MPLAB Snap In-Circuit Debugger User's Guide

Notice to Customers

Important:
All documentation becomes dated, and this manual is no exception. Microchip tools and documentation are constantly evolving to meet customer needs, so some actual dialogs and/or tool descriptions may differ from those in this document. Please refer to our website (www.microchip.com) to obtain the latest documentation available.

Documents are identified with a “DS” number. This number is located on the bottom of each page, in front of the page number. The numbering convention for the DS number is “DSXXXXXXXXN”, where “XXXXX” is the document number and “N” is the revision level of the document.

For the most up-to-date information on development tools, see the MPLAB® X IDE online help. Select the Help menu, and then Topics to open a list of available online help files.

Table of Contents

1. Introduction

The MPLAB® Snap In-Circuit Debugger (PG164100) 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.***

Note: Refer to the MPLAB® X IDE or the MPLAB IPE User's Guides or WebHelp for information on those applications.

1.1 Conventions Used in This Guide

This manual uses the following documentation conventions:

Table 1-1. Documentation Conventions

Description	Represents	Examples
Arial font:		
Italic characters	Referenced books	*MPLAB® IDE User's Guide*
	Emphasized text	...is the *only* compiler...
Initial caps	A window	the Output window
	A dialog	the Settings dialog
	A menu selection	select Enable Programmer
Quotes	A field name in a window or dialog	"Save project before build"
Underlined, italic text with right angle bracket	A menu path	<u>*File>Save*</u>
Bold characters	A dialog button	Click **OK**
	A tab	Click the **Power** tab
N'Rnnnn	A number in verilog format, where N is the total number of digits, R is the radix and n is a digit.	4'b0010, 2'hF1
Text in angle brackets < >	A key on the keyboard	Press <Enter>, <F1>
Courier New font:		
Plain Courier New	Sample source code	`#define START`
	Filenames	`autoexec.bat`
	File paths	`c:\mcc18\h`
	Keywords	`_asm, _endasm, static`
	Command-line options	`-Opa+, -Opa-`
	Bit values	`0, 1`
	Constants	`0xFF, 'A'`
Italic Courier New	A variable argument	*`file.o`*, where *`file`* can be any valid filename
Square brackets [ ]	Optional arguments	`mcc18 [options] file [options]`

...........continued

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

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

1.2 Recommended Reading

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

Multi-Tool Design Advisory (DS51764)

Please read this first! This document contains important information about operational issues that should be considered when using the MPLAB Snap 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 (DS50002027) available as a free download at www.microchip.com/mplabx/.

Release Notes for MPLAB Snap 4 In-Circuit Debugger

For the latest information on using MPLAB Snap In-Circuit Debugger, read the notes under "Release Notes and Support Documentation" on the MPLAB X IDE Start Page. The release notes contain update information and known issues that may not be included in this user's guide.

Processor Extension Pak and Header Specification (DS50001292)

This booklet describes how to install and use headers. Headers are used to better debug selected devices, without the loss of pins or resources. See also the PEP and Header online Help file.

2. About the Debugger

An overview of the MPLAB Snap In-Circuit Debugger system is provided here.

2.1 MPLAB Snap In-Circuit Debugger Description

The MPLAB Snap In-Circuit Debugger 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). The debugger works with Microchip PIC®, AVR® and SAM Flash microcontrollers (MCUs), and dsPIC® Digital Signal Controllers (DSCs). It will also work with 32-bit based microcontrollers, such as CEC and PIC32 devices.

The MPLAB Snap connects to the computer using a high-speed 2.0 USB interface and connects to the target via a Microchip debug 8-pin Single In-Line (SIL) connector. The SIL connector uses two device I/O pins and the reset line to implement in-circuit debugging and In-Circuit Serial Programming™ (ICSP™). MPLAB Snap has all the speed and entry-level features you need to quickly debug your prototype.

The MPLAB Snap programs quickly. It 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 Snap supports advanced interfaces such as 4-wire JTAG and Serial Wire Debug. It is also backward compatible for demo boards, headers 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 Snap is compatible with Microsoft Windows® 7 or later, Linux® and macOS™ platforms.

2.2 MPLAB Snap In-Circuit Debugger Advantages

The MPLAB Snap In-Circuit Debugger provides the following advantages.

Features/Capabilities:

- Connects to computer via high-speed USB 2.0 (480 Mbits/s) cable (not included)
- 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 (refer to the device support list found on your PC, for example, C:\Program Files (X86)\Microchip\ MPLABX\vx.xx\docs\Device Support.htm, where vx.xx is the version of MPLAB X IDE)
- Supports 4-wire JTAG, Serial Wire Debug, UPDI, PDI, SPI programming, debugWIRE and TPI programming
- Backward compatibility for demo boards, headers 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%

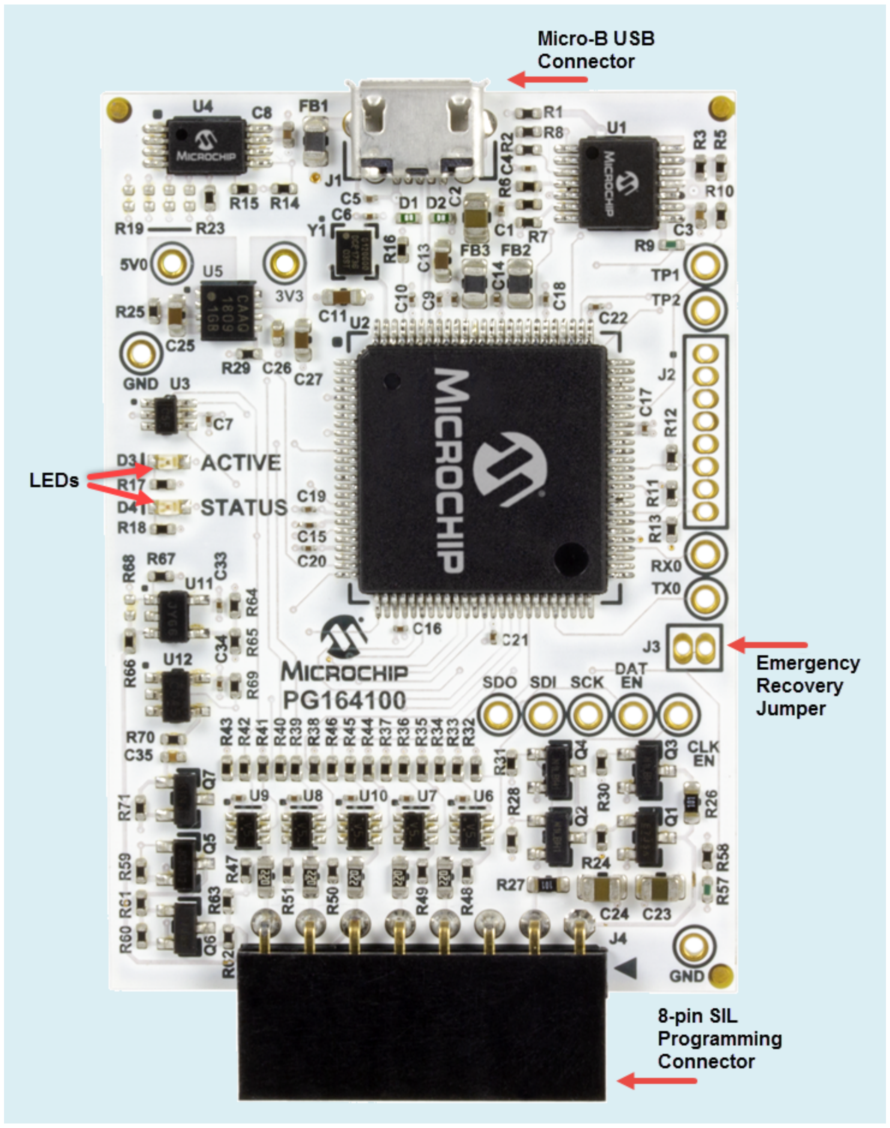
Note: The MPLAB Snap In-Circuit Debugger is powered through its Micro-B USB connector. The target board must be powered from its own power supply.

2.3 MPLAB Snap In-Circuit Debugger Components

The components of the MPLAB Snap In-Circuit Debugger system are:

- an 8-pin SIL connector
- a Micro-B USB connector
- two LEDs
- Emergency Recovery Jumper (not populated)

Figure 2-1. MPLAB® SNAP IN-CIRCUIT DEBUGGER



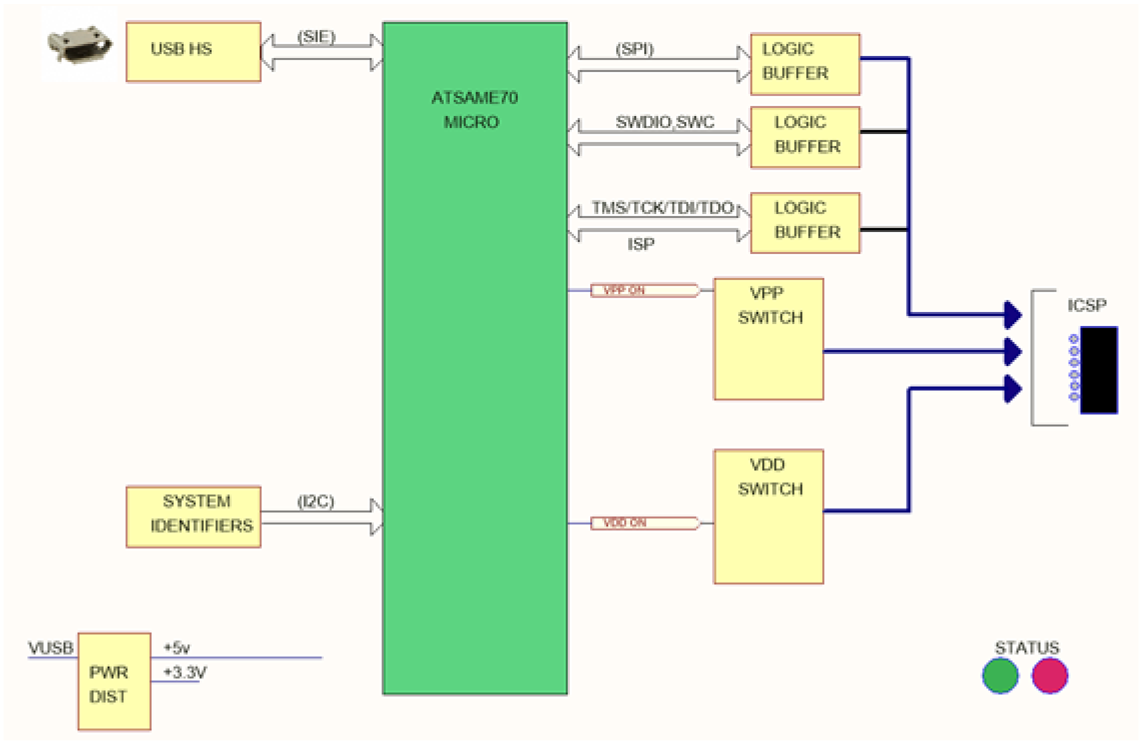
To use the MPLAB Snap In-Circuit Debugger, you will need to supply:

- A full-featured Micro-B USB cable (data and power), no longer than 1.5 meter, to connect to a computer (for example, the Microchip Part Number ATUSBMICROCABLE-XPRO).
- Target board.
- Power supply for target board.
- Any wiring interfaces or cables needed for your application, some available adapters and cables include:
  - AC164110 RJ-11 to ICSP Adapter.
  - AC002021 PM3 ICSP cable.
- Jumper, wire or tweezers for emergency recovery, if needed.

Additional hardware and accessories may be ordered separately from Microchip Direct (www.microshipdirect.com).

- Debugger Adapter Board (Part Number AC002015) - a connectivity board that supports JTAG, SWD and ICSP protocols, useful for debugging AVR® with MPLAB Snap (www.microchipdirect.com/product/search/all/AC102015).
- Transition sockets.
- ICD headers.
- MPLAB porcessor extension paks.

2.4 MPLAB Snap Block Diagram



3. Operation

A simplified theory of operation of the MPLAB Snap 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.

3.1 Debugger to Target Communication

Important: The MPLAB X IDE software must be installed prior to connecting the MPLAB Snap In-Circuit Debugger.

The debugger is connected to the computer via a USB cable for communication and debugger power.

The debugger is connected to the target application for communication and data collection and optional debugger power.

The debugger system configurations are discussed in the following sections.

CAUTION

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

CAUTION

Debugger or Target Damage.
Do not change hardware connections while the debugger or target is powered.

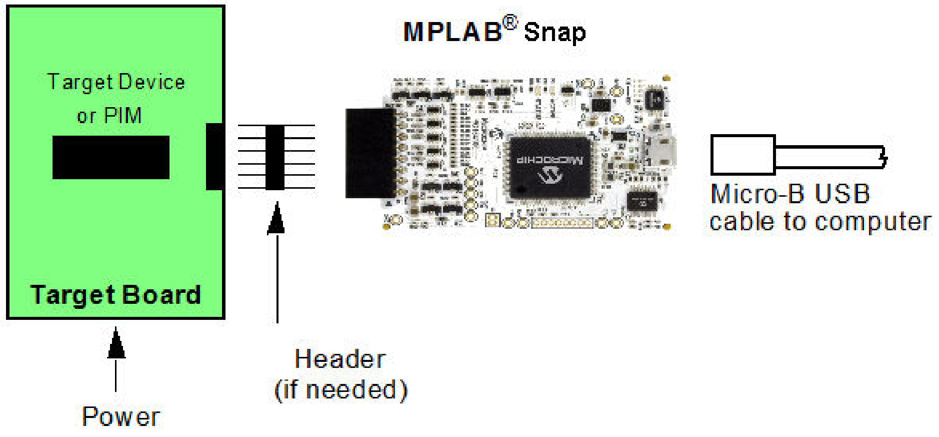
3.1.1 Standard ICSP™ Device Communication

The debugger system can be configured to use standard ICSP communication connection for both programming and debugging functions.

Make sure to align the Pin 1 on the debugger to Pin 1 on the target. The programming connector can be inserted into either:

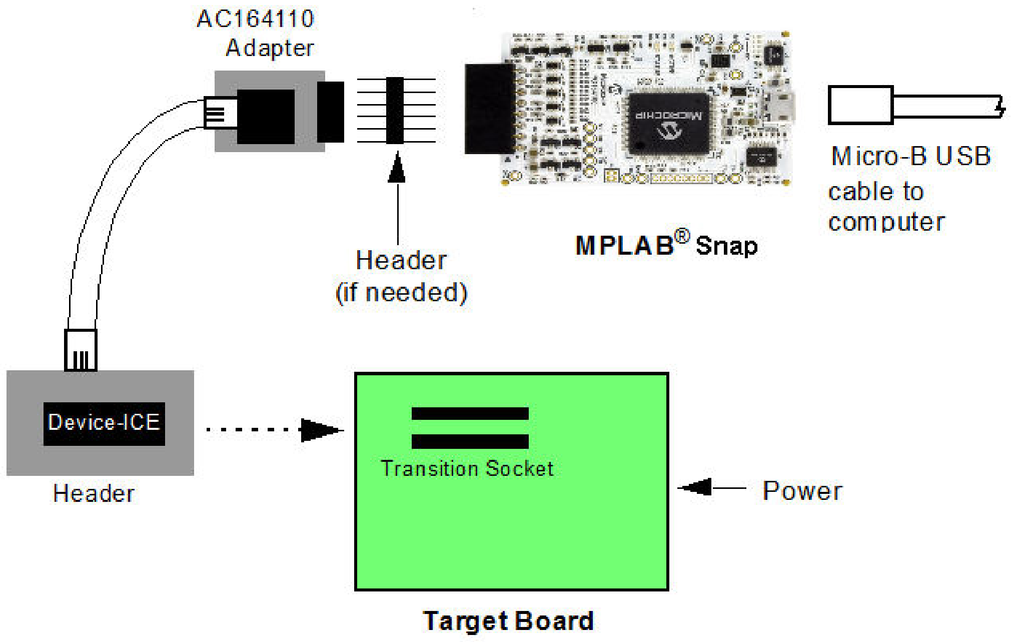
- A matching connector at the target, where the target device is on the target board.

Figure 3-1. Standard Debugger System – Device With On-board ICE Circuitry



- A standard adapter/header board combo (available as a Processor Extension Pak), which is then plugged into the target board.

Figure 3-2. Standard Debugger System – ICE Device



For more on standard communication, see 10.3.1 Standard Communication.

3.2 Target Communication Connections

Important: Refer to the data sheet for the device you are using as well as the application notes and the specific interface for additional information and diagrams.

3.2.1 Standard Communication Target Connection

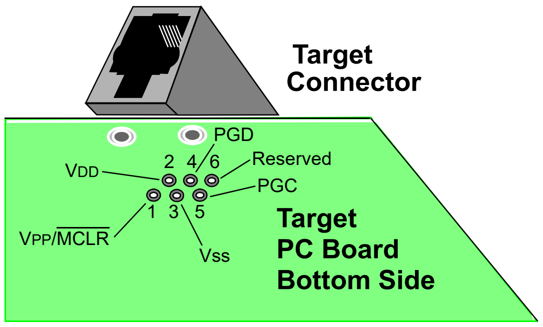
USING SINGLE IN-LINE CONNECTOR

Use the single in-line connector between the MPLAB Snap In-Circuit Debugger and the target board connector (see Figure 3-1 and Standard Debugger System - Device With On-board ICE Circuitry).

USING AN ADAPTER

Use the AC164110 adapter between the MPLAB Snap In-Circuit Debugger and the target device with the modular interface (six conductor) cable. The pin numbering for the connector is shown from the bottom of the target PCB in the following figure.

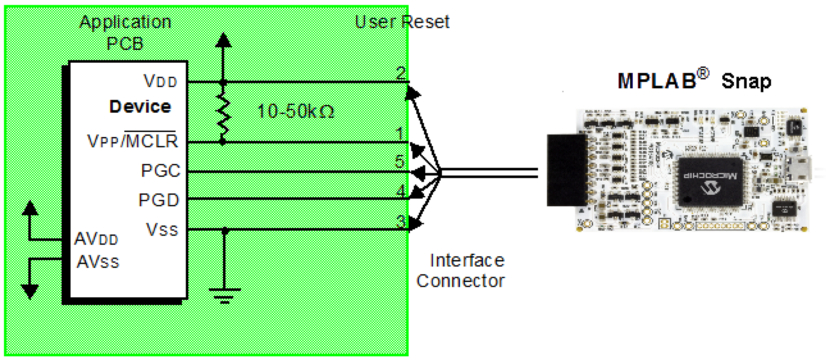
Figure 3-3. Standard RJ-11 Connection at Target



3.2.2 Target Connection Circuitry

The figure below shows the interconnections of the MPLAB Snap In-Circuit Debugger to the connector on the target board. The diagram also shows the wiring from the connector to a device on the target PCB. A pull-up resistor (usually around 10-50 kΩ) 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 3-4. Standard Connection to Target Circuitry

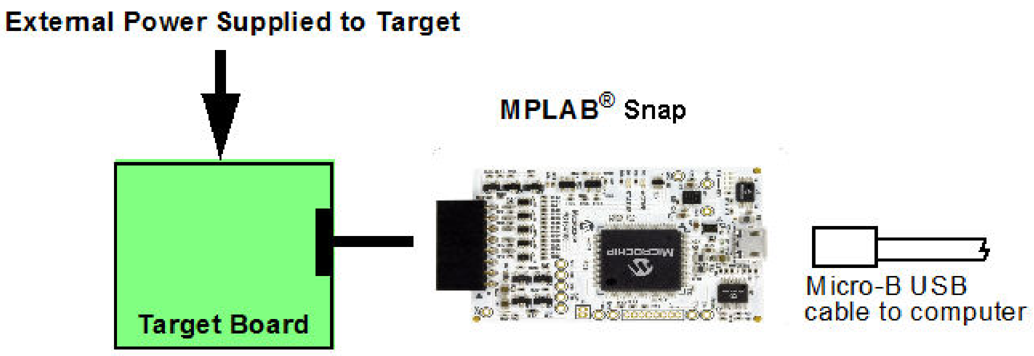


3.2.3 Target Powered

In the following descriptions, only three lines are active and relevant to core debugger operation: pins 1 ($V_{PP}/\overline{MCLR}$), 5 (PGC), and 4 (PGD). Pins 2 (V_{DD}) and 3 (V_{SS}) are shown in Figure 3-4 for completeness.

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

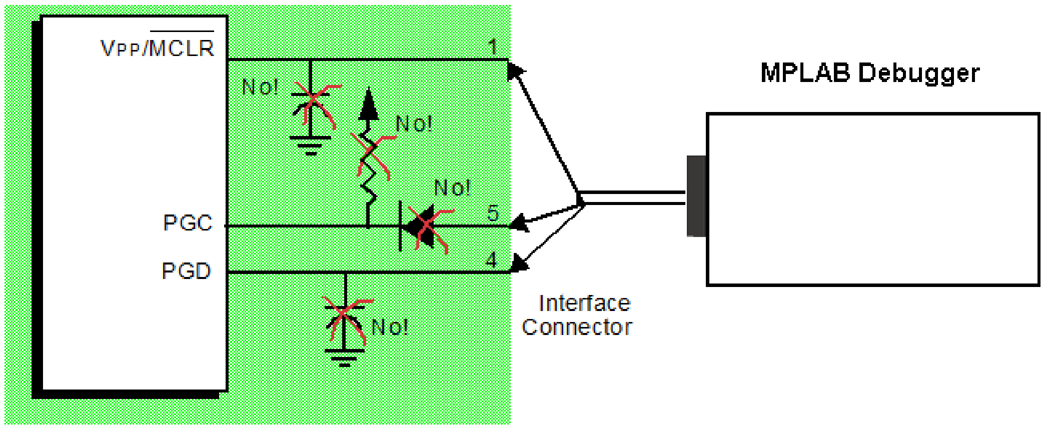
Figure 3-5. Target Powered from External Source



3.2.4 Circuits That Will Prevent the Debugger From Functioning

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

Figure 3-6. Improper Circuit Components



Specifically, these guidelines must be followed:

- Do not use pull-ups on PGC/PGD – they will disrupt the voltage levels, since these lines have programmable pull-down resistors in the debugger.
- Do not use capacitors on PGC/PGD – they will prevent fast transitions on data and clock lines during programming and debugging communications and slow programming times.
- Do not use capacitors on $\overline{MCLR}$ – they will prevent fast transitions of V_{PP}. 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.

3.3 Debugging

There are two steps to using the MPLAB Snap In-Circuit Debugger system as a debugger. The first requires that an application is programmed into the target device (usually with the MPLAB Snap itself). The second uses the internal in-circuit debug hardware of the target Flash device to run and test the application program. These two steps are directly related to the MPLAB X IDE operations:

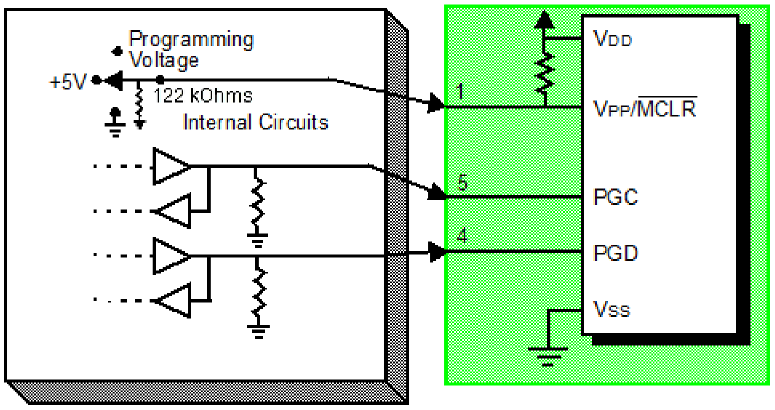
1. Programming the code into the target and activating special debug functions.
2. Debugging the code using features such as breakpoints.

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

If the target device cannot be programmed correctly, the MPLAB Snap In-Circuit Debugger will not be able to debug.

A simplified diagram of some of the internal interface circuitry of the MPLAB Snap In-Circuit Debugger is shown in the figure below.

Figure 3-7. Proper Connections for Programming



No clock is needed for programming on the target device, but power must be supplied. When programming, the debugger puts programming levels on $V_{PP}/\overline{MCLR}$, 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.

3.4 Requirements for Debugging

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

- The debugger must be connected to a computer. It must be powered by the computer via the USB cable and it must be communicating with the MPLAB X IDE software via the Micro-B USB cable. Refer to the MPLAB X IDE Help file titled “Getting Started with MPLAB X IDE,” and navigate through the “Tutorial” to the “Running and Debugging Code” section.
- The debugger must be connected (as shown in the figure in 3.3 Debugging) to the V_{PP}, PGC and PGD pins of the target device with the modular interface cable (or equivalent).
- The target device must have power and a functional, running oscillator. If for any reason, the target device does not run, the MPLAB Snap In-Circuit Debugger will not be able to debug.
- The target device must have its Configuration words programmed correctly. These are set using the MPLAB X IDE.
  - The oscillator Configuration bits should correspond to RC, XT, etc., depending on the target design.
  - For some devices, the Watchdog Timer is enabled by default and needs to be disabled.
  - The target device must not have code protection enabled.
  - The target device must not have table read protection enabled.

– 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.

When the conditions listed above are met, you may proceed to the following sections.

3.4.1 Sequence of Operations Leading to Debugging

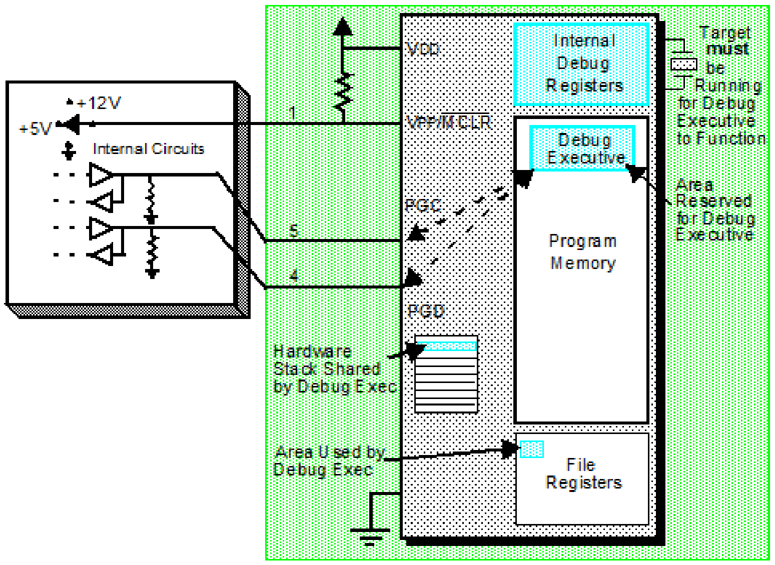
Given that the 3.4 Requirements for Debugging are met, set the MPLAB Snap In-Circuit Debugger as the current tool in MPLAB X IDE. Go to *File > Project Properties* to open the dialog, then under "Hardware Tool," click **Snap**. The following actions can now be performed.

- When *Debug > Debug Main 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 high area of program memory of the target device. Since the debug executive must reside in program memory, the application program must not use this reserved space. Some devices have special memory areas dedicated to the debug executive. Check your device data sheet for details.
- Special "in-circuit debug" registers in the target device are enabled by MPLAB X IDE. These allow the debug executive to be activated by the debugger. For more information on the device's reserved resources, see 3.6 Resources Used by the Debugger.
- The target device is run in Debug mode.

3.4.2 Debugging Details

The figure below illustrates the MPLAB Snap In-Circuit Debugger system when it is ready to begin debugging.

Figure 3-8. MPLAB® Snap In-Circuit Debugger Ready to Begin 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 > Debug Main Project* function 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 (much like an interrupt) and the user's application is effectively halted. The debugger communicates with the debug executive via PGC and PGD, gets the breakpoint status information and sends it back to MPLAB X IDE. MPLAB X IDE then sends a series of queries to the debugger to get information about the target device, such as the 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 Snap In-Circuit Debugger, and MPLAB X IDE will issue an error message.

Another way to set a breakpoint is to select *Debug > Pause*. 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.

3.5 Programming

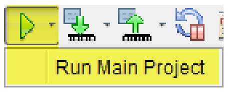
Note: For information on programming, refer to the MPLAB X IDE WebHelp.

Notice: Headers for high voltage devices are not supported by MPLAB Snap.

In MPLAB X IDE, use the MPLAB Snap as a programmer to program a non-ICE/-ICD device, such as a device not on a header board. Set the MPLAB Snap In-Circuit Debugger as the current tool (click the Debug Tool Snap in the navigation window, select *File > Project Properties*, then under "Hardware Tool," click **Snap**) to perform these actions:

- When Run Main Project icon (see below) is selected, 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.

  Figure 3-9. Run Main Project Icon



- 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 $\overline{\text{MCLR}}$ line to Reset and start the target device.

The MPLAB Snap In-Circuit Debugger system programs the target using ICSP. V_{PP}, PGC and PGD lines should be connected as described previously. No clock is required while programming and all modes of the processor can be programmed, including code protection, Watchdog Timer and table read protection.

3.6 Resources Used by the Debugger

For a complete list of resources used by the debugger for your device, see the online Help file in MPLAB X IDE for the MPLAB Snap In-Circuit Debugger. From the MPLAB X IDE "Learn & Discover" page, click **Users Guide & Release Notes** and then click the link for the "Reserved Resources for MPLAB Snap."

4. Debugger Usage

How to install and setup the MPLAB Snap In-Circuit Debugger system is discussed in this section. For instructions on using MPLAB X IDE with the debugger, refer to the online Help accessible from the MPLAB X IDE main menu bar *Help > Tool Help Contents > MPLAB X IDE Help*.

4.1 Installation and Setup

In MPLAB X IDE, refer to the online Help file "Getting Started with MPLAB X IDE" for details on installing the IDE and setting up the debugger to work with it.

In summary:

1. Install MPLAB X IDE.

Tip: Tutorial topics are available in the MPLAB X IDE online Help that is accessible from the main menu bar *Help > Tool Help Contents > MPLAB X IDE Help >Tutorial*.

2. Connect the MPLAB Snap to the computer and allow the default USB drivers to install. For more information on target connections, see Operation.

Important: The debugger cannot power a target board.

3. Select which language toolsuite/compiler you want to use for development and install it on your computer.
4. Launch MPLAB X IDE and open the online Help (*Help > Tool Help Contents > MPLAB X IDE Help*) for detailed instructions on creating and setting up a new project and running and debugging code.

Items of note:

1. Each debugger contains a unique identifier which, when first installed, will be recognized by the operating system, regardless of the computer USB port used.
2. MPLAB X IDE operation connects to the hardware tool at run time (Run or Debug Run). To always be connected to the hardware tool, go to *Tools > Options*, **Embedded** button, **Generic Settings** tab, and check the "Maintain active connection to hardware tool" check box.
3. Configuration bits can only be viewed in the Configuration Bits window. To set them in code, select *Window > Target Memory Views*. Then select "Configuration Bits" form the Memory drop list and select "Read/Write" from the Format drop list to enable access to the settings.

4.2 Debug Tutorial

Refer to the MPLAB X IDE Help file titled "Getting Started with MPLAB X IDE," and navigate through the "Tutorial" to the "Running and Debugging Code" section.

4.3 Quick Debug/Program Reference

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

Notice: For header support, see the Release Notes for MPLAB Snap in MPLAB X IDE v5.25 or greater.

Table 4-1. Debug vs. Program Operation

Item	Debug	Program
Needed Hardware	A computer and target application (Microchip demo board or your own design).	
	Debugger, USB cable, and power supply (if needed).	
	Device with on-board debug circuitry or debug header with special -ICE device.	Device (with or without on-board debug circuitry).
MPLAB X IDE selection	Project Properties, Snap 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 is placed in program memory and other debug resources are 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`.

4.4 Debugger Limitations

For a complete list of debugger limitations for your device, see the online Help file in MPLAB X IDE (*Help > Tool Help Contents > Hardware Tool Reference Help > Limitations - Emulators and Debuggers*).

4.5 Common Debug Features

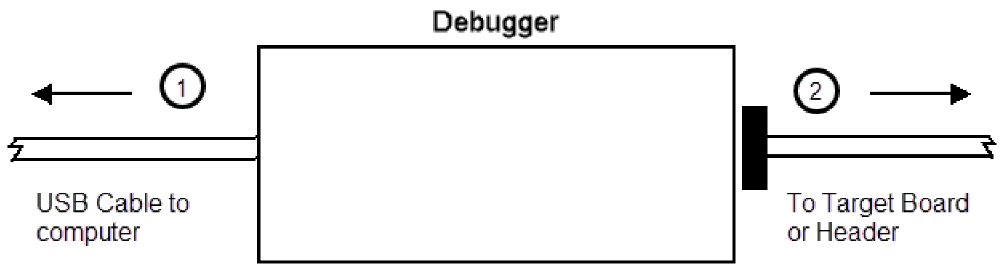
Refer to the online Help file “Getting Started with MPLAB X IDE,” Running an Debugging Code section, for details on debug features. This sections includes:

1. Debug Running the project (build, program and run) from *Debug > Debug Main Project*.
2. Using breakpoints.
3. Stepping through code.
4. Using the Watches window.
5. Viewing Memory, Variables and the Call Stack.
6. Using the Call Graph.

4.6 Connecting the Target Board

1. Connect the Micro-B USB cable between the debugger and the computer, if not already connected.
2. Connect the appropriate cable(s) between the debugger to the target.
3. Connect power to target if needed.

Figure 4-1. Insert Communications and USB Cables



See Debugger to Target Communication for more details and a diagram.

4.7 Setting Up the Target Board

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

4.8 Setting Up MPLAB X IDE

Once the hardware is connected and powered, MPLAB X IDE may be set up for use with the MPLAB Snap In-Circuit Debugger.

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

Refer to the MPLAB X IDE Help for details on installing the software and setting up the debugger to work with it.

4.9 Starting and Stopping Debugging

Note: Refer to the MPLAB X IDE WebHelp for information on menu option icons.

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

- To run your code, select either *Debug > Debug Main Project* or from the Run toolbar.
- To halt your code, select either *Debug > Pause* or from the Debug toolbar.
- To run your code again, select either *Debug > Continue* or from the Debug toolbar.
- To step through your code, select either *Debug > Step Into* or from the Debug toolbar. Be careful not to step into a Sleep instruction or you will have to perform a processor Reset to resume debugging.
- To step over a line of code, select either *Debug > Step Over* or from the Debug toolbar.
- To end code execution, select either *Debug > Finish Debugger Session* or from the Debug toolbar.

- To perform a processor Reset on your code, select either *Debug > Reset* or from the Debug toolbar.

Depending on the device, additional Resets, such as POR/BOR, $\overline{\text{MCLR}}$ and System, may be available. Refer to the product data sheet for more information.

4.10 Viewing Processor Memory and Files

MPLAB X IDE provides several windows for viewing debug and memory information. These are selectable from the Window menu. See the MPLAB X IDE Online Help for more information on using these windows.

- *Window > Target Memory Views* - view data (Data Memory) and code (Execution Memory) in device memory. Other memory can also be viewed as defined by the device including Peripherals, Configuration Bits, CPU Registers, External EBI Memory, External SQI Memory, User ID Memory, etc.
- *Window > Debugging* - view debug information. Select from Variables, Watches, Call Stack, Breakpoints, Stopwatch and many others.

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

4.11 Breakpoint and Stopwatch Usage

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

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

5. Troubleshooting

If you are having problems with MPLAB Snap In-Circuit Debugger operation, start here.

If you are having problems with programming and debugging with AVR microcontroller devices that use the UPDI/PDI/TPI interfaces, the MPLAB Snap may require an external pull-up resistor. Refer to the MPLAB Snap web page (www.microchip.com/developmenttools/ProductDetails/PartNO/PG164100) for the "ETN36_MPLAB Snap AVR Interface Modification PDF."

The Hardware Tool Emergency Boot Firmware Recovery Utility

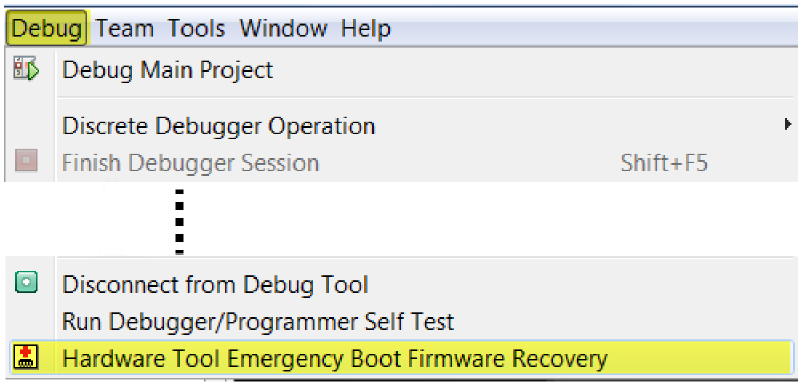
⚠WARNING **Only use this utility to restore hardware tool boot firmware to its factory state. Use only if your hardware tool no longer functions on any machine.**

The debugger may need to be forced into recovery boot mode (reprogrammed) in rare situations. For example, if none of the LEDs are lit when the debugger is connected to the computer.

YOU MUST USE MPLAB X IDE V5.05 OR GREATER TO USED THE EMERGENCY RECOVERY UTILITY.

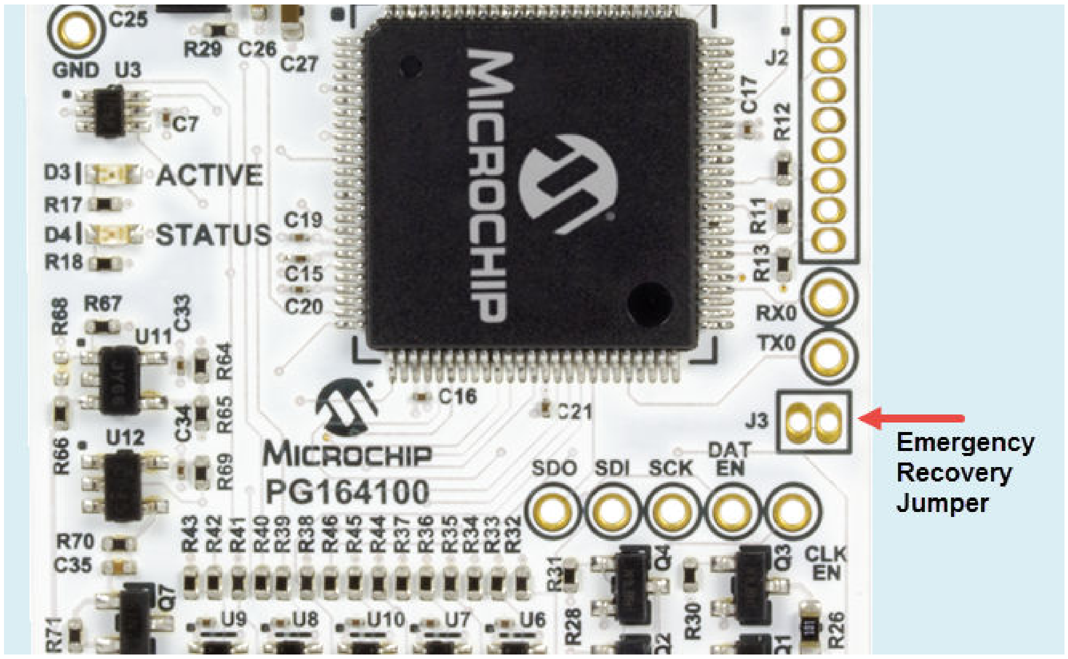
Carefully follow the instructions found in MPLAB X IDE under the main menu options *Debug > Hardware Tool Emergency Boot Firmware Recovery*.

Figure 5-1. Selecting the 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 following figure shows where the emergency recovery jumper is located on the board.

Figure 5-2. Emergency Recovery Jumper



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

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

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 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 3. Operation.
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 communications 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 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. **No device debug circuitry**. The production device may not have debugging capabilities. Use a debug header instead. (See the *"Processor Extension Pak and Debug Header Specification"* (DS50001292) in 1.2 Recommended Reading.)
9. **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.
10. **Incorrect programming pins**. The PGC/PGD pin pairs are not correctly programmed in your Configuration bits (for devices with multiple PGC/PGD pin pairs).
11. **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.
12. **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.
13. **Incorrect connections**. Review the guidelines in 3. Operation for the correct communication connections.
14. **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 Other Things to Consider

5.3.1 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® Snap In-Circuit Debugger functionality.
4. Review debugger setup to ensure proper application setup. For more information, see 3. Operation.
5. Your program speed may be set too high for your circuit. In MPLAB X IDE, go to *File > Project Properties*, select **Snap** in *Categories*, then *Program Options*, *Program Speed* and select a slower speed from the drop-down menu. The default is Normal. In MPLAB IPE, the **Program Speed** option can be found in the Advanced Mode, Settings tab.
6. There may be certain situations where the debugger is not operating properly, a newer tool pack may need to be used, or the debugger needs to be reprogrammed. See the following sections to determine additional actions.

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

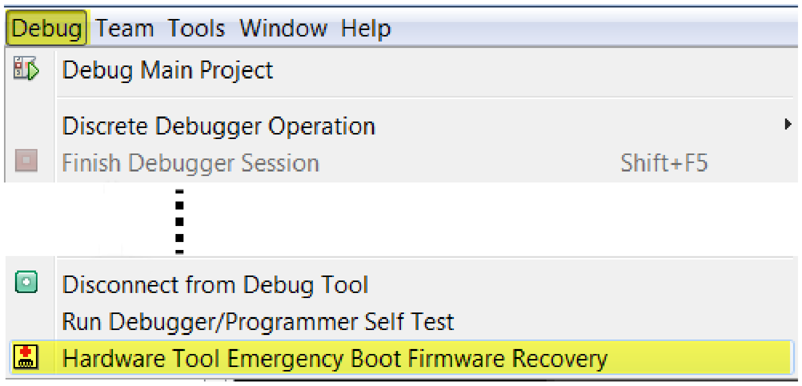
⚠WARNING **Only use this utility to restore hardware tool boot firmware to its factory state. Use only if your hardware tool no longer functions on any machine.**

The debugger may need to be forced into recovery boot mode (reprogrammed) in rare situations; for example, if the debugger has no LED lit.

YOU MUST USE MPLAB X IDE V5.05 OR GREATER TO USE THE EMERGENCY RECOVERY UTILITY.

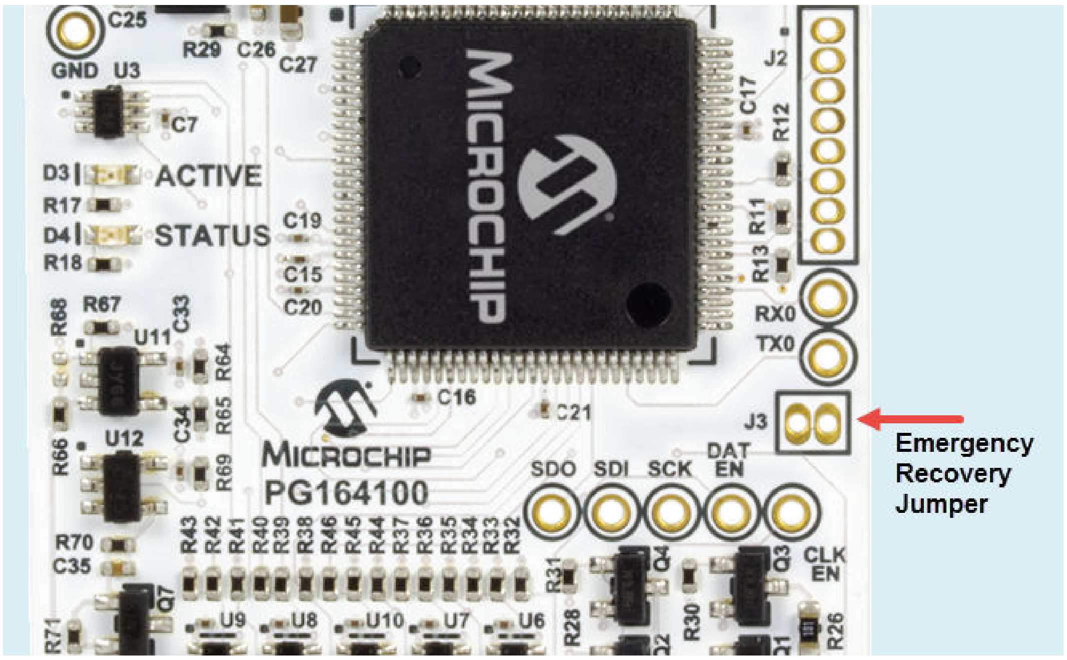
Carefully follow the instructions found in MPLAB X IDE under the main menu options *Debug > Hardware Tool Emergency Boot Firmware Recovery*.

Figure 5-3. 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 jumpers are located on the MPLAB Snap board.

Figure 5-4. Emergency Recovery Jumpers



If the procedure was successful, the recovery wizard displays a success screen. The MPLAB Snap 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 Snap In-Circuit Debugger system.

6.1 How Does it Work?

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

MPLAB Snap In-Circuit Debugger can communicate with Flash silicon via the ICSP™ interface. It uses the debug executive downloaded into program or test memory.

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

The debug executive doesn't run while in Run mode, so there is no throughput reduction when running your code, that is, the debugger doesn't 'steal' any cycles from the target device.

Does the MPLAB Snap 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 Snap 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 Snap In-Circuit Debugger optoisolated or electrically isolated?

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

Will the MPLAB Snap 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 Snap 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 Snap 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 7.2.4 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 and try to Run. If you receive this message after this, or immediately after programming your device, please refer to 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 Snap is busy. Please wait for the current operation to finish.

If you receive this error when attamepting 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.

7.2 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 (see 9.2.2 Debug).
6. Ensure that the cable(s) are of the correct length.

7.2.2 Debugger to Target Communication Error Actions

If the MPLAB Snap 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 Snap 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 theMPLAB Snap 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 Snap 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 (see 9.2 Debugger Options Selection).
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 Snap In-Circuit Debugger was unable to perform a debugging operation. There are numerous reasons why this might occur. See 5.2 Top Reasons Why You Can't Debug and 5.3 Other Things to Consider.

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. Engineering Technical Notes (ETNs)

The following ETNs are related to the MPLAB® Snap In-Circuit Debugger. Please see the product web page for details.

ETN36: MPLAB Snap AVR Interface Modification can be found on the MPLAB Snap product web page at www.microchip.com/developmenttools/ProductDetails/PartNO/PG164100.

9. Debugger Function Summary

A summary of the debugger functions are summarized below.

9.1 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:

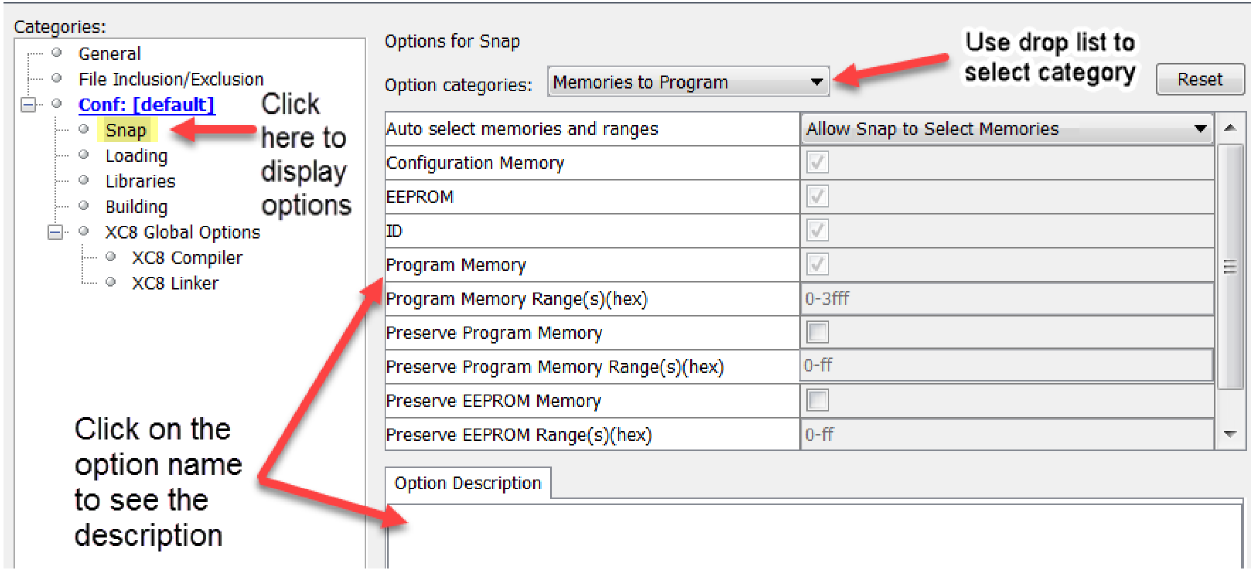
1. Open the Project Properties dialog by doing one of the following:
   1.1. Click on the project name in the Projects window and select *File > Project Properties*.
   or
   1.2. Right click on the project name in the Projects window and select **Properties**.
2. Under **Categories** on the left side, expand "Conf:[default]" to show your debugger, for example PICkit 4 or Snap, etc.
3. Under **Hardware Tools**, find your debugger and click on a serial number (SN) to select a debugger for use in the project, then click **Apply**.

9.2 Debugger Options Selection

Debugger options are set in the Project Properties dialog of MPLAB X IDE. Click on Snap under "Categories" to display the "Options for Snap" (see Figure 1-3). 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 9-1. MPLAB X IDE Options for MPLAB Snap



Options for Snap

Option categories: Memories to Program

Auto select memories and ranges	Allow Snap 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

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

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

The possible option categories may include:

- Memories to Program
- Debug Options
- Program Options
- Freeze Peripherals
- Power
- Tool pack selection
- Firmware

9.2.1 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 9.2.3 Program then all device memory will be erased before programming.

Table 9-1. Memories to Program Option Category

Auto select memories and ranges	**Allow Snapto 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 areas(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.	

9.2.2 Debug

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

Table 9-2. Debug Option Category

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

Table 9-3. 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.		

9.2.3 Program

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

Table 9-4. 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.
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.

9.2.4 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**.

9.2.5 Secure Segment

Select and load debugger firmware.

Table 9-5. 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.

9.2.6 Tool Pack Selection

Select and load debugger firmware.

Table 9-6. 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.

9.2.7 Clock

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

Table 9-7. 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.

9.2.8 Communication

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

Table 9-8. Communication Option Category

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

GPIO vs. UPDI Operation:

When using a high voltage pulse to reactivate the UPDI interface, the reactivation is only temporary, but it will retain the UPDI functionality until the next reset. After the next reset, the pin will go back to the configuration as specified by the RSTPINCFG configuration bits. To have the pin configured as UPDI after a reset, the user will have to change the RSTPINCFG configuration bits back to UPDI.

It is possible to perform a debug session when the RSTPINCFG is configured to GPIO, but the pin will be temporarily configured as UPDI, and the pin will not operate as a GPIO pin.

Table 9-9. SYSCFG0 RSTPINCFG[1:0] Configuration Bits

Values	Function
0x0	GPIO
0x1	UPDI
0x2	RESET
0x3	Reserved

10. Hardware Specification

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

10.1 USB Connector

The MPLAB Snap In-Circuit Debugger is connected to the host computer via a Micro-B USB connector, version 2.0 compliant. The Micro-B USB 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 Snap In-Circuit Debugger is powered through its Micro-B USB connector. The target board is powered from its own supply. The MPLAB Snap cannot power the target board.

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

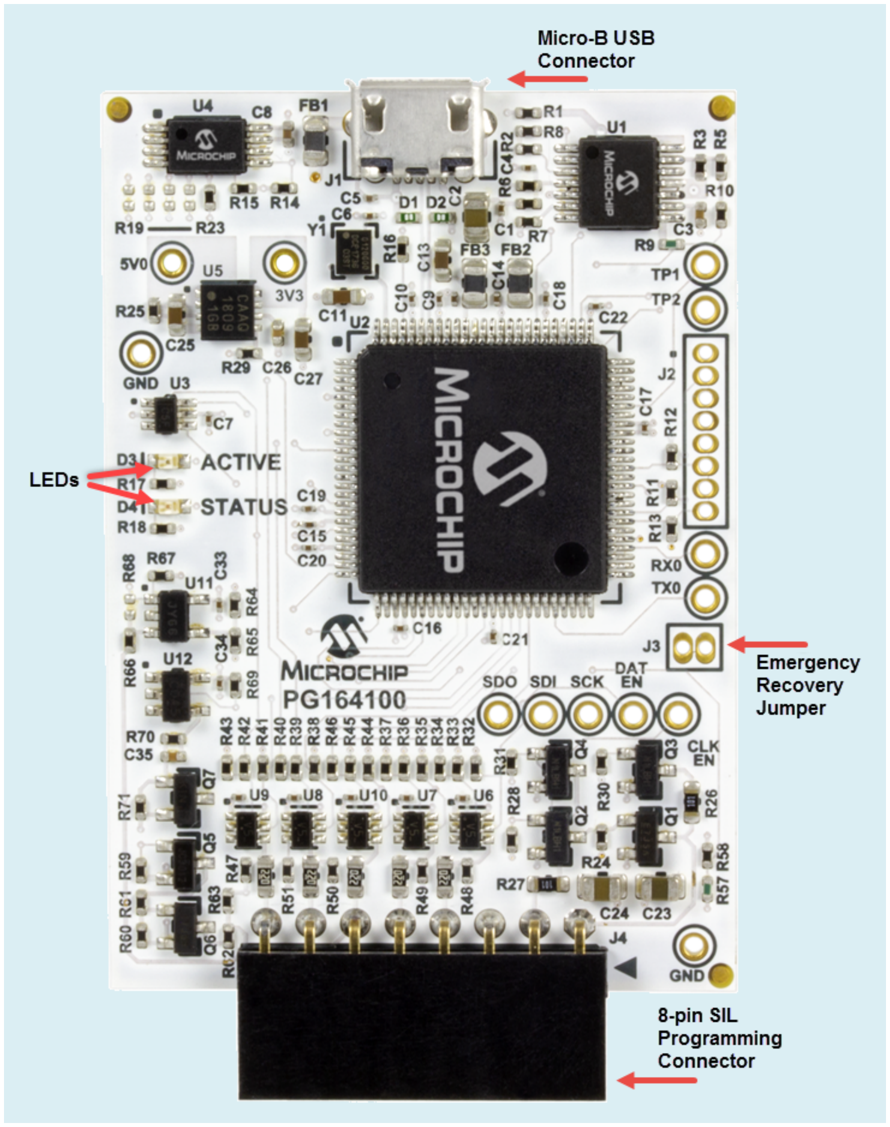
Powered Hubs – If you are going to use a USB hub, make sure it is self-powered. Also, USB ports on computer keyboards do not have enough power for the debugger to operate.

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

10.2 MPLAB Snap In-Circuit Debugger

The debugger consists of an internal main board and an external Micro-B USB connector and an 8-pin SIL connector. On the front of the debugger board has two LEDs and a jumper.

Figure 10-1. MPLAB® SNAP IN-CIRCUIT DEBUGGER



1. Micro-B USB Connector - Used to connect the debugger to the computer with the supplied USB cable.
2. LEDs - Displays the operational modes of the debugger).
3. Pin 1 Marker - This designates the pin 1 location for proper connector alignment.
4. Programming Connector - The connector is an 8-pin SIL header (0.100" spacing) that connects to the target device (see 10.3.2 Pinouts for Interfaces).
5. Emergency Recovery Jumper - For use with the emergency boot firmware recovery utility only.

10.2.1 LEDs

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

Table 10-1. 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 10-2. Error LED Descriptions

Errors	Description
Status, on 3 seconds	Bootloader problem accessing the serial EEPROM.
Status, on 10 seconds	API commands connot be processed by the Bootloader.
Active and Status, fast blink (alternating)	A runtime exception occurred in the tool firmware.
Avtive and Status, fast blink (in tandem)	A runtime exception occurred in the Bootloader.

10.3 Communication Hardware

For standard debugger communication with a target (see 3.1 Debugger to Target Communication and 3.1.1 Standard ICSP Device Communication), either connect the debugger directly to the target or use a header if needed. The debugger has an 8-pin SIL connector. If the target has a 6-pin connector, make sure to align the Pin 1 appropriately.

10.3.1 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.

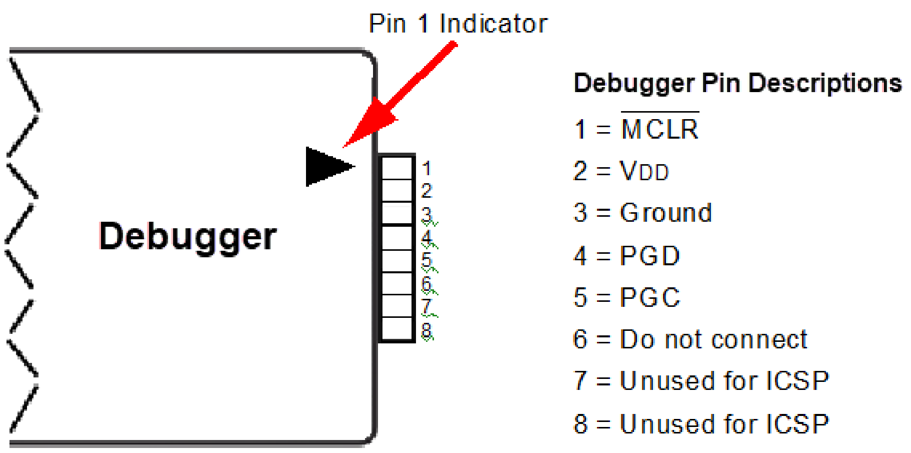
The clock and data connections are interfaces with the following characteristics:

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

Table 10-3. 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.

Figure 10-2. Debugger Connector Pinout



Debugger Pin Descriptions

1 = $\overline{\text{MCLR}}$

2 = VDD

3 = Ground

4 = PGD

5 = PGC

6 = Do not connect

7 = Unused for ICSP

8 = Unused for ICSP

10.3.2 Pinouts for Interfaces

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 10-4. Pinouts for Debug Interfaces

MPLAB Snap			DEBUG								
Connector	Pin #	Pin Name	ICSP (MCHP)	MIPS EJTAG	CORTEX® SWD	AVR® JTAG	AVR ISP(&DW)	UPDI	PDI	debugWIRE	TPI
	1	TVPP	$\overline{\text{MCLR}}$	$\overline{\text{MCLR}}$	$\overline{\text{MCLR}}$						
	2	TVDD	VDD	VIO_REF	VTG	VTG	VTG	VTG	VTG	VTG	VTG
	3	GND	GND	GND	GND	GND	GND	GND	GND	GND	GND
	4	PGD	DAT	TDO	SWO	TDO	MISO	DAT	DAT		DAT
	5	PGC	CLK	TCK	SWCLK	TCK	SCK				CLK
	6	TAUX	AUX			$\overline{\text{RESET}}$	$\overline{\text{RESET}}$		CLK	dW	RST
	7	TTDI		TDI		TDI	MOSI				
	8	TTMS		TMS	SWDIO	TMS					

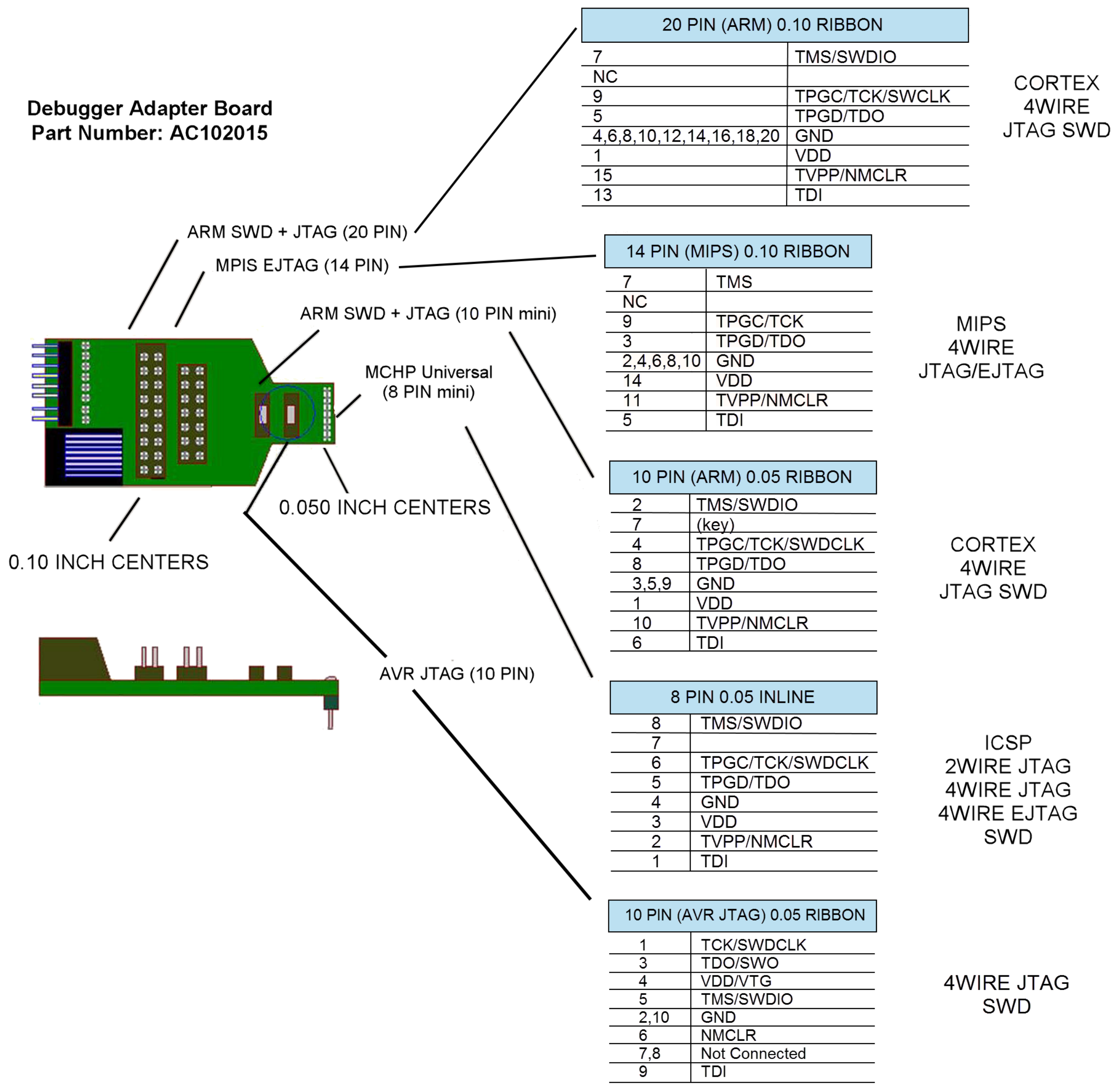
Table 10-5. Pinouts for Data Stream Interfaces

MPLAB Snap	DATA STREAM
Pin #	CDC
1	
2	VTG
3	GND

...........continued	
MPLAB Snap	**DATA STREAM**
Pin #	**CDC**
4	
5	
6	
7	TX (target)
8	RX (target)

Figure 10-3. Debugger Adapter Board (AC102015) Pinouts

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



20 PIN (ARM) 0.10 RIBBON	
7	TMS/SWDIO
NC	
9	TPGC/TCK/SWCLK
5	TPGD/TDO
4,6,8,10,12,14,16,18,20	GND
1	VDD
15	TVPP/NMCLR
13	TDI

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

10 PIN (ARM) 0.05 RIBBON	
2	TMS/SWDIO
7	(key)
4	TPGC/TCK/SWDCLK
8	TPGD/TDO
3,5,9	GND
1	VDD
10	TVPP/NMCLR
6	TDI

8 PIN 0.05 INLINE	
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/SWDCLK
3	TDO/SWO
4	VDD/VTG
5	TMS/SWDIO
2,10	GND
6	NMCLR
7,8	Not Connected
9	TDI

10.4 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:

- 3.2.2 Target Connection Circuitry
- 3.2.4 Circuits That Will Prevent the Debugger From Functioning

11. Revision History

11.1 Revision A (July 2018)

Initial release of this document.

11.2 Revision B (October 2019)

-

11.3 Revision C (July 2020)

- Changed document name from MPLAB Snap In-Circuit Debugger Informaiton Sheet to MPLAB Snap In-Circuit Debugger User's Guide.
- Revised format and renumbered entire document.
- Added extensive information which had been referenced in prior versions.
- Removed firmware project property option and replace it with tool pack selection options.

12. Support

12.1 Warranty Registration

If your development tool package includes a Warranty Registration Card, please complete the card and mail it in promptly. Sending in your Warranty Registration Card entitles you to receive new product updates. Interim software releases are available at the Microchip web site.

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 page, which can be opened by clicking the link above.

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

- **Compilers** – The latest information on Microchip C compilers, assemblers, linkers and other language tools. These include all MPLAB C compilers; all MPLAB assemblers (including MPASM™ assembler); all MPLAB linkers (including MPLINK™ object linker); and all MPLAB librarians (including MPLIB™ object librarian).
- **Emulators** – The latest information on the MPLAB REAL ICE™ emulator.
- **In-Circuit Debuggers** – The latest information on Microchip in-circuit debuggers. These include the MPLAB ICD 3 and MPLAB ICD 4 in-circuit debuggers and PICkit™ 3 and MPLAB PICkit 4 in-circuit debuggers.
- **MPLAB® X IDE** – The latest information on Microchip MPLAB X IDE, the multi-platform (Windows®, Mac OS®, Linux®) Integrated Development Environment for development systems tools.
- **Programmers** – The latest information on Microchip programmers. These include the device (production) programmers MPLAB REAL ICE in-circuit emulator, MPLAB ICD 4 in-circuit debugger, MPLAB PICkit 4 in-circuit debugger, MPLAB PM3 and development (non-production) programmers PICkit 3.
- **Starter/Demo Boards** – These include MPLAB Starter Kit boards, PICDEM demo boards, and various other evaluation boards.

13. 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.

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.

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.

Data Monitor and Control Interface (DMCI)
The Data Monitor and Control Interface, or DMCI, is a tool in MPLAB X IDE. The interface provides dynamic input control of application variables in projects. Application-generated data can be viewed graphically using any of 4 dynamically-assignable graph windows.

Debug/Debugger
See *ICE/ICD*.

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

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

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

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

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

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

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

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

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

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

Emulation/Emulator
See *ICE/ICD*.

Endianness
The ordering of bytes in a multi-byte object.

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

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

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

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

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

Executable Code
Software that is ready to be loaded for execution.

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

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

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

Extended Mode (PIC18 MCUs)
In Extended mode, the compiler will utilize the extended instructions (i.e., `ADDFSR`, `ADDULNK`, `CALLW`, `MOVSF`, `MOVSS`, `PUSHL`, `SUBFSR`, and `SUBULNK`) and the indexed with literal offset addressing.

External Label
A label that has external linkage.

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

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

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

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

External RAM
Off-chip Read/Write memory.

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 IDE/MPLAB X IDE.

Identifier
A function or variable name.

IEEE
Institute of Electrical and Electronics Engineers.

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

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

```
int myVar=5;
```

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

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

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

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

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

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

Interrupt Handler
A routine that processes special code when an interrupt occurs.

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

Interrupt Service Routine (ISR)
Language tools: A function that handles an interrupt.

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

Interrupt Vector
Address of an interrupt service routine or interrupt handler.

L-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.

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

Loop-Back Test Board
Used to test the functionality of the MPLAB REAL ICE in-circuit emulator.

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

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

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

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

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

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

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

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

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

MCU
Microcontroller Unit. An abbreviation for microcontroller. Also uC.

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

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

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

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

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

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

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

MPASM™ Assembler
Microchip Technology's relocatable macro assembler for PIC microcontroller devices, KeeLoq® devices and Microchip memory devices.

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

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

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

MPLAB PM3
A device programmer from Microchip. Programs PIC18 microcontrollers and dsPIC digital signal controllers. Can be used with MPLAB IDE/MPLAB X IDE or stand-alone. Replaces PRO MATE II.

MPLAB REAL ICE™ In-Circuit Emulator
Microchip's next-generation in-circuit emulator that works with MPLAB IDE/MPLAB X IDE. See ICE/ICD.

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

MPLAB Starter Kit for Device
Microchip's starter kits contains everything needed to begin exploring the specified device. View a working application and then debug and program you own changes.

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

MPLINK™ Object Linker
MPLINK linker is an object linker for the Microchip MPASM assembler and the Microchip C18 C compiler. MPLINK linker also may be used with the Microchip MPLIB librarian. MPLINK linker is designed to be used with MPLAB IDE/ MPLAB X IDE, although it is not required.

MRU
Most Recently Used. Refers to files and windows available to be selected from MPLAB IDE/MPLAB X IDE 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
MPLAB IDE/MPLAB X IDE project component.

Non-Extended Mode (PIC18 MCUs)
In Non-Extended mode, the compiler will not utilize the extended instructions nor the indexed with literal offset addressing.

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

Non-Volatile Storage
A storage device whose contents are preserved when its power is off.

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

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 microcontroller families.

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

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

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

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

Precedence
Rules that define the order of evaluation in expressions.

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

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

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

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

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

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

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

Prologue
A portion of compiler-generated code that is responsible for allocating stack space, preserving registers and performing any other machine-specific requirement specified in the 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 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.

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

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

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

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

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

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

Simulator
A software program that models the operation of devices.

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

Skew
The information associated with the execution of an instruction appears on the processor bus at different times. For example, the executed opcodes 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 IDE/MPLAB X IDE window and indicates such current information as cursor position, development mode and device, and active tool bar.

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

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

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

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

Stopwatch
A counter for measuring execution cycles.

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

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

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

Symbol, Absolute
Description

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.

Term
Represents an immediate value such as a definition through the assembly .equ directive.

Toolbar
A row or column of icons that you can click on to execute MPLAB IDE/MPLAB X IDE functions.

Trace
An emulator or simulator function that logs program execution. The emulator logs program execution into its trace buffer which is uploaded to the MPLAB IDE/MPLAB X IDE trace window.

Trace Memory
Trace memory contained within the emulator. Trace memory is sometimes called the trace buffer.

Trace Macro
A macro that will provide trace information from emulator data. Since this is a software trace, the macro must be added to code, the code must be recompiled or reassembled, and the target device must be programmed with this code before trace will work.

Trigger Output
Trigger output refers to an emulator output signal that can be generated at any address or address range, and is independent of the trace and breakpoint settings. Any number of trigger output points can be set.

Trigraphs
Three-character sequences, all starting with ??, that are defined by ISO C as replacements for single characters.

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. USB 1.0/1.1 supports data transfer rates of 12 Mbps. Also referred to as high-speed USB, USB 2.0 supports data rates up to 480 Mbps.

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

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

Warning
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 SIM stimulator, a setup for generation of SCL stimulus.

The Microchip Website

Microchip provides online support via our website at www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

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

Product Change Notification Service

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

To register, go to www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: www.microchip.com/support

Microchip Devices Code Protection Feature

Legal Notice

Trademarks

ISBN: 978-1-5224-6406-8

Quality Management System

For information regarding Microchip's Quality Management Systems, please visit www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
Corporate Office	**Australia - Sydney**	**India - Bangalore**	**Austria - Wels**
2355 West Chandler Blvd.	Tel: 61-2-9868-6733	Tel: 91-80-3090-4444	Tel: 43-7242-2244-39
Chandler, AZ 85224-6199	**China - Beijing**	**India - New Delhi**	Fax: 43-7242-2244-393
Tel: 480-792-7200	Tel: 86-10-8569-7000	Tel: 91-11-4160-8631	**Denmark - Copenhagen**
Fax: 480-792-7277	**China - Chengdu**	**India - Pune**	Tel: 45-4485-5910
Technical Support:	Tel: 86-28-8665-5511	Tel: 91-20-4121-0141	Fax: 45-4485-2829
www.microchip.com/support	**China - Chongqing**	**Japan - Osaka**	**Finland - Espoo**
Web Address:	Tel: 86-23-8980-9588	Tel: 81-6-6152-7160	Tel: 358-9-4520-820
www.microchip.com	**China - Dongguan**	**Japan - Tokyo**	**France - Paris**
Atlanta	Tel: 86-769-8702-9880	Tel: 81-3-6880- 3770	Tel: 33-1-69-53-63-20
Duluth, GA	**China - Guangzhou**	**Korea - Daegu**	Fax: 33-1-69-30-90-79
Tel: 678-957-9614	Tel: 86-20-8755-8029	Tel: 82-53-744-4301	**Germany - Garching**
Fax: 678-957-1455	**China - Hangzhou**	**Korea - Seoul**	Tel: 49-8931-9700
Austin, TX	Tel: 86-571-8792-8115	Tel: 82-2-554-7200	**Germany - Haan**
Tel: 512-257-3370	**China - Hong Kong SAR**	**Malaysia - Kuala Lumpur**	Tel: 49-2129-3766400
Boston	Tel: 852-2943-5100	Tel: 60-3-7651-7906	**Germany - Heilbronn**
Westborough, MA	**China - Nanjing**	**Malaysia - Penang**	Tel: 49-7131-72400
Tel: 774-760-0087	Tel: 86-25-8473-2460	Tel: 60-4-227-8870	**Germany - Karlsruhe**
Fax: 774-760-0088	**China - Qingdao**	**Philippines - Manila**	Tel: 49-721-625370
Chicago	Tel: 86-532-8502-7355	Tel: 63-2-634-9065	**Germany - Munich**
Itasca, IL	**China - Shanghai**	**Singapore**	Tel: 49-89-627-144-0
Tel: 630-285-0071	Tel: 86-21-3326-8000	Tel: 65-6334-8870	Fax: 49-89-627-144-44
Fax: 630-285-0075	**China - Shenyang**	**Taiwan - Hsin Chu**	**Germany - Rosenheim**
Dallas	Tel: 86-24-2334-2829	Tel: 886-3-577-8366	Tel: 49-8031-354-560
Addison, TX	**China - Shenzhen**	**Taiwan - Kaohsiung**	**Israel - Ra'anana**
Tel: 972-818-7423	Tel: 86-755-8864-2200	Tel: 886-7-213-7830	Tel: 972-9-744-7705
Fax: 972-818-2924	**China - Suzhou**	**Taiwan - Taipei**	**Italy - Milan**
Detroit	Tel: 86-186-6233-1526	Tel: 886-2-2508-8600	Tel: 39-0331-742611
Novi, MI	**China - Wuhan**	**Thailand - Bangkok**	Fax: 39-0331-466781
Tel: 248-848-4000	Tel: 86-27-5980-5300	Tel: 66-2-694-1351	**Italy - Padova**
Houston, TX	**China - Xian**	**Vietnam - Ho Chi Minh**	Tel: 39-049-7625286
Tel: 281-894-5983	Tel: 86-29-8833-7252	Tel: 84-28-5448-2100	**Netherlands - Drunen**
Indianapolis	**China - Xiamen**		Tel: 31-416-690399
Noblesville, IN	Tel: 86-592-2388138		Fax: 31-416-690340
Tel: 317-773-8323	**China - Zhuhai**		**Norway - Trondheim**
Fax: 317-773-5453	Tel: 86-756-3210040		Tel: 47-72884388
Tel: 317-536-2380			**Poland - Warsaw**
Los Angeles			Tel: 48-22-3325737
Mission Viejo, CA			**Romania - Bucharest**
Tel: 949-462-9523			Tel: 40-21-407-87-50
Fax: 949-462-9608			**Spain - Madrid**
Tel: 951-273-7800			Tel: 34-91-708-08-90
Raleigh, NC			Fax: 34-91-708-08-91
Tel: 919-844-7510			**Sweden - Gothenberg**
New York, NY			Tel: 46-31-704-60-40
Tel: 631-435-6000			**Sweden - Stockholm**
San Jose, CA			Tel: 46-8-5090-4654
Tel: 408-735-9110			**UK - Wokingham**
Tel: 408-436-4270			Tel: 44-118-921-5800
Canada - Toronto			Fax: 44-118-921-5820
Tel: 905-695-1980			
Fax: 905-695-2078			