



**MPLAB[®] ICD 3
In-Circuit Debugger
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in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates.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, PIC³² logo, rPIC 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, 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.

© 2010, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

 Printed on recycled paper.

ISBN: 978-1-60932-393-6

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® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

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



Table of Contents

Preface	6
Part 1 – Getting Started	
Chapter 1. Overview	
1.1 Introduction	15
1.2 MPLAB ICD 3 In-Circuit Debugger Defined	15
1.3 How the MPLAB ICD 3 In-Circuit Debugger Helps You	16
1.4 MPLAB ICD 3 In-Circuit Debugger Kit Components	16
Chapter 2. Operation	
2.1 Introduction	17
2.2 MPLAB ICD 3 In-Circuit Debugger vs. MPLAB ICE 2000/4000 In-Circuit Emulators	17
2.3 MPLAB ICD 3 In-Circuit Debugger vs. MPLAB ICD 2 Debugger	17
2.4 Debugger To Target Communication	18
2.5 Communication Connections	19
2.6 Debugging with the Debugger	21
2.7 Requirements For Debugging	22
2.8 Programming with the Debugger	24
2.9 Resources Used by the Debugger	24
Chapter 3. Installation	
3.1 Introduction	25
3.2 Installing the Software	25
3.3 Installing the USB Device Drivers	25
3.4 Using the ICD 3 Test Interface Board	25
3.5 Connecting the Target	26
3.6 Setting Up the Target Board	26
3.7 Setting Up MPLAB IDE	27
Chapter 4. General Setup	
4.1 Introduction	29
4.2 Starting the MPLAB IDE Software	29
4.3 Creating a Project	30
4.4 Viewing the Project	30
4.5 Building the Project	30
4.6 Setting Configuration Bits	30
4.7 Setting the Debugger as the Debugger or Programmer	31
4.8 Quick Debug/Program Reference	31
4.9 Debugger/Programmer Limitations	32

MPLAB® ICD 3 In-Circuit Debugger User's Guide

Chapter 5. Tutorial

5.1 Introduction	33
5.2 Setting Up the Environment and Selecting the Device	34
5.3 Creating the Application Code	34
5.4 Running the Project Wizard	37
5.5 Viewing the Project	39
5.6 Viewing Debug Options	40
5.7 Creating a Hex File	41
5.8 Setting Up the Demo Board	43
5.9 Loading Program Code For Debugging	43
5.10 Running Debug Code	44
5.11 Debugging Code Using Breakpoints	44
5.12 Programming the Application	55

Part 2 – Troubleshooting

Chapter 6. Frequently Asked Questions (FAQs)

6.1 Introduction	59
6.2 How Does It Work	59
6.3 What's Wrong	60

Chapter 7. Error Messages

7.1 Introduction	63
7.2 Specific Error Messages	63
7.3 Information Messages	67
7.4 General Corrective Actions	67

Part 3 – Reference

Chapter 8. Basic Debug Functions

8.1 Introduction	73
8.2 Breakpoints and Stopwatch	73

Chapter 9. Debugger Function Summary

9.1 Introduction	75
9.2 Debugging Functions	75
9.3 Debugging Dialogs/Windows	78
9.4 Programming Functions	85
9.5 Settings Dialog	86

Appendix A. Hardware Specification

A.1 Introduction	91
A.2 Highlights	91
A.3 Declaration of Conformity	91
A.4 USB Port/Power	92
A.5 MPLAB ICD 3 Debugger	92
A.6 Standard Communication Hardware	93
A.7 MPLAB ICD 3 Test Interface Board	95
A.8 Target Board Considerations	96

Table of Contents

Glossary	97
Index	117
Worldwide Sales and Service	120

MPLAB® ICD 3 In-Circuit Debugger User's Guide

NOTES:



MPLAB® ICD 3 IN-CIRCUIT DEBUGGER 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) 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 online help. Select the Help menu, and then Topics to open a list of available online help files.

INTRODUCTION

This chapter contains general information that will be useful to know before using the MPLAB ICD 3 in-circuit debugger. Items discussed in this chapter include:

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

DOCUMENT LAYOUT

This document describes how to use the MPLAB ICD 3 in-circuit debugger 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 – Getting Started

- **Chapter 1. Overview** – What the MPLAB ICD 3 in-circuit debugger is, and how it can help you develop your application.
- **Chapter 2. Operation** – The theory of MPLAB ICD 3 in-circuit debugger operation. Explains configuration options.
- **Chapter 3. Installation** – How to install the debugger software and hardware.
- **Chapter 4. General Setup** – How to set up MPLAB IDE to use the debugger.
- **Chapter 5. Tutorial** – A brief tutorial on using the debugger.

Part 2 – Troubleshooting

- **Chapter 6. Frequently Asked Questions (FAQs)** – A list of frequently asked questions, useful for troubleshooting.
- **Chapter 7. Error Messages** – A list of error messages and suggested resolutions.

Part 3 – Reference

- **Chapter 8. Basic Debug Functions** – A description of basic debugger features available in MPLAB IDE when the MPLAB ICD 3 in-circuit debugger is chosen as either the debug or programming tool. This includes the debug features breakpoints, stopwatch, triggering and real-time watches.
- **Chapter 9. Debugger Function Summary** – A summary of debugger functions available in MPLAB IDE when the MPLAB ICD 3 debugger is chosen as the debug or program tool.
- **Appendix A. Hardware Specification** – The hardware and electrical specifications of the debugger system.

CONVENTIONS USED IN THIS GUIDE

This manual uses the following documentation conventions:

DOCUMENTATION CONVENTIONS

Description	Represents	Examples
Arial font:		
Italic characters	Referenced books	<i>MPLAB[®] 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>Save</i></u>
Bold characters	A dialog button	Click OK
	A tab	Click the Power tab
N'Rnnnn	A number in verilog format, where N is the total number of digits, R is the radix and n is a digit.	4'b0010, 2'hF1
Text in angle brackets < >	A key on the keyboard	Press <Enter>, <F1>
Courier New font:		
Plain Courier New	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 Courier New	A variable argument	<i>file.o</i> , where <i>file</i> can be any valid filename
Square brackets []	Optional arguments	mcc18 [options] <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) { ... }

MPLAB® ICD 3 In-Circuit Debugger User's Guide

WARRANTY REGISTRATION

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

RECOMMENDED READING

This user's guide describes how to use MPLAB ICD 3 in-circuit debugger. Other useful documents are listed below. The following Microchip documents are available and recommended as supplemental reference resources.

In-Circuit Debugger Design Advisory (DS51764)

Please read this first! This document contains important information about operational issues that should be considered when using the MPLAB ICD 3 with your target design.

Release Notes for MPLAB ICD 3 In-Circuit Debugger

For the latest information on using the MPLAB ICD 3 in-circuit debugger, read the "Readme for MPLAB ICD 3 Debugger.htm" file (an HTML file) in the Readmes subdirectory of the MPLAB IDE installation directory. The release notes (Readme) contain updated information and known issues that may not be included in this user's guide.

Using MPLAB ICD 3 In-Circuit Debugger Poster (DS51765)

This poster shows you how to hook up the hardware and install the software for the MPLAB ICD 3 in-circuit debugger using standard communications and a target board.

MPLAB ICD 3 In-Circuit Debugger online Help File

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

Header Board Specification (DS51292)

This booklet describes how to install and use MPLAB ICD 3 in-circuit debugger headers. Headers are used to better debug selected devices using special -ICE device versions without the loss of pins or resources.

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. 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, 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. This includes the MPLAB REAL ICE™ and MPLAB ICE 2000 in-circuit emulators.
- **In-Circuit Debuggers** – The latest information on the Microchip in-circuit debuggers. This includes MPLAB ICD 3 in-circuit debuggers and PICKit™ 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 production programmers such as MPLAB REAL ICE in-circuit emulator, MPLAB ICD 3 in-circuit debugger and MPLAB PM3 device programmers. Also included are nonproduction development programmers such as PICSTART® Plus and PICKit 1, 2 and 3.

MPLAB® ICD 3 In-Circuit Debugger User's Guide

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>.

REVISION HISTORY

Revision A (September 2008)

This is the initial release of this document.

Revision B (July 2010)

Section 1.2 MPLAB ICD 3 In-Circuit Debugger Defined – added text regarding production programmer.

Figure 1-1 Basic Debugger System – replaced drawing with photo.

Removed Section 1.5 Device and Feature Support. This information is located in the MPLAB ICD 3 online help through MPLAB IDE.

Section 2.5.4 Debugger Powered – clarified text regarding the VDD line.

Section 2.8 Programming with the Debugger – added text regarding production programming.

Added new Section 3.4 Using the ICD 3 Test Interface Board.

Added new Section 4.8 Quick Debug vs. Program Operation.

Section 5.11 Debugging Code Using Breakpoints – added text regarding breakpoints.

Section 5.11.1 Choosing a Breakpoint Type – added text for hardware and software breakpoints.

Added new Sections 5.11.4 and 5.11.5 for ANDed and Sequenced Breakpoints.

Section 6.3 What's Wrong – added more questions/answers.

Chapter 7 Error Messages – added new section for Informational Messages; added clarifying text for some error messages.

Sections 8.2 Breakpoints and 8.3 Stopwatch – combined sections and renamed Section 8.2 to Breakpoints and Stopwatch.

Added Section 9.3.7 Application In/Out Window.

Modified Section 9.5 Settings Dialog, subsections and tables to add new information.

Added Table A-1 Electrical Logic Table.

Appendix A: ICD 3 Test Interface Board – added additional steps and information.

Glossary – added additional entries.



MPLAB® ICD 3 IN-CIRCUIT DEBUGGER USER'S GUIDE

Part 1 – Getting Started

Chapter 1. Overview.....	15
Chapter 2. Operation.....	17
Chapter 3. Installation.....	25
Chapter 4. General Setup	29
Chapter 5. Tutorial.....	33

MPLAB® ICD 3 In-Circuit Debugger User's Guide

NOTES:

Chapter 1. Overview

1.1 INTRODUCTION

An overview of the MPLAB ICD 3 in-circuit debugger system is given.

- MPLAB ICD 3 In-Circuit Debugger Defined
- How the MPLAB ICD 3 In-Circuit Debugger Helps You
- MPLAB ICD 3 In-Circuit Debugger Kit Components

1.2 MPLAB ICD 3 IN-CIRCUIT DEBUGGER DEFINED

The MPLAB ICD 3 is an in-circuit debugger that is controlled by a PC running MPLAB IDE (v8.15 or greater) software on a Windows® platform. The MPLAB ICD 3 in-circuit debugger is an integral part of the development engineer's toolsuite. The application usage can vary from software development to hardware integration.

The MPLAB ICD 3 in-circuit debugger is a complex debugger system used for hardware and software development of Microchip PIC® microcontrollers (MCUs) and dsPIC® Digital Signal Controllers (DSCs) that are based on In-Circuit Serial Programming™ (ICSP™) and Enhanced In-Circuit Serial Programming 2-wire serial interfaces.

The debugger system will execute code like an actual device because it uses a device with built-in emulation circuitry, instead of a special debugger 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 ICD 3 in-circuit debugger was developed for emulating embedded processors with rich debug facilities which differ from conventional system processors in the following aspects:

- Processors run at maximum speeds
- Capability to incorporate I/O port data input

In addition to debugger functions, the MPLAB ICD 3 in-circuit debugger system also may be used as a device production programmer.

The MPLAB ICD 3 in-circuit debugger may be used for production programming, either through MPLAB IDE or as a command-line programmer by using ICD3CMD (in the Programmer Utilities folder of the MPLAB IDE installation folder).

MPLAB® ICD 3 In-Circuit Debugger User's Guide

1.3 HOW THE MPLAB ICD 3 IN-CIRCUIT DEBUGGER HELPS YOU

The MPLAB ICD 3 in-circuit debugger system allows you to:

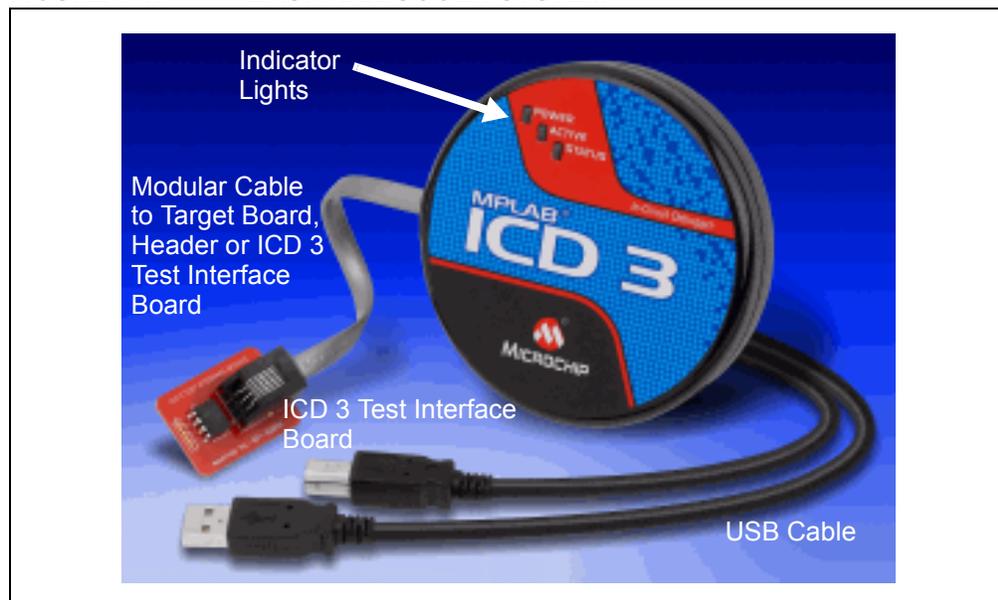
- Debug your application on your own hardware in real time
- Debug with hardware breakpoints
- Debug with software breakpoints
- Set breakpoints based on internal events
- Monitor internal file registers
- Emulate full speed
- Program your device

1.4 MPLAB ICD 3 IN-CIRCUIT DEBUGGER KIT COMPONENTS

The components of the MPLAB ICD 3 in-circuit debugger system kit are:

1. MPLAB ICD 3 with indicator lights
2. USB cable to provide communications between the debugger and a PC and to provide power to the debugger
3. Cable to connect the MPLAB ICD 3 to a header module or target board
4. ICD 3 Test Interface Board
5. MPLAB IDE Quick Start Guide (DS51281)
6. CD-ROM with MPLAB IDE software and online documentation

FIGURE 1-1: BASIC DEBUGGER SYSTEM



Additional hardware that may be ordered separately:

- Transition socket
- ICD headers
- MPLAB IDE processor extension kits

Chapter 2. Operation

2.1 INTRODUCTION

A simplified description of how the MPLAB ICD 3 in-circuit debugger system works is provided here. It is intended to provide enough information so a target board can be designed that is compatible with the debugger 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.

- MPLAB ICD 3 In-Circuit Debugger vs. MPLAB ICE 2000/4000 In-Circuit Emulators
- MPLAB ICD 3 In-Circuit Debugger vs. MPLAB ICD 2 Debugger
- Debugger To Target Communication
- Communication Connections
- Debugging with the Debugger
- Requirements For Debugging
- Programming with the Debugger
- Resources Used by the Debugger

2.2 MPLAB ICD 3 IN-CIRCUIT DEBUGGER VS. MPLAB ICE 2000/4000 IN-CIRCUIT EMULATORS

The MPLAB ICD 3 in-circuit debugger system is a next generation In-Circuit Debugger (ICD) system. It differs from classical in-circuit emulator systems (e.g., MPLAB ICE 2000/4000) in a single, but important way: the production device and emulation device are the same.

This is a great benefit since differences (errata) between the production silicon and emulation silicon are eliminated. Additionally, as devices continue to operate at faster speeds, traditional emulator systems present bottlenecks caused by internal busses that must be carried off-chip to external memories and cannot offer full speed emulation.

Another significant benefit is that there is no lead time between production silicon and emulation silicon. Further, a problem encountered on a production board can be easily debugged without having to install transition sockets and dealing with complicated cabling systems and setups to have access to the application.

2.3 MPLAB ICD 3 IN-CIRCUIT DEBUGGER VS. MPLAB ICD 2 DEBUGGER

The MPLAB ICD 3 in-circuit debugger system is similar in function to the MPLAB ICD 2 in-circuit debugger system, but surpasses it in speed and functionality. The MPLAB ICD 3 also:

- Features USB high speed
- Is USB powered
- Is a hardware accelerator
- Provides a programmable voltage power supply
- Eliminates the RS-232 port
- Includes a diagnostic self-test interface board

MPLAB® ICD 3 In-Circuit Debugger User's Guide

2.4 DEBUGGER TO TARGET COMMUNICATION

The debugger system configurations are 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.

Standard ICSP Device Communication

The debugger system can be configured to use standard ICSP communication for both programming and debugging functions. This 6-pin connection is the same one used by the MPLAB ICD 2 in-circuit debugger.

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 2-1), or (2) inserted into a standard adapter/header board combo (available as a Processor Pak), which is then plugged into the target board (Figure 2-2).

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 debugger.

For more on standard communication, see **Appendix A. "Hardware Specification"**.

FIGURE 2-1: STANDARD DEBUGGER SYSTEM – DEVICE WITH ON-BOARD ICE CIRCUITRY

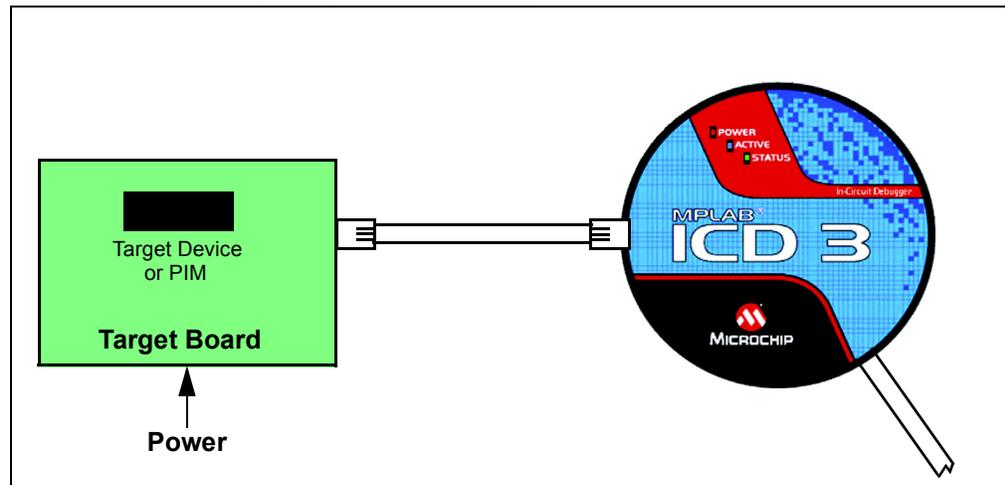
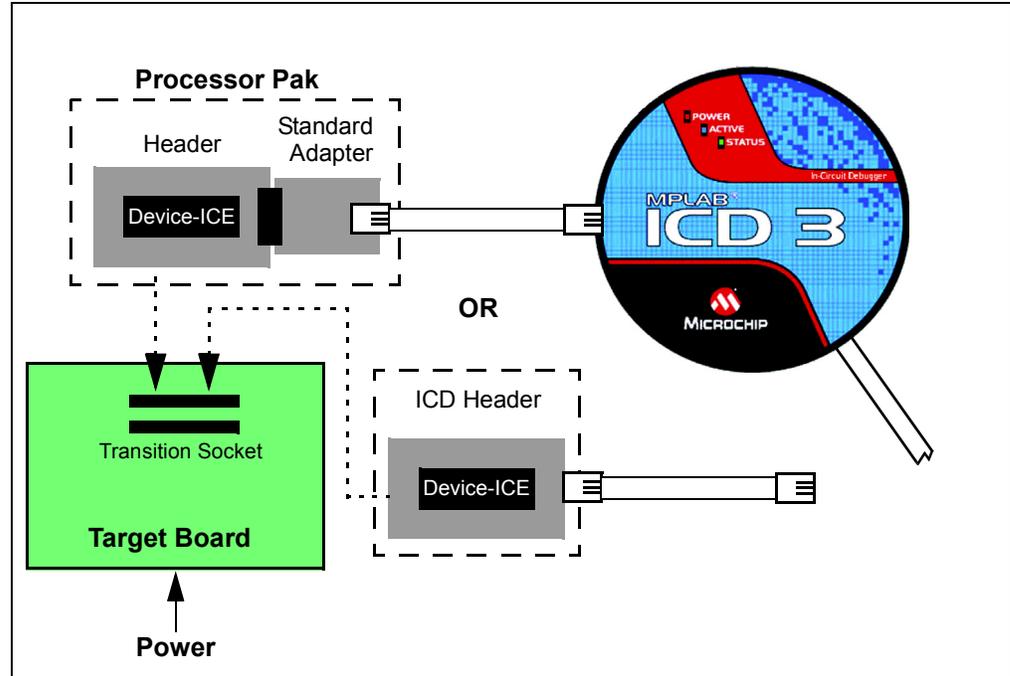


FIGURE 2-2: STANDARD DEBUGGER SYSTEM – ICE DEVICE



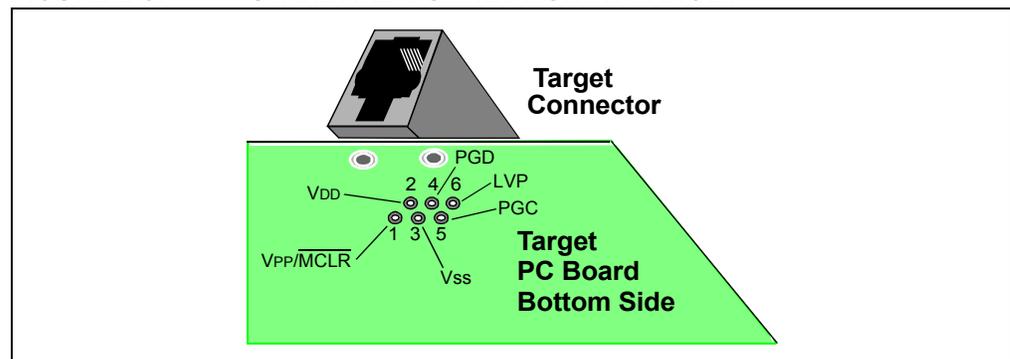
2.5 COMMUNICATION CONNECTIONS

2.5.1 Standard Communication Target Connection

Using the RJ-11 connector, the MPLAB ICD 3 in-circuit debugger 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 2-3.

Note: Cable connections at the debugger 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 A.6.2.3 “Modular Cable Specification”**.

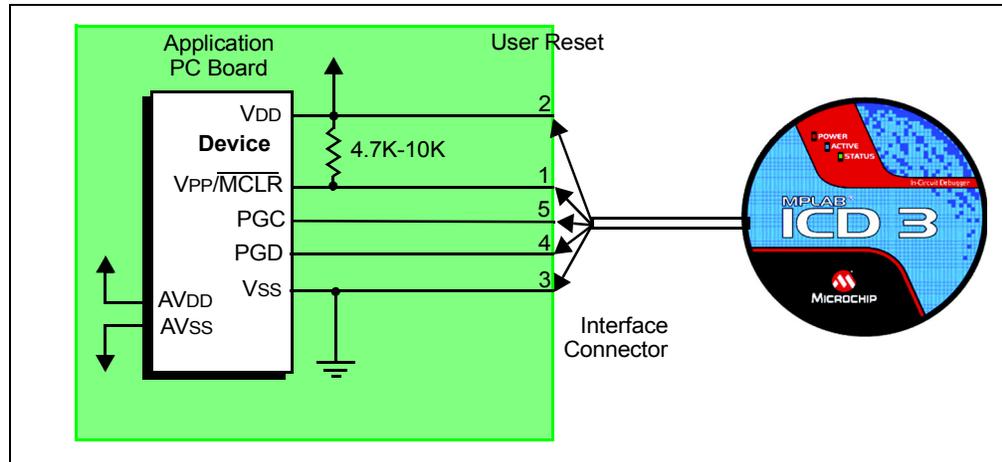
FIGURE 2-3: STANDARD CONNECTION AT TARGET



2.5.2 Target Connection Circuitry

Figure 2-4 shows the interconnections of the MPLAB ICD 3 in-circuit debugger 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 (usually around 10 k Ω) 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 2-4: STANDARD CONNECTION TARGET CIRCUITRY



2.5.3 Target Powered

In the following descriptions, only three lines are active and relevant to core debugger operation: pins 1 (VPP/MCLR), 5 (PGC) and 4 (PGD). Pins 2 (VDD) and 3 (VSS) are shown on Figure 2-4 for completeness. MPLAB ICD 3 has two configurations for powering the target device: internal debugger and external target power.

The recommended source of power is external and derived from the target application. In this configuration, target VDD is sensed by the debugger to allow level translation for the target low voltage operation. If the debugger does not sense voltage on its VDD line (pin 2 of the interface connector), it will not allow communication with the target.

2.5.4 Debugger Powered

The internal debugger power is limited in two aspects: (1) the voltage range is not as wide (3-5V); and (2) the amount of current it can supply is limited to 100 mA. This may be of benefit for very small applications that have the device VDD separated from the rest of the application circuit for independent programming, but is not recommended for general usage as it imposes more current demands from the USB power system derived from the PC.

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

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 debugger to operate.

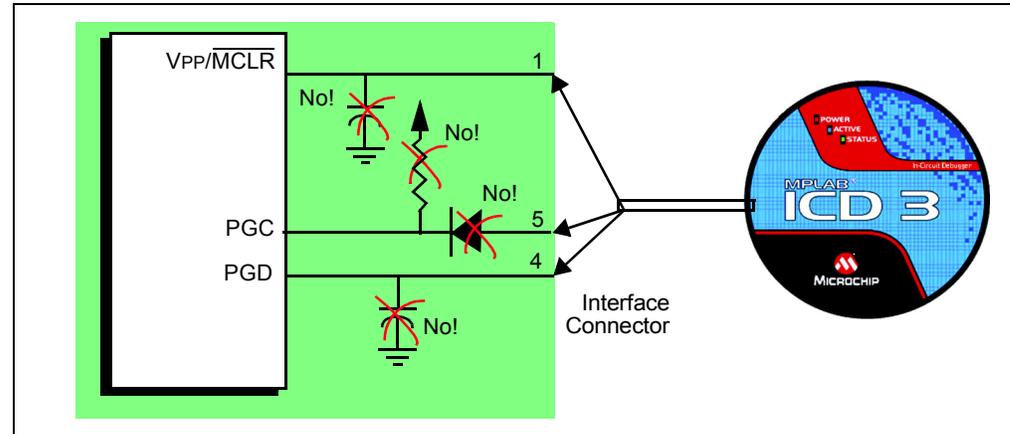
In general, it is recommended that all VDD/AVDD and VSS/AVSS lines be connected to the appropriate levels. Also, devices with a VCAP line (PIC18FXXJ for example) should be connected to the appropriate capacitor or level.

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 ICD 3 in-circuit debugger system, as discussed in the following section.

2.5.5 Circuits That Will Prevent the Debugger From Functioning

Figure 2-5 shows the active debugger lines with some components that will prevent the MPLAB ICD 3 in-circuit debugger system from functioning.

FIGURE 2-5: IMPROPER CIRCUIT COMPONENTS



Specifically, these guidelines must be followed:

- Do not use pull-ups on PGC/PGD – they will disrupt the voltage levels, since these lines have 4.7 kΩ pull-down resistors in the debugger.
- 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 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 debugger and the target device.

2.6 DEBUGGING WITH THE DEBUGGER

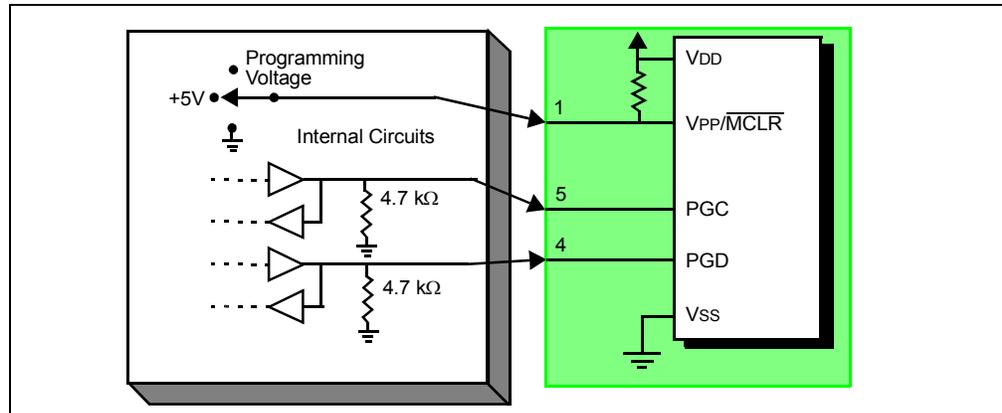
There are two steps to using the MPLAB ICD 3 in-circuit debugger system as a debugger. The first requires that an application be programmed into the target device. The second uses the internal in-circuit debug hardware of the target Flash device to run and test the application program. These two steps 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 debugger to set breakpoints and run.

If the target device cannot be programmed correctly, the MPLAB ICD 3 in-circuit debugger will not be able to debug.

Figure 2-6 shows the basic interconnections required for programming. Note that this is the same as Figure 2-4, but for the sake of clarity, the VDD and VSS lines from the debugger are not shown.

FIGURE 2-6: PROPER CONNECTIONS FOR PROGRAMMING



A simplified diagram of some of the internal interface circuitry of the MPLAB ICD 3 in-circuit debugger is shown. For programming, no clock is needed on the target device, but power must be supplied. When programming, the debugger puts programming levels on VPP/MCLR, 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 ICD 3 in-circuit debugger system, there are critical elements that must be working correctly:

- The debugger must be connected to a PC. It must be powered by the PC via the USB cable, and it must be communicating with the MPLAB IDE software via the USB cable. See **Chapter 3. "Installation"** for details.
- The debugger 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 debugger and target device.
- The target device must have power and a functional, running oscillator. If the target device does not run, for any reason, the MPLAB ICD 3 in-circuit debugger 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.
- LVP should be disabled.

Once the above conditions are met, you may proceed to the following:

- Sequence of Operations Leading to Debugging
- Debugging Details

2.7.1 Sequence of Operations Leading to Debugging

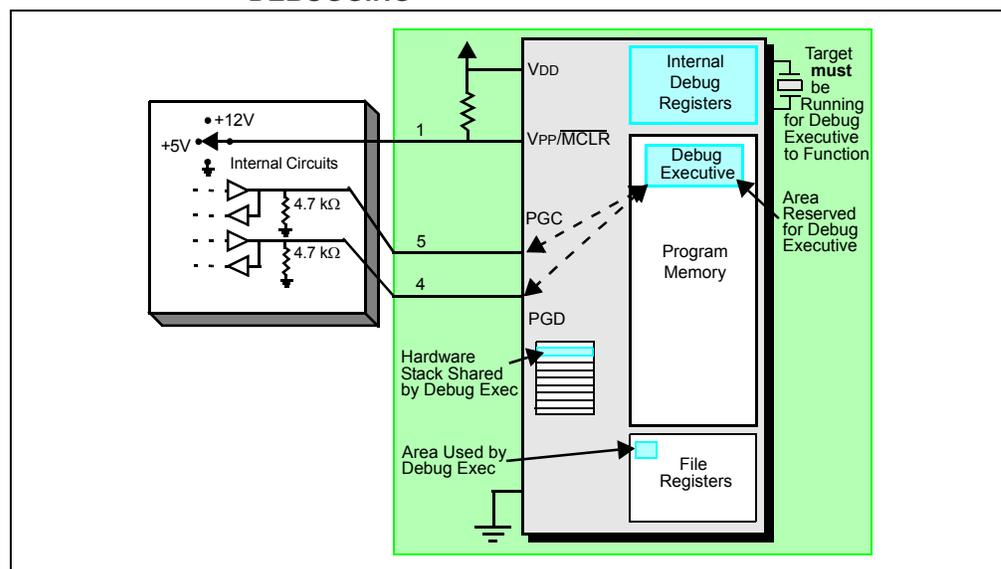
Given that the Requirements For Debugging are met, these actions can be performed when the MPLAB ICD 3 in-circuit debugger is set as the current debugger from the MPLAB IDE menu (*Debugger>Select Tool>MPLAB ICD 3*):

- The application code is compiled/assembled 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. These allow the debug executive to be activated by the debugger.
- The target device is held in Reset by keeping the VPP/MCLR line low.

2.7.2 Debugging Details

Figure 2-7 illustrates the MPLAB ICD 3 in-circuit debugger system when it is ready for debugging.

FIGURE 2-7: MPLAB® ICD 3 IN-CIRCUIT DEBUGGER 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 debugger 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.

MPLAB® ICD 3 In-Circuit Debugger User's Guide

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 debugger 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 debugger 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 ICD 3 in-circuit debugger 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 debugger communications with the debug executive to interrogate the state of the target device.

2.8 PROGRAMMING WITH THE DEBUGGER

Use the MPLAB ICD 3 in-circuit debugger as a programmer to program an actual (non -ICE/-ICD) device, i.e., a device not on a header board. Select “MPLAB ICD 3” 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 debugger 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 ICD 3 in-circuit debugger 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 debugger 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 ICD 3 in-circuit debugger 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 ICD 3 in-circuit debugger may be used for production programming, either through MPLAB IDE or as a command-line programmer by using ICD3CMD (in the Programmer Utilities folder of the MPLAB IDE installation folder.)

2.9 RESOURCES USED BY THE DEBUGGER

For a complete list of resources used by the debugger for your device, please see the online help file in MPLAB IDE for the MPLAB ICD 3 in-circuit debugger.

Chapter 3. Installation

3.1 INTRODUCTION

How to install the MPLAB ICD 3 in-circuit debugger system is discussed.

- Installing the Software
- Installing the USB Device Drivers
- Using the ICD 3 Test Interface Board
- Connecting the Target
- Setting Up the Target Board
- Setting Up MPLAB IDE

3.2 INSTALLING THE SOFTWARE

To install the MPLAB IDE software, first acquire the latest MPLAB IDE installation executable (`MPxxxxxx.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.

Note: MPLAB IDE v8.15 or greater is required to use the MPLAB ICD 3 in-circuit debugger.

3.3 INSTALLING THE USB DEVICE DRIVERS

Installing MPLAB IDE will preinstall the USB device drivers for the MPLAB ICD 3 in-circuit debugger. Therefore, once you have installed MPLAB IDE, connect the debugger 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\ICD 3\Drivers\ddri.htm`

Note: If a new MPLAB ICD 3 is connected to your PC, you will need to reinstall the drivers for the new unit.

3.4 USING THE ICD 3 TEST INTERFACE BOARD

Use the ICD 3 Test Interface Board to verify that the MPLAB ICD 3 is functioning properly:

1. Connect the ICD 3 Test Interface board to the MPLAB ICD 3 using the modular cable supplied.
2. Connect the debugger to the PC.
3. Select “MPLAB ICD 3” on either the Debugger or Programmer menu in MPLAB IDE. From that menu, select “Settings”, **Status** tab, then click the **Run ICD 3 Test Interface** option. A reminder to have the test board connected will pop up. Click **OK**. The status (pass/fail) is displayed in the Output window.

MPLAB® ICD 3 In-Circuit Debugger User's Guide

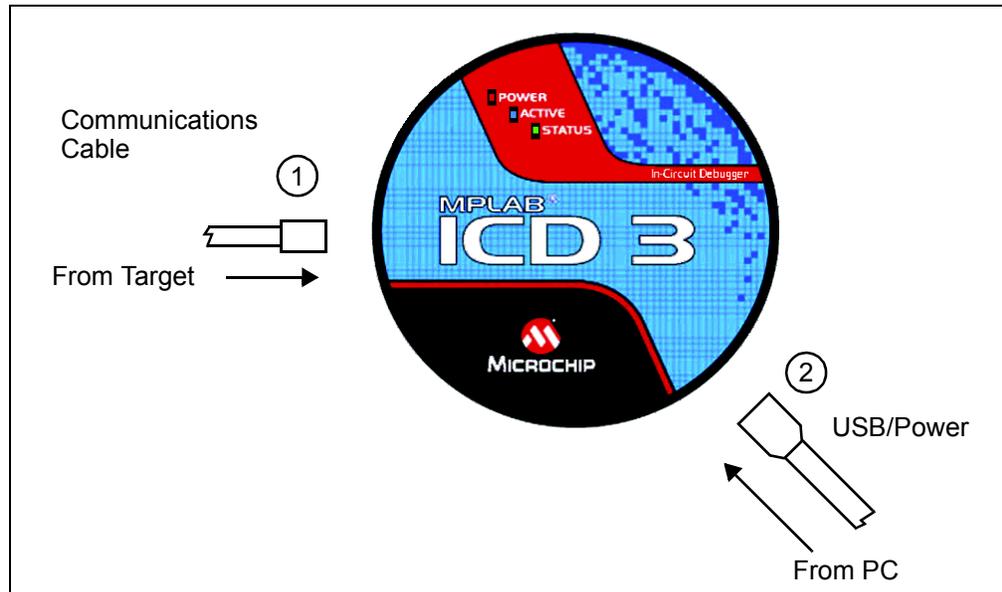
If the debugger passes, your MPLAB ICD 3 is functioning properly. You can disconnect the ICD 3 Test Interface Board and continue with the next section. If the debugger fails, unplug and plug in the debugger. Reconnect to the debugger in MPLAB IDE. If the problem persists contact Microchip.

3.5 CONNECTING THE TARGET

A connection is built-in to select the type of communication with the target. See **Section 2.4 “Debugger To Target Communication”** for more details and a diagram.

1. Plug in the USB/power cable if not already connected.
2. Attach the communication cable(s) between debugger and target.

FIGURE 3-1: INSERT COMMUNICATIONS AND USB/POWER CABLES



3.6 SETTING UP THE TARGET BOARD

The target must be set up for the type of target device to be used.

3.6.1 Using Production Devices

For production devices, the debugger 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 ICD 3 in-circuit debugger 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.

The target board must have a connector to accommodate the communications chosen for the debugger. For connection information, see **Section 2.4 “Debugger To Target Communication”**, “Standard ICSP Device Communication.”

3.6.2 Using ICE Devices

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. For more information on ICE headers, see the “*Header Board Specification*” (DS51292).

Note: In the future, ICD header boards with ICD devices (*Device-ICD*) may be used.

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 headers or processor extension packs. For connection information, see **Section 2.4 “Debugger To Target Communication”**, “Standard ICSP Device Communication.”

3.6.3 Powering the Target

There are a couple of configurations for powering MPLAB ICD 3 and the target.

These are configuration essentials:

- When using the USB connection, MPLAB ICD 3 can be powered from the PC but it can only provide a limited amount of current, up to 100 mA, at VDD from 3-5V to a small target board.
- The desired method is for the target to provide VDD as it can provide a wider voltage range from 2-5V. The additional benefit is that plug-and-play target detection facility is inherited, i.e., MPLAB IDE will let you know in the Output window when it has detected the target and has detected the device.

Note: The target voltage is only used for powering up the drivers for the ICSP interface; the target voltage does not power-up the MPLAB ICD 3. The MPLAB ICD 3 system power is derived strictly from the USB port.

If you have not already done so, connect the MPLAB ICD 3 to the target using the appropriate cables (see **Section 3.5 “Connecting the Target”**). Then power the target. If you are powering the target through the MPLAB ICD 3, see **Section 9.5.7 “Settings Dialog, Power Tab”** for instructions.

3.7 SETTING UP MPLAB IDE

Once the hardware is connected and powered, MPLAB IDE may be set up for use with the MPLAB ICD 3 in-circuit debugger.

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 a project and getting started with MPLAB ICD 3, see **Chapter 4. “General Setup”**.

To walk through the process of programming and debugging a device with the MPLAB ICD 3, see **Chapter 5. “Tutorial”**.

MPLAB® ICD 3 In-Circuit Debugger User's Guide

NOTES:

Chapter 4. General Setup

4.1 INTRODUCTION

How to get started using the MPLAB ICD 3 in-circuit debugger is discussed.

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

4.2 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 online help files – Contains the most up-to-date information on MPLAB IDE and MPLAB ICD 3 in-circuit debugger.
- Readme files – Last minute information on each release is included in `Readme for MPLAB IDE.txt` and `Readme for MPLAB ICD 3 Debugger.txt`. Both files are found in the `Readmes` subdirectory of the MPLAB IDE installation directory.

4.3 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`)

Note: If you do not have a custom linker script in your project, the Project Manager will select the appropriate linker script for you.

4.4 VIEWING THE PROJECT

After the Project Wizard has created a project, the project and its associated files are visible in the 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.

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

4.5 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 ICD 3 in-circuit debugger.

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

Note: On the Project Manager toolbar (*View>Toolbars>Project Manager*), select “Debug” from the drop-down list when using the MPLAB ICD 3 as a debugger, or select “Release” when using it as a programmer.

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

4.6 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:

- **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., PG1/EMUC1 and PG1/EMUD1. Make sure the pins selected here are the same ones physically connected to the device.
- **Oscillator** – Select the configuration setting that matches the target oscillator.

4.7 SETTING THE DEBUGGER AS THE DEBUGGER OR PROGRAMMER

Select *Debugger>Select Tool>MPLAB ICD 3* to choose the MPLAB ICD 3 in-circuit debugger 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 MPLAB ICD 3 status and communications will be displayed on the **MPLAB ICD 3** tab. For more information, see **Section 9.2 “Debugging Functions”** and **Section 9.3 “Debugging Dialogs/Windows”**.

Select *Programmer>Select Programmer>MPLAB ICD 3* to choose the MPLAB ICD 3 in-circuit debugger 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 ICD 3** tab. For more information, see **Section 9.4 “Programming Functions”**.

Select *Debugger>Settings* or *Programmer>Settings* to open the Settings dialog (**Section 9.5 “Settings Dialog”**) and set up options as needed.

If errors occur, see:

- **Chapter 7. “Error Messages”**
- **Chapter 6. “Frequently Asked Questions (FAQs)”**
- **Section A.7 “ICD 3 Test Interface Board”**

4.8 QUICK DEBUG/PROGRAM REFERENCE

The following table is a quick reference for using the MPLAB ICD 3 as either a debug or program tool. Please see previous chapters for information on proper emulator setup and configuration.

TABLE 4-1: DEBUG VS. PROGRAM OPERATION

Item	Debug	Program
Needed Hardware	A PC and target application (Microchip demo board or your own design.)	
	Debugger pod, USB cable, communication cable.	
	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>Select Tool>MPLAB ICD 3</i>	<i>Programmer>Select Programmer>MPLAB ICD 3</i>
	“Debug” from the Build Configuration toolbar.	“Release” from the Build Configuration toolbar.
Linker script (optional) See MPLAB IDE documentation to determine if you need to add a linker script to your project.	Use the “i” version of the linker script, e.g., 18F452i.lkr.	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. 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.
SQTP	N/A	Use the MPLAB PM3 to generate the SQTP file. Then use the debugger to program the device.
Debug features available	All for device – breakpoints, etc.	N/A.

4.9 DEBUGGER/PROGRAMMER LIMITATIONS

For a complete list of debugger limitations for your device, please see the MPLAB ICD 3 online help file in MPLAB IDE by selecting *Help>Topics>MPLAB ICD 3* and click **OK**.

Chapter 5. Tutorial

5.1 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 Timer1 (`timer.c`), is displayed via PORTA on the demo board's LEDs.

Topics covered in this chapter:

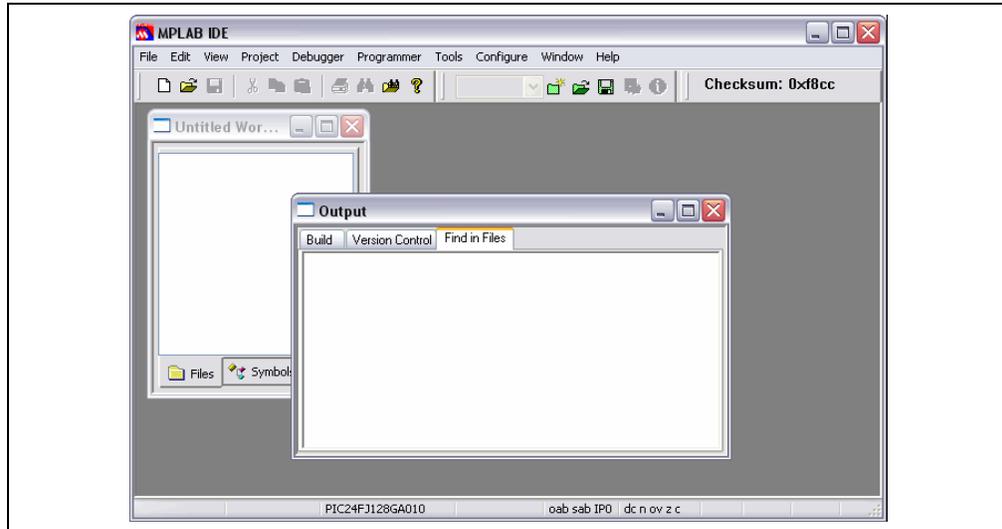
- Setting Up the Environment and Selecting the Device
- Creating the Application Code
- Running the Project Wizard
- Viewing the Project
- Viewing Debug Options
- Creating a Hex File
- Setting Up the Demo Board
- Loading Program Code For Debugging
- Running Debug Code
- Debugging Code Using Breakpoints
- Programming the Application

MPLAB® ICD 3 In-Circuit Debugger User's Guide

5.2 SETTING UP THE ENVIRONMENT AND SELECTING THE DEVICE

Before beginning this tutorial, follow the steps in **Chapter 3. “Installation”** to set up the MPLAB IDE software and MPLAB ICD 3 system hardware. Double click on the MPLAB IDE icon to launch the application. Once launched, the MPLAB IDE desktop should appear (see Figure 5-1).

FIGURE 5-1: MPLAB® IDE DESKTOP



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 ICD 3” in the “Microchip Tool Programmer/Debugger Tool Support” sections should be green.
3. Click **OK**.

5.3 CREATING THE APPLICATION CODE

For this tutorial, two C programs (`counter.c` and `timer.c`) will be used. The code for each is shown below.

1. Using Windows® Explorer, create a project folder and directory, for example, `C:\Projects\ICD3Tut`.
2. Open an Editor window by selecting **File>New**. Enter the code for the first program (see text for `counter.c`) in this window and save to the `project\directory` folder.
3. Open another Editor window by selecting **File>New**. Enter the code for the second program (see text for `timer.c`) in this window and save to the `project\directory` folder.

counter.c

```
/*
 * MPLAB ICD 3 In-Circuit Debugger 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 )

void TimerInit(void);
unsigned char TimerIsOverflowEvent(void);

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

int main(void)
{
    // Set up PortA IOs 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()...
```

MPLAB® ICD 3 In-Circuit Debugger User's Guide

timer.c

```
/*
 * MPLAB ICD 3 In-Circuit Debugger Tutorial
 * Timer program
 *
 * Demo Board:      Explorer 16
 * Processor:       PIC24FJ128GA010
 * Compiler:        MPLAB C30
 * Linker:          MPLAB LINK30
 * Company:         Microchip Technology Incorporated
 *
 */
#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
 *****/

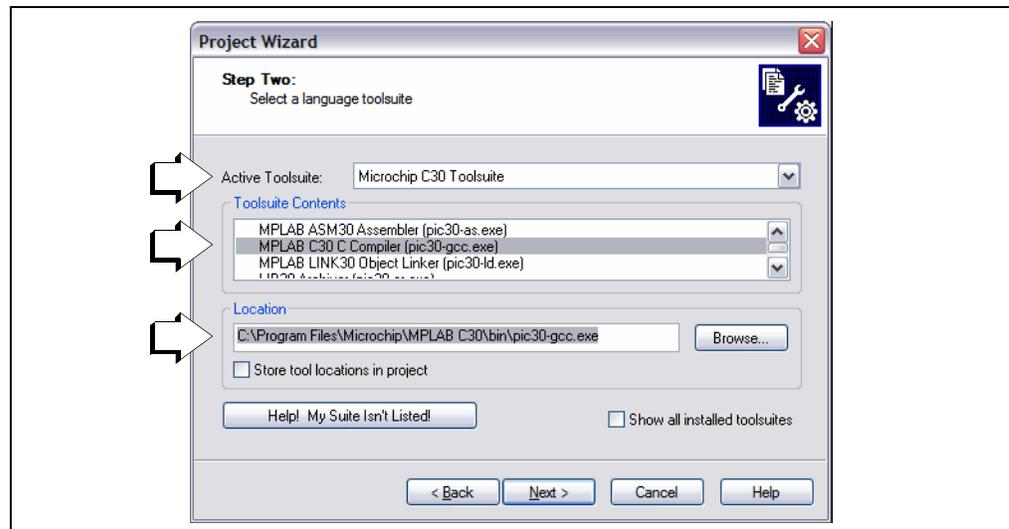
```

5.4 RUNNING THE PROJECT WIZARD

The MPLAB C30 C compiler will be used in this project. You may either purchase the full compiler or download a free student version from the Microchip web site.

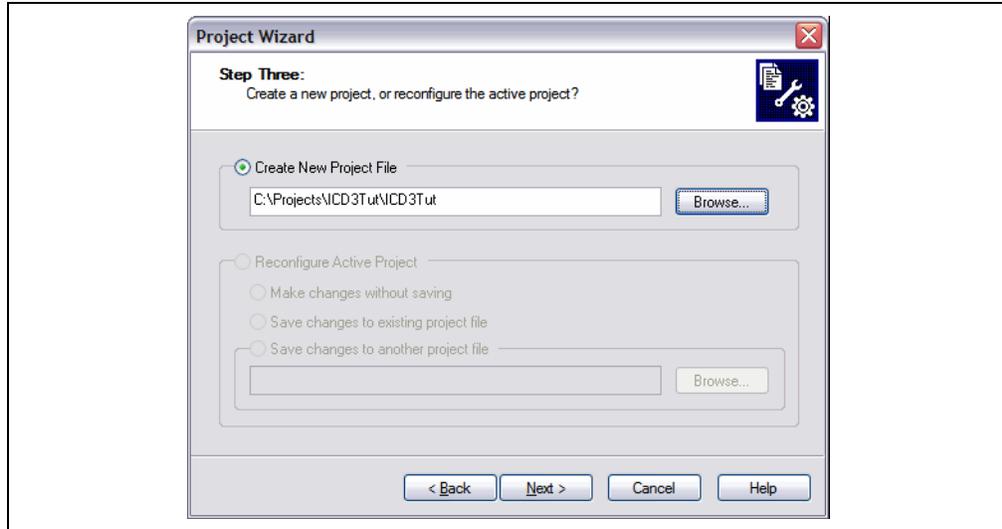
1. To set up this project, select *Project>Project Wizard*. A Welcome screen will appear.
2. Proceed to the second dialog of the wizard. The PIC24FJ128GA010 should be selected.
3. Proceed to the next dialog (Figure 5-2) of the wizard to set up the language tools. In the “Active Toolsuite” pull-down, 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 5-2: PROJECT WIZARD – TOOLSUITE SELECTION



4. Proceed to the next dialog (Figure 5-3) of the wizard to give a name and location to your project. You may **Browse** to find a location.

FIGURE 5-3: PROJECT WIZARD – PROJECT NAME

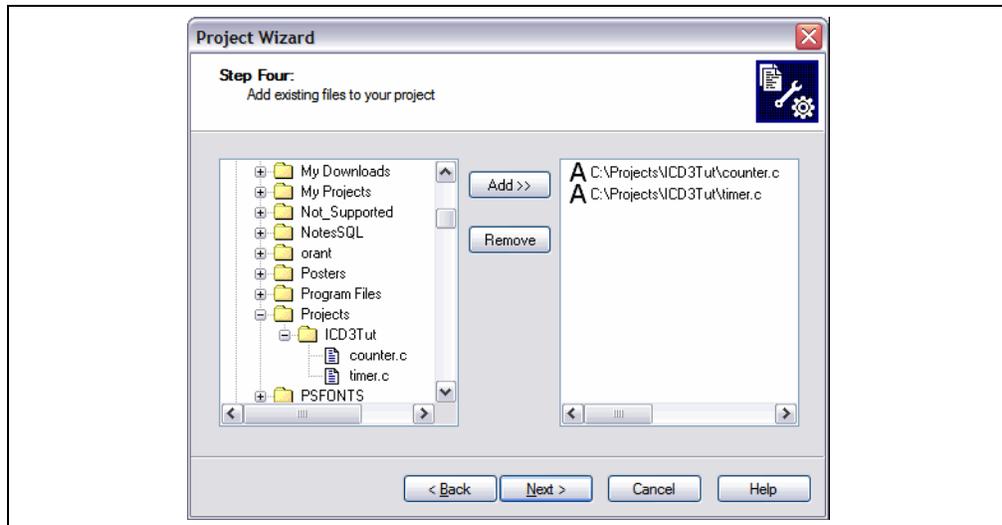


5. Proceed to the next dialog (Figure 5-4) of the wizard where project files can be added. Files can also be added later if something is missed.

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 5-4: PROJECT WIZARD – ADD FILES

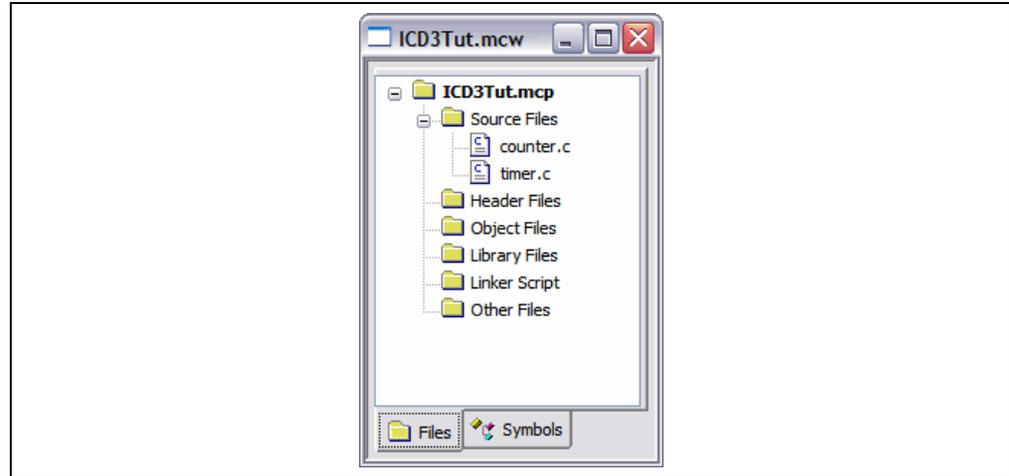


6. Proceed to the Summary screen. If you have made any errors, click **<Back** to return to a previous wizard dialog. If everything is correct, click **Finish**.

5.5 VIEWING THE PROJECT

After exiting the wizard, the MPLAB IDE desktop will again be visible. If the Project window is not open, select *View>Project* to see the Project window (Figure 5-5).

FIGURE 5-5: 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.

5.6 VIEWING DEBUG OPTIONS

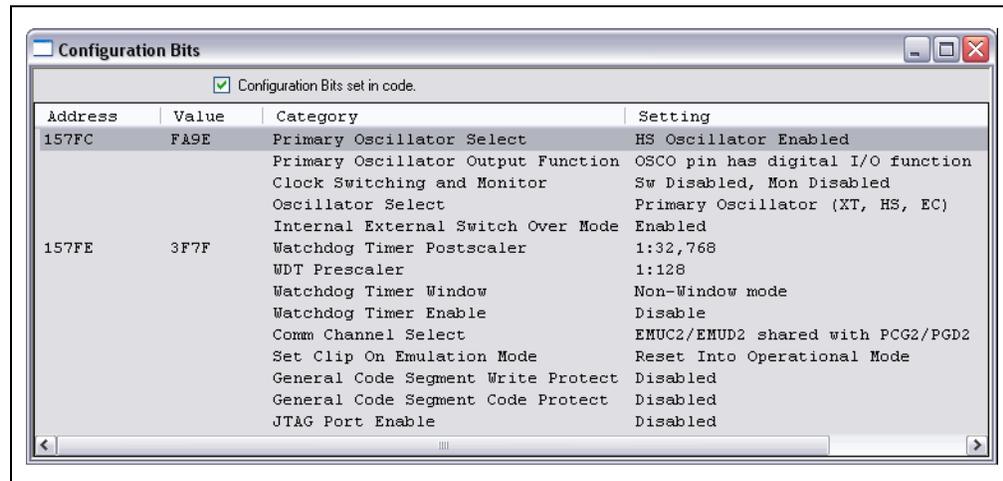
Before you begin debugging your code, review the default settings of several items. In your own projects, you may need to set these items differently.

5.6.1 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” (see Figure 5-6). *Do not change any values for this tutorial.*

FIGURE 5-6: CONFIGURATION BITS WINDOW



5.6.2 Selecting the Debugger as a Debugger

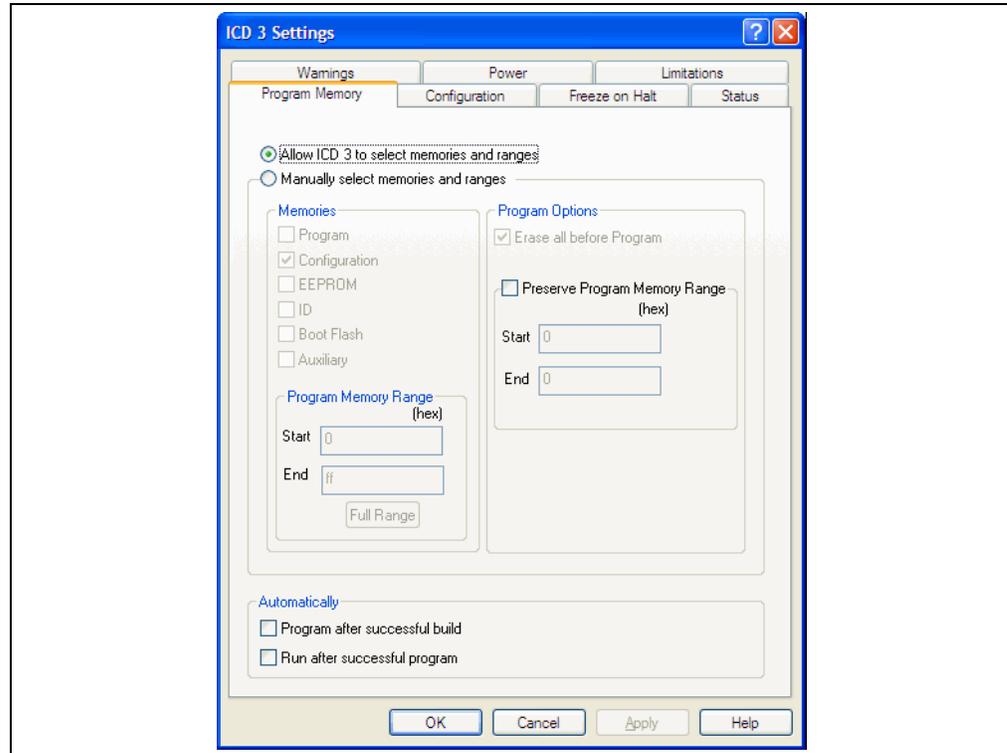
To select MPLAB ICD 3 in-circuit debugger as a debugger, select Debugger>Select Tool>ICD 3. Then:

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. Select **OK** in the message box to allow MPLAB IDE to install the new firmware. Allow several seconds for the firmware to update. Also, since different MPLAB ICD 3 firmware is used for different families of devices, this message box may appear when switching to a different device.
2. The Output window will display information about the firmware update and will show when the MPLAB ICD 3 is connected to the target.
3. The Debugger menu will show available debugger debug options.
4. A Debug toolbar will appear. Mouse over a button to see a pop-up of its function.

5.6.3 Programming Options

To set program options, select *Debugger>Settings* and click on the **Program Memory** tab (see Figure 5-7).

FIGURE 5-7: DEBUGGER PROGRAM MEMORY TAB



Here you may allow the debugger 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 ICD 3 in-circuit debugger 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 check boxes “Program after successful build” and “Run after successful program”. Leave these unchecked for now.

5.7 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.

MPLAB® ICD 3 In-Circuit Debugger User's Guide

The project will build (Figure 5-8), and the resulting .hex file will have the same name as the project (Figure 5-9). 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 5-8 (*Project>Build Options>Project, MPLAB C30 and MPLAB LINK30 tabs*).

FIGURE 5-8: OUTPUT WINDOW

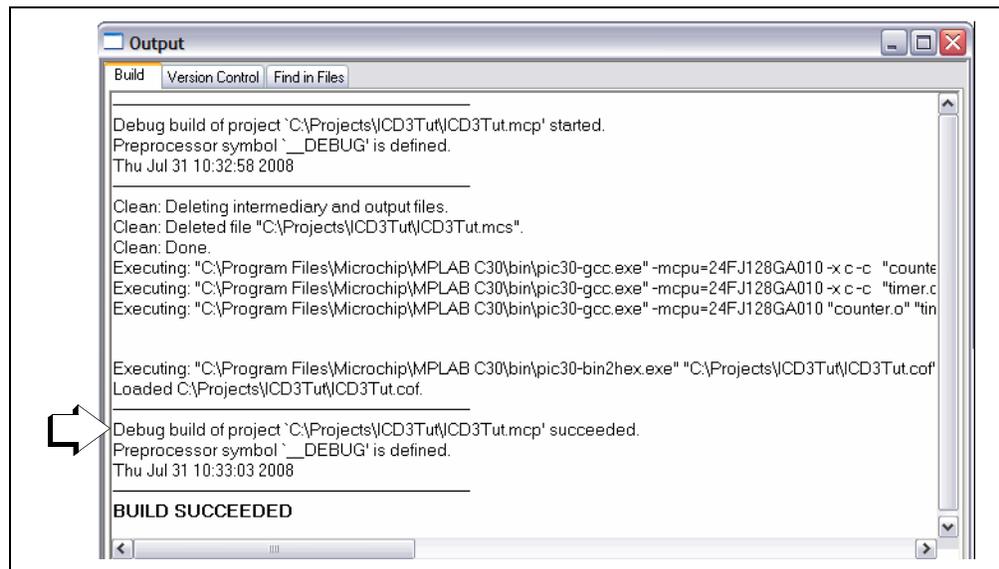
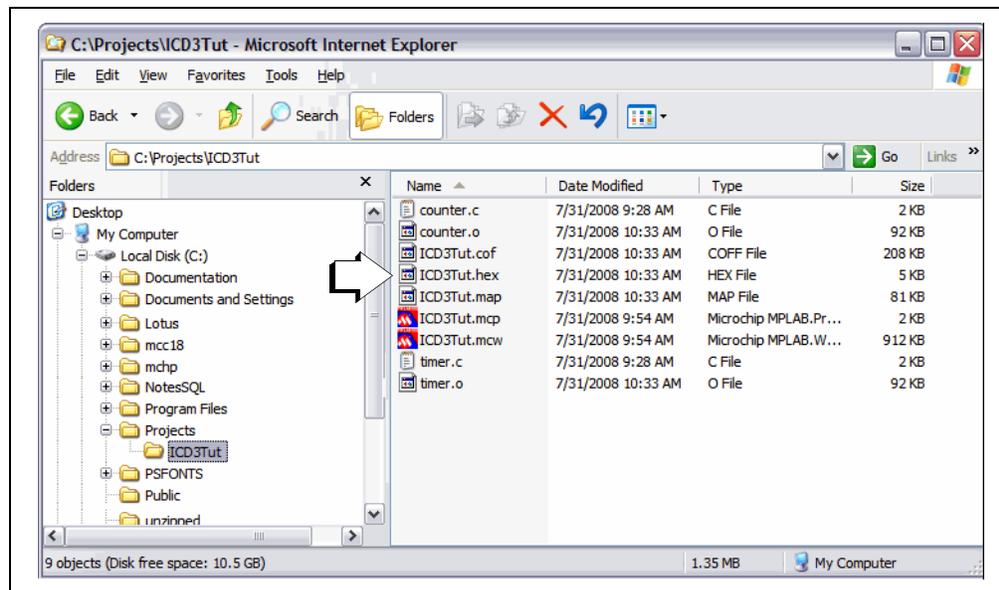


FIGURE 5-9: WINDOWS EXPLORER – PROJECT FILES



5.8 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).

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 debugger 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.

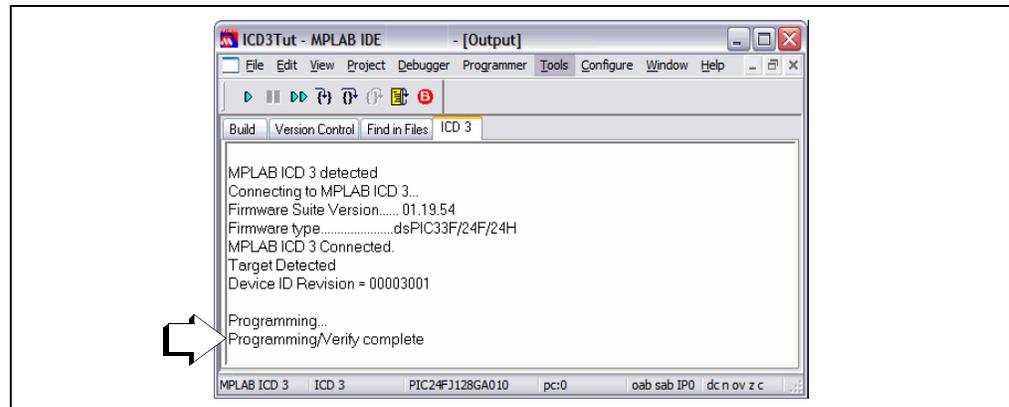
5.9 LOADING PROGRAM CODE FOR DEBUGGING

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 ICD 3 debug functions. Debug code must be programmed into the target device to use the in-circuit debugging capabilities of the MPLAB ICD 3 in-circuit debugger.

During programming, the **ICD 3** tab of the Output dialog shows the current phase of operation. When programming is complete, the dialog should look similar to Figure 5-10.

FIGURE 5-10: OUTPUT WINDOW – MPLAB® ICD 3 TAB



Note: If you have trouble programming your device or communicating with the debugger, unplug the Explorer 16 board and use the self-test board (**Section A.7 “ICD 3 Test Interface Board”**) to verify communications. For additional help, see **Chapter 6. “Frequently Asked Questions (FAQs)”**.

MPLAB® ICD 3 In-Circuit Debugger User's Guide

5.10 RUNNING DEBUG CODE

The MPLAB ICD 3 in-circuit debugger 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	Break-points
Debugger Toolbar								

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 debugger 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.

5.11 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 ICD 3 in-circuit debugger. The first debug feature to be discussed are breakpoints. Breakpoints stop code execution at a selected line of code. Breakpoints allow you to specify conditional program halts so that you may observe memory, register or variable values after a run-time execution. You may set breakpoints in either the file (editor) window, the program memory window or the disassembly window.

- Setting Software Breakpoints

5.11.1 Choosing a Breakpoint Type

The device used in this tutorial has the capability of using either hardware or software breakpoints:

- Hardware Breakpoint – An event whose execution will cause a Halt as a result of a Program Counter match and special logic inside the device.
- Software Breakpoint – An address where execution of the firmware will halt. Usually achieved by a special break instruction.

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).

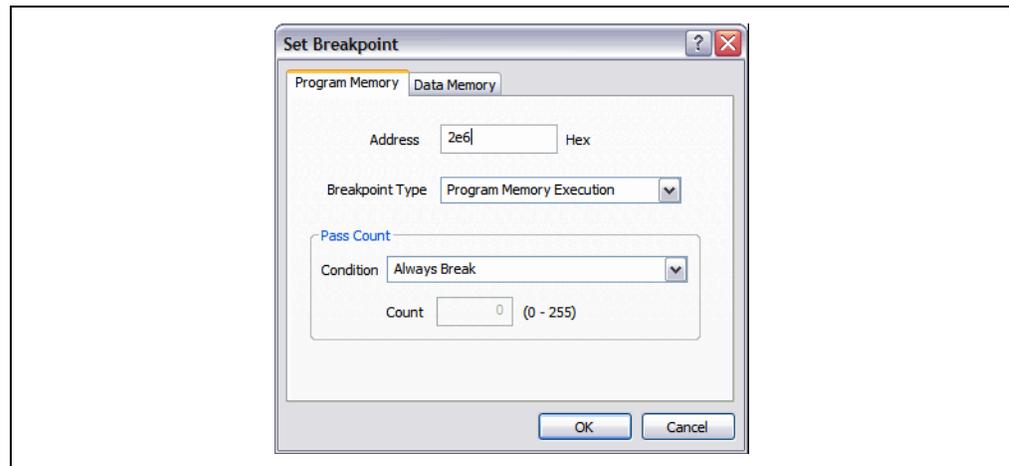
5.11.2 Setting a Single Hardware Breakpoint

To set a single hardware 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`:

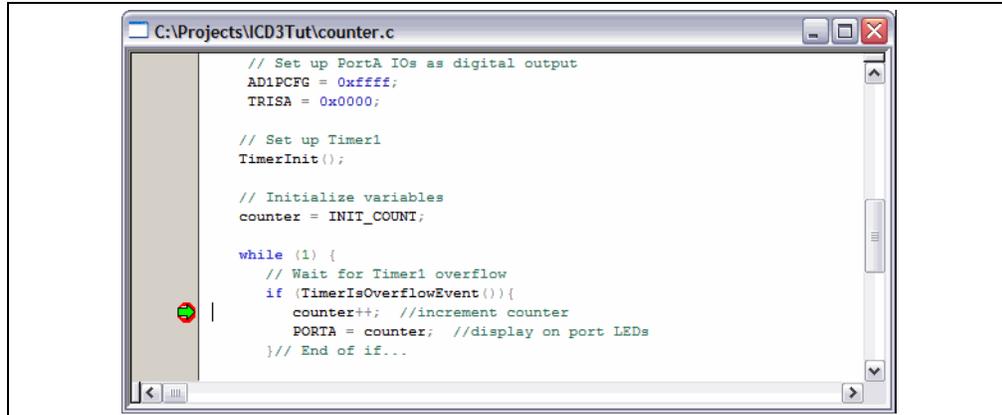
```
counter++; //increment counter
```
3. Double click on the line, or right click on the line and then select *Set Breakpoint* from the shortcut menu. This line is now marked as a breakpoint (B in red stop sign) as shown in Figure 5-11.

FIGURE 5-11: SET BREAKPOINT



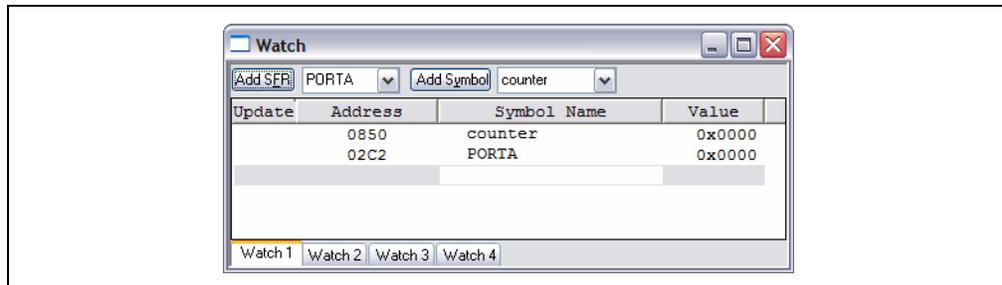
4. Select *Debugger>Run* (or click the **Run** toolbar button) to run the program once again in Real-Time mode. The program will halt at the line marked by the breakpoint, but now there will be a green arrow over the breakpoint symbol (see Figure 5-12).

FIGURE 5-12: PROGRAM HALTED



5. 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 5-13.

FIGURE 5-13: WATCH WINDOW



6. 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.
7. 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.

5.11.3 Setting Multiple Hardware Breakpoints

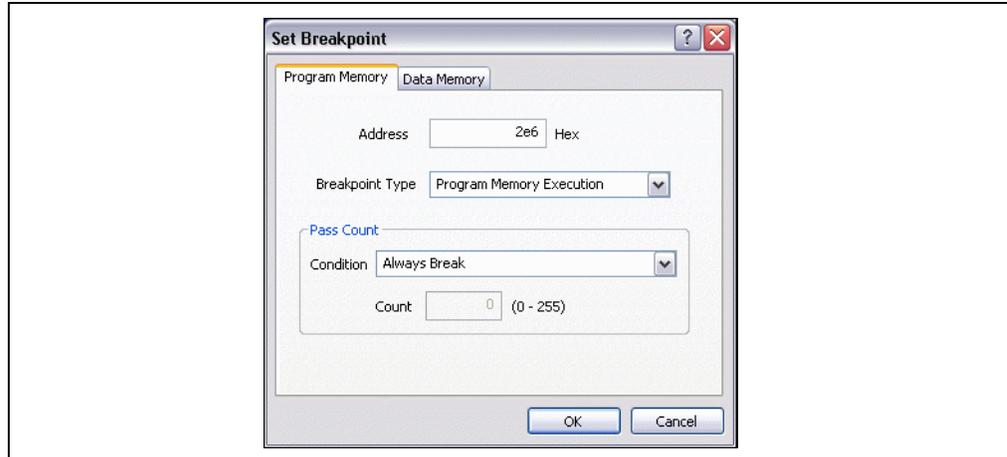
To set multiple breakpoints, either set numerous single breakpoints as specified in the previous section, or use the Breakpoints dialog (see **Section 9.3.1 “Breakpoints Dialog”**). The Breakpoints dialog also allows you to control breakpoint interaction such as sequenced or ANDed breakpoints.

Note: If you exceed the maximum allowed number of breakpoints for your device, MPLAB IDE will warn you. You can then enable software breakpoints.

For this example, place a breakpoint at the `counter++` source line in the `while` loop.

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 “2e6” as the hex address and click **OK**. (See Figure 5-14.)

FIGURE 5-14: SET BREAKPOINTS DIALOG

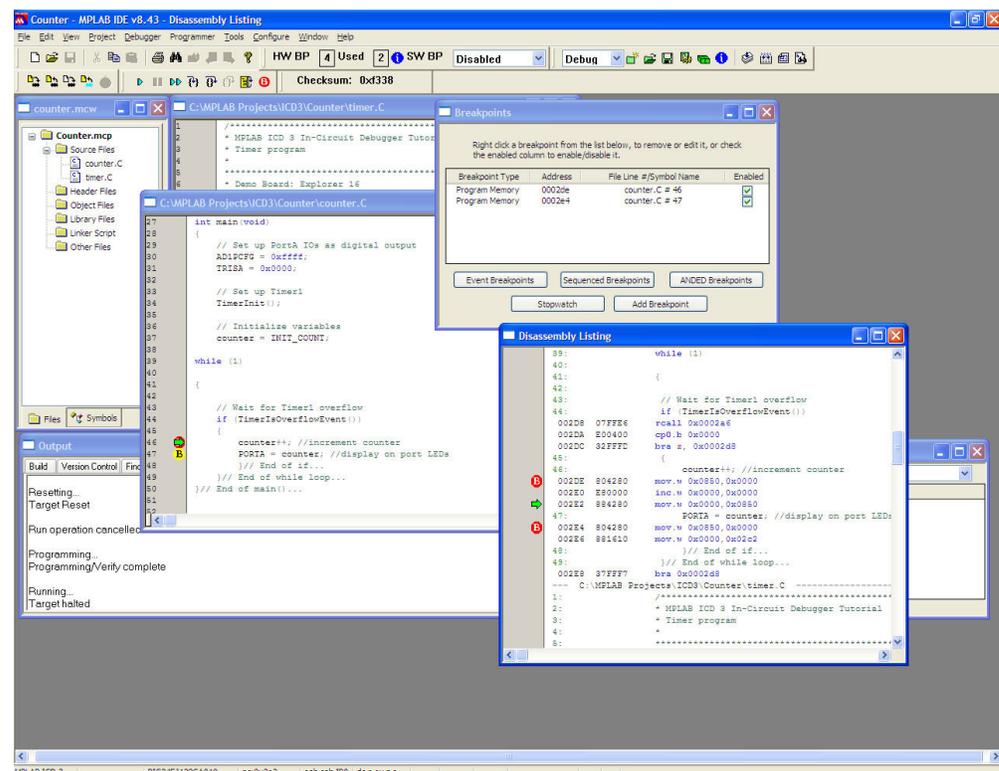


The additional breakpoint will appear below the previous breakpoint in the Breakpoints dialog (Figure 5-15) and also as a breakpoint symbol next to the following line of code:

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

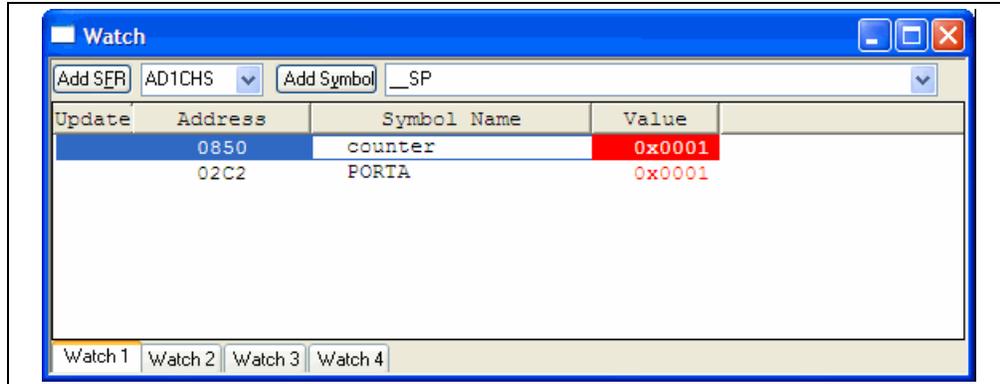
The breakpoint symbol is yellow in this case because it was set based on an address.

FIGURE 5-15: TWO BREAKPOINTS



- 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 (Figure 5-16).

FIGURE 5-16: WATCH WINDOW

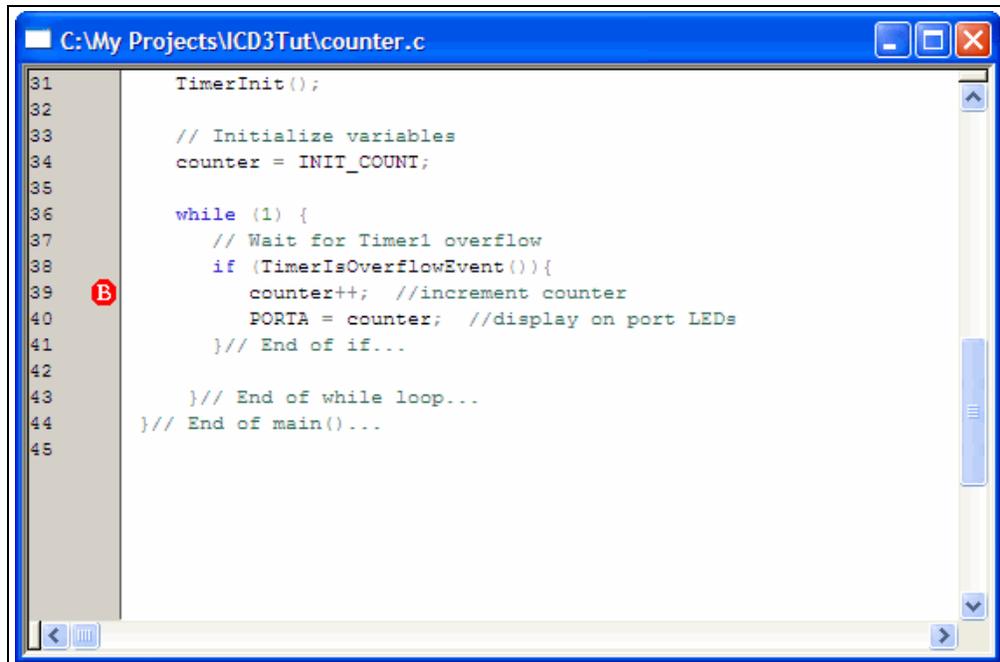


5.11.4 ANDED Breakpoints

ANDED breakpoints allow you to qualify two events at exactly the same bus cycle. In some cases, you may need to qualify an instruction execution cycle with its data counterpart, such as a write to a memory location. In the example in Figure 5-17, `counter` is written once outside of the `while` loop, but you may want to qualify it with a write inside the `while` loop when the `counter` variable has a specific value or simply when it occurs within the loop.

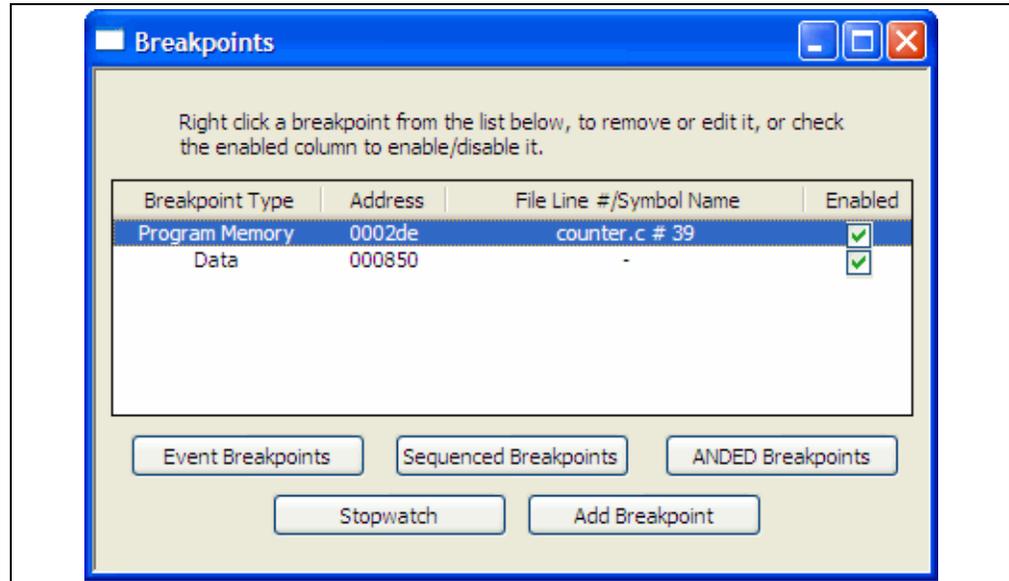
To set a breakpoint on the variable `counter` and to qualify it as an ANDED breakpoint, first double click on the `counter++` as shown in Figure 5-17. This places a program memory breakpoint at the program address corresponding to the C statement “`counter++`.”

FIGURE 5-17: SET ONE BREAKPOINT AT COUNTER VARIABLE



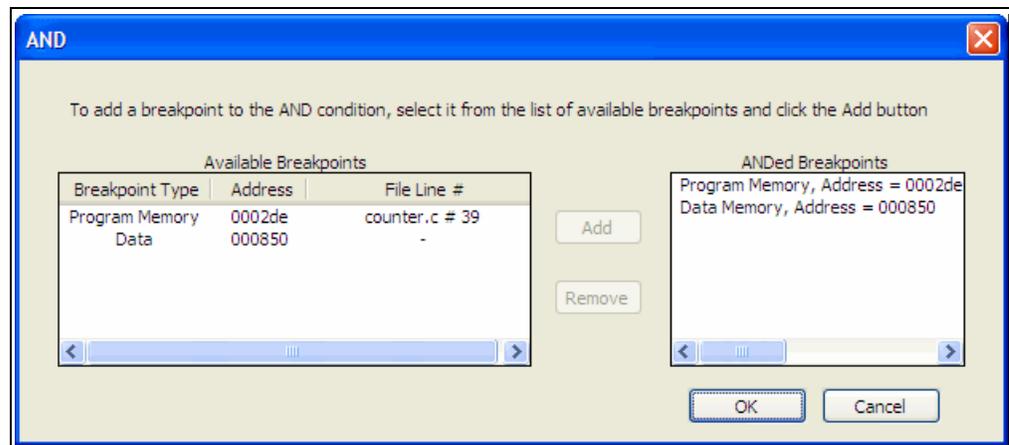
Then, add the second data breakpoint for the `counter` variable at the register file address 0x850 (see Figure 5-18). Notice that the value 0x850 corresponds to the `counter` variable.

FIGURE 5-18: SET SECOND BREAKPOINT



From the Available Breakpoints on the left, select and click **Add** for each one so it shows up in the ANDed Breakpoints box on the right side (see Figure 5-19).

FIGURE 5-19: ANDed BREAKPOINTS



Reset and **Run** the program. Notice that the breakpoint did not occur.

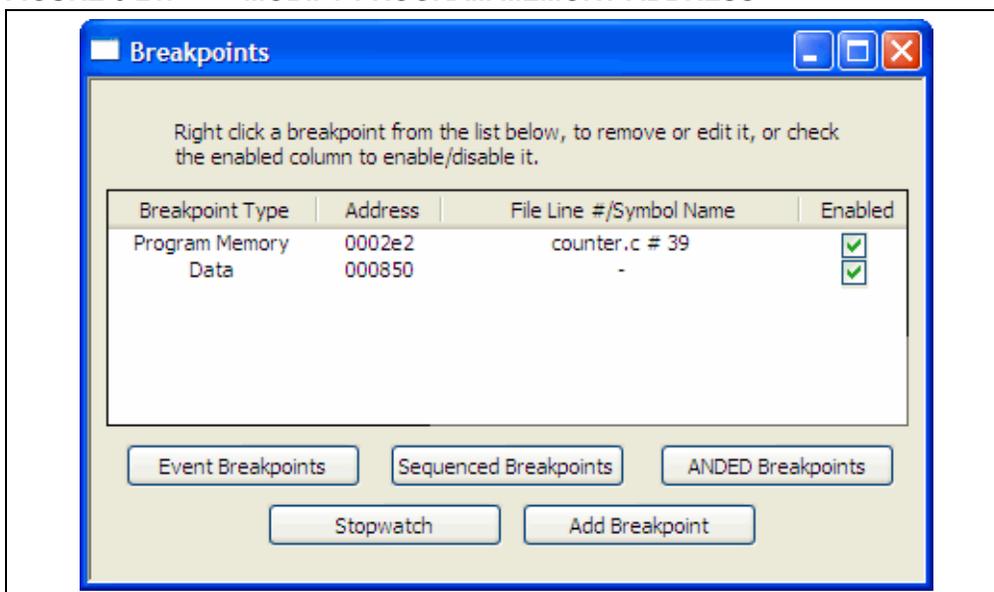
Why? Because for ANDed breakpoints to trigger they must occur at the same time.

Modify the program memory address from 0x2de to 0x2e2 (see Figure 5-20) to coincide with the write to the register at address 0x850 as shown in Figure 5-21.

FIGURE 5-20: DISASSEMBLY VIEW

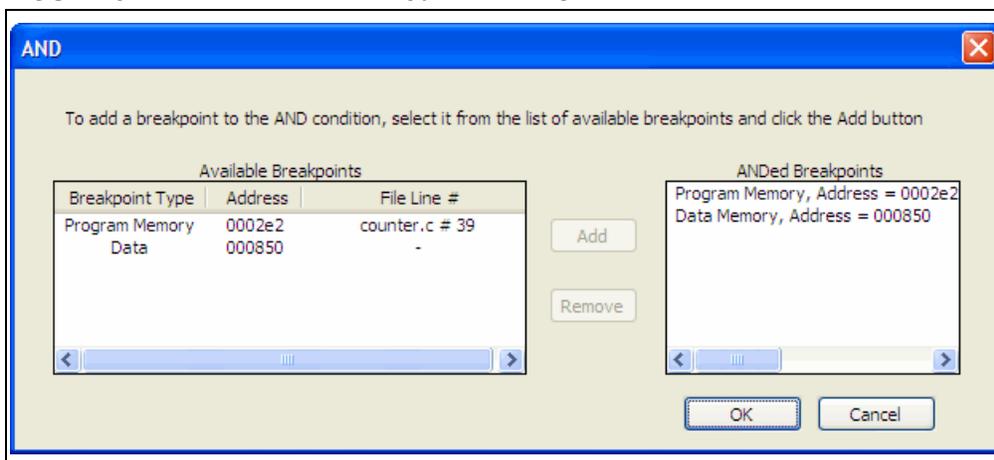
```
46:                                     counter++; //increment counter
002DE  804280  mov.w 0x0850, 0x0000
002E0  E80000  inc.w 0x0000, 0x0000
002E2  884280  mov.w 0x0000, 0x0850
47:                                     PORTA = counter; //display on port LEDs
002E4  804280  mov.w 0x0850, 0x0000
002E6  881610  mov.w 0x0000, 0x02c2
48:                                     }// End of if...
```

FIGURE 5-21: MODIFY PROGRAM MEMORY ADDRESS



Verify that the new address is reflected in the ANDED breakpoint dialog (see Figure 5-22).

FIGURE 5-22: VERIFY ANDED BREAKPOINT

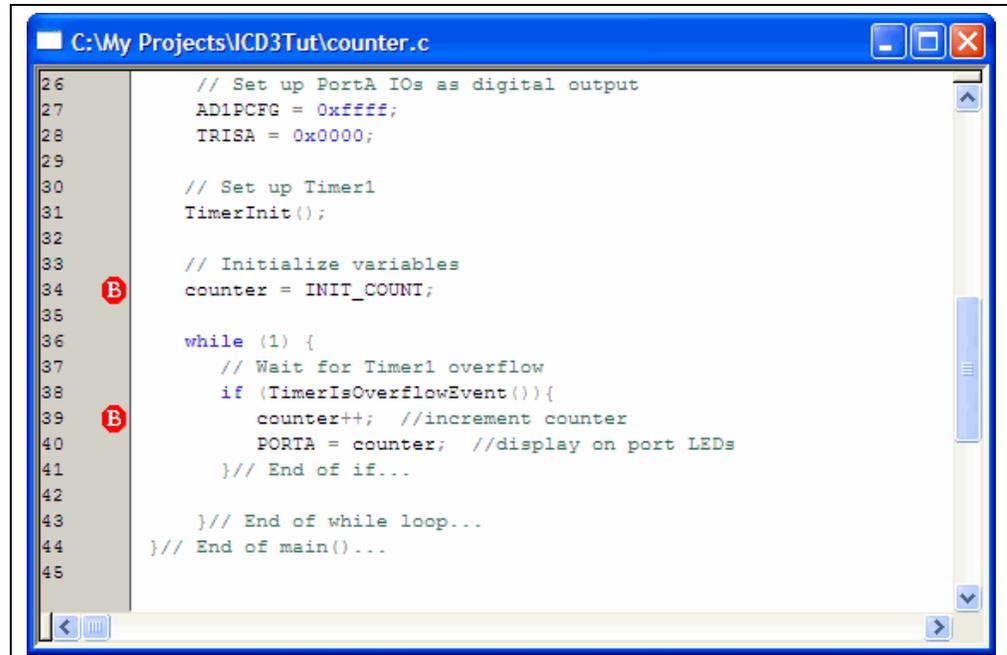


5.11.5 Sequenced Breakpoints

Sequenced breakpoints can be specified to occur in a desired sequence. There can be up to four sequences specified for some devices. The number of breakpoints is device dependent. For a device that has a total of four breakpoints, the first three are the qualifiers and the last one is the trigger. The trigger is the final breakpoint and halts the CPU.

In this example, double click on the `counter` and `counter++` as shown in Figure 5-23.

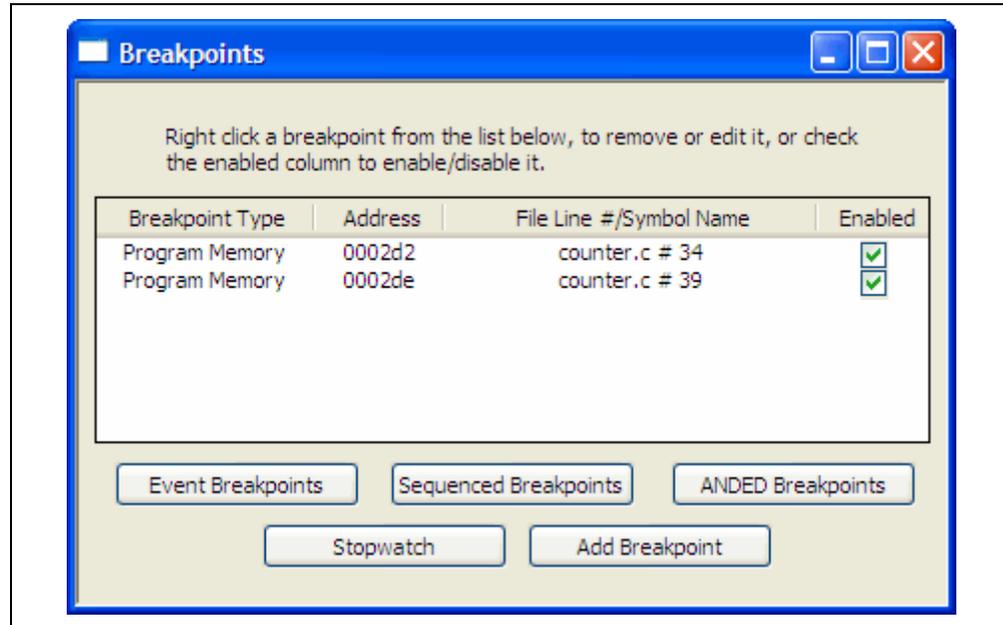
FIGURE 5-23: SET BREAKPOINTS AT COUNTER AND COUNTER++ VARIABLES



MPLAB® ICD 3 In-Circuit Debugger User's Guide

Select *Debugger>Breakpoints* (or select the Breakpoints button from the Debug toolbar) to open the Breakpoints dialog (see Figure 5-24). Note that the address and line numbers for the breakpoints set are already shown.

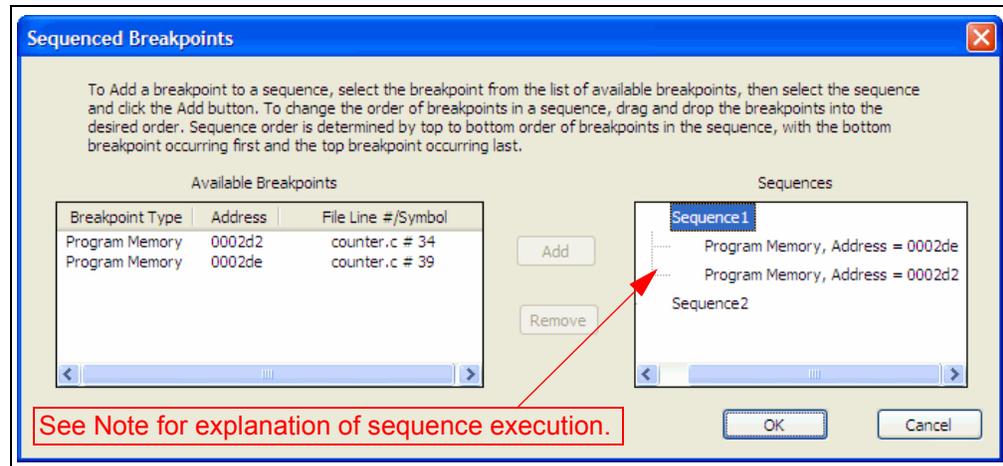
FIGURE 5-24: BREAKPOINTS DIALOG



Click **Sequenced Breakpoints** to open the Sequenced Breakpoints dialog (Figure 5-25). Select each item from the Available Breakpoints box and click **Add** to add each one to the Sequences on the right.

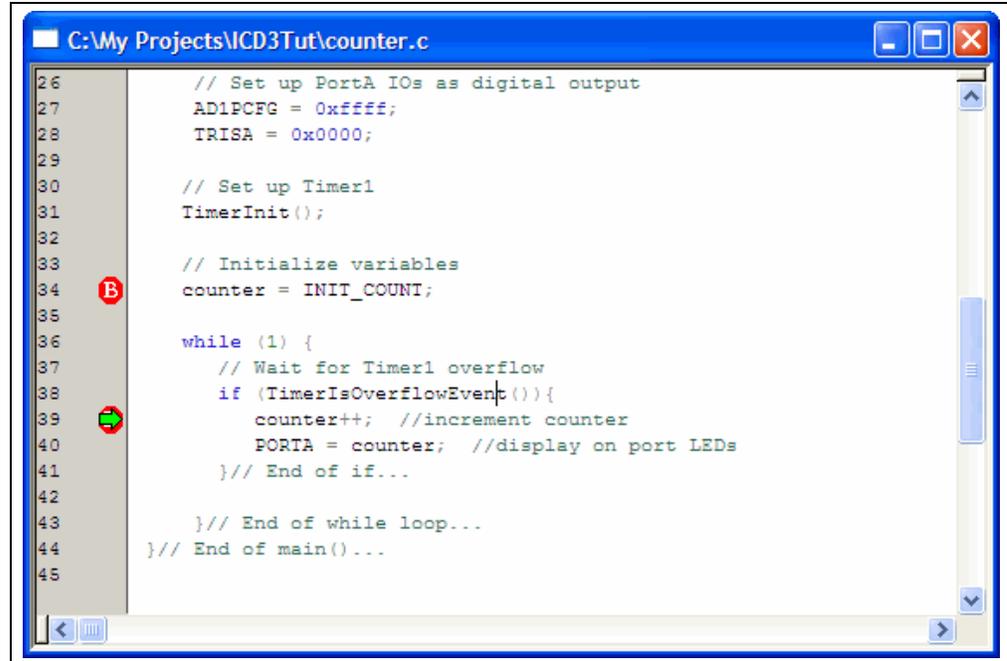
Note: The sequence execution is from the bottom to the top, where the trigger or breakpoint is at the top and the qualifier(s) are at the bottom.

FIGURE 5-25: SEQUENCED BREAKPOINTS DIALOG



Reset and **Run** the program. The program will stop at the second breakpoint defined in the sequence. The first qualifier at line number 34 essentially arms the breakpoint logic so when the breakpoint is executed at line number 39, it halts the CPU. See Figure 5-26.

FIGURE 5-26: SEQUENCED BREAKPOINTS

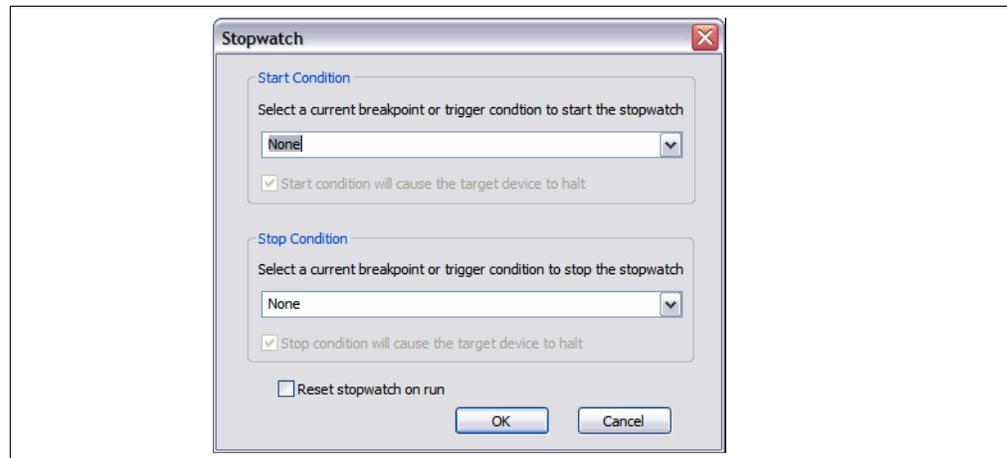


5.11.6 Using the Stopwatch with Breakpoints

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

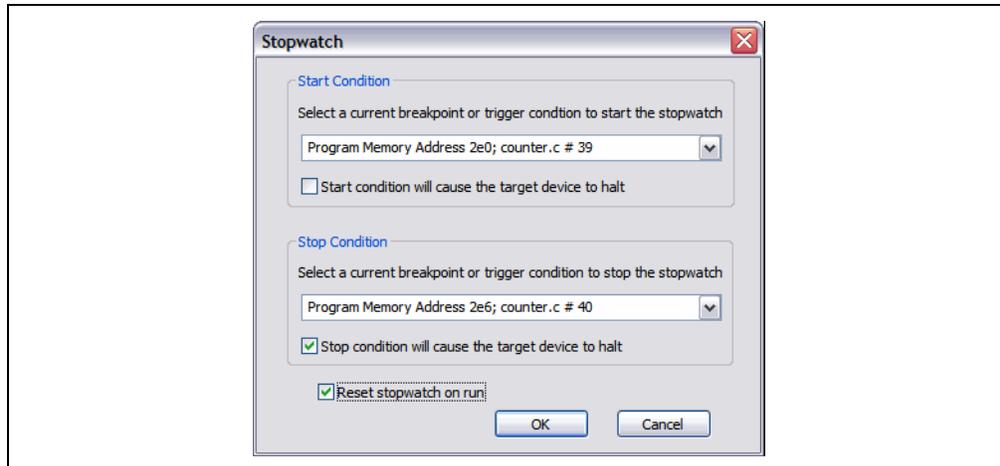
1. Click **Stopwatch** (on the Breakpoints dialog) to open the Stopwatch dialog (Figure 5-27).

FIGURE 5-27: STOPWATCH DIALOG



2. Under “Start Condition”, select the first breakpoint from the list. Then uncheck “Start condition will cause the target device to halt”.
3. Under “Stop Condition”, select the second breakpoint from the list. Then check “Stop condition will cause the target device to halt”.
4. Check “Reset stopwatch on run”. See Figure 5-28.
5. Click **OK**.

FIGURE 5-28: SETTING THE STOPWATCH DIALOG



6. Run the program until it halts. In the Output window, on the **ICD 3** tab, the number of cycles between the two instructions should be shown as:

```
Stopwatch cycle count = 4(decimal)
```

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.

5.11.7 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 5.11.2 "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 5.11.3 "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 a maximum number of breakpoints with software breakpoints, i.e., although this tutorial only uses two, the number of software breakpoints is 999.
- 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 5.11.6 “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**.

5.12 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 ICD 3 in-circuit debugger as the debug tool by selecting *Debugger>Select Tool>None*.
2. Enable the MPLAB ICD 3 in-circuit debugger as the programmer by selecting *Programmer>Select Programmer>ICD 3*.
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 debugger as a debug tool.

1. Disable the MPLAB ICD 3 in-circuit debugger as the programmer by selecting *Programmer>Select Programmer>None*.
2. Enable the MPLAB ICD 3 in-circuit debugger as the debug tool by selecting *Debugger>Select Tool>ICD 3*.
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.

MPLAB® ICD 3 In-Circuit Debugger User's Guide

NOTES:



MPLAB® ICD 3 IN-CIRCUIT DEBUGGER USER'S GUIDE

Part 2 – Troubleshooting

Chapter 6. Frequently Asked Questions (FAQs)	59
Chapter 7. Error Messages.....	63

MPLAB® ICD 3 In-Circuit Debugger User's Guide

NOTES:

Chapter 6. Frequently Asked Questions (FAQs)

6.1 INTRODUCTION

Look here for answers to frequently asked questions about the MPLAB ICD 3 in-circuit debugger system.

- How Does It Work
- What's Wrong

6.2 HOW DOES IT WORK

- **What's in the silicon that allows it to communicate with the MPLAB ICD 3 in-circuit debugger?**

MPLAB ICD 3 in-circuit debugger can communicate with Flash 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 debugger doesn't 'steal' any cycles from the target device.

- **How does the MPLAB ICD 3 in-circuit debugger compare with other in-circuit emulators/debuggers?**

Please refer to **Section 2.2 "MPLAB ICD 3 In-Circuit Debugger vs. MPLAB ICE 2000/4000 In-Circuit Emulators"**.

- **How does MPLAB IDE interface with the MPLAB ICD 3 in-circuit debugger to allow more features than older debuggers?**

MPLAB ICD 3 in-circuit debugger communicates using the debug executive located in the test area. The debug exec is streamlined for more efficient communication. The debugger contains an FPGA, large SRAM Buffers (1Mx8) and a High Speed USB interface. Program memory image is downloaded and is contained in the SRAM to allow faster programming. The FPGA in the debugger 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 ICD 3 in-circuit debugger? For example, could I halt based on a flag going high?**

Traditional emulators use a special debugger chip (-ME) for monitoring. There is no -ME with the MPLAB ICD 3 in-circuit debugger so there are no busses to monitor externally. With the MPLAB ICD 3 in-circuit debugger, 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.

MPLAB® ICD 3 In-Circuit Debugger User's Guide

- **Does the MPLAB ICD 3 in-circuit debugger 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 the AND condition and do PASS counts.
- **Are any of the driver boards optoisolated or electrically isolated?**

They are DC optoisolated, but not AC optoisolated. You cannot apply a floating or high voltage (120V) to the current system.
- **What limitations are there with the standard cable?**

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.
- **Will this slow down the running of the program?**

There is no cycle stealing with the MPLAB ICD 3 in-circuit debugger. The output of data is performed by the state machine in the silicon.
- **Is it possible to debug a dsPIC DSC running at any speed?**

The MPLAB ICD 3 is capable of debugging at any device speed as specified in the device's data sheet.
- **What is the function of pin 6, the LVP pin?**

Pin 6 is reserved for the LVP (Low-Voltage Programming) connection.

6.3 WHAT'S WRONG

- **My PC went into Power-Down/Hibernate mode, and now my debugger won't work. What happened?**

When using the debugger 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).

Frequently Asked Questions (FAQs)

- **When using a 16-bit device, an 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);
}
:
void __attribute__((__interrupt__))
_AltOscillatorFail(void)
{
    INTCON1bits.OSCFAIL = 0;
    while (1);
}
:
```

- Use ASSERTs.

For example: ASSERT (IPL==7)

- **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 debugger 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 debugger 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 debugger as a debugger and then tried to program a production device? Programming a device when the debugger 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 debugger 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 or close and reopen the workspace.

MPLAB® ICD 3 In-Circuit Debugger User's Guide

- **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**. Or, you can exit MPLAB IDE and enable the debugger to start it automatically.

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

Reconnect the debugger. It will begin to erase what had been written so it can restart. This erasing will take about 7 seconds. Please be patient. The LEDs are all on during this process. When it is done, 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:

- **Chapter 9. "Limitations"**
- **Section 2.9 "Resources Used by the Debugger"**

Chapter 7. Error Messages

7.1 INTRODUCTION

The MPLAB ICD 3 in-circuit debugger produces many different error messages; some are specific, some are informational and others can be resolved with general corrective actions.

- Specific Error Messages
- Information Messages
- General Corrective Actions

7.2 SPECIFIC ERROR MESSAGES

MPLAB ICD 3 in-circuit debugger 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.

ICD3Err0001: Failed while writing to program memory.

ICD3Err0002: Failed while writing to EEPROM.

ICD3Err0003: Failed while writing to configuration memory.

See Section 7.4.1 “Read/Write Error Actions”.

ICD3Err0005: ICD 3 is currently busy and cannot be unloaded at this time.

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

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

ICD3Err0006: Failed while writing to user ID memory.

ICD3Err0007: Failed while reading program memory.

ICD3Err0008: Failed while reading EEPROM.

ICD3Err0009: Failed while reading configuration memory.

ICD3Err0010: Failed while reading user ID memory.

See Section 7.4.1 “Read/Write Error Actions”.

ICD3Err0011: Bulk erase failed.

See Section 7.4.1 “Read/Write Error Actions”.

If these do not work, try another device.

ICD3Err0012: Download debug exec failed

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

1. Deselect the debugger 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 debugger as your debug tool and attempt to program your target device again.

If this does not work, see **Section 7.4.4 “Corrupted Installation Actions”**.

ICD3Err0013: NMMR register write failed.

ICD3Err0014: File register write failed.

See **Section 7.4.2 “Debugger-to-Target Communication Error Actions”**.

ICD3Err0015: Data transfer was unsuccessful. %d byte(s) expected, %d byte(s) transferred.

See **Section 7.4.3 “Debugger-to-PC Communication Error Actions”**.

ICD3Err0016: Cannot transmit. ICD 3 not found.

The debugger is not connected to the PC.

ICD3Err0017: File register read failed.

ICD3Err0018: NMMR register read failed.

ICD3Err0019: Failed while reading emulation registers.

ICD3Err0020: Failed while writing emulation registers.

See **Section 7.4.2 “Debugger-to-Target Communication Error Actions”**.

ICD3Err0021: Command not echoed properly. Sent %x, received %x.

ICD3Err0022: Failed to get ICD 3 version information.

ICD3Err0023: Download FPGA failed.

ICD3Err0024: Download RS failed.

ICD3Err0025: Download AP failed.

See **Section 7.4.3 “Debugger-to-PC Communication Error Actions”**.

ICD3Err0026: Download program exec failed.

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

1. Deselect the debugger 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 debugger as your debug tool and attempt to program your target device again.

If this does not work, see **Section 7.4.4 “Corrupted Installation Actions”**.

ICD3Err0027: Bulk transfer failed due to invalid checksum

See **Section 7.4.3 “Debugger-to-PC Communication Error Actions”**.

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

ICD3Err0028: Download device database failed

If you receive this error:

1. Try downloading again. It may be a one-time error. Allow sufficient time for the download to complete and do not disconnect the MPLAB ICD 3 during the download.
2. Try manually downloading. Select *Debugger>Settings, Configuration* tab, and click **Manual Download**. Select the highest number .jam file and click **Open**.

ICD3Err0029: Communication failure. Unexpected command echo response %x received from ICD 3.

See **Section 7.4.3 “Debugger-to-PC Communication Error Actions”**.

ICD3Err0030: 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.

ICD3Err0031: Failed to get PC.

ICD3Err0032: Failed to set PC.

See **Section 7.4.2 “Debugger-to-Target Communication Error Actions”**.

ICD3Err0033: %d bytes expected, %d bytes received.

See **Section 7.4.3 “Debugger-to-PC Communication Error Actions”**.

Use the ICD 3 Test Interface Board to verify your MPLAB ICD 3 is functioning properly. See **Section A.7 “ICD 3 Test Interface Board”**.

ICD3Err0034: This version of MPLAB IDE does not support hardware revision %06x. Please upgrade to the latest version of MPLAB IDE before continuing.

If you receive this error, try manually downloading the latest version of MPLAB IDE. Keep the MPLAB ICD 3 connected to the PC and select *Debugger>Settings, Configuration* tab, and click **Manual Download**. In the Update MPLAB ICD 3 window, select the highest number .jam file and click **Open**. Allow sufficient time for the download to complete. Check the Output window for the completion status.

ICD3Err0035: Failed to get Device ID. If you experience persistent problems communicating, the ICD 3 test interface can be used to help diagnose the problem.

See **Section 7.4.1 “Read/Write Error Actions”** and **Section A.7 “ICD 3 Test Interface Board”**.

ICD3Err0036: MPLAB IDE has lost communication with ICD 3.

See **Section 7.4.3 “Debugger-to-PC Communication Error Actions”**.

ICD3Err0037: Timed out waiting for response from ICD 3.

ICD3Err0038: Failed to initialize ICD 3.

ICD3Err0039: ICD 3 self-test failed.

For this error, the debugger is not responding:

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

ICD3Err0040: 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 7.4.6 “Debug Failure Actions”**.

Use the ICD 3 Test Interface Board to verify your MPLAB ICD 3 is functioning properly. See **Section A.7 “ICD 3 Test Interface Board”**.

ICD3Err0041: While receiving streaming data, ICD 3 has gotten out-of-sync with MPLAB IDE. To correct this you must reset the target device.

First try to Halt, Reset and then Run again. If this does not work:

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

ICD3Err0045: You must connect to a target device to use MPLAB ICD 3.

No power has been found.

1. Ensure VDD and GND are connected between the debugger and target.
2. Ensure that the target is powered.
3. Ensure that the target power is sufficient to be detected by the debugger (see **Appendix A. "Hardware Specification"**.)

ICD3Err0046: An error occurred while trying to read the stopwatch count. The stopwatch count may not be accurate.

See **Section 7.4.2 "Debugger-to-Target Communication Error Actions"**.

ICD3Err0047: Bootloader download failed.

See **Section 7.4.3 "Debugger-to-PC Communication Error Actions"**.

ICD3Err0052: The current ICD 3 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.

ICD3Err0053: Unable to get ICD 3 protocol versions.

See **Section 7.4.3 "Debugger-to-PC Communication Error Actions"**.

ICD3Err0054: MPLAB IDE's ICD 3 protocol definitions are out of date. You must upgrade MPLAB IDE to continue.

Find the latest MPLAB IDE at www.microchip.com.

ICD3Err0055: Unable to set firmware suite version.

ICD3Err0056: Unable to get voltages from ICD 3.

See **Section 7.4.3 "Debugger-to-PC Communication Error Actions"**.

ICD3Err0057: Interface test could not be completed. Please contact your local FAE/CAE to SAR the unit.

Ensure that you are using the ICD 3 self-test board, see **Section A.7 "ICD 3 Test Interface Board"**. Also, see **Section 7.4.2 "Debugger-to-Target Communication Error Actions"**.

ICD3Err0063: Test interface PGC clock line write failure. Please ensure that the tester is properly connected.

ICD3Err0064: Test interface PGD data line write failure.

ICD3Err0065: Test interface PGC clock line read failure.

ICD3Err0066: Test interface PGD data line read failure.

Clock/data not being output from the debugger. Check your connections and try again. Also, see **Section A.7 "ICD 3 Test Interface Board"**.

ICD3Err0067: Failed to set/clear software breakpoint.

Reprogram and try again.

ICD3Err0068: Failed while writing to boot Flash memory.

ICD3Err0069: Failed while reading boot Flash memory.

ICD3Err0070: Failed while writing peripheral memory.

ICD3Err0071: Failed while reading peripheral memory.

See **Section 7.4.1 “Read/Write Error Actions”**.

ICD3Err0072: Unable to send freeze peripheral information.

See **Section 7.4.3 “Debugger-to-PC Communication Error Actions”**.

ICD3Err0073: 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.

ICD3Err0082: Test interface LVP control line failure.

ICD3Err0083: Test interface MCLR level failure.

See **Section A.7 “ICD 3 Test Interface Board”**.

7.3 INFORMATION MESSAGES

ICD3Info0001: ICD3 is functioning properly. If you are still having problems with your target circuit please check the Target Board Considerations section of the online help.

See **Section A.8 “Target Board Considerations”**.

7.4 GENERAL CORRECTIVE ACTIONS

These general corrective actions may solve your problem:

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

7.4.1 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 debugger-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.

7.4.2 Debugger-to-Target Communication Error Actions

The MPLAB ICD 3 in-circuit debugger and the target device are out-of-sync with each other.

1. Select **Reset** and then try the action again.
2. Ensure that the cable(s) used are the correct length.
3. Use the ICD 3 Test Interface Board to verify your MPLAB ICD 3 is functioning properly. See **Section A.7 “ICD 3 Test Interface Board”**.

7.4.3 Debugger-to-PC Communication Error Actions

The MPLAB ICD 3 in-circuit debugger and MPLAB IDE are out-of-sync with each other.

1. Use the ICD 3 Test Interface Board to verify your MPLAB ICD 3 is functioning properly. See **Section A.7 “ICD 3 Test Interface Board”**.
2. Unplug and then plug in the debugger.
3. Reconnect to the debugger.
4. Try the operation again. It is possible the error was a one time glitch.
5. The version of MPLAB IDE installed may be incorrect for the version of firmware loaded on the MPLAB ICD 3 in-circuit debugger. Follow the steps outlined in **Section 7.4.4 “Corrupted Installation Actions”**.

7.4.4 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. Force a manual download of the firmware.
4. If the problem persists contact Microchip.

7.4.5 USB Port Communication Error Actions

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

1. Reconnect to the MPLAB ICD 3 in-circuit debugger
2. Make sure the debugger is physically connected to the PC on the appropriate USB port.
3. Make sure the appropriate USB port has been selected in the debugger 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.

7.4.6 Debug Failure Actions

The MPLAB ICD 3 in-circuit debugger was unable to perform a debugging operation. There are numerous reasons why this might occur:

The 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 MPLAB ICD 3 in-circuit debugger has somehow become physically disconnected from the PC. Check the USB communication cable connection.
4. The debugger has somehow become physically disconnected from the target board. Check the communications cable connection.
5. The device is code-protected. Check your Configuration bits setting for code protection.
6. 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.
7. The debugger is selected as a programmer, and not as a debugger, in MPLAB IDE.
8. Debugger to PC communications has somehow been interrupted. Reconnect to the debugger in MPLAB IDE.
9. The target application has somehow become corrupted or contains errors. For example, the regular linker script was used in the project instead of the debugger version of the linker script (e.g., 18F8722.lkr was used instead of 18F8722i.lkr). Try rebuilding and reprogramming the target application. Then initiate a Power-on Reset of the target.
10. Other configuration settings are interfering with debugging. Any configuration setting that would prevent the target from executing code will also prevent the debugger from putting the code into Debug mode.
11. The debugger cannot always perform the action requested. For example, the debugger cannot set a breakpoint if the target application is currently running.

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 ICD 3 in-circuit debugger functionality. Or, use the self-test board to verify the debugger itself (**Section A.7 “ICD 3 Test Interface Board”**.)
5. Review debugger debug operation to ensure proper application setup (**Chapter 2. “Operation”**.)
6. If the problem persists contact Microchip.

7.4.7 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 7.4.4 “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® ICD 3 IN-CIRCUIT DEBUGGER USER'S GUIDE

Part 3 – Reference

Chapter 8. Basic Debug Functions.....	73
Chapter 9. Debugger Function Summary	75
Appendix A. Hardware Specification.....	91

MPLAB® ICD 3 In-Circuit Debugger User's Guide

NOTES:

Chapter 8. Basic Debug Functions

8.1 INTRODUCTION

Basic MPLAB ICD 3 in-circuit debugger functions of breakpoints and stopwatch are discussed.

8.2 BREAKPOINTS AND STOPWATCH

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

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.

Breakpoints and triggers use the same resources. Therefore, the available number of breakpoints is actually the available number of combined breakpoints/triggers.

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 9.5.2 “Settings Dialog, Configuration Tab”** for more information.)

<p>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.</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 9.3.1 “Breakpoints Dialog”** for more information.

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

1. Determine whether your device supports stopwatch use. See the online Help file, “Device and Feature Support”.
2. Open the Breakpoints dialog (*Debugger>Breakpoints*).
3. Click **Stopwatch** on the Breakpoints dialog to open the Stopwatch dialog.
4. Under “Start Condition”, select a breakpoint from the drop-down list. Also decide whether or not to select “Start condition will cause the target device to halt”.
5. Under “Stop Condition”, select another breakpoint from the drop-down list. Also decide whether or not to select “Stop condition will cause the target device to halt”.
6. Decide if there will be a “Reset stopwatch on run”.
7. Click **OK**.

MPLAB® ICD 3 In-Circuit Debugger User's Guide

NOTES:

Chapter 9. Debugger Function Summary

9.1 INTRODUCTION

A summary of the MPLAB ICD 3 in-circuit debugger functions on menus, in windows and on dialogs is listed here.

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

9.2 DEBUGGING FUNCTIONS

When you select the MPLAB ICD 3 from the Debugger menu, the following items will be added to the MPLAB IDE functions:

- Debugger Menu – additional options are added to the drop-down menu
- Right Mouse Button Debugger Menu – additional options are added to this menu
- Toolbars/Status Bar – a toolbar appears below the menu bar; additional information appears in the status bar

9.2.1 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.

MPLAB® ICD 3 In-Circuit Debugger User's Guide

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 9.3.1 “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.

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

Reads program memory using the debug executive.

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 ICD 3 in-circuit debugger. The progress of this connection is shown on the **ICD 3** tab of the Output dialog.

Settings

Open the Programmer dialog (see **Section 9.5 “Settings Dialog”**). Set up program and firmware options.

9.2.2 Right Mouse Button Debugger Menu

These debugger menu options will appear on the right mouse menus in code displays, such as program memory and source code files. Descriptions of other menu options not listed here can be found in the MPLAB IDE Help or the MPLAB Editor Help.

Set Breakpoint

Set or remove 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.

Center Debug Location

Center the current PC line in the window.

9.2.3 Toolbars/Status Bar

When the MPLAB ICD 3 in-circuit debugger 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 ICD 3), 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 online help for information on the contents of the status bar.

9.3 DEBUGGING DIALOGS/WINDOWS

Open the following debug dialogs and windows using the menu items mentioned in **Section 9.2 “Debugging Functions”**.

- Breakpoints Dialog
 - Set Breakpoint Dialog
 - Stopwatch Dialog
 - Event Breakpoints Dialog
 - Sequenced Breakpoints Dialog
 - ANDed Breakpoints Dialog
- Application In/Out Window (Device Dependent)

9.3.1 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.

9.3.1.1 BREAKPOINT DIALOG WINDOW

Information about each breakpoint is visible in this window.

TABLE 9-1: 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

Debugger Function Summary

9.3.1.2 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 9-2: BREAKPOINT DIALOG BUTTONS

Control	Function	Related Dialog
Add Breakpoint	Add a breakpoint	Section 9.3.2 “Set Breakpoint Dialog”
Stopwatch	Set up the stopwatch	Section 9.3.3 “Stopwatch Dialog”
Event Breakpoints	Set up break on an event	Section 9.3.4 “Event Breakpoints Dialog”
Sequenced Breakpoints	Set up a sequence until break	Section 9.3.5 “Sequenced Breakpoints Dialog”
ANDed Breakpoints	Set up ANDed condition until break	Section 9.3.6 “ANDed Breakpoints Dialog”

9.3.2 Set Breakpoint Dialog

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

Select a breakpoint for the Breakpoints dialog here.

9.3.2.1 PROGRAM MEMORY TAB

Set up a program memory breakpoint here.

TABLE 9-3: 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. <i>Program Memory Execution</i> – break on execution of above address <i>TBLRD Program Memory</i> – break on table read of above address <i>TBLWT Program Memory</i> – break on table write to above address
Pass Count	Break on pass count condition. <i>Always break</i> – always break as specified in “Breakpoint type” <i>Break occurs Count instructions after Event</i> – wait Count (0-255) instructions before breaking after event specified in “Breakpoint type” <i>Event must occur Count times</i> – break only after event specified in “Breakpoint type” occurs Count (0-255) times

MPLAB® ICD 3 In-Circuit Debugger User's Guide

9.3.2.2 DATA MEMORY TAB

Set up a data memory breakpoint here.

TABLE 9-4: 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. <i>X Bus Read</i> – break on an X bus read of above address <i>X Bus Read Specific Byte</i> – break on an X bus read of above address for the specific byte value in “Specific Value” <i>X Bus Read Specific Word</i> – break on an X bus read of above address for the specific word value in “Specific Value” <i>X Bus Write</i> – break on an X bus write of above address <i>X Bus Write Specific Byte</i> – break on an X bus write of above address for the specific byte value in “Specific Value” <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. <i>Always break</i> – always break as specified in “Breakpoint type” <i>Break occurs Count instructions after Event</i> – wait Count (0-255) instructions before breaking after event specified in “Breakpoint type” <i>Event must occur Count times</i> – break only after event specified in “Breakpoint type” occurs Count (0-255) times

9.3.3 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 9-5: STOPWATCH SETUP

Control	Function
Start Condition	Select an available breakpoint or trigger condition to start the stopwatch. Available breakpoints/triggers are those previously added to the breakpoint dialog. Select None to clear the start condition. 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	Select an available breakpoint or trigger condition to stop the stopwatch. Available breakpoints/triggers are those previously added to the breakpoint dialog. Select None to clear the stop condition. 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.

9.3.4 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.

9.3.5 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**.

9.3.6 ANDed Breakpoints 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**.

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.

Show Source

Show/hide the source code listing on the bottom of the window.

Refresh

Refresh the viewable contents of the window.

Properties

Set up window properties.

Debugger Function Summary

9.3.7 Application In/Out Window (Device Dependent)

View application in/out information in this window ([View>Application In/Out](#)).

Note: This window is only available for devices that support the application in/out function (currently PIC32MX5XX/6XX/7XX MCUs) used with the MPLAB REAL ICE in-circuit emulator or MPLAB ICD 3 in-circuit debugger. The application in/out function cannot be used with PIC32 Instruction trace.

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

For more on how to set up the application, see **Section 9.3.7.3 “Application Code Using AppIn/AppOut”** below.

TABLE 9-6: INPUT/OUTPUT WINDOW

Control	Function
Capture	Capture Output display data to a text file. Off: Output display capture off. On: Output display capture on. Enter the path and name of the capture file, or Browse to it. Data sent to the capture file is always appended to it. User Input commands will appear in the file in double brackets, i.e., [[Send more data]]. Note: You must close this window to see appended information appear in the file.
Clear Display	Clear the data currently in the Output display.
Connection	Currently unused (dedicated Application In/Out).
Format	Specify the Output or Input data format: Text, 8-bit Hex, 16-bit Hex or 32-bit Hex.
Output	Output data from the target is displayed here. Also, User Input is displayed in bold after it has been successfully sent. Right-click in the display to open a menu.
User Input	Type in commands to be sent to the target application. Press Enter to send. Right-click in the text box to open a menu. Note: Although the “User Input” text box may hold more than 4 bytes of information, the debug tool will only send 4 bytes at a time to the target. Therefore the target application may need to check the application input register more than once to receive all data sent.

9.3.7.1 OUTPUT DISPLAY MENU

Perform copying, clearing and display formatting on the data in the Output display.

- Copy Selection – Select an area of text with the mouse in the Output display and then use this command to copy it to the PC clipboard. Then you may Paste it into Textpad or another application.
- Clear – removes the data currently in the Output display.
- Display with ASCII – Display the ASCII representation (if any) of the values in the hex data to the right of the data.
- Display with Index – Display a relative byte index of the hex data to the left of the

data.

9.3.7.2 USER INPUT MENU

Perform general editing functions on the text in the User Input text box.

- Undo – Removes previous typing.
- Cut – Remove the selected/highlighted text and place on the PC clipboard.
- Copy – Place the selected/highlighted text on the PC clipboard
- Paste – Insert any text on the PC clipboard at the current cursor position.
- Delete – Remove the selected/highlighted text.
- Select All – Select/highlight all text in this text box.

9.3.7.3 APPLICATION CODE USING APPIN/APPOUT

Use the device-specific header file when building your application to use the macros assigned in the header (per Example 9-1).

Alternatively, the PIC32 MCU library has `printf()` and `scanf()` functions for use with the Application In/Out feature. Refer to the “PIC32 Debug-Support Library” chapter in *32-Bit Language Tools Libraries* (DS51685).

EXAMPLE 9-1: APPLICATION IN/OUT MACRO USAGE

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

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

9.4 PROGRAMMING FUNCTIONS

When you select the MPLAB ICD 3 in-circuit debugger from the Programmer menu, program items will be added to the following MPLAB IDE functions:

- Programmer Menu
- Toolbars/Status Bar

9.4.1 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.

Settings

Open the Programmer dialog (see **Section 9.5 “Settings Dialog”**). Set up program and firmware options.

9.4.2 Toolbars/Status Bar

When the MPLAB ICD 3 in-circuit debugger is selected as a programmer, these toolbars are displayed in MPLAB IDE:

- Basic program toolbar (Blank Check All, Read, Program, Verify, Erase Flash Device).

The selected programmer (MPLAB ICD 3), 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 online help for information on the contents of the status bar.

MPLAB® ICD 3 In-Circuit Debugger User's Guide

9.5 SETTINGS DIALOG

Select either *Debugger>Settings* or *Programmer>Settings* to open the Settings dialog and set up the MPLAB ICD 3 in-circuit debugger.

Note 1: Tabs displayed will depend on the selected device.

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

- Settings Dialog, Program Memory Tab
- Settings Dialog, Configuration Tab
- Settings Dialog, Freeze on Halt Tab
- Settings Dialog, Status Tab
- Settings Dialog, Secure Segment Tab
- Settings Dialog, Warnings Tab
- Settings Dialog, Power Tab
- Settings Dialog, Limitations Tab
- Settings Dialog, Calibration Memory Tab

9.5.1 Settings Dialog, Program Memory Tab

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

TABLE 9-7: PROGRAM MEMORY OPTIONS

Allow ICD 3 to select memories and ranges	If selected, the MPLAB® ICD 3 uses your selected device and default settings to determine what to program.
Manually select memories and ranges	If selected, you must determine the Memories, Program Memory Range, and Program Options for the selected device.
Memories:	
Program	Check to program Program Memory into target.
Configuration	Check to program Configuration bits into target. Note: This memory is always programmed when debugger 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
Program Memory Range:	
Start, End	The starting and ending hex address range in program memory for programming, reading, or verification. 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. Note: The address range does not apply to the Erase function. The Erase function will erase all data on the device.
Program Options:	
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.

Debugger Function Summary

TABLE 9-7: PROGRAM MEMORY OPTIONS (CONTINUED)

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. Uncheck to use EEPROM checkbox functionality under Memories.
Preserve Program Memory Range	Check to preserve memory range. Start, End – The starting and ending hex address range in program memory for preservation.
Use high voltage on MCLR	<i>For PIC24FXXKAXXX devices:</i> Check this option to use <u>high voltage</u> to configure pin as either a normal input pin or as a MCLR Reset. Leave unchecked for a low voltage MCLR Reset. Notes: 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). 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. 3. If you want to program the “MCLR pin enable bit” configuration setting, you must check the “Use high voltage on MCLR” option. 4. If you are using a -ICE header, the setting of this checkbox does not matter.
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> After the application code is successfully programmed into the target device, run the code.

9.5.2 Settings Dialog, Configuration Tab

Configure debugger operation on this tab.

TABLE 9-8: CONFIGURATION ITEMS

Download Firmware	Set up firmware download options.
Auto Download Latest Firmware	Check to allow automatic download of the latest version of firmware for the target device (recommended).
Manual Download	Manually select a firmware file to download to the target device.
Breakpoints	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: <ul style="list-style-type: none"> • Number of breakpoints: limited • Breakpoints are written to debug registers • Time to set breakpoints: minimal • Skidding: yes

TABLE 9-8: CONFIGURATION ITEMS (CONTINUED)

Use Software Breakpoints	<p>Using software breakpoints means:</p> <ul style="list-style-type: none"> • Number of breakpoints: unlimited • Breakpoints are written to program memory • Time to set breakpoints: oscillator speed dependent – can take minutes • Skidding: no <p>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.</p>
--------------------------	--

9.5.3 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 debugger.

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 debugger.

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

9.5.4 Settings Dialog, Status Tab

View the status of your MPLAB ICD 3 system on this tab.

TABLE 9-9: STATUS ITEMS

Versions	
Firmware Suite Version	Debugger firmware suite version. The firmware suite consists of the three items specified below.
FPGA Version	Internal FPGA chip firmware version.
Algorithm Plug-in Version	Debugger algorithm plug-in version. For your selected device, an algorithm is used to support the device plugged into the target.
OS Version	Debugger operating system version.
Voltages	
ICD 3 VPP	Debugger VPP.
ICD 3 VDD	Debugger VDD.
Target VDD	VDD sensed at target.
Refresh Voltages	Sensing of Status tab items occurs when the tab is achieved. To see updates otherwise, click this button.

Debugger Function Summary

9.5.5 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 the Microchip web site.

TABLE 9-10: SECURE SEGMENT OPTIONS

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

9.5.6 Settings Dialog, Warnings Tab

A list of all MPLAB ICD 3 in-circuit debugger 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 7. “Error Messages”**.

9.5.7 Settings Dialog, Power Tab

Set up the power options on this tab.

- Click in the checkbox to enable/disable “Power target circuit from MPLAB ICD 3”.

If you enable (check) this option, the Power on/off button will be enabled on the toolbar. It will initially be in the Power-On state. Every time it is clicked it will toggle to the opposite state. If it is on, it will toggle to off, and if it is off it will toggle to on. If the Power target circuit setting is disabled (unchecked) the Power-on/off button will go back to the disabled state.

Whatever state it is in when the project was last saved will be the state that it is in when the project is reopened.

- Adjust the slide bar to select the voltage. The value in the field changes according to your adjustments.

9.5.8 Settings Dialog, Limitations Tab

View device limitations highlights on this tab. Click **Details** to see all the limitations for the device.

9.5.9 Settings Dialog, Calibration Memory Tab

For devices that have calibration memory, view and change calibration memory values on this tab.

Note: Microchip recommends that you **do not** change the factory calibration settings or your device may not work as expected.

TABLE 9-11: CALIBRATION MEMORY OPTIONS

Last Valid Value	If you have received an error (82 or 83), this is the last value read that was valid before the error.
Last Value	Last calibration value read from memory.
Allow ICD 3 to program calibration memory	Check to program calibration memory with the value specified in New. Values are hexadecimal.

MPLAB[®] ICD 3 In-Circuit Debugger User's Guide

NOTES:

Appendix A. Hardware Specification

A.1 INTRODUCTION

The hardware and electrical specifications of the MPLAB ICD 3 in-circuit debugger system are detailed.

A.2 HIGHLIGHTS

This chapter discusses:

- Declaration of Conformity
- USB Port/Power
- MPLAB ICD 3 Debugger
- Standard Communication Hardware
- ICD 3 Test Interface Board
- Target Board Considerations

A.3 DECLARATION OF CONFORMITY

We

Microchip Technology, Inc.
2355 W. Chandler Blvd.
Chandler, Arizona 85224-6199
USA

hereby declare that the product:

MPLAB® ICD 3 In-Circuit Debugger

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 ICD 3 In-Circuit Debugger

Due to the special nature of the MPLAB ICD 3 in-circuit debugger, 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.).

MPLAB® ICD 3 In-Circuit Debugger User's Guide

A.4 USB PORT/POWER

The MPLAB ICD 3 in-circuit debugger is connected to the host PC via a Universal Serial Bus (USB) port, version 2.0 compliant. The USB connector is located on the side of the pod.

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

System power is derived from the USB interface. The debugger 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 (debugger/programmer).

Note: The MPLAB ICD 3 in-circuit debugger is powered through its USB connection. The target board is powered from its own supply. Alternatively, the MPLAB ICD 3 can power it only if the target consumes less than 100 mA.

Cable Length – The PC-to-debugger cable length for proper operation is shipped in the debugger kit.

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

A.5 MPLAB ICD 3 DEBUGGER

The debugger consists of a main board enclosed in the casing with a USB connector and an RJ-11 connector. On the debugger enclosure are indicator lights (LEDs).

A.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 and LED indicators.

A.5.2 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 debugger is operating normally – standby.
	Red	Lit when an operation has failed.
	Orange	Lit when the debugger is busy.

A.6 STANDARD COMMUNICATION HARDWARE

For standard debugger communication with a target (, “Standard ICSP Device Communication“), use the RJ-11 connector.

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.

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 debugger.

See the “*Header Board Specification*” (DS51292) for more on available header boards.

A.6.1 Standard Communication

The standard communication is the main interface to the target processor. It contains the connections to the high voltage (V_{PP}), V_{DD} sense lines, and clock and data connections required for programming and connecting with the target devices.

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

The V_{DD} sense connection draws very little current from the target processor. The actual power comes from the MPLAB ICD 3 in-circuit debugger system as the V_{DD} sense line is used as a reference only to track the target voltage. The V_{DD} 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 ICD 3 in-circuit debugger 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

FIGURE A-1: 6-PIN STANDARD PINOUT

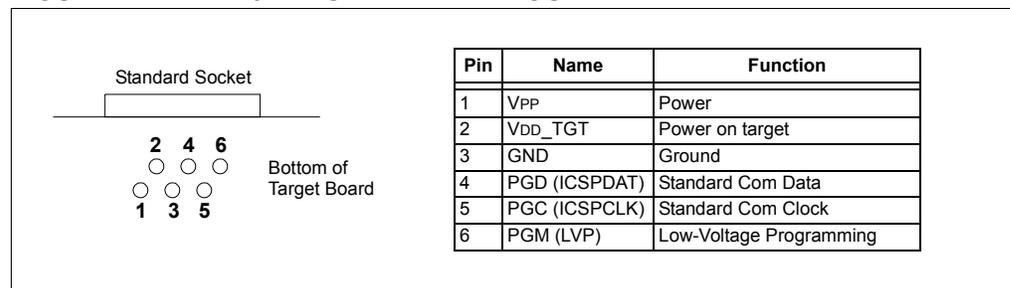


TABLE A-1: ELECTRICAL LOGIC TABLE

Logic Inputs	$V_{IH} = V_{DD} \times 0.7V$ (min.)			
	$V_{IL} = V_{DD} \times 0.3V$ (max.)			
Logic Outputs	$V_{DD} = 5V$	$V_{DD} = 3V$	$V_{DD} = 2.3V$	$V_{DD} = 1.65V$
	$V_{OH} = 3.8V$ min.	$V_{OH} = 2.4V$ min.	$V_{OH} = 1.9V$ min.	$V_{OH} = 1.2V$ min.
	$V_{OL} = 0.55V$ max.	$V_{OL} = 0.55V$ max.	$V_{OL} = 0.3V$ max.	$V_{OL} = 0.45V$ max.

Note: Loading PGC/PGD - 4.7K ohm load to ground.

A.6.2 Modular Cable and Connector

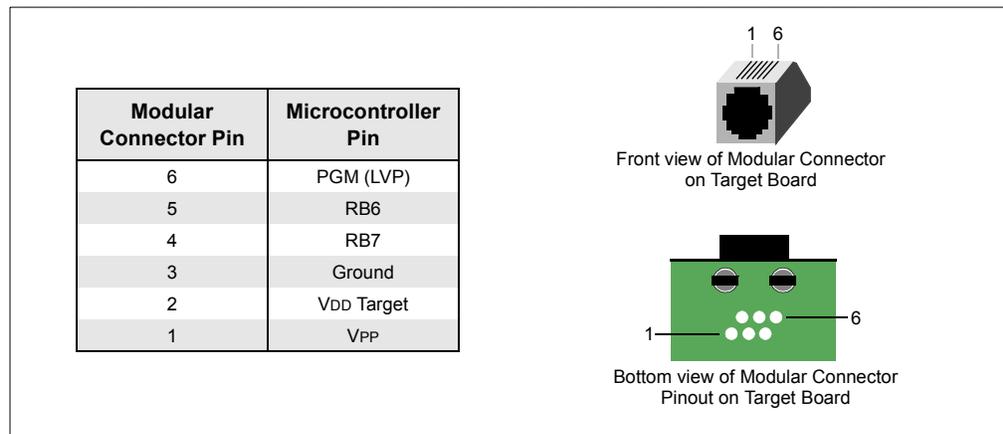
For standard communications, a modular cable connects the debugger and the target application. The specifications for this cable and its connectors are listed below.

A.6.2.1 MODULAR CONNECTOR SPECIFICATION

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

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

FIGURE A-2: MODULAR CONNECTOR PINOUT OF TARGET BOARD

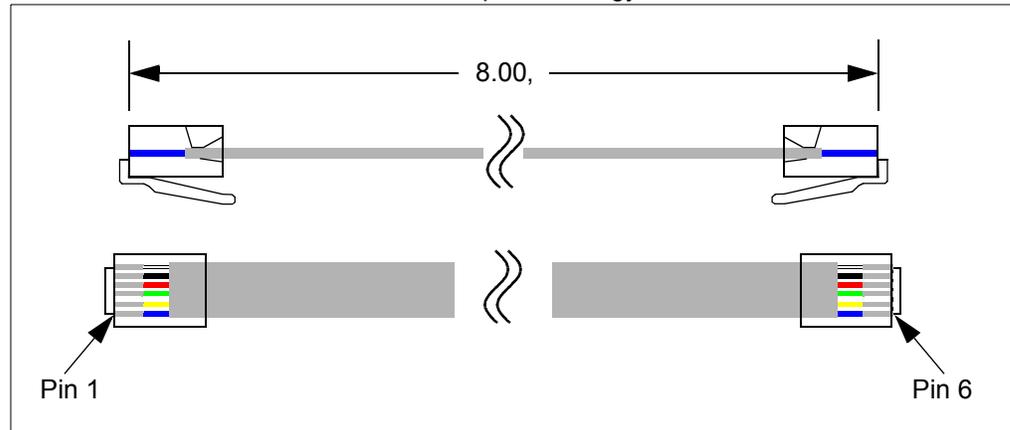


A.6.2.2 MODULAR PLUG SPECIFICATION

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

A.6.2.3 MODULAR CABLE SPECIFICATION

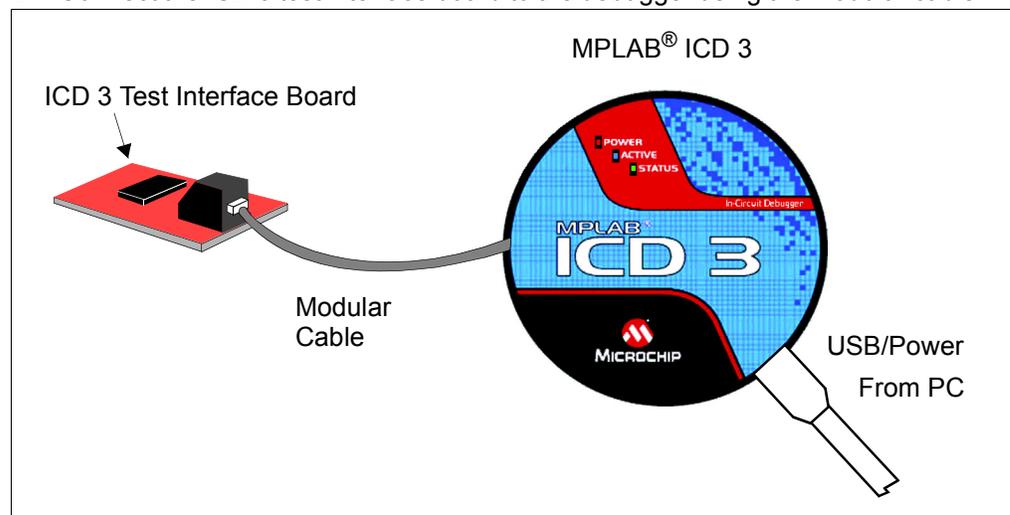
- Manufacturer, Part Number – Microchip Technology, 07-00024



A.7 MPLAB ICD 3 TEST INTERFACE BOARD

This board can be used to verify that the debugger is functioning properly. To use this board:

1. Disconnect the debugger from the target and the PC.
2. Connect the ICD 3 test interface board to the debugger using the modular cable.



3. Connect the debugger to the PC.
4. Select the MPLAB ICD 3 in-circuit debugger as either a debugger or programmer in MPLAB IDE. If you get a device ID mismatch, open the Settings dialog (*Debugger>Settings* or *Programmer>Settings*), go to the **Power** tab and be sure that "Power target circuit from ICD3" is unchecked.
5. Select *Debugger>Reconnect* if selected as a debugger and ignore the message "ICD3Err0045: You must connect to a target device to use MPLAB ICD 3." in the Output window.
6. Once again open the Settings dialog and on the **Status** tab click **Run ICD3 Test Interface**. A reminder to have the test board connected will pop up. Click **OK**.

If the test runs successfully, you'll see the following in the output window:

```
Test interface PGC clock line write succeeded.  
Test interface PGD data line write succeeded.  
Test interface PGC clock line read succeeded.  
Test interface PGD data line read succeeded.  
Test interface LVP control line test succeeded.  
Test interface MCLR level test succeeded.
```

Interface test completed successfully. Your MPLAB ICD 3 is functioning properly and ready to use.

If any test failed, please enter a ticket on <http://support.microchip.com/>, copying and pasting the content of the output window in the problem description.

A.8 TARGET BOARD CONSIDERATIONS

The target board should be powered according to the requirements of the selected device (2.0V-5.5V) and the application.

The debugger does sense target power. There is a 10K Ω load on VDD_TGT.

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

- **Section 2.5.2 “Target Connection Circuitry”**
- **Section 2.5.5 “Circuits That Will Prevent the Debugger From Functioning”**

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

ANSI

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

Application

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

Archive

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 lowercase 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 (PIC18 MCUs)

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

External Label

A label that has external linkage.

External Linkage

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

External Symbol

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

External Symbol Resolution

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

External Input Line

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

External RAM

Off-chip Read/Write memory.

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 `NOF` 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 `NOF` 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 multi-byte 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, KEELQ® 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 (PIC18 MCUs)

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

Non Real Time

Refers to the processor at a breakpoint or executing single-step instructions or MPLAB IDE 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 the 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.

Index

A		E	
Abort Operation.....	76	Erase.....	85
Animate.....	75	Erase All Before Programming.....	41
Application In/Out Window.....	83	Erase Flash Device.....	76
AVdd.....	20	Explorer 16 Demo Board.....	33
AVss.....	20	F	
B		Firmware	
Blank Check.....	85	Cancelled Download.....	62
Breakpoints		Disconnected while Downloading.....	62
Dialog.....	78	Firmware Downloads.....	87
Enabling.....	77	Freeze on Halt.....	60
Hardware.....	45, 73, 87	G	
Setup.....	45, 73, 76	General Corrective Actions.....	67
Software.....	54, 73, 87	H	
Build Configuration.....	24, 69	Halt.....	75
C		Hardware Breakpoints.....	45
Cables		Header Board	
Length.....	92, 95	Specification.....	10
Capacitors.....	20, 21	Hex File.....	41
CD-ROM.....	16	Hibernate mode.....	60, 61, 92
Circuits That Will Prevent the Debugger		Hubs, USB.....	92
From Functioning.....	21	I	
Code Protect.....	22	ICD 3 Test Interface Board.....	16
CodeGuard Security.....	89	ICD Headers.....	16
Command-line Programming.....	24	ICD3CMD.....	24
Configuration Bits.....	22, 30, 40	ICD3Err0001.....	63
Creating a Hex File.....	41	ICD3Err0002.....	63
Customer Notification Service.....	11	ICD3Err0003.....	63
Customer Support.....	12	ICD3Err0005.....	63
D		ICD3Err0006.....	63
Debug		ICD3Err0007.....	63
Executive.....	23	ICD3Err0008.....	63
Registers.....	24	ICD3Err0009.....	63
Debug Mode		ICD3Err0010.....	63
Sequence of Operations.....	22	ICD3Err0011.....	63
Debug Read.....	76	ICD3Err0012.....	64
Debug/Program Quick Reference.....	31	ICD3Err0013.....	64
Debugger Menu.....	44	ICD3Err0014.....	64
Debugging.....	44	ICD3Err0015.....	64
Demo Board.....	43	ICD3Err0016.....	64
Device Debug Resource Toolbar.....	73	ICD3Err0017.....	64
Documentation		ICD3Err0018.....	64
Conventions.....	9	ICD3Err0019.....	64
Layout.....	7	ICD3Err0020.....	64
Download Firmware.....	87	ICD3Err0021.....	64
Driver Board		ICD3Err0022.....	64
Standard.....	93	ICD3Err0023.....	64
Durability, Card Guide.....	92	ICD3Err0024.....	64

MPLAB® ICD 3 In-Circuit Debugger User's Guide

ICD3Err0025	64	MPLAB IDE	25
ICD3Err0026	64	P	
ICD3Err0027	64	PC, Power Down	60, 61, 92
ICD3Err0028	64	PGC	19, 20, 21, 22, 23
ICD3Err0029	65	PGD	19, 20, 21, 22, 23
ICD3Err0030	65	PIC24FJ128GA010, Tutorial	33
ICD3Err0031	65	PIM	18
ICD3Err0032	65	Port A	33
ICD3Err0033	65	Power-Down mode	60, 61, 92
ICD3Err0034	65	Processor Extension Kits	16
ICD3Err0035	65	Program	76, 85
ICD3Err0036	65	Program Memory Tab	41
ICD3Err0037	65	Programming	55
ICD3Err0038	65	Command-line	24
ICD3Err0039	65	Production	15, 24
ICD3Err0040	65	Programming Options	41
ICD3Err0041	66	Project Wizard	30, 37
ICD3Err0045	66	Pull-ups	21
ICD3Err0046	66	Q	
ICD3Err0047	66	Quick Reference	
ICD3Err0052	66	Debug/Program	31
ICD3Err0053	66	R	
ICD3Err0054	66	Read	76, 85
ICD3Err0055	66	Reading, Recommended	10
ICD3Err0056	66	Readme	10
ICD3Err0057	66	Reconnect	76
ICD3Err0063	66	Reserved Resources by Device	24
ICD3Err0064	66	Reset	
ICD3Err0065	66	Processor	76
ICD3Err0066	66	Resistors	21
ICD3Err0067	67	Run	75
ICD3Err0068	67	Running code	44
ICD3Err0069	67	S	
ICD3Err0070	67	Secure Segments	89
ICD3Err0071	67	Selecting Device and Development Mode	34
ICD3Err0072	67	Set a Breakpoint	45
ICD3Err0073	67	Setting Program and Debug Options	42
ICD3Err0082	67	Setting Up Hardware and Software	34
ICD3Err0083	67	Software Breakpoints	54
ICD3Info0001	67	SQTP	31
ICSP	22, 23, 24, 93	Standard Communication	
ICSPCLK	93	Connections	19
ICSPDAT	93	Driver Board	93
Indicator Lights	92	Standard ICSP Device Communication	18
Information Messages	67	Step	75
Internet Address	11	Stopwatch	53, 73
K		T	
Kit Components	16	Table Read Protect	22
L		Target Connection	
LEDs	33, 92	Circuitry	19
Light Icons	34	Improper Circuits	21
Loading Program and Debug Code	43	Standard	19
M		Target Device	22
Microchip Internet Web Site	11	Timer1	33
Modular Interface Cable	22	Toolbar Buttons	44
MPLAB C30	37	Transition Socket	16
MPLAB ICD 3 Defined	15	Specification	10, 27
MPLAB ICD 3 Test Interface Board	95		

Tutorial 33

U

USB.....92, 115

 Cables..... 16

 Device Drivers 25

 Hubs 92

V

Vcap..... 20

Vdd..... 19, 20, 21, 22

Verify 85

Vpp..... 19, 20, 21, 22, 23

Vss..... 19, 20, 21, 22

W

Warranty Registration 10

Watch Window 46

Watchdog Timer.....22, 116

WWW Address..... 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

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 - Chongqing
Tel: 86-23-8980-9588
Fax: 86-23-8980-9500

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 - Xian
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

China - Xiamen
Tel: 86-592-2388138
Fax: 86-592-2388130

China - Zhuhai
Tel: 86-756-3210040
Fax: 86-756-3210049

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-3090-4444
Fax: 91-80-3090-4123

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-2566-1512
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: 886-7-213-7830
Fax: 886-7-330-9305

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