MPLAB ICE 4 In-Circuit Emulator User's Guide

MPLAB ICE 4 In-Circuit Emulator User's Guide

Notice to Development Tools Customers

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

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

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

Table of Contents

1. Preface

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

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

Description	Represents	Examples
Ellipses...	Replaces repeated text	`var_name [, var_name...]`

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

1.2 Recommended Reading

This document describes how to use the MPLAB® ICE 4 In-Circuit Emulator. 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® ICE 4 In-Circuit Emulator with your target design. Refer to the following

Developer Help: Development Tools Design Advisory (most up-to-date)

PDF: *Multi-Tool Design Advisory* (DS51764)

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 (DS50002027) available as a free download at www.microchip.com/mplabx/.

Release Notes for MPLAB® ICE 4 In-Circuit Emulator

For the latest information on using MPLAB ICE 4, 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® ICE 4 In-Circuit Emulator Quick Start Guide Poster (DS50003240)

This poster shows you how to connect the hardware and install the software for the MPLAB ICE 4 using a target board.

2. About the Emulator

The MPLAB ICE 4 in-circuit emulator/programmer (DV244140) is Microchip's latest fast and feature-rich emulation and programming tool for Microchip microcontrollers (MCUs), which include PIC, dsPIC, AVR and SAM 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 ICE 4 connects to your PC through a high-speed USB 3.0/2.0 interface. However, you can also connect using Wi-Fi or Ethernet.

The MPLAB ICE 4 connects to targets using a high-speed edge rate cable, connected at one end to the emulator, and at the other to adapter boards tailored for supported device communication.

The emulator communications 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 ICE 4 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
- Capability to incorporate I/O port data input
- Advanced host communication interfaces (Windows, Linux, and macOS)
- Advanced communication mediums and protocols
- Faster programming times
- Modular design (testability and maintainability)

In addition to emulation functions, the MPLAB ICE 4 system also may be used as a device production programmer.

2.1 Advantages

The MPLAB ICE 4 In-Circuit Emulator system provides the following advantages:

Features/Capabilities:

- Connects to a computer via high-speed USB 3.0/2.0, Ethernet or Wi-Fi.
- Connects to new targets using an edge rate high speed coax cable assembly. Connects to legacy targets using included adapter boards.
- Programs devices using MPLAB X IDE or MPLAB IPE.
- Supports multiple breakpoints, stopwatch, and source code file debugging.
- Debugs your application on your own hardware in real time.
- Sets breakpoints based on internal events.
- Monitors internal file registers.
- Debugs at full speed.
- Configures pin drivers.
- 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/).
- Controls brightness of LEDs.
- Operates within a temperature range of 0-70 degrees Celsius.

Performance/Speed:

- More and faster memory.
- A Real-Time Operating System (RTOS).
- No firmware download delays incurred when switching devices.
- A 32-bit MCU running at 300 MHz.

- A buffer memory of 4 MB.

Safety:

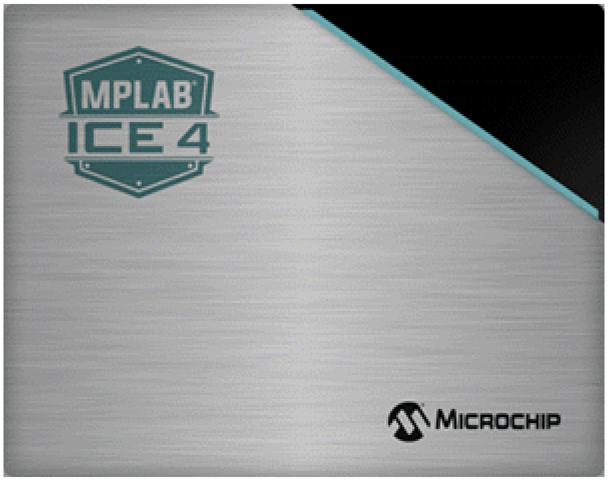
- Receive feedback from debugger when external power supply is needed for target.
- Supports target supply voltages from 1.2V to 5.5V.
- Safely power up to 1A with the 9V DC power supply.
- Protection circuits are added to the probe drivers to guard from power surges from the target.
- V_{DD} and V_{PP} 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.) RED tested.

2.2 Components

The components of the MPLAB ICE 4 In-Circuit Emulator kit box are:

- The rectangular MPLAB ICE 4 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, and communication and debug connectors.
- A Type C USB cable for default computer-to-emulator communication.
- A 9V power supply to power the unit.
- A 6-inch high-speed edge rate cable for connectivity to target devices.
- Various adapter boards and associated cables for support of legacy target connections. Find these in a small box attached under the kit box tray, i.e., **pull the tray out** to find the adapter boards and cables box.

Figure 2-1. Emulator Unit



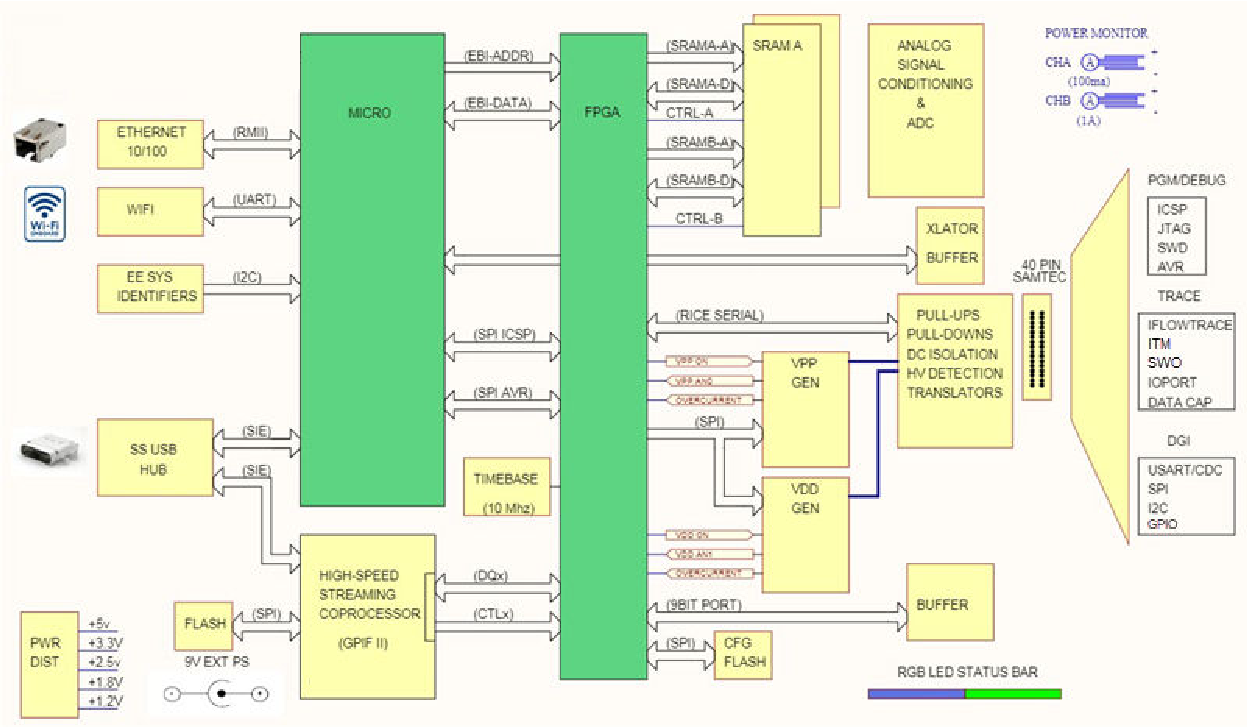
Additional hardware and accessories may be ordered separately from the Microchip Purchasing and Client Services website (www.microchipdirect.com).

- 9V Wall Mount Power Supply (Part Number AC002014). For details, see 10.2. Power Specifications.

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

2.3 Block Diagram

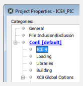
Below is a block diagram of basic MPLAB ICE 4 unit operational capabilities.



2.4 MPLAB ICE 4 and MPLAB X IDE

The MPLAB® ICE 4 In-Circuit Emulator works with the MPLAB X Integrated Development Environment (IDE) to develop target applications.

Download and install the latest version of MPLAB X IDE from the MPLAB X IDE webpage. The user's guide and other documentation may also be found on the webpage. Once the IDE is installed:

MPLAB X IDE MPLAB X IDE vX.XX	Use the desktop icon to launch the IDE.
	Create a new project or open an existing project. Select MPLAB ICE 4 as the hardware tool.
 Project Properties - ICE4_PIC Categories: General File Inclusion/Exclusion Conf: [default] ICE 4 Loading Libraries Building XC8 Global Options	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 9.2. Emulator Options Selection.

3. Connections

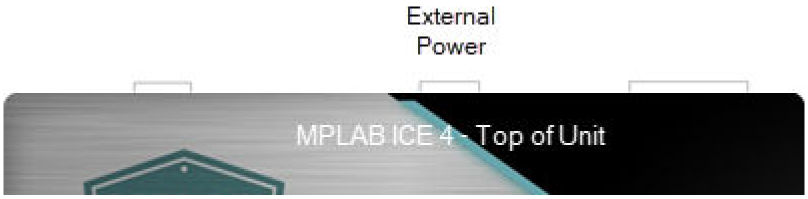
The MPLAB® ICE 4 In-Circuit Emulator hardware setup begins by connecting power, communications, and targets to the emulator. For legacy targets, adapter boards and cables are provided.

3.1 Power and Self Test

An external 9V power supply (AC002014) is **required** to power the MPLAB® ICE 4 In-Circuit Emulator and optionally the target. This power supply comes with the emulator kit. Connect the 3-pronged plug into a power socket and the tip end into the top of the emulator as shown in the figure below.

Note: USB cannot power the MPLAB ICE 4 unit.

Figure 3-1. MPLAB ICE 4 Power



Power-up Self Test

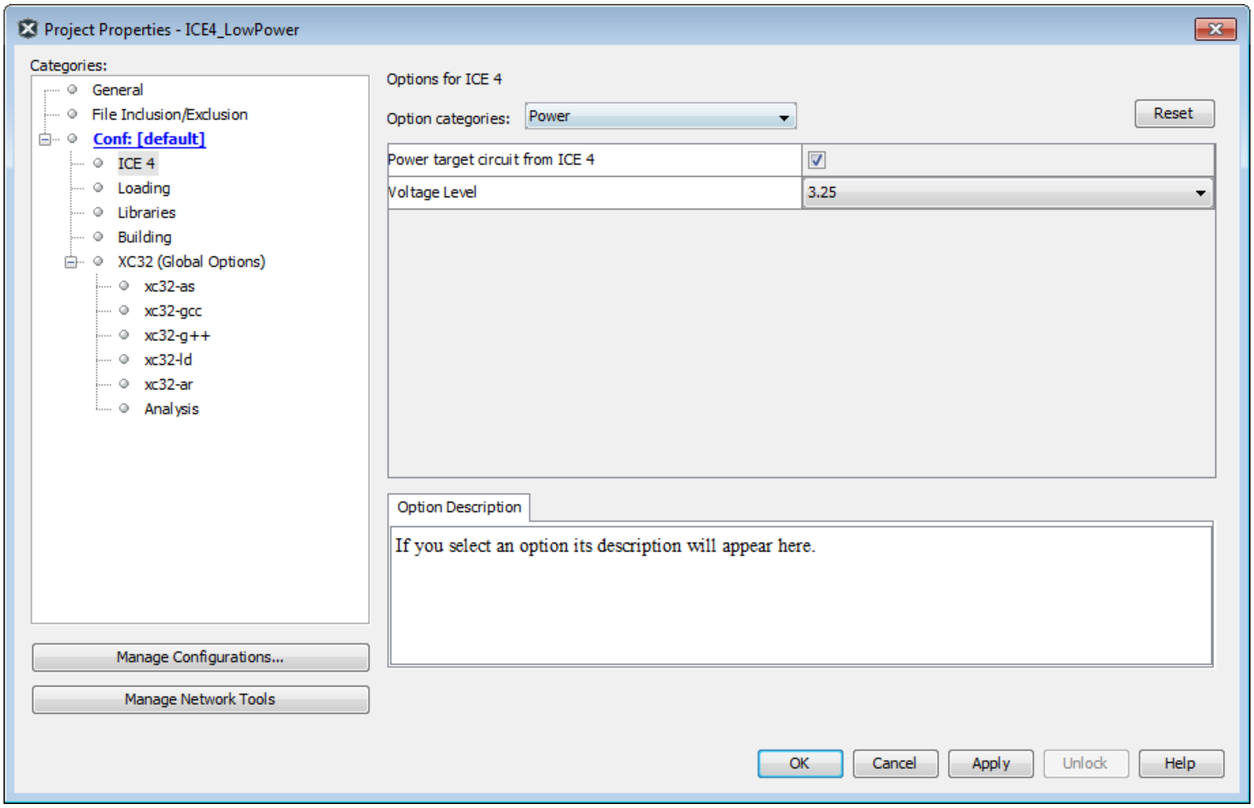
The MPLAB ICE 4 unit preforms a built-in self-test (or BIST) during power-up. Errors that occur during this test are reported in the MPLAB X IDE Output window. Depending on the error, LED colors may indicate the error as well.

Power the Target

It is possible to power the target using the emulator. 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 3-2. Select Power to Target



Option categories: Power

Power target circuit from ICE 4	☑
Voltage Level	3.25

If you select an option its description will appear here.

Related Links

10.2. Power Specifications

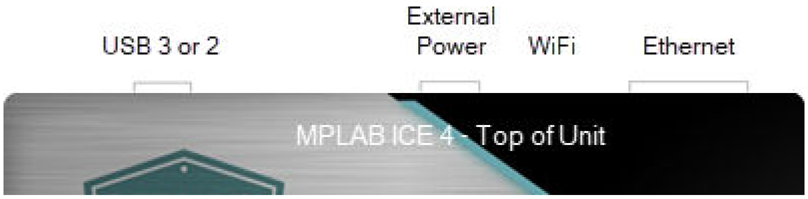
8. Error Messages

10.3. Indicator Lights (LEDs)

3.2 PC Connections

MPLAB® ICE 4 In-Circuit Emulator can connect with the PC (and MPLAB X IDE) using USB3.0/2.0 (default), Wi-Fi or Ethernet. For details on these connections, see 10.4. PC Connection Specifications.

Figure 3-3. MPLAB ICE 4 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 Wi-Fi or Ethernet communication instead of USB has several uses:

- Access a target remotely. The emulator 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 ICE 4 unit is USB using a USB type C cable.

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

Note: Only USB communications can be used for trace.

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 Wi-Fi or Ethernet again.

3.2.2 Wi-Fi and Ethernet - Modes

MPLAB® ICE 4 In-Circuit Emulator supports different modes of Wi-Fi and Ethernet communication.

3.2.2.1 Wi-Fi Access Point (WiFi-AP)

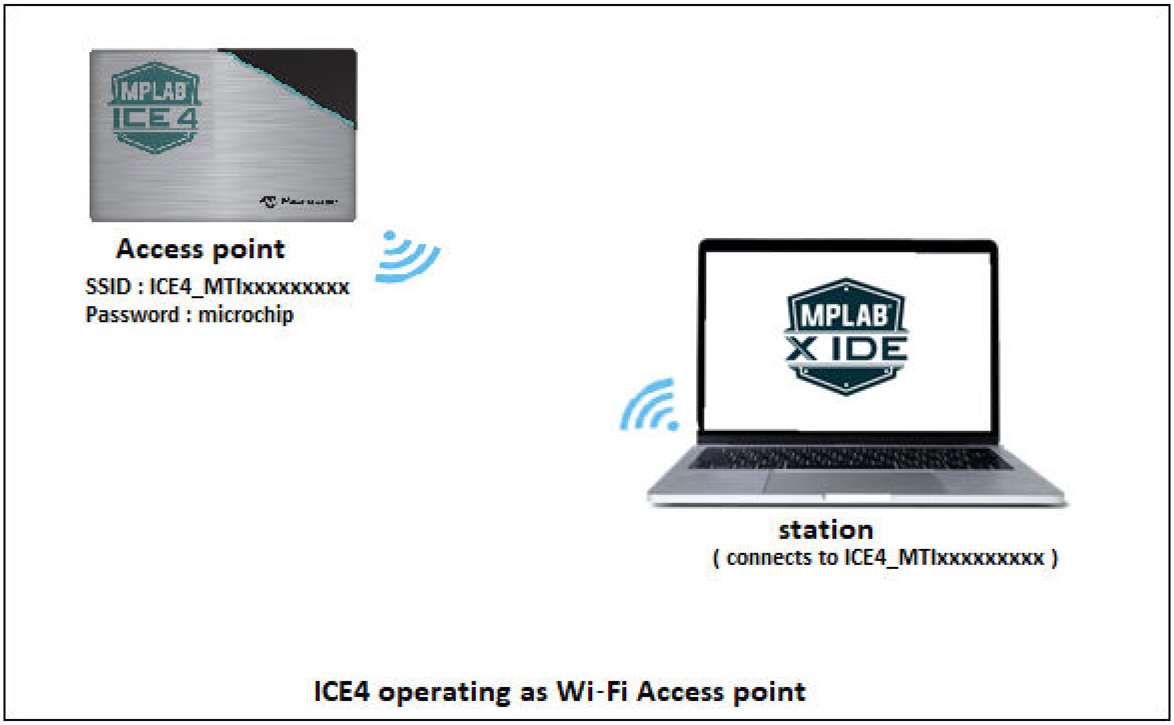
Wi-Fi Access Point (WiFi-AP) mode allows the emulator to create its own network (like a router) and have other devices (e.g., a laptop) connect to it. Use this mode to access the emulator and target via a PC within the range of the Wi-Fi signal.

MPLAB ICE 4 comes up with a preset fixed SSID and Password:

- SSID: "ICE4_MTIxxxxxxxxx" (where xxxxxxxxx is your tool unique serial number)
- password: "microchip"

Note: When MPLAB ICE 4 is connected in WiFi-AP mode, computers with Windows OS may disconnect from other Wi-Fi connections.

Figure 3-4. MPLAB ICE 4 Operating as Access Point

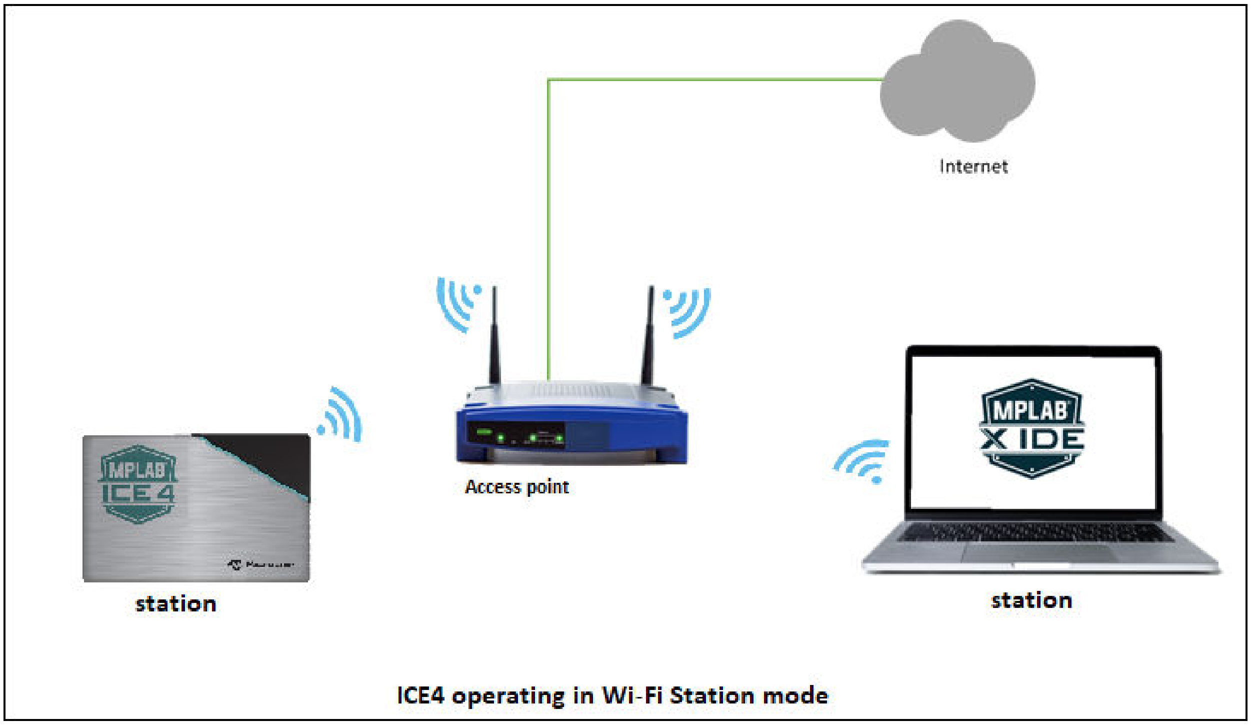


3.2.2.2 Wi-Fi Station Mode (WiFi-STA)

Wi-Fi Station (WiFi-STA) mode tries to connect to your home/office network using its access point router parameters (SSID and password) provided by the user in MPLAB X IDE. Use this mode to provide greater distance/isolation then WiFi-AP between the emulator and target and a PC.

Note: MPLAB ICE 4 supports 2.4 GHz, but **not** 5.0 GHz.

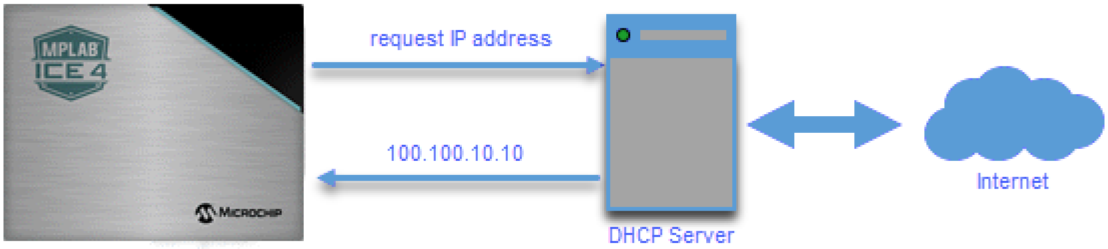
Figure 3-5. MPLAB ICE 4 Operating in Station Mode



3.2.2.3 Ethernet Wired/DHCP/APIPA

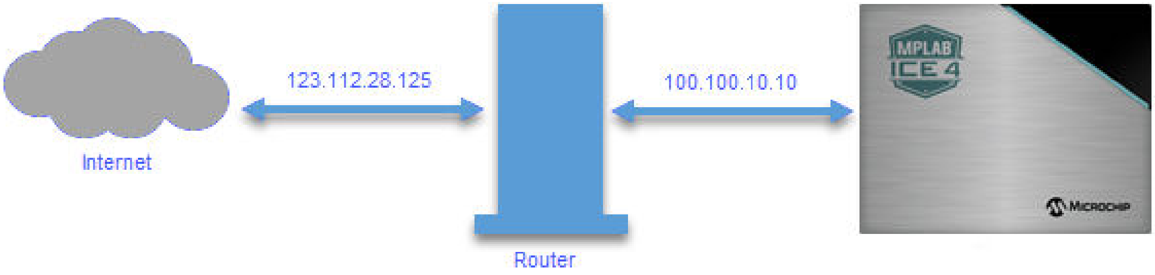
When connecting MPLAB® ICE 4 In-Circuit Emulator to the Ethernet, request a DHCP (or APIPA) IP address.

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



3.2.2.4 Ethernet Static IP

A Static IP address is one that is permanently assigned to your network devices.



3.2.3 Wi-Fi and Ethernet - Setup and Tool Discovery

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

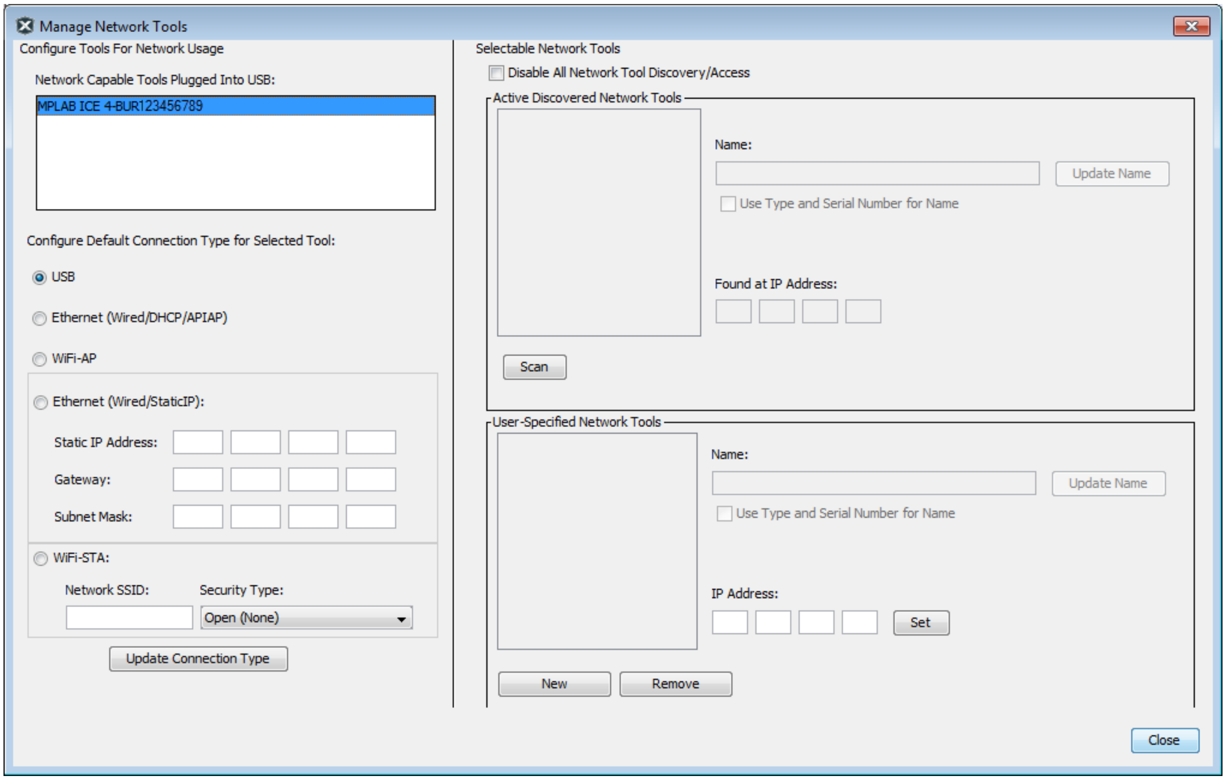
Table 3-1. Ethernet or Wi-Fi Setup and Tool Discovery in MPLAB X IDE

Step	Action
1	Connect the emulator to your PC via the USB cable.
2	Go to *Tools> Manage Network Tools* in MPLAB X IDE (see figure below).
3	Under "Network Capable Tools Plugged into USB," select your emulator.
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. **Wi-Fi STA**: Input SSID, Security type and password, depending on the security type of your home/office router. Click **Update Connection Type**.
5	Unplug the USB cable from your emulator unit.
6	The emulator will restart automatically and come up in the connection mode you selected. Then either: **All Except Wi-Fi AP**: The LEDs will display for either a successful network connection or a network connection failure/error. **Wi-Fi AP**: The normal Wi-Fi scanning process of Windows OS / macOS / Linux OS will scan for available Wi-Fi networks on your PC. Find the tool with SSID "ICE4_MTIxxxxxxxxx" (where xxxxxxxxx is your tool unique serial number) and use the password "microchip" to connect to it.

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

Step	Action
7	Now go back to the "Manage Network Tools" dialog and click on the **Scan** button, which will list your emulator under "Active Discovered Network Tools." Select the checkbox for your tool and close the dialog. **Wi-Fi AP**: On Windows 10 computers, you may see the message “No Internet, Secured” and yet the button will say “Disconnect” showing that there is a connection. This message means that the emulator is connected as a router/AP but not by direct connection (Ethernet.)
8	If your emulator 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).

Figure 3-6. Initial USB Connection



Network Capable Tools Plugged Into USB: MPLAB ICE 4-BUR123456789

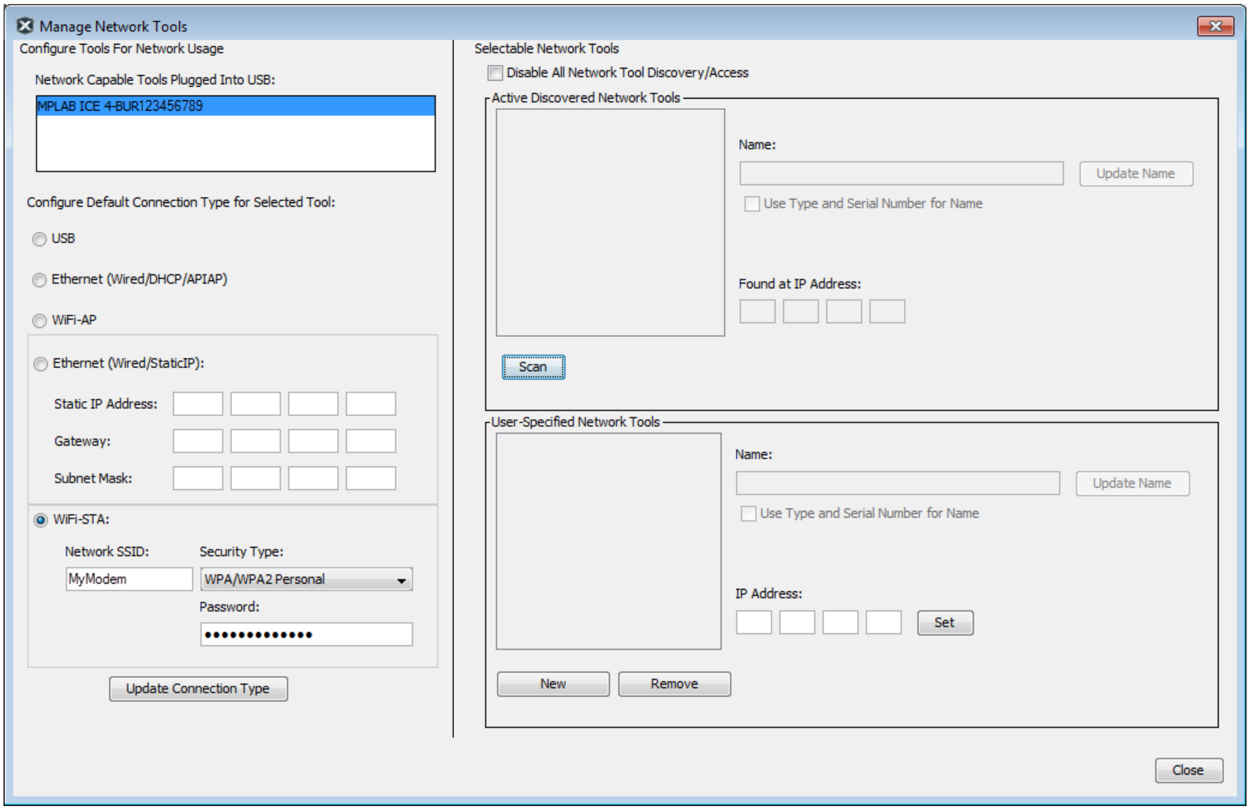
Configure Default Connection Type for Selected Tool:

● USB
○ Ethernet (Wired/DHCP/APIAP)
○ WiFi-AP
○ Ethernet (Wired/StaticIP):
○ WiFi-STA:

Security Type: Open (None)

Manage Network Tools
Configure Tools For Network Usage
Static IP Address:
Gateway:
Subnet Mask:
Network SSID:
Update Connection Type
Selectable Network Tools
□ Disable All Network Tool Discovery/Access
Active Discovered Network Tools
Name:
Update Name
□ Use Type and Serial Number for Name
Found at IP Address:
Scan
User-Specified Network Tools
Name:
Update Name
□ Use Type and Serial Number for Name
IP Address:
Set
New
Remove
Close

Figure 3-7. WiFi-STA Example



Manage Network Tools

Configure Tools For Network Usage

Network Capable Tools Plugged Into USB: MPLAB ICE 4-BUR123456789

Configure Default Connection Type for Selected Tool:

○ USB

○ Ethernet (Wired/DHCP/APIAP)

○ WiFi-AP

○ Ethernet (Wired/StaticIP):

● WiFi-STA:

Network SSID: MyModem

Security Type: WPA/WPA2 Personal

Password: •••••••••••••

Selectable Network Tools

□ Disable All Network Tool Discovery/Access

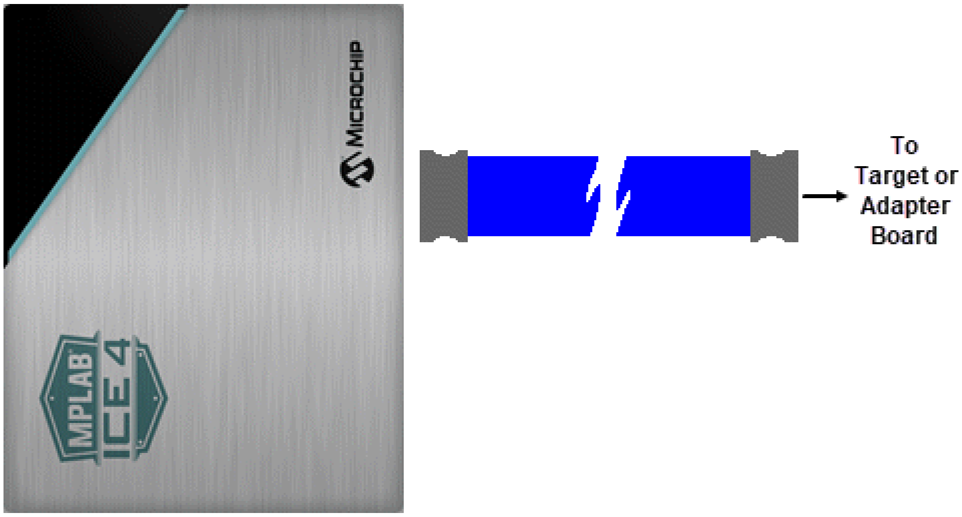
3.3 Target Connections

MPLAB® ICE 4 In-Circuit Emulator connects to a target via a high-speed 40-pin ribbon cable assembly. For legacy target connections, optional adapter boards are available. There is also a Current Sense connection for use when Power Debugging.

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

Note: MPLAB ICE 4 can power the target. For details, see 10.2. Power Specifications. Select powering the target in the Project Properties window, “ICE 4” category, “Power” option category.

Figure 3-8. MPLAB ICE 4 Unit to Target Connection



3.3.1 Target Connection Pinout

The table below shows the pinout of the target end of the cable connected from the MPLAB ICE 4 unit to either an adapter board or a 40-pin connector on the target board.

Table 3-2. Emulator Connector Pin Functions on Target

Pin	Description	Function(s)	Pin	Description	Function(s)
1	CS- A	Power Monitor	2	CS+ A	Power Monitor
3	CS- B	Power Monitor	4	CS+ B	Power Monitor
5	UTIL SDA	Reserved	6	UTIL SCL	Reserved
7	DGI SPI nCS	DGI SPI nCS,PORT6, TRIG6	8	DGI SPI SCK	DGI SPI SCK, SPI SCK, PORT7, TRIG7
9	DGI SPI MOSI	DGI SPI MOSI, SPI DATA, PORT5, TRIG5	10	DGI SPI MISO	DGI SPI MISO, PORT4, TRIG4
11	3V3	Reserved	12	GND	GND
13	DGI GPIO3	DGI GPIO3, PORT3, TRIG3	14	TRCLK	TRCLK, TRACECLK,
15	DGI GPIO2	DGI GPIO2, PORT2, TRIG2	16	GND	GND
17	DGI GPIO1	DGI GPIO1, PORT1, TRIG1	18	TRDAT3	TRDAT3, TRACEDATA(3)
19	DGI GPIO0	DGI GPIO0, PORT0, TRIG0	20	GND	GND
21	5V0	Reserved	22	TRDAT2	TRDAT2, TRACEDATA(2)
23	DGI VCP RXD	DGI RXD, CICD RXD, VCD RXD	24	GND	GND
25	DGI VCP TXD	DGI TXD, CICD TXD, VCD TXD	26	TRDAT1	TRDAT1, TRACEDATA(1)
27	DGI I2C SDA	DGI I2C SDA	28	GND	GND
29	DGI I2C SCL	DGI I2C SCL	30	TRDAT0	TRDAT0, TRACEDATA(0)
31	TV_{DD} PWR	TV_{DD} PWR	32	GND	GND
33	TDI IO	TDI IO, TDI, MOSI	34	TMS IO	TMS IO, SWD IO, TMS
35	TPGC IO	TPGC IO, TPGC, SWCLK, TCK, SCK	36	TAUX IO	TAUX IO, AUX, DW, RESET

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

Pin	Description	Function(s)	Pin	Description	Function(s)
37	TV_{PP} IO	TV_{PP}/MCLR, nMCLR, RST	38	TPGD IO	TPGD IO, TPGD, SWO,TDO, MISO, DAT
39	TV_{DD} PWR	TV_{DD} PWR	40	TV_{DD} PWR	TV_{DD} PWR

Related Links

10.5.1. 40-pin Emulator Unit Connector (J10)

10.5.2. Target Connection Cable

3.3.2 Adapter Boards

For new target designs, it is recommended that a 40-pin connector be used to directly interface with the MPLAB ICE 4 cable assembly, providing the best debugging experience. However for legacy target connections, adapter boards are available in a Adapter Pack (AC244140). Use of adapter boards will likely degrade debug performance.

Table 3-3. Available Adapter Boards

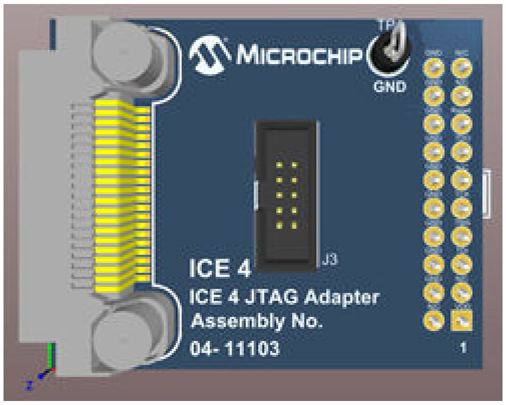
Adapter Board	Device Support	Feature Support
ICE 4 JTAG Adapter	SAM MCUs	Program, Debug
MPLAB ICE 4 ICSP Adapter Board + • ICE 4 AVR JTAG (10 Pin) • ICE 4 AVR (10 Pin Mini) • ICE 4 AVR (6 Pin) • ICE 4 AVR (6 Pin Mini)	AVR MCUs	Program, Debug
MPLAB ICE 4 ICSP Adapter Board	PIC MCUs, dsPIC DSCs	Program, Debug, Instrumented Trace: • Native Trace • SPI Trace • IO Port Trace (needs additional connections)
MPLAB ICE 4 Cortex-M Trace Adapter Board	SAM MCUs	Trace: • ETM • ETB • ITM
MPLAB ICE 4 PIC32 Trace Adapter Board	PIC32M MCUs	PIC32 Instruction Trace

3.3.3 SAM MCUs - JTAG/SWD Adapter Board

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

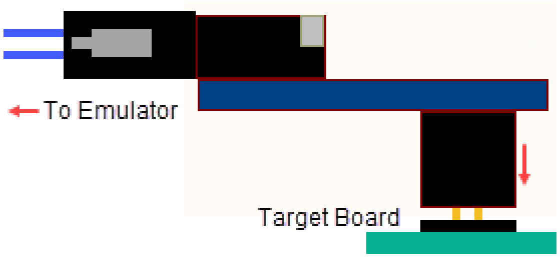
An “ICE 4 JTAG Adapter Board” with a 10-pin (top) or 20-pin (bottom) connector is available for legacy target connections.

Figure 3-9. JTAG/SWD Adapter Board



The adapter board may be plugged into a target board 20-pin connector as shown in the figure.

Figure 3-10. Plug Adapter into Target Board

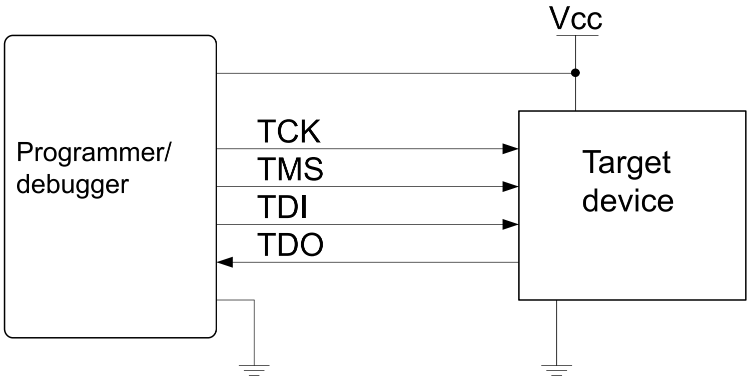


3.3.3.1 JTAG Physical Interface

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, “ICE 4” category, “Communications” option category, and select either “2-wire JTAG” or “4-wire JTAG.” See 9.2. Emulator Options Selection.

Figure 3-11. JTAG Interface Basics



3.3.3.1.1 Connecting to a SAM JTAG Target

The MPLAB ICE 4 using the ICE 4 JTAG Adapter Board 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 3.3.3.1.2. 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.

3.3.3.1.2 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 3-12. SAM JTAG Header Pinout

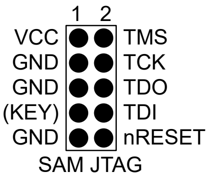


Table 3-4. SAM JTAG Pin Description

Name	Pin	Description
TCK	4	Test Clock (clock signal from the MPLAB ICE 4 into the target device)
TMS	2	Test Mode Select (control signal from the MPLAB ICE 4 into the target device)
TDI	8	Test Data In (data transmitted from the MPLAB ICE 4 into the target device)
TDO	6	Test Data Out (data transmitted from the target device into the MPLAB ICE 4)
nRESET	10	Reset (optional). Not used by MPLAB ICE 4.
VTG	1	Target voltage reference. Not used by MPLAB ICE 4.
GND	3, 5, 9	Ground. All must be connected to ensure that the MPLAB ICE 4 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 pin 1 and GND.

3.3.3.2 SAM SWD Interface

The Arm® SWD interface is a subset of the JTAG interface, making use of TCK and TMS pins. See 3.3.3.1.2. SAM JTAG Pinout (Cortex-M debug connector).

MPLAB ICE 4 is capable of streaming UART-format ITM trace to the host computer. For details see 5.7.2.1. SAM ITM Trace.

3.3.3.2.1 Connecting to a SAM SWD Target

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

3.3.3.2.2 SAM SWD Pinout

For the 10-pin JTAG connector:

Figure 3-13. SAM SWD Header Pinout

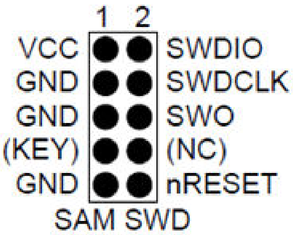


Table 3-5. SAM SWD Pin Description

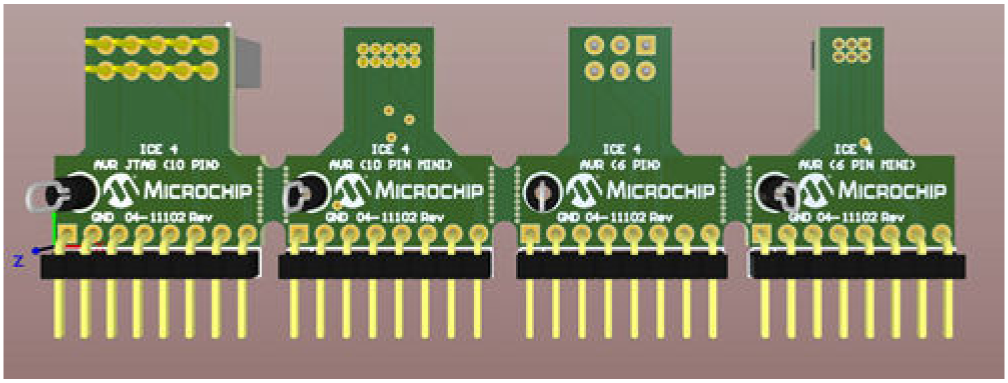
Name	SAM Port Pin	Description
SWDCLK	4	Serial Wire Debug Clock
SWDIO	2	Serial Wire Debug Data Input/Output
SWO	6	Serial Wire Output (optional- not implemented on all devices)
nSRST	10	Reset - not used by MPLAB ICE 4
VTG	1	Target voltage reference - not used by MPLAB ICE 4
GND	3, 5, 9	Ground

3.3.4 AVR MCUs - Adapter Boards

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

The MPLAB ICE 4 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.

Figure 3-14. AVR JTAG Plugin Boards

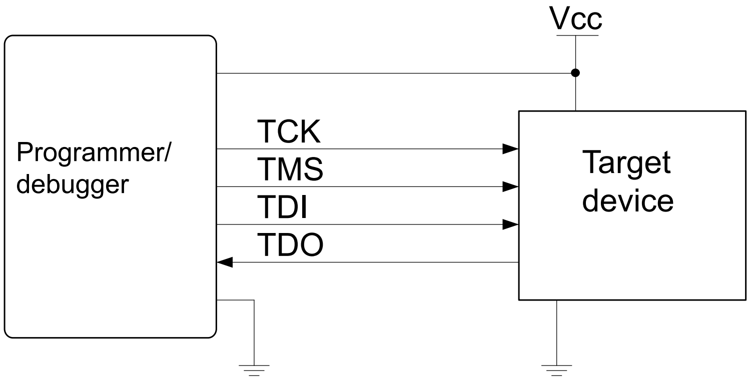


3.3.4.1 JTAG Physical Interface

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, “ICE 4” category, “Communications” option category, and select either “2-wire JTAG” or “4-wire JTAG.” See 9.2. Emulator Options Selection.

Figure 3-15. JTAG Interface Basics



3.3.4.1.1 Connecting to a AVR JTAG Target

The MPLAB ICE 4 using the MPLAB ICE 4 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 shown in 3.3.4.1.2. AVR JTAG Pinout.

3.3.4.1.2 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 4.2. AT Devices - On-Chip Debugging (OCD).

Figure 3-16. AVR JTAG Header Pinout

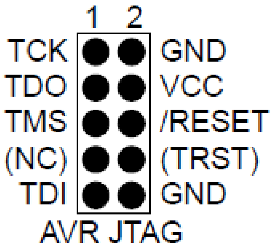


Table 3-6. AVR JTAG Pin Description

Name	Pin	Description
TCK	1	Test Clock (clock signal from the MPLAB ICE 4 into the target device).
TMS	5	Test Mode Select (control signal from the MPLAB ICE 4 into the target device).
TDI	9	Test Data In (data transmitted from the MPLAB ICE 4 into the target device).
TDO	3	Test Data Out (data transmitted from the target device into the MPLAB ICE 4).
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 ICE 4 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 ICE 4 samples the target voltage on this pin in order to power the level converters correctly. The MPLAB ICE 4 draws less than 1mA from this pin in this mode.
GND	2, 10	Ground. All must be connected to ensure that the MPLAB ICE 4 and the target device share the same ground reference.

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

Name	Pin	Description
* Remember to include a decoupling capacitor between pin 1 and GND.		

3.3.4.2 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.

3.3.4.2.1 Connecting to an AVR SPI Target

The recommended pinout for the 6-pin SPI connector is shown in 3.3.4.2.2. 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.

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.

3.3.4.2.2 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 3-17. SPI Header Pinout

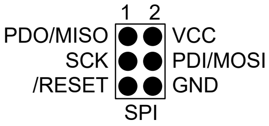


Table 3-7. SPI Pin Mapping

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)		

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

AVR PORT Pins	Target Pins	SPI Pinout
Pin 8 (nTRST)		
Pin 9 (TDI)	MOSI	4
Pin 10 (GND)		

3.3.4.3 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.

3.3.4.3.1 Connecting to an AVR PDI Target

The recommended pinout for the 6-pin PDI connector is shown in 3.3.4.3.2. 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.

3.3.4.3.2 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 3-18. PDI Header Pinout

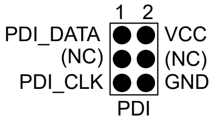


Table 3-8. PDI Pin Mapping

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)		

3.3.4.4 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.

3.3.4.4.1 Connecting to an AVR UPDI Target

The recommended pinout for the 6-pin UPDI connector is shown in 3.3.4.4.2. 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.

3.3.4.4.2 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 3-19. UPDI Header Pinout

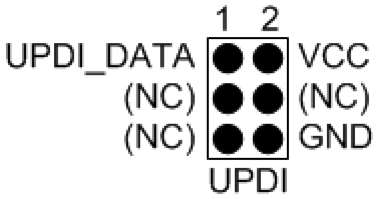


Table 3-9. UPDI Pin Mapping

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

3.3.4.5 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.

3.3.4.5.1 Connecting to an AVR TPI Target

The recommended pinout for the 6-pin TPI connector is shown in 3.3.4.5.2. 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.

3.3.4.5.2 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 3-20. TPI Header Pinout

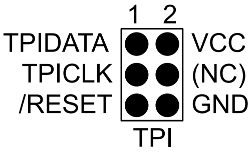


Table 3-10. TPI Pin Mapping

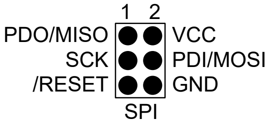
AVR PORT Pins	Target Pins	TPI Pinout
Pin 1 (TCK)	CLOCK	3
Pin 2 (GND)	GND	6
Pin 3 (TDO)	DATA	1
Pin 4 (VTG)	VTG	2
Pin 5 (TMS)		
Pin 6 (nSRST)	/RESET	5
Pin 7 (not connected)		
Pin 8 (nTRST)		
Pin 9 (TDI)		
Pin 10 (GND)		

3.3.4.6 AVR debugWIRE

The debugWIRE interface was developed by Atmel 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 bi-directional half-duplex asynchronous communication with the debugger tool.

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

Figure 3-21. debugWIRE (SPI) Header Pinout



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 3.3.4.2.2. 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) bi-directional I/O pin with pull-up enabled and becomes the communication gateway between target and debugger.

3.3.5 PIC MCUs - ICSP Adapter Board

The MPLAB ICE 4 In-Circuit Emulator supports debug and programming of PIC microcontrollers (MCUs) and dsPIC digital signal controllers (DSCs) through the ICSP™ (In-Circuit Serial Programming™) interface. In addition to debug, this connection supports other emulation features, including PIC Instrumented Trace.

The “MPLAB ICE 4 ICSP Adapter Board” with both 6- or 8-pin RJ-45 and 8-pin SIL connectors is available for legacy target connections.

Figure 3-22. ICSP Adapter Board

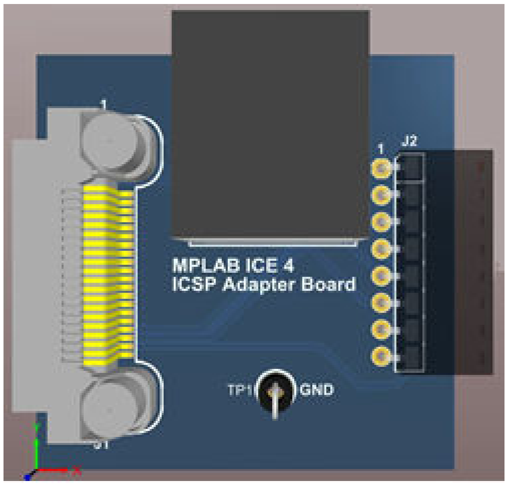


Figure 3-23. Modular Connection to Target

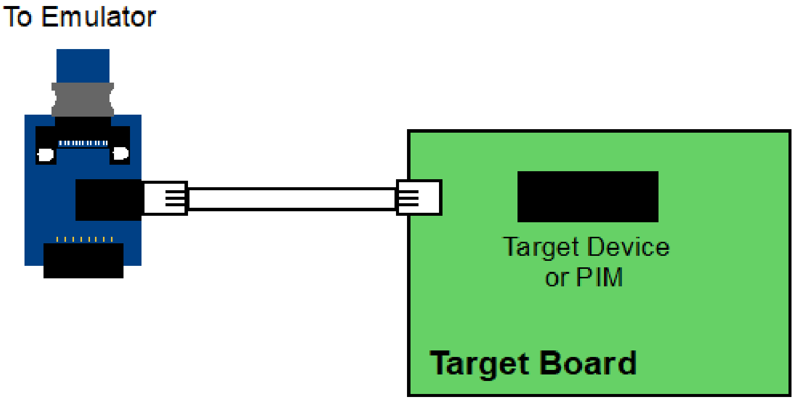
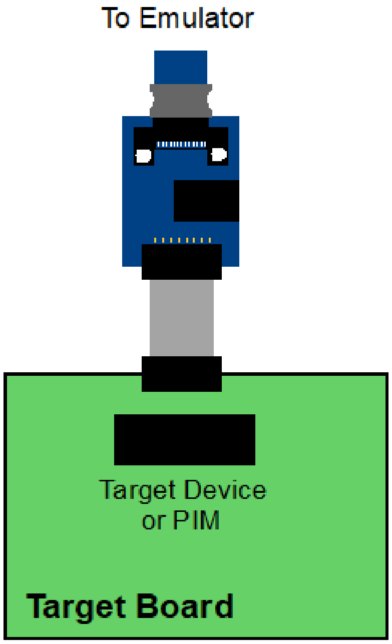


Figure 3-24. SIL Connection to Target



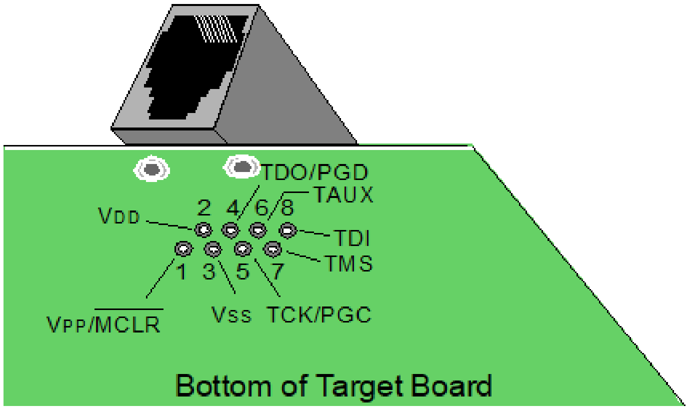
3.3.5.1 ICSP Target Connection

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

Using the RJ-45 connector, the MPLAB ICE 4 In-Circuit Emulator is connected to the target device with a modular interface (6 or 8 conductor) cable. If the 6-pin cable is used, connections 7 (TMS) and 8 (TDI) are not available. The pin numbering for the connector is shown from the bottom of the target PCB in the figure below.

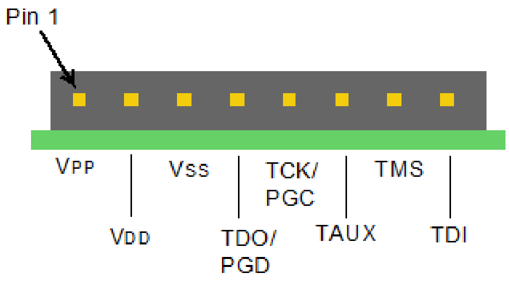
Note: Cable connections on the debugger and target are mirror images of each other, that is, pin 1 on one end of the cable is connected to pin 6 or 8 on the other end of the cable.

Figure 3-25. Standard Connection at Target



Using the Single In-Line (SIL) connector, the MPLAB ICE 4 In-Circuit Emulator is connected to the target device with a ribbon interface (6 or 8 conductor) cable. If the 6-pin cable is used, connections 7 (TMS) and 8 (TDI) are not available. The pin numbering for the connector is shown from the target PCB in the figure below.

Figure 3-26. SIL Connection at Target



3.3.5.2 ICSP Target Connection Circuitry

The figure below shows the interconnections of the MPLAB ICE 4 In-Circuit Emulator to the connector on the target board. The diagram also shows the wiring from the connector to a device on the target PCB. A pull-up resistor Rpu, usually around 10 kΩ, connected from the VPP/MCLR line to the VDD is recommended so that the line may be strobed low to reset the device.

Figure 3-27. ICSP Connection Target Circuitry

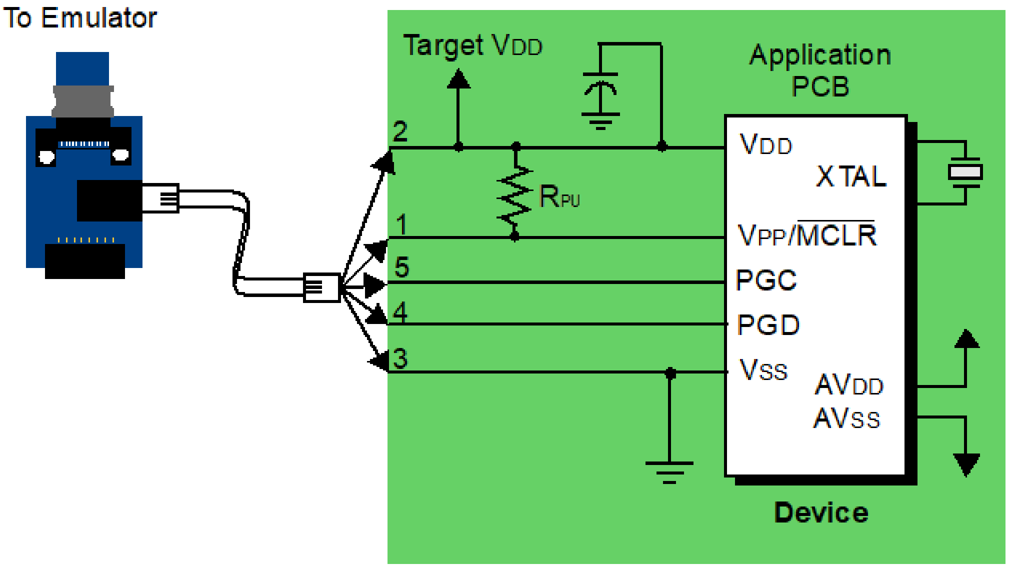


Table 3-11. Target Connector and Related Device I/O

Pin No.	Device I/O	Description
1	Vpp/MCLR	The emulator requires access to Vpp for programming and debugging the device.
2	Vdd	Target Vdd is sensed by the emulator to allow level translation for target low-voltage operation and to detect a device. If the emulator does not sense voltage on its Vdd line, it will not connect with the device.
3	Vss	Target Vss is sensed by the emulator. **Note:** The emulator DOES NOT provide target Vss or ground.
4	PGD	The emulator requires access to PGD and PGC for programming and debugging the device.
5	PGC	

Table 3-12. Circuitry on Device I/O

Device I/O	Description
Vpp/nMCLR and Vdd	A pull-up resistor (10kΩ typical) should be connected from the Vpp/MCLR line to Vdd so that the line may be strobed low to reset the device.
XTAL	The target device must be running with an oscillator for the emulator to function as a debugger.
AVdd, AVss	Not all devices have the AVdd and AVss lines, but if they are present on the target device, all must be connected to the appropriate levels in order for the emulator to operate. This also applies to voltage regulator pins (e.g., ENVREG/DISVREG on PIC24FJ MCUs).
Vdd, Vss, AVdd, AVss	In general, it is recommended per device data sheet that all Vdd/AVdd and Vss/AVss lines be connected to the appropriate levels. For devices with a Vcap pin (e.g., PIC18FXXJ devices), the appropriately-valued capacitor should be placed as close to the Vcap pin as possible.

3.3.5.3 ICSP Target Connection and Power

There are two methods for externally powering the target: from an external power supply or from the emulator.

In the following descriptions, only three lines are active and relevant to core debugger operation: pins 1 (VPP/MCLR), 5 (PGC), and 4 (PGD). Pins 2 (VDD) and 3 (VSS) are shown in Standard Connection Target Circuitry for completeness.

When providing power to the target device, ensure that the target is not exposed to voltages higher than the device VDD rating.

Absolute maximum ratings for the device VDD must not be exceeded. Exposure to the maximum rating conditions for any length of time may affect device reliability.

Functional operation of the device at conditions above the parameters indicated in the device data sheets specification is not recommended.

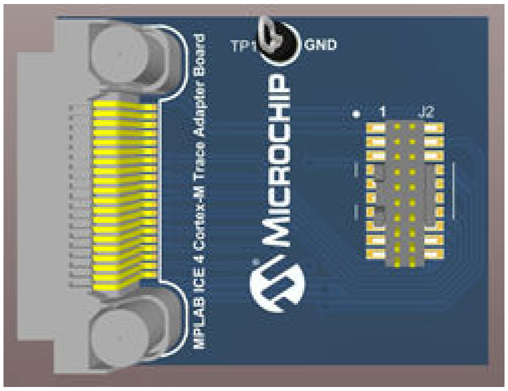
See the device data sheet for required device voltage levels and maximum ratings.

3.3.6 SAM MCUs - Cortex-M Trace Adapter Board

SAM MCUs/MPUs are based on Arm® Cortex®-M processors. Each of these processor families have different types of trace. These will be discussed in the sections below.

The “MPLAB ICE 4 Cortex-M Trace Adapter Board” with a 20-pin mini JTAG connector is available for legacy target connections.

Figure 3-28. SAM (Cortex M) Trace Adapter Board



Related Links

5.7. SAM ITM/SWO Trace

3.3.6.1 SAM ITM Trace

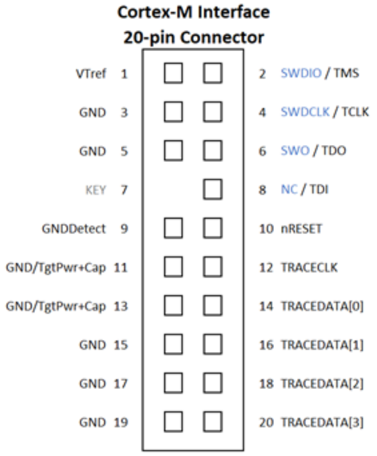
For system trace the processor integrates an Instrumentation Trace Macrocell (ITM) alongside data watchpoints and a profiling unit. ITM trace is found on selected SAM microcontrollers (Cortex®-M3/M4).

One pin of Trace Port Interface Unit (TPIU) will be specified for Serial Wire Output (SWO).

3.3.6.2 SAM Trace Physical Connection

The adapter board supports CoreSight™ 20 Instruction trace.

ITM/SWO is supported as follows: SWDIO and SWDCLK are used for debug and SWO is used for ITM output.



3.3.6.3 SAM Trace Target Connection

The CoreSight20 connector can be used in either standard JTAG (IEEE 1149.1) mode or Serial Wire Debug (SWD) mode. It can also optionally capture up to four bits of parallel trace in TPIU continuous mode.

Table 3-13. CoreSight20 JTAG Connector Pins

Pin	Signal name	Description
1	VTREF	VCC_TARGET_P3V3
2	TMS	Inactive
3	GND	–
4	TCK	Inactive
5	GND	–
6	TDO	Inactive
9	GND	–
10	nSRST	Synchronous Microcontroller Reset

Table 3-14. CoreSight20 Serial Wire Debug (Asynchronous Trace) Pins

Pin	Signal name	Description
2	SWDIO	The Serial Wire Data I/O pin sends and receives serial data to and from the target during debugging. You are advised to series terminate SWDIO close to the target processor.
3	GND	–
4	SWCLK	The Serial Wire Clock pin clocks data into and out of the target during debugging.
5	GND	–
6	TRACESWO	The Serial Wire Output pin can be used to provide trace data to the IDE. You are advised to series terminate SWO close to the target processor.
9	GND	–

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

Pin	Signal name	Description
10	nSRST	Synchronous Microcontroller Reset.

Table 3-15. CoreSight20 Parallel Trace Source Pins

Pin	Signal name	Description
12	TRACECLK	The Trace Clock pin provides the IDE with the clock signal necessary to sample the trace data signals. TRACECLK is PCK3.
13	–	–
14	TRACED0	The Trace Data [0–3] pins provide the IDE with TPIU continuous mode trace data from the target.
15	Ground	–
16	TRACED1	–
17	Ground	–
18	TRACED2	–
19	Ground	–
20	TRACED3	–

3.3.7 PIC Instrumented Trace Adapter Board

Depending on your selected PIC MCU device, one or more trace capabilities (Native, SPI or Port trace) may be available when the emulator is selected as the debug tool. The MPLAB ICE 4 ICSP Adapter Board is available to support this type of trace for legacy target connections.

Related Links

5.5. Instrumented Trace for PIC MCUs and dsPIC DSCs

3.3.7.1 Native Trace Connections

To use Native trace, use the 3.3.5. PIC MCUs - ICSP Adapter Board. The communications connection will carry the trace information using the PGD/PGC/EMUC/EMUD pins. However, the selected device must have this feature. If it does not, one of the other trace methods may be used.

For more on this type of trace, see 5.5.3.1. Native Trace.

3.3.7.2 SPI Trace Connections

To use SPI trace, use the 3.3.5. PIC MCUs - ICSP Adapter Board with an 8-pin connection. The communications connection will carry the trace information using the TMS/TDI pins. However, the selected device must have the SPI feature. If it does not, one of the other trace methods may be used.

For more on this type of trace, see 5.5.3.2. SPI Trace.

3.3.7.3 IO Port Trace Connections

To use IO Port trace, you will need to use the 40-pin cable to connect to the target and NOT the 3.3.5. PIC MCUs - ICSP Adapter Board. Parallel trace is possible using selected pins of the high-speed cable as an IO Port.

For this trace configuration, seven (7) lines of data and one (1) line for clock are transmitted. PORTx must be a port with 8 pins that has all 8 pins available for trace. The port does not have to be one physical port but can be made up of pins from more than one port. The port pins must not be multiplexed with the currently-used PGC and PGD pins.

For allowable PORTx configurations:

1. Right click on your project in the Projects window and select “Properties.”
2. In the Project Properties window, click on the “ICE4” category.
3. Select the “Trace and Profiling” option category from the drop down list.

4. Under “Data Collection Selection,” select the trace that is supported for your device, e.g., “User Instrumented Trace.”
5. Under “Communications Medium,” select “I/O Port.”
6. Under “I/O Port Selection,” select your port configuration from the list.

A basic configuration is shown in the following table.

Note: External triggers uses the same target pins.

Target Connector Pin	Pin Name	Content
8	PORT7[1]	Clock
7	PORT6	Data
9	PORT5	Data
10	PORT4	Data
13	PORT3	Data
15	PORT2	Data
17	PORT1	Data
19	PORT0	Data

[1] Use a 10 kW pull-down resistor for noise reduction.

As described in 4.3.2.1. Circuits That Will Prevent the Emulator From Functioning, do not use pull-up or pull-down resistors, capacitors or diodes on port pins, except as specified.

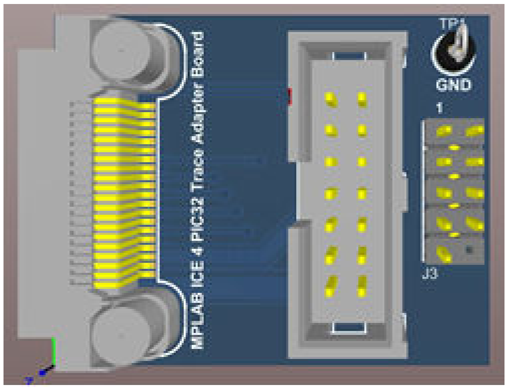
For more on this type of trace, see 5.5.3.3. I/O Port Trace.

3.3.8 PIC32 Instruction Trace Adapter Board

PIC32 Instruction Trace is only available for PIC32M MCU devices. Also, only some PIC32M MCU devices have the trace feature. Consult your device data sheet for details.

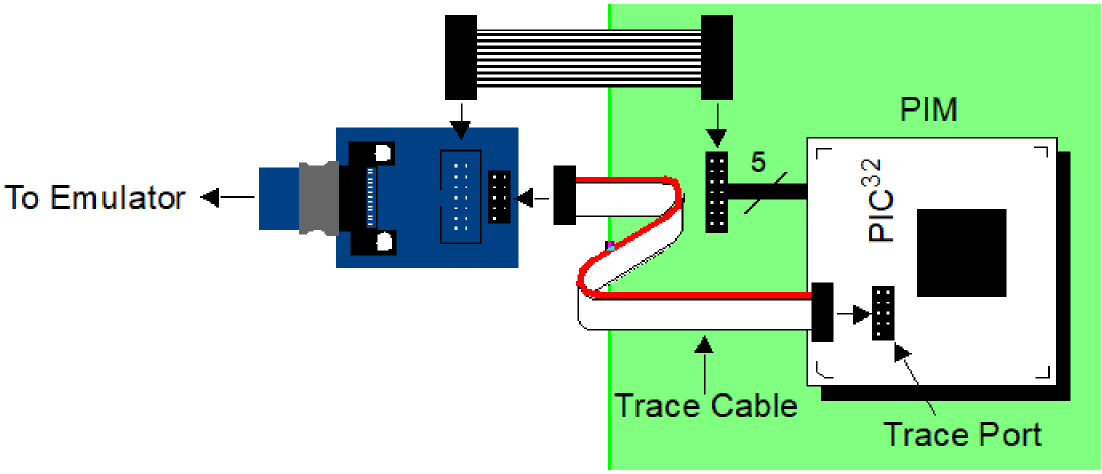
The “MPLAB ICE 4 PIC32 Trace Adapter Board” is available for legacy target connections.

Figure 3-29. PIC32 Instruction Trace Adapter Board



The adapter board can be plugged into the target board via the 14-pin EJTAG connector to access debug pins from a PIM or device. Then the 9-pin Trace connector can be used to connect to the Trace port on top of the PIM or a device port pin connector.

Figure 3-30. Adapter Board to PIC32 PIM



Related Links

5.6. Instruction Trace for PIC32M MCUs

3.3.8.1 PIC32 Instruction Trace - 14-pin EJTAG Connection

Below are tables for the 14-pin EJTAG connection on the adapter board and at the target. The trace port connections are discussed in the following topics.

Table 3-16. Adapter Board and Target Connection Pinout

Pin	Function	Pin	Function
1	NC	2	GND
3	TDI	4	GND
5	TPGD	6	GND
7	TMS	8	GND
9	TPGC	10	GND
11	TVPP	12	Key
13	NC	14	TVDD_PWR

3.3.8.2 PIC32 Instruction Trace - PIM Support

Only PIC32 Plug-In Modules (PIMs) that have a trace connector support PIC32 Instruction Trace. See the table below. For more information on available PIMs, see the Microchip website.

Table 3-17. List of PIC32 PIMs

PIM No.	PIM Name	Trace Connector
MA320001	PIC32MX 100 P QFP to 100 P PIM	✓
MA320002	PIC32MX USB PIM	✓
MA320002-2	PIC32MX450/470 100-pin to 100-pin USB PIM	✓
MA320003	PIC32 CAN-USB PIM	✓

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

PIM No.	PIM Name	Trace Connector
MA320011	PIC32MX 1xx/2xx PIM	✗
MA320012	PIC32MZ2048EC 100-100 P PIM	✓
MA320013	PIC32MX270F256 PIM for Bluetooth Audio Development Kit	✗
MA320014	PIC32MX270F256 PIM	✗
MA320015	PIC32MX570F512L USB/CAN Explorer 16 PIM	✓
MA320016	PIC32MZ Audio 144 P PIM for Bluetooth Audio Development Kit	✓
MA320017	PIC32MX270F512L PIM for Bluetooth Audio Development Kit	✓
MA320018	PIC32MZ EF Audio 144-pin PIM for Bluetooth Audio Development Kit	✓
MA320019	PIC32MZ EF PIM	✓
MA320020	PIC32MM0064GPL036 General Purpose PIM	✗
MA320021	PIC32MX254F256 PIM for Explorer 16	✗
MA320023	PIC32MM0256GPM064 General Purpose PIM	✗
MA320024	PIC32MK1024 Motor Control Plug In Module	✗

3.3.8.3 PIC32 Instruction Trace - PIM Trace Connection

Whether connecting the adapter board to a header on a PIC32M target board or to the trace port on a PIC32M PIM, the pin configuration should be the same.

Figure 3-31. PIC32 PIM Diagram

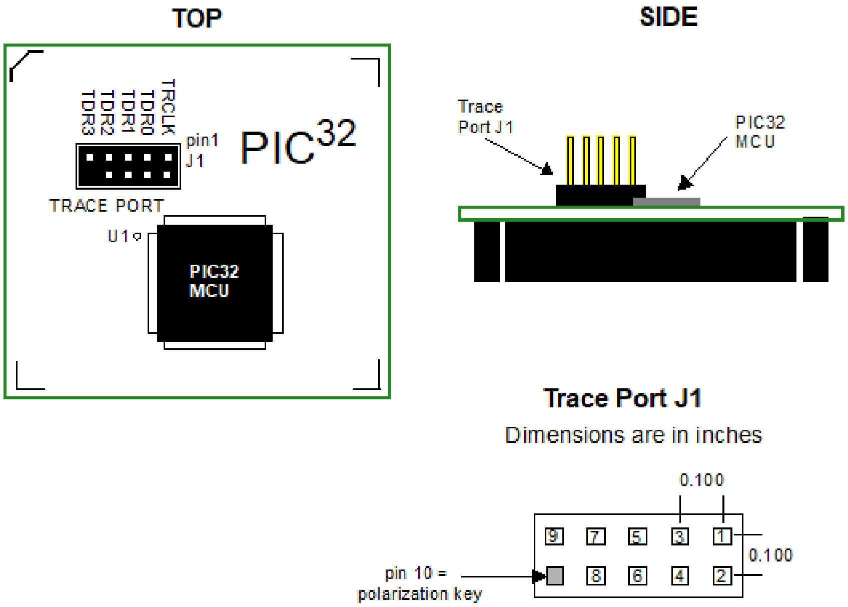
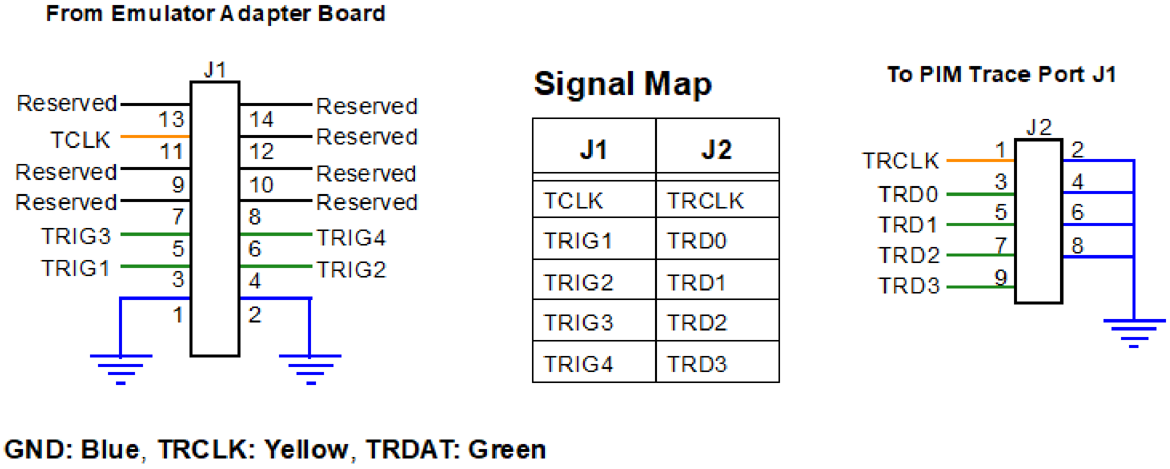


Figure 3-32. Adapter Board Pin Connection Diagram



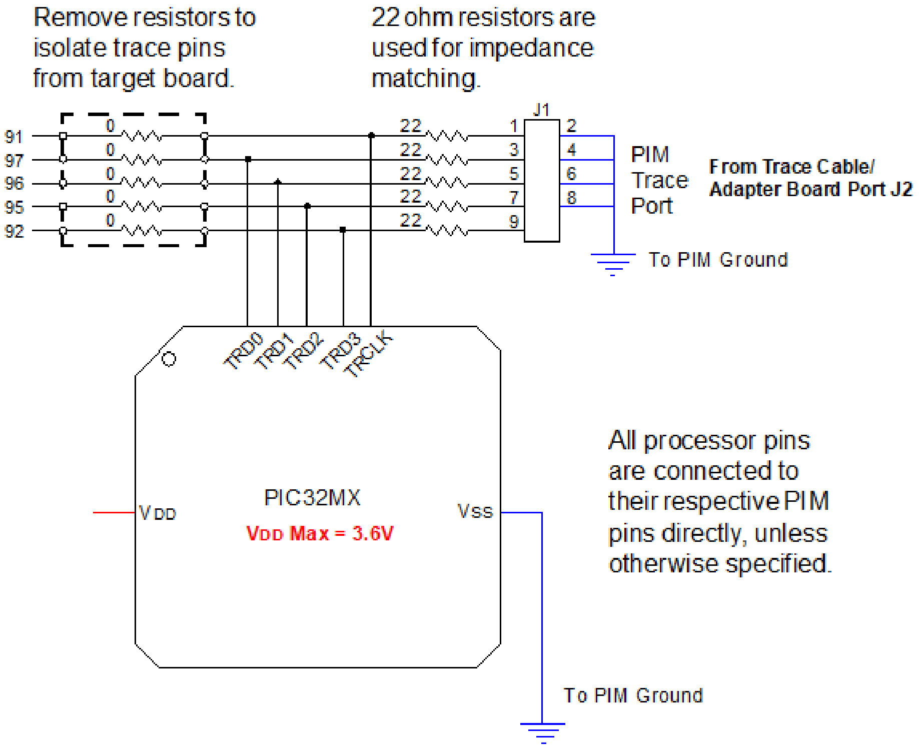
J1	J2
TCLK	TRCLK
TRIG1	TRD0
TRIG2	TRD1
TRIG3	TRD2
TRIG4	TRD3

3.3.8.4 PIC32 Instruction Trace - Target Trace Connection

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

- Termination series resistors will need to be added. For details, see the figure below.
- Depending on your board routing and loading of the signals used for trace, it is a good idea to place 0 ohm resistors that can be unpopulated to isolate the trace signals TRCLK and TRD3:0.

Figure 3-33. PIC32 PIM Pin Connection Diagram


Remove resistors to isolate trace pins from target board.
22 ohm resistors are used for impedance matching.
91
97
96
95
92
0
0
0
0
0
22
22
22
22
22
J1
1
3
5
7
9
2
4
6
8
PIM Trace Port
From Trace Cable/ Adapter Board Port J2
To PIM Ground
TRD0
TRD1
TRD2
TRD3
TRCLK
PIC32MX
VDD Max = 3.6V
VDD
Vss
All processor pins are connected to their respective PIM pins directly, unless otherwise specified.
To PIM Ground

3.3.8.5 PIC32 Instruction Trace - Target ICSP Connection

As an alternate way of tracing, you can use an existing ICSP connector on your target board. Connect these lines from the PIC32 Trace Adapter Board 14-pin connector to a Target ICSP SIL connector:

PIC32 Trace adapter board		Target Board ICSP SIL (PICkit) interface	
Pin Number	Signal Name	Pin Number	Signal Name
11	TVPP_IO	1	MCLR
14	TVDD_PWR	2	VDD_PWR
2/4/6/8/10	GND	3	GND
5	TPGD_IO	4	PGD
9	TPGC_IO	5	PGC

Related Links

3.3.8.1. PIC32 Instruction Trace - 14-pin EJTAG Connection

3.3.5.1. ICSP Target Connection

4. Operation

A simplified theory of operation of the MPLAB ICE 4 In-Circuit Emulator system is provided here. It is intended to provide enough information so that a target board can be designed that is compatible with the emulator 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. Emulator 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 AT Devices - On-Chip Debugging (OCD)

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

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

Run Mode
When in Run mode, the execution of code is completely independent of the MPLAB ICE 4. The MPLAB ICE 4 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.

4.2.1 SAM Devices with JTAG/SWD

All SAM devices feature the SWD interface for programming and debugging. Also, some SAM devices feature a JTAG interface with identical functionality. Check the device data sheet for supported interfaces of that device.

4.2.1.1 Arm® CoreSight™ Components

Microchip Arm® Cortex®-M based microcontrollers implement CoreSight™ compliant OCD components. The features of these components can vary from device to device. For further information, consult the device's data sheet as well as CoreSight documentation provided by Arm.

4.2.1.2 Arm® Cortex®-M Types and Debug Features

The following Arm® Cortex®-M types are currently used in SAM MCU devices. To determine the type used in a specific SAM MCU, see the data sheet for that device.

Cortex-M Types	Debug Support	Trace Support
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.	ITM, TPIU
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.	Data Trace, ITM, TPIU

4.2.2 tinyAVR®, megaAVR®, and XMEGA® Devices

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

- 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.
- New tinyAVR devices have a UPDI interface, which is used for programming and debugging.

Table 4-1. Programming and Debugging Interfaces Summary

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

4.2.2.1 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 4.2.2.4. debugWIRE OCD Features.

4.2.2.1.1 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 is 12 V tolerant and can be configured to be used as /RESET or GPIO. For further details on how to use the pin in these configurations, see 3.3.4.4. AVR UPDI.

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

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

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

- Pull-up resistors on the UPDI line must not be smaller than 10 kΩ. A pull-down resistor should not be used, or it should be removed when using UPDI. The UPDI physical is push-pull capable, so only a weak pull-up resistor is required to prevent false Start bit triggering when the line is idle.
- If the UPDI pin is to be used as a 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 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.

4.2.2.1.2 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.

4.2.2.2 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])

4.2.2.2.1 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 via 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 ICE 4 will assert the RESET line while connecting. If this line is wired correctly, it will force the target AVR device into Reset, thereby allowing a JTAG connection.

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

Tip:
Be sure to check the “use external reset” checkbox in both the programming dialog and debug options dialog in Atmel Studio to allow the MPLAB ICE 4 to assert the RESET line and re-enable the JTAG interface on devices which are running code which disables the JTAG interface by setting the JTAG disable bit.

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 ICE 4 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 ICE 4 may still poll the device as the target voltage drops below the AVR’s minimum operating voltage.

4.2.2.3 AVR XMEGA OCD Features

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

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

4.2.2.3.1 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 Atmel Studio will tell you if the breakpoint cannot be set, and why. Data breakpoints have priority over program breakpoints if software breakpoints are available.

External Reset and PDI Physical

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

JTAGEN Fuse

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

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

If the JTAGEN fuse is programmed, the JTAG interface can still be disabled in firmware by setting the JTAG disable bit in the MCU Control Register. This will render code un-debuggable, and should not be done when attempting a debug session. If such code is already executing on the Microchip AVR device when starting a debug session, the MPLAB ICE 4 will assert the RESET line while connecting. If this line is wired correctly, it will force the target AVR device into Reset, thereby allowing a JTAG connection.

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

Tip:
Be sure to check the “use external reset” checkbox in both the programming dialog and debug options dialog in Atmel Studio to allow the MPLAB ICE 4 to assert the RESET line and re-enable the JTAG interface on devices which are running code which disables the JTAG interface by setting the JTAG disable bit.

Debugging with Sleep for ATxmegaA1 rev H and Earlier

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

- Go into the MPLAB ICE 4. 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

4.2.2.4 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

4.2.2.4.1 debugWIRE Special Considerations

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

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

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

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

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

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

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

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

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

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

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

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

4.2.2.4.2 debugWIRE Software Breakpoints

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

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

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

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

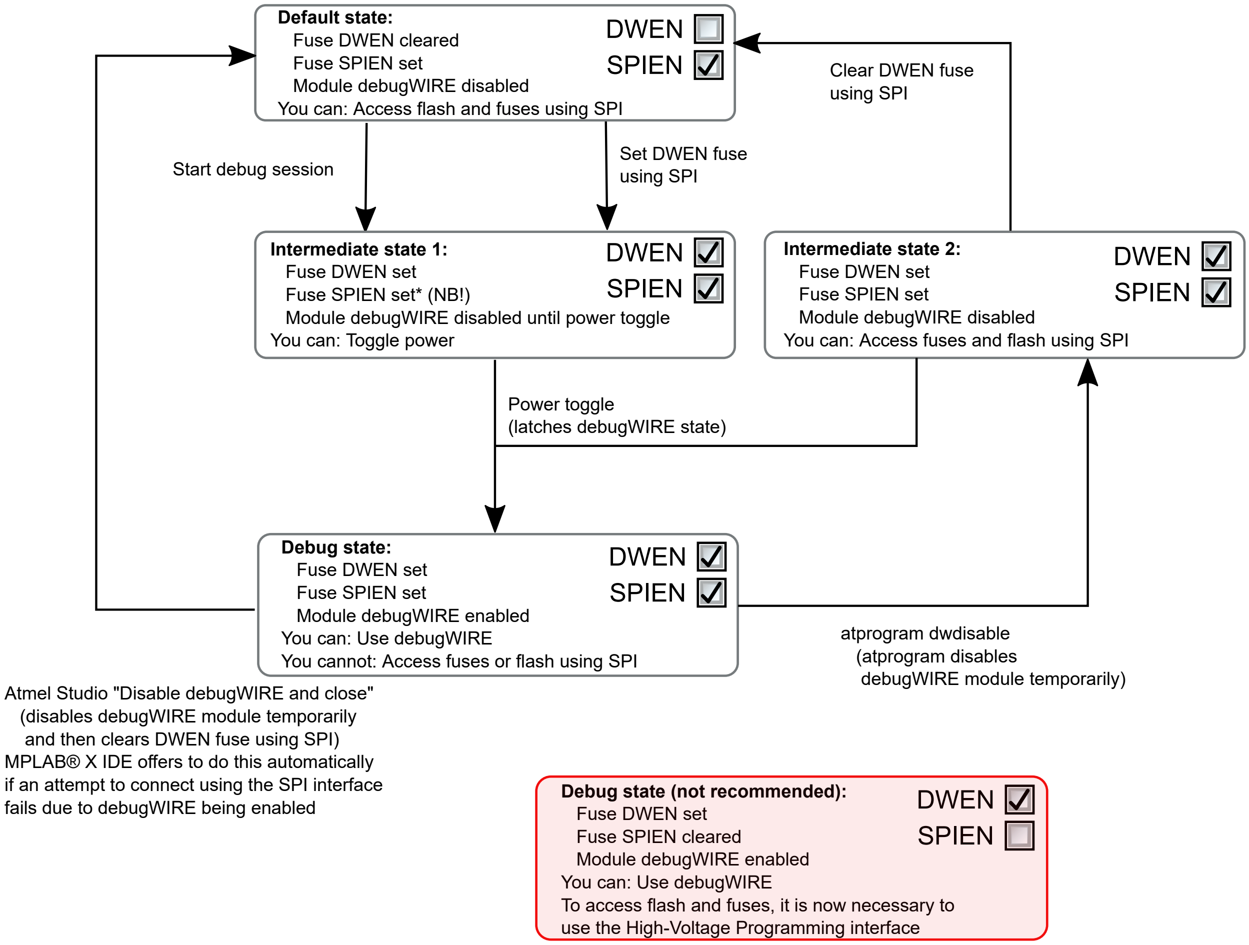
4.2.2.4.3 Understanding debugWIRE and the DWEN Fuse

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

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

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

Figure 4-1. Understanding debugWIRE and the DWEN Fuse


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

4.2.2.5 Advanced Debugging (AVR® JTAG/debugWIRE devices)

I/O Peripherals

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

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

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

Single Stepping I/O Access

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

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

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

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

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

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

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

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

4.3 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.

4.3.1 Emulator Basic Features

MPLAB ICE 4 in-circuit emulator has the following basic debug features.

4.3.1.1 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.

4.3.1.2 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.

4.3.1.3 Use Breakpoints

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

4.3.1.3.1 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 IDE documentation for more on this feature. Not all devices have software breakpoints.

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

4.3.1.3.2 Hardware or Software Breakpoint Selection

To select hardware or software breakpoints:

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

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

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

Table 4-2. Hardware vs. Software Breakpoints

Feature	Hardware Breakpoints	Software Breakpoints
Number of breakpoints	Limited	Unlimited
Breakpoints written to*	Internal debug registers	Flash Program Memory

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

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

4.3.1.4 Use the Stopwatch

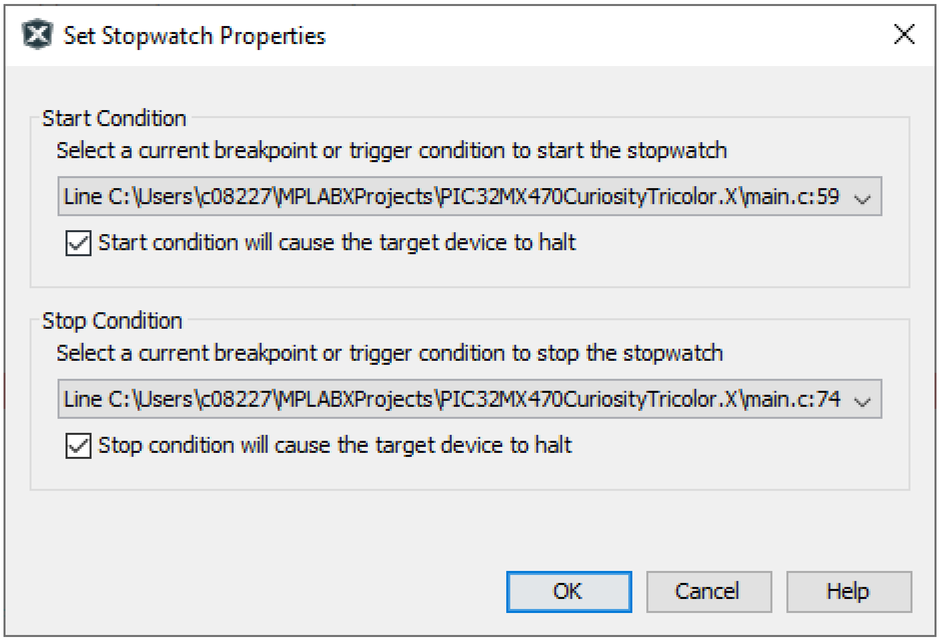
Use the stopwatch to determine the timing between two breakpoints.

Note: The stopwatch uses breakpoint resources.

To use the Stopwatch:

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

Figure 4-2. Stopwatch Setup



Set Stopwatch Properties

Start Condition

Select a current breakpoint or trigger condition to start the stopwatch

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

☑ Start condition will cause the target device to halt

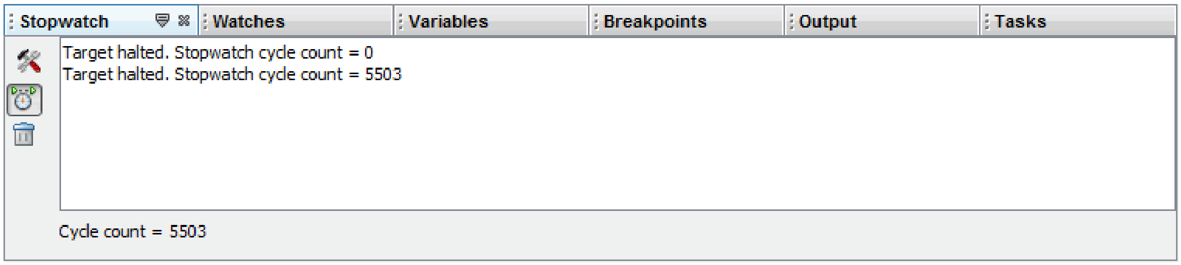
Stop Condition

Select a current breakpoint or trigger condition to stop the stopwatch

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

☑ Stop condition will cause the target device to halt

Figure 4-3. Stopwatch Window with Content



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

Cycle count = 5503

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

Table 4-3. 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.

4.3.1.5 Set Freeze Peripherals

You can select to “Freeze on Halt”, which allows you to freeze selected peripherals on a halt. For more on these functions, see 9.2.5. Freeze Peripherals.

4.3.2 ICSP Debugging

There are two steps to using MPLAB ICE 4 In-Circuit Emulator as a debugger. The first requires that an application is programmed into the target device (MPLAB ICE 4 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:

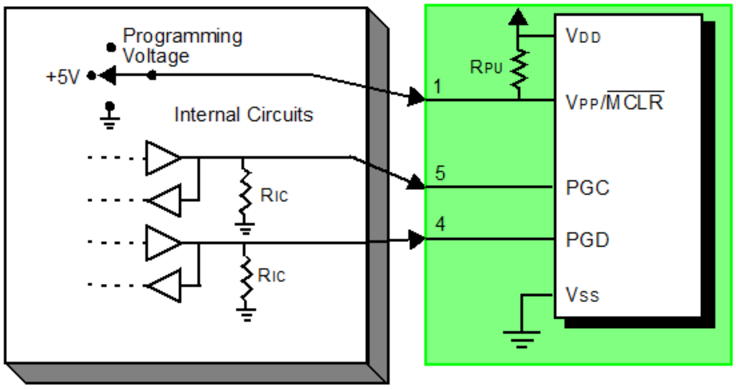
1. Programming the code into the target and activating special debug functions (see the next section for details).
2. Using the debugger to set breakpoints and run.

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

If the target device cannot be programmed correctly, the MPLAB ICE 4 will not be able to debug it.

A simplified diagram of some of the internal interface circuitry of the MPLAB ICE 4 is shown in the figure below. In the figure, Rpu=10 kΩ typical and Ric=4.7 kΩ.

Figure 4-4. Proper Connections for ICSP Programming

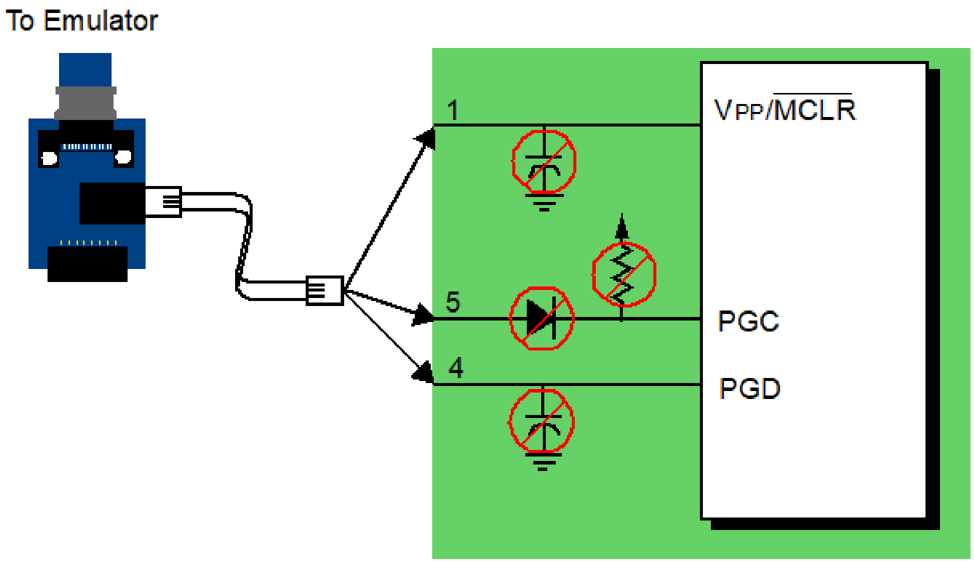


For programming, no clock is needed on the target device, but power must be supplied. When programming, the debugger puts programming levels on VPP/MCLR, sends clock pulses on PGC, and serial data via PGD. To verify that the part has been programmed correctly, clocks are sent to PGC and data is read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

4.3.2.1 Circuits That Will Prevent the Emulator From Functioning

The figure below shows the active debugger lines with some components that will prevent the MPLAB ICE 4 In-Circuit Emulator from functioning.

Figure 4-5. Improper Circuit Components



In particular, these guidelines must be followed:

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

4.3.2.2 Sequence of Operations Leading to Debugging

Given that the 4.3.2.4. Requirements for Debugging are met, set the MPLAB ICE 4 In-Circuit Emulator as the current tool in MPLAB X IDE. Go to *File> Project Properties* to open the dialog, and then under "Hardware Tool," click **ICE 4**.

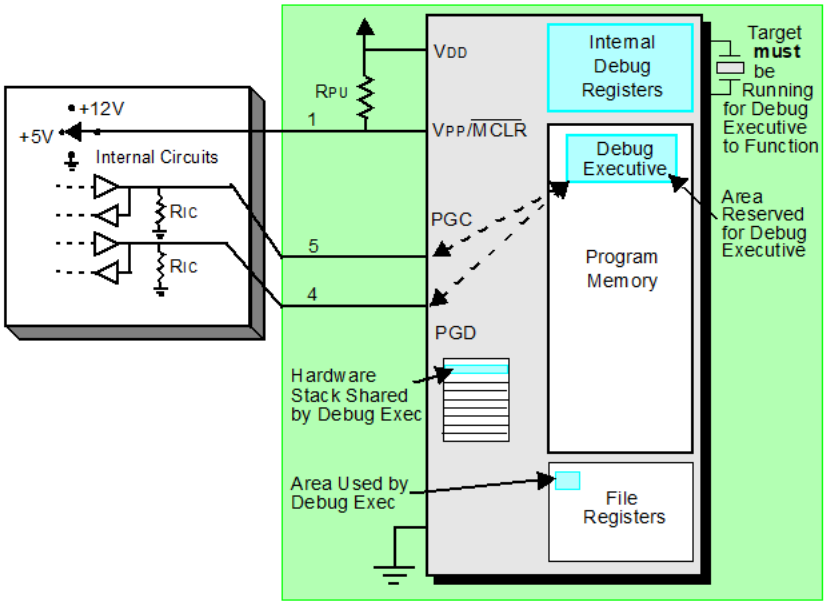
The following actions can now be performed:

- 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 the debug executive must reside in program memory, the application program must not use this reserved space. Some devices have special memory areas dedicated to the debug executive. Check your device data sheet for details.
- Special "in-circuit debug" registers in the target device are enabled by MPLAB X IDE. These allow the debug executive to be activated by the debugger. For more information on the device's reserved resources, see 4.3.2.5. Resources Used by the Debugger.
- The target device is run in Debug mode.

4.3.2.3 Debugging Details

The figure below illustrates the MPLAB ICE 4 In-Circuit Emulator system when it is ready to begin debugging. In the figure, Rpu=10 kΩ typical and Ric=4.7 kΩ.

Figure 4-6. MPLAB ICE 4 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. Commands on PGC and PGD communicate directly to these registers to set the breakpoint address.

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

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

The debug executive runs like an application in program memory. It uses some locations on the stack for its temporary variables. If the device does not run, for whatever reason (no oscillator, faulty power supply connection, shorts on the target board, etc.), then the debug executive cannot communicate to the MPLAB ICE 4, and MPLAB X IDE will issue an error message.

Another way to set a breakpoint is to select *Debug > Pause*. This toggles the PGC and PGD lines so that the in-circuit debug mechanism of the target device switches the Program Counter from the user’s code in program memory to the debug executive. Again, the target application program is effectively halted, and MPLAB X IDE uses the debugger communications with the debug executive to interrogate the state of the target device.

4.3.2.4 Requirements for Debugging

To debug (set breakpoints, see registers, etc.) with the MPLAB ICE 4 In-Circuit Emulator system, there are critical elements that must be working correctly:

- The emulator 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 ICE 4 In-Circuit Emulator 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.

4.3.2.5 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."

4.3.2.6 Programming

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

In the MPLAB X IDE, use the MPLAB ICE 4 as a programmer to program a non-ICE/-ICD device, that is, a device not on a header board. Set the MPLAB ICE 4 as the current tool (click the Debug Tool ICE 4 in the navigation window, then select *File > Project Properties* from the main menu to open the dialog, then under "Hardware Tool," click **ICE 4**) 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.

▷ ▾	Run Main Project Icon

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

4.3.3 PIC32M JTAG Debugging

PIC32M MCUs can be setup for debugging using either the 3.3.8. PIC32 Instruction Trace Adapter Board or 40-pin high speed cable, with or without trace.

4.3.3.1 PIC32M Types of JTAG

JTAG Method (PIC32M MCU):

- 2-wire JTAG: Microchip proprietary JTAG
- 4-wire JTAG: Traditional JTAG (IEEE1149.1)

What is 2-wire JTAG?

2-wire JTAG is a proprietary debug interface defined by Microchip. Based on 4-wire JTAG device internally, using a 2-wire converter that multiplexes the TMS, TDI, TDO JTAG lines to a single data line to the outside and demultiplexes it to the inside. Requires a special conversion unit in the device. On older devices, this conversion unit was only activated in case the device was in the special 2-wire programming mode, but did not allow full debugging using the 2-wire protocol.

4.3.3.2 PIC32M Debug Features

Basic Debug

Basic debug functions are available:

- Run, Halt, Reset
- Step Into, Step Over

Hardware Breakpoints

Hardware (HW) breakpoints are available and depend on the device used.

- PIC32: 6

Software/Flash Breakpoints

Software (SW) or “Flash” breakpoints are available as described below:

- Linux and Mac operating systems: Flash Breakpoints are not supported with the J-Link Base model.
- Windows operating systems: After all HW breakpoints are used, a "Flash Breakpoints Notice" dialog will open. Proceed by clicking one of the buttons described in the table below.

Table 4-4. Flash Breakpoint Notice Dialog

Button	Operation
Yes	Use unlimited Flash breakpoints on a trial basis
No	Go to a website to get a license that supports unlimited Flash breakpoints
Install existing license	Install an existing license that supported unlimited Flash breakpoints

5. Emulator Features

In addition to basic debug features, the emulator has advanced debug features or the implementations of the 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."

The following table specifies which emulator features are supported on each connection type.

Table 5-1. Supported Emulator 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	X	X	X	X	X
Data Capture, Runtime Watches	X	X	X		
PIC Instrumented Trace	X	X			
PIC32M Instruction Trace			X		
SAM ITM/SWO Trace					X
PC Sampling		X			
PC Profiling			X		
Debugger Polling				X	X
Power Monitoring	X		X	X	X
* Not all devices in a family have support. See features section for details.					

5.1 USB CDC Virtual COM Port

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

Use this virtual port to access the emulator 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 SPI and 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 steam 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.
SPI*	0x20	Communicates directly over SPI in Client mode.
UART USART*	0x21	Communicates directly over UART in Client mode. Communicates directly over USART in Client mode.
I^{2}C*	0x22	Communicates directly over I^{2}C in Client mode.
GPIO	0x30	Monitors and controls the state of GPIO pins.
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 5.10. Debugger Polling.
Reserved	0xFF	Special identifier used to indicate no interface.
* Coming Next Release		

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 SPI Interface

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

Important: The SPI hardware module uses an active-low Chip Select (CS) signal. Any data sent when the CS pin is high will be ignored.

The SPI interface is under the **DGI** section of the Data Sources (left) pane. When an SPI connection is selected, the SPI settings are displayed in the lower section of this pane.

Table 5-3. SPI Settings

Field Name	Values	Usage
Char Length	5, 6, 7, or 8 bits	Number of bits in each transfer
Mode	• Clock idle normally low, Sample data on rising edge • Clock idle normally low, Sample data on falling edge • Clock idle normally high, Sample data on falling edge • Clock idle normally high, Sample data on rising edge	SPI mode, controlling clock phase and sampling.
Force CS Sync	Check to enable.	The SPI interface is only enabled after the Chip Select line has toggled twice.
Kit-side Timestamping	Check to enable.	Target timestamping

5.2.1.3 USART Interface

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

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

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

Table 5-4. 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

5.2.1.4 I2C Interface

The I2C **source** streams the raw values received on the I2C interface.

The **I2C Configuration** options are displayed under the **I2C** interface in the **DGI** section of the left pane.

The I2C interface is under the **DGI** section of the Data Sources (left) pane. When an I2C connection is selected, the I2C settings are displayed in the lower section of this pane.

Table 5-5. I2C Settings

Field Name	Values	Usage
Speed	0	The expected operation speed of the interface in Hertz helps the client device adjust the timings. Up to 400 kHz is supported.

...........continued		
Field Name	**Values**	**Usage**
Address	1	Address of the client device, 7-bit address : 8 to 120.
Kit-side Timestamping	Check to enable.	Target timestamping

5.2.1.5 GPIO Interface

The GPIO interface contains the bit values of the enabled Debug GPIO pins. A packet of unsigned 8-bit data is transmitted every time a pin toggles.

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

Table 5-6. Configuration

Field Name	Values	Usage
GPIO 0 Change Triggers Bus Read	ON, OFF	Monitor change on GPIO pin 0 to trigger a bus read
GPIO 1 Change Triggers Bus Read	ON, OFF	Monitor change on GPIO pin 1 to trigger a bus read
GPIO 2 Change Triggers Bus Read	ON, OFF	Monitor change on GPIO pin 2 to trigger a bus read
GPIO 3 Change Triggers Bus Read	ON, OFF	Monitor change on GPIO pin 3 to trigger a bus read

Important: When plotting the Debug GPIO data source, all GPIOs are sampled but only those GPIOs that have change triggers enabled will trigger a sample on change. For example, if GPIO n (n = 0,1,2) all have “GPIO n Change Triggers Bus Read” disabled, but GPIO 3 has this function enabled, then GPIO values will only be sampled when GPIO 3 changes; that is, all four GPIO values will be read only when GPIO 3 changes.

5.2.1.6 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 5-7. Power Settings Controls

Control	Value	Usage
Enabled Channels	A, A+B	Either enable channel A only or both channels A and B. Channel A is always enabled.
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 ICE 4 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 Data Capture and Runtime Watches - PIC MCUs and dsPIC DSCs

Data capture is an on-board debug feature of the device. When a value is written to an SFR whose address matches that in the Capture Data Address register, a trigger and data capture occurs. You cannot access data capture directly; you must select an application that uses data capture, such as the MPLAB Data Visualizer.

Runtime Watches are selected symbols in a Watches window that change as the program runs. You can select a symbol to be runtime in a Watches window. See MPLAB X IDE documentation for details.

8- and 16-Bit Devices

Not all 8- and 16-bit devices support data capture and/or runtime watches. A list of supported features by device is available in “Hardware Tool Debug Features by Device” found under *Help>Release Notes*.

When watching symbols, an 8-bit PIC device can only watch 8-bit variables and a 16-bit PIC device can only watch 16-bit variable.

For 8- and 16-bit devices, data captures, runtime watches and hardware breakpoints use the same registers/ resources. For example, if you use a data capture resource for a symbol, you will not be able to use a hardware breakpoint or runtime watch resource for that symbol.

32-Bit Devices

Not all 32-bit devices support data capture and/or runtime watches. A list of supported features by device is available in “Hardware Tool Debug Features by Device” found under *Help>Release Notes*.

For PIC32 devices, hardware breakpoints do not use data capture or runtime watch resources. However data captures and runtime watches do use the same resources. Therefore, if you use a data capture resource for a symbol, you will not be able to use a runtime watch resource for that symbol, and vice versa.

5.3.1 Data Capture and Streaming Data

Data capture provides streaming data from a device.

To set up data capture:

1. Build the project (In the Projects window, right click on the project name and select “Build”). The project must be built to see the available symbols: *Project Properties>Loading*, check “Load symbols when programming or building for production (slows process)”.
2. Select *Window>Debugging>Watches* to open the Watches window.
3. Right click in the window and select “New Watch”. Select the symbol or SFR wish to watch in the New Watch window. Click **OK**.
4. Begin a debug session . View data in Watches window.

5.3.2 Runtime Watches and Streaming Data (PIC32M MCUs)

For PIC32MZ/MK devices without data capture but with instruction trace, the trace mechanism can be used for runtime watches. If you have a device PIM with a trace connector, you can use the 3.3.8. PIC32 Instruction Trace Adapter Board to connect a trace cable to enable runtime watches. If you just have a device, you can use the high-speed 40-pin ribbon cable to access trace pins from a connector on your board (see 3.3.1. Target Connection Pinout.)

To set up runtime watches:

1. Build the project (In the Projects window, right click on the project name and select “Build”). The project must be built to see the available symbols: *Project Properties>Loading*, check “Load symbols when programming or building for production (slows process)”.
2. Select *Window>Debugging>Watches* to open the Watches window.
3. Right click in the window and select “New Runtime Watch”. Select the symbol or SFR you wish to watch in the New Run Time Watch window. Click **OK**.
4. Begin a debug session . View data in Watches window.

5.3.3 Runtime Watches and the Watches Window

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

- Watches – Window > Debugging menu
- Memory – Window > PIC Memory Views menu

To set up runtime watches:

1. Build the project (In the Projects window, right click on the project name and select “Build”). The project must be built to see the available symbols: *Project Properties>Loading*, check “Load symbols when programming or building for production (slows process)”.
2. Select *Window>Debugging>Watches* to open the Watches window.
3. Right click in the window and select “New Runtime Watch”. Select the symbol or SFR you wish to watch in the New Run Time Watch window. Click **OK**.
4. Begin a debug session . Watch variable values change in the Watches window.
5. To see the watched variable change in a Memory window, Pause (halt) and open a Memory window containing the watched variable. Continue the program and watch the values change in this window.

5.4 CI/CD Support

MPLAB® ICE 4 In-Circuit Emulator 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-5000xxxx) 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 emulator may be included using one of the wizard screens (see table below.)

The ICE 4 Breakout Board (AC244141) will also be required to connect the MPLAB ICE 4 to the target.

Table 5-8. CI/CD Wizard - Hardware Testing

Option	Description
Enable Hardware Testing	Enable the emulator as a test tool.
Configuration to Build and Run on ICE4	Select a project configuration that uses the emulator; either default or one dedicated to hardware use.

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

Option	Description
IP Address of ICE4	Enter the IP address of the emulator you wish to use. If you do not know the IP address of the emulator, you can use `ipconfig` or a similar tool to search the system to which the emulator 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.

5.5 Instrumented Trace for PIC MCUs and dsPIC DSCs

This section discusses the available types of instrumented trace and how to use them with PIC microcontrollers (MCUs) and dsPIC digital signal controllers (DSCs).

5.5.1 Requirements for Trace

The following is required to use trace for PIC 8- and 16-bit devices:

- MPLAB X IDE.
- MPLAB XC8 C compiler or MPLAB XC16 C compiler.
- A target board with debug and trace connections to a device that supports trace.
- 3.3.5. PIC MCUs - ICSP Adapter Board OR a target board with a connector to the high-speed 40-pin ribbon cable.

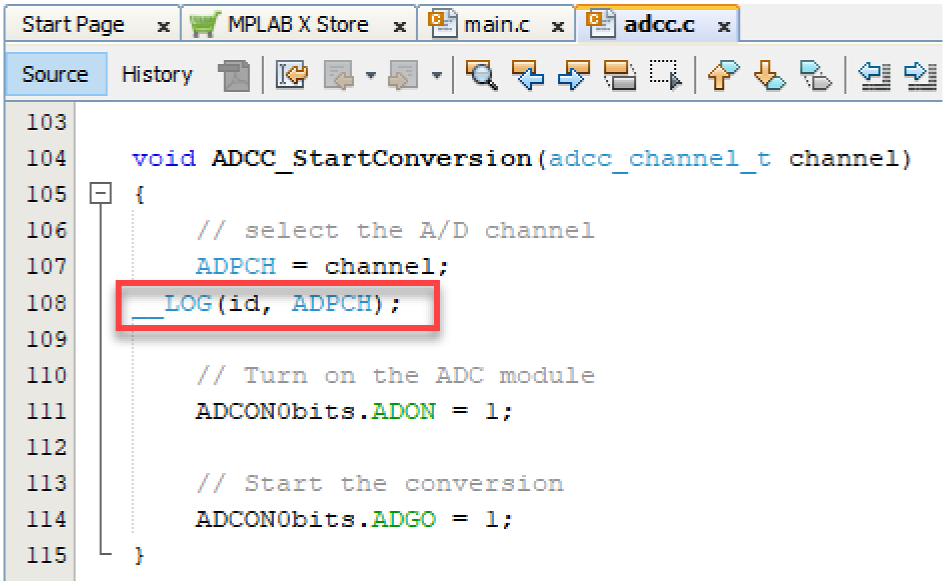
The following are limitations of trace:

- For 8- and 16-bit devices, in-line assembly code (assembly code within C code) cannot be traced.
- For additional limitations by device, see *Help>Help Contents>Hardware Tool Reference>Limitations*.
- For additional limitations by trace type, see 5.5.3. Trace Methods.

5.5.2 How Trace Works

Trace for the MPLAB ICE 4 in-circuit emulator (Instrumented Trace) is a solution for providing basic trace information. Through the use of `TRACE()` and `LOG()` macros, you can report program locations or variable values to MPLAB X IDE while the application is running and examine them via the Trace window once the application halts. You may type these macro names in manually or right click in the editor and select the macro to be inserted from the context menu. To log a variable value, the variable should be highlighted before selecting from the context menu.

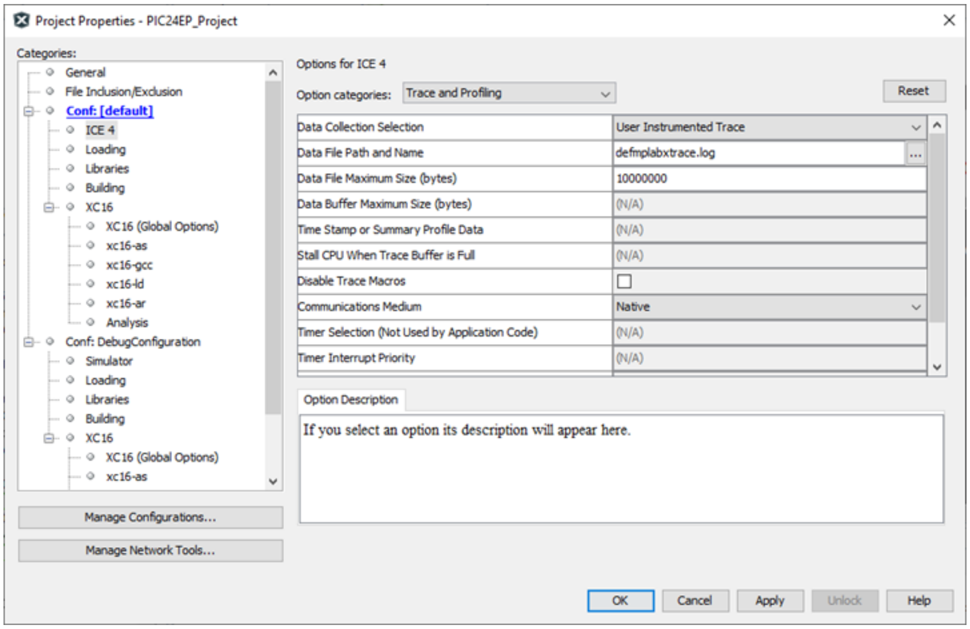
Figure 5-1. Example of Inserted Log Macro



```
void ADCC_StartConversion(adcc_channel_t channel)
{
    // select the A/D channel
    ADPCH = channel;
  __LOG(id, ADPCH);

    // Turn on the ADC module
    ADCON0bits.ADON = 1;

    // Start the conversion
    ADCON0bits.ADGO = 1;
}
```

Trace methods and related options are located on the Project Properties dialog, “ICE4” Category, “Trace and Profiling” Options Category page. The choices found under “Communications Medium” are “Native Trace” (utilizes PGC/PGD communication lines), “SPI Trace” or “I/O Port Trace.” Not every method is available on every part; i.e., the options are device specific. The Instrumented Trace library supports C and assembly projects on PIC18F MCU devices, but only C projects on 16-bit devices.

Figure 5-2. Project Properties – Trace Selection



Options for ICE 4

Option categories: Trace and Profiling

Data Collection Selection	User Instrumented Trace
Data File Path and Name	defmplabxtrace.log
Data File Maximum Size (bytes)	10000000
Data Buffer Maximum Size (bytes)	(N/A)
Time Stamp or Summary Profile Data	(N/A)
Stall CPU When Trace Buffer is Full	(N/A)
Disable Trace Macros	□
Communications Medium	Native
Timer Selection (Not Used by Application Code)	(N/A)
Timer Interrupt Priority	(N/A)

Option Description

If you select an option its description will appear here.

The trace and log information transmitted is identical regardless of the trace method used. For `TRACE()`, a single value in the range of 64-127 is sent. A label generated using this number is automatically inserted into the code, so MPLAB X IDE can identify in the trace buffer the location that sent the value. For `LOG()`, a two-byte header is sent, followed by the value of the variable being logged. The first byte indicates the variable type, and is a value between 0 and 63. The second byte indicates the location that sent the variable. Here, the location is represented by a value between 0 and 127.

Interrupts are disabled during every `TRACE()` or `LOG()` call. This is to ensure that trace or log statements at an interrupt level do not interfere with a trace or log statement that may already be in progress at the application level.

A similar argument holds for protecting statements within a low priority interrupt from being corrupted by those from a high priority interrupt.

5.5.3 Trace Methods

Currently there are three instrumented trace methods. All are language-tool-version dependent, and stream data in real time to MPLAB X IDE. The pluses and minuses of using each trace method, as well as communication information, are summarized below.

Trace Method	Speed	Code Size Impact	Real Time Op	Pin Usage	Device Feature Needed	High Speed Communication
Native Trace	Fast	Large	Close	None	Built-in debug	Greater than 15 MIPS[1]
SPI Trace	Faster	Medium	Closer	SPI pins	SPI	Yes[1]
I/O Port Trace	Fastest	Small	Closest	9 pins of Debug Connector	None	Yes[2]

1. The actual cutoff speed may vary depending on layout, noise, and similar considerations.
2. Requires connection from device port to emulator debug connector (8-pin port + 1 pin clock).

5.5.3.1 Native Trace

Native trace can be used with the information conveyed via the PGD/PGC or EMUC/EMUD pins. This two-wire interface uses the trace macro format (see 5.5.5. Setting Up Trace in MPLAB X IDE). Either a 6- or 8-pin ICSP™ connection can be used for Native trace.

For 8-bit and 16-bit devices, Native trace requires that you enter the clock speed (Project Properties dialog, ICE4 category, Clock options category).

If Native trace is used, then data captures cannot be used as these features use the same device resource. Breakpoints are still available, but be aware that Native trace will use one breakpoint for tracing. This will NOT be reflected on the Device Debug Resource toolbar.

To use data capture triggers, you must disable Native trace (see 5.5.8. Disabling Trace).

Related Links

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

5.5.3.2 SPI Trace

SPI trace can be used only with an 8-pin ICSP™ connection. Trace clock and data are provided through pins 7 (DAT) and 8 (CLK). The device does not have to be operating at high speeds to use this feature.

When you dedicate these pins to tracing, any multiplexed function on these pins cannot be used by the application.

For devices with remappable peripheral pins, be aware that the SPI trace macro does not touch any PPS register and does not need to know how the peripheral is mapped to a certain pin – it will write to the SPI1 or SPI2 selected in MPLAB X IDE.

SPI trace does require that you enter the clock speed in the *Project Properties>ICE4>Clock* options category.

The SPI interface uses the trace macro format (see 5.5.5. Setting Up Trace in MPLAB X IDE).

5.5.3.3 I/O Port Trace

I/O Port trace can be used with either standard or high-speed communications. Trace clock and data are provided from a device 8-pin I/O port through the emulator's logic probe connector.

The I/O port must have all 8 pins available for trace. The port must not be multiplexed with the currently-used PGC and PGD pins. Therefore, review the data sheet of the selected device to determine the uninitialized/default port pin states and change them as necessary.

For hardware connections, see 3.3.7.3. IO Port Trace Connections.

The port interface uses the trace macro format (see 5.5.5. Setting Up Trace in MPLAB X IDE).

5.5.4 Hardware Setup

Preliminaries

1. Use USB communication between the PC and MPLAB ICE 4. Other communication types do not support trace.
2. Find devices that support Native, SPI or I/O Port trace – see *Help>Release Notes> Debug Features Support>Hardware Tool Debug Features by Device*.
3. Design the target board to have a connector for emulator-target communication if using the 3.3.5. PIC MCUs - ICSP Adapter Board. Alternately design the target board to connect to the high-speed 40-pin ribbon cable with pins for both debug and trace (see 3.3.1. Target Connection Pinout).
   Note: If using I/O Port trace, you will need to use the high-speed 40-pin ribbon cable for communications and access to port pins.
4. When using SPI trace, pins 7 (DAT) and 8 (CLK) are used. Therefore, you cannot use the other functions multiplexed on these pins. When using I/O Port trace, the port used for trace must not be multiplexed with the currently-used PGC and PGD pins.

Set Up Hardware

To use the PIC Instrumentation Trace feature with your own board do the following:

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

5.5.5 Setting Up Trace in MPLAB X IDE

To set up MPLAB X IDE to use trace for the MPLAB ICE 4 in-circuit emulator:

1. Right click on the project name and select “Properties”. In the Project Properties dialog click on “ICE4” under “Categories.”
2. Under “Option categories,” select “Clock”. For data capture and trace, the emulator needs to know the instruction cycle speed.
3. Under “Option categories,” select “Trace and Profiling.”
4. Under “Data Collection Selection,” choose “User Instrumented Trace.”
5. Under “Communications Medium,” choose either “Native,” “I/O PORT” or “SPI” trace.
6. Set up any other trace-related options (see 9.2.6. Trace and Profiling).
7. Click **OK**.

5.5.6 Running Trace

1. Debug Run (*Debug>Debug Project)* your application.
2. Pause the application.
3. View the trace data in the Trace window (*Window>Debugging>Trace*). For each `__TRACE` macro, the line of code following the macro will appear in the trace window each time it is passed. For each `__LOG` macro, the selected variable in the line of code following the macro will appear in the trace window each time it is passed.
   Note: To trace multiple lines of code or variables, you must place a macro before each line/variable that you wish to trace.

Repeat these steps each time you change a trace point.

5.5.7 Tracing Tips

When using `__TRACE` and `__LOG` macros in your code, consider the following:

- Focus on one area of an application and place `__TRACE` and `__LOG` macros so that they form a “flow” in the Trace window. That way, you can follow the execution flow and debug the application based on missing/incorrect trace points or an abrupt end to the trace flow.

- Use __TRACE and __LOG macros with conditional statements in your code to aid in debugging. Example: When a variable reaches a certain value, start logging it.

```
If(var > 5)
{
__LOG(ID, var)
}
```

- Leave __TRACE and __LOG macros in your code for future debugging, if this is allowable, i.e., go to Project Properties>ICE4>Trace and select “Disable Trace Macros.”

5.5.8 Disabling Trace

To temporarily turn off trace data collection:

1. Select *File>Project Properties>ICE4>Trace and Profiling*.
2. Check “Disable Trace Macros.”
3. Click **OK**.

To disable the full trace capability:

1. Remove all trace and log macros from code.
2. Select *File>Project Properties>ICE4>Trace and Profiling*.
3. Under “Data Collection Selection,” choose “Off.”
4. Click **OK**.

5.5.9 Resource Usage Examples

The following examples are for illustration only. Your results may vary based upon compiler/assembler version, command line options, MPLAB X IDE version, size of data variable being logged, interrupt state, and device in use. All examples include argument setup, function call, and return time in their cycle counts.

The PIC18FXxxJ MCU examples are compiled/assembled for non-priority interrupt usage (30 instructions). For priority interrupt usage, the value is 57; and for no interrupt usage, the value is 15.

The dsPIC33F DSC examples show nine instructions specified in the 16-bit library size for memcpy().

Table 5-9. PIC18FxxJ MCU Running at 4MHz (1 MIPS) with Assembly Project

	Native	SPI	I/O Port
Library Size (in instructions)	23 + 30	37 + 30	25 + 30
GPRs Used (in bytes)	8	6	6
__TRACE(id) instruction cycles	80	54	42
__LOG(id, BYTE) instruction cycles	168	90	57

Table 5-10. PIC18FxxJ MCU Running at 40MHz (10 MIPS) with C Project

	Native	SPI	I/O Port
Library Size (in instructions)	75 + 30	87 + 30	112 + 30
GPRs Used (in bytes)	10	8	8
__TRACE(id) instruction cycles	79	71	55
__LOG(id, INT) instruction cycles	225	169	162

Table 5-11. dsPIC33F DSC Running at 10 MIPS with C Project

	Native	SPI	I/O Port
Library Size (in instructions)	87 + 9	92 + 9	93 + 9

...........continued			
	Native	SPI	I/O Port
GPRs Used (in bytes)	18	14	0
`__TRACE(id)` instruction cycles	80	53	32
`__LOG(id, INT)` instruction cycles	212	124	106

dsPIC33F device running at 16 MIPS with C project

Table 5-12. dsPIC33F DSC Running at 16 MIPS with C Project

	Native	SPI	I/O Port
`__TRACE(id)` instruction cycles	88	53	32
`__LOG(id, INT)` instruction cycles	227	138	106

Table 5-13. dsPIC33F DSC Running at 34 MIPS with C Project

	Native	SPI	I/O Port
`__TRACE(id)` instruction cycles	100	53	32
`__LOG(id, INT)` instruction cycles	251	152	106

5.5.10 More on Trace/Log ID Numbers

MPLAB X IDE will automatically generate the ID numbers required for a trace or log macro. However, to understand the method behind the numbering, read further.

You can have 64 trace points and 64 log points. These limits are determined by port trace (8 bits). Bit 7 is used as a clock and Bit 6 is used as a flag which indicates either a trace record (1) or a log record (0).

For a trace record, the low order bits represent the trace number (nnnnnn). You could say 0-63 are the legal trace numbers and require the trace flag be set, but it was just easier to combine the flag with the number and say the valid numbers are 64-127.

clock	1	n	n	n	n	n	n
bit 7	bit 6						bit 0

For a log record, the low order bits represent the log number (nnnnnnn).

clock	0	n	n	n	n	n	n
bit 7	bit 6						bit 0

5.5.11 Trace Quick Reference

If you are new to using the MPLAB ICE 4 in-circuit emulator trace feature, it is recommended that you read through the entire trace section for a full understanding. Use this section as a quick reference for trace.

1. Select *File>Project Properties>ICE4>Trace*. Enable trace, choose the trace method and set other trace options.
2. Select *Window>Debugging>Trace* to open the trace window in which to view trace data.
3. Right click in your code to enter trace macros (`__TRACE`, `__LOG`) as desired.
4. Debug Run your project.

5.6 Instruction Trace for PIC32M MCUs

This section will discuss trace for PIC32M devices and how to use it. Not all PIC32M devices have instruction trace. See *Help>Release Notes> Debug Features Support>Hardware Tool Debug Features by Device*.

5.6.1 Requirements for Instruction Trace

The following is required to use trace for PIC32M 32-bit devices:

- MPLAB X IDE and MPLAB XC32/32++ C compiler.
- PIC32 Plug-In Module (PIM) containing a device that supports trace and a trace port OR a target board with debug and trace connections to a device that supports trace.
- MPLAB ICE 4 PIC32 Trace Adapter Board and the 12-inch trace cable OR a target board with a connector to the high-speed 40-pin ribbon cable.

5.6.2 How Instruction Trace Works

PIC32 instruction trace uses a MIPS32 iFlowtrace™ mechanism, which is a non-intrusive hardware instruction trace. You can use this trace to capture every instruction executed by the device. The trace data is sent from the device using the pins TRCLK and TRD3:0 to the emulator. The emulator streams this data to a trace buffer on the PC that acts like a rolling FIFO.

The amount of trace data is limited only by the size of the trace buffer. This buffer can fill quickly even when set to the maximum size, so it is wise to determine exactly what you need to capture.

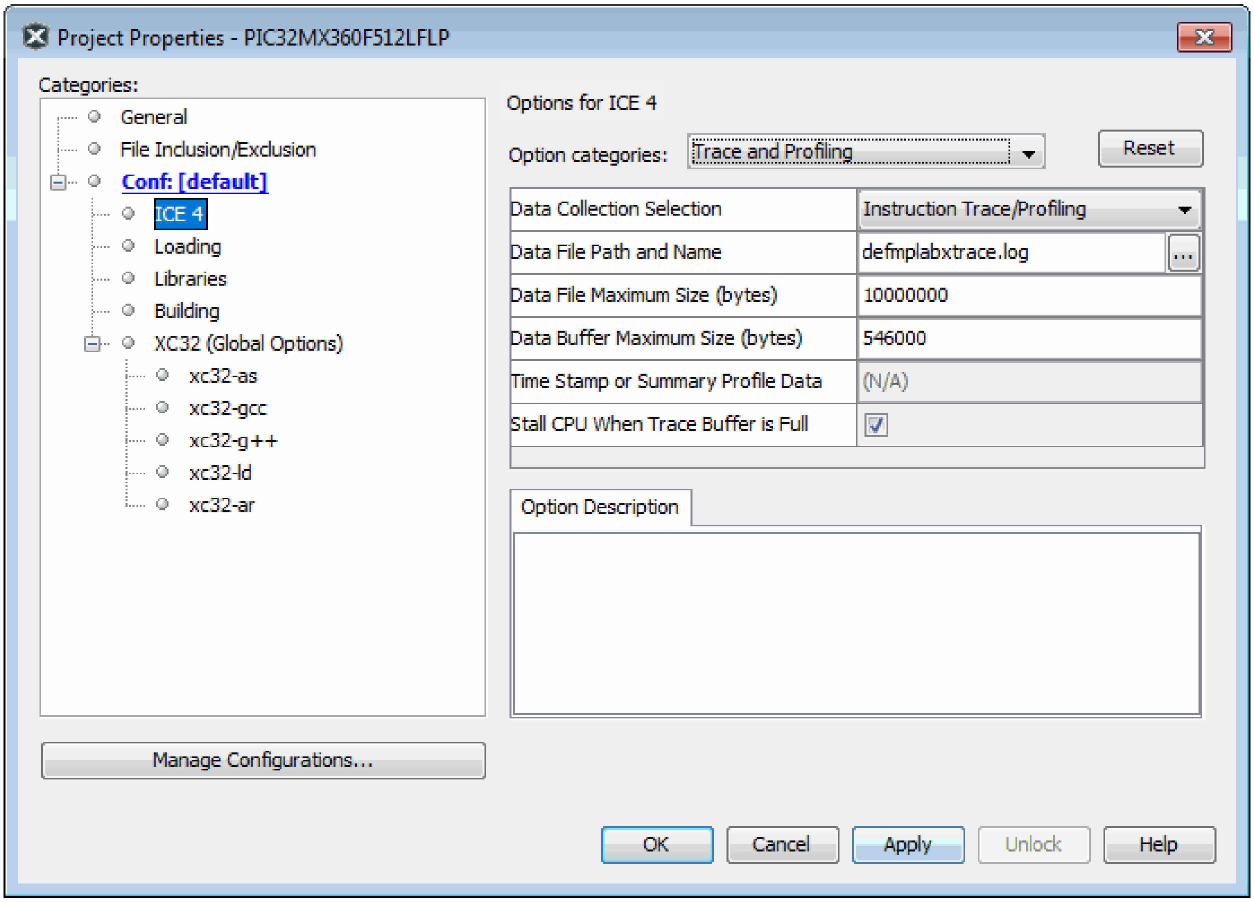
Enable and set trace options in *Project Properties>ICE4>Trace and Profiling*. Here you may set:

- Data selection – enable/disable trace and select the type of trace.
- Data file path and name – the location of trace file.
- Data file maximum size – the size of trace file.
- Data Buffer maximum size – the size of the trace buffer.

The maximum off-chip (PC) trace buffer size is 22MB. Execution will not halt when this external buffer is full.

See also 9.2.6. Trace and Profiling.

Figure 5-3. Instruction Trace Options



Project Properties - PIC32MX360F512LFLP

Categories:

- General
- File Inclusion/Exclusion
- Conf: [default]
  - ICE 4
  - Loading
  - Libraries
  - Building
  - XC32 (Global Options)
    - xc32-as
    - xc32-gcc
    - xc32-g++
    - xc32-ld
    - xc32-ar

Options for ICE 4

Option categories: Trace and Profiling

Data Collection Selection	Instruction Trace/Profiling
Data File Path and Name	defmplabxtrace.log
Data File Maximum Size (bytes)	10000000
Data Buffer Maximum Size (bytes)	546000
Time Stamp or Summary Profile Data	(N/A)
Stall CPU When Trace Buffer is Full	☑

5.6.3 Hardware Setup

Preliminaries

1. Use USB communication between the PC and MPLAB ICE 4. Other communication types do not support trace.
2. Find devices that support PIC32M instruction trace – see *Help>Release Notes> Debug Features Support>Hardware Tool Debug Features by Device*.
3. Design the target board to have a connector for emulator-target communication and a connector for trace pins if using the 3.3.8. PIC32 Instruction Trace Adapter Board. Alternately design the target board to connect to the high-speed cable with pins for both debug and trace (see 3.3.1. Target Connection Pinout).
4. When using trace, pins TRCLK and TRD3:0 are used. Therefore, you cannot use the other functions multiplexed on these pins.
   For PIC32MX360F512L, multiplexed functions are RG14:12 and RA7:6. Check your device data sheet for details.

Set Up Hardware – PIC32M MCU on PIM

To use the PIC32M Instruction Trace feature with the PIM do the following:

1. Plug the PIM into the unpowered target board.
2. Install communication cable between the emulator or adapter board and the communication connector on the target board.
3. If using the adapter board, connect the trace cable between the adapter board and trace connector on the PIM.
4. Power the target.

Set Up Hardware – PIC32M MCU on Target

To use the PIC32M Instruction Trace feature with your own board do the following:

1. The target board should be unpowered.
2. Install communication cable between the emulator 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.

5.6.4 Set Up Instruction Trace in MPLAB X IDE

To set up MPLAB X IDE to use trace for the MPLAB ICE 4 in-circuit emulator:

1. Right click on the project name and select “Properties.” In the Project Properties dialog, click on “ICE4” (under “Categories”).
2. Under “Option categories,” select “Trace and Profiling.”
3. Under “Data Collection Selection,” choose “Instruction Trace/Profiling.”
4. Set up any other trace-related options.
5. Click **OK**.

On a Debug Run, trace will continue to fill the trace buffer with data, rolling over when the buffer is full, until a program Halt.

5.6.5 Viewing Instruction Trace Data

When trace is enabled and code is run, trace data will be collected by the emulator. Once the device is halted, trace data will be decoded and displayed in the Trace window (*Window>Debugging>Trace*).

5.7 SAM ITM/SWO Trace

ITM (Instrumentation Trace Macrocell) trace outputs UART-format data using the Serial Wire Output (SWO). Not all SAM devices have ITM trace. See *Help>Release Notes> Debug Features Support>Hardware Tool Debug Features by Device*.

5.7.1 Requirements for ITM Trace

The following is required to use trace for SAM devices:

- MPLAB X IDE and MPLAB XC32/32++ C compiler.
- A target board with debug and trace connections to a device that supports trace.
- 3.3.6. SAM MCUs - Cortex-M Trace Adapter Board OR a target board with a connector to the high-speed 40-pin ribbon cable.

5.7.2 How ITM/SWO Trace Works

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

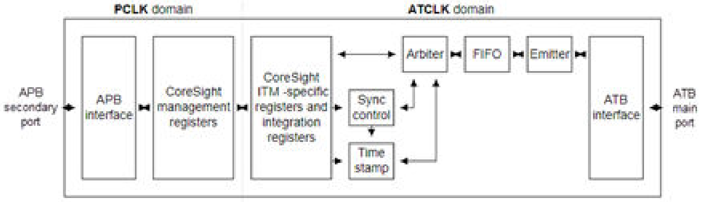
- support `printf` style debugging.
- trace OS and application events.
- emit diagnostic system information.

5.7.2.1 SAM ITM Trace

MPLAB ICE 4 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 ICE 4 and is sent over the trace interface to the host computer. The maximum reliable data rate is about 4 MB/s at this time.

Note: You can set the ITM baud rate in MPLAB X IDE. See 5.7.4. Set Up ITM in MPLAB X IDE.

Figure 5-4. 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 *First In First Out* (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 .

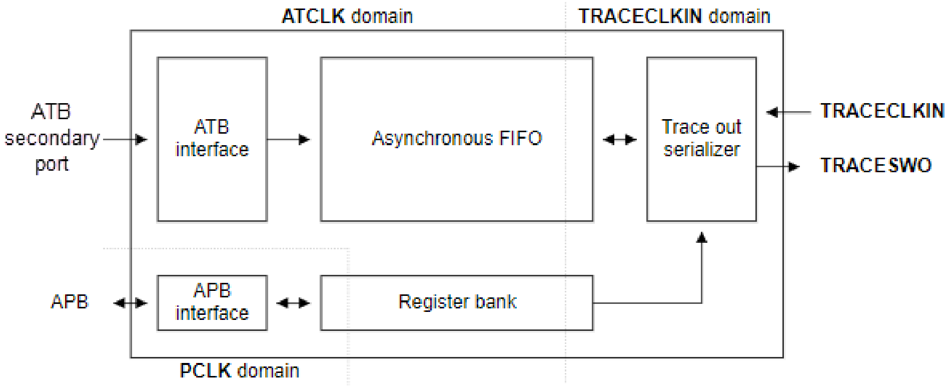
5.7.2.2 SAM Serial Wire Output (SWO)

The CoreSight 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). It is a TPIU-like device that supports a limited subset of the full TPIU functionality, to minimize gate count for a simple debug solution.

Compared to the 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 5-5. SWO Block Diagram



5.7.3 Hardware Setup

Preliminaries

1. Use USB communication between the PC and MPLAB ICE 4. 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 emulator-target communication and a connector for trace pins if using the 3.3.3. SAM MCUs - JTAG/SWD Adapter Board. Alternately design the target board to connect to the high-speed cable with pins for both debug and trace (see 3.3.1. Target Connection Pinout).
4. When using trace, the TRACESWO pin 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 emulator 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.

5.7.4 Set Up ITM in MPLAB X IDE

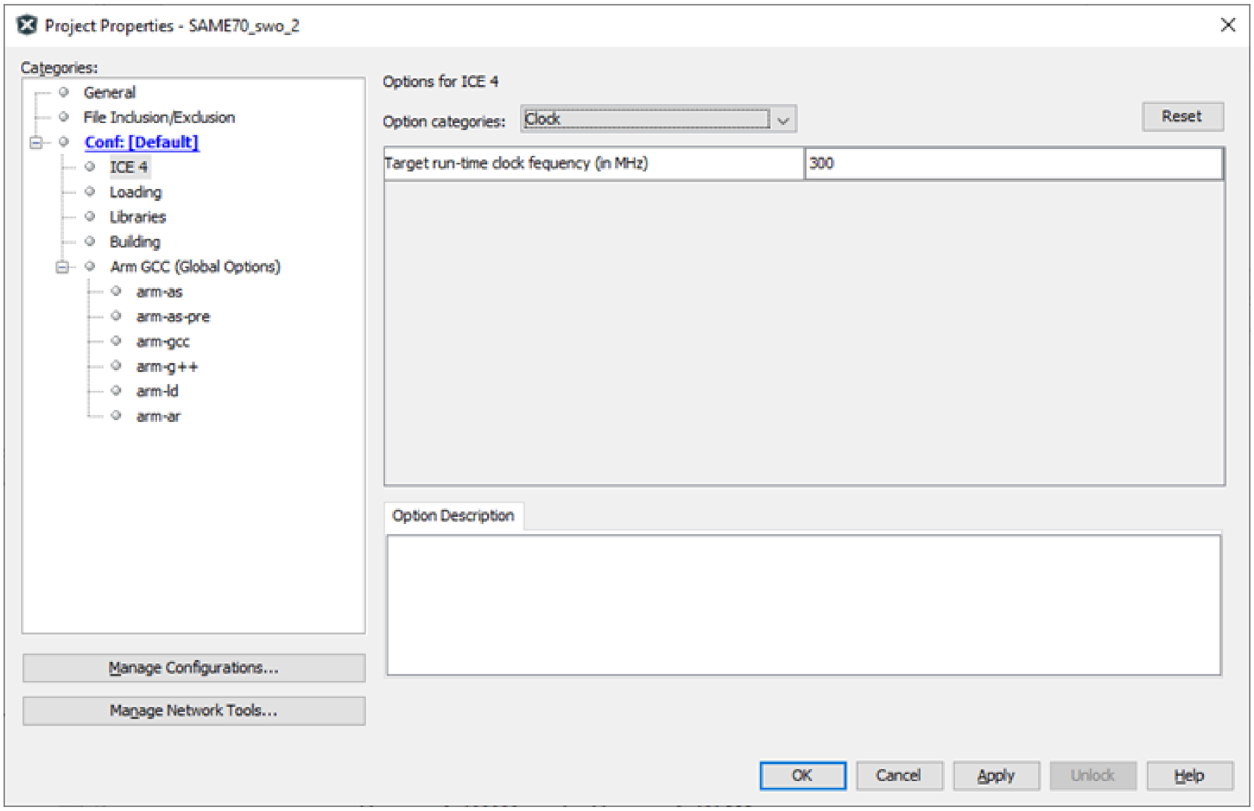
To set up MPLAB X IDE to use ITM for the MPLAB ICE 4 in-circuit emulator:

1. Right click on the project name in the Projects window and select “Properties.”
2. In the Project Properties dialog, click on “ICE4” (under “Categories”).
3. Set up the clock and trace as described in the following sections.

5.7.4.1 Setup the Clock

Under “Option categories,” select “Clock.” Enter the target run-time frequency.

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



Option categories: Clock

Target run-time clock fequency (in MHz)	300

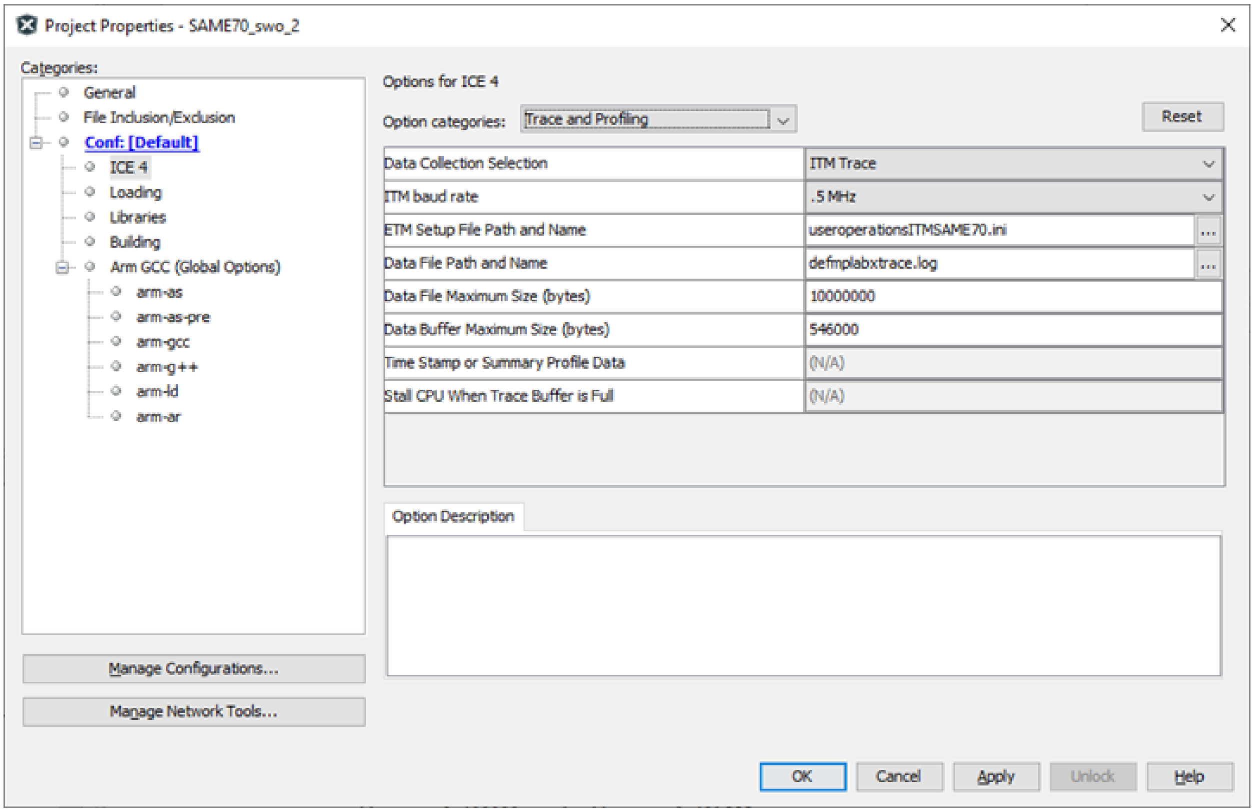
5.7.4.2 Setup ITM Trace

Under “Option categories,” select “Trace and Profiling.”

1. Under “Data Collection Selection,” choose “ITM Trace.”
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 emulator 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 5.7.6. Software Implementation.
4. Change any logging setup as desired.
5. Click **OK** when done.



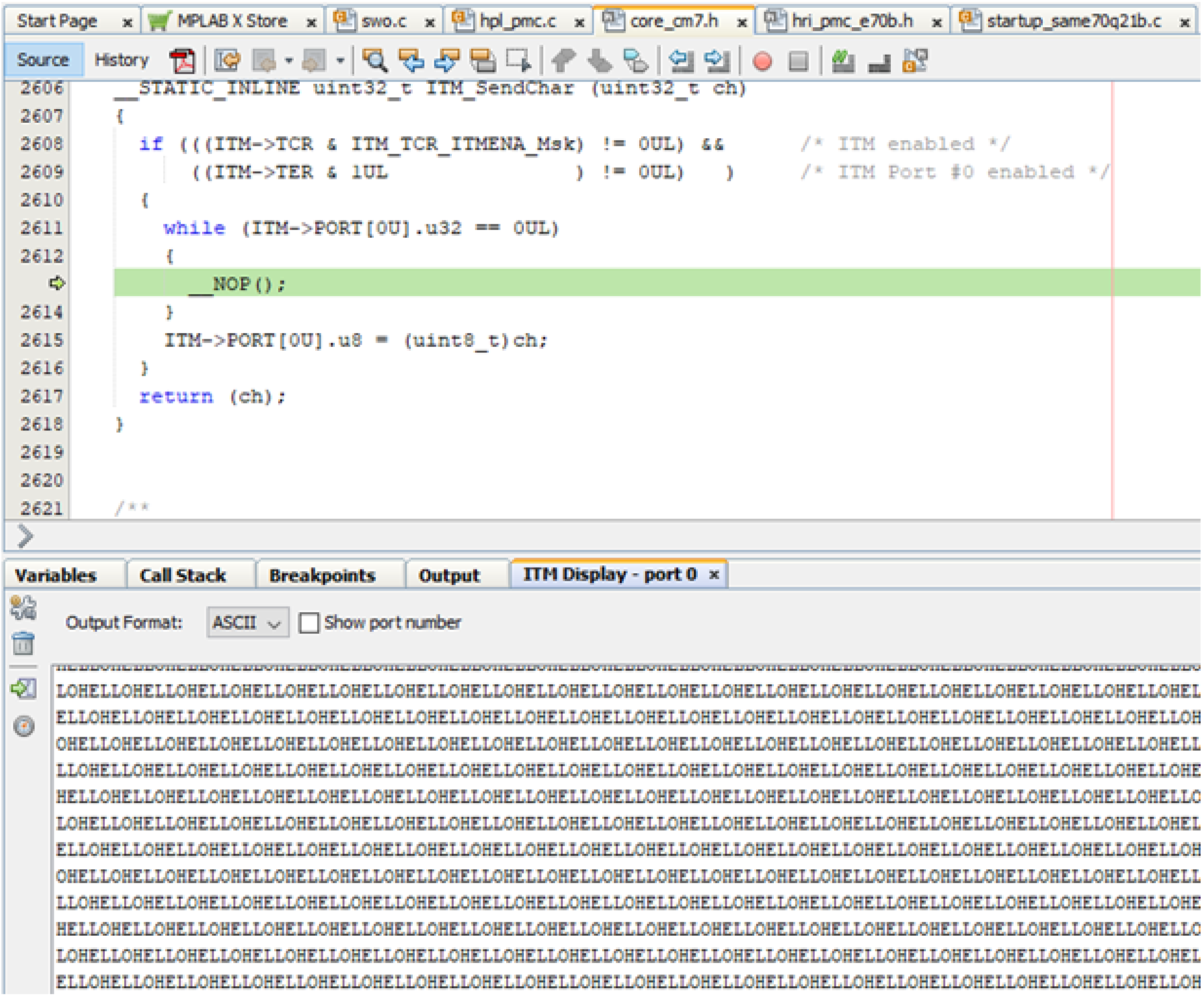
Option categories: Trace and Profiling

Data Collection Selection	ITM Trace
ITM baud rate	.5 MHz
ETM Setup File Path and Name	useroperationsITMSAME70.ini
Data File Path and Name	defmplabxtrace.log
Data File Maximum Size (bytes)	10000000
Data Buffer Maximum Size (bytes)	546000
Time Stamp or Summary Profile Data	(N/A)
Stall CPU When Trace Buffer is Full	(N/A)

5.7.5 Viewing ITM Data

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

Figure 5-6. Example Output in ITM Display



```
__STATIC_INLINE uint32_t ITM_SendChar (uint32_t ch)
{
  if (((ITM->TCR & ITM_TCR_ITMENA_Msk) != 0UL) &&      /* ITM enabled */
     ((ITM->TER & 1UL               ) != 0UL)   )     /* ITM Port #0 enabled */
  {
    while (ITM->PORT[0U].u32 == 0UL)
    {
      __NOP();
    }
    ITM->PORT[0U].u8 = (uint8_t)ch;
  }
  return (ch);
}


/**
```

Output Format: ASCII □ Show port number

```
LOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHEL
ELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOH
OHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELL
LLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHE
HELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLO
LOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHEL
ELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOH
OHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELL
LLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHE
HELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLO
LOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHEL
ELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOH
```

Related Links

9.3.2. ITM Window and Related Dialogs

5.7.6 Software Implementation

SAM E70 and SAM E54 devices 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 an MDK-ARM 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
```

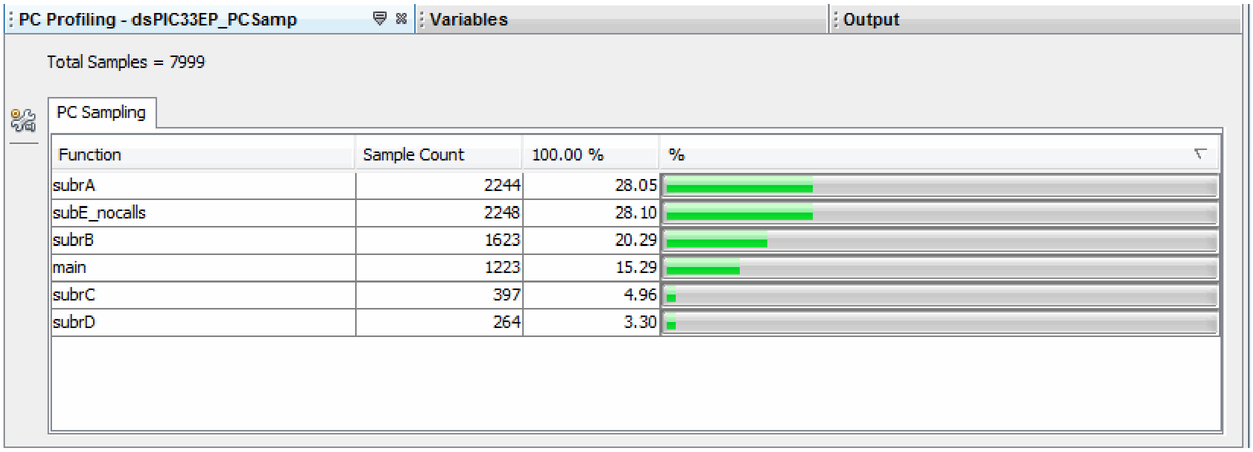
5.8 PC Sampling – 16-Bit PIC MCUs Only

PC sampling is a method of examining C code to determine the percentage of time that is spent in each function. This information can show you where your program time is being spent so you can work to optimize your code.

For PC sampling, a device timer is set up to take samples of program execution and display the results in the PC profiling window.

PC profiling is similar to PC sampling. For details see 5.9. PC Profiling – 32-Bit MCUs Only.

Figure 5-7. PC Sampling - 16-bit Device



Total Samples = 7999

Function	Sample Count	100.00 %	%
subrA	2244	28.05	
subE_nocalls	2248	28.10	
subrB	1623	20.29	
main	1223	15.29	
subrC	397	4.96	
subrD	264	3.30	

5.8.1 Requirements

Currently, to use PC sampling your project must be set up for:

- a supported device:
  - PIC24F, PIC24EP
  - dsPIC33FJ, dsPIC33E
- an MPLAB XC16 C Compiler version 1.10 or above

5.8.2 Clock Setup

To set up the clock:

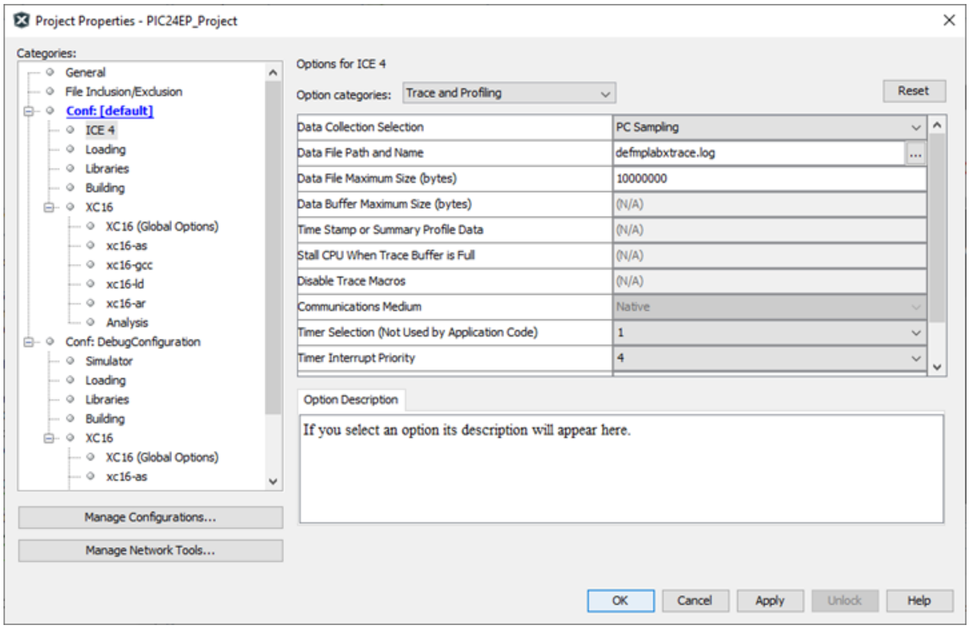
1. Open the Project properties window (*File>Project Properties*).
2. Go to *ICE4>Clock*. Ensure this value matches the actual target speed, i.e., as set by the Configuration bits in code.
3. Click **Apply**.

5.8.3 Sampling Setup

To set up sampling:

1. Select *ICE4>Trace and Profiling*. Under “Data Collection Selection,” select “PC Sampling.”
2. Set up your data file and timer in this window. For reference, see 9.2.6. Trace and Profiling.
   Note: Ensure the timer you select is not used in your program.
3. Click **OK**.

Figure 5-8. PC Sampling – Selection and Setup



Options for ICE 4

Option categories: Trace and Profiling

Data Collection Selection	PC Sampling
Data File Path and Name	defmplabxtrace.log
Data File Maximum Size (bytes)	10000000
Data Buffer Maximum Size (bytes)	(N/A)
Time Stamp or Summary Profile Data	(N/A)
Stall CPU When Trace Buffer is Full	(N/A)
Disable Trace Macros	(N/A)
Communications Medium	Native
Timer Selection (Not Used by Application Code)	1
Timer Interrupt Priority	4

Option Description

If you select an option its description will appear here.

5.8.4 Operation

To generate data:

1. Select *Window>Debugging>PC Profiling*. This will open the PC Profiling window.
2. Run your code and then halt.
3. View the sampling data in the window. Data is only displayed on halt.
4. Right click in the window to pop up a menu to either clear the data or reload the data.

You may also display this data in the Code Profiling plugin, available for purchase at Microchip Gallery:

gallery.microchip.com/

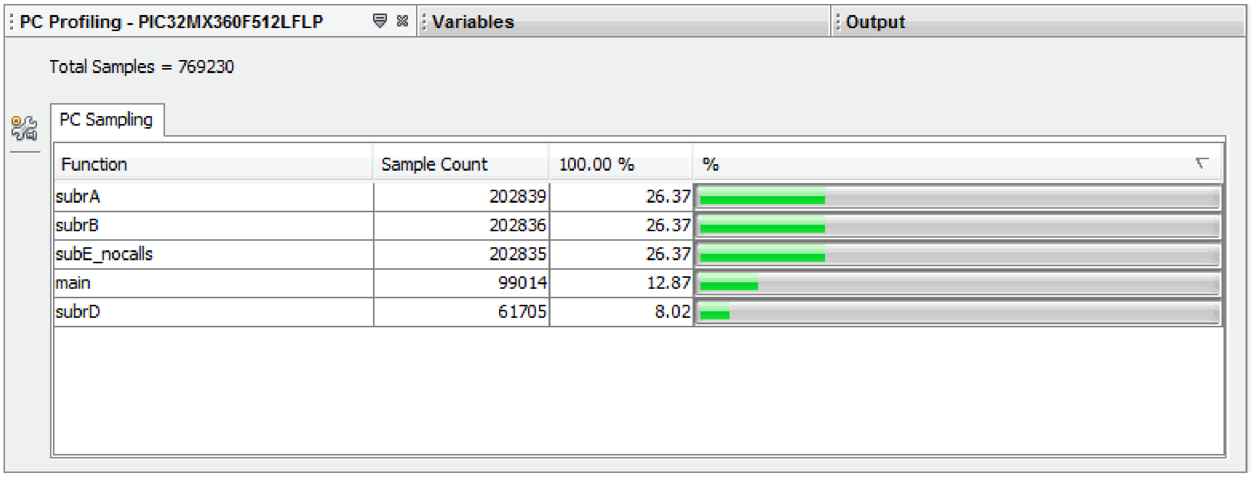
5.9 PC Profiling – 32-Bit MCUs Only

PC profiling is a method for examining C code to determine the percentage of time that is spent in each function. This information can show you where your program time is being spent so you can work to optimize your code.

For PC profiling, every program counter (PC) trace sample is taken from the trace (data) buffer and profiled. This data is displayed in the PC Profiling window. In this example, the trace buffer contains 769,230 PC sample data points. Of these, 99014 (or 12.87%) were associated with the `main()` function, 202,839 (or 26.37%) with the `subrA()` function, etc.

PC profiling is similar to PC sampling. For details see 5.8. PC Sampling – 16-Bit PIC MCUs Only.

Figure 5-9. PC Profiling - 32-bit Device



Total Samples = 769230

Function	Sample Count	100.00 %	%
subrA	202839	26.37	
subrB	202836	26.37	
subE_nocalls	202835	26.37	
main	99014	12.87	
subrD	61705	8.02	

5.9.1 Requirements

Currently, to use PC profiling your project must be set up for these supported devices:

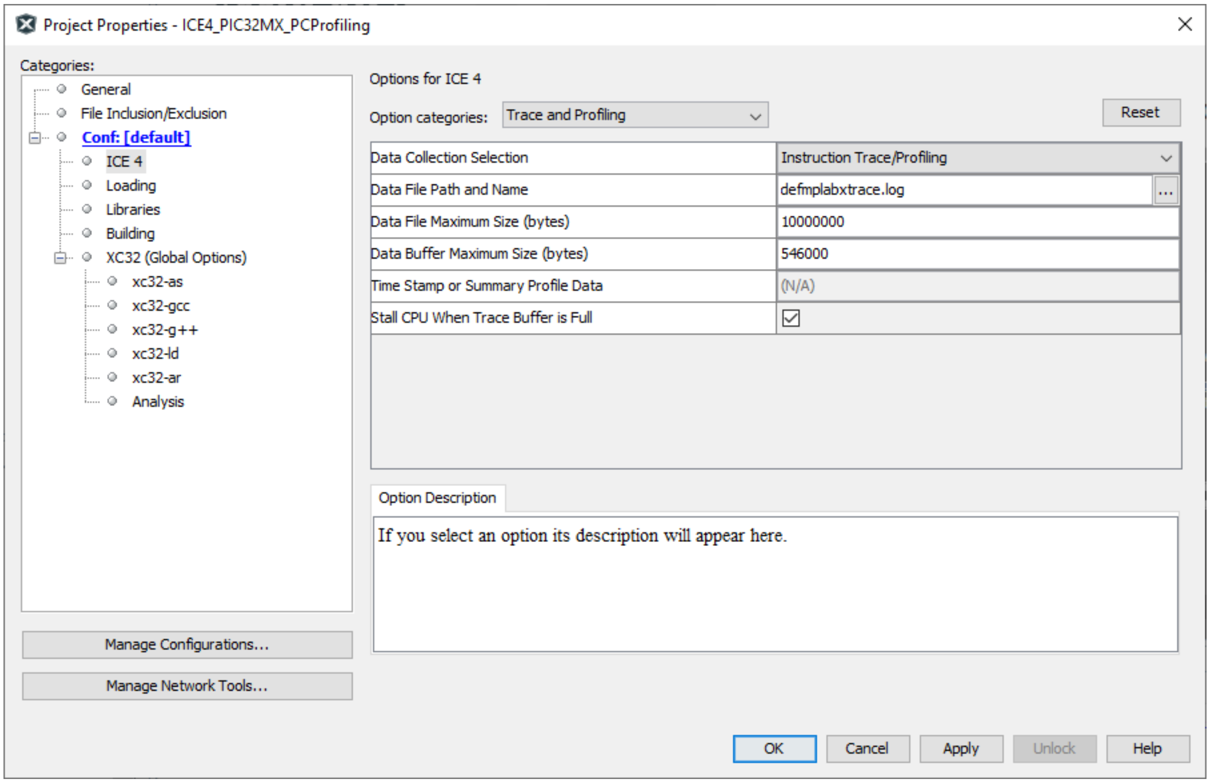
- PIC32MX with data capture – See *Help>Release Notes>Debug Features Support>Hardware Tool Debug Features by Device*.
  Note: Other PIC32 devices are NOT supported.

5.9.2 Profiling Setup

To set up profiling:

1. Set up your hardware for PIC32 Instruction trace (see 3.3.8. PIC32 Instruction Trace Adapter Board).
2. Open the Project properties window (*File>Project Properties*).
3. Select *ICE 4>Trace and Profiling*. Under “Data Collection Selection,” select “Instruction Trace/Profiling.”
4. Set up your data file in this window. For reference, see 9.2.6. Trace and Profiling. Then click **OK**.

Figure 5-10. PC Profiling – Selection and Setup



Project Properties - ICE4_PIC32MX_PCProfiling

Categories:

- General
- File Inclusion/Exclusion
- Conf: [default]
  - ICE 4
  - Loading
  - Libraries
  - Building
  - XC32 (Global Options)
    - xc32-as
    - xc32-gcc
    - xc32-g++
    - xc32-ld
    - xc32-ar
    - Analysis

Options for ICE 4

Option categories: Trace and Profiling

Data Collection Selection	Instruction Trace/Profiling
Data File Path and Name	defmplabxtrace.log
Data File Maximum Size (bytes)	10000000
Data Buffer Maximum Size (bytes)	546000
Time Stamp or Summary Profile Data	(N/A)
Stall CPU When Trace Buffer is Full	☑

Option Description

If you select an option its description will appear here.

5.9.3 Operation

To generate data:

1. Select *Window>Debugging>PC Profiling*. This will open the PC Profiling window.
2. Run your code and then pause/halt.
3. View the profiling data in the window. Data is only displayed on halt.
4. Right click in the window to pop up a menu to either clear the data or reload the data.

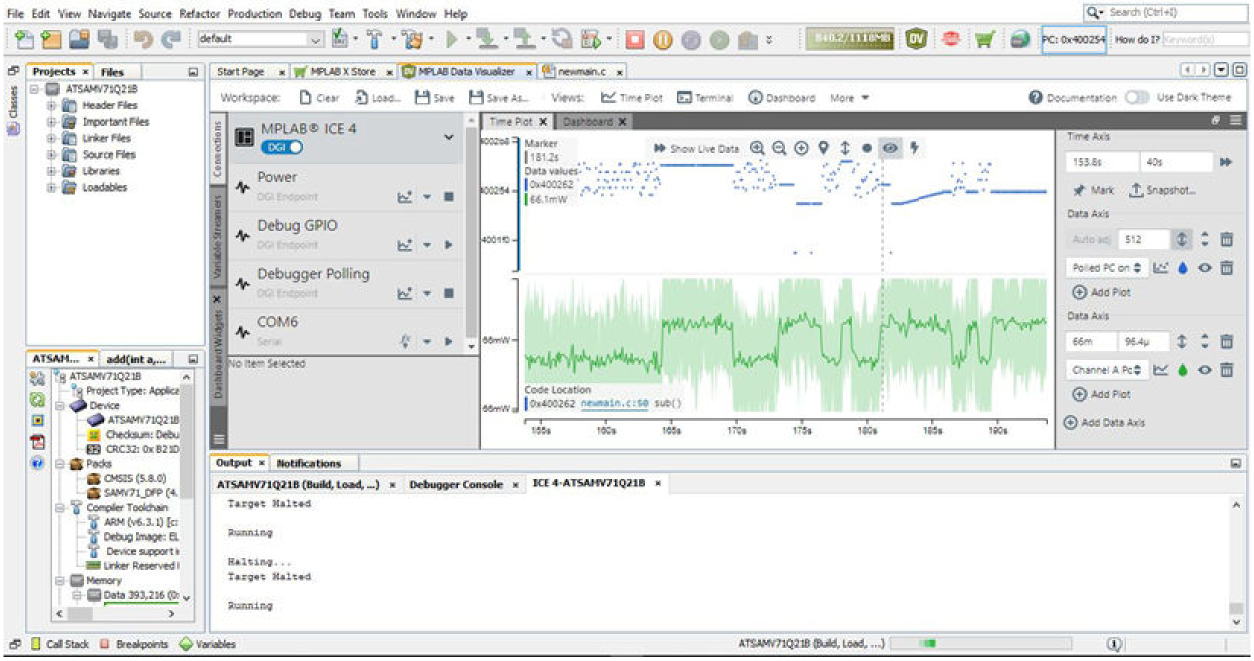
You may also display this data in the Code Profiling plugin, available for purchase at Microchip Gallery:

gallery.microchip.com/

5.10 Debugger Polling

MPLAB ICE 4 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.



```
Target Halted

Running

Halting...
Target Halted

Running
```

5.10.1 Requirements

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

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

5.10.2 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 DV 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.

5.11 Power Monitor

The MPLAB ICE 4 in-circuit emulator can be a power monitor. Power monitoring means capturing power data, such as current values. This feature uses two independent current sensing channels for measuring and optimizing the power consumption of a design.

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

The emulator 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.

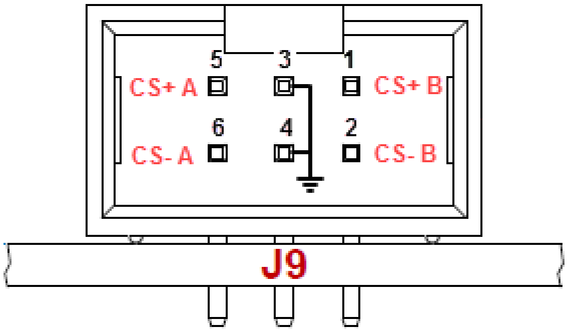
Note: You can only use USB communication with the data visualizer.

For more on this software, see: MPLAB Data Visualizer Product Page

5.11.1 Getting Started with Power Monitoring

The 6-pin current sense connector is located next to the 40-pin connector on the MPLAB ICE 4 unit. See also 10.5.4. Current Sense Module.

Figure 5-11. Current Sense Connections - Front View



Some examples of connections for different power measurements are listed below. Not all devices support all configurations.

Figure 5-12. Measure MCU Power

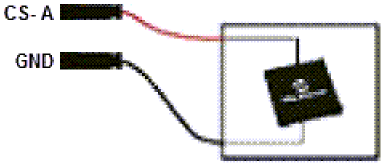
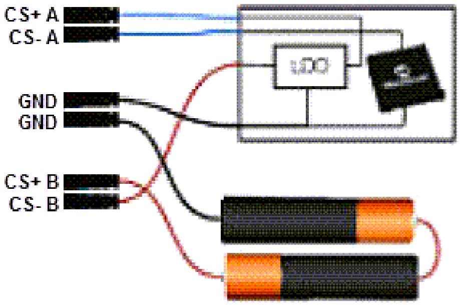


Figure 5-13. Separately Measure MCU and Board Power



5.11.2 Detailed Use Cases

Power monitor use cases can be found in the MPLAB Data Visualizer documentation. The hardware used is the Atmel Power Debugger, but the setup is similar to what would be needed for the MPLAB® ICE 4 In-Circuit Emulator.

A simple use case using the MPLAB ICE 4 is detailed below.

5.11.2.1 Low-Power Application

Using MPLAB ICE 4 with MPLAB X IDE and MPLAB Data Visualizer, learn how the emulator can be used to measure, analyze, understand, and optimize the power consumption of a typical low-power application. There is also an example of code instrumentation using the Data Gateway Interface (DGI).

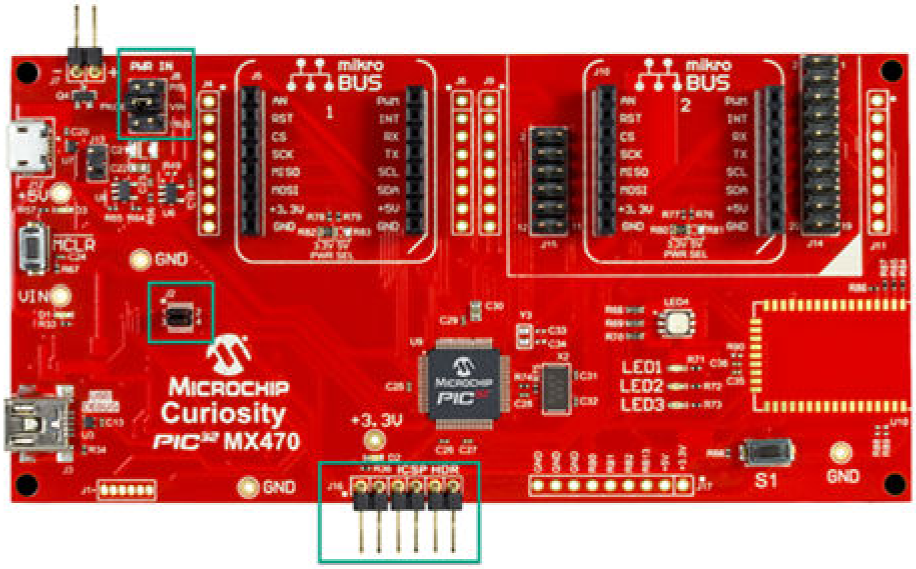
5.11.2.1.1 Requirements

To be able to work through this example, the following is required:

- A computer with MPLAB X IDE v6.00 (or later) installed. The latest MPLAB Data Visualizer plugin is available from the IDE.
- Microchip MPLAB ICE 4 Kit (cabling included)
- Microchip PIC32MX470 Curiosity board or similar target board
- Pin headers: one 6-pin 100-mil ICSP header
- Access to basic soldering equipment

5.11.2.1.2 Hardware Setup

To get started a few modifications need to be made to the hardware. The areas of the PIC32MX470 Curiosity board that need modification are shown in the figure below.



In order to use the MPLAB ICE 4 with the PIC32MX470 Curiosity board, the ICSP header area (J16) must be populated.

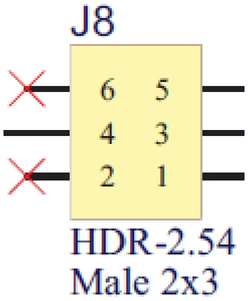
To do: Mount a 6-pin ICSP header on the Curiosity board.

The PIC32MX470 Curiosity board already contains a programmer/debugger (PK4oB). So to use the MPLAB ICE 4 as a programmer/debugger, the on-board debugger must be disabled.

To do: Remove jumper J2 to disconnect the on-board debugger.

The MPLAB ICE 4 will be used to power the PIC32MX470 Curiosity board. This will be set up in software later. Since neither USB connections will be used to power the board, the Power In jumper (J8) must select an external + 5V power supply.

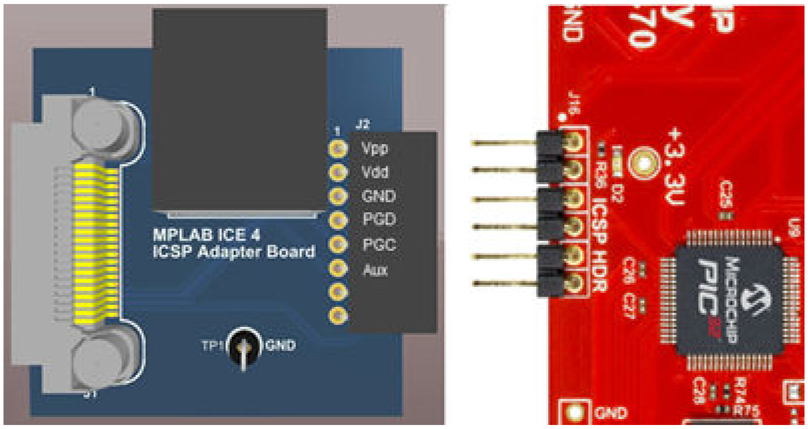
To do: Move J8 jumper to position 5-3 (P/S).



5.11.2.1.3 Connections

Once the hardware modifications have been made, connect the MPLAB ICE 4 to the target by connecting the MPLAB ICE 4 target cable assembler into the populated ICSP header on the PIC32MX470 Curiosity board.

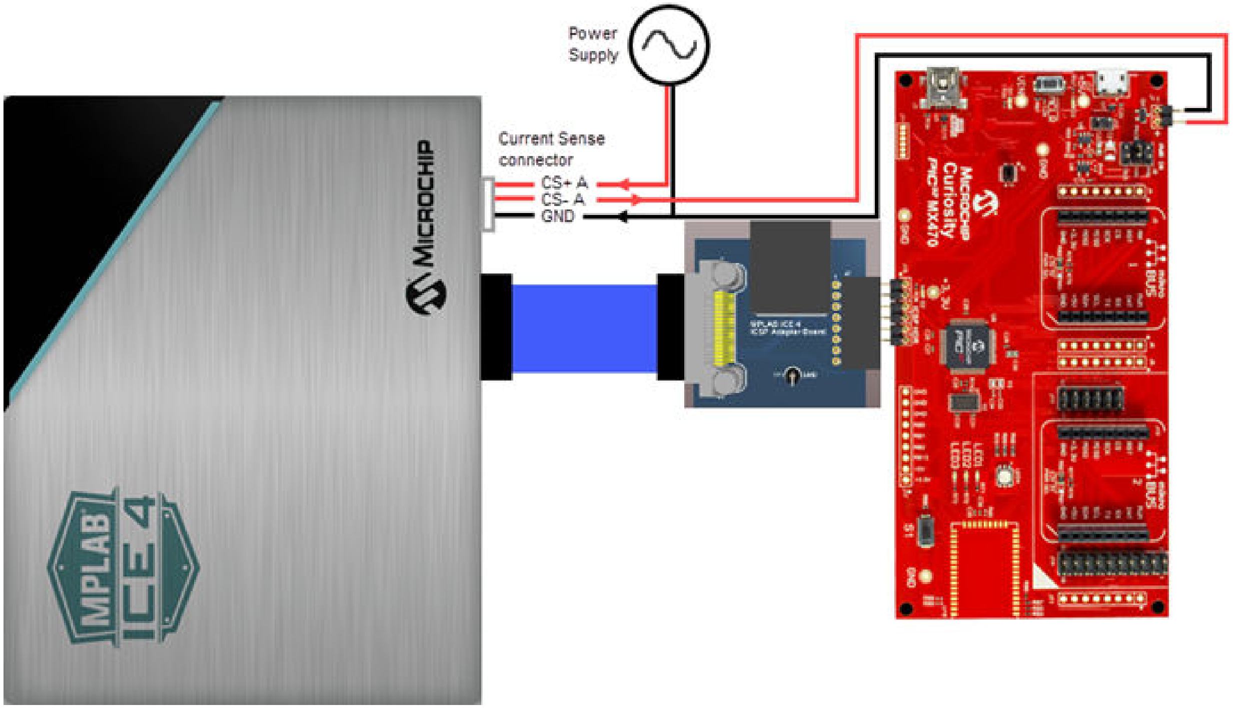
To do: Plug one end of the high-speed cable into the 40-pin connector on the emulator unit. Plug the other end into the MPLAB ICE 4 ICSP Adapter board. Connect the 8-pin SIL connector on the adapter board into the target board ICSP header. Ensure that the header pin 1 goes into the first socket of the SIL connector, as below.



In order to measure the current consumed by the target, current from an external source will need to pass through the emulator current sense A pins to the 5V external power header on the target board.

To do: The power supply connects to the current sense connector CS+ A and GND. Then CS- A is connected to the + pin of the 5V header. The - pin will connect back to a current sense GND pin.

Your setup should look something like this:



Related Links

10.5.4. Current Sense Module

5.11.2.1.4 Project Creation

In MPLAB X IDE, click the **New Project** button and set up a new stand-alone project.

1. Device: PIC32MX470F512H
2. Tool: MPLAB ICE 4
3. Compiler: MPLAB XC32 v2.50

In the new project, right click on “Source Files” in the Projects window and select *new>main.c*. Then enter the code below in the editor and save.

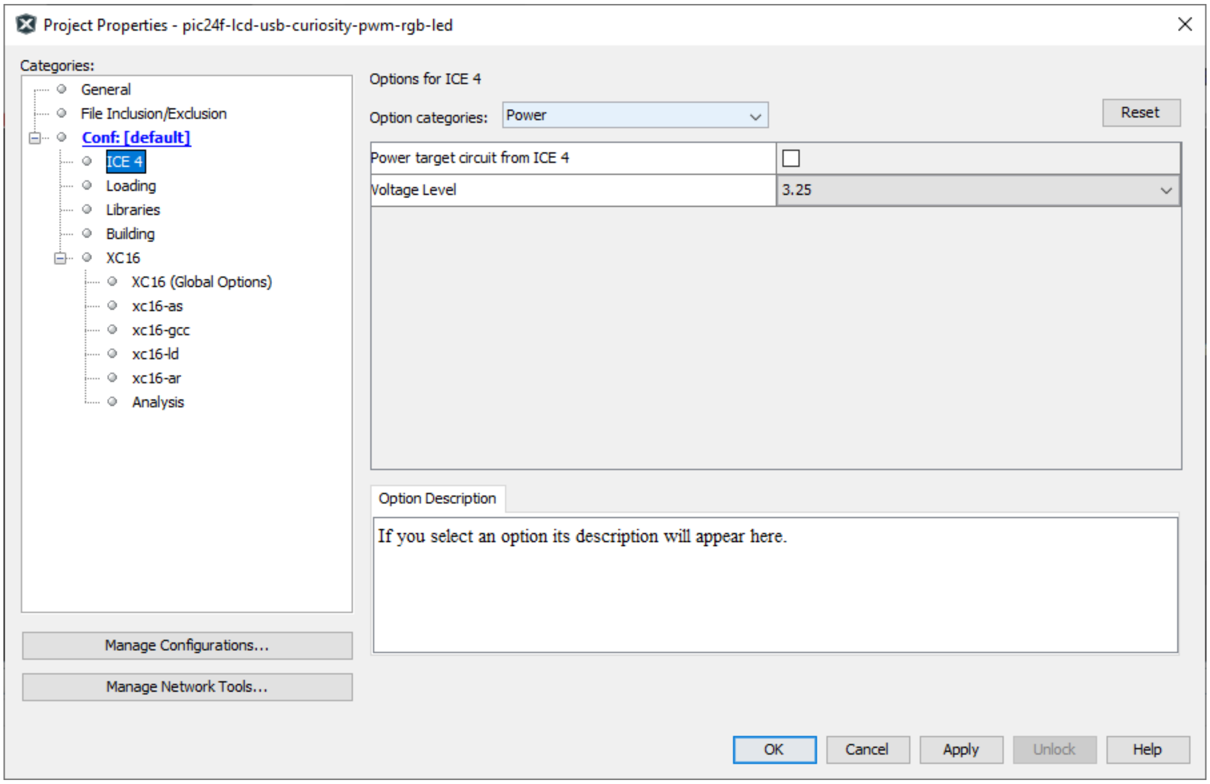
To produce current on the target board that is more than a single value, the target should be actively doing something. The following code for the target PIC32MX470F512H will flash LEDs on the board.

```
// PIC32MX470F512H Configuration Bit Settings

// 'C' source line config statements

// DEVCFG3
#pragma config USERID = 0xFFFF          // Enter Hexadecimal value (Enter Hexadecimal value)
#pragma config FSRSSEL = PRIORITY_7     // Shadow Register Set Priority Select (SRS Priority
7)
#pragma config PMDL1WAY = ON            // Peripheral Module Disable Configuration (Allow
only one reconfiguration)
#pragma config IOL1WAY = ON             // Peripheral Pin Select Configuration (Allow only
one reconfiguration)
#pragma config FUSBIDIO = ON            // USB USID Selection (Controlled by the USB Module)
#pragma config FVBUSONIO = ON           // USB VBUS ON Selection (Controlled by USB Module)

// DEVCFG2
#pragma config FPLLIDIV = DIV_12        // PLL Input Divider (12x Divider)
#pragma config FPLLMUL = MUL_24         // PLL Multiplier (24x Multiplier)
#pragma config UPLLIDIV = DIV_12        // USB PLL Input Divider (12x Divider)
#pragma config UPLLEN = OFF             // USB PLL Enable (Disabled and Bypassed)
#pragma config FPLLODIV = DIV_256       // System PLL Output Clock Divider (PLL Divide by 256)

// DEVCFG1
```

```
#pragma config FNOSC = FRCDIV           // Oscillator Selection Bits (Fast RC Osc w/Div-by-N
(FRCDIV))
#pragma config FSOSCEN = ON             // Secondary Oscillator Enable (Enabled)
#pragma config IESO = OFF               // Internal/External Switch Over (Disabled)
#pragma config POSCMOD = OFF            // Primary Oscillator Configuration (Primary osc
disabled)
#pragma config OSCIOFNC = OFF           // CLKO Output Signal Active on the OSCO Pin
(Disabled)
#pragma config FPBDIV = DIV_8           // Peripheral Clock Divisor (Pb_Clk is Sys_Clk/8)
#pragma config FCKSM = CSDCMD           // Clock Switching and Monitor Selection (Clock
Switch Disable, FSCM Disabled)
#pragma config WDTPS = PS1048576        // Watchdog Timer Postscaler (1:1048576)
#pragma config WINDIS = OFF             // Watchdog Timer Window Enable (Watchdog Timer is in
Non-Window Mode)
#pragma config FWDTEN = OFF             // Watchdog Timer Enable (WDT Disabled (SWDTEN Bit
Controls))
#pragma config FWDTWINSZ = WINSZ_25     // Watchdog Timer Window Size (Window Size is 25%)

// DEVCFG0
#pragma config DEBUG = OFF              // Background Debugger Enable (Debugger is Disabled)
#pragma config JTAGEN = OFF             // JTAG Enable (JTAG Disabled)
#pragma config ICESEL = ICS_PGx2        // ICE/ICD Comm Channel Select (Communicate on PGEC2/
PGED2)
#pragma config PWP = OFF                // Program Flash Write Protect (Disable)
#pragma config BWP = OFF                // Boot Flash Write Protect bit (Protection Disabled)
#pragma config CP = OFF                 // Code Protect (Protection Disabled)

// #pragma config statements should precede project file includes.
// Use project enums instead of #define for ON and OFF.

#include <xc.h>

// Three LEDs: Red, Green, Yellow
#define LEDS_ON 0xD0
#define LEDS_OFF 0x00

void delay (void)
{
    int n = 50000;
    while(n>0) {n--;}
}

int main(void) {

    // Port E access
    TRISE = 0x0000;   // set all port bits to be output

    while(1) {

        // delay value change
        delay();

        LATE = LEDS_ON; // write to port latch

        // delay value change
        delay();

        LATE = LEDS_OFF; // write to port latch

    }

    return -1;
}
```

5.11.2.1.5 Software Setup

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



Option categories: Power

Power target circuit from ICE 4	□
Voltage Level	3.25

If you select an option its description will appear here.

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

Troubleshooting:

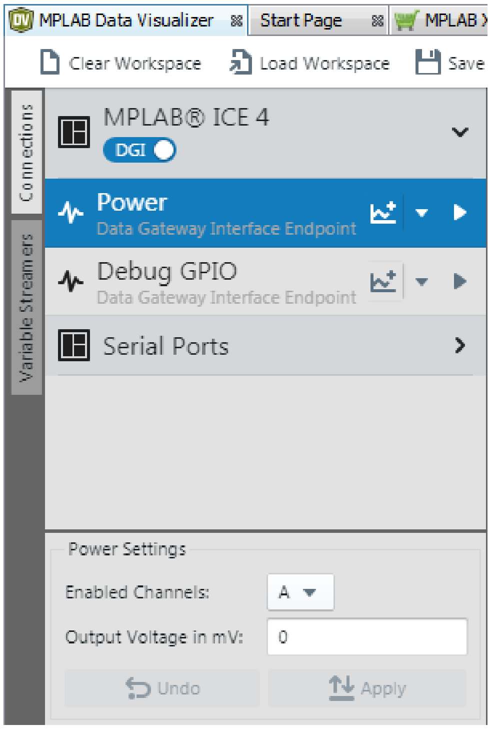
- If the project fails to build, check that you have copied and pasted the code fully. Also look at the error messages in the Output window for additional help.
- If MPLAB X IDE has connection issues with the emulator or the target, check your connections.

5.11.2.1.6 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 ICE 4 DGI list, as current sense is being used. Click on it to view Power Setting controls. For this use case, no power from the emulator will be used (Output Voltage = 0).

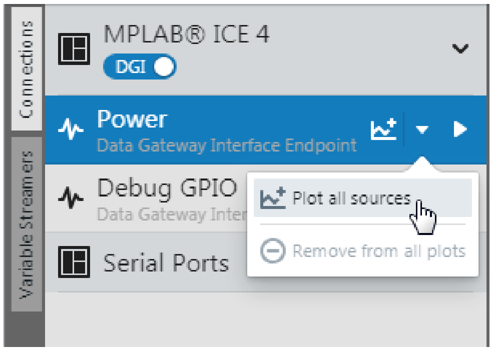
Figure 5-14. MPLAB ICE 4 DGI Options



Enabled Channels: A

Output Voltage in mV: 0

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



6. Troubleshooting First Steps

If you are experiencing problems with MPLAB ICE 4 In-Circuit Emulator operation, the following sections are provided to help.

For information on the meaning of MPLAB ICE 4 unit lights, see 10.3. Indicator Lights (LEDs).

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?** The emulator can only power the target when it is powered by an external 9V power supply. For details see 10.2. Power Specifications.
4. **Are you using a USB hub in your set up?** Is it powered? If you continue to have problems, try using the emulator without the hub (plugged directly into the PC.)
5. **Are you using a communication cable shipped with emulator?** 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

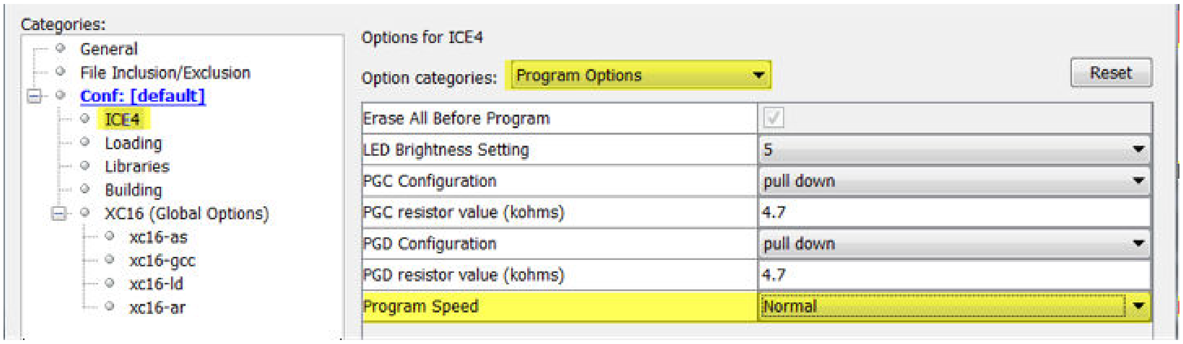
1. **Oscillator not working**. Check your Configuration bits setting for the oscillator. If you are using an external oscillator, try using an internal oscillator. If you are using an internal PLL, make sure your PLL settings are correct.
2. **No power to the target board**. Check the power cable connection to the target or the emulator if powered by the emulator.
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 emulator communication has somehow been interrupted. Reconnect to the emulator in MPLAB X IDE.
6. **Device not seated**. The device is not properly seated on the target board. If the emulator 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 Emulator 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 emulator 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 emulator cannot always perform the action requested. For example, the emulator cannot set a breakpoint if the target application is currently running.

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. Review emulator setup to ensure proper application setup. For more information, see 4. Operation.
4. Your program speed may be set too high for your circuit. In MPLAB X IDE, go to *File>Project Properties*, select **ICE4** 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



Option categories: Program Options

Erase All Before Program	☑
LED Brightness Setting	5
PGC Configuration	pull down
PGC resistor value (kohms)	4.7
PGD Configuration	pull down
PGD resistor value (kohms)	4.7
Program Speed	Normal

5. 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

⚠ CAUTION 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.

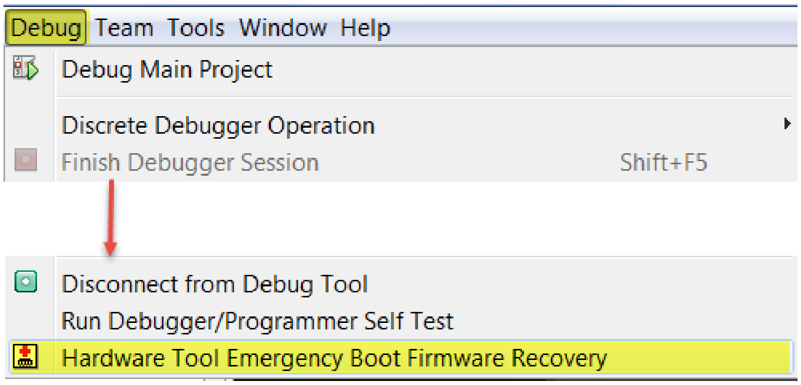
The debugger may need to be forced into recovery boot mode (reprogrammed) in rare situations; for example, if any of the following occurs when the debugger is connected to the computer:

- If the debugger has no LEDs lit.
- If the LEDs are cyan in color.
- If the procedure described in How to Invoke the Bootload Mode was not successful.

Important: YOU MUST USE MPLAB X IDE v6.00 OR GREATER TO USE THE EMERGENCY RECOVERY UTILITY FOR MPLAB ICE 4.

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


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

If the procedure was successful, the recovery wizard displays a success screen. The MPLAB ICE 4 will now be operational and able to communicate with the MPLAB X IDE. If the procedure failed, try it again. If it fails a second time, contact Microchip Support at support.microchip.com.

7. Frequently Asked Questions (FAQ)

Answers to frequently asked questions about the MPLAB ICE 4 In-Circuit Emulator 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 ICE 4 In-Circuit Emulator?
MPLAB ICE 4 can communicate with Flash silicon via the ICSP and other target interfaces. It uses the debug executive located in dedicated memory. For legacy 8-bit PIC devices, the debug executive resides in Program memory.

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

How does MPLAB X IDE interface with the MPLAB ICE 4 In-Circuit Emulator to allow more features than older debug tools?
MPLAB ICE 4 communicates using the debug executive located in a dedicated area of memory. The debug executive is streamlined for more efficient communication. The debugger contains an FPGA, large SRAM Buffers (1Mx8), and a High-Speed USB interface. Program memory image is downloaded and is contained in the SRAM to allow faster programming. The FPGA in the debugger serves as an accelerator for interfacing with the device in-circuit debugger modules.

On traditional 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 ICE 4 In-Circuit Emulator? 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 theMPLAB ICE 4, so there are no buses to monitor externally. With the MPLAB ICE 4, 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 ICE 4 In-Circuit Emulator have complex breakpoints?
Yes. You can break based on a value in a data memory location. You can also do sequenced breakpoints, where several events have to occur before it breaks. However, you can only do two sequences. You can also do the AND condition and do PASS counts.

What limitations are there with the standard cable?
The standard ICSP-RJ11 cable maximum clock frequency is approximately 15 MHz. Device interrogation during debugging occurs at frequencies below this rate regardless of the CPU clock rate. However, some advanced functions are synchronous to the CPU bus cycle (like instrumented trace and data capture). During data capture and when the CPU runs at its highest speed (40 MIPS for example), the actual clock rate through the cable would exceed 15 MHz. In these instances, trace and data capture cannot run reliably and the ICSP-RJ11 cable cannot be used.

Will this slow down the running of the program?
There is no cycle stealing with the MPLAB ICE 4. The output of data is performed by the state machine in the silicon.

Is it possible to debug a dsPIC DSC device running at any speed?
The MPLAB ICE 4 is capable of debugging at any device speed as specified in the device's data sheet.

7.2 What's Wrong?

When MPLAB ICE 4 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 emulator for prolonged periods of time, and especially as a debugger, be sure to disable the Hibernate mode in the Power Options Dialog window of your 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 emulator. 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 ICE 4 In-Circuit Emulator 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

ICE 4 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 ICE 4 is currently busy and cannot remove software breakpoints at this time.
CHECK_4_HIGH_VOLTAGE_VPP=CAUTION: Check that the device selected in MPLAB IDE (%s) is the same one that is physically attached to the debug tool. Selecting a 5V device when a 3.3V device is connected can result in damage to the device when the debugger checks the device ID. Do you wish to continue?
CHECK_PGM_SPEED=You have set the program speed to %s. The circuit on your board may require you to slow the speed down. Please change the setting in the tool properties to low and try the operation again.
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=ICE 4 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 ICE 4 project properties) can mitigate the situation. The abort operation can be used to terminate the read operation if necessary.
DEVICE_ID_REVISION=Device Id Revision.
DEVICE_ID=Device Id.
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 too 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 semiphores.
FAILED_STEPPING=Failed while stepping the target.
FAILED_TO_GET_DEVID=Failed to get Device ID. Please make sure the target device is attached and try the operation again.
FAILED_TO_INIT_TOOL=Failed to initialize ICE 4.
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=ICE 4 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 ICE 4 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=ICE4 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 ICE4 firmware in 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=ICE 4
OPERATION_NOT_SUPPORTED=This operation is not supported for the selected device.
OUTPUTWIN_TITLE=ICE 4.
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=ICE 4 is busy. Please wait for the current operation to finish.
TOOL_VDD=VDD:

TOOL_VPP=VPP:
UNABLE_TO_OBTAIN_RESET_VECTOR=ICE 4 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=ICE 4 was unloaded while still busy. Please unplug and reconnect the USB cable before using ICE 4 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 ICE4 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 ICE 4 In-Circuit Emulator 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 ICE 4 In-Circuit Emulator 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 ICE 4 In-Circuit Emulator.
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 ICE 4 In-Circuit Emulator was unable to perform a debugging operation. There are numerous reasons why this might occur. See section Troubleshooting.

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. Emulator Function Summary

A summary of MPLAB® ICE 4 In-Circuit Emulator functions is provided in the following topics.

9.1 Emulator Selection and Switching

Use the Project Properties dialog to select or switch emulators for a project. To switch you must have more than one MPLAB® ICE 4 In-Circuit Emulator connected to your computer. MPLAB X IDE will differentiate between the two by displaying two different serial numbers.

To select or change the emulator used for a project:

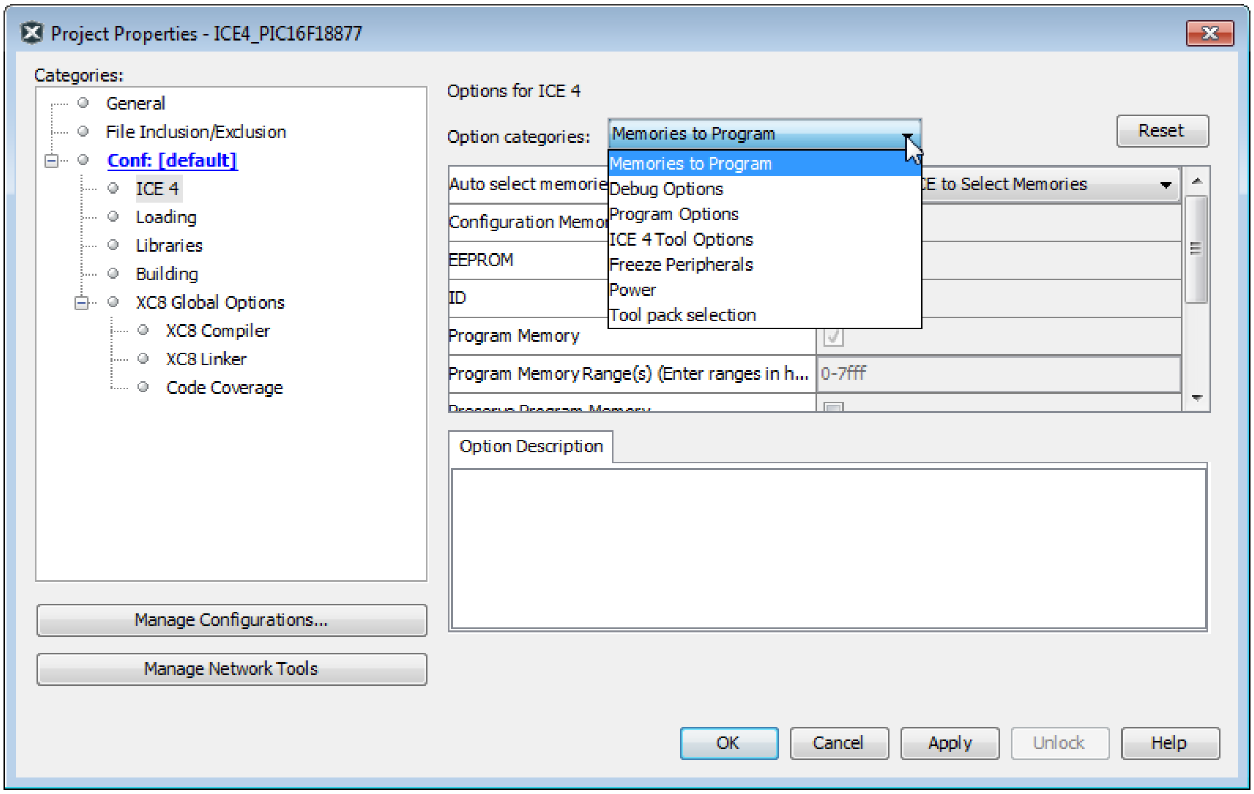
1. Open the Project Properties dialog by doing one of the following:
   a. Click on the project name in the Projects window and select *File>Project Properties*.
   b. Right click on the project name in the Projects window and select "Properties."
2. Under "Categories," click on "Conf: [default]."
3. Under "Hardware Tools," find "ICE 4" and click on a serial number (SN) to select an emulator for use in the project.

9.2 Emulator Options Selection

Set up emulator options on the emulator 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 "ICE 4"
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 - ICE 4 Options



Option categories: Memories to Program

Auto select memorie	...E to Select Memories
Configuration Memo	
EEPROM	
ID	
Program Memory	☑
Program Memory Range(s) (Enter ranges in h...	0-7fff
Preserve Program Memory	□

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 ICE 4 to Select Memories** – The emulator 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.
Memory	Check to program *Memory*. Types of memory include: Instruction RAM, Flash Data, Data Flash, EEPROM, ID, Boot Flash, Auxiliary.
Memory Range(s) (hex)*	The starting and ending hex address range in *Memory*. 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 *Memory*	Check to preserve *Memory* for reprogramming. Types of memory include: Instruction RAM, Flash Data, Data Flash, EEPROM, ID, Boot Flash, Auxiliary. Ensure code is NOT code protected.
Preserve *Memory* Range(s) (hex)*	The starting and ending hex address range in target *Memory* 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 *Tools>Options>Embedded>Generic Settings*, but may be changed here: Use system settings, Run, Halt at main, Halt at reset vector.
Debug reset	System settings may be found under *Tools>Options>Embedded>Generic Settings*, 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 ICE 4 Tool Options

Set up MPLAB ICE 4 specific options.

Table 9-5. ICE 4 Tool Options Category

Programming Mode Entry	Select programming mode entry type: • Use high voltage program mode entry • Use low voltage program mode entry
LED Brightness Setting	Set unit LED brightness on a scale of 1 (darkest) to 10 (lightest).
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 *Peripheral List*	Check to select which peripheral(s) to freeze. Uncheck to unfreeze all peripherals. This option applies to AC244066.
Peripheral List	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 emulator.

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 emulator.

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 ICE4 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 emulator. 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 emulator 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 ICE 4 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 emulator.

Table 9-9. Power Option Category

Power Target Circuit from ICE 4	Check to use power from the emulator. 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 emulator 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 emulator 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 Communications

Select the type of communication to be used between the MPLAB ICE 4 and the target. Communication options depend on project device. Some examples are show below.

For details, see 3.3. Target Connections and 4. Operation.

Table 9-11. Communications Options - ATtiny MCU

Interface	UPDI
Speed	Select the default value to change. Click on this row to see a range of available speed values under Option Description.
High Voltage Activation Mode	No High Voltage, Simple High Voltage Pulse, User Power Toggle. Click on this row to see definitions of selections under Option Description. See also UPDI High-Voltage Activation Information.

Table 9-12. Communications Options - ATmega MCU

Interface	• debugWire • ISP
Speed	Select the default value to change. Click on this row to see a range of available speed values under Option Description. See also Debugging with debugWIRE for AVR® MCUs.

Table 9-13. Communications Options - SAM MCU

Interface	• JTAG • SWD - a subset of JTAG
Speed	Select the default value to change. Click on this row to see a range of available speed values under Option Description.

Table 9-14. Communications Options - PIC32M MCU

JTAG Method	• JTAG 2-wire • JTAG 4-wire

9.2.10 Tool Pack Selection

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

Table 9-15. 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.

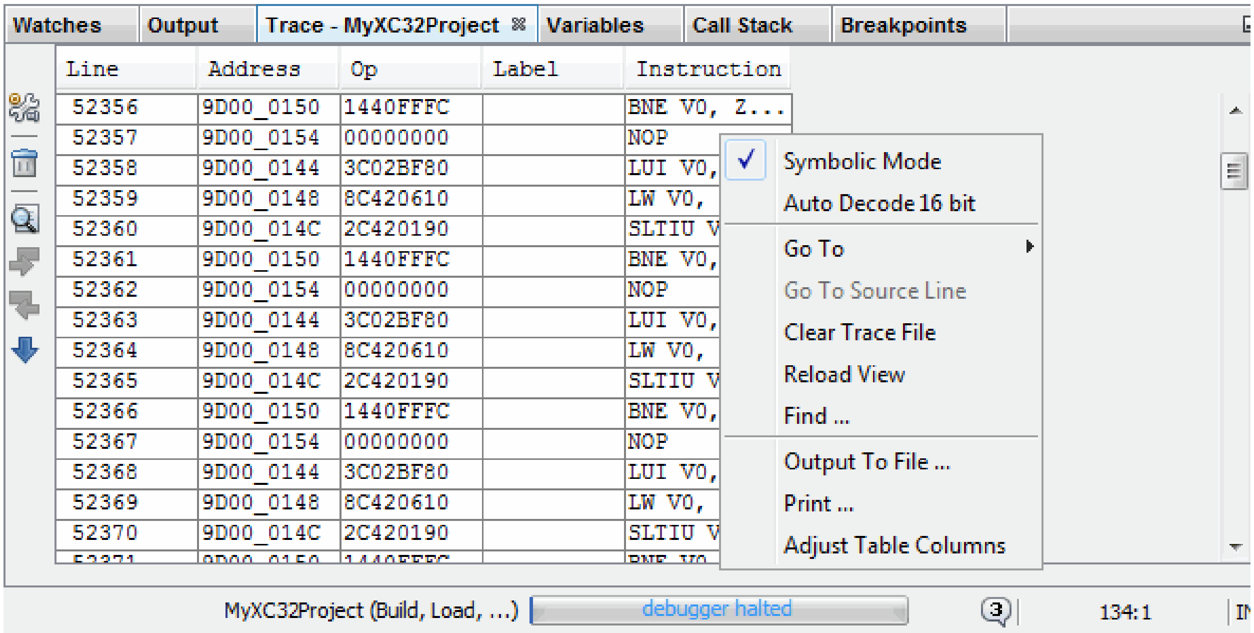
9.3 Emulator Windows & Dialogs

The following topics show windows and dialogs that are used specifically for the emulator or other emulator-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 emulator and the simulator.

Figure 9-2. Trace Window



Line	Address	Op	Label	Instruction
52356	9D00_0150	1440FFFC		BNE V0, Z...
52357	9D00_0154	00000000		NOP
52358	9D00_0144	3C02BF80		LUI V0,
52359	9D00_0148	8C420610		LW V0,
52360	9D00_014C	2C420190		SLTIU V
52361	9D00_0150	1440FFFC		BNE V0,
52362	9D00_0154	00000000		NOP
52363	9D00_0144	3C02BF80		LUI V0,
52364	9D00_0148	8C420610		LW V0,
52365	9D00_014C	2C420190		SLTIU V
52366	9D00_0150	1440FFFC		BNE V0,
52367	9D00_0154	00000000		NOP
52368	9D00_0144	3C02BF80		LUI V0,
52369	9D00_0148	8C420610		LW V0,
52370	9D00_014C	2C420190		SLTIU V
52371	9D00_0150	1440FFFC		BNE V0

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."

Related Links

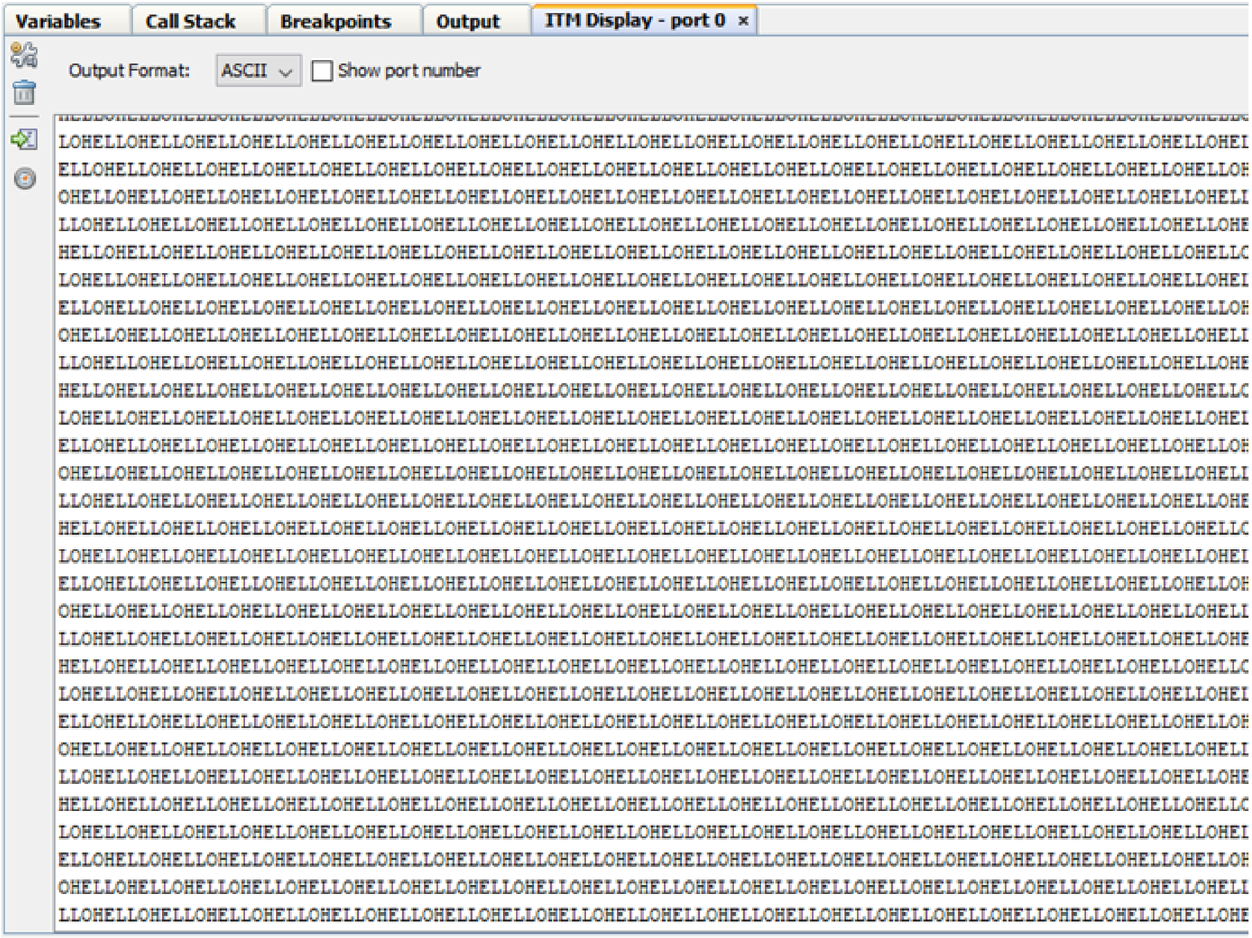
5.5. Instrumented Trace for PIC MCUs and dsPIC DSCs

5.6. Instruction Trace for PIC32M MCUs

9.3.2 ITM Window and Related Dialogs

The SAM Instruction Trace Microcell (ITM) produces a UART-format trace that supports `printf` style debugging.

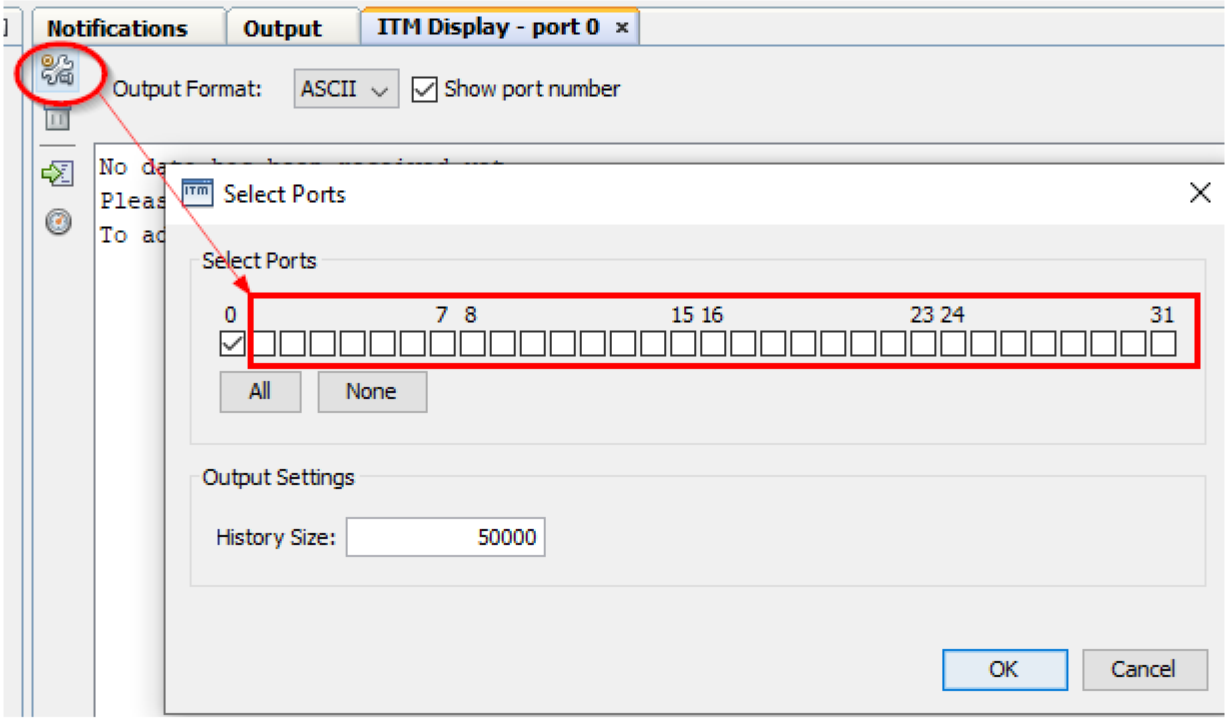
Figure 9-3. ITM Display



Output Format: ASCII ☐ Show port number

```
HELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLO
LOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHEL
ELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOH
OHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELL
LLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHE
HELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLO
LOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHEL
ELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOH
OHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELL
LLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHE
HELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLO
LOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHEL
ELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOH
OHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELL
LLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHE
HELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLO
LOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHEL
ELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOH
OHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELL
LLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHE
HELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLO
LOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHEL
ELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOH
OHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELL
LLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHE
HELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLO
LOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHEL
ELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOH
OHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELL
LLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHELLOHE
```

Figure 9-4. ITM Select Ports Dialog



Output Format: ASCII

☑ Show port number

```
No da
Pleas
To ad
```

☑□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

History Size: 50000

Related Links

5.7.2.1. SAM ITM Trace

10. Hardware Specification

The following sections provide details about the MPLAB ICE 4 unit and related hardware.

10.1 Emulator Unit

The MPLAB ICE 4 in-circuit emulator 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

External Power Connector (J2)

The connector is 9V DC +/- 5% center positive 2.5mm pin inner diameter/5.5mm outer diameter.

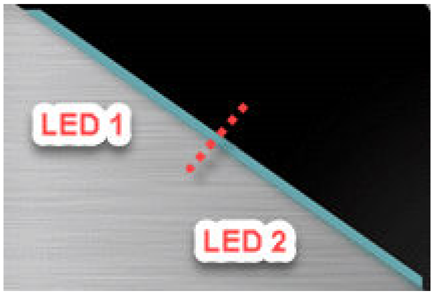
This connection is designed for the 9V Wall Mount Power Supply (Part Number AC002014), 110-220V universal power supply, which can provide power to the emulator and up to 1A of power to a target application.

Related Links

3.1. Power and Self Test

10.3 Indicator Lights (LEDs)

The top of MPLAB ICE 4 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

...........continued		
LED 1	**LED 2**	**Description**
White, slow blink	Blue	MPLAB X IDE/MPLAB IPE has initiated communication with ICE 4 over the network
Red	Blue	Network connection failure/error
Orange	Blue	Power target circuit from ICE 4 checked
Green	Blue	Power target circuit from ICE 4 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

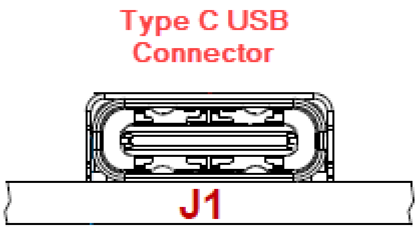
MPLAB® ICE 4 In-Circuit Emulator can be connected to the PC (and MPLAB X IDE) using one of the following:

Connection Type	Connection Details	Programming and Debugging	Trace
USB Type C (default)	SS USB 3.0, HS USB 2.0	USB 2.0 up to 480 Mbps	USB 3.0 up to 4.8 Gbps
Wi-Fi	Direct or via access point	up to 28 Mbps	No
Ethernet	Direct or via network	up to 100 Mbps	No

Connectors available on the MPLAB ICE 4 unit for communication are described in the following sections.

10.4.1 USB Type C Connector (J1) and Cable

A Type C USB connector and cable are provided for USB 3.0 SS or USB 2.0 communication between the debugger and a computer.



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 ICE 4 unit the PC (and MPLAB X IDE). 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 ICE 4 kit.

Figure 10-1. Ethernet Connector on MPLAB ICE 4

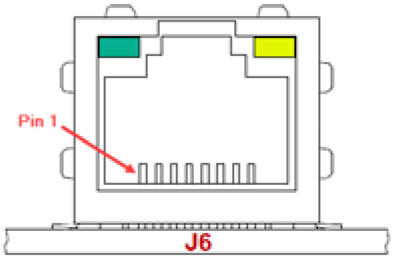


Table 10-2. Ethernet Connector Pinout

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 Target Connection Specifications

The following connectors are available on the MPLAB ICE 4 unit for the communication with the target.

10.5.1 40-pin Emulator Unit Connector (J10)

The 40-pin connector provides a 1x6 header for use with the edge rate high speed coax cable to facilitate communications with the target.

Figure 10-2. 40-pin Connector on MPLAB ICE 4

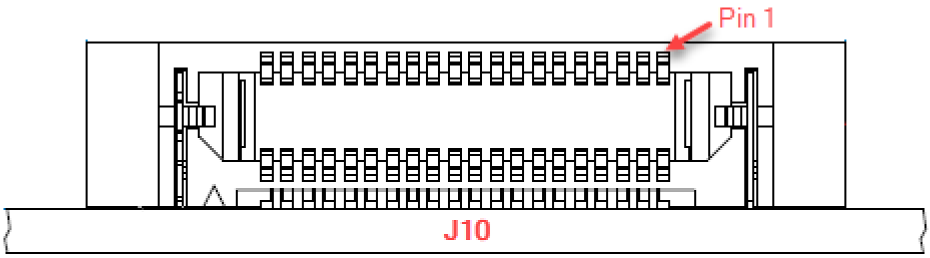


Figure 10-3. Emulator Connector Pinout

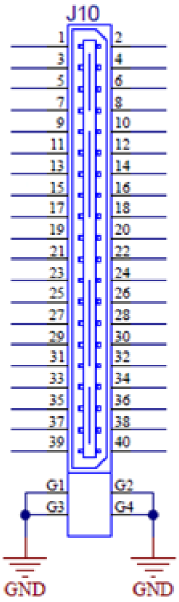


Table 10-4. Emulator Connector Pin Functions on Emulator Unit

Pin	Description	Function(s)	Pin	Description	Function(s)
1	TVDD PWR	TVDD PWR	2	TVDD PWR	TVDD PWR
3	TVPP IO	TVPP/MCLR, nMCLR, RST	4	TPGD IO	TPGD IO, TPGD, SWO,TDO, MISO, DAT
5	TPGC IO	TPGC IO, TPGC, SWCLK, TCK, SCK	6	TAUX IO	TAUX IO, AUX, DW, RESET
7	TDI IO	TDI IO, TDI, MOSI	8	TMS IO	TMS IO, SWD IO, TMS
9	TVDD PWR	TVDD PWR	10	GND	GND
11	DGI I2C SCL	DGI I2C SCL	12	TRDAT0	TRDAT0, TRACEDATA(0)
13	DGI I2C SDA	DGI I2C SDA	14	GND	GND
15	DGI VCP TXD	DGI TXD, CICD TXD, VCD TXD	16	TRDAT1	TRDAT1, TRACEDATA(1)
17	DGI VCP RXD	DGI RXD, CICD RXD, VCD RXD	18	GND	GND
19	5V0	Reserved	20	TRDAT2	TRDAT2, TRACEDATA(2)
21	DGI GPIO0	DGI GPIO0, PORT0, TRIG0	22	GND	GND
23	DGI GPIO1	DGI GPIO1, PORT1, TRIG1	24	TRDAT3	TRDAT3, TRACEDATA(3)

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

Pin	Description	Function(s)	Pin	Description	Function(s)
25	DGI GPIO2	DGI GPIO2, PORT2, TRIG2	26	GND	GND
27	DGI GPIO3	DGI GPIO3, PORT3, TRIG3	28	TRCLK	TRCLK, TRACECLK,
29	3V3	Reserved	30	GND	GND
31	DGI SPI MOSI	DGI SPI MOSI, SPI DATA, PORT5, TRIG5	32	DGI SPI MISO	DGI SPI MISO, PORT4, TRIG4
33	DGI SPI nCS	DGI SPI nCS,PORT6, TRIG6	34	DGI SPI SCK	DGI SPI SCK, SPI SCK, PORT7, TRIG7
35	UTIL SDA	Reserved	36	UTIL SCL	Reserved
37	CS- B	Power Monitor	38	CS+ B	Power Monitor
39	CS- A	Power Monitor	40	CS+ A	Power Monitor

10.5.2 Target Connection Cable

MPLAB® ICE 4 In-Circuit Emulator connects to a 40-pin ribbon cable assembly and either device-specific adapter boards for legacy targets or directly to a newer target. This provides device-supported communications.

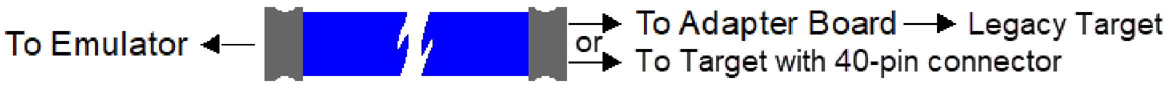


Table 10-5. Cable Assembly Specifications

Specification	Description
Manufacturer	Samtec
Cable Number	ERCD-020-08.00-TED-TED-3-B
Connector Number	ERF8-020-05.0-S-DV-L
Assembly Type	0.8mm Edge Rate High Speed Coax Cable Assembly
Assembly Length	9.125 inches / 23 cm

10.5.3 Adapter Boards and Cables

To support legacy target connections, adapter boards are available both in the MPLAB ICE 4 kit (DV244140) and in a Adapter Pack (AC244140). Cables from the adapter boards to legacy targets also are available in the MPLAB ICE 4 kit and in the Adapter Pack.

Table 10-6. Adapter Board Connectors and Cables

Adapter Board	Connectors	Cables
ICE4 JTAG Adapter	20-pin JTAG	Plugs into target board
	10-pin Mini*	2x5 IDC Socket Double Polarized
MPLAB ICE 4 ICSP Adapter Board + • ICE 4 AVR JTAG (10 Pin) • ICE 4 AVR (10 Pin Mini) • ICE 4 AVR (6 Pin) • ICE 4 AVR (6 Pin Mini)	8-pin SIL + • 10-pin • 10-pin Mini • 6-pin • 6-pin Mini	No cables. Adapter boards plug into complementary sockets on target board.

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

Adapter Board	Connectors	Cables
MPLAB ICE 4 ICSP Adapter Board	8-pin RJ-45	6" 6-pin modular cable, 6" 8-pin modular cable
	8-pin SIL	6" inline cable
MPLAB ICE 4 Cortex-M Trace Adapter Board	20-pin Mini	3" 20-pin Custom Dual Row Socket Assembly
MPLAB ICE 4 PIC32 Trace Adapter Board	14-pin DIL	10" 14-pin Dual Row Socket IDC Assembly
	9-pin trace	N/A

* The 10-pin JTAG/SWD cable P/N CAB1013 included in the kit *may* have a strain relief that prevents it from mounting into a keyed mating shroud. To use the cable:

1. Remove the cable strain relief and shave the excess plastic from the main body of the connector.
2. Insert the cable into a board and mating connector that does not have a shroud or remove the edge plastic of the shroud.
3. If using a demo board, use the alternate ARM debug Adapters and cables included (20 pin SWD trace, 20 pin JTAG/ARM)

To acquire a new cable, contact Microchip Support or purchase a Samtec FFSD-05-D-06.00-01-N or Digi-key SAM8218-ND.

Related Links

3.3.2. Adapter Boards

10.5.4 Current Sense Module

The following describes the Current Sense module used with Power Monitoring.

The MPLAB ICE 4 analog front-end for the current sense module contains two channels, referred to as Channel A and Channel B. Although they operate on similar mechanisms, the two channels are not symmetrical and are used for different purposes.

Both channels are fed into independent ADC channels of MCP3464. This ADC module allows only +ve current measurements on both the channels and at maximum sampling rate of 1kHz (1000sps).

Figure 10-4. Current Sense Connector on MPLAB ICE 4

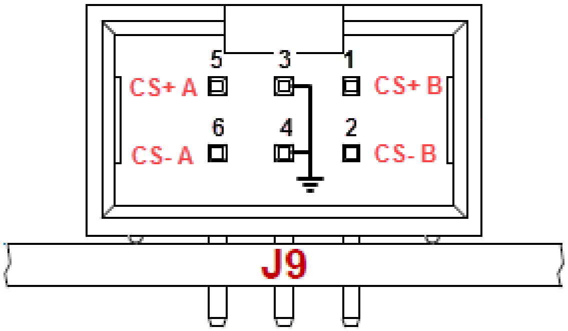


Table 10-7. Channel Specifications

Channel Current and Voltage	Resolution	Full Scale
Channel A - Current	0.6086 uA/step	20mA*
Channel A - Voltage	0.2087 mV/step	6.8V
Channel B - Current	3.043 uA/step	100mA

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

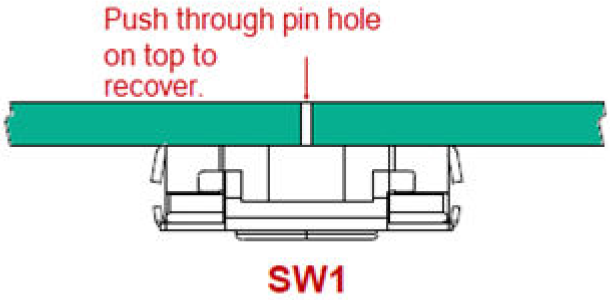
Channel Current and Voltage	Resolution	Full Scale
Channel B - Voltage	0.2087 mV/step	6.8V
* Channel A is limited to 20mA max on units with assembly #10-10203-R5. To increase the overall current range on Channel A to 1A, place an external shunt resistor (0.01 Ω) across the CS+ A and CS- A terminals. When using MPLAB Data Visualizer for range measurements, you could alternatively use channel B.		

10.6 Recovery Specifications

The MPLAB ICE 4 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 10-5. Location of Recovery Switch



10.7 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 (January 2022)

Release of first version of this document.

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.

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.

PART NO.	[X][1]	-	X	/XX	XXX
Device	Tape and Reel Option		Temperature Range	Package	Pattern

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, AnyRate, 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, IntelliMOS, Libero, motorBench, mTouch, Powermite 3, Precision Edge, ProASIC, ProASIC Plus, ProASIC Plus logo, Quiet- Wire, SmartFusion, SyncWorld, Temux, TimeCesium, TimeHub, TimePictra, TimeProvider, TrueTime, WinPath, 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, 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, Inter-Chip Connectivity, JitterBlocker, Knob-on-Display, maxCrypto, maxView, memBrain, Mindi, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, NVM Express, NVMe, 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, SQI, SuperSwitcher, SuperSwitcher II, Switchtec, SynchroPHY, Total Endurance, 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, Symmcom, and Trusted Time 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.

ISBN: 978-1-5224-9576-5

Quality Management System

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

Worldwide Sales and Service

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
www.microchip.com/support
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Austin, TX
Tel: 512-257-3370

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Novi, MI
Tel: 248-848-4000

Houston, TX
Tel: 281-894-5983

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453
Tel: 317-536-2380

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608
Tel: 951-273-7800

Raleigh, NC
Tel: 919-844-7510

New York, NY
Tel: 631-435-6000

San Jose, CA
Tel: 408-735-9110
Tel: 408-436-4270

Canada - Toronto
Tel: 905-695-1980
Fax: 905-695-2078

ASIA/PACIFIC

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

ASIA/PACIFIC

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

EUROPE

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