16-BIT LANGUAGE TOOLS GETTING STARTED

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

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

Table of Contents

Preface 1

Chapter 1. Installation and Overview

1.1 Introduction 7
1.2 Installing/Uninstalling MPLAB IDE 7
1.3 Installing the 16-Bit Assembler, Linker and Utilities 7
1.4 Installing/Uninstalling the 16-Bit C Compiler 7
1.5 Tutorial Overview 8

Chapter 2. Tutorial 1 – Creating A Project

2.1 Introduction 9
2.2 Creating a File 9
2.3 Using the Project Wizard 10
2.4 Using the Project Window 12
2.5 Setting Up Build Options 13
2.6 Building the Project 18
2.7 Troubleshooting Build Errors 18
2.8 Debugging with the MPLAB SIM Simulator 20
2.9 Generating a Map File 23
2.10 Debugging at Assembly Code Level 24
2.11 Exploring Further 26

Chapter 3. Tutorial 2 – Real-Time Interrupt

3.1 Introduction 27
3.2 Using Template Files 27
3.3 Using the Template in a New Project 32
3.4 Debugging with the MPLAB SIM Simulator 39
3.5 Exploring Further 43

Chapter 4. Tutorial 3 – Mixed C and Assembly Files

4.1 Introduction 45
4.2 Getting Project Source Files 45
4.3 Creating and Building the Project 48
4.4 Examining the Program 49
4.5 Exploring Further 53
4.6 Where to Go from Here 53

Index 55

Worldwide Sales and Service 56

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 "DSXXXXXA", where "XXXXX" 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

This chapter contains general information that will be useful to know before using 16-Bit Language Tools. Items discussed include:

- Document Layout
- Conventions Used in this Guide
- Recommended Reading
- The Microchip Web Site
- Development Systems Customer Change Notification Service
- Customer Support

DOCUMENT LAYOUT

This document describes how to use 16-Bit Language Tools as development tools to emulate and debug firmware on a target board. The manual layout is as follows:

- **Chapter 1: Installation and Overview** – How to install the 16-Bit Language Tools on your PC and how they work.
- **Chapter 2: Tutorial 1** – **Creating a Project** – How to set up a project using 16-Bit Language Tools.
- **Chapter 3: Tutorial 2 – Real-Time Interrupt** – How to create a 16-bit application using a real-time interrupt.
- **Chapter 4: Tutorial 3 – Mixed C and Assembly Files** – How to create a 16-bit application using a combination of C and assembly code files.

CONVENTIONS USED IN THIS GUIDE

The following conventions may appear in this documentation:

DOCUMENTATION CONVENTIONS

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

This documentation describes how to use 16-Bit Language Tools. Other useful documents are listed below. The following Microchip documents are available and recommended as supplemental reference resources.

Readme Files

For the latest information on Microchip tools, read the associated Readme files (HTML files) included with the software.

MPLAB® Assembler, Linker and Utilities for PIC24 MCUs and dsPIC® DSCs User's Guide (DS51317)

A guide to using the 16-bit assembler, object linker, object archiver/librarian and various utilities.

MPLAB C Compiler for PIC24 MCUs and dsPIC® DSCs User's Guide (DS51284)

A guide to using the 16-bit C compiler. The 16-bit linker is used with this tool.

16-Bit Language Tools Libraries (DS51456)

A descriptive listing of libraries available for Microchip 16-bit devices. This includes standard (including math) libraries and C compiler built-in functions. DSP and 16-bit peripheral libraries are described in Readme files provided with each peripheral library type.

Device-Specific Documentation

The Microchip website contains many documents that describe 16-bit device functions and features. Among these are:

- Individual and family data sheets
- Family reference manuals
- Programmer's reference manuals

THE MICROCHIP WEB SITE

Microchip provides online support via our web 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 (FAQs), 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, assemblers, linkers and other language tools. These include all MPLAB C compilers; all MPLAB assemblers (including MPASM™ assembler); all MPLAB linkers (including MPLINK™ object linker); and all MPLAB librarians (including MPLIB™ object librarian).
- **Emulators** – The latest information on Microchip in-circuit emulators.These include the MPLAB REAL ICE™, MPLAB ICE 2000 and MPLAB ICE 4000 in-circuit emulators
- **In-Circuit Debuggers** – The latest information on Microchip in-circuit debuggers. These include the MPLAB ICD 2 in-circuit debugger and PICkit™ 2 debug express.
- **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 IDE Project Manager, MPLAB Editor and MPLAB SIM simulator, as well as 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 and PICkit 1and 2 development programmers.

CUSTOMER SUPPORT

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Field Application Engineer (FAE)
- Technical Support

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

NOTES:

Chapter 1. Installation and Overview

1.1 INTRODUCTION

This document provides a step-by-step guide to using 16-bit (PIC24 MCU and dsPIC DSC devices) language tools with the MPLAB® Integrated Development Environment (IDE) v8.10 or later. The MPLAB IDE project manager and the MPLAB SIM simulator are both components of MPLAB IDE and, along with the built-in debugger structure, will be used extensively in this guide.

Items discussed in this chapter are:

- Installing/Uninstalling MPLAB IDE
- Installing the 16-Bit Assembler, Linker and Utilities
- Installing/Uninstalling the 16-Bit C Compiler
- Tutorial Overview

1.2 INSTALLING/UNINSTALLING MPLAB IDE

MPLAB IDE is provided on CD-ROM or is available from www.microchip.com at no charge. Follow the instructions on the CD-ROM or website to install MPLAB IDE. To uninstall, view the instructions in the file `Readme for MPLAB IDE.txt`.

1.3 INSTALLING THE 16-BIT ASSEMBLER, LINKER AND UTILITIES

The MPLAB Assembler and Object Linker for PIC24 MCUs and dsPIC DSCs (formerly MPLAB ASM30 and MPLAB LINK30) are provided free with MPLAB IDE. They are also included in the installation for the MPLAB C Compiler for PIC24 MCUs and dsPIC DSCs (formerly MPLAB C30). To ensure compatibility between all 16-bit tools, the versions of these tools provided with the 16-bit compiler should be used.

1.4 INSTALLING/UNINSTALLING THE 16-BIT C COMPILER

To install the MPLAB C Compiler for PIC24 MCUs and dsPIC DSCs (formerly MPLAB C30) and related 16-bit language tools, do the following:

- Make a backup when installing the 16-bit compiler as an update to a previous version, since existing files may be overwritten.
- Insert the CD-ROM into the PC and execute the installation file. A series of dialogs will step through the installation process. The installation may take a few minutes as it searches for MPLAB IDE and other related files on the PC.
- To follow the examples in this guide, make sure that the check box for EXAMPLES is checked.

To uninstall the 16-bit compiler, open the folder where the compiler is installed and double click on `UNWISE.exe`.

Note: When uninstalling an upgraded version of the 16-bit compiler, the entire installation will be removed. If files have been added to directories after the previous installation, these will not be removed.

1.5 TUTORIAL OVERVIEW

The following tutorials are intended to help an engineer familiar with the C programming language and embedded systems concepts get started using the MPLAB C Compiler for PIC24 MCUs and dsPIC DSCs with MPLAB IDE. This document shows how to create and build projects, how to write code using features of 16-bit devices and how to verify and debug code written for the 16-bit compiler.

These tutorials assume that the 16-bit compiler and MPLAB IDE are installed (see the previous sections.) Please refer to the Microchip web site for 16-bit device data sheets and family reference manuals containing information on processor-specific items such as the Special Function Registers, instruction set and interrupt logic.

Tutorials presented in these chapters for using the 16-bit compiler include:

- **Tutorial 1** which demonstrates how to:
  - set up and build a project
  - run, step and set breakpoints in the example code
  - debug the code
- **Tutorial 2** which demonstrates how to:
  - use templates to create a source file
  - use a real-time interrupt in C
- **Tutorial 3** which demonstrates how to:
  - use the 16-bit compiler with an assembly language DSP routine
  - pass parameters to and from an assembly language module

Chapter 2. Tutorial 1 – Creating A Project

2.1 INTRODUCTION

The simple source code in this tutorial is designed for an MPLAB IDE v8.10 project. It will use the MPLAB SIM simulator for the dsPIC30F6014 device. The tutorial assumes the default MPLAB C Compiler for PIC24 MCUs and dsPIC DSCs (formerly MPLAB C30) installation directory, i.e.,
`C:\Program Files\Microchip\MPLAB C30`.

This tutorial consists of:

- Creating a File
- Using the Project Wizard
- Using the Project Window
- Setting Up Build Options
- Building the Project
- Troubleshooting Build Errors
- Debugging with the MPLAB SIM Simulator
- Generating a Map File
- Debugging at Assembly Code Level
- Exploring Further

2.2 CREATING A FILE

Start MPLAB IDE and select *File>New* to bring up a new empty source file. The source code that should be typed in (or copied and pasted if viewing this electronically) to this New Source File window is shown in Example 2-1.

EXAMPLE 2-1: MyFile.c

```
#include "p30f6014.h"

int counter;            // for TRISB and PORTB declarations
int main (void)
{
   counter = 1;
   TRISB = 0;           // configure PORTB for output
   while(1)             // do forever
   {
      PORTB = counter; // send value of 'counter' out PORTB
      counter++;
   }
   return 0;
}
```

`TRISB` and `PORTB` are Special Function Registers (SFRs) on the dsPIC30F6014 device. `PORTB` is a set of general purpose input/output pins. `TRISB` bits configure the `PORTB` pins as inputs (`1`) or outputs (`0`).

Use *File>Save As* to save this file with the file name `MyFile.c` in the `\examples` folder under the installation folder.

2.3 USING THE PROJECT WIZARD

Select *Project>Project Wizard* to create a new project. The Welcome page will appear. Click **Next>** to continue.

1. At “Step One: Select a Device”, use the pull-down menu to select the dsPIC30F6014 device. Click **Next>** to continue.
2. At “Step Two: Select a language toolsuite”, choose “Microchip C30 Toolsuite” as the “Active Toolsuite”. Then click on each language tool in the toolsuite (under “Toolsuite Contents”) and check or set up its associated executable (Figure 2-1).

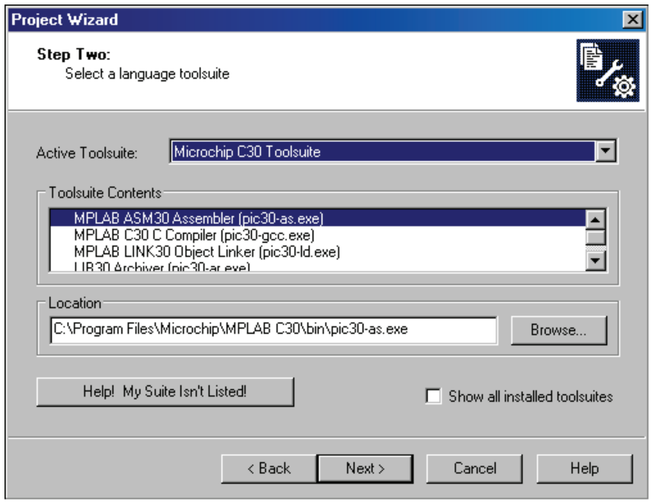
   MPLAB ASM30 Assembler should point to the assembler executable, `pic30-as.exe` under “Location”. If it does not, enter or browse to the executable location, which is by default:
   `C:\Program Files\Microchip\MPLAB C30\bin\pic30-as.exe`

   MPLAB C30 C Compiler should point to the compiler executable, `pic30-gcc.exe` under “Location”. If it does not, enter or browse to the executable location, which is by default:
   `C:\Program Files\Microchip\MPLAB C30\bin\pic30-gcc.exe`

   MPLAB LINK30 Object Linker should point to the linker executable, `pic30-ld.exe` under “Location”. If it does not, enter or browse to the executable location, which is by default:
   `C:\Program Files\Microchip\MPLAB C30\bin\pic30-ld.exe`

   Click **Next>** to continue.

FIGURE 2-1: PROJECT WIZARD – SELECT LANGUAGE TOOLSUITE


Project Wizard

Step Two:
Select a language toolsuite

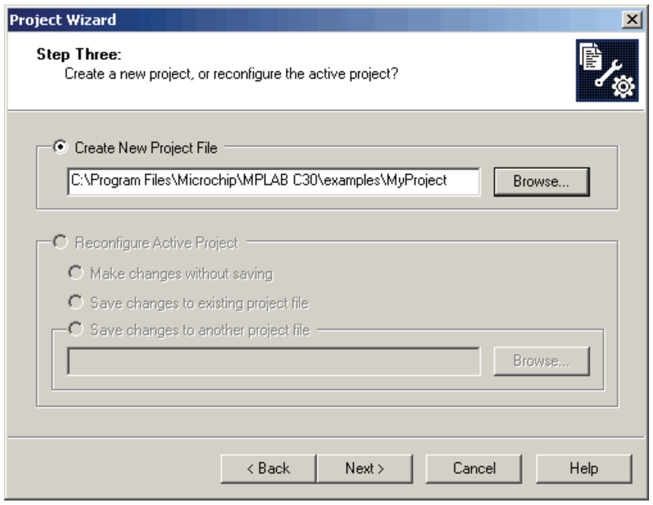
Active Toolsuite: Microchip C30 Toolsuite

Toolsuite Contents
MPLAB ASM30 Assembler (pic30-as.exe)
MPLAB C30 C Compiler (pic30-gcc.exe)
MPLAB LINK30 Object Linker (pic30-ld.exe)
LIB30 Archiver (pic30-ar.exe)

Location
C:\Program Files\Microchip\MPLAB C30\bin\pic30-as.exe

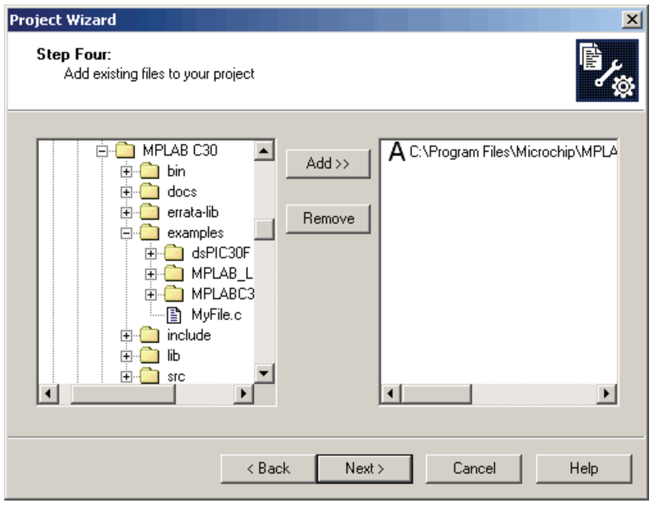
3. At “Step Three: Create new project...”, use **Browse** to go the `\examples` folder in the installation directory for the 16-bit compiler and enter the name of the project as `MyProject`. Click **Save** and then **Next>** to continue.

FIGURE 2-2: PROJECT WIZARD – PROJECT NAME AND DIRECTORY


Project Wizard
Step Three:
Create a new project, or reconfigure the active project?
Create New Project File
C:\Program Files\Microchip\MPLAB C30\examples\MyProject
Browse...
Reconfigure Active Project
Make changes without saving
Save changes to existing project file
Save changes to another project file
Browse...
< Back
Next >
Cancel
Help

4. At “Step Four: Add existing files to your project”, a C file will be added to the project. Select the source file created earlier, `MyFile.c`, in the `\examples` folder. Press **ADD>>** to add it to the list of files to be used for this project (on the right). For information on the letter to the left of the added file, see MPLAB IDE Help, “Projects and Workspaces”. Leave the “A” for this project.

FIGURE 2-3: PROJECT WIZARD – ADD C SOURCE FILE



A linker script file no longer needs be added to a project using the 16-bit C compiler. The linker will automatically find and use the correct linker script for the selected device.

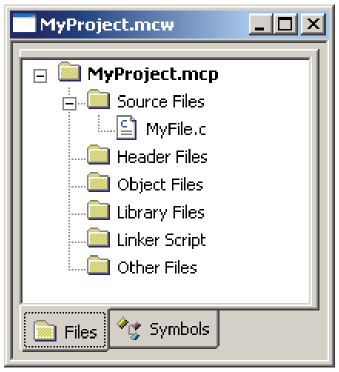
Select **Next>** to continue.

5. At the Summary screen, review the “Project Parameters” to verify that the device, toolsuite and project file location are correct. If you wish to change anything, use **Back** to return to a previous wizard dialog. Click **Finish** to create the new project and workspace.

2.4 USING THE PROJECT WINDOW

Locate the Project window on the MPLAB IDE workspace. The file name of the workspace should appear in the top title bar of the Project window, `MyProject.mcw`, with the file name as the top "node" in the project, `MyProject.mcp`. Project files will be listed in tree format on the **Files** tab.

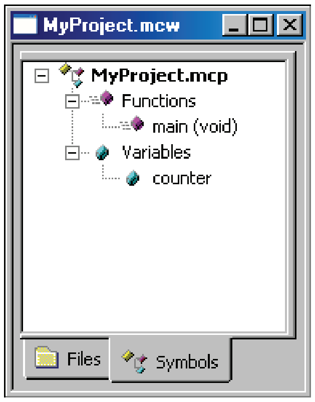
FIGURE 2-4: PROJECT WINDOW – FILES TAB



Note:	If an error was made, highlight a file name and press the Delete key or use the right mouse menu to delete a file. Place the cursor over "Source Files" or other folder and use the right mouse menu to add the proper files to the project.

To view the functions and variables in this short example, click the **Symbols** tab.

FIGURE 2-5: PROJECT WINDOW – SYMBOLS TAB



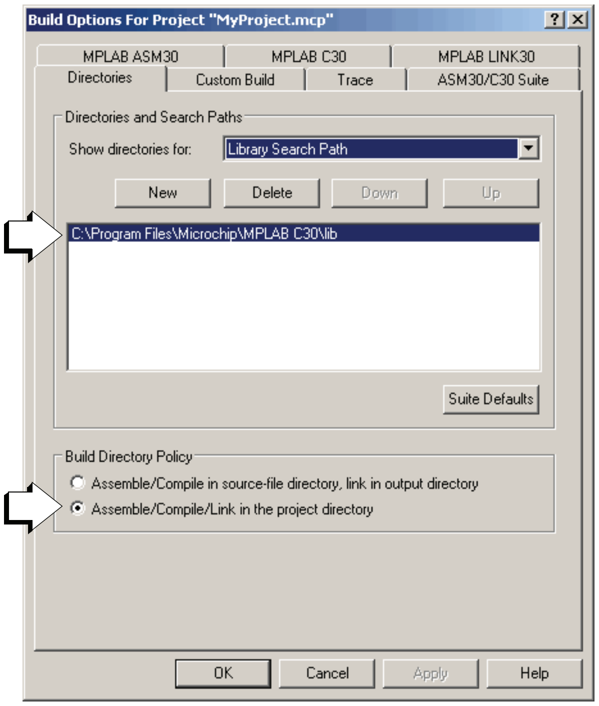
If you do not see any symbols on this tab, right-click in the window to pop up a menu and make sure that "Enable Tag Locators" is checked.

2.5 SETTING UP BUILD OPTIONS

The 16-bit tools are almost ready to be invoked to build the project. However, the project and tool build options need to be checked.

1. Select *Project>Build Options* and click on "Project" to display the Build Options dialog for the entire project.
2. Click the **Directories** tab. For this tutorial, check that the "Library Search Path" is the `\lib` directory of the 16-bit compiler installation directory. Also, "Assemble/Compiler/Link in the project directory" should be selected.

FIGURE 2-6: BUILD OPTIONS DIALOG


Build Options For Project "MyProject.mcp"

Directories and Search Paths

Show directories for: Library Search Path

C:\Program Files\Microchip\MPLAB C30\lib

Build Directory Policy

( ) Assemble/Compile in source-file directory, link in output directory

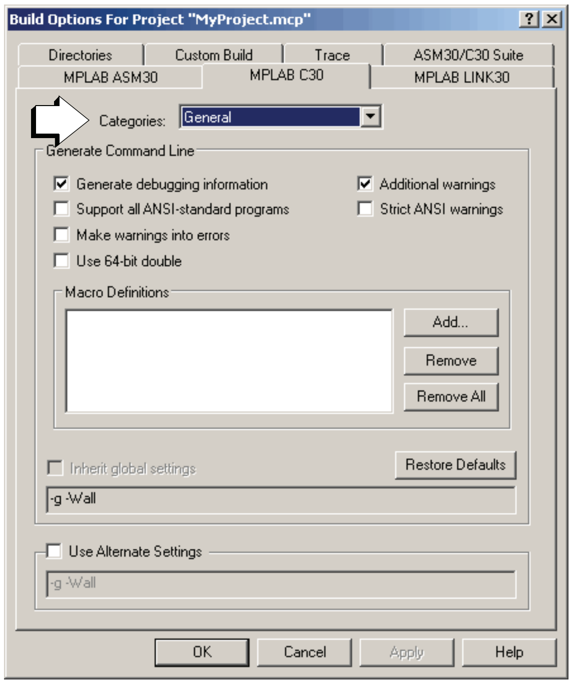
(•) Assemble/Compile/Link in the project directory

The various command-line options that are passed to the 16-bit tools can be set on the tool-specific tabs.

3. Click the **MPLAB C30** tab. There are three dialogs of options: General, Memory Model and Optimization. These are selected in the “Categories” pull-down list and will change the items on the dialog accordingly.

   For this example, you will keep the default command-line options for the 16-bit compiler.

FIGURE 2-7: COMPILER BUILD OPTIONS – GENERAL



Build Options For Project "MyProject.mcp"

Categories: General

Generate Command Line

- [✓] Generate debugging information
- [ ] Support all ANSI-standard programs
- [ ] Make warnings into errors
- [ ] Use 64-bit double
- [✓] Additional warnings
- [ ] Strict ANSI warnings

[ ] Inherit global settings

```
-g -Wall
```

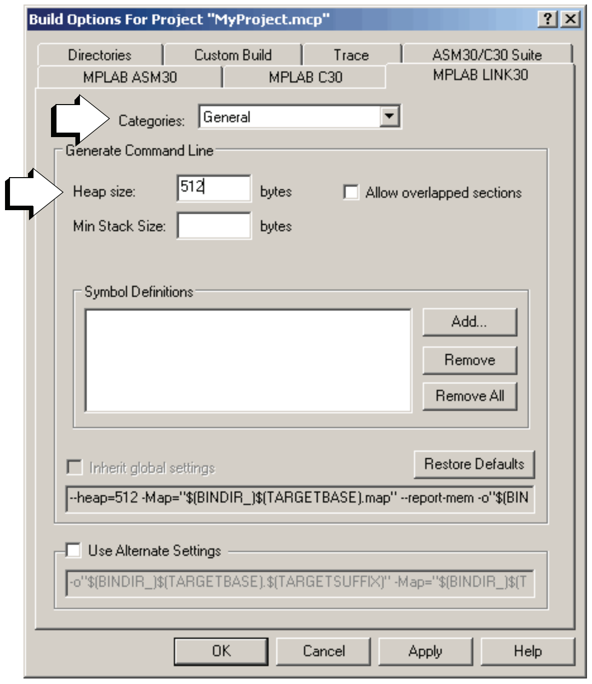
[ ] Use Alternate Settings

```
-g -Wall
```

4. Click the **MPLAB LINK30** tab. There are three dialogs of options: General, Diagnostics and Symbols & Output. These are selected in the “Categories” pull-down list and will change the items on the dialog accordingly.

   The 16-bit linker needs to have a heap entered on its “General” category in order to run Tutorial 3 later in this guide. Enter 512 as the Heap size.

FIGURE 2-8: LINKER BUILD OPTIONS – GENERAL



Categories: General

Generate Command Line

Heap size: 512 bytes

Min Stack Size: bytes

☐ Allow overlapped sections

☐ Inherit global settings

```
--heap=512 -Map="$(BINDIR_)$(TARGETBASE).map" --report-mem -o"$(BIN
```

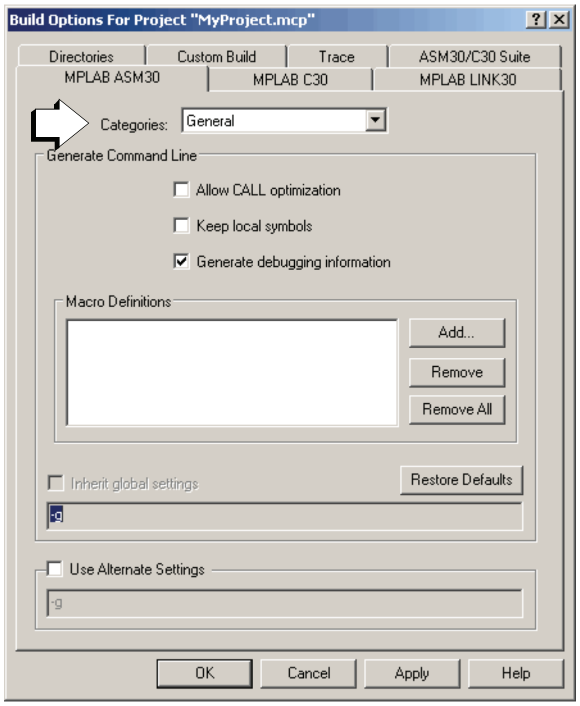
☐ Use Alternate Settings

```
-o"$(BINDIR_)$(TARGETBASE).$(TARGETSUFFIX)" -Map="$(BINDIR_)$(T
```

5. Click the **MPLAB ASM30** tab. There are two dialogs of options: General and Diagnostics. These are selected in the “Categories” pull-down list and will change the items on the dialog accordingly.

   For this example, you will keep the default command-line options for the 16-bit assembler.

FIGURE 2-9: ASSEMBLER BUILD OPTIONS – GENERAL



Build Options For Project "MyProject.mcp"

Categories: General

Generate Command Line

- [ ] Allow CALL optimization
- [ ] Keep local symbols
- [x] Generate debugging information

Macro Definitions

- [ ] Inherit global settings

-g

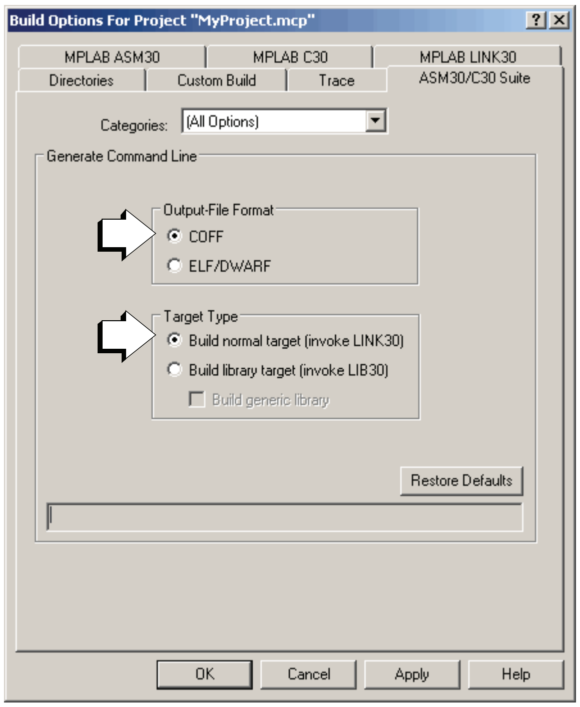
- [ ] Use Alternate Settings

-g

6. Click the **ASM30/C30 Suite** tab. On this tab, the type of output for the assembler and compiler may be selected. Select an output format of either COFF or ELF/DWARF. Select to create regular output (from the linker) or a library (from the librarian).

   For this example, you will keep the default options.

FIGURE 2-10: SUITE BUILD OPTIONS



Build Options For Project "MyProject.mcp"

Categories: (All Options)

Generate Command Line

Output-File Format: [✓ COFF] ELF/DWARF

Target Type: [✓ Build normal target (invoke LINK30)] Build library target (invoke LIB30)

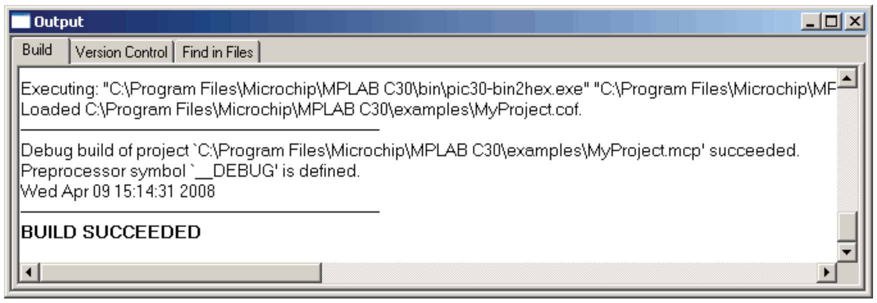
Build generic library

2.6 BUILDING THE PROJECT

Select *Project>Build All* to compile, assemble and link the project. If there are any error or warning messages, they will appear in the Output window.

For this tutorial, the Output window should display no errors and should show a message stating the project "`BUILD SUCCEEDED`." If there were any errors, check to see that the content of the source file matches the text of `MyFile.c` displayed in Example 2-1.

FIGURE 2-11: BUILD OUTPUT WINDOW

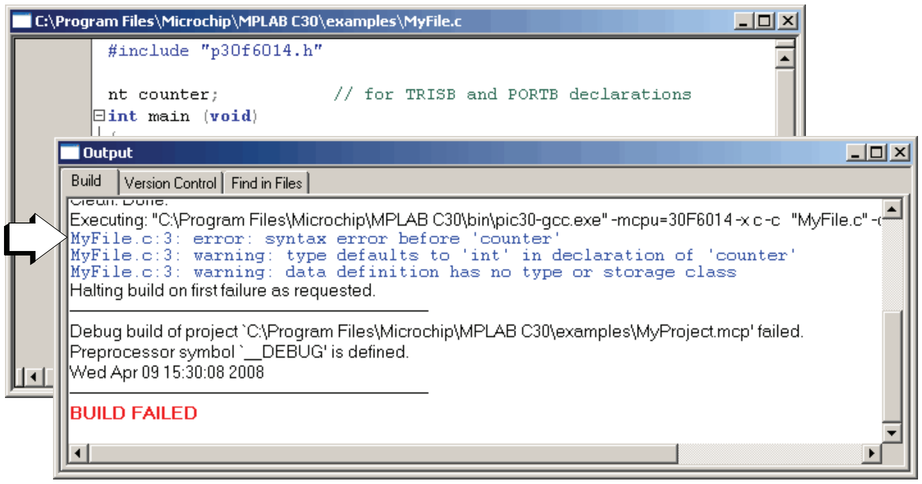


```
Executing: "C:\Program Files\Microchip\MPLAB C30\bin\pic30-bin2hex.exe" "C:\Program Files\Microchip\MF
Loaded C:\Program Files\Microchip\MPLAB C30\examples\MyProject.cof.
----------------------------------------------------------------------
Debug build of project `C:\Program Files\Microchip\MPLAB C30\examples\MyProject.mcp' succeeded.
Preprocessor symbol `__DEBUG' is defined.
Wed Apr 09 15:14:31 2008
----------------------------------------------------------------------
BUILD SUCCEEDED
```

2.7 TROUBLESHOOTING BUILD ERRORS

If errors were reported after building the project, double click on the line with the error message to go directly to the source code line that caused the error. If the example was typed in, the most common errors are misspellings, missing semicolons or unmatched braces. In the following screen, a typo was made. In this example, the letter "`i`" was accidentally omitted in the "`int`" declaration of `main()`. The error message(s) will appear in the Output window.

FIGURE 2-12: BUILD ERROR

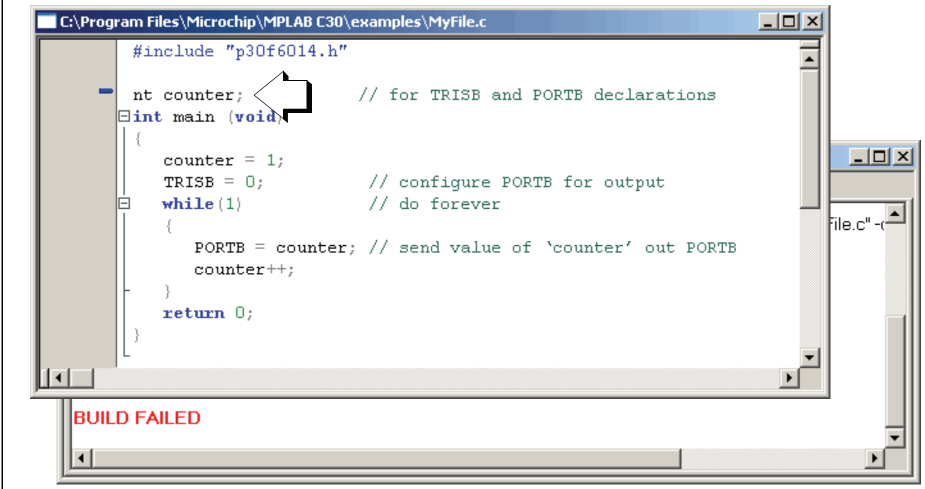


```
#include "p30f6014.h"

nt counter;            // for TRISB and PORTB declarations
int main (void)
```

```
Clean: Done.
Executing: "C:\Program Files\Microchip\MPLAB C30\bin\pic30-gcc.exe" -mcpu=30F6014 -x c -c  "MyFile.c" -
MyFile.c:3: error: syntax error before 'counter'
MyFile.c:3: warning: type defaults to 'int' in declaration of 'counter'
MyFile.c:3: warning: data definition has no type or storage class
Halting build on first failure as requested.
----------------------------------------------------------------------
Debug build of project `C:\Program Files\Microchip\MPLAB C30\examples\MyProject.mcp' failed.
Preprocessor symbol `__DEBUG' is defined.
Wed Apr 09 15:30:08 2008
----------------------------------------------------------------------
BUILD FAILED
```

After double clicking on the error line in the Output window above, the desktop looks like this:

FIGURE 2-13: DOUBLE CLICK TO GO TO SOURCE



```
#include "p30f6014.h"

nt counter;            // for TRISB and PORTB declarations
int main (void)
{
   counter = 1;
   TRISB = 0;           // configure PORTB for output
   while(1)             // do forever
   {
      PORTB = counter; // send value of 'counter' out PORTB
      counter++;
   }
   return 0;
}
```

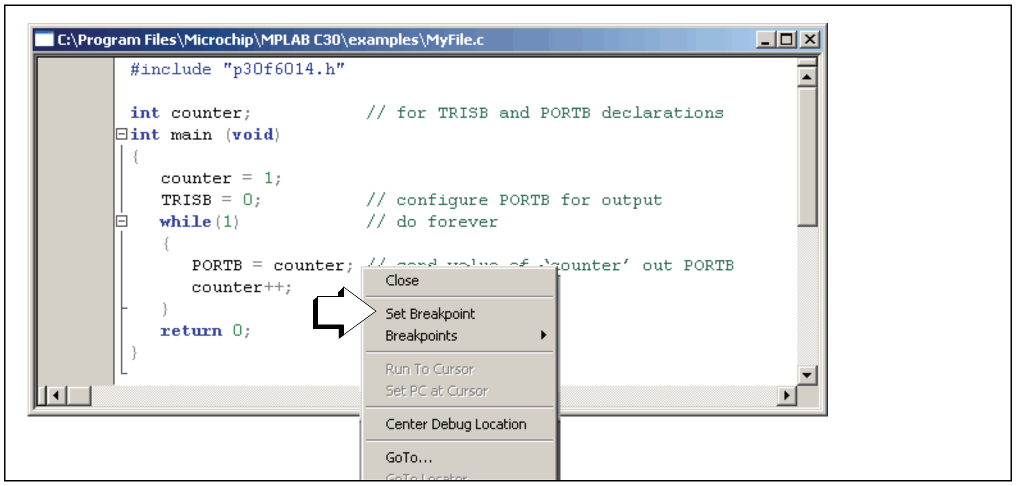
The offending typo "`nt`" is in black text rather than blue – a good indication that something is wrong, since key words are shown in blue color fonts. Typing an "`i`" to make the "`nt`" the proper key word "`int`," results in the text turning blue. Selecting *Project>Project Build All* again produces a successful build.

2.8 DEBUGGING WITH THE MPLAB SIM SIMULATOR

To debug application code, you need the help of a debug tool. In this tutorial, we will use the MPLAB SIM simulator. In the simulator, breakpoints can be set in the source code and the value of variables can be observed with a Watch window.

1. Select the MPLAB SIM simulator as the debugging tool by selecting *Debugger>Select Tool>MPLAB SIM*.
2. Open the source file by double clicking on its name (`MyFile.c`) in the project tree of the Project window. In the source file, place the cursor over the line:
   `PORTB = counter;`
   Then click the right mouse button and select "Set Breakpoint".

FIGURE 2-14: SET BREAKPOINT

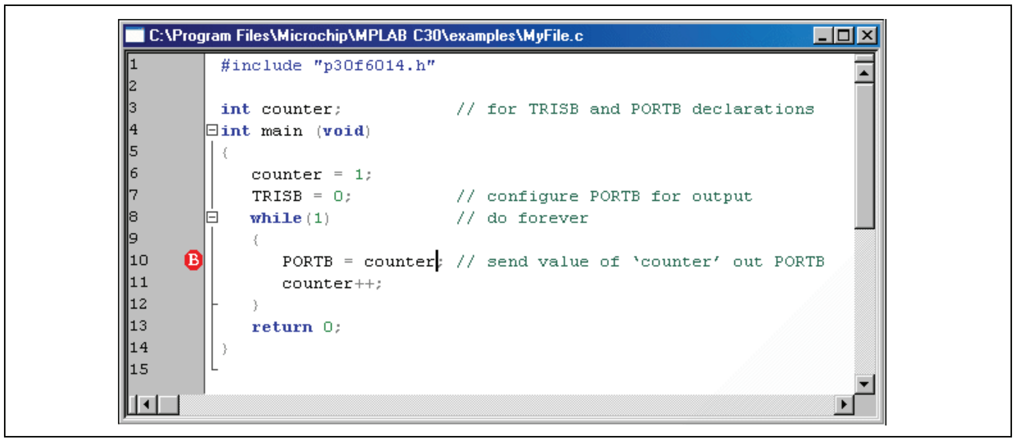


C:\Program Files\Microchip\MPLAB C30\examples\MyFile.c

```
#include "p30f6014.h"

int counter;            // for TRISB and PORTB declarations
int main (void)
{
   counter = 1;
   TRISB = 0;           // configure PORTB for output
   while (1)            // do forever
   {
      PORTB = counter;  // send value of 'counter' out PORTB
      counter++;
   }
   return 0;
}
```

The red stop sign symbol in the margin along the left side of the Source window indicates that the breakpoint has been set and is enabled.

FIGURE 2-15: BREAKPOINT IN SOURCE WINDOW

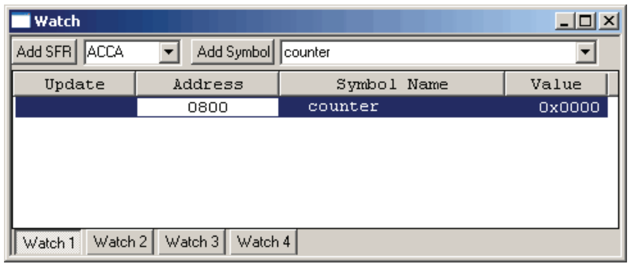


C:\Program Files\Microchip\MPLAB C30\examples\MyFile.c

```
#include "p30f6014.h"

int counter;            // for TRISB and PORTB declarations
int main (void)
{
   counter = 1;
   TRISB = 0;           // configure PORTB for output
   while (1)            // do forever
   {
      PORTB = counter;  // send value of 'counter' out PORTB
      counter++;
   }
   return 0;
}
```

3. Select *View>Watch* to open a Watch window. Select `counter` from the pull-down expandable menu next to **Add Symbol** and then click **Add Symbol**.

FIGURE 2-16: ADD WATCH VARIABLE



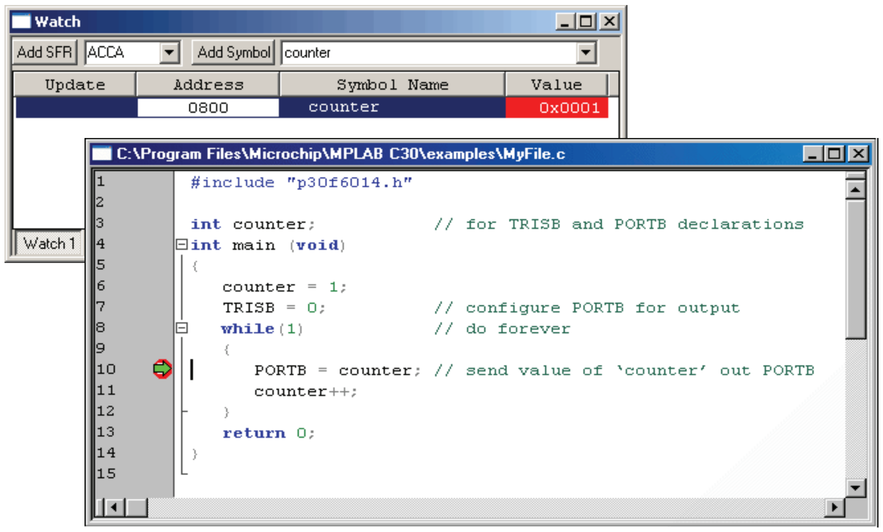
Update	Address	Symbol Name	Value
	0800	counter	0x0000

Note: There are three ways to enter Watch variables: (1) in the method described above, a variable can be picked from a list, (2) the symbol's name can be typed directly in the Symbol Name column in the Watch window or (3) the variable's name can be highlighted in the source text and dragged to the Watch window.

4. Press **Run** on the toolbar to run the program.

The program should halt just before the statement at the breakpoint is executed. The green arrow in the left margin of the Source window points to the next statement to be executed. The Watch window should show `counter` with a value of '`1`'. The value of '`1`' will be shown in red, indicating that this variable has changed.

FIGURE 2-17: RUN TO BREAKPOINT



Update	Address	Symbol Name	Value
	0800	counter	0x0001

```
#include "p30f6014.h"

int counter;          // for TRISB and PORTB declarations
int main (void)
{
   counter = 1;
   TRISB = 0;         // configure PORTB for output
   while(1)           // do forever
   {
      PORTB = counter; // send value of 'counter' out PORTB
      counter++;
   }
   return 0;
}
```

5. Press **Run** again to continue the program. Execution will continue in the `while` loop until it halts again at the line with the breakpoint. The Watch window should show `counter` with a value of ‘`2`’.
6. To step through the source code one statement at a time, use **Step Into** on the toolbar.

   As each statement executes, the green arrow in the margin of the Source window moves to the next statement to be executed.
7. Place the cursor on the line with the breakpoint, and use the right mouse button menu to select “Remove Breakpoint”. Now press the Run button. The “Running...” message should appear on the lower left of the Status bar, and next to it, a moving bar will indicate that the program is running. The Step icon to the right of the Run icon will be grayed out. If the Debugger menu is pulled down, the Step options will also be grayed out. While in the Run mode, these operations are disabled.

   To interrupt a running program, use **Halt** on the toolbar.

   Once the program has stopped (halted), the step icons are no longer grayed out.

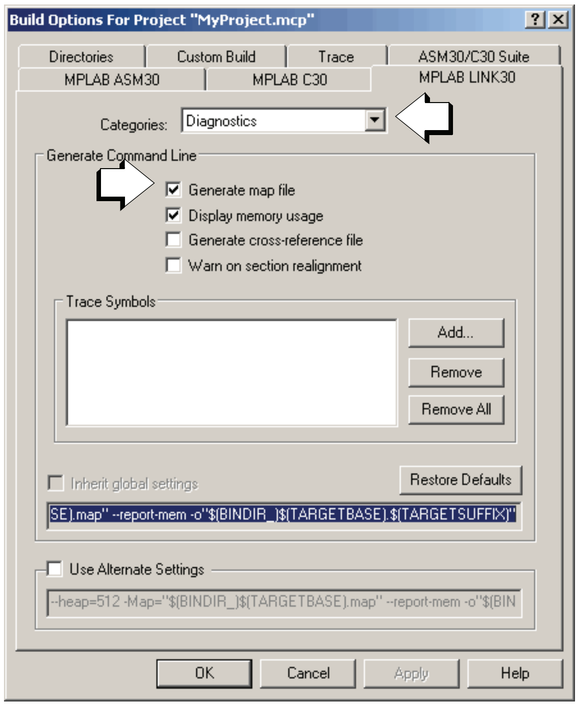
Note: There are two basic modes while debugging: Halt or Run. Most debugging operations are done in Halt mode. In Run mode, most debug functions are not operational. Registers cannot be inspected or changed and a project cannot be rebuilt. Functions that try to access the memory or internal registers of the running target will not be available in Run mode.

2.9 GENERATING A MAP FILE

A map file provides additional information that may be useful in debugging, such as details of memory allocation. This file can be generated by setting the appropriate linker build option.

1. Select *Project>Build Options>Project*, and then click the **MPLAB LINK30** tab.
2. Select “Diagnostics” from “Categories” and then click on the checkbox for “Generate map file”.
3. Click **OK** to save the option.
4. Rebuild the project (*Project>Build All*) to generate the map file.

FIGURE 2-18: GENERATE MAP FILE



Build Options For Project "MyProject.mcp"

Categories: Diagnostics

Generate Command Line

- [✓] Generate map file
- [✓] Display memory usage
- [ ] Generate cross-reference file
- [ ] Warn on section realignment

[ ] Inherit global settings

```
SE).map" --report-mem -o"$(BINDIR_)$(TARGETBASE).$(TARGETSUFFIX)"
```

[ ] Use Alternate Settings

```
--heap=512 -Map="$(BINDIR_)$(TARGETBASE).map" --report-mem -o"$(BIN
```

The map file (`MyProject.map`) is present in the project directory and may be opened by selecting *File>Open*, and then browsing to the project directory. Select Files of Type “All files(*.)” in order to see the map file. The excerpt in Example 2-2 from the `MyProject.map` file shows the program and data memory area usage after `MyProject.C` was compiled.

This information can also be made visible in the Output window by checking the “Display Memory Usage” checkbox on the **MPLAB LINK30** tab, Diagnostics category.

EXAMPLE 2-2: MAP FILE EXCERPT

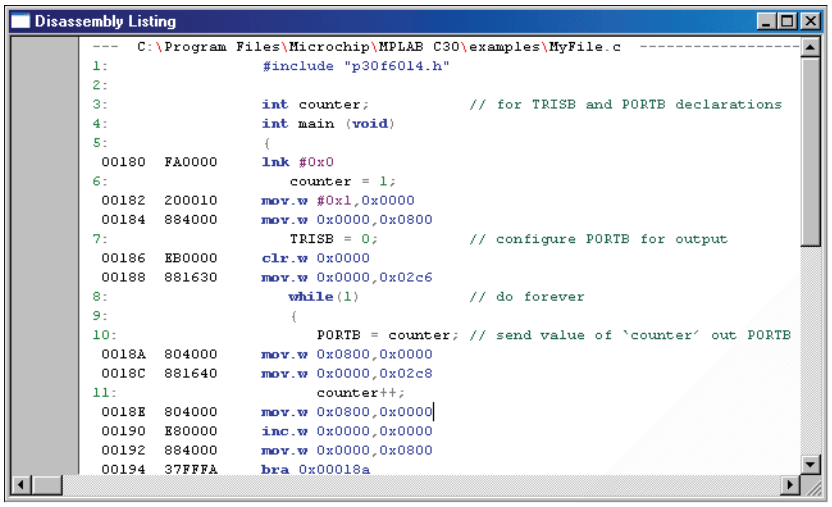
```
Program Memory Usage

section         address   length (PC units)   length (bytes) (dec)
-------         -------   -----------------   --------------------
.reset                0                 0x4              0x6  (6)
.ivt                0x4                0x7c             0xba  (186)
.aivt              0x84                0x7c             0xba  (186)
.text             0x100                0x96             0xe1  (225)
.dinit            0x196                 0x8              0xc  (12)
.isr              0x19e                 0x2              0x3  (3)
         Total program memory used (bytes):            0x26a  (618)  <1%

Data Memory Usage

section         address      alignment gaps    total length  (dec)
-------         -------      --------------    -------------------
.nbss             0x800                   0              0x2  (2)
            Total data memory used (bytes):              0x2  (2)  <1%
```

2.10 DEBUGGING AT ASSEMBLY CODE LEVEL

So far all debugging has been done from the C source file, using functions and variables as defined in the C code. For embedded systems programming, it may be necessary to dig down deeper into the assembly code level. MPLAB IDE provides tools to do both, and shows the correlation between the C code and the generated machine code.

1. Select the MPLAB IDE *View>Disassembly Listing* window to see the source code interspersed with the generated machine and assembly code. This is useful when debugging mixed C and assembly code, and when it is necessary to see the machine code generated from the C source code.

FIGURE 2-19: DISASSEMBLY WINDOW



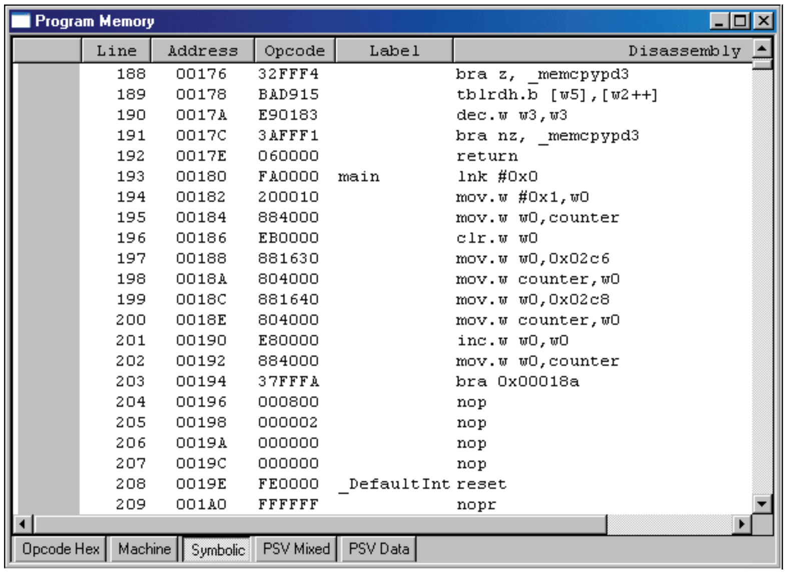
```
---  C:\Program Files\Microchip\MPLAB C30\examples\MyFile.c  -----------------
1:                   #include "p30f6014.h"
2:
3:                   int counter;            // for TRISB and PORTB declarations
4:                   int main (void)
5:                   {
 00180   FA0000      lnk #0x0
6:                      counter = 1;
 00182   200010      mov.w #0x1,0x0000
 00184   884000      mov.w 0x0000,0x0800
7:                      TRISB = 0;           // configure PORTB for output
 00186   EB0000      clr.w 0x0000
 00188   881630      mov.w 0x0000,0x02c6
8:                      while(1)             // do forever
9:                      {
10:                        PORTB = counter; // send value of `counter' out PORTB
 0018A   804000      mov.w 0x0800,0x0000
 0018C   881640      mov.w 0x0000,0x02c8
11:                        counter++;
 0018E   804000      mov.w 0x0800,0x0000
 00190   E80000      inc.w 0x0000,0x0000
 00192   884000      mov.w 0x0000,0x0800
 00194   37FFFA      bra 0x00018a
```

The C source code is shown with the line number from the source code file shown on the left column. The generated machine hex code and the corresponding disassembled instructions are shown with the address in the left column. For the machine code instructions, the left column is the address of the instruction in program memory, followed by the hexadecimal bytes for the instruction and then the dsPIC30F disassembled instruction.

Note: The Disassembly Listing does not display dsPIC30F instructions correctly. Working register operands, W0 through W15, are displayed as hexadecimal numbers, 0x0000 through 0x001E. For example, register W1 is displayed as 0x0002, and register W15 is displayed as 0x001E.

2. Select *View>Program Memory* window to see only the machine and assembly code in program memory.

FIGURE 2-20: PROGRAM MEMORY WINDOW – SYMBOLIC



```
Line  Address  Opcode  Label        Disassembly
 188  00176    32FFF4               bra z, _memcpypd3
 189  00178    BAD915               tblrdh.b [w5],[w2++]
 190  0017A    E90183               dec.w w3,w3
 191  0017C    3AFFF1               bra nz, _memcpypd3
 192  0017E    060000               return
 193  00180    FA0000  main         lnk #0x0
 194  00182    200010               mov.w #0x1,w0
 195  00184    884000               mov.w w0,counter
 196  00186    EB0000               clr.w w0
 197  00188    881630               mov.w w0,0x02c6
 198  0018A    804000               mov.w counter,w0
 199  0018C    881640               mov.w w0,0x02c8
 200  0018E    804000               mov.w counter,w0
 201  00190    E80000               inc.w w0,w0
 202  00192    884000               mov.w w0,counter
 203  00194    37FFFA               bra 0x00018a
 204  00196    000800               nop
 205  00198    000002               nop
 206  0019A    000000               nop
 207  0019C    000000               nop
 208  0019E    FE0000  _DefaultInt  reset
 209  001A0    FFFFFF               nopr
```

By selecting the various tabs at the bottom of the Program Memory window, the code can be viewed with or without symbolic labels, as a raw hex dump, as mixed PSV code and data, or just as PSV data.

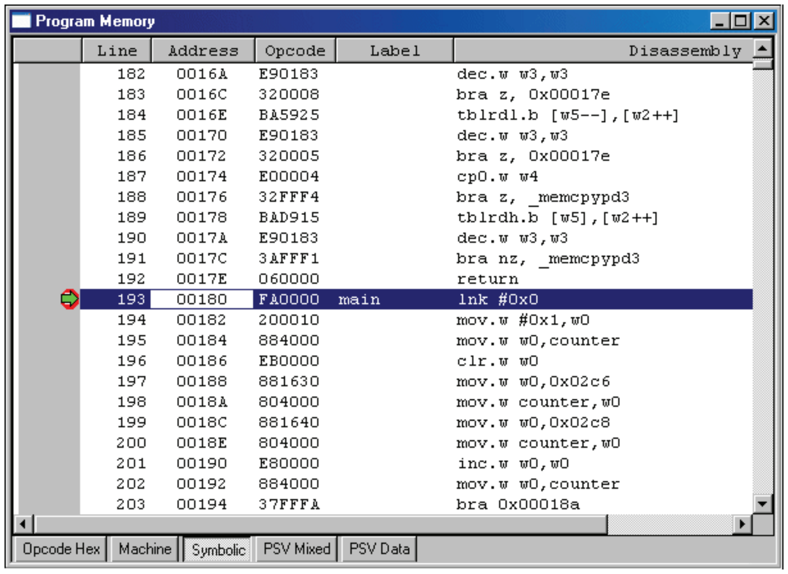
Note: See the device data sheet for more information about PSV data.

Breakpoints can be set, single stepped and all debug functions performed in any of the Source code, Disassembly and Program Memory windows.

3. Make sure the program is halted by pressing the Halt button. In the Program Memory window, click on the Symbolic tab at the bottom to view the code tagged with symbols. Scroll down and click on the line with the label `main`, which corresponds to the `main()` function in the C file. Use the right mouse button to set a breakpoint on `main`. Press the Reset icon (or select to *Debugger>Reset* and select Processor Reset).

4. Now press Run. The program should halt at the breakpoint set at `main`.

FIGURE 2-21: BREAKPOINT IN PROGRAM MEMORY



Program Memory

```
Line  Address  Opcode  Label  Disassembly
182   0016A    E90183         dec.w w3,w3
183   0016C    320008         bra z, 0x00017e
184   0016E    BA5925         tblrdl.b [w5--],[w2++]
185   00170    E90183         dec.w w3,w3
186   00172    320005         bra z, 0x00017e
187   00174    E00004         cp0.w w4
188   00176    32FFF4         bra z, _memcpypd3
189   00178    BAD915         tblrdh.b [w5],[w2++]
190   0017A    E90183         dec.w w3,w3
191   0017C    3AFFF1         bra nz, _memcpypd3
192   0017E    060000         return
193   00180    FA0000  main   lnk #0x0
194   00182    200010         mov.w #0x1,w0
195   00184    884000         mov.w w0,counter
196   00186    EB0000         clr.w w0
197   00188    881630         mov.w w0,0x02c6
198   0018A    804000         mov.w counter,w0
199   0018C    881640         mov.w w0,0x02c8
200   0018E    804000         mov.w counter,w0
201   00190    E80000         inc.w w0,w0
202   00192    884000         mov.w w0,counter
203   00194    37FFFA         bra 0x00018a
```

5. Go back and look at the Source File window (*File>Open*) and the Disassembly window (*View>Disassembly Listing*). The breakpoint should be seen in all three windows. The step function can now be used in any window to single step through C source lines or to single step through the machine code.

2.11 EXPLORING FURTHER

Go ahead and experiment with this example program. Things to explore include:

- Changing the value of `counter` by clicking on its value in the Watch window and typing in a new number.
- Assigning `counter` an initial value of one in its definition. Inspect the source code to see where `counter` is loaded with this value.

Chapter 3. Tutorial 2 – Real-Time Interrupt

3.1 INTRODUCTION

This next tutorial demonstrates real-time interrupt code implemented using the basic “template” file that comes with MPLAB IDE software. Timer1 on the dsPIC30F6014 will be used to generate a recurring interrupt to measure one-second intervals.

This tutorial consists of:

- Using Template Files
- Using the Template in a New Project
- Debugging with the MPLAB SIM Simulator
- Exploring Further

3.2 USING TEMPLATE FILES

Template files are source code files that can serve as a structure to build an application. They make it easy to start a project for an application since the C constructs and formats are provided in a simple file where details of an application can be added. The templates have example C statements for many common features of the MPLAB C Compiler for PIC24 MCUs and dsPIC DSCs (formerly MPLAB C30) source code, including variables and constants, processor-specific include files, interrupt vectors and associated interrupt code, plus areas to insert application code.

The template has comments to help identify key constructs. In many cases, macros are defined to make some things easier. In the simplest form, here is a “stripped-down” template without these comments and macros so its basic structure can be seen:

EXAMPLE 3-1: ELEMENTS OF A TEMPLATE FILE

```
#include <p30fxxxx.h>                           /* proc specific header */

#define CONSTANT1 10                            /* sample constant definitions */
#define CONSTANT2 20

int array1[CONSTANT1] __attribute__((space(xmemory), aligned(32)));
int array2[CONSTANT1] __attribute__((space(ymemory), aligned(32)));
                                                /* arrays with dsPIC30F attributes */

int array3[CONSTANT1] _XBSS(32);                /* arrays with dsPIC30F macros */
int array4[CONSTANT1] _YBSS(32);

int array5[CONSTANT2];                          /* simple array */

int variable1 __attribute__((space(xmemory)));
int variable2 __attribute__((space(ymemory)));
                                                /* variables with attributes */

int variable3;                                  /* simple variable */
int main ( void )                              /* start of main application code */
  {
  /* Application code goes here */
  }

/* Interrupt Service Routine 1 */
void __attribute__((interrupt, auto_psv)) _ADCInterrupt(void)
  {
  /* Interrupt Service Routine code goes here */
  }

/* Interrupt Service Routine 2 */
void __attribute__((interrupt, shadow, auto_psv)) _T1Interrupt(void)
  {
  /* Interrupt Service Routine code goes here */
  }

/* Interrupt Service Routine 3 */
void __attribute__((interrupt(save(variable1,variable2)), auto_psv))
_INT0Interrupt(void)
  {
  /* Interrupt Service Routine code goes here */
  }
```

This template code starts out with the `#include` statement to include the header file that has the dsPIC30F device-specific Special Function Register definitions. Following this are simple constant definitions (`#define`) that can be modified and copied to make a list of constants for the application.

Two array definitions follow to show how to define an array with various attributes, specifying its section in memory, and how it is aligned in the memory architecture of the dsPIC DSC device. Two more array definitions show how to accomplish the same thing using macros defined in the header file. The final array definition, `array5`, is a simple array.

Like arrays, variables can be assigned with attributes (`variable1` and `variable2`), or with no attributes (`variable3`).

A code fragment for `main()` follows. This is where code for the application can be placed. Following `main()` is the code framework for several interrupts.

Actual applications may use different interrupts, different attributes, and will be more complicated than this, but this template provides a simple place to start. The unmodified template can be added to a new project and the project will build with no errors.

Templates are stored in a folder with the 16-bit compiler installation directory named `\support\templates`, and are provided for both assembler and compiler source files in the corresponding `\assembly` and `\c` folders.

The full source code for the C template file for the dsPIC30F6014 is shown in Example 3-2.

EXAMPLE 3-2: `temp_6014.c` TEMPLATE FILE

```
/**********************************************************************
 * This file is a basic template for creating C code for a dsPIC30F    *
 * device.  Copy this file into your project directory and modify or   *
 * add to it as needed.                                                *
 *                                                                     *
 * If interrupts are not used, all code presented for that interrupt   *
 * can be removed or commented out with C-style comment declarations.  *
 *                                                                     *
 * For additional information about dsPIC architecture and language    *
 * tools, refer to the following documents:                            *
 *                                                                     *
 * 16-Bit Language Tools Getting Started                 : DS51316    *
 * MPLAB C Compiler for                                                *
 *          PIC24 MCUs and dsPIC DSCs User's Guide       : DS51284    *
 * MPLAB Assembler, Linker and Utilities for                           *
 *          PIC24 MCUs and dsPIC DSCs User's Guide       : DS51317    *
 * 16-Bit Language Tools Libraries                       : DS51456    *
 *                                                                     *
 * For the latest Datasheets and Family Reference Guides, please       *
 * see the Microchip web site.                                         *
 *                                                                     *
 * Template file has been compiled with MPLAB C30 v3.10.               *
 *                                                                     *
 **********************************************************************
 *                                                                     *
 *    Author:                                                          *
 *    Company:                                                         *
 *    Filename:       temp_6014.c                                      *
 *    Date:           4/9/2008                                         *
 *    File Version:   3.10                                             *
 *    Other Files Required: p30f6014.h, p30f6014.gld                   *
 *    Tools Used: MPLAB IDE -> 8.10                                    *
 *                Compiler  -> 3.10                                    *
 *                                                                     *
 *    Devices Supported:                                               *
 *                dsPIC30F2011                                         *
 *                dsPIC30F3012                                         *
 *                dsPIC30F2012                                         *
 *                dsPIC30F3013                                         *
 *                dsPIC30F3014                                         *
 *                dsPIC30F4013                                         *
 *                dsPIC30F5011                                         *
 *                dsPIC30F6011                                         *
 *                dsPIC30F6012                                         *
 *                dsPIC30F5013                                         *
 *                dsPIC30F6013                                         *
 *                dsPIC30F6014                                         *
 *                                                                     *
 **********************************************************************
 *                                                                     *
 * Other Comments:                                                     *
 *                                                                     *
 * 1) C attributes, designated by the __attribute__ keyword, provide a *
 *    means to specify various characteristics of a variable or        *
 *    function, such as where a particular variable should be placed   *
 *    in memory, whether the variable should be aligned to a certain   *
 *    address boundary, whether a function is an Interrupt Service     *
 *    Routine (ISR), etc. If no special characteristics need to be     *
 *    specified for a variable or function, then attributes are not    *
 *    required. For more information about attributes, refer to the    *
 *    C30 User's Guide.                                                *
 *                                                                     *
 * 2) The space(xmemory) and space(ymemory) attributes are             *
 *    used to place a variable in X data space and Y data space,       *
 *    respectively. Variables accessed by dual-source DSP instructions *
 *    must be defined using these attributes.                          *
 *                                                                     *
 * 3) The aligned(k) attribute, used in variable definitions, is used  *
 *    to align a variable to the nearest higher 'k'-byte address       *
 *    boundary. 'k' must be substituted with a suitable constant       *
 *    number when the ModBuf_X(k) or ModBuf_Y(k) macro is invoked.     *
 *    In most cases, variables are aligned either to avoid potential   *
 *    misaligned memory accesses, or to configure a modulo buffer.     *
 *                                                                     *
 * 4) The interrupt attribute is used to qualify a function as an      *
 *    interrupt service routine. An interrupt routine can be further   *
 *    configured to save certain variables on the stack, using the     *
 *    save(var-list) directive. If constants or string literals may    *
 *    be accessed using the default constants-in-code memory model,    *
 *    then the auto_psv attribute is included. If no constants or      *
 *    string literals will be accessed, the no_auto_psv attribute      *
 *    can be included, saving a few cycles of latency.                 *
 *                                                                     *
 * 5) The shadow attribute is used to set up any function to           *
 *    perform a fast context save using shadow registers.              *
 *                                                                     *
 * 6) Note the use of double-underscores (__) at the start and end of  *
 *    the attribute keyword mentioned above.                           *
 *                                                                     *
 **********************************************************************/

/* The correct device header file will be selected automatically */

#include <p30fxxxx.h>

/* Define constants here                          */

#define CONSTANT1 10
#define CONSTANT2 20

/************* START OF GLOBAL DEFINITIONS **********/

/* Define arrays: array1[], array2[], etc.        */
/* with attributes, as given below                */

/* either using the entire attribute              */
int array1[CONSTANT1] __attribute__((space(xmemory), aligned(32)));
int array2[CONSTANT1] __attribute__((space(ymemory), aligned(32)));

/* or using macros defined in the header file     */
int array3[CONSTANT1] _XBSS(32);
int array4[CONSTANT1] _YBSS(32);

/* Define arrays without attributes               */

int array5[CONSTANT2];   /* array5 is NOT an aligned buffer */

/* ---------------------------------------------- */
/* Define global variables with attributes          */

int variable1 __attribute__((space(xmemory)));
int variable2 __attribute__((space(ymemory)));

/* Define global variables without attributes       */

int variable3;

/************** END OF GLOBAL DEFINITIONS ***********/

/************* START OF MAIN FUNCTION ***************/

int main ( void )
{
/* Code goes here                                   */
}

/****** START OF INTERRUPT SERVICE ROUTINES *********/

/* Replace the interrupt function names with the    */
/* appropriate names depending on interrupt source. */

/* The names of various interrupt functions for     */
/* each device are defined in the linker script.    */

/* Interrupt Service Routine 1                      */
/* No fast context save, and no variables stacked   */

void __attribute__((interrupt, auto_psv)) _ADCInterrupt(void)
{
/* Interrupt Service Routine code goes here         */
}

/* Interrupt Service Routine 2                      */
/* Fast context save (using push.s and pop.s)       */

void __attribute__((interrupt, shadow, auto_psv)) _T1Interrupt(void)
{
/* Interrupt Service Routine code goes here         */
}

/* Interrupt Service Routine 3: INT0Interrupt       */
/* Save and restore variables var1, var2, etc.      */

void __attribute__((interrupt(save(variable1,variable2)), auto_psv))
_INT0Interrupt(void)
{
/* Interrupt Service Routine code goes here         */
}

/********* END OF INTERRUPT SERVICE ROUTINES ********/
```

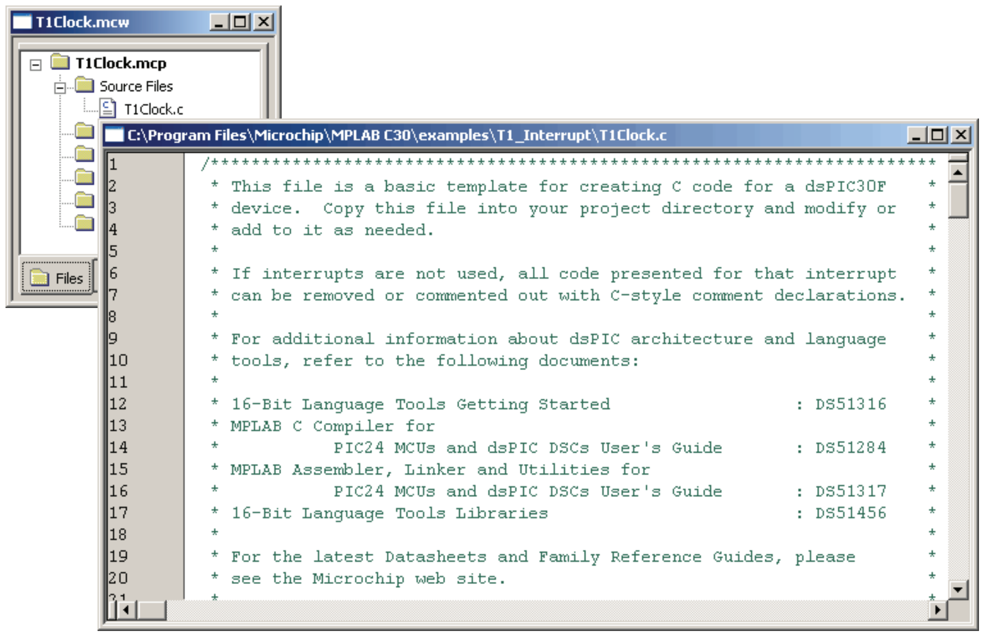
3.3 USING THE TEMPLATE IN A NEW PROJECT

For this tutorial, copy the template described above to a new project directory, following these steps. Go to Windows® Explorer for these folder/file operations.

1. Make a new folder named `\T1_Interrupt` in the `\Examples` directory under the 16-bit compiler installation directory.
2. Copy `C:\Program Files\Microchip\MPLAB C30\support\templates\c\temp_6014.c` to the new `\T1_Interrupt` folder.
3. Rename the copied template file `temp_6014.c` in the `\T1_Interrupt` folder to `T1Clock.c`.
4. Return to MPLAB IDE.

Follow the steps from **Chapter 2. “Tutorial 1 – Creating A Project”** to use the project wizard to create the new project, T1Clock. In this directory, select the C30 Toolsuite, and add `T1Clock.c` as the only source file. After double clicking on the file name, `T1Clock.c` in the Project window, the desktop should look something like this:

FIGURE 3-1: VIEW `T1Clock.c`


T1Clock.mcw
T1Clock.mcp
Source Files
T1Clock.c
Files
C:\Program Files\Microchip\MPLAB C30\examples\T1_Interrupt\T1Clock.c

```
/***********************************************************************
 * This file is a basic template for creating C code for a dsPIC30F    *
 * device.  Copy this file into your project directory and modify or   *
 * add to it as needed.                                                *
 *                                                                     *
 * If interrupts are not used, all code presented for that interrupt   *
 * can be removed or commented out with C-style comment declarations.  *
 *                                                                     *
 * For additional information about dsPIC architecture and language    *
 * tools, refer to the following documents:                            *
 *                                                                     *
 * 16-Bit Language Tools Getting Started                   : DS51316   *
 * MPLAB C Compiler for                                                *
 *          PIC24 MCUs and dsPIC DSCs User's Guide         : DS51284   *
 * MPLAB Assembler, Linker and Utilities for                           *
 *          PIC24 MCUs and dsPIC DSCs User's Guide         : DS51317   *
 * 16-Bit Language Tools Libraries                         : DS51456   *
 *                                                                     *
 * For the latest Datasheets and Family Reference Guides, please       *
 * see the Microchip web site.                                         *
```

Some of the header comments for this generic template can now be removed and application-specific information entered for the new project. The header area at the beginning of the file should contain information on the new project. After editing is finished, it might look something like this:

EXAMPLE 3-3: EDITED `T1Clock.c` HEADER

```
/*********************************************************************
 *                                                                   *
 *    Author:         F. Bar                                         *
 *    Company:        Widgets, Inc.                                  *
 *    Filename:       T1Clock.c                                      *
 *    Date:           4/9/2008                                       *
 *    File Version:   3.10                                           *
 *    Other Files Required: p30f6014.h, p30f6014.gld                 *
 *    Tools Used: MPLAB IDE -> 8.10                                  *
 *                Compiler  -> 3.10                                  *
 *                                                                   *
 *********************************************************************/

#include <p30fxxxx.h>
```

For this tutorial, one constant, two variables and an array need to be defined. Just after the template header, the constants defined are named `CONSTANT1` and `CONSTANT2`. Comment those out, and below the `CONSTANT2` line, add a comment and the definition for `TMR1_PERIOD 0x1388`.

> **Note:** The period 0x1388 = 5000 decimal. The timer will count at a rate one fourth the oscillator frequency. 5000 cycles at 5 MHz (the 20 MHz oscillator is divided by four) yields a time-out for the counter at every 1 ms.

EXAMPLE 3-4: DEFINE A CONSTANT

```
#include <p30fxxxx.h>

/* Define constants here                            */
/* #define CONSTANT1 10
   #define CONSTANT2 20                             */

/* Timer1 period for 1 ms with FOSC = 20 MHz        */
#define TMR1_PERIOD 0x1388
```

Define some variables to track the code operation in this example. Position these in the `GLOBAL DEFINITIONS` area, after the definition of `variable3`. Add two new integer variables, `main_counter` and `irq_counter`. Then, for the interrupt timer routine, create a structure of three unsigned integer variable elements, `timer`, `ticks` and `seconds`, named `RTclock`:

EXAMPLE 3-5: VARIABLE DEFINITIONS

```
/* Define global variables without attributes       */

int variable3;

int main_counter;
int irq_counter;
```

```
struct clockType
    {
    unsigned int timer;    /* countdown timer, milliseconds */
    unsigned int ticks;    /* absolute time, milliseconds */
    unsigned int seconds;  /* absolute time, seconds */
    } RTclock;

/************** END OF GLOBAL DEFINITIONS ***********/
```

The other template code in this tutorial can be left in or commented out. It is probably better to comment it out at this time since these definitions will get compiled and take up memory space. Make sure to comment out all the sample arrays, since they use the macros which can be commented out. Also, as the code grows, it may be difficult to remember which code is used by the application and which was part of the original template.

Note: When using the template, remember that when beginning to code the application, only a few elements of the template may be needed. It may be helpful to comment out those portions of code that are not being used so that later, when similar elements are needed, they can be referred back to as models.

After the section labelled, `END OF GLOBAL DEFINITIONS`, type in the following routine to initialize Timer1 as an interrupt timer using the internal clock:

EXAMPLE 3-6: RESET_CLOCK CODE

```
/************** END OF GLOBAL DEFINITIONS ***********/

void reset_clock(void)
  {
  RTclock.timer = 0;    /* clear software registers */
  RTclock.ticks = 0;
  RTclock.seconds = 0;
  TMR1 = 0;             /* clear timer1 register */
  PR1 = TMR1_PERIOD;    /* set period1 register */
  T1CONbits.TCS = 0;    /* set internal clock source */
  IPC0bits.T1IP = 4;    /* set priority level */
  IFS0bits.T1IF = 0;    /* clear interrupt flag */
  IEC0bits.T1IE = 1;    /* enable interrupts */
  SRbits.IPL = 3;       /* enable CPU priority levels 4-7 */
  T1CONbits.TON = 1;    /* start the timer */
  }

/************* START OF MAIN FUNCTION ***************/
```

This routine uses Special Function Register names, such as `TMR1` and `T1CONbits.TCS` that are defined in the header (`.h`) file. Refer to the data sheet for more information on these control bits and registers for Timer1.

A main routine and an Interrupt Service Routine may need to be written. The most complex routine is the Interrupt Service Routine. It is executed when Timer1 counts down 0x1388 cycles. It increments a counter `sticks` at each of these 1 ms interrupts until it exceeds one thousand. Then it increments the `seconds` variable in the `RTclock` structure and resets `sticks`. This routine should count time in seconds.

In the section labelled, "START OF INTERRUPT SERVICE ROUTINES", find Interrupt Service Routine 2 (where a template for the _T1Interrupt() code is written), and replace the comment:

"/* Interrupt Service Routine code goes here */"

with the lines of code below:

EXAMPLE 3-7: INTERRUPT SERVICE ROUTINE

```
/* Interrupt Service Routine 2                        */
/* Fast context save (using push.s and pop.s)         */

void __attribute__((interrupt, shadow, auto_psv)) _T1Interrupt(void)
  {
  static int sticks=0;

  irq_counter++;

  if (RTclock.timer > 0)      /* if timer is active */
  RTclock.timer -= 1;         /* decrement it */

  RTclock.ticks++;            /* increment ticks counter */

  if (sticks++ == 1000)
    {                         /* if time to rollover */
    sticks = 0;               /* clear seconds ticks */
    RTclock.seconds++;        /* and increment seconds */
    }

  IFS0bits.T1IF = 0;          /* clear interrupt flag */

  }

/* Interrupt Service Routine 3: INT0Interrupt        */
/* Save and restore variables var1, var2, etc.       */
```

There are three sample interrupt functions in the template file. Comment out _INT0Interrupt() because it uses two of the template file sample variables and, as a result, will not compile. _ADCInterrupt() can be commented out too, since it will not be used in this tutorial.

By comparison to the Timer1 interrupt code, the main() code is simple. Type the following in for the body, replacing the line "/* code goes here */":

EXAMPLE 3-8: MAIN CODE

```
/************* START OF MAIN FUNCTION ***************/

int main ( void )
    {
    reset_clock();

    for (;;)
        main_counter++;
    }

/****** START OF INTERRUPT SERVICE ROUTINES *********/
```

The `main()` code is simply a call to our Timer1 initialization routine, followed by an infinite loop, allowing the Timer1 interrupt to function. Typically, an application that made use of this timer would be placed in this loop in place of this test variable, `main_counter`.

The final code should now look like this:

EXAMPLE 3-9: FINAL C CODE FILE

```
/**********************************************************************
 *                                                                    *
 *    Author:         F. Bar                                          *
 *    Company:        Widgets, Inc.                                   *
 *    Filename:       T1Clock.c                                       *
 *    Date:           4/9/2008                                        *
 *    File Version:   3.10                                            *
 *    Other Files Required: p30f6014.h, p30f6014.gld                  *
 *    Tools Used: MPLAB IDE -> 8.10                                   *
 *                Compiler  -> 3.10                                   *
 *                                                                    *
 **********************************************************************/

/* The correct device header file will be selected automatically */

#include <p30fxxxx.h>

/* Define constants here                          */

/*#define CONSTANT1 10
  #define CONSTANT2 20                            */

/* Timer1 period for 1 ms with FOSC = 20 MHz      */
#define TMR1_PERIOD 0x1388

/************* START OF GLOBAL DEFINITIONS **********/


/* Define arrays: array1[], array2[], etc.        */
/* with attributes, as given below                */

/* either using the entire attribute
int array1[CONSTANT1] __attribute__((space(xmemory), aligned(32)));
int array2[CONSTANT1] __attribute__((space(ymemory), aligned(32)));
*/

/* or using macros defined in the header file
int array3[CONSTANT1] _XBSS(32);
int array4[CONSTANT1] _YBSS(32);
*/

/* Define arrays without attributes               */

/* int array5[CONSTANT2];    array5 is NOT an aligned buffer */

/* ---------------------------------------------- */

/* Define global variables with attributes        */
/*
int variable1 __attribute__((space(xmemory)));
int variable2 __attribute__((space(ymemory)));
*/
```

```
/* Define global variables without attributes       */
/*
int variable3;
*/
int main_counter;
int irq_counter;

struct clockType
    {
    unsigned int timer;    /* countdown timer, milliseconds */
    unsigned int ticks;    /* absolute time, milliseconds */
    unsigned int seconds;  /* absolute time, seconds */
    } RTclock;

/************** END OF GLOBAL DEFINITIONS ***********/

oid reset_clock(void)
  {
  RTclock.timer = 0;    /* clear software registers */
  RTclock.ticks = 0;
  RTclock.seconds = 0;
  TMR1 = 0;             /* clear timer1 register */
  PR1 = TMR1_PERIOD;    /* set period1 register */
  T1CONbits.TCS = 0;    /* set internal clock source */
  IPC0bits.T1IP = 4;    /* set priority level */
  IFS0bits.T1IF = 0;    /* clear interrupt flag */
  IEC0bits.T1IE = 1;    /* enable interrupts */
  SRbits.IPL = 3;       /* enable CPU priority levels 4-7 */
  T1CONbits.TON = 1;    /* start the timer */
  }

/************* START OF MAIN FUNCTION ***************/

int main ( void )
{
    reset_clock();

    for (;;)
        main_counter++;
}


/****** START OF INTERRUPT SERVICE ROUTINES *********/

/* Replace the interrupt function names with the    */
/* appropriate names depending on interrupt source. */

/* The names of various interrupt functions for     */
/* each device are defined in the linker script.    */


/* Interrupt Service Routine 1                      */
/* No fast context save, and no variables stacked   */
/*
void __attribute__((interrupt, auto_psv)) _ADCInterrupt(void)
{


 Interrupt Service Routine code goes here
}

/
/* Interrupt Service Routine 2                     */
/* Fast context save (using push.s and pop.s)      */

void __attribute__((interrupt, shadow, auto_psv)) _T1Interrupt(void)
{

  static int sticks=0;

  irq_counter++;

  if (RTclock.timer > 0)      /* if timer is active */
  RTclock.timer -= 1;         /* decrement it */

  RTclock.ticks++;            /* increment ticks counter */

  if (sticks++ == 1000)
    {                         /* if time to rollover */
    sticks = 0;               /* clear seconds ticks */
    RTclock.seconds++;        /* and increment seconds */
    }

  IFS0bits.T1IF = 0;          /* clear interrupt flag */

}


/* Interrupt Service Routine 3: INT0Interrupt       */
/* Save and restore variables var1, var2, etc.      */
/*
void __attribute__((interrupt(save(variable1,variable2)), auto_psv))
_INT0Interrupt(void)
{

 Interrupt Service Routine code goes here

}

*/
/********* END OF INTERRUPT SERVICE ROUTINES ********/
```

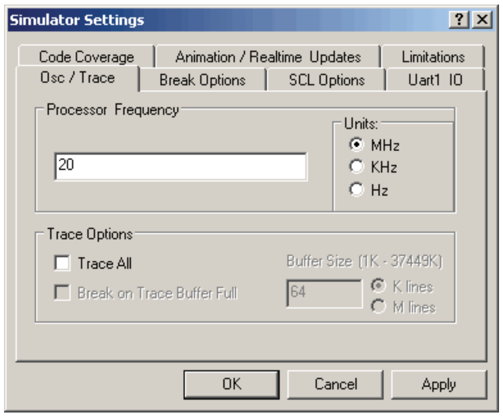
If everything is typed correctly, then selecting *Project>Build All* should result in a successful compilation. Double click on any errors appearing in the Output window to return to the source code to fix typos and rebuild the project until it builds with no errors.

3.4 DEBUGGING WITH THE MPLAB SIM SIMULATOR

The MPLAB SIM simulator can now be used to test the code. Select *Debugger>Select Tool>MPLAB SIM* to choose the simulator. Then set the processor clock speed for the simulator by selecting *Debugger>Settings*. The Oscillator (**Osc/Trace**) tab is used to set the clock frequency of the simulated dsPIC30F6014; set it to 20 MHz.

> **Note:** The simulator runs at a speed determined by the PC, so it will not run at the actual dsPIC30F DSC speed as set by the clock in this dialog. However, all timing calculations are based on this clock setting, so when timing measurements are made using the simulator, times will correspond to those of an actual device running at this frequency.

FIGURE 3-2: STIMULUS OSCILLATOR FREQUENCY


Simulator Settings

Tabs: Code Coverage | Animation / Realtime Updates | Limitations | Osc / Trace | Break Options | SCL Options | Uart1 IO

Processor Frequency: 20

Units: [✓ MHz] KHz Hz

Trace Options

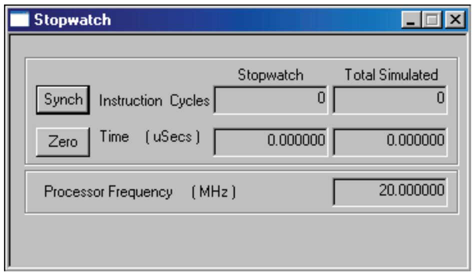
Trace All

Break on Trace Buffer Full

Buffer Size (1K - 37449K): 64 [✓ K lines] M lines

One way to measure time with the simulator is to use the Stopwatch. Select *Debugger>Stopwatch* to view the Stopwatch dialog. The Stopwatch will always clear on simulator Reset.

FIGURE 3-3: SIMULATOR STOPWATCH

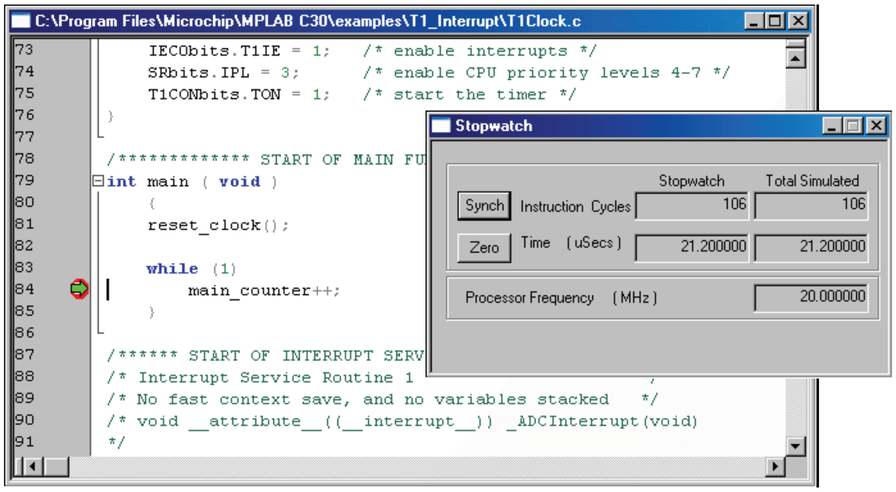

Stopwatch

	Stopwatch	Total Simulated
Instruction Cycles	0	0
Time (uSecs)	0.000000	0.000000

Processor Frequency (MHz) 20.000000

A good first test is to verify that, at a minimum, the program runs. For this purpose, set a breakpoint at the line in `main()` that increments `main_counter` (right mouse click on the line and select "Set Breakpoint"), then press the Run icon or select *Debugger>Run*. The Stopwatch and the screen should like this after the breakpoint is reached.

FIGURE 3-4: TIME MEASUREMENT



C:\Program Files\Microchip\MPLAB C30\examples\T1_Interrupt\T1Clock.c

```
    IECObits.T1IE = 1;    /* enable interrupts */
    SRbits.IPL = 3;       /* enable CPU priority levels 4-7 */
    T1CONbits.TON = 1;    /* start the timer */
}

/************* START OF MAIN FU
int main ( void )
    {
    reset_clock();

    while (1)
        main_counter++;
    }

/****** START OF INTERRUPT SERV
/* Interrupt Service Routine 1
/* No fast context save, and no variables stacked    */
/* void __attribute__((__interrupt__)) _ADCInterrupt(void)
*/
```

Stopwatch

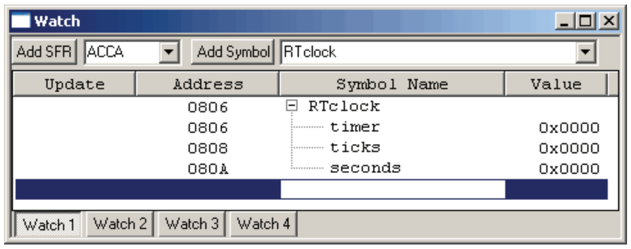
	Stopwatch	Total Simulated
Instruction Cycles	106	106
Time (uSecs)	21.200000	21.200000

Processor Frequency (MHz) 20.000000

If the run was successful, then a Watch window can be set to inspect the program's variables. Select *View>Watch* to bring up a Watch window. Add the variable `RTclock` (from the drop-down box next to **Add Symbol**.)

`RTclock` is a structure, as indicated by the small plus symbol in the box to the left of its name. Click on the box to expand the structure so it looks like this:

FIGURE 3-5: WATCH STRUCTURE VARIABLE

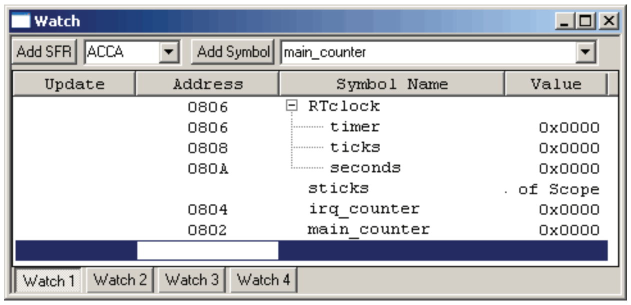


Add SFR: ACCA — Add Symbol: RTclock

Update	Address	Symbol Name	Value
	0806	RTclock	
	0806	timer	0x0000
	0808	ticks	0x0000
	080A	seconds	0x0000

In addition to `RTclock`, add the variables, `sticks`, `irq_counter` and `main_counter`, to the Watch window.

FIGURE 3-6: ALL WATCH VARIABLES



Add SFR: ACCA — Add Symbol: main_counter

Update	Address	Symbol Name	Value
	0806	RTclock	
	0806	timer	0x0000
	0808	ticks	0x0000
	080A	seconds	0x0000
		sticks	. of Scope
	0804	irq_counter	0x0000
	0802	main_counter	0x0000

The Value column may be expanded wider in order to read the text on the `sticks` variable. You will see that it says “Out of Scope”. This means that, unlike `RTclock`, `irq_counter`, and `main_counter`, this is not a global variable, and its value can only be accessed while the function `_T1Interrupt()` is executing.

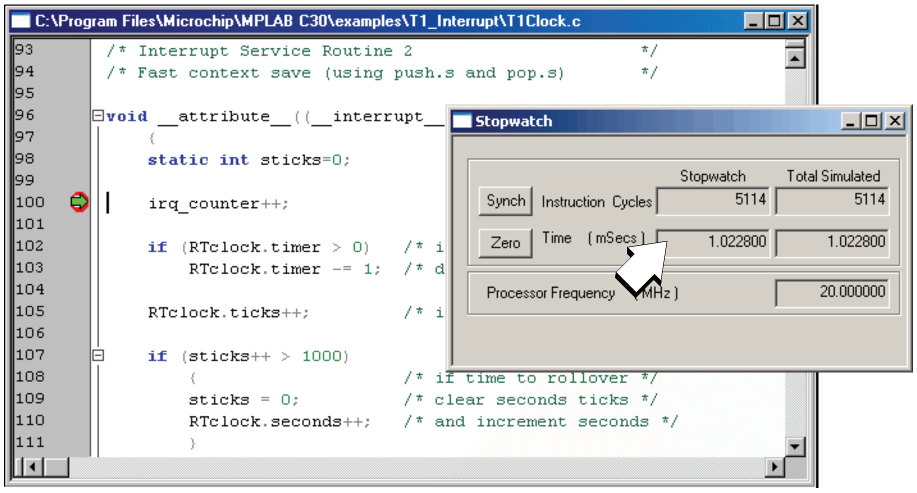
Note: The Address column for `sticks` does not have a value. This is another indication that `sticks` is a local variable.

When inspecting the variables in the Watch window at this first breakpoint, all of them should be equal to zero. This is to be expected, since Timer1 just has been initialized and `counter` has not yet been incremented for the first time.

Press the Step Into icon to step once around the `main()` loop. The value of `main_counter` should now show ‘`0001`’. The interrupt routine has not yet fired. Looking at the Stopwatch window, the elapsed time only increments by a microsecond each time through the `main()` loop. To reach the first interrupt, you would have to step a thousand times (1000 x 1 µs = 1 ms).

In order to test the interrupt functionality, remove the breakpoint at `main_counter++` by clicking on the highlighted line with the right mouse button and select “Remove Breakpoint”. Now select “Set Breakpoint” in the right mouse menu to put a breakpoint in the Interrupt Service Routine at the `irq_counter++` statement. Then, press Run. The Stopwatch should look like this:

FIGURE 3-7: STOPWATCH AT FIRST INTERRUPT


C:\Program Files\Microchip\MPLAB C30\examples\T1_Interrupt\T1Clock.c
Stopwatch
Synch
Zero

```
/* Interrupt Service Routine 2                          */
/* Fast context save (using push.s and pop.s)           */

void __attribute__((__interrupt__
    {
    static int sticks=0;

    irq_counter++;

    if (RTclock.timer > 0)     /* i
        RTclock.timer -= 1;    /* d

    RTclock.ticks++;           /* i

    if (sticks++ > 1000)
        {                      /* if time to rollover */
        sticks = 0;            /* clear seconds ticks */
        RTclock.seconds++;     /* and increment seconds */
        }
```

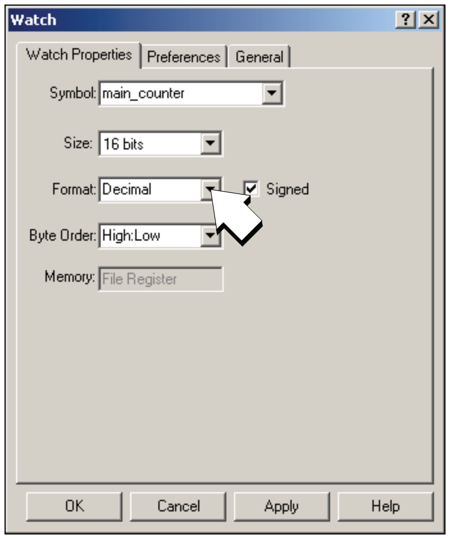
	Stopwatch	Total Simulated
Instruction Cycles	5114	5114
Time (mSecs)	1.022800	1.022800
Processor Frequency (MHz)		20.000000

The value shown in the Time window is 1.0228 ms. This is about what was expected, since the interrupt should happen every millisecond. There was some time since Reset that was counted by the Stopwatch, including the C start-up code and the Timer1 initialization.

Look at the Watch window. The variable, `main_counter`, is showing a value of 0x3E8. To change the radix of this display to decimal, do the following:

1. Click `main_counter` to select the line in the Watch window. Then, using the right mouse button, choose "Properties".
2. In the Watch dialog on the **Watch Properties** tab, select "Decimal" from the "Format" pull-down menu.
3. Click **OK**.

FIGURE 3-8: SET WATCH RADIX

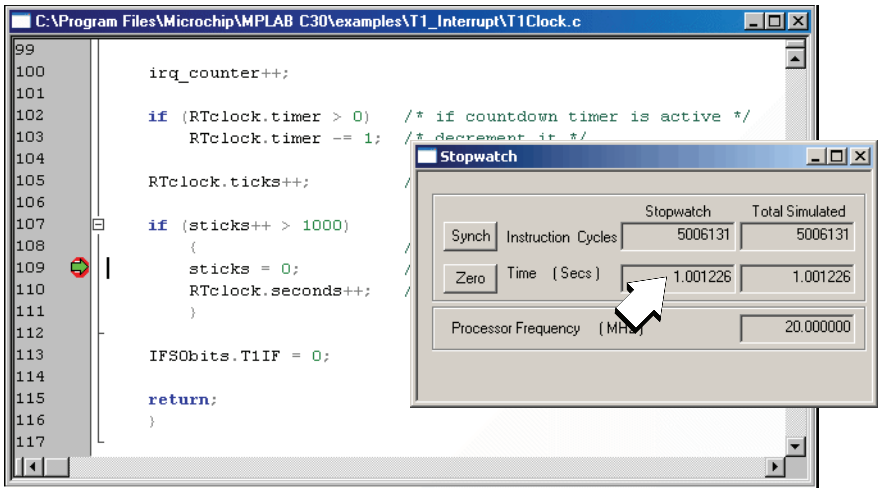


Symbol: main_counter
Size: 16 bits
Format: Decimal ✓ Signed
Byte Order: High:Low
Memory: File Register

The `main_counter` value should now show 1000. Press the Step Into icon a few more times to see the changing variables, especially `sticks` and `irq_counter`, which are incrementing each time the interrupt happens.

Remove the breakpoint from the `irq_counter++;` line, and put a breakpoint inside the conditional statement that increments `sticks` (at the line `sticks = 0;`). Click Run to run and halt at this breakpoint. The window should look like this:

FIGURE 3-9: MEASURE INTERRUPT PERIOD



C:\Program Files\Microchip\MPLAB C30\examples\T1_Interrupt\T1Clock.c

```
    irq_counter++;

    if (RTclock.timer > 0)    /* if countdown timer is active */
        RTclock.timer -= 1;   /* decrement it */

    RTclock.ticks++;          /

    if (sticks++ > 1000)
        {                     /
        sticks = 0;           /
        RTclock.seconds++;    /
        }

    IFS0bits.T1IF = 0;

    return;
    }
```

Stopwatch

		Stopwatch	Total Simulated
Synch	Instruction Cycles	5006131	5006131
Zero	Time (Secs)	1.001226	1.001226
	Processor Frequency (MHz)		20.000000

The Stopwatch Time window shows 1.001226 seconds, which is close to a one-second interrupt. A good time measurement would be to measure the time to the next interrupt. That value could then be subtracted from the current time. Or, since it doesn't matter how much time it took to get here – the main interest is the time between interrupts – press Zero on the Stopwatch and then press Run.

Note: The Stopwatch always tracks total time in the windows on the right side of the dialog. The left windows can be used to time individual measurements. Pressing Zero will not cause the total time to change.

3.5 EXPLORING FURTHER

Go ahead and experiment with this example program. Things to explore include:

- Measure the overhead of the interrupt, calculate how this will affect the timing, and try to adjust the constant, `TMR1_Period`, to adjust the interrupt to get better 1-second accuracy.
- What is the maximum time (in minutes) measured by this routine? What can be done to extend it?
- Add a routine that outputs a two-millisecond pulse every second from a port. Verify the pulse duration with the Stopwatch.

NOTES:

Chapter 4. Tutorial 3 – Mixed C and Assembly Files

4.1 INTRODUCTION

This tutorial will show how to make a project that uses an assembly language routine that is called from a C source file.

This tutorial consists of:

- Getting Project Source Files
- Creating and Building the Project
- Examining the Program
- Exploring Further
- Where to Go from Here

4.2 GETTING PROJECT SOURCE FILES

The files for this tutorial are available in the `\Examples\MPLABC30_Getting_Started` folder and are called `example3.c`, a C source code file, and `modulo.s`, an assembly language file. Create a folder in the `\Examples` folder called `\DSP_ASM` and copy these two files to that new folder. See **Chapter 3. "Tutorial 2 – Real-Time Interrupt"** for how to do this.

For reference, Example 4-1 and Example 4-2 show listings of these two files.

EXAMPLE 4-1: C SOURCE FILE

```
 /*********************************************************************
 *    Filename:     example3.c   for Tutorial 3 in                    *
 *                               16-Bit Language Tools Getting Started *
 *                               The file modulo.s is also used.      *
 *    Date:         04/16/2008                                        *
 *    File Version: 3.10                                              *
 *    Tools used:   MPLAB IDE -> 8.10                                 *
 *                  Compiler  -> 3.10                                 *
 *    Linker File:  p30f6014.gld                                      *
 **********************************************************************/

#include "p30f6014.h"
#include <stdio.h>

/* Length of output buffer (in words) */
#define PRODLEN 20

/* source arrays of 16-bit elements */
unsigned int array1[PRODLEN/2] __attribute__((space(xmemory), aligned(32)));
unsigned int array2[PRODLEN/2] __attribute__((space(ymemory), aligned(32)));

/* output array of 32-bit products defined here */
long array3[PRODLEN];   /* array3 is NOT a circular buffer */

/* Pointer for traversing array */
unsigned int array_index;
```

```
/* 'Point-by-point array multiplication' assembly function prototype */
extern void modulo( unsigned int *, unsigned int *, long *, unsigned int );

int main ( void )
{
/* Set up Modulo addressing for X AGU using W8 and for Y AGU using W10 */
/* Actual Modulo Mode will be turned on in the assembly language routine */

    CORCON |= 0x0001;     /* Enable integer arithmetic */
    XMODSRT = (unsigned int)array1;
    XMODEND = (unsigned int)array1 + PRODLEN - 1;
    YMODSRT = (unsigned int)array2;
    YMODEND = (unsigned int)array2 + PRODLEN - 1;


    while (1)           /* just do this over and over */
    {

/* Initialize array1 and array2 */
        for (array_index = 0; array_index < PRODLEN/2; array_index++)
        {
            array1[array_index] = array1[array_index] + array_index + 1;
            array2[array_index] = array2[array_index] + (array_index+1) * 3;
        }

/* Call assembly subroutine to do point-by-point multiply  */
/* of array1 and array2, with results place in array3.     */
        modulo( array1, array2, array3, PRODLEN-1 );
    }
}
```

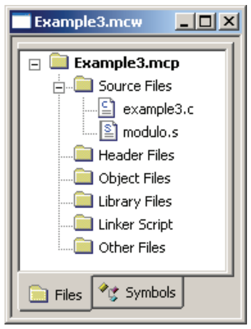
EXAMPLE 4-2: `modulo.s` ASM SOURCE FILE

```
 /*********************************************************************
 *    Filename:     modulo.s     for Tutorial 3 in                    *
 *                               16-Bit Language Tools Getting Started *
 *                               The file example3.c is also used.    *
 *    Date:         04/16/2008                                        *
 *    File Version: 3.10                                              *
 *    Tools used:   MPLAB IDE -> 8.10                                 *
 *                  Compiler  -> 3.10                                 *
 *    Linker File:  p30f6014.gld                                      *
 *    Description: Assembly routine used with example3.c              *
 ********************************************************************
 */

         .text

         .global _modulo
_modulo:

         ; If any of the registers W8 - W15 are used, they should be saved
         ; W0 - W7 may be used without saving
         PUSH   W8
         PUSH   W10

         MOV    #0xC0A8, W8      ; turn on modulo addressing
         MOV    W8, MODCON

         ; The 3 pointers were passed in W0, W1 and W2 when function was called
         ; Transfer pointers to appropriate registers for MPY
         MOV    W0,  W8       ; Initializing X pointer
         MOV    W1,  W10      ; Initializing Y pointer

         ; Clear Accumulator and prefetch 1st pair of numbers
         CLR    A, [W8]+=2, W4, [W10]+=2, W7

         LSR    W3, W3
         RCALL  array_loop    ; do multiply set
         INC2   W8,  W8       ; Change alignment of X pointer
         RCALL  array_loop    ; second multiply set

         POP    W10
         POP    W8

         RETURN
         ; Return to main C program

array_loop:
         ; Set up DO loop with count 'PRODLEN - 1' (passed in W3)
         DO     W3,   here

         ; Do  a point-by-point multiply
         MPY    W4*W7, A, [W8]+=2, W4, [W10]+=2, W7

         ; Store result in a 32-bit array pointed by W2
         MOV    ACCAL, W5
         MOV    W5, [W2++]

         MOV    ACCAH, W5
here:    MOV    W5, [W2++]

         CLR     MODCON         ; turn off modulo addressing

        RETURN

 .end
```

4.3 CREATING AND BUILDING THE PROJECT

Using the Project Wizard, create a new project with these two source files. See **Chapter 2. “Tutorial 1 – Creating A Project”** for how to do this. The Project window should look like this:

FIGURE 4-1: PROJECT WINDOW



This tutorial will use the standard I/O function, `printf()`, to display messages to the Output window. In order to use `printf()`, the build options for the linker need to have the heap enabled. Make sure that the linker build option is set as shown in Figure 2-8 with 512 bytes allocated for the heap.

When building the project (*Project>Build All*), it should compile with no error messages. If an error is received, make sure the project is set up with the same options as for the previous two tutorials.

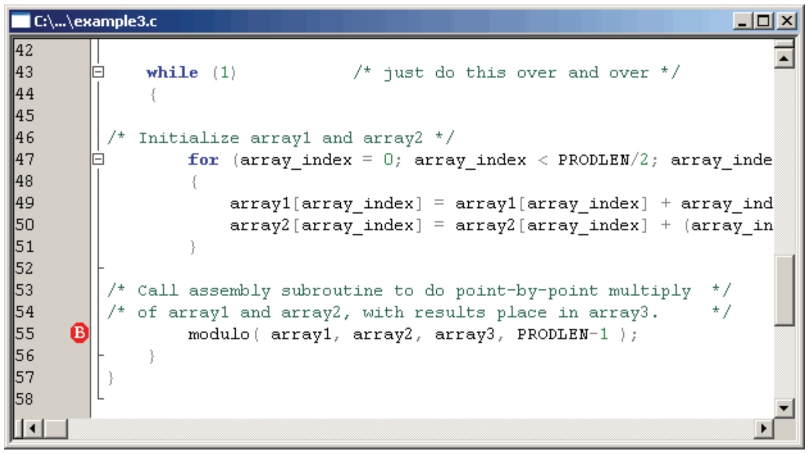
This tutorial sets up three arrays. It fills two of them with a test numerical sequence, then calls an assembly language routine that multiplies the values in the two 16-bit arrays and puts the result into the third 32-bit array. Using modulo arithmetic for addressing, the two source arrays are traversed twice to generate two sets of products in the output array, with the pointer to one array adjusted at the second pass through the multiply loop to change the alignment between the multipliers. Using an assembly language routine ensures that the arithmetic will be done using the DSP features of the dsPIC30F6014.

The assembly language routine takes four parameters: the addresses of each of the three arrays and the array length. It returns its result in the product array. This routine runs in a continual loop, with the source arrays getting increasingly larger numbers as the program repeatedly executes the main endless loop.

4.4 EXAMINING THE PROGRAM

Once the project is set up and successfully built, the operation of the program can be inspected using MPLAB SIM simulator (*Debugger>Select Tool>MPLAB SIM*). Set up and run to a breakpoint on the function that calls the assembly language routine, `modulo()`, from `example3.c`.

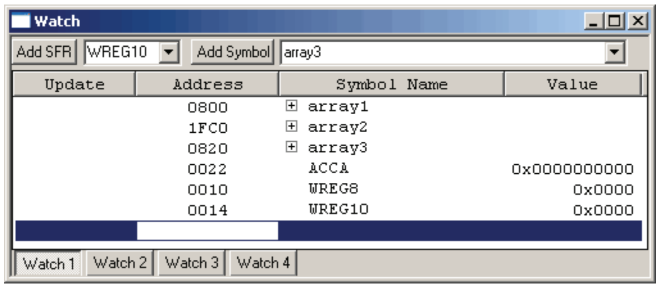
FIGURE 4-2: BREAKPOINT IN C CODE FILE



```
   while (1)            /* just do this over and over */
   {

/* Initialize array1 and array2 */
      for (array_index = 0; array_index < PRODLEN/2; array_inde
      {
         array1[array_index] = array1[array_index] + array_ind
         array2[array_index] = array2[array_index] + (array_in
      }

/* Call assembly subroutine to do point-by-point multiply  */
/* of array1 and array2, with results place in array3.     */
      modulo( array1, array2, array3, PRODLEN-1 );
   }
}
```

Set up a Watch window to look at the variables involved in this calculation. Add the three arrays `array1`, `array2` and `array3`. Also add the SFRs `ACCA`, `WREG8` and `WREG10`. The Watch window should look like this:

FIGURE 4-3: WATCH WINDOW



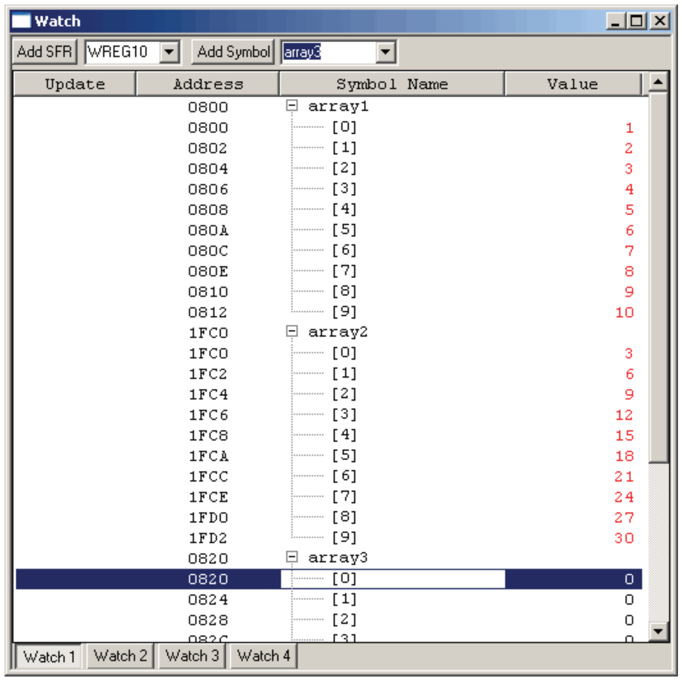
Update	Address	Symbol Name	Value
	0800	⊞ array1	
	1FC0	⊞ array2	
	0820	⊞ array3	
	0022	ACCA	0x0000000000
	0010	WREG8	0x0000
	0014	WREG10	0x0000

Click on the plus symbol to the left of the symbol name to expand the arrays. At this point in the program, both `array1` and `array2` should have been set up with initial values, but `array3` should be all zeros, since the `modulo()` routine has not yet been called.

Click on any element in the array to select the element, and then right click on the element to change the radix of the display. Change the radix for all three arrays to decimal.

Note: Changing the radix for any element of an array changes the radix for all elements in that array.

FIGURE 4-4: ARRAYS SET TO DECIMAL



Watch

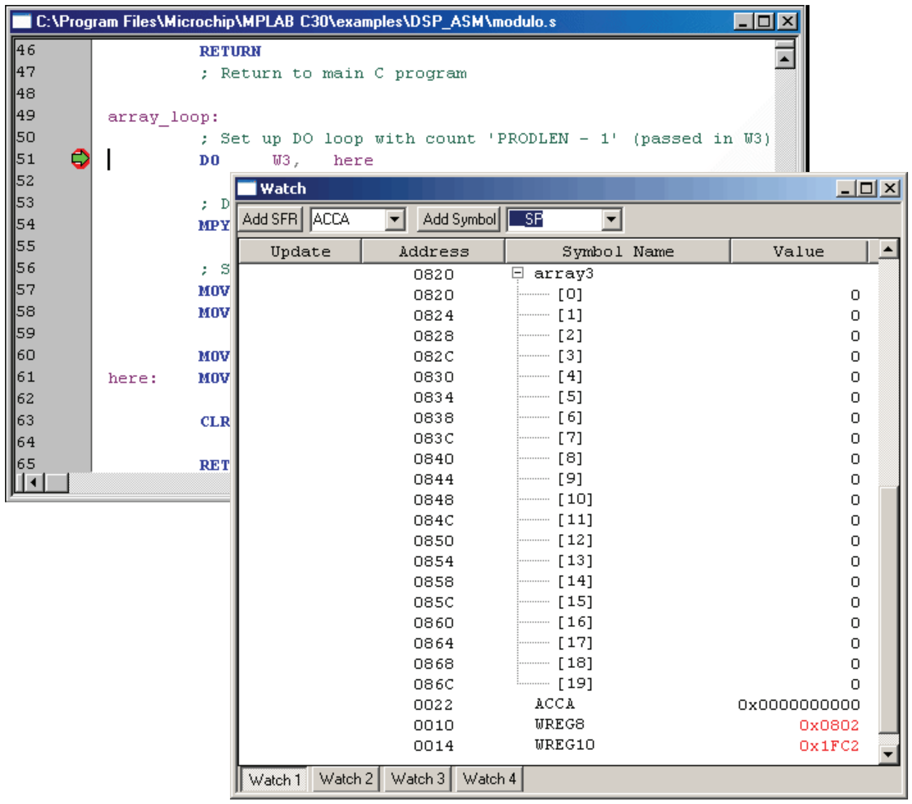
Add SFR: WREG10 | Add Symbol: array3

Update	Address	Symbol Name	Value
	0800	array1	
	0800	[0]	1
	0802	[1]	2
	0804	[2]	3
	0806	[3]	4
	0808	[4]	5
	080A	[5]	6
	080C	[6]	7
	080E	[7]	8
	0810	[8]	9
	0812	[9]	10
	1FC0	array2	
	1FC0	[0]	3
	1FC2	[1]	6
	1FC4	[2]	9
	1FC6	[3]	12
	1FC8	[4]	15
	1FCA	[5]	18
	1FCC	[6]	21
	1FCE	[7]	24
	1FD0	[8]	27
	1FD2	[9]	30
	0820	array3	
	0820	[0]	0
	0824	[1]	0
	0828	[2]	0
	082C	[3]	0

Watch 1 | Watch 2 | Watch 3 | Watch 4

Set a breakpoint in the `modulo.s` file at the start of the `DO` loop. Run to the breakpoint and scroll the Watch window to look at `array3`. It should still be all zeroes.

FIGURE 4-5: BREAKPOINT IN ASSEMBLY CODE FILE



C:\Program Files\Microchip\MPLAB C30\examples\DSP_ASM\modulo.s

```
        RETURN
        ; Return to main C program

array_loop:
        ; Set up DO loop with count 'PRODLEN - 1' (passed in W3)
        DO      W3,    here

        ; D
        MPY

        ; S
        MOV
        MOV

        MOV
here:   MOV

        CLR

        RET
```

Watch

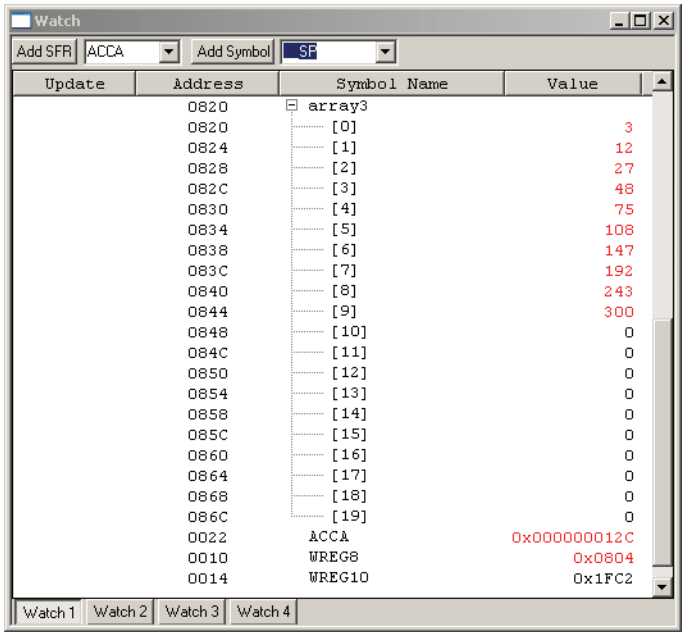
Add SFR: ACCA | Add Symbol: SP

Update	Address	Symbol Name	Value
	0820	array3	
	0820	[0]	0
	0824	[1]	0
	0828	[2]	0
	082C	[3]	0
	0830	[4]	0
	0834	[5]	0
	0838	[6]	0
	083C	[7]	0
	0840	[8]	0
	0844	[9]	0
	0848	[10]	0
	084C	[11]	0
	0850	[12]	0
	0854	[13]	0
	0858	[14]	0
	085C	[15]	0
	0860	[16]	0
	0864	[17]	0
	0868	[18]	0
	086C	[19]	0
	0022	ACCA	0x0000000000
	0010	WREG8	0x0802
	0014	WREG10	0x1FC2

Watch 1 | Watch 2 | Watch 3 | Watch 4

Press Run again, to run once through the `DO` loop. Now `array3` should show values representing the first set of products of each element pair from the source arrays:

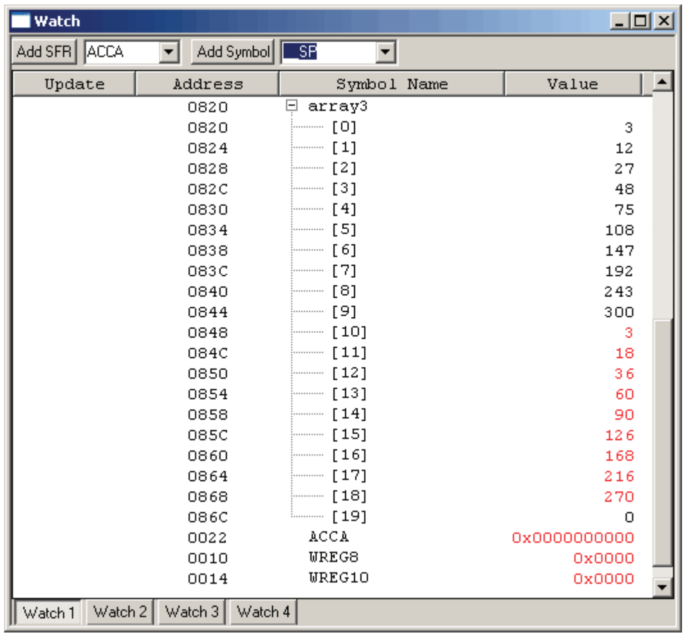
FIGURE 4-6: `array3` RESULTS – 1ST PASS



```
Update   Address   Symbol Name        Value
         0820      array3
         0820        [0]                  3
         0824        [1]                 12
         0828        [2]                 27
         082C        [3]                 48
         0830        [4]                 75
         0834        [5]                108
         0838        [6]                147
         083C        [7]                192
         0840        [8]                243
         0844        [9]                300
         0848        [10]                 0
         084C        [11]                 0
         0850        [12]                 0
         0854        [13]                 0
         0858        [14]                 0
         085C        [15]                 0
         0860        [16]                 0
         0864        [17]                 0
         0868        [18]                 0
         086C        [19]                 0
         0022      ACCA        0x000000012C
         0010      WREG8             0x0804
         0014      WREG10            0x1FC2
```

Run again (run to example3 breakpoint, run to modulo breakpoint once, run to modulo breakpoint again) to see the results for the second pass through the `DO` loop:

FIGURE 4-7: `array3` RESULTS – 2ND PASS



```
Update   Address   Symbol Name        Value
         0820      array3
         0820        [0]                  3
         0824        [1]                 12
         0828        [2]                 27
         082C        [3]                 48
         0830        [4]                 75
         0834        [5]                108
         0838        [6]                147
         083C        [7]                192
         0840        [8]                243
         0844        [9]                300
         0848        [10]                 3
         084C        [11]                18
         0850        [12]                36
         0854        [13]                60
         0858        [14]                90
         085C        [15]               126
         0860        [16]               168
         0864        [17]               216
         0868        [18]               270
         086C        [19]                 0
         0022      ACCA        0x0000000000
         0010      WREG8             0x0000
         0014      WREG10            0x0000
```

Remove the breakpoint from `modulo.s` and press Run to see the next time through the loop. Press Run a few more times to see the values change with subsequent executions of this multiplication process. Finally, remove the breakpoint from `example3.c`.

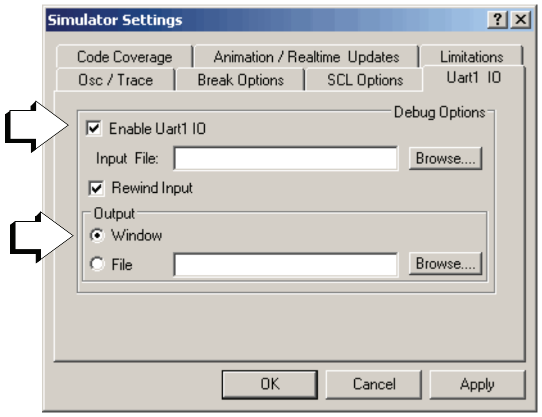
With Watch windows, data can be examined as code is run and halted with breakpoints. The simulator can also output data as it executes, providing a log that can be inspected and sent to other tools for graphing and analysis. Insert a `printf()` statement after the `modulo()` function call to monitor the values in the output array. The code should look like this (added code is bold):

EXAMPLE 4-3: `printf()` MONITOR

```
modulo( array1, array2, array3, PRODLEN-1 );

printf("Product Array\n");

  for (array_index=0; array_index<PRODLEN/2; array_index++)
     printf("%ld\n",array3[array_index]);
```

The `printf()` function uses the UART1 functions of the dsPIC DSC being simulated to write messages either to a file or to the Output window. Select *Debugger>Settings* to bring up the simulator Settings dialog. Click the **UART1 IO** tab, check “Enable UART I/O”, and then select the radio button to send text from the `printf()` statement to the Output window. Click **OK**.

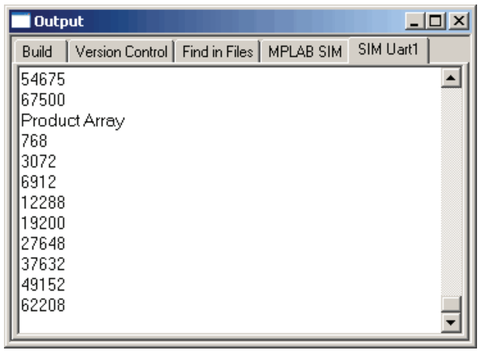
FIGURE 4-8: UART1 I/O – `printf()` SETUP



Enable Uart1 IO [✓]
Rewind Input [✓]
Output: [✓ Window] File

Now recompile your project (*Project>Build All*). Press Run, let it run for a few seconds, then press Halt. If the Output window is not present, enable it on *View>Output*. Click the **SIM UART1** tab. A log of the contents of `array3` should have been generated in the Output window.

FIGURE 4-9: `printf()` OUTPUT



```
54675
67500
Product Array
768
3072
6912
12288
19200
27648
37632
49152
62208
```

4.5 EXPLORING FURTHER

Go ahead and experiment with this example program. Things to explore include:

- Some of the other DSP instructions can be tried to further process the numbers in these arrays.
- Use the `printf()` function to output lists of values that can then be imported into a spreadsheet. Graph the values.
- Use the DMCI (*Tools>DMCI - Data Monitor Control Interface*) to graph values. Click the **Dynamic Data View** tab and then click the checkbox next to "Graph 1" to enable it. Right click in the graph area and select "Configure Data Source" from the menu. In the dialog, click on one of the arrays under "Data Sources", "Global Symbols", "Dynamic", to select that array and then click **OK**. Finally click the **Run** button on the DMCI window to see the graph.
- Further generalize the code so that all of the modulo indexing is set up from within `modulo.s` (i.e., convert these lines from Example 4-1 into assembly code that sets up the modulo addressing parameters from the parameters passed into the array).

```
XMODSRT = (unsigned int)array1;
XMODEND = (unsigned int)array1 + PRODLEN - 1;
YMODSRT = (unsigned int)array2;
YMODEND = (unsigned int)array2 + PRODLEN - 1;
```

4.6 WHERE TO GO FROM HERE

These tutorials were designed to help you gain familiarity with using the 16-bit compiler in the MPLAB IDE environment. There are many features of MPLAB IDE and the 16-bit compiler that were not covered here. For more information, reference the current MPLAB IDE on-line help, "*MPLAB® C Compiler for PIC24 MCUs and dsPIC® DSCs User's Guide"* (DS51284) and *"MPLAB® Assembler, Linker and Utilities for PIC24 MCUs and dsPIC® DSCs User's Guide"* (DS51317) to start using these tools for individual applications.

Instant help can be obtained from MPLAB IDE's on-line help or by logging on to Microchip's web conference for MPLAB C compiler products at www.microchip.com. Go to the Technical Support section and then to the On-Line Discussion Groups. The Development Systems web board also has a section devoted to Microchip C compiler discussion.

By subscribing to the Customer Change Notification service on Microchip's web site, customers can register to be notified of changes to the 16-bit C compiler. Choose the MPLAB C compiler category in Development Tools to receive notices when new versions are available and to receive timely information on the 16-bit compiler.

NOTES:

Index

A

Add Files to Project 11
Arrays 49

B

Breakpoints 20, 21, 49
Build Errors 18
Build Options 13
Build Project 18

C

Clock Reset 34
Creating a Project 9
Customer Notification Service 4
Customer Support 5

D

Disassembly Window 24
DMCI 53
Documentation
  Conventions 2
  Layout 1

G

Graphs 53

I

Installing the 16-Bit Assembler and Linker 7
Installing the 16-Bit C Compiler 7
Internet Address, Microchip 4
Interrupt Period 43
Interrupt Service Routine 35

L

Language Tool Setup 10
Listing Files 23

M

Map Files 23
Mixed C and Assembly Files 45
modulo() 49
MPLAB SIM Simulator 20, 39, 49

N

New Project 10

O

Oscillator Frequency, Stimulus 39
Output Window 18

P

printf() 48
printf() Output 52
Processor Selection 10
Program Memory Window 25
Project Window 12, 48
Project Wizard 10, 48

R

Radix, Set 42
Reading, Recommended 3
Real-Time Interrupt Using a Template File 27
References 53

S

Stopwatch 39
Structures 40

T

Template Files 27

U

UART1 I/O 52

W

Watch Window 21, 22, 49
Web Site, Microchip 4

MICROCHIP

WORLDWIDE SALES AND SERVICE

AMERICAS

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

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

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

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

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

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

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

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

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

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

ASIA/PACIFIC

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

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

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

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

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

China - Nanjing
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

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

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

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

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

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

China - Xiamen
Tel: 86-592-2388138
Fax: 86-592-2388130

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

China - Zhuhai
Tel: 86-756-3210040
Fax: 86-756-3210049

ASIA/PACIFIC

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

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

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

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

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

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

Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068

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

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

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

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

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

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

EUROPE

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

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

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

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

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

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

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

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

01/02/08