Real-Time Data Monitor User's Guide

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

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

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

Trademarks

Printed on recycled paper.

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

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 the Real-Time Data Monitor (RTDM). Items discussed in this chapter include:

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

DOCUMENT LAYOUT

This user's guide describes how to use the Real-Time Data Monitor. The document is organized as follows:

- **Chapter 1. "Introduction"** – This chapter introduces the real-time data monitoring software developed for the MPLAB® Data Monitor and Control Interface (DMCI) integrated on the MPLAB IDE 8.10 or higher. It also outlines requirements for a host PC.
- **Chapter 2. "Getting Started"** – This chapter describes how to run the RTDM code example, CE155, and how to add RTDM code to your application code.
- **Chapter 3. "Application Programming Interface (API)"** – This chapter outlines how the API functions provided in the Real-Time Data Monitor, which can be included in your application software via the Application Programming Interface.

- **Chapter 4. "Protocol"** – This chapter describes the RTDM protocol specification, which is used to debug dsPIC® DSC devices and PIC24H embedded applications at run-time.
- **Chapter 5. "DMCI Operating Modes"** – This chapter outlines the different MPLAB DMCI operating modes used for debugging applications in a real time fashion.

CONVENTIONS USED IN THIS GUIDE

This manual uses the following documentation conventions:

DOCUMENTATION CONVENTIONS

<table>
<tr><th>Description</th><th>Represents</th><th>Examples</th></tr>
<tr><td colspan="3">Arial font:</td></tr>
<tr><td rowspan="2">Italic characters</td><td>Referenced books</td><td>MPLAB® IDE User’s Guide</td></tr>
<tr><td>Emphasized text</td><td>...is the only compiler...</td></tr>
<tr><td rowspan="3">Initial caps</td><td>A window</td><td>the Output window</td></tr>
<tr><td>A dialog</td><td>the Settings dialog</td></tr>
<tr><td>A menu selection</td><td>select Enable Programmer</td></tr>
<tr><td>Quotes</td><td>A field name in a window or dialog</td><td>“Save project before build”</td></tr>
<tr><td>Underlined, italic text with right angle bracket</td><td>A menu path</td><td><u>File>Save</u></td></tr>
<tr><td rowspan="2">Bold characters</td><td>A dialog button</td><td>Click OK</td></tr>
<tr><td>A tab</td><td>Click the Power tab</td></tr>
<tr><td>N‘Rnnnn</td><td>A number in verilog format, where N is the total number of digits, R is the radix and n is a digit.</td><td>4‘b0010, 2‘hF1</td></tr>
<tr><td>Text in angle brackets < ></td><td>A key on the keyboard</td><td>Press <Enter>, <F1></td></tr>
<tr><td colspan="3">Courier New font:</td></tr>
<tr><td rowspan="7">Plain Courier New</td><td>Sample source code</td><td><code>#define START</code></td></tr>
<tr><td>Filenames</td><td><code>autoexec.bat</code></td></tr>
<tr><td>File paths</td><td><code>c:\mcc18\h</code></td></tr>
<tr><td>Keywords</td><td><code>_asm, _endasm, static</code></td></tr>
<tr><td>Command-line options</td><td><code>-Opa+, -Opa-</code></td></tr>
<tr><td>Bit values</td><td><code>0, 1</code></td></tr>
<tr><td>Constants</td><td><code>0xFF, ‘A’</code></td></tr>
<tr><td>Italic Courier New</td><td>A variable argument</td><td><code>file.o</code>, where <code>file</code> can be any valid filename</td></tr>
<tr><td>Square brackets [ ]</td><td>Optional arguments</td><td><code>mcc18 [options] file [options]</code></td></tr>
<tr><td>Curly brackets and pipe character: { | }</td><td>Choice of mutually exclusive arguments; an OR selection</td><td><code>errorlevel {0|1}</code></td></tr>
<tr><td rowspan="2">Ellipses...</td><td>Replaces repeated text</td><td><code>var_name [,
var_name...]</code></td></tr>
<tr><td>Represents code supplied by user</td><td><code>void main (void)
{ ...
}</code></td></tr>
</table>

WARRANTY REGISTRATION

Please complete the enclosed Warranty Registration Card and mail it promptly. Sending in the Warranty Registration Card entitles users to receive new product updates. Interim software releases are available at the Microchip web site.

RECOMMENDED READING

This user’s guide describes how to use the Real-Time Data Monitor. Other useful documents include:

dsPIC30F Family Reference Manual (DS70046)

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

dsPIC33F Family Reference Manual Sections

Refer to these documents for detailed information on dsPIC33F device operation. These reference manual sections explain the operation of the dsPIC33F MCU family architecture and peripheral modules, but do not cover the specifics of each device. Refer to the appropriate device data sheet for device-specific information.

dsPIC30F/dsPIC33F Programmer’s Reference Manual (DS70157)

This manual is a software developer’s reference for the dsPIC30F and dsPIC33F 16-bit MCU families of devices. It describes the instruction set in detail and also provides general information to assist in developing software for the dsPIC30F and dsPIC33F MCU families.

MPLAB® ASM30, MPLAB® LINK30 and Utilities User’s Guide (DS51317)

This document helps you use Microchip Technology’s language tools for dsPIC DSC devices based on GNU technology. The language tools discussed are:

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

MPLAB® C30 C Compiler User’s Guide (DS51284)

This document helps you use Microchip’s MPLAB C30 C compiler for dsPIC DSC devices to develop your application. MPLAB C30 is a GNU-based language tool, based on source code from the Free Software Foundation (FSF). For more information about the FSF, see www.fsf.org.

Other GNU language tools available from Microchip are:

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

MPLAB® IDE Simulator, Editor User’s Guide (DS51025)

Refer to this document for more information pertaining to the installation and implementation of the MPLAB Integrated Development Environment (IDE) software.

To obtain any of these documents, contact the nearest Microchip sales location (see back page) or visit the Microchip web site at: www.microchip.com.

Microsoft® Windows® Manuals

This user’s guide assumes that you are familiar with the Microsoft Windows operating system. Many excellent references exist for this software program and should be referenced for general operation of Windows.

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 and other language tools. These include the MPLAB C18 and MPLAB C30 C compilers; MPASM™ and MPLAB ASM30 assemblers; MPLINK™ and MPLAB LINK30 object linkers; and MPLIB™ and MPLAB LIB30 object librarians.
- **Emulators** – The latest information on Microchip in-circuit emulators. This includes the MPLAB ICE 2000, MPLAB ICE 4000 and MPLAB REAL ICE.
- **In-Circuit Debuggers** – The latest information on the Microchip in-circuit debugger, MPLAB ICD 2.
- **MPLAB® IDE** – The latest information on Microchip MPLAB IDE, the Windows® Integrated Development Environment for development system tools. This list is focused on the MPLAB IDE, MPLAB SIM simulator, MPLAB IDE Project Manager and general editing and debugging features.
- **Programmers** – The latest information on Microchip programmers. These include the MPLAB PM3 and PRO MATE® II device programmers and the PICSTART® Plus and PICkit™ 1 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
- Development Systems Information Line

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

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

DOCUMENT REVISION HISTORY

Revision A (December 2008)

Initial release of this document.

Chapter 1. Introduction

This chapter introduces the Real-Time Data Monitoring (RTDM) software developed for the MPLAB Data Monitor and Control Interface (DMCI), which is integrated on MPLAB IDE 8.10 or higher. These DMCI features address the need for monitoring and modifying data in a real-time fashion.

This user's guide provides information you can use to incorporate the RTDM into your embedded solution. Topics covered include:

- Overview
- Features
- System Requirements

1.1 OVERVIEW

Most existing embedded applications demand more complex and sophisticated debugging tools, providing methods to reduce the development cycle, and therefore, the time-to-market.

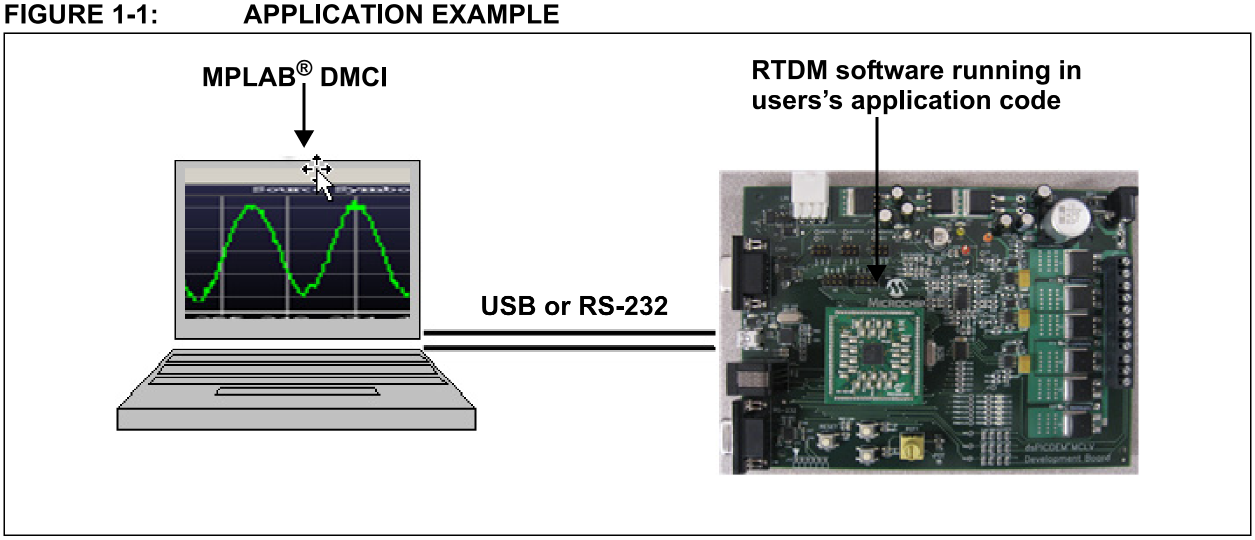
The MPLAB DMCI provides dynamic input control of application variables in MPLAB IDE projects. Application-generated data can be viewed graphically using any one of four dynamically-assignable graph windows.

Applications such as motor control and power conversion require high-speed data monitoring from MPLAB DMCI. Achieving such tasks with the existing debugging tools and the on-chip debugging module, requires the use of an additional communication link between a host PC and a target device.

RTDM, along with MPLAB DMCI (MPLAB 8.10 or higher), creates an alternative link between a host PC and a target device for debugging applications in real-time.

Using these tools for getting data in and out of the target device allows developers to run their applications, while providing the ability to tune the variables and immediately see the effect without halting the application. Figure 1-1 provides an example application.

FIGURE 1-1: APPLICATION EXAMPLE



1.2 FEATURES

RTDM has the following features:

- Runs under Debug mode or user's application
- Fully compatible with MPLAB DMCI
- Provides dynamic access to control and monitor software variables without halting program execution
- No recompiling is required between debug sessions
- Ability to control or view any global variable defined by the target application code
- Provides an alternative link to read/write data from/to the target device
- Uses the RS-232 standard protocol as the primary communication link between the host PC and target device
- Maximum baud rate: 460800 bps
- Configurable to use the UART1 or UART2 modules on the target device
- Supported by all dsPIC30F, dsPIC33F and PIC24H devices

1.3 SYSTEM REQUIREMENTS

The RTDM code requires the following resources on the target device:

- One UART module
- Flash memory used:
  - 2 Kbytes of Flash if the dsPIC Peripheral Library is used, or
  - 600 bytes if the UART driver is configured using hardware-dependent code
- RAM used: 48 bytes plus buffered data
- 1 MIPS

Chapter 2. Getting Started

This chapter describes how to run the RTDM code example, CE155, and how to add RTDM code to your application code. Topics covered include:

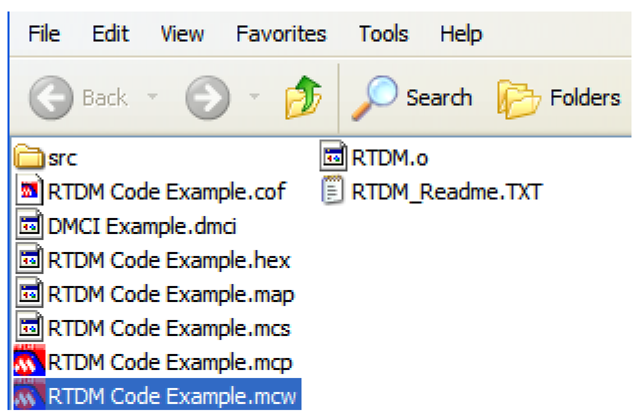
- Running the Real-Time Data Monitor Code Example CE155
- Adding the Real-Time Data Monitor to an Application
- Application Tips and Hints

2.1 RUNNING THE REAL-TIME DATA MONITOR CODE EXAMPLE CE155

This code example shows how to use RTDM to create an alternative link between a host PC and a target device for debugging applications in real-time using MPLAB DMCI (MPLAB 8.10 or higher). You can download the code example, CE155, from the Microchip web site (www.microchip.com/codeexamples). This code example runs on the dsPIC33FJ256GP710 and utilizes the Explorer 16 Development Board (DM240001).

1. Open the RTDM code example by double-clicking the file, `RTDM Code Example.mcw`, as shown in Figure 2-1.

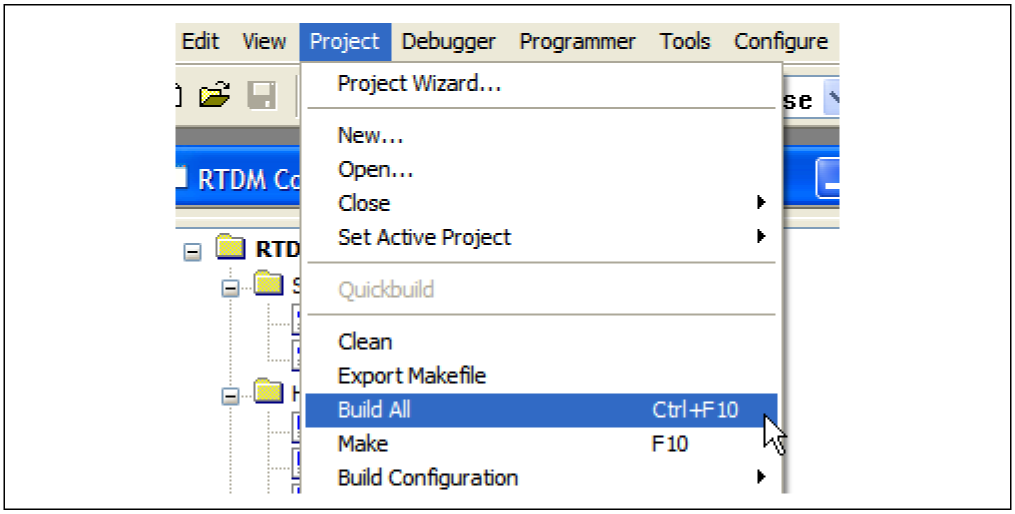
FIGURE 2-1: OPENING THE RTDM CODE EXAMPLE MPLAB® IDE WORK SPACE



2. Once the MPLAB workbench is open, compile the project by selecting <u>*Project>Build All*</u>, as shown in Figure 2-2.

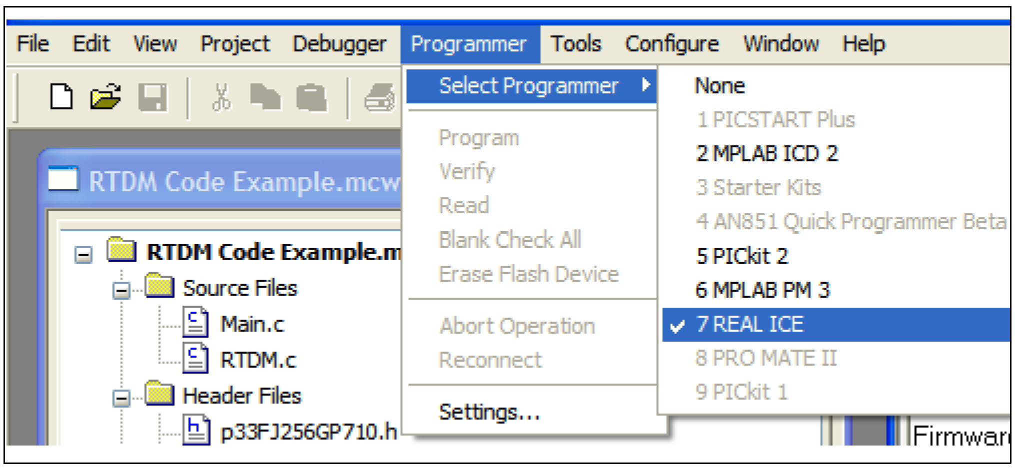
Note:	This step requires the use of the MPLAB C Compiler version 3.10 or higher.

FIGURE 2-2: COMPILING THE PROJECT



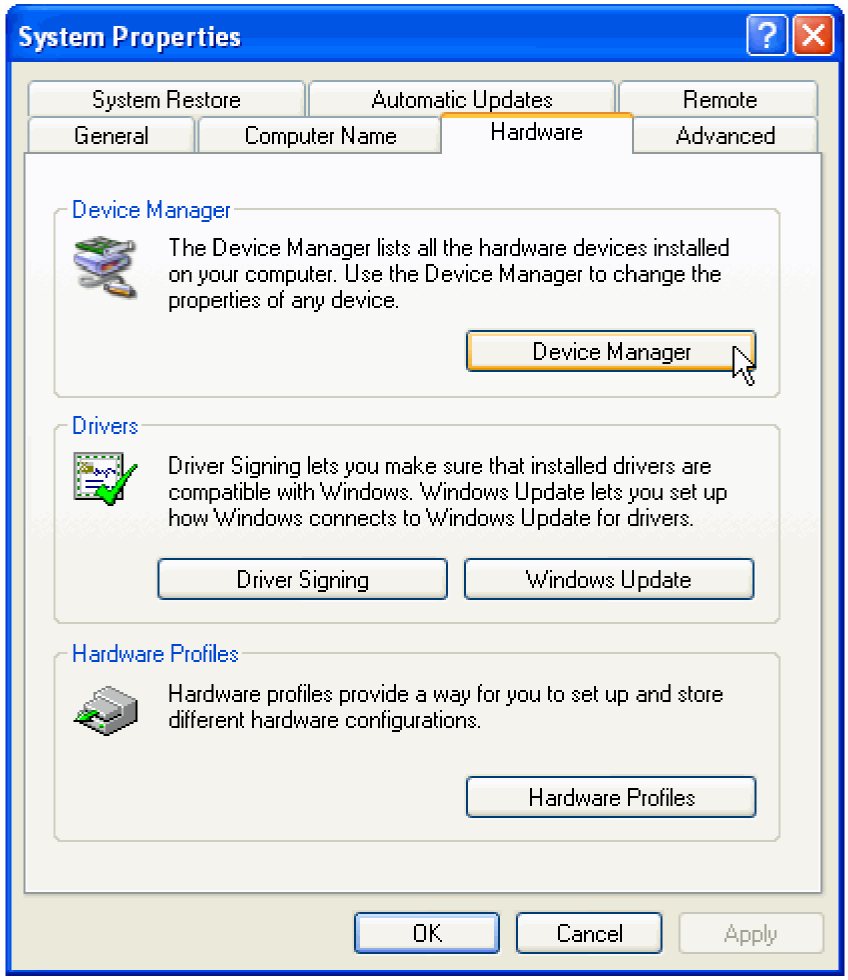
3. Select the desired programmer to program the target device. For example, to choose MPLAB® REAL ICE™, select Programmer>Select Programmer>REAL ICE, as shown in Figure 2-3. Then hit the program button or go to Programmer>program. This action will download the program to the target device.

FIGURE 2-3: SELECTING THE PROGRAMMER



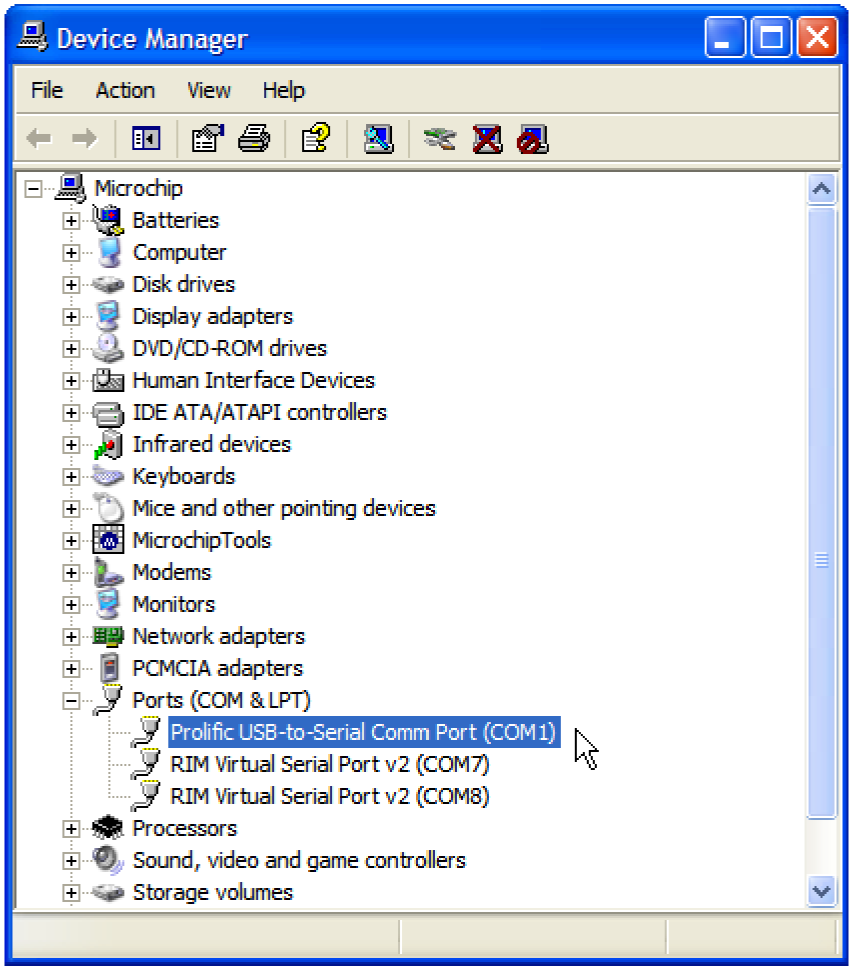
4. Connect the host PC to the target device. If your PC has a serial port, connect the PC port to the DB9 connector P1 (Explorer 16 Development Board), and then identify your COM port. If your PC does not have a serial port, you will need a USB-to-Serial adapter. Please install the USB-to-Serial adaptor USB driver before proceeding. The following steps describe the process for assigning the PC port COM number to your USB-to-Serial adaptor.
   a) Open the Windows® device manager by right-clicking the My Computer icon from your desktop and selecting **Properties**, and then selecting the **Hardware** tab, or by opening the Control Panel (Start>Settings>Control Panel) and clicking **System**. The System Properties window appears, as shown in Figure 2-4.

FIGURE 2-4: OPENING THE WINDOWS® DEVICE MANAGER



b) Click **Device Manager**, and then expand Ports (COM & LPT). Identify your serial port, as shown in Figure 2-5.

FIGURE 2-5: IDENTIFYING THE SERIAL COM PORT NUMBER



c) Make sure your COM port configuration matches the RTDM configuration, Data bits: 8, Parity: None, Stop bits: 1 and Flow Control: None, as shown in Figure 2-6.

d) If you are using MPLAB 8.10, make sure your operating system assigns the COM port 1, 2, 3 or 4 to your serial communication link. To change the COM port assignment, click **Advanced**, and then modify the COM port number as shown in Figure 2-7. Select any number from 1 to 4 and finally, restart your PC.

FIGURE 2-6: SETTING THE COM PORT PROPERTIES

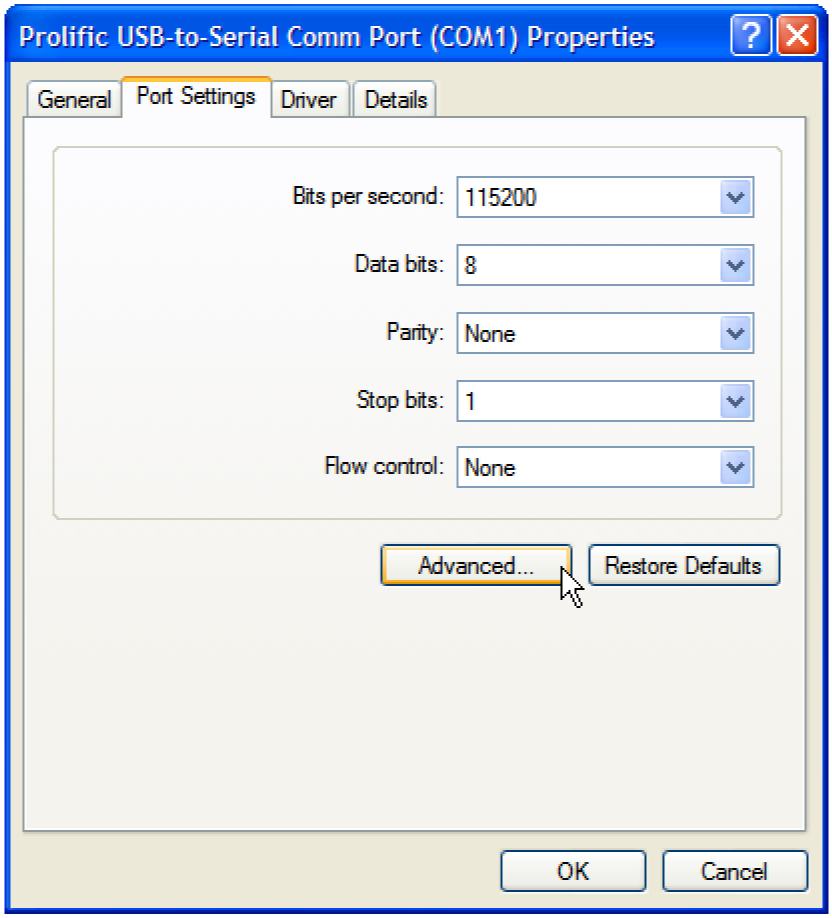
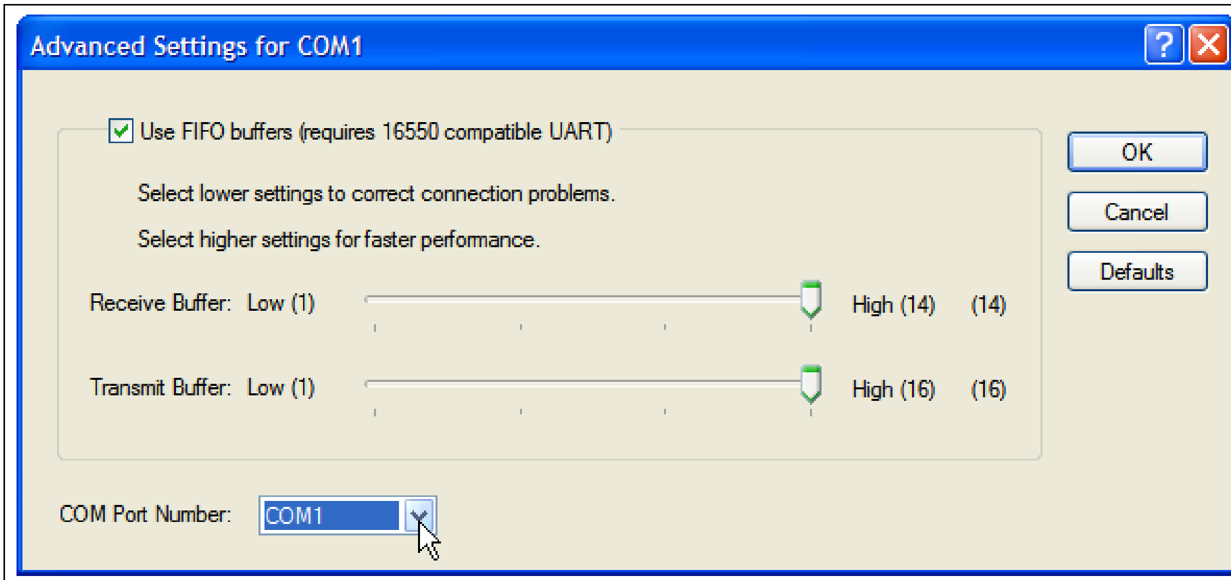


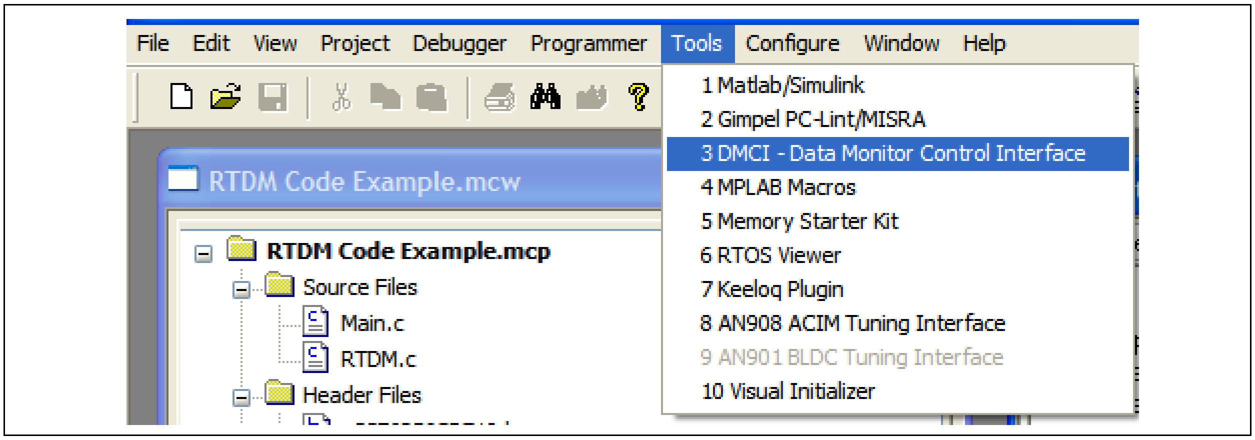
FIGURE 2-7: CHANGING THE COM PORT NUMBER



Note: In MPLAB 8.10, the DMCI only communicates through the COM1-4 ports. In MPLAB 8.14 or higher, DMCI can communicate through the COM1-COM25 ports.

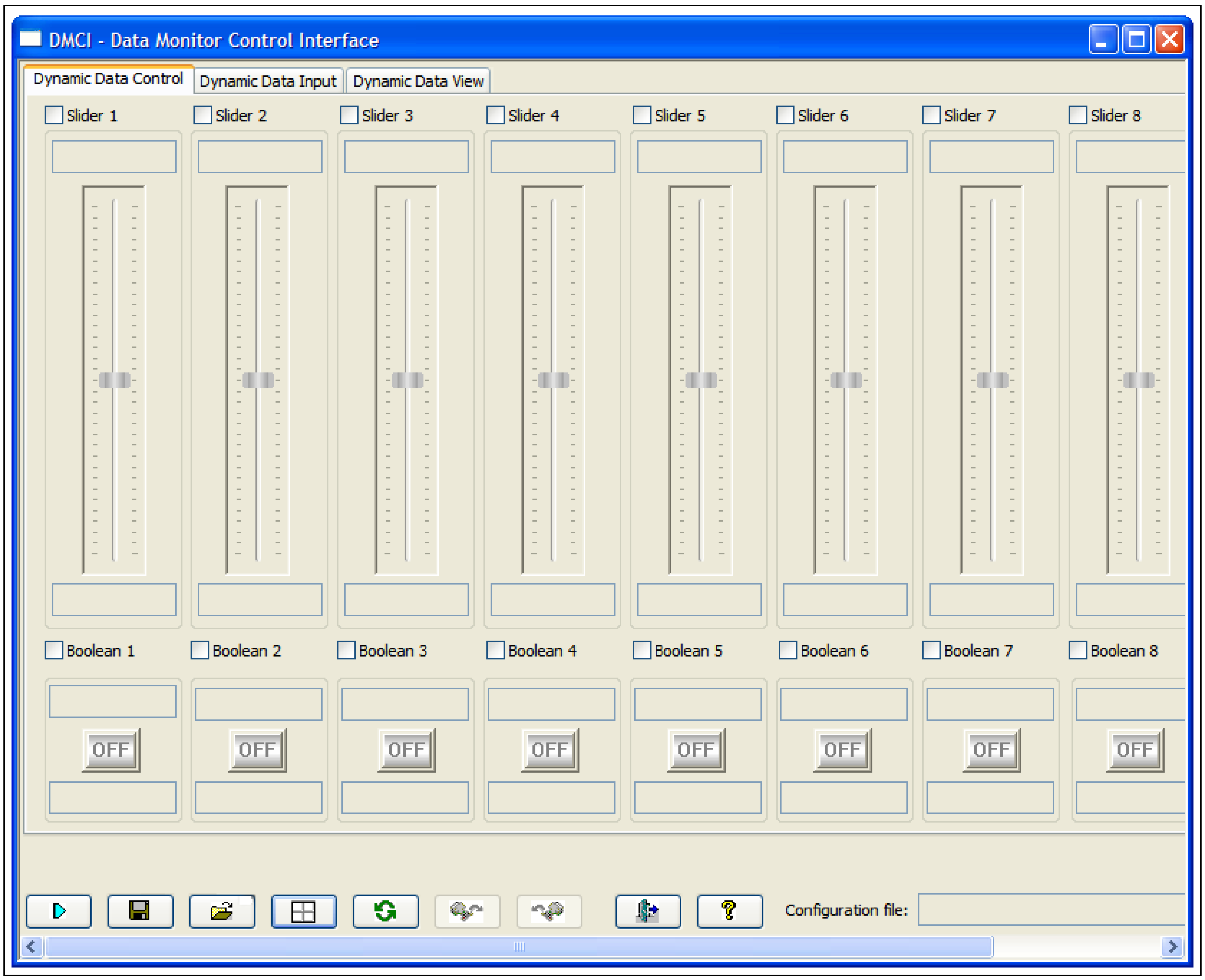
5. Open the DMCI project by selecting *Tools>DMCI - Data Monitor Control Interface*, as shown in Figure 2-8. The DMCI - Data Monitor Control Interface window appears, as shown in Figure 2-9.

FIGURE 2-8: ENABLING DMCI



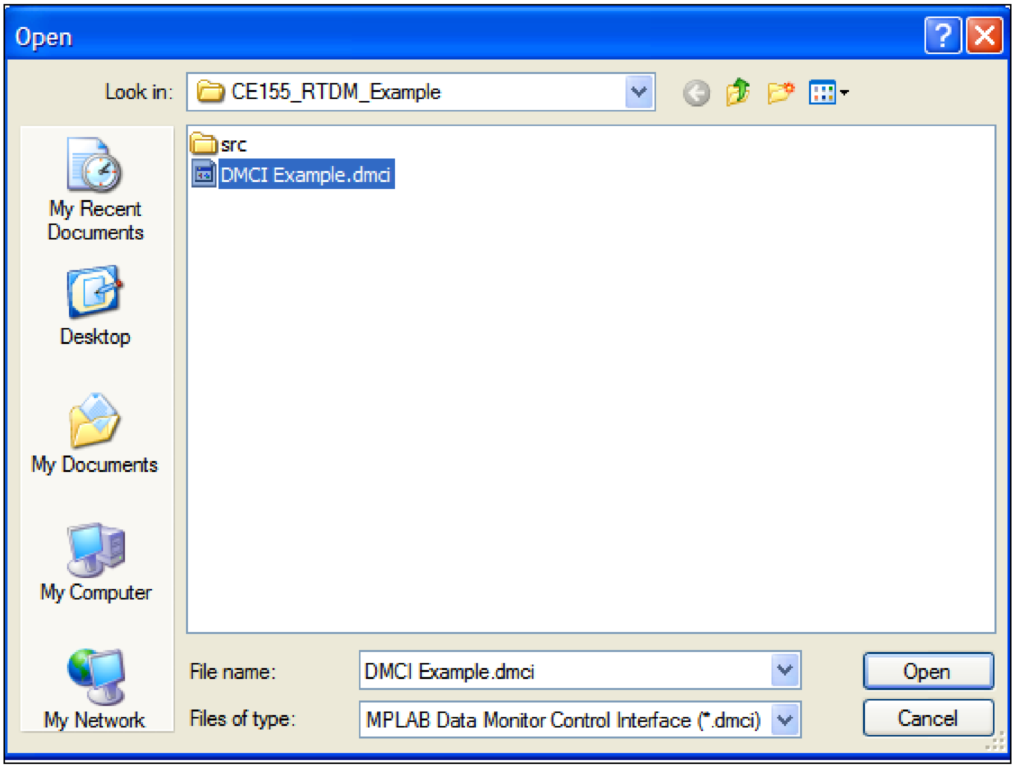
6. Click  to load the profile, as shown in Figure 2-9. The Open dialog appears, as shown in Figure 2-10.

FIGURE 2-9: BLANK DMCI PROJECT



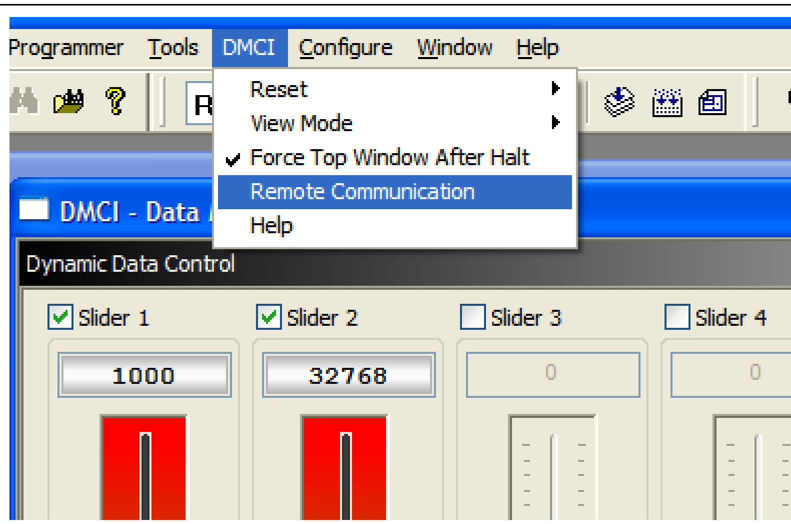
7. Double-click `DMCI Example.dmci` to open the project, as shown in Figure 2-10.

FIGURE 2-10: OPENING THE DMCI PROJECT



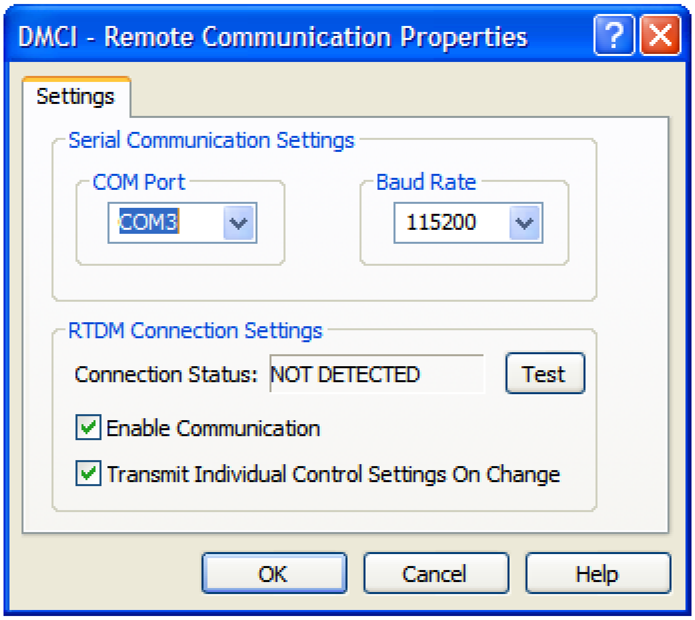
8. Configure the communication properties by selecting *DMCI>Remote Communication* from the MPLAB menu, as shown in Figure 2-11.

FIGURE 2-11: OPENING THE DMCI REMOTE COMMUNICATION PROPERTIES



a) As shown in Figure 2-12, verify the serial communication settings. Make sure that the settings match the RTDM configuration and the COM port number assigned to the serial port. The COM PORT should be set to your COM port, and the BAUD RATE should be set to 115200.

FIGURE 2-12: SETTING THE DMCI REMOTE COMMUNICATION PROPERTIES

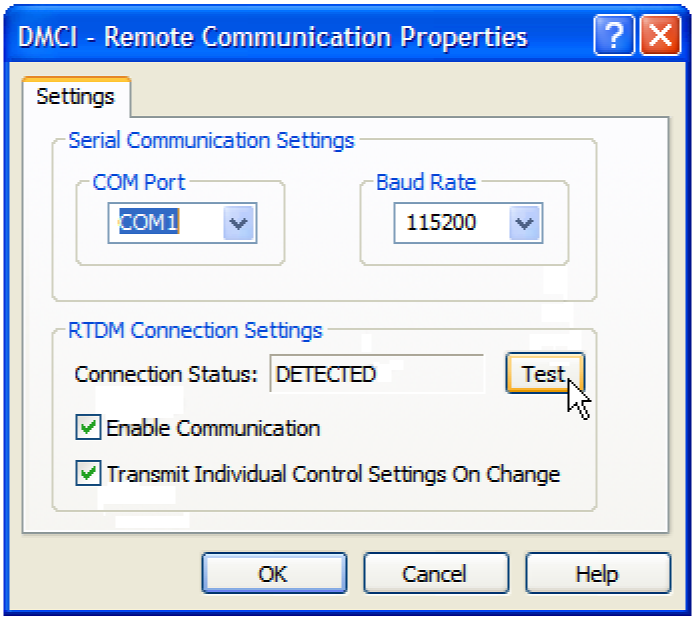


b) Click **TEST**. The DETECTED message should appear. If not, make sure that all of the RTDM settings are correct, you can verify this in the `RTDMUSER.h` file, which can be found in your MPLAB project window. Note that your device must be running for this test to work.

c) Make sure that the RTDM settings match the DMCI communication settings. Check your hardware and double check the PC communication settings again.

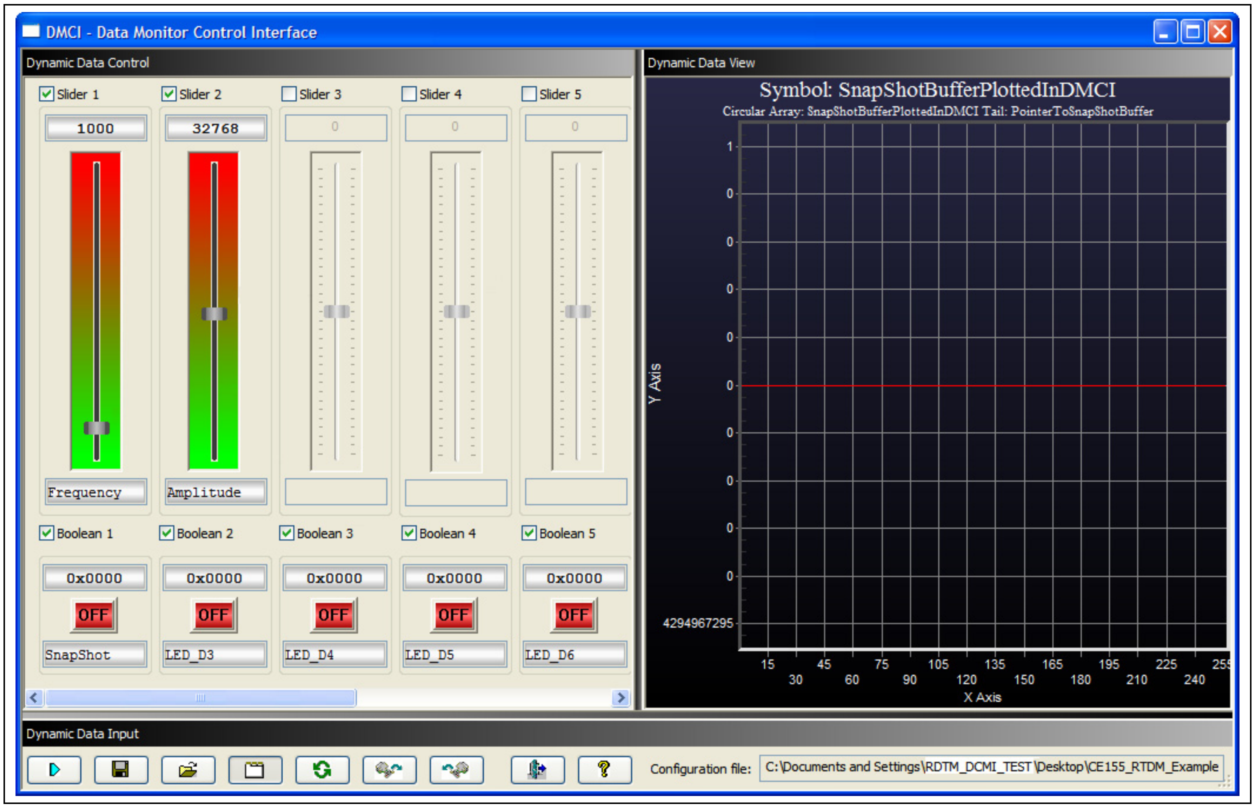
d) As shown in Figure 2-13, select the **Enable Communication Option** and **Transmit Individual Control Settings On Change** options, and then click **OK**.

FIGURE 2-13: DMCI REMOTE COMMUNICATION PROPERTIES



9. Modify and record variables at run time.
   a) In the DMCI window, click the **OFF** button above the Snapshot button to turn the Snapshot feature on. This feature will record the value of “MyVariable” into the SnapShotBuffer.
   b) Click  to update the plot.
   c) You can modify the Frequency and Amplitude by moving the sliders.
   d) You can turn the board LEDs ON or OFF by clicking the LED buttons.
   e) Any time you want to update the plot, repeat sub-steps “a” and “b”.
   f) You can save the DMCI parameters for future use by clicking .

FIGURE 2-14: MODIFYING AND RECORDING VARIABLES AT RUN TIME

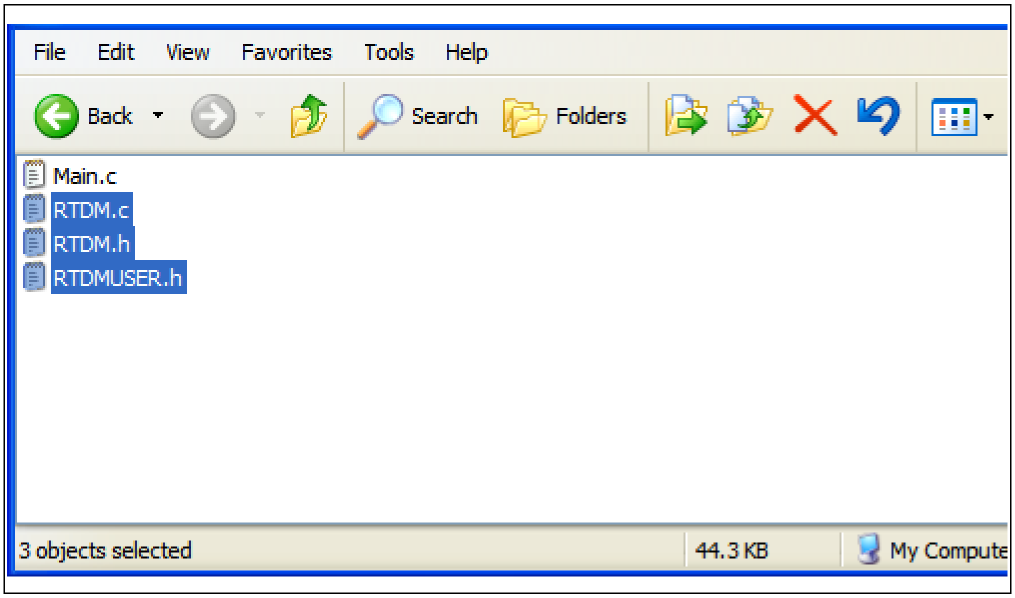


2.2 ADDING THE REAL-TIME DATA MONITOR TO AN APPLICATION

This section describes the steps required to add RTDM into a user's application code.

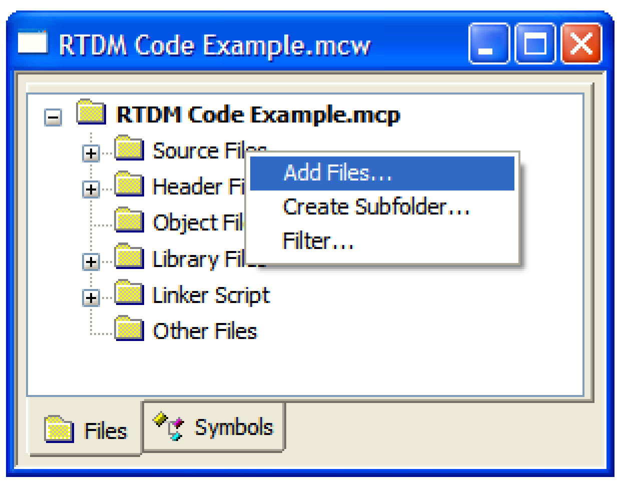
1. Copy the `RTDM.c`, `RTDM.h` and `RTDMUSER.h` files from the RTDM code example project to your project folder.

FIGURE 2-15: ADDING THE RTDM FILES TO THE APPLICATION PROJECT FOLDER



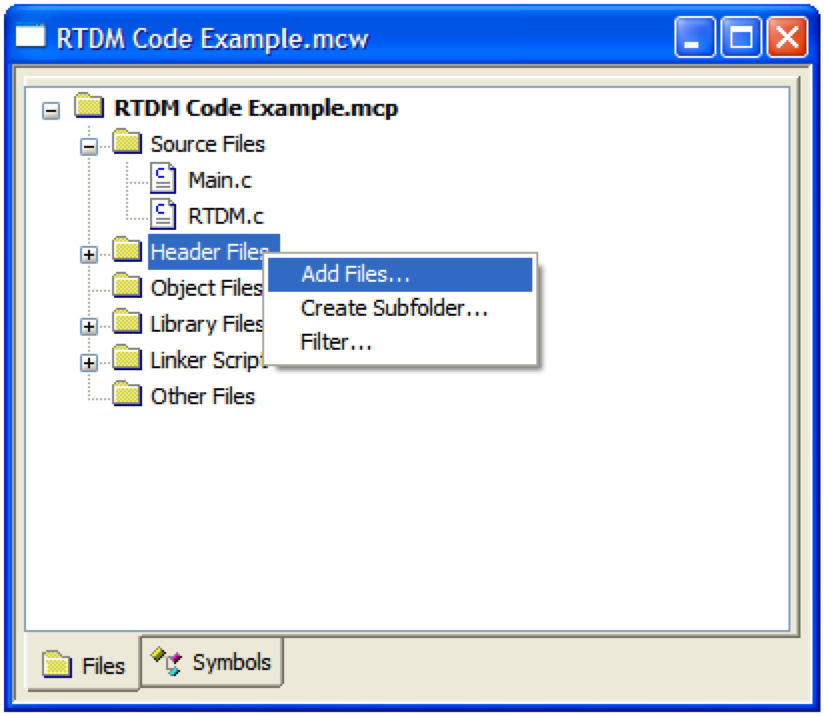
2. In your application MPLAB workspace, right-click the source file folder and add the `RTDM.c` file, as shown in Figure 2-16.

FIGURE 2-16: ADDING THE `RTDM.C` FILE TO THE MPLAB PROJECT



3. In your application MPLAB workspace, right-click the header file folder and add the `RTDM.h` and `RTDM.h` files, as shown in Figure 2-17.

FIGURE 2-17: ADDING FILES TO THE MPLAB PROJECT



4. Open the `RTDMUSER.h` file and define the values for the RTDM macros, as shown in Example 2-1.

EXAMPLE 2-1: DEFINING VALUES FOR RTDM

```
/*************************************
#define RTDM_FCY          29491200

#define RTDM_BAUDRATE     115200

#define RTDM_UART             1

#define RTDM_UART_PRIORITY    3

#define RTDM_RXBUFFERSIZE     32
#define RTDM_MAX_XMIT_LEN     0x1000

#define RTDM_POLLING          YES




#define RTDM_MIN_CODE_SIZE    YES


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

5. If the DMCI Data Viewer is activated, it is required to create a buffer for each graph in order to plot the data snapshot. Also, declare the Boolean variables required to trigger the data recorder, as shown in Example 2-2.

EXAMPLE 2-2: ALLOCATION MEMORY FOR THE DATA BUFFERS

```
/********************* Variables to display data using DMCI *****************************/
#define DATA_BUFFER_SIZE 128 //Size in 16-bit Words

//Buffer to store the data samples for the DMCI data viewer Graph1
unsigned int __attribute__((aligned(DATA_BUFFER_SIZE))) RecorderBuffer1[DATA_BUFFER_SIZE];
//Buffer to store the data samples for the DMCI data viewer Graph2
unsigned int __attribute__((aligned(DATA_BUFFER_SIZE))) RecorderBuffer2[DATA_BUFFER_SIZE];
//Buffer to store the data samples for the DMCI data viewer Graph3
unsigned int __attribute__((aligned(DATA_BUFFER_SIZE))) RecorderBuffer3[DATA_BUFFER_SIZE];
//Buffer to store the data samples for the DMCI data viewer Graph4
unsigned int __attribute__((aligned(DATA_BUFFER_SIZE))) RecorderBuffer4[DATA_BUFFER_SIZE];

unsigned int * PtrRecBuffer1 = &RecorderBuffer1[0]; //Tail pointer for the DMCI Graph1
unsigned int * PtrRecBuffer2 = &RecorderBuffer2[0]; //Tail pointer for the DMCI Graph2
unsigned int * PtrRecBuffer3 = &RecorderBuffer3[0]; //Tail pointer for the DMCI Graph3
unsigned int * PtrRecBuffer4 = &RecorderBuffer4[0]; //Tail pointer for the DMCI Graph4

//Buffer Recorder Upper Limit
unsigned int * RecBuffUpperLimit = RecorderBuffer4 + DATA_BUFFER_SIZE -1;
struct {
            unsigned StartStop :1;
            unsigned Recorder :1;
            unsigned PIControl :1;
            unsigned unused :13;
    }DMCIFlags;
```

6. Call the `RTDM_Start()` function at the beginning of your main function. If you are using a dsPIC DSC device that has a PPS module, it is also required to assign the UART port pins as shown in Example 2-3.

EXAMPLE 2-3: ADDING THE `RTDM_Start()` FUNCTION

```
/*Assigning the TX and RX pins to ports RP7 & RP9 to the dsPIC33FJ32MC204*/
/************** Code section for the low pin count devices **************/
__builtin_write_OSCCONL(OSCCON & (~(1<<6))); // clear bit 6
RPINR18bits.U1RXR = 7;  // Make Pin RP7 U1RX
RPOR4bits.RP9R = 3;     // Make Pin RP9 U1TX
__builtin_write_OSCCONL(OSCCON | (1<<6));    // Set bit 6
/****************************************************************/
RTDM_Start();  // Configure the UART module used by RTDM
               // it also initializes the RTDM variables
```


7. If the polling method is defined, add the `RTDM_ProcessMsgs()` function to the main loop, as shown in Example 2-4. Otherwise, if the interrupt method is selected, proceed to the next step.

> **Note:** In the polling method, the RTDM state machine will be called as soon as the `RTDM_ProcessMsgs()` function is called. Therefore, the incoming and outgoing messages would not be processed until this function is called. It is strongly recommended to call the `RTDM_ProcessMsgs()` function before 10 ms have elapsed.
>
> In the interrupt method, the RTDM state machine will be called as soon as an incoming message is received. If using the interrupt method, it is recommended to enable the nested interrupt method.

EXAMPLE 2-4: ADDING THE `RTDM_ProcessMsgs()` FUNCTION

```
/****************** Infinite Loop **********************************/
for(;;)
    {
    //RTDM process incoming and outgoing messages
    RTDM_ProcessMsgs();
    //Wait for START/STOP button on DMCI window to be pushed
    while(!DMCIFlags.StartStop){RTDM_ProcessMsgs();}

    T2CONbits.TON = 1;              // Start TIMER , enabling speed measurement
    PWM1CON1 = 0x0777;              // enable PWM outputs

    /*ROTOR ALIGNMENT SEQUENCE*/
    Flags.RotorAlignment = 1;       // TURN ON rotor alignment sequence
    Flags.RunMotor = 1;             // Indicating that motor is running
    CurrentPWMDutyCycle = MIN_DUTY_CYCLE;   //Init PWM values
    DesiredPWMDutyCycle = MIN_DUTY_CYCLE;   //Init PWM values
    PWMticks = 0;                   //Init Rotor aligment counter

    /************* Rotor alignment sequence and ramp-up sequence ************/
    for(RampUpCommState=1;RampUpCommState<7;RampUpCommState++)
        {
        RTDM_ProcessMsgs();
        while(++PWMticks<MAX_PWM_TICKS)
            P1OVDCON=PWM_STATE[RampUpCommState];
        PWMticks = 0;
        }
    Flags.RotorAlignment = 0;       // TURN OFF rotor alignment sequence
    PWMticks = MAX_PWM_TICKS+1;     // RAMP UP for breaking the motor IDLE state
    DelayNmSec(RAM_UP_DELAY);       // RAMP UP DELAY

    /****************** Motor is running **********************************/
    while(Flags.RunMotor)           // while motor is running
        {
        //RTDM process incoming and outgoing messages
        RTDM_ProcessMsgs();

    /****************** Stop Motor **********************************/
    //wait for START/STOP button on DMCI window to be pushed
        if (!DMCIFlags.StartStop)
            {
            PWM1CON1 = 0x0700;          // disable PWM outputs
            P1OVDCON = 0x0000;          // override PWM low.
            Flags.RotorAlignment = 0;   // turn on RAMP UP
```

8. Record the variables on the buffers, as shown in Example 2-5.

EXAMPLE 2-5: FILLING THE BUFFERS

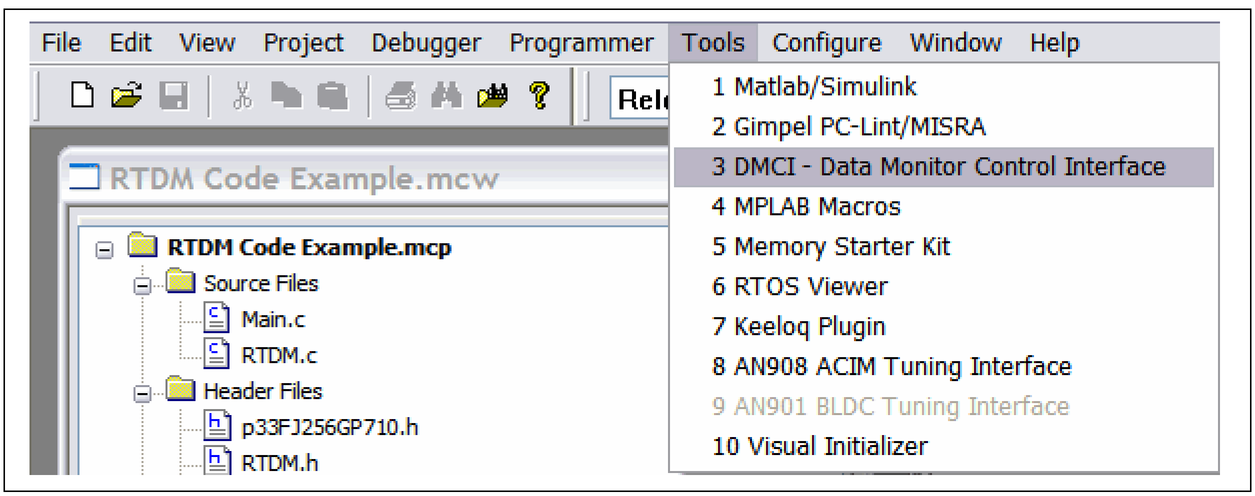
```
void __attribute__((__interrupt__,auto_psv)) _ADC1Interrupt(void)
{
    MotorPhaseA = ADC1BUF1;      //ADC CH1 holds the Phase A value
    MotorPhaseB = ADC1BUF2;      //ADC CH2 holds the Phase B value
    MotorPhaseC = ADC1BUF3;      //ADC CH3 holds the Phase C value
    //Reconstrucs Voltage at the  Motor Neutral Point
    MotorNeutralVoltage = (MotorPhaseA + MotorPhaseB + MotorPhaseC)/3;

    /********************* DMCI Dynamic Data Views  ********************/
    /********************** RECORDING MOTOR PHASE VALUES ***************/
    if(DMCIFlags.Recorder){
     *PtrRecBuffer1++    = MotorPhaseA;
     *PtrRecBuffer2++    = MotorPhaseB;
     *PtrRecBuffer3++    = MotorPhaseC;
     *PtrRecBuffer4++ = MotorPhaseB;

    if(PtrRecBuffer4 > RecBuffUpperLimit){
        PtrRecBuffer1 = RecorderBuffer1;
        PtrRecBuffer2 = RecorderBuffer2;
        PtrRecBuffer3 = RecorderBuffer3;
        PtrRecBuffer4 = RecorderBuffer4;
        DMCIFlags.Recorder = 0;
        }
    }
```

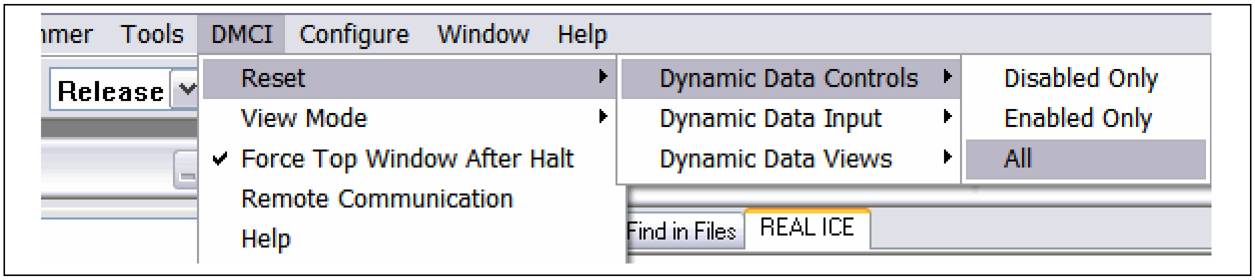
9. Create the DMCI project.
   a) Enable the DMCI tool by selecting *Tools>3 DMCI - Data Monitor Control Interface*, as shown in Figure 2-18.

FIGURE 2-18: ENABLING THE DMCI TOOL



   b) Reset the DMCI controls by selecting the following options from the MPLAB IDE menu, as shown in Figure 2-19:
   - *DMCI>Reset>Dynamic Controls>All*
   - *DMCI>Reset>Dynamic Data Input>All*
   - *DMCI>Reset>Dynamic Data Views>All*

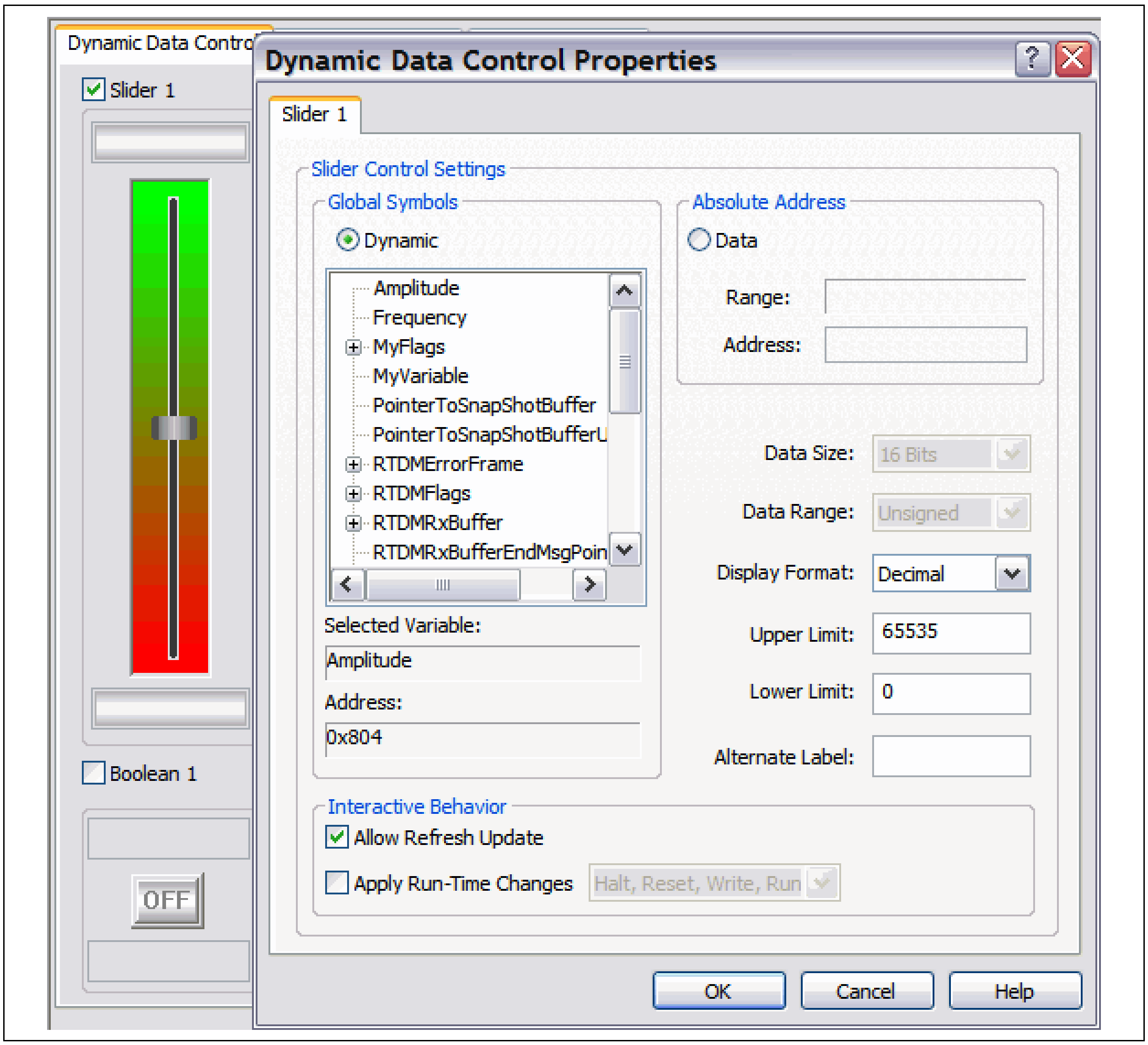
FIGURE 2-19: RESETTING THE DMCI CONTROLS



c) Add variables on the DMCI Dynamic Data Control window:

1. Enable the slider by checking the Slider box.
2. Open the Dynamic Data Control Properties menu by right-clicking the active slider (as shown in Figure 2-20).
3. In the global symbols window, select the variable to be controlled.
4. Set the variable format and the maximum and minimum levels.
5. Click **OK** to save your changes.

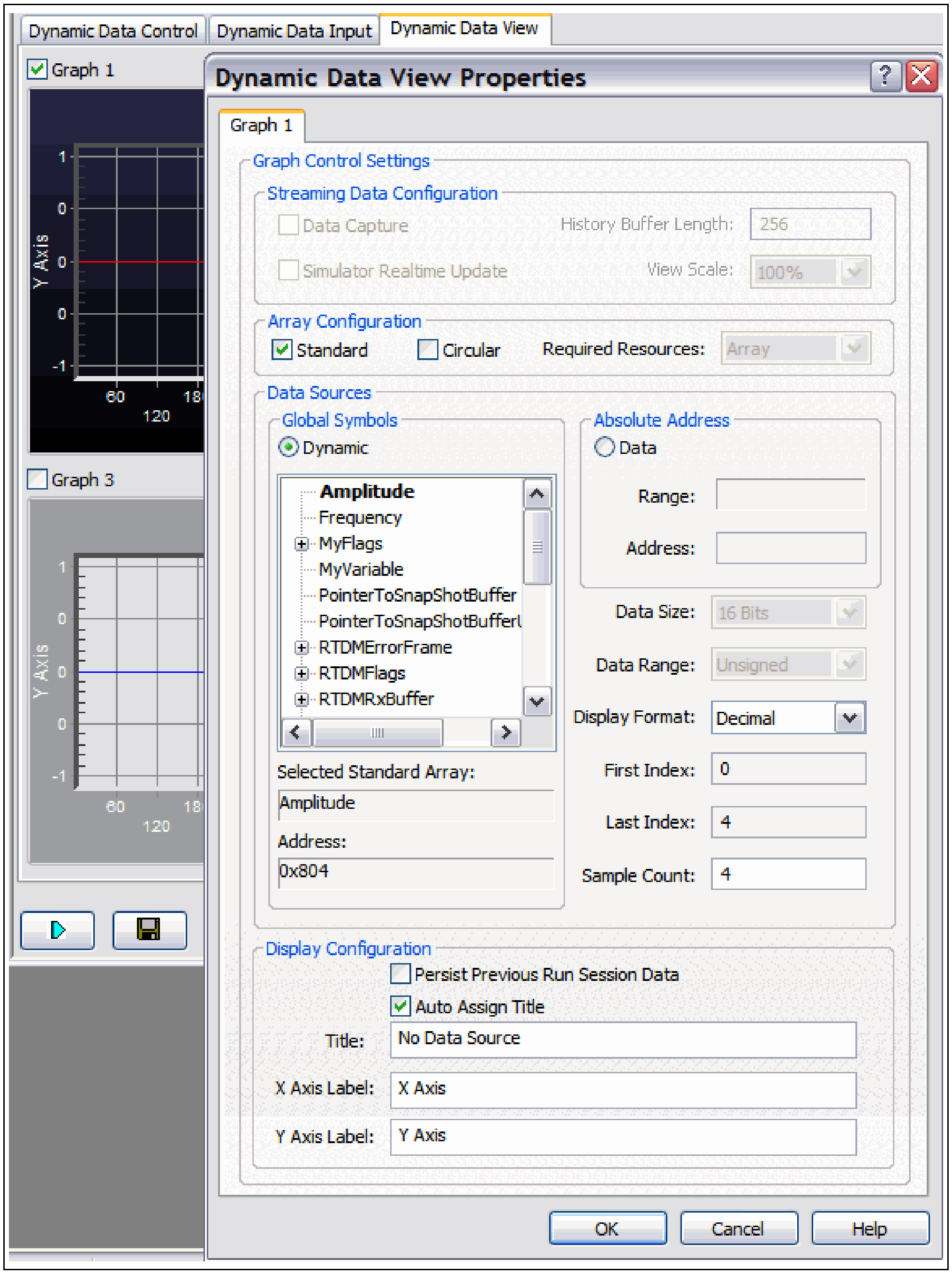
FIGURE 2-20: OPENING THE DYNAMIC DATA CONTROL PROPERTIES



d) Plot the variables in the DMCI Dynamic Data View window.
1. In the DMCI project window, select the Dynamic Data View tab.
2. Enable the graph by checking the graph box.
3. Right-click the active graph and select the configure data source option, as shown in Figure 2-21.
4. In the global symbols window, select the buffer to be plotted.
5. Click **OK** to save your changes.

Note:	It is required to compile the project before adding variables to the DMCI project. DMCI utilizes the `.map` file to obtain the variables information.

FIGURE 2-21: SELECTING THE CONFIGURE DATA SOURCE OPTION



2.3 APPLICATION TIPS AND HINTS

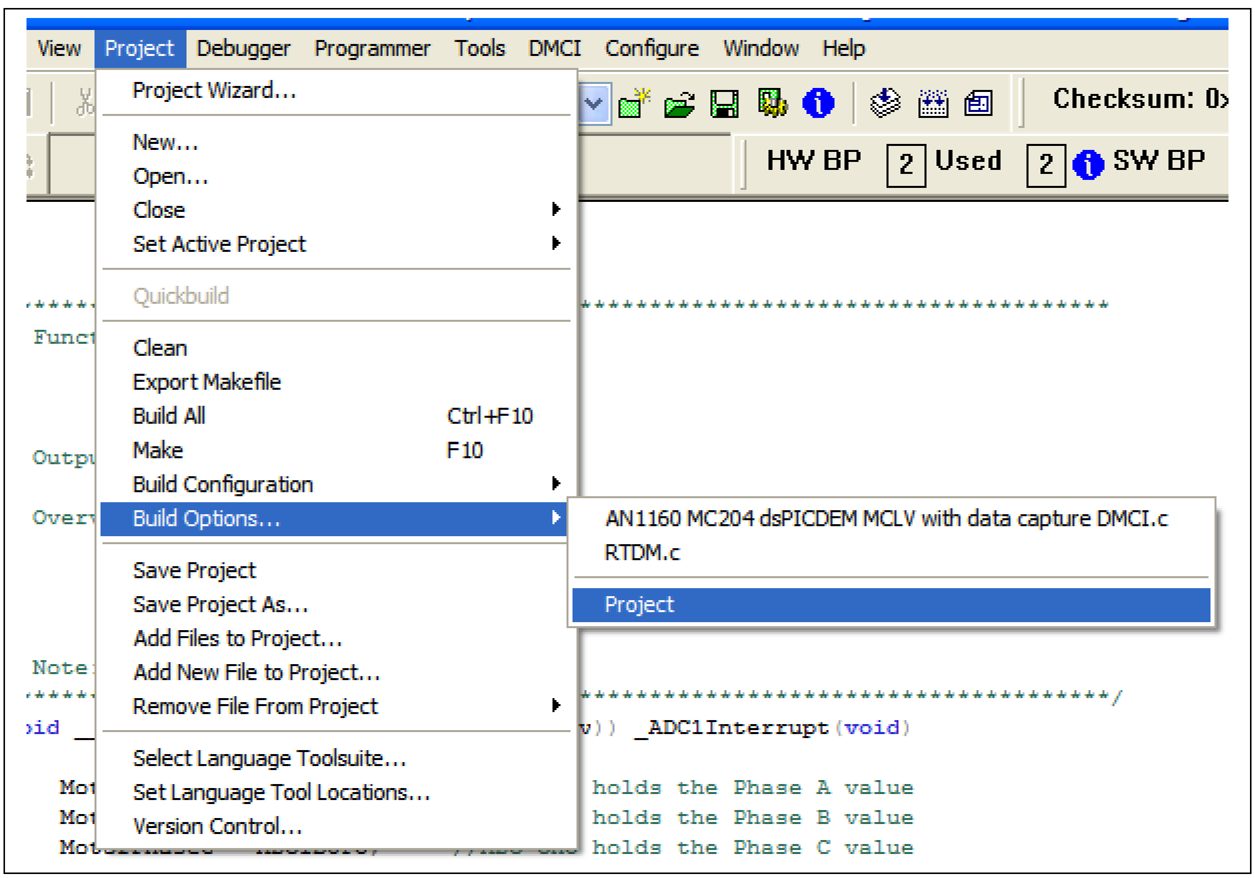
The following are a few tips to consider when using the RTDM code with DMCI.

- Do not write to the data buffer while transmitting its contents. This will corrupt the calculation of the CRC16 checksum, and therefore, the message will be discarded by the host PC. To prevent this situation, always use a triggering condition (either by software or hardware) to enable and disable writing the data buffer.
- To reduce the bit rate error, always make sure that the FCY frequency is a multiple of the selected baud rate. A 7.37 MHz oscillator provides different values that are multiples of the DMCI baud rates.
- When defining the data buffers, make sure that the compiler allocates the buffers in an unused RAM section. Use the “aligned” attribute as much as possible. For more information on this attribute, refer to the *“MPLAB® C COMPILER for PIC24 MCUs and dsPIC® DSCs User’s Guide”* (DS51284).
- To avoid user code malfunction, the priority of the UART interrupt used by the RTDM should be lower than the critical timing interrupts used by the user’s application software.
- Make sure that the memory data model is selected according to the size of the data buffers. While using the RTDM in an application, the compiler should be directed to use the most adequate data memory model according to the dsPIC DSC device and the amount of data desired to display on the DMCI data view. This is particularly useful in dsPIC DSC devices with a considerable size of RAM.

The following procedure describes the steps required to change the memory model.

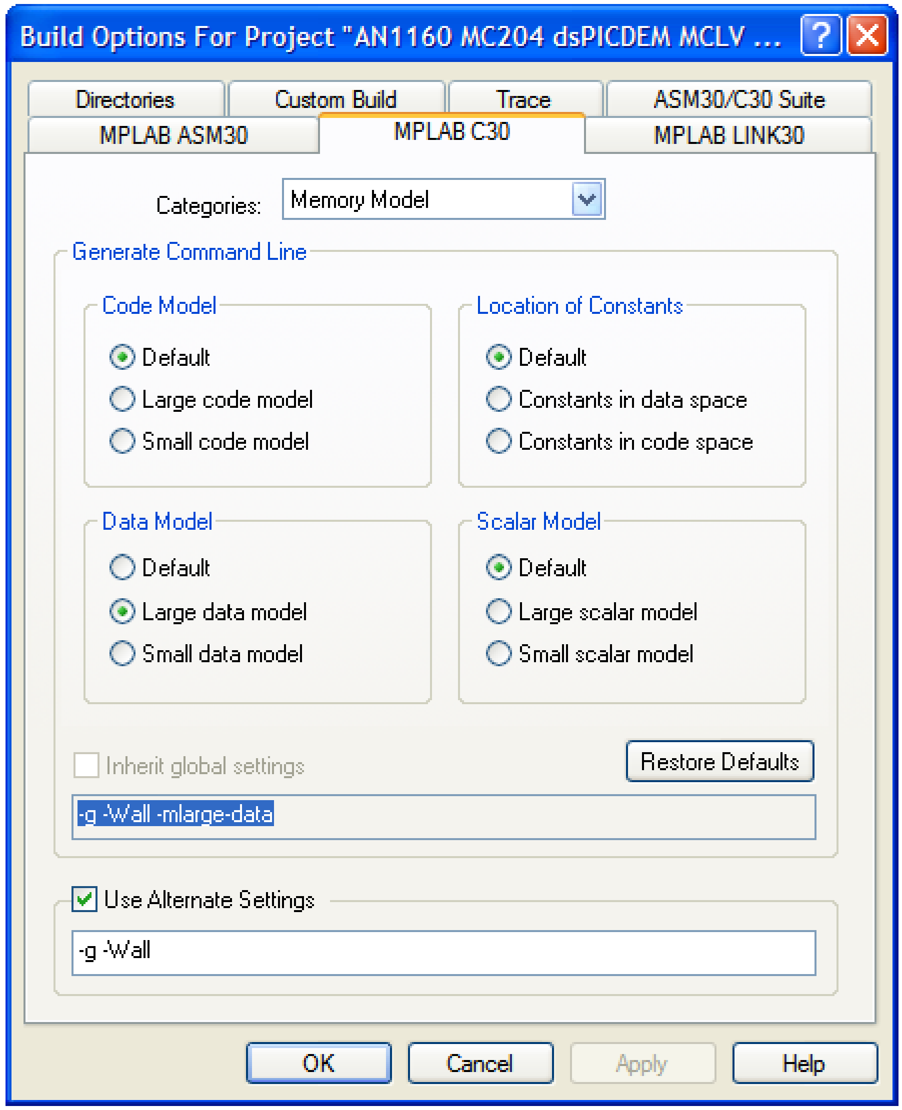
1. From the MPLAB IDE menu, select Project>Build Options>Project, as shown in Figure 2-22.

FIGURE 2-22: SELECTING PROJECT BUILD OPTIONS



2. Click the **MPLAB C30** tab and set the following options. From the Categories drop-down list, select **Memory Model**, and then select the appropriate data model according to the available RAM and the size of the RTDM data buffers. The selection is shown in Figure 2-23.

FIGURE 2-23: ASSIGNING THE MEMORY DATA MODEL



Chapter 3. Application Programming Interface (API)

This chapter describes in detail the Application Programming Interface (API) functions and constants that are available in the Real-Time Data Monitor.

3.1 API FUNCTIONS AND CONSTANTS

The functions and constants are listed below followed by their individual detailed descriptions.

- `RTDM_Start()`
- `RTDM_ProcessMsgs()`
- `CloseRTDM()`
- `RTDM_CumulativeCrc16()`
- `RTDM_FCY`
- `RTDM_BAUDRATE`
- `RTDM_UART`
- `RTDM_UART_PRIORITY`
- `RTDM_RXBUFFERSIZE`
- `RTDM_MAX_XMIT_LEN`
- `RTDM_POLLING`
- `RTDM_MIN_CODE_SIZE`

`RTDM_Start()`

Description

This function initializes the UART that is to be used to exchange data with the host PC. Some processors may have two UART modules; therefore, it is required to specify which UART module is to be used by RTDM.

Include

```
RTDM.h
RTDMUSER.h
```

Prototype

```
int RTDM_Start();
```

Arguments

None.

Return Value

0 if initialization process was successful

-1 if initialization process failed

`RTDM_ProcessMsgs()`

Description

This function processes the message received, and then executes the required task. These tasks are reading a specific memory location, writing a specific memory location, receiving a communication link sanity check command, or are querying for the size of the buffers.

Include

```
RTDM.h
RTDMUSER.h
```

Prototype

```
int RTDM_ProcessMsgs();
```

Arguments

None.

Return Values

0 if there is no message to be processed

-1 if a message is being processed

CloseRTDM()

Description

This function closes the UART used to exchange data with the host PC.

Include

```
RTDM.h
RTDMUSER.h
```

Prototype

```
int RTDM_Close();
```

Arguments

None.

Return Values

Returns '0' when the UART module was successfully closed.

RTDM_CumulativeCrc16()

Description

This function calculates the polynomial for the checksum byte. There are two approaches to calculate this number.

1. "On-the-fly" every time. Saves code space because no const table is required. This approach saves code space but yields slower throughput performance.
2. Using a coefficients table. This approach has faster performance but consumes a higher amount of program memory.

Include

```
RTDM.h
RTDMUSER.h
```

Prototype

```
unsigned int RTDM_CumulativeCrc16 (unsigned char *buf, unsigned
int u16Length, unsigned int u16CRC);
```

Arguments

`unsigned char *buf`	A pointer to the state memory for the data to be used on the checksum calculation
`unsigned int u16Length`	Number of bytes to be computed
`unsigned int u16CRC`	Polynomial value used to calculate the CRC16 checksum

Return Values

CRC16 checksum value

RTDM_FCY

Description

This constant defines the system operating frequency, whose value is used to calculate the value of the BRG register.

Value

System Frequency

RTDM_BAUDRATE

Description

This constant defines the desired baud rate for the UART module to be used by RTDM.

Value

Available options are 38400, 57600, 115200, 230400 and 460800.

> **Note:** UART modules can support higher baud rates, please refer to your transceiver specifications to check if your hardware supports higher baud rates.

RTDM_UART

Description

This constant defines the UART module to be used by RTDM. It has only two possible values: 1 or 2

Value

Depending on the dsPIC DSC device it could be 1 or 2.

RTDM_UART_PRIORITY

Description

This constant defines the UART receiver interrupt priority assigned to receive the RTDM messages.

Value

1 to 7

RTDM_RXBUFFERSIZE

Description

This constant defines the buffer size used by RTDM to handle messages.

Value

32

RTDM_MAX_XMIT_LEN

Description

This constant defines the size in bytes of the maximum number of bytes allowed in the RTDM protocol frame.

Value

4096

RTDM_POLLING

Description

This constant defines the mode that the RTDM will be operating in the user's application. If it is set to YES, the user should place the `RTDM_ProcessMsgs()` function in the main loop. In order to make sure that the messages are being processed, it is recommended that the main loop always polls this function as fast as possible (minimum 10 ms).

If it is set to NO, the `RTDM_ProcessMsgs()` function will be called on the UART receiver Interrupt Service Routine (ISR). If multiple interrupts are enabled, it is required to activate the nested interrupts mode on the dsPIC DSC device.

Value

YES or NO

`RTDM_MIN_CODE_SIZE`

Description

This constant defines the Cyclic Redundancy Check (CRC) algorithm calculation minimum code size. If it is set to YES, the RTDM library will be built including a precalculated polynomial table for the CRC algorithm. This method reduces the device CPU throughput required to calculate the CRC16 checksum.

If it is set to NO, the CRC16 value is calculated from scratch and the predefined polynomial values would not be loaded into the program memory. This mode saves 768 bytes of code, but requires more device CPU throughput to calculate the CRC16 checksum.

Value

YES or NO

Chapter 4. Protocol

This chapter describes the RTDM protocol specification, which is used to debug dsPIC DSC devices and PIC24H embedded applications at run-time. The RS-232 protocol utilized by the RTDM is a set of simple binary structures and conventions, enabling a data exchange between a personal computer and a target device. The serial frame definition is: 1 start bit, 8 data bits, 1 parity bit and 1 stop bit. The maximum serial transfer rate is 460800 bps. Topics covered include:

- The Protocol Model
- Commands
- ERROR Code
- Files

4.1 THE PROTOCOL MODEL

The communication model is based on the GDB Remote Serial Protocol, which is a single master-slave protocol with some modifications according to the nature of the application. This protocol is based on a basic principle – the host PC sends a message with a command and its arguments, and then the target replies with the operation status code and return data. The target device never initiates communication; its replies are specified and always have a known fixed length.

All GDB commands and responses are sent as a packet. A packet is introduced with the character '$', the actual packet-data, and the terminating character '#' followed by a two-digit checksum. For example:

```
$packet-data#checksum
```

The two-digit checksum is a CRC16 calculation of all characters between the leading '$' and the trailing '#'. When either the host or the target device receives a packet, the first response expected is an acknowledgment: either '+' (to indicate the package was received correctly) or '-' (to request retransmission).

The host sends commands, and the target (RTDM incorporated in your program) sends a response. In the case of step and continue commands, the response is only sent when the operation has completed. The packet-data consists of a sequence of characters with the exception of '#' and '$'. These characters are the commands and the data required to execute those commands.

4.2 COMMANDS

The command codes supported by the RTDM protocol are listed in Table 4-1.

TABLE 4-1: COMMANDS SUPPORTED BY RTDM

Function	Data Access	ASCII Code	Code (Hex value)
Read Memory	16-bit word	`m`	6D
Write Memory	16-bit word	`M`	4D
Check Communication Link Sanity	16-bit word	`s`	73

4.2.1 Read Memory Command

This function code is used to read from 1 to 65536 contiguous RAM locations on the target device. The Host Request specifies the starting register address and the number of memory locations formatted in little-endian format. The structure of this command is shown in Table 4-2 and Table 4-3.

TABLE 4-2: HOST PC REQUEST

Frame Arrangements	ASCII Code	Size in Frame (Bytes)	Possible Values
Start Code	`$`	1	0x24
Function Code	`m`	1	0x6D
Starting Address in little-endian format	`[XXXXXXXX]`	4	0x00000000-0xFFFFFFFF
Number Of Bytes in little-endian format (N)	`[XXXX]`	2	0x0000-0xFFFF
End Of Message Code	`#`	1	0x23
CRC16L	`[XX]`	1	0x00-0xFF
CRC16H	`[XX]`	1	0x00-0xFF

TABLE 4-3: TARGET DEVICE REPLY

Frame Arrangements	ASCII Code	Size in Frame (Bytes)	Possible Values
Reply Code	`+`	1	0x2B
Start Code	`$`	1	0x24
Memory Values in little-endian format	`[XX]…. [XX]`	Nx2	0x00-0xFF
End Of Message Code	`#`	1	0x23
CRC16L	`[XX]`	1	0x00-0xFF
CRC16H	`[XX]`	1	0x00-0xFF

An example of reading the 16-bit memory location 0x00001234 is shown in Example 4-1.

EXAMPLE 4-1:

Host PC Command:

```
$m432100002000#[CRC16]
```

Target Device Reply:

```
+$EFBE#[CRC16]
```

4.2.2 Write Memory

This function code is used to write a block of contiguous RAM locations (1 to 65536 registers) on the target device. The requested written values are specified in the packet-data field. The structure of this command is shown in Table 4-4 and Table 4-5.

TABLE 4-4: HOST PC REQUEST

Frame Arrangements	ASCII Code	Size in Frame (Bytes)	Possible Values
Start Code	`$`	1	0x24
Function Code	`M`	1	0x4D
Starting Address in little-endian format	`[XXXXXXXX]`	4	0x00000000-0xFFFFFFFF
Number Of Bytes in little-endian format (N)	`[XXXX]`	2	0x0000-0xFFFF
Values to be written in little-endian format	`[XX]`	Nx2	0x00-0xFF
End Of Message Code	`#`	1	0x23
CRC16L	`[XX]`	1	0x00-0xFF
CRC16H	`[XX]`	1	0x00-0xFF

TABLE 4-5: TARGET DEVICE REPLY

Frame Arrangements	ASCII Code	Size in Frame (Bytes)	Possible Values
Reply Code	`+`	1	0x2B
Start Code	`$`	1	0x24
Command Acknowledge	`O`	1	0x4F
Command Acknowledge	`K`	1	0x4B
End Of Message Code	`#`	1	0x23
CRC16L	`[XX]`	1	0x00-0xFF
CRC16H	`[XX]`	1	0x00-0xFF

An example of writing 0xBEEF value to the memory location 0x00001234 is shown in Example 4-2.

EXAMPLE 4-2:

Host PC Command:

```
$M43210000EFBE#[CRC16]
```

Target Device Reply:

```
+$OK#[CRC16]
```

4.2.3 Communication Link Sanity Check Command

This command is used to check the link status and verify that there is a target device attached. The structure of this command is shown in Table 4-6 and Table 4-7.

TABLE 4-6: HOST PC REQUEST

Frame Arrangements	ASCII Code	Size in Frame (Bytes)	Possible Values
Start Code	`$`	1	0x24
Function Code	`s`	1	0x73
End Of Message Code	`#`	1	0x23
CRC16L	`[XX]`	1	0x00-0xFF
CRC16H	`[XX]`	1	0x00-0xFF

TABLE 4-7: TARGET DEVICE REPLY

Frame Arrangements	ASCII Code	Size in Frame (Bytes)	Possible Values
Reply Code	`+`	1	0x2B
Start Code	`$`	1	0x24
Memory Values	`RTDM`	4	0x5254444D
End Of Message Code	`#`	1	0x23
CRC16L	`[XX]`	1	0x00-0xFF
CRC16H	`[XX]`	1	0x00-0xFF

An example of link sanity request and response is shown in Example 4-3.

EXAMPLE 4-3:

Host PC Command:

```
$s#[CRC16]
```

Target Device Reply:

```
+$RTDM#[CRC16]
```

4.3 ERROR CODE

When the host PC sends a request to the target device it expects a normal response. One of four possible events can occur from the master's query:

- If the target device receives the request without a communication error, and can handle the query normally, it returns a normal response.
- If the target device does not receive the request due to a communication error, no response is returned. The host PC program will eventually process a time-out condition for the request.
- If the target device receives the request, but detects a communication error (parity, CRC, etc.), no response is returned. The Host PC program will eventually process a time-out condition (after 10 ms have elapsed) for the request.
- If the target device receives the request without a communication error, but cannot handle it (for example, if the request is to read a nonexistent output or register), the target device will return an exception response informing to the host PC the nature of the error.

In a normal response, the target device replies with an "OK" command. In an exception response, the target device replies with the "E" command plus the Error code. Table 4-8 shows the possible error code and its meaning.

TABLE 4-8: ERROR CODE

Code	Name	Meaning
01	Illegal Function	The function code received in the query is not an allowable action for the target device. It could also indicate that the target device is in the wrong state to process a request of this type, for example, because it has not been configured and is being asked to return register values.

An example of an error reply when a command is not supported is shown in Example 4-4.

EXAMPLE 4-4:

```
-$E01#[CRC16]
```

4.3.1 Cyclic Redundancy Check (CRC) Generation

The CRC field is two bytes, containing a 16-bit binary value. The CRC value is calculated by the transmitting device, which appends the CRC to the message. The device that receives, recalculates a CRC during receipt of the message, and compares the calculated value to the actual value it received in the CRC field. If the two CRC values are not equal, an error is generated.

The CRC is started by first preloading a 16-bit register to all '1's. Then, a process begins of applying successive 8-bit bytes of the message to the current contents of the register. Only the eight bits of data in each character are used for generating the CRC. Start and stop bits and the parity bit, do not apply to the CRC.

During generation of the CRC, each 8-bit character is exclusive ORed with the register contents. Then, the result is shifted in the direction of the Least Significant bit (LSb), with a zero filled into the Most Significant bit (MSb) position. The LSb is extracted and examined. If the LSb was a '1', the register is then exclusive ORed with a preset, fixed value. If the LSb was a '0', no exclusive OR takes place.

This process is repeated until eight shifts have been performed. After the last (eighth) shift, the next 8-bit character is exclusive ORed with the register's current value, and the process repeats for eight more shifts as described above. The final content of the register, after all the characters of the message have been applied, is the CRC value.

4.3.1.1 CALCULATING CRC

The steps for calculating a CRC are as follows:

1. Load a 16-bit register with FFFF hex (all '1's). Call this the CRC register.
2. Exclusive OR the first 8-bit byte of the message with the low-order byte of the 16-bit CRC register, putting the result in the CRC register.
3. Shift the CRC register one bit to the right (toward the LSb), zero-filling the MSb. Extract and examine the LSb.
4. If the LSb is '0', repeat Step 3 (another shift). Otherwise, if the LSB is '1', Exclusive OR the CRC register with the polynomial value 0xA001 (1010 0000 0000 0001).
5. Repeat steps 3 and 4 until eight shifts have been performed. When this is done, a complete 8-bit byte will have been processed.
6. Repeat steps 2 through 5 for the next 8-bit byte of the message. Continue doing this until all bytes have been processed.
7. The final content of the CRC register is the CRC value.
8. When the CRC is placed into the message, its upper and lower bytes must be swapped.

4.4 FILES

The RTDM code is contained in three files: `RTDM.c`, `RTDM.h`, and `RTDMUSER.h`. In addition, RTDM also requires the dsPIC Peripheral Library file (`libpic30-coff.a`). This library provides the UART initialization routines, the send buffer routines, and the transmit buffer routines.

The `RTDM.c` file contains the RTDM source code and defines the state machine and functions required to receive and send commands to and from the host.

The `RTDM.h` file contains the RTDM function definitions. It also calculates the baud rate deviation for the selected target device system frequency (F_{CY}). When the delta between the F_{CY} and the selected `RTDM_BAUDRATE` is higher by 2% or lower than 2%, the compiler generates an error message.

The `RTDMUSER.h` file defines the RTDM operational mode. This file sets the communication baud rate, the UART module to be used, the UART receiver interrupt priority, the command-reception buffer size, the maximum number of bytes to be sent, the form RTDM state machine is called (polled or interrupt-based), and the CRC16 calculation method.

NOTES:

Chapter 5. DMCI Operating Modes

This chapter describes three different DMCI operating modes. Topics covered include:

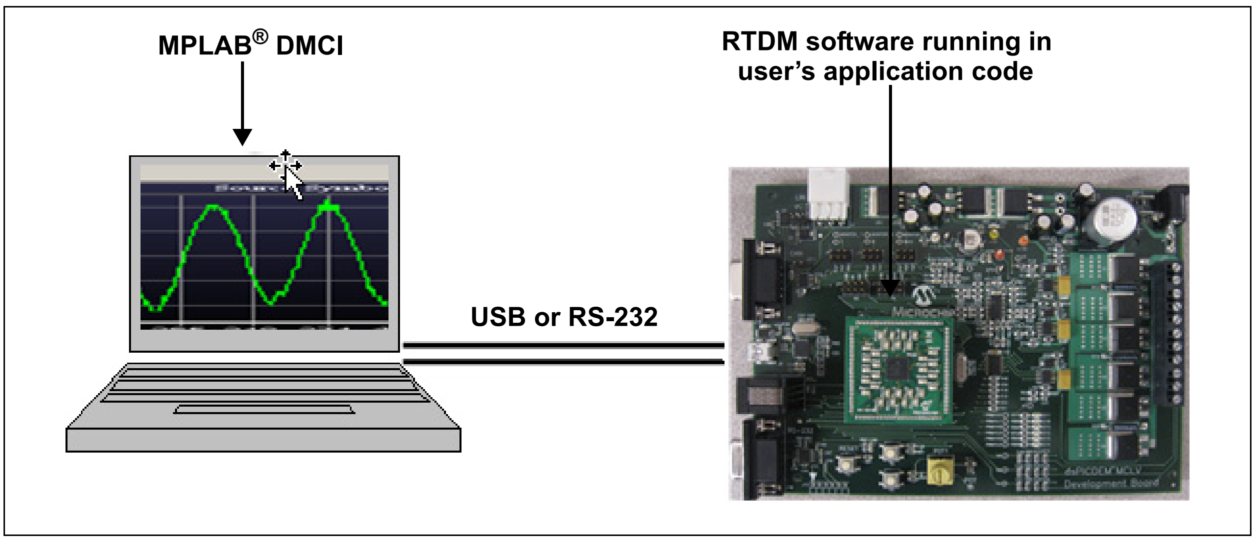
- RTDM Mode
- Data Capture Mode
- Combine Mode

5.1 RTDM MODE

This DMCI mode acquires and modifies data at run time using the alternative communication link explained in **Chapter 1. "Introduction"**. An application example is shown in Figure 5-1.

The steps required to implement this mode are explained in **Section 2.1 "Running the Real-Time Data Monitor Code Example CE155"** and **Section 2.2 "Adding the Real-Time Data Monitor to an Application"**.

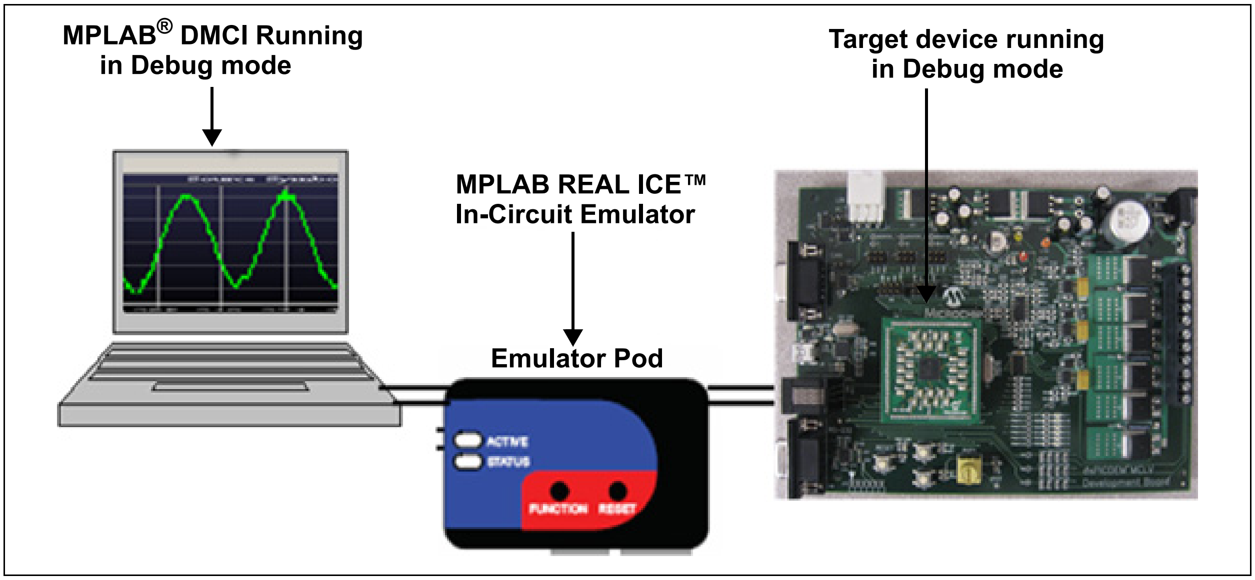
FIGURE 5-1: RTDM-BASED APPLICATION EXAMPLE



5.2 DATA CAPTURE MODE

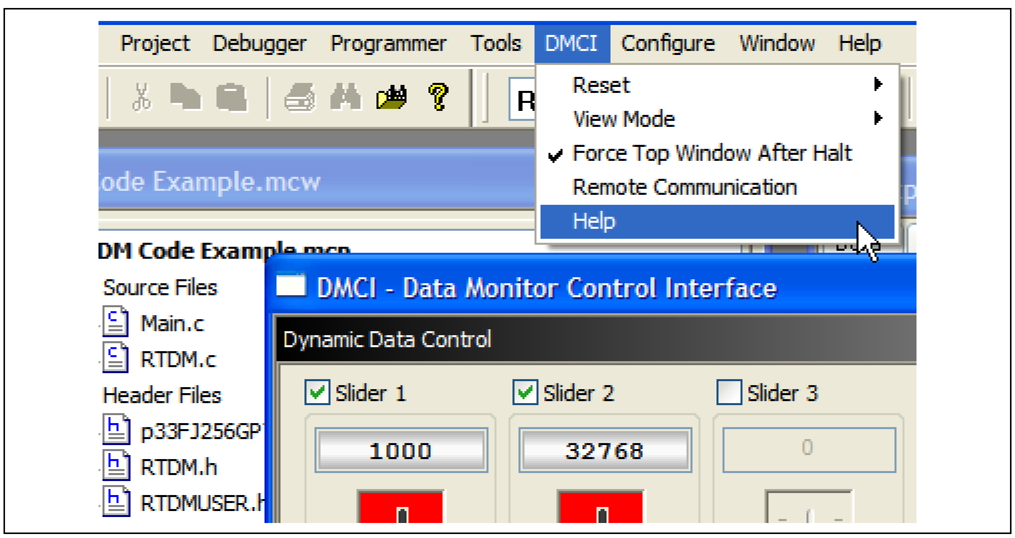
This mode utilizes the MPLAB REAL ICE In-Circuit Emulator and the on-chip debugging module (ICSP™) to read data from RAM. It acquires data as fast as 50 μs and periodically displays the acquired data using the DMCI dynamic data view graphs. It was developed for reading RAM contents without halting execution. This DMCI mode only runs under Debug mode. The number of variables that can be plotted using this method depend on the number of hardware breakpoints available on the target device. An application example is shown in Figure 5-2.

FIGURE 5-2: DATA CAPTURE APPLICATION EXAMPLE



For more information on how to implement this mode, refer to the DMCI Help, as shown in Figure 5-3.

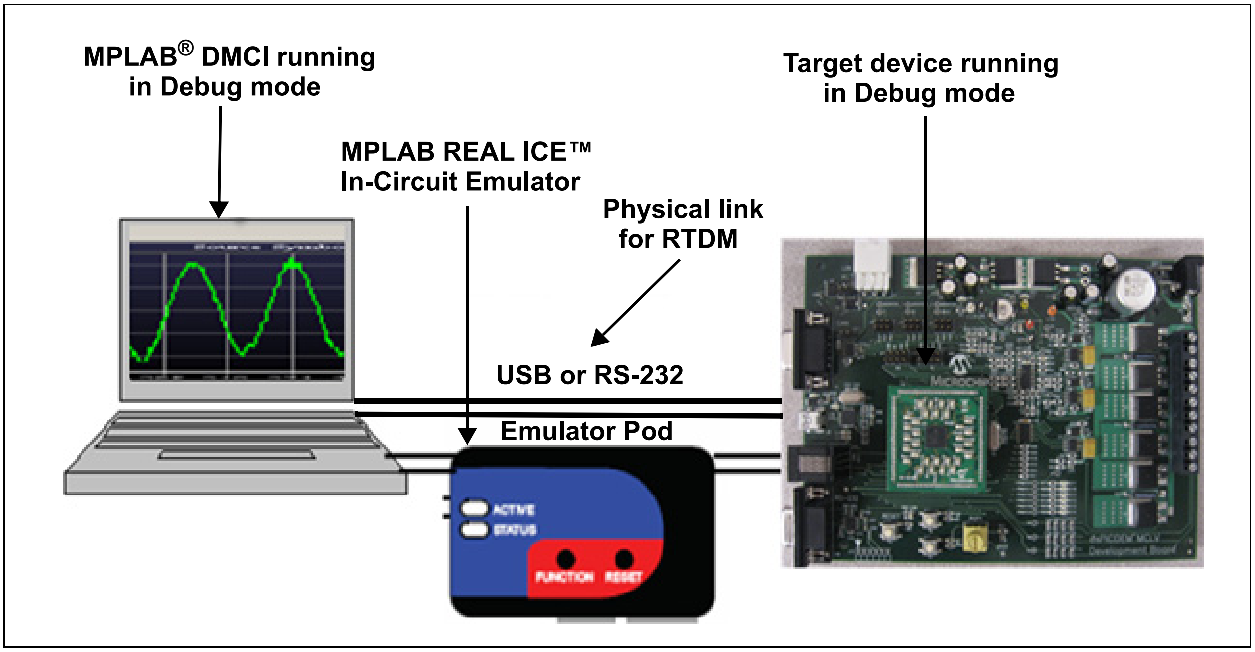
FIGURE 5-3: OPENING THE DMCI HELP FILE



5.3 COMBINE MODE

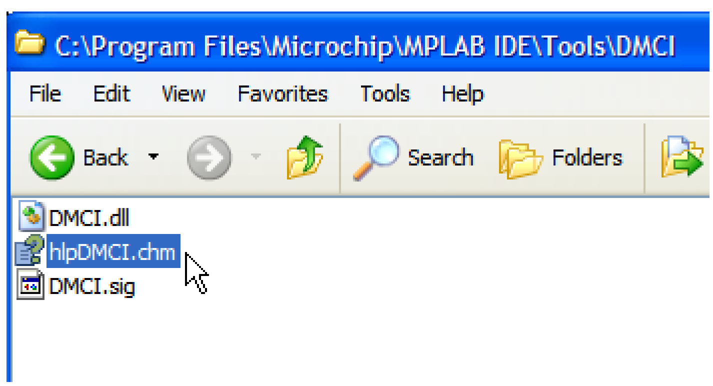
This mode combines the Data Capture and the RTDM running under Debug mode. It utilizes the MPLAB REAL ICE In-Circuit Emulator and the on-chip debugging module (ICSP) to continuously read data from the RAM. It uses RTDM to read and write variables that are updated faster than 50 µs. An application example is shown in Figure 5-4.

FIGURE 5-4: COMBINED MODE APPLICATION EXAMPLE



For more information on how to add the data capture mode to the RTDM mode, refer to the DMCI Help. Figure 5-5 shows the location of the compiled Help file.

FIGURE 5-5: DMCI HELP FILE PATH



NOTES:

Index

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