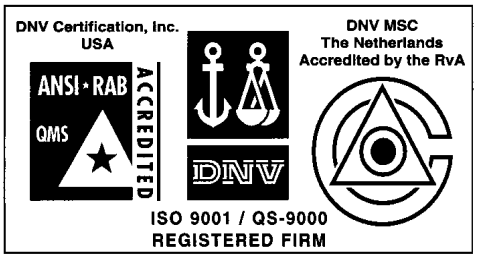
PICDEM™ 17 DEMONSTRATION BOARD USER'S GUIDE

Trademarks



Microchip received QS-9000 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona in July 1999. The Company's quality system processes and procedures are QS-9000 compliant for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs and microperipheral products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001 certified.

PICDEM™ 17 DEMONSTRATION BOARD USER'S GUIDE

Table of Contents

PICDEM™ 17 DEMONSTRATION BOARD USER'S GUIDE

PREFACE

Welcome

Thank you for purchasing the PICDEM 17 product demonstration board for the PIC17C7XX family of microcontrollers from Microchip Technology Incorporated. The PICDEM 17 demonstration board allows you to quickly and easily become familiar with both the PIC17C7XX products and the MPLAB® ICE in-circuit emulator. The PICDEM 17 demonstration board currently supports all 68-pin and 84-pin PLCC products.

The software provided with the PICDEM 17 demonstration board for the PIC17C7XX Monitor runs under Microsoft® Windows® 95 only.

Documentation Layout

This document describes the PICDEM 17 demonstration board. A detailed description of the demonstration software is also provided to give the user an overview of the PIC17C7XX series of PICmicro® MCUs. Detailed usage of the microcontrollers, MPLAB IDE, or MPLAB ICE in-circuit emulator are deferred to the individual product data sheets and user's manuals, respectively.

Chapter 1: Introduction – This chapter introduces the PICDEM 17 demonstration board and provides a brief description of the hardware.

Chapter 2: Using the PICDEM 17 Monitor – This chapter discusses how to use the PICDEM 17 Monitor PC program and the pre-programmed PIC17C756A device.

Chapter 3: Hardware Description – This chapter describes in detail the hardware of the PICDEM 17 demonstration board.

Chapter 4: Using MPLAB C17 C compiler with PICDEM 17 demonstration board – This chapter provides a description of how to write and compile code for execution on the PICDEM 17 demonstration board. Some special files are required for use with the MPLAB C17 C compiler to correctly compile programs to run in the external FLASH program memory.

Chapter 5: `LCD.C` Description – This chapter provides a detailed description of the demonstration program for the PIC17C756A that interfaces to an external LCD panel using an Hitachi HD44780 display controller or equivalent.

Chapter 6: `USART.C` Description – This chapter provides a detailed description of the demonstration program for the PIC17C756A that displays data to the USART.

Chapter 7: `ANALOG.C` Description – This chapter provides a detailed description of the demonstration program for the PIC17C756A that reads an A/D channel and displays the results in the Monitor program window.

Chapter 8: `SWITCH.C` Description – This chapter provides a detailed description of the memory mapping of the PIC17C756A and provides source code to interface to the memory mapped switches and LEDs.

Chapter 9: `I2C.C` Description – This chapter provides a detailed description of the demonstration program for the PIC17C756A that reads and writes data from a 24LC01B Serial EEPROM.

Appendix A: PICDEM 17 Demonstration Board Schematics – This appendix provides the PICDEM 17 demonstration board parts layout diagram and the board schematics.

Appendix B: RS-232 Communication Protocol – This appendix provides the protocol for the PIC17C756A Monitor firmware to communicate to the PC based software.

Appendix C: Floppy Disk Contents – This appendix contains a listing of all files on the included 3.5-inch floppy disk. There is also a description as to the use of each file when compiling programs for the PICDEM 17 demonstration board and the PIC17C7XX microcontrollers.

Appendix D: On-line Support – This appendix provides information on Microchip's electronic support services.

Worldwide Sales & Service – This reference gives the address, telephone, and fax numbers for Microchip Technology Incorporated sales and service locations throughout the world.

Chapter 1. About PICDEM 17 Demonstration Board

1.1 Introduction

This chapter describes the features of the PICDEM 17 demonstration board.

1.2 Highlights

This chapter covers the following topics:

- **Processor Sockets**
- **External FLASH Memory**
- **Memory Mapping**
- **Power Supply**
- **Prototyping Areas**
- **Oscillator Options**
- **RS-232 Serial Ports**
- **Push-button Switches**
- **LEDs**
- **Analog Circuitry**
- **External LCD Interface**
- **CAN Bus Interface**
- **24LC01B Serial EEPROM**
- **Modular Connectors**
- **Pre-programmed Sample**

Note: All following part references can be found in Figure A.1.1 in *Appendix A: PICDEM 17 Demonstration Board Schematics*. For example, the 68-pin PLCC socket for the PIC17C75X microcontrollers is located at U1 on the Parts Layout.

1.3 Processor Sockets

The PICDEM 17 demonstration board supports the following devices:

- 68-pin PLCC socket for the PIC17C75X microcontrollers (U1)
- 84-pin PLCC socket for the PIC17C76X microcontrollers (U7)

1.4 External FLASH Memory

The PICDEM 17 demonstration board supports all operating modes of the PIC17C7XX PICmicro® microcontroller. However, the Monitor program uses the extended microcontroller mode of operation. The FLASH is the AM29F100T device from AMD configured as 64K x 16. The monitor program supports downloading code into the FLASH and then running the program.

1.5 Memory Mapping

Since the PIC17C756A is in the extended microcontroller mode, there are several memory mapped peripherals available to the user. The first is the eight push-button switches mapped at address FFFCh. The eight LEDs are mapped at address FFFDh. There are also two signals that provide decoding for an address of 8 (LE_1) and an address of 16 (LE_2). These signals are located in the digital prototyping area in the upper right-hand corner of the board.

1.6 Power Supply

The PICDEM 17 demonstration board provides a different power supply structure to the user. The only power input capable of powering the entire board is at the connector J1. Any power supply with a 2.1 mm plug capable of delivering +9V, up to 1A, unregulated Alternating Current (AC) or Direct Current (DC) can be used. The digital components on the board are powered from a LM2940T-5.0 that is capable of supplying 1A of current. This digital +5V is available in the digital prototyping area in the upper right-hand corner of the board.

The analog section of the PICDEM 17 demonstration board has a separate power source and voltage reference. A LM78L05 +5V regulator provides 150 mA at +5V for the analog circuits. It is also available in the analog prototyping area in the lower left-hand corner of the board. There is also a precision +4.096V reference that is connected to the (Analog-to-Digital) A/D as well as the analog prototyping area.

1.7 Prototyping Areas

The PICDEM 17 demonstration board provides two prototyping areas, one for analog and one for digital. The PICDEM 17 demonstration board was designed to fully demonstrate the capabilities of the on-chip A/D converter. In addition to the separate analog power supply and voltage reference, the printed circuit board has four layers with two signal layers and a power and ground layer. The analog prototyping area has all A/D channels, analog V_{DD} and V_{SS}, as well as the voltage reference available to the user. The digital prototyping area has all other I/O pins, complete 16-bit address bus, digital V_{DD} and V_{SS}, and additional memory mapped peripheral signals.

1.8 Oscillator Options

The PICDEM 17 demonstration board layout will only accept a canned oscillator. The use of a crystal or ceramic resonator requires that the user modify the board.

1.9 RS-232 Serial Ports

The PICDEM 17 demonstration board supports both USARTs on the PIC17C7XX devices. A level shifting IC is used to convert from the TTL/CMOS levels out of the PIC17C7XX to the RS-232 voltages. USART2 also has the capability to function in a hardware handshaking mode using RTS and CTS. These signals are level shifted and connected to PORTB pins 4 (CTS) and 5 (RTS).

1.10 Push-button Switches

The PICDEM 17 demonstration board has 8 general purpose push-button switches that are available to the user. These switches are not connected directly to I/O pins on the microcontroller but are memory mapped into the address space. The switches are available at address FFFCh.

1.11 LEDs

The PICDEM 17 demonstration board also has 8 LEDs which are also memory mapped. These LEDs may be accessed at address FFFDh. There is no read capability for the LEDs, so the user must have a shadow register in the PIC17C7XX device to keep track of the value on the latch.

1.12 Analog Circuitry

In addition to the power and ground planes on the PICDEM 17 demonstration board, it also has separate analog and digital power supplies. Both of these voltage regulators use connector J1 as the source. If the user desires to power the board from the test points in the prototyping areas, then both sets of test points in the analog and digital must be used. Otherwise one portion of the board will not be operational.

1.13 External LCD Interface

The PICDEM 17 demonstration board provides an interface to an external LCD display that uses the Hitachi HD4478 LCD controller or equivalent device. The LCD is used in the 4-bit interface mode. Four data lines and three control lines are required to operate the LCD. These LCD signals are multiplexed with some of the A/D converter channels. It is therefore important that the A/D is properly initialized before the LCD can be used (refer to Chapter 4). The board layout provides space for the Optrex DMC-50448N 8x2 character display which is available from DigiKey.

1.14 CAN Bus Interface

The PICDEM 17 demonstration board provides a simple interface to a CAN Bus. The Microchip Technology MCP2510 CAN Interface peripheral device with the Phillips PCA82C250 device provide a complete CAN Interface solution. The two modular connectors J4 and J5 allow easy interface to the bus. The canned oscillator socket O2 provides the clock source for the MCP2510 which in turn can also provide the clock source for the PIC17C7XX.

1.15 24LC01B Serial EEPROM

The PICDEM 17 demonstration board has a 24LC01B Serial EEPROM capable of holding 128 bytes of data. This device uses the I^2C™ interface on the PIC17C7XX PICmicro MCU. The modular connectors J4 and J5 can also be used to create an I^2C interface to a peripheral not located on the board.

1.16 Modular Connectors

The modular connectors J4 and J5 are used to connect to external busses. The DIP switch S14 is used to determine if the CAN Bus interface is used or the I^2C interface. If the CAN Bus is the desired interface, the user will have to populate the board with the MCP2510 CAN Bus peripheral from Microchip Technology and the PCA82C250 CAN Bus interface device from Philips Semiconductor. Since the MCP2510 uses the SPI™ interface, DIP switch S13 must be configured for SPI (SCK,SDO,SDI,CS).

1.17 Pre-programmed Sample

A pre-programmed PIC17C756 or PIC17C756A sample is included with the PICDEM 17 demonstration board. It has been programmed with the Monitor firmware to communicate with the Monitor program on the PC. This device should be used with the 16 MHz canned oscillator in the socket labeled O1.

Chapter 2. Using the PICDEM 17 Monitor

2.1 Introduction

This chapter discusses how to use the Monitor program to interact with the PIC17C7XX device on the PICDEM 17 demonstration board.

2.2 Highlights

This chapter includes:

- **Installing the PICDEM 17 Monitor Program**
- **Running the PICDEM 17 Monitor Program**
- **Resetting the PICDEM 17 Demonstration Board**
- **Erasing the FLASH on the PICDEM 17 Demonstration Board**
- **Downloading HEX Files to PICDEM 17 Demonstration Board**
- **Running HEX Files from the PICDEM 17 Demonstration Board**
- **Running Diagnostics on the PICDEM 17 Demonstration Board**
- **Modifying Memory Contents**

2.3 Installing the PICDEM 17 Monitor Program

The PICDEM 17 Monitor program setup routine installs the monitor program, PC monitor source code and the PIC17C756A monitor source code into the MPLAB® IDE directory. Microsoft® Windows® 95 must be running to execute the Monitor Setup program. The files can be installed under the default MPLAB IDE directory, C:\Program Files\MPLAB\756MON, or in another directory.

1. Insert the PICDEM 17 Monitor installation disk in drive A:.
2. From the Program Manager Run option, type A**:SETUP**
   The PICDEM 17 Monitor Setup program displays a Welcome! message box with options to continue or exit. Click **OK** to continue.
3. Setup next displays a dialog to select the directory to install the executable and source code files. If directory other than the default is desired, enter the name and click **OK**.
4. After copying the PICDEM 17 Monitor files, Setup displays a message box with the caption "The PICDEM 17 Monitor installation has completed." Click **OK**.

2.4 Using the PICDEM 17 Monitor Program

The Monitor program uses a serial port on the Host PC. It currently supports COM1 through COM4. The command line to execute the Monitor program should be:

756MON.EXE COM?

where **?** is the desired COM port 1, 2, 3, or 4. The PICDEM 17 Monitor program should look like Figure 2.1.

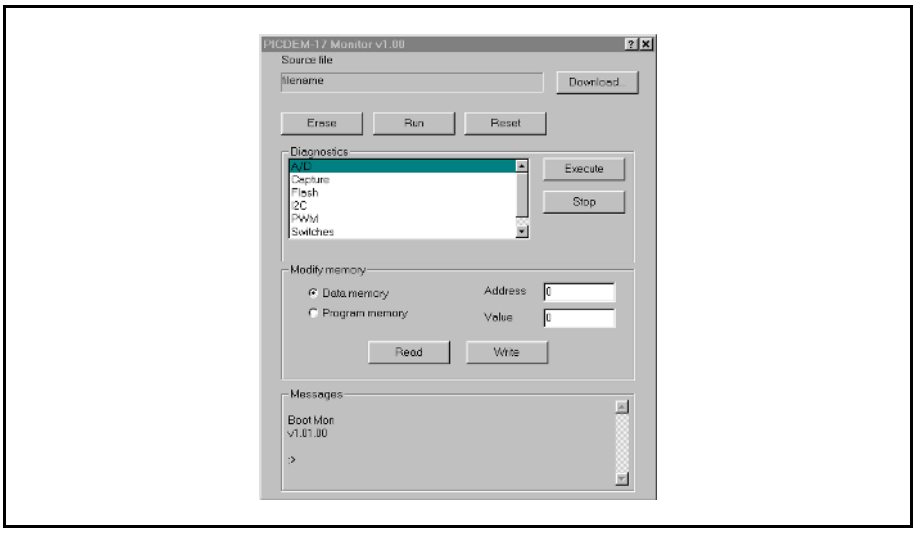


Figure 2.1: PICDEM 17 Monitor Program

Once the Monitor program has been started, connect the serial cable to P2 on the PICDEM 17 demonstration board. Then apply power to the PICDEM 17 demonstration board. The PICDEM 17 demonstration board should display the following message in the message window on the Monitor Program:

Boot Mon
V??.??.??
:>

where **??.??.??** is the version of the Monitor firmware in the PIC17C756A. The Monitor program has many features including:

- Resetting the PIC17C756A on the PICDEM 17 demonstration board
- Erasing the FLASH on the PICDEM 17 demonstration board
- Downloading HEX files to the PICDEM 17 demonstration board
- Running and halting the operation of programs
- Running diagnostics on the various PIC17C756A peripherals
- Reading and writing to Data Memory and external FLASH program memory
- Displaying USART2 activity in message window

2.5 Resetting the PICDEM 17 Demonstration Board

One unique feature of USART2 is the ability to reset the PICmicro® MCU. This is accomplished by connecting the DTR signal from P2 to the MCLR pin of the PICmicro MCU using the circuit in Figure 2.4. Jumper J3 is also provided to disable this feature. To reset the microcontroller, simply click on **Reset** in the Monitor window. The boot message should display on the message area.

2.6 Erasing the FLASH on the PICDEM 17 Demonstration Board

The Monitor program allows the external FLASH program memory device to be erased by clicking on the **Erase** button. This simply erases the FLASH memory. When a download is requested, the PIC17C756A automatically erases the FLASH. Therefore, it is not required that the FLASH be erased using the **Erase** button before a download is initiated.

2.7 Downloading HEX Files to the PICDEM 17 Demonstration Board

To download a HEX file to the PICDEM 17 demonstration board, first click on **Download…**. This will open a Select HEX file window that allows the selection of the desired HEX file to load into the external FLASH program memory on the PICDEM 17.

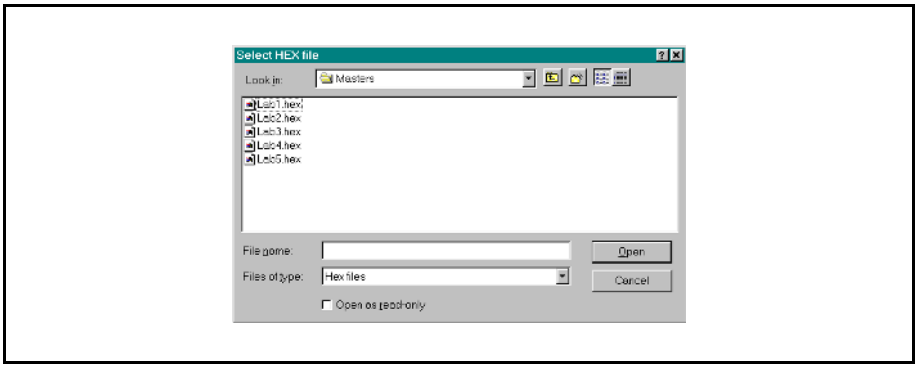


Figure 2.2: Select HEX file Window

Once a HEX file has been selected for downloading to the PICDEM 17 demonstration board, the Monitor will show a **Downloading…** status window that has a bar graph display of percentage download of the file. Initially, there will be a slight pause allowing the PIC17C756A to erase the FLASH memory.

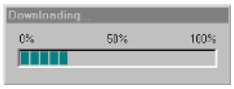


Figure 2.3: Downloading… Window

At the completion of the download, the Monitor status window will have displayed a **Erasure Complete and Download Complete** message. When the message window shows a **:>**, it is ready to process the next command.

2.8 Running HEX files from the PICDEM 17 Demonstration Board

The Monitor program has a Run button that starts the execution of code from the external FLASH program memory. Once a program has been downloaded into the FLASH memory, click the **Run** button to start execution. Control of the PIC17C756A is released by the Monitor firmware by simply writing 4000h to the PCLATH:PCL registers. Refer to Chapter 4*, Using the MPLAB® C17 C Compiler* for more information.

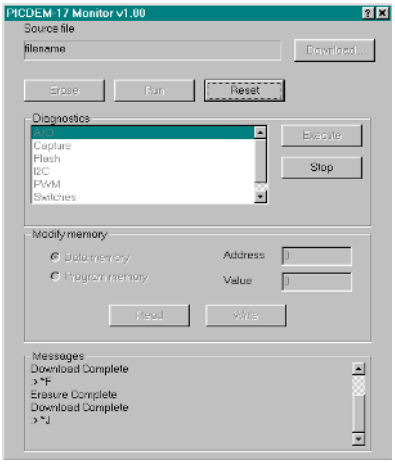


Figure 2.4: Program Running on PICDEM 17 Demonstration Board

The only way to stop the program from running is to click on the **Reset** button in the Monitor window. This resets the PIC17C756A and restores operation to the Monitor firmware. This method allows for all Data memory, interrupt vectors, and stack locations to be available to the user for the target application.

2.9 Running Diagnostics on the PICDEM 17 Demonstration Board

The Monitor firmware on the PIC17C756A provides several diagnostic routines that allow the user to run pretested code on a peripheral. The diagnostics include:

- **A/D** – The A/D diagnostic allows the user to configure the A/D channel, clock source, justification, and reference. The A/D results are displayed in the Monitor message window.

  During A/C Diagnostics, the switches and LEDs take on the following functions:

S5	Increment channel number, 0 – 11 only
S9	Toggle between internal and external voltage references
S10	Toggle between right and left justify
S12	The clock source (FOSC/8, FOSC/32, FOSC/B4, FRC)
D1, D6 – D8	Channel number in HEX format
D9	Internal (OFF) or External (ON) voltage references
D10	Left (OFF) or Right (ON) justify
D11 – D12	A/D clock source

  The diagnostic ends when **Halt** is clicked

- **Capture** – The capture diagnostic allows the user to configure the capture channel and mode. The capture results are displayed in the Monitor message window.

  During Capture Diagnostics, the switches and LEDs take on the following functions:

S5	Increment capture channel, 1 – 4 only
S9	Increment toggle mode
D1, D6	Capture channel
D9, D10	Capture mode

  The diagnostic ends when **Halt** is clicked

- **FLASH** – The external FLASH memory diagnostic simply writes all zero's, all one's, checkerboard, and inverse checkerboard to a portion of memory and checks to make sure that the values were properly written. This diagnostic does not require any user interaction.

- **I^2C** – This diagnostic writes an incrementing count to the 24LC01B Serial EEPROM on the PICDEM 17 demonstration board and verifies that each location has been properly programmed. This diagnostic does not require any interaction from the user.

- **LCD** – This diagnostic configures I/O pins and writes a message to the external LCD panel in 4-bit mode.

portF<0:3>	data lines
portG<1>	E
portG<0>	R/W
portF<7>	RS

  If this diagnostic is selected and these I/O pins are not connected to the LCD panel, then the message will not be displayed correctly. Provided the connections are correct, the diagnostic does not require any interaction from the user.

- **PWM** – This diagnostic allows the user to select the PWM channel and increment or decrement the period and duty cycle of that channel.

  During PWM Diagnostics, the switches and LEDs take on the following functions:

S5	Increments the channel, 1 – 3 only
S7	Decrement the duty cycle down to 0
S8	Increment the duty cycle, no upper limit
S9	Decrement the period down to 0
S10	Increment the period, no upper limit
S12	Selects clock source, Timer1 or Timer2
D1, D6	PWM channel
D12	Timer source Timer1 (OFF) or Timer2 (ON)

  The diagnostic ends when **Halt** is clicked

- **Switches** – This diagnostic tests the functionality of the memory mapped switches and LEDs. Each time a switch is pressed the corresponding LED is turned OFF. This diagnostic ends when all LEDs have been turned OFF.

- **USART2** – This diagnostic writes a message to the Monitor program running on the PC Host. This diagnostic does not require any interaction from the user.

To run a diagnostic, highlight the desired test in the Diagnostics window of the Monitor program. Then click the **Execute** button to start the test. Those tests that do not automatically terminate can be stopped using the **Stop** button.

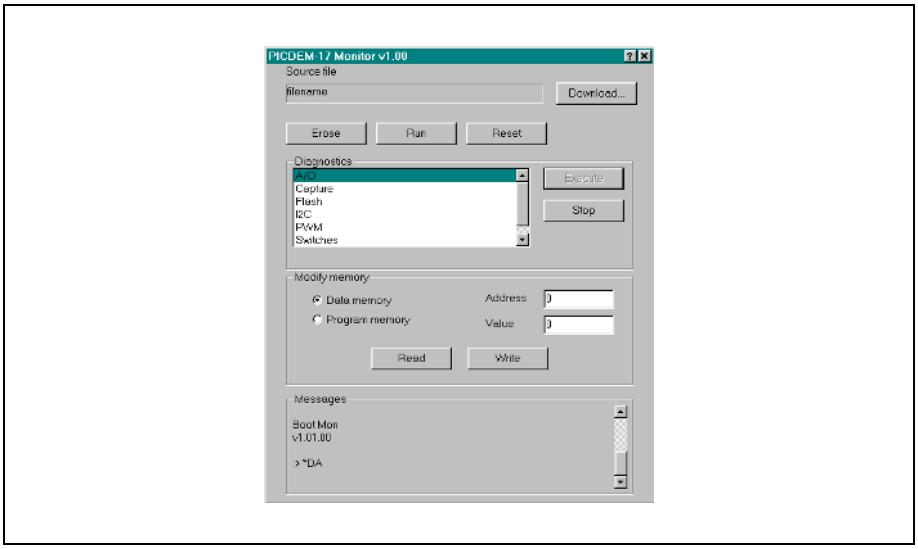


Figure 2.5: Running Diagnostics

2.10 Modifying Memory Contents

The PICDEM 17 Monitor program has the capability to read from any Data memory location or internal/external Program memory location. It can also write to any Data memory location and any external FLASH program memory location. Writes to internal program memory are not allowed because this memory is EPROM based.

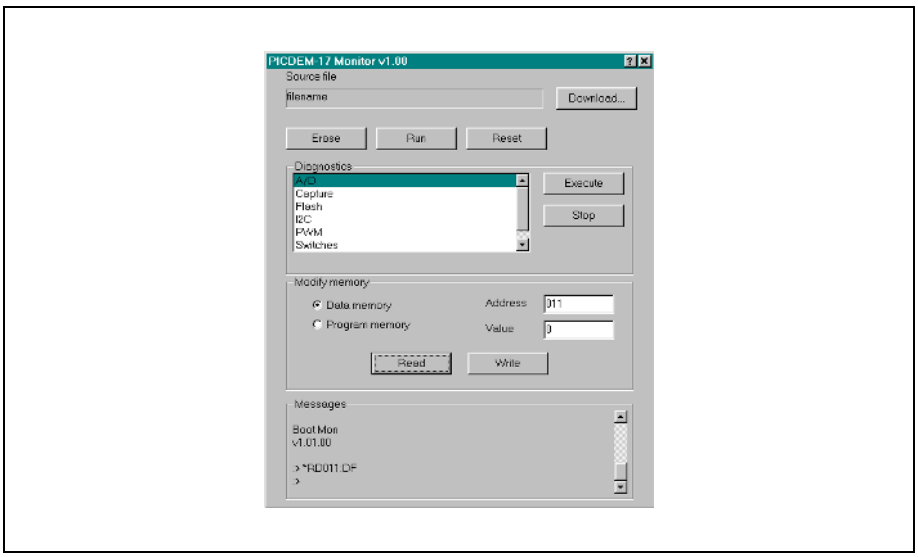


Figure 2.6: Reading Data Memory

To read from a Data memory location, the user must first select the **Data Memory** radio button in the Modify Memory window of the Monitor program. Then the Data memory address must be written into the Address edit box. The format of this value is **bxx**, where **b** is a valid bank number in HEX format, and **xx** is a value from 00h to FFh that indicates the desired address within the bank to read. Then click the **Read** button. The message window will reply with the following message: ***RDbxx:dd**. **bxx** represents the bank and address and **dd** represents the data found at that address in HEX format. For this operation the first three digits in the Address edit box are used when the command is sent to the PICDEM 17 demonstration board. Any additional digits are ignored.

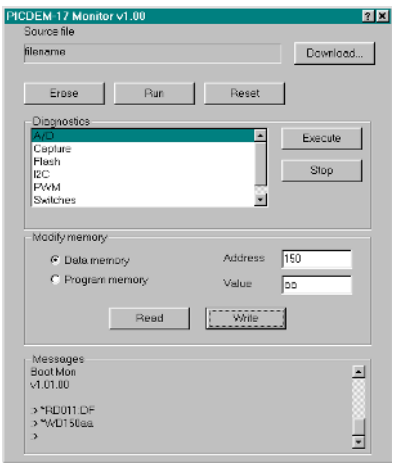


Figure 2.7: Writing Data Memory

To write to a Data memory location, the user must first select the **Data Memory** radio button in the Modify Memory window of the Monitor program. Then the Data memory address must be written into the Address edit box. The format of this value is **bxx**, where **b** is a valid bank number in HEX format, and **xx** is a value from 00h to FFh that indicates the desired address within the bank to write. The user must also enter the value to be written to the Data register in the **Value** edit box. Then click the **Write** button. The message window will reply with the following message: ***WDbxxdd**. **bxx** represents the bank and address and **dd** represents the data written to the address in HEX format. For this operation the first three digits in the **Address** edit box and first two digits in the **Value** edit box are used when the command is sent to the PICDEM 17 demonstration board. Any additional digits are ignored.

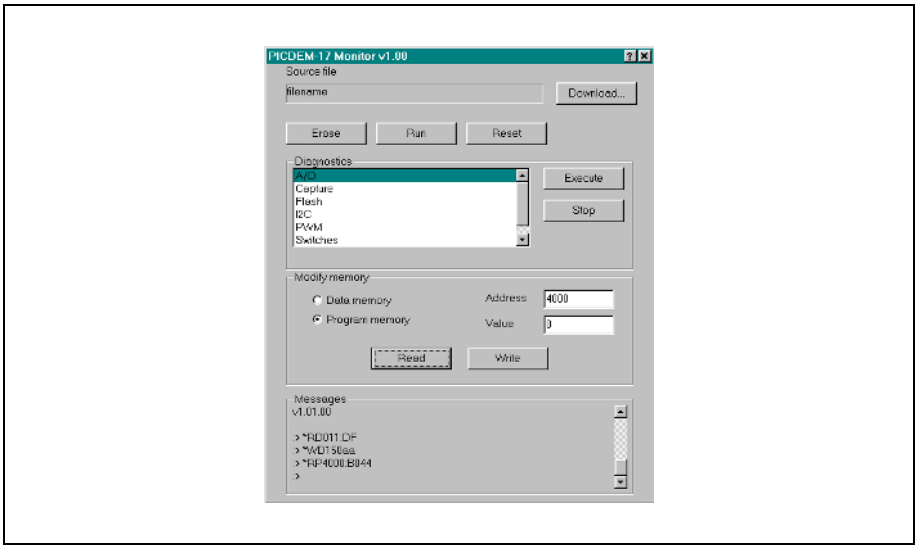


Figure 2.8: Reading Program Memory

To read from a Program memory location, the user must first select the **Program Memory** radio button in the Modify Memory window of the Monitor program. Then the Program memory address must be written into the **Address** edit box. The format of this value is **xxxx**, where **xxxx** is a value from 0000h to FFFFh that indicates the desired address within the 64K address space to read. Then click the **Read** button. The message window will reply with the following message: ***RPxxxx:dddd**. **xxxx** represents the program memory address and **dddd** represents the data at that address in HEX format. For this operation the first four digits in the Address edit box are used when the command is sent to the PICDEM 17 demonstration board. Any additional digits are ignored.

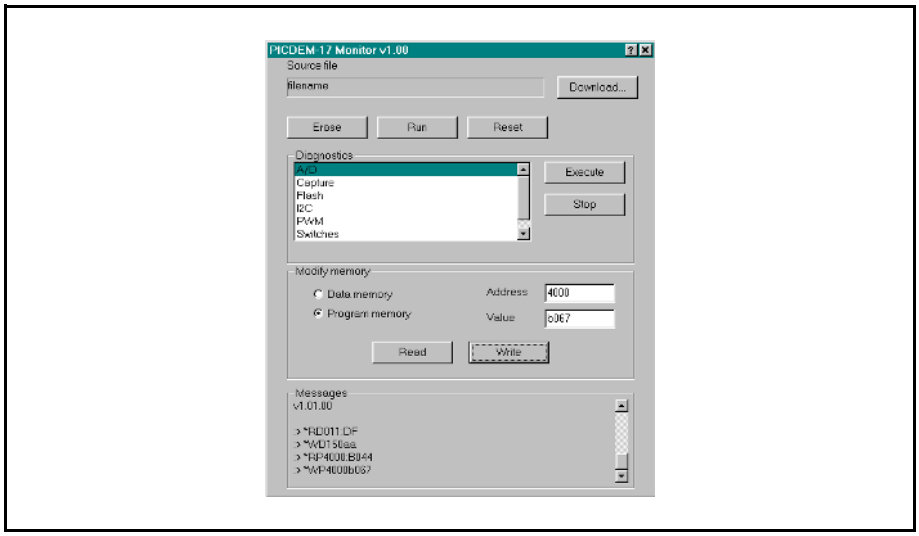


Figure 2.9: Writing Program Memory

To write to a Program memory location, the user must first select the **Program Memory** radio button in the Modify Memory window of the Monitor program. Then the Program memory address must be written into the **Address** edit box. The format of this value is **xxxx**, where **xxxx** is a value from 0000h to FFFFh that indicates the desired address within the bank to write. The user must also enter the value to be written to the program memory location in the **Value** edit box. Then click the **Write** button. The message window will reply with the following message: ***WPxxxxdddd**. **xxxx** represents the program memory address and **dddd** represents the data written to the address in HEX format. For this operation the first four digits of both the **Address** edit box and the **Value** edit box are used when the command is sent to the PICDEM 17 demonstration board. Any additional digits are ignored. The firmware in the PIC17C756A ignores any write program memory commands with an address in the range of 0000h to 3FFFh.

NOTES:

Chapter 3. Hardware Description

3.1 Introduction

The hardware on the PICDEM 17 demonstration board is simple and is intended to illustrate the ease of use and capabilities of the PIC17C7XX family of devices.

3.2 Highlights

This chapter covers the following topics:

- **Port Connections**
- **Push-button Switches**
- **DIP Switches**
- **RS-232 Interface**
- **CAN Bus Interface**
- **Modular Connectors**

3.3 Port Connections

The following bullets list the I/O features and port connections for the PIC17C7XX devices.

- PORTA<0:1> not used
- PORTA<2:3> used for SPI™/I^2C™ interface
- PORTA<4:5> used for USART1 interface
- PORTB<0:3> not used
- PORTB<4:5> used for USART2 handshaking interface (CTS:RTS)
- PORTB<6:7> used for SPI/I^2C interface
- All of PORTC and PORTD are dedicated to the external memory interface
- PORTE<0:2> are used in the external memory interface as ALE, OE, and WR respectively
- PORTF<0:3> are used as the data lines for the external LCD
- PORTF<4:6> not used
- PORTF<7> used as the register select control line to external LCD
- PORTG<0:1> not used
- PORTG<2> tied to analog ground
- PORTG<3> tied to the +4.096V voltage reference
- PORTG<4:5> not used
- PORTG<6:7> used for USART2 interface
- PORTH<0:7> not used
- PORTJ<0:7> not used
- All V_{DD}s tied to digital +5V, all V_{SS} tied to digital ground
- AV_{DD} tied to analog +5V, AV_{SS} tied to analog ground

3.4 Push-button Switches

The PICDEM 17 demonstration board has a total of nine push-button switches. S5 – S12 are the eight general purpose push-button switch inputs that are available to the user. S1 is the $\overline{\text{MCLR}}$ reset push-button switch that resets the PIC17C7XX.

3.5 DIP Switches

The PICDEM 17 demonstration board has two sets of DIP switches that control the SPI/I^2C interfaces. S13 enables either the I^2C or SPI I/O pins from the microcontroller to the peripherals. To use I^2C, make sure the positions labeled SDA and SCL are ON and SCK, SDO, SDI, and CS are OFF. To use SPI, make sure the position labeled SDA and SCL are OFF and SCK, SDO, SDI, and CS are ON. The second set of DIP switches, S14, routes either the CAN Bus I/O or I^2C I/O to the set of modular connectors. This DIP switch is also labeled to SDA/SCL for I^2C or CANH/CANL for CAN Bus.

3.6 RS-232 Interface

The PICDEM 17 demonstration board provides a RS-232 interface device (U3) to convert between RS-232 voltage levels and CMOS/TTL voltage levels. Both USART1 and USART2 I/O pins are routed to this device as well as the I/O pins used for hardware handshaking. The DB9 connector P1 is used for USART1 and P2 is used for USART2. The Monitor program running on the PICDEM 17 demonstration board uses USART2 for the communications channel between the PICDEM 17 demonstration board and the Host PC.

3.7 CAN Bus Interface

The PICDEM 17 demonstration board has the layout for the Microchip MCP2510 CAN Bus peripheral and the PCA82C250 CAN Bus interface. These devices must be provided by the user. To use the CAN Bus interface the user must do the following in addition to adding the components to the board:

- Use a canned oscillator in O2 and remove the oscillator from O1
- Set the DIP switch S13 for SPI (SDA & SCL OFF, SCK, SDO, SDI, & CS ON)
- Set the DIP switch S14 for CAN Bus (SDA & SCL OFF, CANH & CANL ON)

3.8 Modular Connectors

As described previously, the modular connectors can be used to set up a CAN Bus or a I^2C Bus by simply connecting standard 4 or 6-conductor phone cable. Refer to Figure A.1 in Appendix A for the pinout of the connectors.

NOTES:

Chapter 4. Using the MPLAB® C17 C Compiler

4.1 Introduction

This chapter describes the files necessary to compile programs with the MPLAB C17 C compiler to run on the PICDEM 17 demonstration board. A sample linker script file, DEMO756.LKR, is described as well as other support files.

4.2 Highlights

This chapter covers the following topics:

- **Linker Script File, DEMO756.LKR**
- **Startup Code File, C0L17DEM.ASM**
- **Interrupt Code File, INT756LD.ASM**
- **Other Files**

Note 1:	User should refer to MPLAB IDE document that shows project setup, but special version of linker scripts, etc., are used here because of the demonstration board requirements.
2:	Also, refer to the *MPLAB C17 C Compiler User's Guide* (DS51112) for additional information.

4.3 Linker Script File

Any program compiled with the MPLAB C17 C compiler requires the use of a linker script file that documents the areas of memory that the linker is free to use. Programs targeted to run out of external FLASH program memory require a special linker script file that assumes that program memory starts at address 4000h. All of the source code written for the PICDEM 17 demonstration board must be compiled using the large memory model since it will reside at memory addresses larger than 4000h and will be called from a memory address somewhere in the first two pages of memory.

```
//*****************************************************************
//   PIC17C756A MPLAB C v2.0 Linker Script File, Version 1.10
//
//             L a r g e     M o d e l
//
//      For the PICDEM 17 Workshop Demo Board
//
//     (c) Copyright 1997 Microchip Technology
//*****************************************************************

// Add other files to the project
FILESp17c756.o
FILESpmc756l.lib

// ROM area for reset & interrupt vectors
CODEPAGE   PROTECTED NAME=VECTORS START=0x4000   END=0x4027

// User Program memory
// First two pages are used by on-chip monitor program
//CODEPAGE   NAME=PAGE0        START=0x0028    END=0x1FFF
//CODEPAGE   NAME=PAGE1        START=0x2000    END=0x3FFF
// Your program starts in page 2
// First 28h locations of page 2 are vectors
CODEPAGE   NAME=PAGE2        START=0x4028    END=0x5FFF
CODEPAGE   NAME=PAGE3        START=0x6000    END=0x7FFF
CODEPAGE   NAME=PAGE4        START=0x8000    END=0x9FFF
CODEPAGE   NAME=PAGE5        START=0xA000    END=0xBFFF
CODEPAGE   NAME=PAGE6        START=0xC000    END=0xDFFF
CODEPAGE   NAME=PAGE7        START=0xE000    END=0xFFFF

// Special Function registers
// 1. SFRs that are shared (0x00 to 0x0F)
SHAREBANK  PROTECTED  NAME=SFRShareA   START=0x000  END=0x00F
SHAREBANK  PROTECTED  NAME=SFRShareA   START=0x100  END=0x10F
SHAREBANK  PROTECTED  NAME=SFRShareA   START=0x200  END=0x20F
SHAREBANK  PROTECTED  NAME=SFRShareA   START=0x300  END=0x30F
SHAREBANK  PROTECTED  NAME=SFRShareA   START=0x400  END=0x40F
SHAREBANK  PROTECTED  NAME=SFRShareA   START=0x500  END=0x50F
SHAREBANK  PROTECTED  NAME=SFRShareA   START=0x600  END=0x60F
SHAREBANK  PROTECTED  NAME=SFRShareA   START=0x700  END=0x70F

// 2. SFRs in banks 0 - 7
DATABANK   PROTECTED  NAME=SFR0        START=0x010     END=0x017
DATABANK   PROTECTED  NAME=SFR1        START=0x110     END=0x117
DATABANK   PROTECTED  NAME=SFR2        START=0x210     END=0x217
DATABANK   PROTECTED  NAME=SFR3        START=0x310     END=0x317
```

```
DATABANK   PROTECTED  NAME=SFR4        START=0x410     END=0x417
DATABANK   PROTECTED  NAME=SFR5        START=0x510     END=0x517
DATABANK   PROTECTED  NAME=SFR6        START=0x610     END=0x617
DATABANK   PROTECTED  NAME=SSR7        START=0x710     END=0x717

// 3. SFRs shared (0x18 to 0x19)
SHAREBANK  PROTECTED NAME=SFRShareB    START=0x018     END=0x019

// General Purpose Data Memory
// 1. GPRs that are shared (0x1A to 0x1F)
SHAREBANK  NAME=GPRShare     START=0x01A     END=0x01F

// 2. GPRs in banks 0 - 1
DATABANK   NAME=GPR0         START=0x020     END=0x0FF
DATABANK   NAME=GPR1         START=0x120     END=0x1FF
DATABANK   NAME=GPR2         START=0x220     END=0x2FF
DATABANK   NAME=GPR3         START=0x320     END=0x3FF

// Declare a stack
STACK      SIZE=0x20
```

4.4 Startup Code File

The startup code file is used to call the startup function (if enabled), initialize data routine (if enabled), initialize the stack, and then branch to the main function. This file would need to be assembled and linked in with the other files in the project.

```
;*****************************************************************
;**    PIC17Cxx MPLAB C v2.0 Assembly Startup File, Version 1.10
;**              L a r g e    M o d e l
;**     PICDEM 17 Workshop Demo Board Version
;**    (c) Copyright 1997 Microchip Technology
;*****************************************************************

; This is the C startup assembly file for the large model. Please
; refer to chapter 3 in the user's guide for more information.

; The following two statements determine whether you wish to use
; initialized data in your C programs and whether you wish to
; have a __STARTUP() function called upon reset. Please note that
; if you turn USE_STARTUP on, then you must defined a __STARTUP()
; in your code or you will get a linker error.

; #DEFINE USE_INITDATA      ;Uncomment if you use initialized data
; #DEFINE USE_STARTUP       ;Uncomment if you use __STARTUP()

;--------------------- E Q U A T E S ----------------------;
PCL        equ     0x02
PCLATH     equ     0x03

;----------------External variables and labels-------------;
   EXTERN  _stack
   EXTERN  main

#IFDEF  USE_STARTUP
   EXTERN  __STARTUP
#ENDIF

#IFDEF  USE_INITDATA
   EXTERN  copy_init_data
#ENDIF

;*********************************************************;
RESET  CODE  H'4000'    ;Location of reset vector
;---------------------------------------------------------;
;
;---------------------------------------------------------;
; Optionally call __STARTUP()                             ;
;---------------------------------------------------------;
#IFDEF  USE_STARTUP
  movlw  HIGH __STARTUP
  movwf  PCLATH
  lcall  __STARTUP
#ENDIF

;---------------------------------------------------------;
; Branch to startup code                                  ;
;---------------------------------------------------------;
```

```
  movlw HIGH _start
  movwf PCLATH
  movlw LOW _start
  movwf PCL

;*************************************************************;
_start_section   CODE
;-------------------------------------------------------------;
;
;Beginning of startup code
;
_start

;-------------------------------------------------------------;
; Optionally call the routine that copies initlialized data  ;
; from program memory to data memory.                        ;
;-------------------------------------------------------------;
#IFDEF  USE_INITDATA
  movlw  HIGH copy_init_data
  movwf  PCLATH
  lcall  copy_init_data
#ENDIF

;-------------------------------------------------------------;
; Set up the stack for use with function arguments and local ;
; (auto) variables.                                          ;
;-------------------------------------------------------------;
_setup_stack
  BANKSEL  _stack       ;Switch to bank where the stack pointer is
  movlw   _stack+2      ;Store the address of the stack pointer + 2
  movwf   _stack        ;Into the stack pointer

;-------------------------------------------------------------;
; Branch to main() in the C program                          ;
;-------------------------------------------------------------;
  movlw  HIGH main
  movwf  PCLATH
  movlw  LOW main
  movwf  PCL

  END
```

4.5 Interrupt Code File

The interrupt code file is used to setup the interrupt vectors for the INT pin, TMR0 Overflow, T0CKI pin, and the Peripheral Interrupt vector. Included with this code is the context save and restore routines that are called before and after the interrupt service routines. The "real" interrupt vectors of the PIC17C756A located at addresses 0008h, 0010h, 0018h, and 0020h have been remapped to address location 4008h, 4010h, 4018h, and 4020h. The original vectors have code that simply writes the associated vector address of 40??h to the PCLATH:PCL registers. This file would need to be assembled and linked in with the other files in the project.

```
;*****************************************************************
;**    PIC17C756A Interrupt Support File, Version 1.10
;**                L a r g e    M o d e l
;**     PIC1DEM-17 Workshop Demo Board Version
;**    (c) Copyright 1997 Microchip Technology
;*****************************************************************

        list p=17c756
;
;------------------------ Equates --------------------------;
PCL      equ    0x02
PCLATH   equ    0x03
ALUSTA   equ    0x04
BSR      equ    0x0F
WREG     equ    0x0A

SAVEINT_START   equ  0x00FC  ;Start of shared region for
                             ;int.saving

;*********************************************************;
INTSAVE_SEC  UDATA   SAVEINT_START
;---------------------------------------------------------;
; Save registers that absolutely need to be saved!
save_BSR      RES   1  ;Used for saving the BSR - BANK 0 ONLY
save_WREG     RES   1  ;WREG     ;SPACE RESERVED ALSO IN ALL BANKS
save_ALUSTA   RES   1  ;ALUSTA   ;SPACE RESERVED ALSO IN ALL BANKS
save_PCLATH   RES   1  ;PCLATH   ;SPACE RESERVED ALSO IN ALL BANKS
;---------------------------------------------------------;

;*********************************************************;
; save_ALUSTA and save_PCLATH are saved in any of the banks.
; We therefore must reserve the corresponding locations in ALL
; banks. Since BSR and WREG are guaranteed to be saved in
; bank 0, they only need storage locations in bank 0.

;-- Bank 1
INTSAVE_SEC1 UDATA  SAVEINT_START + 0x101
                     RES   1  ;WREG
                     RES   1  ;ALUSTA
                     RES   1  ;CPUSTA
;-- Bank 2
INTSAVE_SEC2 UDATA  SAVEINT_START + 0x201
                     RES   1  ;WREG
```

```
                    RES   1  ;ALUSTA
                    RES   1  ;CPUSTA
;-- Bank 3
INTSAVE_SEC3 UDATA  SAVEINT_START + 0x301
                    RES   1  ;WREG
                    RES   1  ;ALUSTA
                    RES   1  ;CPUSTA
;-----------------------------------------------------------;

;***********************************************************;
VARIABLES  UDATA_OVR
;-----------------------------------------------------------;
; These are the function pointers that have the i.s.r. addresses.
;
fpINT    RES    2
fpTMR0   RES    2
fpT0CKI  RES    2
fpPIV    RES    2

  GLOBAL  fpINT, fpTMR0, fpT0CKI, fpPIV
;-----------------------------------------------------------;

;***********************************************************;
_INT_sec    CODE  H'4008'
;-----------------------------------------------------------;

; Save ALUSTA
  movpf  ALUSTA, save_ALUSTA    ; must save ALUSTA before others
  movpf  WREG,   save_WREG      ; temporarily save off WREG

; Branch to prolog
  movlw HIGH _INT_prolog
  movwf PCLATH
  movlw LOW  _INT_prolog
  movwf PCL
;-----------------------------------------------------------;

;***********************************************************;
_TMR0_sec   CODE  H'4010'
;-----------------------------------------------------------;

; Save ALUSTA
  movpf  ALUSTA, save_ALUSTA    ; must save ALUSTA before others
  movpf  WREG,   save_WREG      ; temporarily save off WREG

; Branch to prolog
  movlw HIGH _TMR0_prolog
  movwf PCLATH
  movlw LOW  _TMR0_prolog
  movwf PCL
;-----------------------------------------------------------;

;***********************************************************;
_T0CKI_sec  CODE  H'4018'
;-----------------------------------------------------------;

; Save ALUSTA
  movpf  ALUSTA, save_ALUSTA    ; must save ALUSTA before others
  movpf  WREG,   save_WREG      ; temporarily save off WREG
```

```
; Branch to prolog
  movlw HIGH _T0CKI_prolog
  movwf PCLATH
  movlw LOW  _T0CKI_prolog
  movwf PCL
;-------------------------------------------------------------;

;*************************************************************;
_PIV_sec  CODE  H'4020'
;-------------------------------------------------------------;

; Save ALUSTA
  movpf  ALUSTA, save_ALUSTA    ; must save ALUSTA before others
  movpf  WREG,   save_WREG      ; temporarily save off WREG

; Branch to prolog
  movlw HIGH _PIV_prolog
  movwf PCLATH
  movlw LOW  _PIV_prolog
  movwf PCL
;-------------------------------------------------------------;

;*************************************************************;
InterruptCode   CODE
;-------------------------------------------------------------;
_INT_prolog

;Save BSR and WREG
  movfp  BSR, PCLATH
  clrf   BSR, 1
  movpf  PCLATH, save_BSR

;Service the interrupt by calling the interrupt
;handling function
  goto   _INT_2
_INT_1
  BANKSEL  fpINT
  movfp  fpINT+1, PCLATH
  movfp  fpINT, PCL
_INT_2
  call   _INT_1     ; push address of next instruction on stack

;Restore WREG and BSR
  clrf   BSR, 1              ;BSR was saved in bank 0
  movfp  save_BSR, BSR       ;Now restore it
  movfp  save_WREG, WREG     ;and WREG as well.

;Restore PCLATH and ALUSTA
  movfp  save_PCLATH, PCLATH
  movfp  save_ALUSTA, ALUSTA

;Return from interrupt
  RETFIE
;-------------------------------------------------------------;

;-------------------------------------------------------------;
_TMR0_prolog
```

```
;Save BSR and WREG
  movfp  BSR, PCLATH
  clrf   BSR, 1
  movpf  PCLATH, save_BSR

;Service the interrupt by calling the interrupt
;handling function
  goto   _TMR0_2
_TMR0_1
  BANKSEL  fpTMR0
  movfp  fpTMR0+1, PCLATH
  movfp  fpTMR0, PCL
_TMR0_2
  call   _TMR0_1    ; push address of next instruction on stack

;Restore WREG and BSR
  clrf   BSR, 1              ;BSR was saved in bank 0
  movfp  save_BSR, BSR       ;Now restore it
  movfp  save_WREG, WREG     ;and WREG as well.

;Restore PCLATH and ALUSTA
  movfp  save_PCLATH, PCLATH
  movfp  save_ALUSTA, ALUSTA

;Return from interrupt
  RETFIE
;-------------------------------------------------------------;

;-------------------------------------------------------------;
_T0CKI_prolog

;Save BSR and WREG
  movfp  BSR, PCLATH
  clrf   BSR, 1
  movpf  PCLATH, save_BSR

;Service the interrupt by calling the interrupt
;handling function
  goto   _T0CKI_2
_T0CKI_1
  BANKSEL  fpT0CKI
  movfp  fpT0CKI+1, PCLATH
  movfp  fpT0CKI, PCL
_T0CKI_2
  call   _T0CKI_1   ; push address of next instruction on stack

;Restore WREG and BSR
  clrf   BSR, 1              ;BSR was saved in bank 0
  movfp  save_BSR, BSR       ;Now restore it
  movfp  save_WREG, WREG     ;and WREG as well.


;Restore PCLATH and ALUSTA
  movfp  save_PCLATH, PCLATH
  movfp  save_ALUSTA, ALUSTA

;Return from interrupt
  RETFIE
;-------------------------------------------------------------;
```

```
;-------------------------------------------------------------;
_PIV_prolog

;Save BSR and WREG
  movfp  BSR, PCLATH
  clrf   BSR, 1
  movpf  PCLATH, save_BSR

;Service the interrupt by calling the interrupt
;handling function
  goto   _PIV_2
_PIV_1
  BANKSEL  fpPIV
  movfp  fpPIV+1, PCLATH
  movfp  fpPIV, PCL
_PIV_2
  call   _PIV_1              ; push address of next instruction on
stack

;Restore WREG and BSR
  clrf   BSR, 1              ;BSR was saved in bank 0
  movfp  save_BSR, BSR       ;Now restore it
  movfp  save_WREG, WREG     ;and WREG as well.

;Restore PCLATH and ALUSTA
  movfp  save_PCLATH, PCLATH
  movfp  save_ALUSTA, ALUSTA

;Return from interrupt
  RETFIE
;-------------------------------------------------------------;

  END
```

4.6 Other Files

One additional file required to compile programs for the PICDEM 17 demonstration board is the associated processor object module. For the PIC17C756A microcontroller this file would be P17C756.ASM. This file would need to be assembled to an object file by a MPASM™ assembler and linked in with the rest of the files.

The only other file that might be used to compile programs for the PICDEM 17 demonstration board may be the library object module. For the PIC17C756A microcontroller the library file PMC756L.LIB would be linked with the rest of the files.

Note 1: User should refer to MPLAB IDE document that shows project setup, but special version of linker scripts, etc., are used here because of the demonstration board requirements.

2: Also, refer to the *MPLAB C17 C Compiler User's Guide* (DS51112) for additional information.

NOTES:

Chapter 5. LCD.C Description

5.1 Introduction

This chapter describes the demonstration program for the PIC17C756A, LCD.C. This program takes an incrementing count and displays it on an external LCD panel using the interface provided on PICDEM 17 demonstration board.

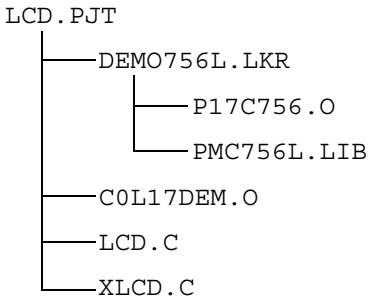
Highlights

This chapter covers the following topics:

- **MPLAB® IDE Project Files**
- **PICmicro® C Libraries**
- **Source Code Descriptions**
- **LCD.C Source Code Listing**
- **XLCD.H Source Code Listing**
- **XLCD.C Source Code Listing**

5.2 MPLAB IDE Project Files

The project LCD.PJT contains the following files:



The files in this project use the following header files:

```
P17C756.H
DELAYS.H
XLCD.H
```

5.3 PICmicro® C Libraries

The files in LCD.PJT use the following libray function(s) contained PMC756L.LIB:

```
Delay10TCY( )
Delay1KTCYx( )
```

5.4 Source Code Descriptions

DEMO756L.LKR	This file is the linker script file for the project.
P17C756.O	This file contains the processor dependent objects, it is included by the linker script file.
PMC756L.LIB	This file contains the library functions compiled for large memory model, it is included by the linker script file.
C0L17DEM.O	This file contains the startup code to initialize the stack, initialize data, call the startup function, and jump to `main( )`.
LCD.C	This file is the main source code file for the project.
XLCD.C	This file contains the source code to interface to the external LCD display.
P17C756.H	This header file contains processor specific items.
DELAYS.H	This is the header file for delay routines in the library.
XLCD.H	This is the header file for the external LCD routines in `XLCD.C`.

5.5 LCD.C Source Code Listing

```
//*****************************************************
//*   LCD.C
//*****************************************************
//*   Rodger Richey
//*   Principal Applications Engineer
//*   Microchip Technology Incorporated
//*****************************************************
//*   2 December 1998
//*   Compiled using MPLAB C17 C Compiler V2.20
//*****************************************************
//*   This program displays a message on a LCD display
//*   that uses the Hitachi HD44780 controller or
//*   equivalent.  The first line shows "Hello..." and
//*   the second line shows "World!".
//*****************************************************
//*   Uses an oscillator of 16MHz
//*****************************************************
#include <p17c756.h>
#include <delays.h>
#include "xlcd.h"

// Constant string arrays in program memory
const rom char Hello[] = "Hello...";
const rom char World[] = "World!";

// Delays for ~18 Tcy
void DelayFor18TCY(void)
{
  Delay10TCY();
  Delay10TCY();
  return;
}

// Delays for ~15ms
void DelayPORXLCD(void)
{
  Delay1KTCYx(70);
  return;
}

// Delays for ~5ms
void DelayXLCD(void)
{
  Delay1KTCYx(20);
  return;
}

void main(void)
{

  // Turn A/D off
  ADCON1 = 0x0e;

  // Configure the external LCD
  OpenXLCD(FOUR_BIT&LINES_5X7);
```

```
  // Print the Hello message to the first line
  putrsXLCD(Hello);

  // Wait for the LCD to finish last command
  while(BusyXLCD());

  // Set the cursor to the start of the 2nd line
  SetDDRamAddr(0x28);

  // Print the World message to the 2nd line
  putrsXLCD(World);

  // Endless loop to end program
  while(1)
  {
      Nop();
      Nop();
  }
}
```

5.6 XLCD.H Source Code Listing

```
#pragma nolist
/*********************************************************************
*   PICmicro C Libraries V2.10
*   Written and Tested using MPLABC V2.10
*********************************************************************
*   Filename:         xlcd.h
*   Date:            14 April 1998
*   File Version:    2.10
*********************************************************************
*   Functions:
*                    Header file
*********************************************************************
*   Revision History:
*      V1.00 - Beta release of Peripheral Libraries for V1.21
*      V2.00 - Release of Peripheral Libraries for V2.00
*      V2.10 - Release of Peripheral Libraries for V2.10
*              Added putrsXLCD to output strings in ROM to XLCD
*********************************************************************
*   Notes:
* - ROM usage varies depending on specified device
* - These libraries routines are written to support the
*   Hitachi HD44780 LCD controller.
* - The user must define the following items:
*       - The LCD interface type (4- or 8-bits)
*       - If 4-bit mode
*       - whether using the upper or lower nibble
*       - The data port
*       - The tris register for data port
*       - The control signal ports and pins
*       - The control signal port tris and pins
*       - The user must provide three delay routines:
*       - DelayFor18TCY() provides a 18 Tcy delay
*       - DelayPORXLCD() provides at least 15ms delay
*       - DelayXLCD() provides at least 5ms delay
*********************************************************************/
#ifndef __XLCD_H
#define __XLCD_H

// Interface type 8-bit or 4-bit
// For 4-bit operation comment out the #define BIT8
//#define BIT8

// When in 4-bit interface define if the data is in the upper
// or lower nibble.  For lower nibble, comment out the #define UPPER
//#define UPPER

// DATA_PORT defines the port on which the LCD
// data lines are connected to
#define DATA_PORT PORTF
#define TRIS_DATA_PORT DDRF

// CTRL_PORT defines the port where the control
// lines are connected
// These are just samples, change to match your application
#define RW_PIN PORTGbits.RG0                    // Port for RW
#define TRIS_RW DDRGbits.RG0                    // TRIS for RW
```

```
#define RS_PIN PORTFbits.RF7                    // Port for RS
#define TRIS_RS DDRFbits.RF7                    // TRIS for RS
#define E_PIN PORTGbits.RG1                     // PORT for E
#define TRIS_E DDRGbits.RG1                     // TRIS for E

// Display ON/OFF Control defines
#define DON                 0b00001111          // Display on
#define DOFF                0b00001011          // Display off
#define CURSOR_ON           0b00001111          // Cursor on
#define CURSOR_OFF          0b00001101          // Cursor off
#define BLINK_ON            0b00001111          // Cursor Blink
#define BLINK_OFF           0b00001110          // Cursor No Blink

// Cursor or Display Shift defines
#define SHIFT_CUR_LEFT      0b00010011          // Cursor shifts to
the left
#define SHIFT_CUR_RIGHT     0b00010111          // Cursor shifts to
the right
#define SHIFT_DISP_LEFT     0b00011011          // Display shifts to
the left
#define SHIFT_DISP_RIGHT 0b00011111             // Display shifts to
the right

// Function Set defines
#define FOUR_BIT            0b00101111          // 4-bit Interface
#define EIGHT_BIT           0b00111111          // 8-bit Interface
#define LINE_5X7            0b00110011          // 5x7 characters,
single line
#define LINE_5X10           0b00110111          // 5x10 characters
#define LINES_5X7           0b00111111          // 5x7 characters,
multiple line

// Other Functions
#define CLS                 0b00000001          // Clear display,
Set DDRAM to 0
#define HOME                0b00000010          // Set DDRAM to 0

void OpenXLCD(static unsigned char);            // Configures I/O
pins for
// external LCD
void SetCGRamAddr(static unsigned char);        // Sets the charac-
ter generator
// address
void SetDDRamAddr(static unsigned char);        // Sets the display
data address
unsigned char BusyXLCD(void);                   // Returns the busy
status of the
// LCD
unsigned char ReadAddrXLCD(void);               // Reads the current
address
char ReadDataXLCD(void);                        // Reads a byte of
data
void WriteCmdXLCD(static unsigned char);        // Writes a command
to the LCD
void WriteDataXLCD(static char);                // Writes a data byte
to the LCD
#define putcXLCD WriteDataXLCD                  // a putc is a write
void putsXLCD(static char *);                   // Writes a string of
characters
```

```
// to the LCD
void putrsXLCD(static const rom char *);       // Writes a string of
characters
// in ROM to the LCD
// User defines these routines according to the oscillator frequency
extern far void DelayFor18TCY(void);
extern far void DelayPORXLCD(void);
extern far void DelayXLCD(void);

#endif
#pragma list
```

5.7 XLCD.C Source Code Listing

```
#include <p17cxx.h>
#include "xlcd.h"

/
************************************************************************
*
*   PICmicro C Libraries V2.10
*   Written and Tested using MPLABC V2.10
************************************************************************
**
*   Filename:         xlcd.c
*   Date:             14 April 1998
*   File Version:     2.10
************************************************************************
**
*   Functions:
*             void OpenXLCD(unsigned char lcdtype)
*             void SetCGRamAddr(unsigned char CGaddr)
*             void SetDDRamAddr(unsigned char DDaddr)
*             unsigned char BusyXLCD(void)
*             unsigned char ReadAddrXLCD(void)
*             char ReadDataXLCD(void)
*             void WriteCmdXLCD(unsigned char cmd)
*             void WriteDataXLCD(char data)
*             void putsXLCD(char *buffer)
*             void putrsXLCD(const rom char *buffer)
************************************************************************
**
*   Revision History:
*      V1.00 - Beta release of Peripheral Libraries for V1.21
*      V2.00 - Release of Peripheral Libraries for V2.00
*      V2.10 - Release of Peripheral Libraries for V2.10
************************************************************************
**
*   Notes:
*      - ROM usage varies depending on specified device
************************************************************************
*/

/
************************************************************************
*
* Function Name:     putsXLCD
* Return Value:      void
* Parameters:        buffer: pointer to string
* Description:       This routine writes a string of bytes to the
*                    Hitachi HD44780 LCD controller. The user
*                    must check to see if the LCD controller is
*                    busy before calling this routine. The data
*                    is written to the character generator RAM or
*                    the display data RAM depending on what the
*                    previous SetxxRamAddr routine was called.
************************************************************************
*/
void putsXLCD(static char *buffer)
{
```

```
    while(*buffer)                              // Write data to LCD
up to null
    {
            while(BusyXLCD());                  // Wait while LCD is
busy
            WriteDataXLCD(*buffer);             // Write character
to LCD
            buffer++;                           // Increment buffer
    }
    return;
}

/
*********************************************************************
*
* Function Name:     putrsXLCD
* Return Value:      void
* Parameters:        buffer: pointer to string
* Description:       This routine writes a string of bytes to the
*                    Hitachi HD44780 LCD controller. The user
*                    must check to see if the LCD controller is
*                    busy before calling this routine. The data
*                    is written to the character generator RAM or
*                    the display data RAM depending on what the
*                    previous SetxxRamAddr routine was called.
*********************************************************************
*/
void putrsXLCD(static const rom char *buffer)
{
    while(*buffer)                              // Write data to LCD
up to null
    {
      while(BusyXLCD());                        // Wait while LCD is
busy
      WriteDataXLCD(*buffer);                   // Write character
to LCD
      buffer++;                                 // Increment buffer
    }
    return;
}

/
*********************************************************************
*
* Function Name:     OpenXLCD
* Return Value:      void
* Parameters:        lcdtype: sets the type of LCD (lines)
* Description:       his routine configures the LCD. Based on
*                    the Hitachi HD44780 LCD controller. The
*                    routine will configure the I/O pins of the
*                    microcontroller, setup the LCD for 4- or
*                    8-bit mode and clear the display. The user
*                    must provide three delay routines:
*                    DelayFor18TCY() provides a 18 Tcy delay
*                    DelayPORXLCD() provides at least 15ms delay
*                    DelayXLCD() provides at least 5ms delay
*********************************************************************
*/
void OpenXLCD(static unsigned char lcdtype)
```

```
{
    // The data bits must be either a 8-bit port or the upper or
    // lower 4-bits of a port. These pins are made into inputs
#ifdef BIT8                                     // 8-bit mode, use
whole port
    DATA_PORT = 0;
    TRIS_DATA_PORT = 0xff;
#else                                           // 4-bit mode
#ifdef UPPER                                    // Upper 4-bits of
the port
    DATA_PORT &= 0x0f;
    TRIS_DATA_PORT |= 0xf0;
#else                                           // Lower 4-bits of
the port
    DATA_PORT &= 0xf0;
    TRIS_DATA_PORT |= 0x0f;
#endif
#endif
    TRIS_RW = 0;                                // All control sig-
nals made outputs
    TRIS_RS = 0;
    TRIS_E = 0;
    RW_PIN = 0;                                 // R/W pin made low
    RS_PIN = 0;                                 // Register select
pin made low
    E_PIN = 0;                                  // Clock pin made low

    // Delay for 15ms to allow for LCD Power on reset
    DelayPORXLCD();

    // Setup interface to LCD
#ifdef BIT8                                     // 8-bit mode inter-
face
    TRIS_DATA_PORT = 0;                         // Data port output
    DATA_PORT = 0b00110000;                     // Function set
cmd(8-bit interface)
#else                                           // 4-bit mode inter-
face
#ifdef UPPER                                    // Upper nibble
interface
    TRIS_DATA_PORT &= 0x0f;
    DATA_PORT &= 0x0f;
    DATA_PORT |= 0b00110000;                    // Function set
cmd(4-bit interface)
#else                                           // Lower nibble
interface
    TRIS_DATA_PORT &= 0xf0;
    DATA_PORT &= 0xf0;
    DATA_PORT |= 0b00000011;                    // Function set
cmd(4-bit interface)
#endif
#endif
    E_PIN = 1;                                  // Clock the cmd in
    DelayFor18TCY();
    E_PIN = 0;

    // Delay for at least 4.1ms
    DelayXLCD();
```

```
    // Setup interface to LCD
#ifdef BIT8                                 // 8-bit interface
    DATA_PORT = 0b00110000;                 // Function set
cmd(8-bit interface)
#else                                       // 4-bit interface
#ifdef UPPER                                // Upper nibble
interface
    DATA_PORT &= 0x0f;                      // Function set
cmd(4-bit interface)
    DATA_PORT |= 0b00110000;
#else                                       // Lower nibble
interface
    DATA_PORT &= 0xf0;                      // Function set
cmd(4-bit interface)
    DATA_PORT |= 0b00000011;
#endif
#endif
    E_PIN = 1;                              // Clock the cmd in
    DelayFor18TCY();
    E_PIN = 0;

    // Delay for at least 100us
    DelayXLCD();

    // Setup interface to LCD
#ifdef BIT8                                 // 8-bit interface
    DATA_PORT = 0b00110000;                 // Function set
cmd(8-bit interface)
#else                                       // 4-bit interface
#ifdef UPPER                                // Upper nibble
interface
    DATA_PORT &= 0x0f;                      // Function set
cmd(4-bit interface)
    DATA_PORT |= 0b00110000;
#else                                       // Lower nibble
interface
    DATA_PORT &= 0xf0;                      // Function set
cmd(4-bit interface)
    DATA_PORT |= 0b00000011;
#endif
#endif
    E_PIN = 1;                              // Clock cmd in
    DelayFor18TCY();
    E_PIN = 0;

    DelayXLCD();

#ifndef BIT8
#ifdef UPPER                                // Upper nibble
interface
    DATA_PORT &= 0x0f;                      // Function set
cmd(4-bit interface)
    DATA_PORT |= 0b00100000;
#else                                       // Lower nibble
interface
    DATA_PORT &= 0xf0;                      // Function set
cmd(4-bit interface)
    DATA_PORT |= 0b00000010;
#endif
```

```
    E_PIN = 1;                                  // Clock cmd in
    DelayFor18TCY();
    E_PIN = 0;
#endif

#ifdef BIT8                                     // 8-bit interface
    TRIS_DATA_PORT = 0xff;// Make data port input
#else                                           // 4-bit interface
#ifdef UPPER                                    // Upper nibble
interface
    TRIS_DATA_PORT |= 0xf0;                     // Make data nibble
input
#else                                           // Lower nibble
interface
    TRIS_DATA_PORT |= 0x0f;                     // Make data nibble
input
#endif
#endif

    // Set data interface width, # lines, font
    while(BusyXLCD());                          // Wait if LCD busy
    WriteCmdXLCD(lcdtype);                      // Function set cmd

    // Turn the display on then off
    while(BusyXLCD());                          // Wait if LCD busy
    WriteCmdXLCD(DOFF&CURSOR_OFF&BLINK_OFF);// Display OFF/Blink
OFF
    while(BusyXLCD());                          // Wait if LCD busy
    WriteCmdXLCD(DON&CURSOR_ON&BLINK_ON);// Display ON/Blink ON

    // Clear display
    while(BusyXLCD());                          // Wait if LCD busy
    WriteCmdXLCD(0x01);                         // Clear display

    // Set entry mode inc, no shift
    while(BusyXLCD());                          // Wait if LCD busy
    WriteCmdXLCD(SHIFT_CUR_LEFT);// Entry Mode

    // Set DD Ram address to 0
    while(BusyXLCD());                          // Wait if LCD busy
    SetDDRamAddr(0);                            // Set Display data
ram address to 0

    return;
}

/
*********************************************************************
*
* Function Name:     WriteCmdXLCD
* Return Value:      void
* Parameters:        cmd: command to send to LCD
* Description:       This routine writes a command to the Hitachi
*                    HD44780 LCD controller. The user must check
*                    to see if the LCD controller is busy before
*                    calling this routine.
*********************************************************************
*/
void WriteCmdXLCD(static unsigned char cmd)
```

```
{
    while(BusyXLCD());
#ifdef BIT8                                     // 8-bit interface
    TRIS_DATA_PORT = 0;                         // Data port output
    DATA_PORT = cmd;                            // Write command to
data port
    RW_PIN = 0;                                 // Set the control
signals
    RS_PIN = 0;                                 // for sending a com-
mand
    DelayFor18TCY();
    E_PIN = 1;                                  // Clock the command
in
    DelayFor18TCY();
    E_PIN = 0;
    DelayFor18TCY();
    TRIS_DATA_PORT = 0xff;// Data port input
#else                                           // 4-bit interface
#ifdef UPPER                                    // Upper nibble
interface
    TRIS_DATA_PORT &= 0x0f;
    DATA_PORT &= 0x0f;
    DATA_PORT |= cmd&0xf0;
#else                                           // Lower nibble
interface
    TRIS_DATA_PORT &= 0xf0;
    DATA_PORT &= 0xf0;
    DATA_PORT |= (cmd>>4)&0x0f;
#endif
    RW_PIN = 0;                                 // Set control sig-
nals for command
    RS_PIN = 0;
    DelayFor18TCY();
    E_PIN = 1;                                  // Clock command in
    DelayFor18TCY();
    E_PIN = 0;
#ifdef UPPER                                    // Upper nibble
interface
    DATA_PORT &= 0x0f;
    DATA_PORT |= (cmd<<4)&0xf0;
#else                                           // Lower nibble
interface
    DATA_PORT &= 0xf0;
    DATA_PORT |= cmd&0x0f;
#endif
    DelayFor18TCY();
    E_PIN = 1;                                  // Clock command in
    DelayFor18TCY();
    E_PIN = 0;
#ifdef UPPER                                    // Make data nibble
input
    TRIS_DATA_PORT |= 0xf0;
#else
    TRIS_DATA_PORT |= 0x0f;
#endif
#endif
    return;
}
```

```
/
*********************************************************************
*
* Function Name:     SetCGRamAddr
* Return Value:      void
* Parameters:        CGaddr: character generator ram address
* Description:       This routine sets the character generator
*                    address of the Hitachi HD44780 LCD
*                    controller. The user must check to see if
*                    the LCD controller is busy before calling
*                    this routine.
*********************************************************************
*/
void SetCGRamAddr(static unsigned char CGaddr)
{
    while(BusyXLCD());
#ifdef BIT8                                     // 8-bit interface
    TRIS_DATA_PORT = 0;                         // Make data port
ouput
    DATA_PORT = CGaddr | 0b01000000;            // Write cmd and
address to port
    RW_PIN = 0;                                 // Set control sig-
nals
    RS_PIN = 0;
    DelayFor18TCY();
    E_PIN = 1;                                  // Clock cmd and
address in
    DelayFor18TCY();
    E_PIN = 0;
    DelayFor18TCY();
    TRIS_DATA_PORT = 0xff;                      // Make data port
inputs
#else                                           // 4-bit interface
#ifdef UPPER                                    // Upper nibble
interface
    TRIS_DATA_PORT &= 0x0f;                     // Make nibble input
    DATA_PORT &= 0x0f;                          // and write upper
nibble
    DATA_PORT |= ((CGaddr | 0b01000000) & 0xf0);
#else                                           // Lower nibble
interface
    TRIS_DATA_PORT &= 0xf0;                     // Make nibble input
    DATA_PORT &= 0xf0;                          // and write upper
nibble
    DATA_PORT |= (((CGaddr |0b01000000)>>4) & 0x0f);
#endif
    RW_PIN = 0;                                 // Set control sig-
nals
    RS_PIN = 0;
    DelayFor18TCY();
    E_PIN = 1;                                  // Clock cmd and
address in
    DelayFor18TCY();
    E_PIN = 0;
#ifdef UPPER                                    // Upper nibble
interface
    DATA_PORT &= 0x0f;                          // Write lower nib-
ble
    DATA_PORT |= ((CGaddr<<4)&0xf0);
```

```
#else                                           // Lower nibble
interface
    DATA_PORT &= 0xf0;                          // Write lower nib-
ble
    DATA_PORT |= (CGaddr&0x0f);
#endif
    DelayFor18TCY();
    E_PIN = 1;                                  // Clock cmd and
address in
    DelayFor18TCY();
    E_PIN = 0;
#ifdef UPPER                                    // Upper nibble
interface
    TRIS_DATA_PORT |= 0xf0;                     // Make inputs
#else                                           // Lower nibble
interface
    TRIS_DATA_PORT |= 0x0f;                     // Make inputs
#endif
#endif
    return;
}

/
*********************************************************************
*
* Function Name:     SetDDRamAddr
* Return Value:      void
* Parameters:        CGaddr: display data address
* Description:       This routine sets the display data address
*                    of the Hitachi HD44780 LCD controller. The
*                    user must check to see if the LCD controller
*                    is busy before calling this routine.
*********************************************************************
*/
void SetDDRamAddr(static unsigned char DDaddr)
{
    while(BusyXLCD());
#ifdef BIT8                                     // 8-bit interface
    TRIS_DATA_PORT = 0;                         // Make port output
    DATA_PORT = DDaddr | 0b10000000;            // Write cmd and
address to port
    RW_PIN = 0;                                 // Set the control
bits
    RS_PIN = 0;
    DelayFor18TCY();
    E_PIN = 1;                                  // Clock the cmd and
address in
    DelayFor18TCY();
    E_PIN = 0;
    DelayFor18TCY();
    TRIS_DATA_PORT = 0xff;                      // Make port input
#else                                           // 4-bit interface
#ifdef UPPER                                    // Upper nibble
interface
    TRIS_DATA_PORT &= 0x0f;                     // Make port output
    DATA_PORT &= 0x0f;                          // and write upper
nibble
    DATA_PORT |= ((DDaddr | 0b10000000) & 0xf0);
```

```
#else                                           // Lower nibble
interface
    TRIS_DATA_PORT &= 0xf0;                     // Make port output
    DATA_PORT &= 0xf0;                          // and write upper
nibble
    DATA_PORT |= (((DDaddr | 0b10000000)>>4) & 0x0f);
#endif
    RW_PIN = 0;                                 // Set control bits
    RS_PIN = 0;
    DelayFor18TCY();
    E_PIN = 1;                                  // Clock the cmd and
address in
    DelayFor18TCY();
    E_PIN = 0;
#ifdef UPPER                                    // Upper nibble
interface
    DATA_PORT &= 0x0f;                          // Write lower nib-
ble
    DATA_PORT |= ((DDaddr<<4)&0xf0);
#else                                           // Lower nibble
interface
    DATA_PORT &= 0xf0;                          // Write lower nib-
ble
    DATA_PORT |= (DDaddr&0x0f);
#endif
    DelayFor18TCY();
    E_PIN = 1;                                  // Clock the cmd and
address in
    DelayFor18TCY();
    E_PIN = 0;
#ifdef UPPER                                    // Upper nibble
interface
    TRIS_DATA_PORT |= 0xf0;                     // Make port input
#else                                           // Lower nibble
interface
    TRIS_DATA_PORT |= 0x0f;                     // Make port input
#endif
#endif
    return;
}

/
*********************************************************************
*
* Function Name:     BusyXLCD
* Return Value:      char: busy status of LCD controller
* Parameters:        void
* Description:       This routine reads the busy status of the
*                    Hitachi HD44780 LCD controller.
*********************************************************************
*/
unsigned char BusyXLCD(void)
{
    RW_PIN = 1;                                 // Set the control
bits for read
    RS_PIN = 0;
    DelayFor18TCY();
    E_PIN = 1;                                  // Clock in the com-
mand
```

```
    DelayFor18TCY();
#ifdef BIT8                                     // 8-bit interface
    if(DATA_PORT.7)                             // Read bit 7 (busy
bit)
    {                                           // If high
            E_PIN = 0;                          // Reset clock line
            RW_PIN = 0;                         // Reset control
line
            return 1;                           // Return TRUE
    }
    else                                        // Bit 7 low
    {
            E_PIN = 0;                          // Reset clock line
            RW_PIN = 0;                         // Reset control
line
            return 0;                           // Return FALSE
    }
#else                                           // 4-bit interface
#ifdef UPPER                                    // Upper nibble
interface
    if(DATA_PORT&0x80)
#else                                           // Lower nibble
interface
    if(DATA_PORT&0x08)
#endif
    {
            E_PIN = 0;                          // Reset clock line
            DelayFor18TCY();
            E_PIN = 1;                          // Clock out other
nibble
            DelayFor18TCY();
            E_PIN = 0;
            RW_PIN = 0;                         // Reset control
line
            return 1;                           // Return TRUE
    }
    else                                        // Busy bit is low
    {
            E_PIN = 0;                          // Reset clock line
            DelayFor18TCY();
            E_PIN = 1;                          // Clock out other
nibble
            DelayFor18TCY();
            E_PIN = 0;
            RW_PIN = 0;                         // Reset control
line
            return 0;                           // Return FALSE
    }
#endif
}

/
*********************************************************************
**
* Function Name:     ReadAddrXLCD
* Return Value:      char: address from LCD controller
* Parameters:        void
* Description:       This routine reads an address byte from the
*                    Hitachi HD44780 LCD controller. The user
```

```
*                   must check to see if the LCD controller is
*                   busy before calling this routine. The address
*                   is read from the character generator RAM or
*                   the display data RAM depending on what the
*                   previous SetxxRamAddr routine was called.
*********************************************************************
**/
unsigned char ReadAddrXLCD(void)
{
    char data;                          // Holds the data retrieved
from the LCD

#ifdef BIT8                             // 8-bit interface
    RW_PIN = 1;                         // Set control bits for the
read
    RS_PIN = 0;
    DelayFor18TCY();
    E_PIN = 1;                          // Clock data out of the LCD
controller
    DelayFor18TCY();
    data = DATA_PORT;                   // Save the data in the reg-
ister
    E_PIN = 0;
    RW_PIN = 0;                         // Reset the control bits
#else                                   // 4-bit interface
    RW_PIN = 1;                         // Set control bits for the
read
    RS_PIN = 0;
    DelayFor18TCY();
    E_PIN = 1;                          // Clock data out of the LCD
controller
    DelayFor18TCY();
#ifdef UPPER                            // Upper nibble interface
    data = DATA_PORT&0xf0;// Read the nibble into the upper nibble
of data
#else                                   // Lower nibble interface
    data = (DATA_PORT<<4)&0xf0;// Read nibble to upper nibble of
data
#endif
    E_PIN = 0;                          // Reset the clock
    DelayFor18TCY();
    E_PIN = 1;                          // Clock out the lower nib-
ble
    DelayFor18TCY();
#ifdef UPPER                            // Upper nibble interface
    data |= (DATA_PORT>>4)&0x0f;// Read nibble to lower nibble of
data
#else                                   // Lower nibble interface
    data |= DATA_PORT&0x0f;             // Read nibble to lower nib-
ble of data
#endif
    E_PIN = 0;
    RW_PIN = 0;                         // Reset the control lines
#endif
    return (data&0x7f);                 // Return the address, Mask
off the busy bit
}
```

```
/
*********************************************************************
*
* Function Name:     ReadDataXLCD
* Return Value:      char: data byte from LCD controller
* Parameters:        void
* Description:       This routine reads a data byte from the
*                    Hitachi HD44780 LCD controller. The user
*                    must check to see if the LCD controller is
*                    busy before calling this routine. The data
*                    is read from the character generator RAM or
*                    the display data RAM depending on what the
*                    previous SetxxRamAddr routine was called.
*********************************************************************
*/
char ReadDataXLCD(void)
{
    char data;

#ifdef BIT8                                     // 8-bit interface
    RS_PIN = 1;                                 // Set the control
bits
    RW_PIN = 1;
    DelayFor18TCY();
    E_PIN = 1;                                  // Clock the data out
of the LCD
    DelayFor18TCY();
    data = DATA_PORT;                           // Read the data
    E_PIN = 0;
    RS_PIN = 0;                                 // Reset the control
bits
    RW_PIN = 0;
#else                                           // 4-bit interface
    RW_PIN = 1;
    RS_PIN = 1;
    DelayFor18TCY();
    E_PIN = 1;                                  // Clock the data out
of the LCD
    DelayFor18TCY();
#ifdef UPPER                                    // Upper nibble
interface
    data = DATA_PORT&0xf0;                      // Read the upper
nibble of data
#else                                           // Lower nibble
interface
    data = (DATA_PORT<<4)&0xf0;                 // read the upper
nibble of data
#endif
    E_PIN = 0;                                  // Reset the clock
line
    DelayFor18TCY();
    E_PIN = 1;                                  // Clock the next
nibble out of the LCD
    DelayFor18TCY();
#ifdef UPPER                                    // Upper nibble
interface
    data |= (DATA_PORT>>4)&0x0f;                // Read the lower
nibble of data
```

```
#else                                           // Lower nibble
interface
    data |= DATA_PORT&0x0f;                     // Read the lower
nibble of data
#endif
    E_PIN = 0;
    RS_PIN = 0;                                 // Reset the control
bits
    RW_PIN = 0;
#endif
    return(data);                               // Return the data
byte
}

/
*********************************************************************
*
* Function Name:     WriteDataXLCD
* Return Value:      void
* Parameters:        data: data byte to be written to LCD
* Description:       This routine writes a data byte to the
*                    Hitachi HD44780 LCD controller. The user
*                    must check to see if the LCD controller is
*                    busy before calling this routine. The data
*                    is written to the character generator RAM or
*                    the display data RAM depending on what the
*                    previous SetxxRamAddr routine was called.
*********************************************************************
*/
void WriteDataXLCD(static char data)
{
    while(BusyXLCD());
#ifdef BIT8                                     // 8-bit interface
    TRIS_DATA_PORT = 0;                         // Make port output
    DATA_PORT = data;                           // Write data to port
    RS_PIN = 1;                                 // Set control bits
    RW_PIN = 0;
    DelayFor18TCY();
    E_PIN = 1;                                  // Clock data into
LCD
    DelayFor18TCY();
    E_PIN = 0;
    RS_PIN = 0;                                 // Reset control
bits
    TRIS_DATA_PORT = 0xff;                      // Make port input
#else                                           // 4-bit interface
#ifdef UPPER                                    // Upper nibble
interface
    TRIS_DATA_PORT &= 0x0f;
    DATA_PORT &= 0x0f;
    DATA_PORT |= data&0xf0;
#else                                           // Lower nibble
interface
    TRIS_DATA_PORT &= 0xf0;
    DATA_PORT &= 0xf0;
    DATA_PORT |= ((data>>4)&0x0f);
#endif
    RS_PIN = 1;                                 // Set control bits
    RW_PIN = 0;
```

```
    DelayFor18TCY();
    E_PIN = 1;                                  // Clock nibble into
LCD
    DelayFor18TCY();
    E_PIN = 0;
#ifdef UPPER                                    // Upper nibble
interface
    DATA_PORT &= 0x0f;
    DATA_PORT |= ((data<<4)&0xf0);
#else                                           // Lower nibble
interface
    DATA_PORT &= 0xf0;
    DATA_PORT |= (data&0x0f);
#endif
    DelayFor18TCY();
    E_PIN = 1;                                  // Clock nibble into
LCD
    DelayFor18TCY();
    E_PIN = 0;
#ifdef UPPER                                    // Upper nibble
interface
    TRIS_DATA_PORT |= 0xf0;
#else                                           // Lower nibble
interface
    TRIS_DATA_PORT |= 0x0f;
#endif
#endif
    return;
}
```

NOTES:

Chapter 6. USART.C Description

6.1 Introduction

This chapter describes the demonstration program for the PIC17C756A, USART.C. This program takes an incrementing count and displays it on the Monitor program using USART2 and hardware handshaking.

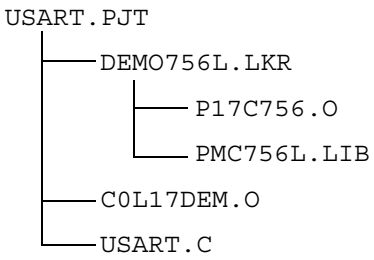
6.2 Highlights

This chapter covers the following topics:

- **MPLAB® IDE Project Files**
- **PICmicro® C Libraries**
- **Source Code Descriptions**
- **USART.C Source Code Listing**

6.3 MPLAB IDE Project Files

The project USART.PJT contains the following files:



The files in this project use the following header files:

```
P17C756.H
DELAYS.H
USART16.H
STDLIB.H
```

6.4 PICmicro C Libraries

The files in USART.PJT use the following library function(s) contained PMC756L.LIB:

```
Delay10KTCYx( )
BusyUSART2( )
putcUSART2( )
OpenUSART2( )
ubtoa( )
```

6.5 Source Code Descriptions

DEMO756L.LKR	This file is the linker script file for the project.
P17C756.O	This file contains the processor dependent objects, it is included by the linker script file.
PMC756L.LIB	This file contains the library functions compiled for large memory model, it is included by the linker script file.
C0L17DEM.O	This file contains the startup code to initialize the stack, initialize data, call the startup function, and jump to `main( )`.
USART.C	This file is the main source code file for the project.
P17C756.H	This header file contains processor specific items.
DELAYS.H	This is the header file for delay routines in the library.
USART16.H	This is the header file for the USART routines in the library.
STDLIB.H	This is the header file for standard library routines.

6.6 USART.C Source Code Listing

```
//*****************************************************
//*   USART.C
//*****************************************************
//*   Rodger Richey
//*   Principal Applications Engineer
//*   Microchip Technology Incorporated
//*****************************************************
//*   2 December 1998
//*   Compiled using MPLAB C17 C Compiler V2.20
//*****************************************************
//*   This program prints an incrementing count from
//*   0 to 255 to the USART. The numbers are displayed
//*   in the Monitor program.
//*****************************************************
//*   Uses an oscillator of 16MHz
//*****************************************************
#include <p17c756.h>
#include <delays.h>
#include <usart16.h>
#include <stdlib.h>

void PutsUSART2(char *data);

// Function to print a string to USART2 using handshaking
void PutsUSART2(char *data)
{
      do
      {
            // Wait for USART to complete prev operation
            while(BusyUSART2());

            // Hardware handshaking for CTS
            while(PORTBbits.RB4);

            // Print a character to USART2
            putcUSART2(*data);

      } while(*data++);  // Increment pointer and check for NULL
      return;
}

void main(void)
{
      unsigned char i;
      char str[5];
      char crlf[3];

      // Initialize the carriage return/linefeed string
      crlf[0] = 0x0d;
      crlf[1] = 0x0a;
      crlf[2] = 0;

      // Initialize USART2 and the hardware handshaking lines
      PORTBbits.RB5 = 1;
      DDRBbits.RB5 = 0;
```

```
OpenUSART2(USART_TX_INT_OFF&USART_RX_INT_OFF&USART_ASYNCH_MODE&
           USART_EIGHT_BIT&USART_CONT_RX,25);
      // Send a carriage return and linefeed
      PutsUSART2(crlf);

      // Initialize the count variable
      i = 0;
      while(1)
      {
             // Convert the count variable to ASCII
             ubtoa(i,str);

             // Print the string
             PutsUSART2(str);

             // Print a carriage return and linefeed
             PutsUSART2(crlf);

             // Wait for a while
             Delay10KTCYx(250);
             Delay10KTCYx(250);

             // Increment the count variable
             i++;
      }
}
```

Chapter 7. ANALOG.C Description

7.1 Introduction

This chapter describes the demonstration program for the PIC17C756A, ANALOG.C. This program continuously converts on channel 0 (which is connected to a precision 4.096V voltage reference) and displays the result in ASCII on the Monitor program. The A/D module is configured to use the AV_{DD} and AV_{SS} pins as the voltage reference.

Highlights

This chapter covers the following topics:

- **MPLAB® IDE Project Files**
- **PICmicro® C Libraries**
- **Source Code Descriptions**
- **ANALOG.C Source Code Listing**

MPLAB IDE Project Files

The project ANALOG.PJT contains the following files:

```
ANALOG.PJT
    ├──DEMO756L.LKR
    │       ├──P17C756.O
    │       └──PMC756L.LIB
    ├──C0L17DEM.O
    └──ANALOG.C
```

The files in this project use the following header files:

```
P17C756.H
DELAYS.H
USART16.H
STDLIB.H
ADC16.H
```

7.2 PICmicro C Libraries

The files in ANALOG.PJT use the following library function(s) contained in PMC756L.LIB:

```
Delay10KTCYx( )
BusyUSART2( )
putcUSART2( )
OpenUSART2( )
OpenADC( )
ConvertADC( )
BusyADC( )
ReadADC( )
uitoa( )
```

7.3 Source Code Descriptions

DEMO756L.LKR	This file is the linker script file for the project.
P17C756.O	This file contains the processor dependent objects, it is included by the linker script file.
PMC756L.LIB	This file contains the library functions compiled for large memory model, it is included by the linker script file.
C0L17DEM.O	This fwwile contains the startup code to initialize the stack, initialize data, call the startup function, and jump to `main( )`.
ANALOG.C	This file is the main source code file for the project.
P17C756.H	This header file contains processor specific items.
DELAYS.H	This is the header file for delay routines in the library.
USART16.H	This is the header file for the USART routines in the library.
STDLIB.H	This is the header file for standard library routines.
ADC16.H	This is the header file for A/D routines in the library.

7.4 ANALOG.C Source Code Listing

```
//*****************************************************
//*   ANALOG.C
//*****************************************************
//*   Rodger Richey
//*   Principal Applications Engineer
//*   Microchip Technology Incorporated
//*****************************************************
//*   2 December 1998
//*   Compiled using MPLAB C17 C Compiler V2.20
//*****************************************************
//*   This program performs an A/D conversion on CH2
//*   and prints the result in ASCII to USART2 which
//*   displays the result on the Monitor program.
//*****************************************************
//*   Uses an oscillator of 16MHz
//*****************************************************
#include <p17c756.h>
#include <delays.h>
#include <usart16.h>
#include <stdlib.h>
#include <adc16.h>

void PutsUSART2(char *data);

// Prints a string to USART2 using hardware handshaking;
void PutsUSART2(char *data)
{
  do
  {
          // Wait for the USART to finish prev operation
          while(BusyUSART2());

          // Wait for CTS
          while(PORTBbits.RB4);

          // Print character to USART2
          putcUSART2(*data);

  } while(*data++);  // Increment pointer and check for NULL
  return;
}

void main(void)
{
  unsigned int result;
  char str[7];
  char crlf[3];

  // Initialize the carriage return/linefeed string
  crlf[0] = 0x0d;
  crlf[1] = 0x0a;
  crlf[2] = 0;

  // Initialize USART2 and the hardware handshaking lines
  PORTBbits.RB5 = 1;
  DDRBbits.RB5 = 0;
```

```
  OpenUSART2(USART_TX_INT_OFF&USART_RX_INT_OFF&USART_ASYNCH_MODE&
           USART_EIGHT_BIT&USART_CONT_RX,25);

  // Initialize the A/D
  OpenADC(ADC_INT_OFF&ADC_FOSC_32&ADC_RIGHT_JUST&ADC_VREF_INT&
        ADC_4ANA_8DIG,ADC_CH0);

  // Print a carriage return and linefeed
  PutsUSART2(crlf);

  while(1)
  {
         ConvertADC();                          // Start a conversion
         while(BusyADC());                      // Wait to complete
         result = ReadADC();                    // Read result
         uitoa(result,str);                     // Convert to ASCII
         PutsUSART2(str);                       // Print string to
USART
         PutsUSART2(crlf);                      // Print crlf to
USART
         Delay10KTCYx(250);                     // Wait a while
         Delay10KTCYx(250);
  }
}
```

Chapter 8. SWITCH.C Description

8.1 Introduction

This chapter describes the demonstration program for the PIC17C756A, SWITCH.C. This program first turns on all the memory mapped LEDs. When the corresponding pushbutton switch is pressed, the microcontroller will toggle the state of the LED.

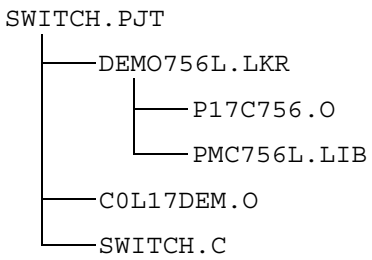
8.2 Highlights

This chapter covers the following topics:

- **MPLAB® IDE Project Files**
- **PICmicro® C Libraries**
- **Source Code Descriptions**
- **SWITCH.C Source Code Listing**

8.3 MPLAB IDE Project Files

The project SWITCH.PJT contains the following files:



The files in this project use the following header files:

P17C756.H

DELAYS.H

8.4 PICmicro C Libraries

The files in SWITCH.PJT use the following library function(s) contained in PMC756L.LIB:

`Delay1KTCYx( )`

8.5 Source Code Descriptions

`DEMO756L.LKR`	This file is the linker script file for the project.
`P17C756.O`	linker script file.
`PMC756L.LIB`	This file contains the library functions compiled for large memory model, it is included by the linker script file.
`C0L17DEM.O`	This file contains the startup code to initialize the stack, initialize data, call the startup function, and jump to `main( ).`
`SWITCH.C`	This file is the main source code file for the project.
`P17C756.H`	This header file contains processor specific items.
`DELAYS.H`	This is the header file for delay routines in the library.

8.6 SWITCH.C Source Code Listing

```
//*****************************************************
//*   SWITCH.C
//*****************************************************
//*   Rodger Richey
//*   Principal Applications Engineer
//*   Microchip Technology Incorporated
//*****************************************************
//*   2 December 1998
//*   Compiled using MPLAB C17 C Compiler V2.20
//*****************************************************
//*   This program toggles the state of the memory
//*   mapped LEDs when the corresponding memory mapped
//*   switch is pressed.
//*****************************************************
//*   Uses an oscillator of 16MHz
//*****************************************************
#include <p17c756.h>
#include <delays.h>

void main(void)
{
  unsigned int LEDbuf;
  unsigned int SWTbuf;
  unsigned int TEMPbuf;
  unsigned int Temp;
  rom int *LEDptr;
  rom int *SWTptr;

  // Initialize pointers to the memory mapped device
  LEDptr = (rom int *)0xfffd;
  SWTptr = (rom int *)0xfffc;

  // Initialize the LEDs and other variables
  *LEDptr = 0x00ff;
  LEDbuf = 0x00ff;
  SWTbuf = 0x00ff;

  while(1)
  {
          // Read the state of the switches
          TEMPbuf = *SWTptr & 0x00ff;

          // Determine what switch states have changed
          Temp = TEMPbuf ^ SWTbuf;

          // If a switch state has changed and the change
          // was a switch press
          if(Temp && TEMPbuf != 0x00ff)
          {
                  // If need to toggle LED low
                  if(Temp&LEDbuf)
                                                  // Mask off desired
LED
                  LEDbuf &= ~Temp;

                  // Else need to toggle LED high
```

```
                else
                                                    // OR in desired LED
                LEDbuf |= Temp;

                // Write new value to LEDs
                *LEDptr = LEDbuf;
            }

            // Update switch state buffer
            SWTbuf = TEMPbuf;

            // Wait for ~16ms
            Delay1KTCYx(64);
        }
    }
```

Chapter 9. I2C.C Description

9.1 Introduction

This chapter describes the demonstration program for the PIC17C756A, I2C.C. This program takes an incrementing count and writes it to the 24LC01B Serial EEPROM and displays the address on the memory mapped LEDs. This program also displays the desired value and the actual value for each location. Before running this program, make sure that S3 positions 1 and 2 are ON and the rest are OFF.

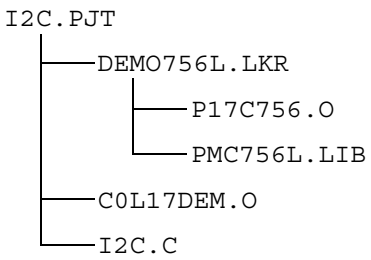
9.2 Highlights

This chapter covers the following topics:

- **MPLAB® IDE Project Files**
- **PICmicro® C Libraries**
- **Source Code Descriptions**
- **I2C.C Source Code Listing**

9.3 MPLAB IDE Project Files

The project I2C.PJT contains the following files:



The files in this project use the following header files:

```
P17C756.H
DELAYS.H
```

9.4 PICmicro C Libraries

The files in I2C.PJT use the following library function(s) contained in PMC756L.LIB:

`Delay1KTCYx( )`

9.5 Source Code Descriptions

`DEMO756L.LKR`	This file is the linker script file for the project.
`P17C756.O`	This file contains the processor dependent objects, it is included by the linker script file.
`PMC756L.LIB`	This file contains the library functions compiled for large memory model, it is included by the linker script file.
`C0L17DEM.O`	This file contains the startup code to initialize the stack, initialize data, call the startup function, and jump to `main( )`.
`I2C.C`	This file is the main source code file for the project.
`P17C756.H`	This header file contains processor specific items.
`DELAYS.H`	This is the header file for delay routines in the library.

9.6 I2C.C Source Code Listing

```
//*****************************************************
//*   I2C.C
//*****************************************************
//*   Rodger Richey
//*   Principal Applications Engineer
//*   Microchip Technology Incorporated
//*****************************************************
//*   2 December 1998
//*   Compiled using MPLAB C17 C Compiler V2.20
//*****************************************************
//*   This program reads and writes to the 24LC01B
//*   Serial EEPROM using the master mode I2C module.
//*   The value of the address is written to each
//*   address of the EEPROM. The program then reads
//*   the value out of the EEPROM and prints both
//*   the desired value and the read value to the
//*   USART2 for display on the Monitor program. The
//*   address is also displayed on the memory mapped
//*   LEDs.
//*****************************************************
//*   Uses an oscillator of 16MHz
//*****************************************************
#include <p17c756.h>
#include <delays.h>
#include <i2c16.h>
#include <usart16.h>
#include <stdlib.h>

void PutsUSART2(char *data);
void PutcUSART2(char data);

// Prints a string to USART2 using hardware handshaking;
void PutsUSART2(char *data)
{
      do
      {
              // Wait for USART2 to finish prev operation
              while(BusyUSART2());

              // Wait for CTS
              while(PORTBbits.RB4);

              // Write a character to USART2
              putcUSART2(*data);

      } while(*data++);  // Increment pointer and check for NULL
      return;
}

// Prints a character to USART2 using hardware handshaking
void PutcUSART2(char data)
{
      while(BusyUSART2());                          // Wait for
USART2 to finish
      while(PORTBbits.RB4);                         // Wait for CTS
```

```
      putcUSART2(data);                              // Print charac-
ter to USART2
      return;
}

void main(void)
{
      rom int *LEDptr;
      unsigned char addr;
      unsigned char byte;
      char str[5];
      char crlf[3];

      // Initialize the carriage return/linefeed string
      crlf[0] = 0x0d;
      crlf[1] = 0x0a;
      crlf[2] = 0;

      // Initialize the pointer to LEDs
      LEDptr = (rom int *)0xfffd;

      // Initialize the I2C module
      OpenI2C(MASTER,SLEW_ON);
      SSPADD = 9;

      // Initialize USART2 and hardware handshaking
      PORTBbits.RB5 = 1;
      DDRBbits.RB5 = 0;

OpenUSART2(USART_TX_INT_OFF&USART_RX_INT_OFF&USART_ASYNCH_MODE&
            USART_EIGHT_BIT&USART_CONT_RX,25);

      // Print a carriage return and linefeed
      PutsUSART2(crlf);

      // Initialize address to 0
      addr = 0;
      while(1)
      {
            *LEDptr = addr;                          // Write address
to LEDs
            // Write a byte to the EEPROM
            EEByteWrite(0xa0,addr,addr);

            // Wait for the EEPROM
            EEAckPolling(0xa0);

            // Read a byte from the EEPROM
            byte = EERandomRead(0xa0,addr);
            ubtoa(addr,str);                         // Convert
address to ASCII
            PutsUSART2(str);                         // Print the
string to USART2
            PutcUSART2(' ');                         // Print a space
to USART2
            ubtoa(byte,str);                         // Convert the
byte to ASCII
            PutsUSART2(str);                         // Print the
string to USART2
```

```
                PutsUSART2(crlf);                       // Print a CRLF
to USART2
                addr++;                                 // Increment the
address
                if(addr > 127)                          // If > 127
                addr = 0;                               //  reset to 0

                Delay10KTCYx(250);                      // Wait a while
        }
}
```

NOTES:

Appendix A. PICDEM 17 Demonstration Board Schematics

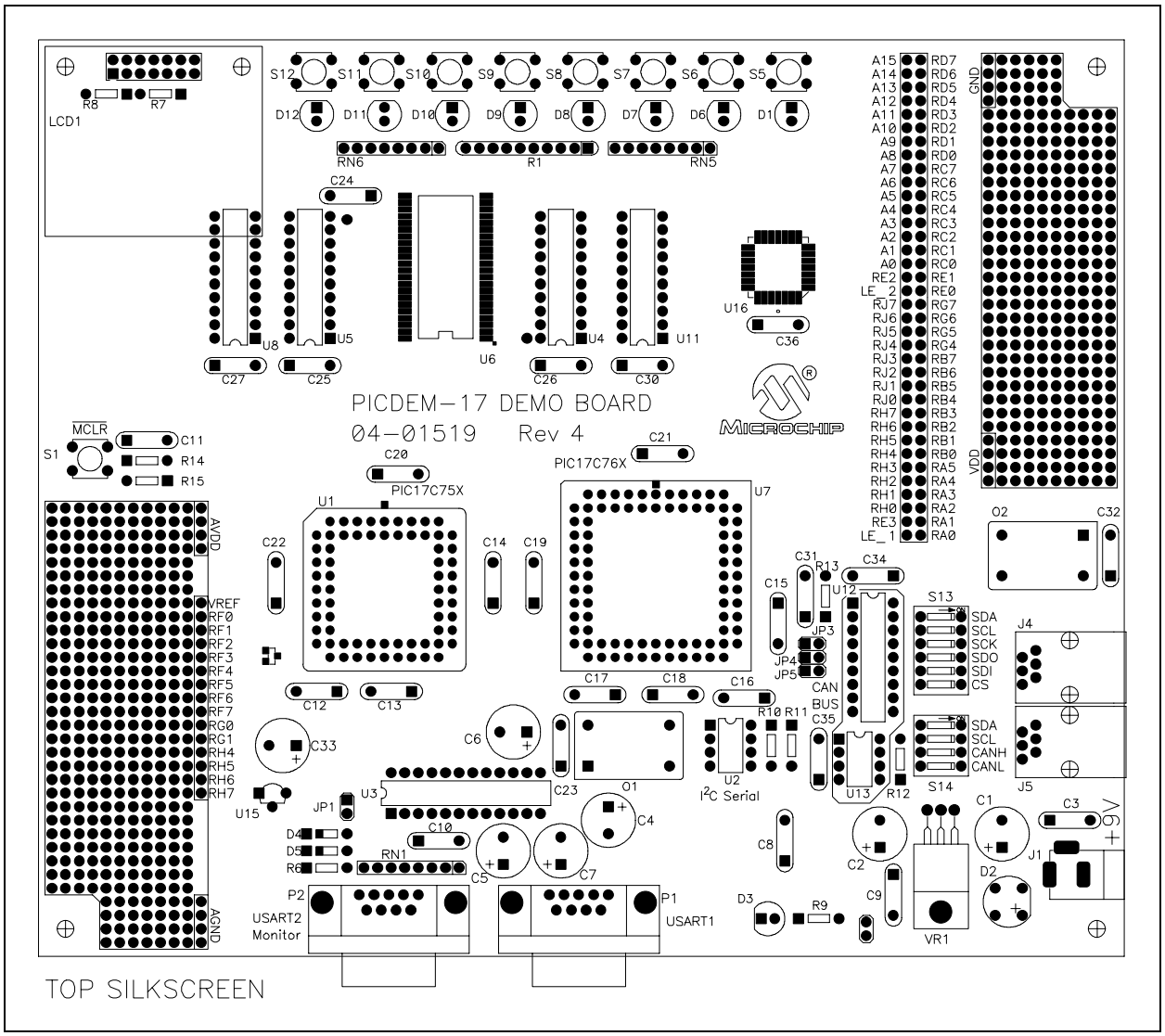
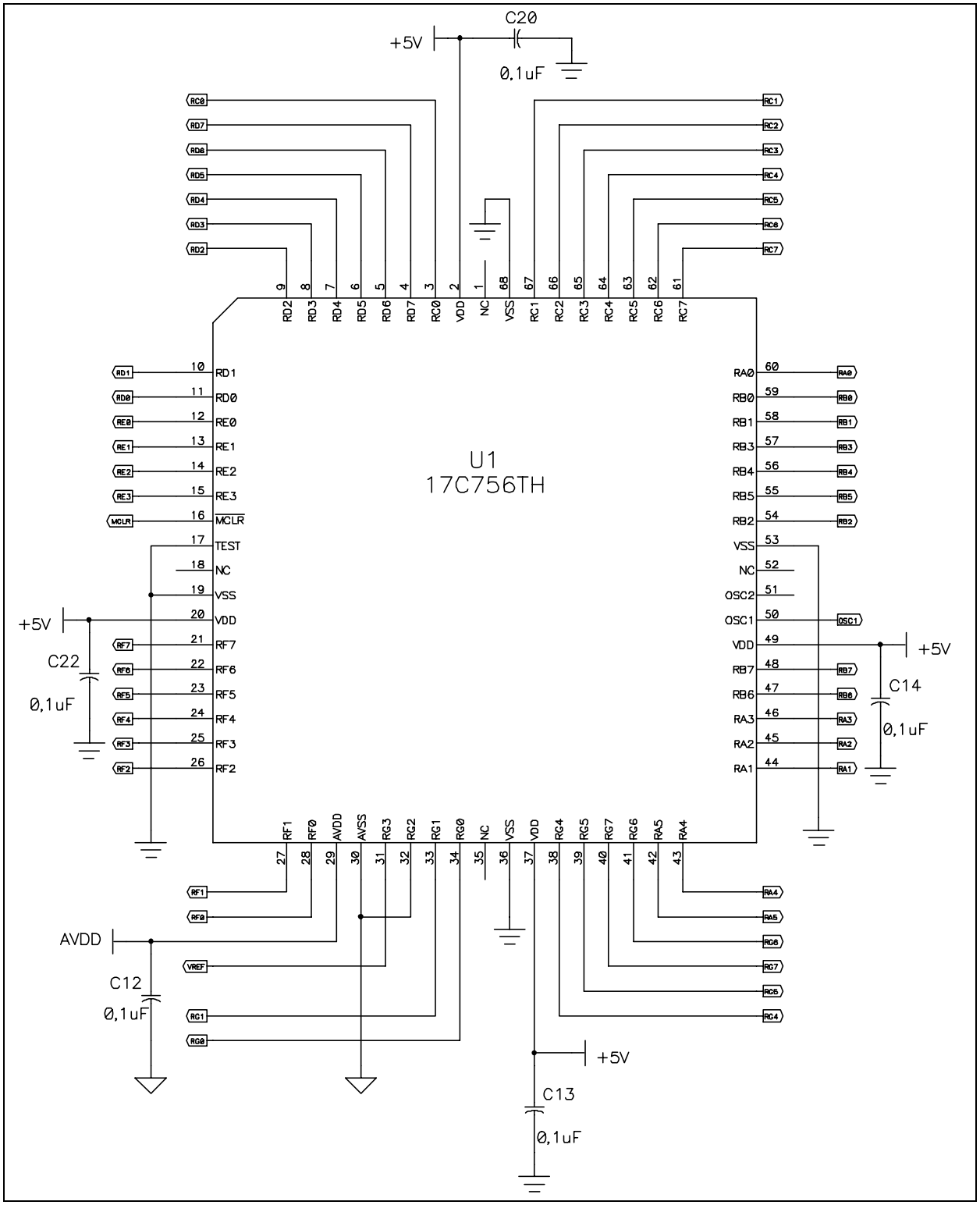
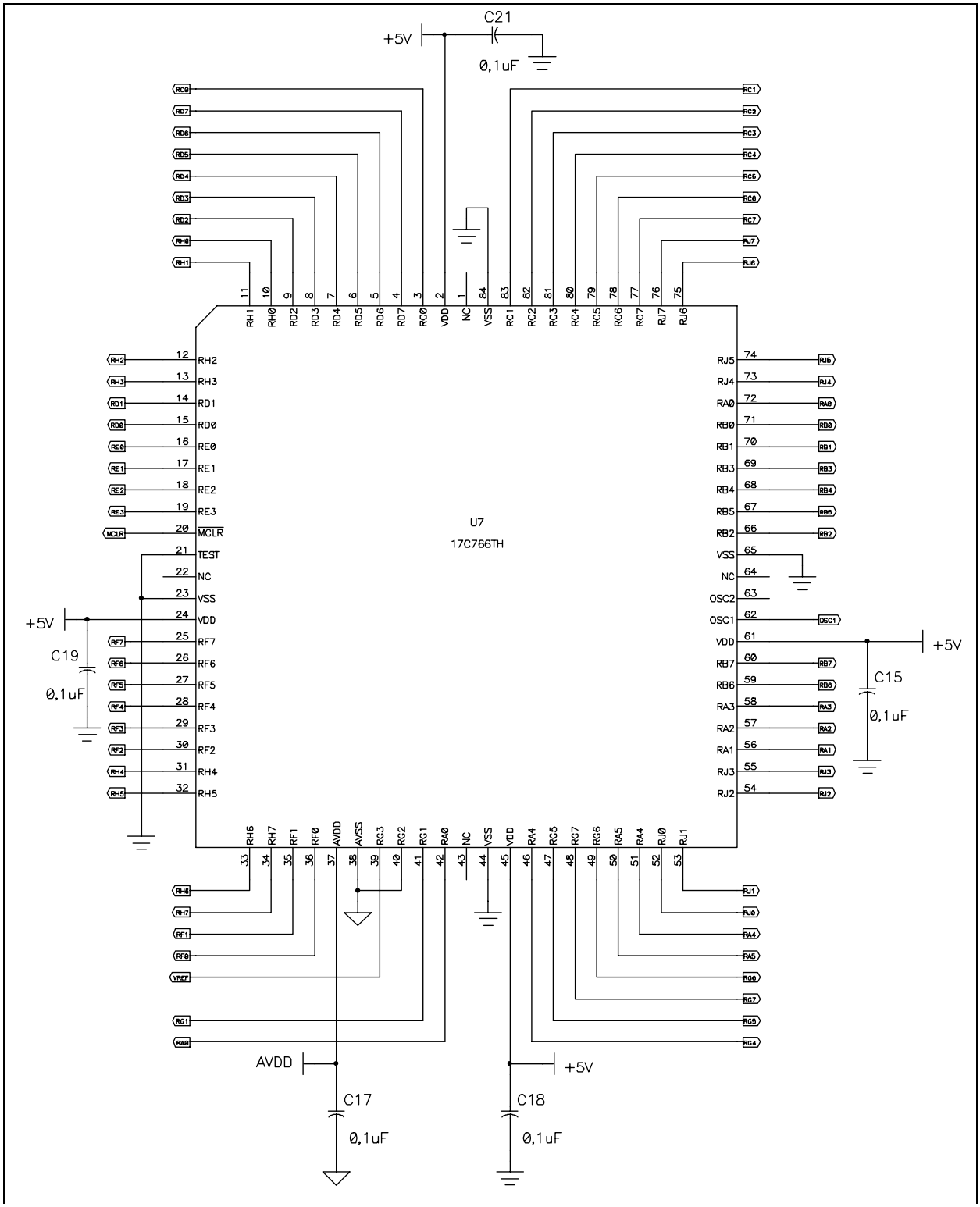


Figure A.1 PICDEM 17 Demonstration Board Parts Layout

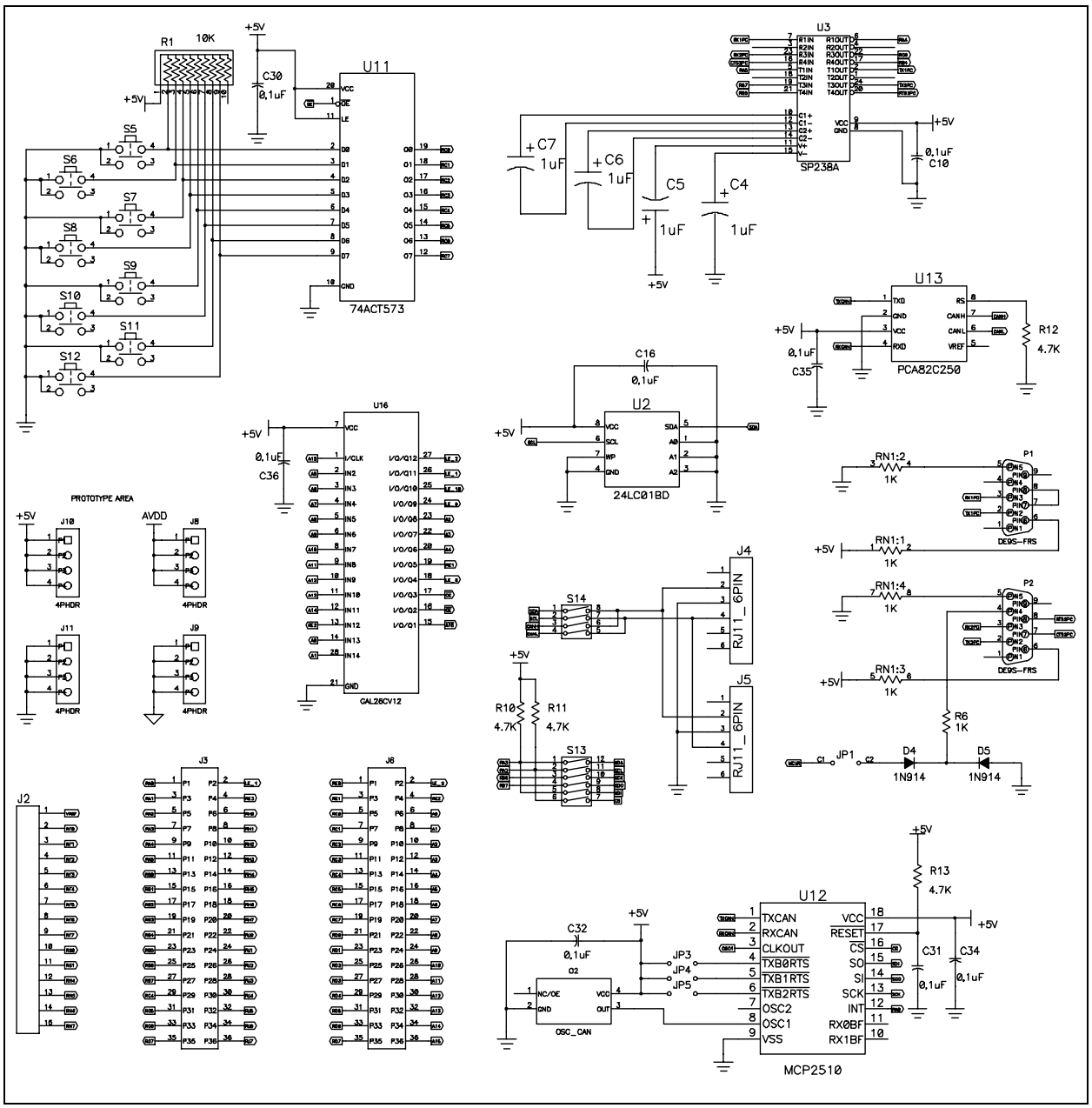
A.1 SCHEMATIC 1



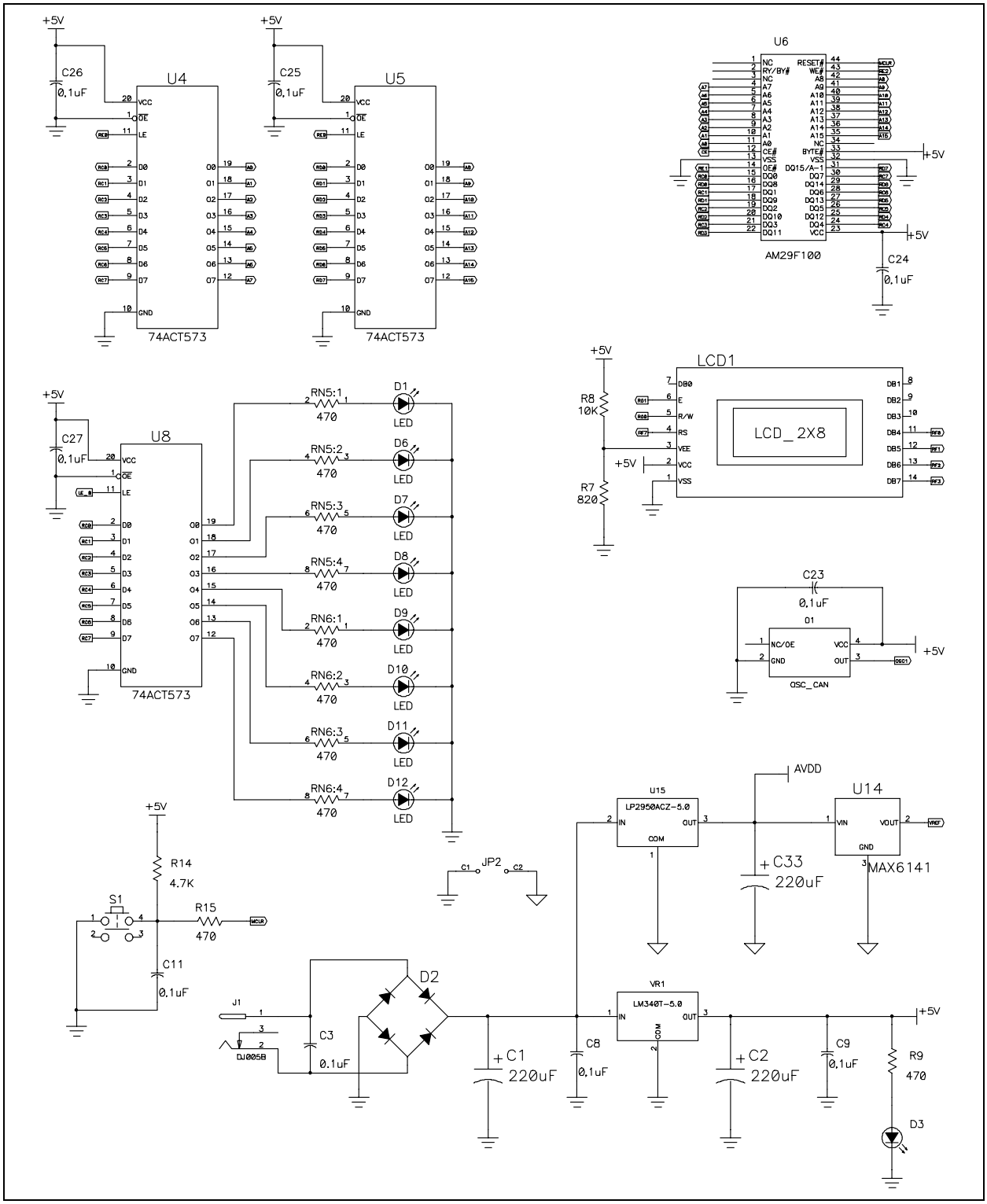
A.2 SCHEMATIC 2



A.3 SCHEMATIC 3



A.4 SCHEMATIC 4



NOTES:

Appendix B. RS-232 Communication Protocol

B.1 INTRODUCTION

This appendix describes the protocol between the PIC17C756A microcontroller and the Host PC.

B.2 OVERVIEW

The PICDEM 17 Monitor program and associated firmware in the PIC17C756A communicate via a RS-232 link between the serial port on the PC and the USART2 module on the PICmicro® MCU. The format is 9600 baud, no parity, 8 data bits, and 1 stop bit. The protocol itself is plain ASCII text, which means that any terminal program can be used to communicate with the firmware. The Monitor program was developed as a convenience to the user. When using a terminal program, the ***H** command will display all the available commands to the user. Descriptions of the operation of the diagnostics are given in Chapter 2.

B.3 DETAILED DESCRIPTION

The following are all the commands and associated descriptions:

- *A Halt Operation
- *DA A/D Diagnostic
- *DC Capture Diagnostic
- *DF Flash Diagnostic
- *DI I^2C Diagnostic
- *DL LCD Diagnostic
- *DP PWM Diagnostic
- *DS Switch Diagnostic
- *DU2 USART2 Diagnostic
- *E Erase FLASH
- *F Download File
- *H Print Help
- *J Run Program
- *RDbxx Read Data Mem
- *Rpyyyy Read Prog Mem
- *WDbxxdd Write Data Mem
- *WPyyyydddd Write Prog Mem

For Data Memory operations:

- b is the bank number
- xx is the data memory address
- dd is the 8-bit data value

For Program Memory operations:

- yyyy is the 16-bit address
- dddd is the 16-bit data

Appendix C. Floppy Disk Contents

C.1 INTRODUCTION

This appendix provides a listing and description of all files included on the 3.5" floppy disk.

C.2 CONTENTS

The floppy disk includes the following files:

- **PICDEM 17 PIC17C756 Monitor Firmware**
  - DIAG.C Diagnostic Routines
  - FLASH.C External FLASH Routines
  - FUNCDEFS.H Function Declarations
  - INTMONL.ASM Interrupt Service Routine
  - 17C756L.LKR Linker Script File
  - TESTLCD.C Main Source Code File
  - UTIL.C Utility Routines
  - XLCD.C External LCD Routines
  - XLCD.H External LCD Header File
  - C0L17.ASM Startup File
  - PMC756L.LIB Library Routines
  - P17C756.ASM Processor Definition File
- **PICDEM 17 PC Monitor Software**
  - 756MON.C Source Code
  - 756MON.H Header File
  - 756MON.RC Resource File
  - 756MON.EXE Executable
- **PICDEM 17 Files**
  - C0L17DEM.ASM Startup File for External Memory programs
  - DEMO756L.LKR Linker Script File for External Memory programs
  - INT756LD.ASM Interrupt Service Routines for External Memory programs
- **Source Code Examples**
  - LCD.C
  - USART.C
  - ANALOG.C
  - SWITCH.C
  - I2C.C
  - XLCD.C
  - XLCD.H

NOTES:

NOTES:

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200 Fax: 480-792-7277
Technical Support: 480-792-7627
Web Address: http://www.microchip.com

Rocky Mountain
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7966 Fax: 480-792-7456

Atlanta
500 Sugar Mill Road, Suite 200B
Atlanta, GA 30350
Tel: 770-640-0034 Fax: 770-640-0307

Austin
Analog Product Sales
8303 MoPac Expressway North
Suite A-201
Austin, TX 78759
Tel: 512-345-2030 Fax: 512-345-6085

Boston
2 Lan Drive, Suite 120
Westford, MA 01886
Tel: 978-692-3848 Fax: 978-692-3821

Boston
Analog Product Sales
Unit A-8-1 Millbrook Tarry Condominium
97 Lowell Road
Concord, MA 01742
Tel: 978-371-6400 Fax: 978-371-0050

Chicago
333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 630-285-0071 Fax: 630-285-0075

Dallas
4570 Westgrove Drive, Suite 160
Addison, TX 75001
Tel: 972-818-7423 Fax: 972-818-2924

Dayton
Two Prestige Place, Suite 130
Miamisburg, OH 45342
Tel: 937-291-1654 Fax: 937-291-9175

Detroit
Tri-Atria Office Building
32255 Northwestern Highway, Suite 190
Farmington Hills, MI 48334
Tel: 248-538-2250 Fax: 248-538-2260

Los Angeles
18201 Von Karman, Suite 1090
Irvine, CA 92612
Tel: 949-263-1888 Fax: 949-263-1338

Mountain View
Analog Product Sales
1300 Terra Bella Avenue
Mountain View, CA 94043-1836
Tel: 650-968-9241 Fax: 650-967-1590

New York
150 Motor Parkway, Suite 202
Hauppauge, NY 11788
Tel: 631-273-5305 Fax: 631-273-5335

San Jose
Microchip Technology Inc.
2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408-436-7950 Fax: 408-436-7955

Toronto
6285 Northam Drive, Suite 108
Mississauga, Ontario L4V 1X5, Canada
Tel: 905-673-0699 Fax: 905-673-6509

ASIA/PACIFIC

China - Beijing
Microchip Technology Beijing Office
Unit 915
New China Hong Kong Manhattan Bldg.
No. 6 Chaoyangmen Beidajie
Beijing, 100027, No. China
Tel: 86-10-85282100 Fax: 86-10-85282104

China - Shanghai
Microchip Technology Shanghai Office
Room 701, Bldg. B
Far East International Plaza
No. 317 Xian Xia Road
Shanghai, 200051
Tel: 86-21-6275-5700 Fax: 86-21-6275-5060

Hong Kong
Microchip Asia Pacific
RM 2101, Tower 2, Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2401-1200 Fax: 852-2401-3431

India
Microchip Technology Inc.
India Liaison Office
Divyasree Chambers
1 Floor, Wing A (A3/A4)
No. 11, O'Shaugnessey Road
Bangalore, 560 025, India
Tel: 91-80-2290061 Fax: 91-80-2290062

Japan
Microchip Technology Intl. Inc.
Benex S-1 6F
3-18-20, Shinyokohama
Kohoku-Ku, Yokohama-shi
Kanagawa, 222-0033, Japan
Tel: 81-45-471- 6166 Fax: 81-45-471-6122

Korea
Microchip Technology Korea
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea
Tel: 82-2-554-7200 Fax: 82-2-558-5934

ASIA/PACIFIC (continued)

Singapore
Microchip Technology Singapore Pte Ltd.
200 Middle Road
#07-02 Prime Centre
Singapore, 188980
Tel: 65-334-8870 Fax: 65-334-8850

Taiwan
Microchip Technology Taiwan
11F-3, No. 207
Tung Hua North Road
Taipei, 105, Taiwan
Tel: 886-2-2717-7175 Fax: 886-2-2545-0139

EUROPE

Australia
Microchip Technology Australia Pty Ltd
Suite 22, 41 Rawson Street
Epping 2121, NSW
Australia
Tel: 61-2-9868-6733 Fax: 61-2-9868-6755

Denmark
Microchip Technology Denmark ApS
Regus Business Centre
Lautrup hoj 1-3
Ballerup DK-2750 Denmark
Tel: 45 4420 9895 Fax: 45 4420 9910

France
Arizona Microchip Technology SARL
Parc d'Activite du Moulin de Massy
43 Rue du Saule Trapu
Batiment A - ler Etage
91300 Massy, France
Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79

Germany
Arizona Microchip Technology GmbH
Gustav-Heinemann Ring 125
D-81739 Munich, Germany
Tel: 49-89-627-144 0 Fax: 49-89-627-144-44

Germany
Analog Product Sales
Lochhamer Strasse 13
D-82152 Martinsried, Germany
Tel: 49-89-895650-0 Fax: 49-89-895650-22

Italy
Arizona Microchip Technology SRL
Centro Direzionale Colleoni
Palazzo Taurus 1 V. Le Colleoni 1
20041 Agrate Brianza
Milan, Italy
Tel: 39-039-65791-1 Fax: 39-039-6899883

United Kingdom
Arizona Microchip Technology Ltd.
505 Eskdale Road
Winnersh Triangle
Wokingham
Berkshire, England RG41 5TU
Tel: 44 118 921 5869 Fax: 44-118 921-5820

01/09/01