MPLAB® PICkit™ Basic In-Circuit Debugger User's Guide

Notice to Development Tools Customers

Important:
All documentation becomes dated, and Development Tools manuals are no exception. Our 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 website (www.microchip.com/) to obtain the latest version of the PDF document.

Documents are identified with a DS number located on the bottom of each page. The DS format is DS<DocumentNumber><Version>, where <DocumentNumber> is an 8-digit number and <Version> is an uppercase letter.

For the most up-to-date information, find help for your tool at onlinedocs.microchip.com/.

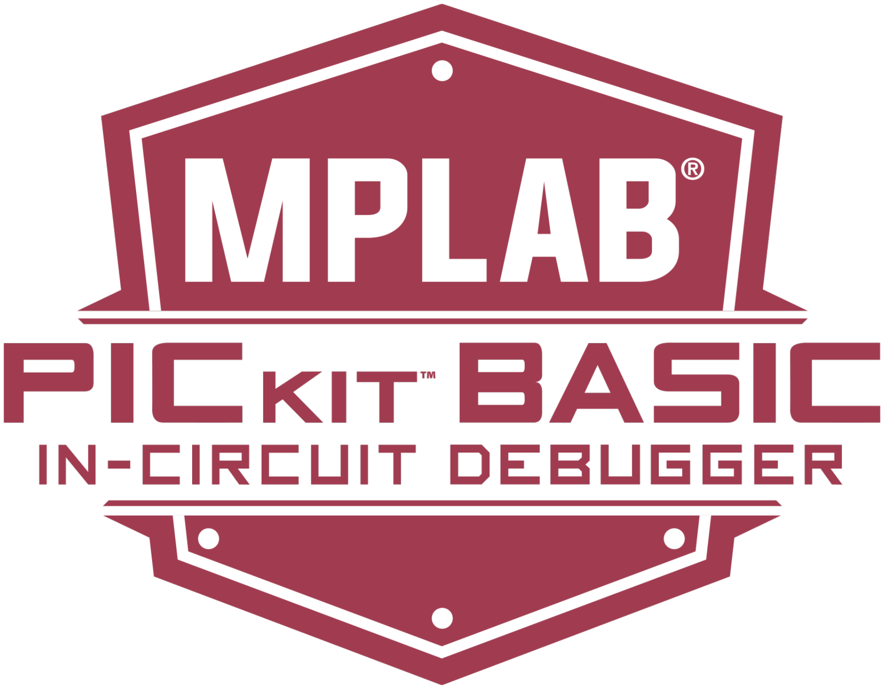


Table of Contents

Notice to Development Tools Customers....1

1. Introduction....4
    1.1. Conventions Used in This Guide....4
    1.2. Recommended Reading....5

2. About the Debugger....6
    2.1. MPLAB PICkit Basic In-Circuit Debugger Advantages....6
    2.2. MPLAB PICkit Basic In-Circuit Debugger Components....8
    2.3. MPLAB PICkit Basic In-Circuit Debugger Block Diagram....9
    2.4. Using MPLAB PICkit Basic In-Circuit Debugger with MPLAB X IDE and MPLAB IPE....9

3. Connections....11
    3.1. Power Connections....11
    3.2. PC Connections....11
    3.3. Target Connections....11

4. Operation....26
    4.1. Quick Debug/Program Reference....26
    4.2. Operational Overview....26
    4.3. About High Voltage....27
    4.4. SAM and PIC32C Arm Devices - On-Chip Debugging....28
    4.5. AVR Devices - On-Chip Debugging (OCD)....28
    4.6. PIC32M MCU - On-Chip Debugging....36
    4.7. PIC MCU/dsPIC DSC - On-Chip Debugging....36

5. Troubleshooting....45
    5.1. Some Questions to Answer First....45
    5.2. Top Reasons Why You Can't Debug....45
    5.3. General....46
    5.4. How to Use the Hardware Tool Emergency Boot Firmware Recovery Utility....46

6. Frequently Asked Questions....48
    6.1. How Does it Work?....48
    6.2. What's Wrong?....48

7. Error Messages....50
    7.1. Types of Error Messages....50
    7.2. General Corrective Actions....57

8. Debugger Function Summary....60
    8.1. Debugger Selection and Switching....60
    8.2. Debugger Options Selection....60
    8.3. Memories to Program....61
    8.4. Debug Options....61
    8.5. Program Options....62
    8.6. PICkit Basic Tool Options....62
    8.7. Freeze Peripherals....62

8.8. Secure Segment........ 63
8.9. Clock........ 63
8.10. Tool Pack Selection........ 63
8.11. Communication........ 63
8.12. Event Recorder........ 63

9. Hardware Specification........ 65
9.1. USB Connector Specifications........ 65
9.2. MPLAB PICkit Basic In-Circuit Debugger........ 65
9.3. Communication Hardware........ 66
9.4. Target Board Considerations........ 67

10. Revision History........ 68
10.1. Revision A (February 2025)........ 68

11. Glossary........ 69

12. Support........ 86
12.1. Warranty Registration........ 86
12.2. myMicrochip Personalized Notification Service........ 86

Microchip Information........ 87
Trademarks........ 87
Legal Notice........ 87
Microchip Devices Code Protection Feature........ 87

1. Introduction

The MPLAB® PICkit™ Basic In-Circuit Debugger (PG164110) is an ultra-low priced debugging solution for projects not requiring high-voltage programming or advanced debug features. Therefore, it supports many of Microchip's newer MCU offerings but not some legacy products. With a nominal feature set, the debugger is geared toward developers who don't require advanced features. ***It is not intended for production programming.***

Important: For low pin count AVR devices with UPDI, MPLAB PICkit Basic cannot generate the high voltage pulse to reactivate the UPDI interface if the UPDI pin is configured as GPIO or RESET by configuring the RSTPINCFG configuration bits. A different tool will need to be used to do this, such as the MPLAB PICkit 5.

1.1 Conventions Used in This Guide

The following conventions may appear in this documentation. In most cases, formatting conforms to the *OASIS Darwin Information Typing Architecture (DITA) Version 1.3 Part 3: All-Inclusive Edition*, 19 June 2018.

Table 1-1. Documentation Conventions

Description	Implementation	Examples
References	DITA: cite	*MPLAB® PICkit™ Basic In-Circuit Debugger User's Guide.*
Emphasized text	Italics	...is the *only* compiler...
A window, window pane or dialog name.	DITA: wintitle	the **Output** window. the **New Watch** dialog.
A field name in a window or dialog.	DITA: uicontrol	Select the **Optimizations** option category.
A menu name or item.	DITA: uicontrol	Select the **File** menu and then **Save**.
A menu path.	DITA: menucascade, uicontrol	**File** > **Save**
A tab	DITA: uicontrol	Click the **Power** tab.
A software button.	DITA: uicontrol	Click the **OK** button.
A key on the keyboard.	DITA: uicontrol	Press the **F1** key.
File names and paths.	DITA: filepath	`C:/Users/User1/Projects`
Source code: inline.	DITA: codeph	Remember to `#define START` at the beginning of your code.

Source code: block.	DITA: codeblock	An example is:
		`#include <xc.h>`
		`main(void) {`
		`  while(1);`
		`}`

User-entered data.	DITA: userinput	Type in a device name, for example `PIC18F47Q10`.
Keywords	DITA: codeph	`static`, `auto`, `extern`
Command-line options.	DITA: codeph	`-Opa+, -Opa-`
Bit values	DITA: codeph	`0, 1`
Constants	DITA: codeph	`0xFF, 'A'`
A variable argument.	DITA: codeph + option	*`file`*`.o`, where *`file`* can be any valid file name.
Optional arguments	Square brackets [ ]	`xc8 [`*`options`*`]` *`files`*

Table 1-1. Documentation Conventions (continued)

Description	Implementation	Examples
Choice of mutually exclusive arguments; an OR selection.	Curly brackets and pipe character: { | }	`errorlevel {0|1}`
Replaces repeated text.	Ellipses...	`var_name [, var_name...]`

Represents code supplied by user.	Ellipses...	`void main (void)`
		`{ ...`
		`}`

A number in verilog format, where N is the total number of digits, R is the radix and n is a digit.	N'Rnnnn	4'b0010, 2'hF1
Device Dependent insignia. Specifies that a feature is not supported on all devices. Please see your device data sheet for details.	[DD]	Assembler Special Operators

1.2 Recommended Reading

This user's guide describes how to use MPLAB PICkit Basic In-Circuit Debugger. Other useful documents are listed below. The following Microchip documents are available and recommended as supplemental reference resources.

Development Tools Design Advisory (DS-50001764)

Please read this first! This document contains important information about operational issues that should be considered when using the MPLAB PICkit Basic In-Circuit Debugger with your target design.

MPLAB X IDE Online Help

This is an essential document to be used with any Microchip hardware tool.

This is an extensive help file for the MPLAB X IDE. It includes an overview of embedded systems, installation requirements, tutorials, details on creating new projects, setting build properties, debugging code, setting configuration bits, setting breakpoints, programming a device, etc. This help file is generally more up-to-date than the printable PDF of the user's guide (DS-50002027) available as a free download at the MPLAB X IDE webpage.

Release Notes for MPLAB PICkit Basic

For the latest information on using MPLAB PICkit Basic In-Circuit Debugger, find the release notes in MPLAB X IDE under **Help** > **Release Notes**. The release notes contain update information and known issues that may not be included in this user's guide.

MPLAB® PICkit™ Basic In-Circuit Debugger Quick Start Guide (DS-5000xxxx)

This reference material shows you how to connect hardware and install software for the MPLAB PICkit Basic In-Circuit Debugger.

2. About the Debugger

The MPLAB® PICkit™ Basic In-Circuit Debugger (PG164110) allows affordable, fast and easy debugging and programming using the powerful graphical user interface of MPLAB® X IDE (Integrated Development Environment) or MPLAB IPE (Integrated Programming Environment). Supported device include:

- PIC® and AVR® microcontrollers (MCUs)
- dsPIC® digital signal controllers (DSCs)
- SAM (Arm® Cortex®-based) MCUs and microprocessors (MPUs)
- CEC (Arm Cortex-based) MCUs

> **Important:** Because MPLAB PICkit Basic does not support high voltage programming, devices that require high voltage to program are not supported. See the MPLAB PICkit Basic programmer (PKBP) and debugger (PKBD) columns in the Device Support List to find out which devices are supported by MPLAB PICkit Basic.

The MPLAB PICkit Basic connects to the computer using a high-speed USB 2.0 interface and connects to the target via a Microchip debug 8-pin Single In-Line (SIL) connector. MPLAB PICkit Basic has all the speed and entry-level features you need to quickly debug your prototype.

The MPLAB PICkit Basic features a powerful 32-bit 300 MHz SAM E70 Arm Cortex-M7 based MCU for quicker debug iterations. Along with its support for a wide target voltage, the MPLAB PICkit Basic supports interfaces such as 4-wire JTAG and Serial Wire Debug. It is also backward compatible for demo boards and target systems using 2-wire JTAG and In-Circuit Serial Programming™.

The debugger system executes code like an actual device because it uses the target device's built-in emulation circuitry, instead of a special debugger chip. All available features of a given device are accessible interactively and can be set and modified by the MPLAB X IDE interface.

The MPLAB PICkit Basic is compatible with any of these platforms:

- Microsoft Windows® OS
- Linux® OS
- macOS®

See the release notes for versions supported.

2.1 MPLAB PICkit Basic In-Circuit Debugger Advantages

The MPLAB® PICkit™ Basic In-Circuit Debugger provides the following advantages.

Features/Capabilities:

- Connects to computer via high-speed USB 2.0 (480 Mbits/s) cable.
- Powered through USB cable. Cannot power target; target must be powered from its own power supply.
- An 8-pin SIL programming connector and the option to use various interfaces.
- Programs devices using MPLAB X IDE or MPLAB IPE.

- Works with many Microchip PIC, dsPIC, AVR, or DSC devices, including 32-bit microcontrollers such as SAM, CEC and PIC32 devices. For details see the MPLAB X IDE Device Support List.
- Supports 4-wire JTAG, Serial Wire Debug, UPDI, PDI, SPI programming, debugWIRE and TPI programming.
- Backward compatibility for demo boards and target systems using 2-wire JTAG and ICSP (In-Circuit Serial Programming).
- Supports multiple hardware and software breakpoints, stopwatch and source code file debugging.
- Debugs your application on your own hardware in real time.
- Sets breakpoints based on internal events.
- Debugs at full target MCU speed.
- Configures pin drivers.
- Adds new device support and features by installing the latest version of MPLAB X IDE (available as a free download at http://www.microchip.com/mplabx/).
- Indicates debugger status via the Active and Status LEDs.

Performance/Speed:

- No firmware download delays incurred when switching devices.
- 32-bit microcontroller using an Arm® Cortex®-M7 core running at 300 MHz.

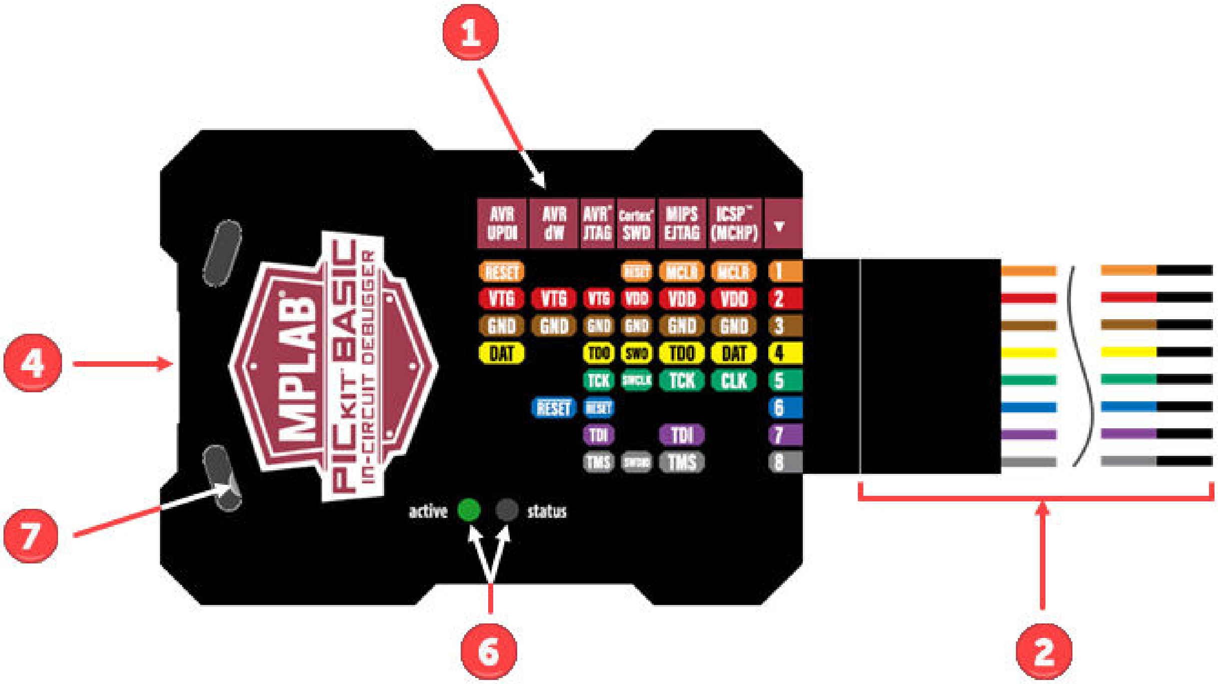
Safety:

- RoHS, CE and China E compliant.
- Supports target supply voltages from 1.2V to 5.0V +/-10%.
  Note: PICkit Basic cannot power the target.

2.2 MPLAB PICkit Basic In-Circuit Debugger Components

The components of the MPLAB PICkit Basic In-Circuit Debugger system are show in the figure and described below.

Figure 2-1. MPLAB PICkit Basic In-Circuit Debugger



1. MPLAB PICkit Basic enclosure with a color-coded signals label on the front. This includes a pin 1 specifier ▼.
2. 8-pin SIL connector with color-coded wires on the bottom of the enclosure. These correspond to the color-coded signals label on the front.
3. Alternately, you could use the 8-pin to 10-pin ARM SWD Adapter Board.
4. USB Type-C® connector on the top of the enclosure.
5. Use the included full-featured USB Type-C® cable (data and power) to connect to a computer.
6. Two status LEDs on the front of the enclosure.
7. Emergency Recovery button accessible through the left side of the enclosure.

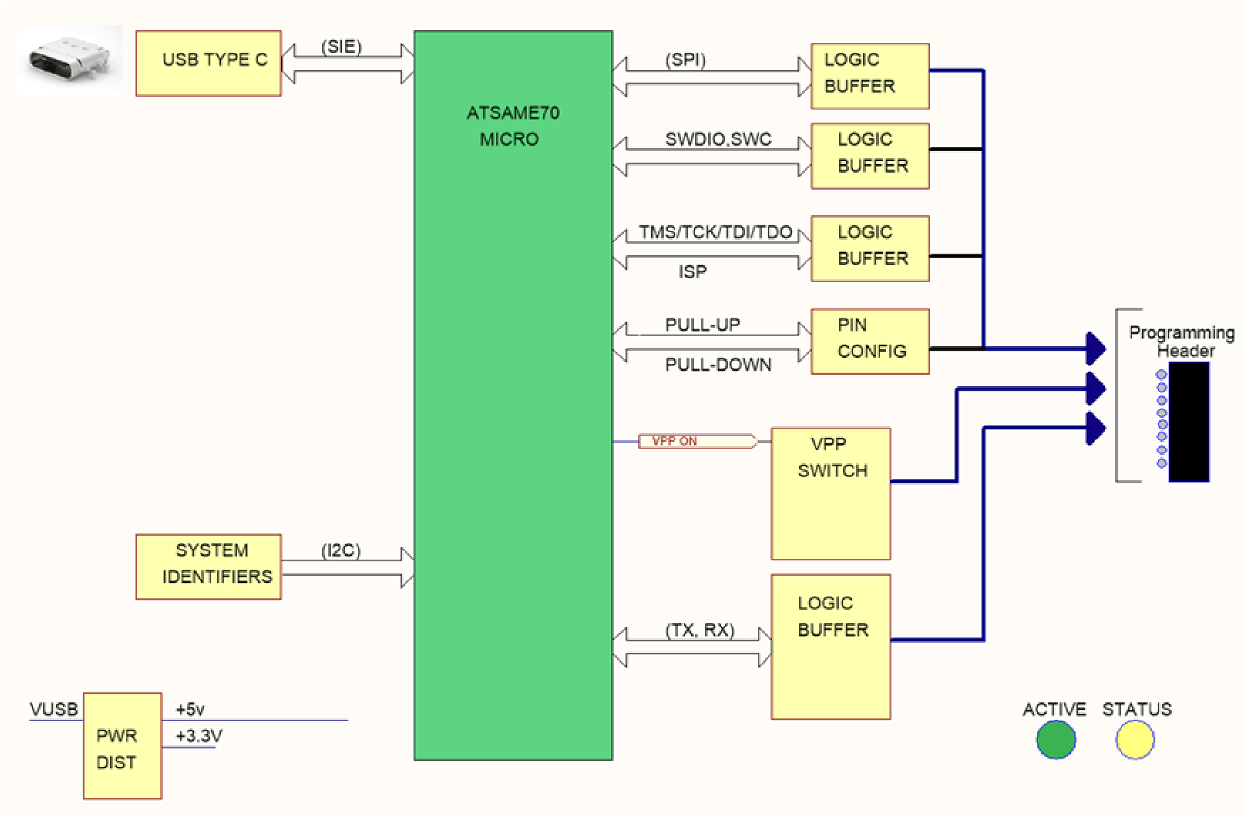
To use the MPLAB PICkit Basic, you will need to supply:

- Target board.
- Power supply for target board.
- Any wiring interfaces or cables needed for your application. Some available adapters and cables include the AC164110 RJ-11 to ICSP Adapter.

Additional hardware and accessories may be ordered separately from Microchip Direct .

- Debugger Adapter Board (Part Number AC002015) - a connectivity board that supports JTAG, SWD and ICSP protocols, useful for debugging AVR® with MPLAB PICkit Basic.

2.3 MPLAB PICkit Basic In-Circuit Debugger Block Diagram



2.4 Using MPLAB PICkit Basic In-Circuit Debugger with MPLAB X IDE and MPLAB IPE

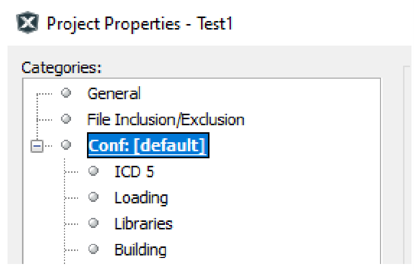
Download and install the latest version of MPLAB® X IDE from the MPLAB X IDE webpage. The MPLAB X IDE installer will install MPLAB X IDE and/or MPLAB IPE.

Using MPLAB® PICkit™ Basic with MPLAB X IDE

The MPLAB PICkit Basic In-Circuit Debugger works with MPLAB X IDE to develop target applications. The *MPLAB® X IDE User's Guide* (DS-50002027) and other documentation may be found on the MPLAB X IDE webpage. Alternately see the MPLAB X IDE WebHelp.

Table 2-1. MPLAB X IDE Overview

MPLAB X IDE MPLAB X IDE vX.XX	Use the desktop icon to launch the IDE.
📁 📂	Create a new project or open an existing project. Select MPLAB PICkit Basic as the hardware tool.

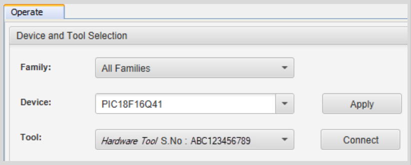
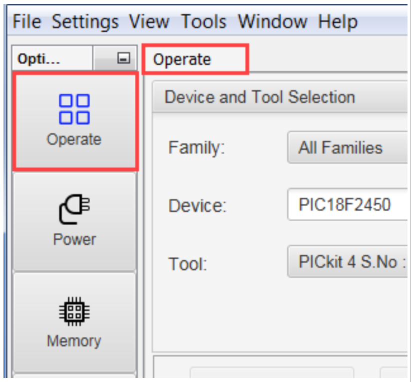
 Project Properties - Test1 Categories: General File Inclusion/Exclusion Conf: [default] ICD 5 Loading Libraries Building	Open the **Project Properties** window by right clicking on the project name and selecting **Properties**. This window is used to set up options for debugging, programming and other features. See MPLAB PICkit Basic option descriptions.

Using MPLAB PICkit Basic with MPLAB IPE

The MPLAB PICkit Basic In-Circuit Debugger works with MPLAB IPE as a programmer. The *MPLAB IPE User's Guide* (DS-50002227) and other documentation may be found on the MPLAB X IDE webpage. Alternately see the MPLAB IPE WebHelp.

There are also command line IPE tools available as an option. See the MPLAB X IDE installation folder, `docs` subfolder.

Table 2-2. MPLAB IPE Overview

	Use the desktop icon to launch the IPE.
 Family: All Families Device: PIC18F16Q41 Tool: *Hardware Tool* S.No : ABC123456789	Select a device to program and then select MPLAB PICkit Basic as the tool.
 File Settings View Tools Window Help Opti... Operate Operate Power Memory Device and Tool Selection Family: All Families Device: PIC18F2450 Tool: PICkit 4 S.No :	Select on a button to **Program**, **Erase**, **Read**, **Verify** or **Blank Check**. For more on MPLAB IPE, including Advanced mode, see the *MPLAB IPE User's Guide*.

3. Connections

The MPLAB® PICkit™ Basic In-Circuit Debugger hardware setup begins by connecting power, communications, and targets to the debugger. See the following sections for details.

3.1 Power Connections

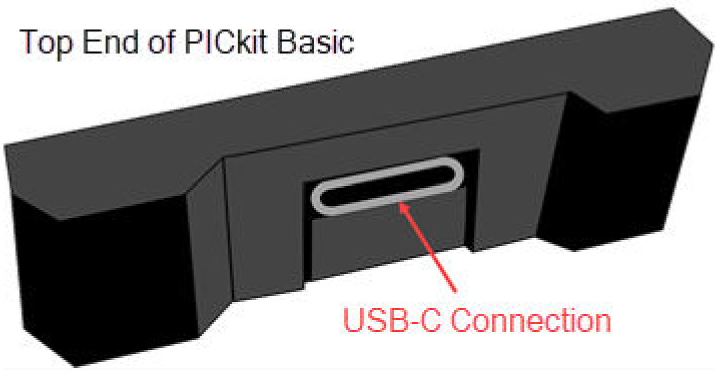
The MPLAB PICkit Basic In-Circuit Debugger is powered through its USB Type-C® connector. See also 9.1. USB Connector Specifications.

MPLAB PICkit Basic cannot power the target. The target board must be powered from its own supply.

3.2 PC Connections

MPLAB® PICkit™ Basic In-Circuit Debugger can connect with the PC (and MPLAB X IDE) using USB (see figure below). It is recommended that you use the cable that comes with the kit to avoid communication issues.

Figure 3-1. USB Power and Communication



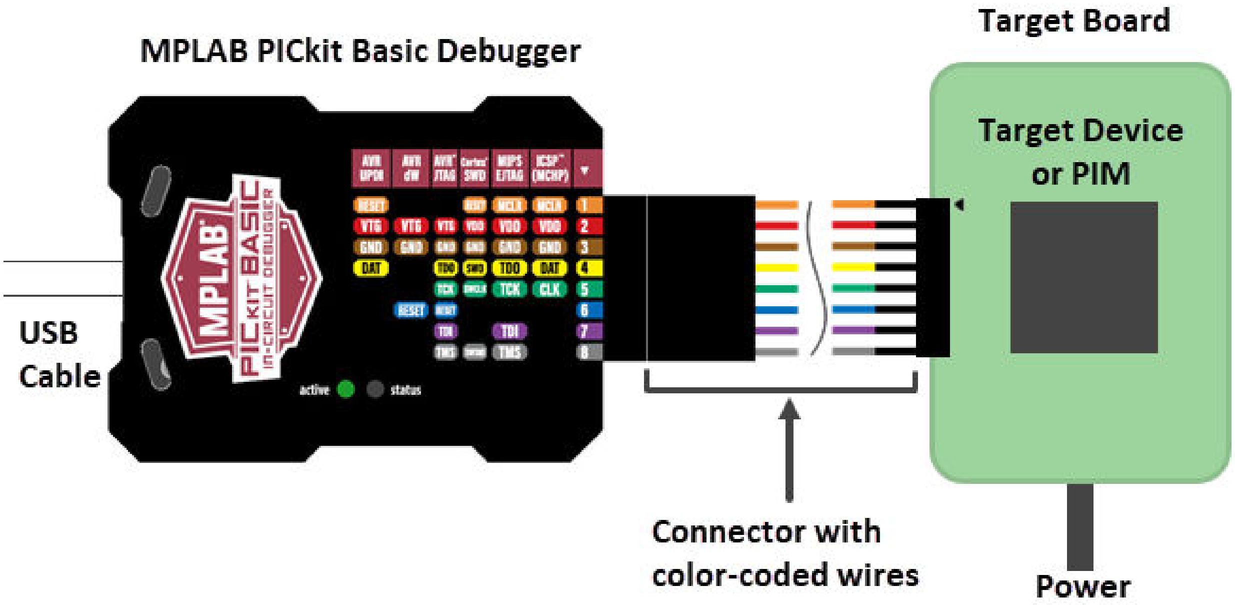
Connection Type	Connection Details	Programming[1]	Debugging[1]	MPLAB Data Visualizer Support
USB Type-C®	HS USB 2.0	Yes	Yes	Yes
1. For speed information, see 9.2.1. Board Specifications.				

3.3 Target Connections

MPLAB® PICkit™ Basic In-Circuit Debugger connects to a target via an 8-pin single inline (SIL) connector with color-coded wires (leads). Make sure to align the Pin 1 on the debugger (▼) to Pin 1 on the target (see figure below). Alternately, you can use the included Arm/SWD adapter board depending on your target.

For other target connections, an optional adapter board is available for sale. Connector and adapter board pin-outs are shown in the following sections.

Figure 3-2. Connection to Target


MPLAB PICkit Basic Debugger
Target Board
Target Device or PIM
USB Cable
MPLAB PICkit BASIC IN-CIRCUIT DEBUGGER
active
status
Connector with color-coded wires
Power

3.3.1 Target Connection Pinout

The programming connector pin functions are different for various devices and interfaces. Refer to the following pinout tables for debug and data stream interfaces.
Note: Refer to the data sheet for the device you are using as well as the application notes for the specific interface for additional information and diagrams.

Table 3-1. Pinouts for Debug Interfaces

MPLAB® PICkit™ Basic Connector		DEBUG INTERFACE									Target[4] Connector	
8-Pin SIL[1]											8-Pin SIL	6-Pin SIL
Pin #	Pin Name	ICSP™ (MCHP)	MIPS EJTAG	Cortex® SWD	AVR® JTAG	AVR debugWIRE	AVR UPDI	AVR PDI	AVR ISP	AVR TPI	Pin #	Pin #
1	TVPP	$\overline{\text{MCLR}}$/ VPP	$\overline{\text{MCLR}}$	$\overline{\text{RESET}}$			$\overline{\text{RESET}}$[3]				1	1
2	TVDD	VDD	VDD/ VDDIO	VDD	VTG	VTG	VTG	VTG	VTG	VTG	2	2
3	GND	GND	GND	GND	GND	GND	GND	GND	GND	GND	3	3
4	PGD	DAT	TDO	SWO[2]	TDO		DAT[3]	DAT	MISO	DAT	4	4
5	PGC	CLK	TCK	SWCLK	TCK				SCK	CLK	5	5
6	TAUX				$\overline{\text{RESET}}$	$\overline{\text{RESET}}$/ dW		CLK	$\overline{\text{RESET}}$	$\overline{\text{RESET}}$	6	6
7	TTDI		TDI		TDI				MOSI		7	
8	TTMS		TMS	SWDIO[2]	TMS						8	

1. Use of a 6-pin header will result in the loss of functions on Pins 7 and 8 affecting EJTAG, JTAG, SWD and ISP.
2. SWO is used for trace (see release notes for trace support) SWDIO is used for debug.

3. Pin may be used for High-Voltage Pulse reactivation of UPDI function depending on device. Usually a low pin count AVR device (see device data sheet for details). However, MPLAB PICkit Basic **cannot generate the high voltage pulse** to reactivate the UPDI interface if the UPDI pin is configured as GPIO or RESET by configuring the RSTPINCFG configuration bits. A different tool will need to be used to do this, such as the MPLAB PICkit™ 5.
4. These are example target connectors that are assumed similar to the debugger unit (SIL).

Table 3-2. Pinouts for Data Stream Interfaces

MPLAB® PICkit™ Basic 8-Pin SIL Connector	DATA STREAM	Target 8-Pin SIL Connector
Pin #	UART / CDC[1]	Pin #
1		1
2	VTG	2
3	GND	3
4		4
5		5
6		6
7	TX (target)	7
8	RX (target)	8

[1] See “Readme for PICkit Basic” for more information.

3.3.2 8-pin to 10-pin Arm SWD Adapter Board

An 8-pin to 10-pin Arm SWD Adapter Board comes in the MPLAB® PICkit™ Basic In-Circuit Debugger box. Use the 8-pin connector for the debugger and the 10-pin connector with the provided 12 cm flat cable to connect to an Arm target for use with SWD.

Figure 3-3. 8-pin to 10-pin Adapter Board

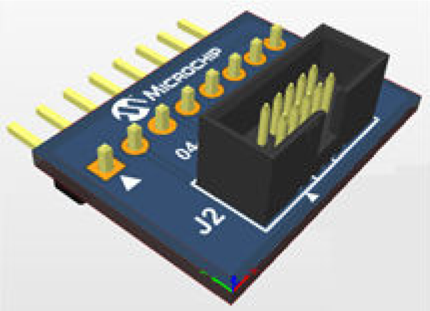
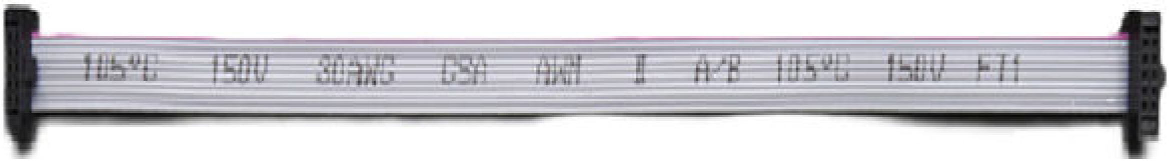


Figure 3-4. Adapter 10-Pin Flat Cable



Related Links

9.3.2. Arm/SWD Adapter Board Schematics

3.3.3 Debugger Adapter Board

For legacy connections, the Debugger Adapter Board (AC102015) is available.

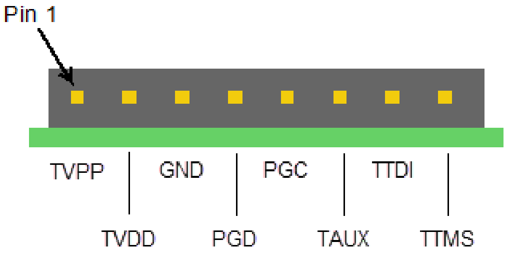
The MPLAB PICkit Basic can be connected to the Debugger Adapter Board as shown in the sections below. Then the various connections on the adapter board can be used to connect to specific targets (see the following topic).

Alternatively the AVR Programming Adapter Board (AC31S18A) is available for use with MPLAB PICkit 5, PICkit 4, PICkit Basic and Snap.

Using the 8-pin Inline Connector

Using the Single In-Line (SIL) connector, the tool is connected directly to the adapter board (ensure pin 1 on the tool lines up with pin 1 on the adapter board). If a 6-pin SIL header is used, connections 7 (TMS) and 8 (TDI) will not be available.

Figure 3-5. SIL Connection at Debugger Adapter Board

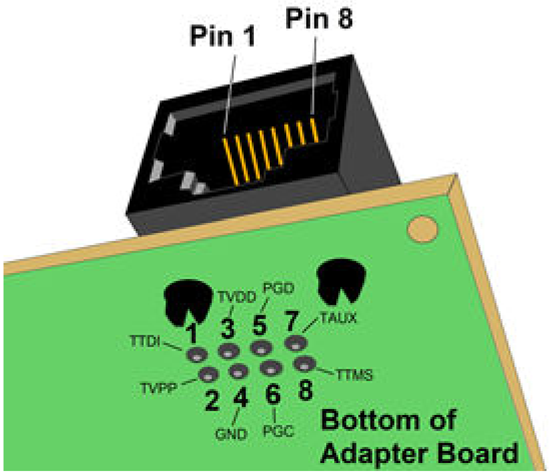


Using the 8-pin Modular Connector

For the MPLAB PICkit Basic it is recommended that you use the 8-pin inline connector described above.

Using the AC164110 - RJ-11 to ICSP Adapter, the debugger can be connected to the modular connector using a 6-pin modular cable resulting in the loss of connection to pins 8 (TMS) and 1 (TDI) at the adapter board.

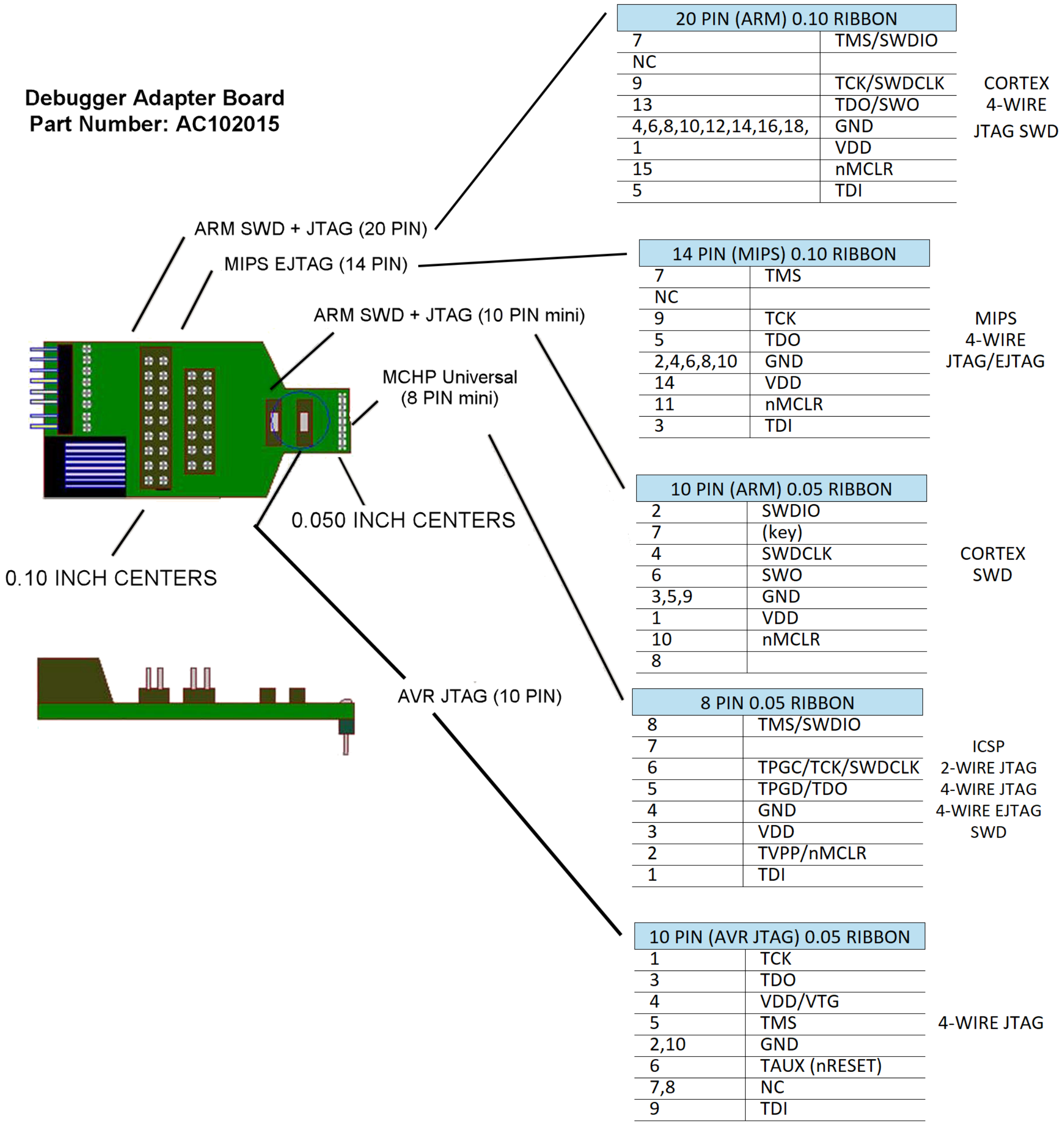
Figure 3-6. RJ-45 at the Debugger Adapter Board



3.3.3.1 Adapter Board Pinout

This is a connectivity board that supports JTAG, SWD, ICSP and AVR protocols.

Figure 3-7. MPLAB PICkit Basic Adapter Board (AC102015) Pinouts



20 PIN (ARM) 0.10 RIBBON	
7	TMS/SWDIO
NC	
9	TCK/SWDCLK
13	TDO/SWO
4,6,8,10,12,14,16,18,	GND
1	VDD
15	nMCLR
5	TDI

14 PIN (MIPS) 0.10 RIBBON	
7	TMS
NC	
9	TCK
5	TDO
2,4,6,8,10	GND
14	VDD
11	nMCLR
3	TDI

10 PIN (ARM) 0.05 RIBBON	
2	SWDIO
7	(key)
4	SWDCLK
6	SWO
3,5,9	GND
1	VDD
10	nMCLR
8	

8 PIN 0.05 RIBBON	
8	TMS/SWDIO
7	
6	TPGC/TCK/SWDCLK
5	TPGD/TDO
4	GND
3	VDD
2	TVPP/nMCLR
1	TDI

10 PIN (AVR JTAG) 0.05 RIBBON	
1	TCK
3	TDO
4	VDD/VTG
5	TMS
2,10	GND
6	TAUX (nRESET)
7,8	NC
9	TDI

3.3.4 SAM/PIC32C MCUs - JTAG/SWD Connections

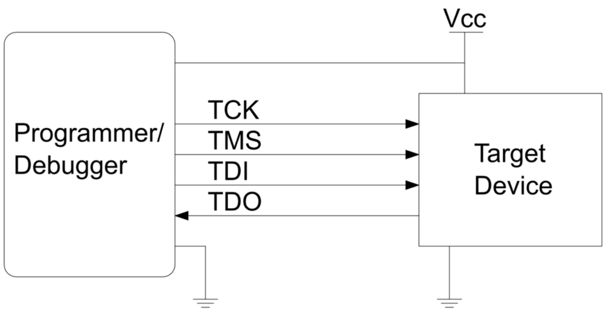
All SAM/PIC32C Arm®-based devices feature the Serial Wire Debug (SWD) interface for programming and debugging. In addition, some devices feature a JTAG interface with identical functionality. Check the device data sheet for supported interfaces of that device.

3.3.4.1 JTAG Physical Interface

The JTAG interface consists of a four-wire Test Access Port (TAP) controller that is compliant with the IEEE® 1149.1 standard. The IEEE standard was developed to provide an industry-standard way to efficiently test circuit board connectivity (Boundary Scan). Microchip AVR and SAM devices have extended this functionality to include full Programming and On-chip Debugging support.

To use this target interface with MPLAB X IDE, open the **Project Properties** window, **PICkit Basic** category, **Communications** option category, and select **4-wire JTAG**.

Figure 3-8. JTAG Interface Basics



3.3.4.1.1 Connecting to a SAM/PIC32C JTAG Target

The MPLAB PICkit Basic provides a direct connection for new designs or a legacy 10-pin 50-mil JTAG connection as well as a legacy 20-pin 100-mil JTAG connection using the adapter board.

Related Links

3.3.3.1. Adapter Board Pinout

3.3.4.1.2 SAM/PIC32C JTAG Pinout (Cortex$^{®}$-M debug connector)

SAM/PIC32C JTAG pin names and descriptions are shown in the table below. Pin numbers are shown for MPLAB PICkit Basic direct connection and Debugger Adapter Board 10-pin and 20-pin connections.

Table 3-3. SAM/PIC32C JTAG Pins and Descriptions

MPLAB PICkit Basic Pin	Adapter Board (10-Pin) Pin	Adapter Board (20-Pin) Pin	Name	Description
8	2	7	TMS	Test Mode Select (control signal from the MPLAB PICkit Basic into the target device).
6	7	2,3,9,11,17,19	NC/AUX	Recommended as not connected.
5	4	9	TCK	Test Clock (clock signal from the MPLAB PICkit Basic into the target device).
4	8	5	TDO	Test Data Out (data transmitted from the target device into the MPLAB PICkit Basic).
3	3, 5, 9	4,6,8,10,12,14,16,18,20	GND	Ground. All must be connected to ensure that the MPLAB PICkit Basic and the target device share the same ground reference.
2	1	1	VDD\VTG*	VDD: MPLAB PICkit Basic providing power to target (optional) or target providing power to MPLAB PICkit Basic (PTG) VTG: Voltage target reference. The MPLAB PICkit Basic samples the target voltage on this pin in order to power the level converters correctly. The MPLAB PICkit Basic draws less than 3mA from this pin in this mode.
1	10	15	$\overline{\text{MCLR}}$	Reset (optional). Used to reset the target device. Connecting this pin is recommended since it allows the MPLAB PICkit Basic to hold the target device in a reset state, which can be essential to debugging in certain scenarios.
7	6	13	TDI	Test Data In (data transmitted from the MPLAB PICkit Basic into the target device).
* Remember to include a decoupling capacitor between this pin and GND. See AN4451: SAMA5D2 Hardware Design Considerations.				

3.3.4.2 SAM/PIC32C SWD Interface

The Arm® SWD interface is a subset of the JTAG interface, making use of TCK and TMS pins.

3.3.4.2.1 Connecting to a SAM/PIC32C SWD Target

The MPLAB PICkit Basic provides a direct connection for new designs or a legacy 10-pin 50-mil SWD connection as well as a legacy 20-pin 100-mil SWD connection using the adapter board.

Related Links

3.3.3.1. Adapter Board Pinout

3.3.4.2.2 SAM/PIC32C SWD Pinout

SAM/PIC32C SWD pin names and descriptions are shown in the table below. Pin numbers are shown for MPLAB PICkit Basic direct connection and Debugger Adapter Board 10-pin and 20-pin connections.

Table 3-4. SAM/PIC32C SWD Pins and Descriptions

MPLAB PICkit Basic Pin	Adapter Board (10-Pin) Pin	Adapter Board (20-Pin) Pin	Name	Description
8	2	7	SWDIO	Serial Wire Debug Data Input/Output.
5	4	9	SWDCLK	Serial Wire Debug Clock.
4	8	5	SWO	Serial Wire Output (used with ITM - not implemented on all devices).
3	3, 5, 9	4,6,8,10,12,14,16,18,20	GND	Ground
2	1	1	VDD\VTG*	VDD: MPLAB PICkit Basic providing power to target (optional) or target providing power to MPLAB PICkit Basic (PTG) VTG: Target voltage reference. The MPLAB PICkit Basic samples the target voltage on this pin in order to power the level converters correctly. The MPLAB PICkit Basic draws less than 1mA from this pin in this mode.
1	10	15	$\overline{\text{MCLR}}$	Reset (optional). Used to reset the target device. Connecting this pin is recommended since it allows the MPLAB PICkit Basic to hold the target device in a reset state, which can be essential to debugging in certain scenarios.
* Remember to include a decoupling capacitor between this pin and GND. See AN4451: SAMA5D2 Hardware Design Considerations.				

3.3.5 AVR MCUs - Various Connections

AVR devices feature various programming and debugging interfaces. Check the device data sheet for supported interfaces of that device.

3.3.5.1 AVR MCUs - JTAG Connections

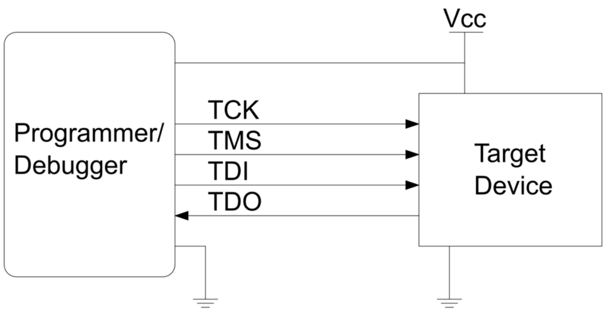
Some AVR devices feature a JTAG interface for programming and debugging. Check the device data sheet for supported interfaces of that device.

3.3.5.1.1 JTAG Physical Interface

The JTAG interface consists of a four-wire Test Access Port (TAP) controller that is compliant with the IEEE® 1149.1 standard. The IEEE standard was developed to provide an industry-standard way to efficiently test circuit board connectivity (Boundary Scan). Microchip AVR and SAM devices have extended this functionality to include full Programming and On-chip Debugging support.

To use this target interface with MPLAB X IDE, open the **Project Properties** window, **PICkit Basic** category, **Communications** option category, and select **4-wire JTAG**.

Figure 3-9. JTAG Interface Basics



Connecting to an AVR JTAG Target

The MPLAB PICkit Basic provides a direct connection for new designs or a legacy 10-pin 50-mil JTAG connection using the adapter board.

Related Links

3.3.3.1. Adapter Board Pinout

AVR JTAG Pinout

AVR JTAG pin names and descriptions are shown in the table below. Pin numbers are shown for MPLAB PICkit Basic direct connection and Debugger Adapter Board 10-pin connection.

Table 3-5. AVR JTAG Pins and Descriptions

MPLAB PICkit Basic Pin	Adapter Board AVR JTAG Pin	Name	Description
1	7, 8	NC	Not Connected.
2	4	VDD\VTG*	VDD: MPLAB PICkit Basic providing power to target (optional) or target providing power to MPLAB PICkit Basic (PTG) VTG: Target voltage reference. The MPLAB PICkit Basic samples the target voltage on this pin in order to power the level converters correctly. The MPLAB PICkit Basic draws less than 3mA from this pin in this mode.
3	2, 10	GND	Ground. All must be connected to ensure that the MPLAB PICkit Basic and the target device share the same ground reference.
4	3	TDO	Test Data Out (data transmitted from the target device into the MPLAB PICkit Basic).
5	1	TCK	Test Clock (clock signal from the MPLAB PICkit Basic into the target device).
6	6	$\overline{\text{RESET}}$/TAUX	Reset (optional). Used to reset the target device. Connecting this pin is recommended since it allows the MPLAB PICkit Basic to hold the target device in a reset state, which can be essential to debugging in certain scenarios.
7	9	TDI	Test Data In (data transmitted from the MPLAB PICkit Basic into the target device).
8	5	TMS	Test Mode Select (control signal from the MPLAB PICkit Basic into the target device).
* Remember to include a decoupling capacitor between this pin and GND. See AVR042: AVR Hardware Design Considerations.			

3.3.5.2 AVR SPI Interface

In-system programming uses the target AVR's internal SPI (Serial Peripheral Interface) to download code into the Flash and EEPROM memories. It is not a debugging interface.

3.3.5.2.1 Connecting to an AVR SPI Target

The MPLAB PICkit Basic provides a direct connection for new designs or a legacy 10-pin 50-mil JTAG connection using the adapter board.

Important:
The SPI interface is effectively disabled when the debugWIRE Enable (DWEN) fuse is programmed, even if the SPIEN fuse is also programmed. To re-enable the SPI interface, the 'disable debugWIRE' command must be issued while in a debugWIRE debugging session. Disabling debugWIRE in this manner requires that the SPIEN fuse is already programmed. If MPLAB X IDE fails to disable debugWIRE, it is probably because the SPIEN fuse is NOT programmed. If this is the case, it is necessary to use a High-Voltage Programming interface to program the SPIEN fuse.

Info:
The SPI interface is often referred to as "ISP" since it was the first in-system programming interface on Microchip AVR products. Other interfaces are now available for in-system programming.

Related Links

3.3.3.1. Adapter Board Pinout

3.3.5.2.2 AVR SPI Pinout

For a legacy application PCB which includes an AVR with the SPI interface, the pinout as shown in the figure below may have been used.

Figure 3-10. Legacy SPI Header Pinout

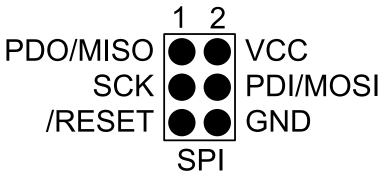


Table 3-6. SPI Pin Mapping

MPLAB PICkit Basic Pin	Adapter Board AVR JTAG Pin	Legacy SPI Pin	Name	Description
1	7 or 8 (NC)			
2	4 (VDD/VTG)	2	VCC/VTG	Target voltage (reference voltage)
3	2 or 10 (GND)	6	GND	Ground
4	3 (TDO)	1	MISO	Host In Client Out (Main In Secondary Out)
5	1 (TCK)	3	SCK	SPI clock
6	6 ($\overline{\text{RESET}}$/TAUX)	5	$\overline{\text{RESET}}$	
7	9 (TDI)	4	MOSI	Host Out Client In (Main Out Secondary In)
8	5 (TMS)			

3.3.5.3 AVR UPDI Interface

The Unified Program and Debug Interface (UPDI) is a Microchip proprietary interface for external programming and on-chip debugging of a device. It is a successor to the PDI two-wire physical interface, which is found on all AVR XMEGA devices. UPDI is a one-wire interface providing a bidirectional half-duplex asynchronous communication with the target device for purposes of programming and debugging.

Currently there are three configurations for UPDI:

1. The UPDI function is on a shared pin that can also be used for $\overline{\text{RESET}}$ or GPIO. If $\overline{\text{RESET}}$ or GPIO has been selected and UPDI is now desired, an HV pulse is required on that pin to reactivate the UPDI functionality.

   This configuration is found on older AVR devices (ATtiny) and requires an HV pulse of 12V.
2. The UPDI function is on a dedicated pin so UPDI is always available. $\overline{\text{RESET}}$ and GPIO share a pin.

   This configuration is found on newer AVR devices (ATmega0, AVR DA/DB etc.) and does not require an HV pulse.
3. The UPDI function is on a shared pin that can also be used for GPIO. $\overline{\text{RESET}}$ is on a dedicated pin. If GPIO has been selected and UPDI is now desired, an HV pulse is required on the $\overline{\text{RESET}}$ pin to reactivate the UPDI functionality.

   This configuration is found on newer AVR devices (AVR DD) and requires an HV pulse of approximately V_{DD} + 2 volts. See the device data sheet for the actual value range.

Consult your device data sheet for the configuration your device uses.

3.3.5.3.1 Connecting to an AVR UPDI Target

The MPLAB PICkit Basic provides a direct connection for new designs or a legacy 10-pin 50-mil JTAG connection for the 6-pin UPDI interface using the adapter board.

Related Links

3.3.3.1. Adapter Board Pinout

3.3.5.3.2 AVR UPDI Pinouts

For a legacy application PCB which includes an AVR with UPDI interfaces, the pinouts as shown in the figures below may have been used for all configurations.

Figure 3-11. Legacy UPDI Header Pinout

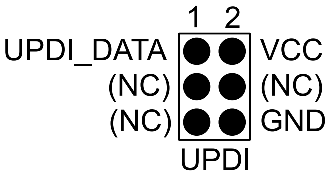


Table 3-7. Legacy UPDI Pin Mapping

MPLAB PICkit Basic Pin	Adapter Board AVR JTAG Pin	Legacy UPDI Pin	Name	Description
1	7 or 8 (NC)			
2	4 (VDD/VTG)	2	VCC/VTG	Target voltage (reference voltage).
3	2 or 10 (GND)	6	GND	Ground
4	3 (TDO)	1	UPDI_DATA*	UPDI data. Other pin functions are possible depending on the device.
5	1 (TCK)			
6	6 ($\overline{\text{RESET}}$/TAUX)			
7	9 (TDI)			
8	5 (TMS)			
* A High-Voltage (HV) pulse on this pin may be necessary to use the UPDI function. See your device data sheet for UPDI configuration.				

3.3.5.4 AVR PDI Interface

The Program and Debug Interface (PDI) is a Microchip proprietary interface for external programming and on-chip debugging of a device. PDI Physical is a 2-pin interface providing a bidirectional half-duplex synchronous communication with the target device.

3.3.5.4.1 Connecting to an AVR PDI Target

The MPLAB PICkit Basic provides a direct connection for new designs or a legacy 10-pin 50-mil JTAG connection for the 6-pin PDI interface using the adapter board.

Related Links

3.3.3.1. Adapter Board Pinout

3.3.5.4.2 AVR PDI Pinout

For a legacy application PCB which includes an AVR with the PDI interface, the pinout as shown in the figure below may have been used.

Figure 3-12. Legacy PDI Header Pinout

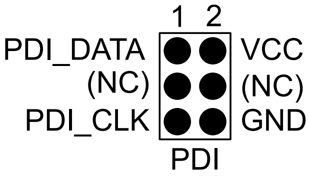


Table 3-8. PDI Pin Mapping

MPLAB PICkit Basic Pin	Adapter Board AVR JTAG Pin	Legacy PDI Pin	Name	Description
1	7 or 8 (NC)			
2	4 (VDD/VTG)	2	VCC/VTG	Target voltage (reference voltage).
3	2 or 10 (GND)	6	GND	Ground
4	3 (TDO)	1	PDI_DATA	PDI Data
5	1 (TCK)			
6	6 ($\overline{\text{RESET}}$/TAUX)	5	PDI_CLK	PDI Clock
7	9 (TDI)			
8	5 (TMS)			

3.3.5.5 AVR TPI Interface

TPI is a programming-only interface for some tinyAVR® devices. It is not a debugging interface and these devices do not have OCD capability.

3.3.5.5.1 Connecting to an AVR TPI Target

The MPLAB PICkit Basic provides a direct connection for new designs or a legacy 10-pin 50-mil JTAG connection using the adapter board.

Related Links

3.3.3.1. Adapter Board Pinout

3.3.5.5.2 AVR TPI Pinout

For a legacy application PCB which includes an AVR with the TPI interface, the pinout as shown in the figure below may have been used.

Figure 3-13. Legacy TPI Header Pinout

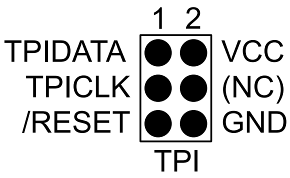


Table 3-9. TPI Pin Mapping

MPLAB PICkit Basic Pin	Adapter Board AVR JTAG Pin	Legacy TPI Pin	Name	Description
1	7 or 8 (NC)			
2	4 (VDD/VTG)	2	VCC/VTG	Target voltage (reference voltage).
3	2 or 10 (GND)	6	GND	Ground
4	3 (TDO)	1	DATA	TPI Data
5	1 (TCK)	3	CLOCK	TPI Clock
6	6 ($\overline{\text{RESET}}$/TAUX)	5	$\overline{\text{RESET}}$	Reset the device.
7	9 (TDI)			
8	5 (TMS)			

3.3.5.6 AVR debugWIRE Interface

The debugWIRE interface is for use on low pin-count devices. Unlike the JTAG interface which uses four pins, debugWIRE makes use of just a single pin ($\overline{\text{RESET}}$) for bidirectional half-duplex asynchronous communication with the debugger tool.

Note:
The debugWIRE interface can not be used as a programming interface. This means that the SPI interface must also be available (as shown in 3.3.5.2. AVR SPI Interface) in order to program the target.

When launching a debug session using debugWIRE, flash will be programmed using the debugWIRE interface. This is not an option which can be considered for factory programming.

When the debugWIRE enable (DWEN) fuse is programmed and lock-bits are un-programmed, the debugWIRE system within the target device is activated. The $\overline{\text{RESET}}$ pin is configured as a wire-AND (open-drain) bidirectional I/O pin with pull-up enabled and becomes the communication gateway between target and debugger.

3.3.5.6.1 Connecting to an AVR debugWIRE Target

The MPLAB PICkit Basic provides a direct connection for new designs or a legacy 10-pin 50-mil JTAG connection for the 6-pin debugWIRE/SPI interface using the adapter board.

Although the debugWIRE interface only requires one signal line ($\overline{\text{RESET}}$), VCC, and GND to operate correctly, it is advised to have access to the full SPI connector so that the debugWIRE interface can be enabled and disabled using SPI programming.

When the DWEN fuse is enabled, the SPI interface is overridden internally for the OCD module to have control of the $\overline{\text{RESET}}$ pin. The debugWIRE OCD is capable of disabling itself temporarily, thus releasing control of the $\overline{\text{RESET}}$ line. The SPI interface is then available again (only if the SPIEN fuse is programmed), allowing the DWEN fuse to be un-programmed using the SPI interface. If power is toggled before the DWEN fuse is un-programmed, the debugWIRE module will again take control of the $\overline{\text{RESET}}$ pin. Normally MPLAB X IDE or Microchip Studio will automatically handle the interface switching, but it can also be done manually using the button on the debugging tab in the properties dialog in Microchip Studio.

Note: It is highly recommended to let MPLAB X IDE or Microchip Studio handle the setting and clearing of the DWEN fuse.

It is not possible to use the debugWIRE interface if the lockbits on the target AVR device are programmed. Always be sure that the lockbits are cleared before programming the DWEN fuse and never set the lockbits while the DWEN fuse is programmed. If both the debugWIRE Enable (DWEN) fuse and lockbits are set, one can use High Voltage Programming to do a chip erase, and thus clear the lockbits. When the lockbits are cleared, the debugWIRE interface will be re-enabled. The SPI Interface is only capable of reading fuses, reading signature, and performing a chip erase when the DWEN fuse is un-programmed.

Related Links

3.3.3.1. Adapter Board Pinout

3.3.5.6.2 AVR debugWIRE Pinout

For a legacy application PCB which includes an AVR with the debugWire interface, the pinout as shown in the figure below may have been used.

Figure 3-14. Legacy debugWIRE (SPI) Header Pinout

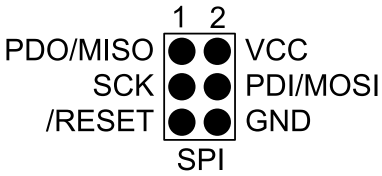


Table 3-10. debugWIRE Pin Mapping

MPLAB PICkit Basic Pin	Adapter Board AVR JTAG Pin	Legacy debugWIRE Pin	Name	Description
1	7 or 8 (NC)			
2	4 (VDD/VTG)	2	VCC/TTG	Target voltage (reference voltage).
3	2 or 10 (GND)	6	GND	Ground
4	3 (TDO)			
5	1 (TCK)			
6	6 ($\overline{\text{RESET}}$/TAUX)	5	$\overline{\text{RESET}}$ or debugWIRE	Use pin to Reset the device or transmit debugWIRE data.
7	9 (TDI)			
8	5 (TMS)			

3.3.6 PIC32M Connections

PIC32M MIPS-based devices use EJTAG for debug and programming.

3.3.6.1 Connecting to a PIC32M EJTAG Target

The MPLAB PICkit Basic provides a direct connection for new designs or a legacy 14-pin 10-mil JTAG/ EJTAG connection using the adapter board.

3.3.6.2 PIC32M EJTAG Pinout - 4-Wire JTAG

PIC32M EJTAG pin names and descriptions are shown in the table below. Pin numbers are shown for MPLAB PICkit Basic direct connection and Debugger Adapter Board 14-pin connection.

Table 3-11. PIC32M JTAG Connector 14-Pin Description

MPLAB PICkit Basic Pin	Adapter Board (14-Pin) Pin	Name	Description
1	11	$\overline{\text{MCLR}}$	Reset (optional). Used to reset the target device. Connecting this pin is recommended since it allows the MPLAB PICkit Basic to hold the target device in a reset state, which can be essential to debugging in certain scenarios.
2	14	VDD	MPLAB PICkit Basic providing power to target (optional) or target providing power to MPLAB PICkit Basic (PTG).
3	2, 4, 6, 8, 10	GND	Ground. All must be connected to ensure that the MPLAB PICkit Basic and the target device share the same ground reference.
4	3	TDO	Test Data Out (data transmitted from the target device into the MPLAB PICkit Basic).
5	9	TCK	Test Clock (clock signal from the MPLAB PICkit Basic into the target device).

Table 3-11. PIC32M JTAG Connector 14-Pin Description (continued)

MPLAB PICkit Basic Pin	Adapter Board (14-Pin) Pin	Name	Description
6	1	NC	Not connected.
7	5	TDI	Test Data In (data transmitted from the MPLAB PICkit Basic into the target device).
8	7	TMS	Test Mode Select (control signal from the MPLAB PICkit Basic into the target device).

3.3.7 PIC MCUs - ICSP Connection

The MPLAB® PICkit™ Basic In-Circuit Debugger supports debugging and programming of PIC microcontrollers (MCUs) and dsPIC digital signal controllers (DSCs) through ICSP™ (In-Circuit Serial Programming™) connections. The PICkit Basic SIL connector uses two device I/O pins and the reset line to implement in-circuit debugging and ICSP.

3.3.7.1 ICSP Target Connection

Connect a debugger directly to a PIC® MCU target using the ICSP® modular connector or inline connector on most MPLAB® debug tools. The connections to the debugger adapter board are the same as connections to target boards.

If the debugger and target have different connections (modular-to-inline or inline-to-modular respectively) a small adapter can be purchased to enable proper connections: "RJ11 to ICSP Adapter" (AC164110).

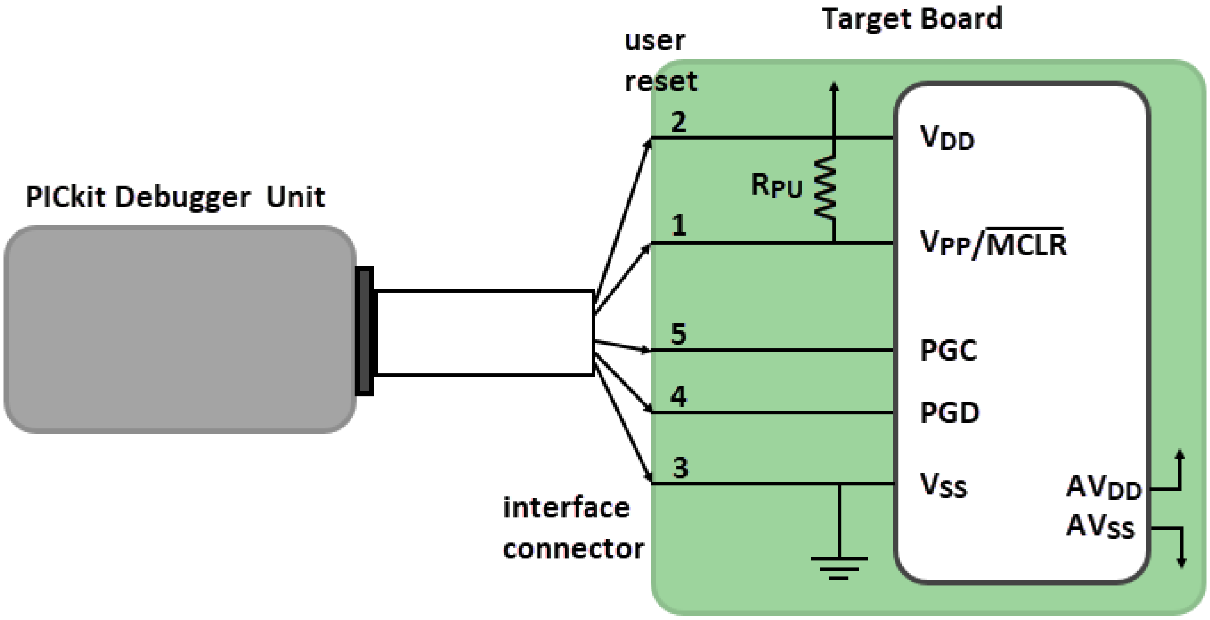
Figure 3-15. 6-Pin RJ11 to ICSP Adapter

Alternatively, the MPLAB PICkit Basic using the adapter board provides an 8-pin 50-mil Microchip Universal connection for the 6-pin and 8-pin ICSP interfaces.

3.3.7.2 ICSP™ Target Connection Circuitry

The figure below shows the interconnections of the MPLAB PICkit Basic In-Circuit Debugger to the ICSP connector on the target board. The diagram also shows the wiring from the connector to a device on the target PCB. A pull-up resistor R_{PU} is recommended to be connected from the V_{PP}/$\overline{MCLR}$ line to V_{DD} so that the line may be strobed low to reset the device.

Figure 3-16. ICSP Target Connection



where Rpu=10-50 kΩ

4. Operation

A simplified theory of operation of the MPLAB® PICkit™ Basic In-Circuit Debugger system is provided here. It is intended to provide enough information so that a target board can be designed that is compatible with the debugger for both debugging and programming operations. The basic theory of in-circuit debugging and programming is discussed so that problems, if encountered, are quickly resolved.

4.1 Quick Debug/Program Reference

The following table is a quick reference for using the MPLAB PICkit Basic In-Circuit Debugger as either a debugging or programming tool.

Attention: MPLAB PICkit Basic does not support debug headers.

Table 4-1. Debug vs. Program Operation

Item	Debug	Program
Needed Hardware	A computer, a target board (Microchip demo board or your own design), cable(s) and an optional adapter board to connect the computer to the target.	
	Debug Tool, USB cable, target power supply.	
	Device with on-board debug circuitry.	Device (with or without on-board debug circuitry).
Needed Software	MPLAB X IDE	MPLAB X IDE or MPLAB IPE.
	Application code (example code, i.e., from MPLAB Discover, or your own code).	Prebuilt code - Hex file.
MPLAB X IDE selection	Project Properties, PICkit Basic as Hardware Tool.	
	Debug Main Project icon .	Make and Program Device icon .
Program Operation	Programs application code into the device. Depending on the selections on the **Project Properties** dialog, this can be any range of program memory. In addition, a small debug executive may be placed in program memory and other debug resources may be reserved.	Programs application code into the device. Depending on the selections on the **Project Properties** dialog, this can be any range of program memory.
Debug Features Available	All for device – breakpoints, etc.	N/A
Serial Quick-Time Programming (SQTP)	N/A	Use the MPLAB IPE to generate the SQTP file.
Command-line Operation	Use MDB command line utility, found by default in `C:\Program Files\Microchip\MPLABX\`*`vx.xx`*`\mplab_platform\bin\mdb.bat`.	Use IPECMD, found by default in `C:\Program Files\Microchip\MPLABX\`*`vx.xx`*`\mplab_platform\mplab_ipe\ipecmd.exe`.

4.2 Operational Overview

Note: Ensure you read 3.2. PC Connections and 3.3. Target Connections before completing Step 5.

To debug and program code using the MPLAB PICkit Basic In-Circuit Debugger:

1. Download and install the latest MPLAB X IDE from the MPLAB X IDE webpage. MPLAB IPE is included in the MPLAB X IDE installer. See MPLAB X IDE documentation for how to create a project for developing application code and how to debug code.
2. Find examples of code in MPLAB Discover or search for Microchip content on GitHub, such as Microchip PIC & AVR Examples.

3. Find a compiler for on your application device on the MPLAB XC Compilers webpage.
4. Purchase MPLAB PICkit Basic and optionally the Debugger Adapter Board.
5. Launch MPLAB X IDE. Plug in MPLAB PICkit Basic to the computer using the USB cable. Ensure you target is correctly connected.
6. Open your project or an example project in MPLAB X IDE. Right click on the project name in the **Projects** tab and select **Properties**. In the **Project Properties** dialog, ensure that **PICkit Basic** is selected under **Connected Hardware Tool**. Then select the **PICkit Basic Category** under a Conf(iguration) and setup options from the **Option Categories** list.
Note: This is where you select power to target if desired.
7. Read the following sections for details on operation for your application device.
8. For a complete list of debugger limitations for your device, see the online Help file in MPLAB X IDE (**Help** > **Help Contents** > **Hardware Tool Reference Help** > **Limitations - Emulators and Debuggers**).

4.3 About High Voltage

The term “High Voltage” has been used to mean different things. Explanations of past and current definitions of this term are discussed below.

AVR® Devices

High-Voltage Programming - HVSP and HVPP

Older AVR devices have a programming interface known as High-Voltage Programming in both serial (HVSP) and parallel (HVPP) variants. In general this interface requires 12V to be applied to the $\overline{\text{RESET}}$ pin for the duration of the programming session.

High-Voltage Programming was sometimes necessary to recover when configuration bits (fuses) were incorrectly set or cleared. Some examples are:

- DWEN and lockbits set: debugWIRE not usable.
- DWEN set, SPIEN cleared: stuck in debugWIRE mode, cannot use SPI.
- JTAGEN cleared: Cannot use JTAG.

Tool Support: No current MPLAB® hardware tool supports this method of programming. Therefore it is important that all warnings about the above issues be heeded and instructions followed.

High-Voltage Pulse - UPDI Pin

Depending on your device, there are currently two configurations that will require a High-Voltage (HV) Pulse to enable the UPDI function on a pin:

1. The UPDI function is on a shared pin that can also be used for $\overline{\text{RESET}}$ or GPIO. If $\overline{\text{RESET}}$ or GPIO has been selected and UPDI is now desired, an HV pulse is required on that pin to reactivate the UPDI functionality.
This configuration is found on older AVR devices and requires an HV pulse of 12V.
2. The UPDI function is on a shared pin that can also be used for GPIO. $\overline{\text{RESET}}$ is on a dedicated pin. If GPIO has been selected and UPDI is now desired, an HV pulse is required on the $\overline{\text{RESET}}$ pin to reactivate the UPDI functionality.
This configuration is found on newer AVR devices and requires an HV pulse of approximately V_{DD} + 2 volts. See the device data sheet for the actual value range.

Tool Support: All current MPLAB hardware tools support either HV pulse except for MPLAB PICkit Basic and MPLAB Snap (do not support high voltage).

PIC® Devices

High-Voltage vs Low-Voltage Programming

For High-Voltage Programming, older PIC devices need to be programmed at high voltage (12V) but newer devices can use lower voltages (a voltage in excess of 9 volts placed on the V_{PP} pin).

For Low-Voltage Programming, the programming voltage V_{PP} will not exceed V_{DD}.

Tool Support: All current MPLAB hardware tools support high and low voltage programming except for:

- MPLAB PICkit Basic and MPLAB Snap (do not support high voltage).
- MPLAB PICkit™ 4 (high voltage programming only when device V_{DD} voltage is at or above 2.8V. Issue fixed on 10-10094-R6; see the label on the back of the unit).

4.4 SAM and PIC32C Arm Devices - On-Chip Debugging

Both SAM and PIC32C microcontrollers are based on Arm® Cortex-M® core. Debug features available depend on the type of core (see table below). Debug connectors support SWD and JTAG.

For more information on which devices have which cores, see 32-bit PIC® and SAM Microcontrollers or your device data sheet. See also CoreSight documentation provided by Arm.

Table 4-2. Cortex-M Debug and Trace Support Summary

Cortex-M Types	Debug Support
Cortex-M0+	Debug Optional: Basic debug functionality includes processor halt, single-step, processor core register access, Reset and HardFault Vector Catch, unlimited software breakpoints, and full system memory access. Also 1/2/3/4 breakpoint, and 1/2 watchpoint functionality.
Cortex-M23	Debug Optional: Basic debug functionality includes processor halt, single-step, processor core register access, reset and HardFault Vector Catch, unlimited software breakpoints, and full system memory access. Also 1/2/3/4 breakpoint, and 1/2/3/4 watchpoint functionality.
Cortex-M4, M4F	Debug Optional: Basic debug functionality includes processor halt, single-step, processor core register access, Vector Catch, unlimited software breakpoints, and full system memory access. Also various breakpoint and 1/4 watchpoint functionality.
Cortex-M7	Cortex-M7 debug functionality includes processor halt, single-step, processor core register access, Vector Catch, unlimited software breakpoints, and full system memory access. The processor also includes support for 4/8 hardware breakpoints and 2/4 watchpoints configured during implementation.

4.5 AVR Devices - On-Chip Debugging (OCD)

An on-chip debug module is a system allowing a developer to monitor and control the execution on a device from an external development platform, usually through a device known as a *debugger* or *debug adapter*.

With an OCD system, the application can be executed whilst maintaining exact electrical and timing characteristics in the target system, and while being able to stop execution conditionally or manually and inspect program flow and memory.

Run Mode

When in Run mode, the execution of code is completely independent of the MPLAB PICkit Basic. The MPLAB PICkit Basic will continuously monitor the target device to see if a break condition has occurred. When this happens, the OCD system will interrogate the device through its debug interface, allowing the user to view the internal state of the device.

Stopped Mode

When a breakpoint is reached, the program execution is halted, but some I/O may continue to run as if no breakpoint had occurred. For example, assume that a USART transmit has just been initiated when a breakpoint is reached. In this case, the USART continues to run at full speed, completing the transmission, even though the core is in Stopped mode.

Hardware Breakpoints

The target OCD module contains several Program Counter comparators implemented in the hardware. When the Program Counter matches the value stored in one of the comparator registers, the OCD enters Stopped mode. Since hardware breakpoints require dedicated hardware on the OCD module, the number of breakpoints available depends upon the size of the OCD module implemented on the target. Usually, one such hardware comparator is 'reserved' by the debugger for internal use.

Software Breakpoints

A software breakpoint is a `BREAK` instruction placed in the program memory on the target device. When this instruction is loaded, program execution will break, and the OCD enters Stopped mode. To continue execution a "start" command has to be given from OCD. Not all Microchip devices have OCD modules supporting the `BREAK` instruction.

4.5.1 AVR Device Interfaces

Note: If you are having problems with programming and debugging with AVR microcontroller devices that use the UPDI/PDI/TPI interfaces, check Engineering Technical Notes (ETNs) for your tool.

The AVR devices feature various programming and debugging interfaces. Check the device data sheet for supported interfaces of that device.

- All AVR E/D devices and some newer tinyAVR and megaAVR devices have a UPDI interface, which is used for programming and debugging.
- Some tinyAVR® devices have a TPI interface. TPI can be used for programming the device only. These devices do not have on-chip debug capability at all.
- Some tinyAVR devices and some megaAVR devices have the debugWIRE interface, which connects to an on-chip debug system known as tinyOCD. All devices with debugWIRE also have the SPI interface for in-system programming.
- Some megaAVR devices have a JTAG interface for programming and debugging, with an on-chip debug system, also known as megaOCD. All devices with JTAG also feature the SPI interface as an alternative interface for in-system programming.
- All AVR XMEGA devices have the PDI interface for programming and debugging. Some AVR XMEGA devices also have a JTAG interface with identical functionality.

Table 4-3. Programming and Debugging Interfaces Summary

AVR Device Families	UPDI	TPI	SPI	debugWIRE	JTAG	PDI
AVR E/D	New devices					
tinyAVR	New devices	Some devices	Some devices	Some devices		
megaAVR	New devices		Some devices	Some devices	Some devices	
AVR XMEGA					Some devices	All devices

4.5.1.1 UPDI OCD Features

The UPDI OCD for newer tinyAVR, newer megaAVR, and AVR E/D devices is based on the UPDI physical interface, which is a single pin programming and debugging interface.

Other tinyAVR and megaAVR features include:

- Memory-mapped access to device address space (NVM, RAM, I/O)
- No limitation on the device clock frequency
- Unlimited number of user program breakpoints
- Two hardware breakpoints

- Support for advanced OCD features
- Non-intrusive run-time chip monitoring without accessing the system registers
- Interface for reading the result of the CRC check of the Flash on a locked device

Other AVR E/D features include:

- Two hardware breakpoints
- Change of flow, interrupt, and software breakpoints
- Run-time read-out of Stack Pointer (SP) register, Program Counter (PC), and Status Register (SREG)
- Register file read- and writable in Stopped mode

For devices where the UPDI pin is shared, it can be re-configured into a GPIO or /RESET pin. For details, see UPDI High-Voltage Activation Information.

For older devices the debugWIRE OCD is available. For more on OCD features, see debugWIRE OCD Features.

4.5.1.1.1 UPDI OCD Special Considerations

The UPDI data pin (UPDI_DATA) can be a dedicated pin or a shared pin, depending on the target AVR® device. A shared UPDI pin will require activation using a High-Voltage (HV) pulse on the UPDI or $\overline{\text{RESET}}$ pin depending on device. See your device data sheet for details.

On devices which include the CRCSCAN module (Cyclic Redundancy Check Memory Scan), this module should not be used in Continuous Background mode while debugging. The OCD module has limited hardware breakpoint comparator resources, so `BREAK` instructions may be inserted into Flash (software breakpoints) when more breakpoints are required, or even during source-level code stepping. The CRC module could incorrectly detect this breakpoint as a corruption of Flash memory contents.

The CRCSCAN module will appear configured to perform a CRC scan before boot. In the case of a CRC mismatch, the device will not boot and appears to be in a locked state. The only way to recover the device from this state is to perform a full chip erase and either program a valid Flash image or disable the pre-boot CRCSCAN (a simple chip erase will result in a blank Flash with invalid CRC and the part will thus still not boot). The software front-end will automatically disable the CRCSCAN fuses when chip erasing a device in this state.

When designing a target application PCB where the UPDI interface will be used, the following considerations must be made for correct operation:

- Pull-up resistors on the UPDI line must not be smaller than 10 kΩ. A pull-down resistor should not be used, or it should be removed when using UPDI. The UPDI physical is push-pull capable, so only a weak pull-up resistor is required to prevent false Start bit triggering when the line is idle.
- If the UPDI pin is to be used as a $\overline{\text{RESET}}$ pin, any stabilizing capacitor must be disconnected when using UPDI, since it will interfere with correct operation of the interface.
- If the UPDI pin is used as $\overline{\text{RESET}}$ or GPIO pin, all external drivers on the line must be disconnected during programming or debugging since they may interfere with the correct operation of the interface.

4.5.1.2 debugWIRE OCD Features

The debugWIRE OCD is a specialized OCD module with a limited feature set specially designed for AVR devices with low pin-count. It supports the following features:

- Complete program flow control
- Full access to all registers and memory areas

- Unlimited user program breakpoints (using `BREAK` instruction)
- Automatic baud rate configuration based on target clock

4.5.1.2.1 debugWIRE Special Considerations

The debugWIRE communication pin (dW) is physically located on the same pin as the external Reset (RESET). An external Reset source is, therefore, not supported when the debugWIRE interface is enabled.

The debugWIRE Enable (DWEN) fuse must be set on the target device for the debugWIRE interface to function. This fuse is by default unprogrammed when the Microchip AVR device is shipped from the factory. The debugWIRE interface itself cannot be used to set this fuse. To set the DWEN fuse, the SPI mode must be used. The software front-end handles this automatically provided that the necessary SPI pins are connected. It can also be set manually using SPI programming in the software front-end.

Either: Attempt to start a debug session on the debugWIRE part. If the debugWIRE interface is not enabled, the software front-end will offer to retry or attempt to enable debugWIRE using SPI programming. If you have the full SPI header connected, debugWIRE will be enabled and you will be asked to toggle power on the target. This is required for the fuse changes to be effective.

Or: Open the programming dialog in Microchip Studio in SPI mode and verify that the signature matches the correct device. Check the DWEN fuse to enable debugWIRE.

Important:
It is important to leave the SPIEN fuse programmed and the RSTDISBL fuse unprogrammed! Not doing this will render the device stuck in debugWIRE mode and High-Voltage programming will be required to revert the DWEN setting.

To disable the debugWIRE interface, use High-Voltage programming to unprogram the DWEN fuse. Alternately, use the debugWIRE interface itself to temporarily disable itself, which will allow SPI programming to take place, provided that the SPIEN fuse is set.

Important:
If the SPIEN fuse was NOT left programmed, the software front-end will not be able to complete this operation and High-Voltage programming must be used.

In MPLAB X IDE, if debugWIRE is enabled on the target device and an SPI programming session is attempted, the IDE will offer to disable debugWIRE first. In Microchip Studio, this must be done manually by, during a debug session, selecting the ‘Disable debugWIRE and Close’ menu option from the ‘Debug’ menu. DebugWIRE will be temporarily disabled, and the software front-end will use SPI programming to unprogram the DWEN fuse.

Having the DWEN fuse programmed enables some parts of the clock system to be running in all sleep modes. This will increase the power consumption of the AVR while in sleep modes. The DWEN Fuse should, therefore, always be disabled when debugWIRE is not used.

When designing a target application PCB where debugWIRE will be used, the following considerations must be made for correct operation:

- Pull-up resistors on the dW/(RESET) line must not be smaller than 10 kΩ. The pull-up resistor is not required for debugWIRE functionality since the debugger tool provides this.
- Any stabilizing capacitor connected to the RESET pin must be disconnected when using debugWIRE since they will interfere with correct operation of the interface.

- All external Reset sources or other active drivers on the RESET line must be disconnected, since they may interfere with the correct operation of the interface.

Never program the lock-bits on the target device. The debugWIRE interface requires that lock-bits are cleared to function correctly.

4.5.1.2.2 debugWIRE Software Breakpoints

The debugWIRE OCD is drastically downscaled when compared to the Microchip megaAVR (JTAG) OCD. This means that it does not have any Program Counter breakpoint comparators available to the user for debugging purposes. One such comparator does exist for purposes of run-to-cursor and single-stepping operations, but additional user breakpoints are not supported in hardware.

Instead, the debugger must make use of the `AVR BREAK` instruction. This instruction can be placed in FLASH, and when loaded for execution, it will cause the AVR CPU to enter Stopped mode. To support breakpoints during debugging, the debugger must insert a `BREAK` instruction into FLASH at the point at which the users request a breakpoint. The original instruction must be cached for later replacement. When single-stepping over a `BREAK` instruction, the debugger has to execute the original cached instruction to preserve program behavior. In extreme cases, the `BREAK` has to be removed from FLASH and replaced later. All these scenarios can cause apparent delays when single-stepping from breakpoints, which will be exacerbated when the target clock frequency is very low.

It is thus recommended to observe the following guidelines, where possible:

- Always run the target at as high a frequency as possible during debugging. The debugWIRE physical interface is clocked from the target clock.
- Try to minimize the number of breakpoint additions and removals, as each one requires a FLASH page to be replaced on the target.
- Try to add or remove a small number of breakpoints at a time, to minimize the number of FLASH page write operations.
- If possible, avoid placing breakpoints on double-word instructions.

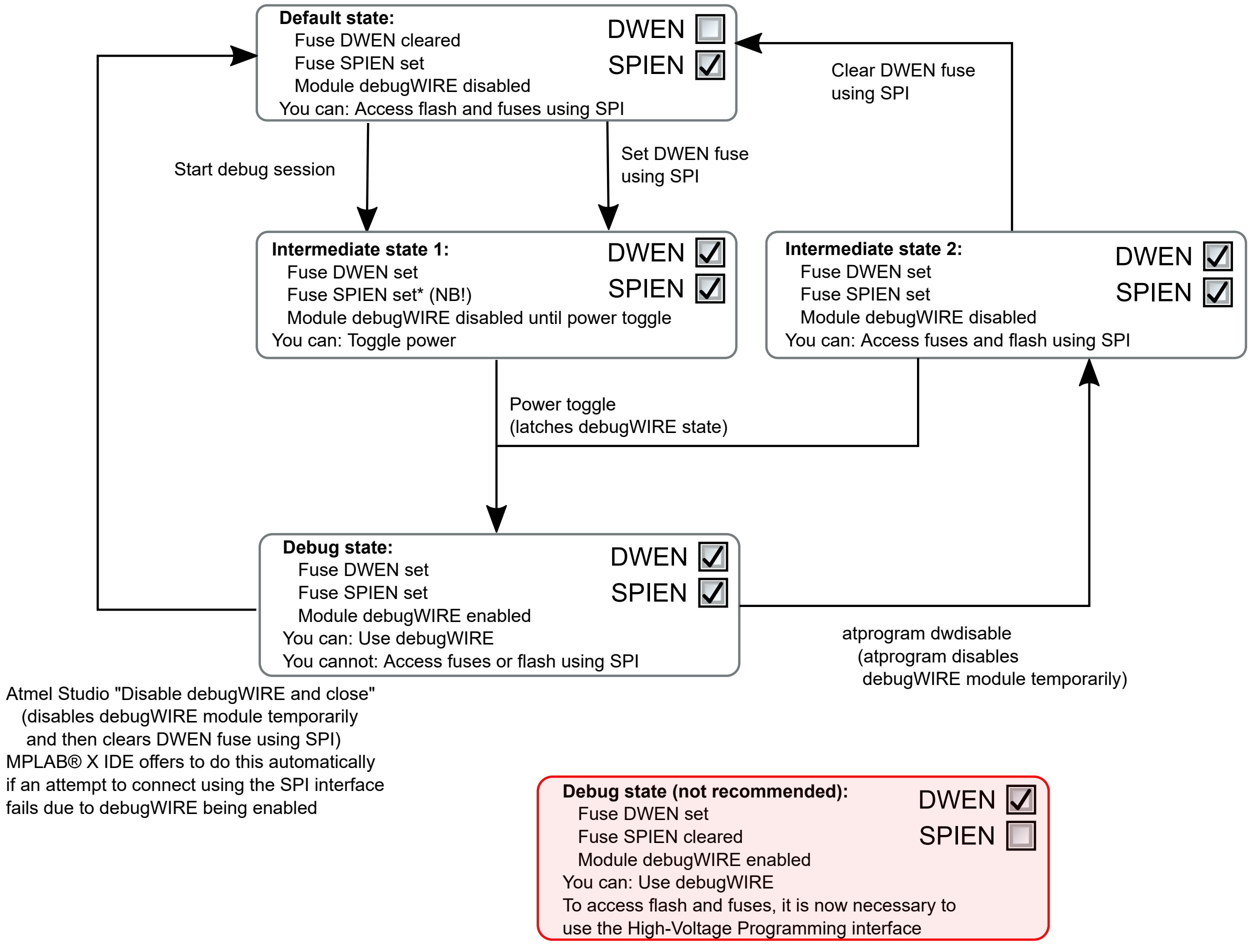
4.5.1.2.3 Understanding debugWIRE and the DWEN Fuse

When enabled, the debugWIRE interface takes control of the device's /RESET pin, which makes it mutually exclusive to the SPI interface, which also needs this pin. When enabling and disabling the debugWIRE module, follow one of these two approaches:

- Let the software front-end take care of things (recommended)
- Set and clear DWEN manually (exercise caution, advanced users only!)

Important: When manipulating DWEN manually, the SPIEN fuse must remain set to avoid having to use High-Voltage programming.

Figure 4-1. Understanding debugWIRE and the DWEN Fuse


Default state:
Fuse DWEN cleared
Fuse SPIEN set
Module debugWIRE disabled
You can: Access flash and fuses using SPI
DWEN □
SPIEN ☑
Clear DWEN fuse using SPI
Start debug session
Set DWEN fuse using SPI
Intermediate state 1:
Fuse DWEN set
Fuse SPIEN set* (NB!)
Module debugWIRE disabled until power toggle
You can: Toggle power
DWEN ☑
SPIEN ☑
Intermediate state 2:
Fuse DWEN set
Fuse SPIEN set
Module debugWIRE disabled
You can: Access fuses and flash using SPI
DWEN ☑
SPIEN ☑
Power toggle
(latches debugWIRE state)
Debug state:
Fuse DWEN set
Fuse SPIEN set
Module debugWIRE enabled
You can: Use debugWIRE
You cannot: Access fuses or flash using SPI
DWEN ☑
SPIEN ☑
atprogram dwdisable
(atprogram disables
debugWIRE module temporarily)
Atmel Studio "Disable debugWIRE and close"
(disables debugWIRE module temporarily
and then clears DWEN fuse using SPI)
MPLAB® X IDE offers to do this automatically
if an attempt to connect using the SPI interface
fails due to debugWIRE being enabled
Debug state (not recommended):
Fuse DWEN set
Fuse SPIEN cleared
Module debugWIRE enabled
You can: Use debugWIRE
To access flash and fuses, it is now necessary to
use the High-Voltage Programming interface
DWEN ☑
SPIEN □

4.5.1.3 megaAVR OCD Features

The megaAVR OCD is based on the JTAG physical interface. It supports the following features:

- Complete program flow control.
- Full access to all registers and memory areas.
- Four program memory (hardware) breakpoints (one is reserved).
- Hardware breakpoints can be combined to form data breakpoints.
- Unlimited number of program breakpoints (using `BREAK`) (except ATmega128[A]).

4.5.1.4 AVR XMEGA OCD Features

The AVR XMEGA OCD is otherwise known as PDI (Program and Debug Interface). Two physical interfaces (JTAG and PDI physical) provide access to the same OCD implementation within the device. It supports the following features:

- Complete program flow control.
- Full access to all registers and memory areas.
- One dedicated program address comparator or symbolic breakpoint (reserved).
- Four hardware comparators.
- Unlimited number of user program breakpoints (using `BREAK` instruction).
- No limitation on system clock frequency.

4.5.1.4.1 AVR® XMEGA® Special Considerations

OCD and Clocking

When the MCU enters Stopped mode, the OCD clock is used as MCU clock. The OCD clock is either the JTAG TCK if the JTAG interface is being used, or the PDI_CLK if the PDI interface is being used.

I/O Modules in Stopped Mode

In contrast to earlier Microchip megaAVR devices, in XMEGA, the I/O modules are stopped in Stop mode. This means that USART transmissions will be interrupted and timers (and PWM) will be stopped.

Hardware Breakpoints

There are four hardware breakpoint comparators - two address comparators and two value comparators. They have certain restrictions:

- All breakpoints must be of the same type (program or data).
- All data breakpoints must be in the same memory area (I/O, SRAM, or XRAM).
- There can only be one breakpoint if the address range is used.

Here are the different combinations that can be set:

- Two single data or program address breakpoints.
- One data or program address range breakpoint.
- Two single data address breakpoints with single value compare.
- One data breakpoint with address range, value range, or both.

MPLAB X IDE and Microchip Studio will tell you if the breakpoint cannot be set, and why. Data breakpoints have priority over program breakpoints if software breakpoints are available.

External Reset and PDI Physical

The PDI physical interface uses the Reset line as the clock. While debugging, the Reset pull-up should be 10k or more or be removed. Any Reset capacitors should be removed. Other external Reset sources should be disconnected.

JTAGEN Fuse

The JTAG interface is enabled using the JTAGEN fuse, which is programmed by default. This allows access to the JTAG programming interface.

Important: If the JTAGEN fuse is unintentionally disabled, it can only be re-enabled using the PDI physical interface.

If the JTAGEN fuse is programmed, the JTAG interface can still be disabled in firmware by setting the JTAG disable bit in the MCU Control Register. This will render code un-debuggable, and should not be done when attempting a debug session. If such code is already executing on the Microchip AVR device when starting a debug session, the MPLAB PICkit Basic will assert the RESET line while connecting. If this line is wired correctly, it will force the target AVR device into Reset, thereby allowing a JTAG connection.

If the JTAG interface is enabled, the JTAG pins cannot be used for alternative pin functions. They will remain dedicated JTAG pins until either the JTAG interface is disabled by setting the JTAG disable bit from the program code, or by clearing the JTAGEN fuse through a programming interface.

Tip:
Be sure to check the “use external reset” checkbox in both the programming dialog and debug options dialog in Microchip Studio to allow the MPLAB PICkit Basic to assert the RESET line and re-enable the JTAG interface on devices which are running code which disables the JTAG interface by setting the JTAG disable bit.

Debugging with Sleep for ATxmegaA1 rev H and Earlier

A bug existed on early versions of ATxmegaA1 devices that prevented the OCD from being enabled while the device was in certain sleep modes. There are two work-arounds to re-enable OCD:

- Go into the MPLAB PICkit Basic. Options in the Tools menu and enable “Always activate external Reset when reprogramming device.”
- Perform a chip erase.

The sleep modes that trigger this bug are:

- Power-Down
- Power-Save
- Standby
- Extended Standby

4.5.1.5 Advanced Debugging (AVR® JTAG/debugWIRE devices)

I/O Peripherals

Most I/O peripherals will continue to run even though the program execution is stopped by a breakpoint. Example: If a breakpoint is reached during a UART transmission, the transmission will be completed and corresponding bits set. The TXC (transmit complete) flag will be set and be available on the next single step of the code even though it normally would happen later in an actual device.

All I/O modules will continue to run in Stopped mode with the following two exceptions:

- Timer/Counters (configurable using the software front-end)
- Watchdog Timer (always stopped to prevent Resets during debugging)

Single Stepping I/O Access

Since the I/O continues to run in Stopped mode, care should be taken to avoid certain timing issues. For example, the code:

```
OUT PORTB, 0xAA
IN TEMP, PINB
```

When running this code normally, the TEMP register would not read back `0xAA` because the data would not yet have been latched physically to the pin by the time it is sampled by the IN operation. A `NOP` instruction must be placed between the `OUT` and the `IN` instruction to ensure that the correct value is present in the PIN register.

However, when single-stepping this function through the OCD, this code will always give `0xAA` in the PIN register since the I/O is running at full speed even when the core is stopped during the single-stepping.

Single Stepping and Timing

Certain registers need to be read or written within a given number of cycles after enabling a control signal. Since the I/O clock and peripherals continue to run at full speed in Stopped mode, single-stepping through such code will not meet the timing requirements. Between two single steps, the I/O clock may have run millions of cycles. To successfully read or write registers with such timing requirements, the whole read or write sequence should be performed as an atomic operation running the device at full speed. This can be done by using a macro or a function call to execute the code or use the run-to-cursor function in the debugging environment.

Accessing 16-Bit Registers

The Microchip AVR peripherals typically contain several 16-bit registers that can be accessed via the 8-bit data bus (e.g., TCNTn of a 16-bit timer). The 16-bit register must be byte accessed using two read or write operations. Breaking in the middle of 16-bit access or single-stepping through this situation may result in erroneous values.

Restricted I/O Register Access

Certain registers cannot be read without affecting their content. Such registers include those which contain flags which are cleared by reading, or buffered data registers (e.g., UDR). The software front-end will prevent reading these registers when in Stopped mode to preserve the intended non-intrusive nature of OCD debugging. Also, some registers cannot safely be written without side-effects occurring. These registers are read-only. For example:

- Flag registers, where a flag is cleared by writing `1` to any bit. These registers are read-only.
- UDR and SPDR registers cannot be read without affecting the state of the module. These registers are not accessible.

4.6 PIC32M MCU - On-Chip Debugging

PIC32M MCU devices support two types of debugging: (1) In-Circuit Serial Programming™ (ICSP™) and debugging using the PGECx and PGEDx pins or (2) 4-wire MIPS® Enhanced JTAG.

The MIPS32 M4K Processor core provides for an Enhanced JTAG (EJTAG) interface for use in the software debug of application and kernel code. In addition to the standard JTAG instructions, special instructions defined in the EJTAG specification define which registers are selected and how they are used. For details on this interface, see your device data sheet.

In addition, there are program and complex data breakpoints. See your device data sheet for details on debug features for your specific PIC32M device.

4.7 PIC MCU/dsPIC DSC - On-Chip Debugging

An on-chip debug module is a system allowing a developer to monitor and control the execution on a device from an external development platform, usually through a device known as a *debugger* or *debug adapter*. With an OCD system, the application can be executed while exact electrical and timing characteristics in the target system (as opposed to a simulator). The system is able to stop execution conditionally or manually and inspect program flow and memory.

For PIC® microcontrollers (MCUs) or dsPIC® digital signal controllers (DSC), some device resources may need to be reserved for debug.

4.7.1 Emulator Basic Features

MPLAB PICkit Basic In-Circuit Debugger has the following basic debug features.

4.7.1.1 Start and Stop Emulation

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

▷	Debug or execute project code in debug mode.
⏸	Pause or halt code execution.

▶	Continue code execution after a pause or halt.
⇩	For paused/halted code, Step Into or execute one instruction. Be careful not to step into a Sleep instruction or you will have to perform a processor Reset to resume emulation.
↷	For paused/halted code, Step Over an instruction.
■	Finish the debug session, which ends code execution.
⟳	Perform a processor Reset. Additional Resets, such as POR/BOR, MCLR and System, may be available, depending on device.

4.7.1.2 View Processor Memory and Files

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

- **Window** > **Target Memory Views** – view the different types of device memory. Depending on the selected device, memory types include Program Memory, File Registers, Configuration Memory, etc.
- **Window** > **Debugging** – view debug information. Select from variables, watches, call stack, breakpoints, stopwatch, and trace.

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

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

4.7.1.3 Use Breakpoints

Use breakpoints to halt code execution at specified lines in your code.

4.7.1.3.1 Breakpoint Resources

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

For PIC32M 32-bit devices, breakpoints use different resources than data captures and runtime watches. So, the available number of breakpoints is independent of the available number of triggers.

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

See Debug Limitations - PIC® MCUs for limitations on breakpoint operation, including the general number of hardware breakpoints per device and hardware breakpoint skidding amounts.

4.7.1.3.2 Hardware or Software Breakpoint Selection

To select hardware or software breakpoints:

1. Select your project in the **Projects** window and then right click to select **Properties**.
2. In **Project Properties**, select **ICE4** > **Debug Options**.
3. Check **Use software breakpoints** to use software breakpoints. Uncheck to use hardware breakpoints.

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

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

Table 4-4. Hardware vs. Software Breakpoints

Feature	Hardware Breakpoints	Software Breakpoints
Number of breakpoints	Limited	Unlimited
Breakpoints written to*	Internal debug registers	Flash Program Memory
Breakpoints applied to**	Program Memory/Data Memory	Program Memory only
Time to set breakpoints	Minimal	Dependent on oscillator speed, time to program Flash Memory and page size.
Breakpoint skidding	Most devices. See the online help, Limitations section, for details.	No
* Where information about the breakpoint is written in the device.		
** What kind of device feature applies to the breakpoint. This is where the breakpoint is set.		

4.7.1.4 Use the Stopwatch

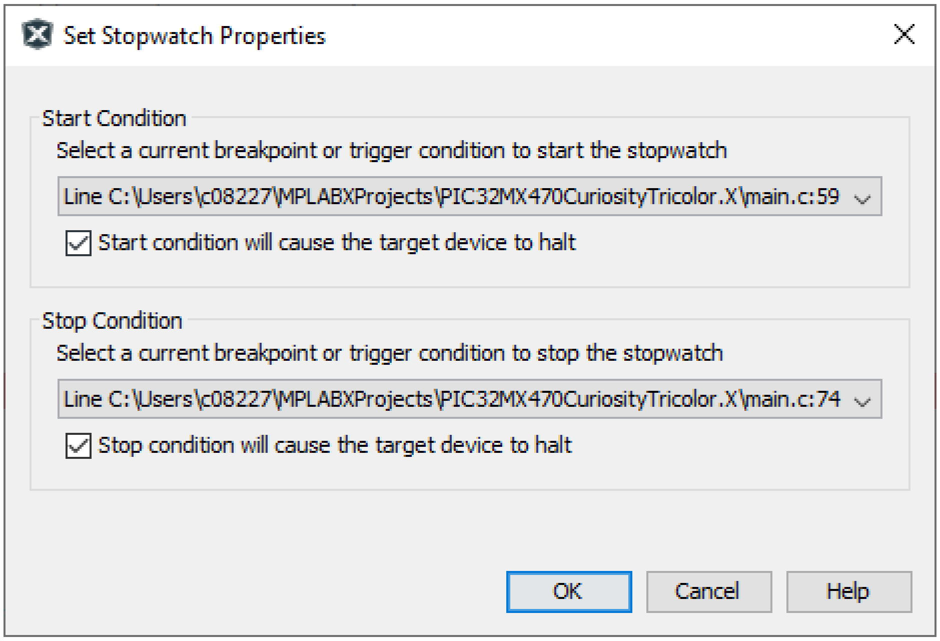
Use the stopwatch to determine the timing between two breakpoints.

Note: The stopwatch uses breakpoint resources.

To use the Stopwatch:

1. Add a breakpoint where you want to start the stopwatch.
2. Add another breakpoint where you want to stop the stopwatch.
3. Select **Window** > **Debugging** > **Stopwatch**. Click on the **Properties** icon on the left of the window and select the start and stop breakpoints.
4. Debug the program again to get the stopwatch timing result.

Figure 4-2. Stopwatch Setup



Set Stopwatch Properties

Start Condition

Select a current breakpoint or trigger condition to start the stopwatch

Line C:\Users\c08227\MPLABXProjects\PIC32MX470CuriosityTricolor.X\main.c:59

☑ Start condition will cause the target device to halt

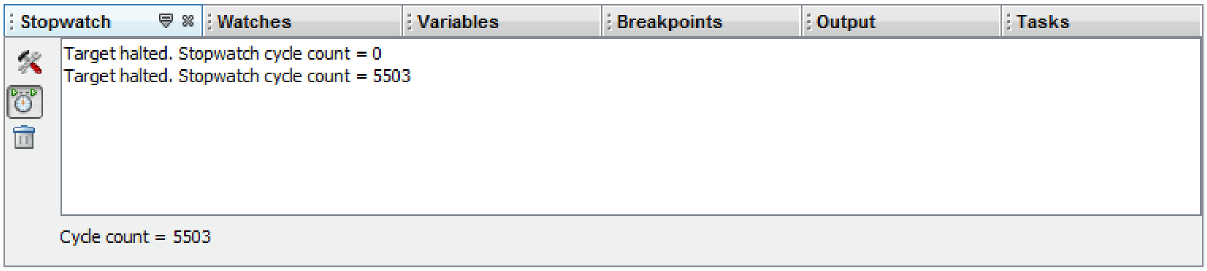
Stop Condition

Select a current breakpoint or trigger condition to stop the stopwatch

Line C:\Users\c08227\MPLABXProjects\PIC32MX470CuriosityTricolor.X\main.c:74

☑ Stop condition will cause the target device to halt

Figure 4-3. Stopwatch Window with Content



```
Target halted. Stopwatch cycle count = 0
Target halted. Stopwatch cycle count = 5503
```

Cycle count = 5503

The stopwatch has the following icons on the left side of the window:

Table 4-5. Stopwatch Icons

Icon	Icon Text	Description
🛠	Properties	Set stopwatch properties. Select one current breakpoint or trigger to start the stopwatch and one to stop the stopwatch.
⏱	Reset Stopwatch on Run	Reset the stopwatch time to zero at the start of a run.
🗑	Clear History	Clear the stopwatch window.
⏱	Clear Stopwatch	(Simulator Only) Reset the stopwatch after you reset the device.

4.7.1.5 Set Freeze Peripherals

You can select to **Freeze on Halt**, which allows you to freeze selected peripherals on a halt. For more on these functions, see Freeze Peripherals.

4.7.2 ICSP Debugging

There are two steps to using MPLAB® PICkit™ Basic In-Circuit Debugger as a debugger. The first requires that an application is programmed into the target device (MPLAB PICkit Basic can be used for this). 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 MPLAB X IDE operations:

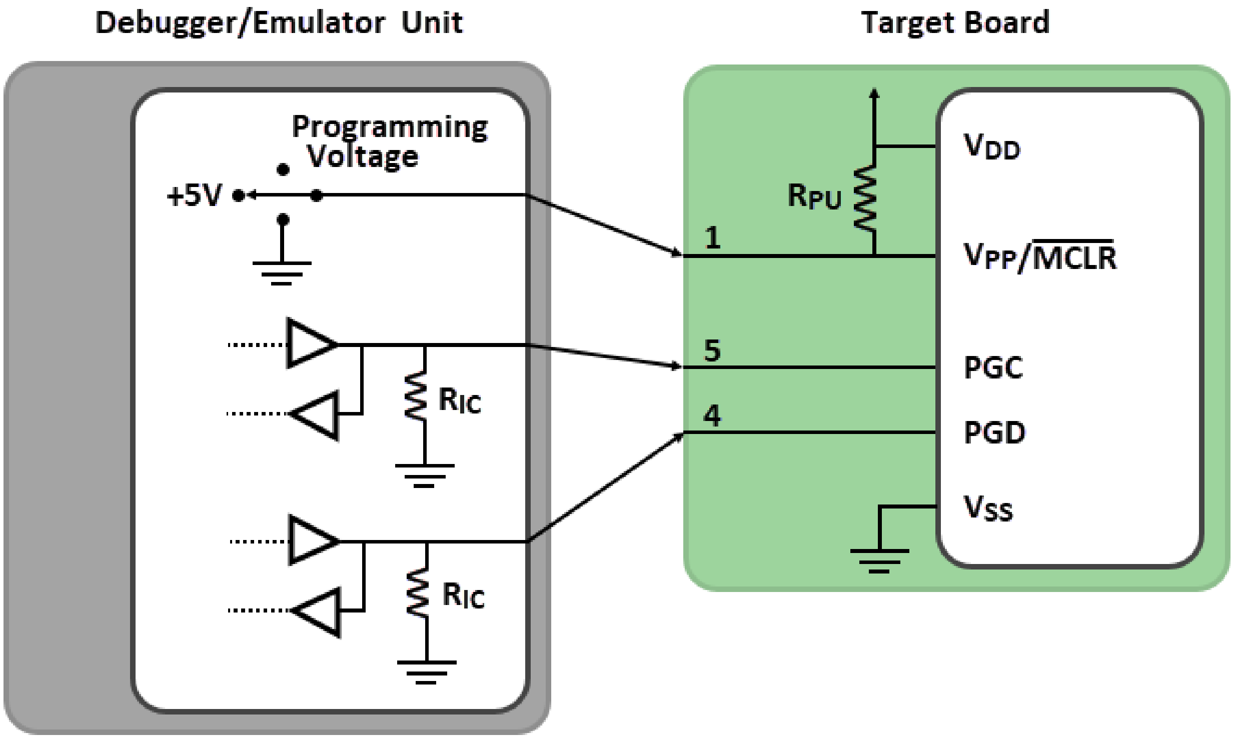
1. Programming the code into the target and activating special debug functions (see the next section for details).
2. Using the debugger to set breakpoints and run.

For more information, refer to the MPLAB X IDE WebHelp.

If the target device cannot be programmed correctly, the MPLAB PICkit Basic will not be able to debug it.

A simplified diagram of some of the internal interface circuitry of the MPLAB PICkit Basic is shown in the figure below.

Figure 4-4. Proper Connections for ICSP Programming



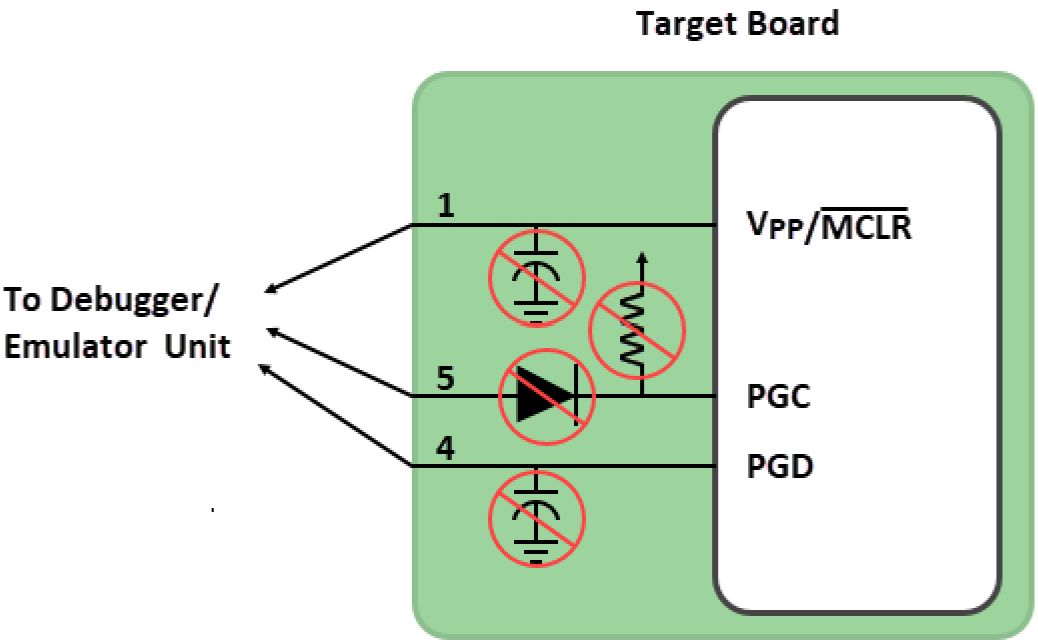
where and Ric=4.7 kΩ and Rpu=10 kΩ typical

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 sequence confirms the debugger and device are communicating correctly.

4.7.2.1 Circuits That Will Prevent the Emulator From Functioning

The figure below shows the active debugger lines with some components that will prevent the MPLAB PICkit Basic In-Circuit Debugger from functioning.

Figure 4-5. Improper Circuit Components



In particular, these guidelines must be followed:

- **Do not use pull-ups on PGC/PGD** – they could disrupt the voltage levels.
- **Do not use capacitors on PGC/PGD** – they will prevent fast transitions on data and clock lines during programming and debugging communications, and slow programming times.
- **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.

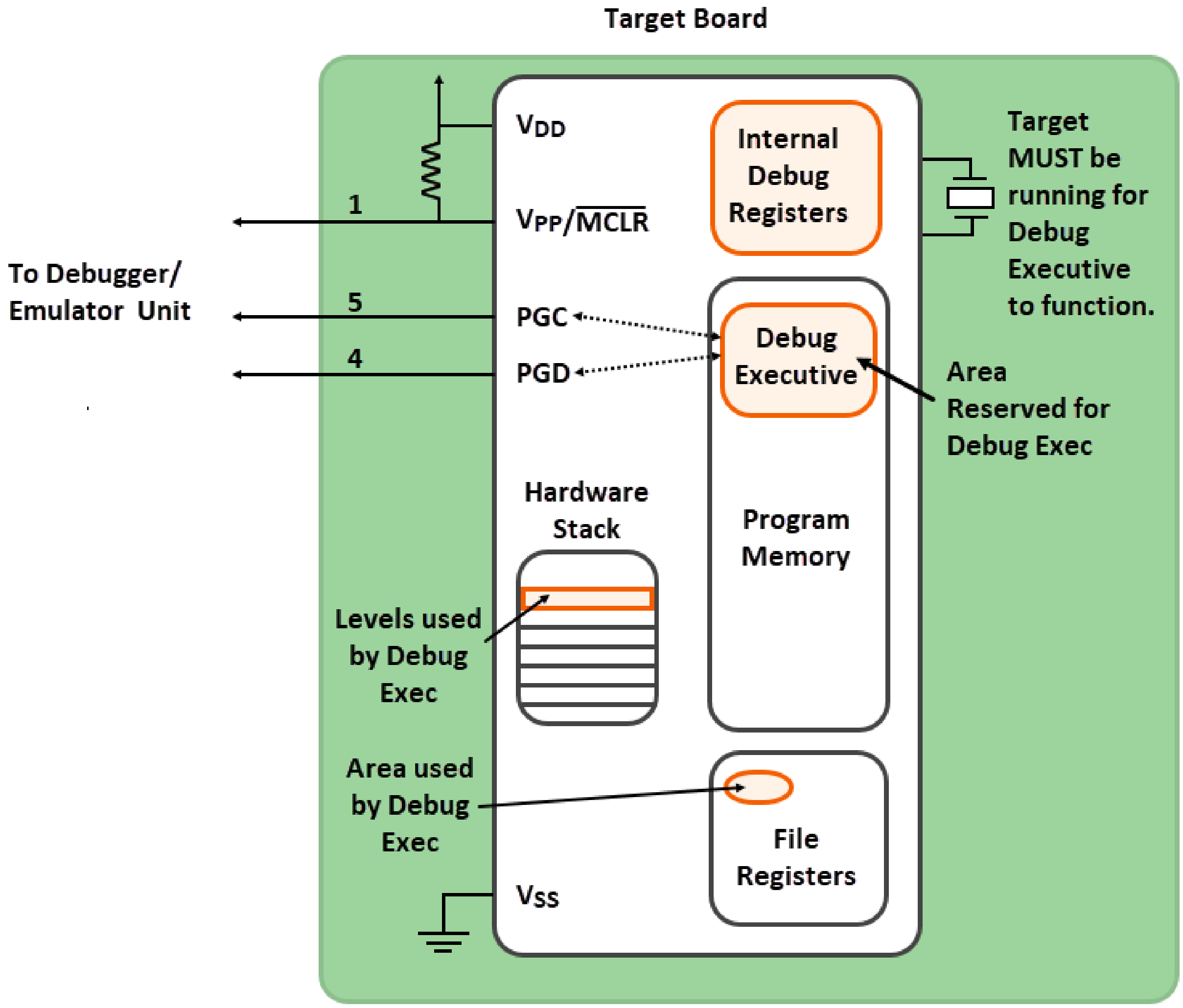
4.7.2.2 Sequence of Operations Leading to Debugging

Given that the 4.7.2.4. Requirements for Debugging are met, set the MPLAB PICkit Basic In-Circuit Debugger as the current tool in MPLAB X IDE. Right click on your project name and select **Properties** to open the **Project Properties** dialog, and then under **Connected Hardware Tool**, select a **MPLAB PICkit Basic**. If you have more than one tool connected, choose based on the serial number next to the tool name.

The following actions can now be performed:

- When **Debug** > **Debug Project** is selected, the application code is programmed into the device's memory via the ICSP protocol as described at the beginning of this section.
- A small "debug executive" program is loaded into the memory of the target device. Since some architectures require that the debug executive must reside in program memory, the application program must not use this reserved space. Some devices have special memory areas dedicated to the debug executive. Check your device data sheet for details.
- Special "in-circuit debug" registers in the target device are enabled by MPLAB X IDE. These allow the debug executive to be activated by the debugger. For more information on the device's reserved resources, see 4.7.2.5. Resources Used by the Debugger.
- The target device is run in Debug mode. To ensure the target will run, review 5.2. Top Reasons Why You Can't Debug, such as oscillator issues.

Figure 4-6. PIC® MCU Using Reserved Resources for Debug



4.7.2.3 Debugging Details

Once the proper connections between the MPLAB PICkit Basic and target have been made and incorrect circuits have been avoided, the debugger is ready to start debugging.

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

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

After the instruction at the breakpoint address is executed, the in-circuit debug mechanism of the target device “fires” and transfers the device’s program counter to the debug executive (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 X IDE. MPLAB X IDE then sends a series of queries to the debugger to get information about the target device, i.e., file register contents and the state of the CPU. These queries are performed by the debug executive.

The debug executive runs 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 (no oscillator, faulty power supply connection, shorts on the target board, etc.), then the debug executive cannot communicate to the MPLAB PICkit Basic, and MPLAB X IDE will issue an error message.

Another way to set a breakpoint is to select the Pause ⏸ button. 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 X IDE uses the debugger communications with the debug executive to interrogate the state of the target device.

4.7.2.4 Requirements for Debugging

To debug (set breakpoints, see registers, etc.) with the MPLAB PICkit Basic In-Circuit Debugger system, there are critical elements that must be working correctly:

- The debugger must be powered, must be connected to a computer, and must be communicating with the MPLAB X IDE software.
- The target device must have power and a functional, running oscillator. If for any reason the target device does not run, the MPLAB PICkit Basic In-Circuit Debugger will not be able to debug it.
- The target device must have its Configuration words programmed correctly. These may be set using code or the Configuration Bits window in MPLAB X IDE.
  - The oscillator Configuration bits should correspond to oscillator types available on the target.
  - For some devices, the Watchdog Timer is enabled by default and needs to be disabled.
  - The target device must not have any type of code protection enabled.
  - The target device must not have table read protection enabled.
- For some devices with more than one PGC/PGD pair, the correct pair needs to be selected in the device's configuration word settings. This only refers to debugging, since programming will work through any PGC/PGD pair.

4.7.2.5 Resources Used by the Debugger

For some devices, device resources must be used for debug. For a complete list of resources used by the debugger for your device, in MPLAB X IDE select **Help** > **Release Notes**. In addition to a section for "Release Notes/Readmes," there is a section for "Reserved Resources." Select either "Reserved Resources by Device Family and Tool" or "Reserved Resources by Device for All Tools."

4.7.2.6 Programming

As for debugging, set the MPLAB PICkit Basic In-Circuit Debugger as the current tool in MPLAB X IDE. Right click on your project name and select **Properties** to open the **Project Properties** dialog, and then under **Connected Hardware Tool**, select a **PICkit Basic**. If you have more than one tool connected, choose based on the serial number next to the tool name.

- Select the **Run Project** icon (▷). The application code is programmed into the device's memory via the ICSP protocol. 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.
- A small "program executive" program may be loaded into the high area of program memory for some target devices.

- Special “in-circuit debug” registers in the target device are disabled by MPLAB X IDE, along with all debug features. This means that a breakpoint cannot be set and register contents cannot be seen or altered.
- The target device is run in Release mode. As a programmer, the debugger can only toggle the MCLR line to Reset and start the target device.

5. Troubleshooting

If you are having problems with MPLAB® PICkit™ Basic In-Circuit Debugger operation, start here.

5.1 Some Questions to Answer First

1. **Which device are you working with?**
   Often an upgrade to a newer version of MPLAB X IDE or MPLAB IPE is required to support newer devices.
2. **Are you using a Microchip demo board or one of your own design? And, have you followed the guidelines for resistors/capacitors for communications connections?**
   See Development Tools Design Advisory.
3. **Have you powered the target?**
   The debugger cannot power the target. Use an external power supply to power the target board.
4. **Are you using a USB hub in your setup? Is it powered?**
   If you continue to have problems, try using the debugger without the hub (plugged directly into the computer).
5. **Are you using the USB cable shipped with the debugger?** Other USB cables may be of poor quality, too long or do not support USB Communication.

5.2 Top Reasons Why You Can't Debug

1. **Oscillator not working**. Check your Configuration bits setting for the oscillator. If you are using an external oscillator, try using an internal oscillator. If you are using an internal PLL, make sure your PLL settings are correct.
2. **No power to the target board**. Check the power cable connection.
3. **Incorrect V_{DD} voltage**. The V_{DD} voltage is outside the specifications for this device. See the device programming specification for details.
4. **Physical disconnect**. The debugger has become physically disconnected from the computer and/or the target board. Check the communication cables' connections.
5. **Communications lost**. Debugger to PC communication has somehow been interrupted. Reconnect to the debugger in MPLAB X IDE or MPLAB IPE.
6. **Device not seated**. The device is not properly seated on the target board. If the debugger is properly connected and the target board is powered, but the device is absent or not plugged in completely, you may receive the following message:
   ```
   Target Device ID (0x0) does not match expected Device ID (0x%x)
   ```
   , where `%x` is the expected device ID.
7. **Device is code-protected**. Check your Configuration bits settings for code protection.
8. **Application code corrupted**. The target application has become corrupted or contains errors. Try rebuilding and reprogramming the target application. Then initiate a Power-On-Reset of the target.
9. **Incorrect programming pins**. The PGC/PGD pin pairs are not correctly programmed in your Configuration bits (for devices with multiple PGC/PGD pin pairs).
10. **Additional setup required**. Other configuration settings are interfering with debugging. Any configuration setting that would prevent the target from executing code will also prevent the debugger from putting the code into Debug mode.
11. **Incorrect brown-out voltage**. Brown-out Detect voltage is greater than the operating voltage V_{DD}. This means the device is in Reset and cannot be debugged.
12. **Incorrect connections**. Review the guidelines in “Connections” section for the correct communication connections.

13. **Invalid request**. The debugger cannot always perform the action requested. For example, the debugger cannot set a breakpoint if the target application is currently running.

5.3 General

1. It is possible the error was a one-time event. Try the operation again.
2. There may be a problem programming in general. As a test, switch to Run mode using the ▷ icon and program the target with the simplest application possible (for example, 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 (for example, over current). Development environments are notoriously hostile to components. Consider trying another target board. Microchip Technology Inc. offers demonstration boards to support most of its microcontrollers. Consider using one of these applications, which are known to work, to verify correct MPLAB® PICkit™ Basic In-Circuit Debugger functionality.
4. Review debugger setup to ensure proper application setup. For more information, see the “Connections” and “Operation” sections.
5. Your program speed may be set too high for your circuit. In MPLAB X IDE, go to **File** > **Project Properties** and select the **PICkit Basic** category, *Program Options* option category. Next to *Program Speed* select a slower speed from the drop-down menu. The default is Normal.
6. There may be certain situations where the debugger is not operating properly and firmware may need to be downloaded or the debugger needs to be reprogrammed. See the following sections to determine additional actions.

5.4 How to Use the Hardware Tool Emergency Boot Firmware Recovery Utility

NOTICE **Only use this utility to restore hardware tool boot firmware to its factory state. Use only if your hardware tool no longer functions on any machine.**

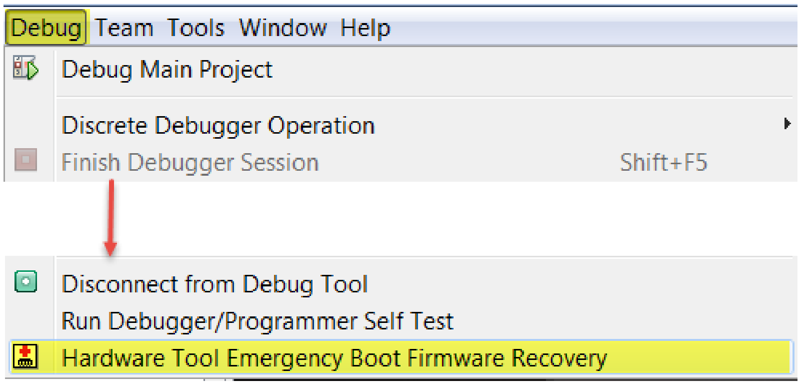
The debugger may need to be forced into recovery boot mode (reprogrammed) in rare situations; for example, if any of the following occurs when the debugger is connected to the computer:

- If the debugger has no LED lit.
- If the procedure described in the previous section was not successful.

YOU MUST HAVE MPLAB X IDE v6.25 OR GREATER TO USE THE EMERGENCY RECOVERY UTILITY FOR MPLAB PICkit Basic.

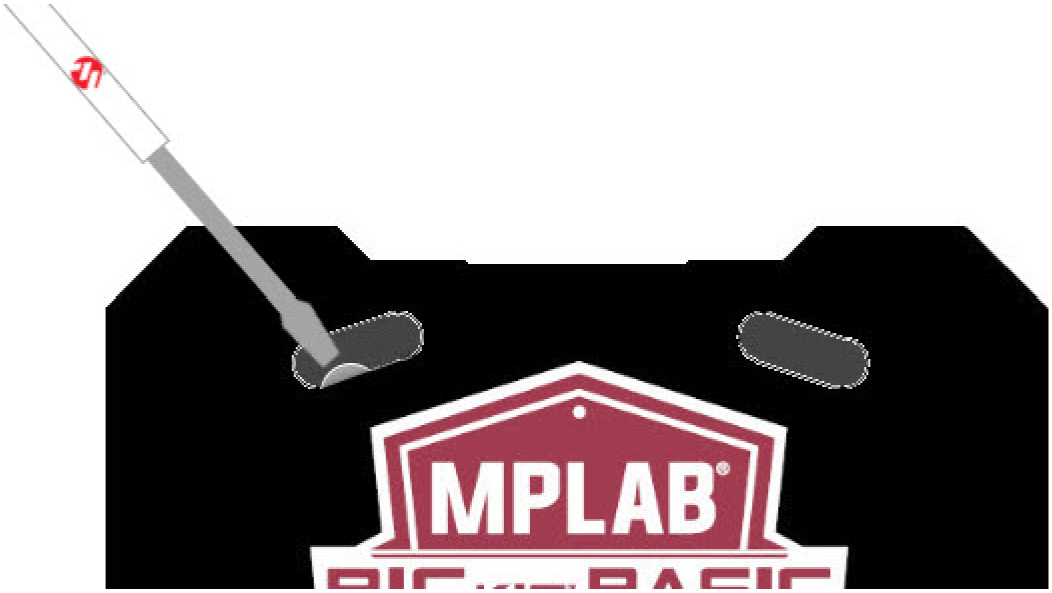
Carefully follow the instructions found in MPLAB X IDE under the main menu options **Debug** > **Hardware Tool Emergency Boot Firmware Recovery**.

Figure 5-1. Selecting Emergency Utility


Debug Team Tools Window Help
Debug Main Project
Discrete Debugger Operation
Finish Debugger Session Shift+F5
Disconnect from Debug Tool
Run Debugger/Programmer Self Test
Hardware Tool Emergency Boot Firmware Recovery

The figure below shows where the emergency recovery button is located on the MPLAB PICkit Basic In-Circuit Debugger. You can use a paperclip or small screwdriver (shown) to press the button.

Figure 5-2. Emergency Recovery Button



If the procedure was successful, the recovery wizard displays a success screen. The MPLAB PICkit Basic will now be operational and able to communicate with the MPLAB X IDE.

If the procedure failed, try it again. If it fails a second time, contact Microchip Support at support.microchip.com.

6. Frequently Asked Questions

Look here for answers to frequently asked questions about the MPLAB® PICkit™ Basic In-Circuit Debugger system.

6.1 How Does it Work?

What's in the silicon that allows it to communicate with the MPLAB PICkit Basic In-Circuit Debugger?

Modules in the silicon allow different types of communication interfaces. The MPLAB PICkit Basic In-Circuit Debugger can communicate with a device via the ICSP™ interface or other interfaces using an adapter board(s). If the communication module allows access to the device build-in debug circuitry, the debugger can access this to perform debug functions.

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

For PIC MCU devices, a debug executive is programmed into program or dedicated memory during a debug session. The debug executive doesn't run while in Run mode, so there is no throughput reduction when running your code; that is, the debugger doesn’t ‘steal’ any cycles from the target device.

Does the MPLAB PICkit Basic In-Circuit Debugger have complex breakpoints like other in-circuit emulators/debuggers?

No. But you can break based on a value in a data memory location or program address.

Does the MPLAB PICkit Basic 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 have to occur before it breaks. However, you can only do two sequences. You can also do the AND condition and do PASS counts.

Is the MPLAB PICkit Basic In-Circuit Debugger optoisolated or electrically isolated?

No. You cannot apply a floating or high voltage (120V) to the current system.

Will the MPLAB PICkit Basic In-Circuit Debugger slow down the running of the program?

No. The device will run at any device speed as specified in the data sheet.

Is it possible to debug a dsPIC DSC device running at any speed?

The MPLAB PICkit Basic In-Circuit Debugger is capable of debugging at any device speed as specified in the device’s data sheet.

6.2 What's Wrong?

Things to consider:

Performing a Verify fails after programming the device. Is this a programming issue?

If **Run Main Project** icon (▷) is selected, the device will automatically run immediately after programming. Therefore, if your code changes the Flash memory, verification could fail. To prevent the code from running after programming, select **Hold in Reset**.

My computer went into power-down/hibernate mode and now my debugger won’t work. What happened?

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

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

For dsPIC30F/33F and PIC24F/H devices, a reserved bit in the peripheral control register (usually either bit 14 or 5) is used as a Freeze bit by the debugger. If you have performed a write to the entire register, you may have overwritten this bit (the bit is user-accessible in Debug mode).

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

When using a 16-bit device, 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, for example,

```
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)`

7. Error Messages

The MPLAB PICkit Basic In-Circuit Debugger produces various error messages; some are specific and others can be resolved with general corrective actions. In general, read any instructions under your error message. If these fail to fix the problem or if there are no instructions, refer to the following sections.

7.1 Types of Error Messages

7.1.1 Debugger-to-Target Communications Errors

Failed to send database

If you receive this error:

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

If these fail to fix the problem or if there are no instructions, see 7.2.3. Debugger to Computer Communication Error Actions.

7.1.2 Corrupted/Outdated Installation Errors

Failed to download firmware

If the hex file exists:

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

If the hex file does not exist:

- Reinstall MPLAB X IDE or MPLAB IPE.

Unable to download debug executive

If you receive this error while attempting to debug:

1. Deselect the debugger as the debug tool.
2. Close your project and then close MPLAB X IDE or MPLAB IPE.
3. Restart MPLAB X IDE or MPLAB IPE and reopen your project.
4. Reselect the debugger as the debug tool and attempt to program the target device again.

Unable to download program executive

If you receive this error while attempting to program:

1. Deselect the debugger as the programmer.
2. Close your project and then close MPLAB X IDE or MPLAB IPE.
3. Restart MPLAB X IDE or MPLAB IPE and reopen your project.
4. Reselect the debugger as the programmer and attempt to program the target device again.

If these actions fail to fix the problem or if there are no instructions, see Corrupted Installation Actions.

7.1.3 Debug Failure Errors

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 if you try to Run before programming your device for the first time. If you receive this message after this, or immediately after programming your device, please refer to 7.2.6. Debug Failure Actions.

The device is code protected.

The device on which you are attempting to operate (read, program, blank check or verify) is code protected, in other words, the code cannot be read or modified. Check your Configuration bits setting for code protection (**Windows** > **Target Memory Views** > **Configuration Bits**).

Disable code protection, set or clear the appropriate Configuration bits in code or in the **Configuration Bits** window according to the device data sheet. Then erase and reprogram the entire device.

If these actions fail to fix the problem, see Debugger to Target Communication Error Actions and 7.2.6. Debug Failure Actions.

7.1.4 Miscellaneous Errors

MPLAB PICkit Basic is busy. Please wait for the current operation to finish.

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 **Finish Debugger Session** ⏹ to stop any running applications. Then, try to deselect the debugger again.
3. Unplug the debugger from the computer. Then, try to deselect the debugger again.
4. Shut down MPLAB X IDE.

7.1.5 List of Error Messages

Table 7-1. Alphabetized List Of Error Messages

AP_VER=Algorithm Plugin Version
AREAS_TO_PROGRAM=The following memory area(s) will be programmed:
AREAS_TO_READ=The following memory area(s) will be read:
AREAS_TO_VERIFY=The following memory area(s) will be verified:
BLANK_CHECK_COMPLETE=Blank check complete, device is blank.
BLANK_CHECK_FAILED=Blank check failed. The device is not blank.
BLANK_CHECKING=Blank Checking...
BOOT_CONFIG_MEMORY=boot config memory
BOOT_VER=Boot Version
BOOTFLASH=boot flash
BP_CANT_B_DELETED_WHEN_RUNNING=software breakpoints cannot be removed while the target is running. The selected breakpoint will be removed the next time the target halts.
CANT_CREATE_CONTROLLER=Unable to find the tool controller class.
CANT_FIND_FILE=Unable to locate file %s.
CANT_OP_BELOW_LVPTHRESH=The voltage level selected %f, is below the minimum erase voltage of %f. The operation cannot continue at this voltage level.
CANT_PGM_USEROTP=The debug tool cannot program User OTP memory because it is not blank. Please exclude User OTP memory from the memories to program or switch to a device with blank User OTP memory.
CANT_PRESERVE_PGM_MEM=Unable to preserve program memory: Invalid range Start = %08x, End = %08x.
CANT_READ_REGISTERS=Unable to read target register(s).
CANT_READ_SERIALNUM=Unable to read the device serial number.

CANT_REGISTER_ALTERNATE_PNP=Unable to register for PNP events for multiple USB product IDs.
CANT_REMOVE_SWPS_BUSY=The PICkit Basic is currently busy and cannot remove software breakpoints at this time.
CHECK_4_HIGH_VOLTAGE_VPP=CAUTION: Check that the device selected in MPLAB IDE (%s) is the same one that is physically attached to the debug tool. Selecting a 5V device when a 3.3V device is connected can result in damage to the device when the debugger checks the device ID. Do you wish to continue?
CHECK_PGM_SPEED=You have set the program speed to %s. The circuit on your board may require you to slow the speed down. Please change the setting in the tool properties to low and try the operation again.
CHECK_SLAVE_DEBUG=Debugging may have failed because the, "Debug" check box in the Slave Core settings of the master project has not been enabled. Please make sure this setting is enabled.
COMM_PROTOCOL_ERROR=A communication error with the debug tool has occurred. The tool will be reset and should re-enumerate shortly.
COMMAND_TIME_OUT=PICkit Basic has timeout out waiting for a response to command %02x.
CONFIGURATION=configuration
CONFIGURATION_MEMORY=configuration memory
CONNECTION_FAILED=Connection Failed
CORRUPTED_STREAMING_DATA=Invalid streaming data has been detected. Run time watch or trace data may no longer be valid. It is recommended that you restart your debug session.
CPM_TO_TARGET_FAILED=An exception occurred during ControlPointMediator.ToTarget().
DATA_FLASH_MEMORY=Data Flash memory
DATA_FLASH=data flash
DEBUG_INFO_PGM_FAILED=Could not enter debug mode because programming the debug information failed. Invalid combinations of config bits may cause this problem.
DEBUG_READ_INFO=Reading the device while in debug mode may take a long time due to the target oscillator speed. Reducing the range that you'd like to read (under the ICD 4 project properties) can mitigate the situation. The abort operation can be used to terminate the read operation if necessary.
DEVICE_ID_REVISION=Device ID Revision
DEVICE_ID=Device ID
DEVICE_INFO_CONFIG_BITS_MASK=Address = %08x, Mask = %08x
DEVICE_INFO_MEMBERS=DeviceInfo: pcAddress = %08x, Vpp = %.2f, useRowEraseIfVoltageIsLow = %s, voltageBelowWhichUseRowErase = %.2f, deviceName = %s, programmerType = %s
DEVICE_INFO_MEMINFO_MEMBERS= DeviceInfo: mask = %04x, exists = %s, startAddr = %08x, endAddr = %08x, rowSize = %04x, rowEraseSize = %04x, addrInc = %04x, widthProgram = %04x
DEVICE_INFO=DeviceInfo: Values:
DEVID_MISMATCH=Target Device ID (0x%x) is an Invalid Device ID. Please check your connections to the Target Device.
DFU_NOT_SUPPORTED=MPLAB X has detected the tool connected has capabilities that this version does not support. Please download the latest version of MPLAB X to use this tool.
DISCONNECT_WHILE_BUSY=The tool was disconnected while it was busy.
EEDATA_MEMORY=EEData memory
EEDATA=EEData
EMPTY_PROGRAM_RANGES=The programming operation did not complete because no memory areas have been selected.
EMULATION_MEMORY_READ_WRITE_ERROR=An error occurred while trying to read/write MPLAB's emulation memory: Address=%08x
END=end
ENSURE_SELF_TEST_READY=Please ensure the RJ-11 cable is connected to the test board before continuing.
ENSURE_SELF_TEST_READY=Please ensure the RJ-11 cable is connected to the test board before continuing. Would you like to continue?
ENV_ID_GROUP=Device Identification
ERASE_COMPLETE=Erase successful
ERASING=Erasing...
FAILED_2_PGM_DEVICE=Failed to program device.
FAILED_CREATING_COM=Unable create communications object.

FAILED_CREATING_DEBUGGER_MODULES=Initialization failed: Failed creating the debugger module.
FAILED_ERASING=Failed to erase the device.
FAILED_ESTABLISHING_COMMUNICATION=Unable to establish tool communications.
FAILED_GETTING_DBG_EXEC=A problem occurred while trying to load the debug executive.
FAILED_GETTING_DEVICE_INFO=Initialization failed: Failed while retrieving device database (.pic) information.
FAILED_GETTING_EMU_INFO=Initialization failed: Failed getting emulation database information.
FAILED_GETTING_HEADER_INFO=Initialization failed: Failed getting header database information.
FAILED_GETTING_PGM_EXEC=A problem occurred while trying to load the program executive.
FAILED_GETTING_TEX=Unable to obtain the ToolExecMediator
FAILED_GETTING_TOOL_INFO=Initialization failed: Failed while retrieving tool database (.ri4) information.
FAILED_INITING_DATABASE=Initialization failed: Unable to initialize the tool database object.
FAILED_INITING_DEBUGHANDLER=Initialization failed: Unable to initialize the DebugHandler object.
FAILED_PARSING_FILE=Failed to parse firmware file: %s
FAILED_READING_EMULATION_REGS=Failed to read emulation memory.
FAILED_READING_MPLAB_MEMORY=Unable to read %s memory from %0x08 to %0x08.
FAILED_READING_SECURE_SEGMENT=A failure occurred while reading secure segment configuration bits.
FAILED_SETTING_PC=Unable to set PC.
FAILED_SETTING_SHADOWS=Failed to properly set shadow registers.
FAILED_SETTING_XMIT_EVENTS=Unable to synchronize run time data semiphores.
FAILED_STEPPING=Failed while stepping the target.
FAILED_TO_GET_DEVID=Failed to get Device ID. Please make sure the target device is attached and try the operation again.
FAILED_TO_INIT_TOOL=Failed to initialize PICkit Basic
FAILED_UPDATING_BP=Failed to update breakpoint:\nFile: %s\naddress: %08x
FAILED_UPDATING_FIRMWARE=Failed to properly update the firmware.
FILE_REGISTER=file register
FIRMWARE_DOWNLOAD_TIMEOUT=PICkit Basic timeout out during the firmware download process.
FLASH_DATA_MEMORY=Flash data memory
FLASH_DATA=flash data
FRCINDEBUG_NEEDS_CLOCKSWITCHING=To use FRC in debug mode the clock switching configuration bits setting must be enabled. Please enable clock switching and retry the requested operation.
FW_DOESNT_SUPPORT_DYNBP=The current PICkit Basic firmware does not support setting run time breakpoints for the selected device. Please download firmware version %02x.%02x.%02x or higher.
GOOD_ID_MISMATCH=Target Device ID (0x%x) is a valid Device ID but does not match the expected Device ID (0x%x) as selected.
HALTING=Halting...
HIGH=High
HOLDMCLR_FAILED=Hold in reset failed.
IDS_SELF_TEST_BOARD_PASSED=PICkit Basic is functioning properly. If you are still having problems with your target circuit please check the Target Board Considerations section of the online help.
IDS_ST_CLKREAD_ERR=Test interface PGC clock line read failure.
IDS_ST_CLKREAD_NO_TEST=Test interface PGC clock line read not tested.
IDS_ST_CLKREAD_SUCCESS=Test interface PGC clock line read succeeded.
IDS_ST_CLKWRITE_ERR=Test interface PGC clock line write failure. Please ensure that the tester is properly connected.
IDS_ST_CLKWRITE_NO_TEST=Test interface PGC clock line write not tested.
IDS_ST_CLKWRITE_SUCCESS=Test interface PGC clock line write succeeded.
IDS_ST_DATREAD_ERR=Test interface PGD data line read failure.
IDS_ST_DATREAD_NO_TEST=Test interface PGD data line read not tested.
IDS_ST_DATREAD_SUCCESS=Test interface PGD data line read succeeded.

IDS_ST_DATWRITE_ERR=Test interface PGD data line write failure.
IDS_ST_DATWRITE_NO_TEST=Test interface PGD data line write not tested.
IDS_ST_DATWRITE_SUCCESS=Test interface PGD data line write succeeded.
IDS_ST_LVP_ERR=Test interface LVP control line failure.
IDS_ST_LVP_NO_TEST=Test interface LVP control line not tested.
IDS_ST_LVP_SUCCESS=Test interface LVP control line test succeeded.
IDS_ST_MCLR_ERR=Test interface $\overline{\text{MCLR}}$ level failure.
IDS_ST_MCLR_NO_TEST=Test interface $\overline{\text{MCLR}}$ level not tested.
IDS_ST_MCLR_SUCCESS=Test interface $\overline{\text{MCLR}}$ level test succeeded.
IDS_TEST_NOT_COMPLETED=Interface test could not be completed. Please contact your local FAE/CAE to SAR the unit.
INCOMPATIBLE_FW=The PICkit Basic firmware is not compatible with the current version of MPLAB X software.
INVALID_ADDRESS=The operation cannot proceed because the %s address is outside the devices address range of 0x%08x - 0x%08x.
JTAG_NEEDS_JTAGEN=The JTAG Adapter requires the JTAG enable configuration bit to be turned on. Please enable this configuration bit before continuing.
MCLR_HOLD_RESET_NO_MAINTAIN_POWER=WARNING: You are powering the target device from PICkit Basic and have not selected the, "Maintain active power" option on the PICkit Basic's Power property page. Without this option, the state of $\overline{\text{MCLR}}$ (hold/release from reset) cannot be guaranteed after the current session has ended.
MCLR_OFF_ID_WARNING=If you are using low voltage programming and the MCLRE config bit on the target device is set to OFF, this may explain why the device ID is incorrect. In this case, please switch to the \"Use high voltage programming mode entry\" Program mode entry setting on the PICkit Basic Program Options property page and try the operation again.
MCLR_OFF_WARNING=If you wish to continue with MCLRE configuration bit set to OFF, switch to the \"Use high voltage programming mode entry\" Program mode entry setting on the PICkit Basic Program Options property page.
MEM_INFO=DeviceInfo: MemInfo values:
MEM_RANGE_ERROR_BAD_END_ADDR=Invalid program range end address %s received. Please check the manual program ranges on the debug tool's, "Memories to Program" property page.
MEM_RANGE_ERROR_BAD_START_ADDR=Invalid program range start address %s received. Please check the manual program ranges on the debug tool's, "Memories to Program" property page.
MEM_RANGE_ERROR_END_LESSTHAN_START=Invalid program range received: end address %s < start address %s. Please check the manual program ranges on the debug tool's, "Memories to Program" property page.
MEM_RANGE_ERROR_ENDADDR_NOT_ALIGNED=Invalid program range received: end address %s is not aligned on a proper 0x%x address boundary. Please check the manual program ranges on the debug tool's, "Memories to Program" property page.
MEM_RANGE_ERROR_STARTADDR_NOT_ALIGNED=Invalid program range received: start address %s is not aligned on a proper 0x%x address boundary. Please check the manual program ranges on the debug tool's, "Memories to Program" property page.
MEM_RANGE_ERROR_UNKNOWN=An unknown error has occurred while trying to validate the user entered memory ranges.
MEM_RANGE_ERROR_WRONG_DATABASE=Unable to access data object while validating user entered memory ranges.
MEM_RANGE_OUT_OF_BOUNDS=The selected program range, %s, does not fall within the proper range for the memory area selected. Please check the manual program ranges on the debug tool's, "Memories to Program" property page.
MEM_RANGE_STRING_MALFORMED=The memory range(s) entered on the, "Memories to Program" property page (%s) is not formatted properly.
MISSING_BOOT_CONFIG_PARAMETER=Unable to find boot config start/end address in database.
MUST_NOT_USE_LVP_WHEN_LVPCFG_OFF=MPLAB has detected that the low voltage configuration bit on the device is off and you have selected the low voltage programming option on the debug tool's property page. If you wish to use the low voltage programming option you must first do the following:\n* Turn off the low voltage programming option on the debug tool's Program Options property page\n* Program the low voltage configuration bit to on\n* Turn on the low voltage programming option on the debug tool's Program Options property page.
MUST_SET_LVPBIT_WITH_LVP=The low voltage programming feature requires the LVP configuration bit to be enabled on the target device. Please enable this configuration bit and try the operation again.
NEW_FIRMWARE_NO_DEVICE=Downloading firmware.
NEW_FIRMWARE=Now Downloading new Firmware for target device: %s
NMMR=NMMR

NO_DYNAMIC_BP_SUPPORT_AT_ALL=The current device does not support the ability to set breakpoints while the devices is running. The breakpoint will be applied prior to the next time you run the device.
NO_PGM_HANDLER=Cannot program software breakpoints. The program handler has not been initialized.
NO_PROGRAMMING_ATTEMPTED=MPLAB's memory is blank so no programming operation was attempted.
NORMAL=Normal
OP_FAILED_FROM_CP=The requested operation failed because the device is code protected.
OpenIDE-Module-Name=PICkit Basic
OPERATION_INFO_MEMBERS=OperationInfo: Type = %s, Mask = %08x, Erase = %s, Production Mode = %s.
OPERATION_INFO_TRANSFER_INFO_MEMBERS=OperationInfo: Start = %x, End = %x, Buffer Length = %d, Type = %s, Mask = %08x.
OPERATION_INFO=OperationInfo: Values:
OPERATION_NOT_SUPPORTED=This operation is not supported for the selected device
OUTPUTWIN_TITLE=PICkit Basic
PERIPHERAL=Peripheral
POWER_ERROR_NO_POWER_SRC=The configuration is set for the target board to supply its own power but no voltage has been detected on VDD. Please ensure you have your target powered up and try again.
POWER_ERROR_POWER_SRC_CONFLICT=The configuration is set for the tool to provide power to the target but there is voltage already detected on VDD. This is a conflict. Please ensure your target is not supplying voltage to the tool and try again.
POWER_ERROR_SLOW_DISCHARGE= There seems to be excessive capacitance on VDD causing a slower system discharge and shutdown. Consider minimizing overall capacitance loading or use power from your target to avoid discharge delays.
POWER_ERROR_UNKNOWN=An unknown power error has occurred.
POWER_ERROR_VDD_TOO_HIGH=The VDD voltage desired is out of range. It exceeds the maximum voltage of 5.5V.
POWER_ERROR_VDD_TOO_LOW=The VDD voltage desired is out of range. It is below the minimum voltage of 1.5V.
POWER_ERROR_VPP_TOO_HIGH=The VPP voltage desired is out of range. It exceeds the maximum voltage of 14.2V.
POWER_ERROR_VPP_TOO_LOW=The VPP voltage desired is out of range. It is below the minimum voltage of 1.5V.
PRESERVE_MEM_RANGE_ERROR_BAD_END_ADDR=Invalid preserve range end address %s received. Please check the manual program ranges on the debug tool's, "Memories to Program" property page.
PRESERVE_MEM_RANGE_ERROR_BAD_START_ADDR=Invalid preserve range start address %s received. Please check the manual program ranges on the debug tool's, "Memories to Program" property page.
PRESERVE_MEM_RANGE_ERROR_END_LESSTHAN_START=Invalid preserve range received: end address %s < start address %s. Please check the manual program ranges on the debug tool's, "Memories to Program" property page.
PRESERVE_MEM_RANGE_ERROR_ENDADDR_NOT_ALIGNED=Invalid preserve range received: end address %s is not aligned on a proper 0x%x address boundary. Please check the manual program ranges on the debug tool's, "Memories to Program" property page.
PRESERVE_MEM_RANGE_ERROR_STARTADDR_NOT_ALIGNED=Invalid preserve range received: start address %s is not aligned on a proper 0x%x address boundary. Please check the manual program ranges on the debug tool's, "Memories to Program" property page.
PRESERVE_MEM_RANGE_ERROR_UNKNOWN=An unknown error has occurred while trying to validate the user entered preserve ranges.
PRESERVE_MEM_RANGE_ERROR_WRONG_DATABASE=Unable to access data object while validating user entered memory ranges.
PRESERVE_MEM_RANGE_MEM_NOT_SELECTED=You have selected to preserve an area of memory but have not selected to program that area. Please check the preserved ranges on the debug tool's "Memories to Program" property page and make sure that any preserved memory is also designated to be programmed.
PRESERVE_MEM_RANGE_OUT_OF_BOUNDS=The selected preserve range, %s, does not fall within the proper range for the memory area selected. Please check the manual program ranges on the debug tool's "Memories to Program" property page.
PRESERVE_MEM_RANGE_STRING_MALFORMED=The preserve memory range(s) entered on the, "Memories to Program" property page (%s) is not formatted properly.
PRESERVE_MEM_RANGE_WONT_BE_PROGRAMMED_AUTO_SELECT=Some or all of the preserve memory ranges (%s) entered on the, "Memories to Program" property page, do not fall under the indicated program range(s) (%s) for the memory selected. Please deselect the "Auto select memories and ranges" option on the "Memories to Program" property page, change to manual mode and adjust your range(s) accordingly.

PRESERVE_MEM_RANGE_WONT_BE_PROGRAMMED=Some or all of the preserve memory ranges (%s) entered on the, "Memories to Program" property page, do not fall under the indicated program range(s) (%s) for the memory selected. Please check the preserved ranges on the debug tool's, "Memories to Program" property page.
PROGRAM_CFG_WARNING=WARNING: You have selected to program configuration memory. Programming invalid values into any of the configuration fields may have unintended consequences. Please make sure that EVERY configuration field has a valid value. If you are not sure, you can read the configuration values off of device first and then change only the fields you are concerned with. Would you like to continue programming?
PROGRAM_COMPLETE=Programming/Verify complete
PROGRAM_MEMORY=program memory
PROGRAM=program
PROGRAMMING_DID_NOT_COMPLETE=Programming did not complete.
READ_COMPLETE=Read complete
READ_DID_NOT_COMPLETE=Read did not complete.
RELEASEMCLR_FAILED=Release from reset failed.
REMOVING_SWBPS_COMPLETE=Removing software breakpoints complete.
REMOVING_SWBPS=Removing software breakpoints...
RESET_FAILED=Failed to reset the device.
RESETTING=Resetting...
RISKY_CFG_RANGE_REMOVED=The configuration memory will not be included in the program operation because the, "Exclude configuration memory from programming" option is set. To change this, go to the Memories to Program property page and uncheck the setting. WARNING: Programming configuration values on this device can cause unintended consequences if all of the configuration values are not properly set. It is advised that you read the configuration values off of device first and then change only the fields you are concerned with.
RUN_INTERRUPT_THREAD_SYNCH_ERROR=An internal run error has occurred. It is advised that you restart your debug session. You may continue running but certain run time features may no longer work properly.
RUN_TARGET_FAILED=Unable to run the target device.
RUNNING=Running
SERIAL_NUM=Serial Number:
SETTING_SWBPS=Setting software breakpoints.......
STACK=stack
START_AND_END_ADDR=start address = 0x%x, end address = 0x%x
START=start
TARGET_DETECTED=Target voltage detected
TARGET_FOUND=Target device %s found.
TARGET_HALTED=Target Halted
TARGET_NOT_READY_4_DEBUG=The target device is not ready for debugging. Please check your configuration bit settings and program the device before proceeding. The most common causes for this failure are oscillator and/or PGC/PGD settings.
TARGET_VDD=Target VDD:
TEST=test
TOOL_INFO_MEMBERS=ToolInfo: speedLevel = %d, PGCResistance = %d, PGDResistance = %d, PGCPullDir = %s, PGDPullDir = %s, ICSPSelected = %s.
TOOL_INFO=ToolInfo: Values:
TOOL_IS_BUSY=PICkit Basic is busy. Please wait for the current operation to finish.
TOOL_SUPPLYING_POWER=PICkit Basic is supplying power to the target (%.2f volts).
TOOL_VDD=VDD:
TOOL_VPP=VPP:
UNABLE_TO_OBTAIN_RESET_VECTOR=PICkit Basic was unable to retrieve the reset vector address. This indicates that no _reset symbol has been defined and may prevent the device from starting up properly.
UNKNOWN_MEMTYPE=Unknown memory type

UNLOAD_WHILE_BUSY=PICkit Basic was unloaded while still busy. Please unplug and reconnect the USB cable before using PICkit Basic again.
UPDATING_APP=Updating firmware application...
UPDATING_BOOTLOADER=Updating firmware bootloader.
USE_LVP_PROGRAMMING=NOTE: If you would like to program this device using low voltage programming, select Cancel on this dialog. Then go to the PICkit Basic node of the project properties and check the Enable Low Voltage Programming check box of the Program Options Option Category pane (low voltage programming is not valid for debugging operations).
USERID_MEMORY=User Id Memory
USERID=user Id
VERIFY_COMPLETE=Verification successful
VERIFY_FAILED=Verify failed
VERSIONS=Versions
VOLTAGE_LEVEL_BAD_VALUE_EX=You have entered an invalid value %s for the Voltage Level on the PICkit Basic Power property page. Please fix this before continuing.
VOLTAGE_LEVEL_BAD_VALUE=Unable to parse the voltage level %s. Please enter a valid voltage entry.
VOLTAGE_LEVEL_OUT_OF_RANGE=The target voltage level you have entered, %.3f, is outside the range of the device %.3f - %.3f.
VOLTAGES=Voltages
WOULD_YOU_LIKE_TO_CONTINUE=Would you like to continue?
WRONG_PICkit Basic_FLAVOR=Your PICkit Basic hardware needs updating please, contactPICkit Basic_Update@microchip.com to get a replacement.

7.2 General Corrective Actions

7.2.1 Read/Write Error Actions

If you receive a read or write error:

1. Did you click **Debug** > **Reset**? 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 Options** for the debugger in the **Project Properties** window.
6. Ensure that the cable(s) are of the correct length.

Related Links

8.4. Debug Options

7.2.2 Debugger to Target Communication Error Actions

If the MPLAB PICkit Basic In-Circuit Debugger and the target device are *not* communicating with each other:

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

7.2.3 Debugger to Computer Communication Error Actions

If the MPLAB PICkit Basic In-Circuit Debugger and MPLAB X IDE or MPLAB IPE are not communicating with each other:

1. Unplug and then plug in the debugger.
2. Reconnect to the debugger.

3. Try the operation again. It is possible the error was a one-time event.
4. The version of MPLAB X IDE or MPLAB IPE installed may be incorrect for the version of firmware loaded on the MPLAB PICkit Basic In-Circuit Debugger. Follow the steps outlined in 7.2.4. Corrupted Installation Actions.
5. There may be an issue with the computer USB port. See section 7.2.5. USB Port Communication Error Actions.

7.2.4 Corrupted Installation Actions

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

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

7.2.5 USB Port Communication Error Actions

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

1. Reconnect to the MPLAB PICkit Basic In-Circuit Debugger.
2. Make sure the debugger is physically connected to the computer on the appropriate USB port.
3. Make sure the appropriate USB port has been selected in the debugger options in the **Project Properties** window.
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.

7.2.6 Debug Failure Actions

The MPLAB PICkit Basic In-Circuit Debugger was unable to perform a debugging operation. Multiple scenarios can result in this error message. A few of these are described below.

1. Device does not have a clock source or the clock source selected in the config bit settings is not functional.
2. PGD (Data) and PGC (Clock) pins on the device are being used by the application. These pins are required by the debugger and should not be controlled by the application during debug mode.
3. Many devices contain more than one PGD/PGC pair. The PGD/PGC to be used for debugging is selected using the configuration bit settings. In programming mode any of the PGD/PGC pairs can be used provided it is connected to the debugging tool. For debug mode, the config bit settings must match the correct PGD/PGC pair physically connected to the MCU.
4. Selecting Run Main Project from MPLAB X IDE when device has not been programmed will generate this message. Use debug mode (▷) instead.

This information is based the Knowledge Base article Why do I get the following error while trying to debug: "ICD3Err0040: The target device is not ready for debugging"?.

7.2.7 Internal Error Actions

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

The most likely cause is a corrupted installation (7.2.4. 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 Support.

8. Debugger Function Summary

A summary of the MPLAB® PICkit™ Basic In-Circuit Debugger functions are summarized below.

8.1 Debugger Selection and Switching

Use the Project Properties dialog to select or switch debuggers for a project. To switch you must have more than one debugger connected to your computer. MPLAB X IDE will differentiate between the debuggers by displaying different serial numbers.

To select or change the debugger used for a project:

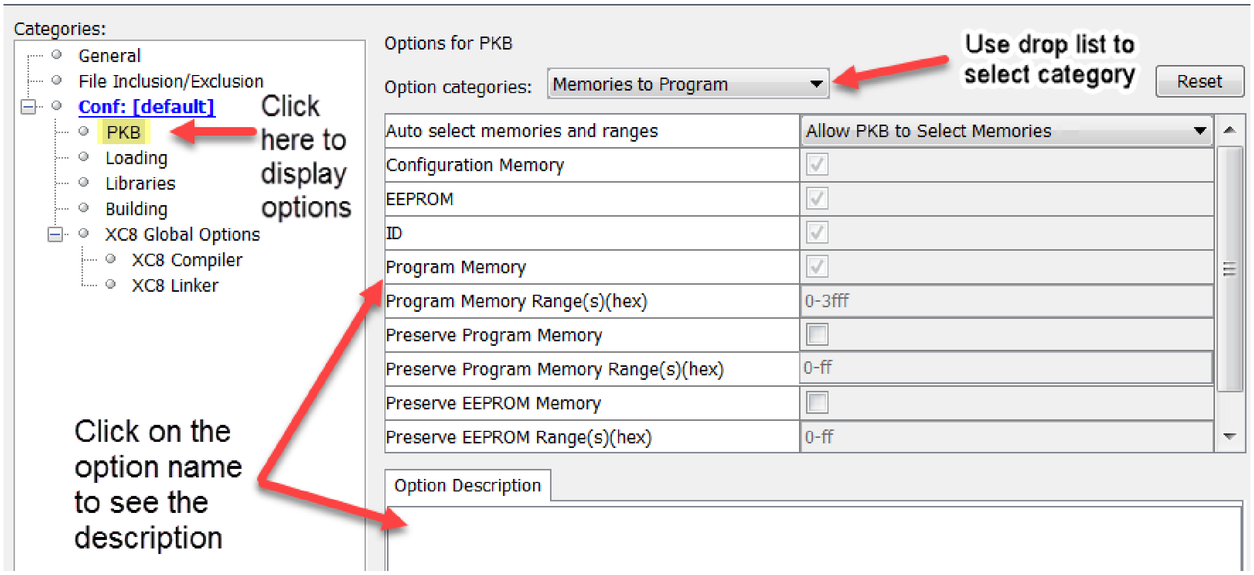
1. Open the **Project Properties** dialog by doing one of the following:
   a. Click on the project name in the **Projects** window and select **File** > **Project Properties**.
   or
   b. Right click on the project name in the **Projects** window and select **Properties**.
2. Under **Categories** on the left side, expand **Conf:[default]** to show PICkit Basic.
3. Under **Hardware Tools**, find **PICkit Basic** and click on a serial number (SN) to select a debugger for use in the project, then click **Apply**.

8.2 Debugger Options Selection

Debugger options are set in the **Project Properties** dialog of MPLAB® X IDE. Click on PKB under **Categories** to display the **Options for PKB** (see figure below). Use the **Options categories** drop-down list to select various options. Click on an option name to see its description in the **Option Description** box below. Click to the right of an option name to select or change it.

Note: The available option categories and the options within those categories are dependent on the device you have selected.

Figure 8-1. MPLAB® X IDE Options for MPLAB PICkit Basic



Categories:
- General
- File Inclusion/Exclusion
- Conf: [default]
  - PKB
  - Loading
  - Libraries
  - Building
  - XC8 Global Options
    - XC8 Compiler
    - XC8 Linker

Options for PKB

Option categories: Memories to Program

Reset

Auto select memories and ranges	Allow PKB to Select Memories
Configuration Memory	☑
EEPROM	☑
ID	☑
Program Memory	☑
Program Memory Range(s)(hex)	0-3fff
Preserve Program Memory	☐
Preserve Program Memory Range(s)(hex)	0-ff
Preserve EEPROM Memory	☐
Preserve EEPROM Range(s)(hex)	0-ff

Option Description

Click here to display options
Use drop list to select category
Click on the option name to see the description

After setting the options, click **Apply** or **OK**. Also click the Refresh Debug Tool icon in the MPLAB X IDE dashboard display to update any changes made.

For the MPLAB IPE, the options for MPLAB PICkit Basic are located in **Settings** > **Advance Mode** > **Settings**. Refer to MPLAB IPE online help for more information.

8.3 Memories to Program

Select the memories to be programmed into the target. The table below shows all the possible options, however, only those options available for your selected device will be displayed in MPLAB X IDE.

Note: If **Erase All Before Program** is selected, as shown in **Program Options**, then all device memory will be erased before programming.

Table 8-1. Memories to Program Option Category

Auto select memories and ranges	**Allow PICkit Basic to Select Memories** - The debugger uses your selected device and default settings to determine what to program. **Manually select memories and ranges** - You select the type and range of memory to program (see below).
Configuration Memory	Check to include **Configuration Memory** in the area(s) to be programmed. This is always programmed in Debug mode.
Boot Flash	Check to include **Boot Flash** memory in the area(s) to be programmed. This is always programmed in Debug mode.
EEPROM	Check to include **EEPROM** memory in the area(s) to be programmed.
ID	Check to program the user ID.
Program Memory	Check to program the target program memory range specified below.
Program Memory Range(s) (hex)	The range(s) of program memory to be programmed. These are the starting and ending hex address range(s) in program memory for programming, reading, or verification. Each range must be two hex numbers (the start and end addresses of the range) separated by a dash. Multiple ranges must be separated by a comma (for example, 0-ff, 200-2ff). Ranges must be aligned on a 0x800 address boundary. **Note:** The address range does not apply to the Erase function. The Erase function will erase all data on the device.
Preserve Program Memory	Enabling this option will cause the current program memory on the device to be read into MPLAB X IDE's memory and then reprogrammed back to the target device when programming is done. The range(s) of program memory that will be preserved is determined by the Preserve Program Memory Range(s) option below. Ensure that code is NOT code protected.
Preserve Program Memory Range(s) (hex)	The range(s) of program memory to be preserved. Each range must be two hex numbers, representing the start and end addresses of the range, separated by a dash. Ranges must be separated by a comma (for example, 0-ff, 200-2ff). Areas are reserved by reading them into MPLAB X IDE and then programming them back down when a program operation occurs. Thus the preserved areas must lie within a memory range that will be programmed.
Preserve (Type of) Memory	Enabling this option will cause the current memory type on the device to be read into MPLAB X IDE's memory and then reprogrammed back to the target device when programming is done. Check to preserve *Memory* for reprogramming, where *Memory* is the type of memory. Types include: EEPROM, ID, Boot Flash, and Auxiliary. Ensure that code is NOT code protected.
Preserve (Type of) Memory Range(s) (hex)*	The range(s) of the memory type to be preserved. Each range must be two hex numbers, representing the start and end addresses of the range, separated by a dash. Ranges must be separated by a comma (for example, 0-ff, 200-2ff). Areas are reserved by reading them into MPLAB X IDE and then programming them back down when a program operation occurs. Thus the preserved areas must lie within a memory range that will be programmed. *Memory* is the type of memory, which includes EEPROM, ID, Boot Flash, and Auxiliary. Ensure that code is NOT code protected.
* If you receive a programming error due to an incorrect range, ensure the range does not exceed available/remaining device memory.	

8.4 Debug Options

If this option is available for the project device, you can select to use software breakpoints.

Table 8-2. Debug Option Category

Debug startup	Begin a debug session after device startup.
Debug reset	Begin a debug session after a reset.
Use Software Breakpoints	Use Software Breakpoints.

Table 8-3. Software vs. Hardware Breakpoints

Features	Software Breakpoints	Hardware Breakpoints
Number of breakpoints	Unlimited	Limited
Breakpoints are written to	Program Memory	Debug Registers
Time to set breakpoints	Oscillator Speed Dependent – can take minutes	Minimal
Skidding	No	Yes
Note: Using software breakpoints for debugging impacts device endurance. Therefore, it is recommended that devices used in this manner not be used as production parts.		

8.5 Program Options

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

Table 8-4. Program Option Category

Erase All Before Program	Enabling this option will cause the entire device to be erased prior to programming the data from MPLAB X IDE. Any memory areas designated to be preserved will be read before the device is erased and reprogrammed on the device when the device is programmed. 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.

8.6 PICkit Basic Tool Options

Options specifically for the PICkit Basic tool.

Table 8-5. Program Option Category

Programming mode entry	Use low voltage programming mode entry - always selected PICkit Basic does not support the high voltage Vpp option for placing the target device in programming mode and only supports the low voltage method (Vpp will not exceed the Vdd supply voltage. Instead a test pattern will be used on Vpp).
Program Speed	Select the speed the debugger will use to program the target as either Low, Normal or High. The default is Normal. If programming should fail, using a slower speed may solve the problem.

8.7 Freeze Peripherals

Select from the list of peripherals to freeze or not freeze on program halt. The available peripherals are device dependent.

PIC12/16/18 MCU Devices

To freeze/unfreeze all device peripherals on halt, check/uncheck the **Freeze on Halt** check box. If this does not halt your desired peripheral, be aware that some peripherals do not have a freeze-on-halt capability and cannot be controlled by the debugger.

dsPIC, PIC24 and PIC32 Devices

Select the peripheral’s check box in the **Peripherals to Freeze on Halt** list 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 do not have a 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**.

8.8 Secure Segment

Select and load debugger firmware.

Table 8-6. Secure Segment Option Category

Segments to be Programmed	Select one of the following: 1. Full Chip Programming (default). 2. Boot, Secure and General Segments. 3. Secure and General Segments. 4. General Segment Only.

8.9 Clock

Set the option to use the fast internal RC (FRC) clock for the selected device.

Table 8-7. Clock Option Category

Use FRC in Debug mode (dsPIC33F and PIC24F/H devices only)	When debugging, use the device fast internal RC (FRC) for clocking instead of the oscillator specified for the application. This is useful when the application clock is slow. Checking this check box will let the application run at the slow speed but debug at the faster FRC speed. Reprogram after changing this setting. **Note:** Peripherals that are not frozen will operate at the FRC speed while debugging.

8.10 Tool Pack Selection

Select and load debugger firmware.

Table 8-8. Tool Pack Selection Category

Tool pack update options	Select either Use latest installed tool pack (recommended) or Use specific tool pack.
Specifically selected version	Press to select which tool pack to use. When pressed, the Select Tool pack dialog opens from which to select the version you want.

8.11 Communication

Set the option(s) to use for your device and type of target communication.

Table 8-9. Communication Option Category

Interface	Select the interface from the available options based on the project device.
Speed (MHz)	Enter a speed based on the available range for the interface.

Important: For low pin count AVR devices with UPDI, PICkit Basic cannot generate the high voltage pulse to reactivate the UPDI interface if the UPDI pin is configured as GPIO or RESET by configuring the RSTPINCFG configuration bits. A different tool will need to be used to do this, such as the MPLAB PICkit 5.

8.12 Event Recorder

Specify options for the Event Recorder. Find out more about the Event Recorder in the *MPLAB® X IDE User's Guide* (DS-50002027) or MPLAB X IDE WebHelp.

Table 8-10. Event Recorder Option Category

Enable	Check to enable Event Recorder
SCVD Files	Specify SCVD files to be used in project

9. Hardware Specification

The hardware and electrical specifications of the MPLAB PICkit Basic In-Circuit Debugger system are detailed in this section.

9.1 USB Connector Specifications

The MPLAB PICkit Basic In-Circuit Debugger is connected to the host computer via a USB Type-C® connector, version 2.0 compliant. The USB Type-C® connector is located on the top of the debugger.

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.

Note: The MPLAB PICkit Basic In-Circuit Debugger is powered through its USB Type-C® connector. The target board is powered from its own supply. The MPLAB PICkit Basic cannot power the target board.

Cable Length – The computer-to-debugger cable, shipped with the debugger kit, is the correct length for proper operation.

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

Computer Hibernate/Power-Down Modes – Disable the hibernate or other power saver modes on your computer to ensure proper USB communications with the debugger.

9.2 MPLAB PICkit Basic In-Circuit Debugger

The debugger unit consists of the following:

1. An internal main board with:
   a. USB Type-C® connector
   b. 8-pin SIL header (0.100" spacing) for target connections
   c. Two LEDs to display the operational modes of the debugger
   d. Emergency Recovery Button for use with the emergency boot firmware recovery utility only
2. Board enclosure with a color-coded signals label.

Also included with the debugger unit:

1. USB Type-C® high-quality, **USB High Speed**, 1.5 meter cable used to connect the debugger to the computer.
2. 8-pin SIL connector with 17 cm color-coded wires that map to the enclosure color-coded signals label.
3. 8-pin to 10-pin ARM SWD Adapter Board to connect to an ARM target for use with SWD.

9.2.1 Board Specifications

The circuit board includes the following features:

- A 32-bit microcontroller using an Arm® Cortex®-M7 core which includes memory for holding the program code image. This image is used for programming the on-board Flash device.
- A USB Type-C® interface capable of USB speeds of 480 Mbps.
- Active and Status LEDs.

9.2.2 LEDs

The MPLAB PICkit Basic has two fixed color LEDs. The Active LED is green and the Status LED is yellow. The expected start-up LED sequence for the PICkit Basic debugger is steady on Green, yellow off. The following tables describe the normal and error LED modes.

Table 9-1. Normal Modes LED Descriptions

LED	Color	Description
Active, on	Green	Power is connected; debugger in standby.
Status, on (or pulsing activity)	Yellow	Debugger is busy; activity during an operation.

Table 9-2. Error LED Descriptions

Errors	Description
Status, on 3 seconds	Bootloader problem accessing the serial EEPROM.
Status, on 10 seconds	API commands cannot be processed by the Bootloader.
Active and Status, fast blink (alternating)	A runtime exception occurred in the tool firmware.
Active and Status, fast blink (in tandem)	A runtime exception occurred in the Bootloader.

9.2.3 Color-Coded Signal Label

MIPS EJTAG	Cortex SWD	debugWIRE	PDI UPDI	AVR ISP	ICSP	Row Number	Row Color
MCLR	MLCR				MLCR	1	Orange
VIO_REF	VTG	VTG	VTG	VTG	VDD	2	Red
GND	GND	GND	GND	GND	GND	3	Brown
TDO	SWO		DAT	MISO	DAT	4	Yellow
TCK	SWCLK	dW		SCK	CLK	5	Green
			CLK	RESET	AUX	6	Blue
TDI				MOSI		7	Purple
TMS	SWDIO					8	Gray

For more detailed information, see 3.3.1. Target Connection Pinout.

9.3 Communication Hardware

For standard debugger communication with a target (see 3.3. Target Connections), connect the debugger directly to the target. The debugger has an 8-pin SIL header. If the target has a 6-pin connector, make sure to align the Pin 1 appropriately.

9.3.1 Standard Communication

The main interface to the target processor is via standard communication. It contains the connections to the V_{DD} reset, clock and data connections that are required for programming and connecting with the target devices.

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

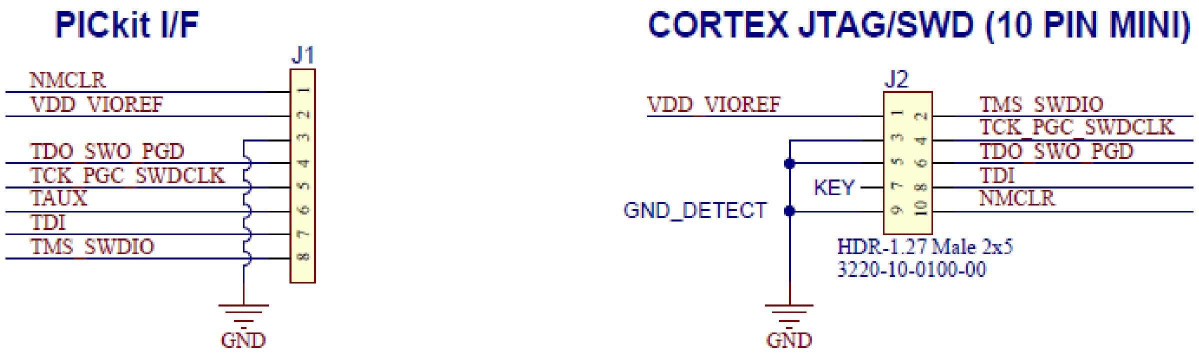
- Clock and data signals are in high-impedance mode (even when no power is applied to the MPLAB PICkit Basic In-Circuit Debugger system).

Table 9-3. Electrical Logic Table

Logic Inputs	V_{IH} = V_{DD} x 0.7V (min.)			
	V_{IL} = V_{DD} x 0.3V (max.)			
Logic Outputs	V_{DD} = 5V	V_{DD} = 3V	V_{DD} = 2.3V	V_{DD} = 1.4V
	V_{OH} = 3.8V min.	V_{OH} = 2.4V min.	V_{OH} = 1.9V min.	V_{OH} = 1.0V min.
	V_{OL} = 0.55V max.	V_{OL} = 0.55V max.	V_{OL} = 0.3V max.	V_{OL} = 0.1V max.

9.3.2 Arm® /SWD Adapter Board Schematics

Figure 9-1. Adapter Board Pins/Schematic



9.4 Target Board Considerations

The target board should be powered according to the requirements of the selected device and the application.

Note: Stresses above those listed under "Absolute Maximum Ratings" in the Electrical Characteristics chapter of the device's data sheet may cause permanent damage to the device. This is a stress rating only and functional operation of the device at those or any other conditions, above those indicated in the operation listings of this specification, is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability.

The debugger does sense target voltage.

Depending on the type of debugger-to-target communication that is used, there are some considerations for target board circuitry.

Related Links

3.3.7.2. ICSP Target Connection Circuitry

4.7.2.1. Circuits That Will Prevent the Emulator From Functioning

10. Revision History

The following is a list of changes by version to this document.

Note: Some revision letters are not used - the letters `I` and `O` - as they can be confused for numbers in some fonts.

10.1 Revision A (February 2025)

Initial release of this document.

11. Glossary

Absolute Section

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

Absolute Variable/Function

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

Access Memory

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

Access Entry Points

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

Address

A value that identifies a location in memory.

Alphabetic Character

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

Alphanumeric

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

ANDed Breakpoints

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

Anonymous Structure

16-bit C Compiler – An unnamed structure.

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

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

ANSI

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

Application

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

Archive/Archiver

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

ASCII

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

Assembly/Assembler

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

Assigned Section

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

Asynchronously

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

Asynchronous Stimulus

Data generated to simulate external inputs to a simulator device.

Attribute

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

Attribute, Section

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

AVR MCUs

AVR® microcontrollers (MCUs) refer to all Microchip AVR 8-bit microcontroller families.

Binary

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

Bookmarks

Use bookmarks to easily locate specific lines in a file.

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

C/C++

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

Calibration Memory

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

Central Processing Unit

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

Clean

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

COFF

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

Command Line Interface

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

Compiled Stack

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

Compiler

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

Conditional Assembly

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

Conditional Compilation

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

Configuration Bits

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

Constant

Represents an immediate value such as a definition through the C code #define directive or the assembly .equ directive.

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.

Debug/Debugger

See *ICE/ICD*.

Debugging Information

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

Deprecated Features

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

Device Programmer

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

Digital Signal Processing\Digital Signal Processor

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

Directives

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

Download

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

dsPIC DSCs

dsPIC® digital signal controllers (DSCs) refer to the Microchip family of microcontrollers with digital signal processing capability.

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/Emulator

See *ICE/ICD*.

Endianness

The ordering of bytes in a multi-byte object.

Environment

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

Epilogue

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

EPROM

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

Error/Error File

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

Event

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

Executable Code

Software that is ready to be loaded for execution.

Export

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

Expressions

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

Extended Microcontroller Mode

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

Extended Mode (PIC18 MCUs)

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

External Label

A label that has external linkage.

External Linkage

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

External Symbol

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

External Symbol Resolution

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

External Input Line

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

External RAM

Off-chip Read/Write memory.

Fatal Error

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

File Registers

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

Filter

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

Fixup

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

Flash

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

FNOP

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

Frame Pointer

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

Free-Standing

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

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 run-time.

Hex Code/Hex File

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

Hexadecimal

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

High Level Language

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

ICE/ICD

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

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

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

ICSP

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

IDE

Integrated Development Environment, as in MPLAB X IDE.

Identifier

A function or variable name.

IEEE

Institute of Electrical and Electronics Engineers.

Import

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

Initialized Data

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

```
int myVar=5;
```

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

Instruction Set

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

Instructions

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

Internal Linkage

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

International Organization for Standardization

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

Interrupt

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

Interrupt Handler

A routine that processes special code when an interrupt occurs.

Interrupt Service Request (IRQ)

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

Interrupt Service Routine (ISR)

Language tools: A function that handles an interrupt.

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

Interrupt Vector

Address of an interrupt service routine or interrupt handler.

L-value

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

Latency

The time between an event and its response.

Library/Librarian

See *Archive/Archiver*.

Linker

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

Linker Script Files

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

Listing Directives

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

Listing File

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

Little Endian

A data ordering scheme for 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.

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 X IDE, i.e., with a make.

Make Project

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

MCU

Microcontroller Unit. An abbreviation for microcontroller. Also uC.

Memory Model

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

Message

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

Microcontroller

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

Microcontroller Mode

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

Microprocessor Mode

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

Mnemonics

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

Module

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

MPLAB® X IDE

Microchip’s Integrated Development Environment. comes with an editor, project manager and simulator.

MPLAB X Simulator

Microchip’s simulator that works with in support of Microchip MCU, DSC and MPU devices.

MPLAB XC C Compilers

Microchip's family of C and C++ compilers comprising of the MPLAB XC8 C compiler (8-bit PIC and AVR device support), MPLAB XC16 C compiler (16-bit PIC device support), MPLAB XC-DSC C compiler (DSC device support) and MPLAB XC32 C/C++ compiler (32-bit PIC and SAM device support.)

MPLAB Xpress IDE

Microchip's Integrated Development Environment in the Cloud. MPLAB Xpress comes with an editor, project manager and simulator.

MPU

Microprocessor Unit. An abbreviation for microprocessor.

MRU

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

Native Data Size

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

Nesting Depth

The maximum level to which macros can include other macros.

Node

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 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/Object File

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

Object File Directives

Directives that are used only when creating an object file.

Octal

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

Off-Chip Memory

Off-chip memory refers to the memory selection option for the PIC18 device where memory may reside on the target board, or where all program memory may be supplied by the emulator. The

Memory tab accessed from **Options** > **Development Mode** provides the Off-Chip Memory selection dialog box.

Opcodes

Operational Codes. See *Mnemonics*.

Operators

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

OTP

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

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 PIC 8-, 16-, and 32-bit microcontroller families.

Plug-ins

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

Pod

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

Power-on-Reset Emulation

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

Pragma

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

Precedence

Rules that define the order of evaluation in expressions.

Production Programmer

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

Profile

For MPLAB X Simulator, a summary listing of executed stimulus by register.

Program Counter

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

Program Counter Unit

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

Program Memory

The memory area in a device where instructions are stored. Also, the memory in the debugger, emulator or simulator containing the downloaded target application firmware.

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 run-time model. This code executes before any user code for a given function.

Prototype System

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

Psect

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

PWM Signals

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

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.

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.

Re-entrant

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

Relaxation

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

Relocatable

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

Relocatable Section

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

Relocation

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

ROM

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

Run

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

Run-time Model

Describes the use of target architecture resources.

Run-time Watch

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

SAM MCUs/MPUs

SAM microcontrollers (MCUs) and microprocessors (MPUs) refer to all Microchip SAM 32-bit microcontroller and microprocessor families.

Scenario

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

Section

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

Section Attribute

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

Sequenced Breakpoints

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

Serialized Quick Turn Programming

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

Shell

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

Simulator

A software program that models the operation of devices.

Single Step

This command steps though code, one instruction at a time. After each instruction, 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, 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 appear on the bus as a fetch during the execution of the previous instruction; the source data address, value, and destination data address appear when the opcodes are actually executed; and the destination data value appears when the next instruction is executed. The trace buffer captures the information that is on the bus at one instance. Therefore, one trace buffer entry will contain execution information for three instructions. The number of captured cycles from one piece of information to another for a single instruction execution is referred to as the skew.

Skid

When a hardware breakpoint is used to halt the processor, one or more additional instructions may be executed before the processor halts. The number of extra instructions executed after the intended breakpoint is referred to as the skid.

Source Code

The form in which a computer program is written by the programmer. Source code is written in a formal programming language which can be translated into machine code or executed by an interpreter.

Source File

An ASCII text file containing source code.

Special Function Registers (SFRs)

The portion of data memory (RAM) dedicated to registers that control I/O processor functions, I/O status, timers or other modes or peripherals.

SQTP

See *Serialized Quick Turn Programming*.

Stack, Hardware

Locations in PIC microcontroller where the return address is stored when a function call is made.

Stack, Software

Memory used by an application for storing return addresses, function parameters, and local variables. This memory is dynamically allocated at run-time by instructions in the program. It allows for re-entrant function calls.

Stack, Compiled

A region of memory managed and allocated by the compiler in which variables are statically assigned space. It replaces a software stack when such mechanisms cannot be efficiently implemented on the target device. It precludes re-entrancy.

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 X IDE** window and indicates such current information as cursor position, development mode and device, and active tool bar.

Step Into

This command is the same as Single Step. Step Into (as opposed to Step Over) follows a CALL instruction into a subroutine.

Step Over

Step Over allows you to debug code without stepping into subroutines. When stepping over a `CALL` instruction, the next breakpoint will be set at the instruction after the `CALL`. If for some reason the subroutine gets into an endless loop or does not return properly, the next breakpoint will never be reached. The Step Over command is the same as Single Step except for its handling of `CALL` instructions.

Step Out

Step Out allows you to step out of a subroutine which you are currently stepping through. This command executes the rest of the code in the subroutine and then stops execution at the return address to the subroutine.

Stimulus

Input to the simulator, i.e., data generated to exercise the response of simulation to external signals. Often the data is put into the form of a list of actions in a text file. Stimulus may be asynchronous, synchronous (pin), clocked and register.

Stopwatch

A counter for measuring execution cycles.

Storage Class

Determines the lifetime of the memory associated with the identified object.

Storage Qualifier

Indicates special properties of the objects being declared (e.g., const).

Symbol

A symbol is a general purpose mechanism for describing the various pieces which comprise a program. These pieces include function names, variable names, section names, file names, struct/enum/union tag names, etc. Symbols in MPLAB X IDE refer mainly to variable names, function names and assembly labels. The value of a symbol after linking is its value in memory.

Symbol, Absolute

Symbols can be made absolute by placing them at a specific address in memory, e.g., `int scanMode __at(0x200);`

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.

Toolbar

A row or column of icons that you can click on to execute functions.

Trace

An emulator or simulator function that logs program execution. The emulator logs program execution into its trace buffer which is uploaded to the 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 computer 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. Currently supported USB versions for Microchip hardware tools are:

USB	Speed Descriptor	Maximum Speed (Megabits/Gigabits per second)
USB 2.0	High Speed	480 Mbps
USB 3.0	SuperSpeed	5 Gbps

Vector

The memory locations that an application will jump to when either a Reset or interrupt occurs.

Volatile

A variable qualifier which prevents the compiler applying optimizations that affect how the variable is accessed in memory.

Warning

MPLAB IDE/MPLAB X IDE: An alert that is provided to warn you of a situation that would cause physical damage to a device, software file, or equipment.

16-bit assembler/compiler: Warnings report conditions that may indicate a problem but do not halt processing.

Watch Variable

A variable that you may monitor during a debugging session in a **Watches** window.

Watches Window

Watches windows contain a list of watch variables that are updated at each breakpoint.

Watchdog Timer (WDT)

A timer on a PIC microcontroller that resets the processor after a selectable length of time. The WDT is enabled or disabled and set up using Configuration bits.

Workbook

For MPLAB X Stimulator, a setup for generation of SCL stimulus.

12. Support

Please refer to the following sections for support issues.

12.1 Warranty Registration

Go to www.microchip.com/mysoftware to register your tool online. If you do not already have a myMicrochip account, you can register for an account at that link. If you already have an account, sign in and click on **Register Hardware Tool**.

Registering your tool online entitles you to receive new product updates. Interim software releases are available at the Microchip website.

12.2 myMicrochip Personalized Notification Service

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

To begin the registration process and select your preferences to receive personalized notifications, go to:

www.microchip.com/pcn

A FAQ and registration details are available on the webpage.

Microchip Information

Trademarks

The “Microchip” name and logo, the “M” logo, and other names, logos, and brands are registered and unregistered trademarks of Microchip Technology Incorporated or its affiliates and/or subsidiaries in the United States and/or other countries (“Microchip Trademarks”). Information regarding Microchip Trademarks can be found at https://www.microchip.com/en-us/about/legal-information/microchip-trademarks.

ISBN: 979-8-3371-0764-6

Legal Notice

This publication and the information herein may be used only with Microchip products, including to design, test, and integrate Microchip products with your application. Use of this information in any other manner violates these terms. Information regarding device applications 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. Contact your local Microchip sales office for additional support or, obtain additional support at www.microchip.com/en-us/support/design-help/client-support-services.

THIS INFORMATION IS PROVIDED BY MICROCHIP “AS IS”. 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 ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE, OR WARRANTIES RELATED TO ITS CONDITION, QUALITY, OR PERFORMANCE.

IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL, OR CONSEQUENTIAL LOSS, DAMAGE, COST, OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE INFORMATION OR ITS USE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROCHIP’S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THE INFORMATION OR ITS USE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THE INFORMATION.

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 unless otherwise stated.

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip products:

- Microchip products meet the specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner, within operating specifications, and under normal conditions.
- Microchip values and aggressively protects its intellectual property rights. Attempts to breach the code protection features of Microchip products are strictly prohibited and may violate the Digital Millennium Copyright Act.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code protection does not mean that we are guaranteeing the product is “unbreakable”. Code protection is constantly evolving. Microchip is committed to continuously improving the code protection features of our products.