MICROCHIP

44-PIN DEMO BOARD
USER'S GUIDE

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

- Microchip products meet the specification contained in their particular Microchip Data Sheet.

- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.

- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.

- Microchip is willing to work with the customer who is concerned about the integrity of their code.

- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

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

Trademarks

QUALITY MANAGEMENT SYSTEM

CERTIFIED BY DNV

ISO/TS 16949:2002

44-PIN DEMO BOARD USER'S GUIDE

Table of Contents

44-Pin Demo Board User's Guide

Preface

NOTICE TO CUSTOMERS

All documentation becomes dated, and this manual is no exception. Microchip tools and documentation are constantly evolving to meet customer needs, so some actual dialogs and/or tool descriptions may differ from those in this document. Please refer to our web site (www.microchip.com) to obtain the latest documentation available.

Documents are identified with a "DS" number. This number is located on the bottom of each page, in front of the page number. The numbering convention for the DS number is "DSXXXXXA", where "XXXXX" is the document number and "A" is the revision level of the document.

For the most up-to-date information on development tools, see the MPLAB® IDE on-line help. Select the Help menu, and then Topics to open a list of available on-line help files.

INTRODUCTION

This chapter contains general information that will be useful to know before using the 44-Pin Demo Board. Items discussed in this chapter include:

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

DOCUMENT LAYOUT

This document describes how to use the 44-Pin Demo Board as a development tool to emulate and debug firmware on a target board. The manual layout is as follows:

- **Chapter 1. "44-Pin Demo Board Overview"** – This chapter provides an overview of the 44-Pin Demo Board for Microchip's 44-pin Thin Quad Flatpack (TQFP) PIC® Microcontroller Units (MCU).
- **Chapter 2. "Mid-Range PIC® Microcontroller Architectural Overview"** – This chapter provides an overview of the mid-range PIC® microcontroller architecture.
- **Chapter 3. "44-Pin Demo Board Lessons"** – This chapter provides lessons that introduce mid-range PIC® MCU assembly instructions and cover basic 44-Pin Demo board features.
- **Appendix A. "Hardware Schematics"** – Illustrates the 44-Pin Demo Board hardware schematic diagram, PCB layout and Bill of Materials.

44-Pin Demo Board User's Guide

CONVENTIONS USED IN THIS GUIDE

This manual uses the following documentation conventions:

DOCUMENTATION CONVENTIONS

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

	Represents code supplied by user	`void main (void)`
		`{ ...`
		`}`

WARRANTY REGISTRATION

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

RECOMMENDED READING

This user's guide describes how to use the 44-Pin Demo Board. Other useful documents are listed below. The following Microchip documents are available and recommended as supplemental reference resources.

PIC16F88X Data Sheet (DS41291)

Consult this document for information regarding the PIC16F88X 28/40/44-Pin Flash-Based, 8-Bit CMOS Microcontrollers with nanoWatt Technology device specification.

PICkit™ 2 Microcontroller Programmer User's Guide (DS51553)

Consult this document for instructions on how to use the PICkit 2 Microcontroller Programmer software and hardware.

MPLAB® ICD User's Guide (DS51184)

Consult this document for more information pertaining to the features and functions of the MPLAB In-Circuit Debugger (ICD) software.

MPLAB® IDE User's Guide (DS51519)

Consult this document for more information pertaining to the installation and features of the MPLAB Integrated Development Environment (IDE) Software.

Readme Files

For the latest information on using other tools, read the tool-specific Readme files in the Readmes subdirectory of the MPLAB IDE installation directory. The Readme files contain update information and known issues that may not be included in this user's guide.

44-Pin Demo Board User's Guide

THE MICROCHIP WEB SITE

Microchip provides online support via our web site at www.microchip.com. This web site is used as a means to make files and information easily available to customers. Accessible by using your favorite Internet browser, the web site contains the following information:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip consultant program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

DEVELOPMENT SYSTEMS CUSTOMER CHANGE NOTIFICATION SERVICE

Microchip's customer notification service helps keep customers current on Microchip products. Subscribers will receive e-mail notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, access the Microchip web site at www.microchip.com, click on Customer Change Notification and follow the registration instructions.

The Development Systems product group categories are:

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

CUSTOMER SUPPORT

Users of Microchip products can receive assistance through several channels:

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

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

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

DOCUMENT REVISION HISTORY

Revision A (August 2006)

- Initial release of this document.

Revision B (December 2006)

- Updated **Chapter 1. "PICkit™ 2 Overview"**.
- Added **Chapter 2. "Mid-Range PIC® Microcontroller Architectural Overview"**.
- Added **Chapter 3. "44-Pin Demo Board Lessons"**.
- Changed PICmicro® to PIC®.
- Changed PICkit® to PICkit™.
- Removed Development Systems Information Line from Customer Support bulleted list.
- Updated schematic in Appendix.

NOTES:

Chapter 1. 44-Pin Demo Board Overview

1.1   INTRODUCTION

The 44-Pin Demo Board is a small and simple demonstration PCB for Microchip's 44-pin Thin Quad Flatpack (TQFP) PIC® Microcontroller Units (MCU). It is populated with a PIC16F887 MCU, eight LEDs, push button and potentiometer. The demo board has several test points to access the I/O pins of the MCU and a surface mount prototyping area. The MCU can be programmed with the PICkit™ 2 Microcontroller Programmer or the MPLAB® ICD 2 using the RJ-11 to 6-pin inline adapter (AC164110).

1.2   HIGHLIGHTS

This chapter discusses:

- Devices supported by the 44-Pin Demo Board
- The 44-Pin Demo Board Overview
- Running the Default Demonstration

1.3   DEVICES SUPPORTED BY THE 44-PIN DEMO BOARD

The 44-Pin Demo Board can be used with virtually any 44-pin Thin Quad Flatpack (TQFP) PIC MCU. The assembled 44-Pin Demo Board is populated with a PIC16F887-I/PT microcontroller.

Additional 44-Pin Demo Boards can be ordered from Microchip Technology and distributors. Part number, DM164120-2, comes with one assembled and two blank 44-Pin Demo Boards. The blank demo board can be used for evaluating or prototyping circuits using any of the 44-pin devices listed below.

44-pin TQFP Flash Devices:

• PIC16F74	• PIC16F747	• PIC16F77
• PIC16F777	• PIC16F871	• PIC16F874A
• PIC16F877A	• PIC16F887	• PIC16F914
• PIC16F917	• PIC18F4220	• PIC18F4221
• PIC18F4320	• PIC18F4321	• PIC18F4331
• PIC18F4410	• PIC18F4420	• PIC18F4423
• PIC18F4431	• PIC18F4450	• PIC18F4455
• PIC18F4480	• PIC18F44J10	• PIC18F4510
• PIC18F4515	• PIC18F4520	• PIC18F4523
• PIC18F4525	• PIC18F4550	• PIC18F4580
• PIC18F4585	• PIC18F45J10	• PIC18F4610
• PIC18F4620	• PIC18F4680	• PIC18F4682
• PIC18F4685		

44-Pin Demo Board User's Guide

1.4   44-PIN DEMO BOARD OVERVIEW

The 44-Pin Demo Board is populated with a PIC16F887 MCU (U1), eight LEDs (DS1-DS8), push button (SW1) and potentiometer (RP1). The board layout is shown in Figure 1-1. The demo board has several test points to access the I/O pins of the MCU and a surface mount prototyping area. The MCU can be programmed with the PICkit™ 2 Microcontroller Programmer from header P1.

FIGURE 1-1:   44-PIN DEMO BOARD
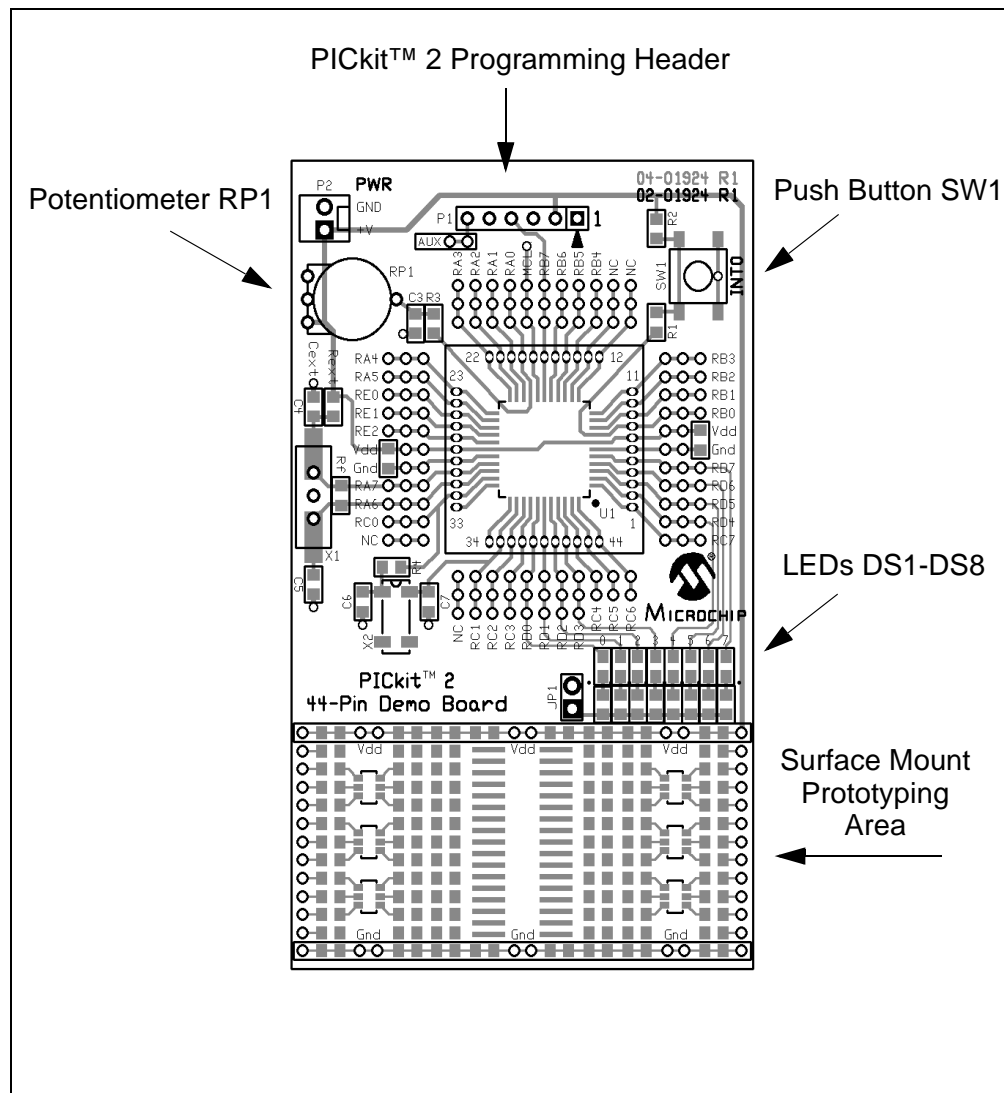
1.5   RUNNING THE DEFAULT DEMONSTRATION

The assembled 44-Pin Demo Board comes preprogrammed with a demonstration program. To use this program, power the demo board (3.0-5.5 VDC) using a PICkit™ 2 Microcontroller Programmer, or a bench power supply connected to header P2. To use the PICkit™ 2 Microcontroller Programmer, connect it to a PC USB port using the USB cable. Start the PICkit™ 2 Microcontroller Programmer PC application and click on the target power box to apply power to the demo board. The demo program will blink the eight red lights in succession. Press the push button switch, labeled SW1, and the sequence of the lights will reverse. Rotate the potentiometer, RP1, and the light sequence will blink at a different rate.

Chapter 2. Mid-Range PIC® Microcontroller Architectural Overview

2.1 INTRODUCTION

This chapter provides a simple overview of the mid-range PIC® microcontroller architecture.

FIGURE 2-1: **SIMPLIFIED MID-RANGE PIC® MICROCONTROLLER BLOCK DIAGRAM**
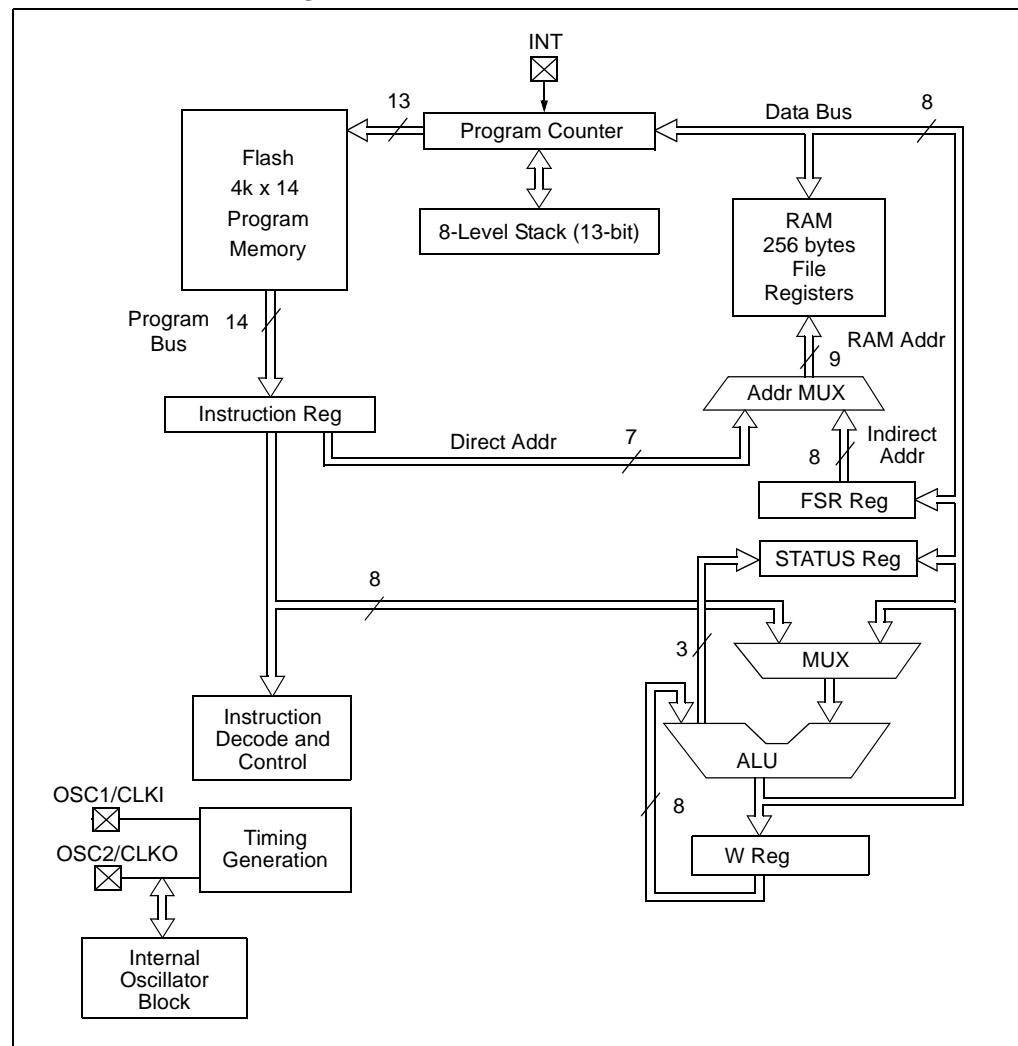
2.2 MEMORY ORGANIZATION

PIC® microcontrollers are designed with separate program and data memory areas. This allows faster execution as the address and data busses are separate and do not have to do double duty.

44-Pin Demo Board User's Guide

Data Memory is held in **file registers**. Instructions referring to file registers use 7 bits, so only 128 file registers can be addressed. Multiple file registers are arranged into "banks". Two bits in the STATUS register, RP0 and RP1, allow accessing different banks. These two bits effectively become the top two bits of the file register address.

The additional banks may or may not be implemented, depending on the device.

Mid-range devices reserve the first 32 addresses of each bank for **Special Function Registers** (SFRs). SFRs are how the program interacts with the peripherals and some core features. The controls and data registers are memory mapped into the SFR space. Addresses above 0x20 to the end of each bank are **General Purpose Registers** (GPRs), where program variables may be stored.

Some frequently used registers may be accessed from any bank. For example, the STATUS register is always available no matter which bank is selected via the RP bits. The last 16 bytes (0x70-0x7F) of the GPRs may also be accessed from any bank.

Program Memory is accessed via a 13-bit Program Counter (PC). The lower 8 bits are accessible via SFR (PCL), and the upper 5 are at a PCLATH. See the PIC16F88X Data Sheet's (DS41291) section on PCL and PCLATH for more details on the PC. PCLATH becomes important when program memory size exceeds 1k instructions, and also for the table look-up in Lesson 12.

Mid-range PIC® MCUs may be clocked by a number of different devices. Unless otherwise noted, the lessons in this manual use the Internal Oscillator running at 4 MHz.

2.3   INSTRUCTION FORMATS

Most instructions follow one of three formats: Byte oriented instructions, Bit oriented instructions and Literal instructions.

Byte instructions contain a 7-bit data address, a destination bit, and a 6-bit op code. The data address plus the RP0 and RP1 bits create a 9-bit data memory address for one operand. The other operand is the Working register (called W or WREG). After the instruction executes, the destination bit (d) specifies whether the result will be stored in the WREG ('w') or back in the original file register ('f'). For example:

```
ADDWF   data,f
```

adds the contents of WREG and file register data, with the result going back into data.

Bit instructions operate on a specific bit within a file register. They contain 7 bits of data address, a 3-bit number and the remaining 4 bits are op code. These instructions may set or clear a specific bit within a file register. They may also be used to test a specific bit within a file register. For example:

```
BSF       STATUS,RP0
```

set the RP0 bit in the STATUS register.

Literal instructions contain the data operand within the instruction. The WREG becomes the other operand. Calls and GOTO's use 11 bits as a literal address.

```
MOVLW   'A'
```

Moves the ASCII value of 'A' (0x41) into the WREG.

2.4   ASSEMBLER BASICS

<u>Numbers in the Assembler</u>

Unless otherwise specified, the assembler assumes any numeric constants in the program are hexadecimal (base 16). Binary (base 2), Octal (base 8), Decimal (base 10), and ASCII coding are also supported.

Hexadecimal	`12 or 0x12 or H'12'`
Decimal	`.12 or D'12'`
Octal	`O'12'`
Binary	`B'00010010'`
ASCII	`A'c' or 'c'`

<u>Org</u> (Origin)

`Org` tells the Assembler an address at which to start generating code. Normally we start coding at the Reset vector address '0000', but it could be anywhere. Baseline devices have a Reset vector at the last location in program memory, so it's good practice to have a `GOTO` instruction pointing to the beginning of the program.

<u>End</u>

`End` tells the assembler to stop assembling. There must be one at the end of the program. It does not necessarily have to be at the end of the file, but nothing after the end statement will be assembled.

<u>Defining Data Memory Locations</u>

There are three ways to name a location (see Example 2-1). All are equivalent in that the location name label will be substituted with the value assigned to it during assembly.

EXAMPLE 2-1:   DEFINING DATA MEMORY

```
#define Length   0x20      ;c-like syntax

Length   equ     0x20      ;equate 0x20 with the symbol

   cblock       0x20      ;start a block of variables
Length                     ;this will be at address 0x20
Width                      ;this will be at address 0x21
Area:2                     ;this is 2 bytes long, starting at
                           ;address 0x22
Girth                      ;this will be at address 0x24
   endc
```

Note that if used as a literal, the label names will take on the value assigned. If used as an address operand in an instruction, the label names point to the contents of the file register with the address of the label's value.

Unless there is a reason to name a specific location address, the `cblock/endc` method is preferred. The advantage is that as variables come and go through the development process, the `cblock` keeps the block to a minimum. Using one of the other methods, you may have to go back and find an unused location.

NOTES:

Chapter 3. 44-Pin Demo Board Lessons

3.1 INTRODUCTION

The following lessons cover basic 44-Pin Demo Board features. Refer to applicable documents as needed. Any updates to the applicable documents are available on Microchip's web site.

The code and hex files may be installed from the PICkit™ 2 CD-ROM under path `Install /Lessons`.

3.2 44-PIN DEMO BOARD LESSONS

- Lesson 1: Hello World (Light a LED)
- Lesson 2: Blink (Delay Loop)
- Lesson 3: Rotate (Move the LED)
- Lesson 4: Analog-to-Digital
- Lesson 5: Variable Speed Rotate
- Lesson 6: Switch Debounce
- Lesson 7: Reversible Variable Speed Rotate
- Lesson 8: Function Calls
- Lesson 9: Timer0
- Lesson 10: Interrupts
- Lesson 11: Indirect Data Addressing
- Lesson 12: Look-up Table (ROM Array)

3.2.1 Lesson 1: Hello World (Light a LED)

The first lesson shows how to turn on a LED. This is the PIC® microcontroller version of "Hello World" and discusses the I/O pin structures.

New Instructions

> BSF   Bit set
>
> BCF   Bit clear

The LEDs are connected to I/O pins RD0 through RD7. When one of these I/O pins drives high, the LED turns on. The I/O pins can be configured for input or output. On start-up, the default is input. The TRIS Special Function Register bits use the convention of '0' for output and '1' for input. We want digital output so these must be configured.

EXAMPLE 3-1:   PICKIT 2, LESSON 1: "HELLO WORLD"

```
; PICkit 2 Lesson 1 - "Hello World"
;
#include <p16F887.inc>
        __CONFIG    _CONFIG1, _LVP_OFF & _FCMEN_OFF & _IESO_OFF &
                    _BOR_OFF & _CPD_OFF & _CP_OFF & _MCLRE_OFF &
                    _PWRTE_ON & _WDT_OFF & _INTRC_OSC_NOCLKOUT
        __CONFIG    _CONFIG2, _WRT_OFF & _BOR21V


    org 0
Start:
    BSF     STATUS,RP0  ; select Register Bank 1
    BCF     TRISD,0     ; make IO Pin RD0 an output
    BCF     STATUS,RP0  ; back to Register Bank 0
    BSF     PORTD,0     ; turn on LED RD0 (DS0)
    GOTO     $          ; wait here
    END
```

Now lets look at the program that makes this happen.

;	Starts a comment. Any text on the line following the semicolon is ignored.
#include	Brings in an include file defining all the Special Function Registers available on the PIC16F887. Also, it defines valid memory areas. These definitions match the names used in the device data sheet.
__Config	Defines the Configuration Word. The labels are defined in the p16F887.inc file. The labels may be logically ANDed together to form the word.
Org 0	Tells the assembler where to start generating code. Code may be generated for any area of the part. Mid-range PIC® microcontroller devices start at address '0', also called the Reset vector.
BCF TRISC,0	Tells the processor to clear a bit in a file register. TRISD is the tri-state register for pin 0 of PORTD. A '1' in the register makes the pin an input; a '0' makes it an output. We want to make it an output, so the bit must be cleared.
BSF PORTD,0	Tells the processor to set pin 0 of PORTD. This will force the I/O pin to a high condition turning on the LED.
GOTO $	Tells the processor to go to the current instruction.

For more information, refer to the I/O Ports section of the PIC16F882/883/884/886/887 Data Sheet (DS41291).

3.2.2   Blink (Delay Loop)

The first lesson showed how to turn on a LED, this lesson shows how to make it blink. While this might seem a trivial change from Lesson 1, it gives a context to explore several more instructions.

New Instructions

CLRF	Clear file register
INCF	Increment file register
DECF	Decrement file register
INCFSZ	Increment file register, Skip next instruction if zero
DECFSZ	Decrement file register, Skip next instruction if zero
GOTO	Jump to a new location in the program

EXAMPLE 3-2:   PICKIT 2, LESSON 2: BLINK

```
Loop
    BSF   PORTD,0     ;turn on LED D0
    BCF   PORTD,0     ;turn off LED D0
    GOTO  Loop        ;do it again
```

While adding a BCF instruction and making it loop will make it blink. It will blink so fast you won't see it, it will only look dim. That loop requires 4 instruction times to execute. The first instruction turns it on. The second one turns it off. The GOTO takes two instruction times, which means it will be on for 25% of the time.

As configured, the PIC® microcontroller executes 1 million instructions per second. At this rate, the blinking needs to be slowed down so that the blinking can be seen, which can be done by using a delay loop.

> **Note:**   Counting cycles – Relating clock speed to instruction speed. The processor requires 4 clocks to execute an instruction. Since the internal oscillator as used in these lessons runs at 4 MHz, the instruction rate is 1 MHz.

Increment or Decrement a File Register

The INCFSZ and DECFSZ instructions add or subtract one from the contents of the file register and skips the next instruction when the result is zero. One use is in the delay loop as shown in Example 3-3.

CLRF Clears the counter location.

DECFSZ Decrements the location, and if the result is zero, the next instruction is skipped.

EXAMPLE 3-3: DELAY LOOP

```
Short Loop

    CLRF      Delay
Loop
    DECFSZ    Delay,f
    GOTO      Loop

Long Loop

    CLRF      Delay1
    CLRF      Delay2
Loop
    DECFSZ    Delay1,f
    GOTO      Loop
    DECFSZ    Delay2,f
    GOTO      Loop
```

The GOTO Loop (in Example 3-3) backs up and does it again. This loop takes 3 instruction times; one for the decrement and two for the GOTO (see note) and the counter will force it to go around 256 times, which takes it a total of 768 instruction times (768 μs) to execute.

Even that is still too fast for the eye to see. It can be slowed down even more by adding a second loop around this one.

The inner loop still takes 768 μs plus 3 for the outer loop, but now it's executed another 256 times, (768 + 3) * 256 = 197376 μs = 0.197s.

> **Note:** GOTO instructions take two instructions due to the pipelined design of the processor. The processor fetches the next instruction while executing the current instruction. When a program branch occurs, the fetched instruction is not executed.

Open Blink.asm and build the lesson. Next, import the hex file into the PICkit 2 and program the device. Note the LED now flashes at about a 2.5 Hz rate.

3.2.3 Lesson 3: Rotate (Move the LED)

Building on Lessons 1 and 2, which showed how to light up a LED and then make it blink with a delay loop, this lesson adds rotation. It will light up DS8 and then shift it to DS7, then DS6 and on down to DS1, and then back to DS8.

New Instructions

MOVLW	Loads WREG with a literal value
MOVWF	Moves the contents of WREG to a file register
MOVF	Moves the contents of a file register, either to WREG or back into the file register (see note)
RRF	Rotate file register right
RLF	Rotate file register left

> **Note:** Moving a file register to itself looks like a NOP at first. However, it has a useful side effect in that the Z flag is set to reflect the value. In other words, MOVF fileregister,f is a convenient way to test whether or not the value is zero without affecting the contents of the WREG.

Rotate Program Flow

- First, initialize the I/O port and the Display,
- Copy the Display variable to the I/O Port, then
- Delay for a little while
- Rotate the display

FIGURE 3-1: ROTATE PROGRAM FLOW
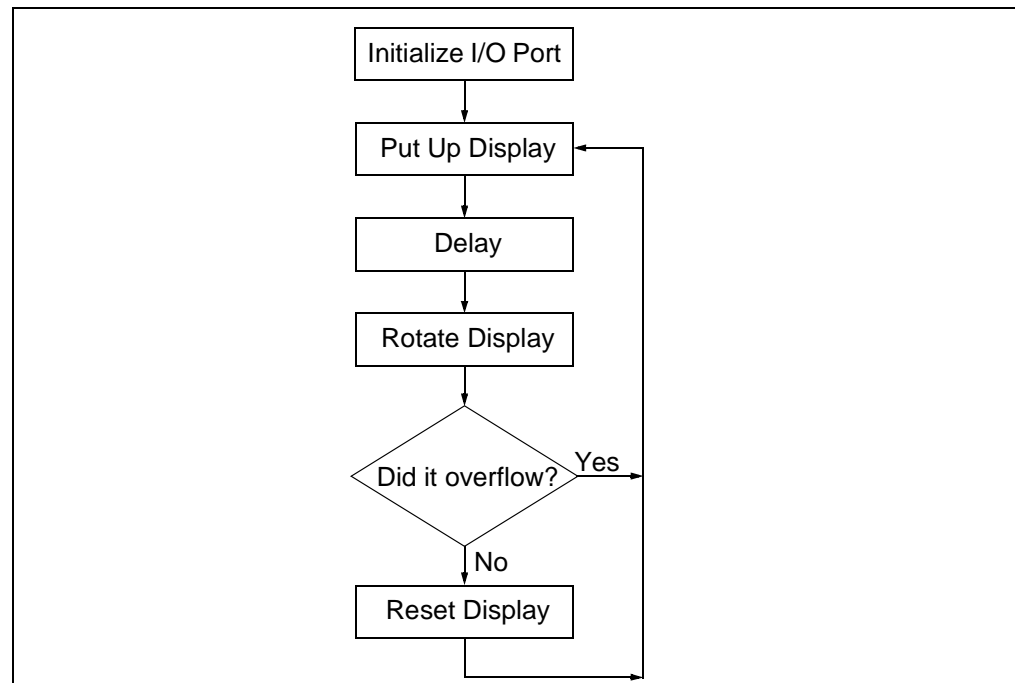
Rotate

The rotate instructions (RRF or RLF) shift all the bits in the file register right or left by one position, through the Carry bit. The Carry bit is shifted into the byte and receives the bit shifted out of the byte. The Carry bit should be cleared before rotation so unwanted bits are not introduced into the display byte. The Carry bit also indicates when the display byte is empty. When it is, reinsert the '1' at bit 3.

PIC microcontrollers have two rotate instructions: Rotate Left (RLF) and Rotate Right (RRF). These instructions rotate the contents of a file register and Carry bit one place. The Carry bit is found in the STATUS Special Function Register.

FIGURE 3-2:   **ROTATE LEFT**

Carry   File Register
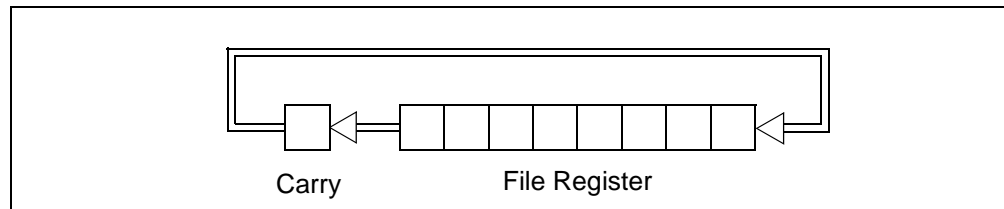
EXAMPLE 3-4:   **ROTATE EXAMPLE**

```
Start:
  BSF         STATUS,RP0 ; select Register Bank 1
  CLRF        TRISD      ; make IO PortD all output
  BCF         STATUS,RP0 ; back to Register Bank 0
  MOVLW       0x80
  MOVWF       Display
MainLoop:
  MOVF        Display,w  ; Copy the display to the LEDs
  MOVWF       PORTD
OndelayLoop:
  DECFSZ      Delay1,f   ; Delay .197 s
  GOTO        OndelayLoop
  DECFSZ      Delay2,f
  GOTO        OndelayLoop

  BCF         STATUS,C   ; ensure the carry bit is clear
  RRF         Display,f  ; rotate display right
  BTFSC       STATUS,C   ; Did the bit rotate into the carry?
  BSF         Display,7  ; yes, put it into bit 7.
  GOTO        MainLoop
```

3.2.4   Lesson 4: Analog-to-Digital

This lesson shows how to configure the ADC, run a conversion, read the analog voltage controlled by the potentiometer (RP1) on the board, and display the high order 8 bits on the display.

The PIC16F887 has an on-board Analog-to-Digital Converter (ADC) with 10 bits of resolution on any of 14 channels. The converter can be referenced to the device's V_{DD} or an external voltage reference. The 44-pin Demo Board references it to V_{DD} as provided by the PICkit 2 Microcontroller Programmer. The answer from the ADC is represented by a ratio of the voltage to the reference.

$$ADC = V/V_{REF} * 1023$$

Converting the answer from the ADC back to voltage requires solving for V.

$$V = ADC/1023 * V_{REF}$$

Two of the three factors on the right side of the equation are constants and may be calculated in advance. This eliminates the need to actually divide, but still requires fixed or floating point multiply to solve the equation on the fly.

However, sometimes, such as when reading a sensor, calculating the voltage is only the first step. There may be additional math to calculate the meaningful data from the sensor. For example, when reading a thermistor, calculating the voltage is only the first step on the way to getting the temperature.

There are other means to convert ADC values, including a straight table look-up or a piece-wise linear interpolation. Each of these represents different speed/memory trade-offs.

The schematic (**Appendix A. "Hardware Schematics"**) shows the wiper on the potentiometer is connected to pin RA0 on the PIC16F887.

Here's the checklist for this lesson:

- Configure PORTA as an analog input, TRISA<0> = 1, ANSEL<0> = 1
- Select justification and V_{REF} source in ADCON1.
- Select clock scaling and channel in ADCON0.

3.2.4.1 ADCON1

The ADCON1 register sets the justification of the 10-bit result in the 16-bit result read through registers ADRESL and ADRESH. Setting the result to Left Justified means the 8 Most Significant bits and read from ADRESH and the 2 Least Significant bits are read from bits 7 and 6 of ADRESL. ADCON1 also sets the voltage reference sources VREF+ and VREF-. VREF- is the voltage at which the result will be zero. VREF+ is the voltage at which the result will be maximum (1023). We select the PIC16F887 VSS and VDD voltages respectively.

REGISTER 3-1:   ADCON1: A/D CONTROL REGISTER 1

R/W-0	U-0	R/W-0	R/W-0	U-0	U-0	U-0	U-0
ADFM	—	VCFG1	VCFG0	—	—	—	—
bit 7							bit 0

Legend:		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared x = Bit is unknown

bit 7   **ADFM:** A/D Conversion Result Format Selection bit
         1 = Right justified
         0 = Left justified

bit 6   **Unimplemented:** Read as '0'

bit 5   **VCFG1:** Voltage Reference bit
         1 = VREF- pin
         0 = Vss

bit 4   **VCFG0:** Voltage Reference bit
         1 = VREF+ pin
         0 = VDD

bit 3-0   **Unimplemented:** Read as '0'

3.2.4.2 ADCON0

ADCON0 controls the ADC operation. Bit 0 turns on the ADC module and bit 1 starts a conversion. Bits <7:6> select the ratio between the processor clock and conversion speed and bits <5:2> select which channel the ADC will operate on. The ratio between the processor clock and conversion speed is important because the ADC needs at least 1.6 μs per bit. Accuracy degrades if the clock speed is too high. As the processor clock speed increases, an increasingly large divider is necessary to keep the conversion bit speed above 1.6 μs. Four MHz gives the fastest conversion rate above the minimum at 8:1 ratio. This results in a conversion speed of 2 μs per bit. Refer to the "TAD vs. Device Operating Frequencies" Table in the Analog-to-Digital section of the PIC16F882/883/884/886/887 Data Sheet (DS41291) for recommended configurations.

For purposes of this lesson, the ADC must be turned on and pointed to channel AN0 on pin RA0.

The ADC needs about 5 μs, after changing channels, to allow the ADC sampling capacitor to settle. Finally, we can start the conversion by setting the GO bit in ADCON0. The bit also serves as the $\overline{\text{DONE}}$ flag. That is, the ADC will clear the same bit when the conversion is complete. The answer is then available in ADRESH:ADRESL. This lesson takes the high order 8 bits of the result and copies them to the display LEDs attached to PORTD.

See the Analog-to-Digital section in the PIC16F882/883/884/886/887 Data Sheet (DS41291) for more details on the ADC module.

REGISTER 3-2: ADCON0: A/D CONTROL REGISTER 0

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ADCS1	ADCS0	CHS3	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	ADON
bit 7							bit 0

Legend:		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared x = Bit is unknown

bit 7-6 **ADCS<1:0>:** A/D Conversion Clock Select bits
00 = FOSC/2
01 = FOSC/8
10 = FOSC/32
11 = FRC (clock derived from a dedicated internal oscillator = 500 kHz max)

bit 5-2 **CHS<3:0>: Analog Channel Select bits**
0000 = AN0
0001 = AN1
0010 = AN2
0011 = AN3
0100 = AN4
0101 = AN5
0110 = AN6
0111 = AN7
1000 = AN8
1001 = AN9
1010 = AN10
1011 = AN11
1100 = AN12
1101 = AN13
1110 = CVREF
1111 = Fixed Ref (0.6 volt fixed reference)

bit 1 **GO/$\overline{\text{DONE}}$:** A/D Conversion Status bit
1 = A/D conversion cycle in progress. Setting this bit starts an A/D conversion cycle. This bit is automatically cleared by hardware when the A/D conversion has completed.
0 = A/D conversion completed/not in progress

bit 0 **ADON:** ADC Enable bit
1 = ADC is enabled
0 = ADC is disabled and consumes no operating current

3.2.5   Lesson 5: Variable Speed Rotate

Lesson 5 combines Lessons 3 and 4 by using the Analog-to-Digital Converter (ADC) to control the speed of rotation.

New Instructions

```
BTFSS          Bit test, skip if set
BTFSC          Bit test, skip if clear
```

A conversion is run on every pass through the main loop. The result controls the length of the outer loop (see Example 3-5).

EXAMPLE 3-5:   VARIABLE SPEED ROTATE EXAMPLE

```
      ...
      BSF     ADCON0,GO_DONE    ; start conversion
      BTFSS   ADCON0,GO_DONE    ; this bit will change to zero when
                                ; the conversion is complete
      GOTO    $-1
      MOVF    ADRESH,w          ; Copy the display to the LEDs
      ADDLW   1
      MOVWF   Delay2

A2DDelayLoop:
      DECFSZ  Delay1,f          ; Delay Loop shortened by the ADResult
                                ; as controlled by the potentiometer.
      GOTO    A2DDelayLoop
      DECFSZ  Delay2,f
      GOTO    A2DDelayLoop
```

FIGURE 3-3:   VARIABLE SPEED ROTATE PROGRAM FLOW
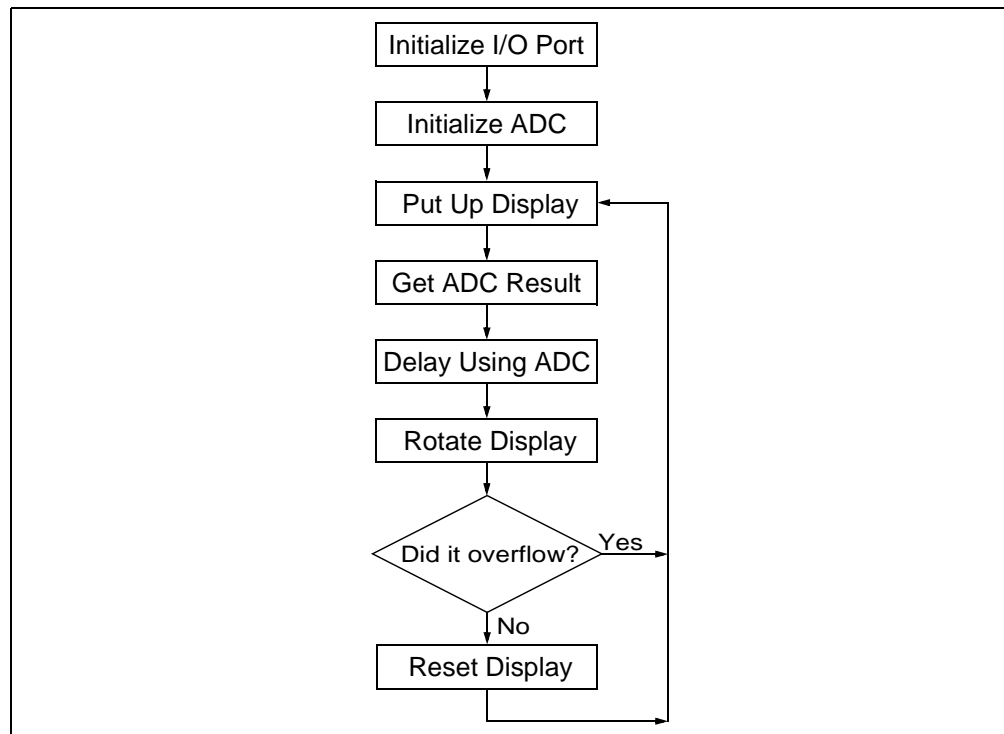
3.2.6 Lesson 6: Switch Debouncing

Mechanical switches play an important and extensive role in practically every computer, microprocessor and microcontroller application. Mechanical switches are inexpensive, simple and reliable. However, switches can be very noisy electrically. The apparent noise is caused by the closing and opening action that seldom results in a clean electrical transition. The connection makes and breaks several, perhaps even hundreds, of times before the final switch state settles.

The problem is known as switch bounce. Some of the intermittent activity is due to the switch contacts actually bouncing off each other. Imagine slapping two billiard balls together. The hard non-resilient material doesn't absorb the kinetic energy of motion.

Instead, the energy dissipates over time and friction in the bouncing action against the forces push the billiard balls together. Hard metal switch contacts react in much the same way. Also, switch contacts are not perfectly smooth. As the contacts move against each other, the imperfections and impurities on the surfaces cause the electrical connection to be interrupted. The result is switch bounce.

The consequences of uncorrected switch bounce can range from being just annoying to catastrophic. For example, imagine advancing the TV channel, but instead of getting the next channel, the selection skips one or two. This is a situation a designer should strive to avoid.

Switch bounce has been a problem even before the earliest computers. The classic solution involved filtering, such as through a resistor-capacitor circuit, or through re-settable shift registers (see Figure 3-4 and Figure 3-5). These methods are still effective but they involve additional cost in material, installation and board real estate. Debouncing in software eliminates these additional costs.

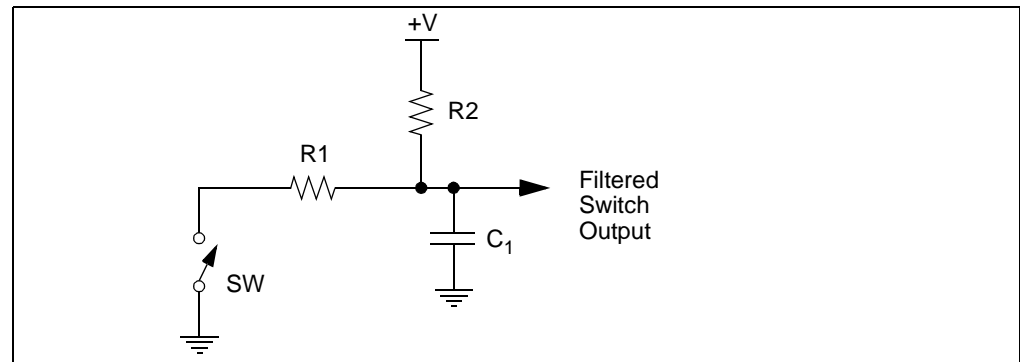**FIGURE 3-4:** **FILTERING DEBOUNCE SOLUTION**

FIGURE 3-5: **SHIFT REGISTER DEBOUNCE SOLUTION**
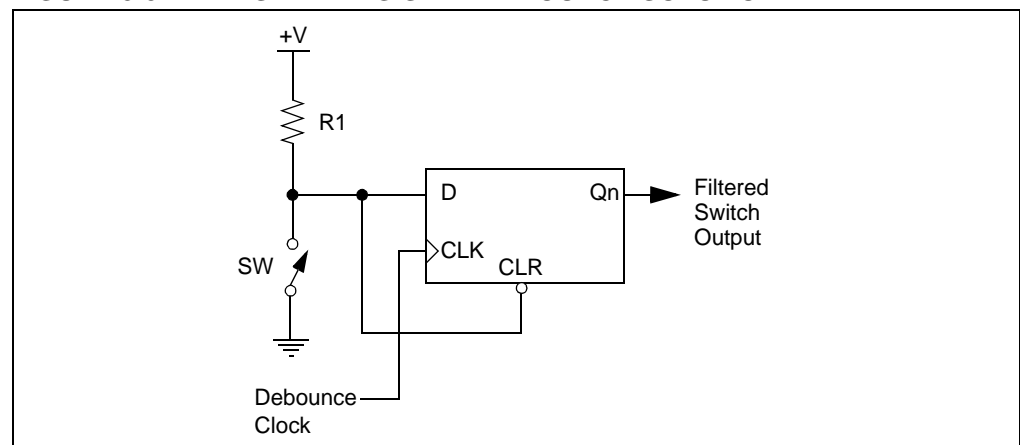
One of the simplest ways to switch debounce is to sample the switch until the signal is stable or continue to sample the signal until no more bounces are detected. How long to continue sampling requires some investigation. However, 5 mS is usually plenty long, while still reacting fast enough that the user won't notice it.

Lesson 6 shows how to sample the line at a 1 mS rate waiting for a number of sequential state changes, which is a simple matter of counting to 5, then resetting the counter every time it's still in the original unchanged state.

The Switch on the 44-Pin Demo Board doesn't bounce much, but it is good practice to debounce all switches in the system.

FIGURE 3-6: SIMPLE SWITCH DEBOUNCE PROGRAM FLOW
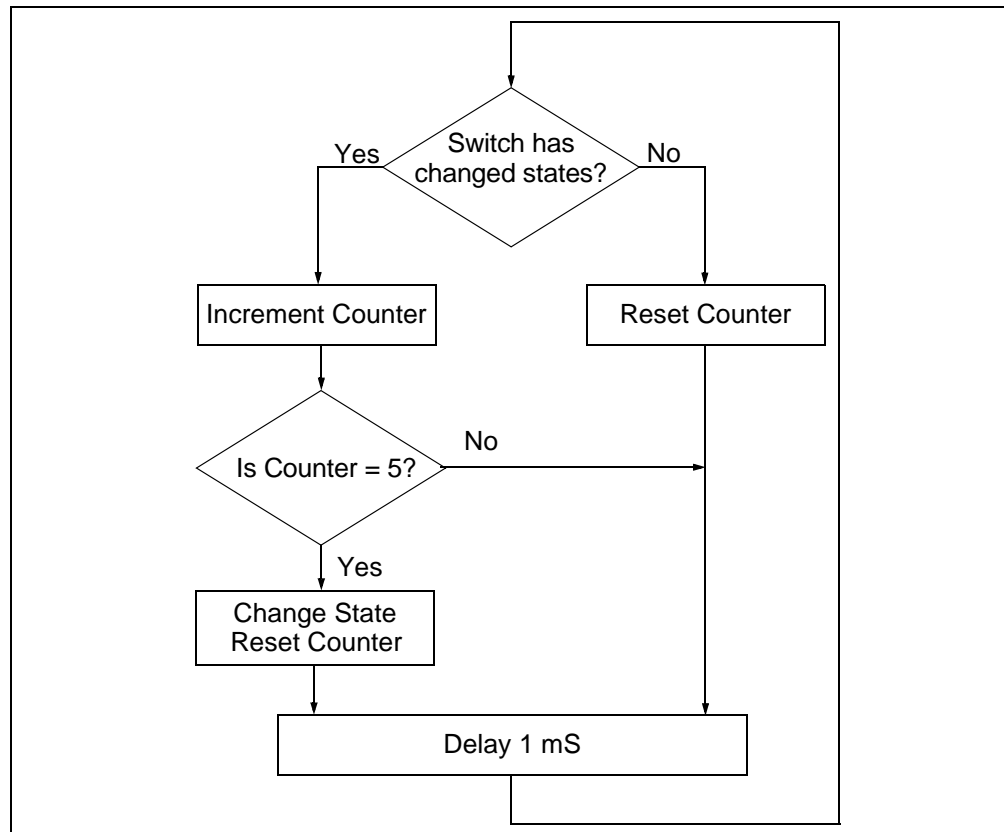
3.2.7 Lesson 7: Reversible Variable Speed Rotate

Lesson 7 combines Lessons 5 and 6 using the button to reverse the direction of rotation when the button is pressed and adjusting the potentiometer to control the speed of rotation.

The program needs to keep track of rotation direction and new code needs to be added to rotate in the other direction.

Lesson 5 rotates right and checks for a '1' in the Carry bit to determine when to restart the sequence. In Lesson 7, we'll also need to rotate left and check for a '1' in bit 7 of the display. When the '1' shows up in bit 7 of the display, insert it into the bit 0 position.

EXAMPLE 3-6:   REVERSIBLE VARIABLE SPEED ROTATE EXAMPLE

```
Original Version:

Rotate:
    RRF        Display,f
    BTFSC      STATUS,C          ; Did the bit rotate into the carry?
    BSF        Display,7         ; yes, put it into bit 7.


Bidirectional Version:

Rotate:
    BCF        STATUS,C          ; ensure the carry bit is clear
    BTFSS      Direction,0
    GOTO       RotateLeft

RotateRight:
    RRF        Display,f
    BTFSC      STATUS,C          ; Did the bit rotate into the carry?
    BSF        Display,7         ; yes, put it into bit 7.


    GOTO       MainLoop
RotateLeft:
    RLF        Display,f
    BTFSC      STATUS,C          ; did it rotate out of the display
    BSF        Display,0         ; yes, put it into bit 0
    GOTO       MainLoop
```

3.2.8   Lesson 8: Function Calls

Lesson 8 shows the reversible LEDs but with the Delay Loop rewritten as a function.

New Instructions

CALL	Invokes functions or subroutines
RETURN	Terminates functions or subroutines
RETLW	Terminates functions or subroutines

Functions or Subroutines are invoked with the CALL instruction and terminated with a RETURN or RETLW instruction. RETURN jumps back to the original program at the location following the CALL. RETLW also returns to the calling program, but loads the WREG with a constant.

The mid-range PIC® microcontroller device's CALL stack can hold up to 8 return addresses.

If a ninth CALL is made, it will overwrite the first one and then the program will not be able to RETURN all the way back.

Passing Arguments

Arguments to the subroutine may be passed in a number of ways. WREG is a convenient place to pass one byte and the FSR may be used to pass another byte, if not otherwise used. If more data must be passed, a buffer must be allocated.

When the Delay function is pulled out to a subroutine, the ADC result is moved into WREG, then the CALL transfers control to the Delay subroutine. The RETURN transfers control to the MOVLW following the CALL.

EXAMPLE 3-7:   FUNCTION CALL EXAMPLE

```
    MOVF       ADRESH,w     ; Move conversion value (delay) to w
    ADDLW      1            ; add 1 otherwise entering with 0 takes
                            ; longer than entering with 1.
    CALL       Delay        ; Call delay function
                            ; returns here when done
...
    GOTO       XXX

; Delay Function.  Enter with number of 771uS delays in Wreg
Delay:
    MOVWF      Delay2
DelayLoop:
    DECFSZ     Delay1,f
    GOTO       DelayLoop
    DECFSZ     Delay2,f
    GOTO       DelayLoop
    RETURN     ; goes back to instruction after call
```

3.2.9   Lesson 9: Timer0

Timer0 is a counter implemented in the processor. It may be used to count processor clock cycles or external events. Lesson 9 configures it to count instruction cycles and set a flag when it rolls over. This frees up the processor to do meaningful work rather than just counting cycles for a delay.

Timer0 is an 8-bit counter with an optional prescaler, which is configured to divide by 256 before reaching the Timer0 counter.

FIGURE 3-7:   **TIMER0 SIMPLIFIED**
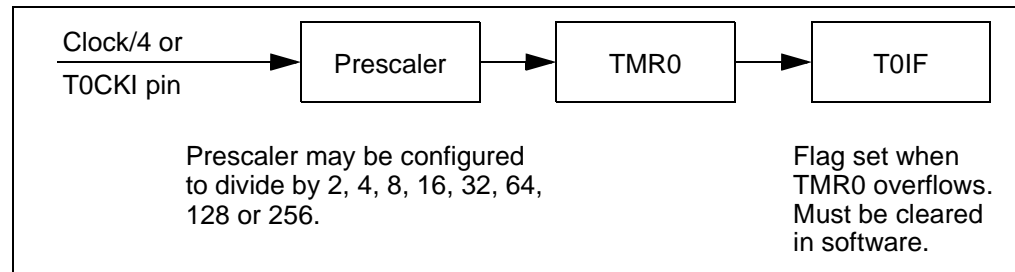
TMR0 is a Special Function Register (SFR) and may be read or modified by the program. The prescaler is not a SFR and thus cannot be read or modified by the program. However, writing to TMR0 clears the prescaler.

The timer may be fed either by the same clock that drives the processor or by an external event. Driven by the processor clock, it increments once for every instruction cycle. This is a convenient method of marking time and is better than delay loops, as it allows the processor to work on the problem rather than waste cycles in delay loops.

The prescaler is configured through the OPTION_REG, see Figure 3-8.

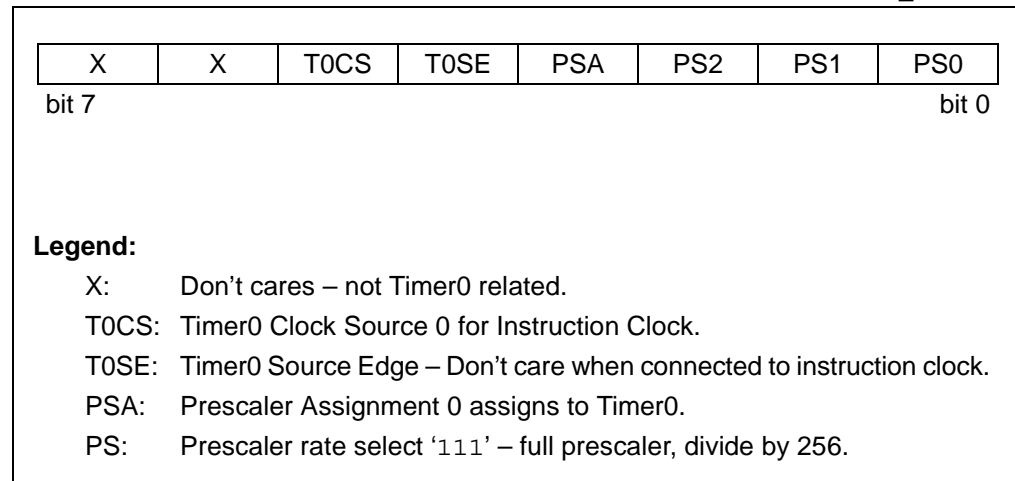**FIGURE 3-8:**   **PRESCALER CONFIGURATION THROUGH OPTION_REG**

| X | X | T0CS | T0SE | PSA | PS2 | PS1 | PS0 |

bit 7   bit 0

Legend:

X: Don't cares – not Timer0 related.

T0CS: Timer0 Clock Source 0 for Instruction Clock.

T0SE: Timer0 Source Edge – Don't care when connected to instruction clock.

PSA: Prescaler Assignment 0 assigns to Timer0.

PS: Prescaler rate select '111' – full prescaler, divide by 256.

Lesson 9 configures Timer0 with the prescaler for a maximum delay on Timer0. The prescaler will divide the processor clock by 256 and Timer0 will divide that by 256 again. Thus, the Timer0 Flag will be set every 65536 μs (0.0000001 second * 256 * 256), or about 15 times a second. The main program sits in a loop waiting for the rollover and when it does, it increments the display and then loops back.

EXAMPLE 3-8:   TIMER0 EXAMPLE

```
   org 0
   BSF        STATUS,RP0  ; Bank 1
   MOVLW      b'00000111' ; configure Timer0.  Sourced from the
                          ; Processor clock
   MOVWF      OPTION_REG  ; Maximum Prescaler
   CLRF       TRISD       ; Make PortD all output
   CLRF       Display
   BCF        STATUS,RP0  ; Bank 0

ForeverLoop:
   BTFSS      INTCON,T0IF ; wait here until Timer0 rolls over
   GOTO       ForeverLoop
   BCF        INTCON,T0IF ; flag must be cleared in software
   INCF       Display,f   ; increment display variable
   MOVF       Display,w   ; send to the LEDs
   MOVWF      PORTD
   GOTO       ForeverLoop
```

3.2.10   Lesson 10: Interrupts

New Instructions

    RETFIE   Return from Interrupt
    SWAPF   Swap nibbles in file register

Interrupt Sources

Most of the peripherals can generate an interrupt, and some of the I/O pins may be configured to generate an interrupt when they change state.

When a peripheral needs service or an event occurs, it sets its interrupt flag. Each interrupt flag is ANDed with its enable bit and then these are ORed together to form a Master Interrupt. This master interrupt is ANDed with the Global Interrupt Enable (GIE). See the Interrupt Logic Figure in the PIC16F882/883/884/886/887 Data Sheet (DS41291) for a complete drawing of the interrupt logic. The enable bits allow the PIC microcontroller to limit the interrupt sources to certain peripherals.

FIGURE 3-9:   INTERRUPT LOGIC SIMPLIFIED
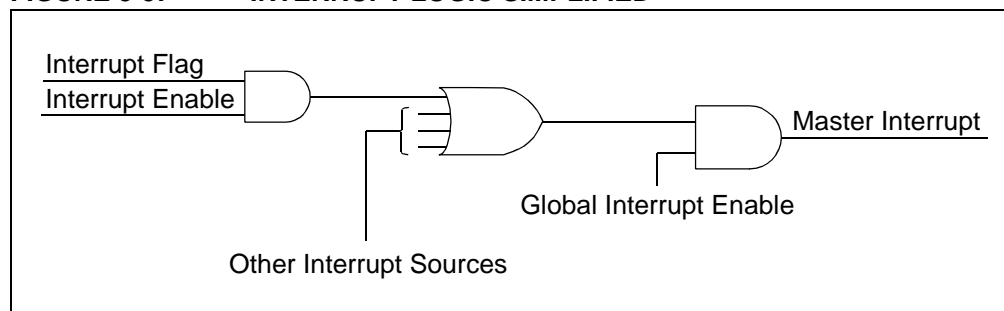
When the master interrupt line is asserted, the PIC microcontroller finishes the current instruction, stores the next address on the CALL stack then jumps to the Interrupt Service Routine (ISR). It also clears the GIE bit, preventing another interrupt from occurring while servicing the current one.

Save Current Context

The first thing the ISR must do is to save the current context of the processor so it can be restored before returning to the main program. Certain SFRs that may be changed in the ISR should be saved, such as the WREG and STATUS registers at the very least. Make sure to save the WREG first, as moving other SFRs into it will destroy the value needing to be saved. The last 16 bytes of each PIC16F887 file register page are unbanked and are good places to save the context, as they may be accessed from any register page without regard to the RP0 and RP1 bits in the STATUS register. The location of unbanked registers may vary from part to part. Check the register map to find the unbanked region for a specific part.

Identify Triggering Event

Next, the ISR has to figure out what triggered the interrupt. It has to check the interrupt flags to determine what caused the interrupt. When it finds the source, it can then service the peripheral.

Restore Context

Once the peripheral is serviced, it needs to restore the context and resume the main program. The "context" is the state of the SFRs when the interrupt occurred. Restoring the context is a little harder than it might seem at first. The obvious method doesn't work because the MOVF W_Temp,w may affect the Z flag, which was restored in the previous instruction. Instead, a pair of SWAPF instructions can restore WREG without affecting the flags in the STATUS register. SWAPF exchanges the high and low nibbles. The first SWAPF switches the nibbles in the file register and the second one switches them back and puts the result in WREG.

EXAMPLE 3-9:   CONTEXT RESTORE

```
;incorrect context restore
MOVF  STATUS_Temp,w
MOVWF STATUS
MOVF  W_Temp            ;this may change the Z bit
                        ;in the Status register

;good context restore
MOVF  STATUS_Temp,w
MOVWF STATUS
SWAPF W_Temp,f          ;swap in place
SWAPF W_Temp,w          ;swap with Wreg destination
```
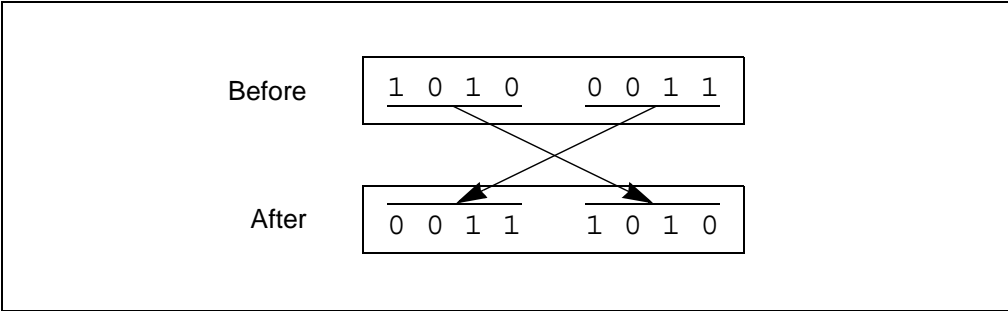
Finally, RETFIE transfers control back to the original program and sets the GIE bit, re-enabling interrupts.

FIGURE 3-10: `SWAPF` **INSTRUCTION**
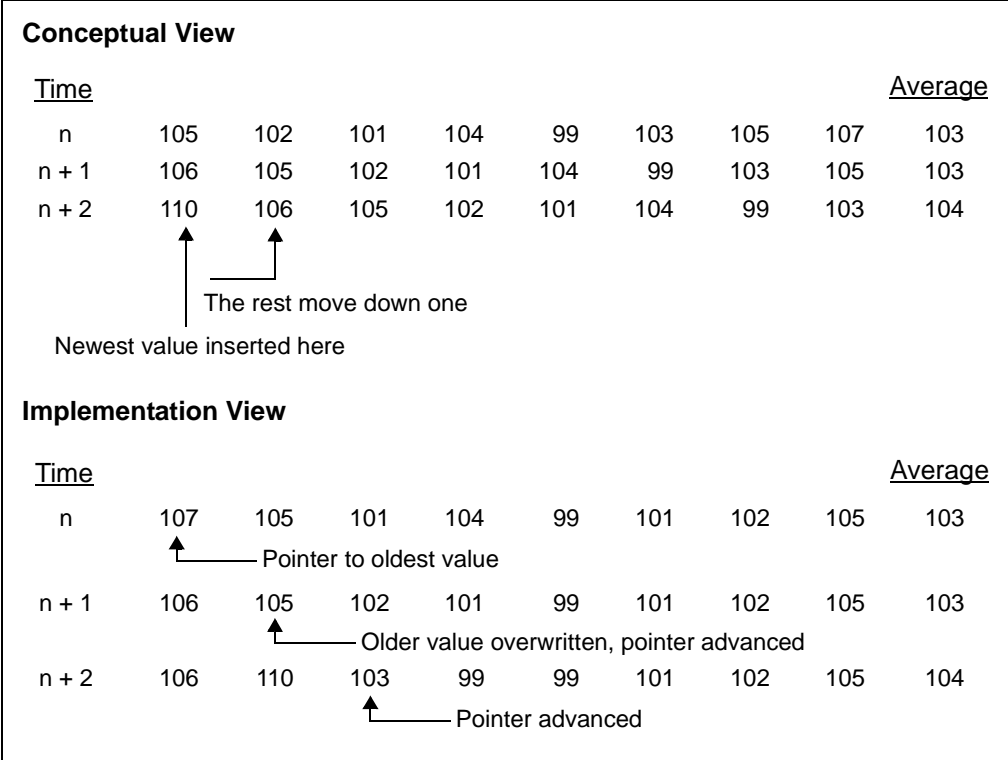

3.2.11   Lesson 11: Indirect Data Addressing

The FSR (File Select Register) allows a file register address to be specified. A subsequent read or write to the INDF (Indirect File register) refers to the file register addressed by the FSR.

This may be used to implement a moving average filter. The moving average keeps a list of the last n values and averages them together. The Filter needs two parts: A circular queue and a function to calculate the average.

FIGURE 3-11:   **MOVING AVERAGES**

Conceptual View

Time									Average
n	105	102	101	104	99	103	105	107	103
n + 1	106	105	102	101	104	99	103	105	103
n + 2	110	106	105	102	101	104	99	103	104

The rest move down one

Newest value inserted here

Implementation View

Time									Average
n	107	105	101	104	99	101	102	105	103
n + 1	106	105	102	101	99	101	102	105	103
n + 2	106	110	103	99	99	101	102	105	104

Pointer to oldest value

Older value overwritten, pointer advanced

Pointer advanced

Calculating averages in a mid-range PIC microcontroller is best accomplished by using the FSR to keep track of where the next value will be inserted. This ensures the oldest value is always overwritten with the newest and doesn't waste time moving values within the memory.

EXAMPLE 3-10:   FILE SELECT REGISTER EXAMPLE

```
;insert new value into a queue, enter with new value in
;Wreg

MOVF      temp                 ;save the latest value
MOVF      QueuePointer,w
MOVWF     FSR                  ;load FSR with the queue pointer
MOVF      temp,w
MOVWF     INDF                 ;Write the new value to the queue
INCF      QueuePointer, f   ;Advance the pointer
```

Lesson 11 adds a Moving Average Filter to the Analog-to-Digital code in Lesson 4. Turning the potentiometer changes the value read by the Analog-to-Digital converter. The averaged value is then sent to the LED display. The averaging filter only runs every 0.2 seconds to slow down the display changes and make it visible. The display appears to count from the old potentiometer position to the new position.

The filter averages the last 8 readings. Choosing a power of two for the number of samples allows division by simple rotates instead of having to use a true division routine.

Additionally, rather than summing the array every time, it's faster to keep a running sum, then subtract out the oldest value in the queue and adding in the new value.

3.2.12   Lesson 12: Look-up Table (ROM Array)

Lesson 8 introduced function calls. Lesson 12 shows how function calls and calculated modification of the Program Counter may be used to implement a look-up table (see Example 3-11).

It is sometimes useful to implement a table to convert from one value to another.

Expressed in a high-level language it might look like this:

y = function(x);

That is for every value of *x*, it returns the corresponding *y* value.

Look-up tables are a fast way to convert an input to meaningful data because the transfer function is pre-calculated and "looked up" rather than calculated on the fly.

PIC microcontrollers implement these by directly modifying the Program Counter (PC). An example might be a function that converts hexadecimal numbers to the ASCII equivalent. Each digit's individual nibble can be pulled out of the number and used as the index to the look-up table. The index advances the PC to the appropriate RETLW instruction to load WREG with the corresponding constant and returns to the calling program.

EXAMPLE 3-11:   LOOK-UP TABLE

```
;Enter with index in Wreg
LookupTable
    ADDWF       PCL,f       ;jump to
    RETLW       '0'         ;index 0
    RETLW       '1'         ;index 1
    ...
    RETLW       'F'         ;index 15
```

If the table falls across a 256 byte program memory boundary, or if somehow the look-up table is called with an out of bounds index value, it will jump to a location out of the table.

Good programming practices dictate a few additional instructions. First, since the table has only sixteen entries, make sure a number no larger than 16 is passed in. The simplest way to do this is to logically AND the contents of WREG before modifying PCL: ANDLW 0x0F. More complex error recovery schemes may be appropriate, depending on the application.

In addition, there are some nuances to be aware of should the table cross a 256 word boundary. The Program Counter is 13 bits wide, but only the lower 8 bits are represented in PCL (see Figure 3-12). The remaining 5 bits are stored in PCLATH. However, an overflow of the lower 8 bits is **not** automatically carried over into PCLATH. Instead, be sure to check for and handle that case in the code. See the PCL and PCLATH section in the PIC16F882/883/884/886/887 Data Sheet (DS41291) for more details of how PCLATH is used.

FIGURE 3-12: **PC LOADING AS DESTINATION OF INSTRUCTION**
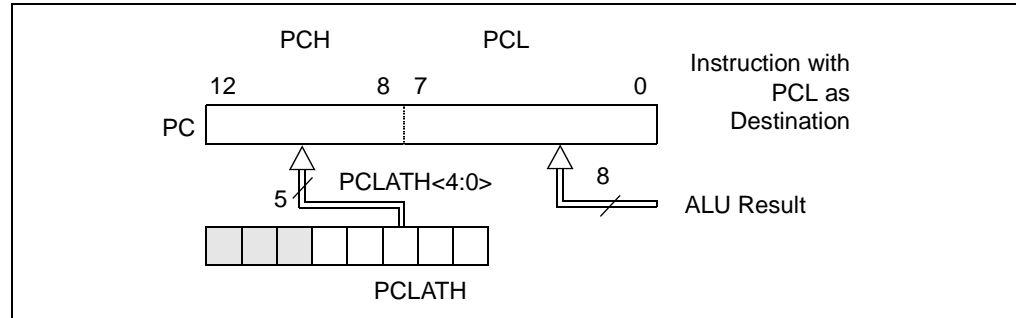
This lesson uses the look-up table to implement a binary to Gray code converter. Gray code is a binary code in which only a single bit changes from one sequence to the next. They are frequently used in encoder applications to avoid wild jumps between states. Binary encoders are typically implemented as an opaque disk with slots sensed by light sensors. Due to different threshold levels on different bits, bits may change at slightly differently times yielding momentary invalid results. Gray code prevents this because only one bit changes from one sequence to the next. The current code is correct until it transitions to the next.

The algorithm to convert between binary and Gray code is fairly complex. For a small number of bits, the table look-up is smaller and faster.

This lesson takes the Analog-to-Digital value upper nibble and converts it to Gray code displayed on the first four LEDs. The code changes one bit at a time as the potentiometer rotates across its range (seeExample 3-12).

Gray Code Converter

Decimal	Binary
0	0000
1	0001
2	0011
3	0010
4	0110
5	0111
6	0101
7	0100
8	1100
9	1101
10	1111
11	1110
12	1010
13	1011
14	1001
15	1000

EXAMPLE 3-12:   CONVERT BINARY TO GRAY CODE

```
BinaryToGrayCode
  ANDLW    0x0F                ;mask off invalid entries
  MOVWF    temp
  MOVLW    high TableStart     ;get high order part of the
                               ;beginning of the table
  MOVWF    PCLATH
  MOVLW    low TableStart      ;load starting address of table
  ADDWF    temp,w              ;add offset
  BTFSC    STATUS,C            ;did it overflow?
  INCF     PCLATH,f            ;yes: increment PCLATH
  MOVWF    PCL                 ;modify PCL
TableStart
  RETLW    b'0000'             ;0
  RETLW    b'0001'             ;1
  RETLW    b'0011'             ;2
  RETLW    b'0010'             ;3
  RETLW    b'0110'             ;4
  RETLW    b'0111'             ;5
  RETLW    b'0101'             ;6
  RETLW    b'0100'             ;7
  RETLW    b'1100'             ;8
  RETLW    b'1101'             ;9
  RETLW    b'1111'             ;10
  RETLW    b'1110'             ;11
  RETLW    b'1010'             ;12
  RETLW    b'1011'             ;13
  RETLW    b'1001'             ;14
  RETLW    b'1000'             ;15
```
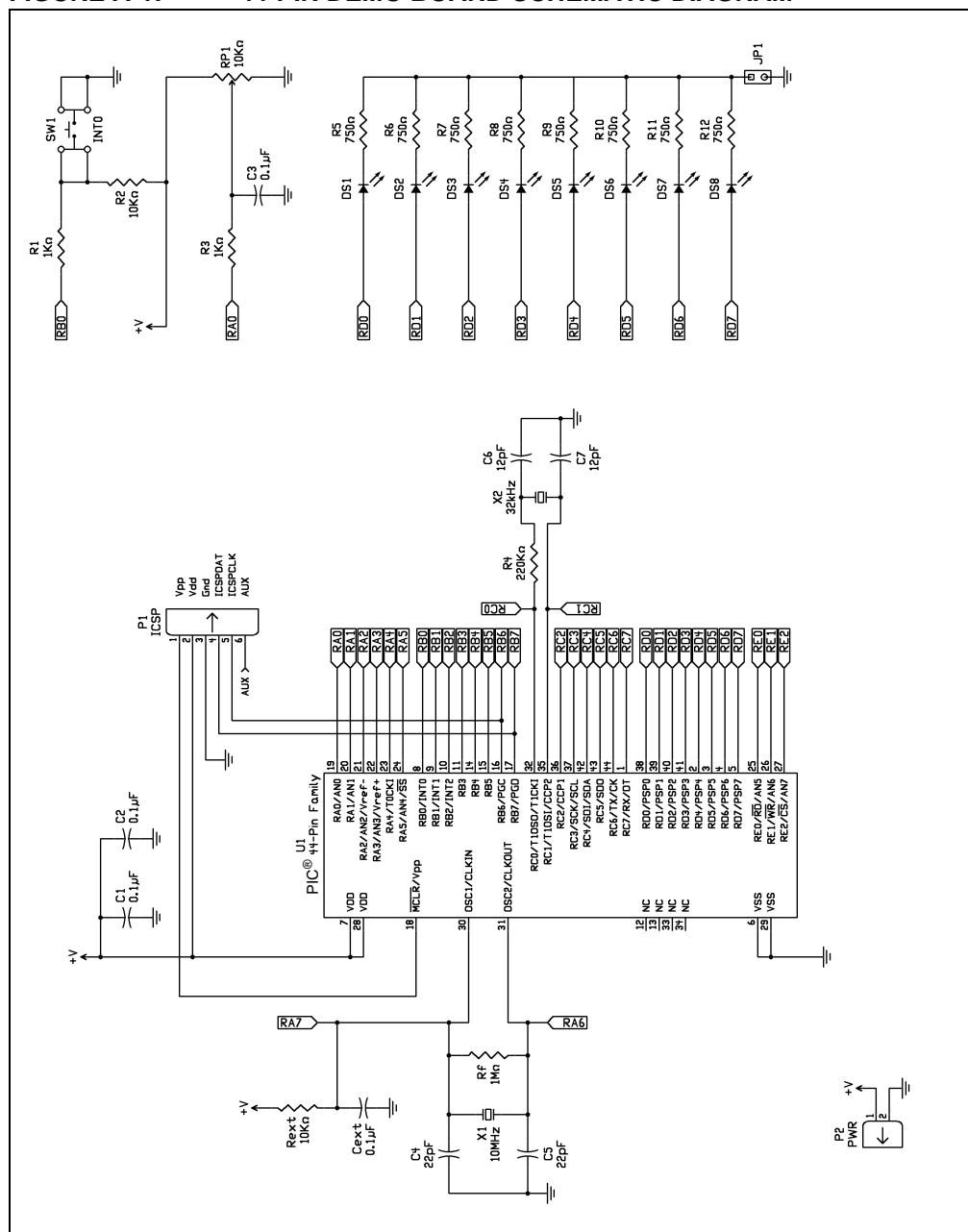
Appendix A. Hardware Schematics

A.1 INTRODUCTION

This appendix contains the 44-Pin Demo Board schematic, PCB layout and Bill of Materials.

FIGURE A-1: **44-PIN DEMO BOARD SCHEMATIC DIAGRAM**

44-Pin Demo Board User's Guide

FIGURE A-2:   **44-PIN DEMO BOARD SILKSCREEN**
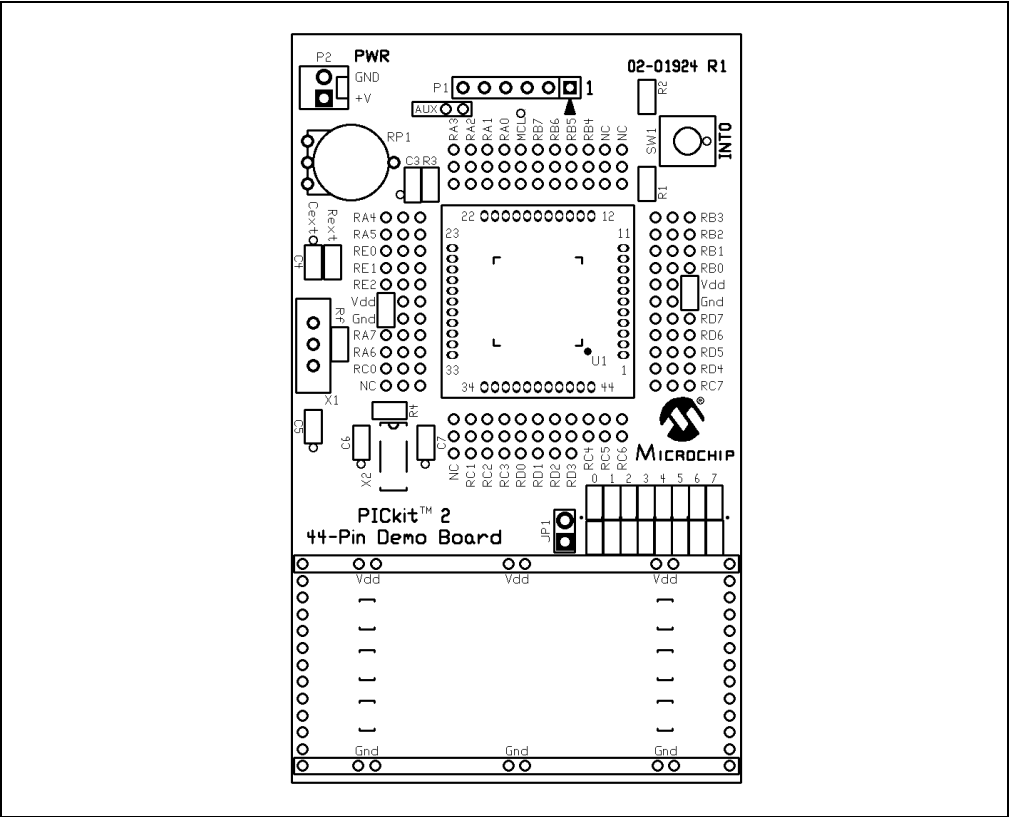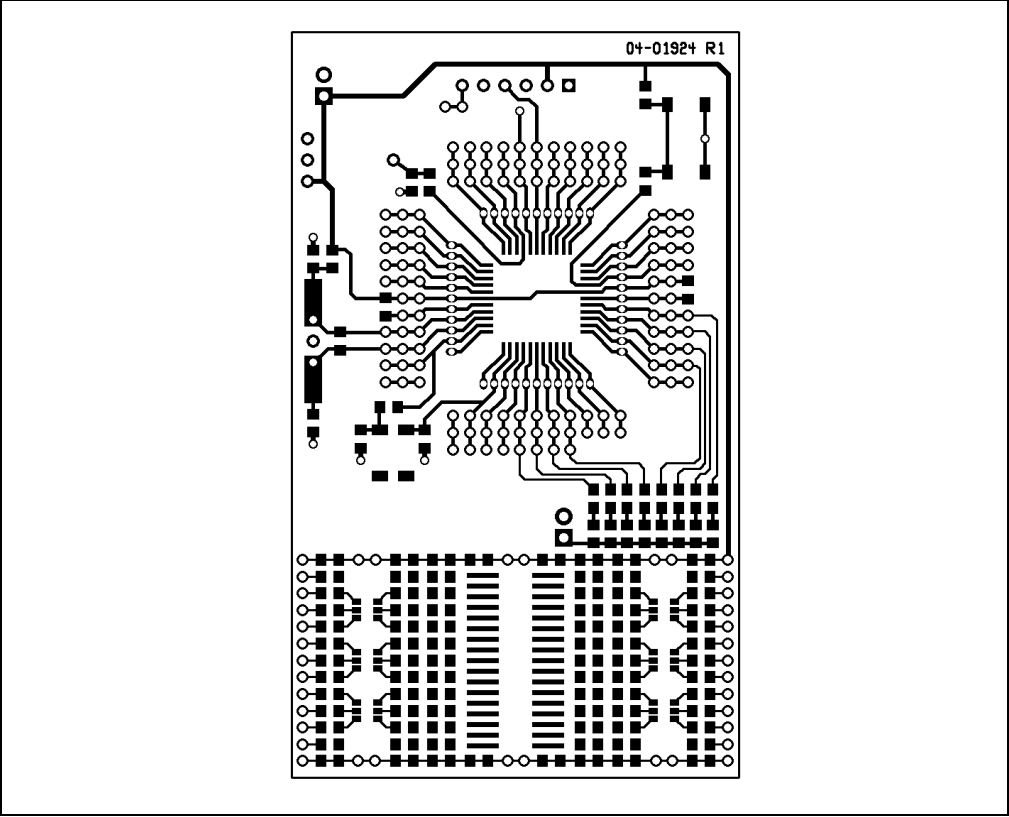

FIGURE A-3:   **44-PIN DEMO BOARD TOP COPPER**
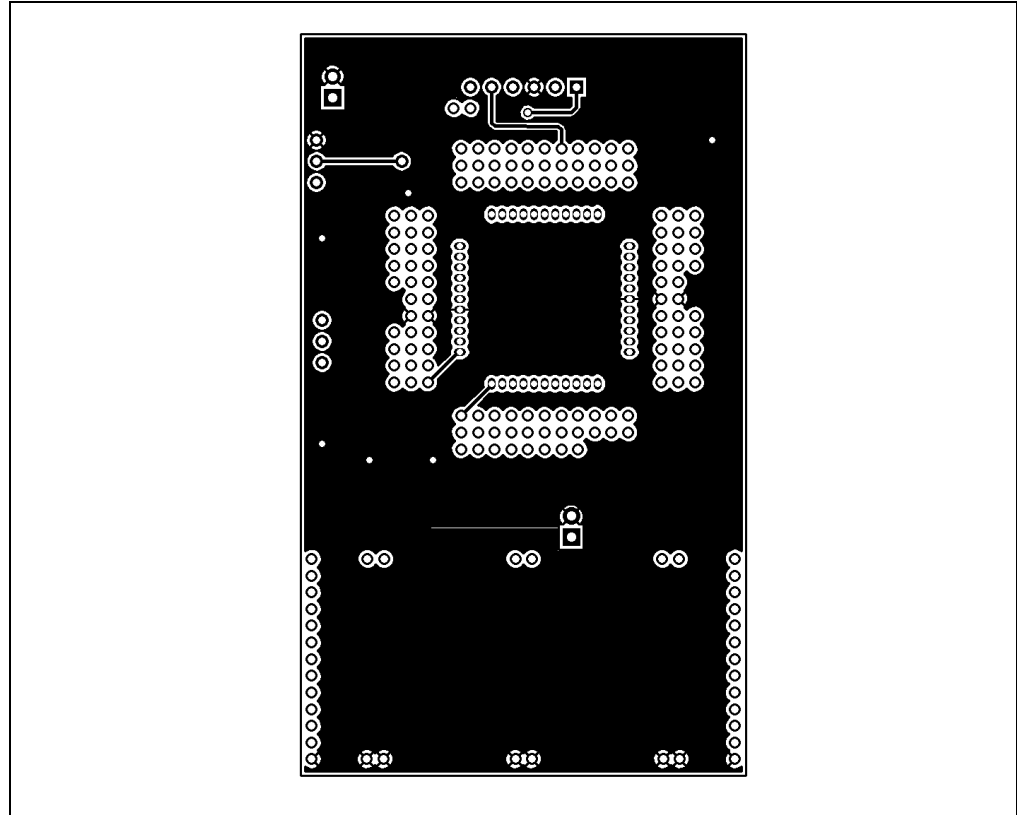
FIGURE A-4:   **44-PIN DEMO BOARD BOTTOM COPPER**

TABLE A-1:   **44-PIN DEMO BOARD BILL OF MATERIALS**

Bill of Materials		
Designation	**Qty**	**Description**
C1, C2, C3	3	Capacitor, Ceramic, 0805 SMT, 0.1 µF, 16V, 5%, X7R
R5-R12	8	Resistor, 0805 SMT, 750Ω, 5%, 1/8W
R1, R3	2	Resistor, 0805 SMT, 1 kΩ, 5%, 1/8W
R2	1	Resistor, 0805 SMT, 10 kΩ, 5%, 1/8W
RP1	1	Potentiometer 10 kΩ, thumbwheel
DS1-DS8	8	LED, 0805 SMT, Red Clear
SW1	1	Switch, push button, momentary
U1 – Microcontroller	1	44-pin PIC® MCU
P1	1	Connector, header, right-angle, 6-pin, 0.100" spacing, 0.025" square
JP1	1	Connector, header, 2-pin, 0.100" spacing, 0.025" square
Rubber feet	4	Bumpon square, 0.40 x 0.10, black

WORLDWIDE SALES AND SERVICE

AMERICAS

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

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

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

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

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

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

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

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

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

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

ASIA/PACIFIC

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

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

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

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

China - Fuzhou
Tel: 86-591-8750-3506
Fax: 86-591-8750-3521

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

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

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

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

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

China - Shunde
Tel: 86-757-2839-5507
Fax: 86-757-2839-5571

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

China - Xian
Tel: 86-29-8833-7250
Fax: 86-29-8833-7256

ASIA/PACIFIC

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

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

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

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

Korea - Gumi
Tel: 82-54-473-4301
Fax: 82-54-473-4302

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

Malaysia - Penang
Tel: 60-4-646-8870
Fax: 60-4-646-5086

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

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

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

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

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

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

EUROPE

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

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

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

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

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

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

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

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

12/08/06

Mouser Electronics

Authorized Distributor

Click to View Pricing, Inventory, Delivery & Lifecycle Information:

[Microchip](#):
  [DM164120-2](#)