AN1354

# Implementing an LCD Using the PIC16F1947 Microcontroller

Authors: *John Mouton*
*Microchip Technology Inc.*

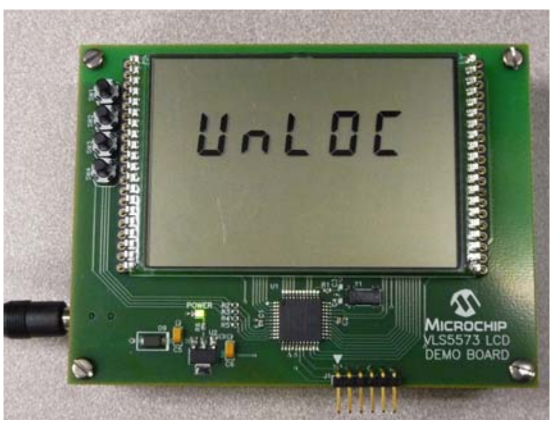
## INTRODUCTION

This application note reviews the process and procedure of implementing a segmented LCD using the PIC16F1947 microcontroller (MCU) in an example application. The example application will be an electronic combination lock. As this application is implemented, the configuration and low-power options associated with the PIC16F1947 LCD module will be discussed.

## GETTING STARTED

For the purposes of this application note, the Varitronix VLS5573 demo board has been chosen; a custom 8-digit LCD made to demonstrate Microchip devices with integrated LCD controllers. This display has numerical digits, a bar graph, and a variety of symbols suitable for a clock, thermometer, or a voltmeter application. See Figure 3 for the LCD segments and Figure 1 for a photo of the application example.

**FIGURE 1: VLS5573 LCD DISPLAY**




### How Much LCD Can the Microcontroller Handle?

When you have decided which segmented LCD display to use for your application and have acquired the LCD data sheet (from the manufacturer), you can begin setting up the MCU to run the LCD via the PIC16F1947 data sheet (DS41414). This application note will take you through the LCD module configuration and code development step by step.

Start by verifying that the microcontroller you have chosen has a sufficient number of LCD segment pins to cover all of the LCD segments used on the particular LCD you have chosen. Many LCDs will have multiplexed segments to reduce the number of MCU pins required. A typical multiplexed LCD will have 2, 3 or 4 common pins. The PIC16F1947 used for this application note can be multiplexed up to 4 times the number of segment pins and uses up to four common pins (see Figure 2).

**FIGURE 2: LCD COMMON PIN CONFIGURATION**

| |
|---|
| Static (1 common) |
| 1/2 multiplex (2 commons) |
| 1/3 multiplex (3 commons) |
| 1/4 multiplex (4 commons) |

This means that, for a given device, it will have a certain number of output pins that can be used to control an LCD display and, with the four different multiplexed common pin configurations, you basically are able to control four times as many LCD segments.

For example, with a 64-pin device with 46 output pins available to control an LCD display X 4 (multiplex configurations), you can control up to 184 LCD segments.

Therefore, with the PIC16F1947 being a 64-pin device, it can control up to 184 LCD segments, which is more than enough for this application. Table 1 shows all of the LCD segment character ID's as shown in Figure 3, with respect to the COMs and display pins.

**TABLE 1: VARITRONIX VLS5573 LCD PINOUT**

| PIN | COM1 | COM2 | COM3 | COM4 |
|---|---|---|---|---|
| 1 | N.C. | N.C. | N.C. | N.C. |
| 2 | COM1 | – – – | – – – | – – – |
| 3 | S12 | – – – | – – – | – – – |
| 4 | S11 | – – – | – – – | – – – |
| 5 | 7D | 7F | 7A | 7E |
| 6 | DP2 | 7G | 7B | 7C |
| 7 | 8D | 8F | 8A | 8E |
| 8 | S13 | 8G | 8B | 8C |
| 9 | X23 | X25 | X26 | X24 |
| 10 | 5B | 5C | – – – | 5G |
| 11 | 5A | 5E | 5G | 5F |
| 12 | 4B | 4C | DP1 | 4G |
| 13 | 4A | 4E | 4D | 4F |
| 14 | 3B | 3C | – – – | 3G |
| 15 | – – – | X22 | X21 | – – – |
| 16 | X17 | X19 | X20 | X18 |
| 17 | X16 | X14 | X13 | X15 |
| 18 | X9 | X11 | X12 | X10 |
| 19 | – – – | COM2 | – – – | – – – |
| 20 | N.C. | N.C. | N.C. | N.C. |
| 21 | N.C. | N.C. | N.C. | N.C. |
| 22 | – – – | – – – | COM3 | – – – |
| 23 | X8 | X6 | X5 | X7 |
| 24 | X1 | X3 | X4 | X2 |
| 25 | – – – | F2 | F1 | F3 |
| 26 | 1A | 1E | F4 | 1F |
| 27 | 1B | 1C | 1D | 1G |
| 28 | 2A | 2E | 2D | 2F |
| 29 | 2B | 2C | COL | 2G |
| 30 | 3A | 3E | 3D | 3F |
| 31 | S9 | 6G | 6B | 6C |
| 32 | 6D | 6F | 6A | 6E |
| 33 | X31 | X33 | X34 | X32 |
| 34 | – – – | X27 | X35 | X30 |
| 35 | – – – | X28 | X36 | X29 |
| 36 | S1 | S3 | S4 | S2 |
| 37 | S6 | – – – | – – – | S7 |
| 38 | S5 | – – – | – – – | S8 |
| 39 | | – – – | – – – | COM4 |
| 40 | N.C. | N.C. | N.C. | N.C. |

## Segment Mapping

Let us look at the LCD data sheet for the LCD used in this application, the Varitronix VLS5573. Every LCD data sheet should have at least two sections; a segment layout (which includes the entire LCD segment layout and the digit segment layout), see Figure 3 and Figure 4, and a LCD pinout table (Table 1). From these two sections you will get most of the information needed to drive the LCD from the microcontroller.

**FIGURE 3: LCD SEGMENT LAYOUT**

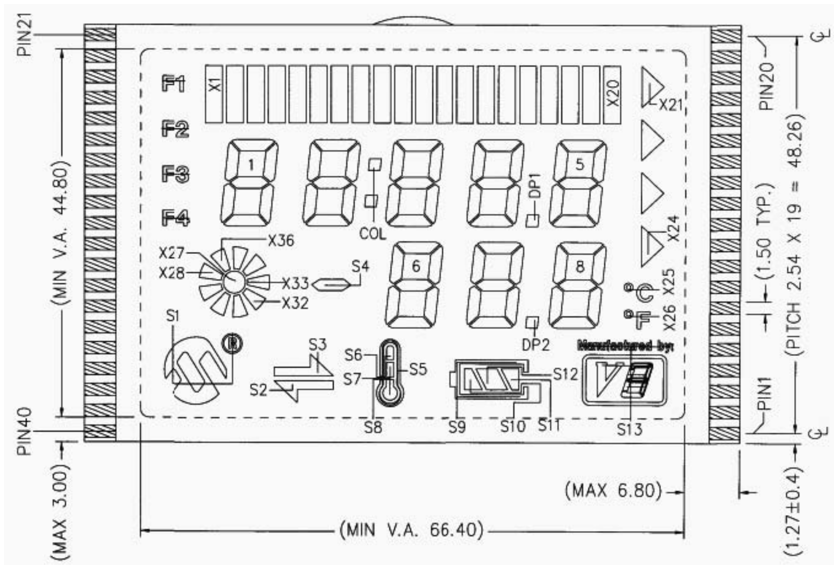
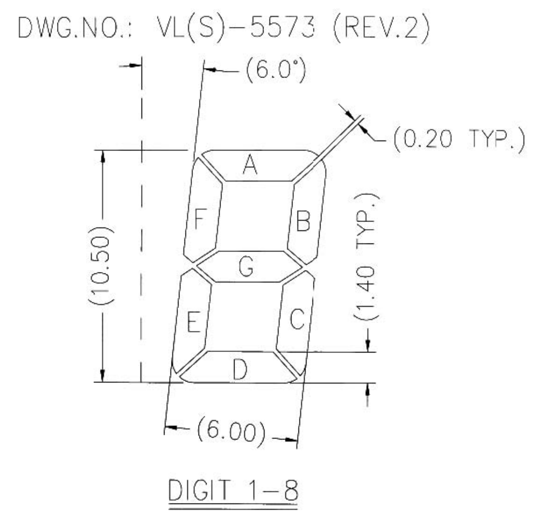



**FIGURE 4: DIGIT SEGMENT LAYOUT**




Now, take a look at which parts of the LCD display are going to be used and which LCD segments on the microcontroller you are going to drive them with. You will map this out by using the LCD segment mapping worksheet as found in the PIC16F1947 data sheet (Table 2). To fill out the LCD segment mapping worksheet, you will need the LCD segment layout (Figure 3), the digit segment layout (Figure 4), and the LCD pinout table (Table 1) from the LCD display data sheet.

### TABLE 2: LCD SEGMENT MAPPING WORKSHEET

| LCD Function | COM0 | | COM1 | | COM2 | | COM3 | |
|---|---|---|---|---|---|---|---|---|
| | LCDDATAx Address | LCD Segment | LCDDATAx Address | LCD Segment | LCDDATAx Address | LCD Segment | LCDDATAx Address | LCD Segment |
| SEG0 | LCDDATA0, 0 | | LCDDATA3, 0 | | LCDDATA6, 0 | | LCDDATA9, 0 | |
| SEG1 | LCDDATA0, 1 | | LCDDATA3, 1 | | LCDDATA6, 1 | | LCDDATAA9, 1 | |
| SEG2 | LCDDATA0, 2 | | LCDDATA3, 2 | | LCDDATA6, 2 | | LCDDATAA9, 2 | |
| SEG3 | LCDDATA0, 3 | | LCDDATA3, 3 | | LCDDATA6, 3 | | LCDDATAA9, 3 | |
| SEG4 | LCDDATA0, 4 | | LCDDATA3, 4 | | LCDDATA6, 4 | | LCDDATAA9, 4 | |
| SEG5 | LCDDATA0, 5 | | LCDDATA3, 5 | | LCDDATA6, 5 | | LCDDATAA9, 5 | |
| SEG6 | LCDDATA0, 6 | | LCDDATA3, 6 | | LCDDATA6, 6 | | LCDDATAA9, 6 | |
| SEG7 | LCDDATA0, 7 | | LCDDATA3, 7 | | LCDDATA6, 7 | | LCDDATAA9, 7 | |
| SEG8 | LCDDATA1, 0 | | LCDDATA4, 0 | | LCDDATA7, 0 | | LCDDATA10, 0 | |
| SEG9 | LCDDATA1, 1 | | LCDDATA4, 1 | | LCDDATA7, 1 | | LCDDATA10, 1 | |
| SEG10 | LCDDATA1, 2 | | LCDDATA4, 2 | | LCDDATA7, 2 | | LCDDATA10, 2 | |
| SEG11 | LCDDATA1, 3 | | LCDDATA4, 3 | | LCDDATA7, 3 | | LCDDATA10, 3 | |
| SEG12 | LCDDATA1, 4 | | LCDDATA4, 4 | | LCDDATA7, 4 | | LCDDATA10, 4 | |
| SEG13 | LCDDATA1, 5 | | LCDDATA4, 5 | | LCDDATA7, 5 | | LCDDATA10, 5 | |
| SEG14 | LCDDATA1, 6 | | LCDDATA4, 6 | | LCDDATA7, 6 | | LCDDATA10, 6 | |
| SEG15 | LCDDATA1, 7 | | LCDDATA4, 7 | | LCDDATA7, 7 | | LCDDATA10, 7 | |
| SEG16 | LCDDATA2, 0 | | LCDDATA5, 0 | | LCDDATA8, 0 | | LCDDATA11, 0 | |
| SEG17 | LCDDATA2, 1 | | LCDDATA5, 1 | | LCDDATA8, 1 | | LCDDATA11, 1 | |
| SEG18 | LCDDATA2, 2 | | LCDDATA5, 2 | | LCDDATA8, 2 | | LCDDATA11, 2 | |
| SEG19 | LCDDATA2, 3 | | LCDDATA5, 3 | | LCDDATA8, 3 | | LCDDATA11, 3 | |
| SEG20 | LCDDATA2, 4 | | LCDDATA5, 4 | | LCDDATA8, 4 | | LCDDATA11, 4 | |
| SEG21 | LCDDATA2, 5 | | LCDDATA5, 5 | | LCDDATA8, 5 | | LCDDATA11, 5 | |
| SEG22 | LCDDATA2, 6 | | LCDDATA5, 6 | | LCDDATA8, 6 | | LCDDATA11, 6 | |
| SEG23 | LCDDATA2, 7 | | LCDDATA5, 7 | | LCDDATA8, 7 | | LCDDATA11, 7 | |
| SEG24 | LCDDATA12, 0 | | LCDDATA15, 0 | | LCDDATA18, 0 | | LCDDATA21, 0 | |
| SEG25 | LCDDATA12, 1 | | LCDDATA15, 1 | | LCDDATA18, 1 | | LCDDATA21, 1 | |
| SEG26 | LCDDATA12, 2 | | LCDDATA15, 2 | | LCDDATA18, 2 | | LCDDATA21, 2 | |
| SEG27 | LCDDATA12, 3 | | LCDDATA15, 3 | | LCDDATA18, 3 | | LCDDATA21, 3 | |
| SEG28 | LCDDATA12, 4 | | LCDDATA15, 4 | | LCDDATA18, 4 | | LCDDATA21, 4 | |
| SEG29 | LCDDATA12, 5 | | LCDDATA15, 5 | | LCDDATA18, 5 | | LCDDATA21, 5 | |
| SEG30 | LCDDATA12, 6 | | LCDDATA15, 6 | | LCDDATA18, 6 | | LCDDATA21, 6 | |
| SEG31 | LCDDATA12, 7 | | LCDDATA15, 7 | | LCDDATA18, 7 | | LCDDATA21, 7 | |
| SEG32 | LCDDATA13, 0 | | LCDDATA16, 0 | | LCDDATA19, 0 | | LCDDATA22, 0 | |
| SEG33 | LCDDATA13, 1 | | LCDDATA16, 1 | | LCDDATA19, 1 | | LCDDATA22, 1 | |
| SEG34 | LCDDATA13, 2 | | LCDDATA16, 2 | | LCDDATA19, 2 | | LCDDATA22, 2 | |
| SEG35 | LCDDATA13, 3 | | LCDDATA16, 3 | | LCDDATA19, 3 | | LCDDATA22, 3 | |
| SEG36 | LCDDATA13, 4 | | LCDDATA16, 4 | | LCDDATA19, 4 | | LCDDATA22, 4 | |
| SEG37 | LCDDATA13, 5 | | LCDDATA16, 5 | | LCDDATA19, 5 | | LCDDATA22, 5 | |
| SEG38 | LCDDATA13, 6 | | LCDDATA16, 6 | | LCDDATA19, 6 | | LCDDATA22, 6 | |
| SEG39 | LCDDATA13, 7 | | LCDDATA16, 7 | | LCDDATA19, 7 | | LCDDATA22, 7 | |
| SEG40 | LCDDATA14, 0 | | LCDDATA17, 0 | | LCDDATA20, 0 | | LCDDATA23, 0 | |
| SEG41 | LCDDATA14, 1 | | LCDDATA17, 1 | | LCDDATA20, 1 | | LCDDATA23, 1 | |
| SEG42 | LCDDATA14, 2 | | LCDDATA17, 2 | | LCDDATA20, 2 | | LCDDATA23, 2 | |
| SEG43 | LCDDATA14, 3 | | LCDDATA17, 3 | | LCDDATA20, 3 | | LCDDATA23, 3 | |
| SEG44 | LCDDATA14, 4 | | LCDDATA17, 4 | | LCDDATA20, 4 | | LCDDATA23, 4 | |
| SEG45 | LCDDATA14, 5 | | LCDDATA17, 5 | | LCDDATA20, 5 | | LCDDATA23, 5 | |

The LCD segment layout shows all of the LCD's display features, location on the display and their segment IDs, as shown in Figure 3. This layout also shows how many digits can be displayed and each digit's individual segment layout, as shown in Figure 4. Table 1 shows all of the LCD display segment IDs, which commons they are on, and which LCD pins they are on. This will allow you to physically connect the specific microcontroller segment pins to the correct LCD display segment pin. The LCD segment mapping worksheet, Table 2, will help you keep track of which microcontroller LCD segment will drive a particular LCD display segment ID of your choosing and shows you which LCDDATA register bit will control the particular segment. See Table 3 for a completed LCD segment map for the electronic combination lock application.

**TABLE 3: COMPLETED LCD SEGMENT MAPPING WORKSHEET**

| LCD Function | COM0 | | COM1 | | COM2 | | COM3 | |
|---|---|---|---|---|---|---|---|---|
| | LCDDATAx Address | LCD Segment | LCDDATAx Address | LCD Segment | LCDDATAx Address | LCD Segment | LCDDATAx Address | LCD Segment |
| SEG0 | LCDDATA0, 0 | | LCDDATA3, 0 | | LCDDATA6, 0 | | LCDDATA9, 0 | |
| SEG1 | LCDDATA0, 1 | 1A | LCDDATA3, 1 | 1E | LCDDATA6, 1 | F4 | LCDDATAA9, 1 | 1F |
| SEG2 | LCDDATA0, 2 | — | LCDDATA3, 2 | F2 | LCDDATA6, 2 | F1 | LCDDATAA9, 2 | F3 |
| SEG3 | LCDDATA0, 3 | 1B | LCDDATA3, 3 | 1C | LCDDATA6, 3 | 1D | LCDDATAA9, 3 | 1G |
| SEG4 | LCDDATA0, 4 | 5B | LCDDATA3, 4 | 5C | LCDDATA6, 4 | — | LCDDATAA9, 4 | 5G |
| SEG5 | LCDDATA0, 5 | 5A | LCDDATA3, 5 | 5E | LCDDATA6, 5 | 5D | LCDDATAA9, 5 | 5F |
| SEG6 | LCDDATA0, 6 | 2A | LCDDATA3, 6 | 2E | LCDDATA6, 6 | 2D | LCDDATAA9, 6 | 2F |
| SEG7 | LCDDATA0, 7 | | LCDDATA3, 7 | | LCDDATA6, 7 | | LCDDATAA9, 7 | |
| SEG8 | LCDDATA1, 0 | | LCDDATA4, 0 | | LCDDATA7, 0 | | LCDDATA10, 0 | |
| SEG9 | LCDDATA1, 1 | | LCDDATA4, 1 | | LCDDATA7, 1 | | LCDDATA10, 1 | |
| SEG10 | LCDDATA1, 2 | | LCDDATA4, 2 | | LCDDATA7, 2 | | LCDDATA10, 2 | |
| SEG11 | LCDDATA1, 3 | 3A | LCDDATA4, 3 | 3E | LCDDATA7, 3 | 3D | LCDDATA10, 3 | 3F |
| SEG12 | LCDDATA1, 4 | | LCDDATA4, 4 | | LCDDATA7, 4 | | LCDDATA10, 4 | |
| SEG13 | LCDDATA1, 5 | | LCDDATA4, 5 | | LCDDATA7, 5 | | LCDDATA10, 5 | |
| SEG14 | LCDDATA1, 6 | | LCDDATA4, 6 | | LCDDATA7, 6 | | LCDDATA10, 6 | |
| SEG15 | LCDDATA1, 7 | | LCDDATA4, 7 | | LCDDATA7, 7 | | LCDDATA10, 7 | |
| SEG16 | LCDDATA2, 0 | 2B | LCDDATA5, 0 | 2C | LCDDATA8, 0 | — | LCDDATA11, 0 | 2G |
| SEG17 | LCDDATA2, 1 | | LCDDATA5, 1 | | LCDDATA8, 1 | | LCDDATA11, 1 | |
| SEG18 | LCDDATA2, 2 | S1 | LCDDATA5, 2 | | LCDDATA8, 2 | | LCDDATA11, 2 | |
| SEG19 | LCDDATA2, 3 | | LCDDATA5, 3 | | LCDDATA8, 3 | | LCDDATA11, 3 | |
| SEG20 | LCDDATA2, 4 | | LCDDATA5, 4 | | LCDDATA8, 4 | | LCDDATA11, 4 | |
| SEG21 | LCDDATA2, 5 | 4B | LCDDATA5, 5 | 4C | LCDDATA8, 5 | — | LCDDATA11, 5 | 4G |
| SEG22 | LCDDATA2, 6 | 4A | LCDDATA5, 6 | 4E | LCDDATA8, 6 | 4D | LCDDATA11, 6 | 4F |
| SEG23 | LCDDATA2, 7 | 3B | LCDDATA5, 7 | 3C | LCDDATA8, 7 | — | LCDDATA11, 7 | 3G |

**Note:** The LCD display segment IDs can be driven by any microcontroller LCD segment function of your choosing.

For example:

| Microcontroller LCD function | LCD Segments |
|---|---|
| SEG1 | 1A,1E,F4,1F |
| or SEG13 | 1A,1E,F4,1F |

**Note:** However, the choice of LCD segment pins will likely be determined by the PCB layout of your application. Your source code will include functions that map to the appropriate bits in the LCDDATA registers.

## LCD Module Configuration

Now, take a look at the registers used to set up the LCD module in the PIC16F1947 device you are using. The LCD module contains the following registers:

- LCD Control register (LCDCON)
- LCD Phase register (LCDPS)
- LCD Reference Ladder register (LCDRL)
- LCD Contrast Control register (LCDCST)
- LCD Reference Voltage Control register (LCDREF)
- LCD Segment Enable registers (LCDSEn)
- LCD Data registers (LCDDATAn)

See Example 1 for the LCD module register settings as used in this application example.

**EXAMPLE 1: INITIALIZATION OF THE LCD MODULE**

```
void lcd_init(void)
{
      LCDSE0 = 0xFE;                    // enable first group of LCD segment outputs
      LCDSE1 = 0x8F;                    // enable second group of LCD segments
      LCDSE2 = 0xFF;                    // enable third group of LCD segments
      LCDSE3 = 0x00;
      LCDSE4 = 0x00;
      LCDSE5 = 0x00;

      LCDDATA0  = 0;                    // clear LCD segment registers
      LCDDATA1  = 0;
      LCDDATA2  = 0;
      LCDDATA3  = 0;
      LCDDATA4  = 0;
      LCDDATA5  = 0;
      LCDDATA6  = 0;
      LCDDATA7  = 0;
      LCDDATA8  = 0;
      LCDDATA9  = 0;
      LCDDATA10 = 0;
      LCDDATA11 = 0;
      LCDDATA12  = 0;                   // clear LCD segment registers
      LCDDATA13  = 0;
      LCDDATA14  = 0;
      LCDDATA15  = 0;
      LCDDATA16  = 0;
      LCDDATA17  = 0;
      LCDDATA18  = 0;
      LCDDATA19  = 0;
      LCDDATA20  = 0;
      LCDDATA21  = 0;
      LCDDATA22 = 0;
      LCDDATA23 = 0;
                             // WAVEFORM TYPE A, LCD MODULE IS ACTIVE
      LCDPS  = 0x20;         // PRESCALER IS 1:1,  BIAS IS 0 (CAN BE STATIC OR 1/3)
                             // LCD MODULE IS ON, DRIVER MODULE IS ENABLED DURING SLEEP
      LCDCON = 0x8B;         // NO WRITE FAIL ERROR, VLDC PINS ARE ENABLED, MULTIPLEX 1/4 BIAS 1/3

      LCDREF = 0x80;

      LCDCST = 0x00;

      LCDRL  = 0xF0;

}
```

## LCD CONTROL REGISTER (LCDCON)

For this application, this register is used to:

- turn the LCD module on,
- have the LCD module off while the microcontroller is in Sleep mode,
- clock source the LCD using the Timer1 oscillator T1OSC and,
- have the LCD segments configured for 1/4 multiplex to give us the maximum number of LCD segments for this device (184 segments) at 1/3 bias.

### Clock Sources

The LCD module has 3 possible clock sources: FOSC/256, T1OSC and LFINTOSC. The first clock source is the system clock (FOSC) divided by 256. This divider ratio is chosen to provide about 1 kHz output when the system clock is 8 MHz. This source is commonly used unless the LCD needs to run while the processor is in Sleep mode, then the second or third clock sources should be used. The second clock source is the T1OSC. This clock source will also give about 1 kHz when a 32 kHz crystal is used with the Timer1 oscillator. The third clock source is the 31 kHz Low-Frequency Internal Oscillator (LFINTOSC), which also provides approximately 1 kHz output.

Which clock source is better for low power? That depends on the microcontroller and the LCD module being used. For the purpose of this application, using the PIC16F1947 in Sleep mode, T1OSC or LFINTOSC can be used. However, if you look in the electrical specification section of the data sheet (DS41414) you will see that using the T1OSC clock will consume the least amount of current of the two. When the microcontroller is not in Sleep mode and you are using Watchdog Timer (WDT), the LFINTOSC has the lower current consumption. But, if you are already using the T1OSC for Sleep mode, use it when you are not in Sleep mode. The difference in current consumption is minimal and it does not make sense to change it.

### Drive Modes

LCD panels come in many flavors depending on the application and the operating environment. LCDs can be classified in two ways. LCDs come in static (or direct) drive or multiplex drive variations. Static drive displays use only one common or backplane signal. Every pixel has its own segment and frontplane line. The common line acts as an “activation” signal, preparing all the pixels that it touches to be turned on by respective segment lines. The segment lines act as a “selector” signal, specifying whether a pixel is turned ON or OFF. When the common line is not activated, the segment lines have no effect on the pixel state.

Frequencies for static drive displays are typically between 30 Hz and 100 Hz, depending on display size and design. Displays can operate at higher frequencies, but this increases power consumption. LCDs mimic a capacitive load, which reduces the load impedance as frequency increases. However, operation below 30 Hz usually results in visible flicker of the segments. LCDs can be overdriven by a combination of voltage, frequency and lower contrast at higher frequencies, which result in cross talk or “ghosting”. Ghosting is the appearance or partial activation of an “off” segment. This condition occurs when high drive voltage and frequency are applied. Because the current is directly proportional to the frequency, the voltage-frequency product must not be exceeded. It is also important to connect all unused segments to the backplane, and not allow them to float.

The main advantage of static drive is that it is simple to implement. You only have to worry about which segment line to turn on and off, while activating the common signal all the time.

Another advantage is that voltage levels can go from rail to rail and does not require multiple intermediate levels, providing more contrast control. The disadvantage is that it requires more pins. Every pixel must have a segment line tied to it, and segment lines are connected to pins on the microcontroller.

Multiplex drive panels reduce the overall amount of interconnections between the LCD and the driver. Basically, multiplex panels have more than one backplane or common, as mentioned earlier in this application note. A multiplex LCD driver produces an amplitude-varying, time synchronized waveform for both the segment and backplanes. These waveforms allow access to one pixel on each of the backplanes. This significantly increases the complexity of the driver. The number of backplanes or common a panel has is referred to as the multiplexing, duty cycle, duty, or MUX ratios.

### Duty or MUX Ratio

Duty cycle, or duty, or MUX ratios indicates the number of commons, normally defined as the inverse of the number of commons/backplanes. For example, if the display has four commons, then the duty ratio is 1/4. The process of refreshing an LCD with n number of backplanes (commons) and m number of frontplanes (segments) is similar to the matrixed keyboard operation. The driver selects one backplane (corresponding to a column on a keyboard) and drives the appropriate voltage levels to all frontplanes associated with that backplane (corresponding to keyboard rows). The remaining backplanes are driven to an unselected voltage level. This process is then repeated for all backplane electrodes of the display.

For more details on the LCDCON register and its bit descriptions and functions, see Register 26-1 of the device data sheet (DS41414).

## LCD PHASE REGISTER (LCDPS)

For this application, this register is used to:

- set the waveform type to type-A,
- set the voltage Bias mode to 1/3 bias,
- set the LCD driver module to active status,
- not allow the LCD data registers to be written to during a specific period of time and,
- set the LCD clock source prescaler to a 1-to-1 ratio.

## WAVEFORM TYPE

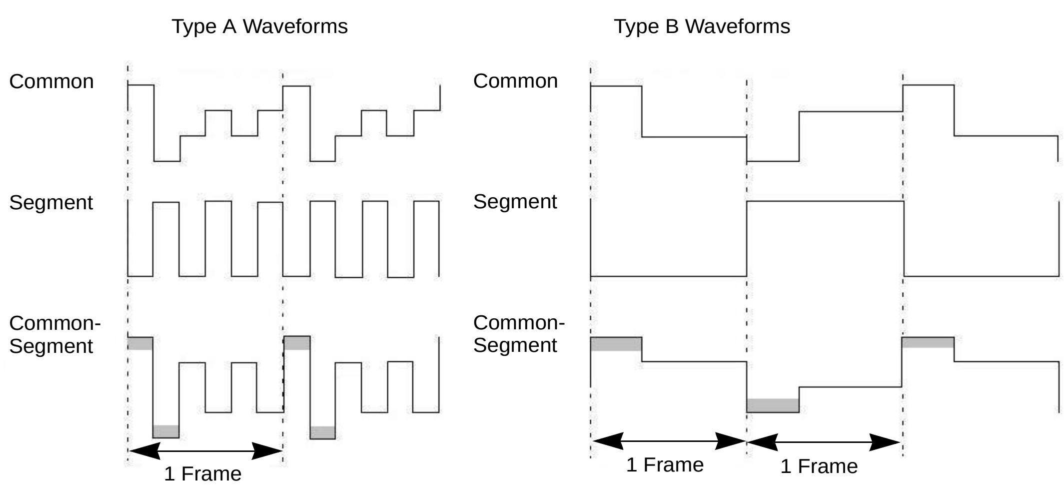
Multiplexed LCDs can be driven by two types of waveforms generated by the LCD module. These are called Type A and Type B in LCD specifications and data sheets (see Figure 5). Given that an AC signal with an average DC bias of 0 volts is required to drive the LCD, type A waveforms take a single frame to maintain 0 volts DC. Type B waveforms take two frames to maintain 0 volts DC. The main difference between the two types of waveforms is in the frequencies of voltages applied to the LCD pixels.

From Figure 5, it is clear that type A waveforms contain many more edges than type B waveforms. The lower frequencies in type B waveforms have one major advantage. Because the LCD presents a capacitive load, the drive current rises with frequency. Therefore, type B waveforms result in lower power consumption. This is especially important in battery powered applications.

The length of the frame frequency or refresh rate is the same for both types of waveform. Therefore, there are no differences between type A and type B waveforms in refresh rate dependent optical parameters of the LCD segments. Contrast is dependent on the light source available, viewing angle, Multiplex mode, and the LCD voltage levels. The first three parameters are directly related to LCD glass and the fourth can be controlled by the LCD driver. The LCD bias adjustment controls the contrast between the LCD segment in On and Off states. This voltage must be optimized for best appearance. A greater voltage separation between common and segment pins allows better contrast.

When will the waveform type matter? And when is one wave form type better than the other? Basically, the waveform type matters when you are using an LCD display that requires two or more commons. So, for waveform type A you will get more contrast control, because the waveform is changing less often, thus the LCD segments spend less time in transition. Waveform type B would be better for LCD displays with larger segment sizes that require more transition time and have shorter or limited refresh rates.

**FIGURE 5: TYPE A VS. TYPE B WAVEFORMS**




**Note:** Shaded area indicates the LCD is energized.

## Voltage Bias

What does bias mean? Bias is the number of voltage steps to be applied to the LCD. To control LCDs with a larger multiplex ratio, you must provide the waveform generator with multiple bias voltage level points. The resulting waveform sent to the LCD segment control lines and backplane/commons contains a stair-stepped waveform. This maintains specific AC voltages across any given segment, dot, and pixel to keep it in its On or Off state. The LCD bias number (for example, 1/3 bias) indicates how many voltage reference points are created to drive a specific LCD. Table 1 shows the relationship between the number of driving bias voltages and the display multiplex ratios typically used.

For the PIC16F1947 microcontroller, the LCD module can be configured for one of three bias types:

a) Static Bias (2 voltage levels: VSS and VLCD)

b) 1/2 Bias (3 voltage levels: VSS, 1/2 VLCD and VLCD)

c) 1/3 Bias (4 voltage levels: VSS, 1/3 VLCD, 2/3 VLCD and VLCD)

Being that the microcontroller and LCD used in this application are configured for a 1/4 multiplex, the Bias mode will be set to 1/3 bias. This will give you the lowest possible LCD voltage, 1/3 VDD.

## Frame Frequency

The LCD prescaler lets you divide the clock frequencies to set the LCD frame clock rate which will allow us to adjust the frame frequency. The frame frequency is the rate at which the backplane or common and the segment outputs of the LCD change, this may also be called 'Refresh Rate'. The range of frame frequencies is from 25 to 250 Hz with the most common being between 50 and 150 Hz. The higher frequencies result in higher power consumption, while lower frequencies cause flicker in the image on the LCD panel. Figure 6 shows the LCD clock generation path used to set the frame frequency. Table 4 shows Figure 6 in equations form for each multiplex configuration. Table 5 gives frame frequencies for several LCD prescaler values.

For more details on the LCDPS register and its bit descriptions and functions, see Register 27-2 of the PIC16F1947 device data sheet (DS41414).

**FIGURE 6: LCD CLOCK GENERATION**

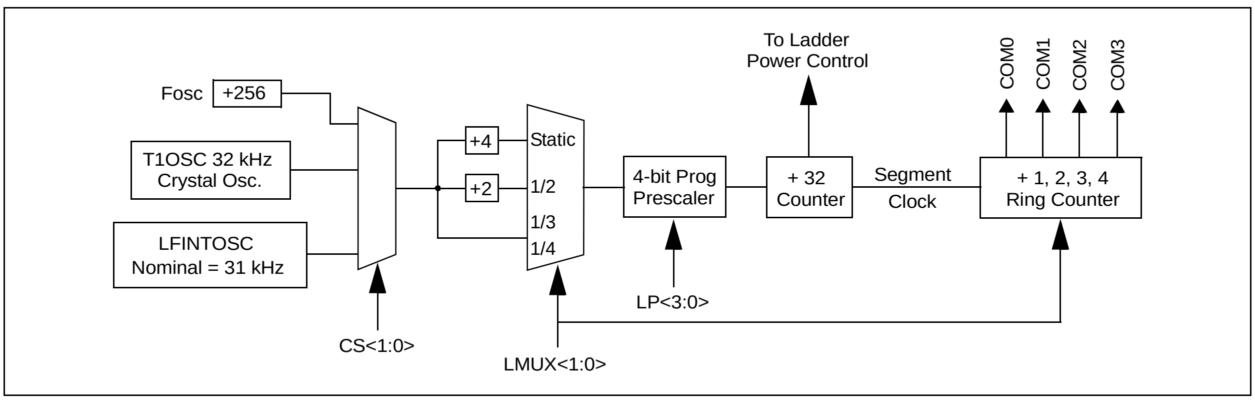



**TABLE 4: FRAME FREQUENCY CALCULATION**

| Multiplex | Frame Frequency = |
|---|---|
| Static | Clock source/(4x1x(LCD Prescaler)x32)) |
| 1/2 | Clock source/(2x2x(LCD Prescaler)x32)) |
| 1/3 | Clock source/(1x3x(LCD Prescaler)x32)) |
| 1/4 | Clock source/(1x4x(LCD Prescaler)x32)) |

**TABLE 5: FRAME FREQUENCIES FOR DIFFERENT PRESCALER VALUES**

| LP<3:0> | Static | 1/2 | 1/3 | 1/4 |
|---|---|---|---|---|
| 2 | 122 | 122 | 162 | 122 |
| 3 | 81 | 81 | 108 | 81 |
| 4 | 61 | 61 | 81 | 61 |
| 5 | 49 | 49 | 65 | 49 |
| 6 | 41 | 41 | 54 | 41 |
| 7 | 35 | 35 | 47 | 35 |

## LCD REFERENCE LADDER REGISTER (LCDRL)

For this application, this register is used to:

- set the LCD reference ladder A time power to Low-power mode,
- set the LCD reference ladder B time power to Low-power mode and,
- set the number of 32 kHz clocks that the A time interval power mode is active to always in 'B' power mode.

## POWER MODES

As an LCD segment is electrically only a capacitor, current is drawn only during the interval where the voltage is switching. To minimize total device current, the LCD internal reference ladder can be operated in a higher power mode for the switching time interval of the LCD segment, and a lower power mode for the remainder of the frame time.

The LCDRL register allows you to switch between two Power modes, 'A' and 'B'. The 'A' Power mode is active for a programmable amount of time, beginning at the time the LCD segment transitions, or switches on. The 'B' Power mode is the remaining time before the LCD segment or common changes again. This will give you flexibility in running the LCD. For example, the LCD can run in high power for a short amount of time and run the LCD in the Low-power mode for a longer amount of time. See Figure 7 and Figure 8 for examples of the two power modes for both waveform types A and B.

**FIGURE 7: LCD INTERNAL REFERENCE LADDER POWER MODE SWITCHING DIAGRAM-TYPE A WAVEFORM (½ MUX, ½ BIAS DRIVE)**

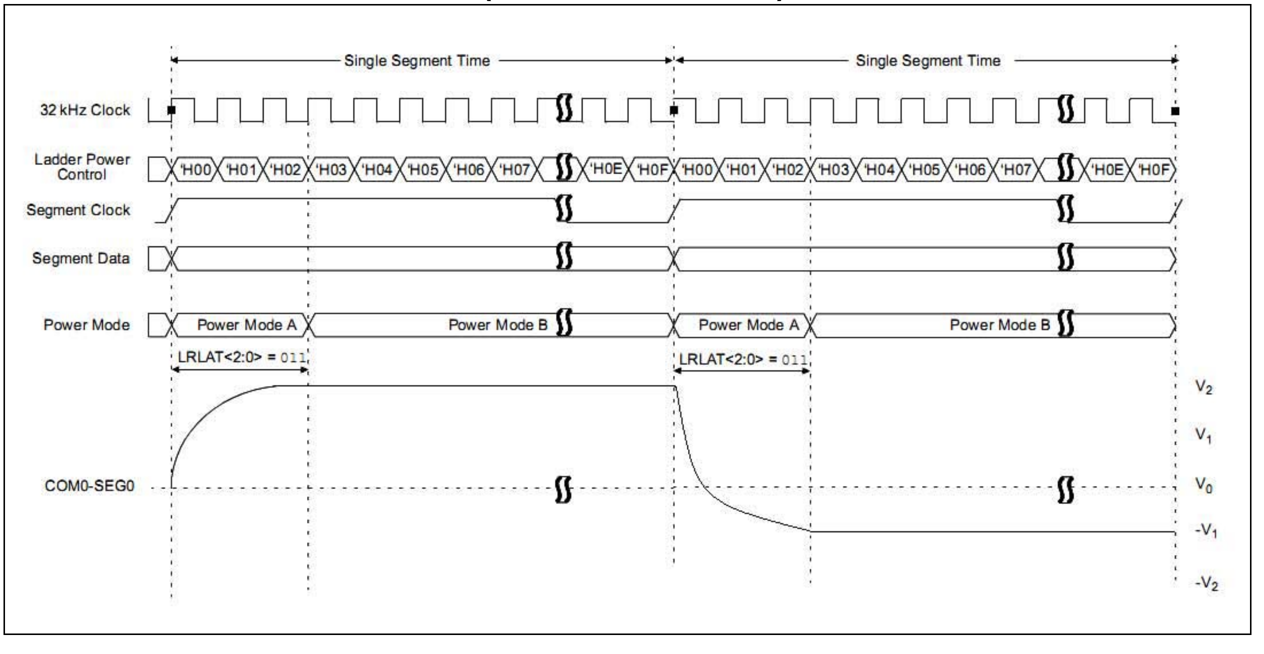
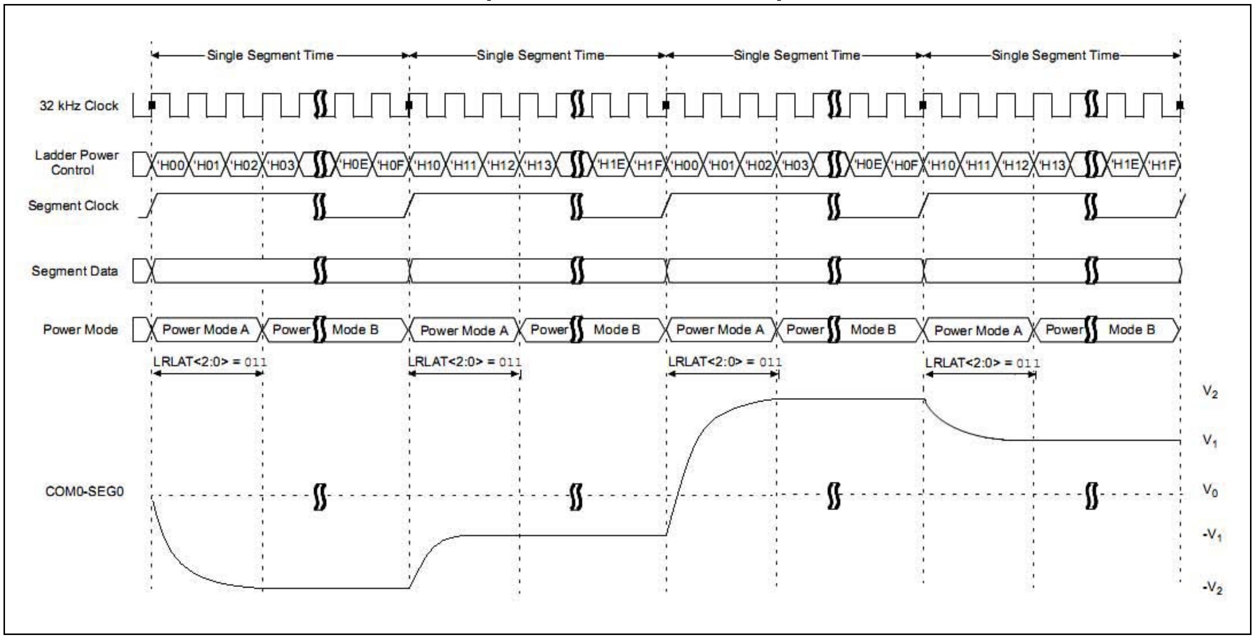


**FIGURE 8: LCD INTERNAL REFERENCE LADDER POWER MODE SWITCHING DIAGRAM-TYPE B WAVEFORM (½ MUX, ½ BIAS DRIVE)**




The internal reference ladder may operate in one of three power modes. The three different power modes are; Low, Medium and High. One power mode is not any better than the others for all microcontrollers. The best power mode for a given microcontroller and/or application will be dependent upon temperature, board leakage, LCD leakage, capacitance of LCD, and your understanding/preference of an acceptable contrast level. Thus, it allows you to trade off LCD contrast for power in a specific application. The larger the LCD glass, the more capacitance is present on a physical LCD segment, thus requiring more current to maintain the same contrast level. The internal reference ladder can also be disconnected for applications that wish to provide an external ladder or to minimize power consumption.

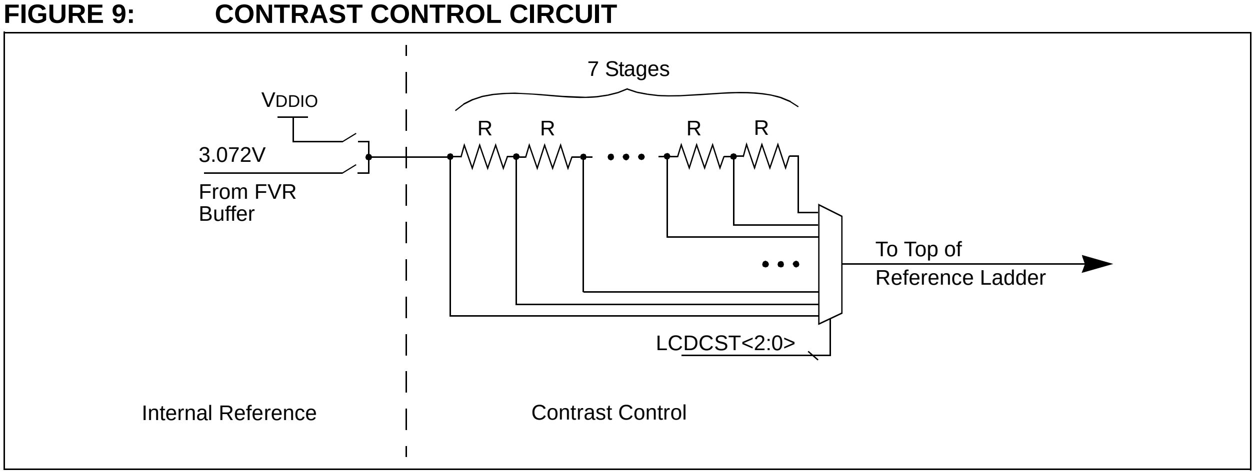
For more details on the LCDRL register and its bit descriptions and functions, see Register 27-7 of the PIC16F1947 device data sheet (DS41414).

## LCD CONTRAST CONTROL REGISTER (LCDCST)

For this application, this register is used to:

- set the LCD contrast resistance to the minimum resistance (i.e., the resistor ladder is shorted). (Therefore, the maximum contrast, where the LCD segments are at their darkest).

**FIGURE 9: CONTRAST CONTROL CIRCUIT**

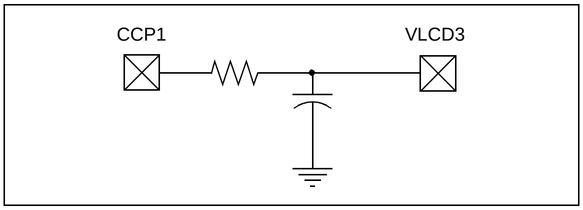


### Contrast Control

The contrast control circuit is used to decrease the output voltage of the signal source by a total of approximately 10%, thus the 7-stage resistor ladder is at maximum resistance (minimum contrast), therefore the LCD segments are at their lightest. Likewise, when you increase the output voltage, the 7-stage resistor ladder is at its minimum resistance (maximum contrast), thus the LCD segments are at their darkest.

So, to save power, you will want to find a contrast setting that will give you enough resistance to see the LCD segments, but does not use maximum voltage. However, keep in mind that the contrast can and will be affected by the temperature of the LCD display glass, for example, the environmental temperature the display is in. Also, you can set up the LCD to use an external contrast control circuit using pulse-width modulation (PWM). This gives you the option of having a larger contrast range (more than 10%) at a faster rate. See Figure 10 for an example of an external contrast control circuit using the pulse-width modulation (PWM) peripheral of the PIC16F1947 microcontroller.

**FIGURE 10: EXTERNAL CONTRAST CONTROL CIRCUIT EXAMPLE**




For more details on the LCDCST register and its bit descriptions and functions, see Register 27-4 of the PIC16F1947 device data sheet (DS41414).

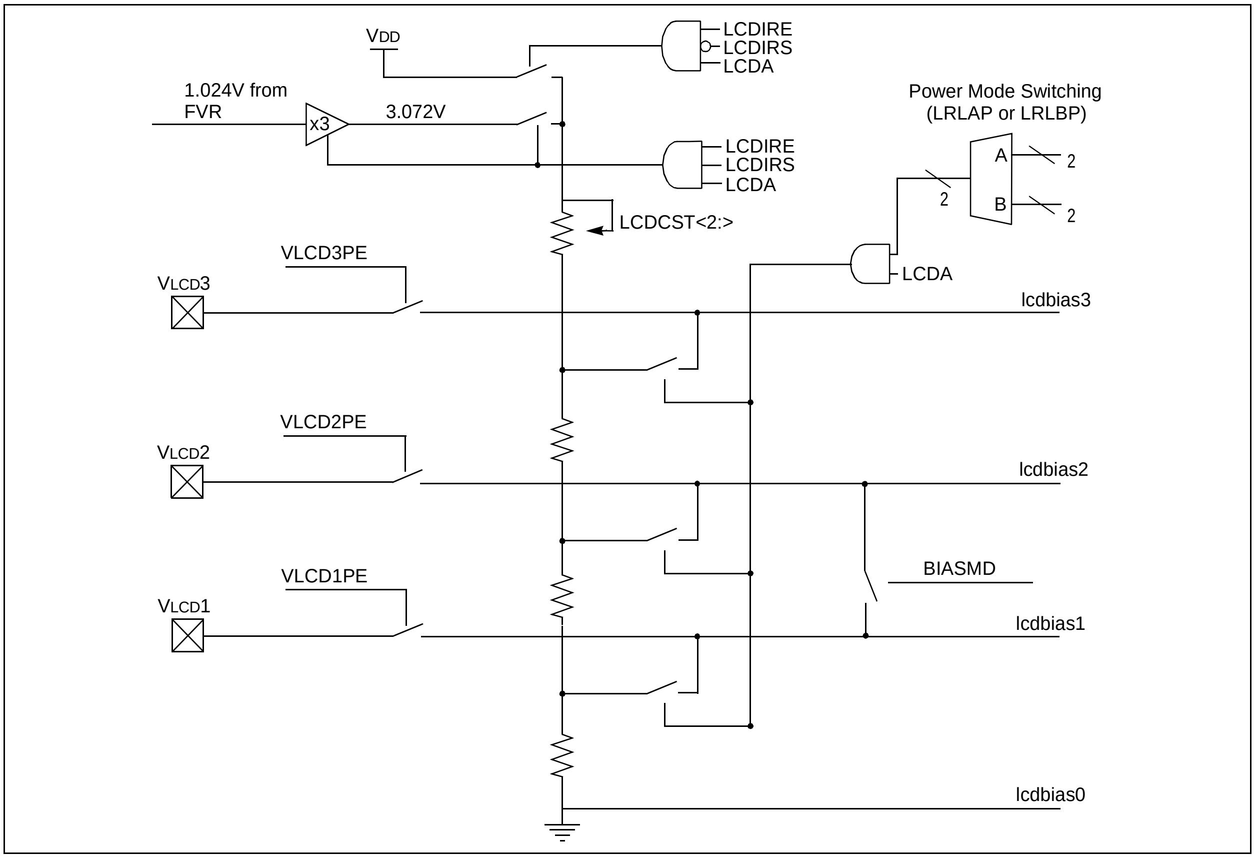
## LCD REFERENCE VOLTAGE CONTROL REGISTER (LCDREF)

For this application, this register is used to:

- set the LCD internal voltage reference on (thus connecting to the internal contrast control circuit),
- set the internal contrast control to be powered by $V_{DD}$,
- set the internal voltage reference ladder to allow the Fixed Voltage Reference (FVR) to shut down when the LCD voltage reference ladder is in power mode ‘B’,
- disconnect the LCD voltage pins from the bias voltage generator

This register gives you the option of two different voltage references to drive the LCD and contrast control circuit. You can select from $V_{DD}$ or the internal Fixed Voltage Reference, as shown in Figure 11. This will allow you to drive a 5V LCD or a 3V LCD display, but only if the microcontroller is running at 5 volts. If the microcontroller is running at 3 volts, you would only be able to run a 3-volt LCD display.

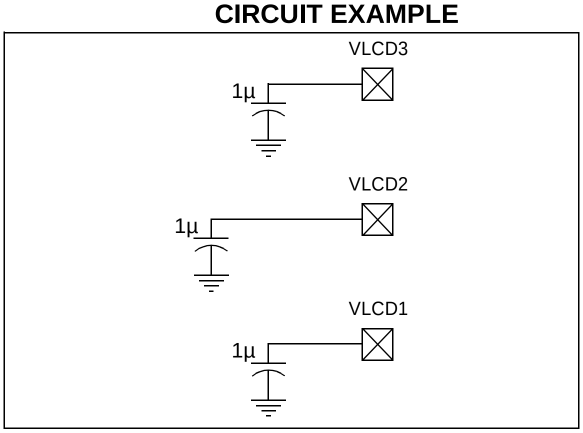
**FIGURE 11: LCD BIAS VOLTAGE GENERATOR BLOCK DIAGRAM**




The block diagram in Figure 11 can also be modified using external capacitors to reduce power consumption. By putting external capacitors on the VLCD3, 2 and 1 pins, the capacitors will charge up when the LCD is not being used and will discharge when the LCD is in use. Thus, reducing the amount of current needed from the resister ladder and allowing you to turn the ladder off. For an example of this modification, see Figure 12.

For more details on the LCDREF register and its bit descriptions and functions, see Register 27-3 of the PIC16F1947 device data sheet (DS41414).

**FIGURE 12: EXTERNAL CAPACITOR CIRCUIT EXAMPLE**




## LCD SEGMENT ENABLE REGISTERS (LCDSen)

For this application, this register is used to enable/turn on all LCD segments that you will be using. Looking at source code Example 1, all of the LCDSEn registers have all segment bits associated to their corresponding LCDDATAn register bits set to '1' . This will enable all of the LCD display segments associated with these registers. Thus, the LCD source code shown in source code Example 2 and in the complete application source code shown in **"Appendix A – Complete Source Code"**, will be able to turn on each LCD display segment as or when needed by the main program.

See **Section "Segment Mapping"** and source code Example 1.

## LCD DATA REGISTERS (LCDDATAn)

For this application, this register is used to make all of the LCD segments either dark or clear as needed while the program is running. As you look at source code Example 1, you will notice that all of the LCDDATAn registers are set to '0'. This will turn all of the LCD display segments associated with these registers off. The LCD source code shown in source code Example 2 and in the complete application source code shown in **"Appendix A – Complete Source Code"**, will turn on (set to '1') each LCD display segment as or when needed by the main program.

For more details on the LCDDATAn register and its bit descriptions and functions, see Register 27-6 of the PIC16F1947 device data sheet (DS41414).

### Tying It All Together.

When you have gone through and determined which LCD display segments you want to drive, which microcontroller LCD segments will drive them, and how you want to configure the LCD module, you need to tie the source code to the application.

For the source code used in this application example, you do not need to use all segments of the LCD display. You need only to use digits 1 through 5 as shown in the LCD segment mapping worksheet, see Table 2. For simplicity purposes, the source code has been written to automatically use these digits as needed. See code Example 2.

## THE APPLICATION

The example application for this document is an electronic combination lock. It works as follows:

Select a 3-digit code – for example: 4, 5, 6.

- Push button 1:

  This button lets you select a number from 0 to 9. Thus, as you press the button, the LCD will display numbers incrementing from 0 to 9, then back to 0, and so on.

- Push button 2:

  This is the Enter button. So, when the LCD displays the first number of the 3-digit code using push button 1, enter the number using push button 2. Repeat this process until all 3 digits are entered, then the LCD will display "UNLOC" (Figure 1).

- Push button 3:

  This is the Reset button. Press this button to reset the device.

- Push button 4:

  This is the LCD contrast control. As you press this button, you will notice the brightness/contrast of the pixels get lighter and lighter, then back to maximum darkness.

See **"Appendix A – Complete Source Code"** for the complete source code and schematic used in this application.

**EXAMPLE 2: LCD SEGMENT IMPLEMENTATION FOR DIGIT 1 (PART 1 OF 2)**

```
/*
 * Update digit 1 - Labeled 1A-1G on the display
 */
lcd_tmp = seg7_cvt(Dig1);

/* Segment A */
if ( lcd_tmp & SEG_A )
    SEG1COM0 = 1;
else
    SEG1COM0 = 0;

/* Segment B */
if ( lcd_tmp & SEG_B )
    SEG3COM0 = 1;
else
    SEG3COM0 = 0;

/* Segment C */
if ( lcd_tmp & SEG_C )
    SEG3COM1 = 1;
else
    SEG3COM1 = 0;

/* Segment D */
if ( lcd_tmp & SEG_D )
    SEG3COM2 = 1;
else
    SEG3COM2 = 0;

/* Segment E */
if ( lcd_tmp & SEG_E )
    SEG1COM1 = 1;
else
    SEG1COM1 = 0;

/* Segment F */
if ( lcd_tmp & SEG_F )
    SEG1COM3 = 1;
else
    SEG1COM3 = 0;

/* Segment G */
if ( lcd_tmp & SEG_G )
    SEG3COM3 = 1;
else
    SEG3COM3 = 0;
```

**EXAMPLE 2: LCD SEGMENT IMPLEMENTATION FOR DIGIT 1 (PART 2 OF 2)**

```
/* Convert the integer value to which segments need to be turned on or off */
unsigned char seg7_cvt(unsigned char digit)
{
    switch (digit)
    {
        case '0':
        case 0:
               return SEG_A | SEG_B | SEG_C | SEG_D | SEG_E | SEG_F;
        case '1':
        case 1:
               return SEG_B | SEG_C;
        case '2':
        case 2:
               return SEG_A | SEG_B | SEG_D | SEG_E | SEG_G;
        case '3':
        case 3:
               return SEG_A | SEG_B | SEG_C | SEG_D | SEG_G;
        case '4':
        case 4:
               return SEG_B | SEG_C | SEG_F | SEG_G;
        case '5':
        case 5:
               return SEG_A | SEG_C | SEG_D | SEG_F | SEG_G;
        case '6':
        case 6:
               return SEG_A | SEG_C | SEG_D | SEG_E | SEG_F | SEG_G;
        case '7':
        case 7:
               return SEG_A | SEG_B | SEG_C;
        case '8':
        case 8:
               return SEG_A | SEG_B | SEG_C | SEG_D | SEG_E | SEG_F | SEG_G;
        case '9':
        case 9:
               return SEG_A | SEG_B | SEG_C | SEG_D | SEG_F | SEG_G;
        case 'L':
               return SEG_F | SEG_E | SEG_D;
        case 'O':
               return SEG_A | SEG_B | SEG_C | SEG_D | SEG_E | SEG_F;
        case 'C':
               return SEG_A | SEG_F | SEG_E | SEG_D;
        case 'U':
               return SEG_F | SEG_E | SEG_D | SEG_C | SEG_B;
        case 'n':
               return SEG_E | SEG_G | SEG_C;
        case ' ':
               return 0;
        case '-':
               return SEG_G;

        default:   /* Display a visible pattern when we have something we don't understand
*/
                 return 0;
        }
}
```

## CONCLUSION

The PIC16F1947 microcontroller is ideally suited for LCD applications such as clocks, meters, thermostats, etc. This application note describes how to configure/ optimize the LCD module for low-power consumption and implement it in a real world application. Nevertheless, the configurations discussed will be more dependent on the application and the LCD display used. However, if you optimize your source code to take advantage of the Sleep mode whenever possible, optimize the LCD module settings to find the perfect balance of contrast vs. power consumption and use good power, ground and noise techniques when designing your applications' circuit, you will be well on your way to maximizing the LCD module of PIC16F1947 microcontroller and its features to enhance any application.

Please refer to application note AN658, "*LCD Fundamentals using the PIC16C92X Microcontrollers*", for a more detailed discussion on how LCD displays are constructed.

**NOTES:**

## APPENDIX A – COMPLETE SOURCE CODE

**FIGURE 1: SCHEMATIC (1 OF 3)**

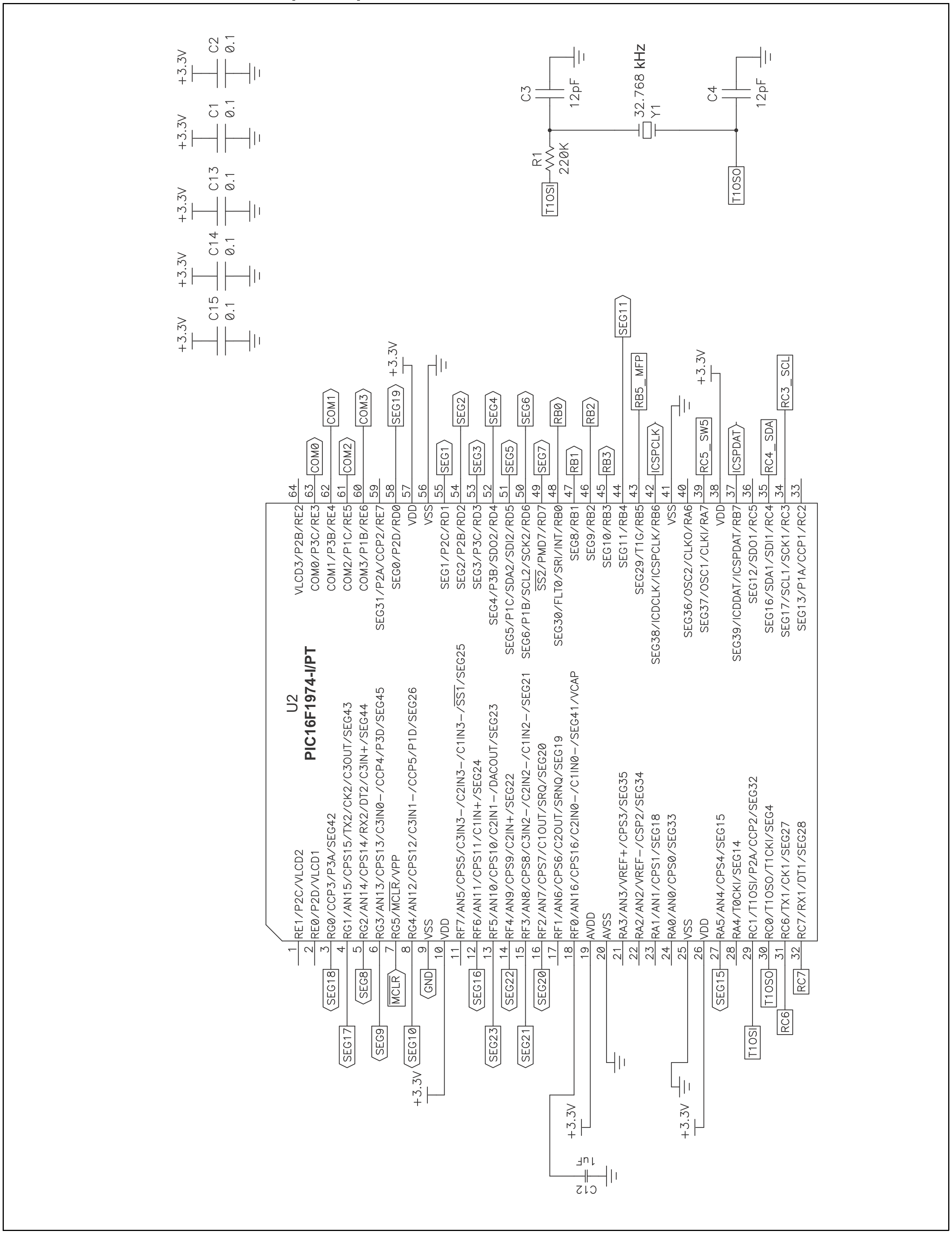


**FIGURE 2: SCHEMATIC (2 OF 3)**

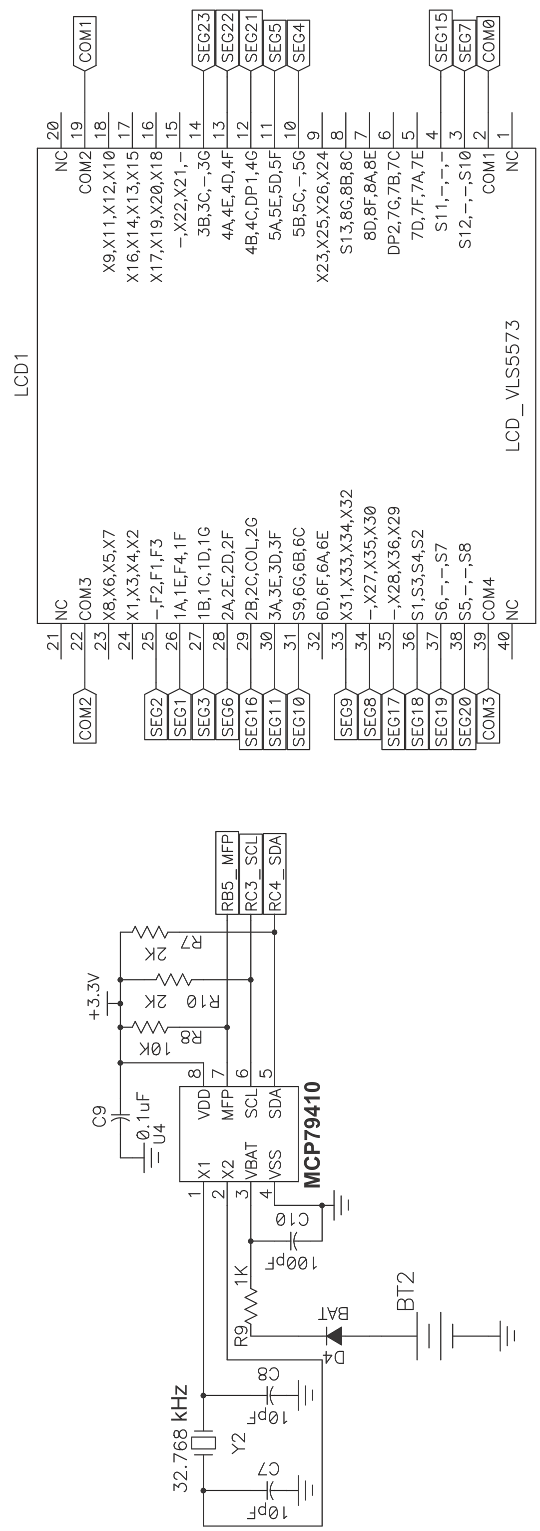
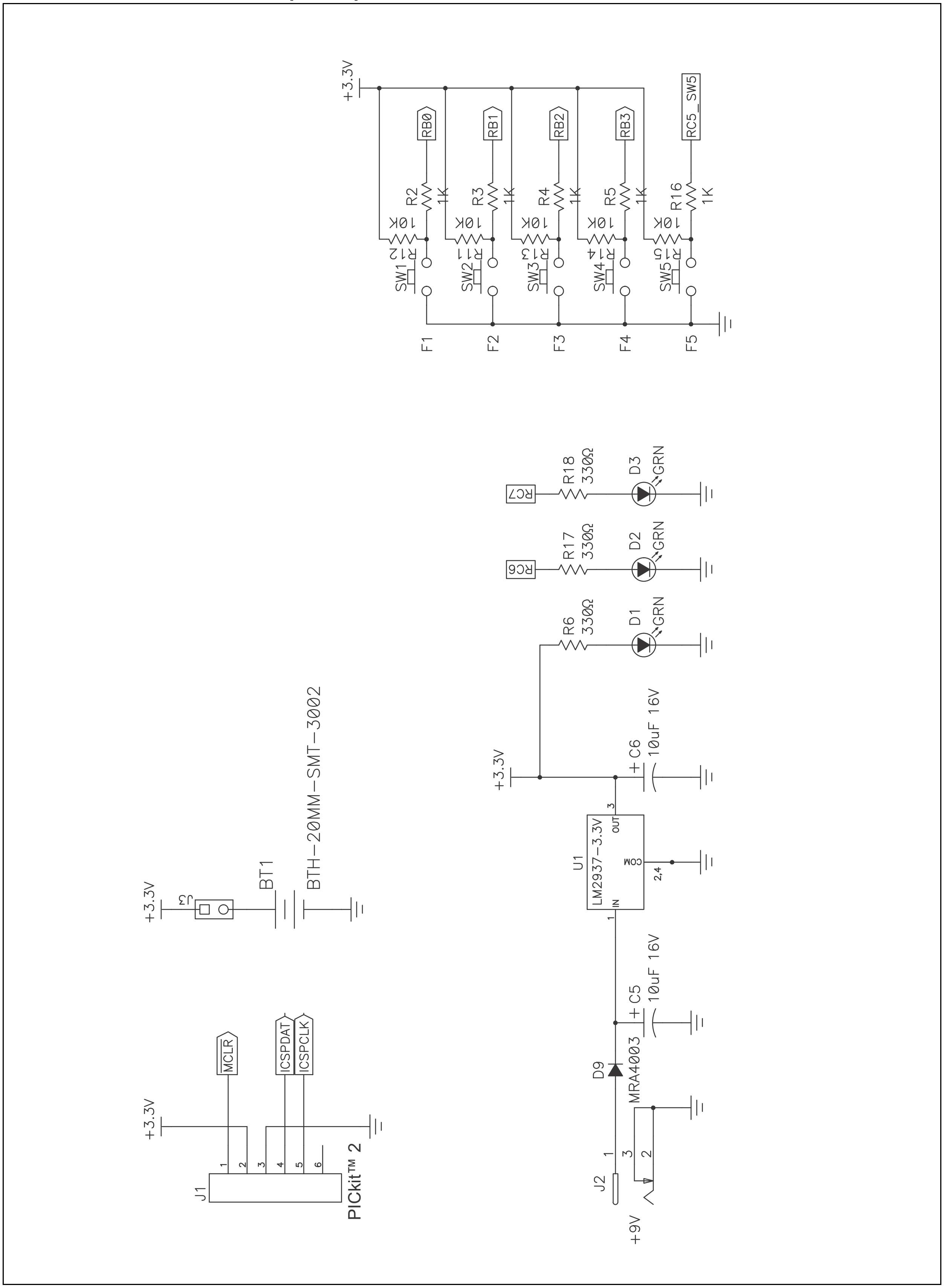


**FIGURE 3: SCHEMATIC (3 OF 3)**



**EXAMPLE : COMPLETE SOURCE CODE**

```
/*
-------------------------------------------------------
    Filename:    1947comboLoc.c
    Date:                March 2-2010
    File Version:   1.0
    Written by:     John Mouton
    Company:        Microchip Technology

    Files required:  pic.h, LCD.c, and lcd.h
*/

#include <htc.h>
#include "lcd.h"

// Setup the configuration word for use with ICD2

__CONFIG(FOSC_INTOSC & WDTE_OFF & MCLRE_ON & PWRTE_ON & BOREN_ON & IESO_OFF &
                   FCMEN_OFF & CP_OFF & CPD_OFF & CLKOUTEN_OFF);
__CONFIG(WRT_OFF & VCAPEN_OFF & PLLEN_OFF & STVREN_ON & DEBUG_OFF & LVP_OFF &
                  BORV_19);

static unsigned charCombination_Number_Flag = 0;// Combonation number flag.

         // Combination Code
static unsigned charCombination_Digit_1 = 4;   //###  You select this value. #####
static unsigned charCombination_Digit_2 = 5;   //###  You select this value. #####
static unsigned charCombination_Digit_3 = 6;   //###  You select this value. #####

                // Declarations
static unsigned char  NUMBER_SELECT_BUTTON = 0;         // flag indicating that SW1 was pressed.
static unsigned char  ENTER_BUTTON = 0;                 // flag indicating that SW2 was pressed.
static unsigned char  RESET_BUTTON = 0;                 // flag indicating that SW3 was pressed.
static unsigned char  CONTRAST_CONTROL_BUTTON = 0;      // flag indicating that SW4 was pressed.
static unsigned char  state_variable = 0;               // state counter variable.
static unsigned char  Number_select_button_counter = 0;     // counter variable for NUM_SELECT flag in
                                                                debounce subroutine.
static unsigned char  Enter_button_counter = 0;             // counter variable for ENTER flag in
                                                                debounce subroutine.
static unsigned char  Reset_button_counter = 0;             // counter variable for RESET flag in
                                                                debounce subroutine.
static unsigned char  Contrast_Control_button_counter = 0;  // counter variable for CONTRAST_CONTROL
                                                                flag in debounce subroutine.
static unsigned char  Number_selection_counter = 0;         // counter variable for Decide_1 function.
static unsigned char  Decide_1_function_state_variable = 0; // state machine count variable for
                                                                Decide_1 function.
static unsigned char  Decide_4_function_state_variable = 0; // state machine count variable for
                                                                Decide_4 function.
static unsigned char  Decide1_output_flag = 0;              // output flag variable for Decide_1
                                                                function.
static unsigned char  Decide2_output_flag = 0;              // output flag variable for Decide_2
                                                                function.
static unsigned char  Decide3_output_flag = 0;              // output flag variable for Decide_3
                                                                function.
static unsigned char  Decide4_output_flag = 0;              // output flag variable for Decide_4
                                                                function.
```

**EXAMPLE : COMPLETE SOURCE CODE (CONTINUED)**

```
unsigned long int  Interrupt_count = 0;                       // 32 bit interrupt counter variable
static unsigned char  lockup_flag;                               // interrupt variable to lockup
                                                                    the device if interrup occures.

                   // function prototype
void INIT(void);                     // device initialization function.
void GET_inputs(void);               // recieve all inputs function.
void Decide_1(void);                 // based on inputs, select a number function.
void Decide_2(void);                 // based on inputs, check/verify number selection function.
void Decide_3(void);                 // based on inputs, check/verify reset function.
void Decide_4(void);                 // based on inputs, check/verify contrast control selection
function.
void DO_outputs(void);               // based on decisions, display proper outputs function.
void leds_on(void);                  // sub-fuction of DO_out function to turn LEDs on.
void leds_off(void);                 // sub-fuction of DO_out function to turn LEDs off.
void interrupt Time_out_int(void);   // interrupt subroutine timeout function.

/*-------------------------------------------------------------------------------------
        Subroutine: INIT
        Parameters: none
        Returns:nothing
        Synopsys:Initializes flags, and variables, sets PORT direction,
                        configures analog/digital pins, and disables the comparator module
-----------------------------------------------------------------------------------*/

void INIT(void) {

    TRISA  = 0b11000000;    // set port A as outputs.
    TRISB  = 0b11000000;          // set port B bits 4,5,6,7 as outputs and RB0,1,2,3 as inputs.
    TRISC  = 0x00;                      // set port C as outputs.
    TRISD  = 0x00;                      // set port D as outputs.
    TRISE  = 0x00;                      // set port E as outputs.
    TRISF  = 0x00;
    TRISG  = 0x00;

   ANSELA  = 0x00;                             // make all analog outputs.
   ANSELE  = 0x00;                             // make all analog outputs.
   ANSELF  = 0x00;
   ANSELG  = 0x00;

    LATA    = 0x00;
    LATB    = 0x00;
    LATC    = 0x00;
    LATD    = 0x00;
    LATE    = 0x00;
    LATF    = 0x00;
    LATG    = 0x00;

    CM1CON0 = 0x07;            // turn off the comparators.
    CM2CON0 = 0x07;
    CM1CON1 = 0x07;
    CM2CON1 = 0x07;
    OPTION_REG = 0x00;         // clear the OPTION_REG.
    INTCON = 0xA0;             // enable peripheral interrupts
    TMR0 = 0;                          // clear timer 0.
    RD4    = 0;                        // set RD4 to 0.
    RB7    = 0;
    PORTA  = 0;                        // CLEAR ALL PORTS, variables, and flags.
    PORTB  = 0;
    PORTC  = 0;
    PORTD  = 0;
    PORTE  = 0;
    PORTF  = 0;
    PORTG  = 0;
```

**EXAMPLE : COMPLETE SOURCE CODE (CONTINUED)**

```
//  Initialize variables

    Number_select_button_counter = 0;
    Enter_button_counter = 0;
    Reset_button_counter = 0;
    Contrast_Control_button_counter = 0;
    Number_selection_counter = 0;
    Decide_1_function_state_variable = 0;
    Decide_4_function_state_variable = 0;
    NUMBER_SELECT_BUTTON = 0;
    ENTER_BUTTON = 0;
    RESET_BUTTON = 0;
    CONTRAST_CONTROL_BUTTON = 0;
    Combination_Number_Flag = 0;
    Decide1_output_flag = 0;
    Decide2_output_flag = 0;
    Decide3_output_flag = 0;
    Decide4_output_flag = 0;

       // Initialize your combination value here.

    Combination_Digit_1 = 4;                  // ###  You select this value. #####
    Combination_Digit_2 = 5;                  // ###  You select this value. #####
    Combination_Digit_3 = 6;                  // ###  You select this value. #####

    // Initializing the Interrupt routine variables.
    Interrupt_count = 0;         // clear interrupt counter.
    lockup_flag = 0;             // clear interrupt lockup flag.

     // Set up the LCD for use
    lcd_init();                  // calls the lcd initialization function.
    SEG18COM0 = 1;   // MCHP logo
}

/*-------------------------------------------------------------------------------------
        Subroutine: main
        Parameters: none
        Returns:nothing
        Synopsys:Main program function
-------------------------------------------------------------------------------------*/

void main(void)
{
        INIT();            //Initialize all registers

        while(1)
        {
        GET_inputs();    //Get inputs from off-chip
        Decide_1();   //Make decisions based on inputs
            Decide_2();        //Make decisions based on inputs
            Decide_3();        //Make decisions based on inputs for Reset
            Decide_4();        //Make decisions based on inputs for Contrast Control
        DO_outputs();    //Do outputs based on decisions
        }
}

/*-------------------------------------------------------------------------------------
        Subroutine: GET_inputs
        Parameters: none
        Returns:nothing
        Synopsys:Gets the inputs from buttons on Mechatronics Demo Board.
-------------------------------------------------------------------------------------*/
```

**EXAMPLE : COMPLETE SOURCE CODE (CONTINUED)**

```
void GET_inputs(void) {

// DEBOUNCE FUNCTION FOR SW1.

        if (RB6 == 0) Number_select_button_counter++;      // this routine eliminates the contact
                                                              bounce of the SW1 button being pressed.
        else          Number_select_button_counter = 0;
        if (Number_select_button_counter >= 9) Number_select_button_counter = 9; // it looks for 8
                                                                                  consective lows before
        if (Number_select_button_counter == 8) NUMBER_SELECT_BUTTON = 1;  // it determines the button
                                                                              is pressed

// DEBOUNCE FUNCTION FOR SW2.

        if (RB7 == 0) Enter_button_counter++; // this routine eliminates the contact bounce of the SW2
                                                 button being pressed.
        else          Enter_button_counter = 0;
        if (Enter_button_counter >= 9) Enter_button_counter  = 9;   // it looks for 8 consective lows
                                                                      before
        if (Enter_button_counter == 8) ENTER_BUTTON = 1; // it determines the button is pressed

// DEBOUNCE FUNCTION FOR SW3.

        if (RA6 == 0) Reset_button_counter++; // this routine eliminates the contact bounce of the SW3
                                                 button being pressed.
        else          Reset_button_counter = 0;
        if (Reset_button_counter >= 9) Reset_button_counter  = 9;       // it looks for 8 consective
                                                                          lows before
        if (Reset_button_counter == 8) RESET_BUTTON = 1;        // it determines the button is pressed

// DEBOUNCE FUNCTION FOR SW4.

        if (RA7 == 0) Contrast_Control_button_counter++; // this routine eliminates the contact bounce
                                                            of the SW4 button being pressed.
        else          Contrast_Control_button_counter = 0;
        if (Contrast_Control_button_counter >= 9) Contrast_Control_button_counter  = 9;   // it looks
                                                                                    for 8 consective lows before
        if (Contrast_Control_button_counter == 8) CONTRAST_CONTROL_BUTTON = 1; // it determines the
                                                                                  button is pressed

}


/*-------------------------------------------------------------------------------------
        Subroutine: Decide_1
        Parameters: none
        Returns:nothing
        Synopsys:This function steps through numbers 0 - 9, then back to 0, each time
                the user presses the SW2 button.
-------------------------------------------------------------------------------------*/

void Decide_1(void) {

             if (NUMBER_SELECT_BUTTON == 1)
                  {
                    Decide1_output_flag = 1;
                    NUMBER_SELECT_BUTTON = 0;
```

**EXAMPLE : COMPLETE SOURCE CODE (CONTINUED)**

```
        switch(Decide_1_function_state_variable){     // State machine for the number
                                                          selection.

            case 0:
                  {
                  Number_selection_counter = 0;
                  Combination_Number_Flag = Number_selection_counter;
                  Decide_1_function_state_variable = 1;
                  }
                  break;

            case 1:
                  {
                  Number_selection_counter = 1;
                  Combination_Number_Flag = Number_selection_counter;
Decide_1_function_state_variable = 2;

                  }
                  break;

            case 2:
                  {
                  Number_selection_counter = 2;
                  Combination_Number_Flag = Number_selection_counter;
        Decide_1_function_state_variable = 3;

                  }
                  break;

            case 3:
                  {
                  Number_selection_counter = 3;
             Combination_Number_Flag = Number_selection_counter;

                  Decide_1_function_state_variable = 4;
                  }
                  break;

            case 4:
                  {
                  Number_selection_counter = 4;
              Combination_Number_Flag = Number_selection_counter;

                  Decide_1_function_state_variable = 5;
                  }
                  break;

            case 5:
                  {
                  Number_selection_counter = 5;
              Combination_Number_Flag = Number_selection_counter;

                  Decide_1_function_state_variable = 6;
                  }
                  break;
```

**EXAMPLE : COMPLETE SOURCE CODE (CONTINUED)**

```
                        case 6:
                              {
                              Number_selection_counter = 6;
                          Combination_Number_Flag = Number_selection_counter;

                              Decide_1_function_state_variable = 7;
                              }
                              break;

                        case 7:
                              {
                              Number_selection_counter = 7;
                              Combination_Number_Flag = Number_selection_counter;
                              Decide_1_function_state_variable = 8;
                              }
                              break;

                        case 8:
                              {
                              Number_selection_counter = 8;
                          Combination_Number_Flag = Number_selection_counter;

                              Decide_1_function_state_variable = 9;
                              }
                              break;

                        case 9:
                              {
                              Number_selection_counter = 9;
                          Combination_Number_Flag = Number_selection_counter;

                              Decide_1_function_state_variable = 0;
                              }
                              break;

                        default:
                              {
                              Decide_1_function_state_variable = 0;
                              }
                              break;

                      }
                    }
}

/*-------------------------------------------------------------------------------------
        Subroutine: Decide_2
        Parameters: none
        Returns:nothing
        Synopsys:This function will check the numbers selected and entered as the three
                digit code to unlock the electronic lock.
-------------------------------------------------------------------------------------*/

void Decide_2(void) {

            if (ENTER_BUTTON == 1)
               {
                  Decide2_output_flag = 1;
                  ENTER_BUTTON =0;
              switch(state_variable){
```

**EXAMPLE : COMPLETE SOURCE CODE (CONTINUED)**

```
                        case 0:
                              if (Combination_Number_Flag == Combination_Digit_1)   // checks the first
                                                            number entered of the three digit combo.
                               {
                                state_variable = 1;                // increments state variable
                                 Decide_1_function_state_variable = 0;
                                 }
                              else
                                 {
                                 state_variable = 0;
                                 Decide_1_function_state_variable = 0;
                                 }
                                 break;
                        case 1:
                             if (Combination_Number_Flag == Combination_Digit_2)    // checks the second
                                                            number entered of the three digit combo.
                                 {
                                 state_variable = 2;            // increments state variable
                                 Decide_1_function_state_variable = 0;
                                 }
                              else
                                 {
                                 state_variable = 0;
                                 Decide_1_function_state_variable = 0;
                                 }
                                 break;
                     case 2:
                              if (Combination_Number_Flag == Combination_Digit_3)   // checks the third
                                                                  number entered of the three digit combo.
                                 {
                                 state_variable = 3;           // increments state variable, thus when
                                                                      the state variable
                                                              // equals 3, the entire three digit combo
                                                                     is verified correct.
                                 Decide_1_function_state_variable = 0;
                                 }
                              else
                                 {
                                 state_variable = 0;
                                 Decide_1_function_state_variable = 0;
                                 }
                                 break;
                        case 3:
                                 {
                                 state_variable = 0;
                                 Decide_1_function_state_variable = 0;
                                 }
                                 break;

                        default:                 // default case.
                                 {
                                 state_variable = 0;
                                 Decide_1_function_state_variable = 0;
                                 }
                                 break;

                            }
                     }
}
```

**EXAMPLE : COMPLETE SOURCE CODE (CONTINUED)**

```
/*----------------------------------------------------------------------------------
        Subroutine: Decide_3
        Parameters: none
        Returns:nothing
        Synopsys:This function will monitor SW3 and if it is pressed, the demo will reset.
----------------------------------------------------------------------------------*/

void Decide_3(void) {

                if (RESET_BUTTON == 1)
               {
                  Decide3_output_flag = 1;
                  RESET_BUTTON =0;
                  INIT();
               }

}

/*----------------------------------------------------------------------------------
        Subroutine: Decide_4
        Parameters: none
        Returns:nothing
        Synopsys:This function will change the brightness of the LCD digits.
----------------------------------------------------------------------------------*/

void Decide_4(void) {

         if (CONTRAST_CONTROL_BUTTON == 1)
               {
                 Decide4_output_flag = 1;
                 LCDCST++;
                 CONTRAST_CONTROL_BUTTON =0;
               }

}

/*----------------------------------------------------------------------------------
        Subroutine: DO_out
        Parameters: none
        Returns:nothing
        Synopsys:This function will display Locked or unlocked
                                based on the outputs of the decision functions.
----------------------------------------------------------------------------------*/

void DO_outputs(void)
{

  if (lockup_flag == 1 )         // if the interrupt lockup flag is set,
                                 // dashed lines will appear on the LCD
     {
     lcd_update ('-','-','-','-','-');
     }

  else
     {
     if(Decide1_output_flag == 1 && Decide2_output_flag == 0 )   // if SW2 is pressed but SW3 isn't,
                                                                      then display the
                                                                 // current number value.
       {
       Decide1_output_flag = 0;
       lcd_update (' ',' ',' ',' ',Number_selection_counter);
       }
```

**EXAMPLE : COMPLETE SOURCE CODE (CONTINUED)**

```
     else if(Decide1_output_flag == 0 && Decide2_output_flag == 1)  // If SW2 was pressed and the
                                              number value you want is etered ( via SW3 )and it is
                                              correct. Then display unlocked, if not display locked.
        {
         Decide1_output_flag = 0;
         Decide2_output_flag = 0;

         if ( state_variable == 3 )
                {
            RB7 = 1;
                lcd_update ('U','n','L','O','C');
                Interrupt_count = 0;                 // The interrupt counter will reset.
                lockup_flag = 0;                     // The interrupt lockup flag will reset.
                TMR0IE = 0;                          // The timer 0 enable bit will turn off so the
                                                     // interrupt routine will not time out even if
                                                     // the electronic combo lock is unlocked.
                }

         else
                {
                lcd_update (' ','L','O','C',' ');
                RB7 = 0;
                }
         }

     else
        {
        Decide1_output_flag = 0;
        Decide2_output_flag = 0;
        }

     }
}

/*-------------------------------------------------------------------------------------
        Subroutine: Time_out_int
        Parameters: none
        Returns:nothing
        Synopsys:This is the interrupt function that will lock you out of the
                electronic combination lock if you don't enter the correct 3 digit
                combination fast enough.
-------------------------------------------------------------------------------------*/

void interrupt Time_out_int(void) {

     if (TMR0IF == 1 )                    // If the Timer 0 interrupt flag is set,
        {                                                // increment the interrupt counter and
         Interrupt_count ++ ;             // clear the Timer 0 interrupt flag.
         TMR0IF = 0;
        }

     if (Interrupt_count == 30000 )       // If the interrupt counter is equal to 30000,
        {                                                   // set the lockup flag.
         lockup_flag = 1;
        }

}

/*
```

**EXAMPLE : COMPLETE SOURCE CODE (CONTINUED)**

```
*********************************************************************************
*             CODE OWNERSHIP AND DISCLAIMER OF LIABILITY

*
* Microchip Technology Incorporated ("Microchip") retains all ownership and
* intellectual property rights in the code accompanying this message and in all
* derivatives hereto.  You may use this code, and any derivatives created by
* any person or entity by or on your behalf, exclusively with Microchip's
* proprietary products.  Your acceptance and/or use of this code constitutes
* agreement to the terms and conditions of this notice.
*
* CODE ACCOMPANYING THIS MESSAGE IS SUPPLIED BY MICROCHIP "AS IS". NO
* WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
* TO, IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY AND FITNESS FOR A
* PARTICULAR PURPOSE APPLY TO THIS CODE, ITS INTERACTION WITH MICROCHIP'S
* PRODUCTS,COMBINATION WITH ANY OTHER PRODUCTS, OR USE IN ANY APPLICATION.
*
* YOU ACKNOWLEDGE AND AGREE THAT, IN NO EVENT, SHALL MICROCHIP BE LIABLE,
* WHETHER IN CONTRACT, WARRANTY, TORT (INCLUDING NEGLIGENCE OR BREACH OF
* STATUTORY DUTY), STRICT LIABILITY, INDEMNITY,CONTRIBUTION, OR OTHERWISE, FOR
* ANY INDIRECT, SPECIAL, PUNITIVE, EXEMPLARY, INCIDENTAL OR CONSEQUENTIAL LOSS,
* DAMAGE, FOR COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE CODE,
* HOWSOEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR
* THE DAMAGES ARE FORESEEABLE.  TO THE FULLEST EXTENT ALLOWABLE BY LAW,
* MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED* TO THIS CODE,
* SHALL NOT EXCEED THE PRICE YOU PAID DIRECTLY TO MICROCHIP SPECIFICALLY TO
* HAVE THIS CODE DEVELOPED.
*
* You agree that you are solely responsible for testing the code and
* determining its suitability.  Microchip has no obligation to modify, test,
* certify, or support the code.
*********************************************************************************/


/*
--------------------------------------------------------
    Filename:    LCD.c
    Date:                March 2-2010
    File Version:   1.0
    Written by:     John Mouton
    Company:       Microchip Technology
*/


#include <pic.h>

#include "lcd.h"

//Function Prototype
void lcd_update(unsigned char Dig1, unsigned char Dig2 , unsigned char Dig3, unsigned char Dig4,
unsigned char Dig5);

/*
 * Update the LCD with the supplied numerical values
 *
 * Note that we use two back to back if statements when comparing bits rather
 * than if-else as it is smaller
 */
```

**EXAMPLE : COMPLETE SOURCE CODE (CONTINUED)**

```
/* Segments in a 7 segment display */
#define SEG_A  0x01
#define SEG_B  0x02
#define SEG_C  0x04
#define SEG_D  0x08
#define SEG_E  0x10
#define SEG_F  0x20
#define SEG_G  0x40

static bank1 unsigned char lcd_tmp;

unsigned char seg7_cvt(unsigned char digit);

void lcd_init(void)
{
    LCDSE0 = 0xFE;                           // enable first group of LCD segment outputs
    LCDSE1 = 0x8F;                           // enable second group of LCD segments
    LCDSE2 = 0xFF;                           // enable third group of LCD segments
    LCDSE3 = 0x00;
    LCDSE4 = 0x00;
    LCDSE5 = 0x00;
    LCDDATA0  = 0;                           // clear LCD segment registers
    LCDDATA1  = 0;
    LCDDATA2  = 0;
    LCDDATA3  = 0;
    LCDDATA4  = 0;
    LCDDATA5  = 0;
    LCDDATA6  = 0;
    LCDDATA7  = 0;
    LCDDATA8  = 0;
    LCDDATA9  = 0;
    LCDDATA10 = 0;
    LCDDATA11 = 0;
    LCDDATA12 = 0;
    LCDDATA13 = 0;
    LCDDATA14 = 0;
    LCDDATA15 = 0;
    LCDDATA16 = 0;
    LCDDATA17 = 0;
    LCDDATA18 = 0;
    LCDDATA19 = 0;
    LCDDATA20 = 0;
    LCDDATA21 = 0;
    LCDDATA22 = 0;
    LCDDATA23 = 0;

                           // WAVEFORM TYPE A, LCD MODULE IS ACTIVE
   LCDPS  = 0x20;          // PRESCALER IS 1:1,  BIAS IS 0 (CAN BE STATIC OR 1/3)
                           // LCD MODULE IS ON, DRIVER MODULE IS ENABLED DURING SLEEP
   LCDCON = 0x8B;          // NO WRITE FAIL ERROR, VLDC PINS ARE ENABLED, MULTIPLEX 1/4
                                                                  BIAS 1/3
   LCDREF = 0x80;
   LCDCST = 0x00;
   LCDRL  = 0xF0;
}
```

**EXAMPLE : COMPLETE SOURCE CODE (CONTINUED)**

```
void lcd_update(unsigned char Dig1, unsigned char Dig2, unsigned char Dig3, unsigned char Dig4,
unsigned char Dig5)
{
/*
 *  Microchip Logo
 */
SEG18COM0 = 1;          /* Turn All off to start */


/*
* Update digit 1 - Labeled 1A-1G on the display
*/
           lcd_tmp = seg7_cvt(Dig1);

           /* Segment A */
           if ( lcd_tmp & SEG_A )
                SEG1COM0 = 1;
        else ( !(lcd_tmp & SEG_A) )
                SEG1COM0 = 0;

        /* Segment B */
        if ( lcd_tmp & SEG_B )
                SEG3COM0 = 1;
        else ( !(lcd_tmp & SEG_B) )
                SEG3COM0 = 0;

        /* Segment C */
        if ( lcd_tmp & SEG_C )
                SEG3COM1 = 1;
        else ( !(lcd_tmp & SEG_C) )
                SEG3COM1 = 0;

        /* Segment D */
        if ( lcd_tmp & SEG_D )
                SEG3COM2 = 1;
        else ( !(lcd_tmp & SEG_D) )
                SEG3COM2 = 0;

        /* Segment E */
        if ( lcd_tmp & SEG_E )
                SEG1COM1 = 1;
        else ( !(lcd_tmp & SEG_E) )
                SEG1COM1 = 0;

        /* Segment F */
        if ( lcd_tmp & SEG_F )
                SEG1COM3 = 1;
        else ( !(lcd_tmp & SEG_F) )
                SEG1COM3 = 0;

        /* Segment G */
        if ( lcd_tmp & SEG_G )
                SEG3COM3 = 1;
        else ( !(lcd_tmp & SEG_G) )
                SEG3COM3 = 0;
```

**EXAMPLE : COMPLETE SOURCE CODE (CONTINUED)**

```
        /*
         * Update digit 2 - Labled 2A-2G on the display
         */
        lcd_tmp = seg7_cvt(Dig2);

        /* Segment A */
        if ( lcd_tmp & SEG_A )
                SEG6COM0 = 1;
        else ( !(lcd_tmp & SEG_A) )
                SEG6COM0 = 0;

        /* Segment B */
        if ( lcd_tmp & SEG_B )
                SEG16COM0 = 1;
        else ( !(lcd_tmp & SEG_B) )
                SEG16COM0 = 0;

        /* Segment C */
        if ( lcd_tmp & SEG_C )
                SEG16COM1 = 1;
        else ( !(lcd_tmp & SEG_C) )
                SEG16COM1 = 0;

        /* Segment D */
        if ( lcd_tmp & SEG_D )
                SEG6COM2 = 1;
        else ( !(lcd_tmp & SEG_D) )
                SEG6COM2 = 0;

        /* Segment E */
        if ( lcd_tmp & SEG_E )
                SEG6COM1 = 1;
        else ( !(lcd_tmp & SEG_E) )
                SEG6COM1 = 0;

        /* Segment F */
        if ( lcd_tmp & SEG_F )
                SEG6COM3 = 1;
        else ( !(lcd_tmp & SEG_F) )
                SEG6COM3 = 0;

        /* Segment G */
        if ( lcd_tmp & SEG_G )
                SEG16COM3 = 1;
        else ( !(lcd_tmp & SEG_G) )
                SEG16COM3 = 0;

        /*
         * Update digit 3 - Labled 3A-3G on the display
         */
        lcd_tmp = seg7_cvt(Dig3);
                SEG22COM3 = 0;

        /* Segment G */
        if ( lcd_tmp & SEG_G )
                SEG21COM3 = 1;
        else ( !(lcd_tmp & SEG_G) )
                SEG21COM3 = 0;
```

**EXAMPLE : COMPLETE SOURCE CODE (CONTINUED)**

```
        /* Segment A */
        if ( lcd_tmp & SEG_A )
                SEG11COM0 = 1;
        else ( !(lcd_tmp & SEG_A) )
                SEG11COM0 = 0;

        /* Segment B */
        if ( lcd_tmp & SEG_B )
                SEG23COM0 = 1;
        else ( !(lcd_tmp & SEG_B) )
                SEG23COM0 = 0;

        /* Segment C */
        if ( lcd_tmp & SEG_C )
                SEG23COM1 = 1;
        else ( !(lcd_tmp & SEG_C) )
                SEG23COM1 = 0;

        /* Segment D */
        if ( lcd_tmp & SEG_D )
                SEG11COM2 = 1;
        else ( !(lcd_tmp & SEG_D) )
                SEG11COM2 = 0;

        /* Segment E */
        if ( lcd_tmp & SEG_E )
                SEG11COM1 = 1;
        else ( !(lcd_tmp & SEG_E) )
                SEG11COM1 = 0;

        /* Segment F */
        if ( lcd_tmp & SEG_F )
                SEG11COM3 = 1;
        else ( !(lcd_tmp & SEG_F) )
                SEG11COM3 = 0;

        /* Segment G */
        if ( lcd_tmp & SEG_G )
                SEG23COM3 = 1;
        else ( !(lcd_tmp & SEG_G) )
                SEG23COM3 = 0;


        /*
         * Update digit 4 - Labeled 4A-4G on the display
         */
        lcd_tmp = seg7_cvt(Dig4);

        /* Segment A */
        if ( lcd_tmp & SEG_A )
                SEG22COM0 = 1;
        else ( !(lcd_tmp & SEG_A) )
                SEG22COM0 = 0;

        /* Segment B */
        if ( lcd_tmp & SEG_B )
                SEG21COM0 = 1;
        else ( !(lcd_tmp & SEG_B) )
                SEG21COM0 = 0;
```

**EXAMPLE : COMPLETE SOURCE CODE (CONTINUED)**

```
        /* Segment C */
        if ( lcd_tmp & SEG_C )
                SEG21COM1 = 1;
        else ( !(lcd_tmp & SEG_C) )
                SEG21COM1 = 0;

        /* Segment D */
        if ( lcd_tmp & SEG_D )
                SEG22COM2 = 1;
        else ( !(lcd_tmp & SEG_D) )
                SEG22COM2 = 0;

        /* Segment E */
        if ( lcd_tmp & SEG_E )
                SEG22COM1 = 1;
        else ( !(lcd_tmp & SEG_E) )
                SEG22COM1 = 0;

        /* Segment F */
        if ( lcd_tmp & SEG_F )
                SEG22COM3 = 1;
        else ( !(lcd_tmp & SEG_F) )

        /*
         * Update digit 5 - Labeled 5A-5G on the display
         */

        lcd_tmp = seg7_cvt(Dig5);

        /* Segment A */
        if ( lcd_tmp & SEG_A )
                SEG5COM0 = 1;
        else ( !(lcd_tmp & SEG_A) )
                SEG5COM0 = 0;

        /* Segment B */
        if ( lcd_tmp & SEG_B )
                SEG4COM0 = 1;
        else ( !(lcd_tmp & SEG_B) )
                SEG4COM0 = 0;

        /* Segment C */
        if ( lcd_tmp & SEG_C )
                SEG4COM1 = 1;
        else ( !(lcd_tmp & SEG_C) )
                SEG4COM1 = 0;

        /* Segment D */
        if ( lcd_tmp & SEG_D )
                SEG5COM2 = 1;
        else ( !(lcd_tmp & SEG_D) )
                SEG5COM2 = 0;

        /* Segment E */
        if ( lcd_tmp & SEG_E )
                SEG5COM1 = 1;
        else ( !(lcd_tmp & SEG_E) )
                SEG5COM1 = 0;
```

**EXAMPLE : COMPLETE SOURCE CODE (CONTINUED)**

```
        /* Segment F */
        if ( lcd_tmp & SEG_F )
                SEG5COM3 = 1;
        else ( !(lcd_tmp & SEG_F) )
                SEG5COM3 = 0;

        /* Segment G */
        if ( lcd_tmp & SEG_G )
                SEG4COM3 = 1;
        else ( !(lcd_tmp & SEG_G) )
                SEG4COM3 = 0;

}

/* Convert the integer value to which segments need to be turned on or off */
unsigned char seg7_cvt(unsigned char digit)
{
    switch (digit)
    {
        case '0':
        case 0:
           return SEG_A | SEG_B | SEG_C | SEG_D | SEG_E | SEG_F;
        case '1':
        case 1:
           return SEG_B | SEG_C;
        case '2':
        case 2:
           return SEG_A | SEG_B | SEG_D | SEG_E | SEG_G;
        case '3':
        case 3:
           return SEG_A | SEG_B | SEG_C | SEG_D | SEG_G;
        case '4':
        case 4:
           return SEG_B | SEG_C | SEG_F | SEG_G;
        case '5':
        case 5:
           return SEG_A | SEG_C | SEG_D | SEG_F | SEG_G;
        case '6':
        case 6:
           return SEG_A | SEG_C | SEG_D | SEG_E | SEG_F | SEG_G;
        case '7':
        case 7:
           return SEG_A | SEG_B | SEG_C;
        case '8':
        case 8:
           return SEG_A | SEG_B | SEG_C | SEG_D | SEG_E | SEG_F | SEG_G;
        case '9':
        case 9:
           return SEG_A | SEG_B | SEG_C | SEG_D | SEG_F | SEG_G;
        case 'L':
               return SEG_F | SEG_E | SEG_D;
        case 'O':
               return SEG_A | SEG_B | SEG_C | SEG_D | SEG_E | SEG_F;
        case 'C':
               return SEG_A | SEG_F | SEG_E | SEG_D;
        case 'U':
               return SEG_F | SEG_E | SEG_D | SEG_C | SEG_B;
```

**EXAMPLE : COMPLETE SOURCE CODE (CONTINUED)**

```
        case 'n':
               return SEG_E | SEG_G | SEG_C;
        case ' ':
               return 0;
        case '-':
               return SEG_G;

        default:       /* Display a visible pattern when we have something we don't understand */
               return 0;
        }
}


/*
--------------------------------------------------------
    Filename:    lcd.h
    Date:                March 2-2010
    File Version:   1.0
    Written by:     John Mouton
    Company:       Microchip Technology
*/

#ifndef _LCD_H
#define _LCD_H


/* Update the LCD with the supplied data */
void lcd_update(unsigned char Dig1, unsigned char Dig2, unsigned char Dig3, unsigned char Dig4,
unsigned char Dig5);

/* Initialize the LCD for use */
void lcd_init(void);

#endif /* _LCD_H */
```

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

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

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



**Trademarks**



Printed on recycled paper.

ISBN: 978-1-60932-534-3

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

*Microchip received ISO/TS-16949:2002 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.*

# 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

**Cleveland**
Independence, OH
Tel: 216-447-0464
Fax: 216-447-0643

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

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

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

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

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

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

## ASIA/PACIFIC

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

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

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

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

**China - Chongqing**
Tel: 86-23-8980-9588
Fax: 86-23-8980-9500

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

**China - Nanjing**
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

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

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

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

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

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

**China - Xian**
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

**China - Xiamen**
Tel: 86-592-2388138
Fax: 86-592-2388130

**China - Zhuhai**
Tel: 86-756-3210040
Fax: 86-756-3210049

## ASIA/PACIFIC

**India - Bangalore**
Tel: 91-80-3090-4444
Fax: 91-80-3090-4123

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

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

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

**Korea - Daegu**
Tel: 82-53-744-4301
Fax: 82-53-744-4302

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

**Malaysia - Kuala Lumpur**
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

**Malaysia - Penang**
Tel: 60-4-227-8870
Fax: 60-4-227-4068

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

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

**Taiwan - Hsin Chu**
Tel: 886-3-6578-300
Fax: 886-3-6578-370

**Taiwan - Kaohsiung**
Tel: 886-7-213-7830
Fax: 886-7-330-9305

**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

08/04/10