

Notice to Development Tools Customers



Important:

All documentation becomes dated, and Development Tools manuals are no exception. Our tools and documentation are constantly evolving to meet customer needs, so some actual dialogs and/or tool descriptions may differ from those in this document. Please refer to our website (www.microchip.com/) to obtain the latest version of the PDF document.

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

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



Table of Contents

Notice to Development Tools Customers.....	1
1. Preface.....	4
1.1. Conventions Used in This Guide.....	4
1.2. Recommended Reading.....	4
2. About the Debugger.....	6
2.1. Advantages.....	6
2.2. Components.....	7
2.3. Block Diagram.....	8
2.4. Using MPLAB® ICD 5 with MPLAB X IDE and MPLAB IPE.....	9
3. Connections.....	11
3.1. Power and Self Test.....	11
3.2. PC Connections.....	12
3.3. Target Connections.....	16
4. Operation.....	32
4.1. MPLAB X IDE Debugging.....	32
4.2. SAM and PIC32C Arm Devices - On-Chip Debugging.....	32
4.3. AVR Devices - On-Chip Debugging (OCD).....	32
4.4. PIC MCU/dsPIC DSC - On-Chip Debugging.....	41
5. Debugger Features.....	49
5.1. USB CDC Virtual COM Port.....	49
5.2. Data Gateway Interface.....	49
5.3. CI/CD Support.....	51
5.4. ARM ITM/SWO Trace.....	52
5.5. SAM (ARM) - Trace and Profiling.....	58
5.6. Debugger Polling.....	59
5.7. Power Monitor.....	60
6. Troubleshooting First Steps.....	63
6.1. Some Questions to Answer First.....	63
6.2. Top Reasons Why You Can't Debug.....	63
6.3. General Considerations.....	64
6.4. How to Use the Hardware Tool Emergency Boot Firmware Recovery Utility.....	65
7. Frequently Asked Questions (FAQ).....	66
7.1. How Does It Work?.....	66
7.2. What's Wrong?.....	66
8. Error Messages.....	68
8.1. Types of Error Messages.....	68
8.2. General Corrective Actions.....	74
9. Debugger Function Summary.....	76
9.1. Debugger Selection and Switching.....	76

9.2. Debugger Options Selection.....	76
9.3. Debugger Windows & Dialogs.....	83
10. Hardware Specification.....	86
10.1. Debugger Unit.....	86
10.2. Power Specifications.....	86
10.3. Indicator Lights (LEDs).....	86
10.4. PC Connection Specifications.....	87
10.5. 8-pin Communication Hardware.....	88
10.6. Communication Hardware.....	91
10.7. Recovery Specifications.....	94
10.8. Target Board Considerations.....	94
11. Revision History.....	96
11.1. Revision A (May 2023).....	96
11.2. Revision B (November 2023).....	96
11.3. Revision C (July 2024).....	96
12. Support.....	97
12.1. Warranty Registration.....	97
12.2. myMicrochip Personalized Notification Service.....	97
Microchip Information.....	98
The Microchip Website.....	98
Product Change Notification Service.....	98
Customer Support.....	98
Product Identification System.....	99
Microchip Devices Code Protection Feature.....	99
Legal Notice.....	99
Trademarks.....	100
Quality Management System.....	101
Worldwide Sales and Service.....	102

1. Preface

MPLAB ICD 5 documentation and support information is discussed in this section.

1.1 Conventions Used in This Guide

The following conventions may appear in this documentation:

Table 1-1. Documentation Conventions

Description	Represents	Examples
Arial font:		
Italic characters	Referenced books	<i>MPLAB® ICD 5 In-Circuit Debugger User's Guide</i>
	Emphasized text	...is the <i>only</i> compiler...
Initial caps	A window	the Output window
	A dialog	the Settings dialog
	A menu selection	Select File and then Save.
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:\Users\User1\Projects
	Keywords	static, auto, extern
	Command-line options	-Opa+, -Opa-
	Bit values	0, 1
	Constants	0xFF, 'A'
Italic Courier New	A variable argument	<i>file.o</i> , where <i>file</i> can be any valid filename
Square brackets []	Optional arguments	xc8 [<i>options</i>] <i>files</i>
Curly brackets and pipe character: { }	Choice of mutually exclusive arguments; an OR selection	errorlevel {0 1}
Ellipses...	Replaces repeated text	<i>var_name</i> [, <i>var_name</i> ...]
	Represents code supplied by user	void main (void) { ... }

1.2 Recommended Reading

This document describes how to use the MPLAB® ICD 5 In-Circuit Debugger. Other useful documents are listed below. The following Microchip documents are available and recommended as supplemental reference resources.

Development Tools Design Advisory

Please read this first!

This document contains important information about operational issues that should be considered when using the MPLAB® ICD 5 In-Circuit Debugger with your target design. Refer to [Development Tools Design Advisory](#) in Developer Help.

MPLAB X IDE WebHelp/User's Guide

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

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

Release Notes for MPLAB® ICD 5 In-Circuit Debugger

For the latest information on using MPLAB ICD 5, select *Help>Release Notes* on the MPLAB X IDE toolbar. The release notes contain update information and known issues that may not be included in this user's guide.

MPLAB® ICD 5 In-Circuit Debugger Quick Start Guide Poster (DS50003240)

This poster shows you how to connect the hardware and install the software for the MPLAB ICD 5 using a target board.

2. About the Debugger

The MPLAB® ICD 5 In-Circuit Debugger/Programmer (DV164055) is Microchip's latest fast and feature-rich emulation and programming tool for Microchip microcontrollers (MCUs), which include PIC®, dsPIC®, AVR® and SAM (Arm®) devices. It debugs and programs with the powerful and easy-to-use graphical user interface of MPLAB X Integrated Development Environment (IDE).

By default, the MPLAB ICD 5 connects to your PC through a high-speed USB 2.0 interface. However, you can also use Ethernet connections.

The MPLAB ICD 5 connects to targets using a flat cable, connected at one end to the debugger, and at the other to the target device communication.

The debugger communicates with devices that have built-in emulation circuitry, instead of special debugger chips, so executes code like an actual device. All available features of a given device are accessible interactively and can be set and modified by the MPLAB X IDE interface.

The MPLAB ICD 5 was developed for debugging embedded processors with rich debug facilities which differ from conventional system processors in the following aspects:

- Processors run at maximum speeds
- Multi-communication mediums (Windows®, Linux®, and macOS®)
- Advanced communication mediums and protocols
- Fast programming times

In addition to emulation functions, the MPLAB ICD 5 system also may be used as a device production programmer.

2.1 Advantages

The MPLAB ICD 5 In-Circuit Debugger system provides the following advantages:

Features/Capabilities:

- Connects to a computer via high-speed USB 2.0 or Ethernet.
- Debugs at full speed.
- Monitors internal file registers.
- Configures pin drivers.
- Connects to new targets using an RJ11 or RJ45 modular cable. Also connects to legacy targets.
- Supports multiple breakpoints, stopwatch, and source code file debugging.
- Programs devices using MPLAB X IDE or MPLAB IPE.
- Debugs application on user's hardware in real time.
- Sets breakpoints based on internal events.
- Field-upgradeable through firmware download.
- Adds new device support and features by installing the latest version of device and tool packs (available as a free download at www.microchip.com/mplabx/).
- Operates within a temperature range of 0-70 degrees Celsius.

Performance/Speed:

- No firmware download delays incurred when switching devices.
- A 32-bit MCU running at 300 MHz with 384K bytes of RAM.
- A buffer memory of 4 MB.

Safety:

- Receive feedback from debugger when external power supply is needed for target.
- Supports target supply voltages from 1.2V to 5.5V.
- Safely power up to 1A with the PoE Power Supply or a PC capable of providing 3A on the USB Type-C® connector.
- Protection circuits are added to the probe drivers to guard from power surges from the target.
- V_{DD} and V_{PP} voltage monitors protect against overvoltage conditions/all lines have over-current protection.
- Power pins are physically isolated until voltage is determined to be safe for connection, programmable resistor value, and direction (pull-up, pull-down, or nonexistent).
- Controlled programming speed provides flexibility to overcome target board design issues.
- CE and RoHS compliant (conforms to industry standards).

2.2 Components

The components of the MPLAB ICD 5 In-Circuit Debugger kit box are:

- The rectangular MPLAB ICD 5 unit housed in a durable black and metallic case, which is accented with an LED indicator bar (see figure). On the sides of the unit are the USB connector, Ethernet connector, power connector, as well as the communication and debug connectors.
- A USB Type-C® to Type-C cable for default computer-to-debugger communication.
- 14 pin cable: JTAG female flat ribbon IDC connector, 15 cm.
- 20 pin cable: JTAG female flat ribbon IDC connector, 12 cm.
- 8 conductor ICSP black modular cable, 6 inches.
- 6 conductor ICSP gray modular cable, 6 inches.
- 10 pin cable: cortex female flat ribbon connector, 12 cm.
- **Figure 2-1.** ICD 5 Box Contents



Figure 2-2. Unit Enclosure

Additional hardware and accessories may be ordered separately from the Microchip Purchasing and Client Services website (www.microchipdirect.com).

A CAT5e/CAT6 Ethernet cable may be purchased elsewhere. Cables that do not have an anti-snap boot fit best.

Additional Power over Ethernet (PoE) power supplies that have been tried and tested with the unit include:

- [Microchip PoE Injector 30W 55V Desktop](#)
- [BV-Tech Gigabit PoE+ Injector 30W](#)
- [TRENDnet Gigabit PoE+ Injector 30W](#)

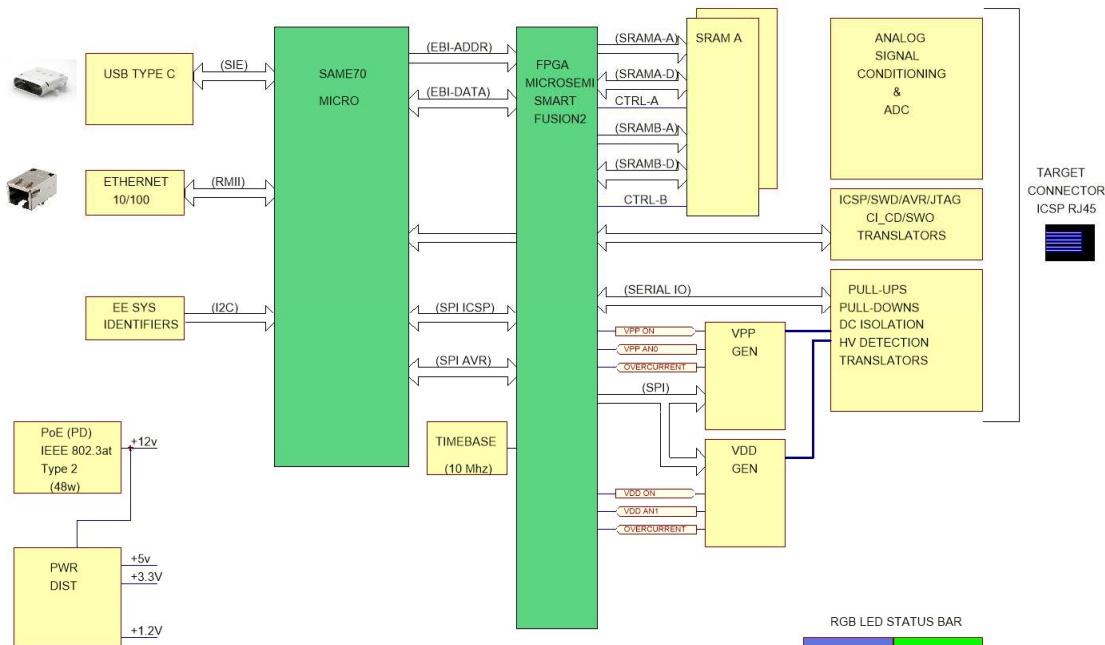
USB isolators that have been tried and tested with the unit include:

- [IF Tools USB Isolator HVC](#)
- [HiFimeUSB Isolator](#)
- [Topping HSO2 USB Isolator](#)

Note: KETEREX KX USB-150 is not able to provide the 500ma minimum that the ICD 5 requires.

2.3 Block Diagram

Below is a block diagram of basic MPLAB ICD 5 unit operational capabilities.



2.4 Using MPLAB® ICD 5 with MPLAB X IDE and MPLAB IPE

Download and install the latest version of MPLAB X IDE from the [MPLAB X IDE](#) webpage. The MPLAB X IDE installer will install MPLAB X IDE and/or MPLAB IPE.

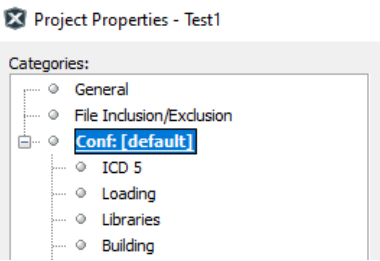
Update to the latest ICD 5 tool pack from within the MPLAB pack manager.

Using MPLAB® ICD 5 with MPLAB X IDE

The MPLAB® ICD 5 In-Circuit Debugger works with MPLAB X IDE to develop target applications. The user's guide and other documentation may be found on the [MPLAB X IDE](#) webpage.

Table 2-1. MPLAB X IDE Overview

	Use the desktop icon to launch the IDE.
	Create a new project or open an existing project. Select MPLAB ICD 5 as the hardware tool.

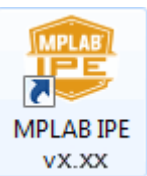
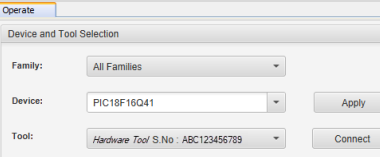
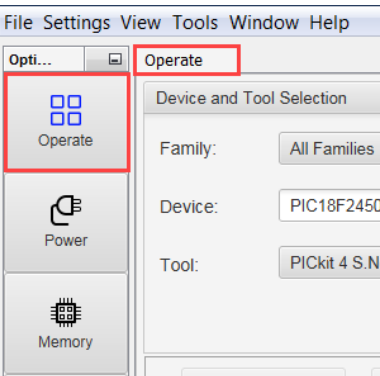
	Open the Project Properties window by right clicking on the project name and selecting "Properties." This window is used to set up options for debugging, programming and other features. See MPLAB ICD 5 option descriptions.
---	---

Using MPLAB® ICD 5 with MPLAB IPE

The MPLAB® ICD 5 In-Circuit Debugger works with MPLAB IPE as a production programmer. The user's guide and other documentation may be found on the [MPLAB IPE](#) webpage.

There are also command line IPE tools as an option.

Table 2-2. MPLAB IPE Overview

	Use the desktop icon to launch the IPE.
	Select a device to program and then select MPLAB ICD 5 as the tool.
	Select on a button to Program, Erase, Read, Verify or Blank Check. For more on MPLAB IPE, including Advanced mode, see the MPLAB IPE User's Guide.

3. Connections

The MPLAB® ICD 5 In-Circuit Debugger hardware setup begins by connecting power, communications, and targets to the debugger. For legacy targets, an adapter board and cables are provided. A Debugger Adapter Board is also provided to expand ICD5 connection capabilities to a wide range of connector types.

3.1 Power and Self Test

MPLAB ICD 5 can be powered by USB-C power or Power over Ethernet (PoE). It does not use an external power supply. The MPLAB ICD 5 can provide power to the target with PoE. PoE also powers the tool if USB is unplugged. For customers who power their MPLAB ICD 5 directly from USB-C, it will depend on the power capabilities of the host PC which MPLAB X IDE will detect once the USB is connected. For details, see [10.2. Power Specifications](#).

Power-up Self Test

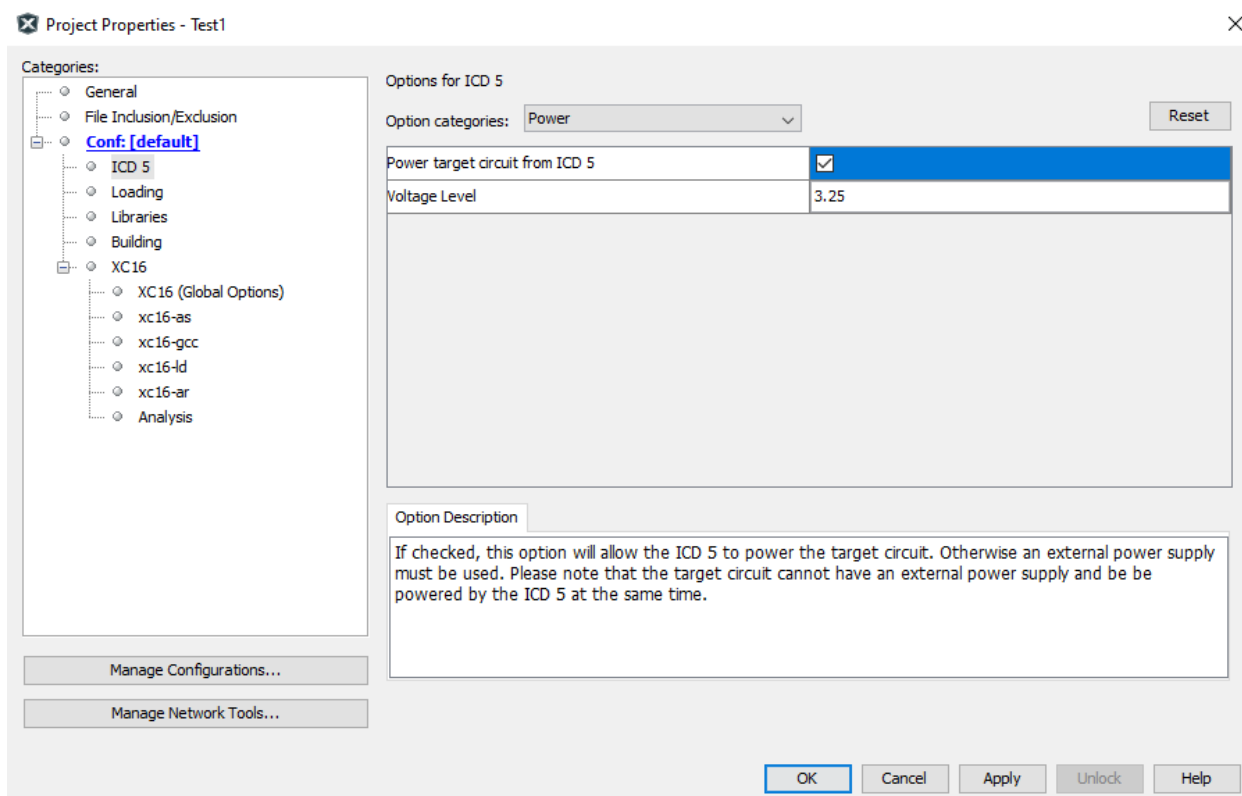
The MPLAB ICD 5 unit performs a built-in self-test (or BIST) during power-up. Errors that occur during this test are reported in the MPLAB X IDE or IPE Output window. Depending on the error, LED colors may indicate the error as well.

Power the Target

It is possible to power the target using the debugger. For details, see [10.2. Power Specifications](#).

Select this option in the Project Properties window (see figure below). Also select the desired target voltage.

Figure 3-1. Select Power to Target



Related Links

[10.2. Power Specifications](#)

[8. Error Messages](#)

[10.3. Indicator Lights \(LEDs\)](#)

3.2 PC Connections

MPLAB® ICD 5 In-Circuit Debugger can connect with the PC (and MPLAB X IDE/MPLAB IPE) using the connections in the table below.

Connection Type	Connection Details	Programming and Debugging*	Trace*	MPLAB Data Visualizer Support
USB Type-C® (default)	HS USB 2.0	Yes (USB 2.0)	Yes (SWO)	Yes
Ethernet	Direct or via network	Yes	No	No

* For speed specifications, see [10.4. PC Connection Specifications](#).

Figure 3-2. MPLAB ICD 5 Power and PC Connections



Begin with the USB (default) connection. Then switch to Wi-Fi or Ethernet using the “Manage Network Tools” dialog found under *Tools>Manage Network Tools*. For details, see the following topics.

Selecting Ethernet communication instead of USB has several uses:

- Access a target remotely. The debugger and target can be in one location and a PC in another.
- Isolate the target. Targets that need to be in a controlled environment can be separate from the PC location.

Related Links

[10.4.1. USB Type-C Connector \(J1\) and Cable](#)

[10.4.2. Ethernet Connector \(J6\) and Cable](#)

3.2.1 USB Default Connection

The default connection between the PC and MPLAB ICD 5 unit is USB using a USB Type-C cable. It is recommended that you use the cable that comes with the kit to avoid communication issues.

Note: Only USB communications can be used for trace.

Note: MPLAB Data Visualizer, whether as an MPLAB X IDE included plugin or as a stand-alone application, can only detect the debugger when it is using USB communications.

If you have problems with the other types of communications, return to USB and then use the Manage Network Tools (MNT) dialog to switch to Ethernet again.

3.2.2 Ethernet Modes

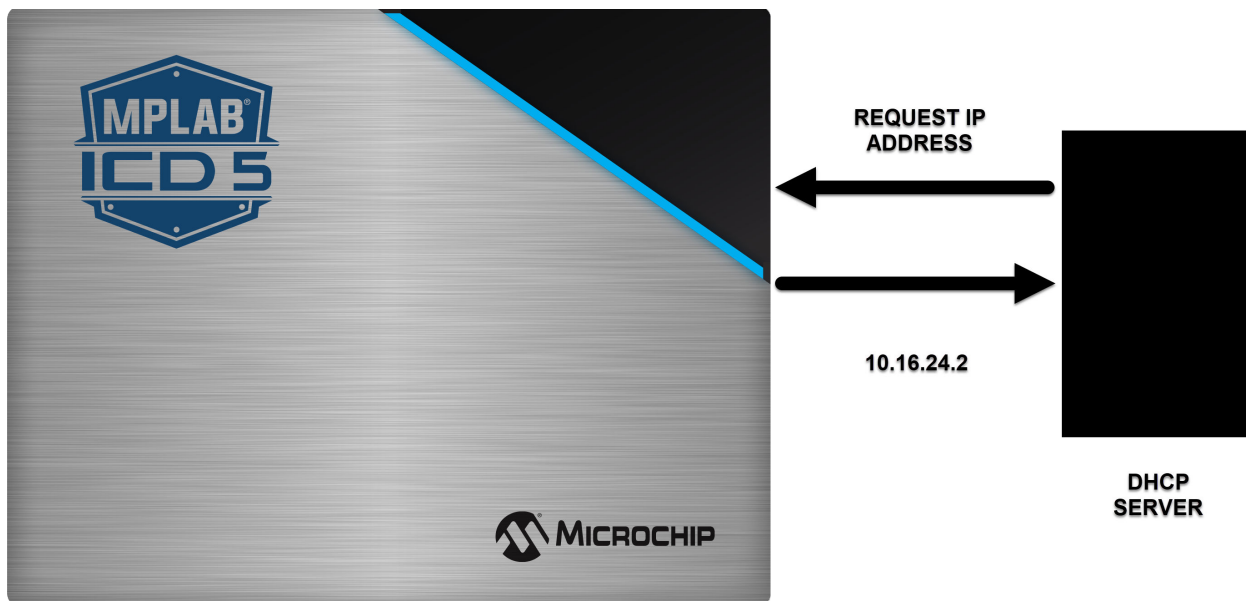
MPLAB® ICD 5 In-Circuit Debugger supports different modes of Ethernet communication.

Note: It is intended to work on a local network and not a wide area network.

3.2.2.1 Ethernet - DHCP

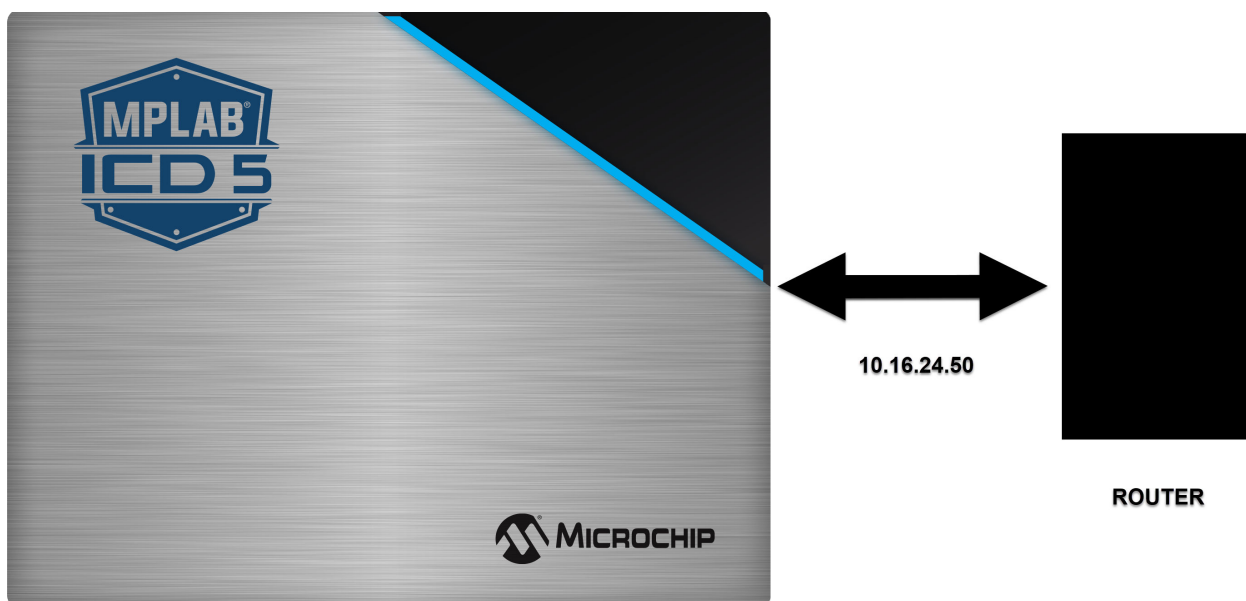
When connecting MPLAB® ICD 5 In-Circuit Debugger to the Ethernet network, you can use DHCP, Static IP or APIPA protocols.

DHCP (Dynamic Host Configuration Protocol) is for assigning dynamic IP addresses to devices that are connected to the network.



3.2.2.2 Ethernet - Static IP

A Static IP address is one that is permanently assigned to your network devices. The Static IP address can be changed by going through the provisioning process, otherwise it is permanent.



3.2.2.3 Ethernet - APIPA

Automatic Private IP Addressing (APIPA) is a feature in Windows operating systems that enables computers to automatically self-configure an IP address and subnet mask when their DHCP server isn't reachable. When connecting the MPLAB ICD 5 In-Circuit Debugger to the Ethernet, if the network does not have a DHCP server or is unavailable, the network device will use APIPA to assign itself an IP address. APIPA is a network capability feature that allows network devices to automatically assign IP addresses to themselves. It was developed for small non-routable networks as it removes the hassle of figuring out which IP addresses are available and manually assigning your device a static IP address.

Note: If your network device is only using a self-assigned APIPA address, it will only be able to connect with other network devices on that same APIPA subnet.

The use of APIPA addresses is useful for a temporary connection when a DHCP server is experiencing issues or needs to be offline for a while or the goal is for a separate or dedicated network. Ideally, the APIPA should be used when troubleshooting DHCP issues with the DHCP server or a device is unable to obtain an IP address. Connecting to a network and using its DHCP server is the recommended practice when using Microchip's network capable debugging tools.

The APIPA IP address range is from 169.254.0.1 to 169.254.255.254.

3.2.3 Ethernet - Setup and Tool Discovery

Follow the steps in the table to set up the desired Ethernet mode and then find the connection.

Before beginning this setup process, you will need to have a PoE injector. Refer to [2.2. Components](#) for PoE tools that have been tested with the debugger. The provisioning - which is setting up the Ethernet configuration in the **Manage Network Tools** window for the desired protocol - is only required one time.

Table 3-1. Ethernet Setup and Tool Discovery in MPLAB® X IDE

Step	Action
1	Connect the debugger to your PC via the USB cable. If you will be using Ethernet communication, a PoE injector is mandatory. Note: A USB connection is required at first to setup Ethernet communication.
2	Go to Tools > Manage Network Tools in MPLAB X IDE.
3	Under "Network Capable Tools Plugged into USB," select your debugger.
4	Under "Configure Default Connection Type for Selected Tool" select the radio button for the connection you want. Ethernet (Wired/StaticIP): Input Static IP Address, Subnet Mask and Gateway. Note: To learn how to obtain this information to input, refer to the paragraph following the figure. Click Update Connection Type .
5	If Ethernet communication was chosen, ensure the PoE injector is connected and then unplug the USB cable from your debugger unit. Note: Keep the Manage Network Tools window open.
6	The debugger will restart automatically and come up in the connection mode you selected. Then: The LEDs will display for either a successful network connection (white) or a network connection failure/error (red).
7	Now go back to the "Manage Network Tools" dialog and click on the Scan button, which will list your debugger under "Active Discovered Network Tools." Select the checkbox for your tool and close the dialog.
8	If your debugger is not found under "Active Discovered Network Tools," you can manually enter information in the "User Specified Network Tools" section. You must know the IP address of the tool (by the way of network admin or static IP assignment).

Your debugger and PoE setup should look like this:

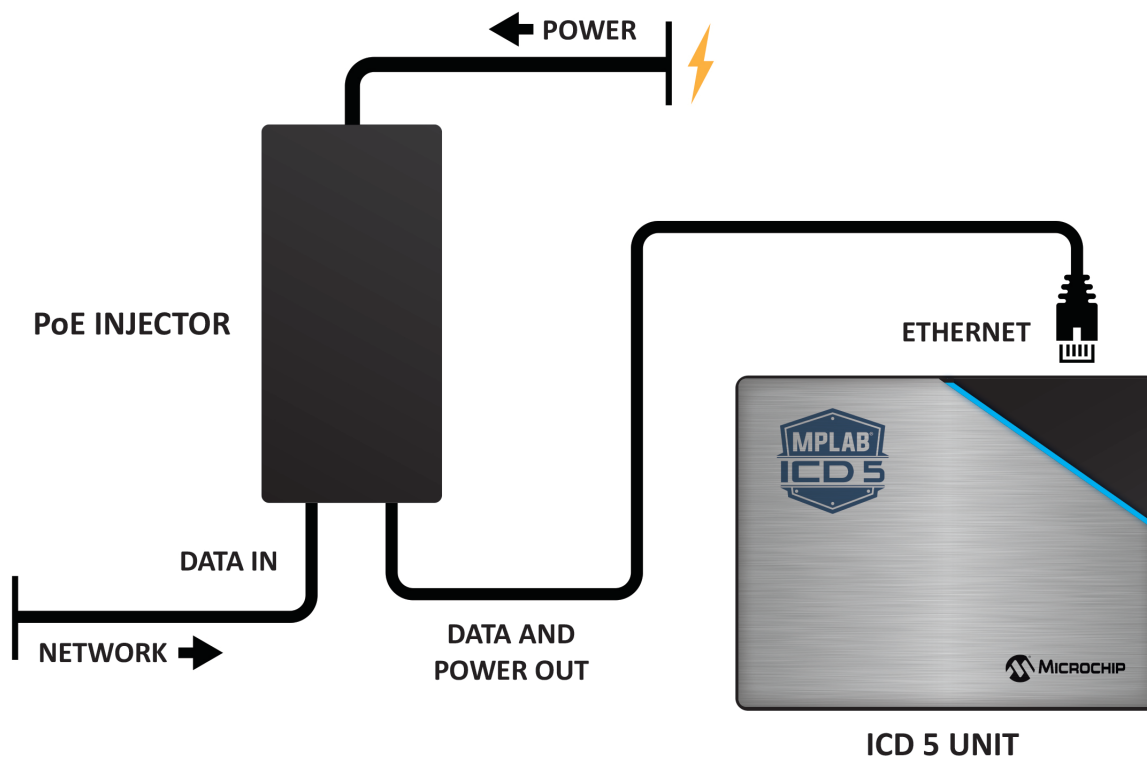
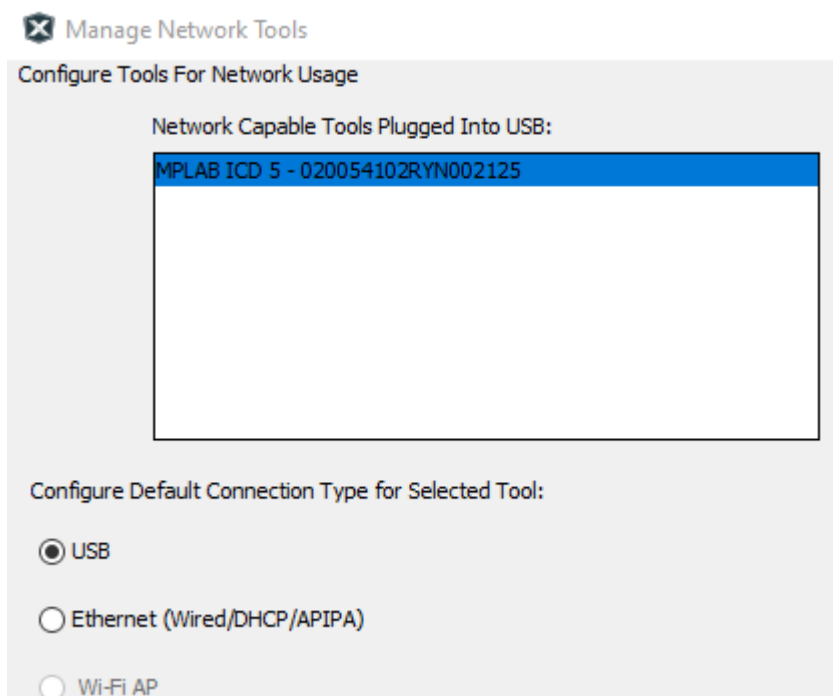


Figure 3-3. Initial USB Connection

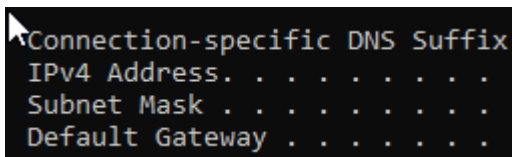


To obtain IP configuration details:

1. Run the Command Prompt application in your operating system by typing `cmd` in your system search bar.

2. Type `ipconfig` into the black window that appears then hit **Enter**.
3. Scroll down to view the IP address, Subnet Mask and Default Gateway.

Figure 3-4. Command Prompt Window



3.3 Target Connections

MPLAB® ICD 5 In-Circuit Debugger connects to a target via an 8-pin flat cable assembly. For legacy target connections, an adapter board is available. There is also a Current Sense connection for use when Power Debugging.

Device and communication types, as well as an available adapter board, are discussed in the following sections.

Note: MPLAB ICD 5 can power the target. For details, see [10.2. Power Specifications](#). Select powering the target in the Project Properties window, "ICD 5" category, "Power" option category.

Figure 3-5. MPLAB ICD 5 Unit to Target Connection

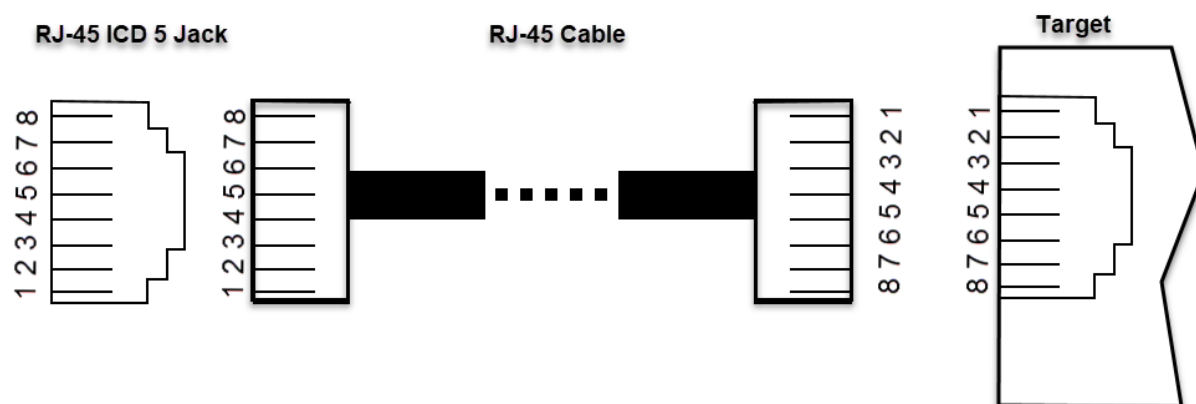


3.3.1 Connecting the Debugger to an RJ-45 Target via an RJ-45 Type Cable

The MPLAB ICD 5 In-Circuit Debugger has an RJ-45 connector for communication to the target. Connect the RJ-45 type cable into the RJ-45 connector. Connect the other end of the cable to the RJ-45 connector on the target.

Refer to the figure below for the pinouts for this connection.

Figure 3-6. RJ-45 Connections to Target



3.3.2 Target Connection Pinouts

The programming connector pin functions are different for various devices and interfaces. Refer to the following pinout tables for debug and data stream interfaces. Legacy 6-pin RJ-11 cable can also be used, however target interfaces which use pins 1 (TMS/SWDIO) and 8 (TDI/MOSI) can not be programmed or debugged.

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

Table 3-2. Pinouts for Debug Interfaces

MPLAB ICD 5		DEBUG										TARGET ⁴	
8-Pin Modular Connector ¹	Pin #	Pin Name	ICSP™ (MCHP)	MIPS EJTAG	Cortex® SWD	AVR® JTAG	AVR debugWIRE	AVR UPDI	AVR PDI	AVR ISP	AVR TPI	8-Pin Modular Connector	6-Pin Modular Connector
	8	TTDI		TDI		TDI				MOSI		1	
	7	TVPP	MCLR /Vpp	MCLR	RESET			RESET ³				2	1
	6	TVDD	VDD	VDD/VDDIO	VDD	VTG	VTG	VTG	VTG	VTG	VTG	3	2
	5	GND	GND	GND	GND	GND	GND	GND	GND	GND	GND	4	3
	4	PGD	DAT	TDO	SWO ²	TDO		DAT ³	DAT	MISO	DAT	5	4
	3	PGC	CLK	TCK	SWCLK	TCK				SCK	CLK	6	5
	2	TAUX				RESET	RESET/dW		CLK	RESET	RESET	7	6
	1	TTMS		TMS	SWDIO ²	TMS						8	
1. Black (8-pin) cable must be used for EJTAG, JTAG, SWD and ISP. 2. SWO is used for trace. SWDIO is for debug. 3. Pin may be used for High-Voltage Pulse reactivation of UPDI function depending on device. See device data sheet for details. 4. These are example target connectors that are assumed similar to the debug unit (modular).													

Figure 3-7. 8-Pin Modular Connector**Table 3-3. Pinouts for Data Stream Interfaces**

MPLAB® ICD 5	DATA STREAM		TARGET ²	
8-Pin Modular Connector	PIC and AVR Devices	SAM Devices ¹	8-Pin Modular Connector	6-Pin Modular Connector
Pin #	DGI UART / CDC	DGI UART / CDC	Pin #	Pin #
8	TX (target)	TX (target)	1	
7			2	1
6	VTG	VTG	3	2
5	GND	GND	4	3
4			5	4
3			6	5
2		RX (target)	7	6
1	RX (target)		8	

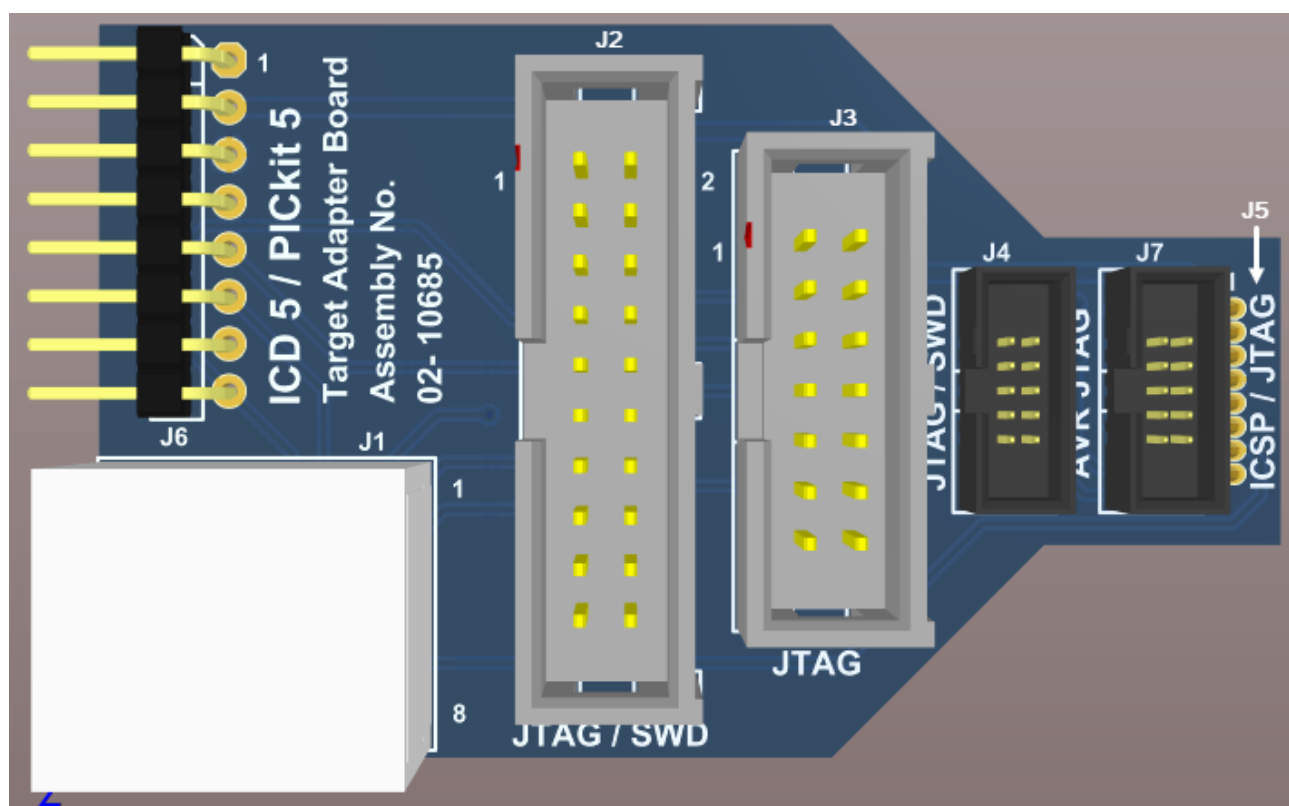
1. RX pin moved because of wiring for other devices.
2. These are example target connectors that are assumed similar to the debug unit (modular).



Note: For 6-pin RJ11 into 8-pin RJ45 socket, pins 1 and 8 are lost.

3.3.3 Debugger Adapter Board

The Debugger Adapter Board is a connectivity board that gives MPLAB ICD 5 and MPLAB PICKit 5 Debuggers cable compatibility to demo boards with Atmel-ICE, Power Debugger and ARM style connectors. It supports JTAG, SWD and ICSP protocols in multiple connector formats. It is useful for debugging AVR Xplained demonstration boards with MPLAB PICKit 5 debuggers.

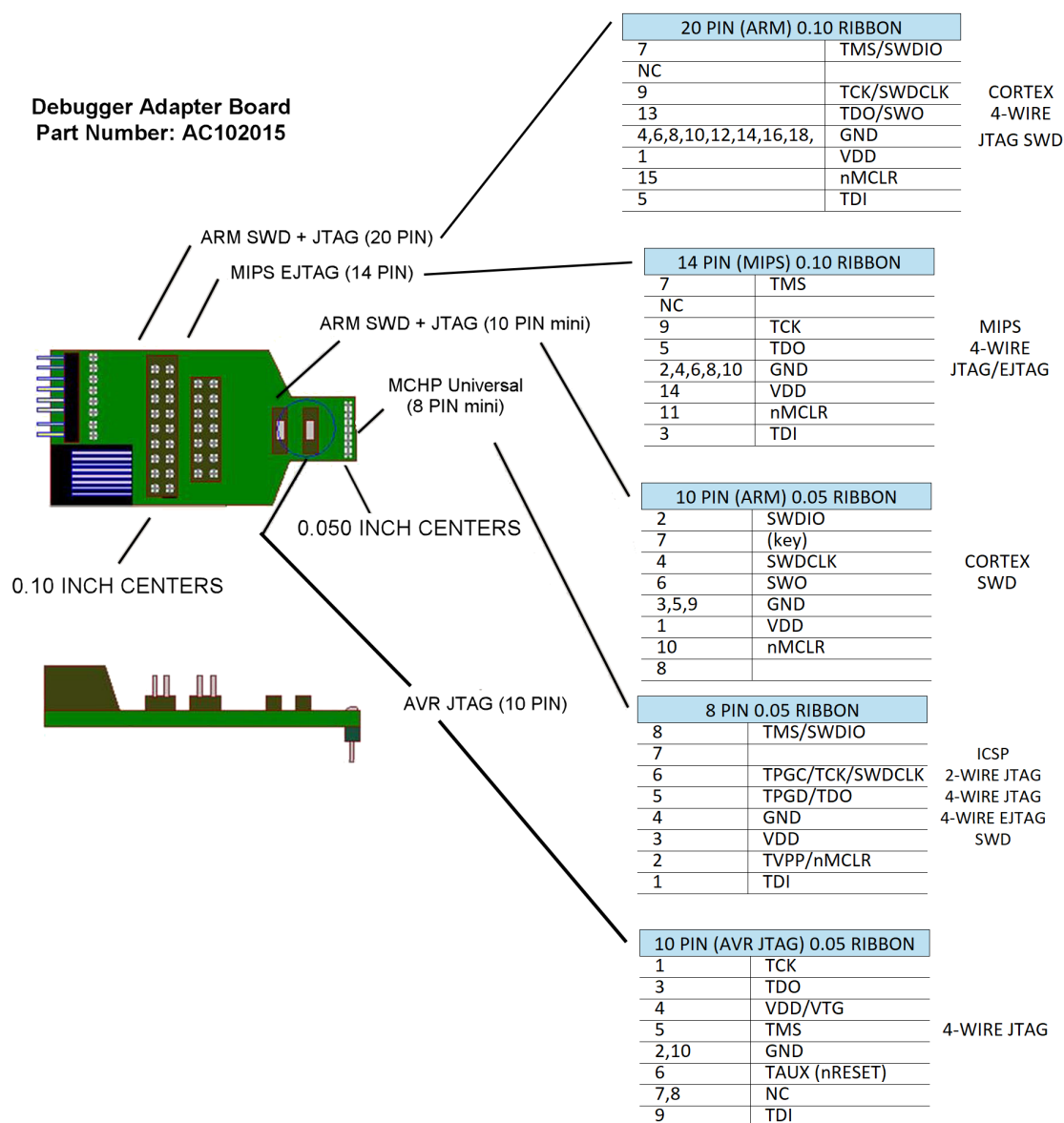
**Table 3-4.** Connection to Device Mappings

Connector	Device	Input/Output
J1	RJ-11 modular connector for ICSP (PIC MCUs)	Input from tool
J2	JTAG/SWD, 20 pin - SAM MCUs	Output to target
J3	JTAG (MIPS EJTAG), 14 pin - PIC32 MCUs	Output to target
J4	JTAG/SWD, 10 pin mini - SAM MCUs	Output to target
J5	ICSP/JTAG, 8 pin mini - PIC MCUs	Output to target
J6	8-pin single inline connector for ICSP (PIC MCUs)	Input from tool
J7	AVR JTAG, 10 pin mini - AVR MCUs	Output to target

3.3.3.1 Adapter Board Pinout

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

Figure 3-8. MPLAB ICD 5 Adapter Board (AC102015) Pinouts



3.3.4 SAM MCUs - JTAG/SWD Interfaces

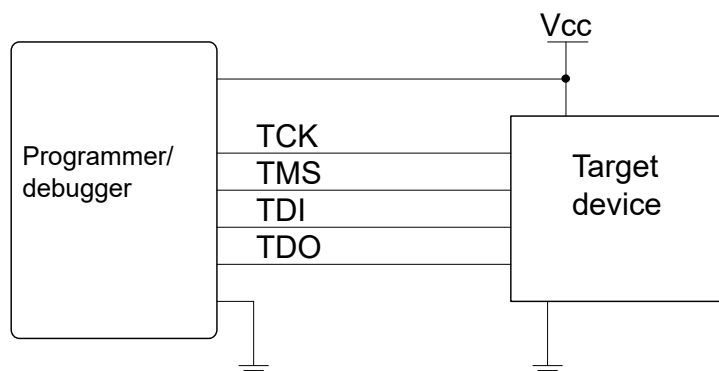
SAM devices feature the Serial Wire Debug (SWD) interface for programming and debugging and/or a JTAG interface with identical functionality. Check the device data sheet for supported interfaces of that device.

3.3.4.1 JTAG Physical Interface

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

To use this target interface with MPLAB X IDE, open the Project Properties window, "ICD 5" category, "Communications" option category, and select JTAG. For MIPS devices, select 2-wire or 4-wire JTAG.

Note: 2-wire JTAG uses standard ICSP pinout.

Figure 3-9. JTAG Interface Basics

3.3.4.1.1 Connecting to a SAM JTAG Target

The MPLAB ICD 5 using the [3.3.4. SAM MCUs - JTAG/SWD Interfaces](#) provides a legacy 10-pin 50-mil JTAG connection as well as a legacy 20-pin 100-mil JTAG connection.

Direct Connection to a 10-pin 50-mil Header

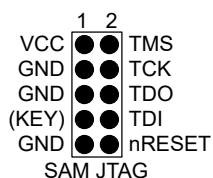
Use the 10-pin 50-mil flat cable to connect directly to a target board with headers complying with the Arm® Cortex® Debug header pinout shown in [3.3.4.1.2. SAM JTAG Pinout \(Cortex-M debug connector\)](#).

Direct Connection to a 20-pin 100-mil Header

Plug the adapter board into targets with a 20-pin 100-mil header.

3.3.4.1.2 SAM JTAG Pinout (Cortex®-M debug connector)

When designing an application PCB which includes a Microchip SAM with the JTAG interface, it is recommended to use the pinout as shown in the figure below.

Figure 3-10. SAM JTAG Header Pinout**Table 3-5. SAM JTAG Pin Description**

Name	Pin	Description
TCK	4	Test Clock (clock signal from the MPLAB ICD 5 into the target device).
TMS	2	Test Mode Select (control signal from the MPLAB ICD 5 into the target device).
TDI	8	Test Data In (data transmitted from the MPLAB ICD 5 into the target device).
TDO	6	Test Data Out (data transmitted from the target device into the MPLAB ICD 5).
nRESET	10	Reset (optional).
VTG	1	Target voltage reference. Not used by MPLAB ICD 5.
GND	3, 5, 9	Ground. All must be connected to ensure that the MPLAB ICD 5 and the target device share the same ground reference.
KEY	7	Connected internally to the TRST pin on the AVR connector. Recommended as not connected.



Tip: Remember to include a decoupling capacitor between VTG and GND.

3.3.4.2 SAM SWD Interface

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

3.3.4.2.1 Connecting to a SAM SWD Target

When connecting to an SWD device, the 10-pin JTAG connector can be used.

3.3.4.2.2 SAM SWD Pinout

For the 10-pin JTAG connector:

Figure 3-11. SAM SWD Header Pinout

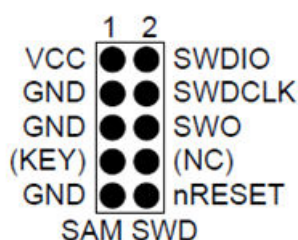


Table 3-6. SAM SWD Pin Description

Name	SAM Port Pin	Description
SWDCLK	4	Serial Wire Debug Clock
SWDIO	2	Serial Wire Debug Data Input/Output
SWO	6	Serial Wire Output (optional- not implemented on all devices)
nSRST	10	Reset
VTG	1	Target voltage reference - not used by MPLAB ICD 5
GND	3, 5, 9	Ground

3.3.5 AVR MCUs Connections

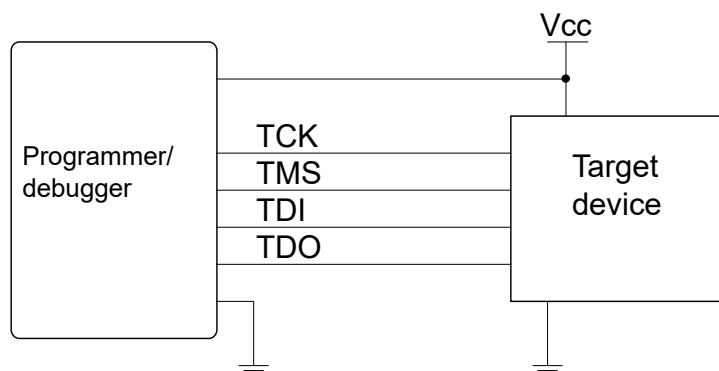
AVR devices feature various programming and debugging interfaces. Check the device datasheet for supported interfaces of that device.

The [MPLAB ICD 5 ICSP Adapter Board](#) may be used with additional AVR 10-pin or 6-pin JTAG plugin boards for legacy target connections. Snap off the plugin board needed (see figure below) and plug into the adapter board 8-pin SIL connector.

3.3.5.1 JTAG Physical Interface

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

To use this target interface with MPLAB X IDE, open the Project Properties window, “ICD 5” category, “Communications” option category. It will say JTAG. Select the JTAG speed.

Figure 3-12. JTAG Interface Basics

3.3.5.2 Connecting to a AVR JTAG Target

The MPLAB ICD 5 using the [MPLAB ICD 5 ICSP Adapter Board](#) and the [AVR JTAG \(10 Pin\) Adapter Board](#) provides a legacy 10-pin 50-mil JTAG connection.

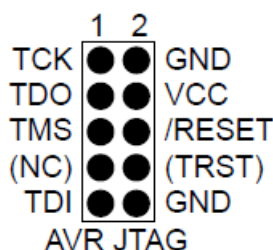
Direct Connection to a 10-pin 50-mil Header

Use the 10-pin 50-mil plugin connector on the “AVR JTAG (10 pin)” adapter board to connect directly to a target board with headers complying with the header pinout.

3.3.5.3 AVR JTAG Pinout

When designing an application PCB, which includes an AVR with the JTAG interface, it is recommended to use the pinout as shown in the figure below.

For other AVR connections, see [4.3. AVR Devices - On-Chip Debugging \(OCD\)](#).

Figure 3-13. AVR JTAG Header Pinout**Table 3-7.** AVR JTAG Pin Description

Name	Pin	Description
TCK	1	Test Clock (clock signal from the MPLAB ICD 5 into the target device).
TMS	5	Test Mode Select (control signal from the MPLAB ICD 5 into the target device).
TDI	9	Test Data In (data transmitted from the MPLAB ICD 5 into the target device).
TDO	3	Test Data Out (data transmitted from the target device into the MPLAB ICD 5).
nTRST	8	Test Reset (optional, only on some AVR devices). Used to reset the JTAG TAP controller.
nSRST	6	Reset (optional). Used to reset the target device. Connecting this pin is recommended since it allows the MPLAB ICD 5 to hold the target device in a reset state, which can be essential to debugging in certain scenarios.
VTG	4*	Target voltage reference. The MPLAB ICD 5 samples the target voltage on this pin in order to power the level converters correctly. The MPLAB ICD 5 draws less than 1mA from this pin in this mode.
GND	2, 10	Ground. All must be connected to ensure that the MPLAB ICD 5 and the target device share the same ground reference.

* Remember to include a decoupling capacitor between VTG and GND.

3.3.5.4 AVR SPI Physical Interface

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

3.3.5.4.1 Connecting to an AVR SPI Target

The recommended pinout for the 6-pin SPI connector is shown in [3.3.5.4.2. AVR SPI Pinout](#).

Connection to a 6-pin 100-mil SPI Header

Use the AVR 6-pin adapter board to connect to a standard 100-mil SPI header.

Connection to a 6-pin 50-mil SPI Header

Use the AVR 6-pin mini adapter board to connect to a standard 50-mil SPI header.



Important:

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

Follow the instructions in the popup window that appears.



Info:

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

3.3.5.4.2 AVR SPI Pinout

When designing an application PCB, which includes an AVR with the SPI interface, the pinout, as shown in the figure below, should be used.

Figure 3-14. SPI Header Pinout

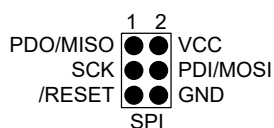


Table 3-8. SPI Pin Mapping

AVR PORT Pins	Target Pins	SPI Pinout
Pin 1 (TCK)	SCK	3
Pin 2 (GND)	GND	6
Pin 3 (TDO)	MISO	1
Pin 4 (VTG)	VTG	2
Pin 5 (TMS)		
Pin 6 (nSRST)	/RESET	5
Pin 7 (not connected)		
Pin 8 (nTRST)		

.....continued

AVR PORT Pins	Target Pins	SPI Pinout
Pin 9 (TDI)	MOSI	4
Pin 10 (GND)		

3.3.5.5 AVR PDI

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

3.3.5.5.1 Connecting to an AVR PDI Target

The recommended pinout for the 6-pin PDI connector is shown in [3.3.5.5.2. AVR PDI Pinout](#).

Connection to a 6-pin 100-mil PDI Header

Use the AVR 6-pin adapter board to connect to connect to a standard 100-mil PDI header.

Connection to a 6-pin 50-mil PDI Header

Use the AVR 6-pin mini adapter board to connect to a standard 50-mil PDI header.

3.3.5.5.2 AVR PDI Pinout

When designing an application PCB, which includes a Microchip AVR with the PDI interface, the pinout shown in the figure below, should be used.

Figure 3-15. PDI Header Pinout

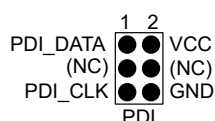


Table 3-9. PDI Pin Mapping

AVR PORT Pin	Target Pins	Microchip STK600 PDI Pinout
Pin 1 (TCK)		
Pin 2 (GND)	GND	6
Pin 3 (TDO)	PDI_DATA	1
Pin 4 (VTG)	VTG	2
Pin 5 (TMS)		
Pin 6 (nSRST)	PDI_CLK	5
Pin 7 (not connected)		
Pin 8 (nTRST)		
Pin 9 (TDI)		
Pin 10 (GND)		

3.3.5.6 AVR UPDI

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

3.3.5.6.1 UPDI and /RESET

The UPDI one-wire interface can be a dedicated pin or a shared pin, depending on the target AVR device. Consult the device data sheet for further information.

When the UPDI interface is on a shared pin, the pin can be configured to be either UPDI, /RESET, or GPIO by setting the RSTPINCFG[1:0] fuses.

The RSTPINCFG[1:0] fuses have the following configurations, as described in the data sheet. The practical implications of each choice are given here.

Table 3-10. RSTPINCFG[1:0] Fuse Configuration

RSTPINCFG[1:0]	Configuration	Usage
00	GPIO	General purpose I/O pin. To access UPDI, a 12V pulse must be applied to this pin. No external Reset source is available.
01	UPDI	Dedicated programming and debugging pin. No external Reset source is available.
10	Reset	Reset signal input. To access UPDI, a 12V pulse must be applied to this pin.
11	Reserved	NA

Note: Older AVR devices have a programming interface, known as “High-Voltage Programming” (both serial and parallel variants exist). In general, this interface requires 12 V to be applied to the /RESET pin for the duration of the programming session. The UPDI interface is an entirely different interface. The UPDI pin is primarily a programming and debugging pin, which can be fused to have an alternative function (/RESET or GPIO). If the alternative function is selected then a 12 V pulse is required on that pin to re-activate the UPDI functionality.

Note: If a design requires the sharing of the UPDI signal due to pin constraints, steps must be taken to ensure that the device can be programmed. To ensure that the UPDI signal can function correctly, as well as to avoid damage to external components from the 12 V pulse, it is recommended to disconnect any components on this pin when attempting to debug or program the device. This can be done using a 0 Ω resistor, which is mounted by default and removed or replaced by a pin header while debugging. This configuration effectively means that programming should be done before mounting the device.

3.3.5.6.2 Connecting to an AVR UPDI Target

The recommended pinout for the 6-pin UPDI connector is shown in [3.3.5.6.3. AVR UPDI Pinout](#).

Connection to a 6-pin 100-mil UPDI Header

Use the AVR 6-pin adapter board to connect to a standard 100-mil UPDI header.

Connection to a 6-pin 50-mil UPDI Header

Use the AVR 6-pin mini adapter board to connect to a standard 50-mil UPDI header.

3.3.5.6.3 AVR UPDI Pinout

When designing an application PCB, which includes a Microchip AVR with the UPDI interface, the pinout shown below should be used.

Figure 3-16. UPDI Header Pinout

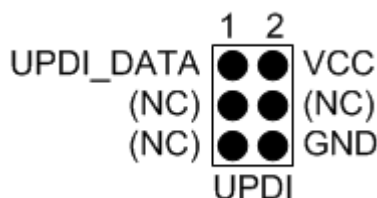


Table 3-11. UPDI Pin Mapping

AVR PORT Pin	Target Pins	Microchip STK600 UPDI Pinout
Pin 1 (TCK)		
Pin 2 (GND)	GND	6

.....continued

AVR PORT Pin	Target Pins	Microchip STK600 UPDI Pinout
Pin 3 (TDO)	UPDI_DATA	1
Pin 4 (VTG)	VTG	2
Pin 5 (TMS)		
Pin 6 (nSRST)		
Pin 7 (Not connected)		
Pin 8 (nTRST)		
Pin 9 (TDI)		
Pin 10 (GND)		

3.3.5.7 AVR TPI

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

3.3.5.7.1 Connecting to an AVR TPI Target

The recommended pinout for the 6-pin TPI connector is shown in [3.3.5.7.2. AVR TPI Pinout](#).

Connection to a 6-pin 100-mil TPI Header

Use the AVR 6-pin adapter board to connect to a standard 100-mil TPI header.

Connection to a 6-pin 50-mil TPI Header

Use the AVR 6-pin mini adapter board to connect to a standard 50-mil TPI header.

3.3.5.7.2 AVR TPI Pinout

When designing an application PCB, which includes an AVR with the TPI interface, the pinout shown in the figure below, should be used.

Figure 3-17. TPI Header Pinout

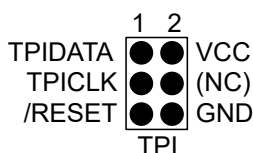


Table 3-12. TPI Pin Mapping

AVR PORT Pins	Target Pins	TPI Pinout
Pin 1 (TCK)	CLOCK	3
Pin 2 (GND)	GND	6
Pin 3 (TDO)	DATA	1
Pin 4 (VTG)	VTG	2
Pin 5 (TMS)		
Pin 6 (nSRST)	/RESET	5
Pin 7 (not connected)		
Pin 8 (nTRST)		
Pin 9 (TDI)		
Pin 10 (GND)		

3.3.5.8 AVR debugWIRE

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

Note:

The debugWIRE interface can not be used as a programming interface. This means that the SPI interface must also be available (as shown in [3.3.5.4.2. AVR SPI Pinout](#)) in order to program the target.

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

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

3.3.5.8.1 AVR Connecting to debugWIRE

The recommended pinout for the 6-pin debugWIRE (SPI) connector is shown in [3.3.5.8.2. AVR debugWIRE Pinout](#).

Connection to a 6-pin 100-mil SPI Header

Use the 6-pin 100-mil tap on the flat cable (included in some kits) to connect to a standard 100-mil SPI header.

Connection to a 6-pin 50-mil SPI Header

Use the adapter board (included in some kits) to connect to a standard 50-mil SPI header.

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

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

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

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

3.3.5.8.2 AVR debugWIRE Pinout

When designing an application PCB which includes an Microchip AVR with the debugWIRE interface, the pinout shown in the figure below should be used.

Figure 3-18. debugWIRE (SPI) Header Pinout

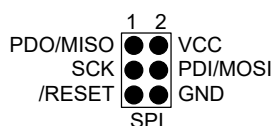


Table 3-13. debugWIRE Pin Mapping

AVR Port Pins	Target pins	debugWIRE Pinout
Pin 1 (TCK)		
Pin 2 (GND)	GND	6
Pin 3 (TDO)		
Pin 4 (VTref)	VTref	2
Pin 5 (TMS)		
Pin 6 (nSRST)	RESET	5
Pin 7 (Not connected)		
Pin 8 (nTRST)		
Pin 9 (TDI)		
Pin 10 (GND)		

3.3.6 PIC32M Connections

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

3.3.6.1 Connecting to a PIC32M EJTAG Target

The MPLAB ICD 5 provides a direct connection for new designs or a legacy 14-pin 10-mil JTAG/EJTAG connection using the adapter board.

3.3.6.2 PIC32M EJTAG Pinout - 4-Wire JTAG

PIC32M EJTAG pin names and descriptions are shown in the table below. Pin numbers are shown for MPLAB ICD 5 direct connection and Debugger Adapter Board 14-pin connection.

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

MPLAB ICD 5 Pin	Adapter Board (14-Pin) Pin	Name	Description
1	11	MCLR	Reset (optional). Used to reset the target device. Connecting this pin is recommended since it allows the MPLAB ICD 5 to hold the target device in a reset state, which can be essential to debugging in certain scenarios.
2	14	VDD	MPLAB ICD 5 providing power to target (optional) or target providing power to MPLAB ICD 5 (PTG).
3	2, 4, 6, 8, 10	GND	Ground. All must be connected to ensure that the MPLAB ICD 5 and the target device share the same ground reference.
4	3	TDO	Test Data Out (data transmitted from the target device into the MPLAB ICD 5).
5	9	TCK	Test Clock (clock signal from the MPLAB ICD 5 into the target device).
6	1	NC	Not connected.
7	5	TDI	Test Data In (data transmitted from the MPLAB ICD 5 into the target device).
8	7	TMS	Test Mode Select (control signal from the MPLAB ICD 5 into the target device).

3.3.7 PIC MCUs - ICSP Connection

The MPLAB® ICD 5 In-Circuit Debugger supports debug and programming of PIC microcontrollers (MCUs) and dsPIC digital signal controllers (DSCs) through ICSP™ (In-Circuit Serial Programming™) connections.

3.3.7.1 ICSP Target Connection

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

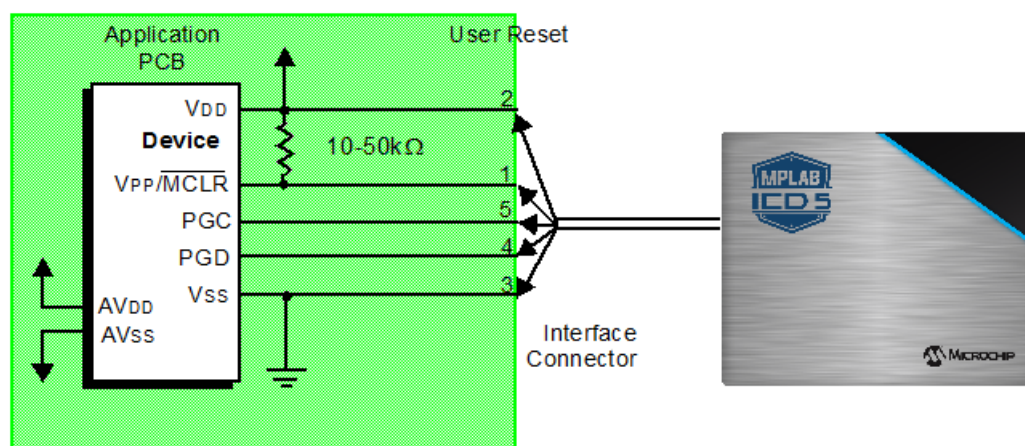
If the debugger and target have different connections (modular-to-inline or inline-to-modular respectively) a small adapter can be purchased to enable proper connections: [“RJ11 to ICSP Adapter” \(AC164110\)](#).

Figure 3-19. 6-Pin RJ11 to ICSP Adapter

Alternatively, the MPLAB ICD 5 using the adapter board provides an 8-pin 50-mil Microchip Universal connection for the 6-pin and 8-pin ICSP interfaces.

3.3.7.2 ICSP Target Connection Circuitry

The figure below shows the interconnections of the MPLAB ICD 5 In-Circuit Debugger to the ICSP connector on the target board. The diagram also shows the wiring from the connector to a device on the target PCB. A pull-up resistor (usually around 10-50 k Ω) 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 3-20. Standard Connection to Target Circuitry

4. Operation

A simplified theory of operation of the MPLAB ICD 5 In-Circuit Debugger system is provided here. It is intended to provide enough information so that a target board can be designed that is compatible with the debugger for both debugging and programming operations. The basic theory of in-circuit debugging and programming is discussed so that problems, if encountered, are quickly resolved.

4.1 MPLAB X IDE Debugging

In the Project Properties window, set up debugging, programming or other options. See

[9.2. Debugger Options Selection](#). Then debug your project .

For details on how to debug an application with MPLAB X IDE, see the user's guide on the [MPLAB X IDE webpage](#) or the WebHelp version on onlinedocs.microchip.com/.

4.2 SAM and PIC32C Arm Devices - On-Chip Debugging

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

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

Table 4-1. Cortex-M Debug and Trace Support Summary

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

4.3 AVR Devices - On-Chip Debugging (OCD)

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

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

Run Mode

When in Run mode, the execution of code is completely independent of the MPLAB ICD 5. The MPLAB ICD 5 will continuously monitor the target device to see if a break condition has occurred. When this happens, the OCD system will interrogate the device through its debug interface, allowing the user to view the internal state of the device.

Stopped Mode

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

Hardware Breakpoints

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

Software Breakpoints

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

4.3.1 AVR Device Interfaces

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

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

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

Table 4-2. Programming and Debugging Interfaces Summary

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

4.3.1.1 AVR E/D OCD - Features

The AVR E/D OCD is based on the UPDI physical interface, which is a single pin programming and debugging interface. Other features include:

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

4.3.1.2 tinyAVR OCD Features

The tinyAVR OCD for new devices is based on the UPDI physical interface, which is a single pin programming and debugging interface. It supports the following features:

- Memory-mapped access to device address space (NVM, RAM, I/O)
- No limitation on the device clock frequency
- Unlimited number of user program breakpoints
- Two hardware breakpoints
- Support for advanced OCD features
- Nonintrusive run-time chip monitoring without accessing the system registers
- Interface for reading the result of the CRC check of the Flash on a locked device

The tinyAVR OCD for older devices is based on debugWIRE. For more on OCD features, see [4.3.1.5. debugWIRE OCD Features](#).

4.3.1.2.1 TinyX-OCD (UPDI) Special Considerations

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

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

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

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

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

4.3.1.2.2 AVR devices with TPI

TPI (Tiny Programming Interface) is present on tinyAVR devices which have no OCD. Debugging of these devices is not possible - TPI is for programming only.

4.3.1.3 megaAVR OCD Features

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

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

- Four program memory (hardware) breakpoints (one is reserved)
- Hardware breakpoints can be combined to form data breakpoints
- Unlimited number of program breakpoints (using `BREAK`) (except ATmega128[A])

4.3.1.3.1 megaAVR® Special Considerations

Software Breakpoints

Since it contains an early version of the OCD module, ATmega128[A] does not support the use of the `BREAK` instruction for software breakpoints.

JTAG Clock

The target clock frequency must be accurately specified in the software front-end before starting a debug session. For synchronization reasons, the JTAG TCK signal must be less than one-fourth of the target clock frequency for reliable debugging. When programming using the JTAG interface, the TCK frequency is limited by the maximum frequency rating of the target device and not the actual clock frequency being used.

When using the internal RC oscillator, be aware that the frequency may vary from device to device and is affected by temperature and V_{CC} changes. Be conservative when specifying the target clock frequency.

OCDEN Fuse

To be able to debug a megaAVR device, the OCDEN fuse must be programmed (by default, OCDEN is unprogrammed). This allows access to the OCD to facilitate debugging the device. The software front-end will always ensure that the OCDEN fuse is programmed when starting a debug session and is left unprogrammed when terminating the session, thereby restricting unnecessary power consumption by the OCD module.

JTAGEN Fuse

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



Important: If the JTAGEN fuse is unintentionally disabled, it can only be re-enabled using SPI or High Voltage programming methods.

If the JTAGEN fuse is programmed, the JTAG interface can still be disabled in firmware by setting the JTAG disable bit in the MCU Control Register. This will render code un-debuggable and should not be done when attempting a debug session. If such code is already executing on the Microchip AVR device when starting a debug session, the MPLAB ICD 5 will assert the $\overline{\text{RESET}}$ line while connecting. If this line is wired correctly, it will force the target AVR device into Reset, thereby allowing a JTAG connection.

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



Tip:

Selecting the “use external reset” checkbox, in both the programming dialog and debug options dialog in Microchip Studio, allows the MPLAB ICD 5 to assert the $\overline{\text{RESET}}$ line. This also re-enables the JTAG interface on devices, which are running code that disables the JTAG interface by setting the JTAG disable bit.

IDR/OCDR Events

The IDR (In-out Data Register) is also known as the OCDR (On-Chip Debug Register) and is used extensively by the debugger to read and write information to the MCU when in Stopped mode during a debug session. When the application program in Run mode writes a byte of data to the OCDR register of the AVR device being debugged, the MPLAB ICD 5 reads this value out and displays it in the message window of the software front-end. The OCDR register is polled every 50 ms, so writing to it at a higher frequency will NOT yield reliable results. When the AVR device loses power while being debugged, spurious OCDR events may be reported. This happens because the MPLAB ICD 5 may still poll the device as the target voltage drops below the AVR's minimum operating voltage.

4.3.1.4 AVR XMEGA OCD Features

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

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

Note: For the ATxmegaA1 family, only revision G or later is supported.

4.3.1.4.1 AVR® XMEGA® Special Considerations

OCD and Clocking

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

I/O Modules in Stopped Mode

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

Hardware Breakpoints

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

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

Here are the different combinations that can be set:

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

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

JTAGEN Fuse

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



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

If the JTAGEN fuse is programmed, the JTAG interface can still be disabled in firmware by setting the JTAG disable bit in the MCU Control Register. This will render code un-debuggable and should not be done when attempting a debug session. If such code is already executing on the Microchip AVR device when starting a debug session, the MPLAB ICD 5 will assert the $\overline{\text{RESET}}$ line while connecting. If this line is wired correctly, it will force the target AVR device into Reset, thereby allowing a JTAG connection.

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



Tip:

Selecting the “use external reset” checkbox, in both the programming dialog and debug options dialog in Microchip Studio, allows the MPLAB ICD 5 to assert the $\overline{\text{RESET}}$ line. This also re-enables the JTAG interface on devices, which are running code that disables the JTAG interface by setting the JTAG disable bit.

Debugging with Sleep for ATxmegaA1 rev H and Earlier

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

- Go into the MPLAB ICD 5. Options in the Tools menu and enable “Always activate external Reset when reprogramming device.”
- Perform a chip erase.

The sleep modes that trigger this bug are:

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

4.3.1.5 debugWIRE OCD Features

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

- Complete program flow control
- Full access to all registers and memory areas
- Unlimited user program breakpoints (using `BREAK` instruction)
- Automatic baud rate configuration based on target clock

4.3.1.5.1 debugWIRE Special Considerations

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

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

- Either:** Attempt to start a debug session on the debugWIRE part. If the debugWIRE interface is not enabled, the software front-end will offer to retry or attempt to enable debugWIRE using SPI programming. If you have the full SPI header connected, debugWIRE will be enabled and you will be asked to toggle power on the target. This is required for the fuse changes to be effective.
- Or:** Open the programming dialog in Microchip Studio in SPI mode and verify that the signature matches the correct device. Check the DWEN fuse to enable debugWIRE.



Important:

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

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



Important:

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

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

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

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

- Pull-up resistors on the $\overline{dW}/(\overline{RESET})$ line must not be smaller than 10 kΩ. The pull-up resistor is not required for debugWIRE functionality since the debugger tool provides this.
- Any stabilizing capacitor connected to the $\overline{RESET}$ pin must be disconnected when using debugWIRE since they will interfere with correct operation of the interface.
- All external Reset sources or other active drivers on the $\overline{RESET}$ line must be disconnected, since they may interfere with the correct operation of the interface.

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

4.3.1.5.2 debugWIRE Software Breakpoints

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

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

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

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

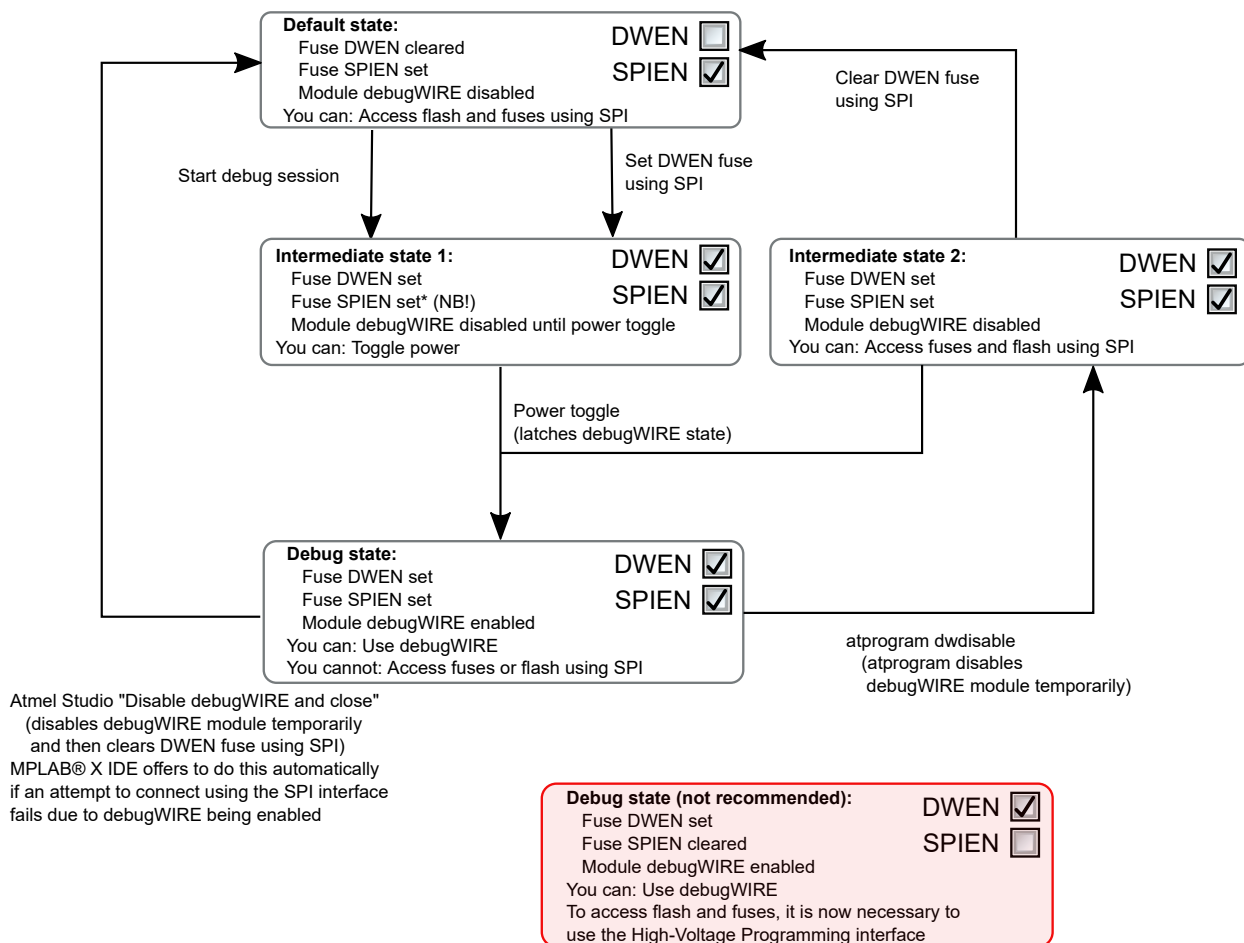
4.3.1.5.3 Understanding debugWIRE and the DWEN Fuse

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

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



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

Figure 4-1. Understanding debugWIRE and the DWEN Fuse

4.3.1.6 Advanced Debugging (AVR® JTAG/debugWIRE devices)

I/O Peripherals

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

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

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

Single Stepping I/O Access

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

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

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

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

Single Stepping and Timing

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

Accessing 16-Bit Registers

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

Restricted I/O Register Access

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

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

4.3.2 PIC32M MCU - On-Chip Debugging

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

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

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

4.4 PIC MCU/dsPIC DSC - On-Chip Debugging

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

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

4.4.1 Basic Debug Features

MPLAB® ICD 5 In-Circuit Debugger has the following basic debug features.

4.4.1.1 Start and Stop Emulation

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

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

4.4.1.2 View Processor Memory and Files

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

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

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

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

4.4.1.3 Use Breakpoints

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

4.4.1.3.1 Breakpoint Resources

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

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

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

See MPLAB X IDE Help>Help Contents>Hardware Tool Reference>Limitations - Emulators and Debuggers for limitations on breakpoint operation, including the general number of hardware breakpoints per device and hardware breakpoint skidding amounts.

4.4.1.3.2 Hardware or Software Breakpoint Selection

To select hardware or software breakpoints:

1. Select your project in the Projects window and then right click to select "Properties."
2. In Project Properties, select *ICD 5>Debug Options*.
3. Check "Use software breakpoints" to use software breakpoints. Uncheck to use hardware breakpoints.

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

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

Table 4-3. Hardware vs. Software Breakpoints

Feature	Hardware Breakpoints	Software Breakpoints
Number of breakpoints	Limited	Unlimited
Breakpoints written to*	Internal debug registers	Flash Program Memory
Breakpoints applied to**	Program Memory/Data Memory	Program Memory only
Time to set breakpoints	Minimal	Dependent on oscillator speed, time to program Flash Memory and page size.
Breakpoint skidding	Most devices. See MPLAB X IDE <i>Help>Help Contents>Hardware Tool Reference>Limitations - Emulators and Debuggers</i> .	No
* Where information about the breakpoint is written in the device.		
** What kind of device feature applies to the breakpoint. This is where the breakpoint is set.		

4.4.1.4 Use the Stopwatch

Use the stopwatch to determine the timing between two breakpoints.

Note: The stopwatch uses breakpoint resources.

To use the Stopwatch:

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

Figure 4-2. Stopwatch Setup

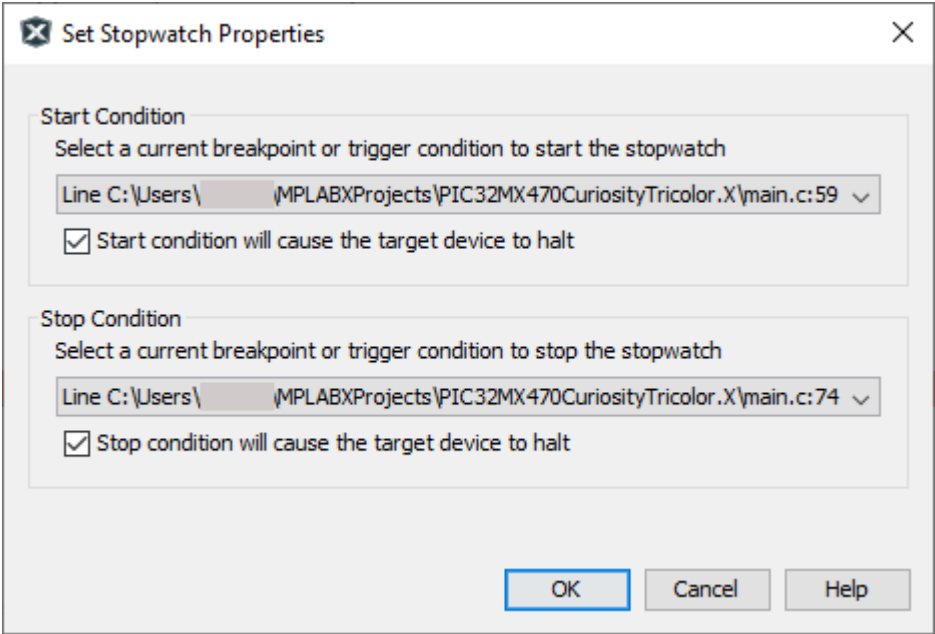
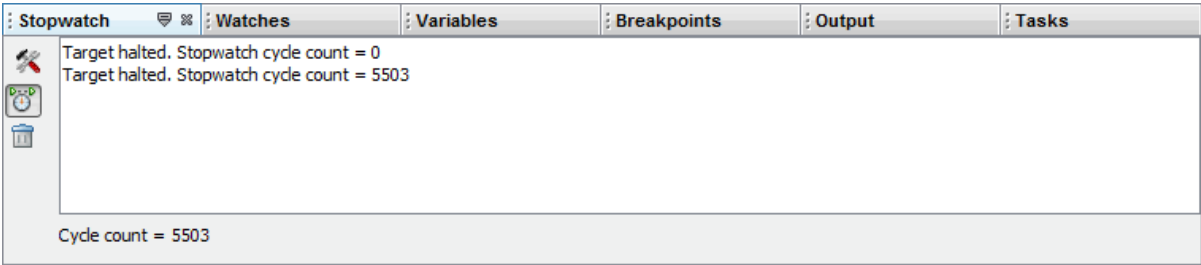


Figure 4-3. Stopwatch Window with Content



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

Table 4-4. Stopwatch Icons

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

4.4.1.5 Set Freeze Peripherals

For some devices and tools, you can select a “Freeze on Halt” option, which allows you to freeze/unfreeze selected peripherals on a halt.

4.4.2 ICSP Debugging

There are two steps to using MPLAB ICD 5 In-Circuit Debugger as a debugger. The first requires that an application is programmed into the target device (MPLAB ICD 5 can be used for this). The second

uses the internal in-circuit debug hardware of the target Flash device to run and test the application program. These two steps are directly related to MPLAB X IDE operations:

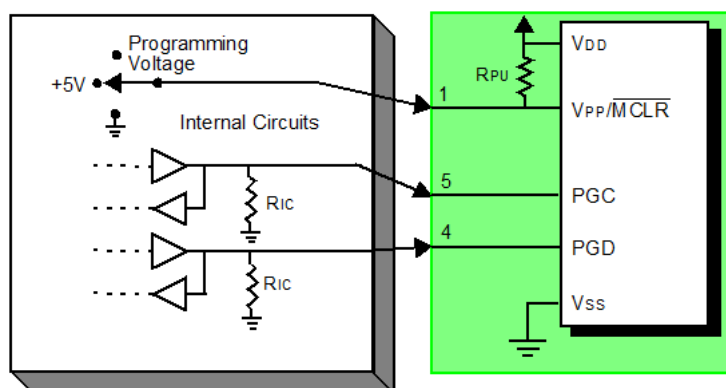
1. Programming the code into the target and activating special debug functions (see the next section for details).
2. Using the debugger to set breakpoints and run.

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

If the target device cannot be programmed correctly, the MPLAB ICD 5 will not be able to debug it.

A simplified diagram of some of the internal interface circuitry of the MPLAB ICD 5 is shown in the figure below. In the figure, $R_{pu}=10\text{ k}\Omega$ typical and $R_{ic}=4.7\text{ k}\Omega$.

Figure 4-4. Proper Connections for ICSP Programming

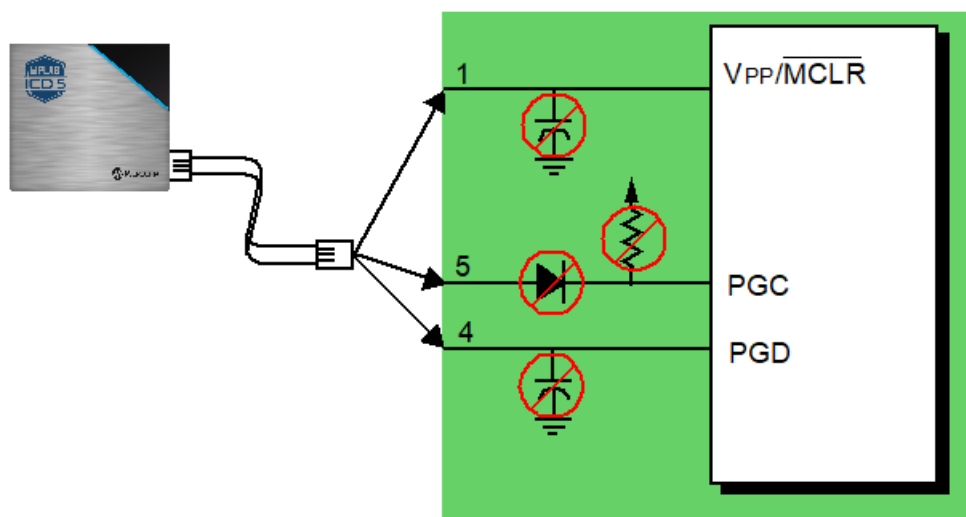


For programming, no clock is needed on the target device, but power must be supplied. When programming, the debugger puts programming levels on VPP/MCLR, sends clock pulses on PGC, and serial data via PGD. To verify that the part has been programmed correctly, clocks are sent to PGC and data is read back from PGD. This sequence confirms the debugger and device are communicating correctly.

4.4.2.1 ICSP Circuits That Will Prevent a Debug Tool From Functioning

The figure below shows the active debugger lines with some components that will prevent the MPLAB ICD 5 In-Circuit Debugger from functioning.

Figure 4-5. Improper Circuit Components



In particular, these guidelines must be followed:

- **Do not use pull-ups on PGC/PGD** – they could disrupt the voltage levels.
- **Do not use capacitors on PGC/PGD** – they will prevent fast transitions on data and clock lines during programming and debugging communications, and slow programming times.
- **Do not use capacitors on $\overline{\text{MCLR}}$** – they will prevent fast transitions of VPP. A simple pull-up resistor is generally sufficient.
- **Do not use diodes on PGC/PGD** – they will prevent bidirectional communication between the debugger and the target device.

4.4.2.2 Sequence of Operations Leading to Debugging

Given that the [4.4.2.4. Requirements for Debugging](#) are met, set the MPLAB ICD 5 In-Circuit Debugger as the current tool in MPLAB X IDE. Go to *File> Project Properties* to open the dialog and then under “Hardware Tool,” click “ICD 5.”

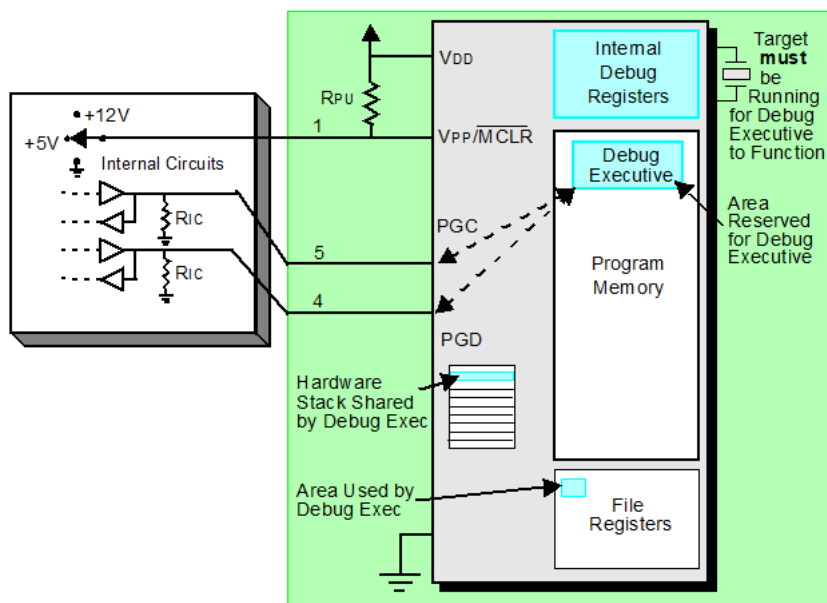
The following actions can now be performed:

- 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 memory of the target device. Since some architectures require that a debug executive must reside in program memory, the application program must not use this reserved space. Some devices have special memory areas dedicated to the debug executive. Check your device data sheet for details.
- Special “in-circuit debug” registers in the target device are enabled by MPLAB X IDE. These allow the debug executive to be activated by the debugger. For more information on the device’s reserved resources, see [4.4.2.5. Resources Used by the Debugger](#).
- The target device is run in Debug mode.

4.4.2.3 Debugging Details

The figure below illustrates the MPLAB ICD 5 In-Circuit Debugger system when it is ready to begin debugging. In the figure, $R_{pu}=10\text{ k}\Omega$ typical and $R_{ic}=4.7\text{ k}\Omega$.

Note: There are programmable pullups and pulldowns instead of fixed resistor.

Figure 4-6. MPLAB ICD 5 Ready to Begin Debugging - PIC MCU

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. If using ICSP only, 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 instructs the debug executive to run the target application. The target starts from the Reset vector and executes until the Program Counter reaches the breakpoint address that was stored previously in the internal debug registers.

After the instruction at the breakpoint address is executed, the in-circuit debug mechanism of the target device “fires” and transfers the device’s program counter to the debug executive (like an interrupt) and the user’s application is effectively halted. The debugger communicates with the debug executive via PGC and PGD, gets the breakpoint status information, and sends it back to MPLAB X IDE. MPLAB X IDE then sends a series of queries to the debugger to get information about the target device, i.e., file register contents and the state of the CPU. These queries are performed by the debug executive.

The debug executive runs like an application in program memory or special test 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 ICD 5, and MPLAB X IDE will issue an error message.

Another way to halt the target 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. The target application program is effectively halted, and MPLAB X IDE uses the debugger communications with the debug executive to interrogate the state of the target device.

4.4.2.4 Requirements for Debugging

To debug (set breakpoints, see registers, etc.) with the MPLAB ICD 5 In-Circuit Debugger system, there are critical elements that must be working correctly:

- The debugger must be powered, must be connected to a computer, and must be communicating with the MPLAB X IDE software.
- The target device must have power and a functional, running oscillator. If for any reason the target device does not run, the MPLAB ICD 5 In-Circuit Debugger will not be able to debug it.
- The target device must have its Configuration words programmed correctly. These may be set using code or the Configuration Bits window in MPLAB X IDE.
 - The oscillator Configuration bits should correspond to oscillator types available on the target.
 - For some devices, the Watchdog Timer is enabled by default and needs to be disabled.
 - The target device must not have any type of code protection enabled.
 - The target device must not have table read protection enabled.
- For some devices with more than one PGC/PGD pair, the correct pair needs to be selected in the device's configuration word settings. This only refers to debugging, since programming will work through any PGC/PGD pair.

4.4.2.5 Resources Used by the Debugger

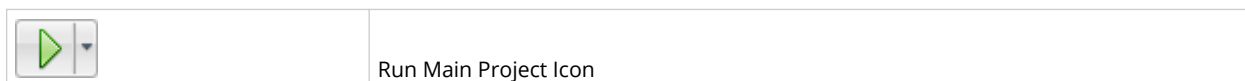
For some devices, device resources must be used for debug. For a complete list of resources used by the debugger for your device, in MPLAB X IDE select *Help > Release Notes*. In addition to a section for "Release Notes/Readmes," there is a section for "Reserved Resources." Select either "Reserved Resources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

4.4.3 Programming

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

In the MPLAB X IDE, use the MPLAB ICD 5 as a programmer to program a non-ICE/-ICD device, that is, a device not on a header board. Set the MPLAB ICD 5 as the current tool (click the Debug Tool ICD 5 in the navigation window, then select *File > Project Properties* from the main menu to open the dialog, then under "Hardware Tool," click "ICD 5") to perform these actions:

- When *Run > 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.



- A small "program executive" program may be loaded into the high area of program memory for some target devices.
- Special "in-circuit debug" registers in the target device are disabled by MPLAB X IDE, along with all debug features. This means that a breakpoint cannot be set and register contents cannot be seen or altered.
- The target device is run in Release mode. As a programmer, the debugger can only toggle the MCLR line to Reset and start the target device.

5. Debugger Features

In addition to basic debug features, the debugger has advanced debug features. Some debug features are dependent on other debug features.

To see if a specific debug feature is available for your device:

- In MPLAB X IDE, on the Help menu select “Release Notes.”
- In addition to **Release Notes/Readmes**, find **Debug Features Support**.
- Click on the link for “Hardware Tool Debug Features by Device.”

In addition to features already mentioned, this table specifies additional debug features.

Table 5-1. Supported Debugger Features

Feature	Microcontroller Family*				
	PIC10/12/16 (8-bit)	PIC24, dsPIC (16-bit)	PIC32 (32-bit)	AVR (8-bit)	SAM (32-bit)
Virtual COM Port	X	X	X	X	X
DGI	X	X	X	X	X
Basic Debug ¹	X	X	X	X	X
SAM ITM/SWO Trace					X
PC Profiling			X		
Debugger Polling				X	X
Power Monitoring	X	X	X	X	X

* Not all devices in a family have support. See feature sections below for details.
 1. Run/halt single step and break point, read/write memory.

5.1 USB CDC Virtual COM Port

Provides a bridge between the target UART and the USB interface, which provides a CDC Virtual “COM” port on USB Host which is a read/write access to a true UART of target.

Use this virtual port to access the debugger when using containers for CI/CD.

CDC/DGI U(S)ART supported Baud rates : 7200, 9600, 14400, 19200, 28800, 38400, 57600, 76800, 115200, 230400, 460800, 921600.

5.2 Data Gateway Interface

The Data Gateway Interface (DGI) is an interface for handling the low-level transport of data from a target MCU. The DGI is available on a selection of tools and on-board debuggers.

The DGI provides several interfaces utilizing the same API for configuration and communication. Each interface implements an abstraction to a physical communication interface, such as UART, or represents a service not directly tied to a physical communication interface, such as the timestamp interface.

The MPLAB Data Visualizer is the application used to control and stream Data Gateway Interfaces. The application may be accessed from within MPLAB X IDE or as a standalone program. For more on this application and DGI, see the [MPLAB Data Visualizer webpage](#).

5.2.1 Interfaces

All functionality of the DGI is centered around the implemented interfaces. All interfaces use the same USB protocol, but every interface has its own configuration parameters and handling of communication. For details, refer to the interface-specific sections. Note that not all interfaces are

available on all boards implementing the DGI device. The available interfaces can be read through the USB protocol.

Table 5-2. List of Interfaces

Name	Identifier	Description
Timestamp	0x00	Service interface which appends timestamps to all received events on associated interfaces.
UART	0x21	Communicates directly over UART in Client mode.
Power	0x40 (data)	Receives data and sync events from the attached power measurement co-processors.
Debugger Polling	0x50	Polling of timestamped samples of the program counter, allowing an insight in the program execution of the device. For more information, see 5.6. Debugger Polling .
Reserved	0xFF	Special identifier used to indicate no interface.

5.2.1.1 Timestamp

The data returned over the timestamp interface is a sequential stream of timestamped packets of data belonging to the interfaces that have timestamping enabled. The first byte in each packet is the interface identifier and will decide how the rest of the packet must be parsed.

The timestamp is relying on a 16-bit timer, which is sampled and embedded into each packet. The timer tick frequency can be read from the timestamp configuration. It is in the area of about half a microsecond. When the timer overflows, a packet will be embedded in the stream to indicate this event. Note that if a data packet is being embedded as the timer overflows, an overflow packet will not be embedded. Instead, it will be indicated in the header of the data packet.

All timestamped packets are generated from module interrupts within the DGI device, which can not be interrupted by the timer overflow interrupt. This means that there is a possibility that the timer has overflowed before the timer was sampled and embedded. To be able to keep the timestamp in sync and accurate for such events the packets are also embedding the timer overflow bit. This bit is sampled after the timer itself, and can potentially be set even if the sampled timer value was in sync.

5.2.1.2 UART Interface

The UART **source** streams the raw values received on the UART interface.

On the Data Sources (left) pane, when the UART source is selected, the UART settings are displayed on the lower section.

Note: Asynchronous serial protocols (e.g., UART protocols used by DGI UART and CDC Virtual COM port interfaces) use the **baud rates** listed in [5.1. USB CDC Virtual COM Port](#).

Table 5-3. USART Settings

Field Name	Values	Usage
Baud Rate	0-2000000	Baud rate for UART interface in Asynchronous mode
Char Length	5, 6, 7, or 8 bits	Number of bits in each transfer
Parity	None, Even, Odd, Mark, or Space	Parity type used for communication
Stop bits	1, 1.5, or 2 bits	Number of Stop bits

5.2.1.3 Power Interface

The Power interface measures the power consumption of the connected circuitry.

Select the Power interface beneath the debug tool DGI. Set up the interface using the controls under "Power Settings."

Table 5-4. Power Settings Controls

Control	Value	Usage
Enabled Channels	A	Enable channel A only. Channel A is always enabled.

.....continued

Control	Value	Usage
Lock Channel A in high range*	Unchecked, checked	Channel A can be locked to the high range to avoid automatic switching to the low range. This allows detection of short spikes in current consumption without critical samples being lost when switching between the ranges.
Output Voltage in mV	Between 1600 mV and 5500 mV, or 0	The MPLAB ICD 5 features an adjustable target supply that can be used to power the target application. This setting enables and controls the output voltage of this supply. A selection of 0 disables the supply.

* Future feature.



Tip: Any setting changes will not take effect until clicking **Apply** in the Power Settings panel. E.g., to enable the Voltage Output, the Output Voltage value set and **Apply** must be clicked before the voltage output will actually be enabled and set accordingly.



Tip: The channel A range lock will not force the debugger to return to the high current range if already running in the low range. Either wait for a current high enough to force it to change, or simply Stop and Start the debugger.



Tip: Each power signal time plot uses system resources. Reduce the number of concurrent plots for better performance.

5.3 CI/CD Support

MPLAB® ICD 5 In-Circuit Debugger can be used as a hardware test tool for Continuous Integration / Continuous Delivery (or Deployment) because of its network communication capabilities. In MPLAB X IDE, use the CI/CD Wizard to create files for Jenkins-Docker or Docker-only integration. See the *CI/CD Wizard in MPLAB X IDE User's Guide* (DS-50003243) or the section in the *MPLAB X IDE User's Guide* (DS-50002027) for details on CI/CD and using the wizard.

If Jenkins Pipeline Files are selected for creation in the wizard, the debugger may be included using one of the wizard screens (see table below.)

Table 5-5. CI/CD Wizard - Hardware Testing

Option	Description
Enable Hardware Testing	Enable the debugger as a test tool.
Configuration to Build and Run on ICD 5	Select a project configuration that uses the debugger; either default or one dedicated to hardware use.
IP Address of ICD 5	Enter the IP address of the debugger you wish to use. If you do not know the IP address of the debugger, you can use <code>ipconfig</code> or a similar tool to search the system to which the debugger is connected for the address.
Enable MPLAB Code Coverage	Enable the MPLAB Code Coverage feature. An MPLAB Analysis Tool Suite license will be required to use this feature. Note: The Code Coverage API plugin must be available on your Jenkins server if coverage reporting is enabled.
Scan Output for Unity Test Results	Enable if the configuration builds Unity test runners and the build job should create a report based on the resulting output. For more info on how to write Unity tests see Unity - Getting Started .

5.4 ARM ITM/SWO Trace

ITM trace outputs UART-format data using the single-pin SWO. ITM has the provision to send this data over multiple ports available from 0 to 31. Details of ITM/SWO trace may be found in the following topics.

Not all SAM devices have ITM trace. See in MPLAB X IDE *Help>Release Notes> Debug Features Support>Hardware Tool Debug Features by Device*.

5.4.1 How ITM Trace Works

All SAM MCUs/MPUs that support ITM trace are Arm® Cortex®-M processor-based devices with [Arm® CoreSight® architecture](#). Currently ITM is supported in Cortex M3, M4, M33, and SC300 cores.

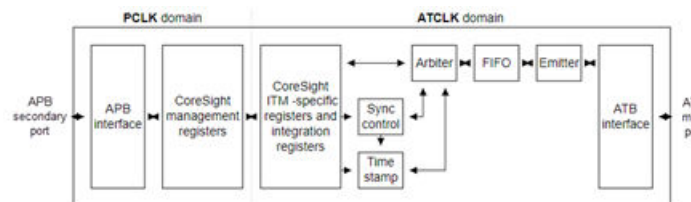
The CoreSight *Instrumentation Trace Macrocell* (ITM) block is a software application driven trace source. Supporting user code generates *SoftWare Instrumentation Trace* (SWIT). In addition, the block provides a coarse-grained timestamp functionality. The main uses for this block are to:

- support `printf` style debugging.
- trace OS and application events.
- emit diagnostic system information.

MPLAB ICD 5 is capable of streaming UART-format ITM trace to the host computer. Trace is captured on the TRACE/SWO pin of the 10-pin header (JTAG TDO pin). Data is buffered internally on the MPLAB ICD 5 and is sent over the trace interface to the host computer.

Note: You can set the ITM baud rate in MPLAB X IDE. See [5.4.5.4. Setup ITM Trace](#).

Figure 5-1. ITM Block Diagram



The ITM contains the following sub-blocks:

Timestamp	Generates timestamp packet.
Sync control	ITM synchronizer.
Arbiter	Arbitrates between synchronous, timestamp, and SWIT packet.
FIFO	ATB <i>First In First Out</i> (FIFO).
Emitter	ATB registered emitter.

Data is written to the stimulus registers using the APB interface. This data is then transmitted on the ATB interface as SWIT packets.

5.4.2 How SWO Works with ITM Trace

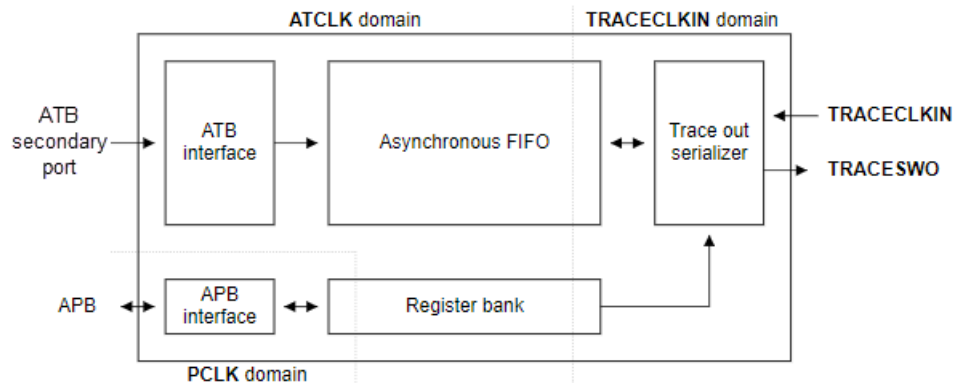
The CoreSight *Serial Wire Output* (SWO) is a trace data drain that acts as a bridge between the on-chip trace data to a data stream that is captured by a *Trace Port Analyzer* (TPA) which is the MPLAB ICD 5 In-Circuit Debugger.

Compared to the *Trace Port Interface Unit* (TPIU), the SWO contains:

- no formatter.

- no pattern generator.
- an 8-bit ATB input.
- no synchronous trace output, that is, no **TRACEDATA**, **TRACECTL**, or **TRACECLK** pins.
- no support for flush, because this is not required.
- no support for triggering.
- no external inputs and outputs (**EXTCTLIN** and **EXTCTLOUT** are not implemented).

Figure 5-2. SWO Block Diagram



5.4.3 Requirements for ITM/SWO Trace

The following is required to use trace for SAM devices:

- MPLAB X IDE v6.10 or greater.
- A target board with debug and trace connections to a device that supports SWO trace.
- For debug and trace support, use one of the following:
 - [MPLAB ICD 5 Cortex-M Trace Adapter Board](#)
 - [ICD 5 JTAG Adapter Board](#) (**Note:** SWO pin multiplexed with JTAG pins)
 - a target board with a connector to the 8-pin flat cable

5.4.4 Hardware Setup

Preliminaries

1. Use USB communication between the PC and MPLAB ICD 5. Other communication types do not support trace.
2. Find devices that support ITM trace – see [Help>Release Notes> Debug Features Support>Hardware Tool Debug Features by Device](#).
3. Design the target board to have a connector for debugger-target communication and for trace pins if using SAM [3.3.3. Debugger Adapter Board](#). Alternately design the target board to connect to the cable with pins for both debug and trace.
4. When using trace, the TRACESWO pin on the target is used. Therefore you cannot use another function multiplexed on that pin.

Set Up Hardware

To use the ITM/SWO feature:

1. The target board should be unpowered.
2. Install the communication cable between the debugger or adapter board and the communication connector on your target board.

3. If using the adapter board, connect the trace cable between the adapter board and trace connector on your target board.
4. Power the target.

5.4.5 Set Up ITM in MPLAB X IDE

An MPLAB X IDE project is needed for code development and debugging. Then project can be configured for debug tool feature support.

5.4.5.1 Create a Project in MPLAB X IDE

Create a project in MPLAB X IDE while being mindful of the [5.4.3. Requirements for ITM/SWO Trace](#). For more on creating a project, consult [MPLAB X IDE documentation](#).

To assist with code development, consider using [MCC MPLAB Harmony](#):

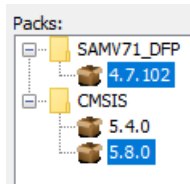
1. Select *Tools>Plugins>Available Plugins>MPLAB Code Configurator* and click **Install**. MPLAB X IDE will need to restart.



2. Click on the  toolbar icon. Please wait for MCC to initialize and install.
3. On the **Content Manager** tab, click to **Select MPLAB Harmony**.
4. Ensure Required Content is downloaded and select any Optional Content needed for your application. Then click **Finish**.
5. Edit `main()` and other files as necessary to create your application.

5.4.5.2 Use CMSIS ITM Functions

Right click on the project name in the Projects window and select "Properties." Under Packs, in addition to the device pack (DFP), CMSIS is also included (see figure below).



The Common Microcontroller Software Interface Standard (CMSIS) is a vendor-independent abstraction layer for microcontrollers that are based on Arm Cortex processors. CMSIS defines generic tool interfaces and enables consistent device support. The CMSIS software interfaces simplify software reuse, reduce the learning curve for microcontroller developers, and improve time to market for new devices.

The ARM CMSIS package comes with specific header files/APIs for sending ITM data. For example, in `<MPLAB_Installation>packs\arm\CMSIS\x.x.x\CMSIS\Core\Include\core_cm7.h`, find the CMSIS function `ITM_SendChar()` that can be used to print a character over ITM/SWO:

```
__STATIC_INLINE uint32_t ITM_SendChar(uint32_t ch)
```

These are the lowest level APIs at the byte level (only writing to PORT 0); however you can customize and develop your own functions for printing the messages to any port 0 through 31 (see [5.4. ARM ITM/SWO Trace](#)).

```
__STATIC_INLINE uint32_t ITM_SendCharPort (uint8_t port, uint32_t ch)
{
    if (((ITM->TCR & ITM_TCR_ITMENA_Msk) != 0UL) &&          /* ITM enabled */
        ((ITM->TER & 1UL << port) != 0UL))                    /* ITM Port enabled */
    {
        while (ITM->PORT[port].u32 == 0UL)
        {

```

```

    __NOP();
}
ITM->PORT[port].u8 = (uint8_t)ch;
}
return (ch);
}

//--SL: same as ITM-fct from CMSIS-header (see above), but with portNum
void ITM_PrintString(const char *s, uint8_t portNo)
{
    while (*s!='\0')
    {
        ITM_SendCharPort(portNo, *s++);
    }
}

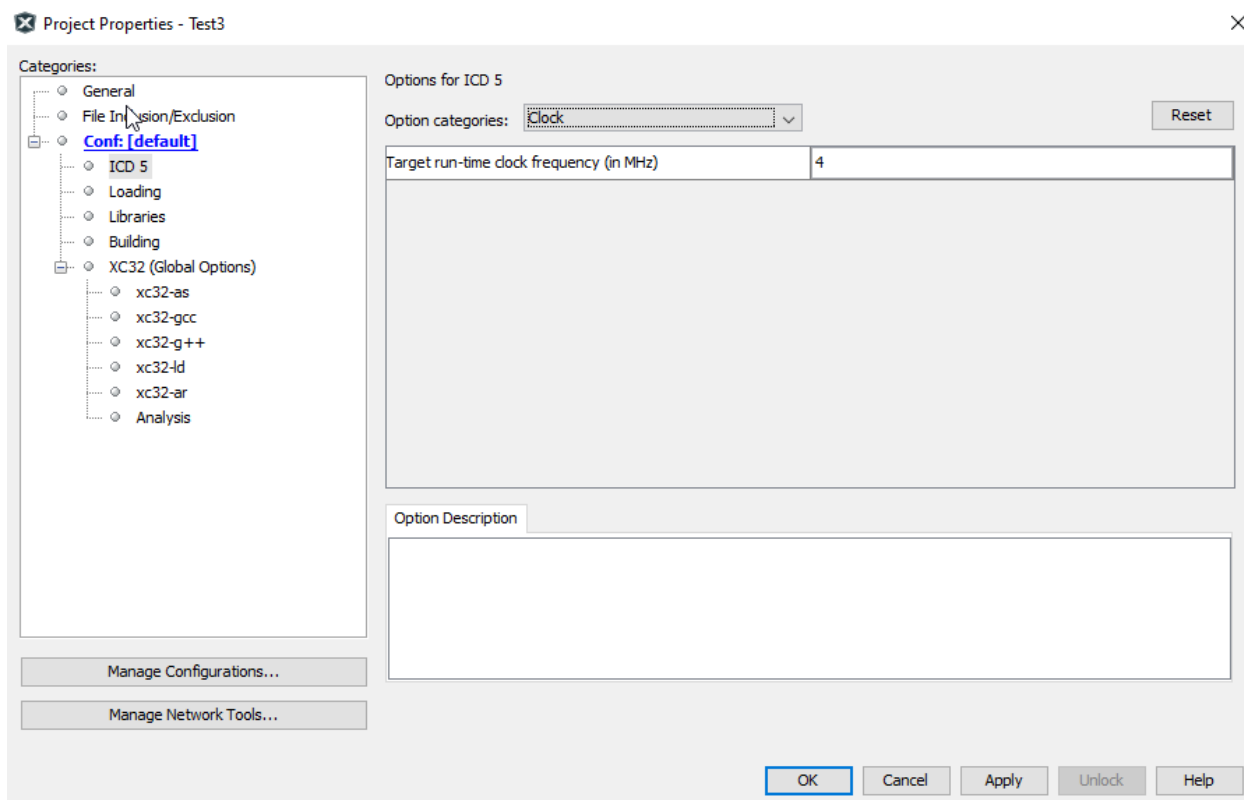
```

Alternately, if you do not want to use CMSIS ITM functions, you can write your own by understanding the ITM PORT registers and ITM configuration/status registers which are available as part of the [ARM CoreSight Documentation](#).

5.4.5.3 Setup the Clock

In the Project Properties dialog, click on “ICD 5” (under “Categories”). Select “Clock” from the drop down and enter the “Target run-time clock frequency (in MHZ)” in “Option categories.”

Figure 5-3. Setup the Clock Test



Note: This does not set the clock but informs the debugger of its value for runtime watch, data capture and trace.

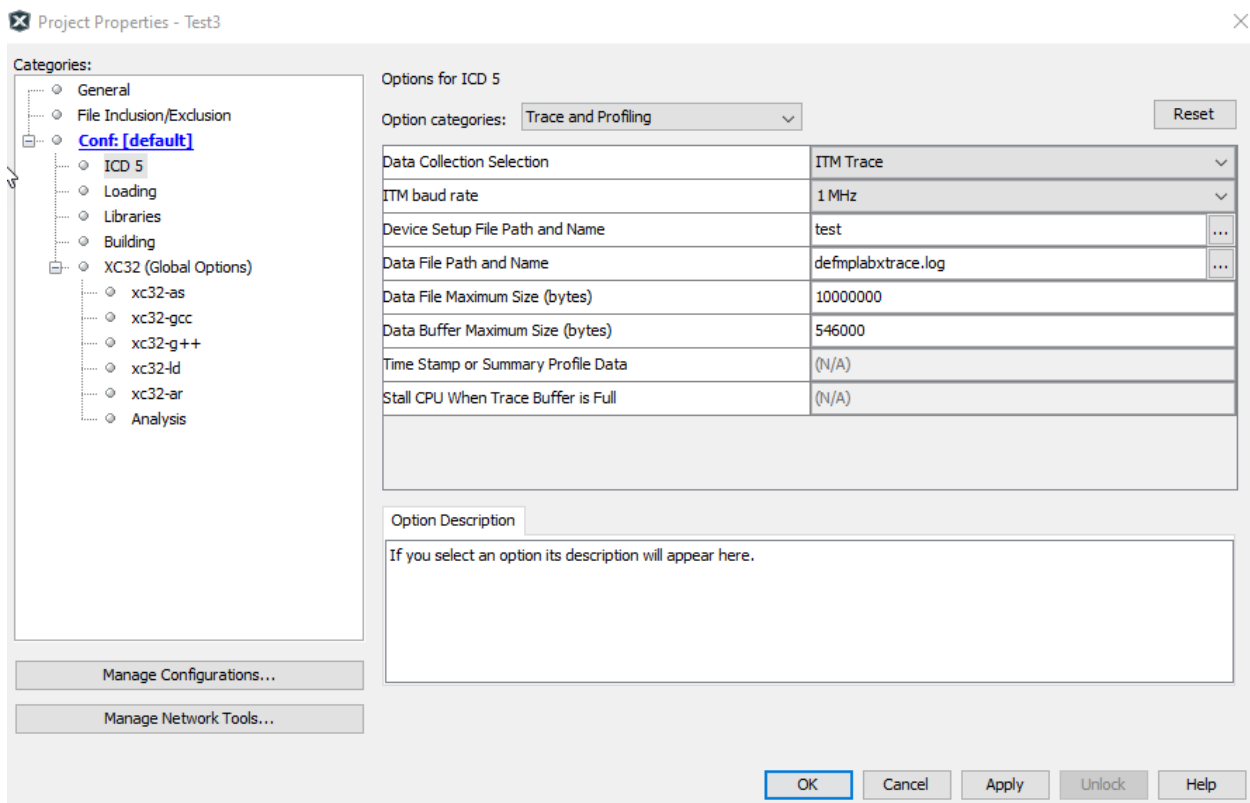
5.4.5.4 Setup ITM Trace

Select “Trace and Profiling” from “Option categories” drop down,

1. Select “ITM TraceUnder” from the “Data Collection Selection” drop down.

2. Select an “ITM baud rate” to specify the SWO speed. The clock and this value will be used to determine the SWO prescaler value.
Note: The debugger has a finite set of SWO baud rates that it can use: 512KHz, 1MHz, 2MHz, and 4MHz.
Note: It is advised that if clock switching is involved in the application, the SWO baud rate is setup for the intended clock rate at which SWO functionality is desired.
3. For some SAM devices, you will need to add a .ini file for additional ITM setup. See [5.4.5.5. Additional Initialization File](#).
4. Change any logging setup as desired.
5. Click **OK** when done.

Figure 5-4. Trace and Profiling



5.4.5.5 Additional Initialization File

SAM E70 and SAM E54 device families have clocking (PLL) that differs from standard Arm devices. Therefore specific device configuration is required. This is done using an `ini` file. This file will be launched on a reset.

useroperationsITMSAME70.ini

```
; Trace Clock Setup
; _WDWORD (0x400E064C, 0x4); // Select Master clock for ITM/ETM
write,0x400E064C, 4

;PMC->PMC_SCER = PMC_SCER_PCK3; // Enable PCK3
write,0x400E0600,0x800
```

useroperationsITMSAME54.ini

```
; Trace Clock Setup
; Enable ITM/ETM Peripheral Generic clock and set it to Master Clock
write,0x40001D3C, 0x40

; Configure PB30 to SWO - GPIO PORT MUX
write,0x410080BF,0x07

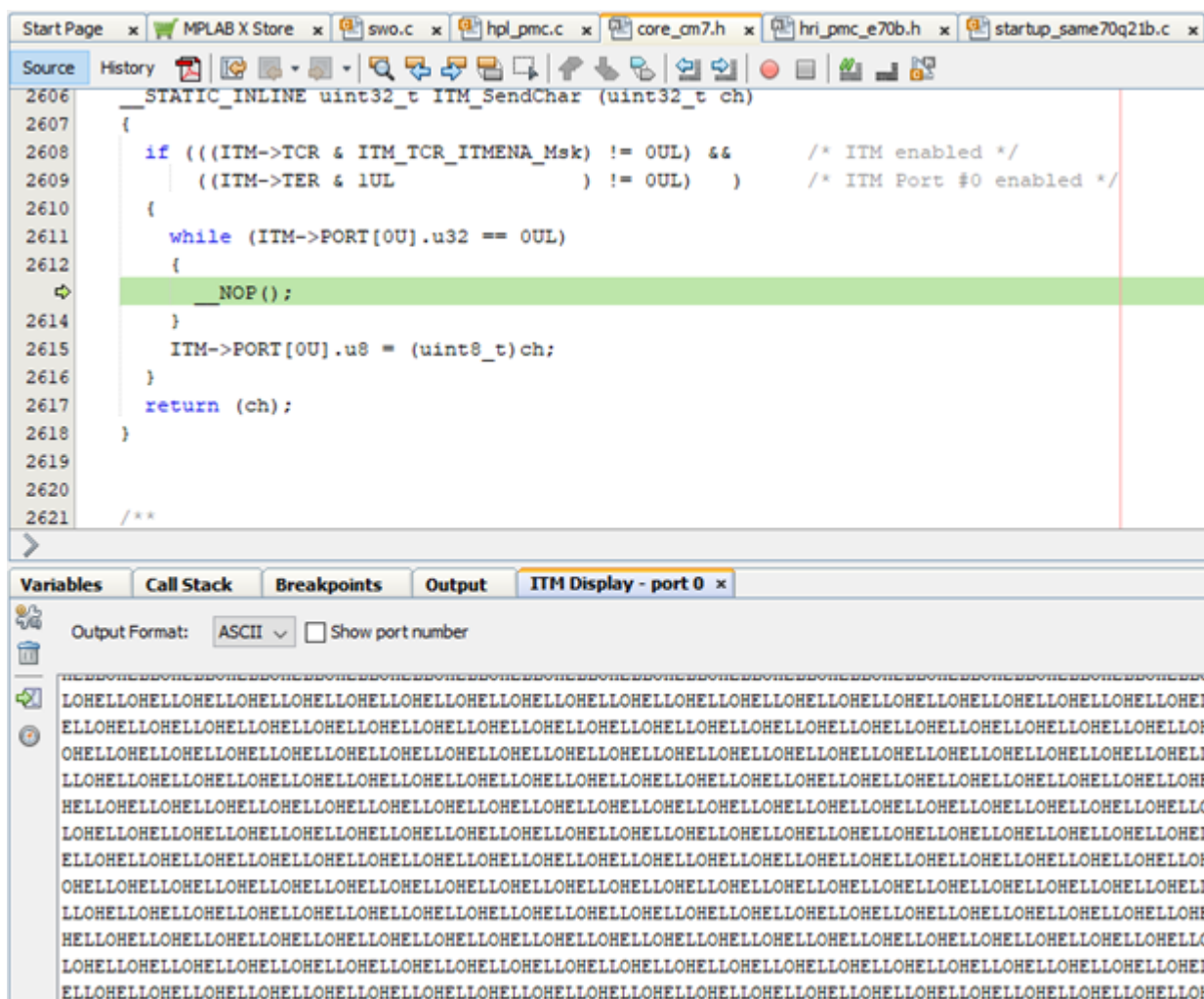
; Configure Pullups for PB30
write,0x410080DE,0x41
```

5.4.6 Viewing ITM Data

On a Debug Run, trace will continue to fill the trace buffer with data, rolling over when the buffer is full or when a Halt is executed. The application will determine how the trace data is used or displayed.

Note: There can be a delay in the display of data in the ITM Display window until there is enough data to fill the buffer. Some data can be lost during processing.

Figure 5-5. Example Output in ITM Display



Related Links

[9.3.2. ITM Window and Related Dialogs](#)

5.5 SAM (ARM) - Trace and Profiling

The SAM D5x/E5x Cortex-M4 processors implements a complete hardware debug solution. This provides high system visibility of the processor and memory through a 2-pin Serial Wire Debug (SWD) port that is ideal for microcontrollers and other small package devices.

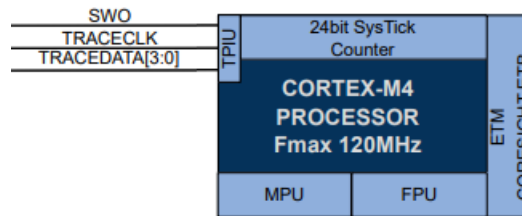
5.5.1 ARM Cortex-M4 Processor - Trace and Profiling

For system trace the processor integrates an Instrumentation Trace Macrocell (ITM) alongside data watchpoints and a profiling unit.

To enable simple and cost-effective profiling of the system events these generate, a stream of software-generated messages, data trace, and profiling information is exported over two different ways:

- Output off chip using the TPIU - through a single pin, called Serial Wire Viewer (SWV). Limited to ITM system trace.
- Internally stored in RAM - using the CoreSight ETB. Bandwidth is then optimal but capacity is limited.

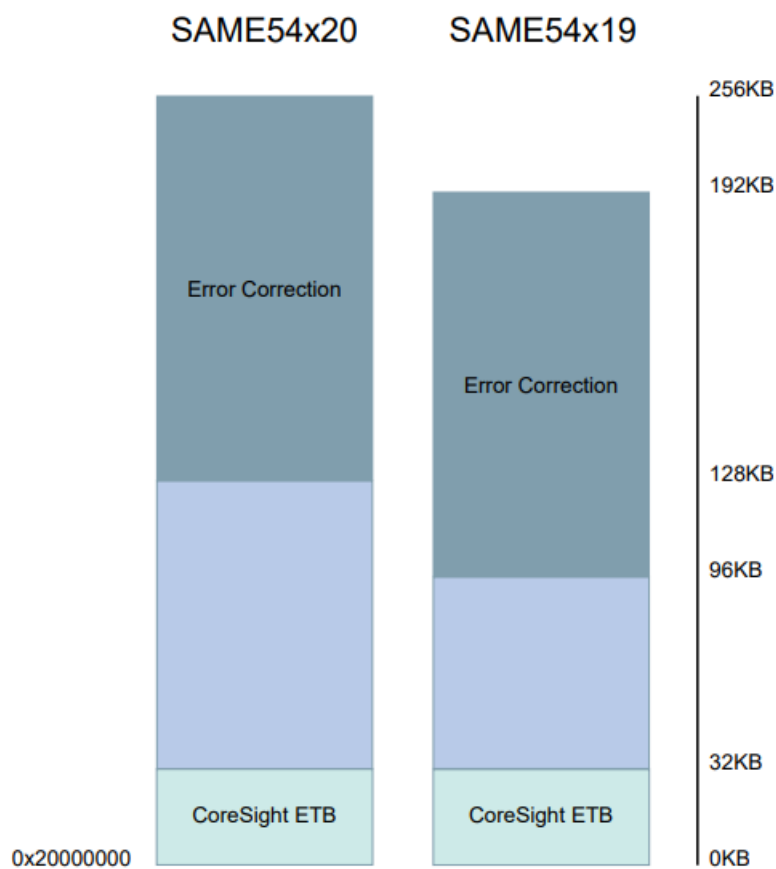
Figure 5-6. Block Diagram



SWV trace data are output on the Serial Wire Output (SWO).

5.5.2 SAM D5x/E5x - ETB Connection

When enabled, the bottom 32 KB system memory space is reserved for CoreSight ETB debug usage. The figure below shows an example where both error correction codes (ECC) and CoreSight ETB are enabled.

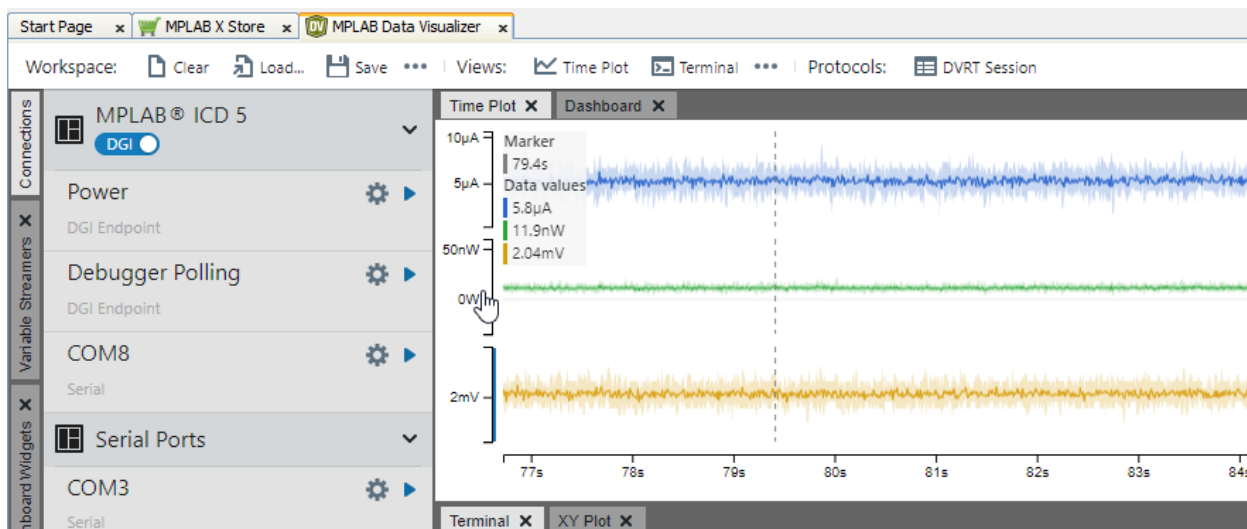
Figure 5-7. Memory with ECC and CoreSight ETB

5.6 Debugger Polling

MPLAB ICD 5 can be instructed by MPLAB Data Visualizer to repeatedly poll the Program Counter (PC) as fast as possible during active debug session with a target device. Though this will not yield a very high percentage of PC sampling or code trace, it can be useful in Code vs Power correlation to trap the areas of code which use more power than intended. As an example, see the figure below.

For more on the MPLAB Data Visualizer, see [MPLAB Data Visualizer webpage](#).

Figure 5-8. Debugger Polling



5.6.1 Requirements

Currently, to use Debugger Polling your project must be set up for these supported devices:

- AVR-8bit devices (UPDI interface)
- ARM-32bit devices (SWD interface)

5.6.2 Operation

The Debugger polling uses the SWD interface of SAM 32-bit devices and UPDI interface of AVR 8-bit devices to access the internal program counter location. It provides timestamped samples of the program counter address, allowing an insight in the program execution of the device.

Note: Debugger polling is only available when MPLAB Data Visualizer  is run from within MPLAB X IDE. This allows the data visualizer access the debug system on the device through the MPLAB X IDE backend.

Note: Debugger polling requires that the debugger is running, i.e., select “Debug Project” in MPLAB X IDE.

5.7 Power Monitor

The MPLAB ICD 5 In-Circuit Debugger can be a power monitor. Power monitoring means capturing power data, such as current values. When power is supplied by the debugger, the Vdd and GND lines can be monitored and power data gathered.

Power Monitoring is available for the following Microchip devices: PIC®, dsPIC®, AVR® and SAM MCU.

DGI (data gateway interface) Power Monitoring is limited to debuggers/kits that support the DGI and have the additional circuitry for power measurement, which includes the ICD 5.

The debugger works with the MPLAB Data Visualizer to provide plots of power data. As of MPLAB X IDE v5.50, the MPLAB DV plugin is included with the IDE. A separate stand-alone version is also available.

Note: You can only use USB communication with the data visualizer.

For more on this software, see the [MPLAB Data Visualizer Product Page](#).

5.7.1 About Power Monitoring

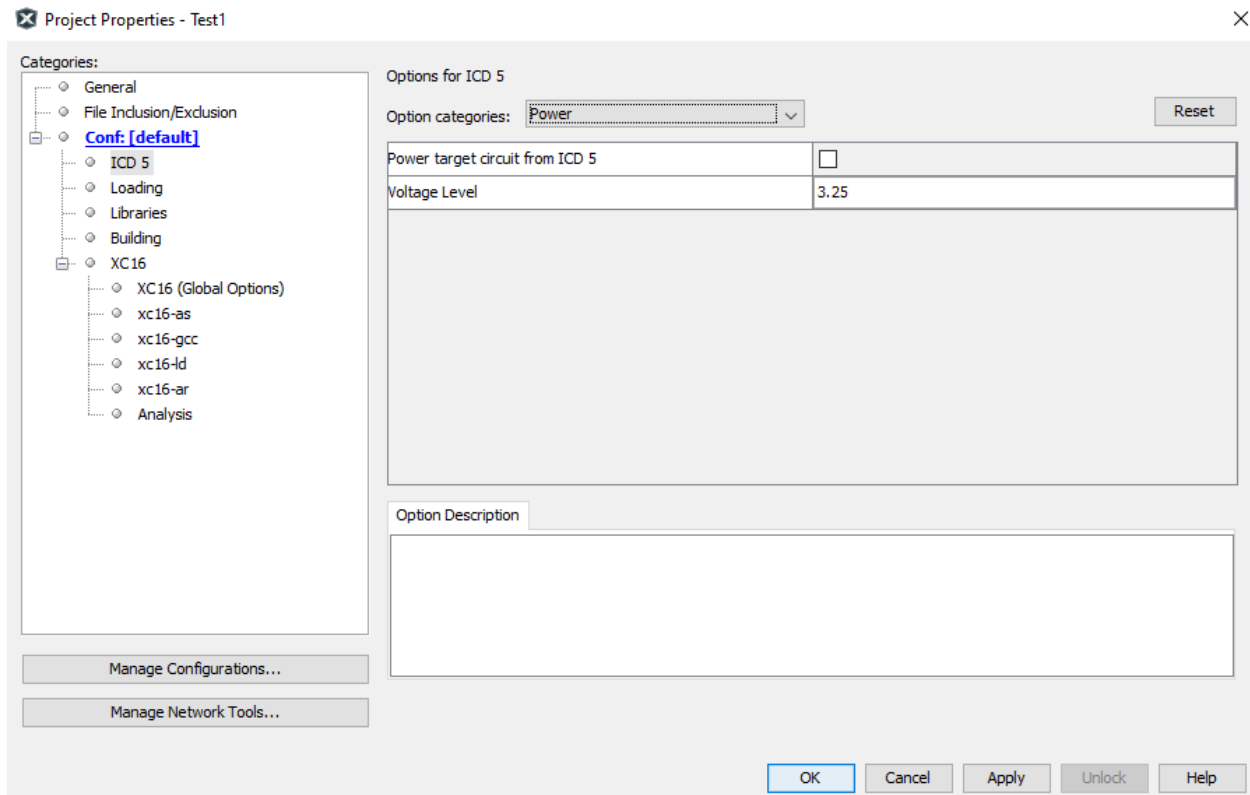
When the MPLAB ICD 5 is connected to and powering the target, the following can be measured:

Table 5-6. Power Monitor Specifications

Current and Voltage	Resolution	Full Scale
Current	29 μ A/step	1.0 A
Voltage	0.2087 mV/step	6.8 V

5.7.2 MPLAB X IDE Setup

In the Project Properties (right click on project name and select “Properties”), ensure that the debugger is not powering the target.



Click the **Make and Program Device** button  to build and program the code into the target device.

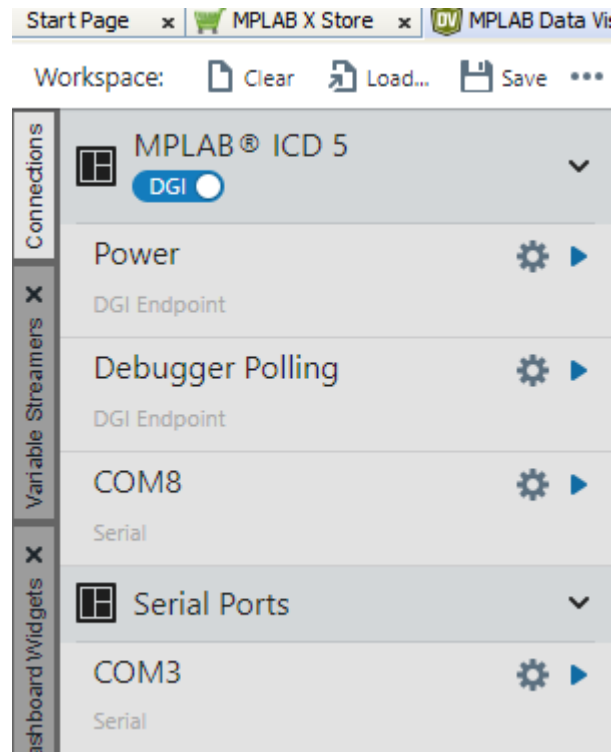
Troubleshooting:

- If the project fails to build, check that you have copied and pasted the sample code fully. Also look at the error messages in the Output window for additional help.
- If MPLAB X IDE has connection issues with the debugger or the target, check your connections.

5.7.3 MPLAB Data Visualizer Displays

MPLAB Data Visualizer may be launched from within MPLAB X IDE or as a stand-alone application.

To open in MPLAB X IDE, select *Window>Debugging>Data Visualizer*. When the data visualizer opens, there will be a Power selection available under the MPLAB ICD 5 DGI list, as current sense is being used. Click on it to view Power Setting controls. For this use case, no power from the debugger will be used (Output Voltage = 0).

Figure 5-9. MPLAB ICD 5 DGI Options

Plot all power sources by clicking on the drop-down arrow and selecting **Plot all sources**. The plot data will start to stream.

6. Troubleshooting First Steps

If you are experiencing problems with MPLAB ICD 5 In-Circuit Debugger operation, the following sections are provided to help.

6.1 Some Questions to Answer First

1. **What device are you working with?** Often an upgrade to a newer pack (DFP/TP) version for MPLAB X IDE is required to support newer devices.
2. **Are you using a Microchip demo board or one of your own design?** Have you followed the guidelines for resistors/capacitors for communications connections? See [3.3. Target Connections](#).
3. **Have you powered the target?** For details see [10.2. Power Specifications](#).
4. **Are you using a USB hub in your setup?** Is it powered? Some hubs may have compatibility issues with MPLAB tools. If you continue to have problems, try using the debugger without the hub (plugged directly into the PC).
5. **Are you using a communication cable shipped with debugger?** If you are using a longer cable, it may have communications errors. If a longer cable is required, consider another type of communication. See [3.2. PC Connections](#).
6. **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.

6.2 Top Reasons Why You Can't Debug

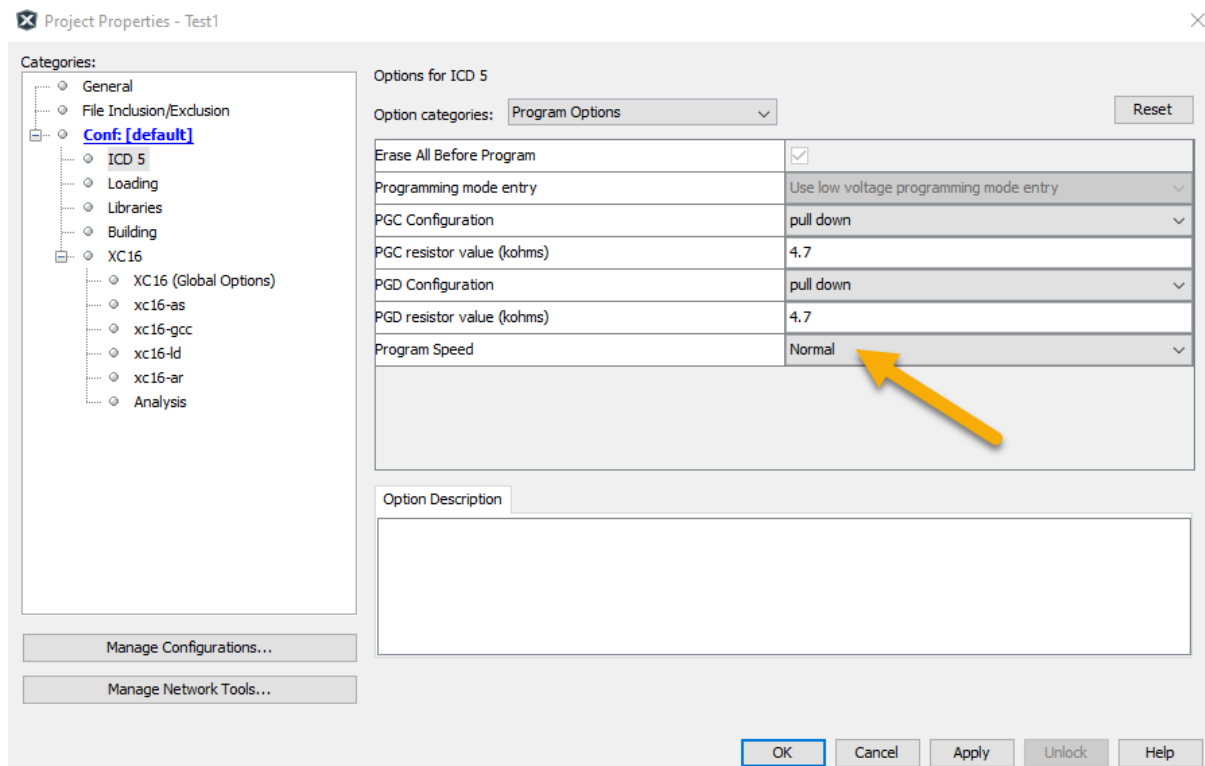
1. **Oscillator not working.** Check your Configuration bits setting for the oscillator. If you are using an external oscillator, try using an internal oscillator then retry debugging. If you are using an internal PLL, make sure your PLL settings are correct.
2. **No power to the target board.** Check the power cable connection to the target or the debugger if powered by the debugger.
3. **Incorrect VDD voltage.** The VDD voltage is outside the specifications for this device. See the device programming specification for details.
4. **Physical disconnect.** The debugger has become physically disconnected from the computer and/or the target board. Check the communications cables' connections.
5. **Communications lost.** The PC to debugger communication has somehow been interrupted. Reconnect to the debugger in MPLAB X IDE.
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 Processor Extension Pak (DS50001292) or Debugger Extension Pak (DS50002243) as required.
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 VDD. This means the device is in Reset and cannot be debugged.
13. **Incorrect connections.** Review the guidelines in [3.3. Target Connections](#) for the correct communication connections.
14. **Invalid request.** The debugger cannot always perform the action requested.

6.3 General Considerations

1. 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.
2. 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.
3. Application is overwriting debugger reserved resources. Check your linker scripts and map files to ensure there is no conflict between the RAM and FLASH areas used by application and debugger.
4. Review debugger setup to ensure proper application setup. For more information, see [4. Operation](#).
5. Your program speed may be set too high for your circuit. In MPLAB X IDE, go to *File>Project Properties*, select **ICD 5** in “Categories,” then “Program Options,” “Program Speed” and select a slower speed from the drop-down menu. The default is “Normal.”

Figure 6-1. Program Speed Option



6. There may be certain situations where the debugger is not operating properly and firmware may need to be downloaded or the debugger needs to be reprogrammed. See the following sections to determine additional actions.

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

NOTICE

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

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

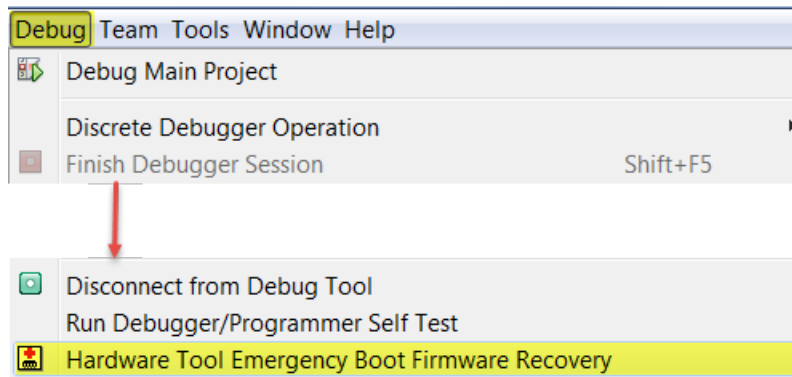
- If the debugger has no LEDs lit.
- If the LEDs are cyan in color.



Important: YOU MUST USE MPLAB X IDE v6.10 OR GREATER TO USE THE EMERGENCY RECOVERY UTILITY FOR MPLAB ICD 5.

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

Figure 6-2. Selecting Emergency Utility



If the procedure was successful, the recovery wizard displays a success screen. The MPLAB ICD 5 will now be operational and able to communicate with the MPLAB X IDE, showing a purple LED color. If the procedure failed, try it again. If it fails a second time, contact Microchip Support at support.microchip.com.

7. Frequently Asked Questions (FAQ)

Answers to frequently asked questions about the MPLAB ICD 5 In-Circuit Debugger system are covered in the following sections.

7.1 How Does It Work?

What's in the silicon that allows it to communicate with the MPLAB ICD 5 In-Circuit Debugger?

MPLAB ICD 5 can communicate with Flash silicon via ICSP and other target interfaces. On some devices, it communicates with a debug executive located in dedicated memory or user memory. For legacy 8-bit PIC devices, the debug executive resides in Program memory. On other devices, it communicates directly with the chip's OCD module.

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

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

How does MPLAB X IDE interface with the MPLAB ICD 5 In-Circuit Debugger to allow more features than older debug tools?

MPLAB ICD 5 communicates using the debug executive located in a dedicated area of memory or using chip's OCD. The debug executive is streamlined for more efficient communication. The debugger contains an FPGA, large SRAM Buffers (1Mx8), and a High-Speed USB interface. The FPGA in the debugger serves as an accelerator for interfacing with the device in-circuit debugger modules.

On traditional debuggers, the data must come out on the bus in order to perform a complex trigger on that data. Is this also required on the MPLAB ICD 5 In-Circuit Debugger? For example, could I halt, based on a flag going high?

Traditional debuggers use a special debugger chip (-ME) for monitoring. There is no -ME with the MPLAB ICD 5, so there are no buses to monitor externally. With the MPLAB ICD 5, rather than using external breakpoints, the built-in breakpoint circuitry of the debug engine is used – the buses and breakpoint logic are monitored inside the part.

Does the MPLAB ICD 5 In-Circuit Debugger support complex breakpoints?

Yes. This is device dependent. If your device has it, then 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.

Will this slow down the running of the program?

There is no cycle stealing with the MPLAB ICD 5.

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

The MPLAB ICD 5 is capable of debugging at any device speed as specified in the device's data sheet.

7.2 What's Wrong?

When MPLAB ICD 5 is not working as expected or not working, please see the sections below for help. See also [8. Error Messages](#).

7.2.1 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, and 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 clear or uncheck the "Enable hibernation" check box. This will ensure that all communication is maintained across all the USB subsystem components.

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

If **Run Main Project**  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** .

7.2.3 During Native Trace, I manually halted my program and now the last trace record has been lost. What happened?

Due to manual Halts being asynchronous, the last piece of data may be dropped. Try running and halting again. Alternatively, use a breakpoint to halt your code.

7.2.4 I set my 16-bit device 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).

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

Some things to consider:

- To determine a Reset source, check the RCON register.
- Handle traps/interrupts in an Interrupt Service Routine (ISR). You should include `trap.c` style code, i.e.,

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

- Use ASSERTs. For example:

```
ASSERT
(IPL==7)
```

8. Error Messages

The MPLAB ICD 5 In-Circuit Debugger produces various error messages; some are specific, some are informational, and others can be resolved with general corrective actions. In general, read any instructions under your error message. If those fail to fix the problem or if there are no instructions, refer to the following sections.

8.1 Types of Error Messages

The following sections group selected error messages into logical categories and propose solutions. The final section lists all error messages.

8.1.1 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.

If the hex file does not exist:

- Reinstall MPLAB X IDE.

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.
3. Restart MPLAB X IDE and reopen your project.
4. Reselect the debugger as the debug tool and try 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.
3. Restart MPLAB X IDE and reopen your project.
4. Reselect the debugger as the programmer and try to program the target device again.

If these actions fail to fix the problem, see Corrupted Installation Actions.

8.1.2 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. If you receive this message after trying to Run, or immediately after programming your device:

The device is code protected.

The device on which you are attempting to operate (read, program, blank check, or verify) is code protected, that is, 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 Debug Failure Actions.

8.1.3 Miscellaneous Errors

ICD 5 is busy. Please wait for the current operation to finish.

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.

8.1.4 List of Error Messages

Table 8-1. Alphabetized List Of Error Messages

AP_VER=Algorithm Plugin Version.
AREAS_TO_PROGRAM=The following memory area(s) will be programmed:
AREAS_TO_READ=The following memory area(s) will be read:
AREAS_TO_VERIFY=The following memory area(s) will be verified:
BLANK_CHECK_COMPLETE=Blank check complete, device is blank.
BLANK_CHECKING=Blank Checking...
BOOT_CONFIG_MEMORY=boot config memory.
BOOT_VER=Boot Version.
BOOTFLASH=boot flash.
BP_CANT_B_DELETED_WHEN_RUNNING=software breakpoints cannot be removed while the target is running. The selected breakpoint will be removed the next time the target halts.
CANT_CREATE_CONTROLLER=Unable to find the tool controller class.
CANT_FIND_FILE=Unable to locate file %s.
CANT_OP_BELOW_LVPTHRESH=The voltage level selected %f, is below the minimum erase voltage of %f. The operation cannot continue at this voltage level.
CANT_PRESERVE_PGM_MEM=Unable to preserve program memory: Invalid range Start = %08x, End = %08x.
CANT_READ_REGISTERS=Unable to read target register(s).
CANT_READ_SERIALNUM=Unable to read the device serial number.
CANT_REMOVE_SWPS_BUSY=The ICD 5 is currently busy and cannot remove software breakpoints at this time.
CHECK_4_HIGH_VOLTAGE_VPP=CAUTION: Check that the device selected in MPLAB X IDE (%s) is the same one that is physically attached to the debug tool. Selecting a 5V device when a 3.3V device is connected can result in damage to the device when the debugger checks the device ID. Do you wish to continue?
CHECK_PGM_SPEED=You have set the program speed to %s. The circuit on your board may require you to slow the speed down. Please change the setting in the tool properties to low and try the operation again.
COMM_PROTOCOL_ERROR=A communication error with the debug tool has occurred. The tool will be reset and should re-enumerate shortly.
COMMAND_TIME_OUT=ICD 5 has timeout out waiting for a response to command %02x.
CONFIGURATION=configuration.
CONFIGURATION_MEMORY=configuration memory.
CONNECTION_FAILED=Connection Failed.
CORRUPTED_STREAMING_DATA=Invalid streaming data has been detected. Run time watch or trace data may no longer be valid. It is recommended that you restart your debug session.
CPM_TO_TARGET_FAILED=An exception occurred during ControlPointMediator.ToTarget().
DATA_FLASH_MEMORY=Data Flash memory.

DATA_FLASH=data flash.
DEBUG_INFO_PGM_FAILED=Could not enter debug mode because programming the debug information failed. Invalid combinations of config bits may cause this problem.
DEBUG_READ_INFO=Reading the device while in debug mode may take a long time due to the target oscillator speed. Reducing the range that you'd like to read (under the ICD 5 project properties) can mitigate the situation. The abort operation can be used to terminate the read operation if necessary.
DEVICE_ID_REVISION=Device Id Revision.
DEVICE_ID=Device Id.
DEVID_MISMATCH=Target Device ID (0x%x) is an Invalid Device ID. Please check your connections to the Target Device.
DISCONNECT_WHILE_BUSY=The tool was disconnected while it was busy.
EEDATA_MEMORY=EEData memory.
EEDATA=EEData.
EMULATION_MEMORY_READ_WRITE_ERROR=An error occurred while trying to read/write MPLAB's emulation memory: Address=%08x.
END=end.
ENSURE_SELF_TEST_READY=Please ensure the RJ-11 cable is connected to the test board before continuing.
ENSURE_SELF_TEST_READY=Please ensure the RJ-11 cable is connected to the test board before continuing. Would you like to continue?
ENV_ID_GROUP=Device Identification.
ERASE_COMPLETE=Erase successful.
ERASING=Erasing...
FAILED_2_PGM_DEVICE=Failed to program device.
FAILED_CREATING_COM=Unable create communications object (RI4Com).
FAILED_CREATING_DEBUGGER_MODULES=Initialization failed: Failed creating the debugger module.
FAILED_ESTABLISHING_COMMUNICATION=Unable to establish tool communications.
FAILED_GETTING_DBG_EXEC=A problem occurred while trying to load the debug executive.
FAILED_GETTING_DEVICE_INFO=Initialization failed: Failed while retrieving device database (.pic) information.
FAILED_GETTING_EMU_INFO=Initialization failed: Failed getting emulation database information.
FAILED_GETTING_HEADER_INFO=Initialization failed: Failed getting header database information.
FAILED_GETTING_PGM_EXEC=A problem occurred while trying to load the program executive.
FAILED_GETTING_TEX=Unable to obtain the ToolExecMediator.
FAILED_GETTING_TOOL_INFO=Initialization failed: Failed while retrieving tool database (.ri4) information.
FAILED_INITING_DATABASE=Initialization failed: Unable to initialize the tool database object.
FAILED_INITING_DEBUGHANDLER=Initialization failed: Unable to initialize the DebugHandler object.
FAILED_PARSING_FILE=Failed to parse firmware file: %s.
FAILED_READING_EMULATION_REGS=Failed to read emulation memory.
FAILED_READING_MPLAB_MEMORY=Unable to read %s memory from %0x08 to %0x08.
FAILED_SETTING_SHADOWS=Failed to properly set shadow registers.
FAILED_SETTING_XMIT_EVENTS=Unable to synchronize run time data semaphores.
FAILED_STEPPING=Failed while stepping the target.
FAILED_TO_GET_DEVID=Failed to get Device ID. Please make sure the target device is attached and try the operation again.
FAILED_TO_INIT_TOOL=Failed to initialize ICD 5.
FAILED_UPDATING_BP=Failed to update breakpoint:\nFile: %s\naddress: %08x.
FAILED_UPDATING_FIRMWARE=Failed to properly update the firmware.
FILE_REGISTER=file register.
FIRMWARE_DOWNLOAD_TIMEOUT=ICD 5 timeout out during the firmware download process.
FLASH_DATA_MEMORY=Flash data memory.
FLASH_DATA=flash data.

FPGA_VER=FPGA Version.
FRCINDEBUG_NEEDS_CLOCKSWITCHING=To use FRC in debug mode the clock switching configuration bits setting must be enabled. Please enable clock switching and retry the requested operation.
FW_DOESNT_SUPPORT_DYNBP=The current ICD 5 firmware does not support setting run time breakpoints for the selected device. Please download firmware version %02x.%02x.%02x or higher.
GOOD_ID_MISMATCH=Target Device ID (0x%x) is a valid Device ID but does not match the expected Device ID (0x%x) as selected.
HALTING=Halting...
HIGH=High.
HOLDMCLR_FAILED=Hold in reset failed.
IDS_SELF_TEST_PASSED=ICD5 is functioning properly. If you are still having problems with your target circuit please check the Target Board Considerations section of the online help.
IDS_ST_CLKREAD_ERR=Test interface PGC clock line read failure.
IDS_ST_CLKREAD_NO_TEST=Test interface PGC clock line read not tested.
IDS_ST_CLKREAD_SUCCESS=Test interface PGC clock line read succeeded.
IDS_ST_CLKWRITE_ERR=Test interface PGC clock line write failure. Please ensure that the tester is properly connected.
IDS_ST_CLKWRITE_NO_TEST=Test interface PGC clock line write not tested.
IDS_ST_CLKWRITE_SUCCESS=Test interface PGC clock line write succeeded.
IDS_ST_DATREAD_ERR=Test interface PGD data line read failure.
IDS_ST_DATREAD_NO_TEST=Test interface PGD data line read not tested.
IDS_ST_DATREAD_SUCCESS=Test interface PGD data line read succeeded.
IDS_ST_DATWRITE_ERR=Test interface PGD data line write failure.
IDS_ST_DATWRITE_NO_TEST=Test interface PGD data line write not tested.
IDS_ST_DATWRITE_SUCCESS=Test interface PGD data line write succeeded.
IDS_ST_LVP_ERR=Test interface LVP control line failure.
IDS_ST_LVP_NO_TEST=Test interface LVP control line not tested.
IDS_ST_LVP_SUCCESS=Test interface LVP control line test succeeded.
IDS_ST_MCLR_ERR=Test interface MCLR level failure.
IDS_ST_MCLR_NO_TEST=Test interface MCLR level not tested.
IDS_ST_MCLR_SUCCESS=Test interface MCLR level test succeeded.
IDS_TEST_NOT_COMPLETED=Interface test could not be completed. Please contact your local FAE/CAE to SAR the unit.
INCOMPATIBLE_FW=The ICD 5 firmware is not compatible with the current version of MPLAB X software.
INVALID_ADDRESS=The operation cannot proceed because the %s address is outside the devices address range of 0x%08x - 0x%08x.
MEM_RANGE_ERROR_BAD_END_ADDR=Invalid program range end address %s received. Please check the manual program ranges on the debug tool's, "Memories to Program" property page.
MEM_RANGE_ERROR_BAD_START_ADDR=Invalid program range start address %s received. Please check the manual program ranges on the debug tool's, "Memories to Program" property page.
MEM_RANGE_ERROR_END_LESSTHAN_START=Invalid program range received: end address %s < start address %s. Please check the manual program ranges on the debug tool's, "Memories to Program" property page.
MEM_RANGE_ERROR_ENDADDR_NOT_ALIGNED=Invalid program range received: end address %s is not aligned on a proper 0x%x address boundary. Please check the manual program ranges on the debug tool's, "Memories to Program" property page.
MEM_RANGE_ERROR_STARTADDR_NOT_ALIGNED=Invalid program range received: start address %s is not aligned on a proper 0x%x address boundary. Please check the manual program ranges on the debug tool's, "Memories to Program" property page.
MEM_RANGE_ERROR_UNKNOWN=An unknown error has occurred while trying to validate the user entered memory ranges.
MEM_RANGE_ERROR_WRONG_DATABASE=Unable to access data object while validating user entered memory ranges.
MEM_RANGE_OUT_OF_BOUNDS=The selected program range, %s, does not fall within the proper range for the memory area selected. Please check the manual program ranges on the debug tool's, "Memories to Program" property page.

MEM_RANGE_STRING_MALFORMED=The memory range(s) entered on the, "Memories to Program" property page (%s) is not formatted properly.
MISSING_BOOT_CONFIG_PARAMETER=Unable to find boot config start/end address in database.
MUST_SET_LVPBIT_WITH_LVP=The low voltage programming feature requires the LVP configuration bit to be enabled on the target device. Please enable this configuration bit and try the operation again.
NEW_FIRMWARE=Now Downloading new Firmware for target device: %s
NMMR=NMMR
NO_DYNAMIC_BP_SUPPORT_AT_ALL=The current device does not support the ability to set breakpoints while the devices is running. The breakpoint will be applied prior to the next time you run the device.
NO_PGM_HANDLER=Cannot program software breakpoints. The program handler has not been initialized.
NORMAL=Normal.
OP_FAILED_FROM_CP=The requested operation failed because the device is code protected.
OpenIDE-Module-Name=ICD 5
OPERATION_NOT_SUPPORTED=This operation is not supported for the selected device.
OUTPUTWIN_TITLE=ICD 5.
PERIPHERAL=Peripheral.
POWER_ERROR_NO_9V=The configuration is set for the tool to provide power to the target but the 9V power jack is not detected. Please ensure the external 9V barrel jack is connected to the tool.
POWER_ERROR_NO_POWER_SRC=The configuration is set for the target board to supply its own power but no voltage has been detected on VDD. Please ensure you have your target powered up and try again.
POWER_ERROR_POWER_SRC_CONFLICT=The configuration is set for the tool to provide power to the target but there is voltage already detected on VDD. This is a conflict. Please ensure your target is not supplying voltage to the tool and try again.
POWER_ERROR_SLOW_DISCHARGE= There seems to be excessive capacitance on VDD causing a slower system discharge and shutdown. Consider minimizing overall capacitance loading or use power from your target to avoid discharge delays.
POWER_ERROR_UNKNOWN=An unknown power error has occurred.
POWER_ERROR_VDD_TOO_HIGH=The VDD voltage desired is out of range. It exceeds the maximum voltage of 5.5V.
POWER_ERROR_VDD_TOO_LOW=The VDD voltage desired is out of range. It is below the minimum voltage of 1.5V.
POWER_ERROR_VPP_TOO_HIGH=The VPP voltage desired is out of range. It exceeds the maximum voltage of 14.2V.
POWER_ERROR_VPP_TOO_LOW=The VPP voltage desired is out of range. It is below the minimum voltage of 1.5V.
PRESERVE_MEM_RANGE_ERROR_BAD_END_ADDR=Invalid preserve range end address %s received. Please check the manual program ranges on the debug tool's, "Memories to Program" property page.
PRESERVE_MEM_RANGE_ERROR_BAD_START_ADDR=Invalid preserve range start address %s received. Please check the manual program ranges on the debug tool's, "Memories to Program" property page.
PRESERVE_MEM_RANGE_ERROR_END_LESSTHAN_START=Invalid preserve range received: end address %s < start address %s. Please check the manual program ranges on the debug tool's, "Memories to Program" property page.
PRESERVE_MEM_RANGE_ERROR_ENDADDR_NOT_ALIGNED=Invalid preserve range received: end address %s is not aligned on a proper 0x%x address boundary. Please check the manual program ranges on the debug tool's, "Memories to Program" property page.
PRESERVE_MEM_RANGE_ERROR_STARTADDR_NOT_ALIGNED=Invalid preserve range received: start address %s is not aligned on a proper 0x%x address boundary. Please check the manual program ranges on the debug tool's, "Memories to Program" property page.
PRESERVE_MEM_RANGE_ERROR_UNKNOWN=An unknown error has occurred while trying to validate the user entered preserve ranges.
PRESERVE_MEM_RANGE_ERROR_WRONG_DATABASE=Unable to access data object while validating user entered memory ranges.
PRESERVE_MEM_RANGE_MEM_NOT_SELECTED=You have selected to preserve an area of memory but have not selected to program that area. Please check the preserved ranges on the debug tool's, "Memories to Program" property page, and make sure that any preserved memory is also designated to be programmed.
PRESERVE_MEM_RANGE_OUT_OF_BOUNDS=The selected preserve range, %s, does not fall within the proper range for the memory area selected. Please check the manual program ranges on the debug tool's, "Memories to Program" property page.

PRESERVE_MEM_RANGE_STRING_MALFORMED=The preserve memory range(s) entered on the, "Memories to Program" property page (%s) is not formatted properly.
PRESERVE_MEM_RANGE_WONT_BE_PROGRAMMED=Some or all of the preserve memory ranges (%s) entered on the, "Memories to Program" property page, do not fall under the indicated program range(s) (%s) for the memory selected. Please check the preserved ranges on the debug tool's, "Memories to Program" property page.
PROGRAM_COMPLETE=Programming/Verify complete.
PROGRAM_MEMORY=program memory.
PROGRAM=program.
PROGRAMMING_DID_NOT_COMPLETE=Programming did not complete.
READ_COMPLETE=Read complete.
READ_DID_NOT_COMPLETE=Read did not complete.
RELEASEMCLR_FAILED=Release from reset failed.
REMOVING_SWBPS_COMPLETE=Removing software breakpoints complete.
REMOVING_SWBPS=Removing software breakpoints...
RESET_FAILED=Failed to reset the device.
RESETTING=Resetting...
RUN_INTERRUPT_THREAD_SYNCH_ERROR=An internal run error has occurred. It is advised that you restart your debug session. You may continue running but certain run time features may no longer work properly.
RUN_TARGET_FAILED=Unable to run the target device.
RUNNING=Running.
SD_RESULT_NO_ERROR=Empty Trace File result
SERIAL_NUM=Serial Number:\n
SETTING_SWBPS=Setting software breakpoints.....
STACK=stack.
START_AND_END_ADDR=start address = 0x%x, end address = 0x%x.
START=start.
TARGET_DETECTED=Target voltage detected.
TARGET_FOUND=Target device %s found.
TARGET_HALTED=Target Halted.
TARGET_NOT_READY_4_DEBUG=The target device is not ready for debugging. Please check your configuration bit settings and program the device before proceeding. The most common causes for this failure are oscillator and/or PGC/PGD settings.
TARGET_VDD=Target VDD:
TEST=test.
TOOL_IS_BUSY=ICD 5 is busy. Please wait for the current operation to finish.
TOOL_VDD=VDD:
TOOL_VPP=VPP:
UNABLE_TO_OBTAIN_RESET_VECTOR=ICD 5 was unable to retrieve the reset vector address. This indicates that no _reset symbol has been defined and may prevent the device from starting up properly.
UNKNOWN_MEMTYPE=Unknown memory type.
UNLOAD_WHILE_BUSY=ICD 5 was unloaded while still busy. Please unplug and reconnect the USB cable before using ICD 5 again.
UPDATING_APP=Updating firmware application...
UPDATING_BOOTLOADER=Updating firmware bootloader...
UPDATING_FPGA=Updating firmware FPGA...
USE_LVP_PROGRAMMING=NOTE: If you would like to program this device using low voltage programming, select Cancel on this dialog. Then go to the ICD5 node of the project properties and check the Enable Low Voltage Programming check box of the Program Options Option Category pane (low voltage programming is not valid for debugging operations).
USERID_MEMORY=User Id Memory.
USERID=user Id.

VERIFY_COMPLETE=Verification successful.
VERIFY_FAILED=Verify failed.
VERSIONS=Versions.
VOLTAGES=Voltages.
WOULD_YOU_LIKE_TO_CONTINUE=Would you like to continue?

8.2 General Corrective Actions

The general corrective actions in the following sections may solve your problem.

8.2.1 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 section Program).
6. Ensure that the cable(s) are of the correct length.

8.2.2 Debugger to Target Communication Error Actions

If the MPLAB ICD 5 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.

8.2.3 Debugger to Computer Communication Error Actions

If the MPLAB ICD 5 In-Circuit Debugger and MPLAB X IDE 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 a pack (DPF/TP) installed in MPLAB X IDE may be incorrect for the target device. See MPLAB X IDE documentation for information on how to install updated packs.
5. There may be an issue with the computer USB port. See section USB Port Communication Error Actions.

8.2.4 Corrupted Installation Actions

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

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

8.2.5 USB Port Communication Error Actions

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

1. Reconnect to the MPLAB ICD 5 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 section 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.

8.2.6 Debug Failure Actions

The MPLAB ICD 5 In-Circuit Debugger was unable to perform a debugging operation. Multiple scenarios can result in this error. A few of these are described below.

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

generate this message. Use debug mode () instead.

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

8.2.7 Internal Error Actions

Internal errors are not expected nor should happen. They are used for internal Microchip development.

The most likely cause is a corrupted installation (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.

9. Debugger Function Summary

A summary of MPLAB® ICD 5 In-Circuit Debugger functions is provided in the following topics.

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 MPLAB® ICD 5 In-Circuit Debugger connected to your computer. MPLAB X IDE will differentiate between the two by displaying two different serial numbers.

To select or change the debugger used for a project:

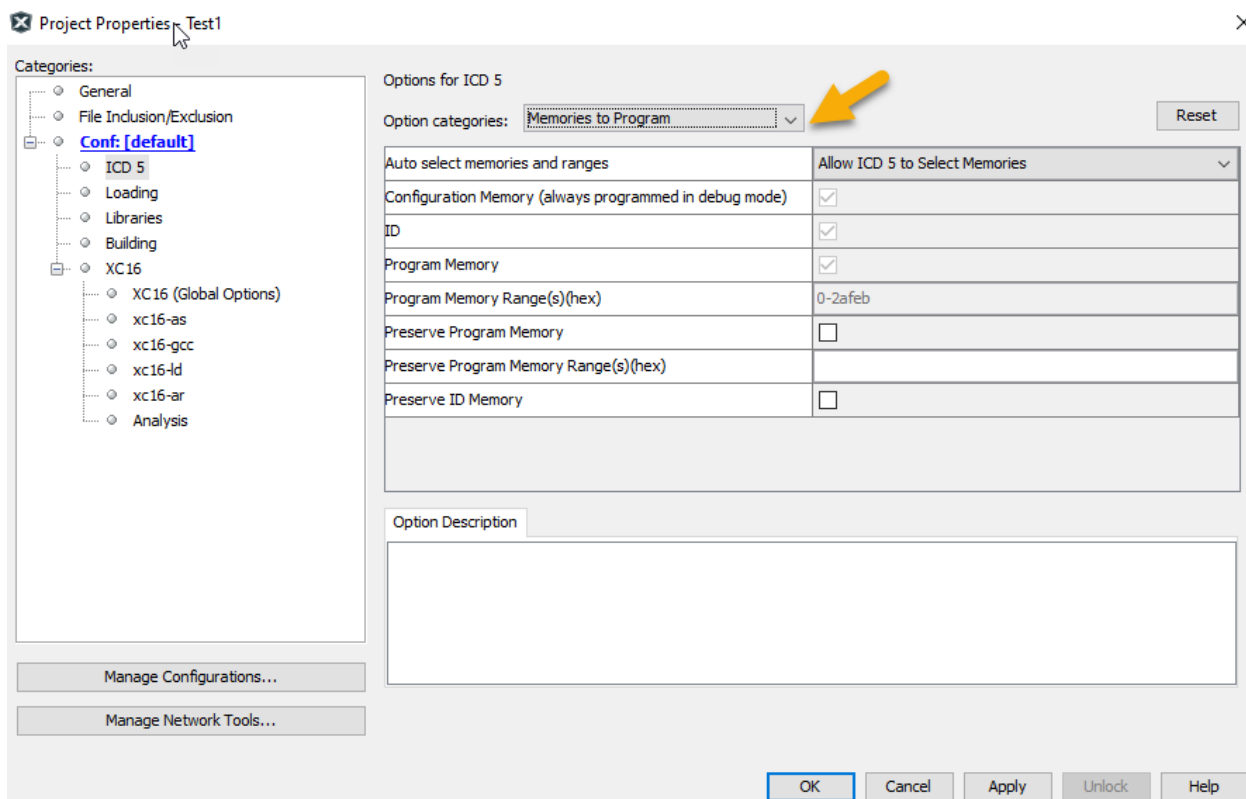
1. Open the Project Properties dialog by doing one of the following:
 - a. Click on the project name in the Projects window and select *File>Project Properties*.
 - b. Right click on the project name in the Projects window and select "Properties."
2. Under "Categories," click on "Conf: [default]."
3. Under "Hardware Tools," find "ICD 5" and click on a serial number (SN) to select an debugger for use in the project.

9.2 Debugger Options Selection

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

1. Open the Project Properties dialog by doing one of the following:
 - a. Click the project name in the Projects window, select *File>Project Properties*.
 - b. Right click the project name in the Projects window, select "Properties."
2. Under "Categories," click on **ICD 5**.
3. Select property pages from "Options categories." Click on an option to see its description in the text box below it. Click to the right of an option to change it.
Note: Options displayed may be different for different devices.

Figure 9-1. Project Properties - ICD 5 Options



9.2.1 Memories to Program

Select the memories to be programmed into the target.

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

Table 9-1. Memories to Program Option Category

Auto select memories and ranges	Allow ICD 5 to Select Memories – The debugger uses your selected device and default settings to determine what to program. Manually select memories and ranges – You select the type and range of memory to program (see below).
Configuration Memory (always programmed in debug mode)	Check to program configuration memory in release mode. For dual partition devices, another selection for partition 2 will be available.
<i>Memory</i>	Check to program <i>Memory</i> . Types of memory include: Instruction RAM, Flash Data, Data Flash, EEPROM, ID, Boot Flash, Auxiliary.
<i>Memory Range(s) (hex)*</i>	The starting and ending hex address range in <i>Memory</i> . Types of memory include: Instruction RAM.
Program Memory	Check to program the target program memory range specified below.
Program Memory Range(s) (hex)*	The starting and ending hex address range in program memory for programming, reading, or verification. Note: The address range does not apply to the Erase function. The Erase function will erase all data on the device.

Preserve Program Memory	Check to not program the target program memory range specified below. Ensure code is NOT code protected.
Preserve Program Memory Range(s) (hex)*	The starting and ending hex address range in target program memory to preserve when programming, reading, or verifying. This memory is read from the target and overlayed with existing MPLAB X IDE memory.
Preserve <i>Memory</i>	Check to preserve <i>Memory</i> for reprogramming. Types of memory include: Instruction RAM, Flash Data, Data Flash, EEPROM, ID, Boot Flash, Auxiliary. Ensure code is NOT code protected.
Preserve <i>Memory</i> Range(s) (hex)*	The starting and ending hex address range in target <i>Memory</i> to preserve when programming, reading, or verifying. Types of memory include: Instruction RAM, Flash Data, Data Flash, EEPROM, Boot Flash, Auxiliary. This memory is read from the target and overlayed with existing MPLAB X IDE memory. Ensure code is NOT code protected.
* If you receive a programming error due to an incorrect range, ensure the range does not exceed available/remaining device memory.	

9.2.2 Debug Options

Select debug options, if available for the project device.

Table 9-2. Debug Options Option Category

Debug startup	System settings may be found under <i>Tools>Options>Embedded>Generic Settings</i> , but may be changed here: Use system settings, Run, Halt at main, Halt at reset vector.
Debug reset	System settings may be found under <i>Tools>Options>Embedded>Generic Settings</i> , but may be changed here: Use system settings, Main, ResetVector.
Use Software Breakpoints	Check to use software breakpoints. Uncheck to use hardware breakpoints. See the discussion below to determine which type is best for your application.
Use Simultaneous Debug	Check to indicate that the project is part of a multi-core simultaneous debug session.

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, it can take minutes	minimal
Skidding	no	yes

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

9.2.3 Program Options

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

Table 9-4. Program Options Option Category

Erase All Before Program	Check to erase all memory before programming begins. Unless programming new or already erased devices, it is important to have this box checked. If it is not checked, the device is not erased and program code will be merged with the code already in the device.
Do not erase auxiliary memory	For devices that support auxiliary memory: Check to not erase aux memory when programming. Uncheck to erase aux memory when programming.

9.2.4 ICD 5 Tool Options

Set up MPLAB ICD 5 specific options.

Table 9-5. ICD 5 Tool Options Category

Programming Mode Entry	Select programming mode entry type: • Use high voltage program mode entry • Use low voltage program mode entry
PGC Configuration	Programming clock pin setup. • none • pull up • pull down
PGC resistor value (kohms)	If pull up or pull down is selected above, enter a resistor value from 0 to 50 kohms.
PGD Configuration	Programming data pin setup. • none • pull up • pull down
PGD resistor value (kohms)	If pull up or pull down is selected above, enter a resistor value from 0 to 50 kohms.

9.2.5 Freeze Peripherals

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

Table 9-6. Freeze Peripherals Option Category

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

PIC12/16/18 MCU Devices

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

dsPIC, PIC24 and PIC32 Devices

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

9.2.6 Trace and Profiling

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

8-Bit and 16-Bit Devices

Options available on this page depend on the trace/profiling features of the project device.

Table 9-7. Trace/Profiling Option Category

Data Collection Selection	Enable/Disable data collection. - Off - Do not collect target data. - User Instrumented Trace. - PC Sampling. - Power Monitor (Target Power Sampling).
Data File Path and Name	Enter or change the path and/or name of the file used to store data. - Enter file name (path will be relative to project) – Recommended. - Enter a path and file name (path will be absolute). - Browse (...) to a file, select “Absolute,” select the file, and click Save (path will be absolute). Note: Do not select “Relative” when browsing to a file or MPLAB X IDE will not be able to find the file. When you run, you will receive a warning message that the path does not exist.
Data File Maximum Size (bytes)	Set the maximum size of the data file. Target power sampling will take 12 bytes or 18 bytes (with PC data) per sample. The file size may be adjusted down to be a multiple of one of those byte sizes depending upon the trace type selected. Other trace data types may use record byte sizes that are different from those described above.
Data Buffer Maximum Size (bytes)	Set the size of the data buffer, up to 54600 bytes (on board the emulator unit). For trace/sampling data that is buffered in memory while the target is running, individual trace or sample entry sizes vary depending upon the trace/sample type and the device and tool being used. It is normally good to make this buffer as large as possible. For example, the enhanced PIC16 with instruction trace uses 1 to 3 bytes for each in-memory entry. Each of those will generate a 13-byte ICD5 instruction trace entry as well. Each such in-memory record will normally be converted to a trace data file entry line, as detailed in the data file size description (refer to the data file size description for trace/sampling file entry sizes).
Stall CPU When Trace Buffer is Full	Stop execution when the trace buffer is full. Set the buffer size in the option described above.
User Instrumented Trace Items	
Disable Trace Macros	Check to temporarily disable trace macros or uncheck to enable trace macros. To disable trace, remove all macros and select “Off” under “Data Collection Selection.”
Communications Medium	Select the trace medium, if available, from the following (device-dependent): Native, I/O Port, SPI.
I/O Port Selection	Specify the device port to be used for I/O port trace. The available combinations for the selected device will be listed.
SPI Selection	Specify the device SPI pins to be used for SPI trace. The available pins for the selected device will be listed.
PC Sampling Items	
Timer Selection (Not Used by Application Code)	Select a device timer to use to count PC samples. Note: You will no longer be able to use this timer in your application, it will be dedicated to PC sampling. Note: You may select only one timer; you cannot combine two timers to get a 32-bit timer. Using one timer of a 32-bit-timer pair will prohibit that pair from operating as a 32-bit timer.
Timer Interrupt Priority	Select an interrupt priority for the timer. Note: Select a priority that is higher than other priorities you have set in your application. If you do not, the other priorities will preempt the sampling priority and you will not capture these samples.
Timer Interval	Enter a sampling interval. This must be integer values (1, 2, 3, and so forth). If you are not capturing data, you may be missing samples (given your current interval). Try adjusting the unit selection and interval, for example, if you had 1 millisecond, try 990 microseconds.

Timer Interval Units	Select a sampling interval unit: • microseconds • milliseconds • seconds • instruction cycles
----------------------	--

32-Bit Devices

Options available on this page depend on the trace/profiling features of the project device.

Table 9-8. Trace/Profiling Option Category

Data Collection Selection	Enable/Disable data collection. • Off - Do not collect target data. • Instruction Trace/Profiling. • User Instrumented Trace. • Power Monitor (Target Power Sampling).
Data File Path and Name	Enter or change the path and/or name of the file used to store data. • Enter file name (path will be relative to project) – Recommended. • Enter a path and file name (path will be absolute). • Browse (...) to a file, select “Absolute,” select the file, and click Save (path will be absolute). Note: Do not select “Relative” when browsing to a file or MPLAB X IDE will not be able to find the file. When you run, you will receive a warning message that the path does not exist.
Data File Maximum Size (bytes)	Set the maximum size of the data file. Each line of instruction trace data in a trace data file requires 13 bytes when using the debugger. Target power sampling will take 12 bytes or 18 bytes (with PC data) per sample. The file size may be adjusted down to be a multiple of one of those byte sizes depending upon the trace type selected. Other trace data types may use record byte sizes different from those described above.
Data Buffer Maximum Size (bytes)	Set the size of the data buffer, up to 54600 bytes (on board the debugger unit). For trace/sampling data that is buffered in memory while the target is running, individual trace or sample entry sizes vary depending on the trace/sample type and the device and tool being used. It is normally good to make this buffer as large as possible. For example, PIC32 instruction trace takes 8 bytes per “frame” which can produce over 50 13-byte ICD 5 instruction trace entries in a trace file.
User Instrumented Trace Items	
Disable Trace Macros	Check to temporarily disable trace macros or uncheck to enable trace macros. To disable trace, remove all macros and select “Off” under “Data Collection Selection.”
Communications Medium	Select the trace medium, if available, from the following (device-dependent): Native

9.2.7 Power

Select whether or not to power the target from the debugger.

Table 9-9. Power Option Category

Power Target Circuit from ICD 5	Check to use power from the debugger. Uncheck to power the target from its own power supply.
Voltage Level	If option above checked, select the target Vdd (2.375V - 5.5V) that the debugger will provide.

9.2.8 Clock

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

Table 9-10. Clock Option Category

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

9.2.9 Communication

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

Table 9-11. Communication Option Category

Interface	Select the interface from the available options based on the project device.
Speed (MHz)	Enter a speed based on the available range for the interface.
High Voltage Activation Mode	<i>This option displays only for AVR® devices with this option.</i> No High Voltage - Default setting. Simple High Voltage Pulse - The tool will try to activate the interface by issuing a high voltage pulse. This procedure is safe if the pin is configured as an input. User Power Toggle - In this mode the user will be prompted to toggle power on the target device. Once the tool detects that the power returns it will issue a high voltage pulse before the target device pin is configured, making the activation procedure as gentle as possible. See also UPDI High-Voltage Activation Information .

9.2.9.1 User Power Toggle Design Considerations

When using the debugger, if the power toggle rise time on Target Vdd is too slow (greater than 10 seconds) the User Power Toggle feature won't work. As an example, for the STK600 using the power switch gives you a too slow rise time but using the VTARGET jumper gives you a fast enough rise time.

For developers creating their own boards, ensure the Vdd rise time is less than 10 seconds.

9.2.9.2 Programming AVR Devices with UPDI

MPLAB ICD 5 supports using the high-voltage mechanism to activate the AVR Unified Program and Debug Interface (UPDI). On low pin count AVR devices with UPDI, the UPDI pin can be configured as GPIO or RESET by configuring the RSTPINCFG configuration bits. To do further programming, the debugger will have to use a high voltage pulse to reactivate the UPDI interface. When using the high voltage pulse, you must make sure that all circuits connected to the UPDI wire can tolerate a pulse of at least 12V.

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 9-12. SYSCFG0 RSTPINCFG[1:0] Configuration Bits

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

9.2.10 Tool Pack Selection

Select to use the latest tool pack or a different version to support the project device.

Table 9-13. Tool Pack Selection Option Category

Tool pack update options	Use latest installed tool pack (recommended) - use the latest tool pack version installed. Use specific tool pack - select from a list of other available tool pack versions.
Specifically selected version	Click here to pop up a dialog with a list of tool pack versions to select.

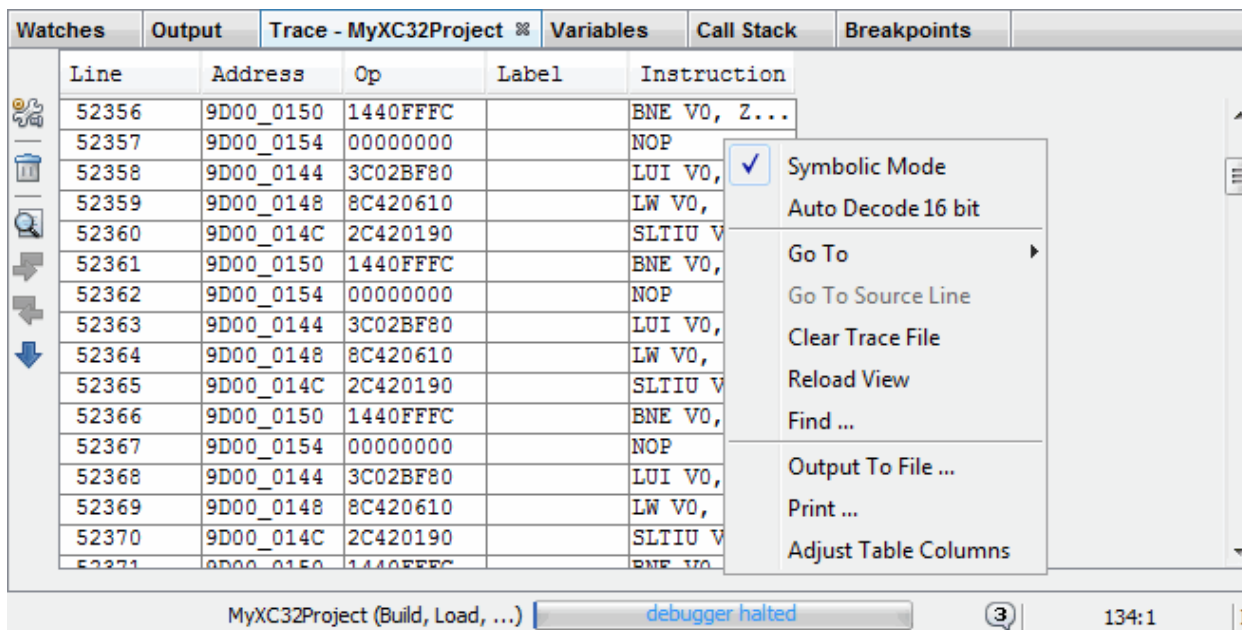
9.3 Debugger Windows & Dialogs

The following topics show windows and dialogs that are used specifically for the debugger or other related debug tools.

9.3.1 Trace Window and Related Dialogs

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

Figure 9-2. Trace Window



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

9.3.2 ITM Window and Related Dialogs

SAM ITM trace produces UART-format data that supports `printf`-style debugging on up to 32 ports.

Figure 9-3. ITM Display

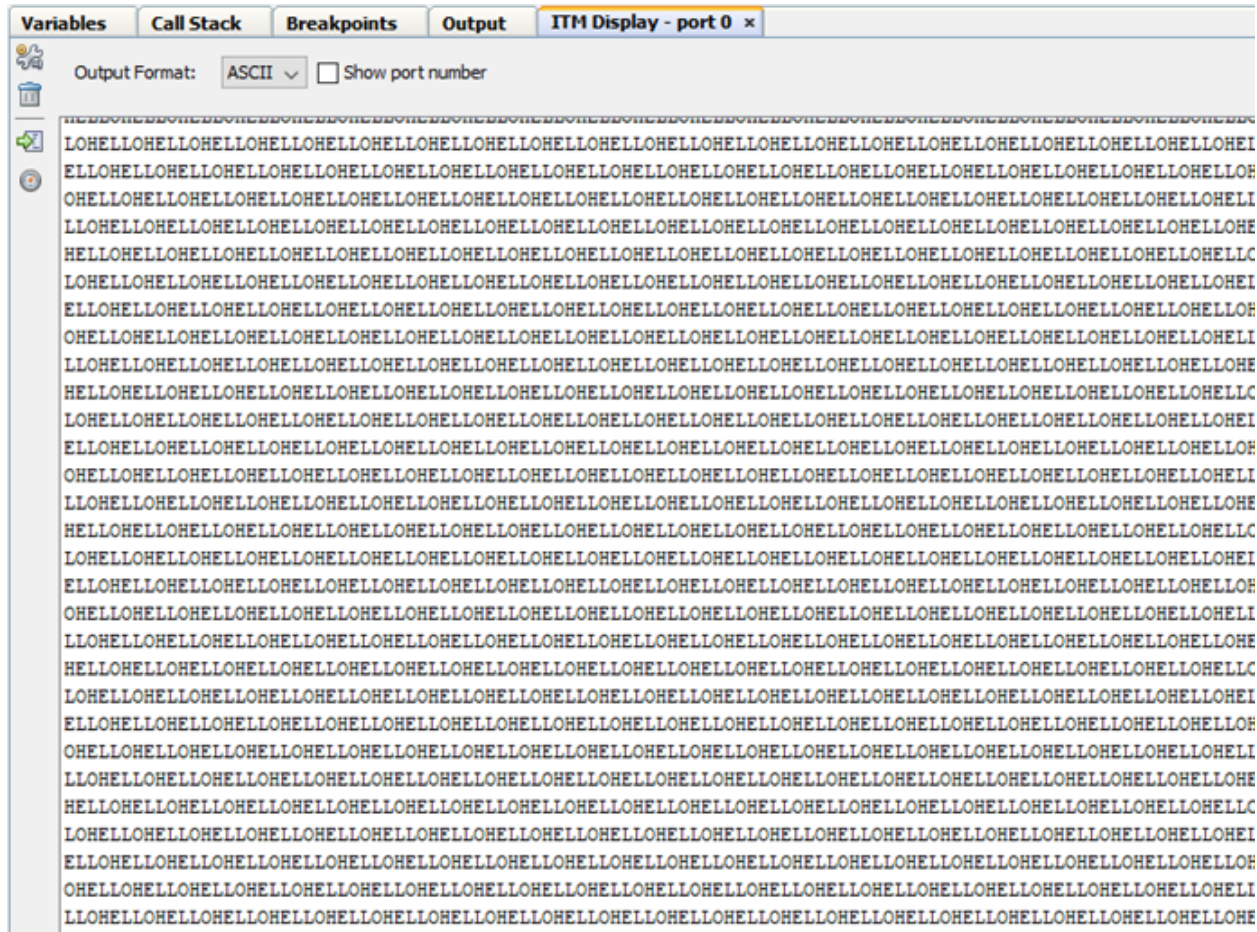
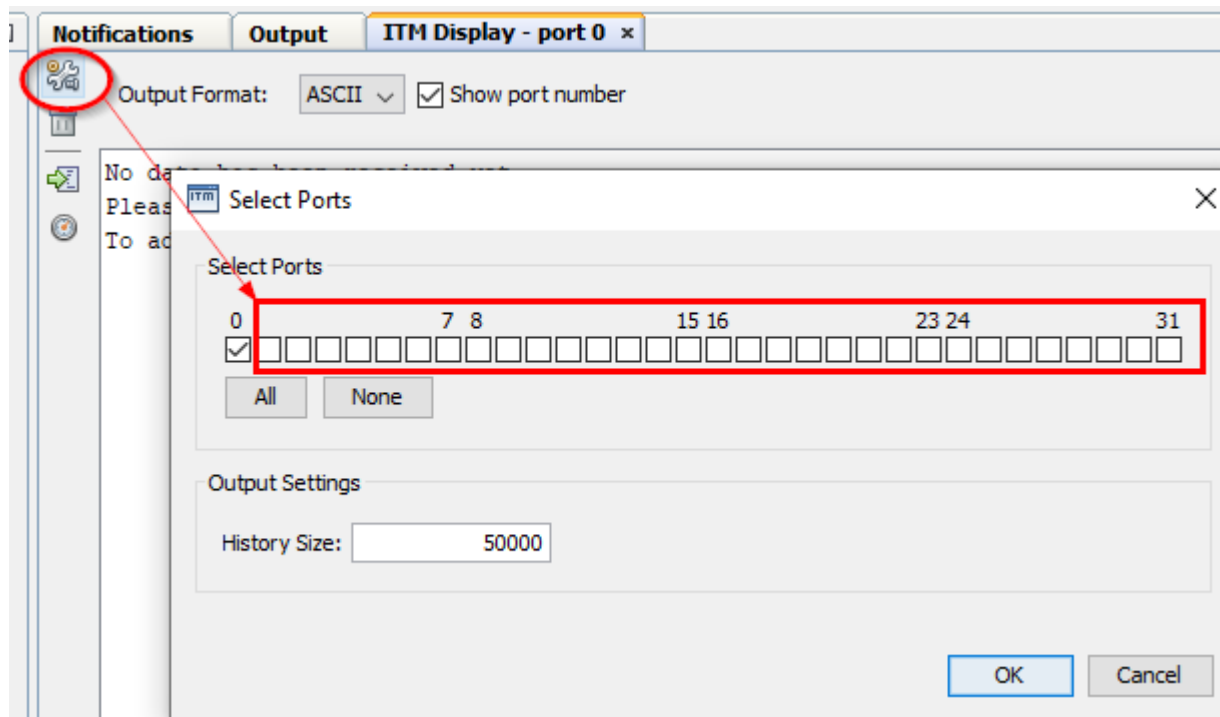


Figure 9-4. ITM Select Ports Dialog

**Related Links**[5.4. ARM ITM/SWO Trace](#)

10. Hardware Specification

The following sections provide details about the MPLAB ICD 5 unit and related hardware.

10.1 Debugger Unit

The MPLAB ICD 5 in-circuit debugger unit is rated as:

- Class A device in a laboratory environment.
- Category C device in a home or office environment.
- Operating Temperature: 0°C - 70°C.

10.2 Power Specifications

Unit power can be provided by:

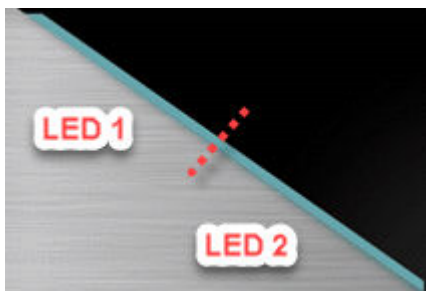
- Ethernet connection using Power over Ethernet (PoE). This can also provide power to the target up to 1A.
- USB-C connection. This can also provide power to the target, but the amount provided will depend on the PC host. When connected, the debugger will determine what can be provided. If <3A is available, MPLAB X IDE will alert the user that the full target power (3A) cannot be supplied.

Related Links

[3.1. Power and Self Test](#)

10.3 Indicator Lights (LEDs)

The top of MPLAB ICD 5 unit has two light pipes butted against to each other, each illuminated by an LED.



The expected start-up sequence for the debugger is:

1. Purple - steady on for approximately 3 seconds.
2. Blue - flashing for approximately 2 seconds while the debugger runs a power-on self-test.
3. Blue - steady on. The debugger is ready.

The following table advises how to read the indicator lights.

Table 10-1. LED and Bootloader Error Descriptions

LED 1	LED 2	Description
Normal Modes		
Blue	Blue	Power is connected; debugger in standby; Network ready to connect
White	Blue	Network connected
White, slow blink	Blue	MPLAB X IDE/MPLAB IPE has initiated communication with ICD 5 over the network

.....continued

LED 1	LED 2	Description
Red	Blue	Network connection failure/error
Yellow	Blue	Power target circuit from ICD 5 checked
Green	Blue	Power target circuit from ICD 5 unchecked
Green, slow blink	Blue	DGI connected
Purple	Purple	Bootloader is running
Yellow	Yellow	Debugger is busy
Red	Red	An operation has failed
Bootloader Errors		
Purple	Red, slow blink	Problem accessing the debugger's serial EEPROM
Purple	Red, fast blink	Bootloader API commands cannot be processed
White, fast blink	White, fast blink	A runtime exception occurred in the tool firmware

10.4 PC Connection Specifications

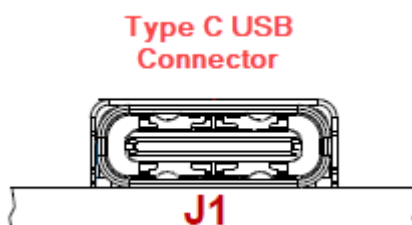
MPLAB® ICD 5 In-Circuit Debugger can be connected to the PC (and MPLAB X IDE/MPLAB IPE) using one of the following connection types. Connection speeds are also shown in the table.

Connection Type	Connection Details	Programming and Debugging	Trace
USB Type-C® (default)	HS USB 2.0	USB 2.0 up to 480 Mbps	
Ethernet	Direct or via network	up to 100 Mbps	No

Connectors available on the MPLAB ICD 5 unit for communication are described in the following sections.

10.4.1 USB Type-C® Connector (J1) and Cable

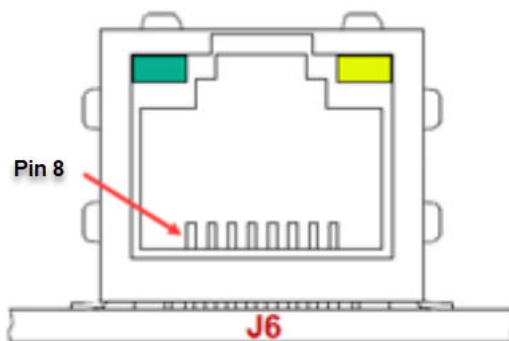
A USB Type-C connector and cable are provided for USB 2.0 communication between the debugger and a computer. It is recommended that you use the cable that comes with the kit to avoid communication issues.



10.4.2 Ethernet Connector (J6) and Cable

The 8-pin RJ-45 type connector and a standard Ethernet CAT5e/CAT6 cable enable Ethernet communications between the MPLAB ICD 5 unit the PC (and MPLAB X IDE/MPLAB IPE). The connector has two built-in LEDs that specify LAN activity. An Ethernet cable that does not have a anti-snag boot is preferred for best fit with the connector.

Note: Ethernet cable not provided with MPLAB ICD 5 kit.

Figure 10-1. Ethernet Connector on MPLAB ICD 5**Table 10-2.** Ethernet Connector Pinout

Pin Number	Function
1	TX+
2	TX-
3	RX+
4	EGND
5	EGND
6	RX-
7	EGND
8	EGND

EGND: Enclosure ground

Table 10-3. Connector LEDs

LED Location	LED Color	LED Function
Top left	Green	LAN ACT
Top right	Yellow	LAN LINK

10.5 8-pin Communication Hardware

For full debugger communication with a target, connect the included RJ-45 cable into the debugger modular RJ-45 jack at one end and an RJ-45 modular connector at the target end. Alternately plug the cable into the Debugger Adapter Board to have access to many device legacy connections. For details see *3.3 Target Connections*.

For details on the 8-pin RJ-45 modular connector and modular cable, see the following sections.

10.5.1 Modular Connector - RJ-45

The *ICD Tool* uses an RJ-45 modular connector and cable to communicate with a target.

The modular connector pins are always numbered in the same order regardless of connector orientation.

Figure 10-2. Modular Connector - RJ-45

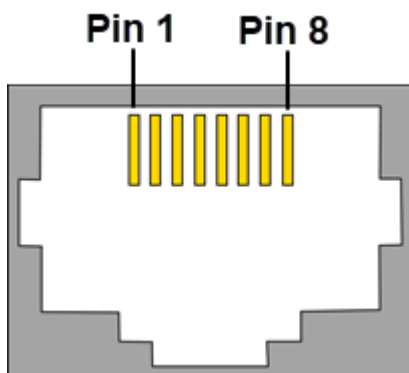
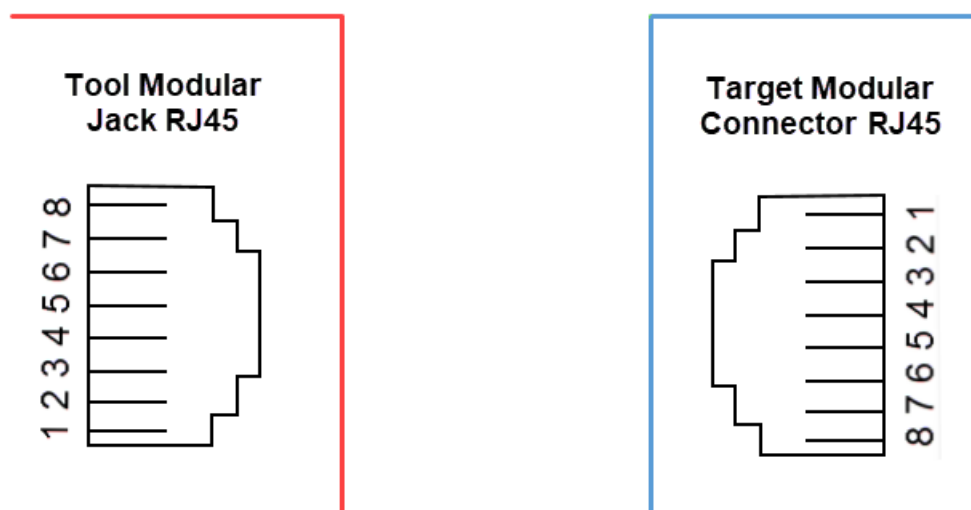


Table 10-4. Modular Connector Orientations

RJ-45 Jack (at ICD Tool)	RJ-45* Connector (at Target)
Bottom of Tool Circuit Board	Bottom of Target Board
* For information about connecting to an RJ-11 at the target, see <i>Connecting an RJ-11 Type Cable to an RJ-45 Jack on the Debugger</i>.	

Pin numbering can be seen in the following diagram. Although the connectors are oriented differently on the tool and on the target, pin 1 is always pin 1 with relation to the connector.

Figure 10-3. Modular Connectors

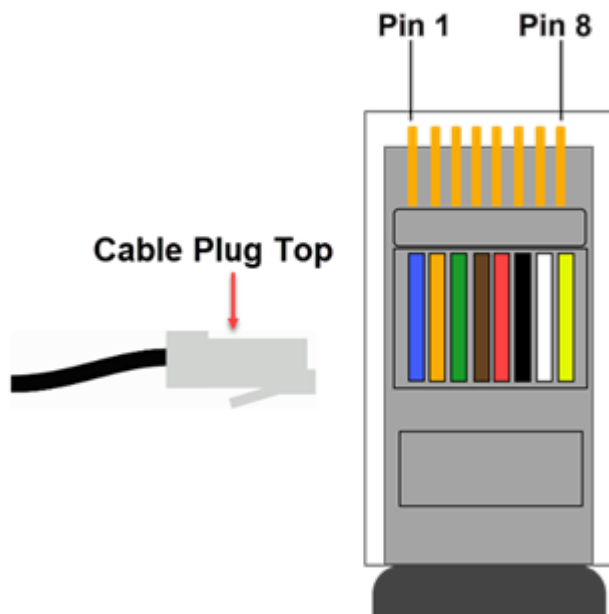


10.5.2 Modular Cable - RJ-45

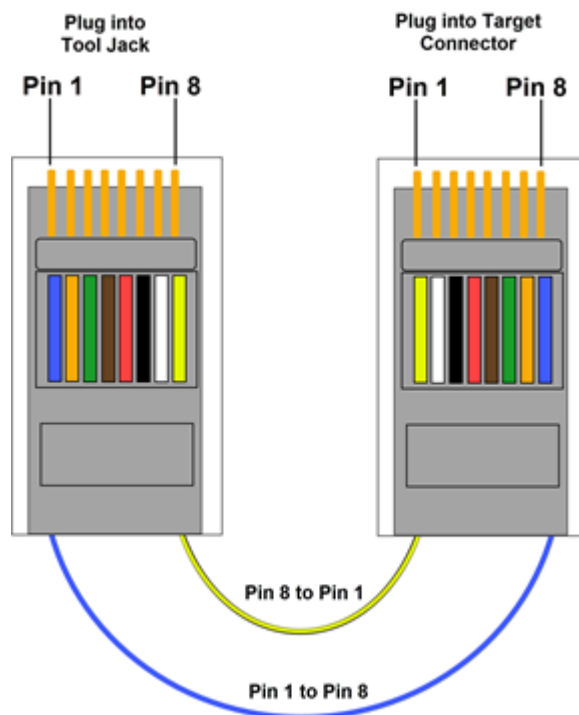
The *ICD Tool* uses an RJ-45 modular connector and cable to communicate with the target.

The modular cable plug pins are always numbered in the same order regardless of the plug orientation.

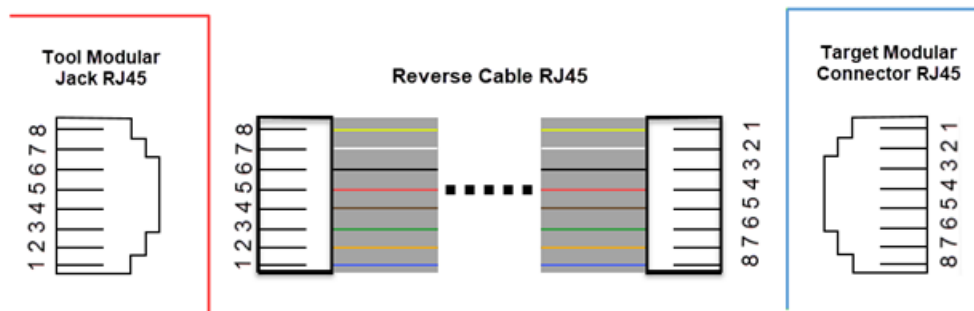
Figure 10-4. Modular (RJ-45) Cable Transparent Plug



The cable is reverse wiring based on the plugs view (see figure below.)

Figure 10-5. Reverse Wiring Cable

Pin numbering can be seen in the following diagram. Although the plugs are oriented differently at the tool and at the target, pin 1 is always pin 1 with relation to the plug.

Figure 10-6. Modular Connectors and Cable

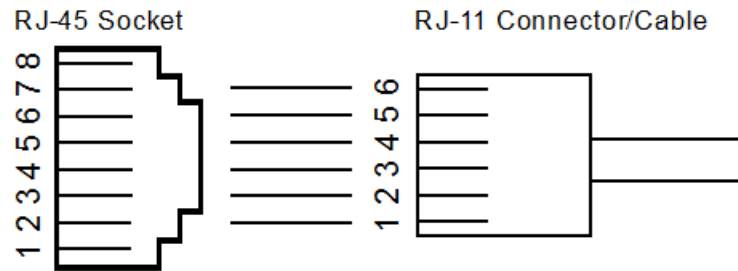
10.6 Communication Hardware

For standard debugger communication with a target, use an adapter with the RJ-11 connector.

10.6.1 Connecting an RJ-11 Type Cable to an RJ-45 Jack on the Debugger

The MPLAB ICD 5 In-Circuit Debugger has an RJ-45 connector for communication to the target. Connect the RJ-11 type cable into the RJ-45 connector by simply inserting it into the center of the RJ-45 connector.

Refer to the figure below for the pinouts for this connection.

Figure 10-7. RJ-45 Socket to RJ-11 Connector Pinout

Pin	RJ-45	Function	Pin	RJ-11
1	TMS	EJTAG Test Mode Select		
2		Reserved	1	
3	PGC (ICSPCLK)	Standard Com Clock/TCK (JTAG Test Clock)	2	PGC (ICSPCLK)
4	PGD (ICSPDAT)	Standard Com Data/TDO (JTAG Test Data Output)	3	PGD (ICSPDAT)
5	GND	Ground	4	GND
6	V _{DD_TGT}	Power on target	5	V _{DD_TGT}
7	V _{PP}	Power	6	V _{PP}
8	TDI	JTAG Test Data Input		

10.6.2 Standard Communication

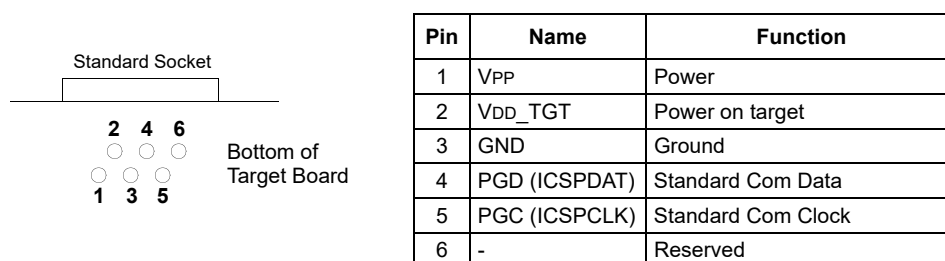
The main interface to the target processor is via standard communication. It contains the connections to the high voltage (V_{PP}), V_{DD} sense lines, as well as clock and data connections required for programming and connecting with the target devices.

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

The V_{DD} sense connection draws very little current from the target processor. The actual power comes from the MPLAB ICD 5 In-Circuit Debugger system, as the V_{DD} sense line is used as a reference only to track the target voltage. The V_{DD} connection is isolated with an optical switch.

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

- Clock and data signals are in high-impedance mode (even when no power is applied to the MPLAB ICD 5 In-Circuit Debugger system).
- Clock and data signals are protected from high voltages caused by faulty target systems, or improper connections.
- Clock and data signals are protected from high current caused from electrical shorts in faulty target systems.

Figure 10-8. 6-Pin Standard Pinout**Table 10-5.** Electrical Logic Table

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

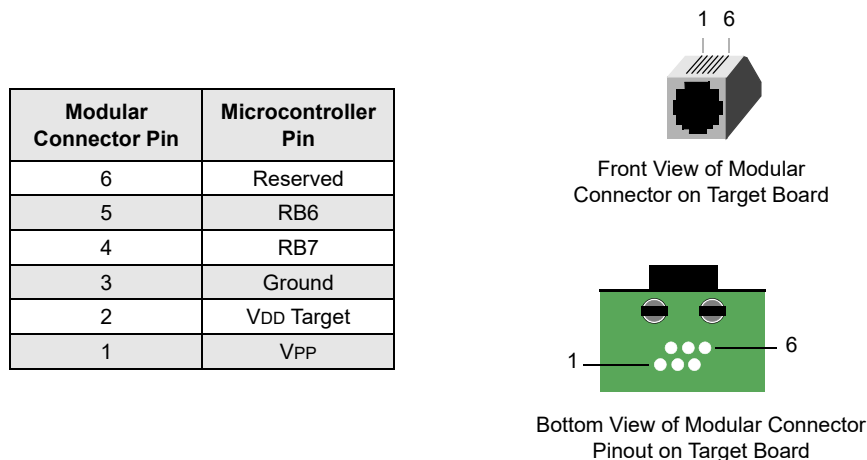
10.6.3 Modular Cable and Connector

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

10.6.3.1 Modular Connector Specification

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

The following table shows how the modular connector pins, for an application, correspond to the microcontroller pins. This configuration provides full in-circuit debugger functionality.

Figure 10-9. Modular Connector Pinout of Target Board

10.6.3.2 Modular Plug Specification

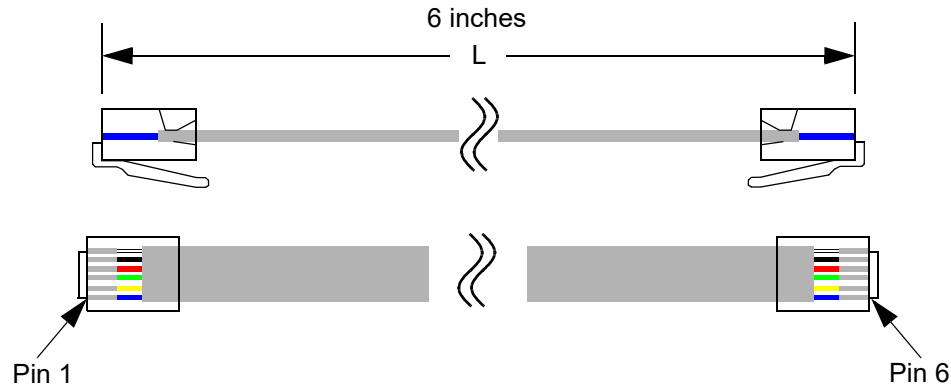
- Manufacturer, Part Number – AMP Incorporated, 5-554710-3

- Distributor, Part Number – Digi-Key, A9117ND

10.6.3.3 Modular Cable Specification

Manufacturer, Part Number – Microchip Technology, 07-00024. The length of this cable (L) is 6 inches. It is not recommended that you use a modular cable longer than 6 inches to avoid potential communication problems.

Figure 10-10. Modular Cable

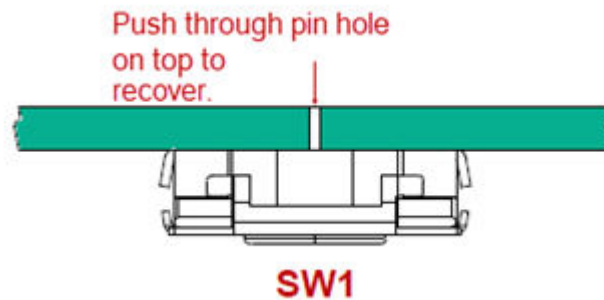


10.7 Recovery Specifications

The MPLAB ICD 5 unit can be placed in Recovery mode - device reset and flash erase - using a paper clip or similar tool through a hole in the bottom of the unit to activate the hardware reset switch (see image below).

The unit also can be placed in Recovery mode using software from the [Hardware Tool Emergency Boot Firmware Recovery](#).

Figure 10-11. Location of Recovery Switch



10.8 Target Board Considerations

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

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. There is a 182K ohm load on VDD_TGT.

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

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

11. Revision History

11.1 Revision A (May 2023)

Initial release of this document.

11.2 Revision B (November 2023)

- Additions to **About the Debugger > Components**:
 - Compatible USB isolators
 - Image of ICD 5 box components
 - Description of box accessories
- Post-initial release additions throughout the document that were non-gating for the release.

11.3 Revision C (July 2024)

- Additions to **Connections > PC Connections > Ethernet Modes**.
 - New subsection discussing APIPA
- Additions to **Connections > Target Connections > Adapter Board Pinout**.
 - Updated pinout board
- New graphics to demonstrate PoE, Ethernet connections.

12. Support

Please refer to the following sections for support issues.

12.1 Warranty Registration

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

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

12.2 myMicrochip Personalized Notification Service

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

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

www.microchip.com/pcn

A FAQ and registration details are available on the webpage.

Microchip Information

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:

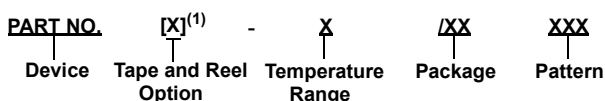
- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

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

Product Identification System

To order or obtain information, e.g., on pricing or delivery, refer to the factory or the listed sales office.



Device:	PIC16F18313, PIC16LF18313, PIC16F18323, PIC16LF18323	
Tape and Reel Option:	Blank	= Standard packaging (tube or tray)
	T	= Tape and Reel ⁽¹⁾
Temperature Range:	I	= -40°C to +85°C (Industrial)
	E	= -40°C to +125°C (Extended)
Package: ⁽²⁾	JQ	= UQFN
	P	= PDIP
	ST	= TSSOP
	SL	= SOIC-14
	SN	= SOIC-8
	RF	= UDFN
Pattern:	QTP, SQTP, Code or Special Requirements (blank otherwise)	

Examples:

- PIC16LF18313- I/P Industrial temperature, PDIP package
- PIC16F18313- E/SS Extended temperature, SSOP package

Notes:

1. Tape and Reel identifier only appears in the catalog part number description. This identifier is used for ordering purposes and is not printed on the device package. Check with your Microchip Sales Office for package availability with the Tape and Reel option.
2. Small form-factor packaging options may be available. Please check www.microchip.com/packaging for small-form factor package availability, or contact your local Sales Office.

Microchip Devices Code Protection Feature

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

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

Legal Notice

This publication and the information herein may be used only with Microchip products, including to design, test, and integrate Microchip products with your application. Use of this information in any other manner violates these terms. Information regarding device applications is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. Contact your local Microchip sales office for

additional support or, obtain additional support at www.microchip.com/en-us/support/design-help/client-support-services.

THIS INFORMATION IS PROVIDED BY MICROCHIP "AS IS". MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE, OR WARRANTIES RELATED TO ITS CONDITION, QUALITY, OR PERFORMANCE.

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

Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Trademarks

The Microchip name and logo, the Microchip logo, Adaptec, AVR, AVR logo, AVR Freaks, BesTime, BitCloud, CryptoMemory, CryptoRF, dsPIC, flexPWR, HELDO, IGLOO, JukeBlox, KeeLoq, Kleer, LANCheck, LinkMD, maXStylus, maXTouch, MediaLB, megaAVR, Microsemi, Microsemi logo, MOST, MOST logo, MPLAB, OptoLyzer, PIC, picoPower, PICSTART, PIC32 logo, PolarFire, Prochip Designer, QTouch, SAM-BA, SenGenuity, SpyNIC, SST, SST Logo, SuperFlash, Symmetricom, SyncServer, Tachyon, TimeSource, tinyAVR, UNI/O, Vectron, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

AgileSwitch, APT, ClockWorks, The Embedded Control Solutions Company, EtherSynch, Flashtec, Hyper Speed Control, HyperLight Load, Libero, motorBench, mTouch, Powermite 3, Precision Edge, ProASIC, ProASIC Plus, ProASIC Plus logo, Quiet- Wire, SmartFusion, SyncWorld, Temux, TimeCesium, TimeHub, TimePictra, TimeProvider, TrueTime, and ZL are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, AnyIn, AnyOut, Augmented Switching, BlueSky, BodyCom, Clockstudio, CodeGuard, CryptoAuthentication, CryptoAutomotive, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, Espresso T1S, EtherGREEN, GridTime, IdealBridge, In-Circuit Serial Programming, ICSP, INICnet, Intelligent Paralleling, IntelliMOS, Inter-Chip Connectivity, JitterBlocker, Knob-on-Display, KoD, maxCrypto, maxView, memBrain, Mindi, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, PowerSmart, PureSilicon, QMatrix, REAL ICE, Ripple Blocker, RTAX, RTG4, SAM-ICE, Serial Quad I/O, simpleMAP, SimpliPHY, SmartBuffer, SmartHLS, SMART-I.S., storClad, SQL, SuperSwitcher, SuperSwitcher II, Switchtec, SynchroPHY, Total Endurance, Trusted Time, TSHARC, USBCheck, VariSense, VectorBlox, VeriPHY, ViewSpan, WiperLock, XpressConnect, and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

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

The Adaptec logo, Frequency on Demand, Silicon Storage Technology, and Symmcom are registered trademarks of Microchip Technology Inc. in other countries.

GestIC is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

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

© 2024, Microchip Technology Incorporated and its subsidiaries. All Rights Reserved.

ISBN: 978-1-6683-4852-9

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 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: www.microchip.com/support Web Address: www.microchip.com	Australia - Sydney Tel: 61-2-9868-6733 China - Beijing Tel: 86-10-8569-7000 China - Chengdu Tel: 86-28-8665-5511 China - Chongqing Tel: 86-23-8980-9588 China - Dongguan Tel: 86-769-8702-9880 China - Guangzhou Tel: 86-20-8755-8029 China - Hangzhou Tel: 86-571-8792-8115 China - Hong Kong SAR Tel: 852-2943-5100 China - Nanjing Tel: 86-25-8473-2460 China - Qingdao Tel: 86-532-8502-7355 China - Shanghai Tel: 86-21-3326-8000 China - Shenyang Tel: 86-24-2334-2829 China - Shenzhen Tel: 86-755-8864-2200 China - Suzhou Tel: 86-186-6233-1526 China - Wuhan Tel: 86-27-5980-5300 China - Xian Tel: 86-29-8833-7252 China - Xiamen Tel: 86-592-2388138 China - Zhuhai Tel: 86-756-3210040	India - Bangalore Tel: 91-80-3090-4444 India - New Delhi Tel: 91-11-4160-8631 India - Pune Tel: 91-20-4121-0141 Japan - Osaka Tel: 81-6-6152-7160 Japan - Tokyo Tel: 81-3-6880-3770 Korea - Daegu Tel: 82-53-744-4301 Korea - Seoul Tel: 82-2-554-7200 Malaysia - Kuala Lumpur Tel: 60-3-7651-7906 Malaysia - Penang Tel: 60-4-227-8870 Philippines - Manila Tel: 63-2-634-9065 Singapore Tel: 65-6334-8870 Taiwan - Hsin Chu Tel: 886-3-577-8366 Taiwan - Kaohsiung Tel: 886-7-213-7830 Taiwan - Taipei Tel: 886-2-2508-8600 Thailand - Bangkok Tel: 66-2-694-1351 Vietnam - Ho Chi Minh Tel: 84-28-5448-2100	Austria - Wels Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 Denmark - Copenhagen Tel: 45-4485-5910 Fax: 45-4485-2829 Finland - Espoo Tel: 358-9-4520-820 France - Paris Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79 Germany - Garching Tel: 49-8931-9700 Germany - Haan Tel: 49-2129-3766400 Germany - Heilbronn Tel: 49-7131-72400 Germany - Karlsruhe Tel: 49-721-625370 Germany - Munich Tel: 49-89-627-144-0 Fax: 49-89-627-144-44 Germany - Rosenheim Tel: 49-8031-354-560 Israel - Ra'anana Tel: 972-9-744-7705 Italy - Milan Tel: 39-0331-742611 Fax: 39-0331-466781 Italy - Padova Tel: 39-049-7625286 Netherlands - Drunen Tel: 31-416-690399 Fax: 31-416-690340 Norway - Trondheim Tel: 47-72884388 Poland - Warsaw Tel: 48-22-3325737 Romania - Bucharest Tel: 40-21-407-87-50 Spain - Madrid Tel: 34-91-708-08-90 Fax: 34-91-708-08-91 Sweden - Gothenberg Tel: 46-31-704-60-40 Sweden - Stockholm Tel: 46-8-5090-4654 UK - Wokingham Tel: 44-118-921-5800 Fax: 44-118-921-5820