



**dsPIC<sup>®</sup>**  
语言工具  
入门

---

**请注意以下有关 Microchip 器件代码保护功能的要点:**

- Microchip 的产品均达到 Microchip 数据手册中所述的技术指标。
- Microchip 确信: 在正常使用的情况下, Microchip 系列产品是当今市场上同类产品中最安全的产品之一。
- 目前, 仍存在着恶意、甚至是非法破坏代码保护功能的行为。就我们所知, 所有这些行为都不是以 Microchip 数据手册中规定的操作规范来使用 Microchip 产品的。这样做的人极可能侵犯了知识产权。
- Microchip 愿与那些注重代码完整性的客户合作。
- Microchip 或任何其他半导体厂商均无法保证其代码的安全性。代码保护并不意味着我们保证产品是“牢不可破”的。

代码保护功能处于持续发展中。Microchip 承诺将不断改进产品的代码保护功能。任何试图破坏 Microchip 代码保护功能的行为均可视为违反了《数字器件千年版权法案 (Digital Millennium Copyright Act)》。如果这种行为导致他人在未经授权的情况下, 能访问您的软件或其他受版权保护的成果, 您有权依据该法案提起诉讼, 从而制止这种行为。

---

提供本文档的中文版本仅为了便于理解。Microchip Technology Inc. 及其分公司和相关公司、各级主管与员工及事务代理机构对译文中可能存在的任何差错不承担任何责任。建议参考 Microchip Technology Inc. 的英文原版文档。

本出版物中所述的器件应用信息及其他类似内容仅为您提供便利, 它们可能由更新之信息所替代。确保应用符合技术规范, 是您自身应负的责任。Microchip 对这些信息不作任何明示或暗示、书面或口头、法定或其他形式的声明或担保, 包括但不限于针对其使用情况、质量、性能、适销性或特定用途的适用性的声明或担保。Microchip 对因这些信息及使用这些信息而引起的后果不承担任何责任。未经 Microchip 书面批准, 不得将 Microchip 的产品用作生命维持系统中的关键组件。在 Microchip 知识产权保护下, 不得暗或以其他方式转让任何许可证。

#### 商标

Microchip 的名称和徽标组合、Microchip 徽标、Accuron、dsPIC、KEELOQ、microID、MPLAB、PIC、PICmicro、PICSTART、PRO MATE、PowerSmart、rfPIC 和 SmartShunt 均为 Microchip Technology Inc. 在美国和其他国家或地区的注册商标。

AmpLab、FilterLab、Migratable Memory、MXDEV、MXLAB、PICMASTER、SEEVAL、SmartSensor 和 The Embedded Control Solutions Company 均为 Microchip Technology Inc. 在美国的注册商标。

Analog-for-the-Digital Age、Application Maestro、dsPICDEM、dsPICDEM.net、dsPICworks、ECAN、ECONOMONITOR、FanSense、FlexROM、fuzzyLAB、In-Circuit Serial Programming、ICSP、ICEPIC、Linear Active Thermistor、MPASM、MPLIB、MPLINK、MPSIM、PICKIT、PICDEM、PICDEM.net、PICLAB、PICKIT、PowerCal、PowerInfo、PowerMate、PowerTool、rfLAB、rfPICDEM、Select Mode、Smart Serial、SmartTel、Total Endurance 和 WiperLock 均为 Microchip Technology Inc. 在美国和其他国家或地区的商标。

SQTP 是 Microchip Technology Inc. 在美国的服务标记。

在此提及的所有其他商标均为各持有公司所有。

© 2005, Microchip Technology Inc. 版权所有。

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

Microchip 位于美国亚利桑那州 Chandler 和 Tempe 及位于加利福尼亚州 Mountain View 的全球总部、设计中心和晶圆生产厂均于 2003 年 10 月通过了 ISO/TS-16949:2002 质量体系认证。公司在 PICmicro® 8 位单片机、KEELOQ® 跳码器件、串行 EEPROM、单片机外设、非易失性存储器 and 模拟产品方面的质量体系流程均符合 ISO/TS-16949:2002。此外, Microchip 在开发系统的设计和生产方面的质量体系也已通过了 ISO 9001:2000 认证。

---

---

**目录**

---

---

前言 .....	1
<b>第 1 章 安装与概述</b>	
1.1 简介 .....	7
1.2 安装 MPLAB ASM30、MPLAB LINK30 和语言工具实用程序 .....	7
1.3 安装 MPLAB C30 .....	7
1.4 卸载 MPLAB C30 .....	7
1.5 教程概述 .....	8
<b>第 2 章 教程 1 — 创建项目</b>	
2.1 简介 .....	9
2.2 创建文件 .....	9
2.3 使用项目向导 .....	10
2.4 使用项目窗口 .....	13
2.5 设置编译选项 .....	14
2.6 编译项目 .....	18
2.7 编译错误疑难解答 .....	18
2.8 使用 MPLAB SIM 软件模拟器进行调试 .....	20
2.9 生成映射文件 .....	23
2.10 汇编代码的调试 .....	24
2.11 深入学习 .....	26
<b>第 3 章 教程 2 — 实时中断</b>	
3.1 简介 .....	27
3.2 使用模板文件 .....	27
3.3 在新项目中使用模板 .....	31
3.4 使用 MPLAB SIM 软件模拟器进行调试 .....	37
3.5 深入学习 .....	41
<b>第 4 章 教程 3 — 混合使用 C 文件与汇编文件</b>	
4.1 简介 .....	43
4.2 获得项目源文件 .....	43
4.3 创建和编译项目 .....	45
4.4 检查程序 .....	46
4.5 深入学习 .....	50
4.6 今后如何使用 .....	50
索引 .....	51
全球销售及服务网点 .....	54

注:

---

---

## 前言

---

---

### 客户须知

所有文档均会更新，本文档也不例外。Microchip 的工具和文档将不断演变以满足客户的需求，因此实际使用中有些对话框和 / 或工具说明可能与本文档所述之内容有所不同。请访问我们的网站 ([www.microchip.com](http://www.microchip.com)) 获取最新文档。

文档均标记有“DS”编号。该编号出现在每页底部的页码之前。DS 编号的命名约定为“DSXXXXA”，其中“XXXX”为文档编号，“A”为文档版本。

欲了解开发工具的最新信息，请参考 MPLAB<sup>®</sup> IDE 在线帮助。从 Help（帮助）菜单选择 Topics（主题），打开现有在线帮助文件列表。

### 简介

本文包含了在使用 dsPIC 语言工具之前需要了解的基本信息，这些信息非常有用。本文讨论的内容包括：

- 关于本指南
- 推荐读物
- Microchip 网站
- 开发系统变更通知客户服务
- 客户支持

### 关于本指南

#### 文档编排

这篇文档介绍了如何利用 dsPIC<sup>®</sup> 语言工具作为开发工具在目标电路板仿真和调试固件。手册的内容安排如下：

- **第 1 章：安装和概述** — 如何在 PC 机上安装 dsPIC 语言工具以及它们是如何工作的。
- **第 2 章：教程 1 — 创建项目** — 如何利用 dsPIC 工具创建项目。
- **第 3 章：教程 2 — 实时中断** — 如何创建使用实时中断的 dsPIC 应用程序。
- **第 4 章：教程 3 — 混合 C 和汇编文件** — 如何创建混合使用 C 和汇编代码文件的 dsPIC 应用程序。

## 本指南使用的约定

本手册使用以下文档约定：

### 文档约定

说明	表示	示例
<b>Arial 字体：</b>		
斜体字符	参考书	<i>MPLAB<sup>®</sup> IDE User's Guide</i>
	需强调的文字	...is the <i>only</i> compiler...
首字母大写	窗口	the Output window
	对话框	the Settings dialog
	菜单选项	select Enable Programmer
引号	窗口或对话框中的字段名	"Save project before build"
下划线、带右尖括号的斜体文本	菜单路径	<i>File&gt;Save</i>
粗体字符	对话框按钮	Click <b>OK</b>
	选项卡	Click the <b>Power</b> tab
'bnnnn'	二进制数， <i>n</i> 为其中一位	'b00100, 'b10
尖括号 <> 中的文本	键盘上的按键	Press <Enter>, <F1>
<b>Courier 字体：</b>		
普通 Courier	源代码示例	#define START
	文件名	autoexec.bat
	文件路径	c:\mcc18\h
	关键字	_asm, _endasm, static
	命令行选项	-Opa+, -Opa-
	位值	0, 1
斜体 Courier	变量参数	<i>file.o</i> , where <i>file</i> can be any valid filename
0xn timer	16 进制数， <i>n</i> 是一个 16 进制位	0xFFFF, 0x007A
中括号 []	可选参数	mcc18 [options] <i>file</i> [options]
大括号和竖线 {}	可选的互斥参数；“或”逻辑	errorlevel {0 1}
省略号 ...	替换重复的文本	var_name [, var_name...]
	代表用户提供的代码	void main (void) { ... }

## 推荐读物

本用户指南介绍如何使用 dsPIC 语言工具。下面列出了其他有帮助的读物。以下 microchip 文档均已提供，推荐作为辅助的参考资料。

### **README 文件**

关于 Microchip 工具的最新信息，请阅读软件附带的 README 文件（ASCII 文本文件）。

### **MPLAB<sup>®</sup> ASM30, MPLAB LINK30 and Utilities User's Guide (DS51317)**

指导使用 dsPIC DSC 汇编器、MPLAB ASM30、dsPIC DSC 链接器、MPLAB LINK30 和各种 dsPIC DSC 实用程序，包括 MPLAB LIB30 存档程序 / 库管理程序。

### **MPLAB<sup>®</sup> C30 C 编译器用户指南 (DS51284C\_CN)**

dsPIC DSC C 编译器指南。MPLAB LINK30 与这个工具配合使用。

### **dsPIC<sup>®</sup> Language Tools Libraries (DS51456)**

DSP、dsPIC 外设和标准（包括数学）库，以及 MPLAB C30 内嵌函数，与 dsPIC 语言工具一起使用。

### **GNU HTML 文档**

在语言工具的光盘中有提供这个文件。它介绍了标准 GNU 开发工具，MPLAB C30 就是以此为基础的。

### **dsPIC30F Data Sheet General Purpose and Sensor Families (DS70083)**

这是 dsPIC30F 数字信号控制器（DSC）的数据手册。总体介绍了此系列芯片及其架构。详细介绍了存储器的构成、DSP 操作和外围功能。其中包括芯片的电气参数。

### **dsPIC30F 系列参考手册 (DS70046C\_CN)**

该系列芯片的参考指南。介绍了 dsPIC30F MCU 系列的架构和外围模块。

### **dsPIC30F Programmer's Reference Manual (DS70030)**

dsPIC30F 芯片编程器的参考指南。包括编程模型和指令集。

### **C 标准信息**

American National Standard for Information Systems – *Programming Language – C*.  
American National Standards Institute (ANSI), 11 West 42nd. Street, New York,  
New York, 10036.

此标准规定了用 C 语言编写程序的格式，并对 C 程序进行了解释。其目的是提高 C 程序在多种计算机系统上的可移植性、可靠性、可维护性及执行效率。

## C 语言参考书籍

Harbison, Samuel P., and Steele, Guy L., *C A Reference Manual*, Fourth Edition, Prentice-Hall, Englewood Cliffs, N.J. 07632.

Kernighan, Brian W., and Ritchie, Dennis M., *The C Programming Language*, Second Edition. Prentice Hall, Englewood Cliffs, N.J. 07632.

Kochan, Steven G., *Programming In ANSI C*, Revised Edition. Hayden Books, Indianapolis, Indiana 46268.

Plauger, P.J., *The Standard C Library*, Prentice-Hall, Englewood Cliffs, N.J. 07632.

Van Sickle, Ted., *Programming Microcontrollers in C*, First Edition. LLH Technology Publishing, Eagle Rock, Virginia 24085.

## MICROCHIP 网站

Microchip 在全球网站 [www.microchip.com](http://www.microchip.com) 上提供在线支持。用户可以在网站上很方便地获得文件和信息。用户可以使用自己喜欢的互联网浏览器访问网站。该网站包含以下信息：

- **产品支持** — 数据手册和勘误表、应用笔记和示例程序、设计资源、用户指南和硬件支持文档、最新的软件版本和归档软件
- **一般技术支持** — 常见问题（FAQ）解答、技术支持请求、在线讨论组以及 Microchip 顾问计划成员名单
- **Microchip 业务** — 产品选型和订购指南、最新的 Microchip 新闻、研讨会与活动安排表、Microchip 销售办事处、代理商及工厂代表列表

## 开发系统变更通知客户服务

Microchip 的客户通知服务帮助客户了解关于 Microchip 产品的最新信息。只要您指定的产品系列或您感兴趣的开发工具出现变动、更新、修订或勘误，您都将收到我们的电子邮件通知。

注册客户可在他们感兴趣的某个产品系列或开发工具发生变更、更新、发布新版本或勘误表时，收到电子邮件通知。

欲注册，请登录 Microchip 网站 [www.microchip.com](http://www.microchip.com)，点击“变更通知客户（Customer Change Notification）”服务并按照注册说明完成注册。

开发系统产品的分类如下：

- **编译器**—关于 Microchip C 编译器和其他语言工具的最新信息。包括 MPLAB C17、MPLAB C18、MPLAB C30 C 编译器；MPASM™ 和 MPLAB ASM30 汇编器；MPLINK™ 和 MPLAB LINK30 目标链接器；以及 MPLIB™ 和 MPLAB LIB30 目标库管理程序。
- **仿真器**—Microchip 在线仿真器的最新信息。包括 MPLAB ICE2000 和 MPLAB ICE4000。
- **在线调试器**—Microchip 在线调试器 MPLAB ICD 2 的最新信息。
- **MPLAB IDE**—Microchip MPLAB IDE，即开发系统工具的 Windows<sup>®</sup> 集成开发环境的最新信息。主要集中在 MPLAB IDE、MPLAB SIM 软件模拟器、MPLAB IDE 项目管理器以及一般的编辑和调试功能。
- **编程器**—Microchip 芯片编程器的最新信息。包括 MPLAB PM3 和 PRO MATE<sup>®</sup> II 芯片编程器和 PICSTART<sup>®</sup> Plus 开发编程器。



## 客户支持

Microchip 产品的用户可通过以下渠道获得帮助：

- 代理商或代表
- 当地销售办事处
- 应用工程师（FAE）
- 技术支持
- 开发系统信息热线

客户应联系其代理商、代表或应用工程师（FAE）寻求支持。当地销售办事处也可为客户提供帮助。本文档后附有销售办事处的联系方式。

也可通过 <http://support.microchip.com> 获得网上技术支持，

此外，我们还设有一条开发系统信息热线，列出了 Microchip 开发系统软件产品的最新版本。此热线还向客户提供如何取得当前可用的升级软件包的信息。

开发系统信息热线电话为：

1-800-755-2345 – 美国和加拿大大部分地区

800-820-6247 – 中国大陆

1-480-792-7302 – 其他国家或地区

注:

---

---

## 第 1 章 安装与概述

---

---

### 1.1 简介

本文通过在 MPLAB 集成开发环境 (IDE) V6.30 或更新的版本中使用 MPLAB C30 的逐步教学法来帮助用户学会使用 dsPIC30F 软件工具。MPLAB IDE 必须先安装在 PC 机中。

在我们提供的光盘中有 MPLAB IDE 软件，也可以免费从 [www.microchip.com](http://www.microchip.com) 上下载。MPLAB IDE 包括项目管理器和 MPLAB SIM 软件模拟器，以及将在本指南中大量地使用到的内嵌调试器。

本章将讨论以下内容：

- 安装 MPLAB ASM30、MPLAB LINK30 和语言工具实用程序
- 安装 MPLAB C30
- 卸载 MPLAB C30
- 教程概述

### 1.2 安装 MPLAB ASM30、MPLAB LINK30 和语言工具实用程序

随 MPLAB IDE 免费提供 MPLAB ASM30 和 MPLAB LINK30。在 MPLAB C30 编译器的安装中也包含了 MPLAB ASM30 和 MPLAB LINK30。为了保证所有 dsPIC30F 工具之间的兼容性，应该使用随 MPLAB C30 编译器提供的这些工具的版本。

### 1.3 安装 MPLAB C30

- 当安装 MPLAB C30 编译器的升级版本时，可能会覆盖 PC 机上现有的文件。必须对可能被修改的文件进行备份。
- 将光盘插入计算机中并执行安装 MPLAB C30vx.xx (x.xx 是当前版本号) 文件。按照安装过程中的一系列对话框的提示进行安装。在安装过程中可能会花费几分钟的时间搜索计算机中的 MPLAB IDE 和其他相关的文件。
- 为使用本指南中的示例来学习，确保 EXAMPLES 前的复选框被选中。

### 1.4 卸载 MPLAB C30

打开编译器安装文件夹并双击 UNWISE.EXE 文件。

**注：** 当卸载 MPLAB C30 的升级版本时将删除整个安装。而上次安装后添加到目录中的文件不会被删除。

## 1.5 教程概述

下面的教程旨在帮助工程师熟悉 C 编程语言以及利用 MPLAB 集成开发环境 (IDE) 和 MPLAB C30 编译器开始开发嵌入式系统的概念。本文档介绍了如何创建和编译项目，如何利用 dsPIC30F 芯片的功能编写代码，以及如何校验和调试利用 MPLAB C30 编写的代码。

首先要安装 MPLAB C30 编译器和 MPLAB IDE v6.30 (或更新版本)。请查阅关于 dsPIC<sup>®</sup> 的文献，如 *dsPIC30F Data Sheet General Purpose and Sensor Families* (DS70083) 和 *dsPIC30F Programmer's Reference Manual* (DS70030)，以便获得关于具体处理器的信息，如特殊功能寄存器、指令集和中断逻辑。

这些章节中关于使用 MPLAB C30 编译器的教程包括：

- **第 2 章**介绍了如何：
  - 创建和编译项目
  - 在示例代码中运行、单步执行和设置断点
  - 调试代码。
- **第 3 章**介绍了如何：
  - 使用模板来创建源文件
  - 在 C 中使用实时中断
- **第 4 章**介绍了如何：
  - 在 MPLAB C30 编译器中使用汇编语言 DSP 子程序
  - 与汇编语言模块之间传递参数

---

---

## 第 2 章 教程 1 — 创建项目

---

---

### 2.1 简介

本教程中简单的源代码是为 MPLAB IDE v6.xx 项目所设计的。它将使用 MPLAB SIM 软件模拟器来模拟 dsPIC30F6014 芯片。本教程中假定目录 C:\pic30\_tools 为 MPLAB C30 编译器的安装目录。本教程包括：

- 创建文件
- 使用项目向导
- 使用项目窗口
- 设置编译选项
- 编译项目
- 编译错误疑难解答
- 使用 MPLAB SIM 软件模拟器进行调试
- 生成映射文件
- 汇编代码的调试
- 深入学习

### 2.2 创建文件

启动 MPLAB IDE v6.30（或更新的版本）并选择 **File>New** 打开一个新的空白的源文件。**例 2-1** 给出了要键入（或者如果在阅读电子文档，拷贝和粘贴到）新源文件窗口的源代码。

**例 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 和 PORTB 是 dsPIC30F6014 芯片的特殊功能寄存器。PORTB 是一组通用输入/输出引脚。TRISB 的位用来配置 PORTB 引脚为输入（1）或输出（0）。

使用 **File>Save As** 将文件另存在安装目录的 \examples 目录下（通常为 C:\pic30\_tools\examples），文件名为 MyFile.c。

## 2.3 使用项目向导

选择 **Project>Project Wizard** 来创建新项目。将出现一个欢迎页面。点击 **Next>** 继续。

1. 在 “Step One: Select a Device” 中，通过下拉菜单选择 dsPIC30F6014 芯片，点击 **Next>** 继续。
2. 在 “Step Two: Select a language toolsuite” 中，选择 “Microchip C30 Toolsuite” 作为 “Active Toolsuite”。然后点击工具包中（在 “Toolsuite Contents” 之下）的每个语言工具并检查或设置与其相关的可执行文件的路径（图 2-1）。

MPLAB ASM30 汇编器应指向 “LOCATION” 下的汇编程序可执行文件 `pic30-as.exe`。如果没有这个文件，应键入或浏览到可执行文件的位置，通常默认为：

```
C:\Program Files\MPLAB IDE\dsPIC_Tools\Bin\pic30-as.exe
```

MPLAB C30 编译器应指向 “LOCATION” 下的编译程序可执行文件 `pic30-gcc.exe`。如果没有这个文件，应键入或浏览到可执行文件的位置，通常默认为：

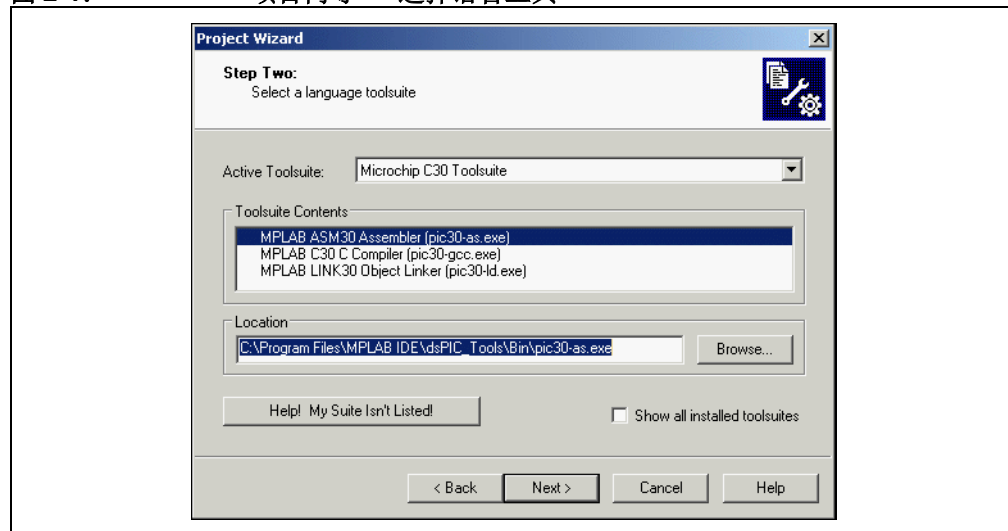
```
C:\pic30_tools\bin\pic30-gcc.exe
```

MPLAB LINK30 目标链接器应指向 “LOCATION” 下的链接程序可执行文件 `pic30-ld.exe`。如果没有这个文件，应键入或浏览到可执行文件的位置，通常默认为：

```
C:\Program Files\MPLAB IDE\dsPIC_Tools\Bin\pic30-ld.exe
```

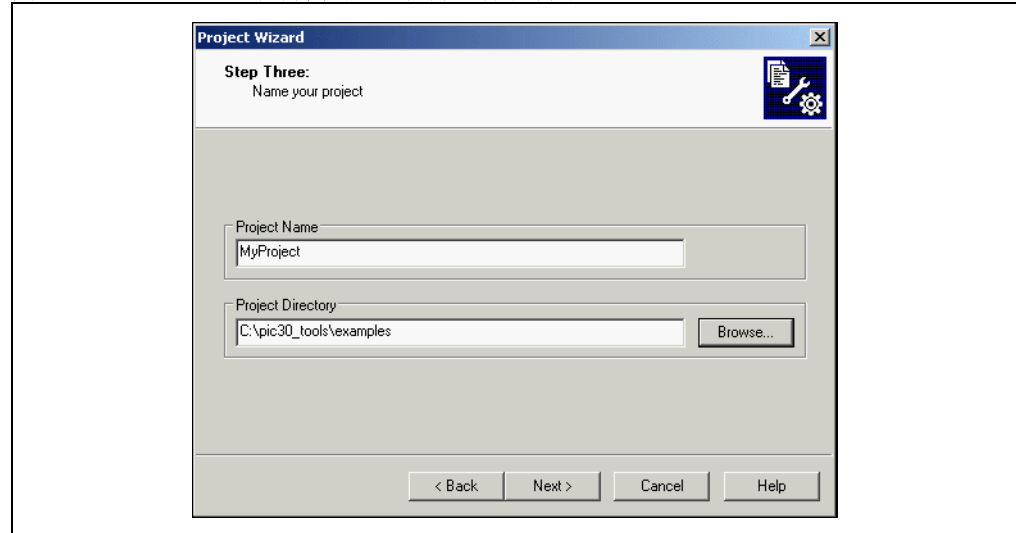
点击 **Next>** 继续。

图 2-1: 项目向导 — 选择语言工具



3. 在 “Step Three: Name your project” 中，键入项目名 `MyProject` 并点击 **BROWSE** 进入 MPLAB C30 安装目录下的 `\examples` 文件夹。然后点击 **NEXT >** 继续

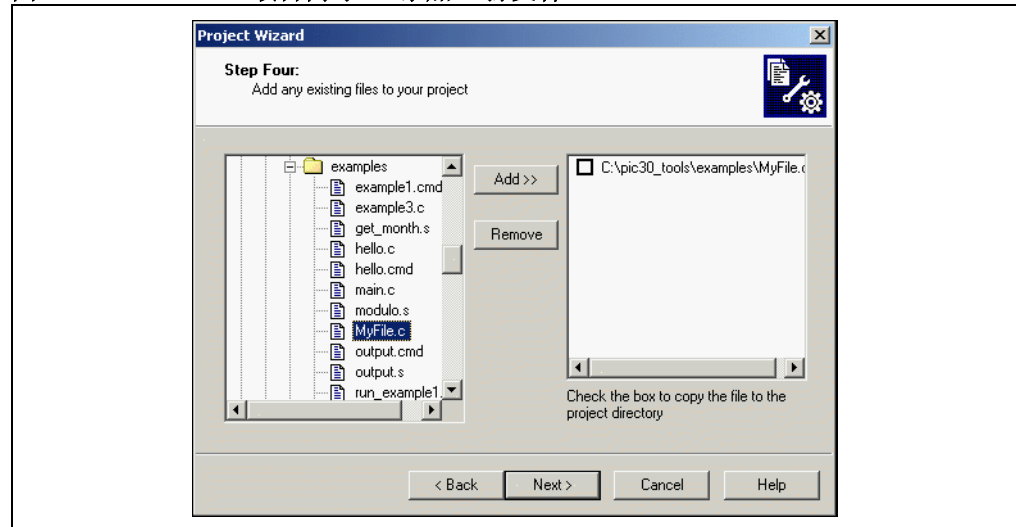
图 2-2: 项目向导 — 项目名称和目录



4. 在 “Step Four: Add any existing files to your project” 中，将添加两个文件到项目中。

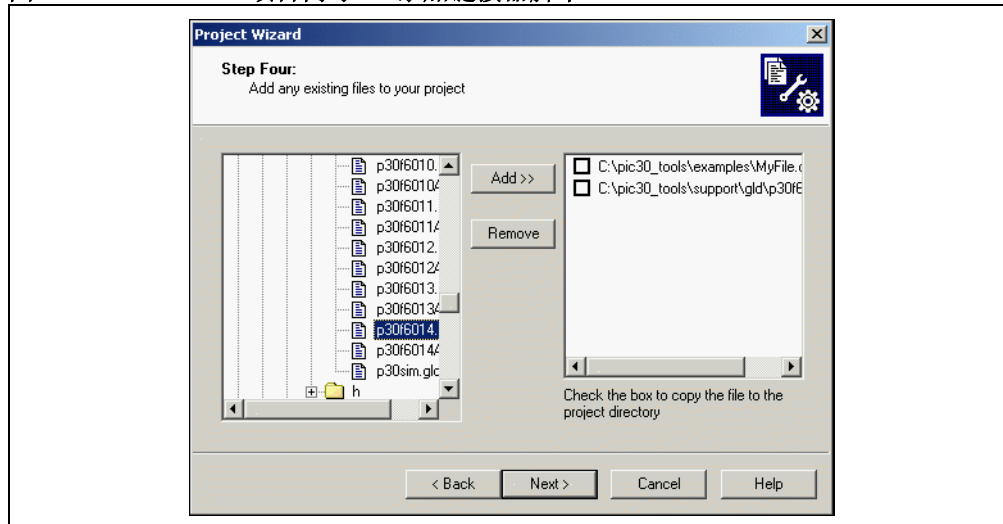
首先，选择先前在 `\examples` 文件夹中生成的源文件 `MyFile.c`。点击 **ADD>>** 将它添加到项目要使用的文件列表中（出现在右边）。

图 2-3: 项目向导 — 添加 C 源文件



其次，必须添加链接描述文件，告知链接器关于 dsPIC30F6014 的存储器构成。链接描述文件位于 MPLAB C30 安装目录下的 \support\gld 文件夹中。向下找到 p30f6014.gld 文件，选中它并点击 **ADD>>** 将它添加到项目中。

图 2-4: 项目向导 — 添加链接器脚本



点击 **Next>** 继续。

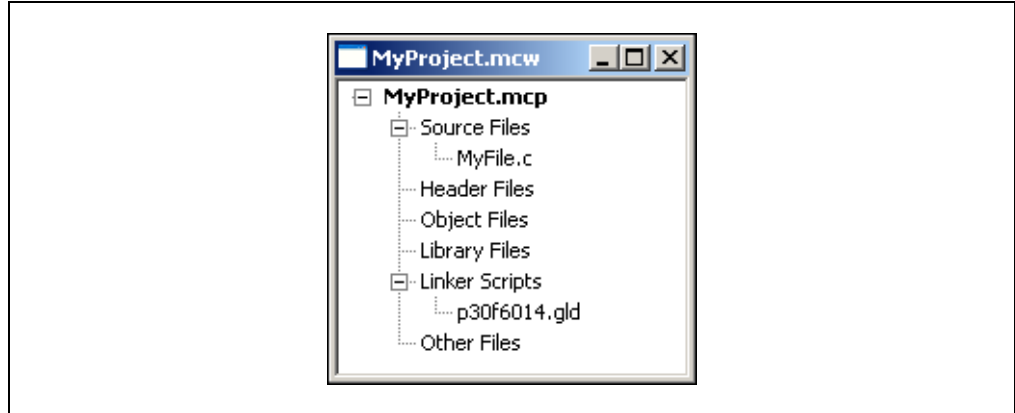
5. 在 Summary（摘要）窗口中重新检查“项目参数”，验证芯片、工具包和项目文件的位置是否正确。如果想修改某一项，可以点击 **Back** 返回上一个对话框。点击 **Finish** 生成新的项目和工作区。



## 2.4 使用项目窗口

项目窗口在 MPLAB IDE 的工作区内。工作区的文件名应出现在项目窗口顶部的标题栏中，MyProject.mcw，项目文件名 MyProject.mcp 作为项目的顶部“节点”。

图 2-5: 项目窗口



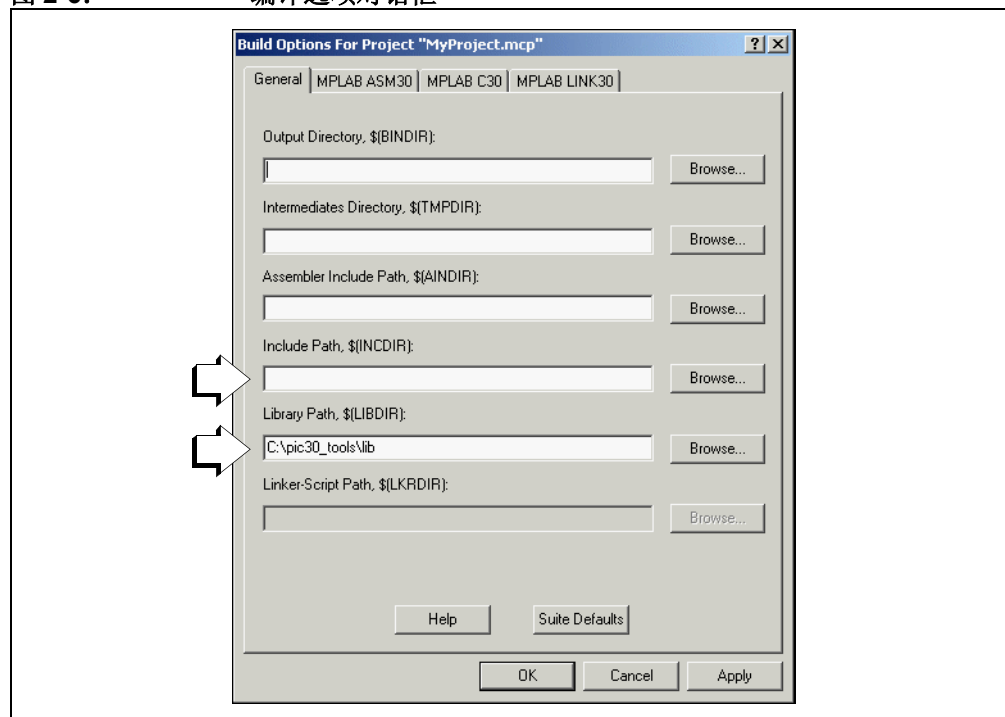
**注：** 如果发生错误，可选中一个文件名并按删除键或通过鼠标右键的菜单来删除。将光标移到“Source Files”或“Linker Scripts”上并通过鼠标右键来向项目添加适当的文件。

## 2.5 设置编译选项

现在，几乎已经可以用 dsPIC30F 工具来编译项目了。但是，需要检查项目和工具编译选项。

1. 选择 **Project>Build Options** 并点击 “Project” 显示整个项目的 Build Options（编译选项）对话框。
2. 选择 **General**（常规）选项卡。在本教程中，不需要为 “Include Path” 添加路径，但对于你自己将来的项目可能需要添加路径。“Library Path” 必须是 MPLAB C30 安装目录下的 \lib 目录。

图 2-6: 编译选项对话框

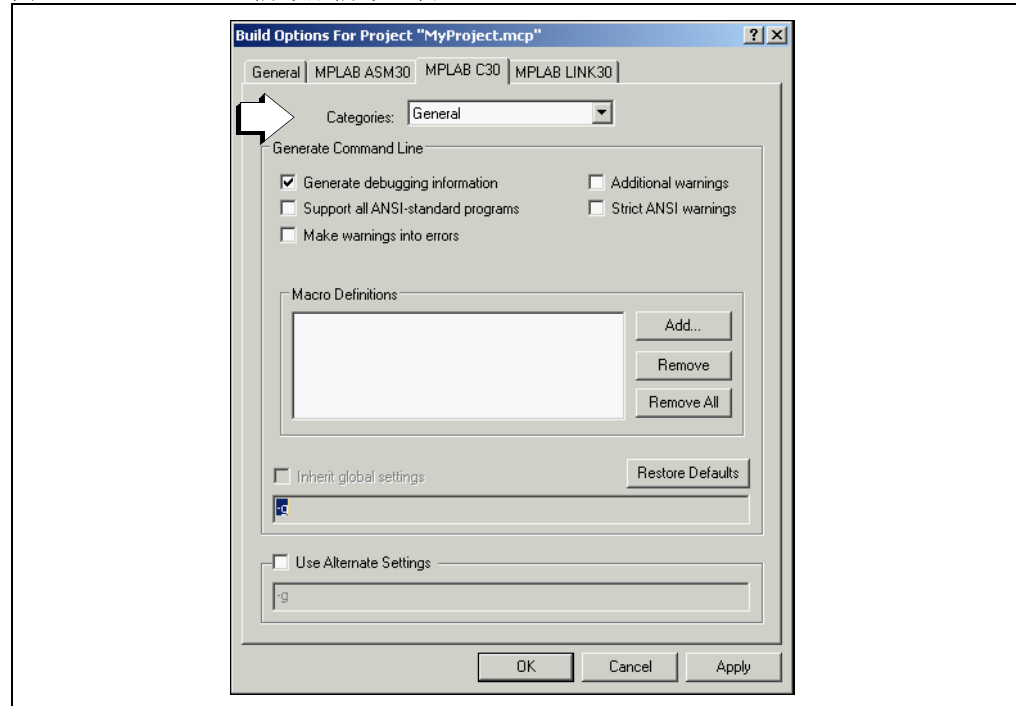


在特定工具的选项卡中可以对传递到 dsPIC 工具的命令行选项进行设置。

3. 点击 **MPLAB C30** 选项卡。MPLAB C30 有三个选项对话框：**General**、**Memory Model**（存储模型）和 **Optimizations**（优化）。这三个选项对话框可在“Categories”下拉菜单中选择，出现在对话框中的内容也将相应发生改变。

在这个例子中，将保持 MPLAB C30 默认的命令行选项不变。

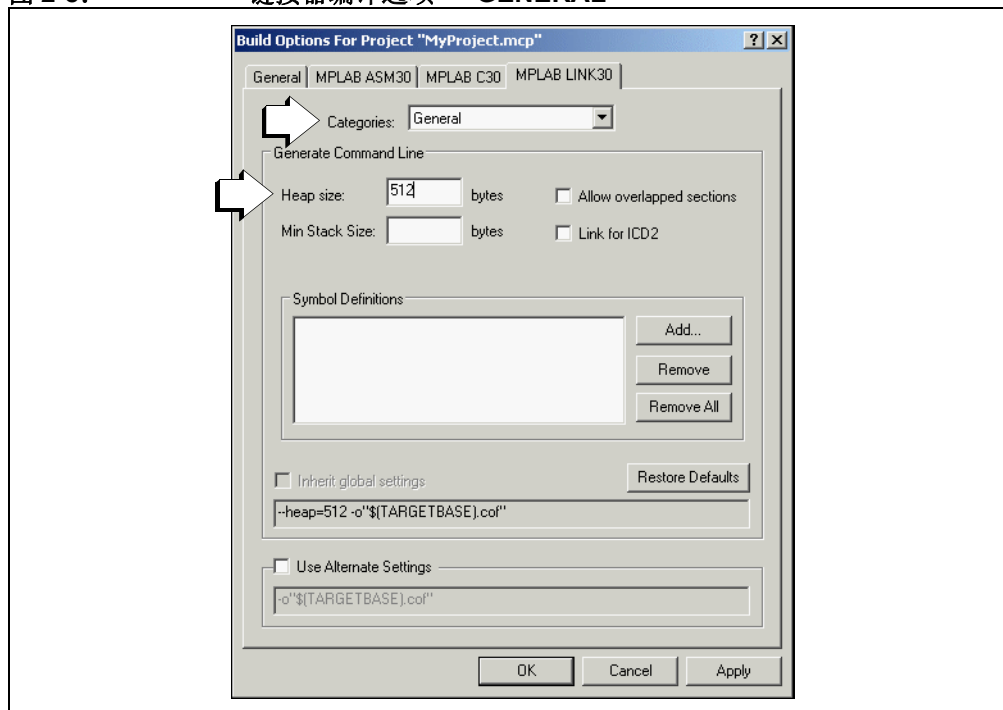
**图 2-7: 编译器编译选项 — GENERAL**



4. 选择 **MPLAB LINK30** 选项卡。MPLAB LINK30 有三个选项对话框：**General**、**Diagnostics**（诊断）和 **Symbols & Output**（符号和输出）。这三个选项对话框可在“**Categories**”下拉菜单中选择，出现在对话框中的内容也将相应发生改变。

为了运行本指南后面的教程 3，需要在 **General** 类中设置一个堆。堆大小设置为 512。

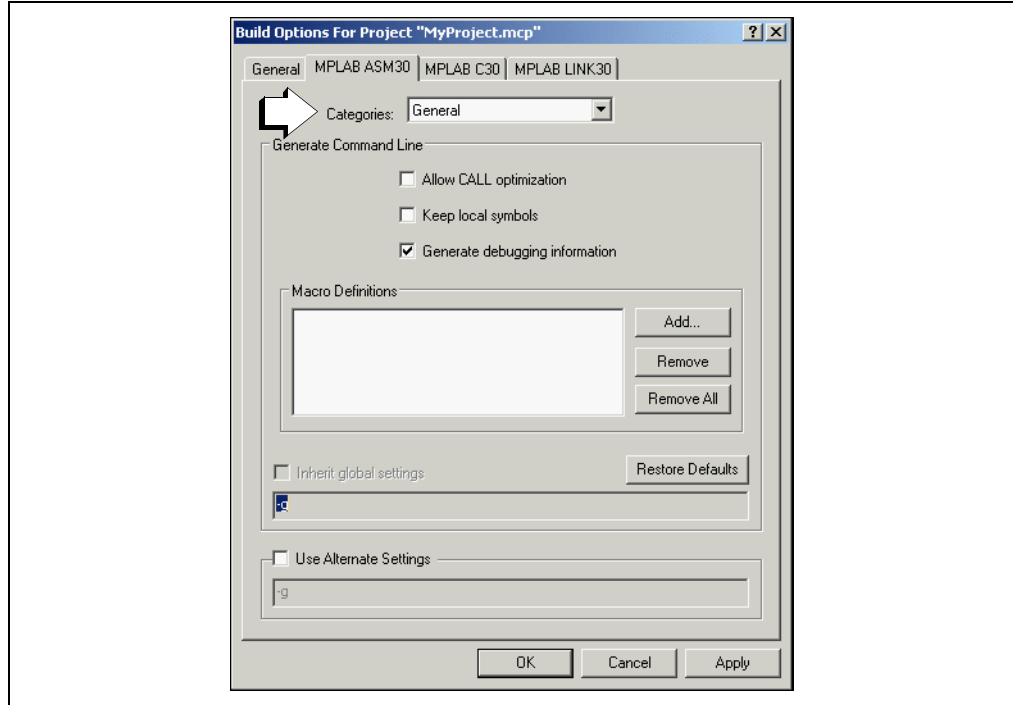
图 2-8: 链接器编译选项 — GENERAL



5. 选择 **MPLAB ASM30** 选项卡。MPLAB ASM30 有两个选项对话框：**General** 和 **Diagnostics**。这两个选项对话框可在 “Categories” 下拉菜单中选择，出现在对话框中的内容也将相应发生改变。

在这个例子中，将保持 MPLAB ASM30 默认的命令行选项不变。

**图 2-9: 汇编器编译选项 — GENERAL**

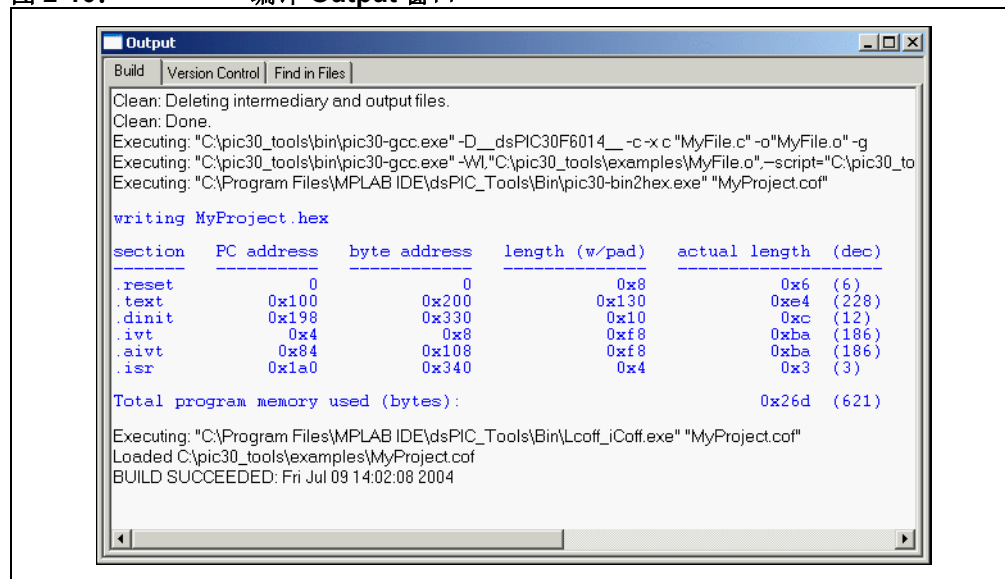


## 2.6 编译项目

选择 **Project>Build All** 对项目进行编译、汇编和链接。如果有任何错误或警告消息，会显示在输出窗口中。

对于本教程来说，**Output**（输出）窗口不应显示错误消息，而应显示表明项目“BUILD SUCCEEDED”（编译成功）的消息。如果有错误，应检查源文件的内容与例 2-1 中 myfile.c 文件的内容是否一致。

图 2-10: 编译 Output 窗口



## 2.7 编译错误疑难解答

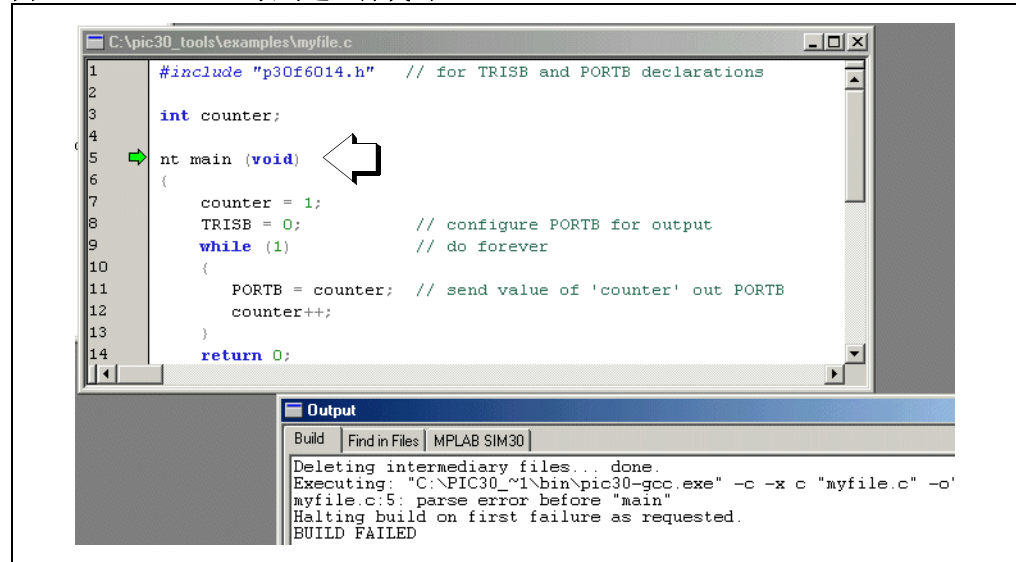
如果在项目编译后出现错误，可双击显示错误消息的行直接进入导致该错误的源代码行。如果您使用的是例子中的代码，那么最常见的错误就是拼写错误、漏掉了分号或大括号不匹配。在下面的屏幕中，出现了一个输入错误。在本例中，对 main() 进行“int”定义时不小心漏掉了字母“i”。这时将在 **Output** 窗口中出现错误消息。

图 2-11: 编译错误



在双击上图 Output 窗口的第三行后，就出现下图所示的窗口：

图 2-12: 双击进入源代码



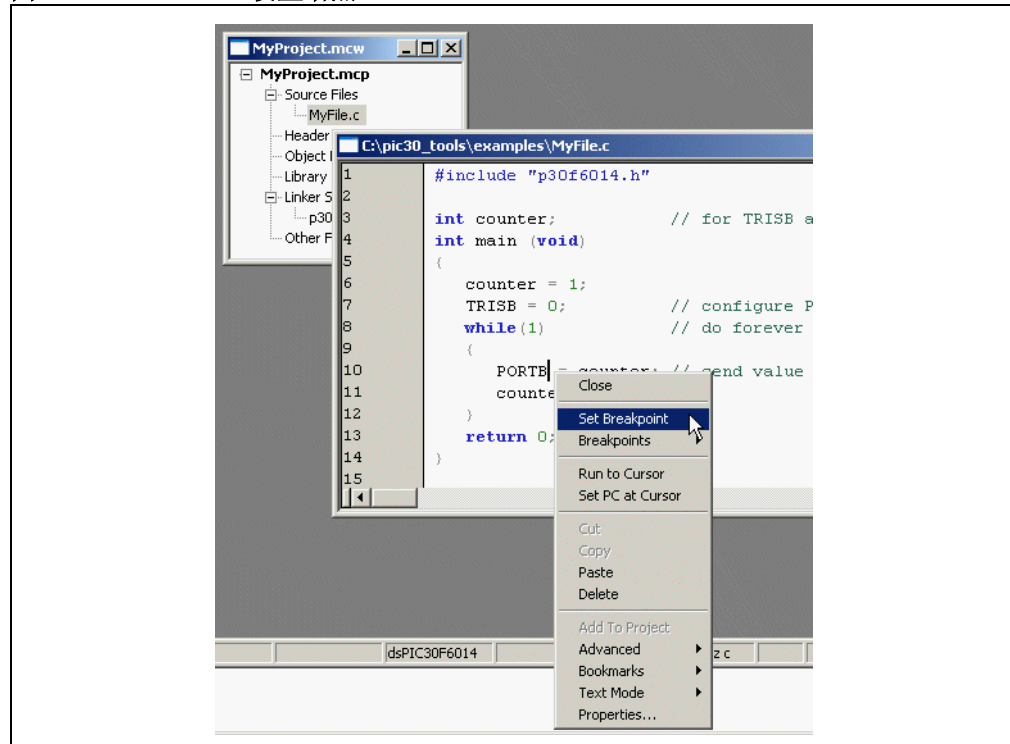
由于关键字都是用蓝色字体显示，而错误的输入“nt”以黑色文本显示，因此很容易识别出出现的错误。键入“i”将“nt”改为正确的关键字“int”，这时文本将变为蓝色。再次选择 **Project>Project Build All** 获得正确的编译结果。

## 2.8 使用 MPLAB SIM 软件模拟器进行调试

要调试应用代码，需要调试工具的帮助。在本教程中，我们使用 MPLAB SIM 软件模拟器。在这个模拟器中可以在源代码中设置断点，并可以在 Watch（观察）窗中对变量的值进行观察。

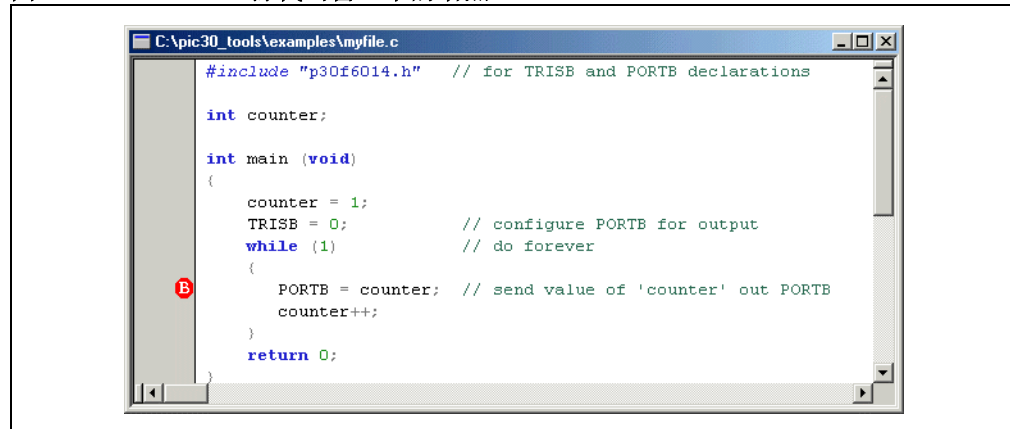
1. 通过选择 **Debugger>Select Tool>MPLAB SIM** 将 MPLAB SIM 软件模拟器作为调试工具。
2. 通过双击项目窗口的项目树中的文件名（MyFile.c）来打开源文件。在源文件中，将光标移动到下面的行上：  
`PORTB = counter;`  
然后通过鼠标右键选择“Set Breakpoint”（设置断点）。

图 2-13: 设置断点



在源代码窗口左边的空白处出现的红色符号表明断点已经设置并激活。

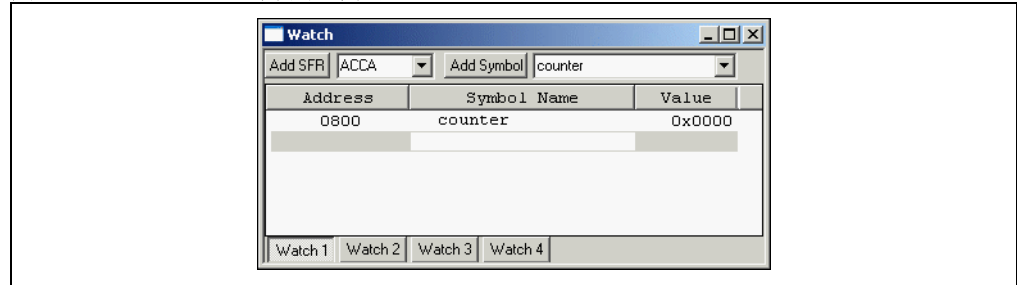
图 2-14: 源代码窗口中的断点





3. 选择 **View>Watch** 打开 Watch 窗口。从 **Add Symbol** 旁边的下拉扩展菜单中选择 **counter**，然后单击 **Add Symbol**。

图 2-15: 添加观察变量



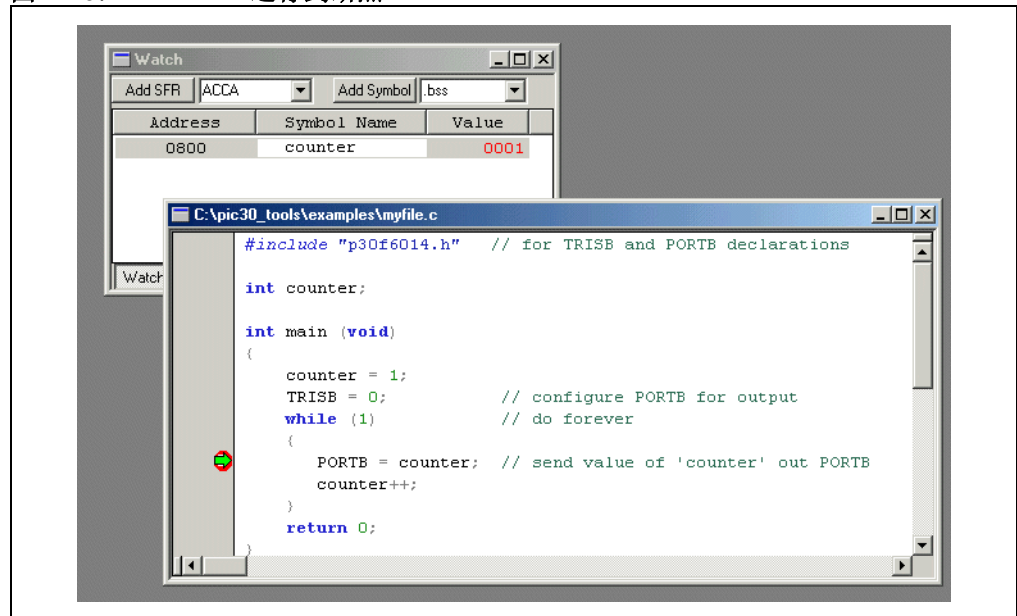
**注:** 有三种方法添加观察变量：（1）如上面所介绍的，从列表中选择；（2）直接在 Watch 窗口的符号名栏中键入变量名。（3）在源代码中选中变量并拖到 Watch 窗口中。

4. 单击工具栏中的 **RUN** 运行程序。



程序将在执行设置了断点的语句之前停下。源代码窗口左边空白处的绿色箭头指向下一个要执行的语句。Watch 窗口中显示此时的 counter 值为“1”。值“1”以红色字体显示，表明变量的值发生了变化。

图 2-16: 运行到断点



5. 点击 **RUN** 继续运行程序。程序将继续执行 while 循环直到再次停止在断点所在的行。Watch 窗口中显示此时的 counter 值为 “2”。
6. 要单步执行源代码，即每次执行一条语句，可以使用工具栏中的 **Step Into** 按钮。



每执行一个语句，源代码窗口左边空白处的绿色箭头都会指向下一个将执行的语句。

7. 将光标移动到设置了断点的行上，用鼠标右键选择 “Remove Breakpoint（删除断点）”。现在按 **Run** 按钮。状态栏的左下方将出现 “Running...” 消息，在它的旁边，一个移动条表明程序正在运行。Run 图标右边的 Step 图标将变成灰色。如果调试器菜单是下拉的，在列表中的 Step 选项也将灰掉。在运行模式下，这些操作都是禁止的。

要中断运行的程序，使用工具栏中的 **Halt** 按钮。



一旦程序运行停止，Step 图标将不再是灰掉的。

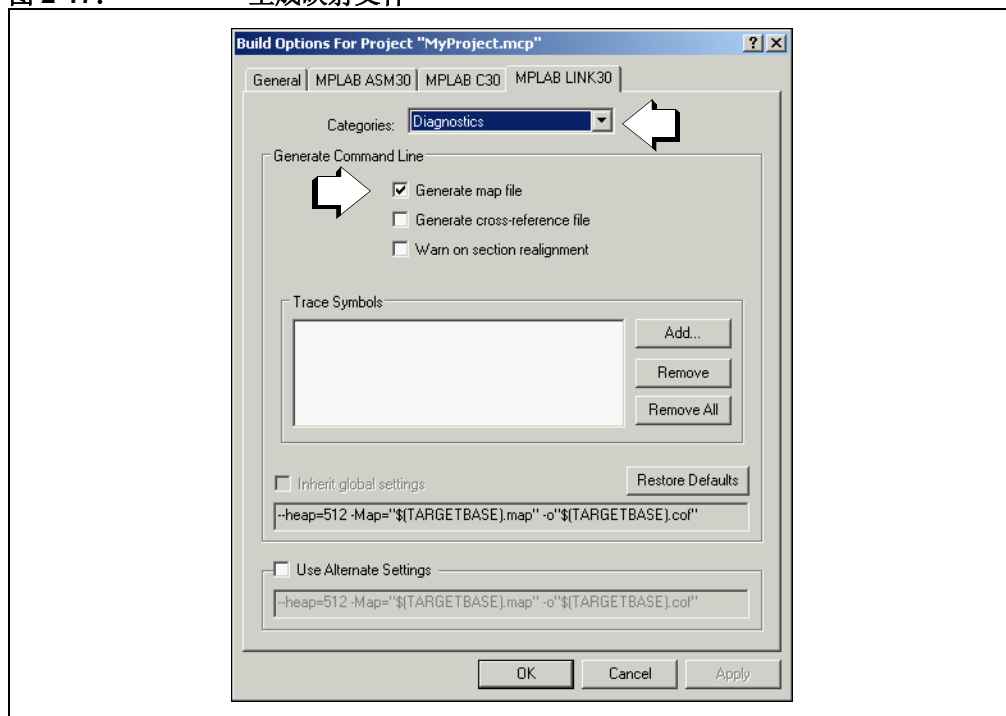
**注：** 调试时有两种基本模式：停止或运行。大多数调试操作都是在停止模式下完成的。在运行模式下，大多数调试功能都是不可操作的，寄存器不能观察或修改，项目不能重新编译。在运行模式下，也无法访问正在运行的目标单片机的存储器或内部寄存器。

## 2.9 生成映射文件

映射文件可提供在调试时有用的附加信息，如存储器分配的详细信息。这个文件可通过设置合适的链接器编译选项来生成。

1. 选择 **Project>Build Options>Project**，然后点击 **MPLAB LINK30** 选项卡。
2. 从“Categories”中选择“Diagnostics”并勾选“Generate map file”复选框。
3. 点击 **OK** 保存设置。
4. 重新编译项目（**Project>Build All**）生成映射文件。

图 2-17: 生成映射文件



映射文件（MyProject.map）出现在项目目录中，可通过选择 **File>Open**，然后浏览至项目目录来打开。选择文件类型为“**All files (\*.\*)**”以便可以看见映射文件。下面这段 MyProject.map 文件的摘录说明了在 MyProject.C 编译后程序存储器和数据存储器的使用。

### 例 2-2: 映射文件摘录

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	0xa0	0xf0	(240)
.dinit	0x1a0	0x8	0xc	(12)
Total program memory used (bytes):			0x276	(630) <1%

Data Memory Usage

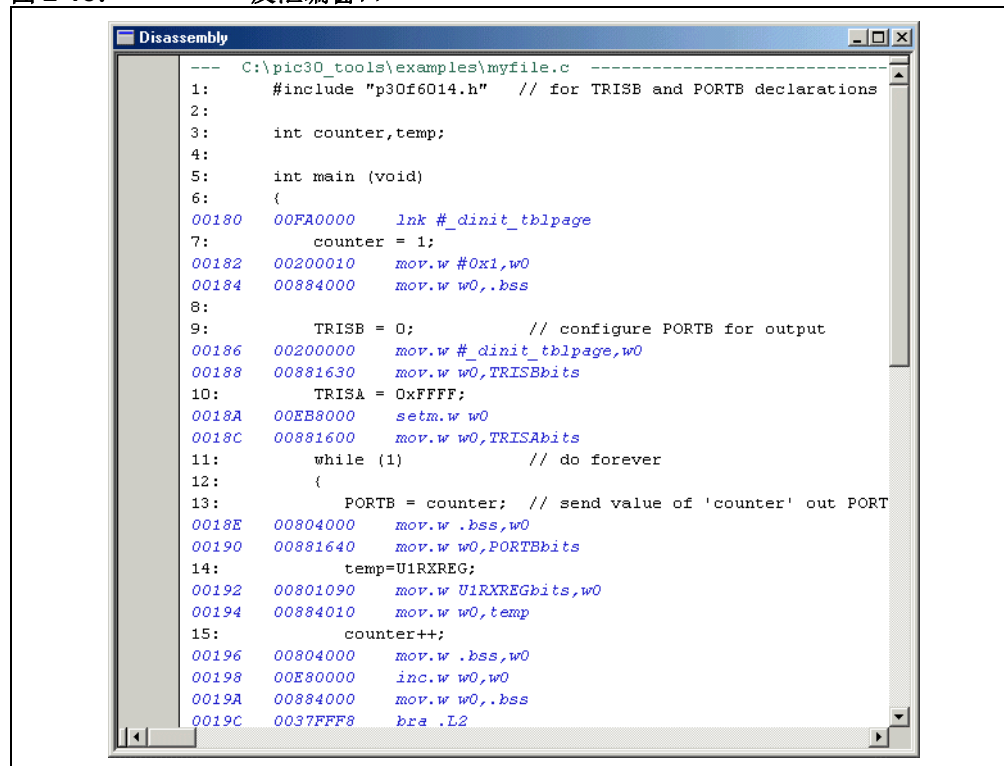
section	address	alignment gaps	total length	(dec)
.bss	0x800	0	0x4	(4)
Total data memory used (bytes):			0x4	(4) <1%

## 2.10 汇编代码的调试

至今为止所有的调试都是在 C 源文件上进行的，使用在 C 代码中定义的函数和变量。对于嵌入式系统编程来说，有时需要编写汇编代码。MPLAB IDE 提供了可同时满足这两种要求的工具，并给出了 C 代码和生成的机器码的关联。

1. 选择 MPLAB IDE 的 **View>Disassembly Listing** 窗口来查看包含源代码以及生成的机器码和汇编代码。这对于调试包含 C 代码和汇编代码的混合程序，以及需要查看由 C 源代码生成的机器码都是很有用的。

图 2-18: 反汇编窗口

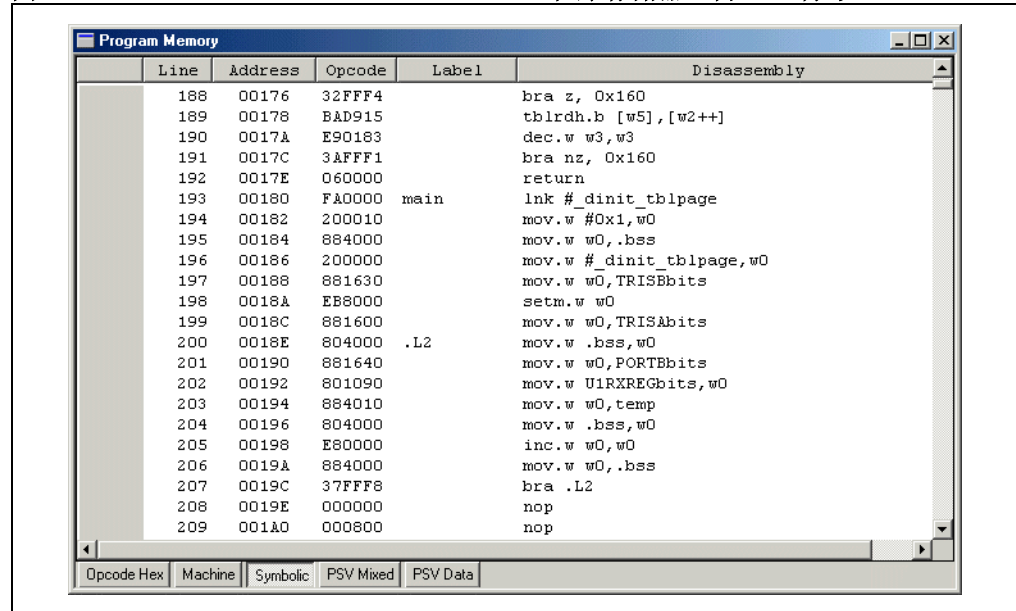


```
--- C:\pic30_tools\examples\myfile.c -----
1:  #include "p30f6014.h" // for TRISB and PORTB declarations
2:
3:  int counter,temp;
4:
5:  int main (void)
6:  {
00180 00FA0000 lnk #_dinit_tblpage
7:      counter = 1;
00182 00200010 mov.w #0x1,w0
00184 00884000 mov.w w0,.bss
8:
9:      TRISB = 0; // configure PORTB for output
00186 00200000 mov.w #_dinit_tblpage,w0
00188 00881630 mov.w w0,TRISBbits
10:     TRISA = 0xFFFF;
0018A 00EB8000 setm.w w0
0018C 00881600 mov.w w0,TRISAbits
11:     while (1) // do forever
12:     {
13:         PORTB = counter; // send value of 'counter' out PORT
0018E 00804000 mov.w .bss,w0
00190 00881640 mov.w w0,PORTBbits
14:         temp=UIRXREG;
00192 00801090 mov.w UIRXREGbits,w0
00194 00884010 mov.w w0,temp
15:         counter++;
00196 00804000 mov.w .bss,w0
00198 00E80000 inc.w w0,w0
0019A 00884000 mov.w w0,.bss
0019C 0037FFF8 bra .L2
```

在 C 源代码的左边给出了源代码在源文件中的行号。在生成的 16 进制机器码和相应反汇编指令的左边，显示了地址。对于机器码指令来说，左列是指令在程序存储器中的地址，其后是指令的 16 进制字节，最后是 dsPIC30F 反汇编指令。

2. 选择 **View>Program Memory** 窗口只查看程序存储器中的机器码和汇编代码。

**图 2-19: PROGRAM MEMORY (程序存储器) 窗口—符号**



通过选择 Program Memory 窗口底部的不同选项卡，可通过符号标号、原始 16 进制映像、混合的 PSV 代码和数据或仅 PSV 数据方式查看代码。

**注：** 关于 PSV 数据的更多信息，参见 dsPIC 器件数据手册。

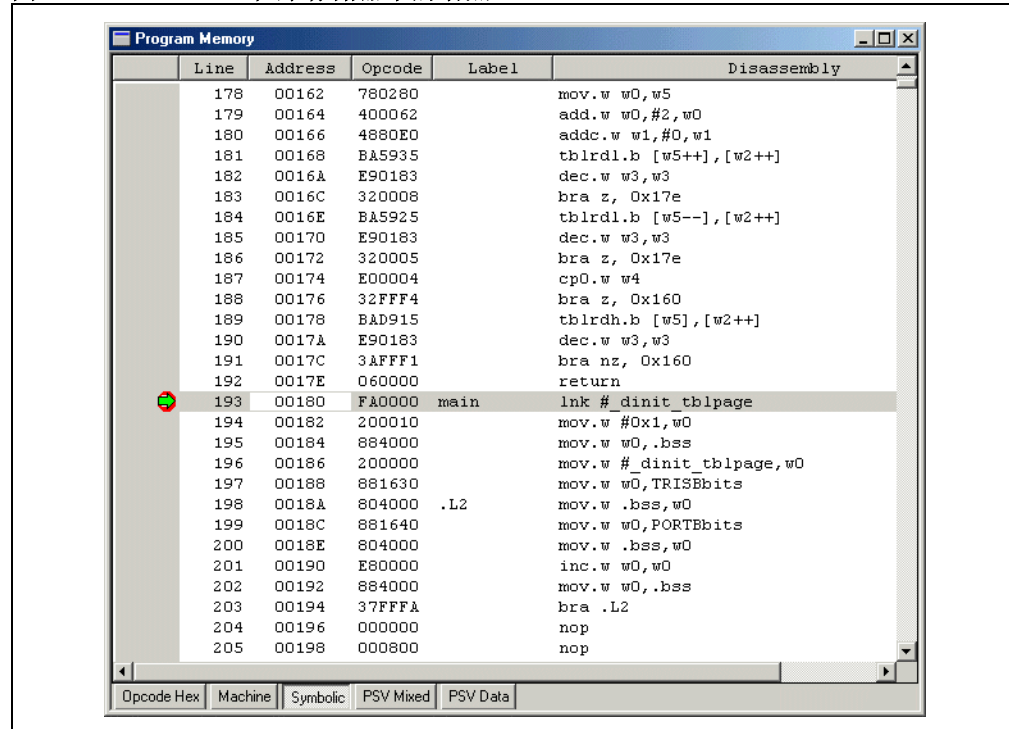
在任何源代码、反汇编和 Program Memory 窗口中都可以设置断点、单步运行及执行所有的调试功能。

3. 确保按下停止按钮后程序停止运行。在 Program Memory 窗口中点击底部的 Symbolic 选项卡查看带有标号的代码。向下滚动至带有 main 标号的行并点击它，该行对应 C 文件中的 main() 函数。通过鼠标右键在 main 上设置断点。按 RESET 按钮（或选择 **Debugger>Reset** 并选择 Processor Set（处理器复位））。



4. 现在点击 RUN。程序将在设置在 main 上的断点处停止。

图 2-20: 程序存储器中的断点



5. 返回到源文件窗口 (*File>Open*) 和反汇编窗口 (*View>Disassembly Listing*)。应该可以在所有三个窗口中看到断点。现在,可以在任何窗口中使用单步运行功能,来单步运行 C 源代码行或单步运行汇编代码。

## 2.11 深入学习

继续实践这个范例程序。需要进一步学习的内容包括:

- 通过点击 Watch 窗口中 counter 的值并输入新的值来改变它的值。
- 在 counter 的定义中为它赋一个初值。观察源代码来查看这个值何时被装入 counter。

---

---

## 第 3 章 教程 2 — 实时中断

---

---

### 3.1 简介

本教程讲述通过使用 MPLAB<sup>®</sup> IDE 软件自带的基本“模板”文件编写实时中断代码。使用 dsPIC30F6104 的 Timer1 产生重复的中断来测量 1 秒的时间间隔。

本教程由以下几个部分组成：

- 使用模板文件
- 在新项目中使用模板
- 使用 MPLAB SIM 软件模拟器进行调试
- 深入学习

### 3.2 使用模板文件

模板文件是可作为基本框架来构建应用程序的源代码文件。利用模板文件可以很容易地开始一个应用程序的项目，因为在这种简单的文件中提供了 C 语言的语法结构和格式，只需向其中添加应用程序的详细代码。模板包含了 MPLAB C30 源代码许多通用功能的示例 C 语句，包括变量和常量、特定处理器的头文件、中断向量和相关的中断代码，以及可向其中插入应用代码的代码段。

模板中还附有注释来帮助识别关键的语法结构。在许多情况下，定义了宏来简化代码。以最简单的形式来说，这里有一个简化的模板，不含注释和宏，我们可以看看它最基本的构成：

#### 例 3-1: 模板文件的元素

```
#include "p30F6014.h" /* proc specific header */

#define CONSTANT1 10 /* sample constant definition */

int array1[CONSTANT1] __attribute__((__space__(xmemory), __aligned__(32))); /* array with dsPIC30F attributes */
int array5[CONSTANT2]; /* simple array */

int variable1 __attribute__((__space__(xmemory))); /* variable with attributes */
int variable3; /* simple variable */

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

void __attribute__((__interrupt__(__save__(variable1,variable2)))) _INT0Interrupt(void) /* interrupt routine code */
{
    /* Interrupt Service Routine code goes here */
}
```

这段模板代码以 #include 语句开头，来包含具有特定处理器（dsPIC30F6014）的针对处理器特殊功能寄存器定义的头文件。随后是一个简单的常量定义（#define），可以对其进行修改和复制，来形成应用程序的一系列常量定义。

后面是一个数组定义，表明了如何定义数组的各种属性，指定其在存储器中的段，以及它在 dsPIC 的存储器结构中的对齐方式。第二个数组定义，array5，是一个简单的数组。

和数组一样，可以为变量指定属性（variable1）或不指定属性（variable3）。

再下来是 main() 的代码段。这里是放置应用代码的地方。在 main () 后面是一个中断的代码结构。

实际的应用程序可能会使用不同的中断、不同的属性，并且可能比这个例子要复杂很多。但这个模板为我们提供了一个简单的开始方式。可将未修改的模板和适当的链接描述文件添加到一个新项目中，这个项目将会毫无错误地通过编译。

模板保存在 dsPIC 工具安装目录下名为 \support\templates 的文件夹中，并在相应的 \asm 和 \c 文件夹中提供了汇编和 C 源文件。

下面是 dsPIC30F6014 的 C 模板文件的完整源代码：

### 例 3-2: TEMP\_6014.C 模板文件

```
/*
 * 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.
 * Add the suitable linker script (e.g., p30f6014.gld) to the project.
 *
 * 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:
 *
 * MPLAB C30 Compiler User's Guide : C30.pdf
 * MPLAB C30 Compiler Reference Guide : R30.pdf
 * dsPIC 30F Assembler, Linker and Utilities User's Guide : ALU.pdf
 * dsPIC 30F 16-bit MCU Family Reference Manual : DS70046
 * dsPIC 30F Sensor and General Purpose Family Data Sheet : DS70083
 * dsPIC 30F Programmer's Reference Manual : DS70030
 *
 * Template file has been compiled with MPLAB C30 V 1.3.
 */
*****
*
* Author:
* Company:
* Filename: temp_6014.c
* Date: 08/20/2004
* File Version: 1.30
* Other Files Required: p30F6014.gld, libpic30.a
* Tools Used: MPLAB GL -> 6.60
* Compiler -> 1.30
* Assembler -> 1.30
* Linker -> 1.30
*
* Devices Supported:
* dsPIC30F2011
* dsPIC30F3012
* dsPIC30F2012
* dsPIC30F3013
* dsPIC30F3014
* 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. *
*
* 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 *
* all the keywords mentioned above. *
*
*****/

/* Include the appropriate header (.h) file, depending on device used */
/* Replace the path shown here with the header path in your system */
/* Example (for dsPIC30F5013): #include "Your_path\p30F5013.h" */

/* Alternatively, the header file may be inserted from the Project */
/* window in the MPLAB IDE */

#include "p30F6014.h"

/* Define constants here */

#define CONSTANT1 10
#define CONSTANT2 20

/* Define macros to simplify attribute declarations */

#define ModBuf_X(k) __attribute__((__space__(xmemory), __aligned__(k)))
#define ModBuf_Y(k) __attribute__((__space__(ymemory), __aligned__(k)))

/***** 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 above */
int array3[CONSTANT1] ModBuf_X(32);
int array4[CONSTANT1] ModBuf_Y(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__)) _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__)) _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)))) _INT0Interrupt(void)
{
/* Interrupt Service Routine code goes here */
}
/***** END OF INTERRUPT SERVICE ROUTINES *****/
```

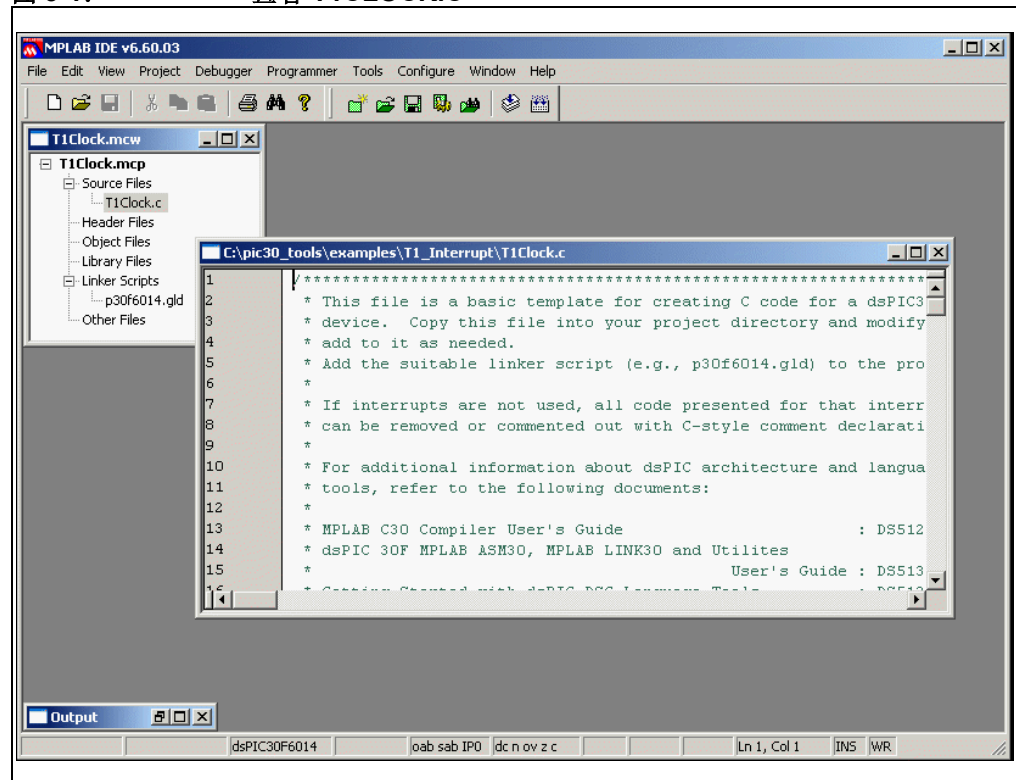
### 3.3 在新项目中使用模板

对于这个教程，可通过以下的步骤将上面介绍的模板复制到新项目目录中。可以在 Windows<sup>®</sup> 资源管理器中对文件夹 / 文件进行操作。

1. 在 MPLAB C30 安装目录下的 \Examples 目录中创建一个名为 \T1\_Interrupt 的新文件夹。
2. 将 C:\pic30\_tools\support\templates\c\temp\_6014.c 复制到这个新的 \T1\_Interrupt 文件夹中。
3. 将 \T1\_Interrupt 文件夹中复制的模板文件 temp\_6014.c 重命名为 T1Clock.c。
4. 返回 MPLAB IDE。

按照第 2 章“教程 1 — 创建项目”中的步骤使用项目向导在这个目录中创建一个新的项目 T1Clock，将 T1Clock.c 作为唯一的源文件添加到项目中，并添加 dsPIC30F6014 的链接描述文件。然后双击项目窗口中的文件名 T1Clock.c，桌面将如下图所示：

图 3-1: 查看 T1CLOCK.C



现在可以把这个通用模板的一些头文件注释删除，并把针对应用的信息输入到新的项目中。文件开头的头文件部分应包含新项目的信息。当编辑完成后，显示如下：

图 3-2: 编辑后的 T1CLOCK.C 头文件

```

1  /*****
2  *
3  *   Author:      F. Bar
4  *   Company:    Widgets, Inc.
5  *   Filename:    T1Clock.c
6  *   Date:       01/7/2003
7  *   File Version: 1.00
8  *   Other Files Required: p30F6014.gld, libpic30.a
9  *   Tools Used:  MPLAB GL -> 6.20
10 *               Compiler -> 1.10
11 *               Assembler -> 1.10
12 *               Linker -> 1.10
13 * *****/
14
15
16 #include "c:\pic30_tools\support\h\p30F6014.h"
17
18 #define CONSTANT1 10
19 #define CONSTANT2 20

```

在这个教程中，需要定义一个常量、两个变量和一个数组。模板中定义的常量名为 CONSTANT1 和 CONSTANT2。把这两个常量的定义注释掉。在 CONSTANT2 行的下面添加一行注释和对 TMR1 \_PERIOD 0x1388 的定义：

```

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

```

**注：** 周期 0x1388= 十进制数 5000。定时器的计数速率为振荡器频率的 1/4。在 5MHZ（20MHZ 振荡器四频分）下 5000 个周期，计数器将每 1ms 出现一次溢出。

在本例中可以定义一些变量对代码操作进行跟踪。可以在 GLOBAL DEFINITIONS 部分的 variable3 定义后定义这些变量。添加两个新的整型变量 main\_counter 和 irq\_counter。然后，为定时器中断服务程序创建一个具有三个无符号整型变量成员 timer、ticks 和 seconds、名为 RTclock 的结构：

### 例 3-3: 变量定义

```

/* 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;

```

这个教程中其他的模板代码可以保留或注释掉。此时把它们注释掉可能更好一点，因为这些定义会被编译并占用存储空间。确认已经将所有示例数组注释掉，因为它们使用了可以注释掉的宏定义。随着代码的增加，也很难记住哪些代码是应用程序使用的，哪些是模板文件原有的。

**注：** 在使用模板时，记住开始编写应用代码时只需要用到少量的模板元素。另外，注释掉不使用的代码部分是很有用的，这样在以后需要类似元素时，可以作为模板重新引用。

在标记为 END OF GLOBAL DEFINITIONS 的段后面输入这个函数，使用内部时钟将 Timer 1 初始化为一个中断定时器（粗体字符为应该输入的内容）：

**例 3-4:                    RESET\_CLOCK 代码**

```
/****** 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 *****/
```

这个函数使用了在头文件 p30F6014.h 中定义的特殊功能寄存器，如 TMR1 和 T1CONbits.TCS。关于 Timer1 的这些控制位和寄存器的更多信息，请参考数据手册。

可能需要编写主函数和中断服务程序。中断服务程序是最复杂的子程序。它在 Timer1 计数 0x1388 个周期时执行。在每个 1ms 的中断递增计数器 sticks，直到 1000 为止。然后递增结构 RTclock 中的变量 seconds，并复位 sticks。这个程序以秒来对时间进行计数。在标记为“START OF INTERRUPT SERVICE ROUTINES”的部分中，在为 T1Interrupt() 代码编写的模板处，用下面的代码行（添加的代码是粗体字符）替换注释“/\* Interrupt Service Routine code goes here \*/”：

## 例 3-5: 中断服务程序

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

void __attribute__((__interrupt__, __shadow__)) _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. */
```

在模板文件中有三个示例中断函数。注释掉 `_INT0Interrupt()`，因为它使用了模板文件中的两个示例变量而不会被编译。`_ADCInterrupt()` 也可以注释掉，因为在本教程中用不到它。

与 **Timer 1** 中断代码相比，`main()` 代码比较简单。在其中输入下面的代码，替换“`/* code goes here */`”行（添加的代码为粗体字符）：

## 例 3-6: 主函数代码

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

int main ( void )
{
    reset_clock();

    for (;;)
        main_counter++;
}

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

`main()` 代码只是对 **Timer 1** 初始化函数的调用，后面是一个无限循环，允许 **Timer 1** 中断工作。一般情况下，使用定时器的应用程序将被放在这个循环中，代替测试变量 `main_counter`。

最终的代码如下：

### 例 3-7: 最终的 C 代码文件

```

/*****
 *
 * Author:          F. Bar
 * Company:        Widgets, Inc.
 * Filename:       T1Clock.c
 * Date:           08/20/2004
 * File Version:   1.30
 * Other Files Required: p30F6014.gld, libpic30.a
 * Tools Used:    MPLAB GL -> 6.60
 *                Compiler -> 1.30
 *                Assembler -> 1.30
 *                Linker -> 1.30
 *****/

#include "c:\pic30_tools\support\h\p30F6014.h"

/* Define constants here */
/* #define CONSTANT1 10
   #define CONSTANT2 20
   */
/* Timer1 period for 1 ms with FOSC = 20 MHz
#define TMR1_PERIOD 0x1388
   */

/* Define macros to simplify attribute declarations */

#define ModBuf_X(k) __attribute__((__space__(xmemory), __aligned__(k)))
#define ModBuf_Y(k) __attribute__((__space__(ymemory), __aligned__(k)))

/***** 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 above
   */
/* int array3[CONSTANT1] ModBuf_X(32);
   int array4[CONSTANT1] ModBuf_Y(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 *****/

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 *****/
int main ( void )
{
    reset_clock();

    while (1)
        main_counter++;
}

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

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

void __attribute__((__interrupt__, __shadow__)) _T1Interrupt(void)
{
    static int sticks=0;

    irq_counter++;

    if (RTclock.timer > 0) /* if countdown 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 */

    return;
}

/* Interrupt Service Routine 3: INT0Interrupt */
/* Save and restore variables var1, var2, etc. */
/* void __attribute__((__interrupt__(__save__(variable1)))) _INT0Interrupt(void) */
*/

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

如果输入正确，选择 **Project>Build All** 可以得到成功的编译。如果有错误，双击 **Output** 窗口中的错误消息返回到源代码中，修复错误并重新编译项目，直到没有错误为止。

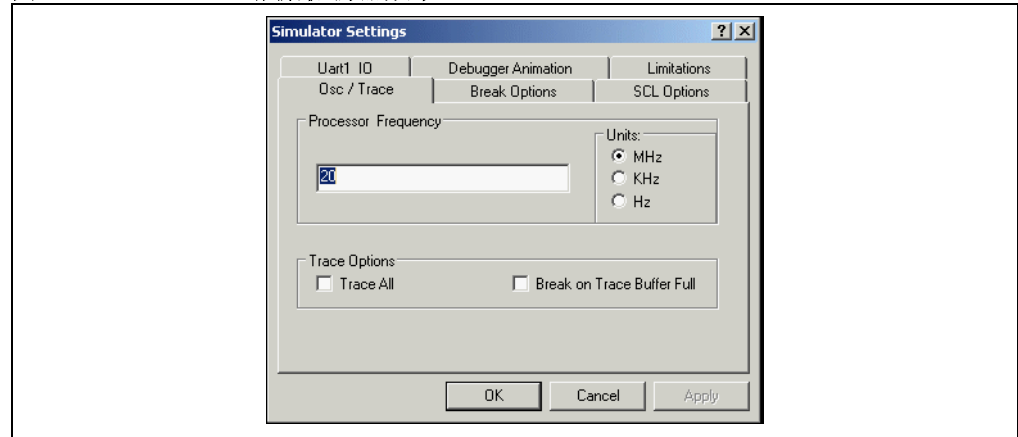


## 3.4 使用 MPLAB SIM 软件模拟器进行调试

现在，可以用 MPLAB SIM 软件模拟器调试代码了。确保选择了 *Debugger>Select Tool>MPLAB SIM*。然后选择 *Debugger>Settings* 对模拟器的处理器时钟速度进行设置。在振荡器（**Osc/Trace**）选项卡对话框中，可以设置模拟的 dsPIC30F6014 的时钟频率。将时钟频率设置为 20MHz。

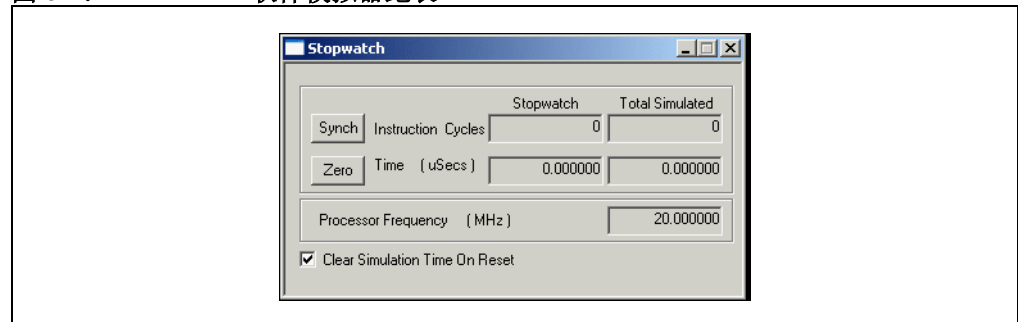
**注：** 软件模拟器的运行速度由 PC 机决定，所以它不可能以对话框中设置的 dsPIC30F MCU 的实际速度运行。但是，所有的定时计算都是基于这个时钟设置的，因此，当使用模拟器进行时序测量时，测量的时间是与器件在该频率下的实际运行时间相对应的。

**图 3-3:** 激励振荡器频率



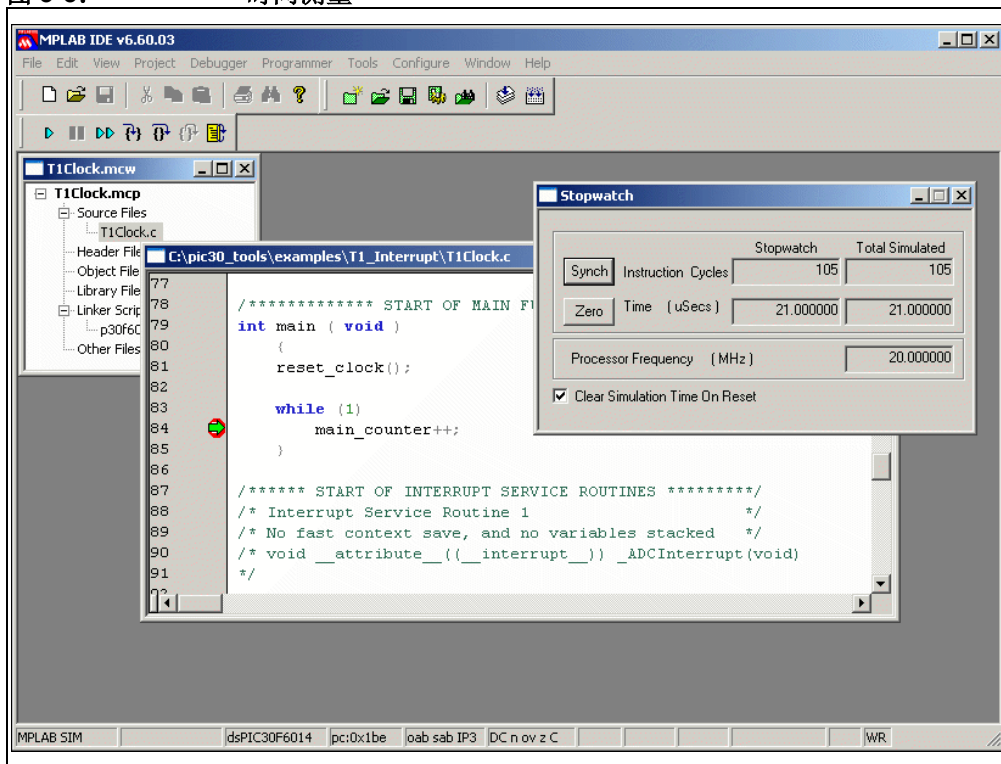
用模拟器测量时间的一个方法是使用跑表功能。选择 *Debugger>Stopwatch* 来打开 Stopwatch（跑表）对话框，确保选中“Clear Simulation on Reset”复选框。

**图 3-4:** 软件模拟器跑表



首先要至少验证一下，程序是否在运行。为此，可以在 main() 中递增 main\_counter 的代码行上设置一个断点（在该行上点击鼠标右键并选择“Set Breakpoint”），然后点击 RUN 图标或选择 **Debugger>Run**。程序运行到断点后，跑表和屏幕应如下图所示。

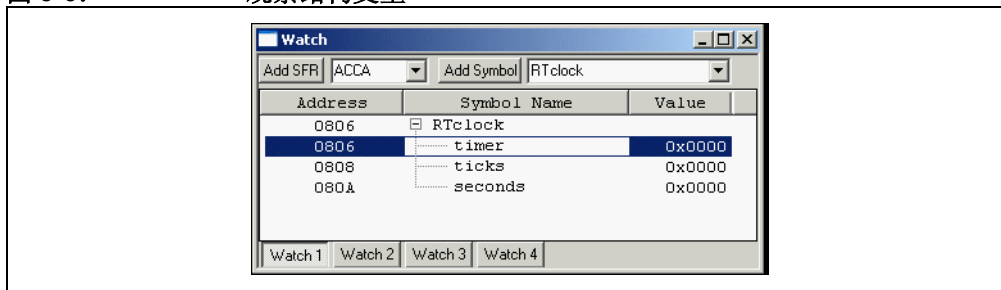
图 3-5: 时间测量



如果运行成功，就可以设置一个 Watch 窗口来检查程序的变量。选择 **View>Watch** 来调出 Watch 窗口。添加变量 RTclock（从 **Add Symbol** 旁边的下拉框中选择）。

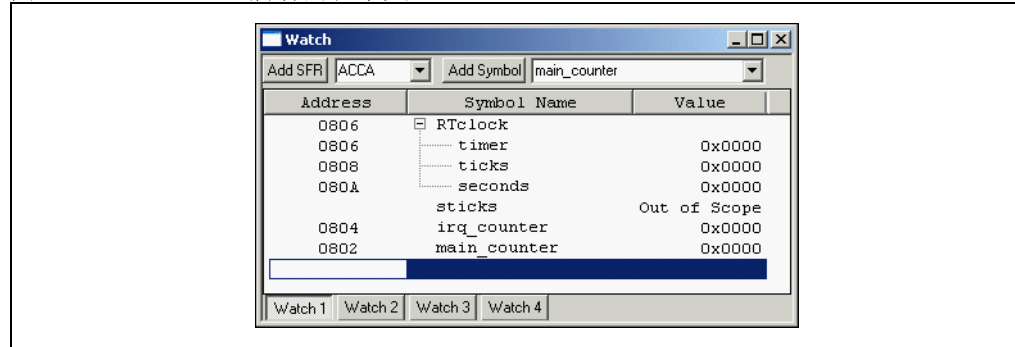
Rtclock 是一个结构，正如其名字左边方框中的小加号所示。点击方框打开这个结构，如下图所示：

图 3-6: 观察结构变量



除 Rtclock 外，在 Watch 窗口中添加变量 sticks、irq\_counter 和 main\_counter。

图 3-7: 所有的观察变量



可以把 Value 这一列拉宽以便读到 sticks 变量在这一列中的内容。你将看到它写着“Out of Scope”。意思是说，与 Rtclock、irq\_counter 和 main\_counter 不同，这不是一个全局变量，只有在执行函数 `_T1Interrupt()` 时才能对它的值进行访问。

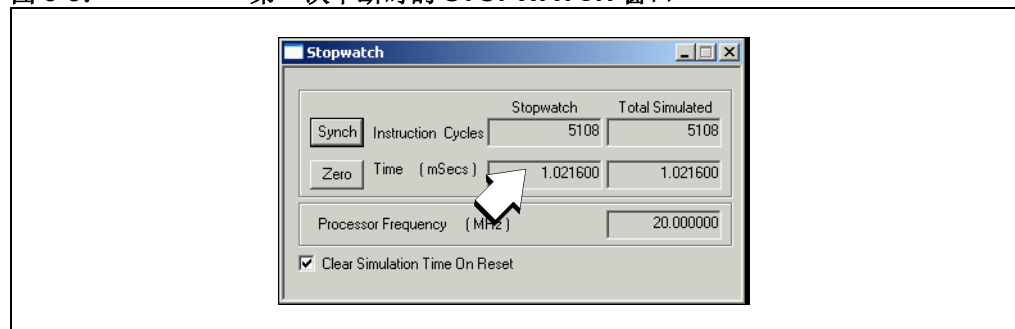
**注：** sticks 的地址列中没有值。这再一次表明它是一个局部变量。

第一次执行到断点时，查看 Watch 窗口中的变量，所有变量都应等于 0。这是可以理解的，因为此时 Timer 1 刚刚被初始化，而 counter 还没开始第一次递增。

点击 Step Into 图标单步执行一次 main() 循环。此时，main\_counter 的值显示为 0001。中断服务程序还没有启动。查看 Stopwatch 窗口，每次 main() 循环用的时间只增加 1 微秒。要进入第一次中断，必须执行 1000 次循环 ( $1000 \times 1 \text{ us} = 1 \text{ ms}$ )。

为了检测中断功能，可先通过鼠标右键点击选中的行并选择“Remove Breakpoint”来去掉设置在 main\_counter++ 上的断点。然后选择右击鼠标菜单中的“Set Breakpoint”在中断服务程序中的 irq\_counter++ 语句上设置一个断点。接下来点击 Run。Stopwatch 窗口将显示如下：

图 3-8: 第一次中断时的 STOPWATCH 窗口

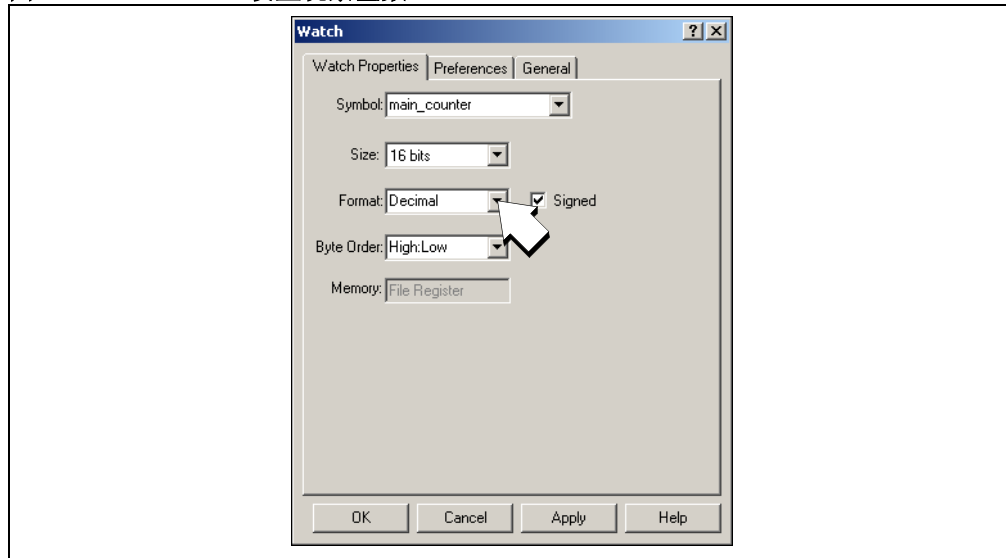


Stopwatch 的 Time 窗口显示的值为 1.0216ms。正如我们所希望的，每一毫秒产生一次中断。复位后需要一段时间，包括 C 启动代码和 Timer 1 初始化的时间，跑表测量出了这段时间。

查看 Watch 窗口。变量 `main_counter` 显示的值为 `0x3E8`。按下列步骤可将显示基数改成十进制：

1. 点击 `main_counter` 选择 Watch 窗口中的行。然后，使用鼠标右键选择“Properties”。
2. 在 Watch 对话框的 Watch Properties 选项卡中，从“Format”下拉菜单中选择“Decimal”。
3. 点击 **OK**。

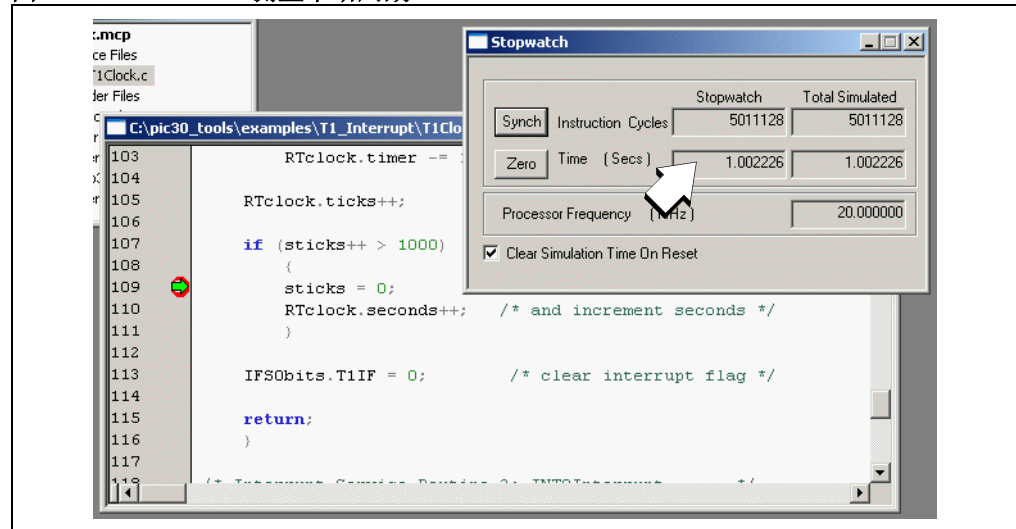
图 3-9: 设置观察基数



现在 `main_counter` 的值应显示为 `1000`。多次点击 **Step Into** 图标查看变量的变化情况，特别是 `sticks` 和 `irq_counter`，每产生一次中断它们的值都会递增。

移除 `irq_counter++;` 行上的断点，将断点设置在递增 `sticks` 的条件语句中（在 `sticks = 0;` 这一行）。点击 **Run** 运行并停止在断点处。窗口将显示如下：

图 3-10: 测量中断周期



Stopwatch 的 Time 窗口显示值为 1.002226 秒，大约是一个 1 秒的中断。准确的时间测量可以测出到达下一个中断的时间。然后从当前的时间减去测出的值。或者说，我们所关注的是两次中断之间的时间——按下 Stopwatch 对话框中的 **Zero**，接着再按 **Run**。因为对于处理器用了多长的时间运行到这我们并不在乎。

**注：** 跑表总是在对话框右边的窗口中跟踪总的时间。左边的窗口用来对独立的测量进行计时。按下 **Zero** 不会导致总的时间发生改变。

### 3.5 深入学习

继续实践一下这个示例程序。需要进一步学习的内容包括：

- 测量中断的开销，计算它是如何影响计时的；试着改变常量 `TMR1_Period` 的值来调节中断以获得更好的 1 秒准确度。
- 用这个程序测量的最大时间（以分钟为单位）是多少？如何提高测量范围？
- 添加一个子程序，在端口每秒输出一个 2 毫秒的脉冲。使用跑表来验证脉冲的宽度。

注:

---

---

## 第 4 章 教程 3 — 混合使用 C 文件与汇编文件

---

---

### 4.1 简介

本教程将介绍如何创建一个混合使用 C 文件和汇编文件的项目，通过 C 源文件调用汇编语言子程序。

本教程由以下几个部分组成：

- 获得项目源文件
- 创建和编译项目
- 检查程序
- 深入学习
- 今后如何使用

### 4.2 获得项目源文件

本教程中使用的文件可以在 \Examples 文件夹中找到。它们分别是 C 源代码文件 example3.c 和汇编语言文件 modulo.s。在 \Examples 文件夹中创建一个名为 \DSP\_ASM 的文件夹，并把这两个文件复制到新的文件夹中。如何做请参考第 3 章“教程 2 — 实时中断”。

作为参考，例 4-1 和例 4-2 给出了这两个文件的程序清单。

#### 例 4-1: C 源文件

```
/*
 * Filename:      example3.c
 * Date:         08/20/2004
 * File Version:  1.30
 * Tools used:   MPLAB    -> 6.60
 *               Compiler  -> 1.30
 *               Assembler -> 1.30
 *               Linker    -> 1.30
 * Linker File:   p30f6014.gld
 */
*****/

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

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

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

/* output array of 32-bit products defined here */
long array3[PRODLLEN/2]; /* 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 *, unsigned int *, 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 + PRODLLEN - 1;
YMODSRT = (unsigned int)array2;
YMODEND = (unsigned int)array2 + PRODLLEN - 1;

/* Initialize 10-element arrays, array1 and array2 */
/* to values 1, 2, ..., 10 */
while (1)          /* just do this over and over */
{
    for (array_index = 0; array_index < PRODLLEN/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 4 parameters:                    */
    /* start addresses of array1, array2 and array3, and PRODLLEN-1 */
    /* in that order                                              */
    modulo( array1, array2, array3, PRODLLEN-1 );
}
}

```

## 例 4-2: MODULO.S 汇编源文件

```

/*****
*   Filename:      modulo.s
*   Date:         08/20/2004
*   File Version: 1.30
*
*   Tools used:  MPLAB    -> 6.60
*                Compiler -> 1.30
*                Assembler -> 1.30
*                Linker   -> 1.30
*
*   Linker File:  p30f6014.gld
*   Description:  Assembly routine used in 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

        ; turn on modulo addressing
        MOV     #0xC0A8, W8
        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 'PRODLLEN - 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++]

        ; turn off modulo addressing
CLR    MODCON

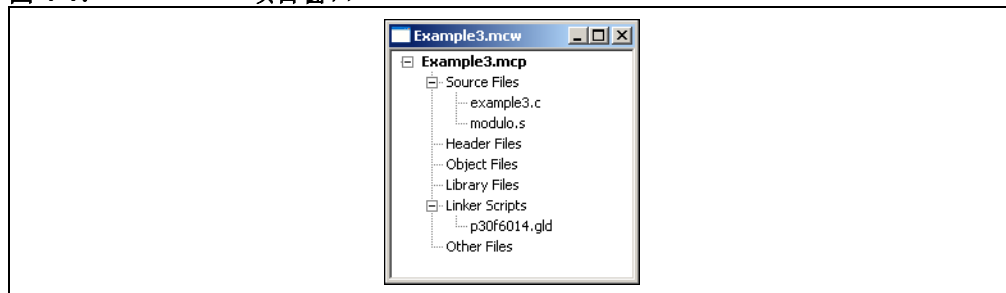
        RETURN

.end
```

## 4.3 创建和编译项目

使用 **Project Wizard** (项目向导)，创建一个新项目，并将这两个源文件和链接描述文件 `p30f6014.gld` 添加到这个项目中。具体做法见第 2 章“教程 1 — 创建项目”。项目窗口应如下图所示：

图 4-1: 项目窗口



本教程将使用标准的 I/O 函数 `printf()` 在 **Output** (输出) 窗口中显示消息。要使用 `printf()`，就要在链接器的 **Build Options** (编译选项) 中使能堆。确保链接器编译选项的设置如图 2-8 所示，为堆分配了 512 字节的空间。

在编译项目时 (*Project > Build All*)，不应出现错误消息。如果收到错误消息，要确认是否使用和前两个教程相同的选项设置了项目。

本教程使用了三个数组。向其中两个数组填充测试数列，然后调用汇编程序将两个 16 位数组中的值相乘，将结果存放在第三个 32 位的数组中。使用模寻址，通过调整在第二次乘法循环中指向其中一个源数组的指针来改变乘数的对齐方式，两个源数组经过两次运算，在输出数组中产生两组乘积。通过汇编语言程序，利用 `dsPIC30F6014` 的 DSP 功能进行运算。

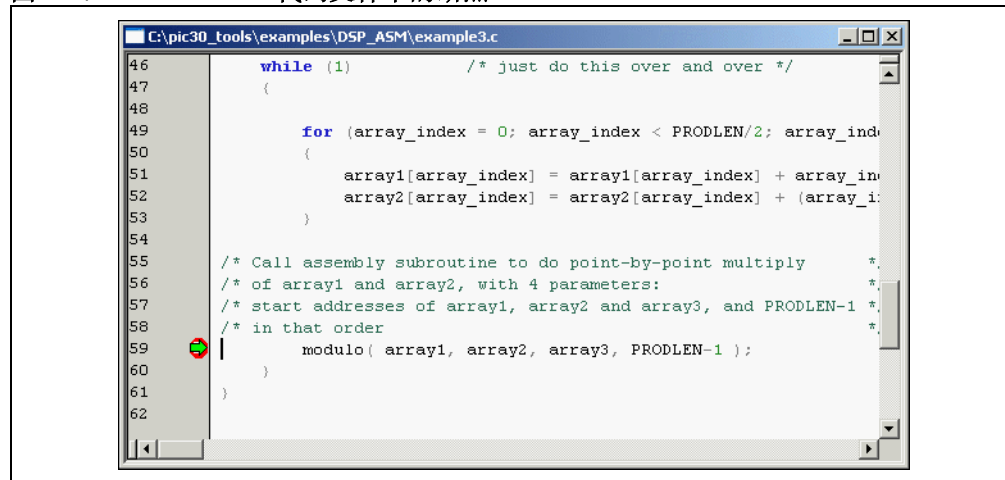
汇编语言程序有四个参数：三个数组的地址和数组长度。结果返回到乘积数组中。

这个程序连续循环运行，当程序重复执行主无限循环时，源数组中的数将不断增大。

## 4.4 检查程序

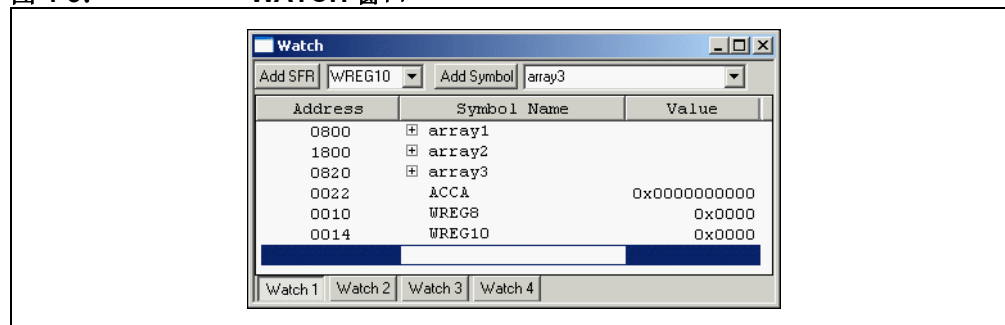
一旦项目创建好并成功编译，就可以使用 MPLAB SIM 软件模拟器 (*Debugger > Select Tool > MPLAB SIM*) 检查程序的操作。在 example3.c 中调用汇编语言程序 modulo() 的函数上设置断点并运行到断点。

图 4-2: C 代码文件中的断点



设置 Watch 窗口来查看运算所涉及的变量。在窗口中添加三个数组 array1、array2 和 array3。还要添加特殊功能寄存器 (Special Function Register, SFR) ACCA、WREG8 和 WREG10。此时 Watch 窗口应显示如下：

图 4-3: WATCH 窗口

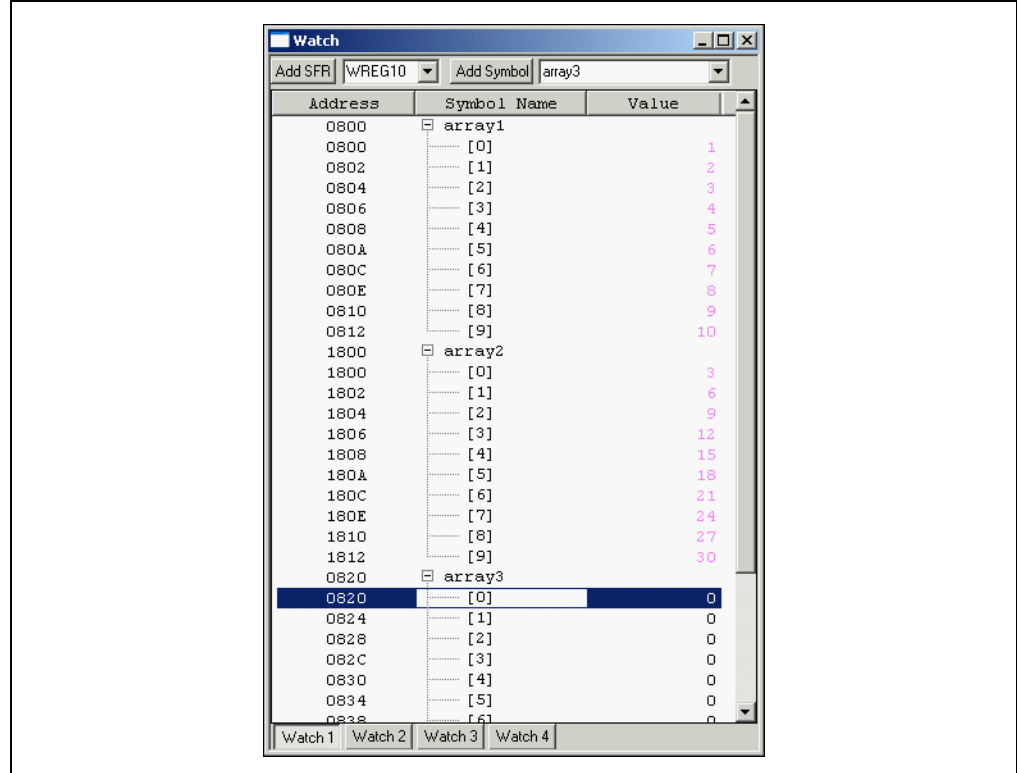


点击符号名左边的加号打开数组。程序运行到这个断点时，array1 和 array2 应该已经设置了初始值，但 array3 应还是全 0，因为还没有调用 modulo() 程序。

点击数组中的任意元素选中它，右击该元素来改变显示基数。将三个数组的基数都改为十进制。

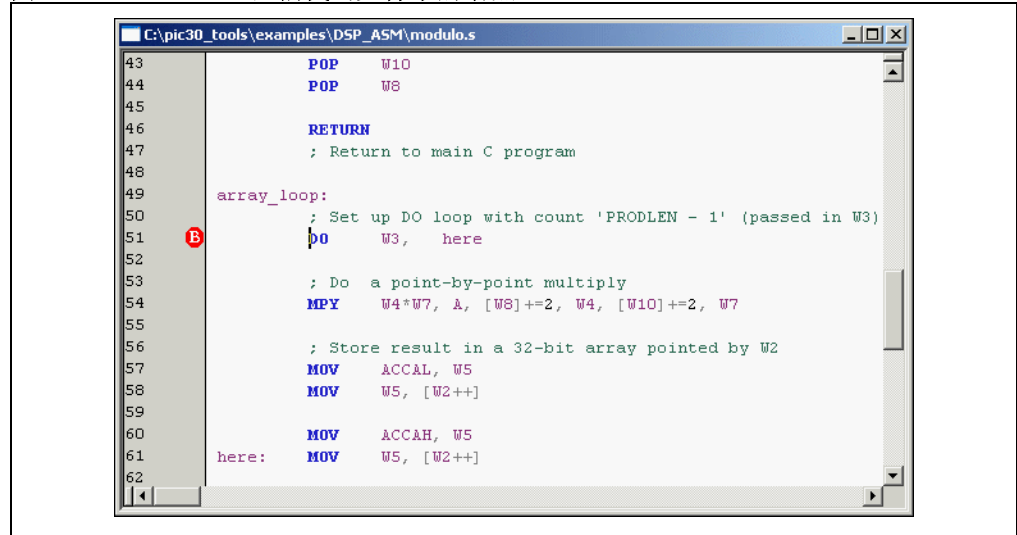
**注：** 改变数组中任意元素的基数将使该数组中所有元素的基数改变。

图 4-4: 数组设置为十进制



在 modulo.s 文件中的 DO 循环开始处设置一个断点。

图 4-5: 汇编代码文件中的断点



运行到断点处并滚动 Watch 窗口查看 array3。它应仍然为 0。再按一次 RUN，运行一次 DO 循环。现在，array3 的上半部分所显示的值代表来自源数组的每个元素对的乘积：

图 4-6: ARRAY3 的结果—第一次循环

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
0022	ACCA	0x00000000
0010	WREG8	0x0802
0014	WREG10	0x1802

再按一次 RUN，查看第二次运行 DO 循环后的结果：

图 4-7: ARRAY3 的结果—第二次循环

Address	Symbol Name	Value
0820	array3	
0820	[0]	12
0824	[1]	48
0828	[2]	108
082C	[3]	192
0830	[4]	300
0834	[5]	432
0838	[6]	588
083C	[7]	768
0840	[8]	972
0844	[9]	1200
0022	ACCA	0x0000004B0
0010	WREG8	0x0804
0014	WREG10	0x1802

移除 modulo.s 中的断点并按 RUN 查看下一次循环的结果。多按几次 RUN，看看执行乘法后值的改变情况。最后，移除 example3.c 中的断点。

当代码运行并停止在断点处时，可以通过 Watch 窗口检测数据。软件模拟器执行程序时也可以输出数据，提供运行记录以供检查和发送到其他工具进行绘图和分析。在 modulo() 函数调用后面插入一个 printf() 语句，来监控输出数组中的值。程序代码如下（粗体字符为添加的代码）：

#### 例 4-3: PRINTF () 监控程序

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

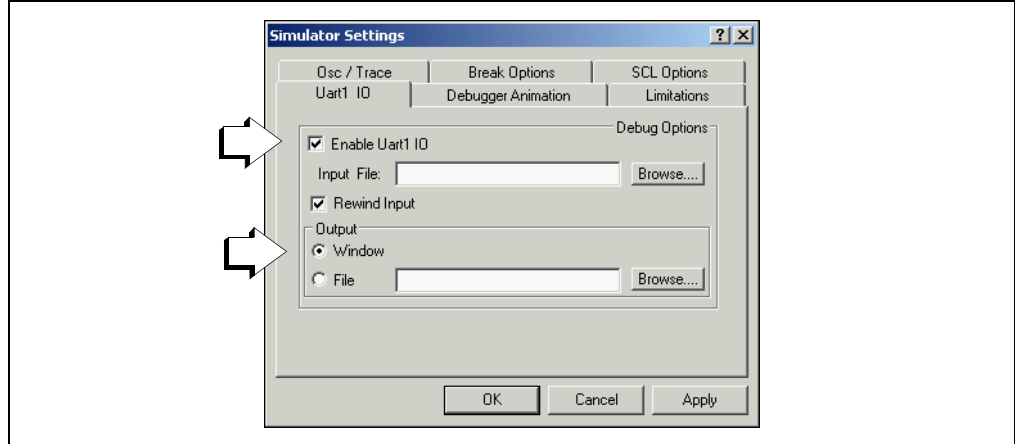
```
printf("Product Array\n");
```

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

## 教程 3 — 混合使用 C 文件与汇编文件

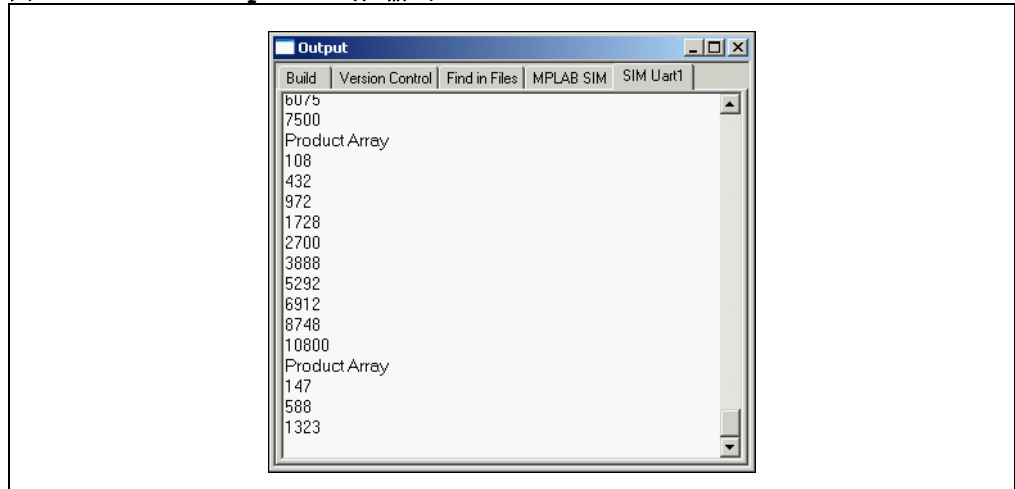
`printf()` 函数使用被模拟的 dsPIC 器件的 UART1 函数来将消息写入一个文件，或是写入输出窗口。选择 **Debugger > Settings** 进入 **Simulator Settings**（模拟器设置）对话框。点击 **UART1 IO** 选项卡，选中“Enable UART1/O”，然后选择单选框将文本内容从 `printf()` 声明发送到输出窗口中。点击 **OK**。

图 4-8: UART1 IO — `printf()` 设置



现在重新编译项目 (**Project > Build All**)，按 **Run**，让它运行几秒钟，然后按 **Halt**。如果输出窗口没有出现，可通过 **View > Output** 来激活。点击 **SIM UART1** 选项卡。输出窗口将产生 `array3` 的内容记录。

图 4-9: `printf()` 输出



## 4.5 深入学习

继续实践一下这个示例程序。需要进一步学习的内容包括：

- 试试用其他的 DSP 指令来处理这些数组中的数字。
- 使用 `printf()` 函数输出可以被导入到电子数据表中的数值列表。画出这些数值的曲线图。
- 进一步使代码通用化，这样可以在 `modulo.s` 中建立所有的模寻址（例如，将例 4-1 中的这些行转变成汇编代码，从传递到数组的参数中创建模寻址参数。）

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

## 4.6 今后如何使用

本教程旨在帮助用户熟悉在 MPLAB IDE 环境中使用 MPLAB C30 编译器。还有许多 MPLAB IDE 和 MPLAB C30 编译器的功能这里没有提到。要获得更多的信息，请参考最新的 MPLAB IDE 在线帮助、《MPLAB C30 编译器用户指南》和 *MPLAB ASM30、MPLAB LINK30 and Utilities User's Guide*，从而开始使用这些工具来开发应用。

通过 MPLAB IDE 在线帮助或登录 [www.microchip.com](http://www.microchip.com) 网站中关于 MPLAB C 产品的网络论坛就可以立即获得帮助。进入技术支持区，然后再进入在线讨论组。开发系统网页中也有一个专门的 MPLAB C30 编译器讨论区。

通过在 Microchip 网站上注册变更通知客户服务，用户可以注册获得关于 MPLAB C30 编译器变更的通知。在开发工具中选择 MPLAB C 编译器类，就可以在新版本发布时得到通知，并获得关于 MPLAB C30 编译器的最新信息。

## 索引

<b>A</b>		<b>P</b>	
安装 MPLAB ASM30、MPLAB LINK30 和语言工具实用程序 .....	7	printf() .....	45
安装 MPLAB ASM30 和 MPLAB LINK30 .....	7	printf() 输出 .....	49
安装 MPLAB C30 .....	7	跑表 .....	37
<b>B</b>		<b>S</b>	
变量定义 .....	32	时钟复位 .....	33
编译错误 .....	18	数组 .....	46
编译项目 .....	18	<b>T</b>	
编译选项 .....	14	添加文件到项目 .....	11
<b>C</b>		通知客户服务 .....	4
参考读物 .....	8, 50	推荐读物 .....	3
程序存储器窗口 .....	25	<b>U</b>	
处理器选择 .....	10	UART1 IO .....	49
创建项目 .....	10	<b>W</b>	
<b>D</b>		Watch 窗口 .....	21, 22, 46
断点 .....	20, 21, 46	WWW 地址 .....	4
<b>F</b>		文档 .....	
反汇编窗口 .....	24	编排 .....	1
<b>H</b>		约定 .....	2
混合使用 C 文件与汇编文件 .....	43	<b>X</b>	
<b>J</b>		卸载 MPLAB C30 .....	7
基数, 设置 .....	40	新项目 .....	10
结构 .....	38	项目窗口 .....	13, 45
<b>K</b>		项目向导 .....	10, 45
客户支持 .....	5	<b>Y</b>	
<b>L</b>		因特网地址 .....	4
列表文件 .....	23	映射文件 .....	23
<b>M</b>		语言工具设置 .....	10
Microchip 网站 .....	1, 4	<b>Z</b>	
MPLAB SIM 软件模拟器 .....	20, 37, 46	振荡器频率, 激励 .....	37
模板文件 .....	27	中断服务程序 .....	34
<b>O</b>		中断周期 .....	41
Output 窗口 .....	18		

注:



注:



## 全球销售及服务中心

### 美洲

公司总部 **Corporate Office**  
2355 West Chandler Blvd.  
Chandler, AZ 85224-6199

Tel: 1-480-792-7200  
Fax: 1-480-792-7277

技术支持:  
<http://support.microchip.com>  
网址: [www.microchip.com](http://www.microchip.com)

#### 亚特兰大 **Atlanta**

Alpharetta, GA  
Tel: 1-770-640-0034  
Fax: 1-770-640-0307

#### 波士顿 **Boston**

Westborough, MA  
Tel: 1-774-760-0087  
Fax: 1-774-760-0088

#### 芝加哥 **Chicago**

Itasca, IL  
Tel: 1-630-285-0071  
Fax: 1-630-285-0075

#### 达拉斯 **Dallas**

Addison, TX  
Tel: 1-972-818-7423  
Fax: 1-972-818-2924

#### 底特律 **Detroit**

Farmington Hills, MI  
Tel: 1-248-538-2250  
Fax: 1-248-538-2260

#### 科科莫 **Kokomo**

Kokomo, IN  
Tel: 1-765-864-8360  
Fax: 1-765-864-8387

#### 洛杉矶 **Los Angeles**

Mission Viejo, CA  
Tel: 1-949-462-9523  
Fax: 1-949-462-9608

#### 圣何塞 **San Jose**

Mountain View, CA  
Tel: 1-650-215-1444  
Fax: 1-650-961-0286

#### 加拿大多伦多 **Toronto**

Mississauga, Ontario,  
Canada  
Tel: 1-905-673-0699  
Fax: 1-905-673-6509

### 亚太地区

中国 - 北京  
Tel: 86-10-8528-2100  
Fax: 86-10-8528-2104

中国 - 成都  
Tel: 86-28-8676-6200  
Fax: 86-28-8676-6599

中国 - 福州  
Tel: 86-591-8750-3506  
Fax: 86-591-8750-3521

中国 - 香港特别行政区  
Tel: 852-2401-1200  
Fax: 852-2401-3431

中国 - 上海  
Tel: 86-21-5407-5533  
Fax: 86-21-5407-5066

中国 - 沈阳  
Tel: 86-24-2334-2829  
Fax: 86-24-2334-2393

中国 - 深圳  
Tel: 86-755-8203-2660  
Fax: 86-755-8203-1760

中国 - 顺德  
Tel: 86-757-2839-5507  
Fax: 86-757-2839-5571

中国 - 青岛  
Tel: 86-532-8502-7355  
Fax: 86-532-8502-7205

中国 - 武汉  
Tel: 86-27-5980-5300  
Fax: 86-27-5980-5118

台湾地区 - 高雄  
Tel: 886-7-536-4818  
Fax: 886-7-536-4803

台湾地区 - 台北  
Tel: 886-2-2500-6610  
Fax: 886-2-2508-0102

台湾地区 - 新竹  
Tel: 886-3-572-9526  
Fax: 886-3-572-6459

### 亚太地区

澳大利亚 **Australia - Sydney**  
Tel: 61-2-9868-6733  
Fax: 61-2-9868-6755

印度 **India - Bangalore**  
Tel: 91-80-2229-0061  
Fax: 91-80-2229-0062

印度 **India - New Delhi**  
Tel: 91-11-5160-8631  
Fax: 91-11-5160-8632

日本 **Japan - Kanagawa**  
Tel: 81-45-471-6166  
Fax: 81-45-471-6122

韩国 **Korea - Seoul**  
Tel: 82-2-554-7200  
Fax: 82-2-558-5932 或  
82-2-558-5934

马来西亚 **Malaysia - Penang**  
Tel: 011-604-646-8870  
Fax: 011-604-646-5086

菲律宾 **Philippines - Manila**  
Tel: 011-632-634-9065  
Fax: 011-632-634-9069

新加坡 **Singapore**  
Tel: 65-6334-8870  
Fax: 65-6334-8850

### 欧洲

奥地利 **Austria - Weis**  
Tel: 43-7242-2244-399  
Fax: 43-7242-2244-393

丹麦 **Denmark - Ballerup**  
Tel: 45-4450-2828  
Fax: 45-4485-2829

法国 **France - Massy**  
Tel: 33-1-69-53-63-20  
Fax: 33-1-69-30-90-79

德国 **Germany - Ismaning**  
Tel: 49-89-627-144-0  
Fax: 49-89-627-144-44

意大利 **Italy - Milan**  
Tel: 39-0331-742611  
Fax: 39-0331-466781

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

英国 **England - Berkshire**  
Tel: 44-118-921-5869  
Fax: 44-118-921-5820