MPLAB® IDE
Quick Start Guide

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

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Trademarks

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

Microchip received ISO/TS-16949:2002 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

MICROCHIP

MPLAB® IDE QUICK START GUIDE

Table of Contents

NOTES:

Chapter 1. What is MPLAB® IDE?

1.1 AN OVERVIEW OF EMBEDDED SYSTEMS

MPLAB IDE is a Windows® Operating System (OS) software program that runs on a PC to develop applications for Microchip microcontrollers and digital signal controllers. It is called an Integrated Development Environment, or IDE, because it provides a single integrated "environment" to develop code for embedded microcontrollers. Experienced embedded systems designers may want to skip ahead to **Section 1.7 "Components of MPLAB IDE"**. It is also recommended that **Section 1.9 "MPLAB IDE On-line Help"** and **Section 1.10 "MPLAB IDE Updates and Version Numbering"** be reviewed. The rest of this chapter briefly explains embedded systems development and how MPLAB IDE is used.

1.1.1 Description of an "Embedded System"

An embedded system is typically a design making use of the power of a small microcontroller, like the Microchip PIC® MCU or dsPIC® Digital Signal Controller (DSCs). These microcontrollers combine a microprocessor unit (like the CPU in a desktop PC) with some additional circuits called "peripherals", plus some additional circuits on the same chip to make a small control module requiring few other external devices. This single device can then be embedded into other electronic and mechanical devices for low-cost digital control.

1.1.2 Differences Between an Embedded Controller and a PC

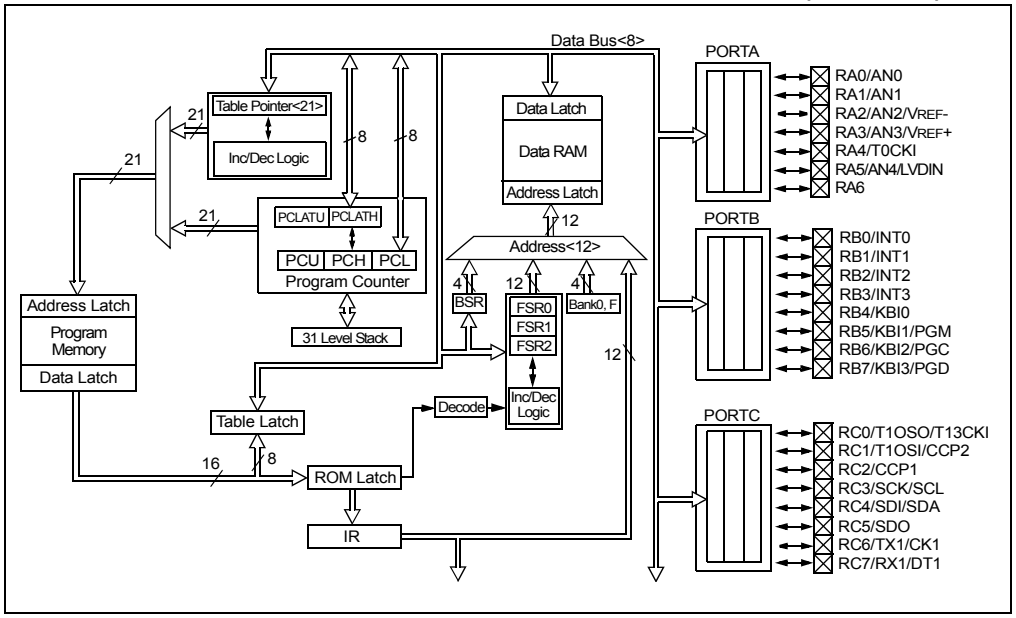
The main difference between an embedded controller and a PC is that the embedded controller is dedicated to one specific task or set of tasks. A PC is designed to run many different types of programs and to connect to many different external devices. An embedded controller has a single program and, as a result, can be made cheaply to include just enough computing power and hardware to perform that dedicated task. A PC has a relatively expensive generalized central processing unit (CPU) at its heart with many other external devices (memory, disk drives, video controllers, network interface circuits, etc.). An embedded system has a low-cost microcontroller unit (MCU) for its intelligence, with many peripheral circuits on the same chip, and with relatively few external devices. Often, an embedded system is an invisible part, or sub-module of another product, such as a cordless drill, refrigerator or garage door opener. The controller in these products does a tiny portion of the function of the whole device. The controller adds low-cost intelligence to some of the critical sub-systems in these devices.

An example of an embedded system is a smoke detector. Its function is to evaluate signals from a sensor and sound an alarm if the signals indicate the presence of smoke. A small program in the smoke detector either runs in an infinite loop, sampling the signal from the smoke sensor, or lies dormant in a low-power "sleep" mode, being awakened by a signal from the sensor. The program then sounds the alarm. The program would possibly have a few other functions, such as a user test function, and a low battery alert. While a PC with a sensor and audio output could be programmed to do the same function, it would not be a cost-effective solution (nor would it run on a nine-volt battery, unattended for years!). Embedded designs use inexpensive microcontrollers to put intelligence into the everyday things in our environment, such as smoke detectors, cameras, cell phones, appliances, automobiles, smart cards and security systems.

1.1.3 Components of a Microcontroller

The PIC MCU has program memory for the firmware, or coded instructions, to run a program. It also has “file register” memory for storage of variables that the program will need for computation or temporary storage. It also has a number of peripheral device circuits on the same chip. Some peripheral devices are called I/O ports. I/O ports are pins on the microcontroller that can be driven high or low to send signals, blink lights, drive speakers – just about anything that can be sent through a wire. Often these pins are bidirectional and can also be configured as inputs allowing the program to respond to an external switch, sensor or to communicate with some external device.

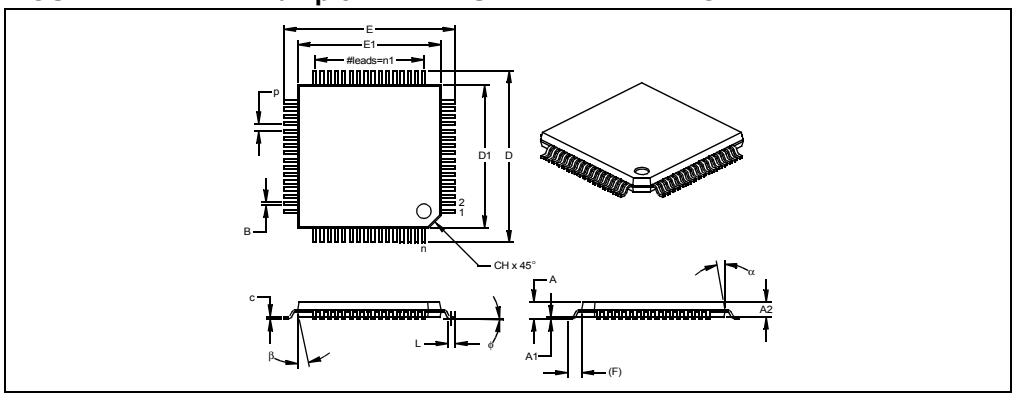
FIGURE 1-1: PIC® MCU DATA SHEET – BLOCK DIAGRAM (EXCERPT)



In order to design such a system, it must be decided which peripherals are needed for an application. Analog-to-Digital Converters (ADCs) allow microcontrollers to connect to sensors and receive changing voltage levels. Serial communication peripherals allow you to stream communications over a few wires to another microcontroller, to a local network or to the internet. Peripherals on the PIC MCU called “timers” accurately measure signal events and generate and capture communications signals, produce precise waveforms, even automatically reset the microcontroller if it gets “hung” or lost due to a power glitch or hardware malfunction. Other peripherals detect if the external power is dipping below dangerous levels so the microcontroller can store critical information and safely shut down before power is completely lost.

The peripherals and the amount of memory an application needs to run a program largely determines which PIC MCU to use. Other factors might include the power consumed by the microcontroller and its “form factor,” i.e., the size and characteristics of the physical package that must reside on the target design.

FIGURE 1-2: Example PIC® MCU DEVICE PACKAGE



1.1.4 Implementing an Embedded System Design with MPLAB IDE

A development system for embedded controllers is a system of programs running on a desktop PC to help write, edit, debug and program code – the intelligence of embedded systems applications – into a microcontroller. MPLAB IDE runs on a PC and contains all the components needed to design and deploy embedded systems applications.

The typical tasks for developing an embedded controller application are:

1. Create the high level design. From the features and performance desired, decide which PIC MCU or dsPIC DSC device is best suited to the application, then design the associated hardware circuitry. After determining which peripherals and pins control the hardware, write the firmware – the software that will control the hardware aspects of the embedded application. A language tool such as an assembler, which is directly translatable into machine code, or a compiler that allows a more natural language for creating programs, should be used to write and edit code. Assemblers and compilers help make the code understandable, allowing function labels to identify code routines with variables that have names associated with their use, and with constructs that help organize the code in a maintainable structure.

FIGURE 1-3: PIC® MCU DATA SHEET – TIMING (EXCERPT)

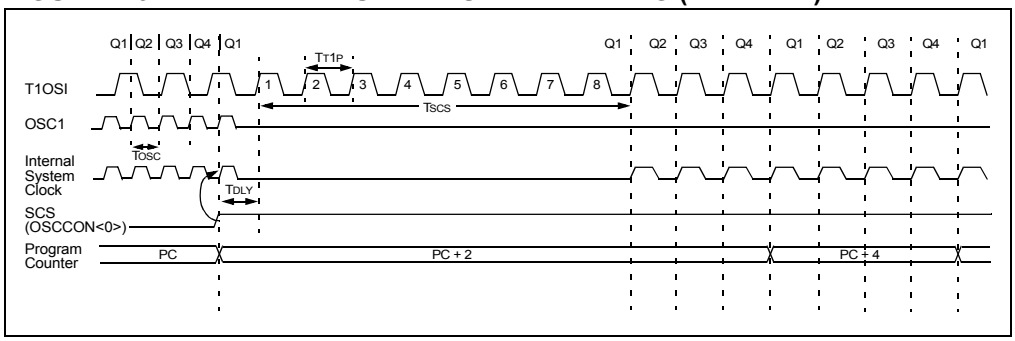
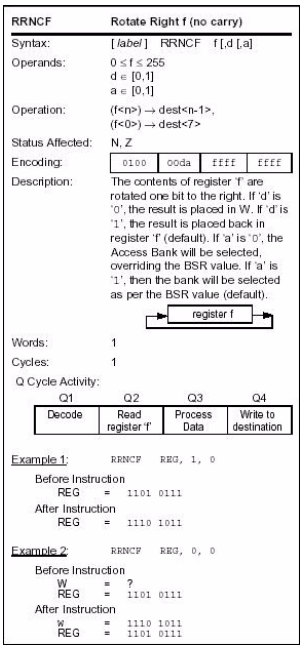


FIGURE 1-4: PIC® MCU DATA SHEET – INSTRUCTIONS (EXCERPT)



RRNCF — Rotate Right f (no carry)

Syntax: [*label*] RRNCF f [,d [,a]

Operands: $0 \le f \le 255$, $d \in [0,1]$, $a \in [0,1]$

Operation: (f<n>) → dest<n-1>, (f<0>) → dest<7>

Status Affected: N, Z

Encoding:

0100	00da	ffff	ffff

Description: The contents of register 'f' are rotated one bit to the right. If 'd' is '0', the result is placed in W. If 'd' is '1', the result is placed back in register 'f' (default). If 'a' is '0', the Access Bank will be selected, overriding the BSR value. If 'a' is '1', then the bank will be selected as per the BSR value (default).

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

Example 1: RRNCF REG, 1, 0

Before Instruction
REG = 1101 0111
After Instruction
REG = 1110 1011

Example 2: RRNCF REG, 0, 0

Before Instruction
W = ?
REG = 1101 0111
After Instruction
W = 1110 1011
REG = 1101 0111

2. Compile, assemble and link the software using the assembler and/or compiler and linker to convert your code into “ones and zeroes” – machine code for the PIC MCUs. This machine code will eventually become the firmware (the code programmed into the microcontroller).
3. Test your code. Usually a complex program does not work exactly the way imagined, and “bugs” need to be removed from the design to get proper results. The debugger allows you to see the “ones and zeroes” execute, related to the source code you wrote, with the symbols and function names from your program. Debugging allows you to experiment with your code to see the value of variables at various points in the program, and to do “what if” checks, changing variable values and stepping through routines.
4. “Burn” the code into a microcontroller and verify that it executes correctly in the finished application.

Of course, each of these steps can be quite complex. The important thing is to concentrate on the details of your own design, while relying upon MPLAB IDE and its components to get through each step without continuously encountering new learning curves.

Step 1 is driven by the designer, although MPLAB IDE can help in modeling circuits and code so that crucial design decisions can be made.

MPLAB IDE really helps with steps 2 through 4. Its Programmer's Editor helps write correct code with the language tools of choice. The editor is aware of the assembler and compiler programming constructs and automatically "color-keys" the source code to help ensure it is syntactically correct. The Project Manager enables you to organize the various files used in your application: source files, processor description header files and library files. When the code is built, you can control how rigorously code will be optimized for size or speed by the compiler and where individual variables and program data will be programmed into the device. You can also specify a "memory model" in order to make the best use of the microcontroller's memory for your application. If the language tools run into errors when building the application, the offending line is shown and can be "double clicked" to go to the corresponding source file for immediate editing. After editing, press the "build" button to try again. Often this write-compile-fix loop is done many times for complex code as the sub-sections are written and tested. MPLAB IDE goes through this loop with maximum speed, allowing you to get on to the next step.

Once the code builds with no errors, it needs to be tested. MPLAB IDE has components called "debuggers" and free software simulators for PIC MCU and dsPIC DSC devices to help test the code. Even if the hardware is not yet finished, you can begin testing the code with the simulator, a software program that simulates the execution of the microcontroller. The simulator can accept a simulated input (stimulus), in order to model how the firmware responds to external signals. The simulator can measure code execution time, single step through code to watch variables and peripherals, and trace the code to generate a detailed record of how the program ran.

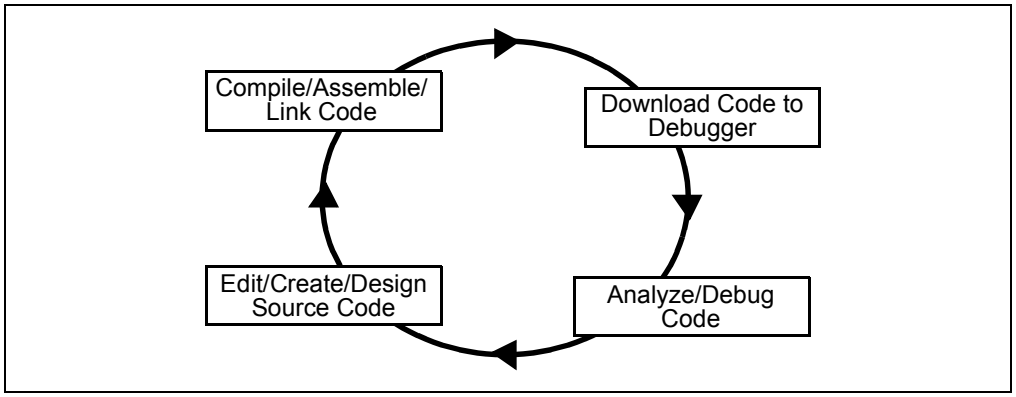
Once the hardware is in a prototype stage, a hardware debugger, such as MPLAB ICE 2000 in-circuit emulator, MPLAB REAL ICE™ in-circuit emulator, or MPLAB ICD 2 in-circuit debugger can be used. These debug tools run the code in real time on your actual application. The MPLAB ICE 2000 emulator physically replaces the microcontroller in the target using a high-speed probe to give you full control over the hardware in your design. The MPLAB REAL ICE in-circuit emulator and MPLAB ICD 2 debugger use special circuitry built into many Microchip MCUs with Flash program memory and can "see into" the target microcontrollers program and data memory. These debuggers can stop and start program execution, allowing you to test the code with the microcontroller in place on the application.

After the application is running correctly, you can program a microcontroller with one of Microchip's device programmers, such as PICSTART® Plus or MPLAB PM3. These programmers verify that the finished code will run as designed. MPLAB IDE supports PIC MCUs and dsPIC Digital Signal Controllers.

1.2 THE DEVELOPMENT CYCLE

The process for writing an application is often described as a development cycle, since it is rare that all the steps from design to implementation can be done flawlessly the first time. More often code is written, tested and then modified in order to produce an application that performs correctly. The Integrated Development Environment allows the embedded systems design engineer to progress through this cycle without the distraction of switching among an array of tools. By using MPLAB IDE, all the functions are integrated, allowing the engineer to concentrate on completing the application without the interruption of separate tools and different modes of operation.

FIGURE 1-5: THE DESIGN CYCLE

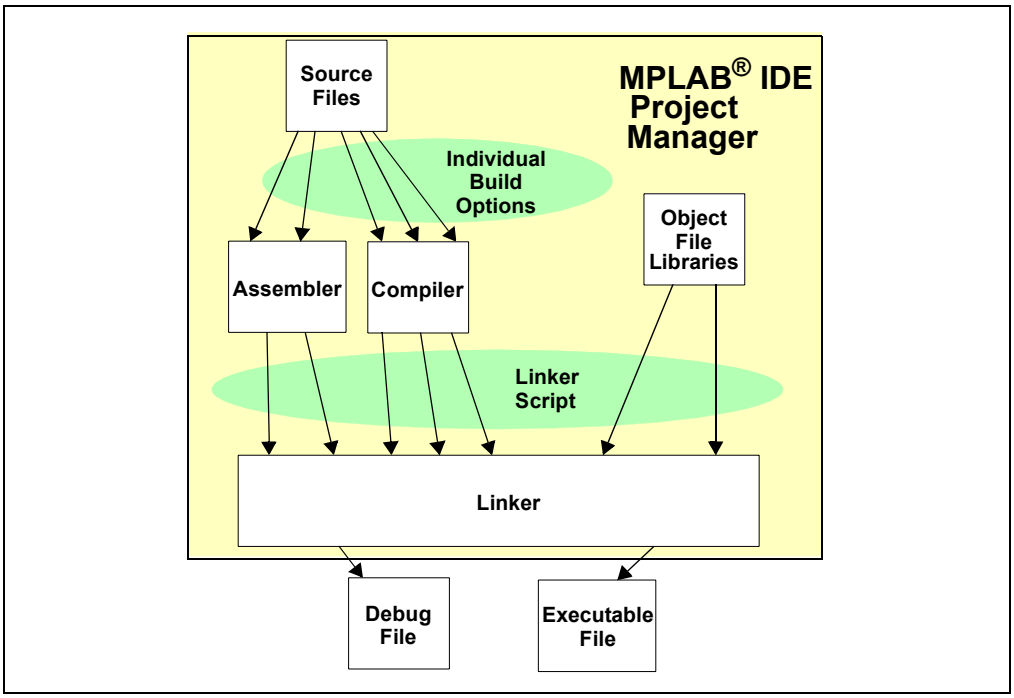


MPLAB IDE is a “wrapper” that coordinates all the tools from a single graphical user interface, usually automatically. For instance, once code is written, it can be converted to executable instructions and downloaded into a microcontroller to see how it works. In this process multiple tools are needed: an editor to write the code, a project manager to organize files and settings, a compiler or assembler to convert the source code to machine code and some sort of hardware or software that either connects to a target microcontroller or simulates the operation of a microcontroller.

1.3 PROJECT MANAGER

The project manager organizes the files to be edited and other associated files so they can be sent to the language tools for assembly or compilation, and ultimately to a linker. The linker has the task of placing the object code fragments from the assembler, compiler and libraries into the proper memory areas of the embedded controller, and ensure that the modules function with each other (or are “linked”). This entire operation from assembly and compilation through the link process is called a project “build”. From the MPLAB IDE project manager, properties of the language tools can be invoked differently for each file, if desired, and a build process integrates all of the language tools operations.

FIGURE 1-6: MPLAB® IDE PROJECT MANAGER



The source files are text files that are written conforming to the rules of the assembler or compiler. The assembler and compiler convert them into intermediate modules of machine code and placeholders for references to functions and data storage. The linker resolves these placeholders and combines all the modules into a file of executable machine code. The linker also produces a debug file which allows MPLAB IDE to relate the executing machine codes back to the source files.

A text editor is used to write the code. It is not a normal text editor, but an editor specifically designed for writing code for Microchip MCUs. It recognizes the constructs in the text and uses color coding to identify various elements, such as instruction mnemonics, C language constructs and comments. The editor supports operations commonly used in writing source code, such as finding matching braces in C, commenting and uncommenting out blocks of code, finding text in multiple files and adding special bookmarks. After the code is written, the editor works with the other tools to display code execution in the debugger. Breakpoints can be set in the editor, and the values of variables can be inspected by hovering the mouse pointer over the variable name. Names of variables can be dragged from source text windows and then dropped into a Watch window.

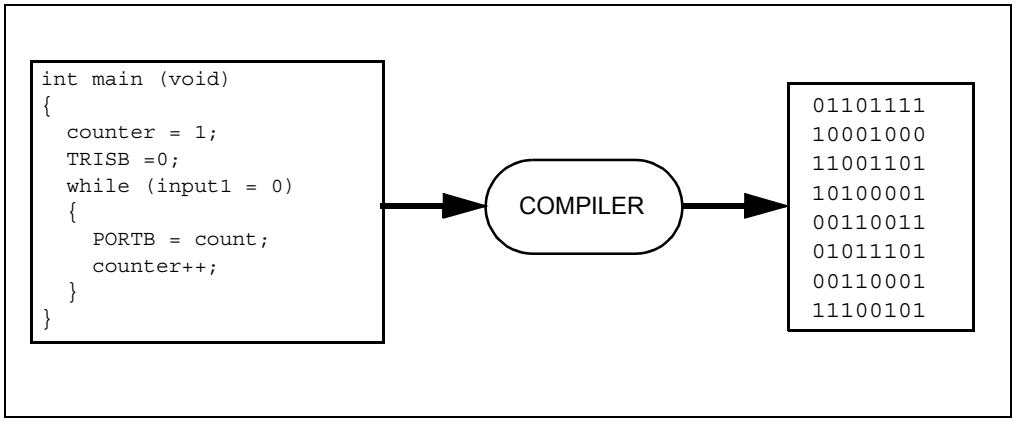
1.4 LANGUAGE TOOLS

Language tools are programs such as cross-assemblers and cross-compilers. Most people are familiar with language tools that run on a PC such as Visual Basic or C compilers. When using language tools for embedded systems, a “cross-assembler” or “cross-compiler” is used. These tools differ from typical compilers in that they run on a PC but produce code to run on another microprocessor, hence they “cross-compile” code for a microcontroller that uses an entirely different set of instructions from the PC.

The language tools also produce a debug file that MPLAB IDE uses to correlate the machine instructions and memory locations with the source code. This bit of integration allows the MPLAB IDE editor to set breakpoints, allows watch windows to view variable contents, and lets you single step through the source code, watching the application execute.

Embedded system language tools also differ somewhat for compilers that run and execute on a PC because they must be very space conscious. The smaller the code produced, the better, because that allows the smallest possible memory for the target, which reduces cost. This means that techniques to optimize and enhance the code using machine specific knowledge are desirable. The size of programs for PCs typically extends into the megabytes for moderately complex programs. The size of simple embedded systems programs may be as small as a thousand bytes or less. A medium size embedded system might need 32K or 64K of code for relatively complex functions. Some embedded systems use megabytes of storage for large tables, user text messages or data logging.

FIGURE 1-7: A COMPILER CONVERTS SOURCE CODE INTO MACHINE INSTRUCTIONS



1.5 TARGET DEBUGGING

In a development environment, the execution of the code is tested on a debugger. The debugger can be a software program that simulates the operation of the microcontroller for testing, or it can be special instrumentation to analyze the program as it executes in the application.

Simulators are built into MPLAB IDE so a program can be tested without any additional hardware. A simulator is a software debugger, and the debugger functions for the simulator are almost identical to the hardware debuggers, allowing a new tool to be learned with ease. Usually a simulator runs somewhat slower than an actual microcontroller, since the CPU in the PC is being used to simulate the operations of the microcontroller. In the case of MPLAB IDE, there are many simulators for each of the PIC MCU and the dsPIC DSC processors.

There are two types of hardware that can be used with MPLAB IDE: programmers and hardware debuggers. A programmer simply burns the machine code from the PC into the internal memory of the target microcontroller. The microcontroller can then be plugged into the application and, hopefully, it will run as designed.

Usually, however, the code does not function exactly as anticipated, and the engineer is tasked with reviewing the code and its operation in the application to determine how to modify the original source code to make it execute as desired. This process is called debugging. As noted previously, the simulator can be used to test how the code will operate, but once a microcontroller is programmed with the firmware, many things outside the scope of the simulator come into play. Using just a programmer, the code could be changed, reprogrammed into the microcontroller and plugged into the target for retest, but this could be a long, laborious cycle if the code is complex, and it is difficult to understand exactly what is going wrong in the hardware.

This is where a hardware debugger is useful. Hardware debuggers can be in-circuit emulators, which use specialized hardware in place of the actual target microcontroller, or they can be in-circuit debuggers, which use microcontrollers that have special built-in debugging features. A hardware debugger, like a simulator, allows the engineer to inspect variables at various points in the code, and single step to follow instructions as the hardware interacts with its specialized circuitry.

Debugging usually becomes urgent near the end of the project design cycle. As deadlines loom, getting the application to function as originally designed is the last step before going into deployment of the product, and often has the most influence on producing delays in getting a product out. That's where an integrated development environment is most important. Doing fine “tweaks” to the code, recompiling, downloading and testing all require time. Using all tools within a single environment will reduce the time around the “cycle.” These last steps, where critical bugs are worked out, are a test for the embedded systems designer. The right tool can save time. With MPLAB IDE many tools can be selected, but they all will have a similar interface, and the learning curve from simulator to low-cost in-circuit debugger to powerful in-circuit emulator is small.

1.6 DEVICE PROGRAMMING

After the application has been debugged and is running in the development environment, it needs to be tested on its own. A device can be programmed with the in-circuit debugger or a device programmer. MPLAB IDE can be set to the programmer function, and the part can be “burned”. The target application can now be observed in its nearly final state. Engineering prototype programmers allow quick prototypes to be made and evaluated. Some applications can be programmed after the device is soldered on the target PC board. Using In-Circuit Serial Programming™ (ICSP™) programming capability, the firmware can be programmed into the application at the time of manufacture, allowing updated revisions to be programmed into an embedded application later in its life cycle. Devices that support in-circuit debugging can even be plugged back into the MPLAB ICD 2 after manufacturing for quality tests and development of next generation firmware.

1.7 COMPONENTS OF MPLAB IDE

The MPLAB IDE has both built-in components and plug-in modules to configure the system for a variety of software and hardware tools.

1.7.1 MPLAB IDE Built-In Components

The built-in components consist of:

- **Project Manager**

The project manager provides integration and communication between the IDE and the language tools.

- **Editor**

The editor is a full-featured programmer's text editor that also serves as a window into the debugger.

- **Assembler/Linker and Language Tools**

The assembler can be used stand-alone to assemble a single file, or can be used with the linker to build a project from separate source files, libraries and recompiled objects. The linker is responsible for positioning the compiled code into memory areas of the target microcontroller.

- **Debugger**

The Microchip debugger allows breakpoints, single stepping, watch windows and all the features of a modern debugger for the MPLAB IDE. It works in conjunction with the editor to reference information from the target being debugged back to the source code.

- **Execution Engines**

There are software simulators in MPLAB IDE for all PIC MCU and dsPIC DSC devices. These simulators use the PC to simulate the instructions and some peripheral functions of the PIC MCU and dsPIC DSC devices. Optional in-circuit emulators and in-circuit debuggers are also available to test code as it runs in the applications hardware.

1.7.2 Additional Optional Components for MPLAB IDE

Optional components can be purchased and added to the MPLAB IDE:

- **Compiler Language Tools**

MPLAB C18 and MPLAB C30 C compilers from Microchip provide fully integrated, optimized code. Along with compilers from HI-TECH, IAR, microEngineering Labs, CCS and Byte Craft, they are invoked by the MPLAB IDE project manager to compile code that is automatically loaded into the target debugger for instant testing and verification.

- **Programmers**

MPLAB PM3, PICSTART® Plus, PICkit™ 1 and 2, as well as the MPLAB ICD 2 debugger and MPLAB REAL ICE in-circuit emulator can program code into target devices. MPLAB IDE offers full control over programming both code and data, as well as the Configuration bits to set the various operating modes of the target microcontrollers or digital signal controllers.

- **In-Circuit Emulators**

The MPLAB REAL ICE and MPLAB ICE 2000 systems are in-circuit emulators for PIC MCU and dsPIC DSC devices. They connect to the PC via I/O ports and allow full control over the operation of microcontroller in the target applications.

- **In-Circuit Debugger**

MPLAB ICD 2 and PICkit 2 provide economic alternatives to an emulator. By using some of the on-chip resources, MPLAB ICD 2 can download code into a target microcontroller inserted in the application, set breakpoints, single step and monitor registers and variables.

1.8 MPLAB IDE DOCUMENTATION

The following documents are available to help you use MPLAB IDE:

- *"MPLAB® IDE Quick Chart"* (DS51410)
- *"MPLAB® IDE Quick Start Guide"* (DS51281) – Chapters 1 and 2 of user's guide.
- *"MPLAB® IDE User's Guide"* (DS51519)

Other documents exist for various Microchip software and hardware tools that work with MPLAB IDE. Check the Microchip web site for downloadable PDF versions of all these documents.

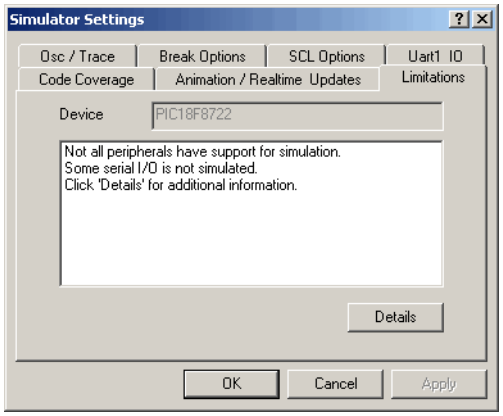
1.9 MPLAB IDE ON-LINE HELP

Since MPLAB IDE is under a constant state of change (see **Section 1.10 "MPLAB IDE Updates and Version Numbering"**) some details in this documentation may change. Dialogs might not appear exactly as they do in this manual, menu lists may be in different order, or may have new items. For this reason, the on-line help is the best reference to the version of MPLAB IDE being used.

MPLAB IDE comes with extensive on-line help, which is constantly being updated. If questions arise while using MPLAB IDE, be sure to check the on-line help for answers. Most importantly, the on-line help lists any restrictions that might exist for a particular tool in support of a particular device. Always try to review this section before working with a new device/tool combination.

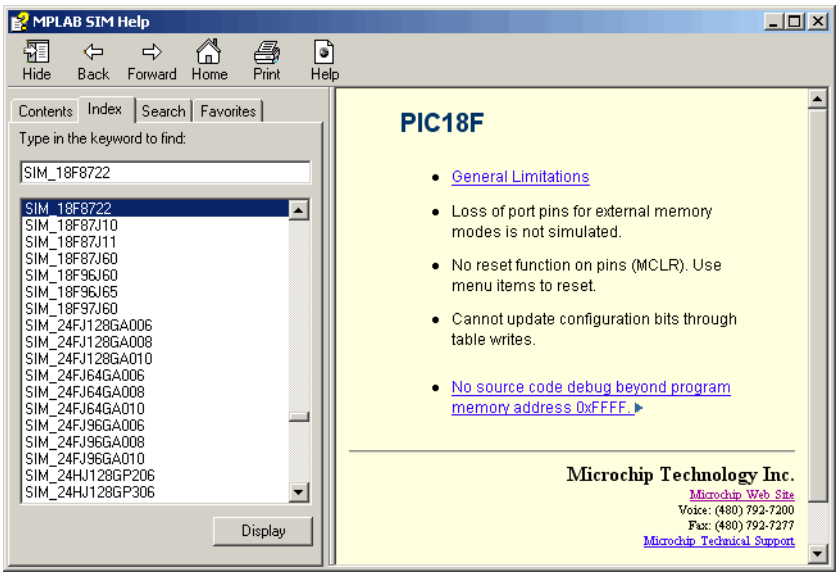
The Limitations tab in the *Debugger>Settings* dialog displays any restrictions the simulator, emulator or in-circuit debugger might have, compared to the actual device being simulated. General limitations are shown in the text area.

FIGURE 1-8: *DEBUGGER>SETTINGS*, LIMITATIONS TAB



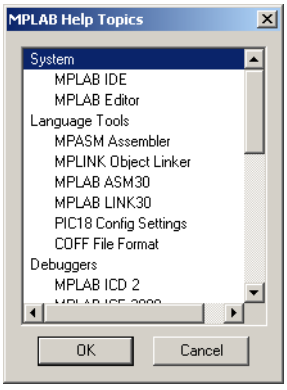
Press the **Details** button to show specific limitations of the device being debugged. From this display, help on general limitations related to the debugger can also be accessed.

FIGURE 1-9: SIMULATOR LIMITATIONS DETAIL



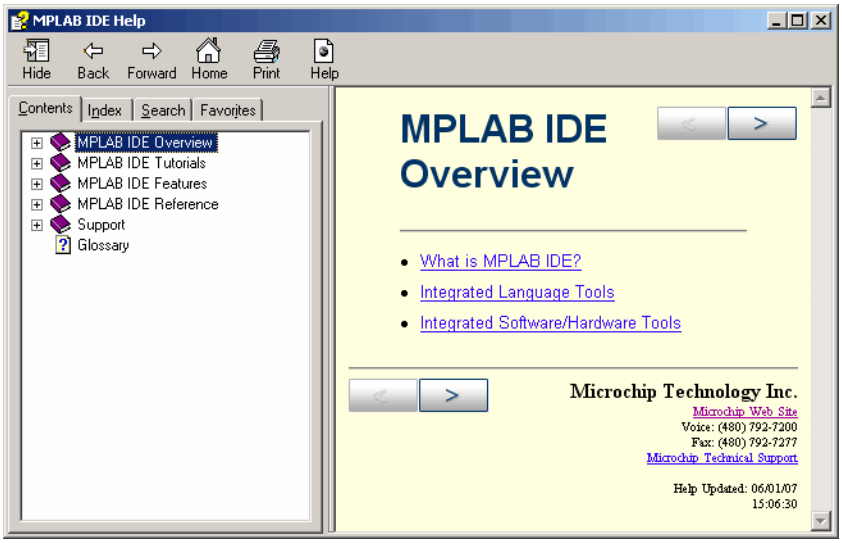
From the main MPLAB IDE Help menu, select *Help>Topics* to get a list of help on MPLAB IDE and all of its components.

FIGURE 1-10: MPLAB® IDE *HELP>TOPICS* MENU



MPLAB IDE Help covers all aspects of MPLAB IDE and all of the Microchip tools. It can be viewed in an outline, as an index and with a search utility for help on any MPLAB IDE topic. It also directs users to other types of assistance, such as the Microchip Update Notification system.

FIGURE 1-11: MPLAB® IDE HELP DIALOG



1.10 MPLAB IDE UPDATES AND VERSION NUMBERING

MPLAB IDE is an evolving program with thousands of users. Microchip Technology is continually designing new microcontrollers with new features. Many new MPLAB IDE features come from customer requests and from internal usage. Continued new designs and the release of new microcontrollers ensure that MPLAB IDE will continue to evolve.

MPLAB IDE is scheduled for a version update approximately every four months to add new device support and new features. Additional “interim” releases are produced in between these major releases. The version numbering scheme for MPLAB IDE reflects whether it is a major production release or an interim release. If the version number ends in a zero, e.g., MPLAB IDE v7.00, v7.10 or v7.20, this identifies a major production release. If the version number ends with a digit other than zero, e.g., v7.11, v7.22 or v7.55, this identifies an interim release. Interim releases are typically provided for early adopters of new devices or components, for quick critical fixes or for previews of new features. These interim releases, although based on production releases that are fully tested, may have components that are not as fully tested, and therefore these releases are not recommended for rigorous design use.

It is advised that production releases be used for development work, unless a new device or component is being used or a particular problem has been fixed in an interim release that enables more productive use of MPLAB IDE. Also, for projects that are midway through development when a new version of MPLAB IDE is released, it is considered “best practice” to not update to the new release unless there is a compelling reason to do so, such as a bug fix on a bug that inhibits the current efforts. The start of a new project is the best time to update to a new release.

Each new release of the MPLAB IDE software has new features implemented, so the printed documentation will inevitably “lag” the on-line help. The on-line help is the best source for any questions about MPLAB IDE.

To be notified of updates to MPLAB IDE and its components, subscribe to the Development Tools section of the Customer Change Notification service on www.microchip.com.

Chapter 2. A Basic Tutorial for MPLAB IDE

2.1 INTRODUCTION

The MPLAB Integrated Development Environment (IDE) is a comprehensive editor, project manager and design desktop for application development of embedded designs using Microchip PIC MCUs and dsPIC DSCs.

The initial use of MPLAB IDE is covered here. How to make projects, edit code and test an application will be the subject of a short tutorial. By going through the tutorial, the basic concepts of the Project Manager, Editor and Debugger can be quickly learned. The complete feature set of MPLAB IDE is covered in later chapters.

This section details the installation and uninstall of MPLAB IDE. It is followed by a simple step-by-step tutorial that creates a project and explains the elementary debug capabilities of MPLAB IDE. Someone unfamiliar with MPLAB IDE will get a basic understanding of using the system to develop an application. No previous knowledge is assumed, and comprehensive technical details of MPLAB IDE and its components are omitted in order to present the basic framework for using MPLAB IDE.

These basic steps will be covered in the tutorial:

- MPLAB IDE Features and Installation
- Tutorial Overview
- Selecting the Device
- Creating the Project
- Setting Up Language Tools
- Naming the Project
- Adding Files to the Project
- Building the Project
- Creating Code
- Building the Project Again
- Testing Code with the Simulator
- Tutorial Summary

2.2 MPLAB IDE FEATURES AND INSTALLATION

MPLAB IDE is a Windows® Operating System (OS) based Integrated Development Environment for the PIC MCU families and the dsPIC Digital Signal Controllers. The MPLAB IDE provides the ability to:

- Create and edit source code using the built-in editor.
- Assemble, compile and link source code.
- Debug the executable logic by watching program flow with the built-in simulator or in real time with in-circuit emulators or in-circuit debuggers.
- Make timing measurements with the simulator or emulator.
- View variables in Watch windows.
- Program firmware into devices with device programmers (for details, consult the user's guide for the specific device programmer).

Note: Selected third party tools are also supported by MPLAB IDE. Check the release notes or readme files for details.

2.2.1 Install/Uninstall MPLAB IDE

To install MPLAB IDE on your system:

Note: For some Windows OSs, administrative access is required in order to install software on a PC.

- If installing from a CD-ROM, place the disk into a CD drive. Follow the on-screen menu to install MPLAB IDE. If no on-screen menu appears, use Windows Explorer to find and execute the CD-ROM menu, `menu.exe`.
- If downloading MPLAB IDE from the Microchip web site (www.microchip.com), locate the download (`.zip`) file, select the file and save it to the PC. Unzip the file and execute the resulting file to install.

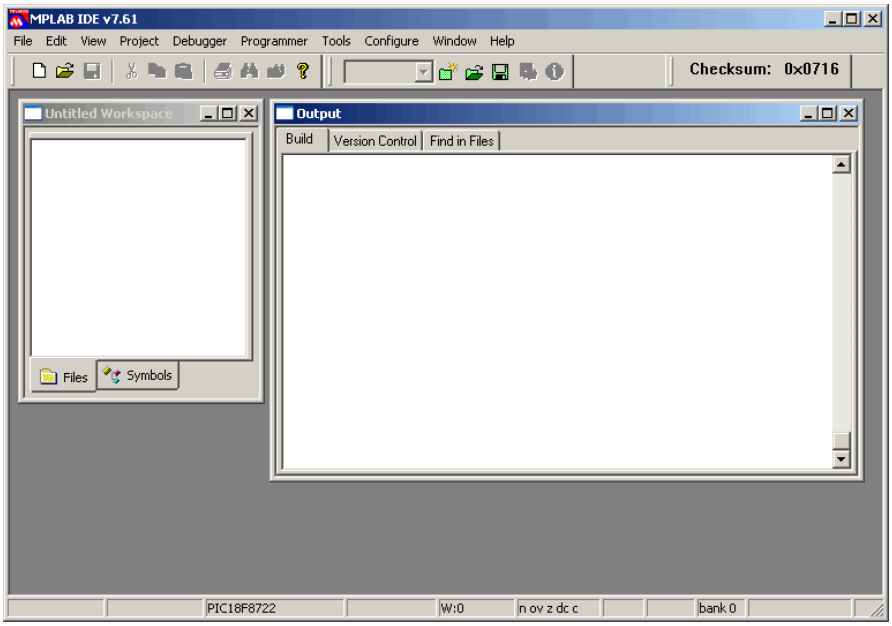
To uninstall MPLAB IDE:

- Select *Start>Settings>Control Panel* to open the Control Panel.
- Double click on Add/Remove Programs. Find MPLAB IDE on the list and click on it.
- Click **Change/Remove** to remove the program from your system.

2.2.2 Running MPLAB IDE

To start MPLAB IDE, double click on the icon installed on the desktop after installation or select *Start>Programs>Microchip>MPLAB IDE vx.xx>MPLAB IDE*. A screen will display the MPLAB IDE logo followed by the MPLAB IDE desktop (Figure 2-1).

FIGURE 2-1: MPLAB® IDE DESKTOP



2.3 TUTORIAL OVERVIEW

In order to create code that is executable by the target PIC MCU, source files need to be put into a project. The code can then be built into executable code using selected language tools (assemblers, compilers, linkers, etc.). In MPLAB IDE, the project manager controls this process.

All projects will have these basic steps:

- Select Device

  The capabilities of MPLAB IDE vary according to which device is selected. Device selection should be completed before starting a project.
- Create Project

  MPLAB IDE Project Wizard will be used to Create a Project.
- Select Language Tools

  In the Project Wizard the language tools will be selected. For this tutorial, the built-in assembler and linker will be used. For other projects, one of the Microchip compilers or other third party tools might be selected.
- Put Files in Project

  Two files will be put into the project, a template file and a linker script. Both of these files exist in sub-folders within the MPLAB IDE folder. It is easy to get started using these two files.
- Create Code

  Some code will be added to the template file to send an incrementing value out an I/O port.
- Build Project

  The project will be built – causing the source files to be assembled and linked into machine code that can run on the selected PIC MCU.
- Test Code with Simulator

  Finally, the code will be tested with the simulator.

The Project Wizard will easily guide us through most of these steps.

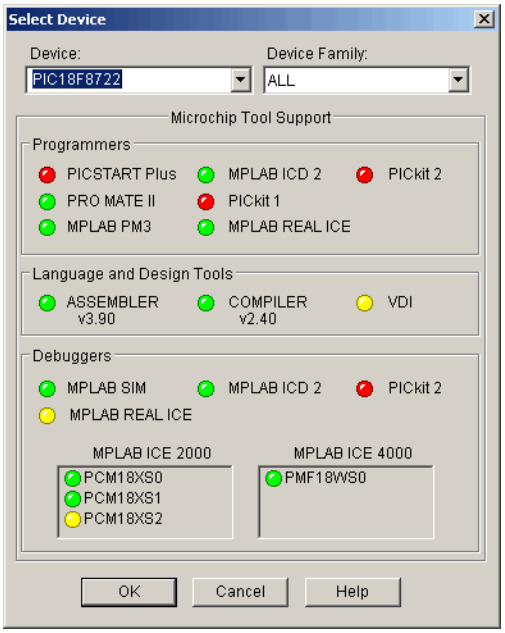
Note: Some aspects of the user interface will change in future product releases and the screen shots in this tutorial may not exactly match the appearance of the MPLAB IDE desktop in later releases. New features will be added as additional parts are released. None of the functions described in this tutorial will be removed, but more features may be added.The on-line help is the most up-to-date reference for the current version of MPLAB IDE.

2.4 SELECTING THE DEVICE

To show menu selections in this document, the menu item from the top row in MPLAB IDE will be shown after the menu name like this *MenuName>MenuItem*. To choose the *Select Device* entry in the *Configure* menu, it would be written as *Configure>Select Device*.

Choose *Configure>Select Device*. In the Device dialog, select the **PIC18F8722** from the list if it's not already selected.

FIGURE 2-2: SELECT DEVICE DIALOG



The "lights" indicate which MPLAB IDE components support this device.

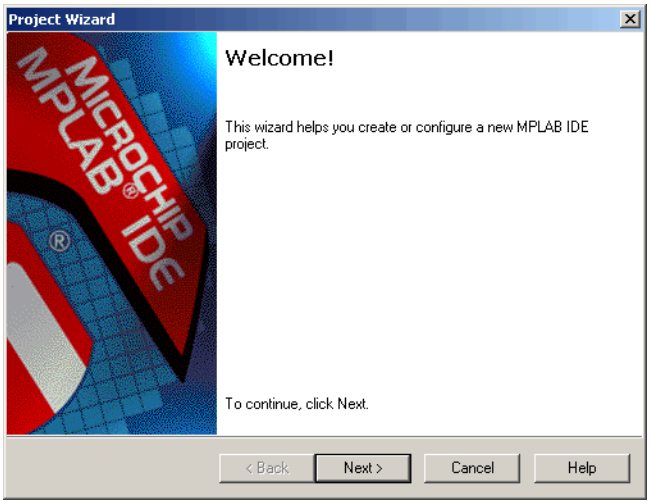
- A green light indicates full support.
- A yellow light indicates preliminary support for an upcoming part by the particular MPLAB IDE tool component. Components with a yellow light instead of a green light are often intended for early adopters of new parts who need quick support and understand that some operations or functions may not be available.
- A red light indicates no support for this device. Support may be forthcoming or inappropriate for the tool, e.g., dsPIC DSC devices cannot be supported on MPLAB ICE 2000.

2.5 CREATING THE PROJECT

The next step is to create a project using the Project Wizard. A project is the way the files are organized to be compiled and assembled. We will use a single assembly file for this project and a linker script. Choose *Project>Project Wizard*.

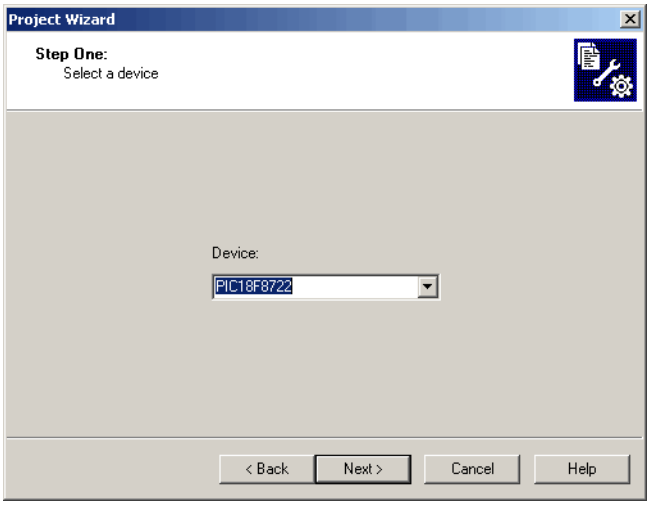
From the Welcome dialog, click on **Next>** to advance.

FIGURE 2-3: PROJECT WIZARD WELCOME



The next dialog (Step One) allows you to select the device, which we've already done. Make sure that it says PIC18F8722. If it does not, select the PIC18F8722 from the drop down menu. Click **Next>**.

FIGURE 2-4: PROJECT WIZARD – SELECT DEVICE



2.6 SETTING UP LANGUAGE TOOLS

Step Two of the Project Wizard sets up the language tools that are used with this project. Select “Microchip MPASM Toolsuite” in the Active Toolsuite list box. Then “MPASM” and “MPLINK” should be visible in the Toolsuite Contents box. Click on each one to see its location. If MPLAB IDE was installed into the default directory, the MPASM™ assembler executable will be:

```
C:\Program Files\Microchip\MPASM Suite\mpasmwin.exe
```

the MPLINK™ linker executable will be:

```
C:\Program Files\Microchip\MPASM Suite\_mplink.exe
```

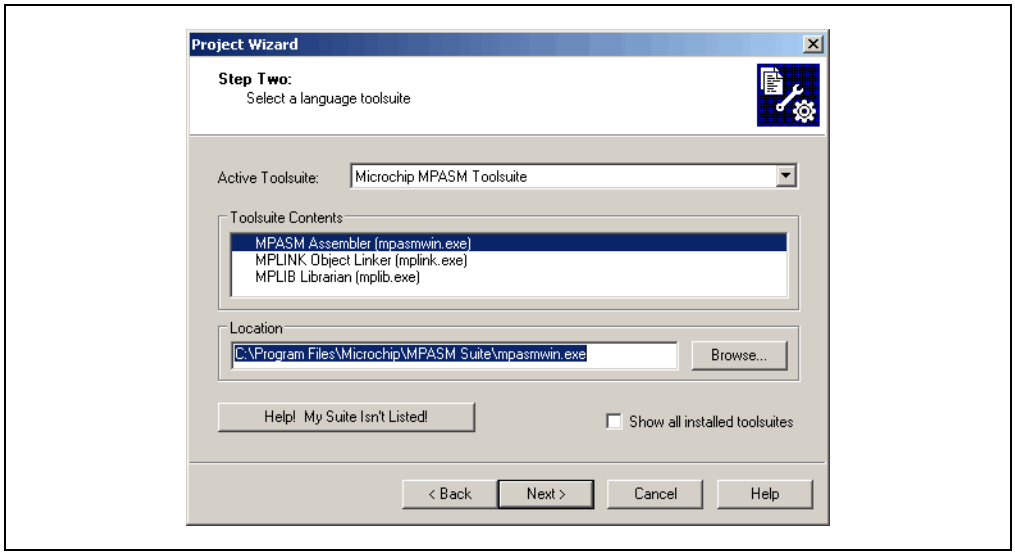
Note: There is also a wrapper executable, `mplink.exe`, which calls `_mplink.exe` (linker), `mp2cod.exe` (output conversion to COD file) and `mp2hex.exe` (output conversion to Hex file). By default, MPLAB IDE will call the linker and Hex converter seperately, with no COD file generation.

and the MPLIB™ librarian executable will be:

```
C:\Program Files\Microchip\MPASM Suite\mplib.exe
```

If these do not show up correctly, use the browse button to set them to the proper files in the MPLAB IDE subfolders.

FIGURE 2-5: PROJECT WIZARD – SELECT LANGUAGE TOOLS

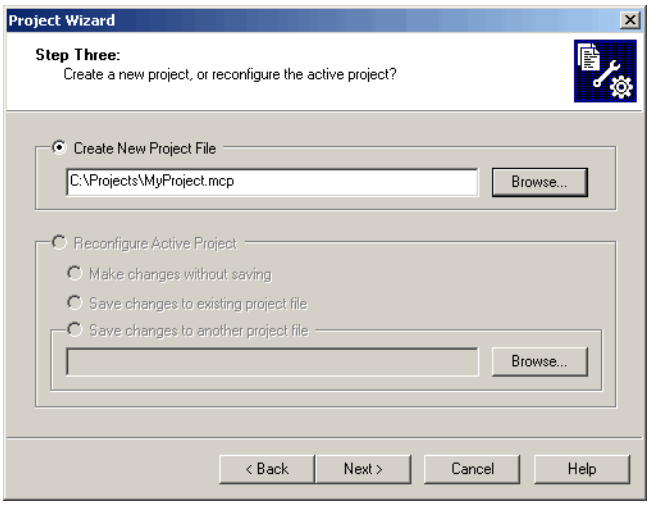


When you are finished, click **Next>**.

2.7 NAMING THE PROJECT

Step Three of the wizard allows you to name the new project and put it into a folder. This sample project will be called `C:\Projects\MyProject.`. Click **Next>**.

FIGURE 2-6: PROJECT WIZARD – NAME PROJECT



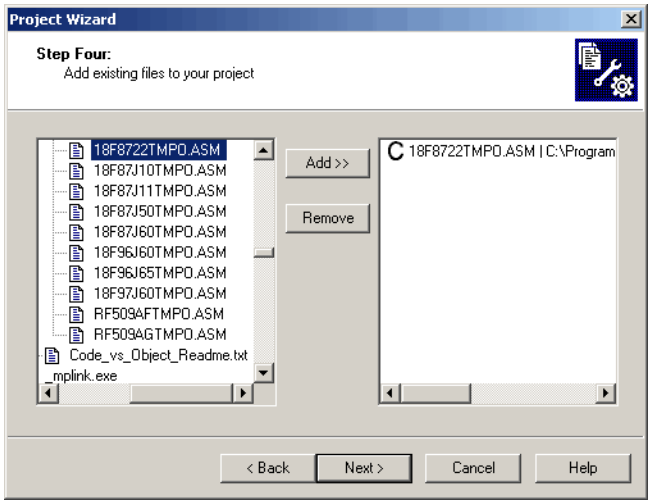
2.8 ADDING FILES TO THE PROJECT

Step Four of the Project Wizard allows file selection for the project. A source file has not yet been selected, so we will use an MPLAB IDE template file. The template files are simple files that can be used to start a project. They have the essential sections for any source file, and contain information that will help you write and organize your code.

There are two template files for each Microchip PIC MCU and dsPIC DSC device: one for absolute code (no linker used) in the `Code` directory and one for relocatable code (linker used) in the `Object` directory. Since we will be using the linker in this tutorial, choose the file named `18F8722.` in the `Object` directory. If MPLAB IDE is installed in the default location, the full path to the file will be:

```
C:\Program Files\Microchip\MPASM Suite\Template\Object
  \18F8722.
```

FIGURE 2-7: PROJECT WIZARD – SELECT TEMPLATE FILE



Press **Add>>** to move the file name to the right panel. Click on the “A“ at the start of the line with the file name three times until a “C” appears. This will enable this file to be copied to our project directory.

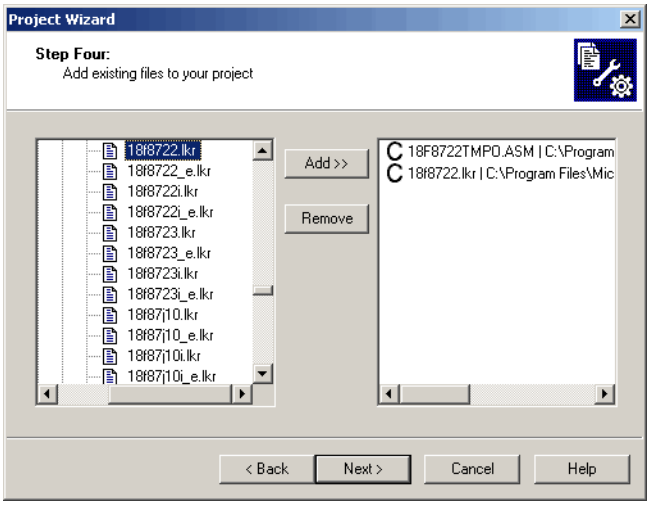
Next, add the second file for our project, the linker script. There is a linker script for each device. These files define the memory configuration and register names for the various parts. Use the file named `18f8722.lkr`. The full path is:

```
C:\Program Files\Microchip\MPASM Suite\LKR\18f8722.lkr
```

Note: There is another linker script named `18f8722i.lkr`, for use with this device when MPLAB ICD 2 is being used (hence the “`i`” in the name). That linker script reserves areas in memory for MPLAB ICD 2. Since the simulator is being used, we don’t need to use that linker script.
There is also a linker script named `18f8722_e.lkr` for use with the device extended instruction set. See the device data sheet for details. The extended instruction set will not be used in this tutorial.

To copy this linker script into our project, click on the “A” three times until a “C” appears.

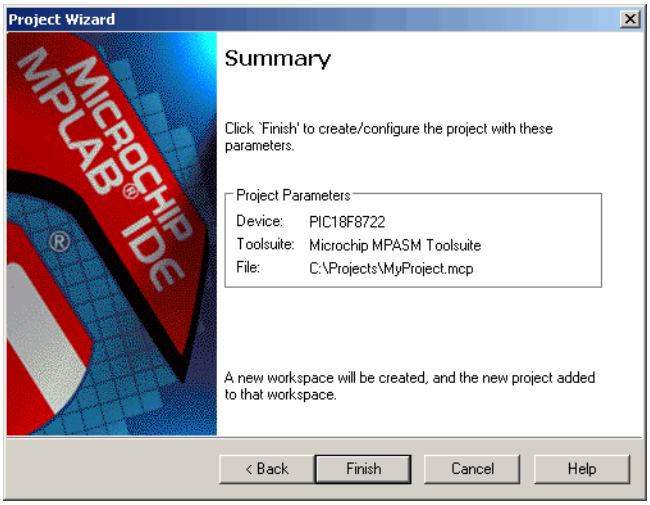
FIGURE 2-8: PROJECT WIZARD – SELECT LINKER SCRIPT



Make sure that your dialog looks like the picture above, and then press **Next>** to finish the Project Wizard.

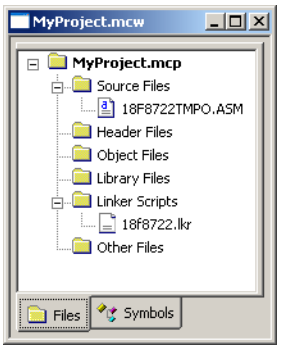
The final screen of the Project Wizard is a summary showing the selected device, the toolsuite and the new project file name.

FIGURE 2-9: PROJECT WIZARD – SUMMARY



After pressing the **Finish** button, review the Project Window on the MPLAB IDE desktop. It should look like Figure 2-10. If the Project Window is not open, select *View>Project*.

FIGURE 2-10: PROJECT WINDOW



TIP: Files can be added and projects saved by using the right mouse button in the project window. In case of error, files can be manually deleted by selecting them and using the right mouse click menu.

2.9 BUILDING THE PROJECT

From the Project menu, we can assemble and link the current files. They don't have any of our code in them yet, but this ensures that the project is set up correctly.

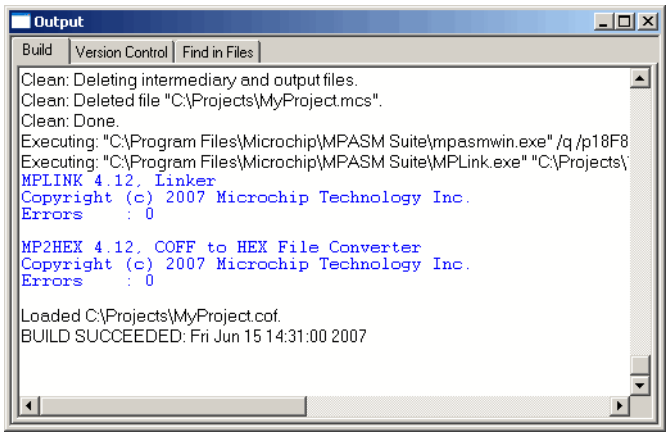
To build the project, select either:

- *Project>Build All*
- Right click on the project name in the project window and select Build All
- Click the Build All icon on the Project toolbar. Hover the mouse over icons to see pop-up text of what they represent.

The Output window shows the result of the build process. There should be no errors or warnings on any step. However, if you do receive errors, go back to the previous sections and check the project assembly steps. Errors will prevent the project from building. If you receive warnings, you may ignore them for this project as they will not prevent the project from building. To turn off the display of warnings, do the following:

- Select *Project>Build Options>Project* and click on the **MPASM Assembler** tab.
- Select "Output" from the "Categories" drop-down list.
- Select "Errors only" from the "Diagnostic level" drop-down list.
- Click **OK**.

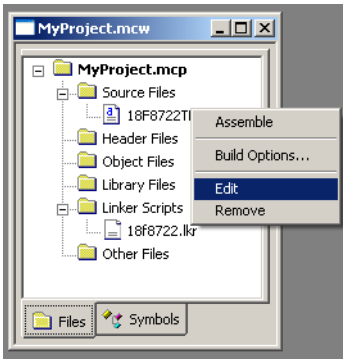
FIGURE 2-11: OUTPUT WINDOW



2.10 CREATING CODE

Open the template file in the project by double clicking on its name in the Project Window, or by selecting it with the cursor and using the right mouse button to bring up the context menu:

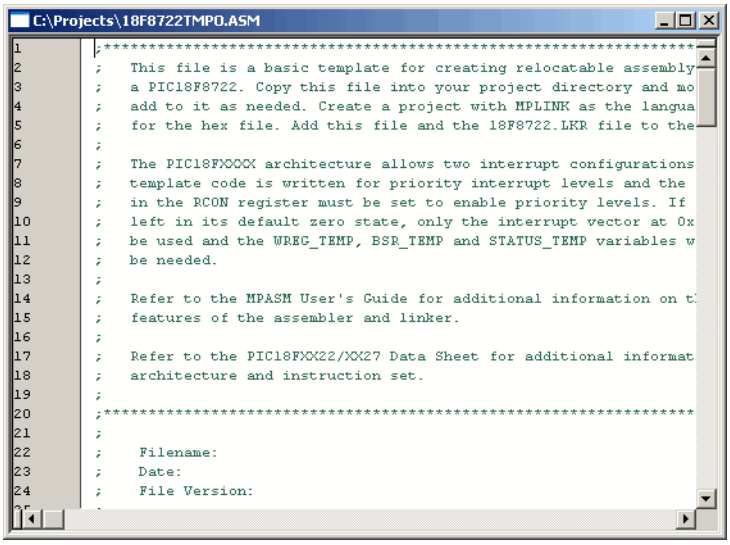
FIGURE 2-12: PROJECT CONTEXT MENU (RIGHT MOUSE CLICK)



The file has some comments at the beginning, and this area can be used as a standard comment information header for the file. For now you'll leave this as it is, but if this were a real project, you could put information about your design here.

Note: Line numbers are shown here. Line numbers may be toggled on/off by right clicking in the editor window, selecting Properties, and then checking/unchecking "Line Numbers" on the **'ASM' File Type** tab of the Editor Options dialog.

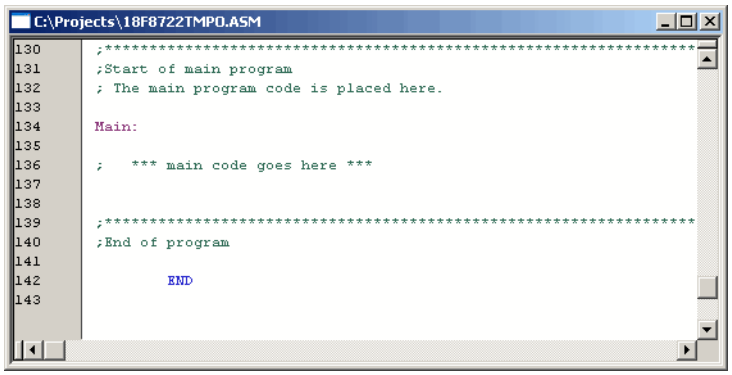
FIGURE 2-13: TEMPLATE FILE



The code in the first part of the file is for more advanced functions such as setting up interrupts and Configuration bits in a final application. These details can be ignored at this point with focus on writing the code. The new code will be placed in the file at the point after the symbol `Main` is defined.

Scroll down to the bottom of the file.

FIGURE 2-14: TEMPLATE FILE – MAIN



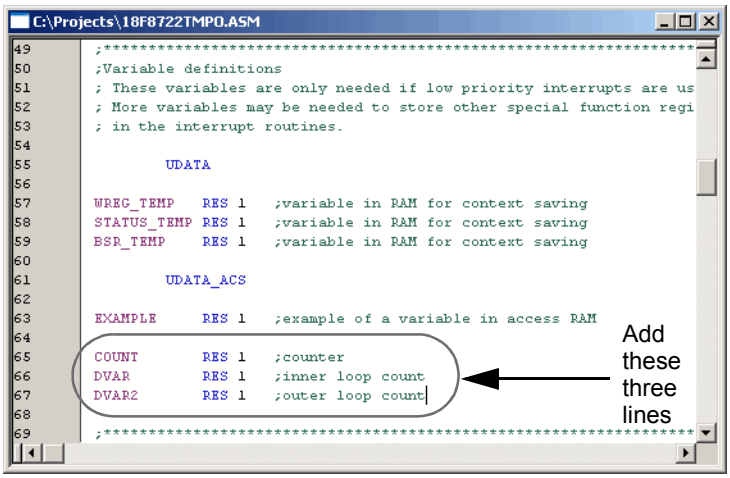
When any source file is opened, you are automatically in the editor. Type in this code beneath `Main:`

```
   clrf      WREG
   movwf     PORTC       ; clear PORTC
   movwf     TRISC       ; configure PORTC as all outputs

Init
   clrf      COUNT,A     ; initialize counter
IncCount
   incf      COUNT,F,A
   movf      COUNT,W,A   ; increase count and
   movwf     PORTC       ; display on PORTC

   call      Delay       ; go to Delay subroutine
   goto      IncCount    ; infinite loop

Delay
   movlw     0x40
   movwf     DVAR2,A     ; set outer delay loop
DelayOuter
   movlw     0xFF
   movwf     DVAR,A      ; set inner delay loop
DelayInner
   decfsz    DVAR,F,A
   goto      DelayInner

   decfsz    DVAR2,F,A
   goto      DelayOuter
   return
```

In this bit of code, we used three variables named COUNT, DVAR and DVAR2. These variables need to be defined in the template file in the UDATA_ACS section for uninitialized data using Access RAM. Using the Access bank will make the code simpler, as there will be no need to keep track of banks (banksel) for each variable.

There is already one variable in this section of the template file, and ours can be added at the end using the same format. Each variable is 8-bit, so only 1 byte each needs to be reserved.

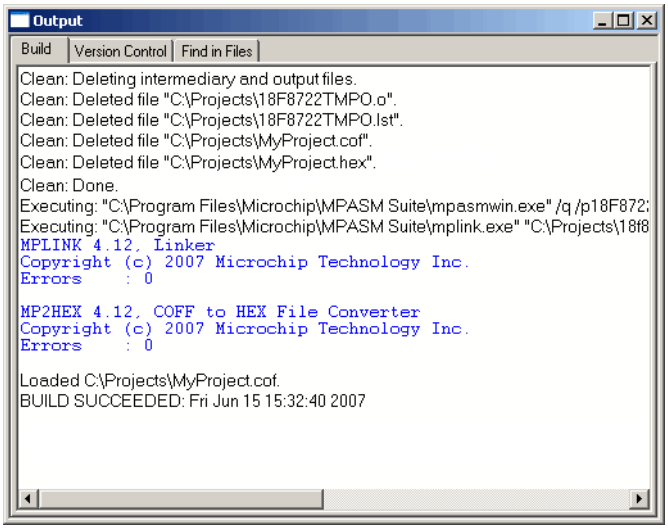
FIGURE 2-15: TEMPLATE FILE – ADD VARIABLES



2.11 BUILDING THE PROJECT AGAIN

Select *Project>Build All* to assemble and link the code. If the code assembled with no errors, the Output Window will look like Figure 2-16.

FIGURE 2-16: BUILD OUTPUT WINDOW



If the code did not assemble and link successfully, check the following items and then build the project again:

- If the assembler reported errors in the Output window, double click on the error and MPLAB IDE will open the corresponding line in the source code with a green arrow in the left margin of the source code window.
- Check the spelling and format of the code entered in the editor window. Make sure the new variables and the special function registers, TRISC and PORTC, are in upper case.
- Check that the correct assembler (MPASM assembler) and linker for PIC MCU devices is being used. Select *Project>Set Language Tool Locations*. Click on the plus boxes to expand the Microchip MPASM toolsuite and its executables. Click MPASM Assembler (`mpasmwin.exe`) and review their location in the display. If the location is correct, click **Cancel**. If it is not, change it and then click **OK**. The default search paths can be empty.

Upon a successful build, the output file generated by the language tool will be loaded. This file contains the object code that can be programmed into a PIC MCU and debugging information so that source code can be debugged and source variables can be viewed symbolically in Watch windows.

Note: The real power of projects is evident when there are many files to be compiled/assembled and linked to form the final executable application – as in a real application. Projects keep track of all of this. Build options can be set for each file that access other features of the language tools, such as report outputs and compiler optimizations.

2.12 TESTING CODE WITH THE SIMULATOR

In order to test the code, software or hardware is needed that will execute the PIC MCU instructions. A debug execution tool is a hardware or software tool that is used to inspect code as it executes a program (in this case `18F8722TMPO.ASM`). Hardware tools such as MPLAB ICE or MPLAB ICD 2 can execute code in real devices. If hardware is not available, the MPLAB SIM simulator can be used to test the code. For this tutorial use MPLAB SIM simulator.

The simulator is a software program that runs on the PC to *simulate* the instructions of the PIC MCU. It does not run in “real time,” since the simulator program is dependent upon the speed of the PC, the complexity of the code, overhead from the operating system and how many other tasks are running. However, the simulator accurately *measures* the time it would take to execute the code if it were operating in real time in an application.

Note: Other debug execution tools include MPLAB REAL ICE in-circuit emulator, MPLAB ICD 2 in-circuit debugger and MPLAB ICE 2000 in-circuit emulator. These are optional hardware tools to test code on the application PC board. Most of the MPLAB IDE debugging operations are the same as the simulator but, unlike the simulator, these tools allow the target PIC MCU to run at full speed in the actual target application.

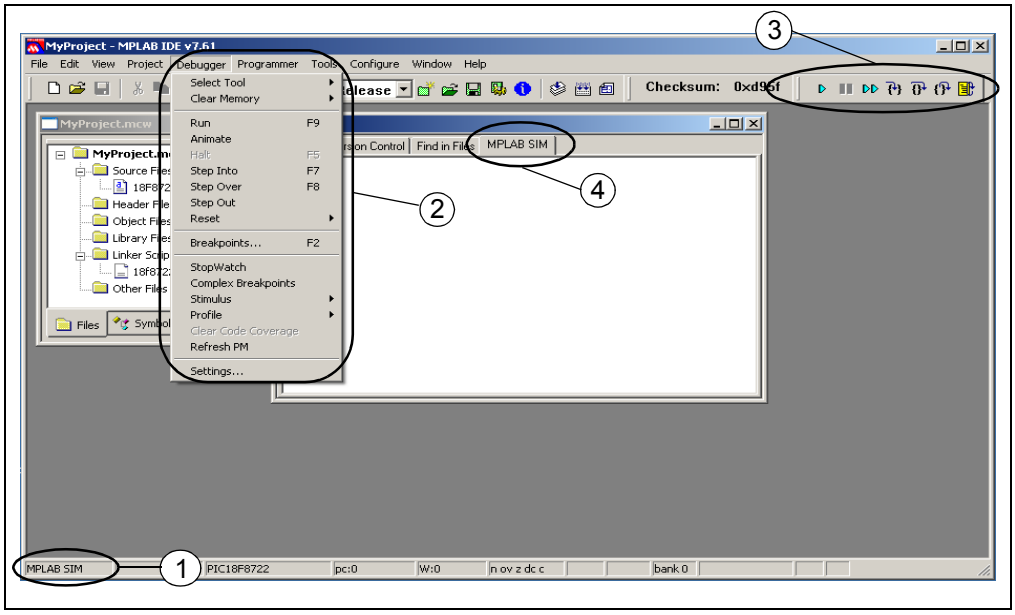
Select the simulator as the debug execution tool. This is done from the *Debugger>Select Tool* pull down menu. After selecting MPLAB SIM, the following changes should be seen (see corresponding numbers in Figure 2-17).

1. The status bar on the bottom of the MPLAB IDE window should change to “MPLAB SIM”.
2. Additional menu items should now appear in the Debugger menu.
3. Additional toolbar icons should appear in the Debug Tool Bar.

TIP: Position the mouse cursor over a toolbar button to see a brief description of the button’s function.

4. An MPLAB SIM tab is added to the Output window.

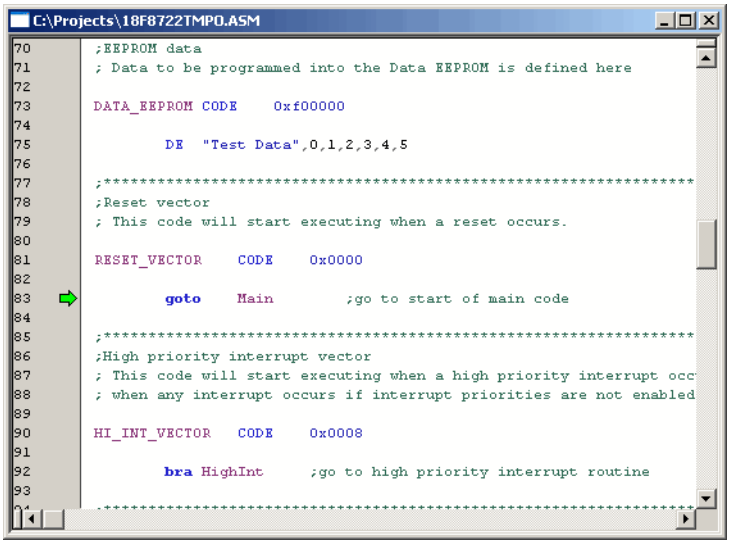
FIGURE 2-17: MPLAB® IDE DESKTOP WITH MPLAB SIM AS DEBUGGER



Now that your project is set up and the debug tool is selected, you should save your workspace setup. Select *File>Save Workspace*.

Next, select *Debugger>Reset>Processor Reset* and a green arrow shows where the program will begin. This was part of the template file. The first instruction in memory jumps to the label called `Main`, where your code was inserted, i.e., it jumps over the vector areas (reset vector, interrupt vectors, etc.) to the user memory space in program memory.

FIGURE 2-18: CODE AFTER PROCESSOR RESET



To single step through the application program, select *Debugger>Step Into*. This will execute the currently indicated line of code and move the arrow to the next line of code to be executed.

There are shortcuts for these commonly used functions in the Debug Tool Bar.

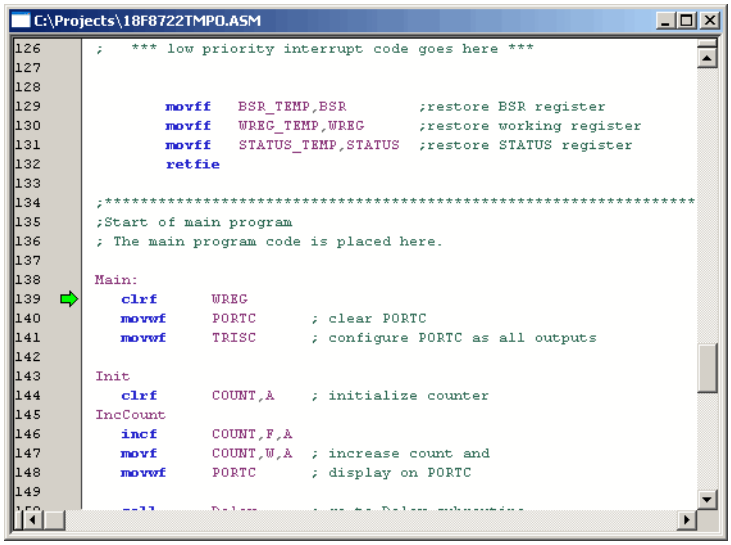
TABLE 2-1: DEBUG SHORT CUT ICONS

Debugger Menu	Toolbar Buttons	Hot Key
Run		F9
Halt		F5
Animate		
Step Into		F7
Step Over		F8
Step Out		
Reset		F6

TIP: Click on the appropriate icon on the toolbar or use the hot key shown next to the menu item. This is usually the best method for repeated stepping.

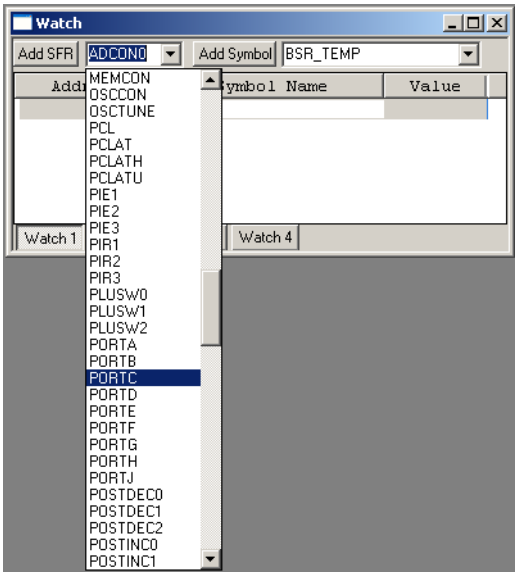
Next, press the Step Into icon or select *Debugger>Step Into* to single step to the code at `Main`.

FIGURE 2-19: CODE AFTER STEP INTO



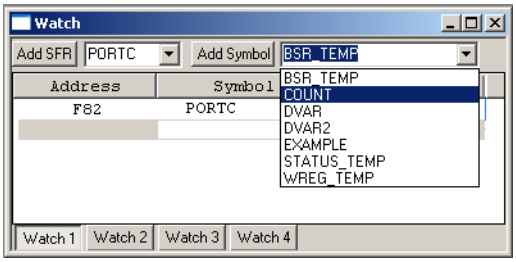
In order to see if the code is operating as intended, sending incrementing values out `PORTC`, watch the values being sent to `PORTC`. Select *View>Watch* to bring up an empty Watch window. There are two pull downs on the top of the Watch window. The one on the left labeled “Add SFR” can be used to add the Special Function Register, `PORTC`, into the watch. Select `PORTC` from the list and then click **Add SFR** to add it to the window.

FIGURE 2-20: WATCH – SELECT PORTC



The pull down on the right, allows symbols to be added from the program. Use this pull down to add the COUNT variable into the Watch window. Select COUNT from the list and then click **Add Symbol** to add it to the window.

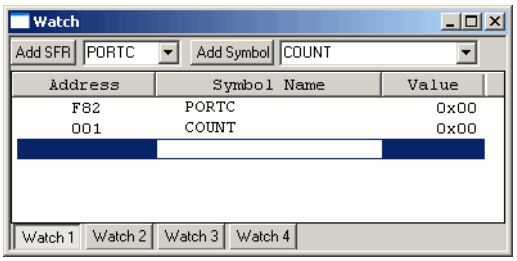
FIGURE 2-21: WATCH – SELECT VARIABLE "COUNT"



The Watch window should now show the address, value and name of the two registers. At this point in the program, they will both be zero.

Note: Items can also be added to the Watch window by either dragging them from the SFR, File Register or Editor window or by clicking directly in the window under symbol name and typing in the item.

FIGURE 2-22: WATCH – RESET VALUES



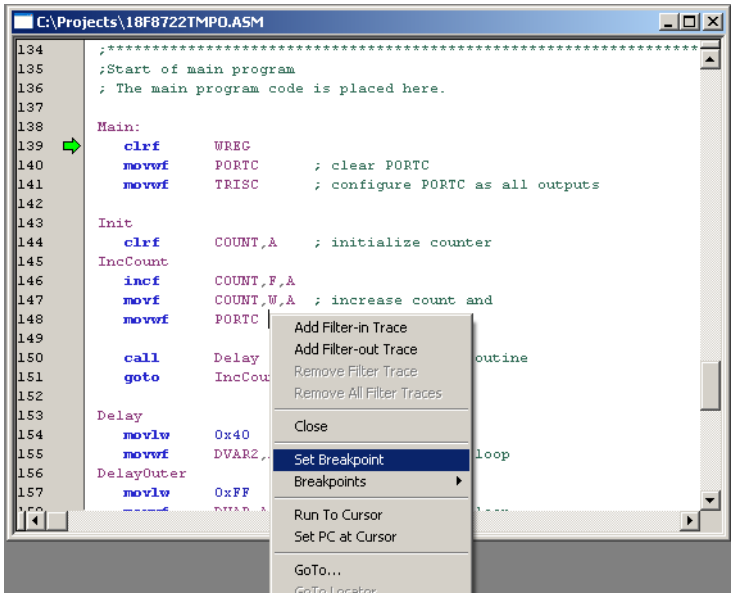
Now that Watch windows have been added, it is good idea to again save your workspace using *File>Save Workspace*.

You could continue single stepping through the code, but instead, set a breakpoint just before the first value is sent out to PORTC. To set a breakpoint, put the cursor on the line following line:

```
movwf      PORTC     ; display COUNT on PORTC
```

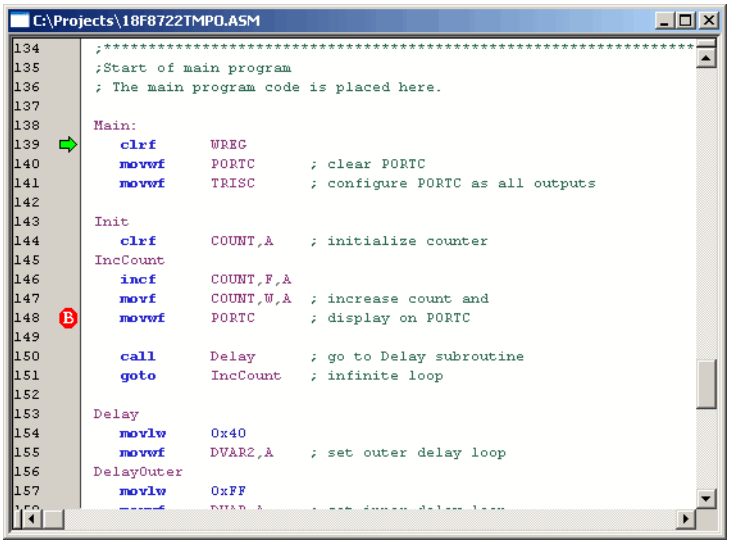
and click the right mouse button.

FIGURE 2-23: DEBUG CONTEXT MENU (RIGHT MOUSE CLICK ON LINE)



Select Set Breakpoint from the context menu. A red "B" will show on the line. (You can also double click on a line to add a breakpoint.)

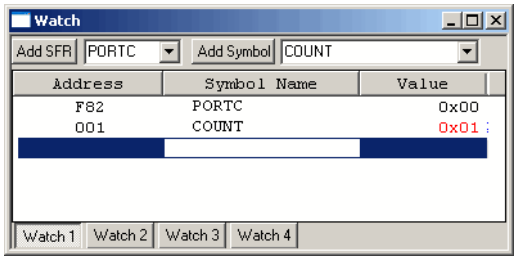
FIGURE 2-24: EDITOR WINDOW – SET BREAKPOINT



Select *Debugger>Run* to run the application. A text message “Running…” will briefly appear on the status bar before the application halts at this first breakpoint.

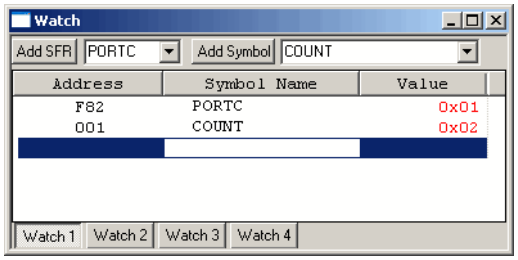
The Watch window should now show that the variable COUNT was incremented by one, but since the breakpoint is at the line before the move to PORTC executes, PORTC still has a value of zero.

FIGURE 2-25: WATCH – AT BREAKPOINT



Press the Run icon to execute the code until it hits this point again. The Watch window should now show both values incremented by one from their previous value.

FIGURE 2-26: WATCH – NEXT BREAKPOINT



This would seem to indicate that the program is working as designed. You can single step through the code, or run the code more times to verify that it is executing properly. If you single step into the delay loop, you will get stuck executing thousands of steps until reaching the end. To exit out of the delay loop, use *Debugger>Step Out*.

If you are interested in calculating your delay time, the data book could be used to determine how long each instruction would take in your delay loop and you would come up with a pretty accurate number. You can also use the MPLAB IDE Stopwatch to measure the delay. Your main interest should be the time each new value of COUNT is being displayed. If you set your breakpoint as was initially done, on the instruction that moves COUNT to PORTC, you can run to the next breakpoint at the same place to measure the time.

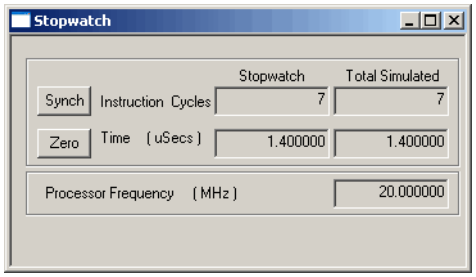
To use the Stopwatch, remove the breakpoint on PORTC by right clicking on the line and selecting “Remove Breakpoint”. Then, right click on the line

```
movf      COUNT,W,A  ; increase count and
```

and select “Set Breakpoint”. Finally, select *Debugger>Reset>Processor Reset*.

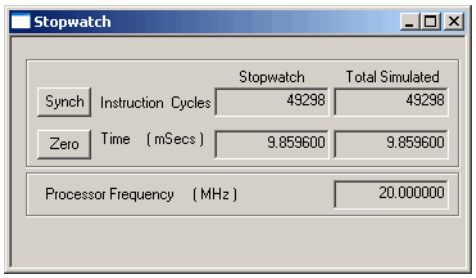
Use *Debugger>StopWatch* to bring up the Stopwatch dialog. Press *Debugger>Run* to run and then halt at the breakpoint. With the default processor frequency of 20 MHz, the Stopwatch should show that it took 1.4 microseconds to reach the first breakpoint.

FIGURE 2-27: STOPWATCH – AT FIRST BREAKPOINT



Execute Run again to go around the loop once, and note that the Stopwatch shows that it took about 9.8596 milliseconds. To change this, you can change the values in the delay loop. To change the Processor Frequency, select *Debugger>Settings*, **Osc/Trace** Tab.

FIGURE 2-28: STOPWATCH – AFTER DELAY



2.13 TUTORIAL SUMMARY

By completing this tutorial, you have performed the major steps for creating, building and testing a simple project. Tasks completed include:

- Selecting the device – the PIC18F8722.
- Using the Project Wizard to create a project, and using the wizard to:
  - select the MPLAB IDE built in MPASM assembler and MPLINK linker language tools,
  - add files for the project: a template file for the device selected and a linker script to build it properly.
- Writing some simple code to send a changing value out an I/O port.
- Building the project.
- And finally, testing the code with the simulator.

These are the essential steps for getting started with MPLAB IDE. You are now ready to continue exploring the capabilities of MPLAB IDE.

NOTES:

NOTES:

WORLDWIDE SALES AND SERVICE

AMERICAS

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

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

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

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

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

Detroit
Farmington Hills, MI
Tel: 248-538-2250
Fax: 248-538-2260

Kokomo
Kokomo, IN
Tel: 765-864-8360
Fax: 765-864-8387

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

Santa Clara
Santa Clara, CA
Tel: 408-961-6444
Fax: 408-961-6445

Toronto
Mississauga, Ontario,
Canada
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon
Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8528-2100
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Fuzhou
Tel: 86-591-8750-3506
Fax: 86-591-8750-3521

China - Hong Kong SAR
Tel: 852-2401-1200
Fax: 852-2401-3431

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8203-2660
Fax: 86-755-8203-1760

China - Shunde
Tel: 86-757-2839-5507
Fax: 86-757-2839-5571

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

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

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-4182-8400
Fax: 91-80-4182-8422

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

India - Pune
Tel: 91-20-2566-1512
Fax: 91-20-2566-1513

Japan - Yokohama
Tel: 81-45-471- 6166
Fax: 81-45-471-6122

Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Penang
Tel: 60-4-646-8870
Fax: 60-4-646-5086

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-572-9526
Fax: 886-3-572-6459

Taiwan - Kaohsiung
Tel: 886-7-536-4818
Fax: 886-7-536-4803

Taiwan - Taipei
Tel: 886-2-2500-6610
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

UK - Wokingham
Tel: 44-118-921-5869
Fax: 44-118-921-5820

06/25/07