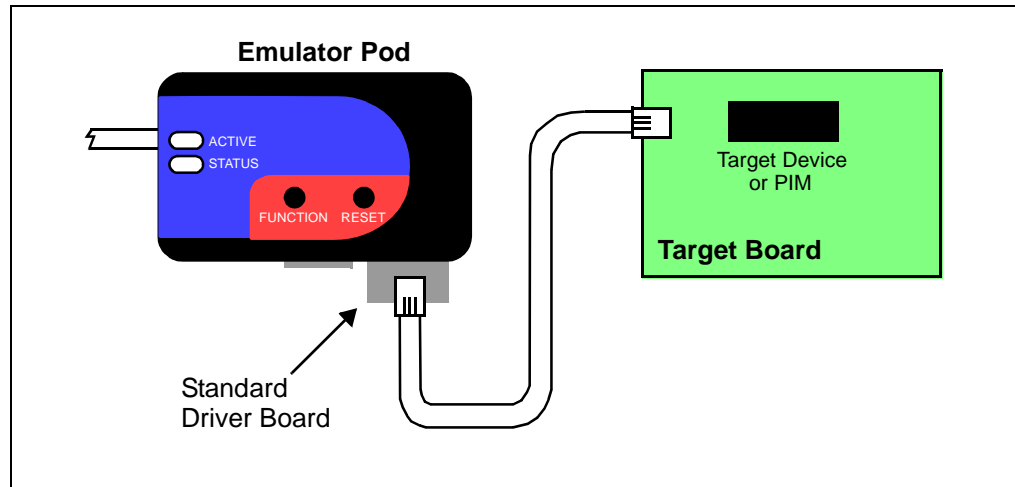
The emulator system can be configured to use standard communication for both programming and debugging functions. This 6-pin connection is the same one used by other Microchip in-circuit debuggers.

The standard driver board is plugged into the emulator pod to configure the system for communication with the target. The modular cable can be either (1) inserted into a matching socket at the target, where the target device is on the target board (Figure 2-2), or (2) inserted into a standard adapter/header board combo (available as a Processor Pak), which is then plugged into the target board (Figure 2-3).

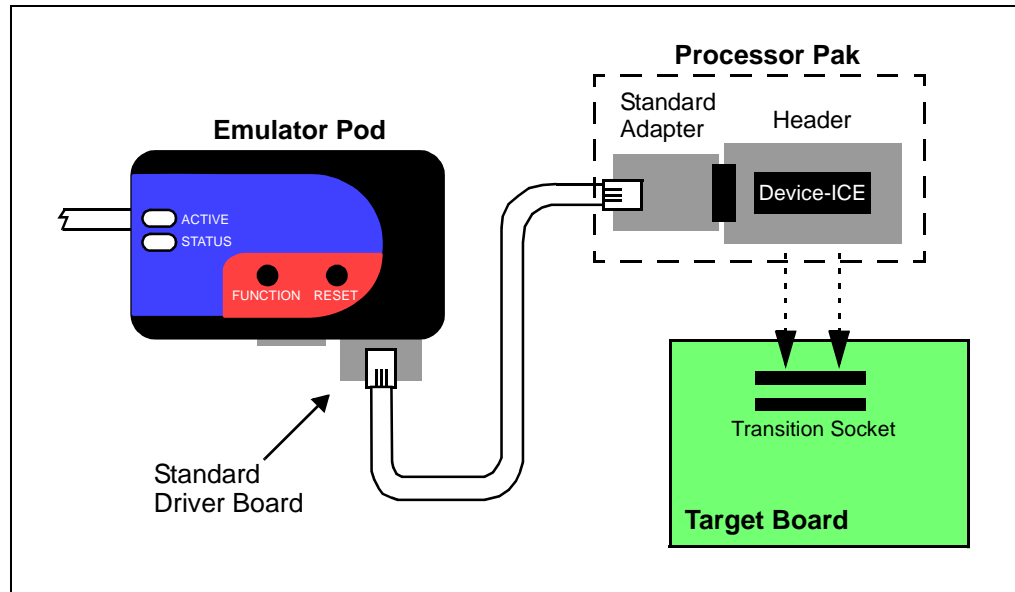
**Note:** Older header boards used a 6-pin (RJ-11) connector instead of an 8-pin connector, so these headers may be connected directly to the emulator.

For more on standard communication, see **Section 13.6 “Standard Communication Hardware”**.

**FIGURE 2-2: STANDARD EMULATOR SYSTEM – DEVICE WITH ON-BOARD ICE CIRCUITRY**



**FIGURE 2-3: STANDARD EMULATOR SYSTEM – ICE DEVICE**



## 2.3.2 High-Speed Communication

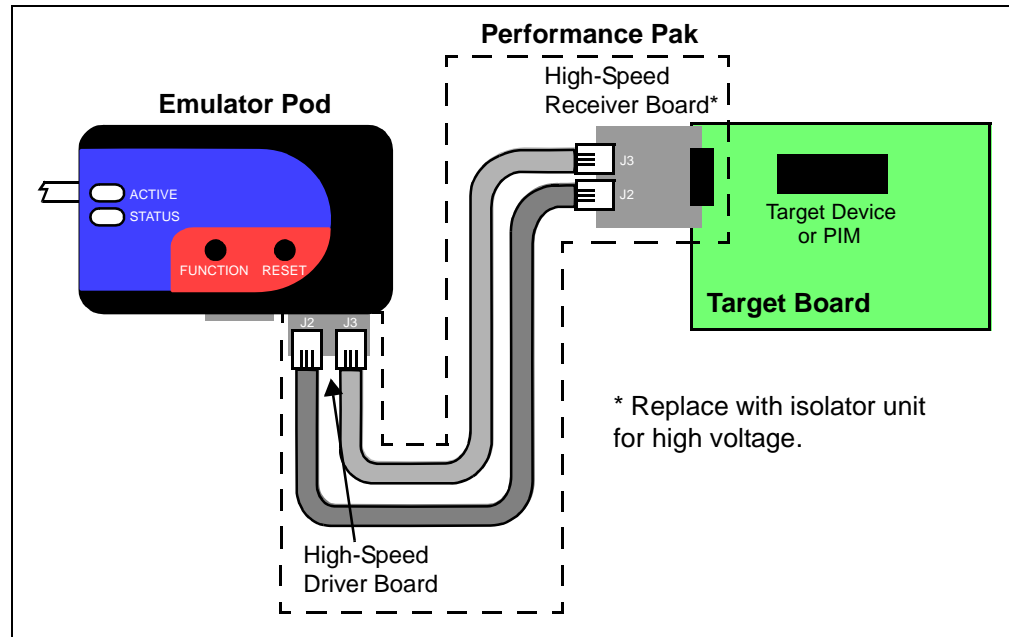
The emulator system can be configured to use high-speed communication for both programming and debugging functions. This connection allows for higher speed operations, a longer distance between the emulator and target, and additional tracing functionality over a standard connection.

The high-speed driver board (from the Performance Pak) is plugged into the emulator pod to configure the system for this type of communication with the target. The modular cables can be inserted into matching sockets at the high-speed receiver board, which is attached via an 8-pin connector into either (1) the target board, with an on-board target device (Figure 2-4), or (2) the header board (from the Processor Pak), which is then plugged into the target board (Figure 2-5).

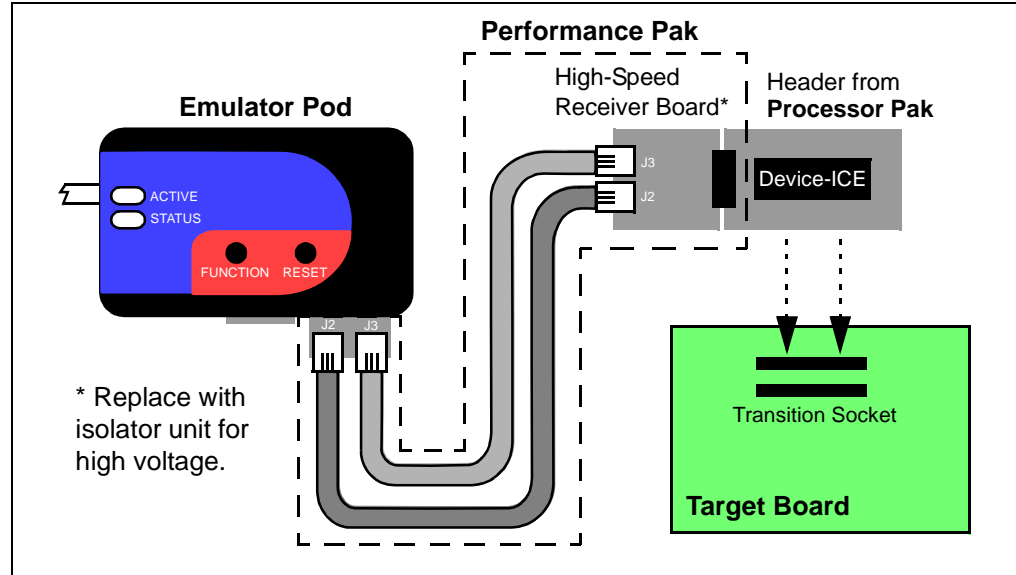
If your application is high-voltage, you will also need to replace the high-speed receiver board with an isolator unit to isolate the target. See **Section 13.8 “MPLAB REAL ICE Isolator unit”**.

For more on high-speed communication, see **Section 13.7 “High-Speed Communication Hardware”**.

**FIGURE 2-4: HIGH-SPEED EMULATOR SYSTEM – DEVICE WITH ON-BOARD ICE CIRCUITRY**



**FIGURE 2-5: HIGH-SPEED EMULATOR SYSTEM – ICE DEVICE**



## 2.4 TARGET COMMUNICATION CONNECTIONS

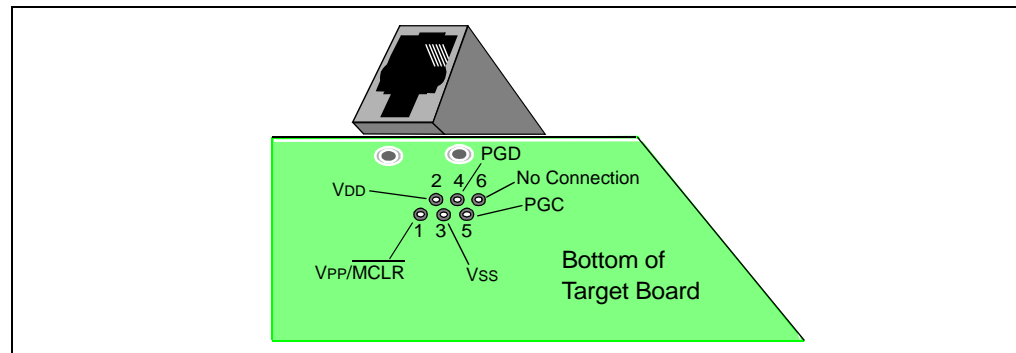
There are two driver boards available to closely match most application requirements. The standard driver board can be used to connect to the myriad of demo boards and applications that contain the RJ11 connector. The high-speed driver/receiver board combination can be used for high-speed applications, for additional trace features, for large (several feet) emulator-to-target distances and for noisy environments.

### 2.4.1 Standard Communication Connection

Using 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 bottom of the target PC board in Figure 2-6.

**Note:** Cable connections at the emulator and 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 13.6.2.1 “Modular Cable Specification”**.

**FIGURE 2-6: STANDARD CONNECTION AT TARGET**

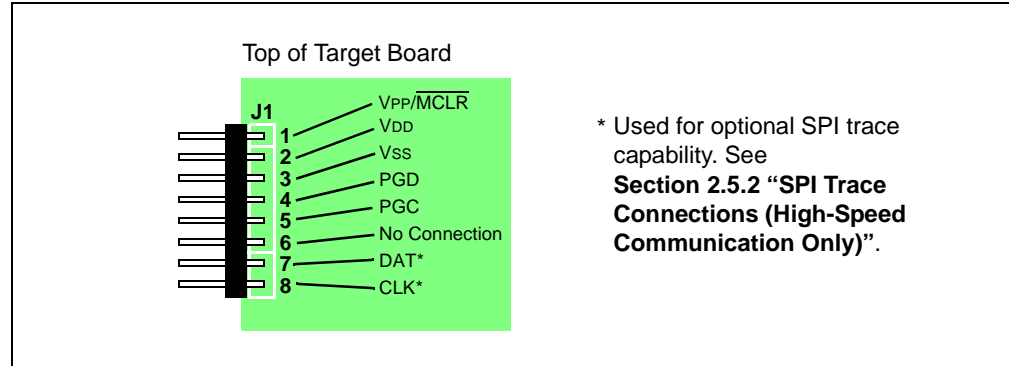


## 2.4.2 High-Speed Communication Connection

Using the high-speed driver/receiver board combination, the MPLAB REAL ICE in-circuit emulator is connected to the target device with an 8-pin interface. The pin numbering for the connector is shown from the top of the target PC board in Figure 2-7.

**Note:** Connections from the emulator to the target are shown in Section 13.7 “High-Speed Communication Hardware”.

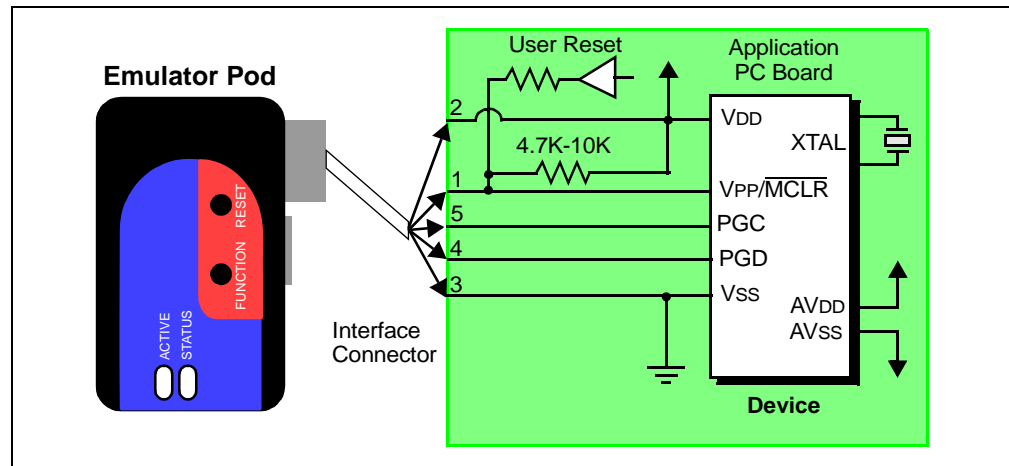
**FIGURE 2-7: HIGH-SPEED CONNECTION AT TARGET**



## 2.4.3 Target Connection Circuitry

Figure 2-8 shows the interconnections of the MPLAB REAL ICE in-circuit emulator to the connector on the target board. The diagram also shows the wiring from the connector to a device on the target PC board. A pull-up resistor (typically 10 kΩ) is recommended to be connected from the VPP/MCLR line to VDD so that the line may be strobed low to reset the device.

**FIGURE 2-8: STANDARD CONNECTION TARGET CIRCUITRY**



In the following descriptions, only three lines are active and relevant to core emulator operation: pins 1 (VPP/MCLR), 5 (PGC) and 4 (PGD). Pins 2 (VDD) and 3 (VSS) are shown on the above diagram for completeness, but are only sensed, not provided or controlled, by the emulator.

Be aware that the target VDD is sensed by the emulator to allow level translation for target low-voltage operation. If the emulator does not sense voltage on its VDD line (pin 2 of the interface connector), it will not operate.

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.

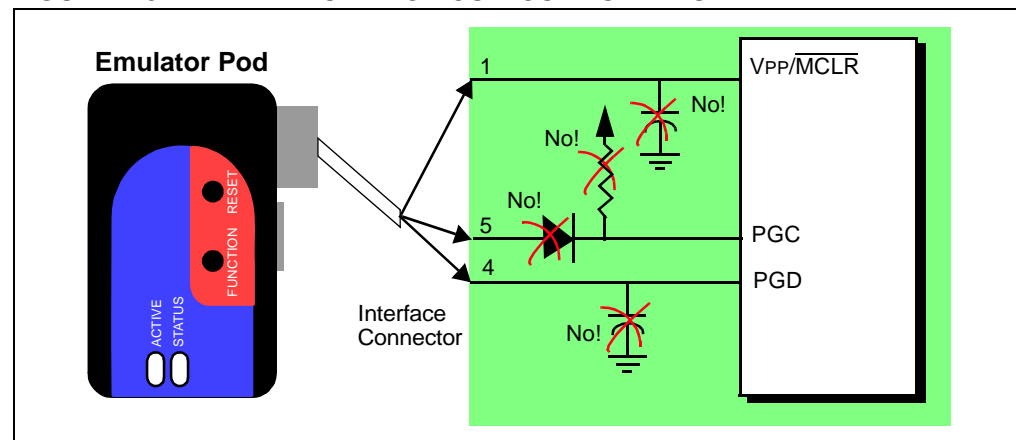
In general, it is recommended per device data sheet that all VDD/AVDD and VSS/AVSS lines be connected to the appropriate levels. Also, devices with a VCAP pin (like PIC18FXXJ devices) should be connected to the appropriate capacitor or other internal regulator device.

**Note:** The interconnection is very simple. Any problems experienced are often caused by other connections or components on these critical lines that interfere with the operation of the MPLAB REAL ICE in-circuit emulator system, as discussed in the next section.

## 2.4.4 Circuits That Will Prevent the Emulator From Functioning

Figure 2-9 shows the active emulator lines with some components that will prevent the MPLAB REAL ICE in-circuit emulator system from functioning.

**FIGURE 2-9: IMPROPER CIRCUIT COMPONENTS**



Specifically, these guidelines must be followed:

- Do not use pull-ups on PGC/PGD – they will divide the voltage levels, since these lines have 4.7 k $\Omega$  pull-down resistors in 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 capacitors on  $\overline{\text{MCLR}}$  – they will prevent fast transitions of VPP. A simple pull-up resistor is generally sufficient.
- Do not use diodes on PGC/PGD – they will prevent bidirectional communication between the emulator and the target device.

For other operational issues, see:

- **Chapter 11. “Error Messages”**
- **Chapter 10. “Frequently Asked Questions (FAQ)”**
- **Section 11.3.6 “Debug Failure Actions”** (Top Reasons Why You Can’t Debug)
- **Section 13.9 “Loop-Back Test Board”**

## 2.5 TRACE CONNECTIONS

When the emulator is selected as the debug tool, it has several trace capabilities, depending on the device selected.

### 2.5.1 Native Trace Connections

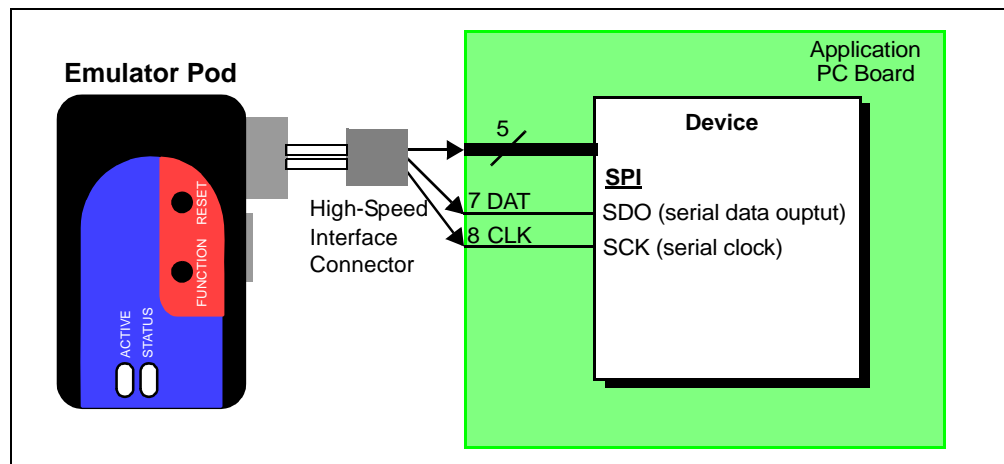
No additional connections are necessary to use Native trace. The communications connection will carry the trace information using the PGD/PGC/EMUC/EMUD pins. However, the selected device must have this feature. If it does not, one of the other trace methods may be used.

For more on this type of trace, see **Section 7.3.3.1 “Native Trace”**.

### 2.5.2 SPI Trace Connections (High-Speed Communication Only)

When using high-speed communications, streaming serial trace is an optional trace available using the device SPI and pins 7 (DAT) and 8 (CLK). Figure 2-10 shows these additional connections. As with pins 4 (PGD) and 5 (PGC) (**Section 2.4.4 “Circuits That Will Prevent the Emulator From Functioning”**), do not use pull-up resistors, capacitors or diodes.

**FIGURE 2-10: SERIAL TRACE CONNECTIONS**



The DAT and CLK lines are intended for use with devices that do not have built-in debug logic that allows tracing to use the PGD/PGC/EMUC/EMUD pins. The DAT line connects to either the target device SPI port SDO1 or SDO2. The CLK line connects to SCK1 or SCK2.

When you dedicate these pins to tracing, then any multiplexed function on these pins can no longer be used by the application.

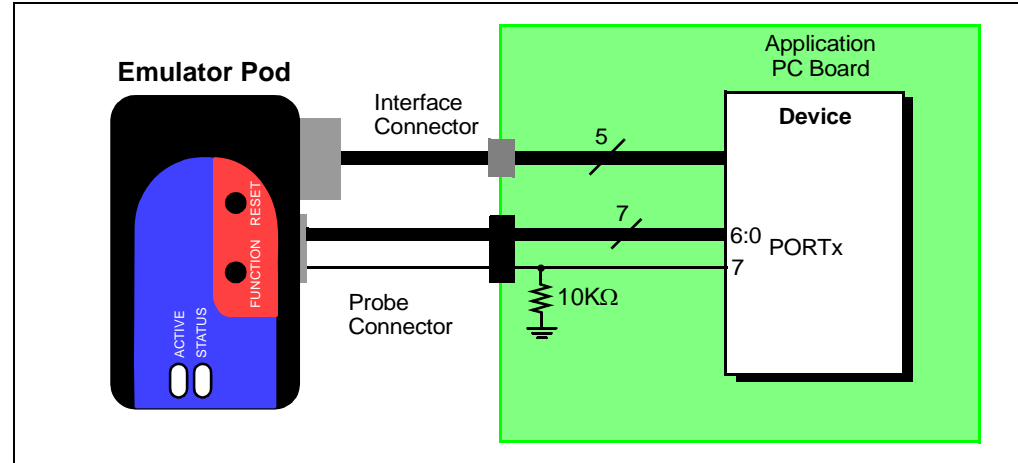
For more on this type of trace, see **Section 7.3.3.3 “SPI Trace”**.



## 2.5.3 I/O Port Trace Connections

Streaming parallel trace is possible using a device 8-pin I/O port and the emulator logic probe connector. This provides greater trace speed and data quantity, but limits emulator-to-target distance by the length of the logic probe connectors. Figure 2-11 shows these additional connections.

**FIGURE 2-11: 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 must not be multiplexed with the currently-used PGC and PGM pins.

A basic configuration is shown in **Table 2-2: “I/O Port Trace Connection Example”**.

**TABLE 2-2: I/O PORT TRACE CONNECTION EXAMPLE**

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

**Note 1:** For pin descriptions, see **Section 13.5.4 “Logic Probe/External Trigger Interface”**.

**2:** Use a 10KΩ pull-down resistor.

As in **Section 2.4.4 “Circuits That Will Prevent the Emulator From Functioning”**, do not use pull-up resistors, capacitors or diodes on port pins, except as specified.

For more on this type of trace, see **Section 7.3.3.2 “I/O Port Trace”**.

## 2.5.4 PIC32 Instruction Trace Connections

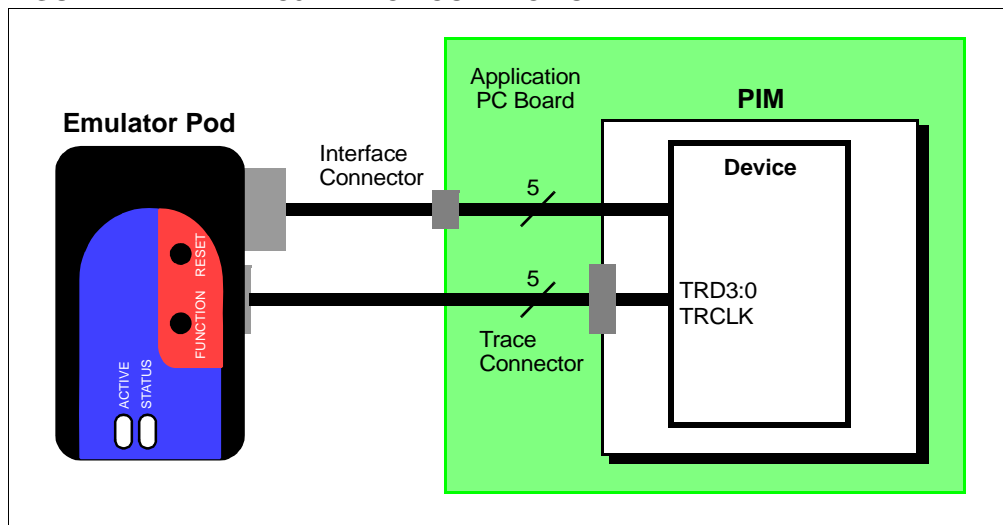
PIC32 Instruction Trace is only available for PIC32MX MCU devices, and it is the only type of trace available for these devices. Also, only some PIC32MX MCU devices have the trace feature. Consult your device data sheet for details.

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

- PIC32MX Plug-In Module (PIM) containing a device that supports trace and a trace port
- PIC32MX Trace Interface Kit containing a 12-inch trace cable and a trace adapter board

To use the PIC32 Instruction Trace feature, see **Section 8.3 “PIC32 Instruction Trace”**.

**FIGURE 2-12: PIC32 TRACE CONNECTION**



## 2.6 DEBUGGING WITH THE EMULATOR

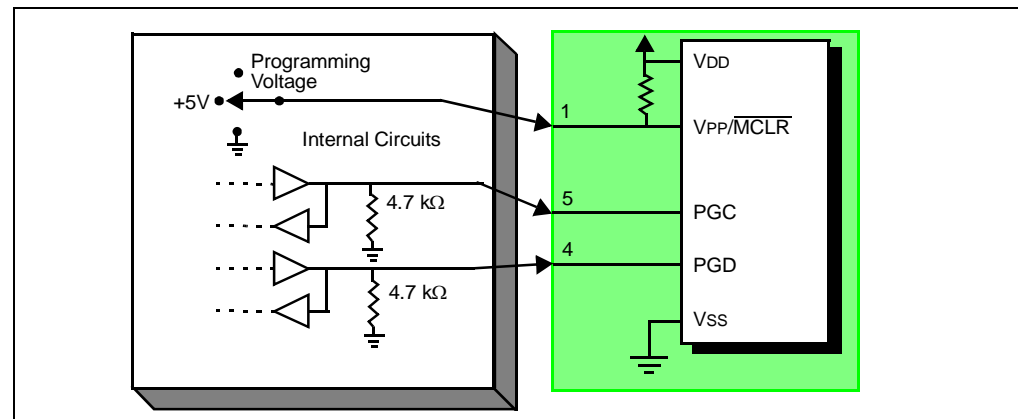
There are two steps to using the MPLAB REAL ICE in-circuit emulator system as a debugger. The first requires that an application be programmed into the target device. The second uses the internal in-circuit debug hardware of the target Flash device to run and test the application program. These two steps are directly related to the MPLAB IDE operations:

1. Programming the code into the target and activating special debug functions (see the next section for details).
2. Using the emulator to set breakpoints and run.

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

Figure 2-13 shows the basic interconnections required for programming. Note that this is the same as Figure 2-8, but for the sake of clarity, the VDD and VSS lines from the emulator are not shown.

**FIGURE 2-13: 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 ICSP protocol of the device under development.

## 2.7 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 3. "Installation"** 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. If the target device does 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 upon the target design.
  - For 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.

## 2.7.1 Sequence of Operations Leading to Debugging

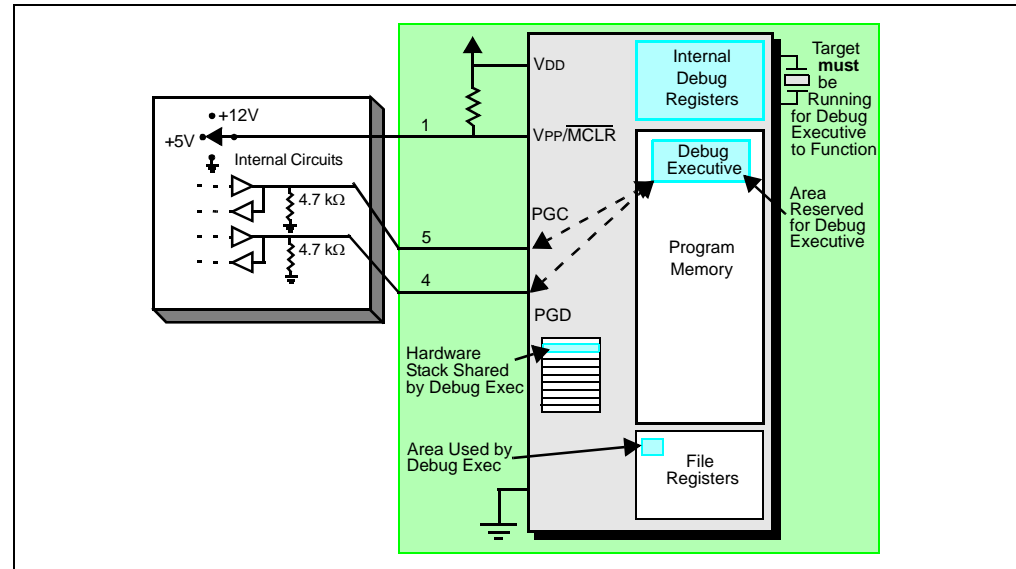
Given that the Requirements For Debugging are met, these actions can be performed when the MPLAB REAL ICE in-circuit emulator is set as the current debugger (*Debugger>Select Tool*):

- The application code is compiled/assembled with the “Build Configuration” list box on the MPLAB IDE toolbar set to “Debug”. Also, it may be set by selecting *Project>Build Configuration>Debug*.
- When *Debugger>Program* is selected, the application code is programmed into the device's memory via the ICSP protocol as described above.
- A small “debug executive” program is loaded into the high area of program memory of the target device. Since 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.
- The target device is held in Reset by keeping the  $V_{PP}/\overline{MCLR}$  line low.

## 2.7.2 Debugging Details

Figure 2-14 illustrates the MPLAB REAL ICE in-circuit emulator system when it is ready for debugging.

**FIGURE 2-14: MPLAB® REAL ICE™ IN-CIRCUIT EMULATOR READY FOR DEBUGGING**



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 *Debugger>Run* function or the Run icon (forward arrow) is usually pressed from MPLAB IDE. The emulator will then tell the debug executive to run. The target will start from the Reset vector and execute until the Program Counter reaches the breakpoint address previously stored 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 the device’s Program Counter 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 get a breakpoint is to press the MPLAB IDE’s **Halt** button (the “pause” symbol to the right of the Run arrow). This toggles the PGC and PGD lines so that the in-circuit debug mechanism of the target device switches the Program Counter 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.

## 2.8 PROGRAMMING WITH THE EMULATOR

Use the MPLAB REAL ICE in-circuit emulator as a programmer to program an actual (non -ICE/-ICD) device, i.e., a device not on a header board. Select "MPLAB REAL ICE" from *Programmer>Select Programmer* and compile/assemble your application code with the "Build Configuration" list box on the MPLAB IDE toolbar set to "Release". Also, it may be set by selecting *Project>Build Configuration>Release*.

All debug features are turned off or removed when the emulator is used as a programmer. When using the *Programmer>Program* selection to program a device, MPLAB IDE will disable the in-circuit debug registers so the MPLAB REAL ICE in-circuit emulator will program only the target application code and the Configuration bits (and EEPROM data, if available and selected) into the target device. The debug executive will not be loaded. As a programmer, the emulator can only toggle the MCLR line to reset and start the target. A breakpoint cannot be set, and register contents cannot be seen or altered.

The MPLAB REAL ICE in-circuit emulator system programs the target using ICSP. Vpp, PGC and PGD lines should be connected as described previously. No clock is required while programming, and all modes of the processor can be programmed, including code protect, Watchdog Timer enabled and table read protect.

The MPLAB REAL ICE in-circuit emulator may be used for production programming, either through MPLAB IDE or as a command-line programmer by using REALICECMD (in the Programmer Utilities folder of the MPLAB IDE installation folder).

## 2.9 RESOURCES USED BY THE EMULATOR

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



# MPLAB® REAL ICE™ IN-CIRCUIT EMULATOR USER'S GUIDE

---

---

## Part 2 – Getting Started

---

---

|                              |    |
|------------------------------|----|
| Chapter 3. Installation..... | 33 |
| Chapter 4. Tutorial.....     | 37 |

# MPLAB® REAL ICE™ In-Circuit Emulator User's Guide

---

NOTES:





# MPLAB® REAL ICE™ IN-CIRCUIT EMULATOR USER'S GUIDE

---

---

## Chapter 3. Installation

---

---

### 3.1 INTRODUCTION

How to install the MPLAB REAL ICE in-circuit emulator system is discussed.

- Installing the Software
- Installing the USB Device Drivers
- Selecting Target Communications
- Setting Up the Target
- Connecting the Logic Probes
- Setting Up MPLAB IDE

### 3.2 INSTALLING THE SOFTWARE

To install the MPLAB IDE software, first acquire the latest MPLAB IDE installation executable (MPxxxxx.exe, where xxxxxx represents the version of MPLAB IDE) from either the Microchip web site (www.microchip.com) or the MPLAB IDE CD-ROM (DS51123). Then run the executable and follow the screens to install MPLAB IDE.

### 3.3 INSTALLING THE USB DEVICE DRIVERS

Installing MPLAB IDE will preinstall the USB device drivers for the MPLAB REAL ICE in-circuit emulator. Therefore, once you have installed MPLAB IDE, connect the emulator to the PC with a USB cable and follow the Windows® "New Hardware Wizard" to automatically install the drivers.

Expanded USB device driver installation instructions may found at:

*MPLAB IDE installation directory\REAL ICE\Drivers\ddri.htm*

**Note:** If you change USB ports/hubs, you do not need to reinstall the drivers since the emulator is serialized.

## 3.4 SELECTING TARGET COMMUNICATIONS

A driver board is inserted into the pod to select the type of communication with the target, either standard (for header boards and many demo boards) or high speed (for target boards over six inches away from the emulator). See **Section 2.3 “Emulator Communications with the PC and Target”** for more details.

### CAUTION

Neither the emulator nor target should be powered when inserting or removing a driver board or damage to the driver board could result.

If you DID NOT have a driver board installed in the emulator when you installed the drivers, unplug the USB/Power cable now. Then proceed with the installation instructions below.

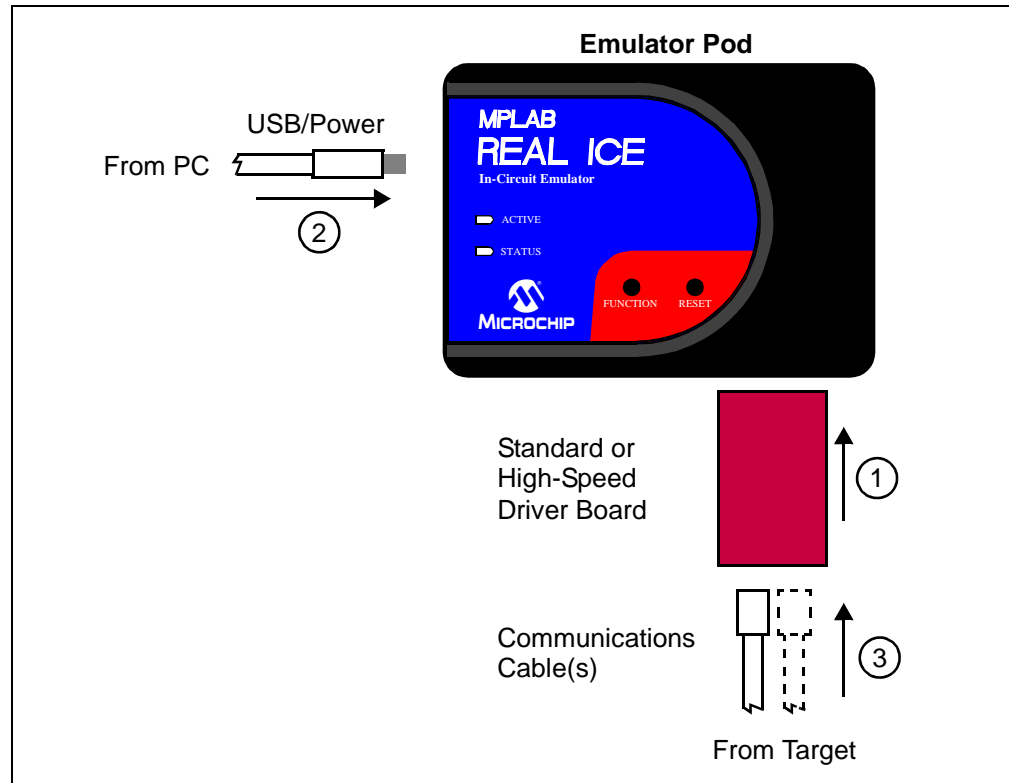
If you DID have a driver board installed in the emulator when you installed the drivers, proceed to step three below.

To install a driver board:

1. Insert the desired driver board into the emulator pod
2. Plug in the USB/power cable
3. Attach the communication cable(s)

To change a driver board, remove target power and unplug the USB, remove the board, insert the other board, and then plug in the USB and power the target.

**FIGURE 3-1: INSERT DRIVER BOARD AND USB/POWER CABLE**



## 3.5 SETTING UP THE TARGET

Once the type of communication has been determined by inserting the corresponding driver board into the emulator, the target must be set up to accommodate this, as well as the type of target device to be used, i.e., regular or ICE.

Some devices have built-in debug circuitry. These “regular” devices may be used directly by the emulator on the target.

Other devices have no built-in debug circuitry. For these devices, a special ICE device (*Device-ICE*) is required, mounted on a header board. For more on header boards, see the “*Header Board Specification*” (in Recommended Reading).

**Note:** Some regular devices have ICE devices available to provide dedicated debug pins and (sometimes) memory.

### 3.5.1 Using Regular Devices

For regular devices, the emulator may be connected directly to the target board. The device on the target board must have built-in debug circuitry in order for the MPLAB REAL ICE in-circuit emulator to perform emulation with it. Consult the device data sheet to see if the device has the needed debug circuitry, i.e., it should have a “Background Debugger Enable” Configuration bit.

**Note:** In the future, devices with circuitry that support ICD may be used, though only standard debug, and not emulator debug, functions will be available.

The target board must have a connector to accommodate to the communications chosen for the emulator. For connection information, see **Section 2.3.1 “Standard Communication”** or **Section 2.3.2 “High-Speed Communication”**.

### 3.5.2 Using ICE Devices and Header Boards

For ICE devices, an ICE header board is required. The header board contains the hardware necessary to emulate a specific device or family of devices.

**Note:** In the future, ICD header boards with ICD devices (*Device-ICD*) may be used, though only standard debug, and not emulator debug, functions will be available.

A transition socket is used with the ICE header to connect the header to the target board. Transition sockets are available in various styles to allow a common header to be connected to one of the supported surface mount package styles. For more information on transition sockets, see the “*Transition Socket Specification*” (DS51194).

Header board layout will be different for standard or high-speed communications. For connection information, see **Section 2.3.1 “Standard Communication”** or **Section 2.3.2 “High-Speed Communication”**.

### 3.5.3 Powering the Target

If you have not already done so, connect the emulator pod to the target using the appropriate cables for the driver board selected (see **Section 3.4 “Selecting Target Communications”**). Then power the target.

**Note:** The emulator cannot power the target.

## 3.6 CONNECTING THE LOGIC PROBES

The logic probes may be connected into the logic probe connector on the emulator pod. These probes will allow halting the MPLAB REAL ICE in-circuit emulator by external triggers, and will provide output triggers to synchronize external equipment such as oscilloscopes and logic analyzers. See **Section 6.3 “External Triggers”** for setup information.

This connector can also be used for trace. See:

- **Section 2.5.3 “I/O Port Trace Connections”**
- **Section 2.5.4 “PIC32 Instruction Trace Connections”**

## 3.7 SETTING UP MPLAB IDE

Once the hardware is connected and powered, MPLAB IDE may be set up for use with the MPLAB REAL ICE in-circuit emulator.

On some devices, you must select the communications channel in the Configuration bits, e.g., PGC1/EMUC1 and PGD1/EMUD1. Make sure the pins selected here are the same ones physically connected to the device.

For more on setting up MPLAB IDE, see **Chapter 5. “General Setup”**.



---

---

## Chapter 4. Tutorial

---

---

### 4.1 INTRODUCTION

This tutorial walks you through the process of developing a simple project using the sample programs `counter.c` and `timer.c`. This is an implementation of the PIC24FJ128GA010 device using the Explorer 16 Demo Board (DM240001). The program `counter.c` is a simple counting program. The incremental count, delayed by using Timer 1 (`timer.c`), is displayed via Port A on the demo board's LEDs.

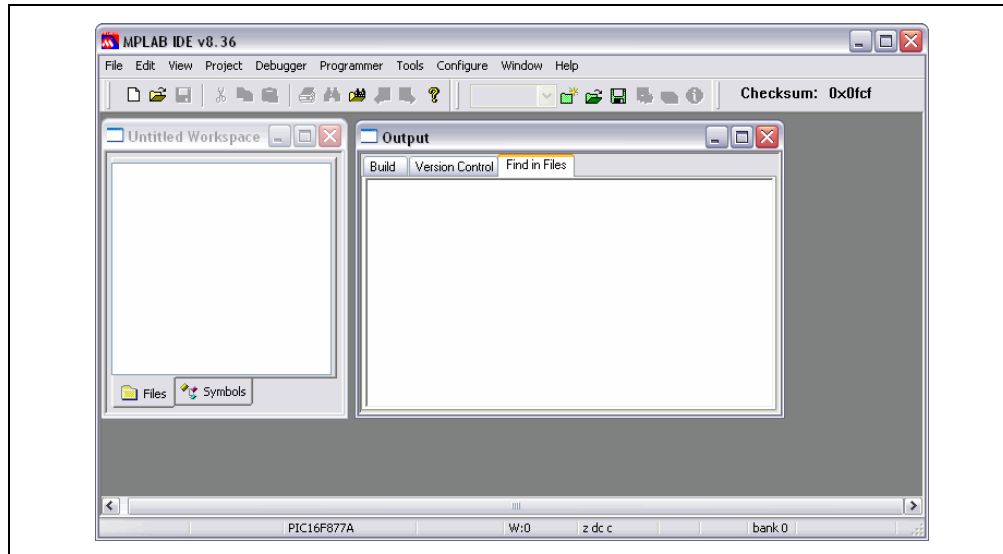
Topics covered in this chapter:

- Setting Up The Environment
- Creating the Application Code
- Running the Project Wizard
- Viewing the Project
- Creating a Hex File
- Viewing Debug Options
- Setting Up the Demo Board
- Loading Program Code For Debugging
- Running Debug Code
- Debugging Code Using Breakpoints
- Debugging Code Using A RunTime Watch
- Debugging Code Using Native Trace
- Programming the Application
- Other Trace Methods – SPI or I/O Port Trace
- Other Trace Methods – PIC32 Instruction Trace

### 4.2 SETTING UP THE ENVIRONMENT

Before beginning this tutorial, follow the steps in **Chapter 3. "Installation"** to set up the MPLAB IDE software and MPLAB REAL ICE system hardware. Double-click on the MPLAB IDE icon to launch the application. Once launched, the MPLAB IDE desktop should appear.

FIGURE 4-1: MPLAB® IDE DESKTOP



## 4.2.1 Selecting the Device

To select the device for this tutorial:

1. Select *Configure>Select Device*.
2. In the Device Selection dialog, choose “PIC24FJ128GA010” from the Device list box. The light icon next to “MPLAB REAL ICE” in the “Microchip Tool Programmer/Debugger Tool Support” sections should be green.
3. Click **OK**.

## 4.2.2 Selecting the Emulator as a Debugger

To select MPLAB REAL ICE in-circuit emulator as a debugger, select *Debugger>Select Tool>REAL ICE*. Then:

1. The Output window will open to display connection information. Depending on the version of MPLAB IDE or the device selected, a message box may appear indicating that the firmware needs to be updated. MPLAB IDE will automatically install the new firmware. Also, since different MPLAB REAL ICE in-circuit emulator firmware is used for different families of devices, this message box may appear when switching to a different device. For more information, see **12.3.13 “Output Window, REAL ICE Tab”**.
2. The Debugger menu will show available emulator debug options.
3. A Debug toolbar will appear. Mouseover a button to see a pop-up of its function.

## 4.3 CREATING THE APPLICATION CODE

For this tutorial, two C programs will be used. The code for each is shown below.

1. Using Windows® Explorer, create a project folder.
2. Open an editor window by selecting *File>New*. Enter the code for the first program (*counter.c*) in this window and save to the project folder.
3. Open another editor window by selecting *File>New*. Enter the code for the second program (*timer.c*) in this window and save to the project folder.

## counter.c

```
/*
 * MPLAB REAL ICE In-Circuit Emulator Tutorial
 * Counting program
 *
 *
 * Demo Board:      Explorer 16
 * Processor:       PIC24FJ128GA010
 * Compiler:        MPLAB C30
 * Linker:          MPLAB LINK30
 * Company:         Microchip Technology Incorporated
 *
 */

#include "p24FJ128GA010.h"

// Set up configuration bits
_CONFIG1( JTAGEN_OFF & GCP_OFF & GWRP_OFF & COE_OFF & FWDTEN_OFF & ICS_PGx2)
_CONFIG2( FCKSM_CSDCMD & OSCIOFNC_ON & POSCMOD_HS & FNOSC_PRI )

// Set up user-defined variables
#define INIT_COUNT 0
unsigned int counter;

int main(void)
{
    // Set up PortA I/Os as digital output
    AD1PCFG = 0xffff;
    TRISA = 0x0000;

    // Set up Timer1
    TimerInit();

    // Initialize variables
    counter = INIT_COUNT;

    while (1) {
        // Wait for Timer1 overflow
        if (TimerIsOverflowEvent()){
            counter++; //increment counter
            PORTA = counter; //display on port LEDs
        } // End of if...

    } // End of while loop...
} // End of main()...
```

## timer.c

```
/*
 * MPLAB REAL ICE In-Circuit Emulator Tutorial
 * Timer program
 *
 *
 * Demo Board:      Explorer 16
 * Processor:       PIC24FJ128GA010
 * Compiler:        MPLAB C30
 * Linker:          MPLAB LINK30
 * Company:         Microchip Technology Incorporated
 *
 */
```

# MPLAB® REAL ICE™ In-Circuit Emulator User's Guide

---

```
*****/
#include "p24FJ128GA010.h"

//declare functions
extern void TimerInit(void);
extern unsigned char TimerIsOverflowEvent(void);

/*****
 * Function:      TimerInit
 *
 * PreCondition:  None.
 *
 * Input:         None.
 *
 * Output:        None.
 *
 * Overview:      Initializes Timer1 for use.
 *
 *****/
void TimerInit(void)
{
    PR1 = 0xFFFF;

    IPC0bits.T1IP = 5;
    T1CON = 0b1000000000010000;
    IFS0bits.T1IF = 0;
}

/*****
 * Function:      TimerIsOverflowEvent
 *
 * PreCondition:  None.
 *
 * Input:         None.
 *
 * Output:        Status.
 *
 * Overview:      Checks for an overflow event, returns TRUE if
 *                an overflow occurred.
 *
 * Note:          This function should be checked at least twice
 *                per overflow period.
 *****/
unsigned char TimerIsOverflowEvent(void)
{
    if (IFS0bits.T1IF)
    {
        IFS0bits.T1IF = 0;
        TMR1 = 0;
        return(1);
    }
    return(0);
}

/*****
 * EOF
 *****/
```

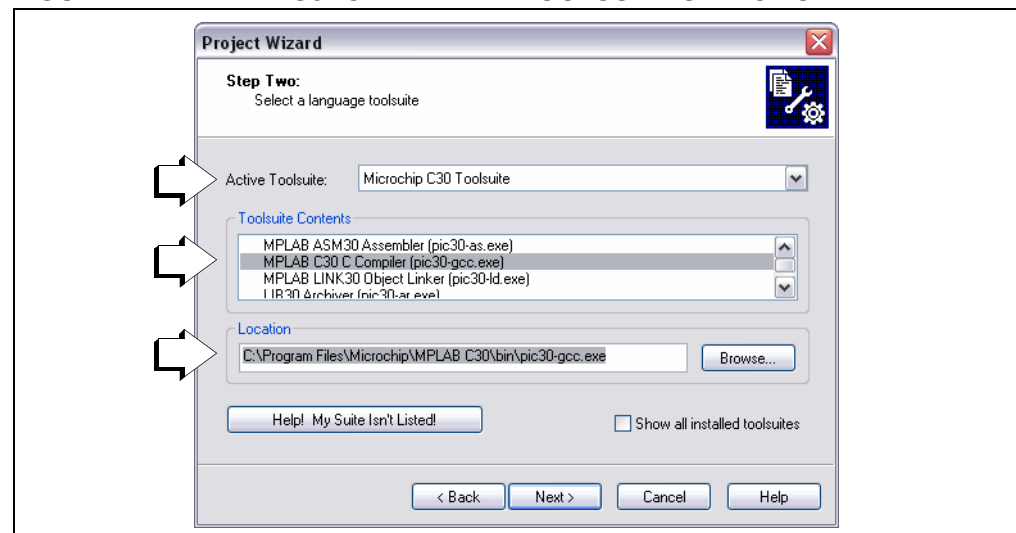


## 4.4 RUNNING THE PROJECT WIZARD

The MPLAB C compiler for PIC24 MCUs and dsPIC DSCs (formerly MPLAB C30) will be used in this project. You may either purchase the full compiler or download a free evaluation version from the Microchip website.

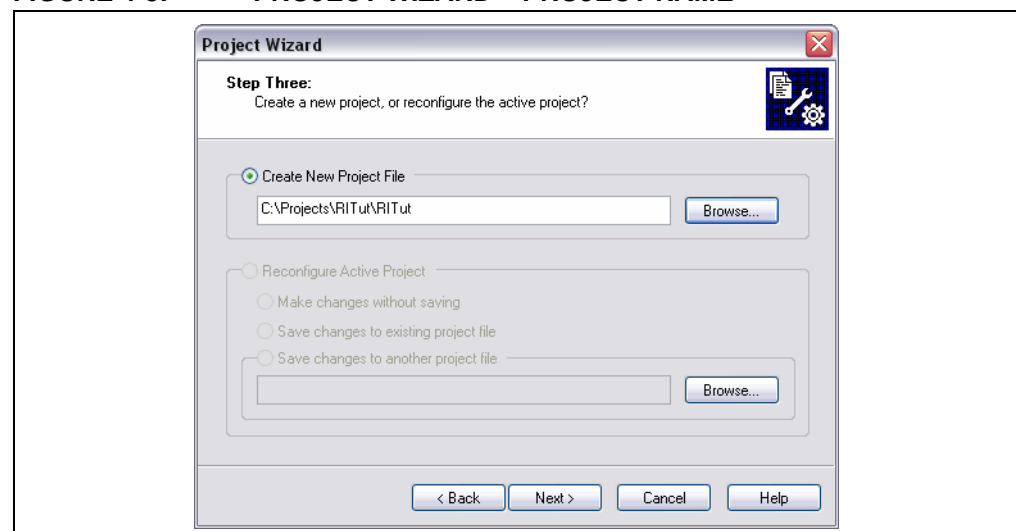
1. To set up this project, select *Project>Project Wizard*. A Welcome screen will appear.
2. Proceed to the second dialog of the wizard. The PIC24FJ128GA010 should be selected.
3. Proceed to the next dialog of the wizard to set up the language tools. In the “Active Toolsuite” pull-down, select “Microchip C30 Toolsuite.” Make sure that the tools are set to the proper executables, by default located in the directory C:\Program Files\Microchip\MPLAB C30\bin. MPLAB C30 should be pointing to pic30-gcc.exe and MPLAB LINK30 should be pointing to pic30-ld.exe.

**FIGURE 4-2: PROJECT WIZARD – TOOLSUITE SELECTION**



4. Proceed to the next dialog of the wizard to give a name and location to your project. You may **Browse** to find a location.

**FIGURE 4-3: PROJECT WIZARD – PROJECT NAME**

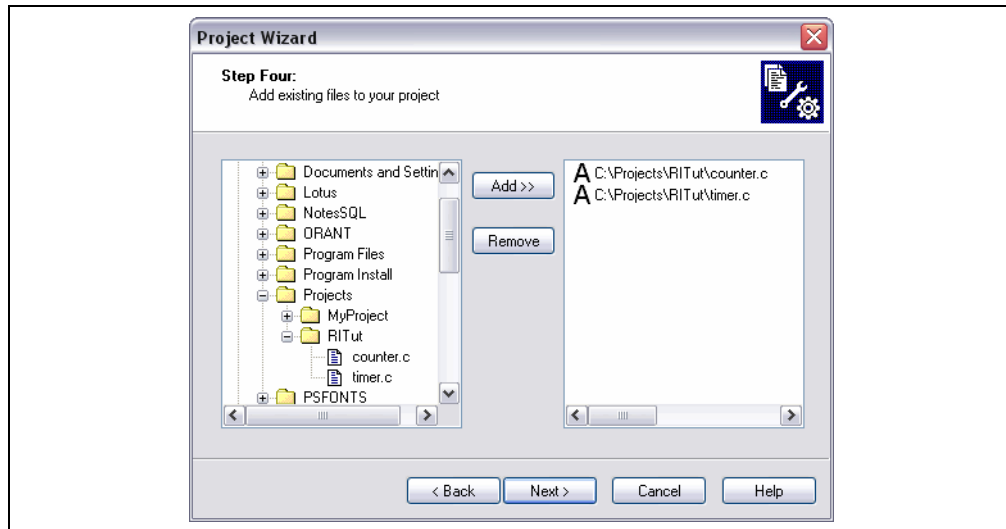


5. Proceed to the next dialog of the wizard where project files can be added. Files can also be added later if something is missed.

For this example, browse to your project directory to find both files. Click on `counter.c` to highlight it and then click on **ADD>>** to add it to the right pane. Click on `timer.c` to highlight it and then click on **ADD>>** to add it to the right pane.

Leave the "A" next to the file name. For more information on what this and other letters mean, click the **Help** button on the dialog.

**FIGURE 4-4: PROJECT WIZARD – ADD FILES**

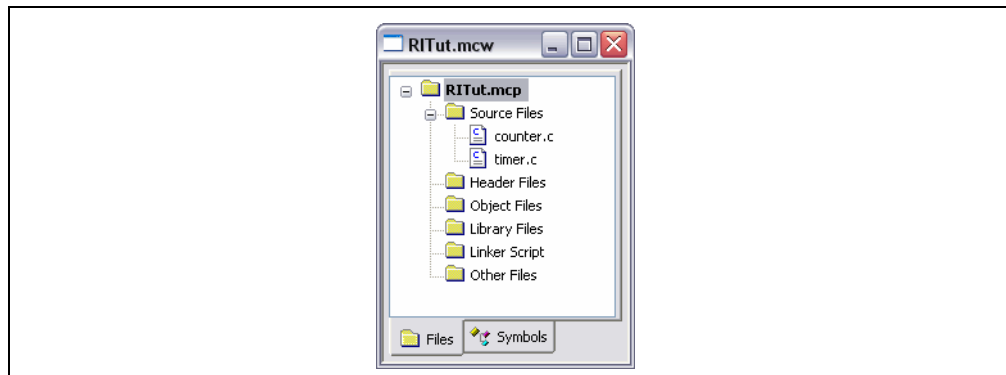


6. Proceed to the Summary screen. If you have made any errors, click **<Back** to return to a previous wizard dialog. If everything is correct, click **Finish**.

## 4.5 VIEWING THE PROJECT

After exiting the wizard, the MPLAB IDE desktop will again be visible. Close all other windows on the desktop to see the Project window.

**FIGURE 4-5: PROJECT WINDOW**



Additional files can be added to the project using the project window. Right click on any line in the project window tree to pop up a menu with additional options for adding and removing files.

**Note:** Although the header file `p24FJ128GA010.h` and a linker script file are used in the project, you do not need to add them to the project; MPLAB IDE will find them for you.

## 4.6 CREATING A HEX FILE

To create a hex file for debugging:

- On the Project toolbar, select “Debug” from the Build Configuration drop-down list.
- Select *Project>Build All* or right click on the project name in the project window and select “Build All” from the popup menu.

The project will build (Figure 4-6), and the resulting `.hex` file will have the same name as the project (Figure 4-7). The hex file is the code that will be programmed into the target device.

**Note:** Depending on the build options selected, your Output window may look different from Figure 4-6 (*Project>Build Options>Project*, **MPLAB C30** and **MPLAB LINK30** tabs.)

**FIGURE 4-6: OUTPUT WINDOW**

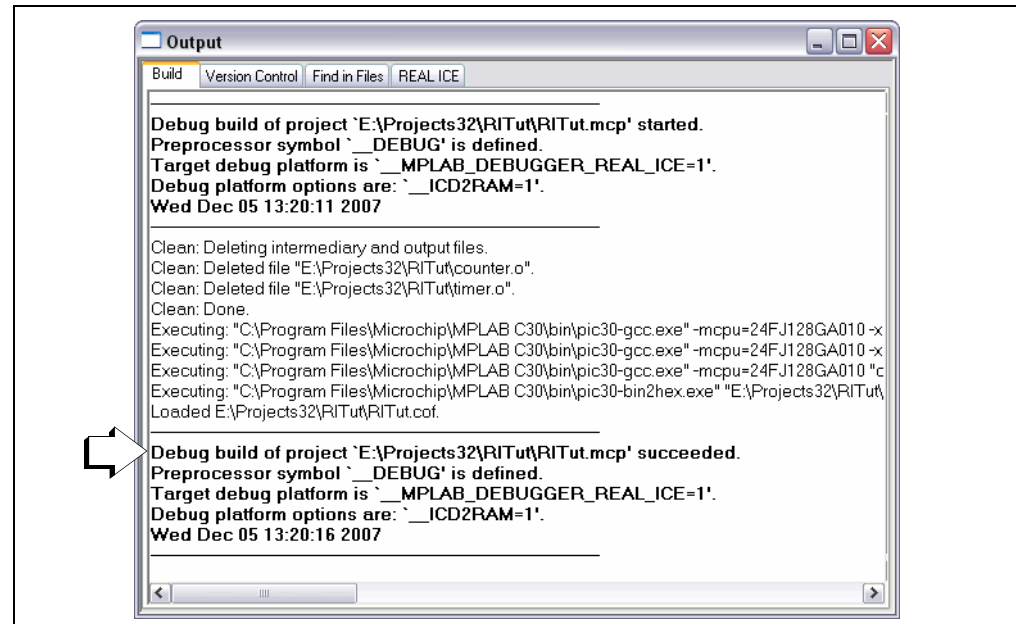
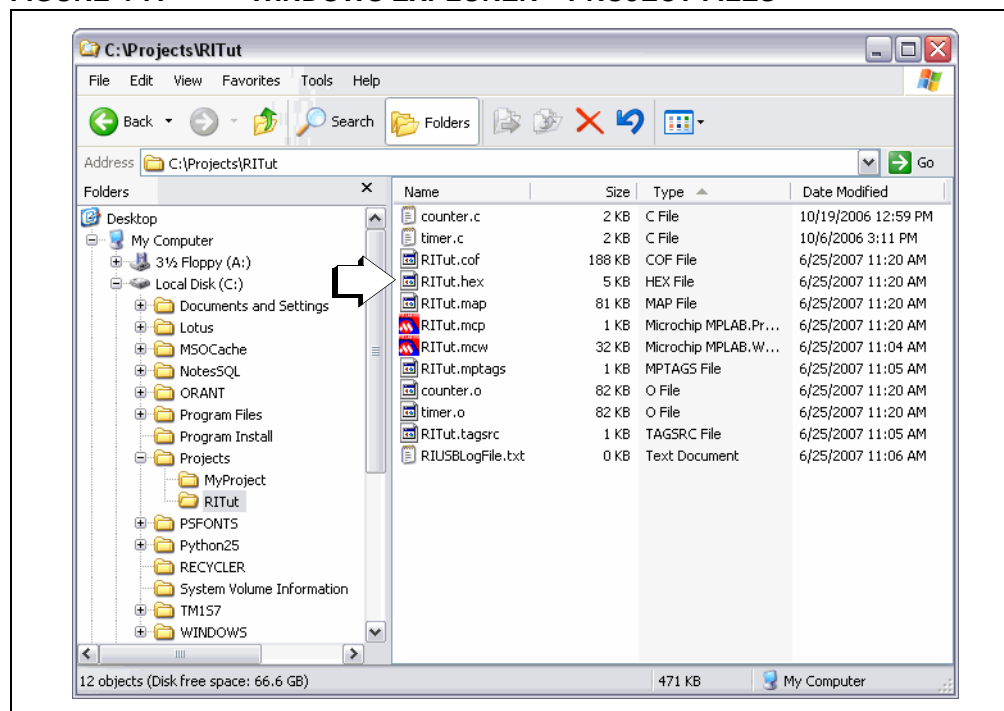


FIGURE 4-7: WINDOWS EXPLORER – PROJECT FILES



## 4.7 VIEWING DEBUG OPTIONS

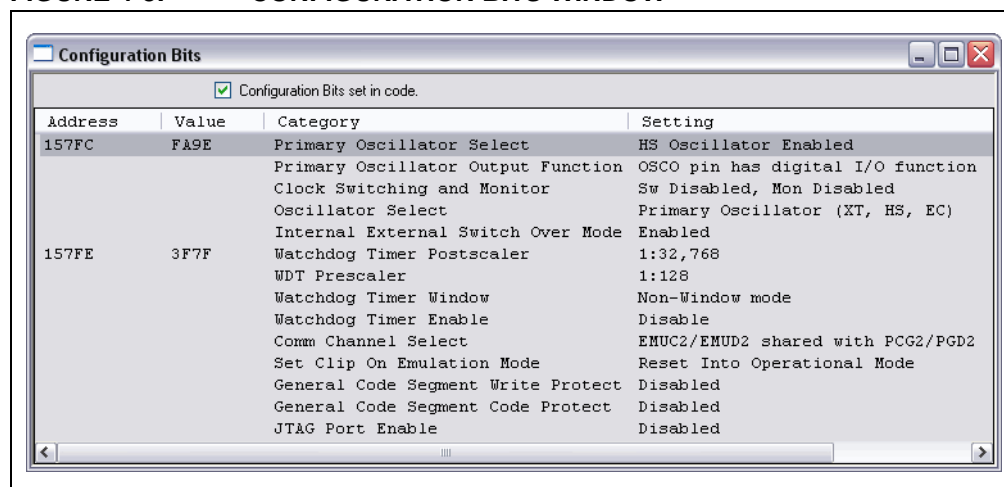
Before you begin debugging your code, review the default settings of several items. In your own projects, you may need to set these items differently.

### 4.7.1 Configuration Bits

In this tutorial, the relevant device Configuration bits are set in the `counter.c` code using the `_CONFIG1` and `_CONFIG2` directives. For information on the function of these PIC24FJ128GA010 configuration register bits, see the *PIC24FJ128GA Family Data Sheet* (DS39747).

Configuration bits also may be set by selecting *Configure>Configuration Bits* and unchecking "Configuration bits set in code". Do not change any values for this tutorial.

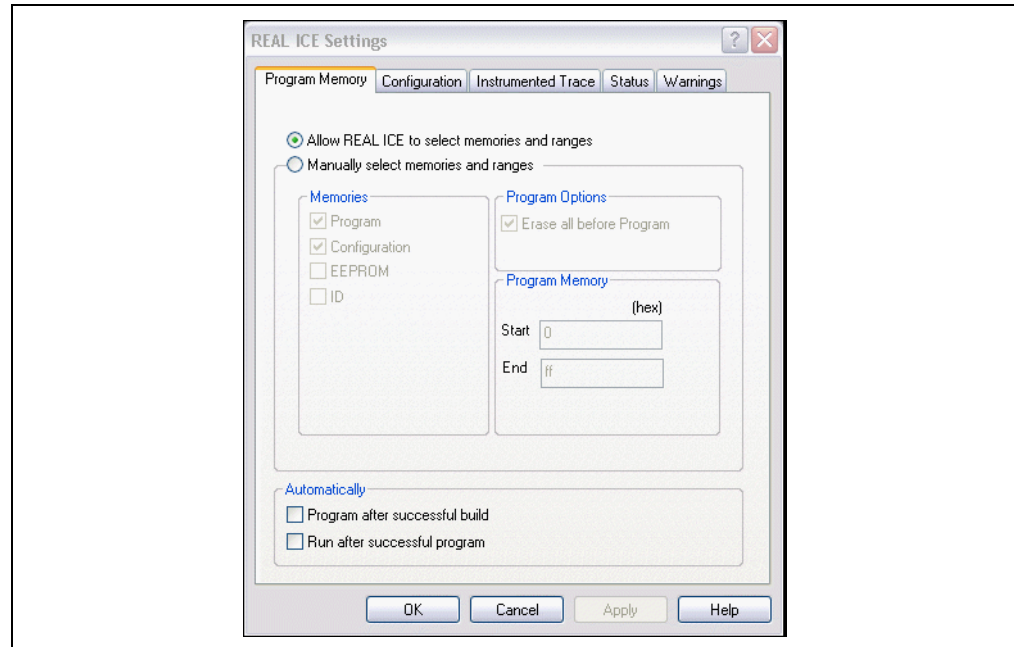
FIGURE 4-8: CONFIGURATION BITS WINDOW



### 4.7.2 Programming Options

To set program options, select *Debugger>Settings* and click on the **Program Memory** tab.

**FIGURE 4-9: EMULATOR PROGRAM MEMORY TAB**



Here you may allow the emulator to automatically choose the programming ranges (recommended) or you may select ranges manually.

- The “Memories” section should have “Program” checked, and “EEPROM” and “ID” unchecked. When using the MPLAB REAL ICE in-circuit emulator as a debugger, Configuration bits will always be programmed and the “Configuration” box will be checked and grayed out.
- For the PIC24FJ devices, all memory will be erased each time the chip is programmed. Therefore, in the “Program Options” section, “Erase all before Program” will have no effect.
- The “Program Memory” addresses (“Start” and “End” address) set the range of program memory that will be read, programmed or verified.

When debugging code, you will frequently repeat the edit, rebuild, reprogram and run sequence. To automate this, there are checkboxes “Program after successful build” and “Run after successful program”. Leave these unchecked for now.

## 4.8 SETTING UP THE DEMO BOARD

Before beginning to debug, make sure the Explorer 16 Demo Board is set up properly. For more information, see the “*Explorer 16 Development Board User's Guide*” (DS51589).

### 4.8.1 Demo Board Settings

Settings for this tutorial should be as follows:

- PIC24FJ128GA010 PIM (Plug-In Module) plugged into the board.
- S2: “PIM” selected; “PIC” selection for devices soldered onto the board.
- J7: “PIC24” selected; the emulator will communicate directly with the PIC24FJ128GA010 and not the on-board PIC18LF4550 USB device.
- JP2: LEDs have been enabled by connecting Jumper 2.
- D1 on: Power being supplied to board.

### 4.8.2 Clock Speed

For data capture and trace, the emulator needs to know the instruction cycle speed. Based on the previous demo board set up, the target oscillator will be 8MHz. This will make instruction cycle speed = 8MHz / 2 = 4MIPS.

Select *Debugger>Settings*, click on the **Clock** tab and enter the clock information.

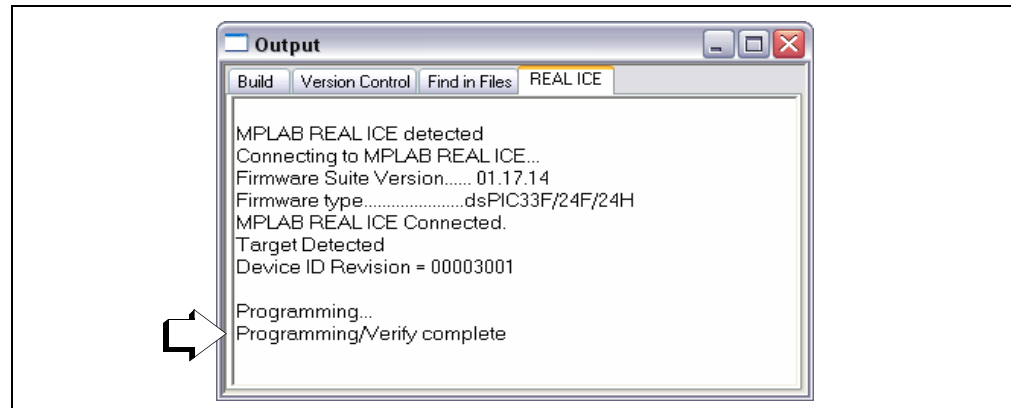
## 4.9 LOADING PROGRAM CODE FOR DEBUGGING

Select *Debugger>Program* to program RITut .hex into the PIC24FJ128GA010 on the Explorer 16 demo board.

**Note:** The debug executive code is automatically programmed in upper program memory for MPLAB REAL ICE debug functions. Debug code must be programmed into the target device to use the in-circuit debugging capabilities of the MPLAB REAL ICE in-circuit emulator.

During programming, the **REAL ICE** tab of the Output dialog shows the current phase of operation. When programming is complete, the dialog should look similar to Figure 4-10.

**FIGURE 4-10: OUTPUT WINDOW – MPLAB® REAL ICE™ TAB**










**Note:** If you have trouble programming your device or communicating with the emulator, unplug the Explorer 16 board and use the Loop-Back Test board (**Section 13.9 “Loop-Back Test Board”**) to verify communications. For additional help, see **Chapter 10. “Frequently Asked Questions (FAQ)”**.

## 4.10 RUNNING DEBUG CODE

The MPLAB REAL ICE in-circuit emulator executes in Real Time or in Step mode.

- Real Time execution occurs when the device is put in the MPLAB IDE's Run mode.
- Step mode execution can be accessed after the processor is halted.

These toolbar buttons can be used for quick access to commonly-used debug operations.

| Debugger Menu   | Run   | Halt  | Animate   | Step Into   | Step Over   | Step Out  | Reset   |
|-----------------|---|---|---|---|---|---|---|
| Toolbar Buttons |  |  |  |  |  |  |  |

Begin in Real Time mode:

1. Open the source files `counter.c` and `timer.c` (double-click on the file names in the Project window or use *File>Open*).
2. Select *Debugger>Run* (or click the **Run** toolbar button).
3. Observe the LEDs. They will be counting up in binary.
4. Select *Debugger>Halt* (or click the **Halt** toolbar button) to stop the program execution.
5. When the emulator halts, one of the open source code windows will pop to the front and a green arrow will indicate where the program halted.

To use Step mode:

1. Select *Debugger>Step Into* (or click the **Step Into** toolbar button) to execute one instruction and then halt. The green arrow in the code listing will move accordingly.
2. Repeat as needed.

The step functions "Step Over" and "Step Out" are used with functions and discussed in the MPLAB IDE documentation.

## 4.11 DEBUGGING CODE USING BREAKPOINTS

The example code in this tutorial has already been debugged and works as expected. However, this code is still useful to demonstrate the debugging features of the MPLAB REAL ICE in-circuit emulator. The first debug feature to be discussed are breakpoints. Breakpoints stop code execution at a selected line of code.

The number of hardware and software breakpoints available and/or used is displayed in the Device Debug Resource toolbar. See the MPLAB IDE documentation for more on this feature.

- Setting Software Breakpoints

## 4.11.1 Choosing a Breakpoint Type

For the device used in this tutorial, you have the choice of using either hardware or software breakpoints.

To set breakpoint options, select *Debugger>Settings* and click on the **Configuration** tab. Select the type of breakpoint that best suits your application needs. For this tutorial, we will begin using the default breakpoint type (hardware breakpoints.)

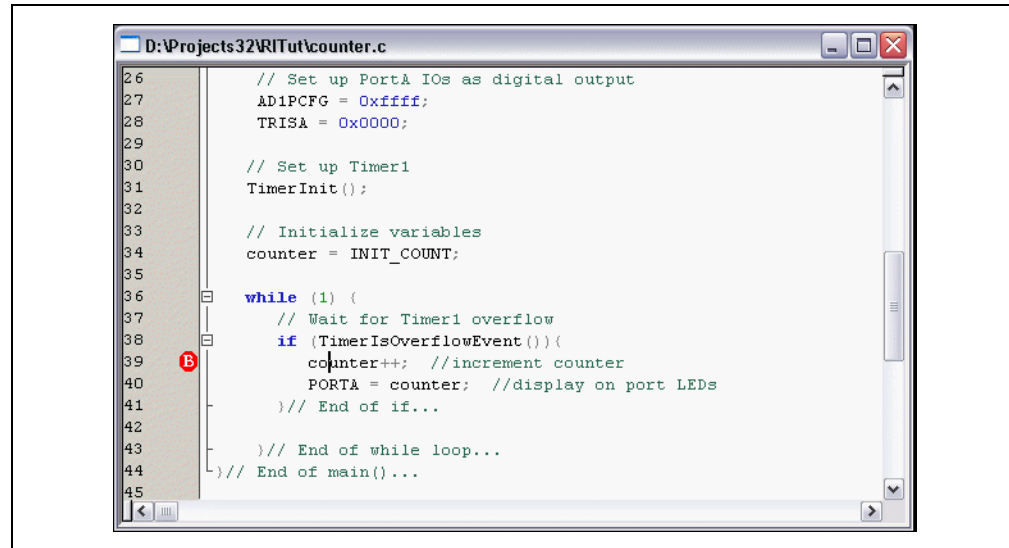
## 4.11.2 Setting a Single Hardware Breakpoint

To set a single breakpoint:

1. Select *Debugger>Reset>Processor Reset* (or click the **Reset** toolbar button) to reset the example program.
2. Highlight or place the cursor on the following line of code from `counter.c`:  

```
counter++; //increment counter
```
3. Double-click on the line, or right click on the line and then select *Set Breakpoint* from the shortcut menu. This line is now marked as a breakpoint (B in red stop sign) as shown in Figure 4-11.

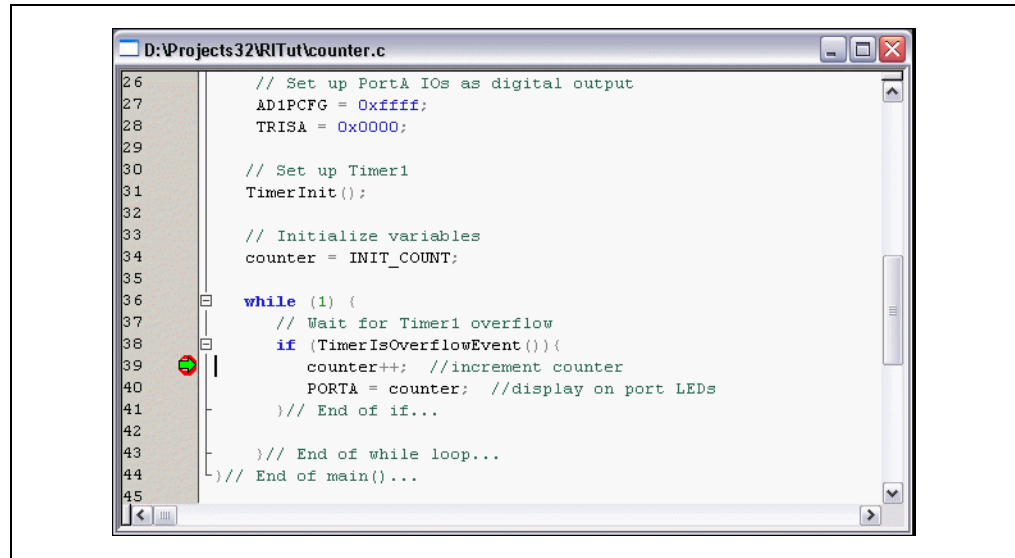
**FIGURE 4-11: SET BREAKPOINT**



4. Select *Debugger>Run* (or click the **Run** toolbar button) to run the program once again in Real-Time mode. The program will halt at the line marked by the breakpoint, but now there will be a green arrow over the breakpoint symbol.

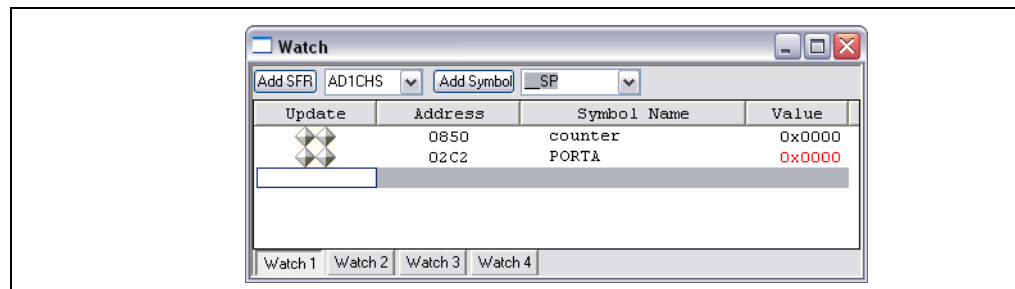


**FIGURE 4-12: PROGRAM HALTED**



- Open a new Watch window to watch the `counter` variable change value as the program executes. Select *View>Watch*. The Watch dialog opens with the **Watch\_1** tab selected. Select “counter” from the list next to **Add Symbol**, and then click the button. `counter` is added to the Watch window. Select “PORTA” from the list next to **Add SFR**, and then click the button. `PORTA` is added to the Watch window. The selected symbols should now be visible in the Watch window as shown in Figure 4-13.

**FIGURE 4-13: WATCH WINDOW**



- Select *Debugger>Run* (or click the **Run** toolbar button) to run the program once again. The program will halt at the breakpoint and you will notice that the value of both variables has incremented by 1.
- Run again as desired to see the values increase. When done, use *Debugger>Reset>Processor Reset* (or click the **Reset** toolbar button) to reset the processor.

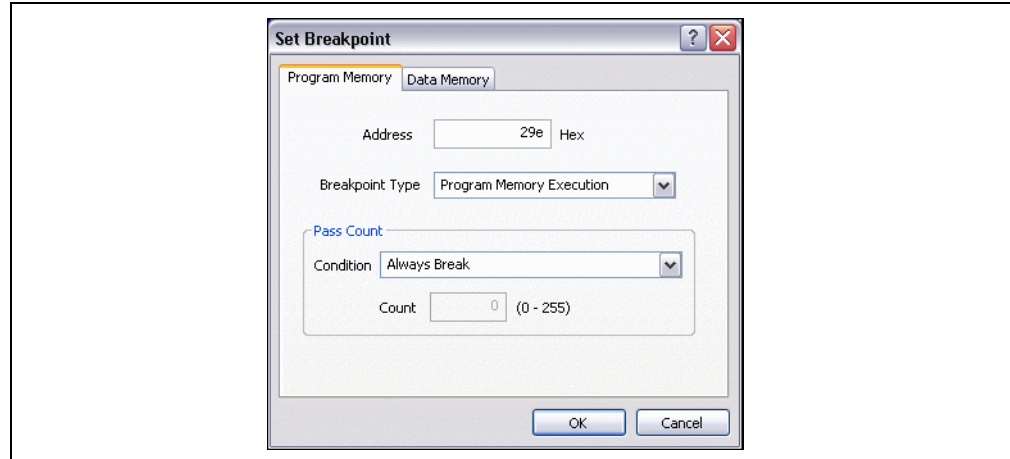
### 4.11.3 Setting Multiple Hardware Breakpoints

To set multiple breakpoints, either set numerous single breakpoints as specified in the previous section or use the Breakpoints dialog (see **Section 12.3.1 “Breakpoints Dialog”**). The Breakpoints dialog also allows you to control breakpoint interaction.

**Note:** If you exceed the maximum allowed number of breakpoints for your device, MPLAB IDE will warn you.

1. Select *Debugger>Breakpoints* to open the Breakpoints dialog. The breakpoint set in the previous section will be displayed in this dialog. Click the **Add Breakpoint** button to add another breakpoint.
2. On the **Program Memory** tab of the Set Breakpoint dialog, enter "29E" as the hex Address and click **OK**.

**FIGURE 4-14: SET BREAKPOINTS DIALOG**

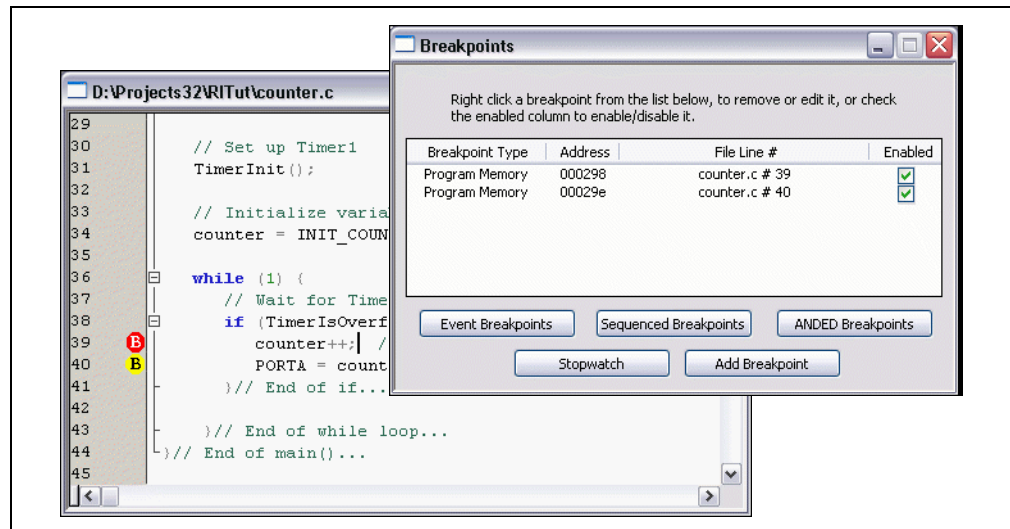


The additional breakpoint will appear below the previous breakpoint in the Breakpoints dialog and also as a breakpoint symbol next to the following line of code:

```
PORTA = counter; //display on port LEDs
```

The breakpoint symbol is yellow in this case because it was set based on an address.

**FIGURE 4-15: TWO BREAKPOINTS**

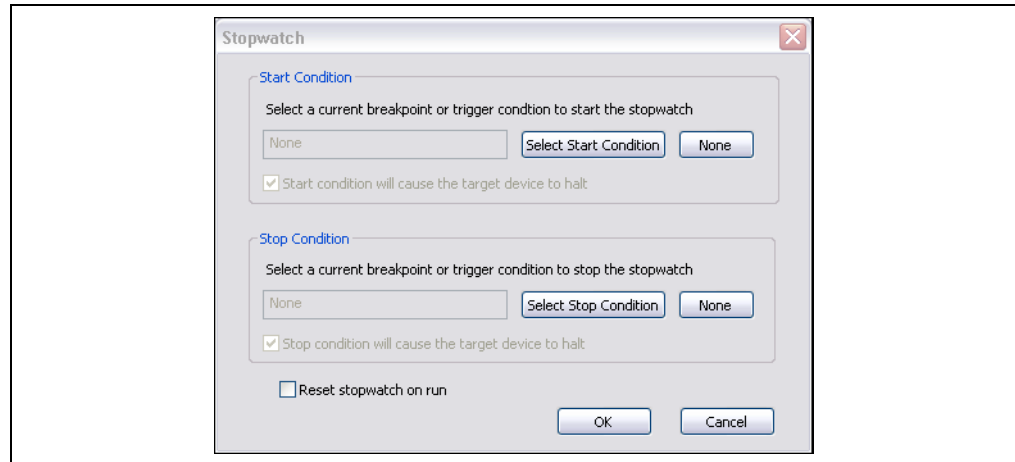


3. Run the program to see it halt at the first breakpoint. The values in the Watch window will not change. Then run again to see it stop at the second breakpoint. (The program may skip past this breakpoint.) Now the values in the Watch window will change.

## 4.11.4 Using the Stopwatch with Breakpoints

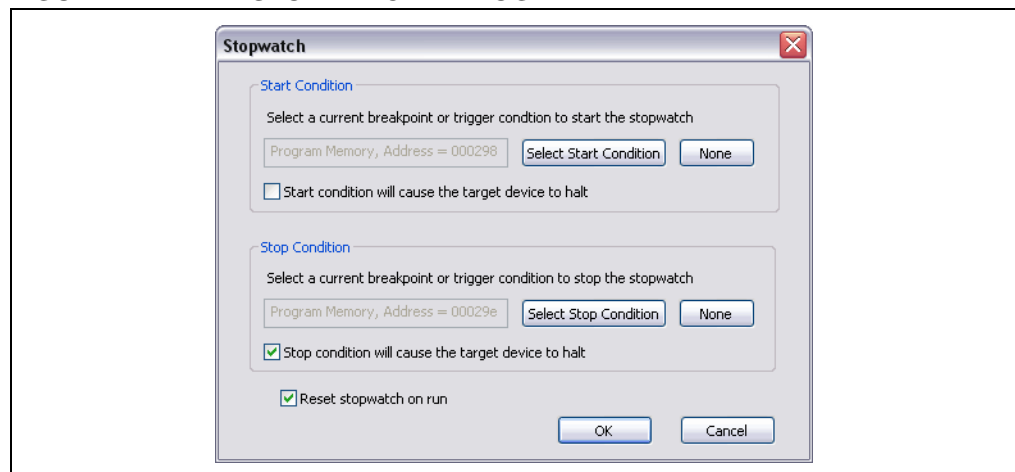
To determine the time between the breakpoints, use the Stopwatch.

**FIGURE 4-16: STOPWATCH DIALOG**



1. Click **Stopwatch** (on the Breakpoints dialog) to open the Stopwatch dialog.

**FIGURE 4-17: STOPWATCH DIALOG**



2. Under “Start Condition”, click **Select Start Condition** and choose the first breakpoint. Then uncheck “Start condition will cause the target device to halt”.
3. Under “Stop Condition”, click **Select Stop Condition** and choose the second breakpoint. Then check “Stop condition will cause the target device to halt”.
4. Check “Reset stopwatch on run”.
5. Click **OK**.
6. Run the program until it halts. In the Output window, on the **REAL ICE** tab, the number of cycles between the two instructions should be shown as:  
Stopwatch cycle count = 3
7. Clear both breakpoints from the code by deleting them from the Breakpoints dialog, double-clicking on each line to remove them, or right clicking on each line and selecting “Remove Breakpoint”. You can also right click and select *Breakpoints>Remove All Breakpoints* to remove both at once.

## 4.11.5 Setting Software Breakpoints

To change the breakpoint type from hardware to software:

- Select *Debugger>Settings* and click on the **Configuration** tab.
- Click the radio button next to “Use Software Breakpoints”.
- Click **OK**.

You will now use software breakpoints instead of the hardware breakpoints used previously.

**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.

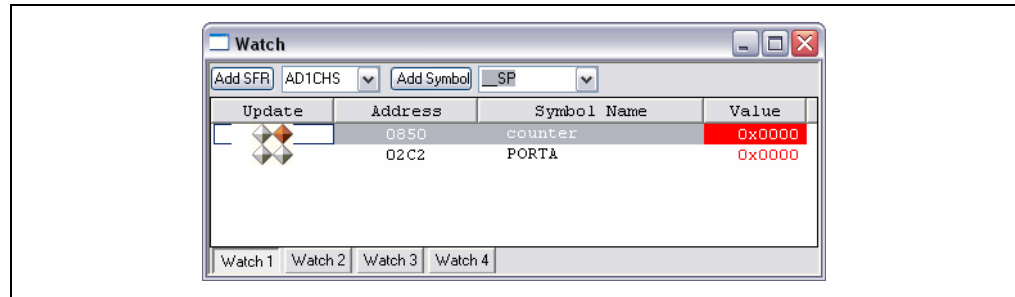
1. To set a single software breakpoint, follow the instructions in **Section 4.11.2 “Setting a Single Hardware Breakpoint”**.
  - When you set a software breakpoint, you will see the following in the Output window:  
Programming software breakpoint(s)...  
Software breakpoint(s) set.
  - If you have already set a hardware breakpoint in this tutorial, the variables will already be added to the Watch window for use with the software breakpoint.
2. To set multiple software breakpoints, follow the instructions in **Section 4.11.3 “Setting Multiple Hardware Breakpoints”**.
  - There is no breakpoint skidding with software breakpoints, i.e., the program halts on the breakpoint. This may affect how you see values change in the Watch window.
  - There is no maximum number of breakpoints with software breakpoints, i.e, although this tutorial only uses two, the number of software breakpoints is unlimited.
3. The stopwatch is meant to be used with hardware breakpoints. However, you can use the stopwatch with software breakpoints, but they will be converted to hardware breakpoints as you select them. In the Output window, you will see:  
Converting breakpoint types...  
Breakpoint type conversion complete.  
Follow the steps as specified in **Section 4.11.4 “Using the Stopwatch with Breakpoints”**.
4. Set the breakpoints to hardware again for the remainder of the tutorial. Select *Debugger>Settings*, click on the **Configuration** tab, click the radio button next to “Use Hardware Breakpoints” and then click **OK**.

## 4.12 DEBUGGING CODE USING A RUNTIME WATCH

Set up a runtime watch to view changes to the `counter` symbol as the program runs. For more information on runtime watches, see either **Section 7.2 “Data Capture and Runtime Watches”** or **Section 8.2 “Data Capture and Runtime Watches”** (PIC32MX devices only).

1. Remove all breakpoints from code. To do this, right click on any line of code and select *Breakpoints>Remove All Breakpoints*.
2. In the Watch window, click on the `counter` Symbol Name to select that line. Then click the second diamond in the first column of that line to enable a runtime watch. (See **Section 12.3.9 “Watch Window - Data Capture/Runtime Watch”** for more information.)

**FIGURE 4-18: WATCH WINDOW SET FOR RUNTIME WATCH**



3. Rebuild the project (*Project>Build All*) and reprogram the target device (*Debugger>Program*).
4. Make sure the Watch window is visible. Then Run the program and watch the `counter` values change real-time in the Watch window.
5. Halt the program.
6. Remove the runtime watch by clicking again on the second (brown) diamond. The diamond then should no longer be colored.
7. Close the Watch window.

## 4.13 DEBUGGING CODE USING NATIVE TRACE

The trace function can be used to collect information on variables and code and store it in a buffer while the code is executing.

In this section, Native trace will be used. For more information about tracing in general using the MPLAB REAL ICE in-circuit emulator, see **Chapter 7. “Debug for 8- and 16-Bit Devices”**.

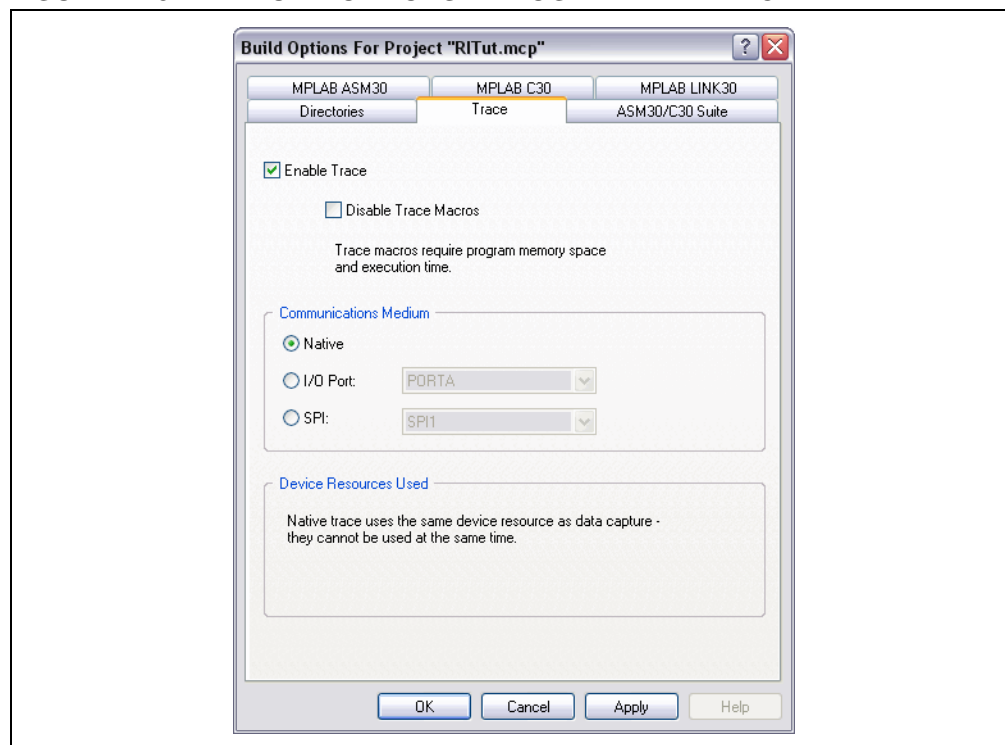
- Note 1:** Trace operation requires 16-bit C compiler v2.04 and above.
- 2:** Real-time data capture triggers (from the previous section) cannot be used at the same time as Native trace.

### 4.13.1 Logging Variable Values

To log a variable value:

1. Select *Project>Build Options>Project, Trace* tab. Check “Enable Trace” and uncheck “Disable Trace Macros”. Then select the type of trace, i.e., “Native Trace” for devices with built-in ICE support. Click **OK**.

**FIGURE 4-19: BUILD OPTIONS DIALOG – NATIVE TRACE**



- Highlight the variable `counter` from the following line of code:

```
counter++; //increment counter
```

Right click on the highlighted variable and select “Log Selected C Value” from the pop-up menu. This causes the following macro line to be inserted above the line containing the variable:

```
__LOG(id,counter);
```

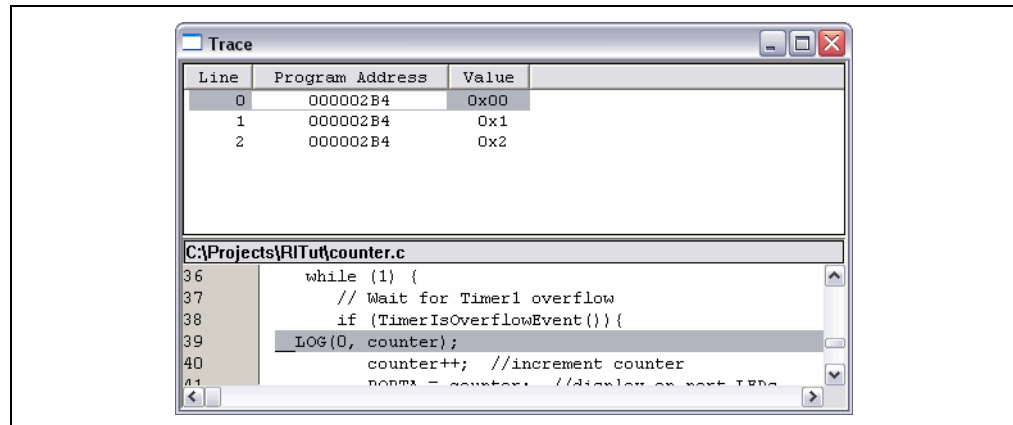
where `id` is a log number auto-generated during build.

- Rebuild the project (*Project>Build All*).
- After rebuilding, a Warning dialog will ask, “File has been modified. Do you want to reload?”. Click **Yes**. When you examine your code, you will find that the log `id` has been replaced with a unique number.

**Note:** To disable this warning and automatically reload, select *Configure>Settings, Other* tab, and check “Automatically reload files that were modified outside of the IDE”. Then click **OK**.

- Reprogram the device (*Debugger>Program*).
- Double-click on the following line to place a breakpoint there:  
`PORTA = counter; //display on port LEDs`
- Reset and run the program until it halts at the breakpoint. Repeat this three times.
- Select *View>Trace* to view the trace data (**Section 12.3.11 “Trace Window”**) or right click and in the Trace window and select “Reload”. You should see variable values logged in this window. To see the related code in the lower portion of the window, you may need to click on a logged value in the upper portion of the window.

**FIGURE 4-20: VIEW TRACE WINDOW – LOG VARIABLE**



To trace multiple variables, you must place a macro before each variable that you wish to trace.

## 4.13.2 Tracing Lines of Code

To trace a line of code:

1. Remove the log macro line from code, i.e., highlight it and hit <Delete>.
2. Highlight or click on the following line of code:

```
counter++; //increment counter
```

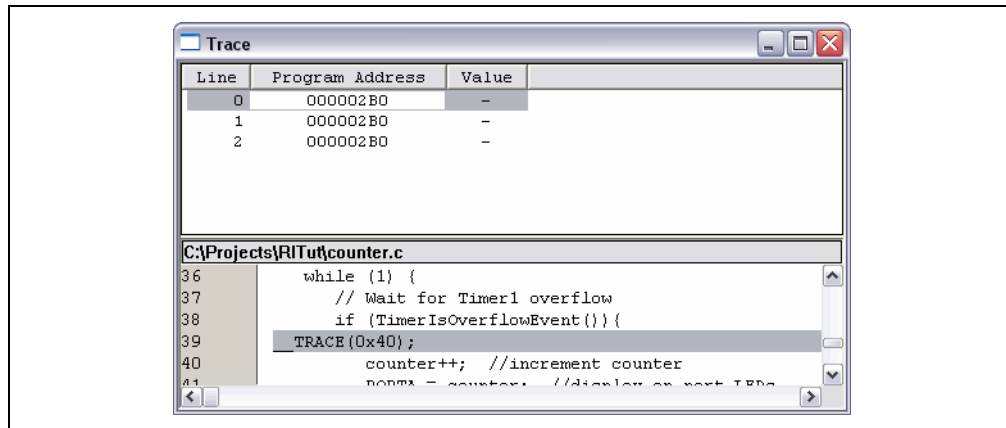
Right click to select "Insert C Line Trace" from the pop-up menu. This causes the following macro line to be inserted above the selected line:

```
__TRACE(id);
```

where `id` is a line trace number auto-generated during the build.

3. Rebuild the project (*Project>Build All*).
4. After rebuilding, a Warning dialog will ask, "File has been modified. Do you want to reload?" (if you have not already disabled this dialog.) Click **Yes**. When you examine your code, you will find that the log `id` has been replaced with a unique number.
5. Reprogram the device (*Debugger>Program*).
6. Run the program until it halts at the breakpoint. Repeat this three times.
7. Select *View>Trace* to view the trace data (**Section 12.3.11 "Trace Window"**) or right click and in the Trace window and select "Reload". You should see address values logged in this window. To see the related code in the lower portion of the window, you may need to click on a logged value in the upper portion of the window.

**FIGURE 4-21: VIEW TRACE WINDOW – TRACE LINE**



To trace multiple lines of code, you must place a macro before each line that you wish to trace.

## 4.13.3 Setting the Size of the Trace Buffer

In this tutorial, a breakpoint was used to ensure that the trace buffer did not overflow with values from an executing program. A "for" instead of "while" loop could be used as well to control the number of trace samples.

To set the size of the trace buffer:

1. Select *Debugger>Settings*, **Trace** tab.
2. Enter a value for the trace buffer, not to exceed the maximum specified on this tab.



## 4.13.4 Disabling Trace

To temporarily disable the trace capability:

1. Select *Project>Build Options>Project, Trace* tab. Check “Disable Trace Macros”. Click **OK**.
2. Rebuild the project (*Project>Build All*).
3. Reprogram the device (*Debugger>Program*).

To permanently disable the trace capability:

1. Remove all trace and log macros from code.
2. Select *Project>Build Options>Project, Trace* tab. Uncheck “Enable Trace”. Click **OK**.
3. Rebuild the project (*Project>Build All*).
4. Reprogram the device (*Debugger>Program*).

## 4.14 PROGRAMMING THE APPLICATION

When the program is successfully debugged and running, the next step is to program the device for stand-alone operation in the finished design. When doing this, the resources reserved for debug are released for use by the application.

To program the application follow these steps:

1. Disable the MPLAB REAL ICE in-circuit emulator as the debug tool by selecting *Debugger>Select Tool>None*.
2. Enable the MPLAB REAL ICE in-circuit emulator as the programmer by selecting *Programmer>Select Programmer>REAL ICE*.
3. *Optional:* Set up the ID in *Configure>ID Memory* (for devices that support ID memory.)
4. Set up the parameters for programming on the *Programmer>Settings, Program Memory* tab.
5. On the Project toolbar, select “Release” from the Build Configuration drop-down list. Then select *Project>Build All*.
6. Select *Programmer>Program*.

The application should now be running on its own. Press the Reset ( $\overline{\text{MCLR}}$ ) button on the demo board to restart the count.

You can modify the program code to wait for a button press before beginning or to terminate the program. Modifying the program will require you to select the emulator as a debug tool.

1. Disable the MPLAB REAL ICE in-circuit emulator as the programmer by selecting *Programmer>Select Programmer>None*.
2. Enable the MPLAB REAL ICE in-circuit emulator as the debug tool by selecting *Debugger>Select Tool>REAL ICE*.
3. Edit the `counter.c` code as desired. (This is left as an exercise for you.)
4. On the Project toolbar, select “Debug” from the Build Configuration drop-down list. Then select *Project>Build All*.
5. Select *Debugger>Program*.
6. Run, step and debug your program as required.

## 4.15 OTHER TRACE METHODS – SPI OR I/O PORT TRACE

The Explorer 16 board does not have the connections to perform either SPI or I/O Port trace. Therefore, it must either be modified or a different board must be used that allows access to the required device SPI and Port pins.

The PIC24FJ128GA010 device and code from the beginning of this tutorial will be used in this section. Although this device has built-in (Native) trace capability, you may wish to use the other types of trace for speed or pin resource reasons.

For devices that do not have Native trace capability, SPI and I/O Port trace are the only forms of trace available. The procedures shown here can be modified to use with other Microchip devices supported by the MPLAB REAL ICE in-circuit emulator.

- Using SPI Trace
- Using I/O Port Trace

### 4.15.1 Using SPI Trace

The Explorer 16 board *does* have a place for connecting with the PICkit 2. This connection can be populated to provide connection to six of the eight high-speed communication pins. The remaining two pins - to the SPI pins SCK and SDO - will have to be hard-wired from two additional connector pins to the appropriate SPI pins on the device. See **Section 2.4.2 “High-Speed Communication Connection”** for details.

**Note:** The Explorer 16 silkscreen label for pin 1 of the PICkit 2 connector is incorrect. This is actually the location of pin 6.

You may also choose to use a target board of your own design that allows for access to the necessary debug and SPI pins. Either way, a hardware connection between the target board and the emulator's high-speed connector is required. See **Section 2.5.2 “SPI Trace Connections (High-Speed Communication Only)”** for more information.

#### 4.15.1.1 HARDWARE SETUP

To set up the hardware to use SPI Trace, do the following:

1. Obtain Microchip's Performance Pak, which contains the emulator high-speed communication boards and cables, if you have not already done so.

**Note:** High-speed communications is required to use SPI Trace.

2. Modify or create a target board, as specified above, so that it accommodates the high-speed connector.
3. Using an unpowered emulator and target board, insert the high-speed driver board into the emulator and the high-speed receiver board into the target board. Connect the boards with the included cables. See **Section 2.3.2 “High-Speed Communication”** for reference.
4. Power the emulator, and then the target board.

## 4.15.1.2 MPLAB IDE SETUP

To set up MPLAB IDE software to use SPI Trace, do the following:

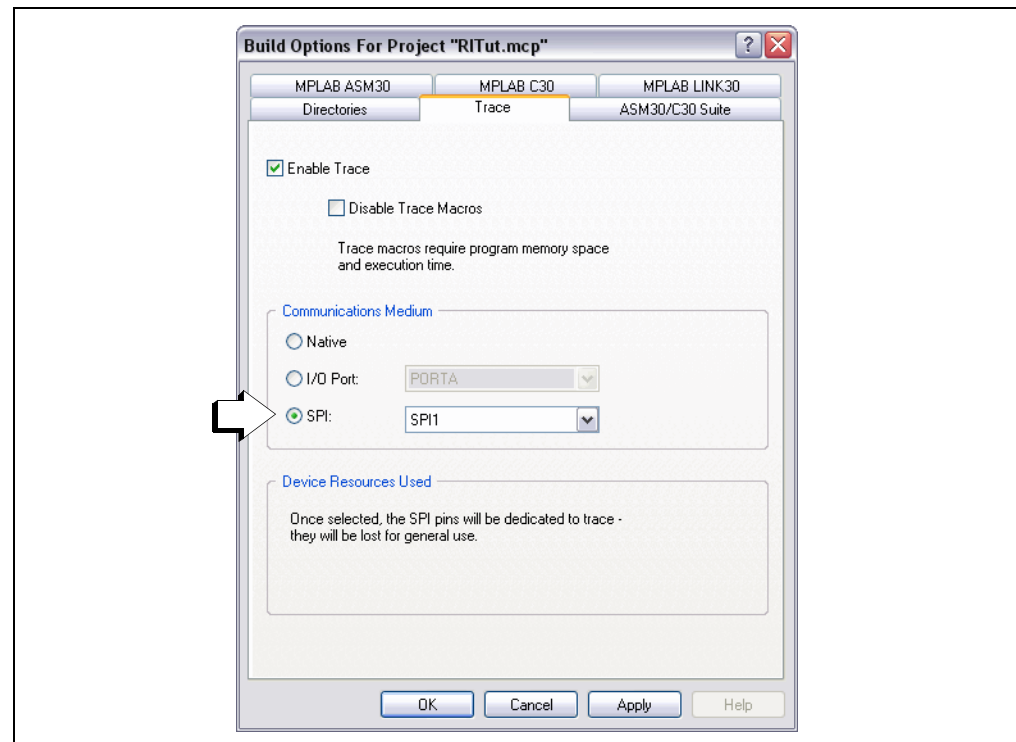
1. Launch MPLAB IDE and open the project from this tutorial, if it is not already open.
2. If the MPLAB REAL ICE in-circuit emulator is selected as the programmer, disable it by selecting *Programmer>Select Programmer>None*.
3. Enable the MPLAB REAL ICE in-circuit emulator as the debug tool by selecting *Debugger>Select Tool>REAL ICE*.
4. On the Project toolbar, select “Debug” from the Build Configuration drop-down list. (For some devices, an *i* version of the linker script is also necessary for debugging, e.g., *18F8722i.lkr*.)

## 4.15.1.3 TRACE SETUP

To log a variable value using SPI Trace:

1. Select *Project>Build Options>Project, Trace* tab. Check “Enable Trace” and uncheck “Disable Trace Macros”. Then select the SPI Trace and choose an SPI port from the drop-down list. Click **OK**.

**FIGURE 4-22: BUILD OPTIONS DIALOG – SPI TRACE**



2. Highlight the variable `counter` from the following line of code:

```
counter++; //increment counter
```

Right click on the highlighted variable and select “Log Selected C Value” from the pop-up menu. This causes the following macro line to be inserted above the line containing the variable:

```
__LOG(id,counter);
```

where `id` is a log number auto-generated during build.

3. Rebuild the project (*Project>Build All*).

4. After rebuilding, a Warning dialog will ask, "File has been modified. Do you want to reload?". Click **Yes**. When you examine your code, you will find that the log `id` has been replaced with a unique number.

**Note:** To disable this warning and automatically reload, select Configure>Settings, **Other** tab, and check "Automatically reload files that were modified outside of the IDE". Then click **OK**.

5. Reprogram the device (Debugger>Program).
6. Double-click on the following line to place a breakpoint there:  

```
PORTA = counter; //display on port LEDs
```
7. Reset and run the program until it halts at the breakpoint. Repeat this three times.
8. Select View>Trace to view the trace data (**Section 12.3.11 "Trace Window"**) or right click and in the Trace window and select "Reload". You should see variable values logged in this window. To see the related code in the lower portion of the window, you may need to click on a logged value in the upper portion of the window.

## 4.15.2 Using I/O Port Trace

You may choose to modify the Explorer 16 demo board or use a target board of your own design to allow for access to the desired port pins. Either way, a hardware connection between the target device's port and the emulator's logic probe connector is required. See **Section 2.5.3 "I/O Port Trace Connections"** for details.

If you design your own board, you will also need to a connector for regular debug pins, i.e., for either standard or high-speed communications. See **Section 2.3 "Emulator Communications with the PC and Target"** for connection information.

### 4.15.2.1 HARDWARE SETUP

To set up the hardware to use I/O Port Trace, do the following:

1. Modify or create a target board, as specified above, so that it accommodates a connection between the emulator and the device port.
2. Using an unpowered emulator and target board, connect the two using either standard or high-speed communications.
3. Connect the emulator's logic probe pins to the target device's port pins using logic probes or other connectors.
4. Power the emulator, and then the target board.

### 4.15.2.2 MPLAB IDE SETUP

To set up MPLAB IDE software to use I/O Port Trace, do the following:

1. Launch MPLAB IDE and open the project from this tutorial, if it is not already open.
2. If the MPLAB REAL ICE in-circuit emulator is selected as the programmer, disable it by selecting Programmer>Select Programmer>None.
3. Enable the MPLAB REAL ICE in-circuit emulator as the debug tool by selecting Debugger>Select Tool>REAL ICE.
4. On the Project toolbar, select "Debug" from the Build Configuration drop-down list. (For some devices, an `i` version of the linker script is also necessary for debugging, e.g., `18F8722i.lkr`.)

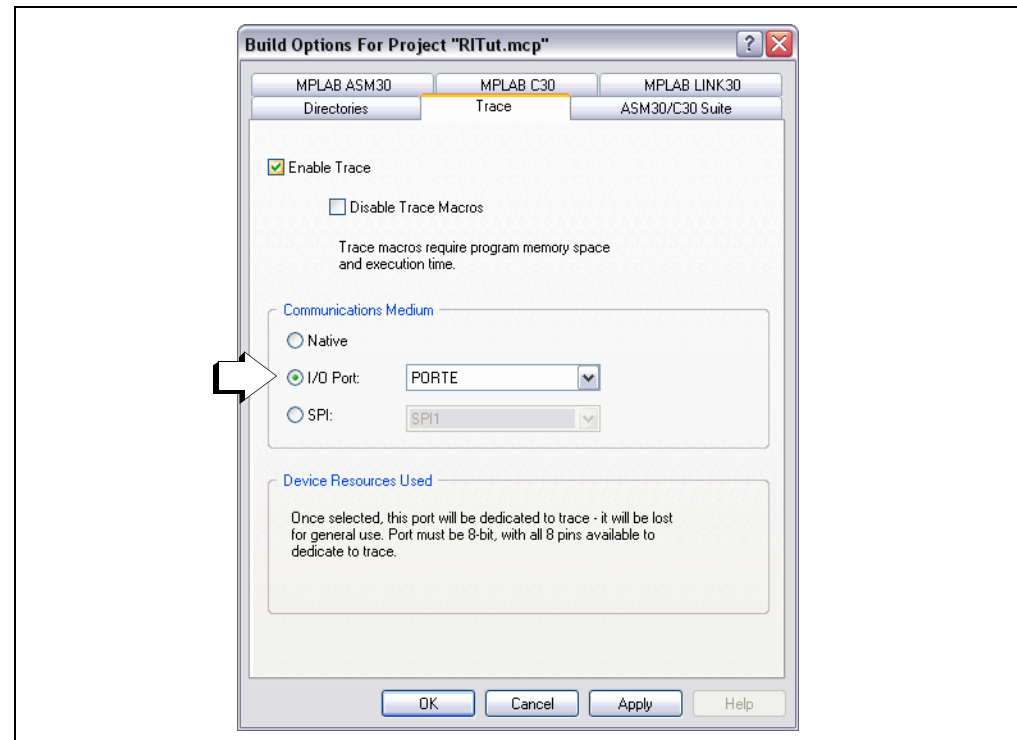
## 4.15.2.3 TRACE SETUP

To log a variable value using I/O Port Trace:

1. Select *Project>Build Options>Project, Trace* tab. Check “Enable Trace” and uncheck “Disable Trace Macros”. Then select the I/O Port Trace and choose an port from the drop-down list. Click **OK**.

**Note:** Determining a port that may be dedicated to trace can be a difficult task on complex devices with many functions multiplexed on port pins. One suggestion is to use the MPLAB VDI visual device initializer. Not only can you use it to create initialization code for your application, but you can add a port to see if it conflicts with any of your other application peripheral pins. MPLAB VDI information is stored with the workspace.

**FIGURE 4-23: BUILD OPTIONS DIALOG – I/O PORT TRACE**



2. Highlight the variable `counter` from the following line of code:

```
counter++; //increment counter
```

Right click on the highlighted variable and select “Log Selected C Value” from the pop-up menu. This causes the following macro line to be inserted above the line containing the variable:

```
__LOG(id, counter);
```

where `id` is a log number auto-generated during build.

3. Rebuild the project (*Project>Build All*).

4. After rebuilding, a Warning dialog will ask, "File has been modified. Do you want to reload?". Click **Yes**. When you examine your code, you will find that the log `id` has been replaced with a unique number.

**Note:** To disable this warning and automatically reload, select Configure>Settings, **Other** tab, and check "Automatically reload files that were modified outside of the IDE". Then click **OK**.

5. Reprogram the device (Debugger>Program).
6. Double-click on the following line to place a breakpoint there:  

```
PORTA = counter; //display on port LEDs
```
7. Reset and run the program until it halts at the breakpoint. Repeat this three times.
8. Select View>Trace to view the trace data (**Section 12.3.11 "Trace Window"**) or right click and in the Trace window and select "Reload". You should see variable values logged in this window. To see the related code in the lower portion of the window, you may need to click on a logged value in the upper portion of the window.

## 4.16 OTHER TRACE METHODS – PIC32 INSTRUCTION TRACE

PIC32 Instruction Trace is only available for PIC32MX MCU devices, and it is the only type of trace available for these devices. Also, only some PIC32MX MCU devices have the trace feature. Consult the device data sheet for details.

For example code and additional supporting hardware, refer to the Microchip website ([www.microchip.com](http://www.microchip.com)).

To use this trace, you will need:

- PIC32MX Plug-In Module (PIM) containing a device that supports trace and a trace port
- PIC32MX Trace Interface Kit containing a 12-inch trace cable and a trace adapter board

The PIC32MX360F512L PIM (MA320001) will plug into an Explorer 16 board. Follow the instructions specified in **Section 8.3.2 “Setting Up and Using Trace”**.

Once the hardware is connected, you enable trace through the *Debugger>Settings, Trace* tab (Figure 4-24). Trace data will appear in the Trace window (Figure 4-25).

For more information, see **Section 8.3 “PIC32 Instruction Trace”**.

**FIGURE 4-24: PIC32 INSTRUCTION TRACE ENABLE**

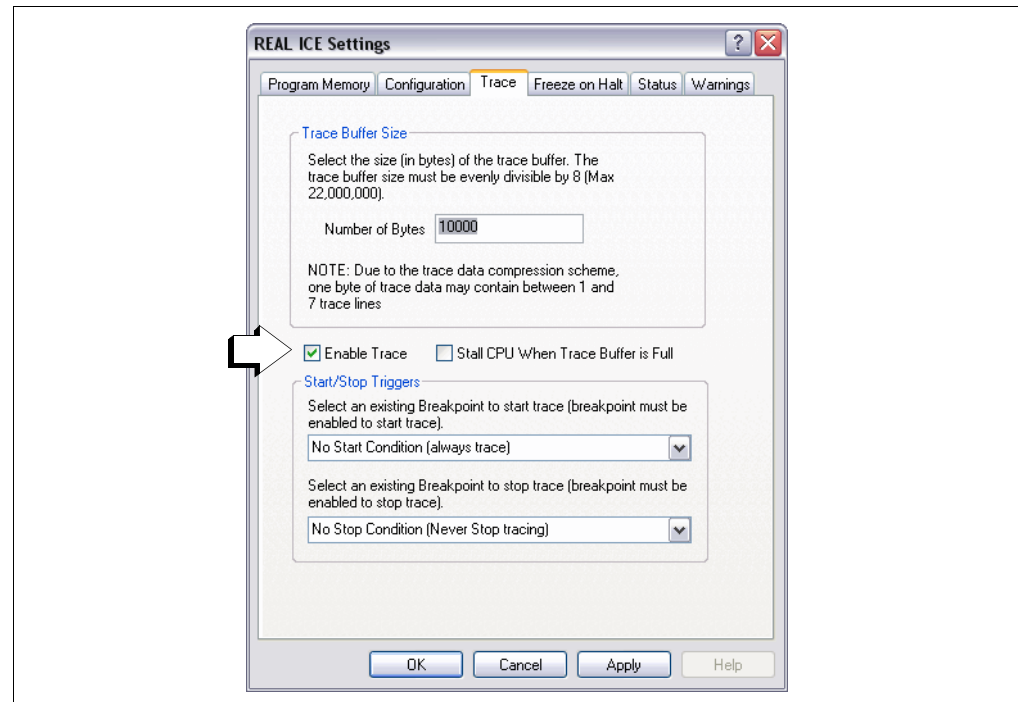


FIGURE 4-25: PIC32 INSTRUCTION TRACE DATA

The screenshot shows the 'Trace' window in the MPLAB REAL ICE In-Circuit Emulator. The window displays a table of instruction trace data and the corresponding source code for the function 'mPORTAClearBits'.

| Line | Program Address | Op       | Label     | Instruction |
|------|-----------------|----------|-----------|-------------|
| 660  | 9D00001C        | 0F40000B | jal       | 0x9d00002c  |
| 661  | 9D000020        | 00000000 | nop       |             |
| 662  | 9D00002C        | 27BDFFFC | addiu     | sp,sp,-4    |
| 663  | 9D000030        | AFB00000 | sw        | s0,0(sp)    |
| 664  | 9D000034        | 3C10BF88 | lui       | s0,0xbf88   |
| 665  | 9D000038        | 26106024 | addiu     | s0,s0,2461: |
| 666  | 9D00003C        | AED40000 | sw        | a0,0(s0)    |
| 667  | 9D000040        | 8FB00000 | lw        | s0,0(sp)    |
| 668  | 9D000044        | 27BD0004 | addiu     | sp,sp,4     |
| 669  | 9D000048        | 03E00008 | jr        | ra          |
| 670  | 9D00004C        | 00000000 | nop       |             |
| 671  | 9D000024        | 0B400009 | endless j | 0x9d000024  |

The source code for the function 'mPORTAClearBits' is shown below the trace data:

```

E:\Projects32\PIC32MX\asm32\ports_control\source\ports_control.S
117      /* function epilogue - restore registers used in this function
118      * from stack and adjust stack-pointer
119      */
120      lw    $s0, 0($sp)
121      addiu $sp, $sp, 4
122
123      /* return to caller */
124      jr    $ra
125      nop
126      .end mPORTAClearBits
    
```





# MPLAB® REAL ICE™ IN-CIRCUIT EMULATOR USER'S GUIDE

---

---

## Part 3 – Features

---

---

|  |    |
|--|----|
| Chapter 5. General Setup .....                   | 67 |
| Chapter 6. Basic Debug Functions.....            | 73 |
| Chapter 7. Debug for 8- and 16-Bit Devices ..... | 75 |
| Chapter 8. Debug for 32-Bit Devices .....        | 83 |

# MPLAB® REAL ICE™ In-Circuit Emulator User's Guide

---

NOTES:



---

---

## Chapter 5. General Setup

---

---

### 5.1 INTRODUCTION

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

- Starting the MPLAB IDE Software
- Creating a Project
- Viewing the Project
- Building the Project
- Setting Configuration Bits
- Setting the Emulator as the Debugger or Programmer
- Quick Debug/Program Reference
- Debugger/Programmer Limitations

### 5.2 STARTING THE MPLAB IDE SOFTWARE

After installing the MPLAB IDE software (**Section 3.2 “Installing the Software”**), invoke it by using any of these methods:

- Select *Start>Programs>Microchip>MPLAB IDE vx.xx>MPLAB IDE*, where vx.xx is the version number.
- Double click the MPLAB IDE desktop icon.
- Execute the file `mplab.exe` in the `\core` subdirectory of the MPLAB IDE installation directory.

For more information on using the software, see:

- “*MPLAB IDE User's Guide*” (DS51519) – Comprehensive guide for using MPLAB IDE.
- “*MPLAB IDE Quick Start Guide*” (DS51281) – Chapters 1 and 2 of the user's guide.
- The on-line help files – The most up-to-date information on MPLAB IDE and MPLAB REAL ICE in-circuit emulator.
- Readme files – Last minute information on each release is included in `Readme for MPLAB IDE.txt` and `Readme for MPLAB REAL ICE Emulator.txt`. Both files are found in the `Readmes` subdirectory of the MPLAB IDE installation directory.

## 5.3 CREATING A PROJECT

The easiest way to create a new project is to select *Project>Project Wizard*. With the help of the Project Wizard, a new project and the language tools for building that project can be created. The wizard will guide you through the process of adding source files, libraries, linker scripts, etc. to the various “nodes” on the project window. See MPLAB IDE documentation for more detail on using this wizard. The basic steps are provided here:

- Select your device (e.g., PIC24FJ128GA010)
- Select a language toolsuite (e.g., Microchip C30 Toolsuite)
- Name the project
- Add application files (e.g., `program.c`, `support.s`, `counter.asm`)
- Add a linker script file (optional) - Most multi-file projects no longer require that you add a linker script file to your project. See the MPLAB IDE documentation for details.

If you do need or want to add a linker script file, default linker scripts directories are listed below. For some debug tools, an `i` version of the linker script is necessary for debugging, e.g., `18F8722i.lkr`.

**MPLINK™ Object Linker** used with:

- MPASM™ Assembler  
C:\Program Files\Microchip\MPASM Suite\LKR
- MPLAB C Compiler for PIC18 MCUs (formerly MPLAB C18)  
C:\MCC18\lkr

**MPLAB Object Linker for PIC24 MCUs and dsPIC DSCs** (formerly MPLAB LINK30) used with:

- MPLAB Assembler for PIC24 MCUs and dsPIC DSCs (formerly MPLAB ASM30)  
C:\Program Files\Microchip\MPLAB ASM30 Suite\Support\gld
- MPLAB C Compiler for PIC24 MCUs and dsPIC DSCs (formerly MPLAB C30)  
C:\Program Files\Microchip\MPLAB C30\support\gld  
C:\pic30\_tools\support\gld

**MPLAB Object Linker for PIC32MX MCUs** (formerly MPLAB LINK32) used with:

- MPLAB Assembler for PIC32MX MCUs (formerly MPLAB ASM32)  
C:\Program Files\Microchip\MPLAB ASM32 Suite\Support\ld
- MPLAB C Compiler for PIC32MX MCUs (formerly MPLAB C32)  
C:\Program Files\Microchip\MPLAB C32\support\ld

## 5.4 VIEWING THE PROJECT

After the Project Wizard has created a project, the project and its associated files are visible in the Project window (*View>Project*). Additional files can be added to the project using the Project window. Right click on any line in the project window tree to pop up a menu with additional options for adding and removing files.

See MPLAB IDE documentation for more detail on using the Project window.

## 5.5 BUILDING THE PROJECT

After the project is created, the application needs to be built. This will create object (hex) code for the application that can be programmed into the target by the MPLAB REAL ICE in-circuit emulator.

To set build options, select *Project>Build Options>Project*.

**Note:** On the Project Manager toolbar, select “Debug” from the drop-down list.

When done, choose *Project>Build All* to build the project.

## 5.6 SETTING CONFIGURATION BITS

Although device Configuration bits may be set in code, they also may be set in the MPLAB IDE Configuration window. Select *Configure>Configuration Bits*. By clicking on the text in the “Settings” column, these can be changed.

Some Configuration bits of interest are:

- **Oscillator** – Make sure the correct mode and other oscillator features are set to match the physical setup on the target board.
- **Watchdog Timer Enable** – On most devices, the Watchdog Timer is enabled initially. It is usually a good idea to disable this bit.
- **Comm Channel Select** – For some devices, you will need to select the communications channel for the device, e.g., PGC1/EMUC1 and PGD1/EMUD1. Make sure the pins selected here are the same ones physically connected to the device.
- **Code Protect/Table Read Protect** – Disable these settings. If the emulator cannot write to program memory or read from a table, it cannot operate properly.
- **JTAG Port Enable** – For PIC32MX devices, this port may need to be disabled to prevent conflicts when using trace and other ICE features where pin conflicts can result.

## 5.7 SETTING THE EMULATOR AS THE DEBUGGER OR PROGRAMMER

Select *Debugger>Select Tool>MPLAB REAL ICE* to choose the MPLAB REAL ICE in-circuit emulator as the debug tool. The Debugger menu and MPLAB IDE toolbar will change to display debug options once the tool is selected. Also, the Output window will open and messages concerning ICE status and communications will be displayed on the **MPLAB REAL ICE** tab. For more information, see **Section 12.2 “Debugging Functions”** and **Section 12.3 “Debugging Dialogs/Windows”**.

Select *Programmer>Select Programmer>MPLAB REAL ICE* to choose the MPLAB REAL ICE in-circuit emulator as the programmer tool. The Programmer menu and MPLAB IDE toolbar will change to display programmer options once the tool is selected. Also, the Output window will open and messages concerning ICE status and communications will be displayed on the **MPLAB REAL ICE** tab. For more information, see **Section 12.4 “Programming Functions”**.

Based on your selected device, updated debug firmware may need to be downloaded into the device. You should allow this to progress automatically. If you need to download manually or if you interrupted the download, see **Chapter 10. “Frequently Asked Questions (FAQ)”**.

# MPLAB® REAL ICE™ In-Circuit Emulator User's Guide

Select *Debugger>Settings* or *Programmer>Settings* to open the Settings dialog (**Section 12.5 “Settings Dialog”**) and set up options as needed. These options include:

- Program Memory – Memory programming options, including preserving a range of program memory and automating the debug built-program-run procedure.
- Configuration – Firmware downloading control and breakpoint setup.
- Trace – Trace buffer and PIC32 instruction trace setup.
- Freeze on Halt – Select peripherals to freeze on halt.
- Status – Check emulator status.
- Clock – Enter target clock information for data capture and trace.
- Secure Segment – CodeGuard Security support.
- Warnings – Enable/disable warnings.

If errors occurs, see:

- **Part 4 – “Troubleshooting”**
- **Section 13.9 “Loop-Back Test Board”**

## 5.8 QUICK DEBUG/PROGRAM REFERENCE

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

**TABLE 5-1: 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 header board with special -ICE device.  | Device (with or without on-board debug circuitry).  |
| MPLAB® IDE selection   | <i>Debugger&gt;Select Tool&gt;REAL ICE</i>  | <i>Programmer&gt;Select Programmer&gt;REAL ICE</i>  |
|  | “Debug” from the Build Configuration toolbar.   | “Release” from the Build Configuration toolbar.   |
| Linker script (optional)<br>See MPLAB IDE documentation to determine if you need to add a linker script to your project. | If you need to add a linker script to your project, use the “i” version of the linker script, e.g., 18F452i.lkr.  | If you need to add a linker script to your project, use the standard linker script, e.g., 18F452.lkr.   |
| Program operation  | Programs application code into the device. Depending on the selections on the Program tab of the Settings dialog, this can be any combination of program memory, EEPROM memory, configuration bits, or ID memory.<br>In addition, a small debug executive is placed in program memory and other debug resources are reserved. | Programs application code into the device. Depending on the selections on the Program tab of the Settings dialog, this can be any combination of program memory, EEPROM memory, configuration bits, or ID memory. |
| Debug features available   | All for device – breakpoints, trace, etc.   | N/A.  |
| SQTP   | N/A   | Use the MPLAB PM3 to generate the SQTP file. Then use the emulator to program the device.   |
| Command-line operation   | N/A   | Use REALICECMD, found by default in: C:\Program Files\Microchip\MPLAB IDE\Programmer Utilities\RealICE.   |

## 5.9 DEBUGGER/PROGRAMMER LIMITATIONS

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

# MPLAB® REAL ICE™ In-Circuit Emulator User's Guide

---

NOTES:



---

---

## Chapter 6. Basic Debug Functions

---

---

### 6.1 INTRODUCTION

Basic MPLAB REAL ICE in-circuit emulator debug functions are discussed.

- Breakpoints and Stopwatch
- External Triggers

### 6.2 BREAKPOINTS AND STOPWATCH

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

Breakpoints and real-time data capture triggers use the same resources. Therefore, the available number of breakpoints is actually the available number of combined breakpoints/triggers.

The number of hardware and software breakpoints available and/or used is displayed in the Device Debug Resource toolbar. See the MPLAB IDE documentation for more on this feature.

To select hardware or software breakpoints:

1. Select *Debugger>Settings* and click the **Configuration** tab.
2. Select the desired type of breakpoints for your application. A list of features for each breakpoint type - hardware or software - is shown under that type. (See **Section 12.5.2 “Settings Dialog, Configuration Tab”** for more information.)

|   |
|---|
| <p><b>Note:</b> Using software breakpoints for debug impacts device endurance. Therefore, it is recommended that devices used in this manner not be used as production parts.</p> |
|---|

To set a breakpoint in code, do one of the following:

- Double-click or right click on a line of code to set up an individual breakpoint.
- Select *Debugger>Breakpoints* to open the Breakpoints dialog and set up multiple breakpoints and breakpoint conditions. See **Section 12.3.1 “Breakpoints Dialog”** for more information.

To determine the time between the breakpoints, use the stopwatch:

1. Open the Breakpoints dialog (*Debugger>Breakpoints*).
1. Click **Stopwatch** on the Breakpoints dialog to open the Stopwatch dialog.
2. Under “Start Condition”, click **Select Start Condition** and choose a breakpoint. Also decide if “Start condition will cause the target device to halt”.
3. Under “Stop Condition”, click **Select Stop Condition** and choose another breakpoint. Also decide if “Stop condition will cause the target device to halt”.
4. Decide if there will be a “Reset stopwatch on run”.
5. Click **OK**.

## 6.3 EXTERNAL TRIGGERS

Select *Debugger>Triggers* to open the Triggers dialog to set up external triggers. See **Section 12.3.7 “Triggers Dialog”** for more information.

Use external triggers to set up hardware triggers using the logic probe port. All pins (whether used or unused) should either be pulled up or grounded. Floating pins may produce false triggers.

To use probe pins as inputs, you must provide the circuitry to drive them (see **Table 13-2: “Logic Probe Electrical Specifications”** for drive levels.)

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

- **Section 7.3.3.2 “I/O Port Trace”**
- **Section 8.3 “PIC32 Instruction Trace”**

For more on external trigger hardware, see **Section 13.5.4 “Logic Probe/External Trigger Interface”**.

---

---

## Chapter 7. Debug for 8- and 16-Bit Devices

---

---

### 7.1 INTRODUCTION

The following debug functions are specific to 8- and 16-bit devices.

- Data Capture and Runtime Watches
- Trace

### 7.2 DATA CAPTURE AND RUNTIME WATCHES

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

- Data Monitoring and Control Interface (DMCI) – Tools menu

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

- Watch – View menu
- File Register – View menu
- Special Function Register (SFR) – View menu

Data captures and runtime watches use the same resource. Therefore, setting one or both uses the resource for the selected symbol.

To set up data captures and/or runtime watches:

1. Select *View>Watch* to open the Watch window to set up data capture and/or runtime watches for specific data addresses. See **Section 12.3.9 “Watch Window - Data Capture/Runtime Watch”** for more information.
2. Select *Debugger>Settings* and click the **Clock** tab. For data capture and trace, the emulator needs to know the instruction cycle speed. (See **Section 12.5.6 “Settings Dialog, Clock Tab”** for more information.) Enter your information here.
3. Rebuild the project (*Project>Build All*) and reprogram the target device (*Debugger>Program*).
4. Run the program. View the data in a DMCI window or watch variable values change in an MPLAB IDE window.

## 7.3 TRACE

This section will discuss the types of available trace for 8- and 16-bit devices and how to use them. See **Section 12.3.11 “Trace Window”** for information on the trace window.

### 7.3.1 Requirements for Trace

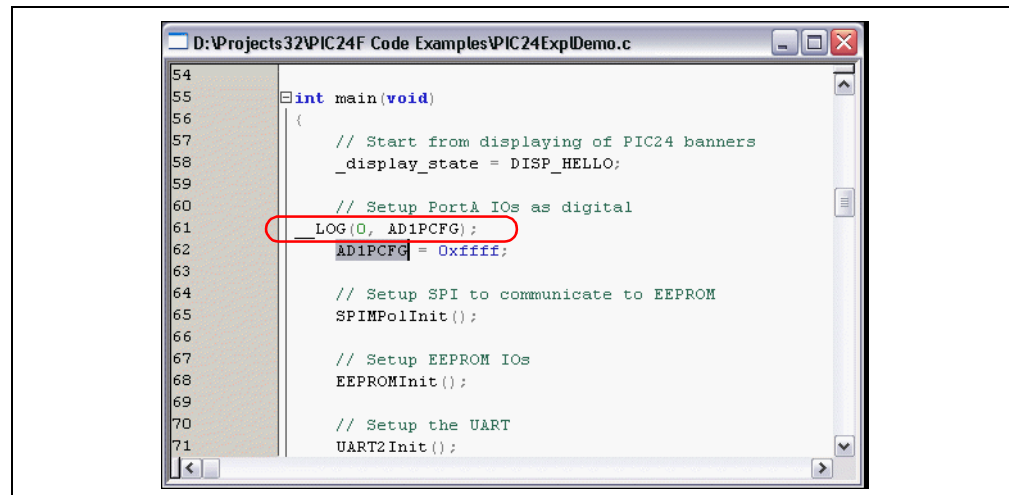
The following is required to use trace:

- For 16-bit devices (dsPIC30F/33F, PIC24F/H): MPLAB IDE v7.43 and above, MPLAB C30 v2.04 and above. For these devices, only C code can be traced, not assembly.
- For 8-bit devices (PIC18): MPLAB IDE v7.52 and above, MPASM toolsuite v5.10 and above, MPLAB C18 v3.10 and above.
- In-line assembly code (assembly code within C code) cannot be traced.

### 7.3.2 How Trace Works

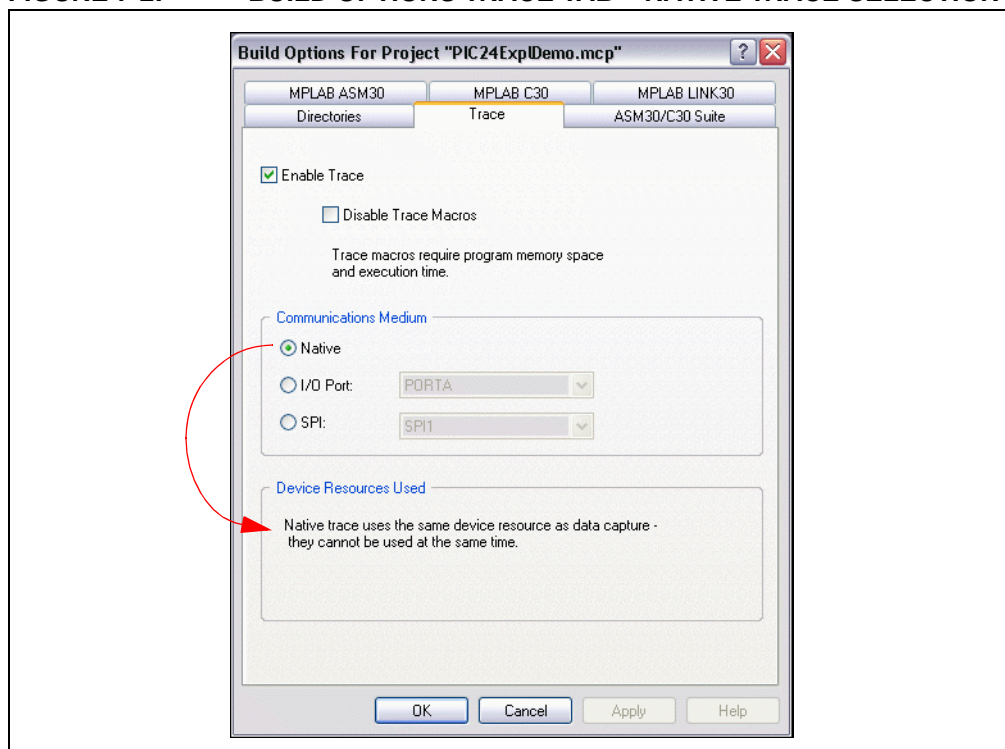
Trace for the MPLAB REAL ICE in-circuit emulator (Instrumented Trace) is a solution for providing basic trace information. Through the use of `TRACE()` and `LOG()` macros, you can report program locations or variable values to MPLAB IDE while the application is running. You may type these macro names 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 7-1: EXAMPLE OF INSERTED LOG MACRO**



There are three trace methods available at this time (see **Section 7.3.3 “Types of Trace”**.) The mediums can be found on the *Project>Build Options>Project, Trace* tab. The choices 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, and C projects only on 16-bit devices.

**FIGURE 7-2: BUILD OPTIONS TRACE TAB – NATIVE TRACE SELECTION**



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 which 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 which sent the variable. Here, the location is represented by a value between 0 and 127. (See **Section 7.3.10 “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 ones from a high priority interrupt.

### 7.3.3 Types of Trace

Currently there are three types of trace. All types are language tool version dependent and stream data 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  | Trace Usage               |  | Communication      |                      |
|----------------|---------------------------|--|--------------------|----------------------|
|                | Advantage                 | Disadvantage                               | Std                | HS                   |
| Native Trace   | No dedicated pins needed. | Device must have built-in debug circuitry. | 15 MIPS or less    | Greater than 15 MIPS |
| I/O Port Trace | Fastest trace method.     | An 8-bit port must be dedicated to trace.  | Yes <sup>(1)</sup> | Yes <sup>(1)</sup>   |
| SPI Trace      | Faster than Native Trace. | SPI pins must be dedicated to trace.       | No                 | Yes                  |

**Note 1:** Also requires connection from device port to emulator logic probe port.

## 7.3.3.1 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 7.3.5 “Setting Up Trace in MPLAB IDE”**).

If Native trace is used, then real-time data capture triggers cannot be used because of hardware constraints. However, breakpoints are still available. To use data capture triggers, you must disable Native trace (see **Section 7.3.8 “Disabling Trace”**).

## 7.3.3.2 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 PGM 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 2.5.3 “I/O Port Trace Connections”**.

The port interface uses the trace macro format (see **Section 7.3.5 “Setting Up Trace in MPLAB IDE”**).

## 7.3.3.3 SPI TRACE

SPI trace can be used only with high-speed communications. Trace clock and data are provided through pins 7 (DAT) and 8 (CLK) of the high-speed connection.

For hardware connections, see **Section 2.5.2 “SPI Trace Connections (High-Speed Communication Only)”**.

The SPI interface uses the trace macro format (see **Section 7.3.5 “Setting Up Trace in MPLAB IDE”**).

## 7.3.4 Setting Up the Project for Trace

Refer to **Chapter 5. “General Setup”** for a discussion of how to set up MPLAB IDE and an MPLAB IDE project to use the MPLAB REAL ICE in-circuit emulator.

To enable trace:

- Select *Project>Build Options>Project, Trace* tab. (See **Section 12.3.12 “Build Options Dialog, Trace Tab (Device Dependent)”**.)
- If it is not already selected, click to check “Enable Trace”. For full trace capability, “Disable Trace Macros” should be unchecked.
- Select the type of trace you want (**Section 7.3.3 “Types of Trace”**).
- Click **OK**.

## 7.3.5 Setting Up Trace in MPLAB IDE

To set up MPLAB IDE to use trace for the MPLAB REAL ICE in-circuit emulator, first set up options on the Settings dialog (*Debugger>Settings*.)

- Click the **Clock** tab. For data capture and trace, the emulator needs to know the instruction cycle speed. (See **Section 12.5.6 “Settings Dialog, Clock Tab”** for more information.)
- Click the **Trace** tab. The trace buffer can be set to a maximum value specified on the tab. The trace buffer is circular, so data will wrap if the maximum is exceeded.

Next, enter trace macros in 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 7.3.10 “More on Trace/Log ID Numbers”**.

**Note:** Inserting a macro into code may modify the logic flow of the program. Please be sure that braces are present where necessary.

- The recording of a variable value is performed much in the same way. First highlight the variable name or expression and 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 auto-generated during build and *selected variable* is the highlighted variable. For more information, see **Section 7.3.10 “More on Trace/Log ID Numbers”**.

- To remove a trace point, simply highlight and then delete the Trace/Log macro.

### 7.3.6 Running Trace

1. On the Project Manager toolbar, select “Debug” from the Build Configuration drop-down list.
2. Rebuild the project (*Project>Build All*).
3. After rebuilding, if there are trace macros in code, a Warning dialog will ask, “File has been modified. Do you want to reload?”. Click **Yes**. When you examine your code, you will find that all *ids* have been replaced with unique numbers.

**Note:** To disable this warning and automatically reload, select *Configure>Settings, Other* tab, and check “Automatically reload files that were modified outside of the IDE”. Then click **OK**.

4. Reprogram the device (*Debugger>Program*).
5. Run the program and then halt, or set a breakpoint to halt.
6. Select *View>Trace* to view the trace data (**Section 12.3.11 “Trace Window”**) or right click and in the Trace window and select “Reload”. 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.

### 7.3.7 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>Build Options>Project, Trace* tab, select “Disable Trace Macros“.)

### 7.3.8 Disabling Trace

To temporarily turn off trace data collection:

1. Select *Project>Build Options>Project, Trace* tab. Check “Disable Trace Macros”. Click **OK**.
2. Rebuild the project (*Project>Build All*).
3. Reprogram the device (*Debugger>Program*).

To disable the full trace capability:

1. Remove all trace and log macros from code.
2. Select *Project>Build Options>Project, Trace* tab. Uncheck “Enable Trace”. Click **OK**.
3. Rebuild the project (*Project>Build All*).
4. Reprogram the device (*Debugger>Program*).

### 7.3.9 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 9 instructions specified in the 16-bit library size for `memcpy()`.

#### EXAMPLE 7-1: 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        |
| <code>__TRACE(id)</code> instruction cycles     | 80      | 54      | 42       |
| <code>__LOG(id, BYTE)</code> instruction cycles | 168     | 90      | 57       |

#### EXAMPLE 7-2: 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        |
| <code>__TRACE(id)</code> instruction cycles    | 79      | 71      | 55       |
| <code>__LOG(id, INT)</code> instruction cycles | 225     | 169     | 162      |



# Debug for 8- and 16-Bit Devices

## EXAMPLE 7-3: 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 7-4: 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 7-5: 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      |

### 7.3.10 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 128 log points. These limits are determined by port trace (8 bits). Bit 7 is used as a clock, thus leaving 7 bits for data (128). Bit 6 is a flag which indicates a trace record instead of a log record.

For a trace record (bit 6 is 1), 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 0 |

For a log record (bit 6 is 0), the low order bits identify the data type (t) and the log number is sent in the next byte (nnnnnnn), thus freeing up a full 128 values.

|       |   |   |   |   |   |   |       |
|-------|---|---|---|---|---|---|-------|
| clock | 0 | t | t | t | t | t | t     |
| bit 7 |   |   |   |   |   |   | bit 0 |

|       |   |   |   |   |   |   |       |
|-------|---|---|---|---|---|---|-------|
| clock | n | n | n | n | n | n | n     |
| bit 7 |   |   |   |   |   |   | bit 0 |

## 7.3.11 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 *Project>Build Options>Project, Trace* tab. For full trace capability, "Enable Trace" should be checked and "Disable Trace Macros" should be unchecked. Select the type of trace you want. (Make sure your hardware can support this choice.)
2. Select *Debugger>Settings*. Click the **Clock** tab and enter the instruction cycle speed. Click the **Trace** tab to change the size of the trace buffer.
3. Right click in your code to enter trace macros (`__TRACE`, `__LOG`) as desired.
4. On the Project Manager toolbar, select **Debug** from the Build Configuration drop-down list.
5. Rebuild your project, reprogram your target device and run your program.

---

---

## Chapter 8. Debug for 32-Bit Devices

---

---

### 8.1 INTRODUCTION

The following debug functions are specific to 32-bit devices:

- Data Capture and Runtime Watches
- PIC32 Instruction Trace

### 8.2 DATA CAPTURE AND RUNTIME WATCHES

At this time, MPLAB REAL ICE in-circuit emulator does not support data capture for 32-bit devices. Runtime watches, however, are supported through DMA Reads and Writes.

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

- Watch – View menu
- File Register – View menu

To set up runtime watches:

1. Select *View>Watch* to open the Watch window to set up runtime watches for specific data addresses. See **Section 12.3.9 “Watch Window - Data Capture/Runtime Watch”** for more information.
2. Select *Debugger>Settings* and click the **Clock** tab. The emulator needs to know the instruction cycle speed. (See **Section 12.5.6 “Settings Dialog, Clock Tab”** for more information.) Enter your information here.
3. Rebuild the project (*Project>Build All*) and reprogram the target device (*Debugger>Program*).
4. Run the program. Watch variable values change in an MPLAB IDE window.
5. Open the Settings dialog again and click on the **Runtime Watch** tab. Slide the slider to see the values change faster or slower.

**Note:** The Runtime Watch can stall the processor if another access is occurring to memory at the same time.

## 8.3 PIC32 INSTRUCTION TRACE

This section will discuss trace for 32-bit devices and how to use it.

- Requirements for Trace
- Setting Up and Using Trace
- Trace Hardware Specifications

### 8.3.1 Requirements for Trace

The following is required to use trace for 32-bit (PIC32MX) devices:

- MPLAB IDE v8.00 and above
- MPLAB ASM32/LINK32/C32 v1.00 and above
- PIC32MX Plug-In Module (PIM) containing a device that supports trace and a trace port
- PIC32MX Trace Interface Kit containing a 12-inch trace cable and a trace adapter board

### 8.3.2 Setting Up and Using Trace

If a PIC32MX MCU device has trace capability, it will be PIC32 Instruction Trace.

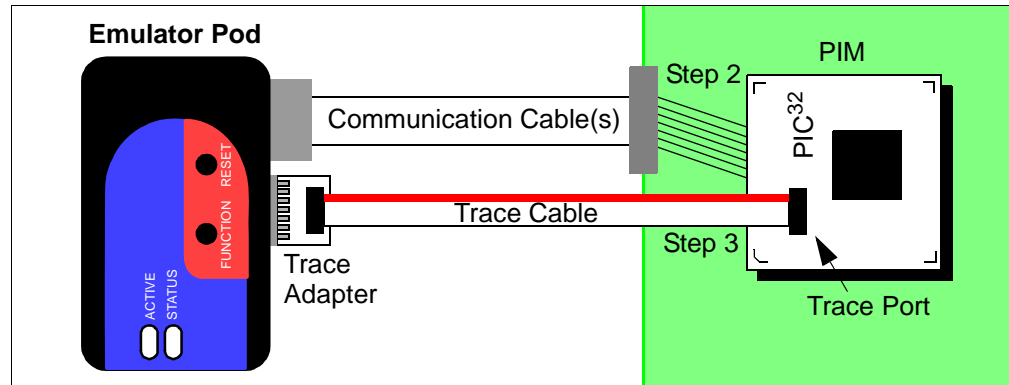
#### 8.3.2.1 HARDWARE SETUP - PIC32MX MCU ON PIM

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 2.4 “Target Communication Connections”**.
3. Connect the trace cable from the trace port on the PIM to the trace adapter board. Orient the cable as show in Figure 8-1.
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 8-1).
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.

FIGURE 8-1: TRACE CONNECTION WITH PIM



## 8.3.2.2 HARDWARE SETUP - PIC32MX MCU ON TARGET

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

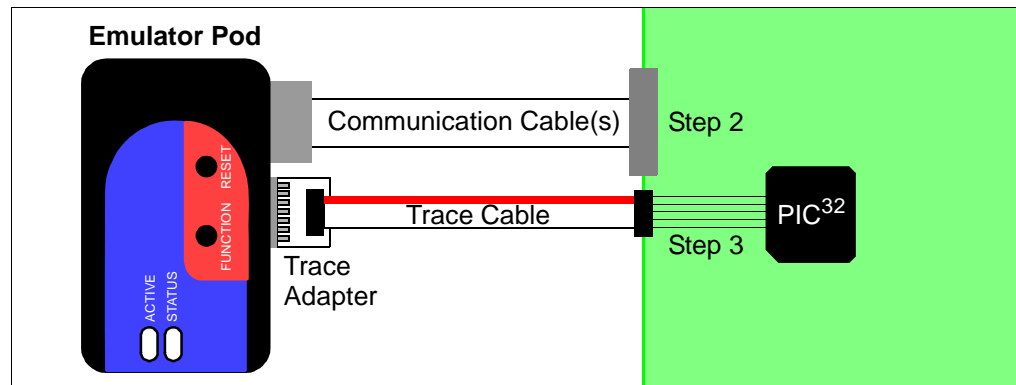
- Termination series resistors will need to be added as depicted in Figure 8-5.
- Depending on your board routing and loading of the signals used for trace; it is a good idea to place 0 ohm resistors that can be unpopulated to isolate the trace signals TRCLK and TRD3:0.

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 2.4 “Target Communication Connections”**.
3. Connect the trace cable from the target board to the trace adapter board. Make sure the PIC32MX MCU on your target board is connected to accommodate trace, as per Figure 8-5.
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 8-2).
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.

**FIGURE 8-2: TRACE CONNECTION WITH DEVICE ON TARGET**



## 8.3.2.3 MPLAB IDE SETUP

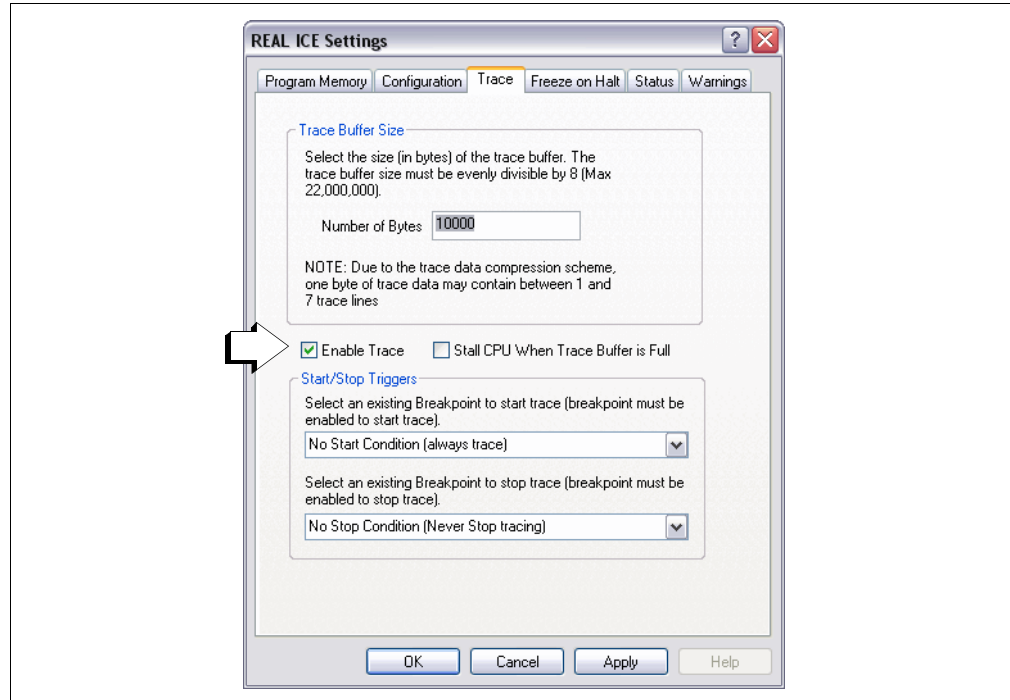
Once the hardware is connected, you enable trace in MPLAB IDE through the *Debugger>Settings, Trace* tab. Simply check/uncheck the “Enable Trace” checkbox to turn trace on/off (Figure 8-3). If no other options are selected, trace will continue to fill the trace buffer with data, rolling over when the buffer is full, until a program Halt.

To “stall” the target CPU when the trace buffer is full, check that option on the **Trace** tab. You can set the size of the trace buffer in the “Trace Buffer Size” section of the dialog. A maximum size is specified in the section text.

To start and/or stop trace on triggers/breaks, set and enable breakpoints in your code and then select them on this tab to start and/or stop trace.

**Note:** Once breakpoints have been used for trace, they will stay assigned as triggers until unselected. That is, even if “Enable Trace” is unchecked, these breakpoints will still be assigned as triggers. You must use the Start/Stop Triggers pull-down menus to unassign these triggers so that they are again breakpoints.

**FIGURE 8-3: PIC32 INSTRUCTION TRACE ENABLE**



### 8.3.2.4 VIEWING TRACE DATA

When trace is enabled and code is run, trace data will appear in the Trace window (*View>Trace*). See **Section 4.16 “Other Trace Methods – PIC32 Instruction Trace”** for an example of trace data in the Trace window.

## 8.3.3 Trace Hardware Specifications

Specifications for hardware that supports PIC32 Instruction Trace are listed below.

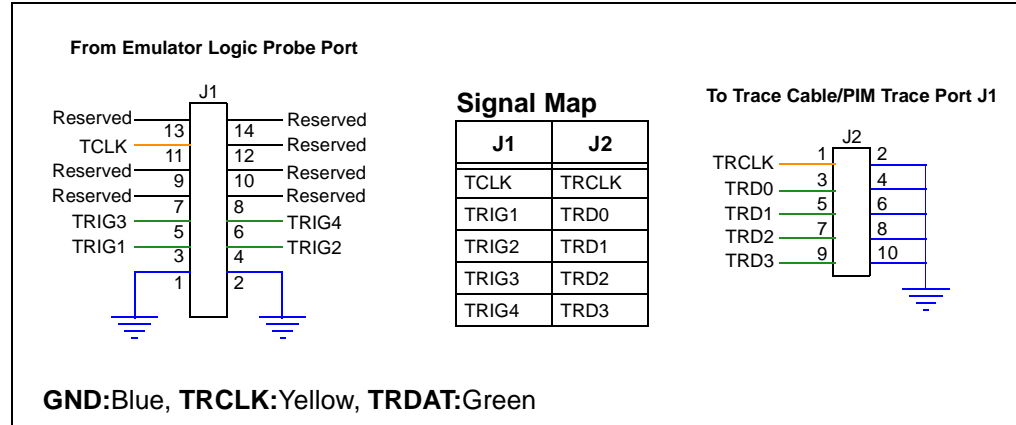
### 8.3.3.1 PIC32MX TRACE INTERFACE KIT (AC244006)

The PIC32MX Trace Interface Kit consists of an adapter board and trace cable. Kit component dimensions and a pin connection diagram for the adapter board are shown below.

**TABLE 8-1: KIT COMPONENT DIMENSIONS IN INCHES**

| Component     | Length | Width | Height |
|---------------|--------|-------|--------|
| Adapter Board | 0.900  | 0.800 | 0.6    |
| Cable         | 12.0   | 0.5   | 0.0625 |

**FIGURE 8-4: ADAPTER BOARD PIN CONNECTION DIAGRAM**



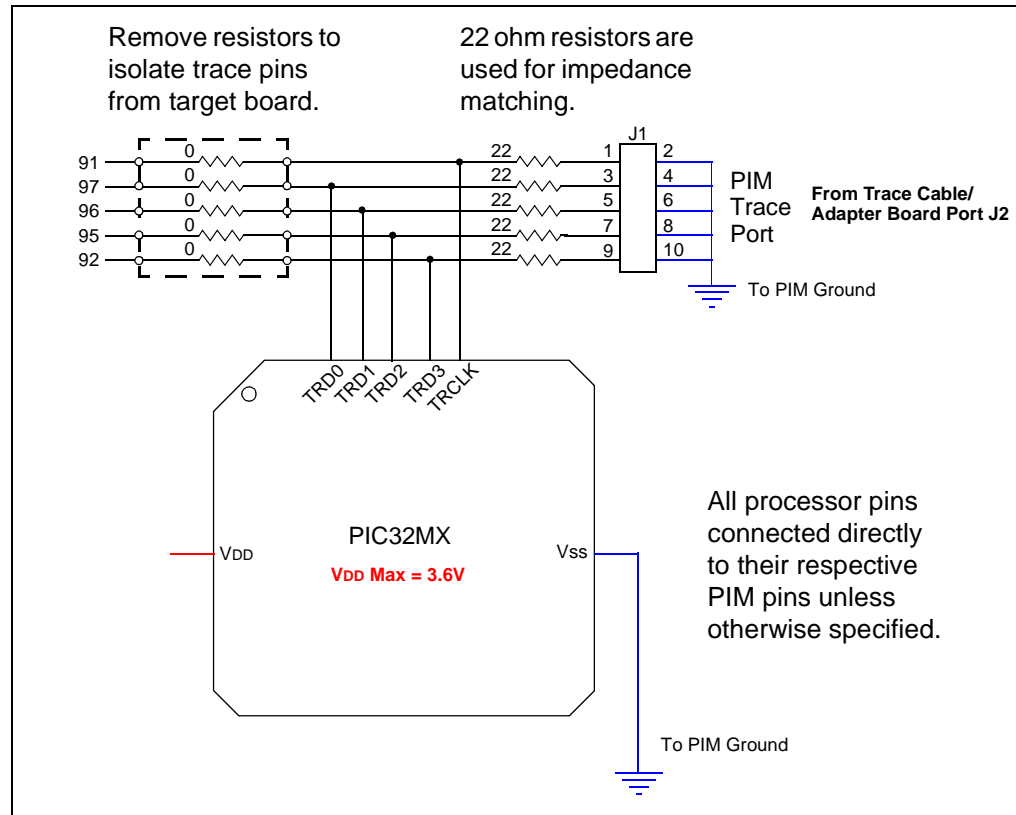
### 8.3.3.2 PIC32MX PIM

The PIC32MX PIM contains a PIC32MX device that supports PIC32 Instruction Trace and a trace port connector. Current PIMs and their dimensions are shown in Table 8-2 (see the Microchip website for up-to-date information). A pin connection diagram is shown in Figure 8-5.

**TABLE 8-2: PIM DIMENSIONS IN INCHES**

| PIM #    | PIM Name      | Device          | Length | Width | Height |
|----------|---------------|-----------------|--------|-------|--------|
| MA320001 | PIC32 PIM     | PIC32MX360F512L | 1.55   | 1.55  | 0.9    |
| MA320002 | PIC32 USB PIM | PIC32MX460F512L | 1.55   | 1.55  | 0.9    |

**FIGURE 8-5: PIC32MX PIM PIN CONNECTION DIAGRAM**



# MPLAB® REAL ICE™ In-Circuit Emulator User's Guide

---

NOTES:





# MPLAB® REAL ICE™ IN-CIRCUIT EMULATOR USER'S GUIDE

---

---

## Part 4 – Troubleshooting

---

---

|  |    |
|--|----|
| Chapter 9. Troubleshooting First Steps .....       | 91 |
| Chapter 10. Frequently Asked Questions (FAQ) ..... | 93 |
| Chapter 11. Error Messages .....                   | 99 |

# MPLAB® REAL ICE™ In-Circuit Emulator User's Guide

---

NOTES:

---

---

## Chapter 9. Troubleshooting First Steps

---

---

### 9.1 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

### 9.2 THE 5 QUESTIONS TO ANSWER FIRST

1. What device are you working with? Often an upgrade to a newer version of MPLAB IDE is required to support newer devices. That is, yellow light = untested support.
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 2. "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 can have communications errors. If longer cables are required, you should consider using high-speed communications. See **Section 2.3.2 "High-Speed Communication"**.

### 9.3 TOP REASONS WHY YOU CAN'T DEBUG

1. The oscillator is not working. Check your Configuration bits setting for the oscillator.
2. The target board is not powered. Check the power cable connection.
3. The emulator has somehow become physically disconnected from the PC and/or the target board. Check the communications cables' connections.
4. The device is code-protected. Check your Configuration bits setting for code protection.
5. You are trying to rebuild the project while in Release mode. Select **Debug** in the Build Configuration drop-down list on the project toolbar, then rebuild the project.
6. The emulator is selected as a programmer, and not as a debugger, in MPLAB IDE.
7. Emulator to PC communications has somehow been interrupted. Reconnect to the emulator in MPLAB IDE.
8. 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.

9. 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.
10. Brown-out Detect voltage is greater than the operating voltage VDD. This means the device is in Reset and cannot be debugged.
11. The emulator cannot always perform the action requested. For example, the emulator cannot set a breakpoint if the target application is currently running.

## 9.4 OTHER THINGS TO CONSIDER

1. It is possible the error was a one-time glitch. Try the operation again.
2. There may be a problem programming in general. As a test, switch to programmer mode 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. Microchip Technology Inc. offers myriad demonstration boards to support most of its microcontrollers. Consider using one of these applications, which are known to work, to verify correct MPLAB REAL ICE in-circuit emulator functionality. Or, use the Loop-Back Test board to verify the emulator itself (**Section 13.9 “Loop-Back Test Board”**.)
5. Review emulator debug operation to ensure proper application setup (**Chapter 2. “Operation”**.)
6. If the problem persists contact Microchip.

---

---

## **Chapter 10. Frequently Asked Questions (FAQ)**

---

---

### **10.1 INTRODUCTION**

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

- How The Emulator Works
- How Trace Works – 8 and 16 Bit Devices
- General Issues

### **10.2 HOW THE EMULATOR WORKS**

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

MPLAB REAL ICE in-circuit emulator can communicate with any silicon via the ICSP interface. It uses the debug executive located in test memory.

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

The debug executive doesn't run while in Run mode, so there is no throughput reduction when running your code, i.e., the emulator doesn't 'steal' any cycles from the target device. However, when you are doing Native trace, each macro inserted takes about 200 instructions. Therefore, this will affect timing.

For more information, see **Section 7.3.9 "Resource Usage Examples"**.

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

Please refer to **Section 2.2 "Tool Comparisons"**.

- How does MPLAB 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.

- On traditional emulators, the data must come out on the bus in order to perform a complex trigger on that data. Is this also required on the MPLAB REAL ICE in-circuit emulator? For example, could I halt based on a flag going high?

Traditional emulators use a special emulator chip (-ME) for monitoring. There is no -ME with the MPLAB REAL ICE in-circuit emulator so there are no busses to monitor externally. With the MPLAB REAL ICE in-circuit emulator, rather than using external breakpoints, the built-in breakpoint circuitry of the debug engine is used; the busses and breakpoint logic are monitored inside the part.

- Does the MPLAB REAL ICE in-circuit emulator have complex breakpoints?  
Yes. You can break based on a value in a data memory location. You can also do sequenced breakpoints, where several events are happening before it breaks, but you can only do 2 sequences instead of 4, as you can in the MPLAB ICE 2000. You can also do an AND condition and PASS counts. See **Section 12.3.1 “Breakpoints Dialog”** for more information.
- One of the probe pins is labeled 5V. How much drive capability does this probe have?  
This is a monitoring function (allows you to see what VDD is actually being applied and used on the driver buffers). The MPLAB REAL ICE in-circuit emulator cannot provide power to the target.
- Are any of the driver boards optoisolated or electrically isolated?  
They are DC optoisolated, but not AC optoisolated. To apply high voltage (120V) to the current system, see **Section 13.8 “MPLAB REAL ICE Isolator unit”**.
- What limitations are there with the 5 or 6 pins only?  
The limitations are with the cable used. The standard ICSP RJ-11 cable does not allow for clock speeds greater than about 15 Mb/sec. dsPIC33F DSCs running at full speed are greater than the 15 Mb/sec limit. For these high-speed applications, the Performance Pak (high-speed) cable interface is required.
- Will this slow down the running of the program?  
There is no cycle stealing with the MPLAB REAL ICE in-circuit emulator. The output of data is performed by the state machine in the silicon.
- How do I connect CLK and DAT when using high-speed communications?  
These connections are optional and used for SPI trace. For more information, see **Section 2.5.2 “SPI Trace Connections (High-Speed Communication Only)”**.
- 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 has a limitation on how fast data can reliably be transmitted when using trace. 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.
- To debug a dsPIC® DSC running at 30 MIPS, is the high-speed board necessary to do even basic debugging?  
Basic debugging at any device frequency can be accomplished with either standard or high-speed (Performance Pak) communications.
- If the high speed board is necessary for a dsPIC DSC to run at 30 MIPS, can this be done using the high speed to standard converter board on the target side?  
It is recommended that at high device operational frequencies, the slower cable not be used. This introduces signal integrity issues, due to the lower quality of cable transmission, when using the RJ-11 converter board.
- If the high-speed receiver board is used, do pins 7-8 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.
- What is the function of pin 6, the auxiliary pin?  
There is no function on pin 6. It is a legacy connection, compatible with the typical ICSP 6-pin header definition.

## 10.3 HOW TRACE WORKS – 8 AND 16 BIT DEVICES

- What's in the silicon that allows it to trace with the MPLAB REAL ICE in-circuit emulator?  
Tracing over the two-wire (ICSP) interface is supported with silicon that contains the Version 2 PIC18F and dsPIC33F/PIC24X in-circuit debugger modules.
- When using trace, is this connection electrically isolated in any way, i.e., do the triggers have any isolation?  
They are buffered and DC adjusted to whatever VDD level you are running. The buffers tristate when off. This minimal isolation makes the system fast and opens the door to adapt to new and faster technologies. However, you may implement more RS-232 isolation (4-6 lines) if desired, but this may impact your speed.
- Can we do trace by using the 5 or 6 ICSP pins only?  
Tracing is possible using the standard ICSP interface.
- When would SPI trace be used? What extra advantage does this have?  
The SPI trace is intended to be used for devices that do not have the advanced debug engine for tracing. These typically would be some PIC18F and all PIC16F MCU devices.
- 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 2.5.2 “SPI Trace Connections (High-Speed Communication Only)”** for more information.
- For SPI trace, which two pins are used?  
SDO (Serial Data Output) → DAT (pin 7)  
SCK (Serial Clock Output) → CLK (pin 8)
- What are the correct port settings to use SPI trace, i.e., mode, sync/async, etc.?  
The setup is taken care of by MPLAB IDE, so you will not need to be concerned about the code required for setting this. Trace will support 64 trace points and 128 log points.  
SPI – Comm Protocol MODE1, clock high, sampled falling edge.
- What is the correct connection for using I/O Port (parallel port) trace?  
The connection varies depending on the PORT used. There are port assignments in MPLAB IDE that are displayed when the PORT is selected in the property sheet. See **Section 2.5.3 “I/O Port Trace Connections”** for more information.
- Can we use any port?  
The port must be available on the device and not multiplexed with the currently used PGC and PGM pins.
- Of the 7 data and one clock, which one is the clock?  
There are 7 bits of data to set up to 128 trace points. The clock is the MSB of the port.
- Are these I/O ports used for trace available as general I/O during debugging?  
For dsPIC30F/33F and PIC24F/H devices, you may write to the opposing 8-bit part of the port provided byte write operations are used. The following example will only write to the high side of the port.  

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

  
For PIC18 devices, once the ports are defined to be used for trace, you **should not** access them in your code.

## 10.4 GENERAL ISSUES

- I cannot get trace to work. What's wrong?

Things to consider:

- Certain tool versions are required to use trace. Please refer to either **Chapter 7. "Debug for 8- and 16-Bit Devices"** or **Chapter 8. "Debug for 32-Bit Devices"**.
  - For dsPIC30F/33F and PIC24F/H devices, only C code can be used with trace, not assembly.
  - In-line assembly code (assembly code within C code) cannot be traced.
  - Native trace and real-time data capture triggers cannot be used together.
  - For Port I/O Trace, all 8 pins must be dedicated to trace (i.e., not multiplexed with the currently used PGC and PGM pins.)
  - For Port I/O Trace, 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 watch window. The value written to LAT should appear on the port pins.
- 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.

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

For dsPIC30F/33F and PIC24F/H 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).

- When using a 16-bit device, unexpected reset occurred. How do I determine what caused it?

Some things to consider:

- To determine a reset source, check the RCON register.
- Handle traps/interrupts in an interrupt service routine (ISR). You should include trap.c style code, i.e.,

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



# Frequently Asked Questions (FAQ)

---

```
void __attribute__((__interrupt__))
_AltOscillatorFail(void)
{
    INTCON1bits.OSCFAIL = 0;
    while (1);
}
:
```

- Use ASSERTs.

- I have finished debugging my code. Now I've programmed my part, but it won't run. What's wrong?

Some things to consider are:

- Have you selected the emulator as a programmer and then tried to program a header board? A header board contains an -ICE/-ICD version of the device and may not function like the actual device. Only program regular devices with the emulator as a programmer. Regular devices include devices that have on-board ICE/ICD circuitry, but are not the special -ICE/-ICD devices found on header boards.

- Have you selected the emulator as a debugger and then tried to program a regular device? Programming a device when the emulator is a debugger will program a debug executive into program memory and set up other device features for debug (see **Section 2.7.1 "Sequence of Operations Leading to Debugging"**.) To program final (release) code, select the emulator as a programmer.

- Have you selected "Release" from the Build Configuration drop-down list or Project menu? You must do this for final (release) code. Rebuild your project, reprogram the device, and try to run your code again.

- I didn't set a software 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.

- I clicked the Cancel button when asked to download the latest firmware, but now I want to download the firmware. How do I do this?

You can download it manually. Select *Debugger>Settings*, **Configuration** tab, and click **Manual Download**. Select the highest number .jam file and click **Open**.

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

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 mins). Please be patient. You will see:

- the orange busy light turn on for about 25 seconds
- the light turn red for about another 25 seconds
- the light turn orange again for about another 25 seconds

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

- I don't see my problem here. Now what?

Try the following resources:

- **Section 2.9 "Resources Used by the Emulator"**
- **Section 11.2 "Specific Error Messages"**
- **Section 11.3 "General Corrective Actions"**

# MPLAB® REAL ICE™ In-Circuit Emulator User's Guide

---

NOTES:

---

---

## Chapter 11. Error Messages

---

---

### 11.1 INTRODUCTION

The MPLAB REAL ICE in-circuit emulator produces many different error messages; some are specific and others can be resolved with general corrective actions.

- Specific Error Messages
- General Corrective Actions

### 11.2 SPECIFIC ERROR MESSAGES

MPLAB REAL ICE in-circuit emulator error messages are listed below in numeric order.

**Note:** Numbers may not yet appear in displayed messages. Use the Search tab on the Help viewer to find your message and highlight it below.

Text in error messages listed below of the form %x (a variable) will display as text relevant to your particular situation in the actual error message.

**RIErr0001: Failed while writing to program memory.**

**RIErr0002: Failed while writing to EEPROM.**

**RIErr0003: Failed while writing to configuration memory.**

See **Section 11.3.1 “Read/Write Error Actions”**.

**RIErr0005: REAL ICE is currently busy and cannot be unloaded at this time.**

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 IDE.

**RIErr0006: Failed while writing to user ID memory.**

**RIErr0007: Failed while reading program memory.**

**RIErr0008: Failed while reading EEPROM.**

**RIErr0009: Failed while reading configuration memory.**

**RIErr0010: Failed while reading user ID memory.**

See **Section 11.3.1 “Read/Write Error Actions”**.

**RIErr0011: Bulk erase failed.**

See **Section 11.3.1 “Read/Write Error Actions”**.

If these do not work, try another device.

**RIErr0012: Download debug exec failed**

If you receive this error while attempting to program from the Debugger menu:

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

If this does not work, see **Section 11.3.4 “Corrupted Installation Actions”**.

**RIErr0013: NMMR register write failed.****RIErr0014: File register write failed.**

See **Section 11.3.2 “Emulator-to-Target Communication Error Actions”**.

**RIErr0015: Data transfer was unsuccessful. %d byte(s) expected, %d byte(s) transferred.**

See **Section 11.3.3 “Emulator-to-PC Communication Error Actions”**.

**RIErr0016: Cannot transmit. REAL ICE not found.**

The emulator is not connected to the PC.

**RIErr0017: File register read failed.****RIErr0018: NMMR register read failed.****RIErr0019: Failed while reading emulation registers.****RIErr0020: Failed while writing emulation registers.**

See **Section 11.3.2 “Emulator-to-Target Communication Error Actions”**.

**RIErr0021: Command not echoed properly. Sent %x, received %x.****RIErr0022: Failed to get REAL ICE version information.****RIErr0023: Download FPGA failed.****RIErr0024: Download RS failed.****RIErr0025: Download AP failed.**

See **Section 11.3.3 “Emulator-to-PC Communication Error Actions”**.

**RIErr0026: Download program exec failed.**

If you receive this error while attempting to program from the Debugger menu:

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

If this does not work, see **Section 11.3.4 “Corrupted Installation Actions”**.

**RIErr0027: Bulk transfer failed due to invalid checksum**

See **Section 11.3.3 “Emulator-to-PC Communication Error Actions”**.

Also, ensure that the cables used are of the correct length.

**RIErr0028: Download device database failed**

If you receive this error:

1. Try downloading again. It may be a one-time error.
2. Try manually downloading. Select *Debugger>Settings*, **Configuration** tab, and click **Manual Download**. Select the highest-number .jam file and click **Open**.

**RIErr0029: Communication failure. Unexpected command echo response %x received from REAL ICE.**

See **Section 11.3.3 “Emulator-to-PC Communication Error Actions”**.

**RIErr0030: Unable to read/find firmware File %s.**

If the Hex file exists:

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

If the Hex file does not exist:

- Reinstall MPLAB IDE.

**RIErr0031: Failed to get PC.**

**RIErr0032: Failed to set PC.**

See **Section 11.3.2 “Emulator-to-Target Communication Error Actions”**.

**RIErr0033: %d bytes expected, %d bytes received.**

See **Section 11.3.3 “Emulator-to-PC Communication Error Actions”**.

**RIErr0034: This version of MPLAB IDE does not support hardware revision %06x. Please upgrade to the latest version of MPLAB IDE before continuing.**

Find the latest MPLAB IDE at [www.microchip.com](http://www.microchip.com).

**RIErr0035: Failed to get Device ID.**

See **Section 11.3.1 “Read/Write Error Actions”**.

**RIErr0036: MPLAB IDE has lost communication with REAL ICE.**

See **Section 11.3.3 “Emulator-to-PC Communication Error Actions”**.

**RIErr0037: Timed out waiting for response from REAL ICE.**

**RIErr0038: Failed to initialize REAL ICE.**

**RIErr0039: REAL ICE self test failed.**

For this error, the emulator is not responding:

1. Unplug and plug in the emulator.
2. Reconnect to the emulator in MPLAB IDE.
3. If the problem persists contact Microchip.

**RIErr0040: The target device is not ready for debugging. Please check your Configuration bit settings and program the device before proceeding.**

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 11.3.6 “Debug Failure Actions”**.

**RIErr0041: While receiving streaming data, REAL ICE has gotten out of synch with MPLAB IDE. To correct this you must reset the target device.**

Data capture or Native trace has gotten out-of-sync with MPLAB IDE. First try to Halt, Reset and then Run again. If this does not work:

1. Unplug and plug in the emulator.
2. Reconnect to the emulator in MPLAB IDE.
3. Check that the target speed is entered on the Clock tab of the Settings dialog.
4. Run again.

**RIErr0045: You must connect to a target device to use MPLAB REAL ICE.**

No power has been found.

1. Ensure VDD and GND are connected between the emulator and target.
2. Ensure that the target is powered.
3. Ensure that the target power is sufficient to be detected by the emulator (see **Chapter 13. “Hardware Specification”**.)

**RIErr0046: An error occurred while trying to read the stopwatch count. The stopwatch count may not be accurate.**

See **Section 11.3.2 “Emulator-to-Target Communication Error Actions”**.

**RIErr0047: Bootloader download failed.**

**RIErr0048: Unable to set trace options.**

See **Section 11.3.3 “Emulator-to-PC Communication Error Actions”**.

**RIErr0049: Unable to open file for streaming data trace. Trace will be turned off.**

The trace file cannot be opened.

1. Make sure the directory is not Read Only. Right click on it and check its Properties. This file is located, by default, in:

C:\Program Files\Microchip\MPLAB IDE\REAL ICE.

2. Deselect and then reselect the emulator as the debug tool.

**RIErr0050: Unable to set probe options.**

See **Section 11.3.3 “Emulator-to-PC Communication Error Actions”**.

**RIErr0051: Unrecognized trace data format received.**

Data capture or Native trace has gotten out of sync with MPLAB IDE. First try to Halt, Reset and then Run again. If this does not work:

1. Unplug and plug in the emulator.
2. Reconnect to the emulator in MPLAB IDE.
3. Check that the target speed is entered on the Clock tab of the Settings dialog.
4. Run again.

**RIErr0052: The current REAL ICE hardware version %x, is out of date. This version of MPLAB IDE will support only version %x or higher.**

Did you click **Cancel** when asked to download the latest firmware? If so, you will need to download it now. Select *Debugger>Settings, Configuration* tab, and click **Manual Download**. Select the highest number .jam file and click **Open**.

If you cannot find any files to download or if this does not work (corrupted file), you will need to get the latest version of MPLAB IDE and install it. Find the latest MPLAB IDE at [www.microchip.com](http://www.microchip.com).

**RIErr0053: Unable to get REAL ICE protocol versions.**

See **Section 11.3.3 “Emulator-to-PC Communication Error Actions”**.

**RIErr0054: MPLAB IDE's REAL ICE protocol definitions are out of date. You must upgrade MPLAB IDE to continue.**

Find the latest MPLAB IDE at [www.microchip.com](http://www.microchip.com).

**RIErr0055: Unable to set firmware suite version.**

**RIErr0056: Unable to get voltages from REAL ICE.**

See **Section 11.3.3 “Emulator-to-PC Communication Error Actions”**.

**RIErr0057: Loopback could not be completed.**

Ensure that you are using the Standard driver board and not the High-Speed driver board for loopback. Also, see **Section 11.3.2 “Emulator-to-Target Communication Error Actions”**.

**RIErr0058: Loopback internal setup failure.**

Emulator power supply failure. Contact Microchip technical support.

**RIErr0059: Loopback clock feedback failure.**

**RIErr0060: Loopback data feedback failure.**

Trigger/port pin failure. Contact Microchip technical support.

**RIErr0061: Loopback VDD not detected. Please ensure your RJ-11 cable is connected between the loopback board and the driver board. Unplug the REAL ICE to try again.**

Try the following:

1. Connect the cable between the loopback board and the Standard driver board.
2. Unplug and then plug in the emulator.

If this does not work, try a different cable and repeat the above steps.

**RIErr0062: Loopback VPP failure.**

Emulator power supply failure. Contact Microchip technical support.

**RIErr0063: Loopback clock write failure.**

**RIErr0064: Loopback data write failure.**

**RIErr0065: Loopback clock read failure.**

**RIErr0066: Loopback data read failure.**

Clock/data not being output from the emulator. Check your connections and try again.

**RIErr0067: Failed to set/clear software breakpoint.**

Reprogram and try again.

**RIErr0068: Failed while writing to boot FLASH memory.**

**RIErr0069: Failed while reading boot FLASH memory.**

**RIErr0070: Failed while writing peripheral memory.**

**RIErr0071: Failed while reading peripheral memory.**

See **Section 11.3.1 “Read/Write Error Actions”**.

**RIErr0072: Unable to send freeze peripheral information.**

See **Section 11.3.3 “Emulator-to-PC Communication Error Actions”**.

**RIErr0073: Device is code protected.**

The device on which you are attempting to operate (read, program, blank check or verify) is code protected, i.e., the code cannot be read or modified. Check your Configuration bits setting for code protection.

To disable code protection, set or clear the appropriate Configuration bits in code or in the Configuration Bits window (*Configure>Configuration Bits*), according to the device data sheet. Then erase and reprogram the *entire* device.

**RIErr0080: Failed setting shadow bit(s).**

A file register read or write failed. See **Section 11.3.2 “Emulator-to-Target Communication Error Actions”**.

## 11.3 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

## 11.3.1 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 cables used are of the correct length.

## 11.3.2 Emulator-to-Target Communication Error Actions

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

1. Select **Reset** and then try the action again.
2. Ensure that the cable(s) used are of the correct length.

## 11.3.3 Emulator-to-PC Communication Error Actions

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

1. Unplug and then plug in the emulator.
1. Reconnect to the emulator.
2. Try the operation again. It is possible the error was a one time glitch.
3. The version of MPLAB 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 11.3.4 "Corrupted Installation Actions"**.

## 11.3.4 Corrupted Installation Actions

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

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

## 11.3.5 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.

## 11.3.6 Debug Failure Actions

The MPLAB REAL ICE in-circuit emulator was unable to perform a debugging operation. There are numerous reasons why this might occur. See **Chapter 9. "Troubleshooting First Steps"**.



## 11.3.7 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 11.3.4 “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.

# MPLAB® REAL ICE™ In-Circuit Emulator User's Guide

---

NOTES:



# MPLAB® REAL ICE™ IN-CIRCUIT EMULATOR USER'S GUIDE

---

---

## Part 5 – Reference

---

---

|   |     |
|---|-----|
| Chapter 12. Emulator Function Summary ..... | 109 |
| Chapter 13. Hardware Specification .....    | 127 |

# MPLAB® REAL ICE™ In-Circuit Emulator User's Guide

---

NOTES:



---

---

## Chapter 12. Emulator Function Summary

---

---

### 12.1 INTRODUCTION

A summary of the MPLAB REAL ICE in-circuit emulator functions on menus, in windows and on dialogs is listed here.

- Debugging Functions
- Debugging Dialogs/Windows
- Programming Functions
- Settings Dialog
- Saved Information

### 12.2 DEBUGGING FUNCTIONS

When you select the MPLAB REAL ICE in-circuit emulator from the Debugger menu, debug items will be added to the following MPLAB IDE functions:

- Debugger Menu
- Right Mouse Button Menu
- Toolbars/Status Bar

#### 12.2.1 Debugger Menu

##### Run F9

Execute program code until a breakpoint is encountered or until Halt is selected.

Execution starts at the current program counter (as displayed in the status bar). The current program counter location is also represented as a pointer in the Program Memory window. While the program is running, several other functions are disabled.

##### Animate

Animate causes the debugger to actually execute single steps while running, updating the values of the registers as it runs.

Animate runs slower than the Run function, but allows you to view changing register values in the Special Function Register window or in the Watch window.

To Halt Animate, use the menu option *Debugger>Halt*, the toolbar Halt or <F5>.

##### Halt F5

Halt (stop) the execution of program code. When you click Halt, status information is updated.

## Step Into F7

Single step through program code.

For assembly code, this command executes one instruction (single or multiple cycle instructions) and then halts. After execution of one instruction, all the windows are updated.

For C code, this command executes one line of C code, which may mean the execution of one or more assembly instruction, and then halts. After execution, all the windows are updated.

**Note:** Do not step into a `SLEEP` instruction.

## Step Over F8

Execute the instruction at the current program counter location. At a `CALL` instruction, Step Over executes the called subroutine and halts at the address following the `CALL`. If the Step Over is too long or appears to “hang”, click Halt.

## Step Out

Not available.

## Reset F6

Issue a Reset sequence to the target processor. This issues a  $\overline{\text{MCLR}}$  to reset the program counter to the Reset vector.

## Breakpoints

Open the Breakpoint dialog (see **Section 12.3.1 “Breakpoints Dialog”**). Set multiple breakpoints in this dialog.

**Note:** You may also right click or double click on a line of code to set a simple breakpoint.

## Triggers

Set up real-time data capture triggers (see **Section 12.3.7 “Triggers Dialog”**).

## Program

Download your code to the target device.

## Read

Read target memory. Information uploaded to MPLAB IDE.

## Erase Flash Device

Erase all Flash memory.

## Debug Read

dsPIC DSCs and PIC24 MCUs only.

Read without terminating the debug session, i.e., the target will not be reset in order to do the read. Debug Read depends on the speed of the target oscillator so it will be very slow at 32 kHz.

## Abort Operation

Abort any programming operation (e.g., program, read, etc.) Terminating an operation will leave the device in an unknown state.

## **Reconnect**

Attempt to re-establish communications between the PC and the MPLAB REAL ICE in-circuit emulator. The progress of this connection is shown on the REAL ICE tab of the Output dialog.

## **Settings**

Open the Programmer dialog (see **Section 12.5 “Settings Dialog”**). Set up program and firmware options.

### **12.2.2 Right Mouse Button Menu**

The following will appear on the right mouse menus in code displays, such as program memory and source code files:

#### **Log Selected Value**

Log the value of the highlighted variable in the trace window. See **Section 7.3.5 “Setting Up Trace in MPLAB IDE”**.

#### **Insert Line Trace**

Log the occurrence of the selected line in the trace window. See **Section 7.3.5 “Setting Up Trace in MPLAB IDE”**.

#### **Set/Remove Breakpoint**

Set or remove a breakpoint at the currently selected line.

#### **Enable/Disable Breakpoint**

Enable or disable a breakpoint at the currently selected line.

#### **Breakpoints**

Remove, enable or disable all breakpoints.

#### **Run To Cursor**

Run the program to the current cursor location. Formerly Run to Here.

#### **Set PC at Cursor**

Set the Program Counter (PC) to the cursor location.

### **12.2.3 Toolbars/Status Bar**

When the MPLAB REAL ICE in-circuit emulator is selected as a debugger, these toolbars are displayed in MPLAB IDE:

- Basic debug toolbar (Run, Halt, Animate, Step Into, Step Over, Step Out, Reset).
- Simple program toolbar (Read, Program, Erase Flash Device).

The selected debug tool (MPLAB REAL ICE), as well as other development information, is displayed in the status bar on the bottom of the MPLAB IDE desktop. Refer to the MPLAB IDE on-line help for information on the contents of the status bar.

## 12.3 DEBUGGING DIALOGS/WINDOWS

Open the following debug dialogs and windows using the menu items mentioned in **Section 12.2 “Debugging Functions”**.

- Breakpoints Dialog
  - Set Breakpoint Dialog
  - Stopwatch Dialog
  - Event Breakpoints Dialog
  - Sequenced Breakpoints Dialog
  - AND Dialog
- Triggers Dialog
  - Add External Trigger Dialog
- Watch Window - Data Capture/Runtime Watch
  - Data Capture Properties Dialog
- Trace Window
- Build Options Dialog, Trace Tab (Device Dependent)
- Output Window, REAL ICE Tab

### 12.3.1 Breakpoints Dialog

To set up breakpoints, select *Debugger>Breakpoints*.

Set up different types of breakpoints in this dialog. Click on **Add Breakpoint** to add breakpoints to the dialog window. Depending on your selected device, there may be other buttons for more advanced breakpoint options.

#### 12.3.1.1 BREAKPOINT DIALOG WINDOW

Information about each breakpoint is visible in this window.

**TABLE 12-1: BREAKPOINT DIALOG WINDOW**

| Control         | Function   |
|-----------------|--|
| Breakpoint Type | Type of breakpoint – program or data             |
| Address         | Hex address of breakpoint location               |
| File Line #     | File name and line number of breakpoint location |
| Enabled         | Check to enable a breakpoint                     |

Once a breakpoint has been added to the window, you may right click on it to open a menu of options:

- Delete – delete selected breakpoint
- Edit/View – open the Set Breakpoint Dialog
- Delete All – delete all listed breakpoints
- Disable All – disable all listed breakpoints



# Emulator Function Summary

## 12.3.1.2 BREAKPOINT DIALOG BUTTONS

Use the buttons to add a breakpoint and set up additional break conditions. Also, a stopwatch is available for use with breakpoints and triggers.

**Note:** Buttons displayed will depend on the selected device.

**TABLE 12-2: BREAKPOINT DIALOG BUTTONS**

| Control               | Function                           | Related Dialog                                |
|-----------------------|------------------------------------|---|
| Add Breakpoint        | Add a breakpoint                   | Section 12.3.2 “Set Breakpoint Dialog”        |
| Stopwatch             | Set up the stopwatch               | Section 12.3.3 “Stopwatch Dialog”             |
| Event Breakpoints     | Set up break on an event           | Section 12.3.4 “Event Breakpoints Dialog”     |
| Sequenced Breakpoints | Set up a sequence until break      | Section 12.3.5 “Sequenced Breakpoints Dialog” |
| ANDED Breakpoints     | Set up ANDED condition until break | Section 12.3.6 “AND Dialog”                   |

### 12.3.2 Set Breakpoint Dialog

Click **Add Breakpoint** in the Breakpoints Dialog to display this dialog.

Select a breakpoint for the Breakpoints dialog here.

#### 12.3.2.1 PROGRAM MEMORY TAB

Set up a program memory breakpoint here.

**TABLE 12-3: PROGRAM MEMORY BREAKPOINT**

| Control         | Function  |
|-----------------|---|
| Address         | Location of breakpoint in hex   |
| Breakpoint Type | The type of program memory breakpoint. See the device data sheet for more information on table reads/writes.<br><i>Program Memory Execution</i> – break on execution of above address<br><i>TBLRD Program Memory</i> – break on table read of above address<br><i>TBLWT Program Memory</i> – break on table write to above address  |
| Pass Count      | Break on pass count condition.<br><i>Always break</i> – always break as specified in “Breakpoint type”<br><i>Break occurs Count instructions after Event</i> – wait Count (0-255) instructions before breaking after event specified in “Breakpoint type”<br><i>Event must occur Count times</i> – break only after event specified in “Breakpoint type” occurs Count (0-255) times |

## 12.3.2.2 DATA MEMORY TAB

Set up a data memory breakpoint here.

**TABLE 12-4: DATA MEMORY BREAKPOINT**

| Control         | Function  |
|-----------------|---|
| Address         | Location of breakpoint in hex   |
| Breakpoint Type | The type of data memory breakpoint. See the device data sheet for more information on X Bus reads/writes.<br><i>X Bus Read</i> – break on an X bus read of above address<br><i>X Bus Read Specific Byte</i> – break on an X bus read of above address for the specific byte value in “Specific Value”<br><i>X Bus Read Specific Word</i> – break on an X bus read of above address for the specific word value in “Specific Value”<br><i>X Bus Write</i> – break on an X bus write of above address<br><i>X Bus Write Specific Byte</i> – break on an X bus write of above address for the specific byte value in “Specific Value”<br><i>X Bus Write Specific Word</i> – break on an X bus write of above address for the specific word value in “Specific Value” |
| Pass Count      | Break on pass count condition.<br><i>Always break</i> – always break as specified in “Breakpoint type”<br><i>Break occurs Count instructions after Event</i> – wait Count (0-255) instructions before breaking after event specified in “Breakpoint type”<br><i>Event must occur Count times</i> – break only after event specified in “Breakpoint type” occurs Count (0-255) times   |

## 12.3.3 Stopwatch Dialog

Click **Stopwatch** in the Breakpoints Dialog to display this dialog.

The stopwatch allows timing from one breakpoint/trigger condition to the next. The stopwatch value is in decimal.

**TABLE 12-5: STOPWATCH SETUP**

| Control                | Function  |
|------------------------|---|
| Start Condition        | Click <b>Select Start Condition</b> to choose an available breakpoint or trigger condition to start the stopwatch. Available breakpoints/triggers are those previously added to the breakpoint dialog.<br>Click <b>None</b> to clear the start condition.<br>To halt the program run on this condition, check the check box next to “Start condition will cause the target device to halt”. |
| Stop Condition         | Click <b>Select Stop Condition</b> to choose an available breakpoint or trigger condition to stop the stopwatch. Available breakpoints/triggers are those previously added to the breakpoint dialog.<br>Click <b>None</b> to clear the stop condition.<br>To halt the program run on this condition, check the check box next to “Stop condition will cause the target device to halt”.     |
| Reset stopwatch on run | Reset the stopwatch values to zero every time the program is run.   |

## 12.3.4 Event Breakpoints Dialog

Click **Event Breakpoints** in the Breakpoints Dialog to display this dialog.

Select a condition where the program will always break:

- Break on Watchdog Timer – Break every time the watchdog timer times out. Make sure the Watchdog Timer is enabled in the Configuration bits.
- Break on `SLEEP` instruction – Break when a `SLEEP` instruction is encountered in the program.

## 12.3.5 Sequenced Breakpoints Dialog

Click **Sequenced Breakpoints** in the Breakpoints Dialog to display this dialog.

Set up a sequential occurrence of breakpoints. Sequence execution of breakpoints is bottom-up; the last breakpoint in the sequence occurs first.

To add a breakpoint to a sequence:

- Select a breakpoint from the list of “Available Breakpoints”. Available breakpoints/triggers are those previously added to the breakpoint dialog.
- Select a sequence for the list of “Sequences”.
- Click **Add**.

To change the order of breakpoints in a sequence, drag-and-drop the breakpoint in the “Sequences list”.

To remove a breakpoint from a sequence:

- Select the breakpoint in the “Sequences” list.
- Click **Remove**.

## 12.3.6 AND Dialog

Click **ANDed Breakpoints** in the Breakpoints Dialog to display this dialog.

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.

To add a breakpoint to the AND condition:

- Select a breakpoint from the list of “Available Breakpoints”. Available breakpoints/triggers are those previously added to the breakpoint dialog.
- Click **Add**.

To remove a breakpoint from a sequence:

- Select the breakpoint in the “ANDed Breakpoints” list.
- Click **Remove**.

## 12.3.7 Triggers Dialog

Where is Data Capture?

To set up triggers, select *Debugger>Triggers*.

Click on **Add External Triggers** to open the Add External Trigger Dialog to set up hardware triggers.

When complete, trigger information will appear in this dialog. Each trigger may be enabled/disabled individually.

## 12.3.8 Add External Trigger Dialog

Click **Add External Triggers** in the Triggers Dialog to display this dialog.

Use this dialog to set up external triggering via the logic probe port. Depending on the processor speed, the amount of skid after the trigger can be significant.

- Select a pin to use as the trigger from the pull-down menu. Once selected, the corresponding pin is highlighted on a figure of the logic probe port as viewed from the front of the emulator.
- Next select the type of trigger.

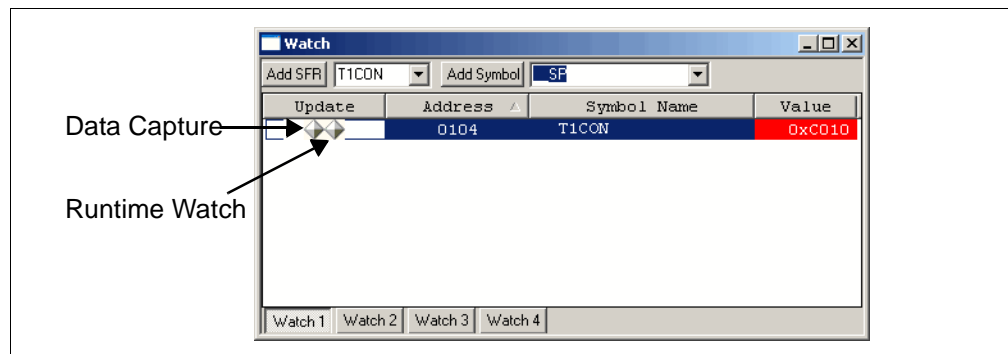
**TABLE 12-6: TRIGGER SETUP**

| Trigger Type | Trigger                             | Action                   |
|--------------|-------------------------------------|--------------------------|
| Input        | Positive or negative edge triggered | Halt or Reset on trigger |
| Output       | High-to-low or Low-to-high pulse    | Assert on Halt or Run    |

## 12.3.9 Watch Window - Data Capture/Runtime Watch

The MPLAB REAL ICE in-circuit emulator tool adds an “Update” column to the Watch window for use in setting up data capture and/or a runtime watch for a data address (Figure 12-1). For information on the Watch window in general, see MPLAB IDE help.

**FIGURE 12-1: WATCH WINDOW WITH UPDATE**



Click on a diamond in the “Update” column to alternately set/remove data capture or a runtime watch.

### 12.3.9.1 MENU OPTIONS

Right click in the “Update” column to see data capture and runtime watch options, or right click on a diamond to pop up a menu with options for only data capture or a runtime watch.

| Update Menu  |  |
|--|--|
| Data Capture Menu  | Runtime Watch Menu   |
| <b>Address, Symbol</b> – Address and Symbol name of item in the Watch window to which data capture will apply. | <b>Address, Symbol</b> – Address and Symbol name of item in the Watch window to which the runtime watch will apply |
| <b>Set Data Capture</b> – Enable data capture function.  | <b>Set Runtime Watch</b> – Enable runtime watch function.  |
| <b>Remove Data Capture</b> – Disable data capture function.  | <b>Remove Runtime Watch</b> – Disable runtime watch function.  |
| <b>Data Capture Properties</b> – Open the Data Capture Properties Dialog.                                      | N/A  |
| <b>Remove All Updates</b> – Disable all updates (data captures and runtime watches).                           | <b>Remove All Updates</b> – Disable all updates (data captures and runtime watches).                               |

## 12.3.9.2 SYMBOL SUPPORT

To see if a symbol supports data capture or runtime watch, add the symbol to the Watch window and see if the above diamonds appear in the Update column next to the symbol. In general, supported symbol types include SFRs, variables, and structure and array members (for some devices.)

**Note:** The emulator simply displays data address (SFR) information that is sent over the device ICE bus. Not all SFRs are on the device ICE bus, notably SFRs in the core.

## 12.3.9.3 DEVICE-SPECIFIC SUPPORT

For PIC18FXXJ, PIC24H, dsPIC30F SMPS and dsPIC33F devices, data captures and runtime watches use the same resource. For more information, see **Chapter 7. “Debug for 8- and 16-Bit Devices”**

For PIC32MX devices, data captures and runtime watches use separate resources. For more information, see **Chapter 8. “Debug for 32-Bit Devices”**.

**Note:** For PIC32MX devices, a maximum of 62 variables may be used for runtime watch.

## 12.3.10 Data Capture Properties Dialog

Select “Data Capture Properties” from the “Update” right click menu in the Watch window to see this dialog. The data capture properties dialog has the following options:

- Address or symbol to capture - The data address (in hex) for capture as specified in the Watch window.
- When to capture entry - Specify if the capture happens on read or write from the X bus.

Data capture can only be performed on register-sized variables.

- For PIC18 MCUs, only byte-sized variables can be captured.
- For dsPIC DSC/PIC24 devices, only 16-bit variables can be captured, such as shorts and ints, but not floats and longs.

Data captures use the same resources as breakpoints, so the maximum available number of breakpoints applies to the maximum number of available data captures and breakpoints. As an example, if 4 breakpoints are available, and two breakpoints are set, up to two data captures may be set also.

If Native trace is used, then data captures cannot be used because of hardware constraints. However, breakpoints are still available.

## 12.3.11 Trace Window

View trace information in this window ([View>Trace](#)). See **Section 7.3.3.1 “Native Trace”** for more on tracing.

Traced code or variable values are visible in the top half of the window.

- Line – trace line number (not the line number in corresponding source code)
- Address – address of the program counter when trace/log was accessed
- Value – value of traced item, if applicable

For each trace item selected/highlighted, the corresponding code line will be shown in the bottom half of the window, if “Show Source” has been selected.

## Trace Window Menu

Below are the menu items in the Trace window right click menu.

### Close

Close this window.

### Find

Opens the Find dialog. In the Find What field, enter a string of text you want to find, or select text from the drop-down list. You can also select text in the edit window or place the cursor over a word you want to search for, before you open the Find dialog.

In the Find dialog you may select any of the available options and the direction you want to search. Up searches backward from the insertion point, Down searches forward.

### Find Next

Find the next instance of Find text.

<F3> repeats the last Find.

<Shift> + <F3> reverses the direction of the last Find.

### Go To

Jump to the specified item:

- Trigger – Jump to the location of the trigger.
- Top – Jump to the top of the window.
- Bottom – Jump to the bottom of the window.
- Go To Trace Line – Go to the trace line specified in the dialog.
- Go To Source Line – Open a File window and go to the source code line corresponding to the selected trace line.

### Show Source

Show/hide the source code listing on the bottom of the window. The window bar dividing the trace and source code may be dragged to resize each portion.

### Reload

Reload the trace memory window with the contents of the trace buffer.

### Output to File

Export the contents of the trace memory window to a file. Uses a Save As dialog, with the addition of cycle and tab information. Enter a “Start” and “End” cycle to write to the file. Also specify if the text is to be tab delimited.

### Print

Print the contents of the trace memory window.

### Refresh

Refresh the viewable contents of the window.

### Properties

Set up window properties.

# Emulator Function Summary

## 12.3.12 Build Options Dialog, Trace Tab (Device Dependent)

To enable/disable trace and select the type of trace, select *Project>Build Options>Project* and then click on the **Trace** tab.

If this tab is visible, then your selected device supports this type of trace. Use this tab to enable and set up emulator trace. If this tab is not visible, see **Section 12.5.3 “Settings Dialog, Trace Tab”** to enable trace.

Make sure you are using a compiler version that supports this emulator function (see **Section 7.3.1 “Requirements for Trace”**).

**TABLE 12-7: TRACE SETUP**

| Control  | Function   | Related Section                                     |
|--|--|---|
| Enable Trace   | Check checkbox to enable trace mechanism.  | <b>Chapter 7. “Debug for 8- and 16-Bit Devices”</b> |
| Disable Trace Macros   | Check checkbox to disable inserted trace macros, i.e., treat macros as comments.                                 | <b>Section 7.3.8 “Disabling Trace”</b>              |
| <b>Communication Medium</b>  | Communication method for trace   |   |
| Native*  | Select to use the device’s native resources for trace.   | <b>Section 7.3.3.1 “Native Trace”</b>               |
| I/O Port   | Select to use a device I/O port for trace, and then select the port you will be using from the pull-down list.   | <b>Section 7.3.3.2 “I/O Port Trace”</b>             |
| SPI*   | Select to use device SPI pins for trace, and then select the SPI port you will be using from the pull-down list. | <b>Section 7.3.3.3 “SPI Trace”</b>                  |
| <b>Device Resources Used</b>   | For the communications medium chosen, a description of device resources used is displayed.                       | <b>Section 7.3.3 “Types of Trace”</b>               |
| * Set your clock speed for these types of trace ( <i>Debugger&gt;Settings, Clock</i> tab.) |  |   |

## 12.3.13 Output Window, REAL ICE Tab

Selecting *View>Output* opens the Output window. This window contains tabbed information about program output. See MPLAB IDE documentation for more on this window.

For the emulator, operational data is displayed on the **REAL ICE** tab of the Output window. This data includes, but is not limited to:

- Connection data (when the emulator is selected as a debugger or programmer)
- Stopwatch data
- Breakpoint operational data
- Trace operational data
- Warnings and errors

When the emulator is selected as a debugger (*Debugger>Select Tool>REAL ICE*), the following connection information is displayed on this tab.

## 12.3.13.1 CONNECTION TO THE EMULATOR

MPLAB IDE will first attempt to find the emulator (MPLAB REAL ICE detected) and then communicate with it (connecting to MPLAB REAL ICE in-circuit emulator). Once communication is established, MPLAB IDE will read the current firmware (operating system) loaded into the emulator and determine if (1) there is a newer version of firmware that needs to be loaded or (2) a different firmware needs to be loaded to support the selected device. If either are true, a dialog box will appear alerting you to the need to load new firmware. It is recommended that you allow the load to continue.

The **REAL ICE** tab will display the process of downloading firmware, checking the install, and verifying the new firmware. Once the download is complete, "MPLAB REAL ICE Connected" will be displayed.

## 12.3.13.2 CONNECTION TO THE TARGET

MPLAB IDE will next attempt to connect to the target (Target Detected). If a target is found, the device ID of the target device will be displayed, as read from the device in hexadecimal format.

**Note:** The MPLAB REAL ICE in-circuit emulator works with devices that have built-in debug circuitry. Therefore, the emulator needs a target device to function as a debugger.

## 12.4 PROGRAMMING FUNCTIONS

When you select the MPLAB REAL ICE in-circuit emulator from the Programmer menu, program items will be added to the following MPLAB IDE functions:

- Programmer Menu
- Toolbars/Status Bar

### 12.4.1 Programmer Menu

#### **Program**

Program specified memory areas: program memory, Configuration bits, ID locations and/or EEPROM data. See the Settings dialog for programming options.

#### **Verify**

Verify programming of specified memory areas: program memory, Configuration bits, ID locations and/or EEPROM data.

#### **Read**

Read specified memory areas: program memory, Configuration bits, ID locations and/or EEPROM data. See the Settings dialog for read options.

#### **Blank Check All**

Check to see that all device memory is erased/blank.

#### **Erase Flash Device**

Erase all Flash memory.

#### **Release from Reset**

Tristate MCLR.

#### **Hold in Reset**

Set MCLR to ground (zero.)



# Emulator Function Summary

## Settings

Open the Programmer dialog (see **Section 12.5 “Settings Dialog”**). Set up program and firmware options.

### 12.4.2 Toolbars/Status Bar

When the MPLAB REAL ICE in-circuit emulator is selected as a programmer, an emulator-specific programmer toolbar is displayed in MPLAB IDE. This toolbar contains buttons with the same functions as the emulator programmer menu.

The selected programmer (MPLAB REAL ICE in-circuit emulator), as well as other programming information, is displayed in the status bar on the bottom of the MPLAB IDE desktop. Refer to the MPLAB IDE on-line help for information on the contents of the status bar.

## 12.5 SETTINGS DIALOG

Select either *Debugger>Settings* or *Programmer>Settings* to open the Settings dialog and set up the MPLAB REAL ICE in-circuit emulator.

**Note:** Settings information may be preserved by saving the MPLAB IDE workspace.

The Settings dialog has the following tabs:

**Note:** Tabs displayed will depend on the selected device.

- Settings Dialog, Program Memory Tab
- Settings Dialog, Configuration Tab
- Settings Dialog, Trace Tab
- Settings Dialog, Freeze on Halt Tab
- Settings Dialog, Status Tab
- Settings Dialog, Clock Tab
- Settings Dialog, Secure Segment Tab
- Settings Dialog, Warnings Tab
- Settings Dialog, Runtime Watch Tab

### 12.5.1 Settings Dialog, Program Memory Tab

This tab allows you to set up debug/programming options.

**TABLE 12-8: MANUAL SELECTION OPTIONS**

|   |   |
|---|---|
| <b>Allow MPLAB REAL ICE to select memories and ranges</b> | The emulator uses your selected device and default settings to determine what to program.   |
| <b>Manually select memories and ranges</b>                | You select the type and range of memory to program (see below.)   |
| <i>Memories</i>   |   |
| Program   | Check to program Program Memory into target.  |
| Configuration   | Check to program Configuration bits into target.<br><b>Note:</b> This memory is always programmed when emulator selected as a debugger. |
| EEPROM  | Check to erase and then program EEPROM memory on target, if available. Uncheck to erase EEPROM memory on target.                        |
| ID  | Check to program ID Memory into target.   |
| Boot Flash  | If supported, check to program boot memory on target.   |

**TABLE 12-8: MANUAL SELECTION OPTIONS (CONTINUED)**

| <i>Program Memory Range (hex)</i>          |  |
|--|--|
| Start, End                                 | The starting and ending hex address range in program memory for programming, reading, or verification.<br>If you receive a programming error due to an incorrect end address, you need to perform a reconnect, correct the end address and program again.<br><b>Note:</b> The address range does not apply to the Erase function. The Erase function will erase all data on the device.  |
| <i>Program Options</i>                     |  |
| 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 not checked, the device is not erased and program code will be merged with the code already in the device.   |
| Preserve EEPROM on Program                 | Check to keep EEPROM memory on target from being overwritten on programming. Target EEPROM memory values are read into MPLAB IDE, erased from the target and then written back to the target.<br>Uncheck to use EEPROM checkbox functionality under Memories.  |
| Use high voltage on MCLR                   | <i>For PIC24FXXKAXXX devices:</i><br>Check this option to use <u>high</u> voltage to configure pin as either a normal input pin or as a MCLR reset.<br>Leave unchecked for a low voltage <u>MCLR</u> reset.<br><br><b>Notes:</b><br>1. As long as the configuration setting is "MCLR pin enabled; RA5 input pin disabled", then you can use Low Voltage Entry (uncheck the "Use high voltage on MCLR" option).<br>2. If you have the configuration setting to "RA5 input pin enabled; MCLR disabled", then you must check the "Use high voltage on MCLR" option.<br>3. If you want to program the "MCLR pin enable bit" configuration setting, you must check the "Use high voltage on MCLR" option.<br>4. If you are using a -ICE header, the setting of this checkbox does not matter. |
| <i>Preserve Program Memory Range (hex)</i> |  |
| Start, End                                 | 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 overlaid with existing MPLAB IDE memory.   |
| <i>Automatically</i>                       | Perform the selected debug options automatically.  |
| Program after successful build             | After the application code successfully builds, program this code into the device.   |
| Run after successful program               | <i>This option is available only in debug mode.</i><br>After the application code is successfully programmed into the target device, run the code.   |

# Emulator Function Summary

## 12.5.2 Settings Dialog, Configuration Tab

Configure emulator operation on this tab.

**TABLE 12-9: CONFIGURATION ITEMS**

|                               |  |
|-------------------------------|--|
| <b>Download Firmware</b>      | Set up firmware download options. If you have problems, see <b>Chapter 10. “Frequently Asked Questions (FAQ)”</b> .  |
| Auto Download Latest Firmware | Check to allow automatic download of the latest version of firmware for the target device ( <b>recommended</b> ).  |
| Manual Download               | Manually select a firmware file to download to the target device.  |
| <b>Breakpoints</b>            | Depending on your selected device, you may be able to use software breakpoints. Review the text beneath each type of breakpoint to determine which is best for your current needs.   |
| Use Hardware Breakpoints      | This is the default/classic mode for breakpoint behavior. Using hardware breakpoints means:<br>Number of breakpoints: limited<br>Breakpoints are written to debug registers<br>Time to set breakpoints: minimal<br>Skidding: yes   |
| Use Software Breakpoints      | Using software breakpoints means:<br>Number of breakpoints: unlimited<br>Breakpoints are written to program memory<br>Time to set breakpoints: oscillator speed dependent – can take minutes<br>Skidding: no<br><b>Note:</b> Using software breakpoints for debug impacts device endurance. Therefore, it is recommended that devices used in this manner not be used as production parts. |

## 12.5.3 Settings Dialog, Trace Tab

Set up trace options on this tab.

**TABLE 12-10: TRACE OPTIONS**

|                                     |  |
|-------------------------------------|--|
| <b>Trace Buffer Size</b>            |  |
| Number of Lines                     | The trace buffer for MPLAB IDE can hold up to 2GB of data. When used with the emulator, each trace line requires 12 bytes of data. Therefore, the buffer can contain a maximum of 165 million trace lines.   |
| Number of Bytes (PIC32MX Devices)   | The maximum size is 22M Bytes. The trace buffer size must be divisible by 8.<br><b>Note:</b> Due to the trace data compression scheme, one byte of trace data may contain between one to seven trace lines.  |
| <b>PIC32MX Instruction Trace</b>    |  |
| Enable Trace                        | If this item is visible, then your selected device supports this type of trace. Use this checkbox to enable emulator trace. If this item is not visible, see <b>Section 12.3.12 “Build Options Dialog, Trace Tab (Device Dependent)”</b> to enable and set up trace.   |
| Stall CPU When Trace Buffer is Full | Use this checkbox to enable this feature. Set the size of the trace buffer in the previous section of this dialog tab.   |
| Start/Stop Triggers                 | To use this feature, set and enable breakpoints in your code, and then use this section to select breakpoints to start and/or stop trace.<br><b>Note:</b> Once breakpoints have been used for trace, they will stay assigned as triggers until unselected. That is, even if “Enable Trace” is unchecked, the breakpoints will still be assigned as triggers. You must use these pull-down menus to unassign the triggers so that they are again breakpoints. |

For more information about trace, see either:

- Chapter 7. “Debug for 8- and 16-Bit Devices”
- Chapter 8. “Debug for 32-Bit Devices”

## 12.5.4 Settings Dialog, Freeze on Halt Tab

Select peripherals to freeze on halt on this tab.

### PIC12/16 MCU Devices

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

There may be device-specific limitations for peripherals that do support freeze. Click to the **Limitations** button to find these.

### PIC18 MCU Devices

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

### dsPIC30F/33F, PIC24F/H and PIC32MX Devices

For peripherals in the list “Peripherals to Freeze on Halt”, check to freeze that peripheral on a halt. Uncheck the peripheral to let it run while the program is halted. If you do not see a peripheral on the list, check “All Other Peripherals”. If this does not halt your desired peripheral, be aware that some peripherals have no freeze on halt capability and cannot be controlled by the emulator.

To select all peripherals, including “All Other Peripherals”, click **Check All**. To deselect all peripherals, including “All Other Peripherals”, click **Uncheck All**.

## 12.5.5 Settings Dialog, Status Tab

View the status of your MPLAB REAL ICE system on this tab.

**TABLE 12-11: STATUS ITEMS**

| <b>Versions</b>           |  |
|---------------------------|--|
| Firmware Suite Version    | Emulator firmware suite version. The firmware suite consists of the three items specified below.   |
| FPGA Version              | Internal FPGA chip firmware version.   |
| Algorithm Plug-in Version | Emulator algorithm plug-in version. For your selected device, an algorithm is used to support the device plugged in to the target.         |
| OS Version                | Emulator operating system version.   |
| <b>Voltages</b>           |  |
| REAL ICE V <sub>PP</sub>  | Emulator pod V <sub>PP</sub> .   |
| REAL ICE V <sub>DD</sub>  | Emulator pod V <sub>DD</sub> .   |
| Target V <sub>DD</sub>    | V <sub>dd</sub> sensed at target.  |
| Refresh Voltages          | Sensing of Status tab items occurs when the tab is activated. To see updates otherwise, click this button.                                 |
| <b>Run Loopback Test</b>  | To check emulator operation, insert the loop-back test board (see <b>Section 13.9 “Loop-Back Test Board”</b> ) and then click this button. |

# Emulator Function Summary

## 12.5.6 Settings Dialog, Clock Tab

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

**TABLE 12-12: CLOCK OPTIONS**

| Instruction Speed |   |
|-------------------|---|
| Speed value       | Enter a value for the "Speed unit" selected.<br><b>Example 1:</b> For a PIC24 MCU and a target clock oscillator at 32 MHz (HS), instruction speed = 32 MHz/2 = 16 MIPS.<br><b>Example 2:</b> 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. |
| Speed unit        | Select either:<br>KIPS – Thousands ( $10^3$ ) of instructions per second<br>MIPS – Millions ( $10^6$ ) of instructions per second   |

## 12.5.7 Settings Dialog, Secure Segment Tab

For CodeGuard™ Security devices, set up secure segment properties on this tab.

For more details on CodeGuard Security functionality, please refer to the CodeGuard Security reference manual for 16-bit devices (DS70180) and dsPIC33F/PIC24H and dsPIC30F device programming specifications found on our website.

**TABLE 12-13: SECURE SEGMENT OPTIONS**

|                       |   |
|-----------------------|---|
| Full Chip Programming | Click to select to program all program memory segments.   |
| Segment Programming   | Click to select segment programming. Select from:<br>- Boot, secure and general segments<br>- Secure and general segments<br>- General segment only |

## 12.5.8 Settings Dialog, Warnings Tab

A list of all MPLAB REAL ICE in-circuit emulator warnings are displayed on this tab.

- Check a warning to enable it. The warning will be displayed in the Output window.
- Uncheck a warning to disable it.

Warnings are not errors and will not stop your project from building. If you receive error messages, please see **Chapter 11. "Error Messages"**.

## 12.5.9 Settings Dialog, Runtime Watch Tab

For PIC32MX devices, select the update rate of watch data at runtime. To set up a runtime watch, see **Section 8.2 "Data Capture and Runtime Watches"**.

## 12.6 SAVED INFORMATION

The following information is stored in an initialization file (REALICE.INI) for retrieval after emulator selection:

- AutoSelectRange - "Allow MPLAB REAL ICE to select memories and ranges" or "Manually select memories and ranges"
- PgmAfterRange - "Program after successful build" (yes/no)
- RunAfterPgm - "Run after successful program" (yes/no)
- EraseB4Pgm - "Erase all before Program" (yes/no)
- AutoDownload - "Auto Download Latest Firmware" (yes/no)
- UseSWBPs - "Use Software Breakpoints" or "Use Hardware Breakpoints"

This file is located in the MPLAB IDE installation directory, REAL ICE subdirectory.

# MPLAB® REAL ICE™ In-Circuit Emulator User's Guide

---

NOTES:



# MPLAB® REAL ICE™ IN-CIRCUIT EMULATOR USER'S GUIDE

---

---

## Chapter 13. Hardware Specification

---

---

### 13.1 INTRODUCTION

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

### 13.2 HIGHLIGHTS

This chapter discusses:

- Declaration of Conformity
- USB Port/Power
- Emulator Pod
- Standard Communication Hardware
- High-Speed Communication Hardware
- MPLAB REAL ICE Isolator unit
- Loop-Back Test Board
- Target Board Considerations

### 13.3 DECLARATION OF CONFORMITY

We

Microchip Technology, Inc.  
2355 W. Chandler Blvd.  
Chandler, Arizona 85224-6199  
USA

hereby declare that the product:

MPLAB® REAL ICE™ In-Circuit Emulator

complies with the following standards, provided that the restrictions stated in the operating manual are observed:

Standards: EN61010-1      Laboratory Equipment

Microchip Technology, Inc.

Date: August 2006

#### Important Information Concerning the Use of the MPLAB REAL ICE In-Circuit Emulator

Due to the special nature of the MPLAB REAL ICE in-circuit emulator, the user is advised that it can generate higher than normal levels of electromagnetic radiation which can interfere with the operation of all kinds of radio and other equipment.

To comply with the European Approval Regulations therefore, the following restrictions must be observed:

1. The development system must be used only in an industrial (or comparable) area.
2. The system must not be operated within 20 meters of any equipment which may be affected by such emissions (radio receivers, TVs etc.).

## 13.4 USB PORT/POWER

The MPLAB REAL ICE in-circuit emulator is connected to the host PC via a Universal Serial Bus (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.

**PC Hibernate/Power-Down modes** – Disable the hibernate or other power saver modes on your PC to ensure proper USB communications with the emulator.

## 13.5 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.

### 13.5.1 Main Board

This component has the interface processor (dsPIC DSC), the 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

### 13.5.2 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. |



## 13.5.3 Indicator Lights (LEDs)

The indicator lights have the following significance.

| LED    | Color  | Description  |
|--------|--------|--|
| Active | Blue   | Lit when power is first applied or when target is connected. |
| Status | Green  | Lit when the emulator is operating normally – standby.       |
|        | Red    | Lit when an operation has failed.                            |
|        | Orange | Lit when the emulator is busy.                               |

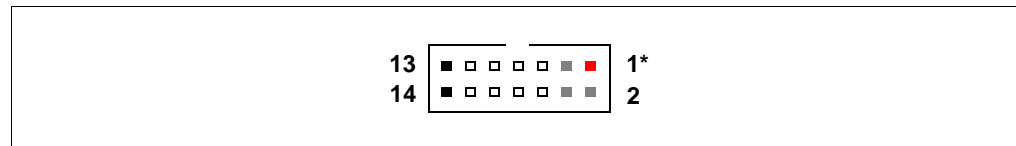
## 13.5.4 Logic Probe/External Trigger Interface

Probes 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 IDE. The external trigger is a pulse of approximately 1.5  $\mu$ s. This value is not deterministic and the external tool should be triggered on a pulse edge.

The inputs are part of a trigger bus.

**FIGURE 13-1: LOGIC PROBE PINOUT ON EMULATOR**



Logic probes may be attached to this connector to give the functionality described in Table 13-1. The probes are color coded and labeled for easy identification.

**TABLE 13-1: LOGIC PROBE PINOUT DESCRIPTION**

| Pin | I/O | Name                | Function                    | Color |
|-----|-----|---------------------|-----------------------------|-------|
| 1   | O   | VDD <sup>(1)</sup>  | 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 <sup>(2)</sup> | 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 <sup>(2)</sup> | 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.

**Note 2:** EXT0 and EXT7 are temporarily used during loop-back test. Ensure that they are not connected together.

The electrical specifications for logic probes are listed in Table 13-2.

**TABLE 13-2: 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 |

## 13.6 STANDARD COMMUNICATION HARDWARE

For standard emulator communication with a target (**Section 2.3.1 “Standard Communication”**), use the standard driver board.

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

**Note:** Older header boards 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 header boards, see the “*Header Board Specification*” (in Recommended Reading).

### 13.6.1 Standard Driver Board

The standard driver board 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.

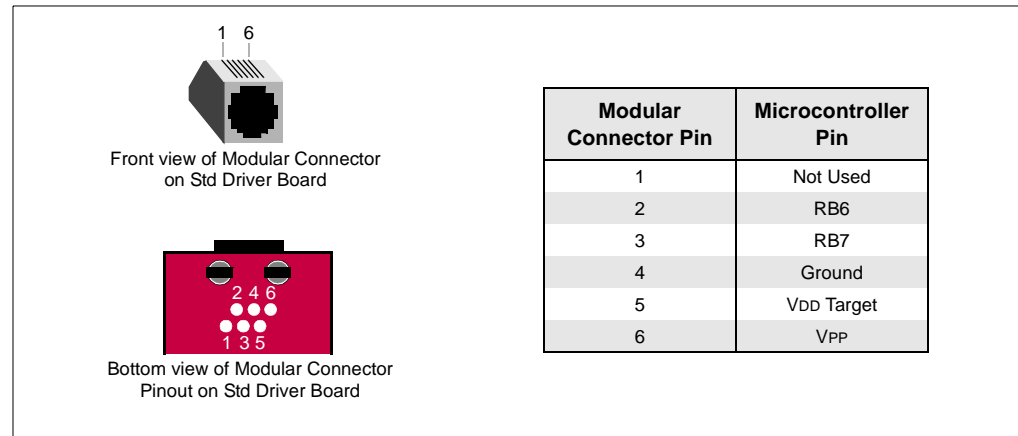
The VDD sense connection draws very little current from the target processor. The actual power comes from the MPLAB REAL ICE in-circuit emulation system as the VDD sense line is used as a reference only to track the target voltage. The VDD connection is isolated with an optical switch.

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 13-2: MODULAR CONNECTOR PINOUT OF STANDARD DRIVER BOARD**

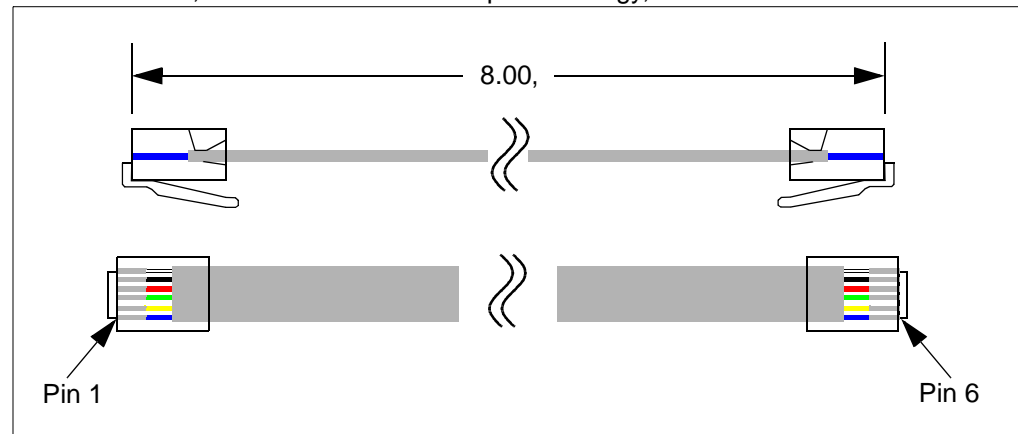


## 13.6.2 Modular Cable and Connector

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

### 13.6.2.1 MODULAR CABLE SPECIFICATION

- Manufacturer, Part Number – Microchip Technology, 07-00024



### 13.6.2.2 MODULAR PLUG SPECIFICATION

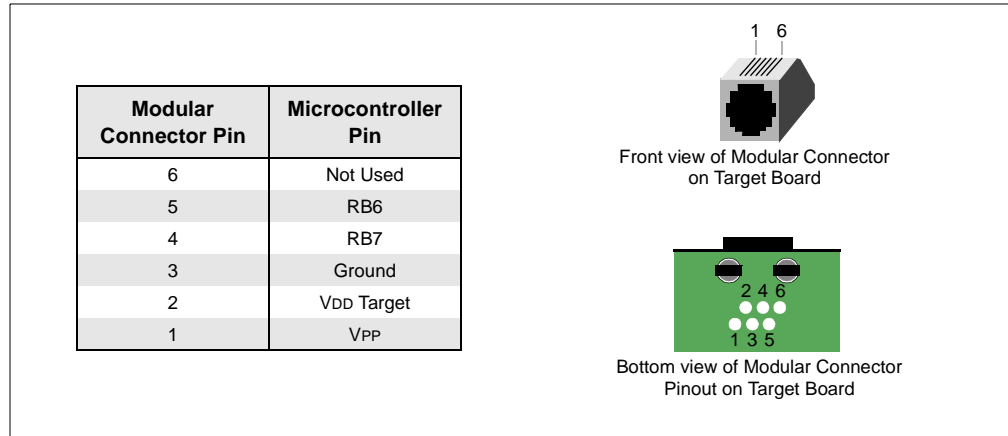
- Manufacturer, Part Number – AMP Incorporated, 5-554710-3
- Distributor, Part Number – Digikey, A9117ND

## 13.6.2.3 MODULAR CONNECTOR SPECIFICATION

- Manufacturer, Part Number – AMP Incorporated, 555165-1
- Distributor, Part Number – Digikey, 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 13-3: MODULAR CONNECTOR PINOUT OF TARGET BOARD**



## 13.7 HIGH-SPEED COMMUNICATION HARDWARE

For high-speed emulator communication with a target (**Section 2.3.2 “High-Speed Communication”**), use the Performance Pak (AC244002). The Performance Pak includes:

- a High-Speed Driver Board
- a High-Speed Receiver Board
- LVDS Cables and Target Pinout

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

**Note:** You will not need the standard adapter board for high-speed communications. Instead, you will plug the 8-pin connector end of the high-speed receiver board directly into the 8-pin connector of the header board.

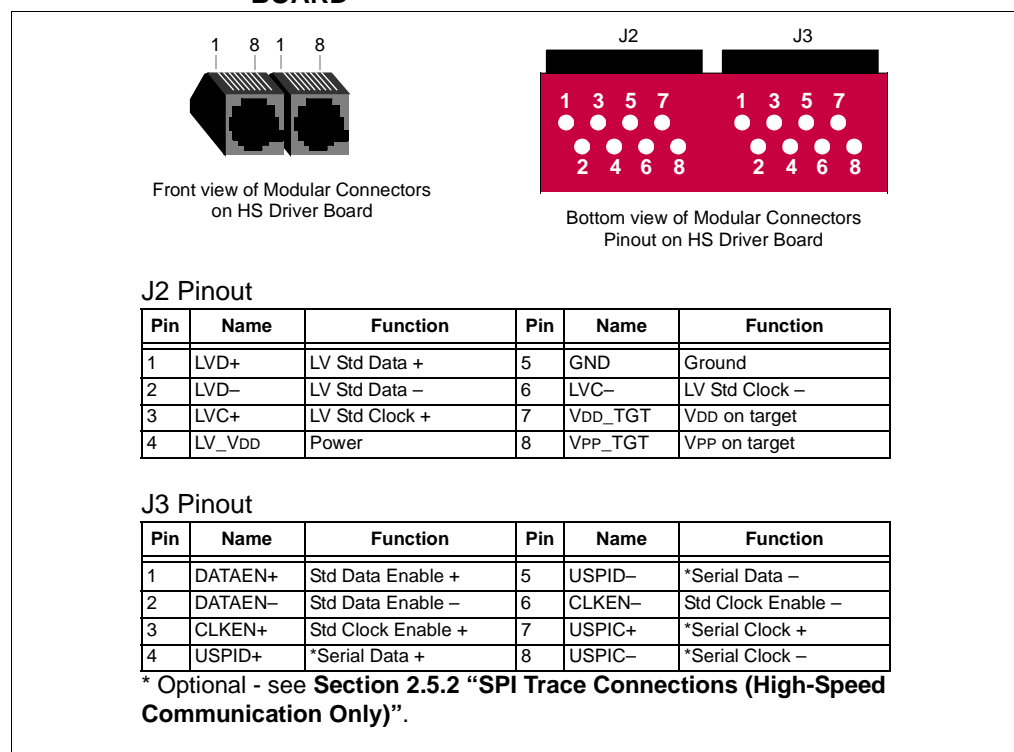
For more on available header boards, see the “*Header Board Specification*” (in Recommended Reading).

## 13.7.1 High-Speed Driver Board

The high-speed driver board consists of two separate multipoint LVDS (Low Voltage Differential Signal) transmitters and receivers for clock and data. Multipoint LVDS requires 100 ohm terminations at each driver output and receiver input, per the standard, and multipoint configurations type 2 receivers are used, as these are intended for control signals or where fail-safe provisions are needed. Even though the standard allows for any combination of drivers, receivers and/or transceivers of up to 32 on the line, only two will be used. The driver board has a port expansion which is controlled by an I<sup>2</sup>C™ interface for sending/receiving status information to/from the emulator. The high-speed driver board assembly is inserted into the emulator pod via the card guide.

**Note:** Data rates up to 40 MIPS are possible.

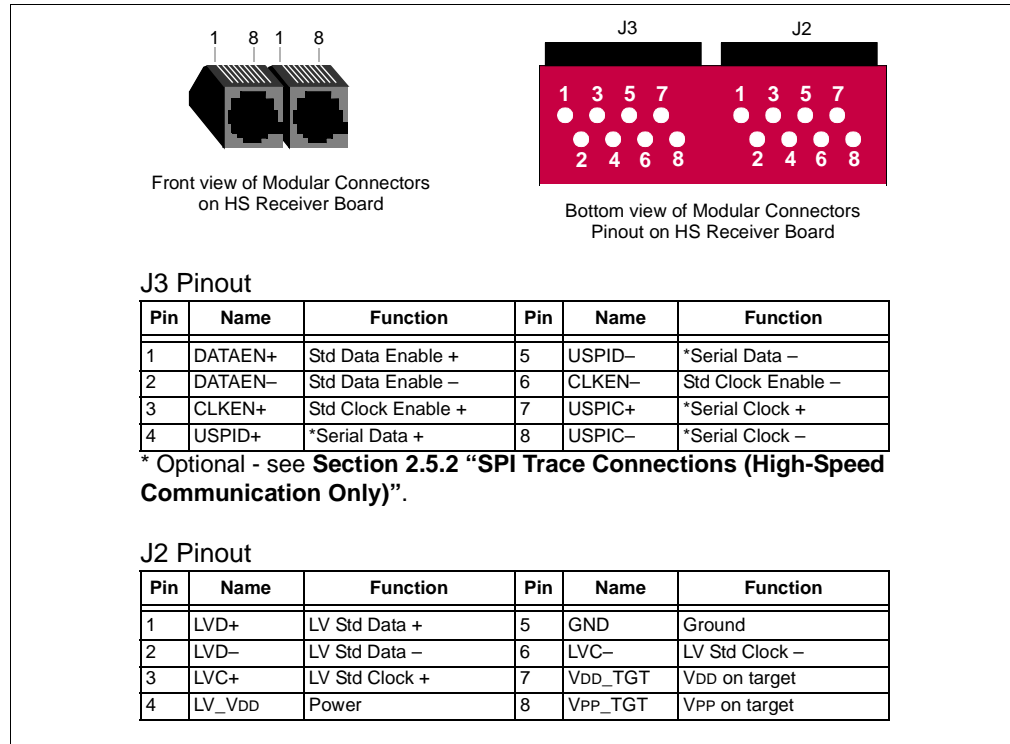
**FIGURE 13-4: MODULAR CONNECTORS PINOUT OF HIGH-SPEED DRIVER BOARD**



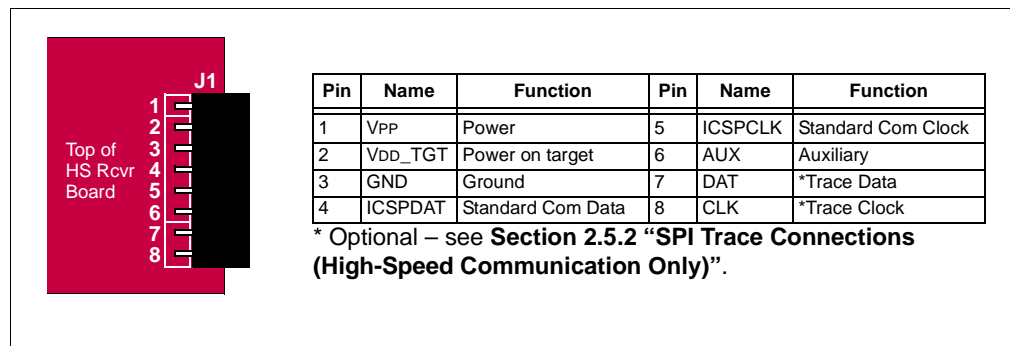
## 13.7.2 High-Speed Receiver Board

A high-speed receiver board assembly is also required when using LVDS connectivity. This board is a counterpart to the high-speed driver board assembly in the pod. When the driver is active on the pod, the receiver is active in the receiver board. Alternatively, when the driver is active on the receiver board, the corresponding receiver is active in the driver board, providing transmitting and receiving capability at both extremes. The receiver board contains an 8-pin, 0.100 inch centers header, and is used to connect to the target board or a header board. The receiver board circuitry may be implemented on the target system to avoid using the receiver board.

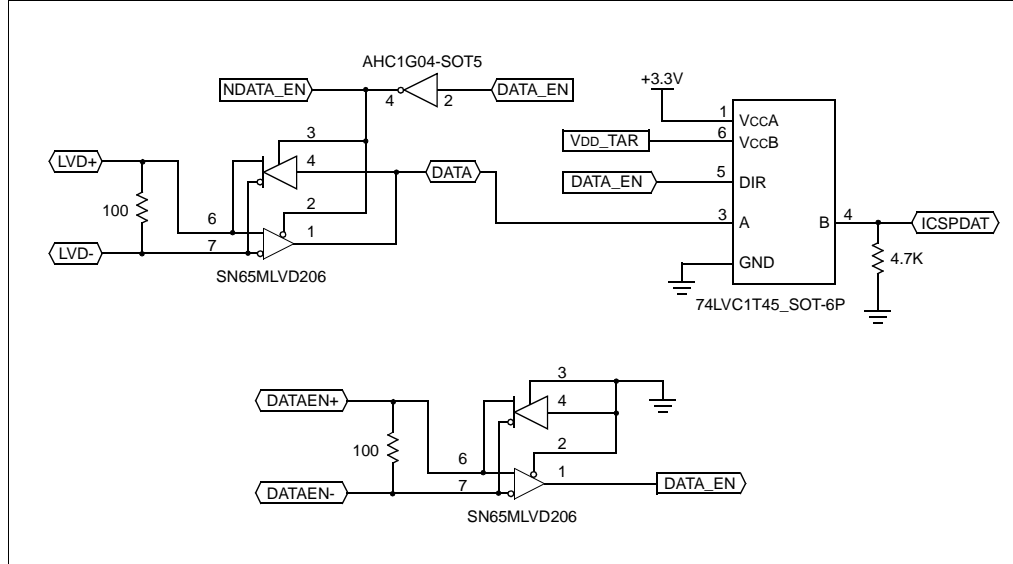
**FIGURE 13-5: MODULAR CONNECTORS PINOUT OF HIGH-SPEED RECEIVER BOARD**



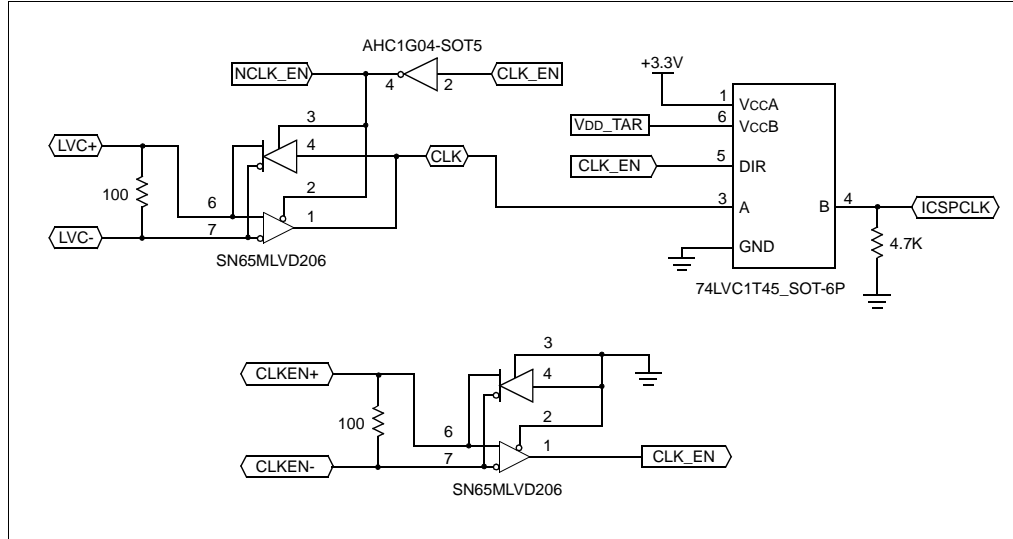
**FIGURE 13-6: 8-PIN HEADER PINOUT OF HIGH-SPEED RECEIVER BOARD**



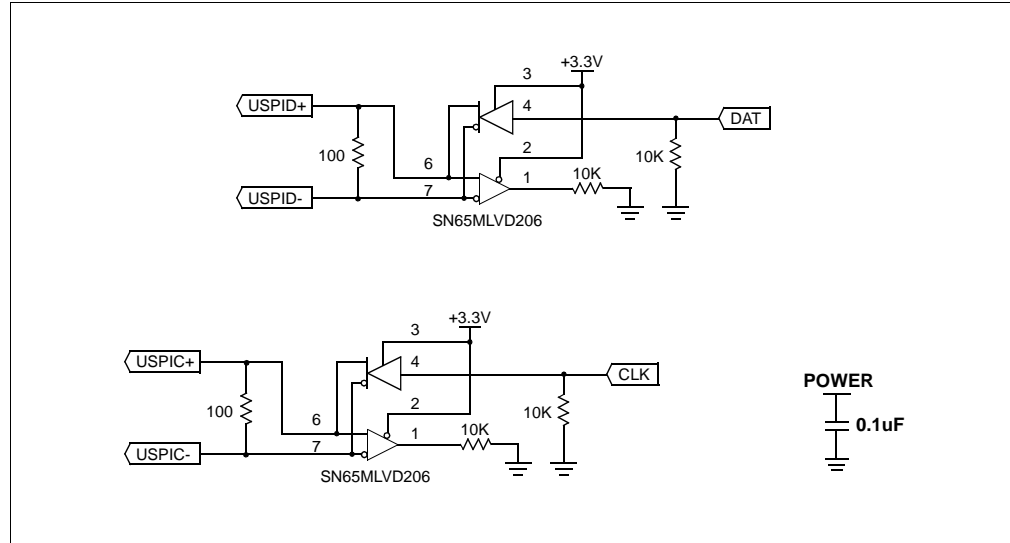
**FIGURE 13-7: RECEIVER BOARD SCHEMATIC – ICSPDAT**



**FIGURE 13-8: RECEIVER BOARD SCHEMATIC – ICSPCLK**



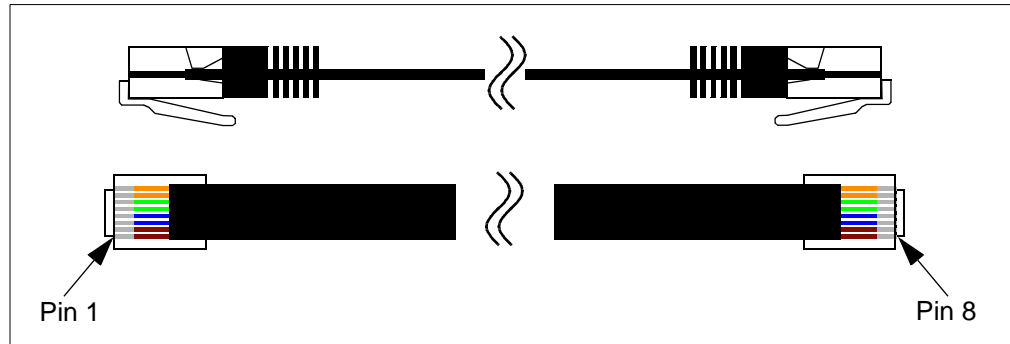
**FIGURE 13-9: RECEIVER BOARD SCHEMATIC – DAT & CLK**



### 13.7.3 LVDS Cables and Target Pinout

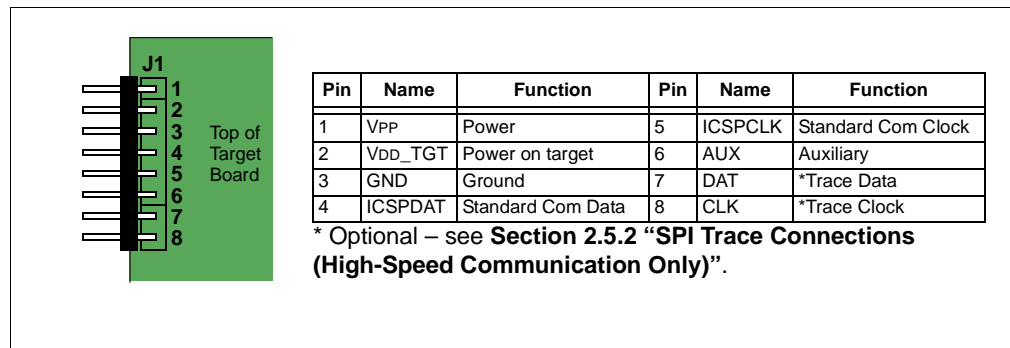
The emulator-to-target cable length for proper operation has been tested for this driver/receiver board combination and is shipped in the Performance Pak. The recommended lengths are 3 feet, with a maximum of 10 feet.

**FIGURE 13-10: LVDS CABLE**



The target board should have the following 8-pin connection pinout to plug in to the receiver board.

**FIGURE 13-11: 8-PIN HEADER PINOUT AT TARGET**





## 13.8 MPLAB REAL ICE ISOLATOR UNIT

The MPLAB REAL ICE Isolator Unit (AC244005) is an useful accessory to the MPLAB REAL ICE in-circuit emulator system. The isolator enables connectivity for AC line and high-voltage applications not referenced to ground. Typically, these are consumer applications such as light dimmers, vacuum cleaners, washing machines, and other types of motor-based systems where the MCU uses a non-isolated power supply.

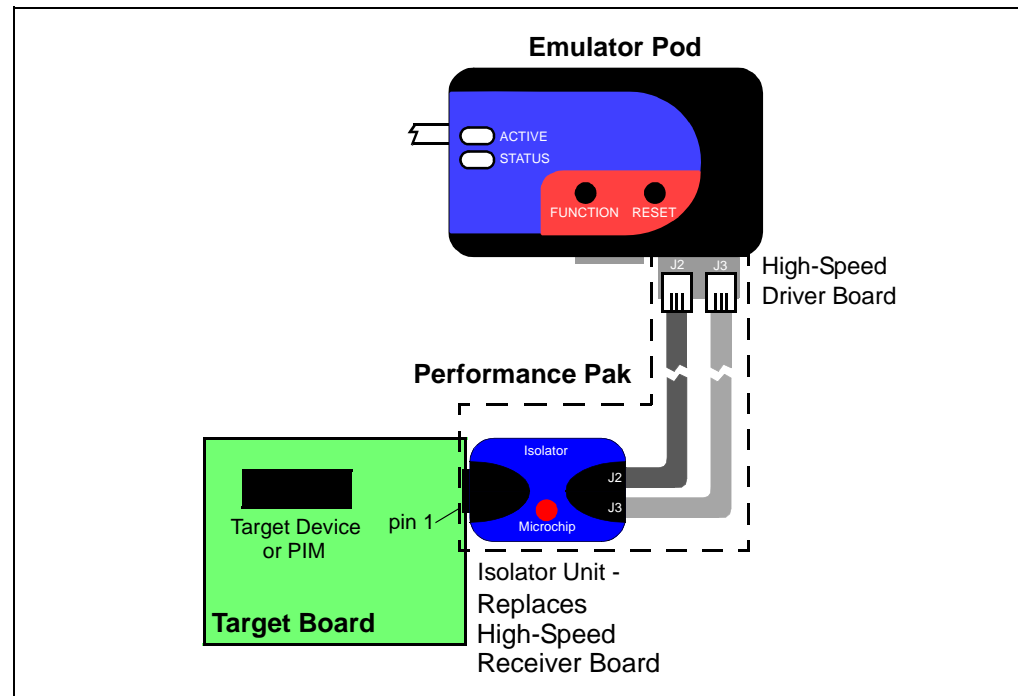
|  <b>DANGER</b> |  |
|---|--|
|                | <b>Do not remove the board enclosure. Depending on your application, you could be exposing yourself to dangerous voltage levels.</b> |

The isolator connects on one end to the Performance Pak high-speed cables. The isolator connects on the other end to the target using an 8-pin, single-line ICSP connector. See Figure 13-12.

To use this board:

1. Obtain a Performance Pak (AC244002). This board can **only** be used with the Performance Pak.
2. Use this board instead of the high-speed receiver board. See **Section 13.7 “High-Speed Communication Hardware”** for pinouts.

**FIGURE 13-12: HIGH-SPEED EMULATOR SYSTEM USING ISOLATOR UNIT**



## 13.8.1 Isolator Device Support

**Note:** Isolation does not support trace.

Because of the complexity and high dynamic voltage range, the isolator currently provides MCU technology support for only the following devices:

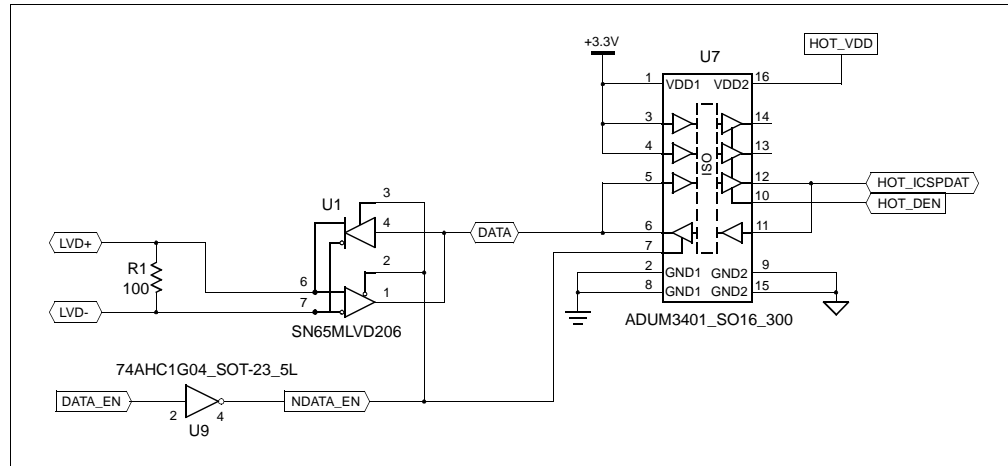
- Some 8-bit devices (PIC18FXXJ)
- All 16-bit devices (dsPIC33F, PIC24F/H)
- All 32-bit devices (PIC32MX)

## 13.8.2 Isolator Unit Design

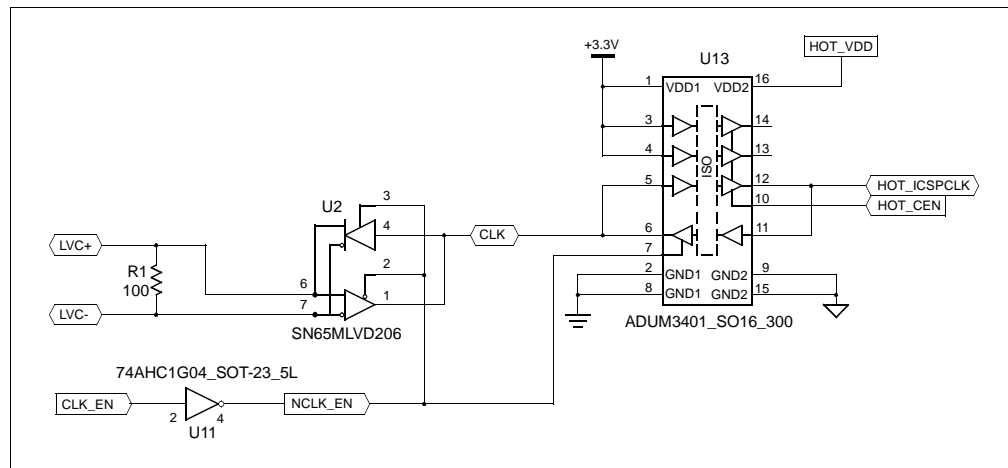
The isolator is a bridge where electrically hot signals are passed through transparently to the emulator. The ICSP interface signals are magnetically or optically isolated providing up to 2.5KV equivalent isolation protection. The isolator is housed in its own enclosure providing an additional measure of safety.

This unit contains the same circuitry as the high-speed receiver board (see **Section 13.7.2 “High-Speed Receiver Board”**) plus isolation circuitry (see the following schematics).

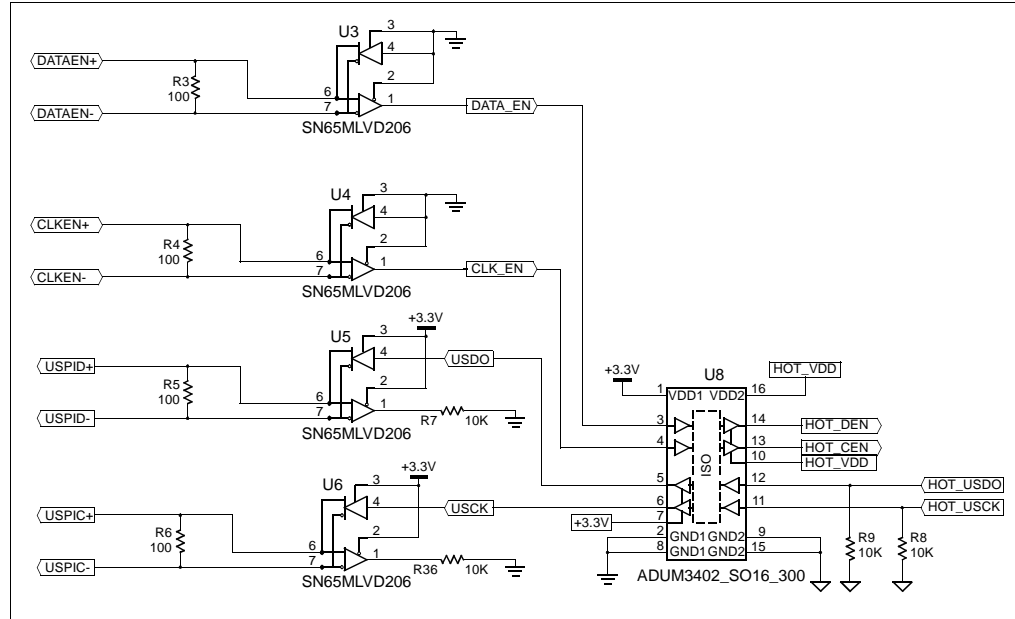
**FIGURE 13-13: ISOLATOR UNIT SCHEMATIC – ICSPDAT**



**FIGURE 13-14: ISOLATOR UNIT SCHEMATIC – ICSPCLK**



**FIGURE 13-15: ISOLATOR UNIT SCHEMATIC – DAT & CLK**



**FIGURE 13-16: ISOLATOR UNIT SCHEMATIC – POWER**

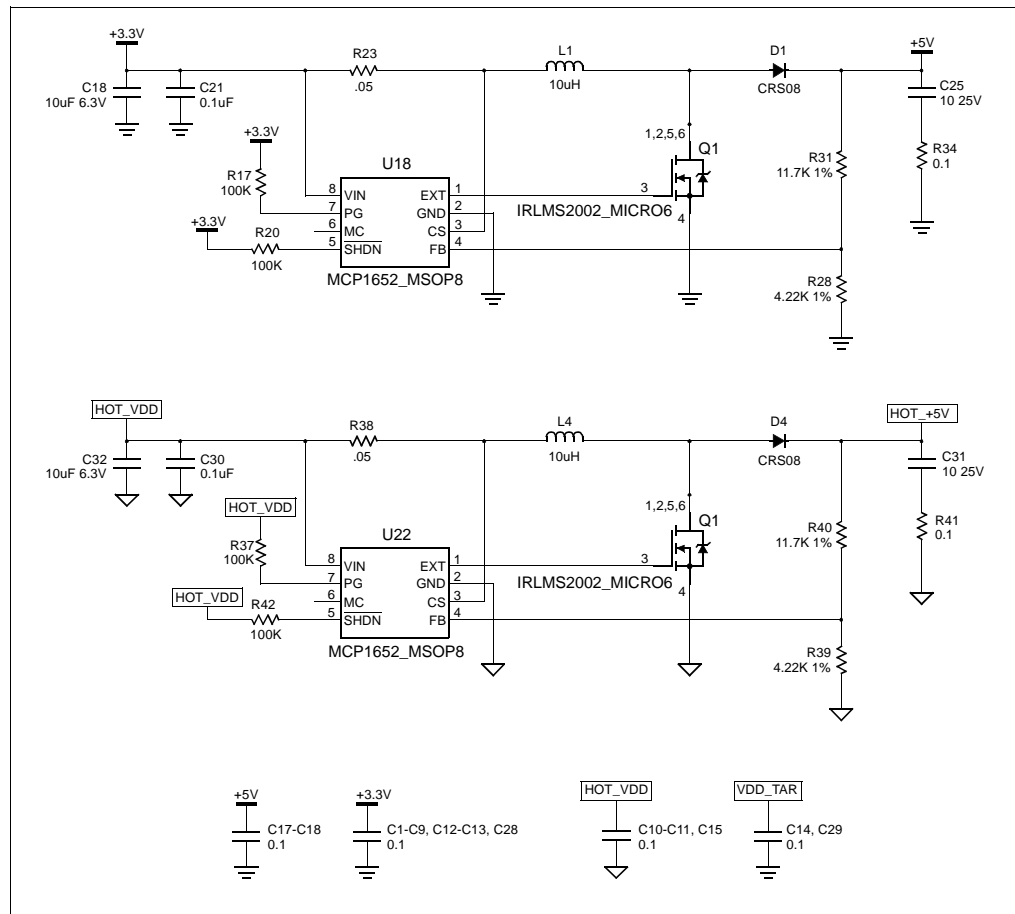
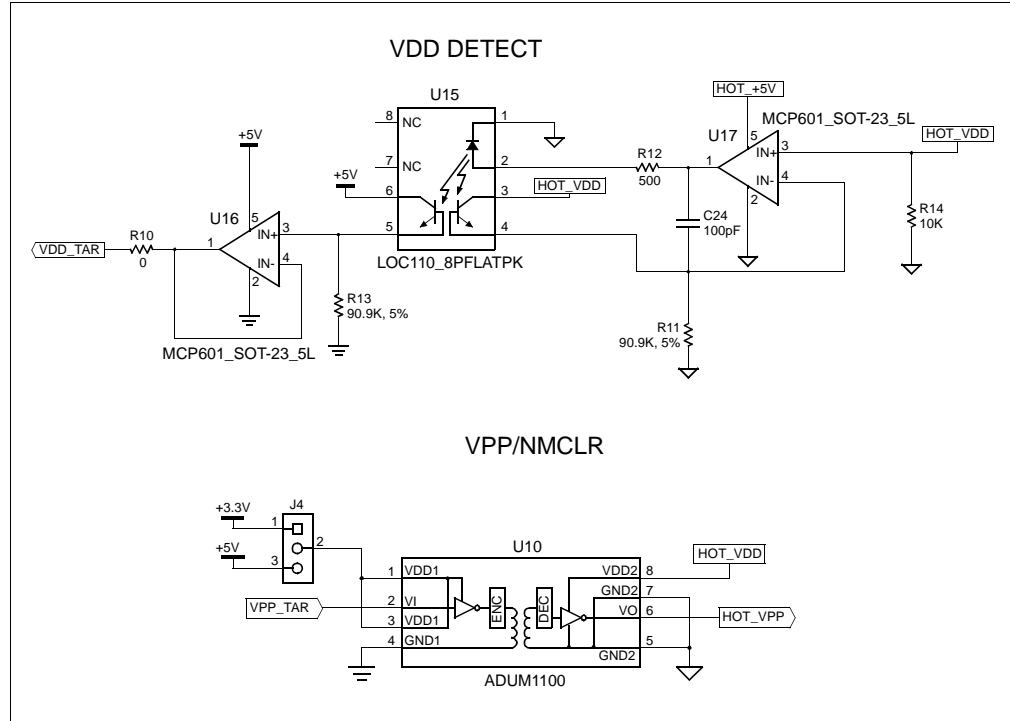


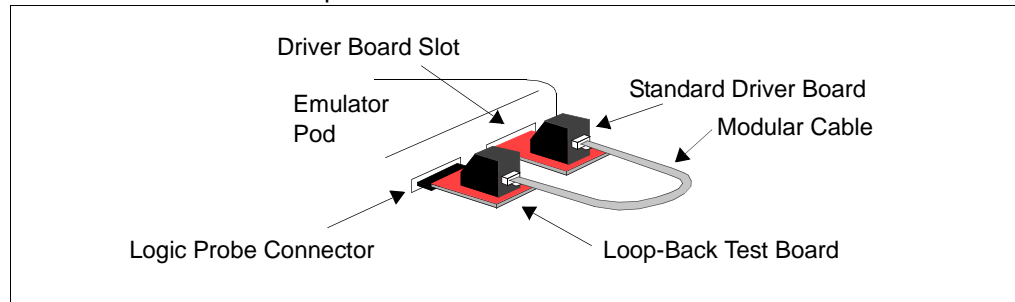
FIGURE 13-17: ISOLATOR UNIT SCHEMATIC – VDD, VPP, MCLR



## 13.9 LOOP-BACK TEST BOARD

This board (included with DV244005) 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. Connect the loop-back test board to the emulator using the modular cable.
4. Plug the loop-back test board into the emulator's logic probe socket, modular cable connector side up



5. Connect the emulator to the PC.
6. Select the MPLAB REAL ICE in-circuit emulator as either a debugger or programmer in MPLAB IDE.
7. Select *Debugger>Settings* or *Programmer>Settings*, **Status** tab, and click **Run Loopback Test**.

MPLAB IDE will detect and run the complete loop-back test and give you a status (PASS/FAIL). The loop-back test board detection works by applying a short pulse on EXT0 and detecting it on EXT7 on the logic probe connector (**Section 13.5.4 “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.

## 13.10 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 K $\Omega$  load on VDD\_TGT.

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

- **Section 2.4.3 “Target Connection Circuitry”**
- **Section 2.4.4 “Circuits That Will Prevent the Emulator From Functioning”**

# MPLAB® REAL ICE™ In-Circuit Emulator User's Guide

---

NOTES:

---

---

## **Appendix A. Revision History**

---

---

### Revision A (September 2006)

- Initial release of this document.

### Revision B (May 2008)

- Additional chapters:
  - Tutorial
  - FAQ
  - Error Messages
- “Debugging and Programming” chapter expanded into several chapters under Part 2 - Features.
- “Device and Feature Support” section added to “Overview” chapter.
- Updates to most existing chapters.

### Revision C (September 2009)

- Two mini-posters replaced by one under “Recommended Reading” in the “Preface”.
- Device and Feature Support tables in “Overview” updated.
- Tool comparison table added in “Operation” chapter.
- Table added to “Quick Debug/Program Reference” in “General Setup”.
- Updated “Data Capture and Runtime Watches” section in “Debug for 32-Bit Devices”.
- “Setting Up and Using Trace” broken up into two sections in “Debug for 32-Bit Devices”: one for PIM usage and one for MCU on board.
- 22 ohm resistors text added to diagram “PIC32MX360F512L PIM Pin Connection Diagram” in “Debug for 32-Bit Devices”.
- Watch symbol support info added in “Emulator Function Summary” chapter.
- Settings dialog info may be saved in workspace note added in “Emulator Function Summary” chapter. New settings also added.
- Isolation unit section added to “Hardware” chapter.
- More connector circuitry detail shown in the “Hardware” chapter.
- Troubleshooting section added with “Troubleshooting First Steps” chapter, “FAQ” and “Errors” chapters.

# MPLAB® REAL ICE™ In-Circuit Emulator User's Guide

---

NOTES:



---

---

## Glossary

---

---

**Absolute Section**

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

**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 arabic 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**

C30 – An unnamed structure.

C18 – 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 caster
{
    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**

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

## **Archiver**

A tool that creates and manipulates libraries.

## **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 lower case letters, digits, symbols and control characters.

## **Assembler**

A language tool that translates assembly language source code into machine code.

## **Assembly Language**

A programming language that describes binary machine code in a symbolic form.

## **Assigned Section**

A 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**

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

## **Attribute, Section**

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

## **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.

Under the Edit menu, select Bookmarks to manage bookmarks. Toggle (enable / disable) a bookmark, move to the next or previous bookmark, or clear all bookmarks.

## **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**

A general-purpose programming language which features economy of expression, modern control flow and data structures, and a rich set of operators.

## **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**

Under the MPLAB IDE Project menu, 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.

## **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 microcontroller 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.

## **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.

## **Debugger**

Hardware that performs debugging.

## **Debugger System**

The debugger systems include the pod, processor module, device adapter, target board, cables, and MPLAB IDE software.

## **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 microcontroller device with digital signal processing capability, i.e., Microchip dsPIC DSC devices.

## **Digital Signal Processing**

The computer manipulation of digital signals, commonly analog signals (sound or image) which have been converted to digital form (sampled).

## **Digital Signal Processor**

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.

## **DSC**

See Digital Signal Controller.

## **DSP**

See Digital Signal Processor.

## **dsPIC DSCs**

dsPIC Digital Signal Controllers (DSCs) refers to all Microchip DSC families.

## **DWARF**

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

## **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**

The process of executing software loaded into emulation memory as if it were firmware residing on a microcontroller device.

**Emulation Memory**

Program memory contained within the emulator.

**Emulator**

Hardware that performs emulation.

**Emulator System**

The MPLAB ICE 2000 and MPLAB ICE 4000 emulator systems include the pod, processor module, device adapter, target board, cables, and MPLAB IDE software. The MPLAB REAL ICE system consists of a pod, a driver (and potentially a receiver) card, target board, cables, and MPLAB IDE software.

**Endianness**

The ordering of bytes in a multi-byte object.

**Environment**

IDE – The particular layout of the desktop for application development.

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 File**

A file containing error messages and diagnostics generated by a language tool.

**Errors**

Errors report problems that make it impossible to continue processing your program. When possible, errors identify the source file name and line number where the problem is apparent.

**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 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**

In Extended mode, the compiler will utilize the extended instructions (i.e., `ADDFSR`, `ADDLW`, `CALLW`, `MOVWF`, `MOVSS`, `PUSHL`, `SUBFSR` and `SUBLW`) 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.

## **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.

## **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>`.

**GPR**

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

**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**

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

**Hex File**

An ASCII file containing hexadecimal addresses and values (hex code) suitable for programming a device.

**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.

**ICD**

In-Circuit Debugger. MPLAB ICD and PICkit (with Debug Express), are Microchip's in-circuit debuggers.

**ICE**

In-Circuit Emulator. MPLAB ICE 2000 and MPLAB ICE 4000 system are Microchip's classic in-circuit emulators. MPLAB REAL ICE system is Microchip's next-generation in-circuit emulator.

**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. MPLAB IDE is Microchip's integrated development environment.

**Identifier**

A function or variable name.

## **IEEE**

Institute of Electrical and Electronics Engineers.

## **Import**

Bring data into the MPLAB 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.

## **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 Request**

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**

ALU30, C18, C30 – A function that handles an interrupt.

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.

## **IRQ**

See Interrupt Request.

## **ISO**

See International Organization for Standardization.

## **ISR**

See Interrupt Service Routine.



**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.

**Librarian**

See Archiver.

**Library**

See Archive.

**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.

**Loop-Back 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.

LVDS differs from normal input/output (I/O) in a few ways:

Normal digital I/O works with 5 volts as a high (binary '1') and 0 volts as a low (binary '0'). When you use a differential, you add a third option (-5 volts), which provides an extra level with which to encode, and results in a higher maximum data transfer rate.

A higher data transfer rate means fewer wires are required, as in UW (Ultra Wide) and UW-2/3 SCSI hard disks, which use only 68 wires. These devices require a high transfer rate over short distances. Using standard I/O transfer, SCSI hard drives would require a lot more than 68 wires.

Low voltage means that the standard 5 volts is replaced by either 3.3 volts or 1.5 volts. LVDS uses a dual wire system, running 180 degrees of each other. This enables noise to travel at the same level, which in turn can get filtered more easily and effectively.

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>.

## **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, i.e., with a `make`.

Under *Project>Build Options>Project, Directories* tab, you must have selected “Assemble/Compile/Link in the project directory” under “Build Directory Policy” for this feature to work.

## **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**

C30 – A representation of the memory available to the application.

C18 – 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.

## **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's in-circuit debuggers that works with MPLAB IDE. The ICDs supports Flash devices with built-in debug circuitry. The main component of each ICD is the pod. A complete system consists of a pod, header board (with a *device-ICD*), target board, cables, and MPLAB IDE software.

## **MPLAB ICE 2000/4000**

**Not recommended for new designs. See the MPLAB REAL ICE in-circuit emulator.**

Microchip's classic in-circuit emulators that work with MPLAB IDE. MPLAB ICE 2000 supports 8-bit PIC MCUs. MPLAB ICE 4000 supports PIC18F and PIC24 MCUs and dsPIC DSCs. The main component of each ICE is the pod. A complete system consists of a pod, processor module, cables, and MPLAB IDE software.

## **MPLAB IDE**

Microchip's Integrated Development Environment. MPLAB 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 or stand-alone. Replaces PRO MATE II.

## **MPLAB REAL ICE™ In-Circuit Emulator**

Microchip's next-generation in-circuit emulators that works with MPLAB IDE. The MPLAB REAL ICE emulator supports PIC MCUs and dsPIC DSCs. The main component of each ICE is the pod. A complete system consists of a pod, a driver (and potentially a receiver) card, cables, and MPLAB IDE software.

## **MPLAB SIM**

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

## **MPLIB™ Object Librarian**

Microchip's librarian that can work with MPLAB 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, though it does not have to be.

## **MRU**

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

## **Native Data Size**

For Native trace, the size of the variable used in a Watch 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 project component.

## **Non-Extended Mode**

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 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.

## **Object Code**

The machine code generated by an assembler or compiler.

## **Object File**

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.

## **One-to-One Project-Workspace Model**

The most common configuration for application development in MPLAB IDE to is have one project in one workspace. Select *Configure>Settings, Projects* tab and check "Use one-to-one project-workspace model".

## **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.

## **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 1, 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.

## **PICSTART Plus**

A developmental device programmer from Microchip. Programs 8-, 14-, 28-, and 40-pin PIC microcontrollers. Must be used with MPLAB IDE software.

## **Plug-ins**

The MPLAB 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

MPLAB REAL ICE system: The box that contains the emulation control circuitry for the ICE device on the header or target board. An ICE device can be a production device with built-in ICE circuitry or a special ICE version of a production device (i.e., *device-ICE*).

MPLAB ICD: The box that contains the debug control circuitry for the ICD device on the header or target board. An ICD device can be a production device with built-in ICD circuitry or a special ICD version of a production device (i.e., *device-ICD*).

MPLAB ICE 2000/4000: The external emulator box that contains emulation memory, trace memory, event and cycle timers, and trace/breakpoint logic.

## 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. MPLAB C30 uses attributes to convey this information.

## Precedence

Rules that define the order of evaluation in expressions.

## PRO MATE II

**No longer in Production. See the MPLAB PM3 device programmer.**

A device programmer from Microchip. Programs most PIC microcontrollers as well as most memory and KEELOQ devices. Can be used with MPLAB IDE or stand-alone.

## 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.

Microchip production programmers, such as MPLAB PM3, MPLAB REAL ICE in-circuit emulator, and MPLAB ICD 3, have been designed with robustness in mind to tolerate these demanding environments.

Some top-end tools have additional accessories. The MPLAB REAL ICE Performance Pak has accelerators to speed up the communication and In-Circuit Serial Programming (ICSP) process. The MPLAB PM3 programmer has interchangeable socket modules to support various devices out-of-circuit.

## 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

ALU30 – 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**

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.

ALU30, C30 – 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.

## **PWM Signals**

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

## **Qualifier**

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

## **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.

## **Real-Time Watch**

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

## **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 ASM30 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**

ALU30 – 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.

## **Scenario**

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

## **Section**

A portion of an application located at a specific address of memory.

## **Section Attribute**

A 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.

## **SFR**

See Special Function Registers.



## **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 version.

## **Simulator**

A software program that models the operation of devices.

## **Single Step**

This command steps through code, one instruction at a time. After each instruction, MPLAB 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 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**

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 typically managed by the compiler when developing code in a high-level language.

## **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 you own changes.

## 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 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 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."

## 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 functions.

**Trace**

An emulator or simulator function that logs program execution. The emulator logs program execution into its trace buffer which is uploaded to MPLAB IDE's 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.

**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.

## **Vector**

The memory locations that an application will jump to when either a reset or interrupt occurs.

## **Warning**

IDE – An alert that is provided to warn you of a situation that would cause physical damage to a device, software file, or equipment.

ALU30, C30 – Warnings report conditions that may indicate a problem, but do not halt processing. In MPLAB C30, warning messages report the source file name and line number, but include the text 'warning:' to distinguish them from error messages.

## **Watch Variable**

A variable that you may monitor during a debugging session in a Watch window.

## **Watch Window**

Watch windows contain a list of watch variables that are updated at each breakpoint.

## **Watchdog Timer**

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.

## **WDT**

See Watchdog Timer.

## **Workbook**

For MPLAB SIM stimulator, a setup for generation of SCL stimulus.

## **WorkSpace**

A workspace contains MPLAB IDE information on the selected device, selected debug tool and/or programmer, open windows and their location, and other IDE configuration settings.



# MPLAB® REAL ICE™ IN-CIRCUIT EMULATOR USER'S GUIDE

## Index

### Numerics

|  |    |
|--|----|
| 16-Bit (Data Memory) Devices, Trace..... | 76 |
| 32-Bit Devices, Trace.....               | 84 |
| 8-Bit (Data Memory) Devices, Trace.....  | 76 |

### A

|                        |     |
|------------------------|-----|
| Abort Operation.....   | 110 |
| AC Line Isolation..... | 137 |
| AC244002.....          | 132 |
| AC244005.....          | 137 |
| AC244006.....          | 86  |
| Animate.....           | 109 |
| AVdd, AVss.....        | 23  |

### B

|                               |                |
|-------------------------------|----------------|
| Blank Check.....              | 120            |
| Breakpoints                   |                |
| Dialog.....                   | 112            |
| Enabling.....                 | 111            |
| Hardware.....                 | 48, 73, 123    |
| Setup.....                    | 48, 73, 110    |
| Software.....                 | 52, 73, 123    |
| Build Configuration.....      | 28, 30, 82, 91 |
| Build Options, Trace tab..... | 78, 82, 119    |

### C

|   |               |
|---|---------------|
| Cables  |               |
| Length.....   | 128, 131, 136 |
| USB.....  | 18            |
| Capacitors.....   | 23            |
| Capture, Data.....  | 75, 83, 116   |
| CD-ROM.....   | 16            |
| Circuits That Will Prevent the Emulator From Functioning..... | 23            |
| CLK.....  | 22, 24        |
| Clock Speed.....  | 46            |
| Code Protect.....   | 28, 69        |
| CodeGuard Security.....                                       | 125           |
| Comm Channel.....   | 69            |
| Command-line Programming.....                                 | 30, 70        |
| Configuration Bits.....                                       | 28, 44, 69    |
| Converter Board, High-Speed to Standard.....                  | 16            |
| CPU, Stall.....   | 85, 123       |
| Creating a Hex File.....                                      | 43            |
| Customer Notification Service.....                            | 11            |
| Customer Support.....   | 12            |

### D

|                                    |                  |
|------------------------------------|------------------|
| DAT.....                           | 22, 24           |
| Data Capture.....                  | 75, 83, 110, 116 |
| Data Rate.....                     | 130, 133         |
| Debug                              |                  |
| Executive.....                     | 29               |
| Registers.....                     | 30               |
| Sequence of Operations.....        | 28               |
| Top Reasons Why You Can't.....     | 91               |
| Debug Read.....                    | 110              |
| Debug/Program Quick Reference..... | 70               |
| Debugger Menu.....                 | 47               |
| Debugging.....                     | 47               |
| Demo Board.....                    | 46               |
| Device and Feature Support.....    | 16               |
| Device Debug Resource Toolbar..... | 47, 73           |
| Device ID.....                     | 120              |
| DMCI.....                          | 75               |
| Documentation                      |                  |
| Conventions.....                   | 9                |
| Layout.....                        | 8                |
| Download Firmware.....             | 123              |
| Driver Board                       |                  |
| High-Speed.....                    | 16, 18, 133      |
| Standard.....                      | 16, 18, 130      |
| Durability, Card Guide.....        | 128              |
| DV244005.....                      | 128              |

### E

|                                   |          |
|-----------------------------------|----------|
| EMUC, EMUD.....                   | 24       |
| Erase.....                        | 120      |
| Erase all before Program.....     | 122      |
| Erase All Before Programming..... | 45       |
| Erase Flash Device.....           | 110      |
| Explorer 16 Demo Board.....       | 37       |
| External Triggers.....            | 116, 129 |
| EXTn.....                         | 25, 129  |

### F

|                                     |     |
|-------------------------------------|-----|
| Firmware                            |     |
| Cancelled Download.....             | 97  |
| Disconnected while Downloading..... | 97  |
| Downloading.....                    | 123 |
| Freeze on Halt.....                 | 96  |

### G

|                                 |     |
|---------------------------------|-----|
| General Corrective Actions..... | 103 |
|---------------------------------|-----|

### H

|                           |     |
|---------------------------|-----|
| Halt.....                 | 109 |
| Hardware Breakpoints..... | 48  |
| Header Board              |     |

# MPLAB® REAL ICE™ In-Circuit Emulator User's Guide

|                                     |                        |                                      |             |
|-------------------------------------|------------------------|--------------------------------------|-------------|
| Specification.....                  | 10                     | Port A .....                         | 37          |
| Hex File .....                      | 43                     | Port Trace.....                      | 25, 78      |
| Hibernate mode.....                 | 96, 97, 128            | PORTx.....                           | 25          |
| High Voltage Isolation .....        | 137                    | Power-Down mode.....                 | 96, 97, 128 |
| High-Speed Communication .....      | 20                     | Preserve Program Memory .....        | 122         |
| Connections .....                   | 22                     | Program.....                         | 110, 120    |
| Driver Board .....                  | 133                    | Program after successful build ..... | 122         |
| Receiver Board .....                | 134                    | Program Memory Tab .....             | 45          |
| Hubs, USB .....                     | 128                    | Programming.....                     | 57          |
| <b>I</b>                            |                        | Command-line.....                    | 30, 70      |
| I/O Port Trace.....                 | 25, 78                 | Options .....                        | 45          |
| ICSP.....                           | 27, 28, 30             | Production .....                     | 30, 70      |
| Indicator Lights .....              | 129                    | Project Wizard .....                 | 41, 68      |
| Instruction Trace .....             | 26, 84                 | Pull-ups .....                       | 23          |
| Internet Address, Microchip.....    | 11                     | Push Buttons .....                   | 128         |
| Isolator Unit .....                 | 16, 137                | <b>Q</b>                             |             |
| <b>J</b>                            |                        | Quick Reference                      |             |
| JTAG .....                          | 69                     | Debug/Program.....                   | 70          |
| <b>K</b>                            |                        | Trace.....                           | 82          |
| Kit Components.....                 | 16                     | <b>R</b>                             |             |
| <b>L</b>                            |                        | Read.....                            | 110, 120    |
| LEDs .....                          | 37, 129                | Reading, Recommended.....            | 10          |
| Light Icons .....                   | 38                     | Readme .....                         | 10          |
| Loading Program and Debug Code..... | 46                     | REAL ICE Tab.....                    | 119         |
| Logic Probe Connector .....         | 36, 129                | Real Time Watch.....                 | 53          |
| External Triggers.....              | 74                     | REALICECMD .....                     | 30, 70      |
| I/O Electrical Specifications.....  | 130                    | Receiver Board, High-Speed.....      | 134         |
| I/O Port Trace .....                | 25                     | Reconnect .....                      | 111         |
| Pinout.....                         | 129                    | Reserved Resources by Device .....   | 30          |
| Logic Probes .....                  | 16, 36                 | Reset                                |             |
| Loop-Back Test Board .....          | 16, 141                | Hold in .....                        | 120         |
| LVDS.....                           | 133, 134               | Release from.....                    | 120         |
| <b>M</b>                            |                        | Reset Processor.....                 | 110         |
| MA320001 .....                      | 87                     | Resistors .....                      | 23          |
| MA320002 .....                      | 87                     | RIErr0001 .....                      | 99          |
| Modular Interface Cable.....        | 27                     | RIErr0002 .....                      | 99          |
| MPLAB C30 .....                     | 41                     | RIErr0003 .....                      | 99          |
| MPLAB IDE .....                     | 33                     | RIErr0005 .....                      | 99          |
| MPLAB REAL ICE Defined .....        | 15                     | RIErr0006 .....                      | 99          |
| MPLAB REAL ICE Detected .....       | 120                    | RIErr0007 .....                      | 99          |
| <b>N</b>                            |                        | RIErr0008 .....                      | 99          |
| Native Trace.....                   | 24, 54, 78             | RIErr0009 .....                      | 99          |
| <b>O</b>                            |                        | RIErr0010 .....                      | 99          |
| Oscillator .....                    | 69                     | RIErr0011 .....                      | 99          |
| Output Window, REAL ICE tab .....   | 119                    | RIErr0012 .....                      | 100         |
| <b>P</b>                            |                        | RIErr0013 .....                      | 100         |
| Parallel Trace .....                | 25                     | RIErr0014 .....                      | 100         |
| PC, Power Down.....                 | 96, 97, 128            | RIErr0015 .....                      | 100         |
| Performance Pak .....               | 132                    | RIErr0016 .....                      | 100         |
| PGC, PGD.....                       | 21, 22, 23, 24, 27, 29 | RIErr0017 .....                      | 100         |
| PIC24FJ128GA010, Tutorial .....     | 37                     | RIErr0018 .....                      | 100         |
| PIC32 Instruction Trace.....        | 26, 84                 | RIErr0019 .....                      | 100         |
| PIM.....                            | 19, 20, 84             | RIErr0020 .....                      | 100         |
| Pod.....                            | 16, 18                 | RIErr0021 .....                      | 100         |
|                                     |                        | RIErr0022 .....                      | 100         |
|                                     |                        | RIErr0023 .....                      | 100         |
|                                     |                        | RIErr0024 .....                      | 100         |
|                                     |                        | RIErr0025 .....                      | 100         |
|                                     |                        | RIErr0026 .....                      | 100         |

|  |                  |
|--|------------------|
| RIErr0027.....                             | 100              |
| RIErr0028.....                             | 100              |
| RIErr0029.....                             | 100              |
| RIErr0030.....                             | 101              |
| RIErr0031.....                             | 101              |
| RIErr0032.....                             | 101              |
| RIErr0033.....                             | 101              |
| RIErr0034.....                             | 101              |
| RIErr0035.....                             | 101              |
| RIErr0036.....                             | 101              |
| RIErr0037.....                             | 101              |
| RIErr0038.....                             | 101              |
| RIErr0039.....                             | 101              |
| RIErr0040.....                             | 101              |
| RIErr0041.....                             | 101              |
| RIErr0045.....                             | 101              |
| RIErr0046.....                             | 102              |
| RIErr0047.....                             | 102              |
| RIErr0048.....                             | 102              |
| RIErr0049.....                             | 102              |
| RIErr0050.....                             | 102              |
| RIErr0051.....                             | 102              |
| RIErr0052.....                             | 102              |
| RIErr0053.....                             | 102              |
| RIErr0054.....                             | 102              |
| RIErr0055.....                             | 102              |
| RIErr0056.....                             | 102              |
| RIErr0057.....                             | 102              |
| RIErr0058.....                             | 102              |
| RIErr0059.....                             | 102              |
| RIErr0060.....                             | 102              |
| RIErr0061.....                             | 103              |
| RIErr0062.....                             | 103              |
| RIErr0063.....                             | 103              |
| RIErr0064.....                             | 103              |
| RIErr0065.....                             | 103              |
| RIErr0066.....                             | 103              |
| RIErr0067.....                             | 103              |
| RIErr0068.....                             | 103              |
| RIErr0069.....                             | 103              |
| RIErr0070.....                             | 103              |
| RIErr0071.....                             | 103              |
| RIErr0072.....                             | 103              |
| RIErr0073.....                             | 103              |
| RIErr0080.....                             | 103              |
| Run.....                                   | 109              |
| Run after successful program.....          | 122              |
| Running code.....                          | 47               |
| Runtime Watch.....                         | 75, 83, 116, 125 |
| <b>S</b>                                   |                  |
| Schematics, 100-Pin PIM.....               | 87               |
| Secure Segments.....                       | 125              |
| Selecting Device and Development Mode..... | 38               |
| Serial Trace.....                          | 24               |
| Set a Breakpoint.....                      | 48               |
| Setting Program and Debug Options.....     | 43, 44           |
| Setting Up Hardware and Software.....      | 37               |
| Software Breakpoints.....                  | 52               |
| SPI Trace.....                             | 24, 78           |
| SQTP.....                                  | 70               |

|                             |         |
|-----------------------------|---------|
| Stall CPU.....              | 85, 123 |
| Standard Communication..... | 18      |
| Connections.....            | 21      |
| Driver Board.....           | 130     |
| Start/Stop Triggers.....    | 123     |
| Step.....                   | 110     |
| Stopwatch.....              | 51, 73  |

## T

|                            |                  |
|----------------------------|------------------|
| Table Read Protect.....    | 28               |
| Target Connection.....     |                  |
| Circuitry.....             | 22               |
| High-Speed.....            | 22               |
| I/O Port.....              | 25               |
| Improper Circuits.....     | 23               |
| SPI.....                   | 24               |
| Standard.....              | 21               |
| Target Detected.....       | 120              |
| Target Device.....         | 27               |
| Timer1.....                | 37               |
| Toolbar Buttons.....       | 47               |
| Trace.....                 | 15               |
| I/O Port.....              | 25, 78           |
| Native.....                | 24, 78           |
| PIC32 Instruction.....     | 26, 84           |
| SPI.....                   | 24, 78, 134, 136 |
| Trace Quick Reference..... | 82               |
| Trace tab.....             | 78, 82, 119      |
| Trace Window.....          | 117              |
| Transition Socket.....     | 16               |
| Specification.....         | 10, 35           |
| TRCLK.....                 | 84, 85           |
| TRDn.....                  | 84, 85           |
| Triggers.....              | 74, 115          |
| External.....              | 74, 129          |
| Triggers, Start/Stop.....  | 86, 123          |
| Tutorial.....              | 37               |

## U

|                           |          |
|---------------------------|----------|
| Update, Watch Window..... | 116      |
| USB.....                  | 128, 163 |
| Cables.....               | 16, 18   |
| Device Drivers.....       | 33       |
| Hubs.....                 | 128      |

## V

|               |                     |
|---------------|---------------------|
| Vcap.....     | 23                  |
| Vdd, Vss..... | 21, 22, 23, 27, 129 |
| Verify.....   | 120                 |
| Vpp.....      | 21, 22, 23, 27, 28  |

## W

|                          |             |
|--------------------------|-------------|
| Watch.....               |             |
| Real Time.....           | 53          |
| Runtime.....             | 75, 83, 116 |
| Runtime Update Rate..... | 125         |
| Symbols.....             | 117         |
| Window.....              | 49          |
| Watchdog Timer.....      | 28, 69, 164 |
| Web Site, Microchip..... | 11          |



## 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://support.microchip.com>  
Web Address:  
[www.microchip.com](http://www.microchip.com)

**Atlanta**  
Duluth, GA  
Tel: 678-957-9614  
Fax: 678-957-1455

**Boston**  
Westborough, MA  
Tel: 774-760-0087  
Fax: 774-760-0088

**Chicago**  
Itasca, IL  
Tel: 630-285-0071  
Fax: 630-285-0075

**Cleveland**  
Independence, OH  
Tel: 216-447-0464  
Fax: 216-447-0643

**Dallas**  
Addison, TX  
Tel: 972-818-7423  
Fax: 972-818-2924

**Detroit**  
Farmington Hills, MI  
Tel: 248-538-2250  
Fax: 248-538-2260

**Kokomo**  
Kokomo, IN  
Tel: 765-864-8360  
Fax: 765-864-8387

**Los Angeles**  
Mission Viejo, CA  
Tel: 949-462-9523  
Fax: 949-462-9608

**Santa Clara**  
Santa Clara, CA  
Tel: 408-961-6444  
Fax: 408-961-6445

**Toronto**  
Mississauga, Ontario,  
Canada  
Tel: 905-673-0699  
Fax: 905-673-6509

### ASIA/PACIFIC

**Asia Pacific Office**  
Suites 3707-14, 37th Floor  
Tower 6, The Gateway  
Harbour City, Kowloon  
Hong Kong  
Tel: 852-2401-1200  
Fax: 852-2401-3431

**Australia - Sydney**  
Tel: 61-2-9868-6733  
Fax: 61-2-9868-6755

**China - Beijing**  
Tel: 86-10-8528-2100  
Fax: 86-10-8528-2104

**China - Chengdu**  
Tel: 86-28-8665-5511  
Fax: 86-28-8665-7889

**China - Hong Kong SAR**  
Tel: 852-2401-1200  
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: 86-21-5407-5533  
Fax: 86-21-5407-5066

**China - Shenyang**  
Tel: 86-24-2334-2829  
Fax: 86-24-2334-2393

**China - Shenzhen**  
Tel: 86-755-8203-2660  
Fax: 86-755-8203-1760

**China - Wuhan**  
Tel: 86-27-5980-5300  
Fax: 86-27-5980-5118

**China - Xiamen**  
Tel: 86-592-2388138  
Fax: 86-592-2388130

**China - Xian**  
Tel: 86-29-8833-7252  
Fax: 86-29-8833-7256

**China - Zhuhai**  
Tel: 86-756-3210040  
Fax: 86-756-3210049

### ASIA/PACIFIC

**India - Bangalore**  
Tel: 91-80-3090-4444  
Fax: 91-80-3090-4080

**India - New Delhi**  
Tel: 91-11-4160-8631  
Fax: 91-11-4160-8632

**India - Pune**  
Tel: 91-20-2566-1512  
Fax: 91-20-2566-1513

**Japan - Yokohama**  
Tel: 81-45-471- 6166  
Fax: 81-45-471-6122

**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: 886-3-6578-300  
Fax: 886-3-6578-370

**Taiwan - Kaohsiung**  
Tel: 886-7-536-4818  
Fax: 886-7-536-4803

**Taiwan - Taipei**  
Tel: 886-2-2500-6610  
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

**France - Paris**  
Tel: 33-1-69-53-63-20  
Fax: 33-1-69-30-90-79

**Germany - Munich**  
Tel: 49-89-627-144-0  
Fax: 49-89-627-144-44

**Italy - Milan**  
Tel: 39-0331-742611  
Fax: 39-0331-466781

**Netherlands - Drunen**  
Tel: 31-416-690399  
Fax: 31-416-690340

**Spain - Madrid**  
Tel: 34-91-708-08-90  
Fax: 34-91-708-08-91

**UK - Wokingham**  
Tel: 44-118-921-5869  
Fax: 44-118-921-5820