



**Getting Started with
dsPIC30F
Digital Signal Controllers
User's 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.

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates.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. 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 ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip's products as critical components in life support systems is not authorized except with express written approval by Microchip. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights.

Trademarks

The Microchip name and logo, the Microchip logo, Accuron, dsPIC, KEELOQ, microID, MPLAB, PIC, PICmicro, PICSTART, PRO MATE, PowerSmart, rfPIC, and SmartShunt are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.


Amplab, FilterLab, Migratable Memory, MXDEV, MXLAB, PICMASTER, SEEVAL, SmartSensor and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, dsPICDEM, dsPICDEM.net, dsPICworks, ECAN, ECONOMONITOR, FanSense, FlexROM, fuzzyLAB, In-Circuit Serial Programming, ICSP, ICEPIC, Linear Active Thermistor, MPASM, MPLIB, MPLINK, MPSIM, PICKit, PICDEM, PICDEM.net, PICLAB, PICtail, PowerCal, PowerInfo, PowerMate, PowerTool, rLAB, rfPICDEM, Select Mode, Smart Serial, SmartTel, Total Endurance and WiperLock 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.

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

© 2005, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

 Printed on recycled paper.

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

Microchip received ISO/TS-16949:2002 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona and Mountain View, California in October 2003. The Company's quality system processes and procedures are for its PICmicro® 8-bit MCUs, 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.

Table of Contents

Preface	1
Chapter 1. The dsPIC30F Digital Signal Controller	
1.1 Introduction	7
1.2 Architecture	7
1.3 Device Variants	13
1.4 Applications	14
Chapter 2. The Microchip Development Tools	
2.1 Introduction	15
2.2 MPLAB IDE	15
2.3 Language Tools	16
2.4 Debugging Tools	17
2.5 Programming Tools	20
2.6 Development Boards	21
Chapter 3. MPLAB Integrated Development Environment	
3.1 MPLAB IDE Overview	25
3.2 Projects and Workspaces	26
3.3 Creating a Project	26
3.4 Building Code	30
Chapter 4. The MPLAB SIM Simulator	
4.1 MPLAB SIM Overview	33
4.2 Opening the Project	34
4.3 Selecting the Simulator	34
4.4 Resetting the Code	35
4.5 Stepping Through the Code	35
4.6 Running the Code	36
4.7 The Debug Toolbar and Hotkeys	36
4.8 Breakpoints	37
4.9 Watch Window	38
4.10 Simulator Settings	39
4.11 Stopwatch	40
4.12 Trace Buffer	41

Chapter 5. The MPLAB ICD 2 In-Circuit Debugger

5.1	MPLAB ICD 2 Overview	43
5.2	Setting Up the MPLAB ICD 2	44
5.3	Programming the dsPIC Device	46
5.4	Resetting the Code	46
5.5	Stepping Through the Code	47
5.6	Running the Code	48
5.7	The Debug Toolbar and Hotkeys	48
5.8	Breakpoints	49
5.9	Watch Window	50
5.10	Advanced Breakpoints	52

Chapter 6. MPLAB ICE 4000 In-Circuit Emulator

6.1	MPLAB ICE 4000 Overview	55
6.2	Opening the Project	57
6.3	Special Emulator Devices	57
6.4	Selecting the MPLAB ICE 4000	58
6.5	MPLAB ICE 4000 Settings	59
6.6	Resetting the Code	61
6.7	Stepping Through the Code	61
6.8	Running the Code	62
6.9	The Debug Toolbar and Hotkeys	62
6.10	Breakpoints	63
6.11	Watch Window	64
6.12	Stopwatch	66
6.13	Trace Buffer	67
6.14	Complex Triggers	68

Chapter 7. The MPLAB ASM30 Assembler

7.1	MPLAB ASM30 Assembler Overview	71
7.2	Commonly Used Directives	72
7.3	Example Code	75

Chapter 8. MPLAB C30 C Compiler

8.1	MPLAB C30 C Compiler Overview	81
8.2	MPLAB C30 C Compiler Projects	81
8.3	Creating a Project with the Project Wizard	82
8.4	Setting the Build Options	86
8.5	Building the Project	87
8.6	Language Features	87
8.7	Example Code	88

Chapter 9. The MPLAB LINK30 Linker

9.1	MPLAB LINK30 Linker Overview	91
9.2	Linker Script Files	92

Appendix A. Code for dsPICDEM 1.1 General Purpose Development Board	
A.1	Flash LED with dsPIC30F6014.s 99
A.2	Flash LED with dsPIC30F6014.c 102
Appendix B. Code for dsPICDEM Starter Demonstration Board	
B.1	Flash LED with dsPIC30F6012.s 105
B.2	Flash LED with dsPIC30F6012.c 108
Appendix C. Code for dsPICDEM 28-Pin Starter Demonstration Board	
C.1	Flash LED with dsPIC30F2010.s 111
C.2	Flash LED with dsPIC30F2010.c 114
Appendix D. Code for dsPICDEM 2 Development Board	
D.1	Flash LED with dsPIC30F4011.s 117
D.2	Flash LED with dsPIC30F4011.c 120
Index	123
Worldwide Sales and Service	126

NOTES:

Preface

NOTICE TO CUSTOMERS

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

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

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

INTRODUCTION

Welcome to your top source for comprehensive microcontroller solutions. Microchip Technology, the world leader in 8-bit microcontroller shipments, now offers the dsPIC[®] family of 16-bit digital signal controllers. Targeted at a wide variety of applications, the dsPIC30F offers the flexibility and control of a microcontroller with the computation and throughput capabilities of a digital signal processor.

The dsPIC30F is supported by several development tools centered around the industry-leading MPLAB[®] Integrated Development Environment (IDE). In this guide, you'll learn how to use the MPLAB IDE and related assembler, compiler, linker, simulator, debugger and emulator tools. All the tools are covered, so you'll find this guide useful, even if you don't have hardware yet. Hands-on tutorials maximize your learning experience and minimize your learning time. These tutorials use the MPLAB ICD 2 In-Circuit Debugger with either the dsPICDEM[™] Starter Demonstration Board or the dsPICDEM[™] 1.1 General Purpose Development Board to both program and debug the device. Tutorials are also available for the dsPICDEM[™] 28-Pin Starter Demonstration Board and the dsPICDEM[™] 2 Development Board.

Items discussed in this Preface include:

- About This Guide
- Recommended Reading
- The Microchip Web Site
- Development Systems Customer Change Notification Service
- Customer Support

ABOUT THIS GUIDE

This Getting Started guide covers the architecture and development tools of the dsPIC family and offers many tips on selecting the right dsPIC device for your design.

Document Layout

The manual is organized as follows:

- **Chapter 1: The dsPIC30F Digital Signal Controller** – This chapter will help you choose the right dsPIC30F device for your design or simply learn more about the features of this capable digital signal controller.
- **Chapter 2: The Microchip Development Tools** – This chapter introduces the MPLAB IDE and acquaints you with the related assembler, compiler, linker, simulator, debugger and emulator tools.
- **Chapter 3: MPLAB Integrated Development Environment** – Microchip provides a powerful development environment called MPLAB IDE, and it's absolutely free! This chapter uses a tutorial format to familiarize you with the MPLAB IDE by creating a project and assembling and linking a program.
- **Chapter 4: The MPLAB SIM Simulator** – The MPLAB SIM30 allows you to debug your code without the dsPIC30F hardware. The simulator is completely integrated into the MPLAB IDE and you can learn how to use it in this chapter.
- **Chapter 5: The MPLAB ICD 2 In-Circuit Debugger** – The MPLAB ICD 2 In-Circuit Debugger gives you the flexibility to debug directly in the dsPIC chip on your own circuit board. The MPLAB ICD 2 is an exceptional value and you'll learn how to use it in this chapter and in other hands-on tutorials in this guide.
- **Chapter 6: MPLAB ICE 4000 In-Circuit Emulator** – This chapter gets you started using the MPLAB ICE 4000 In-Circuit Emulator. The MPLAB ICE 4000 is the most sophisticated debugging tool available for the dsPIC devices. It provides full-speed emulation and visibility into both the instructions and the data paths during execution.
- **Chapter 7: The MPLAB ASM30 Assembler** – In this chapter, the focus shifts to producing code. It describes the general format and provides examples of instructions and directives that are assembled into object code by the MPLAB ASM30 Assembler.
- **Chapter 8: MPLAB C30 C Compiler** – This chapter gets you started generating 'C' code for your dsPIC30F device. The tutorial illustrates how to use the MPLAB C30 C Compiler to combine application source code and libraries to produce object files.
- **Chapter 9: The MPLAB LINK30 Linker** – This chapter examines the MPLAB LINK30 Linker by providing a step-by-step analysis of a linker script file.
- **Appendix A: Code for dsPICDEM 1.1 General Purpose Development Board** – This appendix contains the sample code for the dsPICDEM 1.1 General Purpose Development Board.
- **Appendix B: Code for dsPICDEM Starter Demonstration Board** – This appendix contains the sample code for the dsPICDEM Starter Demonstration Board.
- **Appendix C: Code for dsPICDEM 28-Pin Starter Demonstration Board** – This appendix contains the sample code for the dsPICDEM 28-Pin Starter Demonstration Board.
- **Appendix D: Code for dsPICDEM 2 Development Board** – This appendix contains the sample code for the dsPICDEM 2 Development Board.

Conventions Used in this Guide

This manual uses the following documentation conventions:

DOCUMENTATION CONVENTIONS

Description	Represents	Examples
Arial font:		
Italic characters	Referenced books	<i>MPLAB® IDE User's Guide</i>
	Emphasized text	...is the <i>only</i> compiler...
Initial caps	A window	the Output window
	A dialog	the Settings dialog
	A menu selection	select Enable Programmer
Quotes	A field name in a window or dialog	"Save project before build"
Underlined, italic text with right angle bracket	A menu path	<u><i>File>Save</i></u>
Bold characters	A dialog button	Click OK
	A tab	Click the Power tab
Text in angle brackets < >	A key on the keyboard	Press <Enter>, <F1>
Courier font:		
Plain Courier	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	A variable argument	<i>file.o</i> , where <i>file</i> can be any valid filename
Square brackets []	Optional arguments	mcc18 [options] <i>file</i> [options]
Curly brackets and pipe character: { }	Choice of mutually exclusive arguments; an OR selection	errorlevel {0 1}
Ellipses...	Replaces repeated text	var_name [, var_name...]
	Represents code supplied by user	void main (void) { ... }

RECOMMENDED READING

The following Microchip documents are available and recommended as supplemental reference resources.

dsPIC30F Family Reference Manual (DS70046)

Consult this document for detailed information on the dsPIC30F device operation. The manual explains the operation of the dsPIC30F DSC family architecture and peripheral modules but does not cover the specifics of each device. Refer to the appropriate device data sheet, mentioned below, for device-specific information.

dsPIC30F Data Sheet, Motor Control and Power Conversion Family (DS70082)

Consult this document for information regarding the dsPIC30F Motor Control and Power Conversion devices. Reference information found in this data sheet includes:

- Device memory map
- Device pinout and packaging details
- Device electrical specifications
- List of peripherals included on the device

dsPIC30F Data Sheet, General Purpose and Sensor Families (DS70083)

Consult this document for information regarding the dsPIC30F Sensor and General Purpose devices. Reference information found in this data sheet includes:

- Device memory map
- Device pinout and packaging details
- Device electrical specifications
- List of peripherals included on the device

dsPIC30F Programmer's Reference Manual (DS70030)

This manual is a software developer's reference for the dsPIC30F 16-bit DSC family of devices. This manual describes the instruction set in detail and also provides general information to assist the user in developing software for the dsPIC30F DSC family.

dsPIC30F Family Overview, dsPIC High Performance 16-bit Digital Signal Controller (DS70043)

This document provides an overview of the features and functionality of the dsPIC[®] product family. It helps determine how the dsPIC 16-bit Digital Signal Controller Family fits a specific product application. For detailed information about any of the functionality, refer to the "*dsPIC30F Family Reference Manual*" (DS70046).

MPLAB[®] ASM30, MPLAB[®] LINK30 and Utilities User's Guide (DS51317)

This document details Microchip Technology's language tools for dsPIC devices based on GNU technology. The language tools discussed are:

- MPLAB ASM30 Assembler
- MPLAB LINK30 Linker
- MPLAB LIB30 Archiver/Librarian
- Other Utilities

MPLAB[®] C30 C Compiler User's Guide (DS51284)

The purpose of this document is to help you use Microchip's MPLAB C30 C Compiler for dsPIC devices to develop your application. MPLAB C30 C Compiler is a GNU-based language tool, based on source code from the Free Software Foundation (FSF). For more information about the FSF, see www.fsf.org.

Readme Files

For the latest information on using other tools, read the tool-specific Readme files in the Readme subdirectory of the MPLAB IDE installation directory. The Readme files contain update information and known issues that may not be included in this user's guide.

THE MICROCHIP WEB SITE

Microchip provides online support via our WWW site at www.microchip.com. This web site is used as a means to make files and information easily available to customers. Accessible by using your favorite Internet browser, the web site contains the following information:

- **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 (FAQ), technical support requests, online discussion groups, Microchip consultant 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

DEVELOPMENT SYSTEMS CUSTOMER CHANGE NOTIFICATION SERVICE

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

To register, access the Microchip web site at www.microchip.com, click on Customer Change Notification and follow the registration instructions.

The Development Systems product group categories are:

- **Compilers** – The latest information on Microchip C compilers and other language tools. These include the MPLAB C17, MPLAB C18 and MPLAB C30 C Compilers; MPASM™ and MPLAB ASM30 Assemblers; MPLINK™ and MPLAB LINK30 Object Linkers; and MPLIB™ and MPLAB LIB30 Object Librarians.
- **Emulators** – The latest information on Microchip in-circuit emulators. This includes the MPLAB ICE 2000 and MPLAB ICE 4000.
- **In-Circuit Debuggers** – The latest information on the Microchip in-circuit debugger, MPLAB ICD 2.
- **MPLAB IDE** – The latest information on Microchip MPLAB IDE, the Windows® Integrated Development Environment for development systems tools. This list is focused on the MPLAB IDE, MPLAB SIM and MPLAB SIM30 Simulators, MPLAB IDE Project Manager and general editing and debugging features.
- **Programmers** – The latest information on Microchip programmers. These include the MPLAB PM3 and PRO MATE® II Device Programmers and the PICSTART® Plus Development Programmer.

CUSTOMER SUPPORT

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Field Application Engineer (FAE)
- Technical Support
- Development Systems Information Line

Customers should contact their distributor, representative or field application engineer (FAE) for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in the back of this document.

Technical support is available through the web site at: <http://support.microchip.com>

In addition, there is a Development Systems Information Line which lists the latest versions of Microchip's development systems software products. This line also provides information on how customers can receive currently available upgrade kits.

The Development Systems Information Line numbers are:

1-800-755-2345 – United States and most of Canada

1-480-792-7302 – Other International Locations

Chapter 1. The dsPIC30F Digital Signal Controller

1.1 INTRODUCTION

The 16-bit dsPIC30F Digital Signal Controller (DSC) is Microchip's newest and most advanced processor family. In this chapter, you will learn about the features of this processor, the different dsPIC devices available and how to choose the most suitable dsPIC device for your application.

The dsPIC30F is an advanced 16-bit processor that offers true DSP capability with the fundamental real-time control capabilities of a microcontroller. Prioritized interrupts, extensive built-in peripherals and power management features are combined with a full-featured DSP engine. Dual 40-bit accumulators, single-cycle 16x16 MAC, 40-bit barrel shifter, dual-operand fetches and zero-overhead looping are among the features that make this a very capable DSC.

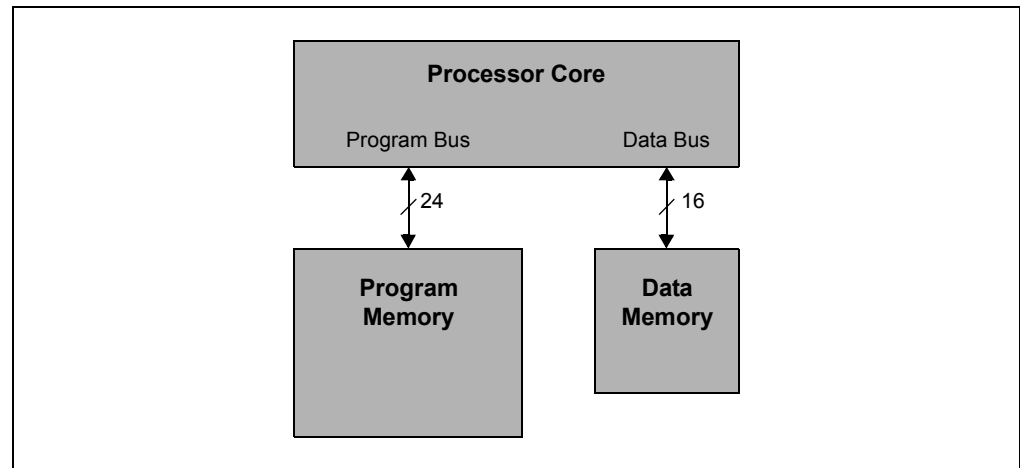
If you don't understand these terms, don't worry. They are covered in more detail in this chapter.

1.2 ARCHITECTURE

1.2.1 Harvard Architecture

The dsPIC processor uses a modified Harvard architecture with separate program and data memory buses, as shown in Figure 1-1.

FIGURE 1-1: SEPARATE DATA AND PROGRAM BUSSES



The Harvard architecture allows different size data (16 bits) and instruction (24 bits) words. This design improves the efficiency of the instruction set. It also allows faster processing because the dsPIC processor can prefetch the next instruction from program memory while it executes the current instruction that accesses data RAM.

1.2.2 Program Memory and Program Counter

The Program Counter (PC) is 24 bits wide and addresses up to 4M x 24 bits of user program memory space. The Program Counter increments by two for each 24-bit instruction, which simplifies the addressing of 16-bit data constants stored in program memory. Program memory space contains the Reset location, Interrupt Vector Tables, user program memory, data EEPROM and configuration memory (see the Program Memory Map in Figure 1-2).

The processor begins program execution at the Reset location 0x000000. This location should be user-programmed with a `GOTO` instruction, which branches to the start of the code. The `GOTO` instruction at the Reset location is followed by the Interrupt Vector Tables. Program memory for the code starts after the vector tables at address 0x100.

Program looping is accomplished with minimal overhead with the `DO` and `REPEAT` instructions; both can be interrupted at any time. These features make repetitive DSP algorithms very efficient, while maintaining the ability to handle real-time events.

1.2.3 Data Memory

The data space is 64 Kbytes and is treated as one linear address space by most instructions. When certain DSP instructions, known as DSP multiply instructions, are used, the memory is split into two blocks called X and Y data memory (see the Data Memory Map in Figure 1-2). As a result, these DSP instructions support dual operand reads; that is, data can be simultaneously fetched from X memory and from Y memory for a single instruction. The X and Y data space boundary is fixed for any given device. When not doing DSP instructions, all the memory is treated as a single block of X memory.

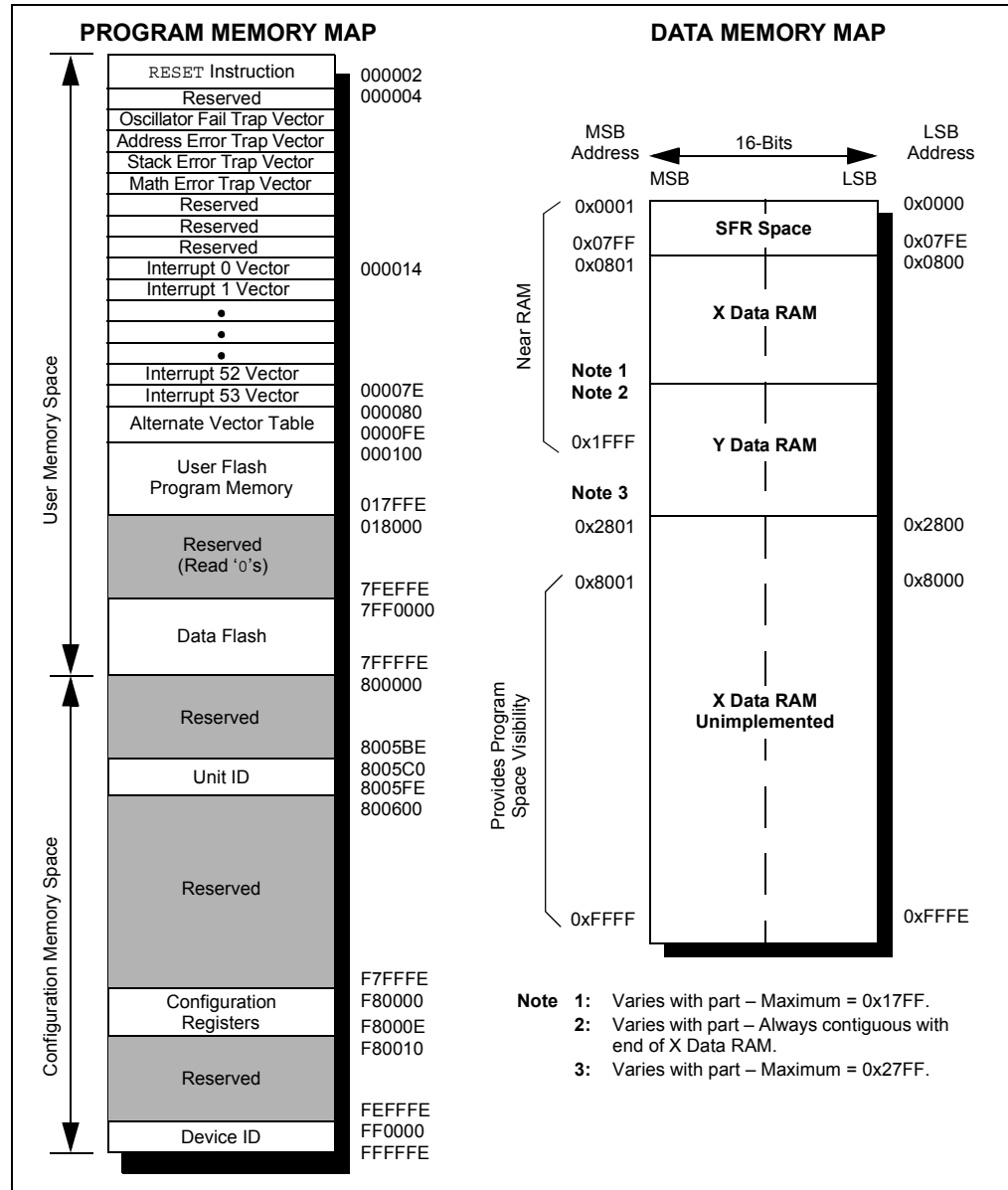
The first 2-Kbyte block of data memory is allocated to the Special Function Registers (SFRs). The SFRs are control and status registers for core and peripheral functions in the dsPIC devices.

After the SFRs, up to 8 Kbytes are implemented as data RAM. This RAM is general purpose memory that can be used for data storage. It is split into X and Y memory for DSP instructions.

The first 8 Kbytes of data space (i.e., all 2 Kbytes of SFRs and the first 6 Kbytes of data RAM) are called Near RAM. This region of RAM can be accessed directly via file register instructions. Some instructions cannot directly access RAM that is not near and must use indirect addressing.

The last 32 Kbytes of data RAM space are not implemented but can be mapped into program space for Program Space Visibility (PSV). PSV allows tables in program memory to be read as though they were in data RAM. (This feature can be quite useful for accessing DSP filter coefficients.)

FIGURE 1-2: PROGRAM AND DATA MEMORY



1.2.4 Working Register Array

The dsPIC devices have sixteen 16-bit working registers. The last working register (W15) always operates as the software stack pointer. W15 cannot be used for other purposes. The remaining working registers can act as a data register, data address pointer or address offset register. The software stack is used for return addresses of interrupts and calls, as well as `PUSH` and `POP` instructions. C compilers make extensive use of the software stack for storing local variables.

1.2.5 Data Addressing Modes

The CPU supports Inherent (no operand), Relative, Literal, Memory Direct, Register Direct and Register Indirect Addressing modes. Relax, these are not as complicated as they sound. Each instruction that addresses data memory can use some of the available addressing modes. As many as six addressing modes are supported for each instruction. The working registers are used extensively as address pointers for the indirect addressing modes. They can be modified (e.g., incremented) and used as pointers in the same instruction.

1.2.6 Modulo and Bit-Reversed Addressing

Modulo addressing allows circular buffers to be implemented with no processor overhead to check the boundaries of the buffer. The pointer for the buffer can be set up to automatically wrap around to the beginning of the buffer after it reaches the end and vice versa. This can be done in both X and Y memory, significantly reducing the overhead for DSP algorithms.

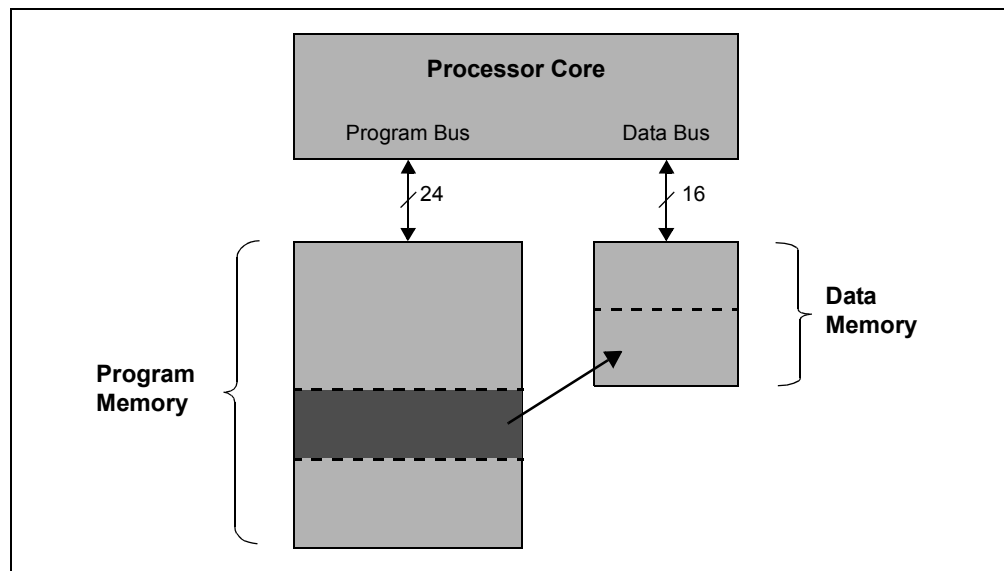
Bit-reversed addressing greatly simplifies input or output data reordering for radix-2 FFT algorithms. Bit-reversed addressing is supported by X memory.

1.2.7 Program Space Visibility

The upper 32 Kbytes of the data space memory map can optionally be mapped into program space at any 16K program word (32-Kbyte) boundary defined by the 8-bit Program Space Visibility Page (PSVPAG) register.

The program-to-data space mapping feature lets any instruction access program space as if it were data space. This capability is useful for look-up tables, especially tables of filter coefficients in DSP algorithms.

FIGURE 1-3: PROGRAM SPACE VISIBILITY



1.2.8 Instruction Set

The dsPIC30F instruction set has two classes of instructions: MCU instructions and DSP instructions. These two instruction classes are seamlessly integrated into the architecture and are carried out from a single execution unit. The instruction set includes many addressing modes and was designed for optimum C compiler efficiency.

With just a few exceptions, instructions execute in a single cycle. The exceptions are instructions that change the program flow (*BRA*, *CALL*, etc.), the double-word move (*MOV.D*) instruction and program memory read/write (table) instructions.

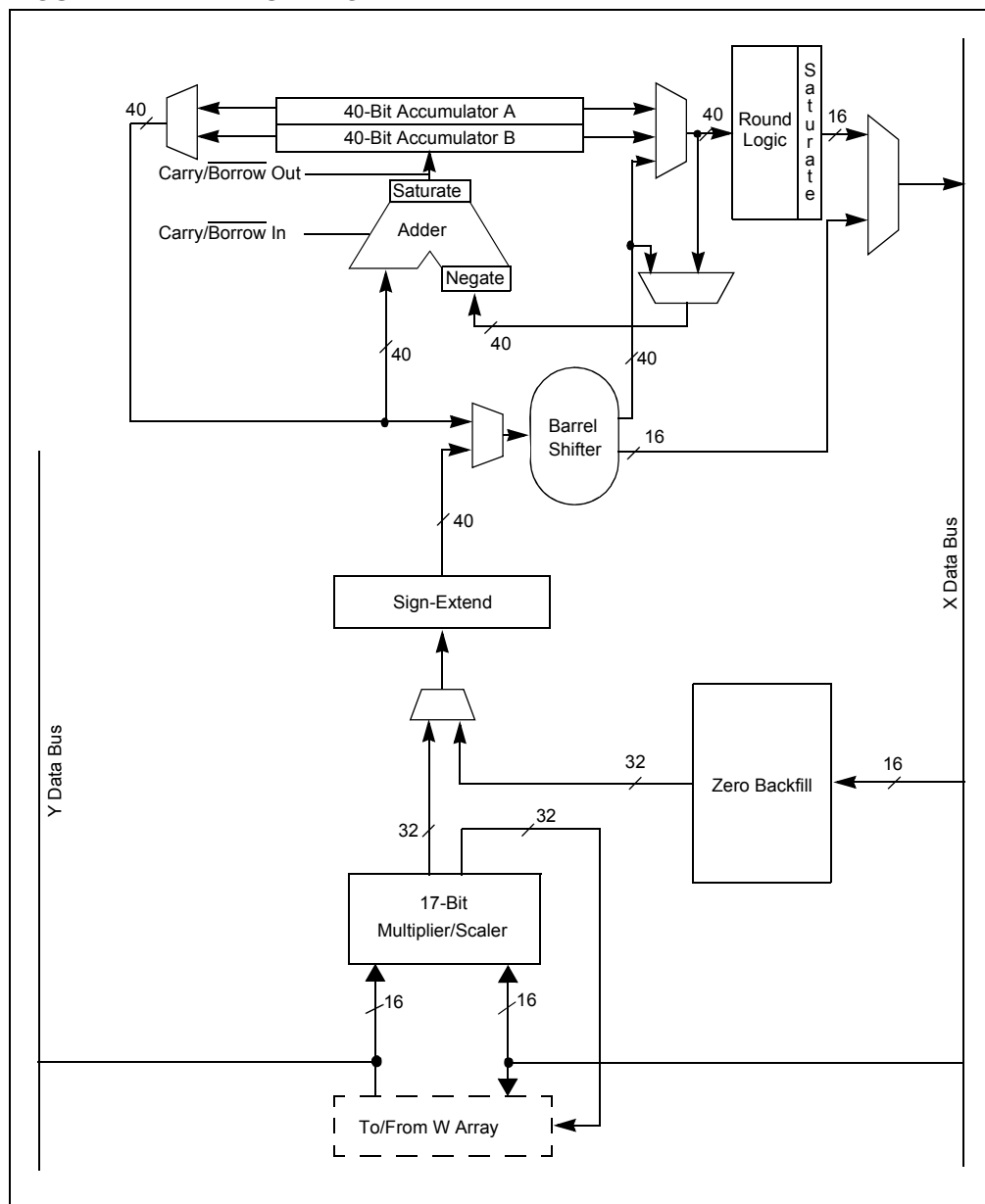
For most instructions, the dsPIC30F is capable of executing a data memory read, a working register data read, a data memory write and a program memory (instruction) read, all during one instruction cycle. As a result, three-operand instructions can be supported, allowing $A + B = C$ type operations to be executed in a single cycle.

1.2.9 DSP Engine

The DSP engine (Figure 1-4) features a high-speed, 17-bit x 17-bit fixed-point multiplier, a 40-bit ALU (Arithmetic Logic Unit), two 40-bit saturating accumulators and a 40-bit bidirectional barrel shifter. The barrel shifter is capable of shifting a 40-bit value up to 15 bits right, or up to 16 bits left, in a single cycle.

The DSP instructions operate seamlessly with all other instructions and have been designed for optimal real-time performance. The MAC instruction and other associated instructions can concurrently fetch two data operands from memory while multiplying two W registers. This is possible because the data memory is split into X and Y memory spaces for DSP instructions.

FIGURE 1-4: DSP ENGINE



1.2.10 Interrupts

The dsPIC30F has a vectored interrupt scheme. Each interrupt source has its own vector and can be dynamically assigned one of seven priority levels. The interrupt entry and return latencies are fixed, providing deterministic timing for real-time applications.

The Interrupt Vector Table (IVT) resides in program memory, immediately following the instruction at the Reset location, as shown in Figure 1-5. The IVT contains 62 vectors consisting of up to eight non-maskable (always enabled) error trap vectors and up to 54 sources of interrupt. Each interrupt vector contains the 24-bit wide starting address of the associated Interrupt Service Routine (ISR).

The Alternate Interrupt Vector Table (AIVT) is located after the IVT in program memory. If the ALTIVT bit is set, all the interrupts and error traps will use the alternate vectors instead of the default vectors. The alternate vectors are organized in the same manner as the default vectors and provide a means to switch between an application and a test, setup or bootloader environment without requiring the interrupt vectors to be reprogrammed.

FIGURE 1-5: INTERRUPT VECTOR TABLE

Decreasing Priority ↓	IVT ↑	Reset – GOTO Instruction	0x000000
		Reset – GOTO Address	0x000002
		Reserved	0x000004
		Oscillator Fail Trap Vector	
		Address Error Trap Vector	
		Stack Error Trap Vector	
		Math Error Trap Vector	
		Reserved Vector	
		Reserved Vector	
		Reserved Vector	
		Interrupt 0 Vector	0x000014
		Interrupt 1 Vector	
	AIVT ↑	—	
		—	
		—	
		Interrupt 52 Vector	
		Interrupt 53 Vector	0x00007E
		Reserved	0x000080
		Reserved	0x000082
		Reserved	0x000084
		Oscillator Fail Trap Vector	
		Stack Error Trap Vector	
		Address Error Trap Vector	
		Math Error Trap Vector	
		Reserved Vector	
		Reserved Vector	
		Reserved Vector	
		Interrupt 0 Vector	0x000094
		Interrupt 1 Vector	
		—	
		—	
		—	
		Interrupt 52 Vector	
		Interrupt 53 Vector	0x0000FE

1.2.11 System and Power Management

Modern applications often require flexible operating modes to conserve battery power, reduce EMI and handle Fault conditions. The dsPIC device has many system and power management features. For example, it has several oscillator modes with clock switching and oscillator failure detection. There are several power-saving modes that can selectively shut down and wake-up parts of the processor and peripherals. There are other safety features, such as Low-Voltage Detection, Brown-out Reset, Watchdog Timer Reset and several error traps.

1.2.12 Peripherals

The dsPIC devices are available with a wide range of peripherals to suit a diverse assortment of applications. The main peripherals include:

- I/O Ports
- Timers
- Input Capture
- Output Compare/PWM
- Motor Control PWM
- Quadrature Encoder
- 10-bit or 12-bit A/D Converter
- UART
- SPI™
- I²C™
- Data Converter (CODEC) Interface
- Controller Area Network (CAN)

Each device variant has a subset of these peripherals.

1.3 DEVICE VARIANTS

The dsPIC devices fall into three broad families. They are characterized this way to help you pick the most suitable part for your application:

- General Purpose
- Motor Control/Power Conversion
- Sensor

1.3.1 General Purpose Family

The general purpose devices are 40 to 80-pin parts ideal for a variety of 16-bit embedded applications. This variant family has:

- 12-bit, 100-ksps A/D Converter
- Dual UARTs
- CODEC Interface (most devices)
- CAN Interface (most devices)
- Timers, Input Capture, Output Compare
- UART, SPI, I²C Serial Interfaces

The parts with CODEC interfaces can support many audio applications.

1.3.2 Motor Control and Power Conversion Family

The motor control devices are 28 to 80-pin parts designed to support motor control applications. They are also suited for Uninterruptible Power Supplies (UPS), inverters, switched mode power supplies and related equipment. This variant family has:

- 10-bit, 500-ksps A/D Converter
- Motor Control PWM
- Quadrature Encoder
- Timers, Input Capture, Output Compare
- UART, SPI, I²C, CAN Serial Interfaces

1.3.3 Sensor Family

The sensor devices are small 18 to 28-pin parts designed to support low-cost embedded control applications. They have most of the features of the general purpose family but fewer of each peripheral. This variant family has:

- 12-bit, 100-ksps A/D Converter
- Timers, Input Capture, Output Compare
- UART, SPI, I²C Serial Interfaces

1.4 APPLICATIONS

Now that you have a basic understanding of the dsPIC architecture, you can consider the suitability of the dsPIC device for your particular application. There are endless possibilities, but here are the most common applications for the dsPIC devices:

1.4.1 Motor Control

The dsPIC device is ideal for motor control that needs more than a basic 8-bit microcontroller. Brushless DC, AC Induction and Switch Reluctance motors can all be controlled with a dsPIC device. The applications might require sensorless control, torque management, variable speed, position or servo control. Noise reduction and energy efficiency applications can also be handled.

1.4.2 Power Conversion and Monitoring

The fast A/D converter and multiple PWM modules make the dsPIC devices well-suited for many power conversion and power management applications. Uninterruptible power supplies (UPS), inverters and power management units for complex equipment can all be handled.

1.4.3 Internet Connectivity

Ethernet and modem applications for Internet connectivity are supported with Microchip's ready to use TCP/IP, Ethernet driver and Soft Modem application libraries.

1.4.4 Speech and Audio

The dsPIC device can support many audio applications, such as noise and echo cancellation, speech recognition and speech playback. It can also be used as a companion chip to a main DSP in high-end audio application to handle other tasks, such as digital tuning, equalizers and more.

1.4.5 Sensor Control

The smaller dsPIC devices are ideal for advanced sensor control. The A/D converter and serial communication peripherals, combined with the power management features, make it possible to create smart sensor interface modules.

1.4.6 Automotive

Microchip Technology Inc. is QS-9000 and ISO/TS-16949 certified and has automotive temperature grades parts. Traditionally, our products have had long life cycles to support product life cycles typical of automotive applications.

Chapter 2. The Microchip Development Tools

2.1 INTRODUCTION

Now that you've decided which dsPIC device suits your application, you'll need development tools. The development process can be broken down into three distinct steps:

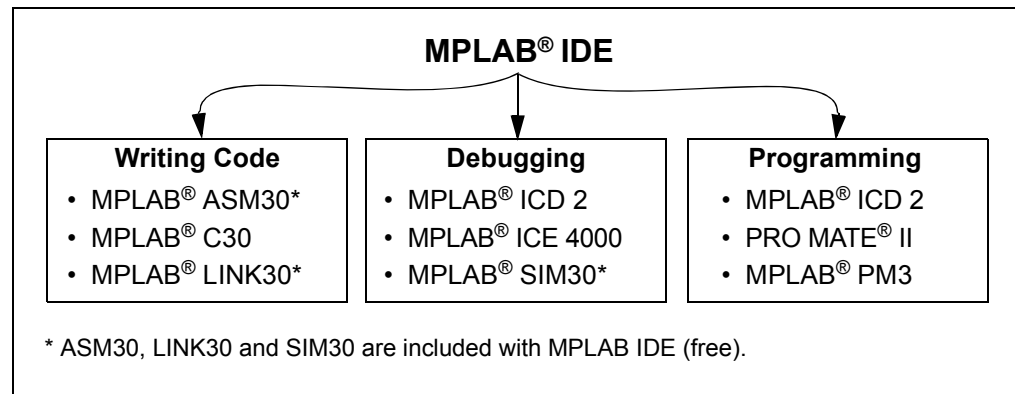
- Writing your code
- Debugging the code
- Programming devices

You'll need a tool to serve each of these functions and the key to supporting the development tools is the MPLAB[®] Integrated Development Environment (IDE). To get started, you'll find that the MPLAB ICD 2 In-Circuit Debugger offers the most cost-effective debugging and programming solution. The MPLAB ICD 2 can be used with any of the dsPIC development boards making an ideal learning platform.

2.2 MPLAB IDE

The MPLAB IDE allows you to develop a project from beginning to end, all within the same environment. You don't need to use a separate editor, assembler/compiler and programming utility to create, debug and program your applications. The MPLAB IDE can control all aspects of this process, as illustrated in Figure 2-1 and remember, MPLAB IDE is **free**!

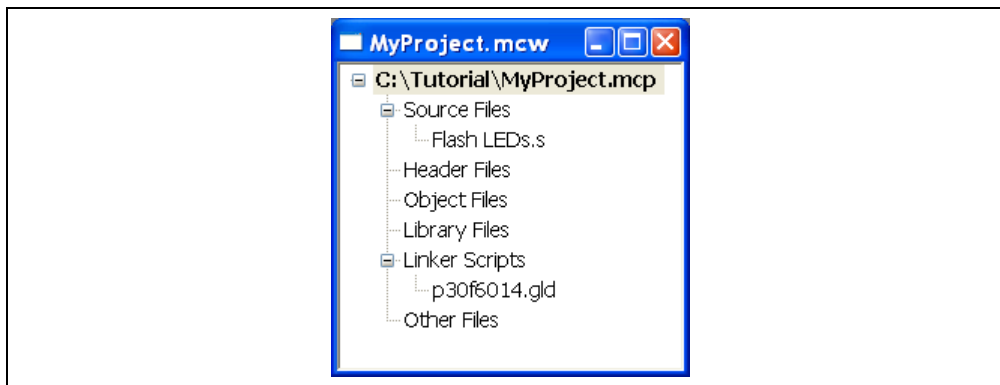
FIGURE 2-1: MPLAB[®] INTEGRATED DEVELOPMENT ENVIRONMENT



2.2.1 Projects

The MPLAB IDE includes tools to create and use projects and workspaces. A workspace stores all the settings for a project so that you can swap between projects with minimum effort. The Project Wizard allows projects to be easily created with a few mouse clicks. You can conveniently add and remove files in a project using the Project window view (Figure 2-2).

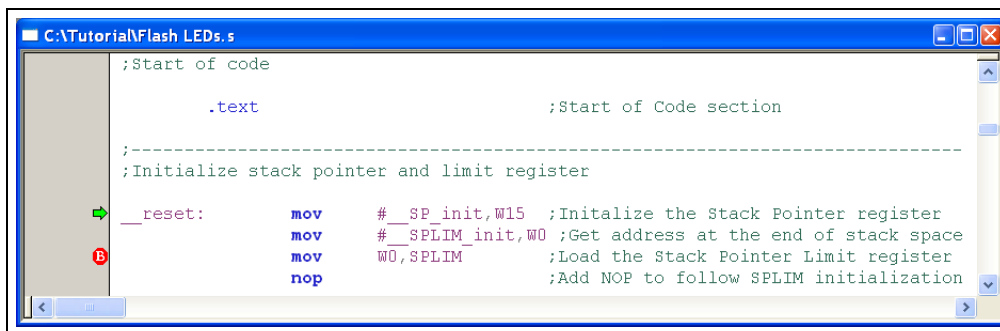
FIGURE 2-2: PROJECT WINDOW



2.2.2 Editor

The editor is an integral part of the MPLAB IDE and provides many features that make writing your code easy: syntax highlighting, automatic indentation, brace matching, block commenting, bookmarks and many others. The Editor window (Figure 2-3) directly supports the debugging tools, showing the current execution position, break and trace points, mouseover viewing of variables and so forth.

FIGURE 2-3: EDITOR WINDOW



2.3 LANGUAGE TOOLS

2.3.1 Assembler/Linker

MPLAB IDE includes the MPLAB ASM30 Assembler and the MPLAB LINK30 Linker based on the industry standard GNU toolsuite. You don't need to purchase any additional software to develop code. MPLAB ASM30 Assembler assembles source files into object files, which the linker converts to an output hex file, along with any library (archive) files that may be included in the project.

2.3.2 Compilers

For those who need a C compiler, Microchip offers the MPLAB C30 C Compiler. Available for purchase separately, MPLAB C30 C Compiler allows your code to be more portable, readable and maintainable. MPLAB C30 C Compiler can be used from within MPLAB IDE to give you seamlessly integrated code development, debugging and programming.

Aside from the MPLAB C30 C Compiler, dsPIC compilers are also available from third party manufacturers. HI-TECH Software (www.htsoft.com), CCS, Inc. (www.ccsinfo.com) and IAR Systems (www.iar.com) all have C compilers that support the dsPIC family.

For those familiar with the other PICmicro® compilers from these manufacturers, their dsPIC offerings would be a logical choice. By reducing the need to learn a whole new compiler, these compilers offer an easy way to migrate to the dsPIC family.

2.3.3 Template, Include and Linker Script Files

Want to start writing some code, but don't know how to begin? Then take a look at the template files in the MPLAB ASM30 Assembler directory that is part of MPLAB IDE. These templates can be copied and used to form the basis of your own code. You'll also find processor include files; they define all the register and bit names and their locations, consistent with the data sheet definitions. Linker script files provide the linker with a memory map of the dsPIC devices for proper automatic code and data placement.

2.3.4 Application Notes

Not sure how to implement your design? Just want to brush up on your design skills? Got some time to kill? Then check out our web site (www.microchip.com) for the latest application notes. We are always adding more application notes to provide you with examples on how to use the dsPIC devices in an ever-expanding array of applications.

2.4 DEBUGGING TOOLS

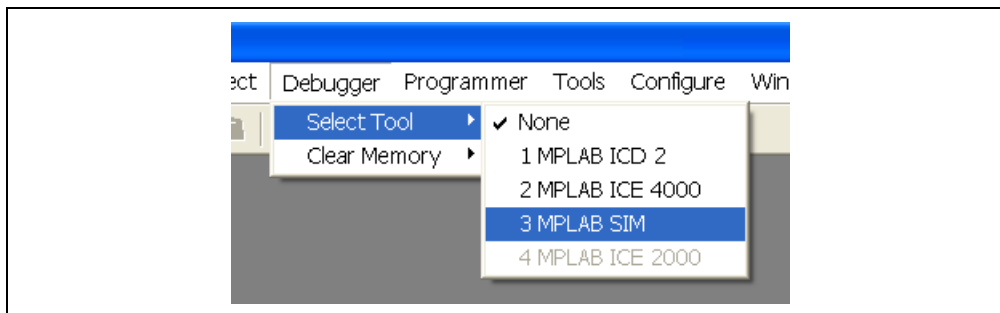
Three different debugging tools can be used with MPLAB IDE: the simulator (MPLAB SIM30 Software Simulator), the in-circuit debugger (MPLAB ICD 2) and the in-circuit emulator (MPLAB ICE 4000). All of these debuggers give you the ability to step through code, run till you choose to halt or hit a breakpoint, watch registers update and view memory contents. Each has its own particular advantages and disadvantages.

2.4.1 MPLAB SIM30 Software Simulator

The MPLAB SIM30 Software Simulator is a powerful debugging tool included with MPLAB IDE (Figure 2-4). The simulator runs on your PC and simulates code execution in the dsPIC devices. Not only can the simulator be used to mimic code execution, but it can also be used to respond to simulated external inputs and peripheral operations, and measure code execution time.

The MPLAB SIM30 Software Simulator offers a quick and easy way to debug code without the need for external hardware. It is particularly useful for testing mathematical operations and DSP functions when repeatable data from a file can be provided. Often, it can be challenging to test code on an analog signal in real hardware because of the difficulty in duplicating the data. By supplying sampled or synthesized data as stimulus, testing is made easier.

FIGURE 2-4: SIMULATOR SELECTION MENU IN MPLAB® IDE



The MPLAB SIM30 Software Simulator has all the basic debugging features and some more advanced features:

- Stopwatch – for timing code execution.
- Stimulus – for simulating external inputs and data reception.
- Trace – for viewing recorded execution.

FIGURE 2-5: MPLAB® ICE 4000



2.4.2 MPLAB ICE 4000

The MPLAB ICE 4000 In-Circuit Emulator (Figure 2-5) is a full-featured debugging tool, capable of emulating all of the dsPIC30F devices at full speed. It is the most powerful debugging tool we offer and gives excellent visibility into the processor. It is fully integrated into MPLAB IDE with a USB interface, which allows MPLAB IDE to update memory and data views very quickly.

MPLAB ICE 4000 is a modular system that supports a variety of processors and package options. Use the “*Product Selector Guide*” available on our web site (www.microchip.com) to select the correct processor module, device adapter and transition socket to emulate the particular device that you wish to use.

The MPLAB ICE 4000 has all the basic debugging features and many more advanced features:

- Complex trigger settings – to detect event sequences such as writes to registers.
- Stopwatch – for timing code execution.
- Trace – for viewing recorded execution.
- Logic probes – to trigger on external signals and generate triggers for test equipment.

FIGURE 2-6: MPLAB® ICD 2

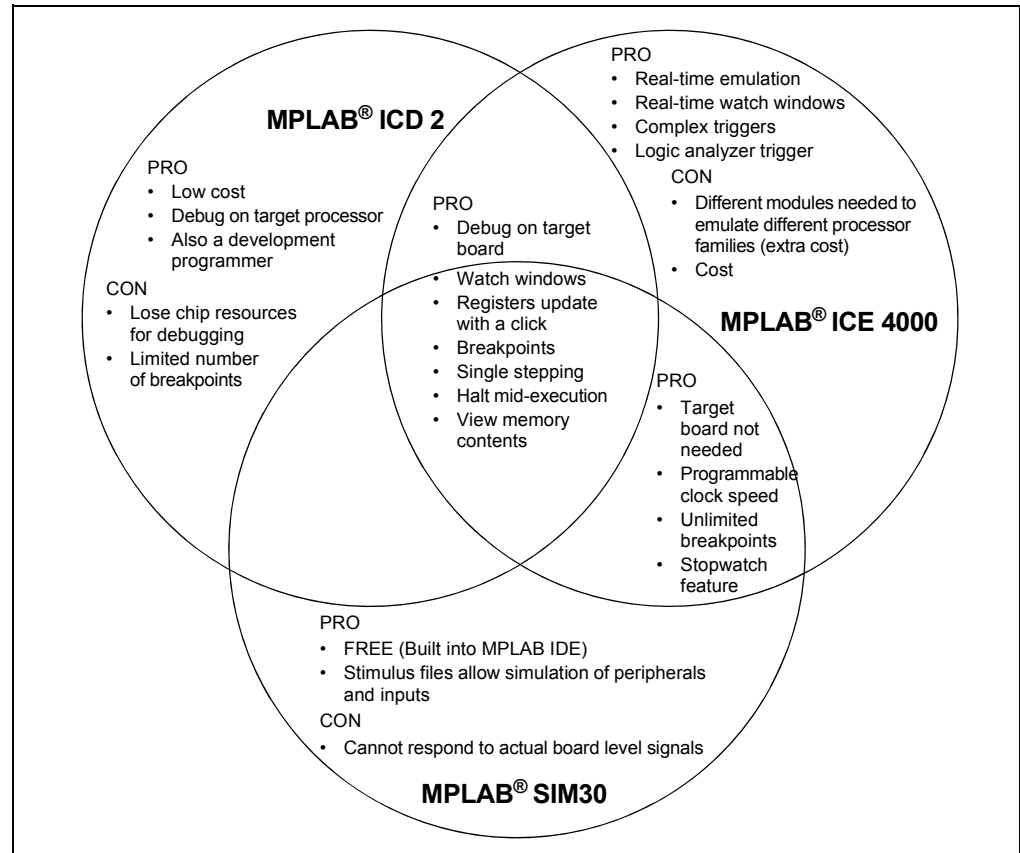


2.4.3 MPLAB ICD 2

The MPLAB ICD 2 In-Circuit Debugger is a very cost-effective debugging tool that allows code to be tested on the target circuit board. For those who don't want the added costs associated with an MPLAB ICE 4000 and can do without its sophisticated features, the MPLAB ICD 2 is a viable alternative. The MPLAB ICD 2 allows you to debug dsPIC devices directly in your target board. You can also use it to program devices in-circuit.

Although MPLAB ICD 2 provides basic debugging functions, it lacks such features of the MPLAB ICE 4000 as trace memory and complex triggers. Similarly, the free MPLAB SIM30 Software Simulator allows you to debug your code but lacks features included in MPLAB ICD 2. Figure 2-7 is a summary comparison of these three tools.

FIGURE 2-7: COMPARISON OF MPLAB® TOOLS



2.5 PROGRAMMING TOOLS

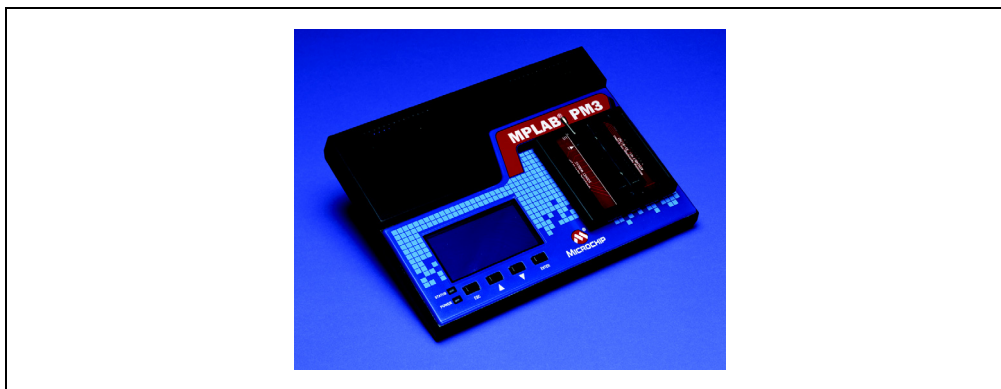
Two programmers can be used with MPLAB IDE to program dsPIC devices: MPLAB PM3 and MPLAB ICD 2. Each has its own particular advantages and disadvantages.

MPLAB PM3 can program all package types and has more programming options and memory than the MPLAB ICD 2. However, both can program parts in-circuit. The MPLAB ICD 2 is an in-circuit debugger that can also program parts in-circuit.

Note: As a general rule, the MPLAB PM3 is your best choice for production programming. The MPLAB ICD 2 is your best choice for testing code during development if the boards support in-circuit programming.

The older PRO MATE II programmer also supports the dsPIC devices but has been superseded by the newer MPLAB PM3.

FIGURE 2-8: MPLAB® PM3 UNIVERSAL DEVICE PROGRAMMER



2.5.1 MPLAB PM3 Universal Device Programmer

The MPLAB PM3 (Figure 2-8) is the preferred choice for those wanting to purchase a production programmer. It consists of a basic programmer unit and interchangeable socket modules to support various device packages. It can be controlled from MPLAB IDE, from a command-line utility, or it can operate stand-alone. MPLAB PM3 includes the following features:

- Built-in support for In-Circuit Serial Programming.
- Serialized programming for unique ID numbers.
- Safe mode for code security.
- High-speed programming and download through USB.
- Secure digital and multimedia card slot for convenient program storage.

2.5.2 MPLAB ICD 2

In addition to being an in-circuit debugger, the MPLAB ICD 2 can also be used as a low-cost development programmer. You can use it to program parts in-circuit, directly on your target board, as well as DIP packages out of circuit with our universal programming module.

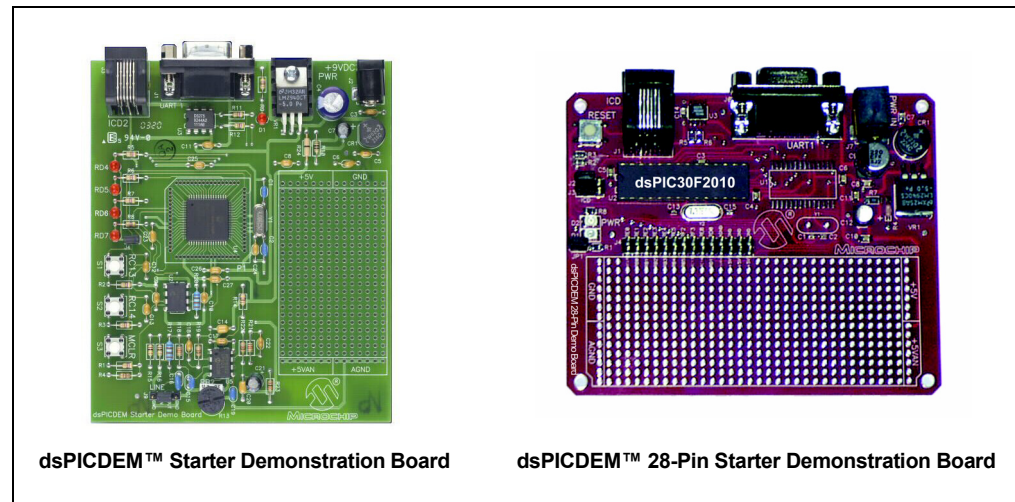
2.6 DEVELOPMENT BOARDS

Several dsPIC development boards are available to simplify code development and testing. These boards are very useful for new users getting started with the dsPIC processors because they include example code and tutorials.

Note: The example code in this guide is provided in four versions to run on the following development boards:

- dsPICDEM Starter Demonstration Board
- dsPICDEM 28-Pin Starter Demonstration Board
- dsPICDEM 1.1 General Purpose Development Board
- dsPICDEM 2 Development Board

FIGURE 2-9: dsPICDEM™ STARTER DEMONSTRATION BOARDS



2.6.1 dsPICDEM Starter Demonstration Board

The dsPICDEM Starter Demonstration Board (on the left in Figure 2-9) is a low-cost demo board that uses a dsPIC30F6012 processor and has a connector for programming and debugging with the MPLAB ICD 2. It includes the following features:

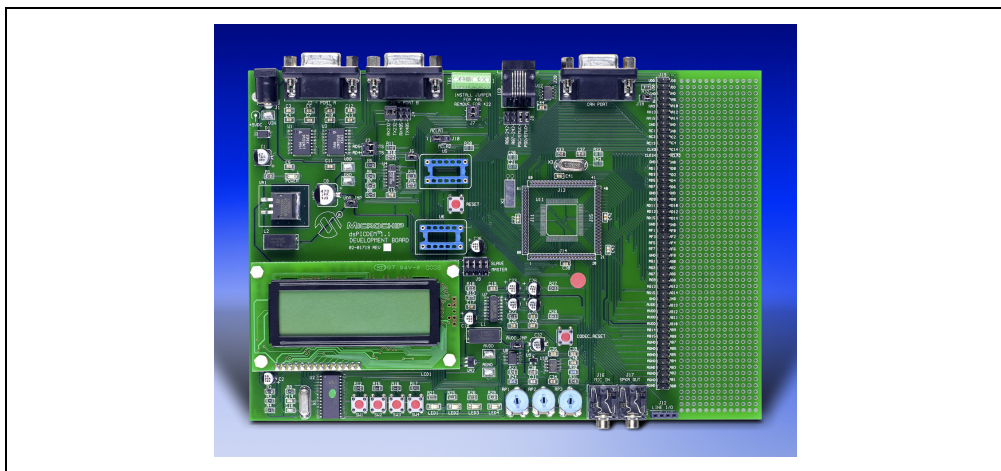
- RS-232 interface for use with the UART
- Switches and LEDs for I/O
- Analog output controlled via a digital pot
- Analog input from potentiometer
- Buffered external analog input
- Wire wrap area for prototyping

2.6.2 dsPICDEM 28-Pin Starter Demonstration Board

The dsPICDEM 28-Pin Starter Demonstration Board (on the right in Figure 2-9) is another low-cost demo board for the dsPIC devices. This board uses a dsPIC30F2010 28-pin processor and has a connector for programming and debugging with the MPLAB ICD 2. It also has the following features:

- RS-232 interface
- One LED
- Header for access to all device I/O pins
- 28-pin DIP socket and layout pad for 28-pin SOIC device
- Wire wrap area for prototyping

FIGURE 2-10: dsPICDEM™ 1.1 GENERAL PURPOSE DEVELOPMENT BOARD



2.6.3 dsPICDEM 1.1 General Purpose Development Board

The dsPICDEM 1.1 General Purpose Development Board (Figure 2-10) is a relatively sophisticated general purpose board that uses a dsPIC30F6014 processor. It includes the following features in addition to most of the features of the dsPICDEM Starter Demonstration Board:

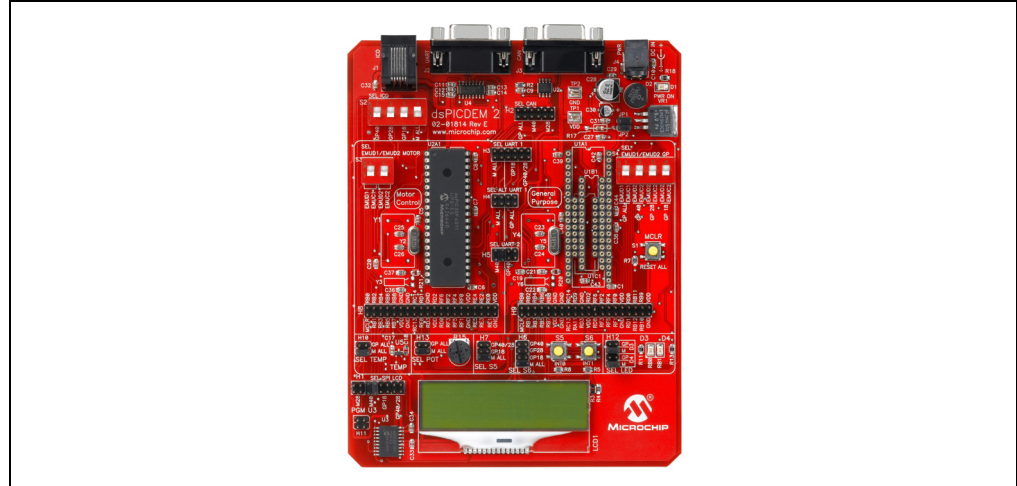
- Graphic and text LCD display module
- Header pins for MPLAB ICE 4000 device adapter
- Controller Area Network (CAN) interface
- RS-232, RS-422 and RS-485 UART interface
- CODEC analog input and output for use with the DCI interface

2.6.4 dsPICDEM 2 Development Board

The dsPICDEM 2 Development Board (Figure 2-11) is a multipurpose development board that helps you create embedded applications using dsPIC30F Digital Signal Controllers in 18, 28 and 40-pin PDIP and SPDIP packages. Key features of the dsPICDEM 2 Development Board include:

- dsPIC30F4011 40-pin PDIP sample device
- Multiple sockets for 18, 28 and 40-pin DIP devices
- Connector for MPLAB ICD 2 In-Circuit Debugger
- RS-232 interface
- CAN interface
- Temperature sensor, analog potentiometer and push button switches to simulate A/D inputs
- LCD screen and LED indicators

FIGURE 2-11: dsPICDEM™ 2 DEVELOPMENT BOARD



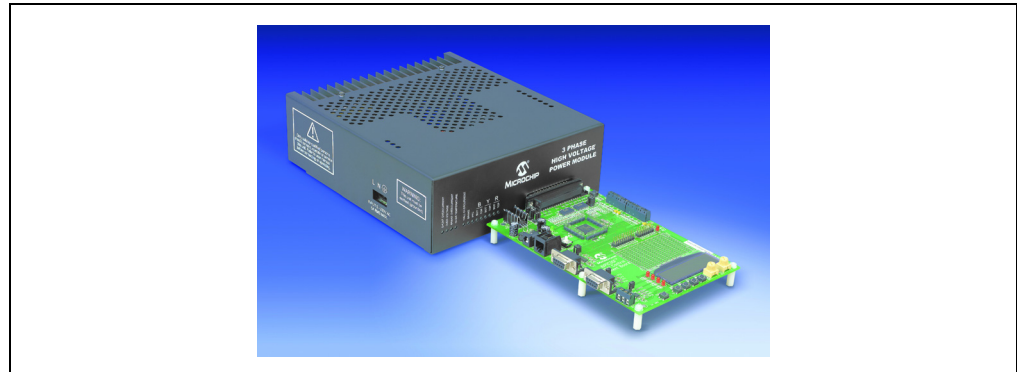
2.6.5 dsPICDEM MC1 Motor Control Development System

The dsPICDEM MC1 Motor Control Development System provides the application developer with three main components for quick prototyping and validation of Brushless DC Motor (BLDC), Permanent Magnet Alternating Current (PMAC) and AC Induction Motor (ACIM) applications. The three main components are:

- dsPICDEM MC1 Motor Control Development System
- dsPICDEM MC1L 3-Phase Low-Voltage Power Module
- dsPICDEM MC1H 3-Phase High-Voltage Power Module

The dsPICDEM MC1 Motor Control Development System contains a dsPIC30F6010 and supports a custom interface header, which allows different motor power modules to be connected to the PCB. The control board also has connectors for mechanical position sensors, such as incremental rotary encoders and hall effect sensors, and a breadboard area for custom circuits.

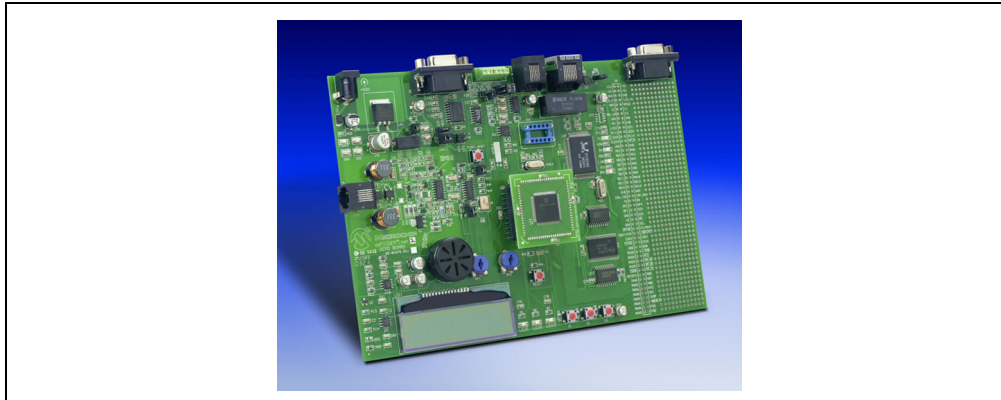
FIGURE 2-12: dsPICDEM™ MC1 MOTOR CONTROL DEVELOPMENT SYSTEM



The dsPICDEM MC1L 3-Phase Low-Voltage Power Module is optimized for 3-phase motor applications that require a DC bus voltage less than 50 volts and can deliver up to 400W power output. The low-voltage module is intended to power BLDC and PMAC motors.

The dsPICDEM MC1H 3-Phase High-Voltage Power Module is optimized for 3-phase motor applications that require DC bus voltages up to 400 volts and can deliver up to 1 kW power output. The high-voltage module has an active power factor correction circuit that is controlled by the dsPIC30F device. This power module is intended for AC induction motor and power inverter applications that operate directly from AC line voltage.

FIGURE 2-13: dsPICDEM.net™ CONNECTIVITY DEVELOPMENT BOARDS



2.6.6 dsPICDEM.net 1 and dsPICDEM.net 2 Connectivity Development Boards

The dsPICDEM.net 1 and dsPICDEM.net 2 Connectivity Development Boards provide a basic platform for developing and evaluating various connectivity solutions and implementing TCP/IP protocol layers combined with V.22bis/V.22 ITU specifications across PSTN or Ethernet communication channels. The board comes with an ITU-T compliant V.22bis/V.22 modem demonstration program loaded on the installed dsPIC30F6014 device. This program enables you to connect and transfer data between the dsPIC Soft Modem and an ITU-T compliant reference modem. Control of the dsPIC Soft Modem is supported via AT commands communicated using the on-chip UART channel. Also included are CMX-MicroNet™ Web and FTP Server demonstration files that, when downloaded into the dsPIC30F6014 device, demonstrate two TCP/IP stack-based applications over the Ethernet data link layer.

Both dsPICDEM.net 1 and 2 support the dsPIC30F5013 and dsPIC30F6014 devices and have Ethernet and PSTN interfaces. The dsPICDEM.net 1 supports FCC/JATE PSTN and the dsPICDEM.net 2 supports CTR-21 PSTN.

2.6.7 Next Step – Learn to Use MPLAB IDE

Now that you have learned about the development tools available for the dsPIC devices, it's time to start using the MPLAB Integrated Development Environment (IDE).

You must learn to use the MPLAB IDE before you can compile, program or debug, so please proceed to **Chapter 3. “MPLAB Integrated Development Environment”**.

Chapter 3. MPLAB Integrated Development Environment

3.1 MPLAB IDE OVERVIEW

Now that you've been introduced to the dsPIC30F and its development tools, you're probably itching to write some code. As discussed in **Chapter 2. "The Microchip Development Tools"**, the MPLAB IDE software is used throughout the whole code development process: for writing, compiling, debugging and programming. It has the following main features:

- Project Manager – for organizing code files
- Editor – for typing code
- Assembler and Linker – for assembling and building code
- Compiler Interface – for compiling code with separate compilers
- Simulator – for testing code operation
- Debugger/Emulator Interface – for testing code with a separate debugger or emulator
- Programmer Interface – for programming parts with a separate programmer

Instead of wasting time with a dry, boring discussion of these features, let's do a quick tutorial. Learning by doing is always effective.

First, install and run the latest MPLAB IDE software. No kidding – this step is important. The software is updated quite regularly to add new features and support for the latest devices. The latest version of MPLAB IDE can be obtained from the Microchip Technology web site (www.microchip.com).

The source code files for this tutorial are available with this document on the Microchip Technology web site or on CD. We will use the `Flash LEDs with dsPIC30F6014.s` file for the tutorial. You will not need any hardware, but later, you will be able to run the code on a dsPIC30F6014 using the dsPICDEM 1.1 General Purpose Development Board.

The Project Wizard in MPLAB IDE is a great way to create new projects, making it a very simple process. Before starting, create a folder called `C:\Tutorial` and copy the `Flash LEDs with dsPIC30F6014.s` file into the folder. If the files are copied from a CD, they have read-only attributes. Remember to change the attributes if the file needs to be edited.

Note: If you have a dsPICDEM Starter Demonstration Board, you can use the `Flash LEDs with dsPIC30F6012.s` file instead. For the dsPICDEM 28-Pin Starter Demonstration Board, you can use the `Flash LED with dsPIC30F2010.s` file and for the dsPICDEM 2 Development Board, you can use the `Flash LED with dsPIC30F4011.s` file. These files contain very similar code and functionality.

3.2 PROJECTS AND WORKSPACES

Generally, everything in MPLAB IDE is done within a project.

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

A workspace contains one or more projects, information on the selected device, debug tool and/or programmer, open windows and their location, and other IDE configuration settings. Usually, you will have one project in one workspace. (This can be changed by going to the *Configure>Settings* menu.)

3.3 CREATING A PROJECT

3.3.1 Projects and Workspaces

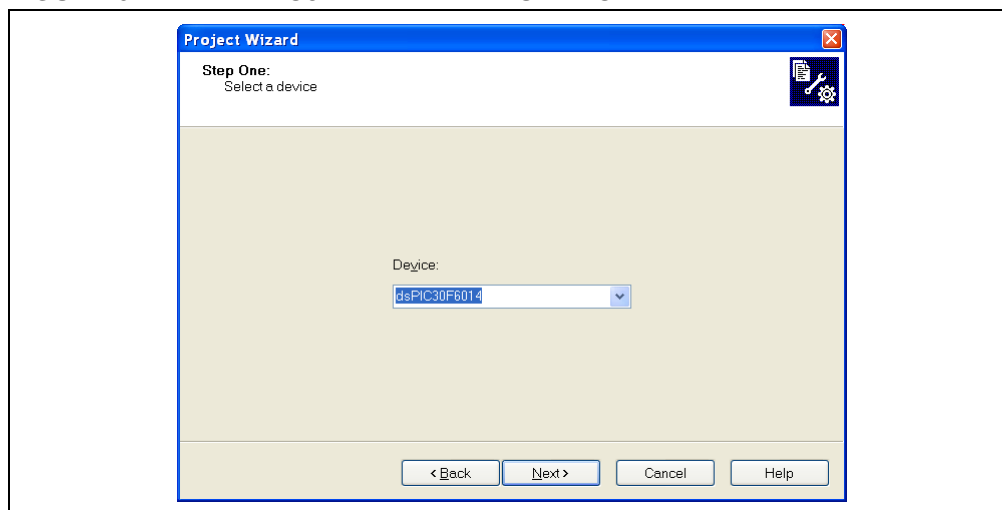
Start MPLAB IDE and close any open workspace with the *File>Close Workspace* menu. Then, use the *Project>Project Wizard* menu to start the Project Wizard. When the Welcome screen appears, click **Next** to continue.

Step 1 – Select a Device

The next screen (Figure 3-1) allows you to choose the part. Select “dsPIC30F6014” from the pull-down menu.

Note: If you are using one of the other dsPIC development boards, you'll need to pick the appropriate device for that board. For the dsPICDEM Starter Demonstration Board, select the dsPIC30F6012. For the dsPICDEM 28-Pin Starter Demonstration Board, select the dsPIC30F2010. For the dsPICDEM 2 Development Board, select the dsPIC30F4011.

FIGURE 3-1: PROJECT WIZARD – STEP ONE

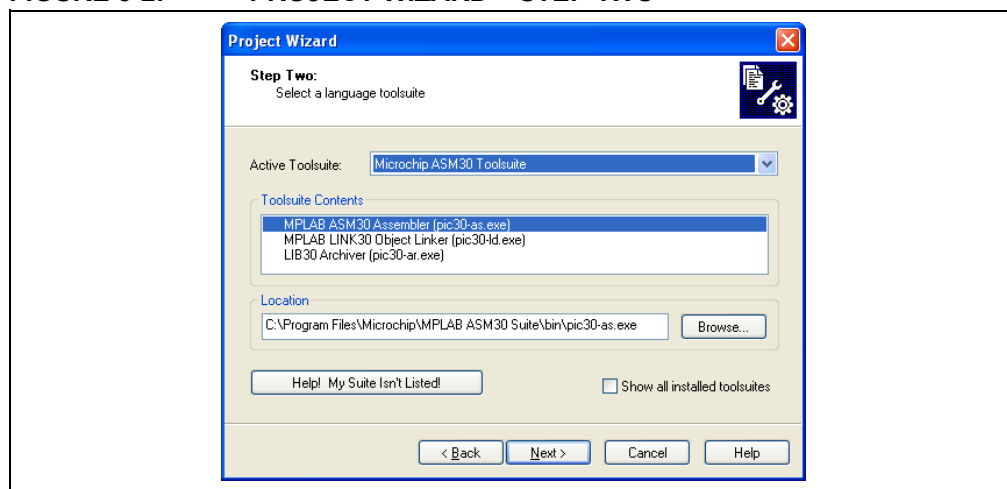


Click **Next** to continue.

Step 2 – Select a Language Toolsuite

The next screen (Figure 3-2) allows you to select the toolsuite. Select the “Microchip ASM30 Toolsuite” from the pull-down menu.

FIGURE 3-2: PROJECT WIZARD – STEP TWO



Check that the executables for the assembler and linker are at these locations:

Assembler:

C:\Program Files\Microchip\MPLAB ASM30 Suite\bin\pic30-as.exe

Linker:

C:\Program Files\Microchip\MPLAB ASM30 Suite\bin\pic30-ld.exe

Archiver:

C:\Program Files\Microchip\MPLAB ASM30 Suite\bin\pic30-ar.exe

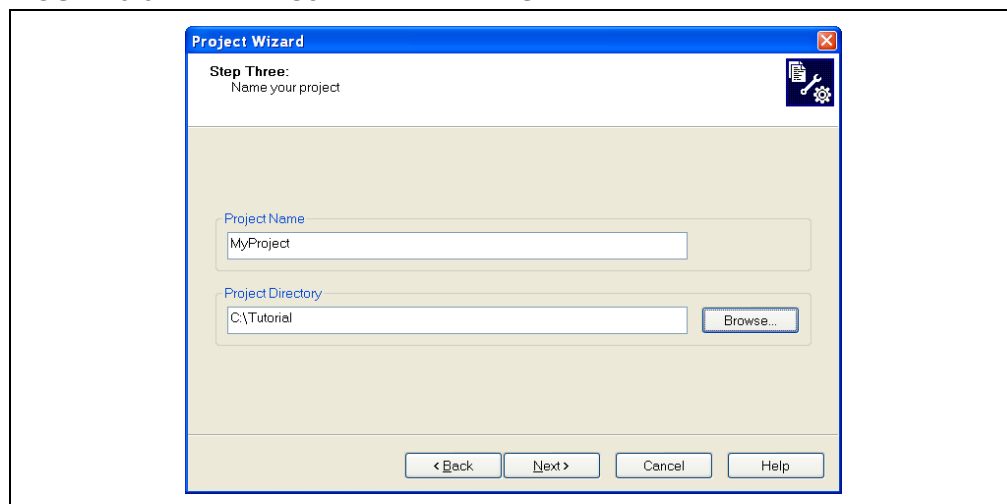
These tool locations assume that the latest version of MPLAB IDE was installed with the default settings.

A red 'X' appears next to toolsuits whose locations are blank. If there is a red 'X', then select the toolsuite and click on the **Browse** button to set the location. Once the toolsuite has been selected and the locations are correct, click **Next** to continue.

Step 3 – Name the Project

The next screen (Figure 3-3) allows you to name the project.

FIGURE 3-3: PROJECT WIZARD – STEP THREE



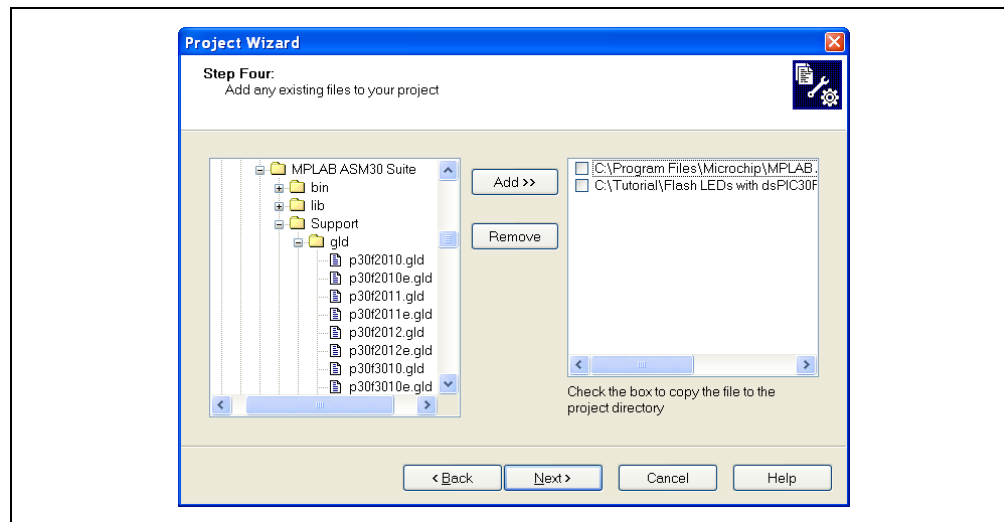
Type “MyProject” for the project name and browse to, or type C:\Tutorial for the project directory.

Click **Next** to continue.

Step 4 – Add Files to the Project

The next screen (Figure 3-4) allows you to add files to the project.

FIGURE 3-4: PROJECT WIZARD – STEP FOUR



Select the Flash LEDs with dsPIC30F6014.s file and click **Add>>** to include the file in the project.

Navigate to the C:\Program Files\Microchip\MPLAB ASM30 Suite\Support\gld folder. Select the p30f6014.gld file and click **Add>>** to include the file in the project. There should now be two files in the project.

Click **Next** to continue. When the summary screen appears, click **Finish**.

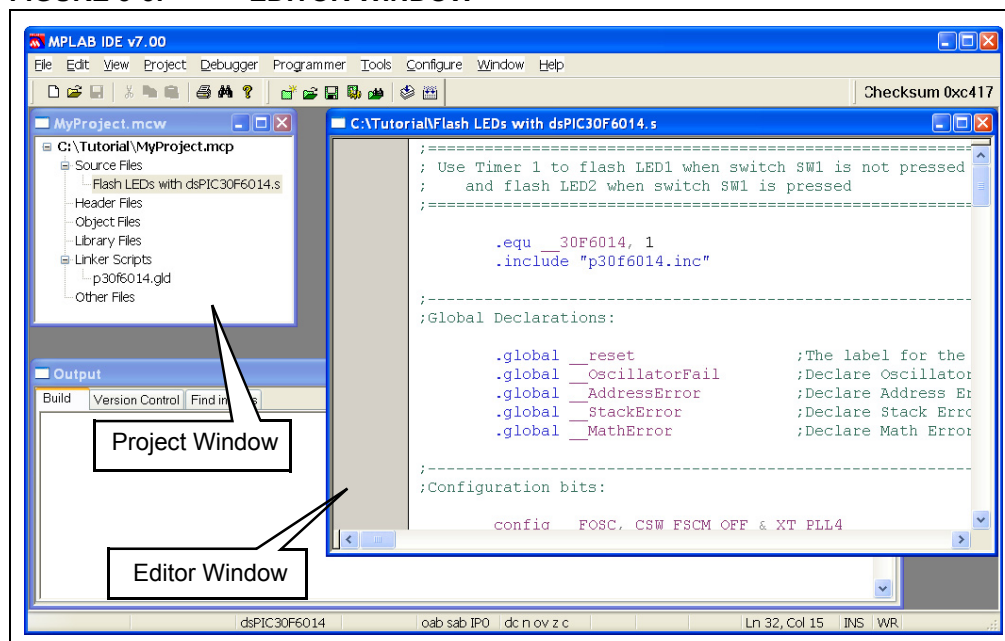
Note: For the dsPICDEM Starter Demonstration Board, select the Flash LEDs with dsPIC30F6012.s and p30f6012.gld files. For the dsPICDEM 28-Pin Starter Demonstration Board, select the Flash LED with dsPIC30F2010.s and p30f2010.gld files. For the dsPICDEM 2 Development Board, select the Flash LED with dsPIC30F4011.s and p30f4011.gld files.

After the Project Wizard completes, the MPLAB IDE Project window will show the Flash LEDs with dsPIC30F6104.s file in the Source Files category and the p30f6014.gld file in the Linker Scripts category. The .gld file is described in much greater depth in **Chapter 9. “The MPLAB LINK30 Linker”**.

If you realize that you have forgotten to add files to your project, you don’t have to restart the Project Wizard. Simply right click on a category in the project tree, select Add Files from the drop-down menu and browse to the file you want to add. You can remove files by right clicking on the file name and selecting **Remove**.

A project file, MyProject.mcp, and workspace file, MyProject.mcw, have now been created by the MPLAB IDE (see Figure 3-5). Double click the Flash LEDs with dsPIC30F6014.s file in the Project window to open the file. The file displays in the Editor window.

FIGURE 3-5: EDITOR WINDOW



3.3.2 Editor

There are several features provided in the editor in MPLAB IDE that makes writing code a much smoother experience. These features include:

- Syntax highlighting
- View and print line numbers
- Search through all project files or within a single file
- Bookmark and jump to specific lines
- Double click on error message to go to the line of code
- Block commenting
- Brace matching
- Variable font and font size

Syntax highlighting is an especially useful feature of MPLAB IDE. Code elements, such as instructions, directives, registers, etc., appear in different colors and fonts. This allows you to easily interpret your code and notice mistakes more quickly.

3.4 BUILDING CODE

3.4.1 Assembling and Linking

Building a project consists of two steps. The first is the Assembly or Compile process, where each source file is converted into an object file (.o extension), containing opcodes or dsPIC instructions. These object files can be used to form libraries, which are added to other projects as code modules, or to generate the final hex file, which is used to program the dsPIC device.

The second step in the building process is Linking. During the link stage, all of the dsPIC instructions and variables from the various object and library files are placed in memory according to the memory map provided by the linker script file.

The linker creates two files:

1. The .hex file, which is a listing of the data to be placed in the dsPIC device's program, EEPROM and configuration memory.
2. The .cof, or COFF (Coded Object File Format) file, contains additional information that is necessary to debug your source code.

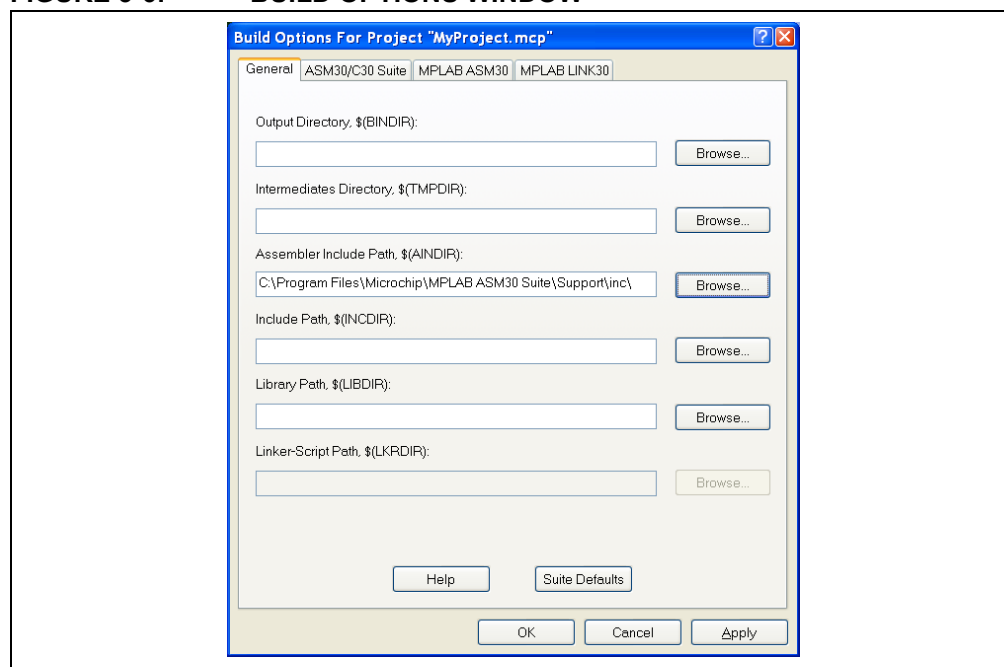
3.4.2 Include Files

Before building, you must tell MPLAB IDE where to find the include files. Near the top of the Flash LEDs with dsPIC30F6014.s file, you will see the line:

```
.include "p30f6014.inc"
```

The p30f6014.inc file contains symbolic information that is needed to refer to Special Function Register bits by name rather than fairly meaningless numbers. To let MPLAB IDE know where to find this file, use the *Project>Build Options>Project* menu to display the Build Options window, as shown in Figure 3-6. Then, click the **Browse** button next to the "Assembler Include Path, \$(AINDIR):" field.

FIGURE 3-6: BUILD OPTIONS WINDOW



Browse to the C:\Program Files\Microchip\MPLAB ASM30 Suite\Support\inc folder and click **Select**. This directory is where MPLAB IDE keeps the include files for all the dsPIC devices that it supports.

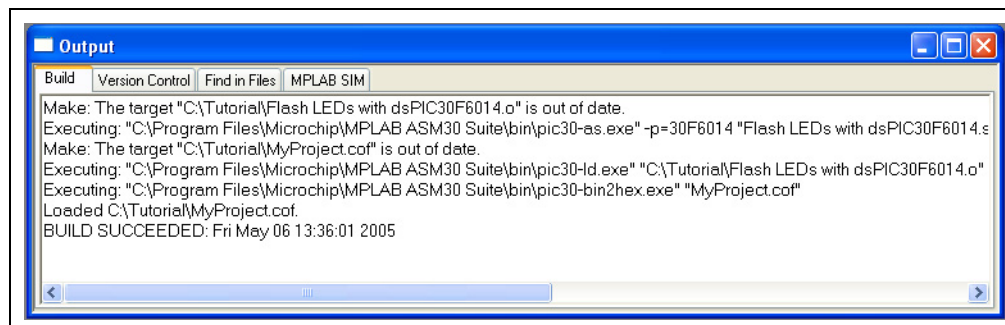
Finally, click **OK** to save the information. The project is now ready to be built.

3.4.3 Building the Project

To build the project, use the *Project>Make* menu. The results of the build will appear in the Output window and should indicate that the build succeeded, as shown in Figure 3-7.

Depending on your Project Build Options, you may see a Memory Usage Report in your Build results.

FIGURE 3-7: OUTPUT WINDOW



3.4.4 Configuration Bits

The code contains configuration bit settings, specified with configuration directives. You can see the settings in the Configuration Bits window using the *Configure>Configuration Bits* menu. These settings can be changed. You can edit the settings by clicking on the text in the Setting column.

3.4.5 Next Step – Debugging

Now that you have built the project successfully, it's time to debug the code. There are several tools you can use. If you want to debug using the simulator, then please continue on to **Chapter 4. “The MPLAB SIM Simulator”** for a tutorial on the simulator. If you wish to use the In-Circuit Debugger (MPLAB ICD 2), then skip ahead to **Chapter 5. “The MPLAB ICD 2 In-Circuit Debugger”**. To use the MPLAB ICE 4000 In-Circuit Emulator, go to **Chapter 6. “MPLAB ICE 4000 In-Circuit Emulator”**.

NOTES:

Chapter 4. The MPLAB SIM Simulator

4.1 MPLAB SIM OVERVIEW

So, you want to test your code but don't want to bother setting up any hardware? Then the MPLAB SIM Simulator is for you. MPLAB SIM Simulator is fully integrated into the MPLAB IDE environment. It is capable of mimicking your code execution on hardware without the need for expensive overhead. You can test external inputs, peripheral transactions and see internal signals on your processor without having to spend any money.

There are limitations to the MPLAB SIM Simulator. The simulator is not capable of reacting to or producing any real world signals. It can't beep buzzers, blink LEDs or interact with other processors. Still, it gives you much flexibility in developing your code and working out its bugs.

The MPLAB SIM Simulator allows you to:

- Modify code and immediately re-execute it
- Inject external stimuli into the simulated processor
- Load register values at specified times

The dsPIC devices have I/O pins multiplexed with other peripherals (and therefore, referred to by more than one name). The simulator recognizes the pin names specified in the standard device headers as valid I/O pins. Therefore, you should refer to the header file for your device (e.g., `p30F6014.inc` or `p30F6014.h`) to determine the correct pin names.

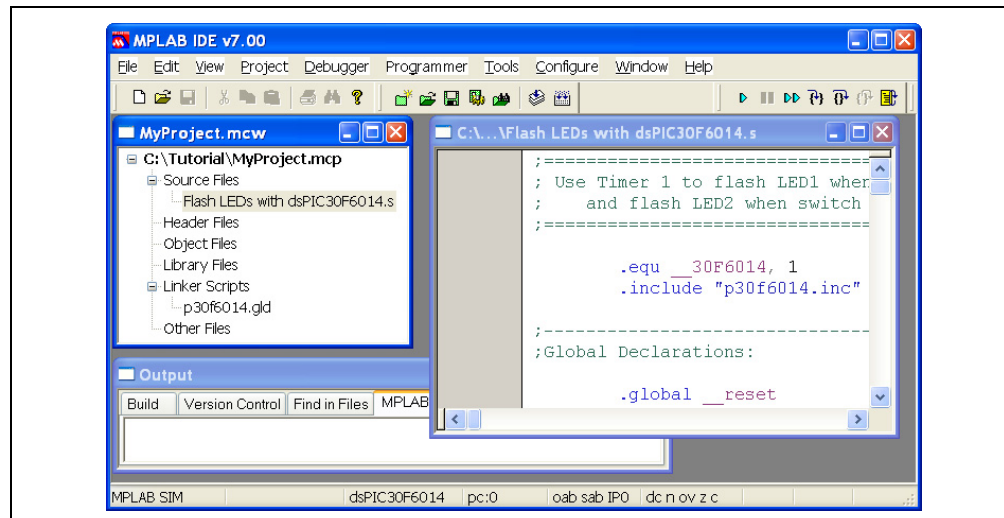
This chapter discusses how to use the simulator. First, you'll open the project that you created in **Section 3.3 "Creating a Project"**. If you haven't yet created the project, please refer to that section now. You will use the simulator to step through the code, create breakpoints, use the stopwatch feature and apply stimulus.

<p>Note: If you created the project for the dsPICDEM Starter Demonstration Board, the dsPICDEM 28-Pin Starter Demonstration Board or the dsPICDEM 2 Development Board, you can use your project and still follow along with this section. The code is very similar.</p>
--

4.2 OPENING THE PROJECT

If it is not already open, open the workspace created in **Section 3.3 “Creating a Project”** by selecting **File>Open Workspace** and browsing to `C:\Tutorial\MyProject.mcw`. The workspace name should be visible in the title bar of the Project window and the name of the project should be visible inside the Project window at the top. Build the project with the **Project>Make** menu to ensure that it is up to date.

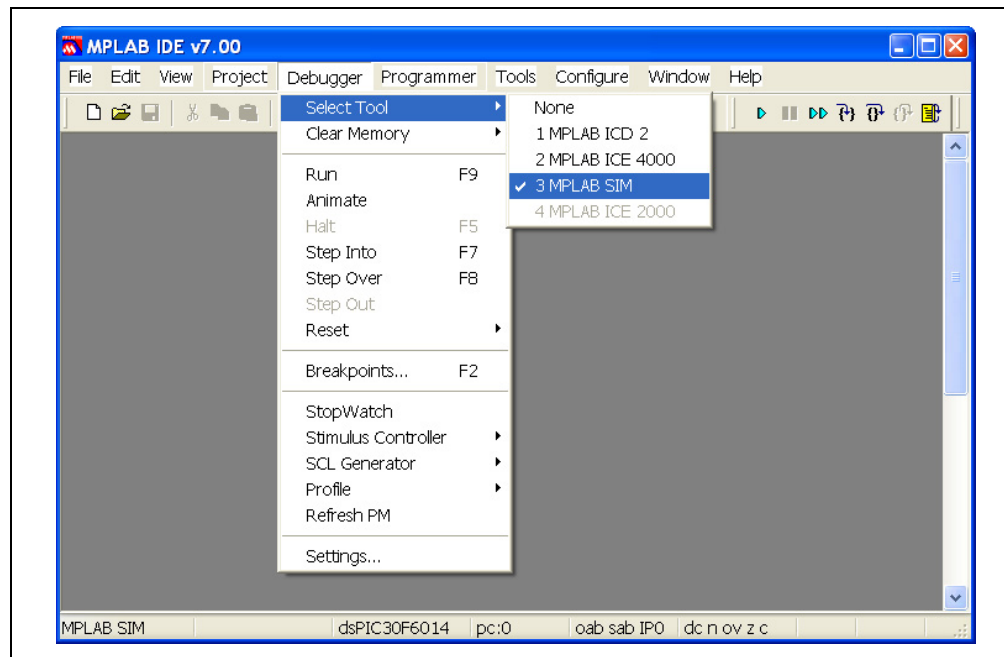
FIGURE 4-1: PROJECT WORKSPACE



4.3 SELECTING THE SIMULATOR

Use the **Debugger>Select Tool>MPLAB SIM** menu to select the MPLAB SIM Simulator, as shown in Figure 4-2. When you do this, simulator operations are added to menus and tool bars. The standard set of debugging options, plus Stopwatch, Stimulus Controller and SCL Generator are added to the Debugger menu in the MPLAB IDE.

FIGURE 4-2: DEBUG TOOL SELECTION MENU



4.4 RESETTING THE CODE

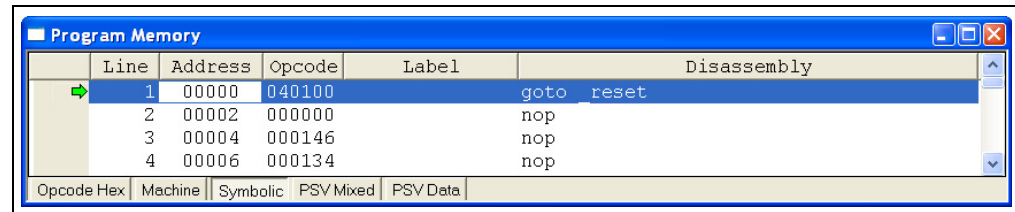
When you select MPLAB SIM, the Program Counter is automatically set to zero, the Reset vector. You can simulate a Power-on Reset with the *Debugger>Reset>Processor Reset* menu. The text "pc:0" appears in the Status bar at the bottom of the MPLAB IDE screen, showing that the Program Counter is zero.

There are four types of Reset, selectable by the Debugger menu:

- MCLR Reset
- Watchdog Timer Reset
- Brown-out Reset
- Processor (Power-on) Reset

Open the Program Memory window (Figure 4-3) with the *View>Program Memory* menu and click on the **Symbolic** tab at the bottom of the window. A green arrow in the gutter points to the first line containing the code at address zero.

FIGURE 4-3: PROGRAM MEMORY WINDOW



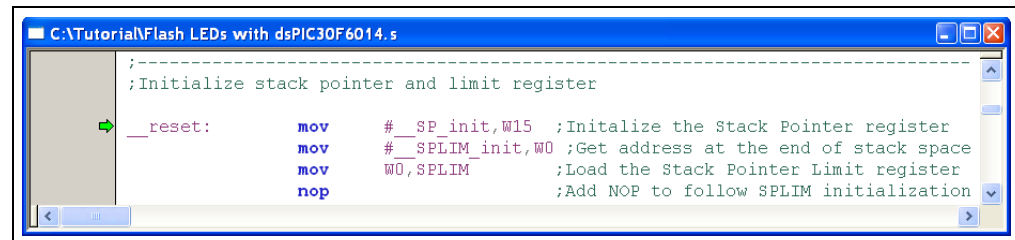
Address zero contains a `goto __reset` instruction that is added automatically by the linker and is not in the source code file.

Close the Program Memory window at this point because all the other instructions are in the source code.

4.5 STEPPING THROUGH THE CODE

You can now step through the source code in the simulator. Use the *Debugger>Step Into* menu to single step the `goto __reset` instruction. The green arrow now points to the first line of executable code in the Flash LEDs with `dsPIC30F6014.s` file.

FIGURE 4-4: STEPPING THROUGH THE CODE



You can use the *Debugger>Step Into* menu to keep stepping through the code. There are several different methods of stepping through your code, all under the *Debugger* menu: Step Into, Step Over and Animate.

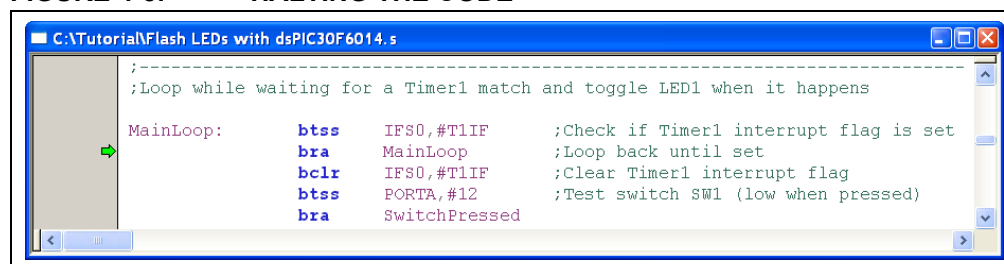
- Step Into executes the current instruction and then halts. If the current instruction is a call to a subroutine, the Program Counter changes to the start of the called function.
- Step Over executes all code up to the next Program Counter location. It is just like the Step Into feature for most instructions. However, if the instruction is a `CALL`, it executes the called subroutine in its entirety and then returns.
- Animate steps continuously through your code. It is equivalent to doing many Step Into operations repetitively until you select Halt.

4.6 RUNNING THE CODE

Select *Debugger>Run* to run your application. The word, “Running...”, with a small moving bar appears in the Status bar. Nothing seems to be happening except for the moving bar. The green arrow in the gutter becomes transparent since the Program Counter is changing and the current execution point cannot be shown.

Select *Debugger>Halt* to stop the program execution. MPLAB IDE updates its windows after the Halt and shows the current execution point with the green arrow (Figure 4-5).

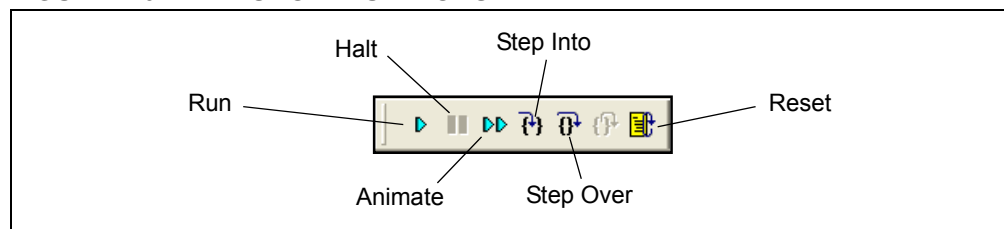
FIGURE 4-5: HALTING THE CODE



4.7 THE DEBUG TOOLBAR AND HOTKEYS

The Debug toolbar provides shortcut icons to control the simulator, as shown in Figure 4-6.

FIGURE 4-6: SHORTCUT ICONS



When you select the MPLAB SIM Simulator, the Debug toolbar automatically opens. You can drag this toolbar anywhere on the desktop for convenience. Click on the appropriate toolbar icon to Run, Halt, Animate, Step Into, Step Over or Reset the program.

MPLAB SIM also uses the following function keys to access the main debugging functions:

- <F5> Halt
- <F6> Reset
- <F7> Single Step
- <F9> Run

Additional functions are available by right clicking on a line of source code. The most important of these are Set Breakpoint and Run to Cursor.

If you do not see the toolbar, you can enable it with the *View>Toolbars>Debug* menu.

4.8 BREAKPOINTS

The simulator gives you the ability to set breakpoints – places in the code where execution will be halted. You can set breakpoints directly in your Source Code window, in the Program Memory window, or in the Disassembly window.

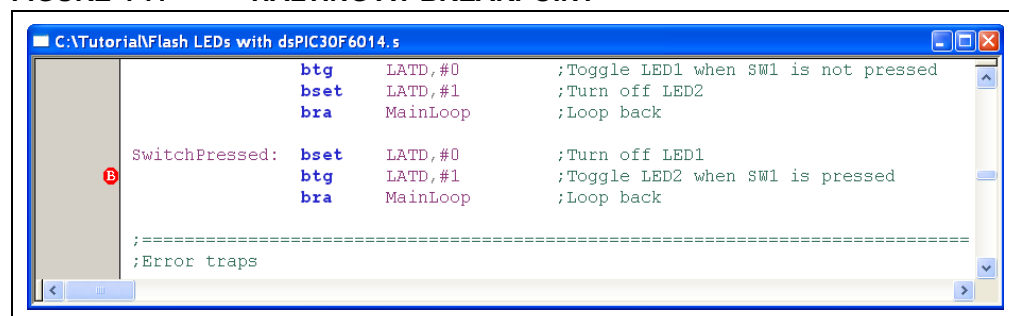
In this example, you'll set a breakpoint where an LED is lit by toggling a bit in PORTD. Scroll down to line 100 and find the `btg LATD, #1` instruction. Note that the line number is shown in the Status bar.

Note: If you are using the Flash LED with `dsPIC30F6012.s` file, set the breakpoint on line 100 on the `btg LATD, #5` instruction. If you are using the Flash LED with `dsPIC30F2010.s` file, then set the breakpoint on line 91 on the `btg LATD, #0` instruction. If you are using the Flash LED with `dsPIC30F4011.s` file, set the breakpoint on line 103 on the `btg LATB, #1` instruction.

Click on the line of code to place the cursor on the correct line, then right mouse click and select Set Breakpoint from the pop-up menu. You can also set a breakpoint by double clicking on the line (if the option has been turned on with the Edit>Properties menu).

Now press <F9> or use the Debugger>Run menu. The program will execute up to the line with the breakpoint, toggling the I/O pin. After halting at the breakpoint, the green arrow appears superimposed on the red stop sign (Figure 4-7).

FIGURE 4-7: HALTING AT BREAKPOINT

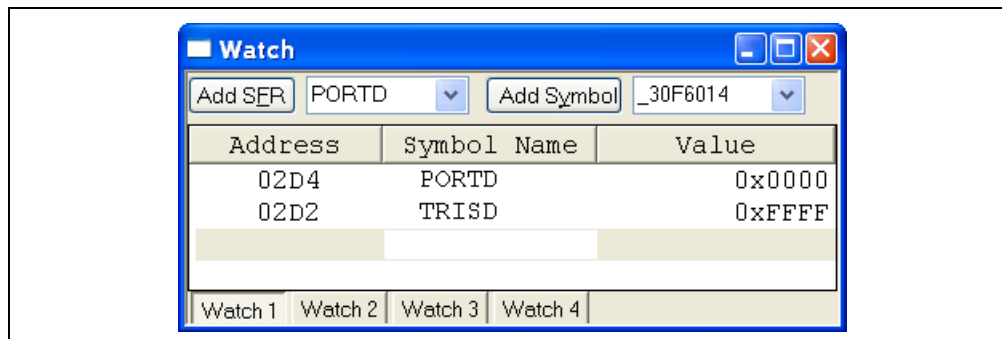


Try combinations of stepping with <F7> and running with <F9>. Notice that every time the code is made to run, it halts at the breakpoint.

4.9 WATCH WINDOW

Several options are available to view memory in the dsPIC simulator. The Watch window is one of the most useful. Use the View>Watch menu to open a Watch window (Figure 4-8).

FIGURE 4-8: WATCH WINDOW



Type “PORTD” in the **Add SFR** (Special Function Register) selection box at the top of the window. Click **Add SFR** to add it to the Watch window list. You can also type the register name directly into the Watch window; add TRISD in the same way.

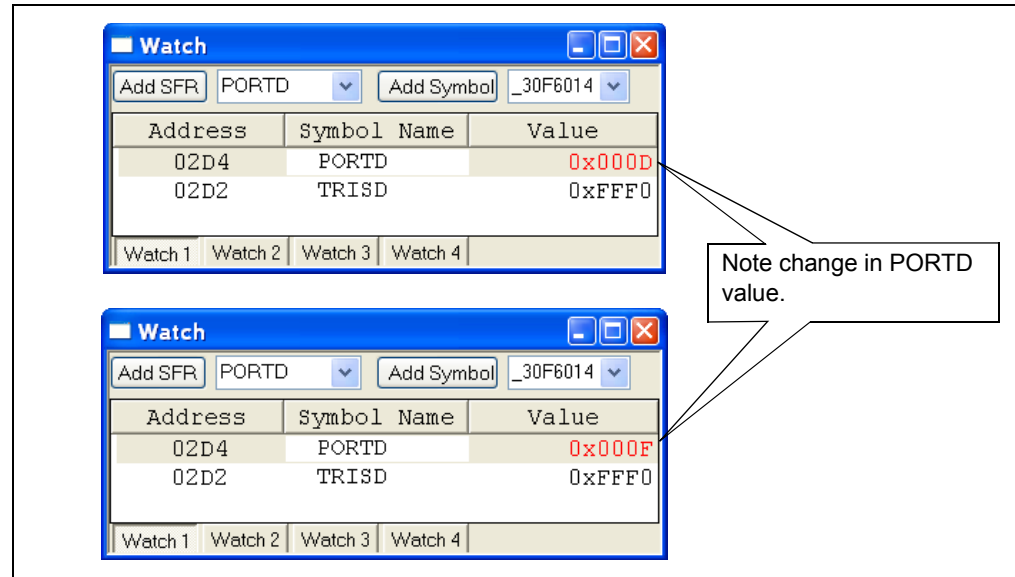
You should now have two Special Function Registers listed in the Watch window. There are columns for the “Address”, “Symbol Name” and the “Value” of the symbol.

Press <F6> to reset the code. Notice that the value of TRISD is 0xFFFF. This is the state of the TRISD register after a Reset. Press <F9> to run the code. The code should execute and halt at the breakpoint set earlier. Notice that the TRISD register has changed to 0xFF0. The code set up the I/O port directions by writing to TRISD and we can see the change in the Watch window. Notice that each time you step or halt, changed values appear in red, whereas unchanged values are black.

Note: If you are using the Flash LED with dsPIC30F6012.a file, then TRISD changes to 0xFF0F. If you are using the Flash LED with dsPIC30F2010.s file, TRISD changes to 0xFFFE. If you are using the Flash LED with dsPIC30F4011.a file, TRISB changes to 0xFFFC.

Note the state of PORTD in the Watch window and then press <F6> to run again. Each time the code runs and halts at the breakpoint, a bit in PORTD changes (see Figure 4-9). This shows the code toggling the pin to turn the LED on or off.

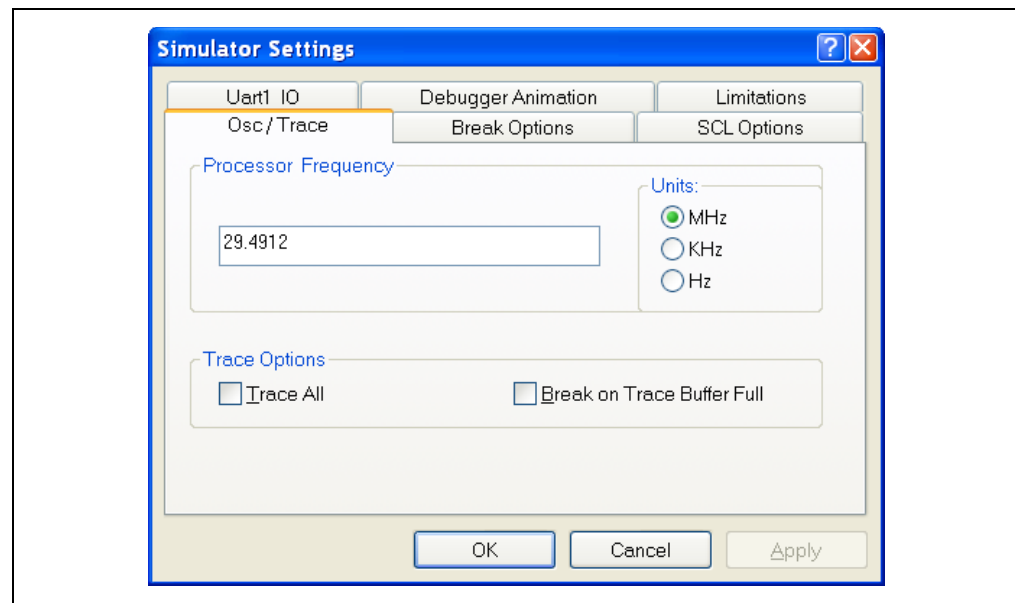
FIGURE 4-9: EXAMPLE OF PORTD BIT CHANGES



4.10 SIMULATOR SETTINGS

In the MPLAB IDE environment, use the *Debugger>Settings* menu to set up various features of the simulator. The oscillator speed must be set up in order to make timing measurements.

FIGURE 4-10: SIMULATOR SETTINGS DIALOG



Select the **Osc/Trace** tab and set the "Processor Frequency" to 29.4912 MHz. Check the **Trace All** checkbox. You'll be using the trace feature later on.

Getting Started with dsPIC30F Digital Signal Controllers

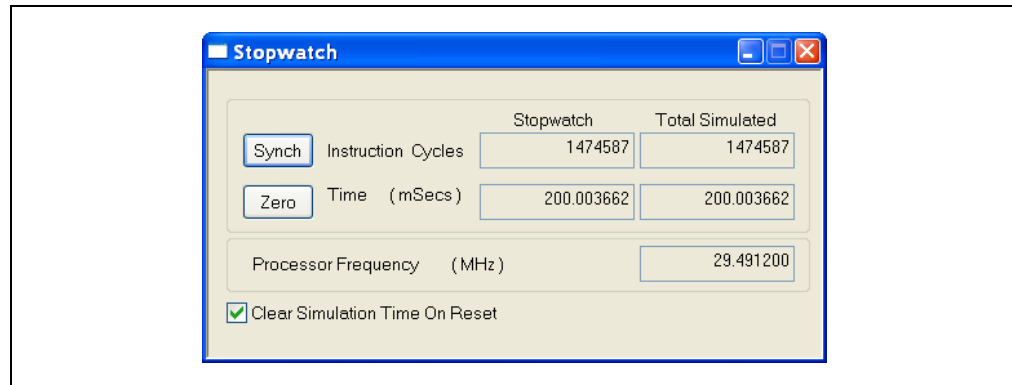
The dsPICDEM 1.1 General Purpose Development Board is shipped with a 7.3728 MHz crystal oscillator and the code selects the 4x PLL option in the configuration bits to step up the frequency by a factor of four. This means that the processor runs at 29.4912 MHz. Each instruction cycle takes four clock cycles, so the instruction cycle rate is 7.3728 MIPS.

Note: If you are using the Flash LED with dsPIC30F6012.s file, then set the processor frequency to 16 MHz instead. The other demo files use the 29.4912 MHz crystal frequency.

4.11 STOPWATCH

We can measure the execution time between two events by using the Stopwatch feature. Open the Stopwatch by clicking *Debugger>Stopwatch*. The Stopwatch keeps track of the number of instruction cycles that are executed and also shows the time that the cycles took. It calculates the time from the “Processor Frequency” that you entered earlier.

FIGURE 4-11: STOPWATCH DIALOG



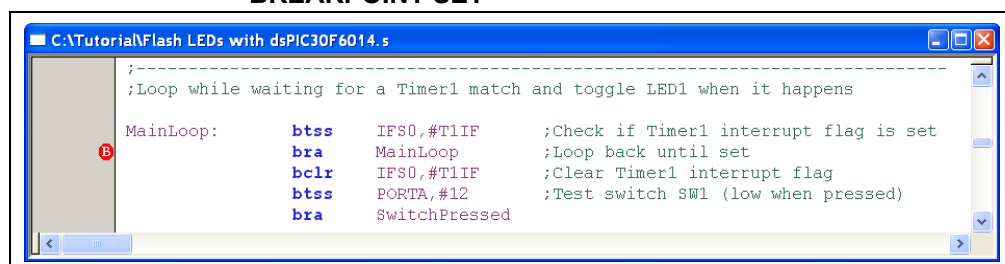
Now try a brief exercise. Press <F6> to Reset and then <F9> to Run. The execution should halt at the breakpoint that you set earlier in **Section 4.8 “Breakpoints”**. Notice that the “Stopwatch” and the “Total Simulated” columns both show 200 ms. The code sets up the Timer1 Period register for 1/5 of a second (200 ms) and the breakpoint is set where the timer period has been detected.

4.12 TRACE BUFFER

The Trace buffer is a handy feature of the MPLAB SIM Simulator. You'll find this feature under the View>ICE Trace menu.

The Trace buffer holds a list of the instructions that have executed. It can hold more than 65,000 instructions. The Trace buffer works in the background and does not need to be explicitly enabled. Simply executing code causes it to be recorded in the Trace buffer.

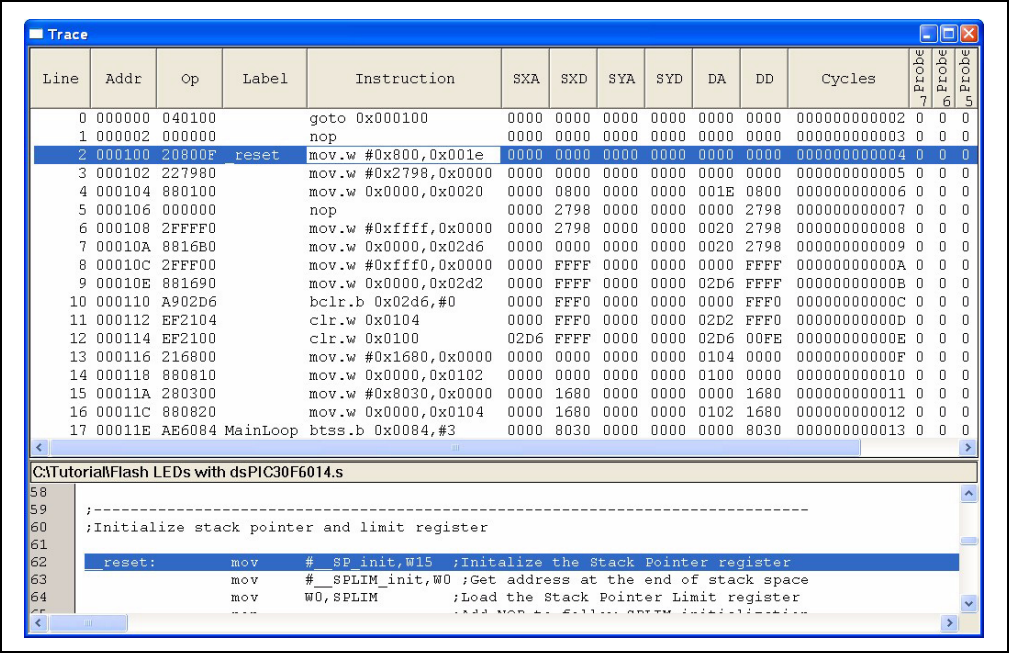
FIGURE 4-12: PROGRAM WINDOW WITH TRACE BUFFER BREAKPOINT SET



As a simple experiment, follow these steps to execute a few lines of code and view them in the Trace:

1. In the Program window, set a breakpoint on the `bra MainLoop` instruction, immediately after the `MainLoop:` address label (see Figure 4-12).
2. Press <F6> to Reset and <F9> to Run. The code will halt at the breakpoint.
3. Select View>ICE Trace to display the Trace buffer. All the instructions that were executed are shown in the Trace window, starting with the `goto __reset` instruction at address zero (see Figure 4-13). If the Trace view is blank, check what filter trace is not selected in the complex trigger setting.
4. Scroll to the very end of the Trace to see the last instruction that was executed. Notice that the Trace recorded the `btss IFS0,#T1IF` instruction immediately before the breakpoint and then stopped.
5. Compare the Trace buffer with the source code; you'll see that all the instructions up to the breakpoint are represented in the Trace window. Notice that when you select a line in the Trace view, the associated source code line is highlighted at the bottom of the window, as shown in Figure 4-13.

FIGURE 4-13: TRACE WINDOW



Chapter 5. The MPLAB ICD 2 In-Circuit Debugger

5.1 MPLAB ICD 2 OVERVIEW

The MPLAB ICD 2 is a development level programmer and in-circuit debugger. Although not as powerful as an in-circuit emulator, it has many benefits.

- The ICD 2 allows you to execute your code on the actual target chip.
- You can run at full speed or step one instruction at a time.
- You can view and modify register contents on-the-fly, as well as set breakpoints in your source code.

For the price, this tool offers excellent value.

5.1.1 Installing the USB Driver

The MPLAB ICD 2 has a USB interface. The USB drivers must be installed before it can be used. The MPLAB ICD 2 can be used with a serial port, but it is easier and much faster to use the USB interface. The instructions for installing the USB drivers can be found in your MPLAB IDE installation directory:

`C:\Program Files\Microchip\MPLAB IDE\ICD2\Drivers\ezicd2.htm`

The instruction files may be different if you are using an operating system other than Windows® XP or Windows® 2000. Look for the appropriate .htm file in the Drivers folder.

5.1.2 Opening the Project

This chapter contains a tutorial that demonstrates how to use the MPLAB ICD 2. First, you'll open the project that you created in the tutorial in **Section 3.3 "Creating a Project"**. If you have not yet created the project, please refer to that section now.

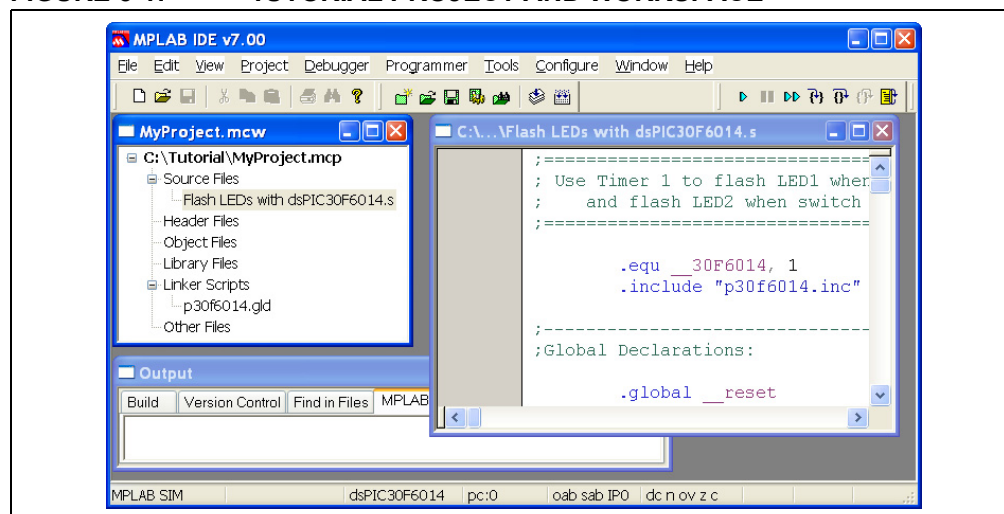
The project code is designed to run on a dsPICDEM 1.1 General Purpose Development Board. You will need a demo board to follow this tutorial.

Note: If you created the project for the dsPICDEM Starter Demonstration Board, the dsPICDEM 28-Pin Starter Demonstration Board or the dsPICDEM 2 Development Board, you can use your project on your demo board and still follow along with this section. The code is very similar. If you use the dsPICDEM 2 Development Board, you will need to ensure that the proper jumpers are inserted (H6-M ALL, H12-M D3 and H12-M D4).

Open the workspace you created in **Section 3.2 "Projects and Workspaces"** (select *File>Open Workspace* and browse to `C:\Tutorial\MyProject.mcw`). The workspace name should be visible in the title bar of the Project window and the name of the project should be visible inside the Project window at the top.

Build the project (*Project>Make* menu) to ensure that it is up to date.

FIGURE 5-1: TUTORIAL PROJECT AND WORKSPACE

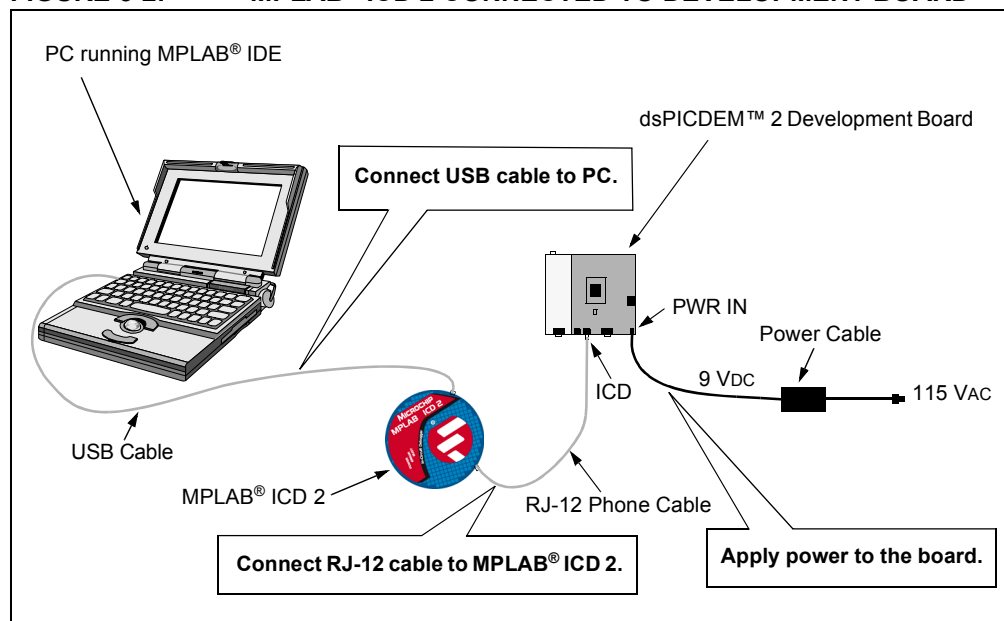


5.2 SETTING UP THE MPLAB ICD 2

Connect your PC to the MPLAB ICD 2 with the USB cable, as shown in Figure 5-2. Apply power to your target board and then connect the MPLAB ICD 2 to your target board using an RJ-12 (telephone type) cable.

Note: Be sure the MPLAB ICD 2 is connected to your PC through the USB before attaching it to the target.

FIGURE 5-2: MPLAB® ICD 2 CONNECTED TO DEVELOPMENT BOARD



Enable the MPLAB ICD 2 as a debugger (*Debugger>Select Tool>MPLAB ICD 2* menu). When you select the MPLAB ICD 2, the standard debugging operations, as well as MPLAB ICD 2 specific ones, are added to the debugger menu and toolbar. It is possible that MPLAB IDE will generate the following warning message in the Output window as soon as the MPLAB ICD 2 is selected in the menu:

"ICDWarn0030: MPLAB ICD 2 is about to download a new operating system"

Don't be alarmed. This condition is quite normal when a processor family is being used for the first time with the MPLAB ICD 2, or when switching between processors. It is also possible that you might see other error messages, depending on how the MPLAB ICD 2 has been set up.

If this is the first time the MPLAB ICD 2 is being used, the ICD 2 Setup Wizard will run. You can run it yourself using the *Debugger>MPLAB ICD 2 Setup Wizard* menu. The Wizard makes setting up the MPLAB ICD 2 a simple matter.

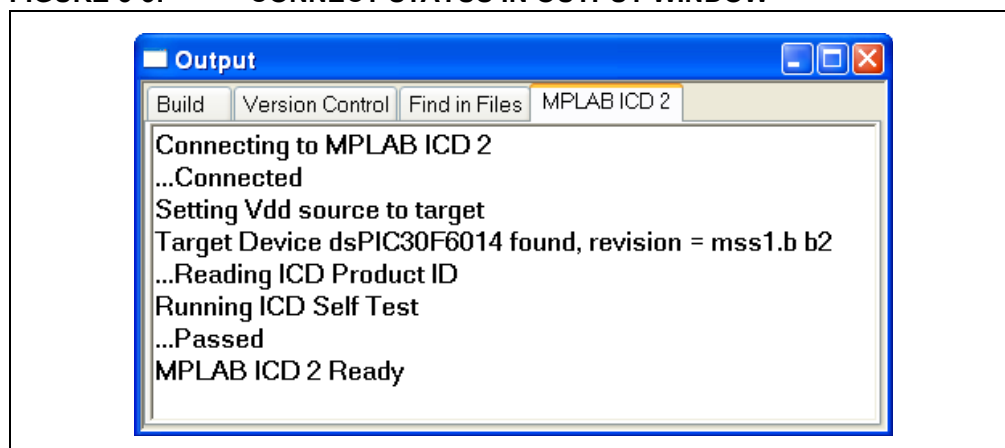
Choose the following options in the Wizard:

- At the Welcome screen, click **Next**.
- Select **USB** as the **Com Port**, click **Next**.
- Select **Target has own power supply**, click **Next**.
- Check **MPLAB IDE automatically connects to the MPLAB ICD 2**, click **Next**.
- Check **MPLAB IDE automatically downloads the required operating system**, click **Next**.
- At the Summary screen, click **Finish**.

It's that simple. The MPLAB ICD 2 is now set up and ready to be used. You can check the settings and make changes with the *Debugger>Settings* menu.

You can now make the MPLAB ICD 2 connect to your target board (use the *Debugger>Connect* menu). The Output window should indicate that the MPLAB ICD 2 has connected and identified the dsPIC30F6014 on your target board, as shown in Figure 5-3.

FIGURE 5-3: CONNECT STATUS IN OUTPUT WINDOW



Your version of silicon might be different, but the Output window should show that the correct target device was found.

Note: If you are using the dsPICDEM Starter Demonstration Board, the dsPIC30F6012 device should be found. If you are using the dsPICDEM 28-Pin Starter Demonstration Board, the dsPIC30F2010 device should be found. If you are using the dsPICDEM 2 Development Board, the dsPIC30F4011 device should be found.

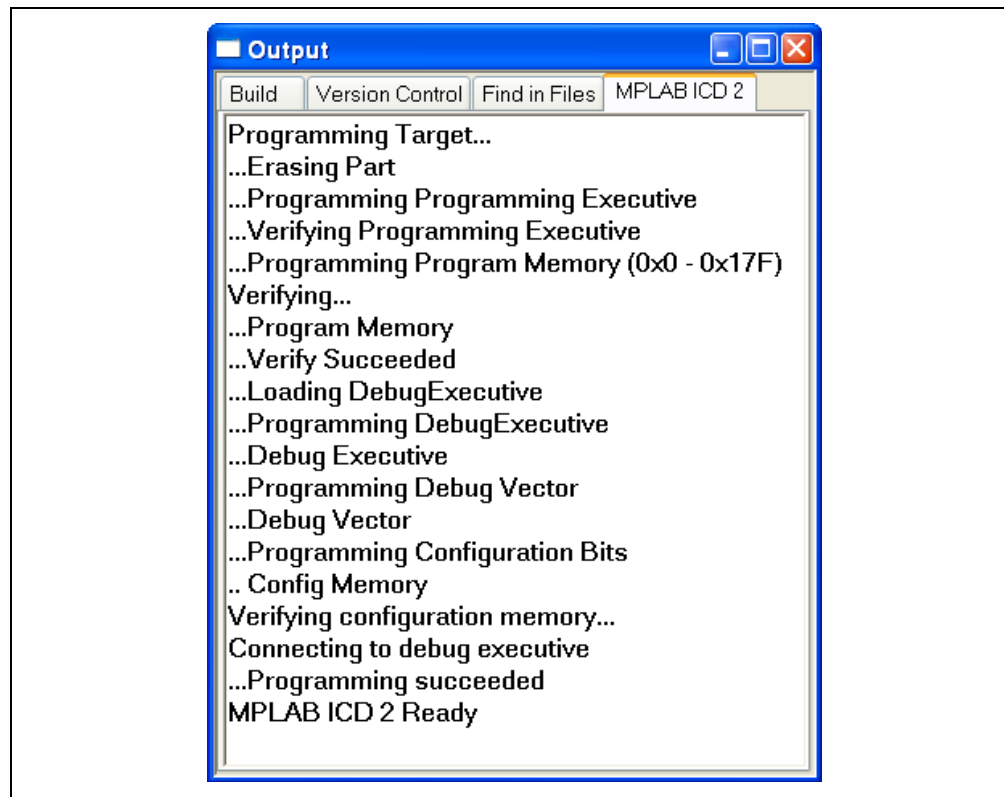
If you receive an "ICDWarn0020: Invalid target device id" message, there is probably a problem with the connections or the power supply. Double click the text "ICDWarn0020" for more help, but you will probably need to check your connections and power.

5.3 PROGRAMMING THE dsPIC DEVICE

Before you can do any debugging, you must program the part. Use the *Debugger>Program* menu to program the device.

The Output window displays the various steps in the programming and verification process. If any error messages or warnings are displayed, double click the message number for more detailed help.

FIGURE 5-4: PROGRAMMING STATUS IN OUTPUT WINDOW



When the status shows “MPLAB ICD 2 Ready”, the dsPIC30F6014 target chip is running the debug executive code, which allows you to step, set breakpoints, view registers and perform other debugging tasks.

It's important to note that the dsPIC device is running under the control of the debug executive code and will not run the code without a command from the MPLAB ICD 2. Later, if you wish to run the code by itself, you can use the MPLAB ICD 2 as a programmer instead of a debugger. The operation is similar, but you use the *Programmer* menu instead of the *Debugger* menu.

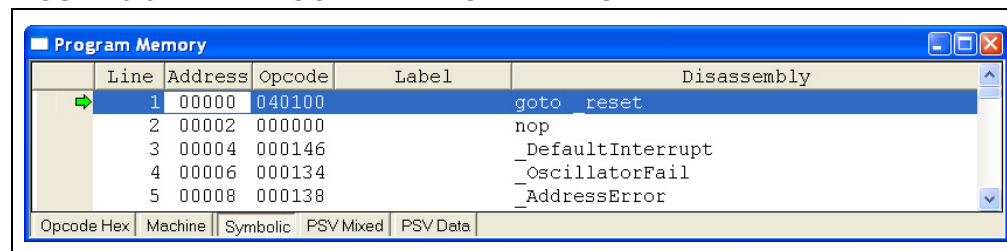
5.4 RESETTING THE CODE

After programming with MPLAB ICD 2 as a debugger, the Program Counter is automatically set to zero, the Reset vector. You can force a Reset with the *Debugger>Reset>Processor Reset* menu. The text “pc:0” appears in the Status bar at the bottom of the MPLAB IDE screen, showing that the Program Counter is zero.

Open the Program Memory window with the *View>Program Memory* menu and click on the **Symbolic** tab at the bottom of the window. There will be a green arrow in the gutter pointing to the first line containing the code at address zero, as shown in Figure 5-5.

Address zero contains a `goto __reset` instruction that is added automatically by the linker and is not in the source code file.

FIGURE 5-5: PROGRAM MEMORY WINDOW



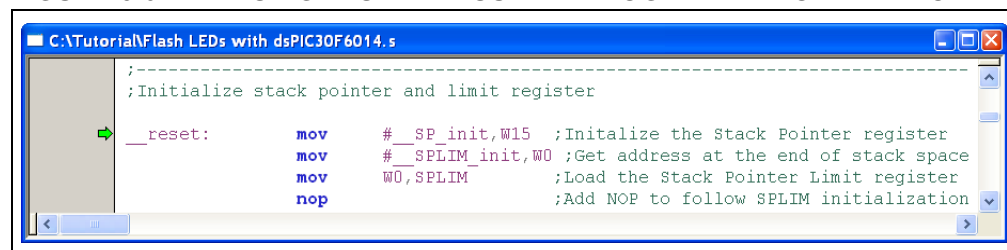
Close the Program Memory window at this point because all the other instructions are in the source code.

5.5 STEPPING THROUGH THE CODE

You can now step through the source code with the MPLAB ICD 2.

Use the **Debugger>Step Into** menu to single step the `goto __reset` instruction. The green arrow now points to the first line of executable code in the Flash LEDs with `dsPIC30F6014.s` file.

FIGURE 5-6: SINGLE STEP RESULT IN PROGRAM MEMORY WINDOW



Use the **Debugger>Step Into** menu to continue stepping through the code. The **Debugger** menu also lets you Step Into, Step Over and Animate the code:

- Step Into executes the current instruction and then halts. If the current instruction is a call to a subroutine, the Program Counter will change to the start of the called function.
- Step Over executes all code up to the next Program Counter location. It is just like the Step Into feature for most instructions. However, if the instruction is a `CALL`, it will execute the called subroutine in its entirety and then return.
- Animate steps continuously through your code. It is equivalent to doing Step Into operations repetitively until you select Halt.

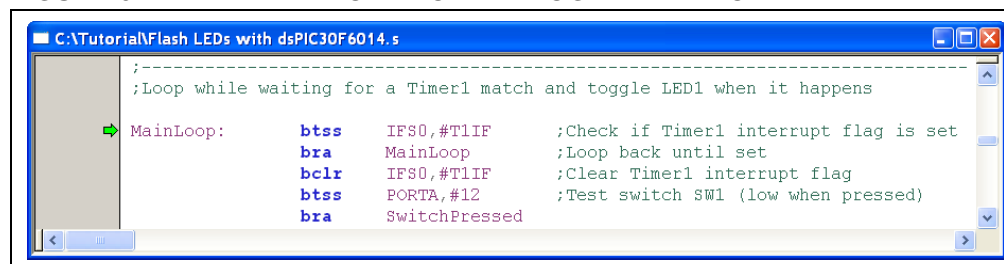
5.6 RUNNING THE CODE

To run your application, select *Debugger>Run*. The MPLAB IDE workspace Status bar displays the word “Running...” and a progress bar.

Nothing appears to be happening on the MPLAB IDE screen except for the moving bar. The green arrow in the gutter becomes transparent, since the Program Counter is changing and the current execution point cannot be shown. The LED on the demo board will be flashing because the code is running in the dsPIC device on the demo board.

To stop program execution, select *Debugger>Halt*. MPLAB IDE updates its windows after the Halt and indicates the current execution point with the green arrow, as shown in Figure 5-7.

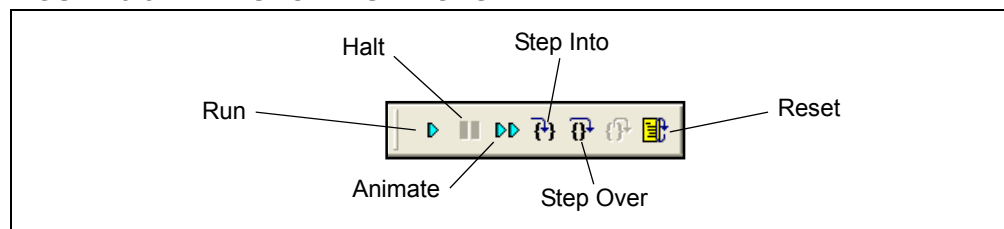
FIGURE 5-7: HALT LOCATION IN PROGRAM WINDOW



5.7 THE DEBUG TOOLBAR AND HOTKEYS

The Debug toolbar provides shortcut icons to control the MPLAB ICD 2, as shown in Figure 5-8.

FIGURE 5-8: SHORTCUT ICONS



When you select the MPLAB ICD 2, the Debug toolbar automatically opens. You can drag this toolbar anywhere on the desktop for convenience. Click on the appropriate toolbar icon to Run, Halt, Animate, Step Into, Step Over or Reset the program.

MPLAB ICD 2 also uses the following function keys to access the main debugging functions:

- <F5> Halt
- <F6> Reset
- <F7> Single Step
- <F9> Run

Additional functions are available by right clicking on a line of source code. The most important of these are Set Breakpoint and Run to Cursor.

If you do not see the toolbar, you can enable it with the *View>Toolbars>Debug* menu.

5.8 BREAKPOINTS

The MPLAB ICD 2 gives you the ability to set breakpoints – places in the code where execution will be halted. You can set breakpoints directly in your Source Code window, in the Program Memory window or in the Disassembly window.

In this example, you'll set a breakpoint where an LED is illuminated by toggling a bit in PORTD. Scroll down to line 95 and find the `btg LATD, #0` instruction. Note that the line number is shown in the Status bar.

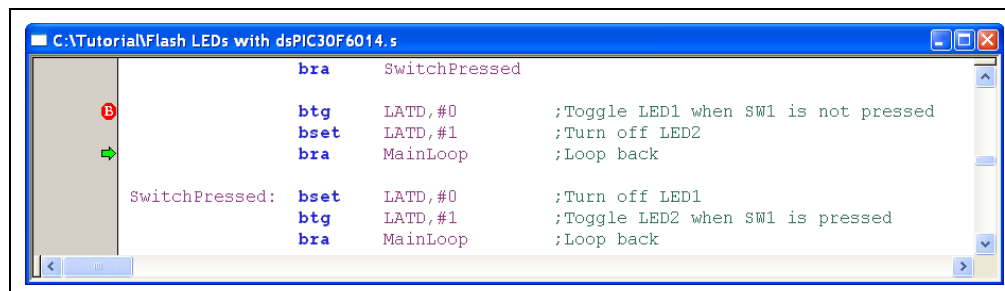
Note: If you're using the Flash LED with `dsPIC30F6012.s` file, set the breakpoint on line 95 on the `btg LATD, #4` instruction. If you're using the Flash LED with `dsPIC30F2010.s` file, set the breakpoint on line 90 on the `bclr IFS0, #T1IF` instruction. If you're using the Flash LED with `dsPIC30F4011.s` file, set the breakpoint on line 98 on the `btg LATB, #0` instruction.

Right click on the code line, then select Set Breakpoint from the pop-up menu.

You can also set a breakpoint by double clicking on the line. This option may need to be turned on with the Edit>Properties menu.

Now press <F9> or select the Debugger>Run menu. The program will halt after executing the line with the breakpoint, toggling the I/O pin. Notice that the LED on the demo board changes each time the code halts. After halting, the green arrow appears two instructions beyond the breakpoint, as shown in Figure 5-9.

FIGURE 5-9: PROGRAM WINDOW SHOWING BREAKPOINTS



One of the limitations of in-circuit debugging on the dsPIC devices is the skew when halting at a breakpoint. By the time the hardware has detected the breakpoint address, another instruction beyond the breakpoint has already been executed. This is not difficult to work around, however, simply adjust where your breakpoints are placed. It can be a little confusing if a branch instruction follows the breakpoint.

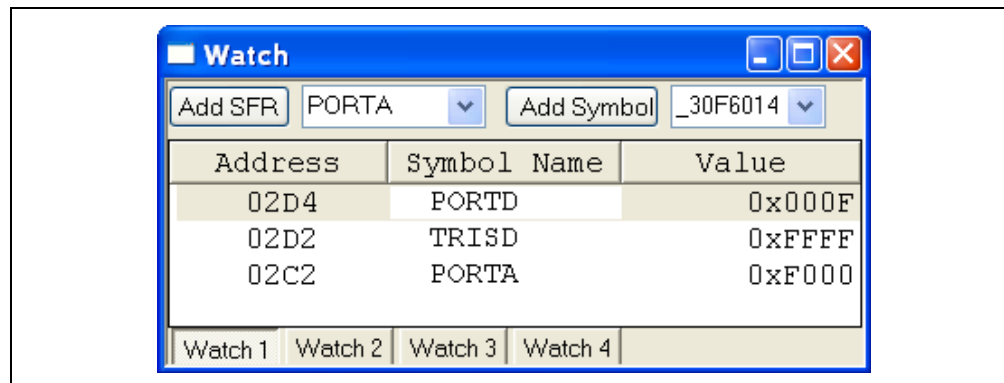
Try combinations of stepping with <F7> and running with <F9>. Notice that every time the code is made to run, it halts after the breakpoint. Try pressing switch SW1 on the development board and notice that the code does not halt while the switch is pressed.

Note: If you are using the dsPICDEM Starter Demonstration Board, press switch S1. If you are using the dsPICDEM 2 Development Board, press switch S5. The dsPICDEM 28-Pin Starter Demonstration Board does not have a switch.

5.9 WATCH WINDOW

There are several ways to view memory while using the MPLAB ICD 2. The Watch window (Figure 5-10) is one of the most useful. The Watch window lets you specify memory locations that you want to observe under different program conditions. To open a Watch window, select the View>Watch menu.

FIGURE 5-10: WATCH WINDOW



For example, to add PORTD to the Watch window, type "PORTD" in the **Add SFR** (Special Function Register) selection box at the top of the window. Then click **Add SFR** to add it to the Watch list.

You can also type the register name directly into the Watch window.

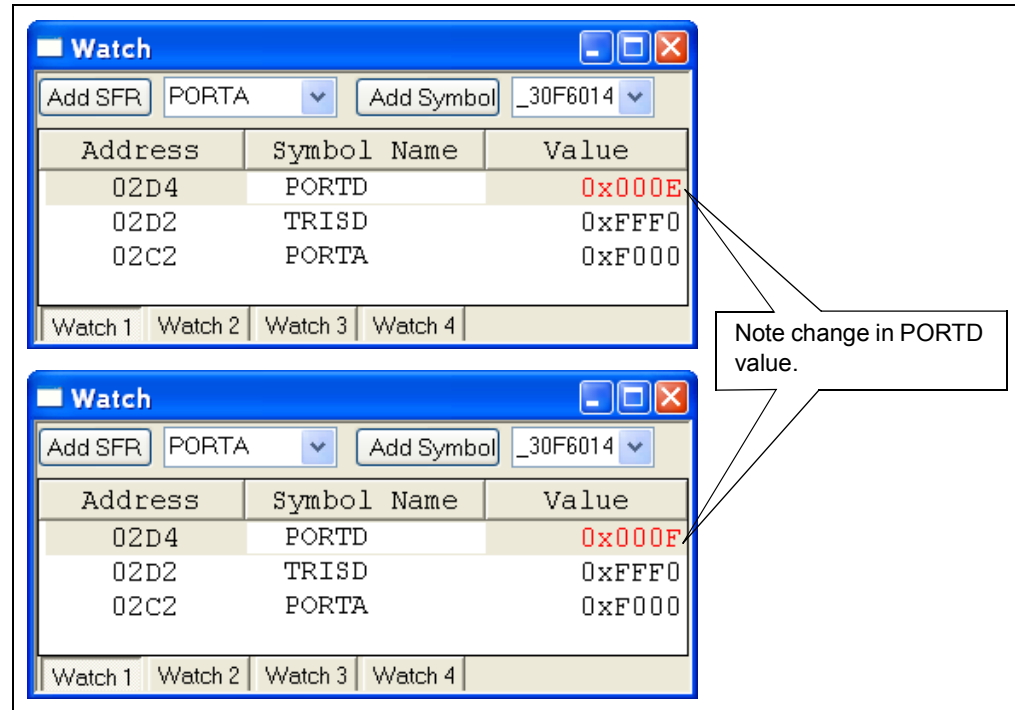
Add TRISD and PORTA in the same way. You should now have three Special Function Registers listed in the Watch window. There are columns for the "Address", "Symbol Name" and "Value" of the symbol.

Press <F6> to reset the code. Notice that the value of TRISD is 0xFFFF. This is the state of the TRISD register after a Reset. Press <F9> to run the code. The code should execute and halt at the breakpoint set earlier. Notice that the TRISD register has changed to 0xFF0. The code set up the I/O port directions by writing to TRISD and you can see the change in the Watch window. Notice that each time you Step or Halt, changed values appear in red, whereas unchanged values are black.

Note: If you are using the Flash LED with dsPIC30F6012.s file, then TRISD changes to 0xFF0F. If you are using the Flash LED with dsPIC30F2010.s file, TRISD changes to 0xFFFE. If you are using the Flash LED with dsPIC30F4011.s file, TRISB changes to 0xFFFC.

Note the state of PORTD in the Watch window and then press <F6> to run again. Each time the code runs and halts at the breakpoint, a bit in PORTD changes. This shows the code toggling the pin to turn the LED on or off.

FIGURE 5-11: WATCH WINDOWS SHOW CHANGING VALUES



Single step through the code by pressing <F7> while pressing and releasing switch SW1. Notice how PORTA changes when the switch is pressed. The switch is on pin RA12 of PORTA and the state of the input pin can be seen in the Watch window.

Note: If you are using the dsPICDEM Starter Demonstration Board, press switch S1 and watch PORTC in the Watch window. The switch is on pin RC13 of PORTC. If you are using the dsPICDEM 2 Development Board, press switch S5 and watch PORTE in the Watch window. The switch is on pin RE8 of PORTE.

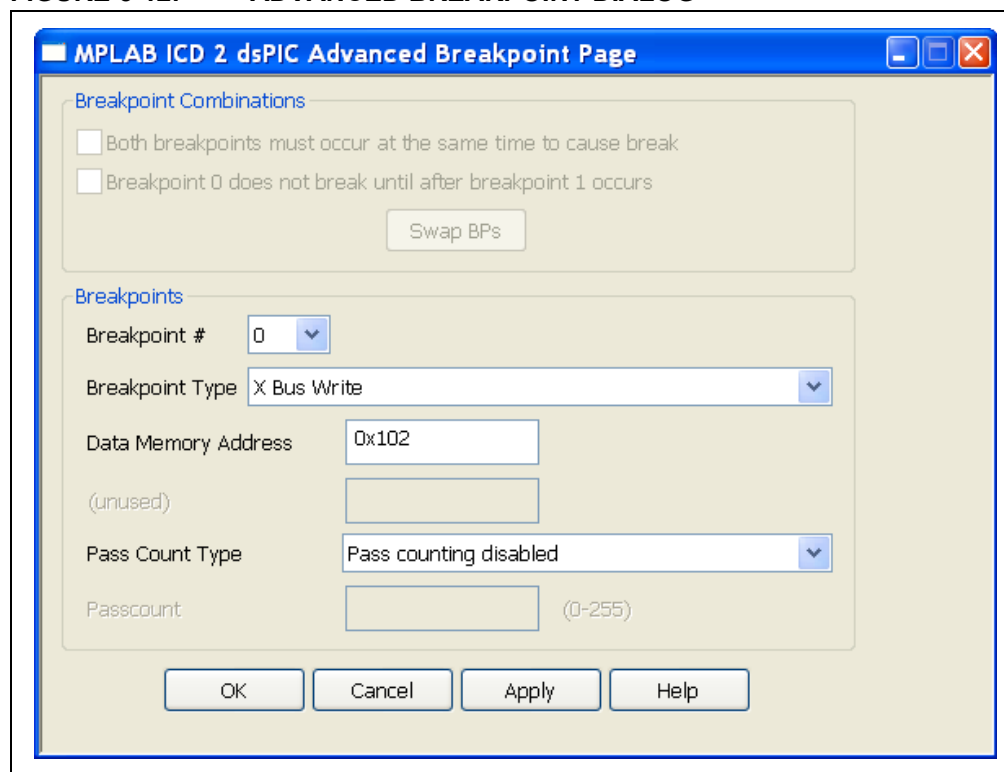
5.10 ADVANCED BREAKPOINTS

The advanced breakpoint feature of the MPLAB ICD 2 allows you to set up to break on a complicated set of conditions. This feature can help you trap an unusual occurrence in your code. In this case, set up an advanced breakpoint to halt when data is written to the PR1 register. You know this will happen because there is a `mov W0, PR1` instruction in the code. To find the address of the PR1 register, you can look in the `p30f6014.gld` linker script file and find the line, `PR1 = 0x0102;`.

Open the Advanced Breakpoint dialog (Figure 5-12) with the *Debugger>Advanced Breakpoints* menu. Two advanced breakpoints are available, but you'll only use Breakpoint # "0". Select "X Bus Write" for the Breakpoint Type to detect a write to address 0x0102, which is in the X data space. Actually, all data memory is accessed through the X bus unless very specific DSP operations are being performed.

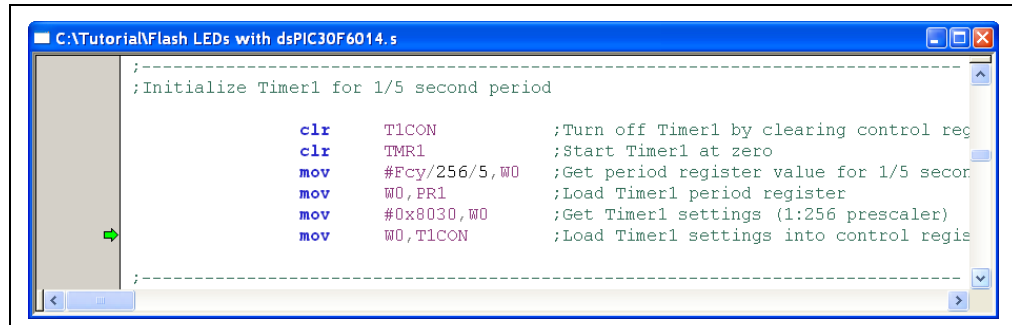
Set the Data Memory Address to "0x102" and leave "Pass counting disabled". Click **OK** to save the advanced breakpoint settings.

FIGURE 5-12: ADVANCED BREAKPOINT DIALOG



Now, press <F6> to reset the code and press <F9> to run. Notice that the code halts and the green arrow points to two instructions after the write to the PR1 register, as shown in Figure 5-13. The two extra cycles are caused by the skew mentioned earlier.

FIGURE 5-13: ADVANCED BREAKPOINT IN PROGRAM WINDOW



This demonstrates a very simple use of the advanced breakpoints. You can also break on reads and writes of specific data in specific memory locations. You can break on table reads and writes to locations in program memory, as well as simple instruction fetches from program memory. A pass counter is available so that a break can be made to occur once the event has occurred several times, or after several instruction cycles have been executed after an event.

Two separate advanced breakpoints are available and can be used in combination so that two independent events must occur to trigger the break. All in all, the advanced breakpoints provide very powerful debugging capabilities more often found in expensive emulators. The options are tremendous if you learn how to use them.

NOTES:

Chapter 6. MPLAB ICE 4000 In-Circuit Emulator

6.1 MPLAB ICE 4000 OVERVIEW

MPLAB ICE 4000 is an In-Circuit Emulator (ICE) designed to work with PIC18 and dsPIC devices. It provides full speed emulation and visibility into both the instruction and data paths during execution.

In addition to the basic Run, Halt, Single Step and Software Breakpoint functions, the MPLAB ICE 4000 provides advanced capabilities, such as instruction/address data monitoring, instruction data trace, complex triggering and code coverage. The MPLAB ICE 4000 can be connected to your target board and used in-circuit as the processor, or it can be used stand-alone for debugging your program.

FIGURE 6-1: MPLAB® ICE 4000 IN-CIRCUIT EMULATOR

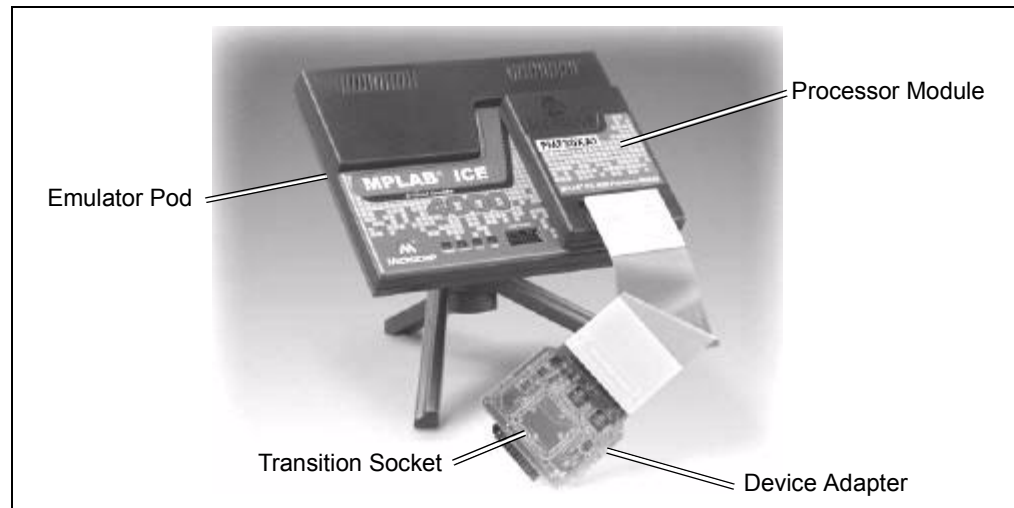


Figure 6-1 shows the main components of the MPLAB ICE 4000.

- The ICE 4000 emulator pod is the interface between your PC and the processor module. It contains hardware to read data from the processor module and send it back to the PC, as well as to take commands and various other data from the PC and send it to the processor module.
- The processor module is the component that actually emulates the specific device.
- The device adapter is an interchangeable assembly that allows the emulator to interface to a target application system.
- The transition socket allows a device adapter to be connected to a target application board designed to support a surface mount package style.

6.1.1 Installing the USB Driver

The MPLAB ICE 4000 has a USB interface. The USB drivers must be installed before you can use the MPLAB ICE 4000. Instructions for installing the USB drivers can be found in your MPLAB IDE installation directory:

`C:\Program Files\Microchip\MPLAB IDE\ICE 4000\Drivers\ezice4k.htm`

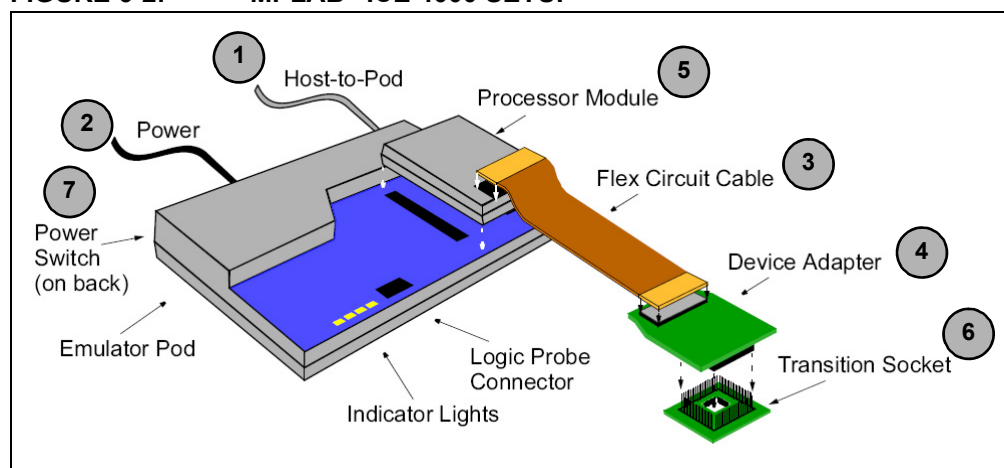
The instruction files may be different if you are using an operating system other than Windows XP or Windows 2000. Please look for the appropriate .htm file in the Drivers folder.

Note: The MPLAB ICE 4000 must be plugged in and turned on for the USB drivers to be installed. Please follow the instructions in the `ezice4k.htm` file before turning on the ICE 4000.

6.1.2 Connecting the MPLAB ICE 4000 Hardware

Connect the emulator and demo board as shown in Figure 6-2.

FIGURE 6-2: MPLAB® ICE 4000 SETUP



1. Connect the MPLAB ICE 4000 pod to your PC with the USB cable.
2. Connect the power supply to the emulator but do not switch it on.
3. Connect the end of the flat flex-circuit cable, marked Emulation Module, to the processor module.
4. Connect the other end of the flex-circuit cable, marked Device Adapter, to the device adapter.
5. Plug the processor module into the connector on top of the emulator pod.
6. Plug the device adapter onto the pins on the demo board. If there are no pins, you may need a transition socket.
7. Turn on power to the emulator pod and the demo board, in that order.

6.2 OPENING THE PROJECT

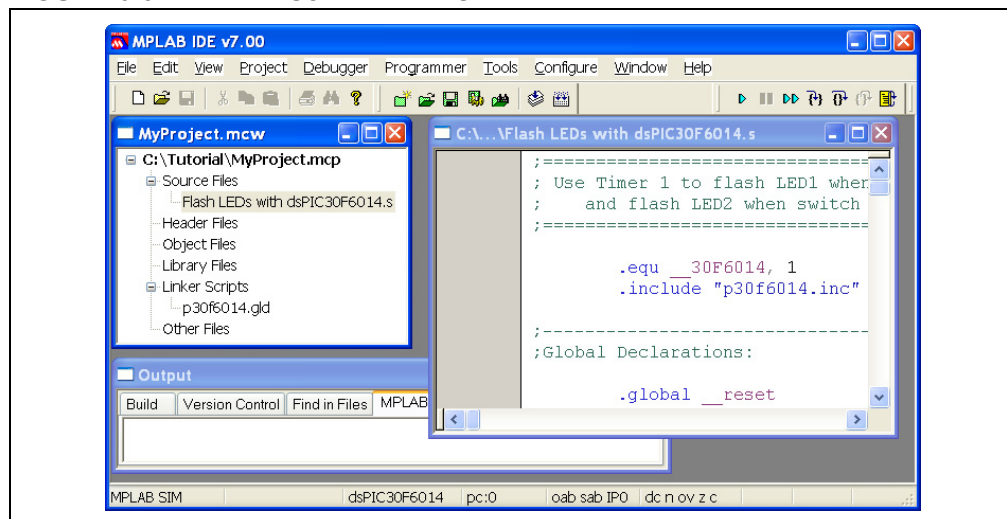
This chapter provides a tutorial that demonstrates how to use the MPLAB ICE 4000. First, you'll open the project that you created in the tutorial in **Section 3.3 “Creating a Project”**. If you have not yet created the project, please refer to that section now. The project code is designed to run on a dsPICDEM 1.1 General Purpose Development Board. You will need a demo board to be able to follow this tutorial.

Note: If you created the project for the dsPICDEM Starter Demonstration Board, the dsPICDEM 28-Pin Starter Demonstration Board or the dsPICDEM 2 Development Board, you can use your project on your demo board and still follow along with this section. The code is very similar.

If it is not already open, open the workspace you created in **Chapter 4. “The MPLAB SIM Simulator”** by selecting **File>Open Workspace** and browsing to C:\Tutorial\MyProject.mcw. The workspace name should be visible in the title bar of the Project window and the name of the project should be visible inside the Project window at the top, as shown in Figure 6-3.

Build the project (**Project>Make** menu) to ensure that it is up to date.

FIGURE 6-3: PROJECT WINDOW



6.3 SPECIAL EMULATOR DEVICES

The ICE 4000 processor module (e.g., PMF30XA1) uses a superset emulator chip that is capable of emulating all the dsPIC30F devices. This is very convenient and minimizes the number of different processor modules needed. However, the emulator chip has a fixed X/Y memory boundary.

As you may recall from **Section 1.2.3 “Data Memory”**, the dsPIC device has two separately addressable data memory spaces for DSP instructions. The boundary between these X and Y memory areas is fixed, but its location differs from device to device. It is necessary to modify the address map to use different memory locations on the ICE 4000 when you use some of the smaller parts, such as the dsPIC30F2010 or dsPIC30F4011. MPLAB IDE warns you when this is necessary, as shown in Figure 6-4.

FIGURE 6-4: XY DATA BOUNDARY WARNING



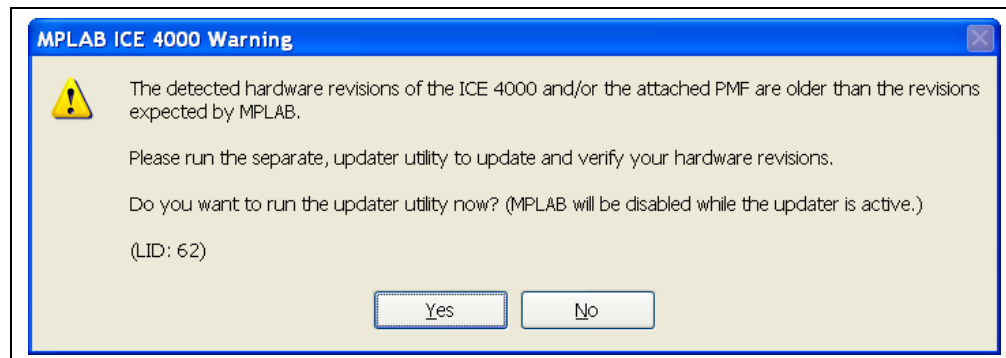
MPLAB IDE supports the different memory maps for the ICE 4000 by providing different part numbers for the affected devices. For example, to emulate a dsPIC30F2010, use the *Configure>Select Device* menu and select the dsPIC30F2010e device instead of the normal dsPIC30F2010. Similarly, add the `p30f2010e.gld` linker script file to your project instead of the `p30f2010.gld` file. **Section 3.3 “Creating a Project”** explains how to set up a project. You'll find that some devices, such as the dsPIC30F6014 and dsPIC30F6012, have the same XY boundary as the emulator chip and do not need a special part number.

6.4 SELECTING THE MPLAB ICE 4000

Enable the MPLAB ICE 4000 as a debugger (*Debugger>Select Tool>MPLAB ICE 4000* menu). The standard debugging operations, as well as MPLAB ICE 4000 specific ones, are added to the debugger menu and toolbar.

It is possible that MPLAB IDE will pop up the warning message shown in Figure 6-5 as soon as the MPLAB ICE 4000 is selected in the menu. Do not be alarmed. This condition is quite normal when the MPLAB ICE 4000 is being used for the first time, or after a new version has been installed.

FIGURE 6-5: MPLAB® ICE 4000 HARDWARE WARNING MESSAGE



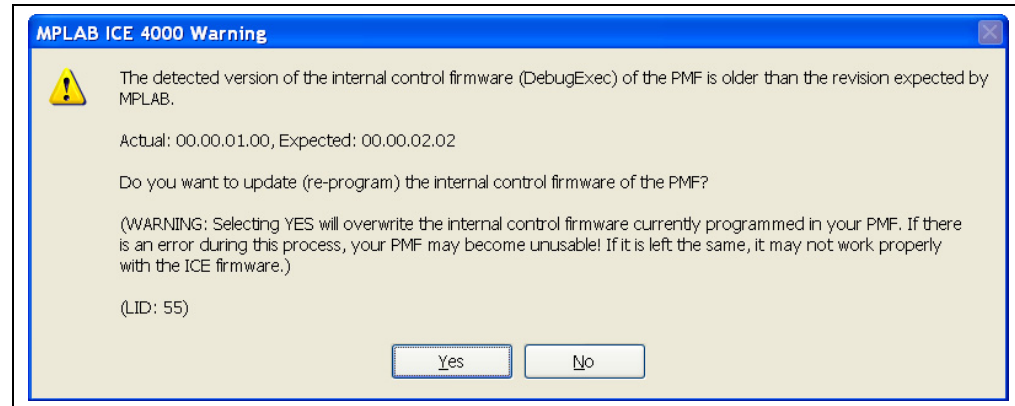
If this warning appears:

1. Click **Yes** to allow the MPLAB ICE 4000 to be updated.
2. When the ICE4K Update window appears, click **UpdateICE4K**.
3. Close the window when the update has completed.

The ICE 4000 updater program configures the programmable logic in the MPLAB ICE 4000 with the latest versions.

After updating the MPLAB ICE 4000 hardware, MPLAB IDE might pop up another warning message box asking to update the firmware in the processor module, as shown in Figure 6-6.

FIGURE 6-6: MPLAB® ICE 4000 FIRMWARE WARNING MESSAGE



If this warning appears, click **Yes** to allow the firmware to be updated.

Note: If updates are performed, you may need to reselect the MPLAB ICE 4000 as a debugger (*Debugger>Select Tool>MPLAB ICE 4000* menu).

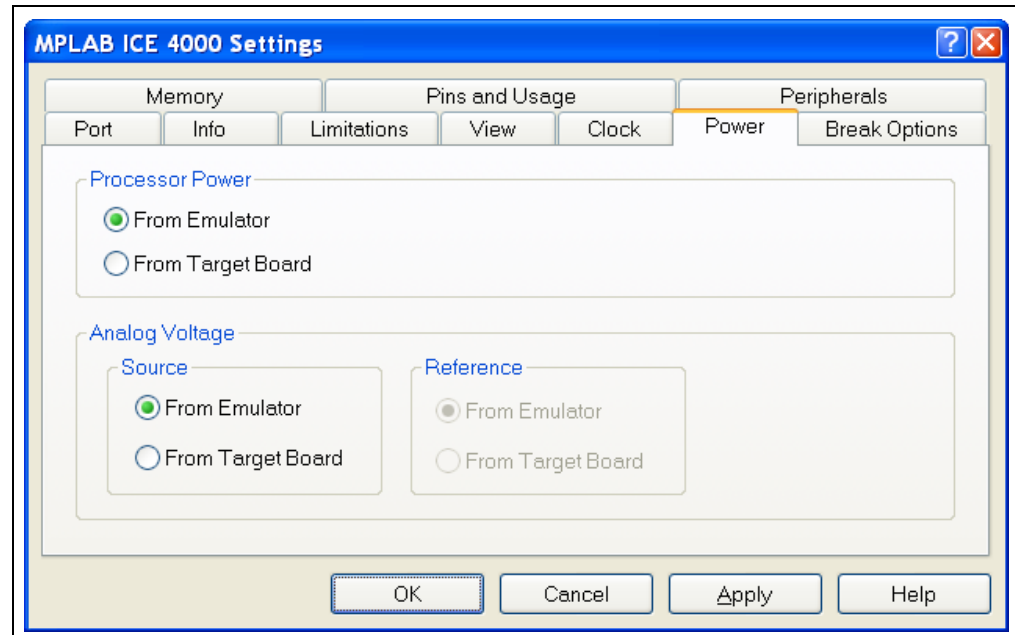
The emulator can take about a minute to initialize. You will hear relays clicking during this process. After initialization has completed, the MPLAB ICE 4000 is ready to be used.

6.5 MPLAB ICE 4000 SETTINGS

Before you start debugging with the MPLAB ICE 4000, you must be sure it is set up correctly:

1. Open the MPLAB ICE 4000 Settings dialog (*Debugger>Settings* menu).
2. Select the **Power** tab and check that all of the options are set to **From Emulator** (see Figure 6-7).

FIGURE 6-7: MPLAB® ICE 4000 SETTINGS – POWER TAB



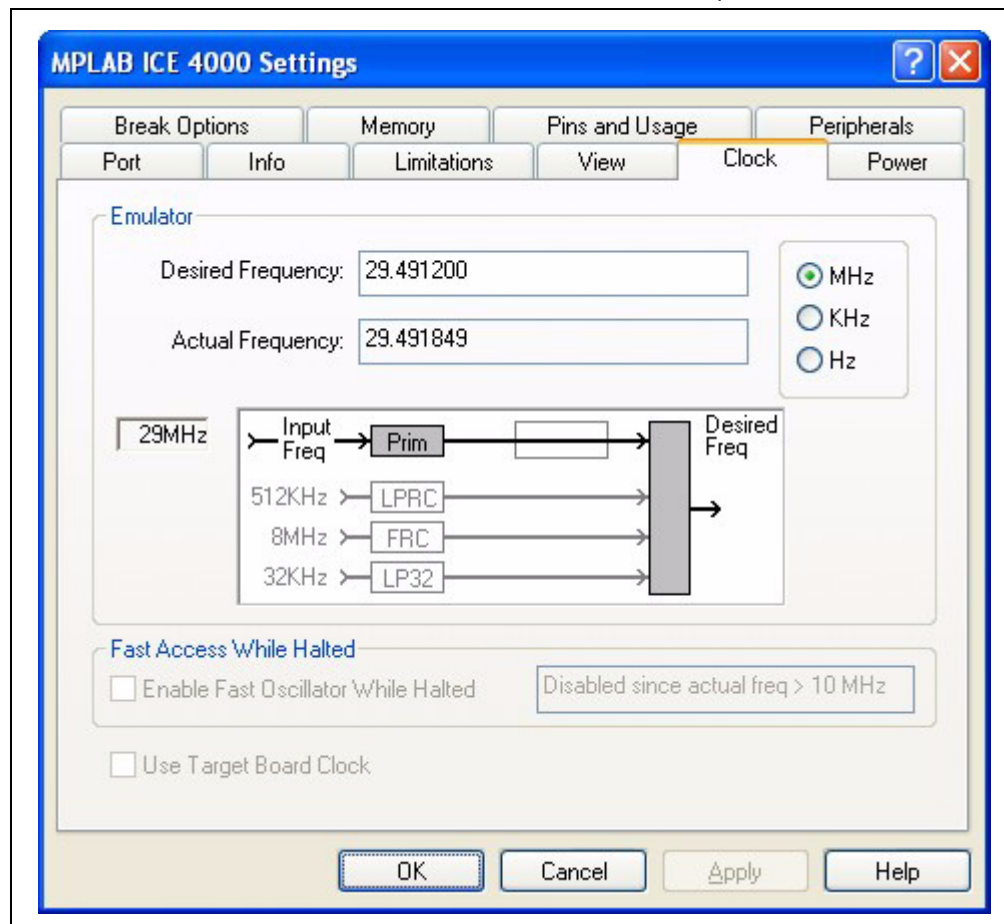
Getting Started with dsPIC30F Digital Signal Controllers

3. Select the **Clock** tab and set the Desired Frequency to 29.4912 MHz (Figure 6-8).

Note: If you are using the Flash LED with dsPIC30F6012.s file, set the processor frequency to 16 MHz. The other demo boards use a processor frequency of 29.4912 MHz.

4. Click **Apply**, then **OK**.

FIGURE 6-8: MPLAB® ICE 4000 SETTINGS – FREQUENCY



The dsPICDEM 1.1 General Purpose Development Board uses a 7.3728 MHz crystal and the code selects the 4x PLL option in the configuration bits, which steps up the frequency by a factor of four.

Thus, the processor runs at 29.4912 MHz. Each instruction cycle takes four clock cycles, so the instruction cycle rate is 7.3728 MHz.

Notice that the **Use Target Board Clock** checkbox is grayed out. You may only use the target clock on the demo board if you also set up the emulator to use target power.

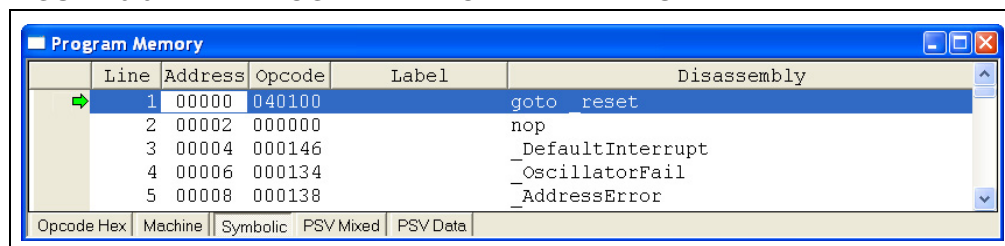
6.6 RESETTING THE CODE

After the MPLAB ICE 4000 is selected as the debugger, the Program Counter is automatically set to zero (the Reset vector). The Status bar at the bottom of the MPLAB IDE screen shows that the Program Counter is zero by displaying the text “pc:0”.

You can force a Reset with the *Debugger>Reset>Processor Reset* menu.

Open the Program Memory window (*View>Program Memory* menu) and click on the **Symbolic** tab at the bottom of the window (see Figure 6-9). A green arrow in the gutter points to the first line containing the code at address zero. Address zero contains a `goto __reset` instruction that is added automatically by the linker and is not in the source code file.

FIGURE 6-9: PROGRAM MEMORY AFTER RESET

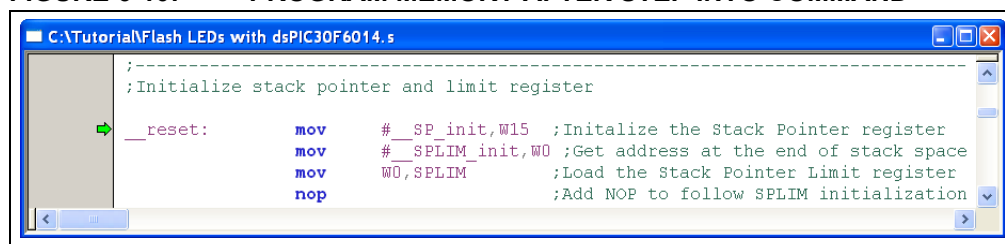


Close the Program Memory window at this point because all the other instructions are in the source code.

6.7 STEPPING THROUGH THE CODE

You can now step through the source code with the MPLAB ICE 4000. Use the *Debugger>Step Into* menu to single step the `goto __reset` instruction. The green arrow now points to the first line of executable code in the Flash LEDs with `dsPIC30F6014.s` file, as shown in Figure 6-10.

FIGURE 6-10: PROGRAM MEMORY AFTER STEP INTO COMMAND



Use the *Debugger>Step Into* menu to continue stepping through the code. The *Debugger* menu also lets you Step Into, Step Over and Animate the code.

- Step Into executes the current instruction and then halts. If the current instruction is a call to a subroutine, the Program Counter will change to the start of the called function.
- Step Over executes all code up to the next Program Counter location. It is just like the Step Into feature for most instructions. However, if the instruction is a `CALL`, it will execute the called subroutine in its entirety and then return.
- Animate steps continuously through your code. It is equivalent to doing Step Into operations repetitively until you select Halt.

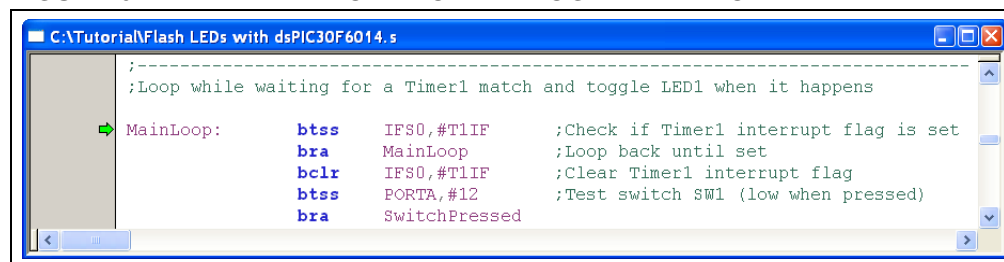
6.8 RUNNING THE CODE

Run your application (select *Debugger>Run*). The MPLAB IDE workspace Status bar displays the word “Running...” and a progress bar.

Nothing appears to be happening on the MPLAB IDE screen except for the moving progress bar. The green arrow in the gutter becomes transparent since the Program Counter is changing and the current execution point cannot be shown. The LED on the demo board will be flashing because the code is running in the dsPIC device on the demo board.

Stop program execution (select *Debugger>Halt*). MPLAB IDE updates its windows after the Halt and indicates the current execution point with the green arrow, as shown in Figure 6-11.

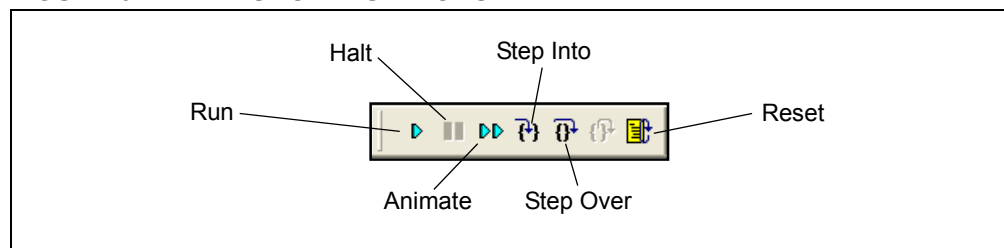
FIGURE 6-11: HALT LOCATION IN PROGRAM WINDOW



6.9 THE DEBUG TOOLBAR AND HOTKEYS

The Debug toolbar provides shortcut icons to control the MPLAB ICE 4000, as shown in Figure 6-12.

FIGURE 6-12: SHORTCUT ICONS



When you select the MPLAB ICE 4000, the Debug toolbar automatically opens (for convenience, you can drag this toolbar anywhere on the desktop). Click on the appropriate icon to Run, Halt, Animate, Step Into, Step Over or Reset the program.

MPLAB ICE 4000 also uses the following function keys to access the main debugging functions:

- <F5> Halt
- <F6> Reset
- <F7> Single Step
- <F9> Run

Additional functions are available by right clicking on a line of source code, which displays a menu of functions. The most important of these are Set Breakpoint and Run to Cursor.

If you do not see the toolbar, you can enable it with the *View>Toolbars>Debug* menu.

6.10 BREAKPOINTS

The MPLAB ICE 4000 allows you to set breakpoints – places in the code where execution will be halted. You can set breakpoints directly in your Source Code window, in the Program Memory window or in the Disassembly window.

In this example, you'll set a breakpoint where an LED is illuminated when a bit in PORTD is toggled. If you scroll down to line 95, you'll find the `btg LATD, #0` instruction. Note that the line number is shown in the Status bar.

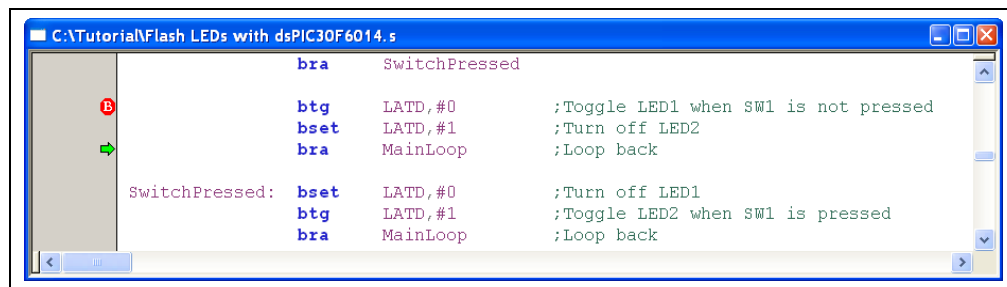
Note: If you are using the Flash LED with `dsPIC30F6012.s` file, set the breakpoint on line 95 on the `btg LATD, #4` instruction. If you are using the Flash LED with `dsPIC30F2010.s` file, set the breakpoint on line 90 on the `bclr IFS0, #T1IF` instruction. If you are using the Flash LED with `dsPIC30F4011.s` file, set the breakpoint on line 98 on the `btg LATB, #0` instruction.

Right click on the code line, then select Set Breakpoint from the pop-up menu.

You can also set a breakpoint by double clicking on the line. This option may need to be turned on with the Edit>Properties menu.

Now press <F9> or select the Debugger>Run menu. The program will halt after executing the line with the breakpoint, toggling the I/O pin. Notice that the LED on the demo board changes each time the code halts. After halting, the green arrow appears two instructions beyond the breakpoint, as shown in Figure 6-13.

FIGURE 6-13: PROGRAM WINDOW SHOWING BREAKPOINTS



One of the limitations of in-circuit debugging on the dsPIC devices is the skew when halting at a breakpoint. By the time the hardware has detected the breakpoint address, another instruction beyond the breakpoint has already been executed. This is not difficult to work around, however, simply adjust where your breakpoints are placed. It can be a little confusing if a branch instruction follows the breakpoint.

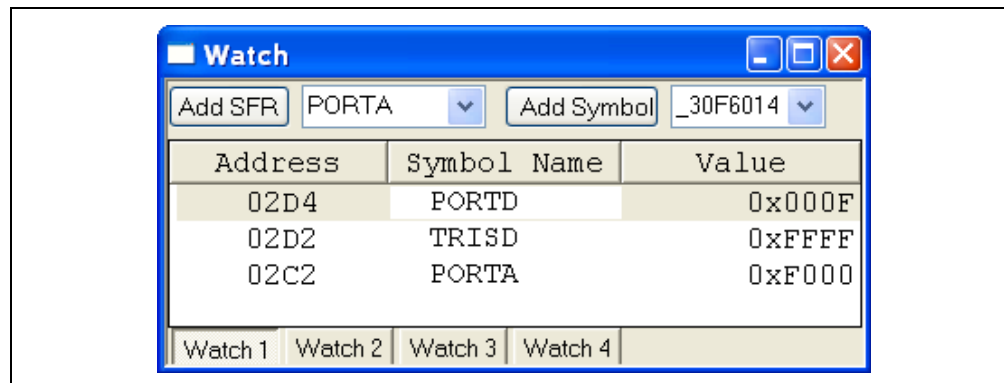
Try combinations of stepping with <F7> and running with <F9>. Notice that every time the code is made to run, it halts after the breakpoint. Try pressing switch SW1 on the development board and notice that the code does not halt while the switch is pressed.

Note: On the dsPICDEM Starter Demonstration Board, use switch S1. On the dsPICDEM 2 Development Board, use switch S5. The dsPICDEM 28-Pin Starter Demonstration Board does not have a switch.

6.11 WATCH WINDOW

There are several ways to view memory while using the MPLAB ICE 4000. The Watch window (Figure 6-14) is one of the most useful. The Watch window lets you specify memory locations that you want to observe under different programming conditions. To open a Watch window, select the View>Watch menu.

FIGURE 6-14: WATCH WINDOW



For example, to add PORTD to the Watch window, type "PORTD" in the **Add SFR** (Special Function Register) selection box at the top of the window. Then click **Add SFR** to add it to the Watch list.

You can also type the register name directly into the Watch window.

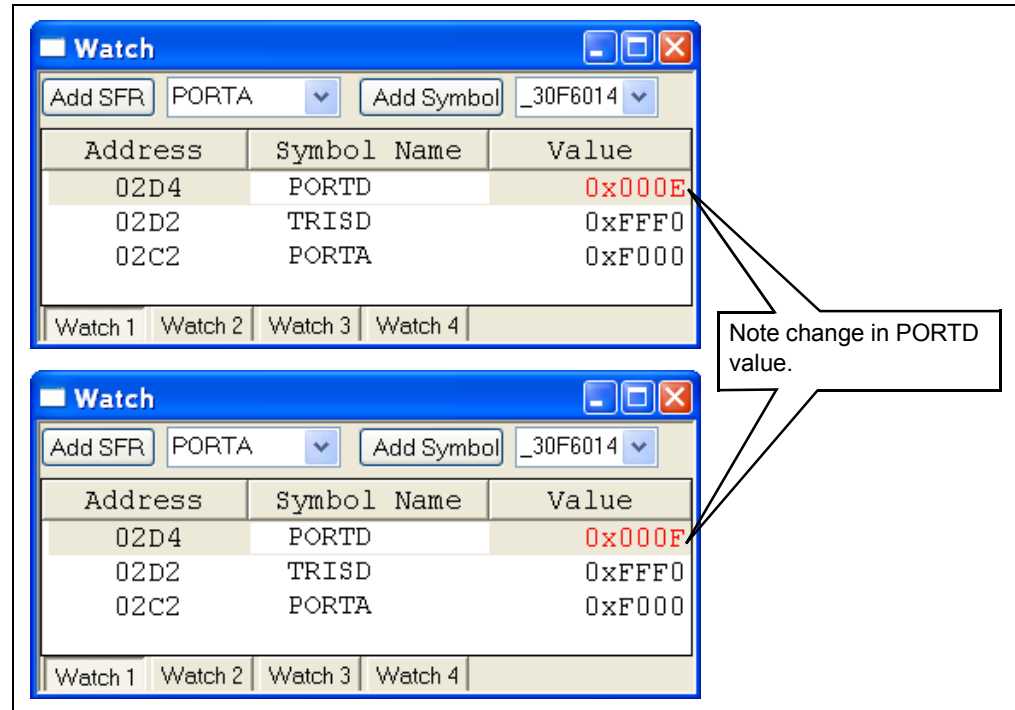
Add TRISD and PORTA in the same way. You should now have three Special Function Registers listed in the Watch window. There are columns for the "Address", "Symbol Name" and "Value" of the symbol.

Press <F6> to reset the code. Notice that the value of TRISD is 0xFFFF. This is the state of the TRISD register after a Reset. Press <F9> to run the code. The code should execute and halt at the breakpoint set earlier. Notice that the TRISD register has changed to 0xFF0. The code set up the I/O port directions by writing to TRISD and you can see the change in the Watch window. Notice that each time you Step or Halt, changed values appear in red, whereas unchanged values are black.

Note: If you are using the Flash LED with dsPIC30F6012.s file, then TRISD changes to 0xFF0F. If you are using the Flash LED with dsPIC30F2010.s file, then TRISD changes to 0xFFFE. If you are using the Flash LED with dsPIC30F4011.s file, then TRISB changes to 0xFFFC.

Note the state of PORTD in the Watch window and then press <F6> to run again. Each time the code runs and halts at the breakpoint, a bit in PORTD changes. This shows the code toggling the pin to turn the LED on or off.

FIGURE 6-15: WATCH WINDOWS SHOW CHANGING VALUES



Single step through the code by pressing <F7> while pressing and releasing switch SW1. Notice how PORTA changes when the switch is pressed. The switch is on pin RA12 of PORTA and the state of the input pin can be seen in the Watch window.

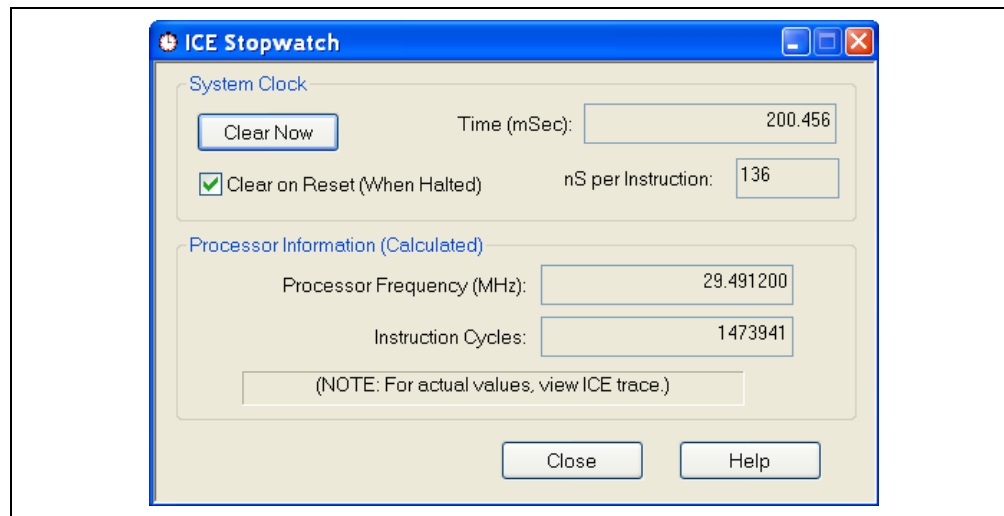
Note: If you are using the dsPICDEM Starter Demonstration Board, press switch S1 and watch PORTC in the Watch window. The switch is on pin RC13 of PORTC. If you are using the dsPICDEM 2 Development Board, press switch S5 and watch PORTE in the Watch window. The switch is on pin RE8 or PORTE.

6.12 STOPWATCH

You can measure the execution time between two events with the Stopwatch feature of MPLAB ICE 4000. The Stopwatch keeps track of the number of instruction cycles that are executed and the amount of time consumed by the cycles. It calculates the time from the “Processor Frequency” that you entered in the settings.

Open the Stopwatch by selecting the *Debugger>Stopwatch* menu.

FIGURE 6-16: MPLAB® ICE 4000 STOPWATCH



Now do this brief exercise:

1. Check **Clear on Reset**. This setting ensures that the Stopwatch will count from zero after you perform a Reset.
2. Click in your Source Code window and then press <F6> to Reset and <F9> to Run. The execution should halt at the breakpoint that you set earlier in **Section 6.10 “Breakpoints”**.
3. Notice that the value of **Time (mSec):** is 200 ms. The code sets up the Timer1 Period register for 1/5 of a second (200 ms). The breakpoint is set where the timer period has been detected.
4. Press <F9> to Run. The code will halt at the breakpoint again.
5. Notice that the Stopwatch time is now 400 ms because another timer period has elapsed.

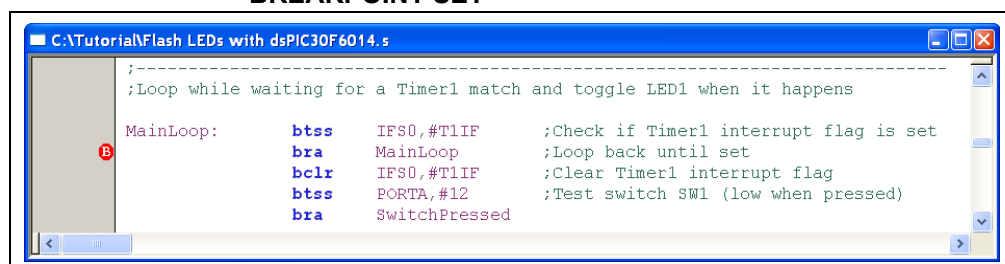
You can start the time from zero at any point by pressing the **Clear Now** button. This gives you the flexibility to easily time individual loops and functions.

6.13 TRACE BUFFER

The Trace buffer is a handy feature of the MPLAB ICE 4000. You'll find this feature under the *View>ICE Trace* menu.

The Trace buffer holds a list of the instructions that have executed. It can hold more than 65,000 instructions. The Trace buffer works in the background and does not need to be explicitly enabled. Simply executing code causes it to be recorded in the Trace buffer.

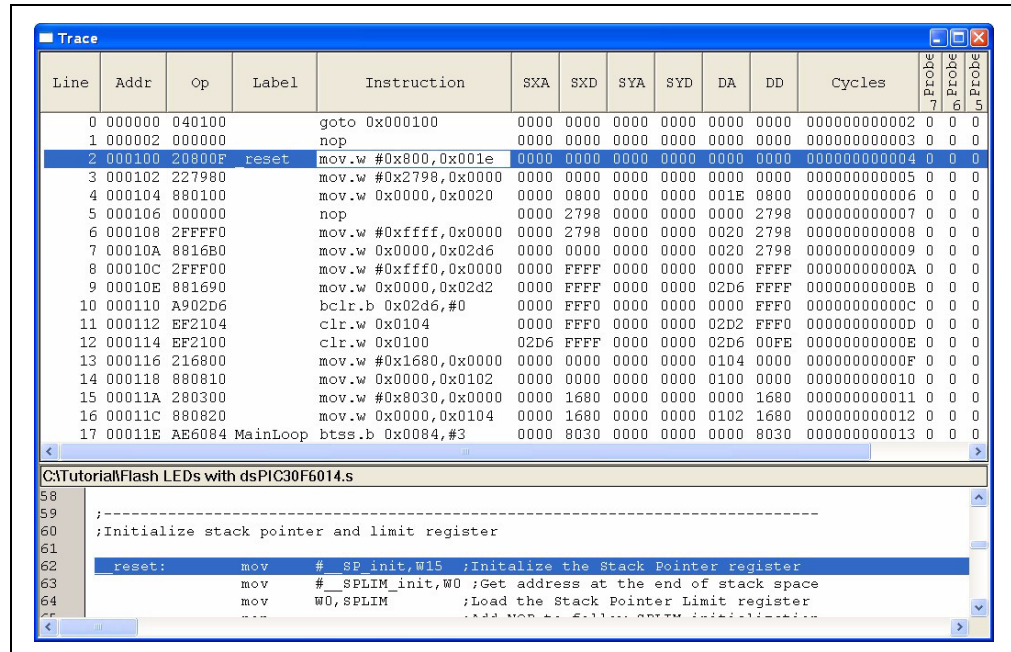
FIGURE 6-17: PROGRAM WINDOW WITH TRACE BUFFER BREAKPOINT SET



As a simple experiment, follow these steps to execute a few lines of code and view them in the Trace:

1. In the Program window, set a breakpoint on the `bra MainLoop` instruction, immediately after the `MainLoop:` address label (see Figure 6-17).
2. Press <F6> to Reset and <F9> to Run. The code will halt at the breakpoint.
3. Select *View>ICE Trace* to display the Trace buffer. All the instructions that were executed are shown in the Trace window, starting with the `goto __reset` instruction at address zero (see Figure 6-18). If the Trace view is blank, check what filter trace is not selected in the **Complex Trigger Settings** (see **Section 6.14 "Complex Triggers"**).
4. Scroll to the very end of the Trace to see the last instruction that was executed. Notice that the Trace recorded the `btss IFS0,#T1IF` instruction immediately before the breakpoint and then stopped.
5. Compare the Trace buffer with the source code; you'll see that all the instructions up to the breakpoint are represented in the Trace window. Notice that when you select a line in the Trace view, the associated source code line is highlighted at the bottom of the window, as shown in Figure 6-18.

FIGURE 6-18: TRACE WINDOW



6.14 COMPLEX TRIGGERS

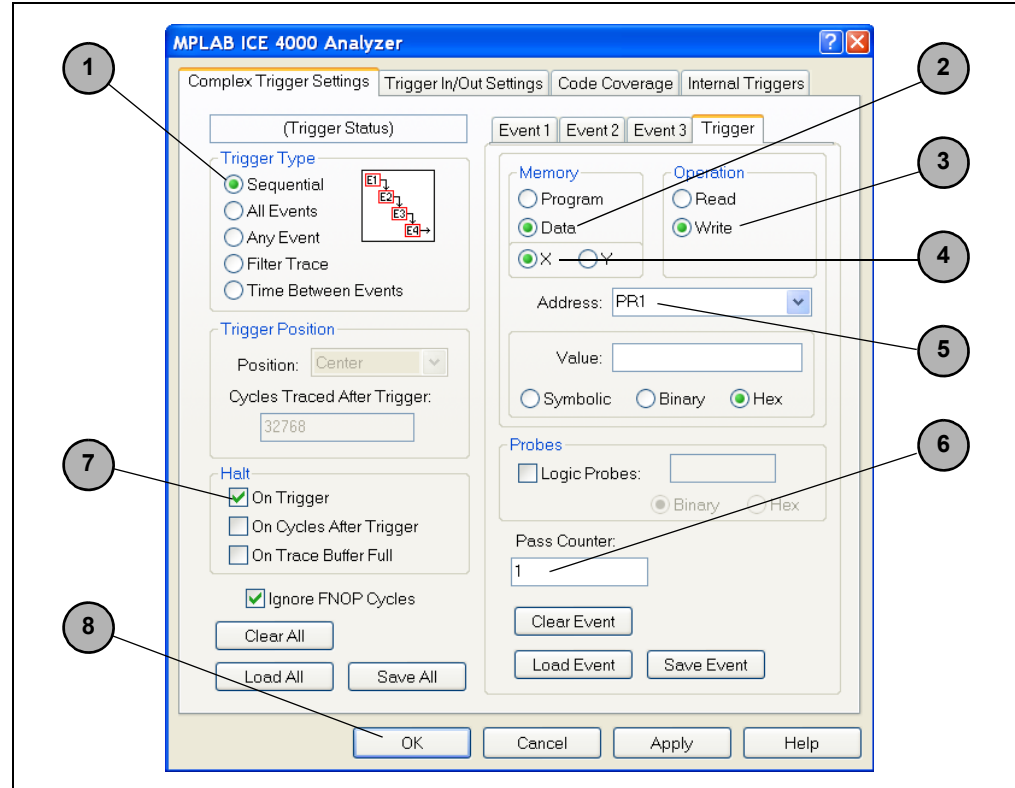
The complex triggers feature allows you set up the MPLAB ICE 4000 to break on a complicated set of conditions. This capability is very useful if you need to trap an unusual occurrence in your code. For illustration, you can set up a complex trigger to halt when data is written to the PR1 register.

You know this will happen because there is a `mov W0, PR1` instruction in the code. To find the address of the PR1 register, look in the `p30f6014.gld` linker script file and find the line, `PR1 = 0x0102;`.

Use the *Debugger>Complex Triggers and Code Coverage* menu to open the MPLAB ICE 4000 Analyzer dialog. This dialog provides individual tabs for:

- **Complex Trigger Settings** – used to break on a combination of up to four events, such as program memory or data memory reads or writes. It is also possible to break on logic signals detected on the **Logic Probes:** input.
- **Trigger In/Out Settings** – used to trigger on a single external trigger input and to output a pulse that can be used to trigger an oscilloscope or other data capture device.
- **Code Coverage** – used to track which program memory instructions are used. It can be useful during testing to see that all the code was actually executed during the tests.
- **Internal Triggers** – additional trigger circuits built into the dsPIC processor modules that allow triggering on a combination of events.

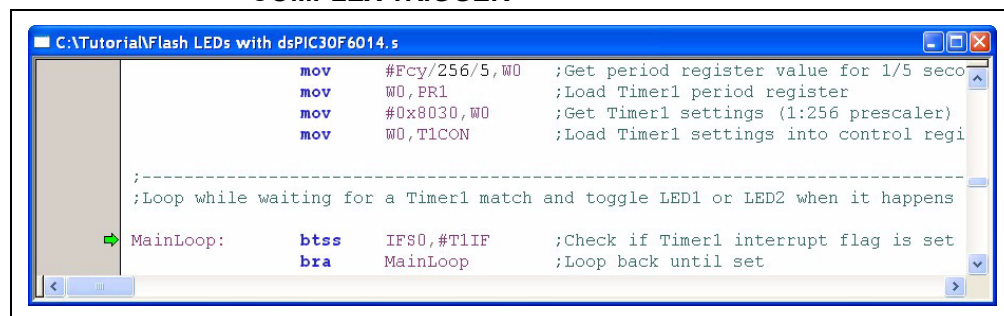
Select the **Complex Trigger Settings** tab and set it up as shown in Figure 6-19 and described in Table 6-1.

FIGURE 6-19: MPLAB® ICE 4000 ANALYZER DIALOG

TABLE 6-1: COMPLEX TRIGGER SETTINGS

Ref	Setting
1	Make sure that the (Trigger Status) is set to Sequential
2	Set the Memory type to Data
3	Set the Operation type to Write (because we want to detect a write to data memory address PR1)
4	Select X under the Memory type (all data memory writes in the dsPIC® devices are to X memory space)
5	Set the Address: to "PR1"
6	Leave the Pass Counter: set to "1"
7	Set the Halt type to On Trigger
8	Click OK to save the Complex Trigger Settings

Press <F6> to Reset and <F9> to Run the code. Notice that execution halts a few cycles after the instruction that writes to the PR1 register has executed, as shown in Figure 6-20.

FIGURE 6-20: PROGRAM WINDOW WITH PROGRAM HALTED AFTER COMPLEX TRIGGER



```
C:\Tutorial\Flash LEDs with dsPIC30F6014.s
mov    #Fcy/256/5,W0    ;Get period register value for 1/5 sec
mov    W0,PR1           ;Load Timer1 period register
mov    #0x8030,W0       ;Get Timer1 settings (1:256 prescaler)
mov    W0,T1CON         ;Load Timer1 settings into control regi

;-----
;Loop while waiting for a Timer1 match and toggle LED1 or LED2 when it happens

MainLoop:    btss    IFS0,#T1IF    ;Check if Timer1 interrupt flag is set
             bra     MainLoop      ;Loop back until set
```

This demonstration is a very simple use of the complex triggers. You can break on reads and writes of specific data to either data or program memory locations. You can incorporate the pass counter to trigger after the desired event has happened several times or after several instruction cycles have been executed after an event.

Four separate trigger events are available. They can be used in combination so that a specific sequence of events must occur to cause the break. The complex triggers provide very powerful debugging capabilities.

Chapter 7. The MPLAB ASM30 Assembler

7.1 MPLAB ASM30 ASSEMBLER OVERVIEW

Now that you know how to create and build a project and use the tools to simulate or debug, let's spend a little time learning how to write code. Since the MPLAB ASM30 Assembler is included with MPLAB IDE, we'll discuss some of the essentials to using this language tool.

The MPLAB IDE assembler is based on open source GNU software, which may seem familiar to some users. The assembler interprets instructions and directives in source code files to generate object code. A linker is used to convert the object code to a final output (Hex) file for programming a part (see **Chapter 9. "The MPLAB LINK30 Linker"**).

Instructions are executed at run time in the dsPIC device. They are the native language of the dsPIC processor. However, the dsPIC instruction set is beyond the scope of this document. For detailed information about the dsPIC instruction set, refer to the *"dsPIC30F Programmer's Reference Manual"* (DS70030).

Directives are interpreted at build-time by the assembler and are used to define sections of memory, initialize constants, declare and define symbols (variables, labels, etc.), substitute text and so forth. A list of directives and their usage is documented in the *"MPLAB[®] ASM30, MPLAB[®] LINK30 and Utilities User's Guide"* (DS51317). A period (".") must precede each directive.

We'll discuss a few of the most commonly used directives so you'll have an idea of what is required when writing your own code. Many of these directives were used in the example code from the preceding tutorials.

Note: This chapter is based on MPLAB ASM30 Assembler version 1.31. Some information may become dated as new versions are released.

7.1.1 General Format of Instructions and Directives

Instructions and directives take the following general forms:

```
[label:]      instruction[operands]    [; comment]
[label:]      directive[arguments]    [; comment]
```

Labels are used to mark locations in code. At link time, labels are evaluated to a memory address; label definitions may begin with a "." (period) and must end with a ":" (colon).

Operands are used by instructions to provide source and destination information. They consist of:

- **Literals** – These are hexadecimal, octal, binary or decimal values. A number sign "#" must precede all literal values.
- **Registers and Memory Addresses** – These are working registers, accumulators, General Purpose Registers (GPRs) and Special Function Registers (SFRs).
- **Condition Codes** – These are Status bits, such as Z (Zero) or C (Carry), used as operands in conditional branch instructions.

Arguments are similar to operands. Arguments are used by directives for source and destination information.

Syntax rules for instructions and directives are summarized in Table 7-1.

TABLE 7-1: SYNTAX RULES

Character	Description	Usage
.	Period	Begins a directive or label
:	Colon	Ends a label
#	Pound	Begins a literal value
;	Semicolon	Begins a single-line comment
/*		Begins multi-line comment
*/		Ends multi-line comment

7.2 COMMONLY USED DIRECTIVES

Some commonly used directives are listed below. They are all given as examples in the dsPIC template files, which you can find in the following directory:

C:\Program Files\Microchip\MPLAB ASM30 Suite\Support\templates\assembly

You will also find the first five directives in our tutorial from **Chapter 4. “The MPLAB SIM Simulator”**, where we learned how to create and build a project. Although the other five directives were not present in our tutorial, you may find yourself in need of one of them.

.equ	equates a value to a symbol
.include	includes another file into the current file
.global	makes a symbol globally visible
.text	starts a section of executable code
.end	ends assembly within a file
.section	starts a section (of code or data, in program or data memory)
.space	allocates space within a section
.bss	adds variables to the uninitialized data section
.data	adds variables to the initialized data section
.hword	declares words of data within a section
.palign	aligns code within a section
.align	aligns data within a section

.equ

One of the common directives in any assembly source file is `.equ`. The `.equ` directive is used to define a symbol and assign it a value. In Example 7-1, the `.equ` directive is used to assign the literal value of 7372800 to the symbol `FCY`. In this context, `FCY` is a constant that can be used throughout the code to represent the instruction cycle frequency.

EXAMPLE 7-1: .equ

```
;Program Specific Constants (literals used in code)
.equ   Fcy, #7372800           ;Instruction cycle rate (Osc x PLL / 4)
;=====
```

.include

The `.include` directive adds the contents of the specified file into the assembly source, at the point it is used, as shown in Example 7-2. One common use of the `.include` directive is to add in definitions from the standard processor include file.

EXAMPLE 7-2: .include

```
.equ   30F6014, 1
.include "p30f6014.inc."
;-----
```

.global

The `.global` directive is used to allow labels defined within the file to be used by other files. In Example 7-3, the `__reset` symbol is made global so that the linker can use it as the address to jump to, from the Reset vector. The `__reset:` label is required to signify the start of code and needs to be present in one of the project's object files (from assembler, compiler or library files).

EXAMPLE 7-3: .global

```
;Global Declarations
.global __reset           ;The label for the first line of code
.global __OscillatorFail  ;Declare Oscillator Fail trap routine 1
.global __AddressError    ;Declare Address Error trap routine 1a
```

.text

This is a special instance of the `.section` directive. The `.text` directive is used to inform the assembler that the code following it is to be placed in an executable section of program memory (see Example 7-4).

EXAMPLE 7-4: .text

```
;Start of code
.text           ;Start of Code section
```

.end

The `.end` directive is used to signify the end of an assembly source file (see Example 7-5).

EXAMPLE 7-5: .end

```
.end           ;End of code in this file
```

.section

The `.section` directive declares a section of memory. This section can be in RAM or in program memory, as determined by the attributes that follow the directive. In Example 7-6, the section named, `MyDataSection`, is placed in uninitialized near data memory. The section named, `MyOtherSection`, is placed in Y data memory. A complete list of section types is contained in the “MPLAB® ASM30, MPLAB® LINK30 and Utilities User's Guide” (DS51317) and in the “dsPIC30F Language Tools Quick Reference Card” (DS51322).

EXAMPLE 7-6: .section

```
;RAM variables
.section MyDataSection, bss, near
Var1: .space 1           ;Allocating space (in bytes) to variable
.section MyOtherSection, ymemory
Array1: .space 20        Allocating space (in bytes) to array
```

.space

The `.space` directive instructs the assembler to reserve space in the current section. In Example 7-7, a one-byte space of memory is reserved for the variable named `Var1`.

EXAMPLE 7-7: .space

```
;RAM variables
        .section MyDataSection, bss, near
Var1:   .space 1           ;Allocating space (in bytes) to variable
```

.bss

The `.bss` directive is a special instance of the `.section` directive. It causes uninitialized data variables to be appended to an uninitialized data section. In Example 7-8, `Var2` will be placed in uninitialized data memory.

EXAMPLE 7-8: .bss

```
;RAM variables
        .bss
Var2:   .space 2           ;Allocating space (in bytes) to variable
```

.data

The `.data` directive is a special instance of the `.section` directive. It causes initialized data variables to be appended to an initialized data section. In Example 7-9, the array, `MyRAM`, will be placed in data memory and the assembler will place the data, `0x1111`, `0x2222` and `0x3333`, in a program memory section.

EXAMPLE 7-9: .data

```
;Initialized RAM variables
        .data
MyRAM:  .hword 0x1111, 0x2222, 0x3333
```

It is important to note that in order to use initialized data, the correct start-up code needs to be added to the project to copy the data to RAM. The run-time start-up module is included in the run-time library, the `libpic30.a` file. This file is present in the `pic30_tools\lib` folder.

Refer to the “*MPLAB® ASM30, MPLAB® LINK30 and Utilities User’s Guide*” (DS51317) for further information on the functions of the start-up module in the run-time library.

.hword

The `.hword` directive declares words of initialized data within a section. It can also declare constant data within program memory. In Example 7-10, the `MyData` array is placed in program memory. The data words, `0x0002`, `0x0003` and `0x0005`, will be stored in adjacent words of program memory. Since program memory is 24 bits wide, the upper byte of each word will be `0x0`.

EXAMPLE 7-10: .hword

```
        .align 2           Align next word to a two byte boundary
MyData: .hword 0x0002, 0x0003, 0x0005
```


.align

The `.align` directive aligns data within a program memory section. In Example 7-11, the variable, `MyData`, will start at an even address (exactly divisible by 2).

EXAMPLE 7-11: .align

```
.section .myconstbuffer, "x"
    .align 2                Align next word to a two byte boundary
MyData:.hword 0x0002, 0x0003, 0x0005
```

.align

The `.align` directive aligns data within a section. In Example 7-12, the variable `Array3` will start at an address that is exactly divisible by 8. The `.align` directive is especially useful when using the modulo addressing feature or the dsPIC30F processor.

EXAMPLE 7-12: .align

```
.bss
    .align 8
Array3:.space 6            Allocating space (in bytes) to variable
```

7.3 EXAMPLE CODE

Having learned about the directives and the format of the instructions, we can now look at an example to see how it all works. Here is an explanation of the code in the Flash LEDs with dsPIC30F6014.s file used in the previous chapters.

Note: The other tutorial files, Flash LEDs with dsPIC30F6012.s, Flash LED with dsPIC30F2010.s and Flash LEDs with dsPIC30F4011.s are very similar and the descriptions below are applicable.

7.3.1 Code Description

The Flash LEDs with dsPIC30F6014.s file starts with comments. In this file, the comments explain the license agreement and what the program is doing (flashing the LEDs depending on the state of switch SW1). Comments are always preceded by a semicolon (;) or, alternatively, C-style block comments (/* */).

The comments are followed by a definition of the `__30F6014` label to allow the include file to check that the correct processor is being used. The standard include file is included to define all the bits in the various SFRs.

EXAMPLE 7-13:

```
=====
; Use Timer 1 to flash LED1 when switch SW1 is not pressed
;   and flash LED2 when switch SW1 is pressed
;=====

.equ    __30F6014, 1
.include "p30f6014.inc"
```

Getting Started with dsPIC30F Digital Signal Controllers

The next section contains global declarations of the `__reset` label and the various error trap labels. This allows the linker to determine the correct address to place in the `goto` instruction at the Reset vector. It also allows the linker to determine the addresses to place in the Interrupt Vector Table for the error trap routines. (Refer to **Section 1.2.10 “Interrupts”** for more information about the error traps and the Interrupt Vector Table.)

EXAMPLE 7-14:

```
;-----  
;Global Declarations  
  
    .global __reset           ;The label for the first line of code  
    .global __OscillatorFail  ;Declare Oscillator Fail trap routine label  
    .global __AddressError    ;Declare Address Error trap routine label  
    .global __StackError      ;Declare Stack Error trap routine label  
    .global __MathError       ;Declare Math Error trap routine label
```

In the next section, the configuration bits are defined so that the processor will be programmed with the correct oscillator mode, Watchdog Timer settings, etc. The Configuration register addresses, such as `__FOSC` and `__FWD`, are defined in the linker script file, `p30f6014.gld`. The Configuration register values, such as `CSW_FSCM_OFF` and `XT_PLL4`, are defined in the processor include file, `p30f6014.inc`.

EXAMPLE 7-15:

```
;-----  
;Configuration bits  
  
    config __FOSC, CSW_FSCM_OFF & XT_PLL4  
    config __FWD, WDT_OFF  
    config __FBORPOR, PBOR_OFF & BORV_27 & PWRT_16 & MCLR_EN  
    config __FGS, CODE_PROT_OFF
```

The `Fcy` label is equated to a value so that frequency changes can easily be made. A single line can now be changed to adapt the code to different oscillator options.

EXAMPLE 7-16:

```
;-----  
;Program Specific Constants (literals used in code)  
  
    .equ    Fcy, #7372800      ;Instruction cycle rate (Osc x PLL / 4)
```

A `.text` directive tells the assembler that the code that follows should be placed in the default code section.

EXAMPLE 7-17:

```
;=====  
;Start of code  
  
    .text                      ;Start of Code section
```

The linker recognizes `__reset:` as a standard label and adds code to branch to it after a Reset. It must be global for the linker to use the label.

After allocating all the RAM variables, the linker finds the largest available space for the stack and assigns the start address to the `__SP_init` label. The code loads this label into the Stack Pointer register, W15, and this sets up the software stack. Notice the “#” sign that indicates a literal value.

The linker also provides the end address of the available stack space and the code loads this value from the `__SPLIM_init` label into the Stack Pointer Limit register, SPLIM. This sets up error checking for stack overflows. A stack error will occur if the stack pointer, W15, equals the address in SPLIM. Notice that this operation is done in two instructions. The W0 register is loaded with the `__SPLIM_init` value and this is then moved into the SPLIM register. It is not possible to code a 16-bit literal value and a 13-bit near memory address into one 24-bit instruction.

EXAMPLE 7-18:

```

;-----
;Initialize stack pointer and limit register

__reset:  mov    #__SP_init, W15      ;Initialize the Stack Pointer register
          mov    #__SPLIM_init, W0    ;Get address at the end of stack space
          mov    W0, SPLIM            ;Load the Stack Pointer Limit register
          nop                          ;Add NOP to follow SPLIM initialization

```

After setting up the stack, the code initializes an I/O port to drive the LEDs on PORTD. The LEDs are on bits 0 to 3 of PORTD and turn on when the pins are driven low. The code sets these bits in the Port Latch register, LATD, so that when the I/O pins are turned into outputs, the LEDs will be off. The code then clears these same bits in the Port Tri-State register, TRISD, so that the I/O pins are turned into outputs. Finally, the code clears bit 0 of LATD to turn one LED on.

EXAMPLE 7-19:

```

;-----
;Initialize LED outputs on PORTD bits 0-3

          mov    #0xffff, W0          ;Initialize LED pin data to off state
          mov    W0, LATD
          mov    #0xfff0, W0          ;Set LED pins as outputs
          mov    W0, TRISD
          bclr   LATD, #0              ;Turn LED1 on

```

Getting Started with dsPIC30F Digital Signal Controllers

Timer1 is initialized for a 1/5-second period so that it will flash cheerfully. The code clears the Timer1 Control register, T1CON, to stop the timer and the Timer1 Count register, TMR1, is cleared so that it will start counting from zero.

The Timer1 Period register, PR1, is loaded with the number of counts in 1/5 second. The assembler calculates this value because we have specified the instruction rate (FCY), divided by the prescaler value (256), multiplied by the time (1/5 second). The prescaler divides the clock rate that increments the timer.

Finally, the Timer1 Control register, T1CON, is written to turn the timer on and to use an internal clock source with a 1:256 prescaler.

EXAMPLE 7-20:

```
-----  
;Initialize Timer1 for 1/5 second period  
  
    clr    T1CON          ;Turn off Timer1 by clearing control register  
    clr    TMR1          ;Start Timer1 at zero  
    mov    #FCY/256/5, W0 ;Get period register value for 1/5 second  
    mov    W0, PR1       ;Load Timer1 period register  
    mov    #0x8030, W0    ;Get Timer1 settings (1:256 prescaler)  
    mov    W0, T1CON      ;Load Timer1 settings into control register
```

The main code loop starts with a label, `MainLoop`:. The Timer1 Interrupt Flag bit, T1IF, in the IFS0 register is tested to see whether the timer count has reached the Period register value. If the period has not yet elapsed, then the code branches back to `MainLoop`.

Once Timer1 has set the T1IF bit, the code skips over the branch instruction so that it can go and flash an LED. The T1IF bit is cleared so that it can be used again to detect the end of the next timer period.

The switch, SW1, is connected to the RA12 pin, so the code tests bit 12 of the PORTA register to see if the switch is being pressed. If the switch is pressed, the branch instruction is executed to go toggle LED2; otherwise, the code skips over the branch instruction and toggles LED1 instead.

LED1 is connected to pin RD0, so bit 0 of LATD is toggled to turn the LED on or off. LED2 is turned off by clearing bit 1 of LATD in case the LED was lit. Notice that when an I/O port is used as an input, the PORTx register is used and when a port is used as an output, the LATx register is used. After changing either of the LEDs, the code branches back to `MainLoop`.

EXAMPLE 7-21:

```
-----  
;Loop while waiting for a Timer1 match and toggle LED1 or LED2 when it happens  
  
MainLoop:    btss    IFS0, #T1IF          ;Check if Timer1 interrupt flag is set  
             bra     MainLoop            ;Loop back until set  
             bclr    IFS0, #T1IF         ;Clear Timer1 interrupt flag  
             btss    PORTA, #12          ;Test switch SW1 (low when pressed)  
             bra     SwitchPressed  
  
             btg     LATD, #0            ;Toggle LED1 when SW1 is not pressed  
             bset    LATD, #1            ;Turn off LED2  
             bra     MainLoop            ;Loop back  
  
SwitchPressed: bset    LATD, #0          ;Turn off LED1  
             btg     LATD, #1            ;Toggle LED2 when SW1 is pressed  
             bra     MainLoop            ;Loop back
```

The error trap routines follow the rest of the code. If the code fails due to a catastrophic error, such as an oscillator failure or a branch to non-existent memory, then the hardware will switch the execution to the appropriate error trap routine. Each routine has a global address label, such as `__OscillatorFail:` and the linker uses these addresses to create the Interrupt Vector Table. Each of these error trap routines turns on an LED and loops endlessly.

EXAMPLE 7-22:

```

;=====
;Error traps

;-----
;Oscillator Fail Error trap routine

    .text                                ;Start of Code section
__OscillatorFail:
    bclr    LATD, #3                    ;Turn LED4 on
    bra     __OscillatorFail            ;Loop forever when oscillator failure occurs

;-----
;Address Error trap routine

__AddressError:
    bclr    LATD, #3                    ;Turn LED4 on
    bra     __AddressError              ;Loop forever when address error occurs

```

After the error trap routines, there is an `.end` directive that indicates that there is no more code to be assembled in this file.

EXAMPLE 7-23:

```

.end                                ;End of code in this file

```

The assembler always generates object files that need to be linked. To learn about the LINK30 linker and how it takes the code and data from the object files and creates the final output files, please jump ahead to **Chapter 9. “The MPLAB LINK30 Linker”**. If you are going to use the MPLAB C30 compiler, then proceed to **Chapter 8. “MPLAB C30 C Compiler”**.

NOTES:

Chapter 8. MPLAB C30 C Compiler

8.1 MPLAB C30 C COMPILER OVERVIEW

Many of you writing software for the dsPIC devices will do so in 'C'. MPLAB C30 C Compiler is an ANSI-compliant C compiler that allows you to write uniform, modular code for the dsPIC digital signal controller that is more portable and easier to understand than assembly. In addition to the advantage of the 'C' language itself, the libraries offered by MPLAB C30 C Compiler make it a powerful compiler. For example, implementing floating point, trigonometrical functions, filters and FFT algorithms can be quite cumbersome in the assembly language. But with the DSP, peripheral and standard math libraries, these routines can be called easily. The modularity of the 'C' language reduces the likelihood of functions interacting.

The intent of this chapter is not to describe the ins and outs of the C language, but rather what you'll need to know to be able to get up and running quickly. For detailed information on the operation of MPLAB C30 C Compiler, refer to the "MPLAB® C30 C Compiler User's Guide" (DS51284).

Note: This chapter is based on MPLAB C30 C Compiler version 1.30. Some information may become dated as new versions are released.

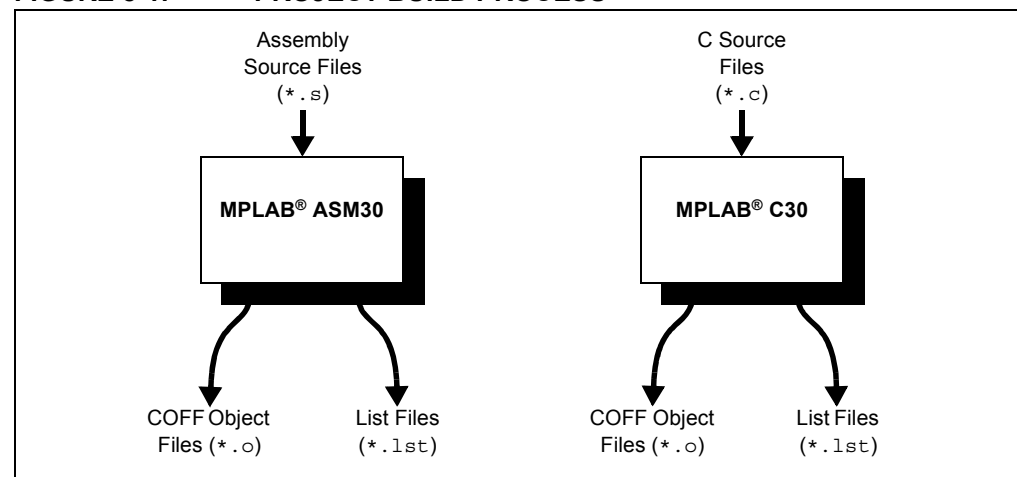
8.2 MPLAB C30 C COMPILER PROJECTS

You learned about MPLAB projects in **Chapter 3. "MPLAB Integrated Development Environment"**, but only used assembly (.s) source files. You will now see that MPLAB C30 C Compiler projects are very similar, but use C language (.c) source files and archive (.a) library files as well.

Recall that building an assembly project is a two-step process. The assembly source (.s) files are each assembled to create object (.o) files and then the object files are linked to create output (.hex and .cof) files, as shown in Figure 8-1.

Building a MPLAB C30 C Compiler project is also a two-step process in which 'C' source (.c) files are compiled to object files and the object files are linked to create output files.

FIGURE 8-1: PROJECT BUILD PROCESS



In addition to 'C' files, the project may include library files that are linked together with the object files. The libraries are created from precompiled object files and are essentially functions that can be used in the project without the need to be compiled.

The project also includes a linker script file for the LINK30 linker. For more information on the linker, please read **Chapter 9. "The MPLAB LINK30 Linker"**.

8.3 CREATING A PROJECT WITH THE PROJECT WIZARD

If you have not already done so, please read **Chapter 3. "MPLAB Integrated Development Environment"** to find out about projects and workspaces in MPLAB IDE. We will forgo any repetition and jump right into creating an MPLAB C30 C Compiler project.

First, install the MPLAB C30 C Compiler. The tutorial that follows assumes that it has been installed to the default location `C:\Program Files\Microchip\MPLAB C30`. If you install it elsewhere, please adjust the paths in the tutorial accordingly.

Note: The tool locations for your environment may be different from those shown here.

Before starting, create a folder for the project files for this tutorial. The folder `C:\Tutorial` is being used in the instructions that follow. If you have already created this folder for a previous tutorial, you can simply add the new file into the folder. Copy the `Flash LEDs with dsPIC30F6014.c` file into the `C:\Tutorial` folder. The files are supplied with this document. If the file is copied from a CD, it has read-only attributes; remember to change the attributes if the file needs to be edited.

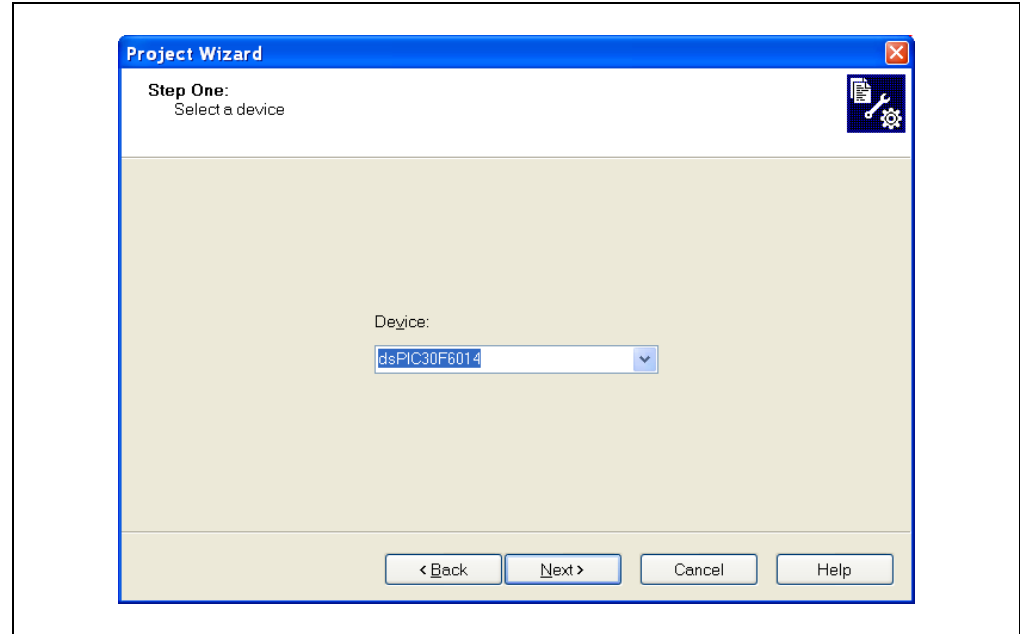
Note: If you have a dsPICDEM Starter Demonstration Board, you can use the `Flash LEDs with dsPIC30F6012.c` file instead; it contains very similar code. For the dsPICDEM 28-Pin Starter Demonstration Board, you can use the `Flash LED with dsPIC30F2010.c` file. For the dsPICDEM 2 Development Board, you can use the `Flash LED with dsPIC30F4011.c` file.

Now, start MPLAB IDE and close any open workspace with the File>Close Workspace menu. The Project Wizard in MPLAB IDE is an easy way to create new projects. Use the Project>Project Wizard menu to start the Project Wizard. When the Welcome screen appears, click **Next>** to continue.

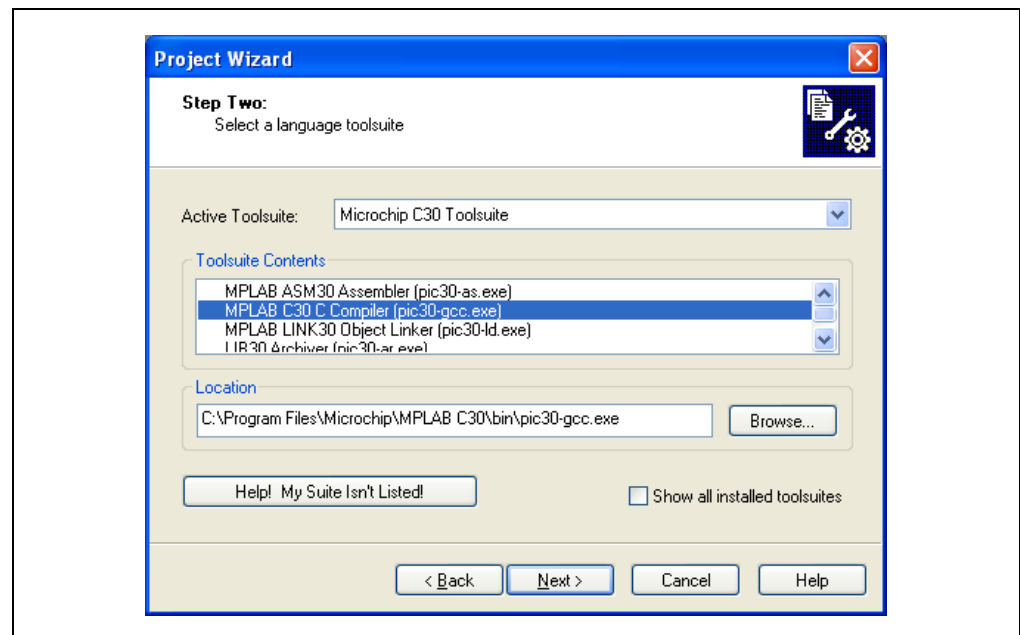
Step 1 – Select a Device

The Project Wizard wants you to select a dsPIC device to work with, as shown in Figure 8-2. Select "dsPIC30F6014" from the pull-down menu, then click **Next>** to continue.

Note: If you are using a different demo board, select the dsPIC30F6012, dsPIC30F2010 or dsPIC30F4011, as appropriate.

FIGURE 8-2: PROJECT WIZARD – STEP ONE**Step 2 – Select a Language Toolsuite**

The next screen (Figure 8-3) expects you to select the toolsuite. Select the “Microchip C30 Toolsuite” from the pull-down menu.

FIGURE 8-3: PROJECT WIZARD – STEP TWO

Check that the executables for the assembler, compiler and linker are at the following locations:

Assembler: C:\Program Files\Microchip\MPLAB C30\bin\pic30-as.exe
 Compiler: C:\Program Files\Microchip\MPLAB C30\bin\pic30-gcc.exe
 Linker: C:\Program Files\Microchip\MPLAB C30\bin\pic30-ld.exe
 Archiver: C:\Program Files\Microchip\MPLAB C30\bin\pic30-ar.exe

Getting Started with dsPIC30F Digital Signal Controllers

These tool locations assume that the MPLAB C30 C Compiler was installed with the default settings.

Note: A red 'X' appears next to toolsuite locations that are missing information.

If a red 'X' appears next to a toolsuite, select the toolsuite and click on the **Browse** button to set the location. Once the toolsuite has been selected and the locations are correct, click **Next** to continue.

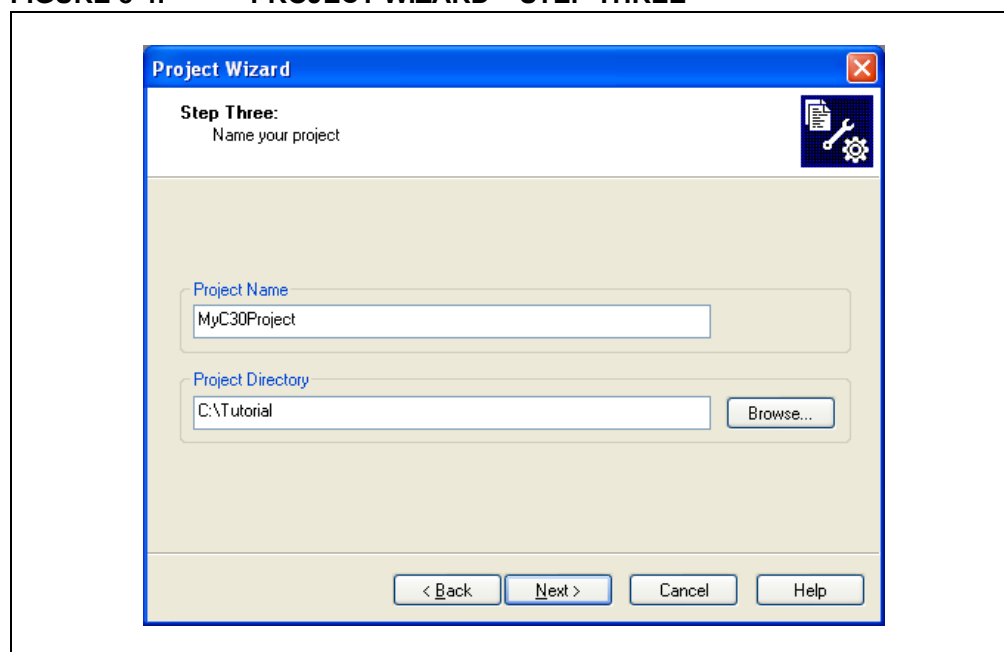
Step 3 – Name your Project

The next screen (Figure 8-4) wants you to name the project.

Type "MyC30Project" for the project name and browse to, or type C:\Tutorial for the project directory.

Click **Next>** to continue.

FIGURE 8-4: PROJECT WIZARD – STEP THREE



Step 4 – Adding Files to the Project

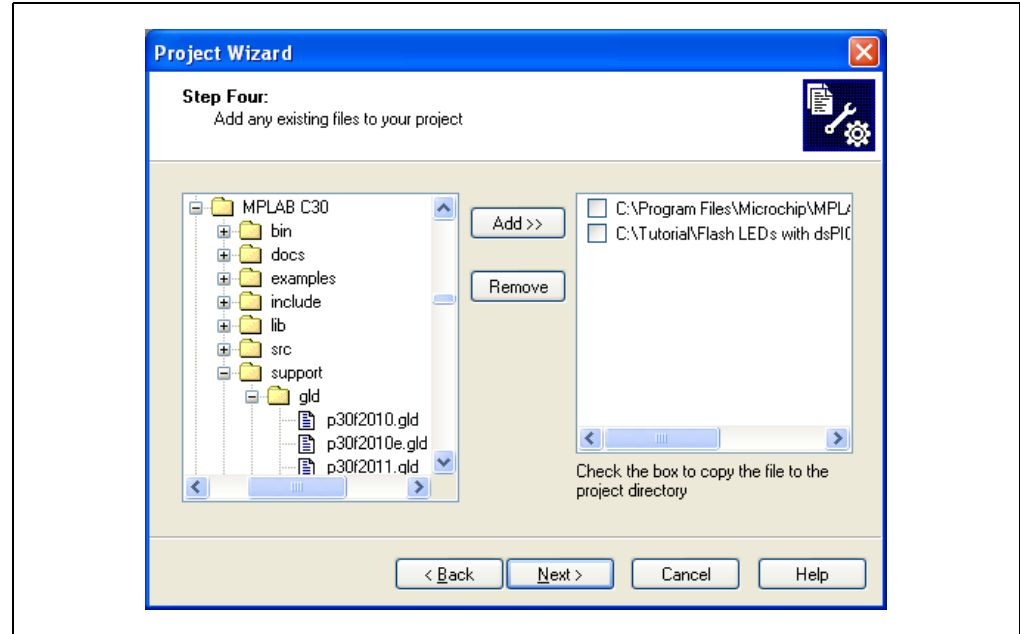
The next screen allows you to add files to the project.

Select the Flash LEDs with dsPIC30F6014.c file from the Tutorial folder and click **Add>>** to include this file in the project.

Navigate to the C:\Program Files\Microchip\MPLAB C30\Support\gld folder. Select the p30f6014.gld file and click **Add>>** to include the file in the project.

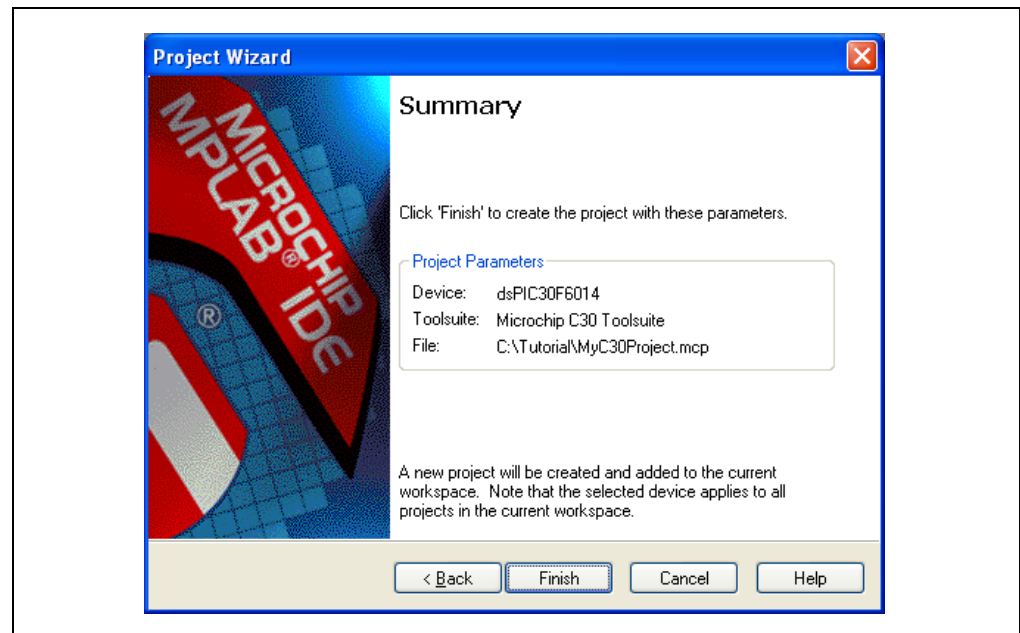
There should now be two files in the project, as shown in Figure 8-5. Click **Next>** to continue.

FIGURE 8-5: PROJECT WIZARD – STEP FOUR



The Project Wizard Summary screen displays the Project Parameters you have just set up, as shown in Figure 8-6.

FIGURE 8-6: PROJECT WIZARD SUMMARY SCREEN



Click **Finish**.

Note: If you are using the dsPICDEM Starter Demonstration Board, select the files applicable to the dsPIC30F6012 instead. If you are using the dsPICDEM 28-Pin Starter Demonstration Board, select the files applicable to the dsPIC30F2010. If you are using the dsPICDEM 2 Development Board, select the files applicable to the dsPIC30F4011.

After the Project Wizard completes, MPLAB IDE will have a Project window showing the Flash LEDs with dsPIC30F6104.c file in the Source Files category and the p30f6014.gld file in the Linker Scripts category. The GLD file is described in much greater depth in **Chapter 9. “The MPLAB LINK30 Linker”**.

If you realize that you have forgotten to add files to your project, you don't have to restart the Project Wizard. Simply right click on a category in the project tree, select “Add Files” from the drop-down menu and browse to the file you want to add. You can remove files by right clicking on the file name and selecting **Remove**.

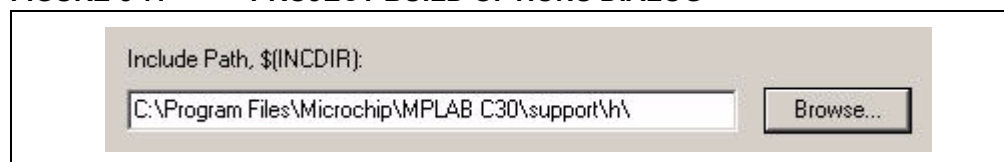
A project file, MyC30Project.mcp, and workspace file, MyC30Project.mcw, have now been created by the MPLAB IDE. Double click the Flash LEDs with dsPIC30F6014.c file in the Project window to open the file. The file displays in the Editor window.

8.4 SETTING THE BUILD OPTIONS

Before you can build your project, you will need to set the build options. These settings are used by MPLAB IDE to locate files, generate debugging information, control optimization and create diagnostic files.

Use the *Project>Build Options>Project* menu to tell MPLAB IDE where to find the header files. When the Build Options screen displays, click on the **Browse** button for the “Include Path” and browse to C:\Program Files\Microchip\MPLAB C30\support\h. Click **OK** to add this path, as shown in Figure 8-7.

FIGURE 8-7: PROJECT BUILD OPTIONS DIALOG

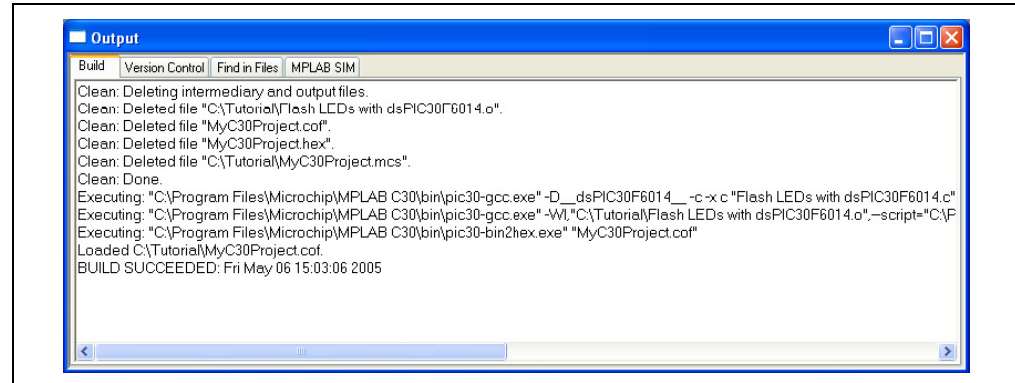


8.5 BUILDING THE PROJECT

The project is now ready to be built. Select the *Project>Make* menu. The results of the build will appear in the Output window and this should indicate that the build succeeded, as shown in Figure 8-8.

Depending on your Project Build Options, you may see a Memory Usage Report in your Build results.

FIGURE 8-8: BUILD STATUS



Now that you have built the project successfully, you can run or debug the code. You can go back to **Chapter 4. “The MPLAB SIM Simulator”**, **Chapter 5. “The MPLAB ICD 2 In-Circuit Debugger”** or **Chapter 6. “MPLAB ICE 4000 In-Circuit Emulator”**.

8.6 LANGUAGE FEATURES

8.6.1 ANSI C Standard

The MPLAB C30 C Compiler is an ANSI C compiler with various extensions to support specific features and capabilities of the dsPIC devices. The compiler includes a complete ANSI C standard library and there is an optimizer to generate efficient compact code.

8.6.2 Standard Header File

A standard header file, such as `p30f6014.h` used in our example code, should always be included in each ‘C’ file with an `#include` statement. The header file contains the declarations for all the Special Function Registers (SFRs) and their bits so that they can be used in the code. The linker obtains the addresses of the SFRs from the linker script file, `p30f6014.gld`, in our example.

8.6.3 `__attribute__` keyword

The MPLAB C30 C Compiler uses the `__attribute__` keyword (note the double underscore prefix and suffix) to indicate compiler specific actions for functions and variables that cannot be done with standard ‘C’ syntax. It can be used to define sections, control how program memory is used, optimize functions, specify interrupt functions and so forth. It is similar to the `#pragma` directive used in some other compilers, such as MPLAB C18 C Compiler.

8.7 EXAMPLE CODE

Having learned more about the compiler, you can now look at an example to see how it all works. Here is an explanation of the example code in the Flash LEDs with dsPIC30F6014.c file used in the tutorial.

Note: The other tutorial files, Flash LEDs with dsPIC30F6012.c, Flash LED with dsPIC30F2010.c and Flash LED with dsPIC30F4011.c are very similar and the descriptions below are applicable.

The file starts with comments, preceded by “/*” or “//” (see Example 8-1). The standard header file is then included to define all the Special Function Registers (SFRs). There is a header file for each dsPIC controller supported by the MPLAB C30 C Compiler.

EXAMPLE 8-1: COMMENTS AND HEADER FILE REFERENCE

```
/******  
*                               Software License Agreement                               *  
*                                                                                       *  
*   The software supplied herewith by Microchip Technology Incorporated                 *  
*   (the "Company") for its dsPIC controller is intended and supplied to               *  
*   you, the Company's customer, for use solely and exclusively on                     *  
*   Microchip dsPIC products. The software is owned by the Company and/or            *  
*   its supplier, and is protected under applicable copyright laws. All               *  
*   rights are reserved. Any use in violation of the foregoing                       *  
*   restrictions may subject the user to criminal sanctions under                     *  
*   applicable laws, as well as to civil liability for the breach of the               *  
*   terms and conditions of this license.                                             *  
*                                                                                       *  
*   THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES,                *  
*   WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO,            *  
*   IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR               *  
*   PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT, IN ANY                   *  
*   CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL                 *  
*   DAMAGES, FOR ANY REASON WHATSOEVER.                                              *  
*                                                                                       *  
*****/  
//=====   
// Use Timer 1 to flash LED1 when switch SW1 is not pressed   
//   and flash LED2 when switch SW1 is pressed   
//=====   
  
#include "p30F6014.h"
```

The configuration bits are defined using macros to declare the configuration bit settings, as shown in Example 8-2. This ensures that the configuration settings will be included in the output hex file.

EXAMPLE 8-2: CONFIGURATION BITS

```
//-----   
//Configuration bits   
  
_FOSC(CSW_FSCM_OFF & XT_PLL4);   
_FWDTP(WDT_OFF);   
_FBORPOR(PBOR_OFF & BORV_27 & PWRT_16 & MCLR_EN);   
_FGS(CODE_PROT_OFF);
```

The processor frequency is defined in order to set up a timer later in the code. The symbol, `Fcy`, is given a value in a `#define` statement, as shown in Example 8-3.

EXAMPLE 8-3: PROCESSOR FREQUENCY DEFINITION

```
//-----  
//Program Specific Constants  
  
#define Fcy 7372800                //Instruction cycle rate (Osc x PLL / 4)
```

The `Main()` routine starts the executable code and has an integer return value in accordance with the ANSI 'C' standard. It is important to note that the linker will add start-up code that calls the `Main` routine. The first part of the `Main` routine (Example 8-4) sets up the I/O pins for the LEDs by initializing the output latch and turning the LED pins into outputs.

EXAMPLE 8-4: BEGINNING OF MAIN ROUTINE

```
//-----  
//Main routine  
//Set up LEDs and timer, wait for timer periods, and flash one of the two LEDs  
  
int main(void)  
{  
    LATD = 0xffff;                //Initialize LED pin data to off state  
    TRISD = 0xffff0;              //Set LED pins (PORTD bits 0-3) as outputs  
    LATDbits.LATD0 = 0;           //Turn LED1 on (active low)
```

The code then initializes `Timer1` for a 1/5-second period and turns the timer on, as shown in Example 8-5.

EXAMPLE 8-5: TIMER INITIALIZATION

```
T1CON = 0;                        //Turn off Timer1 and clear settings  
TMR1 = 0;                        //Start Timer1 at zero  
PR1 = Fcy/256/5;                 //Set period register value for 1/5 second  
T1CON = 0x8030;                  //Turn on Timer1 with 1:256 prescaler
```

Inside the `Main()` routine, there is a `while(1)` loop to ensure that code execution never leaves `main` and stops. Inside the infinite loop, the code waits for the `Timer1` interrupt flag to be set (see Example 8-6). The interrupt flag is set by the hardware when the timer matches the Period register, even though interrupts are not enabled.

EXAMPLE 8-6: WHILE LOOP

```
while(1)                          //Loop forever  
{  
    while(!IFS0bits.T1IF){}        //Wait for timer period
```

After the timer period has elapsed, the code clears the interrupt flag to be ready to test it again later. The `RA12` input pin is tested to see if switch `SW1` has been pressed. Depending on the state of the input pin, one of the LEDs is toggled and the other is turned off. As shown in Example 8-7, the code that waits for the timer and toggles the LED repeats endlessly.

Getting Started with dsPIC30F Digital Signal Controllers

EXAMPLE 8-7: MAIN ROUTINE

```
IFS0bits.T1IF = 0;           //Clear timer flag for next period
if(PORTAbits.RA12)           //Check if SW1 is pressed
{
    LATDbits.LATD0 ^= 1;      //Toggle LED1 when SW1 not pressed
    LATDbits.LATD1 = 1;       //Turn off LED2
}
else
{
    LATDbits.LATD0 = 1;       //Turn off LED1
    LATDbits.LATD1 ^= 1;      //Toggle LED2 when SW1 is pressed
}
}

//End of main()
```

Next are the error trap routines, as shown in Example 8-8. If the code fails due to a catastrophic error, such as an oscillator failure or a branch to non-existent memory, the hardware switches the execution to the appropriate error trap routine. Each routine has a function name, such as `_OscillatorFail`, that is recognized and used by the linker to create the Interrupt Vector Table. The error trap routines turn on an LED and loop endlessly.

EXAMPLE 8-8: ERROR TRAP ROUTINES

```
//=====
//Error traps
//-----
//Oscillator Fail Error trap routine

void _ISR _OscillatorFail(void)
{
    LATDbits.LATD3 = 0;        //Turn LED4 on
    while(1);                  //Wait forever
}
//-----
//Address Error trap routine

void _ISR _AddressError(void)
{
    LATDbits.LATD3 = 0;        //Turn LED4 on
    while(1);                  //Wait forever
}
//-----
//Stack Error trap routine

void _ISR _StackError(void)
{
    LATDbits.LATD3 = 0;        //Turn LED4 on
    while(1);                  //Wait forever
}
//-----
//Math (Arithmetic) Error trap routine

void _ISR _MathError(void)
{
    LATDbits.LATD3 = 0;        //Turn LED4 on
    while(1);                  //Wait forever
}
```

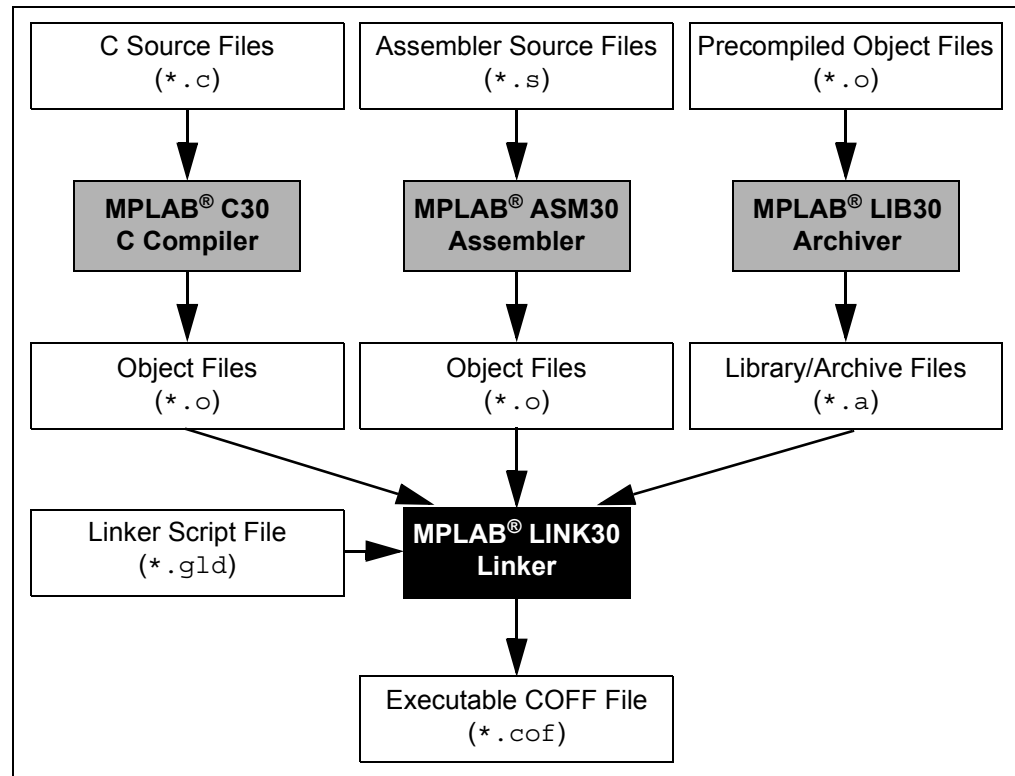
This simple code example and the tutorial should explain the basics of using the MPLAB C30 C Compiler. Remember that the compiler generates object files and the MPLAB LINK30 Linker uses the object files to place the code and variables into memory and generate the output files. To learn more about the linker, please proceed to **Chapter 9. “The MPLAB LINK30 Linker”**.

Chapter 9. The MPLAB LINK30 Linker

9.1 MPLAB LINK30 LINKER OVERVIEW

The MPLAB LINK30 linker translates object files and archive (library) files and combines them into the executable program that runs the dsPIC device. Object files are created by the MPLAB ASM30 Assembler and MPLAB C30 C Compiler. Archive files are created by the MPLAB[®] LIB30 Archiver/Librarian. As illustrated in Figure 9-1, these files are then translated and linked to form a Common Object File Format (COFF) file that actually controls the dsPIC device.

FIGURE 9-1: MPLAB[®] LINK30 LINKER OPERATION



The linker essentially links all compiled and assembled files in the project together, to form one executable file that can be programmed into a part, simulated or emulated. The hex file and map file are created from the COFF file.

Note: This chapter is based on MPLAB LINK30 Linker version 1.31. Some information may become dated as new versions are released.

9.2 LINKER SCRIPT FILES

The linker uses a linker script file to know where sections of memory are located and to know the memory ranges implemented on a specific part. The linker script file supports the construction of Interrupt Vector Tables and the allocation of the software stack. It also assigns the addresses of the Special Function Registers (SFRs).

The linker script file has the following categories of information that we will briefly discuss in the rest of this chapter by looking at the `p30f6014.gld` file as an example:

- Output File Format and Entry Point
- Memory Region Information
- Base Memory Address
- Input/Output Section Map
- Range Checking for Near and X Data Memory
- Interrupt Vector Tables
- SFR Addresses

9.2.1 Output File Format and Entry Point

The first several lines of a linker script define the output format, processor family and entry point:

EXAMPLE 9-1:

```
/*
** Linker Script for p30f6014
*/

OUTPUT_FORMAT("coff-pic30")
OUTPUT_ARCH("pic30")
EXTERN(__resetPRI)
EXTERN(__resetALT)
ENTRY(__reset)
```

Notice that the entry label is `__reset`. If you have a global label called `__reset` in your code, that will be where the code starts executing.

9.2.2 Memory Region Information

The next section of the linker script file defines the various memory regions for the device. The information in this section tells the linker how much memory is available on the device. Each memory region is range checked as sections are added during the link process. If any region overflows, a link error is reported.

EXAMPLE 9-2:

```
/*
** Memory Regions
*/
MEMORY
(
    data (a!xr)      : ORIGIN = 0x800,    LENGTH = 8096
    program (xr)      : ORIGIN = 0x100,    LENGTH = ((48K * 2) - 0x100)
    reset             : ORIGIN = 0,        LENGTH = (4)
    ivt               : ORIGIN = 0x04,     LENGTH = (62 * 2)
    aivt              : ORIGIN = 0x84,     LENGTH = (62 * 2)
    __FOSC             : ORIGIN = 0xF80000, LENGTH = (2)
    __FWDT             : ORIGIN = 0xF80002, LENGTH = (2)
    __FBORPOR          : ORIGIN = 0xF80004, LENGTH = (2)
    __CONFIG4          : ORIGIN = 0xF80006, LENGTH = (2)
    __CONFIG5          : ORIGIN = 0xF80008, LENGTH = (2)
    __FGS              : ORIGIN = 0xF8000A, LENGTH = (2)
    eedata             : ORIGIN = 0x7FF000, LENGTH = (4096)
)
```

9.2.3 Base Memory Address

This portion of the linker script defines the starting addresses of several sections into which the linker will place code or data. Each base address is defined as a symbol and the symbols are used to specify load addresses in the section map that follows.

EXAMPLE 9-3:

```
/*
** Base Memory Addresses - Program Memory
*/
__RESET_BASE = 0:      /* Reset Instruction */
__IVT_BASE   = 0x04;   /* Interrupt Vector Table */
__AIVT_BASE  = 0x84;   /* Alternate Interrupt Vector Table */
__CODE_BASE  = 0x100;  /* Handles, User Code, Library Code */

/*
** Base Memory Addresses - Data Memory
*/
__SFR_BASE   = 0;      /* Memory-mapped SFRs */
__DATA_BASE  = 0x800;   /* X and General Purpose Data Memory */
__YDATA_BASE = 0x1800; /* Y Data Memory for DSP Instructions */
```

9.2.4 Input/Output Section Map

The section map is the heart of the linker script file. It defines how input sections are mapped to output sections. Note that input sections are portions of an application that are defined in source code, while output sections are created by the linker. Generally, several input sections may be combined into a single output section.

For example, suppose that an application consists of five different functions and each function is defined in a separate source file. Together, these source files produce five input sections. The linker combines these input sections into a single output section. Only the output section has an absolute address. Input sections are always relocatable.

If any input or output sections are empty, there is no penalty or storage cost for the linked application. Most applications will use only a few of the many sections that appear in the section map.

This is how the section map starts:

EXAMPLE 9-4:

```
/*
===== Section Map =====
*/

SECTIONS
(
```

Consider the first section of program memory in Example 9-5. The program memory starts at address zero (`__RESET_BASE` is defined as a base memory address in Example 9-3) and there is space for a two-word instruction before the Interrupt Vector Table starts. The section is loaded with data to form a two-word `GOTO __reset` instruction. You can look at the encoding for a `GOTO` instruction in the “*dsPIC30F Programmer’s Reference Manual*” (DS70030) to see how the instruction has been constructed.

EXAMPLE 9-5:

```
/*
===== Program Memory =====
*/

/*
** RESET Instruction
*/
.reset __RESET_BASE:
(
    SHORT(ABSOLUTE(__reset));
    SHORT(0x04);
    SHORT(ABSOLUTE(__reset) >> 16) & 0x7F;
    SHORT(0);
) >reset
```

The `.text` section collects executable code input sections from all of the application's input files and puts them into one output section. The order of some input sections is defined to ensure proper operation of the MPLAB C30 C Compiler. For example, the `.handle` section is used for function pointers and is loaded first. This is followed by the library sections, `.libc`, `.libm` and `.libdsp`. The math library is in the middle so that it can be called efficiently from the standard 'C' library as well as the DSP library. Other libraries are then followed by the rest of the code.

EXAMPLE 9-6:

```
/*
** User Code and Library Code
*/
.text __CODE_BASE:
{
    *(.handle);
    *(.libc) *(.libm) *(.libdsp); /* Keep together in this order*/
    *(.lib*);
    *(.text);
} >program
```

The rest of the section maps follow, to define all the different types of program memory, RAM, EEPROM and configuration memory sections.

Note: It is possible to create your own user-defined output sections in program and data memory. There are examples showing how to do this in the linker script files.

9.2.5 Range Checking for Near and X Data Memory

Range check expressions are included for the X Data Memory space and the Near Data Memory space. Range checking for all other sections is provided as the memory regions are filled. A link error will be reported if any section extends beyond its assigned memory region.

Note that the X Data Memory space limit varies by device, while the Near Data Memory space limit is fixed at 8 Kbytes, or address 0x2000.

EXAMPLE 9-7:

```
/*
** Calculate overflow of X and Near data space
*/
__X_OVERFLOW      =      (((__exdata != __bxdata) &&
                          (__exdata > __YDATA_BASE)) ?
                          (__exdata - __YDATA_BASE) : 0);
__NEAR_OVERFLOW    =      (((__endata != __bndata) &&
                          (__endata > 0x2000)) ?
                          (__endata - 0x2000) : 0);
```

9.2.6 Interrupt Vector Tables

The primary and alternate Interrupt Vector Tables are defined in a second section map, near the end of the linker script file. Here is a simple explanation of the table, using the oscillator fail error trap as an example:

If the symbol, `__OscillatorFail`, is defined, the address of that symbol is used; otherwise, the address of symbol, `__DefaultInterrupt`, is used instead. This means that if you have not provided an interrupt routine, then a default routine will be called. If you have not provided a default interrupt handler (a function with the name `__DefaultInterrupt`), then the linker will generate one automatically. The simplest default interrupt handler is a `RESET` instruction.

EXAMPLE 9-8:

```
/*
** Section Map for Interrupt Vector Tables
*/
SECTIONS
{

/*
** Primary Interrupt Vector Table
*/
.ivt __IVT_BASE:
{
    LONG(DEFINED(__ReservedTrap0) ? ABSOLUTE(__ReservedTrap0) :
        ABSOLUTE(__DefaultInterrupt));
    LONG(DEFINED(__OscillatorFail) ? ABSOLUTE(__OscillatorFail) :
        ABSOLUTE(__DefaultInterrupt));
    LONG(DEFINED(__AddressError) ? ABSOLUTE(__AddressError) :
        ABSOLUTE(__DefaultInterrupt));
    LONG(DEFINED(__StackError) ? ABSOLUTE(__StackError) :
        ABSOLUTE(__DefaultInterrupt));
    LONG(DEFINED(__MathError) ? ABSOLUTE(__MathError) :
        ABSOLUTE(__DefaultInterrupt));
    .
    .
    .
} >ivt
```

9.2.7 SFR Addresses

Absolute addresses for the Special Function Registers (SFRs) are defined as a series of symbol definitions. Two versions of each SFR address are included, with and without a leading underscore. This is to enable both 'C' and assembly language programmers to refer to the SFR using the same name. By convention, the C compiler adds a leading underscore to every label.

EXAMPLE 9-9:

```

/*=====
**      Register Definitions
** (Core and Peripheral Registers in Data Space)
**=====
**
**=====
**      dsPIC Core Register Definitions
**
**=====
_WREG0 = 0x0000;
_WREG0 = 0x0000;
_WREG1 = 0x0002;
_WREG1 = 0x0002;
.
.
.
_CAN1 = 0x0300;
_CAN1 = 0x0300;
_CAN2 = 0x03C0;
_CAN2 = 0x03C0;

/*=====
**end of SFR definitions required in Data Space
**=====*/

```

That may be more data than you thought you needed, but it is important not to be intimidated by the linker and its linker script file. The linker simply follows a procedure to place your code and variables in the available memory.

We hope this *“Getting Started with dsPIC30F Digital Signal Controllers User’s Guide”* has helped you become comfortable using the dsPIC30F devices and the Microchip development tools. Good luck with all your dsPIC designs.

NOTES:



Getting Started with dsPIC30F Digital Signal Controllers

Appendix A. Code for dsPICDEM 1.1 General Purpose Development Board

Software License Agreement

The software supplied herewith by Microchip Technology Incorporated (the "Company") is intended and supplied to you, the Company's customer, for use solely and exclusively with products manufactured by the Company.

The software is owned by the Company and/or its supplier, and is protected under applicable copyright laws. All rights are reserved. Any use in violation of the foregoing restrictions may subject the user to criminal sanctions under applicable laws, as well as to civil liability for the breach of the terms and conditions of this license.

THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

A.1 FLASH LED WITH dsPIC30F6014.s

This appendix contains the sample code for the dsPICDEM 1.1 General Purpose Development Board. This tutorial example was written for the MPLAB ASM30 Assembler and the MPLAB C30 C Compiler.

```
=====;
;
;           Software License Agreement
;
; The software supplied herewith by Microchip Technology Incorporated
; (the "Company") for its dsPIC controller is intended and supplied to
; you, the Company's customer, for use solely and exclusively on
; Microchip dsPIC products. The software is owned by the Company and/or
; its supplier, and is protected under applicable copyright laws. All
; rights are reserved. Any use in violation of the foregoing
; restrictions may subject the user to criminal sanctions under
; applicable laws, as well as to civil liability for the breach of the
; terms and conditions of this license.
;
; THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES,
; WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO,
; IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
; PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT, IN ANY
; CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL
; DAMAGES, FOR ANY REASON WHATSOEVER.
;
;=====;
;
; Use Timer 1 to flash LED1 when switch SW1 is not pressed
; and flash LED2 when switch SW1 is pressed
;
;=====;

.equ __30F6014, 1
.include "p30f6014.inc"

;-----;
```

Getting Started with dsPIC30F Digital Signal Controllers

```
;Global Declarations

.global __reset                ;The label for the first line of code
.global __OscillatorFail      ;Declare Oscillator Fail trap routine label
.global __AddressError        ;Declare Address Error trap routine label
.global __StackError          ;Declare Stack Error trap routine label
.global __MathError           ;Declare Math Error trap routine label

;-----
;Configuration bits

config __FOSC, CSW_FSCM_OFF & XT_PLL4
config __FWDTP, WDT_OFF
config __FBORPOR, PBOR_OFF & BORV_27 & PWRT_16 & MCLR_EN
config __FGS, CODE_PROT_OFF

;-----
;Program Specific Constants (literals used in code)

.equ          Fcy, #7372800      ;Instruction cycle rate (Osc x PLL / 4)

;=====
;Start of code

.text                            ;Start of Code section

;-----
;Initialize stack pointer and limit register

__reset:    mov     #__SP_init, W15      ;Inititalize the Stack Pointer register
            mov     #__SPLIM_init, W0    ;Get address at the end of stack space
            mov     W0, SPLIM            ;Load the Stack Pointer Limit register
            nop                          ;Add NOP to follow SPLIM initialization

;-----
;Initialize LED outputs on PORTD bits 0-3

            mov     #0xffff, W0          ;Initialize LED pin data to off state
            mov     W0, LATD
            mov     #0xffff0, W0         ;Set LED pins as outputs
            mov     W0, TRISD
            bclr    LATD, #0             ;Turn LED1 on

;-----
;Initialize Timer1 for 1/5 second period

            clr     T1CON                ;Turn off Timer1 by clearing control register
            clr     TMR1                 ;Start Timer1 at zero
            mov     #Fcy/256/5, W0       ;Get period register value for 1/5 second
            mov     W0, PR1              ;Load Timer1 period register
            mov     #0x8030, W0          ;Get Timer1 settings (1:256 prescaler)
            mov     W0, T1CON            ;Load Timer1 settings into control register

;-----
;Loop while waiting for a Timer1 match and toggle LED1 or LED2 when it happens

MainLoop:   btss    IFS0, #T1IF          ;Check if Timer1 interrupt flag is set
            bra     MainLoop             ;Loop back until set
            bclr    IFS0, #T1IF          ;Clear Timer1 interrupt flag
            btss    PORTA, #12           ;Test switch SW1 (low when pressed)
            bra     SwitchPressed
```

Code for dsPICDEM 1.1 General Purpose Development Board

```
        btg     LATD, #0                ;Toggle LED1 when SW1 is not pressed
        bset    LATD, #1                ;Turn off LED2
        bra     MainLoop                ;Loop back

SwitchPressed:bset    LATD, #0                ;Turn off LED1
        btg     LATD, #1                ;Toggle LED2 when SW1 is pressed
        bra     MainLoop                ;Loop back

;=====
;Error traps

;-----
;Oscillator Fail Error trap routine

        .text                          ;Start of Code section
__OscillatorFail:
        bclr    LATD, #3                ;Turn LED4 on
        bra     __OscillatorFail        ;Loop forever when oscillator failure occurs

;-----
;Address Error trap routine

__AddressError:
        bclr    LATD, #3                ;Turn LED4 on
        bra     __AddressError          ;Loop forever when address error occurs

;-----
;Stack Error trap routine

__StackError:
        bclr    LATD, #3                ;Turn LED4 on
        bra     __StackError            ;Loop forever when stack error occurs

;-----
;Math (Arithmetic) Error trap routine

__MathError:
        bclr    LATD, #3                ;Turn LED4 on
        bra     __MathError             ;Loop forever when math error occurs

;=====

        .end                          ;End of code in this file
```

A.2 FLASH LED WITH dsPIC30F6014.c

```
/*
 * Software License Agreement
 *
 * The software supplied herewith by Microchip Technology Incorporated
 * (the "Company") for its dsPIC controller is intended and supplied to
 * you, the Company's customer, for use solely and exclusively on
 * Microchip dsPIC products. The software is owned by the Company and/or
 * its supplier, and is protected under applicable copyright laws. All
 * rights are reserved. Any use in violation of the foregoing
 * restrictions may subject the user to criminal sanctions under
 * applicable laws, as well as to civil liability for the breach of the
 * terms and conditions of this license.
 *
 * THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES,
 * WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO,
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT, IN ANY
 * CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL
 * DAMAGES, FOR ANY REASON WHATSOEVER.
 *
 * Use Timer 1 to flash LED1 when switch SW1 is not pressed
 * and flash LED2 when switch SW1 is pressed
 *
 */

#include "p30F6014.h"

//-----
//Configuration bits

_FOSC(CSW_FSCM_OFF & XT_PLL4);
_FWDT(WDT_OFF);
_FBORPOR(PBOR_OFF & BORV_27 & PWRT_16 & MCLR_EN);
_FGS(CODE_PROT_OFF);

//-----
//Program Specific Constants

#define Fcy 7372800 //Instruction cycle rate (Osc x PLL / 4)

//=====
//Main routine
//Set up LEDs and timer, wait for timer periods, and flash one of the two LEDs

int main(void)
{
    LATD = 0xffff; //Initialize LED pin data to off state
    TRISD = 0xfff0; //Set LED pins (PORTD bits 0-3) as outputs
    LATDbits.LATD0 = 0; //Turn LED1 on (active low)

    T1CON = 0; //Turn off Timer1 and clear settings
    TMR1 = 0; //Start Timer1 at zero
    PR1 = Fcy/256/5; //Set period register value for 1/5 second
    T1CON = 0x8030; //Turn on Timer1 with 1:256 prescaler

    while(1) //Loop forever
```

```
{
    while(!IFS0bits.T1IF){}                //Wait for timer period

    IFS0bits.T1IF = 0;                      //Clear timer flag for next period
    if(PORTAbits.RA12)                     //Check if SW1 is pressed
    {
        LATDbits.LATD0 ^= 1;               //Toggle LED1 when SW1 not pressed
        LATDbits.LATD1 = 1;               //Turn off LED2
    }
    else
    {
        LATDbits.LATD0 = 1;               //Turn off LED1
        LATDbits.LATD1 ^= 1;             //Toggle LED2 when SW1 is pressed
    }
}

}                                           //End of main()

//=====
//Error traps

//-----
//Oscillator Fail Error trap routine

void _ISR _OscillatorFail(void)
{
    LATDbits.LATD3 = 0;                   //Turn LED4 on
    while(1);                             //Wait forever
}

//-----
//Address Error trap routine

void _ISR _AddressError(void)
{
    LATDbits.LATD3 = 0;                   //Turn LED4 on
    while(1);                             //Wait forever
}

//-----
//Stack Error trap routine

void _ISR _StackError(void)
{
    LATDbits.LATD3 = 0;                   //Turn LED4 on
    while(1);                             //Wait forever
}

//-----
//Math (Arithmetic) Error trap routine

void _ISR _MathError(void)
{
    LATDbits.LATD3 = 0;                   //Turn LED4 on
    while(1);                             //Wait forever
}
```

NOTES:

Appendix B. Code for dsPICDEM Starter Demonstration Board

B.1 FLASH LED WITH dsPIC30F6012.s

This appendix contains the sample code for the dsPICDEM Starter Demonstration Board. This tutorial example was written for the MPLAB ASM30 Assembler and the C30 C Compiler.

```

;=====;
;
;           Software License Agreement
;
; The software supplied herewith by Microchip Technology Incorporated
; (the "Company") for its dsPIC controller is intended and supplied to
; you, the Company's customer, for use solely and exclusively on
; Microchip dsPIC products. The software is owned by the Company and/or
; its supplier, and is protected under applicable copyright laws. All
; rights are reserved. Any use in violation of the foregoing
; restrictions may subject the user to criminal sanctions under
; applicable laws, as well as to civil liability for the breach of the
; terms and conditions of this license.
;
; THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES,
; WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO,
; IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
; PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT, IN ANY
; CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL
; DAMAGES, FOR ANY REASON WHATSOEVER.
;
;=====;

; Use Timer 1 to flash LED RD4 when switch S1 is not pressed
;   and flash LED RD5 when switch S1 is pressed
;
;=====

.equ __30F6012, 1
.include "p30f6012.inc"

;-----
;Global Declarations

.global __reset                ;The label for the first line of code
.global __OscillatorFail       ;Declare Oscillator Fail trap routine label
.global __AddressError         ;Declare Address Error trap routine label
.global __StackError           ;Declare Stack Error trap routine label
.global __MathError             ;Declare Math Error trap routine label

;-----
;Configuration bits

config __FOSC, CSW_FSCM_OFF & XT_PLL4
config __FWDTP, WDT_OFF
config __FBORPOR, PBOR_OFF & BORV_27 & PWRT_16 & MCLR_EN
config __FGS, CODE_PROT_OFF

```

Getting Started with dsPIC30F Digital Signal Controllers

```
;-----  
;Program Specific Constants (literals used in code)  
  
    .equ    Fcy, #4000000                ;Instruction cycle rate (Osc x PLL / 4)  
  
;=====  
;Start of code  
  
    .text                                ;Start of Code section  
  
;-----  
;Initialize stack pointer and limit register  
  
__reset:    mov     #__SP_init, W15        ;Initialize the Stack Pointer register  
            mov     #__SPLIM_init, W0     ;Get address at the end of stack space  
            mov     W0, SPLIM              ;Load the Stack Pointer Limit register  
            nop                               ;Add NOP to follow SPLIM initialization  
  
;-----  
;Initialize LED outputs on PORTD bits 4-7  
  
            mov     #0xff0f, W0           ;Initialize LED pin data to off state  
            mov     W0, LATD  
            mov     #0xff0f, W0           ;Set LED pins as outputs  
            mov     W0, TRISD  
            bset    LATD, #4              ;Turn LED RD4 on  
  
;-----  
;Initialize Timer1 for 1/5 second period  
  
            clr     T1CON                  ;Turn off Timer1 by clearing control register  
            clr     TMR1                  ;Start Timer1 at zero  
            mov     #Fcy/256/5, W0        ;Get period register value for 1/5 second  
            mov     W0, PR1               ;Load Timer1 period register  
            mov     #0x8030, W0           ;Get Timer1 settings (1:256 prescaler)  
            mov     W0, T1CON              ;Load Timer1 settings into control register  
  
;-----  
;Loop while waiting for a Timer1 match and toggle LED1 when it happens  
  
MainLoop:   btss    IFS0, #T1IF           ;Check if Timer1 interrupt flag is set  
            bra     MainLoop              ;Loop back until set  
            bclr    IFS0, #T1IF           ;Clear Timer1 interrupt flag  
            btss    PORTC, #13            ;Test switch S1 (low when pressed)  
            bra     SwitchPressed  
  
            btg     LATD, #4              ;Toggle LED RD4  
            bclr    LATD, #5              ;Turn off LED RD5  
            bra     MainLoop              ;Loop back  
  
SwitchPressed:bclr    LATD, #4            ;Turn off LED RD4  
            btg     LATD, #5              ;Toggle LED RD5  
            bra     MainLoop              ;Loop back
```


Code for dsPICDEM Starter Demonstration Board

```
;=====
;Error traps

;-----
;Oscillator Fail Error trap routine

        .text                                ;Start of Code section
__OscillatorFail:
        bclr    LATD, #7                    ;Turn LED RD7 on
        bra     __OscillatorFail            ;Loop forever when oscillator failure occurs

;-----
;Address Error trap routine

__AddressError:
        bclr    LATD, #7                    ;Turn LED RD7 on
        bra     __AddressError              ;Loop forever when address error occurs

;-----
;Stack Error trap routine

__StackError:
        bclr    LATD, #7                    ;Turn LED RD7 on
        bra     __StackError                ;Loop forever when stack error occurs

;-----
;Math (Arithmetic) Error trap routine

__MathError:
        bclr    LATD, #7                    ;Turn LED RD7 on
        bra     __MathError                 ;Loop forever when math error occurs

;=====

        .end                                ;End of code in this file
```

B.2 FLASH LED WITH dsPIC30F6012.c

```
/*
*****
*                               Software License Agreement                               *
*                               *                               *
*   The software supplied herewith by Microchip Technology Incorporated               *
*   (the "Company") for its dsPIC controller is intended and supplied to             *
*   you, the Company's customer, for use solely and exclusively on                   *
*   Microchip dsPIC products. The software is owned by the Company and/or           *
*   its supplier, and is protected under applicable copyright laws. All              *
*   rights are reserved. Any use in violation of the foregoing                      *
*   restrictions may subject the user to criminal sanctions under                   *
*   applicable laws, as well as to civil liability for the breach of the             *
*   terms and conditions of this license.                                           *
*                               *                               *
*   THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES,              *
*   WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO,          *
*   IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR             *
*   PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT, IN ANY                  *
*   CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL               *
*   DAMAGES, FOR ANY REASON WHATSOEVER.                                           *
*****
*
*   Use Timer 1 to flash LED RD4 when switch S1 is not pressed
*   and flash LED RD5 when switch S1 is pressed
*
*****/

#include "p30F6012.h"

//-----
//Configuration bits

_FOSC(CSW_FSCM_OFF & XT_PLL4);
_FWDT(WDT_OFF);
_FBORPOR(PBOR_OFF & BORV_27 & PWRT_16 & MCLR_EN);
_FGS(CODE_PROT_OFF);

//-----
//Program Specific Constants

#define Fcy 4000000 //Instruction cycle rate (Osc x PLL / 4)

//=====
//Main routine
//Set up LEDs and timer, wait for timer periods, and flash one of the two LEDs

int main(void)
{
    LATD = 0xff0f; //Initialize LED pin data to off state
    TRISD = 0xff0f; //Set LED pins as outputs
    LATDbits.LATD4 = 1; //Turn LED RD4 on

    T1CON = 0; //Turn off Timer1 and clear settings
    TMR1 = 0; //Start Timer1 at zero
    PR1 = Fcy/256/5; //Set period register value for 1/5 second
    T1CON = 0x8030; //Turn on Timer1 with 1:256 prescaler

    while(1) //Loop forever
```

```
{
    while(!IFS0bits.T1IF){}                //Wait for timer period

    IFS0bits.T1IF = 0;                      //Clear timer flag for next period
    if(PORTCbits.RC13)                      //Check if S1 is pressed (low when pressed)
    {
        LATDbits.LATD4 ^= 1;                //Toggle LED RD4 when S1 not pressed
        LATDbits.LATD5 = 0;                //Turn off LED RD5
    }
    else
    {
        LATDbits.LATD4 = 0;                //Turn off LED RD4
        LATDbits.LATD5 ^= 1;                //Toggle LED RD5 when S1 is pressed
    }
}

//End of main()

//=====
//Error traps

//-----
//Oscillator Fail Error trap routine

void _ISR _OscillatorFail(void)
{
    LATDbits.LATD7 = 0;                    //Turn LED RD7 on
    while(1);                             //Wait forever
}

//-----
//Address Error trap routine

void _ISR _AddressError(void)
{
    LATDbits.LATD7 = 0;                    //Turn LED RD7 on
    while(1);                             //Wait forever
}

//-----
//Stack Error trap routine

void _ISR _StackError(void)
{
    LATDbits.LATD7 = 0;                    //Turn LED RD7 on
    while(1);                             //Wait forever
}

//-----
//Math (Arithmetic) Error trap routine

void _ISR _MathError(void)
{
    LATDbits.LATD7 = 0;                    //Turn LED RD7 on
    while(1);                             //Wait forever
}
```

NOTES:

Appendix C. Code for dsPICDEM 28-Pin Starter Demonstration Board

C.1 FLASH LED WITH dsPIC30F2010.s

This appendix contains the sample code for the dsPICDEM 28-Pin Starter Demonstration Board. This tutorial example was written for the MPLAB ASM30 Assembler and the C30 C Compiler.

```

;=====
;
;           Software License Agreement
;
; The software supplied herewith by Microchip Technology Incorporated
; (the "Company") for its dsPIC controller is intended and supplied to
; you, the Company's customer, for use solely and exclusively on
; Microchip dsPIC products. The software is owned by the Company and/or
; its supplier, and is protected under applicable copyright laws. All
; rights are reserved. Any use in violation of the foregoing
; restrictions may subject the user to criminal sanctions under
; applicable laws, as well as to civil liability for the breach of the
; terms and conditions of this license.
;
; THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES,
; WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO,
; IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
; PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT, IN ANY
; CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL
; DAMAGES, FOR ANY REASON WHATSOEVER.
;
;=====
;
; Use Timer 1 to flash LED
;
;=====

.equ __30F2010, 1
.include "p30f2010.inc"

;-----
;Global Declarations

.global __reset                ;The label for the first line of code
.global __OscillatorFail      ;Declare Oscillator Fail trap routine label
.global __AddressError        ;Declare Address Error trap routine label
.global __StackError          ;Declare Stack Error trap routine label
.global __MathError           ;Declare Math Error trap routine label

;-----
;Configuration bits

config __FOSC, CSW_FSCM_OFF & XT_PLL4
config __FWDTP, WDT_OFF
config __FBORPOR, PBOR_OFF & BORV_27 & PWRT_16 & MCLR_EN
config __FGS, CODE_PROT_OFF

```

Getting Started with dsPIC30F Digital Signal Controllers

```
;-----  
;Program Specific Constants (literals used in code)  
  
    .equ    Fcy, #7372800                ;Instruction cycle rate (Osc x PLL / 4)  
  
;=====   
;Start of code  
  
    .text                                ;Start of Code section  
  
;-----  
;Initialize stack pointer and limit register  
  
__reset:    mov     #__SP_init, W15        ;Initialize the Stack Pointer register  
            mov     #__SPLIM_init, W0     ;Get address at the end of stack space  
            mov     W0, SPLIM              ;Load the Stack Pointer Limit register  
            nop                               ;Add NOP to follow SPLIM initialization  
  
;-----  
;Initialize LED output on PORTD bit 0  
  
            mov     #0xffff, W0           ;Initialize LED pin data to off state  
            mov     W0, LATD  
            mov     #0xffff, W0           ;Set LED pin as output  
            mov     W0, TRISD  
            bset    LATD, #0               ;Turn LED on  
  
;-----  
;Initialize Timer1 for 1/5 second period  
  
            clr     T1CON                  ;Turn off Timer1 by clearing control register  
            clr     TMR1                   ;Start Timer1 at zero  
            mov     #Fcy/256/5,W0         ;Get period register value for 1/5 second  
            mov     W0, PR1                ;Load Timer1 period register  
            mov     #0x8030,W0            ;Get Timer1 settings (1:256 prescaler)  
            mov     W0, T1CON              ;Load Timer1 settings into control register  
  
;-----  
;Loop while waiting for a Timer1 match and toggle LED1 when it happens  
  
MainLoop:   btss    IFS0, #T1IF           ;Check if Timer1 interrupt flag is set  
            bra     MainLoop              ;Loop back until set  
            bclr    IFS0, #T1IF           ;Clear Timer1 interrupt flag  
            btg     LATD, #0              ;Toggle LED  
            bra     MainLoop              ;Loop back  
  
;=====   
;Error traps  
  
;-----  
;Oscillator Fail Error trap routine  
  
    .text                                ;Start of Code section  
__OscillatorFail:  
    bclr    LATD, #0                      ;Turn LED on  
    bra     __OscillatorFail              ;Loop forever when oscillator failure occurs  
  
;-----  
;Address Error trap routine
```

Code for dsPICDEM 28-Pin Starter Demonstration Board

```
__AddressError:
    bclr    LATD, #0           ;Turn LED on
    bra     __AddressError     ;Loop forever when address error occurs

;-----
;Stack Error trap routine

__StackError:
    bclr    LATD, #0           ;Turn LED on
    bra     __StackError       ;Loop forever when stack error occurs

;-----
;Math (Arithmetic) Error trap routine

__MathError:
    bclr    LATD, #0           ;Turn LED on
    bra     __MathError        ;Loop forever when math error occurs

;=====

    .end                      ;End of code in this file
```

C.2 FLASH LED WITH dsPIC30F2010.c

```
/******
 *                               Software License Agreement                               *
 *
 * The software supplied herewith by Microchip Technology Incorporated
 * (the "Company") for its dsPIC controller is intended and supplied to
 * you, the Company's customer, for use solely and exclusively on
 * Microchip dsPIC products. The software is owned by the Company and/or
 * its supplier, and is protected under applicable copyright laws. All
 * rights are reserved. Any use in violation of the foregoing
 * restrictions may subject the user to criminal sanctions under
 * applicable laws, as well as to civil liability for the breach of the
 * terms and conditions of this license.
 *
 * THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES,
 * WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO,
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT, IN ANY
 * CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL
 * DAMAGES, FOR ANY REASON WHATSOEVER.
 *
 *****/

 * Use Timer 1 to flash LED
 *
 *****/

#include "p30F2010.h"

//-----
//Configuration bits

_FOSC(CSW_FSCM_OFF & XT_PLL4);
_FWDT(WDT_OFF);
_FBORPOR(PBOR_OFF & BORV_27 & PWRT_16 & MCLR_EN);
_FGS(CODE_PROT_OFF);

//-----
//Program Specific Constants

#define Fcy 7372800 //Instruction cycle rate (Osc x PLL / 4)

//=====
//Main routine
//Set up LEDs and timer, wait for timer periods, and flash one of the two LEDs

int main(void)
{
    LATD = 0xfffe; //Initialize LED pin data to off state
    TRISD = 0xfffe; //Set LED pin as output
    LATDbits.LATD0 = 1; //Turn LED on

    T1CON = 0; //Turn off Timer1 and clear settings
    TMR1 = 0; //Start Timer1 at zero
    PR1 = Fcy/256/5; //Set period register value for 1/5 second
    T1CON = 0x8030; //Turn on Timer1 with 1:256 prescaler
```



```
while(1)                                //Loop forever
{
    while(!IFS0bits.T1IF){}             //Wait for timer period
    IFS0bits.T1IF = 0;                   //Clear timer flag for next period
    LATDbits.LATD0 ^= 1;                 //Toggle LED
}
//End of main()

//=====
//Error traps

//-----
//Oscillator Fail Error trap routine

void _ISR _OscillatorFail(void)
{
    LATDbits.LATD0 = 1;                  //Turn LED on
    while(1);                           //Wait forever
}

//-----
//Address Error trap routine

void _ISR _AddressError(void)
{
    LATDbits.LATD0 = 1;                  //Turn LED on
    while(1);                           //Wait forever
}

//-----
//Stack Error trap routine

void _ISR _StackError(void)
{
    LATDbits.LATD0 = 1;                  //Turn LED on
    while(1);                           //Wait forever
}

//-----
//Math (Arithmetic) Error trap routine

void _ISR _MathError(void)
{
    LATDbits.LATD0 = 1;                  //Turn LED on
    while(1);                           //Wait forever
}
```

NOTES:

Appendix D. Code for dsPICDEM 2 Development Board

D.1 FLASH LED WITH dsPIC30F4011.s

This appendix contains the sample code for the dsPICDEM 2 Development Board. This tutorial example was written for the MPLAB ASM30 Assembler and the C30 C Compiler.

```

;=====;
;
;           Software License Agreement
;
;
; The software supplied herewith by Microchip Technology Incorporated
; (the "Company") for its dsPIC controller is intended and supplied to
; you, the Company's customer, for use solely and exclusively on
; Microchip dsPIC products. The software is owned by the Company and/or
; its supplier, and is protected under applicable copyright laws. All
; rights are reserved. Any use in violation of the foregoing
; restrictions may subject the user to criminal sanctions under
; applicable laws, as well as to civil liability for the breach of the
; terms and conditions of this license.
;
;
; THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES,
; WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO,
; IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
; PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT, IN ANY
; CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL
; DAMAGES, FOR ANY REASON WHATSOEVER.
;
;=====;
;
; Use Timer 1 to flash LED D3 when switch S5 is not pressed
; and flash LED D4 when switch S5 is pressed
; Jumpers H6-M ALL, H12-M D3, and H12-M D4 must be in place
;
;=====;

.equ __30F4011, 1
.include "p30f4011.inc"

;-----;
;Global Declarations

.global __reset                ;The label for the first line of code
.global __OscillatorFail       ;Declare Oscillator Fail trap routine label
.global __AddressError         ;Declare Address Error trap routine label
.global __StackError           ;Declare Stack Error trap routine label
.global __MathError            ;Declare Math Error trap routine label

```

Getting Started with dsPIC30F Digital Signal Controllers

```
;-----  
;Configuration bits  
  
    config __FOSC, CSW_FSCM_OFF & XT_PLL4  
    config __FWDTP, WDT_OFF  
    config __FBORPOR, PBOR_OFF & BORV_27 & PWRT_16 & MCLR_EN  
    config __FGS, CODE_PROT_OFF  
  
;-----  
;Program Specific Constants (literals used in code)  
  
    .equ    Fcy, #7372800                ;Instruction cycle rate (Osc x PLL / 4)  
  
;=====   
;Start of code  
  
    .text                                ;Start of Code section  
  
;-----  
;Initialize stack pointer and limit register  
  
__reset:    mov     #__SP_init, W15      ;Initialize the Stack Pointer register  
            mov     #__SPLIM_init, W0    ;Get address at the end of stack space  
            mov     W0, SPLIM            ;Load the Stack Pointer Limit register  
            nop                               ;Add NOP to follow SPLIM initialization  
  
;-----  
;Initialize LED outputs on PORTD bits 0-3  
  
            mov     #0x0000, W0          ;Initialize LED pin data to off state  
            mov     W0, LATB  
            mov     #0x0003, W0          ;Set LED pins as digital, not analog  
            mov     W0, ADPCFG  
            mov     #0xffffc, W0         ;Set LED pins as outputs  
            mov     W0, TRISB  
            bset    LATB, #0             ;Turn LED D3 on  
  
;-----  
;Initialize Timer1 for 1/5 second period  
  
            clr     T1CON                ;Turn off Timer1 by clearing control register  
            clr     TMR1                 ;Start Timer1 at zero  
            mov     #Fcy/256/5,W0        ;Get period register value for 1/5 second  
            mov     W0, PR1              ;Load Timer1 period register  
            mov     #0x8030,W0           ;Get Timer1 settings (1:256 prescaler)  
            mov     W0, T1CON            ;Load Timer1 settings into control register  
  
;-----  
;Loop while waiting for a Timer1 match and toggle LED1 or LED2 when it happens  
  
MainLoop:   btss    IFS0, #T1IF          ;Check if Timer1 interrupt flag is set  
            bra     MainLoop             ;Loop back until set  
            bclr    IFS0, #T1IF          ;Clear Timer1 interrupt flag  
            btss    PORTE, #8            ;Test switch S5 (low when pressed)  
            bra     SwitchPressed  
  
            btg     LATB, #0             ;Toggle LED D3 when S5 is not pressed  
            bclr    LATB, #1             ;Turn off LED D4  
            bra     MainLoop             ;Loop back
```

Code for dsPICDEM 2 Development Board

```
SwitchPressed:bclr    LATB, #0           ;Turn off LED D3
                   btg     LATB, #1       ;Toggle LED D4 when S5 is pressed
                   bra     MainLoop       ;Loop back

;=====
;Error traps

;-----
;Oscillator Fail Error trap routine

        .text                               ;Start of Code section
__OscillatorFail:
        bset    LATB, #1           ;Turn LED D4 on
        bra     __OscillatorFail    ;Loop forever when oscillator failure occurs

;-----
;Address Error trap routine

__AddressError:
        bset    LATB, #1           ;Turn LED D4 on
        bra     __AddressError      ;Loop forever when address error occurs

;-----
;Stack Error trap routine

__StackError:
        bset    LATB, #1           ;Turn LED D4 on
        bra     __StackError        ;Loop forever when stack error occurs

;-----
;Math (Arithmetic) Error trap routine

__MathError:
        bset    LATB, #1           ;Turn LED D4 on
        bra     __MathError         ;Loop forever when math error occurs

;=====

        .end                               ;End of code in this file
```

D.2 FLASH LED WITH dsPIC30F4011.c

```
/******
 *                               Software License Agreement                               *
 *                                                                                       *
 * The software supplied herewith by Microchip Technology Incorporated                   *
 * (the "Company") for its dsPIC controller is intended and supplied to                 *
 * you, the Company's customer, for use solely and exclusively on                     *
 * Microchip dsPIC products. The software is owned by the Company and/or             *
 * its supplier, and is protected under applicable copyright laws. All                 *
 * rights are reserved. Any use in violation of the foregoing                         *
 * restrictions may subject the user to criminal sanctions under                      *
 * applicable laws, as well as to civil liability for the breach of the                 *
 * terms and conditions of this license.                                              *
 *                                                                                       *
 * THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES,                 *
 * WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO,             *
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR                 *
 * PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT, IN ANY                     *
 * CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL                   *
 * DAMAGES, FOR ANY REASON WHATSOEVER.                                                *
 *                                                                                       *
 *****/

 *
 * Use Timer 1 to flash LED D3 when switch S5 is not pressed
 * and flash LED D4 when switch S5 is pressed
 * Jumpers H6-M ALL, H12-M D3, and H12-M D4 must be in place
 *
 *****/

#include "p30F4011.h"

//-----
//Configuration bits

_FOSC(CSW_FSCM_OFF & XT_PLL4);
_FWDT(WDT_OFF);
_FBORPOR(PBOR_OFF & BORV_27 & PWRT_16 & MCLR_EN);
_FGS(CODE_PROT_OFF);

//-----
//Program Specific Constants

#define Fcy 7372800 //Instruction cycle rate (Osc x PLL / 4)

//=====
//Main routine
//Set up LEDs and timer, wait for timer periods, and flash one of the two LEDs

int main(void)
{
    static int MyVar1, MyVar2, MyVar3, MyVar4, MyVar5;

    MyVar1 = 1000;
    MyVar2 = 20;
    MyVar3 = MyVar1 + MyVar2;
    MyVar4 = MyVar1 * MyVar2;
    MyVar5 = MyVar4 / MyVar3;
```

```
LATB = 0x0000;           //Initialize LED pin data to off state
TRISB = 0xffffc;         //Set LED pins as outputs
LATBbits.LATB0 = 1;      //Turn LED D3 on

T1CON = 0;               //Turn off Timer1 and clear settings
TMR1 = 0;                //Start Timer1 at zero
PR1 = Fcy/256/5;         //Set period register value for 1/5 second
T1CON = 0x8030;          //Turn on Timer1 with 1:256 prescaler

while(1)                 //Loop forever
{
    while(!IFS0bits.T1IF){} //Wait for timer period

    IFS0bits.T1IF = 0;     //Clear timer flag for next period
    if(PORTEbits.RE8)     //Check if S5 is pressed
    {
        LATBbits.LATB0 ^= 1; //Toggle LED D3 when S5 not pressed
        LATBbits.LATB1 = 0;  //Turn off LED D4
    }
    else
    {
        LATBbits.LATB0 = 0;  //Turn off LED D3
        LATBbits.LATB1 ^= 1; //Toggle LED D4 when S5 is pressed
    }
}

//End of main()

//=====
//Error traps

//-----
//Oscillator Fail Error trap routine

void _ISR _OscillatorFail(void)
{
    LATBbits.LATB1 = 1;     //Turn LED D4 on
    while(1);              //Wait forever
}

//-----
//Address Error trap routine

void _ISR _AddressError(void)
{
    LATBbits.LATB1 = 1;     //Turn LED D4 on
    while(1);              //Wait forever
}

//-----
//Stack Error trap routine

void _ISR _StackError(void)
{
    LATBbits.LATB1 = 1;     //Turn LED D4 on
    while(1);              //Wait forever
}
```

Getting Started with dsPIC30F Digital Signal Controllers

```
//-----  
//Math (Arithmetic) Error trap routine  
  
void _ISR _MathError(void)  
{  
    LATBbits.LATB1 = 1;           //Turn LED D4 on  
    while(1);                     //Wait forever  
}
```


Index

A

Application Notes	17
Applications.....	14
Automotive.....	14
Internet Connectivity	14
Motor Control	14
Power Conversion, Monitoring.....	14
Sensor Control.....	14
Speech and Audio	14
Architecture.....	7
Harvard	7

B

Building the Project	31
----------------------------	----

C

C Compilers	17
Code	
Animate.....	47, 61
Building	30
Resetting.....	35, 46, 61
Running	36, 48, 62
Step Into	47, 61
Step Over.....	47, 61
Stepping Through	47
Configuration Bits.....	31
CTR-21 PSTN.....	24
Customer Change Notification Service	5
Customer Support.....	6

D

Data Addressing Modes.....	9
Bit-Reversed	10
Modulo	10
Data Memory	8
Data Memory Map	9
Debug Toolbar	36
Debugging.....	31
Demo Code	
dsPICDEM 1.1 General Purpose	
Development Board.....	99
Flash LED with dsPIC30F6014.c.....	102
Flash LED with dsPIC30F6014.s.....	99
dsPICDEM 2 Development Board	117
Flash LED with dsPIC30F4011.c.....	120
Flash LED with dsPIC30F4011.s.....	117
dsPICDEM 28-Pin Starter	
Demonstration Board	111
Flash LED with dsPIC30F2010.c.....	114
Flash LED with dsPIC30F2010.s.....	111

dsPICDEM Starter	
Demonstration Board.....	105
Flash LED with dsPIC30F6012.c.....	108
Flash LED with dsPIC30F6012.s.....	105
Development Boards.....	21–24
dsPICDEM 1.1 General Purpose	
Development Board.....	22
dsPICDEM 2 Development Board	22
dsPICDEM 28-Pin Starter	
Demonstration Board.....	21
dsPICDEM MC1 Motor Control	
Development System	23
dsPICDEM MC1H 3-Phase	
High-Voltage Power Module.....	24
dsPICDEM MC1L 3-Phase	
Low-Voltage Power Module.....	23
dsPICDEM Starter	
Demonstration Board.....	21
dsPICDEM.net 1 Connectivity	
Development Board.....	24
dsPICDEM.net 2 Connectivity	
Development Board.....	24
Device Variants	13
General Purpose Family	13
Motor Control and Power	
Conversion Family	13
Sensor Family	13
Directives	
.align	75
.bss	74
.data	74
.end	73
.equ	72
.global	73
.hword	74
.include.....	72
.palign	75
.section.....	73
.space	74
.text	73
Documentation	
Conventions	3
Organization.....	2
Recommended Reading	4
DSP Engine.....	11
dsPICDEM 1.1	22
dsPICDEM 2	22
dsPICDEM 28-Pin Starter Demo Board	21
dsPICDEM MC1.....	23

Getting Started with dsPIC30F Digital Signal Controllers

dsPICDEM MC1H	24	MPLAB C30	
dsPICDEM MC1L	23	Building the Project	87
dsPICDEM Starter Demo Board.....	21	Example Code.....	88
dsPICDEM.net 1	24	Beginning of Main Routine.....	89
dsPICDEM.net 2	24	Comments, Header File	
E		Reference	88
Ethernet Interface.....	24	Configuration Bits.....	88
Example Code for MPLAB ASM30.....	75–79	Error Trap Routines	90
Comments.....	75	Main Routine.....	90
Configuration Bits.....	76	Processor Frequency Definition.....	89
Constants.....	76	Timer Initialization	89
Global Declarations.....	76	While Loop	89
Initialization	77	Language Features	87
F		__attribute__ keyword.....	87
FCC/JATE PSTN.....	24	ANSI C Standard	87
Function Keys	36	Standard Header File.....	87
G		Overview	81
GNU Toolsuite.....	16	Projects	81
H		Setting Build Options.....	86
Hotkeys	36	MPLAB C30 C Compiler.....	81–90
I		MPLAB ICD 2	20, 43–53
Include Files	30	Advanced Breakpoint Feature.....	52
Instruction Set	10	Breakpoints	49
DSP Instructions	10, 11	Debug Toolbar	48
MAC	11	Hotkeys	48
MCU Instructions	10	Overview	43
Instructions and Directives		Setting Up	44
General Format.....	71	Watch Window	50
Syntax Rules.....	72	MPLAB ICE 4000	18, 55–70
Internet Address	5	Breakpoints	63
Interrupts	12	Complex Triggers.....	68
Alternate Interrupt		Debug Toolbar	62
Vector Table (AIVT).....	12	Hotkeys	62
Interrupt Vector Table (IVT)	12	Overview	55
L		Selecting as Debugger.....	58
Linker Files	30	Settings	59
.cof	30	Setup.....	56
.hex	30	Special Emulator Devices	57
Linker Script Files.....	92	Stopwatch Feature	66
Base Memory Address.....	93	Trace Buffer	67
Input/Output Section Map	94	Trace Window	67
Interrupt Vector Tables	96	USB Driver	56
Memory Region Information.....	93	Watch Window	64
Output File Format and Entry Point.....	92	MPLAB IDE	15, 25–31
Range Check Expressions.....	95	Debugging Tools	17
SFR Addresses.....	97	Comparison.....	19
M		MPLAB ICE 4000.....	18
Microchip Internet Web Site	5	MPLAB SIM30 Simulator	17
MPLAB ASM30	16	Editor.....	16, 29
Instructions and Directives.....	71	Language Tools	16
Overview	71	C Compilers	17
MPLAB ASM30 Assembler	71–79	MPLAB ASM30 Assembler.....	16
		MPLAB LINK30 Linker	16
		Overview	25
		Programming Tools.....	20
		MPLAB ICD 2 In-Circuit Debugger	20
		MPLAB PM3 Universal	
		Device Programmer.....	20

Project Wizard	26	R	
Projects and Workspaces	16, 26	Resets	
Template, Include and		Brown-out Reset	35
Linker Script Files	17	MCLR	35
MPLAB LINK30	16	Processor (Power-on) Reset	35
MPLAB LINK30 Linker	91–97	Watchdog Timer Reset	35
Overview	91	Resetting Code	61
MPLAB PM3	20	Running Code	62
MPLAB SIM		S	
Breakpoints	37	SFR, See Special Function Registers.	
Opening the Project	34	Special Function Registers	8
Overview	33	Stopwatch	
Selecting the Simulator	34	MPLAB ICE 4000	66
Simulator Settings	39	System and Power Management	12
Stopwatch	40	T	
Trace Buffer	41	TCP/IP	24
Trace Window	41	U	
Watch Window	38	USB Driver	
MPLAB SIM Simulator	33–42	Installing	43
MPLAB SIM30	17	V	
N		V.22bis/V.22 ITU	24
Near RAM	8	W	
P		Working Register Array	9
Peripherals	13	WWW Address	5
Program Counter	8		
Program Memory	8		
Space	8		
Program Memory Map	9		
Program Space Visibility	8		
Program Space Visibility (PSV)	10		
Program Space Visibility Page Register	10		
Programming the dsPIC Device	46		
Project Wizard	82		
Creating a Project	82		
PSTN Interface	24		
PSV, See Program Space Visibility.			
PSVPAG Register, See Program Space			
Visibility Page Register.			



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

Alpharetta, GA
Tel: 770-640-0034
Fax: 770-640-0307

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

San Jose

Mountain View, CA
Tel: 650-215-1444
Fax: 650-961-0286

Toronto

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

ASIA/PACIFIC

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: 86-28-8676-6200
Fax: 86-28-8676-6599

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 - 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 - Qingdao

Tel: 86-532-502-7355
Fax: 86-532-502-7205

ASIA/PACIFIC

India - Bangalore

Tel: 91-80-2229-0061
Fax: 91-80-2229-0062

India - New Delhi

Tel: 91-11-5160-8631
Fax: 91-11-5160-8632

Japan - Kanagawa

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

Korea - Seoul

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

Malaysia - Penang

Tel: 011-604-646-8870
Fax: 011-604-646-5086

Philippines - Manila

Tel: 011-632-634-9065
Fax: 011-632-634-9069

Singapore

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

Taiwan - Kaohsiung

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

Taiwan - Taipei

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

Taiwan - Hsinchu

Tel: 886-3-572-9526
Fax: 886-3-572-6459

EUROPE

Austria - Weis

Tel: 43-7242-2244-399
Fax: 43-7242-2244-393

Denmark - Ballerup

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

France - Massy

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

Germany - Ismaning

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

England - Berkshire

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