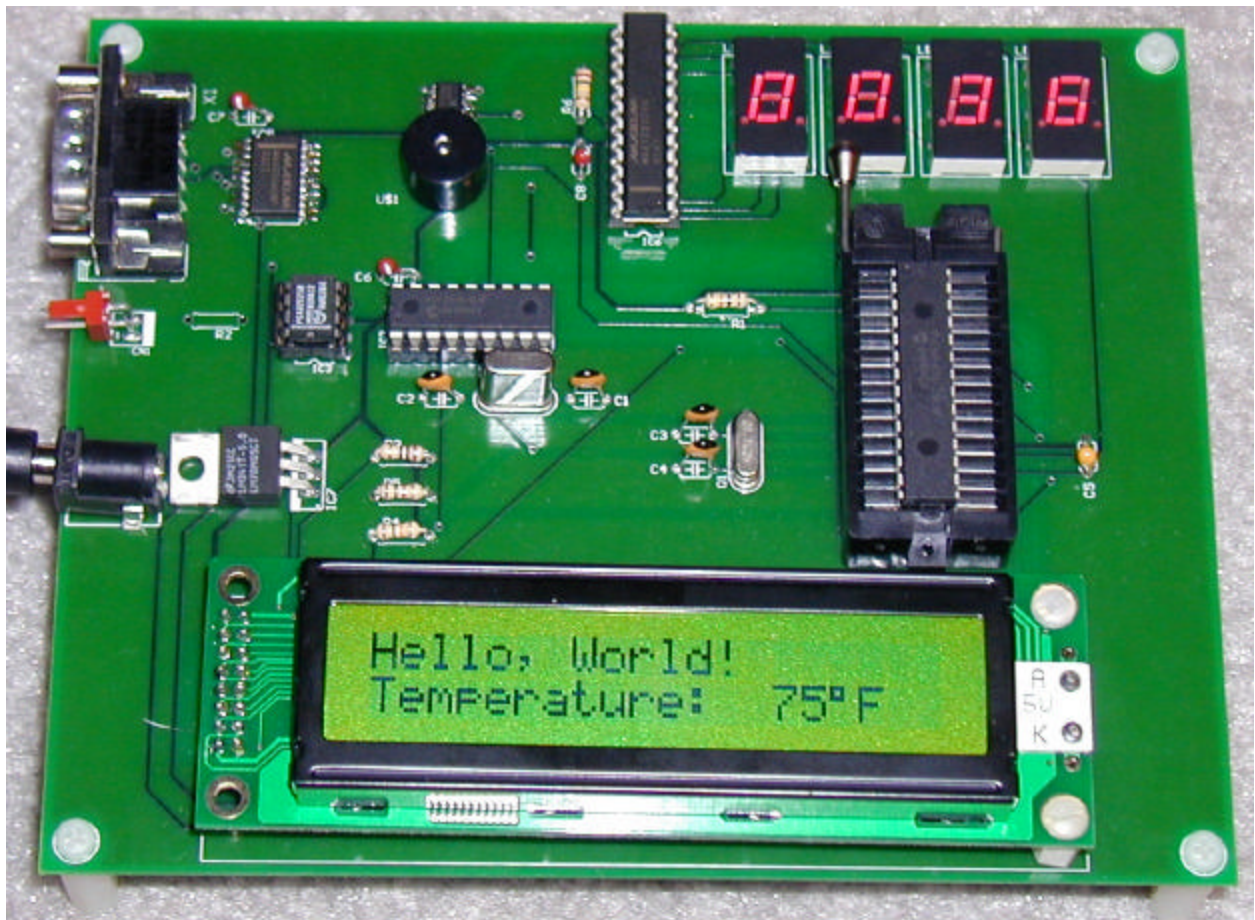


# Microchip PIC16F876 Development Board

## User Manual



# Table of Contents

<b>1. Overview .....</b>	<b>3</b>
<b>2. Board Design .....</b>	<b>4</b>
<b>A. Functional Design.....</b>	<b>4</b>
<b>B. Schematics &amp; Design Parts List.....</b>	<b>5</b>
<b>C. Layout .....</b>	<b>10</b>
<b>3. Board Operation .....</b>	<b>11</b>
<b>A. Getting Started .....</b>	<b>11</b>
<b>B. Component Description.....</b>	<b>12</b>
<b>C. Getting Input .....</b>	<b>15</b>
<b>D. PIC Programming.....</b>	<b>16</b>
<b>4. Application Development .....</b>	<b>17</b>
<b>A. Development Languages &amp; Tools .....</b>	<b>17</b>
<b>B. Software Libraries .....</b>	<b>18</b>
<b>C. Initialization &amp; Diagnostics Routine.....</b>	<b>36</b>
<b>5. Additional Resources .....</b>	<b>39</b>

## 1. Overview

The PIC16F876 Development Board was designed as part of a Senior Design project in Electrical and Computer Engineering at Iowa State University. The purpose of this project was to develop a low-cost development board that future Senior Design teams could use to develop a PIC microcontroller-based solution for their own projects. This board was designed for the 16F876, a 28-pin PIC that should provide an appropriate level of performance and versatility for most design projects.

The board contains several useful components, including:

- 28-pin ZIF (zero input force) socket to easily remove and replace the PIC;
- RS-232 serial port;
- CAN (controller area network) interface;
- 4 7-segment LEDs;
- 20x2 character LCD;
- temperature sensor;
- speaker;
- AC adapter for providing power to the board with a standard wall socket.

The schematics showing the connections of these components are given in *Section 2: Board Design*. The operation of the development board and of each of these subsystems is described in detail in *Section 3: Board Operation*.

A PIC programmer can be purchased separately from the development board to program and re-program code into the non-volatile Flash memory of the PIC. The P16PRO40 programmer was used in the development of this project. Both the programmer and the development board have ZIF sockets for mounting the PIC, allowing the user to easily swap the PIC between the two units. Usage of the programmer is also described in *Section 3: Board Operation*.

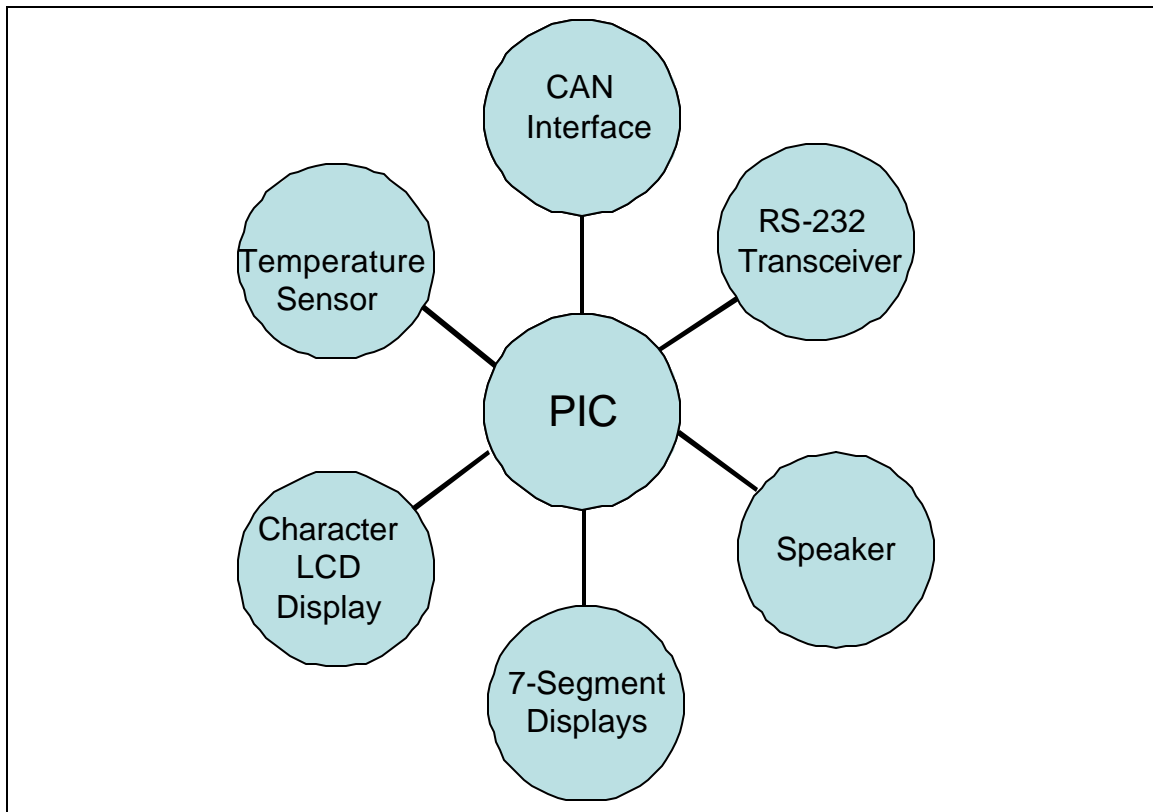
Along with the board hardware, software libraries have been written for each of the board components. These libraries take care of the basic functionality for the hardware on the board, allowing the user to concentrate on writing code for his or her specific application. Software for the PIC can be written in assembly or in another higher-level language (e.g. C). The HI-TECH Software ANSI C PICC Compiler has been obtained for use in the department, and can be used for PIC software development for any Senior Design teams. Information on the software modules included with the board and guidelines for developing your own software can be found in *Section 4: Application Development*.

*Note: This User Manual assumes that the reader has some background experience with microcontrollers and low-level hardware/software development. The source code that was written for the development board is in C, as well as all of the code examples. All Senior Design students should have the knowledge necessary to understand and use this manual.*

## 2. Board Design

### A. Functional Design

From a functional perspective, the board components are arranged as shown in Figure 2.1. The PIC is the centerpiece of the board, and all other components communicate with the microcontroller. The PIC accepts input from the RS-232 transceiver, from the CAN interface, and from the temperature sensor. It can also send output to the RS-232 transceiver, the CAN interface, the speaker, the 7-segment displays, and the character LCD display.



**Figure 2. 1. Functional block diagram.**

## B. Schematics & Design Parts List

At a physical level, many components are needed in the board design to enable the functionality described in the previous section. A complete listing of these components is given in Table 2.1. Every part required to build a PIC16F876 Development Board is included in this table.

**Table 2. 1. Design parts list.**

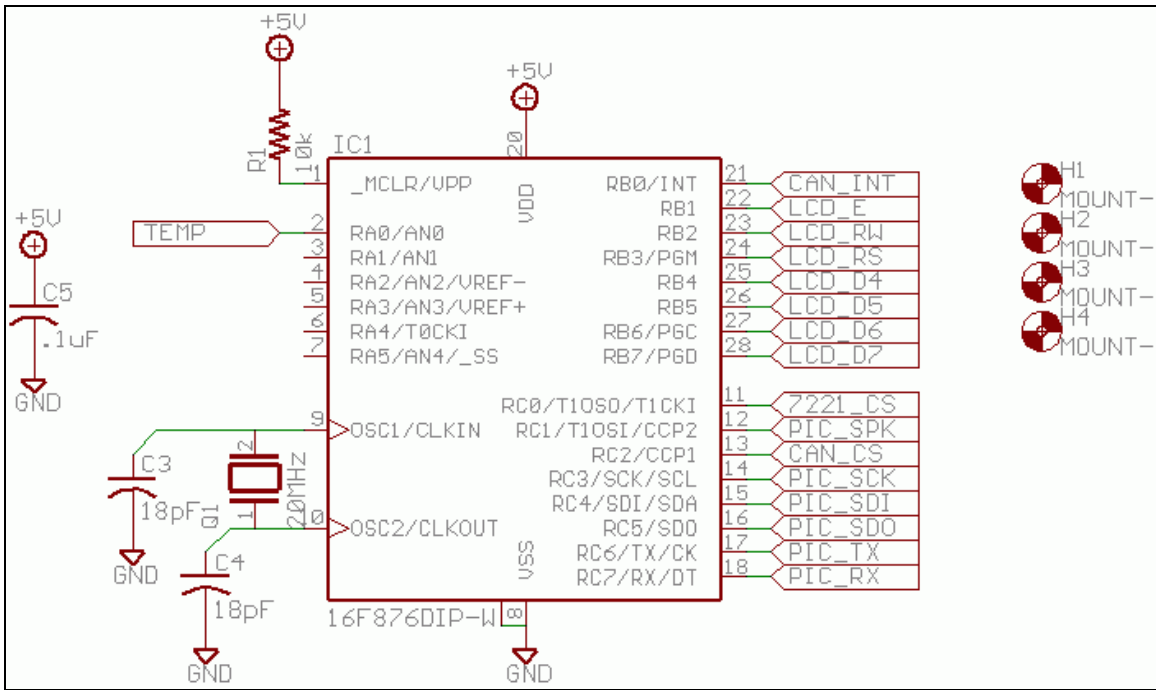
Part	Value	Part	Value
C1	18pF	LCD	LCD
C2	18pF	LED1	DIG0
C3	18pF	LED2	DIG1
C4	18pF	LED3	DIG2
C5	.1uF	LED4	DIG3
C6	.1uF	LED5	Power LED
C7	.1uF	Q1	20MHz
C8	.1uF	Q2	20MHz
C9	33uF	R1	10k
CN1	CAN Connector	R2	120
IC1	16F876	R3	10
IC2	MCP2510	R4	10k
IC3	PCA82C250	R5	1k
IC4	LM34	R6	100k
IC5	MAX233A	R7	440
IC6	MAX7221	US1	SPEAKER1
IC7	LM78M05	X1	Serial Connector
J1	POWER		

Schematics and layout for the circuit board were done using CadSoft EAGLE. The schematics on the following pages show the circuit diagrams for each of the major subsystems on the board. Schematics are included for the following subsystems:

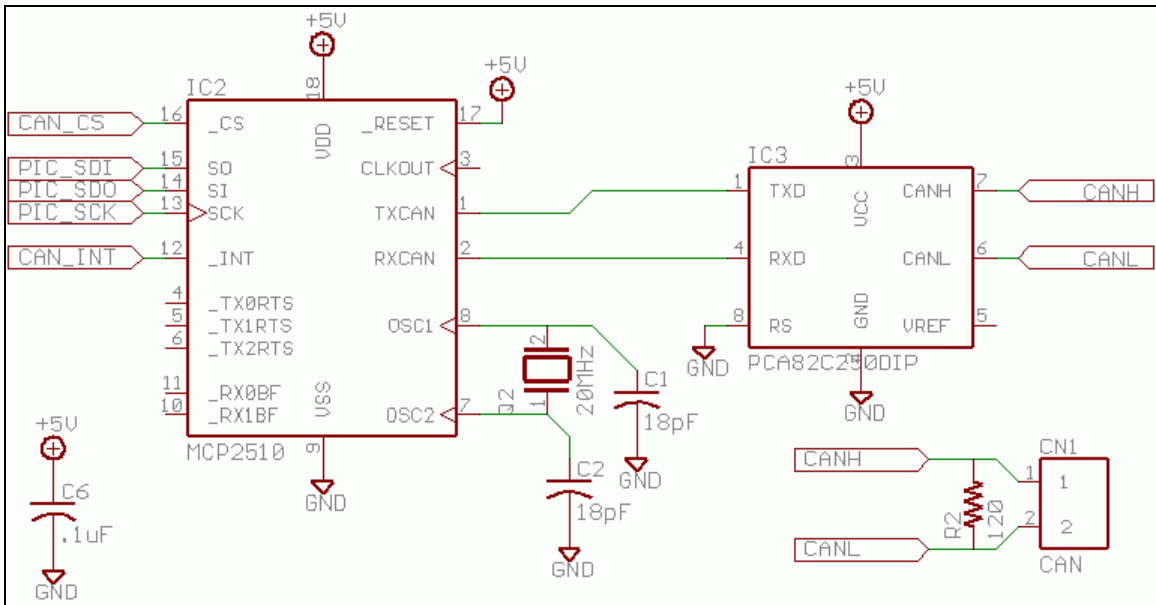
- PIC16F876 (Figure 2.2);
- CAN interface, with the MCP2510 and the PCA82C250 (Figure 2.3);
- 7-segment LEDs, including the MAX7221 display controller (Figure 2.4);
- RS-232 serial interface, using the MAX233A line driver/receiver (Figure 2.5);
- LM34 temperature sensor (Figure 2.6);
- power supply and power system (Figures 2.7 and 2.8).

For descriptions of the operation of all of these subsystems, please refer to *Section 3: Board Operation*. For more detail on the design of the board, please refer to the final report on the Senior Design project website.

# Microchip PIC16F876 Development Board User Manual

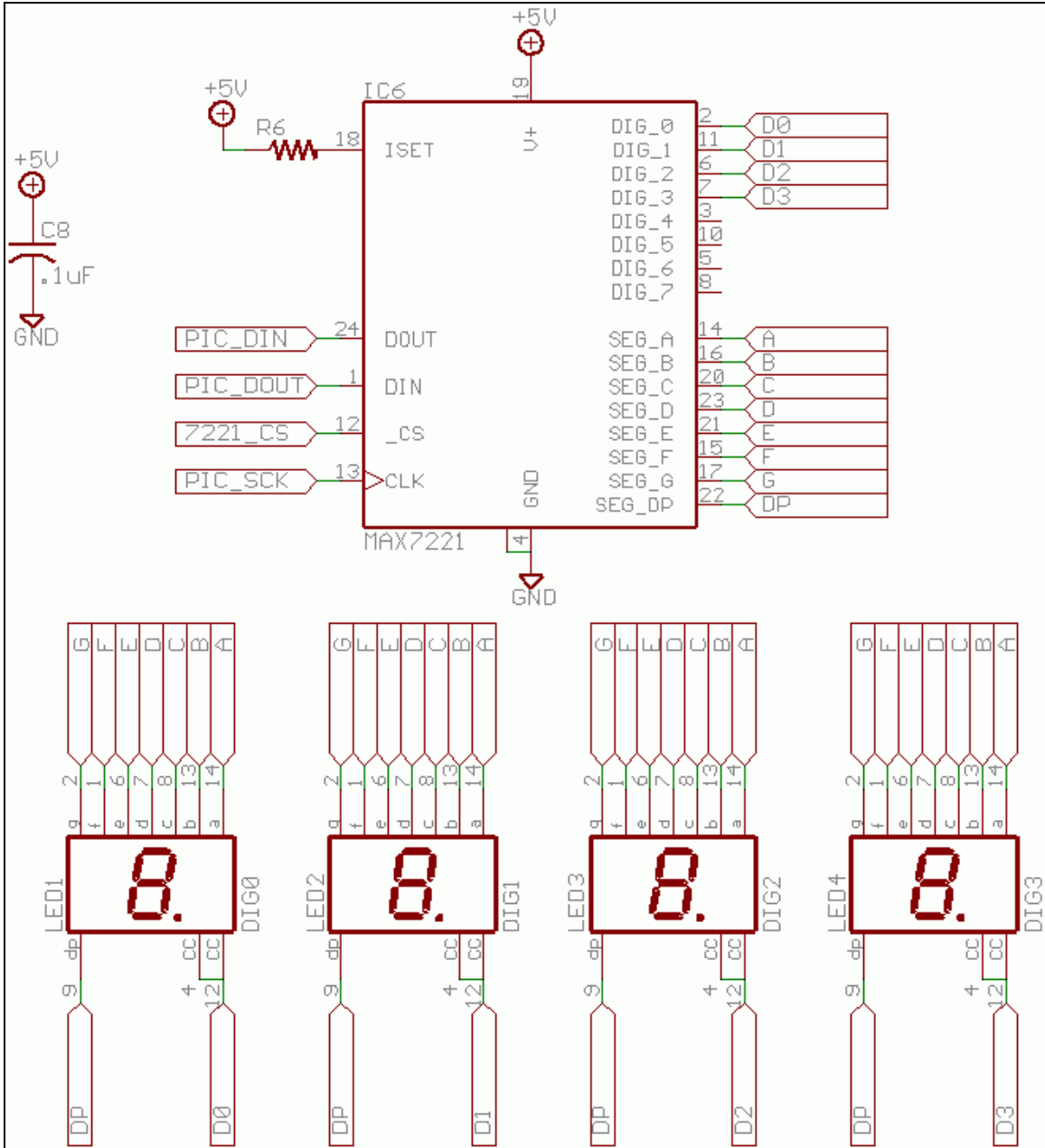


**Figure 2. 2. PIC pin schematic.**



**Figure 2. 3. CAN interface schematic.**

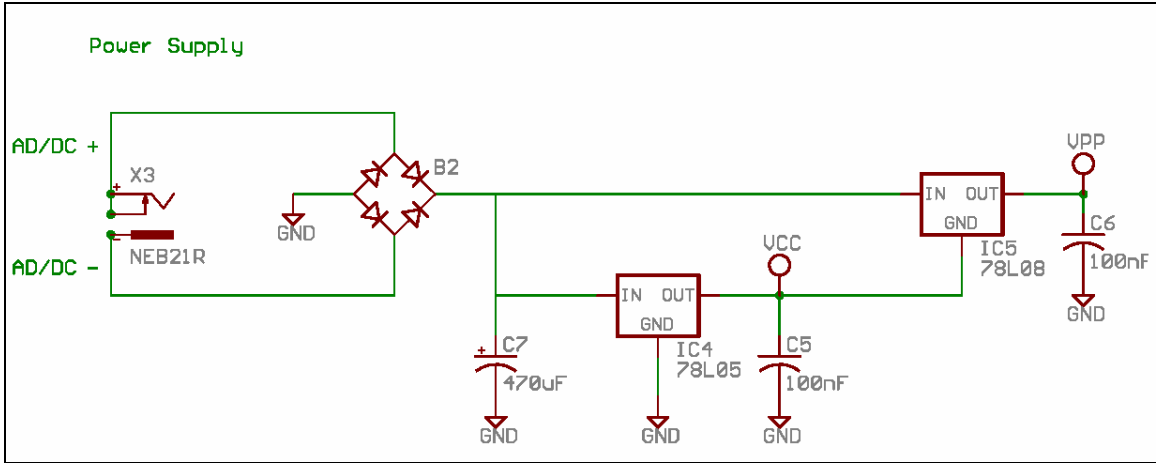
# Microchip PIC16F876 Development Board User Manual



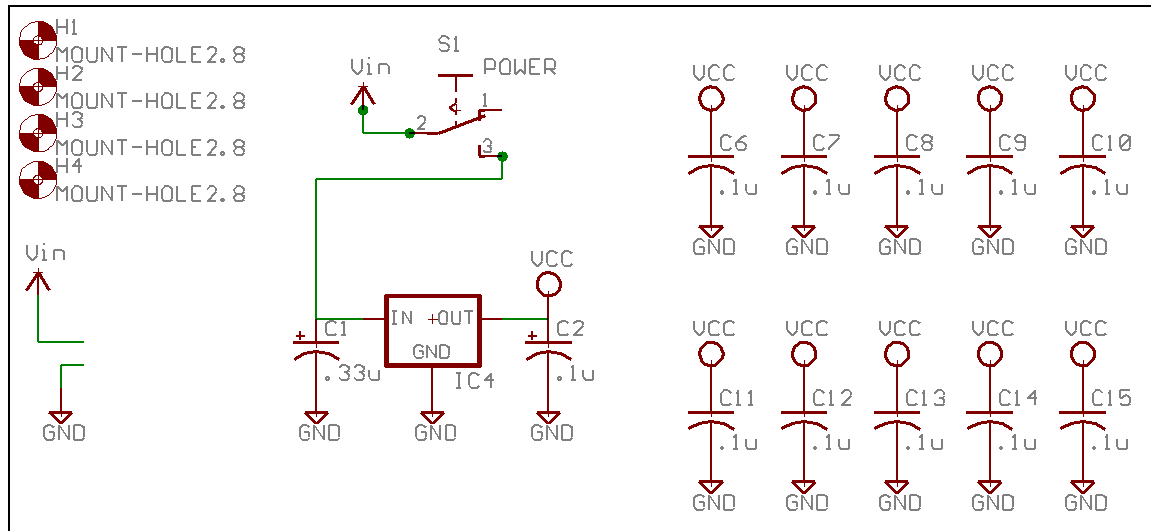
**Figure 2. 4. 7-segment display schematic.**



# Microchip PIC16F876 Development Board User Manual



**Figure 2. 7. Power supply schematic.**



**Figure 2. 8. Power system schematics.**



## 3. Board Operation

### A. Getting Started

The PIC16F876 Development Board plugs into a standard wall socket with the included AC adapter. The board does not include an on/off switch, so the PIC begins executing the code in its instruction memory immediately after it receives power. Obviously, if no code has yet been programmed into the PIC, the board will not do anything after being plugged in. Instructions for programming the PIC are given in *Section 3.D: PIC Programming*.

An initialization routine has been written to perform the necessary setup procedures and execute simple tests to verify that the board is working properly. This routine (or at least the vital parts of it) should be programmed into the PIC and performed every time the board is started to ensure proper operation. It currently performs the following operations:

- selects the HS (high-speed crystal) oscillator type;
- enables the power-up timer;
- disables the watchdog timer;
- disables low-voltage programming;
- sets up the I/O pins;
- configures the ADC (analog-to-digital converter);
- turns the speaker on and emits a start-up beep;
- initializes and writes a test string to the LCD;
- initializes and tests the 7-segment LEDs with a simple count-down;
- reads the temperature and displays it on the LCD.

The source code for this routine can be found in *Section 4.C: Initialization & Diagnostics Routine*. It can be modified as necessary to perform any other initializations that might be necessary for your particular application, or to perform additional functionality tests. For example, it might be beneficial to include functions to test for proper operation of the CAN and RS-232 interfaces.

## B. Component Description

**PIC16F876:** The Microchip PIC16F876 is a 28-pin microcontroller with a number of features that would be helpful in providing a microcontroller-based solution for a wide variety of Senior Design projects. Some of these features include:

- easy-to-learn instruction set (only 35 single word instructions);
- 20 MHz operating frequency;
- 8192 14-bit words of Flash program memory;
- 368 bytes of data memory;
- 256 bytes of additional EEPROM data memory;
- 13 programmable interrupts;
- SPI, I<sup>2</sup>C, and USART serial I/O;
- 22 I/O ports;
- 5 10-bit analog-to-digital channels;
- 3 timers;
- 2 capture/compare modules;
- PWM (pulse width modulation);
- 8-level hardware stack.

The 16F876 is well suited to a wide variety of advanced automotive, industrial, appliance and consumer applications. Its program memory space is implemented with electrically re-programmable Flash memory, which is very desirable for making quick code changes during the project development stages. The PIC orchestrates the operation of the rest of the components on the board, which provide input and output for the microcontroller.

Code for the PIC can be written using either the native assembly language or another language such as C. More information on the software libraries that have already been written for the board, as well as guidelines for writing your own software, can be found in *Section 4: Application Development*.

The development board does not include an on-board programmer for the PIC. The PIC can be programmed with a separate programmer module. Fairly inexpensive programmers can be purchased from a number of sources. The programmer that was used for the development of this board was the P16PRO40. This programmer is available for use by any Senior Design team in the Senior Design labs. Instructions on how to use this programmer are given in *Section 3.D: PIC Programming*.

If much of the PIC's functionality seems foreign to you, don't despair! For more in-depth information on using the PIC and its many capabilities, the best place to go is the Microchip website at <http://www.microchip.com>. In addition to a datasheet for the 16F876, the site has a number of tutorials and other documentation to help learn how to configure the PIC and use its numerous

Microchip PIC16F876 Development Board  
User Manual

functions. There are also several other user-friendly resources on the Internet that introduce beginners to the basics of using the PIC. Some of these sites are listed in *Section 5: Additional Resources*.

**RS-232 serial port:** RS-232 is a standard bus protocol used for serial communications, most notably for communicating with personal computer serial ports. The PIC16F876 Development Board includes a RS-232 port for communicating with a computer or any other device that communicates via RS-232. In this way, a program such as HyperTerminal could be used to provide input to the PIC when connected to a computer via a standard serial cable. Another possible use might be to connect a serial keyboard to the board as another means of providing input.

The PIC is able to communicate via RS-232 through the on-board MAX233A serial transceiver. This device connects to the serial port, and generates and receives the signals used in the RS-232 protocol. It also provides the conversion from the board's 5V power supply to the +/-12V needed to communicate with RS-232.

The MAX233A is sold by Maxim, and its data sheet can be found at <http://www.maxim-ic.com>. The website also includes several application notes about working with the RS-232 protocol.

**CAN interface:** CAN (Controller Area Network) is a serial bus protocol that is widely used in real-time control applications. It was originally developed for use in automotive electronics systems, but its use has since extended to machine automation and control, building automation, medical applications, the railway industry, and many other areas. The CAN bus protocol can operate at speeds up to 1 Mbit per second, has excellent error detection and correction capabilities, and is very robust in poor electrical environments.

The PIC16F876 Development Board includes an interface for communicating with other devices over a CAN network. This interface is comprised of the Microchip MCP2510 and the Philips PCA82C250. The MCP2510 is an 18-pin CAN controller that communicates with the PIC over its SPI serial interface. This chip provides the logic necessary for receiving and transmitting messages over a CAN bus. It includes three transmit and two receive buffers, reducing the amount of management needed from the PIC. This device interfaces with the CAN bus through another chip, the PCA82C250. This chip provides the differential transmit and receive capabilities for the MCP2510 controller.

There are a few different resources for learning more about the CAN protocol. The Microchip website has an application note (AN713) that describes the basics of CAN. Another introduction to the CAN protocol can be found at <http://www.can-cia.de/can/protocol/>.

Microchip PIC16F876 Development Board  
User Manual

**7-segment LEDs:** The board includes four simple 7-segment LEDs for digital output. These LEDs can be used for a variety of functions, including a system clock, a counter, or any other application where some sort of digital feedback would be useful. They are controlled with the Maxim MAX7221 LED display driver. Like the CAN controller, it communicates with the PIC using an SPI serial interface. This device is a common-cathode display driver that operates serially at 10 MHz. It includes a number of features for easy and flexible control of the LEDs. More detailed information on this device can be found in its datasheet and supporting documentation at <http://www.maxim-ic.com>.

**Character LCD:** The character LCD is a standard display that is capable of providing two rows and twenty columns of standard ASCII text. This device interfaces directly with the PIC using seven of the microcontroller's pins. Software libraries allow the user to easily initialize the display, clear the display, write a single character, write a string, and send other commands to control the display's operation. More information on using the LCD is given in *Section 4: Application Development*.

**Temperature sensor:** The National Semiconductor LM34 is a precision temperature sensor. Its output is calibrated to be linearly proportional to degrees Fahrenheit, at a scale factor of 10.0 mV/deg F. It is rated over a very wide range of -50 to 300 deg F, enabling its use in a wide variety of applications. This device interfaces directly with one I/O pin on the PIC. The voltage input from the temperature sensor can be converted to a digital value using the PIC's ADC. An example of its use is included in the initialization routine given in *Section 4: Application Development*.

**Speaker:** The piezoelectric speaker provides rudimentary audio output for the board. It interfaces directly with a single pin on the PIC. The initialization routine in *Section 4: Application Development* includes a simple illustration of its use.

**Voltage regulator:** The National LM78M05 is a 3-terminal positive voltage regulator. It regulates the voltage from the external power supply down to the 5V supply required by the board.

## C. Getting Input

User input to the PIC can be provided by two possible means: with the CAN interface, or via the RS-232 serial port. Both options allow the board to interface with a host of possible external devices. One of the most convenient options is to connect the board directly to a personal computer with a serial cable. Data can easily be sent to or received from the board using a program such as HyperTerminal.

These simple steps will enable the board to communicate with a Windows PC:

- 1) Using a standard serial cable, connect the board to the serial port on the back of your PC.
- 2) Open HyperTerminal (Start -> Programs -> Accessories -> Communications).
- 3) The program should open a *New Connection...* dialog box. (If not simply select that command from the File menu.) Enter a name for the connection.
- 4) Specify COM1 (or whatever serial port the board is connected to) in the *Connect using* drop-down box.
- 5) Specify the proper port settings:
  - o Baud rate as specified in software on the PIC
  - o 8 data bits
  - o No parity
  - o 1 stop bit
  - o Hardware flow control
- 6) That's it! The program should now be ready to communicate with the PIC.

## D. PIC Programming

In order for the board to do anything, the PIC's Flash instruction memory obviously must be programmed with code for the PIC to execute. The PIC16F876 Development Board does not include an on-board programmer to do this; a separate programmer module must be used. A number of inexpensive programmers can be purchased or easily constructed. One readily available programmer is the P16PRO40; one of these is available for use in the Senior Design lab in Town Hall.

The P16PRO40 includes a ZIF socket for the PIC, and it interfaces with the PC via the parallel port. This programmer takes the compiled hex code from the PC and programs it into the PIC. An additional PC software package is necessary to interface with the PIC programmer. One such program is WinPicProg, available free of charge at <http://www.winpicprog.co.uk>. The necessary settings for this program to function properly with the P16PRO40 are shown in the screenshot in Figure 3.1.

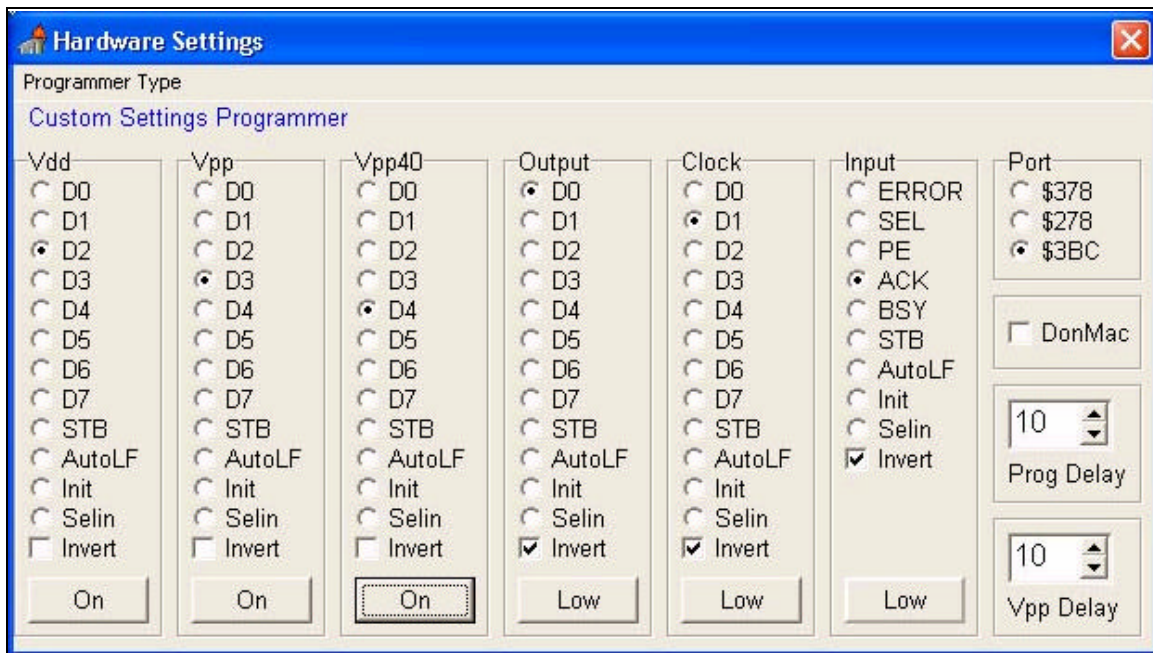


Figure 3. 1. WinPicProg settings.

## 4. Application Development

### A. Development Languages & Tools

It is possible to develop software applications using either the PIC's native Assembly language or C. The necessary IDE (Integrated Development Environment) packages for either language are readily available. The best language to use depends on the specific application. Here are some of the advantages and disadvantages to using either language:

#### Assembly

- + Good for writing efficient and compact code (e.g. for real-time applications)
- + Small instruction set is easy to learn
- Longer development time
- Must learn the PIC instruction set
- Code is less readable

#### C

- + No need to learn another programming language
- + Quick development time
- + Code is easy to read and document
- Code may be less efficient - not as much low-level control

The IDE used for developing Assembly code is MPLAB, Microchip's proprietary IDE supporting all of the company's PIC devices. It can be downloaded for free from the Microchip website. The toolset has most of the standard features for writing and debugging code found in other IDEs, including a color-coded source code text editor, multiple project workspaces, breakpoints, etc.

The PICC C compiler from HI-TECH Software is the most readily available package for C programming for the PIC. This compiler supports full ANSI C, with the exception of recursion (the PIC's stack is too small to support this). It is also fully compatible with the MPLAB IDE, for easy integration of C and Assembly code. This software package, which normally costs near \$1000, is available for free student use in the Senior Design lab.

Once the source code in either Assembly or C has been compiled and assembled into hex data, it is ready to be programmed into the PIC. This can be done using the procedure previously described in *Section 3.D: PIC Programming*.

## B. Software Libraries

Software libraries have been written to provide basic functionality for the CAN controller, the LCD, the 7-segment displays, and the RS-232 serial port. There are also libraries that implement delays for proper timing and configure the SPI port on the PIC.

The next few pages of this manual contain the source code for these libraries. These libraries can simply be included as a header file in any C code that is written for applications using these devices. The functions included in each of these libraries are briefly described here. The comments within the source code should be sufficient to determine how to properly use these functions.

### **mcp2510.h** – Functions for using the MCP2510 (CAN controller)

- **MCP2510\_SETSPI()**  
Sets up SPI port for use w/the MCP2510
- **MCP2510\_STATUS(port, mask)**  
Returns the MCP2510 status byte
- **MCP2510\_WRITE(port, mask, addr, data)**  
Writes data to register at addr
- **MCP2510\_WRITE\_SEQUENCE(port, mask, startaddr, data, len)**  
Writes data of length len starting at startaddr
- **MCP2510\_READ(port, mask, addr)**  
Returns data at register addr
- **MCP2510\_RTS(port, mask, rtsnum)**  
Request-to-send, transmits packet. rtsnum = RTS0, RTS1 or RTS2
- **MCP2510\_BITMODIFY(port, mask, addr, bitmask, data)**  
Sets or resets data's bits in addr, masked by bitmask
- **MCP2510\_RESET(port, mask)**  
A software reset of the MCP2510
- **MCP2510\_TRANSMIT\_STD(port, mask, id, dlc, data)**  
Transmit a standard data frame
- **MCP2510\_TRANSMIT\_EXT(port, mask, id, eid, dlc, data)**  
Transmit an extended data frame
- **MCP2510\_TRANSMIT\_REM(port, mask, id, eid)**  
Transmit a remote data frame

### **lcd.h** – Functions for controlling and sending output to the LCD

- **LCD\_OUT\_COMMAND(unsigned char cmd)**  
Sends a command byte to the LCD
- **LCD\_OUT\_UPPER\_NYBBLE(unsigned char nybble)**  
Sends a command nybble (4 bits) to the LCD
- **LCD\_CLEAR()**  
Clear the display

Microchip PIC16F876 Development Board  
User Manual

- LCD\_HOME()  
Home the cursor
- LCD\_SET\_CGRAM(x)  
Set the CGRAM to x
- LCD\_SET\_DDRAM(x)  
Set the DDRAM to x
- LCD\_OUT\_DATA(unsigned char data)  
Sends a data byte to the LCD
- LCD\_INITIALIZE(unsigned char lines)  
Initializes the display
- LCD\_PUTS(const char\* s, unsigned char len)  
Outputs a string to the LCD

**max7221.h** – Functions for controlling the 7-segment display driver

- MAX7221\_SETSPI()  
Sets up SPI port for use w/the MAX7221
- MAX7221\_WRITE(unsigned char reg, unsigned char value)  
Used by other functions to send output and commands to the MAX7221
- MAX7221\_INTENSITY(unsigned char x)  
Send intensity value to the MAX7221 (used by MAX7221\_INIT)
- MAX7221\_SCAN\_LIMIT(unsigned char x)  
Send scan limit value to the MAX7221 (used by MAX7221\_INIT)
- MAX7221\_INIT()  
Initializes the MAX7221
- MAX7221\_TESTMODE(unsigned char enable)  
Enable/disable test mode
- MAX7221\_WRITEVAL(unsigned int value)  
Write a value to the display (value in decimal)
- MAX7221\_WRITENUM(unsigned int value)  
Write a value to the display (value in hexadecimal)

**serial.h** – Functions for using the RS-232 serial port

- SERIAL\_CONFIG(baudrategenval)  
Sets up baud rate
- SERIAL\_SEND(data)  
Send 8-bit data
- SERIAL\_RECEIVE()  
Receive 8-bit data
- SERIAL\_SEND16(data)  
Send 16-bit data
- SERIAL\_RECEIVE16()  
Receive 16-bit data

**delay.h** – Delay functions for the PIC

- DELAY\_US(x)  
Delay specified number of microseconds
- DELAY\_MS(x)  
Delay specified number of milliseconds

**spi.h** – Functions for using the SPI port on the PIC

- SPI\_CONFIG(stat, con)  
Sets SSPSTAT and SSPCON registers
- SPI\_SEND(data)  
Sends 8-bit data out over SPI
- SPI\_RECEIVE()  
Returns 8-bit data received over SPI
- SPI\_SEND16(data)  
Sends 16-bit (uint) data over SPI (MSB->LSB)
- SPI\_RECEIVE16()  
Returns 16-bit (uint) data received over SPI

# Microchip PIC16F876 Development Board User Manual

## mcp2510.h – Functions for using the MCP2510 (CAN controller)

```
/* mcp2510.h -- Functions for using the MCP2510 on PICs
**
** Functions available:
**   MCP2510_SETSPI()
**     - Sets up SPI port for use w/the MCP2510.
**   MCP2510_STATUS(port, mask)
**     - Returns the MCP2510 status byte.
**   MCP2510_WRITE(port, mask, addr, data)
**     - Writes data to register at addr.
**   MCP2510_WRITE_SEQUENCE(port, mask, startaddr, data, len)
**     - Writes data of length length starting at startaddr.
**   MCP2510_READ(port, mask, addr)
**     - Returns data at register addr.
**   MCP2510_RTS(port, mask, rtsnum)
**     - Request-to-send, transmits packet.  rtsnum = RTS0, RTS1 or RTS2.
**   MCP2510_BITMODIFY(port, mask, addr, bitmask, data)
**     - Sets or resets data's bits in addr, masked by bitmask.
**   MCP2510_RESET(port, mask)
**     - A software reset of the MCP2510.
**   MCP2510_TRANSMIT_STD(port, mask, id, dlc, data)
**     - Transmit a standard data frame.
**   MCP2510_TRANSMIT_EXT(port, mask, id, eid, dlc, data)
**     - Transmit an extended data frame.
**   MCP2510_TRANSMIT_REM(port, mask, id, eid)
**     - Transmit a remote data frame.
**
** Variables port and mask are the data port and bitmask to find
** the MCP2510 chip-select line.  i.e. if the MCP2510 is on PORTA's
** bit #4, port = &PORTA, mask = 0b00010000 = 0x10.  Recommend this
** be defined in the main program code rather than stated every use.
**
** Updates:
**   04/29/2002 - Nick Veys
**     - Inital Revision.
**   05/30/2002 - Nick Veys
**     - Added SPI_SETUP's to a few functions.
**     - Small documentation tweaks.
**   10/25/2002 - Nick Veys
**     - Added transmission routines.  NEED OPTIMIZATION!!
**   10/26/2002 - Nick Veys
**     - Optimized transmission routines. :)
**     - Added WRITE_SEQUENCE function.
*/

#ifndef __MCP2510_H
#define __MCP2510_H

#include "spi.h"
#include "delay.h"

#define RXF0SIDH    0x00    // Acceptance Filter 0 Std Identifier High
#define RXF0SIDL    0x01    // Acceptance Filter 0 Std Identifier Low
#define RXF0EID8    0x02    // Acceptance Filter 0 Ext Identifier High
#define RXF0EID0    0x03    // Acceptance Filter 0 Ext Identifier Low
#define RXF1SIDH    0x04    // Acceptance Filter 1 Std Identifier High
#define RXF1SIDL    0x05    // Acceptance Filter 1 Std Identifier Low
#define RXF1EID8    0x06    // Acceptance Filter 1 Ext Identifier High
#define RXF1EID0    0x07    // Acceptance Filter 1 Ext Identifier Low
#define RXF2SIDH    0x08    // Acceptance Filter 2 Std Identifier High
#define RXF2SIDL    0x09    // Acceptance Filter 2 Std Identifier Low
```

# Microchip PIC16F876 Development Board User Manual

```

#define RXF2EID8    0x0A           // Acceptance Filter 2 Ext Identifier High
#define RXF2EID0    0x0B           // Acceptance Filter 2 Ext Identifier Low
#define BFPCTRL     0x0C           // RXnBF Pin Control and Status Register
#define TXRTSCTRL   0x0D           // TXnRTS Pin Control and Status Register
#define CANSTAT     0x0E           // CAN Status Register
#define CANCTRL     0x0F           // CAN Control Register

#define RXF3SIDH    0x10           // Acceptance Filter 3 Std Identifier High
#define RXF3SIDL    0x11           // Acceptance Filter 3 Std Identifier Low
#define RXF3EID8    0x12           // Acceptance Filter 3 Ext Identifier High
#define RXF3EID0    0x13           // Acceptance Filter 3 Ext Identifier Low
#define RXF4SIDH    0x14           // Acceptance Filter 4 Std Identifier High
#define RXF4SIDL    0x15           // Acceptance Filter 4 Std Identifier Low
#define RXF4EID8    0x16           // Acceptance Filter 4 Ext Identifier High
#define RXF4EID0    0x17           // Acceptance Filter 4 Ext Identifier Low
#define RXF5SIDH    0x18           // Acceptance Filter 5 Std Identifier High
#define RXF5SIDL    0x19           // Acceptance Filter 5 Std Identifier Low
#define RXF5EID8    0x1A           // Acceptance Filter 5 Ext Identifier High
#define RXF5EID0    0x1B           // Acceptance Filter 5 Ext Identifier Low
#define TEC         0x1C           // Transmitter error counter
#define REC         0x1D           // Reciever error counter

#define RXM0SIDH    0x20           // Acceptance Mask 0 Std Identifier High
#define RXM0SIDL    0x21           // Acceptance Mask 0 Std Identifier Low
#define RXM0EID8    0x22           // Acceptance Mask 0 Ext Identifier High
#define RXM0EID0    0x23           // Acceptance Mask 0 Ext Identifier Low
#define RXM1SIDH    0x24           // Acceptance Mask 1 Std Identifier High
#define RXM1SIDL    0x25           // Acceptance Mask 1 Std Identifier Low
#define RXM1EID8    0x26           // Acceptance Mask 1 Ext Identifier High
#define RXM1EID0    0x27           // Acceptance Mask 1 Ext Identifier Low
#define CNF3        0x28           // Configuration Register 3
#define CNF2        0x29           // Configuration Register 2
#define CNF1        0x2A           // Configuration Register 1
#define CANINTE     0x2B           // Interrupt Enable Register
#define CANINTF     0x2C           // Interrupt Flag Register
#define EFLG        0x2D           // Error flag register

#define TXB0CTRL    0x30           // Transmit Buffer 0 Control Register
#define TXB0SIDH    0x31           // Transmit Buffer 0 Std Identifier High
#define TXB0SIDL    0x32           // Transmit Buffer 0 Std Identifier Low
#define TXB0EID8    0x33           // Transmit Buffer 0 Ext Identifier High
#define TXB0EID0    0x34           // Transmit Buffer 0 Ext Identifier Low
#define TXB0DLC     0x35           // Transmit Buffer 0 Data Length Code
#define TXB0D0     0x36           // Transmit Buffer 0 Data Byte 0
#define TXB0D1     0x37           // Transmit Buffer 0 Data Byte 1
#define TXB0D2     0x38           // Transmit Buffer 0 Data Byte 2
#define TXB0D3     0x39           // Transmit Buffer 0 Data Byte 3
#define TXB0D4     0x3A           // Transmit Buffer 0 Data Byte 4
#define TXB0D5     0x3B           // Transmit Buffer 0 Data Byte 5
#define TXB0D6     0x3C           // Transmit Buffer 0 Data Byte 6
#define TXB0D7     0x3D           // Transmit Buffer 0 Data Byte 7

#define TXB1CTRL    0x40           // Transmit Buffer 1 Control Register
#define TXB1SIDH    0x41           // Transmit Buffer 1 Std Identifier High
#define TXB1SIDL    0x42           // Transmit Buffer 1 Std Identifier Low
#define TXB1EID8    0x43           // Transmit Buffer 1 Ext Identifier High
#define TXB1EID0    0x44           // Transmit Buffer 1 Ext Identifier Low
#define TXB1DLC     0x45           // Transmit Buffer 1 Data Length Code
#define TXB1D0     0x46           // Transmit Buffer 1 Data Byte 0
#define TXB1D1     0x47           // Transmit Buffer 1 Data Byte 1
#define TXB1D2     0x48           // Transmit Buffer 1 Data Byte 2
#define TXB1D3     0x49           // Transmit Buffer 1 Data Byte 3
#define TXB1D4     0x4A           // Transmit Buffer 1 Data Byte 4

```

# Microchip PIC16F876 Development Board User Manual

```

#define TXB1D5      0x4B          // Transmit Buffer 1 Data Byte 5
#define TXB1D6      0x4C          // Transmit Buffer 1 Data Byte 6
#define TXB1D7      0x4D          // Transmit Buffer 1 Data Byte 7

#define TXB2CTRL    0x50          // Transmit Buffer 2 Control Register
#define TXB2SIDH    0x51          // Transmit Buffer 2 Std Identifier High
#define TXB2SIDL    0x52          // Transmit Buffer 2 Std Identifier Low
#define TXB2EID8    0x53          // Transmit Buffer 2 Ext Identifier High
#define TXB2EID0    0x54          // Transmit Buffer 2 Ext Identifier Low
#define TXB2DLC     0x55          // Transmit Buffer 2 Data Length Code
#define TXB2D0      0x56          // Transmit Buffer 2 Data Byte 0
#define TXB2D1      0x57          // Transmit Buffer 2 Data Byte 1
#define TXB2D2      0x58          // Transmit Buffer 2 Data Byte 2
#define TXB2D3      0x59          // Transmit Buffer 2 Data Byte 3
#define TXB2D4      0x5A          // Transmit Buffer 2 Data Byte 4
#define TXB2D5      0x5B          // Transmit Buffer 2 Data Byte 5
#define TXB2D6      0x5C          // Transmit Buffer 2 Data Byte 6
#define TXB2D7      0x5D          // Transmit Buffer 2 Data Byte 7

#define RXB0CTRL    0x60          // Recieve Buffer 0 Control Register
#define RXB0SIDH    0x61          // Recieve Buffer 0 Std Identifier High
#define RXB0SIDL    0x62          // Recieve Buffer 0 Std Identifier Low
#define RXB0EID8    0x63          // Recieve Buffer 0 Ext Identifier High
#define RXB0EID0    0x64          // Recieve Buffer 0 Ext Identifier Low
#define RXB0DLC     0x65          // Recieve Buffer 0 Data Length Code
#define RXB0D0      0x66          // Recieve Buffer 0 Data Byte 0
#define RXB0D1      0x67          // Recieve Buffer 0 Data Byte 1
#define RXB0D2      0x68          // Recieve Buffer 0 Data Byte 2
#define RXB0D3      0x69          // Recieve Buffer 0 Data Byte 3
#define RXB0D4      0x6A          // Recieve Buffer 0 Data Byte 4
#define RXB0D5      0x6B          // Recieve Buffer 0 Data Byte 5
#define RXB0D6      0x6C          // Recieve Buffer 0 Data Byte 6
#define RXB0D7      0x6D          // Recieve Buffer 0 Data Byte 7

#define RXB1CTRL    0x70          // Recieve Buffer 1 Control Register
#define RXB1SIDH    0x71          // Recieve Buffer 1 Std Identifier High
#define RXB1SIDL    0x72          // Recieve Buffer 1 Std Identifier Low
#define RXB1EID8    0x73          // Recieve Buffer 1 Ext Identifier High
#define RXB1EID0    0x74          // Recieve Buffer 1 Ext Identifier Low
#define RXB1DLC     0x75          // Recieve Buffer 1 Data Length Code
#define RXB1D0      0x76          // Recieve Buffer 1 Data Byte 0
#define RXB1D1      0x77          // Recieve Buffer 1 Data Byte 1
#define RXB1D2      0x78          // Recieve Buffer 1 Data Byte 2
#define RXB1D3      0x79          // Recieve Buffer 1 Data Byte 3
#define RXB1D4      0x7A          // Recieve Buffer 1 Data Byte 4
#define RXB1D5      0x7B          // Recieve Buffer 1 Data Byte 5
#define RXB1D6      0x7C          // Recieve Buffer 1 Data Byte 6
#define RXB1D7      0x7D          // Recieve Buffer 1 Data Byte 7

#define CMD_READ    0x03          // Read Command
#define CMD_WRITE   0x02          // Write Command
#define CMD_BITMODIFY 0x05        // Bit-modify Command
#define CMD_READSTATUS 0xA0       // Read Status Command (poll)
#define CMD_RESET   0xC0          // Reset Command

#define RTS0 0x81          // Request-to-send Commands
#define RTS1 0x82
#define RTS2 0x84

#define MCP2510_SETSPI() SPI_CONFIG(0x00, 0x10)

unsigned char MCP2510_STATUS(unsigned char* port, unsigned char mask)
{

```

# Microchip PIC16F876 Development Board User Manual

```
    unsigned char temp;
    MCP2510_SETSPI();                // setup SPI port for MCP2510
    *port &= ~mask;                  // select MCP2510
    SPI_SEND(CMD_READSTATUS);        // read status command
    temp = SPI_RECIEVE();             // get it!
    temp = SPI_RECIEVE();            // it's sent twice
    *port |= mask;                   // de-select MCP2510
    return temp;                     // return it!
}

void MCP2510_WRITE(unsigned char* port, unsigned char mask,
                  unsigned char addr, unsigned char data)
{
    MCP2510_SETSPI();                // setup SPI port for MCP2510
    *port &= ~mask;                  // select MCP2510
    SPI_SEND(CMD_WRITE);             // send write command
    SPI_SEND(addr);                  // send register address
    SPI_SEND(data);                  // send the data
    *port |= mask;                   // de-select MCP2510
}

void MCP2510_WRITE_SEQUENCE(unsigned char* port, unsigned char mask,
                           unsigned char startaddr, unsigned char* data, unsigned
char len)
{
    unsigned char i;
    MCP2510_SETSPI();                // setup SPI port for MCP2510
    *port &= ~mask;                  // select MCP2510
    SPI_SEND(CMD_WRITE);             // send write command
    SPI_SEND(startaddr);             // send register starting address
    for(i = 0; i < len; i++) SPI_SEND(data[i]); // send the data
    *port |= mask;                   // de-select MCP2510
}

unsigned char MCP2510_READ(unsigned char* port, unsigned char mask, unsigned char
addr)
{
    unsigned char temp;
    MCP2510_SETSPI();                // setup SPI port for MCP2510
    *port &= ~mask;                  // select MCP2510
    SPI_SEND(CMD_READ);              // send read command
    SPI_SEND(addr);                  // send register address
    temp = SPI_RECIEVE();            // get the data
    *port |= mask;                   // de-select MCP2510
    return temp;                     // return the data
}

void MCP2510_READ_SEQUENCE(unsigned char* port, unsigned char mask,
                          unsigned char startaddr, unsigned char* dest, unsigned char
len)
{
    unsigned char i;
    MCP2510_SETSPI();                // setup SPI port for MCP2510
    *port &= ~mask;                  // select MCP2510
    SPI_SEND(CMD_READ);              // send read command
    SPI_SEND(startaddr);             // send register starting address
    for(i = 0; i < len; i++) dest[i] = SPI_RECIEVE(); // get the data
    *port |= mask;                   // de-select MCP2510
}

void MCP2510_RTS(unsigned char* port, unsigned char mask, unsigned char rtsnum)
{
    MCP2510_SETSPI();                // setup SPI port for MCP2510
```

# Microchip PIC16F876 Development Board User Manual

```
*port &= ~mask;                // select MCP2510
SPI_SEND(rtsnum);              // must be RTS0, RTS1 or RTS2
*port |= mask;                 // de-select MCP2510
}

void MCP2510_BITMODIFY(unsigned char* port, unsigned char mask,
                      unsigned char addr, unsigned char bitmask, unsigned char data)
{
    MCP2510_SETSPI();          // setup SPI port for MCP2510
    *port &= ~mask;            // select MCP2510
    SPI_SEND(CMD_BITMODIFY);   // bit modify command
    SPI_SEND(addr);           // send register address
    SPI_SEND(bitmask);        // send bitmask
    SPI_SEND(data);           // send data
    *port |= mask;            // de-select MCP2510
}

void MCP2510_RESET(unsigned char* port, unsigned char mask)
{
    MCP2510_SETSPI();          // setup SPI port for MCP2510
    *port &= ~mask;            // select MCP2510
    SPI_SEND(CMD_RESET);      // reset MCP2510 command
    *port |= mask;            // de-select MCP2510
    DELAY_US(25);             // let it reset
}

void MCP2510_TRANSMIT_STD(unsigned char* port, unsigned char mask,
                          unsigned char* id, unsigned char dlc, unsigned char* data)
{
    unsigned char status;
    unsigned char packet[13]; // storage for complete packet

    packet[0] = ((id[0] & 0x07) << 5) + ((id[1] & 0xF8) >> 3);
    packet[1] = (id[1] & 0x07) << 5;
    packet[2] = 0;
    packet[3] = 0;
    packet[4] = dlc & 0x0F;
    packet[5] = data[0];
    packet[6] = data[1];
    packet[7] = data[2];
    packet[8] = data[3];
    packet[9] = data[4];
    packet[10] = data[5];
    packet[11] = data[6];
    packet[12] = data[7];

    status = MCP2510_STATUS(port, mask); // get status register

    MCP2510_SETSPI(); // setup SPI port for MCP2510
    *port &= ~mask; // select MCP2510

STDLOOP:
    if(!(status & 0x02)) // TX0 is free
    {
        MCP2510_WRITE_SEQUENCE(port, mask, TXB0SIDH, packet, 13);
        MCP2510_RTS(port, mask, RTS0);
    }
    else if(!(status & 0x10)) // TX1 is free
    {
        MCP2510_WRITE_SEQUENCE(port, mask, TXB1SIDH, packet, 13);
        MCP2510_RTS(port, mask, RTS1);
    }
    else if(!(status & 0x40)) // TX2 is free
```

## Microchip PIC16F876 Development Board User Manual

```

{
    MCP2510_WRITE_SEQUENCE(port, mask, TXB2SIDH, packet, 13);
    MCP2510_RTS(port, mask, RTS2);
}
else goto STDLOOP;                                // nothing free, try again

*port |= mask;                                    // de-select MCP2510
}

void MCP2510_TRANSMIT_EXT(unsigned char* port, unsigned char mask,
                          unsigned char* id, unsigned char* eid,
                          unsigned char dlc, unsigned char* data)
{
    unsigned char status;                          // storage for MCP2510's status byte
    unsigned char packet[13];                      // storage for complete packet

    packet[0] = ((id[0] & 0x07) << 5) + ((id[1] & 0xF8) >> 3);
    packet[1] = (((id[1] & 0x07) << 5) + (eid[0] & 0x03)) | 0x08;
    packet[2] = eid[1];
    packet[3] = eid[2];
    packet[4] = dlc & 0x0F;
    packet[5] = data[0];
    packet[6] = data[1];
    packet[7] = data[2];
    packet[8] = data[3];
    packet[9] = data[4];
    packet[10] = data[5];
    packet[11] = data[6];
    packet[12] = data[7];

    status = MCP2510_STATUS(port, mask);           // get status register

    MCP2510_SETSPI();                              // setup SPI port for MCP2510
    *port &= ~mask;                                // select MCP2510

EXTLOOP:
    if(!(status & 0x02))                           // TX0 is free
    {
        MCP2510_WRITE_SEQUENCE(port, mask, TXB0SIDH, packet, 13);
        MCP2510_RTS(port, mask, RTS0);
    }
    else if(!(status & 0x10))                       // TX1 is free
    {
        MCP2510_WRITE_SEQUENCE(port, mask, TXB1SIDH, packet, 13);
        MCP2510_RTS(port, mask, RTS1);
    }
    else if(!(status & 0x40))                       // TX2 is free
    {
        MCP2510_WRITE_SEQUENCE(port, mask, TXB2SIDH, packet, 13);
        MCP2510_RTS(port, mask, RTS2);
    }
    else goto EXTLOOP;                              // nothing free, try again

    *port |= mask;                                  // de-select MCP2510
}

void MCP2510_TRANSMIT_REM(unsigned char* port, unsigned char mask,
                          unsigned char* id, unsigned char* eid)
{
    unsigned char status;                          // storage for complete packet
    unsigned char packet[5];

    packet[0] = ((id[0] & 0x07) << 5) + ((id[1] & 0xF8) >> 3);

```

## Microchip PIC16F876 Development Board User Manual

```
packet[1] = (((id[1] & 0x07) << 5) + (eid[0] & 0x03)) | 0x08;
packet[2] = eid[1];
packet[3] = eid[2];
packet[4] = 0x40;

status = MCP2510_STATUS(port, mask);           // get status register

MCP2510_SETSPI();                             // setup SPI port for MCP2510
*port &= ~mask;                               // select MCP2510

REMLOOP:
if(!(status & 0x02))                          // TX0 is free
{
    MCP2510_WRITE_SEQUENCE(port, mask, TXB0SIDH, packet, 5);
    MCP2510_RTS(port, mask, RTS0);
}
else if(!(status & 0x10))                    // TX1 is free
{
    MCP2510_WRITE_SEQUENCE(port, mask, TXB1SIDH, packet, 5);
    MCP2510_RTS(port, mask, RTS1);
}
else if(!(status & 0x40))                    // TX2 is free
{
    MCP2510_WRITE_SEQUENCE(port, mask, TXB2SIDH, packet, 5);
    MCP2510_RTS(port, mask, RTS2);
}
else goto REMLOOP;                          // nothing free, try again

*port |= mask;                               // de-select MCP2510
}

#endif
```

# Microchip PIC16F876 Development Board User Manual

## lcd.h – Functions for controlling and sending output to the LCD

```
#ifndef __LCD_H
#define __LCD_H

#include <pic.h>
#include "delay.h"

#define LCD_E    RB1    // LCD Enable control line.
#define LCD_RW   RB2    // LCD Read/Write control line.
#define LCD_RS   RB3    // LCD Register-Select control line.
#define LCD_DB4  RB4
#define LCD_DB5  RB5
#define LCD_DB6  RB6
#define LCD_DB7  RB7

void LCD_OUT_COMMAND(unsigned char cmd)           // Output a command byte
{
    LCD_RS = 0;                                   // Command mode
    LCD_RW = 0;                                   // Write mode

    LCD_DB7 = (cmd & 0x80) ? 1:0;                 // Bit 7 -> DB7
    LCD_DB6 = (cmd & 0x40) ? 1:0;                 // Bit 6 -> DB6
    LCD_DB5 = (cmd & 0x20) ? 1:0;                 // Bit 5 -> DB5
    LCD_DB4 = (cmd & 0x10) ? 1:0;                 // Bit 4 -> DB4

    LCD_E = 1;                                    // Strobe E high
    DELAY_US(1);                                  // Wait a wee bit
    LCD_E = 0;                                    // Strobe E low

    LCD_DB7 = (cmd & 0x08) ? 1:0;                 // Bit 3 -> DB7
    LCD_DB6 = (cmd & 0x04) ? 1:0;                 // Bit 2 -> DB6
    LCD_DB5 = (cmd & 0x02) ? 1:0;                 // Bit 1 -> DB5
    LCD_DB4 = (cmd & 0x01) ? 1:0;                 // Bit 0 -> DB4

    LCD_E = 1;                                    // Strobe E high
    DELAY_US(1);                                  // Wait a wee bit
    LCD_E = 0;                                    // Strobe E low
    DELAY_MS(15);                                 // Wait for instruction to complete
}

void LCD_OUT_UPPER_NYBBLE(unsigned char nybble)   // Output upper nybble (command)
{
    LCD_RS = 0;
    LCD_RW = 0;

    LCD_DB7 = (nybble & 0x80) ? 1:0;              // Bit 7 -> DB7
    LCD_DB6 = (nybble & 0x40) ? 1:0;              // Bit 6 -> DB6
    LCD_DB5 = (nybble & 0x20) ? 1:0;              // Bit 5 -> DB5
    LCD_DB4 = (nybble & 0x10) ? 1:0;              // Bit 4 -> DB4

    LCD_E = 1;                                    // Strobe E high
    DELAY_US(1);                                  // Wait a wee bit
    LCD_E = 0;                                    // Strobe E low
    DELAY_MS(15);                                 // Wait for instruction to complete
}

void LCD_OUT_DATA(unsigned char data)             // Output a data byte
{
    LCD_RS = 1;                                    // Data mode
    LCD_RW = 0;                                    // Write mode

    LCD_DB7 = (data & 0x80) ? 1:0;                // Bit 7 -> DB7
```

## Microchip PIC16F876 Development Board User Manual

```

LCD_DB6 = (data & 0x40) ? 1:0;           // Bit 6 -> DB6
LCD_DB5 = (data & 0x20) ? 1:0;           // Bit 5 -> DB5
LCD_DB4 = (data & 0x10) ? 1:0;           // Bit 4 -> DB4

LCD_E = 1;                               // Strobe E high
DELAY_US(1);                             // Wait a wee bit
LCD_E = 0;                               // Strobe E low

LCD_DB7 = (data & 0x08) ? 1:0;           // Bit 3 -> DB7
LCD_DB6 = (data & 0x04) ? 1:0;           // Bit 2 -> DB6
LCD_DB5 = (data & 0x02) ? 1:0;           // Bit 1 -> DB5
LCD_DB4 = (data & 0x01) ? 1:0;           // Bit 0 -> DB4

LCD_E = 1;                               // Strobe E high
DELAY_US(1);                             // Wait a wee bit
LCD_E = 0;                               // Strobe E low
DELAY_MS(15);                            // Wait for instruction to complete
}

#define LCD_CLEAR() LCD_OUT_COMMAND(0x01) // Clear command
#define LCD_HOME() LCD_OUT_COMMAND(0x02)
#define LCD_SET_CGRAM(x) LCD_OUT_COMMAND(0x40 | (x & 0x3F))
#define LCD_SET_DDRAM(x) LCD_OUT_COMMAND(0x80 | (x & 0x7F))

void LCD_INITIALIZE(unsigned char lines) // Initialize the display
{
    LCD_E = 0;                            // Initialize all control signals
    LCD_RW = 0;
    LCD_RS = 0;

    DELAY_MS(20);                         // Wait >15ms for Vcc to rise.

    LCD_OUT_UPPER_NYBBLE(0x30);           // Function Set (8-bit)
    LCD_OUT_UPPER_NYBBLE(0x30);           // Function Set (8-bit)
    LCD_OUT_UPPER_NYBBLE(0x30);           // Function Set (8-bit)
    LCD_OUT_UPPER_NYBBLE(0x20);           // Function Set (4-bit)

    if(lines == 1)
        LCD_OUT_COMMAND(0x20);           // 4-bit interface, 1 line, 5x7 font
    else if(lines == 2)
        LCD_OUT_COMMAND(0x28);           // 4-bit interface, 2 lines, 5x7 font

    LCD_OUT_COMMAND(0x0C);                // Display on, no cursor, no blink
    LCD_CLEAR();                          // Display clear
    LCD_OUT_COMMAND(0x06);                // Auto-increment cursor
}

void LCD_PUTS(const char* s, unsigned char len)
{
    unsigned char i;

    for(i = 0; i < len; i++)
    {
        LCD_OUT_DATA(s[i]);               // Put out each character
    }
}

#endif

```

# Microchip PIC16F876 Development Board User Manual

## **max7221.h** – Functions for controlling the 7-segment display driver

```
// Seven segment display

#ifndef __MAX7221_H
#define __MAX7221_H

#include <pic.h>
#include "spi.h"

#define MAX7221_CS          RC0 // Chip Select.

#define DIG0_ADDR          0x01
#define DIG1_ADDR          0x02
#define DIG2_ADDR          0x03
#define DIG3_ADDR          0x04
#define DIG4_ADDR          0x05
#define DIG5_ADDR          0x06
#define DIG6_ADDR          0x07
#define DIG7_ADDR          0x08
#define DECODE_MODE        0x09
#define INTENSITY          0x0A
#define SCAN_LIMIT         0x0B
#define SHUTDOWN           0x0C
#define DISPLAY_TEST       0x0F

#define MAX7221_SETSPI() SPI_CONFIG(0x40, 0x01)

void MAX7221_WRITE(unsigned char reg, unsigned char value)
{
    MAX7221_SETSPI();                // set up SPI port for MAX7221
    MAX7221_CS = 0;                  // select MAX7221
    SPI_SEND16((unsigned int)((reg<<8)+value)); // send value
    MAX7221_CS = 1;                  // de-select MAX7221
}

void MAX7221_INTENSITY(unsigned char x)
{
    MAX7221_SETSPI();                // set up SPI port for MAX7221
    MAX7221_CS = 0;                  // select MAX7221
    SPI_SEND16((unsigned int)((INTENSITY<<8)+x)); // send value
    MAX7221_CS = 1;                  // de-select MAX7221
}

void MAX7221_SCAN_LIMIT(unsigned char x)
{
    MAX7221_SETSPI();                // set up SPI port for MAX7221
    MAX7221_CS = 0;                  // select MAX7221
    SPI_SEND16((unsigned int)((SCAN_LIMIT<<8)+x)); // send value
    MAX7221_CS = 1;                  // de-select MAX7221
}

void MAX7221_INIT()
{
    MAX7221_INTENSITY(0x8);          // set intensity
    MAX7221_SCAN_LIMIT(0x3);         // set scan limit (4 digits)
    MAX7221_WRITE(DECODE_MODE, 0xF); //
    MAX7221_WRITE(SHUTDOWN, 0x1);   // enable display
}

void MAX7221_TESTMODE(unsigned char enable)
{

```

# Microchip PIC16F876 Development Board User Manual

```
    if(enable) MAX7221_WRITE(DISPLAY_TEST, 0x01);        // enable test mode
    else MAX7221_WRITE(DISPLAY_TEST, 0x00);            // disable test mode
}

void MAX7221_WRITENUM(unsigned int value)
{
    unsigned int dig0, dig1, dig2, dig3;
    dig0 = (value%10000)/1000;
    dig1 = (value%1000)/100;
    dig2 = (value%100)/10;
    dig3 = (value%10);

    MAX7221_WRITE(DIG0_ADDR, (unsigned int)dig0);
    MAX7221_WRITE(DIG1_ADDR, (unsigned int)dig1);
    MAX7221_WRITE(DIG2_ADDR, (unsigned int)dig2);
    MAX7221_WRITE(DIG3_ADDR, (unsigned int)dig3);
}

void MAX7221_WRITEVAL(unsigned int value)
{
    MAX7221_WRITE(DIG0_ADDR, ((value&0xF000)>>12));
    MAX7221_WRITE(DIG1_ADDR, ((value&0x0F00)>>8));
    MAX7221_WRITE(DIG2_ADDR, ((value&0x00F0)>>4));
    MAX7221_WRITE(DIG3_ADDR, value&0x000F);
}

#endif
```

## **serial.h - Functions for using the RS-232 serial port**

```
/* serial.h -- Functions for using the UART on PICs that have them.
**          Defaults to ASYNC, 8N1, low speed
**
** Functions available:
**     SERIAL_CONFIG(baudrategenval)
**         - Sets up baud rate.
**     SERIAL_SEND(data)
**         - Send 8bit data.
**     SERIAL_RECIEVE()
**         -
**     SERIAL_SEND16(data)
**         - Send 16bit data.
**     SERIAL_RECIEVE16()
**         -
**
** This library requires PIC_XTAL to be #define'd before inclusion.
** the MHz and KHz suffixes may be used for a x1000 or x1.
**
** Updates:
**     04/29/2002 - Nick Veys
**         - Inital Revision.
**     05/06/2002 - Nick Veys
**         - Added most stuff, need broader MCU support, better TX delay.
```

# Microchip PIC16F876 Development Board User Manual

```
**      - Reception not implemented, need good methodology first.
*/

#ifndef __SERIAL_H
#define __SERIAL_H

#ifndef MHz
#define      MHz *1000L
#endif
#ifndef KHz
#define      KHz *1
#endif

#include <pic.h>
#include "delay.h"

void SERIAL_CONFIG(unsigned char baudrategenval)
{
    TXEN = 0;
    SPEN = 0;
    #if defined(_16F873) || defined(_16F874) || defined(_16F875) || \
        defined(_16F876) || defined(_16F877)
        TRISC |= 0b10000000;           // RC7 = in
        TRISC &= 0b10111111;         // RC6 = out
    #else
    #error Sorry!  serial.h does not support this PIC
    #endif

    SPBRG = baudrategenval;           // set baud rate generator
    SPEN = 1;                          // serial port enabled
    TXEN = 1;                          // enable serial transmission
}

void SERIAL_SEND(unsigned char data)
{
    TXREG = data;                       // load TXREG
    while(!TRMT);                       // wait for flag
}

void SERIAL_SEND16(unsigned int data)    // call SEND twice
{
    SERIAL_SEND((unsigned char)((data & 0xFF00) >> 8));
    SERIAL_SEND((unsigned char)(data & 0x00FF));
}

#endif
```

## **delay.h** – Delay functions for the PIC

```
/* delay.h  --  Delay functions for the PIC
**
** Functions available:
**     DELAY_US(x)
```

# Microchip PIC16F876 Development Board User Manual

```
**      - Delay specified number of microseconds.
**      DELAY_MS(x)
**      - Delay specified number of milliseconds.
**
** Note that there are range limits: x must not exceed 255 - for xtal
** frequencies > 12MHz the range for DELAY_US is even smaller.
**
** DELAY_US is implemented as an 'inline' style macro function.
**
** This library requires PIC_XTAL to be #define'd before inclusion.
** the MHz and KHz suffixes may be used for a x1000 or x1.
**
** Updates:
** 04/29/2002 - Nick Veys
** - Inital Revision.
** - Set up rough timing.
**/
```

```
#ifndef __DELAY_H
#define __DELAY_H
```

```
#ifndef MHz
#define      MHz *1000L
#endif
#ifndef KHz
#define      KHz *1
#endif
```

```
#ifndef PIC_XTAL
#error PIC_XTAL not defined.
#endif
```

```
#if PIC_XTAL >= 12MHz
#define DELAY_US(x)\
    {\
        unsigned char _dcnt;\
        _dcnt = (x)*((PIC_XTAL)/(12MHz));\
        while(--_dcnt != 0) continue;\
    }
#else
#define DELAY_US(x)\
    {\
        unsigned char _dcnt;\
        _dcnt = (x)/((12MHz)/(PIC_XTAL))|1;\
        while(--_dcnt != 0) continue;\
    }
#endif
```

```
void DELAY_MS(unsigned char cnt)
{
    #if PIC_XTAL <= 2MHz
    do
    {
        DELAY_US(996);
    } while(--cnt);
    #endif

    #if PIC_XTAL > 2MHz
    unsigned char i;
    do
    {
        i = 4;
    do
```

# Microchip PIC16F876 Development Board User Manual

```
    {
        DELAY_US(250);
    } while(--i);
} while(--cnt);
#endif
}
```

```
#endif
```

## **spi.h** – Functions for using the SPI port on the PIC

```
/* spi.h -- Functions for using the SPI port on PICs that have them.
**
** Functions available:
**   SPI_CONFIG(stat, con)
**     - Sets SSPSTAT and SSPCON registers.
**   SPI_SEND(data)
**     - Sends 8-bit data out over SPI.
**   SPI_RECIEVE()
**     - Returns 8-bit data recieved over SPI.
**   SPI_SEND16(data)
**     - Sends 16-bit (uint) data over SPI.  MSB->LSB.
**   SPI_RECIEVE16()
**     - Returns 16-bit (uint) data recieved over SPI.
**
** Updates:
**   04/29/2002 - Nick Veys
**     - Inital Revision.
**
*/

#ifndef __SPI_H
#define __SPI_H

#include <pic.h>

void SPI_CONFIG(unsigned char stat, unsigned char con)
{
    SSPEN = 0;                // disable SPI port for reconfiguration
    SSPSTAT = stat;          // set SSPSTAT
    SSPCON = con;            // set SSPCON
    SSPEN = 1;                // re-enable SPI port
}

void SPI_SEND(unsigned char data)
{
    SSPBUF = data;           // write out to buffer
    while(!SSPIF);          // wait for flag
    SSPIF = 0;               // clear flag
}

unsigned char SPI_RECEIVE()
```

## Microchip PIC16F876 Development Board User Manual

```
{
    SSPBUF = 0;                // load 0
    while(!SSPIF);            // wait for data received
    SSPIF = 0;                 // handle flag
    return SSPBUF;            // return the value
}

void SPI_SEND16(unsigned int data)        // call SEND twice
{
    SPI_SEND((unsigned char)((data & 0xFF00) >> 8));
    SPI_SEND((unsigned char)(data & 0x00FF));
}

unsigned int SPI_RECEIVE16()              // call RECEIVE twice
{
    return((unsigned int)(SPI_RECEIVE() << 8) + SPI_RECEIVE());
}

#endif
```

## C. Initialization & Diagnostics Routine

An initialization routine has been written to perform the necessary setup procedures and execute simple tests to verify that the board is working properly. This routine (or at least the vital parts of it) should be programmed into the PIC and performed every time the board is started to ensure proper operation. It performs the following operations:

- selects the HS (high-speed crystal) oscillator type;
- enables the power-up timer;
- disables the watchdog timer;
- disables low-voltage programming;
- sets up the I/O pins;
- configures the ADC (analog-to-digital converter);
- turns the speaker on and emits a start-up beep;
- initializes and writes a test string to the LCD;
- initializes and tests the 7-segment LEDs with a simple count-down;
- reads the temperature and displays it on the LCD.

This code provides good examples of how the PIC accepts input from various devices on the board, manipulates the data, and sends output. In particular, it illustrates the usage of the ADC (included in the PIC), the speaker, the LCD, the temperature sensor, and the 7-segment LEDs.

```
/* init.c - Initialization routine for PIC Board
*/

#define PIC_XTAL 20MHz

#include <pic.h>
#include "delay.h"
//#include "mcp2510.h"
#include "lcd.h"
#include "max7221.h"

/* HS oscillator
** Power-up timer enabled
** Watchdog disabled
** Low-voltage programming disabled */
__CONFIG(HS & PWRTEN & BOREN & WDTDIS & LVPDIS);

#define SPEAKER RC1

unsigned int READ_TEMPERATURE()
{
    unsigned int x;

    ADCON0 = 0x80;
    ADON = 1; // Turn ADC on
    DELAY_US(10); // Wait for it to start up
    ADGO = 1; // Start the conversion
    while(ADGO); // Wait for it to complete
    ADON = 0; // The the ADC off
    ADIF = 0; // Clear interrupt flag
}
```

# Microchip PIC16F876 Development Board User Manual

```
x = (ADRESH << 8) + ADRESL;
x = ((long) x * 500) / 1024;
return x;
}

void separateDigits(unsigned int num, unsigned char* retval)
{
    retval[2] = num % 10;                // Get ones digit
    num -= retval[2];

    retval[1] = num % 100;               // Get tens digit
    num -= retval[1];
    retval[1] /= 10;

    retval[0] = num % 1000;              // Get hundreds digit
    num -= retval[0];
    retval[0] /= 100;

    retval[0] += '0';                    // Make them ASCII
    retval[1] += '0';
    retval[2] += '0';

    if(retval[0] == '0')                  // check for leading 0's
    {
        if(retval[1] == '0')
        {
            retval[0] = retval[1] = ' ';
        }
        else
            retval[0] = ' ';
    }
}

void main()
{
    unsigned char temp[3];
    unsigned int timer;
    int count;

    TRISB = 0b00000001;                  // set up i/o pins
    TRISC = 0b10010000;

    ADCON1 = 0x84;                       // configure ADC

    SPEAKER = 0;                          // Speaker on
    DELAY_MS(25);                          // Beep for 25ms
    SPEAKER = 1;                          // Speaker off

    LCD_INITIALIZE(2);                     // Initialize the 2-line LCD
    MAX7221_INIT();                        // Initialize the seven segment display

    MAX7221_WRITEVAL(0x0050);

    LCD_PUTS("Hello, World!", 13);         // Write test string

    timer = 50;
    count = 5;

    while(1)                               // Temperature update loop
    {
        LCD_SET_DDRAM(40);                 // Line 2
        separateDigits(READ_TEMPERATURE(), temp); // Break up temperature digits
    }
}
```

## Microchip PIC16F876 Development Board User Manual

```
LCD_PUTS("Temperature: ", 13);           // Temperature header string
LCD_PUTS(temp, 3);                       // put actual temp value out
LCD_OUT_DATA(0xDF);                      // degree symbol
LCD_OUT_DATA('F');                      // Fahrenheit

DELAY_MS(100);                           // wait 100ms til next update
    if(timer!=-1)
    {
        if(count==0)
        {
            count=5;
            MAX7221_WRITENUM(timer);
            timer--;
        }
        else
            count--;
    }
}
```

## 5. Additional Resources

### **Dec02-12 Project Web Site**

<http://www.kilohm.net/seniord/>  
<http://seniord.ee.iastate.edu/dec0212/>

CIA – CAN in Automation (CAN protocol overview)  
<http://www.can-cia.de/can/protocol/>

Fairchild Semiconductor  
<http://www.fairchildsemi.com>

HI-TECH Software  
<http://www.htsoft.com>

Maxim Integrated Products  
<http://www.maxim-ic.com>

National Semiconductor  
<http://www.national.com>

ON Semiconductor  
<http://www.onsemi.com>

P16PRO & PICALL PIC Programmer  
<http://www.picallw.com>

Philips Electronics  
<http://www.philips.com>

Planet Microchip  
<http://www.microchip.com>

Veys.com (PIC tutorials)  
<http://www.veys.com>

WinPicProg Development Site  
<http://www.winpicprog.co.uk>